MPLAB® Harmony Help - Volume IV - MPLAB Harmony Framework Reference

MPLAB Harmony Integrated Software Framework v1.11

© 2013-2017 Microchip Technology Inc. All rights reserved.

Volume IV: MPLAB Harmony Framework Reference

This volume provides API reference information for the framework libraries included in your installation of MPLAB Harmony.

Framework Overview

This section provides an overview of the MPLAB Harmony Framework.

Introduction

This section introduces the overview to the API reference for the MPLAB Harmony Framework.

Description

The MPLAB Harmony development platform and ecosystem is based on a layered framework of interoperable library modules. Library modules, except for the stateless peripheral libraries (PLIBs), are active and normally implement one or more independent, but cooperative state machines. These state machines service requests made by the application or other “client” modules through a set of interface (or API) functions. Different library modules serve different purposes: from core system services, to drivers for peripheral devices, to middleware layers supporting communication protocols and software capabilities.

This volume is a programmer reference that details the interfaces to the libraries that make up MPLAB Harmony and explains how to use the libraries individually to accomplish the tasks for which they were designed.

For more introductory information that (1) explains how MPLAB Harmony generally works and (2) describes how to develop MPLAB Harmony libraries and applications, refer to the following volumes.

Volume I: Getting Started With MPLAB Harmony

This volume introduces MPLAB Harmony in detail. The section What is MPLAB Harmony? is particularly designed for those who are completely new to MPLAB Harmony.

Volume III: MPLAB Harmony Development

This volume provides information on how to develop MPLAB Harmony compatible libraries and applications. The following sections in particular will help you to gain a better understanding of how MPLAB Harmony library modules work.

Key Concepts

This section describes a number of concepts that are key to understanding how the MPLAB Harmony framework operates, including fundamental concepts of modularity, relationships between library clients, interfaces, instances and implementations, as well as information about MPLAB Harmony’s state-machine programming model and blocking guidelines.

MPLAB Harmony Compatibility Guide

This section lists and describes the requirements to develop a module that is compatible with MPLAB Harmony.

MPLAB Harmony Driver Development Guide

This section describes how to develop device drivers for MPLAB Harmony.

Bluetooth Stack Library Help

This section provides information on the PIC32 Bluetooth Stack Library that is available in MPLAB Harmony.

Description

The PIC32 Bluetooth Stack Library consists of two distinct libraries:

- PIC32 Bluetooth Basic Stack Library - this library provides "Core" Bluetooth functionality, which was previously referred to as "SPP-only"
- PIC32 Bluetooth Audio Stack Library - this library, which must be purchased, provides audio features

The PIC32 Bluetooth Stack Library is provided in binary form with associated APIs to enable connection, control and data to and from Bluetooth-enabled devices with PIC32 microcontrollers.

Introduction

Describes the PIC32 Bluetooth Stack Library.

Description

Important! The PIC32 Bluetooth Stack Library is created under license from Searan LLC, and uses the dotstack™ framework. This library has been qualified for use and a QDID is available from the Bluetooth SIG. More information about the qualification process can be found in the *"PIC32 Bluetooth Audio Development Kit Reference Guide"* (DS70005140), which is available from the Microchip website: www.microchip.com.

The PIC32 Bluetooth Stack Library is available only in binary format. When a license is acquired it may be for SPP (data) only, or with access to SBC or AAC decoders. The different licenses come with a number of specific applications. For more information on the license options please see http://www.microchip.com/pic32btsuites.

PIC32 Bluetooth Stack Library

This library provides all of the functionality of the Bluetooth Stack interfaces including profiles and protocols at the lower levels. These are utilized in a number of demonstration applications used in the PIC32 Bluetooth Audio Development Kit. Specific demonstrations to illustrate the various connection options, data and audio communications and control interfaces are available for purchase. Please refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for more information. This library interface is provided in the C language.

This library provides all of the functionality of the Bluetooth Stack interfaces including profiles and protocols at the lower levels. These are utilized in a number of demonstration applications used in the PIC32 Bluetooth Audio Development Kit, PIC32 Bluetooth Starter Kit and PIC32MZ EC Starter Kit development board attached to Multimedia Expansion Board II (MEB II).

PIC32 Bluetooth Audio Stack Library

Note: The PIC32 Bluetooth Audio Stack Library is a Premium product, and is not included in the standard release of MPLAB Harmony and must be purchased. For information on purchasing the Premium PIC32 Bluetooth Audio Stack Library, please refer to the MPLAB Harmony Premium Products page by visiting:

http://www.microchip.com/pagehandler/en-us/devtools/mplabharmony/home.html

This library provides full functionality of the PIC32 Bluetooth Stack Library that includes Serial Port Profile (SPP) and Audio protocols and profiles such as AVDTP, AVCTP, A2DP, and AVRCP. The library interface is provided in the C language.

The PIC32 Bluetooth Audio Stack Library is a premium product and is not included in the standard release of MPLAB Harmony and must be purchased separately. For information on purchasing the premium PIC32 Bluetooth Audio Stack Library, please refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio).

This library provides all of the functionality of the Bluetooth Stack interfaces including profiles and protocols at the lower levels. These are utilized in a number of demonstration applications used in the PIC32 Bluetooth Audio Development Kit, PIC32 Bluetooth Starter Kit and PIC32MZ EC Starter Kit development board attached to Multimedia Expansion Board II (MEB II).

Demonstrations

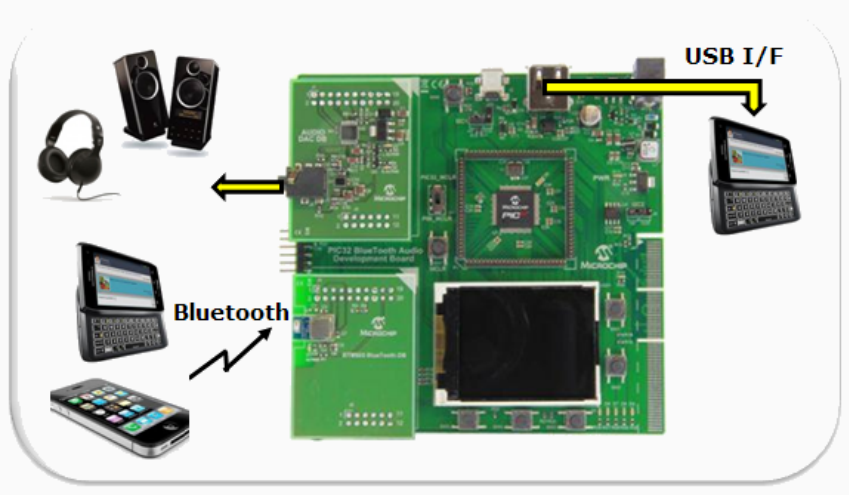
Refer to PIC32 Bluetooth Stack Library Demonstrations for information on the demonstrations that are available for the PIC32 Bluetooth Stack Library. In addition, other specific demonstrations to illustrate the various connection options, data and audio communications and control interfaces are available for purchase. Please refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for more information.

Bluetooth Stack Licensing

This section describes how to license the stack.

Description

The license to the stack must be procured from Microchip, and it is specific to the Microchip PIC32 microcontroller architecture. Some licenses require a one-time fee conveyed when a customer purchases the demonstration applications for the PIC32 Bluetooth Audio Development Kit (P/N DV320032).



There are four potential options in the license, each with the benefits of the previous.

- *SPP (data only)* - This is a special version of the library available to carry data only. The decoder options have been removed in this case. The license is available for Microchip PIC products, but is free of charge. Special demonstrations and the library for this product can be found under the SPP section by visiting http://www.microchip.com/pic32btsuites.
- *SBC Decoder* - This Bluetooth license also grants the user the ability to use the baseline audio SBC decoder, which is provided as a separate library. The demonstrations include stand-alone Bluetooth audio streaming using this decoder, and other applications which combine the use of USB and Bluetooth audio.
- *AAC Decoder* - This Bluetooth license also grants the user the ability to use the audio AAC decoder if a connected devices supports this form of streaming. For devices that only support SBC, the SBC decoder is also included. All demonstrations of the (1) license are included. In addition, new demonstrations are included that make specific use of the AAC decoder both with and without USB options.
- *Break-in Feature with SBC Decoder* - The break-in feature enables multi-connection support and a "Party mode" where multiple handset devices can take turns sharing the Bluetooth connection to stream SBC audio. This also supports the single connection SBC audio streaming in SPP.

Note: All licenses are available through Microchip direct. More details on the specific application demonstrations can be found in the *"PIC32 Bluetooth Audio Development Kit Reference Guide"* (DS70005140). Visit http://www.microchip.com/pic32btsuites for the available demonstrations and licensing options.

Using the Library

This topic describes the basic architecture of the PIC32 Bluetooth Stack Library and provides information and examples on its use.

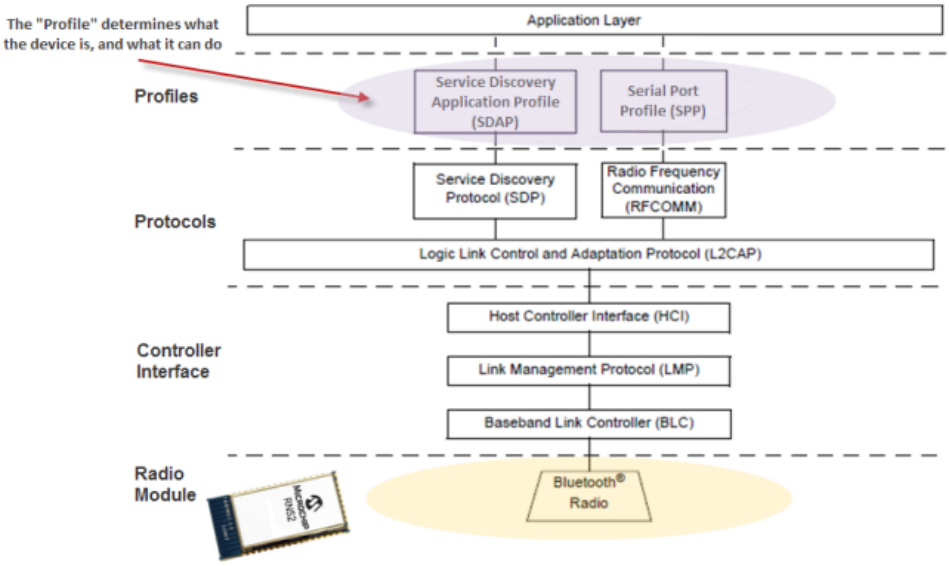
Description

The interface to the Bluetooth stack is driven by a number of header files. The stack header files form a collection of APIs for each protocol and profile of the Bluetooth stack. These files can be found within the following MPLAB Harmony directory, which is included when you license the stack and the example applications that are provided:

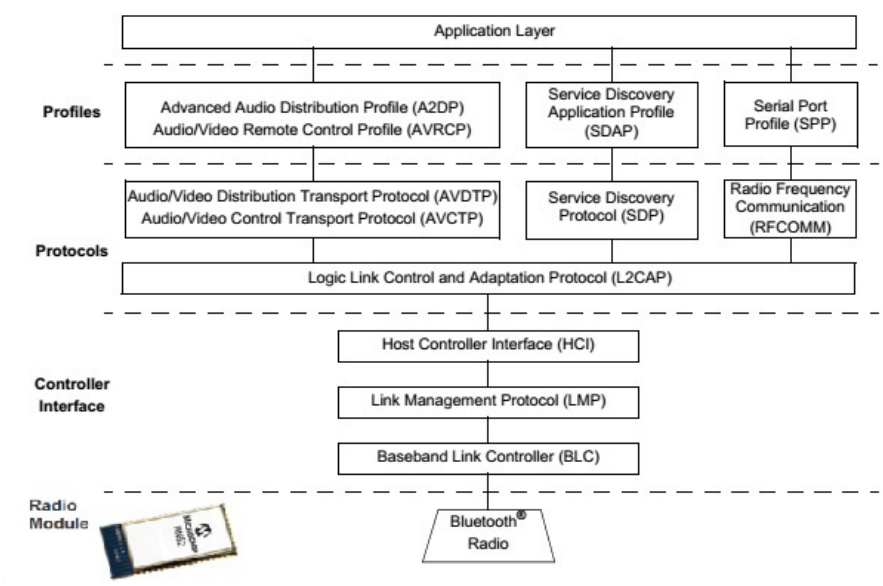
- `<install_dir>/framework/bluetooth/cdbt` for the PIC32 Bluetooth Basic Stack
- `<install_dir>/framework/bluetooth/premium/audio/cdbt` for the PIC32 Bluetooth Audio Stack

The following two figures illustrate the Bluetooth Stacks.

PIC32 Bluetooth Basic Stack



PIC32 Bluetooth Audio Stack



Library Overview

Provides information on the APIs provided in the library.

Description

The library interface routines are divided into various sub-sections, which address one of the profiles or the overall operation of the PIC32 Bluetooth Stack Library.

Library Interface Section	Description
Bluetooth Support	Contains functions for basic Bluetooth support.
A2DP	Contains functions for the Advanced Audio Distribution Profile (see **Note**).
AVCTP	Contains functions for the Audio/Video Control Transport Protocol (see **Note**).
AVDTP	Contains functions for the Audio/Video Distribution Transport Protocol (see **Note**).
AVRCP	Contains functions for the Audio/Video Remote Control Profile (see **Note**).
GAP	Contains functions for the Generic Access Profile.
HCI	Contains functions for the Host Controller Interface.

L2CAP	Contains functions for the Logical Link Control and Adaptation Protocol.
RFCOMM	Contains functions for the RF Communication Lower Layer.
SBC Decoder	Contains functions for the SBC Decoder
SDP	Contains functions for the Service Discovery Protocol.
SPP	Contains functions for the Serial Port Profile.
SSP	Contains functions for Secure Simple Pairing.
Misc.	Contains miscellaneous functions not noted elsewhere in the stack.
Utility	Contains functions that support the stack functions.

Note: The PIC32 Bluetooth Audio Stack is an extension of the PIC32 Bluetooth Basic Stack and provides the additional features to support audio. The following sections cover the audio features of the PIC32 Bluetooth Stack.

- AVDTP Functions
- AVDTP Data Types and Constants
- AVCTP Functions
- AVCTP Data Types and Constants
- A2DP Functions
- A2DP Data Types and Constants
- AVRCP Functions
- AVRCP Data Types and Constants

Configuring the Library

This topic provides information on configuring the library.

AVCTP Configuration Macros

	Name	Description
	__AVCTP_CONFIG_H	This is macro __AVCTP_CONFIG_H.

	AVCTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	• defgroup avctp_config Configuration
		• ingroup avctp
		*
		• This module describes parameters used to configure AVCTP layer.
		*
		• dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.
		* code #include "cdbt/bt/bt_std.h"
		// HCI, L2CAP and SDP must always be present
		// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...
		// L2CAP configuration parameters #define L2CAP_MAX_CMD_BUFFERS ... #define L2CAP_MAX_FRAME_BUFFERS ... #define L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...
		// SDP... more

	AVCTP_ALLOCATE_BUFFERS_VARS	brief Maximum number of message buffers ingroup avctp_config details This parameter defines the maximum number of buffer available for sending message.
	AVCTP_MAX_CHANNELS	brief Maximum number of AVCTP channels ingroup avctp_config details This parameter defines the maximum number of channels a local device can have with remote devices.
	AVCTP_MAX_TRANSPORT_CHANNELS	brief Maximum number of AVCTP transports ingroup avctp_config details This parameter defines the maximum number of transports a local device can have with remote devices. This value should not exceed AVCTP_MAX_CHANNELS.

AVDTP Configuration Macros

	Name	Description
	AVDTP_ALLOCATE_BUFFERS_FUNCTION	This is macro AVDTP_ALLOCATE_BUFFERS_FUNCTION.

	AVDTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	• defgroup avdtp_config Configuration
		• ingroup avdtp
		*
		• This module describes parameters used to configure AVDTP layer.
		*
		• dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.
		* code #include "cdbt/bt/bt_std.h"
		// HCI, L2CAP and SDP must always be present
		// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...
		// L2CAP configuration parameters #define L2CAP_MAX_CMD_BUFFERS ... #define L2CAP_MAX_FRAME_BUFFERS ... #define L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...
		// SDP... more

	AVDTP_ALLOCATE_BUFFERS_VARS	This is macro AVDTP_ALLOCATE_BUFFERS_VARS.
	AVDTP_CMD_ABORT	This is macro AVDTP_CMD_ABORT.
	AVDTP_CMD_CLOSE	This is macro AVDTP_CMD_CLOSE.
	AVDTP_CMD_DISCOVER	This is macro AVDTP_CMD_DISCOVER.
	AVDTP_CMD_GET_CAPABILITIES	This is macro AVDTP_CMD_GET_CAPABILITIES.
	AVDTP_CMD_GET_CONFIGURATION	This is macro AVDTP_CMD_GET_CONFIGURATION.
	AVDTP_CMD_OPEN	This is macro AVDTP_CMD_OPEN.
	AVDTP_CMD_RECONFIGURE	This is macro AVDTP_CMD_RECONFIGURE.
	AVDTP_CMD_SECURITY_CONTROL	This is macro AVDTP_CMD_SECURITY_CONTROL.
	AVDTP_CMD_SET_CONFIGURATION	This is macro AVDTP_CMD_SET_CONFIGURATION.
	AVDTP_CMD_START	This is macro AVDTP_CMD_START.
	AVDTP_CMD_SUSPEND	This is macro AVDTP_CMD_SUSPEND.
	AVDTP_CODEC_CONFIG_BUFFER_LEN	brief Size of the buffer used to store codec specific configuration. ingroup avdtp_config details Each codec uses unique configuration which can take different amount of memory. This parameter defines the size of the buffer for storing codec's configuration. The value of 16 is sufficient for SBC, AAC and MPEG1,2. If vendor specific codec is to be used this value may need to increased.
	AVDTP_MAX_CMD_BUFFERS	brief Maximum number of command buffers. ingroup avdtp_config details This parameter defines the number of buffers reserved for sending commands to a remote device over its control channel. Each channel uses its own buffers so the total number of buffers is AVDTP_MAX_REMOTE_DEVICES * AVDTP_MAX_CMD_BUFFERS. The minimum value is 1. The maximum value is 255. 2 is usually sufficient.
	AVDTP_MAX_CMD_PARAM_LEN	brief Maximum length of control command parameters ingroup avdtp_config details This parameter defines the maximum length of all command parameters. The value should not exceed AVDTP_MAX_TX_BUFFER_LEN - 2 (command header).
	AVDTP_MAX_REMOTE_DEVICES	brief Maximum number of remote devices a local device can be connected to ingroup avdtp_config details This parameter defines the number of remote devices a local device can have simultaneous connections to (i.e. control channels). This value should not exceed AVDTP_MAX_STREAMS. For each remote device AVDTP creates one control channel regardless of the number of streams between the local and the remote devices. Assuming that the local devices wants to have only one channel with each remote device and if AVDTP_MAX_REMOTE_DEVICES > AVDTP_MAX_STREAMS all memory reserved for devices in excess of AVDTP_MAX_STREAMS will be wasted. The minimum value is... more

	AVDTP_MAX_SEP	brief Maximum number of SEPs that can be exposed by a local device ingroup avdtp_config details This parameter defines the number of SEPs an application can expose to remote devices. The minimum value is 1. The maximum value is 255.
	AVDTP_MAX_STREAMS	brief Maximum number of streams that can be exposed by a local device ingroup avdtp_config details This parameter defines the number of streams an application can open between local and remote devices. This value can be different from AVDTP_MAX_SEP but should not exceed it. Since each SEP can only be used once the local device can only have as much streams as there are SEPs. If AVDTP_MAX_STREAMS > AVDTP_MAX_SEP all memory reserved for streams in excess of AVDTP_MAX_SEP will be wasted. The minimum value is 1. The maximum value is 255.
	AVDTP_MAX_TRANSPORT_CHANNELS	brief Maximum number of transport channels. ingroup avdtp_config details Depending on the SEP capabilities (multiplexing, recovery, reporting) each stream may need up to 3 transport channels. E.g., if multiplexing is not supported and recovery and reporting are supported a stream will use 3 transport channels - 1 for media transport session, 1 for recovery transport session and 1 for reporting transport session. If multiplexing is not supported and only reporting is supported the stream will use 2 transport channels - 1 for media transport session and 1 for reporting transport session. If multiplexing is supported the stream will need only... more
	AVDTP_MAX_TX_BUFFER_LEN	brief Size of the transmit buffer. ingroup avdtp_config details This parameter defines the size of the buffer used to send AVDTP control commands to L2CAP layer. Each control channel has it own buffer so the total amount of memory allocated for these buffers is AVDTP_MAX_TX_BUFFER_LEN) * AVDTP_MAX_REMOTE_DEVICES. The minimum value is 1. The maximum value is 255. The value of 32 is usually sufficient.

AVRCP Configuration Macros

	Name	Description
	__AVRCP_CONFIG_H	This is macro __AVRCP_CONFIG_H.

	AVRCP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	• defgroup avrcp_config Configuration
		• ingroup avrcp
		*
		• This module describes parameters used to configure AVRCP layer.
		*
		• dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.
		* code #include "cdbt/bt/bt_std.h"
		// HCI, L2CAP and SDP must always be present
		// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...
		// L2CAP configuration parameters #define L2CAP_MAX_CMD_BUFFERS ... #define L2CAP_MAX_FRAME_BUFFERS ... #define L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...
		// SDP... more

	AVRCP_ALLOCATE_BUFFERS_VARS	This is macro AVRCP_ALLOCATE_BUFFERS_VARS.
	AVRCP_CMD_TIMEOUT	brief Command timeout ingroup avrcp_config details This parameter defines the amount of time in milliseconds AVRCP waits for a response to a request. If not defined the default value of 10000 (10 secconds) is used.
	AVRCP_MAX_CHANNELS	brief Maximum number of remote devices a local device can be connected to ingroup avrcp_config details This parameter defines the number of remote devices a local device can have simultaneous connections to (i.e. channels). This value should not exceed AVCTP_MAX_CHANNELS.

	AVRCP_MAX_CMD_BUFFERS	brief Maximum number of command buffers. ingroup avrcp_config details This parameter defines the number of buffers reserved for sending commands to a remote device over its control channel. Each channel uses its own buffers so the total number of buffers is AVRCP_MAX_CHANNELS * AVRCP_MAX_CMD_BUFFERS. The minimum value is 1. The maximum value is 255. If not define one buffer for each channel is reserved.
	AVRCP_MAX_CMD_PARAM_LEN	brief Maximum length of command parameters ingroup avrcp_config details This parameter defines the maximum length of all command parameters. If not defined the default value of 512 is used.
	AVRCP_MAX_DEVICE_NAME_LEN	brief Maximum length of device name ingroup avrcp_config details This parameter defines the size of the buffer used to store device's name while searching for nearby targets with bt_avrcp_find_targets. If the name of the device is longer than AVRCP_MAX_DEVICE_NAME_LEN it is truncated to AVRCP_MAX_DEVICE_NAME_LEN. If not defined the default value of 20 is used.
	AVRCP_MAX_SEARCH_RESULTS	brief Maximum number of devices to find ingroup avrcp_config details This parameter defines the maximum number of devices bt_avrcp_find_targets can find. If not defined the default value of 7 is used.

Bluetooth Support Configuration Macros

	Name	Description
	ATT_ALLOCATE_BUFFERS	This is function ATT_ALLOCATE_BUFFERS.
	ATT_CLIENT_ALLOCATE_BUFFERS	This is function ATT_CLIENT_ALLOCATE_BUFFERS.
	FTP_ALLOCATE_BUFFERS	This is function FTP_ALLOCATE_BUFFERS.
	GATT_CLIENT_ALLOCATE_BUFFERS	This is function GATT_CLIENT_ALLOCATE_BUFFERS.
	HCI_ALLOCATE_BUFFERS	This is function HCI_ALLOCATE_BUFFERS.
	HFP_ALLOCATE_BUFFERS	This is function HFP_ALLOCATE_BUFFERS.
	HID_ALLOCATE_BUFFERS	This is function HID_ALLOCATE_BUFFERS.
	HSP_AG_ALLOCATE_BUFFERS	This is function HSP_AG_ALLOCATE_BUFFERS.
	HSP_ALLOCATE_BUFFERS	This is function HSP_ALLOCATE_BUFFERS.
	IAP_ALLOCATE_BUFFERS	This is function IAP_ALLOCATE_BUFFERS.
	IAP_BT_ALLOCATE_BUFFERS	This is function IAP_BT_ALLOCATE_BUFFERS.
	IAP2_ALLOCATE_BUFFERS	This is function IAP2_ALLOCATE_BUFFERS.
	L2CAP_ALLOCATE_BUFFERS	This is function L2CAP_ALLOCATE_BUFFERS.
	OBEX_ALLOCATE_BUFFERS	This is function OBEX_ALLOCATE_BUFFERS.
	PBAP_ALLOCATE_BUFFERS	This is function PBAP_ALLOCATE_BUFFERS.
	RFCOMM_ALLOCATE_BUFFERS	This is function RFCOMM_ALLOCATE_BUFFERS.
	SDP_ALLOCATE_BUFFERS	This is function SDP_ALLOCATE_BUFFERS.
	SMP_ALLOCATE_BUFFERS	This is function SMP_ALLOCATE_BUFFERS.
	SPP_ALLOCATE_BUFFERS	This is function SPP_ALLOCATE_BUFFERS.
	BT_ENABLE_BLE	This is macro BT_ENABLE_BLE.
	BT_INCLUDE_RFCOMM	This is macro BT_INCLUDE_RFCOMM.
	BT_LOG_LEVEL_MAX	This is macro BT_LOG_LEVEL_MAX.
	BT_LOG_LEVEL_MIN	This is macro BT_LOG_LEVEL_MIN.
	__BT_APP_CONFIG_H	This is macro __BT_APP_CONFIG_H.
	BT_INCLUDE_IAP	This is macro BT_INCLUDE_IAP.
	BT_INCLUDE_IAP2	This is macro BT_INCLUDE_IAP2.
	IAP_BT_MAX_TRANSPORTS	This is macro IAP_BT_MAX_TRANSPORTS.
	IAP_MAX_SESSIONS	This is macro IAP_MAX_SESSIONS.
	IAP_RX_BUFFER_SIZE	This is macro IAP_RX_BUFFER_SIZE.
	IAP2_MAX_PACKET_SIZE	This is macro IAP2_MAX_PACKET_SIZE.
	IAP2_MAX_SESSIONS	This is macro IAP2_MAX_SESSIONS.

HCI Configuration Macros

	Name	Description
	BT_ENABLE_SCO	This is macro BT_ENABLE_SCO.
	BT_HCI_EVT_AUTHENTICATION_COMPLETE_HANDLER	This is macro BT_HCI_EVT_AUTHENTICATION_COMPLETE_HANDLER.

	BT_HCI_EVT_CONNECTION_COMPLETE_HANDLER	This is macro BT_HCI_EVT_CONNECTION_COMPLETE_HANDLER.
	BT_HCI_EVT_CONNECTION_REQUEST_HANDLER	This is macro BT_HCI_EVT_CONNECTION_REQUEST_HANDLER.
	BT_HCI_EVT_ENCRYPTION_CHANGE_HANDLER	This is macro BT_HCI_EVT_ENCRYPTION_CHANGE_HANDLER.
	BT_HCI_EVT_EXTENDED_INQUIRY_RESULT_HANDLER	This is macro BT_HCI_EVT_EXTENDED_INQUIRY_RESULT_HANDLER.
	BT_HCI_EVT_INQUIRY_COMPLETE_HANDLER	This is macro BT_HCI_EVT_INQUIRY_COMPLETE_HANDLER.
	BT_HCI_EVT_INQUIRY_RESULT_HANDLER	This is macro BT_HCI_EVT_INQUIRY_RESULT_HANDLER.
	BT_HCI_EVT_INQUIRY_RESULT_WITH_RSSI_HANDLER	This is macro BT_HCI_EVT_INQUIRY_RESULT_WITH_RSSI_HANDLER.
	BT_HCI_EVT_LINK_KEY_NOTIFICATION_HANDLER	This is macro BT_HCI_EVT_LINK_KEY_NOTIFICATION_HANDLER.
	BT_HCI_EVT_LINK_KEY_REQUEST_HANDLER	This is macro BT_HCI_EVT_LINK_KEY_REQUEST_HANDLER.
	BT_HCI_EVT_MODE_CHANGE_HANDLER	This is macro BT_HCI_EVT_MODE_CHANGE_HANDLER.
	BT_HCI_EVT_PIN_CODE_REQUEST_HANDLER	This is macro BT_HCI_EVT_PIN_CODE_REQUEST_HANDLER.
	BT_HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE_HANDLER	This is macro BT_HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE_HANDLER.
	BT_HCI_EVT_ROLE_CHANGE_HANDLER	This is macro BT_HCI_EVT_ROLE_CHANGE_HANDLER.
	BT_HCI_EVT_SYNCH_CONNECTION_COMPLETE_HANDLER	This is macro BT_HCI_EVT_SYNCH_CONNECTION_COMPLETE_HANDLER.
	BT_LE_EVT_HANDLER	This is macro BT_LE_EVT_HANDLER.
	BT_SSP_EVT_HANDLER	This is macro BT_SSP_EVT_HANDLER.
	HCI_ALLOCATE_BUFFERS_FUNCTION	This is macro HCI_ALLOCATE_BUFFERS_FUNCTION.
	HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR	This is macro HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR.
	HCI_ALLOCATE_BUFFERS_VARS	This is macro HCI_ALLOCATE_BUFFERS_VARS.
	HCI_DECLARE_LE_CONN_STATES	This is macro HCI_DECLARE_LE_CONN_STATES.
	HCI_DECLARE_LE_CTRL_STATE	This is macro HCI_DECLARE_LE_CTRL_STATE.
	HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL	This is macro HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL.
	HCI_INIT_LE_CONN_STATES	This is macro HCI_INIT_LE_CONN_STATES.
	HCI_INIT_LE_CTRL_STATE	This is macro HCI_INIT_LE_CTRL_STATE.
	HCI_INIT_SCO_HANDLERS	This is macro HCI_INIT_SCO_HANDLERS.
	HCI_INIT_SSP_HANDLERS	This is macro HCI_INIT_SSP_HANDLERS.
	HCI_L2CAP_BUFFER_LEN	This is macro HCI_L2CAP_BUFFER_LEN.
	HCI_SIZEOF_LE_CONN_STATES	This is macro HCI_SIZEOF_LE_CONN_STATES.
	HCI_SIZEOF_LE_CTRL_STATE	This is macro HCI_SIZEOF_LE_CTRL_STATE.
	HCI_TX_BUFFER_LEN	This is macro HCI_TX_BUFFER_LEN.
	__HCI_CONFIG_EVENT_HANDLERS_H	This is macro __HCI_CONFIG_EVENT_HANDLERS_H.
	__HCI_CONFIG_H	This is macro __HCI_CONFIG_H.
	HCI_MAX_CONNECT_ATTEMPTS	This is macro HCI_MAX_CONNECT_ATTEMPTS.

L2CAP Configuration Macros

	Name	Description
	L2CAP_ALLOCATE_BUFFERS_FUNCTION	This is macro L2CAP_ALLOCATE_BUFFERS_FUNCTION.
	L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	This is macro L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR.
	L2CAP_ALLOCATE_BUFFERS_VARS	This is macro L2CAP_ALLOCATE_BUFFERS_VARS.
	L2CAP_FIXED_CHANNELS_DECL	This is macro L2CAP_FIXED_CHANNELS_DECL.

	L2CAP_HCI_PACKET_TYPE	• brief L2CAP_HCI_PACKET_TYPE. • ingroup btconfig * • details Defines a set of packets that link manager is allowed to use when calling bt_l2cap_connect. • The default value is to enable all packet types. enable all packet types
	L2CAP_HCI_PAGE_SCAN_REPETITION_MODE	brief L2CAP_HCI_PAGE_SCAN_REPETITION_MODE. ingroup btconfig details Defines a default value of the page scan repetition mode when calling bt_l2cap_connect. Must be set to one of the following values: HCI_PAGE_SCAN_REPETITION_MODE_R0 HCI_PAGE_SCAN_REPETITION_MODE_R1 HCI_PAGE_SCAN_REPETITION_MODE_R2 The default value is HCI_PAGE_SCAN_REPETITION_MODE_R0.
	L2CAP_HCI_ROLE_SWITCH	brief L2CAP_HCI_ROLE_SWITCH. ingroup btconfig details Defines a default value of the role switch parameter when calling bt_l2cap_connect. Must be set to one of the following values: HCI_ROLE_SWITCH_ALLOW HCI_ROLE_SWITCH_DISALLOW The default value is to allow the role switch.
	L2CAP_IDLE_CONNECTION_TIMEOUT	seconds
	PACK_CONFIG_REQUEST	This is macro PACK_CONFIG_REQUEST.
	PACK_CONFIG_RESPONSE	This is macro PACK_CONFIG_RESPONSE.
	PACK_CONN_REQUEST	This is macro PACK_CONN_REQUEST.
	PACK_CONN_RESPONSE	This is macro PACK_CONN_RESPONSE.
	PACK_DCONN_REQUEST	This is macro PACK_DCONN_REQUEST.
	PACK_DCONN_RESPONSE	This is macro PACK_DCONN_RESPONSE.
	PACK_INFO_REQUEST	This is macro PACK_INFO_REQUEST.
	PACK_INFO_RESPONSE	This is macro PACK_INFO_RESPONSE.
	PROCESS_CONFIG_REQ	This is macro PROCESS_CONFIG_REQ.
	PROCESS_CONFIG_RES	This is macro PROCESS_CONFIG_RES.
	PROCESS_CONN_REQ	This is macro PROCESS_CONN_REQ.
	PROCESS_CONN_RES	This is macro PROCESS_CONN_RES.
	PROCESS_DCONN_REQ	This is macro PROCESS_DCONN_REQ.
	PROCESS_DCONN_RES	This is macro PROCESS_DCONN_RES.
	PROCESS_INFO_REQ	This is macro PROCESS_INFO_REQ.
	PROCESS_INFO_RES	This is macro PROCESS_INFO_RES.
	READ_CONFIG_REQUEST	This is macro READ_CONFIG_REQUEST.
	READ_CONFIG_RESPONSE	This is macro READ_CONFIG_RESPONSE.
	READ_CONN_REQUEST	This is macro READ_CONN_REQUEST.
	READ_CONN_RESPONSE	This is macro READ_CONN_RESPONSE.
	READ_DCONN_REQUEST	This is macro READ_DCONN_REQUEST.
	READ_DCONN_RESPONSE	This is macro READ_DCONN_RESPONSE.
	READ_INFO_REQUEST	This is macro READ_INFO_REQUEST.
	READ_INFO_RESPONSE	This is macro READ_INFO_RESPONSE.
	__L2CAP_CONFIG_H	This is macro __L2CAP_CONFIG_H.
	__L2CAP_CONFIG_HANDLERS_H	This is macro __L2CAP_CONFIG_HANDLERS_H.
	L2CAP_DECL_ERETR_FUNCTIONS	This is macro L2CAP_DECL_ERETR_FUNCTIONS.
	L2CAP_MAX_FIXED_CHANNELS	This is macro L2CAP_MAX_FIXED_CHANNELS.

RFCOMM Configuration Macros

	Name	Description
	RFCOMM_ALLOCATE_BUFFERS_FUNCTION	This is macro RFCOMM_ALLOCATE_BUFFERS_FUNCTION.
	RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR	This is macro RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR.
	RFCOMM_ALLOCATE_BUFFERS_VARS	This is macro RFCOMM_ALLOCATE_BUFFERS_VARS.
	RFCOMM_BUFFER_SIZE	This is macro RFCOMM_BUFFER_SIZE.

	RFCOMM_ENABLE_MULTIDEVICE_CHANNELS	brief Enable multi-device server channels. ingroup rfcomm_config details Normally each server channel can be used only once. I.e. if device A connected to channel 1, device B cannot connect to channel 1 until device A disconnects. With this option it is possible to make channels accept connections from several devices at the same time. I.e., if RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is TRUE both device A and device B can connect to channel 1 at the same time.
	RFCOMM_INFO_LEN	brief Maximum size of the data portion of a UIH frame. ingroup rfcomm_config details This parameter defines the maximum size of the data portion of a UIH frame. If CFC is used the actual length of the data portion will be 1 byte less. This value must be less than or equal to HCI_L2CAP_BUFFER_LEN - RFCOMM_FRAME_HEADER_LEN - L2CAP_HEADER_LEN.
	RFCOMM_LOCAL_CREDIT	brief The number of receive buffers. ingroup rfcomm_config details This parameter defines the number of received UIH frames that can be stored on the local device. The flow control mechanism used in RFCOMM ensures that the remote side of the link always knows how many free buffers left on the local device. When the number of free buffers reaches 0, the transmitter stops sending data frames until the receiver frees some buffers. The RFCOMM layer does not actually allocate space for buffers. It uses RFCOMM_LOCAL_CREDIT to keep track of free buffers and report them to the remote side. Actual memory... more
	RFCOMM_MAX_CMD_BUFFERS	brief Maximum number of command buffers. ingroup rfcomm_config details This parameter defines the maximum number of commands that can be sent at the same time. It is usually enough to reserve 2 buffers for each DLC excluding control DLC. Therefore, this value can be defined as rn #define RFCOMM_MAX_CMD_BUFFERS (RFCOMM_MAX_DLCS - 1) * 2
	RFCOMM_MAX_DLCS	brief Maximum number of DLCs ingroup rfcomm_config details This parameter defines the maximum number of DLCs on each session. This value should be at least 2 because each session uses one DLC to convey multiplexer control messages. All other DLCs are used to emulate serial ports.
	RFCOMM_MAX_SERVER_CHANNELS	brief Maximum number of Server channels ingroup rfcomm_config details This parameter defines the maximum number of server channels exposed by the local device. This value should not exceed RFCOMM_MAX_DLCS - 1.
	RFCOMM_MAX_SESSIONS	brief Maximum number of remote devices a local device can be connected to ingroup rfcomm_config details This parameter defines the maximum number of remote devices a local device can have simultaneous connections to. This value should not exceed HCI_MAX_HCI_CONNECTIONS.
	__RFCOMM_CONFIG_H	This is macro __RFCOMM_CONFIG_H.
	RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL	This is macro RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL.

SDP Configuration Macros

	Name	Description

	SDP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	• defgroup sdp_config Configuration
		• ingroup sdp
		*
		• This module describes parameters used to configure SDP.
		*
		• dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.
		* code #include "cdbt/bt/bt_std.h"
		// HCI and L2CAP must always be present // SDP is required only if stack is running in dual mode. This is the default mode. // To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol // must be defined: // #define BT_BLE_SINGLE_MODE
		// HCI configuration parameters #define... more

	SDP_ALLOCATE_BUFFERS_VARS	This is macro SDP_ALLOCATE_BUFFERS_VARS.

	SDP_MAX_ATTRIBUTE_RESULT_LEN	brief Maximum number of attributes to find ingroup sdp_config details This parameter defines the maximum number of attributes withing a service record the SDP server will return to the client.
	SDP_MAX_PDU_BUFFERS	brief Maximum number of SDP server PDU buffers. ingroup sdp_config details This parameter defines the maximum number of responses the SDP server can send at the same time.
	SDP_MAX_SEARCH_RESULT_LEN	brief Maximum number of service records to find. ingroup sdp_config details This parameter defines the maximum number of service records the SDP server will return to the client.
	__SDP_CONFIG_H	This is macro __SDP_CONFIG_H.

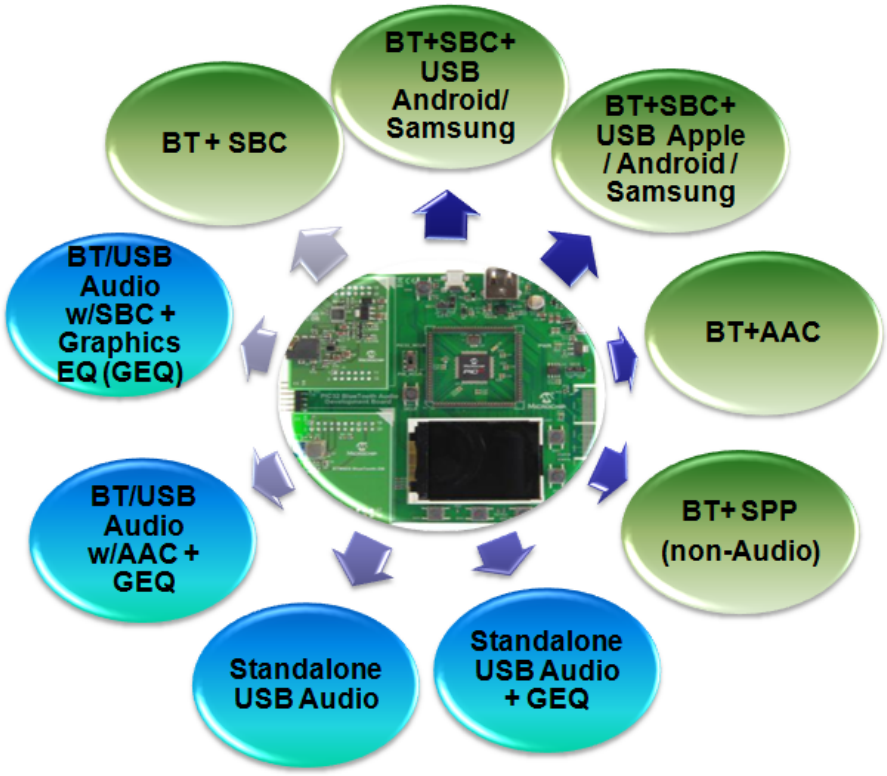
SPP Configuration Macros

	Name	Description
	SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	This is macro SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR.
	SPP_ALLOCATE_BUFFERS_VARS	This is macro SPP_ALLOCATE_BUFFERS_VARS.
	SPP_DECLARE_FRAME_BUFFERS	This is macro SPP_DECLARE_FRAME_BUFFERS.
	SPP_DISABLE_BUFFERING	This is macro SPP_DISABLE_BUFFERING.
	SPP_FRAME_BUFFERS_RAM_SIZE	This is macro SPP_FRAME_BUFFERS_RAM_SIZE.
	SPP_FRAME_BUFFERS_SIZE	This is macro SPP_FRAME_BUFFERS_SIZE.
	SPP_MAX_PORTS	brief Maximum number of SPP ports. ingroup spp_config details This parameter defines the maximum number of SPP port that can be open between the local and remote devices. If RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is FALSE (default) this value should be equal to RFCOMM_MAX_SERVER_CHANNELS. If RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is TRUE this value should be between RFCOMM_MAX_SERVER_CHANNELS and RFCOMM_MAX_SERVER_CHANNELS * RFCOMM_MAX_SESSIONS.
	__SPP_CONFIG_H	This is macro __SPP_CONFIG_H.

Description

Each major function, protocol and prototype within the library has a configuration header file. Generally these are not modified to utilize the PIC32 Bluetooth Stack with the demonstrations provided. The configuration files for each function are found in the `<install_dir>/framework/bluetooth/cdbt` directory for the PIC32 Bluetooth Basic Stack. The configuration is located in `<sub-directory>_config.h` for that particular function.

In the overall application a number of library controls are exposed to the user. These functions can be found in `user_config.h` in the application folder provided. The user of these configurations is referenced in the *"PIC32 Bluetooth Audio Development Kit Reference Guide*" (DS70005140), which is available from the Microchip website (www.microchip.com).



Bluetooth Support Configuration Macros

ATT_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
ATT_ALLOCATE_BUFFERS();
```

Description

This is function ATT_ALLOCATE_BUFFERS.

ATT_CLIENT_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
ATT_CLIENT_ALLOCATE_BUFFERS();
```

Description

This is function ATT_CLIENT_ALLOCATE_BUFFERS.

FTP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
FTP_ALLOCATE_BUFFERS();
```

Description

This is function FTP_ALLOCATE_BUFFERS.

GATT_CLIENT_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
GATT_CLIENT_ALLOCATE_BUFFERS();
```

Description

This is function GATT_CLIENT_ALLOCATE_BUFFERS.

HCI_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HCI_ALLOCATE_BUFFERS();
```

Description

This is function HCI_ALLOCATE_BUFFERS.

HFP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HFP_ALLOCATE_BUFFERS();
```

Description

This is function HFP_ALLOCATE_BUFFERS.

HID_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HID_ALLOCATE_BUFFERS();
```

Description

This is function HID_ALLOCATE_BUFFERS.

HSP_AG_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HSP_AG_ALLOCATE_BUFFERS();
```

Description

This is function HSP_AG_ALLOCATE_BUFFERS.

HSP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HSP_ALLOCATE_BUFFERS();
```

Description

This is function HSP_ALLOCATE_BUFFERS.

IAP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
IAP_ALLOCATE_BUFFERS();
```

Description

This is function IAP_ALLOCATE_BUFFERS.

IAP_BT_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
IAP_BT_ALLOCATE_BUFFERS();
```

Description

This is function IAP_BT_ALLOCATE_BUFFERS.

IAP2_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
IAP2_ALLOCATE_BUFFERS();
```

Description

This is function IAP2_ALLOCATE_BUFFERS.

L2CAP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
L2CAP_ALLOCATE_BUFFERS();
```

Description

This is function L2CAP_ALLOCATE_BUFFERS.

OBEX_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
OBEX_ALLOCATE_BUFFERS();
```

Description

This is function OBEX_ALLOCATE_BUFFERS.

PBAP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
PBAP_ALLOCATE_BUFFERS();
```

Description

This is function PBAP_ALLOCATE_BUFFERS.

RFCOMM_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
RFCOMM_ALLOCATE_BUFFERS();
```

Description

This is function RFCOMM_ALLOCATE_BUFFERS.

SDP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
SDP_ALLOCATE_BUFFERS();
```

Description

This is function SDP_ALLOCATE_BUFFERS.

SMP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
SMP_ALLOCATE_BUFFERS();
```

Description

This is function SMP_ALLOCATE_BUFFERS.

SPP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
SPP_ALLOCATE_BUFFERS();
```

Description

This is function SPP_ALLOCATE_BUFFERS.

BT_ENABLE_BLE Macro

File

bt_oem_config.h

C

```
#define BT_ENABLE_BLE
```

Description

This is macro BT_ENABLE_BLE.

BT_INCLUDE_RFCOMM Macro

File

bt_oem_config.h

C

```
#define BT_INCLUDE_RFCOMM
```

Description

This is macro BT_INCLUDE_RFCOMM.

BT_LOG_LEVEL_MAX Macro

File

bt_oem_config.h

C

```
#define BT_LOG_LEVEL_MAX BT_LOG_LEVEL_OFF
```

Description

This is macro BT_LOG_LEVEL_MAX.

BT_LOG_LEVEL_MIN Macro

File

bt_oem_config.h

C

```
#define BT_LOG_LEVEL_MIN BT_LOG_LEVEL
```

Description

This is macro BT_LOG_LEVEL_MIN.

__BT_APP_CONFIG_H Macro

File

bt_oem_config.h

C

```
#define __BT_APP_CONFIG_H
```

Description

This is macro __BT_APP_CONFIG_H.

BT_INCLUDE_IAP Macro

File

bt_oem_config.h

C

```
#define BT_INCLUDE_IAP
```

Description

This is macro BT_INCLUDE_IAP.

BT_INCLUDE_IAP2 Macro

File

bt_oem_config.h

C

```
#define BT_INCLUDE_IAP2
```

Description

This is macro BT_INCLUDE_IAP2.

IAP_BT_MAX_TRANSPORTS Macro

File

bt_oem_config.h

C

```
#define IAP_BT_MAX_TRANSPORTS IAPEA_MAX_SESSIONS
```

Description

This is macro IAP_BT_MAX_TRANSPORTS.

IAP_MAX_SESSIONS Macro

File

bt_oem_config.h

C

```
#define IAP_MAX_SESSIONS IAPEA_MAX_SESSIONS
```

Description

This is macro IAP_MAX_SESSIONS.

IAP_RX_BUFFER_SIZE Macro

File

bt_oem_config.h

C

```
#define IAP_RX_BUFFER_SIZE IAPEA_MAX_PACKET_SIZE
```

Description

This is macro IAP_RX_BUFFER_SIZE.

IAP2_MAX_PACKET_SIZE Macro

File

bt_oem_config.h

C

```
#define IAP2_MAX_PACKET_SIZE IAPEA_MAX_PACKET_SIZE
```

Description

This is macro IAP2_MAX_PACKET_SIZE.

IAP2_MAX_SESSIONS Macro

File

bt_oem_config.h

C

```
#define IAP2_MAX_SESSIONS IAPEA_MAX_SESSIONS
```

Description

This is macro IAP2_MAX_SESSIONS.

AVCTP Configuration Macros

__AVCTP_CONFIG_H Macro

File

avctp_config.h

C

```
#define __AVCTP_CONFIG_H
```

Description

This is macro __AVCTP_CONFIG_H.

AVCTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

avctp_config.h

C

```
#define AVCTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

- defgroup avctp_config Configuration

- ingroup avctp

*

- This module describes parameters used to configure AVCTP layer.

*

- dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

* code #include "cdbt/bt/bt_std.h"

// HCI, L2CAP and SDP must always be present

// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...

// L2CAP configuration parameters #define L2CAP_MAX_CMD_BUFFERS ... #define L2CAP_MAX_FRAME_BUFFERS ... #define L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...

// SDP configuration parameters #define SDP_MAX_SEARCH_RESULT_LEN ... #define SDP_MAX_ATTRIBUTE_RESULT_LEN ...

// Depending on protocols and profiles used below goes configuration parameters // for each used module. E.g., to use and configure AVRCP, // the following values must be defined:

#define BT_INCLUDE_AVCTP // tells dotstack to compile in AVCTP support #define AVCTP_MAX_CHANNELS ... #define AVCTP_MAX_TRANSPORT_CHANNELS ... #define AVCTP_MAX_RX_MESSAGE_LEN ... #define AVCTP_MAX_MESSAGE_BUFFERS ...

#include "cdbt/bt/bt_oem_config.h"

endcode

AVCTP_ALLOCATE_BUFFERS_VARS Macro

File

avctp_config.h

C

```
#define AVCTP_ALLOCATE_BUFFERS_VARS \
    bt_avctp_channel_t                  _avctp_channels[AVCTP_MAX_CHANNELS];    \
    const bt_byte                       _avctp_max_channels = AVCTP_MAX_CHANNELS; \
    bt_avctp_transport_t                _avctp_transports[AVCTP_MAX_TRANSPORT_CHANNELS];    \
    const bt_byte                       _avctp_max_transports = AVCTP_MAX_TRANSPORT_CHANNELS; \
    const bt_uint                       _avctp_max_rx_message_len = AVCTP_MAX_RX_MESSAGE_LEN; \
    bt_byte                             _avctp_rx_buffers[(AVCTP_MAX_RX_MESSAGE_LEN) *
(AVCTP_MAX_TRANSPORT_CHANNELS)]; \
    bt_buffer_header_t                  _avctp_message_buffer_headers[(AVCTP_MAX_MESSAGE_BUFFERS) *
(AVCTP_MAX_TRANSPORT_CHANNELS)];    \
    bt_avctp_message_t                  _avctp_message_buffers[(AVCTP_MAX_MESSAGE_BUFFERS) *
(AVCTP_MAX_TRANSPORT_CHANNELS)];    \
    const bt_byte                       _avctp_max_message_buffers = AVCTP_MAX_MESSAGE_BUFFERS; \
    \
    AVCTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR    \
```

Description

brief Maximum number of message buffers ingroup avctp_config

details This parameter defines the maximum number of buffer available for sending message.

AVCTP_MAX_CHANNELS Macro

File

avctp_config.h

C

```
#define AVCTP_MAX_CHANNELS
```

Description

brief Maximum number of AVCTP channels ingroup avctp_config

details This parameter defines the maximum number of channels a local device can have with remote devices.

AVCTP_MAX_TRANSPORT_CHANNELS Macro

File

avctp_config.h

C

```
#define AVCTP_MAX_TRANSPORT_CHANNELS
```

Description

brief Maximum number of AVCTP transports ingroup avctp_config

details This parameter defines the maximum number of transports a local device can have with remote devices. This value should not exceed AVCTP_MAX_CHANNELS.

AVDTP Configuration Macros

AVDTP_ALLOCATE_BUFFERS_FUNCTION Macro

File

avdtp_config.h

C

```
#define AVDTP_ALLOCATE_BUFFERS_FUNCTION \
    void _bt_avdtp_allocate_buffers(void)    \
    {    \
    } \
```

Description

This is macro AVDTP_ALLOCATE_BUFFERS_FUNCTION.

AVDTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

avdtp_config.h

C

```
#define AVDTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

- defgroup avdtp_config Configuration
- ingroup avdtp
- *
- This module describes parameters used to configure AVDTP layer.
- *
- dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

* code #include "cdbt/bt/bt_std.h"

// HCI, L2CAP and SDP must always be present

// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...

// L2CAP configuration parameters #define L2CAP_MAX_CMD_BUFFERS ... #define L2CAP_MAX_FRAME_BUFFERS ... #define L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...

// SDP configuration parameters #define SDP_MAX_SEARCH_RESULT_LEN ... #define SDP_MAX_ATTRIBUTE_RESULT_LEN ...

// Depending on protocols and profiles used below goes configuration parameters // for each used module. E.g., to use and configure AVDTP & A2DP (this one does not need configuration), // the following values must be defined:

#define BT_INCLUDE_AVDTP // tells dotstack to compile in AVDTP support #define AVDTP_MAX_SEP ... #define AVDTP_MAX_STREAMS ... #define AVDTP_MAX_REMOTE_DEVICES ... #define AVDTP_MAX_CMD_BUFFERS ... #define AVDTP_MAX_TRANSPORT_CHANNELS ... #define AVDTP_MAX_TX_BUFFER_LEN ... #define AVDTP_MAX_CMD_PARAM_LEN ... #define AVDTP_CODEC_CONFIG_BUFFER_LEN ...

#include "cdbt/bt/bt_oem_config.h"
endcode

AVDTP_ALLOCATE_BUFFERS_VARS Macro

File

avdtp_config.h

C

```
#define AVDTP_ALLOCATE_BUFFERS_VARS \
    bt_avdtp_sep_t                      _avdtp_seps[AVDTP_MAX_SEP];    \
    const bt_byte                       _avdtp_max_seps = AVDTP_MAX_SEP; \
    bt_avdtp_stream_t                   _avdtp_streams[AVDTP_MAX_STREAMS];    \
    const bt_byte                       _avdtp_max_streams = AVDTP_MAX_STREAMS; \
    bt_avdtp_control_channel_t          _avdtp_control_channels[AVDTP_MAX_REMOTE_DEVICES];    \
    const bt_byte                       _avdtp_max_control_channels = AVDTP_MAX_REMOTE_DEVICES; \
    const bt_byte                       _avdtp_max_tx_buffer_len = AVDTP_MAX_TX_BUFFER_LEN; \
    bt_byte                             _avdtp_tx_buffers[(AVDTP_MAX_TX_BUFFER_LEN) *
(AVDTP_MAX_REMOTE_DEVICES)]; \
    bt_avdtp_transport_channel_t        _avdtp_transport_channels[AVDTP_MAX_TRANSPORT_CHANNELS];    \
    const bt_byte                       _avdtp_max_transport_channels = AVDTP_MAX_TRANSPORT_CHANNELS; \
    bt_buffer_header_t                  _avdtp_cmd_buffer_headers[(AVDTP_MAX_CMD_BUFFERS) *
(AVDTP_MAX_REMOTE_DEVICES)];    \
    bt_avdtp_control_cmd_t              _avdtp_cmd_buffers[(AVDTP_MAX_CMD_BUFFERS) *
(AVDTP_MAX_REMOTE_DEVICES)];    \
    const bt_byte                       _avdtp_max_cmd_buffers = AVDTP_MAX_CMD_BUFFERS; \
    bt_byte                             _avdtp_cmd_param_buffers[(AVDTP_MAX_CMD_BUFFERS) *
(AVDTP_MAX_REMOTE_DEVICES) * (AVDTP_MAX_CMD_PARAM_LEN)];    \
    const bt_uint                       _avdtp_max_cmd_param_len = AVDTP_MAX_CMD_PARAM_LEN;    \
    bt_byte                             _avdtp_codec_cfg_buffers[(AVDTP_MAX_STREAMS) * 2 *
(AVDTP_CODEC_CONFIG_BUFFER_LEN)];    \
    const bt_byte                       _avdtp_max_codec_config_buffer_len =
AVDTP_CODEC_CONFIG_BUFFER_LEN;    \
    bt_buffer_header_t                  _avdtp_sep_cfg_buffer_headers[(AVDTP_MAX_STREAMS) * 2];    \
    bt_avdtp_sep_capabilities_t         _avdtp_sep_cfg_buffers[(AVDTP_MAX_STREAMS) * 2];    \
    bt_avdtp_sep_capabilities_t         _avdtp_rcv_sep_caps;    \
    bt_byte                             _avdtp_rcv_sep_codec_cfg_buffer[(AVDTP_CODEC_CONFIG_BUFFER_LEN)];
\
    bt_byte                             _avdtp_listen_sep_buffers[(AVDTP_MAX_STREAMS) * (AVDTP_MAX_SEP)];
\
    bt_byte                             _avdtp_l2cap_media_packet_buffer[HCI_L2CAP_BUFFER_LEN -
L2CAP_HEADER_LEN];    \
    \
    AVDTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR    \
```

Description

This is macro AVDTP_ALLOCATE_BUFFERS_VARS.

AVDTP_CMD_ABORT Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_ABORT 10
```

Description

This is macro AVDTP_CMD_ABORT.

AVDTP_CMD_CLOSE Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_CLOSE 8
```

Description

This is macro AVDTP_CMD_CLOSE.

AVDTP_CMD_DISCOVER Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_DISCOVER 1
```

Description

This is macro AVDTP_CMD_DISCOVER.

AVDTP_CMD_GET_CAPABILITIES Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_GET_CAPABILITIES 2
```

Description

This is macro AVDTP_CMD_GET_CAPABILITIES.

AVDTP_CMD_GET_CONFIGURATION Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_GET_CONFIGURATION 4
```

Description

This is macro AVDTP_CMD_GET_CONFIGURATION.

AVDTP_CMD_OPEN Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_OPEN 6
```

Description

This is macro AVDTP_CMD_OPEN.

AVDTP_CMD_RECONFIGURE Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_RECONFIGURE 5
```

Description

This is macro AVDTP_CMD_RECONFIGURE.

AVDTP_CMD_SECURITY_CONTROL Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_SECURITY_CONTROL 11
```

Description

This is macro AVDTP_CMD_SECURITY_CONTROL.

AVDTP_CMD_SET_CONFIGURATION Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_SET_CONFIGURATION 3
```

Description

This is macro AVDTP_CMD_SET_CONFIGURATION.

AVDTP_CMD_START Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_START 7
```

Description

This is macro AVDTP_CMD_START.

AVDTP_CMD_SUSPEND Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_SUSPEND 9
```

Description

This is macro AVDTP_CMD_SUSPEND.

AVDTP_CODEC_CONFIG_BUFFER_LEN Macro

File

avdtp_config.h

C

```
#define AVDTP_CODEC_CONFIG_BUFFER_LEN 16
```

Description

brief Size of the buffer used to store codec specific configuration. ingroup avdtp_config

details Each codec uses unique configuration which can take different amount of memory. This parameter defines the size of the buffer for storing codec's configuration. The value of 16 is sufficient for SBC, AAC and MPEG1,2. If vendor specific codec is to be used this value may need to increased.

AVDTP_MAX_CMD_BUFFERS Macro

File

avdtp_config.h

C

```
#define AVDTP_MAX_CMD_BUFFERS
```

Description

brief Maximum number of command buffers. ingroup avdtp_config

details This parameter defines the number of buffers reserved for sending commands to a remote device over its control channel. Each channel uses its own buffers so the total number of buffers is AVDTP_MAX_REMOTE_DEVICES * AVDTP_MAX_CMD_BUFFERS. The minimum value is 1. The maximum value is 255. 2 is usually sufficient.

AVDTP_MAX_CMD_PARAM_LEN Macro

File

avdtp_config.h

C

```
#define AVDTP_MAX_CMD_PARAM_LEN (AVDTP_MAX_TX_BUFFER_LEN - 2)
```

Description

brief Maximum length of control command parameters ingroup avdtp_config

details This parameter defines the maximum length of all command parameters. The value should not exceed AVDTP_MAX_TX_BUFFER_LEN - 2 (command header).

AVDTP_MAX_REMOTE_DEVICES Macro

File

avdtp_config.h

C

```
#define AVDTP_MAX_REMOTE_DEVICES
```

Description

brief Maximum number of remote devices a local device can be connected to ingroup avdtp_config

details This parameter defines the number of remote devices a local device can have simultaneous connections to (i.e. control channels). This value should not exceed AVDTP_MAX_STREAMS. For each remote device AVDTP creates one control channel regardless of the number of streams between the local and the remote devices. Assuming that the local devices wants to have only one channel with each remote device and if AVDTP_MAX_REMOTE_DEVICES > AVDTP_MAX_STREAMS all memory reserved for devices in excess of AVDTP_MAX_STREAMS will be wasted. The minimum value is 1. The maximum value is 255.

AVDTP_MAX_SEP Macro

File

avdtp_config.h

C

```
#define AVDTP_MAX_SEP
```

Description

brief Maximum number of SEPs that can be exposed by a local device ingroup avdtp_config

details This parameter defines the number of SEPs an application can expose to remote devices. The minimum value is 1. The maximum value is 255.

AVDTP_MAX_STREAMS Macro

File

avdtp_config.h

C

```
#define AVDTP_MAX_STREAMS
```

Description

brief Maximum number of streams that can be exposed by a local device ingroup avdtp_config

details This parameter defines the number of streams an application can open between local and remote devices. This value can be different from AVDTP_MAX_SEP but should not exceed it. Since each SEP can only be used once the local device can only have as much streams as there are SEPs. If AVDTP_MAX_STREAMS > AVDTP_MAX_SEP all memory reserved for streams in excess of AVDTP_MAX_SEP will be wasted. The minimum value is 1. The maximum value is 255.

AVDTP_MAX_TRANSPORT_CHANNELS Macro

File

avdtp_config.h

C

```
#define AVDTP_MAX_TRANSPORT_CHANNELS
```

Description

brief Maximum number of transport channels. ingroup avdtp_config

details Depending on the SEP capabilities (multiplexing, recovery, reporting) each stream may need up to 3 transport channels. E.g., if multiplexing is not supported and recovery and reporting are supported a stream will use 3 transport channels - 1 for media transport session, 1 for recovery transport session and 1 for reporting transport session. If multiplexing is not supported and only reporting is supported the stream will use 2 transport channels - 1 for media transport session and 1 for reporting transport session. If multiplexing is supported the stream will need only 1 transport channel for all supported transport session. With multiplexing even different streams can share the same transport channel. dotstack currently does not support multiplexing, recovery and reporting. So each stream needs its own transport channel for it media transport session. Hence, AVDTP_MAX_TRANSPORT_CHANNELS must be equal to AVDTP_MAX_STREAMS

AVDTP_MAX_TX_BUFFER_LEN Macro

File

avdtp_config.h

C

```
#define AVDTP_MAX_TX_BUFFER_LEN 32
```

Description

brief Size of the transmit buffer. ingroup avdtp_config

details This parameter defines the size of the buffer used to send AVDTP control commands to L2CAP layer. Each control channel has it own buffer so the total amount of memory allocated for these buffers is AVDTP_MAX_TX_BUFFER_LEN) * AVDTP_MAX_REMOTE_DEVICES. The minimum value is 1. The maximum value is 255. The value of 32 is usually sufficient.

AVRCP Configuration Macros

__AVRCP_CONFIG_H Macro

File

avrcp_config.h

C

```
#define __AVRCP_CONFIG_H
```

Description

This is macro __AVRCP_CONFIG_H.

AVRCP_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

avrcp_config.h

C

```
#define AVRCP_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

- defgroup avrcp_config Configuration
- ingroup avrcp

*

- This module describes parameters used to configure AVRCP layer.

*

- dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

* code #include "cdbt/bt/bt_std.h"

// HCI, L2CAP and SDP must always be present

// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...

// L2CAP configuration parameters #define L2CAP_MAX_CMD_BUFFERS ... #define L2CAP_MAX_FRAME_BUFFERS ... #define L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...

// SDP configuration parameters #define SDP_MAX_SEARCH_RESULT_LEN ... #define SDP_MAX_ATTRIBUTE_RESULT_LEN ...

// Depending on protocols and profiles used below goes configuration parameters // for each used module. E.g., to use and configure AVRCP, // the following values must be defined:

#define BT_INCLUDE_AVCTP // tells dotstack to compile in AVCTP support #define AVCTP_MAX_CHANNELS ... #define AVCTP_MAX_TRANSPORT_CHANNELS ... #define AVCTP_MAX_RX_MESSAGE_LEN ... #define AVCTP_MAX_MESSAGE_BUFFERS ...

#define BT_INCLUDE_AVRCP // tells dotstack to compile in AVRCP support #define AVRCP_MAX_CHANNELS ... #define AVRCP_MAX_CMD_BUFFERS ... #define AVRCP_MAX_CMD_PARAM_LEN ... #define AVRCP_MAX_SEARCH_RESULTS ... #define AVRCP_MAX_DEVICE_NAME_LEN ... #define AVRCP_CMD_TIMEOUT ...

#include "cdbt/bt/bt_oem_config.h"

endcode

AVRCP_ALLOCATE_BUFFERS_VARS Macro

File

avrcp_config.h

C

```
#define AVRCP_ALLOCATE_BUFFERS_VARS \
    bt_avrcp_channel_t  _avrcp_channels[AVRCP_MAX_CHANNELS];    \
    const bt_byte       _avrcp_max_channels = AVRCP_MAX_CHANNELS; \
    \
    bt_buffer_header_t  _avrcp_cmd_buffer_headers[(AVRCP_MAX_CMD_BUFFERS) * (AVRCP_MAX_CHANNELS)];    \
    bt_av_command_t     _avrcp_cmd_buffers[(AVRCP_MAX_CMD_BUFFERS) * (AVRCP_MAX_CHANNELS)];    \
    const bt_byte       _avrcp_max_cmd_buffers = AVRCP_MAX_CMD_BUFFERS; \
    bt_byte             _avrcp_cmd_param_buffers[(AVRCP_MAX_CMD_BUFFERS) * (AVRCP_MAX_CMD_PARAM_LEN) *
(AVRCP_MAX_CHANNELS)];    \
    const bt_int        _avrcp_max_cmd_param_len = AVRCP_MAX_CMD_PARAM_LEN;    \
    const bt_byte       _avrcp_max_search_results = AVRCP_MAX_SEARCH_RESULTS;    \
    bt_avrcp_device_t   _avrcp_devices_buffer[AVRCP_MAX_SEARCH_RESULTS];    \
    const bt_byte       _avrcp_max_device_name_len = AVRCP_MAX_DEVICE_NAME_LEN;    \
    bt_byte             _avrcp_device_name_buffers[(AVRCP_MAX_SEARCH_RESULTS) * (AVRCP_MAX_DEVICE_NAME_LEN
+ 1)];    \
    const bt_byte       _avrcp_cmd_timeout = (AVRCP_CMD_TIMEOUT) / 100;    \
    \
    void (*_bt_avrcp_response_sent_handler)(bt_avrcp_mgr_t* mgr, bt_avctp_evt_response_sent_t* evt_param) =
AVRCP_RESPONSE_SENT_HANDLER;    \
    void (*_bt_avrcp_command_handler)(bt_avrcp_mgr_t* mgr, bt_avctp_evt_command_received_t* evt_param) =
```

```
AVRCP_COMMAND_HANDLER;    \
    void (*_bt_avrcp_command_sent_handler)(bt_avrcp_mgr_t* mgr, bt_avctp_evt_command_sent_t* evt_param) =
AVRCP_COMMAND_SENT_HANDLER;    \
    void (*_bt_avrcp_response_handler)(bt_avrcp_mgr_t* mgr, bt_avctp_evt_response_received_t* evt_param) =
AVRCP_RESPONSE_HANDLER;    \
    \
    AVRCP_ALLOCATE_BUFFERS_RAM_SIZE_VAR    \
```

Description

This is macro AVRCP_ALLOCATE_BUFFERS_VARS.

AVRCP_CMD_TIMEOUT Macro

File

avrcp_config.h

C

```
#define AVRCP_CMD_TIMEOUT 10000
```

Description

brief Command timeout ingroup avrcp_config

details This parameter defines the amount of time in milliseconds AVRCP waits for a response to a request. If not defined the default value of 10000 (10 secconds) is used.

AVRCP_MAX_CHANNELS Macro

File

avrcp_config.h

C

```
#define AVRCP_MAX_CHANNELS
```

Description

brief Maximum number of remote devices a local device can be connected to ingroup avrcp_config

details This parameter defines the number of remote devices a local device can have simultaneous connections to (i.e. channels). This value should not exceed AVCTP_MAX_CHANNELS.

AVRCP_MAX_CMD_BUFFERS Macro

File

avrcp_config.h

C

```
#define AVRCP_MAX_CMD_BUFFERS 1
```

Description

brief Maximum number of command buffers. ingroup avrcp_config

details This parameter defines the number of buffers reserved for sending commands to a remote device over its control channel. Each channel uses its own buffers so the total number of buffers is AVRCP_MAX_CHANNELS * AVRCP_MAX_CMD_BUFFERS. The minimum value is 1. The maximum value is 255. If not define one buffer for each channel is reserved.

AVRCP_MAX_CMD_PARAM_LEN Macro

File

avrcp_config.h

C

```
#define AVRCP_MAX_CMD_PARAM_LEN 512
```

Description

brief Maximum length of command parameters ingroup avrcp_config

details This parameter defines the maximum length of all command parameters. If not defined the default value of 512 is used.

AVRCP_MAX_DEVICE_NAME_LEN Macro

File

avrcp_config.h

C

```
#define AVRCP_MAX_DEVICE_NAME_LEN 20
```

Description

brief Maximum length of device name ingroup avrcp_config

details This parameter defines the size of the buffer used to store device's name while searching for nearby targets with bt_avrcp_find_targets. If the name of the device is longer than AVRCP_MAX_DEVICE_NAME_LEN it is truncated to AVRCP_MAX_DEVICE_NAME_LEN. If not defined the default value of 20 is used.

AVRCP_MAX_SEARCH_RESULTS Macro

File

avrcp_config.h

C

```
#define AVRCP_MAX_SEARCH_RESULTS 7
```

Description

brief Maximum number of devices to find ingroup avrcp_config

details This parameter defines the maximum number of devices bt_avrcp_find_targets can find. If not defined the default value of 7 is used.

HCI Configuration Macros

BT_ENABLE_SCO Macro

File

hci_config.h

C

```
#define BT_ENABLE_SCO BT_FALSE
```

Description

This is macro BT_ENABLE_SCO.

BT_HCI_EVT_AUTHENTICATION_COMPLETE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_AUTHENTICATION_COMPLETE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_AUTHENTICATION_COMPLETE_HANDLER.

BT_HCI_EVT_CONNECTION_COMPLETE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_CONNECTION_COMPLETE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_CONNECTION_COMPLETE_HANDLER.

BT_HCI_EVT_CONNECTION_REQUEST_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_CONNECTION_REQUEST_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_CONNECTION_REQUEST_HANDLER.

BT_HCI_EVT_ENCRYPTION_CHANGE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_ENCRYPTION_CHANGE_HANDLER bt_hci_evt_encryption_change_handler
```

Description

This is macro BT_HCI_EVT_ENCRYPTION_CHANGE_HANDLER.

BT_HCI_EVT_EXTENDED_INQUIRY_RESULT_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_EXTENDED_INQUIRY_RESULT_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_EXTENDED_INQUIRY_RESULT_HANDLER.

BT_HCI_EVT_INQUIRY_COMPLETE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_INQUIRY_COMPLETE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_INQUIRY_COMPLETE_HANDLER.

BT_HCI_EVT_INQUIRY_RESULT_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_INQUIRY_RESULT_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_INQUIRY_RESULT_HANDLER.

BT_HCI_EVT_INQUIRY_RESULT_WITH_RSSI_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_INQUIRY_RESULT_WITH_RSSI_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_INQUIRY_RESULT_WITH_RSSI_HANDLER.

BT_HCI_EVT_LINK_KEY_NOTIFICATION_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_LINK_KEY_NOTIFICATION_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_LINK_KEY_NOTIFICATION_HANDLER.

BT_HCI_EVT_LINK_KEY_REQUEST_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_LINK_KEY_REQUEST_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_LINK_KEY_REQUEST_HANDLER.

BT_HCI_EVT_MODE_CHANGE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_MODE_CHANGE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_MODE_CHANGE_HANDLER.

BT_HCI_EVT_PIN_CODE_REQUEST_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_PIN_CODE_REQUEST_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_PIN_CODE_REQUEST_HANDLER.

BT_HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE_HANDLER.

BT_HCI_EVT_ROLE_CHANGE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_ROLE_CHANGE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_ROLE_CHANGE_HANDLER.

BT_HCI_EVT_SYNCH_CONNECTION_COMPLETE_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_HCI_EVT_SYNCH_CONNECTION_COMPLETE_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_HCI_EVT_SYNCH_CONNECTION_COMPLETE_HANDLER.

BT_LE_EVT_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_LE_EVT_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_LE_EVT_HANDLER.

BT_SSP_EVT_HANDLER Macro

File

hci_config_event_handlers.h

C

```
#define BT_SSP_EVT_HANDLER bt_hci_evt_default_handler
```

Description

This is macro BT_SSP_EVT_HANDLER.

HCI_ALLOCATE_BUFFERS_FUNCTION Macro

File

hci_config.h

C

```
#define HCI_ALLOCATE_BUFFERS_FUNCTION \
    void _hci_allocate_buffers(void)    \
    {    \
        bt_byte i;    \
        _phci_ctrl->connections = _hci_connections;    \
        _zero_memory(_hci_connections, sizeof(bt_hci_conn_state_t) * HCI_MAX_HCI_CONNECTIONS);    \
        for (i = 0; i < HCI_MAX_HCI_CONNECTIONS; i++)    \
        {    \
            _hci_connections[i].recv_data = &_conn_state_recv_buffers[i * (HCI_L2CAP_BUFFER_LEN)];    \
            HCI_INIT_LE_CONN_STATES;    \
        }    \
        HCI_INIT_SCO_HANDLERS    \
        HCI_INIT_SSP_HANDLERS    \
        HCI_INIT_LE_CTRL_STATE    \
    } \
```

Description

This is macro HCI_ALLOCATE_BUFFERS_FUNCTION.

HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

hci_config.h

C

```
#define HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

This is macro HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR.

HCI_ALLOCATE_BUFFERS_VARS Macro

File

hci_config.h

C

```
#define HCI_ALLOCATE_BUFFERS_VARS \
    bt_buffer_header_t  _hci_cmd_buffer_headers[HCI_MAX_CMD_BUFFERS];    \
    bt_hci_command_t    _hci_cmd_buffers[HCI_MAX_CMD_BUFFERS];    \
    const bt_byte       _hci_max_cmd_buffers = HCI_MAX_CMD_BUFFERS; \
    bt_byte             _hci_cmd_param_buffers[HCI_MAX_CMD_BUFFERS * HCI_MAX_CMD_PARAM_LEN];    \
    const bt_byte       _hci_max_cmd_param_len = HCI_MAX_CMD_PARAM_LEN;    \
    \
    bt_buffer_header_t  _hci_send_data_buffer_headers[HCI_MAX_DATA_BUFFERS];    \
    bt_hci_data_t       _hci_send_data_buffers[HCI_MAX_DATA_BUFFERS];    \
    const bt_byte       _hci_max_data_buffers = HCI_MAX_DATA_BUFFERS; \
    \
    bt_hci_conn_state_t _hci_connections[HCI_MAX_HCI_CONNECTIONS];    \
    const bt_byte       _hci_max_hci_connections = HCI_MAX_HCI_CONNECTIONS; \
    \
    bt_byte             _recv_buffer[HCI_RX_BUFFER_LEN];    \
    bt_byte             _send_buffer[HCI_TX_BUFFER_LEN];    \
    const bt_uint       _hci_rcv_buffer_len = HCI_RX_BUFFER_LEN;    \
    const bt_uint       _hci_tx_buffer_len = HCI_TX_BUFFER_LEN;    \
    \
    bt_byte             _conn_state_recv_buffers[(HCI_L2CAP_BUFFER_LEN) * (HCI_MAX_HCI_CONNECTIONS)];    \
    const bt_uint       _hci_l2cap_buffer_len = HCI_L2CAP_BUFFER_LEN;    \
    \
    const bt_bool       _hci_enable_ctrl_to_host_flow_control = HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL;    \
```

```
\
const bt_byte       _hci_max_connect_attempts = HCI_MAX_CONNECT_ATTEMPTS;    \
\
const bt_bool       _hci_enable_sco = BT_ENABLE_SCO;    \
void (*_hci_recv_sco_data_packet_fp)(bt_byte* pbuf);    \
\
void (*_bt_ssp_init)(void);    \
void (*_bt_ssp_evt_handler)(bt_hci_event_t* evt);    \
\
void (*_bt_hci_le_init)(bt_hci_le_ctrl_state_t* le_ctrl_state);    \
HCI_DECLARE_LE_CONN_STATES    \
HCI_DECLARE_LE_CTRL_STATE    \
\
HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR    \
```

Description

This is macro HCI_ALLOCATE_BUFFERS_VARS.

HCI_DECLARE_LE_CONN_STATES Macro

File

hci_config.h

C

```
#define HCI_DECLARE_LE_CONN_STATES
```

Description

This is macro HCI_DECLARE_LE_CONN_STATES.

HCI_DECLARE_LE_CTRL_STATE Macro

File

hci_config.h

C

```
#define HCI_DECLARE_LE_CTRL_STATE
```

Description

This is macro HCI_DECLARE_LE_CTRL_STATE.

HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL Macro

File

hci_config.h

C

```
#define HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL BT_FALSE
```

Description

This is macro HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL.

HCI_INIT_LE_CONN_STATES Macro

File

hci_config.h

C

```
#define HCI_INIT_LE_CONN_STATES \
    _hci_connections[i].le_conn_state = NULL;
```

Description

This is macro HCI_INIT_LE_CONN_STATES.

HCI_INIT_LE_CTRL_STATE Macro

File

hci_config.h

C

```
#define HCI_INIT_LE_CTRL_STATE \
    _bt_hci_le_init = NULL;
```

Description

This is macro HCI_INIT_LE_CTRL_STATE.

HCI_INIT_SCO_HANDLERS Macro

File

hci_config.h

C

```
#define HCI_INIT_SCO_HANDLERS \
    _hci_recv_sco_data_packet_fp = NULL;
```

Description

This is macro HCI_INIT_SCO_HANDLERS.

HCI_INIT_SSP_HANDLERS Macro

File

hci_config.h

C

```
#define HCI_INIT_SSP_HANDLERS \
    _bt_ssp_evt_handler = NULL;    \
    _bt_ssp_init = NULL;
```

Description

This is macro HCI_INIT_SSP_HANDLERS.

HCI_L2CAP_BUFFER_LEN Macro

File

hci_config.h

C

```
#define HCI_L2CAP_BUFFER_LEN (HCI_RX_BUFFER_LEN - HCI_ACL_DATA_HEADER_LEN - HCI_TRANSPORT_HEADER_LEN)
```

Description

This is macro HCI_L2CAP_BUFFER_LEN.

HCI_SIZEOF_LE_CONN_STATES Macro

File

hci_config.h

C

```
#define HCI_SIZEOF_LE_CONN_STATES 0
```

Description

This is macro HCI_SIZEOF_LE_CONN_STATES.

HCI_SIZEOF_LE_CTRL_STATE Macro

File

hci_config.h

C

```
#define HCI_SIZEOF_LE_CTRL_STATE 0
```

Description

This is macro HCI_SIZEOF_LE_CTRL_STATE.

HCI_TX_BUFFER_LEN Macro

File

hci_config.h

C

```
#define HCI_TX_BUFFER_LEN HCI_RX_BUFFER_LEN
```

Description

This is macro HCI_TX_BUFFER_LEN.

__HCI_CONFIG_EVENT_HANDLERS_H Macro

File

hci_config_event_handlers.h

C

```
#define __HCI_CONFIG_EVENT_HANDLERS_H
```

Description

This is macro __HCI_CONFIG_EVENT_HANDLERS_H.

__HCI_CONFIG_H Macro

File

hci_config.h

C

```
#define __HCI_CONFIG_H
```

Description

This is macro __HCI_CONFIG_H.

HCI_MAX_CONNECT_ATTEMPTS Macro

File

hci_config.h

C

```
#define HCI_MAX_CONNECT_ATTEMPTS 4
```

Description

This is macro HCI_MAX_CONNECT_ATTEMPTS.

L2CAP Configuration Macros

L2CAP_ALLOCATE_BUFFERS_FUNCTION Macro

File

l2cap_config.h

C

```
#define L2CAP_ALLOCATE_BUFFERS_FUNCTION \
    void _l2cap_allocate_buffers()    \
    {    \
        bt_byte i;    \
        _l2cap_hci_connect_packet_type = L2CAP_HCI_PACKET_TYPE;    \
        _l2cap_hci_page_scan_repetition_mode = L2CAP_HCI_PAGE_SCAN_REPETITION_MODE;    \
        _l2cap_hci_role_switch = L2CAP_HCI_ROLE_SWITCH;    \
        _l2cap_idle_hci_connection_timeout = L2CAP_IDLE_CONNECTION_TIMEOUT;    \
        \
        _zero_memory(_l2cap_psms, sizeof(bt_l2cap_psm_t) * (L2CAP_MAX_PSMS) * (L2CAP_MAX_MANAGERS));    \
        _zero_memory(_l2cap_channels, sizeof(bt_l2cap_channel_t) * (L2CAP_MAX_CHANNELS) *
(L2CAP_MAX_MANAGERS));    \
        if (_l2cap_channels_ext)    \
            _zero_memory(_l2cap_channels_ext, sizeof(bt_l2cap_channel_ext_t) * (L2CAP_MAX_CHANNELS) *
(L2CAP_MAX_MANAGERS));    \
        if (_l2cap_max_fixed_channels)    \
            _zero_memory(_l2cap_fixed_channels, sizeof(bt_l2cap_fixed_channel_t) *
_l2cap_max_fixed_channels);    \
        \
        for (i = 0; i < L2CAP_MAX_MANAGERS; i++)    \
        {    \
            _mgrs[i]._psms = &_l2cap_psms[i * L2CAP_MAX_PSMS];    \
            _mgrs[i]._channels = &_l2cap_channels[i * (L2CAP_MAX_CHANNELS)];    \
            bt_init_buffer_mgr(&_mgrs[i].connect_params_mgr, L2CAP_MAX_CHANNELS,
sizeof(bt_l2cap_connect_params_t), &_l2cap_connect_params_headers[(L2CAP_MAX_CHANNELS) * i],
&_l2cap_connect_params[(L2CAP_MAX_CHANNELS) * i]);    \
            \
            if (_l2cap_channels_ext)    \
            {    \
                int j;    \
                \
                for (j = 0; j < L2CAP_MAX_CHANNELS; j++)    \
                {    \
                    _mgrs[i]._channels[j].ext = &_l2cap_channels_ext[i * (L2CAP_MAX_CHANNELS) + j];    \
                }    \
            }    \
            if (_l2cap_max_fixed_channels)    \
                _mgrs[i]._fixed_channels = &_l2cap_fixed_channels[i * _l2cap_max_fixed_channels];    \
        }    \
    }    \
```

Description

This is macro L2CAP_ALLOCATE_BUFFERS_FUNCTION.

L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

l2cap_config.h

C

```
#define L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

This is macro L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR.

L2CAP_ALLOCATE_BUFFERS_VARS Macro

File

l2cap_config.h

C

```
#define L2CAP_ALLOCATE_BUFFERS_VARS \
    bt_buffer_header_t  _l2cap_cmd_buffer_headers[L2CAP_MAX_CMD_BUFFERS];    \
    bt_l2cap_command_t  _l2cap_cmd_buffers[L2CAP_MAX_CMD_BUFFERS];    \
    const bt_byte       _l2cap_max_cmd_buffers = L2CAP_MAX_CMD_BUFFERS; \
    \
    bt_l2cap_psm_t      _l2cap_psms[L2CAP_MAX_PSMS * L2CAP_MAX_MANAGERS];    \
    const bt_byte       _l2cap_max_psms = L2CAP_MAX_PSMS; \
    bt_l2cap_channel_t  _l2cap_channels[(L2CAP_MAX_CHANNELS) * L2CAP_MAX_MANAGERS];    \
    const bt_byte       _l2cap_max_channels = L2CAP_MAX_CHANNELS; \
    \
    bt_uint             _l2cap_hci_connect_packet_type;    \
    bt_byte             _l2cap_hci_page_scan_repetition_mode;    \
    bt_byte             _l2cap_hci_role_switch;    \
    bt_long             _l2cap_idle_hci_connection_timeout;    \
    \
    bt_buffer_header_t  _l2cap_connect_params_headers[(L2CAP_MAX_CHANNELS) * (L2CAP_MAX_MANAGERS)];    \
    bt_l2cap_connect_params_t  _l2cap_connect_params[(L2CAP_MAX_CHANNELS) * (L2CAP_MAX_MANAGERS)];    \
    \
    L2CAP_DECL_ERETR_FUNCTIONS    \
    \
    L2CAP_FIXED_CHANNELS_DECL    \
    \
    L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR    \
```

Description

This is macro L2CAP_ALLOCATE_BUFFERS_VARS.

L2CAP_FIXED_CHANNELS_DECL Macro

File

l2cap_config.h

C

```
#define L2CAP_FIXED_CHANNELS_DECL \
    bt_l2cap_fixed_channel_t _l2cap_fixed_channels_buffer[L2CAP_MAX_FIXED_CHANNELS *
L2CAP_MAX_MANAGERS];    \
    bt_l2cap_fixed_channel_t* _l2cap_fixed_channels = &_l2cap_fixed_channels_buffer[0];    \
    bt_byte _l2cap_max_fixed_channels = L2CAP_MAX_FIXED_CHANNELS;
```

Description

This is macro L2CAP_FIXED_CHANNELS_DECL.

L2CAP_HCI_PACKET_TYPE Macro

File

l2cap_config.h

C

```
#define L2CAP_HCI_PACKET_TYPE \
    HCI_BB_PACKET_TYPE_DM1 | \
    HCI_BB_PACKET_TYPE_DH1 | \
    HCI_BB_PACKET_TYPE_DM3 | \
    HCI_BB_PACKET_TYPE_DH3 | \
    HCI_BB_PACKET_TYPE_DM5 | \
    HCI_BB_PACKET_TYPE_DH5
```

Description

- brief L2CAP_HCI_PACKET_TYPE.
- ingroup btconfig

*

- details Defines a set of packets that link manager is allowed to use when calling bt_l2cap_connect.
- The default value is to enable all packet types.

enable all packet types

L2CAP_HCI_PAGE_SCAN_REPETITION_MODE Macro

File

l2cap_config.h

C

```
#define L2CAP_HCI_PAGE_SCAN_REPETITION_MODE HCI_PAGE_SCAN_REPETITION_MODE_R0
```

Description

brief L2CAP_HCI_PAGE_SCAN_REPETITION_MODE. ingroup btconfig

details Defines a default value of the page scan repetition mode when calling bt_l2cap_connect. Must be set to one of the following values: HCI_PAGE_SCAN_REPETITION_MODE_R0 HCI_PAGE_SCAN_REPETITION_MODE_R1 HCI_PAGE_SCAN_REPETITION_MODE_R2

The default value is HCI_PAGE_SCAN_REPETITION_MODE_R0.

L2CAP_HCI_ROLE_SWITCH Macro

File

l2cap_config.h

C

```
#define L2CAP_HCI_ROLE_SWITCH HCI_ROLE_SWITCH_ALLOW
```

Description

brief L2CAP_HCI_ROLE_SWITCH. ingroup btconfig

details Defines a default value of the role switch parameter when calling bt_l2cap_connect. Must be set to one of the following values: HCI_ROLE_SWITCH_ALLOW HCI_ROLE_SWITCH_DISALLOW The default value is to allow the role switch.

L2CAP_IDLE_CONNECTION_TIMEOUT Macro

File

l2cap_config.h

C

```
#define L2CAP_IDLE_CONNECTION_TIMEOUT 5 /* seconds */
```

Description

seconds

PACK_CONFIG_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define PACK_CONFIG_REQUEST NULL
```

Description

This is macro PACK_CONFIG_REQUEST.

PACK_CONFIG_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define PACK_CONFIG_RESPONSE NULL
```

Description

This is macro PACK_CONFIG_RESPONSE.

PACK_CONN_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define PACK_CONN_REQUEST NULL
```

Description

This is macro PACK_CONN_REQUEST.

PACK_CONN_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define PACK_CONN_RESPONSE NULL
```

Description

This is macro PACK_CONN_RESPONSE.

PACK_DCONN_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define PACK_DCONN_REQUEST NULL
```

Description

This is macro PACK_DCONN_REQUEST.

PACK_DCONN_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define PACK_DCONN_RESPONSE NULL
```

Description

This is macro PACK_DCONN_RESPONSE.

PACK_INFO_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define PACK_INFO_REQUEST NULL
```

Description

This is macro PACK_INFO_REQUEST.

PACK_INFO_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define PACK_INFO_RESPONSE NULL
```

Description

This is macro PACK_INFO_RESPONSE.

PROCESS_CONFIG_REQ Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_CONFIG_REQ _process_unknown_req
```

Description

This is macro PROCESS_CONFIG_REQ.

PROCESS_CONFIG_RES Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_CONFIG_RES _process_unknown_res
```

Description

This is macro PROCESS_CONFIG_RES.

PROCESS_CONN_REQ Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_CONN_REQ _process_unknown_req
```

Description

This is macro PROCESS_CONN_REQ.

PROCESS_CONN_RES Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_CONN_RES _process_unknown_res
```

Description

This is macro PROCESS_CONN_RES.

PROCESS_DCONN_REQ Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_DCONN_REQ _process_unknown_req
```

Description

This is macro PROCESS_DCONN_REQ.

PROCESS_DCONN_RES Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_DCONN_RES _process_unknown_res
```

Description

This is macro PROCESS_DCONN_RES.

PROCESS_INFO_REQ Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_INFO_REQ _process_unknown_req
```

Description

This is macro PROCESS_INFO_REQ.

PROCESS_INFO_RES Macro

File

l2cap_config_handlers.h

C

```
#define PROCESS_INFO_RES _process_unknown_res
```

Description

This is macro PROCESS_INFO_RES.

READ_CONFIG_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define READ_CONFIG_REQUEST NULL
```

Description

This is macro READ_CONFIG_REQUEST.

READ_CONFIG_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define READ_CONFIG_RESPONSE NULL
```

Description

This is macro READ_CONFIG_RESPONSE.

READ_CONN_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define READ_CONN_REQUEST NULL
```

Description

This is macro READ_CONN_REQUEST.

READ_CONN_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define READ_CONN_RESPONSE NULL
```

Description

This is macro READ_CONN_RESPONSE.

READ_DCONN_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define READ_DCONN_REQUEST NULL
```

Description

This is macro READ_DCONN_REQUEST.

READ_DCONN_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define READ_DCONN_RESPONSE NULL
```

Description

This is macro READ_DCONN_RESPONSE.

READ_INFO_REQUEST Macro

File

l2cap_config_handlers.h

C

```
#define READ_INFO_REQUEST NULL
```

Description

This is macro READ_INFO_REQUEST.

READ_INFO_RESPONSE Macro

File

l2cap_config_handlers.h

C

```
#define READ_INFO_RESPONSE NULL
```

Description

This is macro READ_INFO_RESPONSE.

__L2CAP_CONFIG_H Macro

File

l2cap_config.h

C

```
#define __L2CAP_CONFIG_H
```

Description

This is macro __L2CAP_CONFIG_H.

__L2CAP_CONFIG_HANDLERS_H Macro

File

l2cap_config_handlers.h

C

```
#define __L2CAP_CONFIG_HANDLERS_H
```

Description

This is macro __L2CAP_CONFIG_HANDLERS_H.

L2CAP_DECL_ERETR_FUNCTIONS Macro

File

l2cap_config.h

C

```
#define L2CAP_DECL_ERETR_FUNCTIONS \
        bt_l2cap_channel_ext_t*  _l2cap_channels_ext = NULL;    \
        \
        void (*_l2cap_eretr_recv_fp)(bt_l2cap_mgr_p pmgr, bt_l2cap_channel_t* pch, bt_byte* pdata, bt_int
len) = NULL;    \
        bt_bool (*_l2cap_eretr_send_data_fp)(bt_l2cap_channel_t* pch, bt_byte* data, bt_int len,
bt_l2cap_send_data_callback_fp cb, void* cb_param) = NULL;    \
        bt_bool (*_l2cap_eretr_send_smart_data_fp)(bt_l2cap_channel_t* pch, bt_packet_t* packet, bt_int
len, bt_l2cap_send_data_callback_fp cb, void* cb_param) = NULL;    \
```

```
        bt_bool (*_l2cap_eretr_handle_xmit_event_fp)(bt_l2cap_xmit_event_param_t* param) = NULL;    \
        void (*_l2cap_eretr_pack_config_request_fp)(bt_l2cap_channel_t* channel, bt_byte* buffer, bt_int
buffer_len, bt_int* offset) = NULL;
```

Description

This is macro L2CAP_DECL_ERETR_FUNCTIONS.

L2CAP_MAX_FIXED_CHANNELS Macro

File

l2cap_config.h

C

```
#define L2CAP_MAX_FIXED_CHANNELS 0
```

Description

This is macro L2CAP_MAX_FIXED_CHANNELS.

RFCOMM Configuration Macros

RFCOMM_ALLOCATE_BUFFERS_FUNCTION Macro

File

rfcomm_config.h

C

```
#define RFCOMM_ALLOCATE_BUFFERS_FUNCTION \
    void _rfcomm_allocate_buffers(void)    \
    {    \
    }    \
```

Description

This is macro RFCOMM_ALLOCATE_BUFFERS_FUNCTION.

RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

rfcomm_config.h

C

```
#define RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

This is macro RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR.

RFCOMM_ALLOCATE_BUFFERS_VARS Macro

File

rfcomm_config.h

C

```
#define RFCOMM_ALLOCATE_BUFFERS_VARS \
    bt_rfcomm_session_t _rfcomm_sessions[RFCOMM_MAX_SESSIONS];    \
    const bt_byte       _rfcomm_max_sessions = RFCOMM_MAX_SESSIONS;    \
    bt_rfcomm_dlc_t     _rfcomm_dlcs[(RFCOMM_MAX_DLCS) * (RFCOMM_MAX_SESSIONS)];    \
    const bt_byte       _rfcomm_max_dlcs = RFCOMM_MAX_DLCS;    \
    bt_rfcomm_server_channel_t _rfcomm_channels[(RFCOMM_MAX_SERVER_CHANNELS) * (RFCOMM_MAX_SESSIONS)];    \
    const bt_byte       _rfcomm_max_channels = RFCOMM_MAX_SERVER_CHANNELS;    \
    const bt_uint       _rfcomm_pdu_size = (RFCOMM_INFO_LEN) + (RFCOMM_FRAME_HEADER_LEN);    \
    const bt_uint       _rfcomm_info_len = RFCOMM_INFO_LEN;    \
    bt_buffer_header_t  _rfcomm_cmd_buffer_headers[RFCOMM_MAX_CMD_BUFFERS];    \
```

```
bt_rfcomm_command_t _rfcomm_cmd_buffers[RFCOMM_MAX_CMD_BUFFERS];    \
const bt_byte       _rfcomm_max_cmd_buffers = RFCOMM_MAX_CMD_BUFFERS;    \
const bt_byte       _rfcomm_local_credit = RFCOMM_LOCAL_CREDIT;    \
const bt_bool       _rfcomm_enable_multidevice_channels = RFCOMM_ENABLE_MULTIDEVICE_CHANNELS;    \
RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL    \
RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR    \
```

Description

This is macro RFCOMM_ALLOCATE_BUFFERS_VARS.

RFCOMM_BUFFER_SIZE Macro

File

rfcomm_config.h

C

```
#define RFCOMM_BUFFER_SIZE (RFCOMM_INFO_LEN)
```

Description

This is macro RFCOMM_BUFFER_SIZE.

RFCOMM_ENABLE_MULTIDEVICE_CHANNELS Macro

File

rfcomm_config.h

C

```
#define RFCOMM_ENABLE_MULTIDEVICE_CHANNELS BT_TRUE
```

Description

brief Enable multi-device server channels. ingroup rfcomm_config

details Normally each server channel can be used only once. I.e. if device A connected to channel 1, device B cannot connect to channel 1 until device A disconnects. With this option it is possible to make channels accept connections from several devices at the same time. I.e., if RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is TRUE both device A and device B can connect to channel 1 at the same time.

RFCOMM_INFO_LEN Macro

File

rfcomm_config.h

C

```
#define RFCOMM_INFO_LEN
```

Description

brief Maximum size of the data portion of a UIH frame. ingroup rfcomm_config

details This parameter defines the maximum size of the data portion of a UIH frame. If CFC is used the actual length of the data portion will be 1 byte less. This value must be less than or equal to HCI_L2CAP_BUFFER_LEN - RFCOMM_FRAME_HEADER_LEN - L2CAP_HEADER_LEN.

RFCOMM_LOCAL_CREDIT Macro

File

rfcomm_config.h

C

```
#define RFCOMM_LOCAL_CREDIT
```

Description

brief The number of receive buffers. ingroup rfcomm_config

details This parameter defines the number of received UIH frames that can be stored on the local device. The flow control mechanism used in RFCOMM ensures that the remote side of the link always knows how many free buffers left on the local device. When the number of free buffers

reaches 0, the transmitter stops sending data frames until the receiver frees some buffers. The RFCOMM layer does not actually allocate space for buffers. It uses RFCOMM_LOCAL_CREDIT to keep track of free buffers and report them to the remote side. Actual memory allocation is done in SPP layer.

RFCOMM_MAX_CMD_BUFFERS Macro

File

rfcomm_config.h

C

```
#define RFCOMM_MAX_CMD_BUFFERS
```

Description

brief Maximum number of command buffers. ingroup rfcomm_config

details This parameter defines the maximum number of commands that can be sent at the same time. It is usually enough to reserve 2 buffers for each DLC excluding control DLC. Therefore, this value can be defined as rn #define RFCOMM_MAX_CMD_BUFFERS (RFCOMM_MAX_DLCS - 1) * 2

RFCOMM_MAX_DLCS Macro

File

rfcomm_config.h

C

```
#define RFCOMM_MAX_DLCS
```

Description

brief Maximum number of DLCs ingroup rfcomm_config

details This parameter defines the maximum number of DLCs on each session. This value should be at least 2 because each session uses one DLC to convey multiplexer control messages. All other DLCs are used to emulate serial ports.

RFCOMM_MAX_SERVER_CHANNELS Macro

File

rfcomm_config.h

C

```
#define RFCOMM_MAX_SERVER_CHANNELS
```

Description

brief Maximum number of Server channels ingroup rfcomm_config

details This parameter defines the maximum number of server channels exposed by the local device. This value should not exceed RFCOMM_MAX_DLCS - 1.

RFCOMM_MAX_SESSIONS Macro

File

rfcomm_config.h

C

```
#define RFCOMM_MAX_SESSIONS
```

Description

brief Maximum number of remote devices a local device can be connected to ingroup rfcomm_config

details This parameter defines the maximum number of remote devices a local device can have simultaneous connections to. This value should not exceed HCI_MAX_HCI_CONNECTIONS.

__RFCOMM_CONFIG_H Macro

File

rfcomm_config.h

C

```
#define __RFCOMM_CONFIG_H
```

Description

This is macro __RFCOMM_CONFIG_H.

RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL Macro

File

rfcomm_config.h

C

```
#define RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL const bt_byte _rfcomm_local_credit_send_threshold = 1;
```

Description

This is macro RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL.

SDP Configuration Macros

SDP_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

sdp_config.h

C

```
#define SDP_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

- defgroup sdp_config Configuration
- ingroup sdp
- *
- This module describes parameters used to configure SDP.
- *
- dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

* code #include "cdbt/bt/bt_std.h"

// HCI and L2CAP must always be present // SDP is required only if stack is running in dual mode. This is the default mode. // To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol // must be defined: // #define BT_BLE_SINGLE_MODE

// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...

// L2CAP configuration parameters #define L2CAP_MAX_CMD_BUFFERS ... #define L2CAP_MAX_FRAME_BUFFERS ... #define L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...

// SDP configuration parameters #define SDP_MAX_SEARCH_RESULT_LEN ... #define SDP_MAX_ATTRIBUTE_RESULT_LEN ...

#include "cdbt/bt/bt_oem_config.h"

endcode

SDP_ALLOCATE_BUFFERS_VARS Macro

File

sdp_config.h

C

```
#define SDP_ALLOCATE_BUFFERS_VARS \
        bt_buffer_header_t           _sdp_packet_buffer_headers[SDP_MAX_PDU_BUFFERS];    \
        bt_sdp_packet_t              _sdp_packet_buffers[SDP_MAX_PDU_BUFFERS];    \
        const bt_byte                _sdp_max_buffers = SDP_MAX_PDU_BUFFERS; \
        bt_buffer_header_t           _sdp_client_packet_buffer_headers[SDP_MAX_PDU_BUFFERS];    \
        bt_sdp_packet_t              _sdp_client_packet_buffers[SDP_MAX_PDU_BUFFERS];    \
        const bt_byte                _sdp_client_max_buffers = SDP_MAX_PDU_BUFFERS; \
        const bt_uint                _sdp_max_search_result_len = SDP_MAX_SEARCH_RESULT_LEN;    \
        const bt_uint                _sdp_max_attribute_result_len = SDP_MAX_ATTRIBUTE_RESULT_LEN;    \
        bt_sr_handle_t               _sdp_found_sr_lists_buffers[SDP_MAX_TRANSACTIONS *
SDP_MAX_SEARCH_RESULT_LEN];    \
        bt_sdp_found_attr_list_t     _sdp_found_attr_lists_buffers[SDP_MAX_TRANSACTIONS *
SDP_MAX_SEARCH_RESULT_LEN];    \
        bt_byte*                     _sdp_found_attr_list_buffers[SDP_MAX_TRANSACTIONS *
SDP_MAX_SEARCH_RESULT_LEN * SDP_MAX_ATTRIBUTE_RESULT_LEN];    \
        \
        bt_buffer_mgr_t              _sdp_tran_buffer_mgr2;    \
        bt_buffer_header_t           _sdp_tran_buffer_headers2[SDP_MAX_TRANSACTIONS];    \
        bt_sdp_transaction_t         _sdp_tran_buffers2[SDP_MAX_TRANSACTIONS];    \
        \
        bt_buffer_mgr_t              _sdp_service_tran_buffer_mgr;    \
        bt_buffer_header_t           _sdp_service_tran_buffer_headers[SDP_MAX_TRANSACTIONS];    \
        bt_sdp_service_transaction_t _sdp_service_tran_buffers[SDP_MAX_TRANSACTIONS];    \
        \
        bt_bool                  (*_sdp_start_fp)(bt_l2cap_mgr_p l2cap_mgr, const bt_byte* sdp_db, bt_uint
sdp_db_len) = &bt_sdp_start;    \
        SDP_ALLOCATE_BUFFERS_RAM_SIZE_VAR    \
```

Description

This is macro SDP_ALLOCATE_BUFFERS_VARS.

SDP_MAX_ATTRIBUTE_RESULT_LEN Macro

File

sdp_config.h

C

```
#define SDP_MAX_ATTRIBUTE_RESULT_LEN
```

Description

brief Maximum number of attributes to find ingroup sdp_config

details This parameter defines the maximum number of attributes withing a service record the SDP server will return to the client.

SDP_MAX_PDU_BUFFERS Macro

File

sdp_config.h

C

```
#define SDP_MAX_PDU_BUFFERS
```

Description

brief Maximum number of SDP server PDU buffers. ingroup sdp_config

details This parameter defines the maximum number of responses the SDP server can send at the same time.

SDP_MAX_SEARCH_RESULT_LEN Macro

File

sdp_config.h

C

```
#define SDP_MAX_SEARCH_RESULT_LEN
```

Description

brief Maximum number of service records to find. ingroup sdp_config

details This parameter defines the maximum number of service records the SDP server will return to the client.

__SDP_CONFIG_H Macro

File

sdp_config.h

C

```
#define __SDP_CONFIG_H
```

Description

This is macro __SDP_CONFIG_H.

SPP Configuration Macros

SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR Macro

File

spp_config.h

C

```
#define SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR
```

Description

This is macro SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR.

SPP_ALLOCATE_BUFFERS_VARS Macro

File

spp_config.h

C

```
#define SPP_ALLOCATE_BUFFERS_VARS \
    bt_spp_port_t     _spp_ports[SPP_MAX_PORTS];    \
    const bt_byte     _spp_max_ports = SPP_MAX_PORTS;    \
    const bt_byte     _spp_disable_buffering = SPP_DISABLE_BUFFERING;    \
    SPP_DECLARE_FRAME_BUFFERS    \
    SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR    \
```

Description

This is macro SPP_ALLOCATE_BUFFERS_VARS.

SPP_DECLARE_FRAME_BUFFERS Macro

File

spp_config.h

C

```
#define SPP_DECLARE_FRAME_BUFFERS \
    bt_byte*          _spp_frame_buffers = NULL;    \
    bt_int*           _spp_frame_len = NULL;
```

Description

This is macro SPP_DECLARE_FRAME_BUFFERS.

SPP_DISABLE_BUFFERING Macro

File

spp_config.h

C

```
#define SPP_DISABLE_BUFFERING 1
```

Description

This is macro SPP_DISABLE_BUFFERING.

SPP_FRAME_BUFFERS_RAM_SIZE Macro

File

spp_config.h

C

```
#define SPP_FRAME_BUFFERS_RAM_SIZE 0
```

Description

This is macro SPP_FRAME_BUFFERS_RAM_SIZE.

SPP_FRAME_BUFFERS_SIZE Macro

File

spp_config.h

C

```
#define SPP_FRAME_BUFFERS_SIZE
```

Description

This is macro SPP_FRAME_BUFFERS_SIZE.

SPP_MAX_PORTS Macro

File

spp_config.h

C

```
#define SPP_MAX_PORTS
```

Description

brief Maximum number of SPP ports. ingroup spp_config

details This parameter defines the maximum number of SPP port that can be open between the local and remote devices. If RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is FALSE (default) this value should be equal to RFCOMM_MAX_SERVER_CHANNELS. If RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is TRUE this value should be between RFCOMM_MAX_SERVER_CHANNELS and RFCOMM_MAX_SERVER_CHANNELS * RFCOMM_MAX_SESSIONS.

__SPP_CONFIG_H Macro

File

spp_config.h

C

```
#define __SPP_CONFIG_H
```

Description

This is macro __SPP_CONFIG_H.

Building the Library

The library is provided in binary form only, and comes prebuilt when you receive the applications that carry the Bluetooth license.

Library Interface

A2DP Data Types and Constants

	Name	Description
	_bt_a2dp_aac_config_t	This is type bt_a2dp_aac_config_t.
	_bt_a2dp_event_u	brief Parameter to an application callback. ingroup a2dp details This union is used to pass event specific data to the A2DP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	_bt_a2dp_evt_open_and_start_stream_completed_s	brief Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event ingroup a2dp details A pointer to this structure is passed to the A2DP application callback as a valid member of the bt_a2dp_event_t union - bt_a2dp_event_t::open_and_start_stream_completed - when A2DP completed a "open & start stream" request.
	_bt_a2dp_mgr_t	brief A2DP manager. ingroup a2dp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with ::bt_a2dp_get_mgr.
	_bt_a2dp_mpeg_config_t	This is type bt_a2dp_mpeg_config_t.
	_bt_a2dp_sbc_config_t	This is type bt_a2dp_sbc_config_t.
	_bt_a2dp_sbc_packet_info_t	This is type bt_a2dp_sbc_packet_info_t.
	bt_a2dp_aac_config_t	This is type bt_a2dp_aac_config_t.
	bt_a2dp_event_t	This is type bt_a2dp_event_t.
	bt_a2dp_evt_open_and_start_stream_completed_t	brief Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event ingroup a2dp details A pointer to this structure is passed to the A2DP application callback as a valid member of the bt_a2dp_event_t union - bt_a2dp_event_t::open_and_start_stream_completed - when A2DP completed a "open & start stream" request.
	bt_a2dp_find_server_callback_fp	brief Notify the application of the result of searching for a remote A2DP entity (sourse or sink) ingroup a2dp details This function is called by the A2DP layer when searching for an A2DP entity on a remote device has completed. param supported_features Features supported by a remote A2DP entity. param found c TRUE if an A2DP entity has been found on the remote device. c FALSE otherwise. param param pointer to arbitrary data passed to the bt_a2dp_find_source() or bt_a2dp_find_sink function through its c callback_param parameter.
	bt_a2dp_mgr_callback_fp	brief A2DP application callback. ingroup a2dp details In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call that function whenever a new event has been generated. param mgr A2DP manager. param evt A2DP event. The event can be one of the following values: @arg A2DP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg A2DP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg A2DP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local... more

	bt_a2dp_mgr_t	This is type bt_a2dp_mgr_t.
	bt_a2dp_mpeg_config_t	This is type bt_a2dp_mpeg_config_t.
	bt_a2dp_sbc_config_t	This is type bt_a2dp_sbc_config_t.
	bt_a2dp_sbc_packet_info_t	This is type bt_a2dp_sbc_packet_info_t.
	__A2DP_CODEC_AAC_H	This is macro __A2DP_CODEC_AAC_H.
	__A2DP_CODEC_MPEG_H	This is macro __A2DP_CODEC_MPEG_H.
	__A2DP_CODEC_SBC_H	This is macro __A2DP_CODEC_SBC_H.
	__A2DP_H	This is macro __A2DP_H.
	__A2DP_PRIVATE_H	This is macro __A2DP_PRIVATE_H.
	A2DP_EVT_ABORT_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.
	A2DP_EVT_ABORT_STREAM_REQUESTED	< This event is generated when a local device received "abort stream" request.
	A2DP_EVT_CLOSE_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "close stream" request.
	A2DP_EVT_CLOSE_STREAM_REQUESTED	< This event is generated when a local device received "close stream" request.
	A2DP_EVT_CTRL_CHANNEL_CONNECTED	< This event is generated when a control channel between two AVDTP entities has been established.
	A2DP_EVT_CTRL_CHANNEL_DISCONNECTED	< This event is generated when a control channel between two AVDTP entities has been terminated.
	A2DP_EVT_CTRL_CONNECTION_FAILED	< This event is generated when a local device failed to create a control channel between two AVDTP entities.
	A2DP_EVT_DISCOVER_SEP_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "discover" request.
	A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.
	A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.
	A2DP_EVT_MEDIA_PACKET_RECEIVED	< This event is generated when a local device received a media packet.
	A2DP_EVT_MEDIA_PACKET_SEND_FAILED	< This event is generated when a local device failed to send a media packet.
	A2DP_EVT_MEDIA_PACKET_SENT	< This event is generated when a local device sent a media packet.
	A2DP_EVT_NOTHING	This is macro A2DP_EVT_NOTHING.
	A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED	< This event is generated when a local device completed "open and start" request.
	A2DP_EVT_OPEN_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "open stream" request.
	A2DP_EVT_OPEN_STREAM_REQUESTED	< This event is generated when a local device received "open stream" request.
	A2DP_EVT_RECONFIGURE_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.
	A2DP_EVT_RECONFIGURE_STREAM_REQUESTED	< This event is generated when a local device received "change stream configuration" request.
	A2DP_EVT_SEP_CAPABILITIES_RECEIVED	< This event is generated when a local device received a positive response to a "get SEP capabilities" request.
	A2DP_EVT_SEP_INFO_RECEIVED	< This event is generated for each SEP contained in a positive response to a "discover" request.
	A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.
	A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED	< This event is generated when a local device received "set stream configuration" request.
	A2DP_EVT_START_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "start stream" request.
	A2DP_EVT_START_STREAM_REQUESTED	< This event is generated when a local device received "start stream" request.

	A2DP_EVT_STREAM_ABORTED	< This event is generated when a local device has successfully aborted a stream. < This event follows the A2DP_EVT_ABORT_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream abortion was initiated by the local device.
	A2DP_EVT_STREAM_CLOSED	< This event is generated when a local device has successfully closed a stream. < This event follows the A2DP_EVT_CLOSE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream closing was initiated by the local device.
	A2DP_EVT_STREAM_CONFIGURATION_RECEIVED	< This event is generated when a local device received a positive response to a "get stream configuration" request.
	A2DP_EVT_STREAM_CONFIGURED	< This event is generated when a local device has successfully configured a stream. < This event follows the A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the upper layer has accepted it. < This event is not generated if stream configuration was initiated by the local device.
	A2DP_EVT_STREAM_OPENED	< This event is generated when a local device has successfully opened a stream. < This event follows the A2DP_EVT_OPEN_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream opening was initiated by the local device.
	A2DP_EVT_STREAM_RECONFIGURED	< This event is generated when a local device has successfully reconfigured a stream. < This event follows the A2DP_EVT_RECONFIGURE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream reconfiguration was initiated by the local device.
	A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.
	A2DP_EVT_STREAM_STARTED	< This event is generated when a local device has successfully started a stream. < This event follows the A2DP_EVT_START_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream starting was initiated by the local device.
	A2DP_EVT_STREAM_SUSPENDED	< This event is generated when a local device has successfully suspended a stream. < This event follows the A2DP_EVT_SUSPEND_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream suspension was initiated by the local device.
	A2DP_EVT_SUSPEND_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.
	A2DP_EVT_SUSPEND_STREAM_REQUESTED	< This event is generated when a local device received "suspend stream" request.
	A2DP_MANAGER_STATE_CONNECTING	This is macro A2DP_MANAGER_STATE_CONNECTING.
	A2DP_MANAGER_STATE_IDLE	This is macro A2DP_MANAGER_STATE_IDLE.
	A2DP_SINK_FEATURE_AMPLIFIER	< Amplifier
	A2DP_SINK_FEATURE_HEADPHONE	< Headphone
	A2DP_SINK_FEATURE_RECORDER	< Recorder
	A2DP_SINK_FEATURE_SPEAKER	< Speaker
	A2DP_SOURCE_FEATURE_MICROPHONE	< Mic
	A2DP_SOURCE_FEATURE_MIXER	< Mixer
	A2DP_SOURCE_FEATURE_PLAYER	< Player
	A2DP_SOURCE_FEATURE_TUNER	< Tuner
	AAC_CHANNELS_1	This is macro AAC_CHANNELS_1.
	AAC_CHANNELS_2	This is macro AAC_CHANNELS_2.
	AAC_CHANNELS_ALL	This is macro AAC_CHANNELS_ALL.
	AAC_OBJECT_TYPE_MPEG_2_LC	This is macro AAC_OBJECT_TYPE_MPEG_2_LC.
	AAC_OBJECT_TYPE_MPEG_4_LC	This is macro AAC_OBJECT_TYPE_MPEG_4_LC.
	AAC_OBJECT_TYPE_MPEG_4_LTP	This is macro AAC_OBJECT_TYPE_MPEG_4_LTP.
	AAC_OBJECT_TYPE_MPEG_4_SCALABLE	This is macro AAC_OBJECT_TYPE_MPEG_4_SCALABLE.
	AAC_SAMPLING_FREQUENCY_11025	This is macro AAC_SAMPLING_FREQUENCY_11025.
	AAC_SAMPLING_FREQUENCY_12000	This is macro AAC_SAMPLING_FREQUENCY_12000.

	AAC_SAMPLING_FREQUENCY_16000	This is macro AAC_SAMPLING_FREQUENCY_16000.
	AAC_SAMPLING_FREQUENCY_22050	This is macro AAC_SAMPLING_FREQUENCY_22050.
	AAC_SAMPLING_FREQUENCY_24000	This is macro AAC_SAMPLING_FREQUENCY_24000.
	AAC_SAMPLING_FREQUENCY_32000	This is macro AAC_SAMPLING_FREQUENCY_32000.
	AAC_SAMPLING_FREQUENCY_44100	This is macro AAC_SAMPLING_FREQUENCY_44100.
	AAC_SAMPLING_FREQUENCY_48000	This is macro AAC_SAMPLING_FREQUENCY_48000.
	AAC_SAMPLING_FREQUENCY_64000	This is macro AAC_SAMPLING_FREQUENCY_64000.
	AAC_SAMPLING_FREQUENCY_8000	This is macro AAC_SAMPLING_FREQUENCY_8000.
	AAC_SAMPLING_FREQUENCY_88200	This is macro AAC_SAMPLING_FREQUENCY_88200.
	AAC_SAMPLING_FREQUENCY_96000	This is macro AAC_SAMPLING_FREQUENCY_96000.
	AAC_SAMPLING_FREQUENCY_ALL	This is macro AAC_SAMPLING_FREQUENCY_ALL.
	AAC_VBR_NOT_SUPPORTED	This is macro AAC_VBR_NOT_SUPPORTED.
	AAC_VBR_SUPPORTED	This is macro AAC_VBR_SUPPORTED.
	bt_a2dp_abort_stream	brief Suspend a stream. ingroup a2dp details This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE. As a result of this operation the A2DP_EVT_ABORT_STREAM_COMPLETED event will be generated. This operation cannot be rejected. The p evt_param.abort_stream_requested.err_code is always == AVDTP_ERROR_SUCCESS. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.
	bt_a2dp_add_media_rx_buffer	brief Add a media packet buffer to a receive queue ingroup a2dp details The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more
	bt_a2dp_add_media_tx_buffer	brief Add a media packet buffer to a send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. When the packet has been successfully sent a A2DP_EVT_MEDIA_PACKET_SENT is generated. Otherwise a A2DP_EVT_MEDIA_PACKET_SEND_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as A2DP has removed it from the queue and gave up... more
	bt_a2dp_call_codec	This is macro bt_a2dp_call_codec.
	bt_a2dp_cancel_listen	brief Cancel listening for incoming connections. ingroup a2dp details This function removes a SEP from a list of SEPS which a stream can use for incoming requests. param mgr A2DP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if the function succeeds. li c FALSE otherwise.

	bt_a2dp_close_stream	brief Close a stream. ingroup a2dp details This function tries to close a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN or AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the A2DP_EVT_CLOSE_STREAM_COMPLETED event will be generated. If the stream has been closed the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == the error code sent by the remote. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds,... more
	bt_a2dp_connect	brief Connect to a remote device. ingroup a2dp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP_EVT_CTRL_CHANNEL_CONNECTED or A2DP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr A2DP manager. param remote_addr The address of a remote device. return li c TRUE if connection establishment has been started. li c FALSE... more
	bt_a2dp_connect_ex	brief Connect to a remote device. ingroup a2dp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP_EVT_CTRL_CHANNEL_CONNECTED or A2DP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr A2DP manager. param remote_addr The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following... more
	bt_a2dp_create_stream	brief Create a stream. ingroup a2dp details This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort. param mgr A2DP manager. return li c Stream handle if the function succeeds. li c 0 otherwise.
	bt_a2dp_destroy_stream	brief Destroy a stream. ingroup a2dp details This function frees memory used by the stream. The stream has to exist and be in the "idle" state for this function to succeed. I.e. the stream has to be closed or aborted before this function can be called. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_a2dp_disconnect	brief Disconnect from a remote device. ingroup a2dp details This function closes a control and transport channels on all streams associated with the remote device specified by the p remote_addr. As a result of this operation the following events will be generated: @arg A2DP_EVT_MEDIA_PACKET_RECEIVED: if a stream's receive queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg A2DP_EVT_MEDIA_PACKET_SENT: if a stream's send queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg A2DP_EVT_STREAM_CLOSED: this event is generate if a stream is in... more

	bt_a2dp_discover	brief Discover SEPs on a remote device. ingroup a2dp details This function asks the remote device to send a list of all available SEPs. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_INFO_RECEIVED: this event is generated for every SEP received from the remote device. the p evt_param.sep_info_received contains SEP information. @arg A2DP_EVT_DISCOVER_COMPLETED: this event is generated after the last A2DP_EVT_SEP_INFO_RECEIVED if the remote accepted the request and the p evt_param.discover_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.discover_completed.err_code == the error code sent by the remote. param mgr A2DP... more
	bt_a2dp_find_codec	brief Find a codec ingroup a2dp details A2DP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make our implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of A2DP has to register a callback function (one per codec type) for each codec it wishes to support. That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in... more
	bt_a2dp_get_capabilities	brief Get remote SEP capabilities. ingroup a2dp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after A2DP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr A2DP manager.... more
	bt_a2dp_get_hci_connection	brief Get HCI connection for a stream ingroup a2dp details This function returns a pointer to a structure that describes an HCI connection a stream is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect. param mgr A2DP manager. param strm_handle Stream handle. return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a stream specified by the p... more
	bt_a2dp_get_stream_codec_config	brief Get the configuration of the codec currently used with the stream. ingroup a2dp details This function returns a pointer to a structure that contains configuration of the codec currently used with the stream. The structure returned depends on the codec. The dotstack defines structures only for SBC, MPEG-1,2 and MPEG-2,4 AAC codecs: @arg SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h) @arg MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h) @arg MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h) param mgr A2DP manager. param strm_handle Stream handle. return li The codec's configuration if strm_handle specifies a valid stream and the stream is in one of the following... more

	bt_a2dp_get_stream_codec_type	brief Get the type of the codec currently used with the stream. ingroup a2dp details This function returns the type of the codec currently used with the stream. param mgr A2DP manager. param strm_handle Stream handle. return @arg The type of the codec if strm_handle specifies a valid stream and the stream is in one of the following states: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING @arg The result will be one of the following values: AVDTP_CODEC_TYPE_SBC: SBC AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP... more
	bt_a2dp_get_stream_local_sep_id	brief Get stream's local SEP ID. ingroup a2dp details This function returns the ID of the local SEP associated with the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The ID of the local SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_a2dp_get_stream_remote_address	brief Get stream's remote BT address. ingroup a2dp details This function returns the address of the remote device associated with the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The address of the remote device if strm_handle specifies a valid stream. li NULL otherwise.
	bt_a2dp_get_stream_remote_sep_id	brief Get stream's remote SEP ID. ingroup a2dp details This function returns the ID of the remote SEP associated with the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The ID of the remote SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_a2dp_get_stream_state	brief Get local stream state. ingroup a2dp details This function returns local state of a stream specified by the p strm_handle. No request is sent to the remote party. param mgr A2DP manager. param strm_handle Stream handle. return The state of the stream. The result will be one of the following values: @arg AVDTP_STREAM_STATE_IDLE: The stream is idle. This can mean two things. The stream specified by p strm_handle does not exist or the stream is closed. @arg AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS: The stream is opening transport channels. @arg AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS: The stream is closing transport channels. @arg AVDTP_STREAM_STATE_CONFIGURED: The... more
	bt_a2dp_listen	brief Listen for incoming connections. ingroup a2dp details This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream. param mgr A2DP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if... more

	bt_a2dp_reconfigure_stream	brief Reconfigure stream. ingroup a2dp details This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the A2DP_EVT_STREAM_RECONFIGURE_COMPLETED event will be generated. If reconfiguration was a success the p evt_param.stream_reconfigure_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.stream_reconfigure_completed.err_code == the error code sent by the remote. param mgr A2DP manager. param strm_handle Stream handle. param caps New stream configuration. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be... more
	bt_a2dp_register_sink	brief Register a Sink SEP with the local A2DP manager. ingroup a2dp details This function is used to add a sink SEP to a list of SEPs supported by the local A2DP entity. param mgr A2DP manager. param caps The capabilities of a SEP. return li c ID of a SEP if the function succeeds. li c FALSE otherwise.
	bt_a2dp_register_source	brief Register a Source SEP with the local A2DP manager. ingroup a2dp details This function is used to add a source SEP to a list of SEPs supported by the local A2DP entity. param mgr A2DP manager. param caps The capabilities of a SEP. return li c ID of a SEP if the function succeeds. li c FALSE otherwise.
	bt_a2dp_remove_media_rx_buffer	brief Remove a media packet buffer from a receive queue ingroup a2dp details The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more
	bt_a2dp_remove_media_tx_buffer	brief Remove a media packet buffer from a send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling ::bt_a2dp_remove_media_tx_buffer. param mgr A2DP manager. param strm_handle Stream handle. param buffer Pointer to a structure... more
	bt_a2dp_start_stream	brief Start a stream. ingroup a2dp details This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the A2DP_EVT_START_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.start_stream_requested.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the p evt_param.start_stream_requested.err_code == the error code sent... more

	bt_a2dp_suspend_stream	brief Suspend a stream. ingroup a2dp details This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the A2DP_EVT_SUSPEND_STREAM_COMPLETED event will be generated. If the stream has been suspended the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == the error code sent by the remote. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the... more
	MPEG_BITRATE_0000	This is macro MPEG_BITRATE_0000.
	MPEG_BITRATE_0001	This is macro MPEG_BITRATE_0001.
	MPEG_BITRATE_0010	This is macro MPEG_BITRATE_0010.
	MPEG_BITRATE_0011	This is macro MPEG_BITRATE_0011.
	MPEG_BITRATE_0100	This is macro MPEG_BITRATE_0100.
	MPEG_BITRATE_0101	This is macro MPEG_BITRATE_0101.
	MPEG_BITRATE_0110	This is macro MPEG_BITRATE_0110.
	MPEG_BITRATE_0111	This is macro MPEG_BITRATE_0111.
	MPEG_BITRATE_1000	This is macro MPEG_BITRATE_1000.
	MPEG_BITRATE_1001	This is macro MPEG_BITRATE_1001.
	MPEG_BITRATE_1010	This is macro MPEG_BITRATE_1010.
	MPEG_BITRATE_1011	This is macro MPEG_BITRATE_1011.
	MPEG_BITRATE_1100	This is macro MPEG_BITRATE_1100.
	MPEG_BITRATE_1101	This is macro MPEG_BITRATE_1101.
	MPEG_BITRATE_1110	This is macro MPEG_BITRATE_1110.
	MPEG_BITRATE_ALL	This is macro MPEG_BITRATE_ALL.
	MPEG_CHANNEL_MODE_ALL	This is macro MPEG_CHANNEL_MODE_ALL.
	MPEG_CHANNEL_MODE_DUAL_CHANNEL	This is macro MPEG_CHANNEL_MODE_DUAL_CHANNEL.
	MPEG_CHANNEL_MODE_JOINT_STEREO	This is macro MPEG_CHANNEL_MODE_JOINT_STEREO.
	MPEG_CHANNEL_MODE_MONO	This is macro MPEG_CHANNEL_MODE_MONO.
	MPEG_CHANNEL_MODE_STEREO	This is macro MPEG_CHANNEL_MODE_STEREO.
	MPEG_CRC_PROTECTION_NOT_SUPPORTED	This is macro MPEG_CRC_PROTECTION_NOT_SUPPORTED.
	MPEG_CRC_PROTECTION_SUPPORTED	This is macro MPEG_CRC_PROTECTION_SUPPORTED.
	MPEG_LAYER_1	This is macro MPEG_LAYER_1.
	MPEG_LAYER_2	This is macro MPEG_LAYER_2.
	MPEG_LAYER_3	This is macro MPEG_LAYER_3.
	MPEG_LAYER_ALL	This is macro MPEG_LAYER_ALL.
	MPEG_MPF_1	This is macro MPEG_MPF_1.
	MPEG_MPF_2	This is macro MPEG_MPF_2.
	MPEG_SAMPLING_FREQUENCY_16000	This is macro MPEG_SAMPLING_FREQUENCY_16000.
	MPEG_SAMPLING_FREQUENCY_22050	This is macro MPEG_SAMPLING_FREQUENCY_22050.
	MPEG_SAMPLING_FREQUENCY_24000	This is macro MPEG_SAMPLING_FREQUENCY_24000.
	MPEG_SAMPLING_FREQUENCY_32000	This is macro MPEG_SAMPLING_FREQUENCY_32000.
	MPEG_SAMPLING_FREQUENCY_44100	This is macro MPEG_SAMPLING_FREQUENCY_44100.
	MPEG_SAMPLING_FREQUENCY_48000	This is macro MPEG_SAMPLING_FREQUENCY_48000.
	MPEG_SAMPLING_FREQUENCY_ALL	This is macro MPEG_SAMPLING_FREQUENCY_ALL.
	MPEG_VBR_NOT_SUPPORTED	This is macro MPEG_VBR_NOT_SUPPORTED.
	MPEG_VBR_SUPPORTED	This is macro MPEG_VBR_SUPPORTED.
	SBC_ALLOCATION_METHOD_ALL	This is macro SBC_ALLOCATION_METHOD_ALL.
	SBC_ALLOCATION_METHOD_LOUDNESS	This is macro SBC_ALLOCATION_METHOD_LOUDNESS.
	SBC_ALLOCATION_METHOD_SNR	This is macro SBC_ALLOCATION_METHOD_SNR.
	SBC_BLOCK_LENGTH_12	This is macro SBC_BLOCK_LENGTH_12.
	SBC_BLOCK_LENGTH_16	This is macro SBC_BLOCK_LENGTH_16.
	SBC_BLOCK_LENGTH_4	This is macro SBC_BLOCK_LENGTH_4.

	SBC_BLOCK_LENGTH_8	This is macro SBC_BLOCK_LENGTH_8.
	SBC_BLOCK_LENGTH_ALL	This is macro SBC_BLOCK_LENGTH_ALL.
	SBC_CHANNEL_MODE_ALL	This is macro SBC_CHANNEL_MODE_ALL.
	SBC_CHANNEL_MODE_DUAL_CHANNEL	This is macro SBC_CHANNEL_MODE_DUAL_CHANNEL.
	SBC_CHANNEL_MODE_JOINT_STEREO	This is macro SBC_CHANNEL_MODE_JOINT_STEREO.
	SBC_CHANNEL_MODE_MONO	This is macro SBC_CHANNEL_MODE_MONO.
	SBC_CHANNEL_MODE_STEREO	This is macro SBC_CHANNEL_MODE_STEREO.
	SBC_SAMPLING_FREQUENCY_16000	This is macro SBC_SAMPLING_FREQUENCY_16000.
	SBC_SAMPLING_FREQUENCY_32000	This is macro SBC_SAMPLING_FREQUENCY_32000.
	SBC_SAMPLING_FREQUENCY_44100	This is macro SBC_SAMPLING_FREQUENCY_44100.
	SBC_SAMPLING_FREQUENCY_48000	This is macro SBC_SAMPLING_FREQUENCY_48000.
	SBC_SAMPLING_FREQUENCY_ALL	This is macro SBC_SAMPLING_FREQUENCY_ALL.
	SBC_SUBBANDS_4	This is macro SBC_SUBBANDS_4.
	SBC_SUBBANDS_8	This is macro SBC_SUBBANDS_8.
	SBC_SUBBANDS_ALL	This is macro SBC_SUBBANDS_ALL.
	bt_a2dp_get_all_capabilities	brief Get remote SEP capabilities. ingroup a2dp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after A2DP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr A2DP manager.... more
	bt_a2dp_get_configuration	This is macro bt_a2dp_get_configuration.
	bt_a2dp_open_stream	This is macro bt_a2dp_open_stream.
	bt_a2dp_clear_media_tx_queue	brief Clear send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. The consumer can remove all packets from the queue before they have been sent to a remote device by calling ::bt_a2dp_clear_media_tx_queue. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails... more
	bt_a2dp_get_stream_config	brief Get stream's configuration. ingroup a2dp details This function returns a pointer to a structure holding current configuration of the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The stream's configuration if strm_handle specifies a valid stream and the stream is in one of the following state: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING li NULL otherwise.
	bt_a2dp_get_stream_direction	This is macro bt_a2dp_get_stream_direction.
	bt_a2dp_report_delay	This is macro bt_a2dp_report_delay.

	bt_a2dp_set_media_tx_queue_limit	brief Set limit on the send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. By default the send queue can contain unlimited number of packets. The consumer can set a limit on how many packets are held in the queue. In this case when new packet is added to the queue and the length of... more
	A2DP_EVT_SET_STREAM_CONFIGURATION	This is macro A2DP_EVT_SET_STREAM_CONFIGURATION.

A2DP Functions

	Name	Description
	_bt_a2dp_avdtp_mgr_callback	This is function _bt_a2dp_avdtp_mgr_callback.
	bt_a2dp_aac_codec_handler	This is function bt_a2dp_aac_codec_handler.
	bt_a2dp_find_server	This is function bt_a2dp_find_server.
	bt_a2dp_find_sink	brief Find sink ingroup a2dp details This function looks for a sink on a remote device specified by c deviceAddress and, if found, returns features supported by the sink. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer... more
	bt_a2dp_find_source	brief Find source ingroup a2dp details This function looks for a source on a remote device specified by c deviceAddress and, if found, returns features supported by the source. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer... more
	bt_a2dp_get_mgr	brief Return a pointer to an instance of the A2DP manager. ingroup a2dp details This function returns a pointer to an instance of the A@DP manager. There is only one instance of the manager allocated by the stack. The pointer is passed as the first parameter to all A2DP functions.
	bt_a2dp_init	brief Initialize the A2DP layer. ingroup a2dp details This function initializes the A2DP layer of the stack. It must be called prior to any other A2DP function can be called.
	bt_a2dp_mpeg_codec_handler	This is function bt_a2dp_mpeg_codec_handler.
	bt_a2dp_open_and_start_stream	brief Open & start a stream ingroup a2dp details Opening a stream involves sending 3 requests to a remote device - "set configuration", "open stream" and "start stream". Each event generates its own event which must be handled and acted accordingly by the application. To make the use of API easier dotstack combines all these requests in one request called "open & start stream". dotstack sends necessary requests in a proper sequence, handles responses and generates only one event (A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED) at the end. If any of the individual requests has failed the event's parameter bt_a2dp_event_t::open_and_start_stream_completed is populated with... more
	bt_a2dp_register_aac_codec	brief Register default AAC codec ingroup a2dp details This function adds AAC codec implemented by dotstack to the list of known codecs. For more information about codecs see description of ::bt_avdtp_register_codec. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use AAC codec it must call this function when it is initializing. note dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration. param mgr A2DP manager.

	bt_a2dp_register_callback	brief Register a A2DP application callback. ingroup a2dp details In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values: @arg A2DP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg A2DP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg A2DP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs.... more
	bt_a2dp_register_mpeg_codec	brief Register default MPEG codec ingroup a2dp details This function adds MPEG codec implemented by dotstack to the list of known codecs. For more information about codecs see description of ::bt_avdtp_register_codec. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use MPEG-1,2 codec it must call this function when it is initializing. note dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration. param mgr A2DP manager.
	bt_a2dp_sbc_codec_handler	This is function bt_a2dp_sbc_codec_handler.
	bt_a2dp_start	brief Start the A2DP layer. ingroup a2dp details This function makes the A2DP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required. param mgr AVDTP manager. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_a2dp_set_configuration	This is function bt_a2dp_set_configuration.

AVCTP Data Types and Constants

	Name	Description
	_bt_avctp_channel_t	brief AVCTP channel description ingroup avctp details This structure is used to hold information about an AVCTP channel.
	_bt_avctp_event_u	brief Parameter to an application callback. ingroup avctp details This union is used to pass event specific data to the AVCTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	_bt_avctp_evt_channel_connected_t	brief Parameter to AVCTP_EVT_CHANNEL_CONNECTED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_connected - when a channel between two devices has been established.
	_bt_avctp_evt_channel_disconnected_t	brief Parameter to AVCTP_EVT_CHANNEL_DISCONNECTED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_disconnected - when a channel between two devices has been terminated.
	_bt_avctp_evt_command_cancelled_t	brief Parameter to AVCTP_EVT_COMMAND_CANCELLED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_cancelled - when sending a command message has been canceled.
	_bt_avctp_evt_command_received_t	brief Parameter to AVCTP_EVT_COMMAND_RECEIVED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_received - when a local device received a command message.
	_bt_avctp_evt_command_sent_t	brief Parameter to AVCTP_EVT_COMMAND_SENT event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_sent - when a local device finished sending a command message.

	_bt_avctp_evt_connection_failed_t	brief Parameter to AVCTP_EVT_CONNECTION_FAILED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::connection_failed - when a channel between two devices could not be established.
	_bt_avctp_evt_response_cancelled_t	brief Parameter to AVCTP_EVT_RESPONSE_CANCELLED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_cancelled - when sending a response message has been canceled.
	_bt_avctp_evt_response_received_t	brief Parameter to AVCTP_EVT_RESPONSE_RECEIVED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_received - when a local device received a response message.
	_bt_avctp_evt_response_sent_t	brief Parameter to AVCTP_EVT_RESPONSE_SENT event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_sent - when a local device finished sending a response message.
	_bt_avctp_message_t	brief AVCTP message description ingroup avctp details This structure is used to hold information about an AVCTP message.
	_bt_avctp_mgr_t	brief AVCTP manager. ingroup avctp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avctp_get_mgr().
	_bt_avctp_packet_t	This is type bt_avctp_packet_t.
	_bt_avctp_transport_t	brief AVCTP transport description ingroup avctp details This structure is used to hold information about an AVCTP transport.
	bt_avctp_channel_t	This is type bt_avctp_channel_t.
	bt_avctp_event_t	brief Parameter to an application callback. ingroup avctp details This union is used to pass event specific data to the AVCTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	bt_avctp_evt_channel_connected_t	brief Parameter to AVCTP_EVT_CHANNEL_CONNECTED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_connected - when a channel between two devices has been established.
	bt_avctp_evt_channel_disconnected_t	brief Parameter to AVCTP_EVT_CHANNEL_DISCONNECTED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_disconnected - when a channel between two devices has been terminated.
	bt_avctp_evt_command_cancelled_t	brief Parameter to AVCTP_EVT_COMMAND_CANCELLED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_cancelled - when sending a command message has been canceled.
	bt_avctp_evt_command_received_t	brief Parameter to AVCTP_EVT_COMMAND_RECEIVED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_received - when a local device received a command message.

	bt_avctp_evt_command_sent_t	brief Parameter to AVCTP_EVT_COMMAND_SENT event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_sent - when a local device finished sending a command message.
	bt_avctp_evt_connection_failed_t	brief Parameter to AVCTP_EVT_CONNECTION_FAILED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::connection_failed - when a channel between two devices could not be established.
	bt_avctp_evt_response_cancelled_t	brief Parameter to AVCTP_EVT_RESPONSE_CANCELLED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_cancelled - when sending a response message has been canceled.
	bt_avctp_evt_response_received_t	brief Parameter to AVCTP_EVT_RESPONSE_RECEIVED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_received - when a local device received a response message.
	bt_avctp_evt_response_sent_t	brief Parameter to AVCTP_EVT_RESPONSE_SENT event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_sent - when a local device finished sending a response message.
	bt_avctp_message_t	This is type bt_avctp_message_t.
	bt_avctp_mgr_callback_fp	brief AVCTP application callback. ingroup avctp details In order to be notified of various events a consumer of the AVCTP layer has to register a callback function (done with bt_avctp_set_callback()). The stack will call that function whenever a new event has been generated. param mgr AVCTP manager. param evt AVCTP event. The event can be one of the following values: @arg AVCTP_EVT_CHANNEL_CONNECTED Channel connected. @arg AVCTP_EVT_CHANNEL_DISCONNECTED Channel disconnected. @arg AVCTP_EVT_CONNECTION_FAILED Channel connection failed (generated only if connection has been initiated by the local device). @arg AVCTP_EVT_COMMAND_RECEIVED Command received. @arg AVCTP_EVT_RESPONE_RECEIVED Response received. @arg AVCTP_EVT_COMMAND_SENT Command sent. @arg AVCTP_EVT_RESPONSE_SENT Response... more
	bt_avctp_mgr_t	This is type bt_avctp_mgr_t.
	bt_avctp_packet_t	This is type bt_avctp_packet_t.
	bt_avctp_transport_t	This is type bt_avctp_transport_t.
	__AVCTP_H	This is macro __AVCTP_H.
	__AVCTP_PACKET_H	This is macro __AVCTP_PACKET_H.
	__AVCTP_PRIVATE_H	This is macro __AVCTP_PRIVATE_H.
	AVCTP_CHANNEL_FLAG_LISTENING	This is macro AVCTP_CHANNEL_FLAG_LISTENING.
	AVCTP_CHANNEL_FLAG_SENDING	This is macro AVCTP_CHANNEL_FLAG_SENDING.
	AVCTP_CHANNEL_STATE_CONNECTED	This is macro AVCTP_CHANNEL_STATE_CONNECTED.
	AVCTP_CHANNEL_STATE_CONNECTING	This is macro AVCTP_CHANNEL_STATE_CONNECTING.
	AVCTP_CHANNEL_STATE_DISCONNECTING	This is macro AVCTP_CHANNEL_STATE_DISCONNECTING.
	AVCTP_CHANNEL_STATE_FREE	This is macro AVCTP_CHANNEL_STATE_FREE.
	AVCTP_CHANNEL_STATE_IDLE	This is macro AVCTP_CHANNEL_STATE_IDLE.
	AVCTP_ERROR_BAD_STATE	This is macro AVCTP_ERROR_BAD_STATE.
	AVCTP_ERROR_SUCCESS	This is macro AVCTP_ERROR_SUCCESS.
	AVCTP_EVT_CHANNEL_CONNECTED	< This event is generated when a channel between two AVCTP entities has been established.

	AVCTP_EVT_CHANNEL_DISCONNECTED	< This event is generated when a channel between two AVCTP entities has been terminated.
	AVCTP_EVT_COMMAND_CANCELLED	< This event is generated when a command has been canceled.
	AVCTP_EVT_COMMAND_RECEIVED	< This event is generated when a local device received a command.
	AVCTP_EVT_COMMAND_SENT	< This event is generated when a local device finished sending a command.
	AVCTP_EVT_CONNECTION_FAILED	< This event is generated when a local device failed to create a channel between two AVCTP entities.
	AVCTP_EVT_NOTHING	addtogroup avrcp @{ @name Events details The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.
	AVCTP_EVT_RESPONE_RECEIVED	< This event is generated when a local device received a response.
	AVCTP_EVT_RESPONSE_CANCELLED	< This event is generated when a response has been canceled.
	AVCTP_EVT_RESPONSE_SENT	< This event is generated when a local device finished sending a response.
	AVCTP_MANAGER_STATE_IDLE	This is macro AVCTP_MANAGER_STATE_IDLE.
	AVCTP_MESSAGE_PACKET_TYPE_CONTINUE	This is macro AVCTP_MESSAGE_PACKET_TYPE_CONTINUE.
	AVCTP_MESSAGE_PACKET_TYPE_END	This is macro AVCTP_MESSAGE_PACKET_TYPE_END.
	AVCTP_MESSAGE_PACKET_TYPE_SINGLE	This is macro AVCTP_MESSAGE_PACKET_TYPE_SINGLE.
	AVCTP_MESSAGE_PACKET_TYPE_START	This is macro AVCTP_MESSAGE_PACKET_TYPE_START.
	AVCTP_MESSAGE_TYPE_COMMAND	This is macro AVCTP_MESSAGE_TYPE_COMMAND.
	AVCTP_MESSAGE_TYPE_RESPONSE	This is macro AVCTP_MESSAGE_TYPE_RESPONSE.
	AVCTP_TRANSPORT_FLAG_RX_MESSAGE_STARTED	This is macro AVCTP_TRANSPORT_FLAG_RX_MESSAGE_STARTED.
	AVCTP_TRANSPORT_FLAG_SENDING	This is macro AVCTP_TRANSPORT_FLAG_SENDING.
	AVCTP_TRANSPORT_STATE_CONNECTED	This is macro AVCTP_TRANSPORT_STATE_CONNECTED.
	AVCTP_TRANSPORT_STATE_CONNECTING	This is macro AVCTP_TRANSPORT_STATE_CONNECTING.
	AVCTP_TRANSPORT_STATE_DISCONNECTING	This is macro AVCTP_TRANSPORT_STATE_DISCONNECTING.
	AVCTP_TRANSPORT_STATE_FREE	This is macro AVCTP_TRANSPORT_STATE_FREE.
	AVCTP_TRANSPORT_STATE_IDLE	This is macro AVCTP_TRANSPORT_STATE_IDLE.
	AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET	This is macro AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET.

AVCTP Functions

	Name	Description
	_bt_avctp_allocate_channel	This is function _bt_avctp_allocate_channel.
	_bt_avctp_allocate_message	This is function _bt_avctp_allocate_message.
	_bt_avctp_allocate_transport	This is function _bt_avctp_allocate_transport.
	_bt_avctp_find_channel	This is function _bt_avctp_find_channel.
	_bt_avctp_find_transport	This is function _bt_avctp_find_transport.
	_bt_avctp_free_channel	This is function _bt_avctp_free_channel.
	_bt_avctp_free_message	This is function _bt_avctp_free_message.
	_bt_avctp_free_transport	This is function _bt_avctp_free_transport.
	_bt_avctp_init_message_buffers	This is function _bt_avctp_init_message_buffers.
	_bt_avctp_init_signal	This is function _bt_avctp_init_signal.
	_bt_avctp_l2cap_read_data_callback	This is function _bt_avctp_l2cap_read_data_callback.
	_bt_avctp_packet_assembler	This is function _bt_avctp_packet_assembler.
	_bt_avctp_send_ipid	This is function _bt_avctp_send_ipid.
	_bt_avctp_set_signal	This is function _bt_avctp_set_signal.
	_bt_avrcp_init_cmd_buffers	This is function _bt_avrcp_init_cmd_buffers.
	bt_avctp_cancel_command	brief Cancel a command message. ingroup avctp details If a message has not yet been sent to the remote device, it can be canceled (i.e. removed from send queue) by calling this function. param channel AVCTP channel. param tran_id Transaction Id. This value is obtained by calling bt_avctp_send_command. return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.

	bt_avctp_cancel_listen	brief Cancel listening for incoming connections. ingroup avctp details This function stops listening for incoming connections on the specified channel. param channel AVCTP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avctp_cancel_response	brief Cancel a response message. ingroup avctp details If a message has not yet been sent to the remote device, it can be canceled (i.e. removed from send queue) by calling this function. param channel AVCTP channel. param tran_id Transaction Id. This value is obtained by calling bt_avctp_send_command. return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.
	bt_avctp_connect	brief Connect to a remote device. ingroup avctp details This function establishes a connection to a remote device specified by the p remote_address. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVCTP callback. The events generated will either be AVCTP_EVT_CHANNEL_CONNECTED or AVCTP_EVT_CONNECTION_FAILED. param channel AVCTP channel. param remote_address The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following values:... more
	bt_avctp_create_channel	brief Allocate AVCTP channel ingroup avctp details This function allocates a new incoming AVCTP channel. The channel is intended to be used to accept a connection from a remote device. There can be only one channel for each combination of c profile_id and c psm. param mgr AVCTP manager. param profile_id Profile Id param psm The PSM on which the underlying L2CAP channel will listen and accept incoming connections. param l2cap_mode Underlying L2CAP channel mode. This currently can only be CMODE_BASIC. return li A pointer to the new AVCTP channel if the function succeeds. li c NULL otherwise.
	bt_avctp_create_outgoing_channel	brief Allocate AVCTP channel ingroup avctp details This function allocates a new outgoing AVCTP channel. The channel is intended to be used to create a connection to a remote device. There can be multiple channels with the same c profile_id and c psm. param mgr AVCTP manager. param profile_id Profile Id param psm The PSM on which the underlying L2CAP channel will listen and accept incoming connections. param l2cap_mode Underlying L2CAP channel mode. This currently can only be CMODE_BASIC. return li A pointer to the new AVCTP channel if the function succeeds. li c NULL otherwise.
	bt_avctp_destroy_channel	brief Destroy AVCTP channel. ingroup avctp details This function frees memory used by the channel. The channel has to exist and be in the "idle" state for this function to succeed. I.e. the channel has to be disconnected before this function can be called. param channel AVCTP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avctp_disconnect	brief Disconnect from a remote device. ingroup avctp details This function closes a connection to a remote device. param channel AVCTP channel. return li c TRUE if disconnection has been started. li c FALSE otherwise. No events will be generated.
	bt_avctp_get_channel_remote_address	brief Get channel's remote BT address. ingroup avctp details This function returns the address of the remote device associated with the channel. param channel AVCTP channel. return li The address of the remote device if channel is connected. li NULL otherwise.
	bt_avctp_get_channel_state	brief Get channel state ingroup avctp details This function return current state of the specified channel param channel AVCTP channel. return li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING

	bt_avctp_get_hci_connection	brief Get HCI connection for a channel ingroup avctp details This function returns a pointer to a structure that describes an HCI connection a channel is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect. param channel AVCTP channel. return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a channel specified by the p channel parameter li does... more
	bt_avctp_get_mgr	brief Return a pointer to an instance of the AVCTP manager. ingroup avctp details This function returns a pointer to an instance of the AVCTP manager. There is only one instance of the manager allocated by the stack.
	bt_avctp_init	brief Initialize the AVCTP layer. ingroup avctp details This function initializes the AVCTP layer of the stack. It must be called prior to any other AVCTP function can be called.
	bt_avctp_listen	brief Listen for incoming connections. ingroup avctp details This function enables incoming connections on the specified AVCTP channel. param channel AVCTP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avctp_send_command	brief Send a command message to a remote device. ingroup avctp details This function sends a command message to a remote device. param channel AVCTP channel. param data Message body. param data_len Message body length. param tran_id Pointer to a bt_byte where AVRCP will write transaction id assigned to the message. return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.
	bt_avctp_send_response	brief Send a response message to a remote device. ingroup avctp details This function sends a response message to a remote device. param channel AVCTP channel. param tran_id Transaction Id. This value is obtained by calling bt_avctp_send_command. param data Message body. param data_len Message body length. return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.
	bt_avctp_set_callback	brief Register a AVCTP application callback. ingroup avctp details In order to be notified of various events a consumer of the AVCTP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values: @arg AVCTP_EVT_CHANNEL_CONNECTED Channel connected. @arg AVCTP_EVT_CHANNEL_DISCONNECTED Channel disconnected. @arg AVCTP_EVT_CONNECTION_FAILED Channel connection failed (generated only if connection has been initiated by the local device). @arg AVCTP_EVT_COMMAND_RECEIVED Command received. @arg AVCTP_EVT_RESPONE_RECEIVED Response received. @arg AVCTP_EVT_COMMAND_SENT Command sent. @arg AVCTP_EVT_RESPONSE_SENT Response... more
	bt_avctp_start	This is function bt_avctp_start.

AVDTP Data Types and Constants

	Name	Description
	_bt_avdtp_codec_op_decode_t	There is currently no use for this structure.
	_bt_avdtp_codec_op_encode_t	There is currently no use for this structure.
	_bt_avdtp_codec_op_param_u	brief Parameter to a codec handler. ingroup avdtp details This union is used to pass operation specific data to a codec handler. Which member of the union points to a valid structure depends on the operation.
	_bt_avdtp_codec_op_parse_config_t	brief Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::parse - when ADVDTP needs to parse codec's capabilities/configuration received from the remote device.
	_bt_avdtp_codec_op_parse_packet_t	There is currently no use for this structure.

	_bt_avdtp_codec_op_serialize_config_t	brief Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::serialize - when ADVDTP needs to serialize codec's capabilities/configuration for sending to the remote device.
	_bt_avdtp_codec_t	brief Codec handler description. ingroup avdtp details This structure is used to register a codec handler for parsing/serializing codec capabilities and configuration. See description of the ::bt_avdtp_register_codec for more details.
	_bt_avdtp_control_channel_t	This is type bt_avdtp_control_channel_t.
	_bt_avdtp_control_cmd_t	This is type bt_avdtp_control_cmd_t.
	_bt_avdtp_ctrl_evt_data_received_t	This is type bt_avdtp_ctrl_evt_data_received_t.
	_bt_avdtp_event_u	brief Parameter to an application callback. ingroup avdtp details This union is used to pass event specific data to the AVDTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	_bt_avdtp_evt_abort_stream_requested_t	brief Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::abort_stream_requested - when AVDTP received a "abort stream" request.
	_bt_avdtp_evt_close_stream_completed_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_completed - when AVDTP received a response to a "close stream" request.
	_bt_avdtp_evt_close_stream_requested_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_requested - when AVDTP received a "close stream" request.
	_bt_avdtp_evt_ctrl_channel_connected_s	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_connected - when a control channel between two devices has been established.
	_bt_avdtp_evt_ctrl_channel_disconnected_s	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_disconnected - when a control channel between two devices has been terminated.
	_bt_avdtp_evt_discover_completed_t	brief Parameter to AVDTP_EVT_DISCOVER_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::discover_completed - when AVDTP completed discovering SEPs available on a remote device.

	_bt_avdtp_evt_get_sep_capabilities_completed_t	brief Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_sep_capabilities_completed - when AVDTP received a response to a "get SEP capabilities" request. AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_SEP_CAPABILITIES_RECEIVED - is generate with a pointer to a structure that holds actual SEP's capabilities.
	_bt_avdtp_evt_get_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_stream_configuration_completed - when AVDTP received a response to a "get stream configuration" request. AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED - is generate with a pointer to a structure that hold actual stream's configuration.
	_bt_avdtp_evt_media_packet_received_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_received - when ADVDTP received a media packet from the remote device.
	_bt_avdtp_evt_media_packet_send_failed_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_send_failed - when ADVDTP failed to send a media packet to the remote device.
	_bt_avdtp_evt_media_packet_sent_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_sent - when ADVDTP sent a media packet to the remote device.
	_bt_avdtp_evt_open_stream_completed_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_completed - when AVDTP received a response to a "open stream" request.
	_bt_avdtp_evt_open_stream_requested_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_requested - when AVDTP received a "open stream" request.

	_bt_avdtp_evt_reconfigure_stream_requested_t	brief Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::reconfigure_stream_requested - when AVDTP received a "change stream configuration" request.
	_bt_avdtp_evt_sep_capabilities_received_t	brief Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED events ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_capabilities_received - when AVDTP received a positive response to a "get SEP capabilities" or "get stream configuration" request.
	_bt_avdtp_evt_sep_info_received_t	brief Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_info_received - when AVDTP received positive result to a "discover" request. AVDTP_EVT_SEP_INFO_RECEIVED is generated for every SEP received from the remote device.
	_bt_avdtp_evt_set_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_completed - when AVDTP received a response to a "set stream configuration" request.
	_bt_avdtp_evt_set_stream_configuration_requested_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_requested - when AVDTP received a "set stream configuration" request.
	_bt_avdtp_evt_start_stream_completed_t	brief Parameter to AVDTP_EVT_START_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_completed - when AVDTP received a response to a "start stream" request.
	_bt_avdtp_evt_start_stream_requested_t	brief Parameter to AVDTP_EVT_START_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_requested - when AVDTP received a "start stream" request.
	_bt_avdtp_evt_stream_aborted_t	brief Parameter to AVDTP_EVT_STREAM_ABORTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully aborted.

	_bt_avdtp_evt_stream_closed_t	brief Parameter to AVDTP_EVT_STREAM_CLOSED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully closed.
	_bt_avdtp_evt_stream_configured_t	brief Parameter to AVDTP_EVT_STREAM_CONFIGURED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_configured - to notify the AVDTP consumer that a stream configuration has been successfully completed.
	_bt_avdtp_evt_stream_opened_t	brief Parameter to AVDTP_EVT_STREAM_OPENED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_opened - to notify the AVDTP consumer that a stream has been successfully opened.
	_bt_avdtp_evt_stream_reconfigure_completed_t	brief Parameter to AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigure_completed - when AVDTP received a response to a "change stream configuration" request.
	_bt_avdtp_evt_stream_reconfigured_t	brief Parameter to AVDTP_EVT_STREAM_RECONFIGURED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigured - to notify the AVDTP consumer that a stream configuration has been successfully changed.
	_bt_avdtp_evt_stream_security_control_completed_t	brief Parameter to AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::security_control_completed - when AVDTP received a response to a "exchange content protection control data" request.
	_bt_avdtp_evt_stream_started_t	brief Parameter to AVDTP_EVT_STREAM_STARTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_started - to notify the AVDTP consumer that a stream has been successfully started.
	_bt_avdtp_evt_stream_suspended_t	brief Parameter to AVDTP_EVT_STREAM_SUSPENDED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_suspended - to notify the AVDTP consumer that a stream has been successfully suspended.

	_bt_avdtp_evt_suspend_stream_completed_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_completed - when AVDTP received a response to a "suspend stream" request.
	_bt_avdtp_evt_suspend_stream_requested_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_requested - when AVDTP received a "suspend stream" request.
	_bt_avdtp_mgr_t	brief AVDTP manager. ingroup avdtp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avdtp_get_mgr().
	_bt_avdtp_sep_capabilities_t	brief SEP capabilities ingroup avdtp details This structure is used to hold SEP capabilities.
	_bt_avdtp_sep_t	brief SEP description ingroup avdtp details This structure is used to hold information about SEPs available on a local device.
	_bt_avdtp_stream_t	brief Stream description ingroup avdtp details This structure is used to hold information about streams available on a local device.
	_bt_avdtp_transport_channel_t	brief Transport channel description ingroup avdtp details This structure is used to hold information about transport channels available on a local device.
	_bt_avdtp_transport_session_t	brief Transport session description ingroup avdtp details This structure is used to hold information about transport sessions available on a local device.
	_bt_media_packet_t	brief Media packet buffer ingroup avdtp details This structure is used to receive and send media packet from/to the remote device. See more information about usage of this structure in descriptions of ::bt_avdtp_add_media_rx_buffer and ::bt_avdtp_add_media_tx_buffer.
	bt_avdtp_codec_handler_fp	brief Codec handler. ingroup avdtp details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it... more
	bt_avdtp_codec_op_decode_t	There is currently no use for this structure.
	bt_avdtp_codec_op_encode_t	There is currently no use for this structure.
	bt_avdtp_codec_op_param_t	This is type bt_avdtp_codec_op_param_t.
	bt_avdtp_codec_op_parse_config_t	brief Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::parse - when ADVDTP needs to parse codec's capabilities/configuration received from the remote device.
	bt_avdtp_codec_op_parse_packet_t	There is currently no use for this structure.

	bt_avdtp_codec_op_serialize_config_t	brief Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::serialize - when ADVDTP needs to serialize codec's capabilities/configuration for sending to the remote device.
	bt_avdtp_codec_t	brief Codec handler description. ingroup avdtp details This structure is used to register a codec handler for parsing/serializing codec capabilities and configuration. See description of the ::bt_avdtp_register_codec for more details.
	bt_avdtp_control_channel_t	This is type bt_avdtp_control_channel_t.
	bt_avdtp_control_cmd_t	This is type bt_avdtp_control_cmd_t.
	bt_avdtp_ctrl_evt_data_received_t	This is type bt_avdtp_ctrl_evt_data_received_t.
	bt_avdtp_event_t	brief Parameter to an application callback. ingroup avdtp details This union is used to pass event specific data to the AVDTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	bt_avdtp_evt_abort_stream_requested_t	brief Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::abort_stream_requested - when AVDTP received a "abort stream" request.
	bt_avdtp_evt_close_stream_completed_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_completed - when AVDTP received a response to a "close stream" request.
	bt_avdtp_evt_close_stream_requested_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_requested - when AVDTP received a "close stream" request.
	bt_avdtp_evt_ctrl_channel_connected_t	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_connected - when a control channel between two devices has been established.
	bt_avdtp_evt_ctrl_channel_disconnected_t	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_disconnected - when a control channel between two devices has been terminated.
	bt_avdtp_evt_discover_completed_t	brief Parameter to AVDTP_EVT_DISCOVER_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::discover_completed - when AVDTP completed discovering SEPs available on a remote device.

	bt_avdtp_evt_get_sep_capabilities_completed_t	brief Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_sep_capabilities_completed - when AVDTP received a response to a "get SEP capabilities" request. AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_SEP_CAPABILITIES_RECEIVED - is generate with a pointer to a structure that holds actual SEP's capabilities.
	bt_avdtp_evt_get_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_stream_configuration_completed - when AVDTP received a response to a "get stream configuration" request. AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED - is generate with a pointer to a structure that hold actual stream's configuration.
	bt_avdtp_evt_media_packet_received_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_received - when ADVDTP received a media packet from the remote device.
	bt_avdtp_evt_media_packet_send_failed_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_send_failed - when ADVDTP failed to send a media packet to the remote device.
	bt_avdtp_evt_media_packet_sent_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_sent - when ADVDTP sent a media packet to the remote device.
	bt_avdtp_evt_open_stream_completed_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_completed - when AVDTP received a response to a "open stream" request.
	bt_avdtp_evt_open_stream_requested_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_requested - when AVDTP received a "open stream" request.

	bt_avdtp_evt_reconfigure_stream_requested_t	brief Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::reconfigure_stream_requested - when AVDTP received a "change stream configuration" request.
	bt_avdtp_evt_sep_capabilities_received_t	brief Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED events ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_capabilities_received - when AVDTP received a positive response to a "get SEP capabilities" or "get stream configuration" request.
	bt_avdtp_evt_sep_info_received_t	brief Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_info_received - when AVDTP received positive result to a "discover" request. AVDTP_EVT_SEP_INFO_RECEIVED is generated for every SEP received from the remote device.
	bt_avdtp_evt_set_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_completed - when AVDTP received a response to a "set stream configuration" request.
	bt_avdtp_evt_set_stream_configuration_requested_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_requested - when AVDTP received a "set stream configuration" request.
	bt_avdtp_evt_start_stream_completed_t	brief Parameter to AVDTP_EVT_START_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_completed - when AVDTP received a response to a "start stream" request.
	bt_avdtp_evt_start_stream_requested_t	brief Parameter to AVDTP_EVT_START_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_requested - when AVDTP received a "start stream" request.
	bt_avdtp_evt_stream_aborted_t	brief Parameter to AVDTP_EVT_STREAM_ABORTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully aborted.

	bt_avdtp_evt_stream_closed_t	brief Parameter to AVDTP_EVT_STREAM_CLOSED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully closed.
	bt_avdtp_evt_stream_configured_t	brief Parameter to AVDTP_EVT_STREAM_CONFIGURED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_configured - to notify the AVDTP consumer that a stream configuration has been successfully completed.
	bt_avdtp_evt_stream_opened_t	brief Parameter to AVDTP_EVT_STREAM_OPENED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_opened - to notify the AVDTP consumer that a stream has been successfully opened.
	bt_avdtp_evt_stream_reconfigure_completed_t	brief Parameter to AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigure_completed - when AVDTP received a response to a "change stream configuration" request.
	bt_avdtp_evt_stream_reconfigured_t	brief Parameter to AVDTP_EVT_STREAM_RECONFIGURED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigured - to notify the AVDTP consumer that a stream configuration has been successfully changed.
	bt_avdtp_evt_stream_security_control_completed_t	brief Parameter to AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::security_control_completed - when AVDTP received a response to a "exchange content protection control data" request.
	bt_avdtp_evt_stream_started_t	brief Parameter to AVDTP_EVT_STREAM_STARTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_started - to notify the AVDTP consumer that a stream has been successfully started.
	bt_avdtp_evt_stream_suspended_t	brief Parameter to AVDTP_EVT_STREAM_SUSPENDED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_suspended - to notify the AVDTP consumer that a stream has been successfully suspended.

	bt_avdtp_evt_suspend_stream_completed_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_completed - when AVDTP received a response to a "suspend stream" request.
	bt_avdtp_evt_suspend_stream_requested_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_requested - when AVDTP received a "suspend stream" request.
	bt_avdtp_mgr_callback_fp	brief AVDTP application callback. ingroup avdtp details In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call that function whenever a new event has been generated. param mgr AVDTP manager. param evt AVDTP event. The event can be one of the following values: @arg AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED:... more
	bt_avdtp_mgr_t	brief AVDTP manager. ingroup avdtp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avdtp_get_mgr().
	bt_avdtp_sep_capabilities_t	brief SEP capabilities ingroup avdtp details This structure is used to hold SEP capabilities.
	bt_avdtp_sep_t	This is type bt_avdtp_sep_t.
	bt_avdtp_stream_t	brief Stream description ingroup avdtp details This structure is used to hold information about streams available on a local device.
	bt_avdtp_transport_channel_t	brief Transport channel description ingroup avdtp details This structure is used to hold information about transport channels available on a local device.
	bt_avdtp_transport_op_callback_fp	This is type bt_avdtp_transport_op_callback_fp.
	bt_avdtp_transport_session_t	brief Transport session description ingroup avdtp details This structure is used to hold information about transport sessions available on a local device.
	bt_media_packet_t	This is type bt_media_packet_t.
	__AVDTP_CONFIG_H	This is macro __AVDTP_CONFIG_H.
	__AVDTP_CONTROL_H	This is macro __AVDTP_CONTROL_H.
	__AVDTP_H	This is macro __AVDTP_H.
	__AVDTP_PRIVATE_H	This is macro __AVDTP_PRIVATE_H.
	AVDTP_CODEC_OPCODE_PARSE_CONFIG	< Parse codec configuration.
	AVDTP_CODEC_OPCODE_PARSE_PACKET	This is macro AVDTP_CODEC_OPCODE_PARSE_PACKET.
	AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG	< Serialize codec configuration.
	AVDTP_CODEC_TYPE_ATRAC	< ATRAC (proprietary codec owned by Sony Corporation).
	AVDTP_CODEC_TYPE_MPEG1_2_AUDIO	< MPEG-1,2 (optional).
	AVDTP_CODEC_TYPE_MPEG2_4_AAC	< MPEG-2,4 AAC (optional, used in Apple's products).
	AVDTP_CODEC_TYPE_NON_A2DP	< Vendor specific.
	AVDTP_CODEC_TYPE_SBC	< SBC (mandatory to support in A2DP profile).

	AVDTP_CTLR_CHANNEL_EVT_CONNECTED	This is macro AVDTP_CTLR_CHANNEL_EVT_CONNECTED.
	AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED	This is macro AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED.
	AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED	This is macro AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED.
	AVDTP_CTLR_CHANNEL_EVT_NOTHING	This is macro AVDTP_CTLR_CHANNEL_EVT_NOTHING.
	AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED	This is macro AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED.
	AVDTP_CTRL_CHANNEL_STATE_CONNECTED	This is macro AVDTP_CTRL_CHANNEL_STATE_CONNECTED.
	AVDTP_CTRL_CHANNEL_STATE_CONNECTING	This is macro AVDTP_CTRL_CHANNEL_STATE_CONNECTING.
	AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED	This is macro AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED.
	AVDTP_CTRL_MESSAGE_TYPE_ACCEPT	This is macro AVDTP_CTRL_MESSAGE_TYPE_ACCEPT.
	AVDTP_CTRL_MESSAGE_TYPE_COMMAND	This is macro AVDTP_CTRL_MESSAGE_TYPE_COMMAND.
	AVDTP_CTRL_MESSAGE_TYPE_FLD	This is macro AVDTP_CTRL_MESSAGE_TYPE_FLD.
	AVDTP_CTRL_MESSAGE_TYPE_REJECT	This is macro AVDTP_CTRL_MESSAGE_TYPE_REJECT.
	AVDTP_CTRL_PACKET_TYPE_CONTINUE	This is macro AVDTP_CTRL_PACKET_TYPE_CONTINUE.
	AVDTP_CTRL_PACKET_TYPE_END	This is macro AVDTP_CTRL_PACKET_TYPE_END.
	AVDTP_CTRL_PACKET_TYPE_FLD	This is macro AVDTP_CTRL_PACKET_TYPE_FLD.
	AVDTP_CTRL_PACKET_TYPE_SIGNLE	This is macro AVDTP_CTRL_PACKET_TYPE_SIGNLE.
	AVDTP_CTRL_PACKET_TYPE_START	This is macro AVDTP_CTRL_PACKET_TYPE_START.
	AVDTP_ERROR_BAD_ACP_SEID	< The requested command indicates an invalid ACP SEP ID (not addressable)
	AVDTP_ERROR_BAD_CP_FORMAT	< The format of Content Protection Service Capability is not correct.
	AVDTP_ERROR_BAD_HEADER_FORMAT	< The request packet header format is invalid.
	AVDTP_ERROR_BAD_LENGTH	< The request packet length is not match the assumed length.
	AVDTP_ERROR_BAD_MEDIA_TRANSPORT_FORMAT	< The format of Media Transport Capability is not correct.
	AVDTP_ERROR_BAD_MULTIPLEXING_FORMAT	< The format of Multiplexing Service Capability is not correct.
	AVDTP_ERROR_BAD_PAYLOAD_FORMAT	< The requested command has an incorrect payload format.
	AVDTP_ERROR_BAD_RECOVERY_FORMAT	< The format of Recovery Service Capability is not correct.
	AVDTP_ERROR_BAD_RECOVERY_TYPE	< The requested Recovery Type is not defined in AVDTP.
	AVDTP_ERROR_BAD_ROHC_FORMAT	< The format of Header Compression Service Capability is not correct.
	AVDTP_ERROR_BAD_SERV_CATEGORY	< The value of Service Category in the request packet is not defined in AVDTP.
	AVDTP_ERROR_BAD_STATE	< The stream is in state that does not permit executing commands.
	AVDTP_ERROR_FAILED_TO_CONNECT_CONTROL	< An attempt to establish a control channel has failed.
	AVDTP_ERROR_FAILED_TO_CONNECT_TRANSPORT	< An attempt to establish a transport channel has failed.
	AVDTP_ERROR_INVALID_CAPABILITIES	< The reconfigure command is an attempt to reconfigure a transport service capabilities of the SEP. Reconfigure is only permitted for application service capabilities.
	AVDTP_ERROR_NOT_SUPPORTED_COMMAND	< The requested command is not supported by the device.
	AVDTP_ERROR_SEP_IN_USE	< The SEP is in use.
	AVDTP_ERROR_SEP_NOT_IN_USE	< The SEP is not in use.
	AVDTP_ERROR_SUCCESS	< The operation completed with no errors.
	AVDTP_ERROR_UNSUPPORTED_CONFIGURAION	< Configuration not supported.
	AVDTP_EVT_ABORT_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.
	AVDTP_EVT_ABORT_STREAM_REQUESTED	< This event is generated when a local device received "abort stream" request.
	AVDTP_EVT_CLOSE_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "close stream" request.
	AVDTP_EVT_CLOSE_STREAM_REQUESTED	< This event is generated when a local device received "close stream" request.

	AVDTP_EVT_CTRL_CHANNEL_CONNECTED	< This event is generated when a control channel between two AVDTP entities has been established.
	AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED	< This event is generated when a control channel between two AVDTP entities has been terminated.
	AVDTP_EVT_CTRL_CONNECTION_FAILED	< This event is generated when a local device failed to create a control channel between two AVDTP entities.
	AVDTP_EVT_DISCOVER_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "discover" request.
	AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.
	AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.
	AVDTP_EVT_LAST	This is macro AVDTP_EVT_LAST.
	AVDTP_EVT_MEDIA_PACKET_RECEIVED	< This event is generated when a local device received a media packet.
	AVDTP_EVT_MEDIA_PACKET_SEND_FAILED	< This event is generated when a local device failed to send a media packet.
	AVDTP_EVT_MEDIA_PACKET_SENT	< This event is generated when a local device sent a media packet.
	AVDTP_EVT_NULL	This is macro AVDTP_EVT_NULL.
	AVDTP_EVT_OPEN_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "open stream" request.
	AVDTP_EVT_OPEN_STREAM_REQUESTED	< This event is generated when a local device received "open stream" request.
	AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED	< This event is generated when a local device received "change stream configuration" request.
	AVDTP_EVT_SEP_CAPABILITIES_RECEIVED	< This event is generated when a local device received a positive response to a "get SEP capabilities" request.
	AVDTP_EVT_SEP_INFO_RECEIVED	< This event is generated for each SEP contained in a positive response to a "discover" request.
	AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.
	AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED	< This event is generated when a local device received "set stream configuration" request.
	AVDTP_EVT_START_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "start stream" request.
	AVDTP_EVT_START_STREAM_REQUESTED	< This event is generated when a local device received "start stream" request.
	AVDTP_EVT_STREAM_ABORTED	< This event is generated when a local device has successfully aborted a stream. < This event follows the AVDTP_EVT_ABORT_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream abortion was initiated by the local device.
	AVDTP_EVT_STREAM_CLOSED	< This event is generated when a local device has successfully closed a stream. < This event follows the AVDTP_EVT_CLOSE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream closing was initiated by the local device.
	AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED	< This event is generated when a local device received a positive response to a "get stream configuration" request.
	AVDTP_EVT_STREAM_CONFIGURED	< This event is generated when a local device has successfully configured a stream. < This event follows the AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the upper layer has accepted it. < This event is not generated if stream configuration was initiated by the local device.

	AVDTP_EVT_STREAM_OPENED	< This event is generated when a local device has successfully opened a stream. < This event follows the AVDTP_EVT_OPEN_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream opening was initiated by the local device.
	AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.
	AVDTP_EVT_STREAM_RECONFIGURED	< This event is generated when a local device has successfully reconfigured a stream. < This event follows the AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream reconfiguration was initiated by the local device.
	AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.
	AVDTP_EVT_STREAM_STARTED	< This event is generated when a local device has successfully started a stream. < This event follows the AVDTP_EVT_START_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream starting was initiated by the local device.
	AVDTP_EVT_STREAM_SUSPENDED	< This event is generated when a local device has successfully suspended a stream. < This event follows the AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream suspension was initiated by the local device.
	AVDTP_EVT_SUSPEND_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.
	AVDTP_EVT_SUSPEND_STREAM_REQUESTED	< This event is generated when a local device received "suspend stream" request.
	AVDTP_MANAGER_STATE_CONNECTING	This is macro AVDTP_MANAGER_STATE_CONNECTING.
	AVDTP_MANAGER_STATE_IDLE	This is macro AVDTP_MANAGER_STATE_IDLE.
	AVDTP_MAX_STREAM_TRANSPORT_SESSION	This is macro AVDTP_MAX_STREAM_TRANSPORT_SESSION.
	AVDTP_MEDIA_TYPE_AUDIO	< Audio.
	AVDTP_MEDIA_TYPE_MULTIMEDIA	< Both Audio & Video.
	AVDTP_MEDIA_TYPE_VIDEO	< Video.
	AVDTP_SEP_CAPABILITY_FLAG_CONTENT_PROTECTION	< Content Prortection. A SEP is capable of transferring content protection packets.
	AVDTP_SEP_CAPABILITY_FLAG_HEADER_COMPRESSION	< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.
	AVDTP_SEP_CAPABILITY_FLAG_MEDIA_CODEC	< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.
	AVDTP_SEP_CAPABILITY_FLAG_MEDIA_TRANSPORT	< Media. A SEP is capable of transferring media (audio, video or both) packets.
	AVDTP_SEP_CAPABILITY_FLAG_MULTIPLEXING	< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.
	AVDTP_SEP_CAPABILITY_FLAG_RECOVERY	< Recovery. A SEP is capable of transferring recovery packets.
	AVDTP_SEP_CAPABILITY_FLAG_REPORTING	< Reporting. A SEP is capable of transferring reporting packets.
	AVDTP_SEP_SERVICE_CAPABILITY_CONTENT_PROTECTION	< Content Prortection. A SEP is capable of transferring content protection packets.
	AVDTP_SEP_SERVICE_CAPABILITY_HEADER_COMPRESSION	< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.
	AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_CODEC	< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.
	AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_TRANSPORT	< Media. A SEP is capable of transferring media (audio, video or both) packets.
	AVDTP_SEP_SERVICE_CAPABILITY_MULTIPLEXING	< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.

	AVDTP_SEP_SERVICE_CAPABILITY_RECOVERY	< Recovery. A SEP is capable of transferring recovery packets.
	AVDTP_SEP_SERVICE_CAPABILITY_REPORTING	< Reporting. A SEP is capable of transferring reporting packets.
	AVDTP_SEP_STATE_FREE	This is macro AVDTP_SEP_STATE_FREE.
	AVDTP_SEP_STATE_IDLE	This is macro AVDTP_SEP_STATE_IDLE.
	AVDTP_SEP_TYPE_SINK	< Sink (usually a device like a headphones or BMW).
	AVDTP_SEP_TYPE_SOURCE	< Source (usually a device like a phone, desktop or laptop).
	AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS	< The stream is closing transport channels.
	AVDTP_STREAM_FLAG_LISTENING	This is macro AVDTP_STREAM_FLAG_LISTENING.
	AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS	< The stream is opening transport channels.
	AVDTP_STREAM_STATE_ABORTING	< The stream is aborting. This means that all transport channels associated with the stream are being closed. < After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.
	AVDTP_STREAM_STATE_CLOSING	< The stream is closing. This means that all transport channels associated with the stream are being closed. < After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.
	AVDTP_STREAM_STATE_CONFIGURED	< The stream has been configured.
	AVDTP_STREAM_STATE_IDLE	< The stream is idle. This can mean two things. The stream specified by strm_handle does not exist < or the stream is closed.
	AVDTP_STREAM_STATE_OPEN	< The stream has been opened.
	AVDTP_STREAM_STATE_STREAMING	< The stream has been started. Depending on the local SEP type (source or sink) it means < that the stream can send or receive media packets.
	AVDTP_TC_OPCODE_CONNECT	This is macro AVDTP_TC_OPCODE_CONNECT.
	AVDTP_TC_OPCODE_DISCONNECT	This is macro AVDTP_TC_OPCODE_DISCONNECT.
	AVDTP_TC_STATUS_ERROR	This is macro AVDTP_TC_STATUS_ERROR.
	AVDTP_TC_STATUS_SUCCESS	This is macro AVDTP_TC_STATUS_SUCCESS.
	AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED	This is macro AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED.
	AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED	This is macro AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED.
	AVDTP_TRANSPORT_SESSION_TYPE_MEDIA	< Media (audio or video).
	AVDTP_TRANSPORT_SESSION_TYPE_RECOVERY	< Recovery (currently not supported).
	AVDTP_TRANSPORT_SESSION_TYPE_REPORTING	< Reporting (currently not supported).
	bt_avdtp_connect	brief Connect to a remote device. ingroup avdtp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP_EVT_CTRL_CHANNEL_CONNECTED or AVDTP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr AVDTP manager. param remote_addr The address of a remote device. return li c TRUE if connection establishment has been started. li c FALSE... more

	bt_avdtp_connect_ex	brief Connect to a remote device. ingroup avdtp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP_EVT_CTRL_CHANNEL_CONNECTED or AVDTP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr AVDTP manager. param remote_addr The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following... more
	AVDTP_CMD_DELAYREPORT	This is macro AVDTP_CMD_DELAYREPORT.
	AVDTP_CMD_GET_ALL_CAPABILITIES	This is macro AVDTP_CMD_GET_ALL_CAPABILITIES.
	AVDTP_CONTENT_PROTECTION_METHOD_SCMS_T	This is macro AVDTP_CONTENT_PROTECTION_METHOD_SCMS_T.
	AVDTP_CTRL_MESSAGE_TYPE_GENERAL_REJECT	This is macro AVDTP_CTRL_MESSAGE_TYPE_GENERAL_REJECT.
	AVDTP_SCMS_T_CP_BIT	This is macro AVDTP_SCMS_T_CP_BIT.
	AVDTP_SCMS_T_L_BIT	This is macro AVDTP_SCMS_T_L_BIT.
	bt_avdtp_evt_delay_report_completed_t	brief Parameter to AVDTP_EVT_DELAYREPORT_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::delay_report_completed - when AVDTP received a response to a "delay report" request.
	bt_cp_header_t	This is type bt_cp_header_t.
	_bt_avdtp_evt_delay_report_completed_t	brief Parameter to AVDTP_EVT_DELAYREPORT_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::delay_report_completed - when AVDTP received a response to a "delay report" request.
	_bt_cp_header_s	This is record _bt_cp_header_s.
	_bt_avdtp_evt_ctrl_connection_failed_s	brief Parameter to AVDTP_EVT_CTRL_CONNECTION_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_connection_failed - when a control channel between two devices has been established.
	_bt_avdtp_evt_set_stream_configuration_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration - when AVDTP received a "set stream configuration" request.
	AVDTP_EVT_DELAYREPORT_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "delay report" request.
	AVDTP_EVT_SET_STREAM_CONFIGURATION	This is macro AVDTP_EVT_SET_STREAM_CONFIGURATION.
	AVDTP_SEP_CAPABILITY_FLAG_DELAY_REPORTING	< Delat reporitng.
	AVDTP_SEP_SERVICE_CAPABILITY_DELAY_REPORTING	< Delay Reporting.

	bt_avdtp_evt_ctrl_connection_failed_t	brief Parameter to AVDTP_EVT_CTRL_CONNECTION_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_connection_failed - when a control channel between two devices has been established.
	bt_avdtp_evt_set_stream_configuration_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration - when AVDTP received a "set stream configuration" request.

AVDTP Functions

	Name	Description
	_bt_avdtp_add_param_byte	This is function _bt_avdtp_add_param_byte.
	_bt_avdtp_add_param_uint	This is function _bt_avdtp_add_param_uint.
	_bt_avdtp_allocate_buffers	This is function _bt_avdtp_allocate_buffers.
	_bt_avdtp_allocate_cmd	This is function _bt_avdtp_allocate_cmd.
	_bt_avdtp_allocate_sep	This is function _bt_avdtp_allocate_sep.
	_bt_avdtp_allocate_sep_config	This is function _bt_avdtp_allocate_sep_config.
	_bt_avdtp_allocate_stream	This is function _bt_avdtp_allocate_stream.
	_bt_avdtp_allocate_transport_channel	This is function _bt_avdtp_allocate_transport_channel.
	_bt_avdtp_allocate_transport_session_id	This is function _bt_avdtp_allocate_transport_session_id.
	_bt_avdtp_begin_tc_channel_operation	This is function _bt_avdtp_begin_tc_channel_operation.
	_bt_avdtp_commit_tc_channel_operation	This is function _bt_avdtp_commit_tc_channel_operation.
	_bt_avdtp_control_channel_accept_handler	This is function _bt_avdtp_control_channel_accept_handler.
	_bt_avdtp_control_channel_cmd_handler	This is function _bt_avdtp_control_channel_cmd_handler.
	_bt_avdtp_control_channel_event_handler	This is function _bt_avdtp_control_channel_event_handler.
	_bt_avdtp_control_channel_reject_handler	This is function _bt_avdtp_control_channel_reject_handler.
	_bt_avdtp_execute_tc_channel_operation	This is function _bt_avdtp_execute_tc_channel_operation.
	_bt_avdtp_find_listening_stream	This is function _bt_avdtp_find_listening_stream.
	_bt_avdtp_find_stream	This is function _bt_avdtp_find_stream.
	_bt_avdtp_find_stream_by_remote_sep_id	This is function _bt_avdtp_find_stream_by_remote_sep_id.
	_bt_avdtp_find_stream_by_sep_id	This is function _bt_avdtp_find_stream_by_sep_id.
	_bt_avdtp_free_cmd	This is function _bt_avdtp_free_cmd.
	_bt_avdtp_free_sep_config	This is function _bt_avdtp_free_sep_config.
	_bt_avdtp_free_stream	This is function _bt_avdtp_free_stream.
	_bt_avdtp_free_transport_channel	This is function _bt_avdtp_free_transport_channel.
	_bt_avdtp_get_control_channel	This is function _bt_avdtp_get_control_channel.
	_bt_avdtp_init_cmd_buffers	This is function _bt_avdtp_init_cmd_buffers.
	_bt_avdtp_init_sep_config_buffers	This is function _bt_avdtp_init_sep_config_buffers.
	_bt_avdtp_init_signal	This is function _bt_avdtp_init_signal.
	_bt_avdtp_is_sep_inuse	This is function _bt_avdtp_is_sep_inuse.
	_bt_avdtp_open_control_channel_ex	This is function _bt_avdtp_open_control_channel_ex.
	_bt_avdtp_open_control_channel_ex	This is function _bt_avdtp_open_control_channel_ex.
	_bt_avdtp_register_transport_channel_for_operation	This is function _bt_avdtp_register_transport_channel_for_operation.
	_bt_avdtp_send_command	This is function _bt_avdtp_send_command.
	_bt_avdtp_send_media_packet	This is function _bt_avdtp_send_media_packet.
	_bt_avdtp_set_signal	This is function _bt_avdtp_set_signal.
	_bt_avdtp_transport_l2cap_read_data_callback	This is function _bt_avdtp_transport_l2cap_read_data_callback.
	_bt_avdtp_transport_l2cap_state_changed_callback	This is function _bt_avdtp_transport_l2cap_state_changed_callback.
	_bt_avdtp_write_caps	This is function _bt_avdtp_write_caps.

	bt_avdtp_abort_stream	brief Suspend a stream. ingroup avdtp details This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE. As a result of this operation the AVDTP_EVT_ABORT_STREAM_COMPLETED event will be generated. This operation cannot be rejected. The p evt_param.abort_stream_requested.err_code is always == AVDTP_ERROR_SUCCESS. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.
	bt_avdtp_add_media_rx_buffer	brief Add a media packet buffer to a receive queue ingroup avdtp details The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more
	bt_avdtp_add_media_tx_buffer	brief Add a media packet buffer to a send queue ingroup avdtp details When the consumer of AVDTP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. When the packet has been successfully sent a AVDTP_EVT_MEDIA_PACKET_SENT is generated. Otherwise a AVDTP_EVT_MEDIA_PACKET_SEND_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as AVDTP has removed it from the queue and gave up... more
	bt_avdtp_cancel_listen	brief Cancel listening for incoming connections. ingroup avdtp details This function removes a SEP from a list of SEPS which a stream can use for incoming requests. param mgr AVDTP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avdtp_close_stream	brief Close a stream. ingroup avdtp details This function tries to close a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN or AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the AVDTP_EVT_CLOSE_STREAM_COMPLETED event will be generated. If the stream has been closed the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds,... more
	bt_avdtp_create_stream	brief Create a stream. ingroup avdtp details This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort. param mgr AVDTP manager. return li c Stream handle if the function succeeds. li c 0 otherwise.
	bt_avdtp_destroy_stream	brief Destroy a stream. ingroup avdtp details This function frees memory used by the stream. The stream has to exist and be in the "idle" state for this function to succeed. I.e. the stream has to be closed or aborted before this function can be called. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise.

	bt_avdtp_disconnect	brief Disconnect from a remote device. ingroup avdtp details This function closes a control and transport channels on all streams associated with the remote device specified by the p remote_addr. As a result of this operation the following events will be generated: @arg AVDTP_EVT_MEDIA_PACKET_RECEIVED: if a stream's receive queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg AVDTP_EVT_MEDIA_PACKET_SENT: if a stream's send queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg AVDTP_EVT_STREAM_CLOSED: this event is generate if a stream is in... more
	bt_avdtp_discover	brief Discover SEPs on a remote device. ingroup avdtp details This function asks the remote device to send a list of all available SEPs. As a result of this operation the following events will be generated: @arg AVDTP_EVT_SEP_INFO_RECEIVED: this event is generated for every SEP received from the remote device. the p evt_param.sep_info_received contains SEP information. @arg AVDTP_EVT_DISCOVER_COMPLETED: this event is generated after last AVDTP_EVT_SEP_INFO_RECEIVED if the remote accepted the request and the p evt_param.discover_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.discover_completed.err_code == the error code sent by the remote. param mgr AVDTP... more
	bt_avdtp_find_codec	brief Find a codec ingroup avdtp details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store... more
	bt_avdtp_get_capabilities	brief Get remote SEP capabilities. ingroup avdtp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after AVDTP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr AVDTP manager.... more
	bt_avdtp_get_configuration	brief Get stream configuration. ingroup avdtp details This function requests stream configuration from a remote device. As a result of this operation the following events will be generated: @arg AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: this event is generated if the remote accepted the request. the p ebt_para.sep_capabilities_received.caps will contain current stream configuration. @arg AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: If the remote accepted the request the p evt_param.get_stream_configuration_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_stream_configuration_completed.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the actual... more

	bt_avdtp_get_hci_connection	brief Get HCI connection for a stream ingroup avdtp details This function returns a pointer to a structure that describes an HCI connection a stream is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call bt_hci_disconnect. param mgr AVDTP manager. param strm_handle Stream handle. return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a stream specified by the p strm_handle... more
	bt_avdtp_get_mgr	brief Return a pointer to an instance of the AVDTP manager. ingroup avdtp details This function returns a pointer to an instance of the AVDTP manager. There is only one instance of the manager allocated by the stack. The pointer is passed as the first parameter to all AVDTP functions.
	bt_avdtp_get_sep	brief Get a SEP info by its ID. ingroup avdtp details This function returns a pointer to bt_avdtp_sep_t structure that describes a SEP previously registered with bt_avdtp_register_sep. param mgr AVDTP manager. param sep_id The ID of a SEP. return li c Pointer to bt_avdtp_sep_t if the SEP is in the list of registered SEPs. li c NULL otherwise.
	bt_avdtp_get_stream_codec_config	brief Get the configuration of the codec currently used with the stream. ingroup avdtp details This function returns a pointer to a structure that contains configuration of the codec currently used with the stream. The structure returned depends on the codec. The dotstack defines structures only for SBC, MPEG-1,2 and MPEG-2,4 AAC codecs: @arg SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h) @arg MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h) @arg MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h) param mgr AVDTP manager. param strm_handle Stream handle. return li The codec's configuration if strm_handle specifies a valid stream and the stream is in one of the following... more
	bt_avdtp_get_stream_codec_type	brief Get the type of the codec currently used with the stream. ingroup avdtp details This function returns the type of the codec currently used with the stream. param mgr AVDTP manager. param strm_handle Stream handle. return @arg The type of the codec if strm_handle specifies a valid stream and the stream is in one of the following states: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING @arg The result will be one of the following values: AVDTP_CODEC_TYPE_SBC: SBC AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP... more
	bt_avdtp_get_stream_config	brief Get stream's configuration. ingroup avdtp details This function returns a pointer to a structure holding the current configuration of stream. param mgr AVDTP manager. param strm_handle Stream handle. return li The stream's configuration if strm_handle specifies a valid stream and the stream is in one of the following state: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING li NULL otherwise.
	bt_avdtp_get_stream_local_sep_id	brief Get stream's local SEP ID. ingroup avdtp details This function returns the ID of the local SEP associated with the stream. param mgr AVDTP manager. param strm_handle Stream handle. return li The ID of the local SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_avdtp_get_stream_remote_address	brief Get stream's remote BT address. ingroup avdtp details This function returns the address of the remote device associated with the stream. param mgr AVDTP manager. param strm_handle Stream handle. return li The address of the remote device if strm_handle specifies a valid stream. li NULL otherwise.

	bt_avdtp_get_stream_remote_sep_id	brief Get stream's remote SEP ID. ingroup avdtp details This function returns the ID of the remote SEP associated with the stream. param mgr AVDTP manager. param strm_handle Stream handle. return li The ID of the remote SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_avdtp_get_stream_state	brief Get local stream state. ingroup avdtp details This function returns local state of a stream specified by the p strm_handle. No request is sent to the remote party. param mgr AVDTP manager. param strm_handle Stream handle. return The state of the stream. The result will be one of the following values: @arg AVDTP_STREAM_STATE_IDLE: The stream is idle. This can mean two things. The stream specified by p strm_handle does not exist or the stream is closed. @arg AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS: The stream is opening transport channels. @arg AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS: The stream is closing transport channels. @arg AVDTP_STREAM_STATE_CONFIGURED: The... more
	bt_avdtp_init	brief Initialize the AVDTP layer. ingroup avdtp details This function initializes the AVDTP layer of the stack. It must be called prior to any other AVDTP function can be called.
	bt_avdtp_listen	brief Listen for incoming connections. ingroup avdtp details This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream. param mgr AVDTP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if... more
	bt_avdtp_open_stream	brief Open a stream. ingroup avdtp details This function tries to open a stream by sending a request to the remote party. The stream has to be already configured with a bt_avdtp_set_configuration call. As a result of this operation the AVDTP_EVT_OPEN_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.open_stream_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to open the stream, the p evt_param.open_stream_completed.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function... more
	bt_avdtp_reconfigure_stream	brief Reconfigure stream. ingroup avdtp details This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event will be generated. If reconfiguration was a success the p evt_param.stream_reconfigure_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.stream_reconfigure_completed.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. param caps New stream configuration. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be... more
	bt_avdtp_register_callback	brief Register a AVDTP application callback. ingroup avdtp details In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values: @arg AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote... more

	bt_avdtp_register_codec	brief Register a codec ingroup avdtp details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store... more
	bt_avdtp_register_sep	brief Register a SEP with the local AVDTP manager. ingroup avdtp details This function is used to make a list of SEPs supported by the local ADVTP entity. param mgr AVDTP manager. param type The type of a SEP. The type can be one of the following values: @arg AVDTP_SEP_TYPE_SOURCE: The SEP is a source. @arg AVDTP_SEP_TYPE_SINK: The SEP is a sink. param caps The capabilities of a SEP. return li c ID of a SEP if the function succeeds. li c FALSE otherwise.
	bt_avdtp_remove_media_rx_buffer	brief Remove a media packet buffer from a receive queue ingroup avdtp details The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more
	bt_avdtp_remove_media_tx_buffer	brief Remove a media packet buffer from a send queue ingroup avdtp details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling bt_avdtp_remove_media_tx_buffer. param mgr AVDTP manager. param strm_handle Stream handle. param buffer Pointer to a structure... more
	bt_avdtp_security_control	brief Exchange content protection control data. ingroup avdtp details This function tries to establish content protection by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE, AVDTP_STREAM_STATE_CLOSING, AVDTP_STREAM_STATE_ABORTING. As a result of this operation the AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event will be generated. If the stream's content protection data has been accepted by the remote party the p evt_param.security_control_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.security_control_completed.err_code == the error code sent by the remote. note The dotstack does not support content protection. Although the request can be sent it will not affect... more
	bt_avdtp_set_configuration	brief Set stream configuration. ingroup avdtp details This function tries to configure a stream before opening it. As a result of this operation the AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event will be generated. If configuration was a success the p evt_param.set_stream_configuration_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.set_stream_configuration_completed.err_code == the error code sent by the remote and p evt_param.set_stream_configuration_completed.svc_category == the value of the first Service Category to fail. param mgr AVDTP manager. param strm_handle Stream handle. param remote_addr The address of a remote device. param seid_int Local SEP ID. param seid_acp Remote SEP ID. param caps Stream configuration. return li c TRUE if... more
	bt_avdtp_start	brief Start the AVDTP layer. ingroup avdtp details This function makes the AVDTP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required. param mgr AVDTP manager. return li c TRUE if the function succeeds. li c FALSE otherwise.

	bt_avdtp_start_stream	brief Start a stream. ingroup avdtp details This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the AVDTP_EVT_START_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.start_stream_requested.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the p evt_param.start_stream_requested.err_code == the error code sent... more
	bt_avdtp_suspend_stream	brief Suspend a stream. ingroup avdtp details This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the AVDTP_EVT_SUSPEND_STREAM_COMPLETED event will be generated. If the stream has been suspended the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the... more
	bt_avdtp_unregister_codec	brief Unregister a codec ingroup avdtp details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store... more
	_bt_avdtp_add_param_uintn	This is function _bt_avdtp_add_param_uintn.
	_bt_avdtp_read_caps	This is function _bt_avdtp_read_caps.
	bt_avdtp_get_l2cap_channel	This is function bt_avdtp_get_l2cap_channel.
	bt_avdtp_clear_media_tx_queue	brief Clear send queue ingroup avdtp details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. The consumer can remove all packets from the queue before they have been sent to a remote device by calling ::bt_avdtp_clear_media_tx_queue. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails... more
	bt_avdtp_get_all_capabilities	brief Get remote SEP capabilities. ingroup avdtp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after AVDTP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr AVDTP manager.... more
	bt_avdtp_get_stream_direction	This is function bt_avdtp_get_stream_direction.

	bt_avdtp_report_delay	brief Report delay value of a Sink to a Source. ingroup avdtp details This function sends the delay value of a Sink to a Source. This enables synchronous playback of audio and video. Delay reports are always sent from the Sink to the Source. If the Sink's delay report has been accepted by the Source the p evt_param.delay_report_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.delay_report_completed.err_code == the error code sent by the Source. param mgr AVDTP manager. param strm_handle Stream handle. param delay The delay value in 1/10 milliseconds. return li c TRUE if the function succeeds, i.e. the actual... more
	bt_avdtp_set_media_tx_queue_limit	brief Set limit on the send queue ingroup avdtp details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. By default the send queue can contain unlimited number of packets. The consumer can set a limit on how many packets are held in the queue. In this case when new packet is added to the queue and the length of... more

AVRCP Data Types and Constants

	Name	Description
	_bt_av_add_to_now_playing_s	brief Parameter to AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::add_to_now_playing_status - when a local device received a response to a "add to now playing" request.
	_bt_av_battery_status_of_ct_s	brief Parameter to AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::battery_status_of_ct - when a local device received a "battery status of controller" command.
	_bt_av_capability_company_id_s	brief Parameter to AVRCP_EVT_COMPANY_ID_LIST_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::company_id - when a local device received a response to a "get company id" request.
	_bt_av_capability_event_id_s	brief Parameter to AVRCP_EVT_EVENT_ID_LIST_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::supported_event_id - when a local device received a response to a "get supported events" request.
	_bt_av_command_t	This is record _bt_av_command_t.

	_bt_av_displayable_character_set_s	brief Parameter to AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::displayable_character_set - when a local device received a "displayable chracter set command" request.
	_bt_av_element_attribute_s	brief Media element attribute ingroup avrcp details This structure is used to store media element attribute.
	_bt_av_element_attributes_s	brief Parameter to AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::element_attributes - when a local device received a response to a "get media element attributes" request.
	_bt_av_element_id_s	brief Media element UID ingroup avrcp details This structure is used to store media element UID.
	_bt_av_get_element_attributes_s	brief Parameter to AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::get_element_attributes - when a local device received a "get element attributes" request.
	_bt_av_notification_addressed_player_changed_s	brief Parameter to AVRCP_EVT_ADDRESSED_PLAYER_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:addressed_player - when a local device received a "addressed player changed" notification.
	_bt_av_notification_app_setting_changed_s	This is type bt_av_notification_app_setting_changed_t.
	_bt_av_notification_battery_status_s	brief Parameter to AVRCP_EVT_BATT_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:battery_status - when a local device received a "battery status changed" notification.
	_bt_av_notification_playback_pos_changed_s	brief Parameter to AVRCP_EVT_PLAYBACK_POS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:playback_pos - when a local device received a "playback position changed" notification.

	_bt_av_notification_playback_status_changed_s	brief Parameter to AVRCP_EVT_PLAYBACK_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:play_status - when a local device received a "play status changed" notification.
	_bt_av_notification_s	brief Parameter to the following events: li AVRCP_EVT_PLAYBACK_STATUS_CHANGED li AVRCP_EVT_TRACK_CHANGED li AVRCP_EVT_PLAYBACK_POS_CHANGED li AVRCP_EVT_BATT_STATUS_CHANGED li AVRCP_EVT_SYSTEM_STATUS_CHANGED li AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED li AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED li AVRCP_EVT_ADDRESSED_PLAYER_CHANGED li AVRCP_EVT_UIDS_CHANGED li AVRCP_EVT_VOLUME_CHANGED ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification - when a local device received one of the following notifications from the target: li Play status changed li Track changed changed li Playback position changed li Battery status changed li System status changed li Addressed player changed li UIDs changed li Volume changed li Player application setting changed... more
	_bt_av_notification_system_status_changed_s	brief Parameter to AVRCP_EVT_SYSTEM_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:system_status - when a local device received a "system status changed" notification.
	_bt_av_notification_track_changed_s	brief Parameter to AVRCP_EVT_TRACK_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:track - when a local device received a "track changed" notification.
	_bt_av_notification_uids_changed_s	brief Parameter to AVRCP_EVT_UIDS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:uids - when a local device received a "UIDs changed" notification.
	_bt_av_notification_volume_changed_s	brief Parameter to AVRCP_EVT_VOLUME_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:volume - when a local device received a "UIDs changed" notification.

	_bt_av_play_item_s	brief Parameter to AVRCP_EVT_PLAY_ITEM_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_item_status - when a local device received a response to a "play item" request.
	_bt_av_play_status_s	brief Parameter to AVRCP_EVT_GET_PLAY_STATUS_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_status - when a local device received a response to a "get play status" request.
	_bt_av_player_setting_current_values_s	brief Parameter to AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_current_values - when a local device received a response to a "get current player setting attribute values" request.
	_bt_av_player_setting_values_s	brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values - when a local device received a response to a "get player setting attribute values" request.
	_bt_av_player_setting_values_text_s	brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values_text - when a local device received a response to a "get player setting attribute values displayable text" request.
	_bt_av_player_settings_s	brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings - when a local device received a response to a "get supported player setting attributes" request.
	_bt_av_player_settings_text_s	brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings_text - when a local device received a response to a "get player setting attributes displayable text" request.
	_bt_av_player_text_s	This is type bt_av_player_text_t.

	_bt_av_register_notification_t	brief Parameter to AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::register_notification - when a local device received a "register notification" request.
	_bt_av_response_t	brief AV/C response header ingroup avrcp details This structure is used to store fields present in every AV/C response.
	_bt_av_set_absolute_volume_s	brief Parameter to AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::absolute_volume - when a local device received a response to a "set absolute volume" request.
	_bt_av_set_addressed_player_s	brief Parameter to AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::addressed_player - when a local device received a response to a "set addressed player" request.
	_bt_avrcp_channel_t	brief AVRCP channel description ingroup avrcp details This structure is used to hold information about an AVRCP channel.
	_bt_avrcp_device_s	This is type bt_avrcp_device_t.
	_bt_avrcp_event_u	brief Parameter to an application callback. ingroup avrcp details This union is used to pass event specific data to the AVRCP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	_bt_avrcp_evt_channel_connected_t	brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_CONNECTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_connected - when a control channel between two devices has been established.
	_bt_avrcp_evt_channel_disconnected_t	brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_disconnected - when a control channel between two devices has been terminated.

	_bt_avrcp_evt_connection_failed_t	brief Parameter to AVRCP_EVT_CONTROL_CONNECTION_FAILED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::connection_failed - when a local device failed to create a control channel between two AVRCP entities.
	_bt_avrcp_evt_panel_command_received_t	brief Parameter to AVRCP_EVT_PANEL_COMMAND_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_command_received - when a local device received a PASS THROUGH command.
	_bt_avrcp_evt_panel_response_received_t	brief Parameter to AVRCP_EVT_PANEL_RESPONSE_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_response_received - when a local device received a response to a PASS THROUGH command.
	_bt_avrcp_evt_register_events_completed_t	This is type bt_avrcp_evt_register_events_completed_t.
	_bt_avrcp_evt_search_completed_s	brief Parameter to AVRCP_EVT_SEARCH_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::device_search - when searching for nearby devices has finished.
	_bt_avrcp_mgr_t	brief AVRCP manager. ingroup avrcp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avrcp_get_mgr().
	bt_av_add_to_now_playing_t	brief Parameter to AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::add_to_now_playing_status - when a local device received a response to a "add to now playing" request.
	bt_av_battery_status_of_ct_t	brief Parameter to AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::battery_status_of_ct - when a local device received a "battery status of controller" command.
	bt_av_capability_company_id_t	brief Parameter to AVRCP_EVT_COMPANY_ID_LIST_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::company_id - when a local device received a response to a "get company id" request.

	bt_av_capability_event_id_t	brief Parameter to AVRCP_EVT_EVENT_ID_LIST_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::supported_event_id - when a local device received a response to a "get supported events" request.
	bt_av_command_t	This is type bt_av_command_t.
	bt_av_displayable_character_set_t	brief Parameter to AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::displayable_character_set - when a local device received a "displayable chracter set command" request.
	bt_av_element_attribute_t	brief Media element attribute ingroup avrcp details This structure is used to store media element attribute.
	bt_av_element_attributes_t	brief Parameter to AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::element_attributes - when a local device received a response to a "get media element attributes" request.
	bt_av_element_id_t	brief Media element UID ingroup avrcp details This structure is used to store media element UID.
	bt_av_get_element_attributes_t	brief Parameter to AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::get_element_attributes - when a local device received a "get element attributes" request.
	bt_av_notification_addressed_player_changed_t	brief Parameter to AVRCP_EVT_ADDRESSED_PLAYER_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:addressed_player - when a local device received a "addressed player changed" notification.
	bt_av_notification_app_setting_changed_t	This is type bt_av_notification_app_setting_changed_t.
	bt_av_notification_battery_status_t	brief Parameter to AVRCP_EVT_BATT_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:battery_status - when a local device received a "battery status changed" notification.

	bt_av_notification_playback_pos_changed_t	brief Parameter to AVRCP_EVT_PLAYBACK_POS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:playback_pos - when a local device received a "playback position changed" notification.
	bt_av_notification_playback_status_changed_t	brief Parameter to AVRCP_EVT_PLAYBACK_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:play_status - when a local device received a "play status changed" notification.
	bt_av_notification_system_status_changed_t	brief Parameter to AVRCP_EVT_SYSTEM_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:system_status - when a local device received a "system status changed" notification.
	bt_av_notification_t	brief Parameter to the following events: li AVRCP_EVT_PLAYBACK_STATUS_CHANGED li AVRCP_EVT_TRACK_CHANGED li AVRCP_EVT_PLAYBACK_POS_CHANGED li AVRCP_EVT_BATT_STATUS_CHANGED li AVRCP_EVT_SYSTEM_STATUS_CHANGED li AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED li AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED li AVRCP_EVT_ADDRESSED_PLAYER_CHANGED li AVRCP_EVT_UIDS_CHANGED li AVRCP_EVT_VOLUME_CHANGED ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification - when a local device received one of the following notifications from the target: li Play status changed li Track changed changed li Playback position changed li Battery status changed li System status changed li Addressed player changed li UIDs changed li Volume changed li Player application setting changed... more
	bt_av_notification_track_changed_t	brief Parameter to AVRCP_EVT_TRACK_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:track - when a local device received a "track changed" notification.
	bt_av_notification_uids_changed_t	brief Parameter to AVRCP_EVT_UIDS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:uids - when a local device received a "UIDs changed" notification.

	bt_av_notification_volume_changed_t	brief Parameter to AVRCP_EVT_VOLUME_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:volume - when a local device received a "UIDs changed" notification.
	bt_av_play_item_t	brief Parameter to AVRCP_EVT_PLAY_ITEM_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_item_status - when a local device received a response to a "play item" request.
	bt_av_play_status_t	brief Parameter to AVRCP_EVT_GET_PLAY_STATUS_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_status - when a local device received a response to a "get play status" request.
	bt_av_player_setting_current_values_t	brief Parameter to AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_current_values - when a local device received a response to a "get current player setting attribute values" request.
	bt_av_player_setting_values_t	brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values - when a local device received a response to a "get player setting attribute values" request.
	bt_av_player_setting_values_text_t	brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values_text - when a local device received a response to a "get player setting attribute values displayable text" request.
	bt_av_player_settings_t	brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings - when a local device received a response to a "get supported player setting attributes" request.

	bt_av_player_settings_text_t	brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings_text - when a local device received a response to a "get player setting attributes displayable text" request.
	bt_av_player_text_t	This is type bt_av_player_text_t.
	bt_av_register_notification_t	brief Parameter to AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::register_notification - when a local device received a "register notification" request.
	bt_av_response_t	brief AV/C response header ingroup avrcp details This structure is used to store fields present in every AV/C response.
	bt_av_set_absolute_volume_t	brief Parameter to AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::absolute_volume - when a local device received a response to a "set absolute volume" request.
	bt_av_set_addressed_player_t	brief Parameter to AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::addressed_player - when a local device received a response to a "set addressed player" request.
	bt_avrcp_channel_t	This is type bt_avrcp_channel_t.
	bt_avrcp_device_t	This is type bt_avrcp_device_t.
	bt_avrcp_event_t	brief Parameter to an application callback. ingroup avrcp details This union is used to pass event specific data to the AVRCP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	bt_avrcp_evt_channel_connected_t	brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_CONNECTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_connected - when a control channel between two devices has been established.

	bt_avrcp_evt_channel_disconnected_t	brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_disconnected - when a control channel between two devices has been terminated.
	bt_avrcp_evt_connection_failed_t	brief Parameter to AVRCP_EVT_CONTROL_CONNECTION_FAILED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::connection_failed - when a local device failed to create a control channel between two AVRCP entities.
	bt_avrcp_evt_panel_command_received_t	brief Parameter to AVRCP_EVT_PANEL_COMMAND_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_command_received - when a local device received a PASS THROUGH command.
	bt_avrcp_evt_panel_response_received_t	brief Parameter to AVRCP_EVT_PANEL_RESPONSE_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_response_received - when a local device received a response to a PASS THROUGH command.
	bt_avrcp_evt_register_events_completed_t	This is type bt_avrcp_evt_register_events_completed_t.
	bt_avrcp_evt_search_completed_t	brief Parameter to AVRCP_EVT_SEARCH_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::device_search - when searching for nearby devices has finished.
	bt_avrcp_find_callback_fp	brief Find Controller/Target callback. ingroup avrcp details This callback is called when search for Controller/Target has finished. param found This can be one of the following values: li c TRUE if Controller/Target was found. li c FALSE otherwise. param callback_param A pointer to an arbitrary data set by a call to bt_avrcp_find_target/bt_avrcp_find_controller.

	bt_avrcp_mgr_callback_fp	brief AVRCP application callback. ingroup avrcp details In order to be notified of various events a consumer of the AVRCP layer has to register a callback function (done with bt_avrcp_start()). The stack will call that function whenever a new event has been generated. param mgr AVRCP manager. param evt AVRCP event. The event can be one of the following values: @arg AVRCP_EVT_CONTROL_CHANNEL_CONNECTED This event is generated when a control channel between two AVRCP entities has been established. @arg AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED This event is generated when a control channel between two AVRCP entities has been terminated. @arg AVRCP_EVT_CONTROL_CONNECTION_FAILED This event is... more
	bt_avrcp_mgr_t	This is type bt_avrcp_mgr_t.
	__AVRCP_COMMAND_H	This is macro __AVRCP_COMMAND_H.
	__AVRCP_H	This is macro __AVRCP_H.
	__AVRCP_PRIVATE_H	This is macro __AVRCP_PRIVATE_H.
	AVC_BATTERY_STATUS_CRITICAL	This is macro AVC_BATTERY_STATUS_CRITICAL.
	AVC_BATTERY_STATUS_EXTERNAL	This is macro AVC_BATTERY_STATUS_EXTERNAL.
	AVC_BATTERY_STATUS_FULL_CHARGE	This is macro AVC_BATTERY_STATUS_FULL_CHARGE.
	AVC_BATTERY_STATUS_NORMAL	• addtogroup avrcp • @{ • @name Battery status
	AVC_BATTERY_STATUS_WARNING	This is macro AVC_BATTERY_STATUS_WARNING.
	AVC_CAPABILITY_COMPANY_ID	This is macro AVC_CAPABILITY_COMPANY_ID.
	AVC_CAPABILITY_EVENTS_SUPPORTED	This is macro AVC_CAPABILITY_EVENTS_SUPPORTED.
	AVC_CMD_GENERAL_POWER	This is macro AVC_CMD_GENERAL_POWER.
	AVC_CMD_GENERAL_RESERVE	This is macro AVC_CMD_GENERAL_RESERVE.
	AVC_CMD_GENERAL_SUBUNIT_INFO	This is macro AVC_CMD_GENERAL_SUBUNIT_INFO.
	AVC_CMD_GENERAL_UNIT_INFO	This is macro AVC_CMD_GENERAL_UNIT_INFO.
	AVC_CMD_GENERAL_VENDOR_DEPENDENT	This is macro AVC_CMD_GENERAL_VENDOR_DEPENDENT.
	AVC_CMD_GENERAL_VERSION	This is macro AVC_CMD_GENERAL_VERSION.
	AVC_CMD_PANEL_PASS_THROUGH	This is macro AVC_CMD_PANEL_PASS_THROUGH.
	AVC_CTYPE_CONTROL	• addtogroup avrcp • @{ • @name Command types
	AVC_CTYPE_GENERAL_INQUORY	This is macro AVC_CTYPE_GENERAL_INQUORY.
	AVC_CTYPE_NOTIFY	This is macro AVC_CTYPE_NOTIFY.
	AVC_CTYPE_SPECIFIC_IQUIRY	This is macro AVC_CTYPE_SPECIFIC_IQUIRY.
	AVC_CTYPE_STATUS	This is macro AVC_CTYPE_STATUS.
	AVC_EVENT_ADDRESSED_PLAYER_CHANGED	< The Addressed Player has been changed, see 6.9.2.
	AVC_EVENT_AVAILABLE_PLAYERS_CHANGED	< The available players have changed, see 6.9
	AVC_EVENT_BATT_STATUS_CHANGED	< Change in battery status
	AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED	This is macro AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED.

	AVC_EVENT_FLAG_ALL	This is macro AVC_EVENT_FLAG_ALL.
	AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED	This is macro AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED.
	AVC_EVENT_FLAG_BATT_STATUS_CHANGED	This is macro AVC_EVENT_FLAG_BATT_STATUS_CHANGED.
	AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED	This is macro AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED.
	AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED	This is macro AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED.
	AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED	• addtogroup avrcp • @{ • @name Notifications mask
	AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED	This is macro AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED.
	AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED	This is macro AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED.
	AVC_EVENT_FLAG_TRACK_CHANGED	This is macro AVC_EVENT_FLAG_TRACK_CHANGED.
	AVC_EVENT_FLAG_TRACK_REACHED_END	This is macro AVC_EVENT_FLAG_TRACK_REACHED_END.
	AVC_EVENT_FLAG_TRACK_REACHED_START	This is macro AVC_EVENT_FLAG_TRACK_REACHED_START.
	AVC_EVENT_FLAG_UIDS_CHANGED	This is macro AVC_EVENT_FLAG_UIDS_CHANGED.
	AVC_EVENT_FLAG_VOLUME_CHANGED	This is macro AVC_EVENT_FLAG_VOLUME_CHANGED.
	AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED	< The content of the Now Playing list has changed, see 6.9.5.
	AVC_EVENT_PLAYBACK_POS_CHANGED	< Change in playback position. Returned after the specified playback notification change notification interval
	AVC_EVENT_PLAYBACK_STATUS_CHANGED	< Change in playback status of the current track.
	AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED	< Change in player application setting
	AVC_EVENT_SYSTEM_STATUS_CHANGED	< Change in system status
	AVC_EVENT_TRACK_CHANGED	< Change of current track
	AVC_EVENT_TRACK_REACHED_END	< Reached end of a track
	AVC_EVENT_TRACK_REACHED_START	< Reached start of a track
	AVC_EVENT_UIDS_CHANGED	< The UIDs have changed, see 6.10.3.3.
	AVC_EVENT_VOLUME_CHANGED	< The volume has been changed locally on the TG, see 6.13.3
	AVC_FLAG_BROWSING_CMD	This is macro AVC_FLAG_BROWSING_CMD.
	AVC_FLAG_PANEL_CLICK	This is macro AVC_FLAG_PANEL_CLICK.
	AVC_FLAG_RESPONSE	This is macro AVC_FLAG_RESPONSE.
	AVC_MAXEVENTS	This is macro AVC_MAXEVENTS.
	AVC_MEDIA_ATTR_FLAG_ALBUM	This is macro AVC_MEDIA_ATTR_FLAG_ALBUM.
	AVC_MEDIA_ATTR_FLAG_ALL	This is macro AVC_MEDIA_ATTR_FLAG_ALL.
	AVC_MEDIA_ATTR_FLAG_ARTIST	This is macro AVC_MEDIA_ATTR_FLAG_ARTIST.
	AVC_MEDIA_ATTR_FLAG_GENRE	This is macro AVC_MEDIA_ATTR_FLAG_GENRE.
	AVC_MEDIA_ATTR_FLAG_NUMBER	This is macro AVC_MEDIA_ATTR_FLAG_NUMBER.
	AVC_MEDIA_ATTR_FLAG_PLAYING_TIME	This is macro AVC_MEDIA_ATTR_FLAG_PLAYING_TIME.
	AVC_MEDIA_ATTR_FLAG_TITLE	• addtogroup avrcp • @{ • @name Media attribute bitmask
	AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER	This is macro AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER.

	AVC_MEDIA_ATTR_ID_ALBUM	This is macro AVC_MEDIA_ATTR_ID_ALBUM.
	AVC_MEDIA_ATTR_ID_ARTIST	This is macro AVC_MEDIA_ATTR_ID_ARTIST.
	AVC_MEDIA_ATTR_ID_GENRE	This is macro AVC_MEDIA_ATTR_ID_GENRE.
	AVC_MEDIA_ATTR_ID_NUMBER	This is macro AVC_MEDIA_ATTR_ID_NUMBER.
	AVC_MEDIA_ATTR_ID_PLAYING_TIME	This is macro AVC_MEDIA_ATTR_ID_PLAYING_TIME.
	AVC_MEDIA_ATTR_ID_TITLE	• addtogroup avrcp • @{ • @name Media attribute IDs
	AVC_MEDIA_ATTR_ID_TOTAL_NUMBER	This is macro AVC_MEDIA_ATTR_ID_TOTAL_NUMBER.
	AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM	Folder Item, Media Element Item The virtual filesystem containing the Browsed media content of the browsed player
	AVC_NOW_PLAYING	Media Element Item The Now Playing list (or queue) Addressed of the addressed player
	AVC_PACKET_TYPE_CONTINUE	This is macro AVC_PACKET_TYPE_CONTINUE.
	AVC_PACKET_TYPE_END	This is macro AVC_PACKET_TYPE_END.
	AVC_PACKET_TYPE_SINGLE	This is macro AVC_PACKET_TYPE_SINGLE.
	AVC_PACKET_TYPE_START	This is macro AVC_PACKET_TYPE_START.
	AVC_PANEL_BUTTON_PRESSED	This is macro AVC_PANEL_BUTTON_PRESSED.
	AVC_PANEL_BUTTON_RELEASED	This is macro AVC_PANEL_BUTTON_RELEASED.
	AVC_PANEL_OPID_0	define AVC_PANEL_OPID_RESERVER 0x10 define AVC_PANEL_OPID_RESERVER 0x11 define AVC_PANEL_OPID_RESERVER 0x12 define AVC_PANEL_OPID_RESERVER 0x13 define AVC_PANEL_OPID_RESERVER 0x14 define AVC_PANEL_OPID_RESERVER 0x15 define AVC_PANEL_OPID_RESERVER 0x16 define AVC_PANEL_OPID_RESERVER 0x17 define AVC_PANEL_OPID_RESERVER 0x18 define AVC_PANEL_OPID_RESERVER 0x19 define AVC_PANEL_OPID_RESERVER 0x1A define AVC_PANEL_OPID_RESERVER 0x1B define AVC_PANEL_OPID_RESERVER 0x1C define AVC_PANEL_OPID_RESERVER 0x1D define AVC_PANEL_OPID_RESERVER 0x1E define AVC_PANEL_OPID_RESERVER 0x1F
	AVC_PANEL_OPID_1	This is macro AVC_PANEL_OPID_1.
	AVC_PANEL_OPID_2	This is macro AVC_PANEL_OPID_2.
	AVC_PANEL_OPID_3	This is macro AVC_PANEL_OPID_3.
	AVC_PANEL_OPID_4	This is macro AVC_PANEL_OPID_4.
	AVC_PANEL_OPID_5	This is macro AVC_PANEL_OPID_5.
	AVC_PANEL_OPID_6	This is macro AVC_PANEL_OPID_6.
	AVC_PANEL_OPID_7	This is macro AVC_PANEL_OPID_7.
	AVC_PANEL_OPID_8	This is macro AVC_PANEL_OPID_8.
	AVC_PANEL_OPID_9	This is macro AVC_PANEL_OPID_9.
	AVC_PANEL_OPID_A	This is macro AVC_PANEL_OPID_A.
	AVC_PANEL_OPID_ANGLE	define AVC_PANEL_OPID_RESERVED 0x4E define AVC_PANEL_OPID_RESERVED 0x4F
	AVC_PANEL_OPID_APPS_MENU	This is macro AVC_PANEL_OPID_APPS_MENU.
	AVC_PANEL_OPID_B	This is macro AVC_PANEL_OPID_B.
	AVC_PANEL_OPID_BACKWARD	This is macro AVC_PANEL_OPID_BACKWARD.
	AVC_PANEL_OPID_C	This is macro AVC_PANEL_OPID_C.
	AVC_PANEL_OPID_CHANNEL_DOWN	This is macro AVC_PANEL_OPID_CHANNEL_DOWN.
	AVC_PANEL_OPID_CHANNEL_UP	define AVC_PANEL_OPID_RESRVED 0x2D define AVC_PANEL_OPID_RESRVED 0x2E define AVC_PANEL_OPID_RESRVED 0x2F
	AVC_PANEL_OPID_CLEAR	This is macro AVC_PANEL_OPID_CLEAR.

	AVC_PANEL_OPID_CONTENTS_MENU	This is macro AVC_PANEL_OPID_CONTENTS_MENU.
	AVC_PANEL_OPID_D	This is macro AVC_PANEL_OPID_D.
	AVC_PANEL_OPID_DISPLAY_INFORMATION	This is macro AVC_PANEL_OPID_DISPLAY_INFORMATION.
	AVC_PANEL_OPID_DOT	This is macro AVC_PANEL_OPID_DOT.
	AVC_PANEL_OPID_DOWN	This is macro AVC_PANEL_OPID_DOWN.
	AVC_PANEL_OPID_EJECT	This is macro AVC_PANEL_OPID_EJECT.
	AVC_PANEL_OPID_ENTER	This is macro AVC_PANEL_OPID_ENTER.
	AVC_PANEL_OPID_EXIT	This is macro AVC_PANEL_OPID_EXIT.
	AVC_PANEL_OPID_F1	define AVC_PANEL_OPID_RESERVED 0x6D define AVC_PANEL_OPID_RESERVED 0x6E define AVC_PANEL_OPID_RESERVED 0x6F define AVC_PANEL_OPID_RESERVED 0x70
	AVC_PANEL_OPID_F2	This is macro AVC_PANEL_OPID_F2.
	AVC_PANEL_OPID_F3	This is macro AVC_PANEL_OPID_F3.
	AVC_PANEL_OPID_F4	This is macro AVC_PANEL_OPID_F4.
	AVC_PANEL_OPID_F5	This is macro AVC_PANEL_OPID_F5.
	AVC_PANEL_OPID_F6	This is macro AVC_PANEL_OPID_F6.
	AVC_PANEL_OPID_F7	This is macro AVC_PANEL_OPID_F7.
	AVC_PANEL_OPID_F8	This is macro AVC_PANEL_OPID_F8.
	AVC_PANEL_OPID_F9	This is macro AVC_PANEL_OPID_F9.
	AVC_PANEL_OPID_FAST_FORWARD	This is macro AVC_PANEL_OPID_FAST_FORWARD.
	AVC_PANEL_OPID_FAVORITE_MENU	This is macro AVC_PANEL_OPID_FAVORITE_MENU.
	AVC_PANEL_OPID_FORWARD	This is macro AVC_PANEL_OPID_FORWARD.
	AVC_PANEL_OPID_HELP	This is macro AVC_PANEL_OPID_HELP.
	AVC_PANEL_OPID_INPUT_SELECT	This is macro AVC_PANEL_OPID_INPUT_SELECT.
	AVC_PANEL_OPID_KEYBORD_FUNCTION	This is macro AVC_PANEL_OPID_KEYBORD_FUNCTION.
	AVC_PANEL_OPID_LEFT	This is macro AVC_PANEL_OPID_LEFT.
	AVC_PANEL_OPID_LEFT_DOWN	This is macro AVC_PANEL_OPID_LEFT_DOWN.
	AVC_PANEL_OPID_LEFT_UP	This is macro AVC_PANEL_OPID_LEFT_UP.
	AVC_PANEL_OPID_LINKED_CONTENT	This is macro AVC_PANEL_OPID_LINKED_CONTENT.
	AVC_PANEL_OPID_LIST	This is macro AVC_PANEL_OPID_LIST.
	AVC_PANEL_OPID_LIVE_TV	This is macro AVC_PANEL_OPID_LIVE_TV.
	AVC_PANEL_OPID_LOCK	This is macro AVC_PANEL_OPID_LOCK.
	AVC_PANEL_OPID_MUTE	This is macro AVC_PANEL_OPID_MUTE.
	AVC_PANEL_OPID_MUTE_FUNCTION	This is macro AVC_PANEL_OPID_MUTE_FUNCTION.
	AVC_PANEL_OPID_NEXT_DAY	This is macro AVC_PANEL_OPID_NEXT_DAY.
	AVC_PANEL_OPID_ON_DEMAND_MENU	This is macro AVC_PANEL_OPID_ON_DEMAND_MENU.
	AVC_PANEL_OPID_PAGE_DOWN	This is macro AVC_PANEL_OPID_PAGE_DOWN.
	AVC_PANEL_OPID_PAGE_UP	This is macro AVC_PANEL_OPID_PAGE_UP.
	AVC_PANEL_OPID_PAUSE	This is macro AVC_PANEL_OPID_PAUSE.
	AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION	This is macro AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION.
	AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION	This is macro AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION.
	AVC_PANEL_OPID_PIP_DOWN	This is macro AVC_PANEL_OPID_PIP_DOWN.
	AVC_PANEL_OPID_PIP_MOVE	This is macro AVC_PANEL_OPID_PIP_MOVE.
	AVC_PANEL_OPID_PIP_UP	This is macro AVC_PANEL_OPID_PIP_UP.
	AVC_PANEL_OPID_PLAY	This is macro AVC_PANEL_OPID_PLAY.

	AVC_PANEL_OPID_PLAY_FUNCTION	define AVC_PANEL_OPID_RESERVED 0x56 define AVC_PANEL_OPID_RESERVED 0x57 define AVC_PANEL_OPID_RESERVED 0x58 define AVC_PANEL_OPID_RESERVED 0x5A define AVC_PANEL_OPID_RESERVED 0x5B define AVC_PANEL_OPID_RESERVED 0x5C define AVC_PANEL_OPID_RESERVED 0x5D define AVC_PANEL_OPID_RESERVED 0x5E define AVC_PANEL_OPID_RESERVED 0x5F
	AVC_PANEL_OPID_POWER_STATE_FUNCTION	This is macro AVC_PANEL_OPID_POWER_STATE_FUNCTION.
	AVC_PANEL_OPID_POWER_TOGGLE	This is macro AVC_PANEL_OPID_POWER_TOGGLE.
	AVC_PANEL_OPID_PREVIOUS_CHANNEL	This is macro AVC_PANEL_OPID_PREVIOUS_CHANNEL.
	AVC_PANEL_OPID_PREVIOUS_DAY	This is macro AVC_PANEL_OPID_PREVIOUS_DAY.
	AVC_PANEL_OPID_RECORD	This is macro AVC_PANEL_OPID_RECORD.
	AVC_PANEL_OPID_RECORD_FUNCTION	This is macro AVC_PANEL_OPID_RECORD_FUNCTION.
	AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION	This is macro AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION.
	AVC_PANEL_OPID_REWIND	This is macro AVC_PANEL_OPID_REWIND.
	AVC_PANEL_OPID_RF_BYPASS	This is macro AVC_PANEL_OPID_RF_BYPASS.
	AVC_PANEL_OPID_RIGHT	This is macro AVC_PANEL_OPID_RIGHT.
	AVC_PANEL_OPID_RIGHT_DOWN	This is macro AVC_PANEL_OPID_RIGHT_DOWN.
	AVC_PANEL_OPID_RIGHT_UP	This is macro AVC_PANEL_OPID_RIGHT_UP.
	AVC_PANEL_OPID_ROOT_MENU	This is macro AVC_PANEL_OPID_ROOT_MENU.
	AVC_PANEL_OPID_SELECT	• addtogroup avrcp • @{ • @name AV/C Panel PASS THROUGH operation IDs
	AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION	This is macro AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION.
	AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION	This is macro AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION.
	AVC_PANEL_OPID_SELECT_DISK_FUNCTION	This is macro AVC_PANEL_OPID_SELECT_DISK_FUNCTION.
	AVC_PANEL_OPID_SETUP_MENU	This is macro AVC_PANEL_OPID_SETUP_MENU.
	AVC_PANEL_OPID_SKIP	This is macro AVC_PANEL_OPID_SKIP.
	AVC_PANEL_OPID_SOUND_SELECT	This is macro AVC_PANEL_OPID_SOUND_SELECT.
	AVC_PANEL_OPID_STOP	This is macro AVC_PANEL_OPID_STOP.
	AVC_PANEL_OPID_STOP_FUNCTION	This is macro AVC_PANEL_OPID_STOP_FUNCTION.
	AVC_PANEL_OPID_SUBPICTURE	This is macro AVC_PANEL_OPID_SUBPICTURE.
	AVC_PANEL_OPID_TUNE_FUNCTION	This is macro AVC_PANEL_OPID_TUNE_FUNCTION.
	AVC_PANEL_OPID_UP	This is macro AVC_PANEL_OPID_UP.
	AVC_PANEL_OPID_VENDOR_UNIQUE	This is macro AVC_PANEL_OPID_VENDOR_UNIQUE.
	AVC_PANEL_OPID_VOLUME_DOWN	This is macro AVC_PANEL_OPID_VOLUME_DOWN.
	AVC_PANEL_OPID_VOLUME_UP	This is macro AVC_PANEL_OPID_VOLUME_UP.
	AVC_PANEL_OPID_ZOOM	This is macro AVC_PANEL_OPID_ZOOM.
	AVC_PDUID_ABORT_CONTINUING_RESPONSE	This is macro AVC_PDUID_ABORT_CONTINUING_RESPONSE.
	AVC_PDUID_ADD_TO_NOW_PLAYING	This is macro AVC_PDUID_ADD_TO_NOW_PLAYING.

	AVC_PDUID_CHANGE_PATH	This is macro AVC_PDUID_CHANGE_PATH.
	AVC_PDUID_GENERAL_REJECT	This is macro AVC_PDUID_GENERAL_REJECT.
	AVC_PDUID_GET_CURRENT_PLAYER_APPLICATION_SETTING_VALUE	This is macro AVC_PDUID_GET_CURRENT_PLAYER_APPLICATION_SETTING_VALUE.
	AVC_PDUID_GET_ELEMENT_ATTRIBUTES	This is macro AVC_PDUID_GET_ELEMENT_ATTRIBUTES.
	AVC_PDUID_GET_FOLDER_ITEMS	This is macro AVC_PDUID_GET_FOLDER_ITEMS.
	AVC_PDUID_GET_ITEM_ATTRIBUTES	This is macro AVC_PDUID_GET_ITEM_ATTRIBUTES.
	AVC_PDUID_GET_PLAY_STATUS	This is macro AVC_PDUID_GET_PLAY_STATUS.
	AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_ATTRIBUTE_TEXT	This is macro AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_ATTRIBUTE_TEXT.
	AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_VALUE_TEXT	This is macro AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_VALUE_TEXT.
	AVC_PDUID_GETCAPABILITIES	This is macro AVC_PDUID_GETCAPABILITIES.
	AVC_PDUID_INFORM_BATTERY_STATUS_OF_CT	This is macro AVC_PDUID_INFORM_BATTERY_STATUS_OF_CT.
	AVC_PDUID_INFORM_DISPLAYABLE_CHARACTER_SET	This is macro AVC_PDUID_INFORM_DISPLAYABLE_CHARACTER_SET.
	AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_ATTRIBUTES	This is macro AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_ATTRIBUTES.
	AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_VALUES	This is macro AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_VALUES.
	AVC_PDUID_PLAY_ITEM	This is macro AVC_PDUID_PLAY_ITEM.
	AVC_PDUID_REGISTER_NOTIFICATION	This is macro AVC_PDUID_REGISTER_NOTIFICATION.
	AVC_PDUID_REQUEST_CONTINUING_RESPONSE	This is macro AVC_PDUID_REQUEST_CONTINUING_RESPONSE.
	AVC_PDUID_SEARCH	This is macro AVC_PDUID_SEARCH.
	AVC_PDUID_SET_ABSOLUTE_VOLUME	This is macro AVC_PDUID_SET_ABSOLUTE_VOLUME.
	AVC_PDUID_SET_ADDRESSED_PLAYER	This is macro AVC_PDUID_SET_ADDRESSED_PLAYER.
	AVC_PDUID_SET_BROWSED_PLAYER	This is macro AVC_PDUID_SET_BROWSED_PLAYER.
	AVC_PDUID_SET_PLAYER_APPLICATION_SETTING_VALUE	This is macro AVC_PDUID_SET_PLAYER_APPLICATION_SETTING_VALUE.
	AVC_PLAY_STATUS_ERROR	This is macro AVC_PLAY_STATUS_ERROR.
	AVC_PLAY_STATUS_FW_SEEK	This is macro AVC_PLAY_STATUS_FW_SEEK.
	AVC_PLAY_STATUS_PAUSED	This is macro AVC_PLAY_STATUS_PAUSED.
	AVC_PLAY_STATUS_PLAYING	This is macro AVC_PLAY_STATUS_PLAYING.
	AVC_PLAY_STATUS_REV_SEEK	This is macro AVC_PLAY_STATUS_REV_SEEK.
	AVC_PLAY_STATUS_STOPPED	• addtogroup avrcp • @{ • @name Play status
	AVC_PLAYER_SETTING_EQUALIZER_OFF	This is macro AVC_PLAYER_SETTING_EQUALIZER_OFF.
	AVC_PLAYER_SETTING_EQUALIZER_ON	This is macro AVC_PLAYER_SETTING_EQUALIZER_ON.
	AVC_PLAYER_SETTING_EQUALIZER_STATUS	This is macro AVC_PLAYER_SETTING_EQUALIZER_STATUS.
	AVC_PLAYER_SETTING_REPEAT_ALL_TRACKS	This is macro AVC_PLAYER_SETTING_REPEAT_ALL_TRACKS.

	AVC_PLAYER_SETTING_REPEAT_GROUP	This is macro AVC_PLAYER_SETTING_REPEAT_GROUP.
	AVC_PLAYER_SETTING_REPEAT_MODE_OFF	This is macro AVC_PLAYER_SETTING_REPEAT_MODE_OFF.
	AVC_PLAYER_SETTING_REPEAT_MODE_STATUS	This is macro AVC_PLAYER_SETTING_REPEAT_MODE_STATUS.
	AVC_PLAYER_SETTING_REPEAT_SINGLE_TRACK	This is macro AVC_PLAYER_SETTING_REPEAT_SINGLE_TRACK.
	AVC_PLAYER_SETTING_SCAN_ALL_TRACKS	This is macro AVC_PLAYER_SETTING_SCAN_ALL_TRACKS.
	AVC_PLAYER_SETTING_SCAN_GROUP	This is macro AVC_PLAYER_SETTING_SCAN_GROUP.
	AVC_PLAYER_SETTING_SCAN_OFF	This is macro AVC_PLAYER_SETTING_SCAN_OFF.
	AVC_PLAYER_SETTING_SCAN_STATUS	This is macro AVC_PLAYER_SETTING_SCAN_STATUS.
	AVC_PLAYER_SETTING_SHUFFLE_ALL_TRACKS	This is macro AVC_PLAYER_SETTING_SHUFFLE_ALL_TRACKS.
	AVC_PLAYER_SETTING_SHUFFLE_GROUP	This is macro AVC_PLAYER_SETTING_SHUFFLE_GROUP.
	AVC_PLAYER_SETTING_SHUFFLE_OFF	This is macro AVC_PLAYER_SETTING_SHUFFLE_OFF.
	AVC_PLAYER_SETTING_SHUFFLE_STATUS	This is macro AVC_PLAYER_SETTING_SHUFFLE_STATUS.
	AVC_RESPONSE_ACCEPTED	This is macro AVC_RESPONSE_ACCEPTED.
	AVC_RESPONSE_CHANGED	This is macro AVC_RESPONSE_CHANGED.
	AVC_RESPONSE_IMPLEMENTED	This is macro AVC_RESPONSE_IMPLEMENTED.
	AVC_RESPONSE_IN_TRANSITION	This is macro AVC_RESPONSE_IN_TRANSITION.
	AVC_RESPONSE_INTERIM	This is macro AVC_RESPONSE_INTERIM.
	AVC_RESPONSE_NOT_IMPLEMENTED	• addtogroup avrcp • @{ • @name Response types
	AVC_RESPONSE_REJECTED	This is macro AVC_RESPONSE_REJECTED.
	AVC_RESPONSE_STABLE	This is macro AVC_RESPONSE_STABLE.
	AVC_RESPONSE_TIMEOUT	This is macro AVC_RESPONSE_TIMEOUT.
	AVC_SCOPE_MEDIA_PLAYER_LIST	Media Player Item Contains all available media players None
	AVC_SEARCH	Media Element Item The results of a search operation Browsed on the browsed player
	AVC_SUBUNIT_ID_EXTENDED_TO_NEXT_BYTE	This is macro AVC_SUBUNIT_ID_EXTENDED_TO_NEXT_BYTE.
	AVC_SUBUNIT_ID_IGNORE	This is macro AVC_SUBUNIT_ID_IGNORE.
	AVC_SUBUNIT_TYPE_AUDIO	This is macro AVC_SUBUNIT_TYPE_AUDIO.
	AVC_SUBUNIT_TYPE_BULLETIN_BOARD	This is macro AVC_SUBUNIT_TYPE_BULLETIN_BOARD.
	AVC_SUBUNIT_TYPE_CA	This is macro AVC_SUBUNIT_TYPE_CA.
	AVC_SUBUNIT_TYPE_CAMERA	This is macro AVC_SUBUNIT_TYPE_CAMERA.
	AVC_SUBUNIT_TYPE_CAMERA_STORAGE	This is macro AVC_SUBUNIT_TYPE_CAMERA_STORAGE.
	AVC_SUBUNIT_TYPE_DISC	This is macro AVC_SUBUNIT_TYPE_DISC.
	AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE	This is macro AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE.
	AVC_SUBUNIT_TYPE_MONITOR	• addtogroup avrcp • @{ • @name Subunit types
	AVC_SUBUNIT_TYPE_PANEL	This is macro AVC_SUBUNIT_TYPE_PANEL.
	AVC_SUBUNIT_TYPE_PRINTER	This is macro AVC_SUBUNIT_TYPE_PRINTER.

	AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER	This is macro AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER.
	AVC_SUBUNIT_TYPE_TUNER	This is macro AVC_SUBUNIT_TYPE_TUNER.
	AVC_SUBUNIT_TYPE_UNIT	This is macro AVC_SUBUNIT_TYPE_UNIT.
	AVC_SUBUNIT_TYPE_VENDOR_UNIQUE	This is macro AVC_SUBUNIT_TYPE_VENDOR_UNIQUE.
	AVC_VOLUME_MAX	100
	AVC_VOLUME_MIN	0
	AVRCP_BTSIG_COMPANY_ID	This is macro AVRCP_BTSIG_COMPANY_ID.
	AVRCP_CHANNEL_FLAG_LISTENING	This is macro AVRCP_CHANNEL_FLAG_LISTENING.
	AVRCP_CHANNEL_FLAG_PLAY_STATUS_REQUESTED	This is macro AVRCP_CHANNEL_FLAG_PLAY_STATUS_REQUESTED.
	AVRCP_CHANNEL_FLAG_REGISTERING_NOTIFICATIONS	This is macro AVRCP_CHANNEL_FLAG_REGISTERING_NOTIFICATIONS.
	AVRCP_CHANNEL_FLAG_SENDING	This is macro AVRCP_CHANNEL_FLAG_SENDING.
	AVRCP_CHANNEL_STATE_CONNECTED	This is macro AVRCP_CHANNEL_STATE_CONNECTED.
	AVRCP_CHANNEL_STATE_CONNECTING	This is macro AVRCP_CHANNEL_STATE_CONNECTING.
	AVRCP_CHANNEL_STATE_DISCONNECTING	This is macro AVRCP_CHANNEL_STATE_DISCONNECTING.
	AVRCP_CHANNEL_STATE_FREE	This is macro AVRCP_CHANNEL_STATE_FREE.
	AVRCP_CHANNEL_STATE_IDLE	This is macro AVRCP_CHANNEL_STATE_IDLE.
	AVRCP_COMMAND_TYPE_BROWSING	This is macro AVRCP_COMMAND_TYPE_BROWSING.
	AVRCP_COMMAND_TYPE_CONTROL	This is macro AVRCP_COMMAND_TYPE_CONTROL.
	AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED	< This event is generated when a local device received a response to a "add to now playing" request.
	AVRCP_EVT_ADDRESSED_PLAYER_CHANGED	< This event is generated when a local device received a "addressed player changed" notification.
	AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED	< This event is generated when a local device received a "available players changed" notification.
	AVRCP_EVT_BATT_STATUS_CHANGED	< This event is generated when a local device received a "battery status changed" notification.
	AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED	< This event is generated when a local device received a "battery status of controller" command.
	AVRCP_EVT_BROWSING_CHANNEL_CONNECTED	< This event is generated when a browsing channel between two AVRCP entities has been established.
	AVRCP_EVT_BROWSING_CHANNEL_DISCONNECTED	< This event is generated when a browsing channel between two AVRCP entities has been terminated.
	AVRCP_EVT_BROWSING_CONNECTION_FAILED	< This event is generated when a local device failed to create a browsing channel between two AVRCP entities.
	AVRCP_EVT_COMPANY_ID_LIST_RECEIVED	< This event is generated when a local device received a response to a "get company id" request.
	AVRCP_EVT_CONTROL_CHANNEL_CONNECTED	< This event is generated when a control channel between two AVRCP entities has been established.
	AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED	< This event is generated when a control channel between two AVRCP entities has been terminated.
	AVRCP_EVT_CONTROL_CONNECTION_FAILED	< This event is generated when a local device failed to create a control channel between two AVRCP entities.
	AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED	< This event is generated when a local device received a "displayable chracter set command" request.

	AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED	< This event is generated when a local device received a "get element attributes" request.
	AVRCP_EVT_EVENT_ID_LIST_RECEIVED	< This event is generated when a local device received a response to a "get supported events" request.
	AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED	< This event is generated when a local device received a response to a "get media element attributes" request.
	AVRCP_EVT_GET_PLAY_STATUS_RECEIVED	< This event is generated when a local device received a response to a "get play status" request.
	AVRCP_EVT_INFORM_BATTERY_STATUS_OF_CT_COMPLETED	< This event is generated when a local device received a response to a "inform battery status" request.
	AVRCP_EVT_INFORM_DISPLAYABLE_CHARACTER_SET_COMPLETED	< This event is generated when a local device received a response to a "inform displayable character set" request.
	AVRCP_EVT_NOTHING	addtogroup avrcp @{ @name Events details The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.
	AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED	< This event is generated when a local device received a "now playing content changed" notification.
	AVRCP_EVT_PANEL_COMMAND_RECEIVED	< This event is generated when a local device received a PASS THROUGH command.
	AVRCP_EVT_PANEL_RESPONSE_RECEIVED	< This event is generated when a local device received a response to a PASS THROUGH command.
	AVRCP_EVT_PLAY_ITEM_COMPLETED	< This event is generated when a local device received a response to a "play item" request.
	AVRCP_EVT_PLAYBACK_POS_CHANGED	< This event is generated when a local device received a "playback position changed" notification.
	AVRCP_EVT_PLAYBACK_STATUS_CHANGED	< This event is generated when a local device received a "play status changed" notification.
	AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED	< This event is generated when a local device received a "player application setting changed" notification.
	AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED	< This event is generated when a local device received a response to a "get current player setting attribute values" request.
	AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED	< This event is generated when a local device received a response to a "get supported player setting attributes" request.
	AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED	< This event is generated when a local device received a response to a "get player setting attributes displayable text" request.
	AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED	< This event is generated when a local device received a response to a "get player setting attribute values" request.
	AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED	< This event is generated when a local device received a response to a "get player setting attribute values displayable text" request.
	AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED	< This event is generated when a local device received a "register notification" request.
	AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED	< This event is generated when a local device received a response to a "register notification" request.
	AVRCP_EVT_SEARCH_COMPLETED	< This event is generated when a local device completed searching for nearby targets.
	AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED	< This event is generated when a local device received a response to a "set absolute volume" request.
	AVRCP_EVT_SET_ABSOLUTE_VOLUME_REQUESTED	< This event is generated when a local device received a "set absolute volume" request.

	AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED	< This event is generated when a local device received a response to a "set addressed player" request.
	AVRCP_EVT_SET_PLAYER_SETTING_VALUES_COMPLETED	< This event is generated when a local device received a response to a "set player setting attribute values" request.
	AVRCP_EVT_SYSTEM_STATUS_CHANGED	< This event is generated when a local device received a "system status changed" notification.
	AVRCP_EVT_TRACK_CHANGED	< This event is generated when a local device received a "track changed changed" notification.
	AVRCP_EVT_TRACK_REACHED_END	< This event is generated when a local device received a "track reached end" notification.
	AVRCP_EVT_TRACK_REACHED_START	< This event is generated when a local device received a "track reached start" notification.
	AVRCP_EVT_UIDS_CHANGED	< This event is generated when a local device received a "UIDs changed" notification.
	AVRCP_EVT_VOLUME_CHANGED	< This event is generated when a local device received a "volume changed" notification.
	AVRCP_MANAGER_FLAG_SEARCHING	This is macro AVRCP_MANAGER_FLAG_SEARCHING.
	AVRCP_MANAGER_STATE_IDLE	This is macro AVRCP_MANAGER_STATE_IDLE.
	AVRCP_MAX_ELEMENT_ATTRIBUTES	This is macro AVRCP_MAX_ELEMENT_ATTRIBUTES.
	bt_avrcp_0_click	This is macro bt_avrcp_0_click.
	bt_avrcp_0_press	This is macro bt_avrcp_0_press.
	bt_avrcp_1_click	This is macro bt_avrcp_1_click.
	bt_avrcp_1_press	This is macro bt_avrcp_1_press.
	bt_avrcp_2_click	This is macro bt_avrcp_2_click.
	bt_avrcp_2_press	This is macro bt_avrcp_2_press.
	bt_avrcp_3_click	This is macro bt_avrcp_3_click.
	bt_avrcp_3_press	This is macro bt_avrcp_3_press.
	bt_avrcp_4_click	This is macro bt_avrcp_4_click.
	bt_avrcp_4_press	This is macro bt_avrcp_4_press.
	bt_avrcp_5_click	This is macro bt_avrcp_5_click.
	bt_avrcp_5_press	This is macro bt_avrcp_5_press.
	bt_avrcp_6_click	This is macro bt_avrcp_6_click.
	bt_avrcp_6_press	This is macro bt_avrcp_6_press.
	bt_avrcp_7_click	This is macro bt_avrcp_7_click.
	bt_avrcp_7_press	This is macro bt_avrcp_7_press.
	bt_avrcp_8_click	This is macro bt_avrcp_8_click.
	bt_avrcp_8_press	This is macro bt_avrcp_8_press.
	bt_avrcp_9_click	This is macro bt_avrcp_9_click.
	bt_avrcp_9_press	This is macro bt_avrcp_9_press.
	bt_avrcp_angle_click	This is macro bt_avrcp_angle_click.
	bt_avrcp_angle_press	This is macro bt_avrcp_angle_press.
	bt_avrcp_backward_click	This is macro bt_avrcp_backward_click.
	bt_avrcp_backward_press	This is macro bt_avrcp_backward_press.
	bt_avrcp_channel_down_click	This is macro bt_avrcp_channel_down_click.
	bt_avrcp_channel_down_press	This is macro bt_avrcp_channel_down_press.
	bt_avrcp_channel_up_click	This is macro bt_avrcp_channel_up_click.
	bt_avrcp_channel_up_press	This is macro bt_avrcp_channel_up_press.
	bt_avrcp_clear_click	This is macro bt_avrcp_clear_click.
	bt_avrcp_clear_press	This is macro bt_avrcp_clear_press.
	bt_avrcp_content_menu_click	This is macro bt_avrcp_content_menu_click.
	bt_avrcp_contents_menu_press	This is macro bt_avrcp_contents_menu_press.
	bt_avrcp_display_info_click	This is macro bt_avrcp_display_info_click.
	bt_avrcp_display_info_press	This is macro bt_avrcp_display_info_press.
	bt_avrcp_dot_click	This is macro bt_avrcp_dot_click.

	bt_avrcp_dot_press	This is macro bt_avrcp_dot_press.
	bt_avrcp_down_click	This is macro bt_avrcp_down_click.
	bt_avrcp_down_press	This is macro bt_avrcp_down_press.
	bt_avrcp_eject_click	This is macro bt_avrcp_eject_click.
	bt_avrcp_eject_press	This is macro bt_avrcp_eject_press.
	bt_avrcp_enter_click	This is macro bt_avrcp_enter_click.
	bt_avrcp_enter_press	This is macro bt_avrcp_enter_press.
	bt_avrcp_exit_click	This is macro bt_avrcp_exit_click.
	bt_avrcp_exit_press	This is macro bt_avrcp_exit_press.
	bt_avrcp_f1_click	This is macro bt_avrcp_f1_click.
	bt_avrcp_f1_press	This is macro bt_avrcp_f1_press.
	bt_avrcp_f2_click	This is macro bt_avrcp_f2_click.
	bt_avrcp_f2_press	This is macro bt_avrcp_f2_press.
	bt_avrcp_f3_click	This is macro bt_avrcp_f3_click.
	bt_avrcp_f3_press	This is macro bt_avrcp_f3_press.
	bt_avrcp_f4_click	This is macro bt_avrcp_f4_click.
	bt_avrcp_f4_press	This is macro bt_avrcp_f4_press.
	bt_avrcp_f5_click	This is macro bt_avrcp_f5_click.
	bt_avrcp_f5_press	This is macro bt_avrcp_f5_press.
	bt_avrcp_f6_click	This is macro bt_avrcp_f6_click.
	bt_avrcp_f6_press	This is macro bt_avrcp_f6_press.
	bt_avrcp_f7_click	This is macro bt_avrcp_f7_click.
	bt_avrcp_f7_press	This is macro bt_avrcp_f7_press.
	bt_avrcp_f8_click	This is macro bt_avrcp_f8_click.
	bt_avrcp_f8_press	This is macro bt_avrcp_f8_press.
	bt_avrcp_f9_click	This is macro bt_avrcp_f9_click.
	bt_avrcp_f9_press	This is macro bt_avrcp_f9_press.
	bt_avrcp_fast_forward_click	This is macro bt_avrcp_fast_forward_click.
	bt_avrcp_fast_forward_press	This is macro bt_avrcp_fast_forward_press.
	bt_avrcp_favorite_menu_click	This is macro bt_avrcp_favorite_menu_click.
	bt_avrcp_favorive_menu_press	This is macro bt_avrcp_favorive_menu_press.
	bt_avrcp_forward_click	This is macro bt_avrcp_forward_click.
	bt_avrcp_forward_press	This is macro bt_avrcp_forward_press.
	bt_avrcp_help_click	This is macro bt_avrcp_help_click.
	bt_avrcp_help_press	This is macro bt_avrcp_help_press.
	bt_avrcp_input_select_click	This is macro bt_avrcp_input_select_click.
	bt_avrcp_input_select_press	This is macro bt_avrcp_input_select_press.
	bt_avrcp_left_click	This is macro bt_avrcp_left_click.
	bt_avrcp_left_down_click	This is macro bt_avrcp_left_down_click.
	bt_avrcp_left_down_press	This is macro bt_avrcp_left_down_press.
	bt_avrcp_left_press	This is macro bt_avrcp_left_press.
	bt_avrcp_left_up_click	This is macro bt_avrcp_left_up_click.
	bt_avrcp_left_up_press	This is macro bt_avrcp_left_up_press.
	bt_avrcp_mute_click	This is macro bt_avrcp_mute_click.
	bt_avrcp_mute_press	This is macro bt_avrcp_mute_press.
	bt_avrcp_page_down_click	This is macro bt_avrcp_page_down_click.
	bt_avrcp_page_down_press	This is macro bt_avrcp_page_down_press.
	bt_avrcp_page_up_click	This is macro bt_avrcp_page_up_click.
	bt_avrcp_page_up_press	This is macro bt_avrcp_page_up_press.
	bt_avrcp_pause_click	This is macro bt_avrcp_pause_click.
	bt_avrcp_pause_press	This is macro bt_avrcp_pause_press.
	bt_avrcp_play_click	This is macro bt_avrcp_play_click.
	bt_avrcp_play_press	This is macro bt_avrcp_play_press.
	bt_avrcp_power_click	This is macro bt_avrcp_power_click.
	bt_avrcp_power_press	This is macro bt_avrcp_power_press.

	bt_avrcp_previous_channel_click	This is macro bt_avrcp_previous_channel_click.
	bt_avrcp_previous_channel_press	This is macro bt_avrcp_previous_channel_press.
	bt_avrcp_record_click	This is macro bt_avrcp_record_click.
	bt_avrcp_record_press	This is macro bt_avrcp_record_press.
	bt_avrcp_rewind_click	This is macro bt_avrcp_rewind_click.
	bt_avrcp_rewind_press	This is macro bt_avrcp_rewind_press.
	bt_avrcp_right_click	This is macro bt_avrcp_right_click.
	bt_avrcp_right_down_click	This is macro bt_avrcp_right_down_click.
	bt_avrcp_right_down_press	This is macro bt_avrcp_right_down_press.
	bt_avrcp_right_press	This is macro bt_avrcp_right_press.
	bt_avrcp_right_up_click	This is macro bt_avrcp_right_up_click.
	bt_avrcp_right_up_press	This is macro bt_avrcp_right_up_press.
	bt_avrcp_root_menu_click	This is macro bt_avrcp_root_menu_click.
	bt_avrcp_root_menu_press	This is macro bt_avrcp_root_menu_press.
	bt_avrcp_select_click	This is macro bt_avrcp_select_click.
	bt_avrcp_select_press	Panel operations
	bt_avrcp_setup_menu_click	This is macro bt_avrcp_setup_menu_click.
	bt_avrcp_setup_menu_press	This is macro bt_avrcp_setup_menu_press.
	bt_avrcp_sound_select_click	This is macro bt_avrcp_sound_select_click.
	bt_avrcp_sound_select_press	This is macro bt_avrcp_sound_select_press.
	bt_avrcp_stop_click	This is macro bt_avrcp_stop_click.
	bt_avrcp_stop_press	This is macro bt_avrcp_stop_press.
	bt_avrcp_subpicture_click	This is macro bt_avrcp_subpicture_click.
	bt_avrcp_subpicture_press	This is macro bt_avrcp_subpicture_press.
	bt_avrcp_up_click	This is macro bt_avrcp_up_click.
	bt_avrcp_up_press	This is macro bt_avrcp_up_press.
	bt_avrcp_volume_down_click	This is macro bt_avrcp_volume_down_click.
	bt_avrcp_volume_down_press	This is macro bt_avrcp_volume_down_press.
	bt_avrcp_volume_up_click	This is macro bt_avrcp_volume_up_click.
	bt_avrcp_volume_up_press	This is macro bt_avrcp_volume_up_press.
	bt_avrcp_0_release	This is macro bt_avrcp_0_release.
	bt_avrcp_1_release	This is macro bt_avrcp_1_release.
	bt_avrcp_2_release	This is macro bt_avrcp_2_release.
	bt_avrcp_3_release	This is macro bt_avrcp_3_release.
	bt_avrcp_4_release	This is macro bt_avrcp_4_release.
	bt_avrcp_5_release	This is macro bt_avrcp_5_release.
	bt_avrcp_6_release	This is macro bt_avrcp_6_release.
	bt_avrcp_7_release	This is macro bt_avrcp_7_release.
	bt_avrcp_8_release	This is macro bt_avrcp_8_release.
	bt_avrcp_9_release	This is macro bt_avrcp_9_release.
	bt_avrcp_angle_release	This is macro bt_avrcp_angle_release.
	bt_avrcp_backward_release	This is macro bt_avrcp_backward_release.
	bt_avrcp_channel_down_release	This is macro bt_avrcp_channel_down_release.
	bt_avrcp_channel_up_release	This is macro bt_avrcp_channel_up_release.
	bt_avrcp_clear_release	This is macro bt_avrcp_clear_release.
	bt_avrcp_content_menu_release	This is macro bt_avrcp_content_menu_release.
	bt_avrcp_display_info_release	This is macro bt_avrcp_display_info_release.
	bt_avrcp_dot_release	This is macro bt_avrcp_dot_release.
	bt_avrcp_down_release	This is macro bt_avrcp_down_release.
	bt_avrcp_eject_release	This is macro bt_avrcp_eject_release.
	bt_avrcp_enter_release	This is macro bt_avrcp_enter_release.
	bt_avrcp_exit_release	This is macro bt_avrcp_exit_release.
	bt_avrcp_f1_release	This is macro bt_avrcp_f1_release.
	bt_avrcp_f2_release	This is macro bt_avrcp_f2_release.
	bt_avrcp_f3_release	This is macro bt_avrcp_f3_release.

	bt_avrcp_f4_release	This is macro bt_avrcp_f4_release.
	bt_avrcp_f5_release	This is macro bt_avrcp_f5_release.
	bt_avrcp_f6_release	This is macro bt_avrcp_f6_release.
	bt_avrcp_f7_release	This is macro bt_avrcp_f7_release.
	bt_avrcp_f8_release	This is macro bt_avrcp_f8_release.
	bt_avrcp_f9_release	This is macro bt_avrcp_f9_release.
	bt_avrcp_fast_forward_release	This is macro bt_avrcp_fast_forward_release.
	bt_avrcp_favorite_menu_release	This is macro bt_avrcp_favorite_menu_release.
	bt_avrcp_forward_release	This is macro bt_avrcp_forward_release.
	bt_avrcp_help_release	This is macro bt_avrcp_help_release.
	bt_avrcp_input_select_release	This is macro bt_avrcp_input_select_release.
	bt_avrcp_left_down_release	This is macro bt_avrcp_left_down_release.
	bt_avrcp_left_release	This is macro bt_avrcp_left_release.
	bt_avrcp_left_up_release	This is macro bt_avrcp_left_up_release.
	bt_avrcp_mute_release	This is macro bt_avrcp_mute_release.
	bt_avrcp_page_down_release	This is macro bt_avrcp_page_down_release.
	bt_avrcp_page_up_release	This is macro bt_avrcp_page_up_release.
	bt_avrcp_pause_release	This is macro bt_avrcp_pause_release.
	bt_avrcp_play_release	This is macro bt_avrcp_play_release.
	bt_avrcp_power_release	This is macro bt_avrcp_power_release.
	bt_avrcp_previous_channel_release	This is macro bt_avrcp_previous_channel_release.
	bt_avrcp_record_release	This is macro bt_avrcp_record_release.
	bt_avrcp_rewind_release	This is macro bt_avrcp_rewind_release.
	bt_avrcp_right_down_release	This is macro bt_avrcp_right_down_release.
	bt_avrcp_right_release	This is macro bt_avrcp_right_release.
	bt_avrcp_right_up_release	This is macro bt_avrcp_right_up_release.
	bt_avrcp_root_menu_release	This is macro bt_avrcp_root_menu_release.
	bt_avrcp_select_release	This is macro bt_avrcp_select_release.
	bt_avrcp_setup_menu_release	This is macro bt_avrcp_setup_menu_release.
	bt_avrcp_sound_select_release	This is macro bt_avrcp_sound_select_release.
	bt_avrcp_stop_release	This is macro bt_avrcp_stop_release.
	bt_avrcp_subpicture_release	This is macro bt_avrcp_subpicture_release.
	bt_avrcp_up_release	This is macro bt_avrcp_up_release.
	bt_avrcp_volume_down_release	This is macro bt_avrcp_volume_down_release.
	bt_avrcp_volume_up_release	This is macro bt_avrcp_volume_up_release.
	__AVRCP_CONFIG_EVENT_HANDLERS_H	This is macro __AVRCP_CONFIG_EVENT_HANDLERS_H.
	AVRCP_COMMAND_HANDLER	This is macro AVRCP_COMMAND_HANDLER.
	AVRCP_COMMAND_SENT_HANDLER	This is macro AVRCP_COMMAND_SENT_HANDLER.
	AVRCP_RESPONSE_HANDLER	This is macro AVRCP_RESPONSE_HANDLER.
	AVRCP_RESPONSE_SENT_HANDLER	This is macro AVRCP_RESPONSE_SENT_HANDLER.

AVRCP Functions

	Name	Description
	_bt_avrcp_allocate_browsing_cmd	This is function _bt_avrcp_allocate_browsing_cmd.
	_bt_avrcp_allocate_bt_specific_cmd	This is function _bt_avrcp_allocate_bt_specific_cmd.
	_bt_avrcp_allocate_bt_specific_response	This is function _bt_avrcp_allocate_bt_specific_response.
	_bt_avrcp_allocate_channel	void _bt_avctp_l2cap_read_data_callback(struct _channel *pch, bt_byte* pdata, bt_int len); bt_avctp_channel_t* _bt_avctp_find_channel(bt_avctp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_uint profile_id);
	_bt_avrcp_allocate_cmd	This is function _bt_avrcp_allocate_cmd.
	_bt_avrcp_allocate_response	This is function _bt_avrcp_allocate_response.
	_bt_avrcp_allocate_simple_panel_cmd	This is function _bt_avrcp_allocate_simple_panel_cmd.
	_bt_avrcp_allocate_simple_panel_response	This is function _bt_avrcp_allocate_simple_panel_response.

	_bt_avrcp_find_channel	This is function _bt_avrcp_find_channel.
	_bt_avrcp_free_channel	This is function _bt_avrcp_free_channel.
	_bt_avrcp_free_cmd	This is function _bt_avrcp_free_cmd.
	_bt_avrcp_get_tick_count	This is function _bt_avrcp_get_tick_count.
	_bt_avrcp_handle_add_to_now_playing	This is function _bt_avrcp_handle_add_to_now_playing.
	_bt_avrcp_handle_command	This is function _bt_avrcp_handle_command.
	_bt_avrcp_handle_command_sent	This is function _bt_avrcp_handle_command_sent.
	_bt_avrcp_handle_get_capabilities	This is function _bt_avrcp_handle_get_capabilities.
	_bt_avrcp_handle_get_current_player_application_setting_value	This is function _bt_avrcp_handle_get_current_player_application_setting_value.
	_bt_avrcp_handle_get_element_attributes	This is function _bt_avrcp_handle_get_element_attributes.
	_bt_avrcp_handle_get_play_status	This is function _bt_avrcp_handle_get_play_status.
	_bt_avrcp_handle_get_player_application_setting_attribute_text	This is function _bt_avrcp_handle_get_player_application_setting_attribute_text.
	_bt_avrcp_handle_get_player_application_setting_value_text	This is function _bt_avrcp_handle_get_player_application_setting_value_text.
	_bt_avrcp_handle_inform_battery_status_of_ct	This is function _bt_avrcp_handle_inform_battery_status_of_ct.
	_bt_avrcp_handle_inform_displayable_character_set	This is function _bt_avrcp_handle_inform_displayable_character_set.
	_bt_avrcp_handle_list_player_application_setting_attributes	This is function _bt_avrcp_handle_list_player_application_setting_attributes.
	_bt_avrcp_handle_list_player_application_setting_values	This is function _bt_avrcp_handle_list_player_application_setting_values.
	_bt_avrcp_handle_play_item	This is function _bt_avrcp_handle_play_item.
	_bt_avrcp_handle_register_notification	This is function _bt_avrcp_handle_register_notification.
	_bt_avrcp_handle_request_continuing_response	This is function _bt_avrcp_handle_request_continuing_response.
	_bt_avrcp_handle_response	This is function _bt_avrcp_handle_response.
	_bt_avrcp_handle_response_sent	This is function _bt_avrcp_handle_response_sent.
	_bt_avrcp_handle_set_absolute_volume	This is function _bt_avrcp_handle_set_absolute_volume.
	_bt_avrcp_handle_set_addressed_player	This is function _bt_avrcp_handle_set_addressed_player.
	_bt_avrcp_handle_set_player_application_setting_value	This is function _bt_avrcp_handle_set_player_application_setting_value.
	_bt_avrcp_init_signal	void _bt_avctp_init_message_buffers();
	_bt_avrcp_init_timer	This is function _bt_avrcp_init_timer.
	_bt_avrcp_register_next_notification	This is function _bt_avrcp_register_next_notification.
	_bt_avrcp_register_pending_notification	This is function _bt_avrcp_register_pending_notification.
	_bt_avrcp_send_notifications	This is function _bt_avrcp_send_notifications.
	_bt_avrcp_set_signal	This is function _bt_avrcp_set_signal.
	_bt_avrcp_start_timer	This is function _bt_avrcp_start_timer.
	_bt_avrcp_tg_handle_get_capabilities	This is function _bt_avrcp_tg_handle_get_capabilities.
	_bt_avrcp_tg_handle_get_element_attributes	This is function _bt_avrcp_tg_handle_get_element_attributes.
	_bt_avrcp_tg_handle_get_play_status	This is function _bt_avrcp_tg_handle_get_play_status.
	_bt_avrcp_tg_handle_inform_battery_status_of_ct	This is function _bt_avrcp_tg_handle_inform_battery_status_of_ct.
	_bt_avrcp_tg_handle_inform_displayable_character_set	This is function _bt_avrcp_tg_handle_inform_displayable_character_set.
	_bt_avrcp_tg_handle_register_notification	This is function _bt_avrcp_tg_handle_register_notification.
	_bt_avrcp_tg_handle_set_absolute_volume	This is function _bt_avrcp_tg_handle_set_absolute_volume.
	_bt_avrcp_write_command_header	This is function _bt_avrcp_write_command_header.
	bt_avrcp_abort_continuing_response	This is function bt_avrcp_abort_continuing_response.

	bt_avrcp_add_to_now_playing	brief Add to "now playing" list. ingroup avrcp details This function adds a media element specified by p element_id to the "now playing" list on the target. param channel AVRCP channel. param scope The scope in which the p element_id is valid. This value can be on the following values: li AVC_SCOPE_MEDIA_PLAYER_LIST li AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM li AVC_SEARCH li AVC_NOW_PLAYING param element_id UID of the media element to be added to the "now playing" list. param counter UID counter. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_avrcpt_request_continuing_response	This is function bt_avrcp_avrcpt_request_continuing_response.
	bt_avrcp_cancel_find	brief Cancel finding Targets ingroup avrcp details This function stops AVRCP layer looking for targets on nearby devices. As a result of this operation the AVRCP_EVT_SEARCH_COMPLETED event will be generated. return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.
	bt_avrcp_cancel_listen	brief Cancel listening for incoming connections. ingroup avrcp details This function stops listening for incoming connections on a specified channel. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_connect	brief Connect to a remote device. ingroup avrcp details This function establishes a connection to a remote device specified by the p remote_address. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVRCP callback. The events generated will either be AVRCP_EVT_CONTROL_CHANNEL_CONNECTED or AVRCP_EVT_CONTROL_CONNECTION_FAILED. param channel AVRCP channel. param remote_address The address of a remote device. return li c TRUE if connection establishment has been started. li c FALSE otherwise.... more
	bt_avrcp_create_channel	brief Allocate AVRCP channel ingroup avrcp details This function allocates a new incoming AVRCP channel. The channel is intended to be used to accept a connection from a remote device. There can be only one incoming channel. param mgr AVRCP manager. param create_browsing_channel Defines weather a browsing channel will be created. return li A pointer to the new AVRCP channel if the function succeeds. li c NULL otherwise.
	bt_avrcp_create_outgoing_channel	brief Allocate AVRCP channel ingroup avrcp details This function allocates a new outgoing AVRCP channel. The channel is intended to be used to create a connection to a remote device. There can be multiple outgoing channels. param mgr AVRCP manager. param create_browsing_channel Defines weather a browsing channel will be created. return li A pointer to the new AVRCP channel if the function succeeds. li c NULL otherwise.
	bt_avrcp_destroy_channel	brief Destroy AVRCP channel. ingroup avrcp details This function frees memory used by the channel. The channel has to exist and be in the "idle" state for this function to succeed. I.e. the channel has to be disconnected before this function can be called. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_disconnect	brief Disconnect from a remote device. ingroup avrcp details This function closes a connection to a remote device. param channel AVRCP channel. return li c TRUE if disconnection has been started. li c FALSE otherwise. No events will be generated.

	bt_avrcp_find_controller	brief Find Controller ingroup avrcp details This function looks for a controller on a remote device specified by p deviceAddress. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer to arbitrary data to be passed to the p... more
	bt_avrcp_find_target	brief Find Target ingroup avrcp details This function looks for a target on a remote device specified by p deviceAddress. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer to arbitrary data to be passed to the p... more
	bt_avrcp_find_targets	brief Find Targets ingroup avrcp details This function looks for targets on nearby devices. The AVRCP_EVT_SEARCH_COMPLETED event is generated when the search has completed. param search_length The amount of time the search will be performed for. return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.
	bt_avrcp_get_browsing_channel_state	brief Get AVCTP browsing channel state. ingroup avrcp details This function returns status of the AVCTP browsing channel. param channel AVRCP channel. return Returns of the following values: li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING
	bt_avrcp_get_channel_remote_address	brief Get channel's remote BT address. ingroup avrcp details This function returns the address of the remote device associated with the channel. param channel AVRCP channel. return li The address of the remote device if channel is connected. li NULL otherwise.
	bt_avrcp_get_company_id_list	brief Get Company ID list. ingroup avrcp details This function requests a list of company id's supported by the remote device param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_get_control_channel_state	brief Get AVCTP control channel state. ingroup avrcp details This function returns status of the AVCTP control channel. param channel AVRCP channel. return Returns of the following values: li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING

	bt_avrcp_get_current_player_application_setting_value	brief Get current player setting values. ingroup avrcp details This function requests a list of current set values for the player application on the target. The list of attribute ids whose values have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_setting_current_values_t::setting_id_list to a list of player setting attribute ids, bt_av_player_setting_current_values_t::count to the number of entries in the list, bt_av_player_setting_current_values_t::setting_value_id_list to a buffer where returned values will be stored. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_current_values_t structure initialized as stated above. return li c TRUE if the function succeeds. li c... more
	bt_avrcp_get_element_attributes	brief Get media element attributes. ingroup avrcp details This function requests the attributes of the element specified with p element_id. note Currently p element_id is ignored. The AVRCP specification defines that only UID 0 can be used to return the attributes of the current track. param channel AVRCP channel. param element_id UID of the media element whose attributes are requested. param attr_mask Bitmask that defines which attributes are requested. This value can be a combination of the following values: li AVC_MEDIA_ATTR_FLAG_TITLE li AVC_MEDIA_ATTR_FLAG_ARTIST li AVC_MEDIA_ATTR_FLAG_ALBUM li AVC_MEDIA_ATTR_FLAG_NUMBER li AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER li AVC_MEDIA_ATTR_FLAG_GENRE li AVC_MEDIA_ATTR_FLAG_PLAYING_TIME li AVC_MEDIA_ATTR_FLAG_ALL return li c TRUE if... more
	bt_avrcp_get_hci_connection	brief Get HCI connection for a channel ingroup avrcp details This function returns a pointer to a structure that describes an HCI connection a channel is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect. param channel AVRCP channel. return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a channel specified by the p channel parameter li does... more
	bt_avrcp_get_mgr	brief Return a pointer to an instance of the AVRCP manager. ingroup avrcp details This function returns a pointer to an instance of the AVRCP manager. There is only one instance of the manager allocated by the stack.
	bt_avrcp_get_play_status	brief Get playback status. ingroup avrcp details This function requests the status of the currently playing media at the target. param channel AVRCP channel. param repeat_interval Interval in milliseconds at which AVRCP polls the target for playback status. If 0 is passed polling is stopped. return li c TRUE if the function succeeds. li c FALSE otherwise.

	bt_avrcp_get_player_application_setting_attr_text	brief Get player setting attribute text. ingroup avrcp details This function requests the target device to provide supported player application setting attribute displayable text for the provided player application setting attributes. The list of attribute ids whose displayable text have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_settings_text_t::setting_id_list to a list of player setting attribute ids, bt_av_player_settings_text_t::count to the number of entries in the list, bt_av_player_settings_text_t::setting_text_list to a buffer where returned values will be stored. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_settings_text_t structure initialized as stated above. return li c... more
	bt_avrcp_get_player_application_setting_value_text	brief Get player setting value text. ingroup avrcp details This function request the target device to provide target supported player application setting value displayable text for the provided player application setting attribute values. The list of attribute ids whose value displayable text have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_setting_values_text_t::setting_value_id_list to a list of player setting attribute value ids, bt_av_player_setting_values_text_t::count to the number of entries in the list, bt_av_player_setting_values_text_t::setting_value_text_list to a buffer where returned values will be stored. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_values_text_t structure initialized as stated... more
	bt_avrcp_get_subuint_info	brief Get subunit info ingroup avrcp details This function is used to request subunit info from the target. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_get_supported_event_id_list	brief Get supported events. ingroup avrcp details This function requests a list of events supported by the remote device param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_get_unit_info	brief Get unit info ingroup avrcp details This function is used to request unit info from the target. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_inform_battery_status	brief Inform controller's battery status. ingroup avrcp details This function is used to inform the target about the controller's battery status. param channel AVRCP channel. param status Battery status. This can be one of the following values: li AVC_BATTERY_STATUS_NORMAL li AVC_BATTERY_STATUS_WARNING li AVC_BATTERY_STATUS_CRITICAL li AVC_BATTERY_STATUS_EXTERNAL li AVC_BATTERY_STATUS_FULL_CHARGE return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_inform_displayable_character_set	brief Inform displayable character set. ingroup avrcp details This function informs the list of character set supported by the controller to the target. param channel AVRCP channel. param charset_list List of displayable character sets. param charset_count Number of entries in the list. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_init_controller	brief Initialize AVRCP to be used in controller mode. ingroup avrcp details This function initializes the AVRCP layer of the stack in controller mode. It must be called prior to any other AVRCP function can be called.

	bt_avrcp_init_target	brief Initialize AVRCP to be used in target mode. ingroup avrcp details This function initializes the AVRCP layer of the stack in target mode. It must be called prior to any other AVRCP function can be called. param company_id The 24-bit unique ID obtained from the IEEE Registration Authority Committee. If the vendor of a TG device does not have the unique ID, the value 0xFFFFFF may be used. param supported_events Bitmask that specifies events supported by the target. This value can be a combination of the following values: li AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED li AVC_EVENT_FLAG_TRACK_CHANGED li AVC_EVENT_FLAG_TRACK_REACHED_END li AVC_EVENT_FLAG_TRACK_REACHED_START li AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED li... more
	bt_avrcp_list_player_application_setting_attributes	brief Get supported player setting attributes. ingroup avrcp details This function request the target device to provide target supported player application setting attributes. The list of attribute ids is stored in the setting_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_settings_t::setting_id_list to a buffer where returned values will be stored and bt_av_player_settings_t::count to the number of entries in the list. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_settings_t structure initialized as stated above. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_list_player_application_setting_values	brief Get player setting attribute values. ingroup avrcp details This function requests the target device to list the set of possible values for the requested player application setting attribute. The list of attribute value ids is stored in the setting_value_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_setting_values_t::setting_value_id_list to a buffer where returned values will be stored and bt_av_player_setting_values_t::count to the number of entries in the list. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_values_t structure initialized as stated above. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_listen	brief Listen for incoming connections. ingroup avrcp details This function enables incoming connections on the specified AVRCP channel. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_play_item	brief Play media item. ingroup avrcp details This function starts playing an item indicated by the UID. param channel AVRCP channel. param scope The scope in which the p element_id is valid. This value can be on the following values: li AVC_SCOPE_MEDIA_PLAYER_LIST li AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM li AVC_SEARCH li AVC_NOW_PLAYING param element_id UID of the media element to be played. param counter UID counter. return li c TRUE if the function succeeds. li c FALSE otherwise.

	bt_avrcp_register_notification	brief Register notification. ingroup avrcp details This function registers with the target to receive notifications asynchronously based on specific events occurring. param channel AVRCP channel. param event_id Event Id. This value can be one of the following values: li AVC_EVENT_PLAYBACK_STATUS_CHANGED li AVC_EVENT_TRACK_CHANGED li AVC_EVENT_TRACK_REACHED_END li AVC_EVENT_TRACK_REACHED_START li AVC_EVENT_PLAYBACK_POS_CHANGED li AVC_EVENT_BATT_STATUS_CHANGED li AVC_EVENT_SYSTEM_STATUS_CHANGED li AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED li AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED li AVC_EVENT_AVAILABLE_PLAYERS_CHANGED li AVC_EVENT_ADDRESSED_PLAYER_CHANGED li AVC_EVENT_UIDS_CHANGED li AVC_EVENT_VOLUME_CHANGED param playback_interval The time interval (in seconds) at which the change in playback position will be notified. Applicable for AVC_EVENT_PLAYBACK_POS_CHANGED event. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_register_notifications	brief Register notifications. ingroup avrcp details This function registers with the target to receive notifications asynchronously based on specific events occurring. This function is used to register multiple notifications with one call. note This function cannot be used to register for AVC_EVENT_PLAYBACK_POS_CHANGED event. param channel AVRCP channel. param event_mask Bitmask that specifies which events to register for. This value can be a combination of the following values: li AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED li AVC_EVENT_FLAG_TRACK_CHANGED li AVC_EVENT_FLAG_TRACK_REACHED_END li AVC_EVENT_FLAG_TRACK_REACHED_START li AVC_EVENT_FLAG_BATT_STATUS_CHANGED li AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED li AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED li AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED li AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED li AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED li AVC_EVENT_FLAG_UIDS_CHANGED li AVC_EVENT_FLAG_VOLUME_CHANGED li AVC_EVENT_FLAG_ALL return li c TRUE if the function... more
	bt_avrcp_send_button_click	brief Send AV/C Panel Subunit "click" PASS THROUGH command ingroup avrcp details This function is used to send a button click. Two PATH THROUGTH commands are sent. The first command has button state set to AVC_PANEL_BUTTON_PRESSED. The second command gas button state set to AVC_PANEL_BUTTON_RELEASED param channel AVRCP channel. param button_id Operation Id. This value can be on of the AVC_PANEL_OPID_... constants return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_send_cmd	brief Send AVRCP command ingroup avrcp details This function sends a command to the remote device. param channel AVRCP channel. param command Command to be sent. return li c TRUE if the function succeeds. li c FALSE otherwise.

	bt_avrcp_send_panel_control	brief Send AV/C Panel Subunit "control" PASS THROUGH command ingroup avrcp details This function is used to send AV/C Panel Subunit PASS THROUGH command with command type set to AVC_CTYPE_CONTROL. param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants param button_state Button state. This can be on of the following values: li AVC_PANEL_BUTTON_PRESSED li AVC_PANEL_BUTTON_RELEASED return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_send_press_panel_control	brief Send AV/C Panel Subunit "pressed" PASS THROUGH command ingroup avrcp details This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC_PANEL_BUTTON_PRESSED. param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_send_simple_panel_cmd	brief Send AV/C Panel Subunit PASS THROUGH command ingroup avrcp details This function is used to send AV/C Panel Subunit PASS THROUGH command to the target. param channel AVRCP channel. param ctype Command type. This value can be on of the following values: li AVC_CTYPE_CONTROL 0 li AVC_CTYPE_STATUS 1 li AVC_CTYPE_SPECIFIC_IQUIRY 2 li AVC_CTYPE_NOTIFY 3 li AVC_CTYPE_GENERAL_INQUORY 4 param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants param button_state Button state. This can be on of the following values: li AVC_PANEL_BUTTON_PRESSED li AVC_PANEL_BUTTON_RELEASED return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_set_absolute_volume	brief Set absolute volume. ingroup avrcp details This function is used to set an absolute volume to be used by the rendering device. param channel AVRCP channel. param volume Volume return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_set_addressed_player	brief Set addressed player. ingroup avrcp details This function is used to inform the target of which media player the controller wishes to control. param channel AVRCP channel. param player_id Player Id. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_set_player_application_setting_value	brief Set player setting attribute values. ingroup avrcp details This function requests to set the player application setting list of player application setting values on the target device for the corresponding defined list of setting attributes. for the requested player application setting attribute. The list of attribute value ids is stored in the setting_value_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_setting_values_t::setting_value_id_list to a buffer where returned values will be stored and bt_av_player_setting_values_t::count to the number of entries in the list. param channel AVRCP channel. param attr_id_list List of setting attribute ids. param attr_value_list List of... more
	bt_avrcp_start	brief Start the AVRCP layer. ingroup avrcp details In order to be notified of various events a consumer of the AVRCP layer has to register a callback function. The stack will call the callback function whenever a new event has been generated passing the code of the event as the second parameter. bt_avrcp_start() stores pointers to the c callback and c callback_param in the bt_avrcp_mgr_t structure. param mgr AVRCP manager. param callback The callback function that will be called when the AVRCP generates an event. param callback_param A pointer to arbitrary data to be passed to the c callback callback.... more

	bt_avrcp_tg_send_element_attributes	brief Send media element attributes ingroup avrcp details This function is used to send the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_SYSTEM_STATUS_CHANGED event will be notified. param status Battery status
	bt_avrcp_tg_set_absolute_volume	brief Set absolute volume ingroup avrcp details This function is used to set the absolute volume when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_VOLUME_CHANGED event will be notified. param track_id Track Id param song_length The length of the current track. param song_position The position of the current track.
	bt_avrcp_tg_set_battery_status	brief Set battery status ingroup avrcp details This function is used to set the battery status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_BATT_STATUS_CHANGED event will be notified. param status Battery status
	bt_avrcp_tg_set_channel_absolute_volume	This is function bt_avrcp_tg_set_channel_absolute_volume.
	bt_avrcp_tg_set_current_track	brief Set current track ingroup avrcp details This function is used to set the current track when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_TRACK_CHANGED event will be notified. param track_id Track Id param song_length The length of the current track. param song_position The position of the current track.
	bt_avrcp_tg_set_play_status	local target control * • brief Set playback status • ingroup avrcp * • details This function is used to set the playback status when AVRCP is running in target mode. • If there are active connections with controllers, the ones that registered • for AVC_EVENT_PLAYBACK_STATUS_CHANGED event will be notified. * • param song_length The length of the current track. • param song_position The position of the current track. • param play_status Playback status. This value can be one of the following values: • li AVC_PLAY_STATUS_STOPPED • li AVC_PLAY_STATUS_PLAYING • li AVC_PLAY_STATUS_PAUSED • li AVC_PLAY_STATUS_FW_SEEK • li AVC_PLAY_STATUS_REV_SEEK
	bt_avrcp_tg_set_system_status	brief Set system status ingroup avrcp details This function is used to set the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_SYSTEM_STATUS_CHANGED event will be notified. param status Battery status
	_bt_avrcp_allocate_browsing_response	This is function _bt_avrcp_allocate_browsing_response.
	_bt_avrcp_general_reject	This is function _bt_avrcp_general_reject.
	_bt_avrcp_send_rejected_response	This is function _bt_avrcp_send_rejected_response.

	bt_avrcp_send_release_panel_control	brief Send AV/C Panel Subunit "released" PASS THROUGH command ingroup avrcp details This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC_PANEL_BUTTON_RELEASED. param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants return li c TRUE if the function succeeds. li c FALSE otherwise.

Bluetooth Support Data Types and Constants

	Name	Description
	ARG_NOT_USED	This is macro ARG_NOT_USED.
	BT_ASSERT	This is macro BT_ASSERT.
	BT_FALSE	This is macro BT_FALSE.
	BT_LINKKEY_LENGTH	This is macro BT_LINKKEY_LENGTH.
	BT_LOG	This is macro BT_LOG.
	BT_LOG_EX	This is macro BT_LOG_EX.
	BT_LOG_LEVEL_ALL	This is macro BT_LOG_LEVEL_ALL.
	BT_LOG_LEVEL_DEBUG	This is macro BT_LOG_LEVEL_DEBUG.
	BT_LOG_LEVEL_ERROR	This is macro BT_LOG_LEVEL_ERROR.
	BT_LOG_LEVEL_INFO	This is macro BT_LOG_LEVEL_INFO.
	BT_LOG_LEVEL_OFF	This is macro BT_LOG_LEVEL_OFF.
	BT_LOGADDR	This is macro BT_LOGADDR.
	BT_LOGADDR_EX	This is macro BT_LOGADDR_EX.
	BT_LOGINT	This is macro BT_LOGINT.
	BT_LOGINT_EX	This is macro BT_LOGINT_EX.
	BT_LOGLINKKEY	This is macro BT_LOGLINKKEY.
	BT_LOGLINKKEY_EX	This is macro BT_LOGLINKKEY_EX.
	BT_LOGMEMORY	This is macro BT_LOGMEMORY.
	BT_LOGMEMORY_EX	This is macro BT_LOGMEMORY_EX.
	BT_LOGWRITE	This is macro BT_LOGWRITE.
	BT_MAKE_BDADDR	This is macro BT_MAKE_BDADDR.
	BT_MAKE_BDADDR_LE	This is macro BT_MAKE_BDADDR_LE.
	BT_NO	This is macro BT_NO.
	BT_TRUE	This is macro BT_TRUE.
	BT_YES	This is macro BT_YES.
	CDS_LAST_DEVICE_ADDR	This is macro CDS_LAST_DEVICE_ADDR.
	CDS_SIGNATURE	This is macro CDS_SIGNATURE.
	CDS_SIGNATURE_ADDR	This is macro CDS_SIGNATURE_ADDR.
	FALSE	This is macro FALSE.
	HCI_BDADDR_LEN	This is macro HCI_BDADDR_LEN.
	HIDS_LAST_DEVICE_ADDR	This is macro HIDS_LAST_DEVICE_ADDR.
	HIDS_SIGNATURE	This is macro HIDS_SIGNATURE.
	HIDS_SIGNATURE_ADDR	This is macro HIDS_SIGNATURE_ADDR.
	LKS_FIRST_KEY_ADDR	This is macro LKS_FIRST_KEY_ADDR.
	LKS_MAX_LINK_KEYS	This is macro LKS_MAX_LINK_KEYS.
	LKS_SIGNATURE	This is macro LKS_SIGNATURE.
	LKS_SIGNATURE_ADDR	This is macro LKS_SIGNATURE_ADDR.
	LOG	This is macro LOG.
	LOG_EX	This is macro LOG_EX.
	LOGADDR	This is macro LOGADDR.
	LOGADDR_EX	This is macro LOGADDR_EX.
	LOGCLEAR	This is macro LOGCLEAR.
	LOGINT	This is macro LOGINT.
	LOGINT_EX	This is macro LOGINT_EX.
	LOGMEMORY	This is macro LOGMEMORY.

	LOGMEMORY_EX	This is macro LOGMEMORY_EX.
	LOGWRITE	This is macro LOGWRITE.
	NULL	This is macro NULL.
	TRUE	This is macro TRUE.
	_bt_linkkey_notification_t	This is type bt_linkkey_notification_t.
	_bt_linkkey_request_t	This is type bt_linkkey_request_t.
	_bt_linkkey_t	This is type bt_linkkey_t.
	_bt_signal_t	This is record _bt_signal_t.
	_bt_timer_id_enum	This is type bt_timer_id.
	_bt_uuid_s	This is type bt_uuid_t.
	_bt_bdaddr_s	This is type bt_bdaddr_t.
	bt_bdaddr_cp	This is type bt_bdaddr_cp.
	bt_bdaddr_p	This is type bt_bdaddr_p.
	bt_bdaddr_t	This is type bt_bdaddr_t.
	bt_bool	This is type bt_bool.
	bt_byte_cp	This is type bt_byte_cp.
	bt_byte_p	This is type bt_byte_p.
	bt_char	This is type bt_char.
	bt_char_cp	This is type bt_char_cp.
	bt_char_p	This is type bt_char_p.
	bt_id	This is type bt_id.
	bt_int_cp	This is type bt_int_cp.
	bt_int_p	This is type bt_int_p.
	bt_linkkey_cp	This is type bt_linkkey_cp.
	bt_linkkey_notification_t	This is type bt_linkkey_notification_t.
	bt_linkkey_p	This is type bt_linkkey_p.
	bt_linkkey_request_t	This is type bt_linkkey_request_t.
	bt_linkkey_t	This is type bt_linkkey_t.
	bt_long_cp	This is type bt_long_cp.
	bt_long_p	This is type bt_long_p.
	bt_oem_recv_callback_fp	• Receive callback. This callback function is called when a receive operation initiated by bt_oem_recv() has completed. @param len Number of received bytes. The value of this parameter should always be the same as the number of bytes requested in a call to bt_oem_recv().
	bt_oem_send_callback_fp	• Send callback. This callback function is called when a send operation initiated by bt_oem_send() has completed.
	bt_signal_handler_fp	This is type bt_signal_handler_fp.
	bt_signal_t	This is type bt_signal_t.
	bt_storage_callback_fp	brief Storage callback. details This callback is called when a non-volatile storage operation completes.
	bt_sys_callback_fp	• System start callback. This callback function is called when system start initiated by bt_sys_start() has completed. @param success Success of the operation: c BT_TRUE if successfull, c BT_FALSE otherwise. @param param Callback parameter that was specified when bt_sys_start() was called.
	bt_timer_callback_fp	brief Timer callback. details This callback is called when a timer expires.
	bt_timer_id	This is type bt_timer_id.
	bt_uint_cp	This is type bt_uint_cp.
	bt_uint_p	This is type bt_uint_p.
	bt_ulong_cp	This is type bt_ulong_cp.
	bt_ulong_p	This is type bt_ulong_p.
	bt_uuid_cp	This is type bt_uuid_cp.
	bt_uuid_p	This is type bt_uuid_p.

	bt_uuid_t	This is type bt_uuid_t.
	bt_uuid16	This is type bt_uuid16.
	bt_uuid32	This is type bt_uuid32.

Bluetooth Support Functions

	Name	Description
	bt_bdaddr_is_null	This is function bt_bdaddr_is_null.
	bt_bdaddrs_are_equal	This is function bt_bdaddrs_are_equal.
	bt_log_bdaddr	This is function bt_log_bdaddr.
	bt_log_int	This is function bt_log_int.
	bt_log_linkkey	This is function bt_log_linkkey.
	bt_log_memory	This is function bt_log_memory.
	bt_log_msg	This is function bt_log_msg.
	bt_oem_assert	This is function bt_oem_assert.
	bt_oem_get_device_class	This is function bt_oem_get_device_class.
	bt_oem_get_device_name	This is function bt_oem_get_device_name.
	bt_oem_get_pin_code	This is function bt_oem_get_pin_code.
	bt_oem_linkkey_notification	This is function bt_oem_linkkey_notification.
	bt_oem_linkkey_request	This is function bt_oem_linkkey_request.
	bt_oem_log_write	brief Output log message. ingroup log details DotStack calls this function to output its debug information. Implementation should output or store the specified message to whatever device or medium where it can be examined and analyzed.
	bt_oem_recv	• Receive data. This function is called by the HCI layer when it needs more data from the HCI controller. Implementation of this function must receive the specified number of bytes from the HCI controller and call the provided callback function. @param buffer Pointer to a buffer for the received data. The buffer must be long enough to accommodate the number of bytes specified by the par len parameter. @param len Number of bytes to receive. @param callback A callback function that must be called when the requested number of bytes have been received.
	bt_oem_schedule_signals	This is function bt_oem_schedule_signals.
	bt_oem_send	• Send data. This function is called by the HCI layer when it needs to send data to the HCI controller. Implementation of this function must send the specified number of bytes to the HCI controller and call the provided callback function. @param buffer Pointer to the data to be sent . @param len Number of bytes to send. @param callback A callback function that must be called when all data have been sent.
	bt_oem_storage_get_capacity	brief Get non-volatile storage capacity. ingroup stg details Implementation of this function must return the capacity of its non-volatile storage.
	bt_oem_storage_read	brief Read from the non-volatile storage. ingroup stg details This function is called by the stack to read from the non-volatile storage. This function must be implemented by the application. When this function is called the application must start a read operation. When the number of bytes specified by the c len parameter is read, the application must call the callback function specified by the c callback parameter. The application does not have to read the whole number of bytes during the call to this function. It may complete reading later and then call the completion callback. The stack guarantees... more
	bt_oem_storage_start	brief Begin a sequence of non-volatile storage operations. ingroup stg details DotStack calls this function when it starts a sequence of non-volatile storage operations. When the sequence is finished, DotStack will call bt_oem_storage_stop().
	bt_oem_storage_stop	brief End a sequence of non-volatile storage operations. ingroup stg details DotStack calls this function when it finishes executing a sequence of non-volatile storage operations.

	bt_oem_storage_write	brief Write to non-volatile storage. ingroup stg details This function is called by the stack to write data to the non-volatile storage. This function must be implemented by the application. When this function is called, the application must start writing specified data to the non-volatile storage. When all data has been written, the application must call the callback function passed in the c callback parameter. The application does not have to complete the write operation during the call to this function. It may complete the operation later and then call the callback function. In this case, the application does not... more
	bt_oem_timer_clear	brief Clear timer. details This function must be implemented by the application. When this function is called the application must clear the specified timer. If it is already expired and a callback is currently pending, the application should also take measures to cancel the callback. param timerId ID of the timer to clear.
	bt_oem_timer_set	brief Set timer. details This function must be implemented by the application. When it is called, the application must set the specified timer. When the timer expires, the application must call the passed callback function. The function must not wait until the timer expires. It must set the timer and exit immediately. param timerId ID of the timer to set. param milliseconds Timer interval in milliseconds param callback Timer expiration callback function.
	bt_signal_process_pending	This is function bt_signal_process_pending.
	bt_signal_register	This is function bt_signal_register.
	bt_signal_set	This is function bt_signal_set.
	bt_signal_unregister	This is function bt_signal_unregister.
	bt_sys_get_connectable	This is function bt_sys_get_connectable.
	bt_sys_get_discoverable	This is function bt_sys_get_discoverable.
	bt_sys_get_l2cap_manager	• Get the L2CAP manager. This function returns the L2CAP manager. The L2CAP manager is created as part of the start up sequence. @return The L2CAP manager.
	bt_sys_init	• Initialize the Bluetooth system. This function initializes all internal variables of HCI, L2CAP and SDP modules. It must be called by the application before it can access any functionality provided by the library. In addition to this initialization function the application must call initialization functions of all other profile modules the application is intended to use. E.g., if the application is using the SPP module the bt_spp_init() must be called right after calling bt_sys_init(). This function essentially calls bt_sys_init_ex(HCI_LINK_POLICY_ENABLE_ALL) so all link policy setting are enabled.
	bt_sys_set_modes	This is function bt_sys_set_modes.
	bt_sys_start	• Start the Bluetooth system. After all modules used by the application have been initialized this function should be called to start the Bluetooth system operation. During the start up sequence it will reset and initialize the HCI controller and then create the L2CAP manager. The application will be notified when the start up sequence completes by calling the provided callback function. Also, the caller must provide an SDP database. @param discoverable defines whether the device is discoverable after reset. @param connectable defines whether the device is connectable after reset. @param sdp_db SDP database data. @param sdp_db_len Length of SDP database... more
	HCITR_BCSP_ALLOCATE_BUFFERS	This is function HCITR_BCSP_ALLOCATE_BUFFERS.
	IAPEA_ALLOCATE_BUFFERS	This is function IAPEA_ALLOCATE_BUFFERS.
	bt_oem_ssp_callback	This is function bt_oem_ssp_callback.
	bt_signal_init	This is function bt_signal_init.
	bt_sys_init_ex	• Initialize the Bluetooth system. This function initializes all internal variables of HCI, L2CAP and SDP modules. It must be called by the application before it can access any functionality provided by the library. In addition to this initialization function the application must call initialization functions of all other profile modules the application is intended to use. E.g., if the application is using the SPP module the bt_spp_init() must be called right after calling bt_sys_init(). Also, the caller must provide an SDP database. @param default_link_policy default link policy settings. This is a bitmask that defines the initial value of... more

	HCRP_ALLOCATE_BUFFERS	This is function HCRP_ALLOCATE_BUFFERS.
	MAP_ALLOCATE_BUFFERS	This is function MAP_ALLOCATE_BUFFERS.
	bt_hcitr_3wire_cancel_recv_packet	This is function bt_hcitr_3wire_cancel_recv_packet.
	bt_q_contains	This is function bt_q_contains.

GAP Data Types and Constants

	Name	Description
	_bt_device_t	This is type bt_device_t.
	bt_device_t	This is type bt_device_t.
	bt_find_devices_callback_fp	This is type bt_find_devices_callback_fp.

GAP Functions

	Name	Description
	bt_gap_find_devices	This is function bt_gap_find_devices.

HCI Data Types and Constants

	Name	Description
	bt_hci_add_param_hconn	brief Add connection handle parameter to an HCI command ingroup hci
	bt_hci_add_param_lap	This is macro bt_hci_add_param_lap.
	bt_hci_add_param_uint	brief Add unsigned int parameter to an HCI command ingroup hci
	bt_hci_add_param_ulong	brief Add unsigned long parameter to an HCI command ingroup hci
	bt_hci_authenticate	This is macro bt_hci_authenticate.
	bt_hci_exit_sniff_mode	brief Cancel sniff mode
	bt_hci_get_evt_param_hconn	brief Get connection handle parameter from HCI event ingroup hci
	bt_hci_get_evt_param_uint	This is macro bt_hci_get_evt_param_uint.
	bt_hci_get_evt_param_ulong	This is macro bt_hci_get_evt_param_ulong.
	bt_hci_get_param_hconn	brief Get connection handle parameter from HCI command ingroup hci
	bt_hci_le_set_adevrtising_enable	This is macro bt_hci_le_set_adevrtising_enable.
	bt_hci_role_change	This is macro bt_hci_role_change.
	bt_hci_set_encryption	This is macro bt_hci_set_encryption.
	bt_hci_sniff_mode	brief Put local device to sniff mode
	bt_hci_sniff_subrating	This is macro bt_hci_sniff_subrating.
	bt_hcitr_3wire_init	This is macro bt_hcitr_3wire_init.
	bt_hcitr_3wire_reset	This is macro bt_hcitr_3wire_reset.
	bt_hcitr_bcsp_init	This is macro bt_hcitr_bcsp_init.
	bt_hcitr_bcsp_reset	This is macro bt_hcitr_bcsp_reset.
	COD_MAJOR_AUDIO	This is macro COD_MAJOR_AUDIO.
	COD_MAJOR_COMPUTER	This is macro COD_MAJOR_COMPUTER.
	COD_MAJOR_HEALTH	This is macro COD_MAJOR_HEALTH.
	COD_MAJOR_IMAGING	This is macro COD_MAJOR_IMAGING.
	COD_MAJOR_MISC	This is macro COD_MAJOR_MISC.
	COD_MAJOR_NET_ACCESS_POINT	This is macro COD_MAJOR_NET_ACCESS_POINT.
	COD_MAJOR_PERIPHERAL	This is macro COD_MAJOR_PERIPHERAL.
	COD_MAJOR_PHONE	This is macro COD_MAJOR_PHONE.
	COD_MAJOR_TOY	This is macro COD_MAJOR_TOY.
	COD_MAJOR_UNCATEGORIZED	This is macro COD_MAJOR_UNCATEGORIZED.
	COD_MAJOR_WEARABLE	This is macro COD_MAJOR_WEARABLE.
	COD_MINOR_AV_CAMCORDER	This is macro COD_MINOR_AV_CAMCORDER.
	COD_MINOR_AV_CAR_AUDIO	This is macro COD_MINOR_AV_CAR_AUDIO.
	COD_MINOR_AV_GAMING	This is macro COD_MINOR_AV_GAMING.
	COD_MINOR_AV_HANDSFREE	This is macro COD_MINOR_AV_HANDSFREE.

	COD_MINOR_AV_HEADPHONES	This is macro COD_MINOR_AV_HEADPHONES.
	COD_MINOR_AV_HEADSET	This is macro COD_MINOR_AV_HEADSET.
	COD_MINOR_AV_HIFI_AUDIO	This is macro COD_MINOR_AV_HIFI_AUDIO.
	COD_MINOR_AV_LOUDSPEAKER	This is macro COD_MINOR_AV_LOUDSPEAKER.
	COD_MINOR_AV_MICROPHONE	This is macro COD_MINOR_AV_MICROPHONE.
	COD_MINOR_AV_PORTABLE_AUDIO	This is macro COD_MINOR_AV_PORTABLE_AUDIO.
	COD_MINOR_AV_RESERVED	This is macro COD_MINOR_AV_RESERVED.
	COD_MINOR_AV_RESERVERD2	This is macro COD_MINOR_AV_RESERVERD2.
	COD_MINOR_AV_SET_TOP_BOX	This is macro COD_MINOR_AV_SET_TOP_BOX.
	COD_MINOR_AV_UNCATEGORIZED	This is macro COD_MINOR_AV_UNCATEGORIZED.
	COD_MINOR_AV_VCR	This is macro COD_MINOR_AV_VCR.
	COD_MINOR_AV_VIDE_DISPLAY_AND_LOUDSPEAKER	This is macro COD_MINOR_AV_VIDE_DISPLAY_AND_LOUDSPEAKER.
	COD_MINOR_AV_VIDEO_CAMERA	This is macro COD_MINOR_AV_VIDEO_CAMERA.
	COD_MINOR_AV_VIDEO_CONFERENCING	This is macro COD_MINOR_AV_VIDEO_CONFERENCING.
	COD_MINOR_AV_VIDEO_MONITOR	This is macro COD_MINOR_AV_VIDEO_MONITOR.
	COD_MINOR_COMPUTER_DESKTOP	This is macro COD_MINOR_COMPUTER_DESKTOP.
	COD_MINOR_COMPUTER_HANDHELD	This is macro COD_MINOR_COMPUTER_HANDHELD.
	COD_MINOR_COMPUTER_LAPTOP	This is macro COD_MINOR_COMPUTER_LAPTOP.
	COD_MINOR_COMPUTER_PALMSIZED	This is macro COD_MINOR_COMPUTER_PALMSIZED.
	COD_MINOR_COMPUTER_SERVER	This is macro COD_MINOR_COMPUTER_SERVER.
	COD_MINOR_COMPUTER_UNCATEGORIZED	This is macro COD_MINOR_COMPUTER_UNCATEGORIZED.
	COD_MINOR_COMPUTER_WEARABLE	This is macro COD_MINOR_COMPUTER_WEARABLE.
	COD_MINOR_HEALTH_BPM	This is macro COD_MINOR_HEALTH_BPM.
	COD_MINOR_HEALTH_DATA_DISPLAY	This is macro COD_MINOR_HEALTH_DATA_DISPLAY.
	COD_MINOR_HEALTH_GLUCOSE_METER	This is macro COD_MINOR_HEALTH_GLUCOSE_METER.
	COD_MINOR_HEALTH_HEART_MONITOR	This is macro COD_MINOR_HEALTH_HEART_MONITOR.
	COD_MINOR_HEALTH_PULSE_OXIMETER	This is macro COD_MINOR_HEALTH_PULSE_OXIMETER.
	COD_MINOR_HEALTH_THERMOMETER	This is macro COD_MINOR_HEALTH_THERMOMETER.
	COD_MINOR_HEALTH_UNCATEGORIZED	This is macro COD_MINOR_HEALTH_UNCATEGORIZED.
	COD_MINOR_HEALTH_WEIGHING_SCALE	This is macro COD_MINOR_HEALTH_WEIGHING_SCALE.
	COD_MINOR_IMAGING_CAMERA	This is macro COD_MINOR_IMAGING_CAMERA.
	COD_MINOR_IMAGING_DISPLAY	This is macro COD_MINOR_IMAGING_DISPLAY.
	COD_MINOR_IMAGING_PRINTER	This is macro COD_MINOR_IMAGING_PRINTER.
	COD_MINOR_IMAGING_SCANNER	This is macro COD_MINOR_IMAGING_SCANNER.

	COD_MINOR_IMAGING_UNCATEGORIZED	This is macro COD_MINOR_IMAGING_UNCATEGORIZED.
	COD_MINOR_LAN_01_17	This is macro COD_MINOR_LAN_01_17.
	COD_MINOR_LAN_17_33	This is macro COD_MINOR_LAN_17_33.
	COD_MINOR_LAN_33_50	This is macro COD_MINOR_LAN_33_50.
	COD_MINOR_LAN_50_67	This is macro COD_MINOR_LAN_50_67.
	COD_MINOR_LAN_67_83	This is macro COD_MINOR_LAN_67_83.
	COD_MINOR_LAN_83_99	This is macro COD_MINOR_LAN_83_99.
	COD_MINOR_LAN_FULLY_AVAILABLE	This is macro COD_MINOR_LAN_FULLY_AVAILABLE.
	COD_MINOR_LAN_NO_SERVICE	This is macro COD_MINOR_LAN_NO_SERVICE.
	COD_MINOR_LAN_UNCATEGORIZED	This is macro COD_MINOR_LAN_UNCATEGORIZED.
	COD_MINOR_PERIPHERAL_CARD_READER	This is macro COD_MINOR_PERIPHERAL_CARD_READER.
	COD_MINOR_PERIPHERAL_COMBO	This is macro COD_MINOR_PERIPHERAL_COMBO.
	COD_MINOR_PERIPHERAL_DIGITIZER	This is macro COD_MINOR_PERIPHERAL_DIGITIZER.
	COD_MINOR_PERIPHERAL_GAMEPAD	This is macro COD_MINOR_PERIPHERAL_GAMEPAD.
	COD_MINOR_PERIPHERAL_JOYSTICK	This is macro COD_MINOR_PERIPHERAL_JOYSTICK.
	COD_MINOR_PERIPHERAL_KEYBOARD	This is macro COD_MINOR_PERIPHERAL_KEYBOARD.
	COD_MINOR_PERIPHERAL_MOUSE	This is macro COD_MINOR_PERIPHERAL_MOUSE.
	COD_MINOR_PERIPHERAL_OTHER	This is macro COD_MINOR_PERIPHERAL_OTHER.
	COD_MINOR_PERIPHERAL_REMOTE	This is macro COD_MINOR_PERIPHERAL_REMOTE.
	COD_MINOR_PERIPHERAL_SENSING	This is macro COD_MINOR_PERIPHERAL_SENSING.
	COD_MINOR_PERIPHERAL_UNCATEGORIZED	This is macro COD_MINOR_PERIPHERAL_UNCATEGORIZED.
	COD_MINOR_PHONE_CELLULAR	This is macro COD_MINOR_PHONE_CELLULAR.
	COD_MINOR_PHONE_CORDLESS	This is macro COD_MINOR_PHONE_CORDLESS.
	COD_MINOR_PHONE_ISDN	This is macro COD_MINOR_PHONE_ISDN.
	COD_MINOR_PHONE_SMART	This is macro COD_MINOR_PHONE_SMART.
	COD_MINOR_PHONE_UNCATEGORIZED	This is macro COD_MINOR_PHONE_UNCATEGORIZED.
	COD_MINOR_PHONE_WIREDMODEM	This is macro COD_MINOR_PHONE_WIREDMODEM.
	COD_MINOR_TOY_CONTROLLER	This is macro COD_MINOR_TOY_CONTROLLER.
	COD_MINOR_TOY_DOLL	This is macro COD_MINOR_TOY_DOLL.
	COD_MINOR_TOY_GAME	This is macro COD_MINOR_TOY_GAME.
	COD_MINOR_TOY_ROBOT	This is macro COD_MINOR_TOY_ROBOT.
	COD_MINOR_TOY_UNCATEGORIZED	This is macro COD_MINOR_TOY_UNCATEGORIZED.
	COD_MINOR_TOY_VEHICLE	This is macro COD_MINOR_TOY_VEHICLE.
	COD_MINOR_WEARABLE_GLASSES	This is macro COD_MINOR_WEARABLE_GLASSES.
	COD_MINOR_WEARABLE_HELMET	This is macro COD_MINOR_WEARABLE_HELMET.
	COD_MINOR_WEARABLE_JACKET	This is macro COD_MINOR_WEARABLE_JACKET.

	COD_MINOR_WEARABLE_PAGER	This is macro COD_MINOR_WEARABLE_PAGER.
	COD_MINOR_WEARABLE_UNCATEGORIZED	This is macro COD_MINOR_WEARABLE_UNCATEGORIZED.
	COD_MINOR_WEARABLE_WATCH	This is macro COD_MINOR_WEARABLE_WATCH.
	COS_AUDIO	This is macro COS_AUDIO.
	COS_CAPTURING	This is macro COS_CAPTURING.
	COS_INFORMATION	This is macro COS_INFORMATION.
	COS_LIMITED_DISCOVERABLE_MODE	This is macro COS_LIMITED_DISCOVERABLE_MODE.
	COS_NETWORKING	This is macro COS_NETWORKING.
	COS_OBJECTTRANSFER	This is macro COS_OBJECTTRANSFER.
	COS_POSITIONING	This is macro COS_POSITIONING.
	COS_RENDERING	This is macro COS_RENDERING.
	COS_TELEPHONY	This is macro COS_TELEPHONY.
	HCI_ACCEPT_CONNECTION_REQUEST	This is macro HCI_ACCEPT_CONNECTION_REQUEST.
	HCI_ACCEPT_SYNCH_CONNECTION_REQUEST	This is macro HCI_ACCEPT_SYNCH_CONNECTION_REQUEST.
	HCI_ACL_DATA_HEADER_LEN	This is macro HCI_ACL_DATA_HEADER_LEN.
	HCI_AUTHENTICATION_REQUESTED	This is macro HCI_AUTHENTICATION_REQUESTED.
	HCI_BB_PACKET_TYPE_DH1	This is macro HCI_BB_PACKET_TYPE_DH1.
	HCI_BB_PACKET_TYPE_DH3	This is macro HCI_BB_PACKET_TYPE_DH3.
	HCI_BB_PACKET_TYPE_DH5	This is macro HCI_BB_PACKET_TYPE_DH5.
	HCI_BB_PACKET_TYPE_DM1	This is macro HCI_BB_PACKET_TYPE_DM1.
	HCI_BB_PACKET_TYPE_DM3	This is macro HCI_BB_PACKET_TYPE_DM3.
	HCI_BB_PACKET_TYPE_DM5	This is macro HCI_BB_PACKET_TYPE_DM5.
	HCI_BB_PACKET_TYPE_NO_2_DH1	This is macro HCI_BB_PACKET_TYPE_NO_2_DH1.
	HCI_BB_PACKET_TYPE_NO_2_DH3	This is macro HCI_BB_PACKET_TYPE_NO_2_DH3.
	HCI_BB_PACKET_TYPE_NO_2_DH5	This is macro HCI_BB_PACKET_TYPE_NO_2_DH5.
	HCI_BB_PACKET_TYPE_NO_3_DH1	This is macro HCI_BB_PACKET_TYPE_NO_3_DH1.
	HCI_BB_PACKET_TYPE_NO_3_DH3	This is macro HCI_BB_PACKET_TYPE_NO_3_DH3.
	HCI_BB_PACKET_TYPE_NO_3_DH5	This is macro HCI_BB_PACKET_TYPE_NO_3_DH5.
	HCI_C2H_BROADCAST_NOT_PARCKED_SLAVE	This is macro HCI_C2H_BROADCAST_NOT_PARCKED_SLAVE.
	HCI_C2H_BROADCAST_P2P	This is macro HCI_C2H_BROADCAST_P2P.
	HCI_C2H_BROADCAST_PARCKED_SLAVE	This is macro HCI_C2H_BROADCAST_PARCKED_SLAVE.
	HCI_C2H_BROADCAST_RESERVED	This is macro HCI_C2H_BROADCAST_RESERVED.
	HCI_CHANGE_CONNECTION_LINK_KEY	This is macro HCI_CHANGE_CONNECTION_LINK_KEY.
	HCI_CHANGE_CONNECTION_PACKET_TYPE	This is macro HCI_CHANGE_CONNECTION_PACKET_TYPE.
	HCI_CMD_HEADER_LEN	This is macro HCI_CMD_HEADER_LEN.
	HCI_CMD_STATUS_BEING_SENT	This is macro HCI_CMD_STATUS_BEING_SENT.
	HCI_CMD_STATUS_PENDING	This is macro HCI_CMD_STATUS_PENDING.
	HCI_CMD_STATUS_WAITING_RESPONSE	This is macro HCI_CMD_STATUS_WAITING_RESPONSE.

	HCI_CONFIG_BECOME_MASTER	This is macro HCI_CONFIG_BECOME_MASTER.
	HCI_CONFIG_ENABLE_AUTHENTICATION	This is macro HCI_CONFIG_ENABLE_AUTHENTICATION.
	HCI_CONFIG_ENABLE_ENCRYPTION	This is macro HCI_CONFIG_ENABLE_ENCRYPTION.
	HCI_CONN_ROLE_MASTER	This is macro HCI_CONN_ROLE_MASTER.
	HCI_CONN_ROLE_SLAVE	This is macro HCI_CONN_ROLE_SLAVE.
	HCI_CONN_STATE_AUTHENTICATING	This is macro HCI_CONN_STATE_AUTHENTICATING.
	HCI_CONN_STATE_CLOSED	This is macro HCI_CONN_STATE_CLOSED.
	HCI_CONN_STATE_OPEN	This is macro HCI_CONN_STATE_OPEN.
	HCI_CONN_TYPE_ACL	This is macro HCI_CONN_TYPE_ACL.
	HCI_CONN_TYPE_ESCO	This is macro HCI_CONN_TYPE_ESCO.
	HCI_CONN_TYPE_SCO	This is macro HCI_CONN_TYPE_SCO.
	HCI_CONNECTABLE	This is macro HCI_CONNECTABLE.
	HCI_CONTROLLER	This is macro HCI_CONTROLLER.
	HCI_CREATE_CONNECTION	This is macro HCI_CREATE_CONNECTION.
	HCI_CREATE_CONNECTION_CANCEL	This is macro HCI_CREATE_CONNECTION_CANCEL.
	HCI_CREATE_NEW_UNIT_KEY	This is macro HCI_CREATE_NEW_UNIT_KEY.
	HCI_CTRL_LISTENER_ACL_DATA	This is macro HCI_CTRL_LISTENER_ACL_DATA.
	HCI_CTRL_LISTENER_EVENT	This is macro HCI_CTRL_LISTENER_EVENT.
	HCI_CTRL_LISTENER_SCO_DATA	This is macro HCI_CTRL_LISTENER_SCO_DATA.
	HCI_CTRL_STATE_CLOSED	This is macro HCI_CTRL_STATE_CLOSED.
	HCI_CTRL_STATE_CONNECTABLE	This is macro HCI_CTRL_STATE_CONNECTABLE.
	HCI_CTRL_STATE_DISCOVERABLE	This is macro HCI_CTRL_STATE_DISCOVERABLE.
	HCI_CTRL_STATE_INIT	This is macro HCI_CTRL_STATE_INIT.
	HCI_CTRL_STATE_READY	This is macro HCI_CTRL_STATE_READY.
	HCI_CTRL_STATE_SLEEP	This is macro HCI_CTRL_STATE_SLEEP.
	HCI_CTRL_STATE_WAKING_UP	This is macro HCI_CTRL_STATE_WAKING_UP.
	HCI_DATA_BUFFER_STATE_FREE	This is macro HCI_DATA_BUFFER_STATE_FREE.
	HCI_DATA_BUFFER_STATE_USED	This is macro HCI_DATA_BUFFER_STATE_USED.
	HCI_DATA_STATUS_BEING_SENT	This is macro HCI_DATA_STATUS_BEING_SENT.
	HCI_DATA_STATUS_PENDING	This is macro HCI_DATA_STATUS_PENDING.
	HCI_DEFAULT_ACL_CONFIG	Default value for acl_config parameter of bt_hci_connect()
	HCI_DELETE_STORED_LINK_KEY	This is macro HCI_DELETE_STORED_LINK_KEY.
	HCI_DISCONNECT	This is macro HCI_DISCONNECT.
	HCI_DISCOVERABLE	This is macro HCI_DISCOVERABLE.
	HCI_EIR_FEC_NOT_REQUIRED	This is macro HCI_EIR_FEC_NOT_REQUIRED.
	HCI_EIR_FEC_REQUIRED	This is macro HCI_EIR_FEC_REQUIRED.
	HCI_EIR_TYPE_DEVICE_ID	This is macro HCI_EIR_TYPE_DEVICE_ID.
	HCI_EIR_TYPE_FLAGS	This is macro HCI_EIR_TYPE_FLAGS.
	HCI_EIR_TYPE_LOCAL_NAME_COMPLETE	This is macro HCI_EIR_TYPE_LOCAL_NAME_COMPLETE.
	HCI_EIR_TYPE_LOCAL_NAME_SHORTENED	This is macro HCI_EIR_TYPE_LOCAL_NAME_SHORTENED.
	HCI_EIR_TYPE_MANUFACTURER_SPECIFIC	This is macro HCI_EIR_TYPE_MANUFACTURER_SPECIFIC.
	HCI_EIR_TYPE_OOB_COD	This is macro HCI_EIR_TYPE_OOB_COD.

	HCI_EIR_TYPE_OOB_HASH	This is macro HCI_EIR_TYPE_OOB_HASH.
	HCI_EIR_TYPE_OOB_RANDOMIZER	This is macro HCI_EIR_TYPE_OOB_RANDOMIZER.
	HCI_EIR_TYPE_TX_POWER_LEVEL	This is macro HCI_EIR_TYPE_TX_POWER_LEVEL.
	HCI_EIR_TYPE_UUID128_LIST_COMPLETE	This is macro HCI_EIR_TYPE_UUID128_LIST_COMPLETE.
	HCI_EIR_TYPE_UUID128_LIST_MORE_AVAILABLE	This is macro HCI_EIR_TYPE_UUID128_LIST_MORE_AVAILABLE.
	HCI_EIR_TYPE_UUID16_LIST_COMPLETE	This is macro HCI_EIR_TYPE_UUID16_LIST_COMPLETE.
	HCI_EIR_TYPE_UUID16_LIST_MORE_AVAILABLE	This is macro HCI_EIR_TYPE_UUID16_LIST_MORE_AVAILABLE.
	HCI_EIR_TYPE_UUID32_LIST_COMPLETE	This is macro HCI_EIR_TYPE_UUID32_LIST_COMPLETE.
	HCI_EIR_TYPE_UUID32_LIST_MORE_AVAILABLE	This is macro HCI_EIR_TYPE_UUID32_LIST_MORE_AVAILABLE.
	HCI_ENABLE_DEVICE_UNDER_TEST_MODE	This is macro HCI_ENABLE_DEVICE_UNDER_TEST_MODE.
	HCI_ENCRYPTION_OFF	This is macro HCI_ENCRYPTION_OFF.
	HCI_ENCRYPTION_ON	This is macro HCI_ENCRYPTION_ON.
	HCI_ENHANCED_FLUSH	This is macro HCI_ENHANCED_FLUSH.
	HCI_ERR_ACL_CONN_ALREADY_EXISTS	This is macro HCI_ERR_ACL_CONN_ALREADY_EXISTS.
	HCI_ERR_AUTHENTICATION_FAILURE	This is macro HCI_ERR_AUTHENTICATION_FAILURE.
	HCI_ERR_CONN_REJECT_LIMITED_RESOURCES	This is macro HCI_ERR_CONN_REJECT_LIMITED_RESOURCES.
	HCI_ERR_INVALID_PARAMETERS	This is macro HCI_ERR_INVALID_PARAMETERS.
	HCI_ERR_MEMORY_CAPACITY_EXCEEDED	This is macro HCI_ERR_MEMORY_CAPACITY_EXCEEDED.
	HCI_ERR_SCO_CONN_LIMIT_EXCEEDED	This is macro HCI_ERR_SCO_CONN_LIMIT_EXCEEDED.
	HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED	This is macro HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED.
	HCI_ERR_SUCCESS	• addtogroup hci • @{ • @name Errors
	HCI_ERR_UNSPECIFIED	This is macro HCI_ERR_UNSPECIFIED.
	HCI_EVT_ALL_HCI_EVENTS	Used in bt_hci_ctrl_register_event_listener to indicate that the listener is called for all HCI events (excluding dotstack internal ones)
	HCI_EVT_AUTHENTICATION_COMPLETE	This is macro HCI_EVT_AUTHENTICATION_COMPLETE.
	HCI_EVT_CHANGE_CONN_LINK_COMPLETE	This is macro HCI_EVT_CHANGE_CONN_LINK_COMPLETE.
	HCI_EVT_CMD_SEND_FINISHED	This is macro HCI_EVT_CMD_SEND_FINISHED.
	HCI_EVT_CMD_SEND_STARTED	dotstack internal events
	HCI_EVT_COMMAND_COMPLETE	This is macro HCI_EVT_COMMAND_COMPLETE.
	HCI_EVT_COMMAND_COMPLETE_PARAM_LEN	This is macro HCI_EVT_COMMAND_COMPLETE_PARAM_LEN.
	HCI_EVT_COMMAND_STATUS	This is macro HCI_EVT_COMMAND_STATUS.
	HCI_EVT_CONN_PACKET_TYPE_CHANGED	This is macro HCI_EVT_CONN_PACKET_TYPE_CHANGED.

	HCI_EVT_CONNECTION_COMPLETE	This is macro HCI_EVT_CONNECTION_COMPLETE.
	HCI_EVT_CONNECTION_REQUEST	This is macro HCI_EVT_CONNECTION_REQUEST.
	HCI_EVT_DATA_BUFFER_OVERFLOW	This is macro HCI_EVT_DATA_BUFFER_OVERFLOW.
	HCI_EVT_DISCONNECTION_COMPLETE	This is macro HCI_EVT_DISCONNECTION_COMPLETE.
	HCI_EVT_ENCRYPTION_CHANGE	This is macro HCI_EVT_ENCRYPTION_CHANGE.
	HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE	This is macro HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE.
	HCI_EVT_ENHANCED_FLUSH_COMPLETE	This is macro HCI_EVT_ENHANCED_FLUSH_COMPLETE.
	HCI_EVT_EXTENDED_INQUIRY_RESULT	This is macro HCI_EVT_EXTENDED_INQUIRY_RESULT.
	HCI_EVT_FIRST	This is macro HCI_EVT_FIRST.
	HCI_EVT_FLOW_SPECIFICATION_COMPLETE	This is macro HCI_EVT_FLOW_SPECIFICATION_COMPLETE.
	HCI_EVT_FLUSH_OCCURED	This is macro HCI_EVT_FLUSH_OCCURED.
	HCI_EVT_HARDWARE_ERROR	This is macro HCI_EVT_HARDWARE_ERROR.
	HCI_EVT_INQUIRY_COMPLETE	• addtogroup hci • @{ • @name Events
	HCI_EVT_INQUIRY_RESULT	This is macro HCI_EVT_INQUIRY_RESULT.
	HCI_EVT_INQUIRY_RESULT_WITH_RSSI	This is macro HCI_EVT_INQUIRY_RESULT_WITH_RSSI.
	HCI_EVT_IO_CAPABILITY_REQUEST	This is macro HCI_EVT_IO_CAPABILITY_REQUEST.
	HCI_EVT_IO_CAPABILITY_RESPONSE	This is macro HCI_EVT_IO_CAPABILITY_RESPONSE.
	HCI_EVT_KEYPRESS_NOTIFICATION	This is macro HCI_EVT_KEYPRESS_NOTIFICATION.
	HCI_EVT_LAST	This is macro HCI_EVT_LAST.
	HCI_EVT_LE_META_EVENT	This is macro HCI_EVT_LE_META_EVENT.
	HCI_EVT_LINK_IS_BUSY	This is macro HCI_EVT_LINK_IS_BUSY.
	HCI_EVT_LINK_IS_IDLE	This is macro HCI_EVT_LINK_IS_IDLE.
	HCI_EVT_LINK_KEY_NOTIFICATION	This is macro HCI_EVT_LINK_KEY_NOTIFICATION.
	HCI_EVT_LINK_KEY_REQUEST	This is macro HCI_EVT_LINK_KEY_REQUEST.
	HCI_EVT_LINK_SUPERVISION_TO_CHANGED	This is macro HCI_EVT_LINK_SUPERVISION_TO_CHANGED.
	HCI_EVT_LOOPBACK_COMMAND	This is macro HCI_EVT_LOOPBACK_COMMAND.
	HCI_EVT_MASTER_LINK_KEY_COMPLETE	This is macro HCI_EVT_MASTER_LINK_KEY_COMPLETE.
	HCI_EVT_MAX_SLOTS_CHANGE	This is macro HCI_EVT_MAX_SLOTS_CHANGE.
	HCI_EVT_MODE_CHANGE	This is macro HCI_EVT_MODE_CHANGE.
	HCI_EVT_NUM_OF_COMPLETED_PACKETS	This is macro HCI_EVT_NUM_OF_COMPLETED_PACKETS.
	HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE	This is macro HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE.
	HCI_EVT_PIN_CODE_REQUEST	This is macro HCI_EVT_PIN_CODE_REQUEST.
	HCI_EVT_QOS_SETUP_COMPLETE	This is macro HCI_EVT_QOS_SETUP_COMPLETE.
	HCI_EVT_QOS_VIOLATION	This is macro HCI_EVT_QOS_VIOLATION.

	HCI_EVT_READ_CLOCK_OFFSET_COMPLETE	This is macro HCI_EVT_READ_CLOCK_OFFSET_COMPLETE.
	HCI_EVT_READ_RMT_EXT_FEATURES_COMP	This is macro HCI_EVT_READ_RMT_EXT_FEATURES_COMP.
	HCI_EVT_READ_RMT_SUP_FEATURES_COMP	This is macro HCI_EVT_READ_RMT_SUP_FEATURES_COMP.
	HCI_EVT_READ_RMT_VERSION_INFO_COMP	This is macro HCI_EVT_READ_RMT_VERSION_INFO_COMP.
	HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE	This is macro HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE.
	HCI_EVT_REMOTE_OOB_DATA_REQUEST	This is macro HCI_EVT_REMOTE_OOB_DATA_REQUEST.
	HCI_EVT_RETURN_LINK_KEYS	This is macro HCI_EVT_RETURN_LINK_KEYS.
	HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF	This is macro HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF.
	HCI_EVT_ROLE_CHANGE	This is macro HCI_EVT_ROLE_CHANGE.
	HCI_EVT_SIMPLE_PAIRING_COMPLETE	This is macro HCI_EVT_SIMPLE_PAIRING_COMPLETE.
	HCI_EVT_SNIFF_SUBRATING	This is macro HCI_EVT_SNIFF_SUBRATING.
	HCI_EVT_SYNCH_CONNECTION_CHANGED	This is macro HCI_EVT_SYNCH_CONNECTION_CHANGED.
	HCI_EVT_SYNCH_CONNECTION_COMPLETE	This is macro HCI_EVT_SYNCH_CONNECTION_COMPLETE.
	HCI_EVT_USER_CONFIRMATION_REQUEST	This is macro HCI_EVT_USER_CONFIRMATION_REQUEST.
	HCI_EVT_USER_PASSKEY_NOTIFICATION	This is macro HCI_EVT_USER_PASSKEY_NOTIFICATION.
	HCI_EVT_USER_PASSKEY_REQUEST	This is macro HCI_EVT_USER_PASSKEY_REQUEST.
	HCI_EXIT_PARK_STATE	This is macro HCI_EXIT_PARK_STATE.
	HCI_EXIT_PERIODIC_INQUIRY_MODE	This is macro HCI_EXIT_PERIODIC_INQUIRY_MODE.
	HCI_EXIT_SNIFF_MODE	This is macro HCI_EXIT_SNIFF_MODE.
	HCI_FLOW_SPECIFICATION	This is macro HCI_FLOW_SPECIFICATION.
	HCI_FLUSH	This is macro HCI_FLUSH.
	HCI_H2C_BROADCAST_ACTIVE_SLAVE	This is macro HCI_H2C_BROADCAST_ACTIVE_SLAVE.
	HCI_H2C_BROADCAST_NO_BROADCATS	This is macro HCI_H2C_BROADCAST_NO_BROADCATS.
	HCI_H2C_BROADCAST_PARCKED_SLAVE	This is macro HCI_H2C_BROADCAST_PARCKED_SLAVE.
	HCI_H2C_BROADCAST_RESERVED	This is macro HCI_H2C_BROADCAST_RESERVED.
	HCI_HOLD_MODE	addtogroup hci @{ @name Link policy commands details The Link Policy Commands provide methods for the Host to affect how the Link Manager manages the piconet.
	HCI_HOST_BUFFER_SIZE	This is macro HCI_HOST_BUFFER_SIZE.
	HCI_HOST_NUM_OF_COMPLETED_PACKETS	This is macro HCI_HOST_NUM_OF_COMPLETED_PACKETS.
	HCI_INQUIRY	addtogroup hci @{ @name Link control commands details The Link Control commands allow a Controller to control connections to other BR/EDR Controllers.

	HCI_INQUIRY_CANCEL	This is macro HCI_INQUIRY_CANCEL.
	HCI_INQUIRY_MODE_EXTENDED	This is macro HCI_INQUIRY_MODE_EXTENDED.
	HCI_INQUIRY_MODE_STANDARD	This is macro HCI_INQUIRY_MODE_STANDARD.
	HCI_INQUIRY_MODE_WITH_RSSI	This is macro HCI_INQUIRY_MODE_WITH_RSSI.
	HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY	This is macro HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY.
	HCI_IO_CAPABILITY_REQUEST_REPLY	This is macro HCI_IO_CAPABILITY_REQUEST_REPLY.
	HCI_LE_ADD_DEVICE_TO_WHITE_LIST	This is macro HCI_LE_ADD_DEVICE_TO_WHITE_LIST.
	HCI_LE_ADDRESS_TYPE_PUBLIC	This is macro HCI_LE_ADDRESS_TYPE_PUBLIC.
	HCI_LE_ADDRESS_TYPE_RANDOM	This is macro HCI_LE_ADDRESS_TYPE_RANDOM.
	HCI_LE_ADVERTISING_FLAG_BREDR_NOT_SUPPORTED	This is macro HCI_LE_ADVERTISING_FLAG_BREDR_NOT_SUPPORTED.
	HCI_LE_ADVERTISING_FLAG_GENERAL_DISCOVERABLE_MODE	This is macro HCI_LE_ADVERTISING_FLAG_GENERAL_DISCOVERABLE_MODE.
	HCI_LE_ADVERTISING_FLAG_LIMITED_DISCOVERABLE_MODE	This is macro HCI_LE_ADVERTISING_FLAG_LIMITED_DISCOVERABLE_MODE.
	HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_CONTROLLER	This is macro HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_CONTROLLER.
	HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_HOST	This is macro HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_HOST.
	HCI_LE_AVERTISING_DISABLED	This is macro HCI_LE_AVERTISING_DISABLED.
	HCI_LE_AVERTISING_ENABLED	This is macro HCI_LE_AVERTISING_ENABLED.
	HCI_LE_CLEAR_WHITE_LIST	This is macro HCI_LE_CLEAR_WHITE_LIST.
	HCI_LE_CONNECTION_UPDATE	This is macro HCI_LE_CONNECTION_UPDATE.
	HCI_LE_CREATE_CONNECTION	This is macro HCI_LE_CREATE_CONNECTION.
	HCI_LE_CREATE_CONNECTION_CANCEL	This is macro HCI_LE_CREATE_CONNECTION_CANCEL.
	HCI_LE_DISABLED	This is macro HCI_LE_DISABLED.
	HCI_LE_DUPLICATE_FILTERING_DISABLED	This is macro HCI_LE_DUPLICATE_FILTERING_DISABLED.
	HCI_LE_DUPLICATE_FILTERING_ENABLED	This is macro HCI_LE_DUPLICATE_FILTERING_ENABLED.
	HCI_LE_ENABLED	This is macro HCI_LE_ENABLED.
	HCI_LE_ENCRYPT	This is macro HCI_LE_ENCRYPT.
	HCI_LE_EVT_ADVERTISING_REPORT	This is macro HCI_LE_EVT_ADVERTISING_REPORT.
	HCI_LE_EVT_CONNECTION_COMPLETE	This is macro HCI_LE_EVT_CONNECTION_COMPLETE.
	HCI_LE_EVT_CONNECTION_UPDATE_COMPLETE	This is macro HCI_LE_EVT_CONNECTION_UPDATE_COMPLETE.
	HCI_LE_EVT_LONG_TERM_KEY_REQUEST	This is macro HCI_LE_EVT_LONG_TERM_KEY_REQUEST.
	HCI_LE_EVT_READ_REMOTE_USED_FEATURES_COMPLETE	This is macro HCI_LE_EVT_READ_REMOTE_USED_FEATURES_COMPLETE.
	HCI_LE_FILTER_POLICY_NOT_USED	White list is not used to determine which advertiser to connect to. Peer_Address_Type and Peer_Address shall be used.

	HCI_LE_FILTER_POLICY_WHITE_LIST	White list is used to determine which advertiser to connect to. Peer_Address_Type and Peer_Address shall be ignored
	HCI_LE_FLAG_SHARED_ACL_BUFFERS	This is macro HCI_LE_FLAG_SHARED_ACL_BUFFERS.
	HCI_LE_FLAG_SIMULTANEOUS_LE_BREDR	This is macro HCI_LE_FLAG_SIMULTANEOUS_LE_BREDR.
	HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY	This is macro HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY.
	HCI_LE_LONG_TERM_KEY_REQUEST_REPLY	This is macro HCI_LE_LONG_TERM_KEY_REQUEST_REPLY.
	HCI_LE_MAX_AD_LEN	This is macro HCI_LE_MAX_AD_LEN.
	HCI_LE_RAND	This is macro HCI_LE_RAND.
	HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER	This is macro HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER.
	HCI_LE_READ_BUFFER_SIZE	This is macro HCI_LE_READ_BUFFER_SIZE.
	HCI_LE_READ_CHANNEL_MAP	This is macro HCI_LE_READ_CHANNEL_MAP.
	HCI_LE_READ_LOCAL_SUPPORTED_FEATURES	This is macro HCI_LE_READ_LOCAL_SUPPORTED_FEATURES.
	HCI_LE_READ_REMOTE_USED_FEATURES	This is macro HCI_LE_READ_REMOTE_USED_FEATURES.
	HCI_LE_READ_SUPPORTED_STATES	This is macro HCI_LE_READ_SUPPORTED_STATES.
	HCI_LE_READ_WHITE_LIST_SIZE	This is macro HCI_LE_READ_WHITE_LIST_SIZE.
	HCI_LE_RECEIVE_TEST	This is macro HCI_LE_RECEIVE_TEST.
	HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST	This is macro HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST.
	HCI_LE_SCAN_DISABLED	This is macro HCI_LE_SCAN_DISABLED.
	HCI_LE_SCAN_ENABLED	This is macro HCI_LE_SCAN_ENABLED.
	HCI_LE_SCAN_EVT_CANCEL_FAILED	This is macro HCI_LE_SCAN_EVT_CANCEL_FAILED.
	HCI_LE_SCAN_EVT_CANCELLED	This is macro HCI_LE_SCAN_EVT_CANCELLED.
	HCI_LE_SCAN_EVT_DEVICE_FOUND	This is macro HCI_LE_SCAN_EVT_DEVICE_FOUND.
	HCI_LE_SCAN_EVT_START_FAILED	This is macro HCI_LE_SCAN_EVT_START_FAILED.
	HCI_LE_SCAN_EVT_STARTED	This is macro HCI_LE_SCAN_EVT_STARTED.
	HCI_LE_SCAN_FILTER_POLICY_ALL	This is macro HCI_LE_SCAN_FILTER_POLICY_ALL.
	HCI_LE_SCAN_FILTER_POLICY_WHITE_LIST	This is macro HCI_LE_SCAN_FILTER_POLICY_WHITE_LIST.
	HCI_LE_SCAN_TYPE_ACTIVE	This is macro HCI_LE_SCAN_TYPE_ACTIVE.
	HCI_LE_SCAN_TYPE_PASSIVE	This is macro HCI_LE_SCAN_TYPE_PASSIVE.
	HCI_LE_SET_ADVERTISE_ENABLE	This is macro HCI_LE_SET_ADVERTISE_ENABLE.
	HCI_LE_SET_ADVERTISING_DATA	This is macro HCI_LE_SET_ADVERTISING_DATA.
	HCI_LE_SET_ADVERTISING_PARAMETERS	This is macro HCI_LE_SET_ADVERTISING_PARAMETERS.
	HCI_LE_SET_EVENT_MASK	addtogroup hci @{ @name LE controller commands details The LE Controller Commands provide access and control to various capabilities of the Bluetooth hardware, as well as methods for the Host to affect how the Link Layer manages the piconet, and controls connections.

	HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION	This is macro HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION.
	HCI_LE_SET_RANDOM_ADDRESS	This is macro HCI_LE_SET_RANDOM_ADDRESS.
	HCI_LE_SET_SCAN_ENABLE	This is macro HCI_LE_SET_SCAN_ENABLE.
	HCI_LE_SET_SCAN_PARAMETERS	This is macro HCI_LE_SET_SCAN_PARAMETERS.
	HCI_LE_SET_SCAN_RESPONSE_DATA	This is macro HCI_LE_SET_SCAN_RESPONSE_DATA.
	HCI_LE_SIMULTANEOUS_DISABLED	This is macro HCI_LE_SIMULTANEOUS_DISABLED.
	HCI_LE_SIMULTANEOUS_ENABLED	This is macro HCI_LE_SIMULTANEOUS_ENABLED.
	HCI_LE_START_ENCRYPTION	This is macro HCI_LE_START_ENCRYPTION.
	HCI_LE_TEST_END	This is macro HCI_LE_TEST_END.
	HCI_LE_TRANSMITTER_TEST	This is macro HCI_LE_TRANSMITTER_TEST.
	HCI_LINK_KEY_LEN	This is macro HCI_LINK_KEY_LEN.
	HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY	This is macro HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY.
	HCI_LINK_KEY_REQUEST_REPLY	This is macro HCI_LINK_KEY_REQUEST_REPLY.
	HCI_LINK_KEY_TYPE_COMBINATION	This is macro HCI_LINK_KEY_TYPE_COMBINATION.
	HCI_LINK_KEY_TYPE_LOCAL_UNIT	This is macro HCI_LINK_KEY_TYPE_LOCAL_UNIT.
	HCI_LINK_KEY_TYPE_REMOTE_UNIT	This is macro HCI_LINK_KEY_TYPE_REMOTE_UNIT.
	HCI_LINK_POLICY_ENABLE_HOLD_MODE	This is macro HCI_LINK_POLICY_ENABLE_HOLD_MODE.
	HCI_LINK_POLICY_ENABLE_PARK_STATE	This is macro HCI_LINK_POLICY_ENABLE_PARK_STATE.
	HCI_LINK_POLICY_ENABLE_ROLE_SWITCH	This is macro HCI_LINK_POLICY_ENABLE_ROLE_SWITCH.
	HCI_LINK_POLICY_ENABLE_SNIFF_MODE	This is macro HCI_LINK_POLICY_ENABLE_SNIFF_MODE.
	HCI_LINK_TYPE_BD_EDR	This is macro HCI_LINK_TYPE_BD_EDR.
	HCI_LINK_TYPE_LE	This is macro HCI_LINK_TYPE_LE.
	HCI_MASTER_LINK_KEY	This is macro HCI_MASTER_LINK_KEY.
	HCI_MAX_DATA_BUFFERS	This is macro HCI_MAX_DATA_BUFFERS.
	HCI_MAX_EVENT_PARAM_LEN	This is macro HCI_MAX_EVENT_PARAM_LEN.
	HCI_MAX_PARAM_LEN	This is macro HCI_MAX_PARAM_LEN.
	HCI_MAX_PIN_LENGTH	This is macro HCI_MAX_PIN_LENGTH.
	HCI_OPCODE	This is macro HCI_OPCODE.
	HCI_PACKET_BOUNDARY_CONTINUE	This is macro HCI_PACKET_BOUNDARY_CONTINUE.
	HCI_PACKET_BOUNDARY_FIRST	This is macro HCI_PACKET_BOUNDARY_FIRST.
	HCI_PACKET_BOUNDARY_FIRST_AUTO_FLUSH	This is macro HCI_PACKET_BOUNDARY_FIRST_AUTO_FLUSH.
	HCI_PACKET_BOUNDARY_FIRST_NO_AUTO_FLUSH	This is macro HCI_PACKET_BOUNDARY_FIRST_NO_AUTO_FLUSH.
	HCI_PACKET_TYPE_ACL_DATA	This is macro HCI_PACKET_TYPE_ACL_DATA.
	HCI_PACKET_TYPE_COMMAND	This is macro HCI_PACKET_TYPE_COMMAND.
	HCI_PACKET_TYPE_EVENT	This is macro HCI_PACKET_TYPE_EVENT.
	HCI_PACKET_TYPE_NONE	This is macro HCI_PACKET_TYPE_NONE.
	HCI_PACKET_TYPE_SCO_DATA	This is macro HCI_PACKET_TYPE_SCO_DATA.

	HCI_PAGE_SCAN_REPETITION_MODE_R0	This is macro HCI_PAGE_SCAN_REPETITION_MODE_R0.
	HCI_PAGE_SCAN_REPETITION_MODE_R1	This is macro HCI_PAGE_SCAN_REPETITION_MODE_R1.
	HCI_PAGE_SCAN_REPETITION_MODE_R2	This is macro HCI_PAGE_SCAN_REPETITION_MODE_R2.
	HCI_PARAM_LEN_BD_ADDR	This is macro HCI_PARAM_LEN_BD_ADDR.
	HCI_PARAM_LEN_BYTE	This is macro HCI_PARAM_LEN_BYTE.
	HCI_PARAM_LEN_DEV_CLASS	This is macro HCI_PARAM_LEN_DEV_CLASS.
	HCI_PARAM_LEN_HANDLE	This is macro HCI_PARAM_LEN_HANDLE.
	HCI_PARAM_LEN_INT	This is macro HCI_PARAM_LEN_INT.
	HCI_PARAM_LEN_LONG	This is macro HCI_PARAM_LEN_LONG.
	HCI_PARAM_TYPE_BD_ADDR	This is macro HCI_PARAM_TYPE_BD_ADDR.
	HCI_PARAM_TYPE_BYTE	This is macro HCI_PARAM_TYPE_BYTE.
	HCI_PARAM_TYPE_DEV_CLASS	This is macro HCI_PARAM_TYPE_DEV_CLASS.
	HCI_PARAM_TYPE_HANDLE	This is macro HCI_PARAM_TYPE_HANDLE.
	HCI_PARAM_TYPE_INT	This is macro HCI_PARAM_TYPE_INT.
	HCI_PARAM_TYPE_LONG	This is macro HCI_PARAM_TYPE_LONG.
	HCI_PARAM_TYPE_STRING	This is macro HCI_PARAM_TYPE_STRING.
	HCI_PARK_STATE	This is macro HCI_PARK_STATE.
	HCI_PERIODIC_INQUIRY_MODE	This is macro HCI_PERIODIC_INQUIRY_MODE.
	HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY	This is macro HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY.
	HCI_PIN_CODE_REQUEST_REPLY	This is macro HCI_PIN_CODE_REQUEST_REPLY.
	HCI_POWER_MODE_ACTIVE	This is macro HCI_POWER_MODE_ACTIVE.
	HCI_POWER_MODE_HOLD	This is macro HCI_POWER_MODE_HOLD.
	HCI_POWER_MODE_PARK	This is macro HCI_POWER_MODE_PARK.
	HCI_POWER_MODE_SNIFF	This is macro HCI_POWER_MODE_SNIFF.
	HCI_QOS_SETUP	This is macro HCI_QOS_SETUP.
	HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE	This is macro HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE.
	HCI_READ_AFH_CHANNEL_MAP	This is macro HCI_READ_AFH_CHANNEL_MAP.
	HCI_READ_AUTHENTICATION_ENABLE	This is macro HCI_READ_AUTHENTICATION_ENABLE.
	HCI_READ_AUTOMATIC_FLASH_TIMEOUT	This is macro HCI_READ_AUTOMATIC_FLASH_TIMEOUT.
	HCI_READ_BD_ADDR	This is macro HCI_READ_BD_ADDR.
	HCI_READ_BUFFER_SIZE	This is macro HCI_READ_BUFFER_SIZE.
	HCI_READ_CLASS_OF_DEVICE	This is macro HCI_READ_CLASS_OF_DEVICE.
	HCI_READ_CLOCK_COMMAND	This is macro HCI_READ_CLOCK_COMMAND.
	HCI_READ_CLOCK_OFFSET	This is macro HCI_READ_CLOCK_OFFSET.
	HCI_READ_CONNECTION_ACCEPT_TIMEOUT	This is macro HCI_READ_CONNECTION_ACCEPT_TIMEOUT.
	HCI_READ_CURRENT_IAC_LAP	This is macro HCI_READ_CURRENT_IAC_LAP.
	HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING	This is macro HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING.
	HCI_READ_DEFAULT_POLICY_SETTINGS	This is macro HCI_READ_DEFAULT_POLICY_SETTINGS.
	HCI_READ_ENCRYPTION_MODE	This is macro HCI_READ_ENCRYPTION_MODE.
	HCI_READ_EXTENDED_INQUIRY_RESPONSE	This is macro HCI_READ_EXTENDED_INQUIRY_RESPONSE.

	HCI_READ_FAILED_CONTACT_COUNTER	addtogroup hci @{ @name Status Parameters details The Controller modifies all status parameters. These parameters provide information about the current state of the Link Manager and Baseband in the BR/EDR Controller and the PAL in an AMP Controller. The host device cannot modify any of these parameters other than to reset certain specific parameters.
	HCI_READ_HOLD_MODE_ACTIVITY	This is macro HCI_READ_HOLD_MODE_ACTIVITY.
	HCI_READ_INQUIRY_MODE	This is macro HCI_READ_INQUIRY_MODE.
	HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL	This is macro HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL.
	HCI_READ_INQUIRY_SCAN_ACTIVITY	This is macro HCI_READ_INQUIRY_SCAN_ACTIVITY.
	HCI_READ_INQUIRY_SCAN_TYPE	This is macro HCI_READ_INQUIRY_SCAN_TYPE.
	HCI_READ_LE_HOST_SUPPORT	This is macro HCI_READ_LE_HOST_SUPPORT.
	HCI_READ_LINK_POLICY_SETTINGS	This is macro HCI_READ_LINK_POLICY_SETTINGS.
	HCI_READ_LINK_QUALITY	This is macro HCI_READ_LINK_QUALITY.
	HCI_READ_LINK_SUPERVISION_TIMEOUT	This is macro HCI_READ_LINK_SUPERVISION_TIMEOUT.
	HCI_READ_LMP_HANDLE	This is macro HCI_READ_LMP_HANDLE.
	HCI_READ_LOCAL_EXTENDED_FEATURES	This is macro HCI_READ_LOCAL_EXTENDED_FEATURES.
	HCI_READ_LOCAL_NAME	This is macro HCI_READ_LOCAL_NAME.
	HCI_READ_LOCAL_OOB_DATA	This is macro HCI_READ_LOCAL_OOB_DATA.
	HCI_READ_LOCAL_SUPPORTED_COMMANDS	This is macro HCI_READ_LOCAL_SUPPORTED_COMMANDS.
	HCI_READ_LOCAL_SUPPORTED_FEATURES	This is macro HCI_READ_LOCAL_SUPPORTED_FEATURES.
	HCI_READ_LOCAL_VERSION_INFORMATION	addtogroup hci @{ @name Informational Parameters details The Informational Parameters are fixed by the manufacturer of the Bluetooth hardware. These parameters provide information about the BR/EDR Controller and the capabilities of the Link Manager and Baseband in the BR/EDR Controller and PAL in the AMP Controller. The host device cannot modify any of these parameters.
	HCI_READ_LOOPBACK_MODE	addtogroup hci @{ @name Testing Commands details The Testing commands are used to provide the ability to test various functional capabilities of the Bluetooth hardware.
	HCI_READ_NUM_BROADCST_RETR	This is macro HCI_READ_NUM_BROADCST_RETR.
	HCI_READ_NUM_OF_SUPPORTED_IAC	This is macro HCI_READ_NUM_OF_SUPPORTED_IAC.
	HCI_READ_PAGE_SCAN_ACTIVITY	This is macro HCI_READ_PAGE_SCAN_ACTIVITY.
	HCI_READ_PAGE_SCAN_PERIOD_MODE	This is macro HCI_READ_PAGE_SCAN_PERIOD_MODE.
	HCI_READ_PAGE_SCAN_TYPE	This is macro HCI_READ_PAGE_SCAN_TYPE.
	HCI_READ_PAGE_TIMEOUT	This is macro HCI_READ_PAGE_TIMEOUT.
	HCI_READ_PIN_TYPE	This is macro HCI_READ_PIN_TYPE.

	HCI_READ_REFRESH_ENCRYPTION_KEY	This is macro HCI_READ_REFRESH_ENCRYPTION_KEY.
	HCI_READ_REMOTE_EXTENDED_FEATURES	This is macro HCI_READ_REMOTE_EXTENDED_FEATURES.
	HCI_READ_REMOTE_SUPPORTED_FEATURES	This is macro HCI_READ_REMOTE_SUPPORTED_FEATURES.
	HCI_READ_REMOTE_VERSION_INFORMATION	This is macro HCI_READ_REMOTE_VERSION_INFORMATION.
	HCI_READ_RSSI	This is macro HCI_READ_RSSI.
	HCI_READ_SCAN_ENABLE	This is macro HCI_READ_SCAN_ENABLE.
	HCI_READ_SIMPLE_PAIRING_MODE	This is macro HCI_READ_SIMPLE_PAIRING_MODE.
	HCI_READ_STORED_LINK_KEY	This is macro HCI_READ_STORED_LINK_KEY.
	HCI_READ_SYNC_FLOW_CONTROL_ENABLE	This is macro HCI_READ_SYNC_FLOW_CONTROL_ENABLE.
	HCI_READ_TRANSMIT_POWER_LEVEL	This is macro HCI_READ_TRANSMIT_POWER_LEVEL.
	HCI_READ_VOICE_SETTING	This is macro HCI_READ_VOICE_SETTING.
	HCI_REJECT_CONNECTION_REQUEST	This is macro HCI_REJECT_CONNECTION_REQUEST.
	HCI_REJECT_SYNCH_CONNECTION_REQUEST	This is macro HCI_REJECT_SYNCH_CONNECTION_REQUEST.
	HCI_REMOTE_NAME_REQUEST	This is macro HCI_REMOTE_NAME_REQUEST.
	HCI_REMOTE_NAME_REQUEST_CANCEL	This is macro HCI_REMOTE_NAME_REQUEST_CANCEL.
	HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY	This is macro HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY.
	HCI_REMOTE_OOB_DATA_REQUEST_REPLY	This is macro HCI_REMOTE_OOB_DATA_REQUEST_REPLY.
	HCI_RESET	This is macro HCI_RESET.
	HCI_RESET_FAILED_CONTACT_COUNTER	This is macro HCI_RESET_FAILED_CONTACT_COUNTER.
	HCI_ROLE_DISCOVERY	This is macro HCI_ROLE_DISCOVERY.
	HCI_ROLE_SWITCH_ALLOW	This is macro HCI_ROLE_SWITCH_ALLOW.
	HCI_ROLE_SWITCH_DISALLOW	This is macro HCI_ROLE_SWITCH_DISALLOW.
	HCI_SCAN_INQUIRY	Flags for HCI_READ_SCAN_ENABLE/HCI_WRITE_SCAN_ENABLE
	HCI_SCAN_PAGE	This is macro HCI_SCAN_PAGE.
	HCI_SCO_CONTENT_FORMAT_AIR_CODING_A_LAW	This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_A_LAW.
	HCI_SCO_CONTENT_FORMAT_AIR_CODING_CSVD	This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_CSVD.
	HCI_SCO_CONTENT_FORMAT_AIR_CODING_N_LAW	This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_N_LAW.
	HCI_SCO_CONTENT_FORMAT_AIR_CODING_TRANSPARENT	This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_TRANSPARENT.
	HCI_SCO_CONTENT_FORMAT_INPUT_CODING_A_LAW	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_A_LAW.
	HCI_SCO_CONTENT_FORMAT_INPUT_CODING_LINEAR	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_LINEAR.

	HCI_SCO_CONTENT_FORMAT_INPUT_CODING_N_LAW	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_N_LAW.
	HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_1_COMPLEMENT	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_1_COMPLEMENT.
	HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_2_COMPLEMENT	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_2_COMPLEMENT.
	HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_SIGN_MAGNITUDE	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_SIGN_MAGNITUDE.
	HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_UNSIGNED	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_UNSIGNED.
	HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_16	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_16.
	HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_8	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_8.
	HCI_SCO_DATA_HEADER_LEN	This is macro HCI_SCO_DATA_HEADER_LEN.
	HCI_SCO_MAX_DATA_LEN	This is macro HCI_SCO_MAX_DATA_LEN.
	HCI_SCO_MAX_LATENCY_DONTCARE	This is macro HCI_SCO_MAX_LATENCY_DONTCARE.
	HCI_SCO_PACKET_TYPE_ALL	This is macro HCI_SCO_PACKET_TYPE_ALL.
	HCI_SCO_PACKET_TYPE_EV3	This is macro HCI_SCO_PACKET_TYPE_EV3.
	HCI_SCO_PACKET_TYPE_EV4	This is macro HCI_SCO_PACKET_TYPE_EV4.
	HCI_SCO_PACKET_TYPE_EV5	This is macro HCI_SCO_PACKET_TYPE_EV5.
	HCI_SCO_PACKET_TYPE_HV1	This is macro HCI_SCO_PACKET_TYPE_HV1.
	HCI_SCO_PACKET_TYPE_HV2	This is macro HCI_SCO_PACKET_TYPE_HV2.
	HCI_SCO_PACKET_TYPE_HV3	This is macro HCI_SCO_PACKET_TYPE_HV3.
	HCI_SCO_PACKET_TYPE_NO_2_EV3	This is macro HCI_SCO_PACKET_TYPE_NO_2_EV3.
	HCI_SCO_PACKET_TYPE_NO_2_EV5	This is macro HCI_SCO_PACKET_TYPE_NO_2_EV5.
	HCI_SCO_PACKET_TYPE_NO_3_EV3	This is macro HCI_SCO_PACKET_TYPE_NO_3_EV3.
	HCI_SCO_PACKET_TYPE_NO_3_EV5	This is macro HCI_SCO_PACKET_TYPE_NO_3_EV5.
	HCI_SCO_RTX_EFFORT_DONTCARE	This is macro HCI_SCO_RTX_EFFORT_DONTCARE.
	HCI_SCO_RTX_EFFORT_NO_RETRANSMISSION	This is macro HCI_SCO_RTX_EFFORT_NO_RETRANSMISSION.
	HCI_SCO_RTX_EFFORT_OPTIMIZE_LINK_QUALITY	This is macro HCI_SCO_RTX_EFFORT_OPTIMIZE_LINK_QUALITY.
	HCI_SCO_RTX_EFFORT_OPTIMIZE_POWER_CONSUMPTION	This is macro HCI_SCO_RTX_EFFORT_OPTIMIZE_POWER_CONSUMPTION.
	HCI_SCO_RX_BANDWIDTH_DONTCARE	This is macro HCI_SCO_RX_BANDWIDTH_DONTCARE.
	HCI_SCO_TX_BANDWIDTH_DONTCARE	This is macro HCI_SCO_TX_BANDWIDTH_DONTCARE.
	HCI_SEND_DATA_STATUS_INTERRUPTED	This is macro HCI_SEND_DATA_STATUS_INTERRUPTED.
	HCI_SEND_DATA_STATUS_SUCCESS	This is macro HCI_SEND_DATA_STATUS_SUCCESS.
	HCI_SEND_KEY_PRESS_NOTIFICATION	This is macro HCI_SEND_KEY_PRESS_NOTIFICATION.
	HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION	This is macro HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION.

	HCI_SET_CONNECTION_ENCRYPTION	This is macro HCI_SET_CONNECTION_ENCRYPTION.
	HCI_SET_CTRL_TO_HOST_FLOW_CONTROL	This is macro HCI_SET_CTRL_TO_HOST_FLOW_CONTROL.
	HCI_SET_EVENT_FILTER	This is macro HCI_SET_EVENT_FILTER.
	HCI_SET_EVENT_MASK	addtogroup hci @{ @name Controller & Baseband commands details The Controller & Baseband Commands provide access and control to various capabilities of the Bluetooth hardware.
	HCI_SETUP_SYNCHRONOUS_CONNECTION	This is macro HCI_SETUP_SYNCHRONOUS_CONNECTION.
	HCI_SNIFF_MODE	This is macro HCI_SNIFF_MODE.
	HCI_SNIFF_SUBRATING	This is macro HCI_SNIFF_SUBRATING.
	HCI_SWITCH_ROLE	This is macro HCI_SWITCH_ROLE.
	HCI_TRANSPORT_HEADER_LEN	This is macro HCI_TRANSPORT_HEADER_LEN.
	HCI_UART_PACKET_TYPE_ACL_DATA	This is macro HCI_UART_PACKET_TYPE_ACL_DATA.
	HCI_UART_PACKET_TYPE_COMMAND	defgroup hcitr_uart HCI UART (H4) transport protocol ingroup hcitr details This module describes functions used to initialize and start HCI UART transport protocol. The transport uses common interface for exchanging data between the host CPU and HCI controller defined in bt_hcitr.h. This interface consist of two functions that must be implemented by the application: li bt_oem_send() li bt_oem_recv()
	HCI_UART_PACKET_TYPE_EVENT	This is macro HCI_UART_PACKET_TYPE_EVENT.
	HCI_UART_PACKET_TYPE_SCO_DATA	This is macro HCI_UART_PACKET_TYPE_SCO_DATA.
	HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY	This is macro HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY.
	HCI_USER_CONFIRMATION_REQUEST_REPLY	This is macro HCI_USER_CONFIRMATION_REQUEST_REPLY.
	HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY	This is macro HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY.
	HCI_USER_PASSKEY_REQUEST_REPLY	This is macro HCI_USER_PASSKEY_REQUEST_REPLY.
	HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE	This is macro HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE.
	HCI_WRITE_AUTHENTICATION_ENABLE	This is macro HCI_WRITE_AUTHENTICATION_ENABLE.
	HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT	This is macro HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT.
	HCI_WRITE_CLASS_OF_DEVICE	This is macro HCI_WRITE_CLASS_OF_DEVICE.
	HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT	This is macro HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT.
	HCI_WRITE_CURRENT_IAC_LAP	This is macro HCI_WRITE_CURRENT_IAC_LAP.
	HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING	This is macro HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING.
	HCI_WRITE_DEFAULT_POLICY_SETTINGS	This is macro HCI_WRITE_DEFAULT_POLICY_SETTINGS.
	HCI_WRITE_ENCRYPTION_MODE	This is macro HCI_WRITE_ENCRYPTION_MODE.

	HCI_WRITE_EXTENDED_INQUIRY_RESPONSE	This is macro HCI_WRITE_EXTENDED_INQUIRY_RESPONSE.
	HCI_WRITE_EXTENDED_INQUIRY_RESPONSE_PARAM_LEN	This is macro HCI_WRITE_EXTENDED_INQUIRY_RESPONSE_PARAM_LEN.
	HCI_WRITE_HOLD_MODE_ACTIVITY	This is macro HCI_WRITE_HOLD_MODE_ACTIVITY.
	HCI_WRITE_INQUIRY_MODE	This is macro HCI_WRITE_INQUIRY_MODE.
	HCI_WRITE_INQUIRY_SCAN_ACTIVITY	This is macro HCI_WRITE_INQUIRY_SCAN_ACTIVITY.
	HCI_WRITE_INQUIRY_SCAN_TYPE	This is macro HCI_WRITE_INQUIRY_SCAN_TYPE.
	HCI_WRITE_INQUIRY_TX_POWER_LEVEL	This is macro HCI_WRITE_INQUIRY_TX_POWER_LEVEL.
	HCI_WRITE_LE_HOST_SUPPORT	This is macro HCI_WRITE_LE_HOST_SUPPORT.
	HCI_WRITE_LINK_POLICY_SETTINGS	This is macro HCI_WRITE_LINK_POLICY_SETTINGS.
	HCI_WRITE_LINK_SUPERVISION_TIMEOUT	This is macro HCI_WRITE_LINK_SUPERVISION_TIMEOUT.
	HCI_WRITE_LOCAL_NAME	This is macro HCI_WRITE_LOCAL_NAME.
	HCI_WRITE_LOCAL_NAME_PARAM_LEN	This is macro HCI_WRITE_LOCAL_NAME_PARAM_LEN.
	HCI_WRITE_LOOPBACK_MODE	This is macro HCI_WRITE_LOOPBACK_MODE.
	HCI_WRITE_NUM_BROADCST_RETR	This is macro HCI_WRITE_NUM_BROADCST_RETR.
	HCI_WRITE_PAGE_SCAN_ACTIVITY	This is macro HCI_WRITE_PAGE_SCAN_ACTIVITY.
	HCI_WRITE_PAGE_SCAN_PERIOD_MODE	This is macro HCI_WRITE_PAGE_SCAN_PERIOD_MODE.
	HCI_WRITE_PAGE_SCAN_TYPE	This is macro HCI_WRITE_PAGE_SCAN_TYPE.
	HCI_WRITE_PAGE_TIMEOUT	This is macro HCI_WRITE_PAGE_TIMEOUT.
	HCI_WRITE_PIN_TYPE	This is macro HCI_WRITE_PIN_TYPE.
	HCI_WRITE_SCAN_ENABLE	This is macro HCI_WRITE_SCAN_ENABLE.
	HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE	This is macro HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE.
	HCI_WRITE_SIMPLE_PAIRING_MODE	This is macro HCI_WRITE_SIMPLE_PAIRING_MODE.
	HCI_WRITE_STORED_LINK_KEY	This is macro HCI_WRITE_STORED_LINK_KEY.
	HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE	This is macro HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE.
	HCI_WRITE_VOICE_SETTING	This is macro HCI_WRITE_VOICE_SETTING.
	HCITR_3WIRE_DEFAULT_ACK_TIMEOUT	This is macro HCITR_3WIRE_DEFAULT_ACK_TIMEOUT.
	HCITR_BCSP_DEFAULT_ACK_TIMEOUT	This is macro HCITR_BCSP_DEFAULT_ACK_TIMEOUT.
	LM_PACKET_TYPE_DH1	This is macro LM_PACKET_TYPE_DH1.
	LM_PACKET_TYPE_DH3	This is macro LM_PACKET_TYPE_DH3.
	LM_PACKET_TYPE_DH5	This is macro LM_PACKET_TYPE_DH5.
	LM_PACKET_TYPE_DM1	This is macro LM_PACKET_TYPE_DM1.
	LM_PACKET_TYPE_DM3	This is macro LM_PACKET_TYPE_DM3.
	LM_PACKET_TYPE_DM5	This is macro LM_PACKET_TYPE_DM5.
	OGF_CTRL_BASEBAND	This is macro OGF_CTRL_BASEBAND.
	OGF_INFORMATION	This is macro OGF_INFORMATION.
	OGF_LE	This is macro OGF_LE.
	OGF_LINK_CONTROL	This is macro OGF_LINK_CONTROL.
	OGF_LINK_POLICY	This is macro OGF_LINK_POLICY.

	OGF_STATUS	This is macro OGF_STATUS.
	OGF_TESTING	This is macro OGF_TESTING.
	OGF_VENDOR	This is macro OGF_VENDOR.
	bt_hci_cmd_callback_fp	This is type bt_hci_cmd_callback_fp.
	bt_hci_cmd_listener_fp	This is type bt_hci_cmd_listener_fp.
	bt_hci_command_p	This is type bt_hci_command_p.
	bt_hci_command_t	This is type bt_hci_command_t.
	bt_hci_conn_state_p	This is type bt_hci_conn_state_p.
	bt_hci_conn_state_t	This is type bt_hci_conn_state_t.
	bt_hci_connect_callback_fp	brief HCI connect callback. ingroup hci details This typedef defines a type for the callback function that is called when HCI connect operation initiated by a call to bt_hci_connect() is complete. param status Operation status. c It is 0 if connection was successfully established. param pconn pointer to a structure representing the established connection. param param pointer to arbitrary data passed to the bt_hci_connect() function through its c param parameter.
	bt_hci_ctrl_listener_t	This is type bt_hci_ctrl_listener_t.
	bt_hci_ctrl_state_t	This is type bt_hci_ctrl_state_t.
	bt_hci_data_buffer_p	This is type bt_hci_data_buffer_p.
	bt_hci_data_buffer_t	This is type bt_hci_data_buffer_t.
	bt_hci_data_callback_fp	This is type bt_hci_data_callback_fp.
	bt_hci_data_listener_fp	This is type bt_hci_data_listener_fp.
	bt_hci_data_p	This is type bt_hci_data_p.
	bt_hci_data_t	This is type bt_hci_data_t.
	bt_hci_disconnect_callback_fp	brief HCI disconnect callback. ingroup hci details This typedef defines a type for the callback function that is called when an HCI connection has been terminated. param status Operation status. c It is 0 if connection has been successfully terminated. param reason Reason for disconnection. param pconn pointer to a structure representing the connection. param param pointer to arbitrary data associated with an event listener.
	bt_hci_event_e	This is type bt_hci_event_e.
	bt_hci_event_handler_ex_fp	This is type bt_hci_event_handler_ex_fp.
	bt_hci_event_handler_fp	This is type bt_hci_event_handler_fp.
	bt_hci_event_listener_fp	This is type bt_hci_event_listener_fp.
	bt_hci_event_p	This is type bt_hci_event_p.
	bt_hci_event_t	This is type bt_hci_event_t.
	bt_hci_evt_authentication_complete_t	This is type bt_hci_evt_authentication_complete_t.
	bt_hci_evt_command_complete_t	This is type bt_hci_evt_command_complete_t.
	bt_hci_evt_command_status_t	This is type bt_hci_evt_command_status_t.
	bt_hci_evt_connection_complete_t	This is type bt_hci_evt_connection_complete_t.
	bt_hci_evt_connection_request_t	This is type bt_hci_evt_connection_request_t.
	bt_hci_evt_disconnection_complete_t	This is type bt_hci_evt_disconnection_complete_t.
	bt_hci_evt_encryption_change_t	This is type bt_hci_evt_encryption_change_t.
	bt_hci_evt_mode_change_t	This is type bt_hci_evt_mode_change_t.
	bt_hci_evt_role_change_t	This is type bt_hci_evt_role_change_t.
	bt_hci_hconn_p	This is type bt_hci_hconn_p.
	bt_hci_hconn_t	This is type bt_hci_hconn_t.
	bt_hci_inquiry_callback_fp	This is type bt_hci_inquiry_callback_fp.
	bt_hci_inquiry_response_t	This is type bt_hci_inquiry_response_t.
	bt_hci_le_advertising_report_t	This is type bt_hci_le_advertising_report_t.

	bt_hci_le_conn_state_t	This is type bt_hci_le_conn_state_t.
	bt_hci_le_connect_parameters_t	This is type bt_hci_le_connect_parameters_t.
	bt_hci_le_ctrl_state_t	This is type bt_hci_le_ctrl_state_t.
	bt_hci_le_evt_connection_updated_t	This is type bt_hci_le_evt_connection_updated_t.
	bt_hci_le_evt_read_remote_used_features_completed_t	This is type bt_hci_le_evt_read_remote_used_features_completed_t.
	bt_hci_le_evt_read_support_params_t	This is type bt_hci_le_evt_read_support_params_t.
	bt_hci_le_scan_callback_fp	This is type bt_hci_le_scan_callback_fp.
	bt_hci_link_key_t	This is type bt_hci_link_key_t.
	bt_hci_listener_t	This is type bt_hci_listener_t.
	bt_hci_read_inquiry_mode_callback_fp	This is type bt_hci_read_inquiry_mode_callback_fp.
	bt_hci_read_inquiry_scan_activity_callback_fp	This is type bt_hci_read_inquiry_scan_activity_callback_fp.
	bt_hci_read_inquiry_scan_type_callback_fp	This is type bt_hci_read_inquiry_scan_type_callback_fp.
	bt_hci_read_page_scan_activity_callback_fp	This is type bt_hci_read_page_scan_activity_callback_fp.
	bt_hci_read_page_scan_period_mode_callback_fp	This is type bt_hci_read_page_scan_period_mode_callback_fp.
	bt_hci_read_page_scan_type_callback_fp	This is type bt_hci_read_page_scan_type_callback_fp.
	bt_hci_read_page_timeout_callback_fp	This is type bt_hci_read_page_timeout_callback_fp.
	bt_hci_request_remote_name_callback_fp	This is type bt_hci_request_remote_name_callback_fp.
	bt_hci_sco_read_data_callback_fp	This is type bt_hci_sco_read_data_callback_fp.
	bt_hci_start_callback_fp	brief HCI initialization callback. ingroup hci details This typedef defines a function pointer type for the HCI initialization callback functions. Such a function must be passed to the bt_hci_start() function. param success Specifies whether HCI initialization succeeded or not.
	bt_hci_stop_callback_fp	brief HCI stop callback. ingroup hci details This typedef defines a function pointer type for the HCI stop callback functions. Such a function must be passed to the bt_hci_stop() function.
	bt_hci_transport_recv_packet_callback_fp	This is type bt_hci_transport_recv_packet_callback_fp.
	bt_hci_transport_send_packet_callback_fp	This is type bt_hci_transport_send_packet_callback_fp.
	bt_hcitr_tih4_power_callback_fp	This is type bt_hcitr_tih4_power_callback_fp.
	bt_hcitr_tih4_power_event_e	This is type bt_hcitr_tih4_power_event_e.
	bt_le_evt_handler	This is type bt_le_evt_handler.
	hci_transport_t	This is type hci_transport_t.
	pf_hci_sleep_callback	This is type pf_hci_sleep_callback.
	pf_hci_wakeup_callback	This is type pf_hci_wakeup_callback.
	pf_l2cap_receive_callback	This is type pf_l2cap_receive_callback.
	_bt_hci_command_s	This is type bt_hci_command_t.
	_bt_hci_conn_state_s	This is record _bt_hci_conn_state_s.
	_bt_hci_ctrl_listener_t	This is record _bt_hci_ctrl_listener_t.
	_bt_hci_ctrl_state_s	This is type bt_hci_ctrl_state_t.
	_bt_hci_data_buffer_s	This is type bt_hci_data_buffer_t.
	_bt_hci_data_s	This is type bt_hci_data_t.
	_bt_hci_event_e	This is type bt_hci_event_e.
	_bt_hci_event_s	This is type bt_hci_event_t.

	_bt_hci_evt_authentication_complete_t	This is type bt_hci_evt_authentication_complete_t.
	_bt_hci_evt_command_complete_t	This is type bt_hci_evt_command_complete_t.
	_bt_hci_evt_command_status_t	This is type bt_hci_evt_command_status_t.
	_bt_hci_evt_connection_complete_t	This is type bt_hci_evt_connection_complete_t.
	_bt_hci_evt_connection_request_t	This is type bt_hci_evt_connection_request_t.
	_bt_hci_evt_disconnection_complete_t	This is type bt_hci_evt_disconnection_complete_t.
	_bt_hci_evt_encryption_change_s	This is type bt_hci_evt_encryption_change_t.
	_bt_hci_evt_mode_change_t	This is type bt_hci_evt_mode_change_t.
	_bt_hci_evt_role_change_t	This is type bt_hci_evt_role_change_t.
	_bt_hci_inquiry_response_t	This is type bt_hci_inquiry_response_t.
	_bt_hci_le_advertising_report_t	This is record _bt_hci_le_advertising_report_t.
	_bt_hci_le_conn_state_t	This is type bt_hci_le_conn_state_t.
	_bt_hci_le_connect_parameters_t	This is type bt_hci_le_connect_parameters_t.
	_bt_hci_le_ctrl_state_t	This is type bt_hci_le_ctrl_state_t.
	_bt_hci_le_evt_connection_updated_t	This is type bt_hci_le_evt_connection_updated_t.
	_bt_hci_le_evt_read_remote_used_features_completed_t	This is type bt_hci_le_evt_read_remote_used_features_completed_t.
	_bt_hci_le_evt_read_support_params_t	This is type bt_hci_le_evt_read_support_params_t.
	_bt_hci_link_key_s	This is type bt_hci_link_key_t.
	_bt_hci_listener_t	This is record _bt_hci_listener_t.
	_hci_transport_t	This is type hci_transport_t.
	_hcitr_tih4_power_event_e	This is type bt_hcitr_tih4_power_event_e.
	HCI_DISCOVERABLE_MODE_GENERAL	This is macro HCI_DISCOVERABLE_MODE_GENERAL.
	HCI_DISCOVERABLE_MODE_LIMITED	This is macro HCI_DISCOVERABLE_MODE_LIMITED.
	HCI_EIR_TYPE_3D_Information_Data	This is macro HCI_EIR_TYPE_3D_Information_Data.
	HCI_EIR_TYPE_ADVERTISING_INTERVAL	This is macro HCI_EIR_TYPE_ADVERTISING_INTERVAL.
	HCI_EIR_TYPE_APPEARANCE	This is macro HCI_EIR_TYPE_APPEARANCE.
	HCI_EIR_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS	This is macro HCI_EIR_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS.
	HCI_EIR_TYPE_LE_ROLE	This is macro HCI_EIR_TYPE_LE_ROLE.
	HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_CONFIRMATION_VALUE	This is macro HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_CONFIRMATION_VALUE.
	HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_RANDOM_VALUE	This is macro HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_RANDOM_VALUE.
	HCI_EIR_TYPE_PUBLIC_TARGET_ADDRESS	This is macro HCI_EIR_TYPE_PUBLIC_TARGET_ADDRESS.
	HCI_EIR_TYPE_RANDOM_TARGET_ADDRESS	This is macro HCI_EIR_TYPE_RANDOM_TARGET_ADDRESS.
	HCI_EIR_TYPE_SERVICE_DATA	This is macro HCI_EIR_TYPE_SERVICE_DATA.
	HCI_EIR_TYPE_SERVICE_DATA_UUID128	This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID128.
	HCI_EIR_TYPE_SERVICE_DATA_UUID16	This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID16.
	HCI_EIR_TYPE_SERVICE_DATA_UUID32	This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID32.
	HCI_EIR_TYPE_SIMPLE_PAIRING_HASH_C_256	This is macro HCI_EIR_TYPE_SIMPLE_PAIRING_HASH_C_256.

	HCI_EIR_TYPE_SIMPLE_PAIRING_RANDOMIZER_R_256	This is macro HCI_EIR_TYPE_SIMPLE_PAIRING_RANDOMIZER_R_256.
	HCI_EIR_TYPE_SLAVE_CONN_INTERVAL_RANGE	This is macro HCI_EIR_TYPE_SLAVE_CONN_INTERVAL_RANGE.
	HCI_EIR_TYPE_SM_OOB_FLAGS	This is macro HCI_EIR_TYPE_SM_OOB_FLAGS.
	HCI_EIR_TYPE_SM_TK_VALUE	This is macro HCI_EIR_TYPE_SM_TK_VALUE.
	HCI_EIR_TYPE_SOLICITATION_UUID128_LIST	This is macro HCI_EIR_TYPE_SOLICITATION_UUID128_LIST.
	HCI_EIR_TYPE_SOLICITATION_UUID16_LIST	This is macro HCI_EIR_TYPE_SOLICITATION_UUID16_LIST.
	HCI_EIR_TYPE_SOLICITATION_UUID32_LIST	This is macro HCI_EIR_TYPE_SOLICITATION_UUID32_LIST.
	HCI_ERR_CONNECTION_TIMEOUT	This is macro HCI_ERR_CONNECTION_TIMEOUT.
	HCI_INIT_FLAG_IGNORE_TOTAL_NUM_ACL_DATA_PACKETS	The total number of ACL data packets read from the controller will be ignored. The stack will assume that he controller can accept only 1 ACL packet and sends HCI_EVT_NUM_OF_COMPLETED_PACKETS for each packet it has processed. This seems to be for controller working over SDIO (at least for Marvell's 88W8777).
	HCI_INIT_FLAG_SEND_HCI_RESET	HCI reset will be sent when bt_hci_init is called
	HCI_LE_ADV_CHANNEL_MAP_ENABLE_37	This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_37.
	HCI_LE_ADV_CHANNEL_MAP_ENABLE_38	This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_38.
	HCI_LE_ADV_CHANNEL_MAP_ENABLE_39	This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_39.
	HCI_LE_ADV_CHANNEL_MAP_ENABLE_ALL	This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_ALL.
	HCI_LE_ADV_CHANNEL_MAP_RESERVED	This is macro HCI_LE_ADV_CHANNEL_MAP_RESERVED.
	HCI_LE_ADV_TYPE_CONN_UNDIRECT	Connectable undirected advertising (ADV_IND) (default)
	HCI_LE_ADV_TYPE_DIRECT_HIGH	Connectable high duty cycle directed advertising (ADV_DIRECT_IND, high duty cycle)
	HCI_LE_ADV_TYPE_DIRECT_LOW	Connectable low duty cycle directed advertising (ADV_DIRECT_IND, low duty cycle)
	HCI_LE_ADV_TYPE_NONCONN	Non connectable undirected advertising (ADV_NONCONN_IND)
	HCI_LE_ADV_TYPE_SCAN	Scannable undirected advertising (ADV_SCAN_IND)
	HCI_LE_RANDOM_ADDRESS_TYPE_NON_RESOLVABLE	This is macro HCI_LE_RANDOM_ADDRESS_TYPE_NON_RESOLVABLE.
	HCI_LE_RANDOM_ADDRESS_TYPE_RESOLVABLE	This is macro HCI_LE_RANDOM_ADDRESS_TYPE_RESOLVABLE.
	HCI_LE_RANDOM_ADDRESS_TYPE_STATIC	This is macro HCI_LE_RANDOM_ADDRESS_TYPE_STATIC.
	HCI_LINK_POLICY_ENABLE_ALL	This is macro HCI_LINK_POLICY_ENABLE_ALL.
	bt_hci_le_cancel_connect	This is macro bt_hci_le_cancel_connect.
	bt_hci_send_pin_code	This is macro bt_hci_send_pin_code.
	HCI_ERR_PAIRING_NOT_ALLOWED	This is macro HCI_ERR_PAIRING_NOT_ALLOWED.
	HCI_ERR_PIN_OR_KEY_MISSING	This is macro HCI_ERR_PIN_OR_KEY_MISSING.

	HCI_PACKET_BOUNDARY_COMPLETE	This is macro HCI_PACKET_BOUNDARY_COMPLETE.
	HCI_SUCCESS	This is macro HCI_SUCCESS.

HCI Functions

	Name	Description
	bt_hci_add_param_bdaddr	brief Add BD address parameter to an HCI command ingroup hci
	bt_hci_add_param_byte	brief Add byte parameter to an HCI command ingroup hci
	bt_hci_add_param_cod	brief Add class of device parameter to an HCI command ingroup hci
	bt_hci_add_param_int	brief Add int parameter to an HCI command ingroup hci
	bt_hci_add_param_linkkey	brief Add link key parameter to an HCI command ingroup hci
	bt_hci_add_param_long	brief Add long parameter to an HCI command ingroup hci
	bt_hci_add_param_string	brief Add string parameter to an HCI command ingroup hci
	bt_hci_alloc_canned_command	brief Allocate and initialize an HCI command structure for a canned (pre-formatted) command. ingroup hci
	bt_hci_alloc_command	brief Allocate and initialize an HCI command structure. ingroup hci
	bt_hci_alloc_data_buffer	This is function bt_hci_alloc_data_buffer.
	bt_hci_allocate_write_eir_command	This is function bt_hci_allocate_write_eir_command.
	bt_hci_authenticate_ex	This is function bt_hci_authenticate_ex.
	bt_hci_cancel_find_devices	brief Stop inquiry
	bt_hci_cancel_request_remote_name	brief Cancel remote device name request
	bt_hci_cancel_send_acl_data	brief Cancel sending data over ACL connection
	bt_hci_connect	brief Connect to a remote device. ingroup hci details This function tries to establish an HCI connection with a remote device specified by the Bluetooth address c dest. Upon completion, the callback function specified by the c callback parameter is called. param dest Bluetooth address of the remote device. param packet_type param role_switch param acl_config param callback Pointer to a callback function that is called when the connect operation completes. param param Pointer to arbitrary data that is to be passed to the callback function. return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function... more
	bt_hci_connect_sco	This is function bt_hci_connect_sco.
	bt_hci_ctrl_register_data_listener	This is function bt_hci_ctrl_register_data_listener.
	bt_hci_ctrl_register_listener	This is function bt_hci_ctrl_register_listener.
	bt_hci_ctrl_unregister_listener	This is function bt_hci_ctrl_unregister_listener.
	bt_hci_disconnect	brief Abort connection ingroup hci
	bt_hci_evt_authentication_complete_handler	This is function bt_hci_evt_authentication_complete_handler.
	bt_hci_evt_change_conn_link_complete_handler	This is function bt_hci_evt_change_conn_link_complete_handler.
	bt_hci_evt_command_complete_handler	This is function bt_hci_evt_command_complete_handler.
	bt_hci_evt_command_status_handler	This is function bt_hci_evt_command_status_handler.
	bt_hci_evt_conn_packet_type_changed_handler	This is function bt_hci_evt_conn_packet_type_changed_handler.
	bt_hci_evt_connection_complete_handler	This is function bt_hci_evt_connection_complete_handler.
	bt_hci_evt_connection_request_handler	This is function bt_hci_evt_connection_request_handler.
	bt_hci_evt_data_buffer_overflow_handler	This is function bt_hci_evt_data_buffer_overflow_handler.
	bt_hci_evt_default_handler	This is function bt_hci_evt_default_handler.
	bt_hci_evt_disconnection_complete_handler	This is function bt_hci_evt_disconnection_complete_handler.
	bt_hci_evt_encryption_change_handler	This is function bt_hci_evt_encryption_change_handler.
	bt_hci_evt_extended_inquiry_result_handler	This is function bt_hci_evt_extended_inquiry_result_handler.
	bt_hci_evt_flow_specification_complete_handler	This is function bt_hci_evt_flow_specification_complete_handler.
	bt_hci_evt_flush_occured_handler	This is function bt_hci_evt_flush_occured_handler.
	bt_hci_evt_hardware_error_handler	This is function bt_hci_evt_hardware_error_handler.
	bt_hci_evt_inquiry_complete_handler	This is function bt_hci_evt_inquiry_complete_handler.
	bt_hci_evt_inquiry_result_handler	This is function bt_hci_evt_inquiry_result_handler.
	bt_hci_evt_inquiry_result_with_rssi_handler	This is function bt_hci_evt_inquiry_result_with_rssi_handler.
	bt_hci_evt_link_key_notification_handler	This is function bt_hci_evt_link_key_notification_handler.
	bt_hci_evt_link_key_request_handler	This is function bt_hci_evt_link_key_request_handler.

	bt_hci_evt_loopback_command_handler	This is function bt_hci_evt_loopback_command_handler.
	bt_hci_evt_master_link_key_complete_handler	This is function bt_hci_evt_master_link_key_complete_handler.
	bt_hci_evt_max_slots_change_handler	This is function bt_hci_evt_max_slots_change_handler.
	bt_hci_evt_mode_change_handler	This is function bt_hci_evt_mode_change_handler.
	bt_hci_evt_num_of_completed_packets_handler	This is function bt_hci_evt_num_of_completed_packets_handler.
	bt_hci_evt_page_scan_repet_mode_change_handler	This is function bt_hci_evt_page_scan_repet_mode_change_handler.
	bt_hci_evt_pin_code_request_handler	This is function bt_hci_evt_pin_code_request_handler.
	bt_hci_evt_qos_setup_complete_handler	This is function bt_hci_evt_qos_setup_complete_handler.
	bt_hci_evt_qos_violation_handler	This is function bt_hci_evt_qos_violation_handler.
	bt_hci_evt_read_clock_offset_complete_handler	This is function bt_hci_evt_read_clock_offset_complete_handler.
	bt_hci_evt_read_rmt_ext_features_comp_handler	This is function bt_hci_evt_read_rmt_ext_features_comp_handler.
	bt_hci_evt_read_rmt_sup_features_comp_handler	This is function bt_hci_evt_read_rmt_sup_features_comp_handler.
	bt_hci_evt_read_rmt_version_info_comp_handler	This is function bt_hci_evt_read_rmt_version_info_comp_handler.
	bt_hci_evt_remote_name_request_complete_handler	This is function bt_hci_evt_remote_name_request_complete_handler.
	bt_hci_evt_return_link_keys_handler	This is function bt_hci_evt_return_link_keys_handler.
	bt_hci_evt_role_change_handler	This is function bt_hci_evt_role_change_handler.
	bt_hci_evt_synch_connection_changed_handler	This is function bt_hci_evt_synch_connection_changed_handler.
	bt_hci_evt_synch_connection_complete_handler	This is function bt_hci_evt_synch_connection_complete_handler.
	bt_hci_exit_park_state	brief Exit park state
	bt_hci_exit_sniff_mode_ex	This is function bt_hci_exit_sniff_mode_ex.
	bt_hci_find_devices	brief Start inquiry
	bt_hci_find_devices_ex	This is function bt_hci_find_devices_ex.
	bt_hci_free_command	brief Free HCI command. ingroup hci
	bt_hci_free_data_buffer	This is function bt_hci_free_data_buffer.
	bt_hci_get_evt_param_bdaddr	brief Get bd address parameter from HCI event ingroup hci
	bt_hci_get_evt_param_byte	brief Get byte parameter from HCI event ingroup hci
	bt_hci_get_evt_param_devclass	brief Get class of device parameter from HCI event ingroup hci
	bt_hci_get_evt_param_int	brief Get int parameter from HCI event ingroup hci
	bt_hci_get_evt_param_linkkey	brief Get link key parameter from HCI event ingroup hci
	bt_hci_get_evt_param_long	brief Get long parameter from HCI event ingroup hci
	bt_hci_get_evt_param_uint	brief Get unsigned int parameter from HCI event ingroup hci
	bt_hci_get_evt_param_ulong	brief Get unsigned long parameter from HCI event ingroup hci
	bt_hci_get_last_cmd_status	This is function bt_hci_get_last_cmd_status.
	bt_hci_get_param_bdaddr	brief Get BD address parameter from HCI command ingroup hci
	bt_hci_get_param_byte	brief Get byte parameter from HCI command ingroup hci
	bt_hci_get_param_int	brief Get int parameter from HCI command ingroup hci
	bt_hci_get_param_linkkey	brief Get link key parameter from HCI command ingroup hci
	bt_hci_get_param_long	brief Get long parameter from HCI command ingroup hci
	bt_hci_get_recv_buffer	This is function bt_hci_get_recv_buffer.
	bt_hci_get_recv_buffer_len	This is function bt_hci_get_recv_buffer_len.
	bt_hci_get_send_buffer	This is function bt_hci_get_send_buffer.
	bt_hci_get_send_buffer_len	This is function bt_hci_get_send_buffer_len.
	bt_hci_init	brief Initialize the HCI layer. ingroup hci This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt_sys_init. This function essentially calls bt_hci_init_ex(HCI_LINK_POLICY_ENABLE_ALL) so all link policy setting are enabled.
	bt_hci_init_data_buffers	This is function bt_hci_init_data_buffers.
	bt_hci_init_data_queues	This is function bt_hci_init_data_queues.
	bt_hci_init_linkkey_buffers	This is function bt_hci_init_linkkey_buffers.
	bt_hci_le_add_device_to_white_list	This is function bt_hci_le_add_device_to_white_list.
	bt_hci_le_advertising_add	This is function bt_hci_le_advertising_add.
	bt_hci_le_advertising_add_local_name	This is function bt_hci_le_advertising_add_local_name.
	bt_hci_le_advertising_device_id_add	This is function bt_hci_le_advertising_device_id_add.

	bt_hci_le_advertising_flags_add	This is function bt_hci_le_advertising_flags_add.
	bt_hci_le_advertising_get_local_name	This is function bt_hci_le_advertising_get_local_name.
	bt_hci_le_advertising_tx_power_level_add	This is function bt_hci_le_advertising_tx_power_level_add.
	bt_hci_le_advertising_uuid128_add	This is function bt_hci_le_advertising_uuid128_add.
	bt_hci_le_advertising_uuid16_add	This is function bt_hci_le_advertising_uuid16_add.
	bt_hci_le_advertising_uuid32_add	This is function bt_hci_le_advertising_uuid32_add.
	bt_hci_le_advertising_vendor_add	This is function bt_hci_le_advertising_vendor_add.
	bt_hci_le_allocate_set_advertising_data_command	This is function bt_hci_le_allocate_set_advertising_data_command.
	bt_hci_le_cancel_find_devices	This is function bt_hci_le_cancel_find_devices.
	bt_hci_le_clear_white_list	This is function bt_hci_le_clear_white_list.
	bt_hci_le_connect_ex	This is function bt_hci_le_connect_ex.
	bt_hci_le_enable	This is function bt_hci_le_enable.
	bt_hci_le_encrypt	This is function bt_hci_le_encrypt.
	bt_hci_le_find_devices	This is function bt_hci_le_find_devices.
	bt_hci_le_get_connect_parameters	This is function bt_hci_le_get_connect_parameters.
	bt_hci_le_init	This is function bt_hci_le_init.
	bt_hci_le_ltk_negative_reply	This is function bt_hci_le_ltk_negative_reply.
	bt_hci_le_ltk_reply	This is function bt_hci_le_ltk_reply.
	bt_hci_le_rand	This is function bt_hci_le_rand.
	bt_hci_le_read_channel_map	This is function bt_hci_le_read_channel_map.
	bt_hci_le_read_remote_used_features	This is function bt_hci_le_read_remote_used_features.
	bt_hci_le_read_support	This is function bt_hci_le_read_support.
	bt_hci_le_read_white_list_size	This is function bt_hci_le_read_white_list_size.
	bt_hci_le_receiver_test	This is function bt_hci_le_receiver_test.
	bt_hci_le_remove_device_from_white_list	This is function bt_hci_le_remove_device_from_white_list.
	bt_hci_le_set_adevrtising_enable_ex	This is function bt_hci_le_set_adevrtising_enable_ex.
	bt_hci_le_set_advertising_parameters	This is function bt_hci_le_set_advertising_parameters.
	bt_hci_le_set_connect_parameters	This is function bt_hci_le_set_connect_parameters.
	bt_hci_le_set_host_channel_classification	This is function bt_hci_le_set_host_channel_classification.
	bt_hci_le_set_random_address	This is function bt_hci_le_set_random_address.
	bt_hci_le_set_scan_enable	This is function bt_hci_le_set_scan_enable.
	bt_hci_le_set_scan_parameters	This is function bt_hci_le_set_scan_parameters.
	bt_hci_le_start_encryption	This is function bt_hci_le_start_encryption.
	bt_hci_le_supported	This is function bt_hci_le_supported.
	bt_hci_le_test_end	This is function bt_hci_le_test_end.
	bt_hci_le_transmitter_test	This is function bt_hci_le_transmitter_test.
	bt_hci_le_update_connection	This is function bt_hci_le_update_connection.
	bt_hci_le_write_support	This is function bt_hci_le_write_support.
	bt_hci_listen	brief Listen for incoming connections. ingroup hci details param callback Pointer to a callback function that is called when a new incoming connection has been established. param param Pointer to arbitrary data that is to be passed to the callback function. return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.
	bt_hci_listen_sco	This is function bt_hci_listen_sco.
	bt_hci_param_eir_add	This is function bt_hci_param_eir_add.
	bt_hci_param_eir_device_id_add	This is function bt_hci_param_eir_device_id_add.
	bt_hci_param_eir_local_name_add	This is function bt_hci_param_eir_local_name_add.
	bt_hci_param_eir_uuid128_add	This is function bt_hci_param_eir_uuid128_add.
	bt_hci_param_eir_uuid16_add	This is function bt_hci_param_eir_uuid16_add.
	bt_hci_param_eir_uuid32_add	This is function bt_hci_param_eir_uuid32_add.
	bt_hci_param_eir_vendor_add	This is function bt_hci_param_eir_vendor_add.
	bt_hci_param_tx_power_level_add	This is function bt_hci_param_tx_power_level_add.
	bt_hci_park_state	brief Put local device to park state
	bt_hci_read_inquiry_mode	brief Get current inquiry mode

	bt_hci_read_inquiry_scan_activity	brief Get current inquiry scan activity configuration
	bt_hci_read_inquiry_scan_type	brief Get current inquiry scan type
	bt_hci_read_page_scan_activity	brief Get current page scan activity configuration
	bt_hci_read_page_scan_period_mode	brief Get current page scan period mode
	bt_hci_read_page_scan_type	brief Get current page scan type
	bt_hci_read_page_timeout	brief Get current page timeout
	bt_hci_register_listener	This is function bt_hci_register_listener.
	bt_hci_reject_pin_code	This is function bt_hci_reject_pin_code.
	bt_hci_request_remote_name	brief Request remote device's name
	bt_hci_role_change_ex	brief Change local device's role
	bt_hci_send_acl_data	brief Send data over ACL connection
	bt_hci_send_cmd	brief Send a command to local device
	bt_hci_send_linkkey	This is function bt_hci_send_linkkey.
	bt_hci_send_sco_data	brief Send data over SCO connection
	bt_hci_set_encryption_ex	brief Set connection encryption
	bt_hci_set_event_listener	listener is triggered by the following events: HCI_EVT_AUTHENTICATION_COMPLETE HCI_EVT_CONNECTION_COMPLETE HCI_EVT_DISCONNECTION_COMPLETE HCI_EVT_ROLE_CHANGE HCI_EVT_MODE_CHANGE
	bt_hci_set_incoming_connection_role	This is function bt_hci_set_incoming_connection_role.
	bt_hci_set_scan	This is function bt_hci_set_scan.
	bt_hci_sniff_mode_ex	This is function bt_hci_sniff_mode_ex.
	bt_hci_sniff_subrating_ex	This is function bt_hci_sniff_subrating_ex.
	bt_hci_start	brief Start HCI layer. ingroup hci details This function starts the HCI layer of the stack. Starting the HCI layer consists essentially of two steps: 1. Make the HCI transport receive packets from the controller. This results in a call to bt_oem_recv. 2. Reset and configure the controller. Upon completion of controller initialization the callback function passed in the c callback parameter is called. param callback Completion callback. Called when controller initialization is complete. param callback_param A pointer to arbitrary data to be passed to the c callback callback. param enable_scan This is a bitmask that defines which scans are enabled... more
	bt_hci_start_no_init	brief Start HCI layer without controller configuration. ingroup hci details This function is similar to bt_hci_start but unlike the former it does not perform the controller configuration. I.e., bt_hci_start_no_init simply calls the HCI transport and makes it receive packets from the controller. The main purpose of this function is make the HCI transport ready to exchange packets if controller needs some vendor specific configuration before it can be used with the stack. E.g., controllers based on CRS8811 chip need loading various values that configure its operating mode using CSR's proprietary protocol. So the application after configuring the HCI transport would... more
	bt_hci_stop	brief Stop HCI layer. ingroup hci details This function makes the HCI layer inoperable. After this call the application must perform the full reset of the HCI transport and stack. param callback Completion callback. Called when the HCI layer has been stopped. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	bt_hci_transport_recv_packet	This is function bt_hci_transport_recv_packet.
	bt_hci_transport_send_cmd	deprecated
	bt_hci_transport_send_data	This is function bt_hci_transport_send_data.
	bt_hci_transport_send_packet	This is function bt_hci_transport_send_packet.
	bt_hci_transport_set_transport	This is function bt_hci_transport_set_transport.
	bt_hci_unregister_listener	This is function bt_hci_unregister_listener.
	bt_hci_write_default_link_policy_settings	This is function bt_hci_write_default_link_policy_settings.
	bt_hci_write_eir	This is function bt_hci_write_eir.

	bt_hci_write_inquiry_mode	brief Configure inquiry mode
	bt_hci_write_inquiry_scan_activity	brief Configure inquiry scan activity
	bt_hci_write_inquiry_scan_type	brief Configure inquiry scan type
	bt_hci_write_local_name	brief Write local device name ingroup hci details The Write_Local_Name command provides the ability to modify the userfriendly name for the BR/EDR Controller.
	bt_hci_write_page_scan_activity	brief Configure page scan activity
	bt_hci_write_page_scan_period_mode	brief Configure page scan period mode
	bt_hci_write_page_scan_type	brief Configure page scan type
	bt_hci_write_page_timeout	brief Configure page timeout
	bt_hcitr_3wire_start	This is function bt_hcitr_3wire_start.
	bt_hcitr_bcsp_init_ex	This is function bt_hcitr_bcsp_init_ex.
	bt_hcitr_bcsp_reset_ex	This is function bt_hcitr_bcsp_reset_ex.
	bt_hcitr_bcsp_start	This is function bt_hcitr_bcsp_start.
	bt_hcitr_packet_init	This is function bt_hcitr_packet_init.
	bt_hcitr_packet_reset	This is function bt_hcitr_packet_reset.
	bt_hcitr_packet_start	This is function bt_hcitr_packet_start.
	bt_hcitr_tih4_init	This is function bt_hcitr_tih4_init.
	bt_hcitr_tih4_reset	This is function bt_hcitr_tih4_reset.
	bt_hcitr_tih4_start	This is function bt_hcitr_tih4_start.
	bt_hcitr_tih4_wake_up	This is function bt_hcitr_tih4_wake_up.
	bt_hcitr_uart_init	brief Initialize HCI UART (H4) transport protocol ingroup hcitr_uart details This function initializes internal structures of the transport. The application must call it as early as possible before bt_hcitr_uart_start and before the stack is initialized and started with bt_sys_init and bt_sys_start.
	bt_hcitr_uart_reset	brief Re-initialize HCI UART (H4) transport protocol ingroup hcitr_uart details This function re-initializes the transport. Currently it simply calls bt_hcitr_uart_init. After calling this function the application must perform the full initialization of the stack by calling bt_sys_init, bt_sys_start and initialization functions of all other profile modules the application is intending to use.
	bt_hcitr_uart_start	brief Start HCI UART (H4) transport protocol ingroup hcitr_uart details This function starts the transport, i.e., makes it able to receive and send packets.
	hci_allocate_conn_state	This is function hci_allocate_conn_state.
	hci_check_aux_info	From hci_aux_info.c
	hci_cq_find_by_bdaddr_and_opcode	This is function hci_cq_find_by_bdaddr_and_opcode.
	hci_cq_find_by_hconn	This is function hci_cq_find_by_hconn.
	hci_cq_find_by_hconn_and_opcode	This is function hci_cq_find_by_hconn_and_opcode.
	hci_cq_find_by_opcode	This is function hci_cq_find_by_opcode.
	hci_get_conn_state	This is function hci_get_conn_state.
	hci_sleep	brief Put local device to sleep
	hci_wakeup	brief Awaken local device
	is_bdaddr_command	This is function is_bdaddr_command.
	is_hconn_command	This is function is_hconn_command.
	_bt_hci_ctrl_notify_data_listeners	This is function _bt_hci_ctrl_notify_data_listeners.
	_bt_hci_ctrl_notify_listeners	This is function _bt_hci_ctrl_notify_listeners.
	_bt_hci_get_tick_count	This is function _bt_hci_get_tick_count.
	_bt_hci_init_signal	This is function _bt_hci_init_signal.
	_bt_hci_init_timer	From hci_timer.c
	_bt_hci_init_transport	This is function _bt_hci_init_transport.
	_bt_hci_le_command_complete_handler	This is function _bt_hci_le_command_complete_handler.
	_bt_hci_notify_listeners	This is function _bt_hci_notify_listeners.
	_bt_hci_set_signal	This is function _bt_hci_set_signal.
	_hci_allocate_buffers	Defined by OEM through library configuration
	_hci_allocate_cmd	This is function _hci_allocate_cmd.
	_hci_free_cmd	This is function _hci_free_cmd.

	_hci_init_cmd_buffers	This is function _hci_init_cmd_buffers.
	_hci_init_cmd_queues	This is function _hci_init_cmd_queues.
	_hci_receive_start	From hci_receive.c
	_hci_recv_sco_data_packet	From hci_sco.c
	_hci_send_commands_from_queue	From hci_send.c
	_hci_send_data	This is function _hci_send_data.
	_hci_send_data_fragment	This is function _hci_send_data_fragment.
	_hci_send_data_from_queue	This is function _hci_send_data_from_queue.
	_bt_le_evt_handler	This is function _bt_le_evt_handler.
	bt_hci_le_allocate_set_scan_response_data_command	This is function bt_hci_le_allocate_set_scan_response_data_command.
	bt_hci_le_cancel_connect_ex	This is function bt_hci_le_cancel_connect_ex.
	bt_hci_le_ibeacon_add	This is function bt_hci_le_ibeacon_add.
	bt_hci_set_vendor_specific_event_handler	This is function bt_hci_set_vendor_specific_event_handler.
	bt_hcitr_3wire_init_ex	This is function bt_hcitr_3wire_init_ex.
	bt_hcitr_3wire_reset_ex	This is function bt_hcitr_3wire_reset_ex.
	_bt_hci_set_init_flags	This is function _bt_hci_set_init_flags.
	hci_get_conn_state_by_bdaddr	This is function hci_get_conn_state_by_bdaddr.
	bt_hci_cancel_find_devices_ex	brief Stop inquiry
	bt_hci_get_inquiry_response_tx_power_level	This is function bt_hci_get_inquiry_response_tx_power_level.
	bt_hci_init_ex	brief Initialize the HCI layer. ingroup hci This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt_sys_init_ex. @param default_link_policy default link policy settings. This is a bitmask that defines the initial value of the link policy settings for all new BR/EDR connections. This value can be a combination of the following values: li HCI_LINK_POLICY_ENABLE_ROLE_SWITCH li HCI_LINK_POLICY_ENABLE_HOLD_MODE li HCI_LINK_POLICY_ENABLE_SNIFF_MODE li HCI_LINK_POLICY_ENABLE_PARK_STATE To enable all settings pass HCI_LINK_POLICY_ENABLE_ALL.
	bt_hci_reset	brief Reset controller. ingroup hci details This function resets the BT controller. param callback Completion callback. Called when the controller has been reset. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	bt_hci_send_pin_code_ex	This is function bt_hci_send_pin_code_ex.
	bt_hci_set_scan_ex	This is function bt_hci_set_scan_ex.
	bt_hci_write_link_policy_settings	This is function bt_hci_write_link_policy_settings.
	bt_hci_write_local_name_ex	This is function bt_hci_write_local_name_ex.

L2CAP Data Types and Constants

	Name	Description
	bt_l2cap_connect	This is macro bt_l2cap_connect.
	bt_l2cap_listen	This is macro bt_l2cap_listen.
	bt_l2cap_test_enable_local_config	This is macro bt_l2cap_test_enable_local_config.
	bt_l2cap_test_enable_remote_config	This is macro bt_l2cap_test_enable_remote_config.
	CHANNEL_SIGNAL_CMD_DISCONNECT_FIXED	This is macro CHANNEL_SIGNAL_CMD_DISCONNECT_FIXED.
	CID_ATT	This is macro CID_ATT.
	CID_LE_SIG	This is macro CID_LE_SIG.
	CID_MAX	This is macro CID_MAX.
	CID_MAX_FIXED	This is macro CID_MAX_FIXED.
	CID_NULL	This is macro CID_NULL.
	CID_RECV	This is macro CID_RECV.
	CID_SIG	This is macro CID_SIG.
	CID_SM	This is macro CID_SM.
	CMODE_BASIC	This is macro CMODE_BASIC.
	CMODE_ERETR	This is macro CMODE_ERETR.

	CMODE_FLOW	This is macro CMODE_FLOW.
	CMODE_RETR	This is macro CMODE_RETR.
	CMODE_STRM	This is macro CMODE_STRM.
	CSTATE_CLOSED	This is macro CSTATE_CLOSED.
	CSTATE_FREE	This is macro CSTATE_FREE.
	CSTATE_OPEN	This is macro CSTATE_OPEN.
	CSTATE_WAIT_CONFIG	This is macro CSTATE_WAIT_CONFIG.
	CSTATE_WAIT_CONFIG_REQ	This is macro CSTATE_WAIT_CONFIG_REQ.
	CSTATE_WAIT_CONFIG_RSP	This is macro CSTATE_WAIT_CONFIG_RSP.
	CSTATE_WAIT_CONNECT	This is macro CSTATE_WAIT_CONNECT.
	CSTATE_WAIT_CONNECT_RSP	This is macro CSTATE_WAIT_CONNECT_RSP.
	CSTATE_WAIT_DISCONNECT	This is macro CSTATE_WAIT_DISCONNECT.
	CTYPE_CL	connectionless bt_l2cap_channel
	CTYPE_CO	connection-oriented bt_l2cap_channel
	GET_F_BIT	This is macro GET_F_BIT.
	GET_FRAME_TYPE	This is macro GET_FRAME_TYPE.
	GET_P_BIT	This is macro GET_P_BIT.
	GET_REQ_SEQ	This is macro GET_REQ_SEQ.
	GET_S_FUNCTION	This is macro GET_S_FUNCTION.
	GET_SAR	This is macro GET_SAR.
	GET_TX_SEQ	This is macro GET_TX_SEQ.
	L2CAP_CHANNEL_FLAG_INCOMING	This is macro L2CAP_CHANNEL_FLAG_INCOMING.
	L2CAP_CHANNEL_FLAG_SENDING	This is macro L2CAP_CHANNEL_FLAG_SENDING.
	L2CAP_CMD_CONFIG_REQUEST	This is macro L2CAP_CMD_CONFIG_REQUEST.
	L2CAP_CMD_CONFIG_RESPONSE	This is macro L2CAP_CMD_CONFIG_RESPONSE.
	L2CAP_CMD_CONN_PARAM_UPDATE_REQUEST	This is macro L2CAP_CMD_CONN_PARAM_UPDATE_REQUEST.
	L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE	This is macro L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE.
	L2CAP_CMD_CONN_REQUEST	This is macro L2CAP_CMD_CONN_REQUEST.
	L2CAP_CMD_CONN_RESPONSE	This is macro L2CAP_CMD_CONN_RESPONSE.
	L2CAP_CMD_DATA_LEN_CMD_REJECT	This is macro L2CAP_CMD_DATA_LEN_CMD_REJECT.
	L2CAP_CMD_DATA_LEN_CONFIG_REQUEST	This is macro L2CAP_CMD_DATA_LEN_CONFIG_REQUEST.
	L2CAP_CMD_DATA_LEN_CONFIG_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_CONFIG_RESPONSE.
	L2CAP_CMD_DATA_LEN_CONN_REQUEST	This is macro L2CAP_CMD_DATA_LEN_CONN_REQUEST.
	L2CAP_CMD_DATA_LEN_CONN_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_CONN_RESPONSE.
	L2CAP_CMD_DATA_LEN_DCONN_REQUEST	This is macro L2CAP_CMD_DATA_LEN_DCONN_REQUEST.
	L2CAP_CMD_DATA_LEN_DCONN_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_DCONN_RESPONSE.
	L2CAP_CMD_DATA_LEN_ECHO_REQUEST	This is macro L2CAP_CMD_DATA_LEN_ECHO_REQUEST.
	L2CAP_CMD_DATA_LEN_ECHO_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_ECHO_RESPONSE.
	L2CAP_CMD_DATA_LEN_INFO_REQUEST	This is macro L2CAP_CMD_DATA_LEN_INFO_REQUEST.
	L2CAP_CMD_DATA_LEN_INFO_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_INFO_RESPONSE.
	L2CAP_CMD_DCONN_REQUEST	This is macro L2CAP_CMD_DCONN_REQUEST.
	L2CAP_CMD_DCONN_RESPONSE	This is macro L2CAP_CMD_DCONN_RESPONSE.
	L2CAP_CMD_ECHO_REQUEST	This is macro L2CAP_CMD_ECHO_REQUEST.
	L2CAP_CMD_ECHO_RESPONSE	This is macro L2CAP_CMD_ECHO_RESPONSE.
	L2CAP_CMD_HEADER_LEN	This is macro L2CAP_CMD_HEADER_LEN.
	L2CAP_CMD_INFO_REQUEST	This is macro L2CAP_CMD_INFO_REQUEST.
	L2CAP_CMD_INFO_RESPONSE	This is macro L2CAP_CMD_INFO_RESPONSE.
	L2CAP_CMD_LAST	This is macro L2CAP_CMD_LAST.

	L2CAP_CMD_REJECT	This is macro L2CAP_CMD_REJECT.
	L2CAP_CMD_RESERVED	This is macro L2CAP_CMD_RESERVED.
	L2CAP_CMD_STATUS_BEING_SENT	This is macro L2CAP_CMD_STATUS_BEING_SENT.
	L2CAP_CMD_STATUS_PENDING	This is macro L2CAP_CMD_STATUS_PENDING.
	L2CAP_CMD_STATUS_WAITING_RESPONSE	This is macro L2CAP_CMD_STATUS_WAITING_RESPONSE.
	L2CAP_CONFIG_RESULT_REJECTED	This is macro L2CAP_CONFIG_RESULT_REJECTED.
	L2CAP_CONFIG_RESULT_SUCCESS	This is macro L2CAP_CONFIG_RESULT_SUCCESS.
	L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER	This is macro L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER.
	L2CAP_CONFIG_RESULT_UNKNOWN_OPTION	This is macro L2CAP_CONFIG_RESULT_UNKNOWN_OPTION.
	L2CAP_CONN_REQ_RESULT_INVALID_PSM	This is macro L2CAP_CONN_REQ_RESULT_INVALID_PSM.
	L2CAP_CONN_REQ_RESULT_NO_RESOURCES	This is macro L2CAP_CONN_REQ_RESULT_NO_RESOURCES.
	L2CAP_CONN_REQ_RESULT_PENDING	This is macro L2CAP_CONN_REQ_RESULT_PENDING.
	L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK	This is macro L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK.
	L2CAP_CONN_REQ_RESULT_SUCCESS	This is macro L2CAP_CONN_REQ_RESULT_SUCCESS.
	L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING	This is macro L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING.
	L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING	This is macro L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING.
	L2CAP_CONN_REQ_STATUS_NO_INFO	This is macro L2CAP_CONN_REQ_STATUS_NO_INFO.
	L2CAP_DEFAULT_ERTX	seconds
	L2CAP_DEFAULT_RTX	seconds
	L2CAP_ECHO_MAX_DATA_LEN	echo request and response
	L2CAP_ERETR_RECV_STATE_RECV	This is macro L2CAP_ERETR_RECV_STATE_RECV.
	L2CAP_ERETR_RECV_STATE_REJ_SENT	This is macro L2CAP_ERETR_RECV_STATE_REJ_SENT.
	L2CAP_ERETR_RECV_STATE_SREJ_SENT	This is macro L2CAP_ERETR_RECV_STATE_SREJ_SENT.
	L2CAP_ERETR_XMIT_STATE_WAIT_ACK	This is macro L2CAP_ERETR_XMIT_STATE_WAIT_ACK.
	L2CAP_ERETR_XMIT_STATE_WAIT_F	This is macro L2CAP_ERETR_XMIT_STATE_WAIT_F.
	L2CAP_ERETR_XMIT_STATE_XMIT	This is macro L2CAP_ERETR_XMIT_STATE_XMIT.
	L2CAP_EXT_BI_QOS	This is macro L2CAP_EXT_BI_QOS.
	L2CAP_EXT_ENHANCED_RETRANSMISSION	This is macro L2CAP_EXT_ENHANCED_RETRANSMISSION.
	L2CAP_EXT_FCS_OPTION	This is macro L2CAP_EXT_FCS_OPTION.
	L2CAP_EXT_FLOW_CONTROL	This is macro L2CAP_EXT_FLOW_CONTROL.
	L2CAP_EXT_RETRANSMISSION	This is macro L2CAP_EXT_RETRANSMISSION.
	L2CAP_EXT_STREAMING	This is macro L2CAP_EXT_STREAMING.
	L2CAP_FRAME_TYPE_I	This is macro L2CAP_FRAME_TYPE_I.
	L2CAP_FRAME_TYPE_S	This is macro L2CAP_FRAME_TYPE_S.
	L2CAP_HCI_CONNECT_PACKET_TYPE	This is macro L2CAP_HCI_CONNECT_PACKET_TYPE.
	L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE	This is macro L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE.
	L2CAP_HCI_CONNECT_ROLE_SWITCH	This is macro L2CAP_HCI_CONNECT_ROLE_SWITCH.
	L2CAP_HEADER_LEN	This is macro L2CAP_HEADER_LEN.
	L2CAP_IDLE_HCI_CONNECTION_TIMEOUT	This is macro L2CAP_IDLE_HCI_CONNECTION_TIMEOUT.
	L2CAP_INFO_NOT_SUPPORTED	REDFLAG: should it be non zero?
	L2CAP_INFO_RESULT_SUCCESS	This is macro L2CAP_INFO_RESULT_SUCCESS.
	L2CAP_INFO_TYPE_CONNECTIONLESS_MTU	information request
	L2CAP_INFO_TYPE_EXTENDED_SUPPORT	This is macro L2CAP_INFO_TYPE_EXTENDED_SUPPORT.
	L2CAP_LINK_TYPE_BD_EDR	This is macro L2CAP_LINK_TYPE_BD_EDR.

	L2CAP_LINK_TYPE_LE	This is macro L2CAP_LINK_TYPE_LE.
	L2CAP_MAX_CMD_QUEUE_LEN	This is macro L2CAP_MAX_CMD_QUEUE_LEN.
	L2CAP_MAX_FRAME_BUFFERS	This is macro L2CAP_MAX_FRAME_BUFFERS.
	L2CAP_MAX_MANAGERS	This is macro L2CAP_MAX_MANAGERS.
	L2CAP_MAX_MTU	This is macro L2CAP_MAX_MTU.
	L2CAP_MAX_OPTIONS	configuration request/response
	L2CAP_MAX_RTX	This is macro L2CAP_MAX_RTX.
	L2CAP_MGR_STATE_FREE	This is macro L2CAP_MGR_STATE_FREE.
	L2CAP_MGR_STATE_LISTENING_L2CAP	This is macro L2CAP_MGR_STATE_LISTENING_L2CAP.
	L2CAP_MGR_STATE_USED	This is macro L2CAP_MGR_STATE_USED.
	L2CAP_MIN_MTU	This is macro L2CAP_MIN_MTU.
	L2CAP_MONITOR_TIMEOUT	This is macro L2CAP_MONITOR_TIMEOUT.
	L2CAP_OPTION_LEN_FLASH_QOS	This is macro L2CAP_OPTION_LEN_FLASH_QOS.
	L2CAP_OPTION_LEN_FLASH_RFC	This is macro L2CAP_OPTION_LEN_FLASH_RFC.
	L2CAP_OPTION_LEN_FLASH_TIMEOUT	This is macro L2CAP_OPTION_LEN_FLASH_TIMEOUT.
	L2CAP_OPTION_LEN_MAX_MTU	This is macro L2CAP_OPTION_LEN_MAX_MTU.
	L2CAP_OPTION_LEN_UNKNOWN	This is macro L2CAP_OPTION_LEN_UNKNOWN.
	L2CAP_OPTION_LEN_UNKNOWN_DATA	21
	L2CAP_OPTION_TYPE_FLASH_QOS	This is macro L2CAP_OPTION_TYPE_FLASH_QOS.
	L2CAP_OPTION_TYPE_FLASH_QOS_FLAG	This is macro L2CAP_OPTION_TYPE_FLASH_QOS_FLAG.
	L2CAP_OPTION_TYPE_FLASH_RFC	This is macro L2CAP_OPTION_TYPE_FLASH_RFC.
	L2CAP_OPTION_TYPE_FLASH_RFC_FLAG	This is macro L2CAP_OPTION_TYPE_FLASH_RFC_FLAG.
	L2CAP_OPTION_TYPE_FLASH_TIMEOUT	This is macro L2CAP_OPTION_TYPE_FLASH_TIMEOUT.
	L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG	This is macro L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG.
	L2CAP_OPTION_TYPE_MAX_MTU	This is macro L2CAP_OPTION_TYPE_MAX_MTU.
	L2CAP_OPTION_TYPE_MAX_MTU_FLAG	This is macro L2CAP_OPTION_TYPE_MAX_MTU_FLAG.
	L2CAP_OPTION_TYPE_UNKNOWN_FLAG	This is macro L2CAP_OPTION_TYPE_UNKNOWN_FLAG.
	L2CAP_PACKET_DATA_TYPE_RAW	This is macro L2CAP_PACKET_DATA_TYPE_RAW.
	L2CAP_PACKET_DATA_TYPE_SMART	This is macro L2CAP_PACKET_DATA_TYPE_SMART.
	L2CAP_REJECT_REASON_INVALID_CHANNEL	This is macro L2CAP_REJECT_REASON_INVALID_CHANNEL.
	L2CAP_REJECT_REASON_MTU_EXCEEDED	This is macro L2CAP_REJECT_REASON_MTU_EXCEEDED.
	L2CAP_REJECT_REASON_NOT_UNDERSTOOD	This is macro L2CAP_REJECT_REASON_NOT_UNDERSTOOD.
	L2CAP_RETR_TIMEOUT	This is macro L2CAP_RETR_TIMEOUT.
	L2CAP_RFC_BASIC	This is macro L2CAP_RFC_BASIC.
	L2CAP_RFC_ENHANCED_RETRANSMISSION	This is macro L2CAP_RFC_ENHANCED_RETRANSMISSION.
	L2CAP_RFC_ERETR_TIMEOUT	3 secs
	L2CAP_RFC_ERETR_TX_WINDOW	This is macro L2CAP_RFC_ERETR_TX_WINDOW.
	L2CAP_RFC_FLOW_CONTROL	This is macro L2CAP_RFC_FLOW_CONTROL.
	L2CAP_RFC_MONITOR_TIMEOUT	12 secs
	L2CAP_RFC_RETRANSMISSION	This is macro L2CAP_RFC_RETRANSMISSION.
	L2CAP_RFC_STREAMING	This is macro L2CAP_RFC_STREAMING.
	L2CAP_SAR_SDU_CONTINUE	This is macro L2CAP_SAR_SDU_CONTINUE.
	L2CAP_SAR_SDU_START	This is macro L2CAP_SAR_SDU_START.
	L2CAP_SAR_SDU_STOP	This is macro L2CAP_SAR_SDU_STOP.
	L2CAP_SAR_UNSEGMENTED	This is macro L2CAP_SAR_UNSEGMENTED.
	L2CAP_SERVICE_TYPE_BEST_EFFORT	This is macro L2CAP_SERVICE_TYPE_BEST_EFFORT.
	L2CAP_SERVICE_TYPE_GUARANTEED	This is macro L2CAP_SERVICE_TYPE_GUARANTEED.
	L2CAP_SERVICE_TYPE_NO_TRAFFIC	This is macro L2CAP_SERVICE_TYPE_NO_TRAFFIC.
	L2CAP_SFUNCTION_REJ	This is macro L2CAP_SFUNCTION_REJ.
	L2CAP_SFUNCTION_RNR	This is macro L2CAP_SFUNCTION_RNR.
	L2CAP_SFUNCTION_RR	This is macro L2CAP_SFUNCTION_RR.

	L2CAP_SFUNCTION_SREJ	This is macro L2CAP_SFUNCTION_SREJ.
	PSM_ATT	This is macro PSM_ATT.
	PSM_AVCTP	This is macro PSM_AVCTP.
	PSM_AVCTP_Browsing	This is macro PSM_AVCTP_Browsing.
	PSM_AVDTP	This is macro PSM_AVDTP.
	PSM_BNEP	This is macro PSM_BNEP.
	PSM_HID_Control	This is macro PSM_HID_Control.
	PSM_HID_Interrupt	This is macro PSM_HID_Interrupt.
	PSM_RFCOMM	This is macro PSM_RFCOMM.
	PSM_SDP	This is macro PSM_SDP.
	SET_F_BIT	This is macro SET_F_BIT.
	SET_FRAME_TYPE	This is macro SET_FRAME_TYPE.
	SET_P_BIT	This is macro SET_P_BIT.
	SET_REQ_SEQ	This is macro SET_REQ_SEQ.
	SET_S_FUNCTION	This is macro SET_S_FUNCTION.
	SET_SAR	This is macro SET_SAR.
	SET_TX_SEQ	This is macro SET_TX_SEQ.
	_bt_l2cap_start_monitor_timer	This is macro _bt_l2cap_start_monitor_timer.
	_bt_l2cap_start_monitor_timer_if_not_running	This is macro _bt_l2cap_start_monitor_timer_if_not_running.
	_bt_l2cap_start_retr_timer	This is macro _bt_l2cap_start_retr_timer.
	_bt_l2cap_start_retr_timer_if_not_running	This is macro _bt_l2cap_start_retr_timer_if_not_running.
	_bt_l2cap_stop_monitor_timer	This is macro _bt_l2cap_stop_monitor_timer.
	_bt_l2cap_stop_retr_timer	This is macro _bt_l2cap_stop_retr_timer.
	_bt_l2cap_cfg_option_t	This is type bt_l2cap_cfg_option_t.
	_bt_l2cap_channel_t	This is record _bt_l2cap_channel_t.
	_bt_l2cap_cmd_config_req_t	This is type bt_l2cap_cmd_config_req_t.
	_bt_l2cap_cmd_config_res_t	This is type bt_l2cap_cmd_config_res_t.
	_bt_l2cap_cmd_conn_param_update_req_t	connection parameter update
	_bt_l2cap_cmd_conn_param_update_res_t	This is type bt_l2cap_cmd_conn_param_update_res_t.
	_bt_l2cap_cmd_connection_req_t	This is type bt_l2cap_cmd_connection_req_t.
	_bt_l2cap_cmd_connection_res_t	This is type bt_l2cap_cmd_connection_res_t.
	_bt_l2cap_cmd_disconnection_req_t	This is type bt_l2cap_cmd_disconnection_req_t.
	_bt_l2cap_cmd_echo_req_t	This is type bt_l2cap_cmd_echo_req_t.
	_bt_l2cap_cmd_echo_res_t	This is type bt_l2cap_cmd_echo_res_t.
	_bt_l2cap_cmd_header_t	This is type bt_l2cap_cmd_header_t.
	_bt_l2cap_cmd_info_req_t	This is type bt_l2cap_cmd_info_req_t.
	_bt_l2cap_cmd_info_res_t	This is type bt_l2cap_cmd_info_res_t.
	_bt_l2cap_cmd_reject_param_t	This is type bt_l2cap_cmd_reject_param_t.
	_bt_l2cap_cmd_reject_t	This is type bt_l2cap_cmd_reject_t.
	_bt_l2cap_command_t	This is type bt_l2cap_command_t.
	_bt_l2cap_eretr_xmit_event_e	This is type bt_l2cap_eretr_xmit_event_e.
	_bt_l2cap_fixed_channel_s	This is type bt_l2cap_fixed_channel_t.
	_bt_l2cap_frame_desc_s	typedef union _bt_l2cap_frame_control_s { struct { bt_byte frame_type:1; bt_byte txSeq:6; bt_byte f:1; bt_byte reqSeq:6; bt_byte sar:2; } iframe; struct { bt_byte frame_type:1; bt_byte reserved:1; bt_byte s:2; bt_byte p:1; bt_byte reserved2:2; bt_byte f:1; bt_byte reqSeq:6; bt_byte reserved3:2; } sframe; } bt_l2cap_frame_control_t;
	_bt_l2cap_mgr_s	This is type bt_l2cap_mgr_t.
	_bt_l2cap_option_flash_timeout_t	This is type bt_l2cap_option_flash_timeout_t.
	_bt_l2cap_option_max_mtu_t	This is type bt_l2cap_option_max_mtu_t.
	_bt_l2cap_option_qos_t	This is type bt_l2cap_option_qos_t.
	_bt_l2cap_option_rfc_t	This is type bt_l2cap_option_rfc_t.
	_bt_l2cap_option_unknown_t	This is type bt_l2cap_option_unknown_t.
	_bt_l2cap_packet_t	This is type bt_l2cap_packet_t.
	_bt_l2cap_psm_s	This is type bt_l2cap_psm_t.

	_bt_l2cap_xmit_event_param_t	This is type bt_l2cap_xmit_event_param_t.
	bt_l2cap_cfg_option_p	This is type bt_l2cap_cfg_option_p.
	bt_l2cap_cfg_option_t	This is type bt_l2cap_cfg_option_t.
	bt_l2cap_channel_t	This is type bt_l2cap_channel_t.
	bt_l2cap_cmd_assembler_fp	This is type bt_l2cap_cmd_assembler_fp.
	bt_l2cap_cmd_config_req_p	This is type bt_l2cap_cmd_config_req_p.
	bt_l2cap_cmd_config_req_t	This is type bt_l2cap_cmd_config_req_t.
	bt_l2cap_cmd_config_res_p	This is type bt_l2cap_cmd_config_res_p.
	bt_l2cap_cmd_config_res_t	This is type bt_l2cap_cmd_config_res_t.
	bt_l2cap_cmd_conn_param_update_req_t	connection parameter update
	bt_l2cap_cmd_conn_param_update_res_t	This is type bt_l2cap_cmd_conn_param_update_res_t.
	bt_l2cap_cmd_connection_req_p	This is type bt_l2cap_cmd_connection_req_p.
	bt_l2cap_cmd_connection_req_t	This is type bt_l2cap_cmd_connection_req_t.
	bt_l2cap_cmd_connection_res_p	This is type bt_l2cap_cmd_connection_res_p.
	bt_l2cap_cmd_connection_res_t	This is type bt_l2cap_cmd_connection_res_t.
	bt_l2cap_cmd_disconnection_req_p	This is type bt_l2cap_cmd_disconnection_req_p.
	bt_l2cap_cmd_disconnection_req_t	This is type bt_l2cap_cmd_disconnection_req_t.
	bt_l2cap_cmd_echo_req_p	This is type bt_l2cap_cmd_echo_req_p.
	bt_l2cap_cmd_echo_req_t	This is type bt_l2cap_cmd_echo_req_t.
	bt_l2cap_cmd_echo_res_p	This is type bt_l2cap_cmd_echo_res_p.
	bt_l2cap_cmd_echo_res_t	This is type bt_l2cap_cmd_echo_res_t.
	bt_l2cap_cmd_header_p	This is type bt_l2cap_cmd_header_p.
	bt_l2cap_cmd_header_t	This is type bt_l2cap_cmd_header_t.
	bt_l2cap_cmd_info_req_p	This is type bt_l2cap_cmd_info_req_p.
	bt_l2cap_cmd_info_req_t	This is type bt_l2cap_cmd_info_req_t.
	bt_l2cap_cmd_info_res_p	This is type bt_l2cap_cmd_info_res_p.
	bt_l2cap_cmd_info_res_t	This is type bt_l2cap_cmd_info_res_t.
	bt_l2cap_cmd_parser_fp	This is type bt_l2cap_cmd_parser_fp.
	bt_l2cap_cmd_reject_p	This is type bt_l2cap_cmd_reject_p.
	bt_l2cap_cmd_reject_param_t	This is type bt_l2cap_cmd_reject_param_t.
	bt_l2cap_cmd_reject_t	This is type bt_l2cap_cmd_reject_t.
	bt_l2cap_command_t	This is type bt_l2cap_command_t.
	bt_l2cap_connect_callback_fp	This is type bt_l2cap_connect_callback_fp.
	bt_l2cap_eretr_xmit_event_e	This is type bt_l2cap_eretr_xmit_event_e.
	bt_l2cap_fixed_channel_t	This is type bt_l2cap_fixed_channel_t.
	bt_l2cap_frame_desc_t	typedef union _bt_l2cap_frame_control_s { struct { bt_byte frame_type:1; bt_byte txSeq:6; bt_byte f:1; bt_byte reqSeq:6; bt_byte sar:2; } iframe; struct { bt_byte frame_type:1; bt_byte reserved:1; bt_byte s:2; bt_byte p:1; bt_byte reserved2:2; bt_byte f:1; bt_byte reqSeq:6; bt_byte reserved3:2; } sframe; } bt_l2cap_frame_control_t;
	bt_l2cap_listen_callback_fp	This is type bt_l2cap_listen_callback_fp.
	bt_l2cap_mgr_p	This is type bt_l2cap_mgr_p.
	bt_l2cap_mgr_t	This is type bt_l2cap_mgr_t.
	bt_l2cap_option_flash_timeout_t	This is type bt_l2cap_option_flash_timeout_t.
	bt_l2cap_option_max_mtu_t	This is type bt_l2cap_option_max_mtu_t.
	bt_l2cap_option_qos_t	This is type bt_l2cap_option_qos_t.
	bt_l2cap_option_rfc_t	This is type bt_l2cap_option_rfc_t.
	bt_l2cap_option_unknown_t	This is type bt_l2cap_option_unknown_t.
	bt_l2cap_packet_t	This is type bt_l2cap_packet_t.
	bt_l2cap_psm_t	This is type bt_l2cap_psm_t.
	bt_l2cap_read_data_callback_fp	This is type bt_l2cap_read_data_callback_fp.
	bt_l2cap_request_handler_fp	This is type bt_l2cap_request_handler_fp.
	bt_l2cap_response_handler_fp	This is type bt_l2cap_response_handler_fp.
	bt_l2cap_send_data_callback_fp	This is type bt_l2cap_send_data_callback_fp.
	bt_l2cap_state_changed_callback_fp	This is type bt_l2cap_state_changed_callback_fp.

	bt_l2cap_xmit_event_param_t	This is type bt_l2cap_xmit_event_param_t.
	cmd_disconnection_res	This is type cmd_disconnection_res.
	pcmd_disconnection_res	This is type pcmd_disconnection_res.
	pf_l2cap_cmd_callback	This is type pf_l2cap_cmd_callback.
	L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT	This is macro L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT.
	L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR	This is macro L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR.
	L2CAP_EXT_EXT_WINDOW_SIZE	This is macro L2CAP_EXT_EXT_WINDOW_SIZE.
	L2CAP_EXT_FEATURES_ENABLED	This is macro L2CAP_EXT_FEATURES_ENABLED.
	L2CAP_EXT_FIXED_CHANNELS	This is macro L2CAP_EXT_FIXED_CHANNELS.
	L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT	This is macro L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT.
	L2CAP_INFO_TYPE_FIXED_CHANNELS	This is macro L2CAP_INFO_TYPE_FIXED_CHANNELS.
	bt_l2cap_channel_ext_t	This is type bt_l2cap_channel_ext_t.
	_bt_l2cap_channel_ext_t	This is record _bt_l2cap_channel_ext_t.
	_bt_l2cap_connect_params_s	This is record _bt_l2cap_connect_params_s.
	L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID	This is macro L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID.
	L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED	This is macro L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED.
	bt_l2cap_connect_params_t	This is type bt_l2cap_connect_params_t.

L2CAP Functions

	Name	Description
	bt_l2cap_alloc_cmd_buffer	This is function bt_l2cap_alloc_cmd_buffer.
	bt_l2cap_alloc_cmd_config_req	This is function bt_l2cap_alloc_cmd_config_req.
	bt_l2cap_alloc_cmd_config_res	This is function bt_l2cap_alloc_cmd_config_res.
	bt_l2cap_alloc_cmd_conn_param_update_req	This is function bt_l2cap_alloc_cmd_conn_param_update_req.
	bt_l2cap_alloc_cmd_conn_param_update_res	This is function bt_l2cap_alloc_cmd_conn_param_update_res.
	bt_l2cap_alloc_cmd_connection_req	This is function bt_l2cap_alloc_cmd_connection_req.
	bt_l2cap_alloc_cmd_connection_res	This is function bt_l2cap_alloc_cmd_connection_res.
	bt_l2cap_alloc_cmd_disconnection_req	This is function bt_l2cap_alloc_cmd_disconnection_req.
	bt_l2cap_alloc_cmd_disconnection_res	This is function bt_l2cap_alloc_cmd_disconnection_res.
	bt_l2cap_alloc_cmd_echo_req	This is function bt_l2cap_alloc_cmd_echo_req.
	bt_l2cap_alloc_cmd_echo_res	This is function bt_l2cap_alloc_cmd_echo_res.
	bt_l2cap_alloc_cmd_info_req	This is function bt_l2cap_alloc_cmd_info_req.
	bt_l2cap_alloc_cmd_info_res	This is function bt_l2cap_alloc_cmd_info_res.
	bt_l2cap_alloc_cmd_reject	This is function bt_l2cap_alloc_cmd_reject.
	bt_l2cap_alloc_frame_buffer	This is function bt_l2cap_alloc_frame_buffer.
	bt_l2cap_allocate_channel	This is function bt_l2cap_allocate_channel.
	bt_l2cap_allocate_fixed_channel	This is function bt_l2cap_allocate_fixed_channel.
	bt_l2cap_allocate_mgr	brief Allocate L2CAP manager. ingroup l2cap details This function allocates and initializes an L2CAP manager structure. One L2CAP manager manages all L2CAP connections for a particular local device. The local device is specified by the c hci_ctrl parameter. param hci_ctrl Pointer to the hci_ctrl_state structure that represents the local device (HCI controller) for which a L2CAP manager is to be allocated. return A pointer to the allocated L2CAP manager structure. The returned L2CAP manager should be freed by a call to bt_l2cap_free_mgr when it is no longer needed.
	bt_l2cap_allocate_psm	This is function bt_l2cap_allocate_psm.

	bt_l2cap_connect_ext	brief Connect to a remote device. ingroup l2cap details This function establishes an L2CAP connection with a remote device on a specific PSM. When connect operation completes a callback function is called. param mgr The L2CAP manager. param remote_addr The address of the remote device. param psm The PSM for the connection. param connect_cb The Callback function that is called when the connect operation completes. param param A pointer to arbitrary data to be passed to the c connect_cb callback. param state_cb The callback function that is called when the state of the established connection changes. return li c TRUE... more
	bt_l2cap_connect_fixed_channel	This is function bt_l2cap_connect_fixed_channel.
	bt_l2cap_disconnect	brief Close L2CAP channel. ingroup l2cap details This function closes an L2CAP channel. The channel can be established either by a call to bt_l2cap_connect() or by an incoming connection request. param ch The L2CAP channel to be closed. return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.
	bt_l2cap_echo	brief Send echo command
	bt_l2cap_find_fixed_channel	This is function bt_l2cap_find_fixed_channel.
	bt_l2cap_find_psm	This is function bt_l2cap_find_psm.
	bt_l2cap_free_cmd_buffer	This is function bt_l2cap_free_cmd_buffer.
	bt_l2cap_free_fixed_channel	This is function bt_l2cap_free_fixed_channel.
	bt_l2cap_free_frame_buffer	This is function bt_l2cap_free_frame_buffer.
	bt_l2cap_free_mgr	brief Release L2CAP manger. ingroup l2cap details This function releases the L2CAP manager structure. param mgr The L2CAP manager structure to be released.
	bt_l2cap_free_psm	This is function bt_l2cap_free_psm.
	bt_l2cap_get_channel	This is function bt_l2cap_get_channel.
	bt_l2cap_get_channel_by_bdaddr_cid	This is function bt_l2cap_get_channel_by_bdaddr_cid.
	bt_l2cap_get_channel_by_hconn_cid	This is function bt_l2cap_get_channel_by_hconn_cid.
	bt_l2cap_get_channel_by_hconn_dest_cid	This is function bt_l2cap_get_channel_by_hconn_dest_cid.
	bt_l2cap_get_channel_by_psm	This is function bt_l2cap_get_channel_by_psm.
	bt_l2cap_get_mgr	This is function bt_l2cap_get_mgr.
	bt_l2cap_init	brief Initialize the L2CAP layer. ingroup l2cap details This function initializes the L2CAP layer of the stack. It must be called prior to any other L2CAP function can be called.
	bt_l2cap_init_channels	This is function bt_l2cap_init_channels.
	bt_l2cap_init_cmd_buffers	This is function bt_l2cap_init_cmd_buffers.
	bt_l2cap_init_frame_buffers	This is function bt_l2cap_init_frame_buffers.
	bt_l2cap_init_psms	This is function bt_l2cap_init_psms.
	bt_l2cap_listen_ext	brief Listen for incoming connections. ingroup l2cap details This function tells the L2CAP manager to listen for incoming connections on a specific PSM. When a connection is established a callback function is called. param mgr The L2CAP manager. param psm The PSM on which the manager will listen and accept incoming connections. param callback The callback function that will be called when an incoming connection is established. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter. return li c TRUE when the function succeeds. li c FALSE otherwise. The... more
	bt_l2cap_listen_fixed_channel	This is function bt_l2cap_listen_fixed_channel.
	bt_l2cap_packet_cmd_assembler	This is function bt_l2cap_packet_cmd_assembler.
	bt_l2cap_packet_data_assembler	This is function bt_l2cap_packet_data_assembler.
	bt_l2cap_read_data	This is function bt_l2cap_read_data.
	bt_l2cap_reject	brief Send reject command (used to reject unknown or invalid commands)
	bt_l2cap_send_cmd	This is function bt_l2cap_send_cmd.

	bt_l2cap_send_data	brief Send data over an L2CAP channel ingroup l2cap details This function sends data over the specified L2CAP channel. param channel The L2CAP channel to send data over. param data The pointer to the data. param len The length of the data. param callback The callback function that is called when sending the data has been completed. return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.
	bt_l2cap_update_conn_parameters	This is function bt_l2cap_update_conn_parameters.
	_bt_l2cap_clear_channel_cmd_queue	This is function _bt_l2cap_clear_channel_cmd_queue.
	_bt_l2cap_eretr_handle_xmit_event	This is function _bt_l2cap_eretr_handle_xmit_event.
	_bt_l2cap_eretr_retr_frames	This is function _bt_l2cap_eretr_retr_frames.
	_bt_l2cap_fcs	This is function _bt_l2cap_fcs.
	_bt_l2cap_get_tick_count	This is function _bt_l2cap_get_tick_count.
	_bt_l2cap_init_signal	This is function _bt_l2cap_init_signal.
	_bt_l2cap_init_timer	This is function _bt_l2cap_init_timer.
	_bt_l2cap_notify_and_remove	This is function _bt_l2cap_notify_and_remove.
	_bt_l2cap_send_commands_from_queue	This is function _bt_l2cap_send_commands_from_queue.
	_bt_l2cap_set_signal	This is function _bt_l2cap_set_signal.
	_l2cap_allocate_buffers	Defined by OEM through library configuration
	_l2cap_data_receive_callback	From l2cap_recv.c
	_pack_cmd_reject	This is function _pack_cmd_reject.
	_pack_config_request	This is function _pack_config_request.
	_pack_config_response	This is function _pack_config_response.
	_pack_conn_param_update_request	This is function _pack_conn_param_update_request.
	_pack_conn_param_update_response	This is function _pack_conn_param_update_response.
	_pack_conn_request	This is function _pack_conn_request.
	_pack_conn_response	This is function _pack_conn_response.
	_pack_dconn_request	This is function _pack_dconn_request.
	_pack_dconn_response	This is function _pack_dconn_response.
	_pack_echo_request	This is function _pack_echo_request.
	_pack_echo_response	This is function _pack_echo_response.
	_pack_info_request	This is function _pack_info_request.
	_pack_info_response	This is function _pack_info_response.
	_process_config_req	From cmd_config.c
	_process_config_res	This is function _process_config_res.
	_process_conn_param_update_req	This is function _process_conn_param_update_req.
	_process_conn_param_update_res	This is function _process_conn_param_update_res.
	_process_conn_req	From cmd_connect.c
	_process_conn_res	This is function _process_conn_res.
	_process_dconn_req	From cmd_disconnect.c
	_process_dconn_res	This is function _process_dconn_res.
	_process_echo_req	From cmd_echo.c
	_process_echo_res	This is function _process_echo_res.
	_process_info_req	From cmd_info.c
	_process_info_res	This is function _process_info_res.
	_process_reject	From cmd_reject.c
	_process_unknown_req	From cmd_unknown.c
	_process_unknown_res	This is function _process_unknown_res.
	_read_cmd_reject	This is function _read_cmd_reject.
	_read_config_request	This is function _read_config_request.
	_read_config_response	This is function _read_config_response.
	_read_conn_param_update_request	This is function _read_conn_param_update_request.
	_read_conn_param_update_response	This is function _read_conn_param_update_response.
	_read_conn_request	This is function _read_conn_request.
	_read_conn_response	This is function _read_conn_response.
	_read_dconn_request	This is function _read_dconn_request.

	_read_dconn_response	This is function _read_dconn_response.
	_read_echo_request	This is function _read_echo_request.
	_read_echo_response	This is function _read_echo_response.
	_read_info_request	This is function _read_info_request.
	_read_info_response	This is function _read_info_response.
	_bt_l2cap_eretr_pack_config_request	This is function _bt_l2cap_eretr_pack_config_request.
	_bt_l2cap_eretr_recv	This is function _bt_l2cap_eretr_recv.
	_bt_l2cap_eretr_send_data	This is function _bt_l2cap_eretr_send_data.
	_bt_l2cap_eretr_send_pending_frames	This is function _bt_l2cap_eretr_send_pending_frames.
	_bt_l2cap_eretr_send_smart_data	This is function _bt_l2cap_eretr_send_smart_data.
	_bt_l2cap_recv_req_seq_and_fbit	This is function _bt_l2cap_recv_req_seq_and_fbit.
	_bt_l2cap_send_ack	This is function _bt_l2cap_send_ack.
	_bt_l2cap_send_i_or_rr_or_rnr	This is function _bt_l2cap_send_i_or_rr_or_rnr.
	_bt_l2cap_send_rej	This is function _bt_l2cap_send_rej.
	_bt_l2cap_send_rnr	This is function _bt_l2cap_send_rnr.
	_bt_l2cap_send_rr	This is function _bt_l2cap_send_rr.
	_bt_l2cap_send_rr_or_rnr	This is function _bt_l2cap_send_rr_or_rnr.
	bt_l2cap_disconnect_ex	This is function bt_l2cap_disconnect_ex.
	bt_l2cap_free_channel	This is function bt_l2cap_free_channel.
	bt_l2cap_hci_has_open_channels	This is function bt_l2cap_hci_has_open_channels.
	bt_l2cap_is_channel_open	This is function bt_l2cap_is_channel_open.
	bt_l2cap_send_config	This is function bt_l2cap_send_config.
	bt_l2cap_send_smart_data	This is function bt_l2cap_send_smart_data.
	_bt_l2cap_process_connect_signal	This is function _bt_l2cap_process_connect_signal.

Misc. Data Types and Constants

	Name	Description
	CSR_SNK_ADC	This is macro CSR_SNK_ADC.
	CSR_SNK_FASTPIPE	This is macro CSR_SNK_FASTPIPE.
	CSR_SNK_FM	This is macro CSR_SNK_FM.
	CSR_SNK_I2S	This is macro CSR_SNK_I2S.
	CSR_SNK_L2CAP	This is macro CSR_SNK_L2CAP.
	CSR_SNK_PCM	This is macro CSR_SNK_PCM.
	CSR_SNK_SCO	This is macro CSR_SNK_SCO.
	CSR_SNK_SPDIF	This is macro CSR_SNK_SPDIF.
	CSR_SRC_ADC	This is macro CSR_SRC_ADC.
	CSR_SRC_FASTPIPE	This is macro CSR_SRC_FASTPIPE.
	CSR_SRC_FM	This is macro CSR_SRC_FM.
	CSR_SRC_I2S	This is macro CSR_SRC_I2S.
	CSR_SRC_L2CAP	This is macro CSR_SRC_L2CAP.
	CSR_SRC_MIC	This is macro CSR_SRC_MIC.
	CSR_SRC_PCM	This is macro CSR_SRC_PCM.
	CSR_SRC_SCO	This is macro CSR_SRC_SCO.
	CSR_SRC_SPDIF	This is macro CSR_SRC_SPDIF.
	CSR_VARID_CACHED_TEMPERATURE	This is macro CSR_VARID_CACHED_TEMPERATURE.
	CSR_VARID_ENABLE_SCO_STREAMS	This is macro CSR_VARID_ENABLE_SCO_STREAMS.
	CSR_VARID_MAP_SCO_AUDIO	This is macro CSR_VARID_MAP_SCO_AUDIO.
	CSR_VARID_PIO	This is macro CSR_VARID_PIO.
	CSR_VARID_PIO_DIRECTION_MASK	This is macro CSR_VARID_PIO_DIRECTION_MASK.
	CSR_VARID_PIO_PROTECT_MASK	This is macro CSR_VARID_PIO_PROTECT_MASK.
	CSR_VARID_RSSI_ACL	This is macro CSR_VARID_RSSI_ACL.
	CSR_VARID_STREAM_CLOSE_SINK	This is macro CSR_VARID_STREAM_CLOSE_SINK.
	CSR_VARID_STREAM_CLOSE_SOURCE	This is macro CSR_VARID_STREAM_CLOSE_SOURCE.
	CSR_VARID_STREAM_CONFIGURE	This is macro CSR_VARID_STREAM_CONFIGURE.
	CSR_VARID_STREAM_CONNECT	This is macro CSR_VARID_STREAM_CONNECT.

	CSR_VARID_STREAM_GET_SINK	This is macro CSR_VARID_STREAM_GET_SINK.
	CSR_VARID_STREAM_GET_SOURCE	This is macro CSR_VARID_STREAM_GET_SOURCE.
	GETRESP_BAD_REQ	This is macro GETRESP_BAD_REQ.
	GETRESP_ERROR	This is macro GETRESP_ERROR.
	GETRESP_NO_ACCESS	This is macro GETRESP_NO_ACCESS.
	GETRESP_NO_SUCH_VARID	This is macro GETRESP_NO_SUCH_VARID.
	GETRESP_NO_VALUE	This is macro GETRESP_NO_VALUE.
	GETRESP_OK	BCCMD GETRESP status codes
	GETRESP_PERMISSION_DENIED	This is macro GETRESP_PERMISSION_DENIED.
	GETRESP_READ_ONLY	This is macro GETRESP_READ_ONLY.
	GETRESP_TOO_BIG	This is macro GETRESP_TOO_BIG.
	GETRESP_WRITE_ONLY	This is macro GETRESP_WRITE_ONLY.
	H	PS key store (0 = use default)
	PS_DEFAULT	This is macro PS_DEFAULT.
	PS_F	This is macro PS_F.
	PS_I	This is macro PS_I.
	PS_RAM	This is macro PS_RAM.
	PS_ROM	This is macro PS_ROM.
	PSKEY_ANA_FREQ	This is macro PSKEY_ANA_FREQ.
	PSKEY_BDADDR	Select PS key definitions
	PSKEY_CLOCK_REQUEST_ENABLE	This is macro PSKEY_CLOCK_REQUEST_ENABLE.
	PSKEY_DEEP_SLEEP_CLEAR_RTS	This is macro PSKEY_DEEP_SLEEP_CLEAR_RTS.
	PSKEY_DEEP_SLEEP_STATE	This is macro PSKEY_DEEP_SLEEP_STATE.
	PSKEY_DEEP_SLEEP_USE_EXTERNAL_CLOCK	This is macro PSKEY_DEEP_SLEEP_USE_EXTERNAL_CLOCK.
	PSKEY_DEEP_SLEEP_WAKE_CTS	This is macro PSKEY_DEEP_SLEEP_WAKE_CTS.
	PSKEY_DIGITAL_AUDIO_BITS_PER_SAMPLE	This is macro PSKEY_DIGITAL_AUDIO_BITS_PER_SAMPLE.
	PSKEY_DIGITAL_AUDIO_CONFIG	This is macro PSKEY_DIGITAL_AUDIO_CONFIG.
	PSKEY_HCI_NOP_DISABLE	This is macro PSKEY_HCI_NOP_DISABLE.
	PSKEY_HOST_INTERFACE	This is macro PSKEY_HOST_INTERFACE.
	PSKEY_HOSTIO_MAP_SCO_PCM	This is macro PSKEY_HOSTIO_MAP_SCO_PCM.
	PSKEY_HOSTIO_UART_RESET_TIMEOUT	This is macro PSKEY_HOSTIO_UART_RESET_TIMEOUT.
	PSKEY_PCM_PULL_CONTROL	This is macro PSKEY_PCM_PULL_CONTROL.
	PSKEY_PIO_DEEP_SLEEP_EITHER_LEVEL	This is macro PSKEY_PIO_DEEP_SLEEP_EITHER_LEVEL.
	PSKEY_UART_BAUDRATE	This is macro PSKEY_UART_BAUDRATE.
	PSKEY_UART_BITRATE	This is macro PSKEY_UART_BITRATE.
	PSKEY_UART_CONFIG_BCSP	This is macro PSKEY_UART_CONFIG_BCSP.
	PSKEY_UART_CONFIG_H4	This is macro PSKEY_UART_CONFIG_H4.
	PSKEY_UART_CONFIG_H5	This is macro PSKEY_UART_CONFIG_H5.
	PSKEY_UART_TX_WINDOW_SIZE	This is macro PSKEY_UART_TX_WINDOW_SIZE.
	PSKEY_VM_DISABLE	This is macro PSKEY_VM_DISABLE.
	SET_PS_VALUE_BDADDR	This is macro SET_PS_VALUE_BDADDR.
	SET_PS_VALUE_UINT16	Macros for defining lists of PS values for use with btx_csr_set_ps_vars().
	SET_PS_VALUE_UINT32	This is macro SET_PS_VALUE_UINT32.
	W	This is macro W.
	BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE	This is macro BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE.
	btx_ti_get_script__CC2560_ServicePack	Alias for the latest script
	btx_ti_get_script__CC2564_BLE_Init	This is macro btx_ti_get_script__CC2564_BLE_Init.
	btx_ti_get_script__CC2564_ServicePack	Aliases for latest scripts
	btx_ti_get_script__CC2564B_BLE_Init	This is macro btx_ti_get_script__CC2564B_BLE_Init.
	btx_ti_get_script__CC2564B_ServicePack	Aliases for latest scripts
	BTX_TI_MODULATION_SCHEME_8_DPSK	This is macro BTX_TI_MODULATION_SCHEME_8_DPSK.
	BTX_TI_MODULATION_SCHEME_CW	This is macro BTX_TI_MODULATION_SCHEME_CW.
	BTX_TI_MODULATION_SCHEME_GFSK	This is macro BTX_TI_MODULATION_SCHEME_GFSK.
	BTX_TI_MODULATION_SCHEME_P4_DPSK	This is macro BTX_TI_MODULATION_SCHEME_P4_DPSK.

	BTX_TI_MODULATION_TYPE_EDR2	This is macro BTX_TI_MODULATION_TYPE_EDR2.
	BTX_TI_MODULATION_TYPE_EDR3	This is macro BTX_TI_MODULATION_TYPE_EDR3.
	BTX_TI_MODULATION_TYPE_GFSK	This is macro BTX_TI_MODULATION_TYPE_GFSK.
	BTX_TI_TEST_PATTERN_ALL_0	This is macro BTX_TI_TEST_PATTERN_ALL_0.
	BTX_TI_TEST_PATTERN_ALL_1	This is macro BTX_TI_TEST_PATTERN_ALL_1.
	BTX_TI_TEST_PATTERN_F0F0	This is macro BTX_TI_TEST_PATTERN_F0F0.
	BTX_TI_TEST_PATTERN_FF00	This is macro BTX_TI_TEST_PATTERN_FF00.
	BTX_TI_TEST_PATTERN_PN15	This is macro BTX_TI_TEST_PATTERN_PN15.
	BTX_TI_TEST_PATTERN_PN9	This is macro BTX_TI_TEST_PATTERN_PN9.
	BTX_TI_TEST_PATTERN_USER	This is macro BTX_TI_TEST_PATTERN_USER.
	BTX_TI_TEST_PATTERN_Z0Z0	This is macro BTX_TI_TEST_PATTERN_Z0Z0.
	_W	This is macro _W.
	_btx_csr_autobaud_buffer_t	This is record _btx_csr_autobaud_buffer_t.
	_btx_csr_bccmd_header_s	This is type btx_csr_bccmd_header_t.
	_btx_csr_cached_temperature_s	This is type btx_csr_cached_temperature_t.
	_btx_csr_exec_script_buffer_t	This is record _btx_csr_exec_script_buffer_t.
	_btx_csr_pio_direction_mask_s	This is type btx_csr_pio_direction_mask_t.
	_btx_csr_pio_protection_mask_s	This is type btx_csr_pio_protection_mask_t.
	_btx_csr_pio_s	This is type btx_csr_pio_t.
	_btx_csr_rssi_acl_s	This is type btx_csr_rssi_acl_t.
	_btx_csr_script_t	This is type btx_csr_script_t.
	_btx_csr_set_ps_vars_buffer_t	This is record _btx_csr_set_ps_vars_buffer_t.
	_btx_csr_strm_get_sink_s	This is type btx_csr_strm_get_sink_t.
	_btx_csr_strm_get_source_s	This is type btx_csr_strm_get_source_t.
	_btx_csr_var_u	This is type btx_csr_var_t.
	_btx_ti_exec_script_buffer_t	This is record _btx_ti_exec_script_buffer_t.
	_btx_ti_exec_script_oem_buffer_t	This is record _btx_ti_exec_script_oem_buffer_t.
	_btx_ti_script_t	This is type btx_ti_script_t.
	btx_csr_autobaud_buffer_t	This is type btx_csr_autobaud_buffer_t.
	btx_csr_autobaud_callback_fp	This is type btx_csr_autobaud_callback_fp.
	btx_csr_bccmd_header_t	This is type btx_csr_bccmd_header_t.
	btx_csr_cached_temperature_t	This is type btx_csr_cached_temperature_t.
	btx_csr_exec_script_buffer_t	This is type btx_csr_exec_script_buffer_t.
	btx_csr_exec_script_callback_fp	This is type btx_csr_exec_script_callback_fp.
	btx_csr_get_ps_var_callback_fp	This is type btx_csr_get_ps_var_callback_fp.
	btx_csr_get_var_callback_fp	This is type btx_csr_get_var_callback_fp.
	btx_csr_pio_direction_mask_t	This is type btx_csr_pio_direction_mask_t.
	btx_csr_pio_protection_mask_t	This is type btx_csr_pio_protection_mask_t.
	btx_csr_pio_t	This is type btx_csr_pio_t.
	btx_csr_rssi_acl_t	This is type btx_csr_rssi_acl_t.
	btx_csr_script_t	This is type btx_csr_script_t.
	btx_csr_set_ps_vars_buffer_t	This is type btx_csr_set_ps_vars_buffer_t.
	btx_csr_set_ps_vars_callback_fp	This is type btx_csr_set_ps_vars_callback_fp.
	btx_csr_set_var_callback_fp	This is type btx_csr_set_var_callback_fp.
	btx_csr_strm_get_sink_t	This is type btx_csr_strm_get_sink_t.
	btx_csr_strm_get_source_t	This is type btx_csr_strm_get_source_t.
	btx_csr_var_t	This is type btx_csr_var_t.
	btx_ti_completion_callback_fp	This is type btx_ti_completion_callback_fp.
	btx_ti_exec_script_buffer_t	This is type btx_ti_exec_script_buffer_t.
	btx_ti_exec_script_oem_buffer_t	This is type btx_ti_exec_script_oem_buffer_t.
	btx_ti_exec_script_oem_callback_fp	This is type btx_ti_exec_script_oem_callback_fp.
	btx_ti_script_t	This is type btx_ti_script_t.
	BTX_TI_A3DP_ROLE_SINK	This is macro BTX_TI_A3DP_ROLE_SINK.
	BTX_TI_A3DP_ROLE_SOURCE	This is macro BTX_TI_A3DP_ROLE_SOURCE.
	BTX_TI_AVPR_DISABLE	This is macro BTX_TI_AVPR_DISABLE.

	BTX_TI_AVPR_DO_NOT_LOAD_A3DP_CODE	This is macro BTX_TI_AVPR_DO_NOT_LOAD_A3DP_CODE.
	BTX_TI_AVPR_ENABLE	This is macro BTX_TI_AVPR_ENABLE.
	BTX_TI_AVPR_LOAD_A3DP_CODE	This is macro BTX_TI_AVPR_LOAD_A3DP_CODE.
	btx_ti_get_script__CC2564B_AVPR_Init	This is macro btx_ti_get_script__CC2564B_AVPR_Init.
	BTX_TI_SBC_ALLOCATION_METHOD_LOUDNESS	This is macro BTX_TI_SBC_ALLOCATION_METHOD_LOUDNESS.
	BTX_TI_SBC_ALLOCATION_METHOD_SNR	This is macro BTX_TI_SBC_ALLOCATION_METHOD_SNR.
	BTX_TI_SBC_BLOCKS_12	This is macro BTX_TI_SBC_BLOCKS_12.
	BTX_TI_SBC_BLOCKS_16	This is macro BTX_TI_SBC_BLOCKS_16.
	BTX_TI_SBC_BLOCKS_4	This is macro BTX_TI_SBC_BLOCKS_4.
	BTX_TI_SBC_BLOCKS_8	This is macro BTX_TI_SBC_BLOCKS_8.
	BTX_TI_SBC_CHANNEL_MODE_DUAL_CHANNEL	This is macro BTX_TI_SBC_CHANNEL_MODE_DUAL_CHANNEL.
	BTX_TI_SBC_CHANNEL_MODE_JOINT_STEREO	This is macro BTX_TI_SBC_CHANNEL_MODE_JOINT_STEREO.
	BTX_TI_SBC_CHANNEL_MODE_MONO	This is macro BTX_TI_SBC_CHANNEL_MODE_MONO.
	BTX_TI_SBC_CHANNEL_MODE_STEREO	This is macro BTX_TI_SBC_CHANNEL_MODE_STEREO.
	BTX_TI_SBC_SAMPLE_FREQUENCY_16000	This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_16000.
	BTX_TI_SBC_SAMPLE_FREQUENCY_32000	This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_32000.
	BTX_TI_SBC_SAMPLE_FREQUENCY_44100	This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_44100.
	BTX_TI_SBC_SAMPLE_FREQUENCY_48000	This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_48000.
	CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN	This is macro CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN.
	CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN_ENABLE	This is macro CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN_ENABLE.
	CSR_I2S_STREAM_CFG_KEY_BITS_PER_SAMPLE	This is macro CSR_I2S_STREAM_CFG_KEY_BITS_PER_SAMPLE.
	CSR_I2S_STREAM_CFG_KEY_CROP_ENABLE	This is macro CSR_I2S_STREAM_CFG_KEY_CROP_ENABLE.
	CSR_I2S_STREAM_CFG_KEY_JUSTIFY_DELAY	This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_DELAY.
	CSR_I2S_STREAM_CFG_KEY_JUSTIFY_FORMAT	This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_FORMAT.
	CSR_I2S_STREAM_CFG_KEY_JUSTIFY_RESOLUTION	This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_RESOLUTION.
	CSR_I2S_STREAM_CFG_KEY_MASTER	This is macro CSR_I2S_STREAM_CFG_KEY_MASTER.
	CSR_I2S_STREAM_CFG_KEY_MCLK	This is macro CSR_I2S_STREAM_CFG_KEY_MCLK.
	CSR_I2S_STREAM_CFG_KEY_POLARITY	This is macro CSR_I2S_STREAM_CFG_KEY_POLARITY.
	CSR_I2S_STREAM_CFG_KEY_RX_START_SAMPLE	This is macro CSR_I2S_STREAM_CFG_KEY_RX_START_SAMPLE.
	CSR_I2S_STREAM_CFG_KEY_SYNC_RATE	This is macro CSR_I2S_STREAM_CFG_KEY_SYNC_RATE.
	CSR_I2S_STREAM_CFG_KEY_TX_START_SAMPLE	This is macro CSR_I2S_STREAM_CFG_KEY_TX_START_SAMPLE.
	CSR_MAX_HQ_PACKET_LEN	This is macro CSR_MAX_HQ_PACKET_LEN.
	CSR_MSG_TYPE_GETREQ	This is macro CSR_MSG_TYPE_GETREQ.
	CSR_MSG_TYPE_GETRESP	This is macro CSR_MSG_TYPE_GETRESP.
	CSR_MSG_TYPE_SETREQ	This is macro CSR_MSG_TYPE_SETREQ.
	CSR_PCM_STREAM_CFG_KEY_LSB_FIRST	This is macro CSR_PCM_STREAM_CFG_KEY_LSB_FIRST.
	CSR_PCM_STREAM_CFG_KEY_MANCHESTER	This is macro CSR_PCM_STREAM_CFG_KEY_MANCHESTER.
	CSR_PCM_STREAM_CFG_KEY_MANCHESTER_SLAVE	This is macro CSR_PCM_STREAM_CFG_KEY_MANCHESTER_SLAVE.
	CSR_PCM_STREAM_CFG_KEY_MASTER	This is macro CSR_PCM_STREAM_CFG_KEY_MASTER.
	CSR_PCM_STREAM_CFG_KEY_MCLK	This is macro CSR_PCM_STREAM_CFG_KEY_MCLK.
	CSR_PCM_STREAM_CFG_KEY_SHORT_SYNC	This is macro CSR_PCM_STREAM_CFG_KEY_SHORT_SYNC.
	CSR_PCM_STREAM_CFG_KEY_SIGN_EXTEND	This is macro CSR_PCM_STREAM_CFG_KEY_SIGN_EXTEND.
	CSR_PCM_STREAM_CFG_KEY_SLOT_COUNT	This is macro CSR_PCM_STREAM_CFG_KEY_SLOT_COUNT.
	CSR_PCM_STREAM_CFG_KEY_SYNC_RATE	This is macro CSR_PCM_STREAM_CFG_KEY_SYNC_RATE.
	CSR_PCM_STREAM_CFG_KEY_TX_TRISTATE	This is macro CSR_PCM_STREAM_CFG_KEY_TX_TRISTATE.
	CSR_VARID_CAPABILITY_DOWNLOAD_INDICATION	This is macro CSR_VARID_CAPABILITY_DOWNLOAD_INDICATION.
	CSR_VARID_CREATE_OPERATOR_C	This is macro CSR_VARID_CREATE_OPERATOR_C.
	CSR_VARID_CREATE_OPERATOR_P	This is macro CSR_VARID_CREATE_OPERATOR_P.
	CSR_VARID_DESTROY_OPERATORS	This is macro CSR_VARID_DESTROY_OPERATORS.
	CSR_VARID_DSPMANAGER_CONFIG_REQUEST	This is macro CSR_VARID_DSPMANAGER_CONFIG_REQUEST.

	CSR_VARID_MESSAGE_FROM_OPERATOR	Select Variable definitions
	CSR_VARID_START_OPERATORS	This is macro CSR_VARID_START_OPERATORS.
	CSR_VARID_STOP_OPERATORS	This is macro CSR_VARID_STOP_OPERATORS.
	CSR_VARID_STREAM_DRAINED_NOTIFICATION	This is macro CSR_VARID_STREAM_DRAINED_NOTIFICATION.
	CSR_VARID_STREAM_TRANSFORM_DISCONNECT	This is macro CSR_VARID_STREAM_TRANSFORM_DISCONNECT.
	PSKEY_BLE_DEFAULT_TX_POWER	This is macro PSKEY_BLE_DEFAULT_TX_POWER.
	PSKEY_DEEP_SLEEP_EXTERNAL_CLOCK_SOURCE_PIO	This is macro PSKEY_DEEP_SLEEP_EXTERNAL_CLOCK_SOURCE_PIO.
	PSKEY_H_HC_FC_MAX_SCO_PKT_LEN	This is macro PSKEY_H_HC_FC_MAX_SCO_PKT_LEN.
	PSKEY_H_HC_FC_MAX_SCO_PKTS	This is macro PSKEY_H_HC_FC_MAX_SCO_PKTS.
	PSKEY_PCM_CLOCK_RATE	This is macro PSKEY_PCM_CLOCK_RATE.
	PSKEY_PCM_CONFIG32	This is macro PSKEY_PCM_CONFIG32.
	PSKEY_PCM_FORMAT	This is macro PSKEY_PCM_FORMAT.
	PSKEY_PCM_SYNC_RATE	This is macro PSKEY_PCM_SYNC_RATE.
	PSKEY_PCM_USE_LOW_JITTER_MODE	This is macro PSKEY_PCM_USE_LOW_JITTER_MODE.
	PSKEY_UART_CONFIG_H4DS	This is macro PSKEY_UART_CONFIG_H4DS.
	BT_LOG_LEVEL_A2DP_PAKCET	This is macro BT_LOG_LEVEL_A2DP_PAKCET.
	DCRH	Number of configuration blocks
	btx_csr_bccmd_callback_fp	This is type btx_csr_bccmd_callback_fp.
	btx_csr_bccmd_listener_t	This is type btx_csr_bccmd_listener_t.
	btx_csr_create_operator_c_t	This is type btx_csr_create_operator_c_t.
	btx_csr_exec_hq_script_buffer_t	This is type btx_csr_exec_hq_script_buffer_t.
	btx_csr_exec_hq_script_callback_fp	This is type btx_csr_exec_hq_script_callback_fp.
	btx_csr_get_script_fp	This is type btx_csr_get_script_fp.
	btx_csr_strm_connect_t	This is type btx_csr_strm_connect_t.
	btx_ti_codec_config_t	This is type btx_ti_codec_config_t.
	_btx_csr_bccmd_listener_t	This is record _btx_csr_bccmd_listener_t.
	_btx_csr_create_operator_c_s	This is type btx_csr_create_operator_c_t.
	_btx_csr_exec_hq_script_buffer_t	This is record _btx_csr_exec_hq_script_buffer_t.
	_btx_csr_strm_connect_s	This is type btx_csr_strm_connect_t.
	_btx_ti_codec_config_s	This is type btx_ti_codec_config_t.
	PSKEY_LC_DEFAULT_TX_POWER	This is macro PSKEY_LC_DEFAULT_TX_POWER.
	PSKEY_LC_MAX_TX_POWER	This is macro PSKEY_LC_MAX_TX_POWER.
	PSKEY_LC_MAX_TX_POWER_NO_RSSI	This is macro PSKEY_LC_MAX_TX_POWER_NO_RSSI.

Misc. Functions

	Name	Description
	btx_csr_alloc_bccmd_getreq	This is function btx_csr_alloc_bccmd_getreq.
	btx_csr_alloc_bccmd_setreq	This is function btx_csr_alloc_bccmd_setreq.
	btx_csr_autobaud	brief Configure controller's UART speed. ingroup btx_csr details This function makes the controller auto-configure its UART speed. The host transport must be set to H4. This function works only with BC6 controllers.
	btx_csr_bc7_sel_host_interface_h4	brief Configure controller's UART speed and host interface. ingroup btx_csr details This function makes the controller auto-configure its UART speed and select H4 as host interface. PS_KEY_HOST_INTERFACE must not be set. PS_KEY_UART_BITRATE must be set to 0. This function works only with BC7 controllers.
	btx_csr_enable_tx	brief Enable/disable transmitter ingroup btx_csr param enable Specifies whether the transmitter should be enable or disabled. param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

	btx_csr_exec_script	brief Patch controller's firmware ingroup btx_csr details This function executes a script that patches the controller's firmware. The c script must point to a structure that contain a complete patch script for a particular controller model and revision. If the revision specified in the script and revision read from the controller are the same btx_csr_patch_controller() loads the script to the controller and calls the c callback with the first parameter TRUE. Otherwise the c callback is called with the first parameter FALSE. If support for multiple firmware revisions is neede use btx_csr_patch_controller(). param script Array of... more
	btx_csr_get_cached_temperature	brief Get chip's cached temperature ingroup btx_csr param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	btx_csr_get_ps_var	This is function btx_csr_get_ps_var.
	btx_csr_get_ps_var_ex	This is function btx_csr_get_ps_var_ex.
	btx_csr_get_rssi_acl	brief Get RSSI ingroup btx_csr details This function retrieves the RSSI for a given HCI ACL handle. param hconn ACL connection handle. param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	btx_csr_get_script__PB_27_R20_BC6ROM_A04	brief Return script for patching BlueCore 6 ingroup btx_csr
	btx_csr_get_script__PB_90_REV6	brief Return script for patching CSR8810 (BlueCore 7) ingroup btx_csr
	btx_csr_get_var	This is function btx_csr_get_var.
	btx_csr_set_ps_var	This is function btx_csr_set_ps_var.
	btx_csr_set_ps_var_ex	This is function btx_csr_set_ps_var_ex.
	btx_csr_set_ps_vars	brief Write PS variables ingroup btx_csr details param ps_vars PS values param buffer A buffer for storing temporary data during function execution. param callback The callback function that will be called when all PS values have been sent to the controller or error occurred. param callback_param A pointer to arbitrary data to be passed to the c callback callback..
	btx_csr_set_ps_vars_ex	brief Write PS variables ingroup btx_csr param ps_vars PS values param buffer A buffer for storing temporary data during function execution. param store param callback The callback function that will be called when all PS values have been sent to the controller or error occurred. param callback_param A pointer to arbitrary data to be passed to the c callback callback..
	btx_csr_set_var	This is function btx_csr_set_var.
	btx_csr_warm_reset	brief Warm reset ingroup btx_csr details This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact.
	btx_csr_warm_reset_ex	brief Warm reset ingroup btx_csr details This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact. Since the controller does not respond to the warm reset command as it starts resetting immediately upon receiving the command, the c callback is called right after the command packet has been transmitted to the controller. param callback The callback function that will be called after the warm reset command has been sent to the controller. param callback_param A pointer to arbitrary data to be passed to the c callback callback.... more
	btx_ti_drpb_enable_rf_calibration	This is function btx_ti_drpb_enable_rf_calibration.
	btx_ti_drpb_set_power_vector	This is function btx_ti_drpb_set_power_vector.

	btx_ti_drpb_tester_con_tx	This is function btx_ti_drpb_tester_con_tx.
	btx_ti_enable_deep_sleep	This is function btx_ti_enable_deep_sleep.
	btx_ti_enable_fast_clock_crystal	This is function btx_ti_enable_fast_clock_crystal.
	btx_ti_enable_low_power_scan	This is function btx_ti_enable_low_power_scan.
	btx_ti_enable_low_power_scan_default	This is function btx_ti_enable_low_power_scan_default.
	btx_ti_exec_script	This is function btx_ti_exec_script.
	btx_ti_exec_script_oem	This is function btx_ti_exec_script_oem.
	btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_36	CC2560 Scripts (Service pack 2.36)
	btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_44	CC2560 Scripts (Service pack 2.44)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_10	CC2564 Scripts (Service pack 2.10)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_10_BLE_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_10_BLE_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_12	CC2564 Scripts (Service pack 2.12)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_12_BLE_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_12_BLE_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_7	CC2564 Scripts (Service pack 2.7)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_AVPR_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_AVPR_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_BLE_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_BLE_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_Short	CC2564 Scripts (Service pack 2.8)
	btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_AVPR_AddOn	This is function btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_AVPR_AddOn.
	btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_BLE_AddOn	This is function btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_0_1	CC2564B Scripts (Service pack 0.1)
	btx_ti_get_script__CC2564B_BT_Service_Pack_0_1_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_0_1_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_0_2	CC2564B Scripts (Service pack 0.2)
	btx_ti_get_script__CC2564B_BT_Service_Pack_0_2_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_0_2_BLE_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_3	CC2564 Scripts (Service pack 2.3)
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_AVPR_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_AVPR_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_BLE_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_BLE_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_DC2DC_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_DC2DC_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_4	CC2564 Scripts (Service pack 2.4)
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_AVPR_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_AVPR_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_BLE_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_BLE_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_DC2DC_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_DC2DC_AddOn.
	btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3	CC2564 Scripts (Service pack 1.3)
	btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3_BLE_Init	This is function btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3_BLE_Init.

	btx_ti_init_ble_controller	This is function btx_ti_init_ble_controller.
	btx_ti_init_controller	This is function btx_ti_init_controller.
	btx_ti_set_uart_baud_rate	This is function btx_ti_set_uart_baud_rate.
	btx_ti_write_bdaddr	This is function btx_ti_write_bdaddr.
	btx_ti_write_hardware_register	This is function btx_ti_write_hardware_register.
	btx_csr_get_script__dsp_script__PB_109_DSP_rev8	brief Return script for patching DSP in CSR8x11 A08 (BlueCore 7) ingroup btx_csr
	btx_csr_get_script__PB_101_CSR8811_CSP28_UART	brief Return script for patching CSR8x11 A06 (BlueCore 7) ingroup btx_csr
	btx_csr_get_script__PB_109_CSR8811_REV16	brief Return script for patching CSR8x11 A08 (BlueCore 7) ingroup btx_csr
	btx_csr_get_script__PB_173_CSR8X11_REV1	brief Return script for patching CSR8x11 A12 (BlueCore 7) ingroup btx_csr
	btx_csr_init	brief Initialize CSR support layer. ingroup btx_csr details This function initializes all internal variables of the CSR support layer. CSR controllers use vendor specific event (0xFF) to carry the BCCMD protocol. They also do not report number of completed packets for BCCMD commands. This function installs a vendor specific event handler that makes sure that callback are called on corresponding vendor specific commands and the number of free command buffers in the controller is kept correct.
	btx_csr_init_hq_script	This is function btx_csr_init_hq_script.
	btx_csr_patch_controller	brief Patch controller's firmware ingroup btx_csr details This function executes a script that patches the controller's firmware. Each entry of the c scripts array must be a complete patch script for a particular controller model and revision. btx_csr_patch_controller() reads the revision number from the controller then tries to find the corresponding script in the c scripts. If there is a matching script it is loaded to the controller and c callback is called with the first parameter TRUE. If no suitable script found c callback is called with the first parameter FALSE. param scripts Array of patch scripts.... more
	btx_csr_register_bccmd_listener	This is function btx_csr_register_bccmd_listener.
	btx_csr_send_dsp_config_data	This is function btx_csr_send_dsp_config_data.
	btx_csr_send_next_hq_script_packet	This is function btx_csr_send_next_hq_script_packet.
	btx_csr_unregister_bccmd_listener	This is function btx_csr_unregister_bccmd_listener.
	btx_ti_a3dp_sink_close_stream	This is function btx_ti_a3dp_sink_close_stream.
	btx_ti_a3dp_sink_codec_config	This is function btx_ti_a3dp_sink_codec_config.
	btx_ti_a3dp_sink_open_stream	This is function btx_ti_a3dp_sink_open_stream.
	btx_ti_a3dp_sink_start_stream	This is function btx_ti_a3dp_sink_start_stream.
	btx_ti_a3dp_sink_stop_stream	This is function btx_ti_a3dp_sink_stop_stream.
	btx_ti_avpr_debug	This is function btx_ti_avpr_debug.
	btx_ti_avpr_enable	This is function btx_ti_avpr_enable.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_14	CC2564 Scripts (Service pack 2.14)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_0	CC2564B Scripts (Service pack 1.0)
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_0_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_0_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_1	CC2564B Scripts (Service pack 1.1)
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_1_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_1_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_2	CC2564B Scripts (Service pack 1.2)
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn.

	btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn.
	btx_ti_le_enable	This is function btx_ti_le_enable.
	btx_ti_set_afh_mode	This is function btx_ti_set_afh_mode.
	btx_ti_write_codec_config	This is function btx_ti_write_codec_config.
	btx_csr_exec_hq_script	This is function btx_csr_exec_hq_script.

Platform Data Types and Constants

	Name	Description
	BYTE_SIZE	This is macro BYTE_SIZE.
	bt_byte	This is type bt_byte.
	bt_int	This is type bt_int.
	bt_long	This is type bt_long.
	bt_uint	This is type bt_uint.
	bt_ulong	This is type bt_ulong.

RFCOMM Data Types and Constants

	Name	Description
	MK_CMD_ADDRESS	define RFCOMM_MS_RTC
	MK_DLCI	This is macro MK_DLCI.
	RFCOMM_CFC_ENABLED	This is macro RFCOMM_CFC_ENABLED.
	RFCOMM_CFC_LOCAL_CREDIT	This is macro RFCOMM_CFC_LOCAL_CREDIT.
	RFCOMM_CFC_MAX_INITIAL_CREDIT	This is macro RFCOMM_CFC_MAX_INITIAL_CREDIT.
	RFCOMM_CMD_STATUS_FC_PENDING	This is macro RFCOMM_CMD_STATUS_FC_PENDING.
	RFCOMM_CMD_STATUS_PENDING	This is macro RFCOMM_CMD_STATUS_PENDING.
	RFCOMM_CMD_STATUS_WAITING_RESPONSE	This is macro RFCOMM_CMD_STATUS_WAITING_RESPONSE.
	RFCOMM_COMMAND	This is macro RFCOMM_COMMAND.
	RFCOMM_CTL_MSG_CLD	This is macro RFCOMM_CTL_MSG_CLD.
	RFCOMM_CTL_MSG_FCOFF	This is macro RFCOMM_CTL_MSG_FCOFF.
	RFCOMM_CTL_MSG_FCON	This is macro RFCOMM_CTL_MSG_FCON.
	RFCOMM_CTL_MSG_MSC	This is macro RFCOMM_CTL_MSG_MSC.
	RFCOMM_CTL_MSG_NSC	This is macro RFCOMM_CTL_MSG_NSC.
	RFCOMM_CTL_MSG_PN	This is macro RFCOMM_CTL_MSG_PN.
	RFCOMM_CTL_MSG_PSC	This is macro RFCOMM_CTL_MSG_PSC.
	RFCOMM_CTL_MSG_RLS	This is macro RFCOMM_CTL_MSG_RLS.
	RFCOMM_CTL_MSG_RPN	This is macro RFCOMM_CTL_MSG_RPN.
	RFCOMM_CTL_MSG_SNC	This is macro RFCOMM_CTL_MSG_SNC.
	RFCOMM_CTL_MSG_TEST	This is macro RFCOMM_CTL_MSG_TEST.
	RFCOMM_DLC_CHANGED_CONN_STATE	This is macro RFCOMM_DLC_CHANGED_CONN_STATE.
	RFCOMM_DLC_CHANGED_REMOTE_MSC	This is macro RFCOMM_DLC_CHANGED_REMOTE_MSC.
	RFCOMM_DLC_CONNECTION_FAILED	This is macro RFCOMM_DLC_CONNECTION_FAILED.
	RFCOMM_DLC_STATE_CLOSED	This is macro RFCOMM_DLC_STATE_CLOSED.
	RFCOMM_DLC_STATE_OPEN	This is macro RFCOMM_DLC_STATE_OPEN.
	RFCOMM_DLCI_CONTROL	This is macro RFCOMM_DLCI_CONTROL.
	RFCOMM_DLCI_FREE	This is macro RFCOMM_DLCI_FREE.
	RFCOMM_ERR_DM	This is macro RFCOMM_ERR_DM.
	RFCOMM_ERR_SUCCESS	This is macro RFCOMM_ERR_SUCCESS.
	RFCOMM_ERR_TIMEOUT	This is macro RFCOMM_ERR_TIMEOUT.
	RFCOMM_FC_TYPE_AGREGATE	This is macro RFCOMM_FC_TYPE_AGREGATE.
	RFCOMM_FC_TYPE_CREDIT	This is macro RFCOMM_FC_TYPE_CREDIT.
	RFCOMM_FLAG_CR	This is macro RFCOMM_FLAG_CR.
	RFCOMM_FLAG_EA	This is macro RFCOMM_FLAG_EA.
	RFCOMM_FLAG_PF	This is macro RFCOMM_FLAG_PF.
	RFCOMM_FRAME_HEADER_LEN	This is macro RFCOMM_FRAME_HEADER_LEN.
	RFCOMM_FRAME_TYPE_DISC	This is macro RFCOMM_FRAME_TYPE_DISC.

	RFCOMM_FRAME_TYPE_DM	This is macro RFCOMM_FRAME_TYPE_DM.
	RFCOMM_FRAME_TYPE_SABM	This is macro RFCOMM_FRAME_TYPE_SABM.
	RFCOMM_FRAME_TYPE_UA	This is macro RFCOMM_FRAME_TYPE_UA.
	RFCOMM_FRAME_TYPE_UI	This is macro RFCOMM_FRAME_TYPE_UI.
	RFCOMM_FRAME_TYPE_UIH	This is macro RFCOMM_FRAME_TYPE_UIH.
	RFCOMM_LINE_STATUS_FRAMING	This is macro RFCOMM_LINE_STATUS_FRAMING.
	RFCOMM_LINE_STATUS_OVERRUN	This is macro RFCOMM_LINE_STATUS_OVERRUN.
	RFCOMM_LINE_STATUS_PARITY	This is macro RFCOMM_LINE_STATUS_PARITY.
	RFCOMM_MAX_INFO_LEN	This is macro RFCOMM_MAX_INFO_LEN.
	RFCOMM_MODEM_STATUS_DV	This is macro RFCOMM_MODEM_STATUS_DV.
	RFCOMM_MODEM_STATUS_FC	This is macro RFCOMM_MODEM_STATUS_FC.
	RFCOMM_MODEM_STATUS_IC	This is macro RFCOMM_MODEM_STATUS_IC.
	RFCOMM_MODEM_STATUS_RTC	This is macro RFCOMM_MODEM_STATUS_RTC.
	RFCOMM_MODEM_STATUS_RTR	This is macro RFCOMM_MODEM_STATUS_RTR.
	RFCOMM_MX_MSG_FCOFF	This is macro RFCOMM_MX_MSG_FCOFF.
	RFCOMM_MX_MSG_FCON	This is macro RFCOMM_MX_MSG_FCON.
	RFCOMM_MX_MSG_MSC	This is macro RFCOMM_MX_MSG_MSC.
	RFCOMM_MX_MSG_NSC	This is macro RFCOMM_MX_MSG_NSC.
	RFCOMM_MX_MSG_PN	This is macro RFCOMM_MX_MSG_PN.
	RFCOMM_MX_MSG_RLS	This is macro RFCOMM_MX_MSG_RLS.
	RFCOMM_MX_MSG_RPN	This is macro RFCOMM_MX_MSG_RPN.
	RFCOMM_MX_MSG_TEST	This is macro RFCOMM_MX_MSG_TEST.
	RFCOMM_RESPONSE	This is macro RFCOMM_RESPONSE.
	RFCOMM_ROLE_INITIATOR	This is macro RFCOMM_ROLE_INITIATOR.
	RFCOMM_ROLE_RESPONDER	This is macro RFCOMM_ROLE_RESPONDER.
	RFCOMM_RPN_BAUD_RATE_1152	This is macro RFCOMM_RPN_BAUD_RATE_1152.
	RFCOMM_RPN_BAUD_RATE_192	This is macro RFCOMM_RPN_BAUD_RATE_192.
	RFCOMM_RPN_BAUD_RATE_2304	This is macro RFCOMM_RPN_BAUD_RATE_2304.
	RFCOMM_RPN_BAUD_RATE_24	This is macro RFCOMM_RPN_BAUD_RATE_24.
	RFCOMM_RPN_BAUD_RATE_384	This is macro RFCOMM_RPN_BAUD_RATE_384.
	RFCOMM_RPN_BAUD_RATE_48	This is macro RFCOMM_RPN_BAUD_RATE_48.
	RFCOMM_RPN_BAUD_RATE_576	This is macro RFCOMM_RPN_BAUD_RATE_576.
	RFCOMM_RPN_BAUD_RATE_72	This is macro RFCOMM_RPN_BAUD_RATE_72.
	RFCOMM_RPN_BAUD_RATE_96	default
	RFCOMM_RPN_DATA_BIT_5	This is macro RFCOMM_RPN_DATA_BIT_5.
	RFCOMM_RPN_DATA_BIT_6	This is macro RFCOMM_RPN_DATA_BIT_6.
	RFCOMM_RPN_DATA_BIT_7	This is macro RFCOMM_RPN_DATA_BIT_7.
	RFCOMM_RPN_DATA_BIT_8	default
	RFCOMM_RPN_FLC_N	This is macro RFCOMM_RPN_FLC_N.
	RFCOMM_RPN_FLC_RTC_INPUT	This is macro RFCOMM_RPN_FLC_RTC_INPUT.
	RFCOMM_RPN_FLC_RTC_OUTPUT	This is macro RFCOMM_RPN_FLC_RTC_OUTPUT.
	RFCOMM_RPN_FLC_RTR_INPUT	This is macro RFCOMM_RPN_FLC_RTR_INPUT.
	RFCOMM_RPN_FLC_RTR_OUTPUT	This is macro RFCOMM_RPN_FLC_RTR_OUTPUT.
	RFCOMM_RPN_FLC_XONOFF_INPUT	This is macro RFCOMM_RPN_FLC_XONOFF_INPUT.
	RFCOMM_RPN_FLC_XONOFF_OUTPUT	This is macro RFCOMM_RPN_FLC_XONOFF_OUTPUT.
	RFCOMM_RPN_PARITY_EVEN	This is macro RFCOMM_RPN_PARITY_EVEN.
	RFCOMM_RPN_PARITY_MARK	This is macro RFCOMM_RPN_PARITY_MARK.
	RFCOMM_RPN_PARITY_N	default
	RFCOMM_RPN_PARITY_ODD	This is macro RFCOMM_RPN_PARITY_ODD.
	RFCOMM_RPN_PARITY_SPACE	This is macro RFCOMM_RPN_PARITY_SPACE.
	RFCOMM_RPN_PARITY_Y	This is macro RFCOMM_RPN_PARITY_Y.
	RFCOMM_RPN_STOP_BIT_1	default
	RFCOMM_RPN_STOP_BIT_1_5	This is macro RFCOMM_RPN_STOP_BIT_1_5.
	RFCOMM_RPN_XOFF_DEFAULT	This is macro RFCOMM_RPN_XOFF_DEFAULT.
	RFCOMM_RPN_XON_DEFAULT	This is macro RFCOMM_RPN_XON_DEFAULT.

	RFCOMM_SERIAL_PORT_CH_1	This is macro RFCOMM_SERIAL_PORT_CH_1.
	RFCOMM_SERIAL_PORT_CH_10	This is macro RFCOMM_SERIAL_PORT_CH_10.
	RFCOMM_SERIAL_PORT_CH_11	This is macro RFCOMM_SERIAL_PORT_CH_11.
	RFCOMM_SERIAL_PORT_CH_12	This is macro RFCOMM_SERIAL_PORT_CH_12.
	RFCOMM_SERIAL_PORT_CH_13	This is macro RFCOMM_SERIAL_PORT_CH_13.
	RFCOMM_SERIAL_PORT_CH_14	This is macro RFCOMM_SERIAL_PORT_CH_14.
	RFCOMM_SERIAL_PORT_CH_15	This is macro RFCOMM_SERIAL_PORT_CH_15.
	RFCOMM_SERIAL_PORT_CH_16	This is macro RFCOMM_SERIAL_PORT_CH_16.
	RFCOMM_SERIAL_PORT_CH_17	This is macro RFCOMM_SERIAL_PORT_CH_17.
	RFCOMM_SERIAL_PORT_CH_18	This is macro RFCOMM_SERIAL_PORT_CH_18.
	RFCOMM_SERIAL_PORT_CH_19	This is macro RFCOMM_SERIAL_PORT_CH_19.
	RFCOMM_SERIAL_PORT_CH_2	This is macro RFCOMM_SERIAL_PORT_CH_2.
	RFCOMM_SERIAL_PORT_CH_20	This is macro RFCOMM_SERIAL_PORT_CH_20.
	RFCOMM_SERIAL_PORT_CH_21	This is macro RFCOMM_SERIAL_PORT_CH_21.
	RFCOMM_SERIAL_PORT_CH_22	This is macro RFCOMM_SERIAL_PORT_CH_22.
	RFCOMM_SERIAL_PORT_CH_23	This is macro RFCOMM_SERIAL_PORT_CH_23.
	RFCOMM_SERIAL_PORT_CH_24	This is macro RFCOMM_SERIAL_PORT_CH_24.
	RFCOMM_SERIAL_PORT_CH_25	This is macro RFCOMM_SERIAL_PORT_CH_25.
	RFCOMM_SERIAL_PORT_CH_26	This is macro RFCOMM_SERIAL_PORT_CH_26.
	RFCOMM_SERIAL_PORT_CH_27	This is macro RFCOMM_SERIAL_PORT_CH_27.
	RFCOMM_SERIAL_PORT_CH_28	This is macro RFCOMM_SERIAL_PORT_CH_28.
	RFCOMM_SERIAL_PORT_CH_29	This is macro RFCOMM_SERIAL_PORT_CH_29.
	RFCOMM_SERIAL_PORT_CH_3	This is macro RFCOMM_SERIAL_PORT_CH_3.
	RFCOMM_SERIAL_PORT_CH_30	This is macro RFCOMM_SERIAL_PORT_CH_30.
	RFCOMM_SERIAL_PORT_CH_4	This is macro RFCOMM_SERIAL_PORT_CH_4.
	RFCOMM_SERIAL_PORT_CH_5	This is macro RFCOMM_SERIAL_PORT_CH_5.
	RFCOMM_SERIAL_PORT_CH_6	This is macro RFCOMM_SERIAL_PORT_CH_6.
	RFCOMM_SERIAL_PORT_CH_7	This is macro RFCOMM_SERIAL_PORT_CH_7.
	RFCOMM_SERIAL_PORT_CH_8	This is macro RFCOMM_SERIAL_PORT_CH_8.
	RFCOMM_SERIAL_PORT_CH_9	This is macro RFCOMM_SERIAL_PORT_CH_9.
	RFCOMM_SESSION_CHANGED_AFC	This is macro RFCOMM_SESSION_CHANGED_AFC.
	RFCOMM_SESSION_CHANGED_CONN_STATE	This is macro RFCOMM_SESSION_CHANGED_CONN_STATE.
	RFCOMM_SESSION_STATE_CONNECTED	This is macro RFCOMM_SESSION_STATE_CONNECTED.
	RFCOMM_SESSION_STATE_DISCONNECTED	This is macro RFCOMM_SESSION_STATE_DISCONNECTED.
	RFCOMM_SESSION_STATE_FREE	This is macro RFCOMM_SESSION_STATE_FREE.
	RFCOMM_TIMEOUT	This is macro RFCOMM_TIMEOUT.
	_bt_rfcomm_command_t	This is type bt_rfcomm_command_t.
	_bt_rfcomm_ctl_msg_t	This is type bt_rfcomm_ctl_msg_t.
	_bt_rfcomm_dlc_t	This is type bt_rfcomm_dlc_t.
	_bt_rfcomm_mgr_t	This is type bt_rfcomm_mgr_t.
	_bt_rfcomm_server_channel_t	This is type bt_rfcomm_server_channel_t.
	_bt_rfcomm_session_listener_t	This is type bt_rfcomm_session_listener_t.
	_bt_rfcomm_session_t	This is type bt_rfcomm_session_t.
	_bt_sdp_data_element_t	This is type bt_sdp_data_element_t.
	bt_rfcomm_cmd_callback_fp	This is type bt_rfcomm_cmd_callback_fp.
	bt_rfcomm_command_p	This is type bt_rfcomm_command_p.
	bt_rfcomm_command_t	This is type bt_rfcomm_command_t.
	bt_rfcomm_ctl_msg_p	This is type bt_rfcomm_ctl_msg_p.
	bt_rfcomm_ctl_msg_t	This is type bt_rfcomm_ctl_msg_t.
	bt_rfcomm_dlc_p	This is type bt_rfcomm_dlc_p.
	bt_rfcomm_dlc_state_callback_fp	This is type bt_rfcomm_dlc_state_callback_fp.
	bt_rfcomm_dlc_t	This is type bt_rfcomm_dlc_t.
	bt_rfcomm_mgr_t	This is type bt_rfcomm_mgr_t.
	bt_rfcomm_read_data_callback_fp	This is type bt_rfcomm_read_data_callback_fp.
	bt_rfcomm_send_data_callback_fp	This is type bt_rfcomm_send_data_callback_fp.

	bt_rfcomm_server_channel_t	This is type bt_rfcomm_server_channel_t.
	bt_rfcomm_session_listener_t	This is type bt_rfcomm_session_listener_t.
	bt_rfcomm_session_p	This is type bt_rfcomm_session_p.
	bt_rfcomm_session_t	This is type bt_rfcomm_session_t.
	bt_rfcomm_state_callback_fp	This is type bt_rfcomm_state_callback_fp.
	RFCOMM_ERR_INTERRUPTED	This is macro RFCOMM_ERR_INTERRUPTED.
	RFCOMM_MIX_INFO_LEN	This is macro RFCOMM_MIX_INFO_LEN.
	RFCOMM_MX_MSG_MAX_DATA_LEN	This is macro RFCOMM_MX_MSG_MAX_DATA_LEN.

RFCOMM Functions

	Name	Description
	bt_rfcomm_allocate_dlc	brief Allocate DLC. ingroup rfcomm details This function allocates a new DLC on the specified RFCOMM session. param session The RFCOMM session. param dlci DLCI of the new DLC. return li A pointer to the new DLC if the function succeeds. li c NULL otherwise.
	bt_rfcomm_allocate_session	brief Allocate RFCOMM session. ingroup rfcomm details This function allocates a new RFCOMM session. param l2cap_mgr The L2CAP manager on which the RFCOMM session is to be created. param callback The callback function that is called when session state changes. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter. return li A pointer to the new RFCOMM session structure if the function succeeds. li c NULL otherwise.
	bt_rfcomm_close_dlc	brief Close DLC. ingroup rfcomm details This function closes a DLC. If DLCI = 0, the parent RFCOMM session is also closed.
	bt_rfcomm_connect	brief Connect to a remote device. ingroup rfcomm details This function establishes an RFCOMM connection with a remote device and opens a data DLC. Changes in the session state are reported through a callback function specified when the session has been allocated via call to bt_rfcomm_allocate_session. Changes in the data DLC are reported through a callback function specified in this call. param remote_addr Address of the remote device. param server_channel A server channel of the remote RFCOMM server. param callback The callback function for reporting changes in DLC state opened by this call. param param An arbitrary data pointer... more
	bt_rfcomm_find_dlc	brief Find DLC
	bt_rfcomm_free_dlc	brief Release DLC. ingroup rfcomm details This function releases the specified DLC. param dlc The DLC to be released.
	bt_rfcomm_free_session	brief Release RFCOMM session. ingroup rfcomm details This function deallocates the specified RFCOMM session. This function does not disconnect the session. It just frees the memory used by the bt_rfcomm_session structure. The session has to be disconnected by calling bt_rfcomm_close_dlc with DLCI = 0 first. param session The RFCOMM session to be deallocated.
	bt_rfcomm_init	brief Initialize the RFCOMM layer. ingroup rfcomm details This function initializes the RFCOMM layer of the stack. It must be called prior to any other RFCOMM function can be called.
	bt_rfcomm_listen	brief Listen for incoming connections. ingroup rfcomm details This function enables incoming connections on the specified RFCOMM session. Changes in the session state are reported through a callback function. param server_channel A server channel on which the RFCOMM session will listen and accept incoming connections. param callback The callback function that is called when session state changes. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter. return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

	bt_rfcomm_open_dlc	brief Open DLC. ingroup rfcomm details This function opens the specified DLC. Before calling this function the RFCOMM session must be already open. This function is not to be used with DLCI = 0. Changes in DLC state are reported through a callback function. param dlc The DLC to open. param callback The callback function for reporting changes in DLC state. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter. return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in... more
	bt_rfcomm_register_listener	This is function bt_rfcomm_register_listener.
	bt_rfcomm_send_credit	This is function bt_rfcomm_send_credit.
	bt_rfcomm_send_data	brief Send data over a DLC. ingroup rfcomm details This function sends data over the specified DLC. Operation completion is reported through callback function. param dlc The DLC. param data A pointer to the data to be sent. param len Data length. param callback The callback function that is called when operation completes. return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.
	bt_rfcomm_unregister_listener	This is function bt_rfcomm_unregister_listener.
	check_fcs	From rfcomm_fcs.c
	read_command	From channel_cmd_recv.c
	rfcomm_cq_ack_cmd	This is function rfcomm_cq_ack_cmd.
	rfcomm_cq_ack_mx_cmd	This is function rfcomm_cq_ack_mx_cmd.
	rfcomm_send_cmd	This is function rfcomm_send_cmd.
	rfcomm_send_commands_from_queue	This is function rfcomm_send_commands_from_queue.
	rfcomm_send_mx_msc_cmd	This is function rfcomm_send_mx_msc_cmd.
	rfcomm_send_mx_pn_cmd	This is function rfcomm_send_mx_pn_cmd.
	_rfcomm_alloc_cmd_buffer	This is function _rfcomm_alloc_cmd_buffer.
	_rfcomm_allocate_buffers	Defined by OEM through library configuration
	_rfcomm_allocate_mx_cmd	This is function _rfcomm_allocate_mx_cmd.
	_rfcomm_free_cmd_buffer	This is function _rfcomm_free_cmd_buffer.
	_rfcomm_get_tick_count	This is function _rfcomm_get_tick_count.
	_rfcomm_init_cmd_buffers	From rfcomm_cmdbuffer.c
	_rfcomm_init_mgr	This is function _rfcomm_init_mgr.
	_rfcomm_init_sessions	From rfcomm_session.c
	_rfcomm_init_timer	This is function _rfcomm_init_timer.
	_rfcomm_l2cap_read_data_callback	From rfcomm_cmd_recv.c
	_rfcomm_mx_process_fc	This is function _rfcomm_mx_process_fc.
	_rfcomm_mx_process_pn	This is function _rfcomm_mx_process_pn.
	_rfcomm_mx_process_rpn	This is function _rfcomm_mx_process_rpn.
	_rfcomm_process_cmd_frame_disc	From frame_disc.c
	_rfcomm_process_cmd_frame_dm	This is function _rfcomm_process_cmd_frame_dm.
	_rfcomm_process_cmd_frame_sabm	This is function _rfcomm_process_cmd_frame_sabm.
	_rfcomm_process_cmd_frame_ua	This is function _rfcomm_process_cmd_frame_ua.
	_rfcomm_process_cmd_frame_uih	From frame_uih.c
	_rfcomm_process_mx_fc_response	This is function _rfcomm_process_mx_fc_response.
	_rfcomm_process_mx_msc_response	This is function _rfcomm_process_mx_msc_response.
	_rfcomm_process_mx_pn_response	This is function _rfcomm_process_mx_pn_response.
	_rfcomm_process_mx_rls_response	This is function _rfcomm_process_mx_rls_response.
	_rfcomm_process_res_frame_disc	This is function _rfcomm_process_res_frame_disc.
	_rfcomm_process_res_frame_dm	This is function _rfcomm_process_res_frame_dm.
	_rfcomm_process_res_frame_sabm	This is function _rfcomm_process_res_frame_sabm.
	_rfcomm_process_res_frame_ua	This is function _rfcomm_process_res_frame_ua.
	_rfcomm_process_res_frame_uih	This is function _rfcomm_process_res_frame_uih.
	_rfcomm_send_command	From rfcomm_cmd_send.c
	_rfcomm_send_dm_response	From frame_dm.c
	_rfcomm_send_mx_fc_cmd	This is function _rfcomm_send_mx_fc_cmd.
	_rfcomm_send_mx_msc_response	This is function _rfcomm_send_mx_msc_response.

	_rfcomm_send_mx_nsc_response	This is function _rfcomm_send_mx_nsc_response.
	_rfcomm_send_mx_rls_cmd	This is function _rfcomm_send_mx_rls_cmd.
	_rfcomm_send_mx_rls_response	This is function _rfcomm_send_mx_rls_response.
	_rfcomm_send_mx_test_response	This is function _rfcomm_send_mx_test_response.
	_rfcomm_send_sabm_cmd	From frame_sabm.c
	_rfcomm_send_ua_response	From frame_ua.c
	_bt_rfcomm_allocate_channel	This is function _bt_rfcomm_allocate_channel.
	_bt_rfcomm_clear_queue	From command_queue.c
	_bt_rfcomm_find_channel	This is function _bt_rfcomm_find_channel.
	_bt_rfcomm_get_mgr	This is function _bt_rfcomm_get_mgr.
	_bt_rfcomm_init_signal	This is function _bt_rfcomm_init_signal.
	_bt_rfcomm_mgr_allocate_session	This is function _bt_rfcomm_mgr_allocate_session.
	_bt_rfcomm_mgr_l2cap_listen_callback	This is function _bt_rfcomm_mgr_l2cap_listen_callback.
	_bt_rfcomm_mgr_notify_listeners	This is function _bt_rfcomm_mgr_notify_listeners.
	_bt_rfcomm_set_signal	This is function _bt_rfcomm_set_signal.
	_calc_fcs	This is function _calc_fcs.
	rfcomm_find_session	This is function rfcomm_find_session.
	_rfcomm_l2cap_state_changed_callback	From rfcomm.c
	bt_rfcomm_cancel_listen	This is function bt_rfcomm_cancel_listen.
	bt_rfcomm_get_frame_length	This is function bt_rfcomm_get_frame_length.
	rfcomm_cq_find_failed_pn	This is function rfcomm_cq_find_failed_pn.

SBC Decoder Data Types and Constants

	Name	Description
	sbc_struct	
	sbc_t	This is type sbc_t.
	__SBC_H	This is macro __SBC_H.
	INP_BUF_SIZE	Bytes
	MAX_SBC_DEC_STATE_SIZE	Bytes
	OUT_BUF_SIZE	Bytes
	SBC_AM_LOUDNESS	Allocation method
	SBC_AM_SNR	This is macro SBC_AM_SNR.
	SBC_BE	This is macro SBC_BE.
	SBC_BLK_12	This is macro SBC_BLK_12.
	SBC_BLK_16	This is macro SBC_BLK_16.
	SBC_BLK_4	Blocks
	SBC_BLK_8	This is macro SBC_BLK_8.
	SBC_FREQ_16000	Sampling frequency
	SBC_FREQ_32000	This is macro SBC_FREQ_32000.
	SBC_FREQ_44100	This is macro SBC_FREQ_44100.
	SBC_FREQ_48000	This is macro SBC_FREQ_48000.
	SBC_LE	Data endianness
	SBC_MODE_DUAL_CHANNEL	This is macro SBC_MODE_DUAL_CHANNEL.
	SBC_MODE_JOINT_STEREO	This is macro SBC_MODE_JOINT_STEREO.
	SBC_MODE_MONO	Channel mode
	SBC_MODE_STEREO	This is macro SBC_MODE_STEREO.
	SBC_SB_4	Sub-bands
	SBC_SB_8	This is macro SBC_SB_8.
	ssize_t	

SBC Decoder Functions

	Name	Description
	sbc_decode	Decodes the encoded SBC frame.
	sbc_get_codesize	Gets the input block size.
	sbc_get_frame_duration	Gets the time one input/output block takes to play in microseconds.

	sbc_get_frame_length	Gets the output block size.
	sbc_get_state_size	Gets the SBC Decoder's state data structure.
	sbc_init	Initializes the SBC Decoder.

SDP Data Types and Constants

	Name	Description
	BEGIN_DE_SEQUENCE	• brief Begin a data element sequence • ingroup sdp * • details BEGIN_DE_SEQUENCE and END_DE_SEQUENCE are used to define a data element sequence which is an array of sdp_data_element structures. • The array is used a search pattern in bt_sdp_request_service_search() and bt_sdp_request_service_attribute(). • For example, to find a AVRCP Target the following code can be used: code const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000, SDP_CLSID_AV_REMOTE_CONTROL }; const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000, SDP_CLSID_AV_REMOTE_CONTROL_TARGET }; BEGIN_DE_SEQUENCE(avrcp_target_service_search, 2) DE_UUID128(AVRCP_AV_REMOTE_CONTROL_CLSID) DE_UUID128(AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID) END_DE_SEQUENCE(avrcp_target_service_search) . . . void findAvrcpTarget(void) { INIT_DE_SEQUENCE(avrcp_target_service_search); bt_sdp_request_service_search(channel, &seq_avrcp_target_service_search, &callback, NULL); } endcode * • param... more
	DE_BOOL	brief Declare a boolean data element ingroup sdp details This macro adds a boolean data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_INT	brief Declare a 1-byte signed integer data element ingroup sdp details This macro adds a 1-byte signed integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_STRING	brief Declare a text string data element ingroup sdp details This macro adds a text string data element to a data element sequence. The length of the generated data element will be the actual length of the string. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_STRING2	brief Declare a text string data element ingroup sdp details This macro adds a text string data element to a data element sequence. The length of the generated data element will be the value specified by the "len" parameter even if the actual length of the string is not equal to the "len" value. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value. param len The length of the data element value.
	DE_UINT	brief Declare a 1-byte unsigned integer data element ingroup sdp details This macro adds a 1-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.

	DE_UINT16	brief Declare a 2-byte unsigned integer data element ingroup sdp details This macro adds a 2-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_URL	brief Declare a URL data element ingroup sdp details This macro adds a URL data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value which is a pointer to a string.
	DE_UUID128	brief Declare a 128-bit UUID data element ingroup sdp details This macro adds a 128-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value. The value must be a name of a variable of type bt_uuid.
	DE_UUID16	brief Declare a 16-bit UUID data element ingroup sdp details This macro adds a 16-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_UUID32	brief Declare a 32-bit UUID data element ingroup sdp details This macro adds a 32-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	END_DE_SEQUENCE	brief End a data element sequence ingroup sdp details BEGIN_DE_SEQUENCE and END_DE_SEQUENCE are used to define a data element sequence which is an array of bt_sdp_data_element structures. param id The data element sequence identifier.
	INIT_DE_SEQUENCE	brief Initialize a data element sequence ingroup sdp details This macro calls a function defined in BEGIN_DE_SEQUENCE which initializes the data element sequence. param id The data element sequence identifier.
	SDP_ATTRID_AdditionalProtocolDescriptorLists	This is macro SDP_ATTRID_AdditionalProtocolDescriptorLists.
	SDP_ATTRID_BluetoothProfileDescriptorList	This is macro SDP_ATTRID_BluetoothProfileDescriptorList.
	SDP_ATTRID_BrowseGroupList	This is macro SDP_ATTRID_BrowseGroupList.
	SDP_ATTRID_ClientExecutableURL	This is macro SDP_ATTRID_ClientExecutableURL.
	SDP_ATTRID_DIPrimaryRecord	This is macro SDP_ATTRID_DIPrimaryRecord.
	SDP_ATTRID_DIProductId	This is macro SDP_ATTRID_DIProductId.
	SDP_ATTRID_DISpecificationId	This is macro SDP_ATTRID_DISpecificationId.
	SDP_ATTRID_DIVendorId	This is macro SDP_ATTRID_DIVendorId.
	SDP_ATTRID_DIVendorIdSource	This is macro SDP_ATTRID_DIVendorIdSource.
	SDP_ATTRID_DIVersion	This is macro SDP_ATTRID_DIVersion.
	SDP_ATTRID_DocumentationURL	This is macro SDP_ATTRID_DocumentationURL.
	SDP_ATTRID_GAPRemoteAudioVolumeControl	This is macro SDP_ATTRID_GAPRemoteAudioVolumeControl.
	SDP_ATTRID_GroupID	This is macro SDP_ATTRID_GroupID.
	SDP_ATTRID_HDPDataExchangeSpecification	This is macro SDP_ATTRID_HDPDataExchangeSpecification.
	SDP_ATTRID_HDPMCAPSupportedProcedures	This is macro SDP_ATTRID_HDPMCAPSupportedProcedures.
	SDP_ATTRID_HDPSuportedFeatures	This is macro SDP_ATTRID_HDPSuportedFeatures.
	SDP_ATTRID_HFPAGNetwork	This is macro SDP_ATTRID_HFPAGNetwork.
	SDP_ATTRID_HFPSupportedFeatures	This is macro SDP_ATTRID_HFPSupportedFeatures.
	SDP_ATTRID_HIDBatteryPower	This is macro SDP_ATTRID_HIDBatteryPower.
	SDP_ATTRID_HIDBootDevice	This is macro SDP_ATTRID_HIDBootDevice.
	SDP_ATTRID_HIDCountryCode	This is macro SDP_ATTRID_HIDCountryCode.
	SDP_ATTRID_HIDDescriptorList	This is macro SDP_ATTRID_HIDDescriptorList.
	SDP_ATTRID_HIDDeviceReleaseNumber	This is macro SDP_ATTRID_HIDDeviceReleaseNumber.
	SDP_ATTRID_HIDDeviceSubclass	This is macro SDP_ATTRID_HIDDeviceSubclass.
	SDP_ATTRID_HIDLANGIDBaseList	This is macro SDP_ATTRID_HIDLANGIDBaseList.

	SDP_ATTRID_HIDNormallyConnectable	This is macro SDP_ATTRID_HIDNormallyConnectable.
	SDP_ATTRID_HIDParserVersion	This is macro SDP_ATTRID_HIDParserVersion.
	SDP_ATTRID_HIDProfileVersion	This is macro SDP_ATTRID_HIDProfileVersion.
	SDP_ATTRID_HIDReconnectInitiate	This is macro SDP_ATTRID_HIDReconnectInitiate.
	SDP_ATTRID_HIDRemoteWake	This is macro SDP_ATTRID_HIDRemoteWake.
	SDP_ATTRID_HIDSDPDisable	This is macro SDP_ATTRID_HIDSDPDisable.
	SDP_ATTRID_HIDSupervisionTimeout	This is macro SDP_ATTRID_HIDSupervisionTimeout.
	SDP_ATTRID_HIDVirtualCable	This is macro SDP_ATTRID_HIDVirtualCable.
	SDP_ATTRID_IconURL	This is macro SDP_ATTRID_IconURL.
	SDP_ATTRID_INVALID	This is macro SDP_ATTRID_INVALID.
	SDP_ATTRID_LanguageBaseAttributeIDList	This is macro SDP_ATTRID_LanguageBaseAttributeIDList.
	SDP_ATTRID_OFFSET_ProviderName	This is macro SDP_ATTRID_OFFSET_ProviderName.
	SDP_ATTRID_OFFSET_ServiceDescription	This is macro SDP_ATTRID_OFFSET_ServiceDescription.
	SDP_ATTRID_OFFSET_ServiceName	This is macro SDP_ATTRID_OFFSET_ServiceName.
	SDP_ATTRID_PrimaryLanguageBaseId	This is macro SDP_ATTRID_PrimaryLanguageBaseId.
	SDP_ATTRID_ProtocolDescriptorList	This is macro SDP_ATTRID_ProtocolDescriptorList.
	SDP_ATTRID_ServiceAvailability	This is macro SDP_ATTRID_ServiceAvailability.
	SDP_ATTRID_ServiceClassIDList	This is macro SDP_ATTRID_ServiceClassIDList.
	SDP_ATTRID_ServiceDatabaseState	This is macro SDP_ATTRID_ServiceDatabaseState.
	SDP_ATTRID_ServiceID	This is macro SDP_ATTRID_ServiceID.
	SDP_ATTRID_ServiceInfoTimeToLive	This is macro SDP_ATTRID_ServiceInfoTimeToLive.
	SDP_ATTRID_ServiceRecordHandle	This is macro SDP_ATTRID_ServiceRecordHandle.
	SDP_ATTRID_ServiceRecordState	This is macro SDP_ATTRID_ServiceRecordState.
	SDP_ATTRID_SupportedFeatures	This is macro SDP_ATTRID_SupportedFeatures.
	SDP_ATTRID_VersionNumberList	This is macro SDP_ATTRID_VersionNumberList.
	SDP_CLIENT_EVT_CONNECTED	This is macro SDP_CLIENT_EVT_CONNECTED.
	SDP_CLIENT_EVT_DISCONNECTED	This is macro SDP_CLIENT_EVT_DISCONNECTED.
	SDP_CLIENT_EVT_NULL	This is macro SDP_CLIENT_EVT_NULL.
	SDP_CLSID_ADVANCED_AUDIO_DISTRIBUTION	This is macro SDP_CLSID_ADVANCED_AUDIO_DISTRIBUTION.
	SDP_CLSID_AUDIO_SINK	This is macro SDP_CLSID_AUDIO_SINK.
	SDP_CLSID_AUDIO_SOURCE	This is macro SDP_CLSID_AUDIO_SOURCE.
	SDP_CLSID_AV_REMOTE_CONTROL	This is macro SDP_CLSID_AV_REMOTE_CONTROL.
	SDP_CLSID_AV_REMOTE_CONTROL_CONTROLLER	This is macro SDP_CLSID_AV_REMOTE_CONTROL_CONTROLLER.
	SDP_CLSID_AV_REMOTE_CONTROL_PROFILE_ID	This is macro SDP_CLSID_AV_REMOTE_CONTROL_PROFILE_ID.
	SDP_CLSID_AV_REMOTE_CONTROL_TARGET	This is macro SDP_CLSID_AV_REMOTE_CONTROL_TARGET.
	SDP_CLSID_AVCTP	This is macro SDP_CLSID_AVCTP.
	SDP_CLSID_AVDTP	This is macro SDP_CLSID_AVDTP.
	SDP_CLSID_BrowseGroupDescriptorServiceClassID	This is macro SDP_CLSID_BrowseGroupDescriptorServiceClassID.
	SDP_CLSID_DialupNetworking	This is macro SDP_CLSID_DialupNetworking.
	SDP_CLSID_GENERIC_AUDIO	This is macro SDP_CLSID_GENERIC_AUDIO.
	SDP_CLSID_HDP	This is macro SDP_CLSID_HDP.
	SDP_CLSID_HDP_SINK	This is macro SDP_CLSID_HDP_SINK.
	SDP_CLSID_HDP_SOURCE	This is macro SDP_CLSID_HDP_SOURCE.
	SDP_CLSID_HFP	This is macro SDP_CLSID_HFP.
	SDP_CLSID_HFP_AG	This is macro SDP_CLSID_HFP_AG.
	SDP_CLSID_HID	This is macro SDP_CLSID_HID.
	SDP_CLSID_HIDProtocol	This is macro SDP_CLSID_HIDProtocol.
	SDP_CLSID_HSP	This is macro SDP_CLSID_HSP.
	SDP_CLSID_HSP_AG	This is macro SDP_CLSID_HSP_AG.
	SDP_CLSID_HSP_HS	This is macro SDP_CLSID_HSP_HS.
	SDP_CLSID_L2CAP	This is macro SDP_CLSID_L2CAP.
	SDP_CLSID_MCAP_CONTROL	This is macro SDP_CLSID_MCAP_CONTROL.
	SDP_CLSID_MCAP_DATA	This is macro SDP_CLSID_MCAP_DATA.
	SDP_CLSID_OBEXFileTransfer	This is macro SDP_CLSID_OBEXFileTransfer.
	SDP_CLSID_OBEXObjectPush	This is macro SDP_CLSID_OBEXObjectPush.

	SDP_CLSID_PBAP_PCE	This is macro SDP_CLSID_PBAP_PCE.
	SDP_CLSID_PBAP_PSE	This is macro SDP_CLSID_PBAP_PSE.
	SDP_CLSID_PNPInformation	This is macro SDP_CLSID_PNPInformation.
	SDP_CLSID_PublicBrowseGroup	This is macro SDP_CLSID_PublicBrowseGroup.
	SDP_CLSID_RFCOMM	This is macro SDP_CLSID_RFCOMM.
	SDP_CLSID_SerialPort	This is macro SDP_CLSID_SerialPort.
	SDP_CLSID_ServiceDiscoveryServerServiceClassID	defgroup sdp SDP This module describe functions and data structures used to start the SDP server and perform SDP queries.
	SDP_DATA_TYPE_ALTERNATIVE	This is macro SDP_DATA_TYPE_ALTERNATIVE.
	SDP_DATA_TYPE_BOOL	This is macro SDP_DATA_TYPE_BOOL.
	SDP_DATA_TYPE_INT	This is macro SDP_DATA_TYPE_INT.
	SDP_DATA_TYPE_INT128	This is macro SDP_DATA_TYPE_INT128.
	SDP_DATA_TYPE_INT16	This is macro SDP_DATA_TYPE_INT16.
	SDP_DATA_TYPE_INT32	This is macro SDP_DATA_TYPE_INT32.
	SDP_DATA_TYPE_INT64	This is macro SDP_DATA_TYPE_INT64.
	SDP_DATA_TYPE_INT8	This is macro SDP_DATA_TYPE_INT8.
	SDP_DATA_TYPE_NIL	This is macro SDP_DATA_TYPE_NIL.
	SDP_DATA_TYPE_SEQUENCE	This is macro SDP_DATA_TYPE_SEQUENCE.
	SDP_DATA_TYPE_STRING	This is macro SDP_DATA_TYPE_STRING.
	SDP_DATA_TYPE_UINT	This is macro SDP_DATA_TYPE_UINT.
	SDP_DATA_TYPE_UINT128	This is macro SDP_DATA_TYPE_UINT128.
	SDP_DATA_TYPE_UINT16	This is macro SDP_DATA_TYPE_UINT16.
	SDP_DATA_TYPE_UINT32	This is macro SDP_DATA_TYPE_UINT32.
	SDP_DATA_TYPE_UINT64	This is macro SDP_DATA_TYPE_UINT64.
	SDP_DATA_TYPE_UINT8	This is macro SDP_DATA_TYPE_UINT8.
	SDP_DATA_TYPE_URL	This is macro SDP_DATA_TYPE_URL.
	SDP_DATA_TYPE_UUID	This is macro SDP_DATA_TYPE_UUID.
	SDP_DATA_TYPE_UUID128	This is macro SDP_DATA_TYPE_UUID128.
	SDP_DATA_TYPE_UUID16	This is macro SDP_DATA_TYPE_UUID16.
	SDP_DATA_TYPE_UUID32	This is macro SDP_DATA_TYPE_UUID32.
	SDP_ERROR_INSUFFICIENT_RESOURCE	This is macro SDP_ERROR_INSUFFICIENT_RESOURCE.
	SDP_ERROR_INVALID_CONTINUATION_STATE	This is macro SDP_ERROR_INVALID_CONTINUATION_STATE.
	SDP_ERROR_INVALID_PDU_SIZE	This is macro SDP_ERROR_INVALID_PDU_SIZE.
	SDP_ERROR_INVALID_REQUEST_SYNTAX	This is macro SDP_ERROR_INVALID_REQUEST_SYNTAX.
	SDP_ERROR_INVALID_SDP_VERSION	This is macro SDP_ERROR_INVALID_SDP_VERSION.
	SDP_ERROR_INVALID_SR_HANDLE	This is macro SDP_ERROR_INVALID_SR_HANDLE.
	SDP_ERROR_RESERVED	This is macro SDP_ERROR_RESERVED.
	SDP_ErrorResponse	This is macro SDP_ErrorResponse.
	SDP_FTP_SERVICE_ID	This is macro SDP_FTP_SERVICE_ID.
	SDP_HID_SERVICE_ID	This is macro SDP_HID_SERVICE_ID.
	SDP_HSP_AG_SERVICE_ID	This is macro SDP_HSP_AG_SERVICE_ID.
	SDP_HSP_HS_SERVICE_ID	This is macro SDP_HSP_HS_SERVICE_ID.
	SDP_MAX_ATTRIBUTE_PATTERN_LEN	This is macro SDP_MAX_ATTRIBUTE_PATTERN_LEN.
	SDP_MAX_DATA_ELEMENT_LEN	This is macro SDP_MAX_DATA_ELEMENT_LEN.
	SDP_MAX_DATA_ELEMENTS	This is macro SDP_MAX_DATA_ELEMENTS.
	SDP_MAX_SEARCH_PATTERN_LEN	This is macro SDP_MAX_SEARCH_PATTERN_LEN.
	SDP_MAX_TRANSACTIONS	typedef struct _sdp_db { bt_int count; bt_sdp_service_record_p *records; } bt_sdp_db, *bt_sdp_db_p;
	SDP_PDU_HEADER_LEN	This is macro SDP_PDU_HEADER_LEN.
	SDP_RFCOMM_SERVICE_ID	This is macro SDP_RFCOMM_SERVICE_ID.
	SDP_ServiceAttributeRequest	This is macro SDP_ServiceAttributeRequest.
	SDP_ServiceAttributeResponse	This is macro SDP_ServiceAttributeResponse.
	SDP_ServiceSearchAttributeRequest	This is macro SDP_ServiceSearchAttributeRequest.
	SDP_ServiceSearchAttributeResponse	This is macro SDP_ServiceSearchAttributeResponse.

	SDP_ServiceSearchRequest	This is macro SDP_ServiceSearchRequest.
	SDP_ServiceSearchResponse	This is macro SDP_ServiceSearchResponse.
	SDP_SR_HANDLE_HDP_SINK	This is macro SDP_SR_HANDLE_HDP_SINK.
	SDP_SR_HANDLE_HDP_SOURCE	This is macro SDP_SR_HANDLE_HDP_SOURCE.
	SDP_SR_HANDLE_HFP_HF	This is macro SDP_SR_HANDLE_HFP_HF.
	SDP_SR_HANDLE_HID	This is macro SDP_SR_HANDLE_HID.
	SDP_SR_HANDLE_HID_KEYBOARD	This is macro SDP_SR_HANDLE_HID_KEYBOARD.
	SDP_SR_HANDLE_HSP_HS	This is macro SDP_SR_HANDLE_HSP_HS.
	SDP_SR_HANDLE_OBEXFileTransfer	This is macro SDP_SR_HANDLE_OBEXFileTransfer.
	SDP_SR_HANDLE_OBEXObjectPush	This is macro SDP_SR_HANDLE_OBEXObjectPush.
	SDP_SR_HANDLE_PNPINFORMATION	This is macro SDP_SR_HANDLE_PNPINFORMATION.
	SDP_SR_HANDLE_RFCOMM	This is macro SDP_SR_HANDLE_RFCOMM.
	SDP_SR_HANDLE_SERVER	This is macro SDP_SR_HANDLE_SERVER.
	SDP_SR_HANDLE_TEST	for simulating service search using continuation state
	bt_sdp_client_evt_connected_t	This is record bt_sdp_client_evt_connected_t.
	bt_sdp_client_evt_disconnected_t	This is record bt_sdp_client_evt_disconnected_t.
	_bt_sdp_found_attr_list_t	This is type bt_sdp_found_attr_list_t.
	_bt_sdp_packet_t	This is type bt_sdp_packet_t.
	_bt_sdp_sequence_t	This is type bt_sdp_sequence_t.
	_bt_sdp_serialization_state_t	This is type bt_sdp_serialization_state_t.
	_bt_sdp_server_attribute_t	This is type bt_sdp_server_attribute_t.
	_bt_sdp_server_data_element_t	Private types
	_bt_sdp_server_record_t	This is type bt_sdp_server_record_t.
	_bt_sdp_service_transaction_t	This is type bt_sdp_service_transaction_t.
	_bt_sdp_transaction_t	This is type bt_sdp_transaction_t.
	bt_sdp_client_callback_fp	This is type bt_sdp_client_callback_fp.
	bt_sdp_data_element_cp	This is type bt_sdp_data_element_cp.
	bt_sdp_data_element_p	This is type bt_sdp_data_element_p.
	bt_sdp_data_element_t	This is type bt_sdp_data_element_t.
	bt_sdp_found_attr_list_t	This is type bt_sdp_found_attr_list_t.
	bt_sdp_packet_t	This is type bt_sdp_packet_t.
	bt_sdp_read_de_callback_ftp	This is type bt_sdp_read_de_callback_ftp.
	bt_sdp_sequence_cp	This is type bt_sdp_sequence_cp.
	bt_sdp_sequence_p	This is type bt_sdp_sequence_p.
	bt_sdp_sequence_t	This is type bt_sdp_sequence_t.
	bt_sdp_serialization_state_p	This is type bt_sdp_serialization_state_p.
	bt_sdp_serialization_state_t	This is type bt_sdp_serialization_state_t.
	bt_sdp_server_attribute_t	This is type bt_sdp_server_attribute_t.
	bt_sdp_server_data_element_t	Private types
	bt_sdp_server_record_t	This is type bt_sdp_server_record_t.
	bt_sdp_service_attribute_callback_fp	This is type bt_sdp_service_attribute_callback_fp.
	bt_sdp_service_search_callback_fp	This is type bt_sdp_service_search_callback_fp.
	bt_sdp_service_transaction_p	This is type bt_sdp_service_transaction_p.
	bt_sdp_service_transaction_t	This is type bt_sdp_service_transaction_t.
	bt_sdp_transaction_t	This is type bt_sdp_transaction_t.
	bt_sr_handle_p	This is type bt_sr_handle_p.
	bt_sr_handle_t	This is type bt_sr_handle_t.
	SDP_CLIENT_EVT_CONNECTION_FAILED	This is macro SDP_CLIENT_EVT_CONNECTION_FAILED.
	SDP_ATTRID_HCRP_1284ID	This is macro SDP_ATTRID_HCRP_1284ID.
	SDP_ATTRID_HCRP_DeviceLocation	This is macro SDP_ATTRID_HCRP_DeviceLocation.
	SDP_ATTRID_HCRP_DeviceName	This is macro SDP_ATTRID_HCRP_DeviceName.
	SDP_ATTRID_HCRP_FriendlyName	This is macro SDP_ATTRID_HCRP_FriendlyName.
	SDP_CLSID_HARD_COPY_CABLE_REPLACEMENT	This is macro SDP_CLSID_HARD_COPY_CABLE_REPLACEMENT.
	SDP_CLSID_HARD_COPY_CONTROL_CHANNEL	This is macro SDP_CLSID_HARD_COPY_CONTROL_CHANNEL.
	SDP_CLSID_HARD_COPY_DATA_CHANNEL	This is macro SDP_CLSID_HARD_COPY_DATA_CHANNEL.

	SDP_CLSID_HARD_COPY_NOTIFICATION	This is macro SDP_CLSID_HARD_COPY_NOTIFICATION.
	SDP_CLSID_HCR_PRINT	This is macro SDP_CLSID_HCR_PRINT.
	SDP_CLSID_HCR_SCAN	This is macro SDP_CLSID_HCR_SCAN.

SDP Functions

	Name	Description
	bt_sdp_de_to_uuid	This is function bt_sdp_de_to_uuid.
	bt_sdp_packet_assembler	This is function bt_sdp_packet_assembler.
	bt_sdp_read_attribute	From sdp_db_utils.c
	bt_sdp_start	brief Start SDP server ingroup sdp details This function starts the SDP server. param l2cap_mgr The L2CAP manager on which the SDP server is to be started. param sdp_db A pointer to the SDP database define with BEGIN_SDP_DB and END_SDP_DB macros. return li c TRUE if the function succeeds. li c FALSE otherwise.
	sdp_compare_uuid_de	This is function sdp_compare_uuid_de.
	sdp_find_attributes	This is function sdp_find_attributes.
	sdp_find_service_records2	From sdp_utils.c
	_sdp_alloc_svc_tran_buffer	This is function _sdp_alloc_svc_tran_buffer.
	_sdp_alloc_tran_buffer	This is function _sdp_alloc_tran_buffer.
	_sdp_find_svc_transaction	This is function _sdp_find_svc_transaction.
	_sdp_find_transaction	This is function _sdp_find_transaction.
	_sdp_free_svc_tran_buffer	This is function _sdp_free_svc_tran_buffer.
	_sdp_free_tran_buffer	This is function _sdp_free_tran_buffer.
	_sdp_get_de_data_len	This is function _sdp_get_de_data_len.
	_sdp_get_de_hdr_len	This is function _sdp_get_de_hdr_len.
	_sdp_init_tran_buffers	From sdp_tran_buffer.c
	_sdp_read_de_header	This is function _sdp_read_de_header.
	_sdp_write_data_element	This is function _sdp_write_data_element.
	bt_sdp_request_service_attribute	brief Search attributes ingroup sdp details This function retrieves attribute values from a service record. param channel The L2CAP channel used to communicate to the remote SDP server. param sr The service record handle specifies the service record from which attribute values are to be retrieved. param pattern The attribute search pattern is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. The pattern buffer must be valid for the duration of the search operation, i.e. until c callback is called for the first time. To define a... more
	bt_sdp_request_service_search	brief Search service records ingroup sdp details This function locates service records on a remote SDP server that match the given service search pattern. param channel The L2CAP channel used to communicate to the remote SDP server. param pattern The service search pattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12. The pattern buffer must be valid for the duration of the search operation, i.e. until c callback is called. To define a data element sequence... more
	_bt_sdp_client_init	This is function _bt_sdp_client_init.

SPP Data Types and Constants

	Name	Description
	SPP_PORT_OPTION_ENCRYPTED	This is macro SPP_PORT_OPTION_ENCRYPTED.
	SPP_PORT_OPTION_MASTER	This is macro SPP_PORT_OPTION_MASTER.
	SPP_PORT_OPTION_SECURE	This is macro SPP_PORT_OPTION_SECURE.
	SPP_PORT_TYPE_INCOMING	This is macro SPP_PORT_TYPE_INCOMING.
	SPP_PORT_TYPE_OUTGOING	This is macro SPP_PORT_TYPE_OUTGOING.
	SPP_RS232_CTS	This is macro SPP_RS232_CTS.
	SPP_RS232_DCD	This is macro SPP_RS232_DCD.

	SPP_RS232_DSR	• @defgroup spp Serial Port Profile (SPP) The DotStack SPP API is a simple API for communicating over a Bluetooth link using the Bluetooth Serial Port Profile. Here are the steps for using this API: • Call the bt_spp_init() function. • Allocate a serial port structure with bt_spp_allocate(). One of the parameters to this function is a pointer to a callback function. That callback function will be called by the stack whenever the state of the serial port changes. • To connect to a remote device call bt_spp_connect(). The stack will notify when the connection is established by calling the state... more
	SPP_RS232_DTR	This is macro SPP_RS232_DTR.
	SPP_RS232_RI	This is macro SPP_RS232_RI.
	SPP_RS232_RTS	This is macro SPP_RS232_RTS.
	bt_spp_find_server_callback_fp	This is type bt_spp_find_server_callback_fp.
	bt_spp_port_event_e	• Serial port event. Values of this enumeration represent serial port events. Events are reported to the application through a callback function. The callback function is specified when the port is allocated using bt_spp_allocate().
	bt_spp_port_state_e	• Serial port state. Values of this enumeration represent states of the serial port.
	bt_spp_port_t	This is type bt_spp_port_t.
	bt_spp_receive_callback_fp	• Serial port receive callback. This callback function is called when a receive operation initiated by bt_spp_receive() completes. @param port Serial port on which the receive operation completed. @param bytes_received Number of received bytes. @param param Callback parameter that was specified when bt_spp_allocate() was called.
	bt_spp_send_callback_fp	• Serial port send callback. This callback function is called when a send operation initiated by bt_spp_send() completes. @param port Serial port on which the send operation completed. @param bytes_sent Number of bytes sent. This parameter is just a convenience as it always specifies the same number of bytes that was passed to bt_spp_send(); @param status Completion status. It is one of the values defined in the ::bt_spp_send_status_e enumeration. @param param Callback parameter that was specified when bt_spp_allocate() was called.
	bt_spp_send_status_e	Send operation status.
	bt_spp_state_callback_fp	• Serial port state callback. This callback function is called whenever the state of a serial port is changed. @param port Serial port which state has changed. @param evt Event describing the nature of state change. It is one of the values defined in the ::bt_spp_port_event_e enumeration. @param param Callback parameter that was specified when bt_spp_allocate() was called.

SPP Functions

	Name	Description
	bt_spp_allocate	• Allocate a serial port. The returned serial port is initially in the ::SPP_PORT_STATE_DISCONNECTED state. To establish a connection with a remote device, call bt_spp_connect(). To listen for incoming connections from other devices, call bt_spp_listen(). The p callback parameter must specify a callback function that will be used to notify about serial port events and state changes. When the port is not needed any more it must be deallocated by bt_spp_deallocate(). The maximum number of serial ports that can be allocated simultaneously is specified by the ::SPP_MAX_PORTS configuration parameter. @param l2cap_mgr L2CAP manager. @param callback Pointer to... more

	bt_spp_cancel_receive	• Cancel receive data. If a receive operation is currently in progress this function will cancel it. After calling this function the receive callback specified in bt_spp_receive() will not be called. If there is no receive operation in progress calling this function has no effect. @param port Serial port.
	bt_spp_cancel_send	• Cancel send data. If a send operation is currently in progress this function will try to cancel it. When the operation is canceled the send callback function will be called with the ::SPP_SEND_STATUS_CANCELED status. If this function is called but the active send operation completes successfuly before the stack can actually cancel it the call back function will still be called with the ::SPP_SEND_STATUS_CANCELED status. If there is no send operation in progess calling this function has no effect. @param port Serial port.
	bt_spp_clr_port_options	This is function bt_spp_clr_port_options.
	bt_spp_connect	• Connect to a remote device. This function initiates a connection to a remote device. When the connection is successfully established the port's callback is called with the ::SPP_PORT_EVENT_CONNECTED event. If connection fails the callback is called with the ::SPP_PORT_EVENT_CONNECTION_FAILED event. The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail. @param port Serial port. @param remote_addr Bluetooth address of the remote device. @param channel RFCOMM server channel on which the connection is to be established. @return c TRUE if successful, c FALSE otherwise.
	bt_spp_deallocate	• Deallocate serial port. This function deallocates the specified port structure and other resources associated with it. The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail. If the function completes successfully the application must not try to access any fields in the structure and must not use it with any other SPP functions. Also, it becomes available for subsequent allocation by bt_spp_port_allocate(). @param port Serial port structure to deallocate. @return c TRUE if successful, c FALSE otherwise.
	bt_spp_disconnect	• Disconnect from the remote device. This function initiates the disconnection process. When it is complete the the port's callback is called with the SPP_PORT_EVENT_DISCONNECTED event. If the port is already in the disconnected state the function does nothing and the callback is not called. @param port Serial port.
	bt_spp_find_server	This is function bt_spp_find_server.
	bt_spp_get_frame_length	• Get frame length. This function returns the RFCOMM frame length used by the RFCOMM protocol. The frame length depends on configuration of DotStack and configuration of the Bluetooth stack running on the remote device. In order to achieve maximum throughput over the serial port connection the application should send and receive data in chunks that are multiple of this frame length. @return RFCOMM frame length in bytes.
	bt_spp_get_hci_connection	• Get SPP port's ACL connection. This function returns a pinter to the structure that describes the ACL connection this port is on. @return Pointer to ACL connection description if the port is connected, NULL otherwise.
	bt_spp_get_local_modem_status	• Get local device's TS 07.10 control signals. This function returns current state of the local device's TS 07.10 controls signals. The signals are defined as a mask of the following constants: SPP_RS232_DSR SPP_RS232_RTS SPP_RS232_RI SPP_RS232_DCD @param port Serial port. @return local device's TS 07.10 control signals.

	bt_spp_get_remote_address	brief Get the address of the remote device this device is connected to. ingroup spp param port Serial port. return li c A pointer to bt_bdaddr structure that contains the address of the remote device.
	bt_spp_get_remote_modem_status	• Get remote device's TS 07.10 control signals. This function returns current state of the remote device's V.24 controls signals. The signals are defined as a mask of the following constants: SPP_RS232_DTR SPP_RS232_CTS SPP_RS232_RI SPP_RS232_DCD @param port Serial port. @return remote device's TS 07.10 control signals.
	bt_spp_init	• Initialize the SPP module. This function initializes all internal variables of the SPP module. It must be called prior to using any other functions in this module.
	bt_spp_listen	• Listen for incoming connections. This function registers the port to accept incoming connections from remote devices on a particular RFCOMM server channel. The specified server channel should be listed in the SDP database. Otherwise, remote devices will not be able to find out which server channel to use. When a remote device successfully establishes a connection on the specified port the port's callback is called with the ::SPP_PORT_EVENT_CONNECTED event. The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail. @param port Serial port. @param channel The RFCOMM server channel on which to listen for connections. @return c TRUE... more
	bt_spp_receive	• Receive data. This function receives data from the serial port connection. The caller must provide a buffer and a callback function. Whenever the port receives data they are copied to the provided buffer and the callback function is called. The callback function is passed the length of received data and the same callback parameter that was specified when the port was allocated with bt_spp_allocate(). This function does not wait until the buffer is filled out completely. Any amount of received data will complete the operation. The port must be in ::SPP_PORT_STATE_CONNECTED state. Otherwise, the function will fail. Also, the... more
	bt_spp_send	• Send data. This function starts sending data over the serial port connection. Along with the data the caller must provide a callback function that is called when all data has been sent. Also, during execution of this operation the port's state callback function is called periodically with the ::SPP_PORT_EVENT_SEND_PROGRESS event. The port must be in ::SPP_PORT_STATE_CONNECTED state. Otherwise, the function will fail. Also, the function will fail if a previously started send operation is still in progress. The callback function is passed the same callback parameter that was specified when the port was allocated with bt_spp_allocate(). @param port Serial... more
	bt_spp_set_dtr	• Set local device's RS-232 DTR signal. Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM_MODEM_STATUS_RTC signal. If there are resources to send a command to the remote device this command will return TRUE. If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported. If the status cannot be changed because there is no resources to send a... more

	bt_spp_set_local_modem_status	• Set local device's TS 07.10 control signals. Changes the state of local device's TS 07.10 control signals and notifies the remote device of the change. If there are resources to send a command to the remote device this command will return TRUE. If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported. If the status cannot be changed because there is no resources to send a command to the remote device this function... more
	bt_spp_set_port_options	This is function bt_spp_set_port_options.
	bt_spp_set_rts	• Set local device's RS-232 RTS signal. Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM_MODEM_STATUS_RTR signal. If there are resources to send a command to the remote device this command will return TRUE. If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported. If the status cannot be changed because there is no resources to send a... more
	_bt_spp_find_port	This is function _bt_spp_find_port.
	_bt_spp_handle_rx	This is function _bt_spp_handle_rx.
	_bt_spp_handle_tx	This is function _bt_spp_handle_tx.
	_bt_spp_rfcomm_read_data_callback	This is function _bt_spp_rfcomm_read_data_callback.
	_bt_spp_rfcomm_send_data_callback	This is function _bt_spp_rfcomm_send_data_callback.

SSP Data Types and Constants

	Name	Description
	OOB_DATA_HASH_LENGTH	Remote OOB Data Request Event
	OOB_DATA_RANDOMIZER_LENGTH	This is macro OOB_DATA_RANDOMIZER_LENGTH.
	SSP_MAX_MANAGERS	This is macro SSP_MAX_MANAGERS.
	_bt_ssp_io_capability	This is type bt_ssp_io_capability.
	_bt_ssp_keypress_notification	Keypress Notification Event
	_bt_ssp_oob_data	This is type bt_ssp_oob_data.
	_bt_spp_port_event_e	• Serial port event. Values of this enumeration represent serial port events. Events are reported to the application through a callback function. The callback function is specified when the port is allocated using bt_spp_allocate().
	_bt_ssp_simple_pairing_complete	Simple Pairing Complete Event
	_bt_spp_port_state_e	• Serial port state. Values of this enumeration represent states of the serial port.
	_bt_ssp_user_confirmation_request	User Confirmation Request Event
	_bt_spp_port_t	• Serial port structure. This structure represents a Bluetooth serial port. Application code may only use those fields that are documented. The rest of the fields are private to the SPP implementation.
	_bt_ssp_user_passkey_notification	User Passkey Notification Event
	_bt_spp_send_status_e	Send operation status.
	_bt_ssp_user_passkey_request	User Passkey Request Event
	_SSP_AUTHENTICATION_REQUIREMENTS	This is type SSP_AUTHENTICATION_REQUIREMENTS.
	_SSP_IO_CAPABILITY	This is type SSP_IO_CAPABILITY.
	_SSP_KEYPRESS_NOTIFICATION_TYPE	This is type SSP_KEYPRESS_NOTIFICATION_TYPE.
	_SSP_MODE	This is type SSP_MODE.
	_SSP_OOB_DATA_PRESENT	This is type SSP_OOB_DATA_PRESENT.

	bt_spp_read_local_oob_data_callback_fp	This is type bt_spp_read_local_oob_data_callback_fp.
	bt_ssp_io_capability	This is type bt_ssp_io_capability.
	bt_ssp_keypress_notification	Keypress Notification Event
	bt_ssp_oob_data	This is type bt_ssp_oob_data.
	bt_ssp_simple_pairing_complete	Simple Pairing Complete Event
	bt_ssp_user_confirmation_request	User Confirmation Request Event
	bt_ssp_user_passkey_notification	User Passkey Notification Event
	bt_ssp_user_passkey_request	User Passkey Request Event
	SSP_AUTHENTICATION_REQUIREMENTS	This is type SSP_AUTHENTICATION_REQUIREMENTS.
	SSP_IO_CAPABILITY	This is type SSP_IO_CAPABILITY.
	SSP_KEYPRESS_NOTIFICATION_TYPE	This is type SSP_KEYPRESS_NOTIFICATION_TYPE.
	SSP_MODE	This is type SSP_MODE.
	SSP_OOB_DATA_PRESENT	This is type SSP_OOB_DATA_PRESENT.
	_SSP_EVENT	This is type SSP_EVENT.
	SSP_EVENT	This is type SSP_EVENT.

SSP Functions

	Name	Description
	bt_ssp_evt_handler	This is function bt_ssp_evt_handler.
	bt_ssp_init	This is function bt_ssp_init.
	bt_ssp_read_local_oob_data	This is function bt_ssp_read_local_oob_data.
	bt_ssp_send_keypress_notification	This is function bt_ssp_send_keypress_notification.
	bt_ssp_send_user_confirmation	This is function bt_ssp_send_user_confirmation.
	bt_ssp_send_user_passkey	This is function bt_ssp_send_user_passkey.
	bt_ssp_set_io_capabilities	This is function bt_ssp_set_io_capabilities.
	bt_ssp_set_mode	This is function bt_ssp_set_mode.
	bt_ssp_set_oob_data	This is function bt_ssp_set_oob_data.
	ssp_evt_io_capability_request	This is function ssp_evt_io_capability_request.
	ssp_evt_io_capability_response	This is function ssp_evt_io_capability_response.
	ssp_evt_keypress_notification	This is function ssp_evt_keypress_notification.
	ssp_evt_oob_data_request	This is function ssp_evt_oob_data_request.
	ssp_evt_ssp_complete	This is function ssp_evt_ssp_complete.
	ssp_evt_user_confirmation_request	This is function ssp_evt_user_confirmation_request.
	ssp_evt_user_passkey_notification	This is function ssp_evt_user_passkey_notification.
	ssp_evt_user_passkey_request	This is function ssp_evt_user_passkey_request.
	_bt_spp_client_init	This is function _bt_spp_client_init.
	bt_spp_cancel_listen	• Stop listening for incoming connections. This function stops the port to accept incoming connections on a given server channel. @param port Serial port. @param channel The RFCOMM server channel to accept connections on. @return c TRUE if successful, c FALSE otherwise.
	bt_spp_find_server_ex	This is function bt_spp_find_server_ex.

Utility Data Types and Constants

	Name	Description
	AT_EVT_CMD_CODE	This is macro AT_EVT_CMD_CODE.
	AT_EVT_CMD_COMPLETED	This is macro AT_EVT_CMD_COMPLETED.
	AT_EVT_CMD_PARAM	This is macro AT_EVT_CMD_PARAM.
	AT_EVT_CMD_READ_CODE	This is macro AT_EVT_CMD_READ_CODE.
	AT_EVT_ERROR	This is macro AT_EVT_ERROR.
	AT_EVT_OK	This is macro AT_EVT_OK.
	AT_EVT_RING	This is macro AT_EVT_RING.
	ATCMD_BUFFER_LEN	This is macro ATCMD_BUFFER_LEN.
	bt_max	This is macro bt_max.
	bt_min	This is macro bt_min.

	BUFFER_HDR_LEN	This is macro BUFFER_HDR_LEN.
	BUFFER_STATE_FREE	for offsetof
	BUFFER_STATE_USED	This is macro BUFFER_STATE_USED.
	max	This is macro max.
	min	This is macro min.
	VCARD_CALL_TYPE_DIALED	This is macro VCARD_CALL_TYPE_DIALED.
	VCARD_CALL_TYPE_MISSED	This is macro VCARD_CALL_TYPE_MISSED.
	VCARD_CALL_TYPE_RECEIVED	This is macro VCARD_CALL_TYPE_RECEIVED.
	VCARD_EVT_PROPERTY_PARAM	This is macro VCARD_EVT_PROPERTY_PARAM.
	VCARD_EVT_PROPERTY_STARTED	This is macro VCARD_EVT_PROPERTY_STARTED.
	VCARD_EVT_PROPERTY_VALUE	This is macro VCARD_EVT_PROPERTY_VALUE.
	VCARD_EVT_VCARD_ENDED	This is macro VCARD_EVT_VCARD_ENDED.
	VCARD_EVT_VCARD_STARTED	This is macro VCARD_EVT_VCARD_STARTED.
	VCARD_PARAM_CALL_TYPE	This is macro VCARD_PARAM_CALL_TYPE.
	VCARD_PARAM_CELL	This is macro VCARD_PARAM_CELL.
	VCARD_PARAM_DIALED	This is macro VCARD_PARAM_DIALED.
	VCARD_PARAM_ENCODING	This is macro VCARD_PARAM_ENCODING.
	VCARD_PARAM_HOME	This is macro VCARD_PARAM_HOME.
	VCARD_PARAM_LANGUAGE	This is macro VCARD_PARAM_LANGUAGE.
	VCARD_PARAM_MISSED	This is macro VCARD_PARAM_MISSED.
	VCARD_PARAM_RECEIVED	This is macro VCARD_PARAM_RECEIVED.
	VCARD_PARAM_TYPE	This is macro VCARD_PARAM_TYPE.
	VCARD_PARAM_VALUE	This is macro VCARD_PARAM_VALUE.
	VCARD_PARAM_WORK	This is macro VCARD_PARAM_WORK.
	VCARD_PARSER_STATE_READ_PARAM_NAME	This is macro VCARD_PARSER_STATE_READ_PARAM_NAME.
	VCARD_PARSER_STATE_READ_PARAM_VALUE	This is macro VCARD_PARSER_STATE_READ_PARAM_VALUE.
	VCARD_PARSER_STATE_READ_TYPE_NAME	This is macro VCARD_PARSER_STATE_READ_TYPE_NAME.
	VCARD_PARSER_STATE_READ_TYPE_VALUE	This is macro VCARD_PARSER_STATE_READ_TYPE_VALUE.
	VCARD_PARSER_STATE_SKIP_PARAM	This is macro VCARD_PARSER_STATE_SKIP_PARAM.
	VCARD_PARSER_STATE_SKIP_TYPE	This is macro VCARD_PARSER_STATE_SKIP_TYPE.
	VCARD_PROPERTY_ADR	Delivery Address
	VCARD_PROPERTY_AGENT	vCard of Person Representing
	VCARD_PROPERTY_BDAY	Birthday
	VCARD_PROPERTY_BEGIN	This is macro VCARD_PROPERTY_BEGIN.
	VCARD_PROPERTY_CATEGORIES	Categories
	VCARD_PROPERTY_CLASS	Class information
	VCARD_PROPERTY_EMAIL	Electronic Mail Address
	VCARD_PROPERTY_END	This is macro VCARD_PROPERTY_END.
	VCARD_PROPERTY_FN	Formatted Name
	VCARD_PROPERTY_GEO	Geographic Position
	VCARD_PROPERTY_KEY	Public Encryption Key
	VCARD_PROPERTY_LABEL	Delivery
	VCARD_PROPERTY_LOGO	Organization Logo
	VCARD_PROPERTY_MAILER	Electronic Mail
	VCARD_PROPERTY_N	Structured Presentation of Name
	VCARD_PROPERTY_NICKNAME	Nickname
	VCARD_PROPERTY_NOTE	Comments
	VCARD_PROPERTY_ORG	Name of Organization
	VCARD_PROPERTY_PHOTO	Associated Image or Photo
	VCARD_PROPERTY_PROID	Product ID
	VCARD_PROPERTY_REV	Revision
	VCARD_PROPERTY_ROLE	Role within the Organization
	VCARD_PROPERTY_SORT_STRING	String used for sorting operations
	VCARD_PROPERTY_SOUND	Pronunciation of Name
	VCARD_PROPERTY_TEL	Telephone Number

	VCARD_PROPERTY_TITLE	Job
	VCARD_PROPERTY_TZ	Time Zone
	VCARD_PROPERTY_UID	Unique ID
	VCARD_PROPERTY_URL	Uniform Resource Locator
	VCARD_PROPERTY_VERSION	vCard Version
	VCARD_PROPERTY_X_IRMC_CALL_DATETIME	Time stamp
	XML_EVT_ATTR_NAME	This is macro XML_EVT_ATTR_NAME.
	XML_EVT_ATTR_VALUE	This is macro XML_EVT_ATTR_VALUE.
	XML_EVT_TAG_ENDED	This is macro XML_EVT_TAG_ENDED.
	XML_EVT_TAG_STARTED	This is macro XML_EVT_TAG_STARTED.
	XML_PARSER_STATE_READ_ATTR_NAME	This is macro XML_PARSER_STATE_READ_ATTR_NAME.
	XML_PARSER_STATE_READ_ATTR_VALUE	This is macro XML_PARSER_STATE_READ_ATTR_VALUE.
	XML_PARSER_STATE_READ_TAG_NAME	This is macro XML_PARSER_STATE_READ_TAG_NAME.
	XML_PARSER_STATE_READ_TAG_VALUE	This is macro XML_PARSER_STATE_READ_TAG_VALUE.
	XML_PARSER_STATE_SKIP_ATTR	This is macro XML_PARSER_STATE_SKIP_ATTR.
	XML_PARSER_STATE_SKIP_COMMENTS	This is macro XML_PARSER_STATE_SKIP_COMMENTS.
	XML_PARSER_STATE_SKIP_PROC_INST	This is macro XML_PARSER_STATE_SKIP_PROC_INST.
	XML_PARSER_STATE_SKIP_TAG	This is macro XML_PARSER_STATE_SKIP_TAG.
	XML_PARSER_STATE_WAIT_ATTR_NAME	This is macro XML_PARSER_STATE_WAIT_ATTR_NAME.
	XML_PARSER_STATE_WAIT_ATTR_VALUE	This is macro XML_PARSER_STATE_WAIT_ATTR_VALUE.
	XML_PARSER_STATE_WAIT_TAG_CLOSE	This is macro XML_PARSER_STATE_WAIT_TAG_CLOSE.
	XML_PARSER_STATE_WAIT_TAG_NAME	This is macro XML_PARSER_STATE_WAIT_TAG_NAME.
	_readbn	This is macro _readbn.
	_readui	This is macro _readui.
	_readuin	This is macro _readuin.
	_readul	This is macro _readul.
	_readuln	This is macro _readuln.
	_writebn	This is macro _writebn.
	_bt_at_parser_t	This is type bt_at_parser_t.
	_bt_buffer_header_t	This is type bt_buffer_header_t.
	_bt_vcard_evt_prop_param_t	This is type bt_vcard_evt_prop_param_t.
	_bt_vcard_evt_prop_t	This is type bt_vcard_evt_prop_t.
	_bt_vcard_parser_t	This is type bt_vcard_parser_t.
	_bt_xml_evt_attribute_t	This is type bt_xml_evt_attribute_t.
	_bt_xml_evt_attribute_value_t	This is type bt_xml_evt_attribute_value_t.
	_bt_xml_evt_tag_started_t	This is type bt_xml_evt_tag_started_t.
	_bt_xml_parser_t	This is type bt_xml_parser_t.
	_bt_buffer_mgr_t	This is type bt_buffer_mgr_t.
	_bt_packet_t	This is type bt_packet_t.
	_bt_queue_element_t	This is type bt_queue_element_t.
	_cmd_disconnection_res	This is type pcmd_disconnection_res.
	bt_at_parser_callback_pf	This is type bt_at_parser_callback_pf.
	bt_at_parser_t	This is type bt_at_parser_t.
	bt_bt_buffer_header_p	This is type bt_bt_buffer_header_p.
	bt_buffer_header_t	This is type bt_buffer_header_t.
	bt_buffer_mgr_p	This is type bt_buffer_mgr_p.
	bt_buffer_mgr_t	This is type bt_buffer_mgr_t.
	bt_packet_assembler_fp	This is type bt_packet_assembler_fp.
	bt_packet_t	This is type bt_packet_t.
	bt_queue_element_t	This is type bt_queue_element_t.
	bt_vcard_evt_prop_param_t	This is type bt_vcard_evt_prop_param_t.
	bt_vcard_evt_prop_t	This is type bt_vcard_evt_prop_t.
	bt_vcard_parser_callback_fp	This is type bt_vcard_parser_callback_fp.
	bt_vcard_parser_t	This is type bt_vcard_parser_t.
	bt_xml_evt_attribute_t	This is type bt_xml_evt_attribute_t.

	bt_xml_evt_attribute_value_t	This is type bt_xml_evt_attribute_value_t.
	bt_xml_evt_tag_started_t	This is type bt_xml_evt_tag_started_t.
	bt_xml_parser_callback_pf	This is type bt_xml_parser_callback_pf.
	bt_xml_parser_t	This is type bt_xml_parser_t.

Utility Functions

	Name	Description
	bt_alloc_buffer	This is function bt_alloc_buffer.
	bt_at_parse_fragment	This is function bt_at_parse_fragment.
	bt_at_parser_reset	This is function bt_at_parser_reset.
	bt_free_buffer	This is function bt_free_buffer.
	bt_get_buffer	This is function bt_get_buffer.
	bt_get_buffer_header	This is function bt_get_buffer_header.
	bt_get_buffer_index	This is function bt_get_buffer_index.
	bt_init_buffer_mgr	This is function bt_init_buffer_mgr.
	bt_q_add	This is function bt_q_add.
	bt_q_get	This is function bt_q_get.
	bt_q_get_head	This is function bt_q_get_head.
	bt_q_get_length	This is function bt_q_get_length.
	bt_q_get_next	This is function bt_q_get_next.
	bt_q_push	This is function bt_q_push.
	bt_q_remove	This is function bt_q_remove.
	bt_q_remove_by_idx	This is function bt_q_remove_by_idx.
	bt_vcard_parse_fragment	This is function bt_vcard_parse_fragment.
	bt_vcard_parser_reset	This is function bt_vcard_parser_reset.
	bt_xml_parse_fragment	This is function bt_xml_parse_fragment.
	bt_xml_parser_reset	This is function bt_xml_parser_reset.
	_compact_buffer	This is function _compact_buffer.
	_expand_buffer	This is function _expand_buffer.
	_read_bdaddr	This is function _read_bdaddr.
	_readb	This is function _readb.
	_readi	This is function _readi.
	_readin	This is function _readin.
	_readl	This is function _readl.
	_readln	This is function _readln.
	_readui	This is function _readui.
	_readuin	This is function _readuin.
	_readul	This is function _readul.
	_readuln	This is function _readuln.
	_str2ulong_dec	This is function _str2ulong_dec.
	_to_lower_case	This is function _to_lower_case.
	_ulong2str	This is function _ulong2str.
	_ulong2str_dec	This is function _ulong2str_dec.
	_write_bdaddr	This is function _write_bdaddr.
	_writeb	This is function _writeb.
	_writei	This is function _writei.
	_writein	This is function _writein.
	_writel	This is function _writel.
	_writeln	This is function _writeln.
	_writes	This is function _writes.
	_writesx	This is function _writesx.
	_zero_memory	This is function _zero_memory.
	_bt_memcpy	This is function _bt_memcpy.
	_is_empty_str	This is function _is_empty_str.
	_str2ulong	This is function _str2ulong.

Description

This section describes the Application Programming Interface (API) functions of the PIC32 Bluetooth Stack Library.

Bluetooth Support Functions

bt_bdaddr_is_null Function

File

bt_bdaddr.h

C

```
bt_bool bt_bdaddr_is_null(const bt_bdaddr_t* bda);
```

Description

This is function bt_bdaddr_is_null.

bt_bdaddrs_are_equal Function

File

bt_bdaddr.h

C

```
bt_bool bt_bdaddrs_are_equal(const bt_bdaddr_t* bda1, const bt_bdaddr_t* bda2);
```

Description

This is function bt_bdaddrs_are_equal.

bt_log_bdaddr Function

File

bt_log.h

C

```
void bt_log_bdaddr(const char* msg, const bt_bdaddr_t* addr);
```

Description

This is function bt_log_bdaddr.

bt_log_int Function

File

bt_log.h

C

```
void bt_log_int(const char* msg, int i);
```

Description

This is function bt_log_int.

bt_log_linkkey Function

File

bt_log.h

C

```
void bt_log_linkkey(const char* msg, const bt_linkkey_t* key);
```

Description

This is function bt_log_linkkey.

bt_log_memory Function

File

bt_log.h

C

```
void bt_log_memory(const char* msg, const bt_byte* ptr, bt_ulong count);
```

Description

This is function bt_log_memory.

bt_log_msg Function

File

bt_log.h

C

```
void bt_log_msg(const char* msg);
```

Description

This is function bt_log_msg.

bt_oem_assert Function

File

bt_oem.h

C

```
void bt_oem_assert(const char* file, int line);
```

Description

This is function bt_oem_assert.

bt_oem_get_device_class Function

File

bt_oem.h

C

```
bt_long bt_oem_get_device_class();
```

Description

This is function bt_oem_get_device_class.

bt_oem_get_device_name Function

File

bt_oem.h

C

```
const char* bt_oem_get_device_name();
```

Description

This is function bt_oem_get_device_name.

bt_oem_get_pin_code Function

File

bt_oem.h

C

```
void bt_oem_get_pin_code(bt_bdaddr_t* bdaddr_remote);
```

Description

This is function bt_oem_get_pin_code.

bt_oem_linkkey_notification Function

File

bt_oem.h

C

```
void bt_oem_linkkey_notification(bt_linkkey_notification_t* lkn);
```

Description

This is function bt_oem_linkkey_notification.

bt_oem_linkkey_request Function

File

bt_oem.h

C

```
void bt_oem_linkkey_request(bt_linkkey_request_t* lkr);
```

Description

This is function bt_oem_linkkey_request.

bt_oem_log_write Function

File

bt_log.h

C

```
void bt_oem_log_write(const char* msg);
```

Description

brief Output log message. ingroup log

details DotStack calls this function to output its debug information. Implementation should output or store the specified message to whatever device or medium where it can be examined and analyzed.

bt_oem_recv Function

File

bt_hcitr.h

C

```
void bt_oem_recv(bt_byte* buffer, bt_uint len, bt_oem_recv_callback_fp callback);
```

Description

- Receive data.

This function is called by the HCI layer when it needs more data from the HCI controller. Implementation of this function must receive the specified number of bytes from the HCI controller and call the provided callback function.

@param buffer Pointer to a buffer for the received data. The buffer must be long enough to accommodate the number of bytes specified by the par len parameter.

@param len Number of bytes to receive.

@param callback A callback function that must be called when the requested number of bytes have been received.

bt_oem_schedule_signals Function

File

bt_oem.h

C

```
void bt_oem_schedule_signals();
```

Description

This is function bt_oem_schedule_signals.

bt_oem_send Function

File

bt_hcitr.h

C

```
void bt_oem_send(const bt_byte* buffer, bt_uint len, bt_oem_send_callback_fp callback);
```

Description

- Send data.

This function is called by the HCI layer when it needs to send data to the HCI controller. Implementation of this function must send the specified number of bytes to the HCI controller and call the provided callback function.

@param buffer Pointer to the data to be sent .

@param len Number of bytes to send.

@param callback A callback function that must be called when all data have been sent.

bt_oem_storage_get_capacity Function

File

bt_storage.h

C

```
bt_uint bt_oem_storage_get_capacity();
```

Description

brief Get non-volatile storage capacity. ingroup stg

details Implementation of this function must return the capacity of its non-volatile storage.

bt_oem_storage_read Function

File

bt_storage.h

C

```
void bt_oem_storage_read(bt_int addr, bt_byte* buffer, bt_int len, bt_storage_callback_fp callback);
```

Description

brief Read from the non-volatile storage. ingroup stg

details This function is called by the stack to read from the non-volatile storage. This function must be implemented by the application. When this function is called the application must start a read operation. When the number of bytes specified by the c len parameter is read, the application must call the callback function specified by the c callback parameter. The application does not have to read the whole number of bytes during the call to this function. It may complete reading later and then call the completion callback. The stack guarantees that the destination data buffer will be available until the application calls the completion callback.

param addr The non-volatile storage address where to read data from. param buffer The receiving buffer. param len The number of bytes to read. param callback The completion callback function.

bt_oem_storage_start Function

File

bt_storage.h

C

```
void bt_oem_storage_start();
```

Description

brief Begin a sequence of non-volatile storage operations. ingroup stg

details DotStack calls this function when it starts a sequence of non-volatile storage operations. When the sequence is finished, DotStack will call bt_oem_storage_stop().

bt_oem_storage_stop Function

File

bt_storage.h

C

```
void bt_oem_storage_stop();
```

Description

brief End a sequence of non-volatile storage operations. ingroup stg

details DotStack calls this function when it finishes executing a sequence of non-volatile storage operations.

bt_oem_storage_write Function

File

bt_storage.h

C

```
void bt_oem_storage_write(bt_int addr, const bt_byte* data, bt_int len, bt_storage_callback_fp callback);
```

Description

brief Write to non-volatile storage. ingroup stg

details This function is called by the stack to write data to the non-volatile storage. This function must be implemented by the application. When this function is called, the application must start writing specified data to the non-volatile storage. When all data has been written, the application must call the callback function passed in the c callback parameter. The application does not have to complete the write operation during the call to this function. It may complete the operation later and then call the callback function. In this case, the application does not have to store the data in an internal buffer. The stack guarantees that the passed data will be present until the completion callback is called by the application.

param addr The persitent storage address where to write data to. param data Pointer to data. param len Data length. param callback The completion callback function.

bt_oem_timer_clear Function

File

bt_timer.h

C

```
void bt_oem_timer_clear(bt_uint timerId);
```

Description

brief Clear timer.

details This function must be implemented by the application. When this function is called the application must clear the specified timer. If it is already expired and a callback is currently pending, the application should also take measures to cancel the callback.

param timerId ID of the timer to clear.

bt_oem_timer_set Function

File

bt_timer.h

C

```
void bt_oem_timer_set(bt_uint timerId, bt_ulong milliseconds, bt_timer_callback_fp callback);
```

Description

brief Set timer.

details This function must be implemented by the application. When it is called, the application must set the specified timer. When the timer expires, the application must call the passed callback function. The function must not wait until the timer expires. It must set the timer and exit immediately.

param timerId ID of the timer to set. param milliseconds Timer interval in milliseconds param callback Timer expiration callback function.

bt_signal_process_pending Function

File

bt_signal.h

C

```
void bt_signal_process_pending();
```

Description

This is function bt_signal_process_pending.

bt_signal_register Function

File

bt_signal.h

C

```
bt_bool bt_signal_register(bt_signal_t* signal, bt_signal_handler_fp handler, void* handler_param);
```

Description

This is function bt_signal_register.

bt_signal_set Function

File

bt_signal.h

C

```
void bt_signal_set(bt_signal_t* signal);
```

Description

This is function bt_signal_set.

bt_signal_unregister Function

File

bt_signal.h

C

```
void bt_signal_unregister(bt_signal_t* signal);
```

Description

This is function bt_signal_unregister.

bt_sys_get_connectable Function

File

bt_system.h

C

```
bt_bool bt_sys_get_connectable();
```

Description

This is function bt_sys_get_connectable.

bt_sys_get_discoverable Function

File

bt_system.h

C

```
bt_bool bt_sys_get_discoverable();
```

Description

This is function bt_sys_get_discoverable.

bt_sys_get_l2cap_manager Function

File

bt_system.h

C

```
bt_l2cap_mgr_t* bt_sys_get_l2cap_manager();
```

Description

- Get the L2CAP manager.

This function returns the L2CAP manager. The L2CAP manager is created as part of the start up sequence.

@return The L2CAP manager.

bt_sys_init Function

File

bt_system.h

C

```
void bt_sys_init();
```

Description

- Initialize the Bluetooth system.

This function initializes all internal variables of HCI, L2CAP and SDP modules. It must be called by the application before it can access any functionality provided by the library. In addition to this initialization function the application must call initialization functions of all other profile modules the application is intended to use. E.g., if the application is using the SPP module the bt_spp_init() must be called right after calling bt_sys_init().

This function essentially calls bt_sys_init_ex(HCI_LINK_POLICY_ENABLE_ALL) so all link policy setting are enabled.

bt_sys_set_modes Function

File

bt_system.h

C

```
void bt_sys_set_modes(bt_bool discoverable, bt_bool connectable, bt_sys_callback_fp callback, void*
```

```
callback_param);
```

Description

This is function bt_sys_set_modes.

bt_sys_start Function

File

bt_system.h

C

```
void bt_sys_start(bt_bool discoverable, bt_bool connectable, const bt_byte* sdp_db, bt_uint sdp_db_len,
bt_sys_callback_fp callback, void* callback_param);
```

Description

- Start the Bluetooth system.

After all modules used by the application have been initialized this function should be called to start the Bluetooth system operation. During the start up sequence it will reset and initialize the HCI controller and then create the L2CAP manager. The application will be notified when the start up sequence completes by calling the provided callback function.

Also, the caller must provide an SDP database.

@param discoverable defines whether the device is discoverable after reset. @param connectable defines whether the device is connectable after reset. @param sdp_db SDP database data. @param sdp_db_len Length of SDP database data. @param callback A callback function that will be called when the start up sequence is complete. @param callback_param An arbitrary pointer that will be passed to the callback function.

HCITR_BCSP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HCITR_BCSP_ALLOCATE_BUFFERS();
```

Description

This is function HCITR_BCSP_ALLOCATE_BUFFERS.

IAPEA_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
IAPEA_ALLOCATE_BUFFERS();
```

Description

This is function IAPEA_ALLOCATE_BUFFERS.

bt_oem_ssp_callback Function

File

bt_oem.h

C

```
void bt_oem_ssp_callback(SSP_EVENT spp_event, void* event_param, void* init_param);
```

Description

This is function bt_oem_ssp_callback.

bt_signal_init Function

File

bt_signal.h

C

```
void bt_signal_init();
```

Description

This is function bt_signal_init.

bt_sys_init_ex Function

File

bt_system.h

C

```
void bt_sys_init_ex(bt_byte default_link_policy);
```

Description

- Initialize the Bluetooth system.

This function initializes all internal variables of HCI, L2CAP and SDP modules. It must be called by the application before it can access any functionality provided by the library. In addition to this initialization function the application must call initialization functions of all other profile modules the application is intended to use. E.g., if the application is using the SPP module the bt_spp_init() must be called right after calling bt_sys_init().

Also, the caller must provide an SDP database.

@param default_link_policy default link policy settings. This is a bitmask that defines the initial value of the link policy settings for all new BR/EDR connections. This value can be a combination of the following values: li HCI_LINK_POLICY_ENABLE_ROLE_SWITCH li HCI_LINK_POLICY_ENABLE_HOLD_MODE li HCI_LINK_POLICY_ENABLE_SNIFF_MODE li HCI_LINK_POLICY_ENABLE_PARK_STATE

To enable all settings pass HCI_LINK_POLICY_ENABLE_ALL.

HCRP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
HCRP_ALLOCATE_BUFFERS();
```

Description

This is function HCRP_ALLOCATE_BUFFERS.

MAP_ALLOCATE_BUFFERS Function

File

bt_oem_config.h

C

```
MAP_ALLOCATE_BUFFERS();
```

Description

This is function MAP_ALLOCATE_BUFFERS.

bt_hcitr_3wire_cancel_recv_packet Function

File

hcitr_3wire.h

C

```
void bt_hcitr_3wire_cancel_recv_packet();
```

Description

This is function bt_hcitr_3wire_cancel_recv_packet.

bt_q_contains Function

File

queue.h

C

```
bt_bool bt_q_contains(bt_queue_element_t* phead, void* element);
```

Description

This is function bt_q_contains.

Bluetooth Support Data Types and Constants

ARG_NOT_USED Macro

File

bt_private.h

C

```
#define ARG_NOT_USED(arg) ((void)(arg))
```

Description

This is macro ARG_NOT_USED.

BT_ASSERT Macro

File

bt_private.h

C

```
#define BT_ASSERT(test) ((test) ? (void)0 : bt_oem_assert(__FILE__,__LINE__))
```

Description

This is macro BT_ASSERT.

BT_FALSE Macro

File

bt_types.h

C

```
#define BT_FALSE 0
```

Description

This is macro BT_FALSE.

BT_LINKKEY_LENGTH Macro

File

bt_types.h

C

```
#define BT_LINKKEY_LENGTH 16
```

Description

This is macro BT_LINKKEY_LENGTH.

BT_LOG Macro

File

bt_log.h

C

```
#define BT_LOG(msg)
```

Description

This is macro BT_LOG.

BT_LOG_EX Macro

File

bt_log.h

C

```
#define BT_LOG_EX(msg, level)
```

Description

This is macro BT_LOG_EX.

BT_LOG_LEVEL_ALL Macro

File

bt_log.h

C

```
#define BT_LOG_LEVEL_ALL 0
```

Description

This is macro BT_LOG_LEVEL_ALL.

BT_LOG_LEVEL_DEBUG Macro

File

bt_log.h

C

```
#define BT_LOG_LEVEL_DEBUG 80
```

Description

This is macro BT_LOG_LEVEL_DEBUG.

BT_LOG_LEVEL_ERROR Macro

File

bt_log.h

C

```
#define BT_LOG_LEVEL_ERROR 240
```

Description

This is macro BT_LOG_LEVEL_ERROR.

BT_LOG_LEVEL_INFO Macro

File

bt_log.h

C

```
#define BT_LOG_LEVEL_INFO 160
```

Description

This is macro BT_LOG_LEVEL_INFO.

BT_LOG_LEVEL_OFF Macro

File

bt_log.h

C

```
#define BT_LOG_LEVEL_OFF 0xFF
```

Description

This is macro BT_LOG_LEVEL_OFF.

BT_LOGADDR Macro

File

bt_log.h

C

```
#define BT_LOGADDR(msg, a)
```

Description

This is macro BT_LOGADDR.

BT_LOGADDR_EX Macro

File

bt_log.h

C

```
#define BT_LOGADDR_EX(msg, a, level)
```

Description

This is macro BT_LOGADDR_EX.

BT_LOGINT Macro

File

bt_log.h

C

```
#define BT_LOGINT(msg, i)
```

Description

This is macro BT_LOGINT.

BT_LOGINT_EX Macro

File

bt_log.h

C

```
#define BT_LOGINT_EX(msg, i, level)
```

Description

This is macro BT_LOGINT_EX.

BT_LOGLINKKEY Macro

File

bt_log.h

C

```
#define BT_LOGLINKKEY(msg, key)
```

Description

This is macro BT_LOGLINKKEY.

BT_LOGLINKKEY_EX Macro

File

bt_log.h

C

```
#define BT_LOGLINKKEY_EX(msg, key, level)
```

Description

This is macro BT_LOGLINKKEY_EX.

BT_LOGMEMORY Macro

File

bt_log.h

C

```
#define BT_LOGMEMORY(msg, ptr, len)
```

Description

This is macro BT_LOGMEMORY.

BT_LOGMEMORY_EX Macro

File

bt_log.h

C

```
#define BT_LOGMEMORY_EX(msg, ptr, count, level)
```

Description

This is macro BT_LOGMEMORY_EX.

BT_LOGWRITE Macro

File

bt_log.h

C

```
#define BT_LOGWRITE(msg)
```

Description

This is macro BT_LOGWRITE.

BT_MAKE_BDADDR Macro

File

bt_types.h

C

```
#define BT_MAKE_BDADDR(bdaddr, maddr, laddr) bdaddr.bd_addr_m = maddr, bdaddr.bd_addr_l = laddr
```

Description

This is macro BT_MAKE_BDADDR.

BT_MAKE_BDADDR_LE Macro

File

bt_types.h

C

```
#define BT_MAKE_BDADDR_LE(bdaddr, addr_type, maddr, laddr) bdaddr.bd_addr_m = maddr | 0x80000000 |
((bt_ulong)addr_type << 30), bdaddr.bd_addr_l = laddr
```

Description

This is macro BT_MAKE_BDADDR_LE.

BT_NO Macro

File

bt_types.h

C

```
#define BT_NO 0
```

Description

This is macro BT_NO.

BT_TRUE Macro

File

bt_types.h

C

```
#define BT_TRUE 1
```

Description

This is macro BT_TRUE.

BT_YES Macro

File

bt_types.h

C

```
#define BT_YES 1
```

Description

This is macro BT_YES.

CDS_LAST_DEVICE_ADDR Macro

File

bt_storage.h

C

```
#define CDS_LAST_DEVICE_ADDR HIDS_LAST_DEVICE_ADDR
```

Description

This is macro CDS_LAST_DEVICE_ADDR.

CDS_SIGNATURE Macro

File

bt_storage.h

C

```
#define CDS_SIGNATURE HIDS_SIGNATURE
```

Description

This is macro CDS_SIGNATURE.

CDS_SIGNATURE_ADDR Macro

File

bt_storage.h

C

```
#define CDS_SIGNATURE_ADDR HIDS_SIGNATURE_ADDR
```

Description

This is macro CDS_SIGNATURE_ADDR.

FALSE Macro

File

bt_private.h

C

```
#define FALSE 0
```

Description

This is macro FALSE.

HCI_BDADDR_LEN Macro

File

bt_storage.h

C

```
#define HCI_BDADDR_LEN 6
```

Description

This is macro HCI_BDADDR_LEN.

HIDS_LAST_DEVICE_ADDR Macro

File

bt_storage.h

C

```
#define HIDS_LAST_DEVICE_ADDR HIDS_SIGNATURE_ADDR + 1
```

Description

This is macro HIDS_LAST_DEVICE_ADDR.

HIDS_SIGNATURE Macro

File

bt_storage.h

C

```
#define HIDS_SIGNATURE 0x8E
```

Description

This is macro HIDS_SIGNATURE.

HIDS_SIGNATURE_ADDR Macro

File

bt_storage.h

C

```
#define HIDS_SIGNATURE_ADDR LKS_SIGNATURE_ADDR + 2 + (HCI_BDADDR_LEN + HCI_LINK_KEY_LEN) * LKS_MAX_LINK_KEYS
```

Description

This is macro HIDS_SIGNATURE_ADDR.

LKS_FIRST_KEY_ADDR Macro

File

bt_storage.h

C

```
#define LKS_FIRST_KEY_ADDR 2
```

Description

This is macro LKS_FIRST_KEY_ADDR.

LKS_MAX_LINK_KEYS Macro

File

bt_storage.h

C

```
#define LKS_MAX_LINK_KEYS 5
```

Description

This is macro LKS_MAX_LINK_KEYS.

LKS_SIGNATURE Macro

File

bt_storage.h

C

```
#define LKS_SIGNATURE 0x9D
```

Description

This is macro LKS_SIGNATURE.

LKS_SIGNATURE_ADDR Macro

File

bt_storage.h

C

```
#define LKS_SIGNATURE_ADDR 0
```

Description

This is macro LKS_SIGNATURE_ADDR.

LOG Macro

File

bt_private.h

C

```
#define LOG(msg) BT_LOG(msg)
```

Description

This is macro LOG.

LOG_EX Macro

File

bt_private.h

C

```
#define LOG_EX(msg, level) BT_LOG_EX(msg, level)
```

Description

This is macro LOG_EX.

LOGADDR Macro

File

bt_private.h

C

```
#define LOGADDR(msg, a) BT_LOGADDR(msg, a)
```

Description

This is macro LOGADDR.

LOGADDR_EX Macro

File

bt_private.h

C

```
#define LOGADDR_EX(msg, a, level) BT_LOGADDR_EX(msg, a, level)
```

Description

This is macro LOGADDR_EX.

LOGCLEAR Macro

File

bt_private.h

C

```
#define LOGCLEAR BT_LOGCLEAR()
```

Description

This is macro LOGCLEAR.

LOGINT Macro

File

bt_private.h

C

```
#define LOGINT(msg, i) BT_LOGINT(msg, i)
```

Description

This is macro LOGINT.

LOGINT_EX Macro

File

bt_private.h

C

```
#define LOGINT_EX(msg, i, level) BT_LOGINT_EX(msg, i, level)
```

Description

This is macro LOGINT_EX.

LOGMEMORY Macro

File

bt_private.h

C

```
#define LOGMEMORY(msg, ptr, count) BT_LOGMEMORY(msg, ptr, count)
```

Description

This is macro LOGMEMORY.

LOGMEMORY_EX Macro

File

bt_private.h

C

```
#define LOGMEMORY_EX(msg, ptr, count, level) BT_LOGMEMORY_EX(msg, ptr, count, level)
```

Description

This is macro LOGMEMORY_EX.

LOGWRITE Macro

File

bt_private.h

C

```
#define LOGWRITE(msg) BT_LOGWRITE(msg)
```

Description

This is macro LOGWRITE.

NULL Macro

File

bt_private.h

C

```
#define NULL 0
```

Description

This is macro NULL.

TRUE Macro

File

bt_private.h

C

```
#define TRUE 1
```

Description

This is macro TRUE.

_bt_signal_t Structure

File

bt_signal.h

C

```
struct _bt_signal_t {
  bt_signal_t* next_signal;
  bt_byte signaled;
  bt_signal_handler_fp handler;
  void* handler_param;
};
```

Description

This is record _bt_signal_t.

bt_bdaddr_cp Type

File

bt_types.h

C

```
typedef const bt_bdaddr_t* bt_bdaddr_cp;
```

Description

This is type bt_bdaddr_cp.

bt_bdaddr_p Type

File

bt_types.h

C

```
typedef bt_bdaddr_t* bt_bdaddr_p;
```

Description

This is type bt_bdaddr_p.

bt_bdaddr_t Structure

File

bt_types.h

C

```
typedef struct _bt_bdaddr_s {
  bt_ulong bd_addr_l;
  bt_ulong bd_addr_m;
} bt_bdaddr_t;
```

Description

This is type bt_bdaddr_t.

bt_bool Type

File

bt_types.h

C

```
typedef bt_int bt_bool;
```

Description

This is type bt_bool.

bt_byte_cp Type

File

bt_types.h

C

```
typedef const bt_byte* bt_byte_cp;
```

Description

This is type bt_byte_cp.

bt_byte_p Type

File

bt_types.h

C

```
typedef bt_byte* bt_byte_p;
```

Description

This is type bt_byte_p.

bt_char Type

File

bt_types.h

C

```
typedef char bt_char;
```

Description

This is type bt_char.

bt_char_cp Type

File

bt_types.h

C

```
typedef const bt_char* bt_char_cp;
```

Description

This is type bt_char_cp.

bt_char_p Type

File

bt_types.h

C

```
typedef bt_char* bt_char_p;
```

Description

This is type bt_char_p.

bt_id Type

File

bt_types.h

C

```
typedef bt_int bt_id;
```

Description

This is type bt_id.

bt_int_cp Type

File

bt_types.h

C

```
typedef const bt_int* bt_int_cp;
```

Description

This is type bt_int_cp.

bt_int_p Type

File

bt_types.h

C

```
typedef bt_int* bt_int_p;
```

Description

This is type bt_int_p.

bt_linkkey_cp Type

File

bt_types.h

C

```
typedef const bt_linkkey_t* bt_linkkey_cp;
```

Description

This is type bt_linkkey_cp.

bt_linkkey_notification_t Structure

File

bt_oem.h

C

```
typedef struct _bt_linkkey_notification_t {
  bt_bdaddr_t bdaddr_remote;
  bt_linkkey_t key;
  bt_byte key_type;
} bt_linkkey_notification_t;
```

Description

This is type bt_linkkey_notification_t.

bt_linkkey_p Type

File

bt_types.h

C

```
typedef bt_linkkey_t* bt_linkkey_p;
```

Description

This is type bt_linkkey_p.

bt_linkkey_request_t Structure

File

bt_oem.h

C

```
typedef struct _bt_linkkey_request_t {
  bt_bdaddr_t bdaddr_remote;
} bt_linkkey_request_t;
```

Description

This is type bt_linkkey_request_t.

bt_linkkey_t Structure

File

bt_types.h

C

```
typedef struct _bt_linkkey_t {
  bt_byte data[BT_LINKKEY_LENGTH];
} bt_linkkey_t;
```

Description

This is type bt_linkkey_t.

bt_long_cp Type

File

bt_types.h

C

```
typedef const bt_long* bt_long_cp;
```

Description

This is type bt_long_cp.

bt_long_p Type

File

bt_types.h

C

```
typedef bt_long* bt_long_p;
```

Description

This is type bt_long_p.

bt_oem_recv_callback_fp Type

File

bt_hcitr.h

C

```
typedef void (* bt_oem_recv_callback_fp)(bt_uint len);
```

Description

- Receive callback.

This callback function is called when a receive operation initiated by bt_oem_recv() has completed.

@param len Number of received bytes. The value of this parameter should always be the same as the number of bytes requested in a call to bt_oem_recv().

bt_oem_send_callback_fp Type

File

bt_hcitr.h

C

```
typedef void (* bt_oem_send_callback_fp)(void);
```

Description

- Send callback.

This callback function is called when a send operation initiated by bt_oem_send() has completed.

bt_signal_handler_fp Type

File

bt_signal.h

C

```
typedef void (* bt_signal_handler_fp)(bt_signal_t* signal, void* param);
```

Description

This is type bt_signal_handler_fp.

bt_signal_t Type

File

bt_signal.h

C

```
typedef struct _bt_signal_t bt_signal_t;
```

Description

This is type bt_signal_t.

bt_storage_callback_fp Type

File

bt_storage.h

C

```
typedef void (* bt_storage_callback_fp)(void);
```

Description

brief Storage callback.

details This callback is called when a non-volatile storage operation completes.

bt_sys_callback_fp Type

File

bt_system.h

C

```
typedef void (* bt_sys_callback_fp)(bt_bool success, void* param);
```

Description

- System start callback.

This callback function is called when system start initiated by bt_sys_start() has completed.

@param success Success of the operation: c BT_TRUE if successfull, c BT_FALSE otherwise.

@param param Callback parameter that was specified when bt_sys_start() was called.

bt_timer_callback_fp Type

File

bt_timer.h

C

```
typedef void (* bt_timer_callback_fp)(void);
```

Description

brief Timer callback.

details This callback is called when a timer expires.

bt_timer_id Enumeration

File

bt_timer.h

C

```
typedef enum _bt_timer_id_enum {
  BT_TIMER_L2CAP,
  BT_TIMER_RFCOMM,
  BT_TIMER_WAKEUP_ACK,
  BT_TIMER_TEST,
  BT_TIMER_HCI,
  BT_TIMER_IAP,
  BT_TIMER_HSP_AG,
  BT_TIMER_HFP_AG,
  BT_TIMER_ATT,
  BT_TIMER_ATT_CLIENT,
  BT_TIMER_AVRCP,
  BT_TIMER_SMP,
  BT_TIMER_IAP2,
  BT_TIMER_HCRP,
  BT_TIMER_3WIRE_T0,
  BT_TIMER_3WIRE_T1,
  BT_TIMER_MAX
} bt_timer_id;
```

Description

This is type bt_timer_id.

bt_uint_cp Type

File

bt_types.h

C

```
typedef const bt_uint* bt_uint_cp;
```

Description

This is type bt_uint_cp.

bt_uint_p Type

File

bt_types.h

C

```
typedef bt_uint* bt_uint_p;
```

Description

This is type bt_uint_p.

bt_ulong_cp Type

File

bt_types.h

C

```
typedef const bt_ulong* bt_ulong_cp;
```

Description

This is type bt_ulong_cp.

bt_ulong_p Type

File

bt_types.h

C

```
typedef bt_ulong* bt_ulong_p;
```

Description

This is type bt_ulong_p.

bt_uuid_cp Type

File

bt_types.h

C

```
typedef const bt_uuid_t* bt_uuid_cp;
```

Description

This is type bt_uuid_cp.

bt_uuid_p Type

File

bt_types.h

C

```
typedef bt_uuid_t* bt_uuid_p;
```

Description

This is type bt_uuid_p.

bt_uuid_t Structure

File

bt_types.h

C

```
typedef struct _bt_uuid_s {
  bt_ulong uuid0;
  bt_ulong uuid1;
  bt_ulong uuid2;
  bt_ulong uuid3;
} bt_uuid_t;
```

Description

This is type bt_uuid_t.

bt_uuid16 Type

File

bt_types.h

C

```
typedef bt_uint bt_uuid16;
```

Description

This is type bt_uuid16.

bt_uuid32 Type

File

bt_types.h

C

```
typedef bt_ulong bt_uuid32;
```

Description

This is type bt_uuid32.

A2DP Functions

_bt_a2dp_avdtp_mgr_callback Function

File

a2dp_private.h

C

```
void _bt_a2dp_avdtp_mgr_callback(bt_avdtp_mgr_t* mgr, bt_byte evt, bt_avdtp_event_t* evt_param, void*
callback_param);
```

Description

This is function _bt_a2dp_avdtp_mgr_callback.

bt_a2dp_aac_codec_handler Function

File

a2dp_codec_aac.h

C

```
bt_byte bt_a2dp_aac_codec_handler(bt_avdtp_codec_t* codec, bt_byte opcode, bt_avdtp_codec_op_param_t*
op_param, bt_avdtp_mgr_t* mgr);
```

Description

This is function bt_a2dp_aac_codec_handler.

bt_a2dp_find_server Function

File

a2dp.h

C

```
bt_bool bt_a2dp_find_server(bt_bdaddr_t* deviceAddress, const bt_uuid_t* service_class_id,
bt_a2dp_find_server_callback_fp callback, bt_sdp_client_callback_fp client_callback, void* callback_param);
```

Description

This is function bt_a2dp_find_server.

bt_a2dp_find_sink Function

File

a2dp.h

C

```
bt_bool bt_a2dp_find_sink(bt_bdaddr_t* deviceAddress, bt_a2dp_find_server_callback_fp callback,
bt_sdp_client_callback_fp client_callback, void* callback_param);
```

Description

brief Find sink ingroup a2dp

details This function looks for a sink on a remote device specified by c deviceAddress and, if found, returns features supported by the sink.

param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED

param callback_param A pointer to arbitrary data to be passed to the c callback and c client_callback callbacks.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_a2dp_find_source Function

File

a2dp.h

C

```
bt_bool bt_a2dp_find_source(bt_bdaddr_t* deviceAddress, bt_a2dp_find_server_callback_fp callback,
bt_sdp_client_callback_fp client_callback, void* callback_param);
```

Description

brief Find source ingroup a2dp

details This function looks for a source on a remote device specified by c deviceAddress and, if found, returns features supported by the source.

param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED

param callback_param A pointer to arbitrary data to be passed to the c callback and c client_callback callbacks.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_a2dp_get_mgr Function

File

a2dp.h

C

```
bt_a2dp_mgr_t* bt_a2dp_get_mgr();
```

Description

brief Return a pointer to an instance of the A2DP manager. ingroup a2dp

details This function returns a pointer to an instance of the A@DP manager. There is only one instance of the manager allocated by the stack. The pointer is passed as the first parameter to all A2DP functions.

bt_a2dp_init Function

File

a2dp.h

C

```
void bt_a2dp_init();
```

Description

brief Initialize the A2DP layer. ingroup a2dp

details This function initializes the A2DP layer of the stack. It must be called prior to any other A2DP function can be called.

bt_a2dp_mpeg_codec_handler Function

File

a2dp_codec_mpeg.h

C

```
bt_byte bt_a2dp_mpeg_codec_handler(bt_avdtp_codec_t* codec, bt_byte opcode, bt_avdtp_codec_op_param_t*
op_param, bt_avdtp_mgr_t* mgr);
```

Description

This is function bt_a2dp_mpeg_codec_handler.

bt_a2dp_open_and_start_stream Function

File

a2dp.h

C

```
bt_bool bt_a2dp_open_and_start_stream(bt_a2dp_mgr_t* mgr, bt_byte strm_handle, bt_bdaddr_t* remote_addr,
bt_byte seid_int, bt_byte seid_acp, const bt_avdtp_sep_capabilities_t* caps);
```

Description

brief Open & start a stream ingroup a2dp

details Opening a stream involves sending 3 requests to a remote device - "set configuration", "open stream" and "start stream". Each event generates its own event which must be handled and acted accordingly by the application. To make the use of API easier dotstack combines all these requests in one request called "open & start stream". dotstack sends necessary requests in a proper sequence, handles responses and generates only one event (A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED) at the end. If any of the individual requests has failed the event's parameter bt_a2dp_event_t::open_and_start_stream_completed is populated with the error code and request id which caused it.

param mgr A2DP manager. param strm_handle Stream handle. param remote_addr The address of a remote device. param seid_int Local SEP ID. param seid_acp Remote SEP ID. param caps Stream configuration.

return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.

bt_a2dp_register_aac_codec Function

File

a2dp.h

C

```
void bt_a2dp_register_aac_codec(bt_a2dp_mgr_t* mgr);
```

Description

brief Register default AAC codec ingroup a2dp

details This function adds AAC codec implemented by dotstack to the list of known codecs. For more information about codecs see description of ::bt_avdtp_register_codec. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use AAC codec it must call this function when it is initializing.

note dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration.

param mgr A2DP manager.

bt_a2dp_register_callback Function

File

a2dp.h

C

```
void bt_a2dp_register_callback(bt_a2dp_mgr_t* mgr, bt_a2dp_mgr_callback_fp callback, void* callback_param);
```

Description

brief Register a A2DP application callback. ingroup a2dp

details In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values:

@arg A2DP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg A2DP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg A2DP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local device received a response to Get SEP capabilities operation. @arg A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Set stream configuration operation. @arg A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Get stream configuration operation. @arg A2DP_EVT_STREAM_RECONFIGURE_COMPLETED: Local device received a response to Reconfigure stream operation. @arg A2DP_EVT_OPEN_STREAM_COMPLETED: Local device received a response to Open stream operation. @arg A2DP_EVT_START_STREAM_COMPLETED: Local device received a response to Start stream operation. @arg A2DP_EVT_CLOSE_STREAM_COMPLETED: Local device received a response to Close stream operation. @arg A2DP_EVT_SUSPEND_STREAM_COMPLETED: Local device received a response to Suspend stream operation. @arg A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: Local device received a response to Stream security control operation. @arg A2DP_EVT_ABORT_STREAM_COMPLETED: Local device received a response to Abort stream operation. @arg A2DP_EVT_SEP_INFO_RECEIVED: SEP information received. @arg A2DP_EVT_SEP_CAPABILITIES_RECEIVED: SEP capabilities received. @arg A2DP_EVT_STREAM_CONFIGURATION_RECEIVED: Stream configuration received.

@arg A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: Remote device requested stream configuration. @arg A2DP_EVT_OPEN_STREAM_REQUESTED: Remote device requested to open a stream. @arg A2DP_EVT_START_STREAM_REQUESTED: Remote device requested to start a stream. @arg A2DP_EVT_CLOSE_STREAM_REQUESTED: Remote device requested to close a stream. @arg A2DP_EVT_SUSPEND_STREAM_REQUESTED: Remote device requested to suspend a stream. @arg A2DP_EVT_ABORT_STREAM_REQUESTED: Remote device requested to abort a stream. @arg A2DP_EVT_RECONFIGURE_STREAM_REQUESTED: Remote device requested to reconfigure a stream. @arg A2DP_EVT_MEDIA_PACKET_RECEIVED: Remote device sent a media packet. @arg A2DP_EVT_STREAM_CONFIGURED: A stream has been configured (This event is generated right after A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the local devices accepted the request). @arg A2DP_EVT_STREAM_RECONFIGURED: A stream has been re-configured (This event is generated right after A2DP_EVT_RECONFIGURE_STREAM_REQUESTED if the local devices accepted the request). @arg A2DP_EVT_STREAM_OPENED: A stream has been opened (This event is generated as a result of local or remote stream opening request). @arg A2DP_EVT_STREAM_STARTED:

A stream has been started (This event is generated right after A2DP_EVT_START_STREAM_REQUESTED if the local devices accepted the request). @arg A2DP_EVT_STREAM_CLOSED: A stream has been close (This event is generated right after A2DP_EVT_CLOSE_STREAM_REQUESTED if the local devices accepted the request). @arg A2DP_EVT_STREAM_SUSPENDED: A stream has been suspended (This event is generated right after A2DP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request). @arg A2DP_EVT_STREAM_ABORTED: A stream has been aborted (This event is generated right after A2DP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request. It is also generated if connection between devices has been terminated by means other than A2DP signaling, e.g. devices going out of range). @arg A2DP_EVT_MEDIA_PACKET_SENT: The local device has successfully sent a media packet to the remote device. @arg A2DP_EVT_MEDIA_PACKET_SEND_FAILED: The local device was not able to send a media packet to the remote device.

@arg A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED This event is generated when a local device completed "open and start" request.

param mgr AVDTP manager. param callback The callback function that will be called when the AVDTP generates an event. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

bt_a2dp_register_mpeg_codec Function

File

a2dp.h

C

```
void bt_a2dp_register_mpeg_codec(bt_a2dp_mgr_t* mgr);
```

Description

brief Register default MPEG codec ingroup a2dp

details This function adds MPEG codec implemented by dotstack to the list of known codecs. For more information about codecs see description of ::bt_avdtp_register_codec. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use MPEG-1,2 codec it must call this function when it is initializing.

note dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration.

param mgr A2DP manager.

bt_a2dp_sbc_codec_handler Function

File

a2dp_codec_sbc.h

C

```
bt_byte bt_a2dp_sbc_codec_handler(bt_avdtp_codec_t* codec, bt_byte opcode, bt_avdtp_codec_op_param_t*
op_param, bt_avdtp_mgr_t* mgr);
```

Description

This is function bt_a2dp_sbc_codec_handler.

bt_a2dp_start Function

File

a2dp.h

C

```
bt_bool bt_a2dp_start(bt_a2dp_mgr_t* mgr);
```

Description

brief Start the A2DP layer. ingroup a2dp

details This function makes the A2DP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required.

param mgr AVDTP manager.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_a2dp_set_configuration Function

File

a2dp.h

C

```
bt_bool bt_a2dp_set_configuration(bt_a2dp_mgr_t* mgr, bt_byte strm_handle, bt_bdaddr_t* remote_addr,
bt_byte seid_int, bt_byte seid_acp, const bt_avdtp_sep_capabilities_t* caps);
```

Description

This is function bt_a2dp_set_configuration.

A2DP Data Types and Constants

_bt_a2dp_event_u Union

File

a2dp.h

C

```
union _bt_a2dp_event_u {
  bt_a2dp_evt_open_and_start_stream_completed_t open_and_start_stream_completed;
  bt_avdtp_evt_ctrl_channel_connected_t ctrl_channel_connected;
  bt_avdtp_evt_ctrl_connection_failed_t ctrl_connection_failed;
  bt_avdtp_evt_ctrl_channel_disconnected_t ctrl_channel_disconnected;
  bt_avdtp_evt_discover_completed_t discover_completed;
  bt_avdtp_evt_sep_info_received_t sep_info_received;
  bt_avdtp_evt_get_sep_capabilities_completed_t get_sep_capabilities_completed;
  bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received;
  bt_avdtp_evt_set_stream_configuration_completed_t set_stream_configuration_completed;
  bt_avdtp_evt_get_stream_configuration_completed_t get_stream_configuration_completed;
  bt_avdtp_evt_stream_reconfigure_completed_t stream_reconfigure_completed;
  bt_avdtp_evt_open_stream_completed_t open_stream_completed;
  bt_avdtp_evt_start_stream_completed_t start_stream_completed;
  bt_avdtp_evt_close_stream_completed_t close_stream_completed;
  bt_avdtp_evt_suspend_stream_completed_t suspend_stream_completed;
  bt_avdtp_evt_stream_security_control_completed_t security_control_completed;
  bt_avdtp_evt_set_stream_configuration_requested_t set_stream_configuration_requested;
  bt_avdtp_evt_reconfigure_stream_requested_t reconfigure_stream_requested;
  bt_avdtp_evt_open_stream_requested_t open_stream_requested;
  bt_avdtp_evt_start_stream_requested_t start_stream_requested;
  bt_avdtp_evt_suspend_stream_requested_t suspend_stream_requested;
  bt_avdtp_evt_close_stream_requested_t close_stream_requested;
  bt_avdtp_evt_abort_stream_requested_t abort_stream_requested;
  bt_avdtp_evt_set_stream_configuration_t set_stream_configuration;
  bt_avdtp_evt_stream_configured_t stream_configured;
  bt_avdtp_evt_stream_reconfigured_t stream_reconfigured;
  bt_avdtp_evt_stream_opened_t stream_opened;
  bt_avdtp_evt_stream_started_t stream_started;
  bt_avdtp_evt_stream_suspended_t stream_suspended;
  bt_avdtp_evt_stream_closed_t stream_closed;
  bt_avdtp_evt_stream_aborted_t stream_aborted;
  bt_avdtp_evt_media_packet_received_t media_packet_received;
  bt_avdtp_evt_media_packet_sent_t media_packet_sent;
  bt_avdtp_evt_media_packet_send_failed_t media_packet_send_failed;
};
```

Members

Members	Description
bt_a2dp_evt_open_and_start_stream_completed_t open_and_start_stream_completed;	< Valid if event is A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED
bt_avdtp_evt_ctrl_channel_connected_t ctrl_channel_connected;	< Valid if event is A2DP_EVT_CTRL_CHANNEL_CONNECTED

bt_avdtp_evt_ctrl_connection_failed_t ctrl_connection_failed;	< Valid if event is A2DP_EVT_CTRL_CONNECTION_FAILED
bt_avdtp_evt_ctrl_channel_disconnected_t ctrl_channel_disconnected;	< Valid if event is A2DP_EVT_CTRL_CHANNEL_DISCONNECTED
bt_avdtp_evt_discover_completed_t discover_completed;	< Valid if event is A2DP_EVT_DISCOVER_COMPLETED
bt_avdtp_evt_sep_info_received_t sep_info_received;	< Valid if event is A2DP_EVT_SEP_INFO_RECEIVED
bt_avdtp_evt_get_sep_capabilities_completed_t get_sep_capabilities_completed;	< Valid if event is A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED
bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received;	< Valid if event is A2DP_EVT_SEP_CAPABILITIES_RECEIVED
bt_avdtp_evt_set_stream_configuration_completed_t set_stream_configuration_completed;	< Valid if event is A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED
bt_avdtp_evt_get_stream_configuration_completed_t get_stream_configuration_completed;	< Valid if event is A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED
bt_avdtp_evt_stream_reconfigure_completed_t stream_reconfigure_completed;	< Valid if event is A2DP_EVT_STREAM_RECONFIGURE_COMPLETED
bt_avdtp_evt_open_stream_completed_t open_stream_completed;	< Valid if event is A2DP_EVT_OPEN_STREAM_COMPLETED
bt_avdtp_evt_start_stream_completed_t start_stream_completed;	< Valid if event is A2DP_EVT_START_STREAM_COMPLETED
bt_avdtp_evt_close_stream_completed_t close_stream_completed;	< Valid if event is A2DP_EVT_CLOSE_STREAM_COMPLETED
bt_avdtp_evt_suspend_stream_completed_t suspend_stream_completed;	< Valid if event is A2DP_EVT_SUSPEND_STREAM_COMPLETED
bt_avdtp_evt_stream_security_control_completed_t security_control_completed;	< Valid if event is A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED
bt_avdtp_evt_set_stream_configuration_requested_t set_stream_configuration_requested;	< Valid if event is A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED
bt_avdtp_evt_reconfigure_stream_requested_t reconfigure_stream_requested;	< Valid if event is A2DP_EVT_RECONFIGURE_STREAM_REQUESTED
bt_avdtp_evt_open_stream_requested_t open_stream_requested;	< Valid if event is A2DP_EVT_OPEN_STREAM_REQUESTED
bt_avdtp_evt_start_stream_requested_t start_stream_requested;	< Valid if event is A2DP_EVT_START_STREAM_REQUESTED
bt_avdtp_evt_suspend_stream_requested_t suspend_stream_requested;	< Valid if event is A2DP_EVT_SUSPEND_STREAM_REQUESTED
bt_avdtp_evt_close_stream_requested_t close_stream_requested;	< Valid if event is A2DP_EVT_CLOSE_STREAM_REQUESTED
bt_avdtp_evt_abort_stream_requested_t abort_stream_requested;	< Valid if event is A2DP_EVT_ABORT_STREAM_REQUESTED
bt_avdtp_evt_stream_configured_t stream_configured;	< Valid if event is A2DP_EVT_STREAM_CONFIGURED
bt_avdtp_evt_stream_reconfigured_t stream_reconfigured;	< Valid if event is A2DP_EVT_STREAM_RECONFIGURED
bt_avdtp_evt_stream_opened_t stream_opened;	< Valid if event is A2DP_EVT_STREAM_OPENED
bt_avdtp_evt_stream_started_t stream_started;	< Valid if event is A2DP_EVT_STREAM_STARTED
bt_avdtp_evt_stream_suspended_t stream_suspended;	< Valid if event is A2DP_EVT_STREAM_SUSPENDED
bt_avdtp_evt_stream_closed_t stream_closed;	< Valid if event is A2DP_EVT_STREAM_CLOSED
bt_avdtp_evt_stream_aborted_t stream_aborted;	< Valid if event is A2DP_EVT_STREAM_ABORTED
bt_avdtp_evt_media_packet_received_t media_packet_received;	< Valid if event is A2DP_EVT_MEDIA_PACKET_RECEIVED
bt_avdtp_evt_media_packet_sent_t media_packet_sent;	< Valid if event is A2DP_EVT_MEDIA_PACKET_SENT
bt_avdtp_evt_media_packet_send_failed_t media_packet_send_failed;	< Valid if event is A2DP_EVT_MEDIA_PACKET_SEND_FAILED

Description

brief Parameter to an application callback. ingroup a2dp

details This union is used to pass event specific data to the A2DP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

_bt_a2dp_mgr_t Structure

File

a2dp.h

C

```
struct _bt_a2dp_mgr_t {
  bt_byte state;
  bt_avdtp_mgr_t* avdtp_mgr;
  bt_avdtp_stream_t* connection_stream;
  bt_a2dp_mgr_callback_fp callback;
  void* callback_param;
};
```

Members

Members	Description
bt_byte state;	< Manager state. This value can be one of the following values: < li A2DP_MANAGER_STATE_IDLE < li A2DP_MANAGER_STATE_CONNECTING
bt_avdtp_mgr_t* avdtp_mgr;	< AVDTP manager.
bt_avdtp_stream_t* connection_stream;	< Pointer to a stream being open with ::bt_a2dp_open_and_start_stream.
bt_a2dp_mgr_callback_fp callback;	< Pointer to a function which a A2DP consumer must register in order to be notified of various events.
void* callback_param;	< Pointer to arbitrary data to be passed to the c callback.

Description

brief A2DP manager. ingroup a2dp

details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with ::bt_a2dp_get_mgr.

bt_a2dp_aac_config_t Structure

File

a2dp_codec_aac.h

C

```
typedef struct _bt_a2dp_aac_config_t {
  bt_byte object_type;
  bt_uint sampling_frequency;
  bt_byte channels;
  bt_byte rfa;
  bt_byte vbr;
  bt_ulong bit_rate;
} bt_a2dp_aac_config_t;
```

Description

This is type bt_a2dp_aac_config_t.

bt_a2dp_event_t Type

File

a2dp.h

C

```
typedef union _bt_a2dp_event_u bt_a2dp_event_t;
```

Description

This is type bt_a2dp_event_t.

bt_a2dp_evt_open_and_start_stream_completed_t Structure

File

a2dp.h

C

```
typedef struct _bt_a2dp_evt_open_and_start_stream_completed_s {
  bt_byte failed_cmd;
  bt_byte err_code;
  bt_byte strm_handle;
  union {
    bt_avdtp_evt_set_stream_configuration_completed_t set_config;
    bt_avdtp_evt_open_stream_completed_t open;
    bt_avdtp_evt_start_stream_completed_t start;
  } evt_params;
} bt_a2dp_evt_open_and_start_stream_completed_t;
```

Members

Members	Description
bt_byte failed_cmd;	< ID of the failed command. This can be one of the following values: < li AVDTP_CMD_SET_CONFIGURATION < li AVDTP_CMD_OPEN < li AVDTP_CMD_START
bt_byte err_code;	< The result of the request. < li If the remote accepted all 3 requests sent during execution of the "open & start" request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_byte strm_handle;	< Stream handle.
bt_avdtp_evt_set_stream_configuration_completed_t set_config;	< If "set configuration" request failed this member contains description of an error returned by the remote party.
bt_avdtp_evt_open_stream_completed_t open;	< If open stream" request failed this member contains description of an error returned by the remote party.
bt_avdtp_evt_start_stream_completed_t start;	< If "start stream" request failed this member contains description of an error returned by the remote party.

Description

brief Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event ingroup a2dp

details A pointer to this structure is passed to the A2DP application callback as a valid member of the bt_a2dp_event_t union - bt_a2dp_event_t::open_and_start_stream_completed - when A2DP completed a "open & start stream" request.

bt_a2dp_find_server_callback_fp Type

File

a2dp.h

C

```
typedef void (* bt_a2dp_find_server_callback_fp)(bt_uint supported_features, bt_bool found, void* param);
```

Description

brief Notify the application of the result of searching for a remote A2DP entity (sourse or sink) ingroup a2dp

details This function is called by the A2DP layer when searching for an A2DP entity on a remote device has completed.

param supported_features Features supported by a remote A2DP entity. param found c TRUE if an A2DP entity has been found on the remote device. c FALSE otherwise. param param pointer to arbitrary data passed to the bt_a2dp_find_source() or bt_a2dp_find_sink function through its c callback_param parameter.

bt_a2dp_mgr_callback_fp Type

File

a2dp.h

C

```
typedef void (* bt_a2dp_mgr_callback_fp)(bt_a2dp_mgr_t* mgr, bt_byte evt, bt_a2dp_event_t* evt_param, void*
callback_param);
```

Description

brief A2DP application callback. ingroup a2dp

details In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call that function whenever a new event has been generated.

param mgr A2DP manager.

param evt A2DP event. The event can be one of the following values: @arg A2DP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg A2DP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg A2DP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local device received a response to Get SEP capabilities operation. @arg A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Set stream configuration operation. @arg A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Get stream configuration operation. @arg A2DP_EVT_STREAM_RECONFIGURE_COMPLETED: Local device received a response to Reconfigure stream operation. @arg A2DP_EVT_OPEN_STREAM_COMPLETED: Local device received a response to Open stream operation. @arg A2DP_EVT_START_STREAM_COMPLETED: Local device received a response to Start stream operation. @arg A2DP_EVT_CLOSE_STREAM_COMPLETED: Local device received a response to Close stream operation. @arg A2DP_EVT_SUSPEND_STREAM_COMPLETED: Local device received a response to Suspend stream operation. @arg A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: Local device received a response to Stream security control operation. @arg A2DP_EVT_ABORT_STREAM_COMPLETED: Local device received a response to Abort stream operation. @arg A2DP_EVT_SEP_INFO_RECEIVED: SEP information received. @arg A2DP_EVT_SEP_CAPABILITIES_RECEIVED: SEP capabilities received. @arg A2DP_EVT_STREAM_CONFIGURATION_RECEIVED: Stream configuration received.

@arg A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: Remote device requested stream configuration. @arg A2DP_EVT_OPEN_STREAM_REQUESTED: Remote device requested to open a stream. @arg A2DP_EVT_START_STREAM_REQUESTED: Remote device requested to start a stream. @arg A2DP_EVT_CLOSE_STREAM_REQUESTED: Remote device requested to close a stream. @arg A2DP_EVT_SUSPEND_STREAM_REQUESTED: Remote device requested to suspend a stream. @arg A2DP_EVT_ABORT_STREAM_REQUESTED: Remote device requested to abort a stream. @arg A2DP_EVT_RECONFIGURE_STREAM_REQUESTED: Remote device requested to reconfigure a stream. @arg A2DP_EVT_MEDIA_PACKET_RECEIVED: Remote device sent a media packet. @arg A2DP_EVT_STREAM_CONFIGURED: A stream has been configured (This event is generated right after A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the local devices accepted the request). @arg A2DP_EVT_STREAM_RECONFIGURED: A stream has been re-configured (This event is generated right after A2DP_EVT_RECONFIGURE_STREAM_REQUESTED if the local devices accepted the request). @arg A2DP_EVT_STREAM_OPENED: A stream has been opened (This event is generated as a result of local or remote stream opening request). @arg A2DP_EVT_STREAM_STARTED: A stream has been started (This event is generated right after A2DP_EVT_START_STREAM_REQUESTED if the local devices accepted the request). @arg A2DP_EVT_STREAM_CLOSED: A stream has been close (This event is generated right after A2DP_EVT_CLOSE_STREAM_REQUESTED if the local devices accepted the request). @arg A2DP_EVT_STREAM_SUSPENDED: A stream has been suspended (This event is generated right after A2DP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request). @arg A2DP_EVT_STREAM_ABORTED: A stream has been aborted (This event is generated right after A2DP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request. It is also generated if connection between devices has been terminated by means other than AVDTP signaling, e.g. devices going out of rage). @arg A2DP_EVT_MEDIA_PACKET_SENT: The local device has successfully sent a media packet to the remote device. @arg A2DP_EVT_MEDIA_PACKET_SEND_FAILED: The local device was not able to send a media packet to the remote device.

@arg A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED Local device completed "open and start" request.

param evt_param Event parameter. Which member of the bt_a2dp_event_t union is valid depends on the event: @arg A2DP_EVT_CTRL_CHANNEL_CONNECTED: c bt_avdtp_evt_ctrl_channel_connected_t ctrl_channel_connected @arg A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: c bt_avdtp_evt_ctrl_channel_disconnected_t ctrl_channel_disconnected @arg A2DP_EVT_CTRL_CONNECTION_FAILED: c NULL @arg A2DP_EVT_DISCOVER_COMPLETED: c bt_avdtp_evt_discover_completed_t discover_completed; @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: c bt_avdtp_evt_get_sep_capabilities_completed_t get_sep_capabilities_completed @arg A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: c bt_avdtp_evt_set_stream_configuration_completed_t set_stream_configuration_completed @arg A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: c bt_avdtp_evt_get_stream_configuration_completed_t get_stream_configuration_completed @arg A2DP_EVT_STREAM_RECONFIGURE_COMPLETED: c bt_avdtp_evt_stream_reconfigure_completed_t stream_reconfigure_completed @arg A2DP_EVT_OPEN_STREAM_COMPLETED: c bt_avdtp_evt_open_stream_completed_t open_stream_completed @arg A2DP_EVT_START_STREAM_COMPLETED: c bt_avdtp_evt_start_stream_completed_t start_stream_completed @arg A2DP_EVT_CLOSE_STREAM_COMPLETED: c bt_avdtp_evt_close_stream_completed_t close_stream_completed @arg A2DP_EVT_SUSPEND_STREAM_COMPLETED: c bt_avdtp_evt_suspend_stream_completed_t suspend_stream_completed @arg A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: c bt_avdtp_evt_stream_security_control_completed_t security_control_completed @arg A2DP_EVT_ABORT_STREAM_COMPLETED: c bt_avdtp_evt_abort_stream_requested_t abort_stream_requested @arg A2DP_EVT_SEP_INFO_RECEIVED: c bt_avdtp_evt_sep_info_received_t sep_info_received @arg A2DP_EVT_SEP_CAPABILITIES_RECEIVED: c bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received @arg A2DP_EVT_STREAM_CONFIGURATION_RECEIVED: c bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received

@arg A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: c bt_avdtp_evt_set_stream_configuration_requested_t set_stream_configuration_requested @arg A2DP_EVT_OPEN_STREAM_REQUESTED: c bt_avdtp_evt_open_stream_requested_t open_stream_requested @arg A2DP_EVT_START_STREAM_REQUESTED: c bt_avdtp_evt_start_stream_requested_t start_stream_requested @arg A2DP_EVT_CLOSE_STREAM_REQUESTED: c bt_avdtp_evt_close_stream_requested_t close_stream_requested @arg

A2DP_EVT_SUSPEND_STREAM_REQUESTED: c bt_avdtp_evt_suspend_stream_requested_t suspend_stream_requested @arg A2DP_EVT_ABORT_STREAM_REQUESTED: c bt_avdtp_evt_abort_stream_requested_t abort_stream_requested @arg A2DP_EVT_RECONFIGURE_STREAM_REQUESTED: c bt_avdtp_evt_reconfigure_stream_requested_t reconfigure_stream_requested @arg A2DP_EVT_MEDIA_PACKET_RECEIVED: c bt_avdtp_evt_media_packet_received_t media_packet_received @arg A2DP_EVT_STREAM_CONFIGURED: c bt_avdtp_evt_stream_configured_t stream_configured @arg A2DP_EVT_STREAM_RECONFIGURED: c bt_avdtp_evt_stream_reconfigured_t stream_reconfigured @arg A2DP_EVT_STREAM_OPENED: c bt_avdtp_evt_stream_opened_t stream_opened @arg A2DP_EVT_STREAM_STARTED: c bt_avdtp_evt_stream_started_t stream_started @arg A2DP_EVT_STREAM_CLOSED: c bt_avdtp_evt_stream_closed_t stream_closed @arg A2DP_EVT_STREAM_SUSPENDED: c bt_avdtp_evt_stream_suspended_t stream_suspended @arg A2DP_EVT_STREAM_ABORTED: c bt_avdtp_evt_stream_aborted_t stream_aborted @arg A2DP_EVT_MEDIA_PACKET_SENT: c bt_avdtp_evt_media_packet_sent_t media_packet_sent @arg A2DP_EVT_MEDIA_PACKET_SEND_FAILED: c bt_avdtp_evt_media_packet_send_failed_t media_packet_send_failed

@arg A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED c bt_a2dp_evt_open_and_start_stream_completed_t open_and_start_stream_completed

param callback_param A pointer to an arbitrary data set by a call to bt_avdtp_register_callback.

bt_a2dp_mgr_t Type

File

a2dp.h

C

```
typedef struct _bt_a2dp_mgr_t bt_a2dp_mgr_t;
```

Description

This is type bt_a2dp_mgr_t.

bt_a2dp_mpeg_config_t Structure

File

a2dp_codec_mpeg.h

C

```
typedef struct _bt_a2dp_mpeg_config_t {
  bt_byte layer;
  bt_byte crc;
  bt_byte channel_mode;
  bt_byte rfa;
  bt_byte mpf;
  bt_byte sampling_frequency;
  bt_byte vbr;
  bt_uint bit_rate;
} bt_a2dp_mpeg_config_t;
```

Description

This is type bt_a2dp_mpeg_config_t.

bt_a2dp_sbc_config_t Structure

File

a2dp_codec_sbc.h

C

```
typedef struct _bt_a2dp_sbc_config_t {
  bt_byte sampling_frequency;
  bt_byte channel_mode;
  bt_byte block_length;
  bt_byte subbands;
  bt_byte allocation_method;
  bt_byte min_bitpool_val;
  bt_byte max_bitpool_val;
} bt_a2dp_sbc_config_t;
```

Description

This is type bt_a2dp_sbc_config_t.

bt_a2dp_sbc_packet_info_t Structure

File

a2dp_codec_sbc.h

C

```
typedef struct _bt_a2dp_sbc_packet_info_t {
  bt_byte frames;
  bt_byte l;
  bt_byte s;
  bt_byte f;
  bt_uint frame_size;
} bt_a2dp_sbc_packet_info_t;
```

Description

This is type bt_a2dp_sbc_packet_info_t.

__A2DP_CODEC_AAC_H Macro

File

a2dp_codec_aac.h

C

```
#define __A2DP_CODEC_AAC_H
```

Description

This is macro __A2DP_CODEC_AAC_H.

__A2DP_CODEC_MPEG_H Macro

File

a2dp_codec_mpeg.h

C

```
#define __A2DP_CODEC_MPEG_H
```

Description

This is macro __A2DP_CODEC_MPEG_H.

__A2DP_CODEC_SBC_H Macro

File

a2dp_codec_sbc.h

C

```
#define __A2DP_CODEC_SBC_H
```

Description

This is macro __A2DP_CODEC_SBC_H.

__A2DP_H Macro

File

a2dp.h

C

```
#define __A2DP_H
```

Description

This is macro __A2DP_H.

__A2DP_PRIVATE_H Macro

File

a2dp_private.h

C

```
#define __A2DP_PRIVATE_H
```

Description

This is macro __A2DP_PRIVATE_H.

A2DP_EVT_ABORT_STREAM_COMPLETED Macro

File

a2dp.h

C

```
#define A2DP_EVT_ABORT_STREAM_COMPLETED AVDTP_EVT_ABORT_STREAM_COMPLETED                ///< This event is
generated when a local device received a response (either positive or negative) to a "abort stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.

A2DP_EVT_ABORT_STREAM_REQUESTED Macro

File

a2dp.h

C

```
#define A2DP_EVT_ABORT_STREAM_REQUESTED AVDTP_EVT_ABORT_STREAM_REQUESTED                ///< This event is
generated when a local device received "abort stream" request.
```

Description

< This event is generated when a local device received "abort stream" request.

A2DP_EVT_CLOSE_STREAM_COMPLETED Macro

File

a2dp.h

C

```
#define A2DP_EVT_CLOSE_STREAM_COMPLETED AVDTP_EVT_CLOSE_STREAM_COMPLETED                ///< This event is
generated when a local device received a response (either positive or negative) to a "close stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "close stream" request.

A2DP_EVT_CLOSE_STREAM_REQUESTED Macro

File

a2dp.h

C

```
#define A2DP_EVT_CLOSE_STREAM_REQUESTED AVDTP_EVT_CLOSE_STREAM_REQUESTED                ///< This event is
generated when a local device received "close stream" request.
```

Description

< This event is generated when a local device received "close stream" request.

A2DP_EVT_CTRL_CHANNEL_CONNECTED Macro

File

a2dp.h

C

```
#define A2DP_EVT_CTRL_CHANNEL_CONNECTED AVDTP_EVT_CTRL_CHANNEL_CONNECTED                ///< This event is
generated when a control channel between two AVDTP entities has been established.
```

Description

< This event is generated when a control channel between two AVDTP entities has been established.

A2DP_EVT_CTRL_CHANNEL_DISCONNECTED Macro

File

a2dp.h

C

```
#define A2DP_EVT_CTRL_CHANNEL_DISCONNECTED AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED            ///< This event
is generated when a control channel between two AVDTP entities has been terminated.
```

Description

< This event is generated when a control channel between two AVDTP entities has been terminated.

A2DP_EVT_CTRL_CONNECTION_FAILED Macro

File

a2dp.h

C

```
#define A2DP_EVT_CTRL_CONNECTION_FAILED AVDTP_EVT_CTRL_CONNECTION_FAILED                ///< This event is
generated when a local device failed to create a control channel between two AVDTP entities.
```

Description

< This event is generated when a local device failed to create a control channel between two AVDTP entities.

A2DP_EVT_DISCOVER_SEP_COMPLETED Macro

File

a2dp.h

C

```
#define A2DP_EVT_DISCOVER_SEP_COMPLETED AVDTP_EVT_DISCOVER_COMPLETED                    ///< This event is
generated when a local device received a response (either positive or negative) to a "discover" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "discover" request.

A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED Macro

File

a2dp.h

C

```
#define A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED        ///< This
event is generated when a local device received a response (either positive or negative) to a "get SEP
```

```
capabilities" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.

A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED Macro

File

a2dp.h

C

```
#define A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED    ///<
This event is generated when a local device received a response (either positive or negative) to a "get
stream configuration" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.

A2DP_EVT_MEDIA_PACKET_RECEIVED Macro

File

a2dp.h

C

```
#define A2DP_EVT_MEDIA_PACKET_RECEIVED AVDTP_EVT_MEDIA_PACKET_RECEIVED                ///< This event is
generated when a local device received a media packet.
```

Description

< This event is generated when a local device received a media packet.

A2DP_EVT_MEDIA_PACKET_SEND_FAILED Macro

File

a2dp.h

C

```
#define A2DP_EVT_MEDIA_PACKET_SEND_FAILED AVDTP_EVT_MEDIA_PACKET_SEND_FAILED             ///< This event is
generated when a local device failed to send a media packet.
```

Description

< This event is generated when a local device failed to send a media packet.

A2DP_EVT_MEDIA_PACKET_SENT Macro

File

a2dp.h

C

```
#define A2DP_EVT_MEDIA_PACKET_SENT AVDTP_EVT_MEDIA_PACKET_SENT                    ///< This event is
generated when a local device sent a media packet.
```

Description

< This event is generated when a local device sent a media packet.

A2DP_EVT_NOTHING Macro

File

a2dp.h

C

```
#define A2DP_EVT_NOTHING AVDTP_EVT_NULL
```

Description

This is macro A2DP_EVT_NOTHING.

A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED Macro

File

a2dp.h

C

```
#define A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED (AVDTP_EVT_LAST + 1)                    ///< This
event is generated when a local device completed "open and start" request.
```

Description

< This event is generated when a local device completed "open and start" request.

A2DP_EVT_OPEN_STREAM_COMPLETED Macro

File

a2dp.h

C

```
#define A2DP_EVT_OPEN_STREAM_COMPLETED AVDTP_EVT_OPEN_STREAM_COMPLETED                ///< This event is
generated when a local device received a response (either positive or negative) to a "open stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "open stream" request.

A2DP_EVT_OPEN_STREAM_REQUESTED Macro

File

a2dp.h

C

```
#define A2DP_EVT_OPEN_STREAM_REQUESTED AVDTP_EVT_OPEN_STREAM_REQUESTED                ///< This event is
generated when a local device received "open stream" request.
```

Description

< This event is generated when a local device received "open stream" request.

A2DP_EVT_RECONFIGURE_STREAM_COMPLETED Macro

File

a2dp.h

C

```
#define A2DP_EVT_RECONFIGURE_STREAM_COMPLETED AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED            ///< This
event is generated when a local device received a response (either positive or negative) to a "change
stream configuration" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.

A2DP_EVT_RECONFIGURE_STREAM_REQUESTED Macro

File

a2dp.h

C

```
#define A2DP_EVT_RECONFIGURE_STREAM_REQUESTED AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED            ///< This
event is generated when a local device received "change stream configuration" request.
```

Description

< This event is generated when a local device received "change stream configuration" request.

A2DP_EVT_SEP_CAPABILITIES_RECEIVED Macro

File

a2dp.h

C

```
#define A2DP_EVT_SEP_CAPABILITIES_RECEIVED AVDTP_EVT_SEP_CAPABILITIES_RECEIVED            ///< This event
is generated when a local device received a positive response to a "get SEP capabilities" request.
```

Description

< This event is generated when a local device received a positive response to a "get SEP capabilities" request.

A2DP_EVT_SEP_INFO_RECEIVED Macro

File

a2dp.h

C

```
#define A2DP_EVT_SEP_INFO_RECEIVED AVDTP_EVT_SEP_INFO_RECEIVED                    ///< This event is
generated for each SEP contained in a positive response to a "discover" request.
```

Description

< This event is generated for each SEP contained in a positive response to a "discover" request.

A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED Macro

File

a2dp.h

C

```
#define A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED    ///<
This event is generated when a local device received a response (either positive or negative) to a "set
stream configuration" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.

A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED Macro

File

a2dp.h

C

```
#define A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED   ///<
This event is generated when a local device received "set stream configuration" request.
```

Description

< This event is generated when a local device received "set stream configuration" request.

A2DP_EVT_START_STREAM_COMPLETED Macro

File

a2dp.h

C

```
#define A2DP_EVT_START_STREAM_COMPLETED AVDTP_EVT_START_STREAM_COMPLETED                ///< This event is
generated when a local device received a response (either positive or negative) to a "start stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "start stream" request.

A2DP_EVT_START_STREAM_REQUESTED Macro

File

a2dp.h

C

```
#define A2DP_EVT_START_STREAM_REQUESTED AVDTP_EVT_START_STREAM_REQUESTED                ///< This event is
generated when a local device received "start stream" request.
```

Description

< This event is generated when a local device received "start stream" request.

A2DP_EVT_STREAM_ABORTED Macro

File

a2dp.h

C

```
#define A2DP_EVT_STREAM_ABORTED AVDTP_EVT_STREAM_ABORTED                        ///< This event is generated
when a local device has successfully aborted a stream.
```

Description

< This event is generated when a local device has successfully aborted a stream. < This event follows the A2DP_EVT_ABORT_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream abortion was initiated by the local device.

A2DP_EVT_STREAM_CLOSED Macro

File

a2dp.h

C

```
#define A2DP_EVT_STREAM_CLOSED AVDTP_EVT_STREAM_CLOSED                        ///< This event is generated
when a local device has successfully closed a stream.
```

Description

< This event is generated when a local device has successfully closed a stream. < This event follows the A2DP_EVT_CLOSE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream closing was initiated by the local device.

A2DP_EVT_STREAM_CONFIGURATION_RECEIVED Macro

File

a2dp.h

C

```
#define A2DP_EVT_STREAM_CONFIGURATION_RECEIVED AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED        ///< This
```

event is generated when a local device received a positive response to a "get stream configuration" request.

Description

< This event is generated when a local device received a positive response to a "get stream configuration" request.

A2DP_EVT_STREAM_CONFIGURED Macro

File

a2dp.h

C

```
#define A2DP_EVT_STREAM_CONFIGURED AVDTP_EVT_STREAM_CONFIGURED                    ///< This event is
generated when a local device has successfully configured a stream.
```

Description

< This event is generated when a local device has successfully configured a stream. < This event follows the A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the upper layer has accepted it. < This event is not generated if stream configuration was initiated by the local device.

A2DP_EVT_STREAM_OPENED Macro

File

a2dp.h

C

```
#define A2DP_EVT_STREAM_OPENED AVDTP_EVT_STREAM_OPENED                        ///< This event is generated
when a local device has successfully opened a stream.
```

Description

< This event is generated when a local device has successfully opened a stream. < This event follows the A2DP_EVT_OPEN_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream opening was initiated by the local device.

A2DP_EVT_STREAM_RECONFIGURED Macro

File

a2dp.h

C

```
#define A2DP_EVT_STREAM_RECONFIGURED AVDTP_EVT_STREAM_RECONFIGURED                ///< This event is
generated when a local device has successfully reconfigured a stream.
```

Description

< This event is generated when a local device has successfully reconfigured a stream. < This event follows the A2DP_EVT_RECONFIGURE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream reconfiguration was initiated by the local device.

A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED Macro

File

a2dp.h

C

```
#define A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED    ///< This
event is generated when a local device received a response (either positive or negative) to a "exchange
content protection control data" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.

A2DP_EVT_STREAM_STARTED Macro

File

a2dp.h

C

```
#define A2DP_EVT_STREAM_STARTED AVDTP_EVT_STREAM_STARTED ///< This event is generated
when a local device has successfully started a stream.
```

Description

< This event is generated when a local device has successfully started a stream. < This event follows the A2DP_EVT_START_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream starting was initiated by the local device.

A2DP_EVT_STREAM_SUSPENDED Macro

File

a2dp.h

C

```
#define A2DP_EVT_STREAM_SUSPENDED AVDTP_EVT_STREAM_SUSPENDED ///< This event is
generated when a local device has successfully suspended a stream.
```

Description

< This event is generated when a local device has successfully suspended a stream. < This event follows the A2DP_EVT_SUSPEND_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream suspension was initiated by the local device.

A2DP_EVT_SUSPEND_STREAM_COMPLETED Macro

File

a2dp.h

C

```
#define A2DP_EVT_SUSPEND_STREAM_COMPLETED AVDTP_EVT_SUSPEND_STREAM_COMPLETED ///< This event
is generated when a local device received a response (either positive or negative) to a "suspend stream"
request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.

A2DP_EVT_SUSPEND_STREAM_REQUESTED Macro

File

a2dp.h

C

```
#define A2DP_EVT_SUSPEND_STREAM_REQUESTED AVDTP_EVT_SUSPEND_STREAM_REQUESTED ///< This event
is generated when a local device received "suspend stream" request.
```

Description

< This event is generated when a local device received "suspend stream" request.

A2DP_MANAGER_STATE_CONNECTING Macro

File

a2dp.h

C

```
#define A2DP_MANAGER_STATE_CONNECTING 1
```

Description

This is macro A2DP_MANAGER_STATE_CONNECTING.

A2DP_MANAGER_STATE_IDLE Macro

File

a2dp.h

C

```
#define A2DP_MANAGER_STATE_IDLE 0
```

Description

This is macro A2DP_MANAGER_STATE_IDLE.

A2DP_SINK_FEATURE_AMPLIFIER Macro

File

a2dp.h

C

```
#define A2DP_SINK_FEATURE_AMPLIFIER 8    ///< Amplifier
```

Description

< Amplifier

A2DP_SINK_FEATURE_HEADPHONE Macro

File

a2dp.h

C

```
#define A2DP_SINK_FEATURE_HEADPHONE 1    ///< Headphone
```

Description

< Headphone

A2DP_SINK_FEATURE_RECORDER Macro

File

a2dp.h

C

```
#define A2DP_SINK_FEATURE_RECORDER 4    ///< Recorder
```

Description

< Recorder

A2DP_SINK_FEATURE_SPEAKER Macro

File

a2dp.h

C

```
#define A2DP_SINK_FEATURE_SPEAKER 2    ///< Speaker
```

Description

< Speaker

A2DP_SOURCE_FEATURE_MICROPHONE Macro

File

a2dp.h

C

```
#define A2DP_SOURCE_FEATURE_MICROPHONE 2    ///< Mic
```

Description

< Mic

A2DP_SOURCE_FEATURE_MIXER Macro

File

a2dp.h

C

```
#define A2DP_SOURCE_FEATURE_MIXER 8    ///< Mixer
```

Description

< Mixer

A2DP_SOURCE_FEATURE_PLAYER Macro

File

a2dp.h

C

```
#define A2DP_SOURCE_FEATURE_PLAYER 1    ///< Player
```

Description

< Player

A2DP_SOURCE_FEATURE_TUNER Macro

File

a2dp.h

C

```
#define A2DP_SOURCE_FEATURE_TUNER 4    ///< Tuner
```

Description

< Tuner

AAC_CHANNELS_1 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_CHANNELS_1 0x2
```

Description

This is macro AAC_CHANNELS_1.

AAC_CHANNELS_2 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_CHANNELS_2 0x1
```

Description

This is macro AAC_CHANNELS_2.

AAC_CHANNELS_ALL Macro

File

a2dp_codec_aac.h

C

```
#define AAC_CHANNELS_ALL 0x3
```

Description

This is macro AAC_CHANNELS_ALL.

AAC_OBJECT_TYPE_MPEG_2_LC Macro

File

a2dp_codec_aac.h

C

```
#define AAC_OBJECT_TYPE_MPEG_2_LC 0x80
```

Description

This is macro AAC_OBJECT_TYPE_MPEG_2_LC.

AAC_OBJECT_TYPE_MPEG_4_LC Macro

File

a2dp_codec_aac.h

C

```
#define AAC_OBJECT_TYPE_MPEG_4_LC 0x40
```

Description

This is macro AAC_OBJECT_TYPE_MPEG_4_LC.

AAC_OBJECT_TYPE_MPEG_4_LTP Macro

File

a2dp_codec_aac.h

C

```
#define AAC_OBJECT_TYPE_MPEG_4_LTP 0x20
```

Description

This is macro AAC_OBJECT_TYPE_MPEG_4_LTP.

AAC_OBJECT_TYPE_MPEG_4_SCALABLE Macro

File

a2dp_codec_aac.h

C

```
#define AAC_OBJECT_TYPE_MPEG_4_SCALABLE 0x10
```

Description

This is macro AAC_OBJECT_TYPE_MPEG_4_SCALABLE.

AAC_SAMPLING_FREQUENCY_11025 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_11025 0x040
```

Description

This is macro AAC_SAMPLING_FREQUENCY_11025.

AAC_SAMPLING_FREQUENCY_12000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_12000 0x020
```

Description

This is macro AAC_SAMPLING_FREQUENCY_12000.

AAC_SAMPLING_FREQUENCY_16000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_16000 0x010
```

Description

This is macro AAC_SAMPLING_FREQUENCY_16000.

AAC_SAMPLING_FREQUENCY_22050 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_22050 0x008
```

Description

This is macro AAC_SAMPLING_FREQUENCY_22050.

AAC_SAMPLING_FREQUENCY_24000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_24000 0x004
```

Description

This is macro AAC_SAMPLING_FREQUENCY_24000.

AAC_SAMPLING_FREQUENCY_32000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_32000 0x002
```

Description

This is macro AAC_SAMPLING_FREQUENCY_32000.

AAC_SAMPLING_FREQUENCY_44100 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_44100 0x001
```

Description

This is macro AAC_SAMPLING_FREQUENCY_44100.

AAC_SAMPLING_FREQUENCY_48000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_48000 0x800
```

Description

This is macro AAC_SAMPLING_FREQUENCY_48000.

AAC_SAMPLING_FREQUENCY_64000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_64000 0x400
```

Description

This is macro AAC_SAMPLING_FREQUENCY_64000.

AAC_SAMPLING_FREQUENCY_8000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_8000 0x080
```

Description

This is macro AAC_SAMPLING_FREQUENCY_8000.

AAC_SAMPLING_FREQUENCY_88200 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_88200 0x200
```

Description

This is macro AAC_SAMPLING_FREQUENCY_88200.

AAC_SAMPLING_FREQUENCY_96000 Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_96000 0x100
```

Description

This is macro AAC_SAMPLING_FREQUENCY_96000.

AAC_SAMPLING_FREQUENCY_ALL Macro

File

a2dp_codec_aac.h

C

```
#define AAC_SAMPLING_FREQUENCY_ALL 0xFFF
```

Description

This is macro AAC_SAMPLING_FREQUENCY_ALL.

AAC_VBR_NOT_SUPPORTED Macro

File

a2dp_codec_aac.h

C

```
#define AAC_VBR_NOT_SUPPORTED 0
```

Description

This is macro AAC_VBR_NOT_SUPPORTED.

AAC_VBR_SUPPORTED Macro

File

a2dp_codec_aac.h

C

```
#define AAC_VBR_SUPPORTED 1
```

Description

This is macro AAC_VBR_SUPPORTED.

bt_a2dp_abort_stream Macro

File

a2dp.h

C

```
#define bt_a2dp_abort_stream(mgr, strm_handle) bt_avdtp_abort_stream(mgr->avdtp_mgr, strm_handle)
```

Description

brief Suspend a stream. ingroup a2dp

details This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE. As a result of this operation the A2DP_EVT_ABORT_STREAM_COMPLETED event will be generated. This operation cannot be rejected. The p evt_param.abort_stream_requested.err_code is always == AVDTP_ERROR_SUCCESS.

param mgr A2DP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_a2dp_add_media_rx_buffer Macro

File

a2dp.h

C

```
#define bt_a2dp_add_media_rx_buffer(mgr, strm_handle, buffer) bt_avdtp_add_media_rx_buffer(mgr->avdtp_mgr,
strm_handle, buffer)
```

Description

brief Add a media packet buffer to a receive queue ingroup a2dp

details The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue or none of the buffers is large enough the received packets is dropped. Each buffer has a field (data_len) that holds the length of the received buffer. This field is never 0 if the buffer contains a packet. If a channel closed regardless of what has caused that and there are still buffers in the queue A2DP generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event for each buffer and sets the data_len to 0. This is to inform the A2DP consumer that the buffer has not been used and can be, for example, deallocated. This function adds a buffer to the receive queue.

param mgr A2DP manager. param strm_handle Stream handle. param buffer Pointer to a structure that holds the buffer and its parameters.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

bt_a2dp_add_media_tx_buffer Macro

File

a2dp.h

C

```
#define bt_a2dp_add_media_tx_buffer(mgr, strm_handle, buffer) bt_avdtp_add_media_tx_buffer(mgr->avdtp_mgr,
strm_handle, buffer)
```

Description

brief Add a media packet buffer to a send queue ingroup a2dp

details When the consumer of A2DP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. When the packet has been successfully sent a A2DP_EVT_MEDIA_PACKET_SENT is generated. Otherwise a A2DP_EVT_MEDIA_PACKET_SEND_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as A2DP has removed it from the queue and gave up control over it. As in the case of received buffers, if a channel closed regardless of what has caused that and there are still buffers in the queue A2DP generates a A2DP_EVT_MEDIA_PACKET_SENT event for each buffer and sets the data_len field to 0. This is to inform the AVDTP consumer that the buffer has not been used and can be, for example, deallocated.

param mgr A2DP manager. param strm_handle Stream handle. param buffer Pointer to a structure that holds the buffer and its parameters.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

bt_a2dp_call_codec Macro

File

a2dp.h

C

```
#define bt_a2dp_call_codec(codec, op, op_param, mgr) codec->codec_handler(codec, op, op_param,
mgr->avdtp_mgr)
```

Description

This is macro bt_a2dp_call_codec.

bt_a2dp_cancel_listen Macro

File

a2dp.h

C

```
#define bt_a2dp_cancel_listen(mgr, strm_handle, sep_id) bt_avdtp_cancel_listen(mgr->avdtp_mgr, strm_handle,
sep_id)
```

Description

brief Cancel listening for incoming connections. ingroup a2dp

details This function removes a SEP from a list of SEPS which a stream can use for incoming requests.

param mgr A2DP manager. param strm_handle Stream handle. param sep_id Local SEP ID.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_a2dp_close_stream Macro

File

a2dp.h

C

```
#define bt_a2dp_close_stream(mgr, strm_handle) bt_avdtp_close_stream(mgr->avdtp_mgr, strm_handle)
```

Description

brief Close a stream. ingroup a2dp

details This function tries to close a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN or AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the A2DP_EVT_CLOSE_STREAM_COMPLETED event will be generated. If the stream has been closed the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == the error code sent by the remote.

param mgr A2DP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_a2dp_connect Macro

File

a2dp.h

C

```
#define bt_a2dp_connect(mgr, remote_addr) bt_avdtp_connect(mgr->avdtp_mgr, remote_addr)
```

Description

brief Connect to a remote device. ingroup a2dp

details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP_EVT_CTRL_CHANNEL_CONNECTED or A2DP_EVT_CTRL_CHANNEL_CONNECTION_FAILED.

param mgr A2DP manager. param remote_addr The address of a remote device.

return li c TRUE if connection establishment has been started. li c FALSE otherwise.

bt_a2dp_connect_ex Macro

File

a2dp.h

C

```
#define bt_a2dp_connect_ex(mgr, remote_addr, acl_config) bt_avdtp_connect_ex(mgr->avdtp_mgr, remote_addr,
acl_config)
```

Description

brief Connect to a remote device. ingroup a2dp

details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP_EVT_CTRL_CHANNEL_CONNECTED or A2DP_EVT_CTRL_CHANNEL_CONNECTION_FAILED.

param mgr A2DP manager. param remote_addr The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following values: li HCI_CONFIG_ENABLE_AUTHENTICATION li HCI_CONFIG_ENABLE_ENCRYPTION li HCI_CONFIG_BECOME_MASTER

return li c TRUE if connection establishment has been started. li c FALSE otherwise.

bt_a2dp_create_stream Macro

File

a2dp.h

C

```
#define bt_a2dp_create_stream(mgr) bt_avdtp_create_stream(mgr->avdtp_mgr)
```

Description

brief Create a stream. ingroup a2dp

details This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort.

param mgr A2DP manager.

return li c Stream handle if the function succeeds. li c 0 otherwise.

bt_a2dp_destroy_stream Macro

File

a2dp.h

C

```
#define bt_a2dp_destroy_stream(mgr, strm_handle) bt_avdtp_destroy_stream(mgr->avdtp_mgr, strm_handle)
```

Description

brief Destroy a stream. ingroup a2dp

details This function frees memory used by the stream. The stream has to exist and be in the "idle" state for this function to succeed. I.e. the stream has to be closed or aborted before this function can be called.

param mgr A2DP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_a2dp_disconnect Macro

File

a2dp.h

C

```
#define bt_a2dp_disconnect(mgr, remote_addr) bt_avdtp_disconnect(mgr->avdtp_mgr, remote_addr)
```

Description

brief Disconnect from a remote device. ingroup a2dp

details This function closes a control and transport channels on all streams associated with the remote device specified by the p remote_addr. As a result of this operation the following events will be generated: @arg A2DP_EVT_MEDIA_PACKET_RECEIVED: if a stream's receive queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg A2DP_EVT_MEDIA_PACKET_SENT: if a stream's send queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg A2DP_EVT_STREAM_CLOSED: this event is generate if a stream is in "closing" state as a result of a request from the remote device or ::bt_a2dp_close_stream call before ::bt_a2dp_disconnect call. @arg A2DP_EVT_STREAM_ABORTED: this event is generated if a stream is in "active" state at the time of bt_avdtp_disconnect call.

param mgr A2DP manager. param remote_addr The address of a remote device.

return li c TRUE if disconnection has been started. li c FALSE otherwise. No events will be generated.

bt_a2dp_discover Macro

File

a2dp.h

C

```
#define bt_a2dp_discover(mgr, remote_addr) bt_avdtp_discover(mgr->avdtp_mgr, remote_addr)
```

Description

brief Discover SEPs on a remote device. ingroup a2dp

details This function asks the remote device to send a list of all available SEPs. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_INFO_RECEIVED: this event is generated for every SEP received from the remote device. the p evt_param.sep_info_received contains SEP information. @arg A2DP_EVT_DISCOVER_COMPLETED: this event is generated after the last A2DP_EVT_SEP_INFO_RECEIVED if the remote accepted the request and the p evt_param.discover_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.discover_completed.err_code == the error code sent by the remote.

param mgr A2DP manager. param remote_addr The address of a remote device.

return li c TRUE if discover request has been sent. li c FALSE otherwise. No events will be generated.

bt_a2dp_find_codec Macro

File

a2dp.h

C

```
#define bt_a2dp_find_codec(mgr, codec_type) bt_avdtp_find_codec(mgr->avdtp_mgr, codec_type)
```

Description

brief Find a codec ingroup a2dp

details A2DP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make our implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of A2DP has to register a callback function (one per codec type) for each codec it wishes to support. That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This function returns a pointer to a structure that holds a pointer to a codec's callback function.

param mgr A2DP manager. param codec_type Codec type. The codec_type can be one of the following values: @arg AVDTP_CODEC_TYPE_SBC: SBC @arg AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) @arg AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) @arg AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) @arg AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails if a callback for a codec type specified in the p codec parameter li has not been previously registered with ::bt_avdtp_register_codec.

bt_a2dp_get_capabilities Macro

File

a2dp.h

C

```
#define bt_a2dp_get_capabilities(mgr, remote_addr, seid_acp) bt_avdtp_get_capabilities(mgr->avdtp_mgr, remote_addr, seid_acp)
```

Description

brief Get remote SEP capabilities. ingroup a2dp

details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after A2DP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote.

param mgr A2DP manager. param remote_addr The address of a remote device. param seid_acp The ID of a remote SEP.

return li c TRUE if discover request has been sent. li c FALSE otherwise. No events will be generated.

bt_a2dp_get_hci_connection Macro

File

a2dp.h

C

```
#define bt_a2dp_get_hci_connection(mgr, strm_handle) bt_avdtp_get_hci_connection(mgr->avdtp_mgr, strm_handle)
```

Description

brief Get HCI connection for a stream ingroup a2dp

details This function returns a pointer to a structure that describes an HCI connection a stream is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect.

param mgr A2DP manager. param strm_handle Stream handle.

return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist or there is no HCI connection between local and remote devices associated with the stream.

note This function has not been implemented.

bt_a2dp_get_stream_codec_config Macro

File

a2dp.h

C

```
#define bt_a2dp_get_stream_codec_config(mgr, strm_handle) bt_avdtp_get_stream_codec_config(mgr->avdtp_mgr, strm_handle)
```

Description

brief Get the configuration of the codec currently used with the stream. ingroup a2dp

details This function returns a pointer to a structure that contains configuration of the codec currently used with the stream. The structure returned depends on the codec. The dotstack defines structures only for SBC, MPEG-1,2 and MPEG-2,4 AAC codecs: @arg SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h) @arg MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h) @arg MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h)

param mgr A2DP manager. param strm_handle Stream handle.

return li The codec's configuration if strm_handle specifies a valid stream and the stream is in one of the following state:

AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING

li NULL otherwise.

bt_a2dp_get_stream_codec_type Macro

File

a2dp.h

C

```
#define bt_a2dp_get_stream_codec_type(mgr, strm_handle) bt_avdtp_get_stream_codec_type(mgr->avdtp_mgr,
strm_handle)
```

Description

brief Get the type of the codec currently used with the stream. ingroup a2dp

details This function returns the type of the codec currently used with the stream.

param mgr A2DP manager. param strm_handle Stream handle.

return @arg The type of the codec if strm_handle specifies a valid stream and the stream is in one of the following states:

AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING

@arg The result will be one of the following values:

AVDTP_CODEC_TYPE_SBC: SBC AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

@arg 0xFF otherwise.

bt_a2dp_get_stream_local_sep_id Macro

File

a2dp.h

C

```
#define bt_a2dp_get_stream_local_sep_id(mgr, strm_handle) bt_avdtp_get_stream_local_sep_id(mgr->avdtp_mgr,
strm_handle)
```

Description

brief Get stream's local SEP ID. ingroup a2dp

details This function returns the ID of the local SEP associated with the stream.

param mgr A2DP manager. param strm_handle Stream handle.

return li The ID of the local SEP if strm_handle specifies a valid stream. li 0 otherwise.

bt_a2dp_get_stream_remote_address Macro

File

a2dp.h

C

```
#define bt_a2dp_get_stream_remote_address(mgr, strm_handle)
bt_avdtp_get_stream_remote_address(mgr->avdtp_mgr, strm_handle)
```

Description

brief Get stream's remote BT address. ingroup a2dp

details This function returns the address of the remote device associated with the stream.

param mgr A2DP manager. param strm_handle Stream handle.

return li The address of the remote device if strm_handle specifies a valid stream. li NULL otherwise.

bt_a2dp_get_stream_remote_sep_id Macro

File

a2dp.h

C

```
#define bt_a2dp_get_stream_remote_sep_id(mgr, strm_handle)
bt_avdtp_get_stream_remote_sep_id(mgr->avdtp_mgr, strm_handle)
```

Description

brief Get stream's remote SEP ID. ingroup a2dp

details This function returns the ID of the remote SEP associated with the stream.

param mgr A2DP manager. param strm_handle Stream handle.

return li The ID of the remote SEP if strm_handle specifies a valid stream. li 0 otherwise.

bt_a2dp_get_stream_state Macro

File

a2dp.h

C

```
#define bt_a2dp_get_stream_state(mgr, strm_handle) bt_avdtp_get_stream_state(mgr->avdtp_mgr, strm_handle)
```

Description

brief Get local stream state. ingroup a2dp

details This function returns local state of a stream specified by the p strm_handle. No request is sent to the remote party.

param mgr A2DP manager. param strm_handle Stream handle.

return The state of the stream. The result will be one of the following values: @arg AVDTP_STREAM_STATE_IDLE: The stream is idle. This can mean two things. The stream specified by p strm_handle does not exist or the stream is closed. @arg AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS: The stream is opening transport channels. @arg AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS: The stream is closing transport channels. @arg AVDTP_STREAM_STATE_CONFIGURED: The stream has been configured. @arg AVDTP_STREAM_STATE_OPEN: The stream has been opened. @arg AVDTP_STREAM_STATE_STREAMING: The stream has been started. Depending on the local SEP type (source or sink) it means that the stream is can send or receive media packets. @arg AVDTP_STREAM_STATE_CLOSING: The stream is closing. This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state. @arg AVDTP_STREAM_STATE_ABORTING: The stream is aborting. This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.

bt_a2dp_listen Macro

File

a2dp.h

C

```
#define bt_a2dp_listen(mgr, strm_handle, sep_id) bt_avdtp_listen(mgr->avdtp_mgr, strm_handle, sep_id)
```

Description

brief Listen for incoming connections. ingroup a2dp

details This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream.

param mgr A2DP manager. param strm_handle Stream handle. param sep_id Local SEP ID.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_a2dp_reconfigure_stream Macro

File

a2dp.h

C

```
#define bt_a2dp_reconfigure_stream(mgr, strm_handle, caps) bt_avdtp_reconfigure_stream(mgr->avdtp_mgr,
strm_handle, caps)
```

Description

brief Reconfigure stream. ingroup a2dp

details This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the A2DP_EVT_STREAM_RECONFIGURE_COMPLETED event will be generated. If reconfiguration was a success the p evt_param.stream_reconfigure_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.stream_reconfigure_completed.err_code == the error code sent by the remote.

param mgr A2DP manager. param strm_handle Stream handle. param caps New stream configuration.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_a2dp_register_sink Macro

File

a2dp.h

C

```
#define bt_a2dp_register_sink(mgr, caps) bt_avdtp_register_sep(mgr->avdtp_mgr, AVDTP_SEP_TYPE_SINK, caps)
```

Description

brief Register a Sink SEP with the local A2DP manager. ingroup a2dp

details This function is used to add a sink SEP to a list of SEPs supported by the local A2DP entity.

param mgr A2DP manager. param caps The capabilities of a SEP.

return li c ID of a SEP if the function succeeds. li c FALSE otherwise.

bt_a2dp_register_source Macro

File

a2dp.h

C

```
#define bt_a2dp_register_source(mgr, caps) bt_avdtp_register_sep(mgr->avdtp_mgr, AVDTP_SEP_TYPE_SOURCE,
caps)
```

Description

brief Register a Source SEP with the local A2DP manager. ingroup a2dp

details This function is used to add a source SEP to a list of SEPs supported by the local A2DP entity.

param mgr A2DP manager. param caps The capabilities of a SEP.

return li c ID of a SEP if the function succeeds. li c FALSE otherwise.

bt_a2dp_remove_media_rx_buffer Macro

File

a2dp.h

C

```
#define bt_a2dp_remove_media_rx_buffer(mgr, strm_handle, buffer)
bt_avdtp_remove_media_rx_buffer(mgr->avdtp_mgr, strm_handle, buffer)
```

Description

brief Remove a media packet buffer from a receive queue ingroup a2dp

details The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue or none of the buffers is large enough the received packets is dropped. Each buffer has a field (data_len) that holds the length of the received buffer. This field is never 0 if the buffer contains a packet. If a channel closed regardless of what has caused that and there are still buffers in the queue A2DP generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event for each buffer and sets the data_len to 0. This is to inform the A2DP consumer that the buffer has not been used and can be, for example, deallocated. This function removes a buffer from the receive queue.

param mgr A2DP manager. param strm_handle Stream handle. param buffer Pointer to a structure that holds the buffer and its parameters.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

bt_a2dp_remove_media_tx_buffer Macro

File

a2dp.h

C

```
#define bt_a2dp_remove_media_tx_buffer(mgr, strm_handle, buffer)
bt_avdtp_remove_media_tx_buffer(mgr->avdtp_mgr, strm_handle, buffer)
```

Description

brief Remove a media packet buffer from a send queue ingroup a2dp

details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling ::bt_a2dp_remove_media_tx_buffer.

param mgr A2DP manager. param strm_handle Stream handle. param buffer Pointer to a structure that holds the buffer and its parameters.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

note This function has not been implemented.

bt_a2dp_start_stream Macro

File

a2dp.h

C

```
#define bt_a2dp_start_stream(mgr, strm_handle) bt_avdtp_start_stream(mgr->avdtp_mgr, strm_handle)
```

Description

brief Start a stream. ingroup a2dp

details This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the A2DP_EVT_START_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.start_stream_requested.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the p evt_param.start_stream_requested.err_code == the error code sent by the remote.

param mgr A2DP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_a2dp_suspend_stream Macro

File

a2dp.h

C

```
#define bt_a2dp_suspend_stream(mgr, strm_handle) bt_avdtp_suspend_stream(mgr->avdtp_mgr, strm_handle)
```

Description

brief Suspend a stream. ingroup a2dp

details This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the A2DP_EVT_SUSPEND_STREAM_COMPLETED event will be generated. If the stream has been suspended the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == the error code sent by the remote.

param mgr A2DP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

MPEG_BITRATE_0000 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0000 0x0080
```

Description

This is macro MPEG_BITRATE_0000.

MPEG_BITRATE_0001 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0001 0x0100
```

Description

This is macro MPEG_BITRATE_0001.

MPEG_BITRATE_0010 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0010 0x0200
```

Description

This is macro MPEG_BITRATE_0010.

MPEG_BITRATE_0011 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0011 0x0400
```

Description

This is macro MPEG_BITRATE_0011.

MPEG_BITRATE_0100 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0100 0x0800
```

Description

This is macro MPEG_BITRATE_0100.

MPEG_BITRATE_0101 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0101 0x1000
```

Description

This is macro MPEG_BITRATE_0101.

MPEG_BITRATE_0110 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0110 0x2000
```

Description

This is macro MPEG_BITRATE_0110.

MPEG_BITRATE_0111 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_0111 0x4000
```

Description

This is macro MPEG_BITRATE_0111.

MPEG_BITRATE_1000 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1000 0x0001
```

Description

This is macro MPEG_BITRATE_1000.

MPEG_BITRATE_1001 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1001 0x0002
```

Description

This is macro MPEG_BITRATE_1001.

MPEG_BITRATE_1010 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1010 0x0004
```

Description

This is macro MPEG_BITRATE_1010.

MPEG_BITRATE_1011 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1011 0x0008
```

Description

This is macro MPEG_BITRATE_1011.

MPEG_BITRATE_1100 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1100 0x0010
```

Description

This is macro MPEG_BITRATE_1100.

MPEG_BITRATE_1101 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1101 0x0020
```

Description

This is macro MPEG_BITRATE_1101.

MPEG_BITRATE_1110 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_1110 0x0040
```

Description

This is macro MPEG_BITRATE_1110.

MPEG_BITRATE_ALL Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_BITRATE_ALL 0x7FFF
```

Description

This is macro MPEG_BITRATE_ALL.

MPEG_CHANNEL_MODE_ALL Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CHANNEL_MODE_ALL 15
```

Description

This is macro MPEG_CHANNEL_MODE_ALL.

MPEG_CHANNEL_MODE_DUAL_CHANNEL Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CHANNEL_MODE_DUAL_CHANNEL 4
```

Description

This is macro MPEG_CHANNEL_MODE_DUAL_CHANNEL.

MPEG_CHANNEL_MODE_JOINT_STEREO Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CHANNEL_MODE_JOINT_STEREO 1
```

Description

This is macro MPEG_CHANNEL_MODE_JOINT_STEREO.

MPEG_CHANNEL_MODE_MONO Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CHANNEL_MODE_MONO 8
```

Description

This is macro MPEG_CHANNEL_MODE_MONO.

MPEG_CHANNEL_MODE_STEREO Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CHANNEL_MODE_STEREO 2
```

Description

This is macro MPEG_CHANNEL_MODE_STEREO.

MPEG_CRC_PROTECTION_NOT_SUPPORTED Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CRC_PROTECTION_NOT_SUPPORTED 0
```

Description

This is macro MPEG_CRC_PROTECTION_NOT_SUPPORTED.

MPEG_CRC_PROTECTION_SUPPORTED Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_CRC_PROTECTION_SUPPORTED 1
```

Description

This is macro MPEG_CRC_PROTECTION_SUPPORTED.

MPEG_LAYER_1 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_LAYER_1 0x04
```

Description

This is macro MPEG_LAYER_1.

MPEG_LAYER_2 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_LAYER_2 0x02
```

Description

This is macro MPEG_LAYER_2.

MPEG_LAYER_3 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_LAYER_3 0x01
```

Description

This is macro MPEG_LAYER_3.

MPEG_LAYER_ALL Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_LAYER_ALL 0x07
```

Description

This is macro MPEG_LAYER_ALL.

MPEG_MPF_1 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_MPF_1 0
```

Description

This is macro MPEG_MPF_1.

MPEG_MPF_2 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_MPF_2 1
```

Description

This is macro MPEG_MPF_2.

MPEG_SAMPLING_FREQUENCY_16000 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_16000 0x20
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_16000.

MPEG_SAMPLING_FREQUENCY_22050 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_22050 0x10
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_22050.

MPEG_SAMPLING_FREQUENCY_24000 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_24000 0x08
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_24000.

MPEG_SAMPLING_FREQUENCY_32000 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_32000 0x04
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_32000.

MPEG_SAMPLING_FREQUENCY_44100 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_44100 0x02
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_44100.

MPEG_SAMPLING_FREQUENCY_48000 Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_48000 0x01
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_48000.

MPEG_SAMPLING_FREQUENCY_ALL Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_SAMPLING_FREQUENCY_ALL 0x3F
```

Description

This is macro MPEG_SAMPLING_FREQUENCY_ALL.

MPEG_VBR_NOT_SUPPORTED Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_VBR_NOT_SUPPORTED 0
```

Description

This is macro MPEG_VBR_NOT_SUPPORTED.

MPEG_VBR_SUPPORTED Macro

File

a2dp_codec_mpeg.h

C

```
#define MPEG_VBR_SUPPORTED 1
```

Description

This is macro MPEG_VBR_SUPPORTED.

SBC_ALLOCATION_METHOD_ALL Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_ALLOCATION_METHOD_ALL 3
```

Description

This is macro SBC_ALLOCATION_METHOD_ALL.

SBC_ALLOCATION_METHOD_LOUDNESS Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_ALLOCATION_METHOD_LOUDNESS 1
```

Description

This is macro SBC_ALLOCATION_METHOD_LOUDNESS.

SBC_ALLOCATION_METHOD_SNR Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_ALLOCATION_METHOD_SNR 2
```

Description

This is macro SBC_ALLOCATION_METHOD_SNR.

SBC_BLOCK_LENGTH_12 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_BLOCK_LENGTH_12 2
```

Description

This is macro SBC_BLOCK_LENGTH_12.

SBC_BLOCK_LENGTH_16 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_BLOCK_LENGTH_16 1
```

Description

This is macro SBC_BLOCK_LENGTH_16.

SBC_BLOCK_LENGTH_4 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_BLOCK_LENGTH_4 8
```

Description

This is macro SBC_BLOCK_LENGTH_4.

SBC_BLOCK_LENGTH_8 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_BLOCK_LENGTH_8 4
```

Description

This is macro SBC_BLOCK_LENGTH_8.

SBC_BLOCK_LENGTH_ALL Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_BLOCK_LENGTH_ALL 15
```

Description

This is macro SBC_BLOCK_LENGTH_ALL.

SBC_CHANNEL_MODE_ALL Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_CHANNEL_MODE_ALL 15
```

Description

This is macro SBC_CHANNEL_MODE_ALL.

SBC_CHANNEL_MODE_DUAL_CHANNEL Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_CHANNEL_MODE_DUAL_CHANNEL 4
```

Description

This is macro SBC_CHANNEL_MODE_DUAL_CHANNEL.

SBC_CHANNEL_MODE_JOINT_STEREO Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_CHANNEL_MODE_JOINT_STEREO 1
```

Description

This is macro SBC_CHANNEL_MODE_JOINT_STEREO.

SBC_CHANNEL_MODE_MONO Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_CHANNEL_MODE_MONO 8
```

Description

This is macro SBC_CHANNEL_MODE_MONO.

SBC_CHANNEL_MODE_STEREO Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_CHANNEL_MODE_STEREO 2
```

Description

This is macro SBC_CHANNEL_MODE_STEREO.

SBC_SAMPLING_FREQUENCY_16000 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SAMPLING_FREQUENCY_16000 8
```

Description

This is macro SBC_SAMPLING_FREQUENCY_16000.

SBC_SAMPLING_FREQUENCY_32000 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SAMPLING_FREQUENCY_32000 4
```

Description

This is macro SBC_SAMPLING_FREQUENCY_32000.

SBC_SAMPLING_FREQUENCY_44100 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SAMPLING_FREQUENCY_44100 2
```

Description

This is macro SBC_SAMPLING_FREQUENCY_44100.

SBC_SAMPLING_FREQUENCY_48000 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SAMPLING_FREQUENCY_48000 1
```

Description

This is macro SBC_SAMPLING_FREQUENCY_48000.

SBC_SAMPLING_FREQUENCY_ALL Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SAMPLING_FREQUENCY_ALL 15
```

Description

This is macro SBC_SAMPLING_FREQUENCY_ALL.

SBC_SUBBANDS_4 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SUBBANDS_4 2
```

Description

This is macro SBC_SUBBANDS_4.

SBC_SUBBANDS_8 Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SUBBANDS_8 1
```

Description

This is macro SBC_SUBBANDS_8.

SBC_SUBBANDS_ALL Macro

File

a2dp_codec_sbc.h

C

```
#define SBC_SUBBANDS_ALL 3
```

Description

This is macro SBC_SUBBANDS_ALL.

bt_a2dp_get_all_capabilities Macro

File

a2dp.h

C

```
#define bt_a2dp_get_all_capabilities(mgr, remote_addr, seid_acp)
bt_avdtp_get_all_capabilities(mgr->avdtp_mgr, remote_addr, seid_acp)
```

Description

brief Get remote SEP capabilities. ingroup a2dp

details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after A2DP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote.

param mgr A2DP manager. param remote_addr The address of a remote device. param seid_acp The ID of a remote SEP.

return li c TRUE if discover request has been sent. li c FALSE otherwise. No events will be generated.

bt_a2dp_get_configuration Macro

File

a2dp.h

C

```
#define bt_a2dp_get_configuration(mgr, strm_handle) bt_avdtp_get_configuration(mgr->avdtp_mgr, strm_handle)
```

Description

This is macro bt_a2dp_get_configuration.

bt_a2dp_open_stream Macro

File

a2dp.h

C

```
#define bt_a2dp_open_stream(mgr, strm_handle) bt_avdtp_open_stream(mgr->avdtp_mgr, strm_handle)
```

Description

This is macro bt_a2dp_open_stream.

bt_a2dp_clear_media_tx_queue Macro

File

a2dp.h

C

```
#define bt_a2dp_clear_media_tx_queue(mgr, strm_handle) bt_avdtp_clear_media_tx_queue(mgr->avdtp_mgr,
strm_handle)
```

Description

brief Clear send queue ingroup a2dp

details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. The consumer can remove all packets from the queue before they have been sent to a remote device by calling ::bt_a2dp_clear_media_tx_queue.

param mgr A2DP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does

not exist. The stream can be in any state to call this function.

bt_a2dp_get_stream_config Macro

File

a2dp.h

C

```
#define bt_a2dp_get_stream_config(mgr, strm_handle) bt_avdtp_get_stream_config(mgr->avdtp_mgr, strm_handle)
```

Description

brief Get stream's configuration. ingroup a2dp

details This function returns a pointer to a structure holding current configuration of the stream.

param mgr A2DP manager. param strm_handle Stream handle.

return li The stream's configuration if strm_handle specifies a valid stream and the stream is in one of the following state:

AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING

li NULL otherwise.

bt_a2dp_get_stream_direction Macro

File

a2dp.h

C

```
#define bt_a2dp_get_stream_direction(mgr, strm_handle) bt_avdtp_get_stream_direction(mgr->avdtp_mgr,
strm_handle)
```

Description

This is macro bt_a2dp_get_stream_direction.

bt_a2dp_report_delay Macro

File

a2dp.h

C

```
#define bt_a2dp_report_delay(mgr, strm_handle, delay) bt_avdtp_report_delay(mgr->avdtp_mgr, strm_handle,
delay)
```

Description

This is macro bt_a2dp_report_delay.

bt_a2dp_set_media_tx_queue_limit Macro

File

a2dp.h

C

```
#define bt_a2dp_set_media_tx_queue_limit(mgr, strm_handle, limit)
bt_avdtp_set_media_tx_queue_limit(mgr->avdtp_mgr, strm_handle, limit)
```

Description

brief Set limit on the send queue ingroup a2dp

details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. By default the send queue can contain unlimited number of packets. The consumer can set a limit on how many packets are held in the queue. In this case when new packet is added to the queue and the length of the queue exceeds the set limit the first packet is removed from the queue. The removed packet is not send to the remote device.

param mgr A2DP manager. param strm_handle Stream handle. param limit Queue limit.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

A2DP_EVT_SET_STREAM_CONFIGURATION Macro

File

a2dp.h

C

```
#define A2DP_EVT_SET_STREAM_CONFIGURATION AVDTP_EVT_SET_STREAM_CONFIGURATION
```

Description

This is macro A2DP_EVT_SET_STREAM_CONFIGURATION.

AVCTP Functions

_bt_avctp_allocate_channel Function

File

avctp_private.h

C

```
bt_avctp_channel_t* _bt_avctp_allocate_channel(bt_avctp_mgr_t* mgr);
```

Description

This is function _bt_avctp_allocate_channel.

_bt_avctp_allocate_message Function

File

avctp_private.h

C

```
bt_avctp_message_t* _bt_avctp_allocate_message(bt_byte msg_type);
```

Description

This is function _bt_avctp_allocate_message.

_bt_avctp_allocate_transport Function

File

avctp_private.h

C

```
bt_avctp_transport_t* _bt_avctp_allocate_transport(bt_avctp_mgr_t* mgr);
```

Description

This is function _bt_avctp_allocate_transport.

_bt_avctp_find_channel Function

File

avctp_private.h

C

```
bt_avctp_channel_t* _bt_avctp_find_channel(bt_avctp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_uint
profile_id, bt_int psm);
```

Description

This is function _bt_avctp_find_channel.

_bt_avctp_find_transport Function

File

avctp_private.h

C

```
bt_avctp_transport_t* _bt_avctp_find_transport(bt_avctp_mgr_t* mgr, bt_l2cap_channel_t* l2cap_channel);
```

Description

This is function _bt_avctp_find_transport.

_bt_avctp_free_channel Function

File

avctp_private.h

C

```
void _bt_avctp_free_channel(bt_avctp_channel_t* channel);
```

Description

This is function _bt_avctp_free_channel.

_bt_avctp_free_message Function

File

avctp_private.h

C

```
void _bt_avctp_free_message(bt_avctp_message_t* message);
```

Description

This is function _bt_avctp_free_message.

_bt_avctp_free_transport Function

File

avctp_private.h

C

```
void _bt_avctp_free_transport(bt_avctp_transport_t* transport);
```

Description

This is function _bt_avctp_free_transport.

_bt_avctp_init_message_buffers Function

File

avctp_private.h

C

```
void _bt_avctp_init_message_buffers();
```

Description

This is function _bt_avctp_init_message_buffers.

_bt_avctp_init_signal Function

File

avctp_private.h

C

```
void _bt_avctp_init_signal();
```

Description

This is function _bt_avctp_init_signal.

_bt_avctp_l2cap_read_data_callback Function

File

avctp_private.h

C

```
void _bt_avctp_l2cap_read_data_callback(struct _bt_l2cap_channel_t * pch, bt_byte* pdata, bt_int len);
```

Description

This is function _bt_avctp_l2cap_read_data_callback.

_bt_avctp_packet_assembler Function

File

avctp_packet.h

C

```
bt_int _bt_avctp_packet_assembler(bt_packet_t* packet, bt_byte* buffer, bt_int buffer_len);
```

Description

This is function _bt_avctp_packet_assembler.

_bt_avctp_send_ipid Function

File

avctp_private.h

C

```
bt_bool _bt_avctp_send_ipid(bt_avctp_transport_t* transport, bt_byte tran_id);
```

Description

This is function _bt_avctp_send_ipid.

_bt_avctp_set_signal Function

File

avctp_private.h

C

```
void _bt_avctp_set_signal();
```

Description

This is function _bt_avctp_set_signal.

_bt_avrcp_init_cmd_buffers Function

File

avctp_private.h

C

```
void _bt_avrcp_init_cmd_buffers();
```

Description

This is function _bt_avrcp_init_cmd_buffers.

bt_avctp_cancel_command Function

File

avctp.h

C

```
void bt_avctp_cancel_command(bt_avctp_channel_t* channel, bt_byte tran_id);
```

Description

brief Cancel a command message. ingroup avctp

details If a message has not yet been sent to the remote device, it can be canceled (i.e. removed from send queue) by calling this function.

param channel AVCTP channel. param tran_id Transaction Id. This value is obtained by calling bt_avctp_send_command.

return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.

bt_avctp_cancel_listen Function

File

avctp.h

C

```
void bt_avctp_cancel_listen(bt_avctp_channel_t* channel);
```

Description

brief Cancel listening for incoming connections. ingroup avctp

details This function stops listening for incoming connections on the specified channel.

param channel AVCTP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avctp_cancel_response Function

File

avctp.h

C

```
void bt_avctp_cancel_response(bt_avctp_channel_t* channel, bt_byte tran_id);
```

Description

brief Cancel a response message. ingroup avctp

details If a message has not yet been sent to the remote device, it can be canceled (i.e. removed from send queue) by calling this function.

param channel AVCTP channel. param tran_id Transaction Id. This value is obtained by calling bt_avctp_send_command.

return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.

bt_avctp_connect Function

File

avctp.h

C

```
bt_bool bt_avctp_connect(bt_avctp_channel_t* channel, bt_bdaddr_t* remote_address, bt_uint acl_config);
```

Description

brief Connect to a remote device. ingroup avctp

details This function establishes a connection to a remote device specified by the p remote_address. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVCTP callback. The events generated will either be AVCTP_EVT_CHANNEL_CONNECTED or AVCTP_EVT_CONNECTION_FAILED.

param channel AVCTP channel. param remote_address The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following values: li HCI_CONFIG_ENABLE_AUTHENTICATION li HCI_CONFIG_ENABLE_ENCRYPTION li HCI_CONFIG_BECOME_MASTER

return li c TRUE if connection establishment has been started. li c FALSE otherwise.

bt_avctp_create_channel Function

File

avctp.h

C

```
bt_avctp_channel_t* bt_avctp_create_channel(bt_avctp_mgr_t* mgr, bt_uint profile_id, bt_int psm, bt_byte l2cap_mode);
```

Description

brief Allocate AVCTP channel ingroup avctp

details This function allocates a new incoming AVCTP channel. The channel is intended to be used to accept a connection from a remote device. There can be only one channel for each combination of c profile_id and c psm.

param mgr AVCTP manager. param profile_id Profile Id param psm The PSM on which the underlying L2CAP channel will listen and accept incoming connections. param l2cap_mode Underlying L2CAP channel mode. This currently can only be CMODE_BASIC.

return li A pointer to the new AVCTP channel if the function succeeds. li c NULL otherwise.

bt_avctp_create_outgoing_channel Function

File

avctp.h

C

```
bt_avctp_channel_t* bt_avctp_create_outgoing_channel(bt_avctp_mgr_t* mgr, bt_uint profile_id, bt_int psm, bt_byte l2cap_mode);
```

Description

brief Allocate AVCTP channel ingroup avctp

details This function allocates a new outgoing AVCTP channel. The channel is intended to be used to create a connection to a remote device. There can be multiple channels with the same c profile_id and c psm.

param mgr AVCTP manager. param profile_id Profile Id param psm The PSM on which the underlying L2CAP channel will listen and accept incoming connections. param l2cap_mode Underlying L2CAP channel mode. This currently can only be CMODE_BASIC.

return li A pointer to the new AVCTP channel if the function succeeds. li c NULL otherwise.

bt_avctp_destroy_channel Function

File

avctp.h

C

```
bt_bool bt_avctp_destroy_channel(bt_avctp_channel_t* channel);
```

Description

brief Destroy AVCTP channel. ingroup avctp

details This function frees memory used by the channel. The channel has to exist and be in the "idle" state for this function to succeed. I.e. the channel has to be disconnected before this function can be called.

param channel AVCTP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avctp_disconnect Function

File

avctp.h

C

```
bt_bool bt_avctp_disconnect(bt_avctp_channel_t* channel);
```

Description

brief Disconnect from a remote device. ingroup avctp

details This function closes a connection to a remote device.

param channel AVCTP channel.

return li c TRUE if disconnection has been started. li c FALSE otherwise. No events will be generated.

bt_avctp_get_channel_remote_address Function

File

avctp.h

C

```
bt_bdaddr_t* bt_avctp_get_channel_remote_address(bt_avctp_channel_t* channel);
```

Description

brief Get channel's remote BT address. ingroup avctp

details This function returns the address of the remote device associated with the channel.

param channel AVCTP channel.

return li The address of the remote device if channel is connected. li NULL otherwise.

bt_avctp_get_channel_state Function

File

avctp.h

C

```
bt_byte bt_avctp_get_channel_state(bt_avctp_channel_t* channel);
```

Description

brief Get channel state ingroup avctp

details This function return current state of the specified channel

param channel AVCTP channel.

return li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING

bt_avctp_get_hci_connection Function

File

avctp.h

C

```
bt_hci_conn_state_t* bt_avctp_get_hci_connection(bt_avctp_channel_t* channel);
```

Description

brief Get HCI connection for a channel ingroup avctp

details This function returns a pointer to a structure that describes an HCI connection a channel is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect.

param channel AVCTP channel.

return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a channel specified by the p channel parameter li does not exist or there is no HCI connection between local and remote devices associated with the channel.

bt_avctp_get_mgr Function

File

avctp.h

C

```
bt_avctp_mgr_t* bt_avctp_get_mgr();
```

Description

brief Return a pointer to an instance of the AVCTP manager. ingroup avctp

details This function returns a pointer to an instance of the AVCTP manager. There is only one instance of the manager allocated by the stack.

bt_avctp_init Function

File

avctp.h

C

```
void bt_avctp_init();
```

Description

brief Initialize the AVCTP layer. ingroup avctp

details This function initializes the AVCTP layer of the stack. It must be called prior to any other AVCTP function can be called.

bt_avctp_listen Function

File

avctp.h

C

```
bt_bool bt_avctp_listen(bt_avctp_channel_t* channel);
```

Description

brief Listen for incoming connections. ingroup avctp

details This function enables incoming connections on the specified AVCTP channel.

param channel AVCTP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avctp_send_command Function

File

avctp.h

C

```
bt_bool bt_avctp_send_command(bt_avctp_channel_t* channel, bt_byte* data, bt_uint data_len, bt_byte*
tran_id);
```

Description

brief Send a command message to a remote device. ingroup avctp

details This function sends a command message to a remote device.

param channel AVCTP channel. param data Message body. param data_len Message body length. param tran_id Pointer to a bt_byte where AVRCP will write transaction id assigned to the message.

return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.

bt_avctp_send_response Function

File

avctp.h

C

```
bt_bool bt_avctp_send_response(bt_avctp_channel_t* channel, bt_byte tran_id, bt_byte* data, bt_uint
data_len);
```

Description

brief Send a response message to a remote device. ingroup avctp

details This function sends a response message to a remote device.

param channel AVCTP channel. param tran_id Transaction Id. This value is obtained by calling bt_avctp_send_command. param data Message body. param data_len Message body length.

return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.

bt_avctp_set_callback Function

File

avctp.h

C

```
void bt_avctp_set_callback(bt_avctp_mgr_t* mgr, bt_avctp_mgr_callback_fp callback, void* callback_param);
```

Description

brief Register a AVCTP application callback. ingroup avctp

details In order to be notified of various events a consumer of the AVCTP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values: @arg AVCTP_EVT_CHANNEL_CONNECTED Channel connected. @arg AVCTP_EVT_CHANNEL_DISCONNECTED Channel disconnected. @arg AVCTP_EVT_CONNECTION_FAILED Channel connection failed (generated only if connection has been initiated by the local device).

@arg AVCTP_EVT_COMMAND_RECEIVED Command received. @arg AVCTP_EVT_RESPONE_RECEIVED Response received. @arg AVCTP_EVT_COMMAND_SENT Command sent. @arg AVCTP_EVT_RESPONSE_SENT Response sent. @arg AVCTP_EVT_COMMAND_CANCELLED Command canceled. @arg AVCTP_EVT_RESPONSE_CANCELLED Response canceled.

param mgr AVCTP manager. param callback The callback function that will be called when the AVCTP generates an event. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

bt_avctp_start Function

File

avctp.h

C

```
bt_bool bt_avctp_start(bt_avctp_mgr_t* mgr);
```

Description

This is function bt_avctp_start.

AVCTP Data Types and Constants

_bt_avctp_channel_t Structure

File

avctp.h

C

```
struct _bt_avctp_channel_t {
  bt_byte state;
  bt_byte flags;
  bt_avctp_transport_t* transport;
  bt_uint profile_id;
  bt_int psm;
  bt_byte l2cap_mode;
  bt_avctp_mgr_t* avctp_mgr;
};
```

Description

brief AVCTP channel description ingroup avctp

details This structure is used to hold information about an AVCTP channel.

_bt_avctp_message_t Structure

File

avctp.h

C

```
struct _bt_avctp_message_t {
  bt_avctp_message_t* next_message;
  bt_byte tran_id;
  bt_byte packet_type;
  bt_byte num_packets;
  bt_byte message_type;
  bt_byte ipid;
  bt_uint profile_id;
  bt_byte* data;
  bt_uint data_len;
};
```

Description

brief AVCTP message description ingroup avctp

details This structure is used to hold information about an AVCTP message.

_bt_avctp_mgr_t Structure

File

avctp.h

C

```
struct _bt_avctp_mgr_t {
  bt_byte state;
  bt_byte flags;
  bt_avctp_channel_t* channels;
  bt_avctp_transport_t* transports;
  bt_avctp_mgr_callback_fp callback;
  void* callback_param;
  bt_avctp_channel_t* opening_channel;
};
```

Members

Members	Description
bt_byte state;	< Manager state. This value can be one of the following values: < li AVCTP_MANAGER_STATE_IDLE < li AVCTP_MANAGER_STATE_CONNECTING

bt_byte flags;	< Additional manager state. This value can be a combination of the following values: < li AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET
bt_avctp_channel_t* channels;	< List of available AVCTP channels.
bt_avctp_transport_t* transports;	< List of available AVCTP transports.
bt_avctp_mgr_callback_fp callback;	< Pointer to a function used to notify the AVCTP consumer about various events.
void* callback_param;	< Pointer to arbitrary data to be passed to the c callback.
bt_avctp_channel_t* opening_channel;	< Pointer to a channle being open.

Description

brief AVCTP manager. ingroup avctp

details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avctp_get_mgr().

_bt_avctp_transport_t Structure

File

avctp.h

C

```
struct _bt_avctp_transport_t {
  bt_byte state;
  bt_byte flags;
  bt_bdaddr_t remote_addr;
  bt_l2cap_channel_t* l2cap_channel;
  bt_avctp_packet_t packet;
  bt_byte* tx_buffer;
  bt_byte* rx_buffer;
  bt_uint rx_message_len;
  bt_byte ref_count;
  bt_queue_element_t* send_cq_head;
  bt_queue_element_t* ack_cq_head;
  bt_byte next_transaction_id;
  bt_avctp_message_t* tx_message;
  bt_uint tx_message_remaining_data_len;
  bt_avctp_message_t rx_message;
  bt_avctp_mgr_t* avctp_mgr;
};
```

Description

brief AVCTP transport description ingroup avctp

details This structure is used to hold information about an AVCTP transport.

bt_avctp_channel_t Type

File

avctp.h

C

```
typedef struct _bt_avctp_channel_t bt_avctp_channel_t;
```

Description

This is type bt_avctp_channel_t.

bt_avctp_event_t Union

File

avctp.h

C

```
typedef union _bt_avctp_event_u {
  bt_avctp_evt_channel_connected_t channel_connected;
  bt_avctp_evt_channel_disconnected_t channel_disconnected;
```

```
  bt_avctp_evt_connection_failed_t connection_failed;
  bt_avctp_evt_command_received_t command_received;
  bt_avctp_evt_response_received_t response_received;
  bt_avctp_evt_command_sent_t command_sent;
  bt_avctp_evt_response_sent_t response_sent;
  bt_avctp_evt_command_cancelled_t command_cancelled;
  bt_avctp_evt_response_cancelled_t response_cancelled;
} bt_avctp_event_t;
```

Members

Members	Description
bt_avctp_evt_channel_connected_t channel_connected;	< Valid if event is AVCTP_EVT_CHANNEL_CONNECTED
bt_avctp_evt_channel_disconnected_t channel_disconnected;	< Valid if event is AVCTP_EVT_CHANNEL_DISCONNECTED
bt_avctp_evt_connection_failed_t connection_failed;	< Valid if event is AVCTP_EVT_CONNECTION_FAILED
bt_avctp_evt_command_received_t command_received;	< Valid if event is AVCTP_EVT_COMMAND_RECEIVED
bt_avctp_evt_response_received_t response_received;	< Valid if event is AVCTP_EVT_RESPONE_RECEIVED
bt_avctp_evt_command_sent_t command_sent;	< Valid if event is AVCTP_EVT_COMMAND_SENT
bt_avctp_evt_response_sent_t response_sent;	< Valid if event is AVCTP_EVT_RESPONSE_SENT
bt_avctp_evt_command_cancelled_t command_cancelled;	< Valid if event is AVCTP_EVT_COMMAND_CANCELLED
bt_avctp_evt_response_cancelled_t response_cancelled;	< Valid if event is AVCTP_EVT_RESPONSE_CANCELLED

Description

brief Parameter to an application callback. ingroup avctp

details This union is used to pass event specific data to the AVCTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

bt_avctp_evt_channel_connected_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_channel_connected_t {
  bt_avctp_channel_t* channel;
} bt_avctp_evt_channel_connected_t;
```

Members

Members	Description
bt_avctp_channel_t* channel;	< AVCTP channel

Description

brief Parameter to AVCTP_EVT_CHANNEL_CONNECTED event ingroup avctp

details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_connected - when a channel between two devices has been established.

bt_avctp_evt_channel_disconnected_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_channel_disconnected_t {
  bt_avctp_channel_t* channel;
} bt_avctp_evt_channel_disconnected_t;
```

Members

Members	Description
bt_avctp_channel_t* channel;	< AVCTP channel

Description

brief Parameter to AVCTP_EVT_CHANNEL_DISCONNECTED event ingroup avctp

details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_disconnected - when a channel between two devices has been terminated.

bt_avctp_evt_command_cancelled_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_command_cancelled_t {
  bt_avctp_channel_t* channel;
  bt_avctp_message_t* command;
} bt_avctp_evt_command_cancelled_t;
```

Members

Members	Description
bt_avctp_channel_t* channel;	< AVCTP channel
bt_avctp_message_t* command;	< AVCTP command message

Description

brief Parameter to AVCTP_EVT_COMMAND_CANCELLED event ingroup avctp

details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_cancelled - when sending a command message has been canceled.

bt_avctp_evt_command_received_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_command_received_t {
  bt_avctp_channel_t* channel;
  bt_avctp_message_t* command;
} bt_avctp_evt_command_received_t;
```

Members

Members	Description
bt_avctp_channel_t* channel;	< AVCTP channel
bt_avctp_message_t* command;	< AVCTP command message

Description

brief Parameter to AVCTP_EVT_COMMAND_RECEIVED event ingroup avctp

details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_received - when a local device received a command message.

bt_avctp_evt_command_sent_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_command_sent_t {
  bt_avctp_channel_t* channel;
```

```
  bt_avctp_message_t* command;
} bt_avctp_evt_command_sent_t;
```

Members

Members	Description
bt_avctp_channel_t* channel;	< AVCTP channel
bt_avctp_message_t* command;	< AVCTP command message

Description

brief Parameter to AVCTP_EVT_COMMAND_SENT event ingroup avctp

details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_sent - when a local device finished sending a command message.

bt_avctp_evt_connection_failed_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_connection_failed_t {
  bt_avctp_channel_t* channel;
} bt_avctp_evt_connection_failed_t;
```

Members

Members	Description
bt_avctp_channel_t* channel;	< AVCTP channel

Description

brief Parameter to AVCTP_EVT_CONNECTION_FAILED event ingroup avctp

details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::connection_failed - when a channel between two devices could not be established.

bt_avctp_evt_response_cancelled_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_response_cancelled_t {
  bt_avctp_channel_t* channel;
  bt_avctp_message_t* response;
} bt_avctp_evt_response_cancelled_t;
```

Members

Members	Description
bt_avctp_channel_t* channel;	< AVCTP channel
bt_avctp_message_t* response;	< AVCTP response message

Description

brief Parameter to AVCTP_EVT_RESPONSE_CANCELLED event ingroup avctp

details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_cancelled - when sending a response message has been canceled.

bt_avctp_evt_response_received_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_response_received_t {
```

```
  bt_avctp_channel_t* channel;
  bt_avctp_message_t* response;
} bt_avctp_evt_response_received_t;
```

Members

Members	Description
bt_avctp_channel_t* channel;	< AVCTP channel
bt_avctp_message_t* response;	< AVCTP response message

Description

brief Parameter to AVCTP_EVT_RESPONSE_RECEIVED event ingroup avctp

details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_received - when a local device received a response message.

bt_avctp_evt_response_sent_t Structure

File

avctp.h

C

```
typedef struct _bt_avctp_evt_response_sent_t {
  bt_avctp_channel_t* channel;
  bt_avctp_message_t* response;
} bt_avctp_evt_response_sent_t;
```

Members

Members	Description
bt_avctp_channel_t* channel;	< AVCTP channel
bt_avctp_message_t* response;	< AVCTP response message

Description

brief Parameter to AVCTP_EVT_RESPONSE_SENT event ingroup avctp

details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_sent - when a local device finished sending a response message.

bt_avctp_message_t Type

File

avctp.h

C

```
typedef struct _bt_avctp_message_t bt_avctp_message_t;
```

Description

This is type bt_avctp_message_t.

bt_avctp_mgr_callback_fp Type

File

avctp.h

C

```
typedef void (* bt_avctp_mgr_callback_fp)(struct _bt_avctp_mgr_t* mgr, bt_byte evt, bt_avctp_event_t*
evt_param, void* callback_param);
```

Description

brief AVCTP application callback. ingroup avctp

details In order to be notified of various events a consumer of the AVCTP layer has to register a callback function (done with bt_avctp_set_callback()). The stack will call that function whenever a new event has been generated.

param mgr AVCTP manager.

param evt AVCTP event. The event can be one of the following values: @arg AVCTP_EVT_CHANNEL_CONNECTED Channel connected. @arg AVCTP_EVT_CHANNEL_DISCONNECTED Channel disconnected. @arg AVCTP_EVT_CONNECTION_FAILED Channel connection failed (generated only if connection has been initiated by the local device).

@arg AVCTP_EVT_COMMAND_RECEIVED Command received. @arg AVCTP_EVT_RESPONE_RECEIVED Response received. @arg AVCTP_EVT_COMMAND_SENT Command sent. @arg AVCTP_EVT_RESPONSE_SENT Response sent. @arg AVCTP_EVT_COMMAND_CANCELLED Command canceled. @arg AVCTP_EVT_RESPONSE_CANCELLED Response canceled.

param evt_param Event parameter. Which member of the bt_avctp_event_t union is valid depends on the event: @arg AVCTP_EVT_CHANNEL_CONNECTED c bt_avctp_evt_channel_connected_t channel_connected @arg AVCTP_EVT_CHANNEL_DISCONNECTED c bt_avctp_evt_channel_disconnected_t channel_disconnected @arg AVCTP_EVT_CONNECTION_FAILED c bt_avctp_evt_connection_failed_t connection_failed @arg AVCTP_EVT_COMMAND_RECEIVED c bt_avctp_evt_command_received_t command_received @arg AVCTP_EVT_RESPONE_RECEIVED c bt_avctp_evt_response_received_t response_received @arg AVCTP_EVT_COMMAND_SENT c bt_avctp_evt_command_sent_t command_sent @arg AVCTP_EVT_RESPONSE_SENT c bt_avctp_evt_response_sent_t response_sent @arg AVCTP_EVT_COMMAND_CANCELLED c bt_avctp_evt_command_cancelled_t command_cancelled @arg AVCTP_EVT_RESPONSE_CANCELLED c bt_avctp_evt_response_cancelled_t response_cancelled

param callback_param A pointer to an arbitrary data set by a call to bt_avctp_set_callback.

bt_avctp_mgr_t Type

File

avctp.h

C

```
typedef struct _bt_avctp_mgr_t bt_avctp_mgr_t;
```

Description

This is type bt_avctp_mgr_t.

bt_avctp_packet_t Structure

File

avctp_packet.h

C

```
typedef struct _bt_avctp_packet_t {
  bt_packet_t header;
  bt_byte tran_id;
  bt_byte packet_type;
  bt_byte num_packets;
  bt_byte message_type;
  bt_byte ipid;
  bt_uint profile_id;
  bt_byte* data;
  bt_uint data_len;
  bt_int data_pos;
} bt_avctp_packet_t;
```

Description

This is type bt_avctp_packet_t.

bt_avctp_transport_t Type

File

avctp.h

C

```
typedef struct _bt_avctp_transport_t bt_avctp_transport_t;
```

Description

This is type bt_avctp_transport_t.

__AVCTP_H Macro

File

avctp.h

C

```
#define __AVCTP_H
```

Description

This is macro __AVCTP_H.

__AVCTP_PACKET_H Macro

File

avctp_packet.h

C

```
#define __AVCTP_PACKET_H
```

Description

This is macro __AVCTP_PACKET_H.

__AVCTP_PRIVATE_H Macro

File

avctp_private.h

C

```
#define __AVCTP_PRIVATE_H
```

Description

This is macro __AVCTP_PRIVATE_H.

AVCTP_CHANNEL_FLAG_LISTENING Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_FLAG_LISTENING 1
```

Description

This is macro AVCTP_CHANNEL_FLAG_LISTENING.

AVCTP_CHANNEL_FLAG_SENDING Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_FLAG_SENDING 2
```

Description

This is macro AVCTP_CHANNEL_FLAG_SENDING.

AVCTP_CHANNEL_STATE_CONNECTED Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_STATE_CONNECTED 3
```

Description

This is macro AVCTP_CHANNEL_STATE_CONNECTED.

AVCTP_CHANNEL_STATE_CONNECTING Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_STATE_CONNECTING 2
```

Description

This is macro AVCTP_CHANNEL_STATE_CONNECTING.

AVCTP_CHANNEL_STATE_DISCONNECTING Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_STATE_DISCONNECTING 4
```

Description

This is macro AVCTP_CHANNEL_STATE_DISCONNECTING.

AVCTP_CHANNEL_STATE_FREE Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_STATE_FREE 0
```

Description

This is macro AVCTP_CHANNEL_STATE_FREE.

AVCTP_CHANNEL_STATE_IDLE Macro

File

avctp.h

C

```
#define AVCTP_CHANNEL_STATE_IDLE 1
```

Description

This is macro AVCTP_CHANNEL_STATE_IDLE.

AVCTP_ERROR_BAD_STATE Macro

File

avrcp.h

C

```
#define AVCTP_ERROR_BAD_STATE 0x31
```

Description

This is macro AVCTP_ERROR_BAD_STATE.

AVCTP_ERROR_SUCCESS Macro

File

avrcp.h

C

```
#define AVCTP_ERROR_SUCCESS 0
```

Description

This is macro AVCTP_ERROR_SUCCESS.

AVCTP_EVT_CHANNEL_CONNECTED Macro

File

avctp.h

C

```
#define AVCTP_EVT_CHANNEL_CONNECTED 1   ///< This event is generated when a channel between two AVCTP
entities has been established.
```

Description

< This event is generated when a channel between two AVCTP entities has been established.

AVCTP_EVT_CHANNEL_DISCONNECTED Macro

File

avctp.h

C

```
#define AVCTP_EVT_CHANNEL_DISCONNECTED 2   ///< This event is generated when a channel between two AVCTP
entities has been terminated.
```

Description

< This event is generated when a channel between two AVCTP entities has been terminated.

AVCTP_EVT_COMMAND_CANCELLED Macro

File

avctp.h

C

```
#define AVCTP_EVT_COMMAND_CANCELLED 54  ///< This event is generated when a command has been canceled.
```

Description

< This event is generated when a command has been canceled.

AVCTP_EVT_COMMAND_RECEIVED Macro

File

avctp.h

C

```
#define AVCTP_EVT_COMMAND_RECEIVED 50  ///< This event is generated when a local device received a command.
```

Description

< This event is generated when a local device received a command.

AVCTP_EVT_COMMAND_SENT Macro

File

avctp.h

C

```
#define AVCTP_EVT_COMMAND_SENT 52  ///< This event is generated when a local device finished sending a
command.
```

Description

< This event is generated when a local device finished sending a command.

AVCTP_EVT_CONNECTION_FAILED Macro

File

avctp.h

C

```
#define AVCTP_EVT_CONNECTION_FAILED 3   ///< This event is generated when a local device failed to create a
channel between two AVCTP entities.
```

Description

< This event is generated when a local device failed to create a channel between two AVCTP entities.

AVCTP_EVT_NOTHING Macro

File

avctp.h

C

```
#define AVCTP_EVT_NOTHING 0
```

Description

addtogroup avrcp @{

@name Events

details The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.

AVCTP_EVT_RESPONE_RECEIVED Macro

File

avctp.h

C

```
#define AVCTP_EVT_RESPONE_RECEIVED 51  ///< This event is generated when a local device received a response.
```

Description

< This event is generated when a local device received a response.

AVCTP_EVT_RESPONSE_CANCELLED Macro

File

avctp.h

C

```
#define AVCTP_EVT_RESPONSE_CANCELLED 55  ///< This event is generated when a response has been canceled.
```

Description

< This event is generated when a response has been canceled.

AVCTP_EVT_RESPONSE_SENT Macro

File

avctp.h

C

```
#define AVCTP_EVT_RESPONSE_SENT 53  ///< This event is generated when a local device finished sending a
response.
```

Description

< This event is generated when a local device finished sending a response.

AVCTP_MANAGER_STATE_IDLE Macro

File

avctp.h

C

```
#define AVCTP_MANAGER_STATE_IDLE 0
```

Description

This is macro AVCTP_MANAGER_STATE_IDLE.

AVCTP_MESSAGE_PACKET_TYPE_CONTINUE Macro

File

avrcp.h

C

```
#define AVCTP_MESSAGE_PACKET_TYPE_CONTINUE 2
```

Description

This is macro AVCTP_MESSAGE_PACKET_TYPE_CONTINUE.

AVCTP_MESSAGE_PACKET_TYPE_END Macro

File

avrcp.h

C

```
#define AVCTP_MESSAGE_PACKET_TYPE_END 3
```

Description

This is macro AVCTP_MESSAGE_PACKET_TYPE_END.

AVCTP_MESSAGE_PACKET_TYPE_SINGLE Macro

File

avrcp.h

C

```
#define AVCTP_MESSAGE_PACKET_TYPE_SINGLE 0
```

Description

This is macro AVCTP_MESSAGE_PACKET_TYPE_SINGLE.

AVCTP_MESSAGE_PACKET_TYPE_START Macro

File

avrcp.h

C

```
#define AVCTP_MESSAGE_PACKET_TYPE_START 1
```

Description

This is macro AVCTP_MESSAGE_PACKET_TYPE_START.

AVCTP_MESSAGE_TYPE_COMMAND Macro

File

avrcp.h

C

```
#define AVCTP_MESSAGE_TYPE_COMMAND 0
```

Description

This is macro AVCTP_MESSAGE_TYPE_COMMAND.

AVCTP_MESSAGE_TYPE_RESPONSE Macro

File

avrcp.h

C

```
#define AVCTP_MESSAGE_TYPE_RESPONSE 1
```

Description

This is macro AVCTP_MESSAGE_TYPE_RESPONSE.

AVCTP_TRANSPORT_FLAG_RX_MESSAGE_STARTED Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_FLAG_RX_MESSAGE_STARTED 4
```

Description

This is macro AVCTP_TRANSPORT_FLAG_RX_MESSAGE_STARTED.

AVCTP_TRANSPORT_FLAG_SENDING Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_FLAG_SENDING 2
```

Description

This is macro AVCTP_TRANSPORT_FLAG_SENDING.

AVCTP_TRANSPORT_STATE_CONNECTED Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_STATE_CONNECTED 3
```

Description

This is macro AVCTP_TRANSPORT_STATE_CONNECTED.

AVCTP_TRANSPORT_STATE_CONNECTING Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_STATE_CONNECTING 2
```

Description

This is macro AVCTP_TRANSPORT_STATE_CONNECTING.

AVCTP_TRANSPORT_STATE_DISCONNECTING Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_STATE_DISCONNECTING 4
```

Description

This is macro AVCTP_TRANSPORT_STATE_DISCONNECTING.

AVCTP_TRANSPORT_STATE_FREE Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_STATE_FREE 0
```

Description

This is macro AVCTP_TRANSPORT_STATE_FREE.

AVCTP_TRANSPORT_STATE_IDLE Macro

File

avctp.h

C

```
#define AVCTP_TRANSPORT_STATE_IDLE 1
```

Description

This is macro AVCTP_TRANSPORT_STATE_IDLE.

AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET Macro

File

avdtp.h

C

```
#define AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET 1
```

Description

This is macro AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET.

AVDTP Functions

_bt_avdtp_add_param_byte Function

File

avdtp_control.h

C

```
bt_bool _bt_avdtp_add_param_byte(bt_avdtp_control_cmd_t* command, bt_byte value);
```

Description

This is function _bt_avdtp_add_param_byte.

_bt_avdtp_add_param_uint Function

File

avdtp_control.h

C

```
bt_bool _bt_avdtp_add_param_uint(bt_avdtp_control_cmd_t* command, bt_uint value);
```

Description

This is function _bt_avdtp_add_param_uint.

_bt_avdtp_allocate_buffers Function

File

avdtp_private.h

C

```
void _bt_avdtp_allocate_buffers();
```

Description

This is function _bt_avdtp_allocate_buffers.

_bt_avdtp_allocate_cmd Function

File

avdtp_private.h

C

```
bt_avdtp_control_cmd_t* _bt_avdtp_allocate_cmd(bt_byte msg_type);
```

Description

This is function _bt_avdtp_allocate_cmd.

_bt_avdtp_allocate_sep Function

File

avdtp_private.h

C

```
bt_avdtp_sep_t* _bt_avdtp_allocate_sep();
```

Description

This is function _bt_avdtp_allocate_sep.

_bt_avdtp_allocate_sep_config Function

File

avdtp_private.h

C

```
bt_avdtp_sep_capabilities_t* _bt_avdtp_allocate_sep_config();
```

Description

This is function _bt_avdtp_allocate_sep_config.

_bt_avdtp_allocate_stream Function

File

avdtp_private.h

C

```
bt_avdtp_stream_t* _bt_avdtp_allocate_stream(bt_avdtp_mgr_t* mgr);
```

Description

This is function _bt_avdtp_allocate_stream.

_bt_avdtp_allocate_transport_channel Function

File

avdtp_private.h

C

```
bt_avdtp_transport_channel_t* _bt_avdtp_allocate_transport_channel(bt_avdtp_mgr_t* mgr, bt_byte
strm_handle, bt_byte tc_id, bt_byte tc_type);
```

Description

This is function _bt_avdtp_allocate_transport_channel.

_bt_avdtp_allocate_transport_session_id Function

File

avdtp_private.h

C

```
bt_byte _bt_avdtp_allocate_transport_session_id(bt_avdtp_mgr_t* mgr, bt_byte ts_id);
```

Description

This is function _bt_avdtp_allocate_transport_session_id.

_bt_avdtp_begin_tc_channel_operation Function

File

avdtp_private.h

C

```
void _bt_avdtp_begin_tc_channel_operation(bt_byte opcode, bt_avdtp_transport_op_callback_fp callback, void*
callback_param);
```

Description

This is function _bt_avdtp_begin_tc_channel_operation.

_bt_avdtp_commit_tc_channel_operation Function

File

avdtp_private.h

C

```
void _bt_avdtp_commit_tc_channel_operation();
```

Description

This is function _bt_avdtp_commit_tc_channel_operation.

_bt_avdtp_control_channel_accept_handler Function

File

avdtp_private.h

C

```
void _bt_avdtp_control_channel_accept_handler(bt_avdtp_control_channel_t* channel, bt_byte cmd_code,
bt_byte trans_id, bt_byte* data, bt_int len, bt_int* offset);
```

Description

This is function _bt_avdtp_control_channel_accept_handler.

_bt_avdtp_control_channel_cmd_handler Function

File

avdtp_private.h

C

```
void _bt_avdtp_control_channel_cmd_handler(bt_avdtp_control_channel_t* channel, bt_byte cmd_code, bt_byte
trans_id, bt_byte* data, bt_int len, bt_int* offset);
```

Description

This is function _bt_avdtp_control_channel_cmd_handler.

_bt_avdtp_control_channel_event_handler Function

File

avdtp_control.h

C

```
void _bt_avdtp_control_channel_event_handler(bt_avdtp_control_channel_t* channel, bt_byte evt, void*
evt_param);
```

Description

This is function _bt_avdtp_control_channel_event_handler.

_bt_avdtp_control_channel_reject_handler Function

File

avdtp_private.h

C

```
void _bt_avdtp_control_channel_reject_handler(bt_avdtp_control_channel_t* channel, bt_byte cmd_code,
bt_byte trans_id, bt_byte* data, bt_int len, bt_int* offset);
```

Description

This is function _bt_avdtp_control_channel_reject_handler.

_bt_avdtp_execute_tc_channel_operation Function

File

avdtp_private.h

C

```
void _bt_avdtp_execute_tc_channel_operation();
```

Description

This is function _bt_avdtp_execute_tc_channel_operation.

_bt_avdtp_find_listening_stream Function

File

avdtp_private.h

C

```
bt_avdtp_stream_t* _bt_avdtp_find_listening_stream(bt_avdtp_mgr_t* mgr, bt_byte sep_id);
```

Description

This is function _bt_avdtp_find_listening_stream.

_bt_avdtp_find_stream Function

File

avdtp_private.h

C

```
bt_avdtp_stream_t* _bt_avdtp_find_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

This is function _bt_avdtp_find_stream.

_bt_avdtp_find_stream_by_remote_sep_id Function

File

avdtp_private.h

C

```
bt_avdtp_stream_t* _bt_avdtp_find_stream_by_remote_sep_id(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr,
bt_byte remote_seid);
```

Description

This is function _bt_avdtp_find_stream_by_remote_sep_id.

_bt_avdtp_find_stream_by_sep_id Function

File

avdtp_private.h

C

```
bt_avdtp_stream_t* _bt_avdtp_find_stream_by_sep_id(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_byte
sep_id);
```

Description

This is function _bt_avdtp_find_stream_by_sep_id.

_bt_avdtp_free_cmd Function

File

avdtp_private.h

C

```
void _bt_avdtp_free_cmd(bt_avdtp_control_cmd_t* command);
```

Description

This is function _bt_avdtp_free_cmd.

_bt_avdtp_free_sep_config Function

File

avdtp_private.h

C

```
void _bt_avdtp_free_sep_config(bt_avdtp_sep_capabilities_t* sep_config);
```

Description

This is function _bt_avdtp_free_sep_config.

_bt_avdtp_free_stream Function

File

avdtp_private.h

C

```
void _bt_avdtp_free_stream(bt_avdtp_stream_t* strm);
```

Description

This is function _bt_avdtp_free_stream.

_bt_avdtp_free_transport_channel Function

File

avdtp_private.h

C

```
void _bt_avdtp_free_transport_channel(bt_avdtp_transport_channel_t* channel);
```

Description

This is function _bt_avdtp_free_transport_channel.

_bt_avdtp_get_control_channel Function

File

avdtp_control.h

C

```
bt_avdtp_control_channel_t* _bt_avdtp_get_control_channel(struct _bt_avdtp_mgr_t* mgr, bt_bdaddr_t*
remote_addr);
```

Description

This is function _bt_avdtp_get_control_channel.

_bt_avdtp_init_cmd_buffers Function

File

avdtp_private.h

C

```
void _bt_avdtp_init_cmd_buffers();
```

Description

This is function _bt_avdtp_init_cmd_buffers.

_bt_avdtp_init_sep_config_buffers Function

File

avdtp_private.h

C

```
void _bt_avdtp_init_sep_config_buffers();
```

Description

This is function _bt_avdtp_init_sep_config_buffers.

_bt_avdtp_init_signal Function

File

avdtp_private.h

C

```
void _bt_avdtp_init_signal();
```

Description

This is function _bt_avdtp_init_signal.

_bt_avdtp_is_sep_inuse Function

File

avdtp_private.h

C

```
bt_bool _bt_avdtp_is_sep_inuse(bt_avdtp_mgr_t* mgr, bt_avdtp_sep_t* sep, bt_bdaddr_t* remote_addr);
```

Description

This is function _bt_avdtp_is_sep_inuse.

_bt_avdtp_open_control_channel_ex Function

File

avdtp_private.h

C

```
bt_bool _bt_avdtp_open_control_channel_ex(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_uint
acl_config);
```

Description

This is function _bt_avdtp_open_control_channel_ex.

_bt_avdtp_open_control_channel_ex Function

File

avdtp_control.h

C

```
bt_bool _bt_avdtp_open_control_channel_ex(struct _bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_uint
acl_config);
```

Description

This is function _bt_avdtp_open_control_channel_ex.

_bt_avdtp_register_transport_channel_for_operation Function

File

avdtp_private.h

C

```
void _bt_avdtp_register_transport_channel_for_operation(bt_avdtp_transport_channel_t* channel);
```

Description

This is function _bt_avdtp_register_transport_channel_for_operation.

_bt_avdtp_send_command Function

File

avdtp_private.h

C

```
void _bt_avdtp_send_command(bt_avdtp_control_channel_t* channel, bt_avdtp_control_cmd_t* cmd);
```

Description

This is function _bt_avdtp_send_command.

_bt_avdtp_send_media_packet Function

File

avdtp_private.h

C

```
bt_bool _bt_avdtp_send_media_packet(bt_avdtp_stream_t* strm);
```

Description

This is function _bt_avdtp_send_media_packet.

_bt_avdtp_set_signal Function

File

avdtp_private.h

C

```
void _bt_avdtp_set_signal();
```

Description

This is function _bt_avdtp_set_signal.

_bt_avdtp_transport_l2cap_read_data_callback Function

File

avdtp_private.h

C

```
void _bt_avdtp_transport_l2cap_read_data_callback(struct _bt_l2cap_channel_t * pch, bt_byte_p pdata, bt_int
len);
```

Description

This is function _bt_avdtp_transport_l2cap_read_data_callback.

_bt_avdtp_transport_l2cap_state_changed_callback Function

File

avdtp_private.h

C

```
void _bt_avdtp_transport_l2cap_state_changed_callback(bt_l2cap_channel_t* pch, bt_int new_state, void*
param);
```

Description

This is function _bt_avdtp_transport_l2cap_state_changed_callback.

_bt_avdtp_write_caps Function

File

avdtp_private.h

C

```
bt_bool _bt_avdtp_write_caps(bt_avdtp_mgr_t* mgr, const bt_avdtp_sep_capabilities_t* caps,
bt_avdtp_stream_t* strm, bt_avdtp_control_cmd_t* command);
```

Description

This is function _bt_avdtp_write_caps.

bt_avdtp_abort_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_abort_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Suspend a stream. ingroup avdtp

details This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE. As a result of this operation the AVDTP_EVT_ABORT_STREAM_COMPLETED event will be generated. This operation cannot be rejected. The p evt_param.abort_stream_requested.err_code is always == AVDTP_ERROR_SUCCESS.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_add_media_rx_buffer Function

File

avdtp.h

C

```
bt_bool bt_avdtp_add_media_rx_buffer(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_media_packet_t* buffer);
```

Description

brief Add a media packet buffer to a receive queue ingroup avdtp

details The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue or none of the buffers is large enough the received packets is dropped. Each buffer has a field (data_len) that holds the length of the received buffer. This field is never 0 if the buffer contains a packet. If a channel closed regardless of what has caused that and there are still buffers in the queue AVDTP generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event for each buffer and sets the data_len to 0. This is to inform the AVDTP consumer that the buffer has not been used and can be, for example, deallocated. This function adds a buffer to the receive queue.

param mgr AVDTP manager. param strm_handle Stream handle. param buffer Pointer to a structure that holds the buffer and its parameters.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

bt_avdtp_add_media_tx_buffer Function

File

avdtp.h

C

```
bt_bool bt_avdtp_add_media_tx_buffer(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_media_packet_t* buffer);
```

Description

brief Add a media packet buffer to a send queue ingroup avdtp

details When the consumer of AVDTP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. When the packet has been successfully sent a AVDTP_EVT_MEDIA_PACKET_SENT is generated. Otherwise a AVDTP_EVT_MEDIA_PACKET_SEND_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as AVDTP has removed it from the queue and gave up control over it. As in the case of received buffers, if a channel closed regardless of what has caused that and there are still buffers in the queue AVDTP generates a AVDTP_EVT_MEDIA_PACKET_SENT event for each buffer and sets the data_len field to 0. This is to inform the AVDTP consumer that the buffer has not been used and can be, for example, deallocated.

param mgr AVDTP manager. param strm_handle Stream handle. param buffer Pointer to a structure that holds the buffer and its parameters.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

bt_avdtp_cancel_listen Function

File

avdtp.h

C

```
void bt_avdtp_cancel_listen(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_byte sep_id);
```

Description

brief Cancel listening for incoming connections. ingroup avdtp

details This function removes a SEP from a list of SEPS which a stream can use for incoming requests.

param mgr AVDTP manager. param strm_handle Stream handle. param sep_id Local SEP ID.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avdtp_close_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_close_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Close a stream. ingroup avdtp

details This function tries to close a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN or AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the AVDTP_EVT_CLOSE_STREAM_COMPLETED event will be generated. If the stream has been closed the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == the error code sent by the remote.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_create_stream Function

File

avdtp.h

C

```
bt_byte bt_avdtp_create_stream(bt_avdtp_mgr_t* mgr);
```

Description

brief Create a stream. ingroup avdtp

details This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort.

param mgr AVDTP manager.

return li c Stream handle if the function succeeds. li c 0 otherwise.

bt_avdtp_destroy_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_destroy_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Destroy a stream. ingroup avdtp

details This function frees memory used by the stream. The stream has to exist and be in the "idle" state for this function to succeed. I.e. the stream has to be closed or aborted before this function can be called.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avdtp_disconnect Function

File

avdtp.h

C

```
bt_bool bt_avdtp_disconnect(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr);
```

Description

brief Disconnect from a remote device. ingroup avdtp

details This function closes a control and transport channels on all streams associated with the remote device specified by the p remote_addr. As a result of this operation the following events will be generated: @arg AVDTP_EVT_MEDIA_PACKET_RECEIVED: if a stream's receive queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg AVDTP_EVT_MEDIA_PACKET_SENT: if a stream's send queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg AVDTP_EVT_STREAM_CLOSED: this event is generate if a stream is in "closing" state as a result of a request from the remote device or bt_avdtp_close_stream call before bt_avdtp_disconnect call @arg AVDTP_EVT_STREAM_ABORTED: this event is generated if a stream is in "active" state at the time of bt_avdtp_disconnect call.

param mgr AVDTP manager. param remote_addr The address of a remote device.

return li c TRUE if disconnection has been started. li c FALSE otherwise. No events will be generated.

bt_avdtp_discover Function

File

avdtp.h

C

```
bt_bool bt_avdtp_discover(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr);
```

Description

brief Discover SEPs on a remote device. ingroup avdtp

details This function asks the remote device to send a list of all available SEPs. As a result of this operation the following events will be generated: @arg AVDTP_EVT_SEP_INFO_RECEIVED: this event is generated for every SEP received from the remote device. the p evt_param.sep_info_received contains SEP information. @arg AVDTP_EVT_DISCOVER_COMPLETED: this event is generated after last AVDTP_EVT_SEP_INFO_RECEIVED if the remote accepted the request and the p evt_param.discover_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.discover_completed.err_code == the error code sent by the remote.

param mgr AVDTP manager. param remote_addr The address of a remote device.

return li c TRUE if discover request has been sent. li c FALSE otherwise. No events will be generated.

bt_avdtp_find_codec Function

File

avdtp.h

C

```
bt_avdtp_codec_t* bt_avdtp_find_codec(bt_avdtp_mgr_t* mgr, bt_byte codec_type);
```

Description

brief Find a codec ingroup avdtp

details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This

function returns a pointer to a structure that holds a pointer to a codec's callback function.

param mgr AVDTP manager. param codec_type Codec type. The codec_type can be one of the following values: @arg AVDTP_CODEC_TYPE_SBC: SBC @arg AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) @arg AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) @arg AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) @arg AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails if a callback for a codec type specified in the p codec parameter li has not been previously registered with bt_avdtp_register_codec.

bt_avdtp_get_capabilities Function

File

avdtp.h

C

```
bt_bool bt_avdtp_get_capabilities(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_byte seid_acp);
```

Description

brief Get remote SEP capabilities. ingroup avdtp

details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after AVDTP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote.

param mgr AVDTP manager. param remote_addr The address of a remote device. param seid_acp The ID of a remote SEP.

return li c TRUE if discover request has been sent. li c FALSE otherwise. No events will be generated.

bt_avdtp_get_configuration Function

File

avdtp.h

C

```
bt_bool bt_avdtp_get_configuration(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get stream configuration. ingroup avdtp

details This function requests stream configuration from a remote device. As a result of this operation the following events will be generated: @arg AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: this event is generated if the remote accepted the request. the p ebt_para.sep_capabilities_received.caps will contain current stream configuration. @arg AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: If the remote accepted the request the p evt_param.get_stream_configuration_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_stream_configuration_completed.err_code == the error code sent by the remote.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_get_hci_connection Function

File

avdtp.h

C

```
bt_hci_conn_state_t* bt_avdtp_get_hci_connection(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get HCI connection for a stream ingroup avdtp

details This function returns a pointer to a structure that describes an HCI connection a stream is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call bt_hci_disconnect.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist or there is no HCI connection between local and remote devices associated with the stream.

note This function has not been implemented.

bt_avdtp_get_mgr Function

File

avdtp.h

C

```
bt_avdtp_mgr_t* bt_avdtp_get_mgr();
```

Description

brief Return a pointer to an instance of the AVDTP manager. ingroup avdtp

details This function returns a pointer to an instance of the AVDTP manager. There is only one instance of the manager allocated by the stack. The pointer is passed as the first parameter to all AVDTP functions.

bt_avdtp_get_sep Function

File

avdtp.h

C

```
bt_avdtp_sep_t* bt_avdtp_get_sep(bt_avdtp_mgr_t* mgr, bt_byte sep_id);
```

Description

brief Get a SEP info by its ID. ingroup avdtp

details This function returns a pointer to bt_avdtp_sep_t structure that describes a SEP previously registered with bt_avdtp_register_sep.

param mgr AVDTP manager. param sep_id The ID of a SEP.

return li c Pointer to bt_avdtp_sep_t if the SEP is in the list of registered SEPs. li c NULL otherwise.

bt_avdtp_get_stream_codec_config Function

File

avdtp.h

C

```
void* bt_avdtp_get_stream_codec_config(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get the configuration of the codec currently used with the stream. ingroup avdtp

details This function returns a pointer to a structure that contains configuration of the codec currently used with the stream. The structure returned depends on the codec. The dotstack defines structures only for SBC, MPEG-1,2 and MPEG-2,4 AAC codecs: @arg SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h) @arg MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h) @arg MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h)

param mgr AVDTP manager. param strm_handle Stream handle.

return li The codec's configuration if strm_handle specifies a valid stream and the stream is in one of the following state:

AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING

li NULL otherwise.

bt_avdtp_get_stream_codec_type Function

File

avdtp.h

C

```
bt_byte bt_avdtp_get_stream_codec_type(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get the type of the codec currently used with the stream. ingroup avdtp

details This function returns the type of the codec currently used with the stream.

param mgr AVDTP manager. param strm_handle Stream handle.

return @arg The type of the codec if strm_handle specifies a valid stream and the stream is in one of the following states:

AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING

@arg The result will be one of the following values:

AVDTP_CODEC_TYPE_SBC: SBC AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files)
AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

@arg 0xFF otherwise.

bt_avdtp_get_stream_config Function

File

avdtp.h

C

```
bt_avdtp_sep_capabilities_t* bt_avdtp_get_stream_config(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get stream's configuration. ingroup avdtp

details This function returns a pointer to a structure holding the current configuration of stream.

param mgr AVDTP manager. param strm_handle Stream handle.

return li The stream's configuration if strm_handle specifies a valid stream and the stream is in one of the following state:

AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING

li NULL otherwise.

bt_avdtp_get_stream_local_sep_id Function

File

avdtp.h

C

```
bt_byte bt_avdtp_get_stream_local_sep_id(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get stream's local SEP ID. ingroup avdtp

details This function returns the ID of the local SEP associated with the stream.

param mgr AVDTP manager. param strm_handle Stream handle.

return li The ID of the local SEP if strm_handle specifies a valid stream. li 0 otherwise.

bt_avdtp_get_stream_remote_address Function

File

avdtp.h

C

```
bt_bdaddr_t* bt_avdtp_get_stream_remote_address(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get stream's remote BT address. ingroup avdtp

details This function returns the address of the remote device associated with the stream.

param mgr AVDTP manager. param strm_handle Stream handle.

return li The address of the remote device if strm_handle specifies a valid stream. li NULL otherwise.

bt_avdtp_get_stream_remote_sep_id Function

File

avdtp.h

C

```
bt_byte bt_avdtp_get_stream_remote_sep_id(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get stream's remote SEP ID. ingroup avdtp

details This function returns the ID of the remote SEP associated with the stream.

param mgr AVDTP manager. param strm_handle Stream handle.

return li The ID of the remote SEP if strm_handle specifies a valid stream. li 0 otherwise.

bt_avdtp_get_stream_state Function

File

avdtp.h

C

```
bt_byte bt_avdtp_get_stream_state(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Get local stream state. ingroup avdtp

details This function returns local state of a stream specified by the p strm_handle. No request is sent to the remote party.

param mgr AVDTP manager. param strm_handle Stream handle.

return The state of the stream. The result will be one of the following values: @arg AVDTP_STREAM_STATE_IDLE: The stream is idle. This can mean two things. The stream specified by p strm_handle does not exist or the stream is closed. @arg AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS: The stream is opening transport channels. @arg AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS: The stream is closing transport channels. @arg AVDTP_STREAM_STATE_CONFIGURED: The stream has been configured. @arg AVDTP_STREAM_STATE_OPEN: The stream has been opened. @arg AVDTP_STREAM_STATE_STREAMING: The stream has been started. Depending on the local SEP type (source or sink) it means that the stream is can send or receive media packets. @arg AVDTP_STREAM_STATE_CLOSING: The stream is closing. This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state. @arg AVDTP_STREAM_STATE_ABORTING: The stream is aborting. This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.

bt_avdtp_init Function

File

avdtp.h

C

```
void bt_avdtp_init();
```

Description

brief Initialize the AVDTP layer. ingroup avdtp

details This function initializes the AVDTP layer of the stack. It must be called prior to any other AVDTP function can be called.

bt_avdtp_listen Function

File

avdtp.h

C

```
bt_bool bt_avdtp_listen(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_byte sep_id);
```

Description

brief Listen for incoming connections. ingroup avdtp

details This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream.

param mgr AVDTP manager. param strm_handle Stream handle. param sep_id Local SEP ID.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avdtp_open_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_open_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Open a stream. ingroup avdtp

details This function tries to open a stream by sending a request to the remote party. The stream has to be already configured with a bt_avdtp_set_configuration call. As a result of this operation the AVDTP_EVT_OPEN_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.open_stream_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to open the stream, the p evt_param.open_stream_completed.err_code == the error code sent by the remote.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_reconfigure_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_reconfigure_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_avdtp_sep_capabilities_t*
caps);
```

Description

brief Reconfigure stream. ingroup avdtp

details This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event will be generated. If reconfiguration was a success the p evt_param.stream_reconfigure_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.stream_reconfigure_completed.err_code == the error code sent by the remote.

param mgr AVDTP manager. param strm_handle Stream handle. param caps New stream configuration.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_register_callback Function

File

avdtp.h

C

```
void bt_avdtp_register_callback(bt_avdtp_mgr_t* mgr, bt_avdtp_mgr_callback_fp callback, void*
callback_param);
```

Description

brief Register a AVDTP application callback. ingroup avdtp

details In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values:

@arg AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED:

Control channel disconnected. @arg AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local device received a response to Get SEP capabilities operation. @arg AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Set stream configuration operation. @arg AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Get stream configuration operation. @arg AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED: Local device received a response to Reconfigure stream operation. @arg AVDTP_EVT_OPEN_STREAM_COMPLETED: Local device received a response to Open stream operation. @arg AVDTP_EVT_START_STREAM_COMPLETED: Local device received a response to Start stream operation. @arg AVDTP_EVT_CLOSE_STREAM_COMPLETED: Local device received a response to Close stream operation. @arg AVDTP_EVT_SUSPEND_STREAM_COMPLETED: Local device received a response to Suspend stream operation. @arg AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: Local device received a response to Stream security control operation. @arg AVDTP_EVT_ABORT_STREAM_COMPLETED: Local device received a response to Abort stream operation. @arg AVDTP_EVT_SEP_INFO_RECEIVED: SEP information received. @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: SEP capabilities received. @arg AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: Stream configuration received.

@arg AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: Remote device requested stream configuration. @arg AVDTP_EVT_OPEN_STREAM_REQUESTED: Remote device requested to open a stream. @arg AVDTP_EVT_START_STREAM_REQUESTED: Remote device requested to start a stream. @arg AVDTP_EVT_CLOSE_STREAM_REQUESTED: Remote device requested to close a stream. @arg AVDTP_EVT_SUSPEND_STREAM_REQUESTED: Remote device requested to suspend a stream. @arg AVDTP_EVT_ABORT_STREAM_REQUESTED: Remote device requested to abort a stream. @arg AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED: Remote device requested to reconfigure a stream. @arg AVDTP_EVT_MEDIA_PACKET_RECEIVED: Remote device sent a media packet. @arg AVDTP_EVT_STREAM_CONFIGURED: A stream has been configured (This event is generated right after AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_RECONFIGURED: A stream has been re-configured (This event is generated right after AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_OPENED: A stream has been opened (This event is generated as a result of local or remote stream opening request). @arg AVDTP_EVT_STREAM_STARTED: A stream has been started (This event is generated right after AVDTP_EVT_START_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_CLOSED: A stream has been close (This event is generated right after AVDTP_EVT_CLOSE_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_SUSPENDED: A stream has been suspended (This event is generated right after AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_ABORTED: A stream has been aborted (This event is generated right after AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request. It is also generated if connection between devices has been terminated by means other than AVDTP signaling, e.g. devices going out of rage). @arg AVDTP_EVT_MEDIA_PACKET_SENT: The local device has successfully sent a media packet to the remote device. @arg AVDTP_EVT_MEDIA_PACKET_SEND_FAILED: The local device was not able to send a media packet to the remote device.

param mgr AVDTP manager. param callback The callback function that will be called when the AVDTP generates an event. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

bt_avdtp_register_codec Function

File

avdtp.h

C

```
bt_bool bt_avdtp_register_codec(bt_avdtp_mgr_t* mgr, bt_avdtp_codec_t* codec);
```

Description

brief Register a codec ingroup avdtp

details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This function adds a codec's callback function to an internal list.

param mgr AVDTP manager. param codec Pointer to a structure specifying codec type and a callback.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails if there already is a callback for a codec type specified in the p codec parameter.

bt_avdtp_register_sep Function

File

avdtp.h

C

```
bt_byte bt_avdtp_register_sep(bt_avdtp_mgr_t* mgr, bt_byte type, const bt_avdtp_sep_capabilities_t* caps);
```

Description

brief Register a SEP with the local AVDTP manager. ingroup avdtp

details This function is used to make a list of SEPs supported by the local ADVTP entity.

param mgr AVDTP manager. param type The type of a SEP. The type can be one of the following values: @arg AVDTP_SEP_TYPE_SOURCE: The SEP is a source. @arg AVDTP_SEP_TYPE_SINK: The SEP is a sink. param caps The capabilities of a SEP.

return li c ID of a SEP if the function succeeds. li c FALSE otherwise.

bt_avdtp_remove_media_rx_buffer Function

File

avdtp.h

C

```
bt_bool bt_avdtp_remove_media_rx_buffer(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_media_packet_t*
buffer);
```

Description

brief Remove a media packet buffer from a receive queue ingroup avdtp

details The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue or none of the buffers is large enough the received packets is dropped. Each buffer has a field (data_len) that holds the length of the received buffer. This field is never 0 if the buffer contains a packet. If a channel closed regardless of what has caused that and there are still buffers in the queue AVDTP generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event for each buffer and sets the data_len to 0. This is to inform the AVDTP consumer that the buffer has not been used and can be, for example, deallocated. This function removes a buffer from the receive queue.

param mgr AVDTP manager. param strm_handle Stream handle. param buffer Pointer to a structure that holds the buffer and its parameters.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

bt_avdtp_remove_media_tx_buffer Function

File

avdtp.h

C

```
bt_bool bt_avdtp_remove_media_tx_buffer(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_media_packet_t*
buffer);
```

Description

brief Remove a media packet buffer from a send queue ingroup avdtp

details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling bt_avdtp_remove_media_tx_buffer.

param mgr AVDTP manager. param strm_handle Stream handle. param buffer Pointer to a structure that holds the buffer and its parameters.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

note This function has not been implemented.

bt_avdtp_security_control Function

File

avdtp.h

C

```
bt_bool bt_avdtp_security_control(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_byte* sc_data, bt_byte sc_data_len);
```

Description

brief Exchange content protection control data. ingroup avdtp

details This function tries to establish content protection by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE, AVDTP_STREAM_STATE_CLOSING, AVDTP_STREAM_STATE_ABORTING. As a result of this operation the AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event will be generated. If the stream's content protection data has been accepted by the remote party the p evt_param.security_control_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.security_control_completed.err_code == the error code sent by the remote.

note The dotstack does not support content protection. Although the request can be sent it will not affect the operation of the AVDTP in any way.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_set_configuration Function

File

avdtp.h

C

```
bt_bool bt_avdtp_set_configuration(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_bdaddr_t* remote_addr, bt_byte seid_int, bt_byte seid_acp, const bt_avdtp_sep_capabilities_t* caps);
```

Description

brief Set stream configuration. ingroup avdtp

details This function tries to configure a stream before opening it. As a result of this operation the AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event will be generated. If configuration was a success the p evt_param.set_stream_configuration_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.set_stream_configuration_completed.err_code == the error code sent by the remote and p evt_param.set_stream_configuration_completed.svc_category == the value of the first Service Category to fail.

param mgr AVDTP manager. param strm_handle Stream handle. param remote_addr The address of a remote device. param seid_int Local SEP ID. param seid_acp Remote SEP ID. param caps Stream configuration.

return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.

bt_avdtp_start Function

File

avdtp.h

C

```
bt_bool bt_avdtp_start(bt_avdtp_mgr_t* mgr);
```

Description

brief Start the AVDTP layer. ingroup avdtp

details This function makes the AVDTP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required.

param mgr AVDTP manager.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avdtp_start_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_start_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Start a stream. ingroup avdtp

details This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the AVDTP_EVT_START_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.start_stream_requested.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the p evt_param.start_stream_requested.err_code == the error code sent by the remote.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_suspend_stream Function

File

avdtp.h

C

```
bt_bool bt_avdtp_suspend_stream(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Suspend a stream. ingroup avdtp

details This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the AVDTP_EVT_SUSPEND_STREAM_COMPLETED event will be generated. If the stream has been suspended the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == the error code sent by the remote.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_unregister_codec Function

File

avdtp.h

C

```
bt_bool bt_avdtp_unregister_codec(bt_avdtp_mgr_t* mgr, bt_byte codec_type);
```

Description

brief Unregister a codec ingroup avdtp

details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This function removes a codec's callback function from an internal list.

param mgr AVDTP manager. param codec_type Codec type. The codec_type can be one of the following values: @arg AVDTP_CODEC_TYPE_SBC: SBC @arg AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) @arg AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) @arg AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) @arg AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails if a callback for a codec type specified in the p codec parameter li has not been previously registered with bt_avdtp_register_codec.

_bt_avdtp_add_param_uintn Function

File

avdtp_control.h

C

```
bt_bool _bt_avdtp_add_param_uintn(bt_avdtp_control_cmd_t* command, bt_uint value);
```

Description

This is function _bt_avdtp_add_param_uintn.

_bt_avdtp_read_caps Function

File

avdtp_private.h

C

```
bt_byte _bt_avdtp_read_caps(bt_avdtp_mgr_t* mgr, bt_avdtp_sep_capabilities_t* caps, bt_byte* data, bt_int
len, bt_int* offset, bt_byte* err_category);
```

Description

This is function _bt_avdtp_read_caps.

bt_avdtp_get_l2cap_channel Function

File

avdtp.h

C

```
bt_l2cap_channel_t* bt_avdtp_get_l2cap_channel(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_byte
transport_session);
```

Description

This is function bt_avdtp_get_l2cap_channel.

bt_avdtp_clear_media_tx_queue Function

File

avdtp.h

C

```
bt_bool bt_avdtp_clear_media_tx_queue(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

brief Clear send queue ingroup avdtp

details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. The consumer can remove all packets from the queue before they have been sent to a remote device by calling ::bt_avdtp_clear_media_tx_queue.

param mgr AVDTP manager. param strm_handle Stream handle.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

bt_avdtp_get_all_capabilities Function

File

avdtp.h

C

```
bt_bool bt_avdtp_get_all_capabilities(bt_avdtp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_byte seid_acp);
```

Description

brief Get remote SEP capabilities. ingroup avdtp

details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following

events will be generated: @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after AVDTP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote.

param mgr AVDTP manager. param remote_addr The address of a remote device. param seid_acp The ID of a remote SEP.

return li c TRUE if discover request has been sent. li c FALSE otherwise. No events will be generated.

bt_avdtp_get_stream_direction Function

File

avdtp.h

C

```
bt_byte bt_avdtp_get_stream_direction(bt_avdtp_mgr_t* mgr, bt_byte strm_handle);
```

Description

This is function bt_avdtp_get_stream_direction.

bt_avdtp_report_delay Function

File

avdtp.h

C

```
bt_bool bt_avdtp_report_delay(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_uint delay);
```

Description

brief Report delay value of a Sink to a Source. ingroup avdtp

details This function sends the delay value of a Sink to a Source. This enables synchronous playback of audio and video. Delay reports are always sent from the Sink to the Source. If the Sink's delay report has been accepted by the Source the p evt_param.delay_report_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.delay_report_completed.err_code == the error code sent by the Source.

param mgr AVDTP manager. param strm_handle Stream handle. param delay The delay value in 1/10 milliseconds.

return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.

bt_avdtp_set_media_tx_queue_limit Function

File

avdtp.h

C

```
bt_bool bt_avdtp_set_media_tx_queue_limit(bt_avdtp_mgr_t* mgr, bt_byte strm_handle, bt_uint limit);
```

Description

brief Set limit on the send queue ingroup avdtp

details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. By default the send queue can contain unlimited number of packets. The consumer can set a limit on how many packets are held in the queue. In this case when new packet is added to the queue and the length of the queue exceeds the set limit the first packet is removed from the queue. The removed packet is not send to the remote device.

param mgr AVDTP manager. param strm_handle Stream handle. param limit Queue limit.

return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails only if a stream specified by the p strm_handle parameter li does not exist. The stream can be in any state to call this function.

AVDTP Data Types and Constants

_bt_avdtp_codec_op_param_u Union

File

avdtp.h

C

```
union _bt_avdtp_codec_op_param_u {
  bt_avdtp_codec_op_parse_config_t parse;
  bt_avdtp_codec_op_serialize_config_t serialize;
  bt_avdtp_codec_op_parse_packet_t parse_packet;
};
```

Members

Members	Description
bt_avdtp_codec_op_parse_config_t parse;	< Valid if operation is AVDTP_CODEC_OPCODE_PARSE_CONFIG.
bt_avdtp_codec_op_serialize_config_t serialize;	< Valid if operation is AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG.
bt_avdtp_codec_op_parse_packet_t parse_packet;	< This member is currently not used.

Description

brief Parameter to a codec handler. ingroup avdtp

details This union is used to pass operation specific data to a codec handler. Which member of the union points to a valid structure depends on the operation.

_bt_avdtp_sep_t Structure

File

avdtp.h

C

```
struct _bt_avdtp_sep_t {
  bt_avdtp_sep_t* next_sep;
  bt_byte id;
  bt_byte type;
  const bt_avdtp_sep_capabilities_t* caps;
  bt_byte state;
};
```

Members

Members	Description
bt_avdtp_sep_t* next_sep;	< Pointer to next SEP
bt_byte id;	< ID of the SEP
bt_byte type;	< Type of the SEP. This can be one of the following values: < li AVDTP_SEP_TYPE_SOURCE < li AVDTP_SEP_TYPE_SINK
const bt_avdtp_sep_capabilities_t* caps;	< SEP capabilities
bt_byte state;	< State of the SEP buffer. This can be one of the following values: < li AVDTP_SEP_STATE_FREE < li AVDTP_SEP_STATE_IDLE

Description

brief SEP description ingroup avdtp

details This structure is used to hold information about SEPs available on a local device.

_bt_media_packet_t Structure

File

avdtp.h

C

```
struct _bt_media_packet_t {
  bt_media_packet_t* next_packet;
```

```
  bt_byte version;
  bt_byte csrc_count;
  bt_bool marker;
  bt_byte payload_type;
  bt_uint seq_number;
  bt_ulong timestamp;
  bt_ulong ssrc;
  bt_ulong* csrc_list;
  bt_byte* data;
  bt_int data_len;
  bt_int max_data_len;
  bt_cp_header_t cp_header;
};
```

Members

Members	Description
bt_media_packet_t* next_packet;	< Pointer to next buffer
bt_byte version;	< Version of the RTP implementation.
bt_byte csrc_count;	< The CSRC count contains the number of CSRC identifiers that follow the fixed header.
bt_bool marker;	< The interpretation of the marker is defined by a profile. It is < intended to allow significant events such as frame boundaries to < be marked in the packet stream.
bt_byte payload_type;	< This field identifies the format of the RTP payload and determines its interpretation by the application. < A profile specifies default static mapping of payload type codes to payload formats.
bt_uint seq_number;	< The sequence number increments by one for each media packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence.
bt_ulong timestamp;	< The Time Stamp reflects the sampling instant of the first octet in the media packet.
bt_ulong ssrc;	< The SSRC field identifies the synchronization source. This < identifier is chosen randomly, with the intent that no two synchronization < sources, within the same media transport < session, shall have the same SSRC identifier.
bt_ulong* csrc_list;	< The CSRC list identifies the contributing sources for the payload contained in this packet.
bt_byte* data;	< Pointer to a buffer to store media data. This pointer must be allocated by the AVDTP consumer < before adding this structure to receive (::bt_avdtp_add_media_rx_buffer) or send queue (::bt_avdtp_add_media_tx_buffer).
bt_int data_len;	< Lentgth of the media data.
bt_int max_data_len;	< Maximum length of the media data that can be stored in the buffer pointed by the c data member.
bt_cp_header_t cp_header;	< Content protection header.

Description

brief Media packet buffer ingroup avdtp

details This structure is used to receive and send media packet from/to the remote device. See more information about usage of this structure in descriptions of ::bt_avdtp_add_media_rx_buffer and ::bt_avdtp_add_media_tx_buffer.

bt_avdtp_codec_handler_fp Type

File

avdtp.h

C

```
typedef bt_byte (* bt_avdtp_codec_handler_fp)(struct _bt_avdtp_codec_t* codec, bt_byte opcode,
bt_avdtp_codec_op_param_t* op_param, struct _bt_avdtp_mgr_t* mgr);
```

Description

brief Codec handler. ingroup avdtp

details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request.n This typedef defines the interace for the callback function.

param codec A pointer to a structure that describes a codec. param opcode The code of an operation to execute. The c opcode can be one of the following values: @arg AVDTP_CODEC_OPCODE_PARSE_CONFIG: The handler has to parse configuration received from the remote device

and store it in a structure defined by the consumer. @arg AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG: The handler has to serialize the data from a consumer defined structure to a format suitable for sending as a part of a AVDTP request. param op_param A pointer to the operation's specific parameters. Parameters are passed as a pointer to the bt_avdtp_codec_op_param_t union. Which member of the union points to a valid structure depends on the value of the c opcode: @arg bt_avdtp_codec_op_param_t::parse: Valid if c opcode == AVDTP_CODEC_OPCODE_PARSE_CONFIG. @arg bt_avdtp_codec_op_param_t::serialize: Valid if c opcode == AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG. param mgr AVDTP manager.

bt_avdtp_codec_op_decode_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_codec_op_decode_t {
  void* codec_config;
  bt_byte* in_buffer;
  bt_int in_len;
  bt_byte* out_buffer;
  bt_int out_len;
} bt_avdtp_codec_op_decode_t;
```

Description

There is currently no use for this structure.

bt_avdtp_codec_op_encode_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_codec_op_encode_t {
  void* codec_config;
  bt_byte* in_buffer;
  bt_int in_len;
  bt_byte* out_buffer;
  bt_int out_len;
} bt_avdtp_codec_op_encode_t;
```

Description

There is currently no use for this structure.

bt_avdtp_codec_op_param_t Type

File

avdtp.h

C

```
typedef union _bt_avdtp_codec_op_param_u bt_avdtp_codec_op_param_t;
```

Description

This is type bt_avdtp_codec_op_param_t.

bt_avdtp_codec_op_parse_config_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_codec_op_parse_config_t {
  void* codec_config;
  bt_byte codec_config_max_size;
  bt_byte* buffer;
  bt_int buffer_len;
```

```
  bt_int offset;
} bt_avdtp_codec_op_parse_config_t;
```

Members

Members	Description
void* codec_config;	< A pointer to a structure defined by the AVDTP consumer where codec's configuration < will be stored by the handler. The format of the structure is totally up to < the AVDTP consumer. The dotstack defines such structures for SBC, MPEG-1,2 and MPEG-2,4 AAC: < @arg SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h) < @arg MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h) < @arg MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h)
bt_byte codec_config_max_size;	< The maximum size of a buffer pointed to by c codec_config field.
bt_byte* buffer;	< A pointer to a buffer holding codec's configuration in OTA format.
bt_int buffer_len;	< The length of the c buffer.
bt_int offset;	< The c buffer points to a complete packet received from the remoted device. The c offset < points to a location in the c buffer where codec's configuration starts.

Description

brief Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation. ingroup avdtp

details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::parse - when ADVDTP needs to parse codec's capabilities/configuration received from the remote device.

bt_avdtp_codec_op_parse_packet_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_codec_op_parse_packet_t {
  void* packet_info;
  bt_media_packet_t* packet;
} bt_avdtp_codec_op_parse_packet_t;
```

Description

There is currently no use for this structure.

bt_avdtp_codec_op_serialize_config_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_codec_op_serialize_config_t {
  void* codec_config;
  bt_byte* buffer;
  bt_int buffer_len;
  bt_int offset;
} bt_avdtp_codec_op_serialize_config_t;
```

Members

Members	Description
void* codec_config;	< A pointer to a structure defined by the AVDTP consumer where codec's configuration < will be read from by the handler. The format of the structure is totally up to < the AVDTP consumer. The dotstack defines such structures for SBC, MPEG-1,2 and MPEG-2,4 AAC: < @arg SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h) < @arg MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h) < @arg MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h)
bt_byte* buffer;	< A pointer to a buffer where the handler has to write codec's configuration in OTA format.
bt_int buffer_len;	< The length of the c buffer.
bt_int offset;	< The c buffer points to a complete packet that will be sent to the remote device. The c offset < points to a location in the c buffer where codec's configuration has to be written.

Description

brief Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation. ingroup avdtp

details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::serialize - when ADVDTP needs to serialize codec's capabilities/configuration for sending to the remote device.

bt_avdtp_codec_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_codec_t {
  struct _bt_avdtp_codec_t* next_codec;
  bt_byte codec_type;
  bt_avdtp_codec_handler_fp codec_handler;
} bt_avdtp_codec_t;
```

Members

Members	Description
bt_byte codec_type;	< Codec type. < The c codec_type can be one of the following values: < @arg AVDTP_CODEC_TYPE_SBC: SBC < @arg AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) < @arg AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) < @arg AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) < @arg AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec
bt_avdtp_codec_handler_fp codec_handler;	< A pointer to a codec handler.

Description

brief Codec handler description. ingroup avdtp

details This structure is used to register a codec handler for parsing/serializing codec capabilities and configuration. See description of the ::bt_avdtp_register_codec for more details.

bt_avdtp_control_channel_t Structure

File

avdtp_control.h

C

```
typedef struct _bt_avdtp_control_channel_t {
  bt_byte state;
  bt_bdaddr_t remote_addr;
  bt_l2cap_channel_t* l2cap_channel;
  bt_byte* tx_buffer;
  bt_queue_element_t* send_cq_head;
  bt_queue_element_t* ack_cq_head;
  bt_byte next_transaction_id;
  bt_byte next_ts_id;
  bt_byte next_tc_id;
  struct _bt_avdtp_mgr_t* avdtp_mgr;
  bt_byte remaining_connect_attemtps;
  bt_bdaddr_t connect_address;
  bt_uint connect_acl_config;
} bt_avdtp_control_channel_t;
```

Description

This is type bt_avdtp_control_channel_t.

bt_avdtp_control_cmd_t Structure

File

avdtp_control.h

C

```
typedef struct _bt_avdtp_control_cmd_t {
  struct _bt_avdtp_control_cmd_t* next_cmd;
  bt_byte code;
  bt_byte type;
  bt_byte transaction_id;
  bt_byte* params;
  bt_uint params_len;
} bt_avdtp_control_cmd_t;
```

Description

This is type bt_avdtp_control_cmd_t.

bt_avdtp_ctrl_evt_data_received_t Structure

File

avdtp_control.h

C

```
typedef struct _bt_avdtp_ctrl_evt_data_received_t {
  bt_byte* data;
  bt_int len;
} bt_avdtp_ctrl_evt_data_received_t;
```

Description

This is type bt_avdtp_ctrl_evt_data_received_t.

bt_avdtp_event_t Union

File

avdtp.h

C

```
typedef union _bt_avdtp_event_u {
  bt_avdtp_evt_ctrl_channel_connected_t ctrl_channel_connected;
  bt_avdtp_evt_ctrl_connection_failed_t ctrl_connection_failed;
  bt_avdtp_evt_ctrl_channel_disconnected_t ctrl_channel_disconnected;
  bt_avdtp_evt_discover_completed_t discover_completed;
  bt_avdtp_evt_sep_info_received_t sep_info_received;
  bt_avdtp_evt_get_sep_capabilities_completed_t get_sep_capabilities_completed;
  bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received;
  bt_avdtp_evt_set_stream_configuration_completed_t set_stream_configuration_completed;
  bt_avdtp_evt_get_stream_configuration_completed_t get_stream_configuration_completed;
  bt_avdtp_evt_stream_reconfigure_completed_t stream_reconfigure_completed;
  bt_avdtp_evt_open_stream_completed_t open_stream_completed;
  bt_avdtp_evt_start_stream_completed_t start_stream_completed;
  bt_avdtp_evt_close_stream_completed_t close_stream_completed;
  bt_avdtp_evt_suspend_stream_completed_t suspend_stream_completed;
  bt_avdtp_evt_stream_security_control_completed_t security_control_completed;
  bt_avdtp_evt_set_stream_configuration_requested_t set_stream_configuration_requested;
  bt_avdtp_evt_reconfigure_stream_requested_t reconfigure_stream_requested;
  bt_avdtp_evt_open_stream_requested_t open_stream_requested;
  bt_avdtp_evt_start_stream_requested_t start_stream_requested;
  bt_avdtp_evt_suspend_stream_requested_t suspend_stream_requested;
  bt_avdtp_evt_close_stream_requested_t close_stream_requested;
  bt_avdtp_evt_abort_stream_requested_t abort_stream_requested;
  bt_avdtp_evt_delay_report_completed_t delay_report_completed;
  bt_avdtp_evt_set_stream_configuration_t set_stream_configuration;
  bt_avdtp_evt_stream_configured_t stream_configured;
  bt_avdtp_evt_stream_reconfigured_t stream_reconfigured;
  bt_avdtp_evt_stream_opened_t stream_opened;
  bt_avdtp_evt_stream_started_t stream_started;
  bt_avdtp_evt_stream_suspended_t stream_suspended;
  bt_avdtp_evt_stream_closed_t stream_closed;
  bt_avdtp_evt_stream_aborted_t stream_aborted;
  bt_avdtp_evt_media_packet_received_t media_packet_received;
  bt_avdtp_evt_media_packet_sent_t media_packet_sent;
```

```
    bt_avdtp_evt_media_packet_send_failed_t media_packet_send_failed;
  } bt_avdtp_event_t;
```

Members

Members	Description
bt_avdtp_evt_ctrl_channel_connected_t ctrl_channel_connected;	< Valid if event is AVDTP_EVT_CTRL_CHANNEL_CONNECTED
bt_avdtp_evt_ctrl_connection_failed_t ctrl_connection_failed;	< Valid if event is AVDTP_EVT_CTRL_CONNECTION_FAILED
bt_avdtp_evt_ctrl_channel_disconnected_t ctrl_channel_disconnected;	< Valid if event is AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED
bt_avdtp_evt_discover_completed_t discover_completed;	< Valid if event is AVDTP_EVT_DISCOVER_COMPLETED
bt_avdtp_evt_sep_info_received_t sep_info_received;	< Valid if event is AVDTP_EVT_SEP_INFO_RECEIVED
bt_avdtp_evt_get_sep_capabilities_completed_t get_sep_capabilities_completed;	< Valid if event is AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED
bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received;	< Valid if event is AVDTP_EVT_SEP_CAPABILITIES_RECEIVED
bt_avdtp_evt_set_stream_configuration_completed_t set_stream_configuration_completed;	< Valid if event is AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED
bt_avdtp_evt_get_stream_configuration_completed_t get_stream_configuration_completed;	< Valid if event is AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED
bt_avdtp_evt_stream_reconfigure_completed_t stream_reconfigure_completed;	< Valid if event is AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED
bt_avdtp_evt_open_stream_completed_t open_stream_completed;	< Valid if event is AVDTP_EVT_OPEN_STREAM_COMPLETED
bt_avdtp_evt_start_stream_completed_t start_stream_completed;	< Valid if event is AVDTP_EVT_START_STREAM_COMPLETED
bt_avdtp_evt_close_stream_completed_t close_stream_completed;	< Valid if event is AVDTP_EVT_CLOSE_STREAM_COMPLETED
bt_avdtp_evt_suspend_stream_completed_t suspend_stream_completed;	< Valid if event is AVDTP_EVT_SUSPEND_STREAM_COMPLETED
bt_avdtp_evt_stream_security_control_completed_t security_control_completed;	< Valid if event is AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED
bt_avdtp_evt_set_stream_configuration_requested_t set_stream_configuration_requested;	< Valid if event is AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED
bt_avdtp_evt_reconfigure_stream_requested_t reconfigure_stream_requested;	< Valid if event is AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED
bt_avdtp_evt_open_stream_requested_t open_stream_requested;	< Valid if event is AVDTP_EVT_OPEN_STREAM_REQUESTED
bt_avdtp_evt_start_stream_requested_t start_stream_requested;	< Valid if event is AVDTP_EVT_START_STREAM_REQUESTED
bt_avdtp_evt_suspend_stream_requested_t suspend_stream_requested;	< Valid if event is AVDTP_EVT_SUSPEND_STREAM_REQUESTED
bt_avdtp_evt_close_stream_requested_t close_stream_requested;	< Valid if event is AVDTP_EVT_CLOSE_STREAM_REQUESTED
bt_avdtp_evt_abort_stream_requested_t abort_stream_requested;	< Valid if event is AVDTP_EVT_ABORT_STREAM_REQUESTED
bt_avdtp_evt_delay_report_completed_t delay_report_completed;	< Valid if event is AVDTP_EVT_DELAYREPORT_COMPLETED
bt_avdtp_evt_stream_configured_t stream_configured;	< Valid if event is AVDTP_EVT_STREAM_CONFIGURED
bt_avdtp_evt_stream_reconfigured_t stream_reconfigured;	< Valid if event is AVDTP_EVT_STREAM_RECONFIGURED
bt_avdtp_evt_stream_opened_t stream_opened;	< Valid if event is AVDTP_EVT_STREAM_OPENED
bt_avdtp_evt_stream_started_t stream_started;	< Valid if event is AVDTP_EVT_STREAM_STARTED
bt_avdtp_evt_stream_suspended_t stream_suspended;	< Valid if event is AVDTP_EVT_STREAM_SUSPENDED
bt_avdtp_evt_stream_closed_t stream_closed;	< Valid if event is AVDTP_EVT_STREAM_CLOSED
bt_avdtp_evt_stream_aborted_t stream_aborted;	< Valid if event is AVDTP_EVT_STREAM_ABORTED
bt_avdtp_evt_media_packet_received_t media_packet_received;	< Valid if event is AVDTP_EVT_MEDIA_PACKET_RECEIVED

bt_avdtp_evt_media_packet_sent_t media_packet_sent;	< Valid if event is AVDTP_EVT_MEDIA_PACKET_SENT
bt_avdtp_evt_media_packet_send_failed_t media_packet_send_failed;	< Valid if event is AVDTP_EVT_MEDIA_PACKET_SEND_FAILED

Description

brief Parameter to an application callback. ingroup avdtp

details This union is used to pass event specific data to the AVDTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

bt_avdtp_evt_abort_stream_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_abort_stream_requested_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_abort_stream_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte strm_handle;	< The handle of a stream to abort.

Description

brief Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::abort_stream_requested - when AVDTP received a "abort stream" request.

bt_avdtp_evt_close_stream_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_close_stream_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_close_stream_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_byte strm_handle;	< Stream handle.

Description

brief Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_completed - when AVDTP received a response to a "close stream" request.

bt_avdtp_evt_close_stream_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_close_stream_requested_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_close_stream_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte strm_handle;	< The handle of a stream to close.

Description

brief Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_requested - when AVDTP received a "close stream" request.

bt_avdtp_evt_ctrl_channel_connected_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_ctrl_channel_connected_s {
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_ctrl_channel_connected_t;
```

Members

Members	Description
bt_bdaddr_t* bdaddr;	< the address of a remote device

Description

brief Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_connected - when a control channel between two devices has been established.

bt_avdtp_evt_ctrl_channel_disconnected_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_ctrl_channel_disconnected_s {
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_ctrl_channel_disconnected_t;
```

Members

Members	Description
bt_bdaddr_t* bdaddr;	< the address of a remote device

Description

brief Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_disconnected - when a control channel between two devices has been terminated.

bt_avdtp_evt_discover_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_discover_completed_t {
  bt_byte err_code;
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_discover_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of discovering. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_bdaddr_t* bdaddr;	< the address of a remote device

Description

brief Parameter to AVDTP_EVT_DISCOVER_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::discover_completed - when AVDTP completed discovering SEPs available on a remote device.

bt_avdtp_evt_get_sep_capabilities_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_get_sep_capabilities_completed_t {
  bt_byte err_code;
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_get_sep_capabilities_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_bdaddr_t* bdaddr;	< the address of a remote device

Description

brief Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_sep_capabilities_completed - when AVDTP received a response to a "get SEP capabilities" request. AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_SEP_CAPABILITIES_RECEIVED - is generate with a pointer to a structure that holds actual SEP's capabilities.

bt_avdtp_evt_get_stream_configuration_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_get_stream_configuration_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_get_stream_configuration_completed_t;
```

Description

brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_stream_configuration_completed - when AVDTP received a response to a "get stream configuration" request. AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED - is generate with a pointer to a structure that hold actual stream's configuration.

bt_avdtp_evt_media_packet_received_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_media_packet_received_t {
  bt_byte strm_handle;
  bt_media_packet_t* packet;
} bt_avdtp_evt_media_packet_received_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream which received a packet.
bt_media_packet_t* packet;	< A pointer to a media packet buffer.

Description

brief Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_received - when ADVDTP received a media packet from the remote device.

bt_avdtp_evt_media_packet_send_failed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_media_packet_send_failed_t {
  bt_byte strm_handle;
  bt_media_packet_t* packet;
} bt_avdtp_evt_media_packet_send_failed_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream which received a packet.
bt_media_packet_t* packet;	< A pointer to a media packet buffer.

Description

brief Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_send_failed - when ADVDTP failed to send a media packet to the remote device.

bt_avdtp_evt_media_packet_sent_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_media_packet_sent_t {
  bt_byte strm_handle;
  bt_media_packet_t* packet;
} bt_avdtp_evt_media_packet_sent_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream which received a packet.
bt_media_packet_t* packet;	< A pointer to a media packet buffer.

Description

brief Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_sent - when ADVDTP sent a media packet to the remote device.

bt_avdtp_evt_open_stream_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_open_stream_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_open_stream_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_byte strm_handle;	< Stream handle.

Description

brief Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_completed - when AVDTP received a response to a "open stream" request.

bt_avdtp_evt_open_stream_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_open_stream_requested_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_open_stream_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte strm_handle;	< The handle of a stream to open.

Description

brief Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_requested - when AVDTP received a "open stream" request.

bt_avdtp_evt_reconfigure_stream_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_reconfigure_stream_requested_t {
  bt_byte err_code;
  bt_byte err_category;
  bt_avdtp_sep_t* sep;
  bt_avdtp_sep_capabilities_t* config;
  bt_byte strm_handle;
} bt_avdtp_evt_reconfigure_stream_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte err_category;	< If local device cannot accept the request it should set c err_category to the value of the forst Service Category that failed.
bt_avdtp_sep_t* sep;	< Description of the local SEP.
bt_avdtp_sep_capabilities_t* config;	< Stream configuration requested by the remote party.
bt_byte strm_handle;	< Stream handle.

Description

brief Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::reconfigure_stream_requested - when AVDTP received a "change stream configuration" request.

bt_avdtp_evt_sep_capabilities_received_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_sep_capabilities_received_t {
  bt_avdtp_sep_capabilities_t* caps;
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_sep_capabilities_received_t;
```

Members

Members	Description
bt_avdtp_sep_capabilities_t* caps;	< SEP capabilities or stream configuration.

Description

brief Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED events ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_capabilities_received - when AVDTP received a positive response to a "get SEP capabilities" or "get stream configuration" request.

bt_avdtp_evt_sep_info_received_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_sep_info_received_t {
  bt_byte sep_id;
```

```
  bt_byte sep_type;
  bt_bool in_use;
  bt_byte media_type;
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_sep_info_received_t;
```

Members

Members	Description
bt_byte sep_id;	< SEP ID.
bt_byte sep_type;	< SEP type. This can be either AVDTP_SEP_TYPE_SOURCE or AVDTP_SEP_TYPE_SINK.
bt_bool in_use;	< A flag indicating if a SEP is already being used.
bt_byte media_type;	< Type of media supported by this SEP. This can be on of the following values: < li AVDTP_MEDIA_TYPE_AUDIO < li AVDTP_MEDIA_TYPE_VIDEO < li AVDTP_MEDIA_TYPE_MULTIMEDIA

Description

brief Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_info_received - when AVDTP received positive result to a "discover" request. AVDTP_EVT_SEP_INFO_RECEIVED is generated for every SEP received from the remote device.

bt_avdtp_evt_set_stream_configuration_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_set_stream_configuration_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
  bt_byte svc_category;
} bt_avdtp_evt_set_stream_configuration_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_byte strm_handle;	< Stream handle.
bt_byte svc_category;	< The value of the first Service Category to fail if the remote rejected the request.

Description

brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_completed - when AVDTP received a response to a "set stream configuration" request.

bt_avdtp_evt_set_stream_configuration_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_set_stream_configuration_requested_t {
  bt_byte err_code;
  bt_byte err_category;
  bt_avdtp_sep_t* sep;
  bt_byte int_sep_id;
  bt_avdtp_sep_capabilities_t* config;
  bt_byte strm_handle;
} bt_avdtp_evt_set_stream_configuration_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte err_category;	< If local device cannot accept the request it should set c err_category to the value of the forst Service Category that failed.
bt_avdtp_sep_t* sep;	< Description of the local SEP.
bt_byte int_sep_id;	< The ID of the remote SEP.
bt_avdtp_sep_capabilities_t* config;	< Stream configuration requested by the remote party.
bt_byte strm_handle;	< Stream handle.

Description

brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_requested - when AVDTP received a "set stream configuration" request.

bt_avdtp_evt_start_stream_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_start_stream_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_start_stream_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_byte strm_handle;	< Stream handle.

Description

brief Parameter to AVDTP_EVT_START_STREAM_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_completed - when AVDTP received a response to a "start stream" request.

bt_avdtp_evt_start_stream_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_start_stream_requested_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_start_stream_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte strm_handle;	< The handle of a stream to start.

Description

brief Parameter to AVDTP_EVT_START_STREAM_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_requested - when AVDTP received a "start stream" request.

bt_avdtp_evt_stream_aborted_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_aborted_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_aborted_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been aborted.

Description

brief Parameter to AVDTP_EVT_STREAM_ABORTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully aborted.

bt_avdtp_evt_stream_closed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_closed_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_closed_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been closed.

Description

brief Parameter to AVDTP_EVT_STREAM_CLOSED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully closed.

bt_avdtp_evt_stream_configured_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_configured_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_configured_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been configured.

Description

brief Parameter to AVDTP_EVT_STREAM_CONFIGURED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_configured - to notify the AVDTP consumer that a stream configuration has been successfully completed.

bt_avdtp_evt_stream_opened_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_opened_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_opened_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been opened.

Description

brief Parameter to AVDTP_EVT_STREAM_OPENED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_opened - to notify the AVDTP consumer that a stream has been successfully opened.

bt_avdtp_evt_stream_reconfigure_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_reconfigure_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
  bt_byte svc_category;
} bt_avdtp_evt_stream_reconfigure_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_byte strm_handle;	< Stream handle.
bt_byte svc_category;	< The value of the first Service Category to fail if the remote rejected the request.

Description

brief Parameter to AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigure_completed - when AVDTP received a response to a "change stream configuration" request.

bt_avdtp_evt_stream_reconfigured_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_reconfigured_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_reconfigured_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been reconfigured.

Description

brief Parameter to AVDTP_EVT_STREAM_RECONFIGURED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigured - to notify the AVDTP consumer that a stream configuration has been successfully changed.

bt_avdtp_evt_stream_security_control_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_security_control_completed_t {
  bt_byte err_code;
} bt_avdtp_evt_stream_security_control_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.

Description

brief Parameter to AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::security_control_completed - when AVDTP received a response to a "exchange content protection control data" request.

bt_avdtp_evt_stream_started_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_started_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_started_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been started.

Description

brief Parameter to AVDTP_EVT_STREAM_STARTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_started - to notify the AVDTP consumer that a stream has been successfully started.

bt_avdtp_evt_stream_suspended_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_stream_suspended_t {
  bt_byte strm_handle;
} bt_avdtp_evt_stream_suspended_t;
```

Members

Members	Description
bt_byte strm_handle;	< The handle of a stream that has been suspended.

Description

brief Parameter to AVDTP_EVT_STREAM_SUSPENDED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_suspended - to notify the AVDTP consumer that a stream has been successfully suspended.

bt_avdtp_evt_suspend_stream_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_suspend_stream_completed_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_suspend_stream_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.
bt_byte strm_handle;	< Stream handle.

Description

brief Parameter to AVDTP_EVT_SUSPEND_STREAM_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_completed - when AVDTP received a response to a "suspend stream" request.

bt_avdtp_evt_suspend_stream_requested_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_suspend_stream_requested_t {
  bt_byte err_code;
  bt_byte strm_handle;
} bt_avdtp_evt_suspend_stream_requested_t;
```

Members

Members	Description
bt_byte err_code;	< The result to be sent to the remote party. < li If local device accepts the configuration requested by the remote device < it should set c err_code to AVDTP_ERROR_SUCCESS. Otherwise it should set c err_code < to one of the AVDTP_ERROR_ constants.
bt_byte strm_handle;	< The handle of a stream to suspend.

Description

brief Parameter to AVDTP_EVT_SUSPEND_STREAM_REQUESTED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_requested - when AVDTP received a "suspend stream" request.

bt_avdtp_mgr_callback_fp Type

File

avdtp.h

C

```
typedef void (* bt_avdtp_mgr_callback_fp)(struct _bt_avdtp_mgr_t* mgr, bt_byte evt, bt_avdtp_event_t*
evt_param, void* callback_param);
```

Description

brief AVDTP application callback. ingroup avdtp

details In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call that function whenever a new event has been generated.

param mgr AVDTP manager.

param evt AVDTP event. The event can be one of the following values: @arg AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local device received a response to Get SEP capabilities operation. @arg AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Set stream configuration operation. @arg AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Get stream configuration operation. @arg AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED: Local device received a response to Reconfigure stream operation. @arg AVDTP_EVT_OPEN_STREAM_COMPLETED: Local device received a response to Open stream operation. @arg AVDTP_EVT_START_STREAM_COMPLETED: Local device received a response to Start stream operation. @arg AVDTP_EVT_CLOSE_STREAM_COMPLETED: Local device received a response to Close stream operation. @arg AVDTP_EVT_SUSPEND_STREAM_COMPLETED: Local device received a response to Suspend stream operation. @arg AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: Local device received a response to Stream security control operation. @arg AVDTP_EVT_ABORT_STREAM_COMPLETED: Local device received a response to Abort stream operation. @arg AVDTP_EVT_SEP_INFO_RECEIVED: SEP information received. @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: SEP capabilities received. @arg AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: Stream configuration received.

@arg AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: Remote device requested stream configuration. @arg AVDTP_EVT_OPEN_STREAM_REQUESTED: Remote device requested to open a stream. @arg AVDTP_EVT_START_STREAM_REQUESTED: Remote device requested to start a stream. @arg AVDTP_EVT_CLOSE_STREAM_REQUESTED: Remote device requested to close a stream. @arg AVDTP_EVT_SUSPEND_STREAM_REQUESTED: Remote device requested to suspend a stream. @arg AVDTP_EVT_ABORT_STREAM_REQUESTED: Remote device requested to abort a stream. @arg AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED: Remote device requested to reconfigure a stream. @arg AVDTP_EVT_MEDIA_PACKET_RECEIVED: Remote device sent a media packet. @arg AVDTP_EVT_STREAM_CONFIGURED: A stream has been configured (This event is generated right after AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_RECONFIGURED: A stream has been re-configured (This event is generated right after AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_OPENED: A stream has been opened (This event is generated as a result of local or remote stream opening request). @arg AVDTP_EVT_STREAM_STARTED: A stream has been started (This event is generated right after AVDTP_EVT_START_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_CLOSED: A stream has been close (This event is generated right after AVDTP_EVT_CLOSE_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_SUSPENDED: A stream has been suspended (This event is generated right after AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request). @arg AVDTP_EVT_STREAM_ABORTED: A stream has been aborted (This event is generated right after AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request. It is also generated if connection between devices has been terminated by means other than AVDTP signaling, e.g. devices going out of rage). @arg AVDTP_EVT_MEDIA_PACKET_SENT: The local device has successfully sent a media packet to the remote device. @arg AVDTP_EVT_MEDIA_PACKET_SEND_FAILED: The local device was not able to send a media packet to the remote device.

param evt_param Event parameter. Which member of the bt_avdtp_event_t union is valid depends on the event: @arg AVDTP_EVT_CTRL_CHANNEL_CONNECTED: c bt_avdtp_evt_ctrl_channel_connected_t ctrl_channel_connected @arg AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: c bt_avdtp_evt_ctrl_channel_disconnected_t ctrl_channel_disconnected @arg AVDTP_EVT_CTRL_CONNECTION_FAILED: c NULL @arg AVDTP_EVT_DISCOVER_COMPLETED: c bt_avdtp_evt_discover_completed_t discover_completed; @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: c bt_avdtp_evt_get_sep_capabilities_completed_t get_sep_capabilities_completed @arg AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: c bt_avdtp_evt_set_stream_configuration_completed_t set_stream_configuration_completed @arg AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: c bt_avdtp_evt_get_stream_configuration_completed_t get_stream_configuration_completed @arg AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED: c bt_avdtp_evt_stream_reconfigure_completed_t stream_reconfigure_completed @arg AVDTP_EVT_OPEN_STREAM_COMPLETED: c bt_avdtp_evt_open_stream_completed_t open_stream_completed @arg AVDTP_EVT_START_STREAM_COMPLETED: c bt_avdtp_evt_start_stream_completed_t start_stream_completed @arg AVDTP_EVT_CLOSE_STREAM_COMPLETED: c bt_avdtp_evt_close_stream_completed_t close_stream_completed @arg AVDTP_EVT_SUSPEND_STREAM_COMPLETED: c bt_avdtp_evt_suspend_stream_completed_t suspend_stream_completed @arg AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: c bt_avdtp_evt_stream_security_control_completed_t security_control_completed @arg AVDTP_EVT_ABORT_STREAM_COMPLETED: c bt_avdtp_evt_abort_stream_requested_t abort_stream_requested @arg AVDTP_EVT_SEP_INFO_RECEIVED: c bt_avdtp_evt_sep_info_received_t sep_info_received @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: c bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received @arg AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: c bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received

@arg AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: c bt_avdtp_evt_set_stream_configuration_requested_t set_stream_configuration_requested @arg AVDTP_EVT_OPEN_STREAM_REQUESTED: c bt_avdtp_evt_open_stream_requested_t open_stream_requested @arg AVDTP_EVT_START_STREAM_REQUESTED: c bt_avdtp_evt_start_stream_requested_t start_stream_requested @arg AVDTP_EVT_CLOSE_STREAM_REQUESTED: c bt_avdtp_evt_close_stream_requested_t close_stream_requested @arg AVDTP_EVT_SUSPEND_STREAM_REQUESTED: c bt_avdtp_evt_suspend_stream_requested_t suspend_stream_requested @arg

AVDTP_EVT_ABORT_STREAM_REQUESTED: c bt_avdtp_evt_abort_stream_requested_t abort_stream_requested @arg AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED: c bt_avdtp_evt_reconfigure_stream_requested_t reconfigure_stream_requested @arg AVDTP_EVT_MEDIA_PACKET_RECEIVED: c bt_avdtp_evt_media_packet_received_t media_packet_received @arg AVDTP_EVT_STREAM_CONFIGURED: c bt_avdtp_evt_stream_configured_t stream_configured @arg AVDTP_EVT_STREAM_RECONFIGURED: c bt_avdtp_evt_stream_reconfigured_t stream_reconfigured @arg AVDTP_EVT_STREAM_OPENED: c bt_avdtp_evt_stream_opened_t stream_opened @arg AVDTP_EVT_STREAM_STARTED: c bt_avdtp_evt_stream_started_t stream_started @arg AVDTP_EVT_STREAM_CLOSED: c bt_avdtp_evt_stream_closed_t stream_closed @arg AVDTP_EVT_STREAM_SUSPENDED: c bt_avdtp_evt_stream_suspended_t stream_suspended @arg AVDTP_EVT_STREAM_ABORTED: c bt_avdtp_evt_stream_aborted_t stream_aborted @arg AVDTP_EVT_MEDIA_PACKET_SENT: c bt_avdtp_evt_media_packet_sent_t media_packet_sent @arg AVDTP_EVT_MEDIA_PACKET_SEND_FAILED: c bt_avdtp_evt_media_packet_send_failed_t media_packet_send_failed

param callback_param A pointer to an arbitrary data set by a call to bt_avdtp_register_callback.

bt_avdtp_mgr_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_mgr_t {
  bt_byte state;
  bt_byte flags;
  bt_avdtp_sep_t* seps;
  bt_byte next_sep_id;
  bt_avdtp_stream_t* streams;
  bt_byte next_stream_handle;
  bt_avdtp_control_channel_t* control_channels;
  bt_avdtp_transport_channel_t* transport_channels;
  bt_avdtp_codec_t* codecs;
  bt_avdtp_mgr_callback_fp callback;
  void* callback_param;
  bt_avdtp_control_cmd_t* pending_command;
  bt_avdtp_stream_t* opening_strm;
  bt_avdtp_stream_t* sending_strm;
} bt_avdtp_mgr_t;
```

Members

Members	Description
bt_byte state;	< Manager state. This value can be one of the following values: < li AVDTP_MANAGER_STATE_IDLE < li AVDTP_MANAGER_STATE_CONNECTING
bt_byte flags;	< Additional manager state. This value can be a combination of the following values: < li AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET
bt_avdtp_sep_t* seps;	< Pointer to a buffer of SEPs available for allocating with c bt_avdtp_register_sep.
bt_byte next_sep_id;	< Holds ID of the next SEP to be allocated. Every time c bt_avdtp_register_sep is called this value is increased by 1.
bt_avdtp_stream_t* streams;	< Pointer to a buffer of streams available for allocating with c bt_avdtp_create_stream.
bt_byte next_stream_handle;	< Holds ID of the next stream to be allocated. Every time c bt_avdtp_create_stream is called this value is increased by 1.
bt_avdtp_control_channel_t* control_channels;	< Pointer to a buffer of available control channles.
bt_avdtp_transport_channel_t* transport_channels;	< Pointer to a buffer of available transport channles.
bt_avdtp_codec_t* codecs;	< A list of supported codecs.
bt_avdtp_mgr_callback_fp callback;	< Pointer to a function which a AVDTP consumer must register in order to be notified of various events.
void* callback_param;	< Pointer to arbitrary data to be passed to the c callback.
bt_avdtp_control_cmd_t* pending_command;	< If local device wants to send a command to a remote device but control channel does not exists this member holds a pointer to the command until the channel is created.
bt_avdtp_stream_t* opening_strm;	< Holds a pointer to a stream being opened by a remote device.
bt_avdtp_stream_t* sending_strm;	< Holds a pointer to a stream sending media packet.

Description

brief AVDTP manager. ingroup avdtp

details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avdtp_get_mgr().

bt_avdtp_sep_capabilities_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_sep_capabilities_t {
  bt_byte categories;
  bt_byte media_type;
  bt_byte codec_type;
  void* codec_config;
  bt_uint cp_type;
  bt_byte* cp_specific;
  bt_byte cp_specific_len;
  bt_byte recovery_type;
  bt_byte max_recovery_window;
  bt_byte max_parity_code_packets;
  bt_byte header_compression_options;
  bt_byte multiplexing_options;
  bt_byte tsid_media;
  bt_byte tcid_media;
  bt_byte tsid_reporting;
  bt_byte tcid_reporting;
  bt_byte tsid_recovery;
  bt_byte tcid_recovery;
} bt_avdtp_sep_capabilities_t;
```

Members

Members	Description
bt_byte categories;	< Defines service capabilities exposed by a SEP to a remote party. This can be a combination of the following values: < li AVDTP_SEP_CAPABILITY_FLAG_MEDIA_TRANSPORT < li AVDTP_SEP_CAPABILITY_FLAG_REPORTING < li AVDTP_SEP_CAPABILITY_FLAG_RECOVERY < li AVDTP_SEP_CAPABILITY_FLAG_CONTENT_PROTECTION < li AVDTP_SEP_CAPABILITY_FLAG_HEADER_COMPRESSION < li AVDTP_SEP_CAPABILITY_FLAG_MULTIPLEXING < li AVDTP_SEP_CAPABILITY_FLAG_MEDIA_CODEC
bt_byte media_type;	< Type of media supported by this SEP. This can be on of the following values: < li AVDTP_MEDIA_TYPE_AUDIO < li AVDTP_MEDIA_TYPE_VIDEO < li AVDTP_MEDIA_TYPE_MULTIMEDIA
bt_byte codec_type;	< Codec type supported by this SEP. This can be on of the following values: < li AVDTP_CODEC_TYPE_SBC < li AVDTP_CODEC_TYPE_MPEG1_2_AUDIO < li AVDTP_CODEC_TYPE_MPEG2_4_AAC < li AVDTP_CODEC_TYPE_ATRAC < li AVDTP_CODEC_TYPE_NON_A2DP
void* codec_config;	< Pointer to a buffer that holds codec specific capabilities
bt_uint cp_type;	< Type of content protection supported by this SEP. < note Content protection is currently not supported by dotstack.
bt_byte* cp_specific;	< Pointer to a buffer holding content protection specific data. < note Content protection is currently not supported by dotstack.
bt_byte cp_specific_len;	< Length of the content protection specific data. < note Content protection is currently not supported by dotstack.
bt_byte recovery_type;	< Type of recovery supported by this SEP. < note Recovery is currently not supported by dotstack.
bt_byte max_recovery_window;	< Recovery window size. < note Recovery is currently not supported by dotstack.
bt_byte max_parity_code_packets;	< Maximum number of parity codec packets. < note Recovery is currently not supported by dotstack.
bt_byte header_compression_options;	< Header compression configuration < note Header compression is currently not supported by dotstack.
bt_byte multiplexing_options;	< Multiplexing configuration < note Multiplexing is currently not supported by dotstack.
bt_byte tsid_media;	< ID of the media transport session
bt_byte tcid_media;	< ID of the media transport channel
bt_byte tsid_reporting;	< ID of the reporting transport session
bt_byte tcid_reporting;	< ID of the reporting transport channel
bt_byte tsid_recovery;	< ID of the recovery transport session

bt_byte tcid_recovery;	< ID of the recovery transport channel

Description

brief SEP capabilities ingroup avdtp

details This structure is used to hold SEP capabilities.

bt_avdtp_sep_t Type

File

avdtp.h

C

```
typedef struct _bt_avdtp_sep_t bt_avdtp_sep_t;
```

Description

This is type bt_avdtp_sep_t.

bt_avdtp_stream_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_stream_t {
  struct _bt_avdtp_stream_t* next_stream;
  bt_byte handle;
  bt_byte state;
  bt_byte flags;
  struct _bt_avdtp_sep_t* sep;
  bt_avdtp_sep_capabilities_t* config;
  bt_byte remote_seid;
  bt_bdaddr_t remote_addr;
  bt_avdtp_transport_session_t sessions[AVDTP_MAX_STREAM_TRANSPORT_SESSION];
  struct _bt_avdtp_mgr_t* mgr;
  bt_byte cur_channel_index;
  bt_byte* listen_sep_list;
  bt_byte listen_sep_count;
  bt_queue_element_t* media_rx_queue;
  bt_queue_element_t* media_tx_queue;
  bt_uint media_tx_queue_limit;
} bt_avdtp_stream_t;
```

Members

Members	Description
struct _bt_avdtp_stream_t* next_stream;	< Pointer to next stream.
bt_byte handle;	< Stream handle. This values is allocated by dotstack and is used to manipulate the stream by the AVDTP consumer.
bt_byte state;	< State of the stream. This value can be one of the following values: < li AVDTP_STREAM_STATE_IDLE < li AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS < li AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS < li AVDTP_STREAM_STATE_CONFIGURED < li AVDTP_STREAM_STATE_OPEN < li AVDTP_STREAM_STATE_STREAMING < li AVDTP_STREAM_STATE_CLOSING < li AVDTP_STREAM_STATE_ABORTING
bt_byte flags;	< Additional stream state. This value can be one of the following values: < li AVDTP_STREAM_FLAG_LISTENING
struct _bt_avdtp_sep_t* sep;	< Local SEP this stream is connected to.
bt_avdtp_sep_capabilities_t* config;	< Current SEP configuration.
bt_byte remote_seid;	< ID of the remote SEP.
bt_bdaddr_t remote_addr;	< BT address of the remote device.
bt_avdtp_transport_session_t sessions[AVDTP_MAX_STREAM_TRANSPORT_SESSION];	< List of transport session available/active on this stream. < note There can be up to 3 (AVDTP_MAX_STREAM_TRANSPORT_SESSION == 3) transport sessions on a stream - media, reporting and recovery. < dotstack supports only media sessions so the other two are never used.

struct _bt_avdtp_mgr_t* mgr;	< AVDTP manager this stream belongs to.
bt_byte cur_channel_index;	< This value is currenlty not used.
bt_byte* listen_sep_list;	< A list of SEPs this channel this channel is listening on (i.e. can accept incoming connection requests).
bt_byte listen_sep_count;	< The number of SEPs in c listen_sep_list.
bt_queue_element_t* media_rx_queue;	< A list of media packet buffer for receiving incoming packets.
bt_queue_element_t* media_tx_queue;	< A list of media packet buffer to be sent to a remote device.
bt_uint media_tx_queue_limit;	< Maximum number of packets that TX queue can hold. 0 - unlimited.

Description

brief Stream description ingroup avdtp

details This structure is used to hold information about streams available on a local device.

bt_avdtp_transport_channel_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_transport_channel_t {
  struct _bt_avdtp_transport_channel_t* next_channel;
  bt_byte id;
  bt_byte type;
  bt_l2cap_channel_t* l2cap_channel;
  bt_byte ref_count;
  bt_byte connect_ref_count;
} bt_avdtp_transport_channel_t;
```

Members

Members	Description
struct _bt_avdtp_transport_channel_t* next_channel;	< Pointer to next channel.
bt_byte id;	< ID of the channel.
bt_byte type;	< Type of the channel. This can be one of the following values: < li AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED < li AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED < note Shared channles (i.e. multiplexing) is currently not supported by dotstack
bt_l2cap_channel_t* l2cap_channel;	< L2CAp channel used to transfer this AVDTP channel's data.
bt_byte ref_count;	< Channel's reference count. This value is intended for use with shared channels. < When ref count reaches 0 the channel is closed. < note Shared channles (i.e. multiplexing) is currently not supported by dotstack
bt_byte connect_ref_count;	< This value is currently not used.

Description

brief Transport channel description ingroup avdtp

details This structure is used to hold information about transport channels available on a local device.

bt_avdtp_transport_op_callback_fp Type

File

avdtp_private.h

C

```
typedef void (* bt_avdtp_transport_op_callback_fp)(bt_byte status, bt_byte opcode, void* param);
```

Description

This is type bt_avdtp_transport_op_callback_fp.

bt_avdtp_transport_session_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_transport_session_t {
  bt_byte id;
  bt_byte type;
  bt_avdtp_transport_channel_t* transport_channel;
} bt_avdtp_transport_session_t;
```

Members

Members	Description
bt_byte id;	< ID of the transport session.
bt_byte type;	< Type of the transport session. This can be one of the following values: < li AVDTP_TRANSPORT_SESSION_TYPE_MEDIA < li AVDTP_TRANSPORT_SESSION_TYPE_REPORTING < li AVDTP_TRANSPORT_SESSION_TYPE_RECOVERY
bt_avdtp_transport_channel_t* transport_channel;	< Transport channel used for this transport session.

Description

brief Transport session description ingroup avdtp

details This structure is used to hold information about transport sessions available on a local device.

bt_media_packet_t Type

File

avdtp.h

C

```
typedef struct _bt_media_packet_t bt_media_packet_t;
```

Description

This is type bt_media_packet_t.

__AVDTP_CONFIG_H Macro

File

avdtp_config.h

C

```
#define __AVDTP_CONFIG_H
```

Description

This is macro __AVDTP_CONFIG_H.

__AVDTP_CONTROL_H Macro

File

avdtp_control.h

C

```
#define __AVDTP_CONTROL_H
```

Description

This is macro __AVDTP_CONTROL_H.

__AVDTP_H Macro

File

avdtp.h

C

```
#define __AVDTP_H
```

Description

This is macro __AVDTP_H.

__AVDTP_PRIVATE_H Macro

File

avdtp_private.h

C

```
#define __AVDTP_PRIVATE_H
```

Description

This is macro __AVDTP_PRIVATE_H.

AVDTP_CODEC_OPCODE_PARSE_CONFIG Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_OPCODE_PARSE_CONFIG 0          ///< Parse codec configuration.
```

Description

< Parse codec configuration.

AVDTP_CODEC_OPCODE_PARSE_PACKET Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_OPCODE_PARSE_PACKET 2
```

Description

This is macro AVDTP_CODEC_OPCODE_PARSE_PACKET.

AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG 1          ///< Serialize codec configuration.
```

Description

< Serialize codec configuration.

AVDTP_CODEC_TYPE_ATRAC Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_TYPE_ATRAC 0x04 ///< ATRAC (proprietary codec owned by Sony Corporation).
```

Description

< ATRAC (proprietary codec owned by Sony Corporation).

AVDTP_CODEC_TYPE_MPEG1_2_AUDIO Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_TYPE_MPEG1_2_AUDIO 0x01 ///< MPEG-1,2 (optional).
```

Description

< MPEG-1,2 (optional).

AVDTP_CODEC_TYPE_MPEG2_4_AAC Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_TYPE_MPEG2_4_AAC 0x02 ///< MPEG-2,4 AAC (optional, used in Apple's products).
```

Description

< MPEG-2,4 AAC (optional, used in Apple's products).

AVDTP_CODEC_TYPE_NON_A2DP Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_TYPE_NON_A2DP 0xFF ///< Vendor specific.
```

Description

< Vendor specific.

AVDTP_CODEC_TYPE_SBC Macro

File

avdtp.h

C

```
#define AVDTP_CODEC_TYPE_SBC 0x00 ///< SBC (mandatory to support in A2DP profile).
```

Description

< SBC (mandatory to support in A2DP profile).

AVDTP_CTLR_CHANNEL_EVT_CONNECTED Macro

File

avdtp_control.h

C

```
#define AVDTP_CTLR_CHANNEL_EVT_CONNECTED 1
```

Description

This is macro AVDTP_CTLR_CHANNEL_EVT_CONNECTED.

AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED Macro

File

avdtp_control.h

C

```
#define AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED 3
```

Description

This is macro AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED.

AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED Macro

File

avdtp_control.h

C

```
#define AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED 2
```

Description

This is macro AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED.

AVDTP_CTLR_CHANNEL_EVT_NOTHING Macro

File

avdtp_control.h

C

```
#define AVDTP_CTLR_CHANNEL_EVT_NOTHING 0
```

Description

This is macro AVDTP_CTLR_CHANNEL_EVT_NOTHING.

AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED 4
```

Description

This is macro AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED.

AVDTP_CTRL_CHANNEL_STATE_CONNECTED Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_CHANNEL_STATE_CONNECTED 1
```

Description

This is macro AVDTP_CTRL_CHANNEL_STATE_CONNECTED.

AVDTP_CTRL_CHANNEL_STATE_CONNECTING Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_CHANNEL_STATE_CONNECTING 2
```

Description

This is macro AVDTP_CTRL_CHANNEL_STATE_CONNECTING.

AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED 0
```

Description

This is macro AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED.

AVDTP_CTRL_MESSAGE_TYPE_ACCEPT Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_MESSAGE_TYPE_ACCEPT 2
```

Description

This is macro AVDTP_CTRL_MESSAGE_TYPE_ACCEPT.

AVDTP_CTRL_MESSAGE_TYPE_COMMAND Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_MESSAGE_TYPE_COMMAND 0
```

Description

This is macro AVDTP_CTRL_MESSAGE_TYPE_COMMAND.

AVDTP_CTRL_MESSAGE_TYPE_FLD Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_MESSAGE_TYPE_FLD 3
```

Description

This is macro AVDTP_CTRL_MESSAGE_TYPE_FLD.

AVDTP_CTRL_MESSAGE_TYPE_REJECT Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_MESSAGE_TYPE_REJECT 3
```

Description

This is macro AVDTP_CTRL_MESSAGE_TYPE_REJECT.

AVDTP_CTRL_PACKET_TYPE_CONTINUE Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_PACKET_TYPE_CONTINUE 0x08
```

Description

This is macro AVDTP_CTRL_PACKET_TYPE_CONTINUE.

AVDTP_CTRL_PACKET_TYPE_END Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_PACKET_TYPE_END 0x0c
```

Description

This is macro AVDTP_CTRL_PACKET_TYPE_END.

AVDTP_CTRL_PACKET_TYPE_FLD Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_PACKET_TYPE_FLD 0x0c
```

Description

This is macro AVDTP_CTRL_PACKET_TYPE_FLD.

AVDTP_CTRL_PACKET_TYPE_SIGNLE Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_PACKET_TYPE_SIGNLE 0
```

Description

This is macro AVDTP_CTRL_PACKET_TYPE_SIGNLE.

AVDTP_CTRL_PACKET_TYPE_START Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_PACKET_TYPE_START 0x04
```

Description

This is macro AVDTP_CTRL_PACKET_TYPE_START.

AVDTP_ERROR_BAD_ACP_SEID Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_ACP_SEID 0x12    ///< The requested command indicates an invalid ACP SEP ID (not addressable)
```

Description

< The requested command indicates an invalid ACP SEP ID (not addressable)

AVDTP_ERROR_BAD_CP_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_CP_FORMAT 0x27    ///< The format of Content Protection Service Capability is not correct.
```

Description

< The format of Content Protection Service Capability is not correct.

AVDTP_ERROR_BAD_HEADER_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_HEADER_FORMAT 0x01    ///< The request packet header format is invalid.
```

Description

< The request packet header format is invalid.

AVDTP_ERROR_BAD_LENGTH Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_LENGTH 0x11    ///< The request packet length is not match the assumed length.
```

Description

< The request packet length is not match the assumed length.

AVDTP_ERROR_BAD_MEDIA_TRANSPORT_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_MEDIA_TRANSPORT_FORMAT 0x23    ///< The format of Media Transport Capability is not
correct.
```

Description

< The format of Media Transport Capability is not correct.

AVDTP_ERROR_BAD_MULTIPLEXING_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_MULTIPLEXING_FORMAT 0x28    ///< The format of Multiplexing Service Capability is
not correct.
```

Description

< The format of Multiplexing Service Capability is not correct.

AVDTP_ERROR_BAD_PAYLOAD_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_PAYLOAD_FORMAT 0x18    ///< The requested command has an incorrect payload format.
```

Description

< The requested command has an incorrect payload format.

AVDTP_ERROR_BAD_RECOVERY_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_RECOVERY_FORMAT 0x25    ///< The format of Recovery Service Capability is not
correct.
```

Description

< The format of Recovery Service Capability is not correct.

AVDTP_ERROR_BAD_RECOVERY_TYPE Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_RECOVERY_TYPE 0x22    ///< The requested Recovery Type is not defined in AVDTP.
```

Description

< The requested Recovery Type is not defined in AVDTP.

AVDTP_ERROR_BAD_ROHC_FORMAT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_ROHC_FORMAT 0x26    ///< The format of Header Compression Service Capability is not
correct.
```

Description

< The format of Header Compression Service Capability is not correct.

AVDTP_ERROR_BAD_SERV_CATEGORY Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_SERV_CATEGORY 0x17    ///< The value of Service Category in the request packet is
not defined in AVDTP.
```

Description

< The value of Service Category in the request packet is not defined in AVDTP.

AVDTP_ERROR_BAD_STATE Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_BAD_STATE 0x31    ///< The stream is in state that does not permit executing commands.
```

Description

< The stream is in state that does not permit executing commands.

AVDTP_ERROR_FAILED_TO_CONNECT_CONTROL Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_FAILED_TO_CONNECT_CONTROL 0x41    ///< An attempt to establish a control channel has
failed.
```

Description

< An attempt to establish a control channel has failed.

AVDTP_ERROR_FAILED_TO_CONNECT_TRANSPORT Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_FAILED_TO_CONNECT_TRANSPORT 0x40    ///< An attempt to establish a transport channel
has failed.
```

Description

< An attempt to establish a transport channel has failed.

AVDTP_ERROR_INVALID_CAPABILITIES Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_INVALID_CAPABILITIES 0x1a    ///< The reconfigure command is an attempt to reconfigure
a transport service capabilities of the SEP. Reconfigure is only permitted for application service
capabilities.
```

Description

< The reconfigure command is an attempt to reconfigure a transport service capabilities of the SEP. Reconfigure is only permitted for application service capabilities.

AVDTP_ERROR_NOT_SUPPORTED_COMMAND Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_NOT_SUPPORTED_COMMAND 0x19    ///< The requested command is not supported by the device.
```

Description

< The requested command is not supported by the device.

AVDTP_ERROR_SEP_IN_USE Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_SEP_IN_USE 0x13    ///< The SEP is in use.
```

Description

< The SEP is in use.

AVDTP_ERROR_SEP_NOT_IN_USE Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_SEP_NOT_IN_USE 0x14    ///< The SEP is not in use.
```

Description

< The SEP is not in use.

AVDTP_ERROR_SUCCESS Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_SUCCESS 0       ///< The operation completed with no errors.
```

Description

< The operation completed with no errors.

AVDTP_ERROR_UNSUPPORTED_CONFIGURAION Macro

File

avdtp.h

C

```
#define AVDTP_ERROR_UNSUPPORTED_CONFIGURAION 0x29    ///< Configuration not supported.
```

Description

< Configuration not supported.

AVDTP_EVT_ABORT_STREAM_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_ABORT_STREAM_COMPLETED 14    ///< This event is generated when a local device received a
response (either positive or negative) to a "abort stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.

AVDTP_EVT_ABORT_STREAM_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_ABORT_STREAM_REQUESTED 55    ///< This event is generated when a local device received
"abort stream" request.
```

Description

< This event is generated when a local device received "abort stream" request.

AVDTP_EVT_CLOSE_STREAM_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_CLOSE_STREAM_COMPLETED 11    ///< This event is generated when a local device received a
response (either positive or negative) to a "close stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "close stream" request.

AVDTP_EVT_CLOSE_STREAM_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_CLOSE_STREAM_REQUESTED 53    ///< This event is generated when a local device received "close stream" request.
```

Description

< This event is generated when a local device received "close stream" request.

AVDTP_EVT_CTRL_CHANNEL_CONNECTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_CTRL_CHANNEL_CONNECTED 1     ///< This event is generated when a control channel between two AVDTP entities has been established.
```

Description

< This event is generated when a control channel between two AVDTP entities has been established.

AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED 2     ///< This event is generated when a control channel between two AVDTP entities has been terminated.
```

Description

< This event is generated when a control channel between two AVDTP entities has been terminated.

AVDTP_EVT_CTRL_CONNECTION_FAILED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_CTRL_CONNECTION_FAILED 3     ///< This event is generated when a local device failed to create a control channel between two AVDTP entities.
```

Description

< This event is generated when a local device failed to create a control channel between two AVDTP entities.

AVDTP_EVT_DISCOVER_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_DISCOVER_COMPLETED 4     ///< This event is generated when a local device received a response (either positive or negative) to a "discover" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "discover" request.

AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED 5     ///< This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.

AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED 7     ///< This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.

AVDTP_EVT_LAST Macro

File

avdtp.h

C

```
#define AVDTP_EVT_LAST 99
```

Description

This is macro AVDTP_EVT_LAST.

AVDTP_EVT_MEDIA_PACKET_RECEIVED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_MEDIA_PACKET_RECEIVED 57    ///< This event is generated when a local device received a media packet.
```

Description

< This event is generated when a local device received a media packet.

AVDTP_EVT_MEDIA_PACKET_SEND_FAILED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_MEDIA_PACKET_SEND_FAILED 66    ///< This event is generated when a local device failed to send a media packet.
```

Description

< This event is generated when a local device failed to send a media packet.

AVDTP_EVT_MEDIA_PACKET_SENT Macro

File

avdtp.h

C

```
#define AVDTP_EVT_MEDIA_PACKET_SENT 65    ///< This event is generated when a local device sent a media packet.
```

Description

< This event is generated when a local device sent a media packet.

AVDTP_EVT_NULL Macro

File

avdtp.h

C

```
#define AVDTP_EVT_NULL 0
```

Description

This is macro AVDTP_EVT_NULL.

AVDTP_EVT_OPEN_STREAM_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_OPEN_STREAM_COMPLETED 9     ///< This event is generated when a local device received a response (either positive or negative) to a "open stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "open stream" request.

AVDTP_EVT_OPEN_STREAM_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_OPEN_STREAM_REQUESTED 51    ///< This event is generated when a local device received "open stream" request.
```

Description

< This event is generated when a local device received "open stream" request.

AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED 56    ///< This event is generated when a local device received "change stream configuration" request.
```

Description

< This event is generated when a local device received "change stream configuration" request.

AVDTP_EVT_SEP_CAPABILITIES_RECEIVED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SEP_CAPABILITIES_RECEIVED 16    ///< This event is generated when a local device received a positive response to a "get SEP capabilities" request.
```

Description

< This event is generated when a local device received a positive response to a "get SEP capabilities" request.

AVDTP_EVT_SEP_INFO_RECEIVED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SEP_INFO_RECEIVED 15    ///< This event is generated for each SEP contained in a positive response to a "discover" request.
```

Description

< This event is generated for each SEP contained in a positive response to a "discover" request.

AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED 6     ///< This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.

AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED 50    ///< This event is generated when a local device received "set stream configuration" request.
```

Description

< This event is generated when a local device received "set stream configuration" request.

AVDTP_EVT_START_STREAM_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_START_STREAM_COMPLETED 10    ///< This event is generated when a local device received a response (either positive or negative) to a "start stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "start stream" request.

AVDTP_EVT_START_STREAM_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_START_STREAM_REQUESTED 52    ///< This event is generated when a local device received "start stream" request.
```

Description

< This event is generated when a local device received "start stream" request.

AVDTP_EVT_STREAM_ABORTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_ABORTED 64    ///< This event is generated when a local device has successfully aborted a stream.
```

Description

< This event is generated when a local device has successfully aborted a stream. < This event follows the AVDTP_EVT_ABORT_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream abortion was initiated by the local device.

AVDTP_EVT_STREAM_CLOSED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_CLOSED 62    ///< This event is generated when a local device has successfully closed a stream.
```

Description

< This event is generated when a local device has successfully closed a stream. < This event follows the AVDTP_EVT_CLOSE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream closing was initiated by the local device.

AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED 17    ///< This event is generated when a local device received a positive response to a "get stream configuration" request.
```

Description

< This event is generated when a local device received a positive response to a "get stream configuration" request.

AVDTP_EVT_STREAM_CONFIGURED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_CONFIGURED 58    ///< This event is generated when a local device has successfully
```

```
configured a stream.
```

Description

< This event is generated when a local device has successfully configured a stream. < This event follows the AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the upper layer has accepted it. < This event is not generated if stream configuration was initiated by the local device.

AVDTP_EVT_STREAM_OPENED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_OPENED 60    ///< This event is generated when a local device has successfully
opened a stream.
```

Description

< This event is generated when a local device has successfully opened a stream. < This event follows the AVDTP_EVT_OPEN_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream opening was initiated by the local device.

AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED 8     ///< This event is generated when a local device
received a response (either positive or negative) to a "change stream configuration" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.

AVDTP_EVT_STREAM_RECONFIGURED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_RECONFIGURED 59    ///< This event is generated when a local device has
successfully reconfigured a stream.
```

Description

< This event is generated when a local device has successfully reconfigured a stream. < This event follows the AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream reconfiguration was initiated by the local device.

AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED 13    ///< This event is generated when a local device
received a response (either positive or negative) to a "exchange content protection control data" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.

AVDTP_EVT_STREAM_STARTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_STARTED 61    ///< This event is generated when a local device has successfully started a stream.
```

Description

< This event is generated when a local device has successfully started a stream. < This event follows the AVDTP_EVT_START_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream starting was initiated by the local device.

AVDTP_EVT_STREAM_SUSPENDED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_STREAM_SUSPENDED 63    ///< This event is generated when a local device has successfully suspended a stream.
```

Description

< This event is generated when a local device has successfully suspended a stream. < This event follows the AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream suspension was initiated by the local device.

AVDTP_EVT_SUSPEND_STREAM_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SUSPEND_STREAM_COMPLETED 12    ///< This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.

AVDTP_EVT_SUSPEND_STREAM_REQUESTED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SUSPEND_STREAM_REQUESTED 54    ///< This event is generated when a local device received "suspend stream" request.
```

Description

< This event is generated when a local device received "suspend stream" request.

AVDTP_MANAGER_STATE_CONNECTING Macro

File

avdtp.h

C

```
#define AVDTP_MANAGER_STATE_CONNECTING 1
```

Description

This is macro AVDTP_MANAGER_STATE_CONNECTING.

AVDTP_MANAGER_STATE_IDLE Macro

File

avdtp.h

C

```
#define AVDTP_MANAGER_STATE_IDLE 0
```

Description

This is macro AVDTP_MANAGER_STATE_IDLE.

AVDTP_MAX_STREAM_TRANSPORT_SESSION Macro

File

avdtp.h

C

```
#define AVDTP_MAX_STREAM_TRANSPORT_SESSION 3
```

Description

This is macro AVDTP_MAX_STREAM_TRANSPORT_SESSION.

AVDTP_MEDIA_TYPE_AUDIO Macro

File

avdtp.h

C

```
#define AVDTP_MEDIA_TYPE_AUDIO 0 ///< Audio.
```

Description

< Audio.

AVDTP_MEDIA_TYPE_MULTIMEDIA Macro

File

avdtp.h

C

```
#define AVDTP_MEDIA_TYPE_MULTIMEDIA 2 ///< Both Audio & Video.
```

Description

< Both Audio & Video.

AVDTP_MEDIA_TYPE_VIDEO Macro

File

avdtp.h

C

```
#define AVDTP_MEDIA_TYPE_VIDEO 1 ///< Video.
```

Description

< Video.

AVDTP_SEP_CAPABILITY_FLAG_CONTENT_PROTECTION Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_CONTENT_PROTECTION 0x08 ///< Content Prortection. A SEP is capable of transferring content protection packets.
```

Description

< Content Prortection. A SEP is capable of transferring content protection packets.

AVDTP_SEP_CAPABILITY_FLAG_HEADER_COMPRESSION Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_HEADER_COMPRESSION 0x10 ///< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.
```

Description

< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.

AVDTP_SEP_CAPABILITY_FLAG_MEDIA_CODEC Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_MEDIA_CODEC 0x40 ///< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.
```

Description

< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.

AVDTP_SEP_CAPABILITY_FLAG_MEDIA_TRANSPORT Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_MEDIA_TRANSPORT 0x01 ///< Media. A SEP is capable of transferring media (audio, video or both) packets.
```

Description

< Media. A SEP is capable of transferring media (audio, video or both) packets.

AVDTP_SEP_CAPABILITY_FLAG_MULTIPLEXING Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_MULTIPLEXING 0x20 ///< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.
```

Description

< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.

AVDTP_SEP_CAPABILITY_FLAG_RECOVERY Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_RECOVERY 0x04 ///< Recovery. A SEP is capable of transferring recovery packets.
```

Description

< Recovery. A SEP is capable of transferring recovery packets.

AVDTP_SEP_CAPABILITY_FLAG_REPORTING Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_REPORTING 0x02 ///< Reporting. A SEP is capable of transferring reporting packets.
```

Description

< Reporting. A SEP is capable of transferring reporting packets.

AVDTP_SEP_SERVICE_CAPABILITY_CONTENT_PROTECTION Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_CONTENT_PROTECTION 4 ///< Content Prortection. A SEP is capable of transferring content protection packets.
```

Description

< Content Prortection. A SEP is capable of transferring content protection packets.

AVDTP_SEP_SERVICE_CAPABILITY_HEADER_COMPRESSION Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_HEADER_COMPRESSION 5 ///< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.
```

Description

< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.

AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_CODEC Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_CODEC 7 ///< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.
```

Description

< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.

AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_TRANSPORT Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_TRANSPORT 1 ///< Media. A SEP is capable of transferring media (audio, video or both) packets.
```

Description

< Media. A SEP is capable of transferring media (audio, video or both) packets.

AVDTP_SEP_SERVICE_CAPABILITY_MULTIPLEXING Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_MULTIPLEXING 6 ///< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.
```

Description

< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.

AVDTP_SEP_SERVICE_CAPABILITY_RECOVERY Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_RECOVERY 3 ///< Recovery. A SEP is capable of transferring recovery packets.
```

Description

< Recovery. A SEP is capable of transferring recovery packets.

AVDTP_SEP_SERVICE_CAPABILITY_REPORTING Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_REPORTING 2 ///< Reporting. A SEP is capable of transferring reporting packets.
```

Description

< Reporting. A SEP is capable of transferring reporting packets.

AVDTP_SEP_STATE_FREE Macro

File

avdtp.h

C

```
#define AVDTP_SEP_STATE_FREE 0
```

Description

This is macro AVDTP_SEP_STATE_FREE.

AVDTP_SEP_STATE_IDLE Macro

File

avdtp.h

C

```
#define AVDTP_SEP_STATE_IDLE 1
```

Description

This is macro AVDTP_SEP_STATE_IDLE.

AVDTP_SEP_TYPE_SINK Macro

File

avdtp.h

C

```
#define AVDTP_SEP_TYPE_SINK 1 ///< Sink (usually a device like a headphones or BMW).
```

Description

< Sink (usually a device like a headphones or BMW).

AVDTP_SEP_TYPE_SOURCE Macro

File

avdtp.h

C

```
#define AVDTP_SEP_TYPE_SOURCE 0 ///< Source (usually a device like a phone, desktop or laptop).
```

Description

< Source (usually a device like a phone, desktop or laptop).

AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS 2      ///< The stream is closing transport channels.
```

Description

< The stream is closing transport channels.

AVDTP_STREAM_FLAG_LISTENING Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_FLAG_LISTENING 1
```

Description

This is macro AVDTP_STREAM_FLAG_LISTENING.

AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS 1      ///< The stream is opening transport channels.
```

Description

< The stream is opening transport channels.

AVDTP_STREAM_STATE_ABORTING Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_STATE_ABORTING 7      ///< The stream is aborting. This means that all transport
channels associated with the stream are being closed.
```

Description

< The stream is aborting. This means that all transport channels associated with the stream are being closed. < After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.

AVDTP_STREAM_STATE_CLOSING Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_STATE_CLOSING 6      ///< The stream is closing. This means that all transport
channels associated with the stream are being closed.
```

Description

< The stream is closing. This means that all transport channels associated with the stream are being closed. < After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.

AVDTP_STREAM_STATE_CONFIGURED Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_STATE_CONFIGURED 3      ///< The stream has been configured.
```

Description

< The stream has been configured.

AVDTP_STREAM_STATE_IDLE Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_STATE_IDLE 0      ///< The stream is idle. This can mean two things. The stream
specified by strm_handle does not exist
```

Description

< The stream is idle. This can mean two things. The stream specified by strm_handle does not exist < or the stream is closed.

AVDTP_STREAM_STATE_OPEN Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_STATE_OPEN 4      ///< The stream has been opened.
```

Description

< The stream has been opened.

AVDTP_STREAM_STATE_STREAMING Macro

File

avdtp.h

C

```
#define AVDTP_STREAM_STATE_STREAMING 5      ///< The stream has been started. Depending on the local SEP
type (source or sink) it means
```

Description

< The stream has been started. Depending on the local SEP type (source or sink) it means < that the stream can send or receive media packets.

AVDTP_TC_OPCODE_CONNECT Macro

File

avdtp_private.h

C

```
#define AVDTP_TC_OPCODE_CONNECT 0
```

Description

This is macro AVDTP_TC_OPCODE_CONNECT.

AVDTP_TC_OPCODE_DISCONNECT Macro

File

avdtp_private.h

C

```
#define AVDTP_TC_OPCODE_DISCONNECT 1
```

Description

This is macro AVDTP_TC_OPCODE_DISCONNECT.

AVDTP_TC_STATUS_ERROR Macro

File

avdtp_private.h

C

```
#define AVDTP_TC_STATUS_ERROR 0xFF
```

Description

This is macro AVDTP_TC_STATUS_ERROR.

AVDTP_TC_STATUS_SUCCESS Macro

File

avdtp_private.h

C

```
#define AVDTP_TC_STATUS_SUCCESS 0x00
```

Description

This is macro AVDTP_TC_STATUS_SUCCESS.

AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED Macro

File

avdtp.h

C

```
#define AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED 0
```

Description

This is macro AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED.

AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED Macro

File

avdtp.h

C

```
#define AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED 1
```

Description

This is macro AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED.

AVDTP_TRANSPORT_SESSION_TYPE_MEDIA Macro

File

avdtp.h

C

```
#define AVDTP_TRANSPORT_SESSION_TYPE_MEDIA 0 ///< Media (audio or video).
```

Description

< Media (audio or video).

AVDTP_TRANSPORT_SESSION_TYPE_RECOVERY Macro

File

avdtp.h

C

```
#define AVDTP_TRANSPORT_SESSION_TYPE_RECOVERY 2 ///< Recovery (currently not supported).
```

Description

< Recovery (currently not supported).

AVDTP_TRANSPORT_SESSION_TYPE_REPORTING Macro

File

avdtp.h

C

```
#define AVDTP_TRANSPORT_SESSION_TYPE_REPORTING 1 ///< Reporting (currently not supported).
```

Description

< Reporting (currently not supported).

bt_avdtp_connect Macro

File

avdtp.h

C

```
#define bt_avdtp_connect(mgr, remote_addr) _bt_avdtp_open_control_channel_ex(mgr, remote_addr,
HCI_CONFIG_ENABLE_AUTHENTICATION | HCI_CONFIG_ENABLE_ENCRYPTION)
```

Description

brief Connect to a remote device. ingroup avdtp

details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP_EVT_CTRL_CHANNEL_CONNECTED or AVDTP_EVT_CTRL_CHANNEL_CONNECTION_FAILED.

param mgr AVDTP manager. param remote_addr The address of a remote device.

return li c TRUE if connection establishment has been started. li c FALSE otherwise.

bt_avdtp_connect_ex Macro

File

avdtp.h

C

```
#define bt_avdtp_connect_ex(mgr, remote_addr, acl_config) _bt_avdtp_open_control_channel_ex(mgr,
remote_addr, acl_config)
```

Description

brief Connect to a remote device. ingroup avdtp

details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP_EVT_CTRL_CHANNEL_CONNECTED or AVDTP_EVT_CTRL_CHANNEL_CONNECTION_FAILED.

param mgr AVDTP manager. param remote_addr The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following values: li HCI_CONFIG_ENABLE_AUTHENTICATION li HCI_CONFIG_ENABLE_ENCRYPTION li HCI_CONFIG_BECOME_MASTER

return li c TRUE if connection establishment has been started. li c FALSE otherwise.

AVDTP_CMD_DELAYREPORT Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_DELAYREPORT 13
```

Description

This is macro AVDTP_CMD_DELAYREPORT.

AVDTP_CMD_GET_ALL_CAPABILITIES Macro

File

avdtp_control.h

C

```
#define AVDTP_CMD_GET_ALL_CAPABILITIES 12
```

Description

This is macro AVDTP_CMD_GET_ALL_CAPABILITIES.

AVDTP_CONTENT_PROTECTION_METHOD_SCMS_T Macro

File

avdtp.h

C

```
#define AVDTP_CONTENT_PROTECTION_METHOD_SCMS_T 2
```

Description

This is macro AVDTP_CONTENT_PROTECTION_METHOD_SCMS_T.

AVDTP_CTRL_MESSAGE_TYPE_GENERAL_REJECT Macro

File

avdtp_control.h

C

```
#define AVDTP_CTRL_MESSAGE_TYPE_GENERAL_REJECT 1
```

Description

This is macro AVDTP_CTRL_MESSAGE_TYPE_GENERAL_REJECT.

AVDTP_SCMS_T_CP_BIT Macro

File

avdtp.h

C

```
#define AVDTP_SCMS_T_CP_BIT 2
```

Description

This is macro AVDTP_SCMS_T_CP_BIT.

AVDTP_SCMS_T_L_BIT Macro

File

avdtp.h

C

```
#define AVDTP_SCMS_T_L_BIT 1
```

Description

This is macro AVDTP_SCMS_T_L_BIT.

bt_avdtp_evt_delay_report_completed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_delay_report_completed_t {
  bt_byte err_code;
} bt_avdtp_evt_delay_report_completed_t;
```

Members

Members	Description
bt_byte err_code;	< The result of the request. < li If the remote accepted the request c err_code == AVDTP_ERROR_SUCCESS. < li Otherwise c err_code == the error code returned by the remote party.

Description

brief Parameter to AVDTP_EVT_DELAYREPORT_COMPLETED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::delay_report_completed - when AVDTP received a response to a "delay report" request.

bt_cp_header_t Type

File

avdtp.h

C

```
typedef struct _bt_cp_header_s bt_cp_header_t;
```

Description

This is type bt_cp_header_t.

_bt_cp_header_s Structure

File

avdtp.h

C

```
struct _bt_cp_header_s {
  union {
    bt_byte scmst;
    struct {
      bt_byte* value;
      bt_uint length;
    } unknown;
  } header;
};
```

Description

This is record _bt_cp_header_s.

AVDTP_EVT_DELAYREPORT_COMPLETED Macro

File

avdtp.h

C

```
#define AVDTP_EVT_DELAYREPORT_COMPLETED 18    ///< This event is generated when a local device received a response (either positive or negative) to a "delay report" request.
```

Description

< This event is generated when a local device received a response (either positive or negative) to a "delay report" request.

AVDTP_EVT_SET_STREAM_CONFIGURATION Macro

File

avdtp.h

C

```
#define AVDTP_EVT_SET_STREAM_CONFIGURATION 19
```

Description

This is macro AVDTP_EVT_SET_STREAM_CONFIGURATION.

AVDTP_SEP_CAPABILITY_FLAG_DELAY_REPORTING Macro

File

avdtp.h

C

```
#define AVDTP_SEP_CAPABILITY_FLAG_DELAY_REPORTING 0x80 ///< Delat reporitng.
```

Description

< Delat reporitng.

AVDTP_SEP_SERVICE_CAPABILITY_DELAY_REPORTING Macro

File

avdtp.h

C

```
#define AVDTP_SEP_SERVICE_CAPABILITY_DELAY_REPORTING 8 ///< Delay Reporting.
```

Description

< Delay Reporting.

bt_avdtp_evt_ctrl_connection_failed_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_ctrl_connection_failed_s {
  bt_bdaddr_t* bdaddr;
} bt_avdtp_evt_ctrl_connection_failed_t;
```

Members

Members	Description
bt_bdaddr_t* bdaddr;	< the address of a remote device

Description

brief Parameter to AVDTP_EVT_CTRL_CONNECTION_FAILED event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_connection_failed - when a control channel between two devices has been established.

bt_avdtp_evt_set_stream_configuration_t Structure

File

avdtp.h

C

```
typedef struct _bt_avdtp_evt_set_stream_configuration_t {
  bt_byte err_code;
  bt_byte err_category;
  bt_avdtp_sep_t* sep;
  bt_byte int_sep_id;
  bt_avdtp_sep_capabilities_t* config;
  bt_byte strm_handle;
  bt_byte src_evt_code;
} bt_avdtp_evt_set_stream_configuration_t;
```

Description

brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION event ingroup avdtp

details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration - when AVDTP received a "set stream configuration" request.

AVRCP Functions

_bt_avrcp_allocate_browsing_cmd Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_browsing_cmd(bt_byte pduid);
```

Description

This is function _bt_avrcp_allocate_browsing_cmd.

_bt_avrcp_allocate_bt_specific_cmd Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_bt_specific_cmd(bt_byte ctype, bt_byte pduid);
```

Description

This is function _bt_avrcp_allocate_bt_specific_cmd.

_bt_avrcp_allocate_bt_specific_response Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_bt_specific_response(bt_byte ctype, bt_byte tran_id, bt_byte pduid);
```

Description

This is function _bt_avrcp_allocate_bt_specific_response.

_bt_avrcp_allocate_channel Function

File

avrcp_private.h

C

```
bt_avrcp_channel_t* _bt_avrcp_allocate_channel(bt_avrcp_mgr_t* mgr);
```

Description

void _bt_avctp_l2cap_read_data_callback(struct _channel *pch, bt_byte* pdata, bt_int len); bt_avctp_channel_t* _bt_avctp_find_channel(bt_avctp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_uint profile_id);

_bt_avrcp_allocate_cmd Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_cmd();
```

Description

This is function _bt_avrcp_allocate_cmd.

_bt_avrcp_allocate_response Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_response(bt_byte tran_id);
```

Description

This is function _bt_avrcp_allocate_response.

_bt_avrcp_allocate_simple_panel_cmd Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_simple_panel_cmd(bt_byte ctype, bt_byte opid, bt_byte button_state);
```

Description

This is function _bt_avrcp_allocate_simple_panel_cmd.

_bt_avrcp_allocate_simple_panel_response Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_simple_panel_response(bt_byte ctype, bt_byte tran_id, bt_byte opid,
bt_byte button_state);
```

Description

This is function _bt_avrcp_allocate_simple_panel_response.

_bt_avrcp_find_channel Function

File

avrcp_private.h

C

```
bt_avrcp_channel_t* _bt_avrcp_find_channel(bt_avrcp_mgr_t* mgr, bt_avctp_channel_t* avctp_channel);
```

Description

This is function _bt_avrcp_find_channel.

_bt_avrcp_free_channel Function

File

avrcp_private.h

C

```
void _bt_avrcp_free_channel(bt_avrcp_channel_t* channel);
```

Description

This is function _bt_avrcp_free_channel.

_bt_avrcp_free_cmd Function

File

avrcp_private.h

C

```
void _bt_avrcp_free_cmd(bt_av_command_t* command);
```

Description

This is function _bt_avrcp_free_cmd.

_bt_avrcp_get_tick_count Function

File

avrcp_private.h

C

```
bt_long _bt_avrcp_get_tick_count();
```

Description

This is function _bt_avrcp_get_tick_count.

_bt_avrcp_handle_add_to_now_playing Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_add_to_now_playing(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data, bt_uint
len);
```

Description

This is function _bt_avrcp_handle_add_to_now_playing.

_bt_avrcp_handle_command Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_command(bt_avrcp_mgr_t* mgr, bt_avctp_evt_command_received_t* evt_param);
```

Description

This is function _bt_avrcp_handle_command.

_bt_avrcp_handle_command_sent Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_command_sent(bt_avrcp_mgr_t* mgr, bt_avctp_evt_command_sent_t* evt_param);
```

Description

This is function _bt_avrcp_handle_command_sent.

_bt_avrcp_handle_get_capabilities Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_get_capabilities(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data, bt_uint
len);
```

Description

This is function _bt_avrcp_handle_get_capabilities.

_bt_avrcp_handle_get_current_player_application_setting_value Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_get_current_player_application_setting_value(bt_avrcp_channel_t* channel, bt_byte
ctype, bt_byte* data, bt_uint len, void* response_buffer);
```

Description

This is function _bt_avrcp_handle_get_current_player_application_setting_value.

_bt_avrcp_handle_get_element_attributes Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_get_element_attributes(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data,
bt_uint len);
```

Description

This is function _bt_avrcp_handle_get_element_attributes.

_bt_avrcp_handle_get_play_status Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_get_play_status(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_handle_get_play_status.

_bt_avrcp_handle_get_player_application_setting_attribute_text Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_get_player_application_setting_attribute_text(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data, bt_uint len, void* response_buffer);
```

Description

This is function _bt_avrcp_handle_get_player_application_setting_attribute_text.

_bt_avrcp_handle_get_player_application_setting_value_text Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_get_player_application_setting_value_text(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data, bt_uint len, void* response_buffer);
```

Description

This is function _bt_avrcp_handle_get_player_application_setting_value_text.

_bt_avrcp_handle_inform_battery_status_of_ct Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_inform_battery_status_of_ct(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_handle_inform_battery_status_of_ct.

_bt_avrcp_handle_inform_displayable_character_set Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_inform_displayable_character_set(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_handle_inform_displayable_character_set.

_bt_avrcp_handle_list_player_application_setting_attributes Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_list_player_application_setting_attributes(bt_avrcp_channel_t* channel, bt_byte
ctype, bt_byte* data, bt_uint len, void* response_buffer);
```

Description

This is function _bt_avrcp_handle_list_player_application_setting_attributes.

_bt_avrcp_handle_list_player_application_setting_values Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_list_player_application_setting_values(bt_avrcp_channel_t* channel, bt_byte ctype,
bt_byte* data, bt_uint len, void* response_buffer);
```

Description

This is function _bt_avrcp_handle_list_player_application_setting_values.

_bt_avrcp_handle_play_item Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_play_item(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_handle_play_item.

_bt_avrcp_handle_register_notification Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_register_notification(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte event_id,
bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_handle_register_notification.

_bt_avrcp_handle_request_continuing_response Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_request_continuing_response(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte*
data, bt_uint len);
```

Description

This is function _bt_avrcp_handle_request_continuing_response.

_bt_avrcp_handle_response Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_response(bt_avrcp_mgr_t* mgr, bt_avctp_evt_response_received_t* evt_param);
```

Description

This is function _bt_avrcp_handle_response.

_bt_avrcp_handle_response_sent Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_response_sent(bt_avrcp_mgr_t* mgr, bt_avctp_evt_response_sent_t* evt_param);
```

Description

This is function _bt_avrcp_handle_response_sent.

_bt_avrcp_handle_set_absolute_volume Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_set_absolute_volume(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data,
bt_uint len);
```

Description

This is function _bt_avrcp_handle_set_absolute_volume.

_bt_avrcp_handle_set_addressed_player Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_set_addressed_player(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte* data,
bt_uint len);
```

Description

This is function _bt_avrcp_handle_set_addressed_player.

_bt_avrcp_handle_set_player_application_setting_value Function

File

avrcp_private.h

C

```
void _bt_avrcp_handle_set_player_application_setting_value(bt_avrcp_channel_t* channel, bt_byte ctype,
bt_byte* data, bt_uint len, void* response_buffer);
```

Description

This is function _bt_avrcp_handle_set_player_application_setting_value.

_bt_avrcp_init_signal Function

File

avrcp_private.h

C

```
void _bt_avrcp_init_signal();
```

Description

void _bt_avctp_init_message_buffers();

_bt_avrcp_init_timer Function

File

avrcp_private.h

C

```
void _bt_avrcp_init_timer();
```

Description

This is function _bt_avrcp_init_timer.

_bt_avrcp_register_next_notification Function

File

avrcp_private.h

C

```
void _bt_avrcp_register_next_notification(bt_avrcp_channel_t* channel);
```

Description

This is function _bt_avrcp_register_next_notification.

_bt_avrcp_register_pending_notification Function

File

avrcp_private.h

C

```
void _bt_avrcp_register_pending_notification(bt_avrcp_channel_t* channel);
```

Description

This is function _bt_avrcp_register_pending_notification.

_bt_avrcp_send_notifications Function

File

avrcp_private.h

C

```
void _bt_avrcp_send_notifications(bt_avrcp_channel_t* channel);
```

Description

This is function _bt_avrcp_send_notifications.

_bt_avrcp_set_signal Function

File

avrcp_private.h

C

```
void _bt_avrcp_set_signal();
```

Description

This is function _bt_avrcp_set_signal.

_bt_avrcp_start_timer Function

File

avrcp_private.h

C

```
void _bt_avrcp_start_timer();
```

Description

This is function _bt_avrcp_start_timer.

_bt_avrcp_tg_handle_get_capabilities Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_get_capabilities(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte tran_id,
bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_get_capabilities.

_bt_avrcp_tg_handle_get_element_attributes Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_get_element_attributes(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte
tran_id, bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_get_element_attributes.

_bt_avrcp_tg_handle_get_play_status Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_get_play_status(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte tran_id,
bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_get_play_status.

_bt_avrcp_tg_handle_inform_battery_status_of_ct Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_inform_battery_status_of_ct(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte
tran_id, bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_inform_battery_status_of_ct.

_bt_avrcp_tg_handle_inform_displayable_character_set Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_inform_displayable_character_set(bt_avrcp_channel_t* channel, bt_byte ctype,
bt_byte tran_id, bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_inform_displayable_character_set.

_bt_avrcp_tg_handle_register_notification Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_register_notification(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte tran_id,
bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_register_notification.

_bt_avrcp_tg_handle_set_absolute_volume Function

File

avrcp_private.h

C

```
void _bt_avrcp_tg_handle_set_absolute_volume(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte tran_id,
bt_byte* data, bt_uint len);
```

Description

This is function _bt_avrcp_tg_handle_set_absolute_volume.

_bt_avrcp_write_command_header Function

File

avrcp_private.h

C

```
bt_int _bt_avrcp_write_command_header(bt_av_command_t* command);
```

Description

This is function _bt_avrcp_write_command_header.

bt_avrcp_abort_continuing_response Function

File

avrcp_private.h

C

```
bt_bool bt_avrcp_abort_continuing_response(bt_avrcp_channel_t* channel, bt_byte pduid);
```

Description

This is function bt_avrcp_abort_continuing_response.

bt_avrcp_add_to_now_playing Function

File

avrcp.h

C

```
bt_bool bt_avrcp_add_to_now_playing(bt_avrcp_channel_t* channel, bt_byte scope, bt_av_element_id_t*
element_id, bt_uint counter);
```

Description

brief Add to "now playing" list. ingroup avrcp

details This function adds a media element specified by p element_id to the "now playing" list on the target.

param channel AVRCP channel. param scope The scope in which the p element_id is valid. This value can be on the following values: li AVC_SCOPE_MEDIA_PLAYER_LIST li AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM li AVC_SEARCH li AVC_NOW_PLAYING

param element_id UID of the media element to be added to the "now playing" list. param counter UID counter.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_avrcpt_request_continuing_response Function

File

avrcp_private.h

C

```
bt_bool bt_avrcp_avrcpt_request_continuing_response(bt_avrcp_channel_t* channel, bt_byte pduid);
```

Description

This is function bt_avrcp_avrcpt_request_continuing_response.

bt_avrcp_cancel_find Function

File

avrcp.h

C

```
bt_bool bt_avrcp_cancel_find();
```

Description

brief Cancel finding Targets ingroup avrcp

details This function stops AVRCP layer looking for targets on nearby devices. As a result of this operation the AVRCP_EVT_SEARCH_COMPLETED event will be generated.

return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.

bt_avrcp_cancel_listen Function

File

avrcp.h

C

```
void bt_avrcp_cancel_listen(bt_avrcp_channel_t* channel);
```

Description

brief Cancel listening for incoming connections. ingroup avrcp

details This function stops listening for incoming connections on a specified channel.

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_connect Function

File

avrcp.h

C

```
bt_bool bt_avrcp_connect(bt_avrcp_channel_t* channel, bt_bdaddr_t* remote_address);
```

Description

brief Connect to a remote device. ingroup avrcp

details This function establishes a connection to a remote device specified by the p remote_address. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVRCP callback. The events generated will either be AVRCP_EVT_CONTROL_CHANNEL_CONNECTED or AVRCP_EVT_CONTROL_CONNECTION_FAILED.

param channel AVRCP channel. param remote_address The address of a remote device.

return li c TRUE if connection establishment has been started. li c FALSE otherwise.

bt_avrcp_create_channel Function

File

avrcp.h

C

```
bt_avrcp_channel_t* bt_avrcp_create_channel(bt_avrcp_mgr_t* mgr, bt_bool create_browsing_channel);
```

Description

brief Allocate AVRCP channel ingroup avrcp

details This function allocates a new incoming AVRCP channel. The channel is intended to be used to accept a connection from a remote device. There can be only one incoming channel.

param mgr AVRCP manager. param create_browsing_channel Defines weather a browsing channel will be created.

return li A pointer to the new AVRCP channel if the function succeeds. li c NULL otherwise.

bt_avrcp_create_outgoing_channel Function

File

avrcp.h

C

```
bt_avrcp_channel_t* bt_avrcp_create_outgoing_channel(bt_avrcp_mgr_t* mgr, bt_bool create_browsing_channel);
```

Description

brief Allocate AVRCP channel ingroup avrcp

details This function allocates a new outgoing AVRCP channel. The channel is intended to be used to create a connection to a remote device. There can be multiple outgoing channels.

param mgr AVRCP manager. param create_browsing_channel Defines weather a browsing channel will be created.

return li A pointer to the new AVRCP channel if the function succeeds. li c NULL otherwise.

bt_avrcp_destroy_channel Function

File

avrcp.h

C

```
bt_bool bt_avrcp_destroy_channel(bt_avrcp_channel_t* channel);
```

Description

brief Destroy AVRCP channel. ingroup avrcp

details This function frees memory used by the channel. The channel has to exist and be in the "idle" state for this function to succeed. I.e. the channel has to be disconnected before this function can be called.

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_disconnect Function

File

avrcp.h

C

```
bt_bool bt_avrcp_disconnect(bt_avrcp_channel_t* channel);
```

Description

brief Disconnect from a remote device. ingroup avrcp

details This function closes a connection to a remote device.

param channel AVRCP channel.

return li c TRUE if disconnection has been started. li c FALSE otherwise. No events will be generated.

bt_avrcp_find_controller Function

File

avrcp.h

C

```
bt_bool bt_avrcp_find_controller(bt_bdaddr_t* deviceAddress, bt_avrcp_find_callback_fp callback,
bt_sdp_client_callback_fp client_callback, void* callback_param);
```

Description

brief Find Controller ingroup avrcp

details This function looks for a controller on a remote device specified by p deviceAddress.

param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer to arbitrary data to be passed to the p callback and p client_callback callbacks.

return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.

bt_avrcp_find_target Function

File

avrcp.h

C

```
bt_bool bt_avrcp_find_target(bt_bdaddr_t* deviceAddress, bt_avrcp_find_callback_fp callback,
bt_sdp_client_callback_fp client_callback, void* callback_param);
```

Description

brief Find Target ingroup avrcp

details This function looks for a target on a remote device specified by p deviceAddress.

param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer to arbitrary data to be passed to the p callback and p client_callback callbacks.

return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.

bt_avrcp_find_targets Function

File

avrcp.h

C

```
bt_bool bt_avrcp_find_targets(bt_byte search_length);
```

Description

brief Find Targets ingroup avrcp

details This function looks for targets on nearby devices. The AVRCP_EVT_SEARCH_COMPLETED event is generated when the search has completed.

param search_length The amount of time the search will be performed for.

return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.

bt_avrcp_get_browsing_channel_state Function

File

avrcp.h

C

```
bt_byte bt_avrcp_get_browsing_channel_state(bt_avrcp_channel_t* channel);
```

Description

brief Get AVCTP browsing channel state. ingroup avrcp

details This function returns status of the AVCTP browsing channel.

param channel AVRCP channel.

return Returns of the following values: li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING

bt_avrcp_get_channel_remote_address Function

File

avrcp.h

C

```
bt_bdaddr_t* bt_avrcp_get_channel_remote_address(bt_avrcp_channel_t* channel);
```

Description

brief Get channel's remote BT address. ingroup avrcp

details This function returns the address of the remote device associated with the channel.

param channel AVRCP channel.

return li The address of the remote device if channel is connected. li NULL otherwise.

bt_avrcp_get_company_id_list Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_company_id_list(bt_avrcp_channel_t* channel);
```

Description

brief Get Company ID list. ingroup avrcp

details This function requests a list of company id's supported by the remote device

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_control_channel_state Function

File

avrcp.h

C

```
bt_byte bt_avrcp_get_control_channel_state(bt_avrcp_channel_t* channel);
```

Description

brief Get AVCTP control channel state. ingroup avrcp

details This function returns status of the AVCTP control channel.

param channel AVRCP channel.

return Returns of the following values: li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING

bt_avrcp_get_current_player_application_setting_value Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_current_player_application_setting_value(bt_avrcp_channel_t* channel,
bt_av_player_setting_current_values_t* response_buffer);
```

Description

brief Get current player setting values. ingroup avrcp

details This function requests a list of current set values for the player application on the target. The list of attribute ids whose values have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_setting_current_values_t::setting_id_list to a list of player setting attribute ids, bt_av_player_setting_current_values_t::count to the number of entries in the list, bt_av_player_setting_current_values_t::setting_value_id_list to a buffer where returned values will be stored.

param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_current_values_t structure initialized as stated above.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_element_attributes Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_element_attributes(bt_avrcp_channel_t* channel, bt_av_element_id_t* element_id,
bt_uint attr_mask);
```

Description

brief Get media element attributes. ingroup avrcp

details This function requests the attributes of the element specified with p element_id.

note Currently p element_id is ignored. The AVRCP specification defines that only UID 0 can be used to return the attributes of the current track.

param channel AVRCP channel. param element_id UID of the media element whose attributes are requested. param attr_mask Bitmask that defines which attributes are requested. This value can be a combination of the following values: li AVC_MEDIA_ATTR_FLAG_TITLE li AVC_MEDIA_ATTR_FLAG_ARTIST li AVC_MEDIA_ATTR_FLAG_ALBUM li AVC_MEDIA_ATTR_FLAG_NUMBER li AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER li AVC_MEDIA_ATTR_FLAG_GENRE li AVC_MEDIA_ATTR_FLAG_PLAYING_TIME li AVC_MEDIA_ATTR_FLAG_ALL

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_hci_connection Function

File

avrcp.h

C

```
bt_hci_conn_state_p* bt_avrcp_get_hci_connection(bt_avrcp_channel_t* channel);
```

Description

brief Get HCI connection for a channel ingroup avrcp

details This function returns a pointer to a structure that describes an HCI connection a channel is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect.

param channel AVRCP channel.

return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a channel specified by the p channel parameter li does not exist or there is no HCI connection between local and remote devices associated with the channel.

bt_avrcp_get_mgr Function

File

avrcp.h

C

```
bt_avrcp_mgr_t* bt_avrcp_get_mgr();
```

Description

brief Return a pointer to an instance of the AVRCP manager. ingroup avrcp

details This function returns a pointer to an instance of the AVRCP manager. There is only one instance of the manager allocated by the stack.

bt_avrcp_get_play_status Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_play_status(bt_avrcp_channel_t* channel, bt_uint repeat_interval);
```

Description

brief Get playback status. ingroup avrcp

details This function requests the status of the currently playing media at the target.

param channel AVRCP channel. param repeat_interval Interval in milliseconds at which AVRCP polls the target for playback status. If 0 is passed polling is stopped.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_player_application_setting_attr_text Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_player_application_setting_attr_text(bt_avrcp_channel_t* channel,
```

```
bt_av_player_settings_text_t* response_buffer);
```

Description

brief Get player setting attribute text. ingroup avrcp

details This function requests the target device to provide supported player application setting attribute displayable text for the provided player application setting attributes. The list of attribute ids whose displayable text have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_settings_text_t::setting_id_list to a list of player setting attribute ids, bt_av_player_settings_text_t::count to the number of entries in the list, bt_av_player_settings_text_t::setting_text_list to a buffer where returned values will be stored.

param channel AVRCP channel. param response_buffer Pointer to bt_av_player_settings_text_t structure initialized as stated above.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_player_application_setting_value_text Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_player_application_setting_value_text(bt_avrcp_channel_t* channel, bt_byte attr_id,
bt_av_player_setting_values_text_t* response_buffer);
```

Description

brief Get player setting value text. ingroup avrcp

details This function request the target device to provide target supported player application setting value displayable text for the provided player application setting attribute values. The list of attribute ids whose value displayable text have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_setting_values_text_t::setting_value_id_list to a list of player setting attribute value ids, bt_av_player_setting_values_text_t::count to the number of entries in the list, bt_av_player_setting_values_text_t::setting_value_text_list to a buffer where returned values will be stored.

param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_values_text_t structure initialized as stated above.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_subuint_info Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_get_subuint_info(bt_avrcp_channel_t* channel);
```

Description

brief Get subunit info ingroup avrcp

details This function is used to request subunit info from the target.

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_supported_event_id_list Function

File

avrcp.h

C

```
bt_bool bt_avrcp_get_supported_event_id_list(bt_avrcp_channel_t* channel);
```

Description

brief Get supported events. ingroup avrcp

details This function requests a list of events supported by the remote device

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_get_unit_info Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_get_unit_info(bt_avrcp_channel_t* channel);
```

Description

brief Get unit info ingroup avrcp

details This function is used to request unit info from the target.

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_inform_battery_status Function

File

avrcp.h

C

```
bt_bool bt_avrcp_inform_battery_status(bt_avrcp_channel_t* channel, bt_byte status);
```

Description

brief Inform controller's battery status. ingroup avrcp

details This function is used to inform the target about the controller's battery status.

param channel AVRCP channel. param status Battery status. This can be one of the following values: li AVC_BATTERY_STATUS_NORMAL li AVC_BATTERY_STATUS_WARNING li AVC_BATTERY_STATUS_CRITICAL li AVC_BATTERY_STATUS_EXTERNAL li AVC_BATTERY_STATUS_FULL_CHARGE

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_inform_displayable_character_set Function

File

avrcp.h

C

```
bt_bool bt_avrcp_inform_displayable_character_set(bt_avrcp_channel_t* channel, bt_uint* charset_list,
bt_byte charset_count);
```

Description

brief Inform displayable character set. ingroup avrcp

details This function informs the list of character set supported by the controller to the target.

param channel AVRCP channel. param charset_list List of displayable character sets. param charset_count Number of entries in the list.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_init_controller Function

File

avrcp.h

C

```
void bt_avrcp_init_controller();
```

Description

brief Initialize AVRCP to be used in controller mode. ingroup avrcp

details This function initializes the AVRCP layer of the stack in controller mode. It must be called prior to any other AVRCP function can be called.

bt_avrcp_init_target Function

File

avrcp.h

C

```
void bt_avrcp_init_target(bt_ulong company_id, bt_uint supported_events);
```

Description

brief Initialize AVRCP to be used in target mode. ingroup avrcp

details This function initializes the AVRCP layer of the stack in target mode. It must be called prior to any other AVRCP function can be called.

param company_id The 24-bit unique ID obtained from the IEEE Registration Authority Committee. If the vendor of a TG device does not have the unique ID, the value 0xFFFFFF may be used. param supported_events Bitmask that specifies events supported by the target. This value can be a combination of the following values: li AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED li AVC_EVENT_FLAG_TRACK_CHANGED li AVC_EVENT_FLAG_TRACK_REACHED_END li AVC_EVENT_FLAG_TRACK_REACHED_START li AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED li AVC_EVENT_FLAG_BATT_STATUS_CHANGED li AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED li AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED li AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED li AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED li AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED li AVC_EVENT_FLAG_UIDS_CHANGED li AVC_EVENT_FLAG_VOLUME_CHANGED

bt_avrcp_list_player_application_setting_attributes Function

File

avrcp.h

C

```
bt_bool bt_avrcp_list_player_application_setting_attributes(bt_avrcp_channel_t* channel,
bt_av_player_settings_t* response_buffer);
```

Description

brief Get supported player setting attributes. ingroup avrcp

details This function request the target device to provide target supported player application setting attributes. The list of attribute ids is stored in the setting_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_settings_t::setting_id_list to a buffer where returned values will be stored and bt_av_player_settings_t::count to the number of entries in the list.

param channel AVRCP channel. param response_buffer Pointer to bt_av_player_settings_t structure initialized as stated above.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_list_player_application_setting_values Function

File

avrcp.h

C

```
bt_bool bt_avrcp_list_player_application_setting_values(bt_avrcp_channel_t* channel, bt_byte attr_id,
bt_av_player_setting_values_t* response_buffer);
```

Description

brief Get player setting attribute values. ingroup avrcp

details This function requests the target device to list the set of possible values for the requested player application setting attribute. The list of attribute value ids is stored in the setting_value_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_setting_values_t::setting_value_id_list to a buffer where returned values will be stored and bt_av_player_setting_values_t::count to the number of entries in the list.

param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_values_t structure initialized as stated above.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_listen Function

File

avrcp.h

C

```
bt_bool bt_avrcp_listen(bt_avrcp_channel_t* channel);
```

Description

brief Listen for incoming connections. ingroup avrcp

details This function enables incoming connections on the specified AVRCP channel.

param channel AVRCP channel.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_play_item Function

File

avrcp.h

C

```
bt_bool bt_avrcp_play_item(bt_avrcp_channel_t* channel, bt_byte scope, bt_av_element_id_t* element_id,
bt_uint counter);
```

Description

brief Play media item. ingroup avrcp

details This function starts playing an item indicated by the UID.

param channel AVRCP channel. param scope The scope in which the p element_id is valid. This value can be on the following values: li AVC_SCOPE_MEDIA_PLAYER_LIST li AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM li AVC_SEARCH li AVC_NOW_PLAYING param element_id UID of the media element to be played. param counter UID counter.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_register_notification Function

File

avrcp.h

C

```
bt_bool bt_avrcp_register_notification(bt_avrcp_channel_t* channel, bt_byte event_id, bt_ulong
playback_interval);
```

Description

brief Register notification. ingroup avrcp

details This function registers with the target to receive notifications asynchronously based on specific events occurring.

param channel AVRCP channel. param event_id Event Id. This value can be one of the following values: li AVC_EVENT_PLAYBACK_STATUS_CHANGED li AVC_EVENT_TRACK_CHANGED li AVC_EVENT_TRACK_REACHED_END li AVC_EVENT_TRACK_REACHED_START li AVC_EVENT_PLAYBACK_POS_CHANGED li AVC_EVENT_BATT_STATUS_CHANGED li AVC_EVENT_SYSTEM_STATUS_CHANGED li AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED li AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED li AVC_EVENT_AVAILABLE_PLAYERS_CHANGED li AVC_EVENT_ADDRESSED_PLAYER_CHANGED li AVC_EVENT_UIDS_CHANGED li AVC_EVENT_VOLUME_CHANGED param playback_interval The time interval (in seconds) at which the change in playback position will be notified. Applicable for AVC_EVENT_PLAYBACK_POS_CHANGED event.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_register_notifications Function

File

avrcp.h

C

```
bt_bool bt_avrcp_register_notifications(bt_avrcp_channel_t* channel, bt_uint event_mask);
```

Description

brief Register notifications. ingroup avrcp

details This function registers with the target to receive notifications asynchronously based on specific events occurring. This function is used to register multiple notifications with one call.

note This function cannot be used to register for AVC_EVENT_PLAYBACK_POS_CHANGED event.

param channel AVRCP channel. param event_mask Bitmask that specifies which events to register for. This value can be a combination of the following values: li AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED li AVC_EVENT_FLAG_TRACK_CHANGED li AVC_EVENT_FLAG_TRACK_REACHED_END li AVC_EVENT_FLAG_TRACK_REACHED_START li AVC_EVENT_FLAG_BATT_STATUS_CHANGED li AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED li AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED li AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED li AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED li AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED li AVC_EVENT_FLAG_UIDS_CHANGED li AVC_EVENT_FLAG_VOLUME_CHANGED li AVC_EVENT_FLAG_ALL

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_send_button_click Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_send_button_click(bt_avrcp_channel_t* channel, bt_byte button_id);
```

Description

brief Send AV/C Panel Subunit "click" PASS THROUGH command ingroup avrcp

details This function is used to send a button click. Two PATH THROUGTH commands are sent. The first command has button state set to AVC_PANEL_BUTTON_PRESSED. The second command gas button state set to AVC_PANEL_BUTTON_RELEASED

param channel AVRCP channel. param button_id Operation Id. This value can be on of the AVC_PANEL_OPID_... constants

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_send_cmd Function

File

avrcp.h

C

```
bt_bool bt_avrcp_send_cmd(bt_avrcp_channel_t* channel, bt_av_command_t* command);
```

Description

brief Send AVRCP command ingroup avrcp

details This function sends a command to the remote device.

param channel AVRCP channel. param command Command to be sent.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_send_panel_control Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_send_panel_control(bt_avrcp_channel_t* channel, bt_byte opid, bt_byte button_state);
```

Description

brief Send AV/C Panel Subunit "control" PASS THROUGH command ingroup avrcp

details This function is used to send AV/C Panel Subunit PASS THROUGH command with command type set to AVC_CTYPE_CONTROL.

param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants param button_state

Button state. This can be on of the following values: li AVC_PANEL_BUTTON_PRESSED li AVC_PANEL_BUTTON_RELEASED

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_send_press_panel_control Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_send_press_panel_control(bt_avrcp_channel_t* channel, bt_byte opid);
```

Description

brief Send AV/C Panel Subunit "pressed" PASS THROUGH command ingroup avrcp

details This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC_PANEL_BUTTON_PRESSED.

param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_send_simple_panel_cmd Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_send_simple_panel_cmd(bt_avrcp_channel_t* channel, bt_byte ctype, bt_byte opid, bt_byte
button_state);
```

Description

brief Send AV/C Panel Subunit PASS THROUGH command ingroup avrcp

details This function is used to send AV/C Panel Subunit PASS THROUGH command to the target.

param channel AVRCP channel. param ctype Command type. This value can be on of the following values: li AVC_CTYPE_CONTROL 0 li AVC_CTYPE_STATUS 1 li AVC_CTYPE_SPECIFIC_IQUIRY 2 li AVC_CTYPE_NOTIFY 3 li AVC_CTYPE_GENERAL_INQUORY 4 param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants param button_state Button state. This can be on of the following values: li AVC_PANEL_BUTTON_PRESSED li AVC_PANEL_BUTTON_RELEASED

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_set_absolute_volume Function

File

avrcp.h

C

```
bt_bool bt_avrcp_set_absolute_volume(bt_avrcp_channel_t* channel, bt_byte volume);
```

Description

brief Set absolute volume. ingroup avrcp

details This function is used to set an absolute volume to be used by the rendering device.

param channel AVRCP channel. param volume Volume

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_set_addressed_player Function

File

avrcp.h

C

```
bt_bool bt_avrcp_set_addressed_player(bt_avrcp_channel_t* channel, bt_uint player_id);
```

Description

brief Set addressed player. ingroup avrcp

details This function is used to inform the target of which media player the controller wishes to control.

param channel AVRCP channel. param player_id Player Id.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_set_player_application_setting_value Function

File

avrcp.h

C

```
bt_bool bt_avrcp_set_player_application_setting_value(bt_avrcp_channel_t* channel, bt_byte* attr_id_list,
bt_byte* attr_value_list, bt_byte attr_id_count);
```

Description

brief Set player setting attribute values. ingroup avrcp

details This function requests to set the player application setting list of player application setting values on the target device for the corresponding defined list of setting attributes. for the requested player application setting attribute. The list of attribute value ids is stored in the setting_value_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_setting_values_t::setting_value_id_list to a buffer where returned values will be stored and bt_av_player_setting_values_t::count to the number of entries in the list.

param channel AVRCP channel. param attr_id_list List of setting attribute ids. param attr_value_list List of setting attribute value ids. param attr_id_count The number of entries in both lists.

return li c TRUE if the function succeeds. li c FALSE otherwise.

bt_avrcp_start Function

File

avrcp.h

C

```
void bt_avrcp_start(bt_avrcp_mgr_t* mgr, bt_avrcp_mgr_callback_fp callback, void* callback_param);
```

Description

brief Start the AVRCP layer. ingroup avrcp

details In order to be notified of various events a consumer of the AVRCP layer has to register a callback function. The stack will call the callback function whenever a new event has been generated passing the code of the event as the second parameter. bt_avrcp_start() stores pointers to the c callback and c callback_param in the bt_avrcp_mgr_t structure.

param mgr AVRCP manager. param callback The callback function that will be called when the AVRCP generates an event. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

bt_avrcp_tg_send_element_attributes Function

File

avrcp.h

C

```
bt_bool bt_avrcp_tg_send_element_attributes(bt_avrcp_channel_t* channel, bt_byte tran_id,
bt_av_element_attribute_t* attrs, bt_byte attr_count);
```

Description

brief Send media element attributes ingroup avrcp

details This function is used to send the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_SYSTEM_STATUS_CHANGED event will be notified.

param status Battery status

bt_avrcp_tg_set_absolute_volume Function

File

avrcp.h

C

```
void bt_avrcp_tg_set_absolute_volume(bt_byte volume);
```

Description

brief Set absolute volume ingroup avrcp

details This function is used to set the absolute volume when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_VOLUME_CHANGED event will be notified.

param track_id Track Id param song_length The length of the current track. param song_position The position of the current track.

bt_avrcp_tg_set_battery_status Function

File

avrcp.h

C

```
void bt_avrcp_tg_set_battery_status(bt_byte status);
```

Description

brief Set battery status ingroup avrcp

details This function is used to set the battery status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_BATT_STATUS_CHANGED event will be notified.

param status Battery status

bt_avrcp_tg_set_channel_absolute_volume Function

File

avrcp.h

C

```
void bt_avrcp_tg_set_channel_absolute_volume(bt_avrcp_channel_t* channel, bt_byte volume);
```

Description

This is function bt_avrcp_tg_set_channel_absolute_volume.

bt_avrcp_tg_set_current_track Function

File

avrcp.h

C

```
void bt_avrcp_tg_set_current_track(bt_av_element_id_t* track_id, bt_ulong song_length, bt_ulong
song_position);
```

Description

brief Set current track ingroup avrcp

details This function is used to set the current track when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_TRACK_CHANGED event will be notified.

param track_id Track Id param song_length The length of the current track. param song_position The position of the current track.

bt_avrcp_tg_set_play_status Function

File

avrcp.h

C

```
void bt_avrcp_tg_set_play_status(bt_ulong song_length, bt_ulong song_position, bt_byte play_status);
```

Description

local target control *

- brief Set playback status
- ingroup avrcp

*

- details This function is used to set the playback status when AVRCP is running in target mode.
- If there are active connections with controllers, the ones that registered
- for AVC_EVENT_PLAYBACK_STATUS_CHANGED event will be notified.

*

- param song_length The length of the current track.
- param song_position The position of the current track.
- param play_status Playback status. This value can be one of the following values:
- li AVC_PLAY_STATUS_STOPPED
- li AVC_PLAY_STATUS_PLAYING
- li AVC_PLAY_STATUS_PAUSED
- li AVC_PLAY_STATUS_FW_SEEK
- li AVC_PLAY_STATUS_REV_SEEK

bt_avrcp_tg_set_system_status Function

File

avrcp.h

C

```
void bt_avrcp_tg_set_system_status(bt_byte status);
```

Description

brief Set system status ingroup avrcp

details This function is used to set the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_SYSTEM_STATUS_CHANGED event will be notified.

param status Battery status

_bt_avrcp_allocate_browsing_response Function

File

avrcp_private.h

C

```
bt_av_command_t* _bt_avrcp_allocate_browsing_response(bt_byte tran_id, bt_byte pduid);
```

Description

This is function _bt_avrcp_allocate_browsing_response.

_bt_avrcp_general_reject Function

File

avrcp_private.h

C

```
bt_bool _bt_avrcp_general_reject(bt_avrcp_channel_t* channel, bt_byte reason, bt_byte tran_id);
```

Description

This is function _bt_avrcp_general_reject.

_bt_avrcp_send_rejected_response Function

File

avrcp_private.h

C

```
void _bt_avrcp_send_rejected_response(bt_avrcp_channel_t* channel, bt_byte tran_id, bt_byte pdu_id);
```

Description

This is function _bt_avrcp_send_rejected_response.

bt_avrcp_send_release_panel_control Function

File

avrcp_command.h

C

```
bt_bool bt_avrcp_send_release_panel_control(bt_avrcp_channel_t* channel, bt_byte opid);
```

Description

brief Send AV/C Panel Subunit "released" PASS THROUGH command ingroup avrcp

details This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC_PANEL_BUTTON_RELEASED.

param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants

return li c TRUE if the function succeeds. li c FALSE otherwise.

AVRCP Data Types and Constants

_bt_av_command_t Structure

File

avrcp_command.h

C

```
struct _bt_av_command_t {
  bt_av_command_t* next_command;
  bt_byte cmd_type;
  bt_byte flags;
  bt_byte ctype;
  bt_byte subunit_type;
  bt_byte subunit_id;
  bt_byte opcode;
  bt_byte pdu_id;
  bt_byte* parameters;
  bt_uint params_len;
  bt_av_response_t* response_buffer;
  bt_long send_time;
  bt_byte tran_id;
};
```

Description

This is record _bt_av_command_t.

_bt_avrcp_channel_t Structure

File

avrcp.h

C

```
struct _bt_avrcp_channel_t {
  bt_byte state;
  bt_byte flags;
  bt_uint cur_event_mask;
  bt_uint new_event_mask;
  bt_byte play_status;
  bt_uint req_play_status_interval;
  bt_ulong play_status_req_time;
  bt_avctp_channel_t* control_channel;
  bt_avctp_channel_t* browsing_channel;
  bt_avrcp_mgr_t* avrcp_mgr;
  bt_queue_element_t* send_cq_head;
  bt_queue_element_t* ack_cq_head;
  bt_uint tg_event_mask;
  bt_uint tg_interim_event_mask;
  bt_uint tg_signalled_event_mask;
  bt_ulong notify_song_position;
  bt_byte register_notification_tran_id[AVC_MAXEVENTS];
  bt_byte tg_next_event_id;
  bt_byte volume;
};
```

Members

Members	Description
bt_uint tg_event_mask;	target data

Description

brief AVRCP channel description ingroup avrcp

details This structure is used to hold information about an AVRCP channel.

_bt_avrcp_mgr_t Structure

File

avrcp.h

C

```
struct _bt_avrcp_mgr_t {
  bt_byte state;
  bt_byte flags;
  bt_ulong company_id;
  bt_uint supported_events;
  bt_ulong song_length;
  bt_ulong song_position;
  bt_byte play_status;
  bt_byte volume;
  bt_byte battery_status;
  bt_byte system_status;
  bt_av_element_id_t current_track_id;
  bt_uint uid_counter;
  bt_avrcp_channel_t* channels;
  bt_avrcp_mgr_callback_fp callback;
  void* callback_param;
};
```

Members

Members	Description
bt_byte state;	< Manager state. This value can be one of the following values: < li AVRCP_MANAGER_STATE_IDLE < li AVRCP_MANAGER_STATE_CONNECTING < li AVRCP_MANAGER_STATE_DISCONNECTING

bt_byte flags;	< Additional manager state. This value can be a combination of the following values: < li AVRCP_MANAGER_FLAG_SEARCHING
bt_ulong company_id;	< The 24-bit unique ID obtained from the IEEE Registration Authority Committee. < If the vendor of a TG device does not have the unique ID, the value 0xFFFFFF may be used.
bt_uint supported_events;	< Events supported by the target. This value can be a combination of the following values: < li AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED < li AVC_EVENT_FLAG_TRACK_CHANGED < li AVC_EVENT_FLAG_TRACK_REACHED_END < li AVC_EVENT_FLAG_TRACK_REACHED_START < li AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED < li AVC_EVENT_FLAG_BATT_STATUS_CHANGED < li AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED < li AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED < li AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED < li AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED < li AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED < li AVC_EVENT_FLAG_UIDS_CHANGED < li AVC_EVENT_FLAG_VOLUME_CHANGED
bt_ulong song_length;	< Current song length
bt_ulong song_position;	< Current song poistion
bt_byte play_status;	< Current playback status
bt_byte volume;	< Current volumne
bt_byte battery_status;	< Current battery statusThis can be one of the following values: < li AVC_BATTERY_STATUS_NORMAL < li AVC_BATTERY_STATUS_WARNING < li AVC_BATTERY_STATUS_CRITICAL < li AVC_BATTERY_STATUS_EXTERNAL < li AVC_BATTERY_STATUS_FULL_CHARGE
bt_byte system_status;	< Current system status
bt_av_element_id_t current_track_id;	< Current track id
bt_uint uid_counter;	< The number of media items in the target.
bt_avrcp_channel_t* channels;	< List of available AVRCP channels.
bt_avrcp_mgr_callback_fp callback;	< Pointer to a function used to notify the AVRCP consumer about various events.
void* callback_param;	< Pointer to arbitrary data to be passed to the c callback.

Description

brief AVRCP manager. ingroup avrcp

details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avrcp_get_mgr().

bt_av_add_to_now_playing_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_add_to_now_playing_s {
  bt_av_response_t header;
  bt_byte status;
} bt_av_add_to_now_playing_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte status;	< The result of the request.

Description

brief Parameter to AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::add_to_now_playing_status - when a local device received a response to a "add to now playing" request.

bt_av_battery_status_of_ct_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_battery_status_of_ct_s {
  bt_av_response_t header;
  bt_byte status;
} bt_av_battery_status_of_ct_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte status;	< Battery status. This can be one of the following values: < li AVC_BATTERY_STATUS_NORMAL < li AVC_BATTERY_STATUS_WARNING < li AVC_BATTERY_STATUS_CRITICAL < li AVC_BATTERY_STATUS_EXTERNAL < li AVC_BATTERY_STATUS_FULL_CHARGE

Description

brief Parameter to AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::battery_status_of_ct - when a local device received a "battery status of controller" command.

bt_av_capability_company_id_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_capability_company_id_s {
  bt_av_response_t header;
  bt_byte count;
  bt_ulong* company_id_list;
} bt_av_capability_company_id_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of supported company ids.
bt_ulong* company_id_list;	< List of supported company ids.

Description

brief Parameter to AVRCP_EVT_COMPANY_ID_LIST_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::company_id - when a local device received a response to a "get company id" request.

bt_av_capability_event_id_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_capability_event_id_s {
  bt_av_response_t header;
  bt_byte count;
  bt_byte* event_id_list;
} bt_av_capability_event_id_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of supported events ids.
bt_byte* event_id_list;	< List of supported event ids.

Description

brief Parameter to AVRCP_EVT_EVENT_ID_LIST_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::supported_event_id - when a local device received a response to a "get supported events" request.

bt_av_command_t Type

File

avrcp_command.h

C

```
typedef struct _bt_av_command_t bt_av_command_t;
```

Description

This is type bt_av_command_t.

bt_av_displayable_character_set_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_displayable_character_set_s {
  bt_av_response_t header;
  bt_byte count;
  bt_uint* charset_list;
} bt_av_displayable_character_set_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of supported characters sets by the controller.
bt_uint* charset_list;	< List of supported characters sets by the controller.

Description

brief Parameter to AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::displayable_character_set - when a local device received a "displayable chracter set command" request.

bt_av_element_attribute_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_element_attribute_s {
  bt_ulong id;
  bt_uint charset;
  bt_uint len;
  bt_byte* value;
} bt_av_element_attribute_t;
```

Members

Members	Description
bt_ulong id;	< Attribute Id.
bt_uint charset;	< Charcater set.
bt_uint len;	< Value length.
bt_byte* value;	< Attribute value.

Description

brief Media element attribute ingroup avrcp

details This structure is used to store media element attribute.

bt_av_element_attributes_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_element_attributes_s {
  bt_av_response_t header;
  bt_byte count;
  bt_av_element_attribute_t* attr_list;
} bt_av_element_attributes_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of attributes returned.
bt_av_element_attribute_t* attr_list;	< List of attribute values

Description

brief Parameter to AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::element_attributes - when a local device received a response to a "get media element attributes" request.

bt_av_element_id_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_element_id_s {
  bt_ulong id_lo;
  bt_ulong id_hi;
} bt_av_element_id_t;
```

Members

Members	Description
bt_ulong id_lo;	< 4 least significant bytes of UID.
bt_ulong id_hi;	< 4 most significant bytes of UID.

Description

brief Media element UID ingroup avrcp

details This structure is used to store media element UID.

bt_av_get_element_attributes_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_get_element_attributes_s {
  bt_av_response_t header;
  bt_av_element_id_t id;
  bt_byte attributes;
} bt_av_get_element_attributes_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_av_element_id_t id;	< Media element UID
bt_byte attributes;	< Bitmask that defines attributes requested. This can be a combination of the following values: < li AVC_MEDIA_ATTR_FLAG_TITLE < li AVC_MEDIA_ATTR_FLAG_ARTIST < li AVC_MEDIA_ATTR_FLAG_ALBUM < li AVC_MEDIA_ATTR_FLAG_NUMBER < li AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER < li AVC_MEDIA_ATTR_FLAG_GENRE < li AVC_MEDIA_ATTR_FLAG_PLAYING_TIME

Description

brief Parameter to AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::get_element_attributes - when a local device received a "get element attributes" request.

bt_av_notification_addressed_player_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_addressed_player_changed_s {
  bt_uint player_id;
  bt_uint uid_counter;
} bt_av_notification_addressed_player_changed_t;
```

Members

Members	Description
bt_uint player_id;	< New player Id
bt_uint uid_counter;	< UID counter

Description

brief Parameter to AVRCP_EVT_ADDRESSED_PLAYER_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:addressed_player - when a local device received a "addressed player changed" notification.

bt_av_notification_app_setting_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_app_setting_changed_s {
  bt_byte setting_id;
  bt_byte setting_value_id;
} bt_av_notification_app_setting_changed_t;
```

Description

This is type bt_av_notification_app_setting_changed_t.

bt_av_notification_battery_status_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_battery_status_s {
  bt_byte status;
} bt_av_notification_battery_status_t;
```

Members

Members	Description
bt_byte status;	< New battery status.This can be one of the following values: < li AVC_BATTERY_STATUS_NORMAL < li AVC_BATTERY_STATUS_WARNING < li AVC_BATTERY_STATUS_CRITICAL < li AVC_BATTERY_STATUS_EXTERNAL < li AVC_BATTERY_STATUS_FULL_CHARGE

Description

brief Parameter to AVRCP_EVT_BATT_STATUS_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:battery_status - when a local device received a "battery status changed" notification.

bt_av_notification_playback_pos_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_playback_pos_changed_s {
  bt_ulong position;
} bt_av_notification_playback_pos_changed_t;
```

Members

Members	Description
bt_ulong position;	< New playback position

Description

brief Parameter to AVRCP_EVT_PLAYBACK_POS_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:playback_pos - when a local device received a "playback position changed" notification.

bt_av_notification_playback_status_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_playback_status_changed_s {
  bt_av_response_t header;
  bt_byte status;
} bt_av_notification_playback_status_changed_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte status;	< Play status. This can be on of the following values: < li AVC_PLAY_STATUS_STOPPED < li AVC_PLAY_STATUS_PLAYING < li AVC_PLAY_STATUS_PAUSED < li AVC_PLAY_STATUS_FW_SEEK < li AVC_PLAY_STATUS_REV_SEEK < li AVC_PLAY_STATUS_ERROR

Description

brief Parameter to AVRCP_EVT_PLAYBACK_STATUS_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:play_status - when a local device received a "play status changed" notification.

bt_av_notification_system_status_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_system_status_changed_s {
  bt_byte status;
} bt_av_notification_system_status_changed_t;
```

Members

Members	Description
bt_byte status;	< New system status

Description

brief Parameter to AVRCP_EVT_SYSTEM_STATUS_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:system_status - when a local device received a "system status changed" notification.

bt_av_notification_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_s {
  bt_av_response_t header;
  union {
    bt_av_notification_playback_status_changed_t play_status;
    bt_av_notification_track_changed_t track;
    bt_av_notification_playback_pos_changed_t playback_pos;
    bt_av_notification_battery_status_t battery_status;
    bt_av_notification_system_status_changed_t system_status;
    bt_av_notification_addressed_player_changed_t addressed_player;
    bt_av_notification_uids_changed_t uids;
    bt_av_notification_volume_changed_t volume;
    bt_av_notification_app_setting_changed_t app_setting;
  } params;
} bt_av_notification_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_av_notification_playback_status_changed_t play_status;	< Valid if notification is AVRCP_EVT_PLAYBACK_STATUS_CHANGED
bt_av_notification_track_changed_t track;	< Valid if notification is AVRCP_EVT_TRACK_CHANGED
bt_av_notification_playback_pos_changed_t playback_pos;	< Valid if notification is AVRCP_EVT_PLAYBACK_POS_CHANGED
bt_av_notification_battery_status_t battery_status;	< Valid if notification is AVRCP_EVT_BATT_STATUS_CHANGED
bt_av_notification_system_status_changed_t system_status;	< Valid if notification is AVRCP_EVT_SYSTEM_STATUS_CHANGED
bt_av_notification_addressed_player_changed_t addressed_player;	< Valid if notification is AVRCP_EVT_ADDRESSED_PLAYER_CHANGED
bt_av_notification_uids_changed_t uids;	< Valid if notification is AVRCP_EVT_UIDS_CHANGED
bt_av_notification_volume_changed_t volume;	< Valid if notification is AVRCP_EVT_VOLUME_CHANGED
bt_av_notification_app_setting_changed_t app_setting;	< Valid if notification is AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED

Description

brief Parameter to the following events: li AVRCP_EVT_PLAYBACK_STATUS_CHANGED li AVRCP_EVT_TRACK_CHANGED li AVRCP_EVT_PLAYBACK_POS_CHANGED li AVRCP_EVT_BATT_STATUS_CHANGED li AVRCP_EVT_SYSTEM_STATUS_CHANGED li AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED li AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED li AVRCP_EVT_ADDRESSED_PLAYER_CHANGED li AVRCP_EVT_UIDS_CHANGED li AVRCP_EVT_VOLUME_CHANGED ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification - when a local device received one of the following notifications from the target: li Play status changed li Track changed changed li Playback position changed li Battery status changed li System status changed li Addressed player changed li UIDs changed li Volume changed li Player application setting changed The notification code defines which member of the bt_av_notification_t::params union is valid @arg AVRCP_EVT_PLAYBACK_STATUS_CHANGED c bt_av_notification_playback_status_changed_t play_status @arg

AVRCP_EVT_TRACK_CHANGED c bt_av_notification_track_changed_t track @arg AVRCP_EVT_PLAYBACK_POS_CHANGED c bt_av_notification_playback_pos_changed_t playback_pos @arg AVRCP_EVT_BATT_STATUS_CHANGED c bt_av_notification_battery_status_t battery_status @arg AVRCP_EVT_SYSTEM_STATUS_CHANGED c bt_av_notification_system_status_changed_t system_status @arg AVRCP_EVT_ADDRESSED_PLAYER_CHANGED c bt_av_notification_addressed_player_changed_t addressed_player @arg AVRCP_EVT_UIDS_CHANGED c bt_av_notification_uids_changed_t uids @arg AVRCP_EVT_VOLUME_CHANGED c bt_av_notification_volume_changed_t volume @arg AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED c bt_av_notification_app_setting_changed_t app_setting

bt_av_notification_track_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_track_changed_s {
  bt_av_element_id_t id;
} bt_av_notification_track_changed_t;
```

Members

Members	Description
bt_av_element_id_t id;	< New track UID

Description

brief Parameter to AVRCP_EVT_TRACK_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:track - when a local device received a "track changed" notification.

bt_av_notification_uids_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_uids_changed_s {
  bt_uint uid_counter;
} bt_av_notification_uids_changed_t;
```

Members

Members	Description
bt_uint uid_counter;	< UID counter

Description

brief Parameter to AVRCP_EVT_UIDS_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:uids - when a local device received a "UIDs changed" notification.

bt_av_notification_volume_changed_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_notification_volume_changed_s {
  bt_byte volume;
} bt_av_notification_volume_changed_t;
```

Members

Members	Description
bt_byte volume;	< Volume

Description

brief Parameter to AVRCP_EVT_VOLUME_CHANGED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:volume - when a local device received a "UIDs changed" notification.

bt_av_play_item_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_play_item_s {
  bt_av_response_t header;
  bt_byte status;
} bt_av_play_item_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte status;	< The result of the request.

Description

brief Parameter to AVRCP_EVT_PLAY_ITEM_COMPLETED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_item_status - when a local device received a response to a "play item" request.

bt_av_play_status_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_play_status_s {
  bt_av_response_t header;
  bt_ulong song_length;
  bt_ulong song_position;
  bt_byte play_status;
} bt_av_play_status_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_ulong song_length;	< Current track length
bt_ulong song_position;	< Current track position
bt_byte play_status;	< Playback status. This can be one of the following values: < li AVC_PLAY_STATUS_STOPPED < li AVC_PLAY_STATUS_PLAYING < li AVC_PLAY_STATUS_PAUSED < li AVC_PLAY_STATUS_FW_SEEK < li AVC_PLAY_STATUS_REV_SEEK < li AVC_PLAY_STATUS_ERROR

Description

brief Parameter to AVRCP_EVT_GET_PLAY_STATUS_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_status - when a local device received a response to a "get play status" request.

bt_av_player_setting_current_values_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_player_setting_current_values_s {
  bt_av_response_t header;
  bt_byte count;
  bt_byte* setting_id_list;
  bt_byte* setting_value_id_list;
} bt_av_player_setting_current_values_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of player setting attribute ids to be returned from the target.
bt_byte* setting_id_list;	< List of player setting attribute ids to be returned from the target.
bt_byte* setting_value_id_list;	< List of current player setting attribute value ids.

Description

brief Parameter to AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_current_values - when a local device received a response to a "get current player setting attribute values" request.

bt_av_player_setting_values_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_player_setting_values_s {
  bt_av_response_t header;
  bt_byte count;
  bt_byte* setting_value_id_list;
} bt_av_player_setting_values_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of supported player setting attribute value ids.
bt_byte* setting_value_id_list;	< List of supported player setting attribute value ids.

Description

brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values - when a local device received a response to a "get player setting attribute values" request.

bt_av_player_setting_values_text_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_player_setting_values_text_s {
  bt_av_response_t header;
  bt_byte count;
  bt_byte* setting_value_id_list;
  bt_av_player_text_t* setting_value_text_list;
} bt_av_player_setting_values_text_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.

bt_byte count;	< The number of player setting attribute value ids for which displayable text is requested.
bt_byte* setting_value_id_list;	< List of player setting attribute value ids for which displayable text is requested.
bt_av_player_text_t* setting_value_text_list;	< List of player setting attribute values displayable text.

Description

brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values_text - when a local device received a response to a "get player setting attribute values displayable text" request.

bt_av_player_settings_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_player_settings_s {
  bt_av_response_t header;
  bt_byte count;
  bt_byte* setting_id_list;
} bt_av_player_settings_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of supported player setting attribute ids.
bt_byte* setting_id_list;	< List of supported player setting attribute ids.

Description

brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings - when a local device received a response to a "get supported player setting attributes" request.

bt_av_player_settings_text_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_player_settings_text_s {
  bt_av_response_t header;
  bt_byte count;
  bt_byte* setting_id_list;
  bt_av_player_text_t* setting_text_list;
} bt_av_player_settings_text_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte count;	< The number of player setting attribute ids for which displayable text is requested.
bt_byte* setting_id_list;	< List of player setting attribute ids for which displayable text is requested.
bt_av_player_text_t* setting_text_list;	< List of player setting attributes displayable text.

Description

brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings_text - when a local device received a response to a "get player setting attributes displayable text" request.

bt_av_player_text_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_player_text_s {
  bt_uint charset;
  bt_byte len;
  bt_byte* text;
} bt_av_player_text_t;
```

Description

This is type bt_av_player_text_t.

bt_av_register_notification_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_register_notification_t {
  bt_av_response_t header;
  bt_byte event_id;
  bt_ulong playback_pos;
} bt_av_register_notification_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte event_id;	< Event Id. This can be one of the following values: < li AVC_EVENT_PLAYBACK_STATUS_CHANGED < li AVC_EVENT_TRACK_CHANGED < li AVC_EVENT_TRACK_REACHED_END < li AVC_EVENT_TRACK_REACHED_START < li AVC_EVENT_PLAYBACK_POS_CHANGED < li AVC_EVENT_BATT_STATUS_CHANGED < li AVC_EVENT_SYSTEM_STATUS_CHANGED < li AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED < li AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED < li AVC_EVENT_AVAILABLE_PLAYERS_CHANGED < li AVC_EVENT_ADDRESSED_PLAYER_CHANGED < li AVC_EVENT_UIDS_CHANGED < li AVC_EVENT_VOLUME_CHANGED
bt_ulong playback_pos;	< Playback position. Used only if c event_id is AVC_EVENT_PLAYBACK_POS_CHANGED

Description

brief Parameter to AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::register_notification - when a local device received a "register notification" request.

bt_av_response_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_response_t {
  bt_avrcp_channel_t* channel;
  bt_byte ctype;
  bt_byte tran_id;
} bt_av_response_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel

bt_byte ctype;	< Response type. This can be one of the following values: < li AVC_RESPONSE_NOT_IMPLEMENTED < li AVC_RESPONSE_ACCEPTED < li AVC_RESPONSE_REJECTED < li AVC_RESPONSE_IN_TRANSITION < li AVC_RESPONSE_STABLE < li AVC_RESPONSE_IMPLEMENTED < li AVC_RESPONSE_CHANGED < li AVC_RESPONSE_INTERIM < li AVC_RESPONSE_TIMEOUT
bt_byte tran_id;	< Transaction Id.

Description

brief AV/C response header ingroup avrcp

details This structure is used to store fields present in every AV/C response.

bt_av_set_absolute_volume_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_set_absolute_volume_s {
  bt_av_response_t header;
  bt_byte volume;
} bt_av_set_absolute_volume_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte volume;	< Volume

Description

brief Parameter to AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::absolute_volume - when a local device received a response to a "set absolute volume" request.

bt_av_set_addressed_player_t Structure

File

avrcp_command.h

C

```
typedef struct _bt_av_set_addressed_player_s {
  bt_av_response_t header;
  bt_byte status;
} bt_av_set_addressed_player_t;
```

Members

Members	Description
bt_av_response_t header;	< Common response header.
bt_byte status;	< The result of changing the addressed player.

Description

brief Parameter to AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::addressed_player - when a local device received a response to a "set addressed player" request.

bt_avrcp_channel_t Type

File

avrcp.h

C

```
typedef struct _bt_avrcp_channel_t bt_avrcp_channel_t;
```

Description

This is type bt_avrcp_channel_t.

bt_avrcp_device_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_device_s {
  bt_bdaddr_t address;
  bt_byte* name;
  bt_byte pg_scan_rpt_mode;
  bt_byte pg_scan_period_mode;
  bt_long cod;
  bt_int clock_offset;
  bt_byte rssi;
} bt_avrcp_device_t;
```

Description

This is type bt_avrcp_device_t.

bt_avrcp_event_t Union

File

avrcp.h

C

```
typedef union _bt_avrcp_event_u {
  bt_avrcp_evt_channel_connected_t channel_connected;
  bt_avrcp_evt_channel_disconnected_t channel_disconnected;
  bt_avrcp_evt_connection_failed_t connection_failed;
  bt_avrcp_evt_panel_response_received_t panel_response_received;
  bt_avrcp_evt_search_completed_t device_search;
  bt_av_response_t response;
  bt_av_capability_company_id_t company_id;
  bt_av_capability_event_id_t supported_event_id;
  bt_av_player_settings_t player_settings;
  bt_av_player_setting_values_t player_setting_values;
  bt_av_player_setting_current_values_t player_setting_current_values;
  bt_av_player_settings_text_t player_settings_text;
  bt_av_player_setting_values_text_t player_setting_values_text;
  bt_av_element_attributes_t element_attributes;
  bt_av_play_status_t play_status;
  bt_av_set_absolute_volume_t absolute_volume;
  bt_av_set_addressed_player_t addressed_player;
  bt_av_play_item_t play_item_status;
  bt_av_add_to_now_playing_t add_to_now_playing_status;
  bt_av_notification_t notification;
  bt_avrcp_evt_register_events_completed_t register_events;
  bt_avrcp_evt_panel_command_received_t panel_command_received;
  bt_av_battery_status_of_ct_t battery_status_of_ct;
  bt_av_displayable_character_set_t displayable_character_set;
  bt_av_get_element_attributes_t get_element_attributes;
  bt_av_register_notification_t register_notification;
} bt_avrcp_event_t;
```

Members

Members	Description
bt_avrcp_evt_channel_connected_t channel_connected;	< Valid if event is AVRCP_EVT_CONTROL_CHANNEL_CONNECTED
bt_avrcp_evt_channel_disconnected_t channel_disconnected;	< Valid if event is AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED

bt_avrcp_evt_connection_failed_t connection_failed;	< Valid if event is AVRCP_EVT_CONTROL_CONNECTION_FAILED
bt_avrcp_evt_panel_response_received_t panel_response_received;	< Valid if event is AVRCP_EVT_PANEL_RESPONSE_RECEIVED
bt_avrcp_evt_search_completed_t device_search;	< Valid if event is AVRCP_EVT_SEARCH_COMPLETED
bt_av_capability_company_id_t company_id;	< Valid if event is AVRCP_EVT_COMPANY_ID_LIST_RECEIVED
bt_av_capability_event_id_t supported_event_id;	< Valid if event is AVRCP_EVT_EVENT_ID_LIST_RECEIVED
bt_av_player_settings_t player_settings;	< Valid if event is AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED
bt_av_player_setting_values_t player_setting_values;	< Valid if event is AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED
bt_av_player_setting_current_values_t player_setting_current_values;	< Valid if event is AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED
bt_av_player_settings_text_t player_settings_text;	< Valid if event is AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED
bt_av_player_setting_values_text_t player_setting_values_text;	< Valid if event is AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED
bt_av_element_attributes_t element_attributes;	< Valid if event is AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED
bt_av_play_status_t play_status;	< Valid if event is AVRCP_EVT_GET_PLAY_STATUS_RECEIVED
bt_av_set_absolute_volume_t absolute_volume;	< Valid if event is AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED
bt_av_set_addressed_player_t addressed_player;	< Valid if event is AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED
bt_av_play_item_t play_item_status;	< Valid if event is AVRCP_EVT_PLAY_ITEM_COMPLETED
bt_av_add_to_now_playing_t add_to_now_playing_status;	< Valid if event is AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED
bt_av_notification_t notification;	< Valid if event is on of the following: < li AVRCP_EVT_PLAYBACK_STATUS_CHANGED < li AVRCP_EVT_TRACK_CHANGED < li AVRCP_EVT_TRACK_REACHED_END < li AVRCP_EVT_TRACK_REACHED_START < li AVRCP_EVT_PLAYBACK_POS_CHANGED < li AVRCP_EVT_BATT_STATUS_CHANGED < li AVRCP_EVT_SYSTEM_STATUS_CHANGED < li AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED < li AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED < li AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED < li AVRCP_EVT_ADDRESSED_PLAYER_CHANGED < li AVRCP_EVT_UIDS_CHANGED < li AVRCP_EVT_VOLUME_CHANGED
bt_avrcp_evt_register_events_completed_t register_events;	< Valid if event AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED
bt_avrcp_evt_panel_command_received_t panel_command_received;	< Valid if event is AVRCP_EVT_PANEL_COMMAND_RECEIVED
bt_av_battery_status_of_ct_t battery_status_of_ct;	< Valid if event is AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED
bt_av_displayable_character_set_t displayable_character_set;	< Valid if event is AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED
bt_av_get_element_attributes_t get_element_attributes;	< Valid if event is AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED
bt_av_register_notification_t register_notification;	< Valid if event is AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED

Description

brief Parameter to an application callback. ingroup avrcp

details This union is used to pass event specific data to the AVRCP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

bt_avrcp_evt_channel_connected_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_channel_connected_t {
  bt_avrcp_channel_t* channel;
} bt_avrcp_evt_channel_connected_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel

Description

brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_CONNECTED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_connected - when a control channel between two devices has been established.

bt_avrcp_evt_channel_disconnected_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_channel_disconnected_t {
  bt_avrcp_channel_t* channel;
} bt_avrcp_evt_channel_disconnected_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel

Description

brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_disconnected - when a control channel between two devices has been terminated.

bt_avrcp_evt_connection_failed_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_connection_failed_t {
  bt_avrcp_channel_t* channel;
} bt_avrcp_evt_connection_failed_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel

Description

brief Parameter to AVRCP_EVT_CONTROL_CONNECTION_FAILED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::connection_failed - when a local device failed to create a control channel between two AVRCP entities.

bt_avrcp_evt_panel_command_received_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_panel_command_received_t {
  bt_avrcp_channel_t* channel;
  bt_byte ctype;
  bt_byte button_status;
  bt_byte opcode;
  bt_byte* params;
  bt_byte params_len;
} bt_avrcp_evt_panel_command_received_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel
bt_byte ctype;	< Command type
bt_byte button_status;	< Button status
bt_byte opcode;	< Operation Id
bt_byte* params;	< Operation parameters
bt_byte params_len;	< Length of the operation parameters

Description

brief Parameter to AVRCP_EVT_PANEL_COMMAND_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_command_received - when a local device received a PASS THROUGH command.

bt_avrcp_evt_panel_response_received_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_panel_response_received_t {
  bt_avrcp_channel_t* channel;
  bt_byte ctype;
  bt_byte button_status;
  bt_byte opcode;
  bt_byte* params;
  bt_byte params_len;
} bt_avrcp_evt_panel_response_received_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel
bt_byte ctype;	< Response type
bt_byte button_status;	< Button status
bt_byte opcode;	< Operation Id
bt_byte* params;	< Operation parameters
bt_byte params_len;	< Length of the operation parameters

Description

brief Parameter to AVRCP_EVT_PANEL_RESPONSE_RECEIVED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_response_received - when a local device received a response to a PASS THROUGH command.

bt_avrcp_evt_register_events_completed_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_register_events_completed_t {
  bt_avrcp_channel_t* channel;
} bt_avrcp_evt_register_events_completed_t;
```

Members

Members	Description
bt_avrcp_channel_t* channel;	< AVRCP channel

Description

This is type bt_avrcp_evt_register_events_completed_t.

bt_avrcp_evt_search_completed_t Structure

File

avrcp.h

C

```
typedef struct _bt_avrcp_evt_search_completed_s {
  bt_avrcp_device_t* devices;
  bt_byte count;
} bt_avrcp_evt_search_completed_t;
```

Members

Members	Description
bt_avrcp_device_t* devices;	< list of found devices
bt_byte count;	< the number of found devices

Description

brief Parameter to AVRCP_EVT_SEARCH_COMPLETED event ingroup avrcp

details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::device_search - when searching for nearby devices has finished.

bt_avrcp_find_callback_fp Type

File

avrcp.h

C

```
typedef void (* bt_avrcp_find_callback_fp)(bt_bool found, void* param);
```

Description

brief Find Controller/Target callback. ingroup avrcp

details This callback is called when search for Controller/Target has finished.

param found This can be one of the following values: li c TRUE if Controller/Target was found. li c FALSE otherwise. param callback_param A pointer to an arbitrary data set by a call to bt_avrcp_find_target/bt_avrcp_find_controller.

bt_avrcp_mgr_callback_fp Type

File

avrcp.h

C

```
typedef void (* bt_avrcp_mgr_callback_fp)(bt_avrcp_mgr_t* mgr, bt_byte evt, bt_avrcp_event_t* evt_param,
void* callback_param);
```

Description

brief AVRCP application callback. ingroup avrcp

details In order to be notified of various events a consumer of the AVRCP layer has to register a callback function (done with bt_avrcp_start()). The stack will call that function whenever a new event has been generated.

param mgr AVRCP manager.

param evt AVRCP event. The event can be one of the following values: @arg AVRCP_EVT_CONTROL_CHANNEL_CONNECTED This event is generated when a control channel between two AVRCP entities has been established. @arg AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED This event is generated when a control channel between two AVRCP entities has been terminated. @arg AVRCP_EVT_CONTROL_CONNECTION_FAILED This event is generated when a local device failed to create a control channel between two AVRCP entities. @arg AVRCP_EVT_BROWSING_CHANNEL_CONNECTED This event is generated when a browsing channel between two AVRCP entities has been established. @arg AVRCP_EVT_BROWSING_CHANNEL_DISCONNECTED This event is generated when a browsing channel between two AVRCP entities has been terminated. @arg AVRCP_EVT_BROWSING_CONNECTION_FAILED This event is generated when a local device failed to create a browsing channel between two AVRCP entities. @arg AVRCP_EVT_SEARCH_COMPLETED This event is generated when a local device completed searching for nearby targets.

@arg AVRCP_EVT_PANEL_RESPONSE_RECEIVED This event is generated when a local device received a response to a PASS THROUGH command. @arg AVRCP_EVT_COMPANY_ID_LIST_RECEIVED This event is generated when a local device received a response to a "get company id" request. @arg AVRCP_EVT_EVENT_ID_LIST_RECEIVED This event is generated when a local device received a response to a "get supported events" request. @arg AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED This event is generated when a local device received a response to a "get supported player setting attributes" request. @arg AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED This event is generated when a local device received a response to a "get player setting attribute values" request. @arg AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED This event is generated when a local device received a response to a "get current player setting attribute values" request. @arg AVRCP_EVT_SET_PLAYER_SETTING_VALUES_COMPLETED This event is generated when a local device received a response to a "set player setting attribute values" request. @arg AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED This event is generated when a local device received a response to a "get player setting attributes displayable text" request. @arg AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED This event is generated when a local device received a response to a "get player setting attribute values displayable text" request. @arg AVRCP_EVT_INFORM_DISPLAYABLE_CHARACTER_SET_COMPLETED This event is generated when a local device received a response to a "inform displayable character set" request. @arg AVRCP_EVT_INFORM_BATTERY_STATUS_OF_CT_COMPLETED This event is generated when a local device received a response to a "inform battery status" request. @arg AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED This event is generated when a local device received a response to a "get media element attributes" request. @arg AVRCP_EVT_GET_PLAY_STATUS_RECEIVED This event is generated when a local device received a response to a "get play status" request. @arg AVRCP_EVT_PLAYBACK_STATUS_CHANGED This event is generated when a local device received a "play status changed" notification. @arg AVRCP_EVT_TRACK_CHANGED This event is generated when a local device received a "track changed changed" notification. @arg AVRCP_EVT_TRACK_REACHED_END This event is generated when a local device received a "track reached end" notification. @arg AVRCP_EVT_TRACK_REACHED_START This event is generated when a local device received a "track reached start" notification. @arg AVRCP_EVT_PLAYBACK_POS_CHANGED This event is generated when a local device received a "playback position changed" notification. @arg AVRCP_EVT_BATT_STATUS_CHANGED This event is generated when a local device received a "battery status changed" notification. @arg AVRCP_EVT_SYSTEM_STATUS_CHANGED This event is generated when a local device received a "system status changed" notification. @arg AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED This event is generated when a local device received a "player application setting changed" notification. @arg AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED This event is generated when a local device received a "now playing content changed" notification. @arg AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED This event is generated when a local device received a "available players changed" notification. @arg AVRCP_EVT_ADDRESSED_PLAYER_CHANGED This event is generated when a local device received a "addressed player changed" notification. @arg AVRCP_EVT_UIDS_CHANGED This event is generated when a local device received a "UIDs changed" notification. @arg AVRCP_EVT_VOLUME_CHANGED This event is generated when a local device received a "volume changed" notification. @arg AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED This event is generated when a local device received a response to a "set absolute volume" request. @arg AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED This event is generated when a local device received a response to a "set addressed player" request. @arg AVRCP_EVT_PLAY_ITEM_COMPLETED This event is generated when a local device received a response to a "play item" request. @arg AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED This event is generated when a local device received a response to a "add to now playing" request. @arg AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED This event is generated when a local device received a response to a "register notification" request.

@arg AVRCP_EVT_PANEL_COMMAND_RECEIVED This event is generated when a local device received a PASS THROUGH command. @arg AVRCP_EVT_SET_ABSOLUTE_VOLUME_REQUESTED This event is generated when a local device received a "set absolute volume" request. @arg AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED This event is generated when a local device received a "battery status of controller" command. @arg AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED This event is generated when a local device received a "displayable chracter set command" request. @arg AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED This event is generated when a local device received a "get element attributes" request.

param evt_param Event parameter. Which member of the bt_avrcp_event_t union is valid depends on the event: @arg AVRCP_EVT_CONTROL_CHANNEL_CONNECTED c bt_avrcp_evt_channel_connected_t channel_connected @arg AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED c bt_avrcp_evt_channel_disconnected_t channel_disconnected @arg AVRCP_EVT_CONTROL_CONNECTION_FAILED c bt_avrcp_evt_connection_failed_t connection_failed @arg AVRCP_EVT_PANEL_RESPONSE_RECEIVED c bt_avrcp_evt_panel_response_received_t panel_response_received @arg AVRCP_EVT_SEARCH_COMPLETED c bt_avrcp_evt_search_completed_t device_search @arg AVRCP_EVT_COMPANY_ID_LIST_RECEIVED c bt_av_capability_company_id_t company_id @arg AVRCP_EVT_EVENT_ID_LIST_RECEIVED c bt_av_capability_event_id_t supported_event_id @arg AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED c bt_av_player_settings_t player_settings @arg AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED c bt_av_player_setting_values_t player_setting_values @arg AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED c bt_av_player_setting_current_values_t player_setting_current_values @arg AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED c bt_av_player_settings_text_t player_settings_text @arg AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED c bt_av_player_setting_values_text_t player_setting_values_text @arg AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED c bt_av_element_attributes_t element_attributes @arg AVRCP_EVT_GET_PLAY_STATUS_RECEIVED c bt_av_play_status_t play_status @arg AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED c bt_av_set_absolute_volume_t absolute_volume @arg AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED c bt_av_set_addressed_player_t addressed_player @arg AVRCP_EVT_PLAY_ITEM_COMPLETED c bt_av_play_item_t play_item_status @arg AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED c bt_av_add_to_now_playing_t add_to_now_playing_status @arg AVRCP_EVT_PLAYBACK_STATUS_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_TRACK_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_TRACK_REACHED_END c bt_av_notification_t notification @arg AVRCP_EVT_TRACK_REACHED_START c bt_av_notification_t notification @arg AVRCP_EVT_PLAYBACK_POS_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_BATT_STATUS_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_SYSTEM_STATUS_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_ADDRESSED_PLAYER_CHANGED c

bt_av_notification_t notification @arg AVRCP_EVT_UIDS_CHANGED c bt_av_notification_t notification @arg AVRCP_EVT_VOLUME_CHANGED c bt_av_notification_t notification

@arg AVRCP_EVT_PANEL_COMMAND_RECEIVED c bt_avrcp_evt_panel_command_received_t panel_command_received @arg AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED c bt_av_battery_status_of_ct_t battery_status_of_ct @arg AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED c bt_av_displayable_character_set_t displayable_character_set @arg AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED c bt_av_get_element_attributes_t get_element_attributes

param callback_param A pointer to an arbitrary data set by a call to bt_avrcp_start.

bt_avrcp_mgr_t Type

File

avrcp.h

C

```
typedef struct _bt_avrcp_mgr_t bt_avrcp_mgr_t;
```

Description

This is type bt_avrcp_mgr_t.

__AVRCP_COMMAND_H Macro

File

avrcp_command.h

C

```
#define __AVRCP_COMMAND_H
```

Description

This is macro __AVRCP_COMMAND_H.

__AVRCP_H Macro

File

avrcp.h

C

```
#define __AVRCP_H
```

Description

This is macro __AVRCP_H.

__AVRCP_PRIVATE_H Macro

File

avrcp_private.h

C

```
#define __AVRCP_PRIVATE_H
```

Description

This is macro __AVRCP_PRIVATE_H.

AVC_BATTERY_STATUS_CRITICAL Macro

File

avrcp_command.h

C

```
#define AVC_BATTERY_STATUS_CRITICAL 2
```

Description

This is macro AVC_BATTERY_STATUS_CRITICAL.

AVC_BATTERY_STATUS_EXTERNAL Macro

File

avrcp_command.h

C

```
#define AVC_BATTERY_STATUS_EXTERNAL 3
```

Description

This is macro AVC_BATTERY_STATUS_EXTERNAL.

AVC_BATTERY_STATUS_FULL_CHARGE Macro

File

avrcp_command.h

C

```
#define AVC_BATTERY_STATUS_FULL_CHARGE 4
```

Description

This is macro AVC_BATTERY_STATUS_FULL_CHARGE.

AVC_BATTERY_STATUS_NORMAL Macro

File

avrcp_command.h

C

```
#define AVC_BATTERY_STATUS_NORMAL 0
```

Description

- addtogroup avrcp
- @{
- @name Battery status

AVC_BATTERY_STATUS_WARNING Macro

File

avrcp_command.h

C

```
#define AVC_BATTERY_STATUS_WARNING 1
```

Description

This is macro AVC_BATTERY_STATUS_WARNING.

AVC_CAPABILITY_COMPANY_ID Macro

File

avrcp_command.h

C

```
#define AVC_CAPABILITY_COMPANY_ID 2
```

Description

This is macro AVC_CAPABILITY_COMPANY_ID.

AVC_CAPABILITY_EVENTS_SUPPORTED Macro

File

avrcp_command.h

C

```
#define AVC_CAPABILITY_EVENTS_SUPPORTED 3
```

Description

This is macro AVC_CAPABILITY_EVENTS_SUPPORTED.

AVC_CMD_GENERAL_POWER Macro

File

avrcp_command.h

C

```
#define AVC_CMD_GENERAL_POWER 0xB2
```

Description

This is macro AVC_CMD_GENERAL_POWER.

AVC_CMD_GENERAL_RESERVE Macro

File

avrcp_command.h

C

```
#define AVC_CMD_GENERAL_RESERVE 0x01
```

Description

This is macro AVC_CMD_GENERAL_RESERVE.

AVC_CMD_GENERAL_SUBUNIT_INFO Macro

File

avrcp_command.h

C

```
#define AVC_CMD_GENERAL_SUBUNIT_INFO 0x31
```

Description

This is macro AVC_CMD_GENERAL_SUBUNIT_INFO.

AVC_CMD_GENERAL_UNIT_INFO Macro

File

avrcp_command.h

C

```
#define AVC_CMD_GENERAL_UNIT_INFO 0x30
```

Description

This is macro AVC_CMD_GENERAL_UNIT_INFO.

AVC_CMD_GENERAL_VENDOR_DEPENDENT Macro

File

avrcp_command.h

C

```
#define AVC_CMD_GENERAL_VENDOR_DEPENDENT 0x00
```

Description

This is macro AVC_CMD_GENERAL_VENDOR_DEPENDENT.

AVC_CMD_GENERAL_VERSION Macro

File

avrcp_command.h

C

```
#define AVC_CMD_GENERAL_VERSION 0xB0
```

Description

This is macro AVC_CMD_GENERAL_VERSION.

AVC_CMD_PANEL_PASS_THROUGH Macro

File

avrcp_command.h

C

```
#define AVC_CMD_PANEL_PASS_THROUGH 0x7C
```

Description

This is macro AVC_CMD_PANEL_PASS_THROUGH.

AVC_CTYPE_CONTROL Macro

File

avrcp_command.h

C

```
#define AVC_CTYPE_CONTROL 0
```

Description

- addtogroup avrcp
- @{
- @name Command types

AVC_CTYPE_GENERAL_INQUORY Macro

File

avrcp_command.h

C

```
#define AVC_CTYPE_GENERAL_INQUORY 4
```

Description

This is macro AVC_CTYPE_GENERAL_INQUORY.

AVC_CTYPE_NOTIFY Macro

File

avrcp_command.h

C

```
#define AVC_CTYPE_NOTIFY 3
```

Description

This is macro AVC_CTYPE_NOTIFY.

AVC_CTYPE_SPECIFIC_IQUIRY Macro

File

avrcp_command.h

C

```
#define AVC_CTYPE_SPECIFIC_IQUIRY 2
```

Description

This is macro AVC_CTYPE_SPECIFIC_IQUIRY.

AVC_CTYPE_STATUS Macro

File

avrcp_command.h

C

```
#define AVC_CTYPE_STATUS 1
```

Description

This is macro AVC_CTYPE_STATUS.

AVC_EVENT_ADDRESSED_PLAYER_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_ADDRESSED_PLAYER_CHANGED 0x0b ///< The Addressed Player has been changed, see 6.9.2.
```

Description

< The Addressed Player has been changed, see 6.9.2.

AVC_EVENT_AVAILABLE_PLAYERS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_AVAILABLE_PLAYERS_CHANGED 0x0a ///< The available players have changed, see 6.9
```

Description

< The available players have changed, see 6.9

AVC_EVENT_BATT_STATUS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_BATT_STATUS_CHANGED 0x06 ///< Change in battery status
```

Description

< Change in battery status

AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED 0x0400
```

Description

This is macro AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED.

AVC_EVENT_FLAG_ALL Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_ALL 0x1FFF
```

Description

This is macro AVC_EVENT_FLAG_ALL.

AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED 0x0200
```

Description

This is macro AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED.

AVC_EVENT_FLAG_BATT_STATUS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_BATT_STATUS_CHANGED 0x0020
```

Description

This is macro AVC_EVENT_FLAG_BATT_STATUS_CHANGED.

AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED 0x0100
```

Description

This is macro AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED.

AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED 0x0010
```

Description

This is macro AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED.

AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED 0x0001
```

Description

- addtogroup avrcp
- @{
- @name Notifications mask

AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED 0x0080
```

Description

This is macro AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED.

AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED 0x0040
```

Description

This is macro AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED.

AVC_EVENT_FLAG_TRACK_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_TRACK_CHANGED 0x0002
```

Description

This is macro AVC_EVENT_FLAG_TRACK_CHANGED.

AVC_EVENT_FLAG_TRACK_REACHED_END Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_TRACK_REACHED_END 0x0004
```

Description

This is macro AVC_EVENT_FLAG_TRACK_REACHED_END.

AVC_EVENT_FLAG_TRACK_REACHED_START Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_TRACK_REACHED_START 0x0008
```

Description

This is macro AVC_EVENT_FLAG_TRACK_REACHED_START.

AVC_EVENT_FLAG_UIDS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_UIDS_CHANGED 0x0800
```

Description

This is macro AVC_EVENT_FLAG_UIDS_CHANGED.

AVC_EVENT_FLAG_VOLUME_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_FLAG_VOLUME_CHANGED 0x1000
```

Description

This is macro AVC_EVENT_FLAG_VOLUME_CHANGED.

AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED 0x09 ///< The content of the Now Playing list has changed,
see 6.9.5.
```

Description

< The content of the Now Playing list has changed, see 6.9.5.

AVC_EVENT_PLAYBACK_POS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_PLAYBACK_POS_CHANGED 0x05 ///< Change in playback position. Returned after the specified
playback notification change notification interval
```

Description

< Change in playback position. Returned after the specified playback notification change notification interval

AVC_EVENT_PLAYBACK_STATUS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_PLAYBACK_STATUS_CHANGED 0x01 ///< Change in playback status of the current track.
```

Description

< Change in playback status of the current track.

AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED 0x08 ///< Change in player application setting
```

Description

< Change in player application setting

AVC_EVENT_SYSTEM_STATUS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_SYSTEM_STATUS_CHANGED 0x07 ///< Change in system status
```

Description

< Change in system status

AVC_EVENT_TRACK_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_TRACK_CHANGED 0x02 ///< Change of current track
```

Description

< Change of current track

AVC_EVENT_TRACK_REACHED_END Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_TRACK_REACHED_END 0x03 ///< Reached end of a track
```

Description

< Reached end of a track

AVC_EVENT_TRACK_REACHED_START Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_TRACK_REACHED_START 0x04 ///< Reached start of a track
```

Description

< Reached start of a track

AVC_EVENT_UIDS_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_UIDS_CHANGED 0x0c ///< The UIDs have changed, see 6.10.3.3.
```

Description

< The UIDs have changed, see 6.10.3.3.

AVC_EVENT_VOLUME_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_EVENT_VOLUME_CHANGED 0x0d ///< The volume has been changed locally on the TG, see 6.13.3
```

Description

< The volume has been changed locally on the TG, see 6.13.3

AVC_FLAG_BROWSING_CMD Macro

File

avrcp_command.h

C

```
#define AVC_FLAG_BROWSING_CMD 2
```

Description

This is macro AVC_FLAG_BROWSING_CMD.

AVC_FLAG_PANEL_CLICK Macro

File

avrcp_command.h

C

```
#define AVC_FLAG_PANEL_CLICK 1
```

Description

This is macro AVC_FLAG_PANEL_CLICK.

AVC_FLAG_RESPONSE Macro

File

avrcp_command.h

C

```
#define AVC_FLAG_RESPONSE 4
```

Description

This is macro AVC_FLAG_RESPONSE.

AVC_MAXEVENTS Macro

File

avrcp_command.h

C

```
#define AVC_MAXEVENTS 0x0d
```

Description

This is macro AVC_MAXEVENTS.

AVC_MEDIA_ATTR_FLAG_ALBUM Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_ALBUM 0x04
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_ALBUM.

AVC_MEDIA_ATTR_FLAG_ALL Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_ALL 0x7F
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_ALL.

AVC_MEDIA_ATTR_FLAG_ARTIST Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_ARTIST 0x02
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_ARTIST.

AVC_MEDIA_ATTR_FLAG_GENRE Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_GENRE 0x20
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_GENRE.

AVC_MEDIA_ATTR_FLAG_NUMBER Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_NUMBER 0x08
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_NUMBER.

AVC_MEDIA_ATTR_FLAG_PLAYING_TIME Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_PLAYING_TIME 0x40
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_PLAYING_TIME.

AVC_MEDIA_ATTR_FLAG_TITLE Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_TITLE 0x01
```

Description

- addtogroup avrcp
- @{
- @name Media attribute bitmask

AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER 0x10
```

Description

This is macro AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER.

AVC_MEDIA_ATTR_ID_ALBUM Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_ALBUM 3
```

Description

This is macro AVC_MEDIA_ATTR_ID_ALBUM.

AVC_MEDIA_ATTR_ID_ARTIST Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_ARTIST 2
```

Description

This is macro AVC_MEDIA_ATTR_ID_ARTIST.

AVC_MEDIA_ATTR_ID_GENRE Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_GENRE 6
```

Description

This is macro AVC_MEDIA_ATTR_ID_GENRE.

AVC_MEDIA_ATTR_ID_NUMBER Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_NUMBER 4
```

Description

This is macro AVC_MEDIA_ATTR_ID_NUMBER.

AVC_MEDIA_ATTR_ID_PLAYING_TIME Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_PLAYING_TIME 7
```

Description

This is macro AVC_MEDIA_ATTR_ID_PLAYING_TIME.

AVC_MEDIA_ATTR_ID_TITLE Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_TITLE 1
```

Description

- addtogroup avrcp
- @{
- @name Media attribute IDs

AVC_MEDIA_ATTR_ID_TOTAL_NUMBER Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_ATTR_ID_TOTAL_NUMBER 5
```

Description

This is macro AVC_MEDIA_ATTR_ID_TOTAL_NUMBER.

AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM Macro

File

avrcp_command.h

C

```
#define AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM 0x01 // Folder Item, Media Element Item  The virtual filesystem containing the   Browsed
```

Description

Folder Item, Media Element Item The virtual filesystem containing the Browsed media content of the browsed player

AVC_NOW_PLAYING Macro

File

avrcp_command.h

C

```
#define AVC_NOW_PLAYING 0x03 // Media Element Item              The Now Playing list (or queue)
Addressed
```

Description

Media Element Item The Now Playing list (or queue) Addressed of the addressed player

AVC_PACKET_TYPE_CONTINUE Macro

File

avrcp_command.h

C

```
#define AVC_PACKET_TYPE_CONTINUE 2
```

Description

This is macro AVC_PACKET_TYPE_CONTINUE.

AVC_PACKET_TYPE_END Macro

File

avrcp_command.h

C

```
#define AVC_PACKET_TYPE_END 3
```

Description

This is macro AVC_PACKET_TYPE_END.

AVC_PACKET_TYPE_SINGLE Macro

File

avrcp_command.h

C

```
#define AVC_PACKET_TYPE_SINGLE 0
```

Description

This is macro AVC_PACKET_TYPE_SINGLE.

AVC_PACKET_TYPE_START Macro

File

avrcp_command.h

C

```
#define AVC_PACKET_TYPE_START 1
```

Description

This is macro AVC_PACKET_TYPE_START.

AVC_PANEL_BUTTON_PRESSED Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_BUTTON_PRESSED 0
```

Description

This is macro AVC_PANEL_BUTTON_PRESSED.

AVC_PANEL_BUTTON_RELEASED Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_BUTTON_RELEASED 1
```

Description

This is macro AVC_PANEL_BUTTON_RELEASED.

AVC_PANEL_OPID_0 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_0 0x20
```

Description

define AVC_PANEL_OPID_RESERVER 0x10 define AVC_PANEL_OPID_RESERVER 0x11 define AVC_PANEL_OPID_RESERVER 0x12 define AVC_PANEL_OPID_RESERVER 0x13 define AVC_PANEL_OPID_RESERVER 0x14 define AVC_PANEL_OPID_RESERVER 0x15 define AVC_PANEL_OPID_RESERVER 0x16 define AVC_PANEL_OPID_RESERVER 0x17 define AVC_PANEL_OPID_RESERVER 0x18 define AVC_PANEL_OPID_RESERVER 0x19 define AVC_PANEL_OPID_RESERVER 0x1A define AVC_PANEL_OPID_RESERVER 0x1B define AVC_PANEL_OPID_RESERVER 0x1C define AVC_PANEL_OPID_RESERVER 0x1D define AVC_PANEL_OPID_RESERVER 0x1E define AVC_PANEL_OPID_RESERVER 0x1F

AVC_PANEL_OPID_1 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_1 0x21
```

Description

This is macro AVC_PANEL_OPID_1.

AVC_PANEL_OPID_2 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_2 0x22
```

Description

This is macro AVC_PANEL_OPID_2.

AVC_PANEL_OPID_3 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_3 0x23
```

Description

This is macro AVC_PANEL_OPID_3.

AVC_PANEL_OPID_4 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_4 0x24
```

Description

This is macro AVC_PANEL_OPID_4.

AVC_PANEL_OPID_5 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_5 0x25
```

Description

This is macro AVC_PANEL_OPID_5.

AVC_PANEL_OPID_6 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_6 0x26
```

Description

This is macro AVC_PANEL_OPID_6.

AVC_PANEL_OPID_7 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_7 0x27
```

Description

This is macro AVC_PANEL_OPID_7.

AVC_PANEL_OPID_8 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_8 0x28
```

Description

This is macro AVC_PANEL_OPID_8.

AVC_PANEL_OPID_9 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_9 0x29
```

Description

This is macro AVC_PANEL_OPID_9.

AVC_PANEL_OPID_A Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_A 0x7A
```

Description

This is macro AVC_PANEL_OPID_A.

AVC_PANEL_OPID_ANGLE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_ANGLE 0x50
```

Description

define AVC_PANEL_OPID_RESERVED 0x4E define AVC_PANEL_OPID_RESERVED 0x4F

AVC_PANEL_OPID_APPS_MENU Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_APPS_MENU 0x0F
```

Description

This is macro AVC_PANEL_OPID_APPS_MENU.

AVC_PANEL_OPID_B Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_B 0x7B
```

Description

This is macro AVC_PANEL_OPID_B.

AVC_PANEL_OPID_BACKWARD Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_BACKWARD 0x4C
```

Description

This is macro AVC_PANEL_OPID_BACKWARD.

AVC_PANEL_OPID_C Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_C 0x7C
```

Description

This is macro AVC_PANEL_OPID_C.

AVC_PANEL_OPID_CHANNEL_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_CHANNEL_DOWN 0x31
```

Description

This is macro AVC_PANEL_OPID_CHANNEL_DOWN.

AVC_PANEL_OPID_CHANNEL_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_CHANNEL_UP 0x30
```

Description

define AVC_PANEL_OPID_RESRVED 0x2D define AVC_PANEL_OPID_RESRVED 0x2E define AVC_PANEL_OPID_RESRVED 0x2F

AVC_PANEL_OPID_CLEAR Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_CLEAR 0x2C
```

Description

This is macro AVC_PANEL_OPID_CLEAR.

AVC_PANEL_OPID_CONTENTS_MENU Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_CONTENTS_MENU 0x0B
```

Description

This is macro AVC_PANEL_OPID_CONTENTS_MENU.

AVC_PANEL_OPID_D Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_D 0x7D
```

Description

This is macro AVC_PANEL_OPID_D.

AVC_PANEL_OPID_DISPLAY_INFORMATION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_DISPLAY_INFORMATION 0x35
```

Description

This is macro AVC_PANEL_OPID_DISPLAY_INFORMATION.

AVC_PANEL_OPID_DOT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_DOT 0x2A
```

Description

This is macro AVC_PANEL_OPID_DOT.

AVC_PANEL_OPID_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_DOWN 0x02
```

Description

This is macro AVC_PANEL_OPID_DOWN.

AVC_PANEL_OPID_EJECT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_EJECT 0x4A
```

Description

This is macro AVC_PANEL_OPID_EJECT.

AVC_PANEL_OPID_ENTER Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_ENTER 0x2B
```

Description

This is macro AVC_PANEL_OPID_ENTER.

AVC_PANEL_OPID_EXIT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_EXIT 0x0D
```

Description

This is macro AVC_PANEL_OPID_EXIT.

AVC_PANEL_OPID_F1 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F1 0x71
```

Description

define AVC_PANEL_OPID_RESERVED 0x6D define AVC_PANEL_OPID_RESERVED 0x6E define AVC_PANEL_OPID_RESERVED 0x6F define AVC_PANEL_OPID_RESERVED 0x70

AVC_PANEL_OPID_F2 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F2 0x72
```

Description

This is macro AVC_PANEL_OPID_F2.

AVC_PANEL_OPID_F3 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F3 0x73
```

Description

This is macro AVC_PANEL_OPID_F3.

AVC_PANEL_OPID_F4 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F4 0x74
```

Description

This is macro AVC_PANEL_OPID_F4.

AVC_PANEL_OPID_F5 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F5 0x75
```

Description

This is macro AVC_PANEL_OPID_F5.

AVC_PANEL_OPID_F6 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F6 0x76
```

Description

This is macro AVC_PANEL_OPID_F6.

AVC_PANEL_OPID_F7 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F7 0x77
```

Description

This is macro AVC_PANEL_OPID_F7.

AVC_PANEL_OPID_F8 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F8 0x78
```

Description

This is macro AVC_PANEL_OPID_F8.

AVC_PANEL_OPID_F9 Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_F9 0x79
```

Description

This is macro AVC_PANEL_OPID_F9.

AVC_PANEL_OPID_FAST_FORWARD Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_FAST_FORWARD 0x49
```

Description

This is macro AVC_PANEL_OPID_FAST_FORWARD.

AVC_PANEL_OPID_FAVORITE_MENU Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_FAVORITE_MENU 0x0C
```

Description

This is macro AVC_PANEL_OPID_FAVORITE_MENU.

AVC_PANEL_OPID_FORWARD Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_FORWARD 0x4B
```

Description

This is macro AVC_PANEL_OPID_FORWARD.

AVC_PANEL_OPID_HELP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_HELP 0x36
```

Description

This is macro AVC_PANEL_OPID_HELP.

AVC_PANEL_OPID_INPUT_SELECT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_INPUT_SELECT 0x34
```

Description

This is macro AVC_PANEL_OPID_INPUT_SELECT.

AVC_PANEL_OPID_KEYBORD_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_KEYBORD_FUNCTION 0x6C
```

Description

This is macro AVC_PANEL_OPID_KEYBORD_FUNCTION.

AVC_PANEL_OPID_LEFT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LEFT 0x03
```

Description

This is macro AVC_PANEL_OPID_LEFT.

AVC_PANEL_OPID_LEFT_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LEFT_DOWN 0x08
```

Description

This is macro AVC_PANEL_OPID_LEFT_DOWN.

AVC_PANEL_OPID_LEFT_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LEFT_UP 0x07
```

Description

This is macro AVC_PANEL_OPID_LEFT_UP.

AVC_PANEL_OPID_LINKED_CONTENT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LINKED_CONTENT 0x3F
```

Description

This is macro AVC_PANEL_OPID_LINKED_CONTENT.

AVC_PANEL_OPID_LIST Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LIST 0x4D
```

Description

This is macro AVC_PANEL_OPID_LIST.

AVC_PANEL_OPID_LIVE_TV Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LIVE_TV 0x39
```

Description

This is macro AVC_PANEL_OPID_LIVE_TV.

AVC_PANEL_OPID_LOCK Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_LOCK 0x3B
```

Description

This is macro AVC_PANEL_OPID_LOCK.

AVC_PANEL_OPID_MUTE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_MUTE 0x43
```

Description

This is macro AVC_PANEL_OPID_MUTE.

AVC_PANEL_OPID_MUTE_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_MUTE_FUNCTION 0x65
```

Description

This is macro AVC_PANEL_OPID_MUTE_FUNCTION.

AVC_PANEL_OPID_NEXT_DAY Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_NEXT_DAY 0x3D
```

Description

This is macro AVC_PANEL_OPID_NEXT_DAY.

AVC_PANEL_OPID_ON_DEMAND_MENU Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_ON_DEMAND_MENU 0x0E
```

Description

This is macro AVC_PANEL_OPID_ON_DEMAND_MENU.

AVC_PANEL_OPID_PAGE_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PAGE_DOWN 0x38
```

Description

This is macro AVC_PANEL_OPID_PAGE_DOWN.

AVC_PANEL_OPID_PAGE_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PAGE_UP 0x37
```

Description

This is macro AVC_PANEL_OPID_PAGE_UP.

AVC_PANEL_OPID_PAUSE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PAUSE 0x46
```

Description

This is macro AVC_PANEL_OPID_PAUSE.

AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION 0x61
```

Description

This is macro AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION.

AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION 0x63
```

Description

This is macro AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION.

AVC_PANEL_OPID_PIP_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PIP_DOWN 0x53
```

Description

This is macro AVC_PANEL_OPID_PIP_DOWN.

AVC_PANEL_OPID_PIP_MOVE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PIP_MOVE 0x52
```

Description

This is macro AVC_PANEL_OPID_PIP_MOVE.

AVC_PANEL_OPID_PIP_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PIP_UP 0x54
```

Description

This is macro AVC_PANEL_OPID_PIP_UP.

AVC_PANEL_OPID_PLAY Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PLAY 0x44
```

Description

This is macro AVC_PANEL_OPID_PLAY.

AVC_PANEL_OPID_PLAY_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PLAY_FUNCTION 0x60
```

Description

define AVC_PANEL_OPID_RESERVED 0x56 define AVC_PANEL_OPID_RESERVED 0x57 define AVC_PANEL_OPID_RESERVED 0x58 define AVC_PANEL_OPID_RESERVED 0x5A define AVC_PANEL_OPID_RESERVED 0x5B define AVC_PANEL_OPID_RESERVED 0x5C define AVC_PANEL_OPID_RESERVED 0x5D define AVC_PANEL_OPID_RESERVED 0x5E define AVC_PANEL_OPID_RESERVED 0x5F

AVC_PANEL_OPID_POWER_STATE_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_POWER_STATE_FUNCTION 0x6B
```

Description

This is macro AVC_PANEL_OPID_POWER_STATE_FUNCTION.

AVC_PANEL_OPID_POWER_TOGGLE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_POWER_TOGGLE 0x40
```

Description

This is macro AVC_PANEL_OPID_POWER_TOGGLE.

AVC_PANEL_OPID_PREVIOUS_CHANNEL Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PREVIOUS_CHANNEL 0x32
```

Description

This is macro AVC_PANEL_OPID_PREVIOUS_CHANNEL.

AVC_PANEL_OPID_PREVIOUS_DAY Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_PREVIOUS_DAY 0x3E
```

Description

This is macro AVC_PANEL_OPID_PREVIOUS_DAY.

AVC_PANEL_OPID_RECORD Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RECORD 0x47
```

Description

This is macro AVC_PANEL_OPID_RECORD.

AVC_PANEL_OPID_RECORD_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RECORD_FUNCTION 0x62
```

Description

This is macro AVC_PANEL_OPID_RECORD_FUNCTION.

AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION 0x66
```

Description

This is macro AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION.

AVC_PANEL_OPID_REWIND Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_REWIND 0x48
```

Description

This is macro AVC_PANEL_OPID_REWIND.

AVC_PANEL_OPID_RF_BYPASS Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RF_BYPASS 0x55
```

Description

This is macro AVC_PANEL_OPID_RF_BYPASS.

AVC_PANEL_OPID_RIGHT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RIGHT 0x04
```

Description

This is macro AVC_PANEL_OPID_RIGHT.

AVC_PANEL_OPID_RIGHT_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RIGHT_DOWN 0x06
```

Description

This is macro AVC_PANEL_OPID_RIGHT_DOWN.

AVC_PANEL_OPID_RIGHT_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_RIGHT_UP 0x05
```

Description

This is macro AVC_PANEL_OPID_RIGHT_UP.

AVC_PANEL_OPID_ROOT_MENU Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_ROOT_MENU 0x09
```

Description

This is macro AVC_PANEL_OPID_ROOT_MENU.

AVC_PANEL_OPID_SELECT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SELECT 0x00
```

Description

- addtogroup avrcp
- @{
- @name AV/C Panel PASS THROUGH operation IDs

AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION 0x6A
```

Description

This is macro AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION.

AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION 0x69
```

Description

This is macro AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION.

AVC_PANEL_OPID_SELECT_DISK_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SELECT_DISK_FUNCTION 0x68
```

Description

This is macro AVC_PANEL_OPID_SELECT_DISK_FUNCTION.

AVC_PANEL_OPID_SETUP_MENU Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SETUP_MENU 0x0A
```

Description

This is macro AVC_PANEL_OPID_SETUP_MENU.

AVC_PANEL_OPID_SKIP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SKIP 0x3C
```

Description

This is macro AVC_PANEL_OPID_SKIP.

AVC_PANEL_OPID_SOUND_SELECT Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SOUND_SELECT 0x33
```

Description

This is macro AVC_PANEL_OPID_SOUND_SELECT.

AVC_PANEL_OPID_STOP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_STOP 0x45
```

Description

This is macro AVC_PANEL_OPID_STOP.

AVC_PANEL_OPID_STOP_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_STOP_FUNCTION 0x64
```

Description

This is macro AVC_PANEL_OPID_STOP_FUNCTION.

AVC_PANEL_OPID_SUBPICTURE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_SUBPICTURE 0x51
```

Description

This is macro AVC_PANEL_OPID_SUBPICTURE.

AVC_PANEL_OPID_TUNE_FUNCTION Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_TUNE_FUNCTION 0x67
```

Description

This is macro AVC_PANEL_OPID_TUNE_FUNCTION.

AVC_PANEL_OPID_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_UP 0x01
```

Description

This is macro AVC_PANEL_OPID_UP.

AVC_PANEL_OPID_VENDOR_UNIQUE Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_VENDOR_UNIQUE 0x7E
```

Description

This is macro AVC_PANEL_OPID_VENDOR_UNIQUE.

AVC_PANEL_OPID_VOLUME_DOWN Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_VOLUME_DOWN 0x42
```

Description

This is macro AVC_PANEL_OPID_VOLUME_DOWN.

AVC_PANEL_OPID_VOLUME_UP Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_VOLUME_UP 0x41
```

Description

This is macro AVC_PANEL_OPID_VOLUME_UP.

AVC_PANEL_OPID_ZOOM Macro

File

avrcp_command.h

C

```
#define AVC_PANEL_OPID_ZOOM 0x3A
```

Description

This is macro AVC_PANEL_OPID_ZOOM.

AVC_PDUID_ABORT_CONTINUING_RESPONSE Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_ABORT_CONTINUING_RESPONSE 0x41
```

Description

This is macro AVC_PDUID_ABORT_CONTINUING_RESPONSE.

AVC_PDUID_ADD_TO_NOW_PLAYING Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_ADD_TO_NOW_PLAYING 0x90
```

Description

This is macro AVC_PDUID_ADD_TO_NOW_PLAYING.

AVC_PDUID_CHANGE_PATH Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_CHANGE_PATH 0x72
```

Description

This is macro AVC_PDUID_CHANGE_PATH.

AVC_PDUID_GENERAL_REJECT Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GENERAL_REJECT 0xa0
```

Description

This is macro AVC_PDUID_GENERAL_REJECT.

AVC_PDUID_GET_CURRENT_PLAYER_APPLICATION_SETTING_VALUE Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_CURRENT_PLAYER_APPLICATION_SETTING_VALUE 0x13
```

Description

This is macro AVC_PDUID_GET_CURRENT_PLAYER_APPLICATION_SETTING_VALUE.

AVC_PDUID_GET_ELEMENT_ATTRIBUTES Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_ELEMENT_ATTRIBUTES 0x20
```

Description

This is macro AVC_PDUID_GET_ELEMENT_ATTRIBUTES.

AVC_PDUID_GET_FOLDER_ITEMS Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_FOLDER_ITEMS 0x71
```

Description

This is macro AVC_PDUID_GET_FOLDER_ITEMS.

AVC_PDUID_GET_ITEM_ATTRIBUTES Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_ITEM_ATTRIBUTES 0x73
```

Description

This is macro AVC_PDUID_GET_ITEM_ATTRIBUTES.

AVC_PDUID_GET_PLAY_STATUS Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_PLAY_STATUS 0x30
```

Description

This is macro AVC_PDUID_GET_PLAY_STATUS.

AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_ATTRIBUTE_TEXT Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_ATTRIBUTE_TEXT 0x15
```

Description

This is macro AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_ATTRIBUTE_TEXT.

AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_VALUE_TEXT Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_VALUE_TEXT 0x16
```

Description

This is macro AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_VALUE_TEXT.

AVC_PDUID_GETCAPABILITIES Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_GETCAPABILITIES 0x10
```

Description

This is macro AVC_PDUID_GETCAPABILITIES.

AVC_PDUID_INFORM_BATTERY_STATUS_OF_CT Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_INFORM_BATTERY_STATUS_OF_CT 0x18
```

Description

This is macro AVC_PDUID_INFORM_BATTERY_STATUS_OF_CT.

AVC_PDUID_INFORM_DISPLAYABLE_CHARACTER_SET Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_INFORM_DISPLAYABLE_CHARACTER_SET 0x17
```

Description

This is macro AVC_PDUID_INFORM_DISPLAYABLE_CHARACTER_SET.

AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_ATTRIBUTES Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_ATTRIBUTES 0x11
```

Description

This is macro AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_ATTRIBUTES.

AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_VALUES Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_VALUES 0x12
```

Description

This is macro AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_VALUES.

AVC_PDUID_PLAY_ITEM Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_PLAY_ITEM 0x74
```

Description

This is macro AVC_PDUID_PLAY_ITEM.

AVC_PDUID_REGISTER_NOTIFICATION Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_REGISTER_NOTIFICATION 0x31
```

Description

This is macro AVC_PDUID_REGISTER_NOTIFICATION.

AVC_PDUID_REQUEST_CONTINUING_RESPONSE Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_REQUEST_CONTINUING_RESPONSE 0x40
```

Description

This is macro AVC_PDUID_REQUEST_CONTINUING_RESPONSE.

AVC_PDUID_SEARCH Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_SEARCH 0x80
```

Description

This is macro AVC_PDUID_SEARCH.

AVC_PDUID_SET_ABSOLUTE_VOLUME Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_SET_ABSOLUTE_VOLUME 0x50
```

Description

This is macro AVC_PDUID_SET_ABSOLUTE_VOLUME.

AVC_PDUID_SET_ADDRESSED_PLAYER Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_SET_ADDRESSED_PLAYER 0x60
```

Description

This is macro AVC_PDUID_SET_ADDRESSED_PLAYER.

AVC_PDUID_SET_BROWSED_PLAYER Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_SET_BROWSED_PLAYER 0x70
```

Description

This is macro AVC_PDUID_SET_BROWSED_PLAYER.

AVC_PDUID_SET_PLAYER_APPLICATION_SETTING_VALUE Macro

File

avrcp_command.h

C

```
#define AVC_PDUID_SET_PLAYER_APPLICATION_SETTING_VALUE 0x14
```

Description

This is macro AVC_PDUID_SET_PLAYER_APPLICATION_SETTING_VALUE.

AVC_PLAY_STATUS_ERROR Macro

File

avrcp_command.h

C

```
#define AVC_PLAY_STATUS_ERROR 0xFF
```

Description

This is macro AVC_PLAY_STATUS_ERROR.

AVC_PLAY_STATUS_FW_SEEK Macro

File

avrcp_command.h

C

```
#define AVC_PLAY_STATUS_FW_SEEK 3
```

Description

This is macro AVC_PLAY_STATUS_FW_SEEK.

AVC_PLAY_STATUS_PAUSED Macro

File

avrcp_command.h

C

```
#define AVC_PLAY_STATUS_PAUSED 2
```

Description

This is macro AVC_PLAY_STATUS_PAUSED.

AVC_PLAY_STATUS_PLAYING Macro

File

avrcp_command.h

C

```
#define AVC_PLAY_STATUS_PLAYING 1
```

Description

This is macro AVC_PLAY_STATUS_PLAYING.

AVC_PLAY_STATUS_REV_SEEK Macro

File

avrcp_command.h

C

```
#define AVC_PLAY_STATUS_REV_SEEK 4
```

Description

This is macro AVC_PLAY_STATUS_REV_SEEK.

AVC_PLAY_STATUS_STOPPED Macro

File

avrcp_command.h

C

```
#define AVC_PLAY_STATUS_STOPPED 0
```

Description

- addtogroup avrcp
- @{
- @name Play status

AVC_PLAYER_SETTING_EQUALIZER_OFF Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_EQUALIZER_OFF 1
```

Description

This is macro AVC_PLAYER_SETTING_EQUALIZER_OFF.

AVC_PLAYER_SETTING_EQUALIZER_ON Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_EQUALIZER_ON 2
```

Description

This is macro AVC_PLAYER_SETTING_EQUALIZER_ON.

AVC_PLAYER_SETTING_EQUALIZER_STATUS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_EQUALIZER_STATUS 1
```

Description

This is macro AVC_PLAYER_SETTING_EQUALIZER_STATUS.

AVC_PLAYER_SETTING_REPEAT_ALL_TRACKS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_REPEAT_ALL_TRACKS 3
```

Description

This is macro AVC_PLAYER_SETTING_REPEAT_ALL_TRACKS.

AVC_PLAYER_SETTING_REPEAT_GROUP Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_REPEAT_GROUP 4
```

Description

This is macro AVC_PLAYER_SETTING_REPEAT_GROUP.

AVC_PLAYER_SETTING_REPEAT_MODE_OFF Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_REPEAT_MODE_OFF 1
```

Description

This is macro AVC_PLAYER_SETTING_REPEAT_MODE_OFF.

AVC_PLAYER_SETTING_REPEAT_MODE_STATUS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_REPEAT_MODE_STATUS 2
```

Description

This is macro AVC_PLAYER_SETTING_REPEAT_MODE_STATUS.

AVC_PLAYER_SETTING_REPEAT_SINGLE_TRACK Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_REPEAT_SINGLE_TRACK 2
```

Description

This is macro AVC_PLAYER_SETTING_REPEAT_SINGLE_TRACK.

AVC_PLAYER_SETTING_SCAN_ALL_TRACKS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SCAN_ALL_TRACKS 2
```

Description

This is macro AVC_PLAYER_SETTING_SCAN_ALL_TRACKS.

AVC_PLAYER_SETTING_SCAN_GROUP Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SCAN_GROUP 3
```

Description

This is macro AVC_PLAYER_SETTING_SCAN_GROUP.

AVC_PLAYER_SETTING_SCAN_OFF Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SCAN_OFF 1
```

Description

This is macro AVC_PLAYER_SETTING_SCAN_OFF.

AVC_PLAYER_SETTING_SCAN_STATUS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SCAN_STATUS 4
```

Description

This is macro AVC_PLAYER_SETTING_SCAN_STATUS.

AVC_PLAYER_SETTING_SHUFFLE_ALL_TRACKS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SHUFFLE_ALL_TRACKS 2
```

Description

This is macro AVC_PLAYER_SETTING_SHUFFLE_ALL_TRACKS.

AVC_PLAYER_SETTING_SHUFFLE_GROUP Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SHUFFLE_GROUP 3
```

Description

This is macro AVC_PLAYER_SETTING_SHUFFLE_GROUP.

AVC_PLAYER_SETTING_SHUFFLE_OFF Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SHUFFLE_OFF 1
```

Description

This is macro AVC_PLAYER_SETTING_SHUFFLE_OFF.

AVC_PLAYER_SETTING_SHUFFLE_STATUS Macro

File

avrcp_command.h

C

```
#define AVC_PLAYER_SETTING_SHUFFLE_STATUS 3
```

Description

This is macro AVC_PLAYER_SETTING_SHUFFLE_STATUS.

AVC_RESPONSE_ACCEPTED Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_ACCEPTED 0x9
```

Description

This is macro AVC_RESPONSE_ACCEPTED.

AVC_RESPONSE_CHANGED Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_CHANGED 0xD
```

Description

This is macro AVC_RESPONSE_CHANGED.

AVC_RESPONSE_IMPLEMENTED Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_IMPLEMENTED 0xC
```

Description

This is macro AVC_RESPONSE_IMPLEMENTED.

AVC_RESPONSE_IN_TRANSITION Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_IN_TRANSITION 0xB
```

Description

This is macro AVC_RESPONSE_IN_TRANSITION.

AVC_RESPONSE_INTERIM Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_INTERIM 0xF
```

Description

This is macro AVC_RESPONSE_INTERIM.

AVC_RESPONSE_NOT_IMPLEMENTED Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_NOT_IMPLEMENTED 0x8
```

Description

- addtogroup avrcp
- @{
- @name Response types

AVC_RESPONSE_REJECTED Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_REJECTED 0xA
```

Description

This is macro AVC_RESPONSE_REJECTED.

AVC_RESPONSE_STABLE Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_STABLE 0xC
```

Description

This is macro AVC_RESPONSE_STABLE.

AVC_RESPONSE_TIMEOUT Macro

File

avrcp_command.h

C

```
#define AVC_RESPONSE_TIMEOUT 0xF0
```

Description

This is macro AVC_RESPONSE_TIMEOUT.

AVC_SCOPE_MEDIA_PLAYER_LIST Macro

File

avrcp_command.h

C

```
#define AVC_SCOPE_MEDIA_PLAYER_LIST 0x00 // Media Player Item                Contains all available media
players    None
```

Description

Media Player Item Contains all available media players None

AVC_SEARCH Macro

File

avrcp_command.h

C

```
#define AVC_SEARCH 0x02 // Media Element Item              The results of a search operation       Browsed
```

Description

Media Element Item The results of a search operation Browsed on the browsed player

AVC_SUBUNIT_ID_EXTENDED_TO_NEXT_BYTE Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_ID_EXTENDED_TO_NEXT_BYTE 0x5
```

Description

This is macro AVC_SUBUNIT_ID_EXTENDED_TO_NEXT_BYTE.

AVC_SUBUNIT_ID_IGNORE Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_ID_IGNORE 0x7
```

Description

This is macro AVC_SUBUNIT_ID_IGNORE.

AVC_SUBUNIT_TYPE_AUDIO Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_AUDIO 0x01
```

Description

This is macro AVC_SUBUNIT_TYPE_AUDIO.

AVC_SUBUNIT_TYPE_BULLETIN_BOARD Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_BULLETIN_BOARD 0x0A
```

Description

This is macro AVC_SUBUNIT_TYPE_BULLETIN_BOARD.

AVC_SUBUNIT_TYPE_CA Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_CA 0x06
```

Description

This is macro AVC_SUBUNIT_TYPE_CA.

AVC_SUBUNIT_TYPE_CAMERA Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_CAMERA 0x07
```

Description

This is macro AVC_SUBUNIT_TYPE_CAMERA.

AVC_SUBUNIT_TYPE_CAMERA_STORAGE Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_CAMERA_STORAGE 0x0B
```

Description

This is macro AVC_SUBUNIT_TYPE_CAMERA_STORAGE.

AVC_SUBUNIT_TYPE_DISC Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_DISC 0x03
```

Description

This is macro AVC_SUBUNIT_TYPE_DISC.

AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE 0x1E
```

Description

This is macro AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE.

AVC_SUBUNIT_TYPE_MONITOR Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_MONITOR 0x00
```

Description

- addtogroup avrcp
- @{
- @name Subunit types

AVC_SUBUNIT_TYPE_PANEL Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_PANEL 0x09
```

Description

This is macro AVC_SUBUNIT_TYPE_PANEL.

AVC_SUBUNIT_TYPE_PRINTER Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_PRINTER 0x02
```

Description

This is macro AVC_SUBUNIT_TYPE_PRINTER.

AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER 0x04
```

Description

This is macro AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER.

AVC_SUBUNIT_TYPE_TUNER Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_TUNER 0x05
```

Description

This is macro AVC_SUBUNIT_TYPE_TUNER.

AVC_SUBUNIT_TYPE_UNIT Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_UNIT 0x1F
```

Description

This is macro AVC_SUBUNIT_TYPE_UNIT.

AVC_SUBUNIT_TYPE_VENDOR_UNIQUE Macro

File

avrcp_command.h

C

```
#define AVC_SUBUNIT_TYPE_VENDOR_UNIQUE 0x1C
```

Description

This is macro AVC_SUBUNIT_TYPE_VENDOR_UNIQUE.

AVC_VOLUME_MAX Macro

File

avrcp_command.h

C

```
#define AVC_VOLUME_MAX 0x7F // 100%
```

Description

100

AVC_VOLUME_MIN Macro

File

avrcp_command.h

C

```
#define AVC_VOLUME_MIN 0 // 0%
```

Description

0

AVRCP_BTSIG_COMPANY_ID Macro

File

avrcp.h

C

```
#define AVRCP_BTSIG_COMPANY_ID 0x001958
```

Description

This is macro AVRCP_BTSIG_COMPANY_ID.

AVRCP_CHANNEL_FLAG_LISTENING Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_FLAG_LISTENING 1
```

Description

This is macro AVRCP_CHANNEL_FLAG_LISTENING.

AVRCP_CHANNEL_FLAG_PLAY_STATUS_REQUESTED Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_FLAG_PLAY_STATUS_REQUESTED 8
```

Description

This is macro AVRCP_CHANNEL_FLAG_PLAY_STATUS_REQUESTED.

AVRCP_CHANNEL_FLAG_REGISTERING_NOTIFICATIONS Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_FLAG_REGISTERING_NOTIFICATIONS 4
```

Description

This is macro AVRCP_CHANNEL_FLAG_REGISTERING_NOTIFICATIONS.

AVRCP_CHANNEL_FLAG_SENDING Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_FLAG_SENDING 2
```

Description

This is macro AVRCP_CHANNEL_FLAG_SENDING.

AVRCP_CHANNEL_STATE_CONNECTED Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_STATE_CONNECTED 3
```

Description

This is macro AVRCP_CHANNEL_STATE_CONNECTED.

AVRCP_CHANNEL_STATE_CONNECTING Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_STATE_CONNECTING 2
```

Description

This is macro AVRCP_CHANNEL_STATE_CONNECTING.

AVRCP_CHANNEL_STATE_DISCONNECTING Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_STATE_DISCONNECTING 4
```

Description

This is macro AVRCP_CHANNEL_STATE_DISCONNECTING.

AVRCP_CHANNEL_STATE_FREE Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_STATE_FREE 0
```

Description

This is macro AVRCP_CHANNEL_STATE_FREE.

AVRCP_CHANNEL_STATE_IDLE Macro

File

avrcp.h

C

```
#define AVRCP_CHANNEL_STATE_IDLE 1
```

Description

This is macro AVRCP_CHANNEL_STATE_IDLE.

AVRCP_COMMAND_TYPE_BROWSING Macro

File

avrcp_command.h

C

```
#define AVRCP_COMMAND_TYPE_BROWSING 1
```

Description

This is macro AVRCP_COMMAND_TYPE_BROWSING.

AVRCP_COMMAND_TYPE_CONTROL Macro

File

avrcp_command.h

C

```
#define AVRCP_COMMAND_TYPE_CONTROL 0
```

Description

This is macro AVRCP_COMMAND_TYPE_CONTROL.

AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED 79    ///< This event is generated when a local device
received a response to a "add to now playing" request.
```

Description

< This event is generated when a local device received a response to a "add to now playing" request.

AVRCP_EVT_ADDRESSED_PLAYER_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_ADDRESSED_PLAYER_CHANGED 73    ///< This event is generated when a local device received
a "addressed player changed" notification.
```

Description

< This event is generated when a local device received a "addressed player changed" notification.

AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED 72    ///< This event is generated when a local device received
a "available players changed" notification.
```

Description

< This event is generated when a local device received a "available players changed" notification.

AVRCP_EVT_BATT_STATUS_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_BATT_STATUS_CHANGED 68    ///< This event is generated when a local device received a
"battery status changed" notification.
```

Description

< This event is generated when a local device received a "battery status changed" notification.

AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED 83    ///< This event is generated when a local device
received a "battery status of controller" command.
```

Description

< This event is generated when a local device received a "battery status of controller" command.

AVRCP_EVT_BROWSING_CHANNEL_CONNECTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_BROWSING_CHANNEL_CONNECTED 4    ///< This event is generated when a browsing channel
between two AVRCP entities has been established.
```

Description

< This event is generated when a browsing channel between two AVRCP entities has been established.

AVRCP_EVT_BROWSING_CHANNEL_DISCONNECTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_BROWSING_CHANNEL_DISCONNECTED 5    ///< This event is generated when a browsing channel
between two AVRCP entities has been terminated.
```

Description

< This event is generated when a browsing channel between two AVRCP entities has been terminated.

AVRCP_EVT_BROWSING_CONNECTION_FAILED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_BROWSING_CONNECTION_FAILED 6    ///< This event is generated when a local device failed
to create a browsing channel between two AVRCP entities.
```

Description

< This event is generated when a local device failed to create a browsing channel between two AVRCP entities.

AVRCP_EVT_COMPANY_ID_LIST_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_COMPANY_ID_LIST_RECEIVED 51    ///< This event is generated when a local device received
a response to a "get company id" request.
```

Description

< This event is generated when a local device received a response to a "get company id" request.

AVRCP_EVT_CONTROL_CHANNEL_CONNECTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_CONTROL_CHANNEL_CONNECTED 1    ///< This event is generated when a control channel
between two AVRCP entities has been established.
```

Description

< This event is generated when a control channel between two AVRCP entities has been established.

AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED 2    ///< This event is generated when a control channel
between two AVRCP entities has been terminated.
```

Description

< This event is generated when a control channel between two AVRCP entities has been terminated.

AVRCP_EVT_CONTROL_CONNECTION_FAILED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_CONTROL_CONNECTION_FAILED 3    ///< This event is generated when a local device failed to
create a control channel between two AVRCP entities.
```

Description

< This event is generated when a local device failed to create a control channel between two AVRCP entities.

AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED 84    ///< This event is generated when a local device
received a "displayable chracter set command" request.
```

Description

< This event is generated when a local device received a "displayable chracter set command" request.

AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED 85    ///< This event is generated when a local device
received a "get element attributes" request.
```

Description

< This event is generated when a local device received a "get element attributes" request.

AVRCP_EVT_EVENT_ID_LIST_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_EVENT_ID_LIST_RECEIVED 52    ///< This event is generated when a local device received a response to a "get supported events" request.
```

Description

< This event is generated when a local device received a response to a "get supported events" request.

AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED 61    ///< This event is generated when a local device received a response to a "get media element attributes" request.
```

Description

< This event is generated when a local device received a response to a "get media element attributes" request.

AVRCP_EVT_GET_PLAY_STATUS_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_GET_PLAY_STATUS_RECEIVED 62    ///< This event is generated when a local device received a response to a "get play status" request.
```

Description

< This event is generated when a local device received a response to a "get play status" request.

AVRCP_EVT_INFORM_BATTERY_STATUS_OF_CT_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_INFORM_BATTERY_STATUS_OF_CT_COMPLETED 60    ///< This event is generated when a local device received a response to a "inform battery status" request.
```

Description

< This event is generated when a local device received a response to a "inform battery status" request.

AVRCP_EVT_INFORM_DISPLAYABLE_CHARACTER_SET_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_INFORM_DISPLAYABLE_CHARACTER_SET_COMPLETED 59    ///< This event is generated when a local device received a response to a "inform displayable character set" request.
```

Description

< This event is generated when a local device received a response to a "inform displayable character set" request.

AVRCP_EVT_NOTHING Macro

File

avrcp.h

C

```
#define AVRCP_EVT_NOTHING 0
```

Description

addtogroup avrcp @{

@name Events

details The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.

AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED 71    ///< This event is generated when a local device received a "now playing content changed" notification.
```

Description

< This event is generated when a local device received a "now playing content changed" notification.

AVRCP_EVT_PANEL_COMMAND_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PANEL_COMMAND_RECEIVED 81    ///< This event is generated when a local device received a PASS THROUGH command.
```

Description

< This event is generated when a local device received a PASS THROUGH command.

AVRCP_EVT_PANEL_RESPONSE_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PANEL_RESPONSE_RECEIVED 50    ///< This event is generated when a local device received a response to a PASS THROUGH command.
```

Description

< This event is generated when a local device received a response to a PASS THROUGH command.

AVRCP_EVT_PLAY_ITEM_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAY_ITEM_COMPLETED 78    ///< This event is generated when a local device received a response to a "play item" request.
```

Description

< This event is generated when a local device received a response to a "play item" request.

AVRCP_EVT_PLAYBACK_POS_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYBACK_POS_CHANGED 67    ///< This event is generated when a local device received a "playback position changed" notification.
```

Description

< This event is generated when a local device received a "playback position changed" notification.

AVRCP_EVT_PLAYBACK_STATUS_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYBACK_STATUS_CHANGED 63    ///< This event is generated when a local device received a "play status changed" notification.
```

Description

< This event is generated when a local device received a "play status changed" notification.

AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED 70    ///< This event is generated when a local device received a "player application setting changed" notification.
```

Description

< This event is generated when a local device received a "player application setting changed" notification.

AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED 55    ///< This event is generated when a local device received a response to a "get current player setting attribute values" request.
```

Description

< This event is generated when a local device received a response to a "get current player setting attribute values" request.

AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED 53    ///< This event is generated when a local device received a response to a "get supported player setting attributes" request.
```

Description

< This event is generated when a local device received a response to a "get supported player setting attributes" request.

AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED 57    ///< This event is generated when a local device received a response to a "get player setting attributes displayable text" request.
```

Description

< This event is generated when a local device received a response to a "get player setting attributes displayable text" request.

AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED 54    ///< This event is generated when a local device received a response to a "get player setting attribute values" request.
```

Description

< This event is generated when a local device received a response to a "get player setting attribute values" request.

AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED 58    ///< This event is generated when a local device received a response to a "get player setting attribute values displayable text" request.
```

Description

< This event is generated when a local device received a response to a "get player setting attribute values displayable text" request.

AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED 86    ///< This event is generated when a local device received a "register notification" request.
```

Description

< This event is generated when a local device received a "register notification" request.

AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED 80    ///< This event is generated when a local device received a response to a "register notification" request.
```

Description

< This event is generated when a local device received a response to a "register notification" request.

AVRCP_EVT_SEARCH_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_SEARCH_COMPLETED 7    ///< This event is generated when a local device completed
searching for nearby targets.
```

Description

< This event is generated when a local device completed searching for nearby targets.

AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED 76    ///< This event is generated when a local device
received a response to a "set absolute volume" request.
```

Description

< This event is generated when a local device received a response to a "set absolute volume" request.

AVRCP_EVT_SET_ABSOLUTE_VOLUME_REQUESTED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_SET_ABSOLUTE_VOLUME_REQUESTED 82    ///< This event is generated when a local device
received a "set absolute volume" request.
```

Description

< This event is generated when a local device received a "set absolute volume" request.

AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED 77    ///< This event is generated when a local device
received a response to a "set addressed player" request.
```

Description

< This event is generated when a local device received a response to a "set addressed player" request.

AVRCP_EVT_SET_PLAYER_SETTING_VALUES_COMPLETED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_SET_PLAYER_SETTING_VALUES_COMPLETED 56    ///< This event is generated when a local
device received a response to a "set player setting attribute values" request.
```

Description

< This event is generated when a local device received a response to a "set player setting attribute values" request.

AVRCP_EVT_SYSTEM_STATUS_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_SYSTEM_STATUS_CHANGED 69    ///< This event is generated when a local device received a
"system status changed" notification.
```

Description

< This event is generated when a local device received a "system status changed" notification.

AVRCP_EVT_TRACK_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_TRACK_CHANGED 64    ///< This event is generated when a local device received a "track
changed changed" notification.
```

Description

< This event is generated when a local device received a "track changed changed" notification.

AVRCP_EVT_TRACK_REACHED_END Macro

File

avrcp.h

C

```
#define AVRCP_EVT_TRACK_REACHED_END 65    ///< This event is generated when a local device received a
"track reached end" notification.
```

Description

< This event is generated when a local device received a "track reached end" notification.

AVRCP_EVT_TRACK_REACHED_START Macro

File

avrcp.h

C

```
#define AVRCP_EVT_TRACK_REACHED_START 66    ///< This event is generated when a local device received a
"track reached start" notification.
```

Description

< This event is generated when a local device received a "track reached start" notification.

AVRCP_EVT_UIDS_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_UIDS_CHANGED 74    ///< This event is generated when a local device received a "UIDs
changed" notification.
```

Description

< This event is generated when a local device received a "UIDs changed" notification.

AVRCP_EVT_VOLUME_CHANGED Macro

File

avrcp.h

C

```
#define AVRCP_EVT_VOLUME_CHANGED 75    ///< This event is generated when a local device received a "volume changed" notification.
```

Description

< This event is generated when a local device received a "volume changed" notification.

AVRCP_MANAGER_FLAG_SEARCHING Macro

File

avrcp.h

C

```
#define AVRCP_MANAGER_FLAG_SEARCHING 1
```

Description

This is macro AVRCP_MANAGER_FLAG_SEARCHING.

AVRCP_MANAGER_STATE_IDLE Macro

File

avrcp.h

C

```
#define AVRCP_MANAGER_STATE_IDLE 0
```

Description

This is macro AVRCP_MANAGER_STATE_IDLE.

AVRCP_MAX_ELEMENT_ATTRIBUTES Macro

File

avrcp_private.h

C

```
#define AVRCP_MAX_ELEMENT_ATTRIBUTES 7
```

Description

This is macro AVRCP_MAX_ELEMENT_ATTRIBUTES.

bt_avrcp_0_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_0_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_0)
```

Description

This is macro bt_avrcp_0_click.

bt_avrcp_0_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_0_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_0)
```

Description

This is macro bt_avrcp_0_press.

bt_avrcp_1_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_1_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_1)
```

Description

This is macro bt_avrcp_1_click.

bt_avrcp_1_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_1_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_1)
```

Description

This is macro bt_avrcp_1_press.

bt_avrcp_2_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_2_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_2)
```

Description

This is macro bt_avrcp_2_click.

bt_avrcp_2_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_2_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_2)
```

Description

This is macro bt_avrcp_2_press.

bt_avrcp_3_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_3_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_3)
```

Description

This is macro bt_avrcp_3_click.

bt_avrcp_3_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_3_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_3)
```

Description

This is macro bt_avrcp_3_press.

bt_avrcp_4_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_4_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_4)
```

Description

This is macro bt_avrcp_4_click.

bt_avrcp_4_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_4_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_4)
```

Description

This is macro bt_avrcp_4_press.

bt_avrcp_5_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_5_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_5)
```

Description

This is macro bt_avrcp_5_click.

bt_avrcp_5_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_5_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_5)
```

Description

This is macro bt_avrcp_5_press.

bt_avrcp_6_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_6_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_6)
```

Description

This is macro bt_avrcp_6_click.

bt_avrcp_6_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_6_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_6)
```

Description

This is macro bt_avrcp_6_press.

bt_avrcp_7_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_7_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_7)
```

Description

This is macro bt_avrcp_7_click.

bt_avrcp_7_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_7_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_7)
```

Description

This is macro bt_avrcp_7_press.

bt_avrcp_8_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_8_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_8)
```

Description

This is macro bt_avrcp_8_click.

bt_avrcp_8_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_8_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_8)
```

Description

This is macro bt_avrcp_8_press.

bt_avrcp_9_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_9_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_9)
```

Description

This is macro bt_avrcp_9_click.

bt_avrcp_9_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_9_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_9)
```

Description

This is macro bt_avrcp_9_press.

bt_avrcp_angle_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_angle_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_ANGLE)
```

Description

This is macro bt_avrcp_angle_click.

bt_avrcp_angle_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_angle_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_ANGLE)
```

Description

This is macro bt_avrcp_angle_press.

bt_avrcp_backward_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_backward_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_BACKWARD)
```

Description

This is macro bt_avrcp_backward_click.

bt_avrcp_backward_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_backward_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_BACKWARD)
```

Description

This is macro bt_avrcp_backward_press.

bt_avrcp_channel_down_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_channel_down_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_CHANNEL_DOWN)
```

Description

This is macro bt_avrcp_channel_down_click.

bt_avrcp_channel_down_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_channel_down_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_CHANNEL_DOWN)
```

Description

This is macro bt_avrcp_channel_down_press.

bt_avrcp_channel_up_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_channel_up_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_CHANNEL_UP)
```

Description

This is macro bt_avrcp_channel_up_click.

bt_avrcp_channel_up_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_channel_up_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_CHANNEL_UP)
```

Description

This is macro bt_avrcp_channel_up_press.

bt_avrcp_clear_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_clear_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_CLEAR)
```

Description

This is macro bt_avrcp_clear_click.

bt_avrcp_clear_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_clear_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_CLEAR)
```

Description

This is macro bt_avrcp_clear_press.

bt_avrcp_content_menu_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_content_menu_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_CONTENTS_MENU)
```

Description

This is macro bt_avrcp_content_menu_click.

bt_avrcp_contents_menu_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_contents_menu_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_CONTENTS_MENU)
```

Description

This is macro bt_avrcp_contents_menu_press.

bt_avrcp_display_info_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_display_info_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_DISPLAY_INFORMATION)
```

Description

This is macro bt_avrcp_display_info_click.

bt_avrcp_display_info_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_display_info_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_DISPLAY_INFORMATION)
```

Description

This is macro bt_avrcp_display_info_press.

bt_avrcp_dot_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_dot_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_DOT)
```

Description

This is macro bt_avrcp_dot_click.

bt_avrcp_dot_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_dot_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_DOT)
```

Description

This is macro bt_avrcp_dot_press.

bt_avrcp_down_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_down_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_DOWN)
```

Description

This is macro bt_avrcp_down_click.

bt_avrcp_down_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_down_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_DOWN)
```

Description

This is macro bt_avrcp_down_press.

bt_avrcp_eject_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_eject_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_EJECT)
```

Description

This is macro bt_avrcp_eject_click.

bt_avrcp_eject_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_eject_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_EJECT)
```

Description

This is macro bt_avrcp_eject_press.

bt_avrcp_enter_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_enter_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_ENTER)
```

Description

This is macro bt_avrcp_enter_click.

bt_avrcp_enter_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_enter_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_ENTER)
```

Description

This is macro bt_avrcp_enter_press.

bt_avrcp_exit_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_exit_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_EXIT)
```

Description

This is macro bt_avrcp_exit_click.

bt_avrcp_exit_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_exit_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_EXIT)
```

Description

This is macro bt_avrcp_exit_press.

bt_avrcp_f1_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f1_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F1)
```

Description

This is macro bt_avrcp_f1_click.

bt_avrcp_f1_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f1_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F1)
```

Description

This is macro bt_avrcp_f1_press.

bt_avrcp_f2_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f2_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F2)
```

Description

This is macro bt_avrcp_f2_click.

bt_avrcp_f2_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f2_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F2)
```

Description

This is macro bt_avrcp_f2_press.

bt_avrcp_f3_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f3_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F3)
```

Description

This is macro bt_avrcp_f3_click.

bt_avrcp_f3_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f3_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F3)
```

Description

This is macro bt_avrcp_f3_press.

bt_avrcp_f4_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f4_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F4)
```

Description

This is macro bt_avrcp_f4_click.

bt_avrcp_f4_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f4_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F4)
```

Description

This is macro bt_avrcp_f4_press.

bt_avrcp_f5_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f5_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F5)
```

Description

This is macro bt_avrcp_f5_click.

bt_avrcp_f5_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f5_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F5)
```

Description

This is macro bt_avrcp_f5_press.

bt_avrcp_f6_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f6_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F6)
```

Description

This is macro bt_avrcp_f6_click.

bt_avrcp_f6_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f6_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F6)
```

Description

This is macro bt_avrcp_f6_press.

bt_avrcp_f7_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f7_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F7)
```

Description

This is macro bt_avrcp_f7_click.

bt_avrcp_f7_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f7_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F7)
```

Description

This is macro bt_avrcp_f7_press.

bt_avrcp_f8_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f8_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F8)
```

Description

This is macro bt_avrcp_f8_click.

bt_avrcp_f8_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f8_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F8)
```

Description

This is macro bt_avrcp_f8_press.

bt_avrcp_f9_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f9_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_F9)
```

Description

This is macro bt_avrcp_f9_click.

bt_avrcp_f9_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f9_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_F9)
```

Description

This is macro bt_avrcp_f9_press.

bt_avrcp_fast_forward_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_fast_forward_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_FAST_FORWARD)
```

Description

This is macro bt_avrcp_fast_forward_click.

bt_avrcp_fast_forward_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_fast_forward_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_FAST_FORWARD)
```

Description

This is macro bt_avrcp_fast_forward_press.

bt_avrcp_favorite_menu_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_favorite_menu_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_FAVORITE_MENU)
```

Description

This is macro bt_avrcp_favorite_menu_click.

bt_avrcp_favorive_menu_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_favorive_menu_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_FAVORIVE_MENU)
```

Description

This is macro bt_avrcp_favorive_menu_press.

bt_avrcp_forward_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_forward_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_FORWARD)
```

Description

This is macro bt_avrcp_forward_click.

bt_avrcp_forward_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_forward_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_FORWARD)
```

Description

This is macro bt_avrcp_forward_press.

bt_avrcp_help_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_help_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_HELP)
```

Description

This is macro bt_avrcp_help_click.

bt_avrcp_help_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_help_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_HELP)
```

Description

This is macro bt_avrcp_help_press.

bt_avrcp_input_select_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_input_select_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_INPUT_SELECT)
```

Description

This is macro bt_avrcp_input_select_click.

bt_avrcp_input_select_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_input_select_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_INPUT_SELECT)
```

Description

This is macro bt_avrcp_input_select_press.

bt_avrcp_left_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_LEFT)
```

Description

This is macro bt_avrcp_left_click.

bt_avrcp_left_down_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_down_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_LEFT_DOWN)
```

Description

This is macro bt_avrcp_left_down_click.

bt_avrcp_left_down_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_down_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_LEFT_DOWN)
```

Description

This is macro bt_avrcp_left_down_press.

bt_avrcp_left_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_LEFT)
```

Description

This is macro bt_avrcp_left_press.

bt_avrcp_left_up_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_up_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_LEFT_UP)
```

Description

This is macro bt_avrcp_left_up_click.

bt_avrcp_left_up_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_up_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_LEFT_UP)
```

Description

This is macro bt_avrcp_left_up_press.

bt_avrcp_mute_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_mute_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_MUTE)
```

Description

This is macro bt_avrcp_mute_click.

bt_avrcp_mute_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_mute_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_MUTE)
```

Description

This is macro bt_avrcp_mute_press.

bt_avrcp_page_down_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_page_down_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_PAGE_DOWN)
```

Description

This is macro bt_avrcp_page_down_click.

bt_avrcp_page_down_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_page_down_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_PAGE_DOWN)
```

Description

This is macro bt_avrcp_page_down_press.

bt_avrcp_page_up_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_page_up_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_PAGE_UP)
```

Description

This is macro bt_avrcp_page_up_click.

bt_avrcp_page_up_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_page_up_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_PAGE_UP)
```

Description

This is macro bt_avrcp_page_up_press.

bt_avrcp_pause_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_pause_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_PAUSE)
```

Description

This is macro bt_avrcp_pause_click.

bt_avrcp_pause_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_pause_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_PAUSE)
```

Description

This is macro bt_avrcp_pause_press.

bt_avrcp_play_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_play_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_PLAY)
```

Description

This is macro bt_avrcp_play_click.

bt_avrcp_play_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_play_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_PLAY)
```

Description

This is macro bt_avrcp_play_press.

bt_avrcp_power_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_power_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_POWER_TOGGLE)
```

Description

This is macro bt_avrcp_power_click.

bt_avrcp_power_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_power_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_POWER_TOGGLE)
```

Description

This is macro bt_avrcp_power_press.

bt_avrcp_previous_channel_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_previous_channel_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_PREVIOUS_CHANNEL)
```

Description

This is macro bt_avrcp_previous_channel_click.

bt_avrcp_previous_channel_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_previous_channel_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_PREVIOUS_CHANNEL)
```

Description

This is macro bt_avrcp_previous_channel_press.

bt_avrcp_record_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_record_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_RECORD)
```

Description

This is macro bt_avrcp_record_click.

bt_avrcp_record_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_record_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_RECORD)
```

Description

This is macro bt_avrcp_record_press.

bt_avrcp_rewind_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_rewind_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_REWIND)
```

Description

This is macro bt_avrcp_rewind_click.

bt_avrcp_rewind_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_rewind_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_REWIND)
```

Description

This is macro bt_avrcp_rewind_press.

bt_avrcp_right_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_RIGHT)
```

Description

This is macro bt_avrcp_right_click.

bt_avrcp_right_down_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_down_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_RIGHT_DOWN)
```

Description

This is macro bt_avrcp_right_down_click.

bt_avrcp_right_down_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_down_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_RIGHT_DOWN)
```

Description

This is macro bt_avrcp_right_down_press.

bt_avrcp_right_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_RIGHT)
```

Description

This is macro bt_avrcp_right_press.

bt_avrcp_right_up_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_up_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_RIGHT_UP)
```

Description

This is macro bt_avrcp_right_up_click.

bt_avrcp_right_up_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_up_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_RIGHT_UP)
```

Description

This is macro bt_avrcp_right_up_press.

bt_avrcp_root_menu_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_root_menu_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_ROOT_MENU)
```

Description

This is macro bt_avrcp_root_menu_click.

bt_avrcp_root_menu_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_root_menu_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_ROOT_MENU)
```

Description

This is macro bt_avrcp_root_menu_press.

bt_avrcp_select_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_select_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_SELECT)
```

Description

This is macro bt_avrcp_select_click.

bt_avrcp_select_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_select_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_SELECT)
```

Description

Panel operations

bt_avrcp_setup_menu_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_setup_menu_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_SETUP_MENU)
```

Description

This is macro bt_avrcp_setup_menu_click.

bt_avrcp_setup_menu_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_setup_menu_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_SETUP_MENU)
```

Description

This is macro bt_avrcp_setup_menu_press.

bt_avrcp_sound_select_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_sound_select_click(channel) bt_avrcp_send_button_click(channel,
AVC_PANEL_OPID_SOUND_SELECT)
```

Description

This is macro bt_avrcp_sound_select_click.

bt_avrcp_sound_select_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_sound_select_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_SOUND_SELECT)
```

Description

This is macro bt_avrcp_sound_select_press.

bt_avrcp_stop_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_stop_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_STOP)
```

Description

This is macro bt_avrcp_stop_click.

bt_avrcp_stop_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_stop_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_STOP)
```

Description

This is macro bt_avrcp_stop_press.

bt_avrcp_subpicture_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_subpicture_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_SUBPICTURE)
```

Description

This is macro bt_avrcp_subpicture_click.

bt_avrcp_subpicture_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_subpicture_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_SUBPICTURE)
```

Description

This is macro bt_avrcp_subpicture_press.

bt_avrcp_up_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_up_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_UP)
```

Description

This is macro bt_avrcp_up_click.

bt_avrcp_up_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_up_press(channel) bt_avrcp_send_press_panel_control(channel, AVC_PANEL_OPID_UP)
```

Description

This is macro bt_avrcp_up_press.

bt_avrcp_volume_down_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_volume_down_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_VOLUME_DOWN)
```

Description

This is macro bt_avrcp_volume_down_click.

bt_avrcp_volume_down_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_volume_down_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_VOLUME_DOWN)
```

Description

This is macro bt_avrcp_volume_down_press.

bt_avrcp_volume_up_click Macro

File

avrcp_command.h

C

```
#define bt_avrcp_volume_up_click(channel) bt_avrcp_send_button_click(channel, AVC_PANEL_OPID_VOLUME_UP)
```

Description

This is macro bt_avrcp_volume_up_click.

bt_avrcp_volume_up_press Macro

File

avrcp_command.h

C

```
#define bt_avrcp_volume_up_press(channel) bt_avrcp_send_press_panel_control(channel,
AVC_PANEL_OPID_VOLUME_UP)
```

Description

This is macro bt_avrcp_volume_up_press.

bt_avrcp_0_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_0_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_0)
```

Description

This is macro bt_avrcp_0_release.

bt_avrcp_1_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_1_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_1)
```

Description

This is macro bt_avrcp_1_release.

bt_avrcp_2_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_2_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_2)
```

Description

This is macro bt_avrcp_2_release.

bt_avrcp_3_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_3_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_3)
```

Description

This is macro bt_avrcp_3_release.

bt_avrcp_4_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_4_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_4)
```

Description

This is macro bt_avrcp_4_release.

bt_avrcp_5_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_5_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_5)
```

Description

This is macro bt_avrcp_5_release.

bt_avrcp_6_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_6_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_6)
```

Description

This is macro bt_avrcp_6_release.

bt_avrcp_7_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_7_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_7)
```

Description

This is macro bt_avrcp_7_release.

bt_avrcp_8_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_8_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_8)
```

Description

This is macro bt_avrcp_8_release.

bt_avrcp_9_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_9_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_9)
```

Description

This is macro bt_avrcp_9_release.

bt_avrcp_angle_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_angle_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_ANGLE)
```

Description

This is macro bt_avrcp_angle_release.

bt_avrcp_backward_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_backward_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_BACKWARD)
```

Description

This is macro bt_avrcp_backward_release.

bt_avrcp_channel_down_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_channel_down_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_CHANNEL_DOWN)
```

Description

This is macro bt_avrcp_channel_down_release.

bt_avrcp_channel_up_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_channel_up_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_CHANNEL_UP)
```

Description

This is macro bt_avrcp_channel_up_release.

bt_avrcp_clear_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_clear_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_CLEAR)
```

Description

This is macro bt_avrcp_clear_release.

bt_avrcp_content_menu_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_content_menu_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_CONTENTS_MENU)
```

Description

This is macro bt_avrcp_content_menu_release.

bt_avrcp_display_info_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_display_info_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_DISPLAY_INFORMATION)
```

Description

This is macro bt_avrcp_display_info_release.

bt_avrcp_dot_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_dot_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_DOT)
```

Description

This is macro bt_avrcp_dot_release.

bt_avrcp_down_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_down_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_DOWN)
```

Description

This is macro bt_avrcp_down_release.

bt_avrcp_eject_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_eject_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_EJECT)
```

Description

This is macro bt_avrcp_eject_release.

bt_avrcp_enter_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_enter_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_ENTER)
```

Description

This is macro bt_avrcp_enter_release.

bt_avrcp_exit_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_exit_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_EXIT)
```

Description

This is macro bt_avrcp_exit_release.

bt_avrcp_f1_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f1_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F1)
```

Description

This is macro bt_avrcp_f1_release.

bt_avrcp_f2_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f2_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F2)
```

Description

This is macro bt_avrcp_f2_release.

bt_avrcp_f3_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f3_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F3)
```

Description

This is macro bt_avrcp_f3_release.

bt_avrcp_f4_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f4_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F4)
```

Description

This is macro bt_avrcp_f4_release.

bt_avrcp_f5_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f5_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F5)
```

Description

This is macro bt_avrcp_f5_release.

bt_avrcp_f6_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f6_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F6)
```

Description

This is macro bt_avrcp_f6_release.

bt_avrcp_f7_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f7_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F7)
```

Description

This is macro bt_avrcp_f7_release.

bt_avrcp_f8_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f8_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F8)
```

Description

This is macro bt_avrcp_f8_release.

bt_avrcp_f9_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_f9_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_F9)
```

Description

This is macro bt_avrcp_f9_release.

bt_avrcp_fast_forward_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_fast_forward_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_FAST_FORWARD)
```

Description

This is macro bt_avrcp_fast_forward_release.

bt_avrcp_favorite_menu_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_favorite_menu_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_FAVORIVE_MENU)
```

Description

This is macro bt_avrcp_favorite_menu_release.

bt_avrcp_forward_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_forward_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_FORWARD)
```

Description

This is macro bt_avrcp_forward_release.

bt_avrcp_help_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_help_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_HELP)
```

Description

This is macro bt_avrcp_help_release.

bt_avrcp_input_select_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_input_select_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_INPUT_SELECT)
```

Description

This is macro bt_avrcp_input_select_release.

bt_avrcp_left_down_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_down_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_LEFT_DOWN)
```

Description

This is macro bt_avrcp_left_down_release.

bt_avrcp_left_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_LEFT)
```

Description

This is macro bt_avrcp_left_release.

bt_avrcp_left_up_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_left_up_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_LEFT_UP)
```

Description

This is macro bt_avrcp_left_up_release.

bt_avrcp_mute_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_mute_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_MUTE)
```

Description

This is macro bt_avrcp_mute_release.

bt_avrcp_page_down_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_page_down_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_PAGE_DOWN)
```

Description

This is macro bt_avrcp_page_down_release.

bt_avrcp_page_up_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_page_up_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_PAGE_UP)
```

Description

This is macro bt_avrcp_page_up_release.

bt_avrcp_pause_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_pause_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_PAUSE)
```

Description

This is macro bt_avrcp_pause_release.

bt_avrcp_play_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_play_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_PLAY)
```

Description

This is macro bt_avrcp_play_release.

bt_avrcp_power_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_power_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_POWER_TOGGLE)
```

Description

This is macro bt_avrcp_power_release.

bt_avrcp_previous_channel_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_previous_channel_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_PREVIOUS_CHANNEL)
```

Description

This is macro bt_avrcp_previous_channel_release.

bt_avrcp_record_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_record_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_RECORD)
```

Description

This is macro bt_avrcp_record_release.

bt_avrcp_rewind_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_rewind_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_REWIND)
```

Description

This is macro bt_avrcp_rewind_release.

bt_avrcp_right_down_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_down_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_RIGHT_DOWN)
```

Description

This is macro bt_avrcp_right_down_release.

bt_avrcp_right_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_RIGHT)
```

Description

This is macro bt_avrcp_right_release.

bt_avrcp_right_up_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_right_up_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_RIGHT_UP)
```

Description

This is macro bt_avrcp_right_up_release.

bt_avrcp_root_menu_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_root_menu_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_ROOT_MENU)
```

Description

This is macro bt_avrcp_root_menu_release.

bt_avrcp_select_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_select_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_SELECT)
```

Description

This is macro bt_avrcp_select_release.

bt_avrcp_setup_menu_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_setup_menu_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_SETUP_MENU)
```

Description

This is macro bt_avrcp_setup_menu_release.

bt_avrcp_sound_select_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_sound_select_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_SOUND_SELECT)
```

Description

This is macro bt_avrcp_sound_select_release.

bt_avrcp_stop_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_stop_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_STOP)
```

Description

This is macro bt_avrcp_stop_release.

bt_avrcp_subpicture_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_subpicture_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_SUBPICTURE)
```

Description

This is macro bt_avrcp_subpicture_release.

bt_avrcp_up_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_up_release(channel) bt_avrcp_send_release_panel_control(channel, AVC_PANEL_OPID_UP)
```

Description

This is macro bt_avrcp_up_release.

bt_avrcp_volume_down_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_volume_down_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_VOLUME_DOWN)
```

Description

This is macro bt_avrcp_volume_down_release.

bt_avrcp_volume_up_release Macro

File

avrcp_command.h

C

```
#define bt_avrcp_volume_up_release(channel) bt_avrcp_send_release_panel_control(channel,
AVC_PANEL_OPID_VOLUME_UP)
```

Description

This is macro bt_avrcp_volume_up_release.

__AVRCP_CONFIG_EVENT_HANDLERS_H Macro

File

avrcp_config_event_handlers.h

C

```
#define __AVRCP_CONFIG_EVENT_HANDLERS_H
```

Description

This is macro __AVRCP_CONFIG_EVENT_HANDLERS_H.

AVRCP_COMMAND_HANDLER Macro

File

avrcp_config_event_handlers.h

C

```
#define AVRCP_COMMAND_HANDLER NULL
```

Description

This is macro AVRCP_COMMAND_HANDLER.

AVRCP_COMMAND_SENT_HANDLER Macro

File

avrcp_config_event_handlers.h

C

```
#define AVRCP_COMMAND_SENT_HANDLER NULL
```

Description

This is macro AVRCP_COMMAND_SENT_HANDLER.

AVRCP_RESPONSE_HANDLER Macro

File

avrcp_config_event_handlers.h

C

```
#define AVRCP_RESPONSE_HANDLER NULL
```

Description

This is macro AVRCP_RESPONSE_HANDLER.

AVRCP_RESPONSE_SENT_HANDLER Macro

File

avrcp_config_event_handlers.h

C

```
#define AVRCP_RESPONSE_SENT_HANDLER NULL
```

Description

This is macro AVRCP_RESPONSE_SENT_HANDLER.

GAP Functions

bt_gap_find_devices Function

File

gap.h

C

```
bt_bool bt_gap_find_devices(bt_device_t* devices, bt_byte max_devices, bt_byte length, bt_byte mode,
bt_find_devices_callback_fp callback, void* callback_param);
```

Description

This is function bt_gap_find_devices.

GAP Data Types and Constants

bt_device_t Structure

File

gap.h

C

```
typedef struct _bt_device_t {
  bt_bdaddr_t bdaddr;
  bt_byte* name;
  bt_byte max_name_len;
  bt_byte pg_scan_rpt_mode;
  bt_byte pg_scan_period_mode;
  bt_long cod;
  bt_int clock_offset;
  bt_byte rssi;
  bt_byte* eir;
  bt_byte eir_len;
  bt_byte max_eir_len;
} bt_device_t;
```

Description

This is type bt_device_t.

bt_find_devices_callback_fp Type

File

gap.h

C

```
typedef void (* bt_find_devices_callback_fp)(bt_byte count, void* callback_param);
```

Description

This is type bt_find_devices_callback_fp.

HCI Functions

bt_hci_add_param_bdaddr Function

File

hci.h

C

```
bt_bool bt_hci_add_param_bdaddr(bt_hci_command_p pcmd, const bt_bdaddr_t* pbdaddr);
```

Description

brief Add BD address parameter to an HCI command ingroup hci

bt_hci_add_param_byte Function

File

hci.h

C

```
bt_bool bt_hci_add_param_byte(bt_hci_command_p pcmd, bt_byte value);
```

Description

brief Add byte parameter to an HCI command ingroup hci

bt_hci_add_param_cod Function

File

hci.h

C

```
bt_bool bt_hci_add_param_cod(bt_hci_command_p pcmd, bt_long value);
```

Description

brief Add class of device parameter to an HCI command ingroup hci

bt_hci_add_param_int Function

File

hci.h

C

```
bt_bool bt_hci_add_param_int(bt_hci_command_p pcmd, bt_int value);
```

Description

brief Add int parameter to an HCI command ingroup hci

bt_hci_add_param_linkkey Function

File

hci.h

C

```
bt_bool bt_hci_add_param_linkkey(bt_hci_command_p pcmd, const bt_linkkey_t* linkkey);
```

Description

brief Add link key parameter to an HCI command ingroup hci

bt_hci_add_param_long Function

File

hci.h

C

```
bt_bool bt_hci_add_param_long(bt_hci_command_p pcmd, bt_long value);
```

Description

brief Add long parameter to an HCI command ingroup hci

bt_hci_add_param_string Function

File

hci.h

C

```
bt_bool bt_hci_add_param_string(bt_hci_command_p pcmd, const char * ps, bt_int len);
```

Description

brief Add string parameter to an HCI command ingroup hci

bt_hci_alloc_canned_command Function

File

hci.h

C

```
bt_hci_command_p bt_hci_alloc_canned_command(const bt_byte* canned_command, bt_hci_cmd_callback_fp
callback);
```

Description

brief Allocate and initialize an HCI command structure for a canned (pre-formatted) command. ingroup hci

bt_hci_alloc_command Function

File

hci.h

C

```
bt_hci_command_p bt_hci_alloc_command(bt_int opcode, bt_hci_cmd_callback_fp callback);
```

Description

brief Allocate and initialize an HCI command structure. ingroup hci

bt_hci_alloc_data_buffer Function

File

hci_data_buffer.h

C

```
bt_hci_data_p bt_hci_alloc_data_buffer();
```

Description

This is function bt_hci_alloc_data_buffer.

bt_hci_allocate_write_eir_command Function

File

hci_eir.h

C

```
bt_hci_command_p bt_hci_allocate_write_eir_command(bt_byte fec_required);
```

Description

This is function bt_hci_allocate_write_eir_command.

bt_hci_authenticate_ex Function

File

hci.h

C

```
bt_bool bt_hci_authenticate_ex(bt_hci_conn_state_t * pconn, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_authenticate_ex.

bt_hci_cancel_find_devices Function

File

hci.h

C

```
bt_bool bt_hci_cancel_find_devices(bt_hci_cmd_callback_fp cb);
```

Description

brief Stop inquiry

bt_hci_cancel_request_remote_name Function

File

hci.h

C

```
bt_bool bt_hci_cancel_request_remote_name(bt_bdaddr_p pbdaddr);
```

Description

brief Cancel remote device name request

bt_hci_cancel_send_acl_data Function

File

hci.h

C

```
void bt_hci_cancel_send_acl_data(bt_byte* data);
```

Description

brief Cancel sending data over ACL connection

bt_hci_connect Function

File

hci.h

C

```
bt_bool bt_hci_connect(bt_bdaddr_p dest, bt_uint packet_type, bt_byte pg_scan_rpt_mode, bt_byte
role_switch, bt_uint acl_config, bt_hci_connect_callback_fp callback, void* param);
```

Description

brief Connect to a remote device. ingroup hci

details This function tries to establish an HCI connection with a remote device specified by the Bluetooth address c dest. Upon completion, the callback function specified by the c callback parameter is called.

param dest Bluetooth address of the remote device. param packet_type param role_switch param acl_config param callback Pointer to a callback function that is called when the connect operation completes. param param Pointer to arbitrary data that is to be passed to the callback function.

return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_hci_connect_sco Function

File

hci.h

C

```
bt_bool bt_hci_connect_sco(bt_hci_conn_state_t* pconn, bt_ulong tx_bandwidth, bt_ulong rcv_bandwidth,
bt_uint max_latency, bt_uint content_format, bt_byte retransmission_effort, bt_uint packet_type,
```

```
bt_hci_connect_callback_fp cb, void * param);
```

Description

This is function bt_hci_connect_sco.

bt_hci_ctrl_register_data_listener Function

File

hci_ctrl_state.h

C

```
bt_bool bt_hci_ctrl_register_data_listener(bt_hci_ctrl_listener_t* listener, bt_byte listener_type);
```

Description

This is function bt_hci_ctrl_register_data_listener.

bt_hci_ctrl_register_listener Function

File

hci_ctrl_state.h

C

```
bt_bool bt_hci_ctrl_register_listener(bt_hci_ctrl_listener_t* listener);
```

Description

This is function bt_hci_ctrl_register_listener.

bt_hci_ctrl_unregister_listener Function

File

hci_ctrl_state.h

C

```
void bt_hci_ctrl_unregister_listener(bt_hci_ctrl_listener_t* listener);
```

Description

This is function bt_hci_ctrl_unregister_listener.

bt_hci_disconnect Function

File

hci.h

C

```
bt_bool bt_hci_disconnect(bt_hci_conn_state_t * pconn);
```

Description

brief Abort connection ingroup hci

bt_hci_evt_authentication_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_authentication_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_authentication_complete_handler.

bt_hci_evt_change_conn_link_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_change_conn_link_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_change_conn_link_complete_handler.

bt_hci_evt_command_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_command_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_command_complete_handler.

bt_hci_evt_command_status_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_command_status_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_command_status_handler.

bt_hci_evt_conn_packet_type_changed_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_conn_packet_type_changed_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_conn_packet_type_changed_handler.

bt_hci_evt_connection_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_connection_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_connection_complete_handler.

bt_hci_evt_connection_request_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_connection_request_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_connection_request_handler.

bt_hci_evt_data_buffer_overflow_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_data_buffer_overflow_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_data_buffer_overflow_handler.

bt_hci_evt_default_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_default_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_default_handler.

bt_hci_evt_disconnection_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_disconnection_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_disconnection_complete_handler.

bt_hci_evt_encryption_change_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_encryption_change_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_encryption_change_handler.

bt_hci_evt_extended_inquiry_result_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_extended_inquiry_result_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_extended_inquiry_result_handler.

bt_hci_evt_flow_specification_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_flow_specification_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_flow_specification_complete_handler.

bt_hci_evt_flush_occured_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_flush_occured_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_flush_occured_handler.

bt_hci_evt_hardware_error_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_hardware_error_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_hardware_error_handler.

bt_hci_evt_inquiry_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_inquiry_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_inquiry_complete_handler.

bt_hci_evt_inquiry_result_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_inquiry_result_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_inquiry_result_handler.

bt_hci_evt_inquiry_result_with_rssi_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_inquiry_result_with_rssi_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_inquiry_result_with_rssi_handler.

bt_hci_evt_link_key_notification_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_link_key_notification_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_link_key_notification_handler.

bt_hci_evt_link_key_request_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_link_key_request_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_link_key_request_handler.

bt_hci_evt_loopback_command_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_loopback_command_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_loopback_command_handler.

bt_hci_evt_master_link_key_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_master_link_key_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_master_link_key_complete_handler.

bt_hci_evt_max_slots_change_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_max_slots_change_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_max_slots_change_handler.

bt_hci_evt_mode_change_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_mode_change_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_mode_change_handler.

bt_hci_evt_num_of_completed_packets_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_num_of_completed_packets_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_num_of_completed_packets_handler.

bt_hci_evt_page_scan_repet_mode_change_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_page_scan_repet_mode_change_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_page_scan_repet_mode_change_handler.

bt_hci_evt_pin_code_request_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_pin_code_request_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_pin_code_request_handler.

bt_hci_evt_qos_setup_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_qos_setup_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_qos_setup_complete_handler.

bt_hci_evt_qos_violation_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_qos_violation_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_qos_violation_handler.

bt_hci_evt_read_clock_offset_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_read_clock_offset_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_read_clock_offset_complete_handler.

bt_hci_evt_read_rmt_ext_features_comp_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_read_rmt_ext_features_comp_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_read_rmt_ext_features_comp_handler.

bt_hci_evt_read_rmt_sup_features_comp_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_read_rmt_sup_features_comp_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_read_rmt_sup_features_comp_handler.

bt_hci_evt_read_rmt_version_info_comp_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_read_rmt_version_info_comp_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_read_rmt_version_info_comp_handler.

bt_hci_evt_remote_name_request_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_remote_name_request_complete_handler(bt_hci_event_p pevt, bt_uint params_len);
```

Description

This is function bt_hci_evt_remote_name_request_complete_handler.

bt_hci_evt_return_link_keys_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_return_link_keys_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_return_link_keys_handler.

bt_hci_evt_role_change_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_role_change_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_role_change_handler.

bt_hci_evt_synch_connection_changed_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_synch_connection_changed_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_synch_connection_changed_handler.

bt_hci_evt_synch_connection_complete_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_evt_synch_connection_complete_handler(bt_hci_event_p pevt);
```

Description

This is function bt_hci_evt_synch_connection_complete_handler.

bt_hci_exit_park_state Function

File

hci.h

C

```
bt_bool bt_hci_exit_park_state(bt_hci_conn_state_t * pconn, bt_hci_cmd_callback_fp cb);
```

Description

brief Exit park state

bt_hci_exit_sniff_mode_ex Function

File

hci.h

C

```
bt_bool bt_hci_exit_sniff_mode_ex(bt_hci_conn_state_t * pconn, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_exit_sniff_mode_ex.

bt_hci_find_devices Function

File

hci.h

C

```
bt_bool bt_hci_find_devices(bt_byte max_responses, bt_byte length, bt_hci_inquiry_callback_fp cb);
```

Description

brief Start inquiry

bt_hci_find_devices_ex Function

File

hci.h

C

```
bt_bool bt_hci_find_devices_ex(bt_byte max_responses, bt_byte length, bt_byte mode,
bt_hci_inquiry_callback_fp cb);
```

Description

This is function bt_hci_find_devices_ex.

bt_hci_free_command Function

File

hci.h

C

```
void bt_hci_free_command(bt_hci_command_p cmd);
```

Description

brief Free HCI command. ingroup hci

bt_hci_free_data_buffer Function

File

hci_data_buffer.h

C

```
void bt_hci_free_data_buffer(bt_hci_data_p p);
```

Description

This is function bt_hci_free_data_buffer.

bt_hci_get_evt_param_bdaddr Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_bdaddr(bt_hci_event_p pevt, bt_bdaddr_p pvalue, bt_int_p poffset);
```

Description

brief Get bd address parameter from HCI event ingroup hci

bt_hci_get_evt_param_byte Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_byte(bt_hci_event_p pevt, bt_byte* pvalue, bt_int* poffset);
```

Description

brief Get byte parameter from HCI event ingroup hci

bt_hci_get_evt_param_devclass Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_devclass(bt_hci_event_p pevt, bt_long_p pvalue, bt_int_p poffset);
```

Description

brief Get class of device parameter from HCI event ingroup hci

bt_hci_get_evt_param_int Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_int(bt_hci_event_p pevt, bt_int* pvalue, bt_int* poffset);
```

Description

brief Get int parameter from HCI event ingroup hci

bt_hci_get_evt_param_linkkey Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_linkkey(bt_hci_event_p pevt, bt_linkkey_p pvalue, bt_int_p poffset);
```

Description

brief Get link key parameter from HCI event ingroup hci

bt_hci_get_evt_param_long Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_long(bt_hci_event_p pevt, bt_long* pvalue, bt_int* poffset);
```

Description

brief Get long parameter from HCI event ingroup hci

bt_hci_get_evt_param_uint Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_uint(bt_hci_event_p pevt, bt_uint* pvalue, bt_int* poffset);
```

Description

brief Get unsigned int parameter from HCI event ingroup hci

bt_hci_get_evt_param_ulong Function

File

hci.h

C

```
bt_bool bt_hci_get_evt_param_ulong(bt_hci_event_p pevt, bt_ulong* pvalue, bt_int* poffset);
```

Description

brief Get unsigned long parameter from HCI event ingroup hci

bt_hci_get_last_cmd_status Function

File

hci.h

C

```
bt_byte bt_hci_get_last_cmd_status();
```

Description

This is function bt_hci_get_last_cmd_status.

bt_hci_get_param_bdaddr Function

File

hci.h

C

```
bt_bool bt_hci_get_param_bdaddr(bt_hci_command_p pcm, bt_bdaddr_p pvalue, bt_int_p poffset);
```

Description

brief Get BD address parameter from HCI command ingroup hci

bt_hci_get_param_byte Function

File

hci.h

C

```
bt_bool bt_hci_get_param_byte(bt_hci_command_p pcmd, bt_byte* pvalue, bt_int* offset);
```

Description

brief Get byte parameter from HCI command ingroup hci

bt_hci_get_param_int Function

File

hci.h

C

```
bt_bool bt_hci_get_param_int(bt_hci_command_p pcmd, bt_int* pvalue, bt_int* poffset);
```

Description

brief Get int parameter from HCI command ingroup hci

bt_hci_get_param_linkkey Function

File

hci.h

C

```
bt_bool bt_hci_get_param_linkkey(bt_hci_command_p pcmd, bt_byte_p pvalue, bt_int_p poffset);
```

Description

brief Get link key parameter from HCI command ingroup hci

bt_hci_get_param_long Function

File

hci.h

C

```
bt_bool bt_hci_get_param_long(bt_hci_command_p pcmd, bt_long* pvalue, bt_int* poffset);
```

Description

brief Get long parameter from HCI command ingroup hci

bt_hci_get_recv_buffer Function

File

hci.h

C

```
bt_byte* bt_hci_get_recv_buffer();
```

Description

This is function bt_hci_get_recv_buffer.

bt_hci_get_recv_buffer_len Function

File

hci.h

C

```
bt_int bt_hci_get_recv_buffer_len();
```

Description

This is function bt_hci_get_recv_buffer_len.

bt_hci_get_send_buffer Function

File

hci.h

C

```
bt_byte* bt_hci_get_send_buffer();
```

Description

This is function bt_hci_get_send_buffer.

bt_hci_get_send_buffer_len Function

File

hci.h

C

```
bt_int bt_hci_get_send_buffer_len();
```

Description

This is function bt_hci_get_send_buffer_len.

bt_hci_init Function

File

hci.h

C

```
void bt_hci_init();
```

Description

brief Initialize the HCI layer. ingroup hci

This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt_sys_init.

This function essentially calls bt_hci_init_ex(HCI_LINK_POLICY_ENABLE_ALL) so all link policy setting are enabled.

bt_hci_init_data_buffers Function

File

hci_data_buffer.h

C

```
void bt_hci_init_data_buffers();
```

Description

This is function bt_hci_init_data_buffers.

bt_hci_init_data_queues Function

File

hci_data_queue.h

C

```
void bt_hci_init_data_queues();
```

Description

This is function bt_hci_init_data_queues.

bt_hci_init_linkkey_buffers Function

File

hci_linkkey_buffer.h

C

```
void bt_hci_init_linkkey_buffers();
```

Description

This is function bt_hci_init_linkkey_buffers.

bt_hci_le_add_device_to_white_list Function

File

hci_le.h

C

```
bt_bool bt_hci_le_add_device_to_white_list(bt_byte address_type, bt_bdaddr_t* address,
bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_add_device_to_white_list.

bt_hci_le_advertising_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_add(bt_byte data_type, const bt_byte* data, bt_byte length, bt_hci_command_p
pcmd);
```

Description

This is function bt_hci_le_advertising_add.

bt_hci_le_advertising_add_local_name Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_add_local_name(const char* local_name, bt_hci_command_t* pcmd);
```

Description

This is function bt_hci_le_advertising_add_local_name.

bt_hci_le_advertising_device_id_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_device_id_add(bt_uint vendor_id_source, bt_uint vendor_id, bt_uint
product_id, bt_uint version, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_device_id_add.

bt_hci_le_advertising_flags_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_flags_add(bt_byte flags, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_flags_add.

bt_hci_le_advertising_get_local_name Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_get_local_name(const bt_byte* data, bt_byte data_len, const bt_byte**
local_name, bt_byte* name_len, bt_byte* name_type);
```

Description

This is function bt_hci_le_advertising_get_local_name.

bt_hci_le_advertising_tx_power_level_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_tx_power_level_add(bt_byte tx_power_level, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_tx_power_level_add.

bt_hci_le_advertising_uuid128_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_uuid128_add(bt_byte data_type, const bt_uuid_t* uuid_list, bt_byte
uuid_list_size, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_uuid128_add.

bt_hci_le_advertising_uuid16_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_uuid16_add(bt_byte data_type, const bt_uint* uuid_list, bt_byte
uuid_list_size, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_uuid16_add.

bt_hci_le_advertising_uuid32_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_uuid32_add(bt_byte data_type, const bt_uuid32* uuid_list, bt_byte
uuid_list_size, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_uuid32_add.

bt_hci_le_advertising_vendor_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_advertising_vendor_add(bt_uint vendor_id, bt_byte* data, bt_byte data_len,
bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_advertising_vendor_add.

bt_hci_le_allocate_set_advertising_data_command Function

File

hci_le.h

C

```
bt_hci_command_t* bt_hci_le_allocate_set_advertising_data_command(bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_allocate_set_advertising_data_command.

bt_hci_le_cancel_find_devices Function

File

hci_le.h

C

```
bt_bool bt_hci_le_cancel_find_devices();
```

Description

This is function bt_hci_le_cancel_find_devices.

bt_hci_le_clear_white_list Function

File

hci_le.h

C

```
bt_bool bt_hci_le_clear_white_list(bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_clear_white_list.

bt_hci_le_connect_ex Function

File

hci_le.h

C

```
bt_bool bt_hci_le_connect_ex(bt_uint scan_interval, bt_uint scan_window, bt_byte filter_policy, bt_byte
peer_address_type, bt_bdaddr_t* peer_address, bt_byte own_address_type, bt_uint conn_interval_min, bt_uint
conn_interval_max, bt_uint conn_latency, bt_uint supervision_timeout, bt_uint min_ce_length, bt_uint
max_ce_length, bt_uint acl_config, bt_hci_connect_callback_fp cb, void * param);
```

Description

This is function bt_hci_le_connect_ex.

bt_hci_le_enable Function

File

hci_le.h

C

```
bt_bool bt_hci_le_enable(bt_bool enabled, bt_bool simultaneous_enabled, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_enable.

bt_hci_le_encrypt Function

File

hci_le.h

C

```
bt_bool bt_hci_le_encrypt(const bt_byte* key, bt_byte key_len, const bt_byte* data, bt_byte data_len,
bt_hci_cmd_callback_fp cb, void* param);
```

Description

This is function bt_hci_le_encrypt.

bt_hci_le_find_devices Function

File

hci_le.h

C

```
bt_bool bt_hci_le_find_devices(bt_byte filter_duplicates, bt_hci_le_scan_callback_fp cb, void* param);
```

Description

This is function bt_hci_le_find_devices.

bt_hci_le_get_connect_parameters Function

File

hci_le.h

C

```
bt_bool bt_hci_le_get_connect_parameters(bt_hci_le_connect_parameters_t* params);
```

Description

This is function bt_hci_le_get_connect_parameters.

bt_hci_le_init Function

File

hci_private.h

C

```
void bt_hci_le_init(bt_hci_le_ctrl_state_t* le_ctrl_state);
```

Description

This is function bt_hci_le_init.

bt_hci_le_ltk_negative_reply Function

File

hci_le.h

C

```
bt_bool bt_hci_le_ltk_negative_reply(struct _bt_hci_conn_state_s* pconn, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_ltk_negative_reply.

bt_hci_le_ltk_reply Function

File

hci_le.h

C

```
bt_bool bt_hci_le_ltk_reply(struct _bt_hci_conn_state_s* pconn, const bt_byte* long_term_key,
bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_ltk_reply.

bt_hci_le_rand Function

File

hci_le.h

C

```
bt_bool bt_hci_le_rand(bt_hci_cmd_callback_fp cb, void* param);
```

Description

This is function bt_hci_le_rand.

bt_hci_le_read_channel_map Function

File

hci_le.h

C

```
bt_bool bt_hci_le_read_channel_map(struct _bt_hci_conn_state_s* pconn, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_read_channel_map.

bt_hci_le_read_remote_used_features Function

File

hci_le.h

C

```
bt_bool bt_hci_le_read_remote_used_features(struct _bt_hci_conn_state_s* pconn, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_read_remote_used_features.

bt_hci_le_read_support Function

File

hci_le.h

C

```
bt_bool bt_hci_le_read_support(bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_read_support.

bt_hci_le_read_white_list_size Function

File

hci_le.h

C

```
bt_bool bt_hci_le_read_white_list_size(bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_read_white_list_size.

bt_hci_le_receiver_test Function

File

hci_le.h

C

```
bt_bool bt_hci_le_receiver_test(bt_byte frequency, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_receiver_test.

bt_hci_le_remove_device_from_white_list Function

File

hci_le.h

C

```
bt_bool bt_hci_le_remove_device_from_white_list(bt_byte address_type, bt_bdaddr_t* address,
bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_remove_device_from_white_list.

bt_hci_le_set_adevrtising_enable_ex Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_adevrtising_enable_ex(bt_byte adv_enable, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_le_set_adevrtising_enable_ex.

bt_hci_le_set_advertising_parameters Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_advertising_parameters(bt_uint adv_interval_min, bt_uint adv_interval_max, bt_byte
adv_type, bt_byte own_address_type, bt_byte direct_address_type, bt_bdaddr_t* direct_address, bt_byte
adv_channel_map, bt_byte adv_filter_policy, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_set_advertising_parameters.

bt_hci_le_set_connect_parameters Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_connect_parameters(bt_hci_le_connect_parameters_t* params);
```

Description

This is function bt_hci_le_set_connect_parameters.

bt_hci_le_set_host_channel_classification Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_host_channel_classification(const bt_byte* channel_map, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_set_host_channel_classification.

bt_hci_le_set_random_address Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_random_address(bt_bdaddr_t* bdaddr, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_set_random_address.

bt_hci_le_set_scan_enable Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_scan_enable(bt_byte scan_enable, bt_byte filter_duplicates, bt_hci_cmd_callback_fp
cb);
```

Description

This is function bt_hci_le_set_scan_enable.

bt_hci_le_set_scan_parameters Function

File

hci_le.h

C

```
bt_bool bt_hci_le_set_scan_parameters(bt_byte scan_type, bt_uint scan_interval, bt_uint scan_window,
bt_byte own_address_type, bt_byte filter_policy, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_set_scan_parameters.

bt_hci_le_start_encryption Function

File

hci_le.h

C

```
bt_bool bt_hci_le_start_encryption(struct _bt_hci_conn_state_s* pconn, const bt_byte* random_number,
bt_uint encrypted_diversifier, const bt_byte* long_term_key, bt_hci_cmd_callback_fp cb, void* param);
```

Description

This is function bt_hci_le_start_encryption.

bt_hci_le_supported Function

File

hci_le.h

C

```
bt_bool bt_hci_le_supported();
```

Description

This is function bt_hci_le_supported.

bt_hci_le_test_end Function

File

hci_le.h

C

```
bt_bool bt_hci_le_test_end(bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_test_end.

bt_hci_le_transmitter_test Function

File

hci_le.h

C

```
bt_bool bt_hci_le_transmitter_test(bt_byte frequency, bt_byte test_data_len, bt_byte packet_payload,
bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_transmitter_test.

bt_hci_le_update_connection Function

File

hci_le.h

C

```
bt_bool bt_hci_le_update_connection(struct _bt_hci_conn_state_s* pconn, bt_uint conn_interval_min, bt_uint
conn_interval_max, bt_uint conn_latency, bt_uint supervision_timeout, bt_uint min_ce_length, bt_uint
max_ce_length, bt_hci_cmd_callback_fp cb, void * param);
```

Description

This is function bt_hci_le_update_connection.

bt_hci_le_write_support Function

File

hci_le.h

C

```
bt_bool bt_hci_le_write_support(bt_bool enabled, bt_bool simultaneous_enabled, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_write_support.

bt_hci_listen Function

File

hci.h

C

```
bt_bool bt_hci_listen(bt_hci_connect_callback_fp cb, void* param);
```

Description

brief Listen for incoming connections. ingroup hci

details

param callback Pointer to a callback function that is called when a new incoming connection has been established. param param Pointer to arbitrary data that is to be passed to the callback function.

return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_hci_listen_sco Function

File

hci.h

C

```
bt_bool bt_hci_listen_sco(bt_ulong tx_bandwidth, bt_ulong rcv_bandwidth, bt_uint max_latency, bt_uint
content_format, bt_byte retransmission_effort, bt_uint packet_type, bt_hci_connect_callback_fp cb, void *
param);
```

Description

This is function bt_hci_listen_sco.

bt_hci_param_eir_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_add(bt_byte data_type, const bt_byte* data, bt_byte length, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_eir_add.

bt_hci_param_eir_device_id_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_device_id_add(bt_uint vendor_id_source, bt_uint vendor_id, bt_uint product_id,
bt_uint version, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_eir_device_id_add.

bt_hci_param_eir_local_name_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_local_name_add(const char* local_name, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_eir_local_name_add.

bt_hci_param_eir_uuid128_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_uuid128_add(bt_byte data_type, const bt_uuid_t* uuid_list, bt_byte uuid_list_size,
bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_eir_uuid128_add.

bt_hci_param_eir_uuid16_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_uuid16_add(bt_byte data_type, const bt_uint* uuid_list, bt_byte uuid_list_size,
bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_eir_uuid16_add.

bt_hci_param_eir_uuid32_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_uuid32_add(bt_byte data_type, const bt_uuid32* uuid_list, bt_byte uuid_list_size,
bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_eir_uuid32_add.

bt_hci_param_eir_vendor_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_eir_vendor_add(bt_uint vendor_id, bt_byte* data, bt_byte data_len, bt_hci_command_p
pcmd);
```

Description

This is function bt_hci_param_eir_vendor_add.

bt_hci_param_tx_power_level_add Function

File

hci_eir.h

C

```
bt_bool bt_hci_param_tx_power_level_add(bt_byte tx_power_level, bt_hci_command_p pcmd);
```

Description

This is function bt_hci_param_tx_power_level_add.

bt_hci_park_state Function

File

hci.h

C

```
bt_bool bt_hci_park_state(bt_hci_conn_state_t * pconn, bt_int min_interval, bt_int max_interval,
bt_hci_cmd_callback_fp cb);
```

Description

brief Put local device to park state

bt_hci_read_inquiry_mode Function

File

hci.h

C

```
bt_bool bt_hci_read_inquiry_mode(bt_hci_read_inquiry_mode_callback_fp cb);
```

Description

brief Get current inquiry mode

bt_hci_read_inquiry_scan_activity Function

File

hci.h

C

```
bt_bool bt_hci_read_inquiry_scan_activity(bt_hci_read_inquiry_scan_activity_callback_fp cb);
```

Description

brief Get current inquiry scan activity configuration

bt_hci_read_inquiry_scan_type Function

File

hci.h

C

```
bt_bool bt_hci_read_inquiry_scan_type(bt_hci_read_inquiry_scan_type_callback_fp cb);
```

Description

brief Get current inquiry scan type

bt_hci_read_page_scan_activity Function

File

hci.h

C

```
bt_bool bt_hci_read_page_scan_activity(bt_hci_read_page_scan_activity_callback_fp cb);
```

Description

brief Get current page scan activity configuration

bt_hci_read_page_scan_period_mode Function

File

hci.h

C

```
bt_bool bt_hci_read_page_scan_period_mode(bt_hci_read_page_scan_period_mode_callback_fp cb);
```

Description

brief Get current page scan period mode

bt_hci_read_page_scan_type Function

File

hci.h

C

```
bt_bool bt_hci_read_page_scan_type(bt_hci_read_page_scan_type_callback_fp cb);
```

Description

brief Get current page scan type

bt_hci_read_page_timeout Function

File

hci.h

C

```
bt_bool bt_hci_read_page_timeout(bt_hci_read_page_timeout_callback_fp cb);
```

Description

brief Get current page timeout

bt_hci_register_listener Function

File

hci_conn_state.h

C

```
bt_bool bt_hci_register_listener(bt_hci_conn_state_t* pconn, bt_hci_listener_t* listener);
```

Description

This is function bt_hci_register_listener.

bt_hci_reject_pin_code Function

File

hci.h

C

```
void bt_hci_reject_pin_code(bt_bdaddr_p pbdaddr_remote);
```

Description

This is function bt_hci_reject_pin_code.

bt_hci_request_remote_name Function

File

hci.h

C

```
bt_bool bt_hci_request_remote_name(bt_bdaddr_p pbdaddr, bt_byte pg_scan_rpt_mode, bt_int clock_offset,
bt_hci_request_remote_name_callback_fp cb);
```

Description

brief Request remote device's name

bt_hci_role_change_ex Function

File

hci.h

C

```
bt_bool bt_hci_role_change_ex(bt_bdaddr_p pbdaddr, bt_byte role, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

brief Change local device's role

bt_hci_send_acl_data Function

File

hci.h

C

```
bt_bool bt_hci_send_acl_data(bt_hci_hconn_t h, bt_byte_p pdata, bt_int len, bt_hci_data_callback_fp cb,
void * param);
```

Description

brief Send data over ACL connection

bt_hci_send_cmd Function

File

hci.h

C

```
bt_bool bt_hci_send_cmd(bt_hci_command_p pcmd);
```

Description

brief Send a command to local device

bt_hci_send_linkkey Function

File

hci.h

C

```
bt_bool bt_hci_send_linkkey(const bt_bdaddr_t* bdaddr_remote, const bt_linkkey_t* link_key,
bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_send_linkkey.

bt_hci_send_sco_data Function

File

hci.h

C

```
bt_bool bt_hci_send_sco_data(bt_hci_hconn_t h, bt_byte_p pdata, bt_uint len, bt_hci_data_callback_fp cb,
void * param);
```

Description

brief Send data over SCO connection

bt_hci_set_encryption_ex Function

File

hci.h

C

```
bt_bool bt_hci_set_encryption_ex(bt_hci_conn_state_t* pconn, bt_byte encryption_enable,
bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

brief Set connection encryption

bt_hci_set_event_listener Function

File

hci_ctrl_state.h

C

```
void bt_hci_set_event_listener(bt_hci_event_listener_fp callback);
```

Description

listener is triggered by the following events: HCI_EVT_AUTHENTICATION_COMPLETE HCI_EVT_CONNECTION_COMPLETE HCI_EVT_DISCONNECTION_COMPLETE HCI_EVT_ROLE_CHANGE HCI_EVT_MODE_CHANGE

bt_hci_set_incoming_connection_role Function

File

hci.h

C

```
void bt_hci_set_incoming_connection_role(bt_byte role);
```

Description

This is function bt_hci_set_incoming_connection_role.

bt_hci_set_scan Function

File

hci.h

C

```
bt_bool bt_hci_set_scan(bt_bool discoverable, bt_ulong discoverable_period, bt_bool connectable, bt_ulong
connectable_period, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_set_scan.

bt_hci_sniff_mode_ex Function

File

hci.h

C

```
bt_bool bt_hci_sniff_mode_ex(bt_hci_conn_state_t * pconn, bt_int min_interval, bt_int max_interval, bt_int
attempt_slots, bt_int timeout, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_sniff_mode_ex.

bt_hci_sniff_subrating_ex Function

File

hci.h

C

```
bt_bool bt_hci_sniff_subrating_ex(bt_hci_conn_state_t* pconn, bt_uint max_latency, bt_uint
min_remote_timeout, bt_uint min_local_timeout, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_sniff_subrating_ex.

bt_hci_start Function

File

hci.h

C

```
bt_bool bt_hci_start(bt_hci_start_callback_fp callback, void* callback_param, bt_byte enable_scan);
```

Description

brief Start HCI layer. ingroup hci

details This function starts the HCI layer of the stack. Starting the HCI layer consists essentially of two steps:

1. Make the HCI transport receive packets from the controller. This results in a call to bt_oem_recv.
2. Reset and configure the controller.

Upon completion of controller initialization the callback function passed in the c callback parameter is called.

param callback Completion callback. Called when controller initialization is complete. param callback_param A pointer to arbitrary data to be passed to the c callback callback. param enable_scan This is a bitmask that defines which scans are enabled during the controller configuration. This value can be a combination of the following values: li HCI_SCAN_INQUIRY (the controller is discoverable) li HCI_SCAN_PAGE (the controller is connectable)

return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_hci_start_no_init Function

File

hci.h

C

```
void bt_hci_start_no_init();
```

Description

brief Start HCI layer without controller configuration. ingroup hci

details This function is similar to bt_hci_start but unlike the former it does not perform the controller configuration. I.e., bt_hci_start_no_init simply calls the HCI transport and makes it receive packets from the controller. The main purpose of this function is make the HCI transport ready to exchange packets if controller needs some vendor specific configuration before it can be used with the stack. E.g., controllers based on CRS8811 chip need loading various values that configure its operating mode using CSR's proprietary protocol. So the application after configuring the HCI transport would call bt_hci_init(), bt_hci_start_no_init() and then load configuration values. Once the vendor specific configuration is done, the application will re-initialize the HCI transport and perform full start of the stack with bt_sys_init() and bt_sys_start().

bt_hci_stop Function

File

hci.h

C

```
void bt_hci_stop(bt_hci_stop_callback_fp callback, void* callback_param);
```

Description

brief Stop HCI layer. ingroup hci

details This function makes the HCI layer inoperable. After this call the application must perform the full reset of the HCI transport and stack.

param callback Completion callback. Called when the HCI layer has been stopped. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

bt_hci_transport_recv_packet Function

File

hci_transport.h

C

```
void bt_hci_transport_recv_packet(bt_byte* buffer, bt_uint len, bt_hci_transport_recv_packet_callback_fp
callback);
```

Description

This is function bt_hci_transport_recv_packet.

bt_hci_transport_send_cmd Function

File

hci_transport.h

C

```
void bt_hci_transport_send_cmd(const bt_byte* buffer, bt_uint len, bt_hci_transport_send_packet_callback_fp
callback);
```

Description

deprecated

bt_hci_transport_send_data Function

File

hci_transport.h

C

```
void bt_hci_transport_send_data(bt_byte* buffer, bt_uint len, bt_hci_transport_send_packet_callback_fp
callback);
```

Description

This is function bt_hci_transport_send_data.

bt_hci_transport_send_packet Function

File

hci_transport.h

C

```
void bt_hci_transport_send_packet(const bt_byte* buffer, bt_uint len,
bt_hci_transport_send_packet_callback_fp callback);
```

Description

This is function bt_hci_transport_send_packet.

bt_hci_transport_set_transport Function

File

hci_transport.h

C

```
void bt_hci_transport_set_transport(const hci_transport_t* transport);
```

Description

This is function bt_hci_transport_set_transport.

bt_hci_unregister_listener Function

File

hci_conn_state.h

C

```
void bt_hci_unregister_listener(bt_hci_conn_state_t* pconn, bt_hci_listener_t* listener);
```

Description

This is function bt_hci_unregister_listener.

bt_hci_write_default_link_policy_settings Function

File

hci.h

C

```
bt_bool bt_hci_write_default_link_policy_settings(bt_uint settings, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_write_default_link_policy_settings.

bt_hci_write_eir Function

File

hci_eir.h

C

```
bt_bool bt_hci_write_eir(bt_hci_command_p pcmd, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_write_eir.

bt_hci_write_inquiry_mode Function

File

hci.h

C

```
bt_bool bt_hci_write_inquiry_mode(bt_byte scanMode, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure inquiry mode

bt_hci_write_inquiry_scan_activity Function

File

hci.h

C

```
bt_bool bt_hci_write_inquiry_scan_activity(bt_uint interval, bt_uint window, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure inquiry scan activity

bt_hci_write_inquiry_scan_type Function

File

hci.h

C

```
bt_bool bt_hci_write_inquiry_scan_type(bt_byte scanType, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure inquiry scan type

bt_hci_write_local_name Function

File

hci.h

C

```
bt_bool bt_hci_write_local_name(const char * device_name, bt_hci_cmd_callback_fp cb);
```

Description

brief Write local device name ingroup hci

details The Write_Local_Name command provides the ability to modify the userfriendly name for the BR/EDR Controller.

bt_hci_write_page_scan_activity Function

File

hci.h

C

```
bt_bool bt_hci_write_page_scan_activity(bt_uint interval, bt_uint window, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure page scan activity

bt_hci_write_page_scan_period_mode Function

File

hci.h

C

```
bt_bool bt_hci_write_page_scan_period_mode(bt_byte mode, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure page scan period mode

bt_hci_write_page_scan_type Function

File

hci.h

C

```
bt_bool bt_hci_write_page_scan_type(bt_byte scanType, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure page scan type

bt_hci_write_page_timeout Function

File

hci.h

C

```
bt_bool bt_hci_write_page_timeout(bt_uint timeout, bt_hci_cmd_callback_fp cb);
```

Description

brief Configure page timeout

bt_hcitr_3wire_start Function

File

hcitr_3wire.h

C

```
void bt_hcitr_3wire_start();
```

Description

This is function bt_hcitr_3wire_start.

bt_hcitr_bcsp_init_ex Function

File

hcitr_bcsp.h

C

```
void bt_hcitr_bcsp_init_ex(bt_ulong ack_timeout);
```

Description

This is function bt_hcitr_bcsp_init_ex.

bt_hcitr_bcsp_reset_ex Function

File

hcitr_bcsp.h

C

```
void bt_hcitr_bcsp_reset_ex(bt_ulong ack_timeout);
```

Description

This is function bt_hcitr_bcsp_reset_ex.

bt_hcitr_bcsp_start Function

File

hcitr_bcsp.h

C

```
void bt_hcitr_bcsp_start();
```

Description

This is function bt_hcitr_bcsp_start.

bt_hcitr_packet_init Function

File

hcitr_packet.h

C

```
void bt_hcitr_packet_init();
```

Description

This is function bt_hcitr_packet_init.

bt_hcitr_packet_reset Function

File

hcitr_packet.h

C

```
void bt_hcitr_packet_reset();
```

Description

This is function bt_hcitr_packet_reset.

bt_hcitr_packet_start Function

File

hcitr_packet.h

C

```
void bt_hcitr_packet_start();
```

Description

This is function bt_hcitr_packet_start.

bt_hcitr_tih4_init Function

File

hcitr_tih4.h

C

```
void bt_hcitr_tih4_init(bt_hcitr_tih4_power_callback_fp callback);
```

Description

This is function bt_hcitr_tih4_init.

bt_hcitr_tih4_reset Function

File

hcitr_tih4.h

C

```
void bt_hcitr_tih4_reset();
```

Description

This is function bt_hcitr_tih4_reset.

bt_hcitr_tih4_start Function

File

hcitr_tih4.h

C

```
void bt_hcitr_tih4_start();
```

Description

This is function bt_hcitr_tih4_start.

bt_hcitr_tih4_wake_up Function

File

hcitr_tih4.h

C

```
void bt_hcitr_tih4_wake_up();
```

Description

This is function bt_hcitr_tih4_wake_up.

bt_hcitr_uart_init Function

File

hcitr_uart.h

C

```
void bt_hcitr_uart_init();
```

Description

brief Initialize HCI UART (H4) transport protocol ingroup hcitr_uart

details This function initializes internal structures of the transport. The application must call it as early as possible before bt_hcitr_uart_start and before the stack is initialized and started with bt_sys_init and bt_sys_start.

bt_hcitr_uart_reset Function

File

hcitr_uart.h

C

```
void bt_hcitr_uart_reset();
```

Description

brief Re-initialize HCI UART (H4) transport protocol ingroup hcitr_uart

details This function re-initializes the transport. Currently it simply calls bt_hcitr_uart_init. After calling this function the application must perform the full initialization of the stack by calling bt_sys_init, bt_sys_start and initialization functions of all other profile modules the application is intending to use.

bt_hcitr_uart_start Function

File

hcitr_uart.h

C

```
void bt_hcitr_uart_start();
```

Description

brief Start HCI UART (H4) transport protocol ingroup hcitr_uart

details This function starts the transport, i.e., makes it able to receive and send packets.

hci_allocate_conn_state Function

File

hci_conn_state.h

C

```
bt_hci_conn_state_p hci_allocate_conn_state(struct _bt_hci_ctrl_state_s * pctrl);
```

Description

This is function hci_allocate_conn_state.

hci_check_aux_info Function

File

hci_private.h

C

```
bt_bool hci_check_aux_info(bt_long bdaddr_l, bt_long bdaddr_m);
```

Description

From hci_aux_info.c

hci_cq_find_by_bdaddr_and_opcode Function

File

hci_cmd_queue.h

C

```
bt_hci_command_p hci_cq_find_by_bdaddr_and_opcode(bt_queue_element_t* head, bt_bdaddr_t bdaddrFind, bt_int
opcode);
```

Description

This is function hci_cq_find_by_bdaddr_and_opcode.

hci_cq_find_by_hconn Function

File

hci_cmd_queue.h

C

```
bt_hci_command_p hci_cq_find_by_hconn(bt_queue_element_t* head, bt_hci_hconn_t hconn);
```

Description

This is function hci_cq_find_by_hconn.

hci_cq_find_by_hconn_and_opcode Function

File

hci_cmd_queue.h

C

```
bt_hci_command_p hci_cq_find_by_hconn_and_opcode(bt_queue_element_t* head, bt_hci_hconn_t hconnFind, bt_int
opcode);
```

Description

This is function hci_cq_find_by_hconn_and_opcode.

hci_cq_find_by_opcode Function

File

hci_cmd_queue.h

C

```
bt_hci_command_p hci_cq_find_by_opcode(bt_queue_element_t* head, bt_int opcode);
```

Description

This is function hci_cq_find_by_opcode.

hci_get_conn_state Function

File

hci_conn_state.h

C

```
bt_hci_conn_state_p hci_get_conn_state(struct _bt_hci_ctrl_state_s * pctrl, bt_hci_hconn_t h);
```

Description

This is function hci_get_conn_state.

hci_sleep Function

File

hci.h

C

```
void hci_sleep(pf_hci_sleep_callback cb);
```

Description

brief Put local device to sleep

hci_wakeup Function

File

hci.h

C

```
void hci_wakeup(pf_hci_wakeup_callback cb);
```

Description

brief Awaken local device

is_bdaddr_command Function

File

hci_utils.h

C

```
bt_bool is_bdaddr_command(bt_int opcode);
```

Description

This is function is_bdaddr_command.

is_hconn_command Function

File

hci_utils.h

C

```
bt_bool is_hconn_command(bt_int opcode);
```

Description

This is function is_hconn_command.

_bt_hci_ctrl_notify_data_listeners Function

File

hci_private.h

C

```
void _bt_hci_ctrl_notify_data_listeners(bt_hci_conn_state_t* connection, const bt_byte* data, bt_int len,
bt_bool sent);
```

Description

This is function _bt_hci_ctrl_notify_data_listeners.

_bt_hci_ctrl_notify_listeners Function

File

hci_private.h

C

```
void _bt_hci_ctrl_notify_listeners(bt_uint event_id, void* params);
```

Description

This is function _bt_hci_ctrl_notify_listeners.

_bt_hci_get_tick_count Function

File

hci_private.h

C

```
bt_ulong _bt_hci_get_tick_count();
```

Description

This is function _bt_hci_get_tick_count.

_bt_hci_init_signal Function

File

hci_signal.h

C

```
void _bt_hci_init_signal();
```

Description

This is function _bt_hci_init_signal.

_bt_hci_init_timer Function

File

hci_private.h

C

```
void _bt_hci_init_timer();
```

Description

From hci_timer.c

_bt_hci_init_transport Function

File

hci_private.h

C

```
void _bt_hci_init_transport();
```

Description

This is function _bt_hci_init_transport.

_bt_hci_le_command_complete_handler Function

File

hci_private.h

C

```
void _bt_hci_le_command_complete_handler(bt_int status, bt_hci_command_t* pcmd, bt_hci_event_t* pevt);
```

Description

This is function _bt_hci_le_command_complete_handler.

_bt_hci_notify_listeners Function

File

hci_private.h

C

```
void _bt_hci_notify_listeners(bt_hci_conn_state_t* pconn, bt_byte event_id, bt_hci_event_e* params);
```

Description

This is function _bt_hci_notify_listeners.

_bt_hci_set_signal Function

File

hci_signal.h

C

```
void _bt_hci_set_signal();
```

Description

This is function _bt_hci_set_signal.

_hci_allocate_buffers Function

File

hci_private.h

C

```
void _hci_allocate_buffers();
```

Description

Defined by OEM through library configuration

_hci_allocate_cmd Function

File

hci_cmd_buffer.h

C

```
bt_hci_command_t* _hci_allocate_cmd();
```

Description

This is function _hci_allocate_cmd.

_hci_free_cmd Function

File

hci_cmd_buffer.h

C

```
void _hci_free_cmd(bt_hci_command_t* pcmd);
```

Description

This is function _hci_free_cmd.

_hci_init_cmd_buffers Function

File

hci_cmd_buffer.h

C

```
void _hci_init_cmd_buffers();
```

Description

This is function _hci_init_cmd_buffers.

_hci_init_cmd_queues Function

File

hci_cmd_queue.h

C

```
void _hci_init_cmd_queues();
```

Description

This is function _hci_init_cmd_queues.

_hci_receive_start Function

File

hci_private.h

C

```
void _hci_receive_start();
```

Description

From hci_receive.c

_hci_recv_sco_data_packet Function

File

hci_private.h

C

```
void _hci_recv_sco_data_packet(bt_byte_p pbuf);
```

Description

From hci_sco.c

_hci_send_commands_from_queue Function

File

hci_private.h

C

```
void _hci_send_commands_from_queue();
```

Description

From hci_send.c

_hci_send_data Function

File

hci_private.h

C

```
bt_bool _hci_send_data(bt_hci_data_p pdata);
```

Description

This is function _hci_send_data.

_hci_send_data_fragment Function

File

hci_private.h

C

```
void _hci_send_data_fragment(bt_hci_data_p pdata);
```

Description

This is function _hci_send_data_fragment.

_hci_send_data_from_queue Function

File

hci_private.h

C

```
void _hci_send_data_from_queue();
```

Description

This is function _hci_send_data_from_queue.

_bt_le_evt_handler Function

File

hci_private.h

C

```
void _bt_le_evt_handler(bt_hci_event_t* evt);
```

Description

This is function _bt_le_evt_handler.

bt_hci_le_allocate_set_scan_response_data_command Function

File

hci_le.h

C

```
bt_hci_command_t* bt_hci_le_allocate_set_scan_response_data_command(bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_hci_le_allocate_set_scan_response_data_command.

bt_hci_le_cancel_connect_ex Function

File

hci_le.h

C

```
bt_bool bt_hci_le_cancel_connect_ex(bt_hci_cmd_callback_fp cb, void * cb_param);
```

Description

This is function bt_hci_le_cancel_connect_ex.

bt_hci_le_ibeacon_add Function

File

hci_le.h

C

```
bt_bool bt_hci_le_ibeacon_add(const bt_uuid_t* uuid, bt_uint major, bt_uint minor, bt_byte tx_power_level,
bt_hci_command_p pcmd);
```

Description

This is function bt_hci_le_ibeacon_add.

bt_hci_set_vendor_specific_event_handler Function

File

hci_evt_handlers.h

C

```
void bt_hci_set_vendor_specific_event_handler(bt_hci_event_handler_fp handler);
```

Description

This is function bt_hci_set_vendor_specific_event_handler.

bt_hcitr_3wire_init_ex Function

File

hcitr_3wire.h

C

```
void bt_hcitr_3wire_init_ex(bt_ulong ack_timeout, bt_bool data_integrity_check_enabled);
```

Description

This is function bt_hcitr_3wire_init_ex.

bt_hcitr_3wire_reset_ex Function

File

hcitr_3wire.h

C

```
void bt_hcitr_3wire_reset_ex(bt_ulong ack_timeout, bt_bool data_integrity_check_enabled);
```

Description

This is function bt_hcitr_3wire_reset_ex.

_bt_hci_set_init_flags Function

File

hci_private.h

C

```
void _bt_hci_set_init_flags(bt_byte flags);
```

Description

This is function _bt_hci_set_init_flags.

hci_get_conn_state_by_bdaddr Function

File

hci_conn_state.h

C

```
bt_hci_conn_state_p hci_get_conn_state_by_bdaddr(struct _bt_hci_ctrl_state_s * pctrl, const bt_bdaddr_t*
pbdaddr_remote, bt_byte conn_type, bt_byte link_type);
```

Description

This is function hci_get_conn_state_by_bdaddr.

bt_hci_cancel_find_devices_ex Function

File

hci.h

C

```
bt_bool bt_hci_cancel_find_devices_ex(bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

brief Stop inquiry

bt_hci_get_inquiry_response_tx_power_level Function

File

hci.h

C

```
bt_byte bt_hci_get_inquiry_response_tx_power_level();
```

Description

This is function bt_hci_get_inquiry_response_tx_power_level.

bt_hci_init_ex Function

File

hci.h

C

```
void bt_hci_init_ex(bt_byte default_link_policy);
```

Description

brief Initialize the HCI layer. ingroup hci

This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt_sys_init_ex.

@param default_link_policy default link policy settings. This is a bitmask that defines the initial value of the link policy settings for all new BR/EDR connections. This value can be a combination of the following values: li HCI_LINK_POLICY_ENABLE_ROLE_SWITCH li HCI_LINK_POLICY_ENABLE_HOLD_MODE li HCI_LINK_POLICY_ENABLE_SNIFF_MODE li HCI_LINK_POLICY_ENABLE_PARK_STATE

To enable all settings pass HCI_LINK_POLICY_ENABLE_ALL.

bt_hci_reset Function

File

hci.h

C

```
bt_bool bt_hci_reset(bt_hci_cmd_callback_fp callback, void* callback_param);
```

Description

brief Reset controller. ingroup hci

details This function resets the BT controller.

param callback Completion callback. Called when the controller has been reset. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

bt_hci_send_pin_code_ex Function

File

hci.h

C

```
bt_bool bt_hci_send_pin_code_ex(bt_bdaddr_p pbdaddr_remote, const char* pin, bt_hci_cmd_callback_fp
callback, void* callback_param);
```

Description

This is function bt_hci_send_pin_code_ex.

bt_hci_set_scan_ex Function

File

hci.h

C

```
bt_bool bt_hci_set_scan_ex(bt_bool discoverable, bt_ulong discoverable_period, bt_byte discoverable_mode,
bt_bool connectable, bt_ulong connectable_period, bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_set_scan_ex.

bt_hci_write_link_policy_settings Function

File

hci.h

C

```
bt_bool bt_hci_write_link_policy_settings(bt_hci_conn_state_t* conn, bt_uint settings,
bt_hci_cmd_callback_fp cb, void* cb_param);
```

Description

This is function bt_hci_write_link_policy_settings.

bt_hci_write_local_name_ex Function

File

hci.h

C

```
bt_bool bt_hci_write_local_name_ex(const char * device_name, bt_hci_cmd_callback_fp cb, void*
callback_param);
```

Description

This is function bt_hci_write_local_name_ex.

HCI Data Types and Constants

bt_hci_add_param_hconn Macro

File

hci.h

C

```
#define bt_hci_add_param_hconn(pcmd, value) \
    bt_hci_add_param_int(pcmd, value)
```

Description

brief Add connection handle parameter to an HCI command ingroup hci

bt_hci_add_param_lap Macro

File

hci.h

C

```
#define bt_hci_add_param_lap(cmd, lap) bt_hci_add_param_cod(cmd, lap)
```

Description

This is macro bt_hci_add_param_lap.

bt_hci_add_param_uint Macro

File

hci.h

C

```
#define bt_hci_add_param_uint(cmd, value) bt_hci_add_param_int(cmd, (bt_int)(value))
```

Description

brief Add unsigned int parameter to an HCI command ingroup hci

bt_hci_add_param_ulong Macro

File

hci.h

C

```
#define bt_hci_add_param_ulong(cmd, value) bt_hci_add_param_long(cmd, (bt_long)(value))
```

Description

brief Add unsigned long parameter to an HCI command ingroup hci

bt_hci_authenticate Macro

File

hci.h

C

```
#define bt_hci_authenticate(pconn, cb) bt_hci_authenticate_ex(pconn, cb, NULL)
```

Description

This is macro bt_hci_authenticate.

bt_hci_exit_sniff_mode Macro

File

hci.h

C

```
#define bt_hci_exit_sniff_mode(pconn, cb) bt_hci_exit_sniff_mode_ex(pconn, cb, NULL)
```

Description

brief Cancel sniff mode

bt_hci_get_evt_param_hconn Macro

File

hci.h

C

```
#define bt_hci_get_evt_param_hconn(pevt, pvalue, poffset) \
    bt_hci_get_evt_param_int(pevt, pvalue, poffset)
```

Description

brief Get connection handle parameter from HCI event ingroup hci

bt_hci_get_evt_param_uint Macro

File

hci.h

C

```
#define bt_hci_get_evt_param_uint(pevt, pvalue, poffset) bt_hci_get_evt_param_int(pevt, (bt_int*)pvalue,
poffset)
```

Description

This is macro bt_hci_get_evt_param_uint.

bt_hci_get_evt_param_ulong Macro

File

hci.h

C

```
#define bt_hci_get_evt_param_ulong(pevt, pvalue, poffset) bt_hci_get_evt_param_long(pevt, (bt_long*)pvalue,
poffset)
```

Description

This is macro bt_hci_get_evt_param_ulong.

bt_hci_get_param_hconn Macro

File

hci.h

C

```
#define bt_hci_get_param_hconn(pcmd, pvalue, poffset) bt_hci_get_param_int(pcmd, pvalue, poffset)
```

Description

brief Get connection handle parameter from HCI command ingroup hci

bt_hci_le_set_adevrtising_enable Macro

File

hci_le.h

C

```
#define bt_hci_le_set_adevrtising_enable(adv_enable, cb) bt_hci_le_set_adevrtising_enable_ex(adv_enable,
cb, NULL)
```

Description

This is macro bt_hci_le_set_adevrtising_enable.

bt_hci_role_change Macro

File

hci.h

C

```
#define bt_hci_role_change(pbdaddr, role, cb) bt_hci_role_change_ex(pbdaddr, role, cb, NULL)
```

Description

This is macro bt_hci_role_change.

bt_hci_set_encryption Macro

File

hci.h

C

```
#define bt_hci_set_encryption(pconn, encryption_enable, cb) bt_hci_set_encryption_ex(pconn,
encryption_enable, cb, NULL)
```

Description

This is macro bt_hci_set_encryption.

bt_hci_sniff_mode Macro

File

hci.h

C

```
#define bt_hci_sniff_mode(pconn, min_interval, max_interval, attempt_slots, timeout, cb) \
    bt_hci_sniff_mode_ex(pconn, min_interval, max_interval, attempt_slots, timeout, cb, NULL)
```

Description

brief Put local device to sniff mode

bt_hci_sniff_subrating Macro

File

hci.h

C

```
#define bt_hci_sniff_subrating(pconn, max_latency, min_remote_timeout, min_local_timeout, cb) \
    bt_hci_sniff_subrating_ex(pconn, max_latency, min_remote_timeout, min_local_timeout, cb, NULL)
```

Description

This is macro bt_hci_sniff_subrating.

bt_hcitr_3wire_init Macro

File

hcitr_3wire.h

C

```
#define bt_hcitr_3wire_init bt_hcitr_3wire_init_ex(HCITR_3WIRE_DEFAULT_ACK_TIMEOUT, BT_FALSE)
```

Description

This is macro bt_hcitr_3wire_init.

bt_hcitr_3wire_reset Macro

File

hcitr_3wire.h

C

```
#define bt_hcitr_3wire_reset bt_hcitr_3wire_reset_ex(HCITR_3WIRE_DEFAULT_ACK_TIMEOUT, BT_FALSE)
```

Description

This is macro bt_hcitr_3wire_reset.

bt_hcitr_bcsp_init Macro

File

hcitr_bcsp.h

C

```
#define bt_hcitr_bcsp_init bt_hcitr_bcsp_init_ex(HCITR_BCSP_DEFAULT_ACK_TIMEOUT)
```

Description

This is macro bt_hcitr_bcsp_init.

bt_hcitr_bcsp_reset Macro

File

hcitr_bcsp.h

C

```
#define bt_hcitr_bcsp_reset bt_hcitr_bcsp_reset_ex(HCITR_BCSP_DEFAULT_ACK_TIMEOUT)
```

Description

This is macro bt_hcitr_bcsp_reset.

COD_MAJOR_AUDIO Macro

File

baseband.h

C

```
#define COD_MAJOR_AUDIO 0x0400
```

Description

This is macro COD_MAJOR_AUDIO.

COD_MAJOR_COMPUTER Macro

File

baseband.h

C

```
#define COD_MAJOR_COMPUTER 0x0100
```

Description

This is macro COD_MAJOR_COMPUTER.

COD_MAJOR_HEALTH Macro

File

baseband.h

C

```
#define COD_MAJOR_HEALTH 0x0900
```

Description

This is macro COD_MAJOR_HEALTH.

COD_MAJOR_IMAGING Macro

File

baseband.h

C

```
#define COD_MAJOR_IMAGING 0x0600
```

Description

This is macro COD_MAJOR_IMAGING.

COD_MAJOR_MISC Macro

File

baseband.h

C

```
#define COD_MAJOR_MISC 0x0000
```

Description

This is macro COD_MAJOR_MISC.

COD_MAJOR_NET_ACCESS_POINT Macro

File

baseband.h

C

```
#define COD_MAJOR_NET_ACCESS_POINT 0x0300
```

Description

This is macro COD_MAJOR_NET_ACCESS_POINT.

COD_MAJOR_PERIPHERAL Macro

File

baseband.h

C

```
#define COD_MAJOR_PERIPHERAL 0x0500
```

Description

This is macro COD_MAJOR_PERIPHERAL.

COD_MAJOR_PHONE Macro

File

baseband.h

C

```
#define COD_MAJOR_PHONE 0x0200
```

Description

This is macro COD_MAJOR_PHONE.

COD_MAJOR_TOY Macro

File

baseband.h

C

```
#define COD_MAJOR_TOY 0x0800
```

Description

This is macro COD_MAJOR_TOY.

COD_MAJOR_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MAJOR_UNCATEGORIZED 0x1f00
```

Description

This is macro COD_MAJOR_UNCATEGORIZED.

COD_MAJOR_WEARABLE Macro

File

baseband.h

C

```
#define COD_MAJOR_WEARABLE 0x0700
```

Description

This is macro COD_MAJOR_WEARABLE.

COD_MINOR_AV_CAMCORDER Macro

File

baseband.h

C

```
#define COD_MINOR_AV_CAMCORDER (0x0d << 2)
```

Description

This is macro COD_MINOR_AV_CAMCORDER.

COD_MINOR_AV_CAR_AUDIO Macro

File

baseband.h

C

```
#define COD_MINOR_AV_CAR_AUDIO (0x08 << 2)
```

Description

This is macro COD_MINOR_AV_CAR_AUDIO.

COD_MINOR_AV_GAMING Macro

File

baseband.h

C

```
#define COD_MINOR_AV_GAMING (0x12 << 2)
```

Description

This is macro COD_MINOR_AV_GAMING.

COD_MINOR_AV_HANDSFREE Macro

File

baseband.h

C

```
#define COD_MINOR_AV_HANDSFREE (0x02 << 2)
```

Description

This is macro COD_MINOR_AV_HANDSFREE.

COD_MINOR_AV_HEADPHONES Macro

File

baseband.h

C

```
#define COD_MINOR_AV_HEADPHONES (0x06 << 2)
```

Description

This is macro COD_MINOR_AV_HEADPHONES.

COD_MINOR_AV_HEADSET Macro

File

baseband.h

C

```
#define COD_MINOR_AV_HEADSET (0x01 << 2)
```

Description

This is macro COD_MINOR_AV_HEADSET.

COD_MINOR_AV_HIFI_AUDIO Macro

File

baseband.h

C

```
#define COD_MINOR_AV_HIFI_AUDIO (0x0a << 2)
```

Description

This is macro COD_MINOR_AV_HIFI_AUDIO.

COD_MINOR_AV_LOUDSPEAKER Macro

File

baseband.h

C

```
#define COD_MINOR_AV_LOUDSPEAKER (0x05 << 2)
```

Description

This is macro COD_MINOR_AV_LOUDSPEAKER.

COD_MINOR_AV_MICROPHONE Macro

File

baseband.h

C

```
#define COD_MINOR_AV_MICROPHONE (0x04 << 2)
```

Description

This is macro COD_MINOR_AV_MICROPHONE.

COD_MINOR_AV_PORTABLE_AUDIO Macro

File

baseband.h

C

```
#define COD_MINOR_AV_PORTABLE_AUDIO (0x07 << 2)
```

Description

This is macro COD_MINOR_AV_PORTABLE_AUDIO.

COD_MINOR_AV_RESERVED Macro

File

baseband.h

C

```
#define COD_MINOR_AV_RESERVED (0x03 << 2)
```

Description

This is macro COD_MINOR_AV_RESERVED.

COD_MINOR_AV_RESERVERD2 Macro

File

baseband.h

C

```
#define COD_MINOR_AV_RESERVERD2 (0x11 << 2)
```

Description

This is macro COD_MINOR_AV_RESERVERD2.

COD_MINOR_AV_SET_TOP_BOX Macro

File

baseband.h

C

```
#define COD_MINOR_AV_SET_TOP_BOX (0x09 << 2)
```

Description

This is macro COD_MINOR_AV_SET_TOP_BOX.

COD_MINOR_AV_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_AV_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_AV_UNCATEGORIZED.

COD_MINOR_AV_VCR Macro

File

baseband.h

C

```
#define COD_MINOR_AV_VCR (0x0b << 2)
```

Description

This is macro COD_MINOR_AV_VCR.

COD_MINOR_AV_VIDE_DISPLAY_AND_LOUDSPEAKER Macro

File

baseband.h

C

```
#define COD_MINOR_AV_VIDE_DISPLAY_AND_LOUDSPEAKER (0x0f << 2)
```

Description

This is macro COD_MINOR_AV_VIDE_DISPLAY_AND_LOUDSPEAKER.

COD_MINOR_AV_VIDEO_CAMERA Macro

File

baseband.h

C

```
#define COD_MINOR_AV_VIDEO_CAMERA (0x0c << 2)
```

Description

This is macro COD_MINOR_AV_VIDEO_CAMERA.

COD_MINOR_AV_VIDEO_CONFERENCING Macro

File

baseband.h

C

```
#define COD_MINOR_AV_VIDEO_CONFERENCING (0x10 << 2)
```

Description

This is macro COD_MINOR_AV_VIDEO_CONFERENCING.

COD_MINOR_AV_VIDEO_MONITOR Macro

File

baseband.h

C

```
#define COD_MINOR_AV_VIDEO_MONITOR (0x0e << 2)
```

Description

This is macro COD_MINOR_AV_VIDEO_MONITOR.

COD_MINOR_COMPUTER_DESKTOP Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_DESKTOP (0x01 << 2)
```

Description

This is macro COD_MINOR_COMPUTER_DESKTOP.

COD_MINOR_COMPUTER_HANDHELD Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_HANDHELD (0x04 << 2)
```

Description

This is macro COD_MINOR_COMPUTER_HANDHELD.

COD_MINOR_COMPUTER_LAPTOP Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_LAPTOP (0x03 << 2)
```

Description

This is macro COD_MINOR_COMPUTER_LAPTOP.

COD_MINOR_COMPUTER_PALMSIZED Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_PALMSIZED (0x05 << 2)
```

Description

This is macro COD_MINOR_COMPUTER_PALMSIZED.

COD_MINOR_COMPUTER_SERVER Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_SERVER (0x02 << 2)
```

Description

This is macro COD_MINOR_COMPUTER_SERVER.

COD_MINOR_COMPUTER_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_COMPUTER_UNCATEGORIZED.

COD_MINOR_COMPUTER_WEARABLE Macro

File

baseband.h

C

```
#define COD_MINOR_COMPUTER_WEARABLE (0x06 << 2)
```

Description

This is macro COD_MINOR_COMPUTER_WEARABLE.

COD_MINOR_HEALTH_BPM Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_BPM (0x01 << 2)
```

Description

This is macro COD_MINOR_HEALTH_BPM.

COD_MINOR_HEALTH_DATA_DISPLAY Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_DATA_DISPLAY (0x07 << 2)
```

Description

This is macro COD_MINOR_HEALTH_DATA_DISPLAY.

COD_MINOR_HEALTH_GLUCOSE_METER Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_GLUCOSE_METER (0x04 << 2)
```

Description

This is macro COD_MINOR_HEALTH_GLUCOSE_METER.

COD_MINOR_HEALTH_HEART_MONITOR Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_HEART_MONITOR (0x06 << 2)
```

Description

This is macro COD_MINOR_HEALTH_HEART_MONITOR.

COD_MINOR_HEALTH_PULSE_OXIMETER Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_PULSE_OXIMETER (0x05 << 2)
```

Description

This is macro COD_MINOR_HEALTH_PULSE_OXIMETER.

COD_MINOR_HEALTH_THERMOMETER Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_THERMOMETER (0x02 << 2)
```

Description

This is macro COD_MINOR_HEALTH_THERMOMETER.

COD_MINOR_HEALTH_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_HEALTH_UNCATEGORIZED.

COD_MINOR_HEALTH_WEIGHING_SCALE Macro

File

baseband.h

C

```
#define COD_MINOR_HEALTH_WEIGHING_SCALE (0x03 << 2)
```

Description

This is macro COD_MINOR_HEALTH_WEIGHING_SCALE.

COD_MINOR_IMAGING_CAMERA Macro

File

baseband.h

C

```
#define COD_MINOR_IMAGING_CAMERA (0x01 << 5)
```

Description

This is macro COD_MINOR_IMAGING_CAMERA.

COD_MINOR_IMAGING_DISPLAY Macro

File

baseband.h

C

```
#define COD_MINOR_IMAGING_DISPLAY (0x01 << 4)
```

Description

This is macro COD_MINOR_IMAGING_DISPLAY.

COD_MINOR_IMAGING_PRINTER Macro

File

baseband.h

C

```
#define COD_MINOR_IMAGING_PRINTER (0x01 << 7)
```

Description

This is macro COD_MINOR_IMAGING_PRINTER.

COD_MINOR_IMAGING_SCANNER Macro

File

baseband.h

C

```
#define COD_MINOR_IMAGING_SCANNER (0x01 << 6)
```

Description

This is macro COD_MINOR_IMAGING_SCANNER.

COD_MINOR_IMAGING_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_IMAGING_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_IMAGING_UNCATEGORIZED.

COD_MINOR_LAN_01_17 Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_01_17 (0x01 << 5)
```

Description

This is macro COD_MINOR_LAN_01_17.

COD_MINOR_LAN_17_33 Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_17_33 (0x02 << 5)
```

Description

This is macro COD_MINOR_LAN_17_33.

COD_MINOR_LAN_33_50 Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_33_50 (0x03 << 5)
```

Description

This is macro COD_MINOR_LAN_33_50.

COD_MINOR_LAN_50_67 Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_50_67 (0x04 << 5)
```

Description

This is macro COD_MINOR_LAN_50_67.

COD_MINOR_LAN_67_83 Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_67_83 (0x05 << 5)
```

Description

This is macro COD_MINOR_LAN_67_83.

COD_MINOR_LAN_83_99 Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_83_99 (0x06 << 5)
```

Description

This is macro COD_MINOR_LAN_83_99.

COD_MINOR_LAN_FULLY_AVAILABLE Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_FULLY_AVAILABLE 0x00
```

Description

This is macro COD_MINOR_LAN_FULLY_AVAILABLE.

COD_MINOR_LAN_NO_SERVICE Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_NO_SERVICE (0x07 << 5)
```

Description

This is macro COD_MINOR_LAN_NO_SERVICE.

COD_MINOR_LAN_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_LAN_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_LAN_UNCATEGORIZED.

COD_MINOR_PERIPHERAL_CARD_READER Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_CARD_READER (0x06 << 2)
```

Description

This is macro COD_MINOR_PERIPHERAL_CARD_READER.

COD_MINOR_PERIPHERAL_COMBO Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_COMBO 0xC0
```

Description

This is macro COD_MINOR_PERIPHERAL_COMBO.

COD_MINOR_PERIPHERAL_DIGITIZER Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_DIGITIZER (0x05 << 2)
```

Description

This is macro COD_MINOR_PERIPHERAL_DIGITIZER.

COD_MINOR_PERIPHERAL_GAMEPAD Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_GAMEPAD (0x02 << 2)
```

Description

This is macro COD_MINOR_PERIPHERAL_GAMEPAD.

COD_MINOR_PERIPHERAL_JOYSTICK Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_JOYSTICK (0x01 << 2)
```

Description

This is macro COD_MINOR_PERIPHERAL_JOYSTICK.

COD_MINOR_PERIPHERAL_KEYBOARD Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_KEYBOARD 0x40
```

Description

This is macro COD_MINOR_PERIPHERAL_KEYBOARD.

COD_MINOR_PERIPHERAL_MOUSE Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_MOUSE 0x80
```

Description

This is macro COD_MINOR_PERIPHERAL_MOUSE.

COD_MINOR_PERIPHERAL_OTHER Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_OTHER 0x00
```

Description

This is macro COD_MINOR_PERIPHERAL_OTHER.

COD_MINOR_PERIPHERAL_REMOTE Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_REMOTE (0x03 << 2)
```

Description

This is macro COD_MINOR_PERIPHERAL_REMOTE.

COD_MINOR_PERIPHERAL_SENSING Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_SENSING (0x04 << 2)
```

Description

This is macro COD_MINOR_PERIPHERAL_SENSING.

COD_MINOR_PERIPHERAL_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_PERIPHERAL_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_PERIPHERAL_UNCATEGORIZED.

COD_MINOR_PHONE_CELLULAR Macro

File

baseband.h

C

```
#define COD_MINOR_PHONE_CELLULAR (0x01 << 2)
```

Description

This is macro COD_MINOR_PHONE_CELLULAR.

COD_MINOR_PHONE_CORDLESS Macro

File

baseband.h

C

```
#define COD_MINOR_PHONE_CORDLESS (0x02 << 2)
```

Description

This is macro COD_MINOR_PHONE_CORDLESS.

COD_MINOR_PHONE_ISDN Macro

File

baseband.h

C

```
#define COD_MINOR_PHONE_ISDN (0x05 << 2)
```

Description

This is macro COD_MINOR_PHONE_ISDN.

COD_MINOR_PHONE_SMART Macro

File

baseband.h

C

```
#define COD_MINOR_PHONE_SMART (0x03 << 2)
```

Description

This is macro COD_MINOR_PHONE_SMART.

COD_MINOR_PHONE_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_PHONE_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_PHONE_UNCATEGORIZED.

COD_MINOR_PHONE_WIREDMODEM Macro

File

baseband.h

C

```
#define COD_MINOR_PHONE_WIREDMODEM (0x04 << 2)
```

Description

This is macro COD_MINOR_PHONE_WIREDMODEM.

COD_MINOR_TOY_CONTROLLER Macro

File

baseband.h

C

```
#define COD_MINOR_TOY_CONTROLLER (0x04 << 2)
```

Description

This is macro COD_MINOR_TOY_CONTROLLER.

COD_MINOR_TOY_DOLL Macro

File

baseband.h

C

```
#define COD_MINOR_TOY_DOLL (0x03 << 2)
```

Description

This is macro COD_MINOR_TOY_DOLL.

COD_MINOR_TOY_GAME Macro

File

baseband.h

C

```
#define COD_MINOR_TOY_GAME (0x05 << 2)
```

Description

This is macro COD_MINOR_TOY_GAME.

COD_MINOR_TOY_ROBOT Macro

File

baseband.h

C

```
#define COD_MINOR_TOY_ROBOT (0x01 << 2)
```

Description

This is macro COD_MINOR_TOY_ROBOT.

COD_MINOR_TOY_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_TOY_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_TOY_UNCATEGORIZED.

COD_MINOR_TOY_VEHICLE Macro

File

baseband.h

C

```
#define COD_MINOR_TOY_VEHICLE (0x02 << 2)
```

Description

This is macro COD_MINOR_TOY_VEHICLE.

COD_MINOR_WEARABLE_GLASSES Macro

File

baseband.h

C

```
#define COD_MINOR_WEARABLE_GLASSES (0x05 << 2)
```

Description

This is macro COD_MINOR_WEARABLE_GLASSES.

COD_MINOR_WEARABLE_HELMET Macro

File

baseband.h

C

```
#define COD_MINOR_WEARABLE_HELMET (0x04 << 2)
```

Description

This is macro COD_MINOR_WEARABLE_HELMET.

COD_MINOR_WEARABLE_JACKET Macro

File

baseband.h

C

```
#define COD_MINOR_WEARABLE_JACKET (0x03 << 2)
```

Description

This is macro COD_MINOR_WEARABLE_JACKET.

COD_MINOR_WEARABLE_PAGER Macro

File

baseband.h

C

```
#define COD_MINOR_WEARABLE_PAGER (0x02 << 2)
```

Description

This is macro COD_MINOR_WEARABLE_PAGER.

COD_MINOR_WEARABLE_UNCATEGORIZED Macro

File

baseband.h

C

```
#define COD_MINOR_WEARABLE_UNCATEGORIZED 0x00
```

Description

This is macro COD_MINOR_WEARABLE_UNCATEGORIZED.

COD_MINOR_WEARABLE_WATCH Macro

File

baseband.h

C

```
#define COD_MINOR_WEARABLE_WATCH (0x01 << 2)
```

Description

This is macro COD_MINOR_WEARABLE_WATCH.

COS_AUDIO Macro

File

baseband.h

C

```
#define COS_AUDIO 0x200000
```

Description

This is macro COS_AUDIO.

COS_CAPTURING Macro

File

baseband.h

C

```
#define COS_CAPTURING 0x080000
```

Description

This is macro COS_CAPTURING.

COS_INFORMATION Macro

File

baseband.h

C

```
#define COS_INFORMATION 0x800000
```

Description

This is macro COS_INFORMATION.

COS_LIMITED_DISCOVERABLE_MODE Macro

File

baseband.h

C

```
#define COS_LIMITED_DISCOVERABLE_MODE 0x002000
```

Description

This is macro COS_LIMITED_DISCOVERABLE_MODE.

COS_NETWORKING Macro

File

baseband.h

C

```
#define COS_NETWORKING 0x020000
```

Description

This is macro COS_NETWORKING.

COS_OBJECTTRANSFER Macro

File

baseband.h

C

```
#define COS_OBJECTTRANSFER 0x100000
```

Description

This is macro COS_OBJECTTRANSFER.

COS_POSITIONING Macro

File

baseband.h

C

```
#define COS_POSITIONING 0x010000
```

Description

This is macro COS_POSITIONING.

COS_RENDERING Macro

File

baseband.h

C

```
#define COS_RENDERING 0x040000
```

Description

This is macro COS_RENDERING.

COS_TELEPHONY Macro

File

baseband.h

C

```
#define COS_TELEPHONY 0x400000
```

Description

This is macro COS_TELEPHONY.

HCI_ACCEPT_CONNECTION_REQUEST Macro

File

hci.h

C

```
#define HCI_ACCEPT_CONNECTION_REQUEST HCI_OPCODE(OGF_LINK_CONTROL, 0x0009)
```

Description

This is macro HCI_ACCEPT_CONNECTION_REQUEST.

HCI_ACCEPT_SYNCH_CONNECTION_REQUEST Macro

File

hci.h

C

```
#define HCI_ACCEPT_SYNCH_CONNECTION_REQUEST HCI_OPCODE(OGF_LINK_CONTROL, 0x0029)
```

Description

This is macro HCI_ACCEPT_SYNCH_CONNECTION_REQUEST.

HCI_ACL_DATA_HEADER_LEN Macro

File

hci.h

C

```
#define HCI_ACL_DATA_HEADER_LEN 4
```

Description

This is macro HCI_ACL_DATA_HEADER_LEN.

HCI_AUTHENTICATION_REQUESTED Macro

File

hci.h

C

```
#define HCI_AUTHENTICATION_REQUESTED HCI_OPCODE(OGF_LINK_CONTROL, 0x0011)
```

Description

This is macro HCI_AUTHENTICATION_REQUESTED.

HCI_BB_PACKET_TYPE_DH1 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_DH1 0x0010
```

Description

This is macro HCI_BB_PACKET_TYPE_DH1.

HCI_BB_PACKET_TYPE_DH3 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_DH3 0x0800
```

Description

This is macro HCI_BB_PACKET_TYPE_DH3.

HCI_BB_PACKET_TYPE_DH5 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_DH5 0x8000
```

Description

This is macro HCI_BB_PACKET_TYPE_DH5.

HCI_BB_PACKET_TYPE_DM1 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_DM1 0x0008
```

Description

This is macro HCI_BB_PACKET_TYPE_DM1.

HCI_BB_PACKET_TYPE_DM3 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_DM3 0x0400
```

Description

This is macro HCI_BB_PACKET_TYPE_DM3.

HCI_BB_PACKET_TYPE_DM5 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_DM5 0x4000
```

Description

This is macro HCI_BB_PACKET_TYPE_DM5.

HCI_BB_PACKET_TYPE_NO_2_DH1 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_NO_2_DH1 0x0002
```

Description

This is macro HCI_BB_PACKET_TYPE_NO_2_DH1.

HCI_BB_PACKET_TYPE_NO_2_DH3 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_NO_2_DH3 0x0100
```

Description

This is macro HCI_BB_PACKET_TYPE_NO_2_DH3.

HCI_BB_PACKET_TYPE_NO_2_DH5 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_NO_2_DH5 0x1000
```

Description

This is macro HCI_BB_PACKET_TYPE_NO_2_DH5.

HCI_BB_PACKET_TYPE_NO_3_DH1 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_NO_3_DH1 0x0004
```

Description

This is macro HCI_BB_PACKET_TYPE_NO_3_DH1.

HCI_BB_PACKET_TYPE_NO_3_DH3 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_NO_3_DH3 0x0200
```

Description

This is macro HCI_BB_PACKET_TYPE_NO_3_DH3.

HCI_BB_PACKET_TYPE_NO_3_DH5 Macro

File

hci.h

C

```
#define HCI_BB_PACKET_TYPE_NO_3_DH5 0x2000
```

Description

This is macro HCI_BB_PACKET_TYPE_NO_3_DH5.

HCI_C2H_BROADCAST_NOT_PARCKED_SLAVE Macro

File

hci.h

C

```
#define HCI_C2H_BROADCAST_NOT_PARCKED_SLAVE 0x1
```

Description

This is macro HCI_C2H_BROADCAST_NOT_PARCKED_SLAVE.

HCI_C2H_BROADCAST_P2P Macro

File

hci.h

C

```
#define HCI_C2H_BROADCAST_P2P 0x0
```

Description

This is macro HCI_C2H_BROADCAST_P2P.

HCI_C2H_BROADCAST_PARCKED_SLAVE Macro

File

hci.h

C

```
#define HCI_C2H_BROADCAST_PARCKED_SLAVE 0x2
```

Description

This is macro HCI_C2H_BROADCAST_PARCKED_SLAVE.

HCI_C2H_BROADCAST_RESERVED Macro

File

hci.h

C

```
#define HCI_C2H_BROADCAST_RESERVED 0x3
```

Description

This is macro HCI_C2H_BROADCAST_RESERVED.

HCI_CHANGE_CONNECTION_LINK_KEY Macro

File

hci.h

C

```
#define HCI_CHANGE_CONNECTION_LINK_KEY HCI_OPCODE(OGF_LINK_CONTROL, 0x0015)
```

Description

This is macro HCI_CHANGE_CONNECTION_LINK_KEY.

HCI_CHANGE_CONNECTION_PACKET_TYPE Macro

File

hci.h

C

```
#define HCI_CHANGE_CONNECTION_PACKET_TYPE HCI_OPCODE(OGF_LINK_CONTROL, 0x000F)
```

Description

This is macro HCI_CHANGE_CONNECTION_PACKET_TYPE.

HCI_CMD_HEADER_LEN Macro

File

hci.h

C

```
#define HCI_CMD_HEADER_LEN 3
```

Description

This is macro HCI_CMD_HEADER_LEN.

HCI_CMD_STATUS_BEING_SENT Macro

File

hci.h

C

```
#define HCI_CMD_STATUS_BEING_SENT 2
```

Description

This is macro HCI_CMD_STATUS_BEING_SENT.

HCI_CMD_STATUS_PENDING Macro

File

hci.h

C

```
#define HCI_CMD_STATUS_PENDING 0
```

Description

This is macro HCI_CMD_STATUS_PENDING.

HCI_CMD_STATUS_WAITING_RESPONSE Macro

File

hci.h

C

```
#define HCI_CMD_STATUS_WAITING_RESPONSE 1
```

Description

This is macro HCI_CMD_STATUS_WAITING_RESPONSE.

HCI_CONFIG_BECOME_MASTER Macro

File

hci.h

C

```
#define HCI_CONFIG_BECOME_MASTER 0x04
```

Description

This is macro HCI_CONFIG_BECOME_MASTER.

HCI_CONFIG_ENABLE_AUTHENTICATION Macro

File

hci.h

C

```
#define HCI_CONFIG_ENABLE_AUTHENTICATION 0x01
```

Description

This is macro HCI_CONFIG_ENABLE_AUTHENTICATION.

HCI_CONFIG_ENABLE_ENCRYPTION Macro

File

hci.h

C

```
#define HCI_CONFIG_ENABLE_ENCRYPTION 0x02
```

Description

This is macro HCI_CONFIG_ENABLE_ENCRYPTION.

HCI_CONN_ROLE_MASTER Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_ROLE_MASTER 0
```

Description

This is macro HCI_CONN_ROLE_MASTER.

HCI_CONN_ROLE_SLAVE Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_ROLE_SLAVE 1
```

Description

This is macro HCI_CONN_ROLE_SLAVE.

HCI_CONN_STATE_AUTHENTICATING Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_STATE_AUTHENTICATING 1
```

Description

This is macro HCI_CONN_STATE_AUTHENTICATING.

HCI_CONN_STATE_CLOSED Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_STATE_CLOSED 0
```

Description

This is macro HCI_CONN_STATE_CLOSED.

HCI_CONN_STATE_OPEN Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_STATE_OPEN 2
```

Description

This is macro HCI_CONN_STATE_OPEN.

HCI_CONN_TYPE_ACL Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_TYPE_ACL 1
```

Description

This is macro HCI_CONN_TYPE_ACL.

HCI_CONN_TYPE_ESCO Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_TYPE_ESCO 2
```

Description

This is macro HCI_CONN_TYPE_ESCO.

HCI_CONN_TYPE_SCO Macro

File

hci_conn_state.h

C

```
#define HCI_CONN_TYPE_SCO 0
```

Description

This is macro HCI_CONN_TYPE_SCO.

HCI_CONNECTABLE Macro

File

hci.h

C

```
#define HCI_CONNECTABLE 0x02
```

Description

This is macro HCI_CONNECTABLE.

HCI_CONTROLLER Macro

File

hci.h

C

```
#define HCI_CONTROLLER _phci_ctrl
```

Description

This is macro HCI_CONTROLLER.

HCI_CREATE_CONNECTION Macro

File

hci.h

C

```
#define HCI_CREATE_CONNECTION HCI_OPCODE(OGF_LINK_CONTROL, 0x0005)
```

Description

This is macro HCI_CREATE_CONNECTION.

HCI_CREATE_CONNECTION_CANCEL Macro

File

hci.h

C

```
#define HCI_CREATE_CONNECTION_CANCEL HCI_OPCODE(OGF_LINK_CONTROL, 0x0008)
```

Description

This is macro HCI_CREATE_CONNECTION_CANCEL.

HCI_CREATE_NEW_UNIT_KEY Macro

File

hci.h

C

```
#define HCI_CREATE_NEW_UNIT_KEY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x000B)
```

Description

This is macro HCI_CREATE_NEW_UNIT_KEY.

HCI_CTRL_LISTENER_ACL_DATA Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_LISTENER_ACL_DATA 1
```

Description

This is macro HCI_CTRL_LISTENER_ACL_DATA.

HCI_CTRL_LISTENER_EVENT Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_LISTENER_EVENT 0
```

Description

This is macro HCI_CTRL_LISTENER_EVENT.

HCI_CTRL_LISTENER_SCO_DATA Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_LISTENER_SCO_DATA 2
```

Description

This is macro HCI_CTRL_LISTENER_SCO_DATA.

HCI_CTRL_STATE_CLOSED Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_CLOSED 0x00
```

Description

This is macro HCI_CTRL_STATE_CLOSED.

HCI_CTRL_STATE_CONNECTABLE Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_CONNECTABLE 0x20
```

Description

This is macro HCI_CTRL_STATE_CONNECTABLE.

HCI_CTRL_STATE_DISCOVERABLE Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_DISCOVERABLE 0x10
```

Description

This is macro HCI_CTRL_STATE_DISCOVERABLE.

HCI_CTRL_STATE_INIT Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_INIT 0x01
```

Description

This is macro HCI_CTRL_STATE_INIT.

HCI_CTRL_STATE_READY Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_READY 0x02
```

Description

This is macro HCI_CTRL_STATE_READY.

HCI_CTRL_STATE_SLEEP Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_SLEEP 0x04
```

Description

This is macro HCI_CTRL_STATE_SLEEP.

HCI_CTRL_STATE_WAKING_UP Macro

File

hci_ctrl_state.h

C

```
#define HCI_CTRL_STATE_WAKING_UP 0x08
```

Description

This is macro HCI_CTRL_STATE_WAKING_UP.

HCI_DATA_BUFFER_STATE_FREE Macro

File

hci_data_buffer.h

C

```
#define HCI_DATA_BUFFER_STATE_FREE 0
```

Description

This is macro HCI_DATA_BUFFER_STATE_FREE.

HCI_DATA_BUFFER_STATE_USED Macro

File

hci_data_buffer.h

C

```
#define HCI_DATA_BUFFER_STATE_USED 1
```

Description

This is macro HCI_DATA_BUFFER_STATE_USED.

HCI_DATA_STATUS_BEING_SENT Macro

File

hci.h

C

```
#define HCI_DATA_STATUS_BEING_SENT 2
```

Description

This is macro HCI_DATA_STATUS_BEING_SENT.

HCI_DATA_STATUS_PENDING Macro

File

hci.h

C

```
#define HCI_DATA_STATUS_PENDING 0
```

Description

This is macro HCI_DATA_STATUS_PENDING.

HCI_DEFAULT_ACL_CONFIG Macro

File

hci.h

C

```
#define HCI_DEFAULT_ACL_CONFIG 0
```

Description

Default value for acl_config parameter of bt_hci_connect()

HCI_DELETE_STORED_LINK_KEY Macro

File

hci.h

C

```
#define HCI_DELETE_STORED_LINK_KEY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0012)
```

Description

This is macro HCI_DELETE_STORED_LINK_KEY.

HCI_DISCONNECT Macro

File

hci.h

C

```
#define HCI_DISCONNECT HCI_OPCODE(OGF_LINK_CONTROL, 0x0006)
```

Description

This is macro HCI_DISCONNECT.

HCI_DISCOVERABLE Macro

File

hci.h

C

```
#define HCI_DISCOVERABLE 0x01
```

Description

This is macro HCI_DISCOVERABLE.

HCI_EIR_FEC_NOT_REQUIRED Macro

File

hci_eir.h

C

```
#define HCI_EIR_FEC_NOT_REQUIRED 0
```

Description

This is macro HCI_EIR_FEC_NOT_REQUIRED.

HCI_EIR_FEC_REQUIRED Macro

File

hci_eir.h

C

```
#define HCI_EIR_FEC_REQUIRED 1
```

Description

This is macro HCI_EIR_FEC_REQUIRED.

HCI_EIR_TYPE_DEVICE_ID Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_DEVICE_ID 0x10
```

Description

This is macro HCI_EIR_TYPE_DEVICE_ID.

HCI_EIR_TYPE_FLAGS Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_FLAGS 0x01
```

Description

This is macro HCI_EIR_TYPE_FLAGS.

HCI_EIR_TYPE_LOCAL_NAME_COMPLETE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_LOCAL_NAME_COMPLETE 0x09
```

Description

This is macro HCI_EIR_TYPE_LOCAL_NAME_COMPLETE.

HCI_EIR_TYPE_LOCAL_NAME_SHORTENED Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_LOCAL_NAME_SHORTENED 0x08
```

Description

This is macro HCI_EIR_TYPE_LOCAL_NAME_SHORTENED.

HCI_EIR_TYPE_MANUFACTURER_SPECIFIC Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_MANUFACTURER_SPECIFIC 0xFF
```

Description

This is macro HCI_EIR_TYPE_MANUFACTURER_SPECIFIC.

HCI_EIR_TYPE_OOB_COD Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_OOB_COD 0x0d
```

Description

This is macro HCI_EIR_TYPE_OOB_COD.

HCI_EIR_TYPE_OOB_HASH Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_OOB_HASH 0x0e
```

Description

This is macro HCI_EIR_TYPE_OOB_HASH.

HCI_EIR_TYPE_OOB_RANDOMIZER Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_OOB_RANDOMIZER 0x0f
```

Description

This is macro HCI_EIR_TYPE_OOB_RANDOMIZER.

HCI_EIR_TYPE_TX_POWER_LEVEL Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_TX_POWER_LEVEL 0x0a
```

Description

This is macro HCI_EIR_TYPE_TX_POWER_LEVEL.

HCI_EIR_TYPE_UUID128_LIST_COMPLETE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_UUID128_LIST_COMPLETE 0x07
```

Description

This is macro HCI_EIR_TYPE_UUID128_LIST_COMPLETE.

HCI_EIR_TYPE_UUID128_LIST_MORE_AVAILABLE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_UUID128_LIST_MORE_AVAILABLE 0x06
```

Description

This is macro HCI_EIR_TYPE_UUID128_LIST_MORE_AVAILABLE.

HCI_EIR_TYPE_UUID16_LIST_COMPLETE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_UUID16_LIST_COMPLETE 0x03
```

Description

This is macro HCI_EIR_TYPE_UUID16_LIST_COMPLETE.

HCI_EIR_TYPE_UUID16_LIST_MORE_AVAILABLE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_UUID16_LIST_MORE_AVAILABLE 0x02
```

Description

This is macro HCI_EIR_TYPE_UUID16_LIST_MORE_AVAILABLE.

HCI_EIR_TYPE_UUID32_LIST_COMPLETE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_UUID32_LIST_COMPLETE 0x05
```

Description

This is macro HCI_EIR_TYPE_UUID32_LIST_COMPLETE.

HCI_EIR_TYPE_UUID32_LIST_MORE_AVAILABLE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_UUID32_LIST_MORE_AVAILABLE 0x04
```

Description

This is macro HCI_EIR_TYPE_UUID32_LIST_MORE_AVAILABLE.

HCI_ENABLE_DEVICE_UNDER_TEST_MODE Macro

File

hci.h

C

```
#define HCI_ENABLE_DEVICE_UNDER_TEST_MODE HCI_OPCODE(OGF_TESTING, 0x0003)
```

Description

This is macro HCI_ENABLE_DEVICE_UNDER_TEST_MODE.

HCI_ENCRYPTION_OFF Macro

File

hci.h

C

```
#define HCI_ENCRYPTION_OFF 0x00
```

Description

This is macro HCI_ENCRYPTION_OFF.

HCI_ENCRYPTION_ON Macro

File

hci.h

C

```
#define HCI_ENCRYPTION_ON 0x01
```

Description

This is macro HCI_ENCRYPTION_ON.

HCI_ENHANCED_FLUSH Macro

File

hci.h

C

```
#define HCI_ENHANCED_FLUSH HCI_OPCODE(OGF_CTRL_BASEBAND, 0x005F)
```

Description

This is macro HCI_ENHANCED_FLUSH.

HCI_ERR_ACL_CONN_ALREADY_EXISTS Macro

File

hci_errors.h

C

```
#define HCI_ERR_ACL_CONN_ALREADY_EXISTS 0x0b
```

Description

This is macro HCI_ERR_ACL_CONN_ALREADY_EXISTS.

HCI_ERR_AUTHENTICATION_FAILURE Macro

File

hci_errors.h

C

```
#define HCI_ERR_AUTHENTICATION_FAILURE 0x05
```

Description

This is macro HCI_ERR_AUTHENTICATION_FAILURE.

HCI_ERR_CONN_REJECT_LIMITED_RESOURCES Macro

File

hci_errors.h

C

```
#define HCI_ERR_CONN_REJECT_LIMITED_RESOURCES 0x0d
```

Description

This is macro HCI_ERR_CONN_REJECT_LIMITED_RESOURCES.

HCI_ERR_INVALID_PARAMETERS Macro

File

hci_errors.h

C

```
#define HCI_ERR_INVALID_PARAMETERS 0x12
```

Description

This is macro HCI_ERR_INVALID_PARAMETERS.

HCI_ERR_MEMORY_CAPACITY_EXCEEDED Macro

File

hci_errors.h

C

```
#define HCI_ERR_MEMORY_CAPACITY_EXCEEDED 0x07
```

Description

This is macro HCI_ERR_MEMORY_CAPACITY_EXCEEDED.

HCI_ERR_SCO_CONN_LIMIT_EXCEEDED Macro

File

hci_errors.h

C

```
#define HCI_ERR_SCO_CONN_LIMIT_EXCEEDED 0x0a
```

Description

This is macro HCI_ERR_SCO_CONN_LIMIT_EXCEEDED.

HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED Macro

File

hci_errors.h

C

```
#define HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED 0x37
```

Description

This is macro HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED.

HCI_ERR_SUCCESS Macro

File

hci_errors.h

C

```
#define HCI_ERR_SUCCESS 0x00
```

Description

- addtogroup hci
- @{
- @name Errors

HCI_ERR_UNSPECIFIED Macro

File

hci_errors.h

C

```
#define HCI_ERR_UNSPECIFIED 0x1F
```

Description

This is macro HCI_ERR_UNSPECIFIED.

HCI_EVT_ALL_HCI_EVENTS Macro

File

hci.h

C

```
#define HCI_EVT_ALL_HCI_EVENTS 0xffff
```

Description

Used in bt_hci_ctrl_register_event_listener to indicate that the listener is called for all HCI events (excluding dotstack internal ones)

HCI_EVT_AUTHENTICATION_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_AUTHENTICATION_COMPLETE 0x06
```

Description

This is macro HCI_EVT_AUTHENTICATION_COMPLETE.

HCI_EVT_CHANGE_CONN_LINK_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_CHANGE_CONN_LINK_COMPLETE 0x09
```

Description

This is macro HCI_EVT_CHANGE_CONN_LINK_COMPLETE.

HCI_EVT_CMD_SEND_FINISHED Macro

File

hci.h

C

```
#define HCI_EVT_CMD_SEND_FINISHED 0xff01
```

Description

This is macro HCI_EVT_CMD_SEND_FINISHED.

HCI_EVT_CMD_SEND_STARTED Macro

File

hci.h

C

```
#define HCI_EVT_CMD_SEND_STARTED 0xff00
```

Description

dotstack internal events

HCI_EVT_COMMAND_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_COMMAND_COMPLETE 0x0E
```

Description

This is macro HCI_EVT_COMMAND_COMPLETE.

HCI_EVT_COMMAND_COMPLETE_PARAM_LEN Macro

File

hci.h

C

```
#define HCI_EVT_COMMAND_COMPLETE_PARAM_LEN 3
```

Description

This is macro HCI_EVT_COMMAND_COMPLETE_PARAM_LEN.

HCI_EVT_COMMAND_STATUS Macro

File

hci.h

C

```
#define HCI_EVT_COMMAND_STATUS 0x0F
```

Description

This is macro HCI_EVT_COMMAND_STATUS.

HCI_EVT_CONN_PACKET_TYPE_CHANGED Macro

File

hci.h

C

```
#define HCI_EVT_CONN_PACKET_TYPE_CHANGED 0x1D
```

Description

This is macro HCI_EVT_CONN_PACKET_TYPE_CHANGED.

HCI_EVT_CONNECTION_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_CONNECTION_COMPLETE 0x03
```

Description

This is macro HCI_EVT_CONNECTION_COMPLETE.

HCI_EVT_CONNECTION_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_CONNECTION_REQUEST 0x04
```

Description

This is macro HCI_EVT_CONNECTION_REQUEST.

HCI_EVT_DATA_BUFFER_OVERFLOW Macro

File

hci.h

C

```
#define HCI_EVT_DATA_BUFFER_OVERFLOW 0x1A
```

Description

This is macro HCI_EVT_DATA_BUFFER_OVERFLOW.

HCI_EVT_DISCONNECTION_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_DISCONNECTION_COMPLETE 0x05
```

Description

This is macro HCI_EVT_DISCONNECTION_COMPLETE.

HCI_EVT_ENCRYPTION_CHANGE Macro

File

hci.h

C

```
#define HCI_EVT_ENCRYPTION_CHANGE 0x08
```

Description

This is macro HCI_EVT_ENCRYPTION_CHANGE.

HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE 0x30
```

Description

This is macro HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE.

HCI_EVT_ENHANCED_FLUSH_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_ENHANCED_FLUSH_COMPLETE 0x39
```

Description

This is macro HCI_EVT_ENHANCED_FLUSH_COMPLETE.

HCI_EVT_EXTENDED_INQUIRY_RESULT Macro

File

hci.h

C

```
#define HCI_EVT_EXTENDED_INQUIRY_RESULT 0x2F
```

Description

This is macro HCI_EVT_EXTENDED_INQUIRY_RESULT.

HCI_EVT_FIRST Macro

File

hci.h

C

```
#define HCI_EVT_FIRST HCI_EVT_INQUIRY_COMPLETE
```

Description

This is macro HCI_EVT_FIRST.

HCI_EVT_FLOW_SPECIFICATION_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_FLOW_SPECIFICATION_COMPLETE 0x21
```

Description

This is macro HCI_EVT_FLOW_SPECIFICATION_COMPLETE.

HCI_EVT_FLUSH_OCCURED Macro

File

hci.h

C

```
#define HCI_EVT_FLUSH_OCCURED 0x11
```

Description

This is macro HCI_EVT_FLUSH_OCCURED.

HCI_EVT_HARDWARE_ERROR Macro

File

hci.h

C

```
#define HCI_EVT_HARDWARE_ERROR 0x10
```

Description

This is macro HCI_EVT_HARDWARE_ERROR.

HCI_EVT_INQUIRY_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_INQUIRY_COMPLETE 0x01
```

Description

- addtogroup hci
- @{
- @name Events

HCI_EVT_INQUIRY_RESULT Macro

File

hci.h

C

```
#define HCI_EVT_INQUIRY_RESULT 0x02
```

Description

This is macro HCI_EVT_INQUIRY_RESULT.

HCI_EVT_INQUIRY_RESULT_WITH_RSSI Macro

File

hci.h

C

```
#define HCI_EVT_INQUIRY_RESULT_WITH_RSSI 0x22
```

Description

This is macro HCI_EVT_INQUIRY_RESULT_WITH_RSSI.

HCI_EVT_IO_CAPABILITY_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_IO_CAPABILITY_REQUEST 0x31
```

Description

This is macro HCI_EVT_IO_CAPABILITY_REQUEST.

HCI_EVT_IO_CAPABILITY_RESPONSE Macro

File

hci.h

C

```
#define HCI_EVT_IO_CAPABILITY_RESPONSE 0x32
```

Description

This is macro HCI_EVT_IO_CAPABILITY_RESPONSE.

HCI_EVT_KEYPRESS_NOTIFICATION Macro

File

hci.h

C

```
#define HCI_EVT_KEYPRESS_NOTIFICATION 0x3C
```

Description

This is macro HCI_EVT_KEYPRESS_NOTIFICATION.

HCI_EVT_LAST Macro

File

hci.h

C

```
#define HCI_EVT_LAST HCI_EVT_LE_META_EVENT
```

Description

This is macro HCI_EVT_LAST.

HCI_EVT_LE_META_EVENT Macro

File

hci.h

C

```
#define HCI_EVT_LE_META_EVENT 0x3E
```

Description

This is macro HCI_EVT_LE_META_EVENT.

HCI_EVT_LINK_IS_BUSY Macro

File

hci.h

C

```
#define HCI_EVT_LINK_IS_BUSY 0xff03
```

Description

This is macro HCI_EVT_LINK_IS_BUSY.

HCI_EVT_LINK_IS_IDLE Macro

File

hci.h

C

```
#define HCI_EVT_LINK_IS_IDLE 0xff02
```

Description

This is macro HCI_EVT_LINK_IS_IDLE.

HCI_EVT_LINK_KEY_NOTIFICATION Macro

File

hci.h

C

```
#define HCI_EVT_LINK_KEY_NOTIFICATION 0x18
```

Description

This is macro HCI_EVT_LINK_KEY_NOTIFICATION.

HCI_EVT_LINK_KEY_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_LINK_KEY_REQUEST 0x17
```

Description

This is macro HCI_EVT_LINK_KEY_REQUEST.

HCI_EVT_LINK_SUPERVISION_TO_CHANGED Macro

File

hci.h

C

```
#define HCI_EVT_LINK_SUPERVISION_TO_CHANGED 0x38
```

Description

This is macro HCI_EVT_LINK_SUPERVISION_TO_CHANGED.

HCI_EVT_LOOPBACK_COMMAND Macro

File

hci.h

C

```
#define HCI_EVT_LOOPBACK_COMMAND 0x19
```

Description

This is macro HCI_EVT_LOOPBACK_COMMAND.

HCI_EVT_MASTER_LINK_KEY_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_MASTER_LINK_KEY_COMPLETE 0x0A
```

Description

This is macro HCI_EVT_MASTER_LINK_KEY_COMPLETE.

HCI_EVT_MAX_SLOTS_CHANGE Macro

File

hci.h

C

```
#define HCI_EVT_MAX_SLOTS_CHANGE 0x1B
```

Description

This is macro HCI_EVT_MAX_SLOTS_CHANGE.

HCI_EVT_MODE_CHANGE Macro

File

hci.h

C

```
#define HCI_EVT_MODE_CHANGE 0x14
```

Description

This is macro HCI_EVT_MODE_CHANGE.

HCI_EVT_NUM_OF_COMPLETED_PACKETS Macro

File

hci.h

C

```
#define HCI_EVT_NUM_OF_COMPLETED_PACKETS 0x13
```

Description

This is macro HCI_EVT_NUM_OF_COMPLETED_PACKETS.

HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE Macro

File

hci.h

C

```
#define HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE 0x20
```

Description

This is macro HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE.

HCI_EVT_PIN_CODE_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_PIN_CODE_REQUEST 0x16
```

Description

This is macro HCI_EVT_PIN_CODE_REQUEST.

HCI_EVT_QOS_SETUP_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_QOS_SETUP_COMPLETE 0x0D
```

Description

This is macro HCI_EVT_QOS_SETUP_COMPLETE.

HCI_EVT_QOS_VIOLATION Macro

File

hci.h

C

```
#define HCI_EVT_QOS_VIOLATION 0x1E
```

Description

This is macro HCI_EVT_QOS_VIOLATION.

HCI_EVT_READ_CLOCK_OFFSET_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_READ_CLOCK_OFFSET_COMPLETE 0x1C
```

Description

This is macro HCI_EVT_READ_CLOCK_OFFSET_COMPLETE.

HCI_EVT_READ_RMT_EXT_FEATURES_COMP Macro

File

hci.h

C

```
#define HCI_EVT_READ_RMT_EXT_FEATURES_COMP 0x23
```

Description

This is macro HCI_EVT_READ_RMT_EXT_FEATURES_COMP.

HCI_EVT_READ_RMT_SUP_FEATURES_COMP Macro

File

hci.h

C

```
#define HCI_EVT_READ_RMT_SUP_FEATURES_COMP 0x0B
```

Description

This is macro HCI_EVT_READ_RMT_SUP_FEATURES_COMP.

HCI_EVT_READ_RMT_VERSION_INFO_COMP Macro

File

hci.h

C

```
#define HCI_EVT_READ_RMT_VERSION_INFO_COMP 0x0C
```

Description

This is macro HCI_EVT_READ_RMT_VERSION_INFO_COMP.

HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE 0x07
```

Description

This is macro HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE.

HCI_EVT_REMOTE_OOB_DATA_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_REMOTE_OOB_DATA_REQUEST 0x35
```

Description

This is macro HCI_EVT_REMOTE_OOB_DATA_REQUEST.

HCI_EVT_RETURN_LINK_KEYS Macro

File

hci.h

C

```
#define HCI_EVT_RETURN_LINK_KEYS 0x15
```

Description

This is macro HCI_EVT_RETURN_LINK_KEYS.

HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF Macro

File

hci.h

C

```
#define HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF 0x3D
```

Description

This is macro HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF.

HCI_EVT_ROLE_CHANGE Macro

File

hci.h

C

```
#define HCI_EVT_ROLE_CHANGE 0x12
```

Description

This is macro HCI_EVT_ROLE_CHANGE.

HCI_EVT_SIMPLE_PAIRING_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_SIMPLE_PAIRING_COMPLETE 0x36
```

Description

This is macro HCI_EVT_SIMPLE_PAIRING_COMPLETE.

HCI_EVT_SNIFF_SUBRATING Macro

File

hci.h

C

```
#define HCI_EVT_SNIFF_SUBRATING 0x2E
```

Description

This is macro HCI_EVT_SNIFF_SUBRATING.

HCI_EVT_SYNCH_CONNECTION_CHANGED Macro

File

hci.h

C

```
#define HCI_EVT_SYNCH_CONNECTION_CHANGED 0x2D
```

Description

This is macro HCI_EVT_SYNCH_CONNECTION_CHANGED.

HCI_EVT_SYNCH_CONNECTION_COMPLETE Macro

File

hci.h

C

```
#define HCI_EVT_SYNCH_CONNECTION_COMPLETE 0x2C
```

Description

This is macro HCI_EVT_SYNCH_CONNECTION_COMPLETE.

HCI_EVT_USER_CONFIRMATION_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_USER_CONFIRMATION_REQUEST 0x33
```

Description

This is macro HCI_EVT_USER_CONFIRMATION_REQUEST.

HCI_EVT_USER_PASSKEY_NOTIFICATION Macro

File

hci.h

C

```
#define HCI_EVT_USER_PASSKEY_NOTIFICATION 0x3B
```

Description

This is macro HCI_EVT_USER_PASSKEY_NOTIFICATION.

HCI_EVT_USER_PASSKEY_REQUEST Macro

File

hci.h

C

```
#define HCI_EVT_USER_PASSKEY_REQUEST 0x34
```

Description

This is macro HCI_EVT_USER_PASSKEY_REQUEST.

HCI_EXIT_PARK_STATE Macro

File

hci.h

C

```
#define HCI_EXIT_PARK_STATE HCI_OPCODE(OGF_LINK_POLICY, 0x0006)
```

Description

This is macro HCI_EXIT_PARK_STATE.

HCI_EXIT_PERIODIC_INQUIRY_MODE Macro

File

hci.h

C

```
#define HCI_EXIT_PERIODIC_INQUIRY_MODE HCI_OPCODE(OGF_LINK_CONTROL, 0x0004)
```

Description

This is macro HCI_EXIT_PERIODIC_INQUIRY_MODE.

HCI_EXIT_SNIFF_MODE Macro

File

hci.h

C

```
#define HCI_EXIT_SNIFF_MODE HCI_OPCODE(OGF_LINK_POLICY, 0x0004)
```

Description

This is macro HCI_EXIT_SNIFF_MODE.

HCI_FLOW_SPECIFICATION Macro

File

hci.h

C

```
#define HCI_FLOW_SPECIFICATION HCI_OPCODE(OGF_LINK_POLICY, 0x0010)
```

Description

This is macro HCI_FLOW_SPECIFICATION.

HCI_FLUSH Macro

File

hci.h

C

```
#define HCI_FLUSH HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0008)
```

Description

This is macro HCI_FLUSH.

HCI_H2C_BROADCAST_ACTIVE_SLAVE Macro

File

hci.h

C

```
#define HCI_H2C_BROADCAST_ACTIVE_SLAVE 0x1
```

Description

This is macro HCI_H2C_BROADCAST_ACTIVE_SLAVE.

HCI_H2C_BROADCAST_NO_BROADCATS Macro

File

hci.h

C

```
#define HCI_H2C_BROADCAST_NO_BROADCATS 0x0
```

Description

This is macro HCI_H2C_BROADCAST_NO_BROADCATS.

HCI_H2C_BROADCAST_PARCKED_SLAVE Macro

File

hci.h

C

```
#define HCI_H2C_BROADCAST_PARCKED_SLAVE 0x2
```

Description

This is macro HCI_H2C_BROADCAST_PARCKED_SLAVE.

HCI_H2C_BROADCAST_RESERVED Macro

File

hci.h

C

```
#define HCI_H2C_BROADCAST_RESERVED 0x3
```

Description

This is macro HCI_H2C_BROADCAST_RESERVED.

HCI_HOLD_MODE Macro

File

hci.h

C

```
#define HCI_HOLD_MODE HCI_OPCODE(OGF_LINK_POLICY, 0x0001)
```

Description

addtogroup hci @{

@name Link policy commands

details The Link Policy Commands provide methods for the Host to affect how the Link Manager manages the piconet.

HCI_HOST_BUFFER_SIZE Macro

File

hci.h

C

```
#define HCI_HOST_BUFFER_SIZE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0033)
```

Description

This is macro HCI_HOST_BUFFER_SIZE.

HCI_HOST_NUM_OF_COMPLETED_PACKETS Macro

File

hci.h

C

```
#define HCI_HOST_NUM_OF_COMPLETED_PACKETS HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0035)
```

Description

This is macro HCI_HOST_NUM_OF_COMPLETED_PACKETS.

HCI_INQUIRY Macro

File

hci.h

C

```
#define HCI_INQUIRY HCI_OPCODE(OGF_LINK_CONTROL, 0x0001)
```

Description

addtogroup hci @{
@name Link control commands
details The Link Control commands allow a Controller to control connections to other BR/EDR Controllers.

HCI_INQUIRY_CANCEL Macro

File

hci.h

C

```
#define HCI_INQUIRY_CANCEL HCI_OPCODE(OGF_LINK_CONTROL, 0x0002)
```

Description

This is macro HCI_INQUIRY_CANCEL.

HCI_INQUIRY_MODE_EXTENDED Macro

File

hci.h

C

```
#define HCI_INQUIRY_MODE_EXTENDED 2
```

Description

This is macro HCI_INQUIRY_MODE_EXTENDED.

HCI_INQUIRY_MODE_STANDARD Macro

File

hci.h

C

```
#define HCI_INQUIRY_MODE_STANDARD 0
```

Description

This is macro HCI_INQUIRY_MODE_STANDARD.

HCI_INQUIRY_MODE_WITH_RSSI Macro

File

hci.h

C

```
#define HCI_INQUIRY_MODE_WITH_RSSI 1
```

Description

This is macro HCI_INQUIRY_MODE_WITH_RSSI.

HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x0034)
```

Description

This is macro HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY.

HCI_IO_CAPABILITY_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_IO_CAPABILITY_REQUEST_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x002B)
```

Description

This is macro HCI_IO_CAPABILITY_REQUEST_REPLY.

HCI_LE_ADD_DEVICE_TO_WHITE_LIST Macro

File

hci.h

C

```
#define HCI_LE_ADD_DEVICE_TO_WHITE_LIST HCI_OPCODE(OGF_LE, 0x0011)
```

Description

This is macro HCI_LE_ADD_DEVICE_TO_WHITE_LIST.

HCI_LE_ADDRESS_TYPE_PUBLIC Macro

File

hci_le.h

C

```
#define HCI_LE_ADDRESS_TYPE_PUBLIC 0
```

Description

This is macro HCI_LE_ADDRESS_TYPE_PUBLIC.

HCI_LE_ADDRESS_TYPE_RANDOM Macro

File

hci_le.h

C

```
#define HCI_LE_ADDRESS_TYPE_RANDOM 1
```

Description

This is macro HCI_LE_ADDRESS_TYPE_RANDOM.

HCI_LE_ADVERTISING_FLAG_BREDR_NOT_SUPPORTED Macro

File

hci_le.h

C

```
#define HCI_LE_ADVERTISING_FLAG_BREDR_NOT_SUPPORTED 0x04
```

Description

This is macro HCI_LE_ADVERTISING_FLAG_BREDR_NOT_SUPPORTED.

HCI_LE_ADVERTISING_FLAG_GENERAL_DISCOVERABLE_MODE Macro

File

hci_le.h

C

```
#define HCI_LE_ADVERTISING_FLAG_GENERAL_DISCOVERABLE_MODE 0x02
```

Description

This is macro HCI_LE_ADVERTISING_FLAG_GENERAL_DISCOVERABLE_MODE.

HCI_LE_ADVERTISING_FLAG_LIMITED_DISCOVERABLE_MODE Macro

File

hci_le.h

C

```
#define HCI_LE_ADVERTISING_FLAG_LIMITED_DISCOVERABLE_MODE 0x01
```

Description

This is macro HCI_LE_ADVERTISING_FLAG_LIMITED_DISCOVERABLE_MODE.

HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_CONTROLLER Macro

File

hci_le.h

C

```
#define HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_CONTROLLER 0x08
```

Description

This is macro HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_CONTROLLER.

HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_HOST Macro

File

hci_le.h

C

```
#define HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_HOST 0x10
```

Description

This is macro HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_HOST.

HCI_LE_AVERTISING_DISABLED Macro

File

hci_le.h

C

```
#define HCI_LE_AVERTISING_DISABLED 0
```

Description

This is macro HCI_LE_AVERTISING_DISABLED.

HCI_LE_AVERTISING_ENABLED Macro

File

hci_le.h

C

```
#define HCI_LE_AVERTISING_ENABLED 1
```

Description

This is macro HCI_LE_AVERTISING_ENABLED.

HCI_LE_CLEAR_WHITE_LIST Macro

File

hci.h

C

```
#define HCI_LE_CLEAR_WHITE_LIST HCI_OPCODE(OGF_LE, 0x0010)
```

Description

This is macro HCI_LE_CLEAR_WHITE_LIST.

HCI_LE_CONNECTION_UPDATE Macro

File

hci.h

C

```
#define HCI_LE_CONNECTION_UPDATE HCI_OPCODE(OGF_LE, 0x0013)
```

Description

This is macro HCI_LE_CONNECTION_UPDATE.

HCI_LE_CREATE_CONNECTION Macro

File

hci.h

C

```
#define HCI_LE_CREATE_CONNECTION HCI_OPCODE(OGF_LE, 0x000D)
```

Description

This is macro HCI_LE_CREATE_CONNECTION.

HCI_LE_CREATE_CONNECTION_CANCEL Macro

File

hci.h

C

```
#define HCI_LE_CREATE_CONNECTION_CANCEL HCI_OPCODE(OGF_LE, 0x000E)
```

Description

This is macro HCI_LE_CREATE_CONNECTION_CANCEL.

HCI_LE_DISABLED Macro

File

hci_le.h

C

```
#define HCI_LE_DISABLED 0
```

Description

This is macro HCI_LE_DISABLED.

HCI_LE_DUPLICATE_FILTERING_DISABLED Macro

File

hci_le.h

C

```
#define HCI_LE_DUPLICATE_FILTERING_DISABLED 0
```

Description

This is macro HCI_LE_DUPLICATE_FILTERING_DISABLED.

HCI_LE_DUPLICATE_FILTERING_ENABLED Macro

File

hci_le.h

C

```
#define HCI_LE_DUPLICATE_FILTERING_ENABLED 1
```

Description

This is macro HCI_LE_DUPLICATE_FILTERING_ENABLED.

HCI_LE_ENABLED Macro

File

hci_le.h

C

```
#define HCI_LE_ENABLED 1
```

Description

This is macro HCI_LE_ENABLED.

HCI_LE_ENCRYPT Macro

File

hci.h

C

```
#define HCI_LE_ENCRYPT HCI_OPCODE(OGF_LE, 0x0017)
```

Description

This is macro HCI_LE_ENCRYPT.

HCI_LE_EVT_ADVERTISING_REPORT Macro

File

hci_le.h

C

```
#define HCI_LE_EVT_ADVERTISING_REPORT 0x02
```

Description

This is macro HCI_LE_EVT_ADVERTISING_REPORT.

HCI_LE_EVT_CONNECTION_COMPLETE Macro

File

hci_le.h

C

```
#define HCI_LE_EVT_CONNECTION_COMPLETE 0x01
```

Description

This is macro HCI_LE_EVT_CONNECTION_COMPLETE.

HCI_LE_EVT_CONNECTION_UPDATE_COMPLETE Macro

File

hci_le.h

C

```
#define HCI_LE_EVT_CONNECTION_UPDATE_COMPLETE 0x03
```

Description

This is macro HCI_LE_EVT_CONNECTION_UPDATE_COMPLETE.

HCI_LE_EVT_LONG_TERM_KEY_REQUEST Macro

File

hci_le.h

C

```
#define HCI_LE_EVT_LONG_TERM_KEY_REQUEST 0x05
```

Description

This is macro HCI_LE_EVT_LONG_TERM_KEY_REQUEST.

HCI_LE_EVT_READ_REMOTE_USED_FEATURES_COMPLETE Macro

File

hci_le.h

C

```
#define HCI_LE_EVT_READ_REMOTE_USED_FEATURES_COMPLETE 0x04
```

Description

This is macro HCI_LE_EVT_READ_REMOTE_USED_FEATURES_COMPLETE.

HCI_LE_FILTER_POLICY_NOT_USED Macro

File

hci_le.h

C

```
#define HCI_LE_FILTER_POLICY_NOT_USED 0 // White list is not used to determine which advertiser to connect
to.
```

Description

White list is not used to determine which advertiser to connect to. Peer_Address_Type and Peer_Address shall be used.

HCI_LE_FILTER_POLICY_WHITE_LIST Macro

File

hci_le.h

C

```
#define HCI_LE_FILTER_POLICY_WHITE_LIST 1 // White list is used to determine which advertiser to connect to.
```

Description

White list is used to determine which advertiser to connect to. Peer_Address_Type and Peer_Address shall be ignored

HCI_LE_FLAG_SHARED_ACL_BUFFERS Macro

File

hci_le.h

C

```
#define HCI_LE_FLAG_SHARED_ACL_BUFFERS 2
```

Description

This is macro HCI_LE_FLAG_SHARED_ACL_BUFFERS.

HCI_LE_FLAG_SIMULTANEOUS_LE_BREDR Macro

File

hci_le.h

C

```
#define HCI_LE_FLAG_SIMULTANEOUS_LE_BREDR 1
```

Description

This is macro HCI_LE_FLAG_SIMULTANEOUS_LE_BREDR.

HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY HCI_OPCODE(OGF_LE, 0x001B)
```

Description

This is macro HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY.

HCI_LE_LONG_TERM_KEY_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_LE_LONG_TERM_KEY_REQUEST_REPLY HCI_OPCODE(OGF_LE, 0x001A)
```

Description

This is macro HCI_LE_LONG_TERM_KEY_REQUEST_REPLY.

HCI_LE_MAX_AD_LEN Macro

File

hci.h

C

```
#define HCI_LE_MAX_AD_LEN 31
```

Description

This is macro HCI_LE_MAX_AD_LEN.

HCI_LE_RAND Macro

File

hci.h

C

```
#define HCI_LE_RAND HCI_OPCODE(OGF_LE, 0x0018)
```

Description

This is macro HCI_LE_RAND.

HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER Macro

File

hci.h

C

```
#define HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER HCI_OPCODE(OGF_LE, 0x0007)
```

Description

This is macro HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER.

HCI_LE_READ_BUFFER_SIZE Macro

File

hci.h

C

```
#define HCI_LE_READ_BUFFER_SIZE HCI_OPCODE(OGF_LE, 0x0002)
```

Description

This is macro HCI_LE_READ_BUFFER_SIZE.

HCI_LE_READ_CHANNEL_MAP Macro

File

hci.h

C

```
#define HCI_LE_READ_CHANNEL_MAP HCI_OPCODE(OGF_LE, 0x0015)
```

Description

This is macro HCI_LE_READ_CHANNEL_MAP.

HCI_LE_READ_LOCAL_SUPPORTED_FEATURES Macro

File

hci.h

C

```
#define HCI_LE_READ_LOCAL_SUPPORTED_FEATURES HCI_OPCODE(OGF_LE, 0x0003)
```

Description

This is macro HCI_LE_READ_LOCAL_SUPPORTED_FEATURES.

HCI_LE_READ_REMOTE_USED_FEATURES Macro

File

hci.h

C

```
#define HCI_LE_READ_REMOTE_USED_FEATURES HCI_OPCODE(OGF_LE, 0x0016)
```

Description

This is macro HCI_LE_READ_REMOTE_USED_FEATURES.

HCI_LE_READ_SUPPORTED_STATES Macro

File

hci.h

C

```
#define HCI_LE_READ_SUPPORTED_STATES HCI_OPCODE(OGF_LE, 0x001C)
```

Description

This is macro HCI_LE_READ_SUPPORTED_STATES.

HCI_LE_READ_WHITE_LIST_SIZE Macro

File

hci.h

C

```
#define HCI_LE_READ_WHITE_LIST_SIZE HCI_OPCODE(OGF_LE, 0x000F)
```

Description

This is macro HCI_LE_READ_WHITE_LIST_SIZE.

HCI_LE_RECEIVE_TEST Macro

File

hci.h

C

```
#define HCI_LE_RECEIVE_TEST HCI_OPCODE(OGF_LE, 0x001D)
```

Description

This is macro HCI_LE_RECEIVE_TEST.

HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST Macro

File

hci.h

C

```
#define HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST HCI_OPCODE(OGF_LE, 0x0012)
```

Description

This is macro HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST.

HCI_LE_SCAN_DISABLED Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_DISABLED 0
```

Description

This is macro HCI_LE_SCAN_DISABLED.

HCI_LE_SCAN_ENABLED Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_ENABLED 1
```

Description

This is macro HCI_LE_SCAN_ENABLED.

HCI_LE_SCAN_EVT_CANCEL_FAILED Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_EVT_CANCEL_FAILED 4
```

Description

This is macro HCI_LE_SCAN_EVT_CANCEL_FAILED.

HCI_LE_SCAN_EVT_CANCELLED Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_EVT_CANCELLED 1
```

Description

This is macro HCI_LE_SCAN_EVT_CANCELLED.

HCI_LE_SCAN_EVT_DEVICE_FOUND Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_EVT_DEVICE_FOUND 2
```

Description

This is macro HCI_LE_SCAN_EVT_DEVICE_FOUND.

HCI_LE_SCAN_EVT_START_FAILED Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_EVT_START_FAILED 3
```

Description

This is macro HCI_LE_SCAN_EVT_START_FAILED.

HCI_LE_SCAN_EVT_STARTED Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_EVT_STARTED 0
```

Description

This is macro HCI_LE_SCAN_EVT_STARTED.

HCI_LE_SCAN_FILTER_POLICY_ALL Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_FILTER_POLICY_ALL 0
```

Description

This is macro HCI_LE_SCAN_FILTER_POLICY_ALL.

HCI_LE_SCAN_FILTER_POLICY_WHITE_LIST Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_FILTER_POLICY_WHITE_LIST 1
```

Description

This is macro HCI_LE_SCAN_FILTER_POLICY_WHITE_LIST.

HCI_LE_SCAN_TYPE_ACTIVE Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_TYPE_ACTIVE 1
```

Description

This is macro HCI_LE_SCAN_TYPE_ACTIVE.

HCI_LE_SCAN_TYPE_PASSIVE Macro

File

hci_le.h

C

```
#define HCI_LE_SCAN_TYPE_PASSIVE 0
```

Description

This is macro HCI_LE_SCAN_TYPE_PASSIVE.

HCI_LE_SET_ADVERTISE_ENABLE Macro

File

hci.h

C

```
#define HCI_LE_SET_ADVERTISE_ENABLE HCI_OPCODE(OGF_LE, 0x000A)
```

Description

This is macro HCI_LE_SET_ADVERTISE_ENABLE.

HCI_LE_SET_ADVERTISING_DATA Macro

File

hci.h

C

```
#define HCI_LE_SET_ADVERTISING_DATA HCI_OPCODE(OGF_LE, 0x0008)
```

Description

This is macro HCI_LE_SET_ADVERTISING_DATA.

HCI_LE_SET_ADVERTISING_PARAMETERS Macro

File

hci.h

C

```
#define HCI_LE_SET_ADVERTISING_PARAMETERS HCI_OPCODE(OGF_LE, 0x0006)
```

Description

This is macro HCI_LE_SET_ADVERTISING_PARAMETERS.

HCI_LE_SET_EVENT_MASK Macro

File

hci.h

C

```
#define HCI_LE_SET_EVENT_MASK HCI_OPCODE(OGF_LE, 0x0001)
```

Description

addtogroup hci @{

@name LE controller commands

details The LE Controller Commands provide access and control to various capabilities of the Bluetooth hardware, as well as methods for the Host to affect how the Link Layer manages the piconet, and controls connections.

HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION Macro

File

hci.h

C

```
#define HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION HCI_OPCODE(OGF_LE, 0x0014)
```

Description

This is macro HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION.

HCI_LE_SET_RANDOM_ADDRESS Macro

File

hci.h

C

```
#define HCI_LE_SET_RANDOM_ADDRESS HCI_OPCODE(OGF_LE, 0x0005)
```

Description

This is macro HCI_LE_SET_RANDOM_ADDRESS.

HCI_LE_SET_SCAN_ENABLE Macro

File

hci.h

C

```
#define HCI_LE_SET_SCAN_ENABLE HCI_OPCODE(OGF_LE, 0x000C)
```

Description

This is macro HCI_LE_SET_SCAN_ENABLE.

HCI_LE_SET_SCAN_PARAMETERS Macro

File

hci.h

C

```
#define HCI_LE_SET_SCAN_PARAMETERS HCI_OPCODE(OGF_LE, 0x000B)
```

Description

This is macro HCI_LE_SET_SCAN_PARAMETERS.

HCI_LE_SET_SCAN_RESPONSE_DATA Macro

File

hci.h

C

```
#define HCI_LE_SET_SCAN_RESPONSE_DATA HCI_OPCODE(OGF_LE, 0x0009)
```

Description

This is macro HCI_LE_SET_SCAN_RESPONSE_DATA.

HCI_LE_SIMULTANEOUS_DISABLED Macro

File

hci_le.h

C

```
#define HCI_LE_SIMULTANEOUS_DISABLED 0
```

Description

This is macro HCI_LE_SIMULTANEOUS_DISABLED.

HCI_LE_SIMULTANEOUS_ENABLED Macro

File

hci_le.h

C

```
#define HCI_LE_SIMULTANEOUS_ENABLED 1
```

Description

This is macro HCI_LE_SIMULTANEOUS_ENABLED.

HCI_LE_START_ENCRYPTION Macro

File

hci.h

C

```
#define HCI_LE_START_ENCRYPTION HCI_OPCODE(OGF_LE, 0x0019)
```

Description

This is macro HCI_LE_START_ENCRYPTION.

HCI_LE_TEST_END Macro

File

hci.h

C

```
#define HCI_LE_TEST_END HCI_OPCODE(OGF_LE, 0x001F)
```

Description

This is macro HCI_LE_TEST_END.

HCI_LE_TRANSMITTER_TEST Macro

File

hci.h

C

```
#define HCI_LE_TRANSMITTER_TEST HCI_OPCODE(OGF_LE, 0x001E)
```

Description

This is macro HCI_LE_TRANSMITTER_TEST.

HCI_LINK_KEY_LEN Macro

File

hci.h

C

```
#define HCI_LINK_KEY_LEN 16
```

Description

This is macro HCI_LINK_KEY_LEN.

HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x000C)
```

Description

This is macro HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY.

HCI_LINK_KEY_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_LINK_KEY_REQUEST_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x000B)
```

Description

This is macro HCI_LINK_KEY_REQUEST_REPLY.

HCI_LINK_KEY_TYPE_COMBINATION Macro

File

hci.h

C

```
#define HCI_LINK_KEY_TYPE_COMBINATION 0x00
```

Description

This is macro HCI_LINK_KEY_TYPE_COMBINATION.

HCI_LINK_KEY_TYPE_LOCAL_UNIT Macro

File

hci.h

C

```
#define HCI_LINK_KEY_TYPE_LOCAL_UNIT 0x01
```

Description

This is macro HCI_LINK_KEY_TYPE_LOCAL_UNIT.

HCI_LINK_KEY_TYPE_REMOTE_UNIT Macro

File

hci.h

C

```
#define HCI_LINK_KEY_TYPE_REMOTE_UNIT 0x02
```

Description

This is macro HCI_LINK_KEY_TYPE_REMOTE_UNIT.

HCI_LINK_POLICY_ENABLE_HOLD_MODE Macro

File

hci.h

C

```
#define HCI_LINK_POLICY_ENABLE_HOLD_MODE 0x0002
```

Description

This is macro HCI_LINK_POLICY_ENABLE_HOLD_MODE.

HCI_LINK_POLICY_ENABLE_PARK_STATE Macro

File

hci.h

C

```
#define HCI_LINK_POLICY_ENABLE_PARK_STATE 0x0008
```

Description

This is macro HCI_LINK_POLICY_ENABLE_PARK_STATE.

HCI_LINK_POLICY_ENABLE_ROLE_SWITCH Macro

File

hci.h

C

```
#define HCI_LINK_POLICY_ENABLE_ROLE_SWITCH 0x0001
```

Description

This is macro HCI_LINK_POLICY_ENABLE_ROLE_SWITCH.

HCI_LINK_POLICY_ENABLE_SNIFF_MODE Macro

File

hci.h

C

```
#define HCI_LINK_POLICY_ENABLE_SNIFF_MODE 0x0004
```

Description

This is macro HCI_LINK_POLICY_ENABLE_SNIFF_MODE.

HCI_LINK_TYPE_BD_EDR Macro

File

hci_conn_state.h

C

```
#define HCI_LINK_TYPE_BD_EDR 0
```

Description

This is macro HCI_LINK_TYPE_BD_EDR.

HCI_LINK_TYPE_LE Macro

File

hci_conn_state.h

C

```
#define HCI_LINK_TYPE_LE 1
```

Description

This is macro HCI_LINK_TYPE_LE.

HCI_MASTER_LINK_KEY Macro

File

hci.h

C

```
#define HCI_MASTER_LINK_KEY HCI_OPCODE(OGF_LINK_CONTROL, 0x0017)
```

Description

This is macro HCI_MASTER_LINK_KEY.

HCI_MAX_DATA_BUFFERS Macro

File

hci_data_buffer.h

C

```
#define HCI_MAX_DATA_BUFFERS 2
```

Description

This is macro HCI_MAX_DATA_BUFFERS.

HCI_MAX_EVENT_PARAM_LEN Macro

File

hci.h

C

```
#define HCI_MAX_EVENT_PARAM_LEN 255
```

Description

This is macro HCI_MAX_EVENT_PARAM_LEN.

HCI_MAX_PARAM_LEN Macro

File

hci.h

C

```
#define HCI_MAX_PARAM_LEN 248
```

Description

This is macro HCI_MAX_PARAM_LEN.

HCI_MAX_PIN_LENGTH Macro

File

hci.h

C

```
#define HCI_MAX_PIN_LENGTH 8
```

Description

This is macro HCI_MAX_PIN_LENGTH.

HCI_OPCODE Macro

File

hci.h

C

```
#define HCI_OPCODE(ogf, ocf) (((int)ogf << 10) | ((int)ocf & 0x3ff))
```

Description

This is macro HCI_OPCODE.

HCI_PACKET_BOUNDARY_CONTINUE Macro

File

hci.h

C

```
#define HCI_PACKET_BOUNDARY_CONTINUE 0x1
```

Description

This is macro HCI_PACKET_BOUNDARY_CONTINUE.

HCI_PACKET_BOUNDARY_FIRST Macro

File

hci.h

C

```
#define HCI_PACKET_BOUNDARY_FIRST 0x2
```

Description

This is macro HCI_PACKET_BOUNDARY_FIRST.

HCI_PACKET_BOUNDARY_FIRST_AUTO_FLUSH Macro

File

hci.h

C

```
#define HCI_PACKET_BOUNDARY_FIRST_AUTO_FLUSH 0x2
```

Description

This is macro HCI_PACKET_BOUNDARY_FIRST_AUTO_FLUSH.

HCI_PACKET_BOUNDARY_FIRST_NO_AUTO_FLUSH Macro

File

hci.h

C

```
#define HCI_PACKET_BOUNDARY_FIRST_NO_AUTO_FLUSH 0x0
```

Description

This is macro HCI_PACKET_BOUNDARY_FIRST_NO_AUTO_FLUSH.

HCI_PACKET_TYPE_ACL_DATA Macro

File

hci.h

C

```
#define HCI_PACKET_TYPE_ACL_DATA 0x02
```

Description

This is macro HCI_PACKET_TYPE_ACL_DATA.

HCI_PACKET_TYPE_COMMAND Macro

File

hci.h

C

```
#define HCI_PACKET_TYPE_COMMAND 0x01
```

Description

This is macro HCI_PACKET_TYPE_COMMAND.

HCI_PACKET_TYPE_EVENT Macro

File

hci.h

C

```
#define HCI_PACKET_TYPE_EVENT 0x04
```

Description

This is macro HCI_PACKET_TYPE_EVENT.

HCI_PACKET_TYPE_NONE Macro

File

hci.h

C

```
#define HCI_PACKET_TYPE_NONE 0x00
```

Description

This is macro HCI_PACKET_TYPE_NONE.

HCI_PACKET_TYPE_SCO_DATA Macro

File

hci.h

C

```
#define HCI_PACKET_TYPE_SCO_DATA 0x03
```

Description

This is macro HCI_PACKET_TYPE_SCO_DATA.

HCI_PAGE_SCAN_REPETITION_MODE_R0 Macro

File

hci.h

C

```
#define HCI_PAGE_SCAN_REPETITION_MODE_R0 0x00
```

Description

This is macro HCI_PAGE_SCAN_REPETITION_MODE_R0.

HCI_PAGE_SCAN_REPETITION_MODE_R1 Macro

File

hci.h

C

```
#define HCI_PAGE_SCAN_REPETITION_MODE_R1 0x01
```

Description

This is macro HCI_PAGE_SCAN_REPETITION_MODE_R1.

HCI_PAGE_SCAN_REPETITION_MODE_R2 Macro

File

hci.h

C

```
#define HCI_PAGE_SCAN_REPETITION_MODE_R2 0x02
```

Description

This is macro HCI_PAGE_SCAN_REPETITION_MODE_R2.

HCI_PARAM_LEN_BD_ADDR Macro

File

hci.h

C

```
#define HCI_PARAM_LEN_BD_ADDR 6
```

Description

This is macro HCI_PARAM_LEN_BD_ADDR.

HCI_PARAM_LEN_BYTE Macro

File

hci.h

C

```
#define HCI_PARAM_LEN_BYTE 1
```

Description

This is macro HCI_PARAM_LEN_BYTE.

HCI_PARAM_LEN_DEV_CLASS Macro

File

hci.h

C

```
#define HCI_PARAM_LEN_DEV_CLASS 3
```

Description

This is macro HCI_PARAM_LEN_DEV_CLASS.

HCI_PARAM_LEN_HANDLE Macro

File

hci.h

C

```
#define HCI_PARAM_LEN_HANDLE 4
```

Description

This is macro HCI_PARAM_LEN_HANDLE.

HCI_PARAM_LEN_INT Macro

File

hci.h

C

```
#define HCI_PARAM_LEN_INT 2
```

Description

This is macro HCI_PARAM_LEN_INT.

HCI_PARAM_LEN_LONG Macro

File

hci.h

C

```
#define HCI_PARAM_LEN_LONG 4
```

Description

This is macro HCI_PARAM_LEN_LONG.

HCI_PARAM_TYPE_BD_ADDR Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_BD_ADDR 4
```

Description

This is macro HCI_PARAM_TYPE_BD_ADDR.

HCI_PARAM_TYPE_BYTE Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_BYTE 0
```

Description

This is macro HCI_PARAM_TYPE_BYTE.

HCI_PARAM_TYPE_DEV_CLASS Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_DEV_CLASS 6
```

Description

This is macro HCI_PARAM_TYPE_DEV_CLASS.

HCI_PARAM_TYPE_HANDLE Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_HANDLE 5
```

Description

This is macro HCI_PARAM_TYPE_HANDLE.

HCI_PARAM_TYPE_INT Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_INT 1
```

Description

This is macro HCI_PARAM_TYPE_INT.

HCI_PARAM_TYPE_LONG Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_LONG 2
```

Description

This is macro HCI_PARAM_TYPE_LONG.

HCI_PARAM_TYPE_STRING Macro

File

hci.h

C

```
#define HCI_PARAM_TYPE_STRING 7
```

Description

This is macro HCI_PARAM_TYPE_STRING.

HCI_PARK_STATE Macro

File

hci.h

C

```
#define HCI_PARK_STATE HCI_OPCODE(OGF_LINK_POLICY, 0x0005)
```

Description

This is macro HCI_PARK_STATE.

HCI_PERIODIC_INQUIRY_MODE Macro

File

hci.h

C

```
#define HCI_PERIODIC_INQUIRY_MODE HCI_OPCODE(OGF_LINK_CONTROL, 0x0003)
```

Description

This is macro HCI_PERIODIC_INQUIRY_MODE.

HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x000E)
```

Description

This is macro HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY.

HCI_PIN_CODE_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_PIN_CODE_REQUEST_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x000D)
```

Description

This is macro HCI_PIN_CODE_REQUEST_REPLY.

HCI_POWER_MODE_ACTIVE Macro

File

hci.h

C

```
#define HCI_POWER_MODE_ACTIVE 0
```

Description

This is macro HCI_POWER_MODE_ACTIVE.

HCI_POWER_MODE_HOLD Macro

File

hci.h

C

```
#define HCI_POWER_MODE_HOLD 1
```

Description

This is macro HCI_POWER_MODE_HOLD.

HCI_POWER_MODE_PARK Macro

File

hci.h

C

```
#define HCI_POWER_MODE_PARK 3
```

Description

This is macro HCI_POWER_MODE_PARK.

HCI_POWER_MODE_SNIFF Macro

File

hci.h

C

```
#define HCI_POWER_MODE_SNIFF 2
```

Description

This is macro HCI_POWER_MODE_SNIFF.

HCI_QOS_SETUP Macro

File

hci.h

C

```
#define HCI_QOS_SETUP HCI_OPCODE(OGF_LINK_POLICY, 0x0007)
```

Description

This is macro HCI_QOS_SETUP.

HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE Macro

File

hci.h

C

```
#define HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0048)
```

Description

This is macro HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE.

HCI_READ_AFH_CHANNEL_MAP Macro

File

hci.h

C

```
#define HCI_READ_AFH_CHANNEL_MAP HCI_OPCODE(OGF_STATUS, 0x0006)
```

Description

This is macro HCI_READ_AFH_CHANNEL_MAP.

HCI_READ_AUTHENTICATION_ENABLE Macro

File

hci.h

C

```
#define HCI_READ_AUTHENTICATION_ENABLE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001F)
```

Description

This is macro HCI_READ_AUTHENTICATION_ENABLE.

HCI_READ_AUTOMATIC_FLASH_TIMEOUT Macro

File

hci.h

C

```
#define HCI_READ_AUTOMATIC_FLASH_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0027)
```

Description

This is macro HCI_READ_AUTOMATIC_FLASH_TIMEOUT.

HCI_READ_BD_ADDR Macro

File

hci.h

C

```
#define HCI_READ_BD_ADDR HCI_OPCODE(OGF_INFORMATION, 0x0009)
```

Description

This is macro HCI_READ_BD_ADDR.

HCI_READ_BUFFER_SIZE Macro

File

hci.h

C

```
#define HCI_READ_BUFFER_SIZE HCI_OPCODE(OGF_INFORMATION, 0x0005)
```

Description

This is macro HCI_READ_BUFFER_SIZE.

HCI_READ_CLASS_OF_DEVICE Macro

File

hci.h

C

```
#define HCI_READ_CLASS_OF_DEVICE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0023)
```

Description

This is macro HCI_READ_CLASS_OF_DEVICE.

HCI_READ_CLOCK_COMMAND Macro

File

hci.h

C

```
#define HCI_READ_CLOCK_COMMAND HCI_OPCODE(OGF_STATUS, 0x0007)
```

Description

This is macro HCI_READ_CLOCK_COMMAND.

HCI_READ_CLOCK_OFFSET Macro

File

hci.h

C

```
#define HCI_READ_CLOCK_OFFSET HCI_OPCODE(OGF_LINK_CONTROL, 0x001F)
```

Description

This is macro HCI_READ_CLOCK_OFFSET.

HCI_READ_CONNECTION_ACCEPT_TIMEOUT Macro

File

hci.h

C

```
#define HCI_READ_CONNECTION_ACCEPT_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0015)
```

Description

This is macro HCI_READ_CONNECTION_ACCEPT_TIMEOUT.

HCI_READ_CURRENT_IAC_LAP Macro

File

hci.h

C

```
#define HCI_READ_CURRENT_IAC_LAP HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0039)
```

Description

This is macro HCI_READ_CURRENT_IAC_LAP.

HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING Macro

File

hci.h

C

```
#define HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING HCI_OPCODE(OGF_CTRL_BASEBAND, 0x005A)
```

Description

This is macro HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING.

HCI_READ_DEFAULT_POLICY_SETTINGS Macro

File

hci.h

C

```
#define HCI_READ_DEFAULT_POLICY_SETTINGS HCI_OPCODE(OGF_LINK_POLICY, 0x000E)
```

Description

This is macro HCI_READ_DEFAULT_POLICY_SETTINGS.

HCI_READ_ENCRYPTION_MODE Macro

File

hci.h

C

```
#define HCI_READ_ENCRYPTION_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0021)
```

Description

This is macro HCI_READ_ENCRYPTION_MODE.

HCI_READ_EXTENDED_INQUIRY_RESPONSE Macro

File

hci.h

C

```
#define HCI_READ_EXTENDED_INQUIRY_RESPONSE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0051)
```

Description

This is macro HCI_READ_EXTENDED_INQUIRY_RESPONSE.

HCI_READ_FAILED_CONTACT_COUNTER Macro

File

hci.h

C

```
#define HCI_READ_FAILED_CONTACT_COUNTER HCI_OPCODE(OGF_STATUS, 0x0001)
```

Description

addtogroup hci @{

@name Status Parameters

details The Controller modifies all status parameters. These parameters provide information about the current state of the Link Manager and Baseband in the BR/EDR Controller and the PAL in an AMP Controller. The host device cannot modify any of these parameters other than to reset certain specific parameters.

HCI_READ_HOLD_MODE_ACTIVITY Macro

File

hci.h

C

```
#define HCI_READ_HOLD_MODE_ACTIVITY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002B)
```

Description

This is macro HCI_READ_HOLD_MODE_ACTIVITY.

HCI_READ_INQUIRY_MODE Macro

File

hci.h

C

```
#define HCI_READ_INQUIRY_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0044)
```

Description

This is macro HCI_READ_INQUIRY_MODE.

HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL Macro

File

hci.h

C

```
#define HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0058)
```

Description

This is macro HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL.

HCI_READ_INQUIRY_SCAN_ACTIVITY Macro

File

hci.h

C

```
#define HCI_READ_INQUIRY_SCAN_ACTIVITY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001D)
```

Description

This is macro HCI_READ_INQUIRY_SCAN_ACTIVITY.

HCI_READ_INQUIRY_SCAN_TYPE Macro

File

hci.h

C

```
#define HCI_READ_INQUIRY_SCAN_TYPE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0042)
```

Description

This is macro HCI_READ_INQUIRY_SCAN_TYPE.

HCI_READ_LE_HOST_SUPPORT Macro

File

hci.h

C

```
#define HCI_READ_LE_HOST_SUPPORT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x006C)
```

Description

This is macro HCI_READ_LE_HOST_SUPPORT.

HCI_READ_LINK_POLICY_SETTINGS Macro

File

hci.h

C

```
#define HCI_READ_LINK_POLICY_SETTINGS HCI_OPCODE(OGF_LINK_POLICY, 0x000C)
```

Description

This is macro HCI_READ_LINK_POLICY_SETTINGS.

HCI_READ_LINK_QUALITY Macro

File

hci.h

C

```
#define HCI_READ_LINK_QUALITY HCI_OPCODE(OGF_STATUS, 0x0003)
```

Description

This is macro HCI_READ_LINK_QUALITY.

HCI_READ_LINK_SUPERVISION_TIMEOUT Macro

File

hci.h

C

```
#define HCI_READ_LINK_SUPERVISION_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0036)
```

Description

This is macro HCI_READ_LINK_SUPERVISION_TIMEOUT.

HCI_READ_LMP_HANDLE Macro

File

hci.h

C

```
#define HCI_READ_LMP_HANDLE HCI_OPCODE(OGF_LINK_CONTROL, 0x0020)
```

Description

This is macro HCI_READ_LMP_HANDLE.

HCI_READ_LOCAL_EXTENDED_FEATURES Macro

File

hci.h

C

```
#define HCI_READ_LOCAL_EXTENDED_FEATURES HCI_OPCODE(OGF_INFORMATION, 0x0004)
```

Description

This is macro HCI_READ_LOCAL_EXTENDED_FEATURES.

HCI_READ_LOCAL_NAME Macro

File

hci.h

C

```
#define HCI_READ_LOCAL_NAME HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0014)
```

Description

This is macro HCI_READ_LOCAL_NAME.

HCI_READ_LOCAL_OOB_DATA Macro

File

hci.h

C

```
#define HCI_READ_LOCAL_OOB_DATA HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0057)
```

Description

This is macro HCI_READ_LOCAL_OOB_DATA.

HCI_READ_LOCAL_SUPPORTED_COMMANDS Macro

File

hci.h

C

```
#define HCI_READ_LOCAL_SUPPORTED_COMMANDS HCI_OPCODE(OGF_INFORMATION, 0x0002)
```

Description

This is macro HCI_READ_LOCAL_SUPPORTED_COMMANDS.

HCI_READ_LOCAL_SUPPORTED_FEATURES Macro

File

hci.h

C

```
#define HCI_READ_LOCAL_SUPPORTED_FEATURES HCI_OPCODE(OGF_INFORMATION, 0x0003)
```

Description

This is macro HCI_READ_LOCAL_SUPPORTED_FEATURES.

HCI_READ_LOCAL_VERSION_INFORMATION Macro

File

hci.h

C

```
#define HCI_READ_LOCAL_VERSION_INFORMATION HCI_OPCODE(OGF_INFORMATION, 0x0001)
```

Description

addtogroup hci @{

@name Informational Parameters

details The Informational Parameters are fixed by the manufacturer of the Bluetooth hardware. These parameters provide information about the BR/EDR Controller and the capabilities of the Link Manager and Baseband in the BR/EDR Controller and PAL in the AMP Controller. The host device cannot modify any of these parameters.

HCI_READ_LOOPBACK_MODE Macro

File

hci.h

C

```
#define HCI_READ_LOOPBACK_MODE HCI_OPCODE(OGF_TESTING, 0x0001)
```

Description

addtogroup hci @{

@name Testing Commands

details The Testing commands are used to provide the ability to test various functional capabilities of the Bluetooth hardware.

HCI_READ_NUM_BROADCST_RETR Macro

File

hci.h

C

```
#define HCI_READ_NUM_BROADCST_RETR HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0029)
```

Description

This is macro HCI_READ_NUM_BROADCST_RETR.

HCI_READ_NUM_OF_SUPPORTED_IAC Macro

File

hci.h

C

```
#define HCI_READ_NUM_OF_SUPPORTED_IAC HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0038)
```

Description

This is macro HCI_READ_NUM_OF_SUPPORTED_IAC.

HCI_READ_PAGE_SCAN_ACTIVITY Macro

File

hci.h

C

```
#define HCI_READ_PAGE_SCAN_ACTIVITY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001B)
```

Description

This is macro HCI_READ_PAGE_SCAN_ACTIVITY.

HCI_READ_PAGE_SCAN_PERIOD_MODE Macro

File

hci.h

C

```
#define HCI_READ_PAGE_SCAN_PERIOD_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x003B)
```

Description

This is macro HCI_READ_PAGE_SCAN_PERIOD_MODE.

HCI_READ_PAGE_SCAN_TYPE Macro

File

hci.h

C

```
#define HCI_READ_PAGE_SCAN_TYPE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0046)
```

Description

This is macro HCI_READ_PAGE_SCAN_TYPE.

HCI_READ_PAGE_TIMEOUT Macro

File

hci.h

C

```
#define HCI_READ_PAGE_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0017)
```

Description

This is macro HCI_READ_PAGE_TIMEOUT.

HCI_READ_PIN_TYPE Macro

File

hci.h

C

```
#define HCI_READ_PIN_TYPE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0009)
```

Description

This is macro HCI_READ_PIN_TYPE.

HCI_READ_REFRESH_ENCRYPTION_KEY Macro

File

hci.h

C

```
#define HCI_READ_REFRESH_ENCRYPTION_KEY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0053)
```

Description

This is macro HCI_READ_REFRESH_ENCRYPTION_KEY.

HCI_READ_REMOTE_EXTENDED_FEATURES Macro

File

hci.h

C

```
#define HCI_READ_REMOTE_EXTENDED_FEATURES HCI_OPCODE(OGF_LINK_CONTROL, 0x001C)
```

Description

This is macro HCI_READ_REMOTE_EXTENDED_FEATURES.

HCI_READ_REMOTE_SUPPORTED_FEATURES Macro

File

hci.h

C

```
#define HCI_READ_REMOTE_SUPPORTED_FEATURES HCI_OPCODE(OGF_LINK_CONTROL, 0x001B)
```

Description

This is macro HCI_READ_REMOTE_SUPPORTED_FEATURES.

HCI_READ_REMOTE_VERSION_INFORMATION Macro

File

hci.h

C

```
#define HCI_READ_REMOTE_VERSION_INFORMATION HCI_OPCODE(OGF_LINK_CONTROL, 0x001D)
```

Description

This is macro HCI_READ_REMOTE_VERSION_INFORMATION.

HCI_READ_RSSI Macro

File

hci.h

C

```
#define HCI_READ_RSSI HCI_OPCODE(OGF_STATUS, 0x0005)
```

Description

This is macro HCI_READ_RSSI.

HCI_READ_SCAN_ENABLE Macro

File

hci.h

C

```
#define HCI_READ_SCAN_ENABLE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0019)
```

Description

This is macro HCI_READ_SCAN_ENABLE.

HCI_READ_SIMPLE_PAIRING_MODE Macro

File

hci.h

C

```
#define HCI_READ_SIMPLE_PAIRING_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0055)
```

Description

This is macro HCI_READ_SIMPLE_PAIRING_MODE.

HCI_READ_STORED_LINK_KEY Macro

File

hci.h

C

```
#define HCI_READ_STORED_LINK_KEY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x000D)
```

Description

This is macro HCI_READ_STORED_LINK_KEY.

HCI_READ_SYNC_FLOW_CONTROL_ENABLE Macro

File

hci.h

C

```
#define HCI_READ_SYNC_FLOW_CONTROL_ENABLE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002E)
```

Description

This is macro HCI_READ_SYNC_FLOW_CONTROL_ENABLE.

HCI_READ_TRANSMIT_POWER_LEVEL Macro

File

hci.h

C

```
#define HCI_READ_TRANSMIT_POWER_LEVEL HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002D)
```

Description

This is macro HCI_READ_TRANSMIT_POWER_LEVEL.

HCI_READ_VOICE_SETTING Macro

File

hci.h

C

```
#define HCI_READ_VOICE_SETTING HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0025)
```

Description

This is macro HCI_READ_VOICE_SETTING.

HCI_REJECT_CONNECTION_REQUEST Macro

File

hci.h

C

```
#define HCI_REJECT_CONNECTION_REQUEST HCI_OPCODE(OGF_LINK_CONTROL, 0x000A)
```

Description

This is macro HCI_REJECT_CONNECTION_REQUEST.

HCI_REJECT_SYNCH_CONNECTION_REQUEST Macro

File

hci.h

C

```
#define HCI_REJECT_SYNCH_CONNECTION_REQUEST HCI_OPCODE(OGF_LINK_CONTROL, 0x002A)
```

Description

This is macro HCI_REJECT_SYNCH_CONNECTION_REQUEST.

HCI_REMOTE_NAME_REQUEST Macro

File

hci.h

C

```
#define HCI_REMOTE_NAME_REQUEST HCI_OPCODE(OGF_LINK_CONTROL, 0x0019)
```

Description

This is macro HCI_REMOTE_NAME_REQUEST.

HCI_REMOTE_NAME_REQUEST_CANCEL Macro

File

hci.h

C

```
#define HCI_REMOTE_NAME_REQUEST_CANCEL HCI_OPCODE(OGF_LINK_CONTROL, 0x001A)
```

Description

This is macro HCI_REMOTE_NAME_REQUEST_CANCEL.

HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x0033)
```

Description

This is macro HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY.

HCI_REMOTE_OOB_DATA_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_REMOTE_OOB_DATA_REQUEST_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x0030)
```

Description

This is macro HCI_REMOTE_OOB_DATA_REQUEST_REPLY.

HCI_RESET Macro

File

hci.h

C

```
#define HCI_RESET HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0003)
```

Description

This is macro HCI_RESET.

HCI_RESET_FAILED_CONTACT_COUNTER Macro

File

hci.h

C

```
#define HCI_RESET_FAILED_CONTACT_COUNTER HCI_OPCODE(OGF_STATUS, 0x0002)
```

Description

This is macro HCI_RESET_FAILED_CONTACT_COUNTER.

HCI_ROLE_DISCOVERY Macro

File

hci.h

C

```
#define HCI_ROLE_DISCOVERY HCI_OPCODE(OGF_LINK_POLICY, 0x0009)
```

Description

This is macro HCI_ROLE_DISCOVERY.

HCI_ROLE_SWITCH_ALLOW Macro

File

hci.h

C

```
#define HCI_ROLE_SWITCH_ALLOW 0x01
```

Description

This is macro HCI_ROLE_SWITCH_ALLOW.

HCI_ROLE_SWITCH_DISALLOW Macro

File

hci.h

C

```
#define HCI_ROLE_SWITCH_DISALLOW 0x00
```

Description

This is macro HCI_ROLE_SWITCH_DISALLOW.

HCI_SCAN_INQUIRY Macro

File

hci.h

C

```
#define HCI_SCAN_INQUIRY 0x01
```

Description

Flags for HCI_READ_SCAN_ENABLE/HCI_WRITE_SCAN_ENABLE

HCI_SCAN_PAGE Macro

File

hci.h

C

```
#define HCI_SCAN_PAGE 0x02
```

Description

This is macro HCI_SCAN_PAGE.

HCI_SCO_CONTENT_FORMAT_AIR_CODING_A_LAW Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_AIR_CODING_A_LAW 0x2
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_A_LAW.

HCI_SCO_CONTENT_FORMAT_AIR_CODING_CSVD Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_AIR_CODING_CSVD 0x0
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_CSVD.

HCI_SCO_CONTENT_FORMAT_AIR_CODING_N_LAW Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_AIR_CODING_N_LAW 0x1
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_N_LAW.

HCI_SCO_CONTENT_FORMAT_AIR_CODING_TRANSPARENT Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_AIR_CODING_TRANSPARENT 0x3
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_TRANSPARENT.

HCI_SCO_CONTENT_FORMAT_INPUT_CODING_A_LAW Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_CODING_A_LAW 0x200
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_A_LAW.

HCI_SCO_CONTENT_FORMAT_INPUT_CODING_LINEAR Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_CODING_LINEAR 0
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_LINEAR.

HCI_SCO_CONTENT_FORMAT_INPUT_CODING_N_LAW Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_CODING_N_LAW 0x100
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_N_LAW.

HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_1_COMPLEMENT Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_1_COMPLEMENT 0
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_1_COMPLEMENT.

HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_2_COMPLEMENT Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_2_COMPLEMENT 0x040
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_2_COMPLEMENT.

HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_SIGN_MAGNITUDE Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_SIGN_MAGNITUDE 0x080
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_SIGN_MAGNITUDE.

HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_UNSIGNED Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_UNSIGNED 0x0C0
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_UNSIGNED.

HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_16 Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_16 0x020
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_16.

HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_8 Macro

File

hci.h

C

```
#define HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_8 0x0
```

Description

This is macro HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_8.

HCI_SCO_DATA_HEADER_LEN Macro

File

hci.h

C

```
#define HCI_SCO_DATA_HEADER_LEN 3
```

Description

This is macro HCI_SCO_DATA_HEADER_LEN.

HCI_SCO_MAX_DATA_LEN Macro

File

hci.h

C

```
#define HCI_SCO_MAX_DATA_LEN 255
```

Description

This is macro HCI_SCO_MAX_DATA_LEN.

HCI_SCO_MAX_LATENCY_DONTCARE Macro

File

hci.h

C

```
#define HCI_SCO_MAX_LATENCY_DONTCARE 0xFFFF
```

Description

This is macro HCI_SCO_MAX_LATENCY_DONTCARE.

HCI_SCO_PACKET_TYPE_ALL Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_ALL 0x003F
```

Description

This is macro HCI_SCO_PACKET_TYPE_ALL.

HCI_SCO_PACKET_TYPE_EV3 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_EV3 0x0008
```

Description

This is macro HCI_SCO_PACKET_TYPE_EV3.

HCI_SCO_PACKET_TYPE_EV4 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_EV4 0x0010
```

Description

This is macro HCI_SCO_PACKET_TYPE_EV4.

HCI_SCO_PACKET_TYPE_EV5 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_EV5 0x0020
```

Description

This is macro HCI_SCO_PACKET_TYPE_EV5.

HCI_SCO_PACKET_TYPE_HV1 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_HV1 0x0001
```

Description

This is macro HCI_SCO_PACKET_TYPE_HV1.

HCI_SCO_PACKET_TYPE_HV2 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_HV2 0x0002
```

Description

This is macro HCI_SCO_PACKET_TYPE_HV2.

HCI_SCO_PACKET_TYPE_HV3 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_HV3 0x0004
```

Description

This is macro HCI_SCO_PACKET_TYPE_HV3.

HCI_SCO_PACKET_TYPE_NO_2_EV3 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_NO_2_EV3 0x0040
```

Description

This is macro HCI_SCO_PACKET_TYPE_NO_2_EV3.

HCI_SCO_PACKET_TYPE_NO_2_EV5 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_NO_2_EV5 0x0100
```

Description

This is macro HCI_SCO_PACKET_TYPE_NO_2_EV5.

HCI_SCO_PACKET_TYPE_NO_3_EV3 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_NO_3_EV3 0x0080
```

Description

This is macro HCI_SCO_PACKET_TYPE_NO_3_EV3.

HCI_SCO_PACKET_TYPE_NO_3_EV5 Macro

File

hci.h

C

```
#define HCI_SCO_PACKET_TYPE_NO_3_EV5 0x0200
```

Description

This is macro HCI_SCO_PACKET_TYPE_NO_3_EV5.

HCI_SCO_RTX_EFFORT_DONTCARE Macro

File

hci.h

C

```
#define HCI_SCO_RTX_EFFORT_DONTCARE 0xFF
```

Description

This is macro HCI_SCO_RTX_EFFORT_DONTCARE.

HCI_SCO_RTX_EFFORT_NO_RETRANSMISSION Macro

File

hci.h

C

```
#define HCI_SCO_RTX_EFFORT_NO_RETRANSMISSION 0x0
```

Description

This is macro HCI_SCO_RTX_EFFORT_NO_RETRANSMISSION.

HCI_SCO_RTX_EFFORT_OPTIMIZE_LINK_QUALITY Macro

File

hci.h

C

```
#define HCI_SCO_RTX_EFFORT_OPTIMIZE_LINK_QUALITY 0x2
```

Description

This is macro HCI_SCO_RTX_EFFORT_OPTIMIZE_LINK_QUALITY.

HCI_SCO_RTX_EFFORT_OPTIMIZE_POWER_CONSUMPTION Macro

File

hci.h

C

```
#define HCI_SCO_RTX_EFFORT_OPTIMIZE_POWER_CONSUMPTION 0x1
```

Description

This is macro HCI_SCO_RTX_EFFORT_OPTIMIZE_POWER_CONSUMPTION.

HCI_SCO_RX_BANDWIDTH_DONTCARE Macro

File

hci.h

C

```
#define HCI_SCO_RX_BANDWIDTH_DONTCARE 0xFFFFFFFF
```

Description

This is macro HCI_SCO_RX_BANDWIDTH_DONTCARE.

HCI_SCO_TX_BANDWIDTH_DONTCARE Macro

File

hci.h

C

```
#define HCI_SCO_TX_BANDWIDTH_DONTCARE 0xFFFFFFFF
```

Description

This is macro HCI_SCO_TX_BANDWIDTH_DONTCARE.

HCI_SEND_DATA_STATUS_INTERRUPTED Macro

File

hci.h

C

```
#define HCI_SEND_DATA_STATUS_INTERRUPTED 1
```

Description

This is macro HCI_SEND_DATA_STATUS_INTERRUPTED.

HCI_SEND_DATA_STATUS_SUCCESS Macro

File

hci.h

C

```
#define HCI_SEND_DATA_STATUS_SUCCESS 0
```

Description

This is macro HCI_SEND_DATA_STATUS_SUCCESS.

HCI_SEND_KEY_PRESS_NOTIFICATION Macro

File

hci.h

C

```
#define HCI_SEND_KEY_PRESS_NOTIFICATION HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0060)
```

Description

This is macro HCI_SEND_KEY_PRESS_NOTIFICATION.

HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION Macro

File

hci.h

C

```
#define HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION HCI_OPCODE(OGF_CTRL_BASEBAND, 0x003F)
```

Description

This is macro HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION.

HCI_SET_CONNECTION_ENCRYPTION Macro

File

hci.h

C

```
#define HCI_SET_CONNECTION_ENCRYPTION HCI_OPCODE(OGF_LINK_CONTROL, 0x0013)
```

Description

This is macro HCI_SET_CONNECTION_ENCRYPTION.

HCI_SET_CTRL_TO_HOST_FLOW_CONTROL Macro

File

hci.h

C

```
#define HCI_SET_CTRL_TO_HOST_FLOW_CONTROL HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0031)
```

Description

This is macro HCI_SET_CTRL_TO_HOST_FLOW_CONTROL.

HCI_SET_EVENT_FILTER Macro

File

hci.h

C

```
#define HCI_SET_EVENT_FILTER HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0005)
```

Description

This is macro HCI_SET_EVENT_FILTER.

HCI_SET_EVENT_MASK Macro

File

hci.h

C

```
#define HCI_SET_EVENT_MASK HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0001)
```

Description

addtogroup hci @{

@name Controller & Baseband commands

details The Controller & Baseband Commands provide access and control to various capabilities of the Bluetooth hardware.

HCI_SETUP_SYNCHRONOUS_CONNECTION Macro

File

hci.h

C

```
#define HCI_SETUP_SYNCHRONOUS_CONNECTION HCI_OPCODE(OGF_LINK_CONTROL, 0x0028)
```

Description

This is macro HCI_SETUP_SYNCHRONOUS_CONNECTION.

HCI_SNIFF_MODE Macro

File

hci.h

C

```
#define HCI_SNIFF_MODE HCI_OPCODE(OGF_LINK_POLICY, 0x0003)
```

Description

This is macro HCI_SNIFF_MODE.

HCI_SNIFF_SUBRATING Macro

File

hci.h

C

```
#define HCI_SNIFF_SUBRATING HCI_OPCODE(OGF_LINK_POLICY, 0x0011)
```

Description

This is macro HCI_SNIFF_SUBRATING.

HCI_SWITCH_ROLE Macro

File

hci.h

C

```
#define HCI_SWITCH_ROLE HCI_OPCODE(OGF_LINK_POLICY, 0x000B)
```

Description

This is macro HCI_SWITCH_ROLE.

HCI_TRANSPORT_HEADER_LEN Macro

File

hci_transport.h

C

```
#define HCI_TRANSPORT_HEADER_LEN 1
```

Description

This is macro HCI_TRANSPORT_HEADER_LEN.

HCI_UART_PACKET_TYPE_ACL_DATA Macro

File

hcitr_uart.h

C

```
#define HCI_UART_PACKET_TYPE_ACL_DATA 2
```

Description

This is macro HCI_UART_PACKET_TYPE_ACL_DATA.

HCI_UART_PACKET_TYPE_COMMAND Macro

File

hcitr_uart.h

C

```
#define HCI_UART_PACKET_TYPE_COMMAND 1
```

Description

defgroup hcitr_uart HCI UART (H4) transport protocol ingroup hcitr

details This module describes functions used to initialize and start HCI UART transport protocol. The transport uses common interface for exchanging data between the host CPU and HCI controller defined in bt_hcitr.h. This interface consist of two functions that must be implemented by the application: li bt_oem_send() li bt_oem_recv()

HCI_UART_PACKET_TYPE_EVENT Macro

File

hcitr_uart.h

C

```
#define HCI_UART_PACKET_TYPE_EVENT 4
```

Description

This is macro HCI_UART_PACKET_TYPE_EVENT.

HCI_UART_PACKET_TYPE_SCO_DATA Macro

File

hcitr_uart.h

C

```
#define HCI_UART_PACKET_TYPE_SCO_DATA 3
```

Description

This is macro HCI_UART_PACKET_TYPE_SCO_DATA.

HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x002D)
```

Description

This is macro HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY.

HCI_USER_CONFIRMATION_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_USER_CONFIRMATION_REQUEST_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x002C)
```

Description

This is macro HCI_USER_CONFIRMATION_REQUEST_REPLY.

HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY Macro

File

hci.h

C

```
#define HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x002F)
```

Description

This is macro HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY.

HCI_USER_PASSKEY_REQUEST_REPLY Macro

File

hci.h

C

```
#define HCI_USER_PASSKEY_REQUEST_REPLY HCI_OPCODE(OGF_LINK_CONTROL, 0x002E)
```

Description

This is macro HCI_USER_PASSKEY_REQUEST_REPLY.

HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0049)
```

Description

This is macro HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE.

HCI_WRITE_AUTHENTICATION_ENABLE Macro

File

hci.h

C

```
#define HCI_WRITE_AUTHENTICATION_ENABLE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0020)
```

Description

This is macro HCI_WRITE_AUTHENTICATION_ENABLE.

HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT Macro

File

hci.h

C

```
#define HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0028)
```

Description

This is macro HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT.

HCI_WRITE_CLASS_OF_DEVICE Macro

File

hci.h

C

```
#define HCI_WRITE_CLASS_OF_DEVICE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0024)
```

Description

This is macro HCI_WRITE_CLASS_OF_DEVICE.

HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT Macro

File

hci.h

C

```
#define HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0016)
```

Description

This is macro HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT.

HCI_WRITE_CURRENT_IAC_LAP Macro

File

hci.h

C

```
#define HCI_WRITE_CURRENT_IAC_LAP HCI_OPCODE(OGF_CTRL_BASEBAND, 0x003A)
```

Description

This is macro HCI_WRITE_CURRENT_IAC_LAP.

HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING Macro

File

hci.h

C

```
#define HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING HCI_OPCODE(OGF_CTRL_BASEBAND, 0x005B)
```

Description

This is macro HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING.

HCI_WRITE_DEFAULT_POLICY_SETTINGS Macro

File

hci.h

C

```
#define HCI_WRITE_DEFAULT_POLICY_SETTINGS HCI_OPCODE(OGF_LINK_POLICY, 0x000F)
```

Description

This is macro HCI_WRITE_DEFAULT_POLICY_SETTINGS.

HCI_WRITE_ENCRYPTION_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_ENCRYPTION_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0022)
```

Description

This is macro HCI_WRITE_ENCRYPTION_MODE.

HCI_WRITE_EXTENDED_INQUIRY_RESPONSE Macro

File

hci.h

C

```
#define HCI_WRITE_EXTENDED_INQUIRY_RESPONSE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0052)
```

Description

This is macro HCI_WRITE_EXTENDED_INQUIRY_RESPONSE.

HCI_WRITE_EXTENDED_INQUIRY_RESPONSE_PARAM_LEN Macro

File

hci.h

C

```
#define HCI_WRITE_EXTENDED_INQUIRY_RESPONSE_PARAM_LEN 241
```

Description

This is macro HCI_WRITE_EXTENDED_INQUIRY_RESPONSE_PARAM_LEN.

HCI_WRITE_HOLD_MODE_ACTIVITY Macro

File

hci.h

C

```
#define HCI_WRITE_HOLD_MODE_ACTIVITY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002C)
```

Description

This is macro HCI_WRITE_HOLD_MODE_ACTIVITY.

HCI_WRITE_INQUIRY_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_INQUIRY_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0045)
```

Description

This is macro HCI_WRITE_INQUIRY_MODE.

HCI_WRITE_INQUIRY_SCAN_ACTIVITY Macro

File

hci.h

C

```
#define HCI_WRITE_INQUIRY_SCAN_ACTIVITY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001E)
```

Description

This is macro HCI_WRITE_INQUIRY_SCAN_ACTIVITY.

HCI_WRITE_INQUIRY_SCAN_TYPE Macro

File

hci.h

C

```
#define HCI_WRITE_INQUIRY_SCAN_TYPE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0043)
```

Description

This is macro HCI_WRITE_INQUIRY_SCAN_TYPE.

HCI_WRITE_INQUIRY_TX_POWER_LEVEL Macro

File

hci.h

C

```
#define HCI_WRITE_INQUIRY_TX_POWER_LEVEL HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0059)
```

Description

This is macro HCI_WRITE_INQUIRY_TX_POWER_LEVEL.

HCI_WRITE_LE_HOST_SUPPORT Macro

File

hci.h

C

```
#define HCI_WRITE_LE_HOST_SUPPORT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x006D)
```

Description

This is macro HCI_WRITE_LE_HOST_SUPPORT.

HCI_WRITE_LINK_POLICY_SETTINGS Macro

File

hci.h

C

```
#define HCI_WRITE_LINK_POLICY_SETTINGS HCI_OPCODE(OGF_LINK_POLICY, 0x000D)
```

Description

This is macro HCI_WRITE_LINK_POLICY_SETTINGS.

HCI_WRITE_LINK_SUPERVISION_TIMEOUT Macro

File

hci.h

C

```
#define HCI_WRITE_LINK_SUPERVISION_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0037)
```

Description

This is macro HCI_WRITE_LINK_SUPERVISION_TIMEOUT.

HCI_WRITE_LOCAL_NAME Macro

File

hci.h

C

```
#define HCI_WRITE_LOCAL_NAME HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0013)
```

Description

This is macro HCI_WRITE_LOCAL_NAME.

HCI_WRITE_LOCAL_NAME_PARAM_LEN Macro

File

hci.h

C

```
#define HCI_WRITE_LOCAL_NAME_PARAM_LEN 248
```

Description

This is macro HCI_WRITE_LOCAL_NAME_PARAM_LEN.

HCI_WRITE_LOOPBACK_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_LOOPBACK_MODE HCI_OPCODE(OGF_TESTING, 0x0002)
```

Description

This is macro HCI_WRITE_LOOPBACK_MODE.

HCI_WRITE_NUM_BROADCST_RETR Macro

File

hci.h

C

```
#define HCI_WRITE_NUM_BROADCST_RETR HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002A)
```

Description

This is macro HCI_WRITE_NUM_BROADCST_RETR.

HCI_WRITE_PAGE_SCAN_ACTIVITY Macro

File

hci.h

C

```
#define HCI_WRITE_PAGE_SCAN_ACTIVITY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001C)
```

Description

This is macro HCI_WRITE_PAGE_SCAN_ACTIVITY.

HCI_WRITE_PAGE_SCAN_PERIOD_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_PAGE_SCAN_PERIOD_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x003C)
```

Description

This is macro HCI_WRITE_PAGE_SCAN_PERIOD_MODE.

HCI_WRITE_PAGE_SCAN_TYPE Macro

File

hci.h

C

```
#define HCI_WRITE_PAGE_SCAN_TYPE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0047)
```

Description

This is macro HCI_WRITE_PAGE_SCAN_TYPE.

HCI_WRITE_PAGE_TIMEOUT Macro

File

hci.h

C

```
#define HCI_WRITE_PAGE_TIMEOUT HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0018)
```

Description

This is macro HCI_WRITE_PAGE_TIMEOUT.

HCI_WRITE_PIN_TYPE Macro

File

hci.h

C

```
#define HCI_WRITE_PIN_TYPE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x000A)
```

Description

This is macro HCI_WRITE_PIN_TYPE.

HCI_WRITE_SCAN_ENABLE Macro

File

hci.h

C

```
#define HCI_WRITE_SCAN_ENABLE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001A)
```

Description

This is macro HCI_WRITE_SCAN_ENABLE.

HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE HCI_OPCODE(OGF_TESTING, 0x0004)
```

Description

This is macro HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE.

HCI_WRITE_SIMPLE_PAIRING_MODE Macro

File

hci.h

C

```
#define HCI_WRITE_SIMPLE_PAIRING_MODE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0056)
```

Description

This is macro HCI_WRITE_SIMPLE_PAIRING_MODE.

HCI_WRITE_STORED_LINK_KEY Macro

File

hci.h

C

```
#define HCI_WRITE_STORED_LINK_KEY HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0011)
```

Description

This is macro HCI_WRITE_STORED_LINK_KEY.

HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE Macro

File

hci.h

C

```
#define HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002F)
```

Description

This is macro HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE.

HCI_WRITE_VOICE_SETTING Macro

File

hci.h

C

```
#define HCI_WRITE_VOICE_SETTING HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0026)
```

Description

This is macro HCI_WRITE_VOICE_SETTING.

HCITR_3WIRE_DEFAULT_ACK_TIMEOUT Macro

File

hcitr_3wire.h

C

```
#define HCITR_3WIRE_DEFAULT_ACK_TIMEOUT 250
```

Description

This is macro HCITR_3WIRE_DEFAULT_ACK_TIMEOUT.

HCITR_BCSP_DEFAULT_ACK_TIMEOUT Macro

File

hcitr_bcsp.h

C

```
#define HCITR_BCSP_DEFAULT_ACK_TIMEOUT 250
```

Description

This is macro HCITR_BCSP_DEFAULT_ACK_TIMEOUT.

LM_PACKET_TYPE_DH1 Macro

File

lm.h

C

```
#define LM_PACKET_TYPE_DH1 0x0010
```

Description

This is macro LM_PACKET_TYPE_DH1.

LM_PACKET_TYPE_DH3 Macro

File

lm.h

C

```
#define LM_PACKET_TYPE_DH3 0x0800
```

Description

This is macro LM_PACKET_TYPE_DH3.

LM_PACKET_TYPE_DH5 Macro

File

lm.h

C

```
#define LM_PACKET_TYPE_DH5 0x8000
```

Description

This is macro LM_PACKET_TYPE_DH5.

LM_PACKET_TYPE_DM1 Macro

File

lm.h

C

```
#define LM_PACKET_TYPE_DM1 0x0008
```

Description

This is macro LM_PACKET_TYPE_DM1.

LM_PACKET_TYPE_DM3 Macro

File

lm.h

C

```
#define LM_PACKET_TYPE_DM3 0x0400
```

Description

This is macro LM_PACKET_TYPE_DM3.

LM_PACKET_TYPE_DM5 Macro

File

lm.h

C

```
#define LM_PACKET_TYPE_DM5 0x4000
```

Description

This is macro LM_PACKET_TYPE_DM5.

OGF_CTRL_BASEBAND Macro

File

hci.h

C

```
#define OGF_CTRL_BASEBAND 0x03
```

Description

This is macro OGF_CTRL_BASEBAND.

OGF_INFORMATION Macro

File

hci.h

C

```
#define OGF_INFORMATION 0x04
```

Description

This is macro OGF_INFORMATION.

OGF_LE Macro

File

hci.h

C

```
#define OGF_LE 0x08
```

Description

This is macro OGF_LE.

OGF_LINK_CONTROL Macro

File

hci.h

C

```
#define OGF_LINK_CONTROL 0x01
```

Description

This is macro OGF_LINK_CONTROL.

OGF_LINK_POLICY Macro

File

hci.h

C

```
#define OGF_LINK_POLICY 0x02
```

Description

This is macro OGF_LINK_POLICY.

OGF_STATUS Macro

File

hci.h

C

```
#define OGF_STATUS 0x05
```

Description

This is macro OGF_STATUS.

OGF_TESTING Macro

File

hci.h

C

```
#define OGF_TESTING 0x06
```

Description

This is macro OGF_TESTING.

OGF_VENDOR Macro

File

hci.h

C

```
#define OGF_VENDOR 0x3F
```

Description

This is macro OGF_VENDOR.

bt_hci_cmd_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_cmd_callback_fp)(bt_int, struct _bt_hci_command_s*, struct _bt_hci_event_s*);
```

Description

This is type bt_hci_cmd_callback_fp.

bt_hci_cmd_listener_fp Type

File

hci_ctrl_state.h

C

```
typedef void (* bt_hci_cmd_listener_fp)(bt_uint event_id, bt_hci_command_t* cmd, void* cb_param);
```

Description

This is type bt_hci_cmd_listener_fp.

bt_hci_command_p Structure

File

hci.h

C

```
typedef struct _bt_hci_command_s {
  struct _bt_hci_command_s* next_cmd;
  bt_int opcode;
  bt_int params_len;
  bt_byte* params;
  bt_hci_cmd_callback_fp callback;
  void* callback_param;
  bt_int packetNum;
} bt_hci_command_t, * bt_hci_command_p;
```

Description

This is type bt_hci_command_p.

bt_hci_command_t Structure

File

hci.h

C

```
typedef struct _bt_hci_command_s {
  struct _bt_hci_command_s* next_cmd;
  bt_int opcode;
  bt_int params_len;
  bt_byte* params;
  bt_hci_cmd_callback_fp callback;
  void* callback_param;
  bt_int packetNum;
} bt_hci_command_t, * bt_hci_command_p;
```

Description

This is type bt_hci_command_t.

bt_hci_conn_state_p Type

File

hci.h

C

```
typedef struct _bt_hci_conn_state_s * bt_hci_conn_state_p;
```

Description

This is type bt_hci_conn_state_p.

bt_hci_conn_state_t Type

File

hci.h

C

```
typedef struct _bt_hci_conn_state_s bt_hci_conn_state_t;
```

Description

This is type bt_hci_conn_state_t.

bt_hci_connect_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_connect_callback_fp)(bt_byte status, bt_hci_conn_state_t *pconn, void* param);
```

Description

brief HCI connect callback. ingroup hci

details This typedef defines a type for the callback function that is called when HCI connect operation initiated by a call to bt_hci_connect() is complete.

param status Operation status. c It is 0 if connection was successfully established. param pconn pointer to a structure representing the established connection. param param pointer to arbitrary data passed to the bt_hci_connect() function through its c param parameter.

bt_hci_ctrl_listener_t Type

File

hci_ctrl_state.h

C

```
typedef struct _bt_hci_ctrl_listener_t bt_hci_ctrl_listener_t;
```

Description

This is type bt_hci_ctrl_listener_t.

bt_hci_ctrl_state_t Structure

File

hci_ctrl_state.h

C

```
typedef struct _bt_hci_ctrl_state_s {
  bt_byte state;
  bt_byte default_link_policy;
  bt_bdaddr_t bdaddr;
  bt_int num_hci_cmd_packets;
  bt_int total_num_acl_data_packets;
  bt_int total_num_sco_data_packets;
  bt_int aclDataPacketLen;
  bt_byte scoDataPacketLen;
  bt_hci_conn_state_t* connections;
  bt_ulong _discoverable_period;
  bt_ulong _connectable_period;
  bt_byte _last_cmd_status;
  bt_hci_le_ctrl_state_t* le_ctrl_state;
  bt_ulong event_mask_l;
  bt_ulong event_mask_h;
  bt_byte incoming_connection_role;
  bt_byte inquiry_response_tx_power_level;
  struct _bt_l2cap_mgr_s* l2cap_mgr;
  pf_l2cap_receive_callback l2cap_data_receive_callback;
  bt_hci_start_callback_fp _init_cb;
  void* _init_param;
  bt_hci_connect_callback_fp _connect_cb;
  void* _connect_param;
  bt_uint _connect_acl_config;
  bt_hci_connect_callback_fp _listen_cb;
  void* _listen_param;
  bt_ulong sco_tx_bandwidth;
  bt_ulong sco_rcv_bandwidth;
  bt_uint sco_max_latency;
  bt_uint sco_content_format;
  bt_byte sco_retransmission_effort;
```

```
    bt_uint sco_packet_type;
    bt_hci_connect_callback_fp sco_listen_cb;
    void* _sco_listen_param;
    bt_hci_connect_callback_fp sco_connect_cb;
    void* _sco_connect_param;
    bt_hci_inquiry_callback_fp inquiry_cb;
    bt_hci_request_remote_name_callback_fp remote_name_cb;
    bt_byte inquiry_max_responses;
    bt_byte inquiry_length;
    pf_hci_sleep_callback sleep_cb;
    pf_hci_wakeup_callback wakeup_cb;
    bt_hci_ctrl_listener_t* listeners;
  } bt_hci_ctrl_state_t;
```

Description

This is type bt_hci_ctrl_state_t.

bt_hci_data_buffer_p Structure

File

hci_data_buffer.h

C

```
typedef struct _bt_hci_data_buffer_s {
  bt_int state;
  bt_hci_data_t packet;
} bt_hci_data_buffer_t, * bt_hci_data_buffer_p;
```

Description

This is type bt_hci_data_buffer_p.

bt_hci_data_buffer_t Structure

File

hci_data_buffer.h

C

```
typedef struct _bt_hci_data_buffer_s {
  bt_int state;
  bt_hci_data_t packet;
} bt_hci_data_buffer_t, * bt_hci_data_buffer_p;
```

Description

This is type bt_hci_data_buffer_t.

bt_hci_data_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_data_callback_fp)(bt_byte status, struct _bt_hci_data_s*);
```

Description

This is type bt_hci_data_callback_fp.

bt_hci_data_listener_fp Type

File

hci_ctrl_state.h

C

```
typedef void (* bt_hci_data_listener_fp)(bt_hci_conn_state_t* connection, const bt_byte* data, bt_int len,
```

```
bt_bool sent, void* cb_param);
```

Description

This is type bt_hci_data_listener_fp.

bt_hci_data_p Structure

File

hci.h

C

```
typedef struct _bt_hci_data_s {
  struct _bt_hci_data_s* next_data;
  bt_hci_hconn_t hconn;
  bt_uint data_len;
  bt_byte_p pdata;
  bt_uint pos;
  bt_byte link_type;
  bt_hci_data_callback_fp callback;
  void * param;
  bt_int packetNum;
} bt_hci_data_t, * bt_hci_data_p;
```

Description

This is type bt_hci_data_p.

bt_hci_data_t Structure

File

hci.h

C

```
typedef struct _bt_hci_data_s {
  struct _bt_hci_data_s* next_data;
  bt_hci_hconn_t hconn;
  bt_uint data_len;
  bt_byte_p pdata;
  bt_uint pos;
  bt_byte link_type;
  bt_hci_data_callback_fp callback;
  void * param;
  bt_int packetNum;
} bt_hci_data_t, * bt_hci_data_p;
```

Description

This is type bt_hci_data_t.

bt_hci_disconnect_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_disconnect_callback_fp)(bt_byte status, bt_byte reason, bt_hci_conn_state_t *pconn,
void* param);
```

Description

brief HCI disconnect callback. ingroup hci

details This typedef defines a type for the callback function that is called when an HCI connection has been terminated.

param status Operation status. c It is 0 if connection has been successfully terminated. param reason Reason for disconnection. param pconn pointer to a structure representing the connection. param param pointer to arbitrary data associated with an event listener.

bt_hci_event_e Union

File

hci_conn_state.h

C

```
typedef union _bt_hci_event_e {
  bt_hci_evt_disconnection_complete_t disconnect;
} bt_hci_event_e;
```

Description

This is type bt_hci_event_e.

bt_hci_event_handler_ex_fp Type

File

hci_evt_handlers.h

C

```
typedef void (* bt_hci_event_handler_ex_fp)(bt_hci_event_p, bt_uint params_len);
```

Description

This is type bt_hci_event_handler_ex_fp.

bt_hci_event_handler_fp Type

File

hci_evt_handlers.h

C

```
typedef void (* bt_hci_event_handler_fp)(bt_hci_event_p);
```

Description

This is type bt_hci_event_handler_fp.

bt_hci_event_listener_fp Type

File

hci.h

C

```
typedef void (* bt_hci_event_listener_fp)(bt_int evcode, void* evt_params, void* cb_param);
```

Description

This is type bt_hci_event_listener_fp.

bt_hci_event_p Structure

File

hci.h

C

```
typedef struct _bt_hci_event_s {
  bt_int evcode;
  bt_int params_len;
  bt_byte* params;
} bt_hci_event_t, * bt_hci_event_p;
```

Description

This is type bt_hci_event_p.

bt_hci_event_t Structure

File

hci.h

C

```
typedef struct _bt_hci_event_s {
  bt_int evcode;
  bt_int params_len;
  bt_byte* params;
} bt_hci_event_t, * bt_hci_event_p;
```

Description

This is type bt_hci_event_t.

bt_hci_evt_authentication_complete_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_authentication_complete_t {
  bt_byte status;
  bt_hci_conn_state_t* pconn;
} bt_hci_evt_authentication_complete_t;
```

Description

This is type bt_hci_evt_authentication_complete_t.

bt_hci_evt_command_complete_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_command_complete_t {
  bt_byte status;
  bt_int opcode;
  bt_hci_command_t* cmd;
} bt_hci_evt_command_complete_t;
```

Description

This is type bt_hci_evt_command_complete_t.

bt_hci_evt_command_status_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_command_status_t {
  bt_byte status;
  bt_int opcode;
  bt_hci_command_t* cmd;
} bt_hci_evt_command_status_t;
```

Description

This is type bt_hci_evt_command_status_t.

bt_hci_evt_connection_complete_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_connection_complete_t {
  bt_byte status;
  bt_hci_conn_state_t* pconn;
  bt_bool incomming;
} bt_hci_evt_connection_complete_t;
```

Description

This is type bt_hci_evt_connection_complete_t.

bt_hci_evt_connection_request_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_connection_request_t {
  bt_bdaddr_t bdaddr;
  bt_long dev_class;
  bt_byte link_type;
} bt_hci_evt_connection_request_t;
```

Description

This is type bt_hci_evt_connection_request_t.

bt_hci_evt_disconnection_complete_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_disconnection_complete_t {
  bt_byte status;
  bt_hci_conn_state_t* pconn;
  bt_byte reason;
} bt_hci_evt_disconnection_complete_t;
```

Description

This is type bt_hci_evt_disconnection_complete_t.

bt_hci_evt_encryption_change_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_encryption_change_s {
  bt_byte status;
  bt_hci_conn_state_t* pconn;
  bt_byte enabled;
} bt_hci_evt_encryption_change_t;
```

Description

This is type bt_hci_evt_encryption_change_t.

bt_hci_evt_mode_change_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_mode_change_t {
  bt_hci_conn_state_t* pconn;
  bt_byte mode;
  bt_int interval;
} bt_hci_evt_mode_change_t;
```

Description

This is type bt_hci_evt_mode_change_t.

bt_hci_evt_role_change_t Structure

File

hci_evt_handlers.h

C

```
typedef struct _bt_hci_evt_role_change_t {
  bt_hci_conn_state_t* pconn;
  bt_byte role;
} bt_hci_evt_role_change_t;
```

Description

This is type bt_hci_evt_role_change_t.

bt_hci_hconn_p Type

File

hci.h

C

```
typedef bt_int * bt_hci_hconn_p;
```

Description

This is type bt_hci_hconn_p.

bt_hci_hconn_t Type

File

hci.h

C

```
typedef bt_int bt_hci_hconn_t;
```

Description

This is type bt_hci_hconn_t.

bt_hci_inquiry_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_inquiry_callback_fp)(bt_byte status, bt_hci_inquiry_response_t* response);
```

Description

This is type bt_hci_inquiry_callback_fp.

bt_hci_inquiry_response_t Structure

File

hci.h

C

```
typedef struct _bt_hci_inquiry_response_t {
  bt_bdaddr_t bdaddr;
  bt_byte pg_scan_rpt_mode;
  bt_byte pg_scan_period_mode;
  bt_long cod;
  bt_int clock_offset;
  bt_byte rssi;
  bt_byte* eir;
  bt_byte eir_len;
} bt_hci_inquiry_response_t;
```

Description

This is type bt_hci_inquiry_response_t.

bt_hci_le_advertising_report_t Type

File

hci_le.h

C

```
typedef struct _bt_hci_le_advertising_report_t bt_hci_le_advertising_report_t;
```

Description

This is type bt_hci_le_advertising_report_t.

bt_hci_le_conn_state_t Structure

File

hci_le.h

C

```
typedef struct _bt_hci_le_conn_state_t {
  bt_byte own_address_type;
  bt_byte peer_address_type;
  bt_uint conn_interval;
  bt_uint conn_latency;
  bt_uint supervision_timeout;
  bt_byte master_clock_accuracy;
} bt_hci_le_conn_state_t;
```

Description

This is type bt_hci_le_conn_state_t.

bt_hci_le_connect_parameters_t Structure

File

hci_le.h

C

```
typedef struct _bt_hci_le_connect_parameters_t {
  bt_uint scan_interval;
  bt_uint scan_window;
  bt_byte filter_policy;
```

```
  bt_byte own_address_type;
  bt_uint conn_interval_min;
  bt_uint conn_interval_max;
  bt_uint conn_latency;
  bt_uint supervision_timeout;
  bt_uint min_ce_length;
  bt_uint max_ce_length;
} bt_hci_le_connect_parameters_t;
```

Description

This is type bt_hci_le_connect_parameters_t.

bt_hci_le_ctrl_state_t Structure

File

hci_le.h

C

```
typedef struct _bt_hci_le_ctrl_state_t {
  bt_byte flags;
  bt_byte total_num_acl_data_packets;
  bt_int acl_data_packet_len;
  struct _bt_security_mgr_t* sm;
  bt_bdaddr_t bdaddr;
  bt_byte advertising_address_type;
  bt_byte connecting_address_type;
  bt_hci_le_connect_parameters_t def_connect_parameters;
  bt_le_evt_handler handler;
  bt_hci_le_scan_callback_fp scan_cb;
  void* scan_cb_param;
  bt_bool (* connect_ex)(bt_uint scan_interval, bt_uint scan_window, bt_byte filter_policy, bt_byte
peer_address_type, bt_bdaddr_t* peer_address, bt_byte own_address_type, bt_uint conn_interval_min, bt_uint
conn_interval_max, bt_uint conn_latency, bt_uint supervision_timeout, bt_uint min_ce_length, bt_uint
max_ce_length, bt_uint acl_config, bt_hci_connect_callback_fp cb, void *param);
  bt_bool (* set_connect_parameters)(bt_hci_le_connect_parameters_t* params);
  bt_bool (* get_connect_parameters)(bt_hci_le_connect_parameters_t* params);
  void (* sm_long_term_key_request_handler)(bt_hci_event_t* evt);
  bt_uint (* sm_get_session_state)(bt_hci_conn_state_t* conn);
  bt_byte (* sm_get_session_key_size)(bt_hci_conn_state_t* conn);
  bt_bool (* sm_authenticate)(bt_hci_conn_state_t* conn, bt_byte auth_req);
} bt_hci_le_ctrl_state_t;
```

Members

Members	Description
bt_hci_le_connect_parameters_t def_connect_parameters;	default connect parameters
bt_bool (* connect_ex)(bt_uint scan_interval, bt_uint scan_window, bt_byte filter_policy, bt_byte peer_address_type, bt_bdaddr_t* peer_address, bt_byte own_address_type, bt_uint conn_interval_min, bt_uint conn_interval_max, bt_uint conn_latency, bt_uint supervision_timeout, bt_uint min_ce_length, bt_uint max_ce_length, bt_uint acl_config, bt_hci_connect_callback_fp cb, void *param);	API calls

Description

This is type bt_hci_le_ctrl_state_t.

bt_hci_le_evt_connection_updated_t Structure

File

hci_le.h

C

```
typedef struct _bt_hci_le_evt_connection_updated_t {
  bt_byte status;
```

```
  struct _bt_hci_conn_state_s* conn;
} bt_hci_le_evt_connection_updated_t;
```

Description

This is type bt_hci_le_evt_connection_updated_t.

bt_hci_le_evt_read_remote_used_features_completed_t Structure

File

hci_le.h

C

```
typedef struct _bt_hci_le_evt_read_remote_used_features_completed_t {
  bt_byte status;
  struct _bt_hci_conn_state_s* conn;
  bt_byte features[8];
} bt_hci_le_evt_read_remote_used_features_completed_t;
```

Description

This is type bt_hci_le_evt_read_remote_used_features_completed_t.

bt_hci_le_evt_read_support_params_t Structure

File

hci_le.h

C

```
typedef struct _bt_hci_le_evt_read_support_params_t {
  bt_byte supported;
  bt_byte simultaneous_supported;
} bt_hci_le_evt_read_support_params_t;
```

Description

This is type bt_hci_le_evt_read_support_params_t.

bt_hci_le_scan_callback_fp Type

File

hci_le.h

C

```
typedef void (* bt_hci_le_scan_callback_fp)(bt_byte evt, bt_hci_le_advertising_report_t* report, void*
param);
```

Description

This is type bt_hci_le_scan_callback_fp.

bt_hci_link_key_t Structure

File

hci_linkkey_buffer.h

C

```
typedef struct _bt_hci_link_key_s {
  bt_bdaddr_t bdaddr;
  bt_byte linkkey[HCI_LINK_KEY_LEN];
} bt_hci_link_key_t;
```

Description

This is type bt_hci_link_key_t.

bt_hci_listener_t Type

File

hci_conn_state.h

C

```
typedef struct _bt_hci_listener_t bt_hci_listener_t;
```

Description

This is type bt_hci_listener_t.

bt_hci_read_inquiry_mode_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_inquiry_mode_callback_fp)(bt_byte status, bt_byte mode);
```

Description

This is type bt_hci_read_inquiry_mode_callback_fp.

bt_hci_read_inquiry_scan_activity_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_inquiry_scan_activity_callback_fp)(bt_byte status, bt_uint interval, bt_uint
window);
```

Description

This is type bt_hci_read_inquiry_scan_activity_callback_fp.

bt_hci_read_inquiry_scan_type_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_inquiry_scan_type_callback_fp)(bt_byte status, bt_byte scanType);
```

Description

This is type bt_hci_read_inquiry_scan_type_callback_fp.

bt_hci_read_page_scan_activity_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_page_scan_activity_callback_fp)(bt_byte status, bt_uint interval, bt_uint
window);
```

Description

This is type bt_hci_read_page_scan_activity_callback_fp.

bt_hci_read_page_scan_period_mode_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_page_scan_period_mode_callback_fp)(bt_byte status, bt_byte mode);
```

Description

This is type bt_hci_read_page_scan_period_mode_callback_fp.

bt_hci_read_page_scan_type_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_page_scan_type_callback_fp)(bt_byte status, bt_byte scanType);
```

Description

This is type bt_hci_read_page_scan_type_callback_fp.

bt_hci_read_page_timeout_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_read_page_timeout_callback_fp)(bt_byte status, bt_uint timeout);
```

Description

This is type bt_hci_read_page_timeout_callback_fp.

bt_hci_request_remote_name_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_request_remote_name_callback_fp)(bt_bdaddr_p pdbaddr, const char* name);
```

Description

This is type bt_hci_request_remote_name_callback_fp.

bt_hci_sco_read_data_callback_fp Type

File

hci_conn_state.h

C

```
typedef void (* bt_hci_sco_read_data_callback_fp)(bt_hci_conn_state_t* pconn, bt_byte_p data, bt_byte len);
```

Description

This is type bt_hci_sco_read_data_callback_fp.

bt_hci_start_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_start_callback_fp)(bt_bool success, void* param);
```

Description

brief HCI initialization callback. ingroup hci

details This typedef defines a function pointer type for the HCI initialization callback functions. Such a function must be passed to the bt_hci_start() function.

param success Specifies whether HCI initialization succeeded or not.

bt_hci_stop_callback_fp Type

File

hci.h

C

```
typedef void (* bt_hci_stop_callback_fp)(void* param);
```

Description

brief HCI stop callback. ingroup hci

details This typedef defines a function pointer type for the HCI stop callback functions. Such a function must be passed to the bt_hci_stop() function.

bt_hci_transport_recv_packet_callback_fp Type

File

hci_transport.h

C

```
typedef void (* bt_hci_transport_recv_packet_callback_fp)(bt_uint len);
```

Description

This is type bt_hci_transport_recv_packet_callback_fp.

bt_hci_transport_send_packet_callback_fp Type

File

hci_transport.h

C

```
typedef void (* bt_hci_transport_send_packet_callback_fp)(void);
```

Description

This is type bt_hci_transport_send_packet_callback_fp.

bt_hcitr_tih4_power_callback_fp Type

File

hcitr_tih4.h

C

```
typedef void (* bt_hcitr_tih4_power_callback_fp)(bt_hcitr_tih4_power_event_e event);
```

Description

This is type bt_hcitr_tih4_power_callback_fp.

bt_hcitr_tih4_power_event_e Enumeration

File

hcitr_tih4.h

C

```
typedef enum _hcitr_tih4_power_event_e {
  HCITR_TIH4_POWER_EVENT_PREPARE_TO_SLEEP,
  HCITR_TIH4_POWER_EVENT_SLEEP,
  HCITR_TIH4_POWER_EVENT_WAKE_UP,
  HCITR_TIH4_POWER_EVENT_AWAKE
} bt_hcitr_tih4_power_event_e;
```

Description

This is type bt_hcitr_tih4_power_event_e.

bt_le_evt_handler Type

File

hci_le.h

C

```
typedef void (* bt_le_evt_handler)(bt_hci_event_t* evt);
```

Description

This is type bt_le_evt_handler.

hci_transport_t Structure

File

hci_transport.h

C

```
typedef struct _hci_transport_t {
  void (* send_packet)(const bt_byte* buffer, bt_uint len, bt_hci_transport_send_packet_callback_fp
callback);
  void (* recv_packet)(bt_byte* buffer, bt_uint len, bt_hci_transport_recv_packet_callback_fp callback);
} hci_transport_t;
```

Description

This is type hci_transport_t.

pf_hci_sleep_callback Type

File

hci.h

C

```
typedef void (* pf_hci_sleep_callback)(bt_bool status);
```

Description

This is type pf_hci_sleep_callback.

pf_hci_wakeup_callback Type

File

hci.h

C

```
typedef void (* pf_hci_wakeup_callback)(bt_bool status);
```

Description

This is type pf_hci_wakeup_callback.

pf_l2cap_receive_callback Type

File

hci_ctrl_state.h

C

```
typedef void (* pf_l2cap_receive_callback)(struct _bt_l2cap_mgr_s*, bt_hci_conn_state_p, bt_byte_p, bt_int
len);
```

Description

This is type pf_l2cap_receive_callback.

_bt_hci_conn_state_s Structure

File

hci_conn_state.h

C

```
struct _bt_hci_conn_state_s {
  bt_hci_hconn_t hconn;
  bt_bdaddr_t bdaddr_remote;
  bt_byte type;
  bt_byte state;
  bt_byte mode;
  bt_byte role;
  bt_long l2cap_channel_close_time;
  bt_uint acl_config;
  bt_byte link_type;
  bt_signal_t signal;
  bt_int outstanding_acl_packet_count;
  bt_hci_le_conn_state_t* le_conn_state;
  bt_ulong data_rx_time;
  bt_ulong data_tx_time;
  bt_byte idle;
  bt_queue_element_t* queue;
  bt_byte* recv_data;
  bt_int recv_data_len;
  bt_int recv_data_pos;
  bt_hci_sco_read_data_callback_fp sco_read_data_callback;
  void* sco_read_data_param;
  bt_hci_listener_t* listeners;
};
```

Members

Members	Description
bt_queue_element_t* queue;	queue for sending outgoing data packets
bt_byte* recv_data;	buffer to receive incoming packets
bt_hci_sco_read_data_callback_fp sco_read_data_callback;	bt_hci_disconnect_callback_fp hci_disconnect_callback; void *hci_disconnect_param;

Description

This is record _bt_hci_conn_state_s.

_bt_hci_ctrl_listener_t Structure

File

hci_ctrl_state.h

C

```
struct _bt_hci_ctrl_listener_t {
  bt_hci_ctrl_listener_t* next_listener;
  bt_byte listener_type;
  bt_uint event_id;
  union {
    void* ptr;
    bt_hci_event_listener_fp hci_event;
    bt_hci_cmd_listener_fp cmd_event;
    bt_hci_data_listener_fp data;
  } callback;
  void* callback_param;
};
```

Description

This is record _bt_hci_ctrl_listener_t.

_bt_hci_le_advertising_report_t Structure

File

hci_le.h

C

```
struct _bt_hci_le_advertising_report_t {
  bt_byte event_type;
  bt_byte address_type;
  bt_bdaddr_t bdaddr;
  bt_byte data_len;
  bt_byte* data;
  bt_byte rssi;
};
```

Description

This is record _bt_hci_le_advertising_report_t.

_bt_hci_listener_t Structure

File

hci_conn_state.h

C

```
struct _bt_hci_listener_t {
  bt_hci_listener_t* next_listener;
  bt_byte event_id;
  union {
    void* ptr;
    bt_hci_disconnect_callback_fp disconnect;
  } callback;
  void* callback_param;
};
```

Description

This is record _bt_hci_listener_t.

HCI_DISCOVERABLE_MODE_GENERAL Macro

File

hci.h

C

```
#define HCI_DISCOVERABLE_MODE_GENERAL 0
```

Description

This is macro HCI_DISCOVERABLE_MODE_GENERAL.

HCI_DISCOVERABLE_MODE_LIMITED Macro

File

hci.h

C

```
#define HCI_DISCOVERABLE_MODE_LIMITED 1
```

Description

This is macro HCI_DISCOVERABLE_MODE_LIMITED.

HCI_EIR_TYPE_3D_Information_Data Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_3D_Information_Data 0x3D
```

Description

This is macro HCI_EIR_TYPE_3D_Information_Data.

HCI_EIR_TYPE_ADVERTISING_INTERVAL Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_ADVERTISING_INTERVAL 0x1A
```

Description

This is macro HCI_EIR_TYPE_ADVERTISING_INTERVAL.

HCI_EIR_TYPE_APPEARANCE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_APPEARANCE 0x19
```

Description

This is macro HCI_EIR_TYPE_APPEARANCE.

HCI_EIR_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS 0x1B
```

Description

This is macro HCI_EIR_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS.

HCI_EIR_TYPE_LE_ROLE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_LE_ROLE 0x1C
```

Description

This is macro HCI_EIR_TYPE_LE_ROLE.

HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_CONFIRMATION_VALUE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_CONFIRMATION_VALUE 0x22
```

Description

This is macro HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_CONFIRMATION_VALUE.

HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_RANDOM_VALUE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_RANDOM_VALUE 0x23
```

Description

This is macro HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_RANDOM_VALUE.

HCI_EIR_TYPE_PUBLIC_TARGET_ADDRESS Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_PUBLIC_TARGET_ADDRESS 0x17
```

Description

This is macro HCI_EIR_TYPE_PUBLIC_TARGET_ADDRESS.

HCI_EIR_TYPE_RANDOM_TARGET_ADDRESS Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_RANDOM_TARGET_ADDRESS 0x18
```

Description

This is macro HCI_EIR_TYPE_RANDOM_TARGET_ADDRESS.

HCI_EIR_TYPE_SERVICE_DATA Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SERVICE_DATA 0x16
```

Description

This is macro HCI_EIR_TYPE_SERVICE_DATA.

HCI_EIR_TYPE_SERVICE_DATA_UUID128 Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SERVICE_DATA_UUID128 0x21
```

Description

This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID128.

HCI_EIR_TYPE_SERVICE_DATA_UUID16 Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SERVICE_DATA_UUID16 0x16
```

Description

This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID16.

HCI_EIR_TYPE_SERVICE_DATA_UUID32 Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SERVICE_DATA_UUID32 0x20
```

Description

This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID32.

HCI_EIR_TYPE_SIMPLE_PAIRING_HASH_C_256 Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SIMPLE_PAIRING_HASH_C_256 0x1D
```

Description

This is macro HCI_EIR_TYPE_SIMPLE_PAIRING_HASH_C_256.

HCI_EIR_TYPE_SIMPLE_PAIRING_RANDOMIZER_R_256 Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SIMPLE_PAIRING_RANDOMIZER_R_256 0x1E
```

Description

This is macro HCI_EIR_TYPE_SIMPLE_PAIRING_RANDOMIZER_R_256.

HCI_EIR_TYPE_SLAVE_CONN_INTERVAL_RANGE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SLAVE_CONN_INTERVAL_RANGE 0x12
```

Description

This is macro HCI_EIR_TYPE_SLAVE_CONN_INTERVAL_RANGE.

HCI_EIR_TYPE_SM_OOB_FLAGS Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SM_OOB_FLAGS 0x11
```

Description

This is macro HCI_EIR_TYPE_SM_OOB_FLAGS.

HCI_EIR_TYPE_SM_TK_VALUE Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SM_TK_VALUE 0x10
```

Description

This is macro HCI_EIR_TYPE_SM_TK_VALUE.

HCI_EIR_TYPE_SOLICITATION_UUID128_LIST Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SOLICITATION_UUID128_LIST 0x15
```

Description

This is macro HCI_EIR_TYPE_SOLICITATION_UUID128_LIST.

HCI_EIR_TYPE_SOLICITATION_UUID16_LIST Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SOLICITATION_UUID16_LIST 0x14
```

Description

This is macro HCI_EIR_TYPE_SOLICITATION_UUID16_LIST.

HCI_EIR_TYPE_SOLICITATION_UUID32_LIST Macro

File

hci_eir.h

C

```
#define HCI_EIR_TYPE_SOLICITATION_UUID32_LIST 0x1F
```

Description

This is macro HCI_EIR_TYPE_SOLICITATION_UUID32_LIST.

HCI_ERR_CONNECTION_TIMEOUT Macro

File

hci_errors.h

C

```
#define HCI_ERR_CONNECTION_TIMEOUT 0x08
```

Description

This is macro HCI_ERR_CONNECTION_TIMEOUT.

HCI_INIT_FLAG_IGNORE_TOTAL_NUM_ACL_DATA_PACKETS Macro

File

hci_private.h

C

```
#define HCI_INIT_FLAG_IGNORE_TOTAL_NUM_ACL_DATA_PACKETS 2 // The total number of ACL data packets read from the controller will be ignored.
```

Description

The total number of ACL data packets read from the controller will be ignored. The stack will assume that he controller can accept only 1 ACL packet and sends HCI_EVT_NUM_OF_COMPLETED_PACKETS for each packet it has processed. This seems to be for controller working over SDIO (at least for Marvell's 88W8777).

HCI_INIT_FLAG_SEND_HCI_RESET Macro

File

hci_private.h

C

```
#define HCI_INIT_FLAG_SEND_HCI_RESET 1 // HCI reset will be sent when bt_hci_init is called
```

Description

HCI reset will be sent when bt_hci_init is called

HCI_LE_ADV_CHANNEL_MAP_ENABLE_37 Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_CHANNEL_MAP_ENABLE_37 0x01
```

Description

This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_37.

HCI_LE_ADV_CHANNEL_MAP_ENABLE_38 Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_CHANNEL_MAP_ENABLE_38 0x02
```

Description

This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_38.

HCI_LE_ADV_CHANNEL_MAP_ENABLE_39 Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_CHANNEL_MAP_ENABLE_39 0x04
```

Description

This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_39.

HCI_LE_ADV_CHANNEL_MAP_ENABLE_ALL Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_CHANNEL_MAP_ENABLE_ALL 0x07
```

Description

This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_ALL.

HCI_LE_ADV_CHANNEL_MAP_RESERVED Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_CHANNEL_MAP_RESERVED 0x00
```

Description

This is macro HCI_LE_ADV_CHANNEL_MAP_RESERVED.

HCI_LE_ADV_TYPE_CONN_UNDIRECT Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_TYPE_CONN_UNDIRECT 0x00 // Connectable undirected advertising (ADV_IND) (default)
```

Description

Connectable undirected advertising (ADV_IND) (default)

HCI_LE_ADV_TYPE_DIRECT_HIGH Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_TYPE_DIRECT_HIGH 0x01 // Connectable high duty cycle directed advertising
(ADV_DIRECT_IND, high duty cycle)
```

Description

Connectable high duty cycle directed advertising (ADV_DIRECT_IND, high duty cycle)

HCI_LE_ADV_TYPE_DIRECT_LOW Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_TYPE_DIRECT_LOW 0x04 // Connectable low duty cycle directed advertising (ADV_DIRECT_IND,
low duty cycle)
```

Description

Connectable low duty cycle directed advertising (ADV_DIRECT_IND, low duty cycle)

HCI_LE_ADV_TYPE_NONCONN Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_TYPE_NONCONN 0x03 // Non connectable undirected advertising (ADV_NONCONN_IND)
```

Description

Non connectable undirected advertising (ADV_NONCONN_IND)

HCI_LE_ADV_TYPE_SCAN Macro

File

hci_le.h

C

```
#define HCI_LE_ADV_TYPE_SCAN 0x02 // Scannable undirected advertising (ADV_SCAN_IND)
```

Description

Scannable undirected advertising (ADV_SCAN_IND)

HCI_LE_RANDOM_ADDRESS_TYPE_NON_RESOLVABLE Macro

File

hci_le.h

C

```
#define HCI_LE_RANDOM_ADDRESS_TYPE_NON_RESOLVABLE 2
```

Description

This is macro HCI_LE_RANDOM_ADDRESS_TYPE_NON_RESOLVABLE.

HCI_LE_RANDOM_ADDRESS_TYPE_RESOLVABLE Macro

File

hci_le.h

C

```
#define HCI_LE_RANDOM_ADDRESS_TYPE_RESOLVABLE 3
```

Description

This is macro HCI_LE_RANDOM_ADDRESS_TYPE_RESOLVABLE.

HCI_LE_RANDOM_ADDRESS_TYPE_STATIC Macro

File

hci_le.h

C

```
#define HCI_LE_RANDOM_ADDRESS_TYPE_STATIC 1
```

Description

This is macro HCI_LE_RANDOM_ADDRESS_TYPE_STATIC.

HCI_LINK_POLICY_ENABLE_ALL Macro

File

hci.h

C

```
#define HCI_LINK_POLICY_ENABLE_ALL 0x000F
```

Description

This is macro HCI_LINK_POLICY_ENABLE_ALL.

bt_hci_le_cancel_connect Macro

File

hci_le.h

C

```
#define bt_hci_le_cancel_connect(cb) bt_hci_le_cancel_connect_ex(cb, NULL)
```

Description

This is macro bt_hci_le_cancel_connect.

bt_hci_send_pin_code Macro

File

hci.h

C

```
#define bt_hci_send_pin_code(pbdaddr_remote, pincode) bt_hci_send_pin_code_ex(pbdaddr_remote, pincode,
NULL, NULL)
```

Description

This is macro bt_hci_send_pin_code.

HCI_ERR_PAIRING_NOT_ALLOWED Macro

File

hci_errors.h

C

```
#define HCI_ERR_PAIRING_NOT_ALLOWED 0x18
```

Description

This is macro HCI_ERR_PAIRING_NOT_ALLOWED.

HCI_ERR_PIN_OR_KEY_MISSING Macro

File

hci_errors.h

C

```
#define HCI_ERR_PIN_OR_KEY_MISSING 0x06
```

Description

This is macro HCI_ERR_PIN_OR_KEY_MISSING.

HCI_PACKET_BOUNDARY_COMPLETE Macro

File

hci.h

C

```
#define HCI_PACKET_BOUNDARY_COMPLETE 0x3
```

Description

This is macro HCI_PACKET_BOUNDARY_COMPLETE.

HCI_SUCCESS Macro

File

hci_errors.h

C

```
#define HCI_SUCCESS 0x00
```

Description

This is macro HCI_SUCCESS.

L2CAP Functions

bt_l2cap_alloc_cmd_buffer Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_header_p bt_l2cap_alloc_cmd_buffer(bt_byte code);
```

Description

This is function bt_l2cap_alloc_cmd_buffer.

bt_l2cap_alloc_cmd_config_req Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_config_req_p bt_l2cap_alloc_cmd_config_req();
```

Description

This is function bt_l2cap_alloc_cmd_config_req.

bt_l2cap_alloc_cmd_config_res Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_config_res_p bt_l2cap_alloc_cmd_config_res();
```

Description

This is function bt_l2cap_alloc_cmd_config_res.

bt_l2cap_alloc_cmd_conn_param_update_req Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_conn_param_update_req_t* bt_l2cap_alloc_cmd_conn_param_update_req();
```

Description

This is function bt_l2cap_alloc_cmd_conn_param_update_req.

bt_l2cap_alloc_cmd_conn_param_update_res Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_conn_param_update_res_t* bt_l2cap_alloc_cmd_conn_param_update_res();
```

Description

This is function bt_l2cap_alloc_cmd_conn_param_update_res.

bt_l2cap_alloc_cmd_connection_req Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_connection_req_p bt_l2cap_alloc_cmd_connection_req();
```

Description

This is function bt_l2cap_alloc_cmd_connection_req.

bt_l2cap_alloc_cmd_connection_res Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_connection_res_p bt_l2cap_alloc_cmd_connection_res();
```

Description

This is function bt_l2cap_alloc_cmd_connection_res.

bt_l2cap_alloc_cmd_disconnection_req Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_disconnection_req_p bt_l2cap_alloc_cmd_disconnection_req();
```

Description

This is function bt_l2cap_alloc_cmd_disconnection_req.

bt_l2cap_alloc_cmd_disconnection_res Function

File

cmdbuffer.h

C

```
pcmd_disconnection_res bt_l2cap_alloc_cmd_disconnection_res();
```

Description

This is function bt_l2cap_alloc_cmd_disconnection_res.

bt_l2cap_alloc_cmd_echo_req Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_echo_req_p bt_l2cap_alloc_cmd_echo_req();
```

Description

This is function bt_l2cap_alloc_cmd_echo_req.

bt_l2cap_alloc_cmd_echo_res Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_echo_res_p bt_l2cap_alloc_cmd_echo_res();
```

Description

This is function bt_l2cap_alloc_cmd_echo_res.

bt_l2cap_alloc_cmd_info_req Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_info_req_p bt_l2cap_alloc_cmd_info_req();
```

Description

This is function bt_l2cap_alloc_cmd_info_req.

bt_l2cap_alloc_cmd_info_res Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_info_res_p bt_l2cap_alloc_cmd_info_res();
```

Description

This is function bt_l2cap_alloc_cmd_info_res.

bt_l2cap_alloc_cmd_reject Function

File

cmdbuffer.h

C

```
bt_l2cap_cmd_reject_p bt_l2cap_alloc_cmd_reject();
```

Description

This is function bt_l2cap_alloc_cmd_reject.

bt_l2cap_alloc_frame_buffer Function

File

frame_buffer.h

C

```
bt_byte_p bt_l2cap_alloc_frame_buffer();
```

Description

This is function bt_l2cap_alloc_frame_buffer.

bt_l2cap_allocate_channel Function

File

chmanager.h

C

```
bt_l2cap_channel_t* bt_l2cap_allocate_channel(bt_l2cap_mgr_p pmgr, bt_int mode, bt_int type);
```

Description

This is function bt_l2cap_allocate_channel.

bt_l2cap_allocate_fixed_channel Function

File

l2cap_fixed_channel.h

C

```
bt_l2cap_fixed_channel_t* bt_l2cap_allocate_fixed_channel(bt_l2cap_mgr_p pmgr, bt_id cid, bt_byte
link_type);
```

Description

This is function bt_l2cap_allocate_fixed_channel.

bt_l2cap_allocate_mgr Function

File

l2cap.h

C

```
bt_l2cap_mgr_t* bt_l2cap_allocate_mgr(bt_hci_ctrl_state_t* hci_ctrl);
```

Description

brief Allocate L2CAP manager. ingroup l2cap

details This function allocates and initializes an L2CAP manager structure. One L2CAP manager manages all L2CAP connections for a particular local device. The local device is specified by the c hci_ctrl parameter.

param hci_ctrl Pointer to the hci_ctrl_state structure that represents the local device (HCI controller) for which a L2CAP manager is to be allocated.

return A pointer to the allocated L2CAP manager structure. The returned L2CAP manager should be freed by a call to bt_l2cap_free_mgr when it is no longer needed.

bt_l2cap_allocate_psm Function

File

l2cap_psm.h

C

```
bt_l2cap_psm_t* bt_l2cap_allocate_psm(bt_l2cap_mgr_p pmgr, bt_int psmId);
```

Description

This is function bt_l2cap_allocate_psm.

bt_l2cap_connect_ext Function

File

l2cap.h

C

```
bt_bool bt_l2cap_connect_ext(bt_l2cap_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_int psm, bt_byte chmode,
bt_uint acl_config, bt_l2cap_connect_callback_fp connect_cb, void* param,
```

```
bt_l2cap_state_changed_callback_fp state_cb);
```

Description

brief Connect to a remote device. ingroup l2cap

details This function establishes an L2CAP connection with a remote device on a specific PSM. When connect operation completes a callback function is called.

param mgr The L2CAP manager. param remote_addr The address of the remote device. param psm The PSM for the connection. param connect_cb The Callback function that is called when the connect operation completes. param param A pointer to arbitrary data to be passed to the c connect_cb callback. param state_cb The callback function that is called when the state of the established connection changes.

return li c TRUE when the function succeeds. li c FALSE otherwise. None of the callback functions is called in this case.

bt_l2cap_connect_fixed_channel Function

File

l2cap.h

C

```
bt_bool bt_l2cap_connect_fixed_channel(bt_l2cap_mgr_p pmgr, bt_bdaddr_p pbdaddr_remote, bt_id cid, bt_uint
acl_config, bt_l2cap_connect_callback_fp cb, void* param, bt_l2cap_state_changed_callback_fp state_cb);
```

Description

This is function bt_l2cap_connect_fixed_channel.

bt_l2cap_disconnect Function

File

l2cap.h

C

```
bt_bool bt_l2cap_disconnect(bt_l2cap_channel_t* ch);
```

Description

brief Close L2CAP channel. ingroup l2cap

details This function closes an L2CAP channel. The channel can be established either by a call to bt_l2cap_connect() or by an incoming connection request.

param ch The L2CAP channel to be closed.

return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_l2cap_echo Function

File

l2cap.h

C

```
bt_bool bt_l2cap_echo(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_byte_p data, bt_int len);
```

Description

brief Send echo command

bt_l2cap_find_fixed_channel Function

File

l2cap_fixed_channel.h

C

```
bt_l2cap_fixed_channel_t* bt_l2cap_find_fixed_channel(const bt_l2cap_mgr_p pmgr, bt_id cid, bt_byte
link_type);
```

Description

This is function bt_l2cap_find_fixed_channel.

bt_l2cap_find_psm Function

File

l2cap_psm.h

C

```
bt_l2cap_psm_t* bt_l2cap_find_psm(const bt_l2cap_mgr_p pmgr, bt_int psmId);
```

Description

This is function bt_l2cap_find_psm.

bt_l2cap_free_cmd_buffer Function

File

cmdbuffer.h

C

```
void bt_l2cap_free_cmd_buffer(void* p);
```

Description

This is function bt_l2cap_free_cmd_buffer.

bt_l2cap_free_fixed_channel Function

File

l2cap_fixed_channel.h

C

```
void bt_l2cap_free_fixed_channel(bt_l2cap_fixed_channel_t* fc);
```

Description

This is function bt_l2cap_free_fixed_channel.

bt_l2cap_free_frame_buffer Function

File

frame_buffer.h

C

```
void bt_l2cap_free_frame_buffer(bt_byte_p data);
```

Description

This is function bt_l2cap_free_frame_buffer.

bt_l2cap_free_mgr Function

File

l2cap.h

C

```
void bt_l2cap_free_mgr(bt_l2cap_mgr_t* mgr);
```

Description

brief Release L2CAP manger. ingroup l2cap

details This function releases the L2CAP manager structure.

param mgr The L2CAP manager structure to be released.

bt_l2cap_free_psm Function

File

l2cap_psm.h

C

```
void bt_l2cap_free_psm(bt_l2cap_psm_t* psm);
```

Description

This is function bt_l2cap_free_psm.

bt_l2cap_get_channel Function

File

chmanager.h

C

```
bt_l2cap_channel_t* bt_l2cap_get_channel(const bt_l2cap_mgr_p pmgr, bt_id cid);
```

Description

This is function bt_l2cap_get_channel.

bt_l2cap_get_channel_by_bdaddr_cid Function

File

chmanager.h

C

```
bt_l2cap_channel_t* bt_l2cap_get_channel_by_bdaddr_cid(const bt_l2cap_mgr_p pmgr, bt_bdaddr_t* bdaddr,
bt_id cid);
```

Description

This is function bt_l2cap_get_channel_by_bdaddr_cid.

bt_l2cap_get_channel_by_hconn_cid Function

File

chmanager.h

C

```
bt_l2cap_channel_t* bt_l2cap_get_channel_by_hconn_cid(const bt_l2cap_mgr_p pmgr, bt_hci_hconn_t hconn,
bt_id cid_dest);
```

Description

This is function bt_l2cap_get_channel_by_hconn_cid.

bt_l2cap_get_channel_by_hconn_dest_cid Function

File

chmanager.h

C

```
bt_l2cap_channel_t* bt_l2cap_get_channel_by_hconn_dest_cid(const bt_l2cap_mgr_p pmgr, bt_hci_hconn_t hconn,
bt_id cid_dest);
```

Description

This is function bt_l2cap_get_channel_by_hconn_dest_cid.

bt_l2cap_get_channel_by_psm Function

File

chmanager.h

C

```
bt_l2cap_channel_t* bt_l2cap_get_channel_by_psm(const bt_l2cap_mgr_p pmgr, bt_int psm);
```

Description

This is function bt_l2cap_get_channel_by_psm.

bt_l2cap_get_mgr Function

File

l2cap.h

C

```
bt_l2cap_mgr_p bt_l2cap_get_mgr(bt_hci_hconn_t h);
```

Description

This is function bt_l2cap_get_mgr.

bt_l2cap_init Function

File

l2cap.h

C

```
void bt_l2cap_init();
```

Description

brief Initialize the L2CAP layer. ingroup l2cap

details This function initializes the L2CAP layer of the stack. It must be called prior to any other L2CAP function can be called.

bt_l2cap_init_channels Function

File

chmanager.h

C

```
void bt_l2cap_init_channels(bt_l2cap_mgr_p pmgr);
```

Description

This is function bt_l2cap_init_channels.

bt_l2cap_init_cmd_buffers Function

File

cmdbuffer.h

C

```
void bt_l2cap_init_cmd_buffers();
```

Description

This is function bt_l2cap_init_cmd_buffers.

bt_l2cap_init_frame_buffers Function

File

frame_buffer.h

C

```
void bt_l2cap_init_frame_buffers();
```

Description

This is function bt_l2cap_init_frame_buffers.

bt_l2cap_init_psms Function

File

l2cap_psm.h

C

```
void bt_l2cap_init_psms(bt_l2cap_mgr_p pmgr);
```

Description

This is function bt_l2cap_init_psms.

bt_l2cap_listen_ext Function

File

l2cap.h

C

```
bt_bool bt_l2cap_listen_ext(bt_l2cap_mgr_t* mgr, bt_int psm, bt_byte chmode, bt_uint acl_config,
bt_l2cap_listen_callback_fp callback, void* param);
```

Description

brief Listen for incoming connections. ingroup l2cap

details This function tells the L2CAP manager to listen for incoming connections on a specific PSM. When a connection is established a callback function is called.

param mgr The L2CAP manager. param psm The PSM on which the manager will listen and accept incoming connections. param callback The callback function that will be called when an incoming connection is established. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter.

return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_l2cap_listen_fixed_channel Function

File

l2cap.h

C

```
bt_bool bt_l2cap_listen_fixed_channel(bt_l2cap_mgr_t* mgr, bt_id cid, bt_byte link_type,
bt_l2cap_listen_callback_fp callback, void* param);
```

Description

This is function bt_l2cap_listen_fixed_channel.

bt_l2cap_packet_cmd_assembler Function

File

l2cap_packet.h

C

```
bt_int bt_l2cap_packet_cmd_assembler(bt_packet_t* packet, bt_byte* buffer, bt_int buffer_len);
```

Description

This is function bt_l2cap_packet_cmd_assembler.

bt_l2cap_packet_data_assembler Function

File

l2cap_packet.h

C

```
bt_int bt_l2cap_packet_data_assembler(bt_packet_t* packet, bt_byte* buffer, bt_int buffer_len);
```

Description

This is function bt_l2cap_packet_data_assembler.

bt_l2cap_read_data Function

File

channel.h

C

```
bt_bool bt_l2cap_read_data(bt_l2cap_channel_t* channel, bt_l2cap_read_data_callback_fp callback);
```

Description

This is function bt_l2cap_read_data.

bt_l2cap_reject Function

File

l2cap.h

C

```
bt_bool bt_l2cap_reject(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_byte cmd_id, bt_byte reason,
bt_l2cap_cmd_reject_param_t* param);
```

Description

brief Send reject command (used to reject unknown or invalid commands)

bt_l2cap_send_cmd Function

File

channel.h

C

```
bt_bool bt_l2cap_send_cmd(struct _bt_l2cap_mgr_s * mgr, struct _bt_hci_conn_state_s * conn,
bt_l2cap_cmd_header_p cmd, pf_l2cap_cmd_callback cb);
```

Description

This is function bt_l2cap_send_cmd.

bt_l2cap_send_data Function

File

channel.h

C

```
bt_bool bt_l2cap_send_data(bt_l2cap_channel_t* channel, const bt_byte_p data, bt_int len,
bt_l2cap_send_data_callback_fp callback, void* cb_param);
```

Description

brief Send data over an L2CAP channel ingroup l2cap

details This function sends data over the specified L2CAP channel.

param channel The L2CAP channel to send data over. param data The pointer to the data. param len The length of the data. param callback The callback function that is called when sending the data has been completed.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_l2cap_update_conn_parameters Function

File

l2cap.h

C

```
bt_bool bt_l2cap_update_conn_parameters(bt_l2cap_channel_t* channel, bt_uint min_interval, bt_uint
max_interval, bt_uint slave_latency, bt_uint supervision_timeout);
```

Description

This is function bt_l2cap_update_conn_parameters.

_bt_l2cap_clear_channel_cmd_queue Function

File

l2cap_private.h

C

```
void _bt_l2cap_clear_channel_cmd_queue(bt_l2cap_channel_t* channel);
```

Description

This is function _bt_l2cap_clear_channel_cmd_queue.

_bt_l2cap_eretr_handle_xmit_event Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_eretr_handle_xmit_event(bt_l2cap_xmit_event_param_t* param);
```

Description

This is function _bt_l2cap_eretr_handle_xmit_event.

_bt_l2cap_eretr_retr_frames Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_eretr_retr_frames(bt_l2cap_channel_t* pch, bt_byte fbit);
```

Description

This is function _bt_l2cap_eretr_retr_frames.

_bt_l2cap_fcs Function

File

l2cap_eretr.h

C

```
bt_uint _bt_l2cap_fcs(const bt_byte* message, bt_int len, bt_uint start_crc);
```

Description

This is function _bt_l2cap_fcs.

_bt_l2cap_get_tick_count Function

File

l2cap_timer.h

C

```
bt_long _bt_l2cap_get_tick_count();
```

Description

This is function _bt_l2cap_get_tick_count.

_bt_l2cap_init_signal Function

File

l2cap_signal.h

C

```
void _bt_l2cap_init_signal();
```

Description

This is function _bt_l2cap_init_signal.

_bt_l2cap_init_timer Function

File

l2cap_timer.h

C

```
void _bt_l2cap_init_timer();
```

Description

This is function _bt_l2cap_init_timer.

_bt_l2cap_notify_and_remove Function

File

l2cap_command_queue.h

C

```
void _bt_l2cap_notify_and_remove(struct _bt_l2cap_mgr_s* pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _bt_l2cap_notify_and_remove.

_bt_l2cap_send_commands_from_queue Function

File

l2cap_command_queue.h

C

```
void _bt_l2cap_send_commands_from_queue(struct _bt_l2cap_mgr_s* pmgr);
```

Description

This is function _bt_l2cap_send_commands_from_queue.

_bt_l2cap_set_signal Function

File

l2cap_signal.h

C

```
void _bt_l2cap_set_signal();
```

Description

This is function _bt_l2cap_set_signal.

_l2cap_allocate_buffers Function

File

l2cap_private.h

C

```
void _l2cap_allocate_buffers();
```

Description

Defined by OEM through library configuration

_l2cap_data_receive_callback Function

File

l2cap_private.h

C

```
void _l2cap_data_receive_callback(bt_l2cap_mgr_p pmg, bt_hci_conn_state_p pconn, bt_byte_p pdata, bt_int
len);
```

Description

From l2cap_recv.c

_pack_cmd_reject Function

File

l2cap_private.h

C

```
bt_int _pack_cmd_reject(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p poffset);
```

Description

This is function _pack_cmd_reject.

_pack_config_request Function

File

l2cap_private.h

C

```
bt_int _pack_config_request(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_config_request.

_pack_config_response Function

File

l2cap_private.h

C

```
bt_int _pack_config_response(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_config_response.

_pack_conn_param_update_request Function

File

l2cap_private.h

C

```
bt_int _pack_conn_param_update_request(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len,
bt_int_p poffset);
```

Description

This is function _pack_conn_param_update_request.

_pack_conn_param_update_response Function

File

l2cap_private.h

C

```
bt_int _pack_conn_param_update_response(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len,
bt_int_p poffset);
```

Description

This is function _pack_conn_param_update_response.

_pack_conn_request Function

File

l2cap_private.h

C

```
bt_int _pack_conn_request(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_conn_request.

_pack_conn_response Function

File

l2cap_private.h

C

```
bt_int _pack_conn_response(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_conn_response.

_pack_dconn_request Function

File

l2cap_private.h

C

```
bt_int _pack_dconn_request(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_dconn_request.

_pack_dconn_response Function

File

l2cap_private.h

C

```
bt_int _pack_dconn_response(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_dconn_response.

_pack_echo_request Function

File

l2cap_private.h

C

```
bt_int _pack_echo_request(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_echo_request.

_pack_echo_response Function

File

l2cap_private.h

C

```
bt_int _pack_echo_response(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_echo_response.

_pack_info_request Function

File

l2cap_private.h

C

```
bt_int _pack_info_request(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_info_request.

_pack_info_response Function

File

l2cap_private.h

C

```
bt_int _pack_info_response(bt_l2cap_cmd_header_t* pcmd, bt_byte_p buffer, bt_int buffer_len, bt_int_p
poffset);
```

Description

This is function _pack_info_response.

_process_config_req Function

File

l2cap_private.h

C

```
void _process_config_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_config.c

_process_config_res Function

File

l2cap_private.h

C

```
void _process_config_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _process_config_res.

_process_conn_param_update_req Function

File

l2cap_private.h

C

```
void _process_conn_param_update_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_t*
pcmd);
```

Description

This is function _process_conn_param_update_req.

_process_conn_param_update_res Function

File

l2cap_private.h

C

```
void _process_conn_param_update_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_t* pcmd);
```

Description

This is function _process_conn_param_update_res.

_process_conn_req Function

File

l2cap_private.h

C

```
void _process_conn_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_connect.c

_process_conn_res Function

File

l2cap_private.h

C

```
void _process_conn_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _process_conn_res.

_process_dconn_req Function

File

l2cap_private.h

C

```
void _process_dconn_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_disconnect.c

_process_dconn_res Function

File

l2cap_private.h

C

```
void _process_dconn_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _process_dconn_res.

_process_echo_req Function

File

l2cap_private.h

C

```
void _process_echo_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_echo.c

_process_echo_res Function

File

l2cap_private.h

C

```
void _process_echo_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _process_echo_res.

_process_info_req Function

File

l2cap_private.h

C

```
void _process_info_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_info.c

_process_info_res Function

File

l2cap_private.h

C

```
void _process_info_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _process_info_res.

_process_reject Function

File

l2cap_private.h

C

```
void _process_reject(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_reject.c

_process_unknown_req Function

File

l2cap_private.h

C

```
void _process_unknown_req(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn, bt_l2cap_cmd_header_p pcmd);
```

Description

From cmd_unknown.c

_process_unknown_res Function

File

l2cap_private.h

C

```
void _process_unknown_res(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is function _process_unknown_res.

_read_cmd_reject Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_cmd_reject(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_cmd_reject.

_read_config_request Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_config_request(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_config_request.

_read_config_response Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_config_response(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_config_response.

_read_conn_param_update_request Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_conn_param_update_request(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_conn_param_update_request.

_read_conn_param_update_response Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_conn_param_update_response(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_conn_param_update_response.

_read_conn_request Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_conn_request(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_conn_request.

_read_conn_response Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_conn_response(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_conn_response.

_read_dconn_request Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_dconn_request(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_dconn_request.

_read_dconn_response Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_dconn_response(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_dconn_response.

_read_echo_request Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_echo_request(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_echo_request.

_read_echo_response Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_echo_response(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_echo_response.

_read_info_request Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_info_request(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_info_request.

_read_info_response Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p _read_info_response(bt_byte_p pdata, bt_int len, bt_int_p poffset);
```

Description

This is function _read_info_response.

_bt_l2cap_eretr_pack_config_request Function

File

l2cap_private.h

C

```
void _bt_l2cap_eretr_pack_config_request(bt_l2cap_channel_t* channel, bt_byte* buffer, bt_int buffer_len,
bt_int* offset);
```

Description

This is function _bt_l2cap_eretr_pack_config_request.

_bt_l2cap_eretr_recv Function

File

l2cap_eretr.h

C

```
void _bt_l2cap_eretr_recv(bt_l2cap_mgr_p pmgr, bt_l2cap_channel_t* pch, bt_byte* pdata, bt_int len);
```

Description

This is function _bt_l2cap_eretr_recv.

_bt_l2cap_eretr_send_data Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_eretr_send_data(bt_l2cap_channel_t* pch, bt_byte* data, bt_int len,
bt_l2cap_send_data_callback_fp cb, void* cb_param);
```

Description

This is function _bt_l2cap_eretr_send_data.

_bt_l2cap_eretr_send_pending_frames Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_eretr_send_pending_frames(bt_l2cap_channel_t* pch, bt_byte fbit);
```

Description

This is function _bt_l2cap_eretr_send_pending_frames.

_bt_l2cap_eretr_send_smart_data Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_eretr_send_smart_data(bt_l2cap_channel_t* pch, bt_packet_t* packet, bt_int len,
bt_l2cap_send_data_callback_fp cb, void* cb_param);
```

Description

This is function _bt_l2cap_eretr_send_smart_data.

_bt_l2cap_recv_req_seq_and_fbit Function

File

l2cap_eretr.h

C

```
void _bt_l2cap_recv_req_seq_and_fbit(bt_l2cap_mgr_p pmgr, bt_l2cap_channel_t* pch, bt_byte req_seq, bt_byte fbit);
```

Description

This is function _bt_l2cap_recv_req_seq_and_fbit.

_bt_l2cap_send_ack Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_send_ack(bt_l2cap_channel_t* pch, bt_byte pbit, bt_byte fbit);
```

Description

This is function _bt_l2cap_send_ack.

_bt_l2cap_send_i_or_rr_or_rnr Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_send_i_or_rr_or_rnr(bt_l2cap_channel_t* pch, bt_byte pbit, bt_byte fbit);
```

Description

This is function _bt_l2cap_send_i_or_rr_or_rnr.

_bt_l2cap_send_rej Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_send_rej(bt_l2cap_channel_t* pch, bt_byte pbit, bt_byte fbit);
```

Description

This is function _bt_l2cap_send_rej.

_bt_l2cap_send_rnr Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_send_rnr(bt_l2cap_channel_t* pch, bt_byte pbit, bt_byte fbit);
```

Description

This is function _bt_l2cap_send_rnr.

_bt_l2cap_send_rr Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_send_rr(bt_l2cap_channel_t* pch, bt_byte pbit, bt_byte fbit);
```

Description

This is function _bt_l2cap_send_rr.

_bt_l2cap_send_rr_or_rnr Function

File

l2cap_eretr.h

C

```
bt_bool _bt_l2cap_send_rr_or_rnr(bt_l2cap_channel_t* pch, bt_byte pbit, bt_byte fbit);
```

Description

This is function _bt_l2cap_send_rr_or_rnr.

bt_l2cap_disconnect_ex Function

File

l2cap.h

C

```
bt_bool bt_l2cap_disconnect_ex(bt_l2cap_channel_t* pch, bt_bool force_hci_disconnect);
```

Description

This is function bt_l2cap_disconnect_ex.

bt_l2cap_free_channel Function

File

chmanager.h

C

```
void bt_l2cap_free_channel(bt_l2cap_channel_t* ch);
```

Description

This is function bt_l2cap_free_channel.

bt_l2cap_hci_has_open_channels Function

File

l2cap.h

C

```
bt_bool bt_l2cap_hci_has_open_channels(bt_hci_conn_state_t* conn);
```

Description

This is function bt_l2cap_hci_has_open_channels.

bt_l2cap_is_channel_open Function

File

l2cap.h

C

```
bt_bool bt_l2cap_is_channel_open(bt_l2cap_channel_t* channel);
```

Description

This is function bt_l2cap_is_channel_open.

bt_l2cap_send_config Function

File

l2cap.h

C

```
bt_bool bt_l2cap_send_config(bt_l2cap_mgr_p pmgr, bt_l2cap_channel_t* pch);
```

Description

This is function bt_l2cap_send_config.

bt_l2cap_send_smart_data Function

File

channel.h

C

```
bt_bool bt_l2cap_send_smart_data(bt_l2cap_channel_t* pch, bt_packet_t* packet, bt_int len,
bt_l2cap_send_data_callback_fp cb, void* cb_param);
```

Description

This is function bt_l2cap_send_smart_data.

_bt_l2cap_process_connect_signal Function

File

l2cap_private.h

C

```
void _bt_l2cap_process_connect_signal(bt_l2cap_mgr_t* mgr);
```

Description

This is function _bt_l2cap_process_connect_signal.

L2CAP Data Types and Constants

bt_l2cap_connect Macro

File

l2cap.h

C

```
#define bt_l2cap_connect(mgr, remote_addr, psm, acl_config, connect_cb, param, state_cb)
bt_l2cap_connect_ext(mgr, remote_addr, psm, CMODE_BASIC, acl_config, connect_cb, param, state_cb)
```

Description

This is macro bt_l2cap_connect.

bt_l2cap_listen Macro

File

l2cap.h

C

```
#define bt_l2cap_listen(mgr, psm, acl_config, callback, param) bt_l2cap_listen_ext(mgr, psm, CMODE_BASIC,
acl_config, callback, param)
```

Description

This is macro bt_l2cap_listen.

bt_l2cap_test_enable_local_config Macro

File

l2cap_test.h

C

```
#define bt_l2cap_test_enable_local_config(enable)
```

Description

This is macro bt_l2cap_test_enable_local_config.

bt_l2cap_test_enable_remote_config Macro

File

l2cap_test.h

C

```
#define bt_l2cap_test_enable_remote_config(enable)
```

Description

This is macro bt_l2cap_test_enable_remote_config.

CHANNEL_SIGNAL_CMD_DISCONNECT_FIXED Macro

File

l2cap_private.h

C

```
#define CHANNEL_SIGNAL_CMD_DISCONNECT_FIXED 0
```

Description

This is macro CHANNEL_SIGNAL_CMD_DISCONNECT_FIXED.

CID_ATT Macro

File

channel.h

C

```
#define CID_ATT 0x0004
```

Description

This is macro CID_ATT.

CID_LE_SIG Macro

File

channel.h

C

```
#define CID_LE_SIG 0x0005
```

Description

This is macro CID_LE_SIG.

CID_MAX Macro

File

channel.h

C

```
#define CID_MAX 0xffff
```

Description

This is macro CID_MAX.

CID_MAX_FIXED Macro

File

channel.h

C

```
#define CID_MAX_FIXED 0x003f
```

Description

This is macro CID_MAX_FIXED.

CID_NULL Macro

File

channel.h

C

```
#define CID_NULL 0x0000
```

Description

This is macro CID_NULL.

CID_RECV Macro

File

channel.h

C

```
#define CID_RECV 0x0002
```

Description

This is macro CID_RECV.

CID_SIG Macro

File

channel.h

C

```
#define CID_SIG 0x0001
```

Description

This is macro CID_SIG.

CID_SM Macro

File

channel.h

C

```
#define CID_SM 0x0006
```

Description

This is macro CID_SM.

CMODE_BASIC Macro

File

channel.h

C

```
#define CMODE_BASIC 0
```

Description

This is macro CMODE_BASIC.

CMODE_ERETR Macro

File

channel.h

C

```
#define CMODE_ERETR 3
```

Description

This is macro CMODE_ERETR.

CMODE_FLOW Macro

File

channel.h

C

```
#define CMODE_FLOW 2
```

Description

This is macro CMODE_FLOW.

CMODE_RETR Macro

File

channel.h

C

```
#define CMODE_RETR 1
```

Description

This is macro CMODE_RETR.

CMODE_STRM Macro

File

channel.h

C

```
#define CMODE_STRM 4
```

Description

This is macro CMODE_STRM.

CSTATE_CLOSED Macro

File

channel.h

C

```
#define CSTATE_CLOSED 0x01
```

Description

This is macro CSTATE_CLOSED.

CSTATE_FREE Macro

File

channel.h

C

```
#define CSTATE_FREE 0x00
```

Description

This is macro CSTATE_FREE.

CSTATE_OPEN Macro

File

channel.h

C

```
#define CSTATE_OPEN 0x40
```

Description

This is macro CSTATE_OPEN.

CSTATE_WAIT_CONFIG Macro

File

channel.h

C

```
#define CSTATE_WAIT_CONFIG (CSTATE_WAIT_CONFIG_RSP | CSTATE_WAIT_CONFIG_REQ)
```

Description

This is macro CSTATE_WAIT_CONFIG.

CSTATE_WAIT_CONFIG_REQ Macro

File

channel.h

C

```
#define CSTATE_WAIT_CONFIG_REQ 0x10
```

Description

This is macro CSTATE_WAIT_CONFIG_REQ.

CSTATE_WAIT_CONFIG_RSP Macro

File

channel.h

C

```
#define CSTATE_WAIT_CONFIG_RSP 0x08
```

Description

This is macro CSTATE_WAIT_CONFIG_RSP.

CSTATE_WAIT_CONNECT Macro

File

channel.h

C

```
#define CSTATE_WAIT_CONNECT 0x02
```

Description

This is macro CSTATE_WAIT_CONNECT.

CSTATE_WAIT_CONNECT_RSP Macro

File

channel.h

C

```
#define CSTATE_WAIT_CONNECT_RSP 0x04
```

Description

This is macro CSTATE_WAIT_CONNECT_RSP.

CSTATE_WAIT_DISCONNECT Macro

File

channel.h

C

```
#define CSTATE_WAIT_DISCONNECT 0x20
```

Description

This is macro CSTATE_WAIT_DISCONNECT.

CTYPE_CL Macro

File

channel.h

C

```
#define CTYPE_CL 1 // connectionless bt_l2cap_channel
```

Description

connectionless bt_l2cap_channel

CTYPE_CO Macro

File

channel.h

C

```
#define CTYPE_CO 0 // connection-oriented bt_l2cap_channel
```

Description

connection-oriented bt_l2cap_channel

GET_F_BIT Macro

File

l2cap_eretr.h

C

```
#define GET_F_BIT(ctl) ((ctl & 0x80) >> 7)
```

Description

This is macro GET_F_BIT.

GET_FRAME_TYPE Macro

File

l2cap_eretr.h

C

```
#define GET_FRAME_TYPE(ctl) (ctl & 1)
```

Description

This is macro GET_FRAME_TYPE.

GET_P_BIT Macro

File

l2cap_eretr.h

C

```
#define GET_P_BIT(ctl) ((ctl & 0x10) >> 4)
```

Description

This is macro GET_P_BIT.

GET_REQ_SEQ Macro

File

l2cap_eretr.h

C

```
#define GET_REQ_SEQ(ctl) ((ctl & 0x3F00) >> 8)
```

Description

This is macro GET_REQ_SEQ.

GET_S_FUNCTION Macro

File

l2cap_eretr.h

C

```
#define GET_S_FUNCTION(ctl) ((ctl & 0x0C) >> 2)
```

Description

This is macro GET_S_FUNCTION.

GET_SAR Macro

File

l2cap_eretr.h

C

```
#define GET_SAR(ctl) ((ctl & 0xC000) >> 14)
```

Description

This is macro GET_SAR.

GET_TX_SEQ Macro

File

l2cap_eretr.h

C

```
#define GET_TX_SEQ(ctl) ((ctl & 0x7E) >> 1)
```

Description

This is macro GET_TX_SEQ.

L2CAP_CHANNEL_FLAG_INCOMING Macro

File

channel.h

C

```
#define L2CAP_CHANNEL_FLAG_INCOMING 0x02
```

Description

This is macro L2CAP_CHANNEL_FLAG_INCOMING.

L2CAP_CHANNEL_FLAG_SENDING Macro

File

channel.h

C

```
#define L2CAP_CHANNEL_FLAG_SENDING 0x01
```

Description

This is macro L2CAP_CHANNEL_FLAG_SENDING.

L2CAP_CMD_CONFIG_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_CONFIG_REQUEST 0x04
```

Description

This is macro L2CAP_CMD_CONFIG_REQUEST.

L2CAP_CMD_CONFIG_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_CONFIG_RESPONSE 0x05
```

Description

This is macro L2CAP_CMD_CONFIG_RESPONSE.

L2CAP_CMD_CONN_PARAM_UPDATE_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_CONN_PARAM_UPDATE_REQUEST 0x12
```

Description

This is macro L2CAP_CMD_CONN_PARAM_UPDATE_REQUEST.

L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE 0x13
```

Description

This is macro L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE.

L2CAP_CMD_CONN_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_CONN_REQUEST 0x02
```

Description

This is macro L2CAP_CMD_CONN_REQUEST.

L2CAP_CMD_CONN_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_CONN_RESPONSE 0x03
```

Description

This is macro L2CAP_CMD_CONN_RESPONSE.

L2CAP_CMD_DATA_LEN_CMD_REJECT Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_CMD_REJECT 2
```

Description

This is macro L2CAP_CMD_DATA_LEN_CMD_REJECT.

L2CAP_CMD_DATA_LEN_CONFIG_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_CONFIG_REQUEST 4
```

Description

This is macro L2CAP_CMD_DATA_LEN_CONFIG_REQUEST.

L2CAP_CMD_DATA_LEN_CONFIG_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_CONFIG_RESPONSE 6
```

Description

This is macro L2CAP_CMD_DATA_LEN_CONFIG_RESPONSE.

L2CAP_CMD_DATA_LEN_CONN_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_CONN_REQUEST 4
```

Description

This is macro L2CAP_CMD_DATA_LEN_CONN_REQUEST.

L2CAP_CMD_DATA_LEN_CONN_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_CONN_RESPONSE 8
```

Description

This is macro L2CAP_CMD_DATA_LEN_CONN_RESPONSE.

L2CAP_CMD_DATA_LEN_DCONN_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_DCONN_REQUEST 4
```

Description

This is macro L2CAP_CMD_DATA_LEN_DCONN_REQUEST.

L2CAP_CMD_DATA_LEN_DCONN_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_DCONN_RESPONSE 4
```

Description

This is macro L2CAP_CMD_DATA_LEN_DCONN_RESPONSE.

L2CAP_CMD_DATA_LEN_ECHO_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_ECHO_REQUEST 0
```

Description

This is macro L2CAP_CMD_DATA_LEN_ECHO_REQUEST.

L2CAP_CMD_DATA_LEN_ECHO_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_ECHO_RESPONSE 0
```

Description

This is macro L2CAP_CMD_DATA_LEN_ECHO_RESPONSE.

L2CAP_CMD_DATA_LEN_INFO_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_INFO_REQUEST 2
```

Description

This is macro L2CAP_CMD_DATA_LEN_INFO_REQUEST.

L2CAP_CMD_DATA_LEN_INFO_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_DATA_LEN_INFO_RESPONSE 4
```

Description

This is macro L2CAP_CMD_DATA_LEN_INFO_RESPONSE.

L2CAP_CMD_DCONN_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_DCONN_REQUEST 0x06
```

Description

This is macro L2CAP_CMD_DCONN_REQUEST.

L2CAP_CMD_DCONN_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_DCONN_RESPONSE 0x07
```

Description

This is macro L2CAP_CMD_DCONN_RESPONSE.

L2CAP_CMD_ECHO_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_ECHO_REQUEST 0x08
```

Description

This is macro L2CAP_CMD_ECHO_REQUEST.

L2CAP_CMD_ECHO_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_ECHO_RESPONSE 0x09
```

Description

This is macro L2CAP_CMD_ECHO_RESPONSE.

L2CAP_CMD_HEADER_LEN Macro

File

command.h

C

```
#define L2CAP_CMD_HEADER_LEN 4
```

Description

This is macro L2CAP_CMD_HEADER_LEN.

L2CAP_CMD_INFO_REQUEST Macro

File

command.h

C

```
#define L2CAP_CMD_INFO_REQUEST 0x0A
```

Description

This is macro L2CAP_CMD_INFO_REQUEST.

L2CAP_CMD_INFO_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_INFO_RESPONSE 0x0B
```

Description

This is macro L2CAP_CMD_INFO_RESPONSE.

L2CAP_CMD_LAST Macro

File

command.h

C

```
#define L2CAP_CMD_LAST L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE
```

Description

This is macro L2CAP_CMD_LAST.

L2CAP_CMD_REJECT Macro

File

command.h

C

```
#define L2CAP_CMD_REJECT 0x01
```

Description

This is macro L2CAP_CMD_REJECT.

L2CAP_CMD_RESERVED Macro

File

command.h

C

```
#define L2CAP_CMD_RESERVED 0x00
```

Description

This is macro L2CAP_CMD_RESERVED.

L2CAP_CMD_STATUS_BEING_SENT Macro

File

command.h

C

```
#define L2CAP_CMD_STATUS_BEING_SENT 2
```

Description

This is macro L2CAP_CMD_STATUS_BEING_SENT.

L2CAP_CMD_STATUS_PENDING Macro

File

command.h

C

```
#define L2CAP_CMD_STATUS_PENDING 0
```

Description

This is macro L2CAP_CMD_STATUS_PENDING.

L2CAP_CMD_STATUS_WAITING_RESPONSE Macro

File

command.h

C

```
#define L2CAP_CMD_STATUS_WAITING_RESPONSE 1
```

Description

This is macro L2CAP_CMD_STATUS_WAITING_RESPONSE.

L2CAP_CONFIG_RESULT_REJECTED Macro

File

command.h

C

```
#define L2CAP_CONFIG_RESULT_REJECTED 0x0002
```

Description

This is macro L2CAP_CONFIG_RESULT_REJECTED.

L2CAP_CONFIG_RESULT_SUCCESS Macro

File

command.h

C

```
#define L2CAP_CONFIG_RESULT_SUCCESS 0x0000
```

Description

This is macro L2CAP_CONFIG_RESULT_SUCCESS.

L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER Macro

File

command.h

C

```
#define L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER 0x0001
```

Description

This is macro L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER.

L2CAP_CONFIG_RESULT_UNKNOWN_OPTION Macro

File

command.h

C

```
#define L2CAP_CONFIG_RESULT_UNKNOWN_OPTION 0x0003
```

Description

This is macro L2CAP_CONFIG_RESULT_UNKNOWN_OPTION.

L2CAP_CONN_REQ_RESULT_INVALID_PSM Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_INVALID_PSM 0x0002
```

Description

This is macro L2CAP_CONN_REQ_RESULT_INVALID_PSM.

L2CAP_CONN_REQ_RESULT_NO_RESOURCES Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_NO_RESOURCES 0x0004
```

Description

This is macro L2CAP_CONN_REQ_RESULT_NO_RESOURCES.

L2CAP_CONN_REQ_RESULT_PENDING Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_PENDING 0x0001
```

Description

This is macro L2CAP_CONN_REQ_RESULT_PENDING.

L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK 0x0003
```

Description

This is macro L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK.

L2CAP_CONN_REQ_RESULT_SUCCESS Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_SUCCESS 0x0000
```

Description

This is macro L2CAP_CONN_REQ_RESULT_SUCCESS.

L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING 0x0001
```

Description

This is macro L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING.

L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING 0x0002
```

Description

This is macro L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING.

L2CAP_CONN_REQ_STATUS_NO_INFO Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_STATUS_NO_INFO 0x0000
```

Description

This is macro L2CAP_CONN_REQ_STATUS_NO_INFO.

L2CAP_DEFAULT_ERTX Macro

File

l2cap.h

C

```
#define L2CAP_DEFAULT_ERTX 60 /* seconds */
```

Description

seconds

L2CAP_DEFAULT_RTX Macro

File

l2cap.h

C

```
#define L2CAP_DEFAULT_RTX 3 /* seconds */
```

Description

seconds

L2CAP_ECHO_MAX_DATA_LEN Macro

File

command.h

C

```
#define L2CAP_ECHO_MAX_DATA_LEN 20
```

Description

echo request and response

L2CAP_ERETR_RECV_STATE_RECV Macro

File

channel.h

C

```
#define L2CAP_ERETR_RECV_STATE_RECV 0
```

Description

This is macro L2CAP_ERETR_RECV_STATE_RECV.

L2CAP_ERETR_RECV_STATE_REJ_SENT Macro

File

channel.h

C

```
#define L2CAP_ERETR_RECV_STATE_REJ_SENT 1
```

Description

This is macro L2CAP_ERETR_RECV_STATE_REJ_SENT.

L2CAP_ERETR_RECV_STATE_SREJ_SENT Macro

File

channel.h

C

```
#define L2CAP_ERETR_RECV_STATE_SREJ_SENT 2
```

Description

This is macro L2CAP_ERETR_RECV_STATE_SREJ_SENT.

L2CAP_ERETR_XMIT_STATE_WAIT_ACK Macro

File

channel.h

C

```
#define L2CAP_ERETR_XMIT_STATE_WAIT_ACK 1
```

Description

This is macro L2CAP_ERETR_XMIT_STATE_WAIT_ACK.

L2CAP_ERETR_XMIT_STATE_WAIT_F Macro

File

channel.h

C

```
#define L2CAP_ERETR_XMIT_STATE_WAIT_F 2
```

Description

This is macro L2CAP_ERETR_XMIT_STATE_WAIT_F.

L2CAP_ERETR_XMIT_STATE_XMIT Macro

File

channel.h

C

```
#define L2CAP_ERETR_XMIT_STATE_XMIT 0
```

Description

This is macro L2CAP_ERETR_XMIT_STATE_XMIT.

L2CAP_EXT_BI_QOS Macro

File

command.h

C

```
#define L2CAP_EXT_BI_QOS 0x00000004
```

Description

This is macro L2CAP_EXT_BI_QOS.

L2CAP_EXT_ENHANCED_RETRANSMISSION Macro

File

command.h

C

```
#define L2CAP_EXT_ENHANCED_RETRANSMISSION 0x00000008
```

Description

This is macro L2CAP_EXT_ENHANCED_RETRANSMISSION.

L2CAP_EXT_FCS_OPTION Macro

File

command.h

C

```
#define L2CAP_EXT_FCS_OPTION 0x00000020
```

Description

This is macro L2CAP_EXT_FCS_OPTION.

L2CAP_EXT_FLOW_CONTROL Macro

File

command.h

C

```
#define L2CAP_EXT_FLOW_CONTROL 0x00000001
```

Description

This is macro L2CAP_EXT_FLOW_CONTROL.

L2CAP_EXT_RETRANSMISSION Macro

File

command.h

C

```
#define L2CAP_EXT_RETRANSMISSION 0x00000002
```

Description

This is macro L2CAP_EXT_RETRANSMISSION.

L2CAP_EXT_STREAMING Macro

File

command.h

C

```
#define L2CAP_EXT_STREAMING 0x00000010
```

Description

This is macro L2CAP_EXT_STREAMING.

L2CAP_FRAME_TYPE_I Macro

File

l2cap.h

C

```
#define L2CAP_FRAME_TYPE_I 0x0
```

Description

This is macro L2CAP_FRAME_TYPE_I.

L2CAP_FRAME_TYPE_S Macro

File

l2cap.h

C

```
#define L2CAP_FRAME_TYPE_S 0x1
```

Description

This is macro L2CAP_FRAME_TYPE_S.

L2CAP_HCI_CONNECT_PACKET_TYPE Macro

File

l2cap.h

C

```
#define L2CAP_HCI_CONNECT_PACKET_TYPE _l2cap_hci_connect_packet_type
```

Description

This is macro L2CAP_HCI_CONNECT_PACKET_TYPE.

L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE Macro

File

l2cap.h

C

```
#define L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE _l2cap_hci_page_scan_repetition_mode
```

Description

This is macro L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE.

L2CAP_HCI_CONNECT_ROLE_SWITCH Macro

File

l2cap.h

C

```
#define L2CAP_HCI_CONNECT_ROLE_SWITCH _l2cap_hci_role_switch
```

Description

This is macro L2CAP_HCI_CONNECT_ROLE_SWITCH.

L2CAP_HEADER_LEN Macro

File

l2cap.h

C

```
#define L2CAP_HEADER_LEN 4
```

Description

This is macro L2CAP_HEADER_LEN.

L2CAP_IDLE_HCI_CONNECTION_TIMEOUT Macro

File

l2cap.h

C

```
#define L2CAP_IDLE_HCI_CONNECTION_TIMEOUT _l2cap_idle_hci_connection_timeout
```

Description

This is macro L2CAP_IDLE_HCI_CONNECTION_TIMEOUT.

L2CAP_INFO_NOT_SUPPORTED Macro

File

command.h

C

```
#define L2CAP_INFO_NOT_SUPPORTED 0x0000     // REDFLAG: should it be non zero?
```

Description

REDFLAG: should it be non zero?

L2CAP_INFO_RESULT_SUCCESS Macro

File

command.h

C

```
#define L2CAP_INFO_RESULT_SUCCESS 0x0000
```

Description

This is macro L2CAP_INFO_RESULT_SUCCESS.

L2CAP_INFO_TYPE_CONNECTIONLESS_MTU Macro

File

command.h

C

```
#define L2CAP_INFO_TYPE_CONNECTIONLESS_MTU 0x0001
```

Description

information request

L2CAP_INFO_TYPE_EXTENDED_SUPPORT Macro

File

command.h

C

```
#define L2CAP_INFO_TYPE_EXTENDED_SUPPORT 0x0002
```

Description

This is macro L2CAP_INFO_TYPE_EXTENDED_SUPPORT.

L2CAP_LINK_TYPE_BD_EDR Macro

File

l2cap_fixed_channel.h

C

```
#define L2CAP_LINK_TYPE_BD_EDR HCI_LINK_TYPE_BD_EDR
```

Description

This is macro L2CAP_LINK_TYPE_BD_EDR.

L2CAP_LINK_TYPE_LE Macro

File

l2cap_fixed_channel.h

C

```
#define L2CAP_LINK_TYPE_LE HCI_LINK_TYPE_LE
```

Description

This is macro L2CAP_LINK_TYPE_LE.

L2CAP_MAX_CMD_QUEUE_LEN Macro

File

l2cap_command_queue.h

C

```
#define L2CAP_MAX_CMD_QUEUE_LEN 5
```

Description

This is macro L2CAP_MAX_CMD_QUEUE_LEN.

L2CAP_MAX_FRAME_BUFFERS Macro

File

frame_buffer.h

C

```
#define L2CAP_MAX_FRAME_BUFFERS 2
```

Description

This is macro L2CAP_MAX_FRAME_BUFFERS.

L2CAP_MAX_MANAGERS Macro

File

l2cap.h

C

```
#define L2CAP_MAX_MANAGERS 1
```

Description

This is macro L2CAP_MAX_MANAGERS.

L2CAP_MAX_MTU Macro

File

l2cap.h

C

```
#define L2CAP_MAX_MTU (_hci_l2cap_buffer_len - L2CAP_HEADER_LEN)
```

Description

This is macro L2CAP_MAX_MTU.

L2CAP_MAX_OPTIONS Macro

File

command.h

C

```
#define L2CAP_MAX_OPTIONS 4
```

Description

configuration request/response

L2CAP_MAX_RTX Macro

File

l2cap.h

C

```
#define L2CAP_MAX_RTX 4
```

Description

This is macro L2CAP_MAX_RTX.

L2CAP_MGR_STATE_FREE Macro

File

l2cap.h

C

```
#define L2CAP_MGR_STATE_FREE 0
```

Description

This is macro L2CAP_MGR_STATE_FREE.

L2CAP_MGR_STATE_LISTENING_L2CAP Macro

File

l2cap.h

C

```
#define L2CAP_MGR_STATE_LISTENING_L2CAP 2
```

Description

This is macro L2CAP_MGR_STATE_LISTENING_L2CAP.

L2CAP_MGR_STATE_USED Macro

File

l2cap.h

C

```
#define L2CAP_MGR_STATE_USED 1
```

Description

This is macro L2CAP_MGR_STATE_USED.

L2CAP_MIN_MTU Macro

File

l2cap.h

C

```
#define L2CAP_MIN_MTU 48
```

Description

This is macro L2CAP_MIN_MTU.

L2CAP_MONITOR_TIMEOUT Macro

File

l2cap.h

C

```
#define L2CAP_MONITOR_TIMEOUT ((L2CAP_RFC_MONITOR_TIMEOUT / 1000) - 2)
```

Description

This is macro L2CAP_MONITOR_TIMEOUT.

L2CAP_OPTION_LEN_FLASH_QOS Macro

File

command.h

C

```
#define L2CAP_OPTION_LEN_FLASH_QOS 22
```

Description

This is macro L2CAP_OPTION_LEN_FLASH_QOS.

L2CAP_OPTION_LEN_FLASH_RFC Macro

File

command.h

C

```
#define L2CAP_OPTION_LEN_FLASH_RFC 9
```

Description

This is macro L2CAP_OPTION_LEN_FLASH_RFC.

L2CAP_OPTION_LEN_FLASH_TIMEOUT Macro

File

command.h

C

```
#define L2CAP_OPTION_LEN_FLASH_TIMEOUT 2
```

Description

This is macro L2CAP_OPTION_LEN_FLASH_TIMEOUT.

L2CAP_OPTION_LEN_MAX_MTU Macro

File

command.h

C

```
#define L2CAP_OPTION_LEN_MAX_MTU 2
```

Description

This is macro L2CAP_OPTION_LEN_MAX_MTU.

L2CAP_OPTION_LEN_UNKNOWN Macro

File

command.h

C

```
#define L2CAP_OPTION_LEN_UNKNOWN 22
```

Description

This is macro L2CAP_OPTION_LEN_UNKNOWN.

L2CAP_OPTION_LEN_UNKNOWN_DATA Macro

File

command.h

C

```
#define L2CAP_OPTION_LEN_UNKNOWN_DATA 9//21
```

Description

21

L2CAP_OPTION_TYPE_FLASH_QOS Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_FLASH_QOS 0x03
```

Description

This is macro L2CAP_OPTION_TYPE_FLASH_QOS.

L2CAP_OPTION_TYPE_FLASH_QOS_FLAG Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_FLASH_QOS_FLAG 0x04
```

Description

This is macro L2CAP_OPTION_TYPE_FLASH_QOS_FLAG.

L2CAP_OPTION_TYPE_FLASH_RFC Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_FLASH_RFC 0x04
```

Description

This is macro L2CAP_OPTION_TYPE_FLASH_RFC.

L2CAP_OPTION_TYPE_FLASH_RFC_FLAG Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_FLASH_RFC_FLAG 0x08
```

Description

This is macro L2CAP_OPTION_TYPE_FLASH_RFC_FLAG.

L2CAP_OPTION_TYPE_FLASH_TIMEOUT Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_FLASH_TIMEOUT 0x02
```

Description

This is macro L2CAP_OPTION_TYPE_FLASH_TIMEOUT.

L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG 0x02
```

Description

This is macro L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG.

L2CAP_OPTION_TYPE_MAX_MTU Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_MAX_MTU 0x01
```

Description

This is macro L2CAP_OPTION_TYPE_MAX_MTU.

L2CAP_OPTION_TYPE_MAX_MTU_FLAG Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_MAX_MTU_FLAG 0x01
```

Description

This is macro L2CAP_OPTION_TYPE_MAX_MTU_FLAG.

L2CAP_OPTION_TYPE_UNKNOWN_FLAG Macro

File

command.h

C

```
#define L2CAP_OPTION_TYPE_UNKNOWN_FLAG 0x10
```

Description

This is macro L2CAP_OPTION_TYPE_UNKNOWN_FLAG.

L2CAP_PACKET_DATA_TYPE_RAW Macro

File

l2cap_packet.h

C

```
#define L2CAP_PACKET_DATA_TYPE_RAW 0
```

Description

This is macro L2CAP_PACKET_DATA_TYPE_RAW.

L2CAP_PACKET_DATA_TYPE_SMART Macro

File

l2cap_packet.h

C

```
#define L2CAP_PACKET_DATA_TYPE_SMART 1
```

Description

This is macro L2CAP_PACKET_DATA_TYPE_SMART.

L2CAP_REJECT_REASON_INVALID_CHANNEL Macro

File

command.h

C

```
#define L2CAP_REJECT_REASON_INVALID_CHANNEL 0x0002
```

Description

This is macro L2CAP_REJECT_REASON_INVALID_CHANNEL.

L2CAP_REJECT_REASON_MTU_EXCEEDED Macro

File

command.h

C

```
#define L2CAP_REJECT_REASON_MTU_EXCEEDED 0x0001
```

Description

This is macro L2CAP_REJECT_REASON_MTU_EXCEEDED.

L2CAP_REJECT_REASON_NOT_UNDERSTOOD Macro

File

command.h

C

```
#define L2CAP_REJECT_REASON_NOT_UNDERSTOOD 0x0000
```

Description

This is macro L2CAP_REJECT_REASON_NOT_UNDERSTOOD.

L2CAP_RETR_TIMEOUT Macro

File

l2cap.h

C

```
#define L2CAP_RETR_TIMEOUT (L2CAP_RFC_ERETR_TIMEOUT / 1000)
```

Description

This is macro L2CAP_RETR_TIMEOUT.

L2CAP_RFC_BASIC Macro

File

command.h

C

```
#define L2CAP_RFC_BASIC 0x00
```

Description

This is macro L2CAP_RFC_BASIC.

L2CAP_RFC_ENHANCED_RETRANSMISSION Macro

File

command.h

C

```
#define L2CAP_RFC_ENHANCED_RETRANSMISSION 0x03
```

Description

This is macro L2CAP_RFC_ENHANCED_RETRANSMISSION.

L2CAP_RFC_ERETR_TIMEOUT Macro

File

command.h

C

```
#define L2CAP_RFC_ERETR_TIMEOUT (3 * 1000)  // 3 secs
```

Description

3 secs

L2CAP_RFC_ERETR_TX_WINDOW Macro

File

command.h

C

```
#define L2CAP_RFC_ERETR_TX_WINDOW 1
```

Description

This is macro L2CAP_RFC_ERETR_TX_WINDOW.

L2CAP_RFC_FLOW_CONTROL Macro

File

command.h

C

```
#define L2CAP_RFC_FLOW_CONTROL 0x02
```

Description

This is macro L2CAP_RFC_FLOW_CONTROL.

L2CAP_RFC_MONITOR_TIMEOUT Macro

File

command.h

C

```
#define L2CAP_RFC_MONITOR_TIMEOUT (12 * 1000) // 12 secs
```

Description

12 secs

L2CAP_RFC_RETRANSMISSION Macro

File

command.h

C

```
#define L2CAP_RFC_RETRANSMISSION 0x01
```

Description

This is macro L2CAP_RFC_RETRANSMISSION.

L2CAP_RFC_STREAMING Macro

File

command.h

C

```
#define L2CAP_RFC_STREAMING 0x04
```

Description

This is macro L2CAP_RFC_STREAMING.

L2CAP_SAR_SDU_CONTINUE Macro

File

l2cap.h

C

```
#define L2CAP_SAR_SDU_CONTINUE 0x3
```

Description

This is macro L2CAP_SAR_SDU_CONTINUE.

L2CAP_SAR_SDU_START Macro

File

l2cap.h

C

```
#define L2CAP_SAR_SDU_START 0x1
```

Description

This is macro L2CAP_SAR_SDU_START.

L2CAP_SAR_SDU_STOP Macro

File

l2cap.h

C

```
#define L2CAP_SAR_SDU_STOP 0x2
```

Description

This is macro L2CAP_SAR_SDU_STOP.

L2CAP_SAR_UNSEGMENTED Macro

File

l2cap.h

C

```
#define L2CAP_SAR_UNSEGMENTED 0x0
```

Description

This is macro L2CAP_SAR_UNSEGMENTED.

L2CAP_SERVICE_TYPE_BEST_EFFORT Macro

File

command.h

C

```
#define L2CAP_SERVICE_TYPE_BEST_EFFORT 0x01
```

Description

This is macro L2CAP_SERVICE_TYPE_BEST_EFFORT.

L2CAP_SERVICE_TYPE_GUARANTEED Macro

File

command.h

C

```
#define L2CAP_SERVICE_TYPE_GUARANTEED 0x02
```

Description

This is macro L2CAP_SERVICE_TYPE_GUARANTEED.

L2CAP_SERVICE_TYPE_NO_TRAFFIC Macro

File

command.h

C

```
#define L2CAP_SERVICE_TYPE_NO_TRAFFIC 0x00
```

Description

This is macro L2CAP_SERVICE_TYPE_NO_TRAFFIC.

L2CAP_SFUNCTION_REJ Macro

File

l2cap.h

C

```
#define L2CAP_SFUNCTION_REJ 0x1
```

Description

This is macro L2CAP_SFUNCTION_REJ.

L2CAP_SFUNCTION_RNR Macro

File

l2cap.h

C

```
#define L2CAP_SFUNCTION_RNR 0x2
```

Description

This is macro L2CAP_SFUNCTION_RNR.

L2CAP_SFUNCTION_RR Macro

File

l2cap.h

C

```
#define L2CAP_SFUNCTION_RR 0x0
```

Description

This is macro L2CAP_SFUNCTION_RR.

L2CAP_SFUNCTION_SREJ Macro

File

l2cap.h

C

```
#define L2CAP_SFUNCTION_SREJ 0x3
```

Description

This is macro L2CAP_SFUNCTION_SREJ.

PSM_ATT Macro

File

l2cap.h

C

```
#define PSM_ATT 0x001F
```

Description

This is macro PSM_ATT.

PSM_AVCTP Macro

File

l2cap.h

C

```
#define PSM_AVCTP 0x0017
```

Description

This is macro PSM_AVCTP.

PSM_AVCTP_Browsing Macro

File

l2cap.h

C

```
#define PSM_AVCTP_Browsing 0x001B
```

Description

This is macro PSM_AVCTP_Browsing.

PSM_AVDTP Macro

File

l2cap.h

C

```
#define PSM_AVDTP 0x0019
```

Description

This is macro PSM_AVDTP.

PSM_BNEP Macro

File

l2cap.h

C

```
#define PSM_BNEP 0x000F
```

Description

This is macro PSM_BNEP.

PSM_HID_Control Macro

File

l2cap.h

C

```
#define PSM_HID_Control 0x0011
```

Description

This is macro PSM_HID_Control.

PSM_HID_Interrupt Macro

File

l2cap.h

C

```
#define PSM_HID_Interrupt 0x0013
```

Description

This is macro PSM_HID_Interrupt.

PSM_RFCOMM Macro

File

l2cap.h

C

```
#define PSM_RFCOMM 0x0003
```

Description

This is macro PSM_RFCOMM.

PSM_SDP Macro

File

l2cap.h

C

```
#define PSM_SDP 0x0001
```

Description

This is macro PSM_SDP.

SET_F_BIT Macro

File

l2cap_eretr.h

C

```
#define SET_F_BIT(ctl, fbit) ctl = ((ctl & ~0x80) | ((fbit & 1) << 7))
```

Description

This is macro SET_F_BIT.

SET_FRAME_TYPE Macro

File

l2cap_eretr.h

C

```
#define SET_FRAME_TYPE(ctl, type) ctl = ((ctl & ~1) | (type & 1))
```

Description

This is macro SET_FRAME_TYPE.

SET_P_BIT Macro

File

l2cap_eretr.h

C

```
#define SET_P_BIT(ctl, pbit) ctl = ((ctl & ~0x10) | ((pbit & 1) << 4))
```

Description

This is macro SET_P_BIT.

SET_REQ_SEQ Macro

File

l2cap_eretr.h

C

```
#define SET_REQ_SEQ(ctl, seq) ctl = ((ctl & ~0x3F00) | ((seq & 0x3F) << 8))
```

Description

This is macro SET_REQ_SEQ.

SET_S_FUNCTION Macro

File

l2cap_eretr.h

C

```
#define SET_S_FUNCTION(ctl, s) ctl = ((ctl & ~0x0C) | ((s & 0x3) << 2))
```

Description

This is macro SET_S_FUNCTION.

SET_SAR Macro

File

l2cap_eretr.h

C

```
#define SET_SAR(ctl, sar) ctl = ((ctl & ~0xC000) | ((seq & 0x3) << 14))
```

Description

This is macro SET_SAR.

SET_TX_SEQ Macro

File

l2cap_eretr.h

C

```
#define SET_TX_SEQ(ctl, seq) ctl = ((ctl & ~0x7E) | ((seq & 0x3F) << 1))
```

Description

This is macro SET_TX_SEQ.

_bt_l2cap_start_monitor_timer Macro

File

l2cap_eretr.h

C

```
#define _bt_l2cap_start_monitor_timer(pch) pch->ext->monitor_timer_start_time = L2CAP_MONITOR_TIMEOUT
```

Description

This is macro _bt_l2cap_start_monitor_timer.

_bt_l2cap_start_monitor_timer_if_not_running Macro

File

l2cap_eretr.h

C

```
#define _bt_l2cap_start_monitor_timer_if_not_running(pch) { if (pch->ext->monitor_timer_start_time == 0)
pch->ext->monitor_timer_start_time = L2CAP_MONITOR_TIMEOUT; }
```

Description

This is macro _bt_l2cap_start_monitor_timer_if_not_running.

_bt_l2cap_start_retr_timer Macro

File

l2cap_eretr.h

C

```
#define _bt_l2cap_start_retr_timer(pch) pch->ext->retr_timer_start_time = L2CAP_RETR_TIMEOUT
```

Description

This is macro _bt_l2cap_start_retr_timer.

_bt_l2cap_start_retr_timer_if_not_running Macro

File

l2cap_eretr.h

C

```
#define _bt_l2cap_start_retr_timer_if_not_running(pch) { if (pch->ext->retr_timer_start_time == 0)
pch->ext->retr_timer_start_time = L2CAP_RETR_TIMEOUT; }
```

Description

This is macro _bt_l2cap_start_retr_timer_if_not_running.

_bt_l2cap_stop_monitor_timer Macro

File

l2cap_eretr.h

C

```
#define _bt_l2cap_stop_monitor_timer(pch) pch->ext->monitor_timer_start_time = 0
```

Description

This is macro _bt_l2cap_stop_monitor_timer.

_bt_l2cap_stop_retr_timer Macro

File

l2cap_eretr.h

C

```
#define _bt_l2cap_stop_retr_timer(pch) pch->ext->retr_timer_start_time = 0
```

Description

This is macro _bt_l2cap_stop_retr_timer.

_bt_l2cap_channel_t Structure

File

channel.h

C

```
struct _bt_l2cap_channel_t {
  bt_id cid;
  struct _bt_l2cap_mgr_s * l2cap_mgr;
  bt_byte mode;
  bt_byte type;
  bt_byte state;
  bt_byte flags;
  bt_id cid_destination;
  bt_int psm;
  bt_int cfg_try_cnt;
  bt_l2cap_read_data_callback_fp _read_data_cb;
  bt_l2cap_send_data_callback_fp _send_data_cb;
  void* _send_data_cb_param;
  bt_l2cap_state_changed_callback_fp _state_changed_cb;
  void* _state_changed_param;
  struct _bt_hci_conn_state_s * hci_conn;
  bt_l2cap_packet_t tx_packet;
  bt_uint response_config_options;
  bt_uint response_mtu;
  bt_int response_flash_timeout;
  bt_l2cap_option_rfc_t response_option_rfc;
  bt_byte response_option_unknown_type;
  bt_l2cap_option_unknown_t response_option_unknown;
  bt_uint request_config_options;
  bt_int request_mtu;
  bt_int request_flash_timeout;
  bt_l2cap_option_rfc_t request_option_rfc;
  bt_byte connect_cmd_id;
  bt_byte signal_command;
  bt_signal_t signal;
  bt_l2cap_channel_ext_t* ext;
};
```

Members

Members	Description
bt_uint response_config_options;	configuration

Description

This is record _bt_l2cap_channel_t.

bt_l2cap_cfg_option_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cfg_option_t {
  bt_byte type;
  union {
    bt_l2cap_option_max_mtu_t max_mtu;
    bt_l2cap_option_flash_timeout_t flash_timeout;
    bt_l2cap_option_rfc_t rfc;
    bt_l2cap_option_unknown_t unknown;
  } opt;
} bt_l2cap_cfg_option_t, * bt_l2cap_cfg_option_p;
```

Members

Members	Description
bt_l2cap_option_rfc_t rfc;	option_qos qos;

Description

This is type bt_l2cap_cfg_option_p.

bt_l2cap_cfg_option_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cfg_option_t {
  bt_byte type;
  union {
    bt_l2cap_option_max_mtu_t max_mtu;
    bt_l2cap_option_flash_timeout_t flash_timeout;
    bt_l2cap_option_rfc_t rfc;
    bt_l2cap_option_unknown_t unknown;
  } opt;
} bt_l2cap_cfg_option_t, * bt_l2cap_cfg_option_p;
```

Members

Members	Description
bt_l2cap_option_rfc_t rfc;	option_qos qos;

Description

This is type bt_l2cap_cfg_option_t.

bt_l2cap_channel_t Type

File

channel.h

C

```
typedef struct _bt_l2cap_channel_t bt_l2cap_channel_t;
```

Description

This is type bt_l2cap_channel_t.

bt_l2cap_cmd_assembler_fp Type

File

l2cap_private.h

C

```
typedef bt_bool (* bt_l2cap_cmd_assembler_fp)(bt_l2cap_cmd_header_t* pcmd, bt_byte* buffer, bt_int
buffer_len, bt_int* poffset);
```

Description

This is type bt_l2cap_cmd_assembler_fp.

bt_l2cap_cmd_config_req_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_config_req_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_int flags;
  bt_int option_count;
```

```
  struct _bt_l2cap_channel_t* channel;
  bt_l2cap_cfg_option_t* options;
} bt_l2cap_cmd_config_req_t, * bt_l2cap_cmd_config_req_p;
```

Description

This is type bt_l2cap_cmd_config_req_p.

bt_l2cap_cmd_config_req_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_config_req_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_int flags;
  bt_int option_count;
  struct _bt_l2cap_channel_t* channel;
  bt_l2cap_cfg_option_t* options;
} bt_l2cap_cmd_config_req_t, * bt_l2cap_cmd_config_req_p;
```

Description

This is type bt_l2cap_cmd_config_req_t.

bt_l2cap_cmd_config_res_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_config_res_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_source;
  bt_int flags;
  bt_int result;
  bt_int option_count;
  struct _bt_l2cap_channel_t* channel;
  bt_l2cap_cfg_option_t* options;
} bt_l2cap_cmd_config_res_t, * bt_l2cap_cmd_config_res_p;
```

Description

This is type bt_l2cap_cmd_config_res_p.

bt_l2cap_cmd_config_res_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_config_res_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_source;
  bt_int flags;
  bt_int result;
  bt_int option_count;
  struct _bt_l2cap_channel_t* channel;
  bt_l2cap_cfg_option_t* options;
} bt_l2cap_cmd_config_res_t, * bt_l2cap_cmd_config_res_p;
```

Description

This is type bt_l2cap_cmd_config_res_t.

bt_l2cap_cmd_conn_param_update_req_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_conn_param_update_req_t {
  bt_l2cap_cmd_header_t header;
  bt_uint min_interval;
  bt_uint max_interval;
  bt_uint slave_latency;
  bt_uint timeout_multiplier;
} bt_l2cap_cmd_conn_param_update_req_t;
```

Description

connection parameter update

bt_l2cap_cmd_conn_param_update_res_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_conn_param_update_res_t {
  bt_l2cap_cmd_header_t header;
  bt_int result;
} bt_l2cap_cmd_conn_param_update_res_t;
```

Description

This is type bt_l2cap_cmd_conn_param_update_res_t.

bt_l2cap_cmd_connection_req_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_connection_req_t {
  bt_l2cap_cmd_header_t header;
  bt_int psm;
  bt_id cid_source;
} bt_l2cap_cmd_connection_req_t, * bt_l2cap_cmd_connection_req_p;
```

Description

This is type bt_l2cap_cmd_connection_req_p.

bt_l2cap_cmd_connection_req_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_connection_req_t {
  bt_l2cap_cmd_header_t header;
  bt_int psm;
  bt_id cid_source;
} bt_l2cap_cmd_connection_req_t, * bt_l2cap_cmd_connection_req_p;
```

Description

This is type bt_l2cap_cmd_connection_req_t.

bt_l2cap_cmd_connection_res_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_connection_res_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_id cid_source;
  bt_int result;
  bt_int status;
} bt_l2cap_cmd_connection_res_t, * bt_l2cap_cmd_connection_res_p;
```

Description

This is type bt_l2cap_cmd_connection_res_p.

bt_l2cap_cmd_connection_res_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_connection_res_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_id cid_source;
  bt_int result;
  bt_int status;
} bt_l2cap_cmd_connection_res_t, * bt_l2cap_cmd_connection_res_p;
```

Description

This is type bt_l2cap_cmd_connection_res_t.

bt_l2cap_cmd_disconnection_req_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_disconnection_req_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_id cid_source;
} bt_l2cap_cmd_disconnection_req_t, * bt_l2cap_cmd_disconnection_req_p;
```

Description

This is type bt_l2cap_cmd_disconnection_req_p.

bt_l2cap_cmd_disconnection_req_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_disconnection_req_t {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_id cid_source;
} bt_l2cap_cmd_disconnection_req_t, * bt_l2cap_cmd_disconnection_req_p;
```

Description

This is type bt_l2cap_cmd_disconnection_req_t.

bt_l2cap_cmd_echo_req_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_echo_req_t {
  bt_l2cap_cmd_header_t header;
  bt_int len;
  bt_byte data[L2CAP_ECHO_MAX_DATA_LEN];
} bt_l2cap_cmd_echo_req_t, * bt_l2cap_cmd_echo_req_p;
```

Description

This is type bt_l2cap_cmd_echo_req_p.

bt_l2cap_cmd_echo_req_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_echo_req_t {
  bt_l2cap_cmd_header_t header;
  bt_int len;
  bt_byte data[L2CAP_ECHO_MAX_DATA_LEN];
} bt_l2cap_cmd_echo_req_t, * bt_l2cap_cmd_echo_req_p;
```

Description

This is type bt_l2cap_cmd_echo_req_t.

bt_l2cap_cmd_echo_res_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_echo_res_t {
  bt_l2cap_cmd_header_t header;
  bt_int len;
  bt_byte data[L2CAP_ECHO_MAX_DATA_LEN];
} bt_l2cap_cmd_echo_res_t, * bt_l2cap_cmd_echo_res_p;
```

Description

This is type bt_l2cap_cmd_echo_res_p.

bt_l2cap_cmd_echo_res_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_echo_res_t {
  bt_l2cap_cmd_header_t header;
  bt_int len;
  bt_byte data[L2CAP_ECHO_MAX_DATA_LEN];
} bt_l2cap_cmd_echo_res_t, * bt_l2cap_cmd_echo_res_p;
```

Description

This is type bt_l2cap_cmd_echo_res_t.

bt_l2cap_cmd_header_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_header_t {
  struct _bt_l2cap_cmd_header_t* next_cmd;
  bt_byte code;
  bt_byte id;
  bt_byte status;
  bt_long send_time;
  bt_int rtx;
  bt_byte rtx_count;
  pf_l2cap_cmd_callback cb;
  struct _bt_hci_conn_state_s * phci_conn;
} bt_l2cap_cmd_header_t, * bt_l2cap_cmd_header_p;
```

Description

This is type bt_l2cap_cmd_header_p.

bt_l2cap_cmd_header_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_header_t {
  struct _bt_l2cap_cmd_header_t* next_cmd;
  bt_byte code;
  bt_byte id;
  bt_byte status;
  bt_long send_time;
  bt_int rtx;
  bt_byte rtx_count;
  pf_l2cap_cmd_callback cb;
  struct _bt_hci_conn_state_s * phci_conn;
} bt_l2cap_cmd_header_t, * bt_l2cap_cmd_header_p;
```

Description

This is type bt_l2cap_cmd_header_t.

bt_l2cap_cmd_info_req_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_info_req_t {
  bt_l2cap_cmd_header_t header;
  bt_int type;
} bt_l2cap_cmd_info_req_t, * bt_l2cap_cmd_info_req_p;
```

Description

This is type bt_l2cap_cmd_info_req_p.

bt_l2cap_cmd_info_req_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_info_req_t {
  bt_l2cap_cmd_header_t header;
  bt_int type;
} bt_l2cap_cmd_info_req_t, * bt_l2cap_cmd_info_req_p;
```

Description

This is type bt_l2cap_cmd_info_req_t.

bt_l2cap_cmd_info_res_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_info_res_t {
  bt_l2cap_cmd_header_t header;
  bt_int type;
  bt_int result;
  union {
    bt_int mtu;
    bt_long mask;
    bt_long fixed_channles[2];
  } data;
} bt_l2cap_cmd_info_res_t, * bt_l2cap_cmd_info_res_p;
```

Description

This is type bt_l2cap_cmd_info_res_p.

bt_l2cap_cmd_info_res_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_info_res_t {
  bt_l2cap_cmd_header_t header;
  bt_int type;
  bt_int result;
  union {
    bt_int mtu;
    bt_long mask;
    bt_long fixed_channles[2];
  } data;
} bt_l2cap_cmd_info_res_t, * bt_l2cap_cmd_info_res_p;
```

Description

This is type bt_l2cap_cmd_info_res_t.

bt_l2cap_cmd_parser_fp Type

File

l2cap_private.h

C

```
typedef bt_l2cap_cmd_header_t* (* bt_l2cap_cmd_parser_fp)(bt_byte* pdata, bt_int len, bt_int* poffset);
```

Description

This is type bt_l2cap_cmd_parser_fp.

bt_l2cap_cmd_reject_p Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_reject_t {
  bt_l2cap_cmd_header_t header;
  bt_uint reason;
  union {
    bt_int actual_mtu;
    struct {
      bt_id cid_local;
      bt_id cid_remote;
    } cid;
  } data;
} bt_l2cap_cmd_reject_t, * bt_l2cap_cmd_reject_p;
```

Description

This is type bt_l2cap_cmd_reject_p.

bt_l2cap_cmd_reject_param_t Union

File

command.h

C

```
typedef union _bt_l2cap_cmd_reject_param_t {
  bt_int actual_mtu;
  struct {
    bt_id cid_local;
    bt_id cid_remote;
  } cid;
} bt_l2cap_cmd_reject_param_t;
```

Description

This is type bt_l2cap_cmd_reject_param_t.

bt_l2cap_cmd_reject_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_cmd_reject_t {
  bt_l2cap_cmd_header_t header;
  bt_uint reason;
  union {
    bt_int actual_mtu;
    struct {
      bt_id cid_local;
      bt_id cid_remote;
    } cid;
  } data;
} bt_l2cap_cmd_reject_t, * bt_l2cap_cmd_reject_p;
```

Description

This is type bt_l2cap_cmd_reject_t.

bt_l2cap_command_t Union

File

command.h

C

```
typedef union _bt_l2cap_command_t {
  bt_l2cap_cmd_reject_t reject;
  bt_l2cap_cmd_connection_req_t connection_req;
  bt_l2cap_cmd_connection_res_t connection_res;
  bt_l2cap_cmd_disconnection_req_t disconnection_req;
  cmd_disconnection_res disconnection_res;
  bt_l2cap_cmd_config_req_t config_req;
  bt_l2cap_cmd_config_res_t config_res;
  bt_l2cap_cmd_echo_req_t echo_req;
  bt_l2cap_cmd_echo_res_t echo_res;
  bt_l2cap_cmd_info_req_t info_req;
  bt_l2cap_cmd_info_res_t info_res;
  bt_l2cap_cmd_conn_param_update_req_t conn_update_req;
  bt_l2cap_cmd_conn_param_update_res_t conn_update_res;
} bt_l2cap_command_t;
```

Description

This is type bt_l2cap_command_t.

bt_l2cap_connect_callback_fp Type

File

l2cap.h

C

```
typedef void (* bt_l2cap_connect_callback_fp)(bt_l2cap_channel_t* pch, void* param);
```

Description

This is type bt_l2cap_connect_callback_fp.

bt_l2cap_eretr_xmit_event_e Enumeration

File

l2cap_eretr.h

C

```
typedef enum _bt_l2cap_eretr_xmit_event_e {
  L2CAP_ERETR_EVENT_DATA_REQUEST,
  L2CAP_ERETR_EVENT_LOCAL_BUSY_DETECTED,
  L2CAP_ERETR_EVENT_LOCAL_BUSY_CLEAR,
  L2CAP_ERETR_EVENT_RECV_REQSEQ_AND_FBIT,
  L2CAP_ERETR_EVENT_RECV_FBIT,
  L2CAP_ERETR_EVENT_RETRANSMIT_TIMER_EXPIRED,
  L2CAP_ERETR_EVENT_MONITOR_TIMER_EXPIRED
} bt_l2cap_eretr_xmit_event_e;
```

Description

This is type bt_l2cap_eretr_xmit_event_e.

bt_l2cap_fixed_channel_t Structure

File

l2cap.h

C

```
typedef struct _bt_l2cap_fixed_channel_s {
```

```
  bt_id cid;
  bt_byte link_type;
  bt_l2cap_listen_callback_fp listen_cb;
  bt_l2cap_connect_callback_fp connect_cb;
  void* listen_cb_param;
  void* connect_cb_param;
} bt_l2cap_fixed_channel_t;
```

Description

This is type bt_l2cap_fixed_channel_t.

bt_l2cap_frame_desc_t Structure

File

channel.h

C

```
typedef struct _bt_l2cap_frame_desc_s {
  bt_uint ctl;
  bt_byte data_type;
  union {
    bt_byte* raw;
    bt_packet_t* packet;
  } data;
  bt_int len;
  bt_byte tx_seq;
  bt_byte retry_count;
  bt_uint fcs;
  bt_l2cap_send_data_callback_fp callback;
  void* callback_param;
} bt_l2cap_frame_desc_t;
```

Description

typedef union _bt_l2cap_frame_control_s { struct { bt_byte frame_type:1; bt_byte txSeq:6; bt_byte f:1; bt_byte reqSeq:6; bt_byte sar:2; } iframe; struct { bt_byte frame_type:1; bt_byte reserved:1; bt_byte s:2; bt_byte p:1; bt_byte reserved2:2; bt_byte f:1; bt_byte reqSeq:6; bt_byte reserved3:2; } sframe; } bt_l2cap_frame_control_t;

bt_l2cap_listen_callback_fp Type

File

l2cap.h

C

```
typedef void (* bt_l2cap_listen_callback_fp)(bt_l2cap_channel_t* pch, void* param);
```

Description

This is type bt_l2cap_listen_callback_fp.

bt_l2cap_mgr_p Structure

File

l2cap.h

C

```
typedef struct _bt_l2cap_mgr_s {
  bt_int state;
  bt_l2cap_psm_t* _psms;
  bt_l2cap_fixed_channel_t* _fixed_channels;
  bt_l2cap_channel_t* _channels;
  bt_queue_element_t* cmd_queue;
  bt_queue_element_t* cmd_res_queue;
  bt_byte max_rtx;
  bt_hci_ctrl_state_t* hci_ctrl;
  bt_l2cap_packet_t cmd_tx_packet;
  bt_queue_element_t* connect_queue;
  bt_buffer_mgr_t connect_params_mgr;
```

```
  struct {
    unsigned int connecting : 1;
    unsigned int sig_cmd : 1;
    unsigned int sig_connect : 1;
    unsigned int sending : 1;
  } flags;
  bt_hci_listener_t hci_disconnect_listener;
  bt_byte connect_chmode;
} bt_l2cap_mgr_t, * bt_l2cap_mgr_p;
```

Members

Members	Description
bt_queue_element_t* connect_queue;	bt_int connect_psm; bt_l2cap_connect_callback_fp connect_cb; void* connect_param; bt_l2cap_state_changed_callback_fp connect_state_cb;

Description

This is type bt_l2cap_mgr_p.

bt_l2cap_mgr_t Structure

File

l2cap.h

C

```
typedef struct _bt_l2cap_mgr_s {
  bt_int state;
  bt_l2cap_psm_t* _psms;
  bt_l2cap_fixed_channel_t* _fixed_channels;
  bt_l2cap_channel_t* _channels;
  bt_queue_element_t* cmd_queue;
  bt_queue_element_t* cmd_res_queue;
  bt_byte max_rtx;
  bt_hci_ctrl_state_t* hci_ctrl;
  bt_l2cap_packet_t cmd_tx_packet;
  bt_queue_element_t* connect_queue;
  bt_buffer_mgr_t connect_params_mgr;
  struct {
    unsigned int connecting : 1;
    unsigned int sig_cmd : 1;
    unsigned int sig_connect : 1;
    unsigned int sending : 1;
  } flags;
  bt_hci_listener_t hci_disconnect_listener;
  bt_byte connect_chmode;
} bt_l2cap_mgr_t, * bt_l2cap_mgr_p;
```

Members

Members	Description
bt_queue_element_t* connect_queue;	bt_int connect_psm; bt_l2cap_connect_callback_fp connect_cb; void* connect_param; bt_l2cap_state_changed_callback_fp connect_state_cb;

Description

This is type bt_l2cap_mgr_t.

bt_l2cap_option_flash_timeout_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_option_flash_timeout_t {
  bt_int timeout;
} bt_l2cap_option_flash_timeout_t;
```

Description

This is type bt_l2cap_option_flash_timeout_t.

bt_l2cap_option_max_mtu_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_option_max_mtu_t {
  bt_uint mtu;
} bt_l2cap_option_max_mtu_t;
```

Description

This is type bt_l2cap_option_max_mtu_t.

bt_l2cap_option_qos_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_option_qos_t {
  bt_byte flags;
  bt_byte service_type;
  bt_long token_rate;
  bt_long token_bucket_size;
  bt_long peak_bandwidth;
  bt_long latency;
  bt_long delay_variation;
} bt_l2cap_option_qos_t;
```

Description

This is type bt_l2cap_option_qos_t.

bt_l2cap_option_rfc_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_option_rfc_t {
  bt_byte mode;
  bt_byte tx_window_size;
  bt_byte max_transmit;
  bt_int retr_timeout;
  bt_int monitor_timeout;
  bt_int mps;
} bt_l2cap_option_rfc_t;
```

Description

This is type bt_l2cap_option_rfc_t.

bt_l2cap_option_unknown_t Structure

File

command.h

C

```
typedef struct _bt_l2cap_option_unknown_t {
  bt_byte data_len;
  bt_byte data[L2CAP_OPTION_LEN_UNKNOWN_DATA];
} bt_l2cap_option_unknown_t;
```

Description

This is type bt_l2cap_option_unknown_t.

bt_l2cap_packet_t Structure

File

l2cap_packet.h

C

```
typedef struct _bt_l2cap_packet_t {
  bt_packet_t header;
  union {
    struct _bt_l2cap_channel_t* channel;
    bt_hci_hconn_t hconn;
  } destination;
  bt_byte data_type;
} bt_l2cap_packet_t;
```

Members

Members	Description
bt_byte data_type;	bt_uint fcs; bt_int ctl;

Description

This is type bt_l2cap_packet_t.

bt_l2cap_psm_t Structure

File

l2cap.h

C

```
typedef struct _bt_l2cap_psm_s {
  bt_int psm;
  bt_uint acl_config;
  bt_byte chmode;
  bt_l2cap_listen_callback_fp listen_cb;
  bt_l2cap_connect_callback_fp connect_cb;
  void* listen_cb_param;
  void* connect_cb_param;
} bt_l2cap_psm_t;
```

Description

This is type bt_l2cap_psm_t.

bt_l2cap_read_data_callback_fp Type

File

channel.h

C

```
typedef void (* bt_l2cap_read_data_callback_fp)(bt_l2cap_channel_t* pch, bt_byte_p pdata, bt_int len);
```

Description

This is type bt_l2cap_read_data_callback_fp.

bt_l2cap_request_handler_fp Type

File

l2cap_private.h

C

```
typedef void (* bt_l2cap_request_handler_fp)(bt_l2cap_mgr_p pmgr, bt_hci_conn_state_p pconn,
bt_l2cap_cmd_header_p pcmd);
```

Description

This is type bt_l2cap_request_handler_fp.

bt_l2cap_response_handler_fp Type

File

l2cap_private.h

C

```
typedef void (* bt_l2cap_response_handler_fp)(bt_l2cap_mgr_p pmgr, bt_l2cap_cmd_header_p pcmd);
```

Description

This is type bt_l2cap_response_handler_fp.

bt_l2cap_send_data_callback_fp Type

File

channel.h

C

```
typedef void (* bt_l2cap_send_data_callback_fp)(bt_l2cap_channel_t* pch, bt_byte_p pdata, bt_int len, void*
param);
```

Description

This is type bt_l2cap_send_data_callback_fp.

bt_l2cap_state_changed_callback_fp Type

File

channel.h

C

```
typedef void (* bt_l2cap_state_changed_callback_fp)(bt_l2cap_channel_t* pch, bt_int new_state, void* param);
```

Description

This is type bt_l2cap_state_changed_callback_fp.

bt_l2cap_xmit_event_param_t Structure

File

l2cap_eretr.h

C

```
typedef struct _bt_l2cap_xmit_event_param_t {
  bt_l2cap_eretr_xmit_event_e ev;
  bt_l2cap_mgr_p pmgr;
  bt_l2cap_channel_t* pch;
  union {
    struct {
      bt_byte req_seq;
      bt_byte fbit;
    } recv_reqseq_and_fbit;
    struct {
      bt_byte fbit;
    } recv_bfit;
    struct {
      bt_l2cap_frame_desc_t* frame;
```

```
    } data_request;
  } params;
} bt_l2cap_xmit_event_param_t;
```

Description

This is type bt_l2cap_xmit_event_param_t.

cmd_disconnection_res Structure

File

command.h

C

```
typedef struct _cmd_disconnection_res {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_id cid_source;
} cmd_disconnection_res, * pcmd_disconnection_res;
```

Description

This is type cmd_disconnection_res.

pcmd_disconnection_res Structure

File

command.h

C

```
typedef struct _cmd_disconnection_res {
  bt_l2cap_cmd_header_t header;
  bt_id cid_destination;
  bt_id cid_source;
} cmd_disconnection_res, * pcmd_disconnection_res;
```

Description

This is type pcmd_disconnection_res.

pf_l2cap_cmd_callback Type

File

command.h

C

```
typedef void (* pf_l2cap_cmd_callback)(struct _bt_l2cap_mgr_s *ppmg, struct _bt_l2cap_cmd_header_t *pcmd);
```

Description

This is type pf_l2cap_cmd_callback.

L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT Macro

File

channel.h

C

```
#define L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT 0x04
```

Description

This is macro L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT.

L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR Macro

File

command.h

C

```
#define L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR 0x00000040
```

Description

This is macro L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR.

L2CAP_EXT_EXT_WINDOW_SIZE Macro

File

command.h

C

```
#define L2CAP_EXT_EXT_WINDOW_SIZE 0x00000100
```

Description

This is macro L2CAP_EXT_EXT_WINDOW_SIZE.

L2CAP_EXT_FEATURES_ENABLED Macro

File

l2cap_private.h

C

```
#define L2CAP_EXT_FEATURES_ENABLED (_l2cap_eretr_recv_fp != NULL)
```

Description

This is macro L2CAP_EXT_FEATURES_ENABLED.

L2CAP_EXT_FIXED_CHANNELS Macro

File

command.h

C

```
#define L2CAP_EXT_FIXED_CHANNELS 0x00000080
```

Description

This is macro L2CAP_EXT_FIXED_CHANNELS.

L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT Macro

File

command.h

C

```
#define L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT 0x00000200
```

Description

This is macro L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT.

L2CAP_INFO_TYPE_FIXED_CHANNELS Macro

File

command.h

C

```
#define L2CAP_INFO_TYPE_FIXED_CHANNELS 0x0003
```

Description

This is macro L2CAP_INFO_TYPE_FIXED_CHANNELS.

bt_l2cap_channel_ext_t Type

File

channel.h

C

```
typedef struct _bt_l2cap_channel_ext_t bt_l2cap_channel_ext_t;
```

Description

This is type bt_l2cap_channel_ext_t.

_bt_l2cap_channel_ext_t Structure

File

channel.h

C

```
struct _bt_l2cap_channel_ext_t {
  bt_bool check_fcs;
  bt_byte max_transmit;
  bt_byte next_tx_seq;
  bt_byte expected_ack_seq;
  bt_byte expected_tx_seq;
  bt_bool remote_busy;
  bt_bool local_busy;
  bt_l2cap_frame_desc_t unacked_frame;
  bt_l2cap_frame_desc_t pending_frame;
  bt_l2cap_frame_desc_t s_frame;
  bt_byte retry_count;
  bt_bool rnr_sent;
  bt_bool rej_actioned;
  bt_byte frames_sent;
  bt_ulong retr_timer_start_time;
  bt_ulong monitor_timer_start_time;
  bt_byte recv_state;
  bt_byte xmit_state;
  bt_l2cap_packet_t tx_s_packet;
};
```

Members

Members	Description
bt_byte expected_tx_seq;	bt_byte req_seq;
bt_bool remote_busy;	bt_byte buffer_seq;
bt_byte retry_count;	bt_byte srej_list[RFC_ERETR_TXWINDOW];
bt_byte frames_sent;	bt_bool srej_actioned; bt_byte srej_save_req_seq; bt_bool send_srej; bt_byte buffer_seq_srej;

Description

This is record _bt_l2cap_channel_ext_t.

_bt_l2cap_connect_params_s Structure

File

l2cap.h

C

```
struct _bt_l2cap_connect_params_s {
  bt_l2cap_connect_params_t* next;
  bt_bdaddr_t addr;
  bt_int psm;
  bt_l2cap_connect_callback_fp callback;
  void* param;
  bt_l2cap_state_changed_callback_fp state_cb;
  bt_byte chmode;
  bt_bool fixed_channel;
  bt_uint acl_config;
};
```

Description

This is record _bt_l2cap_connect_params_s.

L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID 0x0006
```

Description

This is macro L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID.

L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED Macro

File

command.h

C

```
#define L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED 0x0007
```

Description

This is macro L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED.

bt_l2cap_connect_params_t Type

File

l2cap.h

C

```
typedef struct _bt_l2cap_connect_params_s bt_l2cap_connect_params_t;
```

Description

This is type bt_l2cap_connect_params_t.

Platform Data Types and Constants

BYTE_SIZE Macro

File

types.h

C

```
#define BYTE_SIZE 1
```

Description

This is macro BYTE_SIZE.

bt_byte Type

File

types.h

C

```
typedef unsigned char bt_byte;
```

Description

This is type bt_byte.

bt_int Type

File

types.h

C

```
typedef short bt_int;
```

Description

This is type bt_int.

bt_long Type

File

types.h

C

```
typedef long bt_long;
```

Description

This is type bt_long.

bt_uint Type

File

types.h

C

```
typedef unsigned short bt_uint;
```

Description

This is type bt_uint.

bt_ulong Type

File

types.h

C

```
typedef unsigned long bt_ulong;
```

Description

This is type bt_ulong.

RFCOMM Functions

bt_rfcomm_allocate_dlc Function

File

rfcomm.h

C

```
bt_rfcomm_dlc_t* bt_rfcomm_allocate_dlc(bt_rfcomm_session_t* session, bt_byte dlci);
```

Description

brief Allocate DLC. ingroup rfcomm

details This function allocates a new DLC on the specified RFCOMM session.

param session The RFCOMM session. param dlci DLCI of the new DLC.

return li A pointer to the new DLC if the function succeeds. li c NULL otherwise.

bt_rfcomm_allocate_session Function

File

rfcomm.h

C

```
bt_rfcomm_session_t* bt_rfcomm_allocate_session(bt_l2cap_mgr_t* l2cap_mgr);
```

Description

brief Allocate RFCOMM session. ingroup rfcomm

details This function allocates a new RFCOMM session.

param l2cap_mgr The L2CAP manager on which the RFCOMM session is to be created. param callback The callback function that is called when session state changes. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter.

return li A pointer to the new RFCOMM session structure if the function succeeds. li c NULL otherwise.

bt_rfcomm_close_dlc Function

File

rfcomm.h

C

```
void bt_rfcomm_close_dlc(bt_rfcomm_dlc_t* dlc);
```

Description

brief Close DLC. ingroup rfcomm

details This function closes a DLC. If DLCI = 0, the parent RFCOMM session is also closed.

bt_rfcomm_connect Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_connect(bt_bdaddr_p remote_addr, bt_byte server_channel, bt_uint acl_config,
bt_rfcomm_dlc_state_callback_fp callback, void* param);
```

Description

brief Connect to a remote device. ingroup rfcomm

details This function establishes an RFCOMM connection with a remote device and opens a data DLC. Changes in the session state are reported through a callback function specified when the session has been allocated via call to bt_rfcomm_allocate_session. Changes in the data DLC are reported through a callback function specified in this call.

param remote_addr Address of the remote device. param server_channel A server channel of the remote RFCOMM server. param callback The callback function for reporting changes in DLC state opened by this call. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_rfcomm_find_dlc Function

File

rfcomm.h

C

```
bt_rfcomm_dlc_t* bt_rfcomm_find_dlc(bt_rfcomm_session_t* session, bt_byte address);
```

Description

brief Find DLC

bt_rfcomm_free_dlc Function

File

rfcomm.h

C

```
void bt_rfcomm_free_dlc(bt_rfcomm_dlc_t* dlc);
```

Description

brief Release DLC. ingroup rfcomm

details This function releases the specified DLC.

param dlc The DLC to be released.

bt_rfcomm_free_session Function

File

rfcomm.h

C

```
void bt_rfcomm_free_session(bt_rfcomm_session_t* session);
```

Description

brief Release RFCOMM session. ingroup rfcomm

details This function deallocates the specified RFCOMM session. This function does not disconnect the session. It just frees the memory used by the bt_rfcomm_session structure. The session has to be disconnected by calling bt_rfcomm_close_dlc with DLCI = 0 first.

param session The RFCOMM session to be deallocated.

bt_rfcomm_init Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_init();
```

Description

brief Initialize the RFCOMM layer. ingroup rfcomm

details This function initializes the RFCOMM layer of the stack. It must be called prior to any other RFCOMM function can be called.

bt_rfcomm_listen Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_listen(bt_byte server_channel, bt_uint acl_config, bt_rfcomm_dlc_state_callback_fp
callback, void* param);
```

Description

brief Listen for incoming connections. ingroup rfcomm

details This function enables incoming connections on the specified RFCOMM session. Changes in the session state are reported through a callback function.

param server_channel A server channel on which the RFCOMM session will listen and accept incoming connections. param callback The callback function that is called when session state changes. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_rfcomm_open_dlc Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_open_dlc(bt_rfcomm_dlc_t* dlc, bt_rfcomm_dlc_state_callback_fp callback, void* param);
```

Description

brief Open DLC. ingroup rfcomm

details This function opens the specified DLC. Before calling this function the RFCOMM session must be already open. This function is not to be used with DLCI = 0. Changes in DLC state are reported through a callback function.

param dlc The DLC to open. param callback The callback function for reporting changes in DLC state. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_rfcomm_register_listener Function

File

rfcomm_mgr.h

C

```
bt_bool bt_rfcomm_register_listener(bt_rfcomm_session_listener_t* listener);
```

Description

This is function bt_rfcomm_register_listener.

bt_rfcomm_send_credit Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_send_credit(bt_rfcomm_dlc_t* dlc, bt_byte credit);
```

Description

This is function bt_rfcomm_send_credit.

bt_rfcomm_send_data Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_send_data(bt_rfcomm_dlc_t* dlc, void * data, bt_int len, bt_rfcomm_send_data_callback_fp
callback);
```

Description

brief Send data over a DLC. ingroup rfcomm

details This function sends data over the specified DLC. Operation completion is reported through callback function.

param dlc The DLC. param data A pointer to the data to be sent. param len Data length. param callback The callback function that is called when operation completes.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_rfcomm_unregister_listener Function

File

rfcomm_mgr.h

C

```
void bt_rfcomm_unregister_listener(bt_rfcomm_session_listener_t* listener);
```

Description

This is function bt_rfcomm_unregister_listener.

check_fcs Function

File

rfcomm_private.h

C

```
bt_bool check_fcs(bt_byte_p buffer, bt_byte len, bt_byte recv_fcs);
```

Description

From rfcomm_fcs.c

read_command Function

File

l2cap_private.h

C

```
bt_l2cap_cmd_header_p read_command(bt_byte_p pdata, bt_int len, bt_int* offset, bt_l2cap_cmd_header_t*
pheader);
```

Description

From channel_cmd_recv.c

rfcomm_cq_ack_cmd Function

File

rfcomm_cmd_queue.h

C

```
struct _bt_rfcomm_command_t* rfcomm_cq_ack_cmd(struct _bt_rfcomm_dlc_t* pdlc, struct _bt_rfcomm_command_t*
pres);
```

Description

This is function rfcomm_cq_ack_cmd.

rfcomm_cq_ack_mx_cmd Function

File

rfcomm_cmd_queue.h

C

```
void rfcomm_cq_ack_mx_cmd(struct _bt_rfcomm_dlc_t* pdlc, struct _bt_rfcomm_command_t* pres);
```

Description

This is function rfcomm_cq_ack_mx_cmd.

rfcomm_send_cmd Function

File

rfcomm_private.h

C

```
bt_bool rfcomm_send_cmd(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd);
```

Description

This is function rfcomm_send_cmd.

rfcomm_send_commands_from_queue Function

File

rfcomm_cmd_queue.h

C

```
void rfcomm_send_commands_from_queue(struct _bt_rfcomm_session_t * psess);
```

Description

This is function rfcomm_send_commands_from_queue.

rfcomm_send_mx_msc_cmd Function

File

rfcomm_mx.h

C

```
bt_bool rfcomm_send_mx_msc_cmd(bt_rfcomm_dlc_p pdlc, bt_rfcomm_cmd_callback_fp cb);
```

Description

This is function rfcomm_send_mx_msc_cmd.

rfcomm_send_mx_pn_cmd Function

File

rfcomm_mx.h

C

```
void rfcomm_send_mx_pn_cmd(bt_rfcomm_dlc_p pdlc, bt_rfcomm_cmd_callback_fp cb);
```

Description

This is function rfcomm_send_mx_pn_cmd.

_rfcomm_alloc_cmd_buffer Function

File

rfcomm_private.h

C

```
bt_rfcomm_command_p _rfcomm_alloc_cmd_buffer();
```

Description

This is function _rfcomm_alloc_cmd_buffer.

_rfcomm_allocate_buffers Function

File

rfcomm_private.h

C

```
void _rfcomm_allocate_buffers();
```

Description

Defined by OEM through library configuration

_rfcomm_allocate_mx_cmd Function

File

rfcomm_mx.h

C

```
bt_rfcomm_command_p _rfcomm_allocate_mx_cmd(bt_rfcomm_session_p psess, bt_rfcomm_cmd_callback_fp cb);
```

Description

This is function _rfcomm_allocate_mx_cmd.

_rfcomm_free_cmd_buffer Function

File

rfcomm_private.h

C

```
void _rfcomm_free_cmd_buffer(void* p);
```

Description

This is function _rfcomm_free_cmd_buffer.

_rfcomm_get_tick_count Function

File

rfcomm_timer.h

C

```
bt_long _rfcomm_get_tick_count();
```

Description

This is function _rfcomm_get_tick_count.

_rfcomm_init_cmd_buffers Function

File

rfcomm_private.h

C

```
bt_bool _rfcomm_init_cmd_buffers();
```

Description

From rfcomm_cmdbuffer.c

_rfcomm_init_mgr Function

File

rfcomm_mgr.h

C

```
void _rfcomm_init_mgr();
```

Description

This is function _rfcomm_init_mgr.

_rfcomm_init_sessions Function

File

rfcomm_private.h

C

```
void _rfcomm_init_sessions();
```

Description

From rfcomm_session.c

_rfcomm_init_timer Function

File

rfcomm_timer.h

C

```
void _rfcomm_init_timer();
```

Description

This is function _rfcomm_init_timer.

_rfcomm_l2cap_read_data_callback Function

File

rfcomm_private.h

C

```
void _rfcomm_l2cap_read_data_callback(struct _bt_l2cap_channel_t * pch, bt_byte_p pdata, bt_int len);
```

Description

From rfcomm_cmd_recv.c

_rfcomm_mx_process_fc Function

File

rfcomm_mx.h

C

```
void _rfcomm_mx_process_fc(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_byte mx_msg_type);
```

Description

This is function _rfcomm_mx_process_fc.

_rfcomm_mx_process_pn Function

File

rfcomm_mx.h

C

```
void _rfcomm_mx_process_pn(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start, bt_int len);
```

Description

This is function _rfcomm_mx_process_pn.

_rfcomm_mx_process_rpn Function

File

rfcomm_mx.h

C

```
void _rfcomm_mx_process_rpn(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start, bt_int len);
```

Description

This is function _rfcomm_mx_process_rpn.

_rfcomm_process_cmd_frame_disc Function

File

rfcomm_private.h

C

```
void _rfcomm_process_cmd_frame_disc(bt_rfcomm_session_p psess, bt_rfcomm_command_p pcmd);
```

Description

From frame_disc.c

_rfcomm_process_cmd_frame_dm Function

File

rfcomm_private.h

C

```
void _rfcomm_process_cmd_frame_dm(bt_rfcomm_session_p psess, bt_rfcomm_command_p pcmd);
```

Description

This is function _rfcomm_process_cmd_frame_dm.

_rfcomm_process_cmd_frame_sabm Function

File

rfcomm_private.h

C

```
void _rfcomm_process_cmd_frame_sabm(bt_rfcomm_session_p psess, bt_rfcomm_command_p pcmd);
```

Description

This is function _rfcomm_process_cmd_frame_sabm.

_rfcomm_process_cmd_frame_ua Function

File

rfcomm_private.h

C

```
void _rfcomm_process_cmd_frame_ua(bt_rfcomm_session_p psess, bt_rfcomm_command_p pcmd);
```

Description

This is function _rfcomm_process_cmd_frame_ua.

_rfcomm_process_cmd_frame_uih Function

File

rfcomm_private.h

C

```
void _rfcomm_process_cmd_frame_uih(bt_rfcomm_session_p psess, bt_rfcomm_command_p pcmd);
```

Description

From frame_uih.c

_rfcomm_process_mx_fc_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_process_mx_fc_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pres);
```

Description

This is function _rfcomm_process_mx_fc_response.

_rfcomm_process_mx_msc_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_process_mx_msc_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start,
bt_int len);
```

Description

This is function _rfcomm_process_mx_msc_response.

_rfcomm_process_mx_pn_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_process_mx_pn_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pres, bt_int data_start,
bt_int len);
```

Description

This is function _rfcomm_process_mx_pn_response.

_rfcomm_process_mx_rls_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_process_mx_rls_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start,
bt_int len);
```

Description

This is function _rfcomm_process_mx_rls_response.

_rfcomm_process_res_frame_disc Function

File

rfcomm_private.h

C

```
void _rfcomm_process_res_frame_disc(bt_rfcomm_session_p psess, bt_rfcomm_command_p pres);
```

Description

This is function _rfcomm_process_res_frame_disc.

_rfcomm_process_res_frame_dm Function

File

rfcomm_private.h

C

```
void _rfcomm_process_res_frame_dm(bt_rfcomm_session_p psess, bt_rfcomm_command_p pres);
```

Description

This is function _rfcomm_process_res_frame_dm.

_rfcomm_process_res_frame_sabm Function

File

rfcomm_private.h

C

```
void _rfcomm_process_res_frame_sabm(bt_rfcomm_session_p psess, bt_rfcomm_command_p pres);
```

Description

This is function _rfcomm_process_res_frame_sabm.

_rfcomm_process_res_frame_ua Function

File

rfcomm_private.h

C

```
void _rfcomm_process_res_frame_ua(bt_rfcomm_session_p psess, bt_rfcomm_command_p pres);
```

Description

This is function _rfcomm_process_res_frame_ua.

_rfcomm_process_res_frame_uih Function

File

rfcomm_private.h

C

```
void _rfcomm_process_res_frame_uih(bt_rfcomm_session_p psess, bt_rfcomm_command_p pres);
```

Description

This is function _rfcomm_process_res_frame_uih.

_rfcomm_send_command Function

File

rfcomm_private.h

C

```
bt_bool _rfcomm_send_command(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd);
```

Description

From rfcomm_cmd_send.c

_rfcomm_send_dm_response Function

File

rfcomm_private.h

C

```
void _rfcomm_send_dm_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd);
```

Description

From frame_dm.c

_rfcomm_send_mx_fc_cmd Function

File

rfcomm_mx.h

C

```
void _rfcomm_send_mx_fc_cmd(bt_rfcomm_dlc_p pdlc, bt_byte mx_msg_type);
```

Description

This is function _rfcomm_send_mx_fc_cmd.

_rfcomm_send_mx_msc_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_send_mx_msc_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start, bt_int
len);
```

Description

This is function _rfcomm_send_mx_msc_response.

_rfcomm_send_mx_nsc_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_send_mx_nsc_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_byte mx_msg_type);
```

Description

This is function _rfcomm_send_mx_nsc_response.

_rfcomm_send_mx_rls_cmd Function

File

rfcomm_mx.h

C

```
void _rfcomm_send_mx_rls_cmd(bt_rfcomm_dlc_p pdlc);
```

Description

This is function _rfcomm_send_mx_rls_cmd.

_rfcomm_send_mx_rls_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_send_mx_rls_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start, bt_int
len);
```

Description

This is function _rfcomm_send_mx_rls_response.

_rfcomm_send_mx_test_response Function

File

rfcomm_mx.h

C

```
void _rfcomm_send_mx_test_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_int data_start,
bt_int len);
```

Description

This is function _rfcomm_send_mx_test_response.

_rfcomm_send_sabm_cmd Function

File

rfcomm_private.h

C

```
void _rfcomm_send_sabm_cmd(bt_rfcomm_dlc_p pdlc, bt_rfcomm_cmd_callback_fp cb);
```

Description

From frame_sabm.c

_rfcomm_send_ua_response Function

File

rfcomm_private.h

C

```
void _rfcomm_send_ua_response(bt_rfcomm_dlc_p pdlc, bt_rfcomm_command_p pcmd, bt_rfcomm_cmd_callback_fp cb);
```

Description

From frame_ua.c

_bt_rfcomm_allocate_channel Function

File

rfcomm_mgr.h

C

```
bt_rfcomm_server_channel_t* _bt_rfcomm_allocate_channel(bt_byte id);
```

Description

This is function _bt_rfcomm_allocate_channel.

_bt_rfcomm_clear_queue Function

File

rfcomm_private.h

C

```
void _bt_rfcomm_clear_queue(bt_rfcomm_dlc_t* pdlc);
```

Description

From command_queue.c

_bt_rfcomm_find_channel Function

File

rfcomm_mgr.h

C

```
bt_rfcomm_server_channel_t* _bt_rfcomm_find_channel(bt_byte id);
```

Description

This is function _bt_rfcomm_find_channel.

_bt_rfcomm_get_mgr Function

File

rfcomm_mgr.h

C

```
bt_rfcomm_mgr_t* _bt_rfcomm_get_mgr();
```

Description

This is function _bt_rfcomm_get_mgr.

_bt_rfcomm_init_signal Function

File

rfcomm_signal.h

C

```
void _bt_rfcomm_init_signal();
```

Description

This is function _bt_rfcomm_init_signal.

_bt_rfcomm_mgr_allocate_session Function

File

rfcomm_private.h

C

```
bt_rfcomm_session_t* _bt_rfcomm_mgr_allocate_session(bt_bdaddr_t* bdaddr_remote);
```

Description

This is function _bt_rfcomm_mgr_allocate_session.

_bt_rfcomm_mgr_l2cap_listen_callback Function

File

rfcomm_mgr.h

C

```
void _bt_rfcomm_mgr_l2cap_listen_callback(bt_l2cap_channel_t* pch, void* param);
```

Description

This is function _bt_rfcomm_mgr_l2cap_listen_callback.

_bt_rfcomm_mgr_notify_listeners Function

File

rfcomm_mgr.h

C

```
void _bt_rfcomm_mgr_notify_listeners(bt_rfcomm_session_t* session, bt_int evt);
```

Description

This is function _bt_rfcomm_mgr_notify_listeners.

_bt_rfcomm_set_signal Function

File

rfcomm_signal.h

C

```
void _bt_rfcomm_set_signal();
```

Description

This is function _bt_rfcomm_set_signal.

_calc_fcs Function

File

rfcomm_private.h

C

```
bt_byte _calc_fcs(bt_byte_p buffer, bt_byte len);
```

Description

This is function _calc_fcs.

rfcomm_find_session Function

File

rfcomm_private.h

C

```
bt_rfcomm_session_p rfcomm_find_session(bt_l2cap_channel_t* pch);
```

Description

This is function rfcomm_find_session.

_rfcomm_l2cap_state_changed_callback Function

File

rfcomm_private.h

C

```
void _rfcomm_l2cap_state_changed_callback(bt_l2cap_channel_t* pch, bt_int new_state, void* param);
```

Description

From rfcomm.c

bt_rfcomm_cancel_listen Function

File

rfcomm.h

C

```
bt_bool bt_rfcomm_cancel_listen(bt_byte server_channel);
```

Description

This is function bt_rfcomm_cancel_listen.

bt_rfcomm_get_frame_length Function

File

rfcomm.h

C

```
bt_int bt_rfcomm_get_frame_length(bt_rfcomm_dlc_t* dlc);
```

Description

This is function bt_rfcomm_get_frame_length.

rfcomm_cq_find_failed_pn Function

File

rfcomm_cmd_queue.h

C

```
struct _bt_rfcomm_command_t* rfcomm_cq_find_failed_pn(struct _bt_rfcomm_dlc_t* pdlc, struct
_bt_rfcomm_command_t* dm);
```

Description

This is function rfcomm_cq_find_failed_pn.

RFCOMM Data Types and Constants

MK_CMD_ADDRESS Macro

File

rfcomm.h

C

```
#define MK_CMD_ADDRESS(addr, pdlc) ((addr & 0xfc) | pdlc->psess->role | RFCOMM_FLAG_EA)
```

Description

define RFCOMM_MS_RTC

MK_DLCI Macro

File

rfcomm.h

C

```
#define MK_DLCI(bt_server_channel, rfcomm_session) (bt_server_channel << 3 | ((~rfcomm_session->role & 2)
<< 1))
```

Description

This is macro MK_DLCI.

RFCOMM_CFC_ENABLED Macro

File

rfcomm.h

C

```
#define RFCOMM_CFC_ENABLED (pdlc->psess->fc_type == RFCOMM_FC_TYPE_CREDIT)
```

Description

This is macro RFCOMM_CFC_ENABLED.

RFCOMM_CFC_LOCAL_CREDIT Macro

File

rfcomm.h

C

```
#define RFCOMM_CFC_LOCAL_CREDIT _rfcomm_local_credit
```

Description

This is macro RFCOMM_CFC_LOCAL_CREDIT.

RFCOMM_CFC_MAX_INITIAL_CREDIT Macro

File

rfcomm.h

C

```
#define RFCOMM_CFC_MAX_INITIAL_CREDIT 7
```

Description

This is macro RFCOMM_CFC_MAX_INITIAL_CREDIT.

RFCOMM_CMD_STATUS_FC_PENDING Macro

File

rfcomm.h

C

```
#define RFCOMM_CMD_STATUS_FC_PENDING 2
```

Description

This is macro RFCOMM_CMD_STATUS_FC_PENDING.

RFCOMM_CMD_STATUS_PENDING Macro

File

rfcomm.h

C

```
#define RFCOMM_CMD_STATUS_PENDING 0
```

Description

This is macro RFCOMM_CMD_STATUS_PENDING.

RFCOMM_CMD_STATUS_WAITING_RESPONSE Macro

File

rfcomm.h

C

```
#define RFCOMM_CMD_STATUS_WAITING_RESPONSE 1
```

Description

This is macro RFCOMM_CMD_STATUS_WAITING_RESPONSE.

RFCOMM_COMMAND Macro

File

rfcomm.h

C

```
#define RFCOMM_COMMAND 0x02
```

Description

This is macro RFCOMM_COMMAND.

RFCOMM_CTL_MSG_CLD Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_CLD 0xc0
```

Description

This is macro RFCOMM_CTL_MSG_CLD.

RFCOMM_CTL_MSG_FCOFF Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_FCOFF 0x60
```

Description

This is macro RFCOMM_CTL_MSG_FCOFF.

RFCOMM_CTL_MSG_FCON Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_FCON 0x10
```

Description

This is macro RFCOMM_CTL_MSG_FCON.

RFCOMM_CTL_MSG_MSC Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_MSC 0xe0
```

Description

This is macro RFCOMM_CTL_MSG_MSC.

RFCOMM_CTL_MSG_NSC Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_NSC 0x10
```

Description

This is macro RFCOMM_CTL_MSG_NSC.

RFCOMM_CTL_MSG_PN Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_PN 0x80
```

Description

This is macro RFCOMM_CTL_MSG_PN.

RFCOMM_CTL_MSG_PSC Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_PSC 0x40
```

Description

This is macro RFCOMM_CTL_MSG_PSC.

RFCOMM_CTL_MSG_RLS Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_RLS 0x50
```

Description

This is macro RFCOMM_CTL_MSG_RLS.

RFCOMM_CTL_MSG_RPN Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_RPN 0x90
```

Description

This is macro RFCOMM_CTL_MSG_RPN.

RFCOMM_CTL_MSG_SNC Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_SNC 0xd0
```

Description

This is macro RFCOMM_CTL_MSG_SNC.

RFCOMM_CTL_MSG_TEST Macro

File

rfcomm.h

C

```
#define RFCOMM_CTL_MSG_TEST 0x20
```

Description

This is macro RFCOMM_CTL_MSG_TEST.

RFCOMM_DLC_CHANGED_CONN_STATE Macro

File

rfcomm.h

C

```
#define RFCOMM_DLC_CHANGED_CONN_STATE 0
```

Description

This is macro RFCOMM_DLC_CHANGED_CONN_STATE.

RFCOMM_DLC_CHANGED_REMOTE_MSC Macro

File

rfcomm.h

C

```
#define RFCOMM_DLC_CHANGED_REMOTE_MSC 1
```

Description

This is macro RFCOMM_DLC_CHANGED_REMOTE_MSC.

RFCOMM_DLC_CONNECTION_FAILED Macro

File

rfcomm.h

C

```
#define RFCOMM_DLC_CONNECTION_FAILED 2
```

Description

This is macro RFCOMM_DLC_CONNECTION_FAILED.

RFCOMM_DLC_STATE_CLOSED Macro

File

rfcomm.h

C

```
#define RFCOMM_DLC_STATE_CLOSED 0
```

Description

This is macro RFCOMM_DLC_STATE_CLOSED.

RFCOMM_DLC_STATE_OPEN Macro

File

rfcomm.h

C

```
#define RFCOMM_DLC_STATE_OPEN 1
```

Description

This is macro RFCOMM_DLC_STATE_OPEN.

RFCOMM_DLCI_CONTROL Macro

File

rfcomm.h

C

```
#define RFCOMM_DLCI_CONTROL 0
```

Description

This is macro RFCOMM_DLCI_CONTROL.

RFCOMM_DLCI_FREE Macro

File

rfcomm.h

C

```
#define RFCOMM_DLCI_FREE 0xff
```

Description

This is macro RFCOMM_DLCI_FREE.

RFCOMM_ERR_DM Macro

File

rfcomm.h

C

```
#define RFCOMM_ERR_DM 0x02
```

Description

This is macro RFCOMM_ERR_DM.

RFCOMM_ERR_SUCCESS Macro

File

rfcomm.h

C

```
#define RFCOMM_ERR_SUCCESS 0x00
```

Description

This is macro RFCOMM_ERR_SUCCESS.

RFCOMM_ERR_TIMEOUT Macro

File

rfcomm.h

C

```
#define RFCOMM_ERR_TIMEOUT 0x01
```

Description

This is macro RFCOMM_ERR_TIMEOUT.

RFCOMM_FC_TYPE_AGREGATE Macro

File

rfcomm.h

C

```
#define RFCOMM_FC_TYPE_AGREGATE 0
```

Description

This is macro RFCOMM_FC_TYPE_AGREGATE.

RFCOMM_FC_TYPE_CREDIT Macro

File

rfcomm.h

C

```
#define RFCOMM_FC_TYPE_CREDIT 1
```

Description

This is macro RFCOMM_FC_TYPE_CREDIT.

RFCOMM_FLAG_CR Macro

File

rfcomm.h

C

```
#define RFCOMM_FLAG_CR 0x02
```

Description

This is macro RFCOMM_FLAG_CR.

RFCOMM_FLAG_EA Macro

File

rfcomm.h

C

```
#define RFCOMM_FLAG_EA 0x01
```

Description

This is macro RFCOMM_FLAG_EA.

RFCOMM_FLAG_PF Macro

File

rfcomm.h

C

```
#define RFCOMM_FLAG_PF 0x10
```

Description

This is macro RFCOMM_FLAG_PF.

RFCOMM_FRAME_HEADER_LEN Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_HEADER_LEN 5
```

Description

This is macro RFCOMM_FRAME_HEADER_LEN.

RFCOMM_FRAME_TYPE_DISC Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_TYPE_DISC 0x43
```

Description

This is macro RFCOMM_FRAME_TYPE_DISC.

RFCOMM_FRAME_TYPE_DM Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_TYPE_DM 0x0f
```

Description

This is macro RFCOMM_FRAME_TYPE_DM.

RFCOMM_FRAME_TYPE_SABM Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_TYPE_SABM 0x2f
```

Description

This is macro RFCOMM_FRAME_TYPE_SABM.

RFCOMM_FRAME_TYPE_UA Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_TYPE_UA 0x63
```

Description

This is macro RFCOMM_FRAME_TYPE_UA.

RFCOMM_FRAME_TYPE_UI Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_TYPE_UI 0x03
```

Description

This is macro RFCOMM_FRAME_TYPE_UI.

RFCOMM_FRAME_TYPE_UIH Macro

File

rfcomm.h

C

```
#define RFCOMM_FRAME_TYPE_UIH 0xef
```

Description

This is macro RFCOMM_FRAME_TYPE_UIH.

RFCOMM_LINE_STATUS_FRAMING Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_LINE_STATUS_FRAMING x09
```

Description

This is macro RFCOMM_LINE_STATUS_FRAMING.

RFCOMM_LINE_STATUS_OVERRUN Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_LINE_STATUS_OVERRUN x03
```

Description

This is macro RFCOMM_LINE_STATUS_OVERRUN.

RFCOMM_LINE_STATUS_PARITY Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_LINE_STATUS_PARITY x05
```

Description

This is macro RFCOMM_LINE_STATUS_PARITY.

RFCOMM_MAX_INFO_LEN Macro

File

rfcomm.h

C

```
#define RFCOMM_MAX_INFO_LEN _rfcomm_info_len
```

Description

This is macro RFCOMM_MAX_INFO_LEN.

RFCOMM_MODEM_STATUS_DV Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MODEM_STATUS_DV 0x80
```

Description

This is macro RFCOMM_MODEM_STATUS_DV.

RFCOMM_MODEM_STATUS_FC Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MODEM_STATUS_FC 0x02
```

Description

This is macro RFCOMM_MODEM_STATUS_FC.

RFCOMM_MODEM_STATUS_IC Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MODEM_STATUS_IC 0x40
```

Description

This is macro RFCOMM_MODEM_STATUS_IC.

RFCOMM_MODEM_STATUS_RTC Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MODEM_STATUS_RTC 0x04
```

Description

This is macro RFCOMM_MODEM_STATUS_RTC.

RFCOMM_MODEM_STATUS_RTR Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MODEM_STATUS_RTR 0x08
```

Description

This is macro RFCOMM_MODEM_STATUS_RTR.

RFCOMM_MX_MSG_FCOFF Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_FCOFF 0x60
```

Description

This is macro RFCOMM_MX_MSG_FCOFF.

RFCOMM_MX_MSG_FCON Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_FCON 0xA0
```

Description

This is macro RFCOMM_MX_MSG_FCON.

RFCOMM_MX_MSG_MSC Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_MSC 0xE0
```

Description

This is macro RFCOMM_MX_MSG_MSC.

RFCOMM_MX_MSG_NSC Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_NSC 0x10
```

Description

This is macro RFCOMM_MX_MSG_NSC.

RFCOMM_MX_MSG_PN Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_PN 0x80
```

Description

This is macro RFCOMM_MX_MSG_PN.

RFCOMM_MX_MSG_RLS Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_RLS 0x50
```

Description

This is macro RFCOMM_MX_MSG_RLS.

RFCOMM_MX_MSG_RPN Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_RPN 0x90
```

Description

This is macro RFCOMM_MX_MSG_RPN.

RFCOMM_MX_MSG_TEST Macro

File

rfcomm_mx.h

C

```
#define RFCOMM_MX_MSG_TEST 0x20
```

Description

This is macro RFCOMM_MX_MSG_TEST.

RFCOMM_RESPONSE Macro

File

rfcomm.h

C

```
#define RFCOMM_RESPONSE 0x00
```

Description

This is macro RFCOMM_RESPONSE.

RFCOMM_ROLE_INITIATOR Macro

File

rfcomm.h

C

```
#define RFCOMM_ROLE_INITIATOR 0x02
```

Description

This is macro RFCOMM_ROLE_INITIATOR.

RFCOMM_ROLE_RESPONDER Macro

File

rfcomm.h

C

```
#define RFCOMM_ROLE_RESPONDER 0x00
```

Description

This is macro RFCOMM_ROLE_RESPONDER.

RFCOMM_RPN_BAUD_RATE_1152 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_1152 0x07
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_1152.

RFCOMM_RPN_BAUD_RATE_192 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_192 0x04
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_192.

RFCOMM_RPN_BAUD_RATE_2304 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_2304 0x08
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_2304.

RFCOMM_RPN_BAUD_RATE_24 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_24 0x00
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_24.

RFCOMM_RPN_BAUD_RATE_384 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_384 0x05
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_384.

RFCOMM_RPN_BAUD_RATE_48 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_48 0x01
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_48.

RFCOMM_RPN_BAUD_RATE_576 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_576 0x06
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_576.

RFCOMM_RPN_BAUD_RATE_72 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_72 0x02
```

Description

This is macro RFCOMM_RPN_BAUD_RATE_72.

RFCOMM_RPN_BAUD_RATE_96 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_BAUD_RATE_96 0x03 // default
```

Description

default

RFCOMM_RPN_DATA_BIT_5 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_DATA_BIT_5 0x00
```

Description

This is macro RFCOMM_RPN_DATA_BIT_5.

RFCOMM_RPN_DATA_BIT_6 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_DATA_BIT_6 0x01
```

Description

This is macro RFCOMM_RPN_DATA_BIT_6.

RFCOMM_RPN_DATA_BIT_7 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_DATA_BIT_7 0x02
```

Description

This is macro RFCOMM_RPN_DATA_BIT_7.

RFCOMM_RPN_DATA_BIT_8 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_DATA_BIT_8 0x03 // default
```

Description

default

RFCOMM_RPN_FLC_N Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_N 0x00
```

Description

This is macro RFCOMM_RPN_FLC_N.

RFCOMM_RPN_FLC_RTC_INPUT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_RTC_INPUT 0x10
```

Description

This is macro RFCOMM_RPN_FLC_RTC_INPUT.

RFCOMM_RPN_FLC_RTC_OUTPUT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_RTC_OUTPUT 0x20
```

Description

This is macro RFCOMM_RPN_FLC_RTC_OUTPUT.

RFCOMM_RPN_FLC_RTR_INPUT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_RTR_INPUT 0x04
```

Description

This is macro RFCOMM_RPN_FLC_RTR_INPUT.

RFCOMM_RPN_FLC_RTR_OUTPUT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_RTR_OUTPUT 0x08
```

Description

This is macro RFCOMM_RPN_FLC_RTR_OUTPUT.

RFCOMM_RPN_FLC_XONOFF_INPUT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_XONOFF_INPUT 0x01
```

Description

This is macro RFCOMM_RPN_FLC_XONOFF_INPUT.

RFCOMM_RPN_FLC_XONOFF_OUTPUT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_FLC_XONOFF_OUTPUT 0x02
```

Description

This is macro RFCOMM_RPN_FLC_XONOFF_OUTPUT.

RFCOMM_RPN_PARITY_EVEN Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_PARITY_EVEN 0x10
```

Description

This is macro RFCOMM_RPN_PARITY_EVEN.

RFCOMM_RPN_PARITY_MARK Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_PARITY_MARK 0x20
```

Description

This is macro RFCOMM_RPN_PARITY_MARK.

RFCOMM_RPN_PARITY_N Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_PARITY_N 0x00 // default
```

Description

default

RFCOMM_RPN_PARITY_ODD Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_PARITY_ODD 0x00
```

Description

This is macro RFCOMM_RPN_PARITY_ODD.

RFCOMM_RPN_PARITY_SPACE Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_PARITY_SPACE 0x30
```

Description

This is macro RFCOMM_RPN_PARITY_SPACE.

RFCOMM_RPN_PARITY_Y Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_PARITY_Y 0x08
```

Description

This is macro RFCOMM_RPN_PARITY_Y.

RFCOMM_RPN_STOP_BIT_1 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_STOP_BIT_1 0x00 // default
```

Description

default

RFCOMM_RPN_STOP_BIT_1_5 Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_STOP_BIT_1_5 0x04
```

Description

This is macro RFCOMM_RPN_STOP_BIT_1_5.

RFCOMM_RPN_XOFF_DEFAULT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_XOFF_DEFAULT 0x13
```

Description

This is macro RFCOMM_RPN_XOFF_DEFAULT.

RFCOMM_RPN_XON_DEFAULT Macro

File

rfcomm.h

C

```
#define RFCOMM_RPN_XON_DEFAULT 0x11
```

Description

This is macro RFCOMM_RPN_XON_DEFAULT.

RFCOMM_SERIAL_PORT_CH_1 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_1 1
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_1.

RFCOMM_SERIAL_PORT_CH_10 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_10 10
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_10.

RFCOMM_SERIAL_PORT_CH_11 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_11 11
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_11.

RFCOMM_SERIAL_PORT_CH_12 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_12 12
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_12.

RFCOMM_SERIAL_PORT_CH_13 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_13 13
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_13.

RFCOMM_SERIAL_PORT_CH_14 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_14 14
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_14.

RFCOMM_SERIAL_PORT_CH_15 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_15 15
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_15.

RFCOMM_SERIAL_PORT_CH_16 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_16 16
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_16.

RFCOMM_SERIAL_PORT_CH_17 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_17 17
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_17.

RFCOMM_SERIAL_PORT_CH_18 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_18 18
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_18.

RFCOMM_SERIAL_PORT_CH_19 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_19 19
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_19.

RFCOMM_SERIAL_PORT_CH_2 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_2 2
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_2.

RFCOMM_SERIAL_PORT_CH_20 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_20 20
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_20.

RFCOMM_SERIAL_PORT_CH_21 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_21 21
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_21.

RFCOMM_SERIAL_PORT_CH_22 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_22 22
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_22.

RFCOMM_SERIAL_PORT_CH_23 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_23 23
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_23.

RFCOMM_SERIAL_PORT_CH_24 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_24 24
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_24.

RFCOMM_SERIAL_PORT_CH_25 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_25 25
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_25.

RFCOMM_SERIAL_PORT_CH_26 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_26 26
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_26.

RFCOMM_SERIAL_PORT_CH_27 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_27 27
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_27.

RFCOMM_SERIAL_PORT_CH_28 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_28 28
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_28.

RFCOMM_SERIAL_PORT_CH_29 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_29 29
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_29.

RFCOMM_SERIAL_PORT_CH_3 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_3 3
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_3.

RFCOMM_SERIAL_PORT_CH_30 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_30 30
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_30.

RFCOMM_SERIAL_PORT_CH_4 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_4 4
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_4.

RFCOMM_SERIAL_PORT_CH_5 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_5 5
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_5.

RFCOMM_SERIAL_PORT_CH_6 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_6 6
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_6.

RFCOMM_SERIAL_PORT_CH_7 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_7 7
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_7.

RFCOMM_SERIAL_PORT_CH_8 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_8 8
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_8.

RFCOMM_SERIAL_PORT_CH_9 Macro

File

rfcomm.h

C

```
#define RFCOMM_SERIAL_PORT_CH_9 9
```

Description

This is macro RFCOMM_SERIAL_PORT_CH_9.

RFCOMM_SESSION_CHANGED_AFC Macro

File

rfcomm.h

C

```
#define RFCOMM_SESSION_CHANGED_AFC 1
```

Description

This is macro RFCOMM_SESSION_CHANGED_AFC.

RFCOMM_SESSION_CHANGED_CONN_STATE Macro

File

rfcomm.h

C

```
#define RFCOMM_SESSION_CHANGED_CONN_STATE 0
```

Description

This is macro RFCOMM_SESSION_CHANGED_CONN_STATE.

RFCOMM_SESSION_STATE_CONNECTED Macro

File

rfcomm.h

C

```
#define RFCOMM_SESSION_STATE_CONNECTED 1
```

Description

This is macro RFCOMM_SESSION_STATE_CONNECTED.

RFCOMM_SESSION_STATE_DISCONNECTED Macro

File

rfcomm.h

C

```
#define RFCOMM_SESSION_STATE_DISCONNECTED 2
```

Description

This is macro RFCOMM_SESSION_STATE_DISCONNECTED.

RFCOMM_SESSION_STATE_FREE Macro

File

rfcomm.h

C

```
#define RFCOMM_SESSION_STATE_FREE 0
```

Description

This is macro RFCOMM_SESSION_STATE_FREE.

RFCOMM_TIMEOUT Macro

File

rfcomm.h

C

```
#define RFCOMM_TIMEOUT 60
```

Description

This is macro RFCOMM_TIMEOUT.

bt_rfcomm_cmd_callback_fp Type

File

rfcomm.h

C

```
typedef void (* bt_rfcomm_cmd_callback_fp)(struct _bt_rfcomm_dlc_t* dlc, struct _bt_rfcomm_command_t *cmd,
bt_int status);
```

Description

This is type bt_rfcomm_cmd_callback_fp.

bt_rfcomm_command_p Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_command_t {
  struct _bt_rfcomm_command_t* next_cmd;
  bt_byte address;
  bt_byte control;
  bt_byte credit;
  bt_byte_p pdata;
  bt_int len;
  bt_byte status;
  bt_rfcomm_cmd_callback_fp cb;
  bt_rfcomm_send_data_callback_fp send_data_cb;
  bt_long send_time;
  bt_byte mx_params[RFCOMM_MX_MSG_MAX_DATA_LEN];
} bt_rfcomm_command_t, * bt_rfcomm_command_p;
```

Description

This is type bt_rfcomm_command_p.

bt_rfcomm_command_t Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_command_t {
  struct _bt_rfcomm_command_t* next_cmd;
  bt_byte address;
  bt_byte control;
  bt_byte credit;
  bt_byte_p pdata;
  bt_int len;
  bt_byte status;
  bt_rfcomm_cmd_callback_fp cb;
  bt_rfcomm_send_data_callback_fp send_data_cb;
```

```
    bt_long send_time;
    bt_byte mx_params[RFCOMM_MX_MSG_MAX_DATA_LEN];
  } bt_rfcomm_command_t, * bt_rfcomm_command_p;
```

Description

This is type bt_rfcomm_command_t.

bt_rfcomm_ctl_msg_p Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_ctl_msg_t {
  bt_byte type;
  bt_int len;
  bt_byte_p pdata;
} bt_rfcomm_ctl_msg_t, * bt_rfcomm_ctl_msg_p;
```

Description

This is type bt_rfcomm_ctl_msg_p.

bt_rfcomm_ctl_msg_t Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_ctl_msg_t {
  bt_byte type;
  bt_int len;
  bt_byte_p pdata;
} bt_rfcomm_ctl_msg_t, * bt_rfcomm_ctl_msg_p;
```

Description

This is type bt_rfcomm_ctl_msg_t.

bt_rfcomm_dlc_p Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_dlc_t {
  bt_byte id;
  bt_byte priority;
  bt_uint max_frame_size;
  bt_byte cfc_remote_credit;
  bt_byte cfc_freed_local_buffers;
  bt_byte state;
  bt_byte local_ms;
  bt_byte remote_ms;
  bt_byte local_ls;
  bt_byte remote_ls;
  bt_bool buffered;
  bt_rfcomm_read_data_callback_fp read_data_cb;
  bt_rfcomm_dlc_state_callback_fp state_cb;
  void* cb_param;
  bt_rfcomm_packet_t tx_packet;
  struct _bt_rfcomm_session_t * psess;
  bt_queue_element_t* cmd_queue;
} bt_rfcomm_dlc_t, * bt_rfcomm_dlc_p;
```

Description

This is type bt_rfcomm_dlc_p.

bt_rfcomm_dlc_state_callback_fp Type

File

rfcomm.h

C

```
typedef void (* bt_rfcomm_dlc_state_callback_fp)(struct _bt_rfcomm_dlc_t* dlc, bt_int what, void* param);
```

Description

This is type bt_rfcomm_dlc_state_callback_fp.

bt_rfcomm_dlc_t Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_dlc_t {
  bt_byte id;
  bt_byte priority;
  bt_uint max_frame_size;
  bt_byte cfc_remote_credit;
  bt_byte cfc_freed_local_buffers;
  bt_byte state;
  bt_byte local_ms;
  bt_byte remote_ms;
  bt_byte local_ls;
  bt_byte remote_ls;
  bt_bool buffered;
  bt_rfcomm_read_data_callback_fp read_data_cb;
  bt_rfcomm_dlc_state_callback_fp state_cb;
  void* cb_param;
  bt_rfcomm_packet_t tx_packet;
  struct _bt_rfcomm_session_t * psess;
  bt_queue_element_t* cmd_queue;
} bt_rfcomm_dlc_t, * bt_rfcomm_dlc_p;
```

Description

This is type bt_rfcomm_dlc_t.

bt_rfcomm_mgr_t Structure

File

rfcomm_mgr.h

C

```
typedef struct _bt_rfcomm_mgr_t {
  bt_rfcomm_session_listener_t* listeners;
  bt_rfcomm_server_channel_t* channels;
} bt_rfcomm_mgr_t;
```

Description

This is type bt_rfcomm_mgr_t.

bt_rfcomm_read_data_callback_fp Type

File

rfcomm.h

C

```
typedef void (* bt_rfcomm_read_data_callback_fp)(struct _bt_rfcomm_dlc_t* dlc, bt_byte_p data, bt_int len);
```

Description

This is type bt_rfcomm_read_data_callback_fp.

bt_rfcomm_send_data_callback_fp Type

File

rfcomm.h

C

```
typedef void (* bt_rfcomm_send_data_callback_fp)(struct _bt_rfcomm_dlc_t* dlc, bt_byte_p data, bt_int len,
bt_int status);
```

Description

This is type bt_rfcomm_send_data_callback_fp.

bt_rfcomm_server_channel_t Structure

File

rfcomm_mgr.h

C

```
typedef struct _bt_rfcomm_server_channel_t {
  bt_byte id;
  bt_rfcomm_dlc_state_callback_fp listen_cb;
  void* listen_param;
} bt_rfcomm_server_channel_t;
```

Description

This is type bt_rfcomm_server_channel_t.

bt_rfcomm_session_listener_t Structure

File

rfcomm_mgr.h

C

```
typedef struct _bt_rfcomm_session_listener_t {
  struct _bt_rfcomm_session_listener_t* next_listener;
  bt_rfcomm_state_callback_fp callback;
  void* callback_param;
} bt_rfcomm_session_listener_t;
```

Description

This is type bt_rfcomm_session_listener_t.

bt_rfcomm_session_p Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_session_t {
  bt_byte state;
  bt_l2cap_channel_t* pch;
  bt_byte role;
  bt_byte fc_type;
  bt_bool afc_off;
  bt_rfcomm_dlc_t* dlcs;
  bt_rfcomm_dlc_state_callback_fp connect_cb;
  void* connect_param;
  bt_byte connect_server_channel;
  bt_rfcomm_command_p pcmd_cur;
```

```
  bt_l2cap_mgr_p l2cap_mgr;
} bt_rfcomm_session_t, * bt_rfcomm_session_p;
```

Description

This is type bt_rfcomm_session_p.

bt_rfcomm_session_t Structure

File

rfcomm.h

C

```
typedef struct _bt_rfcomm_session_t {
  bt_byte state;
  bt_l2cap_channel_t* pch;
  bt_byte role;
  bt_byte fc_type;
  bt_bool afc_off;
  bt_rfcomm_dlc_t* dlcs;
  bt_rfcomm_dlc_state_callback_fp connect_cb;
  void* connect_param;
  bt_byte connect_server_channel;
  bt_rfcomm_command_p pcmd_cur;
  bt_l2cap_mgr_p l2cap_mgr;
} bt_rfcomm_session_t, * bt_rfcomm_session_p;
```

Description

This is type bt_rfcomm_session_t.

bt_rfcomm_state_callback_fp Type

File

rfcomm.h

C

```
typedef void (* bt_rfcomm_state_callback_fp)(struct _bt_rfcomm_session_t* session, bt_int what, void*
param);
```

Description

This is type bt_rfcomm_state_callback_fp.

RFCOMM_ERR_INTERRUPTED Macro

File

rfcomm.h

C

```
#define RFCOMM_ERR_INTERRUPTED 0x03
```

Description

This is macro RFCOMM_ERR_INTERRUPTED.

RFCOMM_MIX_INFO_LEN Macro

File

rfcomm.h

C

```
#define RFCOMM_MIX_INFO_LEN 31
```

Description

This is macro RFCOMM_MIX_INFO_LEN.

RFCOMM_MX_MSG_MAX_DATA_LEN Macro

File

rfcomm.h

C

```
#define RFCOMM_MX_MSG_MAX_DATA_LEN 10
```

Description

This is macro RFCOMM_MX_MSG_MAX_DATA_LEN.

SBC Decoder Functions

sbc_decode Function

Decodes the encoded SBC frame.

File

sbc.h

C

```
ssize_t sbc_decode(sbc_t * sbc, const void * input, size_t input_len, void * output, size_t output_len,
size_t * written);
```

Returns

This function returns the length of one encoded frame that has just been decoded.

Description

This function decodes the encoded SBC frame.

Remarks

This function is called once to decode one frame and will be called in an infinite loop. The decoder decodes one frame at a time even though there can be several frames in the input buffer. This function returns the length of the encoded frame that has just been decoded. The decoder also populates the sbc_t *sbc structure with the sampling rate, number of sub-bands, bitpool value, mode, and the allocation method.

Parameters

Parameters	Description
*sbc	Pointer to the SBC data structure.
*input	The pointer to the input buffer, where the encoded bit stream is available.
input_len	The number of valid input data bytes available in the above buffer.
*output	The pointer to the output sample buffer where the decoded PCM samples are to be written.
output_len	Maximum length of the output buffer.
*written	The number of output bytes written in the above buffer.

Function

ssize_t sbc_decode(sbc_t *sbc, const void *input, size_t input_len,

void *output, size_t output_len, size_t *written);

sbc_get_codesize Function

Gets the input block size.

File

sbc.h

C

```
size_t sbc_get_codesize(sbc_t * sbc);
```

Returns

This function returns the size of the encoded frame in bytes including the 4-byte header.

Description

This function gets the input block size.

Parameters

Parameters	Description
*sbc	Pointer to the allocated decoder state is passed.

Function

size_t sbc_get_codesize(sbc_t *sbc);

sbc_get_frame_duration Function

Gets the time one input/output block takes to play in microseconds.

File

sbc.h

C

```
uint32_t sbc_get_frame_duration(sbc_t * sbc);
```

Returns

This function returns the duration of the last decoded frame to be played. The duration is in microseconds.

Description

This function gets the time one input/output block takes to play in microseconds.

Parameters

Parameters	Description
*sbc	Pointer to the allocated decoder state is passed.

Function

uint32_t sbc_get_frame_duration(sbc_t *sbc);

sbc_get_frame_length Function

Gets the output block size.

File

sbc.h

C

```
size_t sbc_get_frame_length(sbc_t * sbc);
```

Returns

This function returns the size of the output frame (in bytes) that was decoded last.

Description

This function gets the output block size.

Parameters

Parameters	Description
*sbc	Pointer to the allocated decoder state is passed.

Function

size_t sbc_get_frame_length(sbc_t *sbc);

sbc_get_state_size Function

Gets the SBC Decoder's state data structure.

File

sbc.h

C

```
int16_t sbc_get_state_size();
```

Returns

This function returns the size of the SBC Decoder state structure in bytes.

Description

This function gets the SBC Decoder's state data structure size.

Function

int16_t sbc_get_state_size();

sbc_init Function

Initializes the SBC Decoder.

File

sbc.h

C

```
int32_t sbc_init(sbc_t * sbc, uint32_t flags);
```

Returns

This function always returns the value '0'; however, this return value is not used in the application.

Description

This function initializes the SBC Decoder.

Remarks

This function views the received pointer as structure pointer and clears the entire structure. This function is called only once for each channel before starting the decoding of the first frame.

Parameters

Parameters	Description
*sbc	Pointer to the SBC data structure.
flags	Unused. 0L is always passed.

Function

int32_t sbc_init(sbc_t *sbc, uint32_t flags);

SBC Decoder Data Types and Constants

sbc_struct Structure

File

sbc.h

C

```
struct sbc_struct {
  uint32_t flags;
```

```
    uint8_t frequency;
    uint8_t blocks;
    uint8_t subbands;
    uint8_t mode;
    uint8_t allocation;
    uint8_t bitpool;
    uint8_t endian;
    void * priv;
    void * priv_alloc_base;
    int32_t v_vec_fix[2][256+160];
    int32_t SamplingFreq;
    uint32_t Dword;
    int16_t bitcount;
    int16_t Pos;
    int16_t codeSize;
    int16_t outSize;
    int8_t BlocksIndex;
    int8_t SubBandsIndex;
    int8_t nrof_Chnls;
    int8_t Bits;
    uint8_t CrcCheck;
    uint8_t CrcData[2];
    uint8_t join[8];
    int8_t bits[2][8];
    uint8_t * Buffer;
};
```

Members

Members	Description
int32_t v_vec_fix[2][256+160];	Below variables / buffers added

Section

Data Types

sbc_t Type

File

sbc.h

C

```
typedef struct sbc_struct sbc_t;
```

Description

This is type sbc_t.

__SBC_H Macro

File

sbc.h

C

```
#define __SBC_H
```

Description

This is macro __SBC_H.

INP_BUF_SIZE Macro

File

sbc.h

C

```
#define INP_BUF_SIZE 512     // Bytes
```

Description

Bytes

MAX_SBC_DEC_STATE_SIZE Macro

File

sbc.h

C

```
#define MAX_SBC_DEC_STATE_SIZE 3500    // Bytes
```

Description

Bytes

OUT_BUF_SIZE Macro

File

sbc.h

C

```
#define OUT_BUF_SIZE 512     // Bytes
```

Description

Bytes

SBC_AM_LOUDNESS Macro

File

sbc.h

C

```
#define SBC_AM_LOUDNESS 0x00
```

Description

Allocation method

SBC_AM_SNR Macro

File

sbc.h

C

```
#define SBC_AM_SNR 0x01
```

Description

This is macro SBC_AM_SNR.

SBC_BE Macro

File

sbc.h

C

```
#define SBC_BE 0x01
```

Description

This is macro SBC_BE.

SBC_BLK_12 Macro

File

sbc.h

C

```
#define SBC_BLK_12 0x02
```

Description

This is macro SBC_BLK_12.

SBC_BLK_16 Macro

File

sbc.h

C

```
#define SBC_BLK_16 0x03
```

Description

This is macro SBC_BLK_16.

SBC_BLK_4 Macro

File

sbc.h

C

```
#define SBC_BLK_4 0x00
```

Description

Blocks

SBC_BLK_8 Macro

File

sbc.h

C

```
#define SBC_BLK_8 0x01
```

Description

This is macro SBC_BLK_8.

SBC_FREQ_16000 Macro

File

sbc.h

C

```
#define SBC_FREQ_16000 0x00
```

Description

Sampling frequency

SBC_FREQ_32000 Macro

File

sbc.h

C

```
#define SBC_FREQ_32000 0x01
```

Description

This is macro SBC_FREQ_32000.

SBC_FREQ_44100 Macro

File

sbc.h

C

```
#define SBC_FREQ_44100 0x02
```

Description

This is macro SBC_FREQ_44100.

SBC_FREQ_48000 Macro

File

sbc.h

C

```
#define SBC_FREQ_48000 0x03
```

Description

This is macro SBC_FREQ_48000.

SBC_LE Macro

File

sbc.h

C

```
#define SBC_LE 0x00
```

Description

Data endianness

SBC_MODE_DUAL_CHANNEL Macro

File

sbc.h

C

```
#define SBC_MODE_DUAL_CHANNEL 0x01
```

Description

This is macro SBC_MODE_DUAL_CHANNEL.

SBC_MODE_JOINT_STEREO Macro

File

sbc.h

C

```
#define SBC_MODE_JOINT_STEREO 0x03
```

Description

This is macro SBC_MODE_JOINT_STEREO.

SBC_MODE_MONO Macro

File

sbc.h

C

```
#define SBC_MODE_MONO 0x00
```

Description

Channel mode

SBC_MODE_STEREO Macro

File

sbc.h

C

```
#define SBC_MODE_STEREO 0x02
```

Description

This is macro SBC_MODE_STEREO.

SBC_SB_4 Macro

File

sbc.h

C

```
#define SBC_SB_4 0x00
```

Description

Sub-bands

SBC_SB_8 Macro

File

sbc.h

C

```
#define SBC_SB_8 0x01
```

Description

This is macro SBC_SB_8.

ssize_t Macro

File

sbc.h

C

```
#define ssize_t __int32_t
```

Section

Constants

SDP Functions

bt_sdp_de_to_uuid Function

File

sdp_utils.h

C

```
void bt_sdp_de_to_uuid(bt_sdp_data_element_cp pde, bt_uuid_p puuid);
```

Description

This is function bt_sdp_de_to_uuid.

bt_sdp_packet_assembler Function

File

sdp_packet.h

C

```
bt_int bt_sdp_packet_assembler(bt_packet_t* packet, bt_byte* buffer, bt_int buffer_len);
```

Description

This is function bt_sdp_packet_assembler.

bt_sdp_read_attribute Function

File

sdp_private.h

C

```
bt_bool bt_sdp_read_attribute(bt_sdp_server_attribute_t* attr, const bt_byte* buffer, bt_int len, bt_int*
offset);
```

Description

From sdp_db_utils.c

bt_sdp_start Function

File

sdp.h

C

```
bt_bool bt_sdp_start(bt_l2cap_mgr_p l2cap_mgr, const bt_byte* sdp_db, bt_uint sdp_db_len);
```

Description

brief Start SDP server ingroup sdp

details This function starts the SDP server.

param l2cap_mgr The L2CAP manager on which the SDP server is to be started. param sdp_db A pointer to the SDP database define with BEGIN_SDP_DB and END_SDP_DB macros.

return li c TRUE if the function succeeds. li c FALSE otherwise.

sdp_compare_uuid_de Function

File

sdp_private.h

C

```
bt_bool sdp_compare_uuid_de(bt_sdp_data_element_cp pde1, bt_sdp_data_element_cp pde2);
```

Description

This is function sdp_compare_uuid_de.

sdp_find_attributes Function

File

sdp_private.h

C

```
bt_bool sdp_find_attributes(bt_sr_handle_p h_list, bt_int h_count, bt_byte_p pparams, bt_int params_len,
bt_int offsetInit, bt_byte patternCount, bt_sdp_transaction_t* ptran, bt_int max_attrs);
```

Description

This is function sdp_find_attributes.

sdp_find_service_records2 Function

File

sdp_private.h

C

```
int sdp_find_service_records2(bt_byte_p pparams, bt_int params_len, bt_int offsetInit, bt_byte
patternCount, bt_sdp_service_transaction_p ptran, bt_int max_handles);
```

Description

From sdp_utils.c

_sdp_alloc_svc_tran_buffer Function

File

sdp_private.h

C

```
bt_sdp_service_transaction_p _sdp_alloc_svc_tran_buffer(bt_int id);
```

Description

This is function _sdp_alloc_svc_tran_buffer.

_sdp_alloc_tran_buffer Function

File

sdp_private.h

C

```
bt_sdp_transaction_t* _sdp_alloc_tran_buffer(bt_int id);
```

Description

This is function _sdp_alloc_tran_buffer.

_sdp_find_svc_transaction Function

File

sdp_private.h

C

```
bt_sdp_service_transaction_p _sdp_find_svc_transaction(bt_int id);
```

Description

This is function _sdp_find_svc_transaction.

_sdp_find_transaction Function

File

sdp_private.h

C

```
bt_sdp_transaction_t* _sdp_find_transaction(bt_int id);
```

Description

This is function _sdp_find_transaction.

_sdp_free_svc_tran_buffer Function

File

sdp_private.h

C

```
void _sdp_free_svc_tran_buffer(bt_sdp_service_transaction_p ptran);
```

Description

This is function _sdp_free_svc_tran_buffer.

_sdp_free_tran_buffer Function

File

sdp_private.h

C

```
void _sdp_free_tran_buffer(bt_sdp_transaction_t* ptran);
```

Description

This is function _sdp_free_tran_buffer.

_sdp_get_de_data_len Function

File

sdp_utils.h

C

```
bt_ulong _sdp_get_de_data_len(bt_sdp_data_element_cp pde);
```

Description

This is function _sdp_get_de_data_len.

_sdp_get_de_hdr_len Function

File

sdp_utils.h

C

```
bt_ulong _sdp_get_de_hdr_len(bt_sdp_data_element_cp pde);
```

Description

This is function _sdp_get_de_hdr_len.

_sdp_init_tran_buffers Function

File

sdp_private.h

C

```
bt_bool _sdp_init_tran_buffers();
```

Description

From sdp_tran_buffer.c

_sdp_read_de_header Function

File

sdp_utils.h

C

```
bt_bool _sdp_read_de_header(bt_byte_cp buffer, bt_int len, bt_int_p poffset, bt_byte_p pde_type, bt_byte_p
pde_size_index, bt_ulong_p pde_data_len);
```

Description

This is function _sdp_read_de_header.

_sdp_write_data_element Function

File

sdp_utils.h

C

```
bt_bool _sdp_write_data_element(bt_sdp_data_element_cp pde, bt_byte_p buffer, bt_int len, bt_int_p poffset);
```

Description

This is function _sdp_write_data_element.

bt_sdp_request_service_attribute Function

File

sdp_client.h

C

```
bt_bool bt_sdp_request_service_attribute(bt_l2cap_channel_t* channel, bt_sr_handle_t sr,
bt_sdp_data_element_p pattern, bt_sdp_service_attribute_callback_fp callback, void* callback_param);
```

Description

brief Search attributes ingroup sdp

details This function retrieves attribute values from a service record.

param channel The L2CAP channel used to communicate to the remote SDP server. param sr The service record handle specifies the service

record from which attribute values are to be retrieved. param pattern The attribute search pattern is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. The pattern buffer must be valid for the duration of the search operation, i.e. until c callback is called for the first time. To define a data element sequence use the BEGIN_DE_SEQUENCE and END_DE_SEQUENCE macros. These macros will define a variable whose name is the id of the data element sequence passed to the macros prefixed with "seq_". A pointer to this variable can be used as the value for the pattern parameter.

param callback The callback function that will be called when search has completed.

param callback_param A pointer to arbitrary data to be passed to the c callback callback.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

bt_sdp_request_service_search Function

File

sdp_client.h

C

```
bt_bool bt_sdp_request_service_search(bt_l2cap_channel_t* channel, bt_sdp_data_element_p pattern,
bt_sdp_service_search_callback_fp callback, void* callback_param);
```

Description

brief Search service records ingroup sdp

details This function locates service records on a remote SDP server that match the given service search pattern.

param channel The L2CAP channel used to communicate to the remote SDP server. param pattern The service search pattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12. The pattern buffer must be valid for the duration of the search operation, i.e. until c callback is called. To define a data element sequence use the BEGIN_DE_SEQUENCE and END_DE_SEQUENCE macros. These macros will define a variable whose name is the id of the data element sequence passed to the macros prefixed with "seq_". A pointer to this variable can be used as the value for the pattern parameter. param callback The callback function that will be called when search has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

_bt_sdp_client_init Function

File

sdp_private.h

C

```
void _bt_sdp_client_init();
```

Description

This is function _bt_sdp_client_init.

SDP Data Types and Constants

BEGIN_DE_SEQUENCE Macro

File

sdp.h

C

```
#define BEGIN_DE_SEQUENCE(id, len) \
    static bt_sdp_data_element_t id[len];    \
    static bt_sdp_sequence_t seq_##id = { len, (bt_sdp_data_element_p)id};    \
    static void init_de_sequence_##id() {    \
        static bt_bool initialized = FALSE;    \
        if (!initialized) {    \
            bt_int i = 0;    \
            bt_int max_len = len;    \
            bt_sdp_data_element_t* cur_de = id;    \
            initialized = TRUE;
```

Description

- brief Begin a data element sequence
- ingroup sdp
- *
- details BEGIN_DE_SEQUENCE and END_DE_SEQUENCE are used to define a data element sequence which is an array of sdp_data_element structures.
- The array is used a search pattern in bt_sdp_request_service_search() and bt_sdp_request_service_attribute().
- For example, to find a AVRCP Target the following code can be used:

code const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000, SDP_CLSID_AV_REMOTE_CONTROL }; const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000, SDP_CLSID_AV_REMOTE_CONTROL_TARGET };

BEGIN_DE_SEQUENCE(avrcp_target_service_search, 2) DE_UUID128(AVRCP_AV_REMOTE_CONTROL_CLSID) DE_UUID128(AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID) END_DE_SEQUENCE(avrcp_target_service_search)

. . .

void findAvrcpTarget(void) { INIT_DE_SEQUENCE(avrcp_target_service_search);

bt_sdp_request_service_search(channel, &seq_avrcp_target_service_search, &callback, NULL); }

endcode

*

- param id The data element sequence identifier.
- param len The number of elements in the data element sequence.

DE_BOOL Macro

File

sdp.h

C

```
#define DE_BOOL(value) cur_de->type = SDP_DATA_TYPE_BOOL; cur_de->data.b = value; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a boolean data element ingroup sdp

details This macro adds a boolean data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

DE_INT Macro

File

sdp.h

C

```
#define DE_INT(value) cur_de->type = SDP_DATA_TYPE_INT; cur_de->data.b = value; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a 1-byte signed integer data element ingroup sdp

details This macro adds a 1-byte signed integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

DE_STRING Macro

File

sdp.h

C

```
#define DE_STRING(value) cur_de->type = SDP_DATA_TYPE_STRING; cur_de->data.pstr = value; cur_de++;
```

Description

brief Declare a text string data element ingroup sdp

details This macro adds a text string data element to a data element sequence. The length of the generated data element will be the actual length of the string. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

DE_STRING2 Macro

File

sdp.h

C

```
#define DE_STRING2(value, len) cur_de->type = SDP_DATA_TYPE_UINT; cur_de->data.pstr = value;
cur_de->bytecount = len; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a text string data element ingroup sdp

details This macro adds a text string data element to a data element sequence. The length of the generated data element will be the value specified by the "len" parameter even if the actual length of the string is not equal to the "len" value. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value. param len The length of the data element value.

DE_UINT Macro

File

sdp.h

C

```
#define DE_UINT(value) cur_de->type = SDP_DATA_TYPE_UINT; cur_de->data.b = value; cur_de++; if (++i ==
max_len) return;
```

Description

brief Declare a 1-byte unsigned integer data element ingroup sdp

details This macro adds a 1-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

DE_UINT16 Macro

File

sdp.h

C

```
#define DE_UINT16(value) cur_de->type = SDP_DATA_TYPE_UINT16; cur_de->data.ui = value; cur_de++; if (++i ==
max_len) return;
```

Description

brief Declare a 2-byte unsigned integer data element ingroup sdp

details This macro adds a 2-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

DE_URL Macro

File

sdp.h

C

```
#define DE_URL(value) cur_de->type = SDP_DATA_TYPE_URL; cur_de->data.purl = value; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a URL data element ingroup sdp

details This macro adds a URL data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value which is a pointer to a string.

DE_UUID128 Macro

File

sdp.h

C

```
#define DE_UUID128(value) cur_de->type = SDP_DATA_TYPE_UUID128; cur_de->data.uuid128 = (bt_uuid_t*)&value; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a 128-bit UUID data element ingroup sdp

details This macro adds a 128-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value. The value must be a name of a variable of type bt_uuid.

DE_UUID16 Macro

File

sdp.h

C

```
#define DE_UUID16(value) cur_de->type = SDP_DATA_TYPE_UUID16; cur_de->data.uuid16 = value; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a 16-bit UUID data element ingroup sdp

details This macro adds a 16-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

DE_UUID32 Macro

File

sdp.h

C

```
#define DE_UUID32(value) cur_de->type = SDP_DATA_TYPE_UUID32; cur_de->data.uuid32 = value; cur_de++; if (++i == max_len) return;
```

Description

brief Declare a 32-bit UUID data element ingroup sdp

details This macro adds a 32-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

param value The data element value.

END_DE_SEQUENCE Macro

File

sdp.h

C

```
#define END_DE_SEQUENCE(id) }}
```

Description

brief End a data element sequence ingroup sdp

details BEGIN_DE_SEQUENCE and END_DE_SEQUENCE are used to define a data element sequence which is an array of bt_sdp_data_element structures.

param id The data element sequence identifier.

INIT_DE_SEQUENCE Macro

File

sdp.h

C

```
#define INIT_DE_SEQUENCE(id) init_de_sequence_##id();
```

Description

brief Initialize a data element sequence ingroup sdp

details This macro calls a function defined in BEGIN_DE_SEQUENCE which initializes the data element sequence.

param id The data element sequence identifier.

SDP_ATTRID_AdditionalProtocolDescriptorLists Macro

File

sdp.h

C

```
#define SDP_ATTRID_AdditionalProtocolDescriptorLists 0x000D
```

Description

This is macro SDP_ATTRID_AdditionalProtocolDescriptorLists.

SDP_ATTRID_BluetoothProfileDescriptorList Macro

File

sdp.h

C

```
#define SDP_ATTRID_BluetoothProfileDescriptorList 0x0009
```

Description

This is macro SDP_ATTRID_BluetoothProfileDescriptorList.

SDP_ATTRID_BrowseGroupList Macro

File

sdp.h

C

```
#define SDP_ATTRID_BrowseGroupList 0x0005
```

Description

This is macro SDP_ATTRID_BrowseGroupList.

SDP_ATTRID_ClientExecutableURL Macro

File

sdp.h

C

```
#define SDP_ATTRID_ClientExecutableURL 0x000B
```

Description

This is macro SDP_ATTRID_ClientExecutableURL.

SDP_ATTRID_DIPrimaryRecord Macro

File

sdp.h

C

```
#define SDP_ATTRID_DIPrimaryRecord 0x204
```

Description

This is macro SDP_ATTRID_DIPrimaryRecord.

SDP_ATTRID_DIProductId Macro

File

sdp.h

C

```
#define SDP_ATTRID_DIProductId 0x202
```

Description

This is macro SDP_ATTRID_DIProductId.

SDP_ATTRID_DISpecificationId Macro

File

sdp.h

C

```
#define SDP_ATTRID_DISpecificationId 0x200
```

Description

This is macro SDP_ATTRID_DISpecificationId.

SDP_ATTRID_DIVendorId Macro

File

sdp.h

C

```
#define SDP_ATTRID_DIVendorId 0x201
```

Description

This is macro SDP_ATTRID_DIVendorId.

SDP_ATTRID_DIVendorIdSource Macro

File

sdp.h

C

```
#define SDP_ATTRID_DIVendorIdSource 0x205
```

Description

This is macro SDP_ATTRID_DIVendorIdSource.

SDP_ATTRID_DIVersion Macro

File

sdp.h

C

```
#define SDP_ATTRID_DIVersion 0x203
```

Description

This is macro SDP_ATTRID_DIVersion.

SDP_ATTRID_DocumentationURL Macro

File

sdp.h

C

```
#define SDP_ATTRID_DocumentationURL 0x000A
```

Description

This is macro SDP_ATTRID_DocumentationURL.

SDP_ATTRID_GAPRemoteAudioVolumeControl Macro

File

sdp.h

C

```
#define SDP_ATTRID_GAPRemoteAudioVolumeControl 0x302
```

Description

This is macro SDP_ATTRID_GAPRemoteAudioVolumeControl.

SDP_ATTRID_GroupID Macro

File

sdp.h

C

```
#define SDP_ATTRID_GroupID 0x0200
```

Description

This is macro SDP_ATTRID_GroupID.

SDP_ATTRID_HDPDataExchangeSpecification Macro

File

sdp.h

C

```
#define SDP_ATTRID_HDPDataExchangeSpecification 0x301
```

Description

This is macro SDP_ATTRID_HDPDataExchangeSpecification.

SDP_ATTRID_HDPMCAPSupportedProcedures Macro

File

sdp.h

C

```
#define SDP_ATTRID_HDPMCAPSupportedProcedures 0x302
```

Description

This is macro SDP_ATTRID_HDPMCAPSupportedProcedures.

SDP_ATTRID_HDPSuportedFeatures Macro

File

sdp.h

C

```
#define SDP_ATTRID_HDPSuportedFeatures 0x200
```

Description

This is macro SDP_ATTRID_HDPSuportedFeatures.

SDP_ATTRID_HFPAGNetwork Macro

File

sdp.h

C

```
#define SDP_ATTRID_HFPAGNetwork 0x301
```

Description

This is macro SDP_ATTRID_HFPAGNetwork.

SDP_ATTRID_HFPSupportedFeatures Macro

File

sdp.h

C

```
#define SDP_ATTRID_HFPSupportedFeatures 0x311
```

Description

This is macro SDP_ATTRID_HFPSupportedFeatures.

SDP_ATTRID_HIDBatteryPower Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDBatteryPower 0x209
```

Description

This is macro SDP_ATTRID_HIDBatteryPower.

SDP_ATTRID_HIDBootDevice Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDBootDevice 0x20E
```

Description

This is macro SDP_ATTRID_HIDBootDevice.

SDP_ATTRID_HIDCountryCode Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDCountryCode 0x203
```

Description

This is macro SDP_ATTRID_HIDCountryCode.

SDP_ATTRID_HIDDescriptorList Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDDescriptorList 0x206
```

Description

This is macro SDP_ATTRID_HIDDescriptorList.

SDP_ATTRID_HIDDeviceReleaseNumber Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDDeviceReleaseNumber 0x200
```

Description

This is macro SDP_ATTRID_HIDDeviceReleaseNumber.

SDP_ATTRID_HIDDeviceSubclass Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDDeviceSubclass 0x202
```

Description

This is macro SDP_ATTRID_HIDDeviceSubclass.

SDP_ATTRID_HIDLANGIDBaseList Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDLANGIDBaseList 0x207
```

Description

This is macro SDP_ATTRID_HIDLANGIDBaseList.

SDP_ATTRID_HIDNormallyConnectable Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDNormallyConnectable 0x20D
```

Description

This is macro SDP_ATTRID_HIDNormallyConnectable.

SDP_ATTRID_HIDParserVersion Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDParserVersion 0x201
```

Description

This is macro SDP_ATTRID_HIDParserVersion.

SDP_ATTRID_HIDProfileVersion Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDProfileVersion 0x20B
```

Description

This is macro SDP_ATTRID_HIDProfileVersion.

SDP_ATTRID_HIDReconnectInitiate Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDReconnectInitiate 0x205
```

Description

This is macro SDP_ATTRID_HIDReconnectInitiate.

SDP_ATTRID_HIDRemoteWake Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDRemoteWake 0x20A
```

Description

This is macro SDP_ATTRID_HIDRemoteWake.

SDP_ATTRID_HIDSDPDisable Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDSDPDisable 0x208
```

Description

This is macro SDP_ATTRID_HIDSDPDisable.

SDP_ATTRID_HIDSupervisionTimeout Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDSupervisionTimeout 0x20C
```

Description

This is macro SDP_ATTRID_HIDSupervisionTimeout.

SDP_ATTRID_HIDVirtualCable Macro

File

sdp.h

C

```
#define SDP_ATTRID_HIDVirtualCable 0x204
```

Description

This is macro SDP_ATTRID_HIDVirtualCable.

SDP_ATTRID_IconURL Macro

File

sdp.h

C

```
#define SDP_ATTRID_IconURL 0x000C
```

Description

This is macro SDP_ATTRID_IconURL.

SDP_ATTRID_INVALID Macro

File

sdp.h

C

```
#define SDP_ATTRID_INVALID 0xFFFF
```

Description

This is macro SDP_ATTRID_INVALID.

SDP_ATTRID_LanguageBaseAttributeIDList Macro

File

sdp.h

C

```
#define SDP_ATTRID_LanguageBaseAttributeIDList 0x0006
```

Description

This is macro SDP_ATTRID_LanguageBaseAttributeIDList.

SDP_ATTRID_OFFSET_ProviderName Macro

File

sdp.h

C

```
#define SDP_ATTRID_OFFSET_ProviderName 0x0002
```

Description

This is macro SDP_ATTRID_OFFSET_ProviderName.

SDP_ATTRID_OFFSET_ServiceDescription Macro

File

sdp.h

C

```
#define SDP_ATTRID_OFFSET_ServiceDescription 0x0001
```

Description

This is macro SDP_ATTRID_OFFSET_ServiceDescription.

SDP_ATTRID_OFFSET_ServiceName Macro

File

sdp.h

C

```
#define SDP_ATTRID_OFFSET_ServiceName 0x0000
```

Description

This is macro SDP_ATTRID_OFFSET_ServiceName.

SDP_ATTRID_PrimaryLanguageBaseId Macro

File

sdp.h

C

```
#define SDP_ATTRID_PrimaryLanguageBaseId 0x0100
```

Description

This is macro SDP_ATTRID_PrimaryLanguageBaseId.

SDP_ATTRID_ProtocolDescriptorList Macro

File

sdp.h

C

```
#define SDP_ATTRID_ProtocolDescriptorList 0x0004
```

Description

This is macro SDP_ATTRID_ProtocolDescriptorList.

SDP_ATTRID_ServiceAvailability Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceAvailability 0x0008
```

Description

This is macro SDP_ATTRID_ServiceAvailability.

SDP_ATTRID_ServiceClassIDList Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceClassIDList 0x0001
```

Description

This is macro SDP_ATTRID_ServiceClassIDList.

SDP_ATTRID_ServiceDatabaseState Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceDatabaseState 0x0201
```

Description

This is macro SDP_ATTRID_ServiceDatabaseState.

SDP_ATTRID_ServiceID Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceID 0x0003
```

Description

This is macro SDP_ATTRID_ServiceID.

SDP_ATTRID_ServiceInfoTimeToLive Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceInfoTimeToLive 0x0007
```

Description

This is macro SDP_ATTRID_ServiceInfoTimeToLive.

SDP_ATTRID_ServiceRecordHandle Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceRecordHandle 0x0000
```

Description

This is macro SDP_ATTRID_ServiceRecordHandle.

SDP_ATTRID_ServiceRecordState Macro

File

sdp.h

C

```
#define SDP_ATTRID_ServiceRecordState 0x0002
```

Description

This is macro SDP_ATTRID_ServiceRecordState.

SDP_ATTRID_SupportedFeatures Macro

File

sdp.h

C

```
#define SDP_ATTRID_SupportedFeatures 0x311
```

Description

This is macro SDP_ATTRID_SupportedFeatures.

SDP_ATTRID_VersionNumberList Macro

File

sdp.h

C

```
#define SDP_ATTRID_VersionNumberList 0x0200
```

Description

This is macro SDP_ATTRID_VersionNumberList.

SDP_CLIENT_EVT_CONNECTED Macro

File

sdp_client.h

C

```
#define SDP_CLIENT_EVT_CONNECTED 1
```

Description

This is macro SDP_CLIENT_EVT_CONNECTED.

SDP_CLIENT_EVT_DISCONNECTED Macro

File

sdp_client.h

C

```
#define SDP_CLIENT_EVT_DISCONNECTED 2
```

Description

This is macro SDP_CLIENT_EVT_DISCONNECTED.

SDP_CLIENT_EVT_NULL Macro

File

sdp_client.h

C

```
#define SDP_CLIENT_EVT_NULL 0
```

Description

This is macro SDP_CLIENT_EVT_NULL.

SDP_CLSID_ADVANCED_AUDIO_DISTRIBUTION Macro

File

sdp.h

C

```
#define SDP_CLSID_ADVANCED_AUDIO_DISTRIBUTION 0x110D
```

Description

This is macro SDP_CLSID_ADVANCED_AUDIO_DISTRIBUTION.

SDP_CLSID_AUDIO_SINK Macro

File

sdp.h

C

```
#define SDP_CLSID_AUDIO_SINK 0x110B
```

Description

This is macro SDP_CLSID_AUDIO_SINK.

SDP_CLSID_AUDIO_SOURCE Macro

File

sdp.h

C

```
#define SDP_CLSID_AUDIO_SOURCE 0x110A
```

Description

This is macro SDP_CLSID_AUDIO_SOURCE.

SDP_CLSID_AV_REMOTE_CONTROL Macro

File

sdp.h

C

```
#define SDP_CLSID_AV_REMOTE_CONTROL 0x110E
```

Description

This is macro SDP_CLSID_AV_REMOTE_CONTROL.

SDP_CLSID_AV_REMOTE_CONTROL_CONTROLLER Macro

File

sdp.h

C

```
#define SDP_CLSID_AV_REMOTE_CONTROL_CONTROLLER 0x110F
```

Description

This is macro SDP_CLSID_AV_REMOTE_CONTROL_CONTROLLER.

SDP_CLSID_AV_REMOTE_CONTROL_PROFILE_ID Macro

File

sdp.h

C

```
#define SDP_CLSID_AV_REMOTE_CONTROL_PROFILE_ID 0x110E
```

Description

This is macro SDP_CLSID_AV_REMOTE_CONTROL_PROFILE_ID.

SDP_CLSID_AV_REMOTE_CONTROL_TARGET Macro

File

sdp.h

C

```
#define SDP_CLSID_AV_REMOTE_CONTROL_TARGET 0x110C
```

Description

This is macro SDP_CLSID_AV_REMOTE_CONTROL_TARGET.

SDP_CLSID_AVCTP Macro

File

sdp.h

C

```
#define SDP_CLSID_AVCTP 0x17
```

Description

This is macro SDP_CLSID_AVCTP.

SDP_CLSID_AVDTP Macro

File

sdp.h

C

```
#define SDP_CLSID_AVDTP 0x19
```

Description

This is macro SDP_CLSID_AVDTP.

SDP_CLSID_BrowseGroupDescriptorServiceClassID Macro

File

sdp.h

C

```
#define SDP_CLSID_BrowseGroupDescriptorServiceClassID 0x1001
```

Description

This is macro SDP_CLSID_BrowseGroupDescriptorServiceClassID.

SDP_CLSID_DialupNetworking Macro

File

sdp.h

C

```
#define SDP_CLSID_DialupNetworking 0x1103
```

Description

This is macro SDP_CLSID_DialupNetworking.

SDP_CLSID_GENERIC_AUDIO Macro

File

sdp.h

C

```
#define SDP_CLSID_GENERIC_AUDIO 0x1203
```

Description

This is macro SDP_CLSID_GENERIC_AUDIO.

SDP_CLSID_HDP Macro

File

sdp.h

C

```
#define SDP_CLSID_HDP 0x1400
```

Description

This is macro SDP_CLSID_HDP.

SDP_CLSID_HDP_SINK Macro

File

sdp.h

C

```
#define SDP_CLSID_HDP_SINK 0x1402
```

Description

This is macro SDP_CLSID_HDP_SINK.

SDP_CLSID_HDP_SOURCE Macro

File

sdp.h

C

```
#define SDP_CLSID_HDP_SOURCE 0x1401
```

Description

This is macro SDP_CLSID_HDP_SOURCE.

SDP_CLSID_HFP Macro

File

sdp.h

C

```
#define SDP_CLSID_HFP 0x111E
```

Description

This is macro SDP_CLSID_HFP.

SDP_CLSID_HFP_AG Macro

File

sdp.h

C

```
#define SDP_CLSID_HFP_AG 0x111F
```

Description

This is macro SDP_CLSID_HFP_AG.

SDP_CLSID_HID Macro

File

sdp.h

C

```
#define SDP_CLSID_HID 0x1124
```

Description

This is macro SDP_CLSID_HID.

SDP_CLSID_HIDProtocol Macro

File

sdp.h

C

```
#define SDP_CLSID_HIDProtocol 0x11
```

Description

This is macro SDP_CLSID_HIDProtocol.

SDP_CLSID_HSP Macro

File

sdp.h

C

```
#define SDP_CLSID_HSP 0x1108
```

Description

This is macro SDP_CLSID_HSP.

SDP_CLSID_HSP_AG Macro

File

sdp.h

C

```
#define SDP_CLSID_HSP_AG 0x1112
```

Description

This is macro SDP_CLSID_HSP_AG.

SDP_CLSID_HSP_HS Macro

File

sdp.h

C

```
#define SDP_CLSID_HSP_HS 0x1131
```

Description

This is macro SDP_CLSID_HSP_HS.

SDP_CLSID_L2CAP Macro

File

sdp.h

C

```
#define SDP_CLSID_L2CAP 0x100
```

Description

This is macro SDP_CLSID_L2CAP.

SDP_CLSID_MCAP_CONTROL Macro

File

sdp.h

C

```
#define SDP_CLSID_MCAP_CONTROL 0x001E
```

Description

This is macro SDP_CLSID_MCAP_CONTROL.

SDP_CLSID_MCAP_DATA Macro

File

sdp.h

C

```
#define SDP_CLSID_MCAP_DATA 0x001F
```

Description

This is macro SDP_CLSID_MCAP_DATA.

SDP_CLSID_OBEXFileTransfer Macro

File

sdp.h

C

```
#define SDP_CLSID_OBEXFileTransfer 0x1106
```

Description

This is macro SDP_CLSID_OBEXFileTransfer.

SDP_CLSID_OBEXObjectPush Macro

File

sdp.h

C

```
#define SDP_CLSID_OBEXObjectPush 0x1105
```

Description

This is macro SDP_CLSID_OBEXObjectPush.

SDP_CLSID_PBAP_PCE Macro

File

sdp.h

C

```
#define SDP_CLSID_PBAP_PCE 0x112E
```

Description

This is macro SDP_CLSID_PBAP_PCE.

SDP_CLSID_PBAP_PSE Macro

File

sdp.h

C

```
#define SDP_CLSID_PBAP_PSE 0x112F
```

Description

This is macro SDP_CLSID_PBAP_PSE.

SDP_CLSID_PNPInformation Macro

File

sdp.h

C

```
#define SDP_CLSID_PNPInformation 0x1200
```

Description

This is macro SDP_CLSID_PNPInformation.

SDP_CLSID_PublicBrowseGroup Macro

File

sdp.h

C

```
#define SDP_CLSID_PublicBrowseGroup 0x1002
```

Description

This is macro SDP_CLSID_PublicBrowseGroup.

SDP_CLSID_RFCOMM Macro

File

sdp.h

C

```
#define SDP_CLSID_RFCOMM 0x3
```

Description

This is macro SDP_CLSID_RFCOMM.

SDP_CLSID_SerialPort Macro

File

sdp.h

C

```
#define SDP_CLSID_SerialPort 0x1101
```

Description

This is macro SDP_CLSID_SerialPort.

SDP_CLSID_ServiceDiscoveryServerServiceClassID Macro

File

sdp.h

C

```
#define SDP_CLSID_ServiceDiscoveryServerServiceClassID 0x1000
```

Description

defgroup sdp SDP

This module describe functions and data structures used to start the SDP server and perform SDP queries.

SDP_DATA_TYPE_ALTERNATIVE Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_ALTERNATIVE 7
```

Description

This is macro SDP_DATA_TYPE_ALTERNATIVE.

SDP_DATA_TYPE_BOOL Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_BOOL 5
```

Description

This is macro SDP_DATA_TYPE_BOOL.

SDP_DATA_TYPE_INT Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_INT 2
```

Description

This is macro SDP_DATA_TYPE_INT.

SDP_DATA_TYPE_INT128 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_INT128 0x82
```

Description

This is macro SDP_DATA_TYPE_INT128.

SDP_DATA_TYPE_INT16 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_INT16 0x12
```

Description

This is macro SDP_DATA_TYPE_INT16.

SDP_DATA_TYPE_INT32 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_INT32 0x22
```

Description

This is macro SDP_DATA_TYPE_INT32.

SDP_DATA_TYPE_INT64 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_INT64 0x42
```

Description

This is macro SDP_DATA_TYPE_INT64.

SDP_DATA_TYPE_INT8 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_INT8 SDP_DATA_TYPE_INT
```

Description

This is macro SDP_DATA_TYPE_INT8.

SDP_DATA_TYPE_NIL Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_NIL 0
```

Description

This is macro SDP_DATA_TYPE_NIL.

SDP_DATA_TYPE_SEQUENCE Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_SEQUENCE 6
```

Description

This is macro SDP_DATA_TYPE_SEQUENCE.

SDP_DATA_TYPE_STRING Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_STRING 4
```

Description

This is macro SDP_DATA_TYPE_STRING.

SDP_DATA_TYPE_UINT Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UINT 1
```

Description

This is macro SDP_DATA_TYPE_UINT.

SDP_DATA_TYPE_UINT128 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UINT128 0x81
```

Description

This is macro SDP_DATA_TYPE_UINT128.

SDP_DATA_TYPE_UINT16 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UINT16 0x11
```

Description

This is macro SDP_DATA_TYPE_UINT16.

SDP_DATA_TYPE_UINT32 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UINT32 0x21
```

Description

This is macro SDP_DATA_TYPE_UINT32.

SDP_DATA_TYPE_UINT64 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UINT64 0x41
```

Description

This is macro SDP_DATA_TYPE_UINT64.

SDP_DATA_TYPE_UINT8 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UINT8 SDP_DATA_TYPE_UINT
```

Description

This is macro SDP_DATA_TYPE_UINT8.

SDP_DATA_TYPE_URL Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_URL 8
```

Description

This is macro SDP_DATA_TYPE_URL.

SDP_DATA_TYPE_UUID Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UUID 3
```

Description

This is macro SDP_DATA_TYPE_UUID.

SDP_DATA_TYPE_UUID128 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UUID128 0x23
```

Description

This is macro SDP_DATA_TYPE_UUID128.

SDP_DATA_TYPE_UUID16 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UUID16 SDP_DATA_TYPE_UUID
```

Description

This is macro SDP_DATA_TYPE_UUID16.

SDP_DATA_TYPE_UUID32 Macro

File

sdp.h

C

```
#define SDP_DATA_TYPE_UUID32 0x13
```

Description

This is macro SDP_DATA_TYPE_UUID32.

SDP_ERROR_INSUFFICIENT_RESOURCE Macro

File

sdp.h

C

```
#define SDP_ERROR_INSUFFICIENT_RESOURCE 0x06
```

Description

This is macro SDP_ERROR_INSUFFICIENT_RESOURCE.

SDP_ERROR_INVALID_CONTINUATION_STATE Macro

File

sdp.h

C

```
#define SDP_ERROR_INVALID_CONTINUATION_STATE 0x05
```

Description

This is macro SDP_ERROR_INVALID_CONTINUATION_STATE.

SDP_ERROR_INVALID_PDU_SIZE Macro

File

sdp.h

C

```
#define SDP_ERROR_INVALID_PDU_SIZE 0x04
```

Description

This is macro SDP_ERROR_INVALID_PDU_SIZE.

SDP_ERROR_INVALID_REQUEST_SYNTAX Macro

File

sdp.h

C

```
#define SDP_ERROR_INVALID_REQUEST_SYNTAX 0x03
```

Description

This is macro SDP_ERROR_INVALID_REQUEST_SYNTAX.

SDP_ERROR_INVALID_SDP_VERSION Macro

File

sdp.h

C

```
#define SDP_ERROR_INVALID_SDP_VERSION 0x01
```

Description

This is macro SDP_ERROR_INVALID_SDP_VERSION.

SDP_ERROR_INVALID_SR_HANDLE Macro

File

sdp.h

C

```
#define SDP_ERROR_INVALID_SR_HANDLE 0x02
```

Description

This is macro SDP_ERROR_INVALID_SR_HANDLE.

SDP_ERROR_RESERVED Macro

File

sdp.h

C

```
#define SDP_ERROR_RESERVED 0x00
```

Description

This is macro SDP_ERROR_RESERVED.

SDP_ErrorResponse Macro

File

sdp.h

C

```
#define SDP_ErrorResponse 0x01
```

Description

This is macro SDP_ErrorResponse.

SDP_FTP_SERVICE_ID Macro

File

sdp.h

C

```
#define SDP_FTP_SERVICE_ID 0x1237
```

Description

This is macro SDP_FTP_SERVICE_ID.

SDP_HID_SERVICE_ID Macro

File

sdp.h

C

```
#define SDP_HID_SERVICE_ID 0x1235
```

Description

This is macro SDP_HID_SERVICE_ID.

SDP_HSP_AG_SERVICE_ID Macro

File

sdp.h

C

```
#define SDP_HSP_AG_SERVICE_ID 0x1236
```

Description

This is macro SDP_HSP_AG_SERVICE_ID.

SDP_HSP_HS_SERVICE_ID Macro

File

sdp.h

C

```
#define SDP_HSP_HS_SERVICE_ID 0x1236
```

Description

This is macro SDP_HSP_HS_SERVICE_ID.

SDP_MAX_ATTRIBUTE_PATTERN_LEN Macro

File

sdp.h

C

```
#define SDP_MAX_ATTRIBUTE_PATTERN_LEN 12
```

Description

This is macro SDP_MAX_ATTRIBUTE_PATTERN_LEN.

SDP_MAX_DATA_ELEMENT_LEN Macro

File

sdp.h

C

```
#define SDP_MAX_DATA_ELEMENT_LEN 256
```

Description

This is macro SDP_MAX_DATA_ELEMENT_LEN.

SDP_MAX_DATA_ELEMENTS Macro

File

sdp.h

C

```
#define SDP_MAX_DATA_ELEMENTS 10
```

Description

This is macro SDP_MAX_DATA_ELEMENTS.

SDP_MAX_SEARCH_PATTERN_LEN Macro

File

sdp.h

C

```
#define SDP_MAX_SEARCH_PATTERN_LEN 12
```

Description

This is macro SDP_MAX_SEARCH_PATTERN_LEN.

SDP_MAX_TRANSACTIONS Macro

File

sdp.h

C

```
#define SDP_MAX_TRANSACTIONS 2
```

Description

typedef struct _sdp_db { bt_int count; bt_sdp_service_record_p *records; } bt_sdp_db, *bt_sdp_db_p;

SDP_PDU_HEADER_LEN Macro

File

sdp.h

C

```
#define SDP_PDU_HEADER_LEN 5
```

Description

This is macro SDP_PDU_HEADER_LEN.

SDP_RFCOMM_SERVICE_ID Macro

File

sdp.h

C

```
#define SDP_RFCOMM_SERVICE_ID 0x1234
```

Description

This is macro SDP_RFCOMM_SERVICE_ID.

SDP_ServiceAttributeRequest Macro

File

sdp.h

C

```
#define SDP_ServiceAttributeRequest 0x04
```

Description

This is macro SDP_ServiceAttributeRequest.

SDP_ServiceAttributeResponse Macro

File

sdp.h

C

```
#define SDP_ServiceAttributeResponse 0x05
```

Description

This is macro SDP_ServiceAttributeResponse.

SDP_ServiceSearchAttributeRequest Macro

File

sdp.h

C

```
#define SDP_ServiceSearchAttributeRequest 0x06
```

Description

This is macro SDP_ServiceSearchAttributeRequest.

SDP_ServiceSearchAttributeResponse Macro

File

sdp.h

C

```
#define SDP_ServiceSearchAttributeResponse 0x07
```

Description

This is macro SDP_ServiceSearchAttributeResponse.

SDP_ServiceSearchRequest Macro

File

sdp.h

C

```
#define SDP_ServiceSearchRequest 0x02
```

Description

This is macro SDP_ServiceSearchRequest.

SDP_ServiceSearchResponse Macro

File

sdp.h

C

```
#define SDP_ServiceSearchResponse 0x03
```

Description

This is macro SDP_ServiceSearchResponse.

SDP_SR_HANDLE_HDP_SINK Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_HDP_SINK 0x10009
```

Description

This is macro SDP_SR_HANDLE_HDP_SINK.

SDP_SR_HANDLE_HDP_SOURCE Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_HDP_SOURCE 0x10008
```

Description

This is macro SDP_SR_HANDLE_HDP_SOURCE.

SDP_SR_HANDLE_HFP_HF Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_HFP_HF 0x1000A
```

Description

This is macro SDP_SR_HANDLE_HFP_HF.

SDP_SR_HANDLE_HID Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_HID 0x10001
```

Description

This is macro SDP_SR_HANDLE_HID.

SDP_SR_HANDLE_HID_KEYBOARD Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_HID_KEYBOARD 0x10005
```

Description

This is macro SDP_SR_HANDLE_HID_KEYBOARD.

SDP_SR_HANDLE_HSP_HS Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_HSP_HS 0x10004
```

Description

This is macro SDP_SR_HANDLE_HSP_HS.

SDP_SR_HANDLE_OBEXFileTransfer Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_OBEXFileTransfer 0x10006
```

Description

This is macro SDP_SR_HANDLE_OBEXFileTransfer.

SDP_SR_HANDLE_OBEXObjectPush Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_OBEXObjectPush 0x10007
```

Description

This is macro SDP_SR_HANDLE_OBEXObjectPush.

SDP_SR_HANDLE_PNPINFORMATION Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_PNPINFORMATION 0x10002
```

Description

This is macro SDP_SR_HANDLE_PNPINFORMATION.

SDP_SR_HANDLE_RFCOMM Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_RFCOMM 0x10000
```

Description

This is macro SDP_SR_HANDLE_RFCOMM.

SDP_SR_HANDLE_SERVER Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_SERVER 0x0
```

Description

This is macro SDP_SR_HANDLE_SERVER.

SDP_SR_HANDLE_TEST Macro

File

sdp.h

C

```
#define SDP_SR_HANDLE_TEST 0x10003 // for simulating service search using continuation state
```

Description

for simulating service search using continuation state

bt_sdp_client_evt_connected_t Structure

File

sdp_client.h

C

```
struct bt_sdp_client_evt_connected_t {
  bt_l2cap_channel_t* channel;
};
```

Description

This is record bt_sdp_client_evt_connected_t.

bt_sdp_client_evt_disconnected_t Structure

File

sdp_client.h

C

```
struct bt_sdp_client_evt_disconnected_t {
  bt_l2cap_channel_t* channel;
};
```

Description

This is record bt_sdp_client_evt_disconnected_t.

bt_sdp_client_callback_fp Type

File

sdp_client.h

C

```
typedef void (* bt_sdp_client_callback_fp)(bt_byte evt, void* evt_param, void* param);
```

Description

This is type bt_sdp_client_callback_fp.

bt_sdp_data_element_cp Type

File

sdp.h

C

```
typedef const bt_sdp_data_element_t* bt_sdp_data_element_cp;
```

Description

This is type bt_sdp_data_element_cp.

bt_sdp_data_element_p Type

File

sdp.h

C

```
typedef bt_sdp_data_element_t* bt_sdp_data_element_p;
```

Description

This is type bt_sdp_data_element_p.

bt_sdp_data_element_t Structure

File

sdp.h

C

```
typedef struct _bt_sdp_data_element_t {
  bt_int type;
  bt_int bytecount;
  union {
    bt_ulong init;
    void* pdata;
    char* pstr;
    char* purl;
    struct _bt_sdp_sequence_t* pseq;
    bt_byte b;
    bt_int i;
    bt_uint ui;
    bt_long l;
    bt_ulong ul;
    bt_uuid16 uuid16;
    bt_uuid32 uuid32;
    bt_uuid_p uuid128;
  } data;
} bt_sdp_data_element_t;
```

Description

This is type bt_sdp_data_element_t.

bt_sdp_found_attr_list_t Structure

File

sdp.h

C

```
typedef struct _bt_sdp_found_attr_list_t {
  bt_int found_attr_count;
  const bt_byte** found_attr_list;
} bt_sdp_found_attr_list_t;
```

Members

Members	Description
const bt_byte** found_attr_list;	SDP_MAX_ATTRIBUTE_RESULT_LEN];

Description

This is type bt_sdp_found_attr_list_t.

bt_sdp_packet_t Structure

File

sdp_packet.h

C

```
typedef struct _bt_sdp_packet_t {
  bt_packet_t header;
  bt_byte sdu_type;
  union {
    union {
      struct _bt_sdp_service_transaction_t* service;
      struct _bt_sdp_transaction_t* attribute;
    } transaction;
    struct {
      bt_int trans_id;
      bt_int err_code;
    } error;
    struct {
      bt_int trans_id;
      struct _bt_sdp_data_element_t* pattern;
      bt_int uuid_index;
      bt_int uuid_pos;
    } service_search;
    struct {
      bt_int trans_id;
      struct _bt_sdp_data_element_t* pattern;
      bt_long sr;
    } attribute_search;
  } data;
} bt_sdp_packet_t;
```

Description

This is type bt_sdp_packet_t.

bt_sdp_read_de_callback_ftp Type

File

sdp_client.h

C

```
typedef bt_bool (* bt_sdp_read_de_callback_ftp)(bt_sdp_data_element_p pde, bt_bool seq_start);
```

Description

This is type bt_sdp_read_de_callback_ftp.

bt_sdp_sequence_cp Type

File

sdp.h

C

```
typedef const bt_sdp_sequence_t* bt_sdp_sequence_cp;
```

Description

This is type bt_sdp_sequence_cp.

bt_sdp_sequence_p Type

File

sdp.h

C

```
typedef bt_sdp_sequence_t* bt_sdp_sequence_p;
```

Description

This is type bt_sdp_sequence_p.

bt_sdp_sequence_t Structure

File

sdp.h

C

```
typedef struct _bt_sdp_sequence_t {
  bt_int count;
  bt_sdp_data_element_p elements;
} bt_sdp_sequence_t;
```

Description

This is type bt_sdp_sequence_t.

bt_sdp_serialization_state_p Structure

File

sdp.h

C

```
typedef struct _bt_sdp_serialization_state_t {
  bt_int listIndex;
  bt_int attrIndex;
  bt_byte attrPart;
  bt_int attrPartPos;
  bt_int attr_bytes;
  bt_int first_list;
  bt_int last_list;
  bt_int first_attr;
  bt_int last_attr;
  bt_uint attr_lists_seq_len;
  bt_bool write_seq_len;
} bt_sdp_serialization_state_t, * bt_sdp_serialization_state_p;
```

Description

This is type bt_sdp_serialization_state_p.

bt_sdp_serialization_state_t Structure

File

sdp.h

C

```
typedef struct _bt_sdp_serialization_state_t {
  bt_int listIndex;
  bt_int attrIndex;
  bt_byte attrPart;
  bt_int attrPartPos;
  bt_int attr_bytes;
  bt_int first_list;
  bt_int last_list;
  bt_int first_attr;
  bt_int last_attr;
  bt_uint attr_lists_seq_len;
  bt_bool write_seq_len;
} bt_sdp_serialization_state_t, * bt_sdp_serialization_state_p;
```

Description

This is type bt_sdp_serialization_state_t.

bt_sdp_server_attribute_t Structure

File

sdp_private.h

C

```
typedef struct _bt_sdp_server_attribute_t {
  bt_uint attr_id;
  bt_uint data_len;
  const bt_byte* data;
  bt_sdp_server_data_element_t attr_value;
} bt_sdp_server_attribute_t;
```

Description

This is type bt_sdp_server_attribute_t.

bt_sdp_server_data_element_t Structure

File

sdp_private.h

C

```
typedef struct _bt_sdp_server_data_element_t {
  bt_byte type;
  bt_uint data_len;
  const bt_byte* data;
} bt_sdp_server_data_element_t;
```

Description

Private types

bt_sdp_server_record_t Structure

File

sdp_private.h

C

```
typedef struct _bt_sdp_server_record_t {
  bt_sr_handle_t h;
```

```
  bt_uint data_len;
  const bt_byte* data;
} bt_sdp_server_record_t;
```

Description

This is type bt_sdp_server_record_t.

bt_sdp_service_attribute_callback_fp Type

File

sdp_client.h

C

```
typedef bt_bool (* bt_sdp_service_attribute_callback_fp)(bt_uint attrid, bt_sdp_data_element_p pde, void*
param);
```

Description

This is type bt_sdp_service_attribute_callback_fp.

bt_sdp_service_search_callback_fp Type

File

sdp_client.h

C

```
typedef void (* bt_sdp_service_search_callback_fp)(bt_sr_handle_p shlist, bt_int count, void* param);
```

Description

This is type bt_sdp_service_search_callback_fp.

bt_sdp_service_transaction_p Structure

File

sdp.h

C

```
typedef struct _bt_sdp_service_transaction_t {
  bt_int id;
  bt_int found_sr_count;
  bt_sr_handle_t* found_sr_list;
  bt_uint max_sr_count;
  bt_int next_tran_index;
} bt_sdp_service_transaction_t, * bt_sdp_service_transaction_p;
```

Members

Members	Description
bt_sr_handle_t* found_sr_list;	SDP_MAX_SEARCH_RESULT_LEN];

Description

This is type bt_sdp_service_transaction_p.

bt_sdp_service_transaction_t Structure

File

sdp.h

C

```
typedef struct _bt_sdp_service_transaction_t {
  bt_int id;
  bt_int found_sr_count;
  bt_sr_handle_t* found_sr_list;
```

```
  bt_uint max_sr_count;
  bt_int next_tran_index;
} bt_sdp_service_transaction_t, * bt_sdp_service_transaction_p;
```

Members

Members	Description
bt_sr_handle_t* found_sr_list;	SDP_MAX_SEARCH_RESULT_LEN];

Description

This is type bt_sdp_service_transaction_t.

bt_sdp_transaction_t Structure

File

sdp.h

C

```
typedef struct _bt_sdp_transaction_t {
  bt_int id;
  bt_int found_list_count;
  bt_sdp_found_attr_list_t* found_attr_lists;
  bt_uint max_bytecount;
  bt_bool complete;
  bt_bool continuation;
  bt_sdp_serialization_state_t sr_state;
} bt_sdp_transaction_t;
```

Members

Members	Description
bt_sdp_found_attr_list_t* found_attr_lists;	SDP_MAX_SEARCH_RESULT_LEN];

Description

This is type bt_sdp_transaction_t.

bt_sr_handle_p Type

File

sdp.h

C

```
typedef bt_long * bt_sr_handle_p;
```

Description

This is type bt_sr_handle_p.

bt_sr_handle_t Type

File

sdp.h

C

```
typedef bt_long bt_sr_handle_t;
```

Description

This is type bt_sr_handle_t.

SDP_CLIENT_EVT_CONNECTION_FAILED Macro

File

sdp_client.h

C

```
#define SDP_CLIENT_EVT_CONNECTION_FAILED 3
```

Description

This is macro SDP_CLIENT_EVT_CONNECTION_FAILED.

SDP_ATTRID_HCRP_1284ID Macro

File

sdp.h

C

```
#define SDP_ATTRID_HCRP_1284ID 0x300
```

Description

This is macro SDP_ATTRID_HCRP_1284ID.

SDP_ATTRID_HCRP_DeviceLocation Macro

File

sdp.h

C

```
#define SDP_ATTRID_HCRP_DeviceLocation 0x306
```

Description

This is macro SDP_ATTRID_HCRP_DeviceLocation.

SDP_ATTRID_HCRP_DeviceName Macro

File

sdp.h

C

```
#define SDP_ATTRID_HCRP_DeviceName 0x302
```

Description

This is macro SDP_ATTRID_HCRP_DeviceName.

SDP_ATTRID_HCRP_FriendlyName Macro

File

sdp.h

C

```
#define SDP_ATTRID_HCRP_FriendlyName 0x304
```

Description

This is macro SDP_ATTRID_HCRP_FriendlyName.

SDP_CLSID_HARD_COPY_CABLE_REPLACEMENT Macro

File

sdp.h

C

```
#define SDP_CLSID_HARD_COPY_CABLE_REPLACEMENT 0x1125
```

Description

This is macro SDP_CLSID_HARD_COPY_CABLE_REPLACEMENT.

SDP_CLSID_HARD_COPY_CONTROL_CHANNEL Macro

File

sdp.h

C

```
#define SDP_CLSID_HARD_COPY_CONTROL_CHANNEL 0x0012
```

Description

This is macro SDP_CLSID_HARD_COPY_CONTROL_CHANNEL.

SDP_CLSID_HARD_COPY_DATA_CHANNEL Macro

File

sdp.h

C

```
#define SDP_CLSID_HARD_COPY_DATA_CHANNEL 0x0014
```

Description

This is macro SDP_CLSID_HARD_COPY_DATA_CHANNEL.

SDP_CLSID_HARD_COPY_NOTIFICATION Macro

File

sdp.h

C

```
#define SDP_CLSID_HARD_COPY_NOTIFICATION 0x0016
```

Description

This is macro SDP_CLSID_HARD_COPY_NOTIFICATION.

SDP_CLSID_HCR_PRINT Macro

File

sdp.h

C

```
#define SDP_CLSID_HCR_PRINT 0x1126
```

Description

This is macro SDP_CLSID_HCR_PRINT.

SDP_CLSID_HCR_SCAN Macro

File

sdp.h

C

```
#define SDP_CLSID_HCR_SCAN 0x1127
```

Description

This is macro SDP_CLSID_HCR_SCAN.

SPP Functions

bt_spp_allocate Function

File

spp.h

C

```
bt_spp_port_t* bt_spp_allocate(bt_l2cap_mgr_t* l2cap_mgr, bt_spp_state_callback_fp callback, void*
callback_param);
```

Description

- Allocate a serial port.

The returned serial port is initially in the ::SPP_PORT_STATE_DISCONNECTED state. To establish a connection with a remote device, call bt_spp_connect(). To listen for incoming connections from other devices, call bt_spp_listen(). The p callback parameter must specify a callback function that will be used to notify about serial port events and state changes. When the port is not needed any more it must be deallocated by bt_spp_deallocate(). The maximum number of serial ports that can be allocated simultaneously is specified by the ::SPP_MAX_PORTS configuration parameter.

@param l2cap_mgr L2CAP manager.

@param callback Pointer to a callback function used to notify about serial port events. Cannot be NULL.

@param callback_param An arbitrary pointer that is passed as a parameter to the callback function.

@return A pointer to the ::bt_spp_port_t structure. Returns NULL if the maximum number of ports has been already allocated or the c callback parameter is NULL.

bt_spp_cancel_receive Function

File

spp.h

C

```
void bt_spp_cancel_receive(bt_spp_port_t* port);
```

Description

- Cancel receive data.

If a receive operation is currently in progress this function will cancel it. After calling this function the receive callback specified in bt_spp_receive() will not be called.

If there is no receive operation in progress calling this function has no effect.

@param port Serial port.

bt_spp_cancel_send Function

File

spp.h

C

```
void bt_spp_cancel_send(bt_spp_port_t* port);
```

Description

- Cancel send data.

If a send operation is currently in progress this function will try to cancel it. When the operation is canceled the send callback function will be called with the ::SPP_SEND_STATUS_CANCELED status.

If this function is called but the active send operation completes successfuly before the stack can actually cancel it the call back function will still be called with the ::SPP_SEND_STATUS_CANCELED status.

If there is no send operation in progess calling this function has no effect.

@param port Serial port.

bt_spp_clr_port_options Function

File

spp.h

C

```
void bt_spp_clr_port_options(bt_spp_port_t* port, bt_uint options);
```

Description

This is function bt_spp_clr_port_options.

bt_spp_connect Function

File

spp.h

C

```
bt_bool bt_spp_connect(bt_spp_port_t* port, bt_bdaddr_p remote_addr, bt_byte channel);
```

Description

- Connect to a remote device.

This function initiates a connection to a remote device. When the connection is successfully established the port's callback is called with the ::SPP_PORT_EVENT_CONNECTED event. If connection fails the callback is called with the ::SPP_PORT_EVENT_CONNECTION_FAILED event.

The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail.

@param port Serial port. @param remote_addr Bluetooth address of the remote device. @param channel RFCOMM server channel on which the connection is to be established.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_deallocate Function

File

spp.h

C

```
bt_bool bt_spp_deallocate(bt_spp_port_t* port);
```

Description

- Deallocate serial port.

This function deallocates the specified port structure and other resources associated with it.

The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail.

If the function completes successfully the application must not try to access any fields in the structure and must not use it with any other SPP functions. Also, it becomes available for subsequent allocation by bt_spp_port_allocate().

@param port Serial port structure to deallocate.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_disconnect Function

File

spp.h

C

```
void bt_spp_disconnect(bt_spp_port_t* port);
```

Description

- Disconnect from the remote device.

This function initiates the disconnection process. When it is complete the the port's callback is called with the SPP_PORT_EVENT_DISCONNECTED event.

If the port is already in the disconnected state the function does nothing and the callback is not called.

@param port Serial port.

bt_spp_find_server Function

File

spp.h

C

```
bt_bool bt_spp_find_server(bt_bdaddr_t* deviceAddress, const bt_uuid_t* service_class_id,
bt_spp_find_server_callback_fp callback, void* callback_param);
```

Description

This is function bt_spp_find_server.

bt_spp_get_frame_length Function

File

spp.h

C

```
bt_int bt_spp_get_frame_length(bt_spp_port_t* port);
```

Description

- Get frame length.

This function returns the RFCOMM frame length used by the RFCOMM protocol. The frame length depends on configuration of DotStack and configuration of the Bluetooth stack running on the remote device. In order to achieve maximum throughput over the serial port connection the application should send and receive data in chunks that are multiple of this frame length.

@return RFCOMM frame length in bytes.

bt_spp_get_hci_connection Function

File

spp.h

C

```
bt_hci_conn_state_t* bt_spp_get_hci_connection(const bt_spp_port_t* port);
```

Description

- Get SPP port's ACL connection.

This function returns a pinter to the structure that describes the ACL connection this port is on.

@return Pointer to ACL connection description if the port is connected, NULL otherwise.

bt_spp_get_local_modem_status Function

File

spp.h

C

```
bt_byte bt_spp_get_local_modem_status(const bt_spp_port_t* port);
```

Description

- Get local device's TS 07.10 control signals.

This function returns current state of the local device's TS 07.10 controls signals. The signals are defined as a mask of the following constants: SPP_RS232_DSR SPP_RS232_RTS SPP_RS232_RI SPP_RS232_DCD

@param port Serial port.

@return local device's TS 07.10 control signals.

bt_spp_get_remote_address Function

File

spp.h

C

```
bt_bdaddr_t* bt_spp_get_remote_address(const bt_spp_port_t* port);
```

Description

brief Get the address of the remote device this device is connected to. ingroup spp

param port Serial port.

return li c A pointer to bt_bdaddr structure that contains the address of the remote device.

bt_spp_get_remote_modem_status Function

File

spp.h

C

```
bt_byte bt_spp_get_remote_modem_status(const bt_spp_port_t* port);
```

Description

- Get remote device's TS 07.10 control signals.

This function returns current state of the remote device's V.24 controls signals. The signals are defined as a mask of the following constants: SPP_RS232_DTR SPP_RS232_CTS SPP_RS232_RI SPP_RS232_DCD

@param port Serial port.

@return remote device's TS 07.10 control signals.

bt_spp_init Function

File

spp.h

C

```
void bt_spp_init();
```

Description

- Initialize the SPP module.

This function initializes all internal variables of the SPP module. It must be called prior to using any other functions in this module.

bt_spp_listen Function

File

spp.h

C

```
bt_bool bt_spp_listen(bt_spp_port_t* port, bt_byte channel);
```

Description

- Listen for incoming connections.

This function registers the port to accept incoming connections from remote devices on a particular RFCOMM server channel. The specified server channel should be listed in the SDP database. Otherwise, remote devices will not be able to find out which server channel to use.

When a remote device successfully establishes a connection on the specified port the port's callback is called with the ::SPP_PORT_EVENT_CONNECTED event.

The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail.

@param port Serial port. @param channel The RFCOMM server channel on which to listen for connections.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_receive Function

File

spp.h

C

```
bt_bool bt_spp_receive(bt_spp_port_t* port, void* buffer, bt_int buffer_len, bt_spp_receive_callback_fp
callback);
```

Description

- Receive data.

This function receives data from the serial port connection. The caller must provide a buffer and a callback function. Whenever the port receives data they are copied to the provided buffer and the callback function is called. The callback function is passed the length of received data and the same callback parameter that was specified when the port was allocated with bt_spp_allocate(). This function does not wait until the buffer is filled out completely. Any amount of received data will complete the operation.

The port must be in ::SPP_PORT_STATE_CONNECTED state. Otherwise, the function will fail. Also, the function will fail if a previously started receive operation is still in progress.

@param port Serial port. @param data Pointer to the data to be sent. @param data_len Length of the data. @param callback Send callback function.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_send Function

File

spp.h

C

```
bt_bool bt_spp_send(bt_spp_port_t* port, const void* data, bt_ulong data_len, bt_spp_send_callback_fp
callback);
```

Description

- Send data.

This function starts sending data over the serial port connection. Along with the data the caller must provide a callback function that is called when all data has been sent. Also, during execution of this operation the port's state callback function is called periodically with the ::SPP_PORT_EVENT_SEND_PROGRESS event.

The port must be in ::SPP_PORT_STATE_CONNECTED state. Otherwise, the function will fail. Also, the function will fail if a previously started send operation is still in progress.

The callback function is passed the same callback parameter that was specified when the port was allocated with bt_spp_allocate().

@param port Serial port. @param data Pointer to the data to be sent. @param data_len Length of the data. @param callback Send callback function.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_set_dtr Function

File

spp.h

C

```
bt_byte bt_spp_set_dtr(bt_spp_port_t* port, bt_bool on);
```

Description

- Set local device's RS-232 DTR signal.

Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM_MODEM_STATUS_RTC signal.

If there are resources to send a command to the remote device this command will return TRUE.

If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported.

If the status cannot be changed because there is no resources to send a command to the remote device this function will return FALSE and no events will be reported.

@param port Serial port. @param on Value indicating weather to set or clear the signal.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_set_local_modem_status Function

File

spp.h

C

```
bt_byte bt_spp_set_local_modem_status(bt_spp_port_t* port, bt_byte ms);
```

Description

- Set local device's TS 07.10 control signals.

Changes the state of local device's TS 07.10 control signals and notifies the remote device of the change.

If there are resources to send a command to the remote device this command will return TRUE.

If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported.

If the status cannot be changed because there is no resources to send a command to the remote device this function will return FALSE and no events will be reported.

@param port Serial port. @param ms Signals mask.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_set_port_options Function

File

spp.h

C

```
void bt_spp_set_port_options(bt_spp_port_t* port, bt_uint options);
```

Description

This is function bt_spp_set_port_options.

bt_spp_set_rts Function

File

spp.h

C

```
bt_byte bt_spp_set_rts(bt_spp_port_t* port, bt_bool on);
```

Description

- Set local device's RS-232 RTS signal.

Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM_MODEM_STATUS_RTR signal.

If there are resources to send a command to the remote device this command will return TRUE.

If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported.

If the status cannot be changed because there is no resources to send a command to the remote device this function will return FALSE and no events will be reported.

@param port Serial port. @param on Value indicating weather to set or clear the signal.

@return c TRUE if successful, c FALSE otherwise.

_bt_spp_find_port Function

File

spp_private.h

C

```
bt_spp_port_t* _bt_spp_find_port(bt_rfcomm_dlc_t* dlc);
```

Description

This is function _bt_spp_find_port.

_bt_spp_handle_rx Function

File

spp_private.h

C

```
void _bt_spp_handle_rx(bt_spp_port_t* port);
```

Description

This is function _bt_spp_handle_rx.

_bt_spp_handle_tx Function

File

spp_private.h

C

```
void _bt_spp_handle_tx(bt_spp_port_t* port);
```

Description

This is function _bt_spp_handle_tx.

_bt_spp_rfcomm_read_data_callback Function

File

spp_private.h

C

```
void _bt_spp_rfcomm_read_data_callback(bt_rfcomm_dlc_p pdlc, bt_byte_p pdata, bt_int len);
```

Description

This is function _bt_spp_rfcomm_read_data_callback.

_bt_spp_rfcomm_send_data_callback Function

File

spp_private.h

C

```
void _bt_spp_rfcomm_send_data_callback(bt_rfcomm_dlc_t* dlc, bt_byte* data, bt_int len, bt_int status);
```

Description

This is function _bt_spp_rfcomm_send_data_callback.

SPP Data Types and Constants

SPP_PORT_OPTION_ENCRYPTED Macro

File

spp.h

C

```
#define SPP_PORT_OPTION_ENCRYPTED 0x0002
```

Description

This is macro SPP_PORT_OPTION_ENCRYPTED.

SPP_PORT_OPTION_MASTER Macro

File

spp.h

C

```
#define SPP_PORT_OPTION_MASTER 0x0004
```

Description

This is macro SPP_PORT_OPTION_MASTER.

SPP_PORT_OPTION_SECURE Macro

File

spp.h

C

```
#define SPP_PORT_OPTION_SECURE 0x0001
```

Description

This is macro SPP_PORT_OPTION_SECURE.

SPP_PORT_TYPE_INCOMING Macro

File

spp.h

C

```
#define SPP_PORT_TYPE_INCOMING 0
```

Description

This is macro SPP_PORT_TYPE_INCOMING.

SPP_PORT_TYPE_OUTGOING Macro

File

spp.h

C

```
#define SPP_PORT_TYPE_OUTGOING 1
```

Description

This is macro SPP_PORT_TYPE_OUTGOING.

SPP_RS232_CTS Macro

File

spp.h

C

```
#define SPP_RS232_CTS RFCOMM_MODEM_STATUS_RTR
```

Description

This is macro SPP_RS232_CTS.

SPP_RS232_DCD Macro

File

spp.h

C

```
#define SPP_RS232_DCD RFCOMM_MODEM_STATUS_DV
```

Description

This is macro SPP_RS232_DCD.

SPP_RS232_DSR Macro

File

spp.h

C

```
#define SPP_RS232_DSR RFCOMM_MODEM_STATUS_RTC
```

Description

- @defgroup spp Serial Port Profile (SPP)

The DotStack SPP API is a simple API for communicating over a Bluetooth link using the Bluetooth Serial Port Profile.

Here are the steps for using this API:

- Call the bt_spp_init() function.
- Allocate a serial port structure with bt_spp_allocate(). One of the parameters to

this function is a pointer to a callback function. That callback function will be called by the stack whenever the state of the serial port changes.

- To connect to a remote device call bt_spp_connect(). The stack will notify when the connection is established by calling the state callback function.
- To wait for a connection from a remote device call bt_spp_listen(). The stack will notify when the connection is established by calling the state callback function.
- When the port is connected you can send data with bt_spp_send() and receive data with bt_spp_receive().
- To terminate the connection call bt_spp_disconnect().
- When you are finished using the port deallocate it with bt_spp_deallocate().

SPP_RS232_DTR Macro

File

spp.h

C

```
#define SPP_RS232_DTR RFCOMM_MODEM_STATUS_RTC
```

Description

This is macro SPP_RS232_DTR.

SPP_RS232_RI Macro

File

spp.h

C

```
#define SPP_RS232_RI RFCOMM_MODEM_STATUS_IC
```

Description

This is macro SPP_RS232_RI.

SPP_RS232_RTS Macro

File

spp.h

C

```
#define SPP_RS232_RTS RFCOMM_MODEM_STATUS_RTR
```

Description

This is macro SPP_RS232_RTS.

bt_spp_find_server_callback_fp Type

File

spp.h

C

```
typedef void (* bt_spp_find_server_callback_fp)(bt_byte server_channel, bt_bool found, void* param);
```

Description

This is type bt_spp_find_server_callback_fp.

bt_spp_port_event_e Enumeration

File

spp.h

C

```
typedef enum _bt_spp_port_event_e {
  SPP_PORT_EVENT_CONNECTED = 1,
  SPP_PORT_EVENT_DISCONNECTED,
  SPP_PORT_EVENT_CONNECTION_FAILED,
  SPP_PORT_EVENT_SEND_PROGRESS,
  SPP_PORT_EVENT_REMOTE_MODEM_STATUS_CHANGED,
  SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED,
  SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED
} bt_spp_port_event_e;
```

Members

Members	Description
SPP_PORT_EVENT_CONNECTED = 1	• Connection with a remote device was successfully established. This event is reported independent of which side (local or remote) initiated the connection.
SPP_PORT_EVENT_DISCONNECTED	• Connection with the remote device was terminated. This event is reported independent of which side (local or remote) initiated termination of the connection.
SPP_PORT_EVENT_CONNECTION_FAILED	• Connection to a remote device failed. This event is reported when an attempt to establish a connection using bt_spp_connect() has failed.
SPP_PORT_EVENT_SEND_PROGRESS	• Send operation progress. This event is reported periodically during sending data over the serial port connection. It can be used by the application to track send operation progress.
SPP_PORT_EVENT_REMOTE_MODEM_STATUS_CHANGED	• Remote modem status. This event is reported when a remote device notifies the local device about the status of its V.24 control signals. The actual value of the signals can be obtained by calling bt_spp_get_remote_modem_status(const bt_spp_port_t* port)

SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED	• Local modem status. This event is reported when the local device has successfully sent the state of its V.24 signals to the remote party.
SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED	• Local modem status. This event is reported when the local device failed to send the state of its V.24 signals to the remote party.

Description

- Serial port event.

Values of this enumeration represent serial port events. Events are reported to the application through a callback function. The callback function is specified when the port is allocated using bt_spp_allocate().

bt_spp_port_state_e Enumeration

File

spp.h

C

```
typedef enum _bt_spp_port_state_e {
  SPP_PORT_STATE_FREE,
  SPP_PORT_STATE_DISCONNECTED,
  SPP_PORT_STATE_DISCONNECTING,
  SPP_PORT_STATE_CONNECTING,
  SPP_PORT_STATE_CONNECTED
} bt_spp_port_state_e;
```

Members

Members	Description
SPP_PORT_STATE_FREE	*< Used internally to mark a port structure as available for allocation. The application should never encounter a port in this state.
SPP_PORT_STATE_DISCONNECTED	< Port is not connected.
SPP_PORT_STATE_DISCONNECTING	< Disconnecting from remote device.
SPP_PORT_STATE_CONNECTING	< Connecting to a remote device.
SPP_PORT_STATE_CONNECTED	< Port is connected to a remote device.

Description

- Serial port state.

Values of this enumeration represent states of the serial port.

bt_spp_port_t Type

File

spp.h

C

```
typedef struct _bt_spp_port_t bt_spp_port_t;
```

Description

This is type bt_spp_port_t.

bt_spp_receive_callback_fp Type

File

spp.h

C

```
typedef void (* bt_spp_receive_callback_fp)(bt_spp_port_t* port, bt_int bytes_received, void* param);
```

Description

- Serial port receive callback.

This callback function is called when a receive operation initiated by bt_spp_receive() completes.

@param port Serial port on which the receive operation completed.

@param bytes_received Number of received bytes.

@param param Callback parameter that was specified when bt_spp_allocate() was called.

bt_spp_send_callback_fp Type

File

spp.h

C

```
typedef void (* bt_spp_send_callback_fp)(bt_spp_port_t* port, bt_ulong bytes_sent, bt_spp_send_status_e
status, void* param);
```

Description

- Serial port send callback.

This callback function is called when a send operation initiated by bt_spp_send() completes.

@param port Serial port on which the send operation completed.

@param bytes_sent Number of bytes sent. This parameter is just a convenience as it always specifies the same number of bytes that was passed to bt_spp_send();

@param status Completion status. It is one of the values defined in the ::bt_spp_send_status_e enumeration.

@param param Callback parameter that was specified when bt_spp_allocate() was called.

bt_spp_send_status_e Enumeration

File

spp.h

C

```
typedef enum _bt_spp_send_status_e {
  SPP_SEND_STATUS_SUCCESS = 0,
  SPP_SEND_STATUS_TIMEOUT,
  SPP_SEND_STATUS_NOT_ENOUGH_RESOURCES,
  SPP_SEND_STATUS_CANCELED,
  SPP_SEND_STATUS_INTERRUPTED
} bt_spp_send_status_e;
```

Members

Members	Description
SPP_SEND_STATUS_SUCCESS = 0	< The operation completed successfully.
SPP_SEND_STATUS_TIMEOUT	< The operation timed out.
SPP_SEND_STATUS_NOT_ENOUGH_RESOURCES	< There was not enough resources to complete the operation.
SPP_SEND_STATUS_CANCELED	< The operation was canceled.
SPP_SEND_STATUS_INTERRUPTED	< The operation was interrupted because the connection closed.

Description

Send operation status.

bt_spp_state_callback_fp Type

File

spp.h

C

```
typedef void (* bt_spp_state_callback_fp)(bt_spp_port_t* port, bt_spp_port_event_e evt, void* param);
```

Description

- Serial port state callback.

This callback function is called whenever the state of a serial port is changed.

@param port Serial port which state has changed.

@param evt Event describing the nature of state change. It is one of the values defined in the ::bt_spp_port_event_e enumeration.

@param param Callback parameter that was specified when bt_spp_allocate() was called.

SSP Functions

bt_ssp_evt_handler Function

File

ssp.h

C

```
void bt_ssp_evt_handler(bt_hci_event_t* evt);
```

Description

This is function bt_ssp_evt_handler.

bt_ssp_init Function

File

ssp.h

C

```
void bt_ssp_init();
```

Description

This is function bt_ssp_init.

bt_ssp_read_local_oob_data Function

File

ssp.h

C

```
bt_bool bt_ssp_read_local_oob_data(bt_spp_read_local_oob_data_callback_fp callback);
```

Description

This is function bt_ssp_read_local_oob_data.

bt_ssp_send_keypress_notification Function

File

ssp.h

C

```
bt_bool bt_ssp_send_keypress_notification(bt_bdaddr_p bdaddr_remote, SSP_KEYPRESS_NOTIFICATION_TYPE
keypress_type);
```

Description

This is function bt_ssp_send_keypress_notification.

bt_ssp_send_user_confirmation Function

File

ssp.h

C

```
bt_bool bt_ssp_send_user_confirmation(bt_byte status, bt_ssp_user_confirmation_request* ucr,
bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_ssp_send_user_confirmation.

bt_ssp_send_user_passkey Function

File

ssp.h

C

```
bt_bool bt_ssp_send_user_passkey(bt_byte status, bt_ssp_user_passkey_request* upkr, bt_hci_cmd_callback_fp
cb);
```

Description

This is function bt_ssp_send_user_passkey.

bt_ssp_set_io_capabilities Function

File

ssp.h

C

```
bt_bool bt_ssp_set_io_capabilities(bt_byte status, bt_ssp_io_capability* io_caps, bt_hci_cmd_callback_fp
cb);
```

Description

This is function bt_ssp_set_io_capabilities.

bt_ssp_set_mode Function

File

ssp.h

C

```
bt_bool bt_ssp_set_mode(SSP_MODE mode, bt_hci_cmd_callback_fp callback);
```

Description

This is function bt_ssp_set_mode.

bt_ssp_set_oob_data Function

File

ssp.h

C

```
bt_bool bt_ssp_set_oob_data(bt_byte status, bt_ssp_oob_data* oob_data, bt_hci_cmd_callback_fp cb);
```

Description

This is function bt_ssp_set_oob_data.

ssp_evt_io_capability_request Function

File

ssp_event_handler.h

C

```
void ssp_evt_io_capability_request(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_io_capability_request.

ssp_evt_io_capability_response Function

File

ssp_event_handler.h

C

```
void ssp_evt_io_capability_response(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_io_capability_response.

ssp_evt_keypress_notification Function

File

ssp_event_handler.h

C

```
void ssp_evt_keypress_notification(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_keypress_notification.

ssp_evt_oob_data_request Function

File

ssp_event_handler.h

C

```
void ssp_evt_oob_data_request(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_oob_data_request.

ssp_evt_ssp_complete Function

File

ssp_event_handler.h

C

```
void ssp_evt_ssp_complete(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_ssp_complete.

ssp_evt_user_confirmation_request Function

File

ssp_event_handler.h

C

```
void ssp_evt_user_confirmation_request(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_user_confirmation_request.

ssp_evt_user_passkey_notification Function

File

ssp_event_handler.h

C

```
void ssp_evt_user_passkey_notification(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_user_passkey_notification.

ssp_evt_user_passkey_request Function

File

ssp_event_handler.h

C

```
void ssp_evt_user_passkey_request(bt_hci_event_t* evt);
```

Description

This is function ssp_evt_user_passkey_request.

_bt_spp_client_init Function

File

spp_private.h

C

```
void _bt_spp_client_init();
```

Description

This is function _bt_spp_client_init.

bt_spp_cancel_listen Function

File

spp.h

C

```
bt_bool bt_spp_cancel_listen(bt_spp_port_t* port, bt_byte server_channel);
```

Description

- Stop listening for incoming connections.

This function stops the port to accept incoming connections on a given server channel.

@param port Serial port. @param channel The RFCOMM server channel to accept connections on.

@return c TRUE if successful, c FALSE otherwise.

bt_spp_find_server_ex Function

File

spp.h

C

```
bt_bool bt_spp_find_server_ex(bt_bdaddr_t* deviceAddress, const bt_uuid_t* service_class_id, bt_bool
force_hci_disconnect, bt_spp_find_server_callback_fp callback, bt_sdp_client_callback_fp client_callback,
void* callback_param);
```

Description

This is function bt_spp_find_server_ex.

SSP Data Types and Constants

OOB_DATA_HASH_LENGTH Macro

File

ssp.h

C

```
#define OOB_DATA_HASH_LENGTH 16
```

Description

Remote OOB Data Request Event

OOB_DATA_RANDOMIZER_LENGTH Macro

File

ssp.h

C

```
#define OOB_DATA_RANDOMIZER_LENGTH 16
```

Description

This is macro OOB_DATA_RANDOMIZER_LENGTH.

SSP_MAX_MANAGERS Macro

File

ssp.h

C

```
#define SSP_MAX_MANAGERS 1
```

Description

This is macro SSP_MAX_MANAGERS.

_bt_spp_port_t Structure

File

spp.h

C

```
struct _bt_spp_port_t {
  bt_byte state;
  bt_byte type;
```

```
  bt_byte server_channel;
  bt_rfcomm_dlc_t* rfcomm_dlc;
  bt_byte cur_local_ms;
  bt_ulong listening_server_channels;
  struct _bt_spp_port_flags_t {
    unsigned int sig_tx : 1;
    unsigned int sig_rx : 1;
    unsigned int sig_state : 1;
    unsigned int sig_credit : 1;
    unsigned int open : 1;
    unsigned int listening : 1;
    unsigned int secure : 1;
    unsigned int encrypted : 1;
    unsigned int master : 1;
  } flags;
  bt_signal_t signal;
  bt_spp_state_callback_fp state_cb;
  bt_spp_receive_callback_fp receive_cb;
  bt_spp_send_callback_fp send_cb;
  void* cb_param;
  bt_byte* rx_buffer;
  bt_int rx_buffer_len;
  bt_byte* tx_data;
  bt_ulong tx_data_len;
  bt_ulong tx_remaining_len;
  bt_bool tx_cancel;
  bt_byte remaining_connect_attemtps;
  bt_bdaddr_t connect_address;
  bt_rfcomm_session_listener_t rfcomm_session_listener;
  bt_byte* frame_buffers;
  bt_int* frame_len;
  bt_byte read_frame_index;
  bt_byte write_frame_index;
  bt_int read_frame_pos;
};
```

Members

Members	Description
bt_byte state;	• Port state. The field is set to one of the values defined in the ::bt_spp_port_state_e enumeration. This field must never be modified by the application.

Description

- Serial port structure.

This structure represents a Bluetooth serial port. Application code may only use those fields that are documented. The rest of the fields are private to the SPP implementation.

bt_spp_read_local_oob_data_callback_fp Type

File

ssp.h

C

```
typedef void (* bt_spp_read_local_oob_data_callback_fp)(bt_byte status, bt_ssp_oob_data* oob_data, void*
init_param);
```

Description

This is type bt_spp_read_local_oob_data_callback_fp.

bt_ssp_io_capability Structure

File

ssp.h

C

```
typedef struct _bt_ssp_io_capability {
  bt_bdaddr_t bdaddr_remote;
```

```
  SSP_IO_CAPABILITY io_capability;
  SSP_OOB_DATA_PRESENT oob_data_present;
  SSP_AUTHENTICATION_REQUIREMENTS authentication_requirements;
} bt_ssp_io_capability;
```

Description

This is type bt_ssp_io_capability.

bt_ssp_keypress_notification Structure

File

ssp.h

C

```
typedef struct _bt_ssp_keypress_notification {
  bt_bdaddr_t bdaddr_remote;
  SSP_KEYPRESS_NOTIFICATION_TYPE type;
} bt_ssp_keypress_notification;
```

Description

Keypress Notification Event

bt_ssp_oob_data Structure

File

ssp.h

C

```
typedef struct _bt_ssp_oob_data {
  bt_bdaddr_t bdaddr;
  bt_byte hash[OOB_DATA_HASH_LENGTH];
  bt_byte randomizer[OOB_DATA_RANDOMIZER_LENGTH];
} bt_ssp_oob_data;
```

Description

This is type bt_ssp_oob_data.

bt_ssp_simple_pairing_complete Structure

File

ssp.h

C

```
typedef struct _bt_ssp_simple_pairing_complete {
  bt_byte status;
  bt_bdaddr_t bdaddr_remote;
} bt_ssp_simple_pairing_complete;
```

Description

Simple Pairing Complete Event

bt_ssp_user_confirmation_request Structure

File

ssp.h

C

```
typedef struct _bt_ssp_user_confirmation_request {
  bt_bdaddr_t bdaddr_remote;
  bt_ulong numeric_value;
} bt_ssp_user_confirmation_request;
```

Description

User Confirmation Request Event

bt_ssp_user_passkey_notification Structure

File

ssp.h

C

```
typedef struct _bt_ssp_user_passkey_notification {
  bt_bdaddr_t bdaddr_remote;
  bt_ulong passkey;
} bt_ssp_user_passkey_notification;
```

Description

User Passkey Notification Event

bt_ssp_user_passkey_request Structure

File

ssp.h

C

```
typedef struct _bt_ssp_user_passkey_request {
  bt_bdaddr_t bdaddr_remote;
  bt_ulong passkey;
} bt_ssp_user_passkey_request;
```

Description

User Passkey Request Event

SSP_AUTHENTICATION_REQUIREMENTS Enumeration

File

ssp.h

C

```
typedef enum _SSP_AUTHENTICATION_REQUIREMENTS {
  SSP_MITM_NOT_REQUIRED_NO_BONDING = 0,
  SSP_MITM_REQUIRED_NO_BONDING,
  SSP_MITM_NOT_REQUIRED_DEDICATED_BONDING,
  SSP_MITM_REQUIRED_DEDICATED_BONDING,
  SSP_MITM_NOT_REQUIRED_GENERAL_BONDING,
  SSP_MITM_REQUIRED_GENERAL_BONDING
} SSP_AUTHENTICATION_REQUIREMENTS;
```

Description

This is type SSP_AUTHENTICATION_REQUIREMENTS.

SSP_IO_CAPABILITY Enumeration

File

ssp.h

C

```
typedef enum _SSP_IO_CAPABILITY {
  SSP_IO_CAPABILITY_DISPLAY_ONLY = 0,
  SSP_IO_CAPABILITY_DISPLAY_YESNO,
  SSP_IO_CAPABILITY_KEYBOARD_ONLY,
  SSP_IO_CAPABILITY_NO_INPUT_NO_OUTPUT
} SSP_IO_CAPABILITY;
```

Description

This is type SSP_IO_CAPABILITY.

SSP_KEYPRESS_NOTIFICATION_TYPE Enumeration

File

ssp.h

C

```
typedef enum _SSP_KEYPRESS_NOTIFICATION_TYPE {
  PASSKEY_ENTRY_STYARTED,
  PASSKEY_DIGIT_ENTERED,
  PASSKEY_DIGIT_ERASED,
  PASSKEY_CLEARED,
  PASSKEY_ENTRY_COMPLETED
} SSP_KEYPRESS_NOTIFICATION_TYPE;
```

Description

This is type SSP_KEYPRESS_NOTIFICATION_TYPE.

SSP_MODE Enumeration

File

ssp.h

C

```
typedef enum _SSP_MODE {
  SSP_MODE_DISABLED = 0,
  SSP_MODE_ENABLED
} SSP_MODE;
```

Description

This is type SSP_MODE.

SSP_OOB_DATA_PRESENT Enumeration

File

ssp.h

C

```
typedef enum _SSP_OOB_DATA_PRESENT {
  SSP_OOB_DATA_NOT_PRESENT = 0,
  SSP_OOB_DATA_REMOTE_DEVICE_DATA_PRESENT
} SSP_OOB_DATA_PRESENT;
```

Description

This is type SSP_OOB_DATA_PRESENT.

SSP_EVENT Enumeration

File

ssp_event.h

C

```
typedef enum _SSP_EVENT {
  SSP_EVENT_IO_CAPABILITY_RESPONSE,
  SSP_EVENT_KEYPRESS_NOTIFICATION,
  SSP_EVENT_USER_PASSKEY_REQUEST,
  SSP_EVENT_USER_PASSKEY_NOTIFICATION,
  SSP_EVENT_USER_CONFIRMATION_REQUEST,
  SSP_EVENT_OOB_DATA_REQUEST,
  SSP_EVENT_IO_CAPABILITY_REQUEST,
```

```
    SSP_EVENT_SIMPLE_PAIRING_COMPLETE
} SSP_EVENT;
```

Description

This is type SSP_EVENT.

Misc. Functions

btx_csr_alloc_bccmd_getreq Function

File

csr.h

C

```
bt_hci_command_t* btx_csr_alloc_bccmd_getreq(bt_uint var_id, bt_uint data_word_count,
bt_hci_cmd_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_alloc_bccmd_getreq.

btx_csr_alloc_bccmd_setreq Function

File

csr.h

C

```
bt_hci_command_t* btx_csr_alloc_bccmd_setreq(bt_uint var_id, bt_uint data_word_count,
bt_hci_cmd_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_alloc_bccmd_setreq.

btx_csr_autobaud Function

File

csr.h

C

```
void btx_csr_autobaud(btx_csr_autobaud_buffer_t* buffer, btx_csr_autobaud_callback_fp callback, void*
callback_param);
```

Description

brief Configure controller's UART speed. ingroup btx_csr

details This function makes the controller auto-configure its UART speed. The host transport must be set to H4. This function works only with BC6 controllers.

btx_csr_bc7_sel_host_interface_h4 Function

File

csr.h

C

```
void btx_csr_bc7_sel_host_interface_h4(btx_csr_autobaud_buffer_t* buffer, bt_byte interval,
btx_csr_autobaud_callback_fp callback, void* callback_param);
```

Description

brief Configure controller's UART speed and host interface. ingroup btx_csr

details This function makes the controller auto-configure its UART speed and select H4 as host interface. PS_KEY_HOST_INTERFACE must not be set. PS_KEY_UART_BITRATE must be set to 0. This function works only with BC7 controllers.

btx_csr_enable_tx Function

File

csr.h

C

```
bt_bool btx_csr_enable_tx(bt_bool enable, bt_hci_cmd_callback_fp callback, void* callback_param);
```

Description

brief Enable/disable transmitter ingroup btx_csr

param enable Specifies whether the transmitter should be enable or disabled. param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

btx_csr_exec_script Function

File

csr.h

C

```
void btx_csr_exec_script(const btx_csr_script_t* script, btx_csr_exec_script_buffer_t* buffer,
btx_csr_exec_script_callback_fp callback, void* callback_param);
```

Description

brief Patch controller's firmware ingroup btx_csr

details This function executes a script that patches the controller's firmware. The c script must point to a structure that contain a complete patch script for a particular controller model and revision. If the revision specified in the script and revision read from the controller are the same btx_csr_patch_controller() loads the script to the controller and calls the c callback with the first parameter TRUE. Otherwise the c callback is called with the first parameter FALSE.

If support for multiple firmware revisions is neede use btx_csr_patch_controller().

param script Array of patch scripts. param buffer A buffer for storing temporary data needed for script execution. param callback The callback function that will be called when the script has been executed. param callback_param A pointer to arbitrary data to be passed to the c callback callback..

btx_csr_get_cached_temperature Function

File

csr.h

C

```
bt_bool btx_csr_get_cached_temperature(btx_csr_get_var_callback_fp callback, void* callback_param);
```

Description

brief Get chip's cached temperature ingroup btx_csr

param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

btx_csr_get_ps_var Function

File

csr.h

C

```
bt_bool btx_csr_get_ps_var(bt_uint ps_key, bt_uint value_word_count, btx_csr_get_ps_var_callback_fp
callback, void* callback_param);
```

Description

This is function btx_csr_get_ps_var.

btx_csr_get_ps_var_ex Function

File

csr.h

C

```
bt_bool btx_csr_get_ps_var_ex(bt_uint ps_key, bt_uint value_word_count, bt_uint store,
btx_csr_get_ps_var_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_get_ps_var_ex.

btx_csr_get_rssi_acl Function

File

csr.h

C

```
bt_bool btx_csr_get_rssi_acl(bt_hci_hconn_t hconn, btx_csr_get_var_callback_fp callback, void*
callback_param);
```

Description

brief Get RSSI ingroup btx_csr

details This function retrieves the RSSI for a given HCI ACL handle.

param hconn ACL connection handle. param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

btx_csr_get_script__PB_27_R20_BC6ROM_A04 Function

File

csr.h

C

```
const btx_csr_script_t* btx_csr_get_script__PB_27_R20_BC6ROM_A04();
```

Description

brief Return script for patching BlueCore 6 ingroup btx_csr

btx_csr_get_script__PB_90_REV6 Function

File

csr.h

C

```
const btx_csr_script_t* btx_csr_get_script__PB_90_REV6();
```

Description

brief Return script for patching CSR8810 (BlueCore 7) ingroup btx_csr

btx_csr_get_var Function

File

csr.h

C

```
bt_bool btx_csr_get_var(bt_uint var_id, btx_csr_get_var_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_get_var.

btx_csr_set_ps_var Function

File

csr.h

C

```
bt_bool btx_csr_set_ps_var(bt_uint ps_key, const bt_uint* value, bt_uint value_word_count,
bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_csr_set_ps_var.

btx_csr_set_ps_var_ex Function

File

csr.h

C

```
bt_bool btx_csr_set_ps_var_ex(bt_uint ps_key, const bt_uint* value, bt_uint value_word_count, bt_uint
store, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_csr_set_ps_var_ex.

btx_csr_set_ps_vars Function

File

csr.h

C

```
void btx_csr_set_ps_vars(const bt_uint* ps_vars, btx_csr_set_ps_vars_buffer_t* buffer,
btx_csr_set_ps_vars_callback_fp callback, void* callback_param);
```

Description

brief Write PS variables ingroup btx_csr

details

param ps_vars PS values param buffer A buffer for storing temporary data during function execution. param callback The callback function that will be called when all PS values have been sent to the controller or error occurred. param callback_param A pointer to arbitrary data to be passed to the c callback callback..

btx_csr_set_ps_vars_ex Function

File

csr.h

C

```
void btx_csr_set_ps_vars_ex(const bt_uint* ps_vars, btx_csr_set_ps_vars_buffer_t* buffer, bt_uint store,
btx_csr_set_ps_vars_callback_fp callback, void* callback_param);
```

Description

brief Write PS variables ingroup btx_csr

param ps_vars PS values param buffer A buffer for storing temporary data during function execution. param store param callback The callback function that will be called when all PS values have been sent to the controller or error occurred. param callback_param A pointer to arbitrary data to be passed to the c callback callback..

btx_csr_set_var Function

File

csr.h

C

```
bt_bool btx_csr_set_var(bt_uint var_id, const bt_uint* value, bt_uint value_word_count,
btx_csr_set_var_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_set_var.

btx_csr_warm_reset Function

File

csr.h

C

```
bt_bool btx_csr_warm_reset();
```

Description

brief Warm reset ingroup btx_csr

details This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact.

btx_csr_warm_reset_ex Function

File

csr.h

C

```
bt_bool btx_csr_warm_reset_ex(bt_hci_cmd_callback_fp callback, void* callback_param);
```

Description

brief Warm reset ingroup btx_csr

details This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact. Since the controller does not respond to the warm reset command as it starts resetting immediately upon receiving the command, the c callback is called right after the command packet has been transmitted to the controller.

param callback The callback function that will be called after the warm reset command has been sent to the controller. param callback_param A pointer to arbitrary data to be passed to the c callback callback.

btx_ti_drpb_enable_rf_calibration Function

File

ti.h

C

```
bt_bool btx_ti_drpb_enable_rf_calibration(bt_byte periodic_mode, bt_ulong calibration_procedure, bt_byte
temp_condition, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_drpb_enable_rf_calibration.

btx_ti_drpb_set_power_vector Function

File

ti.h

C

```
bt_bool btx_ti_drpb_set_power_vector(bt_byte modulation_type, const bt_byte* power_vector, bt_byte
epc_max_level_threshold, bt_uint external_pa_mode, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_drpb_set_power_vector.

btx_ti_drpb_tester_con_tx Function

File

ti.h

C

```
bt_bool btx_ti_drpb_tester_con_tx(bt_byte modulation_scheme, bt_byte test_pattern, bt_byte
frequency_channel, bt_byte power_level, bt_ulong generator_initialization_value, bt_ulong
edr_generator_mask, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_drpb_tester_con_tx.

btx_ti_enable_deep_sleep Function

File

ti.h

C

```
bt_bool btx_ti_enable_deep_sleep(bt_bool enable, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_enable_deep_sleep.

btx_ti_enable_fast_clock_crystal Function

File

ti.h

C

```
bt_bool btx_ti_enable_fast_clock_crystal(bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_enable_fast_clock_crystal.

btx_ti_enable_low_power_scan Function

File

ti.h

C

```
bt_bool btx_ti_enable_low_power_scan(bt_bool enable, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_enable_low_power_scan.

btx_ti_enable_low_power_scan_default Function

File

ti.h

C

```
bt_bool btx_ti_enable_low_power_scan_default(bt_bool enable, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_enable_low_power_scan_default.

btx_ti_exec_script Function

File

ti.h

C

```
void btx_ti_exec_script(const btx_ti_script_t* script, btx_ti_exec_script_buffer_t* buffer,
btx_ti_completion_callback_fp callback, void* callback_param);
```

Description

This is function btx_ti_exec_script.

btx_ti_exec_script_oem Function

File

ti.h

C

```
void btx_ti_exec_script_oem(const btx_ti_script_t* script, btx_ti_exec_script_oem_buffer_t* buffer,
btx_ti_exec_script_oem_callback_fp callback, void* callback_param);
```

Description

This is function btx_ti_exec_script_oem.

btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_36 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_36();
```

Description

CC2560 Scripts (Service pack 2.36)

btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_44 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_44();
```

Description

CC2560 Scripts (Service pack 2.44)

btx_ti_get_script__BL6450L_BT_Service_Pack_2_10 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_10();
```

Description

CC2564 Scripts (Service pack 2.10)

btx_ti_get_script__BL6450L_BT_Service_Pack_2_10_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_10_BLE_AddOn();
```

Description

This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_10_BLE_AddOn.

btx_ti_get_script__BL6450L_BT_Service_Pack_2_12 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_12();
```

Description

CC2564 Scripts (Service pack 2.12)

btx_ti_get_script__BL6450L_BT_Service_Pack_2_12_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_12_BLE_AddOn();
```

Description

This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_12_BLE_AddOn.

btx_ti_get_script__BL6450L_BT_Service_Pack_2_7 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_7();
```

Description

CC2564 Scripts (Service pack 2.7)

btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_AVPR_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_AVPR_AddOn();
```

Description

This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_AVPR_AddOn.

btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_BLE_AddOn();
```

Description

This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_BLE_AddOn.

btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_Short Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_Short();
```

Description

CC2564 Scripts (Service pack 2.8)

btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_AVPR_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_AVPR_AddOn();
```

Description

This is function btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_AVPR_AddOn.

btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_BLE_AddOn();
```

Description

This is function btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_BLE_AddOn.

btx_ti_get_script__CC2564B_BT_Service_Pack_0_1 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_0_1();
```

Description

CC2564B Scripts (Service pack 0.1)

btx_ti_get_script__CC2564B_BT_Service_Pack_0_1_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_0_1_BLE_AddOn();
```

Description

This is function btx_ti_get_script__CC2564B_BT_Service_Pack_0_1_BLE_AddOn.

btx_ti_get_script__CC2564B_BT_Service_Pack_0_2 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_0_2();
```

Description

CC2564B Scripts (Service pack 0.2)

btx_ti_get_script__CC2564B_BT_Service_Pack_0_2_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_0_2_BLE_AddOn();
```

Description

This is function btx_ti_get_script__CC2564B_BT_Service_Pack_0_2_BLE_AddOn.

btx_ti_get_script__WL127xL_BT_Service_Pack_2_3 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_3();
```

Description

CC2564 Scripts (Service pack 2.3)

btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_AVPR_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_AVPR_AddOn();
```

Description

This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_AVPR_AddOn.

btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_BLE_AddOn();
```

Description

This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_BLE_AddOn.

btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_DC2DC_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_DC2DC_AddOn();
```

Description

This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_DC2DC_AddOn.

btx_ti_get_script__WL127xL_BT_Service_Pack_2_4 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_4();
```

Description

CC2564 Scripts (Service pack 2.4)

btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_AVPR_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_AVPR_AddOn();
```

Description

This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_AVPR_AddOn.

btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_BLE_AddOn();
```

Description

This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_BLE_AddOn.

btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_DC2DC_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_DC2DC_AddOn();
```

Description

This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_DC2DC_AddOn.

btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3();
```

Description

CC2564 Scripts (Service pack 1.3)

btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3_BLE_Init Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3_BLE_Init();
```

Description

This is function btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3_BLE_Init.

btx_ti_init_ble_controller Function

File

ti.h

C

```
void btx_ti_init_ble_controller(btx_ti_exec_script_buffer_t* buffer, btx_ti_completion_callback_fp
callback, void* callback_param);
```

Description

This is function btx_ti_init_ble_controller.

btx_ti_init_controller Function

File

ti.h

C

```
void btx_ti_init_controller(btx_ti_exec_script_buffer_t* buffer, btx_ti_completion_callback_fp callback,
void* callback_param);
```

Description

This is function btx_ti_init_controller.

btx_ti_set_uart_baud_rate Function

File

ti.h

C

```
bt_bool btx_ti_set_uart_baud_rate(bt_ulong baud_rate, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_set_uart_baud_rate.

btx_ti_write_bdaddr Function

File

ti.h

C

```
bt_bool btx_ti_write_bdaddr(bt_bdaddr_t* bdaddr, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_write_bdaddr.

btx_ti_write_hardware_register Function

File

ti.h

C

```
bt_bool btx_ti_write_hardware_register(bt_ulong address, bt_uint value, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_write_hardware_register.

btx_csr_get_script__dsp_script__PB_109_DSP_rev8 Function

File

csr.h

C

```
const btx_csr_script_t* btx_csr_get_script__dsp_script__PB_109_DSP_rev8();
```

Description

brief Return script for patching DSP in CSR8x11 A08 (BlueCore 7) ingroup btx_csr

btx_csr_get_script__PB_101_CSR8811_CSP28_UART Function

File

csr.h

C

```
const btx_csr_script_t* btx_csr_get_script__PB_101_CSR8811_CSP28_UART();
```

Description

brief Return script for patching CSR8x11 A06 (BlueCore 7) ingroup btx_csr

btx_csr_get_script__PB_109_CSR8811_REV16 Function

File

csr.h

C

```
const btx_csr_script_t* btx_csr_get_script__PB_109_CSR8811_REV16();
```

Description

brief Return script for patching CSR8x11 A08 (BlueCore 7) ingroup btx_csr

btx_csr_get_script__PB_173_CSR8X11_REV1 Function

File

csr.h

C

```
const btx_csr_script_t* btx_csr_get_script__PB_173_CSR8X11_REV1();
```

Description

brief Return script for patching CSR8x11 A12 (BlueCore 7) ingroup btx_csr

btx_csr_init Function

File

csr.h

C

```
void btx_csr_init();
```

Description

brief Initialize CSR support layer. ingroup btx_csr

details This function initializes all internal variables of the CSR support layer. CSR controllers use vendor specific event (0xFF) to carry the BCCMD protocol. They also do not report number of completed packets for BCCMD commands. This function installs a vendor specific event handler that makes sure that callback are called on corresponding vendor specific commands and the number of free command buffers in the controller is kept correct.

btx_csr_init_hq_script Function

File

csr.h

C

```
void btx_csr_init_hq_script(const btx_csr_script_t* script, btx_csr_exec_hq_script_buffer_t* buffer);
```

Description

This is function btx_csr_init_hq_script.

btx_csr_patch_controller Function

File

csr.h

C

```
void btx_csr_patch_controller(const btx_csr_get_script_fp* scripts, bt_int script_count,
btx_csr_exec_script_buffer_t* buffer, btx_csr_exec_script_callback_fp callback, void* callback_param);
```

Description

brief Patch controller's firmware ingroup btx_csr

details This function executes a script that patches the controller's firmware. Each entry of the c scripts array must be a complete patch script for a particular controller model and revision. btx_csr_patch_controller() reads the revision number from the controller then tries to find the corresponding script in the c scripts. If there is a matching script it is loaded to the controller and c callback is called with the first parameter TRUE. If no suitable script found c callback is called with the first parameter FALSE.

param scripts Array of patch scripts. param script_count The number of scripts in c scripts. param buffer A buffer for storing temporary data needed for script execution. param callback The callback function that will be called when the script has been executed. param callback_param A pointer to arbitrary data to be passed to the c callback callback..

btx_csr_register_bccmd_listener Function

File

csr.h

C

```
bt_bool btx_csr_register_bccmd_listener(btx_csr_bccmd_listener_t* listener);
```

Description

This is function btx_csr_register_bccmd_listener.

btx_csr_send_dsp_config_data Function

File

csr.h

C

```
bt_bool btx_csr_send_dsp_config_data(bt_uint total_config_blocks, bt_uint seq_no, bt_uint status, const
bt_uint* data, bt_uint data_word_count, bt_hci_cmd_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_send_dsp_config_data.

btx_csr_send_next_hq_script_packet Function

File

csr.h

C

```
bt_bool btx_csr_send_next_hq_script_packet(bt_uint seq_no, btx_csr_exec_hq_script_buffer_t* buffer);
```

Description

This is function btx_csr_send_next_hq_script_packet.

btx_csr_unregister_bccmd_listener Function

File

csr.h

C

```
void btx_csr_unregister_bccmd_listener(btx_csr_bccmd_listener_t* listener);
```

Description

This is function btx_csr_unregister_bccmd_listener.

btx_ti_a3dp_sink_close_stream Function

File

ti.h

C

```
bt_bool btx_ti_a3dp_sink_close_stream(bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_a3dp_sink_close_stream.

btx_ti_a3dp_sink_codec_config Function

File

ti.h

C

```
bt_bool btx_ti_a3dp_sink_codec_config(bt_byte channels, bt_byte sample_frequency, bt_byte channel_mode,
bt_byte blocks, bt_byte sub_bands, bt_byte allocation_method, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_a3dp_sink_codec_config.

btx_ti_a3dp_sink_open_stream Function

File

ti.h

C

```
bt_bool btx_ti_a3dp_sink_open_stream(bt_byte acl_handle, bt_uint l2cap_cid, bt_hci_cmd_callback_fp
callback);
```

Description

This is function btx_ti_a3dp_sink_open_stream.

btx_ti_a3dp_sink_start_stream Function

File

ti.h

C

```
bt_bool btx_ti_a3dp_sink_start_stream(bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_a3dp_sink_start_stream.

btx_ti_a3dp_sink_stop_stream Function

File

ti.h

C

```
bt_bool btx_ti_a3dp_sink_stop_stream(bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_a3dp_sink_stop_stream.

btx_ti_avpr_debug Function

File

ti.h

C

```
bt_bool btx_ti_avpr_debug(bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_avpr_debug.

btx_ti_avpr_enable Function

File

ti.h

C

```
bt_bool btx_ti_avpr_enable(bt_byte enable, bt_byte load_a3dp_code, bt_byte a3dp_role,
bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_avpr_enable.

btx_ti_get_script__BL6450L_BT_Service_Pack_2_14 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_14();
```

Description

CC2564 Scripts (Service pack 2.14)

btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn();
```

Description

This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn.

btx_ti_get_script__CC2564B_BT_Service_Pack_1_0 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_0();
```

Description

CC2564B Scripts (Service pack 1.0)

btx_ti_get_script__CC2564B_BT_Service_Pack_1_0_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_0_BLE_AddOn();
```

Description

This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_0_BLE_AddOn.

btx_ti_get_script__CC2564B_BT_Service_Pack_1_1 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_1();
```

Description

CC2564B Scripts (Service pack 1.1)

btx_ti_get_script__CC2564B_BT_Service_Pack_1_1_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_1_BLE_AddOn();
```

Description

This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_1_BLE_AddOn.

btx_ti_get_script__CC2564B_BT_Service_Pack_1_2 Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_2();
```

Description

CC2564B Scripts (Service pack 1.2)

btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn();
```

Description

This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn.

btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn Function

File

ti.h

C

```
const btx_ti_script_t* btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn();
```

Description

This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn.

btx_ti_le_enable Function

File

ti.h

C

```
bt_bool btx_ti_le_enable(bt_byte enable, bt_byte load_le_code, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_le_enable.

btx_ti_set_afh_mode Function

File

ti.h

C

```
bt_bool btx_ti_set_afh_mode(bt_hci_hconn_t hconn, bt_bool enable, bt_hci_cmd_callback_fp callback);
```

Description

This is function btx_ti_set_afh_mode.

btx_ti_write_codec_config Function

File

ti.h

C

```
bt_bool btx_ti_write_codec_config(const btx_ti_codec_config_t* codec_config, bt_hci_cmd_callback_fp
callback);
```

Description

This is function btx_ti_write_codec_config.

btx_csr_exec_hq_script Function

File

csr.h

C

```
void btx_csr_exec_hq_script(const btx_csr_script_t* script, btx_csr_exec_hq_script_buffer_t* buffer,
btx_csr_exec_hq_script_callback_fp callback, void* callback_param);
```

Description

This is function btx_csr_exec_hq_script.

Misc. Data Types and Constants

CSR_SNK_ADC Macro

File

csr.h

C

```
#define CSR_SNK_ADC 3
```

Description

This is macro CSR_SNK_ADC.

CSR_SNK_FASTPIPE Macro

File

csr.h

C

```
#define CSR_SNK_FASTPIPE 8
```

Description

This is macro CSR_SNK_FASTPIPE.

CSR_SNK_FM Macro

File

csr.h

C

```
#define CSR_SNK_FM 4
```

Description

This is macro CSR_SNK_FM.

CSR_SNK_I2S Macro

File

csr.h

C

```
#define CSR_SNK_I2S 2
```

Description

This is macro CSR_SNK_I2S.

CSR_SNK_L2CAP Macro

File

csr.h

C

```
#define CSR_SNK_L2CAP 7
```

Description

This is macro CSR_SNK_L2CAP.

CSR_SNK_PCM Macro

File

csr.h

C

```
#define CSR_SNK_PCM 1
```

Description

This is macro CSR_SNK_PCM.

CSR_SNK_SCO Macro

File

csr.h

C

```
#define CSR_SNK_SCO 9
```

Description

This is macro CSR_SNK_SCO.

CSR_SNK_SPDIF Macro

File

csr.h

C

```
#define CSR_SNK_SPDIF 5
```

Description

This is macro CSR_SNK_SPDIF.

CSR_SRC_ADC Macro

File

csr.h

C

```
#define CSR_SRC_ADC 3
```

Description

This is macro CSR_SRC_ADC.

CSR_SRC_FASTPIPE Macro

File

csr.h

C

```
#define CSR_SRC_FASTPIPE 8
```

Description

This is macro CSR_SRC_FASTPIPE.

CSR_SRC_FM Macro

File

csr.h

C

```
#define CSR_SRC_FM 4
```

Description

This is macro CSR_SRC_FM.

CSR_SRC_I2S Macro

File

csr.h

C

```
#define CSR_SRC_I2S 2
```

Description

This is macro CSR_SRC_I2S.

CSR_SRC_L2CAP Macro

File

csr.h

C

```
#define CSR_SRC_L2CAP 7
```

Description

This is macro CSR_SRC_L2CAP.

CSR_SRC_MIC Macro

File

csr.h

C

```
#define CSR_SRC_MIC 6
```

Description

This is macro CSR_SRC_MIC.

CSR_SRC_PCM Macro

File

csr.h

C

```
#define CSR_SRC_PCM 1
```

Description

This is macro CSR_SRC_PCM.

CSR_SRC_SCO Macro

File

csr.h

C

```
#define CSR_SRC_SCO 9
```

Description

This is macro CSR_SRC_SCO.

CSR_SRC_SPDIF Macro

File

csr.h

C

```
#define CSR_SRC_SPDIF 5
```

Description

This is macro CSR_SRC_SPDIF.

CSR_VARID_CACHED_TEMPERATURE Macro

File

csr.h

C

```
#define CSR_VARID_CACHED_TEMPERATURE 0x2872
```

Description

This is macro CSR_VARID_CACHED_TEMPERATURE.

CSR_VARID_ENABLE_SCO_STREAMS Macro

File

csr.h

C

```
#define CSR_VARID_ENABLE_SCO_STREAMS 0x4876
```

Description

This is macro CSR_VARID_ENABLE_SCO_STREAMS.

CSR_VARID_MAP_SCO_AUDIO Macro

File

csr.h

C

```
#define CSR_VARID_MAP_SCO_AUDIO 0x506a
```

Description

This is macro CSR_VARID_MAP_SCO_AUDIO.

CSR_VARID_PIO Macro

File

csr.h

C

```
#define CSR_VARID_PIO 0x681f
```

Description

This is macro CSR_VARID_PIO.

CSR_VARID_PIO_DIRECTION_MASK Macro

File

csr.h

C

```
#define CSR_VARID_PIO_DIRECTION_MASK 0x681e
```

Description

This is macro CSR_VARID_PIO_DIRECTION_MASK.

CSR_VARID_PIO_PROTECT_MASK Macro

File

csr.h

C

```
#define CSR_VARID_PIO_PROTECT_MASK 0x2823
```

Description

This is macro CSR_VARID_PIO_PROTECT_MASK.

CSR_VARID_RSSI_ACL Macro

File

csr.h

C

```
#define CSR_VARID_RSSI_ACL 0x301d
```

Description

This is macro CSR_VARID_RSSI_ACL.

CSR_VARID_STREAM_CLOSE_SINK Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_CLOSE_SINK 0x486c
```

Description

This is macro CSR_VARID_STREAM_CLOSE_SINK.

CSR_VARID_STREAM_CLOSE_SOURCE Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_CLOSE_SOURCE 0x486b
```

Description

This is macro CSR_VARID_STREAM_CLOSE_SOURCE.

CSR_VARID_STREAM_CONFIGURE Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_CONFIGURE 0x505c
```

Description

This is macro CSR_VARID_STREAM_CONFIGURE.

CSR_VARID_STREAM_CONNECT Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_CONNECT 0x505e
```

Description

This is macro CSR_VARID_STREAM_CONNECT.

CSR_VARID_STREAM_GET_SINK Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_GET_SINK 0x505b
```

Description

This is macro CSR_VARID_STREAM_GET_SINK.

CSR_VARID_STREAM_GET_SOURCE Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_GET_SOURCE 0x505a
```

Description

This is macro CSR_VARID_STREAM_GET_SOURCE.

GETRESP_BAD_REQ Macro

File

csr.h

C

```
#define GETRESP_BAD_REQ 0x0004
```

Description

This is macro GETRESP_BAD_REQ.

GETRESP_ERROR Macro

File

csr.h

C

```
#define GETRESP_ERROR 0x0008
```

Description

This is macro GETRESP_ERROR.

GETRESP_NO_ACCESS Macro

File

csr.h

C

```
#define GETRESP_NO_ACCESS 0x0005
```

Description

This is macro GETRESP_NO_ACCESS.

GETRESP_NO_SUCH_VARID Macro

File

csr.h

C

```
#define GETRESP_NO_SUCH_VARID 0x0001
```

Description

This is macro GETRESP_NO_SUCH_VARID.

GETRESP_NO_VALUE Macro

File

csr.h

C

```
#define GETRESP_NO_VALUE 0x0003
```

Description

This is macro GETRESP_NO_VALUE.

GETRESP_OK Macro

File

csr.h

C

```
#define GETRESP_OK 0x0000
```

Description

BCCMD GETRESP status codes

GETRESP_PERMISSION_DENIED Macro

File

csr.h

C

```
#define GETRESP_PERMISSION_DENIED 0x0009
```

Description

This is macro GETRESP_PERMISSION_DENIED.

GETRESP_READ_ONLY Macro

File

csr.h

C

```
#define GETRESP_READ_ONLY 0x0006
```

Description

This is macro GETRESP_READ_ONLY.

GETRESP_TOO_BIG Macro

File

csr.h

C

```
#define GETRESP_TOO_BIG 0x0002
```

Description

This is macro GETRESP_TOO_BIG.

GETRESP_WRITE_ONLY Macro

File

csr.h

C

```
#define GETRESP_WRITE_ONLY 0x0007
```

Description

This is macro GETRESP_WRITE_ONLY.

H Macro

File

patch.h

C

```
#define H(ps_key, val_len, seq) \
    0x01,          /* HCI command packet indicator.*/ \
    0x00, 0xfc,    /* HCI opcode { OGF=0x3f OCF=0x00 } */ \
                   /*     OGF=0x3f implies a manufacturer-sepcific command. */ \
                   /* Parameter length. */ \
    1 + (5 + 3 + val_len) * 2, \
    0xc2,          /* CSR payload descriptor: */ \
                   /*    bit 7: Last fragment = 1 */ \
                   /*    bit 6: First fragment = 1 */ \
                   /*    bits 5-0 Channel ID = 2 (BCCMD) */ \
    W(0x0002),     /* SETREQ message */ \
    W(5+3+val_len), /* length*/ \
    W(seq),        /* sequence number */ \
    W(0x7003),     /* Variable ID of PS command */ \
    W(0x0000),     /* Status (must be 0 in SETREQ) */ \
    \
    W(ps_key),     /* PS key */ \
    W(val_len),    /* Value length in 16-bit words */ \
    W(0)           /* PS key store (0 = use default) */
```

Description

PS key store (0 = use default)

PS_DEFAULT Macro

File

csr.h

C

```
#define PS_DEFAULT 0x0000
```

Description

This is macro PS_DEFAULT.

PS_F Macro

File

csr.h

C

```
#define PS_F 0x0002
```

Description

This is macro PS_F.

PS_I Macro

File

csr.h

C

```
#define PS_I 0x0001
```

Description

This is macro PS_I.

PS_RAM Macro

File

csr.h

C

```
#define PS_RAM 0x0008
```

Description

This is macro PS_RAM.

PS_ROM Macro

File

csr.h

C

```
#define PS_ROM 0x0004
```

Description

This is macro PS_ROM.

PSKEY_ANA_FREQ Macro

File

csr.h

C

```
#define PSKEY_ANA_FREQ 0x01fe
```

Description

This is macro PSKEY_ANA_FREQ.

PSKEY_BDADDR Macro

File

csr.h

C

```
#define PSKEY_BDADDR 0x0001
```

Description

Select PS key definitions

PSKEY_CLOCK_REQUEST_ENABLE Macro

File

csr.h

C

```
#define PSKEY_CLOCK_REQUEST_ENABLE 0x0246
```

Description

This is macro PSKEY_CLOCK_REQUEST_ENABLE.

PSKEY_DEEP_SLEEP_CLEAR_RTS Macro

File

csr.h

C

```
#define PSKEY_DEEP_SLEEP_CLEAR_RTS 0x0252
```

Description

This is macro PSKEY_DEEP_SLEEP_CLEAR_RTS.

PSKEY_DEEP_SLEEP_STATE Macro

File

csr.h

C

```
#define PSKEY_DEEP_SLEEP_STATE 0x0229
```

Description

This is macro PSKEY_DEEP_SLEEP_STATE.

PSKEY_DEEP_SLEEP_USE_EXTERNAL_CLOCK Macro

File

csr.h

C

```
#define PSKEY_DEEP_SLEEP_USE_EXTERNAL_CLOCK 0x03c3
```

Description

This is macro PSKEY_DEEP_SLEEP_USE_EXTERNAL_CLOCK.

PSKEY_DEEP_SLEEP_WAKE_CTS Macro

File

csr.h

C

```
#define PSKEY_DEEP_SLEEP_WAKE_CTS 0x023c
```

Description

This is macro PSKEY_DEEP_SLEEP_WAKE_CTS.

PSKEY_DIGITAL_AUDIO_BITS_PER_SAMPLE Macro

File

csr.h

C

```
#define PSKEY_DIGITAL_AUDIO_BITS_PER_SAMPLE 0x01db
```

Description

This is macro PSKEY_DIGITAL_AUDIO_BITS_PER_SAMPLE.

PSKEY_DIGITAL_AUDIO_CONFIG Macro

File

csr.h

C

```
#define PSKEY_DIGITAL_AUDIO_CONFIG 0x01d9
```

Description

This is macro PSKEY_DIGITAL_AUDIO_CONFIG.

PSKEY_HCI_NOP_DISABLE Macro

File

csr.h

C

```
#define PSKEY_HCI_NOP_DISABLE 0x00f2
```

Description

This is macro PSKEY_HCI_NOP_DISABLE.

PSKEY_HOST_INTERFACE Macro

File

csr.h

C

```
#define PSKEY_HOST_INTERFACE 0x01f9
```

Description

This is macro PSKEY_HOST_INTERFACE.

PSKEY_HOSTIO_MAP_SCO_PCM Macro

File

csr.h

C

```
#define PSKEY_HOSTIO_MAP_SCO_PCM 0x01ab
```

Description

This is macro PSKEY_HOSTIO_MAP_SCO_PCM.

PSKEY_HOSTIO_UART_RESET_TIMEOUT Macro

File

csr.h

C

```
#define PSKEY_HOSTIO_UART_RESET_TIMEOUT 0x01a4
```

Description

This is macro PSKEY_HOSTIO_UART_RESET_TIMEOUT.

PSKEY_PCM_PULL_CONTROL Macro

File

csr.h

C

```
#define PSKEY_PCM_PULL_CONTROL 0x01e2
```

Description

This is macro PSKEY_PCM_PULL_CONTROL.

PSKEY_PIO_DEEP_SLEEP_EITHER_LEVEL Macro

File

csr.h

C

```
#define PSKEY_PIO_DEEP_SLEEP_EITHER_LEVEL 0x21bd
```

Description

This is macro PSKEY_PIO_DEEP_SLEEP_EITHER_LEVEL.

PSKEY_UART_BAUDRATE Macro

File

csr.h

C

```
#define PSKEY_UART_BAUDRATE 0x01be
```

Description

This is macro PSKEY_UART_BAUDRATE.

PSKEY_UART_BITRATE Macro

File

csr.h

C

```
#define PSKEY_UART_BITRATE 0x01ea
```

Description

This is macro PSKEY_UART_BITRATE.

PSKEY_UART_CONFIG_BCSP Macro

File

csr.h

C

```
#define PSKEY_UART_CONFIG_BCSP 0x01bf
```

Description

This is macro PSKEY_UART_CONFIG_BCSP.

PSKEY_UART_CONFIG_H4 Macro

File

csr.h

C

```
#define PSKEY_UART_CONFIG_H4 0x01c0
```

Description

This is macro PSKEY_UART_CONFIG_H4.

PSKEY_UART_CONFIG_H5 Macro

File

csr.h

C

```
#define PSKEY_UART_CONFIG_H5 0x01c1
```

Description

This is macro PSKEY_UART_CONFIG_H5.

PSKEY_UART_TX_WINDOW_SIZE Macro

File

csr.h

C

```
#define PSKEY_UART_TX_WINDOW_SIZE 0x01c6
```

Description

This is macro PSKEY_UART_TX_WINDOW_SIZE.

PSKEY_VM_DISABLE Macro

File

csr.h

C

```
#define PSKEY_VM_DISABLE 0x025d
```

Description

This is macro PSKEY_VM_DISABLE.

SET_PS_VALUE_BDADDR Macro

File

csr.h

C

```
#define SET_PS_VALUE_BDADDR(key, m, l) key, 4, (uint16_t)((((uint32_t)l) >> 16) & 0xFF), (uint16_t)(l
&0xFFFF), (uint16_t)((((uint32_t)l) >> 24) & 0xFF), (uint16_t)(m &0xFFFF)
```

Description

This is macro SET_PS_VALUE_BDADDR.

SET_PS_VALUE_UINT16 Macro

File

csr.h

C

```
#define SET_PS_VALUE_UINT16(key, value) key, 1, value
```

Description

Macros for defining lists of PS values for use with btx_csr_set_ps_vars().

SET_PS_VALUE_UINT32 Macro

File

csr.h

C

```
#define SET_PS_VALUE_UINT32(key, value) key, 2, (uint16_t)((((uint32_t)value) >> 16) & 0xFFFF),
(uint16_t)(value &0xFFFF)
```

Description

This is macro SET_PS_VALUE_UINT32.

W Macro

File

patch.h

C

```
#define W(w) _W((bt_uint)(w))
```

Description

This is macro W.

BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE Macro

File

ti.h

C

```
#define BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE 10
```

Description

This is macro BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE.

btx_ti_get_script__CC2560_ServicePack Macro

File

ti.h

C

```
#define btx_ti_get_script__CC2560_ServicePack btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_44
```

Description

Alias for the latest script

btx_ti_get_script__CC2564_BLE_Init Macro

File

ti.h

C

```
#define btx_ti_get_script__CC2564_BLE_Init btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn
```

Description

This is macro btx_ti_get_script__CC2564_BLE_Init.

btx_ti_get_script__CC2564_ServicePack Macro

File

ti.h

C

```
#define btx_ti_get_script__CC2564_ServicePack btx_ti_get_script__BL6450L_BT_Service_Pack_2_14
```

Description

Aliases for latest scripts

btx_ti_get_script__CC2564B_BLE_Init Macro

File

ti.h

C

```
#define btx_ti_get_script__CC2564B_BLE_Init btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn
```

Description

This is macro btx_ti_get_script__CC2564B_BLE_Init.

btx_ti_get_script__CC2564B_ServicePack Macro

File

ti.h

C

```
#define btx_ti_get_script__CC2564B_ServicePack btx_ti_get_script__CC2564B_BT_Service_Pack_1_2
```

Description

Aliases for latest scripts

BTX_TI_MODULATION_SCHEME_8_DPSK Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_SCHEME_8_DPSK 3
```

Description

This is macro BTX_TI_MODULATION_SCHEME_8_DPSK.

BTX_TI_MODULATION_SCHEME_CW Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_SCHEME_CW 0
```

Description

This is macro BTX_TI_MODULATION_SCHEME_CW.

BTX_TI_MODULATION_SCHEME_GFSK Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_SCHEME_GFSK 1
```

Description

This is macro BTX_TI_MODULATION_SCHEME_GFSK.

BTX_TI_MODULATION_SCHEME_P4_DPSK Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_SCHEME_P4_DPSK 2
```

Description

This is macro BTX_TI_MODULATION_SCHEME_P4_DPSK.

BTX_TI_MODULATION_TYPE_EDR2 Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_TYPE_EDR2 1
```

Description

This is macro BTX_TI_MODULATION_TYPE_EDR2.

BTX_TI_MODULATION_TYPE_EDR3 Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_TYPE_EDR3 2
```

Description

This is macro BTX_TI_MODULATION_TYPE_EDR3.

BTX_TI_MODULATION_TYPE_GFSK Macro

File

ti.h

C

```
#define BTX_TI_MODULATION_TYPE_GFSK 0
```

Description

This is macro BTX_TI_MODULATION_TYPE_GFSK.

BTX_TI_TEST_PATTERN_ALL_0 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_ALL_0 4
```

Description

This is macro BTX_TI_TEST_PATTERN_ALL_0.

BTX_TI_TEST_PATTERN_ALL_1 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_ALL_1 3
```

Description

This is macro BTX_TI_TEST_PATTERN_ALL_1.

BTX_TI_TEST_PATTERN_F0F0 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_F0F0 5
```

Description

This is macro BTX_TI_TEST_PATTERN_F0F0.

BTX_TI_TEST_PATTERN_FF00 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_FF00 6
```

Description

This is macro BTX_TI_TEST_PATTERN_FF00.

BTX_TI_TEST_PATTERN_PN15 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_PN15 1
```

Description

This is macro BTX_TI_TEST_PATTERN_PN15.

BTX_TI_TEST_PATTERN_PN9 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_PN9 0
```

Description

This is macro BTX_TI_TEST_PATTERN_PN9.

BTX_TI_TEST_PATTERN_USER Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_USER 7
```

Description

This is macro BTX_TI_TEST_PATTERN_USER.

BTX_TI_TEST_PATTERN_Z0Z0 Macro

File

ti.h

C

```
#define BTX_TI_TEST_PATTERN_Z0Z0 2
```

Description

This is macro BTX_TI_TEST_PATTERN_Z0Z0.

_W Macro

File

patch.h

C

```
#define _W(w) (bt_byte)((w) & 0xFF), (bt_byte)((w) >> 8)
```

Description

This is macro _W.

_btx_csr_autobaud_buffer_t Structure

File

csr.h

C

```
struct _btx_csr_autobaud_buffer_t {
  bt_byte* recv_buffer;
  bt_uint recv_buffer_len;
  btx_csr_autobaud_callback_fp callback;
  void* callback_param;
};
```

Description

This is record _btx_csr_autobaud_buffer_t.

_btx_csr_exec_script_buffer_t Structure

File

csr.h

C

```
struct _btx_csr_exec_script_buffer_t {
  const btx_csr_script_t* script;
  btx_csr_exec_script_callback_fp callback;
  void* callback_param;
  bt_int current_packet;
};
```

Description

This is record _btx_csr_exec_script_buffer_t.

_btx_csr_set_ps_vars_buffer_t Structure

File

csr.h

C

```
struct _btx_csr_set_ps_vars_buffer_t {
  const bt_uint* ps_vars;
  btx_csr_set_ps_vars_callback_fp callback;
  void* callback_param;
  bt_uint current_var;
};
```

Description

This is record _btx_csr_set_ps_vars_buffer_t.

_btx_ti_exec_script_buffer_t Structure

File

ti.h

C

```
struct _btx_ti_exec_script_buffer_t {
  const btx_ti_script_t* script;
  btx_ti_completion_callback_fp callback;
  void* callback_param;
  bt_int current_packet;
};
```

Description

This is record _btx_ti_exec_script_buffer_t.

_btx_ti_exec_script_oem_buffer_t Structure

File

ti.h

C

```
struct _btx_ti_exec_script_oem_buffer_t {
  const btx_ti_script_t* script;
  btx_ti_exec_script_oem_callback_fp callback;
  void* callback_param;
  bt_int current_packet;
  bt_int state;
  bt_byte rx_buffer[BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE];
};
```

Description

This is record _btx_ti_exec_script_oem_buffer_t.

btx_csr_autobaud_buffer_t Type

File

csr.h

C

```
typedef struct _btx_csr_autobaud_buffer_t btx_csr_autobaud_buffer_t;
```

Description

This is type btx_csr_autobaud_buffer_t.

btx_csr_autobaud_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_autobaud_callback_fp)(bt_bool success, btx_csr_autobaud_buffer_t* buffer);
```

Description

This is type btx_csr_autobaud_callback_fp.

btx_csr_bccmd_header_t Structure

File

csr.h

C

```
typedef struct _btx_csr_bccmd_header_s {
  bt_uint type;
  bt_uint len;
  bt_uint seq_no;
  bt_uint var_id;
  bt_uint status;
  bt_byte* payload;
} btx_csr_bccmd_header_t;
```

Description

This is type btx_csr_bccmd_header_t.

btx_csr_cached_temperature_t Structure

File

csr.h

C

```
typedef struct _btx_csr_cached_temperature_s {
  btx_csr_bccmd_header_t message;
  bt_uint temperature;
} btx_csr_cached_temperature_t;
```

Description

This is type btx_csr_cached_temperature_t.

btx_csr_exec_script_buffer_t Type

File

csr.h

C

```
typedef struct _btx_csr_exec_script_buffer_t btx_csr_exec_script_buffer_t;
```

Description

This is type btx_csr_exec_script_buffer_t.

btx_csr_exec_script_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_exec_script_callback_fp)(bt_bool success, btx_csr_exec_script_buffer_t* buffer);
```

Description

This is type btx_csr_exec_script_callback_fp.

btx_csr_get_ps_var_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_get_ps_var_callback_fp)(bt_uint success, const bt_byte* value, bt_uint len, void*
callback_param);
```

Description

This is type btx_csr_get_ps_var_callback_fp.

btx_csr_get_var_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_get_var_callback_fp)(bt_uint status, bt_uint var_id, btx_csr_var_t* var_value,
void* callback_param);
```

Description

This is type btx_csr_get_var_callback_fp.

btx_csr_pio_direction_mask_t Structure

File

csr.h

C

```
typedef struct _btx_csr_pio_direction_mask_s {
  btx_csr_bccmd_header_t message;
  bt_uint direction;
} btx_csr_pio_direction_mask_t;
```

Description

This is type btx_csr_pio_direction_mask_t.

btx_csr_pio_protection_mask_t Structure

File

csr.h

C

```
typedef struct _btx_csr_pio_protection_mask_s {
  btx_csr_bccmd_header_t message;
  bt_uint protection;
} btx_csr_pio_protection_mask_t;
```

Description

This is type btx_csr_pio_protection_mask_t.

btx_csr_pio_t Structure

File

csr.h

C

```
typedef struct _btx_csr_pio_s {
  btx_csr_bccmd_header_t message;
  bt_uint pio;
} btx_csr_pio_t;
```

Description

This is type btx_csr_pio_t.

btx_csr_rssi_acl_t Structure

File

csr.h

C

```
typedef struct _btx_csr_rssi_acl_s {
  btx_csr_bccmd_header_t message;
  bt_hci_hconn_t hconn;
  bt_uint rssi;
} btx_csr_rssi_acl_t;
```

Description

This is type btx_csr_rssi_acl_t.

btx_csr_script_t Structure

File

csr.h

C

```
typedef struct _btx_csr_script_t {
  const bt_byte* const * packets;
  bt_int packet_count;
  bt_uint revision;
} btx_csr_script_t;
```

Description

This is type btx_csr_script_t.

btx_csr_set_ps_vars_buffer_t Type

File

csr.h

C

```
typedef struct _btx_csr_set_ps_vars_buffer_t btx_csr_set_ps_vars_buffer_t;
```

Description

This is type btx_csr_set_ps_vars_buffer_t.

btx_csr_set_ps_vars_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_set_ps_vars_callback_fp)(bt_bool success, btx_csr_set_ps_vars_buffer_t* buffer);
```

Description

This is type btx_csr_set_ps_vars_callback_fp.

btx_csr_set_var_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_set_var_callback_fp)(bt_uint status, bt_uint var_id, btx_csr_var_t* var_value,
void* callback_param);
```

Description

This is type btx_csr_set_var_callback_fp.

btx_csr_strm_get_sink_t Structure

File

csr.h

C

```
typedef struct _btx_csr_strm_get_sink_s {
  btx_csr_bccmd_header_t message;
  bt_uint sink_id;
} btx_csr_strm_get_sink_t;
```

Description

This is type btx_csr_strm_get_sink_t.

btx_csr_strm_get_source_t Structure

File

csr.h

C

```
typedef struct _btx_csr_strm_get_source_s {
  btx_csr_bccmd_header_t message;
  bt_uint source_id;
} btx_csr_strm_get_source_t;
```

Description

This is type btx_csr_strm_get_source_t.

btx_csr_var_t Union

File

csr.h

C

```
typedef union _btx_csr_var_u {
  btx_csr_bccmd_header_t message;
  btx_csr_cached_temperature_t cached_temperature;
  btx_csr_rssi_acl_t rssi_acl;
  btx_csr_pio_t pio;
  btx_csr_pio_direction_mask_t pio_direction;
  btx_csr_pio_protection_mask_t pio_protection;
  btx_csr_strm_get_sink_t strm_get_sink;
  btx_csr_strm_get_source_t strm_get_source;
  btx_csr_create_operator_c_t create_operator;
  btx_csr_strm_connect_t strm_connect;
} btx_csr_var_t;
```

Description

This is type btx_csr_var_t.

btx_ti_completion_callback_fp Type

File

ti.h

C

```
typedef void (* btx_ti_completion_callback_fp)(bt_bool success, btx_ti_exec_script_buffer_t* buffer, void*
callback_param);
```

Description

This is type btx_ti_completion_callback_fp.

btx_ti_exec_script_buffer_t Type

File

ti.h

C

```
typedef struct _btx_ti_exec_script_buffer_t btx_ti_exec_script_buffer_t;
```

Description

This is type btx_ti_exec_script_buffer_t.

btx_ti_exec_script_oem_buffer_t Type

File

ti.h

C

```
typedef struct _btx_ti_exec_script_oem_buffer_t btx_ti_exec_script_oem_buffer_t;
```

Description

This is type btx_ti_exec_script_oem_buffer_t.

btx_ti_exec_script_oem_callback_fp Type

File

ti.h

C

```
typedef void (* btx_ti_exec_script_oem_callback_fp)(bt_bool success, btx_ti_exec_script_oem_buffer_t*
buffer, void* callback_param);
```

Description

This is type btx_ti_exec_script_oem_callback_fp.

btx_ti_script_t Structure

File

ti.h

C

```
typedef struct _btx_ti_script_t {
  const bt_byte* const * packets;
  bt_int packet_count;
  bt_byte fw_version_x;
  bt_byte fw_version_z;
} btx_ti_script_t;
```

Description

This is type btx_ti_script_t.

BTX_TI_A3DP_ROLE_SINK Macro

File

ti.h

C

```
#define BTX_TI_A3DP_ROLE_SINK 1
```

Description

This is macro BTX_TI_A3DP_ROLE_SINK.

BTX_TI_A3DP_ROLE_SOURCE Macro

File

ti.h

C

```
#define BTX_TI_A3DP_ROLE_SOURCE 0
```

Description

This is macro BTX_TI_A3DP_ROLE_SOURCE.

BTX_TI_AVPR_DISABLE Macro

File

ti.h

C

```
#define BTX_TI_AVPR_DISABLE 0
```

Description

This is macro BTX_TI_AVPR_DISABLE.

BTX_TI_AVPR_DO_NOT_LOAD_A3DP_CODE Macro

File

ti.h

C

```
#define BTX_TI_AVPR_DO_NOT_LOAD_A3DP_CODE 0
```

Description

This is macro BTX_TI_AVPR_DO_NOT_LOAD_A3DP_CODE.

BTX_TI_AVPR_ENABLE Macro

File

ti.h

C

```
#define BTX_TI_AVPR_ENABLE 1
```

Description

This is macro BTX_TI_AVPR_ENABLE.

BTX_TI_AVPR_LOAD_A3DP_CODE Macro

File

ti.h

C

```
#define BTX_TI_AVPR_LOAD_A3DP_CODE 1
```

Description

This is macro BTX_TI_AVPR_LOAD_A3DP_CODE.

btx_ti_get_script__CC2564B_AVPR_Init Macro

File

ti.h

C

```
#define btx_ti_get_script__CC2564B_AVPR_Init btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn
```

Description

This is macro btx_ti_get_script__CC2564B_AVPR_Init.

BTX_TI_SBC_ALLOCATION_METHOD_LOUDNESS Macro

File

ti.h

C

```
#define BTX_TI_SBC_ALLOCATION_METHOD_LOUDNESS 0
```

Description

This is macro BTX_TI_SBC_ALLOCATION_METHOD_LOUDNESS.

BTX_TI_SBC_ALLOCATION_METHOD_SNR Macro

File

ti.h

C

```
#define BTX_TI_SBC_ALLOCATION_METHOD_SNR 1
```

Description

This is macro BTX_TI_SBC_ALLOCATION_METHOD_SNR.

BTX_TI_SBC_BLOCKS_12 Macro

File

ti.h

C

```
#define BTX_TI_SBC_BLOCKS_12 12
```

Description

This is macro BTX_TI_SBC_BLOCKS_12.

BTX_TI_SBC_BLOCKS_16 Macro

File

ti.h

C

```
#define BTX_TI_SBC_BLOCKS_16 16
```

Description

This is macro BTX_TI_SBC_BLOCKS_16.

BTX_TI_SBC_BLOCKS_4 Macro

File

ti.h

C

```
#define BTX_TI_SBC_BLOCKS_4 4
```

Description

This is macro BTX_TI_SBC_BLOCKS_4.

BTX_TI_SBC_BLOCKS_8 Macro

File

ti.h

C

```
#define BTX_TI_SBC_BLOCKS_8 8
```

Description

This is macro BTX_TI_SBC_BLOCKS_8.

BTX_TI_SBC_CHANNEL_MODE_DUAL_CHANNEL Macro

File

ti.h

C

```
#define BTX_TI_SBC_CHANNEL_MODE_DUAL_CHANNEL 1
```

Description

This is macro BTX_TI_SBC_CHANNEL_MODE_DUAL_CHANNEL.

BTX_TI_SBC_CHANNEL_MODE_JOINT_STEREO Macro

File

ti.h

C

```
#define BTX_TI_SBC_CHANNEL_MODE_JOINT_STEREO 3
```

Description

This is macro BTX_TI_SBC_CHANNEL_MODE_JOINT_STEREO.

BTX_TI_SBC_CHANNEL_MODE_MONO Macro

File

ti.h

C

```
#define BTX_TI_SBC_CHANNEL_MODE_MONO 0
```

Description

This is macro BTX_TI_SBC_CHANNEL_MODE_MONO.

BTX_TI_SBC_CHANNEL_MODE_STEREO Macro

File

ti.h

C

```
#define BTX_TI_SBC_CHANNEL_MODE_STEREO 2
```

Description

This is macro BTX_TI_SBC_CHANNEL_MODE_STEREO.

BTX_TI_SBC_SAMPLE_FREQUENCY_16000 Macro

File

ti.h

C

```
#define BTX_TI_SBC_SAMPLE_FREQUENCY_16000 0
```

Description

This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_16000.

BTX_TI_SBC_SAMPLE_FREQUENCY_32000 Macro

File

ti.h

C

```
#define BTX_TI_SBC_SAMPLE_FREQUENCY_32000 1
```

Description

This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_32000.

BTX_TI_SBC_SAMPLE_FREQUENCY_44100 Macro

File

ti.h

C

```
#define BTX_TI_SBC_SAMPLE_FREQUENCY_44100 2
```

Description

This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_44100.

BTX_TI_SBC_SAMPLE_FREQUENCY_48000 Macro

File

ti.h

C

```
#define BTX_TI_SBC_SAMPLE_FREQUENCY_48000 3
```

Description

This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_48000.

CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN 0x207
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN.

CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN_ENABLE Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN_ENABLE 0x206
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN_ENABLE.

CSR_I2S_STREAM_CFG_KEY_BITS_PER_SAMPLE Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_BITS_PER_SAMPLE 0x20A
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_BITS_PER_SAMPLE.

CSR_I2S_STREAM_CFG_KEY_CROP_ENABLE Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_CROP_ENABLE 0x209
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_CROP_ENABLE.

CSR_I2S_STREAM_CFG_KEY_JUSTIFY_DELAY Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_JUSTIFY_DELAY 0x204
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_DELAY.

CSR_I2S_STREAM_CFG_KEY_JUSTIFY_FORMAT Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_JUSTIFY_FORMAT 0x203
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_FORMAT.

CSR_I2S_STREAM_CFG_KEY_JUSTIFY_RESOLUTION Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_JUSTIFY_RESOLUTION 0x208
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_RESOLUTION.

CSR_I2S_STREAM_CFG_KEY_MASTER Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_MASTER 0x202
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_MASTER.

CSR_I2S_STREAM_CFG_KEY_MCLK Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_MCLK 0x201
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_MCLK.

CSR_I2S_STREAM_CFG_KEY_POLARITY Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_POLARITY 0x205
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_POLARITY.

CSR_I2S_STREAM_CFG_KEY_RX_START_SAMPLE Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_RX_START_SAMPLE 0x20C
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_RX_START_SAMPLE.

CSR_I2S_STREAM_CFG_KEY_SYNC_RATE Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_SYNC_RATE 0x200
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_SYNC_RATE.

CSR_I2S_STREAM_CFG_KEY_TX_START_SAMPLE Macro

File

csr.h

C

```
#define CSR_I2S_STREAM_CFG_KEY_TX_START_SAMPLE 0x20B
```

Description

This is macro CSR_I2S_STREAM_CFG_KEY_TX_START_SAMPLE.

CSR_MAX_HQ_PACKET_LEN Macro

File

csr.h

C

```
#define CSR_MAX_HQ_PACKET_LEN 129
```

Description

This is macro CSR_MAX_HQ_PACKET_LEN.

CSR_MSG_TYPE_GETREQ Macro

File

csr.h

C

```
#define CSR_MSG_TYPE_GETREQ 0
```

Description

This is macro CSR_MSG_TYPE_GETREQ.

CSR_MSG_TYPE_GETRESP Macro

File

csr.h

C

```
#define CSR_MSG_TYPE_GETRESP 1
```

Description

This is macro CSR_MSG_TYPE_GETRESP.

CSR_MSG_TYPE_SETREQ Macro

File

csr.h

C

```
#define CSR_MSG_TYPE_SETREQ 2
```

Description

This is macro CSR_MSG_TYPE_SETREQ.

CSR_PCM_STREAM_CFG_KEY_LSB_FIRST Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_LSB_FIRST 0x108
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_LSB_FIRST.

CSR_PCM_STREAM_CFG_KEY_MANCHESTER Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_MANCHESTER 0x104
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_MANCHESTER.

CSR_PCM_STREAM_CFG_KEY_MANCHESTER_SLAVE Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_MANCHESTER_SLAVE 0x106
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_MANCHESTER_SLAVE.

CSR_PCM_STREAM_CFG_KEY_MASTER Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_MASTER 0x102
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_MASTER.

CSR_PCM_STREAM_CFG_KEY_MCLK Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_MCLK 0x101
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_MCLK.

CSR_PCM_STREAM_CFG_KEY_SHORT_SYNC Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_SHORT_SYNC 0x105
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_SHORT_SYNC.

CSR_PCM_STREAM_CFG_KEY_SIGN_EXTEND Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_SIGN_EXTEND 0x107
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_SIGN_EXTEND.

CSR_PCM_STREAM_CFG_KEY_SLOT_COUNT Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_SLOT_COUNT 0x103
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_SLOT_COUNT.

CSR_PCM_STREAM_CFG_KEY_SYNC_RATE Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_SYNC_RATE 0x100
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_SYNC_RATE.

CSR_PCM_STREAM_CFG_KEY_TX_TRISTATE Macro

File

csr.h

C

```
#define CSR_PCM_STREAM_CFG_KEY_TX_TRISTATE 0x109
```

Description

This is macro CSR_PCM_STREAM_CFG_KEY_TX_TRISTATE.

CSR_VARID_CAPABILITY_DOWNLOAD_INDICATION Macro

File

csr.h

C

```
#define CSR_VARID_CAPABILITY_DOWNLOAD_INDICATION 0x101b
```

Description

This is macro CSR_VARID_CAPABILITY_DOWNLOAD_INDICATION.

CSR_VARID_CREATE_OPERATOR_C Macro

File

csr.h

C

```
#define CSR_VARID_CREATE_OPERATOR_C 0x5075
```

Description

This is macro CSR_VARID_CREATE_OPERATOR_C.

CSR_VARID_CREATE_OPERATOR_P Macro

File

csr.h

C

```
#define CSR_VARID_CREATE_OPERATOR_P 0x5076
```

Description

This is macro CSR_VARID_CREATE_OPERATOR_P.

CSR_VARID_DESTROY_OPERATORS Macro

File

csr.h

C

```
#define CSR_VARID_DESTROY_OPERATORS 0x5073
```

Description

This is macro CSR_VARID_DESTROY_OPERATORS.

CSR_VARID_DSPMANAGER_CONFIG_REQUEST Macro

File

csr.h

C

```
#define CSR_VARID_DSPMANAGER_CONFIG_REQUEST 0x101c
```

Description

This is macro CSR_VARID_DSPMANAGER_CONFIG_REQUEST.

CSR_VARID_MESSAGE_FROM_OPERATOR Macro

File

csr.h

C

```
#define CSR_VARID_MESSAGE_FROM_OPERATOR 0x101a
```

Description

Select Variable definitions

CSR_VARID_START_OPERATORS Macro

File

csr.h

C

```
#define CSR_VARID_START_OPERATORS 0x5070
```

Description

This is macro CSR_VARID_START_OPERATORS.

CSR_VARID_STOP_OPERATORS Macro

File

csr.h

C

```
#define CSR_VARID_STOP_OPERATORS 0x5071
```

Description

This is macro CSR_VARID_STOP_OPERATORS.

CSR_VARID_STREAM_DRAINED_NOTIFICATION Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_DRAINED_NOTIFICATION 0x101d
```

Description

This is macro CSR_VARID_STREAM_DRAINED_NOTIFICATION.

CSR_VARID_STREAM_TRANSFORM_DISCONNECT Macro

File

csr.h

C

```
#define CSR_VARID_STREAM_TRANSFORM_DISCONNECT 0x486d
```

Description

This is macro CSR_VARID_STREAM_TRANSFORM_DISCONNECT.

PSKEY_BLE_DEFAULT_TX_POWER Macro

File

csr.h

C

```
#define PSKEY_BLE_DEFAULT_TX_POWER 0x22c8
```

Description

This is macro PSKEY_BLE_DEFAULT_TX_POWER.

PSKEY_DEEP_SLEEP_EXTERNAL_CLOCK_SOURCE_PIO Macro

File

csr.h

C

```
#define PSKEY_DEEP_SLEEP_EXTERNAL_CLOCK_SOURCE_PIO 0x2579
```

Description

This is macro PSKEY_DEEP_SLEEP_EXTERNAL_CLOCK_SOURCE_PIO.

PSKEY_H_HC_FC_MAX_SCO_PKT_LEN Macro

File

csr.h

C

```
#define PSKEY_H_HC_FC_MAX_SCO_PKT_LEN 0x0012
```

Description

This is macro PSKEY_H_HC_FC_MAX_SCO_PKT_LEN.

PSKEY_H_HC_FC_MAX_SCO_PKTS Macro

File

csr.h

C

```
#define PSKEY_H_HC_FC_MAX_SCO_PKTS 0x0014
```

Description

This is macro PSKEY_H_HC_FC_MAX_SCO_PKTS.

PSKEY_PCM_CLOCK_RATE Macro

File

csr.h

C

```
#define PSKEY_PCM_CLOCK_RATE 0x23bc
```

Description

This is macro PSKEY_PCM_CLOCK_RATE.

PSKEY_PCM_CONFIG32 Macro

File

csr.h

C

```
#define PSKEY_PCM_CONFIG32 0x01b3
```

Description

This is macro PSKEY_PCM_CONFIG32.

PSKEY_PCM_FORMAT Macro

File

csr.h

C

```
#define PSKEY_PCM_FORMAT 0x01b6
```

Description

This is macro PSKEY_PCM_FORMAT.

PSKEY_PCM_SYNC_RATE Macro

File

csr.h

C

```
#define PSKEY_PCM_SYNC_RATE 0x23c3
```

Description

This is macro PSKEY_PCM_SYNC_RATE.

PSKEY_PCM_USE_LOW_JITTER_MODE Macro

File

csr.h

C

```
#define PSKEY_PCM_USE_LOW_JITTER_MODE 0x23c9
```

Description

This is macro PSKEY_PCM_USE_LOW_JITTER_MODE.

PSKEY_UART_CONFIG_H4DS Macro

File

csr.h

C

```
#define PSKEY_UART_CONFIG_H4DS 0x01cb
```

Description

This is macro PSKEY_UART_CONFIG_H4DS.

BT_LOG_LEVEL_A2DP_PAKCET Macro

File

bt_log.h

C

```
#define BT_LOG_LEVEL_A2DP_PAKCET 10
```

Description

This is macro BT_LOG_LEVEL_A2DP_PAKCET.

DCRH Macro

File

patch.h

C

```
#define DCRH(block_count, val_len, seq) \
    0x01,         /* HCI command packet indicator.*/ \
    0x00, 0xfc,   /* HCI opcode { OGF=0x3f OCF=0x00 } */ \
    /*     OGF=0x3f implies a manufacturer-sepcific command. */ \
    /* Parameter length. */ \
    1 + (5 + 1 + val_len) * 2, \
    0xc3,         /* CSR payload descriptor: */ \
    /*    bit 7: Last fragment = 1 */ \
    /*    bit 6: First fragment = 1 */ \
    /*    bits 5-0 Channel ID = 2 (BCCMD) */ \
    W(0x0001),    /* GETRESP message */ \
    W(5+1+val_len), /* length*/ \
    W(seq),       /* sequence number */ \
    W(0x101c),    /* Variable ID of DSP manager config request */ \
    W(0x0000),    /* Status (must be 0 in SETREQ) */ \
    \
    W(block_count)     /* Number of configuration blocks */
```

Description

Number of configuration blocks

btx_csr_bccmd_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_bccmd_callback_fp)(btx_csr_bccmd_header_t* message, void* cb_param);
```

Description

This is type btx_csr_bccmd_callback_fp.

btx_csr_bccmd_listener_t Type

File

csr.h

C

```
typedef struct _btx_csr_bccmd_listener_t btx_csr_bccmd_listener_t;
```

Description

This is type btx_csr_bccmd_listener_t.

btx_csr_create_operator_c_t Structure

File

csr.h

C

```
typedef struct _btx_csr_create_operator_c_s {
  btx_csr_bccmd_header_t message;
  bt_uint operator_id;
  bt_uint item_count;
  bt_uint skip_count;
  bt_uint skip_flag;
} btx_csr_create_operator_c_t;
```

Description

This is type btx_csr_create_operator_c_t.

btx_csr_exec_hq_script_buffer_t Type

File

csr.h

C

```
typedef struct _btx_csr_exec_hq_script_buffer_t btx_csr_exec_hq_script_buffer_t;
```

Description

This is type btx_csr_exec_hq_script_buffer_t.

btx_csr_exec_hq_script_callback_fp Type

File

csr.h

C

```
typedef void (* btx_csr_exec_hq_script_callback_fp)(bt_bool success, btx_csr_exec_hq_script_buffer_t*
buffer);
```

Description

This is type btx_csr_exec_hq_script_callback_fp.

btx_csr_get_script_fp Type

File

csr.h

C

```
typedef const btx_csr_script_t* (* btx_csr_get_script_fp)(void);
```

Description

This is type btx_csr_get_script_fp.

btx_csr_strm_connect_t Structure

File

csr.h

C

```
typedef struct _btx_csr_strm_connect_s {
  btx_csr_bccmd_header_t message;
  bt_uint transform_id;
} btx_csr_strm_connect_t;
```

Description

This is type btx_csr_strm_connect_t.

btx_ti_codec_config_t Structure

File

ti.h

C

```
typedef struct _btx_ti_codec_config_s {
  bt_uint clock_rate;
  bt_byte clock_direction;
  bt_ulong frame_sync_freq;
  bt_uint frame_sync_duty_cycle;
  bt_byte frame_sync_edge;
  bt_byte frame_sync_polarity;
  bt_byte frame_sync_multiplier;
  bt_uint ch1_data_out_size;
  bt_uint ch1_data_out_offset;
  bt_byte ch1_data_out_edge;
  bt_uint ch1_data_in_size;
  bt_uint ch1_data_in_offset;
  bt_byte ch1_data_in_edge;
  bt_uint ch2_data_out_size;
  bt_uint ch2_data_out_offset;
  bt_byte ch2_data_out_edge;
  bt_uint ch2_data_in_size;
  bt_uint ch2_data_in_offset;
  bt_byte ch2_data_in_edge;
} btx_ti_codec_config_t;
```

Description

This is type btx_ti_codec_config_t.

_btx_csr_bccmd_listener_t Structure

File

csr.h

C

```
struct _btx_csr_bccmd_listener_t {
  btx_csr_bccmd_listener_t* next_listener;
  btx_csr_bccmd_callback_fp callback;
  void* callback_param;
};
```

Description

This is record _btx_csr_bccmd_listener_t.

_btx_csr_exec_hq_script_buffer_t Structure

File

csr.h

C

```
struct _btx_csr_exec_hq_script_buffer_t {
  const btx_csr_script_t* script;
  bt_int current_packet;
  bt_byte packet[CSR_MAX_HQ_PACKET_LEN];
  bt_bool success;
  btx_csr_exec_hq_script_callback_fp callback;
  void* callback_param;
};
```

Description

This is record _btx_csr_exec_hq_script_buffer_t.

PSKEY_LC_DEFAULT_TX_POWER Macro

File

csr.h

C

```
#define PSKEY_LC_DEFAULT_TX_POWER 0x0021
```

Description

This is macro PSKEY_LC_DEFAULT_TX_POWER.

PSKEY_LC_MAX_TX_POWER Macro

File

csr.h

C

```
#define PSKEY_LC_MAX_TX_POWER 0x0017
```

Description

This is macro PSKEY_LC_MAX_TX_POWER.

PSKEY_LC_MAX_TX_POWER_NO_RSSI Macro

File

csr.h

C

```
#define PSKEY_LC_MAX_TX_POWER_NO_RSSI 0x002d
```

Description

This is macro PSKEY_LC_MAX_TX_POWER_NO_RSSI.

Utility Functions

bt_alloc_buffer Function

File

buffer.h

C

```
void* bt_alloc_buffer(bt_buffer_mgr_p pmgr);
```

Description

This is function bt_alloc_buffer.

bt_at_parse_fragment Function

File

at_parser.h

C

```
void bt_at_parse_fragment(bt_at_parser_t* parser, bt_byte* data, bt_uint len);
```

Description

This is function bt_at_parse_fragment.

bt_at_parser_reset Function

File

at_parser.h

C

```
void bt_at_parser_reset(bt_at_parser_t* parser);
```

Description

This is function bt_at_parser_reset.

bt_free_buffer Function

File

buffer.h

C

```
void bt_free_buffer(bt_buffer_mgr_p pmgr, void* data);
```

Description

This is function bt_free_buffer.

bt_get_buffer Function

File

buffer.h

C

```
void* bt_get_buffer(bt_buffer_mgr_p pmgr, bt_int index);
```

Description

This is function bt_get_buffer.

bt_get_buffer_header Function

File

buffer.h

C

```
bt_buffer_header_t* bt_get_buffer_header(bt_buffer_mgr_p pmgr, bt_int index);
```

Description

This is function bt_get_buffer_header.

bt_get_buffer_index Function

File

buffer.h

C

```
bt_int bt_get_buffer_index(bt_buffer_mgr_p pmgr, void* data);
```

Description

This is function bt_get_buffer_index.

bt_init_buffer_mgr Function

File

buffer.h

C

```
bt_bool bt_init_buffer_mgr(bt_buffer_mgr_p pmgr, bt_int num_buffers, bt_int buffer_size,
bt_buffer_header_t* buffer_headers, void* buffer_data);
```

Description

This is function bt_init_buffer_mgr.

bt_q_add Function

File

queue.h

C

```
void bt_q_add(bt_queue_element_t** phead, void* element);
```

Description

This is function bt_q_add.

bt_q_get Function

File

queue.h

C

```
void* bt_q_get(bt_queue_element_t* head, bt_int index);
```

Description

This is function bt_q_get.

bt_q_get_head Function

File

queue.h

C

```
void* bt_q_get_head(bt_queue_element_t** phead, bt_bool remove);
```

Description

This is function bt_q_get_head.

bt_q_get_length Function

File

queue.h

C

```
bt_int bt_q_get_length(bt_queue_element_t* phead);
```

Description

This is function bt_q_get_length.

bt_q_get_next Function

File

queue.h

C

```
void* bt_q_get_next(void* element);
```

Description

This is function bt_q_get_next.

bt_q_push Function

File

queue.h

C

```
void bt_q_push(bt_queue_element_t** phead, void* element);
```

Description

This is function bt_q_push.

bt_q_remove Function

File

queue.h

C

```
void* bt_q_remove(bt_queue_element_t** phead, void* pdata);
```

Description

This is function bt_q_remove.

bt_q_remove_by_idx Function

File

queue.h

C

```
void* bt_q_remove_by_idx(bt_queue_element_t* head, bt_int index);
```

Description

This is function bt_q_remove_by_idx.

bt_vcard_parse_fragment Function

File

vcard_parser.h

C

```
void bt_vcard_parse_fragment(bt_vcard_parser_t* parser, bt_byte* data, bt_uint len);
```

Description

This is function bt_vcard_parse_fragment.

bt_vcard_parser_reset Function

File

vcard_parser.h

C

```
void bt_vcard_parser_reset(bt_vcard_parser_t* parser);
```

Description

This is function bt_vcard_parser_reset.

bt_xml_parse_fragment Function

File

xml_parser.h

C

```
void bt_xml_parse_fragment(bt_xml_parser_t* parser, bt_byte_cp data, bt_uint len);
```

Description

This is function bt_xml_parse_fragment.

bt_xml_parser_reset Function

File

xml_parser.h

C

```
void bt_xml_parser_reset(bt_xml_parser_t* parser);
```

Description

This is function bt_xml_parser_reset.

_compact_buffer Function

File

bufferutils.h

C

```
void _compact_buffer(bt_byte_p pdst, bt_uint offset, bt_byte_cp psrc, bt_uint len);
```

Description

This is function _compact_buffer.

_expand_buffer Function

File

bufferutils.h

C

```
void _expand_buffer(bt_byte_p pdst, bt_byte_cp psrc, bt_uint len, bt_uint offset);
```

Description

This is function _expand_buffer.

_read_bdaddr Function

File

bufferutils.h

C

```
void _read_bdaddr(bt_bdaddr_p pbdaddr, bt_byte_cp buffer);
```

Description

This is function _read_bdaddr.

_readb Function

File

bufferutils.h

C

```
bt_bool _readb(bt_byte_p pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readb.

_readi Function

File

bufferutils.h

C

```
bt_bool _readi(bt_int_p pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readi.

_readin Function

File

bufferutils.h

C

```
bt_bool _readin(bt_int_p pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readin.

_readl Function

File

bufferutils.h

C

```
bt_bool _readl(bt_long_p pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readl.

_readln Function

File

bufferutils.h

C

```
bt_bool _readln(bt_long_p pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readln.

_readui Function

File

bufferutils.h

C

```
bt_bool _readui(bt_uint_ptr pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readui.

_readuin Function

File

bufferutils.h

C

```
bt_bool _readuin(bt_uint_ptr pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readuin.

_readul Function

File

bufferutils.h

C

```
bt_bool _readul(bt_ulong_ptr pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readul.

_readuln Function

File

bufferutils.h

C

```
bt_bool _readuln(bt_ulong_ptr pvalue, bt_byte_cp pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _readuln.

_str2ulong_dec Function

File

bufferutils.h

C

```
bt_ulong _str2ulong_dec(const char* s);
```

Description

This is function _str2ulong_dec.

_to_lower_case Function

File

bufferutils.h

C

```
void _to_lower_case(bt_byte* str, bt_uint len);
```

Description

This is function _to_lower_case.

_ulong2str Function

File

bufferutils.h

C

```
const char* _ulong2str(bt_ulong data);
```

Description

This is function _ulong2str.

_ulong2str_dec Function

File

bufferutils.h

C

```
const char* _ulong2str_dec(bt_ulong data);
```

Description

This is function _ulong2str_dec.

_write_bdaddr Function

File

bufferutils.h

C

```
void _write_bdaddr(bt_bdaddr_cp pbdaddr, bt_byte_p buffer);
```

Description

This is function _write_bdaddr.

_writeb Function

File

bufferutils.h

C

```
bt_bool _writeb(bt_byte value, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writeb.

_writei Function

File

bufferutils.h

C

```
bt_bool _writei(bt_int value, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writei.

_writein Function

File

bufferutils.h

C

```
bt_bool _writein(bt_int value, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writein.

_writel Function

File

bufferutils.h

C

```
bt_bool _writel(bt_long value, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writel.

_writeln Function

File

bufferutils.h

C

```
bt_bool _writeln(bt_long value, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writeln.

_writes Function

File

bufferutils.h

C

```
bt_bool _writes(const char* pstr, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writes.

_writesx Function

File

bufferutils.h

C

```
bt_bool _writesx(const char* pstr, bt_int str_len, bt_byte_p pb, bt_int buf_len, bt_int_p offset);
```

Description

This is function _writesx.

_zero_memory Function

File

bufferutils.h

C

```
void _zero_memory(void* ptr, size_t len);
```

Description

This is function _zero_memory.

_bt_memcpy Function

File

bufferutils.h

C

```
void _bt_memcpy(bt_byte_p pdst, bt_uint dst_offset, bt_byte_cp psrc, bt_uint len, bt_uint offset);
```

Description

This is function _bt_memcpy.

_is_empty_str Function

File

bufferutils.h

C

```
bt_bool _is_empty_str(const char* s);
```

Description

This is function _is_empty_str.

_str2ulong Function

File

bufferutils.h

C

```
bt_ulong _str2ulong(const char* s);
```

Description

This is function _str2ulong.

Utility Data Types and Constants

AT_EVT_CMD_CODE Macro

File

at_parser.h

C

```
#define AT_EVT_CMD_CODE 3
```

Description

This is macro AT_EVT_CMD_CODE.

AT_EVT_CMD_COMPLETED Macro

File

at_parser.h

C

```
#define AT_EVT_CMD_COMPLETED 6
```

Description

This is macro AT_EVT_CMD_COMPLETED.

AT_EVT_CMD_PARAM Macro

File

at_parser.h

C

```
#define AT_EVT_CMD_PARAM 5
```

Description

This is macro AT_EVT_CMD_PARAM.

AT_EVT_CMD_READ_CODE Macro

File

at_parser.h

C

```
#define AT_EVT_CMD_READ_CODE 4
```

Description

This is macro AT_EVT_CMD_READ_CODE.

AT_EVT_ERROR Macro

File

at_parser.h

C

```
#define AT_EVT_ERROR 2
```

Description

This is macro AT_EVT_ERROR.

AT_EVT_OK Macro

File

at_parser.h

C

```
#define AT_EVT_OK 1
```

Description

This is macro AT_EVT_OK.

AT_EVT_RING Macro

File

at_parser.h

C

```
#define AT_EVT_RING 0
```

Description

This is macro AT_EVT_RING.

ATCMD_BUFFER_LEN Macro

File

at_parser.h

C

```
#define ATCMD_BUFFER_LEN 20
```

Description

This is macro ATCMD_BUFFER_LEN.

bt_max Macro

File

bufferutils.h

C

```
#define bt_max(a,b) (((a) > (b)) ? (a) : (b))
```

Description

This is macro bt_max.

bt_min Macro

File

bufferutils.h

C

```
#define bt_min(a,b) (((a) < (b)) ? (a) : (b))
```

Description

This is macro bt_min.

BUFFER_HDR_LEN Macro

File

buffer.h

C

```
#define BUFFER_HDR_LEN offsetof(bt_buffer, data)
```

Description

This is macro BUFFER_HDR_LEN.

BUFFER_STATE_FREE Macro

File

buffer.h

C

```
#define BUFFER_STATE_FREE 0
```

Description

for offsetof

BUFFER_STATE_USED Macro

File

buffer.h

C

```
#define BUFFER_STATE_USED 1
```

Description

This is macro BUFFER_STATE_USED.

max Macro

File

bufferutils.h

C

```
#define max(a,b) (using_max_is_dangerous = 0, use_bt_max_instead)
```

Description

This is macro max.

min Macro

File

bufferutils.h

C

```
#define min(a,b) (using_min_is_dangerous = 0, use_bt_min_instead)
```

Description

This is macro min.

VCARD_CALL_TYPE_DIALED Macro

File

vcard_parser.h

C

```
#define VCARD_CALL_TYPE_DIALED 0
```

Description

This is macro VCARD_CALL_TYPE_DIALED.

VCARD_CALL_TYPE_MISSED Macro

File

vcard_parser.h

C

```
#define VCARD_CALL_TYPE_MISSED 2
```

Description

This is macro VCARD_CALL_TYPE_MISSED.

VCARD_CALL_TYPE_RECEIVED Macro

File

vcard_parser.h

C

```
#define VCARD_CALL_TYPE_RECEIVED 1
```

Description

This is macro VCARD_CALL_TYPE_RECEIVED.

VCARD_EVT_PROPERTY_PARAM Macro

File

vcard_parser.h

C

```
#define VCARD_EVT_PROPERTY_PARAM 3
```

Description

This is macro VCARD_EVT_PROPERTY_PARAM.

VCARD_EVT_PROPERTY_STARTED Macro

File

vcard_parser.h

C

```
#define VCARD_EVT_PROPERTY_STARTED 2
```

Description

This is macro VCARD_EVT_PROPERTY_STARTED.

VCARD_EVT_PROPERTY_VALUE Macro

File

vcard_parser.h

C

```
#define VCARD_EVT_PROPERTY_VALUE 4
```

Description

This is macro VCARD_EVT_PROPERTY_VALUE.

VCARD_EVT_VCARD_ENDED Macro

File

vcard_parser.h

C

```
#define VCARD_EVT_VCARD_ENDED 1
```

Description

This is macro VCARD_EVT_VCARD_ENDED.

VCARD_EVT_VCARD_STARTED Macro

File

vcard_parser.h

C

```
#define VCARD_EVT_VCARD_STARTED 0
```

Description

This is macro VCARD_EVT_VCARD_STARTED.

VCARD_PARAM_CALL_TYPE Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_CALL_TYPE 0x80
```

Description

This is macro VCARD_PARAM_CALL_TYPE.

VCARD_PARAM_CELL Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_CELL 7
```

Description

This is macro VCARD_PARAM_CELL.

VCARD_PARAM_DIALED Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_DIALED 4
```

Description

This is macro VCARD_PARAM_DIALED.

VCARD_PARAM_ENCODING Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_ENCODING 1
```

Description

This is macro VCARD_PARAM_ENCODING.

VCARD_PARAM_HOME Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_HOME 8
```

Description

This is macro VCARD_PARAM_HOME.

VCARD_PARAM_LANGUAGE Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_LANGUAGE 3
```

Description

This is macro VCARD_PARAM_LANGUAGE.

VCARD_PARAM_MISSED Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_MISSED 6
```

Description

This is macro VCARD_PARAM_MISSED.

VCARD_PARAM_RECEIVED Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_RECEIVED 5
```

Description

This is macro VCARD_PARAM_RECEIVED.

VCARD_PARAM_TYPE Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_TYPE 0
```

Description

This is macro VCARD_PARAM_TYPE.

VCARD_PARAM_VALUE Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_VALUE 2
```

Description

This is macro VCARD_PARAM_VALUE.

VCARD_PARAM_WORK Macro

File

vcard_parser.h

C

```
#define VCARD_PARAM_WORK 9
```

Description

This is macro VCARD_PARAM_WORK.

VCARD_PARSER_STATE_READ_PARAM_NAME Macro

File

vcard_parser.h

C

```
#define VCARD_PARSER_STATE_READ_PARAM_NAME 1
```

Description

This is macro VCARD_PARSER_STATE_READ_PARAM_NAME.

VCARD_PARSER_STATE_READ_PARAM_VALUE Macro

File

vcard_parser.h

C

```
#define VCARD_PARSER_STATE_READ_PARAM_VALUE 3
```

Description

This is macro VCARD_PARSER_STATE_READ_PARAM_VALUE.

VCARD_PARSER_STATE_READ_TYPE_NAME Macro

File

vcard_parser.h

C

```
#define VCARD_PARSER_STATE_READ_TYPE_NAME 0
```

Description

This is macro VCARD_PARSER_STATE_READ_TYPE_NAME.

VCARD_PARSER_STATE_READ_TYPE_VALUE Macro

File

vcard_parser.h

C

```
#define VCARD_PARSER_STATE_READ_TYPE_VALUE 2
```

Description

This is macro VCARD_PARSER_STATE_READ_TYPE_VALUE.

VCARD_PARSER_STATE_SKIP_PARAM Macro

File

vcard_parser.h

C

```
#define VCARD_PARSER_STATE_SKIP_PARAM 4
```

Description

This is macro VCARD_PARSER_STATE_SKIP_PARAM.

VCARD_PARSER_STATE_SKIP_TYPE Macro

File

vcard_parser.h

C

```
#define VCARD_PARSER_STATE_SKIP_TYPE 5
```

Description

This is macro VCARD_PARSER_STATE_SKIP_TYPE.

VCARD_PROPERTY_ADR Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_ADR 7 // Delivery Address
```

Description

Delivery Address

VCARD_PROPERTY_AGENT Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_AGENT 17 // vCard of Person Representing
```

Description

vCard of Person Representing

VCARD_PROPERTY_BDAY Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_BDAY 6 // Birthday
```

Description

Birthday

VCARD_PROPERTY_BEGIN Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_BEGIN 0
```

Description

This is macro VCARD_PROPERTY_BEGIN.

VCARD_PROPERTY_CATEGORIES Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_CATEGORIES 26 // Categories
```

Description

Categories

VCARD_PROPERTY_CLASS Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_CLASS 28 // Class information
```

Description

Class information

VCARD_PROPERTY_EMAIL Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_EMAIL 10 // Electronic Mail Address
```

Description

Electronic Mail Address

VCARD_PROPERTY_END Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_END 1
```

Description

This is macro VCARD_PROPERTY_END.

VCARD_PROPERTY_FN Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_FN 3 // Formatted Name
```

Description

Formatted Name

VCARD_PROPERTY_GEO Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_GEO 13 // Geographic Position
```

Description

Geographic Position

VCARD_PROPERTY_KEY Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_KEY 24 // Public Encryption Key
```

Description

Public Encryption Key

VCARD_PROPERTY_LABEL Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_LABEL 8 // Delivery
```

Description

Delivery

VCARD_PROPERTY_LOGO Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_LOGO 16 // Organization Logo
```

Description

Organization Logo

VCARD_PROPERTY_MAILER Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_MAILER 11 // Electronic Mail
```

Description

Electronic Mail

VCARD_PROPERTY_N Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_N 4 // Structured Presentation of Name
```

Description

Structured Presentation of Name

VCARD_PROPERTY_NICKNAME Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_NICKNAME 25 // Nickname
```

Description

Nickname

VCARD_PROPERTY_NOTE Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_NOTE 19 // Comments
```

Description

Comments

VCARD_PROPERTY_ORG Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_ORG 18 // Name of Organization
```

Description

Name of Organization

VCARD_PROPERTY_PHOTO Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_PHOTO 5 // Associated Image or Photo
```

Description

Associated Image or Photo

VCARD_PROPERTY_PROID Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_PROID 27 // Product ID
```

Description

Product ID

VCARD_PROPERTY_REV Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_REV 20 // Revision
```

Description

Revision

VCARD_PROPERTY_ROLE Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_ROLE 15 // Role within the Organization
```

Description

Role within the Organization

VCARD_PROPERTY_SORT_STRING Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_SORT_STRING 29 // String used for sorting operations
```

Description

String used for sorting operations

VCARD_PROPERTY_SOUND Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_SOUND 21 // Pronunciation of Name
```

Description

Pronunciation of Name

VCARD_PROPERTY_TEL Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_TEL 9 // Telephone Number
```

Description

Telephone Number

VCARD_PROPERTY_TITLE Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_TITLE 14 // Job
```

Description

Job

VCARD_PROPERTY_TZ Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_TZ 12 // Time Zone
```

Description

Time Zone

VCARD_PROPERTY_UID Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_UID 23 // Unique ID
```

Description

Unique ID

VCARD_PROPERTY_URL Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_URL 22 // Uniform Resource Locator
```

Description

Uniform Resource Locator

VCARD_PROPERTY_VERSION Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_VERSION 2 // vCard Version
```

Description

vCard Version

VCARD_PROPERTY_X_IRMC_CALL_DATETIME Macro

File

vcard_parser.h

C

```
#define VCARD_PROPERTY_X_IRMC_CALL_DATETIME 30 // Time stamp
```

Description

Time stamp

XML_EVT_ATTR_NAME Macro

File

xml_parser.h

C

```
#define XML_EVT_ATTR_NAME 2
```

Description

This is macro XML_EVT_ATTR_NAME.

XML_EVT_ATTR_VALUE Macro

File

xml_parser.h

C

```
#define XML_EVT_ATTR_VALUE 3
```

Description

This is macro XML_EVT_ATTR_VALUE.

XML_EVT_TAG_ENDED Macro

File

xml_parser.h

C

```
#define XML_EVT_TAG_ENDED 1
```

Description

This is macro XML_EVT_TAG_ENDED.

XML_EVT_TAG_STARTED Macro

File

xml_parser.h

C

```
#define XML_EVT_TAG_STARTED 0
```

Description

This is macro XML_EVT_TAG_STARTED.

XML_PARSER_STATE_READ_ATTR_NAME Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_READ_ATTR_NAME 5
```

Description

This is macro XML_PARSER_STATE_READ_ATTR_NAME.

XML_PARSER_STATE_READ_ATTR_VALUE Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_READ_ATTR_VALUE 8
```

Description

This is macro XML_PARSER_STATE_READ_ATTR_VALUE.

XML_PARSER_STATE_READ_TAG_NAME Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_READ_TAG_NAME 1
```

Description

This is macro XML_PARSER_STATE_READ_TAG_NAME.

XML_PARSER_STATE_READ_TAG_VALUE Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_READ_TAG_VALUE 7
```

Description

This is macro XML_PARSER_STATE_READ_TAG_VALUE.

XML_PARSER_STATE_SKIP_ATTR Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_SKIP_ATTR 9
```

Description

This is macro XML_PARSER_STATE_SKIP_ATTR.

XML_PARSER_STATE_SKIP_COMMENTS Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_SKIP_COMMENTS 11
```

Description

This is macro XML_PARSER_STATE_SKIP_COMMENTS.

XML_PARSER_STATE_SKIP_PROC_INST Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_SKIP_PROC_INST 10
```

Description

This is macro XML_PARSER_STATE_SKIP_PROC_INST.

XML_PARSER_STATE_SKIP_TAG Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_SKIP_TAG 3
```

Description

This is macro XML_PARSER_STATE_SKIP_TAG.

XML_PARSER_STATE_WAIT_ATTR_NAME Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_WAIT_ATTR_NAME 4
```

Description

This is macro XML_PARSER_STATE_WAIT_ATTR_NAME.

XML_PARSER_STATE_WAIT_ATTR_VALUE Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_WAIT_ATTR_VALUE 6
```

Description

This is macro XML_PARSER_STATE_WAIT_ATTR_VALUE.

XML_PARSER_STATE_WAIT_TAG_CLOSE Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_WAIT_TAG_CLOSE 2
```

Description

This is macro XML_PARSER_STATE_WAIT_TAG_CLOSE.

XML_PARSER_STATE_WAIT_TAG_NAME Macro

File

xml_parser.h

C

```
#define XML_PARSER_STATE_WAIT_TAG_NAME 0
```

Description

This is macro XML_PARSER_STATE_WAIT_TAG_NAME.

_readbn Macro

File

bufferutils.h

C

```
#define _readbn _readb
```

Description

This is macro _readbn.

_readui Macro

File

bufferutils.h

C

```
#define _readui(pvalue, pb, buf_len, offset) _readi((bt_int_p)pvalue, pb, buf_len, offset)
```

Description

This is macro _readui.

_readuin Macro

File

bufferutils.h

C

```
#define _readuin(pvalue, pb, buf_len, offset) _readin((bt_int_p)pvalue, pb, buf_len, offset)
```

Description

This is macro _readuin.

_readul Macro

File

bufferutils.h

C

```
#define _readul(pvalue, pb, buf_len, offset) _readl((bt_long_p)pvalue, pb, buf_len, offset)
```

Description

This is macro _readul.

_readuln Macro

File

bufferutils.h

C

```
#define _readuln(pvalue, pb, buf_len, offset) _readln((bt_long_p)pvalue, pb, buf_len, offset)
```

Description

This is macro _readuln.

_writebn Macro

File

bufferutils.h

C

```
#define _writebn _writeb
```

Description

This is macro _writebn.

bt_at_parser_callback_pf Type

File

at_parser.h

C

```
typedef void (* bt_at_parser_callback_pf)(struct _bt_at_parser_t* parser, bt_byte evt, void* evt_param,
void* callback_param);
```

Description

This is type bt_at_parser_callback_pf.

bt_at_parser_t Structure

File

at_parser.h

C

```
typedef struct _bt_at_parser_t {
  bt_byte state;
  bt_byte buffer[ATCMD_BUFFER_LEN + 1];
  bt_byte write_pos;
  bt_byte param_read_state;
  bt_byte param_read_nesting_level;
  bt_at_parser_callback_pf callback;
  void* callback_param;
} bt_at_parser_t;
```

Description

This is type bt_at_parser_t.

bt_bt_buffer_header_p Structure

File

buffer.h

C

```
typedef struct _bt_buffer_header_t {
  bt_int state;
} bt_buffer_header_t, * bt_bt_buffer_header_p;
```

Description

This is type bt_bt_buffer_header_p.

bt_buffer_header_t Structure

File

buffer.h

C

```
typedef struct _bt_buffer_header_t {
  bt_int state;
} bt_buffer_header_t, * bt_bt_buffer_header_p;
```

Description

This is type bt_buffer_header_t.

bt_buffer_mgr_p Structure

File

buffer.h

C

```
typedef struct _bt_buffer_mgr_t {
  bt_int max_buffers;
  bt_int buffer_size;
  bt_buffer_header_t* headers;
  bt_byte_p buffers;
} bt_buffer_mgr_t, * bt_buffer_mgr_p;
```

Description

This is type bt_buffer_mgr_p.

bt_buffer_mgr_t Structure

File

buffer.h

C

```
typedef struct _bt_buffer_mgr_t {
  bt_int max_buffers;
  bt_int buffer_size;
  bt_buffer_header_t* headers;
  bt_byte_p buffers;
} bt_buffer_mgr_t, * bt_buffer_mgr_p;
```

Description

This is type bt_buffer_mgr_t.

bt_packet_assembler_fp Type

File

packet.h

C

```
typedef bt_int (* bt_packet_assembler_fp)(struct _bt_packet_t* packet, bt_byte* buffer, bt_int buffer_len);
```

Description

This is type bt_packet_assembler_fp.

bt_packet_t Structure

File

packet.h

C

```
typedef struct _bt_packet_t {
  bt_packet_assembler_fp packet_assembler;
  const bt_byte* data;
  bt_int len;
  bt_byte state;
  bt_int data_pos;
  void* param;
} bt_packet_t;
```

Description

This is type bt_packet_t.

bt_queue_element_t Structure

File

queue.h

C

```
typedef struct _bt_queue_element_t {
  struct _bt_queue_element_t* next;
} bt_queue_element_t;
```

Description

This is type bt_queue_element_t.

bt_vcard_evt_prop_param_t Structure

File

vcard_parser.h

C

```
typedef struct _bt_vcard_evt_prop_param_t {
  bt_byte param_id;
  bt_byte* param_value;
  bt_byte param_len;
} bt_vcard_evt_prop_param_t;
```

Description

This is type bt_vcard_evt_prop_param_t.

bt_vcard_evt_prop_t Structure

File

vcard_parser.h

C

```
typedef struct _bt_vcard_evt_prop_t {
  bt_byte prop_id;
  bt_byte* prop_value;
  bt_byte value_len;
  bt_bool final;
} bt_vcard_evt_prop_t;
```

Description

This is type bt_vcard_evt_prop_t.

bt_vcard_parser_callback_fp Type

File

vcard_parser.h

C

```
typedef void (* bt_vcard_parser_callback_fp)(struct _bt_vcard_parser_t* parser, bt_byte evt, void*
evt_param, void* callback_param);
```

Description

This is type bt_vcard_parser_callback_fp.

bt_vcard_parser_t Structure

File

vcard_parser.h

C

```
typedef struct _bt_vcard_parser_t {
  bt_byte state;
  bt_byte* buffer;
  bt_uint buffer_size;
  bt_byte write_pos;
  bt_byte prev_c;
  bt_byte cur_prop_id;
  bt_byte cur_param_id;
  bt_bool vcard_started;
  bt_vcard_parser_callback_fp callback;
  void* callback_param;
} bt_vcard_parser_t;
```

Description

This is type bt_vcard_parser_t.

bt_xml_evt_attribute_t Structure

File

xml_parser.h

C

```
typedef struct _bt_xml_evt_attribute_t {
  bt_byte* name;
  bt_byte len;
} bt_xml_evt_attribute_t;
```

Description

This is type bt_xml_evt_attribute_t.

bt_xml_evt_attribute_value_t Structure

File

xml_parser.h

C

```
typedef struct _bt_xml_evt_attribute_value_t {
  bt_byte* value;
  bt_byte len;
} bt_xml_evt_attribute_value_t;
```

Description

This is type bt_xml_evt_attribute_value_t.

bt_xml_evt_tag_started_t Structure

File

xml_parser.h

C

```
typedef struct _bt_xml_evt_tag_started_t {
  bt_byte* name;
  bt_byte len;
} bt_xml_evt_tag_started_t;
```

Description

This is type bt_xml_evt_tag_started_t.

bt_xml_parser_callback_pf Type

File

xml_parser.h

C

```
typedef void (* bt_xml_parser_callback_pf)(struct _bt_xml_parser_t* parser, bt_byte evt, void* evt_param,
void* callback_param);
```

Description

This is type bt_xml_parser_callback_pf.

bt_xml_parser_t Structure

File

xml_parser.h

C

```
typedef struct _bt_xml_parser_t {
  bt_byte state;
  bt_byte* buffer;
  bt_uint buffer_size;
  bt_byte write_pos;
  bt_byte prev_c;
  bt_byte quote_c;
  bt_xml_parser_callback_pf callback;
  void* callback_param;
} bt_xml_parser_t;
```

Description

This is type bt_xml_parser_t.

Files

Files

Name	Description
a2dp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
a2dp_codec_aac.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
a2dp_codec_mpeg.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
a2dp_codec_sbc.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
a2dp_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avctp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avctp_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avctp_packet.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avctp_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avdtp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avdtp_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avdtp_control.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avdtp_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avrcp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avrcp_command.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avrcp_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
avrcp_config_event_handlers.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
avrcp_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
at_parser.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_bdaddr.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
baseband.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_hcitr.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_log.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_oem.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_oem_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_signal.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_std.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_storage.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_system.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_timer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bt_types.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
buffer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
bufferutils.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
channel.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
chmanager.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
cmdbuffer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

command.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
csr.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
frame_buffer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
gap.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
hci.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_cmd_buffer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_cmd_queue.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_config_event_handlers.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_conn_state.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_ctrl_state.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_data_buffer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_data_queue.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_eir.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_errors.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_evt_handlers.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_le.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
hci_linkkey_buffer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_signal.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_transport.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hci_utils.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hcitr_3wire.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hcitr_bcsp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hcitr_packet.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
hcitr_tih4.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hcitr_uart.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
lm.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_command_queue.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_config_handlers.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_eretr.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_fixed_channel.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
l2cap_packet.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

l2cap_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_psm.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_signal.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_test.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
l2cap_timer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
packet.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
patch.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
queue.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_cmd_queue.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm_mgr.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm_mx.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm_signal.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
rfcomm_timer.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
sbc.h	Provides SBC Decoder Library interface services.
sdp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

sdp_client.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
sdp_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
sdp_packet.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
sdp_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
sdp_utils.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
spp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
spp_config.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
spp_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
ssp.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

ssp_event.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
ssp_event_handler.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
ti.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
ti_private.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
types.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.
vcard_parser.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.
xml_parser.h	Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved. SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements. Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

Description

a2dp.h

Functions

	Name	Description
	bt_a2dp_find_server	This is function bt_a2dp_find_server.
	bt_a2dp_find_sink	brief Find sink ingroup a2dp details This function looks for a sink on a remote device specified by c deviceAddress and, if found, returns features supported by the sink. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer... more

	bt_a2dp_find_source	brief Find source ingroup a2dp details This function looks for a source on a remote device specified by c deviceAddress and, if found, returns features supported by the source. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer... more
	bt_a2dp_get_mgr	brief Return a pointer to an instance of the A2DP manager. ingroup a2dp details This function returns a pointer to an instance of the A@DP manager. There is only one instance of the manager allocated by the stack. The pointer is passed as the first parameter to all A2DP functions.
	bt_a2dp_init	brief Initialize the A2DP layer. ingroup a2dp details This function initializes the A2DP layer of the stack. It must be called prior to any other A2DP function can be called.
	bt_a2dp_open_and_start_stream	brief Open & start a stream ingroup a2dp details Opening a stream involves sending 3 requests to a remote device - "set configuration", "open stream" and "start stream". Each event generates its own event which must be handled and acted accordingly by the application. To make the use of API easier dotstack combines all these requests in one request called "open & start stream". dotstack sends necessary requests in a proper sequence, handles responses and generates only one event (A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED) at the end. If any of the individual requests has failed the event's parameter bt_a2dp_event_t::open_and_start_stream_completed is populated with... more
	bt_a2dp_register_aac_codec	brief Register default AAC codec ingroup a2dp details This function adds AAC codec implemented by dotstack to the list of known codecs. For more information about codecs see description of ::bt_avdtp_register_codec. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use AAC codec it must call this function when it is initializing. note dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration. param mgr A2DP manager.
	bt_a2dp_register_callback	brief Register a A2DP application callback. ingroup a2dp details In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values: @arg A2DP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg A2DP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg A2DP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs.... more
	bt_a2dp_register_mpeg_codec	brief Register default MPEG codec ingroup a2dp details This function adds MPEG codec implemented by dotstack to the list of known codecs. For more information about codecs see description of ::bt_avdtp_register_codec. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use MPEG-1,2 codec it must call this function when it is initializing. note dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration. param mgr A2DP manager.
	bt_a2dp_set_configuration	This is function bt_a2dp_set_configuration.
	bt_a2dp_start	brief Start the A2DP layer. ingroup a2dp details This function makes the A2DP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required. param mgr AVDTP manager. return li c TRUE if the function succeeds. li c FALSE otherwise.

Macros

	Name	Description
	__A2DP_H	This is macro __A2DP_H.
	A2DP_EVT_ABORT_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.
	A2DP_EVT_ABORT_STREAM_REQUESTED	< This event is generated when a local device received "abort stream" request.

	A2DP_EVT_CLOSE_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "close stream" request.
	A2DP_EVT_CLOSE_STREAM_REQUESTED	< This event is generated when a local device received "close stream" request.
	A2DP_EVT_CTRL_CHANNEL_CONNECTED	< This event is generated when a control channel between two AVDTP entities has been established.
	A2DP_EVT_CTRL_CHANNEL_DISCONNECTED	< This event is generated when a control channel between two AVDTP entities has been terminated.
	A2DP_EVT_CTRL_CONNECTION_FAILED	< This event is generated when a local device failed to create a control channel between two AVDTP entities.
	A2DP_EVT_DISCOVER_SEP_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "discover" request.
	A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.
	A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.
	A2DP_EVT_MEDIA_PACKET_RECEIVED	< This event is generated when a local device received a media packet.
	A2DP_EVT_MEDIA_PACKET_SEND_FAILED	< This event is generated when a local device failed to send a media packet.
	A2DP_EVT_MEDIA_PACKET_SENT	< This event is generated when a local device sent a media packet.
	A2DP_EVT_NOTHING	This is macro A2DP_EVT_NOTHING.
	A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED	< This event is generated when a local device completed "open and start" request.
	A2DP_EVT_OPEN_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "open stream" request.
	A2DP_EVT_OPEN_STREAM_REQUESTED	< This event is generated when a local device received "open stream" request.
	A2DP_EVT_RECONFIGURE_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.
	A2DP_EVT_RECONFIGURE_STREAM_REQUESTED	< This event is generated when a local device received "change stream configuration" request.
	A2DP_EVT_SEP_CAPABILITIES_RECEIVED	< This event is generated when a local device received a positive response to a "get SEP capabilities" request.
	A2DP_EVT_SEP_INFO_RECEIVED	< This event is generated for each SEP contained in a positive response to a "discover" request.
	A2DP_EVT_SET_STREAM_CONFIGURATION	This is macro A2DP_EVT_SET_STREAM_CONFIGURATION.
	A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.
	A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED	< This event is generated when a local device received "set stream configuration" request.
	A2DP_EVT_START_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "start stream" request.
	A2DP_EVT_START_STREAM_REQUESTED	< This event is generated when a local device received "start stream" request.
	A2DP_EVT_STREAM_ABORTED	< This event is generated when a local device has successfully aborted a stream. < This event follows the A2DP_EVT_ABORT_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream abortion was initiated by the local device.
	A2DP_EVT_STREAM_CLOSED	< This event is generated when a local device has successfully closed a stream. < This event follows the A2DP_EVT_CLOSE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream closing was initiated by the local device.
	A2DP_EVT_STREAM_CONFIGURATION_RECEIVED	< This event is generated when a local device received a positive response to a "get stream configuration" request.

	A2DP_EVT_STREAM_CONFIGURED	< This event is generated when a local device has successfully configured a stream. < This event follows the A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the upper layer has accepted it. < This event is not generated if stream configuration was initiated by the local device.
	A2DP_EVT_STREAM_OPENED	< This event is generated when a local device has successfully opened a stream. < This event follows the A2DP_EVT_OPEN_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream opening was initiated by the local device.
	A2DP_EVT_STREAM_RECONFIGURED	< This event is generated when a local device has successfully reconfigured a stream. < This event follows the A2DP_EVT_RECONFIGURE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream reconfiguration was initiated by the local device.
	A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.
	A2DP_EVT_STREAM_STARTED	< This event is generated when a local device has successfully started a stream. < This event follows the A2DP_EVT_START_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream starting was initiated by the local device.
	A2DP_EVT_STREAM_SUSPENDED	< This event is generated when a local device has successfully suspended a stream. < This event follows the A2DP_EVT_SUSPEND_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream suspension was initiated by the local device.
	A2DP_EVT_SUSPEND_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.
	A2DP_EVT_SUSPEND_STREAM_REQUESTED	< This event is generated when a local device received "suspend stream" request.
	A2DP_MANAGER_STATE_CONNECTING	This is macro A2DP_MANAGER_STATE_CONNECTING.
	A2DP_MANAGER_STATE_IDLE	This is macro A2DP_MANAGER_STATE_IDLE.
	A2DP_SINK_FEATURE_AMPLIFIER	< Amplifier
	A2DP_SINK_FEATURE_HEADPHONE	< Headphone
	A2DP_SINK_FEATURE_RECORDER	< Recorder
	A2DP_SINK_FEATURE_SPEAKER	< Speaker
	A2DP_SOURCE_FEATURE_MICROPHONE	< Mic
	A2DP_SOURCE_FEATURE_MIXER	< Mixer
	A2DP_SOURCE_FEATURE_PLAYER	< Player
	A2DP_SOURCE_FEATURE_TUNER	< Tuner
	bt_a2dp_abort_stream	brief Suspend a stream. ingroup a2dp details This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE. As a result of this operation the A2DP_EVT_ABORT_STREAM_COMPLETED event will be generated. This operation cannot be rejected. The p evt_param.abort_stream_requested.err_code is always == AVDTP_ERROR_SUCCESS. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.
	bt_a2dp_add_media_rx_buffer	brief Add a media packet buffer to a receive queue ingroup a2dp details The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more

	bt_a2dp_add_media_tx_buffer	brief Add a media packet buffer to a send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. When the packet has been successfully sent a A2DP_EVT_MEDIA_PACKET_SENT is generated. Otherwise a A2DP_EVT_MEDIA_PACKET_SEND_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as A2DP has removed it from the queue and gave up... more
	bt_a2dp_call_codec	This is macro bt_a2dp_call_codec.
	bt_a2dp_cancel_listen	brief Cancel listening for incoming connections. ingroup a2dp details This function removes a SEP from a list of SEPS which a stream can use for incoming requests. param mgr A2DP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_a2dp_clear_media_tx_queue	brief Clear send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. The consumer can remove all packets from the queue before they have been sent to a remote device by calling ::bt_a2dp_clear_media_tx_queue. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails... more
	bt_a2dp_close_stream	brief Close a stream. ingroup a2dp details This function tries to close a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN or AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the A2DP_EVT_CLOSE_STREAM_COMPLETED event will be generated. If the stream has been closed the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == the error code sent by the remote. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds,... more
	bt_a2dp_connect	brief Connect to a remote device. ingroup a2dp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP_EVT_CTRL_CHANNEL_CONNECTED or A2DP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr A2DP manager. param remote_addr The address of a remote device. return li c TRUE if connection establishment has been started. li c FALSE... more

	bt_a2dp_connect_ex	brief Connect to a remote device. ingroup a2dp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP_EVT_CTRL_CHANNEL_CONNECTED or A2DP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr A2DP manager. param remote_addr The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following... more
	bt_a2dp_create_stream	brief Create a stream. ingroup a2dp details This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort. param mgr A2DP manager. return li c Stream handle if the function succeeds. li c 0 otherwise.
	bt_a2dp_destroy_stream	brief Destroy a stream. ingroup a2dp details This function frees memory used by the stream. The stream has to exist and be in the "idle" state for this function to succeed. I.e. the stream has to be closed or aborted before this function can be called. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_a2dp_disconnect	brief Disconnect from a remote device. ingroup a2dp details This function closes a control and transport channels on all streams associated with the remote device specified by the p remote_addr. As a result of this operation the following events will be generated: @arg A2DP_EVT_MEDIA_PACKET_RECEIVED: if a stream's receive queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg A2DP_EVT_MEDIA_PACKET_SENT: if a stream's send queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg A2DP_EVT_STREAM_CLOSED: this event is generate if a stream is in... more
	bt_a2dp_discover	brief Discover SEPs on a remote device. ingroup a2dp details This function asks the remote device to send a list of all available SEPs. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_INFO_RECEIVED: this event is generated for every SEP received from the remote device. the p evt_param.sep_info_received contains SEP information. @arg A2DP_EVT_DISCOVER_COMPLETED: this event is generated after the last A2DP_EVT_SEP_INFO_RECEIVED if the remote accepted the request and the p evt_param.discover_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.discover_completed.err_code == the error code sent by the remote. param mgr A2DP... more
	bt_a2dp_find_codec	brief Find a codec ingroup a2dp details A2DP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make our implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of A2DP has to register a callback function (one per codec type) for each codec it wishes to support. That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in... more

	bt_a2dp_get_all_capabilities	brief Get remote SEP capabilities. ingroup a2dp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after A2DP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr A2DP manager.... more
	bt_a2dp_get_capabilities	brief Get remote SEP capabilities. ingroup a2dp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg A2DP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after A2DP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr A2DP manager.... more
	bt_a2dp_get_configuration	This is macro bt_a2dp_get_configuration.
	bt_a2dp_get_hci_connection	brief Get HCI connection for a stream ingroup a2dp details This function returns a pointer to a structure that describes an HCI connection a stream is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect. param mgr A2DP manager. param strm_handle Stream handle. return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a stream specified by the p... more
	bt_a2dp_get_stream_codec_config	brief Get the configuration of the codec currently used with the stream. ingroup a2dp details This function returns a pointer to a structure that contains configuration of the codec currently used with the stream. The structure returned depends on the codec. The dotstack defines structures only for SBC, MPEG-1,2 and MPEG-2,4 AAC codecs: @arg SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h) @arg MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h) @arg MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h) param mgr A2DP manager. param strm_handle Stream handle. return li The codec's configuration if strm_handle specifies a valid stream and the stream is in one of the following... more

	bt_a2dp_get_stream_codec_type	brief Get the type of the codec currently used with the stream. ingroup a2dp details This function returns the type of the codec currently used with the stream. param mgr A2DP manager. param strm_handle Stream handle. return @arg The type of the codec if strm_handle specifies a valid stream and the stream is in one of the following states: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING @arg The result will be one of the following values: AVDTP_CODEC_TYPE_SBC: SBC AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP... more
	bt_a2dp_get_stream_config	brief Get stream's configuration. ingroup a2dp details This function returns a pointer to a structure holding current configuration of the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The stream's configuration if strm_handle specifies a valid stream and the stream is in one of the following state: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING li NULL otherwise.
	bt_a2dp_get_stream_direction	This is macro bt_a2dp_get_stream_direction.
	bt_a2dp_get_stream_local_sep_id	brief Get stream's local SEP ID. ingroup a2dp details This function returns the ID of the local SEP associated with the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The ID of the local SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_a2dp_get_stream_remote_address	brief Get stream's remote BT address. ingroup a2dp details This function returns the address of the remote device associated with the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The address of the remote device if strm_handle specifies a valid stream. li NULL otherwise.
	bt_a2dp_get_stream_remote_sep_id	brief Get stream's remote SEP ID. ingroup a2dp details This function returns the ID of the remote SEP associated with the stream. param mgr A2DP manager. param strm_handle Stream handle. return li The ID of the remote SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_a2dp_get_stream_state	brief Get local stream state. ingroup a2dp details This function returns local state of a stream specified by the p strm_handle. No request is sent to the remote party. param mgr A2DP manager. param strm_handle Stream handle. return The state of the stream. The result will be one of the following values: @arg AVDTP_STREAM_STATE_IDLE: The stream is idle. This can mean two things. The stream specified by p strm_handle does not exist or the stream is closed. @arg AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS: The stream is opening transport channels. @arg AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS: The stream is closing transport channels. @arg AVDTP_STREAM_STATE_CONFIGURED: The... more

	bt_a2dp_listen	brief Listen for incoming connections. ingroup a2dp details This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream. param mgr A2DP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if... more
	bt_a2dp_open_stream	This is macro bt_a2dp_open_stream.
	bt_a2dp_reconfigure_stream	brief Reconfigure stream. ingroup a2dp details This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the A2DP_EVT_STREAM_RECONFIGURE_COMPLETED event will be generated. If reconfiguration was a success the p evt_param.stream_reconfigure_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.stream_reconfigure_completed.err_code == the error code sent by the remote. param mgr A2DP manager. param strm_handle Stream handle. param caps New stream configuration. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be... more
	bt_a2dp_register_sink	brief Register a Sink SEP with the local A2DP manager. ingroup a2dp details This function is used to add a sink SEP to a list of SEPs supported by the local A2DP entity. param mgr A2DP manager. param caps The capabilities of a SEP. return li c ID of a SEP if the function succeeds. li c FALSE otherwise.
	bt_a2dp_register_source	brief Register a Source SEP with the local A2DP manager. ingroup a2dp details This function is used to add a source SEP to a list of SEPs supported by the local A2DP entity. param mgr A2DP manager. param caps The capabilities of a SEP. return li c ID of a SEP if the function succeeds. li c FALSE otherwise.
	bt_a2dp_remove_media_rx_buffer	brief Remove a media packet buffer from a receive queue ingroup a2dp details The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more
	bt_a2dp_remove_media_tx_buffer	brief Remove a media packet buffer from a send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling ::bt_a2dp_remove_media_tx_buffer. param mgr A2DP manager. param strm_handle Stream handle. param buffer Pointer to a structure... more
	bt_a2dp_report_delay	This is macro bt_a2dp_report_delay.

	bt_a2dp_set_media_tx_queue_limit	brief Set limit on the send queue ingroup a2dp details When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. By default the send queue can contain unlimited number of packets. The consumer can set a limit on how many packets are held in the queue. In this case when new packet is added to the queue and the length of... more
	bt_a2dp_start_stream	brief Start a stream. ingroup a2dp details This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the A2DP_EVT_START_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.start_stream_requested.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the p evt_param.start_stream_requested.err_code == the error code sent... more
	bt_a2dp_suspend_stream	brief Suspend a stream. ingroup a2dp details This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the A2DP_EVT_SUSPEND_STREAM_COMPLETED event will be generated. If the stream has been suspended the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == the error code sent by the remote. param mgr A2DP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the... more

Structures

	Name	Description
	_bt_a2dp_evt_open_and_start_stream_completed_s	brief Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event ingroup a2dp details A pointer to this structure is passed to the A2DP application callback as a valid member of the bt_a2dp_event_t union - bt_a2dp_event_t::open_and_start_stream_completed - when A2DP completed a "open & start stream" request.
	_bt_a2dp_mgr_t	brief A2DP manager. ingroup a2dp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with ::bt_a2dp_get_mgr.
	bt_a2dp_evt_open_and_start_stream_completed_t	brief Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event ingroup a2dp details A pointer to this structure is passed to the A2DP application callback as a valid member of the bt_a2dp_event_t union - bt_a2dp_event_t::open_and_start_stream_completed - when A2DP completed a "open & start stream" request.

Types

	Name	Description
	bt_a2dp_event_t	This is type bt_a2dp_event_t.

	bt_a2dp_find_server_callback_fp	brief Notify the application of the result of searching for a remote A2DP entity (sourse or sink) ingroup a2dp details This function is called by the A2DP layer when searching for an A2DP entity on a remote device has completed. param supported_features Features supported by a remote A2DP entity. param found c TRUE if an A2DP entity has been found on the remote device. c FALSE otherwise. param param pointer to arbitrary data passed to the bt_a2dp_find_source() or bt_a2dp_find_sink function through its c callback_param parameter.
	bt_a2dp_mgr_callback_fp	brief A2DP application callback. ingroup a2dp details In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call that function whenever a new event has been generated. param mgr A2DP manager. param evt A2DP event. The event can be one of the following values: @arg A2DP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg A2DP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg A2DP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local... more
	bt_a2dp_mgr_t	This is type bt_a2dp_mgr_t.

Unions

	Name	Description
	_bt_a2dp_event_u	brief Parameter to an application callback. ingroup a2dp details This union is used to pass event specific data to the A2DP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

a2dp_codec_aac.h

Functions

	Name	Description
	bt_a2dp_aac_codec_handler	This is function bt_a2dp_aac_codec_handler.

Macros

	Name	Description
	__A2DP_CODEC_AAC_H	This is macro __A2DP_CODEC_AAC_H.
	AAC_CHANNELS_1	This is macro AAC_CHANNELS_1.
	AAC_CHANNELS_2	This is macro AAC_CHANNELS_2.
	AAC_CHANNELS_ALL	This is macro AAC_CHANNELS_ALL.
	AAC_OBJECT_TYPE_MPEG_2_LC	This is macro AAC_OBJECT_TYPE_MPEG_2_LC.
	AAC_OBJECT_TYPE_MPEG_4_LC	This is macro AAC_OBJECT_TYPE_MPEG_4_LC.
	AAC_OBJECT_TYPE_MPEG_4_LTP	This is macro AAC_OBJECT_TYPE_MPEG_4_LTP.
	AAC_OBJECT_TYPE_MPEG_4_SCALABLE	This is macro AAC_OBJECT_TYPE_MPEG_4_SCALABLE.
	AAC_SAMPLING_FREQUENCY_11025	This is macro AAC_SAMPLING_FREQUENCY_11025.
	AAC_SAMPLING_FREQUENCY_12000	This is macro AAC_SAMPLING_FREQUENCY_12000.
	AAC_SAMPLING_FREQUENCY_16000	This is macro AAC_SAMPLING_FREQUENCY_16000.
	AAC_SAMPLING_FREQUENCY_22050	This is macro AAC_SAMPLING_FREQUENCY_22050.
	AAC_SAMPLING_FREQUENCY_24000	This is macro AAC_SAMPLING_FREQUENCY_24000.
	AAC_SAMPLING_FREQUENCY_32000	This is macro AAC_SAMPLING_FREQUENCY_32000.
	AAC_SAMPLING_FREQUENCY_44100	This is macro AAC_SAMPLING_FREQUENCY_44100.
	AAC_SAMPLING_FREQUENCY_48000	This is macro AAC_SAMPLING_FREQUENCY_48000.
	AAC_SAMPLING_FREQUENCY_64000	This is macro AAC_SAMPLING_FREQUENCY_64000.
	AAC_SAMPLING_FREQUENCY_8000	This is macro AAC_SAMPLING_FREQUENCY_8000.
	AAC_SAMPLING_FREQUENCY_88200	This is macro AAC_SAMPLING_FREQUENCY_88200.
	AAC_SAMPLING_FREQUENCY_96000	This is macro AAC_SAMPLING_FREQUENCY_96000.

	AAC_SAMPLING_FREQUENCY_ALL	This is macro AAC_SAMPLING_FREQUENCY_ALL.
	AAC_VBR_NOT_SUPPORTED	This is macro AAC_VBR_NOT_SUPPORTED.
	AAC_VBR_SUPPORTED	This is macro AAC_VBR_SUPPORTED.

Structures

	Name	Description
	_bt_a2dp_aac_config_t	This is type bt_a2dp_aac_config_t.
	bt_a2dp_aac_config_t	This is type bt_a2dp_aac_config_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

a2dp_codec_mpeg.h

Functions

	Name	Description
	bt_a2dp_mpeg_codec_handler	This is function bt_a2dp_mpeg_codec_handler.

Macros

	Name	Description
	__A2DP_CODEC_MPEG_H	This is macro __A2DP_CODEC_MPEG_H.
	MPEG_BITRATE_0000	This is macro MPEG_BITRATE_0000.
	MPEG_BITRATE_0001	This is macro MPEG_BITRATE_0001.
	MPEG_BITRATE_0010	This is macro MPEG_BITRATE_0010.
	MPEG_BITRATE_0011	This is macro MPEG_BITRATE_0011.
	MPEG_BITRATE_0100	This is macro MPEG_BITRATE_0100.
	MPEG_BITRATE_0101	This is macro MPEG_BITRATE_0101.
	MPEG_BITRATE_0110	This is macro MPEG_BITRATE_0110.
	MPEG_BITRATE_0111	This is macro MPEG_BITRATE_0111.
	MPEG_BITRATE_1000	This is macro MPEG_BITRATE_1000.
	MPEG_BITRATE_1001	This is macro MPEG_BITRATE_1001.
	MPEG_BITRATE_1010	This is macro MPEG_BITRATE_1010.
	MPEG_BITRATE_1011	This is macro MPEG_BITRATE_1011.
	MPEG_BITRATE_1100	This is macro MPEG_BITRATE_1100.
	MPEG_BITRATE_1101	This is macro MPEG_BITRATE_1101.
	MPEG_BITRATE_1110	This is macro MPEG_BITRATE_1110.
	MPEG_BITRATE_ALL	This is macro MPEG_BITRATE_ALL.
	MPEG_CHANNEL_MODE_ALL	This is macro MPEG_CHANNEL_MODE_ALL.
	MPEG_CHANNEL_MODE_DUAL_CHANNEL	This is macro MPEG_CHANNEL_MODE_DUAL_CHANNEL.
	MPEG_CHANNEL_MODE_JOINT_STEREO	This is macro MPEG_CHANNEL_MODE_JOINT_STEREO.
	MPEG_CHANNEL_MODE_MONO	This is macro MPEG_CHANNEL_MODE_MONO.
	MPEG_CHANNEL_MODE_STEREO	This is macro MPEG_CHANNEL_MODE_STEREO.
	MPEG_CRC_PROTECTION_NOT_SUPPORTED	This is macro MPEG_CRC_PROTECTION_NOT_SUPPORTED.
	MPEG_CRC_PROTECTION_SUPPORTED	This is macro MPEG_CRC_PROTECTION_SUPPORTED.
	MPEG_LAYER_1	This is macro MPEG_LAYER_1.
	MPEG_LAYER_2	This is macro MPEG_LAYER_2.
	MPEG_LAYER_3	This is macro MPEG_LAYER_3.
	MPEG_LAYER_ALL	This is macro MPEG_LAYER_ALL.
	MPEG_MPF_1	This is macro MPEG_MPF_1.
	MPEG_MPF_2	This is macro MPEG_MPF_2.
	MPEG_SAMPLING_FREQUENCY_16000	This is macro MPEG_SAMPLING_FREQUENCY_16000.
	MPEG_SAMPLING_FREQUENCY_22050	This is macro MPEG_SAMPLING_FREQUENCY_22050.
	MPEG_SAMPLING_FREQUENCY_24000	This is macro MPEG_SAMPLING_FREQUENCY_24000.
	MPEG_SAMPLING_FREQUENCY_32000	This is macro MPEG_SAMPLING_FREQUENCY_32000.

	MPEG_SAMPLING_FREQUENCY_44100	This is macro MPEG_SAMPLING_FREQUENCY_44100.
	MPEG_SAMPLING_FREQUENCY_48000	This is macro MPEG_SAMPLING_FREQUENCY_48000.
	MPEG_SAMPLING_FREQUENCY_ALL	This is macro MPEG_SAMPLING_FREQUENCY_ALL.
	MPEG_VBR_NOT_SUPPORTED	This is macro MPEG_VBR_NOT_SUPPORTED.
	MPEG_VBR_SUPPORTED	This is macro MPEG_VBR_SUPPORTED.

Structures

	Name	Description
	_bt_a2dp_mpeg_config_t	This is type bt_a2dp_mpeg_config_t.
	bt_a2dp_mpeg_config_t	This is type bt_a2dp_mpeg_config_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

a2dp_codec_sbc.h

Functions

	Name	Description
	bt_a2dp_sbc_codec_handler	This is function bt_a2dp_sbc_codec_handler.

Macros

	Name	Description
	__A2DP_CODEC_SBC_H	This is macro __A2DP_CODEC_SBC_H.
	SBC_ALLOCATION_METHOD_ALL	This is macro SBC_ALLOCATION_METHOD_ALL.
	SBC_ALLOCATION_METHOD_LOUDNESS	This is macro SBC_ALLOCATION_METHOD_LOUDNESS.
	SBC_ALLOCATION_METHOD_SNR	This is macro SBC_ALLOCATION_METHOD_SNR.
	SBC_BLOCK_LENGTH_12	This is macro SBC_BLOCK_LENGTH_12.
	SBC_BLOCK_LENGTH_16	This is macro SBC_BLOCK_LENGTH_16.
	SBC_BLOCK_LENGTH_4	This is macro SBC_BLOCK_LENGTH_4.
	SBC_BLOCK_LENGTH_8	This is macro SBC_BLOCK_LENGTH_8.
	SBC_BLOCK_LENGTH_ALL	This is macro SBC_BLOCK_LENGTH_ALL.
	SBC_CHANNEL_MODE_ALL	This is macro SBC_CHANNEL_MODE_ALL.
	SBC_CHANNEL_MODE_DUAL_CHANNEL	This is macro SBC_CHANNEL_MODE_DUAL_CHANNEL.
	SBC_CHANNEL_MODE_JOINT_STEREO	This is macro SBC_CHANNEL_MODE_JOINT_STEREO.
	SBC_CHANNEL_MODE_MONO	This is macro SBC_CHANNEL_MODE_MONO.
	SBC_CHANNEL_MODE_STEREO	This is macro SBC_CHANNEL_MODE_STEREO.
	SBC_SAMPLING_FREQUENCY_16000	This is macro SBC_SAMPLING_FREQUENCY_16000.
	SBC_SAMPLING_FREQUENCY_32000	This is macro SBC_SAMPLING_FREQUENCY_32000.
	SBC_SAMPLING_FREQUENCY_44100	This is macro SBC_SAMPLING_FREQUENCY_44100.
	SBC_SAMPLING_FREQUENCY_48000	This is macro SBC_SAMPLING_FREQUENCY_48000.
	SBC_SAMPLING_FREQUENCY_ALL	This is macro SBC_SAMPLING_FREQUENCY_ALL.
	SBC_SUBBANDS_4	This is macro SBC_SUBBANDS_4.
	SBC_SUBBANDS_8	This is macro SBC_SUBBANDS_8.
	SBC_SUBBANDS_ALL	This is macro SBC_SUBBANDS_ALL.

Structures

	Name	Description
	_bt_a2dp_sbc_config_t	This is type bt_a2dp_sbc_config_t.
	_bt_a2dp_sbc_packet_info_t	This is type bt_a2dp_sbc_packet_info_t.
	bt_a2dp_sbc_config_t	This is type bt_a2dp_sbc_config_t.
	bt_a2dp_sbc_packet_info_t	This is type bt_a2dp_sbc_packet_info_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

a2dp_private.h

Functions

	Name	Description
	_bt_a2dp_avdtp_mgr_callback	This is function _bt_a2dp_avdtp_mgr_callback.

Macros

	Name	Description
	__A2DP_PRIVATE_H	This is macro __A2DP_PRIVATE_H.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avctp.h

Functions

	Name	Description
	bt_avctp_cancel_command	brief Cancel a command message. ingroup avctp details If a message has not yet been sent to the remote device, it can be canceled (i.e. removed from send queue) by calling this function. param channel AVCTP channel. param tran_id Transaction Id. This value is obtained by calling bt_avctp_send_command. return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.
	bt_avctp_cancel_listen	brief Cancel listening for incoming connections. ingroup avctp details This function stops listening for incoming connections on the specified channel. param channel AVCTP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avctp_cancel_response	brief Cancel a response message. ingroup avctp details If a message has not yet been sent to the remote device, it can be canceled (i.e. removed from send queue) by calling this function. param channel AVCTP channel. param tran_id Transaction Id. This value is obtained by calling bt_avctp_send_command. return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.
	bt_avctp_connect	brief Connect to a remote device. ingroup avctp details This function establishes a connection to a remote device specified by the p remote_address. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVCTP callback. The events generated will either be AVCTP_EVT_CHANNEL_CONNECTED or AVCTP_EVT_CONNECTION_FAILED. param channel AVCTP channel. param remote_address The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following values:... more
	bt_avctp_create_channel	brief Allocate AVCTP channel ingroup avctp details This function allocates a new incoming AVCTP channel. The channel is intended to be used to accept a connection from a remote device. There can be only one channel for each combination of c profile_id and c psm. param mgr AVCTP manager. param profile_id Profile Id param psm The PSM on which the underlying L2CAP channel will listen and accept incoming connections. param l2cap_mode Underlying L2CAP channel mode. This currently can only be CMODE_BASIC. return li A pointer to the new AVCTP channel if the function succeeds. li c NULL otherwise.

	bt_avctp_create_outgoing_channel	brief Allocate AVCTP channel ingroup avctp details This function allocates a new outgoing AVCTP channel. The channel is intended to be used to create a connection to a remote device. There can be multiple channels with the same c profile_id and c psm. param mgr AVCTP manager. param profile_id Profile Id param psm The PSM on which the underlying L2CAP channel will listen and accept incoming connections. param l2cap_mode Underlying L2CAP channel mode. This currently can only be CMODE_BASIC. return li A pointer to the new AVCTP channel if the function succeeds. li c NULL otherwise.
	bt_avctp_destroy_channel	brief Destroy AVCTP channel. ingroup avctp details This function frees memory used by the channel. The channel has to exist and be in the "idle" state for this function to succeed. I.e. the channel has to be disconnected before this function can be called. param channel AVCTP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avctp_disconnect	brief Disconnect from a remote device. ingroup avctp details This function closes a connection to a remote device. param channel AVCTP channel. return li c TRUE if disconnection has been started. li c FALSE otherwise. No events will be generated.
	bt_avctp_get_channel_remote_address	brief Get channel's remote BT address. ingroup avctp details This function returns the address of the remote device associated with the channel. param channel AVCTP channel. return li The address of the remote device if channel is connected. li NULL otherwise.
	bt_avctp_get_channel_state	brief Get channel state ingroup avctp details This function return current state of the specified channel param channel AVCTP channel. return li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING
	bt_avctp_get_hci_connection	brief Get HCI connection for a channel ingroup avctp details This function returns a pointer to a structure that describes an HCI connection a channel is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect. param channel AVCTP channel. return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a channel specified by the p channel parameter li does... more
	bt_avctp_get_mgr	brief Return a pointer to an instance of the AVCTP manager. ingroup avctp details This function returns a pointer to an instance of the AVCTP manager. There is only one instance of the manager allocated by the stack.
	bt_avctp_init	brief Initialize the AVCTP layer. ingroup avctp details This function initializes the AVCTP layer of the stack. It must be called prior to any other AVCTP function can be called.
	bt_avctp_listen	brief Listen for incoming connections. ingroup avctp details This function enables incoming connections on the specified AVCTP channel. param channel AVCTP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avctp_send_command	brief Send a command message to a remote device. ingroup avctp details This function sends a command message to a remote device. param channel AVCTP channel. param data Message body. param data_len Message body length. param tran_id Pointer to a bt_byte where AVRCP will write transaction id assigned to the message. return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.
	bt_avctp_send_response	brief Send a response message to a remote device. ingroup avctp details This function sends a response message to a remote device. param channel AVCTP channel. param tran_id Transaction Id. This value is obtained by calling bt_avctp_send_command. param data Message body. param data_len Message body length. return li c TRUE if the function succeeds. li c FALSE otherwise. No events will be generated.

	bt_avctp_set_callback	brief Register a AVCTP application callback. ingroup avctp details In order to be notified of various events a consumer of the AVCTP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values: @arg AVCTP_EVT_CHANNEL_CONNECTED Channel connected. @arg AVCTP_EVT_CHANNEL_DISCONNECTED Channel disconnected. @arg AVCTP_EVT_CONNECTION_FAILED Channel connection failed (generated only if connection has been initiated by the local device). @arg AVCTP_EVT_COMMAND_RECEIVED Command received. @arg AVCTP_EVT_RESPONE_RECEIVED Response received. @arg AVCTP_EVT_COMMAND_SENT Command sent. @arg AVCTP_EVT_RESPONSE_SENT Response... more
	bt_avctp_start	This is function bt_avctp_start.

Macros

	Name	Description
	__AVCTP_H	This is macro __AVCTP_H.
	AVCTP_CHANNEL_FLAG_LISTENING	This is macro AVCTP_CHANNEL_FLAG_LISTENING.
	AVCTP_CHANNEL_FLAG_SENDING	This is macro AVCTP_CHANNEL_FLAG_SENDING.
	AVCTP_CHANNEL_STATE_CONNECTED	This is macro AVCTP_CHANNEL_STATE_CONNECTED.
	AVCTP_CHANNEL_STATE_CONNECTING	This is macro AVCTP_CHANNEL_STATE_CONNECTING.
	AVCTP_CHANNEL_STATE_DISCONNECTING	This is macro AVCTP_CHANNEL_STATE_DISCONNECTING.
	AVCTP_CHANNEL_STATE_FREE	This is macro AVCTP_CHANNEL_STATE_FREE.
	AVCTP_CHANNEL_STATE_IDLE	This is macro AVCTP_CHANNEL_STATE_IDLE.
	AVCTP_EVT_CHANNEL_CONNECTED	< This event is generated when a channel between two AVCTP entities has been established.
	AVCTP_EVT_CHANNEL_DISCONNECTED	< This event is generated when a channel between two AVCTP entities has been terminated.
	AVCTP_EVT_COMMAND_CANCELLED	< This event is generated when a command has been canceled.
	AVCTP_EVT_COMMAND_RECEIVED	< This event is generated when a local device received a command.
	AVCTP_EVT_COMMAND_SENT	< This event is generated when a local device finished sending a command.
	AVCTP_EVT_CONNECTION_FAILED	< This event is generated when a local device failed to create a channel between two AVCTP entities.
	AVCTP_EVT_NOTHING	addtogroup avrcp @{ @name Events details The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.
	AVCTP_EVT_RESPONE_RECEIVED	< This event is generated when a local device received a response.
	AVCTP_EVT_RESPONSE_CANCELLED	< This event is generated when a response has been canceled.
	AVCTP_EVT_RESPONSE_SENT	< This event is generated when a local device finished sending a response.
	AVCTP_MANAGER_STATE_IDLE	This is macro AVCTP_MANAGER_STATE_IDLE.
	AVCTP_TRANSPORT_FLAG_RX_MESSAGE_STARTED	This is macro AVCTP_TRANSPORT_FLAG_RX_MESSAGE_STARTED.
	AVCTP_TRANSPORT_FLAG_SENDING	This is macro AVCTP_TRANSPORT_FLAG_SENDING.
	AVCTP_TRANSPORT_STATE_CONNECTED	This is macro AVCTP_TRANSPORT_STATE_CONNECTED.
	AVCTP_TRANSPORT_STATE_CONNECTING	This is macro AVCTP_TRANSPORT_STATE_CONNECTING.
	AVCTP_TRANSPORT_STATE_DISCONNECTING	This is macro AVCTP_TRANSPORT_STATE_DISCONNECTING.
	AVCTP_TRANSPORT_STATE_FREE	This is macro AVCTP_TRANSPORT_STATE_FREE.
	AVCTP_TRANSPORT_STATE_IDLE	This is macro AVCTP_TRANSPORT_STATE_IDLE.

Structures

	Name	Description
	_bt_avctp_channel_t	brief AVCTP channel description ingroup avctp details This structure is used to hold information about an AVCTP channel.
	_bt_avctp_evt_channel_connected_t	brief Parameter to AVCTP_EVT_CHANNEL_CONNECTED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_connected - when a channel between two devices has been established.

	_bt_avctp_evt_channel_disconnected_t	brief Parameter to AVCTP_EVT_CHANNEL_DISCONNECTED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_disconnected - when a channel between two devices has been terminated.
	_bt_avctp_evt_command_cancelled_t	brief Parameter to AVCTP_EVT_COMMAND_CANCELLED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_cancelled - when sending a command message has been canceled.
	_bt_avctp_evt_command_received_t	brief Parameter to AVCTP_EVT_COMMAND_RECEIVED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_received - when a local device received a command message.
	_bt_avctp_evt_command_sent_t	brief Parameter to AVCTP_EVT_COMMAND_SENT event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_sent - when a local device finished sending a command message.
	_bt_avctp_evt_connection_failed_t	brief Parameter to AVCTP_EVT_CONNECTION_FAILED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::connection_failed - when a channel between two devices could not be established.
	_bt_avctp_evt_response_cancelled_t	brief Parameter to AVCTP_EVT_RESPONSE_CANCELLED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_cancelled - when sending a response message has been canceled.
	_bt_avctp_evt_response_received_t	brief Parameter to AVCTP_EVT_RESPONSE_RECEIVED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_received - when a local device received a response message.
	_bt_avctp_evt_response_sent_t	brief Parameter to AVCTP_EVT_RESPONSE_SENT event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_sent - when a local device finished sending a response message.
	_bt_avctp_message_t	brief AVCTP message description ingroup avctp details This structure is used to hold information about an AVCTP message.
	_bt_avctp_mgr_t	brief AVCTP manager. ingroup avctp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avctp_get_mgr().
	_bt_avctp_transport_t	brief AVCTP transport description ingroup avctp details This structure is used to hold information about an AVCTP transport.
	bt_avctp_evt_channel_connected_t	brief Parameter to AVCTP_EVT_CHANNEL_CONNECTED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_connected - when a channel between two devices has been established.
	bt_avctp_evt_channel_disconnected_t	brief Parameter to AVCTP_EVT_CHANNEL_DISCONNECTED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_disconnected - when a channel between two devices has been terminated.
	bt_avctp_evt_command_cancelled_t	brief Parameter to AVCTP_EVT_COMMAND_CANCELLED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_cancelled - when sending a command message has been canceled.
	bt_avctp_evt_command_received_t	brief Parameter to AVCTP_EVT_COMMAND_RECEIVED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_received - when a local device received a command message.
	bt_avctp_evt_command_sent_t	brief Parameter to AVCTP_EVT_COMMAND_SENT event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_sent - when a local device finished sending a command message.
	bt_avctp_evt_connection_failed_t	brief Parameter to AVCTP_EVT_CONNECTION_FAILED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::connection_failed - when a channel between two devices could not be established.

	bt_avctp_evt_response_cancelled_t	brief Parameter to AVCTP_EVT_RESPONSE_CANCELLED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_cancelled - when sending a response message has been canceled.
	bt_avctp_evt_response_received_t	brief Parameter to AVCTP_EVT_RESPONSE_RECEIVED event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_received - when a local device received a response message.
	bt_avctp_evt_response_sent_t	brief Parameter to AVCTP_EVT_RESPONSE_SENT event ingroup avctp details A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_sent - when a local device finished sending a response message.

Types

	Name	Description
	bt_avctp_channel_t	This is type bt_avctp_channel_t.
	bt_avctp_message_t	This is type bt_avctp_message_t.
	bt_avctp_mgr_callback_fp	brief AVCTP application callback. ingroup avctp details In order to be notified of various events a consumer of the AVCTP layer has to register a callback function (done with bt_avctp_set_callback()). The stack will call that function whenever a new event has been generated. param mgr AVCTP manager. param evt AVCTP event. The event can be one of the following values: @arg AVCTP_EVT_CHANNEL_CONNECTED Channel connected. @arg AVCTP_EVT_CHANNEL_DISCONNECTED Channel disconnected. @arg AVCTP_EVT_CONNECTION_FAILED Channel connection failed (generated only if connection has been initiated by the local device). @arg AVCTP_EVT_COMMAND_RECEIVED Command received. @arg AVCTP_EVT_RESPONE_RECEIVED Response received. @arg AVCTP_EVT_COMMAND_SENT Command sent. @arg AVCTP_EVT_RESPONSE_SENT Response... more
	bt_avctp_mgr_t	This is type bt_avctp_mgr_t.
	bt_avctp_transport_t	This is type bt_avctp_transport_t.

Unions

	Name	Description
	_bt_avctp_event_u	brief Parameter to an application callback. ingroup avctp details This union is used to pass event specific data to the AVCTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	bt_avctp_event_t	brief Parameter to an application callback. ingroup avctp details This union is used to pass event specific data to the AVCTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avctp_config.h

Macros

	Name	Description
	__AVCTP_CONFIG_H	This is macro __AVCTP_CONFIG_H.

	AVCTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	• defgroup avctp_config Configuration
		• ingroup avctp
		*
		• This module describes parameters used to configure AVCTP layer.
		*
		• dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.
		* code #include "cdbt/bt/bt_std.h"
		// HCI, L2CAP and SDP must always be present
		// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...
		// L2CAP configuration parameters #define L2CAP_MAX_CMD_BUFFERS ... #define L2CAP_MAX_FRAME_BUFFERS ... #define L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...
		// SDP... more

	AVCTP_ALLOCATE_BUFFERS_VARS	brief Maximum number of message buffers ingroup avctp_config details This parameter defines the maximum number of buffer available for sending message.
	AVCTP_MAX_CHANNELS	brief Maximum number of AVCTP channels ingroup avctp_config details This parameter defines the maximum number of channels a local device can have with remote devices.
	AVCTP_MAX_TRANSPORT_CHANNELS	brief Maximum number of AVCTP transports ingroup avctp_config details This parameter defines the maximum number of transports a local device can have with remote devices. This value should not exceed AVCTP_MAX_CHANNELS.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avctp_packet.h

Functions

	Name	Description
	_bt_avctp_packet_assembler	This is function _bt_avctp_packet_assembler.

Macros

	Name	Description
	__AVCTP_PACKET_H	This is macro __AVCTP_PACKET_H.

Structures

	Name	Description
	_bt_avctp_packet_t	This is type bt_avctp_packet_t.
	bt_avctp_packet_t	This is type bt_avctp_packet_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avctp_private.h

Functions

	Name	Description
	_bt_avctp_allocate_channel	This is function _bt_avctp_allocate_channel.
	_bt_avctp_allocate_message	This is function _bt_avctp_allocate_message.

	_bt_avctp_allocate_transport	This is function _bt_avctp_allocate_transport.
	_bt_avctp_find_channel	This is function _bt_avctp_find_channel.
	_bt_avctp_find_transport	This is function _bt_avctp_find_transport.
	_bt_avctp_free_channel	This is function _bt_avctp_free_channel.
	_bt_avctp_free_message	This is function _bt_avctp_free_message.
	_bt_avctp_free_transport	This is function _bt_avctp_free_transport.
	_bt_avctp_init_message_buffers	This is function _bt_avctp_init_message_buffers.
	_bt_avctp_init_signal	This is function _bt_avctp_init_signal.
	_bt_avctp_l2cap_read_data_callback	This is function _bt_avctp_l2cap_read_data_callback.
	_bt_avctp_send_ipid	This is function _bt_avctp_send_ipid.
	_bt_avctp_set_signal	This is function _bt_avctp_set_signal.
	_bt_avrcp_init_cmd_buffers	This is function _bt_avrcp_init_cmd_buffers.

Macros

	Name	Description
	__AVCTP_PRIVATE_H	This is macro __AVCTP_PRIVATE_H.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avdtp.h

Functions

	Name	Description
	bt_avdtp_abort_stream	brief Suspend a stream. ingroup avdtp details This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE. As a result of this operation the AVDTP_EVT_ABORT_STREAM_COMPLETED event will be generated. This operation cannot be rejected. The p evt_param.abort_stream_requested.err_code is always == AVDTP_ERROR_SUCCESS. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be generated.
	bt_avdtp_add_media_rx_buffer	brief Add a media packet buffer to a receive queue ingroup avdtp details The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more
	bt_avdtp_add_media_tx_buffer	brief Add a media packet buffer to a send queue ingroup avdtp details When the consumer of AVDTP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. When the packet has been successfully sent a AVDTP_EVT_MEDIA_PACKET_SENT is generated. Otherwise a AVDTP_EVT_MEDIA_PACKET_SEND_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as AVDTP has removed it from the queue and gave up... more
	bt_avdtp_cancel_listen	brief Cancel listening for incoming connections. ingroup avdtp details This function removes a SEP from a list of SEPS which a stream can use for incoming requests. param mgr AVDTP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if the function succeeds. li c FALSE otherwise.

	bt_avdtp_clear_media_tx_queue	brief Clear send queue ingroup avdtp details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. The consumer can remove all packets from the queue before they have been sent to a remote device by calling ::bt_avdtp_clear_media_tx_queue. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise. The function fails... more
	bt_avdtp_close_stream	brief Close a stream. ingroup avdtp details This function tries to close a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN or AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the AVDTP_EVT_CLOSE_STREAM_COMPLETED event will be generated. If the stream has been closed the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the p evt_param.bt_avdtp_evt_close_stream_completed_t.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds,... more
	bt_avdtp_create_stream	brief Create a stream. ingroup avdtp details This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort. param mgr AVDTP manager. return li c Stream handle if the function succeeds. li c 0 otherwise.
	bt_avdtp_destroy_stream	brief Destroy a stream. ingroup avdtp details This function frees memory used by the stream. The stream has to exist and be in the "idle" state for this function to succeed. I.e. the stream has to be closed or aborted before this function can be called. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avdtp_disconnect	brief Disconnect from a remote device. ingroup avdtp details This function closes a control and transport channels on all streams associated with the remote device specified by the p remote_addr. As a result of this operation the following events will be generated: @arg AVDTP_EVT_MEDIA_PACKET_RECEIVED: if a stream's receive queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg AVDTP_EVT_MEDIA_PACKET_SENT: if a stream's send queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0 @arg AVDTP_EVT_STREAM_CLOSED: this event is generate if a stream is in... more
	bt_avdtp_discover	brief Discover SEPs on a remote device. ingroup avdtp details This function asks the remote device to send a list of all available SEPs. As a result of this operation the following events will be generated: @arg AVDTP_EVT_SEP_INFO_RECEIVED: this event is generated for every SEP received from the remote device. the p evt_param.sep_info_received contains SEP information. @arg AVDTP_EVT_DISCOVER_COMPLETED: this event is generated after last AVDTP_EVT_SEP_INFO_RECEIVED if the remote accepted the request and the p evt_param.discover_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.discover_completed.err_code == the error code sent by the remote. param mgr AVDTP... more
	bt_avdtp_find_codec	brief Find a codec ingroup avdtp details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store... more

	bt_avdtp_get_all_capabilities	brief Get remote SEP capabilities. ingroup avdtp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after AVDTP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr AVDTP manager.... more
	bt_avdtp_get_capabilities	brief Get remote SEP capabilities. ingroup avdtp details This function asks the remote device to send capabilities of a SEP specified by the p seid_acp. As a result of this operation the following events will be generated: @arg AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the p evt_param.sep_capabilities_received contains SEP capabilities. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after AVDTP_EVT_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the p evt_param.get_sep_capabilities_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_sep_capabilities_completed.err_code == the error code sent by the remote. param mgr AVDTP manager.... more
	bt_avdtp_get_configuration	brief Get stream configuration. ingroup avdtp details This function requests stream configuration from a remote device. As a result of this operation the following events will be generated: @arg AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: this event is generated if the remote accepted the request. the p ebt_para.sep_capabilities_received.caps will contain current stream configuration. @arg AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: If the remote accepted the request the p evt_param.get_stream_configuration_completed.err_code == AVDTP_ERROR_SUCCESS. if the remote rejected the request the p evt_param.get_stream_configuration_completed.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the actual... more
	bt_avdtp_get_hci_connection	brief Get HCI connection for a stream ingroup avdtp details This function returns a pointer to a structure that describes an HCI connection a stream is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call bt_hci_disconnect. param mgr AVDTP manager. param strm_handle Stream handle. return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a stream specified by the p strm_handle... more
	bt_avdtp_get_l2cap_channel	This is function bt_avdtp_get_l2cap_channel.
	bt_avdtp_get_mgr	brief Return a pointer to an instance of the AVDTP manager. ingroup avdtp details This function returns a pointer to an instance of the AVDTP manager. There is only one instance of the manager allocated by the stack. The pointer is passed as the first parameter to all AVDTP functions.
	bt_avdtp_get_sep	brief Get a SEP info by its ID. ingroup avdtp details This function returns a pointer to bt_avdtp_sep_t structure that describes a SEP previously registered with bt_avdtp_register_sep. param mgr AVDTP manager. param sep_id The ID of a SEP. return li c Pointer to bt_avdtp_sep_t if the SEP is in the list of registered SEPs. li c NULL otherwise.
	bt_avdtp_get_stream_codec_config	brief Get the configuration of the codec currently used with the stream. ingroup avdtp details This function returns a pointer to a structure that contains configuration of the codec currently used with the stream. The structure returned depends on the codec. The dotstack defines structures only for SBC, MPEG-1,2 and MPEG-2,4 AAC codecs: @arg SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h) @arg MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h) @arg MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h) param mgr AVDTP manager. param strm_handle Stream handle. return li The codec's configuration if strm_handle specifies a valid stream and the stream is in one of the following... more

	bt_avdtp_get_stream_codec_type	brief Get the type of the codec currently used with the stream. ingroup avdtp details This function returns the type of the codec currently used with the stream. param mgr AVDTP manager. param strm_handle Stream handle. return @arg The type of the codec if strm_handle specifies a valid stream and the stream is in one of the following states: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING @arg The result will be one of the following values: AVDTP_CODEC_TYPE_SBC: SBC AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP... more
	bt_avdtp_get_stream_config	brief Get stream's configuration. ingroup avdtp details This function returns a pointer to a structure holding the current configuration of stream. param mgr AVDTP manager. param strm_handle Stream handle. return li The stream's configuration if strm_handle specifies a valid stream and the stream is in one of the following state: AVDTP_STREAM_STATE_CONFIGURED AVDTP_STREAM_STATE_OPEN AVDTP_STREAM_STATE_STREAMING li NULL otherwise.
	bt_avdtp_get_stream_direction	This is function bt_avdtp_get_stream_direction.
	bt_avdtp_get_stream_local_sep_id	brief Get stream's local SEP ID. ingroup avdtp details This function returns the ID of the local SEP associated with the stream. param mgr AVDTP manager. param strm_handle Stream handle. return li The ID of the local SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_avdtp_get_stream_remote_address	brief Get stream's remote BT address. ingroup avdtp details This function returns the address of the remote device associated with the stream. param mgr AVDTP manager. param strm_handle Stream handle. return li The address of the remote device if strm_handle specifies a valid stream. li NULL otherwise.
	bt_avdtp_get_stream_remote_sep_id	brief Get stream's remote SEP ID. ingroup avdtp details This function returns the ID of the remote SEP associated with the stream. param mgr AVDTP manager. param strm_handle Stream handle. return li The ID of the remote SEP if strm_handle specifies a valid stream. li 0 otherwise.
	bt_avdtp_get_stream_state	brief Get local stream state. ingroup avdtp details This function returns local state of a stream specified by the p strm_handle. No request is sent to the remote party. param mgr AVDTP manager. param strm_handle Stream handle. return The state of the stream. The result will be one of the following values: @arg AVDTP_STREAM_STATE_IDLE: The stream is idle. This can mean two things. The stream specified by p strm_handle does not exist or the stream is closed. @arg AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS: The stream is opening transport channels. @arg AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS: The stream is closing transport channels. @arg AVDTP_STREAM_STATE_CONFIGURED: The... more
	bt_avdtp_init	brief Initialize the AVDTP layer. ingroup avdtp details This function initializes the AVDTP layer of the stack. It must be called prior to any other AVDTP function can be called.
	bt_avdtp_listen	brief Listen for incoming connections. ingroup avdtp details This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream. param mgr AVDTP manager. param strm_handle Stream handle. param sep_id Local SEP ID. return li c TRUE if... more

	bt_avdtp_open_stream	brief Open a stream. ingroup avdtp details This function tries to open a stream by sending a request to the remote party. The stream has to be already configured with a bt_avdtp_set_configuration call. As a result of this operation the AVDTP_EVT_OPEN_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.open_stream_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to open the stream, the p evt_param.open_stream_completed.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function... more
	bt_avdtp_reconfigure_stream	brief Reconfigure stream. ingroup avdtp details This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event will be generated. If reconfiguration was a success the p evt_param.stream_reconfigure_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.stream_reconfigure_completed.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. param caps New stream configuration. return li c TRUE if the function succeeds, i.e. the actual request has been sent to the remote party. li c FALSE otherwise. No events will be... more
	bt_avdtp_register_callback	brief Register a AVDTP application callback. ingroup avdtp details In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values: @arg AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote... more
	bt_avdtp_register_codec	brief Register a codec ingroup avdtp details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store... more
	bt_avdtp_register_sep	brief Register a SEP with the local AVDTP manager. ingroup avdtp details This function is used to make a list of SEPs supported by the local ADVTP entity. param mgr AVDTP manager. param type The type of a SEP. The type can be one of the following values: @arg AVDTP_SEP_TYPE_SOURCE: The SEP is a source. @arg AVDTP_SEP_TYPE_SINK: The SEP is a sink. param caps The capabilities of a SEP. return li c ID of a SEP if the function succeeds. li c FALSE otherwise.
	bt_avdtp_remove_media_rx_buffer	brief Remove a media packet buffer from a receive queue ingroup avdtp details The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue... more
	bt_avdtp_remove_media_tx_buffer	brief Remove a media packet buffer from a send queue ingroup avdtp details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling bt_avdtp_remove_media_tx_buffer. param mgr AVDTP manager. param strm_handle Stream handle. param buffer Pointer to a structure... more

	bt_avdtp_report_delay	brief Report delay value of a Sink to a Source. ingroup avdtp details This function sends the delay value of a Sink to a Source. This enables synchronous playback of audio and video. Delay reports are always sent from the Sink to the Source. If the Sink's delay report has been accepted by the Source the p evt_param.delay_report_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.delay_report_completed.err_code == the error code sent by the Source. param mgr AVDTP manager. param strm_handle Stream handle. param delay The delay value in 1/10 milliseconds. return li c TRUE if the function succeeds, i.e. the actual... more
	bt_avdtp_security_control	brief Exchange content protection control data. ingroup avdtp details This function tries to establish content protection by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE, AVDTP_STREAM_STATE_CLOSING, AVDTP_STREAM_STATE_ABORTING. As a result of this operation the AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event will be generated. If the stream's content protection data has been accepted by the remote party the p evt_param.security_control_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.security_control_completed.err_code == the error code sent by the remote. note The dotstack does not support content protection. Although the request can be sent it will not affect... more
	bt_avdtp_set_configuration	brief Set stream configuration. ingroup avdtp details This function tries to configure a stream before opening it. As a result of this operation the AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event will be generated. If configuration was a success the p evt_param.set_stream_configuration_completed.err_code == AVDTP_ERROR_SUCCESS. Otherwise the p evt_param.set_stream_configuration_completed.err_code == the error code sent by the remote and p evt_param.set_stream_configuration_completed.svc_category == the value of the first Service Category to fail. param mgr AVDTP manager. param strm_handle Stream handle. param remote_addr The address of a remote device. param seid_int Local SEP ID. param seid_acp Remote SEP ID. param caps Stream configuration. return li c TRUE if... more
	bt_avdtp_set_media_tx_queue_limit	brief Set limit on the send queue ingroup avdtp details When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. By default the send queue can contain unlimited number of packets. The consumer can set a limit on how many packets are held in the queue. In this case when new packet is added to the queue and the length of... more
	bt_avdtp_start	brief Start the AVDTP layer. ingroup avdtp details This function makes the AVDTP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required. param mgr AVDTP manager. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avdtp_start_stream	brief Start a stream. ingroup avdtp details This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the AVDTP_EVT_START_STREAM_COMPLETED event will be generated. If the stream has been open the p evt_param.start_stream_requested.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the p evt_param.start_stream_requested.err_code == the error code sent... more
	bt_avdtp_suspend_stream	brief Suspend a stream. ingroup avdtp details This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the AVDTP_EVT_SUSPEND_STREAM_COMPLETED event will be generated. If the stream has been suspended the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the p evt_param.bt_avdtp_evt_suspend_stream_requested_t.err_code == the error code sent by the remote. param mgr AVDTP manager. param strm_handle Stream handle. return li c TRUE if the function succeeds, i.e. the... more

	bt_avdtp_unregister_codec	brief Unregister a codec ingroup avdtp details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store... more

Macros

	Name	Description
	__AVDTP_H	This is macro __AVDTP_H.
	AVDTP_CODEC_OPCODE_PARSE_CONFIG	< Parse codec configuration.
	AVDTP_CODEC_OPCODE_PARSE_PACKET	This is macro AVDTP_CODEC_OPCODE_PARSE_PACKET.
	AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG	< Serialize codec configuration.
	AVDTP_CODEC_TYPE_ATRAC	< ATRAC (proprietary codec owned by Sony Corporation).
	AVDTP_CODEC_TYPE_MPEG1_2_AUDIO	< MPEG-1,2 (optional).
	AVDTP_CODEC_TYPE_MPEG2_4_AAC	< MPEG-2,4 AAC (optional, used in Apple's products).
	AVDTP_CODEC_TYPE_NON_A2DP	< Vendor specific.
	AVDTP_CODEC_TYPE_SBC	< SBC (mandatory to support in A2DP profile).
	AVDTP_CONTENT_PROTECTION_METHOD_SCMS_T	This is macro AVDTP_CONTENT_PROTECTION_METHOD_SCMS_T.
	AVDTP_ERROR_BAD_ACP_SEID	< The requested command indicates an invalid ACP SEP ID (not addressable)
	AVDTP_ERROR_BAD_CP_FORMAT	< The format of Content Protection Service Capability is not correct.
	AVDTP_ERROR_BAD_HEADER_FORMAT	< The request packet header format is invalid.
	AVDTP_ERROR_BAD_LENGTH	< The request packet length is not match the assumed length.
	AVDTP_ERROR_BAD_MEDIA_TRANSPORT_FORMAT	< The format of Media Transport Capability is not correct.
	AVDTP_ERROR_BAD_MULTIPLEXING_FORMAT	< The format of Multiplexing Service Capability is not correct.
	AVDTP_ERROR_BAD_PAYLOAD_FORMAT	< The requested command has an incorrect payload format.
	AVDTP_ERROR_BAD_RECOVERY_FORMAT	< The format of Recovery Service Capability is not correct.
	AVDTP_ERROR_BAD_RECOVERY_TYPE	< The requested Recovery Type is not defined in AVDTP.
	AVDTP_ERROR_BAD_ROHC_FORMAT	< The format of Header Compression Service Capability is not correct.
	AVDTP_ERROR_BAD_SERV_CATEGORY	< The value of Service Category in the request packet is not defined in AVDTP.
	AVDTP_ERROR_BAD_STATE	< The stream is in state that does not permit executing commands.
	AVDTP_ERROR_FAILED_TO_CONNECT_CONTROL	< An attempt to establish a control channel has failed.
	AVDTP_ERROR_FAILED_TO_CONNECT_TRANSPORT	< An attempt to establish a transport channel has failed.
	AVDTP_ERROR_INVALID_CAPABILITIES	< The reconfigure command is an attempt to reconfigure a transport service capabilities of the SEP. Reconfigure is only permitted for application service capabilities.
	AVDTP_ERROR_NOT_SUPPORTED_COMMAND	< The requested command is not supported by the device.
	AVDTP_ERROR_SEP_IN_USE	< The SEP is in use.
	AVDTP_ERROR_SEP_NOT_IN_USE	< The SEP is not in use.
	AVDTP_ERROR_SUCCESS	< The operation completed with no errors.
	AVDTP_ERROR_UNSUPPORTED_CONFIGURAION	< Configuration not supported.
	AVDTP_EVT_ABORT_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.
	AVDTP_EVT_ABORT_STREAM_REQUESTED	< This event is generated when a local device received "abort stream" request.
	AVDTP_EVT_CLOSE_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "close stream" request.
	AVDTP_EVT_CLOSE_STREAM_REQUESTED	< This event is generated when a local device received "close stream" request.
	AVDTP_EVT_CTRL_CHANNEL_CONNECTED	< This event is generated when a control channel between two AVDTP entities has been established.

	AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED	< This event is generated when a control channel between two AVDTP entities has been terminated.
	AVDTP_EVT_CTRL_CONNECTION_FAILED	< This event is generated when a local device failed to create a control channel between two AVDTP entities.
	AVDTP_EVT_DELAYREPORT_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "delay report" request.
	AVDTP_EVT_DISCOVER_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "discover" request.
	AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.
	AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.
	AVDTP_EVT_LAST	This is macro AVDTP_EVT_LAST.
	AVDTP_EVT_MEDIA_PACKET_RECEIVED	< This event is generated when a local device received a media packet.
	AVDTP_EVT_MEDIA_PACKET_SEND_FAILED	< This event is generated when a local device failed to send a media packet.
	AVDTP_EVT_MEDIA_PACKET_SENT	< This event is generated when a local device sent a media packet.
	AVDTP_EVT_NULL	This is macro AVDTP_EVT_NULL.
	AVDTP_EVT_OPEN_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "open stream" request.
	AVDTP_EVT_OPEN_STREAM_REQUESTED	< This event is generated when a local device received "open stream" request.
	AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED	< This event is generated when a local device received "change stream configuration" request.
	AVDTP_EVT_SEP_CAPABILITIES_RECEIVED	< This event is generated when a local device received a positive response to a "get SEP capabilities" request.
	AVDTP_EVT_SEP_INFO_RECEIVED	< This event is generated for each SEP contained in a positive response to a "discover" request.
	AVDTP_EVT_SET_STREAM_CONFIGURATION	This is macro AVDTP_EVT_SET_STREAM_CONFIGURATION.
	AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.
	AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED	< This event is generated when a local device received "set stream configuration" request.
	AVDTP_EVT_START_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "start stream" request.
	AVDTP_EVT_START_STREAM_REQUESTED	< This event is generated when a local device received "start stream" request.
	AVDTP_EVT_STREAM_ABORTED	< This event is generated when a local device has successfully aborted a stream. < This event follows the AVDTP_EVT_ABORT_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream abortion was initiated by the local device.
	AVDTP_EVT_STREAM_CLOSED	< This event is generated when a local device has successfully closed a stream. < This event follows the AVDTP_EVT_CLOSE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream closing was initiated by the local device.
	AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED	< This event is generated when a local device received a positive response to a "get stream configuration" request.
	AVDTP_EVT_STREAM_CONFIGURED	< This event is generated when a local device has successfully configured a stream. < This event follows the AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the upper layer has accepted it. < This event is not generated if stream configuration was initiated by the local device.

	AVDTP_EVT_STREAM_OPENED	< This event is generated when a local device has successfully opened a stream. < This event follows the AVDTP_EVT_OPEN_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream opening was initiated by the local device.
	AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.
	AVDTP_EVT_STREAM_RECONFIGURED	< This event is generated when a local device has successfully reconfigured a stream. < This event follows the AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream reconfiguration was initiated by the local device.
	AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.
	AVDTP_EVT_STREAM_STARTED	< This event is generated when a local device has successfully started a stream. < This event follows the AVDTP_EVT_START_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream starting was initiated by the local device.
	AVDTP_EVT_STREAM_SUSPENDED	< This event is generated when a local device has successfully suspended a stream. < This event follows the AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the upper layer has accepted it. < This event is not generated if stream suspension was initiated by the local device.
	AVDTP_EVT_SUSPEND_STREAM_COMPLETED	< This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.
	AVDTP_EVT_SUSPEND_STREAM_REQUESTED	< This event is generated when a local device received "suspend stream" request.
	AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET	This is macro AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET.
	AVDTP_MANAGER_STATE_CONNECTING	This is macro AVDTP_MANAGER_STATE_CONNECTING.
	AVDTP_MANAGER_STATE_IDLE	This is macro AVDTP_MANAGER_STATE_IDLE.
	AVDTP_MAX_STREAM_TRANSPORT_SESSION	This is macro AVDTP_MAX_STREAM_TRANSPORT_SESSION.
	AVDTP_MEDIA_TYPE_AUDIO	< Audio.
	AVDTP_MEDIA_TYPE_MULTIMEDIA	< Both Audio & Video.
	AVDTP_MEDIA_TYPE_VIDEO	< Video.
	AVDTP_SCMS_T_CP_BIT	This is macro AVDTP_SCMS_T_CP_BIT.
	AVDTP_SCMS_T_L_BIT	This is macro AVDTP_SCMS_T_L_BIT.
	AVDTP_SEP_CAPABILITY_FLAG_CONTENT_PROTECTION	< Content Prortection. A SEP is capable of transferring content protection packets.
	AVDTP_SEP_CAPABILITY_FLAG_DELAY_REPORTING	< Delat reporitng.
	AVDTP_SEP_CAPABILITY_FLAG_HEADER_COMPRESSION	< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.
	AVDTP_SEP_CAPABILITY_FLAG_MEDIA_CODEC	< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.
	AVDTP_SEP_CAPABILITY_FLAG_MEDIA_TRANSPORT	< Media. A SEP is capable of transferring media (audio, video or both) packets.
	AVDTP_SEP_CAPABILITY_FLAG_MULTIPLEXING	< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.
	AVDTP_SEP_CAPABILITY_FLAG_RECOVERY	< Recovery. A SEP is capable of transferring recovery packets.
	AVDTP_SEP_CAPABILITY_FLAG_REPORTING	< Reporting. A SEP is capable of transferring reporting packets.
	AVDTP_SEP_SERVICE_CAPABILITY_CONTENT_PROTECTION	< Content Prortection. A SEP is capable of transferring content protection packets.
	AVDTP_SEP_SERVICE_CAPABILITY_DELAY_REPORTING	< Delay Reporting.
	AVDTP_SEP_SERVICE_CAPABILITY_HEADER_COMPRESSION	< Header Compression. A SEP can use header compression for transferring Media or Recovery packets.

	AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_CODEC	< Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.
	AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_TRANSPORT	< Media. A SEP is capable of transferring media (audio, video or both) packets.
	AVDTP_SEP_SERVICE_CAPABILITY_MULTIPLEXING	< Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.
	AVDTP_SEP_SERVICE_CAPABILITY_RECOVERY	< Recovery. A SEP is capable of transferring recovery packets.
	AVDTP_SEP_SERVICE_CAPABILITY_REPORTING	< Reporting. A SEP is capable of transferring reporting packets.
	AVDTP_SEP_STATE_FREE	This is macro AVDTP_SEP_STATE_FREE.
	AVDTP_SEP_STATE_IDLE	This is macro AVDTP_SEP_STATE_IDLE.
	AVDTP_SEP_TYPE_SINK	< Sink (usually a device like a headphones or BMW).
	AVDTP_SEP_TYPE_SOURCE	< Source (usually a device like a phone, desktop or laptop).
	AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS	< The stream is closing transport channels.
	AVDTP_STREAM_FLAG_LISTENING	This is macro AVDTP_STREAM_FLAG_LISTENING.
	AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS	< The stream is opening transport channels.
	AVDTP_STREAM_STATE_ABORTING	< The stream is aborting. This means that all transport channels associated with the stream are being closed. < After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.
	AVDTP_STREAM_STATE_CLOSING	< The stream is closing. This means that all transport channels associated with the stream are being closed. < After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.
	AVDTP_STREAM_STATE_CONFIGURED	< The stream has been configured.
	AVDTP_STREAM_STATE_IDLE	< The stream is idle. This can mean two things. The stream specified by strm_handle does not exist < or the stream is closed.
	AVDTP_STREAM_STATE_OPEN	< The stream has been opened.
	AVDTP_STREAM_STATE_STREAMING	< The stream has been started. Depending on the local SEP type (source or sink) it means < that the stream can send or receive media packets.
	AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED	This is macro AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED.
	AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED	This is macro AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED.
	AVDTP_TRANSPORT_SESSION_TYPE_MEDIA	< Media (audio or video).
	AVDTP_TRANSPORT_SESSION_TYPE_RECOVERY	< Recovery (currently not supported).
	AVDTP_TRANSPORT_SESSION_TYPE_REPORTING	< Reporting (currently not supported).
	bt_avdtp_connect	brief Connect to a remote device. ingroup avdtp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP_EVT_CTRL_CHANNEL_CONNECTED or AVDTP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr AVDTP manager. param remote_addr The address of a remote device. return li c TRUE if connection establishment has been started. li c FALSE... more

	bt_avdtp_connect_ex	brief Connect to a remote device. ingroup avdtp details This function opens a control channel connection to a remote device specified by the p remote_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP_EVT_CTRL_CHANNEL_CONNECTED or AVDTP_EVT_CTRL_CHANNEL_CONNECTION_FAILED. param mgr AVDTP manager. param remote_addr The address of a remote device. param acl_config ACL link configuration. This can be a combination of the following... more

Structures

	Name	Description
	_bt_avdtp_codec_op_decode_t	There is currently no use for this structure.
	_bt_avdtp_codec_op_encode_t	There is currently no use for this structure.
	_bt_avdtp_codec_op_parse_config_t	brief Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::parse - when ADVDTP needs to parse codec's capabilities/configuration received from the remote device.
	_bt_avdtp_codec_op_parse_packet_t	There is currently no use for this structure.
	_bt_avdtp_codec_op_serialize_config_t	brief Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::serialize - when ADVDTP needs to serialize codec's capabilities/configuration for sending to the remote device.
	_bt_avdtp_codec_t	brief Codec handler description. ingroup avdtp details This structure is used to register a codec handler for parsing/serializing codec capabilities and configuration. See description of the ::bt_avdtp_register_codec for more details.
	_bt_avdtp_evt_abort_stream_requested_t	brief Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::abort_stream_requested - when AVDTP received a "abort stream" request.
	_bt_avdtp_evt_close_stream_completed_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_completed - when AVDTP received a response to a "close stream" request.
	_bt_avdtp_evt_close_stream_requested_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_requested - when AVDTP received a "close stream" request.
	_bt_avdtp_evt_ctrl_channel_connected_s	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_connected - when a control channel between two devices has been established.
	_bt_avdtp_evt_ctrl_channel_disconnected_s	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_disconnected - when a control channel between two devices has been terminated.

	_bt_avdtp_evt_ctrl_connection_failed_s	brief Parameter to AVDTP_EVT_CTRL_CONNECTION_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_connection_failed - when a control channel between two devices has been established.
	_bt_avdtp_evt_delay_report_completed_t	brief Parameter to AVDTP_EVT_DELAYREPORT_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::delay_report_completed - when AVDTP received a response to a "delay report" request.
	_bt_avdtp_evt_discover_completed_t	brief Parameter to AVDTP_EVT_DISCOVER_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::discover_completed - when AVDTP completed discovering SEPs available on a remote device.
	_bt_avdtp_evt_get_sep_capabilities_completed_t	brief Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_sep_capabilities_completed - when AVDTP received a response to a "get SEP capabilities" request. AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_SEP_CAPABILITIES_RECEIVED - is generate with a pointer to a structure that holds actual SEP's capabilities.
	_bt_avdtp_evt_get_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_stream_configuration_completed - when AVDTP received a response to a "get stream configuration" request. AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED - is generate with a pointer to a structure that hold actual stream's configuration.
	_bt_avdtp_evt_media_packet_received_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_received - when ADVDTP received a media packet from the remote device.
	_bt_avdtp_evt_media_packet_send_failed_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_send_failed - when ADVDTP failed to send a media packet to the remote device.
	_bt_avdtp_evt_media_packet_sent_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_sent - when ADVDTP sent a media packet to the remote device.
	_bt_avdtp_evt_open_stream_completed_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_completed - when AVDTP received a response to a "open stream" request.

	_bt_avdtp_evt_open_stream_requested_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_requested - when AVDTP received a "open stream" request.
	_bt_avdtp_evt_reconfigure_stream_requested_t	brief Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::reconfigure_stream_requested - when AVDTP received a "change stream configuration" request.
	_bt_avdtp_evt_sep_capabilities_received_t	brief Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED events ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_capabilities_received - when AVDTP received a positive response to a "get SEP capabilities" or "get stream configuration" request.
	_bt_avdtp_evt_sep_info_received_t	brief Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_info_received - when AVDTP received positive result to a "discover" request. AVDTP_EVT_SEP_INFO_RECEIVED is generated for every SEP received from the remote device.
	_bt_avdtp_evt_set_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_completed - when AVDTP received a response to a "set stream configuration" request.
	_bt_avdtp_evt_set_stream_configuration_requested_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_requested - when AVDTP received a "set stream configuration" request.
	_bt_avdtp_evt_set_stream_configuration_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration - when AVDTP received a "set stream configuration" request.
	_bt_avdtp_evt_start_stream_completed_t	brief Parameter to AVDTP_EVT_START_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_completed - when AVDTP received a response to a "start stream" request.
	_bt_avdtp_evt_start_stream_requested_t	brief Parameter to AVDTP_EVT_START_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_requested - when AVDTP received a "start stream" request.
	_bt_avdtp_evt_stream_aborted_t	brief Parameter to AVDTP_EVT_STREAM_ABORTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully aborted.

	_bt_avdtp_evt_stream_closed_t	brief Parameter to AVDTP_EVT_STREAM_CLOSED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully closed.
	_bt_avdtp_evt_stream_configured_t	brief Parameter to AVDTP_EVT_STREAM_CONFIGURED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_configured - to notify the AVDTP consumer that a stream configuration has been successfully completed.
	_bt_avdtp_evt_stream_opened_t	brief Parameter to AVDTP_EVT_STREAM_OPENED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_opened - to notify the AVDTP consumer that a stream has been successfully opened.
	_bt_avdtp_evt_stream_reconfigure_completed_t	brief Parameter to AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigure_completed - when AVDTP received a response to a "change stream configuration" request.
	_bt_avdtp_evt_stream_reconfigured_t	brief Parameter to AVDTP_EVT_STREAM_RECONFIGURED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigured - to notify the AVDTP consumer that a stream configuration has been successfully changed.
	_bt_avdtp_evt_stream_security_control_completed_t	brief Parameter to AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::security_control_completed - when AVDTP received a response to a "exchange content protection control data" request.
	_bt_avdtp_evt_stream_started_t	brief Parameter to AVDTP_EVT_STREAM_STARTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_started - to notify the AVDTP consumer that a stream has been successfully started.
	_bt_avdtp_evt_stream_suspended_t	brief Parameter to AVDTP_EVT_STREAM_SUSPENDED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_suspended - to notify the AVDTP consumer that a stream has been successfully suspended.
	_bt_avdtp_evt_suspend_stream_completed_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_completed - when AVDTP received a response to a "suspend stream" request.
	_bt_avdtp_evt_suspend_stream_requested_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_requested - when AVDTP received a "suspend stream" request.
	_bt_avdtp_mgr_t	brief AVDTP manager. ingroup avdtp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avdtp_get_mgr().
	_bt_avdtp_sep_capabilities_t	brief SEP capabilities ingroup avdtp details This structure is used to hold SEP capabilities.
	_bt_avdtp_sep_t	brief SEP description ingroup avdtp details This structure is used to hold information about SEPs available on a local device.

	_bt_avdtp_stream_t	brief Stream description ingroup avdtp details This structure is used to hold information about streams available on a local device.
	_bt_avdtp_transport_channel_t	brief Transport channel description ingroup avdtp details This structure is used to hold information about transport channels available on a local device.
	_bt_avdtp_transport_session_t	brief Transport session description ingroup avdtp details This structure is used to hold information about transport sessions available on a local device.
	_bt_cp_header_s	This is record _bt_cp_header_s.
	_bt_media_packet_t	brief Media packet buffer ingroup avdtp details This structure is used to receive and send media packet from/to the remote device. See more information about usage of this structure in descriptions of ::bt_avdtp_add_media_rx_buffer and ::bt_avdtp_add_media_tx_buffer.
	bt_avdtp_codec_op_decode_t	There is currently no use for this structure.
	bt_avdtp_codec_op_encode_t	There is currently no use for this structure.
	bt_avdtp_codec_op_parse_config_t	brief Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::parse - when ADVDTP needs to parse codec's capabilities/configuration received from the remote device.
	bt_avdtp_codec_op_parse_packet_t	There is currently no use for this structure.
	bt_avdtp_codec_op_serialize_config_t	brief Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation. ingroup avdtp details A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::serialize - when ADVDTP needs to serialize codec's capabilities/configuration for sending to the remote device.
	bt_avdtp_codec_t	brief Codec handler description. ingroup avdtp details This structure is used to register a codec handler for parsing/serializing codec capabilities and configuration. See description of the ::bt_avdtp_register_codec for more details.
	bt_avdtp_evt_abort_stream_requested_t	brief Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::abort_stream_requested - when AVDTP received a "abort stream" request.
	bt_avdtp_evt_close_stream_completed_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_completed - when AVDTP received a response to a "close stream" request.
	bt_avdtp_evt_close_stream_requested_t	brief Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_requested - when AVDTP received a "close stream" request.
	bt_avdtp_evt_ctrl_channel_connected_t	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_connected - when a control channel between two devices has been established.
	bt_avdtp_evt_ctrl_channel_disconnected_t	brief Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_channel_disconnected - when a control channel between two devices has been terminated.

	bt_avdtp_evt_ctrl_connection_failed_t	brief Parameter to AVDTP_EVT_CTRL_CONNECTION_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::ctrl_connection_failed - when a control channel between two devices has been established.
	bt_avdtp_evt_delay_report_completed_t	brief Parameter to AVDTP_EVT_DELAYREPORT_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::delay_report_completed - when AVDTP received a response to a "delay report" request.
	bt_avdtp_evt_discover_completed_t	brief Parameter to AVDTP_EVT_DISCOVER_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::discover_completed - when AVDTP completed discovering SEPs available on a remote device.
	bt_avdtp_evt_get_sep_capabilities_completed_t	brief Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_sep_capabilities_completed - when AVDTP received a response to a "get SEP capabilities" request. AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_SEP_CAPABILITIES_RECEIVED - is generate with a pointer to a structure that holds actual SEP's capabilities.
	bt_avdtp_evt_get_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::get_stream_configuration_completed - when AVDTP received a response to a "get stream configuration" request. AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED - is generate with a pointer to a structure that hold actual stream's configuration.
	bt_avdtp_evt_media_packet_received_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_received - when ADVDTP received a media packet from the remote device.
	bt_avdtp_evt_media_packet_send_failed_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_send_failed - when ADVDTP failed to send a media packet to the remote device.
	bt_avdtp_evt_media_packet_sent_t	brief Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_sent - when ADVDTP sent a media packet to the remote device.
	bt_avdtp_evt_open_stream_completed_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_completed - when AVDTP received a response to a "open stream" request.

	bt_avdtp_evt_open_stream_requested_t	brief Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_requested - when AVDTP received a "open stream" request.
	bt_avdtp_evt_reconfigure_stream_requested_t	brief Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::reconfigure_stream_requested - when AVDTP received a "change stream configuration" request.
	bt_avdtp_evt_sep_capabilities_received_t	brief Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED events ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_capabilities_received - when AVDTP received a positive response to a "get SEP capabilities" or "get stream configuration" request.
	bt_avdtp_evt_sep_info_received_t	brief Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_info_received - when AVDTP received positive result to a "discover" request. AVDTP_EVT_SEP_INFO_RECEIVED is generated for every SEP received from the remote device.
	bt_avdtp_evt_set_stream_configuration_completed_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_completed - when AVDTP received a response to a "set stream configuration" request.
	bt_avdtp_evt_set_stream_configuration_requested_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_requested - when AVDTP received a "set stream configuration" request.
	bt_avdtp_evt_set_stream_configuration_t	brief Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration - when AVDTP received a "set stream configuration" request.
	bt_avdtp_evt_start_stream_completed_t	brief Parameter to AVDTP_EVT_START_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_completed - when AVDTP received a response to a "start stream" request.
	bt_avdtp_evt_start_stream_requested_t	brief Parameter to AVDTP_EVT_START_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_requested - when AVDTP received a "start stream" request.
	bt_avdtp_evt_stream_aborted_t	brief Parameter to AVDTP_EVT_STREAM_ABORTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully aborted.

	bt_avdtp_evt_stream_closed_t	brief Parameter to AVDTP_EVT_STREAM_CLOSED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully closed.
	bt_avdtp_evt_stream_configured_t	brief Parameter to AVDTP_EVT_STREAM_CONFIGURED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_configured - to notify the AVDTP consumer that a stream configuration has been successfully completed.
	bt_avdtp_evt_stream_opened_t	brief Parameter to AVDTP_EVT_STREAM_OPENED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_opened - to notify the AVDTP consumer that a stream has been successfully opened.
	bt_avdtp_evt_stream_reconfigure_completed_t	brief Parameter to AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigure_completed - when AVDTP received a response to a "change stream configuration" request.
	bt_avdtp_evt_stream_reconfigured_t	brief Parameter to AVDTP_EVT_STREAM_RECONFIGURED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigured - to notify the AVDTP consumer that a stream configuration has been successfully changed.
	bt_avdtp_evt_stream_security_control_completed_t	brief Parameter to AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::security_control_completed - when AVDTP received a response to a "exchange content protection control data" request.
	bt_avdtp_evt_stream_started_t	brief Parameter to AVDTP_EVT_STREAM_STARTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_started - to notify the AVDTP consumer that a stream has been successfully started.
	bt_avdtp_evt_stream_suspended_t	brief Parameter to AVDTP_EVT_STREAM_SUSPENDED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_suspended - to notify the AVDTP consumer that a stream has been successfully suspended.
	bt_avdtp_evt_suspend_stream_completed_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_COMPLETED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_completed - when AVDTP received a response to a "suspend stream" request.
	bt_avdtp_evt_suspend_stream_requested_t	brief Parameter to AVDTP_EVT_SUSPEND_STREAM_REQUESTED event ingroup avdtp details A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_requested - when AVDTP received a "suspend stream" request.
	bt_avdtp_mgr_t	brief AVDTP manager. ingroup avdtp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avdtp_get_mgr().
	bt_avdtp_sep_capabilities_t	brief SEP capabilities ingroup avdtp details This structure is used to hold SEP capabilities.
	bt_avdtp_stream_t	brief Stream description ingroup avdtp details This structure is used to hold information about streams available on a local device.

	bt_avdtp_transport_channel_t	brief Transport channel description ingroup avdtp details This structure is used to hold information about transport channels available on a local device.
	bt_avdtp_transport_session_t	brief Transport session description ingroup avdtp details This structure is used to hold information about transport sessions available on a local device.

Types

	Name	Description
	bt_avdtp_codec_handler_fp	brief Codec handler. ingroup avdtp details AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it... more
	bt_avdtp_codec_op_param_t	This is type bt_avdtp_codec_op_param_t.
	bt_avdtp_mgr_callback_fp	brief AVDTP application callback. ingroup avdtp details In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call that function whenever a new event has been generated. param mgr AVDTP manager. param evt AVDTP event. The event can be one of the following values: @arg AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. @arg AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. @arg AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). @arg AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. @arg AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED:... more
	bt_avdtp_sep_t	This is type bt_avdtp_sep_t.
	bt_cp_header_t	This is type bt_cp_header_t.
	bt_media_packet_t	This is type bt_media_packet_t.

Unions

	Name	Description
	_bt_avdtp_codec_op_param_u	brief Parameter to a codec handler. ingroup avdtp details This union is used to pass operation specific data to a codec handler. Which member of the union points to a valid structure depends on the operation.
	_bt_avdtp_event_u	brief Parameter to an application callback. ingroup avdtp details This union is used to pass event specific data to the AVDTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	bt_avdtp_event_t	brief Parameter to an application callback. ingroup avdtp details This union is used to pass event specific data to the AVDTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avdtp_config.h

Macros

	Name	Description
	__AVDTP_CONFIG_H	This is macro __AVDTP_CONFIG_H.
	AVDTP_ALLOCATE_BUFFERS_FUNCTION	This is macro AVDTP_ALLOCATE_BUFFERS_FUNCTION.

	AVDTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	• defgroup avdtp_config Configuration
		• ingroup avdtp
		*
		• This module describes parameters used to configure AVDTP layer.
		*
		• dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.
		* code #include "cdbt/bt/bt_std.h"
		// HCI, L2CAP and SDP must always be present
		// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...
		// L2CAP configuration parameters #define L2CAP_MAX_CMD_BUFFERS ... #define L2CAP_MAX_FRAME_BUFFERS ... #define L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...
		// SDP... more

	AVDTP_ALLOCATE_BUFFERS_VARS	This is macro AVDTP_ALLOCATE_BUFFERS_VARS.
	AVDTP_CODEC_CONFIG_BUFFER_LEN	brief Size of the buffer used to store codec specific configuration. ingroup avdtp_config details Each codec uses unique configuration which can take different amount of memory. This parameter defines the size of the buffer for storing codec's configuration. The value of 16 is sufficient for SBC, AAC and MPEG1,2. If vendor specific codec is to be used this value may need to increased.
	AVDTP_MAX_CMD_BUFFERS	brief Maximum number of command buffers. ingroup avdtp_config details This parameter defines the number of buffers reserved for sending commands to a remote device over its control channel. Each channel uses its own buffers so the total number of buffers is AVDTP_MAX_REMOTE_DEVICES * AVDTP_MAX_CMD_BUFFERS. The minimum value is 1. The maximum value is 255. 2 is usually sufficient.
	AVDTP_MAX_CMD_PARAM_LEN	brief Maximum length of control command parameters ingroup avdtp_config details This parameter defines the maximum length of all command parameters. The value should not exceed AVDTP_MAX_TX_BUFFER_LEN - 2 (command header).
	AVDTP_MAX_REMOTE_DEVICES	brief Maximum number of remote devices a local device can be connected to ingroup avdtp_config details This parameter defines the number of remote devices a local device can have simultaneous connections to (i.e. control channels). This value should not exceed AVDTP_MAX_STREAMS. For each remote device AVDTP creates one control channel regardless of the number of streams between the local and the remote devices. Assuming that the local devices wants to have only one channel with each remote device and if AVDTP_MAX_REMOTE_DEVICES > AVDTP_MAX_STREAMS all memory reserved for devices in excess of AVDTP_MAX_STREAMS will be wasted. The minimum value is... more
	AVDTP_MAX_SEP	brief Maximum number of SEPs that can be exposed by a local device ingroup avdtp_config details This parameter defines the number of SEPs an application can expose to remote devices. The minimum value is 1. The maximum value is 255.
	AVDTP_MAX_STREAMS	brief Maximum number of streams that can be exposed by a local device ingroup avdtp_config details This parameter defines the number of streams an application can open between local and remote devices. This value can be different from AVDTP_MAX_SEP but should not exceed it. Since each SEP can only be used once the local device can only have as much streams as there are SEPs. If AVDTP_MAX_STREAMS > AVDTP_MAX_SEP all memory reserved for streams in excess of AVDTP_MAX_SEP will be wasted. The minimum value is 1. The maximum value is 255.

	AVDTP_MAX_TRANSPORT_CHANNELS	brief Maximum number of transport channels. ingroup avdtp_config details Depending on the SEP capabilities (multiplexing, recovery, reporting) each stream may need up to 3 transport channels. E.g., if multiplexing is not supported and recovery and reporting are supported a stream will use 3 transport channels - 1 for media transport session, 1 for recovery transport session and 1 for reporting transport session. If multiplexing is not supported and only reporting is supported the stream will use 2 transport channels - 1 for media transport session and 1 for reporting transport session. If multiplexing is supported the stream will need only... more
	AVDTP_MAX_TX_BUFFER_LEN	brief Size of the transmit buffer. ingroup avdtp_config details This parameter defines the size of the buffer used to send AVDTP control commands to L2CAP layer. Each control channel has it own buffer so the total amount of memory allocated for these buffers is AVDTP_MAX_TX_BUFFER_LEN) * AVDTP_MAX_REMOTE_DEVICES. The minimum value is 1. The maximum value is 255. The value of 32 is usually sufficient.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avdtp_control.h

Functions

	Name	Description
	_bt_avdtp_add_param_byte	This is function _bt_avdtp_add_param_byte.
	_bt_avdtp_add_param_uint	This is function _bt_avdtp_add_param_uint.
	_bt_avdtp_add_param_uintn	This is function _bt_avdtp_add_param_uintn.
	_bt_avdtp_control_channel_event_handler	This is function _bt_avdtp_control_channel_event_handler.
	_bt_avdtp_get_control_channel	This is function _bt_avdtp_get_control_channel.
	_bt_avdtp_open_control_channel_ex	This is function _bt_avdtp_open_control_channel_ex.

Macros

	Name	Description
	__AVDTP_CONTROL_H	This is macro __AVDTP_CONTROL_H.
	AVDTP_CMD_ABORT	This is macro AVDTP_CMD_ABORT.
	AVDTP_CMD_CLOSE	This is macro AVDTP_CMD_CLOSE.
	AVDTP_CMD_DELAYREPORT	This is macro AVDTP_CMD_DELAYREPORT.
	AVDTP_CMD_DISCOVER	This is macro AVDTP_CMD_DISCOVER.
	AVDTP_CMD_GET_ALL_CAPABILITIES	This is macro AVDTP_CMD_GET_ALL_CAPABILITIES.
	AVDTP_CMD_GET_CAPABILITIES	This is macro AVDTP_CMD_GET_CAPABILITIES.
	AVDTP_CMD_GET_CONFIGURATION	This is macro AVDTP_CMD_GET_CONFIGURATION.
	AVDTP_CMD_OPEN	This is macro AVDTP_CMD_OPEN.
	AVDTP_CMD_RECONFIGURE	This is macro AVDTP_CMD_RECONFIGURE.
	AVDTP_CMD_SECURITY_CONTROL	This is macro AVDTP_CMD_SECURITY_CONTROL.
	AVDTP_CMD_SET_CONFIGURATION	This is macro AVDTP_CMD_SET_CONFIGURATION.
	AVDTP_CMD_START	This is macro AVDTP_CMD_START.
	AVDTP_CMD_SUSPEND	This is macro AVDTP_CMD_SUSPEND.
	AVDTP_CTLR_CHANNEL_EVT_CONNECTED	This is macro AVDTP_CTLR_CHANNEL_EVT_CONNECTED.
	AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED	This is macro AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED.
	AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED	This is macro AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED.
	AVDTP_CTLR_CHANNEL_EVT_NOTHING	This is macro AVDTP_CTLR_CHANNEL_EVT_NOTHING.
	AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED	This is macro AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED.
	AVDTP_CTRL_CHANNEL_STATE_CONNECTED	This is macro AVDTP_CTRL_CHANNEL_STATE_CONNECTED.
	AVDTP_CTRL_CHANNEL_STATE_CONNECTING	This is macro AVDTP_CTRL_CHANNEL_STATE_CONNECTING.
	AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED	This is macro AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED.
	AVDTP_CTRL_MESSAGE_TYPE_ACCEPT	This is macro AVDTP_CTRL_MESSAGE_TYPE_ACCEPT.
	AVDTP_CTRL_MESSAGE_TYPE_COMMAND	This is macro AVDTP_CTRL_MESSAGE_TYPE_COMMAND.

	AVDTP_CTRL_MESSAGE_TYPE_FLD	This is macro AVDTP_CTRL_MESSAGE_TYPE_FLD.
	AVDTP_CTRL_MESSAGE_TYPE_GENERAL_REJECT	This is macro AVDTP_CTRL_MESSAGE_TYPE_GENERAL_REJECT.
	AVDTP_CTRL_MESSAGE_TYPE_REJECT	This is macro AVDTP_CTRL_MESSAGE_TYPE_REJECT.
	AVDTP_CTRL_PACKET_TYPE_CONTINUE	This is macro AVDTP_CTRL_PACKET_TYPE_CONTINUE.
	AVDTP_CTRL_PACKET_TYPE_END	This is macro AVDTP_CTRL_PACKET_TYPE_END.
	AVDTP_CTRL_PACKET_TYPE_FLD	This is macro AVDTP_CTRL_PACKET_TYPE_FLD.
	AVDTP_CTRL_PACKET_TYPE_SIGNLE	This is macro AVDTP_CTRL_PACKET_TYPE_SIGNLE.
	AVDTP_CTRL_PACKET_TYPE_START	This is macro AVDTP_CTRL_PACKET_TYPE_START.

Structures

	Name	Description
	_bt_avdtp_control_channel_t	This is type bt_avdtp_control_channel_t.
	_bt_avdtp_control_cmd_t	This is type bt_avdtp_control_cmd_t.
	_bt_avdtp_ctrl_evt_data_received_t	This is type bt_avdtp_ctrl_evt_data_received_t.
	bt_avdtp_control_channel_t	This is type bt_avdtp_control_channel_t.
	bt_avdtp_control_cmd_t	This is type bt_avdtp_control_cmd_t.
	bt_avdtp_ctrl_evt_data_received_t	This is type bt_avdtp_ctrl_evt_data_received_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avdtp_private.h

Functions

	Name	Description
	_bt_avdtp_allocate_buffers	This is function _bt_avdtp_allocate_buffers.
	_bt_avdtp_allocate_cmd	This is function _bt_avdtp_allocate_cmd.
	_bt_avdtp_allocate_sep	This is function _bt_avdtp_allocate_sep.
	_bt_avdtp_allocate_sep_config	This is function _bt_avdtp_allocate_sep_config.
	_bt_avdtp_allocate_stream	This is function _bt_avdtp_allocate_stream.
	_bt_avdtp_allocate_transport_channel	This is function _bt_avdtp_allocate_transport_channel.
	_bt_avdtp_allocate_transport_session_id	This is function _bt_avdtp_allocate_transport_session_id.
	_bt_avdtp_begin_tc_channel_operation	This is function _bt_avdtp_begin_tc_channel_operation.
	_bt_avdtp_commit_tc_channel_operation	This is function _bt_avdtp_commit_tc_channel_operation.
	_bt_avdtp_control_channel_accept_handler	This is function _bt_avdtp_control_channel_accept_handler.
	_bt_avdtp_control_channel_cmd_handler	This is function _bt_avdtp_control_channel_cmd_handler.
	_bt_avdtp_control_channel_reject_handler	This is function _bt_avdtp_control_channel_reject_handler.
	_bt_avdtp_execute_tc_channel_operation	This is function _bt_avdtp_execute_tc_channel_operation.
	_bt_avdtp_find_listening_stream	This is function _bt_avdtp_find_listening_stream.
	_bt_avdtp_find_stream	This is function _bt_avdtp_find_stream.
	_bt_avdtp_find_stream_by_remote_sep_id	This is function _bt_avdtp_find_stream_by_remote_sep_id.
	_bt_avdtp_find_stream_by_sep_id	This is function _bt_avdtp_find_stream_by_sep_id.
	_bt_avdtp_free_cmd	This is function _bt_avdtp_free_cmd.
	_bt_avdtp_free_sep_config	This is function _bt_avdtp_free_sep_config.
	_bt_avdtp_free_stream	This is function _bt_avdtp_free_stream.
	_bt_avdtp_free_transport_channel	This is function _bt_avdtp_free_transport_channel.
	_bt_avdtp_init_cmd_buffers	This is function _bt_avdtp_init_cmd_buffers.
	_bt_avdtp_init_sep_config_buffers	This is function _bt_avdtp_init_sep_config_buffers.
	_bt_avdtp_init_signal	This is function _bt_avdtp_init_signal.
	_bt_avdtp_is_sep_inuse	This is function _bt_avdtp_is_sep_inuse.
	_bt_avdtp_open_control_channel_ex	This is function _bt_avdtp_open_control_channel_ex.
	_bt_avdtp_read_caps	This is function _bt_avdtp_read_caps.
	_bt_avdtp_register_transport_channel_for_operation	This is function _bt_avdtp_register_transport_channel_for_operation.
	_bt_avdtp_send_command	This is function _bt_avdtp_send_command.

	_bt_avdtp_send_media_packet	This is function _bt_avdtp_send_media_packet.
	_bt_avdtp_set_signal	This is function _bt_avdtp_set_signal.
	_bt_avdtp_transport_l2cap_read_data_callback	This is function _bt_avdtp_transport_l2cap_read_data_callback.
	_bt_avdtp_transport_l2cap_state_changed_callback	This is function _bt_avdtp_transport_l2cap_state_changed_callback.
	_bt_avdtp_write_caps	This is function _bt_avdtp_write_caps.

Macros

	Name	Description
	__AVDTP_PRIVATE_H	This is macro __AVDTP_PRIVATE_H.
	AVDTP_TC_OPCODE_CONNECT	This is macro AVDTP_TC_OPCODE_CONNECT.
	AVDTP_TC_OPCODE_DISCONNECT	This is macro AVDTP_TC_OPCODE_DISCONNECT.
	AVDTP_TC_STATUS_ERROR	This is macro AVDTP_TC_STATUS_ERROR.
	AVDTP_TC_STATUS_SUCCESS	This is macro AVDTP_TC_STATUS_SUCCESS.

Types

	Name	Description
	bt_avdtp_transport_op_callback_fp	This is type bt_avdtp_transport_op_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avrcp.h

Functions

	Name	Description
	bt_avrcp_add_to_now_playing	brief Add to "now playing" list. ingroup avrcp details This function adds a media element specified by p element_id to the "now playing" list on the target. param channel AVRCP channel. param scope The scope in which the p element_id is valid. This value can be on the following values: li AVC_SCOPE_MEDIA_PLAYER_LIST li AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM li AVC_SEARCH li AVC_NOW_PLAYING param element_id UID of the media element to be added to the "now playing" list. param counter UID counter. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_cancel_find	brief Cancel finding Targets ingroup avrcp details This function stops AVRCP layer looking for targets on nearby devices. As a result of this operation the AVRCP_EVT_SEARCH_COMPLETED event will be generated. return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.
	bt_avrcp_cancel_listen	brief Cancel listening for incoming connections. ingroup avrcp details This function stops listening for incoming connections on a specified channel. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_connect	brief Connect to a remote device. ingroup avrcp details This function establishes a connection to a remote device specified by the p remote_address. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVRCP callback. The events generated will either be AVRCP_EVT_CONTROL_CHANNEL_CONNECTED or AVRCP_EVT_CONTROL_CONNECTION_FAILED. param channel AVRCP channel. param remote_address The address of a remote device. return li c TRUE if connection establishment has been started. li c FALSE otherwise.... more

	bt_avrcp_create_channel	brief Allocate AVRCP channel ingroup avrcp details This function allocates a new incoming AVRCP channel. The channel is intended to be used to accept a connection from a remote device. There can be only one incoming channel. param mgr AVRCP manager. param create_browsing_channel Defines weather a browsing channel will be created. return li A pointer to the new AVRCP channel if the function succeeds. li c NULL otherwise.
	bt_avrcp_create_outgoing_channel	brief Allocate AVRCP channel ingroup avrcp details This function allocates a new outgoing AVRCP channel. The channel is intended to be used to create a connection to a remote device. There can be multiple outgoing channels. param mgr AVRCP manager. param create_browsing_channel Defines weather a browsing channel will be created. return li A pointer to the new AVRCP channel if the function succeeds. li c NULL otherwise.
	bt_avrcp_destroy_channel	brief Destroy AVRCP channel. ingroup avrcp details This function frees memory used by the channel. The channel has to exist and be in the "idle" state for this function to succeed. I.e. the channel has to be disconnected before this function can be called. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_disconnect	brief Disconnect from a remote device. ingroup avrcp details This function closes a connection to a remote device. param channel AVRCP channel. return li c TRUE if disconnection has been started. li c FALSE otherwise. No events will be generated.
	bt_avrcp_find_controller	brief Find Controller ingroup avrcp details This function looks for a controller on a remote device specified by p deviceAddress. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer to arbitrary data to be passed to the p... more
	bt_avrcp_find_target	brief Find Target ingroup avrcp details This function looks for a target on a remote device specified by p deviceAddress. param deviceAddress The address of a remote device. param callback The callback function that will be called when search has completed. param client_callback The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The c evt parameter of the callback can be one of the following values: li SDP_CLIENT_STATE_IDLE li SDP_CLIENT_STATE_CONNECTING li SDP_CLIENT_STATE_DISCONNECTING li SDP_CLIENT_STATE_CONNECTED param callback_param A pointer to arbitrary data to be passed to the p... more
	bt_avrcp_find_targets	brief Find Targets ingroup avrcp details This function looks for targets on nearby devices. The AVRCP_EVT_SEARCH_COMPLETED event is generated when the search has completed. param search_length The amount of time the search will be performed for. return li c TRUE if the function succeeds. li c FALSE otherwise. the callback is not called in this case.
	bt_avrcp_get_browsing_channel_state	brief Get AVCTP browsing channel state. ingroup avrcp details This function returns status of the AVCTP browsing channel. param channel AVRCP channel. return Returns of the following values: li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING

	bt_avrcp_get_channel_remote_address	brief Get channel's remote BT address. ingroup avrcp details This function returns the address of the remote device associated with the channel. param channel AVRCP channel. return li The address of the remote device if channel is connected. li NULL otherwise.
	bt_avrcp_get_company_id_list	brief Get Company ID list. ingroup avrcp details This function requests a list of company id's supported by the remote device param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_get_control_channel_state	brief Get AVCTP control channel state. ingroup avrcp details This function returns status of the AVCTP control channel. param channel AVRCP channel. return Returns of the following values: li AVCTP_CHANNEL_STATE_FREE li AVCTP_CHANNEL_STATE_IDLE li AVCTP_CHANNEL_STATE_CONNECTING li AVCTP_CHANNEL_STATE_CONNECTED li AVCTP_CHANNEL_STATE_DISCONNECTING
	bt_avrcp_get_current_player_application_setting_value	brief Get current player setting values. ingroup avrcp details This function requests a list of current set values for the player application on the target. The list of attribute ids whose values have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_setting_current_values_t::setting_id_list to a list of player setting attribute ids, bt_av_player_setting_current_values_t::count to the number of entries in the list, bt_av_player_setting_current_values_t::setting_value_id_list to a buffer where returned values will be stored. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_current_values_t structure initialized as stated above. return li c TRUE if the function succeeds. li c... more
	bt_avrcp_get_element_attributes	brief Get media element attributes. ingroup avrcp details This function requests the attributes of the element specified with p element_id. note Currently p element_id is ignored. The AVRCP specification defines that only UID 0 can be used to return the attributes of the current track. param channel AVRCP channel. param element_id UID of the media element whose attributes are requested. param attr_mask Bitmask that defines which attributes are requested. This value can be a combination of the following values: li AVC_MEDIA_ATTR_FLAG_TITLE li AVC_MEDIA_ATTR_FLAG_ARTIST li AVC_MEDIA_ATTR_FLAG_ALBUM li AVC_MEDIA_ATTR_FLAG_NUMBER li AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER li AVC_MEDIA_ATTR_FLAG_GENRE li AVC_MEDIA_ATTR_FLAG_PLAYING_TIME li AVC_MEDIA_ATTR_FLAG_ALL return li c TRUE if... more
	bt_avrcp_get_hci_connection	brief Get HCI connection for a channel ingroup avrcp details This function returns a pointer to a structure that describes an HCI connection a channel is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call ::bt_hci_disconnect. param channel AVRCP channel. return li c Pointer to a structure that describes an HCI connection if the function succeeds. li c NULL otherwise. The function fails only if a channel specified by the p channel parameter li does... more
	bt_avrcp_get_mgr	brief Return a pointer to an instance of the AVRCP manager. ingroup avrcp details This function returns a pointer to an instance of the AVRCP manager. There is only one instance of the manager allocated by the stack.

	bt_avrcp_get_play_status	brief Get playback status. ingroup avrcp details This function requests the status of the currently playing media at the target. param channel AVRCP channel. param repeat_interval Interval in milliseconds at which AVRCP polls the target for playback status. If 0 is passed polling is stopped. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_get_player_application_setting_attr_text	brief Get player setting attribute text. ingroup avrcp details This function requests the target device to provide supported player application setting attribute displayable text for the provided player application setting attributes. The list of attribute ids whose displayable text have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_settings_text_t::setting_id_list to a list of player setting attribute ids, bt_av_player_settings_text_t::count to the number of entries in the list, bt_av_player_settings_text_t::setting_text_list to a buffer where returned values will be stored. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_settings_text_t structure initialized as stated above. return li c... more
	bt_avrcp_get_player_application_setting_value_text	brief Get player setting value text. ingroup avrcp details This function request the target device to provide target supported player application setting value displayable text for the provided player application setting attribute values. The list of attribute ids whose value displayable text have to be returned is passed via the p response_buffer parameter. The caller has to set bt_av_player_setting_values_text_t::setting_value_id_list to a list of player setting attribute value ids, bt_av_player_setting_values_text_t::count to the number of entries in the list, bt_av_player_setting_values_text_t::setting_value_text_list to a buffer where returned values will be stored. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_values_text_t structure initialized as stated... more
	bt_avrcp_get_supported_event_id_list	brief Get supported events. ingroup avrcp details This function requests a list of events supported by the remote device param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_inform_battery_status	brief Inform controller's battery status. ingroup avrcp details This function is used to inform the target about the controller's battery status. param channel AVRCP channel. param status Battery status. This can be one of the following values: li AVC_BATTERY_STATUS_NORMAL li AVC_BATTERY_STATUS_WARNING li AVC_BATTERY_STATUS_CRITICAL li AVC_BATTERY_STATUS_EXTERNAL li AVC_BATTERY_STATUS_FULL_CHARGE return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_inform_displayable_character_set	brief Inform displayable character set. ingroup avrcp details This function informs the list of character set supported by the controller to the target. param channel AVRCP channel. param charset_list List of displayable character sets. param charset_count Number of entries in the list. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_init_controller	brief Initialize AVRCP to be used in controller mode. ingroup avrcp details This function initializes the AVRCP layer of the stack in controller mode. It must be called prior to any other AVRCP function can be called.

	bt_avrcp_init_target	brief Initialize AVRCP to be used in target mode. ingroup avrcp details This function initializes the AVRCP layer of the stack in target mode. It must be called prior to any other AVRCP function can be called. param company_id The 24-bit unique ID obtained from the IEEE Registration Authority Committee. If the vendor of a TG device does not have the unique ID, the value 0xFFFFFF may be used. param supported_events Bitmask that specifies events supported by the target. This value can be a combination of the following values: li AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED li AVC_EVENT_FLAG_TRACK_CHANGED li AVC_EVENT_FLAG_TRACK_REACHED_END li AVC_EVENT_FLAG_TRACK_REACHED_START li AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED li... more
	bt_avrcp_list_player_application_setting_attributes	brief Get supported player setting attributes. ingroup avrcp details This function request the target device to provide target supported player application setting attributes. The list of attribute ids is stored in the setting_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_settings_t::setting_id_list to a buffer where returned values will be stored and bt_av_player_settings_t::count to the number of entries in the list. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_settings_t structure initialized as stated above. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_list_player_application_setting_values	brief Get player setting attribute values. ingroup avrcp details This function requests the target device to list the set of possible values for the requested player application setting attribute. The list of attribute value ids is stored in the setting_value_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_setting_values_t::setting_value_id_list to a buffer where returned values will be stored and bt_av_player_setting_values_t::count to the number of entries in the list. param channel AVRCP channel. param response_buffer Pointer to bt_av_player_setting_values_t structure initialized as stated above. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_listen	brief Listen for incoming connections. ingroup avrcp details This function enables incoming connections on the specified AVRCP channel. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_play_item	brief Play media item. ingroup avrcp details This function starts playing an item indicated by the UID. param channel AVRCP channel. param scope The scope in which the p element_id is valid. This value can be on the following values: li AVC_SCOPE_MEDIA_PLAYER_LIST li AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM li AVC_SEARCH li AVC_NOW_PLAYING param element_id UID of the media element to be played. param counter UID counter. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_register_notification	brief Register notification. ingroup avrcp details This function registers with the target to receive notifications asynchronously based on specific events occurring. param channel AVRCP channel. param event_id Event Id. This value can be one of the following values: li AVC_EVENT_PLAYBACK_STATUS_CHANGED li AVC_EVENT_TRACK_CHANGED li AVC_EVENT_TRACK_REACHED_END li AVC_EVENT_TRACK_REACHED_START li AVC_EVENT_PLAYBACK_POS_CHANGED li AVC_EVENT_BATT_STATUS_CHANGED li AVC_EVENT_SYSTEM_STATUS_CHANGED li AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED li AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED li AVC_EVENT_AVAILABLE_PLAYERS_CHANGED li AVC_EVENT_ADDRESSED_PLAYER_CHANGED li AVC_EVENT_UIDS_CHANGED li AVC_EVENT_VOLUME_CHANGED param playback_interval The time interval (in seconds) at which the change in playback position will be notified. Applicable for AVC_EVENT_PLAYBACK_POS_CHANGED event. return li c TRUE if the function succeeds. li c FALSE otherwise.

	bt_avrcp_register_notifications	brief Register notifications. ingroup avrcp details This function registers with the target to receive notifications asynchronously based on specific events occurring. This function is used to register multiple notifications with one call. note This function cannot be used to register for AVC_EVENT_PLAYBACK_POS_CHANGED event. param channel AVRCP channel. param event_mask Bitmask that specifies which events to register for. This value can be a combination of the following values: li AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED li AVC_EVENT_FLAG_TRACK_CHANGED li AVC_EVENT_FLAG_TRACK_REACHED_END li AVC_EVENT_FLAG_TRACK_REACHED_START li AVC_EVENT_FLAG_BATT_STATUS_CHANGED li AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED li AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED li AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED li AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED li AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED li AVC_EVENT_FLAG_UIDS_CHANGED li AVC_EVENT_FLAG_VOLUME_CHANGED li AVC_EVENT_FLAG_ALL return li c TRUE if the function... more
	bt_avrcp_send_cmd	brief Send AVRCP command ingroup avrcp details This function sends a command to the remote device. param channel AVRCP channel. param command Command to be sent. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_set_absolute_volume	brief Set absolute volume. ingroup avrcp details This function is used to set an absolute volume to be used by the rendering device. param channel AVRCP channel. param volume Volume return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_set_addressed_player	brief Set addressed player. ingroup avrcp details This function is used to inform the target of which media player the controller wishes to control. param channel AVRCP channel. param player_id Player Id. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_set_player_application_setting_value	brief Set player setting attribute values. ingroup avrcp details This function requests to set the player application setting list of player application setting values on the target device for the corresponding defined list of setting attributes. for the requested player application setting attribute. The list of attribute value ids is stored in the setting_value_id_list member of the p response_buffer parameter. The caller has to set bt_av_player_setting_values_t::setting_value_id_list to a buffer where returned values will be stored and bt_av_player_setting_values_t::count to the number of entries in the list. param channel AVRCP channel. param attr_id_list List of setting attribute ids. param attr_value_list List of... more
	bt_avrcp_start	brief Start the AVRCP layer. ingroup avrcp details In order to be notified of various events a consumer of the AVRCP layer has to register a callback function. The stack will call the callback function whenever a new event has been generated passing the code of the event as the second parameter. bt_avrcp_start() stores pointers to the c callback and c callback_param in the bt_avrcp_mgr_t structure. param mgr AVRCP manager. param callback The callback function that will be called when the AVRCP generates an event. param callback_param A pointer to arbitrary data to be passed to the c callback callback.... more
	bt_avrcp_tg_send_element_attributes	brief Send media element attributes ingroup avrcp details This function is used to send the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_SYSTEM_STATUS_CHANGED event will be notified. param status Battery status

	bt_avrcp_tg_set_absolute_volume	brief Set absolute volume ingroup avrcp details This function is used to set the absolute volume when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_VOLUME_CHANGED event will be notified. param track_id Track Id param song_length The length of the current track. param song_position The position of the current track.
	bt_avrcp_tg_set_battery_status	brief Set battery status ingroup avrcp details This function is used to set the battery status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_BATT_STATUS_CHANGED event will be notified. param status Battery status
	bt_avrcp_tg_set_channel_absolute_volume	This is function bt_avrcp_tg_set_channel_absolute_volume.
	bt_avrcp_tg_set_current_track	brief Set current track ingroup avrcp details This function is used to set the current track when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_TRACK_CHANGED event will be notified. param track_id Track Id param song_length The length of the current track. param song_position The position of the current track.
	bt_avrcp_tg_set_play_status	local target control * • brief Set playback status • ingroup avrcp * • details This function is used to set the playback status when AVRCP is running in target mode. • If there are active connections with controllers, the ones that registered • for AVC_EVENT_PLAYBACK_STATUS_CHANGED event will be notified. * • param song_length The length of the current track. • param song_position The position of the current track. • param play_status Playback status. This value can be one of the following values: • li AVC_PLAY_STATUS_STOPPED • li AVC_PLAY_STATUS_PLAYING • li AVC_PLAY_STATUS_PAUSED • li AVC_PLAY_STATUS_FW_SEEK • li AVC_PLAY_STATUS_REV_SEEK
	bt_avrcp_tg_set_system_status	brief Set system status ingroup avrcp details This function is used to set the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_SYSTEM_STATUS_CHANGED event will be notified. param status Battery status

Macros

	Name	Description
	__AVRCP_H	This is macro __AVRCP_H.
	AVCTP_ERROR_BAD_STATE	This is macro AVCTP_ERROR_BAD_STATE.
	AVCTP_ERROR_SUCCESS	This is macro AVCTP_ERROR_SUCCESS.
	AVCTP_MESSAGE_PACKET_TYPE_CONTINUE	This is macro AVCTP_MESSAGE_PACKET_TYPE_CONTINUE.
	AVCTP_MESSAGE_PACKET_TYPE_END	This is macro AVCTP_MESSAGE_PACKET_TYPE_END.
	AVCTP_MESSAGE_PACKET_TYPE_SINGLE	This is macro AVCTP_MESSAGE_PACKET_TYPE_SINGLE.
	AVCTP_MESSAGE_PACKET_TYPE_START	This is macro AVCTP_MESSAGE_PACKET_TYPE_START.
	AVCTP_MESSAGE_TYPE_COMMAND	This is macro AVCTP_MESSAGE_TYPE_COMMAND.

	AVCTP_MESSAGE_TYPE_RESPONSE	This is macro AVCTP_MESSAGE_TYPE_RESPONSE.
	AVRCP_BTSIG_COMPANY_ID	This is macro AVRCP_BTSIG_COMPANY_ID.
	AVRCP_CHANNEL_FLAG_LISTENING	This is macro AVRCP_CHANNEL_FLAG_LISTENING.
	AVRCP_CHANNEL_FLAG_PLAY_STATUS_REQUESTED	This is macro AVRCP_CHANNEL_FLAG_PLAY_STATUS_REQUESTED.
	AVRCP_CHANNEL_FLAG_REGISTERING_NOTIFICATIONS	This is macro AVRCP_CHANNEL_FLAG_REGISTERING_NOTIFICATIONS.
	AVRCP_CHANNEL_FLAG_SENDING	This is macro AVRCP_CHANNEL_FLAG_SENDING.
	AVRCP_CHANNEL_STATE_CONNECTED	This is macro AVRCP_CHANNEL_STATE_CONNECTED.
	AVRCP_CHANNEL_STATE_CONNECTING	This is macro AVRCP_CHANNEL_STATE_CONNECTING.
	AVRCP_CHANNEL_STATE_DISCONNECTING	This is macro AVRCP_CHANNEL_STATE_DISCONNECTING.
	AVRCP_CHANNEL_STATE_FREE	This is macro AVRCP_CHANNEL_STATE_FREE.
	AVRCP_CHANNEL_STATE_IDLE	This is macro AVRCP_CHANNEL_STATE_IDLE.
	AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED	< This event is generated when a local device received a response to a "add to now playing" request.
	AVRCP_EVT_ADDRESSED_PLAYER_CHANGED	< This event is generated when a local device received a "addressed player changed" notification.
	AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED	< This event is generated when a local device received a "available players changed" notification.
	AVRCP_EVT_BATT_STATUS_CHANGED	< This event is generated when a local device received a "battery status changed" notification.
	AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED	< This event is generated when a local device received a "battery status of controller" command.
	AVRCP_EVT_BROWSING_CHANNEL_CONNECTED	< This event is generated when a browsing channel between two AVRCP entities has been established.
	AVRCP_EVT_BROWSING_CHANNEL_DISCONNECTED	< This event is generated when a browsing channel between two AVRCP entities has been terminated.
	AVRCP_EVT_BROWSING_CONNECTION_FAILED	< This event is generated when a local device failed to create a browsing channel between two AVRCP entities.
	AVRCP_EVT_COMPANY_ID_LIST_RECEIVED	< This event is generated when a local device received a response to a "get company id" request.
	AVRCP_EVT_CONTROL_CHANNEL_CONNECTED	< This event is generated when a control channel between two AVRCP entities has been established.
	AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED	< This event is generated when a control channel between two AVRCP entities has been terminated.
	AVRCP_EVT_CONTROL_CONNECTION_FAILED	< This event is generated when a local device failed to create a control channel between two AVRCP entities.
	AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED	< This event is generated when a local device received a "displayable chracter set command" request.
	AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED	< This event is generated when a local device received a "get element attributes" request.
	AVRCP_EVT_EVENT_ID_LIST_RECEIVED	< This event is generated when a local device received a response to a "get supported events" request.
	AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED	< This event is generated when a local device received a response to a "get media element attributes" request.
	AVRCP_EVT_GET_PLAY_STATUS_RECEIVED	< This event is generated when a local device received a response to a "get play status" request.
	AVRCP_EVT_INFORM_BATTERY_STATUS_OF_CT_COMPLETED	< This event is generated when a local device received a response to a "inform battery status" request.

	AVRCP_EVT_INFORM_DISPLAYABLE_CHARACTER_SET_COMPLETED	< This event is generated when a local device received a response to a "inform displayable character set" request.
	AVRCP_EVT_NOTHING	addtogroup avrcp @{ @name Events details The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.
	AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED	< This event is generated when a local device received a "now playing content changed" notification.
	AVRCP_EVT_PANEL_COMMAND_RECEIVED	< This event is generated when a local device received a PASS THROUGH command.
	AVRCP_EVT_PANEL_RESPONSE_RECEIVED	< This event is generated when a local device received a response to a PASS THROUGH command.
	AVRCP_EVT_PLAY_ITEM_COMPLETED	< This event is generated when a local device received a response to a "play item" request.
	AVRCP_EVT_PLAYBACK_POS_CHANGED	< This event is generated when a local device received a "playback position changed" notification.
	AVRCP_EVT_PLAYBACK_STATUS_CHANGED	< This event is generated when a local device received a "play status changed" notification.
	AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED	< This event is generated when a local device received a "player application setting changed" notification.
	AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED	< This event is generated when a local device received a response to a "get current player setting attribute values" request.
	AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED	< This event is generated when a local device received a response to a "get supported player setting attributes" request.
	AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED	< This event is generated when a local device received a response to a "get player setting attributes displayable text" request.
	AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED	< This event is generated when a local device received a response to a "get player setting attribute values" request.
	AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED	< This event is generated when a local device received a response to a "get player setting attribute values displayable text" request.
	AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED	< This event is generated when a local device received a "register notification" request.
	AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED	< This event is generated when a local device received a response to a "register notification" request.
	AVRCP_EVT_SEARCH_COMPLETED	< This event is generated when a local device completed searching for nearby targets.
	AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED	< This event is generated when a local device received a response to a "set absolute volume" request.
	AVRCP_EVT_SET_ABSOLUTE_VOLUME_REQUESTED	< This event is generated when a local device received a "set absolute volume" request.
	AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED	< This event is generated when a local device received a response to a "set addressed player" request.
	AVRCP_EVT_SET_PLAYER_SETTING_VALUES_COMPLETED	< This event is generated when a local device received a response to a "set player setting attribute values" request.
	AVRCP_EVT_SYSTEM_STATUS_CHANGED	< This event is generated when a local device received a "system status changed" notification.
	AVRCP_EVT_TRACK_CHANGED	< This event is generated when a local device received a "track changed changed" notification.
	AVRCP_EVT_TRACK_REACHED_END	< This event is generated when a local device received a "track reached end" notification.

	AVRCP_EVT_TRACK_REACHED_START	< This event is generated when a local device received a "track reached start" notification.
	AVRCP_EVT_UIDS_CHANGED	< This event is generated when a local device received a "UIDs changed" notification.
	AVRCP_EVT_VOLUME_CHANGED	< This event is generated when a local device received a "volume changed" notification.
	AVRCP_MANAGER_FLAG_SEARCHING	This is macro AVRCP_MANAGER_FLAG_SEARCHING.
	AVRCP_MANAGER_STATE_IDLE	This is macro AVRCP_MANAGER_STATE_IDLE.

Structures

	Name	Description
	_bt_avrcp_channel_t	brief AVRCP channel description ingroup avrcp details This structure is used to hold information about an AVRCP channel.
	_bt_avrcp_device_s	This is type bt_avrcp_device_t.
	_bt_avrcp_evt_channel_connected_t	brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_CONNECTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_connected - when a control channel between two devices has been established.
	_bt_avrcp_evt_channel_disconnected_t	brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_disconnected - when a control channel between two devices has been terminated.
	_bt_avrcp_evt_connection_failed_t	brief Parameter to AVRCP_EVT_CONTROL_CONNECTION_FAILED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::connection_failed - when a local device failed to create a control channel between two AVRCP entities.
	_bt_avrcp_evt_panel_command_received_t	brief Parameter to AVRCP_EVT_PANEL_COMMAND_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_command_received - when a local device received a PASS THROUGH command.
	_bt_avrcp_evt_panel_response_received_t	brief Parameter to AVRCP_EVT_PANEL_RESPONSE_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_response_received - when a local device received a response to a PASS THROUGH command.
	_bt_avrcp_evt_register_events_completed_t	This is type bt_avrcp_evt_register_events_completed_t.
	_bt_avrcp_evt_search_completed_s	brief Parameter to AVRCP_EVT_SEARCH_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::device_search - when searching for nearby devices has finished.
	_bt_avrcp_mgr_t	brief AVRCP manager. ingroup avrcp details A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with c bt_avrcp_get_mgr().
	bt_avrcp_device_t	This is type bt_avrcp_device_t.
	bt_avrcp_evt_channel_connected_t	brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_CONNECTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_connected - when a control channel between two devices has been established.
	bt_avrcp_evt_channel_disconnected_t	brief Parameter to AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_disconnected - when a control channel between two devices has been terminated.

	bt_avrcp_evt_connection_failed_t	brief Parameter to AVRCP_EVT_CONTROL_CONNECTION_FAILED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::connection_failed - when a local device failed to create a control channel between two AVRCP entities.
	bt_avrcp_evt_panel_command_received_t	brief Parameter to AVRCP_EVT_PANEL_COMMAND_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_command_received - when a local device received a PASS THROUGH command.
	bt_avrcp_evt_panel_response_received_t	brief Parameter to AVRCP_EVT_PANEL_RESPONSE_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_response_received - when a local device received a response to a PASS THROUGH command.
	bt_avrcp_evt_register_events_completed_t	This is type bt_avrcp_evt_register_events_completed_t.
	bt_avrcp_evt_search_completed_t	brief Parameter to AVRCP_EVT_SEARCH_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::device_search - when searching for nearby devices has finished.

Types

	Name	Description
	bt_avrcp_channel_t	This is type bt_avrcp_channel_t.
	bt_avrcp_find_callback_fp	brief Find Controller/Target callback. ingroup avrcp details This callback is called when search for Controller/Target has finished. param found This can be one of the following values: li c TRUE if Controller/Target was found. li c FALSE otherwise. param callback_param A pointer to an arbitrary data set by a call to bt_avrcp_find_target/bt_avrcp_find_controller.
	bt_avrcp_mgr_callback_fp	brief AVRCP application callback. ingroup avrcp details In order to be notified of various events a consumer of the AVRCP layer has to register a callback function (done with bt_avrcp_start()). The stack will call that function whenever a new event has been generated. param mgr AVRCP manager. param evt AVRCP event. The event can be one of the following values: @arg AVRCP_EVT_CONTROL_CHANNEL_CONNECTED This event is generated when a control channel between two AVRCP entities has been established. @arg AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED This event is generated when a control channel between two AVRCP entities has been terminated. @arg AVRCP_EVT_CONTROL_CONNECTION_FAILED This event is... more
	bt_avrcp_mgr_t	This is type bt_avrcp_mgr_t.

Unions

	Name	Description
	_bt_avrcp_event_u	brief Parameter to an application callback. ingroup avrcp details This union is used to pass event specific data to the AVRCP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.
	bt_avrcp_event_t	brief Parameter to an application callback. ingroup avrcp details This union is used to pass event specific data to the AVRCP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avrcp_command.h

Functions

	Name	Description
	bt_avrcp_get_subuint_info	brief Get subunit info ingroup avrcp details This function is used to request subunit info from the target. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_get_unit_info	brief Get unit info ingroup avrcp details This function is used to request unit info from the target. param channel AVRCP channel. return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_send_button_click	brief Send AV/C Panel Subunit "click" PASS THROUGH command ingroup avrcp details This function is used to send a button click. Two PATH THROUGTH commands are sent. The first command has button state set to AVC_PANEL_BUTTON_PRESSED. The second command gas button state set to AVC_PANEL_BUTTON_RELEASED param channel AVRCP channel. param button_id Operation Id. This value can be on of the AVC_PANEL_OPID_... constants return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_send_panel_control	brief Send AV/C Panel Subunit "control" PASS THROUGH command ingroup avrcp details This function is used to send AV/C Panel Subunit PASS THROUGH command with command type set to AVC_CTYPE_CONTROL. param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants param button_state Button state. This can be on of the following values: li AVC_PANEL_BUTTON_PRESSED li AVC_PANEL_BUTTON_RELEASED return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_send_press_panel_control	brief Send AV/C Panel Subunit "pressed" PASS THROUGH command ingroup avrcp details This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC_PANEL_BUTTON_PRESSED. param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_send_release_panel_control	brief Send AV/C Panel Subunit "released" PASS THROUGH command ingroup avrcp details This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC_PANEL_BUTTON_RELEASED. param channel AVRCP channel. param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants return li c TRUE if the function succeeds. li c FALSE otherwise.
	bt_avrcp_send_simple_panel_cmd	brief Send AV/C Panel Subunit PASS THROUGH command ingroup avrcp details This function is used to send AV/C Panel Subunit PASS THROUGH command to the target. param channel AVRCP channel. param ctype Command type. This value can be on of the following values: li AVC_CTYPE_CONTROL 0 li AVC_CTYPE_STATUS 1 li AVC_CTYPE_SPECIFIC_IQUIRY 2 li AVC_CTYPE_NOTIFY 3 li AVC_CTYPE_GENERAL_INQUORY 4 param opid Operation Id. This value can be on of the AVC_PANEL_OPID_... constants param button_state Button state. This can be on of the following values: li AVC_PANEL_BUTTON_PRESSED li AVC_PANEL_BUTTON_RELEASED return li c TRUE if the function succeeds. li c FALSE otherwise.

Macros

	Name	Description
	__AVRCP_COMMAND_H	This is macro __AVRCP_COMMAND_H.
	AVC_BATTERY_STATUS_CRITICAL	This is macro AVC_BATTERY_STATUS_CRITICAL.
	AVC_BATTERY_STATUS_EXTERNAL	This is macro AVC_BATTERY_STATUS_EXTERNAL.
	AVC_BATTERY_STATUS_FULL_CHARGE	This is macro AVC_BATTERY_STATUS_FULL_CHARGE.
	AVC_BATTERY_STATUS_NORMAL	• addtogroup avrcp • @{ • @name Battery status

	AVC_BATTERY_STATUS_WARNING	This is macro AVC_BATTERY_STATUS_WARNING.
	AVC_CAPABILITY_COMPANY_ID	This is macro AVC_CAPABILITY_COMPANY_ID.
	AVC_CAPABILITY_EVENTS_SUPPORTED	This is macro AVC_CAPABILITY_EVENTS_SUPPORTED.
	AVC_CMD_GENERAL_POWER	This is macro AVC_CMD_GENERAL_POWER.
	AVC_CMD_GENERAL_RESERVE	This is macro AVC_CMD_GENERAL_RESERVE.
	AVC_CMD_GENERAL_SUBUNIT_INFO	This is macro AVC_CMD_GENERAL_SUBUNIT_INFO.
	AVC_CMD_GENERAL_UNIT_INFO	This is macro AVC_CMD_GENERAL_UNIT_INFO.
	AVC_CMD_GENERAL_VENDOR_DEPENDENT	This is macro AVC_CMD_GENERAL_VENDOR_DEPENDENT.
	AVC_CMD_GENERAL_VERSION	This is macro AVC_CMD_GENERAL_VERSION.
	AVC_CMD_PANEL_PASS_THROUGH	This is macro AVC_CMD_PANEL_PASS_THROUGH.
	AVC_CTYPE_CONTROL	• addtogroup avrcp • @{ • @name Command types
	AVC_CTYPE_GENERAL_INQUORY	This is macro AVC_CTYPE_GENERAL_INQUORY.
	AVC_CTYPE_NOTIFY	This is macro AVC_CTYPE_NOTIFY.
	AVC_CTYPE_SPECIFIC_IQUIRY	This is macro AVC_CTYPE_SPECIFIC_IQUIRY.
	AVC_CTYPE_STATUS	This is macro AVC_CTYPE_STATUS.
	AVC_EVENT_ADDRESSED_PLAYER_CHANGED	< The Addressed Player has been changed, see 6.9.2.
	AVC_EVENT_AVAILABLE_PLAYERS_CHANGED	< The available players have changed, see 6.9
	AVC_EVENT_BATT_STATUS_CHANGED	< Change in battery status
	AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED	This is macro AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED.
	AVC_EVENT_FLAG_ALL	This is macro AVC_EVENT_FLAG_ALL.
	AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED	This is macro AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED.
	AVC_EVENT_FLAG_BATT_STATUS_CHANGED	This is macro AVC_EVENT_FLAG_BATT_STATUS_CHANGED.
	AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED	This is macro AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED.
	AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED	This is macro AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED.
	AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED	• addtogroup avrcp • @{ • @name Notifications mask
	AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED	This is macro AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED.
	AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED	This is macro AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED.
	AVC_EVENT_FLAG_TRACK_CHANGED	This is macro AVC_EVENT_FLAG_TRACK_CHANGED.
	AVC_EVENT_FLAG_TRACK_REACHED_END	This is macro AVC_EVENT_FLAG_TRACK_REACHED_END.
	AVC_EVENT_FLAG_TRACK_REACHED_START	This is macro AVC_EVENT_FLAG_TRACK_REACHED_START.
	AVC_EVENT_FLAG_UIDS_CHANGED	This is macro AVC_EVENT_FLAG_UIDS_CHANGED.
	AVC_EVENT_FLAG_VOLUME_CHANGED	This is macro AVC_EVENT_FLAG_VOLUME_CHANGED.
	AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED	< The content of the Now Playing list has changed, see 6.9.5.

	AVC_EVENT_PLAYBACK_POS_CHANGED	< Change in playback position. Returned after the specified playback notification change notification interval
	AVC_EVENT_PLAYBACK_STATUS_CHANGED	< Change in playback status of the current track.
	AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED	< Change in player application setting
	AVC_EVENT_SYSTEM_STATUS_CHANGED	< Change in system status
	AVC_EVENT_TRACK_CHANGED	< Change of current track
	AVC_EVENT_TRACK_REACHED_END	< Reached end of a track
	AVC_EVENT_TRACK_REACHED_START	< Reached start of a track
	AVC_EVENT_UIDS_CHANGED	< The UIDs have changed, see 6.10.3.3.
	AVC_EVENT_VOLUME_CHANGED	< The volume has been changed locally on the TG, see 6.13.3
	AVC_FLAG_BROWSING_CMD	This is macro AVC_FLAG_BROWSING_CMD.
	AVC_FLAG_PANEL_CLICK	This is macro AVC_FLAG_PANEL_CLICK.
	AVC_FLAG_RESPONSE	This is macro AVC_FLAG_RESPONSE.
	AVC_MAXEVENTS	This is macro AVC_MAXEVENTS.
	AVC_MEDIA_ATTR_FLAG_ALBUM	This is macro AVC_MEDIA_ATTR_FLAG_ALBUM.
	AVC_MEDIA_ATTR_FLAG_ALL	This is macro AVC_MEDIA_ATTR_FLAG_ALL.
	AVC_MEDIA_ATTR_FLAG_ARTIST	This is macro AVC_MEDIA_ATTR_FLAG_ARTIST.
	AVC_MEDIA_ATTR_FLAG_GENRE	This is macro AVC_MEDIA_ATTR_FLAG_GENRE.
	AVC_MEDIA_ATTR_FLAG_NUMBER	This is macro AVC_MEDIA_ATTR_FLAG_NUMBER.
	AVC_MEDIA_ATTR_FLAG_PLAYING_TIME	This is macro AVC_MEDIA_ATTR_FLAG_PLAYING_TIME.
	AVC_MEDIA_ATTR_FLAG_TITLE	• addtogroup avrcp • @{ • @name Media attribute bitmask
	AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER	This is macro AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER.
	AVC_MEDIA_ATTR_ID_ALBUM	This is macro AVC_MEDIA_ATTR_ID_ALBUM.
	AVC_MEDIA_ATTR_ID_ARTIST	This is macro AVC_MEDIA_ATTR_ID_ARTIST.
	AVC_MEDIA_ATTR_ID_GENRE	This is macro AVC_MEDIA_ATTR_ID_GENRE.
	AVC_MEDIA_ATTR_ID_NUMBER	This is macro AVC_MEDIA_ATTR_ID_NUMBER.
	AVC_MEDIA_ATTR_ID_PLAYING_TIME	This is macro AVC_MEDIA_ATTR_ID_PLAYING_TIME.
	AVC_MEDIA_ATTR_ID_TITLE	• addtogroup avrcp • @{ • @name Media attribute IDs
	AVC_MEDIA_ATTR_ID_TOTAL_NUMBER	This is macro AVC_MEDIA_ATTR_ID_TOTAL_NUMBER.
	AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM	Folder Item, Media Element Item The virtual filesystem containing the Browsed media content of the browsed player
	AVC_NOW_PLAYING	Media Element Item The Now Playing list (or queue) Addressed of the addressed player
	AVC_PACKET_TYPE_CONTINUE	This is macro AVC_PACKET_TYPE_CONTINUE.
	AVC_PACKET_TYPE_END	This is macro AVC_PACKET_TYPE_END.
	AVC_PACKET_TYPE_SINGLE	This is macro AVC_PACKET_TYPE_SINGLE.
	AVC_PACKET_TYPE_START	This is macro AVC_PACKET_TYPE_START.
	AVC_PANEL_BUTTON_PRESSED	This is macro AVC_PANEL_BUTTON_PRESSED.
	AVC_PANEL_BUTTON_RELEASED	This is macro AVC_PANEL_BUTTON_RELEASED.

	AVC_PANEL_OPID_0	define AVC_PANEL_OPID_RESERVER 0x10 define AVC_PANEL_OPID_RESERVER 0x11 define AVC_PANEL_OPID_RESERVER 0x12 define AVC_PANEL_OPID_RESERVER 0x13 define AVC_PANEL_OPID_RESERVER 0x14 define AVC_PANEL_OPID_RESERVER 0x15 define AVC_PANEL_OPID_RESERVER 0x16 define AVC_PANEL_OPID_RESERVER 0x17 define AVC_PANEL_OPID_RESERVER 0x18 define AVC_PANEL_OPID_RESERVER 0x19 define AVC_PANEL_OPID_RESERVER 0x1A define AVC_PANEL_OPID_RESERVER 0x1B define AVC_PANEL_OPID_RESERVER 0x1C define AVC_PANEL_OPID_RESERVER 0x1D define AVC_PANEL_OPID_RESERVER 0x1E define AVC_PANEL_OPID_RESERVER 0x1F
	AVC_PANEL_OPID_1	This is macro AVC_PANEL_OPID_1.
	AVC_PANEL_OPID_2	This is macro AVC_PANEL_OPID_2.
	AVC_PANEL_OPID_3	This is macro AVC_PANEL_OPID_3.
	AVC_PANEL_OPID_4	This is macro AVC_PANEL_OPID_4.
	AVC_PANEL_OPID_5	This is macro AVC_PANEL_OPID_5.
	AVC_PANEL_OPID_6	This is macro AVC_PANEL_OPID_6.
	AVC_PANEL_OPID_7	This is macro AVC_PANEL_OPID_7.
	AVC_PANEL_OPID_8	This is macro AVC_PANEL_OPID_8.
	AVC_PANEL_OPID_9	This is macro AVC_PANEL_OPID_9.
	AVC_PANEL_OPID_A	This is macro AVC_PANEL_OPID_A.
	AVC_PANEL_OPID_ANGLE	define AVC_PANEL_OPID_RESERVED 0x4E define AVC_PANEL_OPID_RESERVED 0x4F
	AVC_PANEL_OPID_APPS_MENU	This is macro AVC_PANEL_OPID_APPS_MENU.
	AVC_PANEL_OPID_B	This is macro AVC_PANEL_OPID_B.
	AVC_PANEL_OPID_BACKWARD	This is macro AVC_PANEL_OPID_BACKWARD.
	AVC_PANEL_OPID_C	This is macro AVC_PANEL_OPID_C.
	AVC_PANEL_OPID_CHANNEL_DOWN	This is macro AVC_PANEL_OPID_CHANNEL_DOWN.
	AVC_PANEL_OPID_CHANNEL_UP	define AVC_PANEL_OPID_RESRVED 0x2D define AVC_PANEL_OPID_RESRVED 0x2E define AVC_PANEL_OPID_RESRVED 0x2F
	AVC_PANEL_OPID_CLEAR	This is macro AVC_PANEL_OPID_CLEAR.
	AVC_PANEL_OPID_CONTENTS_MENU	This is macro AVC_PANEL_OPID_CONTENTS_MENU.
	AVC_PANEL_OPID_D	This is macro AVC_PANEL_OPID_D.
	AVC_PANEL_OPID_DISPLAY_INFORMATION	This is macro AVC_PANEL_OPID_DISPLAY_INFORMATION.
	AVC_PANEL_OPID_DOT	This is macro AVC_PANEL_OPID_DOT.
	AVC_PANEL_OPID_DOWN	This is macro AVC_PANEL_OPID_DOWN.
	AVC_PANEL_OPID_EJECT	This is macro AVC_PANEL_OPID_EJECT.
	AVC_PANEL_OPID_ENTER	This is macro AVC_PANEL_OPID_ENTER.
	AVC_PANEL_OPID_EXIT	This is macro AVC_PANEL_OPID_EXIT.
	AVC_PANEL_OPID_F1	define AVC_PANEL_OPID_RESERVED 0x6D define AVC_PANEL_OPID_RESERVED 0x6E define AVC_PANEL_OPID_RESERVED 0x6F define AVC_PANEL_OPID_RESERVED 0x70
	AVC_PANEL_OPID_F2	This is macro AVC_PANEL_OPID_F2.
	AVC_PANEL_OPID_F3	This is macro AVC_PANEL_OPID_F3.
	AVC_PANEL_OPID_F4	This is macro AVC_PANEL_OPID_F4.
	AVC_PANEL_OPID_F5	This is macro AVC_PANEL_OPID_F5.
	AVC_PANEL_OPID_F6	This is macro AVC_PANEL_OPID_F6.
	AVC_PANEL_OPID_F7	This is macro AVC_PANEL_OPID_F7.
	AVC_PANEL_OPID_F8	This is macro AVC_PANEL_OPID_F8.
	AVC_PANEL_OPID_F9	This is macro AVC_PANEL_OPID_F9.

	AVC_PANEL_OPID_FAST_FORWARD	This is macro AVC_PANEL_OPID_FAST_FORWARD.
	AVC_PANEL_OPID_FAVORITE_MENU	This is macro AVC_PANEL_OPID_FAVORITE_MENU.
	AVC_PANEL_OPID_FORWARD	This is macro AVC_PANEL_OPID_FORWARD.
	AVC_PANEL_OPID_HELP	This is macro AVC_PANEL_OPID_HELP.
	AVC_PANEL_OPID_INPUT_SELECT	This is macro AVC_PANEL_OPID_INPUT_SELECT.
	AVC_PANEL_OPID_KEYBORD_FUNCTION	This is macro AVC_PANEL_OPID_KEYBORD_FUNCTION.
	AVC_PANEL_OPID_LEFT	This is macro AVC_PANEL_OPID_LEFT.
	AVC_PANEL_OPID_LEFT_DOWN	This is macro AVC_PANEL_OPID_LEFT_DOWN.
	AVC_PANEL_OPID_LEFT_UP	This is macro AVC_PANEL_OPID_LEFT_UP.
	AVC_PANEL_OPID_LINKED_CONTENT	This is macro AVC_PANEL_OPID_LINKED_CONTENT.
	AVC_PANEL_OPID_LIST	This is macro AVC_PANEL_OPID_LIST.
	AVC_PANEL_OPID_LIVE_TV	This is macro AVC_PANEL_OPID_LIVE_TV.
	AVC_PANEL_OPID_LOCK	This is macro AVC_PANEL_OPID_LOCK.
	AVC_PANEL_OPID_MUTE	This is macro AVC_PANEL_OPID_MUTE.
	AVC_PANEL_OPID_MUTE_FUNCTION	This is macro AVC_PANEL_OPID_MUTE_FUNCTION.
	AVC_PANEL_OPID_NEXT_DAY	This is macro AVC_PANEL_OPID_NEXT_DAY.
	AVC_PANEL_OPID_ON_DEMAND_MENU	This is macro AVC_PANEL_OPID_ON_DEMAND_MENU.
	AVC_PANEL_OPID_PAGE_DOWN	This is macro AVC_PANEL_OPID_PAGE_DOWN.
	AVC_PANEL_OPID_PAGE_UP	This is macro AVC_PANEL_OPID_PAGE_UP.
	AVC_PANEL_OPID_PAUSE	This is macro AVC_PANEL_OPID_PAUSE.
	AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION	This is macro AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION.
	AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION	This is macro AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION.
	AVC_PANEL_OPID_PIP_DOWN	This is macro AVC_PANEL_OPID_PIP_DOWN.
	AVC_PANEL_OPID_PIP_MOVE	This is macro AVC_PANEL_OPID_PIP_MOVE.
	AVC_PANEL_OPID_PIP_UP	This is macro AVC_PANEL_OPID_PIP_UP.
	AVC_PANEL_OPID_PLAY	This is macro AVC_PANEL_OPID_PLAY.
	AVC_PANEL_OPID_PLAY_FUNCTION	define AVC_PANEL_OPID_RESERVED 0x56 define AVC_PANEL_OPID_RESERVED 0x57 define AVC_PANEL_OPID_RESERVED 0x58 define AVC_PANEL_OPID_RESERVED 0x5A define AVC_PANEL_OPID_RESERVED 0x5B define AVC_PANEL_OPID_RESERVED 0x5C define AVC_PANEL_OPID_RESERVED 0x5D define AVC_PANEL_OPID_RESERVED 0x5E define AVC_PANEL_OPID_RESERVED 0x5F
	AVC_PANEL_OPID_POWER_STATE_FUNCTION	This is macro AVC_PANEL_OPID_POWER_STATE_FUNCTION.
	AVC_PANEL_OPID_POWER_TOGGLE	This is macro AVC_PANEL_OPID_POWER_TOGGLE.
	AVC_PANEL_OPID_PREVIOUS_CHANNEL	This is macro AVC_PANEL_OPID_PREVIOUS_CHANNEL.
	AVC_PANEL_OPID_PREVIOUS_DAY	This is macro AVC_PANEL_OPID_PREVIOUS_DAY.
	AVC_PANEL_OPID_RECORD	This is macro AVC_PANEL_OPID_RECORD.
	AVC_PANEL_OPID_RECORD_FUNCTION	This is macro AVC_PANEL_OPID_RECORD_FUNCTION.
	AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION	This is macro AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION.
	AVC_PANEL_OPID_REWIND	This is macro AVC_PANEL_OPID_REWIND.
	AVC_PANEL_OPID_RF_BYPASS	This is macro AVC_PANEL_OPID_RF_BYPASS.
	AVC_PANEL_OPID_RIGHT	This is macro AVC_PANEL_OPID_RIGHT.

	AVC_PANEL_OPID_RIGHT_DOWN	This is macro AVC_PANEL_OPID_RIGHT_DOWN.
	AVC_PANEL_OPID_RIGHT_UP	This is macro AVC_PANEL_OPID_RIGHT_UP.
	AVC_PANEL_OPID_ROOT_MENU	This is macro AVC_PANEL_OPID_ROOT_MENU.
	AVC_PANEL_OPID_SELECT	• addtogroup avrcp • @{ • @name AV/C Panel PASS THROUGH operation IDs
	AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION	This is macro AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION.
	AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION	This is macro AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION.
	AVC_PANEL_OPID_SELECT_DISK_FUNCTION	This is macro AVC_PANEL_OPID_SELECT_DISK_FUNCTION.
	AVC_PANEL_OPID_SETUP_MENU	This is macro AVC_PANEL_OPID_SETUP_MENU.
	AVC_PANEL_OPID_SKIP	This is macro AVC_PANEL_OPID_SKIP.
	AVC_PANEL_OPID_SOUND_SELECT	This is macro AVC_PANEL_OPID_SOUND_SELECT.
	AVC_PANEL_OPID_STOP	This is macro AVC_PANEL_OPID_STOP.
	AVC_PANEL_OPID_STOP_FUNCTION	This is macro AVC_PANEL_OPID_STOP_FUNCTION.
	AVC_PANEL_OPID_SUBPICTURE	This is macro AVC_PANEL_OPID_SUBPICTURE.
	AVC_PANEL_OPID_TUNE_FUNCTION	This is macro AVC_PANEL_OPID_TUNE_FUNCTION.
	AVC_PANEL_OPID_UP	This is macro AVC_PANEL_OPID_UP.
	AVC_PANEL_OPID_VENDOR_UNIQUE	This is macro AVC_PANEL_OPID_VENDOR_UNIQUE.
	AVC_PANEL_OPID_VOLUME_DOWN	This is macro AVC_PANEL_OPID_VOLUME_DOWN.
	AVC_PANEL_OPID_VOLUME_UP	This is macro AVC_PANEL_OPID_VOLUME_UP.
	AVC_PANEL_OPID_ZOOM	This is macro AVC_PANEL_OPID_ZOOM.
	AVC_PDUID_ABORT_CONTINUING_RESPONSE	This is macro AVC_PDUID_ABORT_CONTINUING_RESPONSE.
	AVC_PDUID_ADD_TO_NOW_PLAYING	This is macro AVC_PDUID_ADD_TO_NOW_PLAYING.
	AVC_PDUID_CHANGE_PATH	This is macro AVC_PDUID_CHANGE_PATH.
	AVC_PDUID_GENERAL_REJECT	This is macro AVC_PDUID_GENERAL_REJECT.
	AVC_PDUID_GET_CURRENT_PLAYER_APPLICATION_SETTING_VALUE	This is macro AVC_PDUID_GET_CURRENT_PLAYER_APPLICATION_SETTING_VALUE.
	AVC_PDUID_GET_ELEMENT_ATTRIBUTES	This is macro AVC_PDUID_GET_ELEMENT_ATTRIBUTES.
	AVC_PDUID_GET_FOLDER_ITEMS	This is macro AVC_PDUID_GET_FOLDER_ITEMS.
	AVC_PDUID_GET_ITEM_ATTRIBUTES	This is macro AVC_PDUID_GET_ITEM_ATTRIBUTES.
	AVC_PDUID_GET_PLAY_STATUS	This is macro AVC_PDUID_GET_PLAY_STATUS.
	AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_ATTRIBUTE_TEXT	This is macro AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_ATTRIBUTE_TEXT.
	AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_VALUE_TEXT	This is macro AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_VALUE_TEXT.
	AVC_PDUID_GETCAPABILITIES	This is macro AVC_PDUID_GETCAPABILITIES.
	AVC_PDUID_INFORM_BATTERY_STATUS_OF_CT	This is macro AVC_PDUID_INFORM_BATTERY_STATUS_OF_CT.
	AVC_PDUID_INFORM_DISPLAYABLE_CHARACTER_SET	This is macro AVC_PDUID_INFORM_DISPLAYABLE_CHARACTER_SET.

	AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_ATTRIBUTES	This is macro AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_ATTRIBUTES.
	AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_VALUES	This is macro AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_VALUES.
	AVC_PDUID_PLAY_ITEM	This is macro AVC_PDUID_PLAY_ITEM.
	AVC_PDUID_REGISTER_NOTIFICATION	This is macro AVC_PDUID_REGISTER_NOTIFICATION.
	AVC_PDUID_REQUEST_CONTINUING_RESPONSE	This is macro AVC_PDUID_REQUEST_CONTINUING_RESPONSE.
	AVC_PDUID_SEARCH	This is macro AVC_PDUID_SEARCH.
	AVC_PDUID_SET_ABSOLUTE_VOLUME	This is macro AVC_PDUID_SET_ABSOLUTE_VOLUME.
	AVC_PDUID_SET_ADDRESSED_PLAYER	This is macro AVC_PDUID_SET_ADDRESSED_PLAYER.
	AVC_PDUID_SET_BROWSED_PLAYER	This is macro AVC_PDUID_SET_BROWSED_PLAYER.
	AVC_PDUID_SET_PLAYER_APPLICATION_SETTING_VALUE	This is macro AVC_PDUID_SET_PLAYER_APPLICATION_SETTING_VALUE.
	AVC_PLAY_STATUS_ERROR	This is macro AVC_PLAY_STATUS_ERROR.
	AVC_PLAY_STATUS_FW_SEEK	This is macro AVC_PLAY_STATUS_FW_SEEK.
	AVC_PLAY_STATUS_PAUSED	This is macro AVC_PLAY_STATUS_PAUSED.
	AVC_PLAY_STATUS_PLAYING	This is macro AVC_PLAY_STATUS_PLAYING.
	AVC_PLAY_STATUS_REV_SEEK	This is macro AVC_PLAY_STATUS_REV_SEEK.
	AVC_PLAY_STATUS_STOPPED	• addtogroup avrcp • @{ • @name Play status
	AVC_PLAYER_SETTING_EQUALIZER_OFF	This is macro AVC_PLAYER_SETTING_EQUALIZER_OFF.
	AVC_PLAYER_SETTING_EQUALIZER_ON	This is macro AVC_PLAYER_SETTING_EQUALIZER_ON.
	AVC_PLAYER_SETTING_EQUALIZER_STATUS	This is macro AVC_PLAYER_SETTING_EQUALIZER_STATUS.
	AVC_PLAYER_SETTING_REPEAT_ALL_TRACKS	This is macro AVC_PLAYER_SETTING_REPEAT_ALL_TRACKS.
	AVC_PLAYER_SETTING_REPEAT_GROUP	This is macro AVC_PLAYER_SETTING_REPEAT_GROUP.
	AVC_PLAYER_SETTING_REPEAT_MODE_OFF	This is macro AVC_PLAYER_SETTING_REPEAT_MODE_OFF.
	AVC_PLAYER_SETTING_REPEAT_MODE_STATUS	This is macro AVC_PLAYER_SETTING_REPEAT_MODE_STATUS.
	AVC_PLAYER_SETTING_REPEAT_SINGLE_TRACK	This is macro AVC_PLAYER_SETTING_REPEAT_SINGLE_TRACK.
	AVC_PLAYER_SETTING_SCAN_ALL_TRACKS	This is macro AVC_PLAYER_SETTING_SCAN_ALL_TRACKS.
	AVC_PLAYER_SETTING_SCAN_GROUP	This is macro AVC_PLAYER_SETTING_SCAN_GROUP.
	AVC_PLAYER_SETTING_SCAN_OFF	This is macro AVC_PLAYER_SETTING_SCAN_OFF.
	AVC_PLAYER_SETTING_SCAN_STATUS	This is macro AVC_PLAYER_SETTING_SCAN_STATUS.
	AVC_PLAYER_SETTING_SHUFFLE_ALL_TRACKS	This is macro AVC_PLAYER_SETTING_SHUFFLE_ALL_TRACKS.
	AVC_PLAYER_SETTING_SHUFFLE_GROUP	This is macro AVC_PLAYER_SETTING_SHUFFLE_GROUP.
	AVC_PLAYER_SETTING_SHUFFLE_OFF	This is macro AVC_PLAYER_SETTING_SHUFFLE_OFF.

	AVC_PLAYER_SETTING_SHUFFLE_STATUS	This is macro AVC_PLAYER_SETTING_SHUFFLE_STATUS.
	AVC_RESPONSE_ACCEPTED	This is macro AVC_RESPONSE_ACCEPTED.
	AVC_RESPONSE_CHANGED	This is macro AVC_RESPONSE_CHANGED.
	AVC_RESPONSE_IMPLEMENTED	This is macro AVC_RESPONSE_IMPLEMENTED.
	AVC_RESPONSE_IN_TRANSITION	This is macro AVC_RESPONSE_IN_TRANSITION.
	AVC_RESPONSE_INTERIM	This is macro AVC_RESPONSE_INTERIM.
	AVC_RESPONSE_NOT_IMPLEMENTED	• addtogroup avrcp • @{ • @name Response types
	AVC_RESPONSE_REJECTED	This is macro AVC_RESPONSE_REJECTED.
	AVC_RESPONSE_STABLE	This is macro AVC_RESPONSE_STABLE.
	AVC_RESPONSE_TIMEOUT	This is macro AVC_RESPONSE_TIMEOUT.
	AVC_SCOPE_MEDIA_PLAYER_LIST	Media Player Item Contains all available media players None
	AVC_SEARCH	Media Element Item The results of a search operation Browsed on the browsed player
	AVC_SUBUNIT_ID_EXTENDED_TO_NEXT_BYTE	This is macro AVC_SUBUNIT_ID_EXTENDED_TO_NEXT_BYTE.
	AVC_SUBUNIT_ID_IGNORE	This is macro AVC_SUBUNIT_ID_IGNORE.
	AVC_SUBUNIT_TYPE_AUDIO	This is macro AVC_SUBUNIT_TYPE_AUDIO.
	AVC_SUBUNIT_TYPE_BULLETIN_BOARD	This is macro AVC_SUBUNIT_TYPE_BULLETIN_BOARD.
	AVC_SUBUNIT_TYPE_CA	This is macro AVC_SUBUNIT_TYPE_CA.
	AVC_SUBUNIT_TYPE_CAMERA	This is macro AVC_SUBUNIT_TYPE_CAMERA.
	AVC_SUBUNIT_TYPE_CAMERA_STORAGE	This is macro AVC_SUBUNIT_TYPE_CAMERA_STORAGE.
	AVC_SUBUNIT_TYPE_DISC	This is macro AVC_SUBUNIT_TYPE_DISC.
	AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE	This is macro AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE.
	AVC_SUBUNIT_TYPE_MONITOR	• addtogroup avrcp • @{ • @name Subunit types
	AVC_SUBUNIT_TYPE_PANEL	This is macro AVC_SUBUNIT_TYPE_PANEL.
	AVC_SUBUNIT_TYPE_PRINTER	This is macro AVC_SUBUNIT_TYPE_PRINTER.
	AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER	This is macro AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER.
	AVC_SUBUNIT_TYPE_TUNER	This is macro AVC_SUBUNIT_TYPE_TUNER.
	AVC_SUBUNIT_TYPE_UNIT	This is macro AVC_SUBUNIT_TYPE_UNIT.
	AVC_SUBUNIT_TYPE_VENDOR_UNIQUE	This is macro AVC_SUBUNIT_TYPE_VENDOR_UNIQUE.
	AVC_VOLUME_MAX	100
	AVC_VOLUME_MIN	0
	AVRCP_COMMAND_TYPE_BROWSING	This is macro AVRCP_COMMAND_TYPE_BROWSING.
	AVRCP_COMMAND_TYPE_CONTROL	This is macro AVRCP_COMMAND_TYPE_CONTROL.
	bt_avrcp_0_click	This is macro bt_avrcp_0_click.
	bt_avrcp_0_press	This is macro bt_avrcp_0_press.
	bt_avrcp_0_release	This is macro bt_avrcp_0_release.
	bt_avrcp_1_click	This is macro bt_avrcp_1_click.
	bt_avrcp_1_press	This is macro bt_avrcp_1_press.
	bt_avrcp_1_release	This is macro bt_avrcp_1_release.
	bt_avrcp_2_click	This is macro bt_avrcp_2_click.
	bt_avrcp_2_press	This is macro bt_avrcp_2_press.
	bt_avrcp_2_release	This is macro bt_avrcp_2_release.

	bt_avrcp_3_click	This is macro bt_avrcp_3_click.
	bt_avrcp_3_press	This is macro bt_avrcp_3_press.
	bt_avrcp_3_release	This is macro bt_avrcp_3_release.
	bt_avrcp_4_click	This is macro bt_avrcp_4_click.
	bt_avrcp_4_press	This is macro bt_avrcp_4_press.
	bt_avrcp_4_release	This is macro bt_avrcp_4_release.
	bt_avrcp_5_click	This is macro bt_avrcp_5_click.
	bt_avrcp_5_press	This is macro bt_avrcp_5_press.
	bt_avrcp_5_release	This is macro bt_avrcp_5_release.
	bt_avrcp_6_click	This is macro bt_avrcp_6_click.
	bt_avrcp_6_press	This is macro bt_avrcp_6_press.
	bt_avrcp_6_release	This is macro bt_avrcp_6_release.
	bt_avrcp_7_click	This is macro bt_avrcp_7_click.
	bt_avrcp_7_press	This is macro bt_avrcp_7_press.
	bt_avrcp_7_release	This is macro bt_avrcp_7_release.
	bt_avrcp_8_click	This is macro bt_avrcp_8_click.
	bt_avrcp_8_press	This is macro bt_avrcp_8_press.
	bt_avrcp_8_release	This is macro bt_avrcp_8_release.
	bt_avrcp_9_click	This is macro bt_avrcp_9_click.
	bt_avrcp_9_press	This is macro bt_avrcp_9_press.
	bt_avrcp_9_release	This is macro bt_avrcp_9_release.
	bt_avrcp_angle_click	This is macro bt_avrcp_angle_click.
	bt_avrcp_angle_press	This is macro bt_avrcp_angle_press.
	bt_avrcp_angle_release	This is macro bt_avrcp_angle_release.
	bt_avrcp_backward_click	This is macro bt_avrcp_backward_click.
	bt_avrcp_backward_press	This is macro bt_avrcp_backward_press.
	bt_avrcp_backward_release	This is macro bt_avrcp_backward_release.
	bt_avrcp_channel_down_click	This is macro bt_avrcp_channel_down_click.
	bt_avrcp_channel_down_press	This is macro bt_avrcp_channel_down_press.
	bt_avrcp_channel_down_release	This is macro bt_avrcp_channel_down_release.
	bt_avrcp_channel_up_click	This is macro bt_avrcp_channel_up_click.
	bt_avrcp_channel_up_press	This is macro bt_avrcp_channel_up_press.
	bt_avrcp_channel_up_release	This is macro bt_avrcp_channel_up_release.
	bt_avrcp_clear_click	This is macro bt_avrcp_clear_click.
	bt_avrcp_clear_press	This is macro bt_avrcp_clear_press.
	bt_avrcp_clear_release	This is macro bt_avrcp_clear_release.
	bt_avrcp_content_menu_click	This is macro bt_avrcp_content_menu_click.
	bt_avrcp_content_menu_release	This is macro bt_avrcp_content_menu_release.
	bt_avrcp_contents_menu_press	This is macro bt_avrcp_contents_menu_press.
	bt_avrcp_display_info_click	This is macro bt_avrcp_display_info_click.
	bt_avrcp_display_info_press	This is macro bt_avrcp_display_info_press.
	bt_avrcp_display_info_release	This is macro bt_avrcp_display_info_release.
	bt_avrcp_dot_click	This is macro bt_avrcp_dot_click.
	bt_avrcp_dot_press	This is macro bt_avrcp_dot_press.
	bt_avrcp_dot_release	This is macro bt_avrcp_dot_release.
	bt_avrcp_down_click	This is macro bt_avrcp_down_click.
	bt_avrcp_down_press	This is macro bt_avrcp_down_press.
	bt_avrcp_down_release	This is macro bt_avrcp_down_release.
	bt_avrcp_eject_click	This is macro bt_avrcp_eject_click.
	bt_avrcp_eject_press	This is macro bt_avrcp_eject_press.
	bt_avrcp_eject_release	This is macro bt_avrcp_eject_release.
	bt_avrcp_enter_click	This is macro bt_avrcp_enter_click.
	bt_avrcp_enter_press	This is macro bt_avrcp_enter_press.
	bt_avrcp_enter_release	This is macro bt_avrcp_enter_release.
	bt_avrcp_exit_click	This is macro bt_avrcp_exit_click.

	bt_avrcp_exit_press	This is macro bt_avrcp_exit_press.
	bt_avrcp_exit_release	This is macro bt_avrcp_exit_release.
	bt_avrcp_f1_click	This is macro bt_avrcp_f1_click.
	bt_avrcp_f1_press	This is macro bt_avrcp_f1_press.
	bt_avrcp_f1_release	This is macro bt_avrcp_f1_release.
	bt_avrcp_f2_click	This is macro bt_avrcp_f2_click.
	bt_avrcp_f2_press	This is macro bt_avrcp_f2_press.
	bt_avrcp_f2_release	This is macro bt_avrcp_f2_release.
	bt_avrcp_f3_click	This is macro bt_avrcp_f3_click.
	bt_avrcp_f3_press	This is macro bt_avrcp_f3_press.
	bt_avrcp_f3_release	This is macro bt_avrcp_f3_release.
	bt_avrcp_f4_click	This is macro bt_avrcp_f4_click.
	bt_avrcp_f4_press	This is macro bt_avrcp_f4_press.
	bt_avrcp_f4_release	This is macro bt_avrcp_f4_release.
	bt_avrcp_f5_click	This is macro bt_avrcp_f5_click.
	bt_avrcp_f5_press	This is macro bt_avrcp_f5_press.
	bt_avrcp_f5_release	This is macro bt_avrcp_f5_release.
	bt_avrcp_f6_click	This is macro bt_avrcp_f6_click.
	bt_avrcp_f6_press	This is macro bt_avrcp_f6_press.
	bt_avrcp_f6_release	This is macro bt_avrcp_f6_release.
	bt_avrcp_f7_click	This is macro bt_avrcp_f7_click.
	bt_avrcp_f7_press	This is macro bt_avrcp_f7_press.
	bt_avrcp_f7_release	This is macro bt_avrcp_f7_release.
	bt_avrcp_f8_click	This is macro bt_avrcp_f8_click.
	bt_avrcp_f8_press	This is macro bt_avrcp_f8_press.
	bt_avrcp_f8_release	This is macro bt_avrcp_f8_release.
	bt_avrcp_f9_click	This is macro bt_avrcp_f9_click.
	bt_avrcp_f9_press	This is macro bt_avrcp_f9_press.
	bt_avrcp_f9_release	This is macro bt_avrcp_f9_release.
	bt_avrcp_fast_forward_click	This is macro bt_avrcp_fast_forward_click.
	bt_avrcp_fast_forward_press	This is macro bt_avrcp_fast_forward_press.
	bt_avrcp_fast_forward_release	This is macro bt_avrcp_fast_forward_release.
	bt_avrcp_favorite_menu_click	This is macro bt_avrcp_favorite_menu_click.
	bt_avrcp_favorite_menu_release	This is macro bt_avrcp_favorite_menu_release.
	bt_avrcp_favorive_menu_press	This is macro bt_avrcp_favorive_menu_press.
	bt_avrcp_forward_click	This is macro bt_avrcp_forward_click.
	bt_avrcp_forward_press	This is macro bt_avrcp_forward_press.
	bt_avrcp_forward_release	This is macro bt_avrcp_forward_release.
	bt_avrcp_help_click	This is macro bt_avrcp_help_click.
	bt_avrcp_help_press	This is macro bt_avrcp_help_press.
	bt_avrcp_help_release	This is macro bt_avrcp_help_release.
	bt_avrcp_input_select_click	This is macro bt_avrcp_input_select_click.
	bt_avrcp_input_select_press	This is macro bt_avrcp_input_select_press.
	bt_avrcp_input_select_release	This is macro bt_avrcp_input_select_release.
	bt_avrcp_left_click	This is macro bt_avrcp_left_click.
	bt_avrcp_left_down_click	This is macro bt_avrcp_left_down_click.
	bt_avrcp_left_down_press	This is macro bt_avrcp_left_down_press.
	bt_avrcp_left_down_release	This is macro bt_avrcp_left_down_release.
	bt_avrcp_left_press	This is macro bt_avrcp_left_press.
	bt_avrcp_left_release	This is macro bt_avrcp_left_release.
	bt_avrcp_left_up_click	This is macro bt_avrcp_left_up_click.
	bt_avrcp_left_up_press	This is macro bt_avrcp_left_up_press.
	bt_avrcp_left_up_release	This is macro bt_avrcp_left_up_release.
	bt_avrcp_mute_click	This is macro bt_avrcp_mute_click.
	bt_avrcp_mute_press	This is macro bt_avrcp_mute_press.

	bt_avrcp_mute_release	This is macro bt_avrcp_mute_release.
	bt_avrcp_page_down_click	This is macro bt_avrcp_page_down_click.
	bt_avrcp_page_down_press	This is macro bt_avrcp_page_down_press.
	bt_avrcp_page_down_release	This is macro bt_avrcp_page_down_release.
	bt_avrcp_page_up_click	This is macro bt_avrcp_page_up_click.
	bt_avrcp_page_up_press	This is macro bt_avrcp_page_up_press.
	bt_avrcp_page_up_release	This is macro bt_avrcp_page_up_release.
	bt_avrcp_pause_click	This is macro bt_avrcp_pause_click.
	bt_avrcp_pause_press	This is macro bt_avrcp_pause_press.
	bt_avrcp_pause_release	This is macro bt_avrcp_pause_release.
	bt_avrcp_play_click	This is macro bt_avrcp_play_click.
	bt_avrcp_play_press	This is macro bt_avrcp_play_press.
	bt_avrcp_play_release	This is macro bt_avrcp_play_release.
	bt_avrcp_power_click	This is macro bt_avrcp_power_click.
	bt_avrcp_power_press	This is macro bt_avrcp_power_press.
	bt_avrcp_power_release	This is macro bt_avrcp_power_release.
	bt_avrcp_previous_channel_click	This is macro bt_avrcp_previous_channel_click.
	bt_avrcp_previous_channel_press	This is macro bt_avrcp_previous_channel_press.
	bt_avrcp_previous_channel_release	This is macro bt_avrcp_previous_channel_release.
	bt_avrcp_record_click	This is macro bt_avrcp_record_click.
	bt_avrcp_record_press	This is macro bt_avrcp_record_press.
	bt_avrcp_record_release	This is macro bt_avrcp_record_release.
	bt_avrcp_rewind_click	This is macro bt_avrcp_rewind_click.
	bt_avrcp_rewind_press	This is macro bt_avrcp_rewind_press.
	bt_avrcp_rewind_release	This is macro bt_avrcp_rewind_release.
	bt_avrcp_right_click	This is macro bt_avrcp_right_click.
	bt_avrcp_right_down_click	This is macro bt_avrcp_right_down_click.
	bt_avrcp_right_down_press	This is macro bt_avrcp_right_down_press.
	bt_avrcp_right_down_release	This is macro bt_avrcp_right_down_release.
	bt_avrcp_right_press	This is macro bt_avrcp_right_press.
	bt_avrcp_right_release	This is macro bt_avrcp_right_release.
	bt_avrcp_right_up_click	This is macro bt_avrcp_right_up_click.
	bt_avrcp_right_up_press	This is macro bt_avrcp_right_up_press.
	bt_avrcp_right_up_release	This is macro bt_avrcp_right_up_release.
	bt_avrcp_root_menu_click	This is macro bt_avrcp_root_menu_click.
	bt_avrcp_root_menu_press	This is macro bt_avrcp_root_menu_press.
	bt_avrcp_root_menu_release	This is macro bt_avrcp_root_menu_release.
	bt_avrcp_select_click	This is macro bt_avrcp_select_click.
	bt_avrcp_select_press	Panel operations
	bt_avrcp_select_release	This is macro bt_avrcp_select_release.
	bt_avrcp_setup_menu_click	This is macro bt_avrcp_setup_menu_click.
	bt_avrcp_setup_menu_press	This is macro bt_avrcp_setup_menu_press.
	bt_avrcp_setup_menu_release	This is macro bt_avrcp_setup_menu_release.
	bt_avrcp_sound_select_click	This is macro bt_avrcp_sound_select_click.
	bt_avrcp_sound_select_press	This is macro bt_avrcp_sound_select_press.
	bt_avrcp_sound_select_release	This is macro bt_avrcp_sound_select_release.
	bt_avrcp_stop_click	This is macro bt_avrcp_stop_click.
	bt_avrcp_stop_press	This is macro bt_avrcp_stop_press.
	bt_avrcp_stop_release	This is macro bt_avrcp_stop_release.
	bt_avrcp_subpicture_click	This is macro bt_avrcp_subpicture_click.
	bt_avrcp_subpicture_press	This is macro bt_avrcp_subpicture_press.
	bt_avrcp_subpicture_release	This is macro bt_avrcp_subpicture_release.
	bt_avrcp_up_click	This is macro bt_avrcp_up_click.
	bt_avrcp_up_press	This is macro bt_avrcp_up_press.
	bt_avrcp_up_release	This is macro bt_avrcp_up_release.

	bt_avrcp_volume_down_click	This is macro bt_avrcp_volume_down_click.
	bt_avrcp_volume_down_press	This is macro bt_avrcp_volume_down_press.
	bt_avrcp_volume_down_release	This is macro bt_avrcp_volume_down_release.
	bt_avrcp_volume_up_click	This is macro bt_avrcp_volume_up_click.
	bt_avrcp_volume_up_press	This is macro bt_avrcp_volume_up_press.
	bt_avrcp_volume_up_release	This is macro bt_avrcp_volume_up_release.

Structures

	Name	Description
	_bt_av_add_to_now_playing_s	brief Parameter to AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::add_to_now_playing_status - when a local device received a response to a "add to now playing" request.
	_bt_av_battery_status_of_ct_s	brief Parameter to AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::battery_status_of_ct - when a local device received a "battery status of controller" command.
	_bt_av_capability_company_id_s	brief Parameter to AVRCP_EVT_COMPANY_ID_LIST_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::company_id - when a local device received a response to a "get company id" request.
	_bt_av_capability_event_id_s	brief Parameter to AVRCP_EVT_EVENT_ID_LIST_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::supported_event_id - when a local device received a response to a "get supported events" request.
	_bt_av_command_t	This is record _bt_av_command_t.
	_bt_av_displayable_character_set_s	brief Parameter to AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::displayable_character_set - when a local device received a "displayable chracter set command" request.
	_bt_av_element_attribute_s	brief Media element attribute ingroup avrcp details This structure is used to store media element attribute.
	_bt_av_element_attributes_s	brief Parameter to AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::element_attributes - when a local device received a response to a "get media element attributes" request.
	_bt_av_element_id_s	brief Media element UID ingroup avrcp details This structure is used to store media element UID.
	_bt_av_get_element_attributes_s	brief Parameter to AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::get_element_attributes - when a local device received a "get element attributes" request.
	_bt_av_notification_addressed_player_changed_s	brief Parameter to AVRCP_EVT_ADDRESSED_PLAYER_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:addressed_player - when a local device received a "addressed player changed" notification.
	_bt_av_notification_app_setting_changed_s	This is type bt_av_notification_app_setting_changed_t.

	_bt_av_notification_battery_status_s	brief Parameter to AVRCP_EVT_BATT_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:battery_status - when a local device received a "battery status changed" notification.
	_bt_av_notification_playback_pos_changed_s	brief Parameter to AVRCP_EVT_PLAYBACK_POS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:playback_pos - when a local device received a "playback position changed" notification.
	_bt_av_notification_playback_status_changed_s	brief Parameter to AVRCP_EVT_PLAYBACK_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:play_status - when a local device received a "play status changed" notification.
	_bt_av_notification_s	brief Parameter to the following events: li AVRCP_EVT_PLAYBACK_STATUS_CHANGED li AVRCP_EVT_TRACK_CHANGED li AVRCP_EVT_PLAYBACK_POS_CHANGED li AVRCP_EVT_BATT_STATUS_CHANGED li AVRCP_EVT_SYSTEM_STATUS_CHANGED li AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED li AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED li AVRCP_EVT_ADDRESSED_PLAYER_CHANGED li AVRCP_EVT_UIDS_CHANGED li AVRCP_EVT_VOLUME_CHANGED ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification - when a local device received one of the following notifications from the target: li Play status changed li Track changed changed li Playback position changed li Battery status changed li System status changed li Addressed player changed li UIDs changed li Volume changed li Player application setting changed... more
	_bt_av_notification_system_status_changed_s	brief Parameter to AVRCP_EVT_SYSTEM_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:system_status - when a local device received a "system status changed" notification.
	_bt_av_notification_track_changed_s	brief Parameter to AVRCP_EVT_TRACK_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:track - when a local device received a "track changed" notification.
	_bt_av_notification_uids_changed_s	brief Parameter to AVRCP_EVT_UIDS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:uids - when a local device received a "UIDs changed" notification.
	_bt_av_notification_volume_changed_s	brief Parameter to AVRCP_EVT_VOLUME_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:volume - when a local device received a "UIDs changed" notification.
	_bt_av_play_item_s	brief Parameter to AVRCP_EVT_PLAY_ITEM_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_item_status - when a local device received a response to a "play item" request.
	_bt_av_play_status_s	brief Parameter to AVRCP_EVT_GET_PLAY_STATUS_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_status - when a local device received a response to a "get play status" request.

	_bt_av_player_setting_current_values_s	brief Parameter to AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_current_values - when a local device received a response to a "get current player setting attribute values" request.
	_bt_av_player_setting_values_s	brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values - when a local device received a response to a "get player setting attribute values" request.
	_bt_av_player_setting_values_text_s	brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values_text - when a local device received a response to a "get player setting attribute values displayable text" request.
	_bt_av_player_settings_s	brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings - when a local device received a response to a "get supported player setting attributes" request.
	_bt_av_player_settings_text_s	brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings_text - when a local device received a response to a "get player setting attributes displayable text" request.
	_bt_av_player_text_s	This is type bt_av_player_text_t.
	_bt_av_register_notification_t	brief Parameter to AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::register_notification - when a local device received a "register notification" request.
	_bt_av_response_t	brief AV/C response header ingroup avrcp details This structure is used to store fields present in every AV/C response.
	_bt_av_set_absolute_volume_s	brief Parameter to AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::absolute_volume - when a local device received a response to a "set absolute volume" request.
	_bt_av_set_addressed_player_s	brief Parameter to AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::addressed_player - when a local device received a response to a "set addressed player" request.
	bt_av_add_to_now_playing_t	brief Parameter to AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::add_to_now_playing_status - when a local device received a response to a "add to now playing" request.
	bt_av_battery_status_of_ct_t	brief Parameter to AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::battery_status_of_ct - when a local device received a "battery status of controller" command.

	bt_av_capability_company_id_t	brief Parameter to AVRCP_EVT_COMPANY_ID_LIST_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::company_id - when a local device received a response to a "get company id" request.
	bt_av_capability_event_id_t	brief Parameter to AVRCP_EVT_EVENT_ID_LIST_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::supported_event_id - when a local device received a response to a "get supported events" request.
	bt_av_displayable_character_set_t	brief Parameter to AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::displayable_character_set - when a local device received a "displayable chracter set command" request.
	bt_av_element_attribute_t	brief Media element attribute ingroup avrcp details This structure is used to store media element attribute.
	bt_av_element_attributes_t	brief Parameter to AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::element_attributes - when a local device received a response to a "get media element attributes" request.
	bt_av_element_id_t	brief Media element UID ingroup avrcp details This structure is used to store media element UID.
	bt_av_get_element_attributes_t	brief Parameter to AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::get_element_attributes - when a local device received a "get element attributes" request.
	bt_av_notification_addressed_player_changed_t	brief Parameter to AVRCP_EVT_ADDRESSED_PLAYER_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:addressed_player - when a local device received a "addressed player changed" notification.
	bt_av_notification_app_setting_changed_t	This is type bt_av_notification_app_setting_changed_t.
	bt_av_notification_battery_status_t	brief Parameter to AVRCP_EVT_BATT_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:battery_status - when a local device received a "battery status changed" notification.
	bt_av_notification_playback_pos_changed_t	brief Parameter to AVRCP_EVT_PLAYBACK_POS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:playback_pos - when a local device received a "playback position changed" notification.
	bt_av_notification_playback_status_changed_t	brief Parameter to AVRCP_EVT_PLAYBACK_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:play_status - when a local device received a "play status changed" notification.
	bt_av_notification_system_status_changed_t	brief Parameter to AVRCP_EVT_SYSTEM_STATUS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:system_status - when a local device received a "system status changed" notification.

	bt_av_notification_t	brief Parameter to the following events: li AVRCP_EVT_PLAYBACK_STATUS_CHANGED li AVRCP_EVT_TRACK_CHANGED li AVRCP_EVT_PLAYBACK_POS_CHANGED li AVRCP_EVT_BATT_STATUS_CHANGED li AVRCP_EVT_SYSTEM_STATUS_CHANGED li AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED li AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED li AVRCP_EVT_ADDRESSED_PLAYER_CHANGED li AVRCP_EVT_UIDS_CHANGED li AVRCP_EVT_VOLUME_CHANGED ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification - when a local device received one of the following notifications from the target: li Play status changed li Track changed changed li Playback position changed li Battery status changed li System status changed li Addressed player changed li UIDs changed li Volume changed li Player application setting changed... more
	bt_av_notification_track_changed_t	brief Parameter to AVRCP_EVT_TRACK_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:track - when a local device received a "track changed" notification.
	bt_av_notification_uids_changed_t	brief Parameter to AVRCP_EVT_UIDS_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:uids - when a local device received a "UIDs changed" notification.
	bt_av_notification_volume_changed_t	brief Parameter to AVRCP_EVT_VOLUME_CHANGED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:volume - when a local device received a "UIDs changed" notification.
	bt_av_play_item_t	brief Parameter to AVRCP_EVT_PLAY_ITEM_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_item_status - when a local device received a response to a "play item" request.
	bt_av_play_status_t	brief Parameter to AVRCP_EVT_GET_PLAY_STATUS_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_status - when a local device received a response to a "get play status" request.
	bt_av_player_setting_current_values_t	brief Parameter to AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_current_values - when a local device received a response to a "get current player setting attribute values" request.
	bt_av_player_setting_values_t	brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values - when a local device received a response to a "get player setting attribute values" request.
	bt_av_player_setting_values_text_t	brief Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values_text - when a local device received a response to a "get player setting attribute values displayable text" request.

	bt_av_player_settings_t	brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings - when a local device received a response to a "get supported player setting attributes" request.
	bt_av_player_settings_text_t	brief Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings_text - when a local device received a response to a "get player setting attributes displayable text" request.
	bt_av_player_text_t	This is type bt_av_player_text_t.
	bt_av_register_notification_t	brief Parameter to AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::register_notification - when a local device received a "register notification" request.
	bt_av_response_t	brief AV/C response header ingroup avrcp details This structure is used to store fields present in every AV/C response.
	bt_av_set_absolute_volume_t	brief Parameter to AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::absolute_volume - when a local device received a response to a "set absolute volume" request.
	bt_av_set_addressed_player_t	brief Parameter to AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED event ingroup avrcp details A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::addressed_player - when a local device received a response to a "set addressed player" request.

Types

	Name	Description
	bt_av_command_t	This is type bt_av_command_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avrcp_config.h

Macros

	Name	Description
	__AVRCP_CONFIG_H	This is macro __AVRCP_CONFIG_H.

	AVRCP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	• defgroup avrcp_config Configuration
		• ingroup avrcp
		*
		• This module describes parameters used to configure AVRCP layer.
		*
		• dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.
		* code #include "cdbt/bt/bt_std.h"
		// HCI, L2CAP and SDP must always be present
		// HCI configuration parameters #define HCI_MAX_CMD_BUFFERS ... #define HCI_MAX_DATA_BUFFERS ... #define HCI_MAX_HCI_CONNECTIONS ... #define HCI_RX_BUFFER_LEN ... #define HCI_TX_BUFFER_LEN ... #define HCI_L2CAP_BUFFER_LEN ... #define HCI_MAX_CMD_PARAM_LEN ...
		// L2CAP configuration parameters #define L2CAP_MAX_CMD_BUFFERS ... #define L2CAP_MAX_FRAME_BUFFERS ... #define L2CAP_MAX_PSMS ... #define L2CAP_MAX_CHANNELS ...
		// SDP... more

	AVRCP_ALLOCATE_BUFFERS_VARS	This is macro AVRCP_ALLOCATE_BUFFERS_VARS.
	AVRCP_CMD_TIMEOUT	brief Command timeout ingroup avrcp_config details This parameter defines the amount of time in milliseconds AVRCP waits for a response to a request. If not defined the default value of 10000 (10 secconds) is used.
	AVRCP_MAX_CHANNELS	brief Maximum number of remote devices a local device can be connected to ingroup avrcp_config details This parameter defines the number of remote devices a local device can have simultaneous connections to (i.e. channels). This value should not exceed AVCTP_MAX_CHANNELS.
	AVRCP_MAX_CMD_BUFFERS	brief Maximum number of command buffers. ingroup avrcp_config details This parameter defines the number of buffers reserved for sending commands to a remote device over its control channel. Each channel uses its own buffers so the total number of buffers is AVRCP_MAX_CHANNELS * AVRCP_MAX_CMD_BUFFERS. The minimum value is 1. The maximum value is 255. If not define one buffer for each channel is reserved.
	AVRCP_MAX_CMD_PARAM_LEN	brief Maximum length of command parameters ingroup avrcp_config details This parameter defines the maximum length of all command parameters. If not defined the default value of 512 is used.
	AVRCP_MAX_DEVICE_NAME_LEN	brief Maximum length of device name ingroup avrcp_config details This parameter defines the size of the buffer used to store device's name while searching for nearby targets with bt_avrcp_find_targets. If the name of the device is longer than AVRCP_MAX_DEVICE_NAME_LEN it is truncated to AVRCP_MAX_DEVICE_NAME_LEN. If not defined the default value of 20 is used.
	AVRCP_MAX_SEARCH_RESULTS	brief Maximum number of devices to find ingroup avrcp_config details This parameter defines the maximum number of devices bt_avrcp_find_targets can find. If not defined the default value of 7 is used.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

avrcp_config_event_handlers.h

Macros

	Name	Description
	__AVRCP_CONFIG_EVENT_HANDLERS_H	This is macro __AVRCP_CONFIG_EVENT_HANDLERS_H.
	AVRCP_COMMAND_HANDLER	This is macro AVRCP_COMMAND_HANDLER.
	AVRCP_COMMAND_SENT_HANDLER	This is macro AVRCP_COMMAND_SENT_HANDLER.
	AVRCP_RESPONSE_HANDLER	This is macro AVRCP_RESPONSE_HANDLER.
	AVRCP_RESPONSE_SENT_HANDLER	This is macro AVRCP_RESPONSE_SENT_HANDLER.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

avrcp_private.h

Functions

	Name	Description
	_bt_avrcp_allocate_browsing_cmd	This is function _bt_avrcp_allocate_browsing_cmd.
	_bt_avrcp_allocate_browsing_response	This is function _bt_avrcp_allocate_browsing_response.
	_bt_avrcp_allocate_bt_specific_cmd	This is function _bt_avrcp_allocate_bt_specific_cmd.
	_bt_avrcp_allocate_bt_specific_response	This is function _bt_avrcp_allocate_bt_specific_response.
	_bt_avrcp_allocate_channel	void _bt_avctp_l2cap_read_data_callback(struct _channel *pch, bt_byte* pdata, bt_int len); bt_avctp_channel_t* _bt_avctp_find_channel(bt_avctp_mgr_t* mgr, bt_bdaddr_t* remote_addr, bt_uint profile_id);
	_bt_avrcp_allocate_cmd	This is function _bt_avrcp_allocate_cmd.
	_bt_avrcp_allocate_response	This is function _bt_avrcp_allocate_response.
	_bt_avrcp_allocate_simple_panel_cmd	This is function _bt_avrcp_allocate_simple_panel_cmd.
	_bt_avrcp_allocate_simple_panel_response	This is function _bt_avrcp_allocate_simple_panel_response.
	_bt_avrcp_find_channel	This is function _bt_avrcp_find_channel.
	_bt_avrcp_free_channel	This is function _bt_avrcp_free_channel.
	_bt_avrcp_free_cmd	This is function _bt_avrcp_free_cmd.
	_bt_avrcp_general_reject	This is function _bt_avrcp_general_reject.
	_bt_avrcp_get_tick_count	This is function _bt_avrcp_get_tick_count.
	_bt_avrcp_handle_add_to_now_playing	This is function _bt_avrcp_handle_add_to_now_playing.
	_bt_avrcp_handle_command	This is function _bt_avrcp_handle_command.
	_bt_avrcp_handle_command_sent	This is function _bt_avrcp_handle_command_sent.
	_bt_avrcp_handle_get_capabilities	This is function _bt_avrcp_handle_get_capabilities.
	_bt_avrcp_handle_get_current_player_application_setting_value	This is function _bt_avrcp_handle_get_current_player_application_setting_value.
	_bt_avrcp_handle_get_element_attributes	This is function _bt_avrcp_handle_get_element_attributes.
	_bt_avrcp_handle_get_play_status	This is function _bt_avrcp_handle_get_play_status.
	_bt_avrcp_handle_get_player_application_setting_attribute_text	This is function _bt_avrcp_handle_get_player_application_setting_attribute_text.
	_bt_avrcp_handle_get_player_application_setting_value_text	This is function _bt_avrcp_handle_get_player_application_setting_value_text.
	_bt_avrcp_handle_inform_battery_status_of_ct	This is function _bt_avrcp_handle_inform_battery_status_of_ct.
	_bt_avrcp_handle_inform_displayable_character_set	This is function _bt_avrcp_handle_inform_displayable_character_set.
	_bt_avrcp_handle_list_player_application_setting_attributes	This is function _bt_avrcp_handle_list_player_application_setting_attributes.
	_bt_avrcp_handle_list_player_application_setting_values	This is function _bt_avrcp_handle_list_player_application_setting_values.
	_bt_avrcp_handle_play_item	This is function _bt_avrcp_handle_play_item.
	_bt_avrcp_handle_register_notification	This is function _bt_avrcp_handle_register_notification.
	_bt_avrcp_handle_request_continuing_response	This is function _bt_avrcp_handle_request_continuing_response.
	_bt_avrcp_handle_response	This is function _bt_avrcp_handle_response.
	_bt_avrcp_handle_response_sent	This is function _bt_avrcp_handle_response_sent.
	_bt_avrcp_handle_set_absolute_volume	This is function _bt_avrcp_handle_set_absolute_volume.
	_bt_avrcp_handle_set_addressed_player	This is function _bt_avrcp_handle_set_addressed_player.
	_bt_avrcp_handle_set_player_application_setting_value	This is function _bt_avrcp_handle_set_player_application_setting_value.
	_bt_avrcp_init_signal	void _bt_avctp_init_message_buffers();

	_bt_avrcp_init_timer	This is function _bt_avrcp_init_timer.
	_bt_avrcp_register_next_notification	This is function _bt_avrcp_register_next_notification.
	_bt_avrcp_register_pending_notification	This is function _bt_avrcp_register_pending_notification.
	_bt_avrcp_send_notifications	This is function _bt_avrcp_send_notifications.
	_bt_avrcp_send_rejected_response	This is function _bt_avrcp_send_rejected_response.
	_bt_avrcp_set_signal	This is function _bt_avrcp_set_signal.
	_bt_avrcp_start_timer	This is function _bt_avrcp_start_timer.
	_bt_avrcp_tg_handle_get_capabilities	This is function _bt_avrcp_tg_handle_get_capabilities.
	_bt_avrcp_tg_handle_get_element_attributes	This is function _bt_avrcp_tg_handle_get_element_attributes.
	_bt_avrcp_tg_handle_get_play_status	This is function _bt_avrcp_tg_handle_get_play_status.
	_bt_avrcp_tg_handle_inform_battery_status_of_ct	This is function _bt_avrcp_tg_handle_inform_battery_status_of_ct.
	_bt_avrcp_tg_handle_inform_displayable_character_set	This is function _bt_avrcp_tg_handle_inform_displayable_character_set.
	_bt_avrcp_tg_handle_register_notification	This is function _bt_avrcp_tg_handle_register_notification.
	_bt_avrcp_tg_handle_set_absolute_volume	This is function _bt_avrcp_tg_handle_set_absolute_volume.
	_bt_avrcp_write_command_header	This is function _bt_avrcp_write_command_header.
	bt_avrcp_abort_continuing_response	This is function bt_avrcp_abort_continuing_response.
	bt_avrcp_avrcpt_request_continuing_response	This is function bt_avrcp_avrcpt_request_continuing_response.

Macros

	Name	Description
	__AVRCP_PRIVATE_H	This is macro __AVRCP_PRIVATE_H.
	AVRCP_MAX_ELEMENT_ATTRIBUTES	This is macro AVRCP_MAX_ELEMENT_ATTRIBUTES.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

at_parser.h

Functions

	Name	Description
	bt_at_parse_fragment	This is function bt_at_parse_fragment.
	bt_at_parser_reset	This is function bt_at_parser_reset.

Macros

	Name	Description
	AT_EVT_CMD_CODE	This is macro AT_EVT_CMD_CODE.
	AT_EVT_CMD_COMPLETED	This is macro AT_EVT_CMD_COMPLETED.
	AT_EVT_CMD_PARAM	This is macro AT_EVT_CMD_PARAM.
	AT_EVT_CMD_READ_CODE	This is macro AT_EVT_CMD_READ_CODE.
	AT_EVT_ERROR	This is macro AT_EVT_ERROR.
	AT_EVT_OK	This is macro AT_EVT_OK.
	AT_EVT_RING	This is macro AT_EVT_RING.
	ATCMD_BUFFER_LEN	This is macro ATCMD_BUFFER_LEN.

Structures

	Name	Description
	_bt_at_parser_t	This is type bt_at_parser_t.
	bt_at_parser_t	This is type bt_at_parser_t.

Types

	Name	Description
	bt_at_parser_callback_pf	This is type bt_at_parser_callback_pf.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_bdaddr.h

Functions

	Name	Description
	bt_bdaddr_is_null	This is function bt_bdaddr_is_null.
	bt_bdaddrs_are_equal	This is function bt_bdaddrs_are_equal.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

baseband.h

Macros

	Name	Description
	COD_MAJOR_AUDIO	This is macro COD_MAJOR_AUDIO.
	COD_MAJOR_COMPUTER	This is macro COD_MAJOR_COMPUTER.
	COD_MAJOR_HEALTH	This is macro COD_MAJOR_HEALTH.
	COD_MAJOR_IMAGING	This is macro COD_MAJOR_IMAGING.
	COD_MAJOR_MISC	This is macro COD_MAJOR_MISC.
	COD_MAJOR_NET_ACCESS_POINT	This is macro COD_MAJOR_NET_ACCESS_POINT.
	COD_MAJOR_PERIPHERAL	This is macro COD_MAJOR_PERIPHERAL.
	COD_MAJOR_PHONE	This is macro COD_MAJOR_PHONE.
	COD_MAJOR_TOY	This is macro COD_MAJOR_TOY.
	COD_MAJOR_UNCATEGORIZED	This is macro COD_MAJOR_UNCATEGORIZED.
	COD_MAJOR_WEARABLE	This is macro COD_MAJOR_WEARABLE.
	COD_MINOR_AV_CAMCORDER	This is macro COD_MINOR_AV_CAMCORDER.
	COD_MINOR_AV_CAR_AUDIO	This is macro COD_MINOR_AV_CAR_AUDIO.
	COD_MINOR_AV_GAMING	This is macro COD_MINOR_AV_GAMING.
	COD_MINOR_AV_HANDSFREE	This is macro COD_MINOR_AV_HANDSFREE.
	COD_MINOR_AV_HEADPHONES	This is macro COD_MINOR_AV_HEADPHONES.
	COD_MINOR_AV_HEADSET	This is macro COD_MINOR_AV_HEADSET.
	COD_MINOR_AV_HIFI_AUDIO	This is macro COD_MINOR_AV_HIFI_AUDIO.
	COD_MINOR_AV_LOUDSPEAKER	This is macro COD_MINOR_AV_LOUDSPEAKER.
	COD_MINOR_AV_MICROPHONE	This is macro COD_MINOR_AV_MICROPHONE.
	COD_MINOR_AV_PORTABLE_AUDIO	This is macro COD_MINOR_AV_PORTABLE_AUDIO.
	COD_MINOR_AV_RESERVED	This is macro COD_MINOR_AV_RESERVED.
	COD_MINOR_AV_RESERVERD2	This is macro COD_MINOR_AV_RESERVERD2.
	COD_MINOR_AV_SET_TOP_BOX	This is macro COD_MINOR_AV_SET_TOP_BOX.
	COD_MINOR_AV_UNCATEGORIZED	This is macro COD_MINOR_AV_UNCATEGORIZED.
	COD_MINOR_AV_VCR	This is macro COD_MINOR_AV_VCR.
	COD_MINOR_AV_VIDE_DISPLAY_AND_LOUDSPEAKER	This is macro COD_MINOR_AV_VIDE_DISPLAY_AND_LOUDSPEAKER.
	COD_MINOR_AV_VIDEO_CAMERA	This is macro COD_MINOR_AV_VIDEO_CAMERA.
	COD_MINOR_AV_VIDEO_CONFERENCING	This is macro COD_MINOR_AV_VIDEO_CONFERENCING.
	COD_MINOR_AV_VIDEO_MONITOR	This is macro COD_MINOR_AV_VIDEO_MONITOR.
	COD_MINOR_COMPUTER_DESKTOP	This is macro COD_MINOR_COMPUTER_DESKTOP.
	COD_MINOR_COMPUTER_HANDHELD	This is macro COD_MINOR_COMPUTER_HANDHELD.

	COD_MINOR_COMPUTER_LAPTOP	This is macro COD_MINOR_COMPUTER_LAPTOP.
	COD_MINOR_COMPUTER_PALMSIZED	This is macro COD_MINOR_COMPUTER_PALMSIZED.
	COD_MINOR_COMPUTER_SERVER	This is macro COD_MINOR_COMPUTER_SERVER.
	COD_MINOR_COMPUTER_UNCATEGORIZED	This is macro COD_MINOR_COMPUTER_UNCATEGORIZED.
	COD_MINOR_COMPUTER_WEARABLE	This is macro COD_MINOR_COMPUTER_WEARABLE.
	COD_MINOR_HEALTH_BPM	This is macro COD_MINOR_HEALTH_BPM.
	COD_MINOR_HEALTH_DATA_DISPLAY	This is macro COD_MINOR_HEALTH_DATA_DISPLAY.
	COD_MINOR_HEALTH_GLUCOSE_METER	This is macro COD_MINOR_HEALTH_GLUCOSE_METER.
	COD_MINOR_HEALTH_HEART_MONITOR	This is macro COD_MINOR_HEALTH_HEART_MONITOR.
	COD_MINOR_HEALTH_PULSE_OXIMETER	This is macro COD_MINOR_HEALTH_PULSE_OXIMETER.
	COD_MINOR_HEALTH_THERMOMETER	This is macro COD_MINOR_HEALTH_THERMOMETER.
	COD_MINOR_HEALTH_UNCATEGORIZED	This is macro COD_MINOR_HEALTH_UNCATEGORIZED.
	COD_MINOR_HEALTH_WEIGHING_SCALE	This is macro COD_MINOR_HEALTH_WEIGHING_SCALE.
	COD_MINOR_IMAGING_CAMERA	This is macro COD_MINOR_IMAGING_CAMERA.
	COD_MINOR_IMAGING_DISPLAY	This is macro COD_MINOR_IMAGING_DISPLAY.
	COD_MINOR_IMAGING_PRINTER	This is macro COD_MINOR_IMAGING_PRINTER.
	COD_MINOR_IMAGING_SCANNER	This is macro COD_MINOR_IMAGING_SCANNER.
	COD_MINOR_IMAGING_UNCATEGORIZED	This is macro COD_MINOR_IMAGING_UNCATEGORIZED.
	COD_MINOR_LAN_01_17	This is macro COD_MINOR_LAN_01_17.
	COD_MINOR_LAN_17_33	This is macro COD_MINOR_LAN_17_33.
	COD_MINOR_LAN_33_50	This is macro COD_MINOR_LAN_33_50.
	COD_MINOR_LAN_50_67	This is macro COD_MINOR_LAN_50_67.
	COD_MINOR_LAN_67_83	This is macro COD_MINOR_LAN_67_83.
	COD_MINOR_LAN_83_99	This is macro COD_MINOR_LAN_83_99.
	COD_MINOR_LAN_FULLY_AVAILABLE	This is macro COD_MINOR_LAN_FULLY_AVAILABLE.
	COD_MINOR_LAN_NO_SERVICE	This is macro COD_MINOR_LAN_NO_SERVICE.
	COD_MINOR_LAN_UNCATEGORIZED	This is macro COD_MINOR_LAN_UNCATEGORIZED.
	COD_MINOR_PERIPHERAL_CARD_READER	This is macro COD_MINOR_PERIPHERAL_CARD_READER.
	COD_MINOR_PERIPHERAL_COMBO	This is macro COD_MINOR_PERIPHERAL_COMBO.
	COD_MINOR_PERIPHERAL_DIGITIZER	This is macro COD_MINOR_PERIPHERAL_DIGITIZER.
	COD_MINOR_PERIPHERAL_GAMEPAD	This is macro COD_MINOR_PERIPHERAL_GAMEPAD.
	COD_MINOR_PERIPHERAL_JOYSTICK	This is macro COD_MINOR_PERIPHERAL_JOYSTICK.
	COD_MINOR_PERIPHERAL_KEYBOARD	This is macro COD_MINOR_PERIPHERAL_KEYBOARD.
	COD_MINOR_PERIPHERAL_MOUSE	This is macro COD_MINOR_PERIPHERAL_MOUSE.
	COD_MINOR_PERIPHERAL_OTHER	This is macro COD_MINOR_PERIPHERAL_OTHER.
	COD_MINOR_PERIPHERAL_REMOTE	This is macro COD_MINOR_PERIPHERAL_REMOTE.
	COD_MINOR_PERIPHERAL_SENSING	This is macro COD_MINOR_PERIPHERAL_SENSING.
	COD_MINOR_PERIPHERAL_UNCATEGORIZED	This is macro COD_MINOR_PERIPHERAL_UNCATEGORIZED.
	COD_MINOR_PHONE_CELLULAR	This is macro COD_MINOR_PHONE_CELLULAR.
	COD_MINOR_PHONE_CORDLESS	This is macro COD_MINOR_PHONE_CORDLESS.
	COD_MINOR_PHONE_ISDN	This is macro COD_MINOR_PHONE_ISDN.
	COD_MINOR_PHONE_SMART	This is macro COD_MINOR_PHONE_SMART.
	COD_MINOR_PHONE_UNCATEGORIZED	This is macro COD_MINOR_PHONE_UNCATEGORIZED.
	COD_MINOR_PHONE_WIREDMODEM	This is macro COD_MINOR_PHONE_WIREDMODEM.
	COD_MINOR_TOY_CONTROLLER	This is macro COD_MINOR_TOY_CONTROLLER.
	COD_MINOR_TOY_DOLL	This is macro COD_MINOR_TOY_DOLL.
	COD_MINOR_TOY_GAME	This is macro COD_MINOR_TOY_GAME.
	COD_MINOR_TOY_ROBOT	This is macro COD_MINOR_TOY_ROBOT.
	COD_MINOR_TOY_UNCATEGORIZED	This is macro COD_MINOR_TOY_UNCATEGORIZED.
	COD_MINOR_TOY_VEHICLE	This is macro COD_MINOR_TOY_VEHICLE.
	COD_MINOR_WEARABLE_GLASSES	This is macro COD_MINOR_WEARABLE_GLASSES.
	COD_MINOR_WEARABLE_HELMET	This is macro COD_MINOR_WEARABLE_HELMET.
	COD_MINOR_WEARABLE_JACKET	This is macro COD_MINOR_WEARABLE_JACKET.
	COD_MINOR_WEARABLE_PAGER	This is macro COD_MINOR_WEARABLE_PAGER.
	COD_MINOR_WEARABLE_UNCATEGORIZED	This is macro COD_MINOR_WEARABLE_UNCATEGORIZED.

	COD_MINOR_WEARABLE_WATCH	This is macro COD_MINOR_WEARABLE_WATCH.
	COS_AUDIO	This is macro COS_AUDIO.
	COS_CAPTURING	This is macro COS_CAPTURING.
	COS_INFORMATION	This is macro COS_INFORMATION.
	COS_LIMITED_DISCOVERABLE_MODE	This is macro COS_LIMITED_DISCOVERABLE_MODE.
	COS_NETWORKING	This is macro COS_NETWORKING.
	COS_OBJECTTRANSFER	This is macro COS_OBJECTTRANSFER.
	COS_POSITIONING	This is macro COS_POSITIONING.
	COS_RENDERING	This is macro COS_RENDERING.
	COS_TELEPHONY	This is macro COS_TELEPHONY.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_config.h

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_hcitr.h

Functions

	Name	Description
	bt_oem_recv	• Receive data. This function is called by the HCI layer when it needs more data from the HCI controller. Implementation of this function must receive the specified number of bytes from the HCI controller and call the provided callback function. @param buffer Pointer to a buffer for the received data. The buffer must be long enough to accommodate the number of bytes specified by the par len parameter. @param len Number of bytes to receive. @param callback A callback function that must be called when the requested number of bytes have been received.
	bt_oem_send	• Send data. This function is called by the HCI layer when it needs to send data to the HCI controller. Implementation of this function must send the specified number of bytes to the HCI controller and call the provided callback function. @param buffer Pointer to the data to be sent . @param len Number of bytes to send. @param callback A callback function that must be called when all data have been sent.

Types

	Name	Description
	bt_oem_recv_callback_fp	• Receive callback. This callback function is called when a receive operation initiated by bt_oem_recv() has completed. @param len Number of received bytes. The value of this parameter should always be the same as the number of bytes requested in a call to bt_oem_recv().
	bt_oem_send_callback_fp	• Send callback. This callback function is called when a send operation initiated by bt_oem_send() has completed.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_log.h

Functions

	Name	Description
	bt_log_bdaddr	This is function bt_log_bdaddr.
	bt_log_int	This is function bt_log_int.
	bt_log_linkkey	This is function bt_log_linkkey.
	bt_log_memory	This is function bt_log_memory.
	bt_log_msg	This is function bt_log_msg.
	bt_oem_log_write	brief Output log message. ingroup log details DotStack calls this function to output its debug information. Implementation should output or store the specified message to whatever device or medium where it can be examined and analyzed.

Macros

	Name	Description
	BT_LOG	This is macro BT_LOG.
	BT_LOG_EX	This is macro BT_LOG_EX.
	BT_LOG_LEVEL_A2DP_PAKCET	This is macro BT_LOG_LEVEL_A2DP_PAKCET.
	BT_LOG_LEVEL_ALL	This is macro BT_LOG_LEVEL_ALL.
	BT_LOG_LEVEL_DEBUG	This is macro BT_LOG_LEVEL_DEBUG.
	BT_LOG_LEVEL_ERROR	This is macro BT_LOG_LEVEL_ERROR.
	BT_LOG_LEVEL_INFO	This is macro BT_LOG_LEVEL_INFO.
	BT_LOG_LEVEL_OFF	This is macro BT_LOG_LEVEL_OFF.
	BT_LOGADDR	This is macro BT_LOGADDR.
	BT_LOGADDR_EX	This is macro BT_LOGADDR_EX.
	BT_LOGINT	This is macro BT_LOGINT.
	BT_LOGINT_EX	This is macro BT_LOGINT_EX.
	BT_LOGLINKKEY	This is macro BT_LOGLINKKEY.
	BT_LOGLINKKEY_EX	This is macro BT_LOGLINKKEY_EX.
	BT_LOGMEMORY	This is macro BT_LOGMEMORY.
	BT_LOGMEMORY_EX	This is macro BT_LOGMEMORY_EX.
	BT_LOGWRITE	This is macro BT_LOGWRITE.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_oem.h

Functions

	Name	Description
	bt_oem_assert	This is function bt_oem_assert.
	bt_oem_get_device_class	This is function bt_oem_get_device_class.
	bt_oem_get_device_name	This is function bt_oem_get_device_name.

	bt_oem_get_pin_code	This is function bt_oem_get_pin_code.
	bt_oem_linkkey_notification	This is function bt_oem_linkkey_notification.
	bt_oem_linkkey_request	This is function bt_oem_linkkey_request.
	bt_oem_schedule_signals	This is function bt_oem_schedule_signals.
	bt_oem_ssp_callback	This is function bt_oem_ssp_callback.

Structures

	Name	Description
	_bt_linkkey_notification_t	This is type bt_linkkey_notification_t.
	_bt_linkkey_request_t	This is type bt_linkkey_request_t.
	bt_linkkey_notification_t	This is type bt_linkkey_notification_t.
	bt_linkkey_request_t	This is type bt_linkkey_request_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_oem_config.h

Functions

	Name	Description
	ATT_ALLOCATE_BUFFERS	This is function ATT_ALLOCATE_BUFFERS.
	ATT_CLIENT_ALLOCATE_BUFFERS	This is function ATT_CLIENT_ALLOCATE_BUFFERS.
	FTP_ALLOCATE_BUFFERS	This is function FTP_ALLOCATE_BUFFERS.
	GATT_CLIENT_ALLOCATE_BUFFERS	This is function GATT_CLIENT_ALLOCATE_BUFFERS.
	HCI_ALLOCATE_BUFFERS	This is function HCI_ALLOCATE_BUFFERS.
	HCITR_BCSP_ALLOCATE_BUFFERS	This is function HCITR_BCSP_ALLOCATE_BUFFERS.
	HCRP_ALLOCATE_BUFFERS	This is function HCRP_ALLOCATE_BUFFERS.
	HFP_ALLOCATE_BUFFERS	This is function HFP_ALLOCATE_BUFFERS.
	HID_ALLOCATE_BUFFERS	This is function HID_ALLOCATE_BUFFERS.
	HSP_AG_ALLOCATE_BUFFERS	This is function HSP_AG_ALLOCATE_BUFFERS.
	HSP_ALLOCATE_BUFFERS	This is function HSP_ALLOCATE_BUFFERS.
	IAP_ALLOCATE_BUFFERS	This is function IAP_ALLOCATE_BUFFERS.
	IAP_BT_ALLOCATE_BUFFERS	This is function IAP_BT_ALLOCATE_BUFFERS.
	IAP2_ALLOCATE_BUFFERS	This is function IAP2_ALLOCATE_BUFFERS.
	IAPEA_ALLOCATE_BUFFERS	This is function IAPEA_ALLOCATE_BUFFERS.
	L2CAP_ALLOCATE_BUFFERS	This is function L2CAP_ALLOCATE_BUFFERS.
	MAP_ALLOCATE_BUFFERS	This is function MAP_ALLOCATE_BUFFERS.
	OBEX_ALLOCATE_BUFFERS	This is function OBEX_ALLOCATE_BUFFERS.
	PBAP_ALLOCATE_BUFFERS	This is function PBAP_ALLOCATE_BUFFERS.
	RFCOMM_ALLOCATE_BUFFERS	This is function RFCOMM_ALLOCATE_BUFFERS.
	SDP_ALLOCATE_BUFFERS	This is function SDP_ALLOCATE_BUFFERS.
	SMP_ALLOCATE_BUFFERS	This is function SMP_ALLOCATE_BUFFERS.
	SPP_ALLOCATE_BUFFERS	This is function SPP_ALLOCATE_BUFFERS.

Macros

	Name	Description
	__BT_APP_CONFIG_H	This is macro __BT_APP_CONFIG_H.
	BT_ENABLE_BLE	This is macro BT_ENABLE_BLE.
	BT_INCLUDE_IAP	This is macro BT_INCLUDE_IAP.
	BT_INCLUDE_IAP2	This is macro BT_INCLUDE_IAP2.
	BT_INCLUDE_RFCOMM	This is macro BT_INCLUDE_RFCOMM.

	BT_LOG_LEVEL_MAX	This is macro BT_LOG_LEVEL_MAX.
	BT_LOG_LEVEL_MIN	This is macro BT_LOG_LEVEL_MIN.
	IAP_BT_MAX_TRANSPORTS	This is macro IAP_BT_MAX_TRANSPORTS.
	IAP_MAX_SESSIONS	This is macro IAP_MAX_SESSIONS.
	IAP_RX_BUFFER_SIZE	This is macro IAP_RX_BUFFER_SIZE.
	IAP2_MAX_PACKET_SIZE	This is macro IAP2_MAX_PACKET_SIZE.
	IAP2_MAX_SESSIONS	This is macro IAP2_MAX_SESSIONS.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_private.h

Macros

	Name	Description
	ARG_NOT_USED	This is macro ARG_NOT_USED.
	BT_ASSERT	This is macro BT_ASSERT.
	FALSE	This is macro FALSE.
	LOG	This is macro LOG.
	LOG_EX	This is macro LOG_EX.
	LOGADDR	This is macro LOGADDR.
	LOGADDR_EX	This is macro LOGADDR_EX.
	LOGCLEAR	This is macro LOGCLEAR.
	LOGINT	This is macro LOGINT.
	LOGINT_EX	This is macro LOGINT_EX.
	LOGMEMORY	This is macro LOGMEMORY.
	LOGMEMORY_EX	This is macro LOGMEMORY_EX.
	LOGWRITE	This is macro LOGWRITE.
	NULL	This is macro NULL.
	TRUE	This is macro TRUE.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_signal.h

Functions

	Name	Description
	bt_signal_init	This is function bt_signal_init.
	bt_signal_process_pending	This is function bt_signal_process_pending.
	bt_signal_register	This is function bt_signal_register.
	bt_signal_set	This is function bt_signal_set.
	bt_signal_unregister	This is function bt_signal_unregister.

Structures

	Name	Description
	_bt_signal_t	This is record _bt_signal_t.

Types

	Name	Description
	bt_signal_handler_fp	This is type bt_signal_handler_fp.
	bt_signal_t	This is type bt_signal_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_std.h

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_storage.h

Functions

	Name	Description
	bt_oem_storage_get_capacity	brief Get non-volatile storage capacity. ingroup stg details Implementation of this function must return the capacity of its non-volatile storage.
	bt_oem_storage_read	brief Read from the non-volatile storage. ingroup stg details This function is called by the stack to read from the non-volatile storage. This function must be implemented by the application. When this function is called the application must start a read operation. When the number of bytes specified by the c len parameter is read, the application must call the callback function specified by the c callback parameter. The application does not have to read the whole number of bytes during the call to this function. It may complete reading later and then call the completion callback. The stack guarantees... more
	bt_oem_storage_start	brief Begin a sequence of non-volatile storage operations. ingroup stg details DotStack calls this function when it starts a sequence of non-volatile storage operations. When the sequence is finished, DotStack will call bt_oem_storage_stop().
	bt_oem_storage_stop	brief End a sequence of non-volatile storage operations. ingroup stg details DotStack calls this function when it finishes executing a sequence of non-volatile storage operations.
	bt_oem_storage_write	brief Write to non-volatile storage. ingroup stg details This function is called by the stack to write data to the non-volatile storage. This function must be implemented by the application. When this function is called, the application must start writing specified data to the non-volatile storage. When all data has been written, the application must call the callback function passed in the c callback parameter. The application does not have to complete the write operation during the call to this function. It may complete the operation later and then call the callback function. In this case, the application does not... more

Macros

	Name	Description
	CDS_LAST_DEVICE_ADDR	This is macro CDS_LAST_DEVICE_ADDR.
	CDS_SIGNATURE	This is macro CDS_SIGNATURE.
	CDS_SIGNATURE_ADDR	This is macro CDS_SIGNATURE_ADDR.
	HCI_BDADDR_LEN	This is macro HCI_BDADDR_LEN.
	HIDS_LAST_DEVICE_ADDR	This is macro HIDS_LAST_DEVICE_ADDR.
	HIDS_SIGNATURE	This is macro HIDS_SIGNATURE.
	HIDS_SIGNATURE_ADDR	This is macro HIDS_SIGNATURE_ADDR.
	LKS_FIRST_KEY_ADDR	This is macro LKS_FIRST_KEY_ADDR.
	LKS_MAX_LINK_KEYS	This is macro LKS_MAX_LINK_KEYS.

	LKS_SIGNATURE	This is macro LKS_SIGNATURE.
	LKS_SIGNATURE_ADDR	This is macro LKS_SIGNATURE_ADDR.

Types

	Name	Description
	bt_storage_callback_fp	brief Storage callback. details This callback is called when a non-volatile storage operation completes.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_system.h

Functions

	Name	Description
	bt_sys_get_connectable	This is function bt_sys_get_connectable.
	bt_sys_get_discoverable	This is function bt_sys_get_discoverable.
	bt_sys_get_l2cap_manager	• Get the L2CAP manager. This function returns the L2CAP manager. The L2CAP manager is created as part of the start up sequence. @return The L2CAP manager.
	bt_sys_init	• Initialize the Bluetooth system. This function initializes all internal variables of HCI, L2CAP and SDP modules. It must be called by the application before it can access any functionality provided by the library. In addition to this initialization function the application must call initialization functions of all other profile modules the application is intended to use. E.g., if the application is using the SPP module the bt_spp_init() must be called right after calling bt_sys_init(). This function essentially calls bt_sys_init_ex(HCI_LINK_POLICY_ENABLE_ALL) so all link policy setting are enabled.
	bt_sys_init_ex	• Initialize the Bluetooth system. This function initializes all internal variables of HCI, L2CAP and SDP modules. It must be called by the application before it can access any functionality provided by the library. In addition to this initialization function the application must call initialization functions of all other profile modules the application is intended to use. E.g., if the application is using the SPP module the bt_spp_init() must be called right after calling bt_sys_init(). Also, the caller must provide an SDP database. @param default_link_policy default link policy settings. This is a bitmask that defines the initial value of... more
	bt_sys_set_modes	This is function bt_sys_set_modes.
	bt_sys_start	• Start the Bluetooth system. After all modules used by the application have been initialized this function should be called to start the Bluetooth system operation. During the start up sequence it will reset and initialize the HCI controller and then create the L2CAP manager. The application will be notified when the start up sequence completes by calling the provided callback function. Also, the caller must provide an SDP database. @param discoverable defines whether the device is discoverable after reset. @param connectable defines whether the device is connectable after reset. @param sdp_db SDP database data. @param sdp_db_len Length of SDP database... more

Types

	Name	Description
	bt_sys_callback_fp	• System start callback. This callback function is called when system start initiated by bt_sys_start() has completed. @param success Success of the operation: c BT_TRUE if successfull, c BT_FALSE otherwise. @param param Callback parameter that was specified when bt_sys_start() was called.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_timer.h

Enumerations

	Name	Description
	_bt_timer_id_enum	This is type bt_timer_id.
	bt_timer_id	This is type bt_timer_id.

Functions

	Name	Description
	bt_oem_timer_clear	brief Clear timer. details This function must be implemented by the application. When this function is called the application must clear the specified timer. If it is already expired and a callback is currently pending, the application should also take measures to cancel the callback. param timerId ID of the timer to clear.
	bt_oem_timer_set	brief Set timer. details This function must be implemented by the application. When it is called, the application must set the specified timer. When the timer expires, the application must call the passed callback function. The function must not wait until the timer expires. It must set the timer and exit immediately. param timerId ID of the timer to set. param milliseconds Timer interval in milliseconds param callback Timer expiration callback function.

Types

	Name	Description
	bt_timer_callback_fp	brief Timer callback. details This callback is called when a timer expires.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bt_types.h

Macros

	Name	Description
	BT_FALSE	This is macro BT_FALSE.
	BT_LINKKEY_LENGTH	This is macro BT_LINKKEY_LENGTH.
	BT_MAKE_BDADDR	This is macro BT_MAKE_BDADDR.
	BT_MAKE_BDADDR_LE	This is macro BT_MAKE_BDADDR_LE.
	BT_NO	This is macro BT_NO.
	BT_TRUE	This is macro BT_TRUE.
	BT_YES	This is macro BT_YES.

Structures

	Name	Description
	_bt_bdaddr_s	This is type bt_bdaddr_t.
	_bt_linkkey_t	This is type bt_linkkey_t.

	_bt_uuid_s	This is type bt_uuid_t.
	bt_bdaddr_t	This is type bt_bdaddr_t.
	bt_linkkey_t	This is type bt_linkkey_t.
	bt_uuid_t	This is type bt_uuid_t.

Types

	Name	Description
	bt_bdaddr_cp	This is type bt_bdaddr_cp.
	bt_bdaddr_p	This is type bt_bdaddr_p.
	bt_bool	This is type bt_bool.
	bt_byte_cp	This is type bt_byte_cp.
	bt_byte_p	This is type bt_byte_p.
	bt_char	This is type bt_char.
	bt_char_cp	This is type bt_char_cp.
	bt_char_p	This is type bt_char_p.
	bt_id	This is type bt_id.
	bt_int_cp	This is type bt_int_cp.
	bt_int_p	This is type bt_int_p.
	bt_linkkey_cp	This is type bt_linkkey_cp.
	bt_linkkey_p	This is type bt_linkkey_p.
	bt_long_cp	This is type bt_long_cp.
	bt_long_p	This is type bt_long_p.
	bt_uint_cp	This is type bt_uint_cp.
	bt_uint_p	This is type bt_uint_p.
	bt_ulong_cp	This is type bt_ulong_cp.
	bt_ulong_p	This is type bt_ulong_p.
	bt_uuid_cp	This is type bt_uuid_cp.
	bt_uuid_p	This is type bt_uuid_p.
	bt_uuid16	This is type bt_uuid16.
	bt_uuid32	This is type bt_uuid32.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

buffer.h

Functions

	Name	Description
	bt_alloc_buffer	This is function bt_alloc_buffer.
	bt_free_buffer	This is function bt_free_buffer.
	bt_get_buffer	This is function bt_get_buffer.
	bt_get_buffer_header	This is function bt_get_buffer_header.
	bt_get_buffer_index	This is function bt_get_buffer_index.
	bt_init_buffer_mgr	This is function bt_init_buffer_mgr.

Macros

	Name	Description
	BUFFER_HDR_LEN	This is macro BUFFER_HDR_LEN.
	BUFFER_STATE_FREE	for offsetof
	BUFFER_STATE_USED	This is macro BUFFER_STATE_USED.

Structures

	Name	Description
	_bt_buffer_header_t	This is type bt_buffer_header_t.
	_bt_buffer_mgr_t	This is type bt_buffer_mgr_t.
	bt_bt_buffer_header_p	This is type bt_bt_buffer_header_p.
	bt_buffer_header_t	This is type bt_buffer_header_t.
	bt_buffer_mgr_p	This is type bt_buffer_mgr_p.
	bt_buffer_mgr_t	This is type bt_buffer_mgr_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

bufferutils.h

Functions

	Name	Description
	_bt_memcpy	This is function _bt_memcpy.
	_compact_buffer	This is function _compact_buffer.
	_expand_buffer	This is function _expand_buffer.
	_is_empty_str	This is function _is_empty_str.
	_read_bdaddr	This is function _read_bdaddr.
	_readb	This is function _readb.
	_readi	This is function _readi.
	_readin	This is function _readin.
	_readl	This is function _readl.
	_readln	This is function _readln.
	_readui	This is function _readui.
	_readuin	This is function _readuin.
	_readul	This is function _readul.
	_readuln	This is function _readuln.
	_str2ulong	This is function _str2ulong.
	_str2ulong_dec	This is function _str2ulong_dec.
	_to_lower_case	This is function _to_lower_case.
	_ulong2str	This is function _ulong2str.
	_ulong2str_dec	This is function _ulong2str_dec.
	_write_bdaddr	This is function _write_bdaddr.
	_writeb	This is function _writeb.
	_writei	This is function _writei.
	_writein	This is function _writein.
	_writel	This is function _writel.
	_writeln	This is function _writeln.
	_writes	This is function _writes.
	_writesx	This is function _writesx.
	_zero_memory	This is function _zero_memory.

Macros

	Name	Description
	_readbn	This is macro _readbn.
	_readui	This is macro _readui.
	_readuin	This is macro _readuin.
	_readul	This is macro _readul.

	_readuln	This is macro _readuln.
	_writebn	This is macro _writebn.
	bt_max	This is macro bt_max.
	bt_min	This is macro bt_min.
	max	This is macro max.
	min	This is macro min.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

channel.h

Functions

	Name	Description
	bt_l2cap_read_data	This is function bt_l2cap_read_data.
	bt_l2cap_send_cmd	This is function bt_l2cap_send_cmd.
	bt_l2cap_send_data	brief Send data over an L2CAP channel ingroup l2cap details This function sends data over the specified L2CAP channel. param channel The L2CAP channel to send data over. param data The pointer to the data. param len The length of the data. param callback The callback function that is called when sending the data has been completed. return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.
	bt_l2cap_send_smart_data	This is function bt_l2cap_send_smart_data.

Macros

	Name	Description
	CID_ATT	This is macro CID_ATT.
	CID_LE_SIG	This is macro CID_LE_SIG.
	CID_MAX	This is macro CID_MAX.
	CID_MAX_FIXED	This is macro CID_MAX_FIXED.
	CID_NULL	This is macro CID_NULL.
	CID_RECV	This is macro CID_RECV.
	CID_SIG	This is macro CID_SIG.
	CID_SM	This is macro CID_SM.
	CMODE_BASIC	This is macro CMODE_BASIC.
	CMODE_ERETR	This is macro CMODE_ERETR.
	CMODE_FLOW	This is macro CMODE_FLOW.
	CMODE_RETR	This is macro CMODE_RETR.
	CMODE_STRM	This is macro CMODE_STRM.
	CSTATE_CLOSED	This is macro CSTATE_CLOSED.
	CSTATE_FREE	This is macro CSTATE_FREE.
	CSTATE_OPEN	This is macro CSTATE_OPEN.
	CSTATE_WAIT_CONFIG	This is macro CSTATE_WAIT_CONFIG.
	CSTATE_WAIT_CONFIG_REQ	This is macro CSTATE_WAIT_CONFIG_REQ.
	CSTATE_WAIT_CONFIG_RSP	This is macro CSTATE_WAIT_CONFIG_RSP.
	CSTATE_WAIT_CONNECT	This is macro CSTATE_WAIT_CONNECT.
	CSTATE_WAIT_CONNECT_RSP	This is macro CSTATE_WAIT_CONNECT_RSP.
	CSTATE_WAIT_DISCONNECT	This is macro CSTATE_WAIT_DISCONNECT.
	CTYPE_CL	connectionless bt_l2cap_channel
	CTYPE_CO	connection-oriented bt_l2cap_channel
	L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT	This is macro L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT.

	L2CAP_CHANNEL_FLAG_INCOMING	This is macro L2CAP_CHANNEL_FLAG_INCOMING.
	L2CAP_CHANNEL_FLAG_SENDING	This is macro L2CAP_CHANNEL_FLAG_SENDING.
	L2CAP_ERETR_RECV_STATE_RECV	This is macro L2CAP_ERETR_RECV_STATE_RECV.
	L2CAP_ERETR_RECV_STATE_REJ_SENT	This is macro L2CAP_ERETR_RECV_STATE_REJ_SENT.
	L2CAP_ERETR_RECV_STATE_SREJ_SENT	This is macro L2CAP_ERETR_RECV_STATE_SREJ_SENT.
	L2CAP_ERETR_XMIT_STATE_WAIT_ACK	This is macro L2CAP_ERETR_XMIT_STATE_WAIT_ACK.
	L2CAP_ERETR_XMIT_STATE_WAIT_F	This is macro L2CAP_ERETR_XMIT_STATE_WAIT_F.
	L2CAP_ERETR_XMIT_STATE_XMIT	This is macro L2CAP_ERETR_XMIT_STATE_XMIT.

Structures

	Name	Description
	_bt_l2cap_channel_ext_t	This is record _bt_l2cap_channel_ext_t.
	_bt_l2cap_channel_t	This is record _bt_l2cap_channel_t.
	_bt_l2cap_frame_desc_s	typedef union _bt_l2cap_frame_control_s { struct { bt_byte frame_type:1; bt_byte txSeq:6; bt_byte f:1; bt_byte reqSeq:6; bt_byte sar:2; } iframe; struct { bt_byte frame_type:1; bt_byte reserved:1; bt_byte s:2; bt_byte p:1; bt_byte reserved2:2; bt_byte f:1; bt_byte reqSeq:6; bt_byte reserved3:2; } sframe; } bt_l2cap_frame_control_t;
	bt_l2cap_frame_desc_t	typedef union _bt_l2cap_frame_control_s { struct { bt_byte frame_type:1; bt_byte txSeq:6; bt_byte f:1; bt_byte reqSeq:6; bt_byte sar:2; } iframe; struct { bt_byte frame_type:1; bt_byte reserved:1; bt_byte s:2; bt_byte p:1; bt_byte reserved2:2; bt_byte f:1; bt_byte reqSeq:6; bt_byte reserved3:2; } sframe; } bt_l2cap_frame_control_t;

Types

	Name	Description
	bt_l2cap_channel_ext_t	This is type bt_l2cap_channel_ext_t.
	bt_l2cap_channel_t	This is type bt_l2cap_channel_t.
	bt_l2cap_read_data_callback_fp	This is type bt_l2cap_read_data_callback_fp.
	bt_l2cap_send_data_callback_fp	This is type bt_l2cap_send_data_callback_fp.
	bt_l2cap_state_changed_callback_fp	This is type bt_l2cap_state_changed_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

chmanager.h

Functions

	Name	Description
	bt_l2cap_allocate_channel	This is function bt_l2cap_allocate_channel.
	bt_l2cap_free_channel	This is function bt_l2cap_free_channel.
	bt_l2cap_get_channel	This is function bt_l2cap_get_channel.
	bt_l2cap_get_channel_by_bdaddr_cid	This is function bt_l2cap_get_channel_by_bdaddr_cid.
	bt_l2cap_get_channel_by_hconn_cid	This is function bt_l2cap_get_channel_by_hconn_cid.
	bt_l2cap_get_channel_by_hconn_dest_cid	This is function bt_l2cap_get_channel_by_hconn_dest_cid.
	bt_l2cap_get_channel_by_psm	This is function bt_l2cap_get_channel_by_psm.
	bt_l2cap_init_channels	This is function bt_l2cap_init_channels.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

cmdbuffer.h

Functions

	Name	Description
	bt_l2cap_alloc_cmd_buffer	This is function bt_l2cap_alloc_cmd_buffer.
	bt_l2cap_alloc_cmd_config_req	This is function bt_l2cap_alloc_cmd_config_req.
	bt_l2cap_alloc_cmd_config_res	This is function bt_l2cap_alloc_cmd_config_res.
	bt_l2cap_alloc_cmd_conn_param_update_req	This is function bt_l2cap_alloc_cmd_conn_param_update_req.
	bt_l2cap_alloc_cmd_conn_param_update_res	This is function bt_l2cap_alloc_cmd_conn_param_update_res.
	bt_l2cap_alloc_cmd_connection_req	This is function bt_l2cap_alloc_cmd_connection_req.
	bt_l2cap_alloc_cmd_connection_res	This is function bt_l2cap_alloc_cmd_connection_res.
	bt_l2cap_alloc_cmd_disconnection_req	This is function bt_l2cap_alloc_cmd_disconnection_req.
	bt_l2cap_alloc_cmd_disconnection_res	This is function bt_l2cap_alloc_cmd_disconnection_res.
	bt_l2cap_alloc_cmd_echo_req	This is function bt_l2cap_alloc_cmd_echo_req.
	bt_l2cap_alloc_cmd_echo_res	This is function bt_l2cap_alloc_cmd_echo_res.
	bt_l2cap_alloc_cmd_info_req	This is function bt_l2cap_alloc_cmd_info_req.
	bt_l2cap_alloc_cmd_info_res	This is function bt_l2cap_alloc_cmd_info_res.
	bt_l2cap_alloc_cmd_reject	This is function bt_l2cap_alloc_cmd_reject.
	bt_l2cap_free_cmd_buffer	This is function bt_l2cap_free_cmd_buffer.
	bt_l2cap_init_cmd_buffers	This is function bt_l2cap_init_cmd_buffers.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

command.h

Macros

	Name	Description
	L2CAP_CMD_CONFIG_REQUEST	This is macro L2CAP_CMD_CONFIG_REQUEST.
	L2CAP_CMD_CONFIG_RESPONSE	This is macro L2CAP_CMD_CONFIG_RESPONSE.
	L2CAP_CMD_CONN_PARAM_UPDATE_REQUEST	This is macro L2CAP_CMD_CONN_PARAM_UPDATE_REQUEST.
	L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE	This is macro L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE.
	L2CAP_CMD_CONN_REQUEST	This is macro L2CAP_CMD_CONN_REQUEST.
	L2CAP_CMD_CONN_RESPONSE	This is macro L2CAP_CMD_CONN_RESPONSE.
	L2CAP_CMD_DATA_LEN_CMD_REJECT	This is macro L2CAP_CMD_DATA_LEN_CMD_REJECT.
	L2CAP_CMD_DATA_LEN_CONFIG_REQUEST	This is macro L2CAP_CMD_DATA_LEN_CONFIG_REQUEST.
	L2CAP_CMD_DATA_LEN_CONFIG_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_CONFIG_RESPONSE.
	L2CAP_CMD_DATA_LEN_CONN_REQUEST	This is macro L2CAP_CMD_DATA_LEN_CONN_REQUEST.
	L2CAP_CMD_DATA_LEN_CONN_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_CONN_RESPONSE.
	L2CAP_CMD_DATA_LEN_DCONN_REQUEST	This is macro L2CAP_CMD_DATA_LEN_DCONN_REQUEST.
	L2CAP_CMD_DATA_LEN_DCONN_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_DCONN_RESPONSE.
	L2CAP_CMD_DATA_LEN_ECHO_REQUEST	This is macro L2CAP_CMD_DATA_LEN_ECHO_REQUEST.
	L2CAP_CMD_DATA_LEN_ECHO_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_ECHO_RESPONSE.
	L2CAP_CMD_DATA_LEN_INFO_REQUEST	This is macro L2CAP_CMD_DATA_LEN_INFO_REQUEST.

	L2CAP_CMD_DATA_LEN_INFO_RESPONSE	This is macro L2CAP_CMD_DATA_LEN_INFO_RESPONSE.
	L2CAP_CMD_DCONN_REQUEST	This is macro L2CAP_CMD_DCONN_REQUEST.
	L2CAP_CMD_DCONN_RESPONSE	This is macro L2CAP_CMD_DCONN_RESPONSE.
	L2CAP_CMD_ECHO_REQUEST	This is macro L2CAP_CMD_ECHO_REQUEST.
	L2CAP_CMD_ECHO_RESPONSE	This is macro L2CAP_CMD_ECHO_RESPONSE.
	L2CAP_CMD_HEADER_LEN	This is macro L2CAP_CMD_HEADER_LEN.
	L2CAP_CMD_INFO_REQUEST	This is macro L2CAP_CMD_INFO_REQUEST.
	L2CAP_CMD_INFO_RESPONSE	This is macro L2CAP_CMD_INFO_RESPONSE.
	L2CAP_CMD_LAST	This is macro L2CAP_CMD_LAST.
	L2CAP_CMD_REJECT	This is macro L2CAP_CMD_REJECT.
	L2CAP_CMD_RESERVED	This is macro L2CAP_CMD_RESERVED.
	L2CAP_CMD_STATUS_BEING_SENT	This is macro L2CAP_CMD_STATUS_BEING_SENT.
	L2CAP_CMD_STATUS_PENDING	This is macro L2CAP_CMD_STATUS_PENDING.
	L2CAP_CMD_STATUS_WAITING_RESPONSE	This is macro L2CAP_CMD_STATUS_WAITING_RESPONSE.
	L2CAP_CONFIG_RESULT_REJECTED	This is macro L2CAP_CONFIG_RESULT_REJECTED.
	L2CAP_CONFIG_RESULT_SUCCESS	This is macro L2CAP_CONFIG_RESULT_SUCCESS.
	L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER	This is macro L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER.
	L2CAP_CONFIG_RESULT_UNKNOWN_OPTION	This is macro L2CAP_CONFIG_RESULT_UNKNOWN_OPTION.
	L2CAP_CONN_REQ_RESULT_INVALID_PSM	This is macro L2CAP_CONN_REQ_RESULT_INVALID_PSM.
	L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID	This is macro L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID.
	L2CAP_CONN_REQ_RESULT_NO_RESOURCES	This is macro L2CAP_CONN_REQ_RESULT_NO_RESOURCES.
	L2CAP_CONN_REQ_RESULT_PENDING	This is macro L2CAP_CONN_REQ_RESULT_PENDING.
	L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK	This is macro L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK.
	L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED	This is macro L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED.
	L2CAP_CONN_REQ_RESULT_SUCCESS	This is macro L2CAP_CONN_REQ_RESULT_SUCCESS.
	L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING	This is macro L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING.
	L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING	This is macro L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING.
	L2CAP_CONN_REQ_STATUS_NO_INFO	This is macro L2CAP_CONN_REQ_STATUS_NO_INFO.
	L2CAP_ECHO_MAX_DATA_LEN	echo request and response
	L2CAP_EXT_BI_QOS	This is macro L2CAP_EXT_BI_QOS.
	L2CAP_EXT_ENHANCED_RETRANSMISSION	This is macro L2CAP_EXT_ENHANCED_RETRANSMISSION.
	L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR	This is macro L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR.
	L2CAP_EXT_EXT_WINDOW_SIZE	This is macro L2CAP_EXT_EXT_WINDOW_SIZE.
	L2CAP_EXT_FCS_OPTION	This is macro L2CAP_EXT_FCS_OPTION.
	L2CAP_EXT_FIXED_CHANNELS	This is macro L2CAP_EXT_FIXED_CHANNELS.
	L2CAP_EXT_FLOW_CONTROL	This is macro L2CAP_EXT_FLOW_CONTROL.
	L2CAP_EXT_RETRANSMISSION	This is macro L2CAP_EXT_RETRANSMISSION.
	L2CAP_EXT_STREAMING	This is macro L2CAP_EXT_STREAMING.
	L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT	This is macro L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT.
	L2CAP_INFO_NOT_SUPPORTED	REDFLAG: should it be non zero?
	L2CAP_INFO_RESULT_SUCCESS	This is macro L2CAP_INFO_RESULT_SUCCESS.
	L2CAP_INFO_TYPE_CONNECTIONLESS_MTU	information request

	L2CAP_INFO_TYPE_EXTENDED_SUPPORT	This is macro L2CAP_INFO_TYPE_EXTENDED_SUPPORT.
	L2CAP_INFO_TYPE_FIXED_CHANNELS	This is macro L2CAP_INFO_TYPE_FIXED_CHANNELS.
	L2CAP_MAX_OPTIONS	configuration request/response
	L2CAP_OPTION_LEN_FLASH_QOS	This is macro L2CAP_OPTION_LEN_FLASH_QOS.
	L2CAP_OPTION_LEN_FLASH_RFC	This is macro L2CAP_OPTION_LEN_FLASH_RFC.
	L2CAP_OPTION_LEN_FLASH_TIMEOUT	This is macro L2CAP_OPTION_LEN_FLASH_TIMEOUT.
	L2CAP_OPTION_LEN_MAX_MTU	This is macro L2CAP_OPTION_LEN_MAX_MTU.
	L2CAP_OPTION_LEN_UNKNOWN	This is macro L2CAP_OPTION_LEN_UNKNOWN.
	L2CAP_OPTION_LEN_UNKNOWN_DATA	21
	L2CAP_OPTION_TYPE_FLASH_QOS	This is macro L2CAP_OPTION_TYPE_FLASH_QOS.
	L2CAP_OPTION_TYPE_FLASH_QOS_FLAG	This is macro L2CAP_OPTION_TYPE_FLASH_QOS_FLAG.
	L2CAP_OPTION_TYPE_FLASH_RFC	This is macro L2CAP_OPTION_TYPE_FLASH_RFC.
	L2CAP_OPTION_TYPE_FLASH_RFC_FLAG	This is macro L2CAP_OPTION_TYPE_FLASH_RFC_FLAG.
	L2CAP_OPTION_TYPE_FLASH_TIMEOUT	This is macro L2CAP_OPTION_TYPE_FLASH_TIMEOUT.
	L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG	This is macro L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG.
	L2CAP_OPTION_TYPE_MAX_MTU	This is macro L2CAP_OPTION_TYPE_MAX_MTU.
	L2CAP_OPTION_TYPE_MAX_MTU_FLAG	This is macro L2CAP_OPTION_TYPE_MAX_MTU_FLAG.
	L2CAP_OPTION_TYPE_UNKNOWN_FLAG	This is macro L2CAP_OPTION_TYPE_UNKNOWN_FLAG.
	L2CAP_REJECT_REASON_INVALID_CHANNEL	This is macro L2CAP_REJECT_REASON_INVALID_CHANNEL.
	L2CAP_REJECT_REASON_MTU_EXCEEDED	This is macro L2CAP_REJECT_REASON_MTU_EXCEEDED.
	L2CAP_REJECT_REASON_NOT_UNDERSTOOD	This is macro L2CAP_REJECT_REASON_NOT_UNDERSTOOD.
	L2CAP_RFC_BASIC	This is macro L2CAP_RFC_BASIC.
	L2CAP_RFC_ENHANCED_RETRANSMISSION	This is macro L2CAP_RFC_ENHANCED_RETRANSMISSION.
	L2CAP_RFC_ERETR_TIMEOUT	3 secs
	L2CAP_RFC_ERETR_TX_WINDOW	This is macro L2CAP_RFC_ERETR_TX_WINDOW.
	L2CAP_RFC_FLOW_CONTROL	This is macro L2CAP_RFC_FLOW_CONTROL.
	L2CAP_RFC_MONITOR_TIMEOUT	12 secs
	L2CAP_RFC_RETRANSMISSION	This is macro L2CAP_RFC_RETRANSMISSION.
	L2CAP_RFC_STREAMING	This is macro L2CAP_RFC_STREAMING.
	L2CAP_SERVICE_TYPE_BEST_EFFORT	This is macro L2CAP_SERVICE_TYPE_BEST_EFFORT.
	L2CAP_SERVICE_TYPE_GUARANTEED	This is macro L2CAP_SERVICE_TYPE_GUARANTEED.
	L2CAP_SERVICE_TYPE_NO_TRAFFIC	This is macro L2CAP_SERVICE_TYPE_NO_TRAFFIC.

Structures

	Name	Description
	_bt_l2cap_cfg_option_t	This is type bt_l2cap_cfg_option_t.
	_bt_l2cap_cmd_config_req_t	This is type bt_l2cap_cmd_config_req_t.
	_bt_l2cap_cmd_config_res_t	This is type bt_l2cap_cmd_config_res_t.
	_bt_l2cap_cmd_conn_param_update_req_t	connection parameter update
	_bt_l2cap_cmd_conn_param_update_res_t	This is type bt_l2cap_cmd_conn_param_update_res_t.
	_bt_l2cap_cmd_connection_req_t	This is type bt_l2cap_cmd_connection_req_t.
	_bt_l2cap_cmd_connection_res_t	This is type bt_l2cap_cmd_connection_res_t.
	_bt_l2cap_cmd_disconnection_req_t	This is type bt_l2cap_cmd_disconnection_req_t.
	_bt_l2cap_cmd_echo_req_t	This is type bt_l2cap_cmd_echo_req_t.
	_bt_l2cap_cmd_echo_res_t	This is type bt_l2cap_cmd_echo_res_t.
	_bt_l2cap_cmd_header_t	This is type bt_l2cap_cmd_header_t.
	_bt_l2cap_cmd_info_req_t	This is type bt_l2cap_cmd_info_req_t.
	_bt_l2cap_cmd_info_res_t	This is type bt_l2cap_cmd_info_res_t.
	_bt_l2cap_cmd_reject_t	This is type bt_l2cap_cmd_reject_t.
	_bt_l2cap_option_flash_timeout_t	This is type bt_l2cap_option_flash_timeout_t.
	_bt_l2cap_option_max_mtu_t	This is type bt_l2cap_option_max_mtu_t.

	_bt_l2cap_option_qos_t	This is type bt_l2cap_option_qos_t.
	_bt_l2cap_option_rfc_t	This is type bt_l2cap_option_rfc_t.
	_bt_l2cap_option_unknown_t	This is type bt_l2cap_option_unknown_t.
	_cmd_disconnection_res	This is type pcmd_disconnection_res.
	bt_l2cap_cfg_option_p	This is type bt_l2cap_cfg_option_p.
	bt_l2cap_cfg_option_t	This is type bt_l2cap_cfg_option_t.
	bt_l2cap_cmd_config_req_p	This is type bt_l2cap_cmd_config_req_p.
	bt_l2cap_cmd_config_req_t	This is type bt_l2cap_cmd_config_req_t.
	bt_l2cap_cmd_config_res_p	This is type bt_l2cap_cmd_config_res_p.
	bt_l2cap_cmd_config_res_t	This is type bt_l2cap_cmd_config_res_t.
	bt_l2cap_cmd_conn_param_update_req_t	connection parameter update
	bt_l2cap_cmd_conn_param_update_res_t	This is type bt_l2cap_cmd_conn_param_update_res_t.
	bt_l2cap_cmd_connection_req_p	This is type bt_l2cap_cmd_connection_req_p.
	bt_l2cap_cmd_connection_req_t	This is type bt_l2cap_cmd_connection_req_t.
	bt_l2cap_cmd_connection_res_p	This is type bt_l2cap_cmd_connection_res_p.
	bt_l2cap_cmd_connection_res_t	This is type bt_l2cap_cmd_connection_res_t.
	bt_l2cap_cmd_disconnection_req_p	This is type bt_l2cap_cmd_disconnection_req_p.
	bt_l2cap_cmd_disconnection_req_t	This is type bt_l2cap_cmd_disconnection_req_t.
	bt_l2cap_cmd_echo_req_p	This is type bt_l2cap_cmd_echo_req_p.
	bt_l2cap_cmd_echo_req_t	This is type bt_l2cap_cmd_echo_req_t.
	bt_l2cap_cmd_echo_res_p	This is type bt_l2cap_cmd_echo_res_p.
	bt_l2cap_cmd_echo_res_t	This is type bt_l2cap_cmd_echo_res_t.
	bt_l2cap_cmd_header_p	This is type bt_l2cap_cmd_header_p.
	bt_l2cap_cmd_header_t	This is type bt_l2cap_cmd_header_t.
	bt_l2cap_cmd_info_req_p	This is type bt_l2cap_cmd_info_req_p.
	bt_l2cap_cmd_info_req_t	This is type bt_l2cap_cmd_info_req_t.
	bt_l2cap_cmd_info_res_p	This is type bt_l2cap_cmd_info_res_p.
	bt_l2cap_cmd_info_res_t	This is type bt_l2cap_cmd_info_res_t.
	bt_l2cap_cmd_reject_p	This is type bt_l2cap_cmd_reject_p.
	bt_l2cap_cmd_reject_t	This is type bt_l2cap_cmd_reject_t.
	bt_l2cap_option_flash_timeout_t	This is type bt_l2cap_option_flash_timeout_t.
	bt_l2cap_option_max_mtu_t	This is type bt_l2cap_option_max_mtu_t.
	bt_l2cap_option_qos_t	This is type bt_l2cap_option_qos_t.
	bt_l2cap_option_rfc_t	This is type bt_l2cap_option_rfc_t.
	bt_l2cap_option_unknown_t	This is type bt_l2cap_option_unknown_t.
	cmd_disconnection_res	This is type cmd_disconnection_res.
	pcmd_disconnection_res	This is type pcmd_disconnection_res.

Types

	Name	Description
	pf_l2cap_cmd_callback	This is type pf_l2cap_cmd_callback.

Unions

	Name	Description
	_bt_l2cap_cmd_reject_param_t	This is type bt_l2cap_cmd_reject_param_t.
	_bt_l2cap_command_t	This is type bt_l2cap_command_t.
	bt_l2cap_cmd_reject_param_t	This is type bt_l2cap_cmd_reject_param_t.
	bt_l2cap_command_t	This is type bt_l2cap_command_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

config.h

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC.

Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

csr.h

Functions

	Name	Description
	btx_csr_alloc_bccmd_getreq	This is function btx_csr_alloc_bccmd_getreq.
	btx_csr_alloc_bccmd_setreq	This is function btx_csr_alloc_bccmd_setreq.
	btx_csr_autobaud	brief Configure controller's UART speed. ingroup btx_csr details This function makes the controller auto-configure its UART speed. The host transport must be set to H4. This function works only with BC6 controllers.
	btx_csr_bc7_sel_host_interface_h4	brief Configure controller's UART speed and host interface. ingroup btx_csr details This function makes the controller auto-configure its UART speed and select H4 as host interface. PS_KEY_HOST_INTERFACE must not be set. PS_KEY_UART_BITRATE must be set to 0. This function works only with BC7 controllers.
	btx_csr_enable_tx	brief Enable/disable transmitter ingroup btx_csr param enable Specifies whether the transmitter should be enable or disabled. param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	btx_csr_exec_hq_script	This is function btx_csr_exec_hq_script.
	btx_csr_exec_script	brief Patch controller's firmware ingroup btx_csr details This function executes a script that patches the controller's firmware. The c script must point to a structure that contain a complete patch script for a particular controller model and revision. If the revision specified in the script and revision read from the controller are the same btx_csr_patch_controller() loads the script to the controller and calls the c callback with the first parameter TRUE. Otherwise the c callback is called with the first parameter FALSE. If support for multiple firmware revisions is neede use btx_csr_patch_controller(). param script Array of... more
	btx_csr_get_cached_temperature	brief Get chip's cached temperature ingroup btx_csr param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	btx_csr_get_ps_var	This is function btx_csr_get_ps_var.
	btx_csr_get_ps_var_ex	This is function btx_csr_get_ps_var_ex.
	btx_csr_get_rssi_acl	brief Get RSSI ingroup btx_csr details This function retrieves the RSSI for a given HCI ACL handle. param hconn ACL connection handle. param callback The callback function that will be called after the command has completed. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	btx_csr_get_script__dsp_script__PB_109_DSP_rev8	brief Return script for patching DSP in CSR8x11 A08 (BlueCore 7) ingroup btx_csr
	btx_csr_get_script__PB_101_CSR8811_CSP28_UART	brief Return script for patching CSR8x11 A06 (BlueCore 7) ingroup btx_csr
	btx_csr_get_script__PB_109_CSR8811_REV16	brief Return script for patching CSR8x11 A08 (BlueCore 7) ingroup btx_csr
	btx_csr_get_script__PB_173_CSR8X11_REV1	brief Return script for patching CSR8x11 A12 (BlueCore 7) ingroup btx_csr
	btx_csr_get_script__PB_27_R20_BC6ROM_A04	brief Return script for patching BlueCore 6 ingroup btx_csr
	btx_csr_get_script__PB_90_REV6	brief Return script for patching CSR8810 (BlueCore 7) ingroup btx_csr
	btx_csr_get_var	This is function btx_csr_get_var.

	btx_csr_init	brief Initialize CSR support layer. ingroup btx_csr details This function initializes all internal variables of the CSR support layer. CSR controllers use vendor specific event (0xFF) to carry the BCCMD protocol. They also do not report number of completed packets for BCCMD commands. This function installs a vendor specific event handler that makes sure that callback are called on corresponding vendor specific commands and the number of free command buffers in the controller is kept correct.
	btx_csr_init_hq_script	This is function btx_csr_init_hq_script.
	btx_csr_patch_controller	brief Patch controller's firmware ingroup btx_csr details This function executes a script that patches the controller's firmware. Each entry of the c scripts array must be a complete patch script for a particular controller model and revision. btx_csr_patch_controller() reads the revision number from the controller then tries to find the corresponding script in the c scripts. If there is a matching script it is loaded to the controller and c callback is called with the first parameter TRUE. If no suitable script found c callback is called with the first parameter FALSE. param scripts Array of patch scripts.... more
	btx_csr_register_bccmd_listener	This is function btx_csr_register_bccmd_listener.
	btx_csr_send_dsp_config_data	This is function btx_csr_send_dsp_config_data.
	btx_csr_send_next_hq_script_packet	This is function btx_csr_send_next_hq_script_packet.
	btx_csr_set_ps_var	This is function btx_csr_set_ps_var.
	btx_csr_set_ps_var_ex	This is function btx_csr_set_ps_var_ex.
	btx_csr_set_ps_vars	brief Write PS variables ingroup btx_csr details param ps_vars PS values param buffer A buffer for storing temporary data during function execution. param callback The callback function that will be called when all PS values have been sent to the controller or error occurred. param callback_param A pointer to arbitrary data to be passed to the c callback callback..
	btx_csr_set_ps_vars_ex	brief Write PS variables ingroup btx_csr param ps_vars PS values param buffer A buffer for storing temporary data during function execution. param store param callback The callback function that will be called when all PS values have been sent to the controller or error occurred. param callback_param A pointer to arbitrary data to be passed to the c callback callback..
	btx_csr_set_var	This is function btx_csr_set_var.
	btx_csr_unregister_bccmd_listener	This is function btx_csr_unregister_bccmd_listener.
	btx_csr_warm_reset	brief Warm reset ingroup btx_csr details This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact.
	btx_csr_warm_reset_ex	brief Warm reset ingroup btx_csr details This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact. Since the controller does not respond to the warm reset command as it starts resetting immediately upon receiving the command, the c callback is called right after the command packet has been transmitted to the controller. param callback The callback function that will be called after the warm reset command has been sent to the controller. param callback_param A pointer to arbitrary data to be passed to the c callback callback.... more

Macros

	Name	Description
	CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN	This is macro CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN.
	CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN_ENABLE	This is macro CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN_ENABLE.
	CSR_I2S_STREAM_CFG_KEY_BITS_PER_SAMPLE	This is macro CSR_I2S_STREAM_CFG_KEY_BITS_PER_SAMPLE.
	CSR_I2S_STREAM_CFG_KEY_CROP_ENABLE	This is macro CSR_I2S_STREAM_CFG_KEY_CROP_ENABLE.
	CSR_I2S_STREAM_CFG_KEY_JUSTIFY_DELAY	This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_DELAY.
	CSR_I2S_STREAM_CFG_KEY_JUSTIFY_FORMAT	This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_FORMAT.

	CSR_I2S_STREAM_CFG_KEY_JUSTIFY_RESOLUTION	This is macro CSR_I2S_STREAM_CFG_KEY_JUSTIFY_RESOLUTION.
	CSR_I2S_STREAM_CFG_KEY_MASTER	This is macro CSR_I2S_STREAM_CFG_KEY_MASTER.
	CSR_I2S_STREAM_CFG_KEY_MCLK	This is macro CSR_I2S_STREAM_CFG_KEY_MCLK.
	CSR_I2S_STREAM_CFG_KEY_POLARITY	This is macro CSR_I2S_STREAM_CFG_KEY_POLARITY.
	CSR_I2S_STREAM_CFG_KEY_RX_START_SAMPLE	This is macro CSR_I2S_STREAM_CFG_KEY_RX_START_SAMPLE.
	CSR_I2S_STREAM_CFG_KEY_SYNC_RATE	This is macro CSR_I2S_STREAM_CFG_KEY_SYNC_RATE.
	CSR_I2S_STREAM_CFG_KEY_TX_START_SAMPLE	This is macro CSR_I2S_STREAM_CFG_KEY_TX_START_SAMPLE.
	CSR_MAX_HQ_PACKET_LEN	This is macro CSR_MAX_HQ_PACKET_LEN.
	CSR_MSG_TYPE_GETREQ	This is macro CSR_MSG_TYPE_GETREQ.
	CSR_MSG_TYPE_GETRESP	This is macro CSR_MSG_TYPE_GETRESP.
	CSR_MSG_TYPE_SETREQ	This is macro CSR_MSG_TYPE_SETREQ.
	CSR_PCM_STREAM_CFG_KEY_LSB_FIRST	This is macro CSR_PCM_STREAM_CFG_KEY_LSB_FIRST.
	CSR_PCM_STREAM_CFG_KEY_MANCHESTER	This is macro CSR_PCM_STREAM_CFG_KEY_MANCHESTER.
	CSR_PCM_STREAM_CFG_KEY_MANCHESTER_SLAVE	This is macro CSR_PCM_STREAM_CFG_KEY_MANCHESTER_SLAVE.
	CSR_PCM_STREAM_CFG_KEY_MASTER	This is macro CSR_PCM_STREAM_CFG_KEY_MASTER.
	CSR_PCM_STREAM_CFG_KEY_MCLK	This is macro CSR_PCM_STREAM_CFG_KEY_MCLK.
	CSR_PCM_STREAM_CFG_KEY_SHORT_SYNC	This is macro CSR_PCM_STREAM_CFG_KEY_SHORT_SYNC.
	CSR_PCM_STREAM_CFG_KEY_SIGN_EXTEND	This is macro CSR_PCM_STREAM_CFG_KEY_SIGN_EXTEND.
	CSR_PCM_STREAM_CFG_KEY_SLOT_COUNT	This is macro CSR_PCM_STREAM_CFG_KEY_SLOT_COUNT.
	CSR_PCM_STREAM_CFG_KEY_SYNC_RATE	This is macro CSR_PCM_STREAM_CFG_KEY_SYNC_RATE.
	CSR_PCM_STREAM_CFG_KEY_TX_TRISTATE	This is macro CSR_PCM_STREAM_CFG_KEY_TX_TRISTATE.
	CSR_SNK_ADC	This is macro CSR_SNK_ADC.
	CSR_SNK_FASTPIPE	This is macro CSR_SNK_FASTPIPE.
	CSR_SNK_FM	This is macro CSR_SNK_FM.
	CSR_SNK_I2S	This is macro CSR_SNK_I2S.
	CSR_SNK_L2CAP	This is macro CSR_SNK_L2CAP.
	CSR_SNK_PCM	This is macro CSR_SNK_PCM.
	CSR_SNK_SCO	This is macro CSR_SNK_SCO.
	CSR_SNK_SPDIF	This is macro CSR_SNK_SPDIF.
	CSR_SRC_ADC	This is macro CSR_SRC_ADC.
	CSR_SRC_FASTPIPE	This is macro CSR_SRC_FASTPIPE.
	CSR_SRC_FM	This is macro CSR_SRC_FM.
	CSR_SRC_I2S	This is macro CSR_SRC_I2S.
	CSR_SRC_L2CAP	This is macro CSR_SRC_L2CAP.
	CSR_SRC_MIC	This is macro CSR_SRC_MIC.
	CSR_SRC_PCM	This is macro CSR_SRC_PCM.
	CSR_SRC_SCO	This is macro CSR_SRC_SCO.
	CSR_SRC_SPDIF	This is macro CSR_SRC_SPDIF.
	CSR_VARID_CACHED_TEMPERATURE	This is macro CSR_VARID_CACHED_TEMPERATURE.
	CSR_VARID_CAPABILITY_DOWNLOAD_INDICATION	This is macro CSR_VARID_CAPABILITY_DOWNLOAD_INDICATION.
	CSR_VARID_CREATE_OPERATOR_C	This is macro CSR_VARID_CREATE_OPERATOR_C.
	CSR_VARID_CREATE_OPERATOR_P	This is macro CSR_VARID_CREATE_OPERATOR_P.
	CSR_VARID_DESTROY_OPERATORS	This is macro CSR_VARID_DESTROY_OPERATORS.
	CSR_VARID_DSPMANAGER_CONFIG_REQUEST	This is macro CSR_VARID_DSPMANAGER_CONFIG_REQUEST.
	CSR_VARID_ENABLE_SCO_STREAMS	This is macro CSR_VARID_ENABLE_SCO_STREAMS.
	CSR_VARID_MAP_SCO_AUDIO	This is macro CSR_VARID_MAP_SCO_AUDIO.
	CSR_VARID_MESSAGE_FROM_OPERATOR	Select Variable definitions
	CSR_VARID_PIO	This is macro CSR_VARID_PIO.
	CSR_VARID_PIO_DIRECTION_MASK	This is macro CSR_VARID_PIO_DIRECTION_MASK.
	CSR_VARID_PIO_PROTECT_MASK	This is macro CSR_VARID_PIO_PROTECT_MASK.
	CSR_VARID_RSSI_ACL	This is macro CSR_VARID_RSSI_ACL.

	CSR_VARID_START_OPERATORS	This is macro CSR_VARID_START_OPERATORS.
	CSR_VARID_STOP_OPERATORS	This is macro CSR_VARID_STOP_OPERATORS.
	CSR_VARID_STREAM_CLOSE_SINK	This is macro CSR_VARID_STREAM_CLOSE_SINK.
	CSR_VARID_STREAM_CLOSE_SOURCE	This is macro CSR_VARID_STREAM_CLOSE_SOURCE.
	CSR_VARID_STREAM_CONFIGURE	This is macro CSR_VARID_STREAM_CONFIGURE.
	CSR_VARID_STREAM_CONNECT	This is macro CSR_VARID_STREAM_CONNECT.
	CSR_VARID_STREAM_DRAINED_NOTIFICATION	This is macro CSR_VARID_STREAM_DRAINED_NOTIFICATION.
	CSR_VARID_STREAM_GET_SINK	This is macro CSR_VARID_STREAM_GET_SINK.
	CSR_VARID_STREAM_GET_SOURCE	This is macro CSR_VARID_STREAM_GET_SOURCE.
	CSR_VARID_STREAM_TRANSFORM_DISCONNECT	This is macro CSR_VARID_STREAM_TRANSFORM_DISCONNECT.
	GETRESP_BAD_REQ	This is macro GETRESP_BAD_REQ.
	GETRESP_ERROR	This is macro GETRESP_ERROR.
	GETRESP_NO_ACCESS	This is macro GETRESP_NO_ACCESS.
	GETRESP_NO_SUCH_VARID	This is macro GETRESP_NO_SUCH_VARID.
	GETRESP_NO_VALUE	This is macro GETRESP_NO_VALUE.
	GETRESP_OK	BCCMD GETRESP status codes
	GETRESP_PERMISSION_DENIED	This is macro GETRESP_PERMISSION_DENIED.
	GETRESP_READ_ONLY	This is macro GETRESP_READ_ONLY.
	GETRESP_TOO_BIG	This is macro GETRESP_TOO_BIG.
	GETRESP_WRITE_ONLY	This is macro GETRESP_WRITE_ONLY.
	PS_DEFAULT	This is macro PS_DEFAULT.
	PS_F	This is macro PS_F.
	PS_I	This is macro PS_I.
	PS_RAM	This is macro PS_RAM.
	PS_ROM	This is macro PS_ROM.
	PSKEY_ANA_FREQ	This is macro PSKEY_ANA_FREQ.
	PSKEY_BDADDR	Select PS key definitions
	PSKEY_BLE_DEFAULT_TX_POWER	This is macro PSKEY_BLE_DEFAULT_TX_POWER.
	PSKEY_CLOCK_REQUEST_ENABLE	This is macro PSKEY_CLOCK_REQUEST_ENABLE.
	PSKEY_DEEP_SLEEP_CLEAR_RTS	This is macro PSKEY_DEEP_SLEEP_CLEAR_RTS.
	PSKEY_DEEP_SLEEP_EXTERNAL_CLOCK_SOURCE_PIO	This is macro PSKEY_DEEP_SLEEP_EXTERNAL_CLOCK_SOURCE_PIO.
	PSKEY_DEEP_SLEEP_STATE	This is macro PSKEY_DEEP_SLEEP_STATE.
	PSKEY_DEEP_SLEEP_USE_EXTERNAL_CLOCK	This is macro PSKEY_DEEP_SLEEP_USE_EXTERNAL_CLOCK.
	PSKEY_DEEP_SLEEP_WAKE_CTS	This is macro PSKEY_DEEP_SLEEP_WAKE_CTS.
	PSKEY_DIGITAL_AUDIO_BITS_PER_SAMPLE	This is macro PSKEY_DIGITAL_AUDIO_BITS_PER_SAMPLE.
	PSKEY_DIGITAL_AUDIO_CONFIG	This is macro PSKEY_DIGITAL_AUDIO_CONFIG.
	PSKEY_H_HC_FC_MAX_SCO_PKT_LEN	This is macro PSKEY_H_HC_FC_MAX_SCO_PKT_LEN.
	PSKEY_H_HC_FC_MAX_SCO_PKTS	This is macro PSKEY_H_HC_FC_MAX_SCO_PKTS.
	PSKEY_HCI_NOP_DISABLE	This is macro PSKEY_HCI_NOP_DISABLE.
	PSKEY_HOST_INTERFACE	This is macro PSKEY_HOST_INTERFACE.
	PSKEY_HOSTIO_MAP_SCO_PCM	This is macro PSKEY_HOSTIO_MAP_SCO_PCM.
	PSKEY_HOSTIO_UART_RESET_TIMEOUT	This is macro PSKEY_HOSTIO_UART_RESET_TIMEOUT.
	PSKEY_LC_DEFAULT_TX_POWER	This is macro PSKEY_LC_DEFAULT_TX_POWER.
	PSKEY_LC_MAX_TX_POWER	This is macro PSKEY_LC_MAX_TX_POWER.
	PSKEY_LC_MAX_TX_POWER_NO_RSSI	This is macro PSKEY_LC_MAX_TX_POWER_NO_RSSI.
	PSKEY_PCM_CLOCK_RATE	This is macro PSKEY_PCM_CLOCK_RATE.
	PSKEY_PCM_CONFIG32	This is macro PSKEY_PCM_CONFIG32.
	PSKEY_PCM_FORMAT	This is macro PSKEY_PCM_FORMAT.
	PSKEY_PCM_PULL_CONTROL	This is macro PSKEY_PCM_PULL_CONTROL.
	PSKEY_PCM_SYNC_RATE	This is macro PSKEY_PCM_SYNC_RATE.
	PSKEY_PCM_USE_LOW_JITTER_MODE	This is macro PSKEY_PCM_USE_LOW_JITTER_MODE.
	PSKEY_PIO_DEEP_SLEEP_EITHER_LEVEL	This is macro PSKEY_PIO_DEEP_SLEEP_EITHER_LEVEL.
	PSKEY_UART_BAUDRATE	This is macro PSKEY_UART_BAUDRATE.

	PSKEY_UART_BITRATE	This is macro PSKEY_UART_BITRATE.
	PSKEY_UART_CONFIG_BCSP	This is macro PSKEY_UART_CONFIG_BCSP.
	PSKEY_UART_CONFIG_H4	This is macro PSKEY_UART_CONFIG_H4.
	PSKEY_UART_CONFIG_H4DS	This is macro PSKEY_UART_CONFIG_H4DS.
	PSKEY_UART_CONFIG_H5	This is macro PSKEY_UART_CONFIG_H5.
	PSKEY_UART_TX_WINDOW_SIZE	This is macro PSKEY_UART_TX_WINDOW_SIZE.
	PSKEY_VM_DISABLE	This is macro PSKEY_VM_DISABLE.
	SET_PS_VALUE_BDADDR	This is macro SET_PS_VALUE_BDADDR.
	SET_PS_VALUE_UINT16	Macros for defining lists of PS values for use with btx_csr_set_ps_vars().
	SET_PS_VALUE_UINT32	This is macro SET_PS_VALUE_UINT32.

Structures

	Name	Description
	_btx_csr_autobaud_buffer_t	This is record _btx_csr_autobaud_buffer_t.
	_btx_csr_bccmd_header_s	This is type btx_csr_bccmd_header_t.
	_btx_csr_bccmd_listener_t	This is record _btx_csr_bccmd_listener_t.
	_btx_csr_cached_temperature_s	This is type btx_csr_cached_temperature_t.
	_btx_csr_create_operator_c_s	This is type btx_csr_create_operator_c_t.
	_btx_csr_exec_hq_script_buffer_t	This is record _btx_csr_exec_hq_script_buffer_t.
	_btx_csr_exec_script_buffer_t	This is record _btx_csr_exec_script_buffer_t.
	_btx_csr_pio_direction_mask_s	This is type btx_csr_pio_direction_mask_t.
	_btx_csr_pio_protection_mask_s	This is type btx_csr_pio_protection_mask_t.
	_btx_csr_pio_s	This is type btx_csr_pio_t.
	_btx_csr_rssi_acl_s	This is type btx_csr_rssi_acl_t.
	_btx_csr_script_t	This is type btx_csr_script_t.
	_btx_csr_set_ps_vars_buffer_t	This is record _btx_csr_set_ps_vars_buffer_t.
	_btx_csr_strm_connect_s	This is type btx_csr_strm_connect_t.
	_btx_csr_strm_get_sink_s	This is type btx_csr_strm_get_sink_t.
	_btx_csr_strm_get_source_s	This is type btx_csr_strm_get_source_t.
	btx_csr_bccmd_header_t	This is type btx_csr_bccmd_header_t.
	btx_csr_cached_temperature_t	This is type btx_csr_cached_temperature_t.
	btx_csr_create_operator_c_t	This is type btx_csr_create_operator_c_t.
	btx_csr_pio_direction_mask_t	This is type btx_csr_pio_direction_mask_t.
	btx_csr_pio_protection_mask_t	This is type btx_csr_pio_protection_mask_t.
	btx_csr_pio_t	This is type btx_csr_pio_t.
	btx_csr_rssi_acl_t	This is type btx_csr_rssi_acl_t.
	btx_csr_script_t	This is type btx_csr_script_t.
	btx_csr_strm_connect_t	This is type btx_csr_strm_connect_t.
	btx_csr_strm_get_sink_t	This is type btx_csr_strm_get_sink_t.
	btx_csr_strm_get_source_t	This is type btx_csr_strm_get_source_t.

Types

	Name	Description
	btx_csr_autobaud_buffer_t	This is type btx_csr_autobaud_buffer_t.
	btx_csr_autobaud_callback_fp	This is type btx_csr_autobaud_callback_fp.
	btx_csr_bccmd_callback_fp	This is type btx_csr_bccmd_callback_fp.
	btx_csr_bccmd_listener_t	This is type btx_csr_bccmd_listener_t.
	btx_csr_exec_hq_script_buffer_t	This is type btx_csr_exec_hq_script_buffer_t.
	btx_csr_exec_hq_script_callback_fp	This is type btx_csr_exec_hq_script_callback_fp.
	btx_csr_exec_script_buffer_t	This is type btx_csr_exec_script_buffer_t.
	btx_csr_exec_script_callback_fp	This is type btx_csr_exec_script_callback_fp.
	btx_csr_get_ps_var_callback_fp	This is type btx_csr_get_ps_var_callback_fp.
	btx_csr_get_script_fp	This is type btx_csr_get_script_fp.
	btx_csr_get_var_callback_fp	This is type btx_csr_get_var_callback_fp.

	btx_csr_set_ps_vars_buffer_t	This is type btx_csr_set_ps_vars_buffer_t.
	btx_csr_set_ps_vars_callback_fp	This is type btx_csr_set_ps_vars_callback_fp.
	btx_csr_set_var_callback_fp	This is type btx_csr_set_var_callback_fp.

Unions

	Name	Description
	_btx_csr_var_u	This is type btx_csr_var_t.
	btx_csr_var_t	This is type btx_csr_var_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

frame_buffer.h

Functions

	Name	Description
	bt_l2cap_alloc_frame_buffer	This is function bt_l2cap_alloc_frame_buffer.
	bt_l2cap_free_frame_buffer	This is function bt_l2cap_free_frame_buffer.
	bt_l2cap_init_frame_buffers	This is function bt_l2cap_init_frame_buffers.

Macros

	Name	Description
	L2CAP_MAX_FRAME_BUFFERS	This is macro L2CAP_MAX_FRAME_BUFFERS.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

gap.h

Functions

	Name	Description
	bt_gap_find_devices	This is function bt_gap_find_devices.

Structures

	Name	Description
	_bt_device_t	This is type bt_device_t.
	bt_device_t	This is type bt_device_t.

Types

	Name	Description
	bt_find_devices_callback_fp	This is type bt_find_devices_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

hci.h

Functions

	Name	Description
	bt_hci_add_param_bdaddr	brief Add BD address parameter to an HCI command ingroup hci

	bt_hci_add_param_byte	brief Add byte parameter to an HCI command ingroup hci
	bt_hci_add_param_cod	brief Add class of device parameter to an HCI command ingroup hci
	bt_hci_add_param_int	brief Add int parameter to an HCI command ingroup hci
	bt_hci_add_param_linkkey	brief Add link key parameter to an HCI command ingroup hci
	bt_hci_add_param_long	brief Add long parameter to an HCI command ingroup hci
	bt_hci_add_param_string	brief Add string parameter to an HCI command ingroup hci
	bt_hci_alloc_canned_command	brief Allocate and initialize an HCI command structure for a canned (pre-formatted) command. ingroup hci
	bt_hci_alloc_command	brief Allocate and initialize an HCI command structure. ingroup hci
	bt_hci_authenticate_ex	This is function bt_hci_authenticate_ex.
	bt_hci_cancel_find_devices	brief Stop inquiry
	bt_hci_cancel_find_devices_ex	brief Stop inquiry
	bt_hci_cancel_request_remote_name	brief Cancel remote device name request
	bt_hci_cancel_send_acl_data	brief Cancel sending data over ACL connection
	bt_hci_connect	brief Connect to a remote device. ingroup hci details This function tries to establish an HCI connection with a remote device specified by the Bluetooth address c dest. Upon completion, the callback function specified by the c callback parameter is called. param dest Bluetooth address of the remote device. param packet_type param role_switch param acl_config param callback Pointer to a callback function that is called when the connect operation completes. param param Pointer to arbitrary data that is to be passed to the callback function. return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function... more
	bt_hci_connect_sco	This is function bt_hci_connect_sco.
	bt_hci_disconnect	brief Abort connection ingroup hci
	bt_hci_exit_park_state	brief Exit park state
	bt_hci_exit_sniff_mode_ex	This is function bt_hci_exit_sniff_mode_ex.
	bt_hci_find_devices	brief Start inquiry
	bt_hci_find_devices_ex	This is function bt_hci_find_devices_ex.
	bt_hci_free_command	brief Free HCI command. ingroup hci
	bt_hci_get_evt_param_bdaddr	brief Get bd address parameter from HCI event ingroup hci
	bt_hci_get_evt_param_byte	brief Get byte parameter from HCI event ingroup hci
	bt_hci_get_evt_param_devclass	brief Get class of device parameter from HCI event ingroup hci
	bt_hci_get_evt_param_int	brief Get int parameter from HCI event ingroup hci
	bt_hci_get_evt_param_linkkey	brief Get link key parameter from HCI event ingroup hci
	bt_hci_get_evt_param_long	brief Get long parameter from HCI event ingroup hci
	bt_hci_get_evt_param_uint	brief Get unsigned int parameter from HCI event ingroup hci
	bt_hci_get_evt_param_ulong	brief Get unsigned long parameter from HCI event ingroup hci
	bt_hci_get_inquiry_response_tx_power_level	This is function bt_hci_get_inquiry_response_tx_power_level.
	bt_hci_get_last_cmd_status	This is function bt_hci_get_last_cmd_status.
	bt_hci_get_param_bdaddr	brief Get BD address parameter from HCI command ingroup hci
	bt_hci_get_param_byte	brief Get byte parameter from HCI command ingroup hci
	bt_hci_get_param_int	brief Get int parameter from HCI command ingroup hci
	bt_hci_get_param_linkkey	brief Get link key parameter from HCI command ingroup hci
	bt_hci_get_param_long	brief Get long parameter from HCI command ingroup hci
	bt_hci_get_recv_buffer	This is function bt_hci_get_recv_buffer.
	bt_hci_get_recv_buffer_len	This is function bt_hci_get_recv_buffer_len.
	bt_hci_get_send_buffer	This is function bt_hci_get_send_buffer.
	bt_hci_get_send_buffer_len	This is function bt_hci_get_send_buffer_len.
	bt_hci_init	brief Initialize the HCI layer. ingroup hci This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt_sys_init. This function essentially calls bt_hci_init_ex(HCI_LINK_POLICY_ENABLE_ALL) so all link policy setting are enabled.

	bt_hci_init_ex	brief Initialize the HCI layer. ingroup hci This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt_sys_init_ex. @param default_link_policy default link policy settings. This is a bitmask that defines the initial value of the link policy settings for all new BR/EDR connections. This value can be a combination of the following values: li HCI_LINK_POLICY_ENABLE_ROLE_SWITCH li HCI_LINK_POLICY_ENABLE_HOLD_MODE li HCI_LINK_POLICY_ENABLE_SNIFF_MODE li HCI_LINK_POLICY_ENABLE_PARK_STATE To enable all settings pass HCI_LINK_POLICY_ENABLE_ALL.
	bt_hci_listen	brief Listen for incoming connections. ingroup hci details param callback Pointer to a callback function that is called when a new incoming connection has been established. param param Pointer to arbitrary data that is to be passed to the callback function. return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.
	bt_hci_listen_sco	This is function bt_hci_listen_sco.
	bt_hci_park_state	brief Put local device to park state
	bt_hci_read_inquiry_mode	brief Get current inquiry mode
	bt_hci_read_inquiry_scan_activity	brief Get current inquiry scan activity configuration
	bt_hci_read_inquiry_scan_type	brief Get current inquiry scan type
	bt_hci_read_page_scan_activity	brief Get current page scan activity configuration
	bt_hci_read_page_scan_period_mode	brief Get current page scan period mode
	bt_hci_read_page_scan_type	brief Get current page scan type
	bt_hci_read_page_timeout	brief Get current page timeout
	bt_hci_reject_pin_code	This is function bt_hci_reject_pin_code.
	bt_hci_request_remote_name	brief Request remote device's name
	bt_hci_reset	brief Reset controller. ingroup hci details This function resets the BT controller. param callback Completion callback. Called when the controller has been reset. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	bt_hci_role_change_ex	brief Change local device's role
	bt_hci_send_acl_data	brief Send data over ACL connection
	bt_hci_send_cmd	brief Send a command to local device
	bt_hci_send_linkkey	This is function bt_hci_send_linkkey.
	bt_hci_send_pin_code_ex	This is function bt_hci_send_pin_code_ex.
	bt_hci_send_sco_data	brief Send data over SCO connection
	bt_hci_set_encryption_ex	brief Set connection encryption
	bt_hci_set_incoming_connection_role	This is function bt_hci_set_incoming_connection_role.
	bt_hci_set_scan	This is function bt_hci_set_scan.
	bt_hci_set_scan_ex	This is function bt_hci_set_scan_ex.
	bt_hci_sniff_mode_ex	This is function bt_hci_sniff_mode_ex.
	bt_hci_sniff_subrating_ex	This is function bt_hci_sniff_subrating_ex.
	bt_hci_start	brief Start HCI layer. ingroup hci details This function starts the HCI layer of the stack. Starting the HCI layer consists essentially of two steps: 1. Make the HCI transport receive packets from the controller. This results in a call to bt_oem_recv. 2. Reset and configure the controller. Upon completion of controller initialization the callback function passed in the c callback parameter is called. param callback Completion callback. Called when controller initialization is complete. param callback_param A pointer to arbitrary data to be passed to the c callback callback. param enable_scan This is a bitmask that defines which scans are enabled... more

	bt_hci_start_no_init	brief Start HCI layer without controller configuration. ingroup hci details This function is similar to bt_hci_start but unlike the former it does not perform the controller configuration. I.e., bt_hci_start_no_init simply calls the HCI transport and makes it receive packets from the controller. The main purpose of this function is make the HCI transport ready to exchange packets if controller needs some vendor specific configuration before it can be used with the stack. E.g., controllers based on CRS8811 chip need loading various values that configure its operating mode using CSR's proprietary protocol. So the application after configuring the HCI transport would... more
	bt_hci_stop	brief Stop HCI layer. ingroup hci details This function makes the HCI layer inoperable. After this call the application must perform the full reset of the HCI transport and stack. param callback Completion callback. Called when the HCI layer has been stopped. param callback_param A pointer to arbitrary data to be passed to the c callback callback.
	bt_hci_write_default_link_policy_settings	This is function bt_hci_write_default_link_policy_settings.
	bt_hci_write_inquiry_mode	brief Configure inquiry mode
	bt_hci_write_inquiry_scan_activity	brief Configure inquiry scan activity
	bt_hci_write_inquiry_scan_type	brief Configure inquiry scan type
	bt_hci_write_link_policy_settings	This is function bt_hci_write_link_policy_settings.
	bt_hci_write_local_name	brief Write local device name ingroup hci details The Write_Local_Name command provides the ability to modify the userfriendly name for the BR/EDR Controller.
	bt_hci_write_local_name_ex	This is function bt_hci_write_local_name_ex.
	bt_hci_write_page_scan_activity	brief Configure page scan activity
	bt_hci_write_page_scan_period_mode	brief Configure page scan period mode
	bt_hci_write_page_scan_type	brief Configure page scan type
	bt_hci_write_page_timeout	brief Configure page timeout
	hci_sleep	brief Put local device to sleep
	hci_wakeup	brief Awaken local device

Macros

	Name	Description
	bt_hci_add_param_hconn	brief Add connection handle parameter to an HCI command ingroup hci
	bt_hci_add_param_lap	This is macro bt_hci_add_param_lap.
	bt_hci_add_param_uint	brief Add unsigned int parameter to an HCI command ingroup hci
	bt_hci_add_param_ulong	brief Add unsigned long parameter to an HCI command ingroup hci
	bt_hci_authenticate	This is macro bt_hci_authenticate.
	bt_hci_exit_sniff_mode	brief Cancel sniff mode
	bt_hci_get_evt_param_hconn	brief Get connection handle parameter from HCI event ingroup hci
	bt_hci_get_evt_param_uint	This is macro bt_hci_get_evt_param_uint.
	bt_hci_get_evt_param_ulong	This is macro bt_hci_get_evt_param_ulong.
	bt_hci_get_param_hconn	brief Get connection handle parameter from HCI command ingroup hci
	bt_hci_role_change	This is macro bt_hci_role_change.
	bt_hci_send_pin_code	This is macro bt_hci_send_pin_code.
	bt_hci_set_encryption	This is macro bt_hci_set_encryption.
	bt_hci_sniff_mode	brief Put local device to sniff mode
	bt_hci_sniff_subrating	This is macro bt_hci_sniff_subrating.
	HCI_ACCEPT_CONNECTION_REQUEST	This is macro HCI_ACCEPT_CONNECTION_REQUEST.
	HCI_ACCEPT_SYNCH_CONNECTION_REQUEST	This is macro HCI_ACCEPT_SYNCH_CONNECTION_REQUEST.
	HCI_ACL_DATA_HEADER_LEN	This is macro HCI_ACL_DATA_HEADER_LEN.
	HCI_AUTHENTICATION_REQUESTED	This is macro HCI_AUTHENTICATION_REQUESTED.
	HCI_BB_PACKET_TYPE_DH1	This is macro HCI_BB_PACKET_TYPE_DH1.

	HCI_BB_PACKET_TYPE_DH3	This is macro HCI_BB_PACKET_TYPE_DH3.
	HCI_BB_PACKET_TYPE_DH5	This is macro HCI_BB_PACKET_TYPE_DH5.
	HCI_BB_PACKET_TYPE_DM1	This is macro HCI_BB_PACKET_TYPE_DM1.
	HCI_BB_PACKET_TYPE_DM3	This is macro HCI_BB_PACKET_TYPE_DM3.
	HCI_BB_PACKET_TYPE_DM5	This is macro HCI_BB_PACKET_TYPE_DM5.
	HCI_BB_PACKET_TYPE_NO_2_DH1	This is macro HCI_BB_PACKET_TYPE_NO_2_DH1.
	HCI_BB_PACKET_TYPE_NO_2_DH3	This is macro HCI_BB_PACKET_TYPE_NO_2_DH3.
	HCI_BB_PACKET_TYPE_NO_2_DH5	This is macro HCI_BB_PACKET_TYPE_NO_2_DH5.
	HCI_BB_PACKET_TYPE_NO_3_DH1	This is macro HCI_BB_PACKET_TYPE_NO_3_DH1.
	HCI_BB_PACKET_TYPE_NO_3_DH3	This is macro HCI_BB_PACKET_TYPE_NO_3_DH3.
	HCI_BB_PACKET_TYPE_NO_3_DH5	This is macro HCI_BB_PACKET_TYPE_NO_3_DH5.
	HCI_C2H_BROADCAST_NOT_PARCKED_SLAVE	This is macro HCI_C2H_BROADCAST_NOT_PARCKED_SLAVE.
	HCI_C2H_BROADCAST_P2P	This is macro HCI_C2H_BROADCAST_P2P.
	HCI_C2H_BROADCAST_PARCKED_SLAVE	This is macro HCI_C2H_BROADCAST_PARCKED_SLAVE.
	HCI_C2H_BROADCAST_RESERVED	This is macro HCI_C2H_BROADCAST_RESERVED.
	HCI_CHANGE_CONNECTION_LINK_KEY	This is macro HCI_CHANGE_CONNECTION_LINK_KEY.
	HCI_CHANGE_CONNECTION_PACKET_TYPE	This is macro HCI_CHANGE_CONNECTION_PACKET_TYPE.
	HCI_CMD_HEADER_LEN	This is macro HCI_CMD_HEADER_LEN.
	HCI_CMD_STATUS_BEING_SENT	This is macro HCI_CMD_STATUS_BEING_SENT.
	HCI_CMD_STATUS_PENDING	This is macro HCI_CMD_STATUS_PENDING.
	HCI_CMD_STATUS_WAITING_RESPONSE	This is macro HCI_CMD_STATUS_WAITING_RESPONSE.
	HCI_CONFIG_BECOME_MASTER	This is macro HCI_CONFIG_BECOME_MASTER.
	HCI_CONFIG_ENABLE_AUTHENTICATION	This is macro HCI_CONFIG_ENABLE_AUTHENTICATION.
	HCI_CONFIG_ENABLE_ENCRYPTION	This is macro HCI_CONFIG_ENABLE_ENCRYPTION.
	HCI_CONNECTABLE	This is macro HCI_CONNECTABLE.
	HCI_CONTROLLER	This is macro HCI_CONTROLLER.
	HCI_CREATE_CONNECTION	This is macro HCI_CREATE_CONNECTION.
	HCI_CREATE_CONNECTION_CANCEL	This is macro HCI_CREATE_CONNECTION_CANCEL.
	HCI_CREATE_NEW_UNIT_KEY	This is macro HCI_CREATE_NEW_UNIT_KEY.
	HCI_DATA_STATUS_BEING_SENT	This is macro HCI_DATA_STATUS_BEING_SENT.
	HCI_DATA_STATUS_PENDING	This is macro HCI_DATA_STATUS_PENDING.
	HCI_DEFAULT_ACL_CONFIG	Default value for acl_config parameter of bt_hci_connect()
	HCI_DELETE_STORED_LINK_KEY	This is macro HCI_DELETE_STORED_LINK_KEY.
	HCI_DISCONNECT	This is macro HCI_DISCONNECT.
	HCI_DISCOVERABLE	This is macro HCI_DISCOVERABLE.
	HCI_DISCOVERABLE_MODE_GENERAL	This is macro HCI_DISCOVERABLE_MODE_GENERAL.
	HCI_DISCOVERABLE_MODE_LIMITED	This is macro HCI_DISCOVERABLE_MODE_LIMITED.
	HCI_ENABLE_DEVICE_UNDER_TEST_MODE	This is macro HCI_ENABLE_DEVICE_UNDER_TEST_MODE.
	HCI_ENCRYPTION_OFF	This is macro HCI_ENCRYPTION_OFF.
	HCI_ENCRYPTION_ON	This is macro HCI_ENCRYPTION_ON.
	HCI_ENHANCED_FLUSH	This is macro HCI_ENHANCED_FLUSH.
	HCI_EVT_ALL_HCI_EVENTS	Used in bt_hci_ctrl_register_event_listener to indicate that the listener is called for all HCI events (excluding dotstack internal ones)
	HCI_EVT_AUTHENTICATION_COMPLETE	This is macro HCI_EVT_AUTHENTICATION_COMPLETE.
	HCI_EVT_CHANGE_CONN_LINK_COMPLETE	This is macro HCI_EVT_CHANGE_CONN_LINK_COMPLETE.
	HCI_EVT_CMD_SEND_FINISHED	This is macro HCI_EVT_CMD_SEND_FINISHED.

	HCI_EVT_CMD_SEND_STARTED	dotstack internal events
	HCI_EVT_COMMAND_COMPLETE	This is macro HCI_EVT_COMMAND_COMPLETE.
	HCI_EVT_COMMAND_COMPLETE_PARAM_LEN	This is macro HCI_EVT_COMMAND_COMPLETE_PARAM_LEN.
	HCI_EVT_COMMAND_STATUS	This is macro HCI_EVT_COMMAND_STATUS.
	HCI_EVT_CONN_PACKET_TYPE_CHANGED	This is macro HCI_EVT_CONN_PACKET_TYPE_CHANGED.
	HCI_EVT_CONNECTION_COMPLETE	This is macro HCI_EVT_CONNECTION_COMPLETE.
	HCI_EVT_CONNECTION_REQUEST	This is macro HCI_EVT_CONNECTION_REQUEST.
	HCI_EVT_DATA_BUFFER_OVERFLOW	This is macro HCI_EVT_DATA_BUFFER_OVERFLOW.
	HCI_EVT_DISCONNECTION_COMPLETE	This is macro HCI_EVT_DISCONNECTION_COMPLETE.
	HCI_EVT_ENCRYPTION_CHANGE	This is macro HCI_EVT_ENCRYPTION_CHANGE.
	HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE	This is macro HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE.
	HCI_EVT_ENHANCED_FLUSH_COMPLETE	This is macro HCI_EVT_ENHANCED_FLUSH_COMPLETE.
	HCI_EVT_EXTENDED_INQUIRY_RESULT	This is macro HCI_EVT_EXTENDED_INQUIRY_RESULT.
	HCI_EVT_FIRST	This is macro HCI_EVT_FIRST.
	HCI_EVT_FLOW_SPECIFICATION_COMPLETE	This is macro HCI_EVT_FLOW_SPECIFICATION_COMPLETE.
	HCI_EVT_FLUSH_OCCURED	This is macro HCI_EVT_FLUSH_OCCURED.
	HCI_EVT_HARDWARE_ERROR	This is macro HCI_EVT_HARDWARE_ERROR.
	HCI_EVT_INQUIRY_COMPLETE	• addtogroup hci • @{ • @name Events
	HCI_EVT_INQUIRY_RESULT	This is macro HCI_EVT_INQUIRY_RESULT.
	HCI_EVT_INQUIRY_RESULT_WITH_RSSI	This is macro HCI_EVT_INQUIRY_RESULT_WITH_RSSI.
	HCI_EVT_IO_CAPABILITY_REQUEST	This is macro HCI_EVT_IO_CAPABILITY_REQUEST.
	HCI_EVT_IO_CAPABILITY_RESPONSE	This is macro HCI_EVT_IO_CAPABILITY_RESPONSE.
	HCI_EVT_KEYPRESS_NOTIFICATION	This is macro HCI_EVT_KEYPRESS_NOTIFICATION.
	HCI_EVT_LAST	This is macro HCI_EVT_LAST.
	HCI_EVT_LE_META_EVENT	This is macro HCI_EVT_LE_META_EVENT.
	HCI_EVT_LINK_IS_BUSY	This is macro HCI_EVT_LINK_IS_BUSY.
	HCI_EVT_LINK_IS_IDLE	This is macro HCI_EVT_LINK_IS_IDLE.
	HCI_EVT_LINK_KEY_NOTIFICATION	This is macro HCI_EVT_LINK_KEY_NOTIFICATION.
	HCI_EVT_LINK_KEY_REQUEST	This is macro HCI_EVT_LINK_KEY_REQUEST.
	HCI_EVT_LINK_SUPERVISION_TO_CHANGED	This is macro HCI_EVT_LINK_SUPERVISION_TO_CHANGED.
	HCI_EVT_LOOPBACK_COMMAND	This is macro HCI_EVT_LOOPBACK_COMMAND.
	HCI_EVT_MASTER_LINK_KEY_COMPLETE	This is macro HCI_EVT_MASTER_LINK_KEY_COMPLETE.
	HCI_EVT_MAX_SLOTS_CHANGE	This is macro HCI_EVT_MAX_SLOTS_CHANGE.
	HCI_EVT_MODE_CHANGE	This is macro HCI_EVT_MODE_CHANGE.
	HCI_EVT_NUM_OF_COMPLETED_PACKETS	This is macro HCI_EVT_NUM_OF_COMPLETED_PACKETS.
	HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE	This is macro HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE.
	HCI_EVT_PIN_CODE_REQUEST	This is macro HCI_EVT_PIN_CODE_REQUEST.
	HCI_EVT_QOS_SETUP_COMPLETE	This is macro HCI_EVT_QOS_SETUP_COMPLETE.
	HCI_EVT_QOS_VIOLATION	This is macro HCI_EVT_QOS_VIOLATION.
	HCI_EVT_READ_CLOCK_OFFSET_COMPLETE	This is macro HCI_EVT_READ_CLOCK_OFFSET_COMPLETE.
	HCI_EVT_READ_RMT_EXT_FEATURES_COMP	This is macro HCI_EVT_READ_RMT_EXT_FEATURES_COMP.

	HCI_EVT_READ_RMT_SUP_FEATURES_COMP	This is macro HCI_EVT_READ_RMT_SUP_FEATURES_COMP.
	HCI_EVT_READ_RMT_VERSION_INFO_COMP	This is macro HCI_EVT_READ_RMT_VERSION_INFO_COMP.
	HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE	This is macro HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE.
	HCI_EVT_REMOTE_OOB_DATA_REQUEST	This is macro HCI_EVT_REMOTE_OOB_DATA_REQUEST.
	HCI_EVT_RETURN_LINK_KEYS	This is macro HCI_EVT_RETURN_LINK_KEYS.
	HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF	This is macro HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF.
	HCI_EVT_ROLE_CHANGE	This is macro HCI_EVT_ROLE_CHANGE.
	HCI_EVT_SIMPLE_PAIRING_COMPLETE	This is macro HCI_EVT_SIMPLE_PAIRING_COMPLETE.
	HCI_EVT_SNIFF_SUBRATING	This is macro HCI_EVT_SNIFF_SUBRATING.
	HCI_EVT_SYNCH_CONNECTION_CHANGED	This is macro HCI_EVT_SYNCH_CONNECTION_CHANGED.
	HCI_EVT_SYNCH_CONNECTION_COMPLETE	This is macro HCI_EVT_SYNCH_CONNECTION_COMPLETE.
	HCI_EVT_USER_CONFIRMATION_REQUEST	This is macro HCI_EVT_USER_CONFIRMATION_REQUEST.
	HCI_EVT_USER_PASSKEY_NOTIFICATION	This is macro HCI_EVT_USER_PASSKEY_NOTIFICATION.
	HCI_EVT_USER_PASSKEY_REQUEST	This is macro HCI_EVT_USER_PASSKEY_REQUEST.
	HCI_EXIT_PARK_STATE	This is macro HCI_EXIT_PARK_STATE.
	HCI_EXIT_PERIODIC_INQUIRY_MODE	This is macro HCI_EXIT_PERIODIC_INQUIRY_MODE.
	HCI_EXIT_SNIFF_MODE	This is macro HCI_EXIT_SNIFF_MODE.
	HCI_FLOW_SPECIFICATION	This is macro HCI_FLOW_SPECIFICATION.
	HCI_FLUSH	This is macro HCI_FLUSH.
	HCI_H2C_BROADCAST_ACTIVE_SLAVE	This is macro HCI_H2C_BROADCAST_ACTIVE_SLAVE.
	HCI_H2C_BROADCAST_NO_BROADCATS	This is macro HCI_H2C_BROADCAST_NO_BROADCATS.
	HCI_H2C_BROADCAST_PARCKED_SLAVE	This is macro HCI_H2C_BROADCAST_PARCKED_SLAVE.
	HCI_H2C_BROADCAST_RESERVED	This is macro HCI_H2C_BROADCAST_RESERVED.
	HCI_HOLD_MODE	addtogroup hci @{ @name Link policy commands details The Link Policy Commands provide methods for the Host to affect how the Link Manager manages the piconet.
	HCI_HOST_BUFFER_SIZE	This is macro HCI_HOST_BUFFER_SIZE.
	HCI_HOST_NUM_OF_COMPLETED_PACKETS	This is macro HCI_HOST_NUM_OF_COMPLETED_PACKETS.
	HCI_INQUIRY	addtogroup hci @{ @name Link control commands details The Link Control commands allow a Controller to control connections to other BR/EDR Controllers.
	HCI_INQUIRY_CANCEL	This is macro HCI_INQUIRY_CANCEL.
	HCI_INQUIRY_MODE_EXTENDED	This is macro HCI_INQUIRY_MODE_EXTENDED.
	HCI_INQUIRY_MODE_STANDARD	This is macro HCI_INQUIRY_MODE_STANDARD.
	HCI_INQUIRY_MODE_WITH_RSSI	This is macro HCI_INQUIRY_MODE_WITH_RSSI.
	HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY	This is macro HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY.
	HCI_IO_CAPABILITY_REQUEST_REPLY	This is macro HCI_IO_CAPABILITY_REQUEST_REPLY.
	HCI_LE_ADD_DEVICE_TO_WHITE_LIST	This is macro HCI_LE_ADD_DEVICE_TO_WHITE_LIST.
	HCI_LE_CLEAR_WHITE_LIST	This is macro HCI_LE_CLEAR_WHITE_LIST.
	HCI_LE_CONNECTION_UPDATE	This is macro HCI_LE_CONNECTION_UPDATE.
	HCI_LE_CREATE_CONNECTION	This is macro HCI_LE_CREATE_CONNECTION.

	HCI_LE_CREATE_CONNECTION_CANCEL	This is macro HCI_LE_CREATE_CONNECTION_CANCEL.
	HCI_LE_ENCRYPT	This is macro HCI_LE_ENCRYPT.
	HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY	This is macro HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY.
	HCI_LE_LONG_TERM_KEY_REQUEST_REPLY	This is macro HCI_LE_LONG_TERM_KEY_REQUEST_REPLY.
	HCI_LE_MAX_AD_LEN	This is macro HCI_LE_MAX_AD_LEN.
	HCI_LE_RAND	This is macro HCI_LE_RAND.
	HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER	This is macro HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER.
	HCI_LE_READ_BUFFER_SIZE	This is macro HCI_LE_READ_BUFFER_SIZE.
	HCI_LE_READ_CHANNEL_MAP	This is macro HCI_LE_READ_CHANNEL_MAP.
	HCI_LE_READ_LOCAL_SUPPORTED_FEATURES	This is macro HCI_LE_READ_LOCAL_SUPPORTED_FEATURES.
	HCI_LE_READ_REMOTE_USED_FEATURES	This is macro HCI_LE_READ_REMOTE_USED_FEATURES.
	HCI_LE_READ_SUPPORTED_STATES	This is macro HCI_LE_READ_SUPPORTED_STATES.
	HCI_LE_READ_WHITE_LIST_SIZE	This is macro HCI_LE_READ_WHITE_LIST_SIZE.
	HCI_LE_RECEIVE_TEST	This is macro HCI_LE_RECEIVE_TEST.
	HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST	This is macro HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST.
	HCI_LE_SET_ADVERTISE_ENABLE	This is macro HCI_LE_SET_ADVERTISE_ENABLE.
	HCI_LE_SET_ADVERTISING_DATA	This is macro HCI_LE_SET_ADVERTISING_DATA.
	HCI_LE_SET_ADVERTISING_PARAMETERS	This is macro HCI_LE_SET_ADVERTISING_PARAMETERS.
	HCI_LE_SET_EVENT_MASK	addtogroup hci @{ @name LE controller commands details The LE Controller Commands provide access and control to various capabilities of the Bluetooth hardware, as well as methods for the Host to affect how the Link Layer manages the piconet, and controls connections.
	HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION	This is macro HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION.
	HCI_LE_SET_RANDOM_ADDRESS	This is macro HCI_LE_SET_RANDOM_ADDRESS.
	HCI_LE_SET_SCAN_ENABLE	This is macro HCI_LE_SET_SCAN_ENABLE.
	HCI_LE_SET_SCAN_PARAMETERS	This is macro HCI_LE_SET_SCAN_PARAMETERS.
	HCI_LE_SET_SCAN_RESPONSE_DATA	This is macro HCI_LE_SET_SCAN_RESPONSE_DATA.
	HCI_LE_START_ENCRYPTION	This is macro HCI_LE_START_ENCRYPTION.
	HCI_LE_TEST_END	This is macro HCI_LE_TEST_END.
	HCI_LE_TRANSMITTER_TEST	This is macro HCI_LE_TRANSMITTER_TEST.
	HCI_LINK_KEY_LEN	This is macro HCI_LINK_KEY_LEN.
	HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY	This is macro HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY.
	HCI_LINK_KEY_REQUEST_REPLY	This is macro HCI_LINK_KEY_REQUEST_REPLY.
	HCI_LINK_KEY_TYPE_COMBINATION	This is macro HCI_LINK_KEY_TYPE_COMBINATION.
	HCI_LINK_KEY_TYPE_LOCAL_UNIT	This is macro HCI_LINK_KEY_TYPE_LOCAL_UNIT.
	HCI_LINK_KEY_TYPE_REMOTE_UNIT	This is macro HCI_LINK_KEY_TYPE_REMOTE_UNIT.
	HCI_LINK_POLICY_ENABLE_ALL	This is macro HCI_LINK_POLICY_ENABLE_ALL.
	HCI_LINK_POLICY_ENABLE_HOLD_MODE	This is macro HCI_LINK_POLICY_ENABLE_HOLD_MODE.
	HCI_LINK_POLICY_ENABLE_PARK_STATE	This is macro HCI_LINK_POLICY_ENABLE_PARK_STATE.
	HCI_LINK_POLICY_ENABLE_ROLE_SWITCH	This is macro HCI_LINK_POLICY_ENABLE_ROLE_SWITCH.
	HCI_LINK_POLICY_ENABLE_SNIFF_MODE	This is macro HCI_LINK_POLICY_ENABLE_SNIFF_MODE.

	HCI_MASTER_LINK_KEY	This is macro HCI_MASTER_LINK_KEY.
	HCI_MAX_EVENT_PARAM_LEN	This is macro HCI_MAX_EVENT_PARAM_LEN.
	HCI_MAX_PARAM_LEN	This is macro HCI_MAX_PARAM_LEN.
	HCI_MAX_PIN_LENGTH	This is macro HCI_MAX_PIN_LENGTH.
	HCI_OPCODE	This is macro HCI_OPCODE.
	HCI_PACKET_BOUNDARY_COMPLETE	This is macro HCI_PACKET_BOUNDARY_COMPLETE.
	HCI_PACKET_BOUNDARY_CONTINUE	This is macro HCI_PACKET_BOUNDARY_CONTINUE.
	HCI_PACKET_BOUNDARY_FIRST	This is macro HCI_PACKET_BOUNDARY_FIRST.
	HCI_PACKET_BOUNDARY_FIRST_AUTO_FLUSH	This is macro HCI_PACKET_BOUNDARY_FIRST_AUTO_FLUSH.
	HCI_PACKET_BOUNDARY_FIRST_NO_AUTO_FLUSH	This is macro HCI_PACKET_BOUNDARY_FIRST_NO_AUTO_FLUSH .
	HCI_PACKET_TYPE_ACL_DATA	This is macro HCI_PACKET_TYPE_ACL_DATA.
	HCI_PACKET_TYPE_COMMAND	This is macro HCI_PACKET_TYPE_COMMAND.
	HCI_PACKET_TYPE_EVENT	This is macro HCI_PACKET_TYPE_EVENT.
	HCI_PACKET_TYPE_NONE	This is macro HCI_PACKET_TYPE_NONE.
	HCI_PACKET_TYPE_SCO_DATA	This is macro HCI_PACKET_TYPE_SCO_DATA.
	HCI_PAGE_SCAN_REPETITION_MODE_R0	This is macro HCI_PAGE_SCAN_REPETITION_MODE_R0.
	HCI_PAGE_SCAN_REPETITION_MODE_R1	This is macro HCI_PAGE_SCAN_REPETITION_MODE_R1.
	HCI_PAGE_SCAN_REPETITION_MODE_R2	This is macro HCI_PAGE_SCAN_REPETITION_MODE_R2.
	HCI_PARAM_LEN_BD_ADDR	This is macro HCI_PARAM_LEN_BD_ADDR.
	HCI_PARAM_LEN_BYTE	This is macro HCI_PARAM_LEN_BYTE.
	HCI_PARAM_LEN_DEV_CLASS	This is macro HCI_PARAM_LEN_DEV_CLASS.
	HCI_PARAM_LEN_HANDLE	This is macro HCI_PARAM_LEN_HANDLE.
	HCI_PARAM_LEN_INT	This is macro HCI_PARAM_LEN_INT.
	HCI_PARAM_LEN_LONG	This is macro HCI_PARAM_LEN_LONG.
	HCI_PARAM_TYPE_BD_ADDR	This is macro HCI_PARAM_TYPE_BD_ADDR.
	HCI_PARAM_TYPE_BYTE	This is macro HCI_PARAM_TYPE_BYTE.
	HCI_PARAM_TYPE_DEV_CLASS	This is macro HCI_PARAM_TYPE_DEV_CLASS.
	HCI_PARAM_TYPE_HANDLE	This is macro HCI_PARAM_TYPE_HANDLE.
	HCI_PARAM_TYPE_INT	This is macro HCI_PARAM_TYPE_INT.
	HCI_PARAM_TYPE_LONG	This is macro HCI_PARAM_TYPE_LONG.
	HCI_PARAM_TYPE_STRING	This is macro HCI_PARAM_TYPE_STRING.
	HCI_PARK_STATE	This is macro HCI_PARK_STATE.
	HCI_PERIODIC_INQUIRY_MODE	This is macro HCI_PERIODIC_INQUIRY_MODE.
	HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY	This is macro HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY.
	HCI_PIN_CODE_REQUEST_REPLY	This is macro HCI_PIN_CODE_REQUEST_REPLY.
	HCI_POWER_MODE_ACTIVE	This is macro HCI_POWER_MODE_ACTIVE.
	HCI_POWER_MODE_HOLD	This is macro HCI_POWER_MODE_HOLD.
	HCI_POWER_MODE_PARK	This is macro HCI_POWER_MODE_PARK.
	HCI_POWER_MODE_SNIFF	This is macro HCI_POWER_MODE_SNIFF.
	HCI_QOS_SETUP	This is macro HCI_QOS_SETUP.
	HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE	This is macro HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE.
	HCI_READ_AFH_CHANNEL_MAP	This is macro HCI_READ_AFH_CHANNEL_MAP.
	HCI_READ_AUTHENTICATION_ENABLE	This is macro HCI_READ_AUTHENTICATION_ENABLE.
	HCI_READ_AUTOMATIC_FLASH_TIMEOUT	This is macro HCI_READ_AUTOMATIC_FLASH_TIMEOUT.
	HCI_READ_BD_ADDR	This is macro HCI_READ_BD_ADDR.
	HCI_READ_BUFFER_SIZE	This is macro HCI_READ_BUFFER_SIZE.
	HCI_READ_CLASS_OF_DEVICE	This is macro HCI_READ_CLASS_OF_DEVICE.

	HCI_READ_CLOCK_COMMAND	This is macro HCI_READ_CLOCK_COMMAND.
	HCI_READ_CLOCK_OFFSET	This is macro HCI_READ_CLOCK_OFFSET.
	HCI_READ_CONNECTION_ACCEPT_TIMEOUT	This is macro HCI_READ_CONNECTION_ACCEPT_TIMEOUT.
	HCI_READ_CURRENT_IAC_LAP	This is macro HCI_READ_CURRENT_IAC_LAP.
	HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING	This is macro HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING.
	HCI_READ_DEFAULT_POLICY_SETTINGS	This is macro HCI_READ_DEFAULT_POLICY_SETTINGS.
	HCI_READ_ENCRYPTION_MODE	This is macro HCI_READ_ENCRYPTION_MODE.
	HCI_READ_EXTENDED_INQUIRY_RESPONSE	This is macro HCI_READ_EXTENDED_INQUIRY_RESPONSE.
	HCI_READ_FAILED_CONTACT_COUNTER	addtogroup hci @{ @name Status Parameters details The Controller modifies all status parameters. These parameters provide information about the current state of the Link Manager and Baseband in the BR/EDR Controller and the PAL in an AMP Controller. The host device cannot modify any of these parameters other than to reset certain specific parameters.
	HCI_READ_HOLD_MODE_ACTIVITY	This is macro HCI_READ_HOLD_MODE_ACTIVITY.
	HCI_READ_INQUIRY_MODE	This is macro HCI_READ_INQUIRY_MODE.
	HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL	This is macro HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL.
	HCI_READ_INQUIRY_SCAN_ACTIVITY	This is macro HCI_READ_INQUIRY_SCAN_ACTIVITY.
	HCI_READ_INQUIRY_SCAN_TYPE	This is macro HCI_READ_INQUIRY_SCAN_TYPE.
	HCI_READ_LE_HOST_SUPPORT	This is macro HCI_READ_LE_HOST_SUPPORT.
	HCI_READ_LINK_POLICY_SETTINGS	This is macro HCI_READ_LINK_POLICY_SETTINGS.
	HCI_READ_LINK_QUALITY	This is macro HCI_READ_LINK_QUALITY.
	HCI_READ_LINK_SUPERVISION_TIMEOUT	This is macro HCI_READ_LINK_SUPERVISION_TIMEOUT.
	HCI_READ_LMP_HANDLE	This is macro HCI_READ_LMP_HANDLE.
	HCI_READ_LOCAL_EXTENDED_FEATURES	This is macro HCI_READ_LOCAL_EXTENDED_FEATURES.
	HCI_READ_LOCAL_NAME	This is macro HCI_READ_LOCAL_NAME.
	HCI_READ_LOCAL_OOB_DATA	This is macro HCI_READ_LOCAL_OOB_DATA.
	HCI_READ_LOCAL_SUPPORTED_COMMANDS	This is macro HCI_READ_LOCAL_SUPPORTED_COMMANDS.
	HCI_READ_LOCAL_SUPPORTED_FEATURES	This is macro HCI_READ_LOCAL_SUPPORTED_FEATURES.
	HCI_READ_LOCAL_VERSION_INFORMATION	addtogroup hci @{ @name Informational Parameters details The Informational Parameters are fixed by the manufacturer of the Bluetooth hardware. These parameters provide information about the BR/EDR Controller and the capabilities of the Link Manager and Baseband in the BR/EDR Controller and PAL in the AMP Controller. The host device cannot modify any of these parameters.
	HCI_READ_LOOPBACK_MODE	addtogroup hci @{ @name Testing Commands details The Testing commands are used to provide the ability to test various functional capabilities of the Bluetooth hardware.
	HCI_READ_NUM_BROADCST_RETR	This is macro HCI_READ_NUM_BROADCST_RETR.
	HCI_READ_NUM_OF_SUPPORTED_IAC	This is macro HCI_READ_NUM_OF_SUPPORTED_IAC.
	HCI_READ_PAGE_SCAN_ACTIVITY	This is macro HCI_READ_PAGE_SCAN_ACTIVITY.
	HCI_READ_PAGE_SCAN_PERIOD_MODE	This is macro HCI_READ_PAGE_SCAN_PERIOD_MODE.
	HCI_READ_PAGE_SCAN_TYPE	This is macro HCI_READ_PAGE_SCAN_TYPE.

	HCI_READ_PAGE_TIMEOUT	This is macro HCI_READ_PAGE_TIMEOUT.
	HCI_READ_PIN_TYPE	This is macro HCI_READ_PIN_TYPE.
	HCI_READ_REFRESH_ENCRYPTION_KEY	This is macro HCI_READ_REFRESH_ENCRYPTION_KEY.
	HCI_READ_REMOTE_EXTENDED_FEATURES	This is macro HCI_READ_REMOTE_EXTENDED_FEATURES.
	HCI_READ_REMOTE_SUPPORTED_FEATURES	This is macro HCI_READ_REMOTE_SUPPORTED_FEATURES.
	HCI_READ_REMOTE_VERSION_INFORMATION	This is macro HCI_READ_REMOTE_VERSION_INFORMATION.
	HCI_READ_RSSI	This is macro HCI_READ_RSSI.
	HCI_READ_SCAN_ENABLE	This is macro HCI_READ_SCAN_ENABLE.
	HCI_READ_SIMPLE_PAIRING_MODE	This is macro HCI_READ_SIMPLE_PAIRING_MODE.
	HCI_READ_STORED_LINK_KEY	This is macro HCI_READ_STORED_LINK_KEY.
	HCI_READ_SYNC_FLOW_CONTROL_ENABLE	This is macro HCI_READ_SYNC_FLOW_CONTROL_ENABLE.
	HCI_READ_TRANSMIT_POWER_LEVEL	This is macro HCI_READ_TRANSMIT_POWER_LEVEL.
	HCI_READ_VOICE_SETTING	This is macro HCI_READ_VOICE_SETTING.
	HCI_REJECT_CONNECTION_REQUEST	This is macro HCI_REJECT_CONNECTION_REQUEST.
	HCI_REJECT_SYNCH_CONNECTION_REQUEST	This is macro HCI_REJECT_SYNCH_CONNECTION_REQUEST.
	HCI_REMOTE_NAME_REQUEST	This is macro HCI_REMOTE_NAME_REQUEST.
	HCI_REMOTE_NAME_REQUEST_CANCEL	This is macro HCI_REMOTE_NAME_REQUEST_CANCEL.
	HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY	This is macro HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY.
	HCI_REMOTE_OOB_DATA_REQUEST_REPLY	This is macro HCI_REMOTE_OOB_DATA_REQUEST_REPLY.
	HCI_RESET	This is macro HCI_RESET.
	HCI_RESET_FAILED_CONTACT_COUNTER	This is macro HCI_RESET_FAILED_CONTACT_COUNTER.
	HCI_ROLE_DISCOVERY	This is macro HCI_ROLE_DISCOVERY.
	HCI_ROLE_SWITCH_ALLOW	This is macro HCI_ROLE_SWITCH_ALLOW.
	HCI_ROLE_SWITCH_DISALLOW	This is macro HCI_ROLE_SWITCH_DISALLOW.
	HCI_SCAN_INQUIRY	Flags for HCI_READ_SCAN_ENABLE/HCI_WRITE_SCAN_ENABLE
	HCI_SCAN_PAGE	This is macro HCI_SCAN_PAGE.
	HCI_SCO_CONTENT_FORMAT_AIR_CODING_A_LAW	This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_A_LAW.
	HCI_SCO_CONTENT_FORMAT_AIR_CODING_CSVD	This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_CSVD.
	HCI_SCO_CONTENT_FORMAT_AIR_CODING_N_LAW	This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_N_LAW.
	HCI_SCO_CONTENT_FORMAT_AIR_CODING_TRANSPARENT	This is macro HCI_SCO_CONTENT_FORMAT_AIR_CODING_TRANSPARENT.
	HCI_SCO_CONTENT_FORMAT_INPUT_CODING_A_LAW	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_A_LAW.
	HCI_SCO_CONTENT_FORMAT_INPUT_CODING_LINEAR	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_LINEAR.
	HCI_SCO_CONTENT_FORMAT_INPUT_CODING_N_LAW	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_CODING_N_LAW.

	HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_1_COMPLEMENT	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_1_COMPLEMENT.
	HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_2_COMPLEMENT	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_2_COMPLEMENT.
	HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_SIGN_MAGNITUDE	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_SIGN_MAGNITUDE.
	HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_UNSIGNED	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_UNSIGNED.
	HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_16	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_16.
	HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_8	This is macro HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_8.
	HCI_SCO_DATA_HEADER_LEN	This is macro HCI_SCO_DATA_HEADER_LEN.
	HCI_SCO_MAX_DATA_LEN	This is macro HCI_SCO_MAX_DATA_LEN.
	HCI_SCO_MAX_LATENCY_DONTCARE	This is macro HCI_SCO_MAX_LATENCY_DONTCARE.
	HCI_SCO_PACKET_TYPE_ALL	This is macro HCI_SCO_PACKET_TYPE_ALL.
	HCI_SCO_PACKET_TYPE_EV3	This is macro HCI_SCO_PACKET_TYPE_EV3.
	HCI_SCO_PACKET_TYPE_EV4	This is macro HCI_SCO_PACKET_TYPE_EV4.
	HCI_SCO_PACKET_TYPE_EV5	This is macro HCI_SCO_PACKET_TYPE_EV5.
	HCI_SCO_PACKET_TYPE_HV1	This is macro HCI_SCO_PACKET_TYPE_HV1.
	HCI_SCO_PACKET_TYPE_HV2	This is macro HCI_SCO_PACKET_TYPE_HV2.
	HCI_SCO_PACKET_TYPE_HV3	This is macro HCI_SCO_PACKET_TYPE_HV3.
	HCI_SCO_PACKET_TYPE_NO_2_EV3	This is macro HCI_SCO_PACKET_TYPE_NO_2_EV3.
	HCI_SCO_PACKET_TYPE_NO_2_EV5	This is macro HCI_SCO_PACKET_TYPE_NO_2_EV5.
	HCI_SCO_PACKET_TYPE_NO_3_EV3	This is macro HCI_SCO_PACKET_TYPE_NO_3_EV3.
	HCI_SCO_PACKET_TYPE_NO_3_EV5	This is macro HCI_SCO_PACKET_TYPE_NO_3_EV5.
	HCI_SCO_RTX_EFFORT_DONTCARE	This is macro HCI_SCO_RTX_EFFORT_DONTCARE.
	HCI_SCO_RTX_EFFORT_NO_RETRANSMISSION	This is macro HCI_SCO_RTX_EFFORT_NO_RETRANSMISSION.
	HCI_SCO_RTX_EFFORT_OPTIMIZE_LINK_QUALITY	This is macro HCI_SCO_RTX_EFFORT_OPTIMIZE_LINK_QUALITY.
	HCI_SCO_RTX_EFFORT_OPTIMIZE_POWER_CONSUMPTION	This is macro HCI_SCO_RTX_EFFORT_OPTIMIZE_POWER_CONSUMPTION.
	HCI_SCO_RX_BANDWIDTH_DONTCARE	This is macro HCI_SCO_RX_BANDWIDTH_DONTCARE.
	HCI_SCO_TX_BANDWIDTH_DONTCARE	This is macro HCI_SCO_TX_BANDWIDTH_DONTCARE.
	HCI_SEND_DATA_STATUS_INTERRUPTED	This is macro HCI_SEND_DATA_STATUS_INTERRUPTED.
	HCI_SEND_DATA_STATUS_SUCCESS	This is macro HCI_SEND_DATA_STATUS_SUCCESS.
	HCI_SEND_KEY_PRESS_NOTIFICATION	This is macro HCI_SEND_KEY_PRESS_NOTIFICATION.
	HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION	This is macro HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION.
	HCI_SET_CONNECTION_ENCRYPTION	This is macro HCI_SET_CONNECTION_ENCRYPTION.
	HCI_SET_CTRL_TO_HOST_FLOW_CONTROL	This is macro HCI_SET_CTRL_TO_HOST_FLOW_CONTROL.
	HCI_SET_EVENT_FILTER	This is macro HCI_SET_EVENT_FILTER.
	HCI_SET_EVENT_MASK	addtogroup hci @{ @name Controller & Baseband commands details The Controller & Baseband Commands provide access and control to various capabilities of the Bluetooth hardware.
	HCI_SETUP_SYNCHRONOUS_CONNECTION	This is macro HCI_SETUP_SYNCHRONOUS_CONNECTION.

	HCI_SNIFF_MODE	This is macro HCI_SNIFF_MODE.
	HCI_SNIFF_SUBRATING	This is macro HCI_SNIFF_SUBRATING.
	HCI_SWITCH_ROLE	This is macro HCI_SWITCH_ROLE.
	HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY	This is macro HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY.
	HCI_USER_CONFIRMATION_REQUEST_REPLY	This is macro HCI_USER_CONFIRMATION_REQUEST_REPLY.
	HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY	This is macro HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY.
	HCI_USER_PASSKEY_REQUEST_REPLY	This is macro HCI_USER_PASSKEY_REQUEST_REPLY.
	HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE	This is macro HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE.
	HCI_WRITE_AUTHENTICATION_ENABLE	This is macro HCI_WRITE_AUTHENTICATION_ENABLE.
	HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT	This is macro HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT.
	HCI_WRITE_CLASS_OF_DEVICE	This is macro HCI_WRITE_CLASS_OF_DEVICE.
	HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT	This is macro HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT.
	HCI_WRITE_CURRENT_IAC_LAP	This is macro HCI_WRITE_CURRENT_IAC_LAP.
	HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING	This is macro HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING.
	HCI_WRITE_DEFAULT_POLICY_SETTINGS	This is macro HCI_WRITE_DEFAULT_POLICY_SETTINGS.
	HCI_WRITE_ENCRYPTION_MODE	This is macro HCI_WRITE_ENCRYPTION_MODE.
	HCI_WRITE_EXTENDED_INQUIRY_RESPONSE	This is macro HCI_WRITE_EXTENDED_INQUIRY_RESPONSE.
	HCI_WRITE_EXTENDED_INQUIRY_RESPONSE_PARAM_LEN	This is macro HCI_WRITE_EXTENDED_INQUIRY_RESPONSE_PARAM_LEN.
	HCI_WRITE_HOLD_MODE_ACTIVITY	This is macro HCI_WRITE_HOLD_MODE_ACTIVITY.
	HCI_WRITE_INQUIRY_MODE	This is macro HCI_WRITE_INQUIRY_MODE.
	HCI_WRITE_INQUIRY_SCAN_ACTIVITY	This is macro HCI_WRITE_INQUIRY_SCAN_ACTIVITY.
	HCI_WRITE_INQUIRY_SCAN_TYPE	This is macro HCI_WRITE_INQUIRY_SCAN_TYPE.
	HCI_WRITE_INQUIRY_TX_POWER_LEVEL	This is macro HCI_WRITE_INQUIRY_TX_POWER_LEVEL.
	HCI_WRITE_LE_HOST_SUPPORT	This is macro HCI_WRITE_LE_HOST_SUPPORT.
	HCI_WRITE_LINK_POLICY_SETTINGS	This is macro HCI_WRITE_LINK_POLICY_SETTINGS.
	HCI_WRITE_LINK_SUPERVISION_TIMEOUT	This is macro HCI_WRITE_LINK_SUPERVISION_TIMEOUT.
	HCI_WRITE_LOCAL_NAME	This is macro HCI_WRITE_LOCAL_NAME.
	HCI_WRITE_LOCAL_NAME_PARAM_LEN	This is macro HCI_WRITE_LOCAL_NAME_PARAM_LEN.
	HCI_WRITE_LOOPBACK_MODE	This is macro HCI_WRITE_LOOPBACK_MODE.
	HCI_WRITE_NUM_BROADCST_RETR	This is macro HCI_WRITE_NUM_BROADCST_RETR.
	HCI_WRITE_PAGE_SCAN_ACTIVITY	This is macro HCI_WRITE_PAGE_SCAN_ACTIVITY.
	HCI_WRITE_PAGE_SCAN_PERIOD_MODE	This is macro HCI_WRITE_PAGE_SCAN_PERIOD_MODE.
	HCI_WRITE_PAGE_SCAN_TYPE	This is macro HCI_WRITE_PAGE_SCAN_TYPE.
	HCI_WRITE_PAGE_TIMEOUT	This is macro HCI_WRITE_PAGE_TIMEOUT.
	HCI_WRITE_PIN_TYPE	This is macro HCI_WRITE_PIN_TYPE.
	HCI_WRITE_SCAN_ENABLE	This is macro HCI_WRITE_SCAN_ENABLE.
	HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE	This is macro HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE.
	HCI_WRITE_SIMPLE_PAIRING_MODE	This is macro HCI_WRITE_SIMPLE_PAIRING_MODE.
	HCI_WRITE_STORED_LINK_KEY	This is macro HCI_WRITE_STORED_LINK_KEY.

	HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE	This is macro HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE.
	HCI_WRITE_VOICE_SETTING	This is macro HCI_WRITE_VOICE_SETTING.
	OGF_CTRL_BASEBAND	This is macro OGF_CTRL_BASEBAND.
	OGF_INFORMATION	This is macro OGF_INFORMATION.
	OGF_LE	This is macro OGF_LE.
	OGF_LINK_CONTROL	This is macro OGF_LINK_CONTROL.
	OGF_LINK_POLICY	This is macro OGF_LINK_POLICY.
	OGF_STATUS	This is macro OGF_STATUS.
	OGF_TESTING	This is macro OGF_TESTING.
	OGF_VENDOR	This is macro OGF_VENDOR.

Structures

	Name	Description
	_bt_hci_command_s	This is type bt_hci_command_t.
	_bt_hci_data_s	This is type bt_hci_data_t.
	_bt_hci_event_s	This is type bt_hci_event_t.
	_bt_hci_inquiry_response_t	This is type bt_hci_inquiry_response_t.
	bt_hci_command_p	This is type bt_hci_command_p.
	bt_hci_command_t	This is type bt_hci_command_t.
	bt_hci_data_p	This is type bt_hci_data_p.
	bt_hci_data_t	This is type bt_hci_data_t.
	bt_hci_event_p	This is type bt_hci_event_p.
	bt_hci_event_t	This is type bt_hci_event_t.
	bt_hci_inquiry_response_t	This is type bt_hci_inquiry_response_t.

Types

	Name	Description
	bt_hci_cmd_callback_fp	This is type bt_hci_cmd_callback_fp.
	bt_hci_conn_state_p	This is type bt_hci_conn_state_p.
	bt_hci_conn_state_t	This is type bt_hci_conn_state_t.
	bt_hci_connect_callback_fp	brief HCI connect callback. ingroup hci details This typedef defines a type for the callback function that is called when HCI connect operation initiated by a call to bt_hci_connect() is complete. param status Operation status. c It is 0 if connection was successfully established. param pconn pointer to a structure representing the established connection. param param pointer to arbitrary data passed to the bt_hci_connect() function through its c param parameter.
	bt_hci_data_callback_fp	This is type bt_hci_data_callback_fp.
	bt_hci_disconnect_callback_fp	brief HCI disconnect callback. ingroup hci details This typedef defines a type for the callback function that is called when an HCI connection has been terminated. param status Operation status. c It is 0 if connection has been successfully terminated. param reason Reason for disconnection. param pconn pointer to a structure representing the connection. param param pointer to arbitrary data associated with an event listener.
	bt_hci_event_listener_fp	This is type bt_hci_event_listener_fp.
	bt_hci_hconn_p	This is type bt_hci_hconn_p.
	bt_hci_hconn_t	This is type bt_hci_hconn_t.
	bt_hci_inquiry_callback_fp	This is type bt_hci_inquiry_callback_fp.
	bt_hci_read_inquiry_mode_callback_fp	This is type bt_hci_read_inquiry_mode_callback_fp.
	bt_hci_read_inquiry_scan_activity_callback_fp	This is type bt_hci_read_inquiry_scan_activity_callback_fp.
	bt_hci_read_inquiry_scan_type_callback_fp	This is type bt_hci_read_inquiry_scan_type_callback_fp.
	bt_hci_read_page_scan_activity_callback_fp	This is type bt_hci_read_page_scan_activity_callback_fp.
	bt_hci_read_page_scan_period_mode_callback_fp	This is type bt_hci_read_page_scan_period_mode_callback_fp.
	bt_hci_read_page_scan_type_callback_fp	This is type bt_hci_read_page_scan_type_callback_fp.
	bt_hci_read_page_timeout_callback_fp	This is type bt_hci_read_page_timeout_callback_fp.
	bt_hci_request_remote_name_callback_fp	This is type bt_hci_request_remote_name_callback_fp.

	bt_hci_start_callback_fp	brief HCI initialization callback. ingroup hci details This typedef defines a function pointer type for the HCI initialization callback functions. Such a function must be passed to the bt_hci_start() function. param success Specifies whether HCI initialization succeeded or not.
	bt_hci_stop_callback_fp	brief HCI stop callback. ingroup hci details This typedef defines a function pointer type for the HCI stop callback functions. Such a function must be passed to the bt_hci_stop() function.
	pf_hci_sleep_callback	This is type pf_hci_sleep_callback.
	pf_hci_wakeup_callback	This is type pf_hci_wakeup_callback.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_cmd_buffer.h

Functions

	Name	Description
	_hci_allocate_cmd	This is function _hci_allocate_cmd.
	_hci_free_cmd	This is function _hci_free_cmd.
	_hci_init_cmd_buffers	This is function _hci_init_cmd_buffers.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_cmd_queue.h

Functions

	Name	Description
	_hci_init_cmd_queues	This is function _hci_init_cmd_queues.
	hci_cq_find_by_bdaddr_and_opcode	This is function hci_cq_find_by_bdaddr_and_opcode.
	hci_cq_find_by_hconn	This is function hci_cq_find_by_hconn.
	hci_cq_find_by_hconn_and_opcode	This is function hci_cq_find_by_hconn_and_opcode.
	hci_cq_find_by_opcode	This is function hci_cq_find_by_opcode.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_config.h

Macros

	Name	Description
	__HCI_CONFIG_H	This is macro __HCI_CONFIG_H.
	BT_ENABLE_SCO	This is macro BT_ENABLE_SCO.
	HCI_ALLOCATE_BUFFERS_FUNCTION	This is macro HCI_ALLOCATE_BUFFERS_FUNCTION.

	HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR	This is macro HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR.
	HCI_ALLOCATE_BUFFERS_VARS	This is macro HCI_ALLOCATE_BUFFERS_VARS.
	HCI_DECLARE_LE_CONN_STATES	This is macro HCI_DECLARE_LE_CONN_STATES.
	HCI_DECLARE_LE_CTRL_STATE	This is macro HCI_DECLARE_LE_CTRL_STATE.
	HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL	This is macro HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL.
	HCI_INIT_LE_CONN_STATES	This is macro HCI_INIT_LE_CONN_STATES.
	HCI_INIT_LE_CTRL_STATE	This is macro HCI_INIT_LE_CTRL_STATE.
	HCI_INIT_SCO_HANDLERS	This is macro HCI_INIT_SCO_HANDLERS.
	HCI_INIT_SSP_HANDLERS	This is macro HCI_INIT_SSP_HANDLERS.
	HCI_L2CAP_BUFFER_LEN	This is macro HCI_L2CAP_BUFFER_LEN.
	HCI_MAX_CONNECT_ATTEMPTS	This is macro HCI_MAX_CONNECT_ATTEMPTS.
	HCI_SIZEOF_LE_CONN_STATES	This is macro HCI_SIZEOF_LE_CONN_STATES.
	HCI_SIZEOF_LE_CTRL_STATE	This is macro HCI_SIZEOF_LE_CTRL_STATE.
	HCI_TX_BUFFER_LEN	This is macro HCI_TX_BUFFER_LEN.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_config_event_handlers.h

Macros

	Name	Description
	__HCI_CONFIG_EVENT_HANDLERS_H	This is macro __HCI_CONFIG_EVENT_HANDLERS_H.
	BT_HCI_EVT_AUTHENTICATION_COMPLETE_HANDLER	This is macro BT_HCI_EVT_AUTHENTICATION_COMPLETE_HANDLER.
	BT_HCI_EVT_CONNECTION_COMPLETE_HANDLER	This is macro BT_HCI_EVT_CONNECTION_COMPLETE_HANDLER.
	BT_HCI_EVT_CONNECTION_REQUEST_HANDLER	This is macro BT_HCI_EVT_CONNECTION_REQUEST_HANDLER.
	BT_HCI_EVT_ENCRYPTION_CHANGE_HANDLER	This is macro BT_HCI_EVT_ENCRYPTION_CHANGE_HANDLER.
	BT_HCI_EVT_EXTENDED_INQUIRY_RESULT_HANDLER	This is macro BT_HCI_EVT_EXTENDED_INQUIRY_RESULT_HANDLER.
	BT_HCI_EVT_INQUIRY_COMPLETE_HANDLER	This is macro BT_HCI_EVT_INQUIRY_COMPLETE_HANDLER.
	BT_HCI_EVT_INQUIRY_RESULT_HANDLER	This is macro BT_HCI_EVT_INQUIRY_RESULT_HANDLER.
	BT_HCI_EVT_INQUIRY_RESULT_WITH_RSSI_HANDLER	This is macro BT_HCI_EVT_INQUIRY_RESULT_WITH_RSSI_HANDLER.
	BT_HCI_EVT_LINK_KEY_NOTIFICATION_HANDLER	This is macro BT_HCI_EVT_LINK_KEY_NOTIFICATION_HANDLER.
	BT_HCI_EVT_LINK_KEY_REQUEST_HANDLER	This is macro BT_HCI_EVT_LINK_KEY_REQUEST_HANDLER.
	BT_HCI_EVT_MODE_CHANGE_HANDLER	This is macro BT_HCI_EVT_MODE_CHANGE_HANDLER.
	BT_HCI_EVT_PIN_CODE_REQUEST_HANDLER	This is macro BT_HCI_EVT_PIN_CODE_REQUEST_HANDLER.
	BT_HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE_HANDLER	This is macro BT_HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE_HANDLER.
	BT_HCI_EVT_ROLE_CHANGE_HANDLER	This is macro BT_HCI_EVT_ROLE_CHANGE_HANDLER.
	BT_HCI_EVT_SYNCH_CONNECTION_COMPLETE_HANDLER	This is macro BT_HCI_EVT_SYNCH_CONNECTION_COMPLETE_HANDLER.

	BT_LE_EVT_HANDLER	This is macro BT_LE_EVT_HANDLER.
	BT_SSP_EVT_HANDLER	This is macro BT_SSP_EVT_HANDLER.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_conn_state.h

Functions

	Name	Description
	bt_hci_register_listener	This is function bt_hci_register_listener.
	bt_hci_unregister_listener	This is function bt_hci_unregister_listener.
	hci_allocate_conn_state	This is function hci_allocate_conn_state.
	hci_get_conn_state	This is function hci_get_conn_state.
	hci_get_conn_state_by_bdaddr	This is function hci_get_conn_state_by_bdaddr.

Macros

	Name	Description
	HCI_CONN_ROLE_MASTER	This is macro HCI_CONN_ROLE_MASTER.
	HCI_CONN_ROLE_SLAVE	This is macro HCI_CONN_ROLE_SLAVE.
	HCI_CONN_STATE_AUTHENTICATING	This is macro HCI_CONN_STATE_AUTHENTICATING.
	HCI_CONN_STATE_CLOSED	This is macro HCI_CONN_STATE_CLOSED.
	HCI_CONN_STATE_OPEN	This is macro HCI_CONN_STATE_OPEN.
	HCI_CONN_TYPE_ACL	This is macro HCI_CONN_TYPE_ACL.
	HCI_CONN_TYPE_ESCO	This is macro HCI_CONN_TYPE_ESCO.
	HCI_CONN_TYPE_SCO	This is macro HCI_CONN_TYPE_SCO.
	HCI_LINK_TYPE_BD_EDR	This is macro HCI_LINK_TYPE_BD_EDR.
	HCI_LINK_TYPE_LE	This is macro HCI_LINK_TYPE_LE.

Structures

	Name	Description
	_bt_hci_conn_state_s	This is record _bt_hci_conn_state_s.
	_bt_hci_listener_t	This is record _bt_hci_listener_t.

Types

	Name	Description
	bt_hci_listener_t	This is type bt_hci_listener_t.
	bt_hci_sco_read_data_callback_fp	This is type bt_hci_sco_read_data_callback_fp.

Unions

	Name	Description
	_bt_hci_event_e	This is type bt_hci_event_e.
	bt_hci_event_e	This is type bt_hci_event_e.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_ctrl_state.h

Functions

	Name	Description
	bt_hci_ctrl_register_data_listener	This is function bt_hci_ctrl_register_data_listener.
	bt_hci_ctrl_register_listener	This is function bt_hci_ctrl_register_listener.
	bt_hci_ctrl_unregister_listener	This is function bt_hci_ctrl_unregister_listener.
	bt_hci_set_event_listener	listener is triggered by the following events: HCI_EVT_AUTHENTICATION_COMPLETE HCI_EVT_CONNECTION_COMPLETE HCI_EVT_DISCONNECTION_COMPLETE HCI_EVT_ROLE_CHANGE HCI_EVT_MODE_CHANGE

Macros

	Name	Description
	HCI_CTRL_LISTENER_ACL_DATA	This is macro HCI_CTRL_LISTENER_ACL_DATA.
	HCI_CTRL_LISTENER_EVENT	This is macro HCI_CTRL_LISTENER_EVENT.
	HCI_CTRL_LISTENER_SCO_DATA	This is macro HCI_CTRL_LISTENER_SCO_DATA.
	HCI_CTRL_STATE_CLOSED	This is macro HCI_CTRL_STATE_CLOSED.
	HCI_CTRL_STATE_CONNECTABLE	This is macro HCI_CTRL_STATE_CONNECTABLE.
	HCI_CTRL_STATE_DISCOVERABLE	This is macro HCI_CTRL_STATE_DISCOVERABLE.
	HCI_CTRL_STATE_INIT	This is macro HCI_CTRL_STATE_INIT.
	HCI_CTRL_STATE_READY	This is macro HCI_CTRL_STATE_READY.
	HCI_CTRL_STATE_SLEEP	This is macro HCI_CTRL_STATE_SLEEP.
	HCI_CTRL_STATE_WAKING_UP	This is macro HCI_CTRL_STATE_WAKING_UP.

Structures

	Name	Description
	_bt_hci_ctrl_listener_t	This is record _bt_hci_ctrl_listener_t.
	_bt_hci_ctrl_state_s	This is type bt_hci_ctrl_state_t.
	bt_hci_ctrl_state_t	This is type bt_hci_ctrl_state_t.

Types

	Name	Description
	bt_hci_cmd_listener_fp	This is type bt_hci_cmd_listener_fp.
	bt_hci_ctrl_listener_t	This is type bt_hci_ctrl_listener_t.
	bt_hci_data_listener_fp	This is type bt_hci_data_listener_fp.
	pf_l2cap_receive_callback	This is type pf_l2cap_receive_callback.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_data_buffer.h

Functions

	Name	Description
	bt_hci_alloc_data_buffer	This is function bt_hci_alloc_data_buffer.
	bt_hci_free_data_buffer	This is function bt_hci_free_data_buffer.
	bt_hci_init_data_buffers	This is function bt_hci_init_data_buffers.

Macros

	Name	Description
	HCI_DATA_BUFFER_STATE_FREE	This is macro HCI_DATA_BUFFER_STATE_FREE.

	HCI_DATA_BUFFER_STATE_USED	This is macro HCI_DATA_BUFFER_STATE_USED.
	HCI_MAX_DATA_BUFFERS	This is macro HCI_MAX_DATA_BUFFERS.

Structures

	Name	Description
	_bt_hci_data_buffer_s	This is type bt_hci_data_buffer_t.
	bt_hci_data_buffer_p	This is type bt_hci_data_buffer_p.
	bt_hci_data_buffer_t	This is type bt_hci_data_buffer_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_data_queue.h

Functions

	Name	Description
	bt_hci_init_data_queues	This is function bt_hci_init_data_queues.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_eir.h

Functions

	Name	Description
	bt_hci_allocate_write_eir_command	This is function bt_hci_allocate_write_eir_command.
	bt_hci_param_eir_add	This is function bt_hci_param_eir_add.
	bt_hci_param_eir_device_id_add	This is function bt_hci_param_eir_device_id_add.
	bt_hci_param_eir_local_name_add	This is function bt_hci_param_eir_local_name_add.
	bt_hci_param_eir_uuid128_add	This is function bt_hci_param_eir_uuid128_add.
	bt_hci_param_eir_uuid16_add	This is function bt_hci_param_eir_uuid16_add.
	bt_hci_param_eir_uuid32_add	This is function bt_hci_param_eir_uuid32_add.
	bt_hci_param_eir_vendor_add	This is function bt_hci_param_eir_vendor_add.
	bt_hci_param_tx_power_level_add	This is function bt_hci_param_tx_power_level_add.
	bt_hci_write_eir	This is function bt_hci_write_eir.

Macros

	Name	Description
	HCI_EIR_FEC_NOT_REQUIRED	This is macro HCI_EIR_FEC_NOT_REQUIRED.
	HCI_EIR_FEC_REQUIRED	This is macro HCI_EIR_FEC_REQUIRED.
	HCI_EIR_TYPE_3D_Information_Data	This is macro HCI_EIR_TYPE_3D_Information_Data.
	HCI_EIR_TYPE_ADVERTISING_INTERVAL	This is macro HCI_EIR_TYPE_ADVERTISING_INTERVAL.
	HCI_EIR_TYPE_APPEARANCE	This is macro HCI_EIR_TYPE_APPEARANCE.
	HCI_EIR_TYPE_DEVICE_ID	This is macro HCI_EIR_TYPE_DEVICE_ID.
	HCI_EIR_TYPE_FLAGS	This is macro HCI_EIR_TYPE_FLAGS.
	HCI_EIR_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS	This is macro HCI_EIR_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS.

	HCI_EIR_TYPE_LE_ROLE	This is macro HCI_EIR_TYPE_LE_ROLE.
	HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_CONFIRMATION_VALUE	This is macro HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_CONFIRMATION_VALUE.
	HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_RANDOM_VALUE	This is macro HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_RANDOM_VALUE.
	HCI_EIR_TYPE_LOCAL_NAME_COMPLETE	This is macro HCI_EIR_TYPE_LOCAL_NAME_COMPLETE.
	HCI_EIR_TYPE_LOCAL_NAME_SHORTENED	This is macro HCI_EIR_TYPE_LOCAL_NAME_SHORTENED.
	HCI_EIR_TYPE_MANUFACTURER_SPECIFIC	This is macro HCI_EIR_TYPE_MANUFACTURER_SPECIFIC.
	HCI_EIR_TYPE_OOB_COD	This is macro HCI_EIR_TYPE_OOB_COD.
	HCI_EIR_TYPE_OOB_HASH	This is macro HCI_EIR_TYPE_OOB_HASH.
	HCI_EIR_TYPE_OOB_RANDOMIZER	This is macro HCI_EIR_TYPE_OOB_RANDOMIZER.
	HCI_EIR_TYPE_PUBLIC_TARGET_ADDRESS	This is macro HCI_EIR_TYPE_PUBLIC_TARGET_ADDRESS.
	HCI_EIR_TYPE_RANDOM_TARGET_ADDRESS	This is macro HCI_EIR_TYPE_RANDOM_TARGET_ADDRESS.
	HCI_EIR_TYPE_SERVICE_DATA	This is macro HCI_EIR_TYPE_SERVICE_DATA.
	HCI_EIR_TYPE_SERVICE_DATA_UUID128	This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID128.
	HCI_EIR_TYPE_SERVICE_DATA_UUID16	This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID16.
	HCI_EIR_TYPE_SERVICE_DATA_UUID32	This is macro HCI_EIR_TYPE_SERVICE_DATA_UUID32.
	HCI_EIR_TYPE_SIMPLE_PAIRING_HASH_C_256	This is macro HCI_EIR_TYPE_SIMPLE_PAIRING_HASH_C_256.
	HCI_EIR_TYPE_SIMPLE_PAIRING_RANDOMIZER_R_256	This is macro HCI_EIR_TYPE_SIMPLE_PAIRING_RANDOMIZER_R_256.
	HCI_EIR_TYPE_SLAVE_CONN_INTERVAL_RANGE	This is macro HCI_EIR_TYPE_SLAVE_CONN_INTERVAL_RANGE.
	HCI_EIR_TYPE_SM_OOB_FLAGS	This is macro HCI_EIR_TYPE_SM_OOB_FLAGS.
	HCI_EIR_TYPE_SM_TK_VALUE	This is macro HCI_EIR_TYPE_SM_TK_VALUE.
	HCI_EIR_TYPE_SOLICITATION_UUID128_LIST	This is macro HCI_EIR_TYPE_SOLICITATION_UUID128_LIST.
	HCI_EIR_TYPE_SOLICITATION_UUID16_LIST	This is macro HCI_EIR_TYPE_SOLICITATION_UUID16_LIST.
	HCI_EIR_TYPE_SOLICITATION_UUID32_LIST	This is macro HCI_EIR_TYPE_SOLICITATION_UUID32_LIST.
	HCI_EIR_TYPE_TX_POWER_LEVEL	This is macro HCI_EIR_TYPE_TX_POWER_LEVEL.
	HCI_EIR_TYPE_UUID128_LIST_COMPLETE	This is macro HCI_EIR_TYPE_UUID128_LIST_COMPLETE.
	HCI_EIR_TYPE_UUID128_LIST_MORE_AVAILABLE	This is macro HCI_EIR_TYPE_UUID128_LIST_MORE_AVAILABLE.
	HCI_EIR_TYPE_UUID16_LIST_COMPLETE	This is macro HCI_EIR_TYPE_UUID16_LIST_COMPLETE.
	HCI_EIR_TYPE_UUID16_LIST_MORE_AVAILABLE	This is macro HCI_EIR_TYPE_UUID16_LIST_MORE_AVAILABLE.
	HCI_EIR_TYPE_UUID32_LIST_COMPLETE	This is macro HCI_EIR_TYPE_UUID32_LIST_COMPLETE.
	HCI_EIR_TYPE_UUID32_LIST_MORE_AVAILABLE	This is macro HCI_EIR_TYPE_UUID32_LIST_MORE_AVAILABLE.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with

a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_errors.h

Macros

	Name	Description
	HCI_ERR_ACL_CONN_ALREADY_EXISTS	This is macro HCI_ERR_ACL_CONN_ALREADY_EXISTS.
	HCI_ERR_AUTHENTICATION_FAILURE	This is macro HCI_ERR_AUTHENTICATION_FAILURE.
	HCI_ERR_CONN_REJECT_LIMITED_RESOURCES	This is macro HCI_ERR_CONN_REJECT_LIMITED_RESOURCES.
	HCI_ERR_CONNECTION_TIMEOUT	This is macro HCI_ERR_CONNECTION_TIMEOUT.
	HCI_ERR_INVALID_PARAMETERS	This is macro HCI_ERR_INVALID_PARAMETERS.
	HCI_ERR_MEMORY_CAPACITY_EXCEEDED	This is macro HCI_ERR_MEMORY_CAPACITY_EXCEEDED.
	HCI_ERR_PAIRING_NOT_ALLOWED	This is macro HCI_ERR_PAIRING_NOT_ALLOWED.
	HCI_ERR_PIN_OR_KEY_MISSING	This is macro HCI_ERR_PIN_OR_KEY_MISSING.
	HCI_ERR_SCO_CONN_LIMIT_EXCEEDED	This is macro HCI_ERR_SCO_CONN_LIMIT_EXCEEDED.
	HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED	This is macro HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED.
	HCI_ERR_SUCCESS	• addtogroup hci • @{ • @name Errors
	HCI_ERR_UNSPECIFIED	This is macro HCI_ERR_UNSPECIFIED.
	HCI_SUCCESS	This is macro HCI_SUCCESS.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_evt_handlers.h

Functions

	Name	Description
	bt_hci_evt_authentication_complete_handler	This is function bt_hci_evt_authentication_complete_handler.
	bt_hci_evt_change_conn_link_complete_handler	This is function bt_hci_evt_change_conn_link_complete_handler.
	bt_hci_evt_command_complete_handler	This is function bt_hci_evt_command_complete_handler.
	bt_hci_evt_command_status_handler	This is function bt_hci_evt_command_status_handler.
	bt_hci_evt_conn_packet_type_changed_handler	This is function bt_hci_evt_conn_packet_type_changed_handler.
	bt_hci_evt_connection_complete_handler	This is function bt_hci_evt_connection_complete_handler.
	bt_hci_evt_connection_request_handler	This is function bt_hci_evt_connection_request_handler.
	bt_hci_evt_data_buffer_overflow_handler	This is function bt_hci_evt_data_buffer_overflow_handler.
	bt_hci_evt_default_handler	This is function bt_hci_evt_default_handler.
	bt_hci_evt_disconnection_complete_handler	This is function bt_hci_evt_disconnection_complete_handler.
	bt_hci_evt_encryption_change_handler	This is function bt_hci_evt_encryption_change_handler.
	bt_hci_evt_extended_inquiry_result_handler	This is function bt_hci_evt_extended_inquiry_result_handler.
	bt_hci_evt_flow_specification_complete_handler	This is function bt_hci_evt_flow_specification_complete_handler.
	bt_hci_evt_flush_occured_handler	This is function bt_hci_evt_flush_occured_handler.
	bt_hci_evt_hardware_error_handler	This is function bt_hci_evt_hardware_error_handler.
	bt_hci_evt_inquiry_complete_handler	This is function bt_hci_evt_inquiry_complete_handler.
	bt_hci_evt_inquiry_result_handler	This is function bt_hci_evt_inquiry_result_handler.
	bt_hci_evt_inquiry_result_with_rssi_handler	This is function bt_hci_evt_inquiry_result_with_rssi_handler.
	bt_hci_evt_link_key_notification_handler	This is function bt_hci_evt_link_key_notification_handler.
	bt_hci_evt_link_key_request_handler	This is function bt_hci_evt_link_key_request_handler.
	bt_hci_evt_loopback_command_handler	This is function bt_hci_evt_loopback_command_handler.
	bt_hci_evt_master_link_key_complete_handler	This is function bt_hci_evt_master_link_key_complete_handler.

	bt_hci_evt_max_slots_change_handler	This is function bt_hci_evt_max_slots_change_handler.
	bt_hci_evt_mode_change_handler	This is function bt_hci_evt_mode_change_handler.
	bt_hci_evt_num_of_completed_packets_handler	This is function bt_hci_evt_num_of_completed_packets_handler.
	bt_hci_evt_page_scan_repet_mode_change_handler	This is function bt_hci_evt_page_scan_repet_mode_change_handler.
	bt_hci_evt_pin_code_request_handler	This is function bt_hci_evt_pin_code_request_handler.
	bt_hci_evt_qos_setup_complete_handler	This is function bt_hci_evt_qos_setup_complete_handler.
	bt_hci_evt_qos_violation_handler	This is function bt_hci_evt_qos_violation_handler.
	bt_hci_evt_read_clock_offset_complete_handler	This is function bt_hci_evt_read_clock_offset_complete_handler.
	bt_hci_evt_read_rmt_ext_features_comp_handler	This is function bt_hci_evt_read_rmt_ext_features_comp_handler.
	bt_hci_evt_read_rmt_sup_features_comp_handler	This is function bt_hci_evt_read_rmt_sup_features_comp_handler.
	bt_hci_evt_read_rmt_version_info_comp_handler	This is function bt_hci_evt_read_rmt_version_info_comp_handler.
	bt_hci_evt_remote_name_request_complete_handler	This is function bt_hci_evt_remote_name_request_complete_handler.
	bt_hci_evt_return_link_keys_handler	This is function bt_hci_evt_return_link_keys_handler.
	bt_hci_evt_role_change_handler	This is function bt_hci_evt_role_change_handler.
	bt_hci_evt_synch_connection_changed_handler	This is function bt_hci_evt_synch_connection_changed_handler.
	bt_hci_evt_synch_connection_complete_handler	This is function bt_hci_evt_synch_connection_complete_handler.
	bt_hci_set_vendor_specific_event_handler	This is function bt_hci_set_vendor_specific_event_handler.

Structures

	Name	Description
	_bt_hci_evt_authentication_complete_t	This is type bt_hci_evt_authentication_complete_t.
	_bt_hci_evt_command_complete_t	This is type bt_hci_evt_command_complete_t.
	_bt_hci_evt_command_status_t	This is type bt_hci_evt_command_status_t.
	_bt_hci_evt_connection_complete_t	This is type bt_hci_evt_connection_complete_t.
	_bt_hci_evt_connection_request_t	This is type bt_hci_evt_connection_request_t.
	_bt_hci_evt_disconnection_complete_t	This is type bt_hci_evt_disconnection_complete_t.
	_bt_hci_evt_encryption_change_s	This is type bt_hci_evt_encryption_change_t.
	_bt_hci_evt_mode_change_t	This is type bt_hci_evt_mode_change_t.
	_bt_hci_evt_role_change_t	This is type bt_hci_evt_role_change_t.
	bt_hci_evt_authentication_complete_t	This is type bt_hci_evt_authentication_complete_t.
	bt_hci_evt_command_complete_t	This is type bt_hci_evt_command_complete_t.
	bt_hci_evt_command_status_t	This is type bt_hci_evt_command_status_t.
	bt_hci_evt_connection_complete_t	This is type bt_hci_evt_connection_complete_t.
	bt_hci_evt_connection_request_t	This is type bt_hci_evt_connection_request_t.
	bt_hci_evt_disconnection_complete_t	This is type bt_hci_evt_disconnection_complete_t.
	bt_hci_evt_encryption_change_t	This is type bt_hci_evt_encryption_change_t.
	bt_hci_evt_mode_change_t	This is type bt_hci_evt_mode_change_t.
	bt_hci_evt_role_change_t	This is type bt_hci_evt_role_change_t.

Types

	Name	Description
	bt_hci_event_handler_ex_fp	This is type bt_hci_event_handler_ex_fp.
	bt_hci_event_handler_fp	This is type bt_hci_event_handler_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_le.h

Functions

	Name	Description
	bt_hci_le_add_device_to_white_list	This is function bt_hci_le_add_device_to_white_list.
	bt_hci_le_advertising_add	This is function bt_hci_le_advertising_add.
	bt_hci_le_advertising_add_local_name	This is function bt_hci_le_advertising_add_local_name.
	bt_hci_le_advertising_device_id_add	This is function bt_hci_le_advertising_device_id_add.
	bt_hci_le_advertising_flags_add	This is function bt_hci_le_advertising_flags_add.
	bt_hci_le_advertising_get_local_name	This is function bt_hci_le_advertising_get_local_name.
	bt_hci_le_advertising_tx_power_level_add	This is function bt_hci_le_advertising_tx_power_level_add.
	bt_hci_le_advertising_uuid128_add	This is function bt_hci_le_advertising_uuid128_add.
	bt_hci_le_advertising_uuid16_add	This is function bt_hci_le_advertising_uuid16_add.
	bt_hci_le_advertising_uuid32_add	This is function bt_hci_le_advertising_uuid32_add.
	bt_hci_le_advertising_vendor_add	This is function bt_hci_le_advertising_vendor_add.
	bt_hci_le_allocate_set_advertising_data_command	This is function bt_hci_le_allocate_set_advertising_data_command.
	bt_hci_le_allocate_set_scan_response_data_command	This is function bt_hci_le_allocate_set_scan_response_data_command.
	bt_hci_le_cancel_connect_ex	This is function bt_hci_le_cancel_connect_ex.
	bt_hci_le_cancel_find_devices	This is function bt_hci_le_cancel_find_devices.
	bt_hci_le_clear_white_list	This is function bt_hci_le_clear_white_list.
	bt_hci_le_connect_ex	This is function bt_hci_le_connect_ex.
	bt_hci_le_enable	This is function bt_hci_le_enable.
	bt_hci_le_encrypt	This is function bt_hci_le_encrypt.
	bt_hci_le_find_devices	This is function bt_hci_le_find_devices.
	bt_hci_le_get_connect_parameters	This is function bt_hci_le_get_connect_parameters.
	bt_hci_le_ibeacon_add	This is function bt_hci_le_ibeacon_add.
	bt_hci_le_ltk_negative_reply	This is function bt_hci_le_ltk_negative_reply.
	bt_hci_le_ltk_reply	This is function bt_hci_le_ltk_reply.
	bt_hci_le_rand	This is function bt_hci_le_rand.
	bt_hci_le_read_channel_map	This is function bt_hci_le_read_channel_map.
	bt_hci_le_read_remote_used_features	This is function bt_hci_le_read_remote_used_features.
	bt_hci_le_read_support	This is function bt_hci_le_read_support.
	bt_hci_le_read_white_list_size	This is function bt_hci_le_read_white_list_size.
	bt_hci_le_receiver_test	This is function bt_hci_le_receiver_test.
	bt_hci_le_remove_device_from_white_list	This is function bt_hci_le_remove_device_from_white_list.
	bt_hci_le_set_adevrtising_enable_ex	This is function bt_hci_le_set_adevrtising_enable_ex.
	bt_hci_le_set_advertising_parameters	This is function bt_hci_le_set_advertising_parameters.
	bt_hci_le_set_connect_parameters	This is function bt_hci_le_set_connect_parameters.
	bt_hci_le_set_host_channel_classification	This is function bt_hci_le_set_host_channel_classification.
	bt_hci_le_set_random_address	This is function bt_hci_le_set_random_address.
	bt_hci_le_set_scan_enable	This is function bt_hci_le_set_scan_enable.
	bt_hci_le_set_scan_parameters	This is function bt_hci_le_set_scan_parameters.
	bt_hci_le_start_encryption	This is function bt_hci_le_start_encryption.
	bt_hci_le_supported	This is function bt_hci_le_supported.
	bt_hci_le_test_end	This is function bt_hci_le_test_end.
	bt_hci_le_transmitter_test	This is function bt_hci_le_transmitter_test.
	bt_hci_le_update_connection	This is function bt_hci_le_update_connection.
	bt_hci_le_write_support	This is function bt_hci_le_write_support.

Macros

	Name	Description
	bt_hci_le_cancel_connect	This is macro bt_hci_le_cancel_connect.
	bt_hci_le_set_adevrtising_enable	This is macro bt_hci_le_set_adevrtising_enable.

	HCI_LE_ADDRESS_TYPE_PUBLIC	This is macro HCI_LE_ADDRESS_TYPE_PUBLIC.
	HCI_LE_ADDRESS_TYPE_RANDOM	This is macro HCI_LE_ADDRESS_TYPE_RANDOM.
	HCI_LE_ADV_CHANNEL_MAP_ENABLE_37	This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_37.
	HCI_LE_ADV_CHANNEL_MAP_ENABLE_38	This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_38.
	HCI_LE_ADV_CHANNEL_MAP_ENABLE_39	This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_39.
	HCI_LE_ADV_CHANNEL_MAP_ENABLE_ALL	This is macro HCI_LE_ADV_CHANNEL_MAP_ENABLE_ALL.
	HCI_LE_ADV_CHANNEL_MAP_RESERVED	This is macro HCI_LE_ADV_CHANNEL_MAP_RESERVED.
	HCI_LE_ADV_TYPE_CONN_UNDIRECT	Connectable undirected advertising (ADV_IND) (default)
	HCI_LE_ADV_TYPE_DIRECT_HIGH	Connectable high duty cycle directed advertising (ADV_DIRECT_IND, high duty cycle)
	HCI_LE_ADV_TYPE_DIRECT_LOW	Connectable low duty cycle directed advertising (ADV_DIRECT_IND, low duty cycle)
	HCI_LE_ADV_TYPE_NONCONN	Non connectable undirected advertising (ADV_NONCONN_IND)
	HCI_LE_ADV_TYPE_SCAN	Scannable undirected advertising (ADV_SCAN_IND)
	HCI_LE_ADVERTISING_FLAG_BREDR_NOT_SUPPORTED	This is macro HCI_LE_ADVERTISING_FLAG_BREDR_NOT_SUPPORTED.
	HCI_LE_ADVERTISING_FLAG_GENERAL_DISCOVERABLE_MODE	This is macro HCI_LE_ADVERTISING_FLAG_GENERAL_DISCOVERABLE_MODE.
	HCI_LE_ADVERTISING_FLAG_LIMITED_DISCOVERABLE_MODE	This is macro HCI_LE_ADVERTISING_FLAG_LIMITED_DISCOVERABLE_MODE.
	HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_CONTROLLER	This is macro HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_CONTROLLER.
	HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_HOST	This is macro HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_HOST.
	HCI_LE_AVERTISING_DISABLED	This is macro HCI_LE_AVERTISING_DISABLED.
	HCI_LE_AVERTISING_ENABLED	This is macro HCI_LE_AVERTISING_ENABLED.
	HCI_LE_DISABLED	This is macro HCI_LE_DISABLED.
	HCI_LE_DUPLICATE_FILTERING_DISABLED	This is macro HCI_LE_DUPLICATE_FILTERING_DISABLED.
	HCI_LE_DUPLICATE_FILTERING_ENABLED	This is macro HCI_LE_DUPLICATE_FILTERING_ENABLED.
	HCI_LE_ENABLED	This is macro HCI_LE_ENABLED.
	HCI_LE_EVT_ADVERTISING_REPORT	This is macro HCI_LE_EVT_ADVERTISING_REPORT.
	HCI_LE_EVT_CONNECTION_COMPLETE	This is macro HCI_LE_EVT_CONNECTION_COMPLETE.
	HCI_LE_EVT_CONNECTION_UPDATE_COMPLETE	This is macro HCI_LE_EVT_CONNECTION_UPDATE_COMPLETE.
	HCI_LE_EVT_LONG_TERM_KEY_REQUEST	This is macro HCI_LE_EVT_LONG_TERM_KEY_REQUEST.
	HCI_LE_EVT_READ_REMOTE_USED_FEATURES_COMPLETE	This is macro HCI_LE_EVT_READ_REMOTE_USED_FEATURES_COMPLETE.
	HCI_LE_FILTER_POLICY_NOT_USED	White list is not used to determine which advertiser to connect to. Peer_Address_Type and Peer_Address shall be used.

	HCI_LE_FILTER_POLICY_WHITE_LIST	White list is used to determine which advertiser to connect to. Peer_Address_Type and Peer_Address shall be ignored
	HCI_LE_FLAG_SHARED_ACL_BUFFERS	This is macro HCI_LE_FLAG_SHARED_ACL_BUFFERS.
	HCI_LE_FLAG_SIMULTANEOUS_LE_BREDR	This is macro HCI_LE_FLAG_SIMULTANEOUS_LE_BREDR.
	HCI_LE_RANDOM_ADDRESS_TYPE_NON_RESOLVABLE	This is macro HCI_LE_RANDOM_ADDRESS_TYPE_NON_RESOLVABLE.
	HCI_LE_RANDOM_ADDRESS_TYPE_RESOLVABLE	This is macro HCI_LE_RANDOM_ADDRESS_TYPE_RESOLVABLE.
	HCI_LE_RANDOM_ADDRESS_TYPE_STATIC	This is macro HCI_LE_RANDOM_ADDRESS_TYPE_STATIC.
	HCI_LE_SCAN_DISABLED	This is macro HCI_LE_SCAN_DISABLED.
	HCI_LE_SCAN_ENABLED	This is macro HCI_LE_SCAN_ENABLED.
	HCI_LE_SCAN_EVT_CANCEL_FAILED	This is macro HCI_LE_SCAN_EVT_CANCEL_FAILED.
	HCI_LE_SCAN_EVT_CANCELLED	This is macro HCI_LE_SCAN_EVT_CANCELLED.
	HCI_LE_SCAN_EVT_DEVICE_FOUND	This is macro HCI_LE_SCAN_EVT_DEVICE_FOUND.
	HCI_LE_SCAN_EVT_START_FAILED	This is macro HCI_LE_SCAN_EVT_START_FAILED.
	HCI_LE_SCAN_EVT_STARTED	This is macro HCI_LE_SCAN_EVT_STARTED.
	HCI_LE_SCAN_FILTER_POLICY_ALL	This is macro HCI_LE_SCAN_FILTER_POLICY_ALL.
	HCI_LE_SCAN_FILTER_POLICY_WHITE_LIST	This is macro HCI_LE_SCAN_FILTER_POLICY_WHITE_LIST.
	HCI_LE_SCAN_TYPE_ACTIVE	This is macro HCI_LE_SCAN_TYPE_ACTIVE.
	HCI_LE_SCAN_TYPE_PASSIVE	This is macro HCI_LE_SCAN_TYPE_PASSIVE.
	HCI_LE_SIMULTANEOUS_DISABLED	This is macro HCI_LE_SIMULTANEOUS_DISABLED.
	HCI_LE_SIMULTANEOUS_ENABLED	This is macro HCI_LE_SIMULTANEOUS_ENABLED.

Structures

	Name	Description
	_bt_hci_le_advertising_report_t	This is record _bt_hci_le_advertising_report_t.
	_bt_hci_le_conn_state_t	This is type bt_hci_le_conn_state_t.
	_bt_hci_le_connect_parameters_t	This is type bt_hci_le_connect_parameters_t.
	_bt_hci_le_ctrl_state_t	This is type bt_hci_le_ctrl_state_t.
	_bt_hci_le_evt_connection_updated_t	This is type bt_hci_le_evt_connection_updated_t.
	_bt_hci_le_evt_read_remote_used_features_completed_t	This is type bt_hci_le_evt_read_remote_used_features_completed_t.
	_bt_hci_le_evt_read_support_params_t	This is type bt_hci_le_evt_read_support_params_t.
	bt_hci_le_conn_state_t	This is type bt_hci_le_conn_state_t.
	bt_hci_le_connect_parameters_t	This is type bt_hci_le_connect_parameters_t.
	bt_hci_le_ctrl_state_t	This is type bt_hci_le_ctrl_state_t.
	bt_hci_le_evt_connection_updated_t	This is type bt_hci_le_evt_connection_updated_t.
	bt_hci_le_evt_read_remote_used_features_completed_t	This is type bt_hci_le_evt_read_remote_used_features_completed_t.
	bt_hci_le_evt_read_support_params_t	This is type bt_hci_le_evt_read_support_params_t.

Types

	Name	Description
	bt_hci_le_advertising_report_t	This is type bt_hci_le_advertising_report_t.
	bt_hci_le_scan_callback_fp	This is type bt_hci_le_scan_callback_fp.
	bt_le_evt_handler	This is type bt_le_evt_handler.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

hci_linkkey_buffer.h

Functions

	Name	Description
	bt_hci_init_linkkey_buffers	This is function bt_hci_init_linkkey_buffers.

Structures

	Name	Description
	_bt_hci_link_key_s	This is type bt_hci_link_key_t.
	bt_hci_link_key_t	This is type bt_hci_link_key_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_private.h

Functions

	Name	Description
	_bt_hci_ctrl_notify_data_listeners	This is function _bt_hci_ctrl_notify_data_listeners.
	_bt_hci_ctrl_notify_listeners	This is function _bt_hci_ctrl_notify_listeners.
	_bt_hci_get_tick_count	This is function _bt_hci_get_tick_count.
	_bt_hci_init_timer	From hci_timer.c
	_bt_hci_init_transport	This is function _bt_hci_init_transport.
	_bt_hci_le_command_complete_handler	This is function _bt_hci_le_command_complete_handler.
	_bt_hci_notify_listeners	This is function _bt_hci_notify_listeners.
	_bt_hci_set_init_flags	This is function _bt_hci_set_init_flags.
	_bt_le_evt_handler	This is function _bt_le_evt_handler.
	_hci_allocate_buffers	Defined by OEM through library configuration
	_hci_receive_start	From hci_receive.c
	_hci_recv_sco_data_packet	From hci_sco.c
	_hci_send_commands_from_queue	From hci_send.c
	_hci_send_data	This is function _hci_send_data.
	_hci_send_data_fragment	This is function _hci_send_data_fragment.
	_hci_send_data_from_queue	This is function _hci_send_data_from_queue.
	bt_hci_le_init	This is function bt_hci_le_init.
	hci_check_aux_info	From hci_aux_info.c

Macros

	Name	Description
	HCI_INIT_FLAG_IGNORE_TOTAL_NUM_ACL_DATA_PACKETS	The total number of ACL data packets read from the controller will be ignored. The stack will assume that he controller can accept only 1 ACL packet and sends HCI_EVT_NUM_OF_COMPLETED_PACKETS for each packet it has processed. This seems to be for controller working over SDIO (at least for Marvell's 88W8777).
	HCI_INIT_FLAG_SEND_HCI_RESET	HCI reset will be sent when bt_hci_init is called

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a

license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_signal.h

Functions

	Name	Description
	_bt_hci_init_signal	This is function _bt_hci_init_signal.
	_bt_hci_set_signal	This is function _bt_hci_set_signal.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_transport.h

Functions

	Name	Description
	bt_hci_transport_recv_packet	This is function bt_hci_transport_recv_packet.
	bt_hci_transport_send_cmd	deprecated
	bt_hci_transport_send_data	This is function bt_hci_transport_send_data.
	bt_hci_transport_send_packet	This is function bt_hci_transport_send_packet.
	bt_hci_transport_set_transport	This is function bt_hci_transport_set_transport.

Macros

	Name	Description
	HCI_TRANSPORT_HEADER_LEN	This is macro HCI_TRANSPORT_HEADER_LEN.

Structures

	Name	Description
	_hci_transport_t	This is type hci_transport_t.
	hci_transport_t	This is type hci_transport_t.

Types

	Name	Description
	bt_hci_transport_recv_packet_callback_fp	This is type bt_hci_transport_recv_packet_callback_fp.
	bt_hci_transport_send_packet_callback_fp	This is type bt_hci_transport_send_packet_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hci_utils.h

Functions

	Name	Description
	is_bdaddr_command	This is function is_bdaddr_command.
	is_hconn_command	This is function is_hconn_command.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hcitr_3wire.h

Functions

	Name	Description
	bt_hcitr_3wire_cancel_recv_packet	This is function bt_hcitr_3wire_cancel_recv_packet.
	bt_hcitr_3wire_init_ex	This is function bt_hcitr_3wire_init_ex.
	bt_hcitr_3wire_reset_ex	This is function bt_hcitr_3wire_reset_ex.
	bt_hcitr_3wire_start	This is function bt_hcitr_3wire_start.

Macros

	Name	Description
	bt_hcitr_3wire_init	This is macro bt_hcitr_3wire_init.
	bt_hcitr_3wire_reset	This is macro bt_hcitr_3wire_reset.
	HCITR_3WIRE_DEFAULT_ACK_TIMEOUT	This is macro HCITR_3WIRE_DEFAULT_ACK_TIMEOUT.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hcitr_bcsp.h

Functions

	Name	Description
	bt_hcitr_bcsp_init_ex	This is function bt_hcitr_bcsp_init_ex.
	bt_hcitr_bcsp_reset_ex	This is function bt_hcitr_bcsp_reset_ex.
	bt_hcitr_bcsp_start	This is function bt_hcitr_bcsp_start.

Macros

	Name	Description
	bt_hcitr_bcsp_init	This is macro bt_hcitr_bcsp_init.
	bt_hcitr_bcsp_reset	This is macro bt_hcitr_bcsp_reset.
	HCITR_BCSP_DEFAULT_ACK_TIMEOUT	This is macro HCITR_BCSP_DEFAULT_ACK_TIMEOUT.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hcitr_packet.h

Functions

	Name	Description
	bt_hcitr_packet_init	This is function bt_hcitr_packet_init.

	bt_hcitr_packet_reset	This is function bt_hcitr_packet_reset.
	bt_hcitr_packet_start	This is function bt_hcitr_packet_start.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hcitr_tih4.h

Enumerations

	Name	Description
	_hcitr_tih4_power_event_e	This is type bt_hcitr_tih4_power_event_e.
	bt_hcitr_tih4_power_event_e	This is type bt_hcitr_tih4_power_event_e.

Functions

	Name	Description
	bt_hcitr_tih4_init	This is function bt_hcitr_tih4_init.
	bt_hcitr_tih4_reset	This is function bt_hcitr_tih4_reset.
	bt_hcitr_tih4_start	This is function bt_hcitr_tih4_start.
	bt_hcitr_tih4_wake_up	This is function bt_hcitr_tih4_wake_up.

Types

	Name	Description
	bt_hcitr_tih4_power_callback_fp	This is type bt_hcitr_tih4_power_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

hcitr_uart.h

Functions

	Name	Description
	bt_hcitr_uart_init	brief Initialize HCI UART (H4) transport protocol ingroup hcitr_uart details This function initializes internal structures of the transport. The application must call it as early as possible before bt_hcitr_uart_start and before the stack is initialized and started with bt_sys_init and bt_sys_start.
	bt_hcitr_uart_reset	brief Re-initialize HCI UART (H4) transport protocol ingroup hcitr_uart details This function re-initializes the transport. Currently it simply calls bt_hcitr_uart_init. After calling this function the application must perform the full initialization of the stack by calling bt_sys_init, bt_sys_start and initialization functions of all other profile modules the application is intending to use.
	bt_hcitr_uart_start	brief Start HCI UART (H4) transport protocol ingroup hcitr_uart details This function starts the transport, i.e., makes it able to receive and send packets.

Macros

	Name	Description
	HCI_UART_PACKET_TYPE_ACL_DATA	This is macro HCI_UART_PACKET_TYPE_ACL_DATA.
	HCI_UART_PACKET_TYPE_COMMAND	defgroup hcitr_uart HCI UART (H4) transport protocol ingroup hcitr details This module describes functions used to initialize and start HCI UART transport protocol. The transport uses common interface for exchanging data between the host CPU and HCI controller defined in bt_hcitr.h. This interface consist of two functions that must be implemented by the application: li bt_oem_send() li bt_oem_recv()

	HCI_UART_PACKET_TYPE_EVENT	This is macro HCI_UART_PACKET_TYPE_EVENT.
	HCI_UART_PACKET_TYPE_SCO_DATA	This is macro HCI_UART_PACKET_TYPE_SCO_DATA.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

lm.h

Macros

	Name	Description
	LM_PACKET_TYPE_DH1	This is macro LM_PACKET_TYPE_DH1.
	LM_PACKET_TYPE_DH3	This is macro LM_PACKET_TYPE_DH3.
	LM_PACKET_TYPE_DH5	This is macro LM_PACKET_TYPE_DH5.
	LM_PACKET_TYPE_DM1	This is macro LM_PACKET_TYPE_DM1.
	LM_PACKET_TYPE_DM3	This is macro LM_PACKET_TYPE_DM3.
	LM_PACKET_TYPE_DM5	This is macro LM_PACKET_TYPE_DM5.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap.h

Functions

	Name	Description
	bt_l2cap_allocate_mgr	brief Allocate L2CAP manager. ingroup l2cap details This function allocates and initializes an L2CAP manager structure. One L2CAP manager manages all L2CAP connections for a particular local device. The local device is specified by the c hci_ctrl parameter. param hci_ctrl Pointer to the hci_ctrl_state structure that represents the local device (HCI controller) for which a L2CAP manager is to be allocated. return A pointer to the allocated L2CAP manager structure. The returned L2CAP manager should be freed by a call to bt_l2cap_free_mgr when it is no longer needed.
	bt_l2cap_connect_ext	brief Connect to a remote device. ingroup l2cap details This function establishes an L2CAP connection with a remote device on a specific PSM. When connect operation completes a callback function is called. param mgr The L2CAP manager. param remote_addr The address of the remote device. param psm The PSM for the connection. param connect_cb The Callback function that is called when the connect operation completes. param param A pointer to arbitrary data to be passed to the c connect_cb callback. param state_cb The callback function that is called when the state of the established connection changes. return li c TRUE... more
	bt_l2cap_connect_fixed_channel	This is function bt_l2cap_connect_fixed_channel.
	bt_l2cap_disconnect	brief Close L2CAP channel. ingroup l2cap details This function closes an L2CAP channel. The channel can be established either by a call to bt_l2cap_connect() or by an incoming connection request. param ch The L2CAP channel to be closed. return li c TRUE when the function succeeds. li c FALSE otherwise. The callback function is not called in this case.
	bt_l2cap_disconnect_ex	This is function bt_l2cap_disconnect_ex.
	bt_l2cap_echo	brief Send echo command

	bt_l2cap_free_mgr	brief Release L2CAP manger. ingroup l2cap details This function releases the L2CAP manager structure. param mgr The L2CAP manager structure to be released.
	bt_l2cap_get_mgr	This is function bt_l2cap_get_mgr.
	bt_l2cap_hci_has_open_channels	This is function bt_l2cap_hci_has_open_channels.
	bt_l2cap_init	brief Initialize the L2CAP layer. ingroup l2cap details This function initializes the L2CAP layer of the stack. It must be called prior to any other L2CAP function can be called.
	bt_l2cap_is_channel_open	This is function bt_l2cap_is_channel_open.
	bt_l2cap_listen_ext	brief Listen for incoming connections. ingroup l2cap details This function tells the L2CAP manager to listen for incoming connections on a specific PSM. When a connection is established a callback function is called. param mgr The L2CAP manager. param psm The PSM on which the manager will listen and accept incoming connections. param callback The callback function that will be called when an incoming connection is established. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter. return li c TRUE when the function succeeds. li c FALSE otherwise. The... more
	bt_l2cap_listen_fixed_channel	This is function bt_l2cap_listen_fixed_channel.
	bt_l2cap_reject	brief Send reject command (used to reject unknown or invalid commands)
	bt_l2cap_send_config	This is function bt_l2cap_send_config.
	bt_l2cap_update_conn_parameters	This is function bt_l2cap_update_conn_parameters.

Macros

	Name	Description
	bt_l2cap_connect	This is macro bt_l2cap_connect.
	bt_l2cap_listen	This is macro bt_l2cap_listen.
	L2CAP_DEFAULT_ERTX	seconds
	L2CAP_DEFAULT_RTX	seconds
	L2CAP_FRAME_TYPE_I	This is macro L2CAP_FRAME_TYPE_I.
	L2CAP_FRAME_TYPE_S	This is macro L2CAP_FRAME_TYPE_S.
	L2CAP_HCI_CONNECT_PACKET_TYPE	This is macro L2CAP_HCI_CONNECT_PACKET_TYPE.
	L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE	This is macro L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE.
	L2CAP_HCI_CONNECT_ROLE_SWITCH	This is macro L2CAP_HCI_CONNECT_ROLE_SWITCH.
	L2CAP_HEADER_LEN	This is macro L2CAP_HEADER_LEN.
	L2CAP_IDLE_HCI_CONNECTION_TIMEOUT	This is macro L2CAP_IDLE_HCI_CONNECTION_TIMEOUT.
	L2CAP_MAX_MANAGERS	This is macro L2CAP_MAX_MANAGERS.
	L2CAP_MAX_MTU	This is macro L2CAP_MAX_MTU.
	L2CAP_MAX_RTX	This is macro L2CAP_MAX_RTX.
	L2CAP_MGR_STATE_FREE	This is macro L2CAP_MGR_STATE_FREE.
	L2CAP_MGR_STATE_LISTENING_L2CAP	This is macro L2CAP_MGR_STATE_LISTENING_L2CAP.
	L2CAP_MGR_STATE_USED	This is macro L2CAP_MGR_STATE_USED.
	L2CAP_MIN_MTU	This is macro L2CAP_MIN_MTU.
	L2CAP_MONITOR_TIMEOUT	This is macro L2CAP_MONITOR_TIMEOUT.
	L2CAP_RETR_TIMEOUT	This is macro L2CAP_RETR_TIMEOUT.
	L2CAP_SAR_SDU_CONTINUE	This is macro L2CAP_SAR_SDU_CONTINUE.
	L2CAP_SAR_SDU_START	This is macro L2CAP_SAR_SDU_START.
	L2CAP_SAR_SDU_STOP	This is macro L2CAP_SAR_SDU_STOP.
	L2CAP_SAR_UNSEGMENTED	This is macro L2CAP_SAR_UNSEGMENTED.
	L2CAP_SFUNCTION_REJ	This is macro L2CAP_SFUNCTION_REJ.
	L2CAP_SFUNCTION_RNR	This is macro L2CAP_SFUNCTION_RNR.
	L2CAP_SFUNCTION_RR	This is macro L2CAP_SFUNCTION_RR.
	L2CAP_SFUNCTION_SREJ	This is macro L2CAP_SFUNCTION_SREJ.
	PSM_ATT	This is macro PSM_ATT.
	PSM_AVCTP	This is macro PSM_AVCTP.
	PSM_AVCTP_Browsing	This is macro PSM_AVCTP_Browsing.
	PSM_AVDTP	This is macro PSM_AVDTP.

	PSM_BNEP	This is macro PSM_BNEP.
	PSM_HID_Control	This is macro PSM_HID_Control.
	PSM_HID_Interrupt	This is macro PSM_HID_Interrupt.
	PSM_RFCOMM	This is macro PSM_RFCOMM.
	PSM_SDP	This is macro PSM_SDP.

Structures

	Name	Description
	_bt_l2cap_connect_params_s	This is record _bt_l2cap_connect_params_s.
	_bt_l2cap_fixed_channel_s	This is type bt_l2cap_fixed_channel_t.
	_bt_l2cap_mgr_s	This is type bt_l2cap_mgr_t.
	_bt_l2cap_psm_s	This is type bt_l2cap_psm_t.
	bt_l2cap_fixed_channel_t	This is type bt_l2cap_fixed_channel_t.
	bt_l2cap_mgr_p	This is type bt_l2cap_mgr_p.
	bt_l2cap_mgr_t	This is type bt_l2cap_mgr_t.
	bt_l2cap_psm_t	This is type bt_l2cap_psm_t.

Types

	Name	Description
	bt_l2cap_connect_callback_fp	This is type bt_l2cap_connect_callback_fp.
	bt_l2cap_connect_params_t	This is type bt_l2cap_connect_params_t.
	bt_l2cap_listen_callback_fp	This is type bt_l2cap_listen_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_command_queue.h

Functions

	Name	Description
	_bt_l2cap_notify_and_remove	This is function _bt_l2cap_notify_and_remove.
	_bt_l2cap_send_commands_from_queue	This is function _bt_l2cap_send_commands_from_queue.

Macros

	Name	Description
	L2CAP_MAX_CMD_QUEUE_LEN	This is macro L2CAP_MAX_CMD_QUEUE_LEN.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_config.h

Macros

	Name	Description
	__L2CAP_CONFIG_H	This is macro __L2CAP_CONFIG_H.
	L2CAP_ALLOCATE_BUFFERS_FUNCTION	This is macro L2CAP_ALLOCATE_BUFFERS_FUNCTION.
	L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	This is macro L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR.
	L2CAP_ALLOCATE_BUFFERS_VARS	This is macro L2CAP_ALLOCATE_BUFFERS_VARS.

	L2CAP_DECL_ERETR_FUNCTIONS	This is macro L2CAP_DECL_ERETR_FUNCTIONS.
	L2CAP_FIXED_CHANNELS_DECL	This is macro L2CAP_FIXED_CHANNELS_DECL.
	L2CAP_HCI_PACKET_TYPE	• brief L2CAP_HCI_PACKET_TYPE. • ingroup btconfig * • details Defines a set of packets that link manager is allowed to use when calling bt_l2cap_connect. • The default value is to enable all packet types. enable all packet types
	L2CAP_HCI_PAGE_SCAN_REPETITION_MODE	brief L2CAP_HCI_PAGE_SCAN_REPETITION_MODE. ingroup btconfig details Defines a default value of the page scan repetition mode when calling bt_l2cap_connect. Must be set to one of the following values: HCI_PAGE_SCAN_REPETITION_MODE_R0 HCI_PAGE_SCAN_REPETITION_MODE_R1 HCI_PAGE_SCAN_REPETITION_MODE_R2 The default value is HCI_PAGE_SCAN_REPETITION_MODE_R0.
	L2CAP_HCI_ROLE_SWITCH	brief L2CAP_HCI_ROLE_SWITCH. ingroup btconfig details Defines a default value of the role switch parameter when calling bt_l2cap_connect. Must be set to one of the following values: HCI_ROLE_SWITCH_ALLOW HCI_ROLE_SWITCH_DISALLOW The default value is to allow the role switch.
	L2CAP_IDLE_CONNECTION_TIMEOUT	seconds
	L2CAP_MAX_FIXED_CHANNELS	This is macro L2CAP_MAX_FIXED_CHANNELS.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_config_handlers.h

Macros

	Name	Description
	__L2CAP_CONFIG_HANDLERS_H	This is macro __L2CAP_CONFIG_HANDLERS_H.
	PACK_CONFIG_REQUEST	This is macro PACK_CONFIG_REQUEST.
	PACK_CONFIG_RESPONSE	This is macro PACK_CONFIG_RESPONSE.
	PACK_CONN_REQUEST	This is macro PACK_CONN_REQUEST.
	PACK_CONN_RESPONSE	This is macro PACK_CONN_RESPONSE.
	PACK_DCONN_REQUEST	This is macro PACK_DCONN_REQUEST.
	PACK_DCONN_RESPONSE	This is macro PACK_DCONN_RESPONSE.
	PACK_INFO_REQUEST	This is macro PACK_INFO_REQUEST.
	PACK_INFO_RESPONSE	This is macro PACK_INFO_RESPONSE.
	PROCESS_CONFIG_REQ	This is macro PROCESS_CONFIG_REQ.
	PROCESS_CONFIG_RES	This is macro PROCESS_CONFIG_RES.
	PROCESS_CONN_REQ	This is macro PROCESS_CONN_REQ.
	PROCESS_CONN_RES	This is macro PROCESS_CONN_RES.
	PROCESS_DCONN_REQ	This is macro PROCESS_DCONN_REQ.
	PROCESS_DCONN_RES	This is macro PROCESS_DCONN_RES.
	PROCESS_INFO_REQ	This is macro PROCESS_INFO_REQ.
	PROCESS_INFO_RES	This is macro PROCESS_INFO_RES.
	READ_CONFIG_REQUEST	This is macro READ_CONFIG_REQUEST.
	READ_CONFIG_RESPONSE	This is macro READ_CONFIG_RESPONSE.
	READ_CONN_REQUEST	This is macro READ_CONN_REQUEST.
	READ_CONN_RESPONSE	This is macro READ_CONN_RESPONSE.
	READ_DCONN_REQUEST	This is macro READ_DCONN_REQUEST.
	READ_DCONN_RESPONSE	This is macro READ_DCONN_RESPONSE.

	READ_INFO_REQUEST	This is macro READ_INFO_REQUEST.
	READ_INFO_RESPONSE	This is macro READ_INFO_RESPONSE.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_eretr.h

Enumerations

	Name	Description
	_bt_l2cap_eretr_xmit_event_e	This is type bt_l2cap_eretr_xmit_event_e.
	bt_l2cap_eretr_xmit_event_e	This is type bt_l2cap_eretr_xmit_event_e.

Functions

	Name	Description
	_bt_l2cap_eretr_handle_xmit_event	This is function _bt_l2cap_eretr_handle_xmit_event.
	_bt_l2cap_eretr_recv	This is function _bt_l2cap_eretr_recv.
	_bt_l2cap_eretr_retr_frames	This is function _bt_l2cap_eretr_retr_frames.
	_bt_l2cap_eretr_send_data	This is function _bt_l2cap_eretr_send_data.
	_bt_l2cap_eretr_send_pending_frames	This is function _bt_l2cap_eretr_send_pending_frames.
	_bt_l2cap_eretr_send_smart_data	This is function _bt_l2cap_eretr_send_smart_data.
	_bt_l2cap_fcs	This is function _bt_l2cap_fcs.
	_bt_l2cap_recv_req_seq_and_fbit	This is function _bt_l2cap_recv_req_seq_and_fbit.
	_bt_l2cap_send_ack	This is function _bt_l2cap_send_ack.
	_bt_l2cap_send_i_or_rr_or_rnr	This is function _bt_l2cap_send_i_or_rr_or_rnr.
	_bt_l2cap_send_rej	This is function _bt_l2cap_send_rej.
	_bt_l2cap_send_rnr	This is function _bt_l2cap_send_rnr.
	_bt_l2cap_send_rr	This is function _bt_l2cap_send_rr.
	_bt_l2cap_send_rr_or_rnr	This is function _bt_l2cap_send_rr_or_rnr.

Macros

	Name	Description
	_bt_l2cap_start_monitor_timer	This is macro _bt_l2cap_start_monitor_timer.
	_bt_l2cap_start_monitor_timer_if_not_running	This is macro _bt_l2cap_start_monitor_timer_if_not_running.
	_bt_l2cap_start_retr_timer	This is macro _bt_l2cap_start_retr_timer.
	_bt_l2cap_start_retr_timer_if_not_running	This is macro _bt_l2cap_start_retr_timer_if_not_running.
	_bt_l2cap_stop_monitor_timer	This is macro _bt_l2cap_stop_monitor_timer.
	_bt_l2cap_stop_retr_timer	This is macro _bt_l2cap_stop_retr_timer.
	GET_F_BIT	This is macro GET_F_BIT.
	GET_FRAME_TYPE	This is macro GET_FRAME_TYPE.
	GET_P_BIT	This is macro GET_P_BIT.
	GET_REQ_SEQ	This is macro GET_REQ_SEQ.
	GET_S_FUNCTION	This is macro GET_S_FUNCTION.
	GET_SAR	This is macro GET_SAR.
	GET_TX_SEQ	This is macro GET_TX_SEQ.
	SET_F_BIT	This is macro SET_F_BIT.
	SET_FRAME_TYPE	This is macro SET_FRAME_TYPE.
	SET_P_BIT	This is macro SET_P_BIT.
	SET_REQ_SEQ	This is macro SET_REQ_SEQ.
	SET_S_FUNCTION	This is macro SET_S_FUNCTION.
	SET_SAR	This is macro SET_SAR.

	SET_TX_SEQ	This is macro SET_TX_SEQ.

Structures

	Name	Description
	_bt_l2cap_xmit_event_param_t	This is type bt_l2cap_xmit_event_param_t.
	bt_l2cap_xmit_event_param_t	This is type bt_l2cap_xmit_event_param_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_fixed_channel.h

Functions

	Name	Description
	bt_l2cap_allocate_fixed_channel	This is function bt_l2cap_allocate_fixed_channel.
	bt_l2cap_find_fixed_channel	This is function bt_l2cap_find_fixed_channel.
	bt_l2cap_free_fixed_channel	This is function bt_l2cap_free_fixed_channel.

Macros

	Name	Description
	L2CAP_LINK_TYPE_BD_EDR	This is macro L2CAP_LINK_TYPE_BD_EDR.
	L2CAP_LINK_TYPE_LE	This is macro L2CAP_LINK_TYPE_LE.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

l2cap_packet.h

Functions

	Name	Description
	bt_l2cap_packet_cmd_assembler	This is function bt_l2cap_packet_cmd_assembler.
	bt_l2cap_packet_data_assembler	This is function bt_l2cap_packet_data_assembler.

Macros

	Name	Description
	L2CAP_PACKET_DATA_TYPE_RAW	This is macro L2CAP_PACKET_DATA_TYPE_RAW.
	L2CAP_PACKET_DATA_TYPE_SMART	This is macro L2CAP_PACKET_DATA_TYPE_SMART.

Structures

	Name	Description
	_bt_l2cap_packet_t	This is type bt_l2cap_packet_t.
	bt_l2cap_packet_t	This is type bt_l2cap_packet_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

l2cap_private.h

Functions

	Name	Description
	_bt_l2cap_clear_channel_cmd_queue	This is function _bt_l2cap_clear_channel_cmd_queue.
	_bt_l2cap_eretr_pack_config_request	This is function _bt_l2cap_eretr_pack_config_request.
	_bt_l2cap_process_connect_signal	This is function _bt_l2cap_process_connect_signal.
	_l2cap_allocate_buffers	Defined by OEM through library configuration
	_l2cap_data_receive_callback	From l2cap_recv.c
	_pack_cmd_reject	This is function _pack_cmd_reject.
	_pack_config_request	This is function _pack_config_request.
	_pack_config_response	This is function _pack_config_response.
	_pack_conn_param_update_request	This is function _pack_conn_param_update_request.
	_pack_conn_param_update_response	This is function _pack_conn_param_update_response.
	_pack_conn_request	This is function _pack_conn_request.
	_pack_conn_response	This is function _pack_conn_response.
	_pack_dconn_request	This is function _pack_dconn_request.
	_pack_dconn_response	This is function _pack_dconn_response.
	_pack_echo_request	This is function _pack_echo_request.
	_pack_echo_response	This is function _pack_echo_response.
	_pack_info_request	This is function _pack_info_request.
	_pack_info_response	This is function _pack_info_response.
	_process_config_req	From cmd_config.c
	_process_config_res	This is function _process_config_res.
	_process_conn_param_update_req	This is function _process_conn_param_update_req.
	_process_conn_param_update_res	This is function _process_conn_param_update_res.
	_process_conn_req	From cmd_connect.c
	_process_conn_res	This is function _process_conn_res.
	_process_dconn_req	From cmd_disconnect.c
	_process_dconn_res	This is function _process_dconn_res.
	_process_echo_req	From cmd_echo.c
	_process_echo_res	This is function _process_echo_res.
	_process_info_req	From cmd_info.c
	_process_info_res	This is function _process_info_res.
	_process_reject	From cmd_reject.c
	_process_unknown_req	From cmd_unknown.c
	_process_unknown_res	This is function _process_unknown_res.
	_read_cmd_reject	This is function _read_cmd_reject.
	_read_config_request	This is function _read_config_request.
	_read_config_response	This is function _read_config_response.
	_read_conn_param_update_request	This is function _read_conn_param_update_request.
	_read_conn_param_update_response	This is function _read_conn_param_update_response.
	_read_conn_request	This is function _read_conn_request.
	_read_conn_response	This is function _read_conn_response.
	_read_dconn_request	This is function _read_dconn_request.
	_read_dconn_response	This is function _read_dconn_response.
	_read_echo_request	This is function _read_echo_request.
	_read_echo_response	This is function _read_echo_response.
	_read_info_request	This is function _read_info_request.
	_read_info_response	This is function _read_info_response.
	read_command	From channel_cmd_recv.c

Macros

	Name	Description
	CHANNEL_SIGNAL_CMD_DISCONNECT_FIXED	This is macro CHANNEL_SIGNAL_CMD_DISCONNECT_FIXED.
	L2CAP_EXT_FEATURES_ENABLED	This is macro L2CAP_EXT_FEATURES_ENABLED.

Types

	Name	Description
	bt_l2cap_cmd_assembler_fp	This is type bt_l2cap_cmd_assembler_fp.
	bt_l2cap_cmd_parser_fp	This is type bt_l2cap_cmd_parser_fp.
	bt_l2cap_request_handler_fp	This is type bt_l2cap_request_handler_fp.
	bt_l2cap_response_handler_fp	This is type bt_l2cap_response_handler_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_psm.h

Functions

	Name	Description
	bt_l2cap_allocate_psm	This is function bt_l2cap_allocate_psm.
	bt_l2cap_find_psm	This is function bt_l2cap_find_psm.
	bt_l2cap_free_psm	This is function bt_l2cap_free_psm.
	bt_l2cap_init_psms	This is function bt_l2cap_init_psms.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_signal.h

Functions

	Name	Description
	_bt_l2cap_init_signal	This is function _bt_l2cap_init_signal.
	_bt_l2cap_set_signal	This is function _bt_l2cap_set_signal.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_test.h

Macros

	Name	Description
	bt_l2cap_test_enable_local_config	This is macro bt_l2cap_test_enable_local_config.
	bt_l2cap_test_enable_remote_config	This is macro bt_l2cap_test_enable_remote_config.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

l2cap_timer.h

Functions

	Name	Description
	_bt_l2cap_get_tick_count	This is function _bt_l2cap_get_tick_count.
	_bt_l2cap_init_timer	This is function _bt_l2cap_init_timer.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

packet.h

Structures

	Name	Description
	_bt_packet_t	This is type bt_packet_t.
	bt_packet_t	This is type bt_packet_t.

Types

	Name	Description
	bt_packet_assembler_fp	This is type bt_packet_assembler_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

patch.h

Macros

	Name	Description
	_W	This is macro _W.
	DCRH	Number of configuration blocks
	H	PS key store (0 = use default)
	W	This is macro W.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

queue.h

Functions

	Name	Description
	bt_q_add	This is function bt_q_add.
	bt_q_contains	This is function bt_q_contains.

	bt_q_get	This is function bt_q_get.
	bt_q_get_head	This is function bt_q_get_head.
	bt_q_get_length	This is function bt_q_get_length.
	bt_q_get_next	This is function bt_q_get_next.
	bt_q_push	This is function bt_q_push.
	bt_q_remove	This is function bt_q_remove.
	bt_q_remove_by_idx	This is function bt_q_remove_by_idx.

Structures

	Name	Description
	_bt_queue_element_t	This is type bt_queue_element_t.
	bt_queue_element_t	This is type bt_queue_element_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm.h

Functions

	Name	Description
	bt_rfcomm_allocate_dlc	brief Allocate DLC. ingroup rfcomm details This function allocates a new DLC on the specified RFCOMM session. param session The RFCOMM session. param dlci DLCI of the new DLC. return li A pointer to the new DLC if the function succeeds. li c NULL otherwise.
	bt_rfcomm_allocate_session	brief Allocate RFCOMM session. ingroup rfcomm details This function allocates a new RFCOMM session. param l2cap_mgr The L2CAP manager on which the RFCOMM session is to be created. param callback The callback function that is called when session state changes. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter. return li A pointer to the new RFCOMM session structure if the function succeeds. li c NULL otherwise.
	bt_rfcomm_cancel_listen	This is function bt_rfcomm_cancel_listen.
	bt_rfcomm_close_dlc	brief Close DLC. ingroup rfcomm details This function closes a DLC. If DLCI = 0, the parent RFCOMM session is also closed.
	bt_rfcomm_connect	brief Connect to a remote device. ingroup rfcomm details This function establishes an RFCOMM connection with a remote device and opens a data DLC. Changes in the session state are reported through a callback function specified when the session has been allocated via call to bt_rfcomm_allocate_session. Changes in the data DLC are reported through a callback function specified in this call. param remote_addr Address of the remote device. param server_channel A server channel of the remote RFCOMM server. param callback The callback function for reporting changes in DLC state opened by this call. param param An arbitrary data pointer... more
	bt_rfcomm_find_dlc	brief Find DLC
	bt_rfcomm_free_dlc	brief Release DLC. ingroup rfcomm details This function releases the specified DLC. param dlc The DLC to be released.
	bt_rfcomm_free_session	brief Release RFCOMM session. ingroup rfcomm details This function deallocates the specified RFCOMM session. This function does not disconnect the session. It just frees the memory used by the bt_rfcomm_session structure. The session has to be disconnected by calling bt_rfcomm_close_dlc with DLCI = 0 first. param session The RFCOMM session to be deallocated.
	bt_rfcomm_get_frame_length	This is function bt_rfcomm_get_frame_length.
	bt_rfcomm_init	brief Initialize the RFCOMM layer. ingroup rfcomm details This function initializes the RFCOMM layer of the stack. It must be called prior to any other RFCOMM function can be called.

	bt_rfcomm_listen	brief Listen for incoming connections. ingroup rfcomm details This function enables incoming connections on the specified RFCOMM session. Changes in the session state are reported through a callback function. param server_channel A server channel on which the RFCOMM session will listen and accept incoming connections. param callback The callback function that is called when session state changes. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter. return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.
	bt_rfcomm_open_dlc	brief Open DLC. ingroup rfcomm details This function opens the specified DLC. Before calling this function the RFCOMM session must be already open. This function is not to be used with DLCI = 0. Changes in DLC state are reported through a callback function. param dlc The DLC to open. param callback The callback function for reporting changes in DLC state. param param An arbitrary data pointer that will be passed to the callback function specified by the c callback parameter. return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in... more
	bt_rfcomm_send_credit	This is function bt_rfcomm_send_credit.
	bt_rfcomm_send_data	brief Send data over a DLC. ingroup rfcomm details This function sends data over the specified DLC. Operation completion is reported through callback function. param dlc The DLC. param data A pointer to the data to be sent. param len Data length. param callback The callback function that is called when operation completes. return li c TRUE if the function succeeds. li c FALSE otherwise. The callback function is not called in this case.

Macros

	Name	Description
	MK_CMD_ADDRESS	define RFCOMM_MS_RTC
	MK_DLCI	This is macro MK_DLCI.
	RFCOMM_CFC_ENABLED	This is macro RFCOMM_CFC_ENABLED.
	RFCOMM_CFC_LOCAL_CREDIT	This is macro RFCOMM_CFC_LOCAL_CREDIT.
	RFCOMM_CFC_MAX_INITIAL_CREDIT	This is macro RFCOMM_CFC_MAX_INITIAL_CREDIT.
	RFCOMM_CMD_STATUS_FC_PENDING	This is macro RFCOMM_CMD_STATUS_FC_PENDING.
	RFCOMM_CMD_STATUS_PENDING	This is macro RFCOMM_CMD_STATUS_PENDING.
	RFCOMM_CMD_STATUS_WAITING_RESPONSE	This is macro RFCOMM_CMD_STATUS_WAITING_RESPONSE.
	RFCOMM_COMMAND	This is macro RFCOMM_COMMAND.
	RFCOMM_CTL_MSG_CLD	This is macro RFCOMM_CTL_MSG_CLD.
	RFCOMM_CTL_MSG_FCOFF	This is macro RFCOMM_CTL_MSG_FCOFF.
	RFCOMM_CTL_MSG_FCON	This is macro RFCOMM_CTL_MSG_FCON.
	RFCOMM_CTL_MSG_MSC	This is macro RFCOMM_CTL_MSG_MSC.
	RFCOMM_CTL_MSG_NSC	This is macro RFCOMM_CTL_MSG_NSC.
	RFCOMM_CTL_MSG_PN	This is macro RFCOMM_CTL_MSG_PN.
	RFCOMM_CTL_MSG_PSC	This is macro RFCOMM_CTL_MSG_PSC.
	RFCOMM_CTL_MSG_RLS	This is macro RFCOMM_CTL_MSG_RLS.
	RFCOMM_CTL_MSG_RPN	This is macro RFCOMM_CTL_MSG_RPN.
	RFCOMM_CTL_MSG_SNC	This is macro RFCOMM_CTL_MSG_SNC.
	RFCOMM_CTL_MSG_TEST	This is macro RFCOMM_CTL_MSG_TEST.
	RFCOMM_DLC_CHANGED_CONN_STATE	This is macro RFCOMM_DLC_CHANGED_CONN_STATE.
	RFCOMM_DLC_CHANGED_REMOTE_MSC	This is macro RFCOMM_DLC_CHANGED_REMOTE_MSC.
	RFCOMM_DLC_CONNECTION_FAILED	This is macro RFCOMM_DLC_CONNECTION_FAILED.
	RFCOMM_DLC_STATE_CLOSED	This is macro RFCOMM_DLC_STATE_CLOSED.
	RFCOMM_DLC_STATE_OPEN	This is macro RFCOMM_DLC_STATE_OPEN.
	RFCOMM_DLCI_CONTROL	This is macro RFCOMM_DLCI_CONTROL.
	RFCOMM_DLCI_FREE	This is macro RFCOMM_DLCI_FREE.
	RFCOMM_ERR_DM	This is macro RFCOMM_ERR_DM.
	RFCOMM_ERR_INTERRUPTED	This is macro RFCOMM_ERR_INTERRUPTED.
	RFCOMM_ERR_SUCCESS	This is macro RFCOMM_ERR_SUCCESS.
	RFCOMM_ERR_TIMEOUT	This is macro RFCOMM_ERR_TIMEOUT.

	RFCOMM_FC_TYPE_AGREGATE	This is macro RFCOMM_FC_TYPE_AGREGATE.
	RFCOMM_FC_TYPE_CREDIT	This is macro RFCOMM_FC_TYPE_CREDIT.
	RFCOMM_FLAG_CR	This is macro RFCOMM_FLAG_CR.
	RFCOMM_FLAG_EA	This is macro RFCOMM_FLAG_EA.
	RFCOMM_FLAG_PF	This is macro RFCOMM_FLAG_PF.
	RFCOMM_FRAME_HEADER_LEN	This is macro RFCOMM_FRAME_HEADER_LEN.
	RFCOMM_FRAME_TYPE_DISC	This is macro RFCOMM_FRAME_TYPE_DISC.
	RFCOMM_FRAME_TYPE_DM	This is macro RFCOMM_FRAME_TYPE_DM.
	RFCOMM_FRAME_TYPE_SABM	This is macro RFCOMM_FRAME_TYPE_SABM.
	RFCOMM_FRAME_TYPE_UA	This is macro RFCOMM_FRAME_TYPE_UA.
	RFCOMM_FRAME_TYPE_UI	This is macro RFCOMM_FRAME_TYPE_UI.
	RFCOMM_FRAME_TYPE_UIH	This is macro RFCOMM_FRAME_TYPE_UIH.
	RFCOMM_MAX_INFO_LEN	This is macro RFCOMM_MAX_INFO_LEN.
	RFCOMM_MIX_INFO_LEN	This is macro RFCOMM_MIX_INFO_LEN.
	RFCOMM_MX_MSG_MAX_DATA_LEN	This is macro RFCOMM_MX_MSG_MAX_DATA_LEN.
	RFCOMM_RESPONSE	This is macro RFCOMM_RESPONSE.
	RFCOMM_ROLE_INITIATOR	This is macro RFCOMM_ROLE_INITIATOR.
	RFCOMM_ROLE_RESPONDER	This is macro RFCOMM_ROLE_RESPONDER.
	RFCOMM_RPN_BAUD_RATE_1152	This is macro RFCOMM_RPN_BAUD_RATE_1152.
	RFCOMM_RPN_BAUD_RATE_192	This is macro RFCOMM_RPN_BAUD_RATE_192.
	RFCOMM_RPN_BAUD_RATE_2304	This is macro RFCOMM_RPN_BAUD_RATE_2304.
	RFCOMM_RPN_BAUD_RATE_24	This is macro RFCOMM_RPN_BAUD_RATE_24.
	RFCOMM_RPN_BAUD_RATE_384	This is macro RFCOMM_RPN_BAUD_RATE_384.
	RFCOMM_RPN_BAUD_RATE_48	This is macro RFCOMM_RPN_BAUD_RATE_48.
	RFCOMM_RPN_BAUD_RATE_576	This is macro RFCOMM_RPN_BAUD_RATE_576.
	RFCOMM_RPN_BAUD_RATE_72	This is macro RFCOMM_RPN_BAUD_RATE_72.
	RFCOMM_RPN_BAUD_RATE_96	default
	RFCOMM_RPN_DATA_BIT_5	This is macro RFCOMM_RPN_DATA_BIT_5.
	RFCOMM_RPN_DATA_BIT_6	This is macro RFCOMM_RPN_DATA_BIT_6.
	RFCOMM_RPN_DATA_BIT_7	This is macro RFCOMM_RPN_DATA_BIT_7.
	RFCOMM_RPN_DATA_BIT_8	default
	RFCOMM_RPN_FLC_N	This is macro RFCOMM_RPN_FLC_N.
	RFCOMM_RPN_FLC_RTC_INPUT	This is macro RFCOMM_RPN_FLC_RTC_INPUT.
	RFCOMM_RPN_FLC_RTC_OUTPUT	This is macro RFCOMM_RPN_FLC_RTC_OUTPUT.
	RFCOMM_RPN_FLC_RTR_INPUT	This is macro RFCOMM_RPN_FLC_RTR_INPUT.
	RFCOMM_RPN_FLC_RTR_OUTPUT	This is macro RFCOMM_RPN_FLC_RTR_OUTPUT.
	RFCOMM_RPN_FLC_XONOFF_INPUT	This is macro RFCOMM_RPN_FLC_XONOFF_INPUT.
	RFCOMM_RPN_FLC_XONOFF_OUTPUT	This is macro RFCOMM_RPN_FLC_XONOFF_OUTPUT.
	RFCOMM_RPN_PARITY_EVEN	This is macro RFCOMM_RPN_PARITY_EVEN.
	RFCOMM_RPN_PARITY_MARK	This is macro RFCOMM_RPN_PARITY_MARK.
	RFCOMM_RPN_PARITY_N	default
	RFCOMM_RPN_PARITY_ODD	This is macro RFCOMM_RPN_PARITY_ODD.
	RFCOMM_RPN_PARITY_SPACE	This is macro RFCOMM_RPN_PARITY_SPACE.
	RFCOMM_RPN_PARITY_Y	This is macro RFCOMM_RPN_PARITY_Y.
	RFCOMM_RPN_STOP_BIT_1	default
	RFCOMM_RPN_STOP_BIT_1_5	This is macro RFCOMM_RPN_STOP_BIT_1_5.
	RFCOMM_RPN_XOFF_DEFAULT	This is macro RFCOMM_RPN_XOFF_DEFAULT.
	RFCOMM_RPN_XON_DEFAULT	This is macro RFCOMM_RPN_XON_DEFAULT.
	RFCOMM_SERIAL_PORT_CH_1	This is macro RFCOMM_SERIAL_PORT_CH_1.
	RFCOMM_SERIAL_PORT_CH_10	This is macro RFCOMM_SERIAL_PORT_CH_10.
	RFCOMM_SERIAL_PORT_CH_11	This is macro RFCOMM_SERIAL_PORT_CH_11.
	RFCOMM_SERIAL_PORT_CH_12	This is macro RFCOMM_SERIAL_PORT_CH_12.
	RFCOMM_SERIAL_PORT_CH_13	This is macro RFCOMM_SERIAL_PORT_CH_13.
	RFCOMM_SERIAL_PORT_CH_14	This is macro RFCOMM_SERIAL_PORT_CH_14.
	RFCOMM_SERIAL_PORT_CH_15	This is macro RFCOMM_SERIAL_PORT_CH_15.

	RFCOMM_SERIAL_PORT_CH_16	This is macro RFCOMM_SERIAL_PORT_CH_16.
	RFCOMM_SERIAL_PORT_CH_17	This is macro RFCOMM_SERIAL_PORT_CH_17.
	RFCOMM_SERIAL_PORT_CH_18	This is macro RFCOMM_SERIAL_PORT_CH_18.
	RFCOMM_SERIAL_PORT_CH_19	This is macro RFCOMM_SERIAL_PORT_CH_19.
	RFCOMM_SERIAL_PORT_CH_2	This is macro RFCOMM_SERIAL_PORT_CH_2.
	RFCOMM_SERIAL_PORT_CH_20	This is macro RFCOMM_SERIAL_PORT_CH_20.
	RFCOMM_SERIAL_PORT_CH_21	This is macro RFCOMM_SERIAL_PORT_CH_21.
	RFCOMM_SERIAL_PORT_CH_22	This is macro RFCOMM_SERIAL_PORT_CH_22.
	RFCOMM_SERIAL_PORT_CH_23	This is macro RFCOMM_SERIAL_PORT_CH_23.
	RFCOMM_SERIAL_PORT_CH_24	This is macro RFCOMM_SERIAL_PORT_CH_24.
	RFCOMM_SERIAL_PORT_CH_25	This is macro RFCOMM_SERIAL_PORT_CH_25.
	RFCOMM_SERIAL_PORT_CH_26	This is macro RFCOMM_SERIAL_PORT_CH_26.
	RFCOMM_SERIAL_PORT_CH_27	This is macro RFCOMM_SERIAL_PORT_CH_27.
	RFCOMM_SERIAL_PORT_CH_28	This is macro RFCOMM_SERIAL_PORT_CH_28.
	RFCOMM_SERIAL_PORT_CH_29	This is macro RFCOMM_SERIAL_PORT_CH_29.
	RFCOMM_SERIAL_PORT_CH_3	This is macro RFCOMM_SERIAL_PORT_CH_3.
	RFCOMM_SERIAL_PORT_CH_30	This is macro RFCOMM_SERIAL_PORT_CH_30.
	RFCOMM_SERIAL_PORT_CH_4	This is macro RFCOMM_SERIAL_PORT_CH_4.
	RFCOMM_SERIAL_PORT_CH_5	This is macro RFCOMM_SERIAL_PORT_CH_5.
	RFCOMM_SERIAL_PORT_CH_6	This is macro RFCOMM_SERIAL_PORT_CH_6.
	RFCOMM_SERIAL_PORT_CH_7	This is macro RFCOMM_SERIAL_PORT_CH_7.
	RFCOMM_SERIAL_PORT_CH_8	This is macro RFCOMM_SERIAL_PORT_CH_8.
	RFCOMM_SERIAL_PORT_CH_9	This is macro RFCOMM_SERIAL_PORT_CH_9.
	RFCOMM_SESSION_CHANGED_AFC	This is macro RFCOMM_SESSION_CHANGED_AFC.
	RFCOMM_SESSION_CHANGED_CONN_STATE	This is macro RFCOMM_SESSION_CHANGED_CONN_STATE.
	RFCOMM_SESSION_STATE_CONNECTED	This is macro RFCOMM_SESSION_STATE_CONNECTED.
	RFCOMM_SESSION_STATE_DISCONNECTED	This is macro RFCOMM_SESSION_STATE_DISCONNECTED.
	RFCOMM_SESSION_STATE_FREE	This is macro RFCOMM_SESSION_STATE_FREE.
	RFCOMM_TIMEOUT	This is macro RFCOMM_TIMEOUT.

Structures

	Name	Description
	_bt_rfcomm_command_t	This is type bt_rfcomm_command_t.
	_bt_rfcomm_ctl_msg_t	This is type bt_rfcomm_ctl_msg_t.
	_bt_rfcomm_dlc_t	This is type bt_rfcomm_dlc_t.
	_bt_rfcomm_session_t	This is type bt_rfcomm_session_t.
	bt_rfcomm_command_p	This is type bt_rfcomm_command_p.
	bt_rfcomm_command_t	This is type bt_rfcomm_command_t.
	bt_rfcomm_ctl_msg_p	This is type bt_rfcomm_ctl_msg_p.
	bt_rfcomm_ctl_msg_t	This is type bt_rfcomm_ctl_msg_t.
	bt_rfcomm_dlc_p	This is type bt_rfcomm_dlc_p.
	bt_rfcomm_dlc_t	This is type bt_rfcomm_dlc_t.
	bt_rfcomm_session_p	This is type bt_rfcomm_session_p.
	bt_rfcomm_session_t	This is type bt_rfcomm_session_t.

Types

	Name	Description
	bt_rfcomm_cmd_callback_fp	This is type bt_rfcomm_cmd_callback_fp.
	bt_rfcomm_dlc_state_callback_fp	This is type bt_rfcomm_dlc_state_callback_fp.
	bt_rfcomm_read_data_callback_fp	This is type bt_rfcomm_read_data_callback_fp.
	bt_rfcomm_send_data_callback_fp	This is type bt_rfcomm_send_data_callback_fp.
	bt_rfcomm_state_callback_fp	This is type bt_rfcomm_state_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_cmd_queue.h

Functions

	Name	Description
	rfcomm_cq_ack_cmd	This is function rfcomm_cq_ack_cmd.
	rfcomm_cq_ack_mx_cmd	This is function rfcomm_cq_ack_mx_cmd.
	rfcomm_cq_find_failed_pn	This is function rfcomm_cq_find_failed_pn.
	rfcomm_send_commands_from_queue	This is function rfcomm_send_commands_from_queue.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_config.h

Macros

	Name	Description
	__RFCOMM_CONFIG_H	This is macro __RFCOMM_CONFIG_H.
	RFCOMM_ALLOCATE_BUFFERS_FUNCTION	This is macro RFCOMM_ALLOCATE_BUFFERS_FUNCTION.
	RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR	This is macro RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR.
	RFCOMM_ALLOCATE_BUFFERS_VARS	This is macro RFCOMM_ALLOCATE_BUFFERS_VARS.
	RFCOMM_BUFFER_SIZE	This is macro RFCOMM_BUFFER_SIZE.
	RFCOMM_ENABLE_MULTIDEVICE_CHANNELS	brief Enable multi-device server channels. ingroup rfcomm_config details Normally each server channel can be used only once. I.e. if device A connected to channel 1, device B cannot connect to channel 1 until device A disconnects. With this option it is possible to make channels accept connections from several devices at the same time. I.e., if RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is TRUE both device A and device B can connect to channel 1 at the same time.
	RFCOMM_INFO_LEN	brief Maximum size of the data portion of a UIH frame. ingroup rfcomm_config details This parameter defines the maximum size of the data portion of a UIH frame. If CFC is used the actual length of the data portion will be 1 byte less. This value must be less than or equal to HCI_L2CAP_BUFFER_LEN - RFCOMM_FRAME_HEADER_LEN - L2CAP_HEADER_LEN.
	RFCOMM_LOCAL_CREDIT	brief The number of receive buffers. ingroup rfcomm_config details This parameter defines the number of received UIH frames that can be stored on the local device. The flow control mechanism used in RFCOMM ensures that the remote side of the link always knows how many free buffers left on the local device. When the number of free buffers reaches 0, the transmitter stops sending data frames until the receiver frees some buffers. The RFCOMM layer does not actually allocate space for buffers. It uses RFCOMM_LOCAL_CREDIT to keep track of free buffers and report them to the remote side. Actual memory... more
	RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL	This is macro RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL.
	RFCOMM_MAX_CMD_BUFFERS	brief Maximum number of command buffers. ingroup rfcomm_config details This parameter defines the maximum number of commands that can be sent at the same time. It is usually enough to reserve 2 buffers for each DLC excluding control DLC. Therefore, this value can be defined as rn #define RFCOMM_MAX_CMD_BUFFERS (RFCOMM_MAX_DLCS - 1) * 2

	RFCOMM_MAX_DLCS	brief Maximum number of DLCs ingroup rfcomm_config details This parameter defines the maximum number of DLCs on each session. This value should be at least 2 because each session uses one DLC to convey multiplexer control messages. All other DLCs are used to emulate serial ports.
	RFCOMM_MAX_SERVER_CHANNELS	brief Maximum number of Server channels ingroup rfcomm_config details This parameter defines the maximum number of server channels exposed by the local device. This value should not exceed RFCOMM_MAX_DLCS - 1.
	RFCOMM_MAX_SESSIONS	brief Maximum number of remote devices a local device can be connected to ingroup rfcomm_config details This parameter defines the maximum number of remote devices a local device can have simultaneous connections to. This value should not exceed HCI_MAX_HCI_CONNECTIONS.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_mgr.h

Functions

	Name	Description
	_bt_rfcomm_allocate_channel	This is function _bt_rfcomm_allocate_channel.
	_bt_rfcomm_find_channel	This is function _bt_rfcomm_find_channel.
	_bt_rfcomm_get_mgr	This is function _bt_rfcomm_get_mgr.
	_bt_rfcomm_mgr_l2cap_listen_callback	This is function _bt_rfcomm_mgr_l2cap_listen_callback.
	_bt_rfcomm_mgr_notify_listeners	This is function _bt_rfcomm_mgr_notify_listeners.
	_rfcomm_init_mgr	This is function _rfcomm_init_mgr.
	bt_rfcomm_register_listener	This is function bt_rfcomm_register_listener.
	bt_rfcomm_unregister_listener	This is function bt_rfcomm_unregister_listener.

Structures

	Name	Description
	_bt_rfcomm_mgr_t	This is type bt_rfcomm_mgr_t.
	_bt_rfcomm_server_channel_t	This is type bt_rfcomm_server_channel_t.
	_bt_rfcomm_session_listener_t	This is type bt_rfcomm_session_listener_t.
	bt_rfcomm_mgr_t	This is type bt_rfcomm_mgr_t.
	bt_rfcomm_server_channel_t	This is type bt_rfcomm_server_channel_t.
	bt_rfcomm_session_listener_t	This is type bt_rfcomm_session_listener_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_mx.h

Functions

	Name	Description
	_rfcomm_allocate_mx_cmd	This is function _rfcomm_allocate_mx_cmd.
	_rfcomm_mx_process_fc	This is function _rfcomm_mx_process_fc.
	_rfcomm_mx_process_pn	This is function _rfcomm_mx_process_pn.

	_rfcomm_mx_process_rpn	This is function _rfcomm_mx_process_rpn.
	_rfcomm_process_mx_fc_response	This is function _rfcomm_process_mx_fc_response.
	_rfcomm_process_mx_msc_response	This is function _rfcomm_process_mx_msc_response.
	_rfcomm_process_mx_pn_response	This is function _rfcomm_process_mx_pn_response.
	_rfcomm_process_mx_rls_response	This is function _rfcomm_process_mx_rls_response.
	_rfcomm_send_mx_fc_cmd	This is function _rfcomm_send_mx_fc_cmd.
	_rfcomm_send_mx_msc_response	This is function _rfcomm_send_mx_msc_response.
	_rfcomm_send_mx_nsc_response	This is function _rfcomm_send_mx_nsc_response.
	_rfcomm_send_mx_rls_cmd	This is function _rfcomm_send_mx_rls_cmd.
	_rfcomm_send_mx_rls_response	This is function _rfcomm_send_mx_rls_response.
	_rfcomm_send_mx_test_response	This is function _rfcomm_send_mx_test_response.
	rfcomm_send_mx_msc_cmd	This is function rfcomm_send_mx_msc_cmd.
	rfcomm_send_mx_pn_cmd	This is function rfcomm_send_mx_pn_cmd.

Macros

	Name	Description
	RFCOMM_LINE_STATUS_FRAMING	This is macro RFCOMM_LINE_STATUS_FRAMING.
	RFCOMM_LINE_STATUS_OVERRUN	This is macro RFCOMM_LINE_STATUS_OVERRUN.
	RFCOMM_LINE_STATUS_PARITY	This is macro RFCOMM_LINE_STATUS_PARITY.
	RFCOMM_MODEM_STATUS_DV	This is macro RFCOMM_MODEM_STATUS_DV.
	RFCOMM_MODEM_STATUS_FC	This is macro RFCOMM_MODEM_STATUS_FC.
	RFCOMM_MODEM_STATUS_IC	This is macro RFCOMM_MODEM_STATUS_IC.
	RFCOMM_MODEM_STATUS_RTC	This is macro RFCOMM_MODEM_STATUS_RTC.
	RFCOMM_MODEM_STATUS_RTR	This is macro RFCOMM_MODEM_STATUS_RTR.
	RFCOMM_MX_MSG_FCOFF	This is macro RFCOMM_MX_MSG_FCOFF.
	RFCOMM_MX_MSG_FCON	This is macro RFCOMM_MX_MSG_FCON.
	RFCOMM_MX_MSG_MSC	This is macro RFCOMM_MX_MSG_MSC.
	RFCOMM_MX_MSG_NSC	This is macro RFCOMM_MX_MSG_NSC.
	RFCOMM_MX_MSG_PN	This is macro RFCOMM_MX_MSG_PN.
	RFCOMM_MX_MSG_RLS	This is macro RFCOMM_MX_MSG_RLS.
	RFCOMM_MX_MSG_RPN	This is macro RFCOMM_MX_MSG_RPN.
	RFCOMM_MX_MSG_TEST	This is macro RFCOMM_MX_MSG_TEST.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_private.h

Functions

	Name	Description
	_bt_rfcomm_clear_queue	From command_queue.c
	_bt_rfcomm_mgr_allocate_session	This is function _bt_rfcomm_mgr_allocate_session.
	_calc_fcs	This is function _calc_fcs.
	_rfcomm_alloc_cmd_buffer	This is function _rfcomm_alloc_cmd_buffer.
	_rfcomm_allocate_buffers	Defined by OEM through library configuration
	_rfcomm_free_cmd_buffer	This is function _rfcomm_free_cmd_buffer.
	_rfcomm_init_cmd_buffers	From rfcomm_cmdbuffer.c
	_rfcomm_init_sessions	From rfcomm_session.c
	_rfcomm_l2cap_read_data_callback	From rfcomm_cmd_recv.c
	_rfcomm_l2cap_state_changed_callback	From rfcomm.c
	_rfcomm_process_cmd_frame_disc	From frame_disc.c

	_rfcomm_process_cmd_frame_dm	This is function _rfcomm_process_cmd_frame_dm.
	_rfcomm_process_cmd_frame_sabm	This is function _rfcomm_process_cmd_frame_sabm.
	_rfcomm_process_cmd_frame_ua	This is function _rfcomm_process_cmd_frame_ua.
	_rfcomm_process_cmd_frame_uih	From frame_uih.c
	_rfcomm_process_res_frame_disc	This is function _rfcomm_process_res_frame_disc.
	_rfcomm_process_res_frame_dm	This is function _rfcomm_process_res_frame_dm.
	_rfcomm_process_res_frame_sabm	This is function _rfcomm_process_res_frame_sabm.
	_rfcomm_process_res_frame_ua	This is function _rfcomm_process_res_frame_ua.
	_rfcomm_process_res_frame_uih	This is function _rfcomm_process_res_frame_uih.
	_rfcomm_send_command	From rfcomm_cmd_send.c
	_rfcomm_send_dm_response	From frame_dm.c
	_rfcomm_send_sabm_cmd	From frame_sabm.c
	_rfcomm_send_ua_response	From frame_ua.c
	check_fcs	From rfcomm_fcs.c
	rfcomm_find_session	This is function rfcomm_find_session.
	rfcomm_send_cmd	This is function rfcomm_send_cmd.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_signal.h

Functions

	Name	Description
	_bt_rfcomm_init_signal	This is function _bt_rfcomm_init_signal.
	_bt_rfcomm_set_signal	This is function _bt_rfcomm_set_signal.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

rfcomm_timer.h

Functions

	Name	Description
	_rfcomm_get_tick_count	This is function _rfcomm_get_tick_count.
	_rfcomm_init_timer	This is function _rfcomm_init_timer.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

sbc.h

Provides SBC Decoder Library interface services.

Functions

	Name	Description
	sbc_decode	Decodes the encoded SBC frame.
	sbc_get_codesize	Gets the input block size.
	sbc_get_frame_duration	Gets the time one input/output block takes to play in microseconds.
	sbc_get_frame_length	Gets the output block size.
	sbc_get_state_size	Gets the SBC Decoder's state data structure.
	sbc_init	Initializes the SBC Decoder.

Macros

	Name	Description
	__SBC_H	This is macro __SBC_H.
	INP_BUF_SIZE	Bytes
	MAX_SBC_DEC_STATE_SIZE	Bytes
	OUT_BUF_SIZE	Bytes
	SBC_AM_LOUDNESS	Allocation method
	SBC_AM_SNR	This is macro SBC_AM_SNR.
	SBC_BE	This is macro SBC_BE.
	SBC_BLK_12	This is macro SBC_BLK_12.
	SBC_BLK_16	This is macro SBC_BLK_16.
	SBC_BLK_4	Blocks
	SBC_BLK_8	This is macro SBC_BLK_8.
	SBC_FREQ_16000	Sampling frequency
	SBC_FREQ_32000	This is macro SBC_FREQ_32000.
	SBC_FREQ_44100	This is macro SBC_FREQ_44100.
	SBC_FREQ_48000	This is macro SBC_FREQ_48000.
	SBC_LE	Data endianness
	SBC_MODE_DUAL_CHANNEL	This is macro SBC_MODE_DUAL_CHANNEL.
	SBC_MODE_JOINT_STEREO	This is macro SBC_MODE_JOINT_STEREO.
	SBC_MODE_MONO	Channel mode
	SBC_MODE_STEREO	This is macro SBC_MODE_STEREO.
	SBC_SB_4	Sub-bands
	SBC_SB_8	This is macro SBC_SB_8.
	ssize_t	

Structures

	Name	Description
	sbc_struct	

Types

	Name	Description
	sbc_t	This is type sbc_t.

Description

SBC Decoder Library Interface File

SBC Decoder interface services to decode the encoded data received, from the Bluetooth device, which will be executed in audioTask.

File Name

sbc.h

Company

Microchip Technology Inc.

sdp.h

Functions

	Name	Description
	bt_sdp_start	brief Start SDP server ingroup sdp details This function starts the SDP server. param l2cap_mgr The L2CAP manager on which the SDP server is to be started. param sdp_db A pointer to the SDP database define with BEGIN_SDP_DB and END_SDP_DB macros. return li c TRUE if the function succeeds. li c FALSE otherwise.

Macros

	Name	Description
	BEGIN_DE_SEQUENCE	• brief Begin a data element sequence • ingroup sdp * • details BEGIN_DE_SEQUENCE and END_DE_SEQUENCE are used to define a data element sequence which is an array of sdp_data_element structures. • The array is used a search pattern in bt_sdp_request_service_search() and bt_sdp_request_service_attribute(). • For example, to find a AVRCP Target the following code can be used: code const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000, SDP_CLSID_AV_REMOTE_CONTROL }; const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000, SDP_CLSID_AV_REMOTE_CONTROL_TARGET }; BEGIN_DE_SEQUENCE(avrcp_target_service_search, 2) DE_UUID128(AVRCP_AV_REMOTE_CONTROL_CLSID) DE_UUID128(AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID) END_DE_SEQUENCE(avrcp_target_service_search) . . . void findAvrcpTarget(void) { INIT_DE_SEQUENCE(avrcp_target_service_search); bt_sdp_request_service_search(channel, &seq_avrcp_target_service_search, &callback, NULL); } endcode * • param... more
	DE_BOOL	brief Declare a boolean data element ingroup sdp details This macro adds a boolean data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_INT	brief Declare a 1-byte signed integer data element ingroup sdp details This macro adds a 1-byte signed integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_STRING	brief Declare a text string data element ingroup sdp details This macro adds a text string data element to a data element sequence. The length of the generated data element will be the actual length of the string. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.

	DE_STRING2	brief Declare a text string data element ingroup sdp details This macro adds a text string data element to a data element sequence. The length of the generated data element will be the value specified by the "len" parameter even if the actual length of the string is not equal to the "len" value. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value. param len The length of the data element value.
	DE_UINT	brief Declare a 1-byte unsigned integer data element ingroup sdp details This macro adds a 1-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_UINT16	brief Declare a 2-byte unsigned integer data element ingroup sdp details This macro adds a 2-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_URL	brief Declare a URL data element ingroup sdp details This macro adds a URL data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value which is a pointer to a string.
	DE_UUID128	brief Declare a 128-bit UUID data element ingroup sdp details This macro adds a 128-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value. The value must be a name of a variable of type bt_uuid.
	DE_UUID16	brief Declare a 16-bit UUID data element ingroup sdp details This macro adds a 16-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	DE_UUID32	brief Declare a 32-bit UUID data element ingroup sdp details This macro adds a 32-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE. param value The data element value.
	END_DE_SEQUENCE	brief End a data element sequence ingroup sdp details BEGIN_DE_SEQUENCE and END_DE_SEQUENCE are used to define a data element sequence which is an array of bt_sdp_data_element structures. param id The data element sequence identifier.
	INIT_DE_SEQUENCE	brief Initialize a data element sequence ingroup sdp details This macro calls a function defined in BEGIN_DE_SEQUENCE which initializes the data element sequence. param id The data element sequence identifier.
	SDP_ATTRID_AdditionalProtocolDescriptorLists	This is macro SDP_ATTRID_AdditionalProtocolDescriptorLists.
	SDP_ATTRID_BluetoothProfileDescriptorList	This is macro SDP_ATTRID_BluetoothProfileDescriptorList.
	SDP_ATTRID_BrowseGroupList	This is macro SDP_ATTRID_BrowseGroupList.
	SDP_ATTRID_ClientExecutableURL	This is macro SDP_ATTRID_ClientExecutableURL.
	SDP_ATTRID_DIPrimaryRecord	This is macro SDP_ATTRID_DIPrimaryRecord.
	SDP_ATTRID_DIProductId	This is macro SDP_ATTRID_DIProductId.
	SDP_ATTRID_DISpecificationId	This is macro SDP_ATTRID_DISpecificationId.
	SDP_ATTRID_DIVendorId	This is macro SDP_ATTRID_DIVendorId.
	SDP_ATTRID_DIVendorIdSource	This is macro SDP_ATTRID_DIVendorIdSource.
	SDP_ATTRID_DIVersion	This is macro SDP_ATTRID_DIVersion.
	SDP_ATTRID_DocumentationURL	This is macro SDP_ATTRID_DocumentationURL.
	SDP_ATTRID_GAPRemoteAudioVolumeControl	This is macro SDP_ATTRID_GAPRemoteAudioVolumeControl.
	SDP_ATTRID_GroupID	This is macro SDP_ATTRID_GroupID.
	SDP_ATTRID_HCRP_1284ID	This is macro SDP_ATTRID_HCRP_1284ID.
	SDP_ATTRID_HCRP_DeviceLocation	This is macro SDP_ATTRID_HCRP_DeviceLocation.

	SDP_ATTRID_HCRP_DeviceName	This is macro SDP_ATTRID_HCRP_DeviceName.
	SDP_ATTRID_HCRP_FriendlyName	This is macro SDP_ATTRID_HCRP_FriendlyName.
	SDP_ATTRID_HDPDataExchangeSpecification	This is macro SDP_ATTRID_HDPDataExchangeSpecification.
	SDP_ATTRID_HDPMCAPSupportedProcedures	This is macro SDP_ATTRID_HDPMCAPSupportedProcedures.
	SDP_ATTRID_HDPSuportedFeatures	This is macro SDP_ATTRID_HDPSuportedFeatures.
	SDP_ATTRID_HFPAGNetwork	This is macro SDP_ATTRID_HFPAGNetwork.
	SDP_ATTRID_HFPSupportedFeatures	This is macro SDP_ATTRID_HFPSupportedFeatures.
	SDP_ATTRID_HIDBatteryPower	This is macro SDP_ATTRID_HIDBatteryPower.
	SDP_ATTRID_HIDBootDevice	This is macro SDP_ATTRID_HIDBootDevice.
	SDP_ATTRID_HIDCountryCode	This is macro SDP_ATTRID_HIDCountryCode.
	SDP_ATTRID_HIDDescriptorList	This is macro SDP_ATTRID_HIDDescriptorList.
	SDP_ATTRID_HIDDeviceReleaseNumber	This is macro SDP_ATTRID_HIDDeviceReleaseNumber.
	SDP_ATTRID_HIDDeviceSubclass	This is macro SDP_ATTRID_HIDDeviceSubclass.
	SDP_ATTRID_HIDLANGIDBaseList	This is macro SDP_ATTRID_HIDLANGIDBaseList.
	SDP_ATTRID_HIDNormallyConnectable	This is macro SDP_ATTRID_HIDNormallyConnectable.
	SDP_ATTRID_HIDParserVersion	This is macro SDP_ATTRID_HIDParserVersion.
	SDP_ATTRID_HIDProfileVersion	This is macro SDP_ATTRID_HIDProfileVersion.
	SDP_ATTRID_HIDReconnectInitiate	This is macro SDP_ATTRID_HIDReconnectInitiate.
	SDP_ATTRID_HIDRemoteWake	This is macro SDP_ATTRID_HIDRemoteWake.
	SDP_ATTRID_HIDSDPDisable	This is macro SDP_ATTRID_HIDSDPDisable.
	SDP_ATTRID_HIDSupervisionTimeout	This is macro SDP_ATTRID_HIDSupervisionTimeout.
	SDP_ATTRID_HIDVirtualCable	This is macro SDP_ATTRID_HIDVirtualCable.
	SDP_ATTRID_IconURL	This is macro SDP_ATTRID_IconURL.
	SDP_ATTRID_INVALID	This is macro SDP_ATTRID_INVALID.
	SDP_ATTRID_LanguageBaseAttributeIDList	This is macro SDP_ATTRID_LanguageBaseAttributeIDList.
	SDP_ATTRID_OFFSET_ProviderName	This is macro SDP_ATTRID_OFFSET_ProviderName.
	SDP_ATTRID_OFFSET_ServiceDescription	This is macro SDP_ATTRID_OFFSET_ServiceDescription.
	SDP_ATTRID_OFFSET_ServiceName	This is macro SDP_ATTRID_OFFSET_ServiceName.
	SDP_ATTRID_PrimaryLanguageBaseId	This is macro SDP_ATTRID_PrimaryLanguageBaseId.
	SDP_ATTRID_ProtocolDescriptorList	This is macro SDP_ATTRID_ProtocolDescriptorList.
	SDP_ATTRID_ServiceAvailability	This is macro SDP_ATTRID_ServiceAvailability.
	SDP_ATTRID_ServiceClassIDList	This is macro SDP_ATTRID_ServiceClassIDList.
	SDP_ATTRID_ServiceDatabaseState	This is macro SDP_ATTRID_ServiceDatabaseState.
	SDP_ATTRID_ServiceID	This is macro SDP_ATTRID_ServiceID.
	SDP_ATTRID_ServiceInfoTimeToLive	This is macro SDP_ATTRID_ServiceInfoTimeToLive.
	SDP_ATTRID_ServiceRecordHandle	This is macro SDP_ATTRID_ServiceRecordHandle.
	SDP_ATTRID_ServiceRecordState	This is macro SDP_ATTRID_ServiceRecordState.
	SDP_ATTRID_SupportedFeatures	This is macro SDP_ATTRID_SupportedFeatures.
	SDP_ATTRID_VersionNumberList	This is macro SDP_ATTRID_VersionNumberList.
	SDP_CLSID_ADVANCED_AUDIO_DISTRIBUTION	This is macro SDP_CLSID_ADVANCED_AUDIO_DISTRIBUTION.
	SDP_CLSID_AUDIO_SINK	This is macro SDP_CLSID_AUDIO_SINK.
	SDP_CLSID_AUDIO_SOURCE	This is macro SDP_CLSID_AUDIO_SOURCE.
	SDP_CLSID_AV_REMOTE_CONTROL	This is macro SDP_CLSID_AV_REMOTE_CONTROL.
	SDP_CLSID_AV_REMOTE_CONTROL_CONTROLLER	This is macro SDP_CLSID_AV_REMOTE_CONTROL_CONTROLLER.
	SDP_CLSID_AV_REMOTE_CONTROL_PROFILE_ID	This is macro SDP_CLSID_AV_REMOTE_CONTROL_PROFILE_ID.
	SDP_CLSID_AV_REMOTE_CONTROL_TARGET	This is macro SDP_CLSID_AV_REMOTE_CONTROL_TARGET.
	SDP_CLSID_AVCTP	This is macro SDP_CLSID_AVCTP.
	SDP_CLSID_AVDTP	This is macro SDP_CLSID_AVDTP.
	SDP_CLSID_BrowseGroupDescriptorServiceClassID	This is macro SDP_CLSID_BrowseGroupDescriptorServiceClassID.
	SDP_CLSID_DialupNetworking	This is macro SDP_CLSID_DialupNetworking.
	SDP_CLSID_GENERIC_AUDIO	This is macro SDP_CLSID_GENERIC_AUDIO.
	SDP_CLSID_HARD_COPY_CABLE_REPLACEMENT	This is macro SDP_CLSID_HARD_COPY_CABLE_REPLACEMENT.
	SDP_CLSID_HARD_COPY_CONTROL_CHANNEL	This is macro SDP_CLSID_HARD_COPY_CONTROL_CHANNEL.
	SDP_CLSID_HARD_COPY_DATA_CHANNEL	This is macro SDP_CLSID_HARD_COPY_DATA_CHANNEL.
	SDP_CLSID_HARD_COPY_NOTIFICATION	This is macro SDP_CLSID_HARD_COPY_NOTIFICATION.

	SDP_CLSID_HCR_PRINT	This is macro SDP_CLSID_HCR_PRINT.
	SDP_CLSID_HCR_SCAN	This is macro SDP_CLSID_HCR_SCAN.
	SDP_CLSID_HDP	This is macro SDP_CLSID_HDP.
	SDP_CLSID_HDP_SINK	This is macro SDP_CLSID_HDP_SINK.
	SDP_CLSID_HDP_SOURCE	This is macro SDP_CLSID_HDP_SOURCE.
	SDP_CLSID_HFP	This is macro SDP_CLSID_HFP.
	SDP_CLSID_HFP_AG	This is macro SDP_CLSID_HFP_AG.
	SDP_CLSID_HID	This is macro SDP_CLSID_HID.
	SDP_CLSID_HIDProtocol	This is macro SDP_CLSID_HIDProtocol.
	SDP_CLSID_HSP	This is macro SDP_CLSID_HSP.
	SDP_CLSID_HSP_AG	This is macro SDP_CLSID_HSP_AG.
	SDP_CLSID_HSP_HS	This is macro SDP_CLSID_HSP_HS.
	SDP_CLSID_L2CAP	This is macro SDP_CLSID_L2CAP.
	SDP_CLSID_MCAP_CONTROL	This is macro SDP_CLSID_MCAP_CONTROL.
	SDP_CLSID_MCAP_DATA	This is macro SDP_CLSID_MCAP_DATA.
	SDP_CLSID_OBEXFileTransfer	This is macro SDP_CLSID_OBEXFileTransfer.
	SDP_CLSID_OBEXObjectPush	This is macro SDP_CLSID_OBEXObjectPush.
	SDP_CLSID_PBAP_PCE	This is macro SDP_CLSID_PBAP_PCE.
	SDP_CLSID_PBAP_PSE	This is macro SDP_CLSID_PBAP_PSE.
	SDP_CLSID_PNPInformation	This is macro SDP_CLSID_PNPInformation.
	SDP_CLSID_PublicBrowseGroup	This is macro SDP_CLSID_PublicBrowseGroup.
	SDP_CLSID_RFCOMM	This is macro SDP_CLSID_RFCOMM.
	SDP_CLSID_SerialPort	This is macro SDP_CLSID_SerialPort.
	SDP_CLSID_ServiceDiscoveryServerServiceClassID	defgroup sdp SDP This module describe functions and data structures used to start the SDP server and perform SDP queries.
	SDP_DATA_TYPE_ALTERNATIVE	This is macro SDP_DATA_TYPE_ALTERNATIVE.
	SDP_DATA_TYPE_BOOL	This is macro SDP_DATA_TYPE_BOOL.
	SDP_DATA_TYPE_INT	This is macro SDP_DATA_TYPE_INT.
	SDP_DATA_TYPE_INT128	This is macro SDP_DATA_TYPE_INT128.
	SDP_DATA_TYPE_INT16	This is macro SDP_DATA_TYPE_INT16.
	SDP_DATA_TYPE_INT32	This is macro SDP_DATA_TYPE_INT32.
	SDP_DATA_TYPE_INT64	This is macro SDP_DATA_TYPE_INT64.
	SDP_DATA_TYPE_INT8	This is macro SDP_DATA_TYPE_INT8.
	SDP_DATA_TYPE_NIL	This is macro SDP_DATA_TYPE_NIL.
	SDP_DATA_TYPE_SEQUENCE	This is macro SDP_DATA_TYPE_SEQUENCE.
	SDP_DATA_TYPE_STRING	This is macro SDP_DATA_TYPE_STRING.
	SDP_DATA_TYPE_UINT	This is macro SDP_DATA_TYPE_UINT.
	SDP_DATA_TYPE_UINT128	This is macro SDP_DATA_TYPE_UINT128.
	SDP_DATA_TYPE_UINT16	This is macro SDP_DATA_TYPE_UINT16.
	SDP_DATA_TYPE_UINT32	This is macro SDP_DATA_TYPE_UINT32.
	SDP_DATA_TYPE_UINT64	This is macro SDP_DATA_TYPE_UINT64.
	SDP_DATA_TYPE_UINT8	This is macro SDP_DATA_TYPE_UINT8.
	SDP_DATA_TYPE_URL	This is macro SDP_DATA_TYPE_URL.
	SDP_DATA_TYPE_UUID	This is macro SDP_DATA_TYPE_UUID.
	SDP_DATA_TYPE_UUID128	This is macro SDP_DATA_TYPE_UUID128.
	SDP_DATA_TYPE_UUID16	This is macro SDP_DATA_TYPE_UUID16.
	SDP_DATA_TYPE_UUID32	This is macro SDP_DATA_TYPE_UUID32.
	SDP_ERROR_INSUFFICIENT_RESOURCE	This is macro SDP_ERROR_INSUFFICIENT_RESOURCE.
	SDP_ERROR_INVALID_CONTINUATION_STATE	This is macro SDP_ERROR_INVALID_CONTINUATION_STATE.
	SDP_ERROR_INVALID_PDU_SIZE	This is macro SDP_ERROR_INVALID_PDU_SIZE.
	SDP_ERROR_INVALID_REQUEST_SYNTAX	This is macro SDP_ERROR_INVALID_REQUEST_SYNTAX.
	SDP_ERROR_INVALID_SDP_VERSION	This is macro SDP_ERROR_INVALID_SDP_VERSION.
	SDP_ERROR_INVALID_SR_HANDLE	This is macro SDP_ERROR_INVALID_SR_HANDLE.
	SDP_ERROR_RESERVED	This is macro SDP_ERROR_RESERVED.

	SDP_ErrorResponse	This is macro SDP_ErrorResponse.
	SDP_FTP_SERVICE_ID	This is macro SDP_FTP_SERVICE_ID.
	SDP_HID_SERVICE_ID	This is macro SDP_HID_SERVICE_ID.
	SDP_HSP_AG_SERVICE_ID	This is macro SDP_HSP_AG_SERVICE_ID.
	SDP_HSP_HS_SERVICE_ID	This is macro SDP_HSP_HS_SERVICE_ID.
	SDP_MAX_ATTRIBUTE_PATTERN_LEN	This is macro SDP_MAX_ATTRIBUTE_PATTERN_LEN.
	SDP_MAX_DATA_ELEMENT_LEN	This is macro SDP_MAX_DATA_ELEMENT_LEN.
	SDP_MAX_DATA_ELEMENTS	This is macro SDP_MAX_DATA_ELEMENTS.
	SDP_MAX_SEARCH_PATTERN_LEN	This is macro SDP_MAX_SEARCH_PATTERN_LEN.
	SDP_MAX_TRANSACTIONS	typedef struct _sdp_db { bt_int count; bt_sdp_service_record_p *records; } bt_sdp_db, *bt_sdp_db_p;
	SDP_PDU_HEADER_LEN	This is macro SDP_PDU_HEADER_LEN.
	SDP_RFCOMM_SERVICE_ID	This is macro SDP_RFCOMM_SERVICE_ID.
	SDP_ServiceAttributeRequest	This is macro SDP_ServiceAttributeRequest.
	SDP_ServiceAttributeResponse	This is macro SDP_ServiceAttributeResponse.
	SDP_ServiceSearchAttributeRequest	This is macro SDP_ServiceSearchAttributeRequest.
	SDP_ServiceSearchAttributeResponse	This is macro SDP_ServiceSearchAttributeResponse.
	SDP_ServiceSearchRequest	This is macro SDP_ServiceSearchRequest.
	SDP_ServiceSearchResponse	This is macro SDP_ServiceSearchResponse.
	SDP_SR_HANDLE_HDP_SINK	This is macro SDP_SR_HANDLE_HDP_SINK.
	SDP_SR_HANDLE_HDP_SOURCE	This is macro SDP_SR_HANDLE_HDP_SOURCE.
	SDP_SR_HANDLE_HFP_HF	This is macro SDP_SR_HANDLE_HFP_HF.
	SDP_SR_HANDLE_HID	This is macro SDP_SR_HANDLE_HID.
	SDP_SR_HANDLE_HID_KEYBOARD	This is macro SDP_SR_HANDLE_HID_KEYBOARD.
	SDP_SR_HANDLE_HSP_HS	This is macro SDP_SR_HANDLE_HSP_HS.
	SDP_SR_HANDLE_OBEXFileTransfer	This is macro SDP_SR_HANDLE_OBEXFileTransfer.
	SDP_SR_HANDLE_OBEXObjectPush	This is macro SDP_SR_HANDLE_OBEXObjectPush.
	SDP_SR_HANDLE_PNPINFORMATION	This is macro SDP_SR_HANDLE_PNPINFORMATION.
	SDP_SR_HANDLE_RFCOMM	This is macro SDP_SR_HANDLE_RFCOMM.
	SDP_SR_HANDLE_SERVER	This is macro SDP_SR_HANDLE_SERVER.
	SDP_SR_HANDLE_TEST	for simulating service search using continuation state

Structures

	Name	Description
	_bt_sdp_data_element_t	This is type bt_sdp_data_element_t.
	_bt_sdp_found_attr_list_t	This is type bt_sdp_found_attr_list_t.
	_bt_sdp_sequence_t	This is type bt_sdp_sequence_t.
	_bt_sdp_serialization_state_t	This is type bt_sdp_serialization_state_t.
	_bt_sdp_service_transaction_t	This is type bt_sdp_service_transaction_t.
	_bt_sdp_transaction_t	This is type bt_sdp_transaction_t.
	bt_sdp_data_element_t	This is type bt_sdp_data_element_t.
	bt_sdp_found_attr_list_t	This is type bt_sdp_found_attr_list_t.
	bt_sdp_sequence_t	This is type bt_sdp_sequence_t.
	bt_sdp_serialization_state_p	This is type bt_sdp_serialization_state_p.
	bt_sdp_serialization_state_t	This is type bt_sdp_serialization_state_t.
	bt_sdp_service_transaction_p	This is type bt_sdp_service_transaction_p.
	bt_sdp_service_transaction_t	This is type bt_sdp_service_transaction_t.
	bt_sdp_transaction_t	This is type bt_sdp_transaction_t.

Types

	Name	Description
	bt_sdp_data_element_cp	This is type bt_sdp_data_element_cp.
	bt_sdp_data_element_p	This is type bt_sdp_data_element_p.
	bt_sdp_sequence_cp	This is type bt_sdp_sequence_cp.
	bt_sdp_sequence_p	This is type bt_sdp_sequence_p.

	bt_sr_handle_p	This is type bt_sr_handle_p.
	bt_sr_handle_t	This is type bt_sr_handle_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

sdp_client.h

Functions

	Name	Description
	bt_sdp_request_service_attribute	brief Search attributes ingroup sdp details This function retrieves attribute values from a service record. param channel The L2CAP channel used to communicate to the remote SDP server. param sr The service record handle specifies the service record from which attribute values are to be retrieved. param pattern The attribute search pattern is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. The pattern buffer must be valid for the duration of the search operation, i.e. until c callback is called for the first time. To define a... more
	bt_sdp_request_service_search	brief Search service records ingroup sdp details This function locates service records on a remote SDP server that match the given service search pattern. param channel The L2CAP channel used to communicate to the remote SDP server. param pattern The service search pattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12. The pattern buffer must be valid for the duration of the search operation, i.e. until c callback is called. To define a data element sequence... more

Macros

	Name	Description
	SDP_CLIENT_EVT_CONNECTED	This is macro SDP_CLIENT_EVT_CONNECTED.
	SDP_CLIENT_EVT_CONNECTION_FAILED	This is macro SDP_CLIENT_EVT_CONNECTION_FAILED.
	SDP_CLIENT_EVT_DISCONNECTED	This is macro SDP_CLIENT_EVT_DISCONNECTED.
	SDP_CLIENT_EVT_NULL	This is macro SDP_CLIENT_EVT_NULL.

Structures

	Name	Description
	bt_sdp_client_evt_connected_t	This is record bt_sdp_client_evt_connected_t.
	bt_sdp_client_evt_disconnected_t	This is record bt_sdp_client_evt_disconnected_t.

Types

	Name	Description
	bt_sdp_client_callback_fp	This is type bt_sdp_client_callback_fp.
	bt_sdp_read_de_callback_ftp	This is type bt_sdp_read_de_callback_ftp.
	bt_sdp_service_attribute_callback_fp	This is type bt_sdp_service_attribute_callback_fp.
	bt_sdp_service_search_callback_fp	This is type bt_sdp_service_search_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

sdp_config.h

Macros

	Name	Description
	__SDP_CONFIG_H	This is macro __SDP_CONFIG_H.

	SDP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	• defgroup sdp_config Configuration
		• ingroup sdp
		*
		• This module describes parameters used to configure SDP.
		*
		• dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.
		* code #include "cdbt/bt/bt_std.h"
		// HCI and L2CAP must always be present // SDP is required only if stack is running in dual mode. This is the default mode. // To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol // must be defined: // #define BT_BLE_SINGLE_MODE
		// HCI configuration parameters #define... more

	SDP_ALLOCATE_BUFFERS_VARS	This is macro SDP_ALLOCATE_BUFFERS_VARS.
	SDP_MAX_ATTRIBUTE_RESULT_LEN	brief Maximum number of attributes to find ingroup sdp_config details This parameter defines the maximum number of attributes withing a service record the SDP server will return to the client.
	SDP_MAX_PDU_BUFFERS	brief Maximum number of SDP server PDU buffers. ingroup sdp_config details This parameter defines the maximum number of responses the SDP server can send at the same time.
	SDP_MAX_SEARCH_RESULT_LEN	brief Maximum number of service records to find. ingroup sdp_config details This parameter defines the maximum number of service records the SDP server will return to the client.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

sdp_packet.h

Functions

	Name	Description
	bt_sdp_packet_assembler	This is function bt_sdp_packet_assembler.

Structures

	Name	Description
	_bt_sdp_packet_t	This is type bt_sdp_packet_t.
	bt_sdp_packet_t	This is type bt_sdp_packet_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

sdp_private.h

Functions

	Name	Description
	_bt_sdp_client_init	This is function _bt_sdp_client_init.

	_sdp_alloc_svc_tran_buffer	This is function _sdp_alloc_svc_tran_buffer.
	_sdp_alloc_tran_buffer	This is function _sdp_alloc_tran_buffer.
	_sdp_find_svc_transaction	This is function _sdp_find_svc_transaction.
	_sdp_find_transaction	This is function _sdp_find_transaction.
	_sdp_free_svc_tran_buffer	This is function _sdp_free_svc_tran_buffer.
	_sdp_free_tran_buffer	This is function _sdp_free_tran_buffer.
	_sdp_init_tran_buffers	From sdp_tran_buffer.c
	bt_sdp_read_attribute	From sdp_db_utils.c
	sdp_compare_uuid_de	This is function sdp_compare_uuid_de.
	sdp_find_attributes	This is function sdp_find_attributes.
	sdp_find_service_records2	From sdp_utils.c

Structures

	Name	Description
	_bt_sdp_server_attribute_t	This is type bt_sdp_server_attribute_t.
	_bt_sdp_server_data_element_t	Private types
	_bt_sdp_server_record_t	This is type bt_sdp_server_record_t.
	bt_sdp_server_attribute_t	This is type bt_sdp_server_attribute_t.
	bt_sdp_server_data_element_t	Private types
	bt_sdp_server_record_t	This is type bt_sdp_server_record_t.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

sdp_utils.h

Functions

	Name	Description
	_sdp_get_de_data_len	This is function _sdp_get_de_data_len.
	_sdp_get_de_hdr_len	This is function _sdp_get_de_hdr_len.
	_sdp_read_de_header	This is function _sdp_read_de_header.
	_sdp_write_data_element	This is function _sdp_write_data_element.
	bt_sdp_de_to_uuid	This is function bt_sdp_de_to_uuid.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

spp.h

Enumerations

	Name	Description
	_bt_spp_port_event_e	• Serial port event. Values of this enumeration represent serial port events. Events are reported to the application through a callback function. The callback function is specified when the port is allocated using bt_spp_allocate().
	_bt_spp_port_state_e	• Serial port state. Values of this enumeration represent states of the serial port.
	_bt_spp_send_status_e	Send operation status.

	bt_spp_port_event_e	• Serial port event. Values of this enumeration represent serial port events. Events are reported to the application through a callback function. The callback function is specified when the port is allocated using bt_spp_allocate().
	bt_spp_port_state_e	• Serial port state. Values of this enumeration represent states of the serial port.
	bt_spp_send_status_e	Send operation status.

Functions

	Name	Description
	bt_spp_allocate	• Allocate a serial port. The returned serial port is initially in the ::SPP_PORT_STATE_DISCONNECTED state. To establish a connection with a remote device, call bt_spp_connect(). To listen for incoming connections from other devices, call bt_spp_listen(). The p callback parameter must specify a callback function that will be used to notify about serial port events and state changes. When the port is not needed any more it must be deallocated by bt_spp_deallocate(). The maximum number of serial ports that can be allocated simultaneously is specified by the ::SPP_MAX_PORTS configuration parameter. @param l2cap_mgr L2CAP manager. @param callback Pointer to... more
	bt_spp_cancel_listen	• Stop listening for incoming connections. This function stops the port to accept incoming connections on a given server channel. @param port Serial port. @param channel The RFCOMM server channel to accept connections on. @return c TRUE if successful, c FALSE otherwise.
	bt_spp_cancel_receive	• Cancel receive data. If a receive operation is currently in progress this function will cancel it. After calling this function the receive callback specified in bt_spp_receive() will not be called. If there is no receive operation in progress calling this function has no effect. @param port Serial port.
	bt_spp_cancel_send	• Cancel send data. If a send operation is currently in progress this function will try to cancel it. When the operation is canceled the send callback function will be called with the ::SPP_SEND_STATUS_CANCELED status. If this function is called but the active send operation completes successfuly before the stack can actually cancel it the call back function will still be called with the ::SPP_SEND_STATUS_CANCELED status. If there is no send operation in progess calling this function has no effect. @param port Serial port.
	bt_spp_clr_port_options	This is function bt_spp_clr_port_options.
	bt_spp_connect	• Connect to a remote device. This function initiates a connection to a remote device. When the connection is successfully established the port's callback is called with the ::SPP_PORT_EVENT_CONNECTED event. If connection fails the callback is called with the ::SPP_PORT_EVENT_CONNECTION_FAILED event. The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail. @param port Serial port. @param remote_addr Bluetooth address of the remote device. @param channel RFCOMM server channel on which the connection is to be established. @return c TRUE if successful, c FALSE otherwise.
	bt_spp_deallocate	• Deallocate serial port. This function deallocates the specified port structure and other resources associated with it. The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail. If the function completes successfully the application must not try to access any fields in the structure and must not use it with any other SPP functions. Also, it becomes available for subsequent allocation by bt_spp_port_allocate(). @param port Serial port structure to deallocate. @return c TRUE if successful, c FALSE otherwise.

	bt_spp_disconnect	• Disconnect from the remote device. This function initiates the disconnection process. When it is complete the the port's callback is called with the SPP_PORT_EVENT_DISCONNECTED event. If the port is already in the disconnected state the function does nothing and the callback is not called. @param port Serial port.
	bt_spp_find_server	This is function bt_spp_find_server.
	bt_spp_find_server_ex	This is function bt_spp_find_server_ex.
	bt_spp_get_frame_length	• Get frame length. This function returns the RFCOMM frame length used by the RFCOMM protocol. The frame length depends on configuration of DotStack and configuration of the Bluetooth stack running on the remote device. In order to achieve maximum throughput over the serial port connection the application should send and receive data in chunks that are multiple of this frame length. @return RFCOMM frame length in bytes.
	bt_spp_get_hci_connection	• Get SPP port's ACL connection. This function returns a pinter to the structure that describes the ACL connection this port is on. @return Pointer to ACL connection description if the port is connected, NULL otherwise.
	bt_spp_get_local_modem_status	• Get local device's TS 07.10 control signals. This function returns current state of the local device's TS 07.10 controls signals. The signals are defined as a mask of the following constants: SPP_RS232_DSR SPP_RS232_RTS SPP_RS232_RI SPP_RS232_DCD @param port Serial port. @return local device's TS 07.10 control signals.
	bt_spp_get_remote_address	brief Get the address of the remote device this device is connected to. ingroup spp param port Serial port. return li c A pointer to bt_bdaddr structure that contains the address of the remote device.
	bt_spp_get_remote_modem_status	• Get remote device's TS 07.10 control signals. This function returns current state of the remote device's V.24 controls signals. The signals are defined as a mask of the following constants: SPP_RS232_DTR SPP_RS232_CTS SPP_RS232_RI SPP_RS232_DCD @param port Serial port. @return remote device's TS 07.10 control signals.
	bt_spp_init	• Initialize the SPP module. This function initializes all internal variables of the SPP module. It must be called prior to using any other functions in this module.
	bt_spp_listen	• Listen for incoming connections. This function registers the port to accept incoming connections from remote devices on a particular RFCOMM server channel. The specified server channel should be listed in the SDP database. Otherwise, remote devices will not be able to find out which server channel to use. When a remote device successfully establishes a connection on the specified port the port's callback is called with the ::SPP_PORT_EVENT_CONNECTED event. The port must be in ::SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail. @param port Serial port. @param channel The RFCOMM server channel on which to listen for connections. @return c TRUE... more
	bt_spp_receive	• Receive data. This function receives data from the serial port connection. The caller must provide a buffer and a callback function. Whenever the port receives data they are copied to the provided buffer and the callback function is called. The callback function is passed the length of received data and the same callback parameter that was specified when the port was allocated with bt_spp_allocate(). This function does not wait until the buffer is filled out completely. Any amount of received data will complete the operation. The port must be in ::SPP_PORT_STATE_CONNECTED state. Otherwise, the function will fail. Also, the... more

	bt_spp_send	• Send data. This function starts sending data over the serial port connection. Along with the data the caller must provide a callback function that is called when all data has been sent. Also, during execution of this operation the port's state callback function is called periodically with the ::SPP_PORT_EVENT_SEND_PROGRESS event. The port must be in ::SPP_PORT_STATE_CONNECTED state. Otherwise, the function will fail. Also, the function will fail if a previously started send operation is still in progress. The callback function is passed the same callback parameter that was specified when the port was allocated with bt_spp_allocate(). @param port Serial... more
	bt_spp_set_dtr	• Set local device's RS-232 DTR signal. Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM_MODEM_STATUS_RTC signal. If there are resources to send a command to the remote device this command will return TRUE. If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported. If the status cannot be changed because there is no resources to send a... more
	bt_spp_set_local_modem_status	• Set local device's TS 07.10 control signals. Changes the state of local device's TS 07.10 control signals and notifies the remote device of the change. If there are resources to send a command to the remote device this command will return TRUE. If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported. If the status cannot be changed because there is no resources to send a command to the remote device this function... more
	bt_spp_set_port_options	This is function bt_spp_set_port_options.
	bt_spp_set_rts	• Set local device's RS-232 RTS signal. Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM_MODEM_STATUS_RTR signal. If there are resources to send a command to the remote device this command will return TRUE. If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported. If the status cannot be changed because there is no resources to send a... more

Macros

	Name	Description
	SPP_PORT_OPTION_ENCRYPTED	This is macro SPP_PORT_OPTION_ENCRYPTED.
	SPP_PORT_OPTION_MASTER	This is macro SPP_PORT_OPTION_MASTER.
	SPP_PORT_OPTION_SECURE	This is macro SPP_PORT_OPTION_SECURE.
	SPP_PORT_TYPE_INCOMING	This is macro SPP_PORT_TYPE_INCOMING.
	SPP_PORT_TYPE_OUTGOING	This is macro SPP_PORT_TYPE_OUTGOING.
	SPP_RS232_CTS	This is macro SPP_RS232_CTS.
	SPP_RS232_DCD	This is macro SPP_RS232_DCD.

	SPP_RS232_DSR	• @defgroup spp Serial Port Profile (SPP) The DotStack SPP API is a simple API for communicating over a Bluetooth link using the Bluetooth Serial Port Profile. Here are the steps for using this API: • Call the bt_spp_init() function. • Allocate a serial port structure with bt_spp_allocate(). One of the parameters to this function is a pointer to a callback function. That callback function will be called by the stack whenever the state of the serial port changes. • To connect to a remote device call bt_spp_connect(). The stack will notify when the connection is established by calling the state... more
	SPP_RS232_DTR	This is macro SPP_RS232_DTR.
	SPP_RS232_RI	This is macro SPP_RS232_RI.
	SPP_RS232_RTS	This is macro SPP_RS232_RTS.

Structures

	Name	Description
	_bt_spp_port_t	• Serial port structure. This structure represents a Bluetooth serial port. Application code may only use those fields that are documented. The rest of the fields are private to the SPP implementation.

Types

	Name	Description
	bt_spp_find_server_callback_fp	This is type bt_spp_find_server_callback_fp.
	bt_spp_port_t	This is type bt_spp_port_t.
	bt_spp_receive_callback_fp	• Serial port receive callback. This callback function is called when a receive operation initiated by bt_spp_receive() completes. @param port Serial port on which the receive operation completed. @param bytes_received Number of received bytes. @param param Callback parameter that was specified when bt_spp_allocate() was called.
	bt_spp_send_callback_fp	• Serial port send callback. This callback function is called when a send operation initiated by bt_spp_send() completes. @param port Serial port on which the send operation completed. @param bytes_sent Number of bytes sent. This parameter is just a convenience as it always specifies the same number of bytes that was passed to bt_spp_send(); @param status Completion status. It is one of the values defined in the ::bt_spp_send_status_e enumeration. @param param Callback parameter that was specified when bt_spp_allocate() was called.
	bt_spp_state_callback_fp	• Serial port state callback. This callback function is called whenever the state of a serial port is changed. @param port Serial port which state has changed. @param evt Event describing the nature of state change. It is one of the values defined in the ::bt_spp_port_event_e enumeration. @param param Callback parameter that was specified when bt_spp_allocate() was called.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

spp_config.h

Macros

	Name	Description
	__SPP_CONFIG_H	This is macro __SPP_CONFIG_H.
	SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR	This is macro SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR.

	SPP_ALLOCATE_BUFFERS_VARS	This is macro SPP_ALLOCATE_BUFFERS_VARS.
	SPP_DECLARE_FRAME_BUFFERS	This is macro SPP_DECLARE_FRAME_BUFFERS.
	SPP_DISABLE_BUFFERING	This is macro SPP_DISABLE_BUFFERING.
	SPP_FRAME_BUFFERS_RAM_SIZE	This is macro SPP_FRAME_BUFFERS_RAM_SIZE.
	SPP_FRAME_BUFFERS_SIZE	This is macro SPP_FRAME_BUFFERS_SIZE.
	SPP_MAX_PORTS	brief Maximum number of SPP ports. ingroup spp_config details This parameter defines the maximum number of SPP port that can be open between the local and remote devices. If RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is FALSE (default) this value should be equal to RFCOMM_MAX_SERVER_CHANNELS. If RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is TRUE this value should be between RFCOMM_MAX_SERVER_CHANNELS and RFCOMM_MAX_SERVER_CHANNELS * RFCOMM_MAX_SESSIONS.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

spp_private.h

Functions

	Name	Description
	_bt_spp_client_init	This is function _bt_spp_client_init.
	_bt_spp_find_port	This is function _bt_spp_find_port.
	_bt_spp_handle_rx	This is function _bt_spp_handle_rx.
	_bt_spp_handle_tx	This is function _bt_spp_handle_tx.
	_bt_spp_rfcomm_read_data_callback	This is function _bt_spp_rfcomm_read_data_callback.
	_bt_spp_rfcomm_send_data_callback	This is function _bt_spp_rfcomm_send_data_callback.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

ssp.h

Enumerations

	Name	Description
	_SSP_AUTHENTICATION_REQUIREMENTS	This is type SSP_AUTHENTICATION_REQUIREMENTS.
	_SSP_IO_CAPABILITY	This is type SSP_IO_CAPABILITY.
	_SSP_KEYPRESS_NOTIFICATION_TYPE	This is type SSP_KEYPRESS_NOTIFICATION_TYPE.
	_SSP_MODE	This is type SSP_MODE.
	_SSP_OOB_DATA_PRESENT	This is type SSP_OOB_DATA_PRESENT.
	SSP_AUTHENTICATION_REQUIREMENTS	This is type SSP_AUTHENTICATION_REQUIREMENTS.
	SSP_IO_CAPABILITY	This is type SSP_IO_CAPABILITY.
	SSP_KEYPRESS_NOTIFICATION_TYPE	This is type SSP_KEYPRESS_NOTIFICATION_TYPE.
	SSP_MODE	This is type SSP_MODE.
	SSP_OOB_DATA_PRESENT	This is type SSP_OOB_DATA_PRESENT.

Functions

	Name	Description
	bt_ssp_evt_handler	This is function bt_ssp_evt_handler.

	bt_ssp_init	This is function bt_ssp_init.
	bt_ssp_read_local_oob_data	This is function bt_ssp_read_local_oob_data.
	bt_ssp_send_keypress_notification	This is function bt_ssp_send_keypress_notification.
	bt_ssp_send_user_confirmation	This is function bt_ssp_send_user_confirmation.
	bt_ssp_send_user_passkey	This is function bt_ssp_send_user_passkey.
	bt_ssp_set_io_capabilities	This is function bt_ssp_set_io_capabilities.
	bt_ssp_set_mode	This is function bt_ssp_set_mode.
	bt_ssp_set_oob_data	This is function bt_ssp_set_oob_data.

Macros

	Name	Description
	OOB_DATA_HASH_LENGTH	Remote OOB Data Request Event
	OOB_DATA_RANDOMIZER_LENGTH	This is macro OOB_DATA_RANDOMIZER_LENGTH.
	SSP_MAX_MANAGERS	This is macro SSP_MAX_MANAGERS.

Structures

	Name	Description
	_bt_ssp_io_capability	This is type bt_ssp_io_capability.
	_bt_ssp_keypress_notification	Keypress Notification Event
	_bt_ssp_oob_data	This is type bt_ssp_oob_data.
	_bt_ssp_simple_pairing_complete	Simple Pairing Complete Event
	_bt_ssp_user_confirmation_request	User Confirmation Request Event
	_bt_ssp_user_passkey_notification	User Passkey Notification Event
	_bt_ssp_user_passkey_request	User Passkey Request Event
	bt_ssp_io_capability	This is type bt_ssp_io_capability.
	bt_ssp_keypress_notification	Keypress Notification Event
	bt_ssp_oob_data	This is type bt_ssp_oob_data.
	bt_ssp_simple_pairing_complete	Simple Pairing Complete Event
	bt_ssp_user_confirmation_request	User Confirmation Request Event
	bt_ssp_user_passkey_notification	User Passkey Notification Event
	bt_ssp_user_passkey_request	User Passkey Request Event

Types

	Name	Description
	bt_spp_read_local_oob_data_callback_fp	This is type bt_spp_read_local_oob_data_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

ssp_event.h

Enumerations

	Name	Description
	_SSP_EVENT	This is type SSP_EVENT.
	SSP_EVENT	This is type SSP_EVENT.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

ssp_event_handler.h

Functions

	Name	Description
	ssp_evt_io_capability_request	This is function ssp_evt_io_capability_request.
	ssp_evt_io_capability_response	This is function ssp_evt_io_capability_response.
	ssp_evt_keypress_notification	This is function ssp_evt_keypress_notification.
	ssp_evt_oob_data_request	This is function ssp_evt_oob_data_request.
	ssp_evt_ssp_complete	This is function ssp_evt_ssp_complete.
	ssp_evt_user_confirmation_request	This is function ssp_evt_user_confirmation_request.
	ssp_evt_user_passkey_notification	This is function ssp_evt_user_passkey_notification.
	ssp_evt_user_passkey_request	This is function ssp_evt_user_passkey_request.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

ti.h

Functions

	Name	Description
	btx_ti_a3dp_sink_close_stream	This is function btx_ti_a3dp_sink_close_stream.
	btx_ti_a3dp_sink_codec_config	This is function btx_ti_a3dp_sink_codec_config.
	btx_ti_a3dp_sink_open_stream	This is function btx_ti_a3dp_sink_open_stream.
	btx_ti_a3dp_sink_start_stream	This is function btx_ti_a3dp_sink_start_stream.
	btx_ti_a3dp_sink_stop_stream	This is function btx_ti_a3dp_sink_stop_stream.
	btx_ti_avpr_debug	This is function btx_ti_avpr_debug.
	btx_ti_avpr_enable	This is function btx_ti_avpr_enable.
	btx_ti_drpb_enable_rf_calibration	This is function btx_ti_drpb_enable_rf_calibration.
	btx_ti_drpb_set_power_vector	This is function btx_ti_drpb_set_power_vector.
	btx_ti_drpb_tester_con_tx	This is function btx_ti_drpb_tester_con_tx.
	btx_ti_enable_deep_sleep	This is function btx_ti_enable_deep_sleep.
	btx_ti_enable_fast_clock_crystal	This is function btx_ti_enable_fast_clock_crystal.
	btx_ti_enable_low_power_scan	This is function btx_ti_enable_low_power_scan.
	btx_ti_enable_low_power_scan_default	This is function btx_ti_enable_low_power_scan_default.
	btx_ti_exec_script	This is function btx_ti_exec_script.
	btx_ti_exec_script_oem	This is function btx_ti_exec_script_oem.
	btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_36	CC2560 Scripts (Service pack 2.36)
	btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_44	CC2560 Scripts (Service pack 2.44)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_10	CC2564 Scripts (Service pack 2.10)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_10_BLE_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_10_BLE_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_12	CC2564 Scripts (Service pack 2.12)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_12_BLE_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_12_BLE_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_14	CC2564 Scripts (Service pack 2.14)
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_7	CC2564 Scripts (Service pack 2.7)

	btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_AVPR_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_AVPR_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_BLE_AddOn	This is function btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_BLE_AddOn.
	btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_Short	CC2564 Scripts (Service pack 2.8)
	btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_AVPR_AddOn	This is function btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_AVPR_AddOn.
	btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_BLE_AddOn	This is function btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_0_1	CC2564B Scripts (Service pack 0.1)
	btx_ti_get_script__CC2564B_BT_Service_Pack_0_1_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_0_1_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_0_2	CC2564B Scripts (Service pack 0.2)
	btx_ti_get_script__CC2564B_BT_Service_Pack_0_2_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_0_2_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_0	CC2564B Scripts (Service pack 1.0)
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_0_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_0_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_1	CC2564B Scripts (Service pack 1.1)
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_1_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_1_BLE_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_2	CC2564B Scripts (Service pack 1.2)
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn.
	btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn	This is function btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_3	CC2564 Scripts (Service pack 2.3)
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_AVPR_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_AVPR_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_BLE_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_BLE_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_DC2DC_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_DC2DC_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_4	CC2564 Scripts (Service pack 2.4)
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_AVPR_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_AVPR_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_BLE_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_BLE_AddOn.
	btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_DC2DC_AddOn	This is function btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_DC2DC_AddOn.
	btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3	CC2564 Scripts (Service pack 1.3)
	btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3_BLE_Init	This is function btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3_BLE_Init.
	btx_ti_init_ble_controller	This is function btx_ti_init_ble_controller.
	btx_ti_init_controller	This is function btx_ti_init_controller.
	btx_ti_le_enable	This is function btx_ti_le_enable.
	btx_ti_set_afh_mode	This is function btx_ti_set_afh_mode.

	btx_ti_set_uart_baud_rate	This is function btx_ti_set_uart_baud_rate.
	btx_ti_write_bdaddr	This is function btx_ti_write_bdaddr.
	btx_ti_write_codec_config	This is function btx_ti_write_codec_config.
	btx_ti_write_hardware_register	This is function btx_ti_write_hardware_register.

Macros

	Name	Description
	BTX_TI_A3DP_ROLE_SINK	This is macro BTX_TI_A3DP_ROLE_SINK.
	BTX_TI_A3DP_ROLE_SOURCE	This is macro BTX_TI_A3DP_ROLE_SOURCE.
	BTX_TI_AVPR_DISABLE	This is macro BTX_TI_AVPR_DISABLE.
	BTX_TI_AVPR_DO_NOT_LOAD_A3DP_CODE	This is macro BTX_TI_AVPR_DO_NOT_LOAD_A3DP_CODE.
	BTX_TI_AVPR_ENABLE	This is macro BTX_TI_AVPR_ENABLE.
	BTX_TI_AVPR_LOAD_A3DP_CODE	This is macro BTX_TI_AVPR_LOAD_A3DP_CODE.
	BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE	This is macro BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE.
	btx_ti_get_script__CC2560_ServicePack	Alias for the latest script
	btx_ti_get_script__CC2564_BLE_Init	This is macro btx_ti_get_script__CC2564_BLE_Init.
	btx_ti_get_script__CC2564_ServicePack	Aliases for latest scripts
	btx_ti_get_script__CC2564B_AVPR_Init	This is macro btx_ti_get_script__CC2564B_AVPR_Init.
	btx_ti_get_script__CC2564B_BLE_Init	This is macro btx_ti_get_script__CC2564B_BLE_Init.
	btx_ti_get_script__CC2564B_ServicePack	Aliases for latest scripts
	BTX_TI_MODULATION_SCHEME_8_DPSK	This is macro BTX_TI_MODULATION_SCHEME_8_DPSK.
	BTX_TI_MODULATION_SCHEME_CW	This is macro BTX_TI_MODULATION_SCHEME_CW.
	BTX_TI_MODULATION_SCHEME_GFSK	This is macro BTX_TI_MODULATION_SCHEME_GFSK.
	BTX_TI_MODULATION_SCHEME_P4_DPSK	This is macro BTX_TI_MODULATION_SCHEME_P4_DPSK.
	BTX_TI_MODULATION_TYPE_EDR2	This is macro BTX_TI_MODULATION_TYPE_EDR2.
	BTX_TI_MODULATION_TYPE_EDR3	This is macro BTX_TI_MODULATION_TYPE_EDR3.
	BTX_TI_MODULATION_TYPE_GFSK	This is macro BTX_TI_MODULATION_TYPE_GFSK.
	BTX_TI_SBC_ALLOCATION_METHOD_LOUDNESS	This is macro BTX_TI_SBC_ALLOCATION_METHOD_LOUDNESS.
	BTX_TI_SBC_ALLOCATION_METHOD_SNR	This is macro BTX_TI_SBC_ALLOCATION_METHOD_SNR.
	BTX_TI_SBC_BLOCKS_12	This is macro BTX_TI_SBC_BLOCKS_12.
	BTX_TI_SBC_BLOCKS_16	This is macro BTX_TI_SBC_BLOCKS_16.
	BTX_TI_SBC_BLOCKS_4	This is macro BTX_TI_SBC_BLOCKS_4.
	BTX_TI_SBC_BLOCKS_8	This is macro BTX_TI_SBC_BLOCKS_8.
	BTX_TI_SBC_CHANNEL_MODE_DUAL_CHANNEL	This is macro BTX_TI_SBC_CHANNEL_MODE_DUAL_CHANNEL.
	BTX_TI_SBC_CHANNEL_MODE_JOINT_STEREO	This is macro BTX_TI_SBC_CHANNEL_MODE_JOINT_STEREO.
	BTX_TI_SBC_CHANNEL_MODE_MONO	This is macro BTX_TI_SBC_CHANNEL_MODE_MONO.
	BTX_TI_SBC_CHANNEL_MODE_STEREO	This is macro BTX_TI_SBC_CHANNEL_MODE_STEREO.
	BTX_TI_SBC_SAMPLE_FREQUENCY_16000	This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_16000.
	BTX_TI_SBC_SAMPLE_FREQUENCY_32000	This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_32000.
	BTX_TI_SBC_SAMPLE_FREQUENCY_44100	This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_44100.
	BTX_TI_SBC_SAMPLE_FREQUENCY_48000	This is macro BTX_TI_SBC_SAMPLE_FREQUENCY_48000.
	BTX_TI_TEST_PATTERN_ALL_0	This is macro BTX_TI_TEST_PATTERN_ALL_0.
	BTX_TI_TEST_PATTERN_ALL_1	This is macro BTX_TI_TEST_PATTERN_ALL_1.
	BTX_TI_TEST_PATTERN_F0F0	This is macro BTX_TI_TEST_PATTERN_F0F0.
	BTX_TI_TEST_PATTERN_FF00	This is macro BTX_TI_TEST_PATTERN_FF00.
	BTX_TI_TEST_PATTERN_PN15	This is macro BTX_TI_TEST_PATTERN_PN15.
	BTX_TI_TEST_PATTERN_PN9	This is macro BTX_TI_TEST_PATTERN_PN9.
	BTX_TI_TEST_PATTERN_USER	This is macro BTX_TI_TEST_PATTERN_USER.
	BTX_TI_TEST_PATTERN_Z0Z0	This is macro BTX_TI_TEST_PATTERN_Z0Z0.

Structures

	Name	Description
	_btx_ti_codec_config_s	This is type btx_ti_codec_config_t.
	_btx_ti_exec_script_buffer_t	This is record _btx_ti_exec_script_buffer_t.
	_btx_ti_exec_script_oem_buffer_t	This is record _btx_ti_exec_script_oem_buffer_t.

	_btx_ti_script_t	This is type btx_ti_script_t.
	btx_ti_codec_config_t	This is type btx_ti_codec_config_t.
	btx_ti_script_t	This is type btx_ti_script_t.

Types

	Name	Description
	btx_ti_completion_callback_fp	This is type btx_ti_completion_callback_fp.
	btx_ti_exec_script_buffer_t	This is type btx_ti_exec_script_buffer_t.
	btx_ti_exec_script_oem_buffer_t	This is type btx_ti_exec_script_oem_buffer_t.
	btx_ti_exec_script_oem_callback_fp	This is type btx_ti_exec_script_oem_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

ti_private.h

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

types.h

Macros

	Name	Description
	BYTE_SIZE	This is macro BYTE_SIZE.

Types

	Name	Description
	bt_byte	This is type bt_byte.
	bt_int	This is type bt_int.
	bt_long	This is type bt_long.
	bt_uint	This is type bt_uint.
	bt_ulong	This is type bt_ulong.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC.

Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

vcard_parser.h

Functions

	Name	Description
	bt_vcard_parse_fragment	This is function bt_vcard_parse_fragment.
	bt_vcard_parser_reset	This is function bt_vcard_parser_reset.

Macros

	Name	Description
	VCARD_CALL_TYPE_DIALED	This is macro VCARD_CALL_TYPE_DIALED.
	VCARD_CALL_TYPE_MISSED	This is macro VCARD_CALL_TYPE_MISSED.
	VCARD_CALL_TYPE_RECEIVED	This is macro VCARD_CALL_TYPE_RECEIVED.
	VCARD_EVT_PROPERTY_PARAM	This is macro VCARD_EVT_PROPERTY_PARAM.
	VCARD_EVT_PROPERTY_STARTED	This is macro VCARD_EVT_PROPERTY_STARTED.
	VCARD_EVT_PROPERTY_VALUE	This is macro VCARD_EVT_PROPERTY_VALUE.
	VCARD_EVT_VCARD_ENDED	This is macro VCARD_EVT_VCARD_ENDED.
	VCARD_EVT_VCARD_STARTED	This is macro VCARD_EVT_VCARD_STARTED.

	VCARD_PARAM_CALL_TYPE	This is macro VCARD_PARAM_CALL_TYPE.
	VCARD_PARAM_CELL	This is macro VCARD_PARAM_CELL.
	VCARD_PARAM_DIALED	This is macro VCARD_PARAM_DIALED.
	VCARD_PARAM_ENCODING	This is macro VCARD_PARAM_ENCODING.
	VCARD_PARAM_HOME	This is macro VCARD_PARAM_HOME.
	VCARD_PARAM_LANGUAGE	This is macro VCARD_PARAM_LANGUAGE.
	VCARD_PARAM_MISSED	This is macro VCARD_PARAM_MISSED.
	VCARD_PARAM_RECEIVED	This is macro VCARD_PARAM_RECEIVED.
	VCARD_PARAM_TYPE	This is macro VCARD_PARAM_TYPE.
	VCARD_PARAM_VALUE	This is macro VCARD_PARAM_VALUE.
	VCARD_PARAM_WORK	This is macro VCARD_PARAM_WORK.
	VCARD_PARSER_STATE_READ_PARAM_NAME	This is macro VCARD_PARSER_STATE_READ_PARAM_NAME.
	VCARD_PARSER_STATE_READ_PARAM_VALUE	This is macro VCARD_PARSER_STATE_READ_PARAM_VALUE.
	VCARD_PARSER_STATE_READ_TYPE_NAME	This is macro VCARD_PARSER_STATE_READ_TYPE_NAME.
	VCARD_PARSER_STATE_READ_TYPE_VALUE	This is macro VCARD_PARSER_STATE_READ_TYPE_VALUE.
	VCARD_PARSER_STATE_SKIP_PARAM	This is macro VCARD_PARSER_STATE_SKIP_PARAM.
	VCARD_PARSER_STATE_SKIP_TYPE	This is macro VCARD_PARSER_STATE_SKIP_TYPE.
	VCARD_PROPERTY_ADR	Delivery Address
	VCARD_PROPERTY_AGENT	vCard of Person Representing
	VCARD_PROPERTY_BDAY	Birthday
	VCARD_PROPERTY_BEGIN	This is macro VCARD_PROPERTY_BEGIN.
	VCARD_PROPERTY_CATEGORIES	Categories
	VCARD_PROPERTY_CLASS	Class information
	VCARD_PROPERTY_EMAIL	Electronic Mail Address
	VCARD_PROPERTY_END	This is macro VCARD_PROPERTY_END.
	VCARD_PROPERTY_FN	Formatted Name
	VCARD_PROPERTY_GEO	Geographic Position
	VCARD_PROPERTY_KEY	Public Encryption Key
	VCARD_PROPERTY_LABEL	Delivery
	VCARD_PROPERTY_LOGO	Organization Logo
	VCARD_PROPERTY_MAILER	Electronic Mail
	VCARD_PROPERTY_N	Structured Presentation of Name
	VCARD_PROPERTY_NICKNAME	Nickname
	VCARD_PROPERTY_NOTE	Comments
	VCARD_PROPERTY_ORG	Name of Organization
	VCARD_PROPERTY_PHOTO	Associated Image or Photo
	VCARD_PROPERTY_PROID	Product ID
	VCARD_PROPERTY_REV	Revision
	VCARD_PROPERTY_ROLE	Role within the Organization
	VCARD_PROPERTY_SORT_STRING	String used for sorting operations
	VCARD_PROPERTY_SOUND	Pronunciation of Name
	VCARD_PROPERTY_TEL	Telephone Number
	VCARD_PROPERTY_TITLE	Job
	VCARD_PROPERTY_TZ	Time Zone
	VCARD_PROPERTY_UID	Unique ID
	VCARD_PROPERTY_URL	Uniform Resource Locator
	VCARD_PROPERTY_VERSION	vCard Version
	VCARD_PROPERTY_X_IRMC_CALL_DATETIME	Time stamp

Structures

	Name	Description
	_bt_vcard_evt_prop_param_t	This is type bt_vcard_evt_prop_param_t.
	_bt_vcard_evt_prop_t	This is type bt_vcard_evt_prop_t.
	_bt_vcard_parser_t	This is type bt_vcard_parser_t.
	bt_vcard_evt_prop_param_t	This is type bt_vcard_evt_prop_param_t.

	bt_vcard_evt_prop_t	This is type bt_vcard_evt_prop_t.
	bt_vcard_parser_t	This is type bt_vcard_parser_t.

Types

	Name	Description
	bt_vcard_parser_callback_fp	This is type bt_vcard_parser_callback_fp.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

xml_parser.h

Functions

	Name	Description
	bt_xml_parse_fragment	This is function bt_xml_parse_fragment.
	bt_xml_parser_reset	This is function bt_xml_parser_reset.

Macros

	Name	Description
	XML_EVT_ATTR_NAME	This is macro XML_EVT_ATTR_NAME.
	XML_EVT_ATTR_VALUE	This is macro XML_EVT_ATTR_VALUE.
	XML_EVT_TAG_ENDED	This is macro XML_EVT_TAG_ENDED.
	XML_EVT_TAG_STARTED	This is macro XML_EVT_TAG_STARTED.
	XML_PARSER_STATE_READ_ATTR_NAME	This is macro XML_PARSER_STATE_READ_ATTR_NAME.
	XML_PARSER_STATE_READ_ATTR_VALUE	This is macro XML_PARSER_STATE_READ_ATTR_VALUE.
	XML_PARSER_STATE_READ_TAG_NAME	This is macro XML_PARSER_STATE_READ_TAG_NAME.
	XML_PARSER_STATE_READ_TAG_VALUE	This is macro XML_PARSER_STATE_READ_TAG_VALUE.
	XML_PARSER_STATE_SKIP_ATTR	This is macro XML_PARSER_STATE_SKIP_ATTR.
	XML_PARSER_STATE_SKIP_COMMENTS	This is macro XML_PARSER_STATE_SKIP_COMMENTS.
	XML_PARSER_STATE_SKIP_PROC_INST	This is macro XML_PARSER_STATE_SKIP_PROC_INST.
	XML_PARSER_STATE_SKIP_TAG	This is macro XML_PARSER_STATE_SKIP_TAG.
	XML_PARSER_STATE_WAIT_ATTR_NAME	This is macro XML_PARSER_STATE_WAIT_ATTR_NAME.
	XML_PARSER_STATE_WAIT_ATTR_VALUE	This is macro XML_PARSER_STATE_WAIT_ATTR_VALUE.
	XML_PARSER_STATE_WAIT_TAG_CLOSE	This is macro XML_PARSER_STATE_WAIT_TAG_CLOSE.
	XML_PARSER_STATE_WAIT_TAG_NAME	This is macro XML_PARSER_STATE_WAIT_TAG_NAME.

Structures

	Name	Description
	_bt_xml_evt_attribute_t	This is type bt_xml_evt_attribute_t.
	_bt_xml_evt_attribute_value_t	This is type bt_xml_evt_attribute_value_t.
	_bt_xml_evt_tag_started_t	This is type bt_xml_evt_tag_started_t.
	_bt_xml_parser_t	This is type bt_xml_parser_t.
	bt_xml_evt_attribute_t	This is type bt_xml_evt_attribute_t.
	bt_xml_evt_attribute_value_t	This is type bt_xml_evt_attribute_value_t.
	bt_xml_evt_tag_started_t	This is type bt_xml_evt_tag_started_t.
	bt_xml_parser_t	This is type bt_xml_parser_t.

Types

	Name	Description
	bt_xml_parser_callback_pf	This is type bt_xml_parser_callback_pf.

Description

Contains proprietary and confidential information of SEARAN LLC. May not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2011-2016 SEARAN LLC. All Rights Reserved.

SEARAN LLC is the exclusive licensee and developer of dotstack with all its modifications and enhancements.

Contains proprietary and confidential information of CandleDragon and may not be used or disclosed to any other party except in accordance with a license from SEARAN LLC. Copyright (c) 2009, 2010, 2011 CandleDragon. All Rights Reserved.

Bootloader Library Help

This section describes the Bootloader Library that is available in MPLAB Harmony.

Introduction

This library provides a software Bootloader Library that is available on the Microchip family of microcontrollers with a convenient C language interface.

Description

The Bootloader Library can be used to upgrade firmware on a target device without the need for an external programmer or debugger.

A demonstration application, which can be downloaded into the target PIC32 device using the bootloader is included, which provides a personal computer host application to communicate with the bootloader firmware running inside the PIC32 device. This personal computer application is used to perform erase and programming operations.

This library was developed based on the Microchip application note, AN1388 *"PIC32 Bootloader"* (DS01388).

Bootloader Theory

A bootloader is a small application that starts the operation of the device. A bootloader does not fully operate the device, but can perform various functions prior to starting the main application. Such functions can include:

- System checks
- Firmware upgrades
- Application integrity verification
- Starting the application

Using the Library

This topic describes the basic architecture of the Bootloader Library and provides information and examples on its use.

Description

Interface Header File: `bootloader.h`

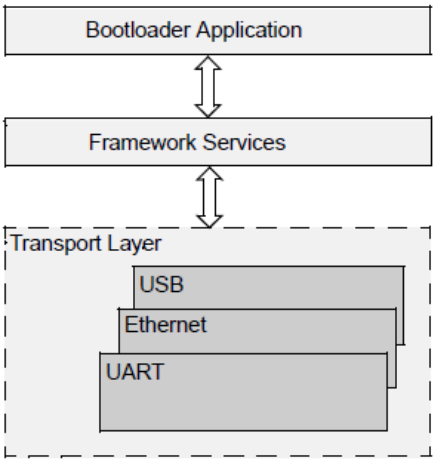
The interface to the Bootloader Library is defined in the `bootloader.h` header file. Any C language source (`.c`) file that uses the Bootloader Library should include `bootloader.h`.

Abstraction Model

This library provides the low-level abstraction of the Bootloader Library module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Bootloader Software Abstraction Block Diagram



Bootloader Application

The main purpose of the Bootloader Application layer is to implement the callback functions necessary to enhance the bootloader. The various callback functions are described below, and are used to handle such tasks as forcing Bootloader mode, erasing and programming memory, and any cleanup before jumping to the application.

Framework Services

The Framework Services provide the main tasks of handling Flash memory. These include:

- Erasing Flash memory
- Programming hex file records into Flash memory
- Computing a CRC check of the Application in Program Memory
- Jumping to the Application

The Framework Services can be enhanced with the callback functions that can handle such things as external memory devices. Doing so requires that the hex files allocate the external memory assignments in a region that would not normally fall into the physical Flash region in the PIC32 device.

The Framework Services take an interface-agnostic approach to the actual communication medium. For the most part, it does not care whether the data comes from a PC Host via USB, UART, or Ethernet, or if it is coming by reading a file from a USB thumb drive or a SD Card. This allows the bootloader to be expanded to handle other communication interfaces with minimal impact to the Framework Services.

Transport Layer

The Transport Layer, called Datastream in the Bootloader Framework code, is used to implement the specific interface to the source of the hex file.

Library Overview

The Library Interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Bootloader Library module.

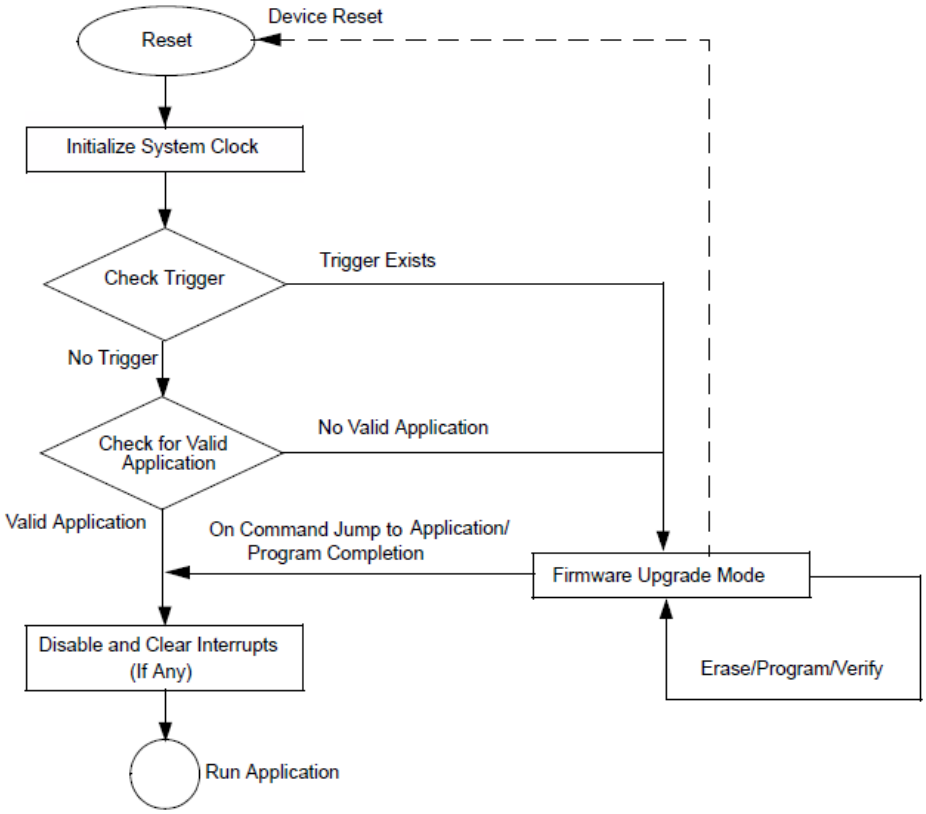
How the Library Works

This topic provides information on how the library works.

Description

The bootloader library is implemented using a framework. The bootloader firmware communicates with the personal computer host application by using a predefined communication protocol.

The following figure illustrates the bootloader architecture. The bootloader framework provides several API functions, which can be called by the bootloader application and the data stream layer. The bootloader framework assists the user to easily modify the bootloader application to adapt to different requirements. Refer to the Library Interface section for the available APIs.



Basic Flow of the Bootloader

Describes the basic flow of the PIC32 Bootloader

Description

The Bootloader code starts executing on a device Reset. If there are no conditions to enter the firmware upgrade mode, the Bootloader starts executing the user application. The Bootloader performs Flash erase/program operations while in the firmware upgrade mode.

Entering the Firmware Upgrade Mode

On a device Reset, the Bootloader forces itself into the firmware upgrade mode if the content of the user application's reset vector address is erased. On PIC32 starter kits, press and hold the switch, SW3, during power-up. While in firmware upgrade mode, the LED labeled LED1 on the PIC32 starter kit will blink.

For your custom bootloader, call the BOOTLOADER_ForceBootloadRegister function to register a custom function. From the callback function, the bootloader can check whatever situation is necessary for forcing the bootloader into Bootloader mode.

Exiting the Firmware Upgrade Mode

For USB HID, Ethernet, or the UART Bootloader, the firmware upgrade mode can be exited either by applying a hard Reset to the device, or by sending a "Jump to Application" command from the PC. For the USB Flash drive Bootloader, the firmware upgrade mode is exited either by a hard Reset or upon completion of firmware programming.

Note: The Bootloader should disable and clear any enabled interrupts before running the user application. The stray interrupts from the Bootloader may interfere with the user application and cause the application to fail. If applicable, interrupts and peripherals should be reinitialized in the user application.

Bootloader Placement in Memory

Describes the Bootloader placement in memory.

Description

The placement of the Bootloader in Flash memory controls several elements of the application and Bootloader interaction. Both have to be placed such that the application will not overwrite the Bootloader, and the Bootloader can properly program the application when it is downloaded.

Figure 1 illustrates two schemes for the Bootloader placement based on the size of the Bootloader. Devices with a large enough Boot Flash

memory can place all of the Bootloader within Boot Flash. Fitting the Bootloader within the Boot Flash memory provides the complete program Flash memory for the user application.

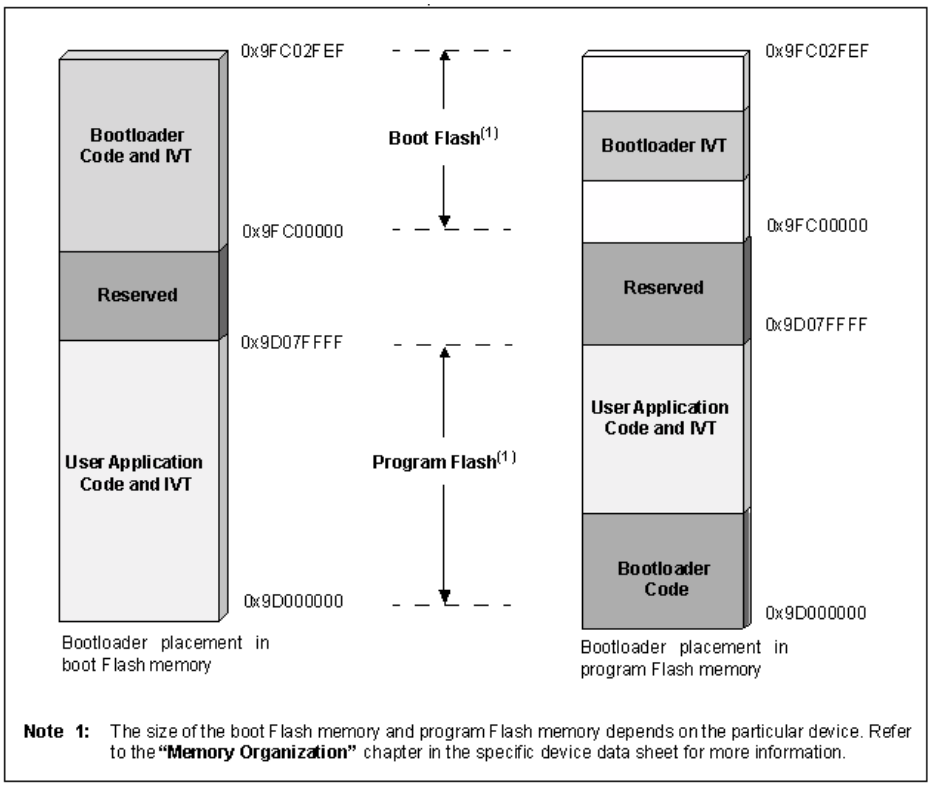
In the case of Bootloaders that exceed the size of PIC32 boot Flash, the Bootloader is split into two parts:

- For devices with 3 KB of Boot Flash, only the reset vector is placed in Boot Flash, with the Interrupt Vector Table (IVT) and C start-up code placed in Program Flash.
- For devices with 12 KB of Boot Flash, the Interrupt Vector Table (IVT) and the C start-up code are placed in Boot Flash, and the remaining portion of the Bootloader is placed inside the Program Flash.

The portion of the Bootloader that resides in Program Flash may be placed at either the beginning or the end of the Program Flash, depending on the needs of the application. The linker scripts generated by Harmony place the Bootloader code at the beginning of Program Flash (0x9D000000).

For PIC32MZ devices, with two 80 KB Boot Flash panels, the Bootloader may or may not fit entirely in one Boot Flash panel. In order to fit some of the Bootloaders, the linker script makes the two Boot Flash panels look like one contiguous Boot Flash memory. Unimplemented areas are blocked using a fill command to the linker.

Figure 1: Bootloader Placement



Handling Device Configuration Bits

Provides information on how device Configuration bits are handled.

Description

The Bootloader does not erase or write the device Configuration words while programming the new application firmware. This is because the device Configuration words settings are shared by both the Bootloader and the user application. Any modification to the device Configuration words may make the Bootloader non-functional. Therefore, it is highly recommended to have common device Configuration words settings for both the user application and the Bootloader.

When combining the Bootloader and Application Hex files in MPLAB X IDE, an error will be generated if the device Configuration words are different. This will be shown as a data conflict error, and the address given will match an address in the device Configuration words. This will indicate which word is in conflict, so that it can be resolved.

The Bootloader's Configuration words can be imported into the Application's MHC configuration file, thus simplifying the process of resolving differences.

Demonstration Application

Provides information on the demonstration applications that can be used with the Bootloader.

Description

Refer to the following sections for information on the demonstrations that are available for the Bootloader:

- *Volume I: Getting Started With MPLAB Harmony > Applications Helps > Bootloader Demonstrations*
- *Volume I: Getting Started With MPLAB Harmony > Applications Help > Examples > Peripheral Library Examples > dma_led_pattern*

PIC32MX Bootloaders

For PIC32MX bootloaders, the MHC configuration for the corresponding application must match the bootloader for the interface in use. MHC will adjust the linker script and certain bootloader definitions in source code to maximize the use of the Flash memory. For example, a UART bootloader must be matched with an application configured for UART bootloading. A USB Device bootloader must be matched with an application configured for USB Device bootloading.

To change the type of bootloader interface the demonstration application is configured for, do the following:

1. Open the dma_led_pattern project in MPLAB X IDE.
2. Select the appropriate configuration for the target device.
3. Open MHC.
4. From the Options tab, expand *Harmony Framework Configuration* > *Bootloader Library*, and select **Use Bootloader Library?**
5. Select the correct interface using the drop-down list next to Bootloader Type.
6. Regenerate the code.
7. Compile the code.
8. The generated `.hex` file can now be used with the corresponding bootloader.

Using the Bootloader Application (UART, USB HID, and Ethernet Bootloaders)

Provides information on using the Bootloader application.

Description

Refer to *Volume I: Getting Started With MPLAB Harmony > Applications Help > Bootloader Demonstrations > Demonstrations > basic > Running the Demonstration* for information on running the demonstration application.

Bootloader Communication Protocol (UART, USB HID, and Ethernet)

Provides information on the Bootloader communication protocol.

Description

The PC host application uses a communication protocol to interact with the Bootloader firmware. The PC host application acts as a master and issues commands to the Bootloader firmware to perform specific operations.

Frame Format

The communication protocol follows the frame format, as shown in Example 1. The frame format remains the same in both directions, that is, from the host application to the Bootloader, and from the Bootloader to the host application.

Example 1: Frame Format

```
[<SOH>…]<SOH>[<DATA>…]<CRCL><CRCH><EOT>
Where:
 <...> Represents a byte
 [...] Represents an optional or variable number of bytes
```

The frame starts with a control character, Start of Header (SOH), and ends with another control character, End of Transmission (EOT). The integrity of the frame is protected by two bytes of Cyclic Redundancy Check (CRC)-16, represented by CRCL (low-byte) and CRCH (high-byte).

Control Characters

Some bytes in the Data field may imitate the control characters, SOH and EOT. The Data Link Escape (DLE) character is used to escape such bytes that could be interpreted as control characters. The Bootloader always accepts the byte following a <DLE> as data, and always sends a <DLE> before any of the control characters.

Table 1: Control Character Descriptions

Control	Hex Value	Description
<SOH>	0x01	Marks the beginning of a frame.
<EOT>	0x04	Marks the end of a frame.
<DLE>	0x10	Data link escape.

Commands

The PC host application can issue the commands listed in Table 2 to the Bootloader. The first byte in the data field carries the command.

Command Value in Hexadecimal	Description
0x01	Read the Bootloader version information.
0x02	Erase the Flash.
0x03	Program the Flash.
0x04	Read the CRC.
0x05	Jump to the application.

Read Bootloader Version Information

The PC host application request for version information to the Bootloader is shown in Example 2.

Example 2: Request
```
[<SOH>…]<SOH>[<0x01>]<CRCL><CRCH><EOT>
```

The Bootloader responds to the PC request for version information in two bytes, as shown in Example 3.

Example 3: Response
```
[<SOH>…]<SOH><0x01><MAJOR_VER><MINOR_VER>
<CRCL><CRCH><EOT>

Where:
MAJOR_VER = Major version of the Bootloader
MINOR_VER = Minor version of the Bootloader
```

Erase Flash

On receiving the erase Flash command from the PC host application, the Bootloader erases that part of the program Flash, which is allocated for the user application. The request frame from the PC host application to the Bootloader is shown in Example 4.

Example 4: Request
```
[<SOH>…]<SOH><0x02><CRCL><CRCH><EOT>
```

The response frame from the Bootloader to the PC host application is shown in Example 5.

Example 5: Response
```
[<SOH>…]<SOH><0x02><CRCL><CRCH><EOT>
```

Program Flash

The PC host application sends one or multiple hex records in Intel Hex format along with the program Flash command. The MPLAB XC32 C/C++ Compiler generates the image in the Intel Hex format. Each line in the Intel hexadecimal file represents a hexadecimal record. Each hexadecimal record starts with a colon (:) and is in ASCII format. The PC host application discards the colon and converts the remaining data from ASCII to hexadecimal, and then sends the data to the Bootloader. The Bootloader extracts the destination address and data from the hex record, and writes the data into program Flash.

The request frame from the PC host application to the Bootloader is shown in Example 6.

Example 6: Request
```
[<SOH>…]<SOH><0x03>[<HEX_RECORD>…]<CRCL>
<CRCH><EOT>
```

Where, `HEX_RECORD` is the Intel Hex record in hexadecimal format.

The response from the Bootloader to the PC host application is shown in Example 7.

Example 7: Response
```
[<SOH>…]<SOH><0x03><CRCL><CRCH><EOT>
```

Read CRC

The read CRC command is used to verify the content of the program Flash after programming. The request frame from the PC host application to the Bootloader is shown in Example 8.

Example 8: Request
```
[<SOH>…]<SOH><0x04><ADRS_LB><ADRS_HB>
<ADRS_UB><ADRS_MB><NUMBYTES_LB><NUMBYTES_HB>
<NUMBYTES_UB><NUMBYTES_MB><CRCL><CRCH><EOT>
```

`ADRS_LB`, `ADRS_HB`, `ADRS_UB` and `ADRS_MB`, as shown in Example 8, represent the 32-bit Flash addresses from where the CRC calculation begins.

`NUMBYTES_LB`, `NUMBYTES_HB`, `NUMBYTES_UB` and `NUMBYTES_MB`, as shown in Example 8, represent the total number of bytes in 32-bit format for which the CRC is to be calculated.

The response from the Bootloader to the PC host application is shown in Example 9.

Example 9: Response
```
[<SOH>…]<SOH><0x04><FLASH_CRCL><FLASH_CRCH>
<CRCL><CRCH><EOT>
```

Jump to Application

The Jump to Application command from the PC host application commands the Bootloader to execute the application. The request frame from the

PC host application to the Bootloader is shown in Example 10.

Example 10: Request
`[<SOH>…]<SOH><0x05><CRCL><CRCH><EOT>`

There is no response to this command from the Bootloader because the Bootloader immediately exits the firmware upgrade mode and begins executing the application.

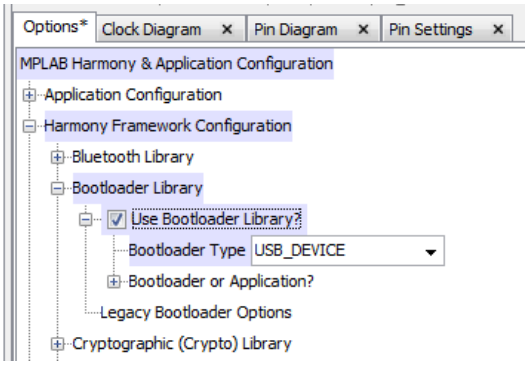
Generating the Application Linker Script

Provides information on generating the application linker script.

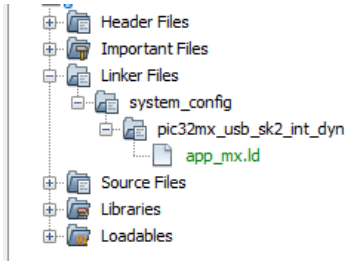
Description

The MPLAB Harmony Configurator (MHC) can generate an application linker script specifically matched to the bootloader type in use. To add the linker script to your application, do the following:

1. In MPLAB X IDE, start MHC, and open the configuration for your application.
2. Navigate to *Harmony Framework Configuration* > *Bootloader Library* and select **Use Bootloader Library?**, as shown in the following figure.



3. Select the bootloader type in use using the Bootloader Type drop-down. For PIC32MX devices, this option must match between Bootloader and Application configurations. The linker scripts and reset address are configured for each based on the size of the bootloader. Therefore, they must be the same for the bootloader to look in the correct location for the application, and jump to the application's reset address.
4. When the application code is generated, a linker script will be created and inserted into the project folder, as shown in the following figure.



Considerations While Moving the Application Image

Describes the procedure to place a user application into a desired program Flash memory region

Description

This section describes the procedure to place a user application into a desired program Flash memory region. It must be ensured that the user application's memory region does not overlap with the memory region reserved for the Bootloader.

The bootloader generated by MHC should be considered a starting point for your product's bootloader. As such, adding new features may cause the bootloader to exceed the size intended in the bootloader linker script. In addition, the bootloaders provided in the `apps/bootloader/basic` folder are configured to the `-O1` compiler optimization. It is not the smallest possible size for the bootloader, and turning on the `-Os` or MIPS16/microMIPS code options will reduce the size of the bootloader. If the size of the bootloader changes, the following steps should be performed to adjust both the bootloader and the application in order to make sure that both fit and make best use of the device memory.

1. Determine the new ending address in Program Flash for the bootloader. This can be done either by using the `.map` file generated by the MPLAB XC32 C/C++ Compiler, or by using the ELFViewer plugin for MPLAB X IDE.
2. Round the address up to the nearest page boundary. For devices with 1K page sizes, the address must end on a 0x400 boundary. For devices with 4K page sizes, the address must end on a 0x1000 boundary.
3. Within the bootloader, change the following files:

- In `btx_mx.ld`, change the length of kseg0_program_mem to the new boundary
- In `system_config.h`, change the value for BOOTLOADER_FLASH_BASE_ADDRESS to match the virtual address determined previously

4. Within the application, change the linker script, `app_mx.ld`, as follows:

- Change _ebase_address to the new boundary, plus enough pages to get the exceptions to start at an address on a 4K (0x1000) boundary.

This is to align it on the boundary required by the MIPS core.

- Change _RESET_ADDR to the new boundary previously determined
- Change the ORIGIN value for kseg0_program_mem to the new boundary

5. Recompile both the bootloader and the application, and test operations.

For PIC32MZ devices, any of the bootloaders can fit entirely into Boot Flash, although some currently take advantage of both panels of Boot Flash. This is done by making the two Boot Flash memories appear as one panel in the linker script, but using a fill command to the linker to block out the area that is not available. When the resulting file is loaded by MPLAB X IDE, you may receive the following warning:

Warning: C:/microchip/harmony/core/apps/bootloader/basic/firmware/basic.X/dist/udp_pic32mz_ef_sk/production/basic.X.production.hex contains code that is located at addresses that do not exist on the PIC32MZxxx.

This warning can be ignored, as it is a warning about the fill memory.

Note: For more information on linker script memory regions, refer to the *"MPLAB® XC32 Assembler, Linker and Utilities User's Guide"* (DS50002186) and the *"MPLAB® XC32 C/C++ Compiler User's Guide"* (DS50001686).

Bootloader Configurations

Provides information on the macros that are available to configure the Bootloader code during compilation.

Description

Most of the configuration options for the bootloaders will be done with the driver or middleware-specific options available in MHC. For example:

- UART - Adjusting the port used and the baud rate
- USB - The value of the Vendor and Product IDs
- Ethernet - The default MAC IDs and IP Address to use
- SD Card - The SPI interface that communicates with the SD card

The following table lists macros that are modified in source code only, and their usage. Depending on user requirements, the values in these macros may need to be changed.

General Macros

Macro	Usage
BOOTLOADER_FLASH_BASE_ADDRESS	Base address of the program Flash reserved for the user application. The address value must point to the beginning of a Flash page, which is dependent upon the Flash page size of the specific device. Note that setting this value to an address below the maximum memory will allow your bootloader to reserve Flash memory as needed.
BOOTLOADER_FLASH_END_ADDRESS	End address of the program Flash reserved for the user application. The address value must point to the end of a Flash page.
BOOTLOADER_RESET_ADDRESS	Address of user Reset vector. The Bootloader branches to this address when it must run the user application.
MAJOR_VERSION	Major version of the Bootloader firmware.
MINOR_VERSION	Minor version of the Bootloader firmware.

Configuring the Library

The configuration of the Bootloader Library is based on the file `system_config.h`.

This header file contains the configuration selection for the Bootloader Library. Based on the selections made, the Bootloader Library will or will not support selected features. These configuration settings will apply to all instances of the Bootloader Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Bootloader Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/bootloader`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/src/bootloader.h	Includes all MPLAB Harmony-compatible function calls for the Bootloader Library.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
<install-dir>framework/bootloader/src/bootloader.c	This file contains the core implementation of the Bootloader Library.
<install-dir>framework/bootloader/src/datastream.c	This file contains the data stream implementation of the Bootloader Library. In addition, a datastream_<type>.c file is included with this file. For example, datastream_usart.c.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
datastream_sdcard.c	Data stream implementation when using the SD Card bootloader configuration.
datastream_udp.c	Data stream implementation when using the Ethernet UDP bootloader configuration.
datastream_usart.c	Data stream implementation when using the USART bootloader configuration.
datastream_usb_hid.c	Data stream implementation when using the USB HID Device bootloader configuration.
datastream_usb_host.c	Data stream implementation when using the USB Host bootloader configuration.

Module Dependencies

The Bootloader Library will depend on the modules used for the communication medium, as listed in the following table.

Bootloader	Module Dependency and MHC Configuration
SD Card	SD Card Driver, configured to use the SPI port and Chip Select lines in use. File System, configured with the following MHC options: • Use File System Auto Mount Feature • Media Type: SYS_FS_MEDIA_TYPE_SD_CARD • FAT File System
USART	USART Static Driver
UDP	TCP/IP, configured with the following MHC options: • IPv4 • Use UDP • One Network Configuration, with interface, host name, and IP address as desired • Enable Stack Deinitialization Operations • Enable Configuration Save/Restore Functionality • Dynamic RAM Size: 20250
USB Device	USB, configured with the following MHC options: • USB Device • Number of Endpoints Used: 2 • Number of Functions: 1 • Device Class: HID • Vendor ID: 0x04d8 • Product ID: 0x003c • Manufacturer String: As desired • Product String: As desired

USB Host	USB, configured with the following MHC options: • USB Host (Recommended) • Use MSD Host Client Driver File System, configured with the following MHC options: • Use File System Auto Mount Feature • Media Type: SYS_FS_MEDIA_TYPE_MSD

Bootloader Sizing and Optimization

Provides bootloader sizing and optimization information.

Description

The example bootloaders provided in the `apps/bootloader/basic` folder have optimization settings for use by most MPLAB XC32 C/C++ Compiler users, which is `-O1`. However, in terms of size, this option does not produce the most optimal code. The following table lists the various bootloaders, and their current size (based on Boot Flash size).

Flash Usage Based on Optimization in Bytes

Peripheral	Device Family	-O1	-Os	-Os MIPS16	-Os microAptiv
SD Card	PIC32MZ(1)	134968	122496	N/A	52110
UART	PIC32MX(1)	7376	6892	5696	N/A
UART	PIC32MZ(1)	12796	12228	N/A	10460
USB Device	PIC32MX	24944	22932	15328	N/A
USB Device	PIC32MZ(1)	38104	34976	N/A	26164
USB Host	PIC32MX	76156	66968	42808	N/A
USB Host	PIC32MZ(1)	85512	75944	N/A	53444
UDP	PIC32MX	73740	72416	48804	N/A
UDP	PIC32MZ(1)	90704	79856	N/A	Not Functional(2)

Notes:
1. The Bootloader resides entirely in Boot Flash. No Program Flash is in use.
2. The UDP bootloader does not compile for PIC32MZ devices when microMIPS is selected.

Application Interrupt Vector Table Placement

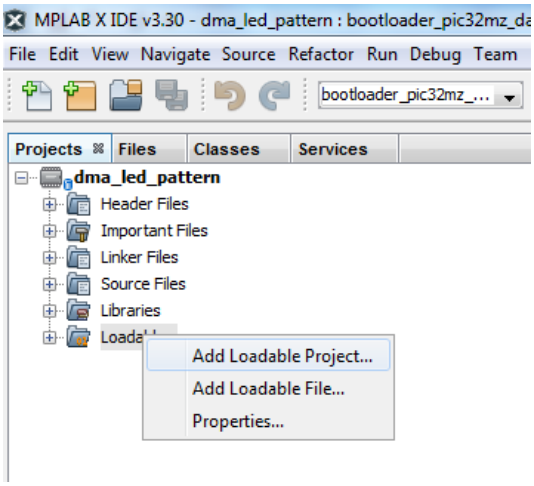
Beginning in MPLAB Harmony v1.08, the application linker scripts are able to have the interrupt vector table within the kseg0_program_mem, along with the rest of the application code. Generally, this means that it can be placed anywhere within the application code, and only the _ebase_address value in the application linker script needs to be updated.

Within the MIPS core, the base address of the interrupt vector table is set by the Ebase register (CP0 Register 15, Select 1). Because of the definition of this register, the interrupt vector table must be aligned on a 4K boundary. This applies to all PIC32 devices. The bootloader will take the start of application Program Memory as the start of the application itself. This means that the interrupt vector table will need to be located at least at the next 4K boundary after that starting address.

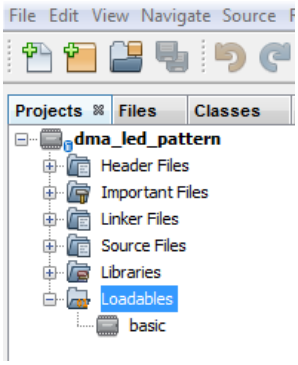
Debugging with a Bootloader and an Application

When developing a system that uses both an application and a bootloader, it is useful to be able to run both as separate projects, but have them both in the device. This involves setting the bootloader as a loadable project to the application. The steps to add the bootloader as the loadable project, are as follows.

1. Right-click the application project's `Loadables` folder, and select **Add Loadable Project…**.



2. Navigate to the project folder of the Bootloader, selecting the correct configuration if there are multiple configurations, and click **Add**.



3. When the program is built, it will generate a single hex file that contains the combined Bootloader and application.
4. When debugging the program, both will be programmed into the PIC32. However, the Bootloader will be executed, and the PIC32 will halt when it reaches the main function of the application. To step through the Bootloader, it is necessary to set a breakpoint in the bootloader code.

Commonly Encountered Situations

When creating the Bootloader and the application, you may encounter some of these situations while combining the two, and errors may be generated by the linker.

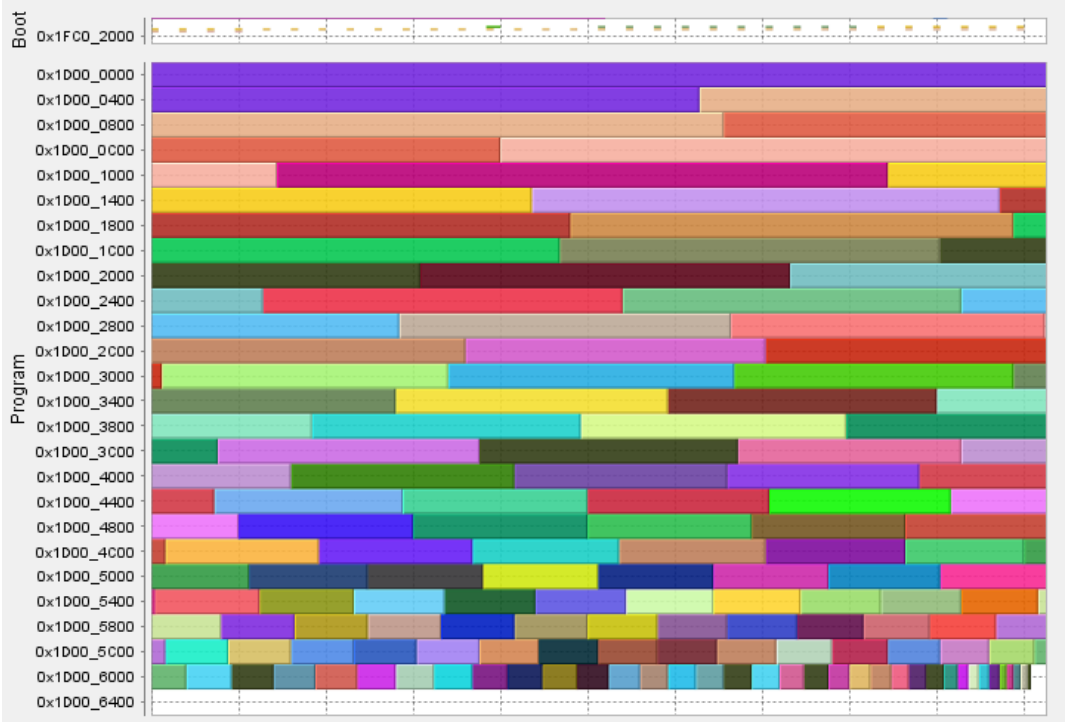
Optimizing the Bootloader

Provides information for optimizing the Bootloader.

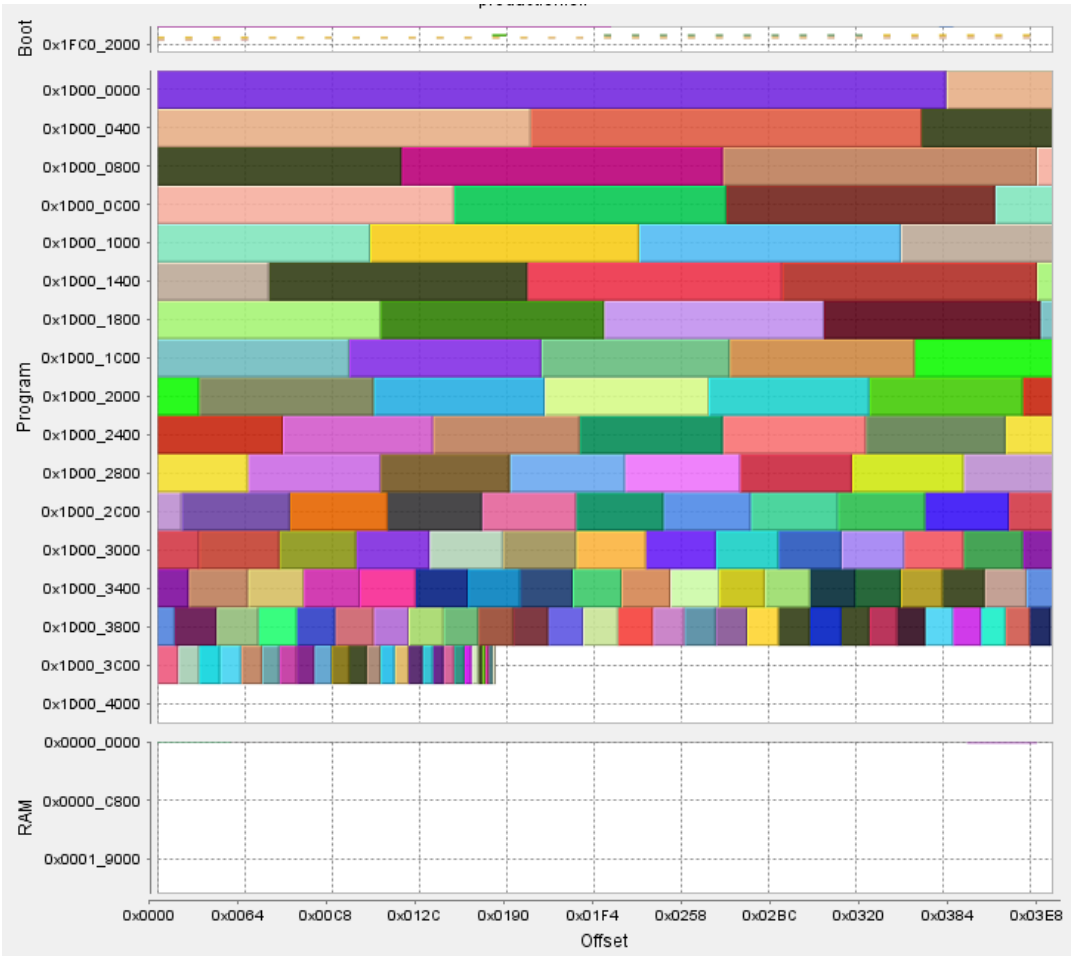
Description

By selecting a different optimization level, or by adding features to the Bootloader, the size of the Bootloader will change. These changes may require adjustments to the linker scripts of the Bootloader and the Application.

The following example will take a PIC32MX USB Device Bootloader, and set optimizations to save space. When the Bootloader is compiled with the default, `-O1`, optimization, it takes 24944 bytes of Flash. This can be seen in the following figure, which graphically shows the mapping of the Bootloader functions and Flash usage.



When the optimizations are set to `-Os`, and MIPS16 code is generated, the Bootloader code size shrinks to 15328 bytes, as shown in the following figure.



Because the Bootloader now ends at a lower address in Program Flash, the space can now be reclaimed for the application. Due to Flash page sizing, the application linker script must take the ending address of the Bootloader, and round it up to the start address of the next page. In this case, the new start address for the application can be 0x9D004000.

The MIPS32 core requires that Interrupt Vector Tables (IVTs) be placed on a 4 KB boundary, Therefore, the IVT for the Application can now be

placed at 0x9D005000, or any address above that in Program Flash that ends in 0x000.

The three lines to change in the application linker script are as follows, with the changes note in **bold** type:

Original Line of Code	Modified Line of Code
`_ebase_address = 0x9D008000;`	`_ebase_address = 0x9D00**5**000;`
`_RESET_ADDR = (0x9D007000);`	`_RESET_ADDR = (0x9D00**4**000);`
`kseg0_program_mem (rx) : ORIGIN = (0x9D000000 + 0x7000), LENGTH = 0x80000 - 0x7000`	`kseg0_program_mem (rx) : ORIGIN = (0x9D000000 + 0x**4**000), LENGTH = 0x80000 - 0x**4**000`

Library Interface

a) Functions

	Name	Description
	Bootloader_Initialize	Initializes the Bootloader Library.
	Bootloader_Tasks	Maintains the Bootloader module state machine. It manages the Bootloader module object list items and responds to Bootloader module primitive events.

b) Data Types and Constants

	Name	Description
	MAJOR_VERSION	Bootloader Major Version Shown From a Read Version on PC
	MINOR_VERSION	Bootloader Minor Version Shown From a Read Version on PC
	BOOTLOADER_CLIENT_STATUS	Enumerated data type that identifies the Bootloader module client status.
	BOOTLOADER_INIT	A structure used to initialize the bootloader defining the different bootloader.
	BOOTLOADER_TYPE	A structure used to initialize the bootloader defining the different bootloader.
	BootInfo	This is variable BootInfo.
	BOOTLOADER_BUFFER	This is type BOOTLOADER_BUFFER.
	BOOTLOADER_CALLBACK	Pointer to a Bootloader callback function data type .
	BOOTLOADER_DATA_CALLBACK	Pointer to a Bootloader callback function data type .

Description

This section describes the Application Programming Interface (API) functions of the Bootloader Library.

Refer to each section for a detailed description.

a) Functions

Bootloader_Initialize Function

Initializes the Bootloader Library.

File

bootloader.h

C

```
void Bootloader_Initialize(const BOOTLOADER_INIT * drvBootloaderInit);
```

Returns

If successful, returns a valid handle to a device layer object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function is used to initialize the Bootloader Library.

Remarks

This routine must be called before other Bootloader Library functions.

Preconditions

None.

Function

void Bootloader_Initialize (const SYS_MODULE_INDEX moduleIndex,
const SYS_MODULE_INIT * const moduleInit);

Bootloader_Tasks Function

Maintains the Bootloader module state machine. It manages the Bootloader module object list items and responds to Bootloader module primitive events.

File

bootloader.h

C

```
void Bootloader_Tasks();
```

Returns

None.

Description

This function maintains the Bootloader module state machine and manages the Bootloader Module object list items and responds to Bootloader Module events. This function should be called from the SYS_Tasks function.

Remarks

This function is normally not called directly by an application.

Preconditions

None.

Example

```
while (true)
{
    Bootloader_Tasks ();

    // Do other tasks
}
```

Parameters

Parameters	Description
index	Object index for the specified module instance.

Function

void Bootloader_Tasks (SYS_MODULE_INDEX index);

b) Data Types and Constants

MAJOR_VERSION Macro

File

bootloader.h

C

```
#define MAJOR_VERSION 4    /* Bootloader Major Version Shown From a Read Version on PC */
```

Description

Bootloader Major Version Shown From a Read Version on PC

MINOR_VERSION Macro

File

bootloader.h

C

```
#define MINOR_VERSION 1    /* Bootloader Minor Version Shown From a Read Version on PC */
```

Description

Bootloader Minor Version Shown From a Read Version on PC

BOOTLOADER_CLIENT_STATUS Enumeration

Enumerated data type that identifies the Bootloader module client status.

File

bootloader.h

C

```
typedef enum {
  BOOTLOADER_CLIENT_STATUS_CLOSED,
  BOOTLOADER_CLIENT_STATUS_READY,
  BOOTLOADER_CLIENT_STATUS_WRITEFAILURE,
  BOOTLOADER_CLIENT_STATUS_WRITESUCCESS
} BOOTLOADER_CLIENT_STATUS;
```

Members

Members	Description
BOOTLOADER_CLIENT_STATUS_CLOSED	Client is closed or the specified handle is invalid
BOOTLOADER_CLIENT_STATUS_READY	Client is ready

Description

Bootloader Client Status

This enumeration defines the possible status of the Bootloader module client. It is returned by the () function.

Remarks

None.

BOOTLOADER_INIT Structure

A structure used to initialize the bootloader defining the different bootloader.

File

bootloader.h

C

```
typedef struct {
  BOOTLOADER_TYPE drvType;
  BOOTLOADER_STATES (* drvTrigger)(void);
} BOOTLOADER_INIT;
```

Description

Bootloader Initialization Type

This structure holds the bootloader types.

Remarks

None.

BOOTLOADER_TYPE Enumeration

A structure used to initialize the bootloader defining the different bootloader.

File

bootloader.h

C

```
typedef enum {
  TYPE_I2C,
  TYPE_USART,
  TYPE_USB_HOST,
  TYPE_USB_DEVICE,
  TYPE_ETHERNET_UDP_PULL,
  TYPE_SD_CARD
} BOOTLOADER_TYPE;
```

Description

Bootloader Type

This structure holds the bootloader types.

Remarks

None.

BootInfo Variable

File

bootloader.h

C

```
const uint8_t BootInfo[2] = { MINOR_VERSION, MAJOR_VERSION };
```

Description

This is variable BootInfo.

BOOTLOADER_BUFFER Union

File

bootloader.h

C

```
typedef union {
  uint8_t buffer[1024];
  struct {
    uint8_t buff1[512];
    uint8_t buff2[512];
  } buffers;
} BOOTLOADER_BUFFER;
```

Description

This is type BOOTLOADER_BUFFER.

BOOTLOADER_CALLBACK Type

Pointer to a Bootloader callback function data type .

File

bootloader.h

C

```
typedef int (* BOOTLOADER_CALLBACK)(void);
```

Description

Bootloader General Callback Function Pointer

This data type defines a pointer to a Bootloader library callback function.

Remarks

This is used for callback on the following events:

- ERASE - Erase any additional areas (i.e. SQI, EBI, SPI memories)
- PROGRAM_COMPLETE - No more data coming
- START_APP - Anything that needs to be done prior to starting application
- BLANK_CHECK - Check external devices for erasure
- FORCE_BOOTLOAD - Any checks the bootloader wants to do before launching application

BOOTLOADER_DATA_CALLBACK Type

Pointer to a Bootloader callback function data type .

File

bootloader.h

C

```
typedef int (* BOOTLOADER_DATA_CALLBACK)(uint32_t address, uint32_t *data, uint32_t size);
```

Description

Bootloader Data to Program Callback Function Pointer

This data type defines a pointer to a Bootloader data callback function. This would be used to program data for an area not recognized by the bootloader as being in progam Flash (i.e. SPI Flash).

Remarks

This is used for callback on the following events:

- DATA_PROGRAM - Data to be programmed
- DATA_VERIFY - Compute the CRC for the given area

Files

Files

Name	Description
bootloader.h	The header file joins all header files used in the Bootloader Library and contains compile options and defaults.

Description

This section lists the source and header files used by the Bootloader Library.

bootloader.h

The header file joins all header files used in the Bootloader Library and contains compile options and defaults.

Enumerations

	Name	Description
	BOOTLOADER_CLIENT_STATUS	Enumerated data type that identifies the Bootloader module client status.
	BOOTLOADER_TYPE	A structure used to initialize the bootloader defining the different bootloader.

Functions

	Name	Description
	Bootloader_Initialize	Initializes the Bootloader Library.
	Bootloader_Tasks	Maintains the Bootloader module state machine. It manages the Bootloader module object list items and responds to Bootloader module primitive events.

Macros

	Name	Description
	MAJOR_VERSION	Bootloader Major Version Shown From a Read Version on PC
	MINOR_VERSION	Bootloader Minor Version Shown From a Read Version on PC

Structures

	Name	Description
	BOOTLOADER_INIT	A structure used to initialize the bootloader defining the different bootloader.

Types

	Name	Description
	BOOTLOADER_CALLBACK	Pointer to a Bootloader callback function data type .
	BOOTLOADER_DATA_CALLBACK	Pointer to a Bootloader callback function data type .

Unions

	Name	Description
	BOOTLOADER_BUFFER	This is type BOOTLOADER_BUFFER.

Variables

	Name	Description
	BootInfo	This is variable BootInfo.

Description

Module for Microchip Bootloader Library

This header file includes all the header files required to use the Microchip Bootloader Library. Library features and options defined in the Bootloader Library configurations will be included in each build.

File Name

bootloader.h

Company

Microchip Technology Inc.

Class B Library Help

This section describes the Class B Library that is available in MPLAB Harmony.

Introduction

This library provides Class B Safety Software routines which can detect the occurrence of Faults in a single channel CPU.

Description

These routines have been developed in accordance with the IEC 60730 standard to support the Class B certification process. These routines can be directly integrated with the end user's application to test and verify the critical functionalities of a controller without affecting the end user's application. The Class B safety software routines can be called periodically at start-up or run time to test the following components:

- CPU Registers
- CPU Program Counter
- Invariable Memory
- Variable Memory
- Clock

NOTE: The term 'IEC 60730 standard' used in this document refers to the "IEC 60730-1 ed.5" Copyright © 2013 IEC, Geneva, Switzerland. www.iec.ch

Overview of the IEC 60730 Standard

Note: Microchip thanks the International Electrotechnical Commission (IEC) for permission to reproduce information from its International Standard IEC 60730-1ed.5 (2013). All such extracts are copyright of IEC, Geneva, Switzerland. All rights reserved. Further information on the IEC is available from www.iec.ch. IEC has no responsibility for the placement and con-text in which the extracts and contents are reproduced by the author, nor is IEC in any way responsible for the other content or accuracy therein.

The IEC 60730 standard defines the test and diagnostic methods that ensure the safe operation of the controlled equipment used in household appliances. Annex H of the IEC 60730 standard classifies the software into the following categories (see Appendix B: "IEC 60730-1 Table H.11.12.7"):

- Class A
- Class B
- Class C

The Class B Safety Software Library implements the important test and diagnostic methods that fall into the Class B category. These methods use various measures to detect and respond to the software-related Faults and errors.

According to the IEC 60730 standard, the controls with functions that fall into the Class B category should have one of the following structures:

- Single Channel with Functional Test - In this structure, the Functional test is executed prior to the application firmware execution
- Single Channel with Periodic Self-Test - In this structure, the Periodic tests are embedded within the firmware, and the self-test occurs periodically while the firmware is in Execution mode
- Dual Channel without Comparison - In this structure, two independent methods execute the specified operations

System Requirements

The following system requirements are recommended to run the Class B Safety Software Library:

- For the tests that require the independent time slot monitoring, the system hardware must be provided with at least two independent clock sources (e.g., crystal oscillator and line frequency).
- The user application determines whether the interrupts need to be enabled or disabled during the execution of the Class B Safety Software Library.

If an interrupt occurs during the execution of a Class B Safety Software Library routine, an unexpected change may occur in any of the registers. Therefore, when the Interrupt Service Routine (ISR) executes, the contents of the register will not match the expected content, and the ISR could return an incorrect result.

Using the Library

This topic describes the basic architecture of the Class B Library and provides information and examples on its use.

Description

Interface Header File: `classb.h`

The interface to the Class B Library is defined in the `classb.h` header file. Any C language source (`.c`) file that uses the Class B Library should include `classb.h`.

Library File:

The Class B Library archive (.a) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the Class B Library interacts with the framework.

Abstraction Model

This library provides the low-level abstraction of the Class B Library module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Class B Software Abstraction Block Diagram

Library Overview

The Library Interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Class B Library module. The Class B Safety Software Library includes several APIs, which are intended to maximize application reliability through Fault detection. These APIs help meet the IEC 60730 standard compliance.

Library Interface Section	Description
General Functions	Provides an error string function, which takes an error and converts it into human-readable text.

How the Library Works

This topic provides information on how the library works.

Description

CPU Register Test

The CPU Register test implements functional test H.2.16.5, as defined by the IEC 60730 standard. It detects stuck-at Faults in the CPU registers. This ensures that the bits in the registers are not stuck at a value of ‘0’ or ‘1’.

The CPU Register test is a non-destructive test. It performs the following major tasks:

- The CPU registers and ghost registers are tested by first successively writing the binary sequences (length is dependent upon architecture), 010101... followed by 101010... into the registers, and then reading the values from these registers for verification.
- The test returns an error code if the returned values do not match.

Program Counter Test

The Program Counter (PC) test implements the functional test H.2.16.5 defined by the IEC 60730 standard. It detects stuck-at Faults in the PC. The PC holds the address of the next instruction to be executed. The test performs the following major tasks:

- Sets the error flag to the number of functions to be called.
- Calls the functions that are located in the Flash memory at different addresses chosen to alternate as many bits as possible in the PC.
- Each of these functions modifies the error flag.
- The error flag is tested after all functions are called.
- If the error flag is cleared, the PC has branched to all the correct locations.

Note: The size of the program memory varies by device. Thus the addresses used to test the PC will vary in order to test as many PC bits as possible. Refer to the specific device data sheet for more details.

Variable Memory Test

The Variable Memory test implements the Periodic Static Memory test H.2.19.6 defined by the IEC 60730 standard. It detects single bit Faults in variable memory. The variable memory contains data, which is intended to vary during program execution. The RAM Memory test is used to determine if any bit of the RAM memory is stuck at ‘1’ or ‘0’. The March Memory test and Checkerboard test are some of the widely used static memory algorithms for checking the DC Faults. The following tests can be implemented using the Class B Safety Software Library:

- Checkerboard
- March C/C- Test
- March B Test
- March C Stack test

March Tests

A March test performs a finite set of operations on every memory cell in a memory array. Each operation performs the following tasks:

- Writes ‘0’ to a memory cell (w0).
- Writes ‘1’ to a memory cell (w1).
- Reads the expected value ‘0’ from a memory cell (r0).
- Reads the expected value ‘1’ from a memory cell (r1).

March Test Notations

> Arrange Addresses in ascending order

< Arrange Addresses in descending order

<> Arrange Addresses in either order

r0 Indicates a read operation (Read ‘0’ from a memory cell)

r1 Indicates a read operation (Read ‘0’ from a memory cell)

w0 Indicates a write operation (Write ‘0’ to a memory cell)

w1 Indicates a write operation (Write ‘1’ to a memory cell)

MARCH C/C- TEST

The March C/C- test is used to detect the following types of Fault in the variable memory:

• Stuck-at Fault

• Addressing Fault

• Transition Fault

• Coupling Fault

The complexity of the March C/C- test is 11n and 10n respectively, where n indicates the number of bits in the memory. This test can be run as either destructive or non-destructive. If run in non- destructive mode buffer space is required to store the contents of the memory to be tested and restored. If needed, this test can be executed at the system start-up before initializing the memory.

March C Algorithm

```
{
 <> (w0); > (r0, w1); > (r1, w0);
 <> (r0); < (r0, w1); < (r1, w0); > (r0)
}
```

Note1: is be removed for C-

MARCH B Test

The March B is a non -redundant test that can detect the following types of Fault:

- Stuck-at
- Linked Idempotent Coupling
- Inversion Coupling

This test is of complexity 17n, where n indicates the number of bits in the memory. This test can be run as either destructive or non-destructive. If run in non-destructive mode buffer space is required to store the contents of the memory to be tested and restored. If needed, this test can be executed at the system start-up before initializing the memory.

MARCH B ALGORITHM

```
{
<> (w0); > (r0, w1, r1, w0, r0, w1); > (r1, w0, w1);
 < (r1, w0, w1, w0); < (r0, w1, w0);
}
```

Checkerboard RAM Test

The Checkerboard RAM test writes the checkerboard patterns to a sequence of adjacent memory locations. This test is performed in units (memory chunks) of 8 bytes. This test can be run as either destructive or non-destructive. If run in non-destructive mode buffer space is required to store the contents of the memory to be tested and restored. If needed, this test can be executed at the system start-up before initializing the memory. This test performs the following major tasks:

- Saves the contents of the memory locations to be tested in the buffer defined by bufferAddress.
- Writes the binary value (length is dependent upon architecture) 101010... to the memory location, ‘N’, and the inverted binary value, 010101..., to the memory location, ‘N+1’, and so on, until the whole memory chunk is filled.
- Reads the contents of all the memory locations in the current chunk and verifies its contents. If the values match, the function continues; otherwise it stops and returns an error.
- Step 2 and 3 are repeated by writing the inverted pattern to the same locations.
- Once a memory chunk is completed the test of the next chunk is started until all of the requested memory area is tested.

Invariable Memory (Flash/EEPROM) Test

The Invariable Memory (Flash/EEPROM) test implements the periodic modified checksum H.2.19.3.1 defined by the IEC 60730 standard. It detects the single bit Faults in the invariable memory. The invariable memory in a system, such as Flash and EEPROM memory, contains data that is not intended to change during the program execution. The Flash/EEPROM Invariable Memory test computes the periodic check-sum using the Cyclic Redundancy Check (CRC-16). The CRC polynomial used to calculate the CRC-16 is shown below.

CRC-16 = 1 1000 0000 0000 0101 = 8005 (hex)

The CRC functions can be used to test the integrity of data stored in either Flash or EEPROM memory. This is done by calculating and returning

the CRC value of the data stored in the location defined in the function call.

- If CRC_Flag is set to 0x00 at system start-up, the reference CRC checksum is computed.
- The reference checksum is stored in the Flash or EEPROM memory and the CRC flag is set to 0xFF.
- The CRC16 calculation function can be called periodically if the CRC flag is set to 0xFF.
- The checksum calculated from step 3 is compared with the reference checksum.
- If both values match, a status bit can be set by the user application to indicate that the invariable memory has passed the test and no errors were found.

FLOWCHART FOR THE INVARIABLE MEMORY TEST

Note: Other than the calls to CLASSB_CRCFlashTest and CLASSB_CRCEEPROMTest, all steps in this flowchart are to be executed by the application firmware.

Clock Test

According to the IEC 60730 standard, only harmonics and sub-harmonics of the clock need to be tested if using a quartz synchronized clock. The Clock test implements frequency monitoring H.2.18.10.1 as defined by the IEC 60730 standard. It verifies the reliability of the system clock (i.e., the system clock should be neither too fast nor too slow). The Clock Test function is used to verify the proper operation of the CPU clock. This test performs the following major tasks:

- The independent clock source (Sosc or T1OSC) is required for the test. The reference clock should be connected to a timer such as Timer1.
- During the test the number of CPU clock cycles each reference clock period is counted.
- If the number of clock cycles is outside a specified range, the function returns an error code.

Clock Test Using Line Frequency

The Clock Test function is used to verify the proper operation of the CPU clock. The 50 Hz/60 Hz line frequency is used as an independent clock source or a reference clock source. The input capture module is used for the period measurement. The 50 Hz/60 Hz line frequency is fed to the Input Capture pin (IC1) of the respective device. This test performs the following major tasks:

- The IC1 input is used as the independent clock source/reference clock source to capture the hardware Timer2. An external reference frequency, usually the line frequency, has to be applied to the IC1 input pin.
- The Input Capture 1 is configured as follows:
- Timer2 is selected as IC1 time base
- Capture is performed on every rising edge
- Capture done event is generated on every second capture.
- The hardware Timer2 prescaler is calculated (based on the input reference frequency and the current PB frequency) as being the smallest divider possible such that the 16 bit Timer2 does not overflow within a period time of the input reference signal. This way, the best resolution is achieved for the given conditions. If no valid pre-scaler value can be obtained an error value is returned.
- The IC1 performs the capture on every rising edge of the input reference frequency. For period measurement, the capture done event is generated after the IC1 module takes two time stamps i.e. after every period of the input reference (20 ms if reference frequency is 50 Hz, 16.66ms if the reference frequency is 60 Hz).
- Once the capture done event is signaled, the two Timer2 captured readings are extracted and the number of elapsed PB clocks is calculated as being the difference between the two readings. If this value crosses the defined boundary limits the function returns an appropriate error value

Calculation Example 1:

System Clock = 80 MHz

PB Clock = 80 MHz (PB divider =1:1)

Input Reference = 50 Hz

T2 Min Divider = floor(PBclk/(65536*RefClk))+1 = 25

Actual T2 Divider = 32.

The number of cycles counted in the Reference clock period is = (80,000,000/32)/50 = 50,000.

Calculation Example 2:

System Clock = 80 MHz

PB Clock = 10 MHz (PB divider =1:8)

Input Reference = 60 Hz

T2 Min Divider = floor(PBclk/(65536*RefClk))+1 = 3

Actual T2 Divider = 4.

The number of cycles counted in the Reference clock period is = (10,000,000/4)/60 = 41,666.

Configuring the Library

The configuration of the Class B Library is based on the file `system_config.h`.

This header file contains the configuration selection for the Class B Library. Based on the selections made, the Class B Library may support the selected features. These configuration settings will apply to all instances of the Class B Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Class B Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/classb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`classb.h`	Includes all MPLAB Harmony-compatible function calls for the Class B Library.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`<file name>`	<add description>

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
`<file name>`	<add description>

Module Dependencies

The Class B Library does not depend on any other modules.

Library Interface

a) Functions

	Name	Description
	CLASSB_ClockLineFreqTest	The CPU Clock Line Test is one of the tests that check the reliability of the system clock. It implements the independent time slot monitoring H.2.18.10.4 as defined by the IEC 60730 standard.
	CLASSB_ClockTest	The CPU Clock test is one of the tests that check the reliability of the system clock. It implements the independent time slot monitoring H.2.18.10.1 as defined by the IEC 60730 standard.
	CLASSB_CPUPCTest	The Program Counter test implements one of the functional tests H.2.16.5 as defined by the IEC 60730 standard.
	CLASSB_CPURegistersTest	The CPU Register test implements the functional test H.2.16.5 as defined by the IEC 60730 standard.
	CLASSB_CRCFlashTest	The Flash CRC16 test implements the periodic modified checksum H.2.19.3.1 as defined by the IEC 60730 standard.
	CLASSB_RAMCheckerBoardTest	The RAM Checker Board test implements one of the functional tests H.2.19.6 as defined by the IEC 60730 standard.
	CLASSB_RAMMarchBTest	The RAM March B test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.
	CLASSB_RAMMarchCStackTest	The RAM March C test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.
	CLASSB_RAMMarchCTest	The RAM March C test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.

	Flash_CRC16	• Function: unsigned int Flash_CRC16(unsigned char* pBuff, unsigned int crcPoly, unsigned int crcReg, unsigned int bSize) * • PreCondition: pBuff valid pointer * • Input: - pBuff: buffer to calculate CRC over • - crcPoly: the generator polynomial to be used • - crcReg: initial value of the CRC LFSR (seed) • - bSize: buffer size, bytes * • Output: value of the CRC * • Side Effects: None * • Overview: Shifts bytes through the CRC shift register. * • Note: Simple (and slow) CRC16 calculation directly based • on the hardware LFSR implementation.

b) Data Types and Constants

	Name	Description
	CLASSBRESULT	This is type CLASSBRESULT.
	CLASSB_RAM_TEST_CYCLE_SIZE	This is macro CLASSB_RAM_TEST_CYCLE_SIZE.
	CLOCK_TEST_INTERRUPT_SOURCE	This is macro CLOCK_TEST_INTERRUPT_SOURCE.
	CLOCK_TEST_REFERENCE_FREQ	Defines the reference frequency that is applied to the SOSC input.
	CLOCK_TEST_TIMER	This is macro CLOCK_TEST_TIMER.
	CRC_08_GEN_POLY	Flash CRC16 Test Generator Polynomials The value of the generator polynomial is used as an input parameter for the CLASSB_CRCFlashTest() function. It specifies what polynomial to be used for the CRC calculation. Following is a list of some of the most commonly used 16 bit Generator Polynomials that can be used. Any other polynomial that has the required fault detection capabilities can be used.
	CRC_16_GEN_POLY	This is macro CRC_16_GEN_POLY.
	CRC_32_GEN_POLY	This is macro CRC_32_GEN_POLY.
	CRC_CCITT_GEN_POLY	This is macro CRC_CCITT_GEN_POLY.
	FLASH_CRC16_MASK	mask to retain the useful CRC result for 16 bit polynomial
	FLASH_CRC16_MSB	mask to obtain the MSb-1, transport to the MSb.
	FLASH_CRC32_MASK	mask to retain the useful CRC result for 32 bit polynomial
	FLASH_CRC32_MSB	mask to obtain the MSb-1, transport to the MSb.
	FLASH_CRC8_MASK	mask to retain the useful CRC result for 8 bit polynomial
	FLASH_CRC8_MSB	mask to obtain the MSb-1, transport to the MSb.
	USE_MARCHC_MINUS	This is macro USE_MARCHC_MINUS.

Description

This section describes the Application Programming Interface (API) functions of the Class B Library.

Refer to each section for a detailed description.

a) Functions

CLASSB_ClockLineFreqTest Function

The CPU Clock Line Test is one of the tests that check the reliability of the system clock. It implements the independent time slot monitoring H.2.18.10.4 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_ClockLineFreqTest(unsigned int clockFrequency, unsigned int lineFrequency, unsigned int tolerance);
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. The monitored CPU clock is within the requested limits.

CLASSB_TEST_FAIL - The test failed. The monitored CPU clock is greater than the specified upper limit; The monitored CPU clock is less than the specified lower limit; The frequency of the provided reference was too low and could not be used.

Description

The CPU Clock Line Test verifies that the system clock is within specified limits. An external frequency applied on Input Capture 1 pin (IC1) is used as the reference clock. The hardware Timer2 that runs on the system Peripheral Bus (PB) clock is used to monitor the CPU clock and Peripheral Bus divider.

The test performs the following major steps:

1. The IC1 input is used as the independent clock source/reference clock source to capture the hardware Timer2. An external reference frequency, usually the line frequency, has to be applied to the IC1 input pin.
2. The Input Capture 1 is configured as follows:

- Timer2 is selected as IC1 time base
- Capture is performed on every rising edge
- Capture done event is generated on every second capture.

3. The hardware Timer2 pre-scaler is calculated (based on the input reference frequency and the current PB frequency) as being the smallest divider possible such that the 16 bit Timer2 does not overflow within a period time of the input reference signal. This way, the best resolution is achieved for the given conditions. If no valid pre-scaler value can be obtained an error value is returned.
4. The IC1 performs the capture on every rising edge of the input reference frequency. For period measurement, the capture done event is generated after the IC1 module takes two time stamps i.e. after every period of the input reference (20 ms if reference frequency is 50 Hz, 16.66ms if the reference frequency is 60 Hz).
5. Once the capture done event is signalled, the 2 Timer2 captured readings are extracted and the number of elapsed PB clocks is calculated as being the difference between the two readings. If this value crosses the defined boundary limits the function returns an appropriate error value, specifying which exactly limit (upper/lower) was violated.

Calculation example 1: System Clock = 80 MHz PB Clock = 80 MHz (PB divider =1:1) Input Reference = 50 Hz

T2 Min Divider = floor(PBclk/(65536*RefClk))+1 = 25 Actual T2 Divider = 32. The number of cycles counted in the Reference clock period is = (80,000,000/32)/50 = 50,000.

Calculation example 2: System Clock = 80 MHz PB Clock = 10 MHz (PB divider =1:8) Input Reference = 60 Hz

T2 Min Divider = floor(PBclk/(65536*RefClk))+1 = 3 Actual T2 Divider = 4. The number of cycles counted in the Reference clock period is = (10,000,000/4)/60 = 41,666.

Remarks

The test uses the hardware Input Capture 1 module. It initializes the module as needed and, after the test is done, it shuts it off. The previous state of IC1 is not preserved/restored.

The test uses the hardware Timer2. It initializes the timer as needed and, after the test is done, shuts off the timer. The previous state of Timer1 is not preserved/restored.

The value of the PB frequency which is used as input by the Timer2 is derived from the system CPU clock by dividing it with the PB divider. The test does not change the value of the PB divider, it uses the current value.

The interrupts should be disabled when executing this test as the time spent in ISR's affects the accurate timing of this routine.

The frequency of signal used as a reference on IC1 input should normally be the line frequency. However, any frequency can be used as long as a a valid Timer2 divider can be obtained (see the example calculation below for the 16 bit Timer2). If the reference frequency is too low, a valid divider for the Timer 2 won't be possible (the maximum divider for Timer2 is 256). You can go to a greater PB divider in this case. If the selected reference frequency is too high the number of Timer2 captured counts will be too small and the measurement won't have enough resolution.

Preconditions

None.

Example

```
int testRes=CLASSB_ClockLineFreqTest(80000000, 60, 80000, 100000);
if(testRes==CLASSB_TEST_PASS)
{
    // process test success
}
else
{
    // process CPU clock high failure
}
```

Parameters

Parameters	Description
clockFrequency	the current system running frequency, Hz
lineFrequency	the frequency of the reference applied to the IC1 input pin, Hz. Usual values are 50/60 Hz
tolerance	the tolerance level of the system oscillator as a percentage (0 - 100).

Function

CLASSBRESULT CLASSB_ClockLineFreqTest(unsigned int clockFrequency, unsigned int lineFrequency, uint8_t tolerance)

CLASSB_ClockTest Function

The CPU Clock test is one of the tests that check the reliability of the system clock. It implements the independent time slot monitoring H.2.18.10.1 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_ClockTest(unsigned int clockFrequency, unsigned int referenceFrequency, unsigned int testLengthMsec, unsigned int tolerance);
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. The monitored CPU clock is within the requested limits.

CLASSB_TEST_FAIL - The test failed. The monitored CPU clock is greater than the specified upper limit; The monitored CPU clock is less than the specified lower limit.

Description

The CPU Clock test verifies that the system clock is within specified limits. The secondary oscillator (SOSC) is used as the reference clock. The CPU Core Timer that runs on the CPU system clock is monitored.

The test performs the following major steps:

1. The LP SOSC is used as the independent clock source/reference clock source connected to hardware Timer1.
2. The CPU Core Timer monitored in this measurement is incremented every other CPU system clock. Usually the system runs on the Primary oscillator with PLL as the clock source to the CPU. However, any clock source except the SOSC itself which is used as a reference is valid for this test.
3. Timer1 is configured to time out after the specified interval of time elapsed (e.g. 10 ms).
4. The content of the Core Timer is saved at the beginning of the measurement, once the Timer 1 is started.
5. When the hardware Timer1 times out another reading of the Core Timer is taken and the difference is made with the start value. This difference value represents the number of CPU clock cycles counted by the Core Timer during the SOSC period of time.
6. If this value crosses the defined boundary limits the function returns an appropriate error value, specifying which exactly limit (upper/lower) was violated.

Remarks

The test uses the hardware Timer1. It initializes the timer as needed and, after the test is done, shuts off the timer. The previous state of Timer1 is not preserved/restored.

The test assumes that the Core Timer is enabled and linearly counting up as it should do during normal system operation. If your code specifically disables the Core Timer, it should enable it before this test is called. If the value in the Core Timer is updated/changed as part of an ISR, this ISR should be disabled.

The interrupts should be disabled when executing this test as the time spent in ISR's affects the accurate timing of this routine.

The SOSC is used as a reference. Use the CLOCK_TEST_SOSC_FREQ for adjustments to this value, if needed.

The value of the CPU clock monitoring time, nMs, is limited because of the use of the hardware Timer1 which is a 16 bit timer. Therefore, the value loaded into this Timer1 register should not exceed 2^16-1.

Preconditions

None.

Example

```
int testRes=CLASSB_ClockTest(80000000, 100, 80000, 100000);
if(testRes==CLASSB_TEST_PASS)
{
    // process test success
}
```

```
else
{
    // process CPU clock high failure
}
```

Parameters

Parameters	Description
clockFrequency	the current system running frequency, Hz
referenceFrequency	the frequency of the reference applied to the SOSC pin, Hz.
testLengthMsec	number of milliseconds to be used for the CPU Clock monitoring. 1 <= testLengthMsec <= 1000
tolerance	the tolerance level of the system oscillator as a percentage (0 - 100).

Function

CLASSBRESULT CLASSB_ClockTest(unsigned int sysClk, int nMs, unsigned int hiClkErr, unsigned int loClkErr)

CLASSB_CPUPCTest Function

The Program Counter test implements one of the functional tests H.2.16.5 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_CPUPCTest();
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. The PC register holds the correct address.

CLASSB_TEST_FAIL - The test failed. The PC register has been detected to hold an incorrect address.

Description

The Program Counter test is a functional test of the PC. It checks that the PC register is not stuck and it properly holds the address of the next instruction to be executed.

The tests performs the following major tasks:

1. The Program Counter test invokes functions that are located in memory at different addresses.
2. That all of the functions is executed is verified.

Remarks

The test uses 3 different functions: CLASSB_CPUPCTestFunction1() CLASSB_CPUPCTestFunction2() CLASSB_CPUPCTestFunction3()

Preconditions

None.

Example

```
int testRes=CLASSB_CPUPCTest();
if(testRes==PC_TEST_PASS)
{
    // process test success
}
else
{
    // process tests failure
}
```

Function

int CLASSB_CPUPCTest (void)

CLASSB_CPURegistersTest Function

The CPU Register test implements the functional test H.2.16.5 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_CPURegistersTest();
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. CPU registers have not been detected to have stuck bits.

CLASSB_TEST_FAIL - The test failed. Some CPU register(s) has been detected to have stuck bits.

Description

This routine detects stuck-at Faults in the CPU registers. This ensures that the bits in the registers are not stuck at a value ?0? or ?1?.

Remarks

This is a non-destructive test.

Interrupts should be disabled when calling this test function.

Preconditions

None.

Example

```
int testRes=CLASSB_CPURegistersTest();
if(testRes==CLASSB_TEST_PASS)
{
    // process test success
}
else
{
    // process tests failure
}
```

Function

int CLASSB_CPURegistersTest (void)

CLASSB_CRCFlashTest Function

The Flash CRC16 test implements the periodic modified checksum H.2.19.3.1 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
unsigned int CLASSB_CRCFlashTest(char* startAddress, char* endAddress, unsigned int crcPoly, unsigned int crcSeed);
```

Returns

The value of the calculated CRC over the specified memory area.

Description

This routine detects the single bit Faults in the invariable memory. The invariable memory in a system, such as Flash and EEPROM memory, contains data that is not intended to vary during the program execution.

The test calculates the 16 bit CRC of the supplied memory area using the standard LFSR (Linear Feedback Shift Register) implementation. It calculates over the memory area between the startAddress and endAddress and returns the CRC Value. The 16 bit CRC is calculated using the supplied generator polynomial and initial seed. Different generator polynomials can be used as indicated above.

Remarks

This is a non-destructive memory test.

The startAddress and endAdress over which the CRC value is calculated are PIC32 variant and application dependent. They are run-time parameters.

Preconditions

None.

Example

```
unsigned int crcRes=CLASSB_CRCFlashTest(startAddress, endAddress, CRC_16_GEN_POLY, 0xffff);
if(crcRes==prevCalculatedCrc)
{
    // process test success
}
else
{
    // process tests failure: the CRC of the memory area has changed.
}
```

Parameters

Parameters	Description
startAddress	start Address of the memory area to start CRC calculation from
endAddress	final address for which the CRC is calculated
crcPoly	the generator polynomial to be used. One of the standard supplied polynomials can be used as well as other user defined ones.
crcSeed	the initial value in the CRC LFSR. The usual recommended value is 0xffff.

Function

unsigned int CLASSB_CRCFlashTest(char* startAddress, char* endAddress, unsigned int crcPoly, unsigned int crcSeed)

CLASSB_RAMCheckerBoardTest Function

The RAM Checker Board test implements one of the functional tests H.2.19.6 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_RAMCheckerBoardTest(unsigned int * startAddress, unsigned int length);
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. RAM area tested has not been detected to have stuck bits.

CLASSB_TEST_FAIL - The test failed. Some RAM area location has been detected to have stuck bits.

Description

This routine detects single bit Faults in the variable memory. This ensures that the bits in the tested RAM are not stuck at a value ?0? or ?1?.

The test writes the checkerboard pattern (0x55555555 followed by 0xaaaaaaaa) to adjacent memory locations starting at ramStartAddress. It performs the following steps:

1. A temporary memory buffer (stack) is tested for reliability.
1. The content of a 64 bytes memory chunk to be tested is saved in temporary memory buffer.
2. Writes the pattern 0x55555555 followed by 0xaaaaaaaa to adjacent memory locations filling up the 64 bytes memory chunk.
3. It reads the memory chunk adjacent locations and checks that the read-back values match the written pattern. If the values match set the success result and go to step 4. Else set the error result and go to step 6.
4. Writes the inverted pattern 0xaaaaaaaa followed by 0x55555555 to adjacent memory locations filling up the 64 bytes memory chunk.
5. It reads the memory chunk adjacent locations and checks that the read-back values match the written pattern. If the values match set the success result. Else set the error result.
6. The content of the tested 64 bytes memory chunk is restored from the temporary memory buffer.
7. If the result shows error the test is done and returns.
8. The address pointer is incremented to point to the next sequential 64 bytes memory chunk and the test is repeated from step 1 until all the number of requested memory locations is tested.

Remarks

This is a non-destructive memory test. The content of the tested memory area is saved and restored. The test operates in 64 bytes long memory chunks at a time.

At least 32 bytes should be available for stack for executing the RAM Checker Board test. The tested RAM area must not overlap the stack.

The Start Address from which the Checker Board test is to be performed is PIC32 variant and application dependent. It is a run-time parameter.

The routine accesses one 4 byte RAM word at a time.

Preconditions

None.

Example

```
int testRes=CLASSB_RAMCheckerBoardTest(startAddress, size);
if(testRes==CLASSB_TEST_PASS)
{
    // process test success
}
else
{
    // process tests failure
}
```

Parameters

Parameters	Description
ramStartAddress	start Address from which the checker Board test is to be performed Must be properly 32 bit aligned.
ramSize	number of consecutive byte locations for which the test is to be performed The size must be a number multiple of 64.

Function

int CLASSB_RAMCheckerBoardTest (int* ramStartAddress, int ramSize)

CLASSB_RAMMarchBTest Function

The RAM March B test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_RAMMarchBTest(unsigned int* ramStartAddress, unsigned int ramSize);
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. RAM area tested has not been detected to have faults.

CLASSB_TEST_FAIL - The test failed. Some RAM area location has been detected to have faults.

Description

This test is a complete and non redundant test capable of detecting stuck-at, linked idempotent coupling or Inversion Coupling faults. This test is of complexity 17n(Where n is the number of bits tested). The test uses word (32-bit) accesses. The address must be properly word aligned and the length of the area to be tested must be an integral multiple of the data width access.

Remarks

This is a destructive memory test. Either exclude from this test RAM areas that have to be preserved or save/restore the memory area before/after running the test or run the test at system startup before the memory and the run time library is initialized (stack needs to be initialized though).

At least 100 bytes should be available for stack for executing the March B test. The tested RAM area must not overlap the stack.

Other statically allocated resources, such as the MPLAB ICD/Real ICE allocated RAM buffers should be excluded from this test.

The Start Address from which the March B test is to be performed is PIC32 variant and application dependent. It is a run-time parameter.

The routine accesses one 4 byte RAM word at a time.

Refer to the AN1229 for details regarding the CLASSB_RAMMarchBTest() and the Class B Software Library.

Preconditions

None.

Example

```
int testRes=CLASSB_RAMMarchBTest(startAddress, size);
if(testRes==CLASSB_TEST_PASS)
```

```
{
    // process test success
}
else
{
    // process tests failure
}
```

Parameters

Parameters	Description
ramStartAddress	start Address from which the March B test is to be performed Must be properly 32 bit aligned.
ramSize	number of consecutive byte locations for which the test is to be performed The size must be a number multiple of 4.

Function

int CLASSB_RAMMarchBTest (int* ramStartAddress, int ramSize)

CLASSB_RAMMarchCStackTest Function

The RAM March C test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_RAMMarchCStackTest(unsigned int* ramStartAddress, unsigned int ramSize);
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. RAM and Stack area tested have not been detected to have faults.

CLASSB_TEST_FAIL - The test failed. Either some RAM or Stack area location has been detected to have faults,

Description

This test is a complete and non redundant test capable of detecting stuck-at, addressing, transition and coupling faults. This test is of complexity 11n(Where n is the number of bits tested). The test uses word (32-bit) accesses. The addresses must be properly word aligned and the lengths of the areas to be tested must be an integral multiple of the data width access.

Remarks

This function is just a helper to March C test both a regular RAM area and a Stack area. First the RAM area is tested using the standard March C test. If the test succeeded, the requested Stack area is saved/copied in the RAM area that has just been tested and then the March C test is run over the Stack area as if it were a regular RAM area. The saved Stack area is restored and the result of the test is returned to the user.

The RAM and Stack areas have to NOT overlap! The Stack grows downwards so the tested area is: [stackTopAddress-stackSize, stackTopAddress] Also the size of the Stack area to be tested has to be less than the size of the RAM area.

The processor SP register is changed to the point to the RAM area while the Stack area is tested. Since running the MARC C test requires at least 128 bytes of stack, the RAM area size should be at least 128 bytes long. Once the Stack area is tested, the SP register is restored.

This is a destructive memory test. Either exclude from this test RAM areas that have to be preserved or save/restore the memory area before/after running the test or run the test at system startup before the memory and the run time library is initialized (stack needs to be initialized though).

At least 128 bytes should be available for stack for executing the March C test. The tested RAM area must not overlap the stack.

Other statically allocated resources, such as the MPLAB ICD/Real ICE allocated RAM buffers should be excluded from this test.

The Start Address from which the March C test is to be performed is PIC32 variant and application dependent. It is a run-time parameter.

The routine accesses one 4 byte RAM word at a time.

Preconditions

None.

Example

```
int testRes=CLASSB_RAMMarchCStackTest(startAddress, size);
if(testRes==CLASSB_TEST_PASS)
{
    // process test success
}
else
```

```
{
    // process tests failure
}
```

Parameters

Parameters	Description
ramStartAddress	start Address of RAM area to be used as temporary stack. The stack will be copied into this memory, and the stack pointer redirected here. Has to NOT overlap the Stack area! Must be properly 32 bit aligned.
ramSize	number of consecutive byte locations for which the test is to be performed The portion of the stack to be tested will be from the Stack Start (Highest address) to this address minus ramSize. This value also defines the amount of memory available as a temporary stack. The size must be a number multiple of 4. The size of the RAM area tested has to be >= 128 bytes.

Function

CLASSBRESULT CLASSB_RAMMarchCStackTest (unsigned int* ramStartAddress,

unsigned int ramSize)

CLASSB_RAMMarchCTest Function

The RAM March C test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.

File

classb_config_template.h

C

```
CLASSBRESULT CLASSB_RAMMarchCTest(unsigned int* ramStartAddress, unsigned int ramSize);
```

Returns

Result identifying the pass/fail status of the test: CLASSB_TEST_PASS - The test passed. RAM area tested has not been detected to have faults.

CLASSB_TEST_FAIL - The test failed. Some RAM area location has been detected to have faults.

Description

This test is a complete and non redundant test capable of detecting stuck-at, addressing, transition and coupling faults. This test is of complexity 11n(Where n is the number of bits tested). The test uses word (32-bit) accesses. The address must be properly word aligned and the length of the area to be tested must be an integral multiple of the data width access.

Remarks

This is a destructive memory test. Either exclude from this test RAM areas that have to be preserved or save/restore the memory area before/after running the test or run the test at system startup before the memory and the run time library is initialized (stack needs to be initialized though).

At least 100 bytes should be available for stack for executing the March C test. The tested RAM area must not overlap the stack.

Other statically allocated resources, such as the MPLAB ICD/Real ICE allocated RAM buffers should be excluded from this test.

The Start Address from which the March C test is to be performed is PIC32 variant and application dependent. It is a run-time parameter.

The routine accesses one 4 byte RAM word at a time.

Preconditions

None.

Example

```
int testRes=CLASSB_RAMMarchCTest(startAddress, size);
if(testRes==CLASSB_TEST_PASS)
{
    // process test success
}
else
{
    // process tests failure
}
```

Parameters

Parameters	Description
ramStartAddress	start Address from which the March C test is to be performed Must be properly 32 bit aligned.

ramSize	number of consecutive byte locations for which the test is to be performed The size must be a number multiple of 4.

Function

CLASSBRESULT CLASSB_RAMMarchCTest(unsigned int* ramStartAddress,
unsigned int ramSize)

Flash_CRC16 Function

File

classb_config_template.h

C

```
unsigned int Flash_CRC16(unsigned char* pBuff, unsigned int crcPoly, unsigned int crcReg, unsigned int
bSize);
```

Description

- Function: unsigned int Flash_CRC16(unsigned char* pBuff, unsigned int crcPoly, unsigned int crcReg, unsigned int bSize)
*
- PreCondition: pBuff valid pointer
*
- Input: - pBuff: buffer to calculate CRC over
- - crcPoly: the generator polynomial to be used
- - crcReg: initial value of the CRC LFSR (seed)
- - bSize: buffer size, bytes
*
- Output: value of the CRC
*
- Side Effects: None
*
- Overview: Shifts bytes through the CRC shift register.
*
- Note: Simple (and slow) CRC16 calculation directly based
- on the hardware LFSR implementation.

b) Data Types and Constants

CLASSBRESULT Enumeration

File

classb_config_template.h

C

```
typedef enum {
  CLASSB_TEST_PASS = 0,
  CLASSB_TEST_FAIL,
  CLASSB_TEST_TIMEOUT,
  CLASSB_TEST_INPROGRESS
} CLASSBRESULT;
```

Description

This is type CLASSBRESULT.

CLASSB_RAM_TEST_CYCLE_SIZE Macro

File

classb_config_template.h

C

```
#define CLASSB_RAM_TEST_CYCLE_SIZE 64
```

Description

This is macro CLASSB_RAM_TEST_CYCLE_SIZE.

CLOCK_TEST_INTERRUPT_SOURCE Macro

File

classb_config_template.h

C

```
#define CLOCK_TEST_INTERRUPT_SOURCE INT_SOURCE_TIMER_1
```

Description

This is macro CLOCK_TEST_INTERRUPT_SOURCE.

CLOCK_TEST_REFERENCE_FREQ Macro

Defines the reference frequency that is applied to the SOSC input.

File

classb_config_template.h

C

```
#define CLOCK_TEST_REFERENCE_FREQ 32768
```

Description

CPU Clock Test Secondary Oscillator (SOSC) reference frequency

This definition is used for internal calculation in the CLASSB_ClockTest() function. Normally this input is driven by a 32.768 KHz xtal. You can adjust this value if the need occurs (temperature drifts, etc).

CLOCK_TEST_TIMER Macro

File

classb_config_template.h

C

```
#define CLOCK_TEST_TIMER TMR_ID_1
```

Description

This is macro CLOCK_TEST_TIMER.

CRC_08_GEN_POLY Macro

File

classb_config_template.h

C

```
#define CRC_08_GEN_POLY 0x07     /* x^8 + x^2 + x + 1*/
```

Description

Flash CRC16 Test Generator Polynomials

The value of the generator polynomial is used as an input parameter for the CLASSB_CRCFlashTest() function. It specifies what polynomial to be used for the CRC calculation.

Following is a list of some of the most commonly used 16 bit Generator Polynomials that can be used. Any other polynomial that has the required fault detection capabilities can be used.

CRC_16_GEN_POLY Macro

File

classb_config_template.h

C

```
#define CRC_16_GEN_POLY 0x8005   /* x^16 + x^15 + x^2 + 1*/
```

Description

This is macro CRC_16_GEN_POLY.

CRC_32_GEN_POLY Macro

File

classb_config_template.h

C

```
#define CRC_32_GEN_POLY 0x04C11DB7  /* x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11 + x^10 + x^8 + x^7 +
x^5 + x^4 + x^2 + x + 1*/
```

Description

This is macro CRC_32_GEN_POLY.

CRC_CCITT_GEN_POLY Macro

File

classb_config_template.h

C

```
#define CRC_CCITT_GEN_POLY 0x1021   /* x^16 + x^12 + x^5 + 1*/
```

Description

This is macro CRC_CCITT_GEN_POLY.

FLASH_CRC16_MASK Macro

File

classb_config_template.h

C

```
#define FLASH_CRC16_MASK ((1L<<16)-1)            // mask to retain the useful CRC result for 16 bit
polynomial
```

Description

mask to retain the useful CRC result for 16 bit polynomial

FLASH_CRC16_MSB Macro

File

classb_config_template.h

C

```
#define FLASH_CRC16_MSB (1L<<(16-1))           // mask to obtain the MSb-1, transport to the MSb.
```

Description

mask to obtain the MSb-1, transport to the MSb.

FLASH_CRC32_MASK Macro

File

classb_config_template.h

C

```
#define FLASH_CRC32_MASK ((1L<<32)-1)            // mask to retain the useful CRC result for 32 bit
polynomial
```

Description

mask to retain the useful CRC result for 32 bit polynomial

FLASH_CRC32_MSB Macro

File

classb_config_template.h

C

```
#define FLASH_CRC32_MSB (1L<<(32-1))            // mask to obtain the MSb-1, transport to the MSb.
```

Description

mask to obtain the MSb-1, transport to the MSb.

FLASH_CRC8_MASK Macro

File

classb_config_template.h

C

```
#define FLASH_CRC8_MASK ((1L<<8)-1)              // mask to retain the useful CRC result for 8 bit
polynomial
```

Description

mask to retain the useful CRC result for 8 bit polynomial

FLASH_CRC8_MSB Macro

File

classb_config_template.h

C

```
#define FLASH_CRC8_MSB (1L<<(8-1))              // mask to obtain the MSb-1, transport to the MSb.
```

Description

mask to obtain the MSb-1, transport to the MSb.

USE_MARCHC_MINUS Macro

File

classb_config_template.h

C

```
#define USE_MARCHC_MINUS
```

Description

This is macro USE_MARCHC_MINUS.

Files

Files

Name	Description
classb_config_template.h	Class B interface header for definitions common to the Class B Library.

Description

This section lists the source and header files used by the Class B Library.

classb_config_template.h

Class B interface header for definitions common to the Class B Library.

Enumerations

	Name	Description
	CLASSBRESULT	This is type CLASSBRESULT.

Functions

	Name	Description
	CLASSB_ClockLineFreqTest	The CPU Clock Line Test is one of the tests that check the reliability of the system clock. It implements the independent time slot monitoring H.2.18.10.4 as defined by the IEC 60730 standard.
	CLASSB_ClockTest	The CPU Clock test is one of the tests that check the reliability of the system clock. It implements the independent time slot monitoring H.2.18.10.1 as defined by the IEC 60730 standard.
	CLASSB_CPUPCTest	The Program Counter test implements one of the functional tests H.2.16.5 as defined by the IEC 60730 standard.
	CLASSB_CPURegistersTest	The CPU Register test implements the functional test H.2.16.5 as defined by the IEC 60730 standard.
	CLASSB_CRCFlashTest	The Flash CRC16 test implements the periodic modified checksum H.2.19.3.1 as defined by the IEC 60730 standard.
	CLASSB_RAMCheckerBoardTest	The RAM Checker Board test implements one of the functional tests H.2.19.6 as defined by the IEC 60730 standard.
	CLASSB_RAMMarchBTest	The RAM March B test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.
	CLASSB_RAMMarchCStackTest	The RAM March C test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.
	CLASSB_RAMMarchCTest	The RAM March C test is one of the Variable Memory tests that implements the Periodic Static Memory test H.2.19.6 as defined by the IEC 60730 standard.
	Flash_CRC16	• Function: unsigned int Flash_CRC16(unsigned char* pBuff, unsigned int crcPoly, unsigned int crcReg, unsigned int bSize) * • PreCondition: pBuff valid pointer * • Input: - pBuff: buffer to calculate CRC over • - crcPoly: the generator polynomial to be used • - crcReg: initial value of the CRC LFSR (seed) • - bSize: buffer size, bytes * • Output: value of the CRC * • Side Effects: None * • Overview: Shifts bytes through the CRC shift register. * • Note: Simple (and slow) CRC16 calculation directly based • on the hardware LFSR implementation.

Macros

	Name	Description
	CLASSB_RAM_TEST_CYCLE_SIZE	This is macro CLASSB_RAM_TEST_CYCLE_SIZE.
	CLOCK_TEST_INTERRUPT_SOURCE	This is macro CLOCK_TEST_INTERRUPT_SOURCE.
	CLOCK_TEST_REFERENCE_FREQ	Defines the reference frequency that is applied to the SOSC input.
	CLOCK_TEST_TIMER	This is macro CLOCK_TEST_TIMER.
	CRC_08_GEN_POLY	Flash CRC16 Test Generator Polynomials The value of the generator polynomial is used as an input parameter for the CLASSB_CRCFlashTest() function. It specifies what polynomial to be used for the CRC calculation. Following is a list of some of the most commonly used 16 bit Generator Polynomials that can be used. Any other polynomial that has the required fault detection capabilities can be used.
	CRC_16_GEN_POLY	This is macro CRC_16_GEN_POLY.
	CRC_32_GEN_POLY	This is macro CRC_32_GEN_POLY.
	CRC_CCITT_GEN_POLY	This is macro CRC_CCITT_GEN_POLY.
	FLASH_CRC16_MASK	mask to retain the useful CRC result for 16 bit polynomial
	FLASH_CRC16_MSB	mask to obtain the MSb-1, transport to the MSb.
	FLASH_CRC32_MASK	mask to retain the useful CRC result for 32 bit polynomial
	FLASH_CRC32_MSB	mask to obtain the MSb-1, transport to the MSb.
	FLASH_CRC8_MASK	mask to retain the useful CRC result for 8 bit polynomial
	FLASH_CRC8_MSB	mask to obtain the MSb-1, transport to the MSb.
	USE_MARCHC_MINUS	This is macro USE_MARCHC_MINUS.

Description

Class B Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Class B Library for all families of Microchip microcontrollers. The definitions in this file are common to the Class B Library.

File Name

classb.h

Company

Microchip Technology Inc.

Crypto Library Help

This section describes the Cryptographic (Crypto) Library that is available in MPLAB Harmony.

Introduction

This library provides a software Cryptographic Library that is available on the Microchip family of microcontrollers with a convenient C language interface.

Description

The Cryptographic Library includes functions to perform encryption, decryption, hashing, authentication, and compression within the embedded application. Random number generation (RNG) functions are also provided.

Block Ciphers

The library provides DES, 3DES, and AES for block cipher needs. Depending on the algorithm in use, CBC and CTR modes are supported.

Public Key Cryptography

The library provides RSA and Elliptic Curve Cryptography (ECC) for Public Key Cryptography, and Diffie-Hellman (DH) for key agreement arrangements.

Hash Functions

The library provides MD5, SHA, SHA-256, SHA-384, and SHA-512 for hashing. These functions do not require keys or initialization vectors (IV).

Random Number Generation Functions

The library provides functions to generate either a single pseudo-random number, or a block of such numbers.

Using the Library

This topic describes the basic architecture of the Cryptographic (Crypto) Library and provides information and examples on its use.

Description

Interface Header File: `crypto.h`

The interface to the Crypto Library is defined in the `crypto.h` header file. Any C language source (`.c`) file that uses the Crypto Library should include `crypto.h`.

Library File:

The Crypto Library archive (`.a`) file is installed with MPLAB Harmony.

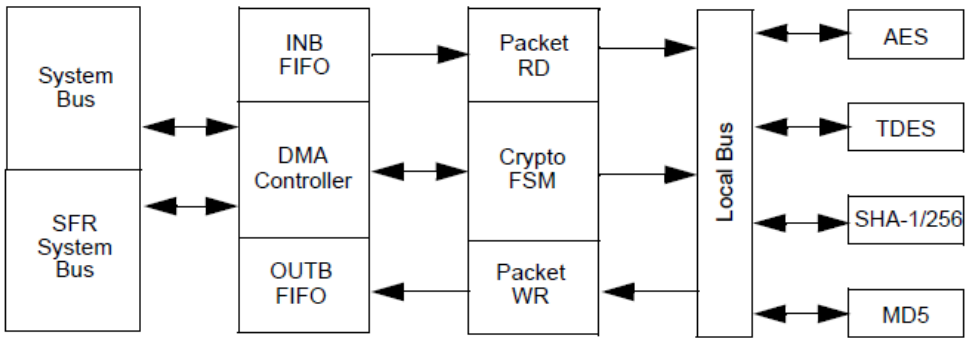
Please refer to the What is MPLAB Harmony? section for how the Crypto Library interacts with the framework.

Abstraction Model

This library provides the low-level abstraction of the Cryptographic Library module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Cryptographic (Crypto) Software Abstraction Block Diagram



Library Overview

The Library Interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Cryptographic (Crypto) Library module.

Library Interface Section	Description
General Functions	Provides an error string function, which takes an error and converts it into human-readable text.
Compression Functions	Provides Huffman compression and decompression functions.
MD5 Functions	Provides data add, finalize, and initialize MD5 functions.
Random Number Generator Functions	Provides get, initialize, and block generate RNG functions.
AES Encryption/Decryption Functions	Provides AES encryption and decryption functions.
ECC Encryption/Decryption Functions	Provides ECC encryption and decryption functions.
RSA Encryption/Decryption Functions	Provides RSA encryption and decryption functions.
Triple DES Encryption/Decryption Functions	Provides 3DES encryption and decryption functions.
HMAC Hash Functions	Provides HMAC data add, finalize, and set key Hash functions.
SHA Hash Functions	Provides SHA data add, finalize, and initialize Hash functions.
SHA256 Hash Functions	Provides SHA256 data add, finalize, and initialize Hash functions.
SHA384 Hash Functions	Provides SHA384 data add, finalize, and initialize Hash functions.
SHA512 Hash Functions	Provides SHA512 data add, finalize, and initialize Hash functions.

Configuring the Library

The configuration of the Cryptographic Library is based on the file `system_config.h`.

This header file contains the configuration selection for the Cryptographic Library. Based on the selections made, the Cryptographic Library may support the selected features. These configuration settings will apply to all instances of the Cryptographic Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Crypto Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/crypto`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`crypto.h`	Includes all MPLAB Harmony-compatible function calls for the Crypto Library.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
<install-dir>/build/framework/crypto/crypto.X	Include this project to bring in the library and all associated functions. There are two build configuration options available for this library: • SW – Software-only configuration for all Microchip PIC32 microcontrollers. • HW – Hardware acceleration configuration for PIC32 devices that have a hardware encryption module For both configurations, adjust the properties if 16-bit code is desired, and set the appropriate version of microcontroller.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
<install-dir>/build/framework/crypto/zlib.X	This project provides the code for Huffman compression in the Crypto Library. If the application does not use Huffman compression, this library may be excluded. There are two build configuration options available for this library: • PIC32MX – for PIC32MX devices • PIC32MZ – for PIC32MZ devices For both configurations, adjust the properties if 16-bit code is desired.

Module Dependencies

The Crypto Library does not depend on any other modules.

Library Interface

a) General Functions

	Name	Description
	CRYPT_RNG_Initialize	Initializes the random number generator.
	CRYPT_ERROR_StringGet	Reports the nature of an error.

b) Compression Functions

	Name	Description
	CRYPT_HUFFMAN_Compress	Compresses a block of data.
	CRYPT_HUFFMAN_DeCompress	Decompresses a block of data.

c) MD5 Functions

	Name	Description
	CRYPT_MD5_DataAdd	Updates the hash with the data provided.
	CRYPT_MD5_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_MD5_Initialize	Initializes the internal structures necessary for MD5 hash calculations.
	CRYPT_MD5_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.

d) Random Number Generator Functions

	Name	Description
	CRYPT_RNG_BlockGenerate	Createa several random numbers.
	CRYPT_RNG_Get	Gets one random number.

e) AES Encryption/Decryption Functions

	Name	Description
	CRYPT_AES_CBC_Decrypt	Performs AES decryption using Cipher-Block-Chaining (CBC).
	CRYPT_AES_CBC_Encrypt	Performs AES encryption using Cipher-Block-Chaining (CBC).
	CRYPT_AES_CTR_Encrypt	Performs AES encryption using Counter (CTR).
	CRYPT_AES_DIRECT_Decrypt	Directs decryption of one block of data.
	CRYPT_AES_DIRECT_Encrypt	Directs encryption of one block of data.

	CRYPT_AES_IvSet	Sets the initialization vector (IV) for AES processing.
	CRYPT_AES_KeySet	Sets the key and initialization vector (IV) for AES processing.

f) ECC Encryption/Decryption Functions

	Name	Description
	CRYPT_ECC_DHE_KeyMake	Creates a new ECC key.
	CRYPT_ECC_DHE_SharedSecretMake	Creates an ECC shared secret between two keys.
	CRYPT_ECC_DSA_HashSign	Signs a message digest.
	CRYPT_ECC_DSA_HashVerify	Verifies an ECC signature.
	CRYPT_ECC_Free	Cleans up an Elliptic Curve Cryptography (ECC) Context.
	CRYPT_ECC_Initialize	Initializes the context for Elliptic Curve Cryptography (ECC).
	CRYPT_ECC_KeySizeGet	Returns the key size in octets.
	CRYPT_ECC_PrivateImport	Imports private key pair in X9.63 format.
	CRYPT_ECC_PublicExport	Exports public ECC key in ANSI X9.63 format.
	CRYPT_ECC_PublicImport	Imports public key in ANSI X9.63 format.
	CRYPT_ECC_SignatureSizeGet	Returns the signature size in octets.

g) RSA Encryption/Decryption Functions

	Name	Description
	CRYPT_RSA_EncryptSizeGet	Gets the size of the RSA Key.
	CRYPT_RSA_Free	Releases the memory used for the key and clean up the context.
	CRYPT_RSA_Initialize	Initializes the internal structures necessary for RSA processing.
	CRYPT_RSA_PrivateDecrypt	Decrypts data using a private key.
	CRYPT_RSA_PrivateKeyDecode	Constructs the Private Key from a DER certificate.
	CRYPT_RSA_PublicEncrypt	Encrypts data using a public key.
	CRYPT_RSA_PublicKeyDecode	Constructs the Public Key from a DER certificate.

h) Triple DES (3DES) Encryption/Decryption Functions

	Name	Description
	CRYPT_TDES_CBC_Decrypt	Decrypts a data block using Triple DES.
	CRYPT_TDES_CBC_Encrypt	Encrypts a data block using Triple DES.
	CRYPT_TDES_IvSet	Sets the Initialization Vector (IV) for a Triple DES operation.
	CRYPT_TDES_KeySet	Initialization of Triple DES context.

i) HMAC Hash Functions

	Name	Description
	CRYPT_HMAC_DataAdd	Adds data to the HMAC calculation.
	CRYPT_HMAC_Finalize	Completes the HMAC calculation and get the results.
	CRYPT_HMAC_SetKey	Initializes the HMAC context and set the key for the hash.

j) SHA Hash functions

	Name	Description
	CRYPT_SHA_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA_Initialize	Initializes the internal structures necessary for SHA hash calculations.
	CRYPT_SHA_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.

k) SHA256 Hash Functions

	Name	Description
	CRYPT_SHA256_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA256_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA256_Initialize	Initializes the internal structures necessary for SHA256 hash calculations.
	CRYPT_SHA256_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.

l) SHA384 Hash Functions

	Name	Description
	CRYPT_SHA384_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA384_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA384_Initialize	Initializes the internal structures necessary for SHA384 hash calculations.

m) SHA512 Hash Functions

	Name	Description
	CRYPT_SHA512_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA512_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA512_Initialize	Initializes the internal structures necessary for SHA512 hash calculations.

n) Data Types and Constants

	Name	Description
	CRYPT_AES_CTX	AES
	CRYPT_ECC_CTX	ECC
	CRYPT_HMAC_CTX	HMAC
	CRYPT_MD5_CTX	MD5
	CRYPT_RNG_CTX	RNG
	CRYPT_RSA_CTX	RSA
	CRYPT_SHA_CTX	SHA
	CRYPT_SHA256_CTX	SHA-256
	CRYPT_SHA384_CTX	SHA-384
	CRYPT_SHA512_CTX	SHA-512
	CRYPT_TDES_CTX	TDES
	MC_CRYPTO_API_H	Defines Microchip CRYPTO API layer

Description

This section describes the Application Programming Interface (API) functions of the Cryptographic Library.

Refer to each section for a detailed description.

a) General Functions

CRYPT_RNG_Initialize Function

Initializes the random number generator.

File

crypto.h

C

```
int CRYPT_RNG_Initialize(CRYPT_RNG_CTX*);
```

Returns

- negative - An error occurred setting up the random number generator.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the context that stores information relative to random number generation.

Preconditions

None.

Example

```
#define RANDOM_BYTE_SZ 32

int           ret;
```

```
CRYPT_RNG_CTX mcRng;
byte          out[RANDOM_BYTE_SZ];

ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RNG_Get(&mcRng, &out[0]);

ret = CRYPT_RNG_BlockGenerate(&mcRng, out, RANDOM_BYTE_SZ);
```

Parameters

Parameters	Description
rng	Pointer to random number generator context.

Function

int CRYPT_RNG_Initialize(CRYPT_RNG_CTX* rng)

CRYPT_ERROR_StringGet Function

Reports the nature of an error.

File

crypto.h

C

```
int CRYPT_ERROR_StringGet(int, char*);
```

Returns

- BAD_FUNC_ARG - A null string was passed for the return message.
- 0 - A null string was not passed for the return message.

Description

This function takes an error and converts it into human-readable text.

Remarks

String needs to be >= 80 chars.

Preconditions

None.

Example

```
char msg[80];

CRYPT_ERR_StringGet(ret, msg);
```

Parameters

Parameters	Description
int	Error code to convert.
str	Pointer to buffer to store the message. Must hold at least 80 characters.

Function

int CRYPT_ERROR_StringGet(int err, char* str)

b) Compression Functions

CRYPT_HUFFMAN_Compress Function

Compresses a block of data.

File

crypto.h

C

```
int CRYPT_HUFFMAN_Compress(unsigned char*, unsigned int, const unsigned char*, unsigned int, unsigned int);
```

Returns

- negative - error code
- positive - bytes stored in out buffer

Description

This function compresses a block of data using Huffman encoding.

Remarks

Output buffer must be large enough to hold the contents of the operation.

Preconditions

None.

Example

```
const unsigned char text[] = "...";
unsigned int inSz = sizeof(text);
unsigned int outSz;
unsigned char cBuffer[1024];

int ret;

ret = CRYPT_HUFFMAN_COMPRESS(cBuffer, sizeof(cBuffer), text, inSz, 0);
```

Parameters

Parameters	Description
out	Pointer to location to store the compressed data.
outSz	Maximum size of the output data in bytes.
in	Point to location of source data.
inSz	Size of the input data in bytes.
flags	Flags to control how compress operates

Function

int CRYPT_HUFFMAN_Compress(unsigned char* out, unsigned int outSz, const unsigned char* in,
unsigned int inSz, unsigned int flags)

CRYPT_HUFFMAN_DeCompress Function

Decompresses a block of data.

File

crypto.h

C

```
int CRYPT_HUFFMAN_DeCompress(unsigned char*, unsigned int, const unsigned char*, unsigned int);
```

Returns

- negative - Error code
- positive - Bytes stored in out buffer

Description

This function decompresses a block of data using Huffman encoding.

Remarks

Output buffer must be large enough to hold the contents of the operation.

Preconditions

None.

Example

```
unsigned char cBuffer[1024];
unsigned char dBuffer[1024];

int ret

ret = CRYPT_HUFFMAN_DeCompress(dBuffer, sizeof(dBuffer), cBuffer, msglen);
```

Parameters

Parameters	Description
out	Pointer to destination buffer
outSz	Size of destination buffer
in	Pointer to source buffer to decompress
inSz	Size of source buffer to decompress

Function

int CRYPT_HUFFMAN_DeCompress(unsigned char* out, unsigned int outSz, const unsigned char* in, unsigned int inSz)

c) MD5 Functions

CRYPT_MD5_DataAdd Function

Updates the hash with the data provided.

File

crypto.h

C

```
int CRYPT_MD5_DataAdd(CRYPT_MD5_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in md5 or input
- 0 - An invalid pointer was not passed to the function

Description

This function updates the hash with the data provided.

Remarks

To preserve the validity of the MD5 hash, nothing must modify the context holding variable between calls to CRYPT_MD5_DataAdd.

Preconditions

The MD5 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_MD5_CTX md5;
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

CRYPT_MD5_Initialize(&md5);
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

int CRYPT_MD5_DataAdd(CRYPT_MD5_CTX* md5, const unsigned char* input, unsigned int sz)

CRYPT_MD5_Finalize Function

Finalizes the hash and puts the result into digest.

File

crypto.h

C

```
int CRYPT_MD5_Finalize(CRYPT_MD5_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in md5 or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the MD5 hash, nothing must modify the context holding variable between calls to CRYPT_MD5_DataAdd and CRYPT_MD5_Finalize.

Preconditions

The MD5 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_MD5_CTX md5;
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

CRYPT_MD5_Initialize(&md5);
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

int CRYPT_MD5_Finalize(CRYPT_MD5_CTX* md5, unsigned char* digest)

CRYPT_MD5_Initialize Function

Initializes the internal structures necessary for MD5 hash calculations.

File

crypto.h

C

```
int CRYPT_MD5_Initialize(CRYPT_MD5_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the internal structures necessary for MD5 hash calculations.

Remarks

All MD5 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_MD5_CTX md5;
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

CRYPT_MD5_Initialize(&md5);
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX structure which holds the hash values.

Function

int CRYPT_MD5_Initialize(CRYPT_MD5_CTX* md5)

CRYPT_MD5_DataSizeSet Function

For PIC32MZ hardware encryption, sets the size of the input data.

File

crypto.h

C

```
int CRYPT_MD5_DataSizeSet(CRYPT_MD5_CTX*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

The PIC32MZ hardware encryption module needs to know the size of the data before it starts processing. This function sets that value.

Remarks

All MD5 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_MD5_CTX md5;
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

    CRYPT_MD5_Initialize(&md5);
    CRYPT_MD5DataSizeSet(&md5, sizeof(buffer));
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX structure which holds the hash values.
msgSize	Size of the data (in bytes) that will be processed.

Function

int CRYPT_MD5_DataSizeSet(CRYPT_MD5_CTX* md5, unsigned int msgSize)

d) Random Number Generator Functions

CRYPT_RNG_BlockGenerate Function

Createa several random numbers.

File

crypto.h

C

```
int CRYPT_RNG_BlockGenerate(CRYPT_RNG_CTX*, unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function generates several random numbers and places them in the space allocated.

Preconditions

RNG context was initialized using the CRYPT_RNG_Initialize function.

Example

```
#define RANDOM_BYTE_SZ 32

int           ret;
CRYPT_RNG_CTX mcRng;
byte          out[RANDOM_BYTE_SZ];

ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RNG_Get(&mcRng, &out[0]);

ret = CRYPT_RNG_BlockGenerate(&mcRng, out, RANDOM_BYTE_SZ);
```

Parameters

Parameters	Description
rng	Pointer to context which saves state between calls.
b	Pointer to buffer to store the random numbers.
sz	Number of 8-bit random numbers to generate.

Function

int CRYPT_RNG_BlockGenerate(CRYPT_RNG_CTX* rng, unsigned char* b, unsigned int sz)

CRYPT_RNG_Get Function

Gets one random number.

File

crypto.h

C

```
int CRYPT_RNG_Get(CRYPT_RNG_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- Less than 0 - An error occurred.
- 0 or greater - Success.

Description

This function gets one random number from the random number generator.

Preconditions

RNG context was initialized using the CRYPT_RNG_Initialize function.

Example

```
#define RANDOM_BYTE_SZ 32

int           ret;
CRYPT_RNG_CTX mcRng;
byte          out[RANDOM_BYTE_SZ];

ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RNG_Get(&mcRng, &out[0]);

ret = CRYPT_RNG_BlockGenerate(&mcRng, out, RANDOM_BYTE_SZ);
```

Parameters

Parameters	Description
rng	Pointer to context which saves state between calls.
b	Pointer to 8-bit location to store the result.

Function

int CRYPT_RNG_Get(CRYPT_RNG_CTX* rng, unsigned char* b)

e) AES Encryption/Decryption Functions

CRYPT_AES_CBC_Decrypt Function

Performs AES decryption using Cipher-Block-Chaining (CBC).

File

crypto.h

C

```
int CRYPT_AES_CBC_Decrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function decrypts a block of data using the AES algorithm in Cipher- Block-Chaining (CBC) mode.

Remarks

The output buffer must be equal in size to the input buffer.

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to CRYPT_AES_KeySet and CRYPT_AES_IvSet.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;
byte          out1[AES_TEST_SIZE];
byte          out2[AES_TEST_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuv", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_CBC_Decrypt(&mcAes, out2, out1, AES_TEST_SIZE);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
out	Pointer to buffer to store the results of the decryption pass.
in	Pointer to buffer holding the data to be decrypted.
inSz	Size of the input data, in bytes.

Function

int CRYPT_AES_CBC_Decrypt(CRYPT_AES_CTX* aes, unsigned char* out,
const unsigned char* in, unsigned int inSz)

CRYPT_AES_CBC_Encrypt Function

Performs AES encryption using Cipher-Block-Chaining (CBC).

File

crypto.h

C

```
int CRYPT_AES_CBC_Encrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function encrypts a block of data using the AES algorithm in Cipher- Block-Chaining (CBC) mode.

Remarks

The output buffer must be equal in size to the input buffer.

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to CRYPT_AES_KeySet and CRYPT_AES_IvSet.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;
byte          out1[AES_TEST_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuv", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_CBC_Encrypt(&mcAes, out1, ourData, AES_TEST_SIZE);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
out	Pointer to buffer to store the results of the encryption pass.
in	Pointer to buffer holding the data to be encrypted.
inSz	Size of the input data, in bytes.

Function

int CRYPT_AES_CBC_Encrypt(CRYPT_AES_CTX* aes, unsigned char* out,
const unsigned char* in, unsigned int inSz)

CRYPT_AES_CTR_Encrypt Function

Performs AES encryption using Counter (CTR).

File

crypto.h

C

```
int CRYPT_AES_CTR_Encrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function encrypts a block of data using the AES algorithm in Counter (CTR) mode.

Remarks

The output buffer must be equal in size to the input buffer.

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to CRYPT_AES_KeySet and CRYPT_AES_IvSet.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;
byte          out1[AES_TEST_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuv", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_CTR_Encrypt(&mcAes, out1, ourData, AES_TEST_SIZE);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
out	Pointer to buffer to store the results of the encryption pass.
in	Pointer to buffer holding the data to be encrypted.
inSz	Size of the input data, in bytes.

Function

int CRYPT_AES_CTR_Encrypt(CRYPT_AES_CTX* aes, unsigned char* out,

const unsigned char* in, unsigned int inSz)

CRYPT_AES_DIRECT_Decrypt Function

Directs decryption of one block of data.

File

crypto.h

C

```
int CRYPT_AES_DIRECT_Decrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function decrypts one block of data, equal to the AES block size.

Remarks

Input and output buffers must be equal in size (CRYPT_AES_BLOCK_SIZE).

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to CRYPT_AES_KeySet and CRYPT_AES_IvSet.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;
byte          out1[CRYPT_AES_BLOCK_SIZE];
byte          out2[CRYPT_AES_BLOCK_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuv", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_DIRECT_Decrypt(&mcAes, out2, out1);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
out	Pointer to buffer to store the results of the decryption.
in	Pointer to buffer where the data to decrypt is located.

Function

int CRYPT_AES_DIRECT_Decrypt(CRYPT_AES_CTX*, unsigned char*,

const unsigned char*)

CRYPT_AES_DIRECT_Encrypt Function

Directs encryption of one block of data.

File

crypto.h

C

```
int CRYPT_AES_DIRECT_Encrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function encrypts one block of data, equal to the AES block size.

Remarks

Input and output buffers must be equal in size (CRYPT_AES_BLOCK_SIZE).

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to CRYPT_AES_KeySet and CRYPT_AES_IvSet.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;
byte          out1[CRYPT_AES_BLOCK_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuv", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_DIRECT_Encrypt(&mcAes, out1, ourData);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.

out	Pointer to buffer to store the results of the encryption.
in	Pointer to buffer where the data to encrypt is located.

Function

int CRYPT_AES_DIRECT_Encrypt(CRYPT_AES_CTX* aes, unsigned char* out,
const unsigned char* in)

CRYPT_AES_IvSet Function

Sets the initialization vector (IV) for AES processing.

File

crypto.h

C

```
int CRYPT_AES_IvSet(CRYPT_AES_CTX*, const unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function sets the IV that AES will use for later processing.

Remarks

Direction, as set previously in CRYPT_AES_KeySet, is preserved.

Preconditions

The key must be set previously with CRYPT_AES_KeySet.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;

strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_IvSet(&mcAes, iv);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
iv	Pointer to buffer holding the initialization vector.

Function

int CRYPT_AES_IvSet(CRYPT_AES_CTX* aes, const unsigned char* iv)

CRYPT_AES_KeySet Function

Sets the key and initialization vector (IV) for AES processing.

File

crypto.h

C

```
int CRYPT_AES_KeySet(CRYPT_AES_CTX*, const unsigned char*, unsigned int, const unsigned char*, int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function sets the key and IV, and the direction (encryption or decryption) that AES will later perform.

Preconditions

None.

Example

```
CRYPT_AES_CTX mcAes;
int           ret;

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuv", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
key	Pointer to buffer holding the key itself.
keyLen	Length of key in bytes.
iv	Pointer to buffer holding the initialization vector.
dir	Which operation (CRYPT_AES_ENCRYPTION or CRYPT_AES_DECRYPTION).

Function

int CRYPT_AES_KeySet(CRYPT_AES_CTX* aes, const unsigned char* key,

unsigned int keylen, const unsigned char* iv, int dir)

f) ECC Encryption/Decryption Functions

CRYPT_ECC_DHE_KeyMake Function

Creates a new ECC key.

File

crypto.h

C

```
int CRYPT_ECC_DHE_KeyMake(CRYPT_ECC_CTX*, CRYPT_RNG_CTX*, int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- MEMORY_E - Could not create the memory buffer for the key.
- 0 - An invalid pointer was not passed to the function.

Description

This function creates a new ECC key.

Preconditions

The context must have been initialized with a call to CRYPT_ECC_Initialize. The random number generator context must have been initialized with a call to CRYPT_RNG_Initialize.

Example

```
CRYPT_ECC_CTX userA;
int           ret;
byte          sharedA[100];
unsigned int  aSz = (unsigned int)sizeof(sharedA);
unsigned int  usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
ret = CRYPT_ECC_DHE_KeyMake(&userA, &mcRng, 32);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.
rng	Pointer to the context for the random number generator.
keySz	The size of the key desired.

Function

int CRYPT_ECC_DHE_KeyMake(CRYPT_ECC_CTX*, CRYPT_RNG_CTX*, int)

CRYPT_ECC_DHE_SharedSecretMake Function

Creates an ECC shared secret between two keys.

File

crypto.h

C

```
int CRYPT_ECC_DHE_SharedSecretMake(CRYPT_ECC_CTX*, CRYPT_ECC_CTX*, unsigned char*, unsigned int, unsigned
int*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- MEMORY_E - Could not create the memory buffer for the shared secret.
- 0 - An invalid pointer was not passed to the function.

Description

This function takes two ECC contexts (one public, one private) and creates a shared secret between the two. The secret conforms to EC-DH from ANSI X9.63.

Preconditions

Both contexts must have been initialized with a call to CRYPT_ECC_Initialize. Both contexts have had their respective keys imported or created.

Example

```
CRYPT_ECC_CTX userA;
CRYPT_ECC_CTX userB;
int           ret;
byte          sharedA[100];
unsigned int  aSz   = (unsigned int)sizeof(sharedA);
unsigned int  usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
ret = CRYPT_ECC_Initialize(&userB);
...
// Make or import the appropriate keys
...
ret = CRYPT_ECC_DHE_SharedSecretMake(&userA, &userB, sharedA, aSz, &usedA);
```

Parameters

Parameters	Description
priv	Pointer to the private ECC context (with the private key).
pub	Pointer to the public ECC context (with the public key).
out	Destination of the shared secret.
outSz	The max size of the shared secret.
usedSz	Resulting size of the shared secret.

Function

int CRYPT_ECC_DHE_SharedSecretMake(CRYPT_ECC_CTX* priv, CRYPT_ECC_CTX* pub,

unsigned char* out, unsigned int outSz, unsigned int* usedSz)

CRYPT_ECC_DSA_HashSign Function

Signs a message digest.

File

crypto.h

C

```
int CRYPT_ECC_DSA_HashSign(CRYPT_ECC_CTX*, CRYPT_RNG_CTX*, unsigned char*, unsigned int, unsigned int*,
const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function takes a message digest and signs it using a private ECC key.

Preconditions

The ECC context must have been initialized with a call to CRYPT_ECC_Initialize. The RNG context must have been initialized with a call to CRYPT_RNG_Initialize. The private key used for the signature must have been imported or created prior to calling this function.

Example

```
CRYPT_ECC_CTX userA;
int           ret;
byte          sig[100];
unsigned int  sigSz = (unsigned int)sizeof(sig);
unsigned int  usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
...
// Import or generate private key
...
ret = CRYPT_ECC_DSA_HashSign(&userA, &mcRng, sig, sigSz, &usedA, ourData,
                             CRYPT_SHA_DIGEST_SIZE);
```

Parameters

Parameters	Description
ecc	Pointer to ECC context which saves state between calls and holds keys.
rng	Pointer to Random Number Generator context.
sig	Destination for the signature.
sigSz	The max size of the signature, in bytes.
usedSz	The resulting size of the signature, in bytes.
in	Pointer to the message digest to sign.
inSz	The length of the digest, in bytes.

Function

int CRYPT_ECC_DSA_HashSign(CRYPT_ECC_CTX* ecc, CRYPT_RNG_CTX* rng, unsigned char* sig,

unsigned int sigSz, unsigned int* usedSz, const unsigned char* in, unsigned int inSz)

CRYPT_ECC_DSA_HashVerify Function

Verifies an ECC signature.

File

crypto.h

C

```
int CRYPT_ECC_DSA_HashVerify(CRYPT_ECC_CTX*, const unsigned char*, unsigned int, unsigned char*, unsigned
int, int*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- MEMORY_E - Memory could not be allocated for the operation.
- 0 - An invalid pointer was not passed to the function.

Description

This function verifies that an ECC signature is valid.

Preconditions

The ECC context must have been initialized with a call to CRYPT_ECC_Initialize. The key used for the signature must have been imported or created prior to calling this function.

Example

```
CRYPT_ECC_CTX userA;
int           ret;
byte          sig[100];
unsigned int  sigSz = (unsigned int)sizeof(sig);
unsigned int  usedA = 0;
int verifyStatus = 0;

ret = CRYPT_ECC_Initialize(&userA);
...
// Import or generate private key
...
ret = CRYPT_ECC_DSA_HashVerify(&userA, sig, sigSz, ourData,
                               CRYPT_SHA_DIGEST_SIZE, &verifyStatus);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.
sig	The signature to verify.
sigSz	The length of the signature (octets).
hash	The hash (message digest) that was signed.
hashSz	The length of the hash (octets).
status	Result of signature (1 == valid, 0 == invalid).

Function

int CRYPT_ECC_DSA_HashVerify(CRYPT_ECC_CTX* ecc, const unsigned char* sig,

unsigned int sigSz, unsigned char* hash, unsigned int hashSz, int* status)

CRYPT_ECC_Free Function

Cleans up an Elliptic Curve Cryptography (ECC) Context.

File

crypto.h

C

```
int CRYPT_ECC_Free(CRYPT_ECC_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function cleans up memory allocated for an ECC Context.

Preconditions

The context must have been initialized previously with a call to CRYPT_ECC_Initialize.

Example

```
CRYPT_ECC_CTX userA;
int           ret;

ret = CRYPT_ECC_Initialize(&userA);
...
ret = CRYPT_ECC_Free(&userA);
```

Parameters

Parameters	Description
ecc	Pointer to context to clean up.

Function

int CRYPT_ECC_Free(CRYPT_ECC_CTX* ecc)

CRYPT_ECC_Initialize Function

Initializes the context for Elliptic Curve Cryptography (ECC).

File

crypto.h

C

```
int CRYPT_ECC_Initialize(CRYPT_ECC_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- -1 - Unable to allocate memory for the keys.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the context used for Elliptic Curve Cryptography (ECC). The context is then passed to calls to perform key building, encryption, decryption, etc.

Preconditions

None.

Example

```
CRYPT_ECC_CTX userA;
int           ret;

ret = CRYPT_ECC_Initialize(&userA);
```

Parameters

Parameters	Description
ecc	Pointer to context to initialize.

Function

int CRYPT_ECC_Initialize(CRYPT_ECC_CTX* ecc)

CRYPT_ECC_KeySizeGet Function

Returns the key size in octets.

File

crypto.h

C

```
int CRYPT_ECC_KeySizeGet(CRYPT_ECC_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - The size of the key, in octets.

Description

This function returns the size of the ECC key, in octets.

Preconditions

The ECC context must have been initialized with a call to CRYPT_ECC_Initialize. The key must have been imported or created prior to calling this function.

Example

```
CRYPT_ECC_CTX userA;
int           ret;
byte          sig[100];
unsigned int  sigSz = (unsigned int)sizeof(sig);
unsigned int  usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
...
// Import or generate private key
...
ret = CRYPT_ECC_KeySizeGet(&userA);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls and contains the key.

Function

int CRYPT_ECC_KeySizeGet(CRYPT_ECC_CTX* ecc)

CRYPT_ECC_PrivateImport Function

Imports private key pair in X9.63 format.

File

crypto.h

C

```
int CRYPT_ECC_PrivateImport(CRYPT_ECC_CTX*, const unsigned char*, unsigned int, const unsigned char*,
unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function imports a public/private key pair in X9.63 format.

Preconditions

The context must have been initialized with a call to CRYPT_ECC_Initialize.

Example

```
CRYPT_ECC_CTX ecc;

CRYPT_ECC_Initialize(&ecc);
...
CRYPT_ECC_PrivateImport(&ecc, priv_key, sizeof(priv_key), pub_key, sizeof(pub_key));
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.

priv	Pointer to the private key.
privSz	Size of the private key, in bytes.
pub	Pointer to the public key.
pubSz	Size of the public key, in bytes.

Function

int CRYPT_ECC_PrivateImport(CRYPT_ECC_CTX* ecc, const unsigned char* priv,
unsigned int privSz, const unsigned char* pub,
unsigned int pubSz)

CRYPT_ECC_PublicExport Function

Exports public ECC key in ANSI X9.63 format.

File

crypto.h

C

```
int CRYPT_ECC_PublicExport(CRYPT_ECC_CTX*, unsigned char*, unsigned int, unsigned int*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- BUFFER_E - The output buffer was too small to store the key.
- 0 - An invalid pointer was not passed to the function.

Description

This function takes an ECC public key and exports it in ANSI X9.63 format.

Preconditions

The context must be initialized previously with a call to CRYPT_ECC_Initialize. The key must also have been constructed with a call to CRYPT_ECC_DHE_KeyMake. A random number generator must all have been initialized with a call to CRYPT_RNG_Initialize.

Example

```
CRYPT_ECC_CTX userA;
int           ret;
byte          sharedA[100];
unsigned int  aSz = (unsigned int)sizeof(sharedA);
unsigned int  usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
ret = CRYPT_ECC_DHE_KeyMake(&userA, &mcRng, 32);
ret = CRYPT_ECC_PublicExport(&userA, sharedA, aSz, &usedA);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.
out	Buffer in which to store the public key.
outSz	The available size of the buffer, in bytes.
usedSz	Return value indicating how many bytes were used.

Function

int CRYPT_ECC_PublicExport(CRYPT_ECC_CTX* ecc, unsigned char* out,
unsigned int outSz, unsigned int* usedSz)

CRYPT_ECC_PublicImport Function

Imports public key in ANSI X9.63 format.

File

crypto.h

C

```
int CRYPT_ECC_PublicImport(CRYPT_ECC_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- MEMORY_E - Memory could not be allocated for the key.
- ASN_PARSE_E - There was a parse error while going through the key.
- 0 - An invalid pointer was not passed to the function.

Description

This function imports a public key in ANSI X9.63 format into the ECC context.

Preconditions

The ECC context must have previously been initialized with a call to CRYPT_ECC_Initialize.

Example

```
CRYPT_ECC_CTX userB;
int           ret;
byte          sharedA[100];
unsigned int  aSz = (unsigned int)sizeof(sharedA);
unsigned int  usedA;

ret = CRYPT_ECC_Initialize(&userB);
...
ret = CRYPT_ECC_PublicImport(&userB, sharedA, usedA);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.
in	Input buffer the holds the public key.
inSz	Size of the input buffer, in bytes.

Function

int CRYPT_ECC_PublicImport(CRYPT_ECC_CTX* ecc, const unsigned char* in,

unsigned int inSz)

CRYPT_ECC_SignatureSizeGet Function

Returns the signature size in octets.

File

crypto.h

C

```
int CRYPT_ECC_SignatureSizeGet(CRYPT_ECC_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - The size of the signature.

Description

This function returns the size of the signature in a given context, in octets.

Preconditions

The ECC context must have been initialized with a call to CRYPT_ECC_Initialize. The keys must have been imported or created prior to calling this function.

Example

```
CRYPT_ECC_CTX userA;
int           ret;
byte          sig[100];
unsigned int  sigSz = (unsigned int)sizeof(sig);
```

```
unsigned int  usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
...
// Import or generate private key
...
ret = CRYPT_ECC_SignatureSizeGet(&userA);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls, and contains the signature.

Function

int CRYPT_ECC_SignatureSizeGet(CRYPT_ECC_CTX* ecc)

g) RSA Encryption/Decryption Functions

CRYPT_RSA_EncryptSizeGet Function

Gets the size of the RSA Key.

File

crypto.h

C

```
int CRYPT_RSA_EncryptSizeGet(CRYPT_RSA_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - Size of the key.

Description

This function retrieves the size of the RSA Key in use for the context.

Preconditions

The context must be initialized with a call to CRYPT_RSA_Initialize and the keys decoded either with CRYPT_RSA_PrivateKeyDecode or CRYPT_RSA_PublicKeyDecode.

Example

```
CRYPT_RSA_CTX mcRsa;
int           ret;
unsigned int  keySz = (unsigned int)sizeof(client_key_der_1024);
byte          out1[256];

ret = CRYPT_RSA_Initialize(&mcRsa);

ret = CRYPT_RSA_PrivateKeyDecode(&mcRsa, client_key_der_1024, keySz);

ret = CRYPT_RSA_EncryptSizeGet(&mcRsa);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.

Function

int CRYPT_RSA_EncryptSizeGet(CRYPT_RSA_CTX* rsa)

CRYPT_RSA_Free Function

Releases the memory used for the key and clean up the context.

File

crypto.h

C

```
int CRYPT_RSA_Free(CRYPT_RSA_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function releases the memory used during RSA processing for storing the public/private key.

Preconditions

The context must have been set up previously with a call to CRYPT_RSA_Initialize.

Example

```
CRYPT_RSA_CTX mcRsa;
int           ret;

ret = CRYPT_RSA_Initialize(&mcRsa);
ret = CRYPT_RSA_Free(&mcRsa);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.

Function

int CRYPT_RSA_Free(CRYPT_RSA_CTX* rsa)

CRYPT_RSA_Initialize Function

Initializes the internal structures necessary for RSA processing.

File

crypto.h

C

```
int CRYPT_RSA_Initialize(CRYPT_RSA_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- -1 - Unable to allocate the memory necessary for the key.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the context used during public-key RSA encryption and decryption.

Preconditions

None.

Example

```
CRYPT_RSA_CTX mcRsa;
int           ret;

ret = CRYPT_RSA_Initialize(&mcRsa);
```

Parameters

Parameters	Description
rsa	Pointer to RSA context which saves state between calls.

Function

int CRYPT_RSA_Initialize(CRYPT_RSA_CTX* rsa)

CRYPT_RSA_PrivateDecrypt Function

Decrypts data using a private key.

File

crypto.h

C

```
int CRYPT_RSA_PrivateDecrypt(CRYPT_RSA_CTX*, unsigned char*, unsigned int, const unsigned char*, unsigned
int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - Size of the actual output.

Description

This function decrypts a data block using a private key.

Preconditions

The context must be initialized using CRYPT_RSA_Initialized and the Private Key Decoded using CRYPT_RSA_PrivateKeyDecode prior to calling this function.

Example

```
CRYPT_RSA_CTX mcRsa;
int           ret;
unsigned int  keySz = (unsigned int)sizeof(client_key_der_1024);
byte          out1[256];

ret = CRYPT_RSA_Initialize(&mcRsa);

ret = CRYPT_RSA_PrivateKeyDecode(&mcRsa, client_key_der_1024, keySz);

ret = CRYPT_RSA_PrivateDecrypt(&mcRsa, out2, sizeof(out2), out1,
                               RSA_TEST_SIZE);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.
out	Pointer to output buffer to store results.
outSz	Size of output buffer.
in	Pointer to source buffer to be decrypted.
inSz	Size of input buffer.

Function

int CRYPT_RSA_PrivateDecrypt(CRYPT_RSA_CTX* rsa, unsigned char* out,

unsigned int outSz, const unsigned char* in, unsigned int inSz)

CRYPT_RSA_PrivateKeyDecode Function

Constructs the Private Key from a DER certificate.

File

crypto.h

C

```
int CRYPT_RSA_PrivateKeyDecode(CRYPT_RSA_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function builds a private key from a DER-formated certificate. DER stands for Distinguished Encoding Rules.

Preconditions

The context must have been initialized with a call to CRYPT_RSA_Initialize.

Example

```
CRYPT_RSA_CTX mcRsa;
int           ret;
unsigned int  keySz = (unsigned int)sizeof(client_key_der_1024);

ret = CRYPT_RSA_Initialize(&mcRsa);

ret = CRYPT_RSA_PrivateKeyDecode(&mcRsa, client_key_der_1024, keySz);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.
in	Pointer to buffer containing the certificate to process to extract the private key.
inSz	Size of the input data in bytes.

Function

int CRYPT_RSA_PrivateKeyDecode(CRYPT_RSA_CTX*, const unsigned char*,

unsigned int)

CRYPT_RSA_PublicEncrypt Function

Encrypts data using a public key.

File

crypto.h

C

```
int CRYPT_RSA_PublicEncrypt(CRYPT_RSA_CTX*, unsigned char*, unsigned int, const unsigned char*, unsigned
int, CRYPT_RNG_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - Size of the actual output.

Description

This function encrypts a data block using a public key.

Preconditions

The context must be initialized using CRYPT_RSA_Initialized and the Public Key Decoded using CRYPT_RSA_PublicKeyDecode prior to calling this function. The random number generator must be initialized with a call to CRYPT_RNG_Initialize.

Example

```
CRYPT_RSA_CTX mcRsa;
CRYPT_RNG_CTX mcRng;
int           ret;
unsigned int  keySz = (unsigned int)sizeof(client_key_der_1024);
byte          out1[256];

ret = CRYPT_RSA_Initialize(&mcRsa);
ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RSA_PublicKeyDecode(&mcRsa, client_key_der_1024, keySz);
```

```
ret = CRYPT_RSA_PublicEncrypt(&mcRsa, out1, sizeof(out1), ourData,
                              RSA_TEST_SIZE, &mcRng);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.
out	Pointer to output buffer to store results.
outSz	Size of output buffer.
in	Pointer to source buffer to be encrypted.
inSz	Size of input buffer.
rng	Pointer to Random Number Generator (RNG) context.

Function

int CRYPT_RSA_PublicEncrypt(CRYPT_RSA_CTX* rsa, unsigned char* out,
unsigned int outSz, const unsigned char* in, unsigned int inSz,
CRYPT_RNG_CTX* rng)

CRYPT_RSA_PublicKeyDecode Function

Constructs the Public Key from a DER certificate.

File

crypto.h

C

```
int CRYPT_RSA_PublicKeyDecode(CRYPT_RSA_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function builds a public key from a DER-formated certificate. DER stands for Distinguished Encoding Rules.

Preconditions

The context must have been initialized with a call to CRYPT_RSA_Initialize.

Example

```
CRYPT_RSA_CTX mcRsa;
int           ret;
unsigned int  keySz = (unsigned int)sizeof(client_key_der_1024);

ret = CRYPT_RSA_Initialize(&mcRsa);

ret = CRYPT_RSA_PublicKeyDecode(&mcRsa, client_key_der_1024, keySz);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.
in	Pointer to buffer containing the certificate to process to extract the public key.
inSz	Size of the input data in bytes.

Function

int CRYPT_RSA_PublicKeyDecode(CRYPT_RSA_CTX* rsa, const unsigned char* in,
unsigned int inSz)

h) Triple DES (3DES) Encryption/Decryption Functions

CRYPT_TDES_CBC_Decrypt Function

Decrypts a data block using Triple DES.

File

crypto.h

C

```
int CRYPT_TDES_CBC_Decrypt(CRYPT_TDES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function decrypts a block of data using a Triple DES algorithm.

Remarks

Input data must have a length a multiple of 8 bytes. Output data will be zero-padded at the end if the original data was not a multiple of 8 bytes long.

Preconditions

The context tdes must be set earlier using CRYPT_TDES_KeySet. The input block must be a multiple of 8 bytes long.

Example

```
CRYPT_TDES_CTX mcDes3;
int            ret;
byte           out1[TDES_SIZE];
byte           out2[TDES_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmn", 24);
strncpy((char*)iv,  "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);

ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);

ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
out	Pointer to output buffer to store the results.
in	Pointer to input buffer for the source of the data.
inSz	Size of the input data buffer.

Function

int CRYPT_TDES_CBC_Decrypt(CRYPT_TDES_CTX* tdes, unsigned char* out, const unsigned char* in, unsigned int inSz)

CRYPT_TDES_CBC_Encrypt Function

Encrypts a data block using Triple DES.

File

crypto.h

C

```
int CRYPT_TDES_CBC_Encrypt(CRYPT_TDES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function encrypts a block of data using a Triple DES algorithm.

Remarks

The input data must be padded at the end with zeros to make the length a multiple of 8.

Preconditions

The context tdes must be set earlier using CRYPT_TDES_KeySet. The input block must be a multiple of 8 bytes long.

Example

```
CRYPT_TDES_CTX mcDes3;
int            ret;
byte           out1[TDES_SIZE];
byte           out2[TDES_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmn", 24);
strncpy((char*)iv,  "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);

ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);

ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
out	Pointer to output buffer to store the results.
in	Pointer to input buffer for the source of the data.
inSz	Size of the input data buffer.

Function

int CRYPT_TDES_CBC_Encrypt(CRYPT_TDES_CTX* tdes, unsigned char* out, const unsigned char* in, unsigned int inSz)

CRYPT_TDES_IvSet Function

Sets the Initialization Vector (IV) for a Triple DES operation.

File

crypto.h

C

```
int CRYPT_TDES_IvSet(CRYPT_TDES_CTX*, const unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function changes the IV of a TDES context, but leaves the Key alone.

Remarks

The IV must be 8 bytes long.

Preconditions

None.

Example

```
CRYPT_TDES_CTX mcDes3;
int            ret;
byte           out1[TDES_SIZE];
byte           out2[TDES_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmn", 24);
strncpy((char*)iv,  "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);

ret = CRYPT_TDES_IvSet(&mcDes3, iv);

ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);

ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
iv	Pointer to buffer holding the initialization vector. Must be 8 bytes in size.

Function

int CRYPT_TDES_IvSet(CRYPT_TDES_CTX* tdes, const unsigned char* iv)

CRYPT_TDES_KeySet Function

Initialization of Triple DES context.

File

crypto.h

C

```
int CRYPT_TDES_KeySet(CRYPT_TDES_CTX*, const unsigned char*, const unsigned char*, int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function sets the key and initialization vector (IV) for a set of Triple-DES operations.

Remarks

The input data must be a multiple of 8 bytes, and must be padded at the end with zeros to meet the length.

Preconditions

None.

Example

```
CRYPT_TDES_CTX mcDes3;
int            ret;
byte           out1[TDES_SIZE];
byte           out2[TDES_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmn", 24);
strncpy((char*)iv,  "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);

ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);
```

```
ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
key	Pointer to buffer holding the key. Must be 24 bytes in size.
iv	Pointer to buffer holding the initialization vector. Must be 8 bytes in size.
dir	Indicates whether encryption or decryption will be done. Can be set to: • CRYPT_TDES_ENCRYPTION - For Encryption operations • CRYPT_TDES_DECRYPTION - Fro Decryption operations

Function

int CRYPT_TDES_KeySet(CRYPT_TDES_CTX* tdes, const unsigned char* key, const unsigned char* iv, int dir)

i) HMAC Hash Functions

CRYPT_HMAC_DataAdd Function

Adds data to the HMAC calculation.

File

crypto.h

C

```
int CRYPT_HMAC_DataAdd(CRYPT_HMAC_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function adds data to the HMAC so that multiple blocks of data can be processed.

Remarks

None.

Preconditions

The CRYPT_HMAC_CTX context must be initialized using the CRYPT_HMAC_SetKey function prior to any call to this function.

Example

```
CRYPT_HMAC_CTX mcHmac;
byte           mcDigest[CRYPT_SHA512_DIGEST_SIZE];

CRYPT_HMAC_SetKey(&mcHmac, CRYPT_HMAC_SHA, key, 4);

CRYPT_HMAC_DataAdd(&mcHmac, ourData, OUR_DATA_SIZE);

CRYPT_HMAC_Finalize(&mcHmac, mcDigest);
```

Parameters

Parameters	Description
hmac	Pointer to context that saves state between calls.
sz	Size of the input data in bytes.

Function

int CRYPT_HMAC_DataAdd(CRYPT_HMAC_CTX*, const unsigned char*, unsigned int)

CRYPT_HMAC_Finalize Function

Completes the HMAC calculation and get the results.

File

crypto.h

C

```
int CRYPT_HMAC_Finalize(CRYPT_HMAC_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function completes the HMAC calculations. The results are placed in the location pointed to by digest.

Remarks

The area pointed to by digest must be large enough to hold the results.

Preconditions

The CRYPT_HMAC_CTX context must be initialized using the CRYPT_HMAC_SetKey function prior to any call to this function.

Example

```
CRYPT_HMAC_CTX mcHmac;
byte           mcDigest[CRYPT_SHA512_DIGEST_SIZE];

CRYPT_HMAC_SetKey(&mcHmac, CRYPT_HMAC_SHA, key, 4);

CRYPT_HMAC_DataAdd(&mcHmac, ourData, OUR_DATA_SIZE);

CRYPT_HMAC_Finalize(&mcHmac, mcDigest);
```

Parameters

Parameters	Description
hmac	Pointer to context which saves state between calls.
digest	Pointer to place to put the final HMAC digest results.

Function

int CRYPT_HMAC_Finalize(CRYPT_HMAC_CTX* hmac, unsigned char* digest)

CRYPT_HMAC_SetKey Function

Initializes the HMAC context and set the key for the hash.

File

crypto.h

C

```
int CRYPT_HMAC_SetKey(CRYPT_HMAC_CTX*, int, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the HMAC context and set the key for the hash.

Remarks

None.

Preconditions

None.

Example

```
CRYPT_HMAC_CTX mcHmac;
byte           mcDigest[CRYPT_SHA512_DIGEST_SIZE];

CRYPT_HMAC_SetKey(&mcHmac, CRYPT_HMAC_SHA, key, 4);

CRYPT_HMAC_DataAdd(&mcHmac, ourData, OUR_DATA_SIZE);

CRYPT_HMAC_Finalize(&mcHmac, mcDigest);
```

Parameters

Parameters	Description
hmac	Pointer to context which saves state between calls.
type	Type of SHA operation to use with HMAC. Must be one of the
following	• CRYPT_HMAC_SHA • CRYPT_HMAC_SHA256 • CRYPT_HMAC_SHA384 • CRYPT_HMAC_SHA512
key	Secret key used for the hash.
sz	Size of the input data in bytes.

Function

int CRYPT_HMAC_SetKey(CRYPT_HMAC_CTX* hmac, int type, const unsigned char* key, unsigned int sz)

j) SHA Hash functions

CRYPT_SHA_DataAdd Function

Updates the hash with the data provided.

File

crypto.h

C

```
int CRYPT_SHA_DataAdd(CRYPT_SHA_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha or input.
- 0 - An invalid pointer was not passed to the function.

Description

This function updates the hash with the data provided.

Remarks

In order to preserve the validity of the SHA hash, nothing must modify the context holding variable between calls to CRYPT_SHA_DataAdd.

Preconditions

The SHA context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA_CTX sha;
uint8_t buffer[1024];
uint8_t shaSum[SHA_DIGEST_SIZE];

CRYPT_SHA_Initialize(&sha);
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));
```

```
CRYPT_SHA_Finalize(&sha, shaSum);
```

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

int CRYPT_SHA_DataAdd(CRYPT_SHA_CTX* sha, const unsigned char* input, unsigned int sz)

CRYPT_SHA_Finalize Function

Finalizes the hash and puts the result into digest.

File

crypto.h

C

```
int CRYPT_SHA_Finalize(CRYPT_SHA_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the SHA hash, nothing must modify the context holding variable between calls to CRYPT_SHA_DataAdd and CRYPT_SHA_Finalize.

Preconditions

The SHA context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA_CTX sha;
uint8_t buffer[1024];
uint8_t shaSum[SHA_DIGEST_SIZE];

CRYPT_SHA_Initialize(&sha);
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA_Finalize(&sha, shaSum);
```

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

int CRYPT_SHA_Finalize(CRYPT_SHA_CTX* sha, unsigned char* digest)

CRYPT_SHA_Initialize Function

Initializes the internal structures necessary for SHA hash calculations.

File

crypto.h

C

```
int CRYPT_SHA_Initialize(CRYPT_SHA_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the internal structures necessary for SHA hash calculations.

Remarks

All SHA hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA_CTX sha;
uint8_t shaSum[SHA_DIGEST_SIZE];

CRYPT_SHA_Initialize(&sha);
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA_Finalize(&sha, shaSum);
```

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.

Function

int CRYPT_SHA_Initialize(CRYPT_SHA_CTX* sha)

CRYPT_SHA_DataSizeSet Function

For PIC32MZ hardware encryption, sets the size of the input data.

File

crypto.h

C

```
int CRYPT_SHA_DataSizeSet(CRYPT_SHA_CTX*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

The PIC32MZ hardware encryption module needs to know the size of the data before it starts processing. This function sets that value.

Remarks

All SHA hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA_CTX sha;
uint8_t buffer[1024];
uint8_t shasum[SHA_DIGEST_SIZE];

    CRYPT_SHA_Initialize(&sha);
    CRYPT_SHADataSizeSet(&sha, sizeof(buffer));
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA_Finalize(&sha, shasum);
```

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.
msgSize	Size of the data (in bytes) that will be processed.

Function

int CRYPT_SHA_DataSizeSet(CRYPT_SHA_CTX* sha, unsigned int msgSize)

k) SHA256 Hash Functions

CRYPT_SHA256_DataAdd Function

Updates the hash with the data provided.

File

crypto.h

C

```
int CRYPT_SHA256_DataAdd(CRYPT_SHA256_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha256 or input.
- 0 - An invalid pointer was not passed to the function.

Description

This function updates the hash with the data provided.

Remarks

In order to preserve the validity of the SHA256 hash, nothing must modify the context holding variable between calls to CRYPT_SHA256_DataAdd.

Preconditions

The SHA256 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA256_CTX sha256;
uint8_t buffer[1024];
uint8_t shaSum[SHA256_DIGEST_SIZE];

CRYPT_SHA256_Initialize(&sha256);
CRYPT_SHA256_DataAdd(&sha256, buffer, sizeof(buffer));
CRYPT_SHA256_Finalize(&sha256, shaSum);
```

Parameters

Parameters	Description
sha256	Pointer to CRYPT_SHA256_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

int CRYPT_SHA256_DataAdd(CRYPT_SHA256_CTX* sha256, const unsigned char* input, unsigned int sz)

CRYPT_SHA256_Finalize Function

Finalizes the hash and puts the result into digest.

File

crypto.h

C

```
int CRYPT_SHA256_Finalize(CRYPT_SHA256_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the SHA256 hash, nothing must modify the context holding variable between calls to CRYPT_SHA256_DataAdd and CRYPT_SHA256_Finalize.

Preconditions

The SHA256 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA256_CTX sha256;
uint8_t buffer[1024];
uint8_t shaSum[SHA256_DIGEST_SIZE];

CRYPT_SHA256_Initialize(&sha256);
CRYPT_SHA256_DataAdd(&sha256, buffer, sizeof(buffer));
CRYPT_SHA256_Finalize(&sha256, shaSum);
```

Parameters

Parameters	Description
sha256	Pointer to CRYPT_SHA256_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

int CRYPT_SHA256_Finalize(CRYPT_SHA256_CTX* sha256, unsigned char* digest)

CRYPT_SHA256_Initialize Function

Initializes the internal structures necessary for SHA256 hash calculations.

File

crypto.h

C

```
int CRYPT_SHA256_Initialize(CRYPT_SHA256_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the internal structures necessary for SHA256 hash calculations.

Remarks

All SHA hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA256_CTX sha;
uint8_t shaSum[SHA256_DIGEST_SIZE];

CRYPT_SHA256_Initialize(&sha);
CRYPT_SHA256_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA256_Finalize(&sha, shaSum);
```

Parameters

Parameters	Description
sha256	Pointer to context which saves state between calls.

Function

int CRYPT_SHA256_Initialize(CRYPT_SHA256_CTX* sha256)

CRYPT_SHA256_DataSizeSet Function

For PIC32MZ hardware encryption, sets the size of the input data.

File

crypto.h

C

```
int CRYPT_SHA256_DataSizeSet(CRYPT_SHA256_CTX*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

The PIC32MZ hardware encryption module needs to know the size of the data before it starts processing. This function sets that value.

Remarks

All SHA256 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA256_CTX sha256;
uint8_t buffer[1024];
uint8_t sha256sum[SHA256_DIGEST_SIZE];

    CRYPT_SHA256_Initialize(&sha256);
    CRYPT_SHA256DataSizeSet(&sha256, sizeof(buffer));
CRYPT_SHA256_DataAdd(&sha256, buffer, sizeof(buffer));
CRYPT_SHA256_Finalize(&sha256, sha256sum);
```

Parameters

Parameters	Description
sha256	Pointer to CRYPT_SHA256_CTX structure which holds the hash values.
msgSize	Size of the data (in bytes) that will be processed.

Function

int CRYPT_SHA256_DataSizeSet(CRYPT_SHA256_CTX* sha256, unsigned int msgSize)

l) SHA384 Hash Functions

CRYPT_SHA384_DataAdd Function

Updates the hash with the data provided.

File

crypto.h

C

```
int CRYPT_SHA384_DataAdd(CRYPT_SHA384_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha384 or input.
- 0 - An invalid pointer was not passed to the function.

Description

This function updates the hash with the data provided.

Remarks

In order to preserve the validity of the SHA384 hash, nothing must modify the context holding variable between calls to CRYPT_SHA384_DataAdd.

Preconditions

The SHA384 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA384_CTX sha384;
uint8_t buffer[1024];
uint8_t shaSum[SHA384_DIGEST_SIZE];

CRYPT_SHA384_Initialize(&sha384);
CRYPT_SHA384_DataAdd(&sha384, buffer, sizeof(buffer));
CRYPT_SHA384_Finalize(&sha384, shaSum);
```

Parameters

Parameters	Description
sha384	Pointer to CRYPT_SHA384_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

int CRYPT_SHA384_DataAdd(CRYPT_SHA384_CTX* sha384, const unsigned char* input, unsigned int sz)

CRYPT_SHA384_Finalize Function

Finalizes the hash and puts the result into digest.

File

crypto.h

C

```
int CRYPT_SHA384_Finalize(CRYPT_SHA384_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha384 or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the SHA384 hash, nothing must modify the context holding variable between calls to CRYPT_SHA384_DataAdd and CRYPT_SHA384_Finalize.

Preconditions

The SHA384 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA384_CTX sha384;
uint8_t buffer[1024];
uint8_t shaSum[SHA384_DIGEST_SIZE];

CRYPT_SHA384_Initialize(&sha384);
CRYPT_SHA384_DataAdd(&sha384, buffer, sizeof(buffer));
CRYPT_SHA384_Finalize(&sha384, shaSum);
```

Parameters

Parameters	Description
sha384	Pointer to CRYPT_SHA384_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

int CRYPT_SHA384_Finalize(CRYPT_SHA384_CTX* sha384, unsigned char* digest)

CRYPT_SHA384_Initialize Function

Initializes the internal structures necessary for SHA384 hash calculations.

File

crypto.h

C

```
int CRYPT_SHA384_Initialize(CRYPT_SHA384_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function
- 0 - An invalid pointer was not passed to the function

Description

This function initializes the internal structures necessary for SHA384 hash calculations.

Remarks

All SHA384 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA384_CTX sha384;
uint8_t shaSum[SHA384_DIGEST_SIZE];

CRYPT_SHA384_Initialize(&sha384);
CRYPT_SHA384_DataAdd(&sha384, buffer, sizeof(buffer));
CRYPT_SHA384_Finalize(&sha384, shaSum);
```

Parameters

Parameters	Description
sha384	Pointer to CRYPT_SHA384_CTX structure which holds the hash values.

Function

int CRYPT_SHA384_Initialize(CRYPT_SHA384_CTX* sha384)

m) SHA512 Hash Functions

CRYPT_SHA512_DataAdd Function

Updates the hash with the data provided.

File

crypto.h

C

```
int CRYPT_SHA512_DataAdd(CRYPT_SHA512_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha512 or input.
- 0 - An invalid pointer was not passed to the function.

Description

This function updates the hash with the data provided.

Remarks

In order to preserve the validity of the SHA512 hash, nothing must modify the context holding variable between calls to CRYPT_SHA512_DataAdd.

Preconditions

The SHA512 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA512_CTX sha512;
uint8_t buffer[1024];
uint8_t sha512Sum[SHA512_DIGEST_SIZE];

CRYPT_SHA512_Initialize(&sha512);
CRYPT_SHA512_DataAdd(&sha512, buffer, sizeof(buffer));
CRYPT_SHA512_Finalize(&sha512, sha512Sum);
```

Parameters

Parameters	Description
sha512	Pointer to CRYPT_SHA512_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

int CRYPT_SHA512_DataAdd(CRYPT_SHA512_CTX* sha512, const unsigned char* input, unsigned int sz)

CRYPT_SHA512_Finalize Function

Finalizes the hash and puts the result into digest.

File

crypto.h

C

```
int CRYPT_SHA512_Finalize(CRYPT_SHA512_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha512 or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the SHA512 hash, nothing must modify the context holding variable between calls to CRYPT_SHA512_DataAdd and CRYPT_SHA512_Finalize.

Preconditions

The SHA512 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA512_CTX sha512;
uint8_t buffer[1024];
uint8_t sha512Sum[SHA512_DIGEST_SIZE];

CRYPT_SHA512_Initialize(&sha512);
CRYPT_SHA512_DataAdd(&sha512, buffer, sizeof(buffer));
CRYPT_SHA512_Finalize(&sha512, sha512Sum);
```

Parameters

Parameters	Description
sha512	Pointer to CRYPT_SHA512_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

int CRYPT_SHA512_Finalize(CRYPT_SHA512_CTX* sha512, unsigned char* digest)

CRYPT_SHA512_Initialize Function

Initializes the internal structures necessary for SHA512 hash calculations.

File

crypto.h

C

```
int CRYPT_SHA512_Initialize(CRYPT_SHA512_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the internal structures necessary for SHA512 hash calculations.

Remarks

All SHA512 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA512_CTX sha512;
uint8_t sha512Sum[SHA512_DIGEST_SIZE];

CRYPT_SHA512_Initialize(&sha512);
CRYPT_SHA512_DataAdd(&sha512, buffer, sizeof(buffer));
CRYPT_SHA512_Finalize(&sha512, sha512Sum);
```

Parameters

Parameters	Description
sha512	Pointer to CRYPT_SHA512_CTX structure which holds the hash values.

Function

int CRYPT_SHA512_Initialize(CRYPT_SHA512_CTX* sha512)

n) Data Types and Constants

CRYPT_AES_CTX Structure

File

crypto.h

C

```
struct CRYPT_AES_CTX {
  int holder[90];
};
```

Members

Members	Description
int holder[90];	big enough to hold internal, but check on init

Description

AES

CRYPT_ECC_CTX Structure

File

crypto.h

C

```
struct CRYPT_ECC_CTX {
  void* holder;
};
```

Description

ECC

CRYPT_HMAC_CTX Structure

File

crypto.h

C

```
struct CRYPT_HMAC_CTX {
  long long holder[80];
};
```

Members

Members	Description
long long holder[80];	big enough to hold internal, but check on init

Description

HMAC

CRYPT_MD5_CTX Structure

File

crypto.h

C

```
struct CRYPT_MD5_CTX {
  int holder[110];
};
```

Members

Members	Description
int holder[110];	big enough to hold internal, but check on init

Description

MD5

CRYPT_RNG_CTX Structure

File

crypto.h

C

```
struct CRYPT_RNG_CTX {
  int holder[66];
};
```

Members

Members	Description
int holder[66];	big enough to hold internal, but check on init

Description

RNG

CRYPT_RSA_CTX Structure

File

crypto.h

C

```
struct CRYPT_RSA_CTX {
  void* holder;
};
```

Description

RSA

CRYPT_SHA_CTX Structure

File

crypto.h

C

```
struct CRYPT_SHA_CTX {
  int holder[110];
};
```

Members

Members	Description
int holder[110];	big enough to hold internal, but check on init

Description

SHA

CRYPT_SHA256_CTX Structure

File

crypto.h

C

```
struct CRYPT_SHA256_CTX {
  int holder[110];
};
```

Members

Members	Description
int holder[110];	big enough to hold internal, but check on init

Description

SHA-256

CRYPT_SHA384_CTX Structure

File

crypto.h

C

```
struct CRYPT_SHA384_CTX {
  long long holder[32];
};
```

Members

Members	Description
long long holder[32];	big enough to hold internal, but check on init

Description

SHA-384

CRYPT_SHA512_CTX Structure

File

crypto.h

C

```
struct CRYPT_SHA512_CTX {
  long long holder[36];
};
```

Members

Members	Description
long long holder[36];	big enough to hold internal, but check on init

Description

SHA-512

CRYPT_TDES_CTX Structure

File

crypto.h

C

```
struct CRYPT_TDES_CTX {
  int holder[100];
};
```

Members

Members	Description
int holder[100];	big enough to hold internal, but check on init

Description

TDES

MC_CRYPTO_API_H Macro

File

crypto.h

C

```
#define MC_CRYPTO_API_H
```

Description

Defines Microchip CRYPTO API layer

Files

Files

Name	Description
crypto.h	Crypto Framework Library header for cryptographic functions.

Description

This section lists the source and header files used by the Crypto Library.

crypto.h

Crypto Framework Library header for cryptographic functions.

Functions

	Name	Description
	CRYPT_AES_CBC_Decrypt	Performs AES decryption using Cipher-Block-Chaining (CBC).
	CRYPT_AES_CBC_Encrypt	Performs AES encryption using Cipher-Block-Chaining (CBC).
	CRYPT_AES_CTR_Encrypt	Performs AES encryption using Counter (CTR).
	CRYPT_AES_DIRECT_Decrypt	Directs decryption of one block of data.
	CRYPT_AES_DIRECT_Encrypt	Directs encryption of one block of data.
	CRYPT_AES_IvSet	Sets the initialization vector (IV) for AES processing.
	CRYPT_AES_KeySet	Sets the key and initialization vector (IV) for AES processing.
	CRYPT_ECC_DHE_KeyMake	Creates a new ECC key.
	CRYPT_ECC_DHE_SharedSecretMake	Creates an ECC shared secret between two keys.
	CRYPT_ECC_DSA_HashSign	Signs a message digest.
	CRYPT_ECC_DSA_HashVerify	Verifies an ECC signature.
	CRYPT_ECC_Free	Cleans up an Elliptic Curve Cryptography (ECC) Context.
	CRYPT_ECC_Initialize	Initializes the context for Elliptic Curve Cryptography (ECC).
	CRYPT_ECC_KeySizeGet	Returns the key size in octets.
	CRYPT_ECC_PrivateImport	Imports private key pair in X9.63 format.
	CRYPT_ECC_PublicExport	Exports public ECC key in ANSI X9.63 format.
	CRYPT_ECC_PublicImport	Imports public key in ANSI X9.63 format.
	CRYPT_ECC_SignatureSizeGet	Returns the signature size in octets.
	CRYPT_ERROR_StringGet	Reports the nature of an error.
	CRYPT_HMAC_DataAdd	Adds data to the HMAC calculation.
	CRYPT_HMAC_Finalize	Completes the HMAC calculation and get the results.
	CRYPT_HMAC_SetKey	Initializes the HMAC context and set the key for the hash.
	CRYPT_HUFFMAN_Compress	Compresses a block of data.
	CRYPT_HUFFMAN_DeCompress	Decompresses a block of data.
	CRYPT_MD5_DataAdd	Updates the hash with the data provided.
	CRYPT_MD5_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.
	CRYPT_MD5_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_MD5_Initialize	Initializes the internal structures necessary for MD5 hash calculations.
	CRYPT_RNG_BlockGenerate	Createa several random numbers.
	CRYPT_RNG_Get	Gets one random number.
	CRYPT_RNG_Initialize	Initializes the random number generator.
	CRYPT_RSA_EncryptSizeGet	Gets the size of the RSA Key.
	CRYPT_RSA_Free	Releases the memory used for the key and clean up the context.
	CRYPT_RSA_Initialize	Initializes the internal structures necessary for RSA processing.
	CRYPT_RSA_PrivateDecrypt	Decrypts data using a private key.
	CRYPT_RSA_PrivateKeyDecode	Constructs the Private Key from a DER certificate.
	CRYPT_RSA_PublicEncrypt	Encrypts data using a public key.

	CRYPT_RSA_PublicKeyDecode	Constructs the Public Key from a DER certificate.
	CRYPT_SHA_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.
	CRYPT_SHA_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA_Initialize	Initializes the internal structures necessary for SHA hash calculations.
	CRYPT_SHA256_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA256_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.
	CRYPT_SHA256_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA256_Initialize	Initializes the internal structures necessary for SHA256 hash calculations.
	CRYPT_SHA384_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA384_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA384_Initialize	Initializes the internal structures necessary for SHA384 hash calculations.
	CRYPT_SHA512_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA512_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA512_Initialize	Initializes the internal structures necessary for SHA512 hash calculations.
	CRYPT_TDES_CBC_Decrypt	Decrypts a data block using Triple DES.
	CRYPT_TDES_CBC_Encrypt	Encrypts a data block using Triple DES.
	CRYPT_TDES_IvSet	Sets the Initialization Vector (IV) for a Triple DES operation.
	CRYPT_TDES_KeySet	Initialization of Triple DES context.

Macros

	Name	Description
	MC_CRYPTO_API_H	Defines Microchip CRYPTO API layer

Structures

	Name	Description
	CRYPT_AES_CTX	AES
	CRYPT_ECC_CTX	ECC
	CRYPT_HMAC_CTX	HMAC
	CRYPT_MD5_CTX	MD5
	CRYPT_RNG_CTX	RNG
	CRYPT_RSA_CTX	RSA
	CRYPT_SHA_CTX	SHA
	CRYPT_SHA256_CTX	SHA-256
	CRYPT_SHA384_CTX	SHA-384
	CRYPT_SHA512_CTX	SHA-512
	CRYPT_TDES_CTX	TDES

Description

Crypto Framework Library Header

This header file contains function prototypes and definitions of the data types and constants that make up the Cryptographic Framework Library for PIC32 families of Microchip microcontrollers.

File Name

crypto.h

Company

Microchip Technology Inc.

Decoder Libraries Help

This section provides descriptions of the Decoder libraries that are available in MPLAB Harmony.

AAC Decoder Library

This section describes the AAC Decoder Library.

Introduction

The AAC Decoder Library is available for the PIC32MX family of microcontrollers.

Description

The AAC Decoder Library is a premium product and is not included in the standard release of MPLAB Harmony and must be purchased separately. Refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for additional information.

This AAC algorithm Non-Modifiable Binary Code is designed for 80 MHz or greater PIC32MX MCUs. Keep in mind, this code requires 62 MIPS peak 34 MIPS average performance, 61 KB Flash and 12 KB RAM without frame buffer memory for operation on the PIC32MX MCU. This product is the non-modifiable binary. A modifiable source-code version of this code is also available on Microchip Direct (SW320013-2). Users are responsible for licensing for their products through Via Licensing.

Assembly Language Issue

There is an assembly language issue in the AAC Decoder Library that misaligns the stack pointer. Contrary to the MIPS programming specification, this error is benign except when the application is built for a device with a hardware Floating Point Unit (FPU), such as the PIC32MZ EF family of devices. In this case, the sub-routine context-saving operations will produce an exception and stop the application. A work around exists for this issue, which involves disabling FPU context-saving for Interrupt Service Routines (ISRs).

The work around is to convert all ISRs from:

```
void __ISR(vector,ipl) MyInterruptHandler(void)
```

and change them to:

```
void __attribute__((vector),interrupt(ipl),nomips16,no_fpu)) MyInterruptHandler(void)
```

where, `vector` and `ipl` are replaced with the actual interrupt vector and interrupt priority level.

An example of this work around can be found in the universal_audio_decoders demonstration.

So as long as no FPU operations occur in ISRs and the routines are called by ISRs, this work around will prevent the misaligned stack pointer from causing an exception. If FPU operations result from an ISR with no_fpu enabled, a general exception will result.

Using the Library

This topic describes the basic architecture of the AAC Decoder Library and provides information and examples on its use.

Description

Interface Header File: `decoder_aac.h`

The interface to the AAC Decoder Library is defined in the `decoder_aac.h` header file. Any C language source (`.c`) file that uses the MP3 Decoder Library should include `decoder_aac.h`.

Please refer to the What is MPLAB Harmony? section for how the AAC Decoder Library interacts with the framework.

Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the AAC Decoder Library module.

Library Interface

a) General Functions

	Name	Description
	AAC_Decoder	This is function AAC_Decoder.
	AAC_WriteWavHeader	Generates the .wav file header, if applicable.
	AACDecoder_GetFrameSize	Used to get the AAC Decoder frame size.
	AACDecoder_GetStateSize	This function allocates memory for each channel being used.

	AACDecoder_Init	Views the received point and clears the structure.
	AACDecoder_InterleaveSamples	This function interleaves left and right channel data.
	isAACdecoder_enabled	This is function isAACdecoder_enabled.
	AAC_Decode	This function decodes AAC data.
	AAC_Initialize	This is function AAC_Initialize.
	AAC_RegisterDecoderEventHandlerCallback	This is function AAC_RegisterDecoderEventHandlerCallback.
	AAC_GetSamplingFrequency	This is function AAC_GetSamplingFrequency.

b) Data Types and Constants

	Name	Description
	INTPCM	
	AAC_DECODER_STATES	This is type AAC_DECODER_STATES.
	WAVE	This is type WAVE.
	AAC_PROFILE	This is macro AAC_PROFILE.
	AAC_SAMPLING_FREQUENCY_INDEX	This is type AAC_SAMPLING_FREQUENCY_INDEX.
	AAC_ERROR_COUNT_MAX	This is macro AAC_ERROR_COUNT_MAX.
	AACDECODER_STATE_SIZE	in Bytes
	FRAME_SIZE	in Samples
	INPUT_BUF_SIZE	in Bytes
	MAX_STACK	in Bytes
	STACK	This is macro STACK.
	AAC_FRAME_HEADER_SIZE	in Bytes
	SetReadBytesInAppData	This is type SetReadBytesInAppData.

Description

This section describes the Application Programming Interface (API) functions of the AAC Decoder Library.

Refer to each section for a detailed description.

a) General Functions

AAC_Decoder Function

File

aac.h

C

```
int16_t AAC_Decoder(uint8_t * input, uint16_t inSize, uint16_t * read, int16_t * output, uint16_t *
written);
```

Description

This is function AAC_Decoder.

AAC_WriteWavHeader Function

Generates the .wav file header, if applicable.

File

decoder_aac.h

C

```
void AAC_WriteWavHeader(void * haacDecoderInfo, WAVE * wav);
```

Returns

None.

Description

This function is called only if the output needs to be written into a .wav file. If the output is for play out or for writing into a .pcm file, this function

shall not be called. In case of wave file generation, the application shall leave 44 bytes space free at the start of the file and begin writing decoded samples from the 45th byte location onwards. Once the entire data had been decoded, this function is called. The function generates the .wav file header using the information available in the state memory and fills the WaveBuf array. The application copies the content of this buffer into the file in the reserved place available at the start of the file.

Preconditions

None.

Example

```
AAC_WriteWavHeader(haacDecoderInfo, wav);
```

Parameters

Parameters	Description
*haacDecoderInfo	The pointer for the structure that stores decoder information
*adts_ptr	pointer to WAVE header buffer

Function

void AAC_WriteWavHeader(void *haacDecoderInfo, WAVE *wav);

AACDecoder_GetFrameSize Function

Used to get the AAC Decoder frame size.

File

decoder_aac.h

C

```
int32_t AACDecoder_GetFrameSize(void * haacDecoderInfo, uint8_t * adts_ptr);
```

Returns

This function returns the size (in bytes) of the next frame to be decoded.

Description

This function is called every time before calling the AAC decode function. The frame size is extracted from the relevant field of the ADTS header for returning and also the frame size (in bits) is updated in the channel state.

Preconditions

None.

Parameters

Parameters	Description
*haacDecoderInfo	The pointer to the ADTS Header is passed. The application extracts the next available seven bytes from the file and passes them as ADTS header.
*adts_ptr	The pointer to the state memory is passed.

Function

int32_t AACDecoder_GetFrameSize(void *haacDecoderInfo, uint8_t *adts_ptr);

AACDecoder_GetStateSize Function

This function allocates memory for each channel being used.

File

decoder_aac.h

C

```
int32_t AACDecoder_GetStateSize();
```

Returns

This function returns the size of the AAC Decoder state structure in bytes.

Description

Based on the return value, the application allocates the required memory for the AAC Decoder state. This function is called only once for each channel before the channel is initialized.

If dynamic memory allocation (malloc) is not to be used, this function need not be called and the AAC Decoder state buffer shall be statically defined at compile time with the size of the AAC Decoder state structure (constant).

Preconditions

None.

Example

```
int32_t aac_State_Size;
aac_State_Size = AACDecoder_GetStateSize();
```

Function

int32_t AACDecoder_GetStateSize(void)

AACDecoder_Init Function

Views the received point and clears the structure.

File

decoder_aac.h

C

```
void AACDecoder_Init(void * haacDecoderInfo, uint8_t * adts_ptr);
```

Returns

None.

Description

This function is called only once for each channel before starting the decoding of the first frame. This function views the received pointer as structure pointer and clears the entire structure. Subsequently it initializes the selective variables with the default values. Also, the information on sampling frequency and number of channels are extracted from the header and are stored in the channel state structure.

After returning from this function, the application resets the file pointer to origin as these seven bytes will be fetched again to extract the frame size through the function mentioned next.

Preconditions

None.

Example

```
int32_t aac_State_Size;
AACDecoder_Init(haacDecoderInfo, appData.data);
```

Parameters

Parameters	Description
*haacDecoderInfo	The pointer for the structure that stores decoder information
*adts_ptr	pointer to AAC header structure

Function

void AACDecoder_Init(void *haacDecoderInfo, uint8_t *adts_ptr);

AACDecoder_InterleaveSamples Function

This function interleaves left and right channel data.

File

decoder_aac.h

C

```
int16_t AACDecoder_InterleaveSamples(INTPCM * pTimeCh0, void * state);
```

Returns

This function returns the number of samples in the I/O buffer:

- 1024 - for mono
- 2048 - for stereo

Description

For stereo audio, the left and right channels are interleaved here and stored in the same I/O buffer. For mono channels, nothing is done.

Preconditions

None.

Example

```
outSize = AACDecoder_InterleaveSamples(outBuf, haacDecoderInfo);
```

Parameters

Parameters	Description
*pState	The pointer to the state memory is passed. The details about the number of channels, mono or stereo, is used.
*outBuf	Pointer to the I/O buffer.

Function

int32_t AACDecoder_InterleaveSamples (void *pState, int16_t *outBuf);

isAACdecoder_enabled Function

File

aac.h

C

```
bool isAACdecoder_enabled();
```

Description

This is function isAACdecoder_enabled.

AAC_Decode Function

This function decodes AAC data.

File

decoder_aac.h

C

```
int16_t AAC_Decode(void * DecoderState, uint8_t * inBufferptr, INTPCM * pTimeData, int16_t * outFlag,
int32_t * restBytes);
```

Returns

This function returns either the value '0' or '1' to indicate the status of the decode operation.

- 0 - indicates success
- 1 - indicates failure or end of available encoded data. Upon receiving the value '1', the application aborts processing the channel.

Description

This function is called once to decode one frame. This function will be called in an infinite while loop until the end of frame is detected or the decode function returns the value '1'.

Preconditions

None.

Example

```
errorStatus = AAC_Decode(haacDecoderInfo, inBuffer, outBuf, &outFlag, &restbit);
```

Parameters

Parameters	Description
*DecoderState	The pointer to the state memory is passed.
*inBufferptr	Pointer to the input buffer, where the encoded bit stream is available. Based on the return value from AACDecoder_GetFrameSize(), the application fetches the needed bytes from the bit stream file and fills in this buffer.
*pTimeData	Pointer to the buffer where the output samples are to be stored
*outFlag	The pointer to a variable for output flag: 0 for no valid data (for initial few frames) 1 for valid data present (for all subsequent frames) After decoding the frame, the function indicates whether valid data is present in the output buffer. • restBytes - Pointer to the variable that indicates how many bytes have not decoded. • Normally the number is 0.

Function

int32_t AAC_Decode(void *DecoderState,uint8_t *inBufferptr,short *pTimeData, int16_t *outFlag)

AAC_Initialize Function

File

aac.h

C

```
bool AAC_Initialize(void * heap, uint16_t size, uint8_t * ptr, SYS_FS_HANDLE aacFilehandle);
```

Description

This is function AAC_Initialize.

AAC_RegisterDecoderEventHandlerCallback Function

File

aac.h

C

```
void AAC_RegisterDecoderEventHandlerCallback(SetReadBytesInAppData fptr);
```

Description

This is function AAC_RegisterDecoderEventHandlerCallback.

AAC_GetSamplingFrequency Function

File

aac.h

C

```
int32_t AAC_GetSamplingFrequency(uint8_t * ptr);
```

Description

This is function AAC_GetSamplingFrequency.

b) Data Types and Constants

INTPCM Type

File

decoder_aac.h

C

```
typedef short INTPCM;
```

Section

Middleware & Other Library Configuration

AAC_DECODER_STATES Enumeration

File

aac.h

C

```
typedef enum {
  AAC_GET_FRAME_SIZE,
  AAC_DECODE_FRAME
} AAC_DECODER_STATES;
```

Description

This is type AAC_DECODER_STATES.

WAVE Structure

File

decoder_aac.h

C

```
typedef struct {
  int32_t riff;
  int32_t len;
  int32_t wave;
  int32_t fmt;
  int32_t sublen;
  int16_t mode;
  int16_t num_chnls;
  int32_t samp_freq;
  int32_t bytes_per_sec;
  int16_t bytes_per_sample;
  int16_t bits_per_sample;
  int32_t data;
  int32_t data_len;
} WAVE;
```

Members

Members	Description
int16_t mode;	1 for uncompressed PCM
int16_t num_chnls;	1 for mono and 2 for stereo
int32_t bytes_per_sec;	bytes_per_sample * samp_freq
int16_t bytes_per_sample;	bits_per_sample/8 * num_channels : bytes per set of sample
int16_t bits_per_sample;	bits per word (left or right - not both)

Description

This is type WAVE.

AAC_PROFILE Macro

File

decoder_aac.h

C

```
#define AAC_PROFILE 1
```

Description

This is macro AAC_PROFILE.

AAC_SAMPLING_FREQUENCY_INDEX Enumeration

File

aac.h

C

```
typedef enum {
  SAMPLING_FREQUENCY_IDX0 = 0,
  SAMPLING_FREQUENCY_IDX1,
  SAMPLING_FREQUENCY_IDX2,
  SAMPLING_FREQUENCY_IDX3,
  SAMPLING_FREQUENCY_IDX4,
  SAMPLING_FREQUENCY_IDX5,
  SAMPLING_FREQUENCY_IDX6,
  SAMPLING_FREQUENCY_IDX7,
  SAMPLING_FREQUENCY_IDX8,
  SAMPLING_FREQUENCY_IDX9,
  SAMPLING_FREQUENCY_IDX10,
  SAMPLING_FREQUENCY_IDX11,
  SAMPLING_FREQUENCY_IDX12,
  SAMPLING_FREQUENCY_IDX13,
  SAMPLING_FREQUENCY_IDX14,
  SAMPLING_FREQUENCY_IDX15
} AAC_SAMPLING_FREQUENCY_INDEX;
```

Description

This is type AAC_SAMPLING_FREQUENCY_INDEX.

AAC_ERROR_COUNT_MAX Macro

File

aac.h

C

```
#define AAC_ERROR_COUNT_MAX 1
```

Description

This is macro AAC_ERROR_COUNT_MAX.

AACDECODER_STATE_SIZE Macro

File

decoder_aac.h

C

```
#define AACDECODER_STATE_SIZE 11032                // in Bytes
```

Description

in Bytes

FRAME_SIZE Macro

File

decoder_aac.h

C

```
#define FRAME_SIZE 1024                 // in Samples
```

Description

in Samples

INPUT_BUF_SIZE Macro

File

decoder_aac.h

C

```
#define INPUT_BUF_SIZE (6144*2/8)          // in Bytes
```

Description

in Bytes

MAX_STACK Macro

File

decoder_aac.h

C

```
#define MAX_STACK 0x5000                // in Bytes
```

Description

in Bytes

STACK Macro

File

decoder_aac.h

C

```
#define STACK 1
```

Description

This is macro STACK.

AAC_FRAME_HEADER_SIZE Macro

File

decoder_aac.h

C

```
#define AAC_FRAME_HEADER_SIZE 7                  // in Bytes
```

Description

in Bytes

SetReadBytesInAppData Type

File

aac.h

C

```
typedef void (* SetReadBytesInAppData)(int32_t val);
```

Description

This is type SetReadBytesInAppData.

Files

Files

Name	Description
aac.h	Frame enumerators.
decoder_aac.h	This file consists of the abstract function and input output buffer size declaration for decoding purposes.

Description

This section lists the source and header files used by the MP3 Decoder Library.

aac.h

Frame enumerators.

Enumerations

	Name	Description
	AAC_DECODER_STATES	This is type AAC_DECODER_STATES.
	AAC_SAMPLING_FREQUENCY_INDEX	This is type AAC_SAMPLING_FREQUENCY_INDEX.

Functions

	Name	Description
	AAC_Decoder	This is function AAC_Decoder.
	AAC_GetSamplingFrequency	This is function AAC_GetSamplingFrequency.
	AAC_Initialize	This is function AAC_Initialize.
	AAC_RegisterDecoderEventHandlerCallback	This is function AAC_RegisterDecoderEventHandlerCallback.
	isAACdecoder_enabled	This is function isAACdecoder_enabled.

Macros

	Name	Description
	AAC_ERROR_COUNT_MAX	This is macro AAC_ERROR_COUNT_MAX.

Types

	Name	Description
	SetReadBytesInAppData	This is type SetReadBytesInAppData.

Description

AAC Decoder Library Support File

This header file consists of frame enumerators.

File Name

aac.h

Company

Microchip Technology Inc.

decoder_aac.h

This file consists of the abstract function and input output buffer size declaration for decoding purposes.

Functions

	Name	Description
	AAC_Decode	This function decodes AAC data.
	AAC_WriteWavHeader	Generates the .wav file header, if applicable.
	AACDecoder_GetFrameSize	Used to get the AAC Decoder frame size.
	AACDecoder_GetStateSize	This function allocates memory for each channel being used.

	AACDecoder_Init	Views the received point and clears the structure.
	AACDecoder_InterleaveSamples	This function interleaves left and right channel data.

Macros

	Name	Description
	AAC_FRAME_HEADER_SIZE	in Bytes
	AAC_PROFILE	This is macro AAC_PROFILE.
	AACDECODER_STATE_SIZE	in Bytes
	FRAME_SIZE	in Samples
	INPUT_BUF_SIZE	in Bytes
	MAX_STACK	in Bytes
	STACK	This is macro STACK.

Structures

	Name	Description
	WAVE	This is type WAVE.

Types

	Name	Description
	INTPCM	

Description

AAC Decoder Library Interface File

The header file consists of function declaration for the abstract functions to invoke decoding . The header file also defines the size of input samples and i/p and o/p buffer. This configuration header must not define any prototypes or data definitions (or include any files that do). It only provides macro definitions for build-time configuration options that are not instantiated until used by another MPLAB Harmony module or application.

File Name

decoder_aac.h

Company

Microchip Technology Inc.

MP3 Decoder Library

This section describes the MP3 Decoder Library.

Introduction

The MP3 Decoder Library is available for the PIC32 family of microcontrollers.

Description

The MP3 Decoder Library is a premium product and is not included in the standard release of MPLAB Harmony and must be purchased separately. Refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for additional information.

The Compact MP3 algorithm is designed to fit in small memory footprint PIC32MX Microcontrollers. This code requires only 28 MIPS of performance (CD Quality audio), 42 KB Flash and 11 KB RAM memory for operation on the PIC32MX. This part number is for Non-modifiable binary code. Source code is also available (refer to the previous web link). Users are responsible for patent-only licensing through Technicolor.

Assembly Language Issue

There is an assembly language issue in the MP3 Decoder Library that misaligns the stack pointer. Contrary to the MIPS programming specification, this error is benign except when the application is built for a device with a hardware Floating Point Unit (FPU), such as the PIC32MZ EF family of devices. In this case, the sub-routine context-saving operations will produce an exception and stop the application. A work around exists for this issue, which involves disabling FPU context-saving for Interrupt Service Routines (ISRs).

The work around is to convert all ISRs from:

```
void __ISR(vector,ipl) MyInterruptHandler(void)
```

and change them to:

```
void __attribute__((vector),interrupt(ipl),nomips16,no_fpu)) MyInterruptHandler(void)
```

where, `vector` and `ipl` are replaced with the actual interrupt vector and interrupt priority level.

An example of this work around can be found in the universal_audio_decoders demonstration.

So as long as no FPU operations occur in ISRs and the routines are called by ISRs, this work around will prevent the misaligned stack pointer from causing an exception. If FPU operations result from an ISR with no_fpu enabled, a general exception will result.

Using the Library

This topic describes the basic architecture of the MP3 Decoder Library and provides information and examples on its use.

Description

Interface Header File: `decoder_mp3.h`

The interface to the MP3 Decoder Library is defined in the `decoder_mp3.h` header file. Any C language source (`.c`) file that uses the MP3 Decoder Library should include `decoder_mp3.h`.

Please refer to the What is MPLAB Harmony? section for how the MP3 Decoder Library interacts with the framework.

Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the MP3 Decoder Library module.

Library Interface

a) General Functions

	Name	Description
	MP3_Decode	This is function MP3_Decode.
	MP3Initialize	The function views the received pointer as a structure pointer and clears the entire structure.
	MP3_EventHandler	This is function MP3_EventHandler.
	MP3Decode	This function is called once to decode one frame.
	MP3DecoderGetStateSize	This function allocates memory used for decoding.
	MP3_ParseVBR	This is function MP3_ParseVBR.
	MP3DecoderInit	The function views the received pointer as a structure pointer and clears the entire structure.
	isMP3decoder_enabled	This is function isMP3decoder_enabled.
	ID3_Initialize	void ID3_Initialize (void);
	ID3_EventHandler	This is function ID3_EventHandler.
	ID3_Parse	This is function ID3_Parse.
	ID3_ParseFrameV22	This is function ID3_ParseFrameV22.
	ID3_ParseFrameV23	This is function ID3_ParseFrameV23.
	ID3_Parse_Frame	This is function ID3_Parse_Frame.
	MP3Decode	This function is called once to decode one frame.
	MP3_GetAudioSize	This is function MP3_GetAudioSize.
	MP3_UpdatePlaytime	This is function MP3_UpdatePlaytime.
	MP3_Initialize	This is function MP3_Initialize.
	MP3_RegisterDecoderEventHandlerCallback	This is function MP3_RegisterDecoderEventHandlerCallback.
	MP3_GetChannels	This is function MP3_GetChannels.

b) Data Types and Constants

	Name	Description
	MP3_FORMAT	
	MP3_STATE_SIZE	11024
	MP3_EVENT	This is type MP3_EVENT.
	MP3_FRAME_HEADER	This is type MP3_FRAME_HEADER.
	MP3_STATE	This is type MP3_STATE.
	MP3_XING_HEADER	This is type MP3_XING_HEADER.
	MP3_ERROR_COUNT_MAX	This is macro MP3_ERROR_COUNT_MAX.
	MP3_HEADER_CHANNELS_DUAL	This is macro MP3_HEADER_CHANNELS_DUAL.
	MP3_HEADER_CHANNELS_JOINT	This is macro MP3_HEADER_CHANNELS_JOINT.

	MP3_HEADER_CHANNELS_MONO	This is macro MP3_HEADER_CHANNELS_MONO.
	MP3_HEADER_CHANNELS_STEREO	This is macro MP3_HEADER_CHANNELS_STEREO.
	MP3_HEADER_SAMPLERATE_32000	This is macro MP3_HEADER_SAMPLERATE_32000.
	MP3_HEADER_SAMPLERATE_44100	This is macro MP3_HEADER_SAMPLERATE_44100.
	MP3_HEADER_SAMPLERATE_48000	This is macro MP3_HEADER_SAMPLERATE_48000.
	MP3_HEADER_SAMPLERATE_RESV	This is macro MP3_HEADER_SAMPLERATE_RESV.
	MP3_PLAYTIME_H	This is macro MP3_PLAYTIME_H.
	MP3_XING_HEADER_START_MONO	This is macro MP3_XING_HEADER_START_MONO.
	MP3_XING_HEADER_START_STEREO	This is macro MP3_XING_HEADER_START_STEREO.
	ID3_EVENT	This is type ID3_EVENT.
	ID3_STATE	This is type ID3_STATE.
	ID3V1_EXTENDED_TAG	This is type ID3V1_EXTENDED_TAG.
	ID3V1_TAG	This is type ID3V1_TAG.
	ID3V2_TAG_HEADER	This is type ID3V2_TAG_HEADER.
	ID3V22_FRAME	This is type ID3V22_FRAME.
	ID3V22_FRAME_HEADER	This is type ID3V22_FRAME_HEADER.
	ID3V23_FRAME	This is type ID3V23_FRAME.
	ID3V23_FRAME_HEADER	This is type ID3V23_FRAME_HEADER.
	ID3_H	This is macro ID3_H.
	ID3_STRING_SIZE	This is macro ID3_STRING_SIZE.
	ID3V22_ALBUM	This is macro ID3V22_ALBUM.
	ID3V22_ARTIST	This is macro ID3V22_ARTIST.
	ID3V22_TITLE	This is macro ID3V22_TITLE.
	ID3V22_ZERO	This is macro ID3V22_ZERO.
	ID3V23_ALBUM	This is macro ID3V23_ALBUM.
	ID3V23_ARTIST	This is macro ID3V23_ARTIST.
	ID3V23_TITLE	This is macro ID3V23_TITLE.
	ID3V23_ZERO	This is macro ID3V23_ZERO.
	MP3_CHANNEL_INFO	
	MP3MPEG1L3_SAMPLES_PER_FRAME	MPEG1 LayerIII, samples per frame
	MP3MPEG2L3_SAMPLES_PER_FRAME	MPEG2/2.5 LayerIII, samples per frame
	MP3_DEC	This is type MP3_DEC.
	DecoderEventHandlerCB	This is type DecoderEventHandlerCB.

Description

This section describes the Application Programming Interface (API) functions of the MP3 Decoder Library.

Refer to each section for a detailed description.

a) General Functions

MP3_Decode Function

File

mp3.h

C

```
bool MP3_Decode(uint8_t * input, uint16_t inSize, uint16_t * read, uint8_t * output, uint16_t * written);
```

Description

This is function MP3_Decode.

MP3Initialize Function

The function views the received pointer as a structure pointer and clears the entire structure.

File

decoder_mp3_microaptiv.h

C

```
bool MP3Initialize(void * state);
```

Returns

This function returns a boolean value.

- 0 - indicates unsuccessful initialization
- 1 - indicates successful initialization

Description

The function views the received pointer as a structure pointer and clears the entire structure. Subsequently, it initializes the selective variable with the default values. The function is called only once before initiating the decoding process.

Preconditions

None.

Example

```
MP3Initialize ( mp3.decoderVars );
```

Parameters

Parameters	Description
*state	The pointer to the allocated state memory is passed.

Function

bool MP3Initialize(void *state);

MP3_EventHandler Function

File

mp3.h

C

```
bool MP3_EventHandler(MP3_EVENT event, uint32_t data);
```

Description

This is function MP3_EventHandler.

MP3Decode Function

This function is called once to decode one frame.

File

decoder_mp3_microaptiv.h

C

```
uint16_t MP3Decode(void * state, uint8_t * in, uint16_t inSize, uint16_t * used, MP3_FORMAT * format, void
* outBuffer, uint16_t * outSize);
```

Returns

This function returns either the value '0' or '1' to indicate the status of the decode operation.

- 1 - indicates success
- 0 - indicates failure or end of available encoded data. Upon receiving the value '0' the application aborts processing the channel.

Description

This function is called once to decode one frame. This function will be called in an infinite while loop until the end of the frame is detected or the decode function returns the value '0'. Unlike most speech applications where the encoded data for each frame is deterministic, this is not the case for audio applications. Therefore, the application provides a portion of input data (usually more than what is necessary for a single frame). The decoder parses the input and decodes it with a portion of the data. The decode function passes the information of how many bytes of data were used to the application so that the application saves the remaining data and presents to the decoder for the subsequent frame.

Preconditions

None.

Example

```
MP3Decode ( mp3.decoderVars, input, inSize, read, &mp3.format, output, written )
```

Parameters

Parameters	Description
*state	The pointer to the state memory is passed.
*in	The pointer to the input buffer , where the encoded bit stream is available.
inSize	A variable of data type uint16_t indicating the number of valid bytes available in the input buffer.
*used	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of bytes consumed from the current frame decoder operation.
*format	The pointer to a channel info structure is passed to the function the decider updates this structure after every call
*outBuffer	The pointer to the output sample buffer where the decided pcm samples are to be written,
*outSize	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of valid bytes available in the output buffer.

Function

uint16_t MP3Decode (void *state, uint8_t *in, uint16_t inSize, uint16_t *used,
MP3_FORMAT *format, void *outBuffer, uint16_t *outSize);

MP3DecoderGetStateSize Function

This function allocates memory used for decoding.

File

decoder_mp3_microaptiv.h

C

```
uint32_t MP3DecoderGetStateSize();
```

Returns

This function returns the size of the MP3 Decoder state structure in bytes.

Description

Based on the return value, the application allocates the required memory for the MP3 Decoder state. This function is called only once before initializing the decoder state. If dynamic memory allocation (malloc) is not needed, it is not necessary to call this function and the MP3 Decoder state buffer will be statically defined at the compile time with the size of the MP3 decoder state structure.

Preconditions

None.

Example

```
Word mp3StateSize;
mp3StateSize = MP3DecoderGetStateSize();
```

Function

uint32_t MP3DecoderGetStateSize(void);

MP3_ParseVBR Function

File

mp3.h

C

```
bool MP3_ParseVBR(uint8_t* data);
```

Description

This is function MP3_ParseVBR.

MP3DecoderInit Function

The function views the received pointer as a structure pointer and clears the entire structure.

File

decoder_mp3_advanced.h

C

```
void MP3DecoderInit(void * State);
```

Returns

None.

Description

The function views the received pointer as a structure pointer and clears the entire structure. Subsequently, it initializes the selective variable with the default values. The function is called only once before initiating the decoding process.

Preconditions

None.

Example

```
MP3Initialize ( mp3.decoderVars );
```

Parameters

Parameters	Description
*state	The pointer to the allocated state memory is passed.

Function

void MP3DecoderInit(void *state);

isMP3decoder_enabled Function

File

mp3.h

C

```
bool isMP3decoder_enabled();
```

Description

This is function isMP3decoder_enabled.

ID3_Initialize Function

File

id3.h

C

```
int32_t ID3_Initialize(uint8_t * buffer);
```

Description

void ID3_Initialize (void);

ID3_EventHandler Function

File

id3.h

C

```
bool ID3_EventHandler(ID3_EVENT event, uint32_t data);
```

Description

This is function ID3_EventHandler.

ID3_Parse Function

File

id3.h

C

```
uint32_t ID3_Parse(uint8_t * buff, uint16_t size);
```

Description

This is function ID3_Parse.

ID3_ParseFrameV22 Function

File

id3.h

C

```
void ID3_ParseFrameV22(ID3V22_FRAME * frame, uint32_t frameSize, char * id, uint16_t event);
```

Description

This is function ID3_ParseFrameV22.

ID3_ParseFrameV23 Function

File

id3.h

C

```
void ID3_ParseFrameV23(ID3V23_FRAME * frame, uint32_t frameSize, char * id, uint16_t event);
```

Description

This is function ID3_ParseFrameV23.

ID3_Parse_Frame Function

File

id3.h

C

```
uint32_t ID3_Parse_Frame(uint8_t * buffer, size_t left, int8_t * ret);
```

Description

This is function ID3_Parse_Frame.

MP3Decode Function

This function is called once to decode one frame.

File

decoder_mp3_advanced.h

C

```
int16_t MP3Decode(void * state, uint8_t * In, uint16_t inSize, uint16_t * used, MP3_CHANNEL_INFO * format,
void * outBuffer, uint16_t * outSize);
```

Returns

This function returns either the value '0' or '1' or '2' to indicate the status of the decode operation.

- 0 - indicates success
- 1 - indicates that no data is available (thus marking end of file)
- 2 - indicates failure or invalid or corrupted data

Description

This function is called once to decode one frame. This function will be called in an infinite while loop until the end of the frame is detected or the decode function returns the value '0'. Unlike most speech applications where the encoded data for each frame is deterministic, this is not the case for audio applications. Therefore, the application provides a portion of input data (usually more than what is necessary for a single frame). The decoder parses the input and decodes it with a portion of the data. The decode function passes the information of how many bytes of data were used to the application so that the application saves the remaining data and presents to the decoder for the subsequent frame.

Preconditions

None.

Example

```
MP3Decode ( mp3.decoderVars, input, inSize, read, &mp3.format, output, written )
```

Parameters

Parameters	Description
*state	The pointer to the state memory is passed.
*in	The pointer to the input buffer , where the encoded bit stream is available.
inSize	A variable of data type uint16_t indicating the number of valid bytes available in the input buffer.
*used	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of bytes consumed from the current frame decoder operation.
*format	The pointer to a channel info structure is passed to the function. The decoder updates this structure after every call. Additional member 'MPEG_Type' to indicate the MPEG version is added in this structure.
*outBuffer	The pointer to the output sample buffer where the decided pcm samples are to be written,
*outSize	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of valid bytes available in the output buffer.

Function

int16_t MP3Decode (void *state, uint8_t *In, uint16_t inSize, uint16_t *used,
MP3_CHANNEL_INFO *format, void *outBuffer, uint16_t *outSize);

MP3_GetAudioSize Function

File

mp3.h

C

```
uint32_t MP3_GetAudioSize();
```

Description

This is function MP3_GetAudioSize.

MP3_UpdatePlaytime Function

File

mp3.h

C

```
uint32_t MP3_UpdatePlaytime();
```

Description

This is function MP3_UpdatePlaytime.

MP3_Initialize Function

File

mp3.h

C

```
bool MP3_Initialize(void * heap, uint16_t size, SYS_FS_HANDLE mp3Filehandle);
```

Description

This is function MP3_Initialize.

MP3_RegisterDecoderEventHandlerCallback Function

File

mp3.h

C

```
void MP3_RegisterDecoderEventHandlerCallback(DecoderEventHandlerCB fptr);
```

Description

This is function MP3_RegisterDecoderEventHandlerCallback.

MP3_GetChannels Function

File

mp3.h

C

```
uint8_t MP3_GetChannels();
```

Description

This is function MP3_GetChannels.

b) Data Types and Constants

MP3_FORMAT Structure

File

decoder_mp3_microaptiv.h

C

```
typedef struct {
  uint16_t Kbps;
  uint16_t sampleRate;
  uint8_t bitsPerSample;
  uint8_t channels;
} MP3_FORMAT;
```

Section

Middleware & Other Library Configuration

MP3_STATE_SIZE Macro

File

mp3.h

C

```
#define MP3_STATE_SIZE 16876//11024
```

Description

11024

MP3_EVENT Enumeration

File

mp3.h

C

```
typedef enum {
  MP3_EVENT_TAG_ARTIST,
  MP3_EVENT_TAG_ALBUM,
  MP3_EVENT_TAG_TITLE,
  MP3_EVENT_STREAM_START,
  MP3_EVENT_SAMPLERATE,
  MP3_EVENT_BITRATE,
  MP3_EVENT_TRACK_TIME
} MP3_EVENT;
```

Members

Members	Description
MP3_EVENT_TAG_ARTIST	Align TAG_ARTIST, TAG_ALBUM and TAG_TITLE with ID3 EVENT_TAGS

Description

This is type MP3_EVENT.

MP3_FRAME_HEADER Union

File

mp3.h

C

```
typedef union {
  uint32_t data;
  struct {
    uint8_t sync0 : 8;
    uint8_t CRC : 1;
    uint8_t layer : 2;
    uint8_t mpeg : 2;
    uint8_t sync1 : 3;
    uint8_t padding : 1;
    uint8_t samprate : 2;
    uint8_t bitrate : 4;
    uint8_t nc : 4;
    uint8_t extend : 2;
    uint8_t stereo : 2;
  }
} MP3_FRAME_HEADER;
```

Description

This is type MP3_FRAME_HEADER.

MP3_STATE Enumeration

File

mp3.h

C

```
typedef enum {
  MP3_STATE_STREAM
} MP3_STATE;
```

Description

This is type MP3_STATE.

MP3_XING_HEADER Structure

File

mp3.h

C

```
typedef struct {
  int8_t tag[4];
  union {
    uint32_t flags;
    struct {
      uint32_t frameInfo : 1;
      uint32_t sizeInfo : 1;
      uint32_t tocInfo : 1;
      uint32_t qualityInfo : 1;
    }
  }
  uint8_t frames[4];
  uint8_t size[4];
  uint8_t toc[100];
  uint8_t quality;
} MP3_XING_HEADER;
```

Description

This is type MP3_XING_HEADER.

MP3_ERROR_COUNT_MAX Macro

File

mp3.h

C

```
#define MP3_ERROR_COUNT_MAX 1
```

Description

This is macro MP3_ERROR_COUNT_MAX.

MP3_HEADER_CHANNELS_DUAL Macro

File

mp3.h

C

```
#define MP3_HEADER_CHANNELS_DUAL 0b10
```

Description

This is macro MP3_HEADER_CHANNELS_DUAL.

MP3_HEADER_CHANNELS_JOINT Macro

File

mp3.h

C

```
#define MP3_HEADER_CHANNELS_JOINT 0b01
```

Description

This is macro MP3_HEADER_CHANNELS_JOINT.

MP3_HEADER_CHANNELS_MONO Macro

File

mp3.h

C

```
#define MP3_HEADER_CHANNELS_MONO 0b11
```

Description

This is macro MP3_HEADER_CHANNELS_MONO.

MP3_HEADER_CHANNELS_STEREO Macro

File

mp3.h

C

```
#define MP3_HEADER_CHANNELS_STEREO 0b00
```

Description

This is macro MP3_HEADER_CHANNELS_STEREO.

MP3_HEADER_SAMPLERATE_32000 Macro

File

mp3.h

C

```
#define MP3_HEADER_SAMPLERATE_32000 0b10
```

Description

This is macro MP3_HEADER_SAMPLERATE_32000.

MP3_HEADER_SAMPLERATE_44100 Macro

File

mp3.h

C

```
#define MP3_HEADER_SAMPLERATE_44100 0b00
```

Description

This is macro MP3_HEADER_SAMPLERATE_44100.

MP3_HEADER_SAMPLERATE_48000 Macro

File

mp3.h

C

```
#define MP3_HEADER_SAMPLERATE_48000 0b01
```

Description

This is macro MP3_HEADER_SAMPLERATE_48000.

MP3_HEADER_SAMPLERATE_RESV Macro

File

mp3.h

C

```
#define MP3_HEADER_SAMPLERATE_RESV 0b11
```

Description

This is macro MP3_HEADER_SAMPLERATE_RESV.

MP3_PLAYTIME_H Macro

File

mp3.h

C

```
#define MP3_PLAYTIME_H
```

Description

This is macro MP3_PLAYTIME_H.

MP3_XING_HEADER_START_MONO Macro

File

mp3.h

C

```
#define MP3_XING_HEADER_START_MONO 17
```

Description

This is macro MP3_XING_HEADER_START_MONO.

MP3_XING_HEADER_START_STEREO Macro

File

mp3.h

C

```
#define MP3_XING_HEADER_START_STEREO 32
```

Description

This is macro MP3_XING_HEADER_START_STEREO.

ID3_EVENT Enumeration

File

id3.h

C

```
typedef enum {
  ID3_EVENT_TAG_ARTIST,
  ID3_EVENT_TAG_ALBUM,
  ID3_EVENT_TAG_TITLE
} ID3_EVENT;
```

Description

This is type ID3_EVENT.

ID3_STATE Enumeration

File

id3.h

C

```
typedef enum {
  ID3_STATE_INIT,
  ID3_STATE_READ_V1,
  ID3_STATE_READ_V2_HEADER,
  ID3_STATE_READ_V2_FRAME,
  ID3_STATE_FINISHED
} ID3_STATE;
```

Description

This is type ID3_STATE.

ID3V1_EXTENDED_TAG Structure

File

id3.h

C

```
typedef struct {
  uint8_t tag[4];
  int8_t title[60];
  int8_t artist[60];
  int8_t album[60];
  uint8_t speed;
  int8_t genre[30];
  int8_t startTime[6];
  int8_t endTime[6];
} ID3V1_EXTENDED_TAG;
```

Description

This is type ID3V1_EXTENDED_TAG.

ID3V1_TAG Structure

File

id3.h

C

```
typedef struct {
  uint8_t tag[3];
  int8_t title[30];
  int8_t artist[30];
```

```
    int8_t album[30];
    int8_t year[4];
    union {
      int8_t comment[30];
      struct {
        int8_t commentShort[28];
        uint8_t zero;
        uint8_t track;
      }
    }
    uint8_t genre;
  } ID3V1_TAG;
```

Description

This is type ID3V1_TAG.

ID3V2_TAG_HEADER Structure

File

id3.h

C

```
typedef struct {
  uint8_t tag[3];
  uint8_t version;
  uint8_t empty;
  uint8_t flag;
  uint8_t size[4];
} ID3V2_TAG_HEADER;
```

Description

This is type ID3V2_TAG_HEADER.

ID3V22_FRAME Structure

File

id3.h

C

```
typedef struct {
  uint8_t id[3];
  uint8_t size[3];
  uint8_t encoding;
  uint8_t contents[1];
} ID3V22_FRAME;
```

Description

This is type ID3V22_FRAME.

ID3V22_FRAME_HEADER Structure

File

id3.h

C

```
typedef struct {
  uint8_t id[3];
  uint8_t size[3];
} ID3V22_FRAME_HEADER;
```

Description

This is type ID3V22_FRAME_HEADER.

ID3V23_FRAME Structure

File

id3.h

C

```
typedef struct {
  uint8_t id[4];
  uint8_t size[4];
  uint8_t flags[2];
  uint8_t encoding;
  uint8_t contents[1];
} ID3V23_FRAME;
```

Description

This is type ID3V23_FRAME.

ID3V23_FRAME_HEADER Structure

File

id3.h

C

```
typedef struct {
  uint8_t id[4];
  uint8_t size[4];
  uint8_t flags[2];
} ID3V23_FRAME_HEADER;
```

Description

This is type ID3V23_FRAME_HEADER.

ID3_H Macro

File

id3.h

C

```
#define ID3_H
```

Description

This is macro ID3_H.

ID3_STRING_SIZE Macro

File

id3.h

C

```
#define ID3_STRING_SIZE 128
```

Description

This is macro ID3_STRING_SIZE.

ID3V22_ALBUM Macro

File

id3.h

C

```
#define ID3V22_ALBUM "TAL"
```

Description

This is macro ID3V22_ALBUM.

ID3V22_ARTIST Macro

File

id3.h

C

```
#define ID3V22_ARTIST "TP1"
```

Description

This is macro ID3V22_ARTIST.

ID3V22_TITLE Macro

File

id3.h

C

```
#define ID3V22_TITLE "TT2"
```

Description

This is macro ID3V22_TITLE.

ID3V22_ZERO Macro

File

id3.h

C

```
#define ID3V22_ZERO "\0\0\0"
```

Description

This is macro ID3V22_ZERO.

ID3V23_ALBUM Macro

File

id3.h

C

```
#define ID3V23_ALBUM "TALB"
```

Description

This is macro ID3V23_ALBUM.

ID3V23_ARTIST Macro

File

id3.h

C

```
#define ID3V23_ARTIST "TPE1"
```

Description

This is macro ID3V23_ARTIST.

ID3V23_TITLE Macro

File

id3.h

C

```
#define ID3V23_TITLE "TIT2"
```

Description

This is macro ID3V23_TITLE.

ID3V23_ZERO Macro

File

id3.h

C

```
#define ID3V23_ZERO "\0\0\0\0"
```

Description

This is macro ID3V23_ZERO.

MP3_CHANNEL_INFO Structure

File

decoder_mp3_advanced.h

C

```
typedef struct {
  int16_t bitsPerWord;
  int16_t channels;
  int32_t samplingRate;
  int32_t bitRate;
  int32_t MPEG_Type;
} MP3_CHANNEL_INFO;
```

Section

Middleware & Other Library Configuration

MP3MPEG1L3_SAMPLES_PER_FRAME Macro

File

mp3.h

C

```
#define MP3MPEG1L3_SAMPLES_PER_FRAME 1152 // MPEG1 LayerIII, samples per frame
```

Description

MPEG1 LayerIII, samples per frame

MP3MPEG2L3_SAMPLES_PER_FRAME Macro

File

mp3.h

C

```
#define MP3MPEG2L3_SAMPLES_PER_FRAME 576  // MPEG2/2.5 LayerIII, samples per frame
```

Description

MPEG2/2.5 LayerIII, samples per frame

MP3_DEC Structure

File

mp3.h

C

```
typedef struct {
  uint16_t mp3SampleRate;
  uint32_t mp3BitRate;
  uint32_t mp3Duration;
  uint32_t firstFramePos;
  bool isVBR;
  uint32_t mp3ValidBytes;
  uint8_t stereo;
} MP3_DEC;
```

Members

Members	Description
uint32_t firstFramePos;	first frame position in file
uint32_t mp3ValidBytes;	for VBR MP3, mp3ValidBytes is the number of bytes in file is given by XING header, for CBR MP3, mp3ValidBytes is MP3 audio file size - first frame position

Description

This is type MP3_DEC.

DecoderEventHandlerCB Type

File

mp3.h

C

```
typedef bool (* DecoderEventHandlerCB)(uint32_t event, uint32_t data);
```

Description

This is type DecoderEventHandlerCB.

Files

Files

Name	Description
mp3.h	MP3 Decoder support API.
id3.h	MP3 Decoder ID3 parser header API.
decoder_mp3_advanced.h	This file consists of the abstract function and input output buffer size declaration for decoding purpose
decoder_mp3_microaptiv.h	This file consists of the abstract function and input output buffer size declaration for decoding purpose

Description

This section lists the source and header files used by the MP3 Decoder Library.

mp3.h

MP3 Decoder support API.

Enumerations

	Name	Description
	MP3_EVENT	This is type MP3_EVENT.
	MP3_STATE	This is type MP3_STATE.

Functions

	Name	Description
	isMP3decoder_enabled	This is function isMP3decoder_enabled.
	MP3_Decode	This is function MP3_Decode.
	MP3_EventHandler	This is function MP3_EventHandler.
	MP3_GetAudioSize	This is function MP3_GetAudioSize.
	MP3_GetChannels	This is function MP3_GetChannels.
	MP3_Initialize	This is function MP3_Initialize.
	MP3_ParseVBR	This is function MP3_ParseVBR.
	MP3_RegisterDecoderEventHandlerCallback	This is function MP3_RegisterDecoderEventHandlerCallback.
	MP3_UpdatePlaytime	This is function MP3_UpdatePlaytime.

Macros

	Name	Description
	MP3_ERROR_COUNT_MAX	This is macro MP3_ERROR_COUNT_MAX.
	MP3_HEADER_CHANNELS_DUAL	This is macro MP3_HEADER_CHANNELS_DUAL.
	MP3_HEADER_CHANNELS_JOINT	This is macro MP3_HEADER_CHANNELS_JOINT.
	MP3_HEADER_CHANNELS_MONO	This is macro MP3_HEADER_CHANNELS_MONO.
	MP3_HEADER_CHANNELS_STEREO	This is macro MP3_HEADER_CHANNELS_STEREO.
	MP3_HEADER_SAMPLERATE_32000	This is macro MP3_HEADER_SAMPLERATE_32000.
	MP3_HEADER_SAMPLERATE_44100	This is macro MP3_HEADER_SAMPLERATE_44100.
	MP3_HEADER_SAMPLERATE_48000	This is macro MP3_HEADER_SAMPLERATE_48000.
	MP3_HEADER_SAMPLERATE_RESV	This is macro MP3_HEADER_SAMPLERATE_RESV.
	MP3_PLAYTIME_H	This is macro MP3_PLAYTIME_H.
	MP3_STATE_SIZE	11024
	MP3_XING_HEADER_START_MONO	This is macro MP3_XING_HEADER_START_MONO.
	MP3_XING_HEADER_START_STEREO	This is macro MP3_XING_HEADER_START_STEREO.
	MP3MPEG1L3_SAMPLES_PER_FRAME	MPEG1 LayerIII, samples per frame
	MP3MPEG2L3_SAMPLES_PER_FRAME	MPEG2/2.5 LayerIII, samples per frame

Structures

	Name	Description
	MP3_DEC	This is type MP3_DEC.
	MP3_XING_HEADER	This is type MP3_XING_HEADER.

Types

	Name	Description
	DecoderEventHandlerCB	This is type DecoderEventHandlerCB.

Unions

	Name	Description
	MP3_FRAME_HEADER	This is type MP3_FRAME_HEADER.

Description

MP3 Decoder Library Interface File

This header file consists of support function declarations.

File Name

mp3.h

Company

Microchip Technology Inc.

id3.h

MP3 Decoder ID3 parser header API.

Enumerations

	Name	Description
	ID3_EVENT	This is type ID3_EVENT.
	ID3_STATE	This is type ID3_STATE.

Functions

	Name	Description
	ID3_EventHandler	This is function ID3_EventHandler.
	ID3_Initialize	void ID3_Initialize (void);
	ID3_Parse	This is function ID3_Parse.
	ID3_Parse_Frame	This is function ID3_Parse_Frame.
	ID3_ParseFrameV22	This is function ID3_ParseFrameV22.
	ID3_ParseFrameV23	This is function ID3_ParseFrameV23.

Macros

	Name	Description
	ID3_H	This is macro ID3_H.
	ID3_STRING_SIZE	This is macro ID3_STRING_SIZE.
	ID3V22_ALBUM	This is macro ID3V22_ALBUM.
	ID3V22_ARTIST	This is macro ID3V22_ARTIST.
	ID3V22_TITLE	This is macro ID3V22_TITLE.
	ID3V22_ZERO	This is macro ID3V22_ZERO.
	ID3V23_ALBUM	This is macro ID3V23_ALBUM.
	ID3V23_ARTIST	This is macro ID3V23_ARTIST.
	ID3V23_TITLE	This is macro ID3V23_TITLE.
	ID3V23_ZERO	This is macro ID3V23_ZERO.

Structures

	Name	Description
	ID3V1_EXTENDED_TAG	This is type ID3V1_EXTENDED_TAG.
	ID3V1_TAG	This is type ID3V1_TAG.
	ID3V2_TAG_HEADER	This is type ID3V2_TAG_HEADER.
	ID3V22_FRAME	This is type ID3V22_FRAME.
	ID3V22_FRAME_HEADER	This is type ID3V22_FRAME_HEADER.
	ID3V23_FRAME	This is type ID3V23_FRAME.
	ID3V23_FRAME_HEADER	This is type ID3V23_FRAME_HEADER.

Description

MP3 Decoder Timekeeping and Delay Routines Definitions Header File

This file provides MP3 Decoder ID3 parser header APIs.

File Name

id3.h

Company

Microchip Technology Inc.

decoder_mp3_advanced.h

This file consists of the abstract function and input output buffer size declaration for decoding purpose

Functions

	Name	Description
	MP3Decode	This function is called once to decode one frame.
	MP3DecoderInit	The function views the received pointer as a structure pointer and clears the entire structure.

Structures

	Name	Description
	MP3_CHANNEL_INFO	

Description

MP3 Decoder Library Interface File

The header file consists of function declaration for the abstract functions to invoke decoding . The header file also defines the size of input samples and i/p and o/p buffer.

File Name

decoder_mp3_advanced.h

Company

Microchip Technology Inc.

decoder_mp3_microaptiv.h

This file consists of the abstract function and input output buffer size declaration for decoding purpose

Functions

	Name	Description
	MP3Decode	This function is called once to decode one frame.
	MP3DecoderGetStateSize	This function allocates memory used for decoding.
	MP3Initialize	The function views the received pointer as a structure pointer and clears the entire structure.

Structures

	Name	Description
	MP3_FORMAT	

Description

MP3 Decoder Library Interface File

The header file consists of function declaration for the abstract functions to invoke decoding . The header file also defines the size of input samples and i/p and o/p buffer.

File Name

decoder_mp3_microaptiv.h

Company

Microchip Technology Inc.

Opus Decoder Library

This section describes the Opus Decoder Library.

Introduction

Introduces the Opus Decoder Library.

Description

Opus is an open, royalty-free, highly versatile audio codec. Opus is unmatched for interactive speech and music transmission over the Internet, but is also intended for storage and streaming applications. It is standardized by the Internet Engineering Task Force (IETF) as RFC 6716, which incorporated technology from Skype's SILK codec and Xiph.Org's CELT codec.

Opus Features

Supported features are:

- Bit rates from 6 kb/s to 510 kb/s
- Sampling rates from 8 kHz (narrowband) to 48 kHz (fullband)
- Frame sizes from 2.5 ms to 60 ms
- Support for both constant bit rate (CBR) and variable bit rate (VBR)
- Audio bandwidth from narrowband to fullband
- Support for speech and music
- Support for mono and stereo
- Support for up to 255 channels (mult-istream frames)
- Dynamically adjustable bit rate, audio bandwidth, and frame size
- Good loss robustness and packet loss concealment (PLC)
- Floating point and fixed-point implementation

The full specification, RFC 6716, including the reference implementation is available from http://www.opus-codec.org. An up-to-date implementation of the Opus standard is available from the Opus Codec downloads page by visiting: https://www.opus-codec.org/downloads/

Typical Applications

- Building and home safety systems; Intercoms
- Smart appliances
- Walkie-talkies
- Toys
- Robots
- Any application using message playback

Resources

Feature	Option	Usage
Opus Library Flash Size	Release Build	< 143 KB
Opus Decoder RAM	Heap Size (Mono mode) Heap Size (Stereo mode)	17 KB 25 KB
Opus Decoding Performance (measured using a PIC32MX270F512L device)	16 kbps Voice 12 kbps Voice	6.8 MCPS/13.6 MIPS 4.8 MCPS/9.6 MIPS

Using the Library

This topic describes the basic architecture of the Opus Decoder Library and provides information and examples on its use.

Description

Interface Header File: `opus.h`

The interface to the Opus Decoder Library is defined in the `opus.h` header file. Any C language source (`.c`) file that uses the Opus Decoder Library should include `opus.h`.

Please refer to the What is MPLAB Harmony? section for how the Opus Decoder Library interacts with the framework.

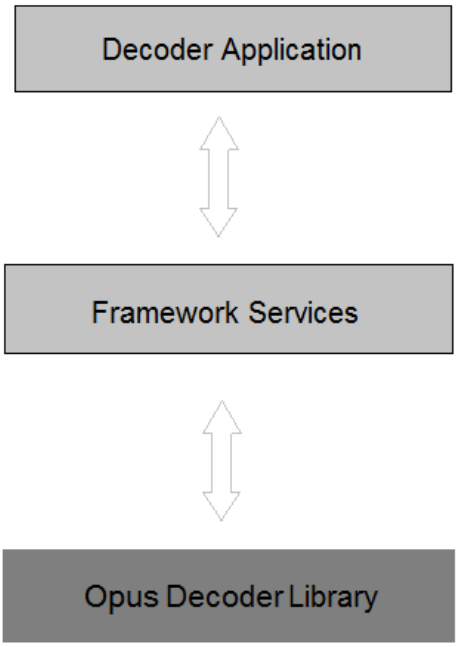
Abstraction Model

Describes the abstraction model for the Opus Decoder Library.

Description

This Opus Library is an Open Source/Free Software patent-free audio compression format designed for interactive speech and music transmission over the Internet. The MPLAB Harmony framework provides an abstraction layer to easily interact with this library. Refer to the universal_audio_decoders demonstration for reference.

Abstraction Model Diagram



Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the Opus Decoder Library module.

How the Library Works

This section describes how to work with the MPLAB Harmony Opus abstraction layer interface, which provides simplifier APIs for using the Opus Decoder Library.

Description

For complete documentation and how to implement Opus, visit https://www.opus-codec.org.

Code Example

The following code provides an example for initializing the Opus Decoder and decoding a packet.

```
void DECODER_Initialize(decoder_type){
  switch(decoder_type){
    case OPUS:
      APP_ERROR_MSG ret = OPUS_Initialize(appDataPtr->fileHandle);
      if(ret == APP_SUCCESS)
      {
        // setup other components,
        // for example, audio CODEC, display, etc...
      }
  }
}

void App_Task()
{
  while(1){
    while(audio is not end){
      // Read one Opus packet
      appData.nBytesRead = OPUS_DiskRead(input_ptr);
     // Decode one packet
     ret = OPUS_Decoder (input_ptr,inSize,read,output,written,outBufSize);
     if(ret == success)
     {
       // continue decoding
     }else{
      // handle decoding errors;
```

```
        }
    }
      // Clean up
      OPUS_Cleanup();
      break;
    }
  }
```

Library Interface

a) General Functions

	Name	Description
	OPUS_ARG_NONNULL	This is function OPUS_ARG_NONNULL.
	OPUS_ARG_NONNULL	This is function OPUS_ARG_NONNULL.
	opus_decoder_create	Allocates and initializes an decoder state.
	opus_decoder_destroy	Frees an OpusDecoder allocated by opus_decoder_create.
	opus_decoder_get_size	Gets the size of an OpusDecoder structure.
	opus_encoder_create	Allocates and initializes an encoder state.
	opus_encoder_destroy	Frees an OpusEncoder allocated by opus_encoder_create.
	opus_encoder_get_size	Gets the size of an OpusEncoder structure.
	isOPUSdecoder_enabled	Checks if Opus decoder is enabled by MHC configuration.
	OPUS_Cleanup	Frees the memory allocated by the Opus Decoder.
	OPUS_Decoder	Called once to decode one Opus packet.
	OPUS_DiskRead	Called once to read one Opus packet.
	OPUS_GetChannels	Returns the number of channels for this Opus audio.
	OPUS_GetSamplingRate	Returns the sampling rate of Opus audio.
	OPUS_Initialize	Initializes the Opus decoder and creates an Opus decoder state structure.

b) Data Types and Constants

	Name	Description
	OpusDecoder	Opus decoder state.
	OpusEncoder	This contains the complete state of an Opus encoder. It is position independent and can be freely copied. See opus_encoder_create,opus_encoder_init.
	OPUS_H	Opus reference implementation API
	OPUS_ERROR_MSG	Ogg Container Structures
	opusDecDcpt	This is type opusDecDcpt.
	sOggPageHdr	header of an Ogg page, full segment info included
	sOpusHeader	This is type sOpusHeader.
	sOpusPktDcpt	decoder data packet descriptor
	sOpusStreamDcpt	info needed by the stream at run-time
	DECODER_OPUS_SUPPORT_H	This is macro DECODER_OPUS_SUPPORT_H.
	OPUS_INPUT_BUFFER_SIZE	MACROS
	OPUS_MAX_FRAME_SIZE	120ms @ 48Khz
	OPUS_OUTPUT_BUFFER_SIZE	This is macro OPUS_OUTPUT_BUFFER_SIZE.

Description

This section describes the Application Programming Interface (API) functions of the Opus Decoder Library.

Refer to each section for a detailed description.

a) General Functions

OPUS_ARG_NONNULL Function

File

opus.h

C

```
OPUS_ARG_NONNULL(1);
```

Description

This is function OPUS_ARG_NONNULL.

OPUS_ARG_NONNULL Function

File

opus.h

C

```
OPUS_ARG_NONNULL(4);
```

Description

This is function OPUS_ARG_NONNULL.

opus_decoder_create Function

Allocates and initializes an decoder state.

File

opus.h

C

```
OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusDecoder * opus_decoder_create(opus_int32 Fs, int channels, int *
error);
```

Returns

- OpusEncoder - Opus encoder state structure.

Description

This function allocates and initializes an decoder state.

Remarks

Internally, Opus stores data at 48000 Hz, so that should be the default value for Fs. However, the decoder can efficiently decode to buffers at 8, 12, 16, and 24 kHz so if for some reason the caller cannot use data at the full sample rate, or knows the compressed data does not use the full frequency range, it can request decoding at a reduced rate. Likewise, the decoder is capable of filling in either mono or interleaved stereo pcm buffers, at the caller's request.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
Fs	Sampling rate of input signal (Hz). This must be one of 8000, 12000, 16000, 24000, or 48000.
channels	Number of channels (1 or 2) in input signal.
error	(out) OPUS_OK Success or opus_errorcodes.

Function

OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusDecoder *opus_decoder_create(opus_int32 Fs,
int channels, int *error)

opus_decoder_destroy Function

Frees an OpusDecoder allocated by opus_decoder_create.

File

opus.h

C

```
OPUS_EXPORT void opus_decoder_destroy(OpusDecoder * st);
```

Returns

None.

Description

This function frees an OpusDecoder allocated by opus_decoder_create.

Remarks

None.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
st	State to be freed.

Function

OPUS_EXPORT void opus_decoder_destroy(OpusDecoder *st)

opus_decoder_get_size Function

Gets the size of an OpusDecoder structure.

File

opus.h

C

```
OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_decoder_get_size(int channels);
```

Returns

- int - The size in bytes.

Description

This function gets the size of an OpusDecoder structure.

Remarks

None.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
channels	Number of channels. This value must be 1 or 2.

Function

OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_decoder_get_size(int channels)

opus_encoder_create Function

Allocates and initializes an encoder state.

File

opus.h

C

```
OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusEncoder * opus_encoder_create(opus_int32 Fs, int channels, int application, int * error);
```

Returns

- OpusEncoder - Opus encoder state structure.

Description

This function allocates and initializes an encoder state. There are three coding modes:

OPUS_APPLICATION_VOIP provides the best quality at a given bit rate for voice signals. It enhances the input signal by high-pass filtering and emphasizing formants and harmonics. Optionally, t includes in-band forward error correction to protect against packet loss. Use this mode for typical VoIP applications. Because of the enhancement, even at high bit rates the output may sound different from the input.

OPUS_APPLICATION_AUDIO provides the best quality at a given bit rate for most non-voice signals like music. Use this mode for music and mixed (music/voice) content, broadcast, and applications requiring less than 15 ms of coding delay.

OPUS_APPLICATION_RESTRICTED_LOWDELAY configures a low-delay mode that disables the speech-optimized mode in exchange for slightly reduced delay. This mode can only be set on a newly initialized or freshly reset encoder because it changes the codec delay.

Remarks

Regardless of the sampling rate and the number channels selected, the Opus encoder can switch to a lower audio bandwidth or number of channels if the bit rate selected is too low. This also means that it is safe to always use 48 kHz stereo input and let the encoder optimize the encoding.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
Fs	Sampling rate of input signal (Hz). This value must be one of 8000, 12000, 16000, 24000, or 48000.
channels	Number of channels (1 or 2) in input signal.
application	Coding mode (OPUS_APPLICATION_VOIP/ OPUS_APPLICATION_AUDIO/ OPUS_APPLICATION_RESTRICTED_LOWDELAY).
error	opus_errorcodes of this function.

Function

OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusEncoder *opus_encoder_create(opus_int32 Fs,

int channels, int application, int *error)

opus_encoder_destroy Function

Frees an OpusEncoder allocated by opus_encoder_create.

File

opus.h

C

```
OPUS_EXPORT void opus_encoder_destroy(OpusEncoder * st);
```

Returns

None.

Description

This function frees an OpusEncoder allocated by opus_encoder_create.

Remarks

None.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
st	State to be freed.

Function

OPUS_EXPORT void opus_encoder_destroy(OpusEncoder *st)

opus_encoder_get_size Function

Gets the size of an OpusEncoder structure.

File

opus.h

C

```
OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_encoder_get_size(int channels);
```

Returns

- int - The size in bytes.

Description

This function gets the size of an OpusEncoder structure.

Remarks

None.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
channels	Number of channels. This value must be 1 or 2.

Function

OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_encoder_get_size(int channels)

isOPUSdecoder_enabled Function

Checks if Opus decoder is enabled by MHC configuration.

File

opus_support.h

C

```
bool isOPUSdecoder_enabled();
```

Returns

None.

Description

The function checks if Opus decoder is enabled by MHC configuration.

Preconditions

None.

Example

```
bool opusEnabled = isOPUSdecoder_enabled ();
```

Function

bool isOPUSdecoder_enabled()

OPUS_Cleanup Function

Frees the memory allocated by the Opus Decoder.

File

opus_support.h

C

```
void OPUS_Cleanup();
```

Returns

None.

Description

This function frees the memory that was allocated by the Opus Decoder and is called when the Opus Decoder ends.

Preconditions

None.

Example

```
OPUS_Cleanup();
```

Function

void OPUS_Cleanup()

OPUS_Decoder Function

Called once to decode one Opus packet.

File

opus_support.h

C

```
OPUS_ERROR_MSG OPUS_Decoder(const uint8_t * input, uint16_t inSize, uint16_t * read, int16_t * output,
uint16_t * written, uint16_t outSize);
```

Returns

The status code of this function.

Description

This function is called once to decode one packet, this function will be called in an infinite while loop until the end of the packet is detected or the decode function returns failure value.

Preconditions

None.

Example

```
OPUS_ERROR_MSG ret = OPUS_Decoder(input, inSize, read, output, written, outSize);
```

Parameters

Parameters	Description
*input	The pointer to the input buffer, where the encoded bit stream is available
inSize	A variable of data type uint16_t indicating the number of valid bytes available in the input buffer
*read	The pointer to a variable is passed to the function. After decoding the packet the function writes the number of bytes consumed from the current frame decoder operation
*output	The pointer to the output sample buffer where the decided PCM samples are to be written
*written	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of valid bytes available in the output buffer
outSize	A variable that indicates the size of output buffer, used for avoiding buffer overwritten.

Function

OPUS_ERROR_MSG OPUS_Decoder(const uint8_t *input, uint16_t inSize, uint16_t *read,
int16_t *output, uint16_t *written, uint16_t outSize)

OPUS_DiskRead Function

Called once to read one Opus packet.

File

opus_support.h

C

```
int32_t OPUS_DiskRead(uint8_t * inBuff);
```

Returns

This function returns the size of this Opus packet.

Description

This function is called once to read one Opus packet.

Preconditions

None.

Example

```
int32_t bytesRead = OPUS_DiskRead(inputBuffer);
```

Parameters

Parameters	Description
*inBuff	The pointer to the input buffer, where the Opus encoded data are to be written.

Function

int32_t OPUS_DiskRead(uint8_t *inBuff)

OPUS_GetChannels Function

Returns the number of channels for this Opus audio.

File

opus_support.h

C

```
uint8_t OPUS_GetChannels();
```

Returns

- -1 - 1 channel, mono mode
- -2 - 2 channels, stereo mode

Description

This function returns the number of channels for this Opus audio.

Preconditions

None.

Example

```
uint8_t channelNumber = OPUS_GetChannels();
```

Function

uint8_t OPUS_GetChannels()

OPUS_GetSamplingRate Function

Returns the sampling rate of Opus audio.

File

opus_support.h

C

```
int32_t OPUS_GetSamplingRate();
```

Returns

The sampling rate of Opus audio.

Description

This function returns the sampling rate of this Opus file, return value is always 48000, Opus playback sample rate should be 48000 as described in the OggOpus spec.

Preconditions

None.

Example

```
int32_t sampleRate = OPUS_GetSamplingRate();
```

Function

int32_t OPUS_GetSamplingRate()

OPUS_Initialize Function

Initializes the Opus decoder and creates an Opus decoder state structure.

File

opus_support.h

C

```
OPUS_ERROR_MSG OPUS_Initialize(const SYS_FS_HANDLE opus_file_handler);
```

Returns

The status code of this function.

Description

This function initializes the Opus decoder and creates an Opus decoder state structure.

Preconditions

None.

Example

```
OPUS_ERROR_MSG ret = OPUS_Initialize(fileHandler);
```

Parameters

Parameters	Description
opus_file_handler	The file handler of current Opus file which will be decoded.

Function

OPUS_ERROR_MSG OPUS_Initialize(const SYS_FS_HANDLE opus_file_handler)

b) Data Types and Constants

OpusDecoder Type

Opus decoder state.

File

opus.h

C

```
typedef struct OpusDecoder OpusDecoder;
```

Description

Opus decoder state.

This structure contains the complete state of an Opus decoder. It is position independent and can be freely copied. See opus_decoder_create,opus_decoder_init.

Remarks

None.

OpusEncoder Type

This contains the complete state of an Opus encoder. It is position independent and can be freely copied. See opus_encoder_create,opus_encoder_init.

File

opus.h

C

```
typedef struct OpusEncoder OpusEncoder;
```

Description

Opus encoder state

This contains the complete state of an Opus encoder. It is position independent and can be freely copied. See opus_encoder_create,opus_encoder_init.

Remarks

None.

OPUS_H Macro

Opus reference implementation API

File

opus.h

C

```
#define OPUS_H
```

File Name

opus.h

OPUS_ERROR_MSG Enumeration

File

opus_support.h

C

```
typedef enum {
  OPUS_SUCCESS = 1,
  OPUS_READ_ERROR,
  OPUS_STREAM_ERROR,
  OPUS_BUFF_ERROR,
  OPUS_STREAM_END,
  OPUS_PLAYBACK_ERROR,
  OPUS_OUT_OF_MEM_ERROR,
  OPUS_DISK_ERROR,
  OPUS_GENERAL_ERROR
} OPUS_ERROR_MSG;
```

Description

Ogg Container Structures

opusDecDcpt Structure

File

opus_support.h

C

```
typedef struct {
  int processedPktNo;
  int currPktNo;
  int nInBytes;
} opusDecDcpt;
```

Members

Members	Description
int processedPktNo;	counter of processed packets
int currPktNo;	number of the currently received packet from the stream
int nInBytes;	bytes available in the input buffer

Description

This is type opusDecDcpt.

sOggPageHdr Structure

File

opus_support.h

C

```
typedef struct {
  int32_t pageCapture;
  int8_t struct_ver;
  int8_t headerFlags;
  int64_t granulePos;
  int32_t streamNo;
  int32_t pageNo;
  int32_t pageCrc;
  uint8_t pageSegments;
  uint8_t segmentTbl[255];
} sOggPageHdr;
```

Members

Members	Description
int32_t pageCapture;	should be OGG_ID_MAGIC
int8_t struct_ver;	version of the Ogg file format. Should be 0 (RFC3533)
int8_t headerFlags;	an eOggHeaderFlags value
int64_t granulePos;	stream dependent position info
int32_t streamNo;	logical bit stream identifier
int32_t pageNo;	page sequence number
int32_t pageCrc;	CRC32 checksum of the page
uint8_t pageSegments;	number of page segments to follow
uint8_t segmentTbl[255];	actually segmentTbl[pageSegments]; contains the lace values for all segments in the page

Description

header of an Ogg page, full segment info included

sOpusHeader Structure

File

opus_support.h

C

```
typedef struct {
  char signature[8];
  uint8_t version;
  uint8_t channels;
  uint16_t preskip;
  uint32_t input_sample_rate;
  uint16_t gain;
  uint8_t channel_mapping;
  int8_t nb_streams;
  int8_t nb_coupled;
  unsigned char stream_map[255];
} sOpusHeader;
```

Members

Members	Description
char signature[8];	Magic signature: "OpusHead
uint8_t version;	Version number: 0x01 for this spec
uint8_t channels;	Number of channels: 1..255
uint16_t preskip;	This is the number of samples (at 48kHz) to discard from the decoder output when starting playback
uint32_t input_sample_rate;	Original input sample rate in Hz, this is not the sample rate to use for playback of the encoded data
uint16_t gain;	output gain in dB
uint8_t channel_mapping;	This byte indicates the order and semantic meaning of various channels encoded in each Opus packet The rest is only used if channel_mapping != 0
int8_t nb_streams;	This field indicates the total number of streams so the decoder can correctly parse the packed Opus packets inside the Ogg packet
int8_t nb_coupled;	Describes the number of streams whose decoders should be configured to produce two channels

Description

This is type sOpusHeader.

sOpusPktDcpt Structure

File

opus_support.h

C

```
typedef struct {
  int pktBytes;
  int pktSeqNo;
} sOpusPktDcpt;
```

Members

Members	Description
int pktBytes;	how many bytes in this packet
int pktSeqNo;	packet sequence number

Description

decoder data packet descriptor

sOpusStreamDcpt Structure

File

opus_support.h

C

```
typedef struct {
  sOggPageHdr pageHdr;
  int segIx;
  int pktIx;
  int prevBytes;
} sOpusStreamDcpt;
```

Members

Members	Description
sOggPageHdr pageHdr;	current page header
int segIx;	current packet segment index in the current page
int pktIx;	current packet index, 0 -> ...
int prevBytes;	previous value of the bytes in the encoded output buffer

Description

info needed by the stream at run-time

DECODER_OPUS_SUPPORT_H Macro

File

opus_support.h

C

```
#define DECODER_OPUS_SUPPORT_H
```

Description

This is macro DECODER_OPUS_SUPPORT_H.

OPUS_INPUT_BUFFER_SIZE Macro

File

opus_support.h

C

```
#define OPUS_INPUT_BUFFER_SIZE (1024*2)
```

Description

MACROS

OPUS_MAX_FRAME_SIZE Macro

File

opus_support.h

C

```
#define OPUS_MAX_FRAME_SIZE (960*6) // 120ms @ 48Khz
```

Description

120ms @ 48Khz

OPUS_OUTPUT_BUFFER_SIZE Macro

File

opus_support.h

C

```
#define OPUS_OUTPUT_BUFFER_SIZE (1024*7)
```

Description

This is macro OPUS_OUTPUT_BUFFER_SIZE.

Files

Files

Name	Description
opus.h	Opus reference implementation API.
opus_support.h	This is file opus_support.h.

Description

This section lists the source and header files used by the Opus Decoder Library.

opus.h

Opus reference implementation API.

Functions

	Name	Description
	OPUS_ARG_NONNULL	This is function OPUS_ARG_NONNULL.
	OPUS_ARG_NONNULL	This is function OPUS_ARG_NONNULL.
	opus_decoder_create	Allocates and initializes an decoder state.
	opus_decoder_destroy	Frees an OpusDecoder allocated by opus_decoder_create.
	opus_decoder_get_size	Gets the size of an OpusDecoder structure.
	opus_encoder_create	Allocates and initializes an encoder state.
	opus_encoder_destroy	Frees an OpusEncoder allocated by opus_encoder_create.
	opus_encoder_get_size	Gets the size of an OpusEncoder structure.

Macros

	Name	Description
	OPUS_H	Opus reference implementation API

Types

	Name	Description
	OpusDecoder	Opus decoder state.
	OpusEncoder	This contains the complete state of an Opus encoder. It is position independent and can be freely copied. See opus_encoder_create,opus_encoder_init.

Description

Opus codec Library

Opus reference implementation API.

File Name

opus.h

Company

Microchip Technology Inc.

opus_support.h

Enumerations

	Name	Description
	OPUS_ERROR_MSG	Ogg Container Structures

Functions

	Name	Description
	isOPUSdecoder_enabled	Checks if Opus decoder is enabled by MHC configuration.
	OPUS_Cleanup	Frees the memory allocated by the Opus Decoder.
	OPUS_Decoder	Called once to decode one Opus packet.
	OPUS_DiskRead	Called once to read one Opus packet.
	OPUS_GetChannels	Returns the number of channels for this Opus audio.
	OPUS_GetSamplingRate	Returns the sampling rate of Opus audio.
	OPUS_Initialize	Initializes the Opus decoder and creates an Opus decoder state structure.

Macros

	Name	Description
	DECODER_OPUS_SUPPORT_H	This is macro DECODER_OPUS_SUPPORT_H.
	OPUS_INPUT_BUFFER_SIZE	MACROS
	OPUS_MAX_FRAME_SIZE	120ms @ 48Khz
	OPUS_OUTPUT_BUFFER_SIZE	This is macro OPUS_OUTPUT_BUFFER_SIZE.

Structures

	Name	Description
	opusDecDcpt	This is type opusDecDcpt.
	sOggPageHdr	header of an Ogg page, full segment info included
	sOpusHeader	This is type sOpusHeader.
	sOpusPktDcpt	decoder data packet descriptor
	sOpusStreamDcpt	info needed by the stream at run-time

Description

This is file opus_support.h.

Speex Decoder Library

This section describes the Speex Decoder Library.

Introduction

Introduces the Speex Decoder Library.

Description

Speex is a Code Excited Linear Prediction (CELP) based open source patent-free audio compression format designed for speech.

Speex Features

- Fixed-point implementation
- Narrowband (8 kHz) and wideband (16 kHz) bit-streams
- Wide range of bit-rates available (from 2.15 kbps to 44 kbps)
- Perceptual enhancement (attempts to reduce the perception of the noise/distortion produced by the encoding/decoding process. In most cases, perceptual enhancement brings the sound further from the original objectively (e.g. considering only SNR), but in the end it still sounds better (subjective improvement)

Visit www.speex.org for further details and complete documentation

Typical Applications

- Answering machines; Voice recorders
- Building and home safety systems; Intercoms
- Smart appliances
- Walkie-talkies; Toys and robots
- Any application using message playback

Resources

Feature	Option	Usage
Speex Library Flash Size	Release Build	< 60 KB
	Debug Build	< 92 KB
Speex Decoder RAM	Dynamic	4 KB
	Stack	2 KB
Speex Decoding Performance	Narrow Mode	280 KB/sec

Using the Library

This topic describes the basic architecture of the Speex Decoder Library and provides information and examples on its use.

Description

Interface Header File: `speex.h`

The interface to the Speex Decoder Library is defined in the `speex.h` header file. Any C language source (`.c`) file that uses the Speex Decoder Library should include `speex.h`.

Please refer to the What is MPLAB Harmony? section for how the Speex Decoder Library interacts with the framework.

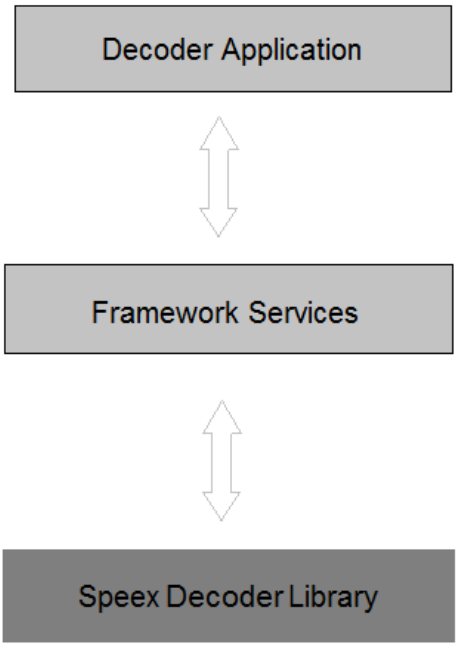
Abstraction Model

Describes the abstraction model for the Speex Decoder Library.

Description

This Speex Library is an Open Source/Free Software patent-free audio compression format designed for speech. The MPLAB Harmony framework provides an abstraction layer to easily interact with this library. Refer to the universal_audio_decoders demonstration for reference.

Abstraction Model Diagram



Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the Speex Decoder Library module.

How the Library Works

This section describes how to work with the MPLAB Harmony Speex abstraction layer interface, which provides simplifier APIs for using the Speex Decoder Library.

Description

For complete documentation and how to implement Speex, visit www.speex.org.

Code Example

The following code provides an example for initializing the Speex Decoder Library and reading and decoding one Speex packet.

```
void Decoder_Initialize(decoder_type){
    switch (decoder_type){
        case SPEEX:
            ret = SPEEX_Initialize();
            if(ret == success)
            {
                // start decoding
            }
    }
}

void App_Task()
{
    while(1){
        while(audio is not end){
            // Read one Speex packet
            appData.nBytesRead = SPEEX_DiskRead(input_ptr);

            // Decode one packet
            ret = SPEEX_Decoder (input_ptr,inSize,read,output,written);
            if(ret == success)
            {
                // continue decoding
            }else{
                // handle decoding errors;
```

```
            }
        }
        // Clean up
        SPEEX_Cleanup();
    }

}
```

Library Interface

a) General Functions

	Name	Description
	isSPEEXdecoder_enabled	Indicates whether the Speex Decoder is enabled.
	SPEEX_Cleanup	Frees the memory allocated by the Speex Decoder.
	SPEEX_Decoder	Called once to decode one packet.
	SPEEX_DiskRead	Reads one packet of Ogg-Speex audio.
	SPEEX_GetBitrate	Returns the bit rate of Ogg-Speex audio.
	SPEEX_GetSamplingRate	Returns the sampling rate of Ogg-Speex audio.
	SPEEX_Initialize	Reads the Speex header page and initializes the Speex Decoder state.

b) Data Types and Constants

	Name	Description
	eSpxFlags	Speex flags
	pProgressFnc	progress display function
	progressDcpt	encoder/decoder progress activity descriptor
	sOggPageSegHdr	header of an Ogg page, full segment info included
	SPEEX_ERROR_MSG	This is type SPEEX_ERROR_MSG.
	spxCdcDcpt	Speex codec descriptor common part of encoder/decoder since we don't take the trouble to have a type to distinguish between descriptors, we need to have a common part
	spxDecDcpt	This is type spxDecDcpt.
	sSpeexHeader	the Speex header, the first page in the Speex stream
	sSpxPktDcpt	decoder data packet descriptor
	sSpxRunDcpt	run-time Speex descriptor obtained from the stream with AudioStreamGetRunInfo()
	sSpxStreamDcpt	info needed by the stream at run-time
	OGG_ID_SPEEX	The Speex packet ID
	SPEEX_INPUT_BUFFER_SIZE	This is macro SPEEX_INPUT_BUFFER_SIZE.
	SPEEX_OUTPUT_BUFFER_SIZE	This is macro SPEEX_OUTPUT_BUFFER_SIZE.
	SPEEX_STRING_LENGTH	The size of the Speex string
	SPEEX_SUPPORT_H	This is macro SPEEX_SUPPORT_H.
	SPEEX_VENDOR_STR	comment in Ogg header
	SPEEX_VERSION	The Speex version string
	SPEEX_VERSION_ID	Version identifier
	SPEEX_VERSION_LENGTH	The size of the Speex version string
	SPX_CODEC_BUFF_DIV	20% is sufficient
	SPX_CODEC_BUFF_MUL	Values to adjust the average decoder buffer size

Description

This section describes the Application Programming Interface (API) functions of the Speex Decoder Library.

Refer to each section for a detailed description.

a) General Functions

isSPEEXdecoder_enabled Function

Indicates whether the Speex Decoder is enabled.

File

speex.h

C

```
bool isSPEEXdecoder_enabled();
```

Returns

This function returns a Boolean value.

- true - Indicates that the Speex Decoder is enabled
- false - Indicates that the Speex Decoder is disabled

Description

This function indicates whether the Speex Decoder is enabled.

Preconditions

None.

Example

```
appData.SPEEX_decoder_enabled   = isSPEEXdecoder_enabled();
```

Function

bool isSPEEXdecoder_enabled()

SPEEX_Cleanup Function

Frees the memory allocated by the Speex Decoder.

File

speex.h

C

```
void SPEEX_Cleanup();
```

Returns

None.

Description

This function frees the memory that was allocated by the Speex Decoder and is called when the Speex Decoder ends.

Preconditions

None.

Example

```
SPEEX_Cleanup();
```

Function

void SPEEX_Cleanup()

SPEEX_Decoder Function

Called once to decode one packet.

File

speex.h

C

```
SPEEX_ERROR_MSG SPEEX_Decoder(uint8_t * input, uint16_t inSize, uint16_t * read, int16_t * output, uint16_t
* written);
```

Returns

This function returns an enum value.

- SPEEX_SUCCESS - Indicates success
- SPEEX_READ_ERROR - Indicates read failure
- SPEEX_STREAM_ERROR - Indicates Ogg-Speex parsing error
- SPEEX_STREAM_END - Indicates end of the stream

Description

This function is called once to decode one frame. This function will be called in an infinite while loop until the end of the frame is detected or the decode function returns failure value.

Preconditions

None.

Example

```
res = SPEEX_Decoder (input,inSize,read,output,written);
```

Parameters

Parameters	Description
*input	The pointer to the input buffer , where the encoded bit stream is available
inSize	A variable of data type uint16_t indicating the number of valid bytes available in the input buffer
*read	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of bytes consumed from the current frame decoder operation.
*output	The pointer to the output sample buffer where the decided PCM samples are to be written,
*written	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of valid bytes available in the output buffer.

Function

SPEEX_ERROR_MSG SPEEX_Decoder(uint8_t *input, uint16_t inSize, uint16_t *read,
int16_t *output, uint16_t *written);

SPEEX_DiskRead Function

Reads one packet of Ogg-Speex audio.

File

speex.h

C

```
int32_t SPEEX_DiskRead(uint8_t * inBuff);
```

Returns

This function returns the size of reading bytes.

Description

This function is called once to read one packet.

Preconditions

None.

Example

```
appData.nBytesRead = SPEEX_DiskRead(ptr);
```

Parameters

Parameters	Description
*inBuff	The pointer to the input buffer, where one packet data is to to be written

Function

int32_t SPEEX_DiskRead(uint8_t *inBuff)

SPEEX_GetBitrate Function

Returns the bit rate of Ogg-Speex audio.

File

speex.h

C

```
int32_t SPEEX_GetBitrate();
```

Returns

This function returns the bit rate.

Description

This function returns the bit rate rate of Ogg-Speex audio.

Preconditions

This function can only be called after the SPEEX_Initialize function.

Example

```
decoderBitrate = SPEEX_GetBitrate()/1000;
```

Function

int32_t SPEEX_GetBitrate()

SPEEX_GetSamplingRate Function

Returns the sampling rate of Ogg-Speex audio.

File

speex.h

C

```
int32_t SPEEX_GetSamplingRate();
```

Returns

This function returns the sampling rate.

Description

This function returns the sampling rate of Ogg-Speex audio.

Preconditions

This function can only be called after the SPEEX_Initialize function.

Example

```
sampling_frequency = SPEEX_GetSamplingRate();
```

Function

int32_t SPEEX_GetSamplingRate()

SPEEX_Initialize Function

Reads the Speex header page and initializes the Speex Decoder state.

File

speex.h

C

```
SPEEX_ERROR_MSG SPEEX_Initialize(uintptr_t spx_file_handle);
```

Returns

This function returns an enum value.

- SPEEX_SUCCESS - Indicates success
- SPEEX_READ_ERROR - Indicates read failure
- SPEEX_STREAM_ERROR - Indicates Ogg-Speex parsing error
- SPEEX_STREAM_END - Indicates end of the stream

Description

The function internally calls the Speex Codec Decoder initialization function. The function is called only once before initiating the decoding process.

Preconditions

Call SPEEX_Cleanup function before this function for safety.

Example

```
APP_ERROR_MSG res = SPEEX_Initialize(spx_file_handle);
```

Function

SPEEX_ERROR_MSG SPEEX_Initialize(uintptr_t spx_file_handle);

b) Data Types and Constants

eSpxFlags Enumeration

File

speex.h

C

```
typedef enum {
  SPX_FLAG_WB = 0x1,
  SPX_FLAG_VBR = 0x2,
  SPX_FLAG_BRATE_OVR = 0x4,
  SPX_FLAG_PRCPT_ENH = 0x8
} eSpxFlags;
```

Members

Members	Description
SPX_FLAG_WB = 0x1	wide/narrow band
SPX_FLAG_VBR = 0x2	VBR
SPX_FLAG_BRATE_OVR = 0x4	bit rate setting overrides the encoder quality
SPX_FLAG_PRCPT_ENH = 0x8	decoder perceptual enhancement on/off

Description

Speex flags

Remarks

at most 8 flags are supported right now!

pProgressFnc Type

File

speex.h

C

```
typedef void (* pProgressFnc)(int nBytes);
```

Description

progress display function

progressDcpt Structure

File

speex.h

C

```
typedef struct {
  int progressStep;
  int progressCnt;
  pProgressFnc progressFnc;
} progressDcpt;
```

Members

Members	Description
int progressStep;	no of bytes to process before calling the progress callback
int progressCnt;	current counter
pProgressFnc progressFnc;	progress callback

Description

encoder/decoder progress activity descriptor

sOggPageSegHdr Structure

File

speex.h

C

```
typedef struct {
  int32_t pageCapture;
  int8_t struct_ver;
  int8_t headerFlags;
  int64_t granulePos;
  int32_t streamNo;
  int32_t pageNo;
  int32_t pageCrc;
  int8_t pageSegments;
  uint8_t segmentTbl[255];
} sOggPageSegHdr;
```

Members

Members	Description
int32_t pageCapture;	should be OGG_ID_MAGIC
int8_t struct_ver;	version of the Ogg file format. Should be 0 (RFC3533)
int8_t headerFlags;	an eOggHeaderFlags value
int64_t granulePos;	stream dependent position info
int32_t streamNo;	logical bit stream identifier
int32_t pageNo;	page sequence number
int32_t pageCrc;	CRC32 checksum of the page
int8_t pageSegments;	number of page segments to follow
uint8_t segmentTbl[255];	actually segmentTbl[pageSegments]; contains the lace values for all segments in the page

Description

header of an Ogg page, full segment info included

SPEEX_ERROR_MSG Enumeration

File

speex.h

C

```
typedef enum {
  SPEEX_SUCCESS = 1,
  SPEEX_READ_ERROR,
  SPEEX_STREAM_ERROR,
  SPEEX_BUFF_ERROR,
  SPEEX_STREAM_END,
  SPEEX_PLAYBACK_ERROR,
  SPEEX_OUT_OF_MEM_ERROR,
  SPEEX_DISK_ERROR,
  SPEEX_GENERAL_ERROR
} SPEEX_ERROR_MSG;
```

Description

This is type SPEEX_ERROR_MSG.

spxCdcDcpt Structure

File

speex.h

C

```
typedef struct {
  SpeexBits spxBits;
  void* spxState;
  int inBuffSize;
  int outBuffSize;
  int nOutBytes;
} spxCdcDcpt;
```

Members

Members	Description
SpeexBits spxBits;	codec control structure
void* spxState;	codec state
int inBuffSize;	input buffer size
int nOutBytes;	bytes available in the output buffer

Description

Speex codec descriptor common part of encoder/decoder since we don't take the trouble to have a type to distinguish between descriptors, we need to have a common part

spxDecDcpt Structure

File

speex.h

C

```
typedef struct {
  spxCdcDcpt cdc;
  int framesPerPacket;
  int frameSize;
  int processedPktNo;
  int currPktNo;
  int nInBytes;
  int outFrameSize;
} spxDecDcpt;
```

Members

Members	Description
spxCdcDcpt cdc;	common part
int framesPerPacket;	frames per Ogg packet
int frameSize;	size of the frames
int processedPktNo;	counter of processed packets

int currPktNo;	number of the currently received packet from the stream
int nInBytes;	bytes available in the input buffer

Description

This is type spxDecDcpt.

sSpeexHeader Structure

File

speex.h

C

```
typedef struct {
  char speexString[SPEEX_STRING_LENGTH];
  char speexVer[SPEEX_VERSION_LENGTH];
  int32_t speexVerId;
  int32_t headerSize;
  int32_t sampleRate;
  int32_t wBand;
  int32_t modeBitsStreamVer;
  int32_t nChannels;
  int32_t bitRate;
  int32_t frameSamples;
  int32_t vbr;
  int32_t framesPerPacket;
  int32_t extraHeaders;
  int32_t reserved1;
  int32_t reserved2;
} sSpeexHeader;
```

Members

Members	Description
char speexString[SPEEX_STRING_LENGTH];	identify the Speex bit-stream: OGG_ID_SPEEX
char speexVer[SPEEX_VERSION_LENGTH];	Speex version that encoded the file
int32_t speexVerId;	Speex version (for checking compatibility)
int32_t headerSize;	sizeof(SpeexHeader)
int32_t sampleRate;	Sampling rate used
int32_t wBand;	0 for narrowband, 1 for wideband
int32_t modeBitsStreamVer;	Version ID of the bit-stream
int32_t nChannels;	Number of channels encoded
int32_t bitRate;	Bit-rate used
int32_t frameSamples;	Size of frames, samples
int32_t vbr;	1 for a VBR encoding, 0 otherwise
int32_t framesPerPacket;	Number of frames stored per Ogg packet
int32_t extraHeaders;	Number of additional headers after the comments
int32_t reserved1;	Reserved for future use, 0
int32_t reserved2;	Reserved for future use, 0

Description

the Speex header, the first page in the Speex stream

sSpxPktDcpt Structure

File

speex.h

C

```
typedef struct {
  int pktBytes;
  int pktSeqNo;
} sSpxPktDcpt;
```

Members

Members	Description
int pktBytes;	how many bytes in this packet
int pktSeqNo;	packet sequence number

Description

decoder data packet descriptor

sSpxRunDcpt Structure

File

speex.h

C

```
typedef struct {
  int streamNo;
  int streamVer;
  unsigned short frameSamples;
  unsigned short bitRate;
  unsigned char framesPerPacket;
  unsigned char packetsPerPage;
  unsigned char complexity;
  unsigned char qualFactor;
  char spxFlags;
  char reserved[3];
} sSpxRunDcpt;
```

Members

Members	Description
int streamNo;	stream number inside the container
int streamVer;	stream version
unsigned short frameSamples;	Size of frames, in samples
unsigned short bitRate;	encoder bit rate
unsigned char framesPerPacket;	Number of frames stored per Ogg packet, <10
unsigned char packetsPerPage;	number of packets in an Ogg page <= 255
unsigned char complexity;	encoder complexity 1-10
unsigned char qualFactor;	encoder quality factor 1-10
char spxFlags;	eSpxFlags: run time flags, wideband, VBR
char reserved[3];	future expansion/padding

Description

run-time Speex descriptor obtained from the stream with AudioStreamGetRunInfo()

sSpxStreamDcpt Structure

File

speex.h

C

```
typedef struct {
  sSpxRunDcpt runDcpt;
  sOggPageSegHdr pageHdr;
  int segIx;
  int pktIx;
  int prevBytes;
} sSpxStreamDcpt;
```

Members

Members	Description
sSpxRunDcpt runDcpt;	global info

sOggPageSegHdr pageHdr;	current page header
int segIx;	current packet segment index in the current page
int pktIx;	current packet index, 0 -> ...
int prevBytes;	previous value of the bytes in the encoded output buffer

Description

info needed by the stream at run-time

OGG_ID_SPEEX Macro

File

speex.h

C

```
#define OGG_ID_SPEEX "Speex   "    // The Speex packet ID
```

Description

The Speex packet ID

SPEEX_INPUT_BUFFER_SIZE Macro

File

speex.h

C

```
#define SPEEX_INPUT_BUFFER_SIZE 1024
```

Description

This is macro SPEEX_INPUT_BUFFER_SIZE.

SPEEX_OUTPUT_BUFFER_SIZE Macro

File

speex.h

C

```
#define SPEEX_OUTPUT_BUFFER_SIZE 1024*7
```

Description

This is macro SPEEX_OUTPUT_BUFFER_SIZE.

SPEEX_STRING_LENGTH Macro

File

speex.h

C

```
#define SPEEX_STRING_LENGTH 8           // The size of the Speex string
```

Description

The size of the Speex string

SPEEX_SUPPORT_H Macro

File

speex.h

C

```
#define SPEEX_SUPPORT_H
```

Description

This is macro SPEEX_SUPPORT_H.

SPEEX_VENDOR_STR Macro

File

speex.h

C

```
#define SPEEX_VENDOR_STR "Encoded by Microchip Audio Library ver 1.0"    // comment in Ogg header
```

Description

comment in Ogg header

SPEEX_VERSION Macro

File

speex.h

C

```
#define SPEEX_VERSION "speex-1.2beta3"    // The Speex version string
```

Description

The Speex version string

SPEEX_VERSION_ID Macro

File

speex.h

C

```
#define SPEEX_VERSION_ID 1                    // Version identifier
```

Description

Version identifier

SPEEX_VERSION_LENGTH Macro

File

speex.h

C

```
#define SPEEX_VERSION_LENGTH 20               // The size of the Speex version string
```

Description

The size of the Speex version string

SPX_CODEC_BUFF_DIV Macro

File

speex.h

C

```
#define SPX_CODEC_BUFF_DIV 5    // 20% is sufficient
```

Description

20% is sufficient

SPX_CODEC_BUFF_MUL Macro

File

speex.h

C

```
#define SPX_CODEC_BUFF_MUL 6    // Values to adjust the average decoder buffer size
```

Description

Values to adjust the average decoder buffer size

Files

Files

Name	Description
speex.h	This file is an abstraction layer of the Speex Library.

Description

This section lists the source and header files used by the Speex Decoder Library.

speex.h

This file is an abstraction layer of the Speex Library.

Enumerations

	Name	Description
	eSpxFlags	Speex flags
	SPEEX_ERROR_MSG	This is type SPEEX_ERROR_MSG.

Functions

	Name	Description
	isSPEEXdecoder_enabled	Indicates whether the Speex Decoder is enabled.
	SPEEX_Cleanup	Frees the memory allocated by the Speex Decoder.
	SPEEX_Decoder	Called once to decode one packet.
	SPEEX_DiskRead	Reads one packet of Ogg-Speex audio.
	SPEEX_GetBitrate	Returns the bit rate of Ogg-Speex audio.
	SPEEX_GetSamplingRate	Returns the sampling rate of Ogg-Speex audio.
	SPEEX_Initialize	Reads the Speex header page and initializes the Speex Decoder state.

Macros

	Name	Description
	OGG_ID_SPEEX	The Speex packet ID
	SPEEX_INPUT_BUFFER_SIZE	This is macro SPEEX_INPUT_BUFFER_SIZE.
	SPEEX_OUTPUT_BUFFER_SIZE	This is macro SPEEX_OUTPUT_BUFFER_SIZE.
	SPEEX_STRING_LENGTH	The size of the Speex string
	SPEEX_SUPPORT_H	This is macro SPEEX_SUPPORT_H.
	SPEEX_VENDOR_STR	comment in Ogg header
	SPEEX_VERSION	The Speex version string
	SPEEX_VERSION_ID	Version identifier
	SPEEX_VERSION_LENGTH	The size of the Speex version string
	SPX_CODEC_BUFF_DIV	20% is sufficient
	SPX_CODEC_BUFF_MUL	Values to adjust the average decoder buffer size

Structures

	Name	Description
	progressDcpt	encoder/decoder progress activity descriptor
	sOggPageSegHdr	header of an Ogg page, full segment info included
	spxCdcDcpt	Speex codec descriptor common part of encoder/decoder since we don't take the trouble to have a type to distinguish between descriptors, we need to have a common part
	spxDecDcpt	This is type spxDecDcpt.
	sSpeexHeader	the Speex header, the first page in the Speex stream
	sSpxPktDcpt	decoder data packet descriptor
	sSpxRunDcpt	run-time Speex descriptor obtained from the stream with AudioStreamGetRunInfo()
	sSpxStreamDcpt	info needed by the stream at run-time

Types

	Name	Description
	pProgressFnc	progress display function

Description

Speex Decoder Library Interface Definition

This file contains the Speex Decoder-specific definitions and function prototypes.

File Name

speex.h

Company

Microchip Technology Inc.

WMA Decoder Library

This section describes the WMA Decoder Library.

Introduction

The WMA Decoder Library is available for the PIC32 family of microcontrollers.

Description

The WMA Decoder Library is a premium product and is not included in the standard release of MPLAB Harmony and must be purchased separately. The library is only available in binary format, and is only available to Microsoft Windows Media Component Licensees. For licensing information, please visit: http://windows.microsoft.com/en-us/windows/windowsmedia-components-licensing. Refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for additional information.

Windows Media Audio (WMA) developed by Microsoft, is a format enabling the storage of digital audio using the Lossy compression algorithm.

The Microchip WMA Decoder can decode audio signals sampled at up to 48 kHz with up to two discrete channels. The WMA Decoder also supports VBR and CBR encoded audio stream. In most circumstances, `.wma` files are contained in Advance Systems Format (ASF), which is supported by the WMA Decoder. The WMA Decoder library is optimized (C/ASM) and is available for all PIC32MX devices.

Assembly Language Issue

There is an assembly language issue in the WMA Decoder Library that misaligns the stack pointer. Contrary to the MIPS programming specification, this error is benign except when the application is built for a device with a hardware Floating Point Unit (FPU), such as the PIC32MZ EF family of devices. In this case, the sub-routine context-saving operations will produce an exception and stop the application. A work around exists for this issue, which involves disabling FPU context-saving for Interrupt Service Routines (ISRs).

The work around is to convert all ISRs from:

```
void __ISR(vector,ipl) MyInterruptHandler(void)
```

and change them to:

```
void __attribute__((vector),interrupt(ipl),nomips16,no_fpu)) MyInterruptHandler(void)
```

where, `vector` and `ipl` are replaced with the actual interrupt vector and interrupt priority level.

An example of this work around can be found in the universal_audio_decoders demonstration.

So as long as no FPU operations occur in ISRs and the routines are called by ISRs, this work around will prevent the misaligned stack pointer from causing an exception. If FPU operations result from an ISR with no_fpu enabled, a general exception will result.

Using the Library

This topic describes the basic architecture of the WMA Decoder Library and provides information and examples on its use.

Description

Interface Header File: `decoder_wma.h`

The interface to the WMA Decoder Library is defined in the `decoder_wma.h` header file. Any C language source (`.c`) file that uses the MP3 Decoder Library should include `decoder_wma.h`.

Please refer to the What is MPLAB Harmony? section for how the WMA Decoder Library interacts with the framework.

Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the WMA Decoder Library module.

Library Interface

a) General Functions

	Name	Description
	stpnr	This is function stpnr.
	WMA_Decode	This function is called once to decode one or multiple frames.
	WMA_Decoder_Init	This function reads the pointers and clears the structure.
	WMA_FileGetPCM	Extracts the 16-bit samples.
	WMA_GetEncodeData	Retrieves encoded data from the WAVE file header.
	WMA_GetStateSize	This function requires the required memory for the WMA Decoder state.
	WMA_PacketData	Extracts the encoded bitstream from the packet's payload.
	WMA_PacketDatasize	This function determines the packet size for decoding.
	WMA_Parser	Parses the WMA data.
	WMA_Decoder	This is function WMA_Decoder.
	WMA_SamplingFrequency_Get	This is function WMA_SamplingFrequency_Get.
	WMA_Header_Current_Offset	This function returns the amount of data used and updates the pointer.
	WMA_WaveHdr	Generates the .wav file header and fills the WavBuf array.
	WMA_FileDecodeClr	Clears all of the allocated memory.
	WMA_FreeMemory	This is function WMA_FreeMemory.
	isWMAdecoder_enabled	This is function isWMAdecoder_enabled.
	WMA_BitRate_Get	This is function WMA_BitRate_Get.
	WMA_GetHeaderPacketOffset	This is function WMA_GetHeaderPacketOffset.
	WMA_Initialize	This is function WMA_Initialize.
	WMA_RegisterAppCallback	This is function WMA_RegisterAppCallback.
	WMA_GetChannels	This is function WMA_GetChannels.

b) Data Types and Constants

	Name	Description
	SYS_DEBUG_BUFFER_DMA_READY	This should be defined in system_config.h. It is added here as a build safe-guard.
	WMA_DECODER_STATES	This is type WMA_DECODER_STATES.
	WMA_ERROR_COUNT_MAX	This is macro WMA_ERROR_COUNT_MAX.
	WMA_H	This is macro WMA_H.
	Chnl_Info	This is type Chnl_Info.
	WMA_Status	This is type WMA_Status.
	BYTES_PER_SAMPLE	This is macro BYTES_PER_SAMPLE.
	MAX_SAMPLES	in Bytes #define WAVE_HEADER_SIZE 44 //in Bytes
	WMA_MAX_INPUT_BUFFER_SIZE	in Bytes
	GetReadBytesInAppData	This is type GetReadBytesInAppData.
	SetReadBytesReadFlagInAppData	This is type SetReadBytesReadFlagInAppData.

Description

This section describes the Application Programming Interface (API) functions of the WMA Decoder Library.

Refer to each section for a detailed description.

a) General Functions

stpnr Function

File

decoder_wma.h

C

```
static __inline int stpnr(int x);
```

Description

This is function stpnr.

WMA_Decode Function

This function is called once to decode one or multiple frames.

File

decoder_wma.h

C

```
int32_t WMA_Decode(void* paudec, int* pcSamplesReady, int* paudecState);
```

Returns

This function returns the error type.

Description

This function is called once to decode one or multiple frames. This function will be called in an infinite while loop until the end-of-frame is detected or the decode function returns an error value. Unlike most speech applications, where the encoded data for each frame is deterministic, this is not the case in audio applications. Therefore, the application provides a chunk of input packet data (usually more than what is necessary for a single frame). The decoder parses them and decodes with a portion of the data. The decode function passes the information of how many bytes of data were used to the application so that the application saves the remaining data and presents to the decoder for the subsequent frame.

Remarks

None.

Preconditions

None.

Example

```
Int res;
res = WMA_Decode(WMA_State, &nDecodedSamples, &chnlInfo.WMAState);
```

Parameters

Parameters	Description
paudec	The pointer to the WMA Decoder handler
pcSamplesReady	The pointer to the number of decoded samples
paudecState	The pointer to the state of the decoder

Function

int32_t WMA_Decode (void* paudec, int* pcSamplesReady, int* paudecState);

WMA_Decoder_Init Function

This function reads the pointers and clears the structure.

File

decoder_wma.h

C

```
WMA_Status WMA_Decoder_Init(void* pState, Chnl_Info * chnl, char * WMADataBuf);
```

Returns

This function returns the error result.

Description

This function views the received pointer as structure pointer and clears the entire structure. Subsequently, it initializes the selective variables with the default values. This function is called only once for each file before starting the decoding of an entire packet.

Preconditions

None.

Example

```
void *WMA_State;
Chnl_Info chnlInfo;
WMA_Status WMA_status;
char WMA_InBuf[MAX_INPUT_BUF];

WMA_Status = WMA_Decoder_Init (WMA_State, &chnlInfo, WMA_InBuf);
```

Parameters

Parameters	Description
pState	The pointer to the allocated state memory.
*chnl	Holds the essential audio file information and state of the file
WMADataBuf	The pointer to the input buffer

Function

WMA_Status WMA_Decoder_Init (void* pState, Chnl_Info *chnl, char * WMADataBuf);

WMA_FileGetPCM Function

Extracts the 16-bit samples.

File

decoder_wma.h

C

```
uint32_t WMA_FileGetPCM(void * pInt, short * pCh, uint32_t max_nsamples);
```

Returns

This function returns the number of samples.

Description

This function is used to extract the 16-bit samples.

Preconditions

None.

Example

```
nDecodedSamples = WMA_FileGetPCM (WMA_State, WMA_OutBuf, nDecodedSamples);
```

Parameters

Parameters	Description
*pInt	The pointer to the WMA handler
*pCh	The pointer to the output data
max_nsamples	Total number of samples

Function

uint32_t WMA_FileGetPCM (void *pInt, short *pCh, uint32_t max_nsamples);

WMA_GetEncodeData Function

Retrieves encoded data from the WAVE file header.

File

decoder_wma.h

C

```
void WMA_GetEncodeData(void * fWMA, char * pData, int pDtSz);
```

Returns

None.

Description

The function is used to retrieve the encoded data from the WAVE file header.

Preconditions

None.

Parameters

Parameters	Description
*fWMA	The WMA file
*pData	The pointer to the data
pDtSz	The pointer to the data size

Function

void WMA_GetEncodeData(void *fWMA, char *pData, int pDtSz);

WMA_GetStateSize Function

This function requires the required memory for the WMA Decoder state.

File

decoder_wma.h

C

```
uint32_t WMA_GetStateSize();
```

Returns

This function returns the size of the WMA Decoder state structure in bytes.

Description

Based on the return value, the application allocates the required memory for the WMA Decoder state. This function is called only once for each channel before the channel is initialized. If dynamic memory allocation (malloc) is not needed, it is not necessary to call this function and the WMA Decoder state buffer will be statically defined at compile time with the size of the WMA Decoder state structure (constant).

Preconditions

None.

Example

```
unsigned long nDecodedSamples;
nDecodedSamples = WMA_GetStateSize();
```

Function

uint32_t WMA_GetStateSize(void);

WMA_PacketData Function

Extracts the encoded bitstream from the packet's payload.

File

decoder_wma.h

C

```
int WMA_PacketData(void * p, char * InBuf, int residue);
```

Returns

None.

Description

This function is used to extract the encoded bitstream from the packet’s payload. This function is repeated until there are no bytes to decode.

Preconditions

None.

Example

```
void *WMA_State;
Chnl_Info chnlInfo;
Int res, residue;
res = WMA_PacketData(WMA_State,WMA_InBuf,residue);
```

Parameters

Parameters	Description
*p	The pointer to the WMA handler
InBuf	The pointer to the input buffer
residue	The pointer to the input file

Function

int WMA_PacketData(void *p, char * InBuf, int residue);

WMA_PacketDatasize Function

This function determines the packet size for decoding.

File

decoder_wma.h

C

```
int32_t WMA_PacketDatasize(void * p);
```

Returns

This function returns the packet data size.

Description

This function determines the packet size for decoding and sends request to the main program to read the number of bytes data for decoding. This function is called repetitively until there are no bytes to decode.

Preconditions

None.

Example

```
void *WMA_State;
int packetdatasize;
packetdatasize = WMA_PacketDatasize(WMA_State);
```

Parameters

Parameters	Description
*p	The pointer to the WMA handler

Function

int32_t WMA_PacketDatasize(void *p);

WMA_Parser Function

Parses the WMA data.

File

decoder_wma.h

C

```
int WMA_Parser(void * pInt);
```

Returns

None.

Description

This function is used to parse the WMA data.

Preconditions

None.

Parameters

Parameters	Description
*pInt	The pointer to the WMA handler

Function

int WMA_Parser(void *pInt);

WMA_Decoder Function

File

wma.h

C

```
int16_t WMA_Decoder(uint8_t * input, uint16_t inSize, uint16_t * read, int16_t * output, uint16_t *
written);
```

Description

This is function WMA_Decoder.

WMA_SamplingFrequency_Get Function

File

wma.h

C

```
int32_t WMA_SamplingFrequency_Get();
```

Description

This is function WMA_SamplingFrequency_Get.

WMA_Header_Current_Offset Function

This function returns the amount of data used and updates the pointer.

File

decoder_wma.h

C

```
uint32_t WMA_Header_Current_Offset(void * pInt);
```

Returns

This function returns the amount of data used.

Description

This function returns the amount of data used and updates the file pointer location to read input data from the current offset after parsing the header information. This function is called only once for each file before starting the decoding of entire packets.

Preconditions

None.

Example

```
Uint32_t packet_offset;
packet_offset = WMA_Header_Current_Offset(WMA_State);
```

Parameters

Parameters	Description
*pInt	The pointer to the WMA Handler

Function

uint32_t WMA_Header_Current_Offset(tWMAFileStateInternal *pInt)

WMA_WaveHdr Function

Generates the .wav file header and fills the WavBuf array.

File

decoder_wma.h

C

```
void WMA_WaveHdr(char * WavHdr, void * hWMA, uint32_t DataLen);
```

Returns

None.

Description

For wave file generation, the application will leave 44 bytes of free space at the beginning of the file and begins writing decoded samples from the 45th byte location onwards. Once the entire data had been decoded, this function is called. The function generates the .wav file header and fills the WavBuf array. The application copies the contents of this buffer into the reserved place available at the beginning of the file.

Preconditions

None.

Example

```
WMA_WaveHdr(WavBuf, WMA_State, DataLen);
```

Parameters

Parameters	Description
*WavHdr	The pointer to the header buffer of a WAVE file. The buffer size is 44 bytes.
*hWMA	The pointer to the WMA Handler
DataLen	Integer variable indicating the total number of samples written into the file

Function

void WMA_WaveHdr(char *WavHdr, void *hWMA, uint32_t DataLen);

WMA_FileDecodeClr Function

Clears all of the allocated memory.

File

decoder_wma.h

C

```
void WMA_FileDecodeClr(void* WMA_State);
```

Returns

None.

Description

This function is used to clear all of the allocated WMA state memory.

Preconditions

None.

Function

void WMA_FileDecodeClr(void* WMA_State);

WMA_FreeMemory Function

File

wma.h

C

```
void WMA_FreeMemory();
```

Description

This is function WMA_FreeMemory.

isWMAdecoder_enabled Function

File

wma.h

C

```
bool isWMAdecoder_enabled();
```

Description

This is function isWMAdecoder_enabled.

WMA_BitRate_Get Function

File

wma.h

C

```
int32_t WMA_BitRate_Get();
```

Description

This is function WMA_BitRate_Get.

WMA_GetHeaderPacketOffset Function

File

wma.h

C

```
int32_t WMA_GetHeaderPacketOffset();
```

Description

This is function WMA_GetHeaderPacketOffset.

WMA_Initialize Function

File

wma.h

C

```
void WMA_Initialize(SYS_FS_HANDLE wmaFilehandle, uint32_t inputBufferSize);
```

Description

This is function WMA_Initialize.

WMA_RegisterAppCallback Function

File

wma.h

C

```
void WMA_RegisterAppCallback(SetReadBytesReadFlagInAppData fptr0, GetReadBytesInAppData fptr1);
```

Description

This is function WMA_RegisterAppCallback.

WMA_GetChannels Function

File

wma.h

C

```
uint8_t WMA_GetChannels();
```

Description

This is function WMA_GetChannels.

b) Data Types and Constants

SYS_DEBUG_BUFFER_DMA_READY Macro

File

sys_debug.h

C

```
#define SYS_DEBUG_BUFFER_DMA_READY
```

Description

This should be defined in system_config.h. It is added here as a build safe-guard.

WMA_DECODER_STATES Enumeration

File

wma.h

C

```
typedef enum {
  WMA_GET_FRAME_SIZE,
  WMA_DECODE_FRAME
} WMA_DECODER_STATES;
```

Description

This is type WMA_DECODER_STATES.

WMA_ERROR_COUNT_MAX Macro

File

wma.h

C

```
#define WMA_ERROR_COUNT_MAX 1
```

Description

This is macro WMA_ERROR_COUNT_MAX.

WMA_H Macro

File

wma.h

C

```
#define WMA_H
```

Description

This is macro WMA_H.

Chnl_Info Structure

File

decoder_wma.h

C

```
typedef struct {
  long SamplesPerSec;
  long Channels;
  long BitRate;
  long Version;
  long ByteUsed;
  int WMAState;
  long PacketSize;
} Chnl_Info;
```

Members

Members	Description
long SamplesPerSec;	Sampling Frequency
long Channels;	No of Channels
long BitRate;	Bit Rate
long Version;	WMA Version
long ByteUsed;	Used Input Data

int WMAState;	Input or Decode State
long PacketSize;	Size of Packet

Description

This is type Chnl_Info.

WMA_Status Enumeration

File

decoder_wma.h

C

```
typedef enum {
  WMA_NO_ERR = 0,
  WMA_NEED_DATA = 1,
  WMA_ERR = 2
} WMA_Status;
```

Description

This is type WMA_Status.

BYTES_PER_SAMPLE Macro

File

decoder_wma.h

C

```
#define BYTES_PER_SAMPLE 2
```

Description

This is macro BYTES_PER_SAMPLE.

MAX_SAMPLES Macro

File

decoder_wma.h

C

```
#define MAX_SAMPLES 2048                    //in Bytes
```

Description

in Bytes #define WAVE_HEADER_SIZE 44 //in Bytes

WMA_MAX_INPUT_BUFFER_SIZE Macro

File

decoder_wma.h

C

```
#define WMA_MAX_INPUT_BUFFER_SIZE 1024*20   //in Bytes
```

Description

in Bytes

GetReadBytesInAppData Type

File

wma.h

C

```
typedef int32_t (* GetReadBytesInAppData)();
```

Description

This is type GetReadBytesInAppData.

SetReadBytesReadFlagInAppData Type

File

wma.h

C

```
typedef void (* SetReadBytesReadFlagInAppData)(int32_t val, bool b);
```

Description

This is type SetReadBytesReadFlagInAppData.

Files

Files

Name	Description
wma.h	WMA Decoder support API.
decoder_wma.h	This file consists of the abstract function and input output buffer size declaration for decoding purpose

Description

This section lists the source and header files used by the MP3 Decoder Library.

wma.h

WMA Decoder support API.

Enumerations

	Name	Description
	WMA_DECODER_STATES	This is type WMA_DECODER_STATES.

Functions

	Name	Description
	isWMAdecoder_enabled	This is function isWMAdecoder_enabled.
	WMA_BitRate_Get	This is function WMA_BitRate_Get.
	WMA_Decoder	This is function WMA_Decoder.
	WMA_FreeMemory	This is function WMA_FreeMemory.
	WMA_GetChannels	This is function WMA_GetChannels.
	WMA_GetHeaderPacketOffset	This is function WMA_GetHeaderPacketOffset.
	WMA_Initialize	This is function WMA_Initialize.
	WMA_RegisterAppCallback	This is function WMA_RegisterAppCallback.
	WMA_SamplingFrequency_Get	This is function WMA_SamplingFrequency_Get.

Macros

	Name	Description
	WMA_ERROR_COUNT_MAX	This is macro WMA_ERROR_COUNT_MAX.
	WMA_H	This is macro WMA_H.

Types

	Name	Description
	GetReadBytesInAppData	This is type GetReadBytesInAppData.

	SetReadBytesReadFlagInAppData	This is type SetReadBytesReadFlagInAppData.

Description

MP3 Decoder Library Interface File

This header file consists of support function declarations.

File Name

wma.h

Company

Microchip Technology Inc.

decoder_wma.h

This file consists of the abstract function and input output buffer size declaration for decoding purpose

Enumerations

	Name	Description
	WMA_Status	This is type WMA_Status.

Functions

	Name	Description
	stpnr	This is function stpnr.
	WMA_Decode	This function is called once to decode one or multiple frames.
	WMA_Decoder_Init	This function reads the pointers and clears the structure.
	WMA_FileDecodeClr	Clears all of the allocated memory.
	WMA_FileGetPCM	Extracts the 16-bit samples.
	WMA_GetEncodeData	Retrieves encoded data from the WAVE file header.
	WMA_GetStateSize	This function requires the required memory for the WMA Decoder state.
	WMA_Header_Current_Offset	This function returns the amount of data used and updates the pointer.
	WMA_PacketData	Extracts the encoded bitstream from the packet's payload.
	WMA_PacketDatasize	This function determines the packet size for decoding.
	WMA_Parser	Parses the WMA data.
	WMA_WaveHdr	Generates the .wav file header and fills the WavBuf array.

Macros

	Name	Description
	BYTES_PER_SAMPLE	This is macro BYTES_PER_SAMPLE.
	MAX_SAMPLES	in Bytes #define WAVE_HEADER_SIZE 44 //in Bytes
	WMA_MAX_INPUT_BUFFER_SIZE	in Bytes

Structures

	Name	Description
	Chnl_Info	This is type Chnl_Info.

Description

WMA Decoder Library Interface File

The header file consists of function declaration for the abstract functions to invoke decoding . The header file also defines the size of input samples and i/p and o/p buffer.

File Name

decoder_wma.h

Company

Microchip Technology Inc.

Driver Libraries Help

This section provides descriptions of the Driver libraries that are available in MPLAB Harmony.

Driver Library Overview

This topic provides help for the MPLAB Harmony driver libraries. It includes a general driver usage overview, as well as sections providing a programmer's reference for each driver that describes its interface and explains how to use it.

Introduction

Introduces MPLAB Harmony device drivers and explains common usage concepts.

Description

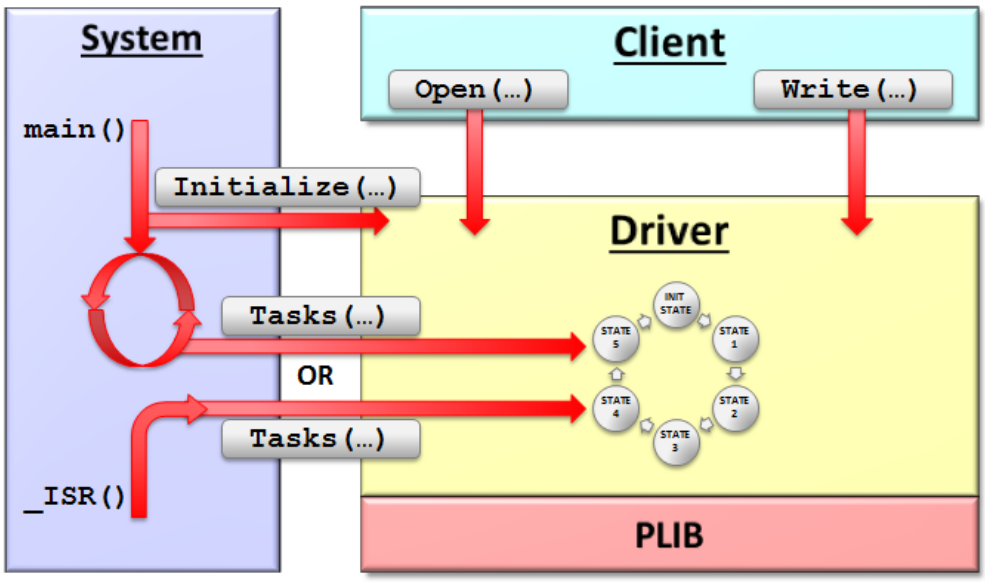
MPLAB Harmony device drivers (usually referred to as "drivers") provide simple, highly abstracted C-language interfaces to peripherals and other resources. A driver's interface allows applications and other client modules to easily interact with the peripheral it controls using consistent usage models. Some functions are similar on all drivers, while other functions are unique to a particular type of driver or peripheral. However, driver interface functions are generally independent of the details of how a given peripheral is implemented on any specific hardware or of how many instances of that peripheral exist in a given system.

Drivers normally utilize MPLAB Harmony Peripheral Libraries (PLIBs) to access and control peripheral hardware that is built into the processor (and is directly addressable by it). However, drivers can also support external peripheral hardware by calling another driver that directly controls a built-in peripheral to which the external peripheral is connected. For example, an SD Card driver may use a SPI driver to access its external SD Card Flash device. A driver may even be completely abstracted away from any hardware (utilizing no peripheral hardware at all), simply controlling some software resource (such as a buffer queue) or providing some service (such as data formatting or encryption). Using this method, driver and other modules may be "stacked" into layers of software, with each responsible for the details of managing its own resources while hiding those details from client modules that use them.

Regardless of the type of peripheral or resource that a MPLAB Harmony driver manages, a driver has the following fundamental responsibilities:

- Provide a common system-level interface to the resource
- Provide a highly abstracted file system style client interface to the resource
- Manage the state of the peripheral or resource
- Manage access to the resource

A driver's system interface can be thought of as being a horizontal interface and its client interface can be thought of as being a vertical interface, as shown in the following block diagram.



The horizontal or "system" interface provides functions to initialize the driver and keep it running. To keep a driver running, a system loop or ISR function (but never both in the same system) calls its state machine "tasks" function repeatedly, as necessary. Therefore, a driver's system interface is normally only called by code that is generated by the MPLAB Harmony Configurator (MHC) when you select and configure the driver. Its purpose is to ensure that the driver works independently (conceptually in the background), providing the capabilities it implements. By contrast, the application (or any other "client" of the driver) normally only interacts with the driver's vertical "client" interface (often thought of as the driver's API). The client interface provides functions to open the driver for use and interact with it, reading or writing data or performing device-type specific operations. The client interface is what allows the application to access the peripheral in a safe and easy way without worrying about the details of the driver or what other clients it may be serving.

The following sections describe in general terms how to use these two interfaces and give specific examples to help illustrate the concepts. The

subsequent help sections for each individual driver describe their specific interfaces in detail; listing all supported functions, parameters, and return values as well as their data types and expected behavior. You may also refer to the MPLAB Harmony Driver Development guide for additional information on MPLAB Harmony drivers and for information on how to develop your own drivers, if needed.

Using a Driver's System Interface

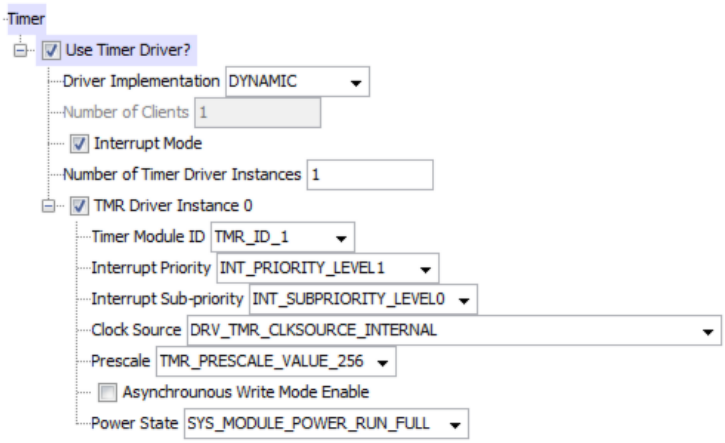
Introduces the System Interface of a MPLAB Harmony device driver and explains its usage.

Description

An MPLAB Harmony driver's system interface provides functions to initialize, deinitialize, and reinitialize an instance of a driver, as well as functions to maintain its state machine (and/or implement its Interrupt Service Routine) and check its current "running" status. Normally, as an MPLAB Harmony application developer or a developer of a "client" module that uses the driver, you will not call the system interface functions directly. The MHC generates calls to the system interface functions of any driver that is used in a project when it generates the system configuration files. Exactly which functions are called and exactly how they're called depends on the configuration options selected in the project's active configuration.

For example, when the box next to "Use Timer Driver?" is selected in the MHC Options tree (within *MPLAB Harmony & Application Configuration* > *Harmony Framework Configuration* > *Drivers* > *Timer*), as shown in the following figure, the MHC will generate all necessary definitions and function calls for the Timer Driver's system interface.

Example Timer Driver MHC Options



These configuration selections, which are set by default once "Use Timer Driver" is selected, will cause the MHC to generate the following definitions in the `system_config.h` header file for the main project's current configuration when **Generate Code** is clicked.

Example Driver Options in system_config.h

```
/*** Timer Driver Configuration ***/
#define DRV_TMR_INTERRUPT_MODE            true
#define DRV_TMR_INSTANCES_NUMBER          1
#define DRV_TMR_CLIENTS_NUMBER            1

/*** Timer Driver 0 Configuration ***/
#define DRV_TMR_PERIPHERAL_ID_IDX0         TMR_ID_1
#define DRV_TMR_INTERRUPT_SOURCE_IDX0      INT_SOURCE_TIMER_1
#define DRV_TMR_CLOCK_SOURCE_IDX0          DRV_TMR_CLKSOURCE_INTERNAL
#define DRV_TMR_PRESCALE_IDX0              TMR_PRESCALE_VALUE_256
#define DRV_TMR_OPERATION_MODE_IDX0        DRV_TMR_OPERATION_MODE_16_BIT
#define DRV_TMR_ASYNC_WRITE_ENABLE_IDX0    false
#define DRV_TMR_POWER_STATE_IDX0           SYS_MODULE_POWER_RUN_FULL
```

It is important to notice that the Driver Implementation selection in the MHC graphical interface does not correlate to a `#define` statement in the `system_config.h` file. Instead, it determines which implementation of the driver this configuration will use. Drivers may have more than one implementation. For example, most drivers have both static and dynamic implementations. A static implementation is usually the smaller of the two, but it is only capable of controlling one instance of a peripheral. An equivalent dynamic implementation will be larger, but it is capable of managing multiple instances of the same type of peripheral using a single instance of the source code (and thus, one instance of the object code). Some drivers may have additional implementations, each one optimized for a different usage. The Driver Implementation pull-down control in the MHC graphical interface allows you to select which implementation the current configuration will use. Normally, you can use only a single implementation of a driver in a given configuration. If you change driver implementations, it changes which implementation is used for all all instances of a peripheral.

The number of instances option, for example, Number of Timer Driver Instances, which correlates to the DRV_TMR_INSTANCES_NUMBER definition, determines how many instances of a static driver implementation will be generated or how many instances of a peripheral a dynamic driver implementation will manage. Drivers may also be designed to allow multiple different clients (applications or other modules) to share the same instance of a peripheral or resource. Therefore, a driver will have an option to determine a maximum number of simultaneous clients that it can support. For example, Number of Clients (DRV_TMR_CLIENTS_NUMBER) in the Timer Driver, which is fixed at one (1) and cannot be

changed, which indicates that the Timer Driver is a single-client driver). The last implementation-specific configuration option in this example is the "Interrupt Mode" (DRV_TMR_INTERRUPT_MODE) setting. This option determines if the implementation is configured to run polled or interrupt driven (discussed further, in a following section). MPLAB Harmony drivers are generally designed to run most effectively in an interrupt-driven configuration, but they can also be run in a polled configuration to simplify debugging or to support task prioritization in an RTOS configuration.

The remaining configuration options are all instance-specific initialization options. For a dynamic implementation of a driver, these options are passed into the driver's Initialize function through an "init" data structure, as shown in the following example.

Example Driver Init Structure in system_init.c

```
const DRV_TMR_INIT drvTmr0InitData =
{
    .moduleInit.sys.powerState = DRV_TMR_POWER_STATE_IDX0,
    .tmrId = DRV_TMR_PERIPHERAL_ID_IDX0,
    .clockSource = DRV_TMR_CLOCK_SOURCE_IDX0,
    .prescale = DRV_TMR_PRESCALE_IDX0,
    .mode = DRV_TMR_OPERATION_MODE_16_BIT,
    .interruptSource = DRV_TMR_INTERRUPT_SOURCE_IDX0,
    .asyncWriteEnable = false,
};
```

The exact meaning and usage of these options are described in the **Configuring the Library** section in the Help documentation for each library. The live MHC Help windowpane displays the associated help section whenever you select one of these options in the options tree.

There is one instance-specific initialization option of which you should take special notice: the peripheral ID option (.tmrId, in the Timer Driver example shown). This initialization option associates the driver instance (a zero-based index number) with the peripheral-hardware instance number, as defined by the data sheet for the processor in use. For a dynamic driver, this association is actually made when the driver's initialize function is called and passes a pointer to the init data structure, as shown in the following code example.

Example Driver Initialize Call in system_init.c

```
/* Initialize Drivers */
sysObj.drvTmr0 = DRV_TMR_Initialize(DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&drvTmr0InitData);
```

In this example, the driver index (DRV_TMR_INDEX_0) is defined as a numeric constant with a value of zero (0). This line of code associates driver instance 0 with hardware timer instance 1 by calling the DRV_TMR_Initialize function from the system initialization code and passing a pointer to the `drvTmr0InitData` structure. As shown earlier, the Timer Driver's init structure contains the value TMR_ID_1 (defined by the timer peripheral library), in its `.tmrId` data member.

In a static implementation, the driver peripheral ID macro (DRV_TMR_PERIPHERAL_ID_IDX0) defined in `system_config.h` is hard-coded into the driver's instance-specific initialization function when it is generated by the MHC, instead of defining an "init" structure, as shown in the following example; however, the effect is the same.

Example Static Driver Initialize Function

```
void DRV_TMR0_Initialize(void)
{
    PLIB_TMR_Stop(DRV_TMR_PERIPHERAL_ID_IDX0);
    PLIB_TMR_ClockSourceSelect(DRV_TMR_PERIPHERAL_ID_IDX0, DRV_TMR_CLOCK_SOURCE_IDX0);
    PLIB_TMR_PrescaleSelect(DRV_TMR_PERIPHERAL_ID_IDX0, DRV_TMR_PRESCALE_IDX0);
    PLIB_TMR_Mode16BitEnable(DRV_TMR_PERIPHERAL_ID_IDX0);
    PLIB_TMR_Counter16BitClear(DRV_TMR_PERIPHERAL_ID_IDX0);
    PLIB_TMR_Period16BitSet(DRV_TMR_PERIPHERAL_ID_IDX0, 0);
}
```

The DRV_TMR0_Initialize function (with an instance number '0' in the name) in the previous example, is a static version of the DRV_TMR_Initialize system interface function. The call to this function is created by the MHC when it generates the system code. Therefore, that call is always generated with the correct name and with the correct instance number in the name. However, when calling client interface functions (open, close, read, write, etc.) from your own applications, you *should not* use an instance number in the function name. Dynamic drivers implement the client interface functions without any index numbers in their names. Instead, they use an index or handle parameter to identify the instance of the driver with which to interact. Also, when using static implementations of the drivers, the dynamic API functions are mapped (using the index or handle parameter) to the appropriate static function with the index number in its name. Therefore, calling the dynamic API function makes your application always portable, using whichever driver instance is configured to the index value with which you open the driver.

Note: Calling the static versions of the interface function (with the index numbers in their names) is not prohibited. However, it will limit the portability of your application.

Understanding this mechanism is critical to understanding how to access the desired peripheral hardware instance. Therefore, it is worth looking at a few demonstration applications to see how it is used. Also, refer to *Volume III: MPLAB Harmony Development > Key Concepts > Key One-to-Many Relationships* for additional information on the concepts of having multiple implementations, instances, and clients.

Something else worth noting about the previous example call to the Timer Driver's initialize functions is that when using a dynamic implementation, it returns a value called an "object handle". In the previous example, that object handle was stored in a system configuration object data member (`sysObj.drvTmr0`). Object handles returned by module initialization functions are stored in a system configuration structure normally named `sysObj`. The definition of this structure is generated in the `system_definitions.h` header file the MHC, as shown in the following example.

Example System Object Data Structure Definition in system_definitions.h

```
typedef struct
{
    SYS_MODULE_OBJ  sysDevcon;
```

```
    SYS_MODULE_OBJ  drvTmr0;

} SYSTEM_OBJECTS;

extern SYSTEM_OBJECTS sysObj;
```

As shown in the previous example, this structure is “extern’d” for use by the other system files. It should not be used by application or library files, only by the system files for a single configuration. The `sysObj` structure is defined (and allocated in memory) by the `system_init.c` file, as shown in the following example.

Example System sysObj Definition in system_init.c

```
/* Structure to hold the object handles for the modules in the system. */
SYSTEM_OBJECTS sysObj;
```

For this discussion, you can ignore the `sysDevcon` member of the SYSTEM_OBJECTS structure as it will contain the handle for a different library. The important thing to note is that the `drvTmr0` member must be passed into the Timer Driver’s other system interface functions so that the driver has access to the data it needs manage that specific instance of itself (and the associated peripheral hardware), as shown by the following timer ISR example.

Example Timer ISR in system_interrupt.c

```
void __ISR(_TIMER_1_VECTOR, ipl1AUTO) IntHandlerDrvTmrInstance0(void)
{
    DRV_TMR_Tasks(sysObj.drvTmr0);
}
```

In this ISR example, there are three important things to notice.

First, the ISR function itself is associated with a specific vector through the `__ISR` macro. Different interrupt vectors are associated with different peripheral instances and interrupts on different processors. That is why MPLAB Harmony ISR vector functions are generated in the configuration-specific `system_interrupt.c` file instead of being part of the driver library itself.

Second, the DRV_TMR_Tasks function implements the actual ISR logic of the TMR driver. Most MPLAB Harmony drivers are designed to run interrupt driven and their tasks functions implement the software state machine logic necessary to keep the driver’s interrupt sequence moving from one interrupt to the next until the driver’s task is complete.

Third, the `sysObj.drvTmr0` object handle’s value is passed into the driver’s tasks function so that it has access to the data it requires to control instance zero (0) of the Timer Driver and its associated hardware instance, which must match the ISR vector instance from which it is called.

By default, the Timer Driver is configured to run interrupt-driven, as shown previously. This is not necessarily true for all drivers. However, most drivers (including the Timer Driver) can run in a Polled mode by simply changing the configuration settings. For example, by clearing the "Interrupt Mode" option in the MHC configuration tree and regenerating the configuration code, the previous example ISR will be removed from `system_interrupt.c` and a call to the Timer Driver’s tasks function will be added to the polled system tasks function, as shown by the following `system_tasks.c` example code.

Example Call to Timer Tasks from system_tasks.c

```
void SYS_Tasks ( void )
{
    /* Maintain system services */
    SYS_DEVCON_Tasks(sysObj.sysDevcon);

    /* Maintain Device Drivers */
    DRV_TMR_Tasks(sysObj.drvTmr0);

    /* Maintain the application's state machine. */
    APP_Tasks();
}
```

In this example, the Timer Driver’s tasks function is called from the polled loop in main by the SYS_Tasks function. The driver’s tasks must still receive the `sysObj.drvTmr0` object handle value and its logic operates in exactly the same way, with one exception. Because the driver is now polled, the DRV_TMR_INTERRUPT_MODE option is now defined as false. This causes the driver to be built so that it does not enable its own interrupt, allowing it to run in the polled loop and to not require an ISR.

For additional information on the device driver system interface, refer to *Volume III: MPLAB Harmony Development > MPLAB Harmony Driver Development Guide > System Interface* and to the documentation for the individual system interface functions for the driver in question.

Using a Driver's Client Interface

Introduces the Client Interface (or API) of a MPLAB Harmony device driver and explains common usage models.

Description

Applications (or any other “client” of a MPLAB Harmony device driver) normally only interact with the driver’s client interface (often called its API). The client interface provides functions to “open” the driver (creating a link between the client and the driver) and interact with it, to transfer data or perform operations that are specific to a given type of device, and to “close” the driver (releasing the link). Once a driver has been configured and the configuration code has been generated, the application can assume that the driver will be initialized by the system-wide initialization function (SYS_Initialize) and that its tasks functions will be called as required from either the system-wide tasks function (SYS_Tasks) or from the appropriate ISR, depending upon how the driver was designed and configured.

To interact with the driver, a client must first call the driver's open function. This is necessary because all other client interface functions require a "handle" to the device driver that is returned by the open function, as shown in the following example.

Example Call to a Driver's Open Function

```
appData.handleTmr = DRV_TMR_Open(APP_TMR_DRV_INDEX, DRV_IO_INTENT_EXCLUSIVE);
if( DRV_HANDLE_INVALID != appData.handleTmr )
{
    // Advance to next application state.
}
```

In this example, the first parameter to the DRV_TMR_Open function is the APP_TMR_DRV_INDEX macro, which is a constant defined to the value of the desired driver instance index number in the `system_config.h` header file. This value must be the same as the index number used when the desired driver was initialized (as shown in the previous section). This is how the client becomes associated with a specific instance of a driver.

The second parameter identifies how the client intends to use the driver. Here, the client wants to have exclusive access to the driver. This means that no other client can currently have an active handle to this driver or this call will fail and return a value of DRV_HANDLE_INVALID. Drivers can also be opened as shared, as blocking or non-blocking and for reading, writing, or both. Refer to the help for the DRV_IO_INTENT data type for additional information about the IO intent parameter of driver open functions. This parameter is merely an advisory parameter. How it is used by the driver is implementation dependent and will be described in the driver's help documentation.

Finally, if the open function was successful, the returned value will be a valid handle to the driver instance. This value is opaque and meaningless to the caller, but it must be passed back to the driver as the first parameter to every other client interface function provided by the driver. A valid handle identifies both the instance of the driver with which the caller interacts and it identifies the client performing the call. This means that, two different client applications or modules opening the same driver in the same system at the same time will receive different values for their "opened" handle. If, for any reason, the driver cannot support the "open" request (it is not finished initializing itself, it has already been opened for exclusive access, or cannot accept new open requests for any reason), it will return a value of DRV_HANDLE_INVALID, indicating the client cannot use it at this time. The DRV_HANDLE_INVALID value is the only non-opaque value that a client should consider meaningful. All other values are only meaningful to the driver that provided them.

Note: The `appData.handleTmr` variable in the previous example is a member of the application's `appData` structure. This structure is generated by the MHC as part of the initial application template and should be used to hold an applications state variables.

When the client is finished using a driver, it may close it, as shown in the following example.

Example Call to a Driver's Close Function

```
DRV_TMR_Close(appData.handleTmr);
```

This action releases the link to the driver, invalidating the handle and releasing any resources allocated by the driver to track requests from the client. Notice that the close function demonstrates the use of the driver handle, requiring it as a parameter. However, after the close function returns, the handle value cannot be used again. Therefore, the client should not call the driver's close function until it is done using the driver or it will have to call open again and obtain a new handle to use the driver again. In fact, since many embedded applications are always running, they often do not bother to close drivers they use. But, applications that can go idle or that can be stopped and restarted or that need to share a driver with other clients, but want to conserve resources, or that want use the driver exclusively, can close a driver when they are finished with it for a time and reopen it later when needed. In fact, this is a good way to share a single-client driver, or a driver that supports exclusive access, allowing each client to open it and use it only when a valid handle is obtained.

Using a Driver in an Application

Describes how to write a state-machine based application that uses a MPLAB Harmony driver.

Description

MPLAB Harmony generally treats all software modules, including applications, as state machines that have an "initialize" function and a "tasks" function. In fact, when not using a RTOS, it essentially treats the entire system as one large state machine that runs in a common super loop in the "main" function, as shown in the following code example.

Example Main Function

```
int main ( void )
{
    SYS_Initialize(NULL);

    while(true)
    {
        SYS_Tasks();
    }

    return (EXIT_FAILURE);
}
```

For the purpose of this discussion, it is important to understand that the application's APP_Initialize function is called from the SYS_Initialize function, along with the initialization of functions of all drivers and other libraries before execution enters the endless `while(true)` super loop that continuously calls the system-wide SYS_Tasks function. The application's APP_Tasks function is then called from the SYS_Tasks function inside of the super loop, along with all other polled modules in the system. If you are not already familiar with the organization of an MPLAB

Harmony project, please refer to *Volume I: Getting Started With MPLAB Harmony > What is MPLAB Harmony?* for more information.

An application that uses a driver must define a DRV_HANDLE variable, as shown in the following example application header file.

Example Driver Application Header (app.h)

```
#include "driver/usart/drv_usart.h"

typedef enum
{
    APP_STATE_SETUP=0,
    APP_STATE_MESSAGE_SEND,
    APP_STATE_MESSAGE_WAIT,
    APP_STATE_DONE

} APP_STATES;


typedef struct
{
    APP_STATES  state;
    DRV_HANDLE  usart;
    char *      message;

} APP_DATA;
```

In this previous example, the driver handle variable is named `usart`. To keep the application well organized, it is common to keep all of the application's state variables (including one called "state" that holds the current state of the application's state machine) in a common structure (APP_DATA). This structure must be allocated in the application's source file (usually named `app.c`) and initialized by the application's initialization function, as shown in the following example.

Example Driver Application Initialization

```
APP_DATA appData;

void APP_Initialize ( void )
{
    /* Place the App in its initial state. */
    appData.state   = APP_STATE_SETUP;
    appData.usart   = DRV_HANDLE_INVALID;
    appData.message = "Hello World\n";
}
```

The APP_Initialze function must initialize the state variable (appData.state) to put the application's state machine in its initial state (the APP_STATE_SETUP value from the APP_STATES enumeration). It must also initialize the driver-handle variable (`appData.usart`), so that the state machine knows it is not yet valid, and any other application variables (like the string pointer, `appData.message`).

Once the application's data structure has been initialized, it is safe for the system (the main and SYS_Tasks functions) to call the application's APP_Tasks function from the super loop to keep it running. The APP_Tasks function then executes state transition code as it switches between states, as demonstrated by the following example.

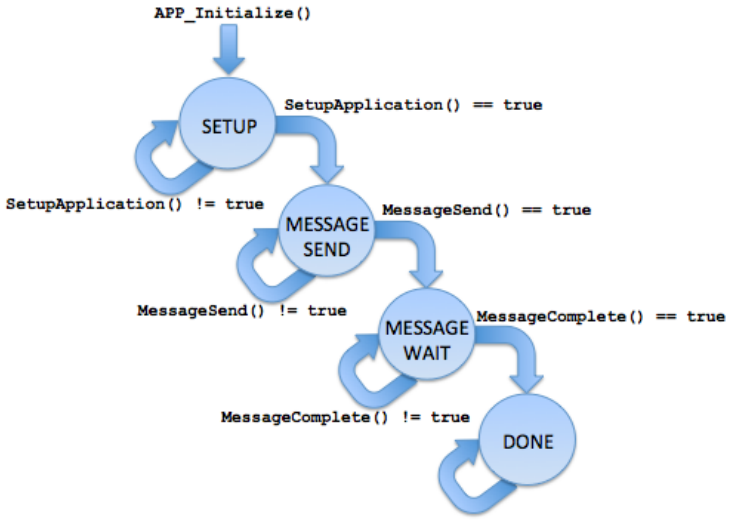
Example Application State Machine Using a Driver

```
void APP_Tasks ( void )
{
    switch ( appData.state )
    {
        case APP_STATE_SETUP:
        {
            if (SetupApplication() == true)
            {
                appData.state = APP_STATE_MESSAGE_SEND;
            }
            break;
        }

        case APP_STATE_MESSAGE_SEND:
        {
            if (MessageSend() == true)
            {
                appData.state = APP_STATE_MESSAGE_WAIT;
            }
            break;
        }

        case APP_STATE_MESSAGE_WAIT:
        {
```

```
            if (MessageComplete() == true)
            {
                appData.state = APP_STATE_DONE;
            }
            break;
        }

        case APP_STATE_DONE:
        default:
        {
            break;
        }
    }
}
```

There are numerous ways to implement a state machine. However, in this example, the application changes state when the APP_Tasks function assigns a new value from the APP_STATES enumeration to the `appData.states` variable. This happens when one of the state transition function returns true. The end result is an overall application state machine execution that retries each state transition until it succeeds before moving on to the next state, as shown in the following diagram.

Application State Machine



Note: The APP_STATE_ prefix and all inter-word underscores were removed from the state names to simplify the diagram.

After APP_Initialize places the state machine in its initial APP_STATE_SETUP state, the APP_Tasks function will call the SetupApplication function when it is called. When SetupApplication returns true indicating it has completed its task, the state machine advances to the next state. Otherwise, it stays in the same state and retries the tasks in the SetupApplication function. This pattern repeats for the APP_STATE_MESSAGE_SEND state and the `MessageSend` function as well as the APP_STATE_MESSAGE_WAIT state and the `MessageComplete` function. When all functions have returned true, the state machine to transitions to the APP_STATE_DONE state where it unconditionally stays having completed its tasks.

The sum total of the tasks performed by each transition function completes the overall task of the application. For an application that uses a driver like this example, this includes opening the driver, sending the message, and closing the driver when the message has been sent. How each individual transition function in this example application accomplishes its portion of the overall task, is described in the examples in the following sections to demonstrate how drivers are commonly used.

Opening a Driver

Describes how to open a driver in a state-machine based application.

Description

To use a MPLAB Harmony driver, an application (or other client) must call the driver's "open" function and obtain a valid handle to it, as shown by the following code example.

Example Opening a Driver

```
static bool SetupApplication ( void )
{
    if (appData.usart == DRV_HANDLE_INVALID)
    {
```

```
        appData.usart = DRV_USART_Open(APP_USART_DRIVER_INDEX,
                                       (DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NONBLOCKING));
    }

    if (appData.usart == DRV_HANDLE_INVALID)
    {
        return false;
    }

    return true;
}
```

This example demonstrates the implementation of a state-transition function in a state machine-based application (as shown in the previous Using a Driver in an Application section). The `SetupApplication` function assumes that the `appData.usart` variable has been initialized to a value of DRV_HANDLE_INVALID when the application's state machine was initialized. Therefore, it checks this variable every time it is called to see if it has already completed its task. If `appData.usart` contains a value of DRV_HANDLE_INVALID, this indicates that the driver has not yet been successfully opened, causing the function to attempt to open the driver by calling DRV_USART_Open.

If the USART driver is ready and able to support a new client it will return a valid handle. If it is not ready or able to accept a new client, the driver will return DRV_HANDLE_INVALID and the `SetupApplication` function will return false and the application will stay in the same state and try to open the driver again the next time its state machine tasks function is called. When DRV_USART_Open returns a valid handle (a handle that is not equal to DRV_HANDLE_INVALID), the `SetupApplication` function returns true, allowing the application's state machine to advance.

This technique allows the application to try repeatedly to open the driver until it succeeds and guarantees that the application's state machine will not advance until it has done so. A more sophisticated application might use a time-out mechanism or some other error handling logic to take alternative action if it cannot open the driver in an acceptable time period. However, this simple implementation demonstrates the basic concept of how an MPLAB Harmony application (or any other client module) can safely open a driver before attempting to use it.

Using Driver Interface Functions

Describes how to use a device driver's synchronous client interface functions, such as those that read and write data.

Description

To use a MPLAB Harmony driver's client interface, the application must first obtain a valid handle from the driver's "open" function. The examples in this section assume that that has already occurred and that the value of the USART driver handle in the appData.usart variable is valid. The following example code demonstrates the implementation of a state transition function in a state machine-based application (as shown in the previous Using a Driver in an Application section) that writes data to a USART driver for transmission on the associated USART peripheral.

Example Writing Data To a Driver

```
static bool MessageSend ( void )
{
    size_t  count;
    size_t  length = strlen(appData.message);

    count = DRV_USART_Write(appData.usart, appData.message, length);

    appData.message += count;

    if (count == length)
    {
        return true;
    }

    return false;
}
```

In this example, the appData.message variable is a char pointer pointing to a null-terminated C-language string that was defined and initialized, as shown in the Using a Driver in an Application section. When `MessageSend` function is first called by the application's state machine, it points to the first character in the string to be transmitted. The function calculates the current length of the message string (using the standard C-language strlen function) and calls the driver's DRV_USART_Write function, passing it the valid driver handle (`appData.usart`) along with the pointer to the message string and its length, to transmit the message string on the associated USART.

If the driver is configured for blocking, the DRV_USART_Write function will not return until it has processed all of the data in the message string. However, that usually requires the use of a RTOS. Normally, in a bare-metal system (one that does not use a RTOS), MPLAB Harmony drivers are used in a non-blocking mode. In that case, a driver will perform as much of a task as it can when one of its interface functions is called without blocking. This means that the function will then return immediately, not waiting for the task to complete, and provide information on how much of the task was completed so the client can react appropriately. In this example, the DRV_USART_Write function will return a count of the number of bytes that were processed by the USART driver by this call to the function.

The `MessageSend` function captures the number of bytes processed by the DRV_USART_Write function in a local count variable. It then effectively removes those bytes from the message string by incrementing the pointer by count bytes (`appData.message` is a char pointer that increments by the size of one byte for every '1' added to it). Then, the `MessageSend` function checks to see if it was able to write the entire string by comparing the value of count to the value of length that it calculated before calling the driver's write function. If the two are equal, the task is

complete and the MessageSend function returns true and the application's state machine can continue to the next state. If the two values are not equal, this indicates there are remaining bytes in the message string. The MessageSend function returns false and the application must stay in the same state so that the function can attempt to send the remaining bytes next time it is called. A driver only accepts data when it can process it; therefore, the client can call its data transfer function as many times as necessary, even when the function returns bytes processed if it cannot accept more data at that time.

When a client has called a driver interface function there are really only two possibilities. Either the operation has completed when the function returns, or the operation continues after the function has returned. If the operation completes immediately, the client can continue on without taking further action. However, in this example, while the USART driver may have accepted some of the bytes in the message string (perhaps copying them to an internal hardware or software FIFO buffer), it still takes some time to transmit the data over the USART peripheral. In many cases the client may need to know when the operation has actually completed. For this reason, most drivers provide one or more status functions that client applications may call to determine the current status of an operation, as demonstrated in the following example.

Example Using a Driver Status Function

```
static bool MessageComplete ( void )
{
    if (DRV_USART_ClientStatus(appData.usart) == DRV_USART_CLIENT_STATUS_BUSY)
    {
        return false;
    }
    return true;
}
```

This example extends the previous one and assumes that the MessageSend function has returned true and the application has moved to a new state where it calls this function to determine when the driver is idle, which indicates that the message has been completely transmitted. To do that, the MessageComplete function calls the DRV_USART_ClientStatus function. If its return value is DRV_USART_CLIENT_STATUS_BUSY, the USART driver is still working on a previous request by the client. If any other status value is returned, this indicates that the driver is no longer busy with a current request and the MessageComplete function returns true so that the client application's state machine can move on. A more sophisticated example would check for other possible status values that might indicate some error has occurred and take appropriate action. However, this example is sufficient to demonstrate the concept of checking a driver status function to determine when it is safe to move to another state.

Since the client application stays in the same state calling the status function each time its tasks function is called until the desired status is returned, it is effectively polling the status as if it were in a while loop. In fact, it is in the system-wide while loop. However, by not trapping the CPU within its own internal while loop, the application allows other modules (including, potentially, the driver it is using) to continue running and servicing requests. Failing to allow the rest of the system to run can result in a deadlock where the polling application is waiting for a status; however, the driver it is polling will never be able to provide the expected status, as the driver's own tasks function is not allowed to run. This is why it is important to use the technique described here to "poll" status from modules outside of the current module.

Using Asynchronous and Callback Functions

Describes how to use an asynchronous interface function to start a driver operation and receive a callback when the operation is complete.

Description

When a client calls a function that is part of an asynchronous interface, the function starts the request and returns immediately, without finishing the request. The client can then either poll a status function to determine when the request has finished (as demonstrated in the Using Driver Interface Functions section) or it can utilize a callback function to receive a notification from the driver when the request has finished. So, the difference between an asynchronous interface and a synchronous interface is that a synchronous interface may finish all or part of the request before returning, whereas an asynchronous interface will always return immediately having only started the request. Determination of when the request has completed is handled separately.

The examples in this section reimplement some of the code from the example application described in the previous sections to demonstrate how to use asynchronous queuing and callback interfaces instead of the synchronous status-polling interface demonstrated in the Using Driver Interface Functions section. To use an asynchronous interface, we will first add a couple of new variables to our example application's data structure, as shown by the following structure definition.

Example Driver Application Header (app.h)

```
typedef struct
{
    APP_STATES              state;
    DRV_HANDLE              usart;
    char *                  message;
    DRV_USART_BUFFER_HANDLE messageHandle;
    bool                    messageDone;

} APP_DATA;
```

The state, usart, and message members of the APP_DATA structure are used in exactly the same way as they were in the previous examples. The messageHandle variable will be explained later and the messageDone variable is a Boolean flag used by the callback function to indicate to the application's state machine that the message has been completely processed by the driver. Using these new mechanisms results in very minor changes to the application's state machine, as shown in the following example APP_Initialize and APP_Tasks implementations.

Example Driver Application State Machine (app.c)

```
void APP_Initialize ( void )
{
    appData.state           = APP_STATE_SETUP;
    appData.usart           = DRV_HANDLE_INVALID;
    appData.message         = APP_MESSAGE;
    appData.messageHandle   = DRV_USART_BUFFER_HANDLE_INVALID;
}

void APP_Tasks ( void )
{
    switch ( appData.state )
    {
        case APP_STATE_SETUP:
        {
            if (SetupApplication() == true)
            {
                appData.state = APP_STATE_MESSAGE_SEND;
            }
            break;
        }

        case APP_STATE_MESSAGE_SEND:
        {
            if (MessageSend() == true)
            {
                appData.state = APP_STATE_MESSAGE_WAIT;
            }
            break;
        }

        case APP_STATE_MESSAGE_WAIT:
        {
            if (appData.messageDone)
            {
                DRV_USART_Close(appData.usart);
                appData.state = APP_STATE_DONE;
            }
            break;
        }

        case APP_STATE_DONE:
        default:
        {
            break;
        }
    }
}
```

As described previously, the `SetupApplication` state transition function opens the USART driver and the `MessageSend` function sends the message to it. However, there is no need for a `MessageComplete` state transition function. Instead, the application must implement a callback function that will set the `appData.messageDone` Boolean flag when the driver calls the application "back" to indicate that the message has been sent.

Note: The `AppInitialize` function initializes the `state`, `usart`, and `message` members of the `appData` structure as previously described. And, it also initializes the `messageHandle` member with an invalid value to indicate that the message has not yet been sent. However, it does not initialize the `messageDone` flag because it is more appropriate to clear the flag elsewhere, immediately before calling the driver to send the message.

To use a callback mechanism requires the client to implement and register a callback function. A client must register this function after opening the driver, but prior to calling the driver to initiate the operation. This is often done in the same state transition that opens the driver, as shown in the following `SetupApplication` example.

Example Registering a Driver Callback Function

```
static void BufferDone ( DRV_USART_BUFFER_EVENT  event,
                         DRV_USART_BUFFER_HANDLE bufferHandle,
                         uintptr_t context )
{
    APP_DATA *pAppData = (APP_DATA *)context;

    if (event == DRV_USART_BUFFER_EVENT_COMPLETE)
```

```
    {
        if (bufferHandle == pAppData->messageHandle)
        {
            pAppData->messageDone = true;
            return;
        }
    }

    /* Error */
    return;
}


static bool SetupApplication ( void )
{
    if (appData.usart == DRV_HANDLE_INVALID)
    {
        appData.usart = DRV_USART_Open(APP_USART_DRIVER_INDEX,
                                       (DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NONBLOCKING));
    }

    if (appData.usart == DRV_HANDLE_INVALID)
    {
        return false;
    }

    DRV_USART_BufferEventHandlerSet(appData.usart, BufferDone, (uintptr_t)&appData);
    return true;
}
```

This code block implements both the `BufferDone` callback function and the application's `SetupApplication` state transition function. After successfully opening the driver, the `SetupApplication` function calls the DRV_USART_BufferEventHandlerSet function and passes it the driver handle (`appData.usart`) once it is valid, along with the address of the `BufferDone` callback function and a context value.

The context value can be anything that will fit in an integer large enough to hold a pointer (it is a `uintptr_t` variable). However, this parameter is most commonly used to pass a pointer to the caller's own data structure as demonstrated here (even though it is not strictly necessary). This is done primarily to support multi-instance clients. (Refer to *Volume III: MPLAB Harmony Development > Key Concepts* for information on multiple instances.) A multi-instance client is designed to manage multiple instances of itself by allocating multiple instances of its own data structure, but only one instance of its object code. Passing a pointer to the data structure in the context variable identifies the specific instance that was used when calling the driver.

Once the callback function has been registered with the driver, the application can transition to a state where it attempts to initiate an asynchronous operation. The following example demonstrates the use of a buffer-queuing write function to transmit a message over the USART.

Example Queuing a Buffer to a Driver

```
static bool MessageSend ( void )
{
    appData.messageDone = false;
    DRV_USART_BufferAddWrite(appData.usart, &appData.messageHandle,
                             appData.message, strlen(appData.message));

    if (appData.messageHandle == DRV_USART_BUFFER_HANDLE_INVALID)
    {
        return false;
    }

    return true;
}
```

Before attempting to send the message, this implementation of the `MessageSend` state transition function clears the `appData.messageDone` flag so it can detect when the message has completed. Then, it calls the DRV_USART_BufferAddWrite function to queue up the buffer containing the message to be transmitted by the USART driver. To that function, it passes the USART driver handle (`appData.usart`), the address of the `appData.messageHandle` variable, the pointer to the message buffer (`appData.message`), and the size of the buffer in bytes as calculated by the `strlen` function. The USART driver then adds this buffer to its internal queue of buffers to transmit and provides a handle to the caller that identifies that buffer's place in the queue by storing it to the `appData.messageHandle` variable.

If, for some reason, the driver is unable to successfully queue up the buffer (perhaps the queue is full), it will assign a value of DRV_USART_BUFFER_HANDLE_INVALID to the `appData.messageHandle` variable. If that happens, the `MessageSend` function returns false and the application will stay in the same state and retry the operation again next time its tasks function is called. But, if the operation succeeds, the application advances to the next state.

Once the driver completes the operation, it will call the client's callback function. As shown in the `BufferDone` code example, the driver passes it an enumeration value that identifies which event has just occurred (the `DRV_USART_BUFFER_EVENT_COMPLETE` value) in the event parameter. It also passes it the handle of the buffer that has just completed (`bufferHandle`). The client can use the `bufferHandle` value to verify that it

matches the value stored in the appData.bufferHandle variable to uniquely identify an individual buffer. This is very useful when a client queues up multiple buffers at the same, which is being shown in this example as a demonstration.

The context parameter to the BufferDone function contains a pointer to the application's global (appData) data structure. (This is the same value that was passed in the context parameter to the DRV_USART_BufferEventHandlerSet function.) While not strictly necessary in this example, it is very useful for multi-instance clients such as dynamic device drivers and middleware to identify which instance of the client requested the operation. The callback function simply casts the context value back into a pointer to the client's own data structure's data type (APP_DATA in this example) and uses it to access the structure members. (Again, please refer to *Volume III: MPLAB Harmony Development > Key Concepts* for information on multiple instances.)

The callback function uses the event parameter to identify why the callback occurred. If it was called to indicate that the buffer has been processed, the event parameter will contain the value DRV_USART_BUFFER_EVENT_COMPLETE. If it contains any other value an error has occurred. The BufferDone callback also checks to verify that the buffer that completed was the same buffer that it queued up by comparing the bufferHandle value it was passed with the value assigned to the appData.messageHandle variable when the application called DRV_USART_BufferAddWrite. It accesses the message handle value it saved using the pAppData pointer given to it through the context parameter just. Once it has verified that the buffer it queued has completed, it sets the pAppData->messageDone flag to notify the application's state machine and execution returns to the driver.

Note: It is important to understand that the MessageDone callback function executes in the context of the driver, not the application. Depending on how the system is configured, this means that it may be called from within the driver's ISR context or from another thread context if using a RTOS.

In this example, the APP_Tasks application state machine function is essentially the same as the state machine for the synchronous example. The only difference is that when the application is in the APP_STATE_MESSAGE_WAIT state, it checks the appData.messageDone flag to determine when to close the driver and transition to the APP_STATE_DONE state instead of calling a transition function. (It could still do this in a state transition function, but it was done differently in this example to emphasize the concept.)

The advantage of using an asynchronous interface over a synchronous one is that it allows the client's state machine to continue on, potentially doing something else while the requested operation completes. Whereas a synchronous interface has the possibility of blocking the client's state machine until the operation finishes (when used in a RTOS configuration). An asynchronous interface will always return immediately without blocking (whether a RTOS is used or not). Because of this, most asynchronous interfaces will also allow queuing of more than one operation at a time. This allows client applications to keep a driver continuously busy by keeping the driver's queue full, maximizing data throughput or operation speed. By contrast, a synchronous interface requires one operation to complete before the synchronous function can be called again to cause the next one to begin.

The cost of this capability is that an asynchronous interface has the added complexity of a callback function (if the client cares when the operation finishes) and the fact that a callback function may be called from within the driver's ISR context, depending on how the driver was designed and configured. This fact generally restricts what can be done within the callback function. For example, it is usually a bad idea to perform lengthy processing within a callback function as it will block all lower priority ISRs (as well as the main loop or other threads) while that processing occurs. Also, it is usually best to not call back into the driver's own interface functions unless those functions are documented as being safe to call from within the driver's callback context. Many interface functions (particularly data transfer and data queuing functions) must use semaphores or mutexes to protect their internal data structures in RTOS environments and those constructs cannot be used from within an ISR.

It is also important to not make non-atomic (read-modify-write) accesses to the client's own state data from within the callback function, as the client cannot protect itself against an interrupt that is owned by the driver. That is why a separate Boolean flag variable is commonly used to indicate to the client that the callback has occurred. Most other processing should occur in the client's state machine. It is usually best to simply capture the event and return as quickly as possible from the callback function and let the application's state machine tasks function perform any lengthy processing or calling back into the driver.

Please refer to *Volume III: MPLAB Harmony Development* for additional information.

Library Interface

Constants

	Name	Description
	DRV_CONFIG_NOT_SUPPORTED	Not supported configuration.
	DRV_HANDLE_INVALID	Invalid device handle.
	DRV_IO_ISBLOCKING	Returns if the I/O intent provided is blocking
	DRV_IO_ISEXCLUSIVE	Returns if the I/O intent provided is non-blocking.
	DRV_IO_ISNONBLOCKING	Returns if the I/O intent provided is non-blocking.
	_DRV_COMMON_H	This is macro _DRV_COMMON_H.
	_PLIB_UNSUPPORTED	Abstracts the use of the unsupported attribute defined by the compiler.

Data Types

	Name	Description
	DRV_CLIENT_STATUS	Identifies the current status/state of a client's connection to a driver.
	DRV_HANDLE	Handle to an opened device driver.
	DRV_IO_BUFFER_TYPES	Identifies to which buffer a device operation will apply.

	DRV_IO_INTENT	Identifies the intended usage of the device when it is opened.

Description

Data Types

DRV_CLIENT_STATUS Enumeration

Identifies the current status/state of a client's connection to a driver.

File

driver_common.h

C

```
typedef enum {
  DRV_CLIENT_STATUS_ERROR_EXTENDED = -10,
  DRV_CLIENT_STATUS_ERROR = -1,
  DRV_CLIENT_STATUS_CLOSED = 0,
  DRV_CLIENT_STATUS_BUSY = 1,
  DRV_CLIENT_STATUS_READY = 2,
  DRV_CLIENT_STATUS_READY_EXTENDED = 10
} DRV_CLIENT_STATUS;
```

Members

Members	Description
DRV_CLIENT_STATUS_ERROR_EXTENDED = -10	Indicates that a driver-specific error has occurred.
DRV_CLIENT_STATUS_ERROR = -1	An unspecified error has occurred.
DRV_CLIENT_STATUS_CLOSED = 0	The driver is closed, no operations for this client are ongoing, and/or the given handle is invalid.
DRV_CLIENT_STATUS_BUSY = 1	The driver is currently busy and cannot start additional operations.
DRV_CLIENT_STATUS_READY = 2	The module is running and ready for additional operations
DRV_CLIENT_STATUS_READY_EXTENDED = 10	Indicates that the module is in a driver-specific ready/run state.

Description

Driver Client Status

This enumeration identifies the current status/state of a client's link to a driver.

Remarks

The enumeration used as the return type for the client-level status routines defined by each device driver or system module (for example, DRV_USART_ClientStatus) must be based on the values in this enumeration.

DRV_HANDLE Type

Handle to an opened device driver.

File

driver_common.h

C

```
typedef uintptr_t DRV_HANDLE;
```

Description

Device Handle

This handle identifies the open instance of a device driver. It must be passed to all other driver routines (except the initialization, deinitialization, or power routines) to identify the caller.

Remarks

Every application or module that wants to use a driver must first call the driver's open routine. This is the only routine that is absolutely required for

every driver.

If a driver is unable to allow an additional module to use it, it must then return the special value DRV_HANDLE_INVALID. Callers should check the handle returned for this value to ensure this value was not returned before attempting to call any other driver routines using the handle.

DRV_IO_BUFFER_TYPES Enumeration

Identifies to which buffer a device operation will apply.

File

driver_common.h

C

```
typedef enum {
  DRV_IO_BUFFER_TYPE_NONE = 0x00,
  DRV_IO_BUFFER_TYPE_READ = 0x01,
  DRV_IO_BUFFER_TYPE_WRITE = 0x02,
  DRV_IO_BUFFER_TYPE_RW = DRV_IO_BUFFER_TYPE_READ|DRV_IO_BUFFER_TYPE_WRITE
} DRV_IO_BUFFER_TYPES;
```

Members

Members	Description
DRV_IO_BUFFER_TYPE_NONE = 0x00	Operation does not apply to any buffer
DRV_IO_BUFFER_TYPE_READ = 0x01	Operation applies to read buffer
DRV_IO_BUFFER_TYPE_WRITE = 0x02	Operation applies to write buffer
DRV_IO_BUFFER_TYPE_RW = DRV_IO_BUFFER_TYPE_READ|DRV_IO_BUFFER_TYPE_WRITE	Operation applies to both read and write buffers

Description

Device Driver IO Buffer Identifier

This enumeration identifies to which buffer (read, write, both, or neither) a device operation will apply. This is used for "flush" (or similar) operations.

DRV_IO_INTENT Enumeration

Identifies the intended usage of the device when it is opened.

File

driver_common.h

C

```
typedef enum {
  DRV_IO_INTENT_READ,
  DRV_IO_INTENT_WRITE,
  DRV_IO_INTENT_READWRITE,
  DRV_IO_INTENT_BLOCKING,
  DRV_IO_INTENT_NONBLOCKING,
  DRV_IO_INTENT_EXCLUSIVE,
  DRV_IO_INTENT_SHARED
} DRV_IO_INTENT;
```

Members

Members	Description
DRV_IO_INTENT_READ	Read
DRV_IO_INTENT_WRITE	Write
DRV_IO_INTENT_READWRITE	Read and Write
DRV_IO_INTENT_BLOCKING	The driver will block and will return when the operation is complete
DRV_IO_INTENT_NONBLOCKING	The driver will return immediately
DRV_IO_INTENT_EXCLUSIVE	The driver will support only one client at a time
DRV_IO_INTENT_SHARED	The driver will support multiple clients at a time

Description

Device Driver I/O Intent

This enumeration identifies the intended usage of the device when the caller opens the device. It identifies the desired behavior of the device

driver for the following:

- Blocking or non-blocking I/O behavior (do I/O calls such as read and write block until the operation is finished or do they return immediately and require the caller to call another routine to check the status of the operation)
- Support reading and/or writing of data from/to the device
- Identify the buffering behavior (sometimes called "double buffering" of the driver. Indicates if the driver should maintain its own read/write buffers and copy data to/from these buffers to/from the caller's buffers.
- Identify the DMA behavior of the peripheral

Remarks

The buffer allocation method is not identified by this enumeration. Buffers can be allocated statically at build time, dynamically at run-time, or even allocated by the caller and passed to the driver for its own usage if a driver-specific routine is provided for such. This choice is left to the design of the individual driver and is considered part of its interface.

These values can be considered "flags". One selection from each of the groups below can be ORed together to create the complete value passed to the driver's open routine.

Constants

DRV_CONFIG_NOT_SUPPORTED Macro

Not supported configuration.

File

driver_common.h

C

```
#define DRV_CONFIG_NOT_SUPPORTED (((unsigned short) -1))
```

Description

Not supported configuration

If the configuration option is not supported on an instance of the peripheral, use this macro to equate to that configuration. This option should be listed as a possible value in the description of that configuration option.

DRV_HANDLE_INVALID Macro

Invalid device handle.

File

driver_common.h

C

```
#define DRV_HANDLE_INVALID (((DRV_HANDLE) -1))
```

Description

Invalid Device Handle

If a driver is unable to allow an additional module to use it, it must then return the special value DRV_HANDLE_INVALID. Callers should check the handle returned for this value to ensure this value was not returned before attempting to call any other driver routines using the handle.

Remarks

None.

DRV_IO_ISBLOCKING Macro

Returns if the I/O intent provided is blocking

File

driver_common.h

C

```
#define DRV_IO_ISBLOCKING(intent) (intent & DRV_IO_INTENT_BLOCKING)
```

Description

Device Driver Blocking Status Macro

This macro returns if the I/O intent provided is blocking.

Remarks

None.

DRV_IO_ISEXCLUSIVE Macro

Returns if the I/O intent provided is non-blocking.

File

driver_common.h

C

```
#define DRV_IO_ISEXCLUSIVE(intent) (intent & DRV_IO_INTENT_EXCLUSIVE)
```

Description

Device Driver Exclusive Status Macro

This macro returns if the I/O intent provided is non-blocking.

Remarks

None.

DRV_IO_ISNONBLOCKING Macro

Returns if the I/O intent provided is non-blocking.

File

driver_common.h

C

```
#define DRV_IO_ISNONBLOCKING(intent) (intent & DRV_IO_INTENT_NONBLOCKING )
```

Description

Device Driver Non Blocking Status Macro

This macro returns if the I/ intent provided is non-blocking.

Remarks

None.

_DRV_COMMON_H Macro

File

driver_common.h

C

```
#define _DRV_COMMON_H
```

Description

This is macro _DRV_COMMON_H.

_PLIB_UNSUPPORTED Macro

Abstracts the use of the unsupported attribute defined by the compiler.

File

driver_common.h

C

```
#define _PLIB_UNSUPPORTED
```

Description

Unsupported Attribute Abstraction

This macro nulls the definition of the _PLIB_UNSUPPORTED macro, to support compilation of the drivers for all different variants.

Remarks

None.

Example

void _PLIB_UNSUPPORTED PLIB_USART_Enable(USART_MODULE_ID index);

This function will not generate a compiler error if the interface is not defined for the selected device.

Files

Files

Name	Description
driver.h	This file aggregates all of the driver library interface headers.
driver_common.h	This file defines the common macros and definitions used by the driver definition and implementation headers.

Description

driver.h

This file aggregates all of the driver library interface headers.

Description

Driver Library Interface Header Definitions

Driver Library Interface Header This file aggregates all of the driver library interface headers so client code only needs to include this one single header to obtain prototypes and definitions for the interfaces to all driver libraries. A device driver provides a simple well-defined interface to a hardware peripheral that can be used without operating system support or that can be easily ported to a variety of operating systems. A driver has the fundamental responsibilities:

- Providing a highly abstracted interface to a peripheral
- Controlling access to a peripheral
- Managing the state of a peripheral

Remarks

The directory in which this file resides should be added to the compiler's search path for header files.

File Name

drv.h

Company

Microchip Technology Inc.

driver_common.h

This file defines the common macros and definitions used by the driver definition and implementation headers.

Enumerations

	Name	Description
	DRV_CLIENT_STATUS	Identifies the current status/state of a client's connection to a driver.
	DRV_IO_BUFFER_TYPES	Identifies to which buffer a device operation will apply.
	DRV_IO_INTENT	Identifies the intended usage of the device when it is opened.

Macros

	Name	Description
	_DRV_COMMON_H	This is macro _DRV_COMMON_H.
	_PLIB_UNSUPPORTED	Abstracts the use of the unsupported attribute defined by the compiler.
	DRV_CONFIG_NOT_SUPPORTED	Not supported configuration.
	DRV_HANDLE_INVALID	Invalid device handle.
	DRV_IO_ISBLOCKING	Returns if the I/O intent provided is blocking
	DRV_IO_ISEXCLUSIVE	Returns if the I/O intent provided is non-blocking.
	DRV_IO_ISNONBLOCKING	Returns if the I/O intent provided is non-blocking.

Types

	Name	Description
	DRV_HANDLE	Handle to an opened device driver.

Description

Driver Common Header Definitions

This file defines the common macros and definitions used by the driver definition and the implementation header.

Remarks

The directory in which this file resides should be added to the compiler's search path for header files.

File Name

drv_common.h

Company

Microchip Technology Inc.

ADC Driver Library

This section describes the Analog-to-Digital Converter (ADC) Driver Library.

Introduction

This Analog-to-Digital Converter (ADC) driver provides an interface to manage the ADC module on the Microchip family of microcontrollers.

Description

An ADC is a vital part of any system that interfaces to real-world signals. While there are many techniques for analog-to-digital conversion, the Microchip family of microcontrollers uses Successive Approximation as one of its primary techniques.

Through MHC, this driver provides APIs to interact with the ADC module.

Note: Only Static implementation is supported for the ADC Driver Library.

Library Interface

Functions

	Name	Description
	DRV_ADC_Deinitialize	Deinitializes the DRV_ADC_Initialize driver module **Implementation:** Static
	DRV_ADC_Initialize	Initializes the ADC driver. **Implementation:** Static
	DRV_ADC_SamplesAvailable	Identifies if specified ADC Driver input has any samples available to read. **Implementation:** Static
	DRV_ADC_SamplesRead	Reads the converted sample data from ADC input Data buffer. **Implementation:** Static
	DRV_ADC_Start	Starts the software trigger for the ADC driver sampling and converting analog to digital values. **Implementation:** Static

	DRV_ADC_Stop	Stops the Global Software Level Trigger from continuing triggering for converting ADC data. **Implementation:** Static
	DRV_ADCx_Close	Closes the ADC instance for the specified driver index. **Implementation:** Static
	DRV_ADCx_Open	Opens the ADC instance for the specified driver index. **Implementation:** Static

Description

This section lists the interface routines, data types, constants and macros for the library.

Functions

DRV_ADC_Deinitialize Function

Deinitializes the DRV_ADC_Initialize driver module

Implementation: Static

File

help_drv_adc.h

C

```
void DRV_ADC_Deinitialize();
```

Returns

None.

Description

This function deinitializes the ADC Driver module for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

None.

Preconditions

None.

Function

void DRV_ADC_Deinitialize(void)

DRV_ADC_Initialize Function

Initializes the ADC driver.

Implementation: Static

File

help_drv_adc.h

C

```
void DRV_ADC_Initialize();
```

Returns

None.

Description

This function initializes the ADC Driver module for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

This function must be called before any other ADC function is called. This function should only be called once during system initialization.

Preconditions

None.

Function

void DRV_ADC_Initialize(void)

DRV_ADC_SamplesAvailable Function

Identifies if specified ADC Driver input has any samples available to read.

Implementation: Static

File

help_drv_adc.h

C

```
bool DRV_ADC_SamplesAvailable(uint8_t bufIndex);
```

Returns

- true - When ADC data buffer is available to be read
- false - When ADC data buffer is not available

Description

This function identifies whether the specified ADC Driver input has any samples available to read.

Remarks

None.

Preconditions

The following functions have been called:

- DRV_ADC_Initialize
- DRV_ADCx_Open
- DRV_ADC_Start or other triggered by source setup in MHC

Parameters

Parameters	Description
uint8_t bufIndex	ADC input number (ANx)

Function

bool DRV_ADC_SamplesAvailable(uint8_t bufIndex);

DRV_ADC_SamplesRead Function

Reads the converted sample data from ADC input Data buffer.

Implementation: Static

File

help_drv_adc.h

C

```
uint32_t DRV_ADC_SamplesRead(uint8_t bufIndex);
```

Returns

uint32_t - ADC converted sample data.

Description

This function returns the converted sample data from ADC input Data buffer.

Remarks

None.

Preconditions

The following functions have been called:

- DRV_ADC_Initialize
- DRV_ADCx_Open
- DRV_ADC_Start or other triggered by source setup in MHC

Parameters

Parameters	Description
uint8_t bufIndex	Analog input number (ANx)

Function

uint32_t DRV_ADC_SamplesRead(uint8_t bufIndex);

DRV_ADC_Start Function

Starts the software trigger for the ADC driver sampling and converting analog to digital values.

Implementation: Static

File

help_drv_adc.h

C

```
void DRV_ADC_Start();
```

Returns

None.

Description

This function provides a global edge and level trigger for the ADC driver to start the conversion.

Remarks

None.

Preconditions

The following functions have been called:

- DRV_ADC_Initialize
- DRV_ADCx_Open

Function

void DRV_ADC_Start(void);

DRV_ADC_Stop Function

Stops the Global Software Level Trigger from continuing triggering for converting ADC data.

Implementation: Static

File

help_drv_adc.h

C

```
void DRV_ADC_Stop();
```

Returns

None.

Description

This function stops the Global Software Level Trigger from continuing triggering for converting ADC data.

Remarks

None.

Preconditions

The following functions have been called:

- DRV_ADC_Initialize
- DRV_ADCx_Open

Function

void DRV_ADC_Stop(void);

DRV_ADCx_Close Function

Closes the ADC instance for the specified driver index.

Implementation: Static

File

help_drv_adc.h

C

```
void DRV_ADCx_Close();
```

Returns

None.

Description

This function closes the specified driver instance (where 'x' is the instance number) making it ready for clients to use it.

Remarks

'x' indicates the instance number.

Preconditions

DRV_ADC_Initialize has been called.

Function

void DRV_ADCx_Close(void)

DRV_ADCx_Open Function

Opens the ADC instance for the specified driver index.

Implementation: Static

File

help_drv_adc.h

C

```
void DRV_ADCx_Open();
```

Returns

None.

Description

This function opens the specified driver instance (where 'x' is the instance number) making it ready for clients to use it.

Remarks

'x' indicates the instance number.

Preconditions

DRV_ADC_Initialize has been called.

Function

void DRV_ADCx_Open(void)

Camera Driver Libraries

This section describes the Camera Driver Libraries.

Introduction

This section provides information on the Camera Driver libraries that are provided in MPLAB Harmony and describes the APIs that are common to all drivers.

Library Interface

a) Common Driver Functions

	Name	Description
	DRV_CAMERA_Close	Closes an opened instance of an CAMERA module driver.
	DRV_CAMERA_Deinitialize	Deinitializes the index instance of the CAMERA module.
	DRV_CAMERA_Initialize	Initializes hardware and data for the index instance of the CAMERA module.
	DRV_CAMERA_Open	Opens the specified instance of the Camera driver for use and provides an "open instance" handle.
	DRV_CAMERA_Reinitialize	Reinitializes hardware and data for the index instance of the CAMERA module.
	DRV_CAMERA_Status	Provides the current status of the index instance of the CAMERA module.
	DRV_CAMERA_Tasks	This is function DRV_CAMERA_Tasks.

b) Common Data Types and Constants

	Name	Description
	DRV_CAMERA_INIT	Defines the data required to initialize or reinitialize the CAMERA driver.
	DRV_CAMERA_INTERRUPT_PORT_REMAP	Defines the data required to initialize the CAMERA driver interrupt port remap.
	DRV_CAMERA_INDEX_0	Camera driver index definitions.
	DRV_CAMERA_INDEX_1	This is macro DRV_CAMERA_INDEX_1.
	DRV_CAMERA_INDEX_COUNT	Number of valid CAMERA driver indices.
	CAMERA_MODULE_ID	This is type CAMERA_MODULE_ID.

Description

Camera Driver APIs that are common to all Camera drivers.

a) Common Driver Functions

DRV_CAMERA_Close Function

Closes an opened instance of an CAMERA module driver.

File

drv_camera.h

C

```
void DRV_CAMERA_Close(DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened instance of an CAMERA module driver, making the specified handle invalid.

Remarks

None.

Preconditions

The DRV_CAMERA_Initialize routine must have been called for the specified CAMERA device instance and the DRV_CAMERA_Status must

have returned SYS_STATUS_READY.

DRV_CAMERA_Open must have been called to obtain a valid opened device handle.

Example

```
myCameraHandle = DRV_CAMERA_Open(DRV_CAMERA_ID_1, DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE);

DRV_CAMERA_Close(myCameraHandle);
```

Parameters

Parameters	Description
drvHandle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_CAMERA_Close (const DRV_HANDLE drvHandle)

DRV_CAMERA_Deinitialize Function

Deinitializes the index instance of the CAMERA module.

File

drv_camera.h

C

```
void DRV_CAMERA_Deinitialize(const SYS_MODULE_INDEX index);
```

Returns

None.

Description

This function deinitializes the index instance of the CAMERA module, disabling its operation (and any hardware for driver modules). It deinitializes only the specified module instance. It also resets all the internal data structures and fields for the specified instance to the default settings.

Remarks

None.

Preconditions

The DRV_CAMERA_Initialize function should have been called before calling this function.

Example

```
SYS_STATUS   cameraStatus;

DRV_CAMERA_Deinitialize(DRV_CAMERA_ID_1);

cameraStatus = DRV_CAMERA_Status(DRV_CAMERA_ID_1);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the CAMERA module to be deinitialized

Function

void DRV_CAMERA_Deinitialize (const SYS_MODULE_ID index)

DRV_CAMERA_Initialize Function

Initializes hardware and data for the index instance of the CAMERA module.

File

drv_camera.h

C

```
SYS_MODULE_OBJ DRV_CAMERA_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

This function initializes hardware for the index instance of the CAMERA module, using the hardware initialization given data. It also initializes any internal driver data structures making the driver ready to be opened.

Remarks

None.

Preconditions

None.

Example

```
DRV_CAMERA_INIT_DATA        cameraInitData;
SYS_STATUS                  cameraStatus;

// Populate the cameraInitData structure
cameraInitData.moduleInit.powerState = SYS_MODULE_POWER_RUN_FULL;
cameraInitData.moduleInit.moduleCode = (DRV_CAMERA_INIT_DATA_MASTER | DRV_CAMERA_INIT_DATA_SLAVE);

DRV_CAMERA_Initialize(DRV_CAMERA_ID_1, (SYS_MODULE_INIT*)&cameraInitData);
cameraStatus = DRV_CAMERA_Status(DRV_CAMERA_ID_1);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the CAMERA module to be initialized
data	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and the default initialization is to be used.

Function

void DRV_CAMERA_Initialize (const CAMERA_MODULE_ID index,
const SYS_MODULE_INIT *const data)

DRV_CAMERA_Open Function

Opens the specified instance of the Camera driver for use and provides an "open instance" handle.

File

drv_camera.h

C

```
DRV_HANDLE DRV_CAMERA_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a value identifying both the caller and the module instance). If an error occurs, the returned value is DRV_HANDLE_INVALID.

Description

This function opens the specified instance of the Camera module for use and provides a handle that is required to use the remaining driver routines.

This function opens a specified instance of the Camera module driver for use by any client module and provides an "open instance" handle that must be provided to any of the other Camera driver operations to identify the caller and the instance of the Camera driver/hardware module.

Preconditions

The DRV_CAMERA_Initialize routine must have been called for the specified CAMERA device instance and the DRV_CAMERA_Status must have returned SYS_STATUS_READY.

Example

```
DRV_HANDLE                cameraHandle;
DRV_CAMERA_CLIENT_STATUS   cameraClientStatus;
```

```
cameraHandle = DRV_CAMERA_Open(DRV_CAMERA_ID_1, DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE);
if (DRV_HANDLE_INVALID == cameraHandle)
{
    // Handle open error
}

cameraClientStatus = DRV_CAMERA_ClientStatus(cameraHandle);

// Close the device when it is no longer needed.
DRV_CAMERA_Close(cameraHandle);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the CAMERA module to be opened.
intent	Flags parameter identifying the intended usage and behavior of the driver. Multiple flags may be ORed together to specify the intended usage of the device. See the DRV_IO_INTENT definition.

Function

DRV_HANDLE DRV_CAMERA_Open (const SYS_MODULE_INDEX index,
const DRV_IO_INTENT intent)

DRV_CAMERA_Reinitialize Function

Reinitializes hardware and data for the index instance of the CAMERA module.

File

drv_camera.h

C

```
void DRV_CAMERA_Reinitialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT *const data);
```

Returns

None.

Description

This function reinitializes hardware for the index instance of the CAMERA module, using the hardware initialization given data. It also reinitializes any internal driver data structures making the driver ready to be opened.

Remarks

None.

Preconditions

The DRV_CAMERA_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_INIT cameraInit;
SYS_STATUS      cameraStatus;

DRV_CAMERA_Reinitialize(DRV_CAMERA_ID_1, &cameraStatus);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the CAMERA module to be reinitialized
data	Pointer to the data structure containing any data necessary to reinitialize the hardware. This pointer may be null if no data is required and default configuration is to be used.

Function

void DRV_CAMERA_Reinitialize(const SYS_MODULE_ID index,
const SYS_MODULE_INIT *const data)

DRV_CAMERA_Status Function

Provides the current status of the index instance of the CAMERA module.

File

drv_camera.h

C

```
SYS_STATUS DRV_CAMERA_Status(const SYS_MODULE_INDEX index);
```

Returns

The current status of the index instance.

Description

This function provides the current status of the index instance of the CAMERA module.

Remarks

None.

Preconditions

The DRV_CAMERA_Initialize function should have been called before calling this function.

Function

SYS_STATUS DRV_CAMERA_Status (const CAMERA_MODULE_ID index)

DRV_CAMERA_Tasks Function

File

drv_camera.h

C

```
void DRV_CAMERA_Tasks(SYS_MODULE_OBJ object);
```

Description

This is function DRV_CAMERA_Tasks.

b) Common Data Types and Constants

DRV_CAMERA_INIT Structure

Defines the data required to initialize or reinitialize the CAMERA driver.

File

drv_camera.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  int cameraId;
  SYS_MODULE_OBJ (* drvInitialize)(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
  DRV_HANDLE (* drvOpen)(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
  INT_SOURCE interruptSource;
  DRV_CAMERA_INTERRUPT_PORT_REMAP interruptPort;
  uint16_t orientation;
  uint16_t horizontalResolution;
  uint16_t verticalResolution;
} DRV_CAMERA_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
int cameraId;	ID
uint16_t orientation;	Orientation of the display (given in degrees of 0,90,180,270)
uint16_t horizontalResolution;	Horizontal Resolution of the displayed orientation in Pixels

Description

CAMERA Driver Initialization Data

This data type defines the data required to initialize or reinitialize the CAMERA driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

DRV_CAMERA_INTERRUPT_PORT_REMAP Structure

Defines the data required to initialize the CAMERA driver interrupt port remap.

File

drv_camera.h

C

```
typedef struct {
  PORTS_REMAP_INPUT_FUNCTION inputFunction;
  PORTS_REMAP_INPUT_PIN inputPin;
  PORTS_ANALOG_PIN analogPin;
  PORTS_PIN_MODE pinMode;
  PORTS_CHANNEL channel;
  PORTS_DATA_MASK dataMask;
} DRV_CAMERA_INTERRUPT_PORT_REMAP;
```

Description

CAMERA Driver Interrupt Port Remap Initialization Data

This data type defines the data required to initialize the CAMERA driver interrupt port remap.

Remarks

None.

DRV_CAMERA_INDEX_0 Macro

Camera driver index definitions.

File

drv_camera.h

C

```
#define DRV_CAMERA_INDEX_0 0
```

Description

Camera Driver Module Index Numbers

These constants provide the Camera driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_CAMERA_Initialize and DRV_CAMERA_Open functions to identify the driver instance in use.

DRV_CAMERA_INDEX_1 Macro

File

drv_camera.h

C

```
#define DRV_CAMERA_INDEX_1 1
```

Description

This is macro DRV_CAMERA_INDEX_1.

DRV_CAMERA_INDEX_COUNT Macro

Number of valid CAMERA driver indices.

File

drv_camera.h

C

```
#define DRV_CAMERA_INDEX_COUNT 1
```

Description

CAMERA Driver Module Index Count

This constant identifies the number of valid CAMERA driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific header files defined as part of the peripheral libraries.

CAMERA_MODULE_ID Enumeration

File

drv_camera.h

C

```
typedef enum {
  CAMERA_MODULE_OVM7690
} CAMERA_MODULE_ID;
```

Description

This is type CAMERA_MODULE_ID.

Files

Files

Name	Description
drv_camera.h	Camera device driver interface file.

Description

drv_camera.h

Camera device driver interface file.

Enumerations

	Name	Description
	CAMERA_MODULE_ID	This is type CAMERA_MODULE_ID.

Functions

	Name	Description
	DRV_CAMERA_Close	Closes an opened instance of an CAMERA module driver.
	DRV_CAMERA_Deinitialize	Deinitializes the index instance of the CAMERA module.
	DRV_CAMERA_Initialize	Initializes hardware and data for the index instance of the CAMERA module.
	DRV_CAMERA_Open	Opens the specified instance of the Camera driver for use and provides an "open instance" handle.
	DRV_CAMERA_Reinitialize	Reinitializes hardware and data for the index instance of the CAMERA module.
	DRV_CAMERA_Status	Provides the current status of the index instance of the CAMERA module.
	DRV_CAMERA_Tasks	This is function DRV_CAMERA_Tasks.

Macros

	Name	Description
	DRV_CAMERA_INDEX_0	Camera driver index definitions.
	DRV_CAMERA_INDEX_1	This is macro DRV_CAMERA_INDEX_1.
	DRV_CAMERA_INDEX_COUNT	Number of valid CAMERA driver indices.

Structures

	Name	Description
	DRV_CAMERA_INIT	Defines the data required to initialize or reinitialize the CAMERA driver.
	DRV_CAMERA_INTERRUPT_PORT_REMAP	Defines the data required to initialize the CAMERA driver interrupt port remap.

Description

Camera Driver Interface

The Camera driver provides a abstraction to all camera drivers.

File Name

drv_camera.h

Company

Microchip Technology Inc.

OVM7690 Camera Driver Library

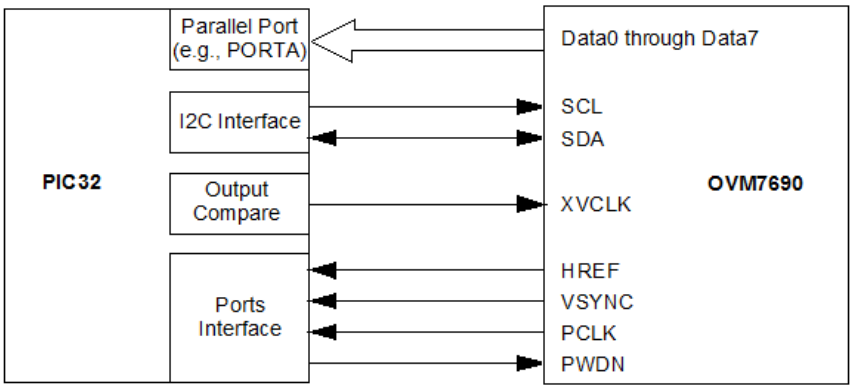
This topic describes the OVM7690 Camera Driver Library.

Introduction

The OVM7690 Camera Driver provides a high-level interface to manage the OmniVision Technologies, Inc. OVM7690 640x480 CameraCube™ device (referred to as the OVM7690) that is interfaced with serial and parallel ports to a Microchip microcontroller for providing camera solutions.

Description

The OVM7690 640x480 CameraCube™ device (referred to as the OVM7690) can be interfaced to a Microchip microcontroller using the I2C serial interface and parallel port interface. The I2C serial interface is used for control command transfer. The I2C module from the microcontroller is connected to the SCCB serial interface of the OVM7690. The parallel port interface is used to transfer pixel data from the OVM7690 to the microcontroller. There are few other signals from the camera to be interfaced with the microcontroller. The XVCLK pin of the camera is driven by the Output Compare module. Frame synchronization signals such as HREF and VSYNC from the camera are connected to suitable pins supporting change notification within the microcontroller. The PCLK pin of the camera drives the pixel clock and is connected at the pin of the microcontroller supporting external interrupts. The PWDN pin of the camera supports camera power-down mode and is connected at any output port pin of the microcontroller. A typical interface of the OVM7690 to a PIC32 device is provided in the following diagram:



Using the Library

This topic describes the basic architecture of the OVM7690 Camera Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_camera_ovm7690.h`

The interface to the Camera Driver Library is defined in the `drv_camera_ovm7690.h` header file.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address the overall operation of the OVM7690 Camera Driver Library.

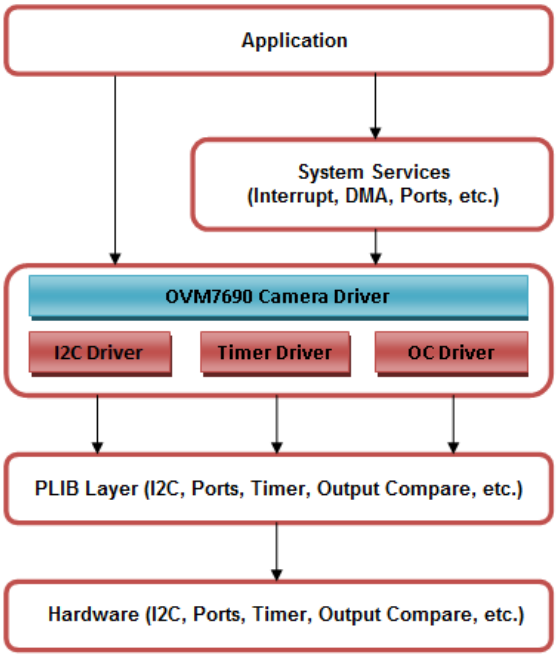
Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization and deinitialization.
Client Setup Functions	Provides open and close functions.
Camera-specific Functions	Provides APIs that are camera-specific.
Other Functions	Provides miscellaneous driver-specific functions such as register set functions, among others.

Abstraction Model

This library provides a low-level abstraction of the OVM7690 Camera Driver Library on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The OVM7690 Camera Driver is modeled using the abstraction model, as shown in the following diagram.



How the Library Works

Provides information on how the OVM7690 Camera Driver Library works.

Description

The library provides interfaces to support:

- System functionality
- Client functionality

System Initialization

The system performs the Initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the OVM7690 would be initialized with the following configuration settings that are supported by the specific OVM7690 device hardware:

- Camera ID: OVM7690 ID
- Source Port: Address of source port to which the pixel data is received
- Horizontal Sync Channel: Channel of the pin to be configured as horizontal sync
- Horizontal Sync Position: Horizontal sync port pin position from selected port channel
- Vertical Sync Channel: Channel the pin to be configured as vertical sync
- Vertical Sync Position: Vertical sync port pin position from selected port channel
- Horizontal Sync Interrupt Source
- Vertical Sync Interrupt Source
- DMA Channel: DMA channel to transfer pixel data from camera to frame buffer
- DMA Channel Trigger Source
- Bits Per Pixel: Bits per pixel to define the size of frame line

The DRV_CAMERA_OVM7690_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handler returned by the Initialize Interface would be used by the other interfaces such as DRV_CAMERA_OVM7690_Deinitialize.

Client Access

For the application to start using an instance of the module, it must call the DRV_CAMERA_OVM7690_Open function. The DRV_CAMERA_OVM7690_Open function provides a driver handle to the OVM7690 Camera Driver instance for operations. If the driver is deinitialized using the function DRV_CAMERA_OVM7690_Deinitialize function, the application must call the DRV_CAMERA_OVM7690_Open function again to set up the instance of the driver.

Client Operations

Client operations provide the API interface for control command and pixel data transfer from the OVM7690 Camera Driver to the Graphics Frame Buffer.

Configuring the Library

Macros

	Name	Description
	DRV_OVM7690_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.

Description

The configuration of the OVM7690 Camera Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the OVM7690 Camera Driver build. Based on the selections made here and the system setup, the OVM7690 Camera Driver may support the selected features. These configuration settings will apply to all instances of the driver.

This header can be placed anywhere in the application specific folders and the path of this header needs to be presented to the include search for a successful build. Refer to the Applications Help section for more details.

Control Commands

The following OVM7690-specific control commands are provided:

- DRV_CAMERA_OVM7690_FrameBufferAddressSet
- DRV_CAMERA_OVM7690_Start
- DRV_CAMERA_OVM7690_Stop
- DRV_CAMERA_OVM7690_FrameRectSet

Application Process

An application needs to perform following steps:

1. The system should have completed necessary setup initializations.
2. The I2C driver object should have been initialized by calling DRV_I2C_Initialize.
3. The Timer driver object should have been initialized by calling DRV_Timer_Initialize,
4. The Output Control driver object should have been initialized by calling DRV_OC_Initialize,
5. The OVM7690 Camera Driver object should have been initialized by calling DRV_CAMERA_OVM7690_Initialize,
6. Open the OVM7690 Camera Driver client by calling DRV_CAMERA_OVM7690_Open.
7. Pass the Graphics Frame buffer address to OVM7690 Camera Driver by calling DRV_CAMERA_OVM7690_FrameBufferAddressSet.
8. Set the Frame Rectangle area by calling DRV_CAMERA_OVM7690_FrameRectSet.
9. Set Other Camera settings such as: soft reset, enabling pclk, enabling href, enabling vsync, output color format, reversing HREF polarity, gating clock to the HREF, pixel clock frequency, sub-sampling mode by calling DRV_CAMERA_OVM7690_RegisterSet.
10. Start the OVM7690 Camera by calling DRV_CAMERA_OVM7690_Start.

DRV_OVM7690_INTERRUPT_MODE Macro

Controls operation of the driver in the interrupt or polled mode.

File

drv_ovm7690_config_template.h

C

```
#define DRV_OVM7690_INTERRUPT_MODE false
```

Description

OVM7690 Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of OVM7690 operation is desired
- false - Select if polling mode of OVM7690 operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

Building the Library

This section lists the files that are available in the OVM7690 Camera Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/camera/ovm7690`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_camera_ovm7690.h`	This file provides the interface definitions of the OVM7690 Camera Driver.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/drv_camera_ovm7690.c`	This file contains the implementation of the OVM7690 Camera Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The OVM7690 Camera Driver Library depends on the following modules:

- I2C Driver Library
- Output Compare Driver Library
- Timer Driver Library

Library Interface

a) System Functions

	Name	Description
	DRV_CAMERA_OVM7690_Initialize	Initializes the OVM7690 Camera instance for the specified driver index.
	DRV_CAMERA_OVM7690_Deinitialize	Deinitializes the specified instance of the OVM7690 Camera Driver module.
	DRV_CAMERA_OVM7690_RegisterSet	Sets the camera OVM7690 configuration registers.
	DRV_CAMERA_OVM7690_Tasks	Maintains the OVM7690 state machine.

b) Client Setup Functions

	Name	Description
	DRV_CAMERA_OVM7690_Open	Opens the specified OVM7690 Camera Driver instance and returns a handle to it.
	DRV_CAMERA_OVM7690_Close	Closes an opened instance of the OVM7690 Camera Driver.

c) Camera-specific Functions

	Name	Description
	DRV_CAMERA_OVM7690_FrameBufferAddressSet	Sets the framebuffer address.
	DRV_CAMERA_OVM7690_FrameRectSet	Sets the frame rectangle set.
	DRV_CAMERA_OVM7690_Start	Starts camera rendering to the display.
	DRV_CAMERA_OVM7690_Stop	Stops rendering the camera Pixel data.

d) Other Functions

	Name	Description
	DRV_CAMERA_OVM7690_HsyncEventHandler	Horizontal synchronization event handler.
	DRV_CAMERA_OVM7690_VsyncEventHandler	Vertical synchronization event handler .

	_DRV_CAMERA_OVM7690_DMAEventHandler	This is function _DRV_CAMERA_OVM7690_DMAEventHandler.
	_DRV_CAMERA_OVM7690_delayMS	This is function _DRV_CAMERA_OVM7690_delayMS.
	_DRV_CAMERA_OVM7690_HardwareSetup	This is function _DRV_CAMERA_OVM7690_HardwareSetup.

e) Data Types and Constants

	Name	Description
	DRV_CAMERA_OVM7690_CLIENT_OBJ	OVM7690 Camera Driver client object.
	DRV_CAMERA_OVM7690_CLIENT_STATUS	Identifies OVM7690 Camera possible client status.
	DRV_CAMERA_OVM7690_ERROR	Identifies OVM7690 Camera possible errors.
	DRV_CAMERA_OVM7690_INIT	OVM7690 Camera Driver initialization parameters.
	DRV_CAMERA_OVM7690_OBJ	OVM7690 Camera Driver instance object.
	DRV_CAMERA_OVM7690_RECT	OVM7690 Camera window rectangle coordinates.
	DRV_CAMERA_OVM7690_REG12_OP_FORMAT	Lists OVM7690 Camera device register addresses.
	DRV_CAMERA_OVM7690_INDEX_0	OVM7690 driver index definitions.
	DRV_CAMERA_OVM7690_INDEX_1	This is macro DRV_CAMERA_OVM7690_INDEX_1.
	DRV_CAMERA_OVM7690_REG12_SOFT_RESET	OVM7690 Camera Driver Register 0x12 Soft reset flag.
	DRV_CAMERA_OVM7690_SCCB_READ_ID	OVM7690 Camera SCCB Interface device Read Slave ID.
	DRV_CAMERA_OVM7690_SCCB_WRITE_ID	OVM7690 Camera SCCB Interface device Write Slave ID.

Description

This section describes the Application Programming Interface (API) functions of the Camera Driver Library.

a) System Functions

DRV_CAMERA_OVM7690_Initialize Function

Initializes the OVM7690 Camera instance for the specified driver index.

File

drv_camera_ovm7690.h

C

```
SYS_MODULE_OBJ DRV_CAMERA_OVM7690_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the OVM7690 Camera Driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the OVM7690 Camera module ID. Refer to the description of the DRV_CAMERA_OVM7690_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This function must be called before any other OVM7690 Camera Driver function is called.

This function should only be called once during system initialization unless DRV_CAMERA_OVM7690_Deinitialize is called to deinitialize the driver instance. This function will NEVER block for hardware access.

Preconditions

None.

Example

```
// The following code snippet shows an example OVM7690 driver initialization.

DRV_CAMERA_OVM7690_INIT     cameraInit;
SYS_MODULE_OBJ              objectHandle;

cameraInit.cameraID                 = CAMERA_MODULE_OVM7690;
cameraInit.sourcePort               = (void *)&PORTK,
```

```
cameraInit.hsyncInterruptSource    = INT_SOURCE_CHANGE_NOTICE_A,
cameraInit.vsyncInterruptSource    = INT_SOURCE_CHANGE_NOTICE_J,
cameraInit.dmaChannel              = DRV_CAMERA_OVM7690_DMA_CHANNEL_INDEX,
cameraInit.dmaTriggerSource        = DMA_TRIGGER_EXTERNAL_2,
cameraInit.bpp                     = GFX_CONFIG_COLOR_DEPTH,

objectHandle = DRV_CAMERA_OVM7690_Initialize( DRV_CAMERA_OVM7690_INDEX_0,
                                              (SYS_MODULE_INIT*)&cameraInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_CAMERA_OVM7690_Initialize
(
const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init
)

DRV_CAMERA_OVM7690_Deinitialize Function

Deinitializes the specified instance of the OVM7690 Camera Driver module.

File

drv_camera_ovm7690.h

C

```
void DRV_CAMERA_OVM7690_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the specified instance of the OVM7690 Camera Driver module, disabling its operation (and any hardware), and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This function will NEVER block waiting for hardware.

Preconditions

Function DRV_CAMERA_OVM7690_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object; //  Returned from DRV_CAMERA_OVM7690_Initialize
SYS_STATUS          status;

DRV_CAMERA_OVM7690_Deinitialize(object);

status = DRV_CAMERA_OVM7690_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_CAMERA_OVM7690_Initialize function

Function

void DRV_CAMERA_OVM7690_Deinitialize(SYS_MODULE_OBJ object)

DRV_CAMERA_OVM7690_RegisterSet Function

Sets the camera OVM7690 configuration registers.

File

drv_camera_ovm7690.h

C

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_RegisterSet(DRV_CAMERA_OVM7690_REGISTER_ADDRESS regIndex,
uint8_t regValue);
```

Returns

- DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE - Invalid driver Handle.
- DRV_CAMERA_OVM7690_ERROR_NONE - No error.

Description

This function sets the OVM7690 Camera configuration registers using the SCCB interface.

Remarks

This function can be used separately or within an interface.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

The SCCB interface also must have been initialized to configure the OVM7690 Camera Driver.

Example

```
DRV_HANDLE handle;
uint8_t reg12 = DRV_CAMERA_OVM7690_REG12_SOFT_RESET;

 handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
 if (DRV_HANDLE_INVALID == handle)
 {
     //error
     return;
 }

 if ( DRV_CAMERA_OVM7690_RegisterSet( DRV_CAMERA_OVM7690_REG12_REG_ADDR,
                                      reg12 ) !=
                                      DRV_CAMERA_OVM7690_ERROR_NONE )
 {
     //error
     return;
 }
```

Parameters

Parameters	Description
regIndex	Defines the OVM7690 configuration register addresses.
regValue	Defines the register value to be set.

Function

DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_RegisterSet

(

DRV_CAMERA_OVM7690_REGISTER_ADDRESS regIndex,

uint8_t regValue
)

DRV_CAMERA_OVM7690_Tasks Function

Maintains the OVM7690 state machine.

File

drv_camera_ovm7690.h

C

```
void DRV_CAMERA_OVM7690_Tasks(SYS_MODULE_OBJ object);
```

Function

void DRV_CAMERA_OVM7690_Tasks(SYS_MODULE_OBJ object);

b) Client Setup Functions

DRV_CAMERA_OVM7690_Open Function

Opens the specified OVM7690 Camera Driver instance and returns a handle to it.

File

drv_camera_ovm7690.h

C

```
DRV_HANDLE DRV_CAMERA_OVM7690_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the function returns a valid open instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Errors can occur:

- if the number of client objects allocated via DRV_CAMERA_OVM7690_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client
- if the driver is not ready to be opened, typically when the initialize function has not completed execution

Description

This function opens the specified OVM7690 Camera Driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

Remarks

The handle returned is valid until the DRV_CAMERA_OVM7690_Close function is called. This function will NEVER block waiting for hardware.If the requested intent flags are not supported, the function will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application.

Preconditions

Function DRV_CAMERA_OVM7690_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

```
DRV_HANDLE DRV_CAMERA_OVM7690_Open
(
const    SYS_MODULE_INDEX index,
const    DRV_IO_INTENT ioIntent
)
```

DRV_CAMERA_OVM7690_Close Function

Closes an opened instance of the OVM7690 Camera Driver.

File

drv_camera_ovm7690.h

C

```
void DRV_CAMERA_OVM7690_Close(DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened instance of the OVM7690 Camera Driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this function, the handle passed in "handle" must not be used with any of the remaining driver routines (with one possible exception described in the "Remarks" section). A new handle must be obtained by calling DRV_CAMERA_OVM7690_Open before the caller may use the driver again

Remarks

Usually there is no need for the client to verify that the Close operation has completed. The driver will abort any ongoing operations when this function is called.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_USART_Open
DRV_CAMERA_OVM7690_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's Open function

Function

void DRV_CAMERA_OVM7690_Close(DRV_Handle handle)

c) Camera-specific Functions

DRV_CAMERA_OVM7690_FrameBufferAddressSet Function

Sets the framebuffer address.

File

drv_camera_ovm7690.h

C

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_FrameBufferAddressSet(DRV_HANDLE handle, void * frameBuffer);
```

Returns

- DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE - Invalid driver Handle.
- DRV_CAMERA_OVM7690_ERROR_NONE - No error.

Description

This function will set the framebuffer address. This framebuffer address will point to the location at which frame data is to be rendered. This buffer is shared with the display controller to display the frame on the display.

Remarks

This function is mandatory. A valid framebuffer address must be set to display the camera data.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;
uint16_t frameBuffer[DISP_VER_RESOLUTION][DISP_HOR_RESOLUTION];

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    //error
    return;
}

if ( DRV_CAMERA_OVM7690_FrameBufferAddressSet( handle, (void *) frameBuffer ) !=
                                    DRV_CAMERA_OVM7690_ERROR_NONE )
{
    //error
    return;
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's Open function

Function

DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_FrameBufferAddressSet

(

DRV_HANDLE handle,

void * frameBuffer

)

DRV_CAMERA_OVM7690_FrameRectSet Function

Sets the frame rectangle set.

File

drv_camera_ovm7690.h

C

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_FrameRectSet(DRV_HANDLE handle, uint32_t left, uint32_t top,
uint32_t right, uint32_t bottom);
```

Returns

- DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE - Invalid driver Handle.
- DRV_CAMERA_OVM7690_ERROR_NONE - No error.

Description

This function sets the frame rectangle coordinates. The frame within the rectangle is copied to the framebuffer. The left and top values are expected to be less than right and bottom respectively. Left, top, right, and bottom values are also expected to be within range of screen coordinates. Internally it calls the DRV_CAMERA_OVM7690_RegisterSet function to set the respective registers. The rectangle coordinates are also maintained in the driver object.

Remarks

This function is optional if default values are expected to be used.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

The SCCB interface also must have been initialized to configure the OVM7690 Camera Driver.

Example

```
DRV_HANDLE handle;
uint32_t left   = 0x69;
uint32_t top    = 0x0E;
uint32_t right  = DISP_HOR_RESOLUTION + 0x69;
uint32_t bottom = DISP_VER_RESOLUTION + 0x69;

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    //error
    return;
}

if ( DRV_CAMERA_OVM7690_FrameRectSet( handle, left, top, right, bottom ) !=
                                      DRV_CAMERA_OVM7690_ERROR_NONE )
{
    //error
    return;
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's Open function
left	left frame coordinate
top	top frame coordinate
right	right frame coordinate
bottom	bottom frame coordinate

Function

DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_FrameRectSet

(

DRV_HANDLE handle,

uint32_t left,

uint32_t top,

uint32_t right,

uint32_t bottom

)

DRV_CAMERA_OVM7690_Start Function

Starts camera rendering to the display.

File

drv_camera_ovm7690.h

C

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_Start(DRV_HANDLE handle);
```

Returns

- DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE - Invalid driver Handle.
- DRV_CAMERA_OVM7690_ERROR_NONE - No error.

Description

This function starts the camera rendering to the display by writing the pixel data to the framebuffer. The framebuffer is shared between the OVM7690 Camera and the display controller.

Remarks

This function is mandatory. Camera module will not update the framebuffer without calling this function.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

DRV_CAMERA_OVM7690_FrameBufferAddressSet must have been called to set a valid framebuffer address.

Example

```
DRV_HANDLE handle;
uint16_t frameBuffer[DISP_VER_RESOLUTION][DISP_HOR_RESOLUTION];

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    //error
    return;
}

if ( DRV_CAMERA_OVM7690_FrameBufferAddressSet( handle, (void *) frameBuffer ) !=
                                    DRV_CAMERA_OVM7690_ERROR_NONE )
{
    //error
    return;
}

if ( DRV_CAMERA_OVM7690_Start( handle ) !=
                                    DRV_CAMERA_OVM7690_ERROR_NONE )
{
    //error
    return;
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's Open function

Function

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_Start
(
    DRV_HANDLE handle
);
```

DRV_CAMERA_OVM7690_Stop Function

Stops rendering the camera Pixel data.

File

drv_camera_ovm7690.h

C

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_Stop(DRV_HANDLE handle);
```

Returns

- DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE - Invalid driver Handle.
- DRV_CAMERA_OVM7690_ERROR_NONE - No error.

Description

This function starts the camera rendering to the display by writing the pixel data to the framebuffer. The framebuffer is shared between the OVM7690 Camera and the display controller.

Remarks

This function only disables the interrupt for HSYNC and VSYNC. To stop the camera the power-down pin needs to be toggled to an active-high value., which will stop the camera internal clock and maintain the register values.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    //error
    return;
}

if ( DRV_CAMERA_OVM7690_Stop( handle ) !=
                                    DRV_CAMERA_OVM7690_ERROR_NONE )
{
    //error
    return;
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's Open function.

Function

```
DRV_CAMERA_OVM7690_ERROR DRV_CAMERA_OVM7690_Stop
(
    DRV_HANDLE handle
);
```

d) Other Functions

DRV_CAMERA_OVM7690_HsyncEventHandler Function

Horizontal synchronization event handler.

File

drv_camera_ovm7690.h

C

```
void DRV_CAMERA_OVM7690_HsyncEventHandler(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is called when the OVM7690 Camera sends a Horizontal Sync Pulse on the HSYNC line. It sets the next line address in the DMA module.

Remarks

This function is mandatory.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_CAMERA_OVM7690_INIT     cameraInit;
SYS_MODULE_OBJ              objectHandle;

cameraInit.cameraID                 = CAMERA_MODULE_OVM7690;
cameraInit.sourcePort               = (void *)&PORTK,
cameraInit.hsyncInterruptSource     = INT_SOURCE_CHANGE_NOTICE_A,
cameraInit.vsyncInterruptSource     = INT_SOURCE_CHANGE_NOTICE_J,
cameraInit.dmaChannel               = DRV_CAMERA_OVM7690_DMA_CHANNEL_INDEX,
cameraInit.dmaTriggerSource         = DMA_TRIGGER_EXTERNAL_2,
cameraInit.bpp                      = GFX_CONFIG_COLOR_DEPTH,

objectHandle = DRV_CAMERA_OVM7690_Initialize( DRV_CAMERA_OVM7690_INDEX_0,
                                              (SYS_MODULE_INIT*)&cameraInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    //error
    return;
}

void __ISR( HSYNC_ISR_VECTOR) _Ovm7690HSyncHandler(void)
{
    DRV_CAMERA_OVM7690_HsyncEventHandler(objectHandle);

    SYS_INT_SourceStatusClear(HSYNC_INTERRUPT_SOURCE);
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_CAMERA_OVM7690_Initialize function

Function

void DRV_CAMERA_OVM7690_HsyncEventHandler(SYS_MODULE_OBJ object)

DRV_CAMERA_OVM7690_VsyncEventHandler Function

Vertical synchronization event handler .

File

drv_camera_ovm7690.h

C

```
void DRV_CAMERA_OVM7690_VsyncEventHandler(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is called when the OVM7690 Camera sends a Vertical Sync Pulse on the VSYNC line. It clears the number of lines drawn variable.

Remarks

This function is mandatory.

Preconditions

The DRV_CAMERA_OVM7690_Initialize function must have been called for the specified OVM7690 Camera Driver instance.

DRV_CAMERA_OVM7690_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_CAMERA_OVM7690_INIT     cameraInit;
SYS_MODULE_OBJ              objectHandle;

cameraInit.cameraID                 = CAMERA_MODULE_OVM7690;
cameraInit.sourcePort               = (void *)&PORTK,
cameraInit.hsyncInterruptSource     = INT_SOURCE_CHANGE_NOTICE_A,
cameraInit.vsyncInterruptSource     = INT_SOURCE_CHANGE_NOTICE_J,
cameraInit.dmaChannel               = DRV_CAMERA_OVM7690_DMA_CHANNEL_INDEX,
cameraInit.dmaTriggerSource         = DMA_TRIGGER_EXTERNAL_2,
cameraInit.bpp                      = GFX_CONFIG_COLOR_DEPTH,

objectHandle = DRV_CAMERA_OVM7690_Initialize( DRV_CAMERA_OVM7690_INDEX_0,
                                              (SYS_MODULE_INIT*)&cameraInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}

handle = DRV_CAMERA_OVM7690_Open(DRV_CAMERA_OVM7690_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    //error
    return;
}

void __ISR( VSYNC_ISR_VECTOR) _Ovm7690VSyncHandler(void)
{
    DRV_CAMERA_OVM7690_VsyncEventHandler(objectHandle);

    SYS_INT_SourceStatusClear(VSYNC_INTERRUPT_SOURCE);
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_CAMERA_OVM7690_Initialize function

Function

void DRV_CAMERA_OVM7690_VsyncEventHandler(SYS_MODULE_OBJ object)

_DRV_CAMERA_OVM7690_DMAEventHandler Function

File

drv_camera_ovm7690.h

C

```
void _DRV_CAMERA_OVM7690_DMAEventHandler(SYS_DMA_TRANSFER_EVENT event, SYS_DMA_CHANNEL_HANDLE handle,
uintptr_t contextHandle);
```

Description

This is function _DRV_CAMERA_OVM7690_DMAEventHandler.

_DRV_CAMERA_OVM7690_delayMS Function

File

drv_camera_ovm7690.h

C

```
void _DRV_CAMERA_OVM7690_delayMS(unsigned int delayMs);
```

Description

This is function _DRV_CAMERA_OVM7690_delayMS.

_DRV_CAMERA_OVM7690_HardwareSetup Function

File

drv_camera_ovm7690.h

C

```
void _DRV_CAMERA_OVM7690_HardwareSetup(DRV_CAMERA_OVM7690_OBJ * dObj);
```

Description

This is function _DRV_CAMERA_OVM7690_HardwareSetup.

e) Data Types and Constants

DRV_CAMERA_OVM7690_CLIENT_OBJ Structure

OVM7690 Camera Driver client object.

File

drv_camera_ovm7690.h

C

```
typedef struct {
  DRV_CAMERA_OVM7690_OBJ * hDriver;
  DRV_IO_INTENT ioIntent;
  bool inUse;
  DRV_CAMERA_OVM7690_ERROR error;
  DRV_CAMERA_OVM7690_CLIENT_STATUS status;
} DRV_CAMERA_OVM7690_CLIENT_OBJ;
```

Members

Members	Description
DRV_CAMERA_OVM7690_OBJ * hDriver;	The hardware instance object associated with the client
DRV_IO_INTENT ioIntent;	The I/O intent with which the client was opened
bool inUse;	This flags indicates if the object is in use or is available
DRV_CAMERA_OVM7690_ERROR error;	Driver Error
DRV_CAMERA_OVM7690_CLIENT_STATUS status;	Client status

Description

OVM7690 Camera Driver Client Object.

This structure provides a definition of the OVM7690 Camera Driver client object.

Remarks

These values are been updated into the DRV_CAMERA_OVM7690_Open function.

DRV_CAMERA_OVM7690_CLIENT_STATUS Enumeration

Identifies OVM7690 Camera possible client status.

File

drv_camera_ovm7690.h

C

```
typedef enum {
  DRV_CAMERA_OVM7690_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR,
  DRV_CAMERA_OVM7690_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED,
  DRV_CAMERA_OVM7690_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY,
  DRV_CAMERA_OVM7690_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY
} DRV_CAMERA_OVM7690_CLIENT_STATUS;
```

Members

Members	Description
DRV_CAMERA_OVM7690_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR	An error has occurred.
DRV_CAMERA_OVM7690_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED	The driver is closed, no operations for this client are ongoing, and/or the given handle is invalid.
DRV_CAMERA_OVM7690_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY	The driver is currently busy and cannot start additional operations.
DRV_CAMERA_OVM7690_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY	The module is running and ready for additional operations

Description

OVM7690 Camera Client Status.

This enumeration defines possible OVM7690 Camera Client Status.

Remarks

This enumeration values are set by driver interfaces: DRV_CAMERA_OVM7690_Open and DRV_CAMERA_OVM7690_Close.

DRV_CAMERA_OVM7690_ERROR Enumeration

Identifies OVM7690 Camera possible errors.

File

drv_camera_ovm7690.h

C

```
typedef enum {
  DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE,
  DRV_CAMERA_OVM7690_ERROR_NONE
} DRV_CAMERA_OVM7690_ERROR;
```

Members

Members	Description
DRV_CAMERA_OVM7690_ERROR_INVALID_HANDLE	OVM7690 Camera Driver Invalid Handle
DRV_CAMERA_OVM7690_ERROR_NONE	OVM7690 Camera Driver error none

Description

OVM7690 Camera Error flag

This enumeration defines possible OVM7690 Camera errors.

Remarks

This enumeration values are returned by driver interfaces in case of errors.

DRV_CAMERA_OVM7690_INIT Structure

OVM7690 Camera Driver initialization parameters.

File

drv_camera_ovm7690.h

C

```
typedef struct {
  CAMERA_MODULE_ID cameraID;
  void * sourcePort;
  PORTS_CHANNEL hsyncChannel;
  PORTS_BIT_POS hsyncPosition;
  PORTS_CHANNEL vsyncChannel;
  PORTS_BIT_POS vsyncPosition;
  INT_SOURCE hsyncInterruptSource;
  INT_SOURCE vsyncInterruptSource;
  DMA_CHANNEL dmaChannel;
  DMA_TRIGGER_SOURCE dmaTriggerSource;
  uint16_t bpp;
} DRV_CAMERA_OVM7690_INIT;
```

Members

Members	Description
CAMERA_MODULE_ID cameraID;	Camera module ID
void * sourcePort;	Source Port Address
PORTS_CHANNEL hsyncChannel;	HSYNC pin channel
PORTS_BIT_POS hsyncPosition;	HSYNC pin bit position
PORTS_CHANNEL vsyncChannel;	VSYNC pin channel
PORTS_BIT_POS vsyncPosition;	VSYNC pin bit position
INT_SOURCE hsyncInterruptSource;	HSYNC Interrupt Source
INT_SOURCE vsyncInterruptSource;	VSYNC Interrupt Source
DMA_CHANNEL dmaChannel;	DMA channel
DMA_TRIGGER_SOURCE dmaTriggerSource;	DMA trigger source
uint16_t bpp;	Bits per pixel

Description

OVM7690 Camera Initialization parameters

This structure defines OVM7690 Camera Driver initialization parameters.

Remarks

These values should be passed into the DRV_CAMERA_OVM7690_Initialize function.

DRV_CAMERA_OVM7690_OBJ Structure

OVM7690 Camera Driver instance object.

File

drv_camera_ovm7690.h

C

```
typedef struct {
  CAMERA_MODULE_ID moduleId;
  SYS_STATUS status;
  bool inUse;
  bool isExclusive;
  size_t nClients;
  PORTS_CHANNEL hsyncChannel;
  PORTS_BIT_POS hsyncPosition;
  PORTS_CHANNEL vsyncChannel;
  PORTS_BIT_POS vsyncPosition;
  INT_SOURCE hsyncInterruptSource;
  INT_SOURCE vsyncInterruptSource;
  SYS_DMA_CHANNEL_HANDLE dmaHandle;
  DMA_CHANNEL dmaChannel;
  DMA_TRIGGER_SOURCE dmaTriggerSource;
  bool dmaTransferComplete;
  void * sourcePort;
  uint32_t frameLineCount;
```

```
  uint32_t frameLineSize;
  void * frameLineAddress;
  void * frameBufferAddress;
  DRV_CAMERA_OVM7690_RECT rect;
  uint16_t bpp;
} DRV_CAMERA_OVM7690_OBJ;
```

Members

Members	Description
CAMERA_MODULE_ID moduleId;	The module index associated with the object
SYS_STATUS status;	The status of the driver
bool inUse;	Flag to indicate this object is in use
bool isExclusive;	Flag to indicate that driver has been opened exclusively.
size_t nClients;	Keeps track of the number of clients • that have opened this driver
PORTS_CHANNEL hsyncChannel;	HSYNC pin channel
PORTS_BIT_POS hsyncPosition;	HSYNC pin bit position
PORTS_CHANNEL vsyncChannel;	VSYNC pin channel
PORTS_BIT_POS vsyncPosition;	VSYNC pin bit position
INT_SOURCE hsyncInterruptSource;	HSYNC Interrupt Source
INT_SOURCE vsyncInterruptSource;	VSYNC Interrupt Source
SYS_DMA_CHANNEL_HANDLE dmaHandle;	DMA Handle
DMA_CHANNEL dmaChannel;	Read DMA channel
DMA_TRIGGER_SOURCE dmaTriggerSource;	DMA Trigger Source
bool dmaTransferComplete;	DMA Transfer Complete Flag
void * sourcePort;	Source Port Address
uint32_t frameLineCount;	Frame Line Count
uint32_t frameLineSize;	Frame Line Size
void * frameLineAddress;	Frame Line Address
void * frameBufferAddress;	Framebuffer Address
DRV_CAMERA_OVM7690_RECT rect;	Window Rectangle
uint16_t bpp;	Bits per pixel supported

Description

OVM7690 Camera Driver Instance Object

This structure provides a definition of the OVM7690 Camera Driver instance object.

Remarks

These values are been updated into the DRV_CAMERA_OVM7690_Initialize function.

DRV_CAMERA_OVM7690_RECT Structure

OVM7690 Camera window rectangle coordinates.

File

drv_camera_ovm7690.h

C

```
typedef struct {
  uint32_t left;
  uint32_t top;
  uint32_t right;
  uint32_t bottom;
} DRV_CAMERA_OVM7690_RECT;
```

Members

Members	Description
uint32_t left;	OVM7690 Camera Window left coordinate
uint32_t top;	OVM7690 Camera Window top coordinate
uint32_t right;	OVM7690 Camera Window right coordinate

uint32_t bottom;	OVM7690 Camera Window bottom coordinate

Description

OVM7690 Camera Window Rect

This structure defines window rectangle co-ordinates as left, right, top, and bottom.

Remarks

These values should be passed into the DRV_CAMERA_OVM7690_FrameRectSet function.

DRV_CAMERA_OVM7690_REG12_OP_FORMAT Enumeration

Lists OVM7690 Camera device register addresses.

File

drv_camera_ovm7690.h

C

```
typedef enum {
  DRV_CAMERA_OVM7690_REG12_OP_FORMAT_RAW_2
} DRV_CAMERA_OVM7690_REG12_OP_FORMAT;
```

Members

Members	Description
DRV_CAMERA_OVM7690_REG12_OP_FORMAT_RAW_2	Bayer Raw Format

Description

OVM7690 Camera Device Register Addresses.

This enumeration defines the list of device register addresses.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_CAMERA_OVM7690_RegisterSet function. Refer to the specific device data sheet for more information.

DRV_CAMERA_OVM7690_INDEX_0 Macro

OVM7690 driver index definitions.

File

drv_camera_ovm7690.h

C

```
#define DRV_CAMERA_OVM7690_INDEX_0 0
```

Description

OVM7690 Camera Driver Module Index

These constants provide OVM7690 Camera Driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_CAMERA_OVM7690_Initialize and DRV_CAMERA_OVM7690_Open routines to identify the driver instance in use.

DRV_CAMERA_OVM7690_INDEX_1 Macro

File

drv_camera_ovm7690.h

C

```
#define DRV_CAMERA_OVM7690_INDEX_1 1
```

Description

This is macro DRV_CAMERA_OVM7690_INDEX_1.

DRV_CAMERA_OVM7690_REG12_SOFT_RESET Macro

OVM7690 Camera Driver Register 0x12 Soft reset flag.

File

drv_camera_ovm7690.h

C

```
#define DRV_CAMERA_OVM7690_REG12_SOFT_RESET
```

Description

OVM7690 Camera Driver Soft reset flag.

This macro provides a definition of the OVM7690 Camera Register 0x12 Soft reset flag.

Remarks

These constants should be used in place of hard-coded numeric literals.

DRV_CAMERA_OVM7690_SCCB_READ_ID Macro

OVM7690 Camera SCCB Interface device Read Slave ID.

File

drv_camera_ovm7690.h

C

```
#define DRV_CAMERA_OVM7690_SCCB_READ_ID
```

Description

OVM7690 Camera Driver SCCB Read ID

This macro provides a definition of the OVM7690 Camera SCCB Interface device Read Slave ID.

Remarks

These constants should be used in place of hard-coded numeric literals.

DRV_CAMERA_OVM7690_SCCB_WRITE_ID Macro

OVM7690 Camera SCCB Interface device Write Slave ID.

File

drv_camera_ovm7690.h

C

```
#define DRV_CAMERA_OVM7690_SCCB_WRITE_ID
```

Description

OVM7690 Camera Driver SCCB Write ID

This macro provides a definition of the OVM7690 Camera SCCB Interface device Write Slave ID.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_CAMERA_OVM7690_RegisterSet function to identify the OVM7690 Camera SCCB Interface device Write Slave ID.

Files

Files

Name	Description
drv_camera_ovm7690.h	OVM7690 Camera Driver local data structures.
drv_ovm7690_config_template.h	OVM7690 Device Driver configuration template.

Description

drv_camera_ovm7690.h

OVM7690 Camera Driver local data structures.

Enumerations

	Name	Description
	DRV_CAMERA_OVM7690_CLIENT_STATUS	Identifies OVM7690 Camera possible client status.
	DRV_CAMERA_OVM7690_ERROR	Identifies OVM7690 Camera possible errors.
	DRV_CAMERA_OVM7690_REG12_OP_FORMAT	Lists OVM7690 Camera device register addresses.

Functions

	Name	Description
	_DRV_CAMERA_OVM7690_delayMS	This is function _DRV_CAMERA_OVM7690_delayMS.
	_DRV_CAMERA_OVM7690_DMAEventHandler	This is function _DRV_CAMERA_OVM7690_DMAEventHandler.
	_DRV_CAMERA_OVM7690_HardwareSetup	This is function _DRV_CAMERA_OVM7690_HardwareSetup.
	DRV_CAMERA_OVM7690_Close	Closes an opened instance of the OVM7690 Camera Driver.
	DRV_CAMERA_OVM7690_Deinitialize	Deinitializes the specified instance of the OVM7690 Camera Driver module.
	DRV_CAMERA_OVM7690_FrameBufferAddressSet	Sets the framebuffer address.
	DRV_CAMERA_OVM7690_FrameRectSet	Sets the frame rectangle set.
	DRV_CAMERA_OVM7690_HsyncEventHandler	Horizontal synchronization event handler.
	DRV_CAMERA_OVM7690_Initialize	Initializes the OVM7690 Camera instance for the specified driver index.
	DRV_CAMERA_OVM7690_Open	Opens the specified OVM7690 Camera Driver instance and returns a handle to it.
	DRV_CAMERA_OVM7690_RegisterSet	Sets the camera OVM7690 configuration registers.
	DRV_CAMERA_OVM7690_Start	Starts camera rendering to the display.
	DRV_CAMERA_OVM7690_Stop	Stops rendering the camera Pixel data.
	DRV_CAMERA_OVM7690_Tasks	Maintains the OVM7690 state machine.
	DRV_CAMERA_OVM7690_VsyncEventHandler	Vertical synchronization event handler .

Macros

	Name	Description
	DRV_CAMERA_OVM7690_INDEX_0	OVM7690 driver index definitions.
	DRV_CAMERA_OVM7690_INDEX_1	This is macro DRV_CAMERA_OVM7690_INDEX_1.
	DRV_CAMERA_OVM7690_REG12_SOFT_RESET	OVM7690 Camera Driver Register 0x12 Soft reset flag.
	DRV_CAMERA_OVM7690_SCCB_READ_ID	OVM7690 Camera SCCB Interface device Read Slave ID.
	DRV_CAMERA_OVM7690_SCCB_WRITE_ID	OVM7690 Camera SCCB Interface device Write Slave ID.

Structures

	Name	Description
	DRV_CAMERA_OVM7690_CLIENT_OBJ	OVM7690 Camera Driver client object.
	DRV_CAMERA_OVM7690_INIT	OVM7690 Camera Driver initialization parameters.
	DRV_CAMERA_OVM7690_OBJ	OVM7690 Camera Driver instance object.
	DRV_CAMERA_OVM7690_RECT	OVM7690 Camera window rectangle coordinates.

Description

OVM7690 Camera Driver Local Data Structures

This header file provides the local data structures for the OVM7690 Camera Driver Library.

File Name

drv_camera_ovm7690.h

Company

Microchip Technology Inc.

drv_ovm7690_config_template.h

OVM7690 Device Driver configuration template.

Macros

	Name	Description
	DRV_OVM7690_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.

Description

OVM7690 Device Driver Configuration Template

This header file contains the build-time configuration selections for the OVM7690 device driver. This is the template file which give all possible configurations that can be made. This file should not be included in any project.

File Name

drv_ovm7690_config_template.h

Company

Microchip Technology Inc.

CAN Driver Library

This section describes the CAN Driver Library.

Introduction

The CAN Static Driver provides a high-level interface to manage the CAN module on the Microchip family of microcontrollers.

Description

Through MHC, this driver provides an API to initialize the CAN module, as well as the baud rate. The API also allows simple transmit and receive functionality.

Library Interface

Function(s)

	Name	Description
	DRV_CAN_ChannelMessageReceive	Receives a message on a channel for the specified driver index. **Implementation:** Static
	DRV_CAN_ChannelMessageTransmit	Transmits a message on a channel for the specified driver index. **Implementation:** Static
	DRV_CAN_Close	Closes the CAN instance for the specified driver index. **Implementation:** Static
	DRV_CAN_Deinitialize	Deinitializes the DRV_CAN_Initialize instance that has been called for the specified driver index. **Implementation:** Static
	DRV_CAN_Initialize	Initializes the CAN instance for the specified driver index. **Implementation:** Static
	DRV_CAN_Open	Opens the CAN instance for the specified driver index. **Implementation:** Static

Description

This section describes the Application Programming Interface (API) functions of the CAN Driver Library.

Function(s)

DRV_CAN_ChannelMessageReceive Function

Receives a message on a channel for the specified driver index.

Implementation: Static

File

help_drv_can.h

C

```
bool DRV_CAN_ChannelMessageReceive(CAN_CHANNEL channelNum, int address, uint8_t DLC, uint8_t* message);
```

Returns

- true - When a message has been received
- false - When a message has not been received

Description

This routine receives data into a buffer from the CAN bus according to the channel, address, and data length given.

Remarks

This routine receives a standard or extended messages based upon the CAN Driver setup.

Preconditions

DRV_CAN_Initialize has been called.

Parameters

Parameters	Description
CAN_CHANNEL channelNum	CAN channel to use
int address	CAN address to receive on
uint8_t DLC	Data Length Code of Message
uint8_t* message	Pointer to put the message data to receive

Function

bool DRV_CAN_ChannelMessageReceive(CAN_CHANNEL channelNum, int address,
uint8_t DLC, uint8_t* message);

DRV_CAN_ChannelMessageTransmit Function

Transmits a message on a channel for the specified driver index.

Implementation: Static

File

help_drv_can.h

C

```
bool DRV_CAN_ChannelMessageTransmit(CAN_CHANNEL channelNum, int address, uint8_t DLC, uint8_t* message);
```

Returns

Boolean "true" when a message has been transmitted.

Description

This routine transmits a data buffer on the CAN bus according to the channel, address, and data length given.

Remarks

This routine receives a standard or extended messages based upon the CAN Driver setup.

Preconditions

DRV_CAN_Initialize has been called.

Parameters

Parameters	Description
CAN_CHANNEL channelNum	CAN channel to use
int address	CAN address to transmit on
uint8_t DLC	Data Length Code of Message
uint8_t* message	Pointer to the message data to send

Function

bool DRV_CAN_ChannelMessageTransmit(CAN_CHANNEL channelNum, int address,
uint8_t DLC, uint8_t* message);

DRV_CAN_Close Function

Closes the CAN instance for the specified driver index.

Implementation: Static

File

help_drv_can.h

C

```
void DRV_CAN_Close();
```

Returns

None.

Description

This routine closes the CAN driver instance for the specified driver instance, making it ready for clients to use it.

Preconditions

DRV_CAN_Initialize has been called.

Function

void DRV_CAN_Close(void)

DRV_CAN_Deinitialize Function

Deinitializes the DRV_CAN_Initialize instance that has been called for the specified driver index.

Implementation: Static

File

help_drv_can.h

C

```
void DRV_CAN_Deinitialize();
```

Returns

None.

Description

This routine deinitializes the CAN Driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Preconditions

None.

Function

void DRV_CAN_Deinitialize(void)

DRV_CAN_Initialize Function

Initializes the CAN instance for the specified driver index.

Implementation: Static

File

help_drv_can.h

C

```
void DRV_CAN_Initialize();
```

Returns

None.

Description

This routine initializes the CAN Driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

This routine must be called before any other CAN routine is called. This routine should only be called once during system initialization.

Preconditions

None.

Function

void DRV_CAN_Initialize(void)

DRV_CAN_Open Function

Opens the CAN instance for the specified driver index.

Implementation: Static

File

help_drv_can.h

C

```
void DRV_CAN_Open();
```

Returns

None.

Description

This routine opens the CAN Driver instance for the specified driver instance, making it ready for clients to use it.

Preconditions

DRV_CAN_Initialize has been called.

Function

void DRV_CAN_Open(void)

Codec Driver Libraries

This section describes the Codec Driver Libraries available in MPLAB Harmony.

AK4384 Codec Driver Library

This topic describes the AK4384 Codec Driver Library.

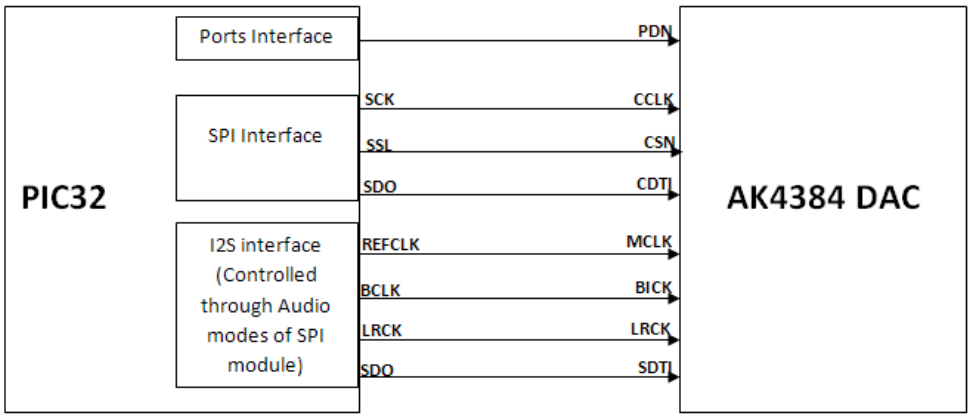
Introduction

This library provides an interface to manage the AK4384 106 dB 192 kHz 24-Bit DAC that is serially interfaced to a Microchip microcontroller for providing Audio Solutions.

Description

The AK4384 module is 24-bit Audio DAC from Asahi Kasei Microdevices Corporation. The AK4384 can be interfaced to Microchip microcontrollers through SPI and I2S serial interfaces. SPI interface is used for control command transfer. The I2S interface is used for Audio data output.

A typical interface of AK4384 to a Microchip PIC32 device is provided in the following diagram:



Features

The AK4384 Codec Driver supports the following features:

- Sampling Rate Ranging from 8 kHz to 192 kHz
- 128 times Oversampling (Normal Speed mode)
- 64 times Oversampling (Double Speed mode)
- 32 times Oversampling (Quad Speed mode)
- Digital de-emphasis for 32k, 44.1k and 48 kHz sampling
- Soft mute
- Digital Attenuator (Linear 256 steps)
- I/F format:
- 24-bit MSB justified
- 24/20/16-bit LSB justified
- I2S
- Master clock:
- 256 fs, 384 fs, 512 fs, 768 fs, or 1152 fs (Normal Speed mode)
- 128 fs, 192 fs, 256 fs, or 384 fs (Double Speed mode)
- 128 fs or 192 fs (Quad Speed mode)

Using the Library

This topic describes the basic architecture of the AK4384 Codec Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_ak4384.h`

The interface to the AK4384 Codec Driver library is defined in the `drv_ak4384.h` header file. Any C language source (`.c`) file that uses the AK4384 Codec Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

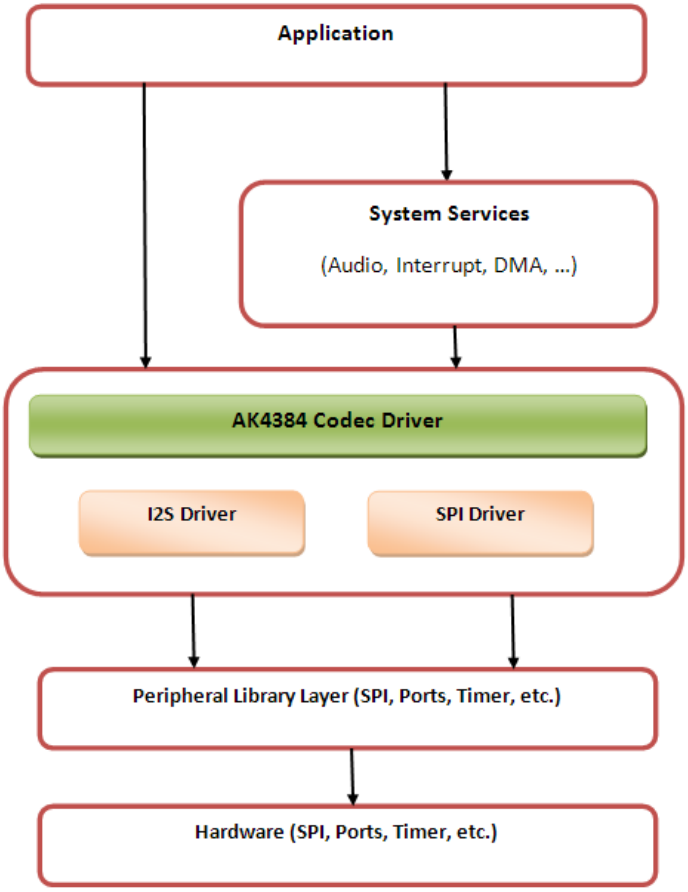
Abstraction Model

This library provides a low-level abstraction of the AK4384 Codec Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The abstraction model shown in the following diagram depicts how the AK4384 Codec Driver is positioned in the MPLAB Harmony framework. The AK4384 Codec Driver uses the SPI and I2S drivers for control and audio data transfers to the AK4384 module.

AK4384 Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The AK4384 Codec Driver Library provides an API interface to transfer control commands and digital audio data to the serially interfaced AK4384 DAC module. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the AK4384 Codec Driver Library.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Setup Functions	Provides open and close functions.
Codec Specific Functions	Provides functions that are Codec-specific.
Data Transfer Functions	Provides data transfer functions.
Other Functions	Provides driver specific miscellaneous functions such as sampling rate setting, control command functions, etc.
Data Types and Constants	These data types and constants are required while interacting and setting up the AK4384 Codec Driver Library.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

System Access

This topic provides information on system initialization, implementations, and provides a system access code example.

Description

System Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the AK4384 module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_AK4384_INIT or by using Initialization Overrides) that are supported by the specific AK4384 device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- SPI driver module index. The module index should be same as the one used in initializing the SPI Driver.
- I2S driver module index. The module index should be same as the one used in initializing the I2S Driver.
- Sampling rate
- Master clock detection mode
- Power down pin port initialization
- Queue size for the audio data transmit buffer

The DRV_AK4384_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_ AK4384_Deinitialize, DRV_ AK4384_Status and DRV_I2S_Tasks.

Implementations

The AK4384 Codec Driver can have the following implementations:

Implementation	Description	MPLAB Harmony Components
Implementation 1	Dedicated hardware for control (SPI) and data (I2S) interface.	Standard MPLAB Harmony drivers for SPI and I2S interfaces.
Implementation 2	Dedicated hardware for data (I2S) interface. Ports pins for control interface.	Standard MPLAB Harmony drivers for I2S interface. Virtual MPLAB Harmony drivers for SPI interface.
Implementation 3	Dedicated hardware for data (I2S) interface. Ports pins for control.	Standard MPLAB Harmony drivers for I2S interface. An internal bit-banged implementation of control interface in the AK4384 Codec Driver.

If Implementation 3 is in use, while initializing fields of DRV_AK4384_INIT structure, the SPI Driver module index initialization is redundant. The user can pass a dummy value.

For Implementation 3, the user has to additionally initialize parameters to support bit-banged control interface implementation. These additional parameters can be passed by assigning values to the respective macros in `system_config.h`.

Example:

```
DRV_AK4384_INIT drvak4384Init =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .volume = 120,
    .mclkMode = DRV_AK4384_MCLK_MODE_MANUAL,
    .queueSizeTransmit = 2,
};

/*
 The SPI module index should be same as the one used in
 initializing the SPI driver.
 The SPI module index initialization is redundant
 if Implementation 3 is in use.
 */
drvak4384Init.spiDriverModuleIndex = DRV_SPI_INDEX_0;

/*
 The I2S module index should be same as the one used in
 initializing the I2S driver.
 */
drvak4384Init.i2sDriverModuleIndex = DRV_I2S_INDEX_0;

ak4384DevObject = DRV_AK4384_Initialize(DRV_AK4384_INDEX_0, (SYS_MODULE_INIT *) &drvak4384Init);
if (SYS_MODULE_OBJ_INVALID == ak4384DevObject)
{
```

```
    // Handle error
}
```

Task Routine

The DRV_AK4384_Tasks will be called from the System Task Service.

Client Access

This topic describes client access and includes a code example.

Description

For the application to start using an instance of the module, it must call the DRV_AK4384_Open function. The DRV_AK4384_Open provides a driver handle to the AK4384 Codec Driver instance for operations. If the driver is deinitialized using the function DRV_AK4384_Deinitialize, the application must call the DRV_AK4384_Open function again to set up the instance of the driver.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

Note: It is necessary to check the status of driver initialization before opening a driver instance. The status of the AK4384 Codec Driver can be known by calling DRV_AK4384_Status.

Example:

```
DRV_HANDLE handle;
SYS_STATUS ak4384Status;
    ak4384Status = DRV_AK4384_Status(sysObjects.ak4384DevObject);
        if (SYS_STATUS_READY == ak4384Status)
        {
                // The driver can now be opened.
                appData.ak4384Client.handle = DRV_AK4384_Open
                                              (DRV_AK4384_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
                if(appData.ak4384Client.handle != DRV_HANDLE_INVALID)
                {
                    appData.state = APP_STATE_AK4384_SET_BUFFER_HANDLER;
                }
                else
                {
                    SYS_DEBUG(0, "Find out what's wrong \r\n");
                }
    }
    else
    {
        /* AK4384 Driver Is not ready */
                ;
    }
```

Client Operations

This topic describes client operations and provides a code example.

Description

Client operations provide the API interface for control command and audio data transfer to the AK4384 Codec.

The following AK4384 Codec specific control command functions are provided:

Notes:

1. The calling and execution of the following functions does not guarantee that the function (and its associated Codec command) has been set in the Codec peer interfaced through the SPI. It just means that the submission of the command has started over the SPI.
2. Regarding Note 1, the user should not call the following functions consecutively, which could result in unexpected behavior. If needed, the user should confirm the completion status of a function before calling any of the other functions.
3. To know the completion status of the following functions, users can register a command event callback handler by calling the function 'DRV_AK4384_CommandEventHandlerSet'. The callback handler will be called when the last submitted command (submitted by calling one of the following functions) has completed.

- DRV_AK4384_SamplingRateSet
- DRV_AK4384_SamplingRateGet
- DRV_AK4384_VolumeSet
- DRV_AK4384_VolumeGet
- DRV_AK4384_MuteOn

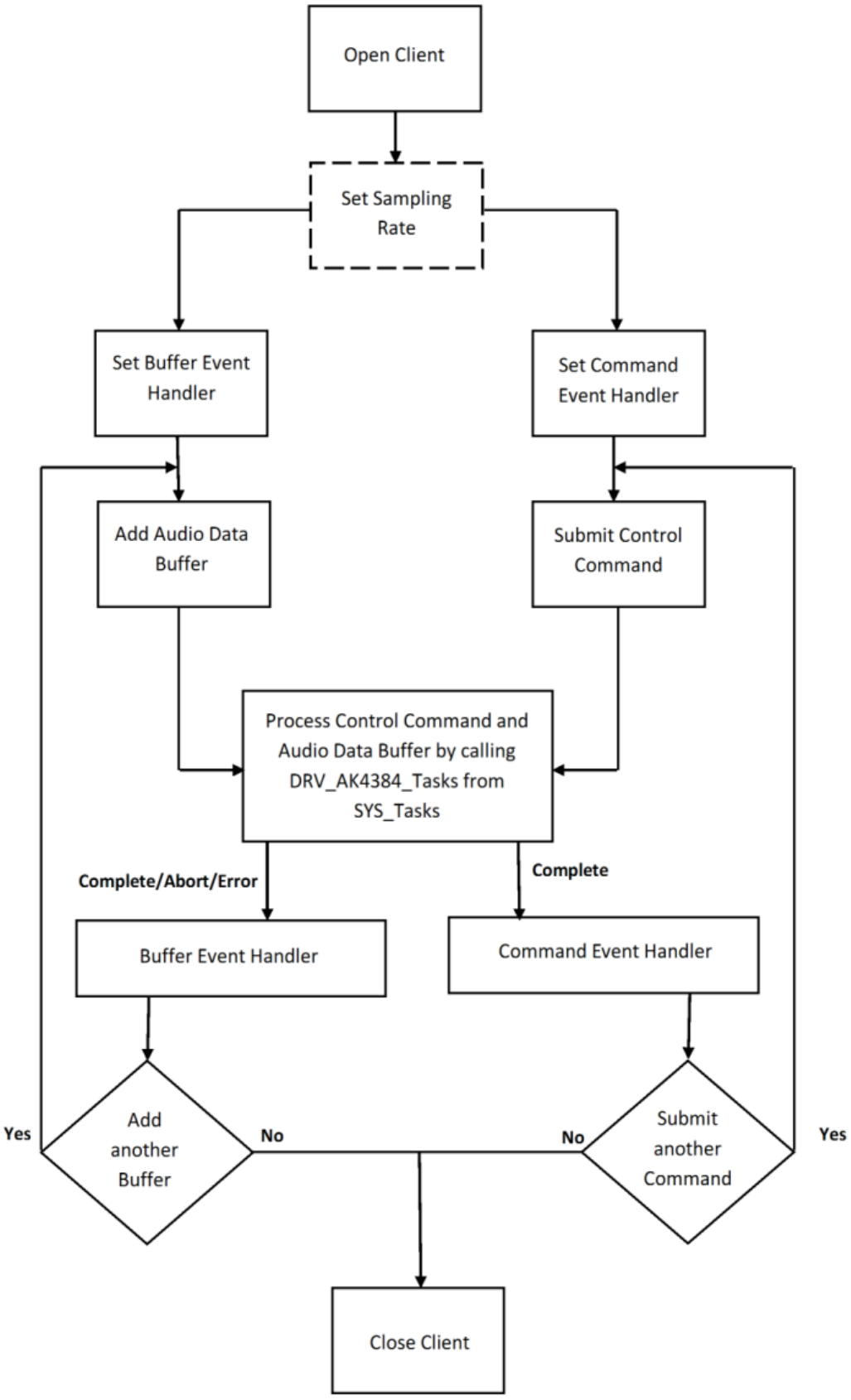
- DRV_AK4384_MuteOff
- DRV_AK4384_ZeroDetectEnable
- DRV_AK4384_ZeroDetectDisable
- DRV_AK4384_ZeroDetectModeSet
- DRV_AK4384_ZeroDetectInvertEnable
- DRV_AK4384_ZeroDetectInvertDisable
- DRV_AK4384_ChannelOutputInvertEnable
- DRV_AK4384_ChannelOutputInvertDisable
- DRV_AK4384_SlowRollOffFilterEnable
- DRV_AK4384_SlowRollOffFilterDisable
- DRV_AK4384_DeEmphasisFilterSet

These functions schedule a non-blocking control command transfer operation. These functions submit the control command request to the AK4384 Codec. A notification for the submitted requests can be received by registering a command callback event with the driver. The driver notifies by calling the callback on successfully transmitting the command to the AK4384 Codec module.

The function DRV_AK4384_BufferAddWrite is a buffered data operation functions. This function schedules non-blocking audio data transfer operation. The function adds the request to the hardware instance queues and returns a buffer handle. The requesting client also registers a callback event with the driver. The driver notifies the client with DRV_AK4384_BUFFER_EVENT_COMPLETE, DRV_AK4384_BUFFER_EVENT_ERROR, or DRV_AK4384_BUFFER_EVENT_ABORT events.

The submitted control commands and audio buffer add requests are processed under DRV_AK4384_Tasks function. This function is called from the SYS_Tasks routine.

The following diagram illustrates the control commands and audio buffered data operations.



Note: It is not necessary to close and reopen the client between multiple transfers.

An application using the buffered functionality needs to perform the following steps:

1. The system should have completed necessary setup and initializations.
2. The I2S Driver object should have been initialized by calling DRV_I2S_Initialize.

3. The SPI Driver object should have been initialized by calling DRV_SPI_Initialize.
4. The AK4384 Codec Driver object should be initialized by calling DRV_AK4384_Initialize.
5. The necessary sampling rate value should be set up by calling DRV_AK4384_ SamplingRateSet.
6. Register buffer event handler for the client handle by calling DRV_AK4384_BufferEventHandlerSet.
7. Register command event handler for the client handle by calling DRV_AK4384_CommandEventHandlerSet.
8. Submit a command by calling specific command API.
9. Add a buffer to initiate the data transfer by calling DRV_AK4384_BufferAddWrite.
10. The submitted command and Audio data processing happens b calling DRV_AK4384_Tasks from SYS_Tasks.
11. Repeat steps 9 through 10 to handle multiple buffer transmission and reception.
12. When the client is done, it can use DRV_AK4384_Close to close the client handle.

Example:

```
typedef enum
{
    APP_STATE_AK4384_OPEN,
    APP_STATE_AK4384_SET_COMMAND_HANDLER,
    APP_STATE_AK4384_SET_BUFFER_HANDLER,
    APP_STATE_AK4384_SET_SAMPLING_RATE_COMMAND,
    APP_STATE_AK4384_ADD_BUFFER,
    APP_STATE_AK4384_WAIT_FOR_BUFFER_COMPLETE,
    APP_STATE_AK4384_BUFFER_COMPLETE
} APP_STATES;

typedef struct
{
    DRV_HANDLE handle;
    DRV_AK4384_BUFFER_HANDLE writeBufHandle;
    DRV_AK4384_BUFFER_EVENT_HANDLER bufferHandler;
    DRV_AK4384_COMMAND_EVENT_HANDLER commandHandler;
    uintptr_t context;
    uint8_t *txbufferObject;
    size_t bufferSize;

} APP_AK4384_CLIENT;

typedef struct
{
    /* Application's current state*/
    APP_STATES state;
    /* USART client handle */
    APP_AK4384_CLIENT ak4384Client;
} APP_DATA;
APP_DATA appData;
SYS_MODULE_OBJ ak4384DevObject;
DRV_AK4384_INIT drvak4384Init =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .volume = 120,
    .mclkMode = DRV_AK4384_MCLK_MODE_MANUAL,
    .queueSizeTransmit = 2,
};

void SYS_Initialize(void * data)
{
    /*
     The SPI module index should be same as the one used in
     initializing the SPI driver.
     The SPI module index initialization is redundant
     if Implementation 3 (Described in System Access) is in use.
     */
    drvak4384Init.spiDriverModuleIndex = DRV_SPI_INDEX_0;

    /*
     The I2S module index should be same as the one used in
     initializing the I2S driver.
     */
    drvak4384Init.i2sDriverModuleIndex = DRV_I2S_INDEX_0;
```

```
    ak4384DevObject = DRV_AK4384_Initialize(DRV_AK4384_INDEX_0, (SYS_MODULE_INIT *) & drvak4384Init);
    if (SYS_MODULE_OBJ_INVALID == ak4384DevObject) {
        // Handle error
    }
}

void APP_Tasks (void )
{
    switch(appData.state)
    {
        /* Open the ak4384 client and get an Handle */
        case APP_STATE_AK4384_OPEN:
        {
            SYS_STATUS ak4384Status;
            ak4384Status = DRV_AK4384_Status(sysObjects.ak4384DevObject);
            if (SYS_STATUS_READY == ak4384Status)
            {
                // This means the driver can now be opened.
                appData.ak4384Client.handle = DRV_AK4384_Open(DRV_AK4384_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
                if(appData.ak4384Client.handle != DRV_HANDLE_INVALID)
                {
                    appData.state = APP_STATE_AK4384_SET_COMMAND_HANDLER;
                }
                else
                {
                    SYS_DEBUG(0, "Find out what is wrong \r\n");
                }
            }
            else
            {
                /* Wait for AK4384 to Initialize */
                ;
            }
        }
        break;

        /* Register a command event handler */
        case APP_STATE_AK4384_SET_COMMAND_HANDLER:
        {
            DRV_AK4384_CommandEventHandlerSet(appData.ak4384Client.handle,
                    appData.ak4384Client.commandHandler,
                    appData.ak4384Client.context);
            appData.state = APP_STATE_AK4384_SET_BUFFER_HANDLER;
        }
        break;


        /* Register a buffer event handler */
        case APP_STATE_AK4384_SET_BUFFER_HANDLER:
        {
            DRV_AK4384_BufferEventHandlerSet(appData.ak4384Client.handle,
                    appData.ak4384Client.bufferHandler,
                    appData.ak4384Client.context);
            appData.state = APP_STATE_AK4384_SET_SAMPLING_RATE_COMMAND;
        }
        break;

        /* Submit a set sampling rate command */
        case APP_STATE_AK4384_SET_SAMPLING_RATE_COMMAND:
        {
            DRV_AK4384_SamplingRateSet(appData.ak4384Client.handle,48000);
            appData.state = APP_STATE_AK4384_ADD_BUFFER;
        }
        break;

        /* Add the Audio buffer to be transmitted */
        case APP_STATE_AK4384_ADD_BUFFER:
        {
            DRV_AK4384_BufferAddWrite(appData.ak4384Client.handle, &appData.ak4384Client.writeBufHandle,
```

```
            appData.ak4384Client.txbufferObject, appData.ak4384Client.bufferSize);
            if(appData.ak4384Client.writeBufHandle != DRV_AK4384_BUFFER_HANDLE_INVALID)
            {
                appData.state = APP_STATE_AK4384_WAIT_FOR_BUFFER_COMPLETE;
            }
            else
            {
                SYS_DEBUG(0, "Find out what is wrong \r\n");
            }
        }
        break;

    /* Audio Buffer transmission under process */
        case APP_STATE_AK4384_WAIT_FOR_BUFFER_COMPLETE:
        {
        }
        break;

    /* Audio Buffer transmission completed */
        case APP_STATE_AK4384_BUFFER_COMPLETE:
        {
            /* Add another buffer */
            appData.state = APP_STATE_AK4384_ADD_BUFFER;
        }
        break;

        default:
        {
        }
        break;
    }

}

void APP_AK4384CommandEventHandler(uintptr_t context )
{

    // Last submitted command successful. Take action as needed.
}

void APP_AK4384BufferEventHandler(DRV_AK4384_BUFFER_EVENT event,
        DRV_AK4384_BUFFER_HANDLE handle, uintptr_t context )
{

    switch(event)
    {
        case DRV_AK4384_BUFFER_EVENT_COMPLETE:
        {
            // Can set appData.state = APP_STATE_AK4384_BUFFER_COMPLETE;
        // Take Action as needed

        }
        break;
        case DRV_AK4384_BUFFER_EVENT_ERROR:
        {
        // Take Action as needed

        } break;

        case DRV_AK4384_BUFFER_EVENT_ABORT:
        {
        // Take Action as needed

        } break;

    }
}

void SYS_Tasks(void)
```

```
{
    DRV_AK4384_Tasks(ak4384DevObject);
    APP_Tasks();
}
```

Configuring the Library

Macros

	Name	Description
	DRV_AK4384_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4384_CONTROL_CLOCK	Sets up clock frequency for the control interface (SPI)
	DRV_AK4384_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4384_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4384_TIMER_DRIVER_MODULE_INDEX	Identifies the Timer Module Index for custom virtual SPI driver implementation.
	DRV_AK4384_TIMER_PERIOD	Identifies the period for the bit bang timer.
	DRV_AK4384_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1, and 48K sampling frequency
	DRV_AK4384_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1 and 48K sampling frequency

Description

The configuration of the AK4384 Codec Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the AK4384 Codec Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the AK4384 Codec Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_AK4384_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_CLIENTS_NUMBER DRV_AK4384_INSTANCES_NUMBER
```

Description

AK4384 Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. Typically only one client could be connected to one hardware instance. This value represents the total number of clients to be supported across all hardware instances. Therefore, if there are five AK4384 hardware interfaces, this number will be 5.

Remarks

None.

DRV_AK4384_CONTROL_CLOCK Macro

Sets up clock frequency for the control interface (SPI)

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_CONTROL_CLOCK
```

Description

AK4384 Control Interface Clock Speed configuration

Sets up clock frequency for the control interface (SPI). The maximum value supported is 5MHZ.

Remarks

1. This Macro is useful only when a hardware SPI module is not available(used) or a virtual SPI driver is not available(used) for the control interface to the AK4384 CODEC.
2. This constant needs to defined only for a bit banged implementation of control interface with in the driver.

DRV_AK4384_INPUT_REFCLOCK Macro

Identifies the input REFCLOCK source to generate the MCLK to codec.

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_INPUT_REFCLOCK
```

Description

AK4384 Input reference clock

Identifies the input REFCLOCK source to generate the MCLK to codec.

Remarks

None.

DRV_AK4384_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_INSTANCES_NUMBER
```

Description

AK4384 driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of AK4384 CODEC modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None.

DRV_AK4384_TIMER_DRIVER_MODULE_INDEX Macro

Identifies the Timer Module Index for custom virtual SPI driver implementation.

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_TIMER_DRIVER_MODULE_INDEX
```

Description

AK4384 Timer Module Index

Identifies the Timer Module Index for custom virtual SPI driver implementation. The AK4384 uses SPI protocol for control interface. The Timer Module Index is needed by AK4384 driver to implement a virtual SPI driver for control command exchange with the AK4384 CODEC.

Remarks

1. This Macro is useful only when a hardware SPI module is not available(used) or a virtual SPI driver is not available(used) for the control interface to the AK4384 CODEC.
2. This constant needs to defined only for a bit banged implementation of control interface with in the driver.

DRV_AK4384_TIMER_PERIOD Macro

Identifies the period for the bit bang timer.

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_TIMER_PERIOD
```

Description

AK4384 Timer Period

Identifies the period for the bit bang timer after which the timer interrupt should occur. The value assigned should align with the expected control interface clock defined by AK4384_CONTROL_CLOCK.

Remarks

1. This Macro is useful only when a hardware SPI module is not available(used) or a virtual SPI driver is not available(used) for the control interface to the AK4384 CODEC.
2. This constant needs to defined only for a bit banged implementation of control interface with in the driver.

DRV_AK4384_BCLK_BIT_CLK_DIVISOR Macro

Sets up the BCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1, and 48K sampling frequency

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_BCLK_BIT_CLK_DIVISOR
```

Description

AK4384 BCLK to LRCK Ratio to Generate Audio Stream

Sets up the BCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1 and 48K I2S sampling frequency

Following BCLK to LRCK ratios are supported 16bit LSB Justified >=32fs 20bit LSB Justified >=40fs 24bit MSB Justified >=48fs 24bit I2S Compatible >=48fs 24bit LSB Justified >=48fs

Typical values for the divisor are 1,2,4 and 8

Remarks

None.

DRV_AK4384_MCLK_SAMPLE_FREQ_MULTPLIER Macro

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1 and 48K sampling frequency

File

drv_ak4384_config_template.h

C

```
#define DRV_AK4384_MCLK_SAMPLE_FREQ_MULTPLIER
```

Description

AK4384 MCLK to LRCK Ratio to Generate Audio Stream

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1, and 48K I2S sampling frequency

Supported MCLK to LRCK Ratios are as below 256fs, 384fs, 512fs, 768fs or 1152fs [Normal Speed Mode(8kHz~48kHz)] 128fs, 192fs, 256fs or 384fs [Double Speed Mode(60kHz~96kHz)] 128fs, 192fs [Quad Speed Mode(120kHz~192kHz)]

Remarks

None

Configuring the MHC

Provides examples on how to configure the MPLAB Harmony Configurator (MHC) for a specific driver.

Description

The following three figures show examples of MHC configurations for the AK4384 Codec Driver, I2S Driver, and the Timer Driver.

Figure 1: AK4384 Codec Driver MHC Configuration

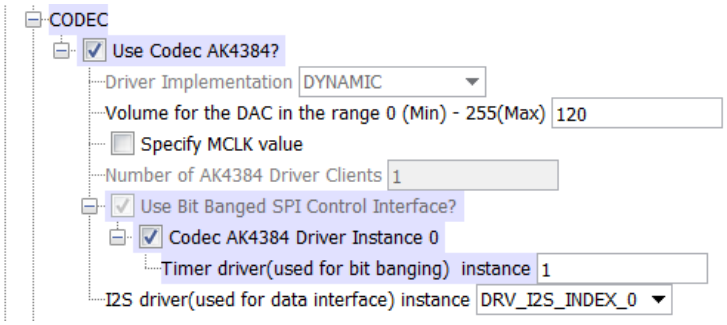


Figure 2: I2S Driver MHC Configuration

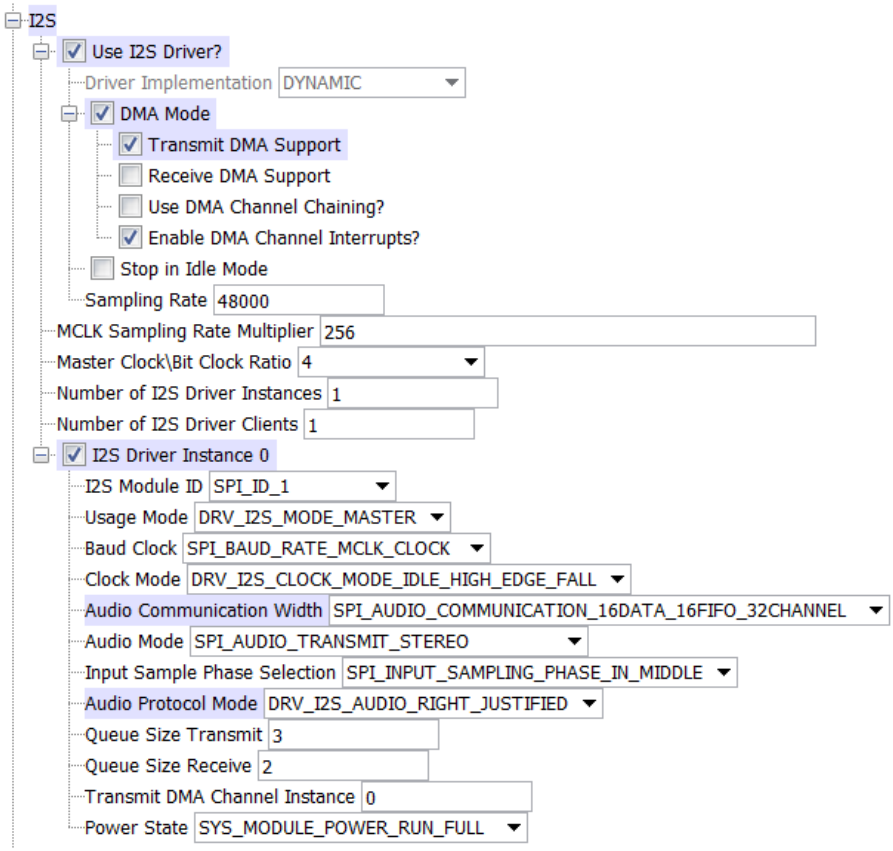
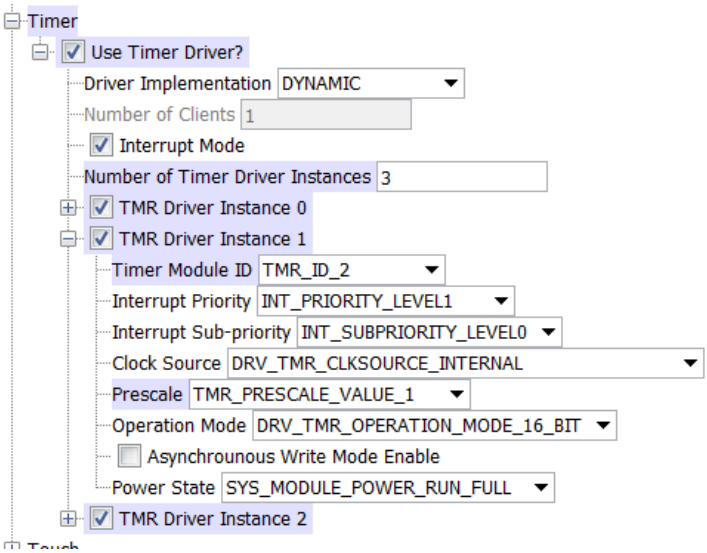


Figure 3: Timer Driver MHC Configuration



Building the Library

This section lists the files that are available in the AK4384 Codec Driver Library.

Description

This section list the files that are available in the `/src` folder of the AK4384 Codec Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/codec/ak4384`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ak4384.h`	Header file that exports the driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_ak4384_bit_banged_control_interface.c`	This file contains implementation of the AK4384 Codec Driver with a custom bit-banged implementation for control interface driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
`/src/dynamic/drv_ak4384_virtual_control_interface.c`	This file contains implementation of the AK4384 Codec Driver with a virtual SPI driver as control interface driver. **Note:** This file is currently unsupported.
`/src/dynamic/drv_ak4384.c`	This file contains the core implementation of the AK4384 Codec Driver **Note:** This file currently unsupported.

Module Dependencies

The AK4384 Driver Library depends on the following modules:

- I2S Driver Library
- SPI Driver Library
- Timer Driver Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_AK4384_Initialize	Initializes hardware and data for the instance of the AK4384 DAC module. **Implementation:** Dynamic
	DRV_AK4384_Deinitialize	Deinitializes the specified instance of the AK4384 driver module. **Implementation:** Dynamic
	DRV_AK4384_Status	Gets the current status of the AK4384 driver module. **Implementation:** Dynamic
	DRV_AK4384_Tasks	Maintains the driver's control and data interface state machine. **Implementation:** Dynamic

b) Client Setup Functions

	Name	Description
	DRV_AK4384_Open	Opens the specified AK4384 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_AK4384_Close	Closes an opened-instance of the AK4384 driver. **Implementation:** Dynamic

c) Codec Specific Functions

	Name	Description
	DRV_AK4384_ChannelOutputInvertDisable	Disables output polarity of the selected Channel. **Implementation:** Dynamic
	DRV_AK4384_ChannelOutputInvertEnable	Enables output polarity of the selected channel. **Implementation:** Dynamic
	DRV_AK4384_DeEmphasisFilterSet	Allows specifies enabling of digital de-emphasis filter. **Implementation:** Dynamic
	DRV_AK4384_MuteOff	Disables AK4384 output for soft mute. **Implementation:** Dynamic
	DRV_AK4384_MuteOn	Allows AK4384 output for soft mute on. **Implementation:** Dynamic
	DRV_AK4384_SamplingRateGet	This function gets the sampling rate set on the DAC AK4384. **Implementation:** Dynamic
	DRV_AK4384_SamplingRateSet	This function sets the sampling rate of the media stream. **Implementation:** Dynamic
	DRV_AK4384_SlowRollOffFilterDisable	Disables Slow Roll-off filter function. **Implementation:** Dynamic
	DRV_AK4384_SlowRollOffFilterEnable	Enables Slow Roll-off filter function. **Implementation:** Dynamic
	DRV_AK4384_VolumeGet	This function gets the volume for AK4384 Codec. **Implementation:** Dynamic
	DRV_AK4384_VolumeSet	This function sets the volume for AK4384 Codec. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectDisable	Disables AK4384 channel-independent zeros detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectEnable	Enables AK4384 channel-independent zeros detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectInvertDisable	Disables inversion of polarity for zero detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectInvertEnable	Enables inversion of polarity for zero detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectModeSet	Sets mode of AK4384 channel-independent zeros detect function. **Implementation:** Dynamic

d) Data Transfer Functions

	Name	Description
	DRV_AK4384_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic
	DRV_AK4384_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Dynamic
	DRV_AK4384_BufferCombinedQueueSizeGet	This function returns the number of bytes queued (to be processed) in the buffer queue. **Implementation:** Dynamic
	DRV_AK4384_BufferProcessedSizeGet	This function returns number of bytes that have been processed for the specified buffer. **Implementation:** Dynamic

e) Other Functions

	Name	Description
	DRV_AK4384_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished. **Implementation:** Dynamic
	DRV_AK4384_VersionGet	Returns the version of the AK4384 driver. **Implementation:** Dynamic
	DRV_AK4384_VersionStrGet	Returns the version of AK4384 driver in string format. **Implementation:** Dynamic
	DRV_AK4384_BufferQueueFlush	This function flushes off the buffers associated with the client object. **Implementation:** Dynamic

f) Data Types and Constants

	Name	Description
	DRV_AK4384_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4384_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4384_BUFFER_EVENT_HANDLER	Pointer to a AK4384 Driver Buffer Event handler function.
	DRV_AK4384_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4384_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4384_COMMAND_EVENT_HANDLER	Pointer to a AK4384 Driver Command Event Handler Function
	DRV_AK4384_DEEMPHASIS_FILTER	Identifies de-emphasis filter function.
	DRV_AK4384_INIT	Defines the data required to initialize or reinitialize the AK4384 driver.
	DRV_AK4384_MCLK_MODE	Identifies the mode of master clock to AK4384 DAC.
	DRV_AK4384_ZERO_DETECT_MODE	Identifies Zero Detect Function mode
	DRV_AK4384_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4384_COUNT	Number of valid AK4384 driver indices.
	DRV_AK4384_INDEX_0	AK4384 driver index definitions.
	DRV_AK4384_INDEX_1	This is macro DRV_AK4384_INDEX_1.
	DRV_AK4384_INDEX_2	This is macro DRV_AK4384_INDEX_2.
	DRV_AK4384_INDEX_3	This is macro DRV_AK4384_INDEX_3.
	DRV_AK4384_INDEX_4	This is macro DRV_AK4384_INDEX_4.
	DRV_AK4384_INDEX_5	This is macro DRV_AK4384_INDEX_5.

Description

This section describes the API functions of the AK4384 Codec Driver library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_AK4384_Initialize Function

Initializes hardware and data for the instance of the AK4384 DAC module.

Implementation: Dynamic

File

drv_ak4384.h

C

```
SYS_MODULE_OBJ DRV_AK4384_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the AK4384 driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the 'init' parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized.

Remarks

This routine must be called before any other AK4384 routine is called.

This routine should only be called once during system initialization unless DRV_AK4384_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

DRV_I2S_Initialize must be called before calling this function to initialize the data interface of this CODEC driver. DRV_SPI_Initialize must be called if SPI driver is used for handling the control interface of this CODEC driver.

Example

```
DRV_AK4384_INIT                init;
SYS_MODULE_OBJ                 objectHandle;

init.moduleInit.value          = SYS_MODULE_POWER_RUN_FULL;
init.spiDriverModuleIndex      = DRV_SPI_INDEX_0;  // This will be ignored for a custom
                                                   // control interface driver implementation
init.i2sDriverModuleIndex      = DRV_I2S_INDEX_0;
init.mclkMode                  = DRV_AK4384_MCLK_MODE_MANUAL;
init.audioDataFormat           = DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_I2S;
init.powerDownPortChannel      = PORT_CHANNEL_G;
init.powerDownBitPosition      = PORTS_BIT_POS_15;

objectHandle = DRV_AK4384_Initialize(DRV_AK4384_0, (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to be initialized
init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

```
SYS_MODULE_OBJ  DRV_AK4384_Initialize
(
const       SYS_MODULE_INDEX drvIndex,
const       SYS_MODULE_INIT *const init
);
```

DRV_AK4384_Deinitialize Function

Deinitializes the specified instance of the AK4384 driver module.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the AK4384 driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_AK4384_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_AK4384_Initialize
SYS_STATUS          status;


DRV_AK4384_Deinitialize(object);

status = DRV_AK4384_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4384_Initialize routine

Function

void DRV_AK4384_Deinitialize(SYS_MODULE_OBJ object)

DRV_AK4384_Status Function

Gets the current status of the AK4384 driver module.

Implementation: Dynamic

File

drv_ak4384.h

C

```
SYS_STATUS DRV_AK4384_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized

SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed

SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed

SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This routine provides the current status of the AK4384 driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_AK4384_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4384_Initialize
SYS_STATUS          ak4384Status;

ak4384Status = DRV_AK4384_Status(object);
if (SYS_STATUS_READY == ak4384Status)
{
    // This means the driver can be opened using the
    // DRV_AK4384_Open function.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4384_Initialize routine

Function

SYS_STATUS DRV_AK4384_Status(SYS_MODULE_OBJ object)

DRV_AK4384_Tasks Function

Maintains the driver's control and data interface state machine.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal control and data interface state machine and implement its control and data interface implementations. This function should be called from the SYS_Tasks function.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4384_Initialize

while (true)
{
    DRV_AK4384_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_AK4384_Initialize)

Function

void DRV_AK4384_Tasks(SYS_MODULE_OBJ object);

b) Client Setup Functions

DRV_AK4384_Open Function

Opens the specified AK4384 driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_ak4384.h

C

```
DRV_HANDLE DRV_AK4384_Open(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Errors can occur under following conditions:

- if the number of client objects allocated via DRV_AK4384_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid
- if the ioIntent options passed are not relevant to this driver

Description

This routine opens the specified AK4384 driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options are not relevant to this driver. All the data transfer functions of this driver are non blocking.

Only DRV_IO_INTENT_WRITE is a valid ioIntent option as AK4384 is DAC only.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_AK4384_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

Function DRV_AK4384_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_AK4384_Open(DRV_AK4384_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

DRV_HANDLE DRV_AK4384_Open

(

const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT ioIntent
)

DRV_AK4384_Close Function

Closes an opened-instance of the AK4384 driver.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This routine closes an opened-instance of the AK4384 driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_AK4384_Open before the caller may use the driver again

Remarks

Usually there is no need for the driver client to verify that the Close operation has completed. The driver will abort any ongoing operations when this routine is called.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_AK4384_Open

DRV_AK4384_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_Close(DRV_Handle handle)

c) Codec Specific Functions

DRV_AK4384_ChannelOutputInvertDisable Function

Disables output polarity of the selected Channel.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ChannelOutputInvertDisable(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan);
```

Returns

None.

Description

This function disables output polarity of the selected Channel.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ChannelOutputInvertDisable(myAK4384Handle, DRV_AK4384_CHANNEL_LEFT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
chan	Left or Right channel

Function

void DRV_AK4384_ChannelOutputInvertDisable(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan)

DRV_AK4384_ChannelOutputInvertEnable Function

Enables output polarity of the selected channel.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ChannelOutputInvertEnable(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan);
```

Returns

None.

Description

This function enables output polarity of the selected channel.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ChannelOutputInvertEnable(myAK4384Handle, DRV_AK4384_CHANNEL_LEFT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
chan	Left or Right channel

Function

void DRV_AK4384_ChannelOutputInvertEnable(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan)

DRV_AK4384_DeEmphasisFilterSet Function

Allows specifies enabling of digital de-emphasis filter.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_DeEmphasisFilterSet(DRV_HANDLE handle, DRV_AK4384_DEEMPHASIS_FILTER filter);
```

Returns

None.

Description

This function allows specifies enabling of digital de-emphasis for 32, 44.1 or 48 kHz sampling rates (tc = 50/15 µs)

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_DeEmphasisFilterSet(myAK4384Handle, DRV_AK4384_DEEMPHASIS_FILTER_44_1KHZ)
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
filter	Specifies Enable of de-emphasis filter

Function

void DRV_AK4384_DeEmphasisFilterSet

(

DRV_HANDLE handle,

DRV_AK4384_DEEMPHASIS_FILTER filter

)

DRV_AK4384_MuteOff Function

Disables AK4384 output for soft mute.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_MuteOff(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables AK4384 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

    DRV_AK4384_MuteOff(myAK4384Handle); //AK4384 output soft mute disabled
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_MuteOff(DRV_HANDLE handle)

DRV_AK4384_MuteOn Function

Allows AK4384 output for soft mute on.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_MuteOn(DRV_HANDLE handle);
```

Returns

None.

Description

This function Enables AK4384 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_MuteOn(myAK4384Handle);  //AK4384 output soft muted
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_MuteOn(DRV_HANDLE handle);

DRV_AK4384_SamplingRateGet Function

This function gets the sampling rate set on the DAC AK4384.

Implementation: Dynamic

File

drv_ak4384.h

C

```
uint32_t DRV_AK4384_SamplingRateGet(DRV_HANDLE handle);
```

Returns

None.

Description

This function gets the sampling rate set on the DAC AK4384.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
uint32_t baudRate;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

baudRate = DRV_AK4384_SamplingRateGet(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_AK4384_SamplingRateGet(DRV_HANDLE handle)

DRV_AK4384_SamplingRateSet Function

This function sets the sampling rate of the media stream.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate);
```

Returns

None.

Description

This function sets the media sampling rate for the client handle.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_SamplingRateSet(myAK4384Handle, 48000);  //Sets 48000 media sampling rate
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
baudRate	Baud Rate to be set

Function

void DRV_AK4384_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate)

DRV_AK4384_SlowRollOffFilterDisable Function

Disables Slow Roll-off filter function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_SlowRollOffFilterDisable(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables Slow Roll-off filter function. Sharp Roll-off filter function gets enabled.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_SlowRollOffFilterDisable(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_SlowRollOffFilterDisable(DRV_HANDLE handle);

DRV_AK4384_SlowRollOffFilterEnable Function

Enables Slow Roll-off filter function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_SlowRollOffFilterEnable(DRV_HANDLE handle);
```

Returns

None.

Description

This function enables Slow Roll-off filter function.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_SlowRollOffFilterEnable(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_SlowRollOffFilterEnable(DRV_HANDLE handle);

DRV_AK4384_VolumeGet Function

This function gets the volume for AK4384 Codec.

Implementation: Dynamic

File

drv_ak4384.h

C

```
uint8_t DRV_AK4384_VolumeGet(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan);
```

Returns

None.

Description

This functions gets the current volume programmed to the DAC AK4384.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
uint8_t volume;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

  volume = DRV_AK4384_VolumeGet(myAK4384Handle, DRV_AK4384_CHANNEL_LEFT_RIGHT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
chan	Audio channel volume to get.

Function

uint8_t DRV_AK4384_VolumeGet(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan)

DRV_AK4384_VolumeSet Function

This function sets the volume for AK4384 Codec.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_VolumeSet(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan, uint8_t volume);
```

Returns

None.

Description

This functions sets the volume value from 0-255, which can attenuate from 0 dB to –48 dB and mute.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

    DRV_AK4384_VolumeSet(myAK4384Handle, DRV_AK4384_CHANNEL_LEFT_RIGHT, 120); //Step 120 volume
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

chan	Audio channel volume to be set
volume	volume value from 0-255, which can attenuate from 0 dB to –48 dB and mute

Function

void DRV_AK4384_VolumeSet(DRV_HANDLE handle, DRV_AK4384_CHANNEL chan, uint8_t volume)

DRV_AK4384_ZeroDetectDisable Function

Disables AK4384 channel-independent zeros detect function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ZeroDetectDisable(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables AK4384 channel-independent zeros detect function.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ZeroDetectDisable(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_ZeroDetectDisable(DRV_HANDLE handle)

DRV_AK4384_ZeroDetectEnable Function

Enables AK4384 channel-independent zeros detect function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ZeroDetectEnable(DRV_HANDLE handle);
```

Returns

None.

Description

This function enables AK4384 channel-independent zeros detect function.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ZeroDetectEnable(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_ZeroDetectEnable(DRV_HANDLE handle)

DRV_AK4384_ZeroDetectInvertDisable Function

Disables inversion of polarity for zero detect function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ZeroDetectInvertDisable(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables inversion of polarity for zero detect function. DZF goes “H” at Zero Detection.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ZeroDetectInvertDisable(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_ZeroDetectInvertDisable(DRV_HANDLE handle)

DRV_AK4384_ZeroDetectInvertEnable Function

Enables inversion of polarity for zero detect function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ZeroDetectInvertEnable(DRV_HANDLE handle);
```

Returns

None.

Description

This function enables inversion of polarity for zero detect function. DZF goes “L” at Zero Detection

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ZeroDetectInvertEnable(myAK4384Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4384_ZeroDetectInvertEnable(DRV_HANDLE handle)

DRV_AK4384_ZeroDetectModeSet Function

Sets mode of AK4384 channel-independent zeros detect function.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_ZeroDetectModeSet(DRV_HANDLE handle, DRV_AK4384_ZERO_DETECT_MODE zdMode);
```

Returns

None.

Description

This function sets mode of AK4384 channel-independent zeros detect function

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

DRV_AK4384_ZeroDetectModeSet(myAK4384Handle, DRV_AK4384_ZERO_DETECT_MODE_ANDED);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
zdMode	Specifies zero detect function mode.

Function

void DRV_AK4384_ZeroDetectModeSet

(

DRV_HANDLE handle,

DRV_AK4384_ZERO_DETECT_MODE zdMode

)

d) Data Transfer Functions

DRV_AK4384_BufferAddWrite Function

Schedule a non-blocking driver write operation.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_BufferAddWrite(const DRV_HANDLE handle, DRV_AK4384_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4384_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4384_BUFFER_HANDLE_INVALID if:

- a buffer could not be allocated to the request
- the input buffer pointer is NULL
- the buffer size is '0'
- the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4384_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4384_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4384 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4384 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 device instance and the DRV_AK4384_Status must have returned SYS_STATUS_READY.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE must have been specified in the DRV_AK4384_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4384_BUFFER_HANDLE bufferHandle;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

// Client registers an event handler with driver

DRV_AK4384_BufferEventHandlerSet(myAK4384Handle,
                APP_AK4384BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4384_BufferAddWrite(myAK4384handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4384_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4384BufferEventHandler(DRV_AK4384_BUFFER_EVENT event,
        DRV_AK4384_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4384_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4384_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the AK4384 instance as return by the DRV_AK4384_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK4384_BufferAddWrite
(
const DRV_HANDLE handle,
DRV_AK4384_BUFFER_HANDLE *bufferHandle,
void *buffer, size_t size
)

DRV_AK4384_BufferEventHandlerSet Function

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_BufferEventHandlerSet(DRV_HANDLE handle, const DRV_AK4384_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls DRV_AK4384_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4384_BUFFER_HANDLE bufferHandle;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

// Client registers an event handler with driver

DRV_AK4384_BufferEventHandlerSet(myAK4384Handle,
                APP_AK4384BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4384_BufferAddWrite(myAK4384handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4384_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4384BufferEventHandler(DRV_AK4384_BUFFER_EVENT event,
        DRV_AK4384_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
```

```
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4384_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4384_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4384_BufferEventHandlerSet

(

DRV_HANDLE handle,

const DRV_AK4384_BUFFER_EVENT_HANDLER eventHandler,

const uintptr_t contextHandle

)

DRV_AK4384_BufferCombinedQueueSizeGet Function

This function returns the number of bytes queued (to be processed) in the buffer queue.

Implementation: Dynamic

File

drv_ak4384.h

C

```
size_t DRV_AK4384_BufferCombinedQueueSizeGet(DRV_HANDLE handle);
```

Returns

Returns the number of the bytes that have been processed for this buffer. Returns 0 for an invalid or an expired client handle.

Description

This function returns the number of bytes queued (to be processed) in the buffer queue associated with the driver instance to which the calling client belongs. The client can use this function to know number of bytes that is in the queue to be transmitted.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

One of DRV_AK4384_BufferAddRead/DRV_AK4384_BufferAddWrite function must have been called and buffers should have been queued for transmission.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
size_t bufferQueuedSize;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4384_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_AK4384_Open function.

// Client registers an event handler with driver. This is done once

DRV_AK4384_BufferEventHandlerSet(myAK4384Handle, APP_AK4384BufferEventHandle,
                                                    (uintptr_t)&myAppObj);

DRV_AK4384_BufferAddRead(myAK4384handle,&bufferHandle,
                                  myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4384_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// The data is being processed after adding the buffer to the queue.
// The user can get to know dynamically available data in the queue to be
// transmitted by calling DRV_AK4384_BufferCombinedQueueSizeGet
bufferQueuedSize = DRV_AK4384_BufferCombinedQueueSizeGet(myAK4384Handle);
```

Parameters

Parameters	Description
handle	Opened client handle associated with a driver object.

Function

size_t DRV_AK4384_BufferCombinedQueueSizeGet(DRV_HANDLE handle)

DRV_AK4384_BufferProcessedSizeGet Function

This function returns number of bytes that have been processed for the specified buffer.

Implementation: Dynamic

File

drv_ak4384.h

C

```
size_t DRV_AK4384_BufferProcessedSizeGet(DRV_AK4384_BUFFER_HANDLE bufferHandle);
```

Returns

Returns the number of the bytes that have been processed for this buffer. Returns 0 for an invalid or an expired buffer handle.

Description

This function returns number of bytes that have been processed for the specified buffer. The client can use this function, in a case where the buffer has terminated due to an error, to obtain the number of bytes that have been processed. If this function is called on a invalid buffer handle, or if the buffer handle has expired, the function returns 0.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified I2S driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

One of DRV_AK4384_BufferAddRead, DRV_AK4384_BufferAddWrite function must have been called and a valid buffer handle returned.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4384_BUFFER_HANDLE bufferHandle;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

// Client registers an event handler with driver. This is done once

DRV_AK4384_BufferEventHandlerSet(myAK4384Handle, APP_AK4384BufferEventHandle,
                                                    (uintptr_t)&myAppObj);

DRV_AK4384_BufferAddRead(myAK4384handle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4384_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_AK4384BufferEventHandler(DRV_AK4384_BUFFER_EVENT event,
        DRV_AK4384_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_AK4384_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4384_BUFFER_EVENT_ERROR:

            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred.

            processedBytes = DRV_AK4384_BufferProcessedSizeGet(bufferHandle);

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferhandle	Handle of the buffer of which the processed number of bytes to be obtained.

Function

size_t DRV_AK4384_BufferProcessedSizeGet(DRV_AK4384_BUFFER_HANDLE bufferHandle)

e) Other Functions

DRV_AK4384_CommandEventHandlerSet Function

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_CommandEventHandlerSet(DRV_HANDLE handle, const DRV_AK4384_COMMAND_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

When a client calls DRV_AK4384_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "AK4384 CODEC Specific Client Routines" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the command has completed, it does not need to register a callback.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

// Client registers an event handler with driver

DRV_AK4384_CommandEventHandlerSet(myAK4384Handle,
                APP_AK4384CommandEventHandler, (uintptr_t)&myAppObj);

DRV_AK4384_DeEmphasisFilterSet(myAK4384Handle, DRV_AK4384_DEEMPHASIS_FILTER_44_1KHZ)

// Event is received when
// the buffer is processed.

void APP_AK4384CommandEventHandler(uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
            // Last Submitted command is completed.
            // Perform further processing here
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.

context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4384_CommandEventHandlerSet
(
DRV_HANDLE handle,
const DRV_AK4384_COMMAND_EVENT_HANDLER eventHandler,
const uintptr_t contextHandle
)

DRV_AK4384_VersionGet Function

Returns the version of the AK4384 driver.

Implementation: Dynamic

File

drv_ak4384.h

C

```
uint32_t DRV_AK4384_VersionGet();
```

Returns

Returns the version of AK4384 driver.

Description

The version number returned from the DRV_AK4384_VersionGet function is an unsigned integer in the following decimal format. * 10000 + * 100 + Where the numbers are represented in decimal and the meaning is the same as above. Note that there is no numerical representation of release type.

Remarks

None.

Example 1

For version "0.03a", return: 0 * 10000 + 3 * 100 + 0 For version "1.00", return: 1 * 100000 + 0 * 100 + 0

Example 2

```
uint32_t ak4384version;
ak4384version = DRV_AK4384_VersionGet();
```

Function

uint32_t DRV_AK4384_VersionGet(void)

DRV_AK4384_VersionStrGet Function

Returns the version of AK4384 driver in string format.

Implementation: Dynamic

File

drv_ak4384.h

C

```
int8_t* DRV_AK4384_VersionStrGet();
```

Returns

returns a string containing the version of AK4384 driver.

Description

The DRV_AK4384_VersionStrGet function returns a string in the format: ".[.][]" Where: is the AK4384 driver's version number. is the AK4384 driver's version number. is an optional "patch" or "dot" release number (which is not included in the string if it equals '00'). is an optional release

type ('a' for alpha, 'b' for beta not the entire word spelled out) that is not included if the release is a production version (i.e., not an alpha or beta). The String does not contain any spaces.

Remarks

None.

Preconditions

None.

Example 1

"0.03a" "1.00"

Example 2

```
int8_t *ak4384string;
ak4384string = DRV_AK4384_VersionStrGet();
```

Function

int8_t* DRV_AK4384_VersionStrGet(void)

DRV_AK4384_BufferQueueFlush Function

This function flushes off the buffers associated with the client object.

Implementation: Dynamic

File

drv_ak4384.h

C

```
void DRV_AK4384_BufferQueueFlush(const DRV_HANDLE handle);
```

Returns

None.

Description

This function flushes off the buffers associated with the client object and disables the DMA channel used for transmission.

Remarks

None.

Preconditions

The DRV_AK4384_Initialize routine must have been called for the specified AK4384 driver instance.

DRV_AK4384_Open must have been called to obtain a valid opened device handle.

One of DRV_AK4384_BufferAddRead/DRV_AK4384_BufferAddWrite function must have been called and buffers should have been queued for transmission.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
size_t bufferQueuedSize;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4384_BUFFER_HANDLE bufferHandle;

// myAK4384Handle is the handle returned
// by the DRV_AK4384_Open function.

// Client registers an event handler with driver. This is done once

DRV_AK4384_BufferEventHandlerSet(myAK4384Handle, APP_AK4384BufferEventHandle,
                                                    (uintptr_t)&myAppObj);

DRV_AK4384_BufferAddRead(myAK4384handle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4384_BUFFER_HANDLE_INVALID == bufferHandle)
```

```
{
    // Error handling here
}

// The data is being processed after adding the buffer to the queue.
// The user can stop the data processing and flushoff the data
// in the queue by calling DRV_AK4384_BufferQueueFlush
DRV_AK4384_BufferQueueFlush(myAK4384Handle);
```

Parameters

Parameters	Description
handle	Opened client handle associated with a driver object.

Function

void DRV_AK4384_BufferQueueFlush(DRV_HANDLE handle)

f) Data Types and Constants

DRV_AK4384_AUDIO_DATA_FORMAT Enumeration

Identifies the Serial Audio data interface format.

File

drv_ak4384.h

C

```
typedef enum {
  DRV_AK4384_AUDIO_DATA_FORMAT_16BIT_RIGHT_JUSTIFIED = 0,
  DRV_AK4384_AUDIO_DATA_FORMAT_20BIT_RIGHT_JUSTIFIED,
  DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_LEFT_JUSTIFIED,
  DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_I2S,
  DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_RIGHT_JUSTIFIED
} DRV_AK4384_AUDIO_DATA_FORMAT;
```

Members

Members	Description
DRV_AK4384_AUDIO_DATA_FORMAT_16BIT_RIGHT_JUSTIFIED = 0	16 bit Right Justified Audio data format
DRV_AK4384_AUDIO_DATA_FORMAT_20BIT_RIGHT_JUSTIFIED	20 bit Right Justified Audio data format
DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_LEFT_JUSTIFIED	24 bit Left Justified Audio data format
DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_I2S	24 bit I2S Audio data format
DRV_AK4384_AUDIO_DATA_FORMAT_24BIT_RIGHT_JUSTIFIED	24 bit Right Justified Audio data format

Description

AK4384 Audio data format

This enumeration identifies Serial Audio data interface format.

Remarks

None.

DRV_AK4384_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_ak4384.h

C

```
typedef enum {
  DRV_AK4384_BUFFER_EVENT_COMPLETE,
  DRV_AK4384_BUFFER_EVENT_ERROR,
  DRV_AK4384_BUFFER_EVENT_ABORT
```

```
} DRV_AK4384_BUFFER_EVENT;
```

Members

Members	Description
DRV_AK4384_BUFFER_EVENT_COMPLETE	Data was transferred successfully.
DRV_AK4384_BUFFER_EVENT_ERROR	Error while processing the request
DRV_AK4384_BUFFER_EVENT_ABORT	Data transfer aborted (Applicable in DMA mode)

Description

AK4384 Driver Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_AK4384_BufferAddWrite function.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_AK4384_BufferEventHandlerSet function when a buffer transfer request is completed.

DRV_AK4384_BUFFER_EVENT_HANDLER Type

Pointer to a AK4384 Driver Buffer Event handler function.

File

drv_ak4384.h

C

```
typedef void (* DRV_AK4384_BUFFER_EVENT_HANDLER)(DRV_AK4384_BUFFER_EVENT event, DRV_AK4384_BUFFER_HANDLE
bufferHandle, uintptr_t contextHandle);
```

Returns

None.

Description

AK4384 Driver Buffer Event Handler Function

This data type defines the required function signature for the AK4384 driver buffer event handling callback function. A client must register a pointer to a buffer event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_AK4384_BUFFER_EVENT_COMPLETE, this means that the data was transferred successfully.

If the event is DRV_AK4384_BUFFER_EVENT_ERROR, this means that the data was not transferred successfully. The bufferHandle parameter contains the buffer handle of the buffer that failed. The DRV_AK4384_BufferProcessedSizeGet function can be called to find out how many bytes were processed.

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4384_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The buffer handle in bufferHandle expires after this event handler exits. In that the buffer object that was allocated is deallocated by the driver after the event handler exits.

The event handler function executes in the data driver (I2S) peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

DRV_AK4384_BufferAddWrite function can be called in the event handler to add a buffer to the driver queue.

Example

```
void APP_MyBufferEventHandler( DRV_AK4384_BUFFER_EVENT event,
                               DRV_AK4384_BUFFER_HANDLE bufferHandle,
                               uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
```

```
    {
        case DRV_AK4384_BUFFER_EVENT_COMPLETE:
            // Handle the completed buffer.
        break;

        case DRV_AK4384_BUFFER_EVENT_ERROR:
        default:
            // Handle error.
        break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the event relates
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4384_BUFFER_HANDLE Type

Handle identifying a write buffer passed to the driver.

File

drv_ak4384.h

C

```
typedef uintptr_t DRV_AK4384_BUFFER_HANDLE;
```

Description

AK4384 Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_AK4384_BufferAddWrite function. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_AK4384_CHANNEL Enumeration

Identifies Left/Right Audio channel

File

drv_ak4384.h

C

```
typedef enum {
  DRV_AK4384_CHANNEL_LEFT,
  DRV_AK4384_CHANNEL_RIGHT,
  DRV_AK4384_CHANNEL_LEFT_RIGHT,
  DRV_AK4384_NUMBER_OF_CHANNELS
} DRV_AK4384_CHANNEL;
```

Description

AK4384 Audio Channel

This enumeration identifies Left/Right Audio channel

Remarks

None.

DRV_AK4384_COMMAND_EVENT_HANDLER Type

Pointer to a AK4384 Driver Command Event Handler Function

File

drv_ak4384.h

C

```
typedef void (* DRV_AK4384_COMMAND_EVENT_HANDLER)(uintptr_t contextHandle);
```

Returns

None.

Description

AK4384 Driver Command Event Handler Function

This data type defines the required function signature for the AK4384 driver command event handling callback function.

A command is a control instruction to the AK4384 Codec. For example, Mute ON/OFF, Zero Detect Enable/Disable, etc.

A client must register a pointer to a command event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive command related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

The occurrence of this call back means that the last control command was transferred successfully.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4384_CommandEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) of the client that made the buffer add request.

The event handler function executes in the control data driver interrupt context. It is recommended of the application to not perform process intensive or blocking operations with in this function.

Example

```
void APP_AK4384CommandEventHandler( uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    // Last Submitted command is completed.
    // Perform further processing here
}
```

Parameters

Parameters	Description
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4384_DEEMPHASIS_FILTER Enumeration

Identifies de-emphasis filter function.

File

drv_ak4384.h

C

```
typedef enum {
  DRV_AK4384_DEEMPHASIS_FILTER_44_1KHZ,
  DRV_AK4384_DEEMPHASIS_FILTER_OFF,
  DRV_AK4384_DEEMPHASIS_FILTER_48KHZ,
  DRV_AK4384_DEEMPHASIS_FILTER_32KHZ
} DRV_AK4384_DEEMPHASIS_FILTER;
```

Members

Members	Description
DRV_AK4384_DEEMPHASIS_FILTER_44_1KHZ	De-Emphasis filter for 44.1kHz.
DRV_AK4384_DEEMPHASIS_FILTER_OFF	De-Emphasis filter Off This is the default setting.

DRV_AK4384_DEEMPHASIS_FILTER_48KHZ	De-Emphasis filter for 48kHz.
DRV_AK4384_DEEMPHASIS_FILTER_32KHZ	De-Emphasis filter for 32kHz.

Description

AK4384 De-Emphasis Filter

This enumeration identifies the settings for de-emphasis filter function.

Remarks

None.

DRV_AK4384_INIT Structure

Defines the data required to initialize or reinitialize the AK4384 driver.

File

drv_ak4384.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX spiDriverModuleIndex;
  SYS_MODULE_INDEX i2sDriverModuleIndex;
  uint8_t volume;
  DRV_AK4384_MCLK_MODE mclkMode;
} DRV_AK4384_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX spiDriverModuleIndex;	Identifies control module(SPI) driver ID for control interface of Codec
SYS_MODULE_INDEX i2sDriverModuleIndex;	Identifies data module(I2S) driver ID for data interface of Codec
uint8_t volume;	Volume
DRV_AK4384_MCLK_MODE mclkMode;	Set MCLK mode.

Description

AK4384 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the AK4384 Codec driver.

Remarks

None.

DRV_AK4384_MCLK_MODE Enumeration

Identifies the mode of master clock to AK4384 DAC.

File

drv_ak4384.h

C

```
typedef enum {
  DRV_AK4384_MCLK_MODE_MANUAL,
  DRV_AK4384_MCLK_MODE_AUTO
} DRV_AK4384_MCLK_MODE;
```

Members

Members	Description
DRV_AK4384_MCLK_MODE_MANUAL	Master clock frequency mode Manual
DRV_AK4384_MCLK_MODE_AUTO	Master clock frequency mode Auto This is the default mode.

Description

AK4384 Master clock frequency mode

This enumeration identifies mode of master clock to AK4384 DAC. In Manual Setting Mode, the sampling speed is set by setting DFS0/1 bits in Control Register 2. The frequency of MCLK at each sampling speed is set automatically. In Auto Setting Mode, the MCLK frequency is detected automatically

Remarks

None.

DRV_AK4384_ZERO_DETECT_MODE Enumeration

Identifies Zero Detect Function mode

File

drv_ak4384.h

C

```
typedef enum {
  DRV_AK4384_ZERO_DETECT_MODE_CHANNEL_SEPARATED,
  DRV_AK4384_ZERO_DETECT_MODE_ANDED
} DRV_AK4384_ZERO_DETECT_MODE;
```

Members

Members	Description
DRV_AK4384_ZERO_DETECT_MODE_CHANNEL_SEPARATED	Zero Detect channel separated. When the input data at each channel is continuously zeros for 8192 LRCK cycles, DZF pin of each channel goes to “H” This is the default mode.
DRV_AK4384_ZERO_DETECT_MODE_ANDED	Zero Detect Anded DZF pins of both channels go to “H” only when the input data at both channels are continuously zeros for 8192 LRCK cycles

Description

AK4384 Zero Detect mode

This enumeration identifies the mode of zero detect function

Remarks

None.

DRV_AK4384_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_ak4384.h

C

```
#define DRV_AK4384_BUFFER_HANDLE_INVALID ((DRV_AK4384_BUFFER_HANDLE)(-1))
```

Description

AK4384 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_AK4384_BufferAddWrite function if the buffer add request was not successful.

Remarks

None.

DRV_AK4384_COUNT Macro

Number of valid AK4384 driver indices.

File

drv_ak4384.h

C

```
#define DRV_AK4384_COUNT
```

Description

AK4384 Driver Module Count

This constant identifies the maximum number of AK4384 Driver instances that should be defined by the application. Defining more instances than this constant will waste RAM memory space.

This constant can also be used by the application to identify the number of AK4384 instances on this microcontroller.

Remarks

This value is device-specific.

DRV_AK4384_INDEX_0 Macro

AK4384 driver index definitions.

File

drv_ak4384.h

C

```
#define DRV_AK4384_INDEX_0 0
```

Description

Driver AK4384 Module Index

These constants provide AK4384 driver index definition.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_AK4384_Initialize and DRV_AK4384_Open routines to identify the driver instance in use.

DRV_AK4384_INDEX_1 Macro

File

drv_ak4384.h

C

```
#define DRV_AK4384_INDEX_1 1
```

Description

This is macro DRV_AK4384_INDEX_1.

DRV_AK4384_INDEX_2 Macro

File

drv_ak4384.h

C

```
#define DRV_AK4384_INDEX_2 2
```

Description

This is macro DRV_AK4384_INDEX_2.

DRV_AK4384_INDEX_3 Macro

File

drv_ak4384.h

C

```
#define DRV_AK4384_INDEX_3 3
```

Description

This is macro DRV_AK4384_INDEX_3.

DRV_AK4384_INDEX_4 Macro

File

drv_ak4384.h

C

```
#define DRV_AK4384_INDEX_4 4
```

Description

This is macro DRV_AK4384_INDEX_4.

DRV_AK4384_INDEX_5 Macro

File

drv_ak4384.h

C

```
#define DRV_AK4384_INDEX_5 5
```

Description

This is macro DRV_AK4384_INDEX_5.

Files

Files

Name	Description
drv_ak4384.h	AK4384 Codec Driver Interface header file
drv_ak4384_config_template.h	AK4384 Codec Driver Configuration Template.

Description

This section lists the source and header files used by the AK4384Codec Driver Library.

drv_ak4384.h

AK4384 Codec Driver Interface header file

Enumerations

	Name	Description
	DRV_AK4384_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4384_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4384_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4384_DEEMPHASIS_FILTER	Identifies de-emphasis filter function.
	DRV_AK4384_MCLK_MODE	Identifies the mode of master clock to AK4384 DAC.
	DRV_AK4384_ZERO_DETECT_MODE	Identifies Zero Detect Function mode

Functions

	Name	Description
	DRV_AK4384_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic
	DRV_AK4384_BufferCombinedQueueSizeGet	This function returns the number of bytes queued (to be processed) in the buffer queue. **Implementation:** Dynamic
	DRV_AK4384_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Dynamic

	DRV_AK4384_BufferProcessedSizeGet	This function returns number of bytes that have been processed for the specified buffer. **Implementation:** Dynamic
	DRV_AK4384_BufferQueueFlush	This function flushes off the buffers associated with the client object. **Implementation:** Dynamic
	DRV_AK4384_ChannelOutputInvertDisable	Disables output polarity of the selected Channel. **Implementation:** Dynamic
	DRV_AK4384_ChannelOutputInvertEnable	Enables output polarity of the selected channel. **Implementation:** Dynamic
	DRV_AK4384_Close	Closes an opened-instance of the AK4384 driver. **Implementation:** Dynamic
	DRV_AK4384_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished. **Implementation:** Dynamic
	DRV_AK4384_DeEmphasisFilterSet	Allows specifies enabling of digital de-emphasis filter. **Implementation:** Dynamic
	DRV_AK4384_Deinitialize	Deinitializes the specified instance of the AK4384 driver module. **Implementation:** Dynamic
	DRV_AK4384_Initialize	Initializes hardware and data for the instance of the AK4384 DAC module. **Implementation:** Dynamic
	DRV_AK4384_MuteOff	Disables AK4384 output for soft mute. **Implementation:** Dynamic
	DRV_AK4384_MuteOn	Allows AK4384 output for soft mute on. **Implementation:** Dynamic
	DRV_AK4384_Open	Opens the specified AK4384 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_AK4384_SamplingRateGet	This function gets the sampling rate set on the DAC AK4384. **Implementation:** Dynamic
	DRV_AK4384_SamplingRateSet	This function sets the sampling rate of the media stream. **Implementation:** Dynamic
	DRV_AK4384_SlowRollOffFilterDisable	Disables Slow Roll-off filter function. **Implementation:** Dynamic
	DRV_AK4384_SlowRollOffFilterEnable	Enables Slow Roll-off filter function. **Implementation:** Dynamic
	DRV_AK4384_Status	Gets the current status of the AK4384 driver module. **Implementation:** Dynamic
	DRV_AK4384_Tasks	Maintains the driver's control and data interface state machine. **Implementation:** Dynamic
	DRV_AK4384_VersionGet	Returns the version of the AK4384 driver. **Implementation:** Dynamic
	DRV_AK4384_VersionStrGet	Returns the version of AK4384 driver in string format. **Implementation:** Dynamic
	DRV_AK4384_VolumeGet	This function gets the volume for AK4384 Codec. **Implementation:** Dynamic
	DRV_AK4384_VolumeSet	This function sets the volume for AK4384 Codec. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectDisable	Disables AK4384 channel-independent zeros detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectEnable	Enables AK4384 channel-independent zeros detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectInvertDisable	Disables inversion of polarity for zero detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectInvertEnable	Enables inversion of polarity for zero detect function. **Implementation:** Dynamic
	DRV_AK4384_ZeroDetectModeSet	Sets mode of AK4384 channel-independent zeros detect function. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_AK4384_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4384_COUNT	Number of valid AK4384 driver indices.
	DRV_AK4384_INDEX_0	AK4384 driver index definitions.
	DRV_AK4384_INDEX_1	This is macro DRV_AK4384_INDEX_1.
	DRV_AK4384_INDEX_2	This is macro DRV_AK4384_INDEX_2.
	DRV_AK4384_INDEX_3	This is macro DRV_AK4384_INDEX_3.
	DRV_AK4384_INDEX_4	This is macro DRV_AK4384_INDEX_4.
	DRV_AK4384_INDEX_5	This is macro DRV_AK4384_INDEX_5.

Structures

	Name	Description
	DRV_AK4384_INIT	Defines the data required to initialize or reinitialize the AK4384 driver.

Types

	Name	Description
	DRV_AK4384_BUFFER_EVENT_HANDLER	Pointer to a AK4384 Driver Buffer Event handler function.
	DRV_AK4384_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4384_COMMAND_EVENT_HANDLER	Pointer to a AK4384 Driver Command Event Handler Function

Description

AK4384 Codec Driver Interface

The AK4384 Codec device driver interface provides a simple interface to manage the AK4384 106 dB 192 kHz 24-Bit DAC that can be interfaced Microchip Microcontroller. This file provides the interface definition for the AK4384 Codec device driver.

File Name

drv_ak4384.h

Company

Microchip Technology Inc.

drv_ak4384_config_template.h

AK4384 Codec Driver Configuration Template.

Macros

	Name	Description
	DRV_AK4384_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1, and 48K sampling frequency
	DRV_AK4384_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4384_CONTROL_CLOCK	Sets up clock frequency for the control interface (SPI)
	DRV_AK4384_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4384_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4384_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for 32, 44.1 and 48K sampling frequency
	DRV_AK4384_TIMER_DRIVER_MODULE_INDEX	Identifies the Timer Module Index for custom virtual SPI driver implementation.
	DRV_AK4384_TIMER_PERIOD	Identifies the period for the bit bang timer.

Description

AK4384 Codec Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_ak4384_config_template.h

Company

Microchip Technology Inc.

AK4642 Codec Driver Library

This topic describes the AK4642 Codec Driver Library.

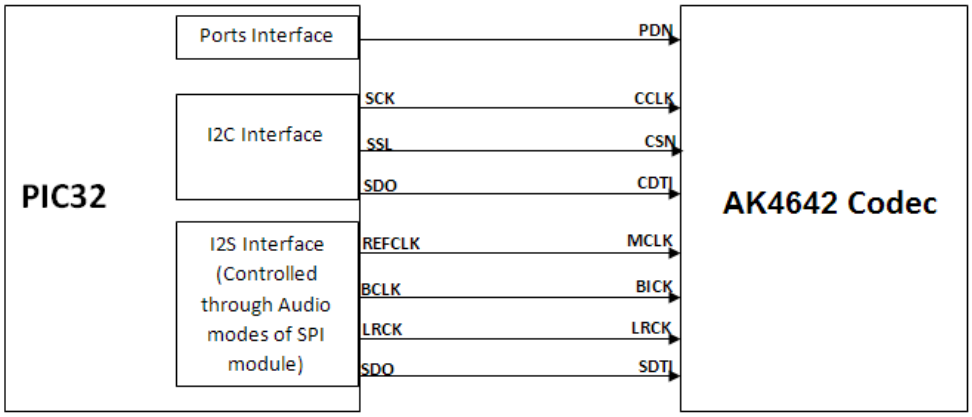
Introduction

This library provides an interface to manage the AK4642 Codec that is serially interfaced to a Microchip microcontroller for providing Audio Solutions.

Description

The AK4642 module is 16/24-bit Audio Codec from Asahi Kasei Microdevices Corporation. The AK4642 can be interfaced to Microchip microcontrollers through I2C and I2S serial interfaces. The I2C interface is used for control command transfer. The I2S interface is used for Audio data output.

A typical interface of AK4642 to a Microchip PIC32 device is provided in the following diagram:



Features

The AK4642 Codec Driver supports the following features:

- Audio Interface Format: MSB first
- ADC: 16-bit MSB justified, 16/24-bit I2S
- DAC: 16-bit MSB justified, 16bit LSB justified, 16/24-bit I2S
- Sampling Frequency Range: 8 kHz to 48 kHz
- Digital Volume Control: +12dB ~ .115dB, 0.5dB Step
- SoftMute: On and Off
- Master Clock Frequencies: 32 fs/64 fs/128fs/256fs

Using the Library

This topic describes the basic architecture of the AK4642 Codec Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_ak4642.h`

The interface to the AK4642 Codec Driver library is defined in the `drv_ak4642.h` header file. Any C language source (`.c`) file that uses the AK4642 Codec Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

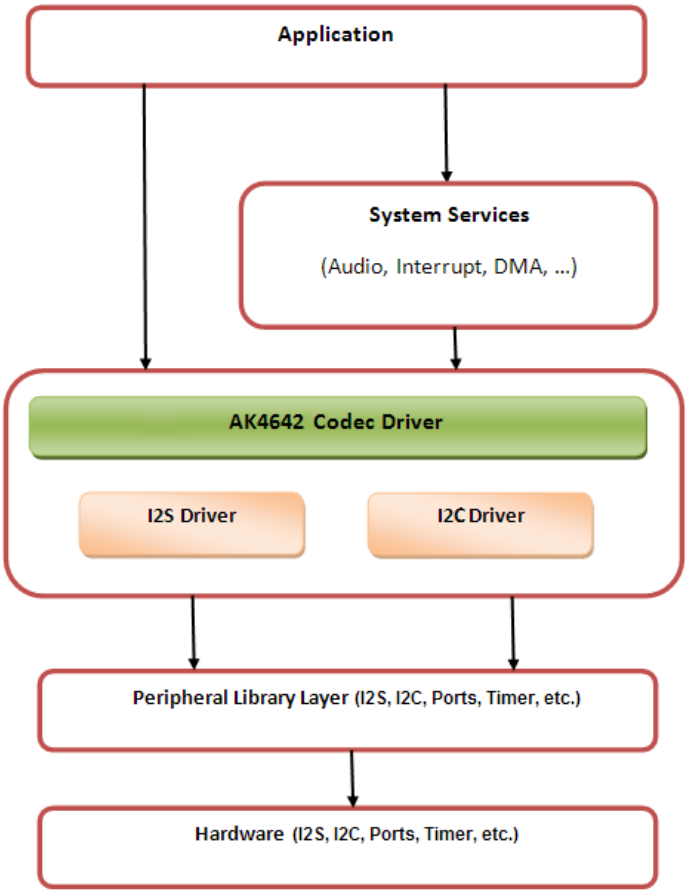
Abstraction Model

This library provides a low-level abstraction of the AK4642 Codec Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The abstraction model shown in the following diagram depicts how the AK4642 Codec Driver is positioned in the MPLAB Harmony framework. The AK4642 Codec Driver uses the SPI and I2S drivers for control and audio data transfers to the AK4642 module.

AK4642 Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The AK4642 Codec Driver Library provides an API interface to transfer control commands and digital audio data to the serially interfaced AK4642 DAC module. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the AK4642 Codec Driver Library.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Setup Functions	Provides open and close functions.
Codec Specific Functions	Provides functions that are codec specific.
Data Transfer Functions	Provides data transfer functions.
Other Functions	Provides driver specific miscellaneous functions such as sampling rate setting, control command functions, etc.
Data Types and Constants	These data types and constants are required while interacting and setting up the AK4642 Codec Driver Library.

How the Library Works

The library provides interfaces to support:

- System Functionality

- Client Functionality

System Access

This topic provides information on system initialization, implementations, and provides a system access code example.

Description

System Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the AK4642 module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_AK4642_INIT or by using Initialization Overrides) that are supported by the specific AK4642 device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- I2C driver module index. The module index should be same as the one used in initializing the I2C Driver.
- I2S driver module index. The module index should be same as the one used in initializing the I2S Driver.
- Sampling rate
- Master clock detection mode
- Power down pin port initialization

The DRV_AK4642_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_ AK4642_Deinitialize, DRV_ AK4642_Status and DRV_I2S_Tasks.

Implementations

The AK4642 Codec Driver can have the following implementations:

Implementation	Description	MPLAB Harmony Components
Implementation 1	Dedicated hardware for control (I2C) and data (I2S) interface.	Standard MPLAB Harmony drivers for I2C and I2S interfaces.
Implementation 2	Dedicated hardware for data (I2S) interface. Ports pins for control interface.	Standard MPLAB Harmony drivers for I2S interface. Virtual MPLAB Harmony drivers for I2C interface.

Example:

```
DRV_AK4642_INIT drvak4642Init =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .i2sDriverModuleIndex = DRV_AK4642_I2S_DRIVER_MODULE_INDEX_IDX0,
    .i2cDriverModuleIndex = DRV_AK4642_I2C_DRIVER_MODULE_INDEX_IDX0,
    .volume = DRV_AK4642_VOLUME,
};

/*
  The I2C and I2S module index should be same as the one used in
  initializing the I2C and I2S drivers.
 */

ak4642DevObject = DRV_AK4642_Initialize(DRV_AK4642_INDEX_0, (SYS_MODULE_INIT *) &drvak4642Init);
if (SYS_MODULE_OBJ_INVALID == ak4642DevObject)
{
    // Handle error
}
```

Task Routine

The DRV_AK4642_Tasks will be called from the System Task Service.

Client Access

This topic describes client access and includes a code example.

Description

For the application to start using an instance of the module, it must call the DRV_AK4642_Open function. The DRV_AK4642_Open provides a driver handle to the AK4642 Codec Driver instance for operations. If the driver is deinitialized using the function DRV_AK4642_Deinitialize, the application must call the DRV_AK4642_Open function again to set up the instance of the driver.

For the various options available for IO_INTENT, please refer to

Data Types and Constants in the Library Interface section.

Note: It is necessary to check the status of driver initialization before opening a driver instance. The status of the AK4642 Codec Driver can be known by calling DRV_AK4642_Status.

Example:

```
DRV_HANDLE handle;
SYS_STATUS ak4642Status;
    ak4642Status = DRV_AK4642_Status(sysObjects.ak4642DevObject);
        if (SYS_STATUS_READY == ak4642Status)
        {
               // The driver can now be opened.
               appData.ak4642Client.handle = DRV_AK4642_Open
                                            (DRV_AK4642_INDEX_0,
                                             DRV_IO_INTENT_WRITE |
                                             DRV_IO_INTENT_EXCLUSIVE );
               if(appData.ak4642Client.handle != DRV_HANDLE_INVALID)
               {
                   appData.state = APP_STATE_AK4642_SET_BUFFER_HANDLER;
               }
               else
               {
                   SYS_DEBUG(0, "Find out what's wrong \r\n");
               }
    }
    else
    {
        /* AK4642 Driver Is not ready */
                ;
    }
```

Client Operations

This topic describes client operations and provides a code example.

Description

Client operations provide the API interface for control command and audio data transfer to the AK4642 Codec.

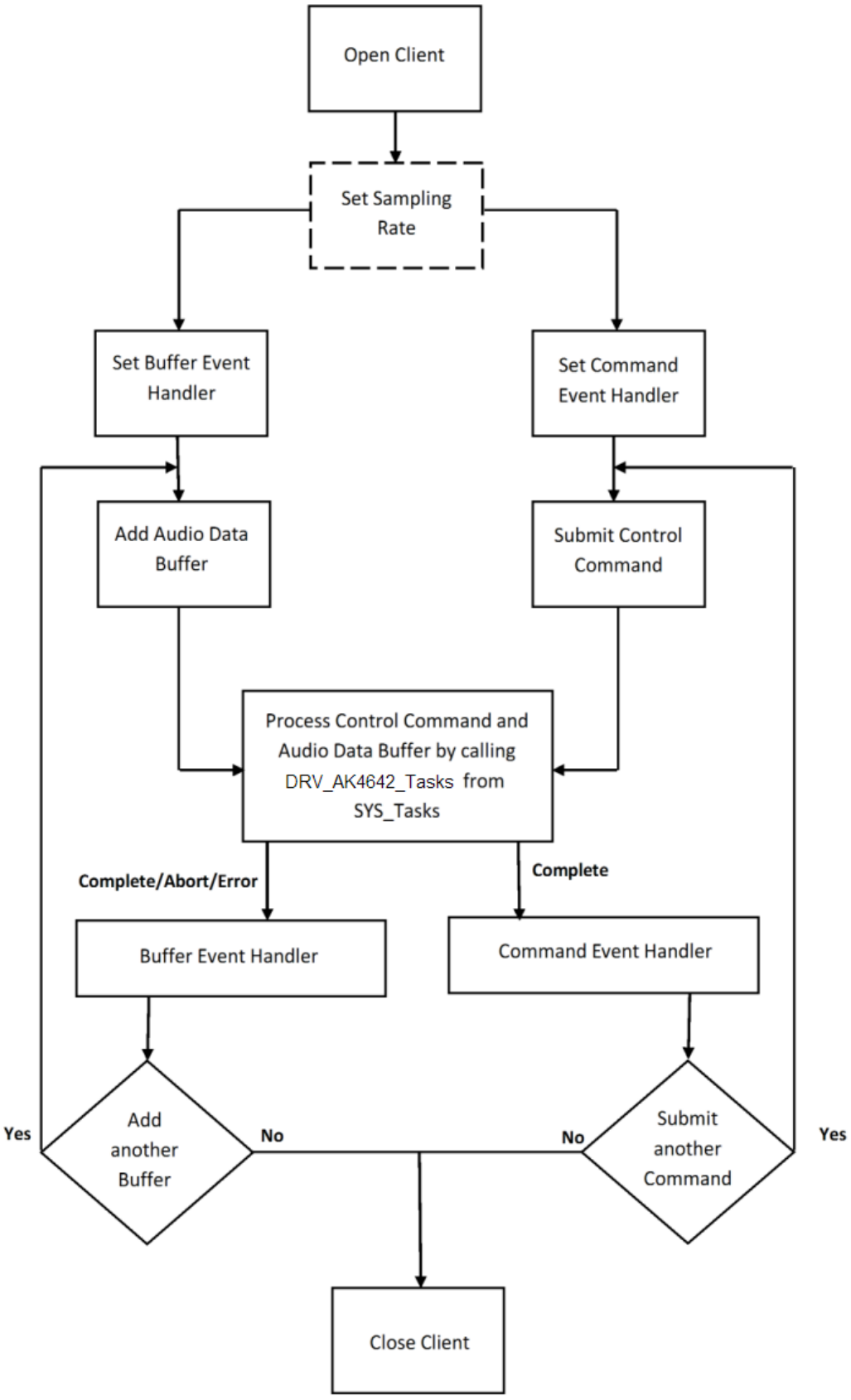
The following AK4642 Codec specific control command functions are provided:

- DRV_AK4642_SamplingRateSet
- DRV_AK4642_SamplingRateGet
- DRV_AK4642_VolumeSet
- DRV_AK4642_VolumeGet
- DRV_AK4642_MuteOn
- DRV_AK4642_MuteOff
- DRV_AK4642_IntExtMicSet
- DRV_AK4642_MonoStereoMicSet

These functions schedule a non-blocking control command transfer operation. These functions submit the control command request to the I2C Driver transmit queue, where the request is processed immediately if it is the first request, or it is processed when the previous request is complete.

DRV_AK4642_BufferAddWrite, DRV_AK4642_BufferAddRead, and DRV_AK4642_BufferAddWriteRead are buffered data operation functions. These functions schedule non-blocking audio data transfer operations. These functions add the request to the I2S Driver transmit or receive buffer queue depending on the request type, and are executed immediately if it is the first buffer, or executed later when the previous buffer is complete. The driver notifies the client with DRV_AK4642_BUFFER_EVENT_COMPLETE, DRV_AK4642_BUFFER_EVENT_ERROR, or DRV_AK4642_BUFFER_EVENT_ABORT events.

The following diagram illustrates the control commands and audio buffered data operations.



Note: It is not necessary to close and reopen the client between multiple transfers.

An application using the buffered functionality needs to perform the following steps:

1. The system should have completed necessary setup and initializations.
2. The I2S driver object should have been initialized by calling DRV_I2S_Initialize.

3. The I2C driver object should have been initialized by calling DRV_I2C_Initialize.
4. The AK4642 driver object should be initialized by calling DRV_AK4642_Initialize.
5. The necessary sampling rate value should be set up by calling DRV_AK4642_ SamplingRateSet.
6. Register buffer event handler for the client handle by calling DRV_AK4642_BufferEventHandlerSet.
7. Submit a command by calling specific command API.
8. Add a buffer to initiate the data transfer by calling DRV_AK4642_BufferAddWrite, DRV_AK4642_BufferAddRead, and DRV_AK4642_BufferAddWriteRead.
9. Call the DRV_AK4642_BufferAddWrite, DRV_AK4642_BufferAddRead, or DRV_AK4642_BufferAddWriteRead function for handling multiple buffer transmissions or receptions.
10. When the client is done, it can use DRV_AK4642_Close to close the client handle.

Example:

```
typedef enum
{
    APP_STATE_AK4642_OPEN,
    APP_STATE_AK4642_SET_BUFFER_HANDLER,
    APP_STATE_AK4642_ADD_FIRST_BUFFER_READ,
    APP_STATE_AK4642_ADD_BUFFER_OUT,
    APP_STATE_AK4642_ADD_BUFFER_IN,
    APP_STATE_AK4642_WAIT_FOR_BUFFER_COMPLETE,
} APP_STATES;

typedef struct
{
    DRV_HANDLE handle;
    DRV_AK4642_BUFFER_HANDLE writereadBufHandle;
    DRV_AK4642_BUFFER_EVENT_HANDLER bufferEventHandler;
    uintptr_t context;
    uint8_t *txbufferObject;
    uint8_t *rxbufferObject;
    size_t bufferSize;

} APP_AK4642_CLIENT;

typedef struct
{
    /* Application's current state*/
    APP_STATES state;
    /* USART client handle */
    APP_AK4642_CLIENT ak4642Client;
} APP_DATA;
APP_DATA appData;
SYS_MODULE_OBJ ak4642DevObject;
DRV_AK4642_INIT drvak4642Init =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .i2sDriverModuleIndex = DRV_AK4642_I2S_DRIVER_MODULE_INDEX_IDX0,
    .i2cDriverModuleIndex = DRV_AK4642_I2C_DRIVER_MODULE_INDEX_IDX0,
    .volume = DRV_AK4642_VOLUME,
};

void SYS_Initialize(void * data)
{
    /* Initialize Drivers */
    DRV_I2C0_Initialize();
    sysObj.drvI2S0 = DRV_I2S_Initialize(DRV_I2S_INDEX_0, (SYS_MODULE_INIT *)
                                        &drvI2S0InitData);

    sysObj.drvak4642Codec0 = DRV_AK4642_Initialize(DRV_AK4642_INDEX_0,
                             (SYS_MODULE_INIT *)&drvak4642Codec0InitData);

    /* Initialize System Services */
    SYS_INT_Initialize();
}

void APP_Tasks (void )
{
    switch(appData.state)
```

```
{
    case APP_STATE_AK4642_OPEN:
    {
        SYS_STATUS status;
        status = DRV_CODEC_Status(sysObjdrvCodec0);
        if (SYS_STATUS_READY == status)
        {
        /* A client opens the driver object to get an Handle */
        appData.ak4642Client.handle = DRV_AK4642_Open(DRV_AK4642_INDEX_0,
                           DRV_IO_INTENT_WRITE|DRV_IO_INTENT_EXCLUSIVE);
        if(appData.ak4642Client.handle != DRV_HANDLE_INVALID)
        {
            appData.state = APP_STATE_AK4642_SET_BUFFER_HANDLER;
        }
        else
        {
           /* Got an Invalid Handle. Wait for AK4642 to Initialize */
        }
      }
    }
    break;

    /* Set a handler for the audio buffer completion event */
    case APP_STATE_AK4642_SET_BUFFER_HANDLER:
    {

       DRV_AK4642_BufferEventHandlerSet(appData.ak4642Client.handle,
                appData.ak4642Client.bufferEventHandler,
                appData.ak4642Client.context);

        appData.state = APP_STATE_AK4642_ADD_FIRST_BUFFER_READ;
    }
    break;

    case APP_STATE_AK4642_ADD_FIRST_BUFFER_READ:
    {
        DRV_AK4642_BufferAddWriteRead(appData.ak4642Client.handle,
                &appData.ak4642Client.writeReadBufHandle,
                appData.ak4642Client.txbufferObject,
                appData.ak4642Client.rxbufferObject,
                appData.ak4642Client.bufferSize);
        if(appData.ak4642Client.writeReadBufHandle != DRV_AK4642_BUFFER_HANDLE_INVALID)
        {
            appData.state = APP_STATE_AK4642_WAIT_FOR_BUFFER_COMPLETE;
        }
        else
        {
            SYS_DEBUG(0, "Find out what is wrong \r\n");
        }

    }
    break;
    /* Add an audio buffer to the ak4642 driver to be transmitted to
     * AK4642 CODEC */
    case APP_STATE_AK4642_ADD_BUFFER_OUT:
    {

        DRV_AK4642_BufferAddWrite(appData.ak4642Client.handle, &appData.ak4642Client.writeBufHandle,
        appData.ak4642Client.txbufferObject, appData.ak4642Client.bufferSize);
        if(appData.ak4642Client.writeBufHandle != DRV_AK4642_BUFFER_HANDLE_INVALID)
        {

            appData.state = APP_STATE_AK4642_WAIT_FOR_BUFFER_COMPLETE;
        }
        else
        {
            SYS_DEBUG(0, "Find out what is wrong \r\n");
        }
    }
```

```
            break;
            /* Add an audio buffer to the ak4642 driver to be received
             * AK4642 CODEC */
            case APP_STATE_AK4642_ADD_BUFFER_IN:
            {

                DRV_AK4642_BufferAddRead(appData.ak4642Client.handle, &appData.ak4642Client.readBufHandle,
                appData.ak4642Client.rxbufferObject, appData.ak4642Client.bufferSize);


                if(appData.ak4642Client.readBufHandle != DRV_AK4642_BUFFER_HANDLE_INVALID)
                {
                    appData.state = APP_STATE_AK4642_ADD_BUFFER_OUT;
                }
                else
                {
                    SYS_DEBUG(0, "Find out what is wrong \r\n");
                }
            }
            break;
            /* Audio data Transmission under process */
            case APP_STATE_AK4642_WAIT_FOR_BUFFER_COMPLETE:
            {
                /*Do nothing*/
            }
            break;

            default:
            {
            }
            break;
        }

    }

    /*******************************************************
     * Application AK4642 buffer Event handler.
     * This function is called back by the AK4642 driver when
     * a AK4642 data buffer RX completes.
     *******************************************************/
    void APP_AK4642MicBufferEventHandler(DRV_AK4642_BUFFER_EVENT event,
            DRV_AK4642_BUFFER_HANDLE handle, uintptr_t context )
    {
        static uint8_t cnt = 0;

        switch(event)
        {
            case DRV_AK4642_BUFFER_EVENT_COMPLETE:
            {

                bufnum ^= 1;

                if(bufnum ==0)
                {
                    appData.ak4642Client.rxbufferObject = (uint8_t *) micbuf1;
                    appData.ak4642Client.txbufferObject = (uint8_t *) micbuf2;
                }
                else if(bufnum ==1)
                {
                    appData.ak4642Client.rxbufferObject = (uint8_t *) micbuf2;
                    appData.ak4642Client.txbufferObject = (uint8_t *) micbuf1;
                }

                    DRV_AK4642_BufferAddWriteRead(appData.ak4642Client.handle,
                    &appData.ak4642Client.writeReadBufHandle,
                    appData.ak4642Client.txbufferObject,
                    appData.ak4642Client.rxbufferObject,
                    appData.ak4642Client.bufferSize);
                    appData.state = APP_STATE_AK4642_WAIT_FOR_BUFFER_COMPLETE;
```

```
            }
            break;
            case DRV_AK4642_BUFFER_EVENT_ERROR:
            {
            } break;

            case DRV_AK4642_BUFFER_EVENT_ABORT:
            {
            } break;

        }
    }

    void SYS_Tasks(void)
    {
        DRV_AK4642_Tasks(ak4642DevObject);
        APP_Tasks();
    }
```

Configuring the Library

Macros

	Name	Description
	DRV_AK4642_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4642_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4642_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4642_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4642_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4642_MCLK_SOURCE	Indicate the input clock frequency to generate the MCLK to codec.

Description

The configuration of the AK4642 Codec Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the AK4642 Codec Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the AK4642 Codec Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_AK4642_BCLK_BIT_CLK_DIVISOR Macro

Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

File

drv_ak4642_config_template.h

C

```
#define DRV_AK4642_BCLK_BIT_CLK_DIVISOR
```

Description

AK4642 BCLK to LRCK Ratio to Generate Audio Stream

Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

Following BCLK to LRCK ratios are supported 16bit data 16 bit channel :- 32fs, hence divisor would be 8 16bit data 32 bit channel :- 64fs, hence divisor would be 4

Remarks

None.

DRV_AK4642_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_ak4642_config_template.h

C

```
#define DRV_AK4642_CLIENTS_NUMBER DRV_AK4642_INSTANCES_NUMBER
```

Description

AK4642 Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. Typically only one client could be connected to one hardware instance. This value represents the total number of clients to be supported across all hardware instances. Therefore, if there are five AK4642 hardware interfaces, this number will be 5.

Remarks

None.

DRV_AK4642_INPUT_REFCLOCK Macro

Identifies the input REFCLOCK source to generate the MCLK to codec.

File

drv_ak4642_config_template.h

C

```
#define DRV_AK4642_INPUT_REFCLOCK
```

Description

AK4642 Input reference clock

Identifies the input REFCLOCK source to generate the MCLK to codec.

Remarks

None.

DRV_AK4642_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_ak4642_config_template.h

C

```
#define DRV_AK4642_INSTANCES_NUMBER
```

Description

AK4642 driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of AK4642 CODEC modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None.

DRV_AK4642_MCLK_SAMPLE_FREQ_MULTPLIER Macro

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

File

drv_ak4642_config_template.h

C

```
#define DRV_AK4642_MCLK_SAMPLE_FREQ_MULTPLIER
```

Description

AK4642 MCLK to LRCK Ratio to Generate Audio Stream

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency I2S sampling frequency

Supported MCLK to Sampling frequency Ratios are as below 256fs, 384fs, 512fs, 768fs or 1152fs

Remarks

None

DRV_AK4642_MCLK_SOURCE Macro

Indicate the input clock frequency to generate the MCLK to codec.

File

drv_ak4642_config_template.h

C

```
#define DRV_AK4642_MCLK_SOURCE
```

Description

AK4642 Data Interface Master Clock Speed configuration

Indicate the input clock frequency to generate the MCLK to codec.

Remarks

None.

Configuring the MHC

Provides examples on how to configure the MPLAB Harmony Configurator (MHC) for a specific driver.

Description

The following three figures show examples of MHC configurations for the AK4642 Codec Driver, I2S Driver, and the I2C Driver.

Figure 1: AK4642 Codec Driver MHC Configuration

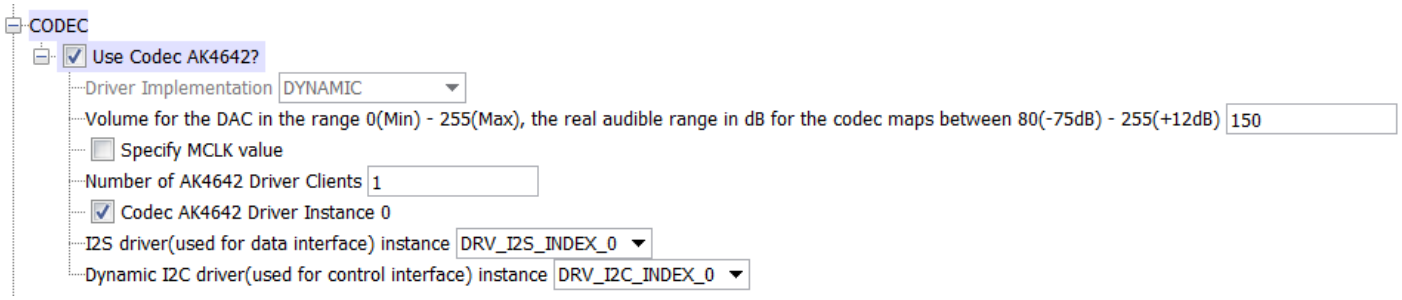


Figure 2: I2S Driver MHC Configuration

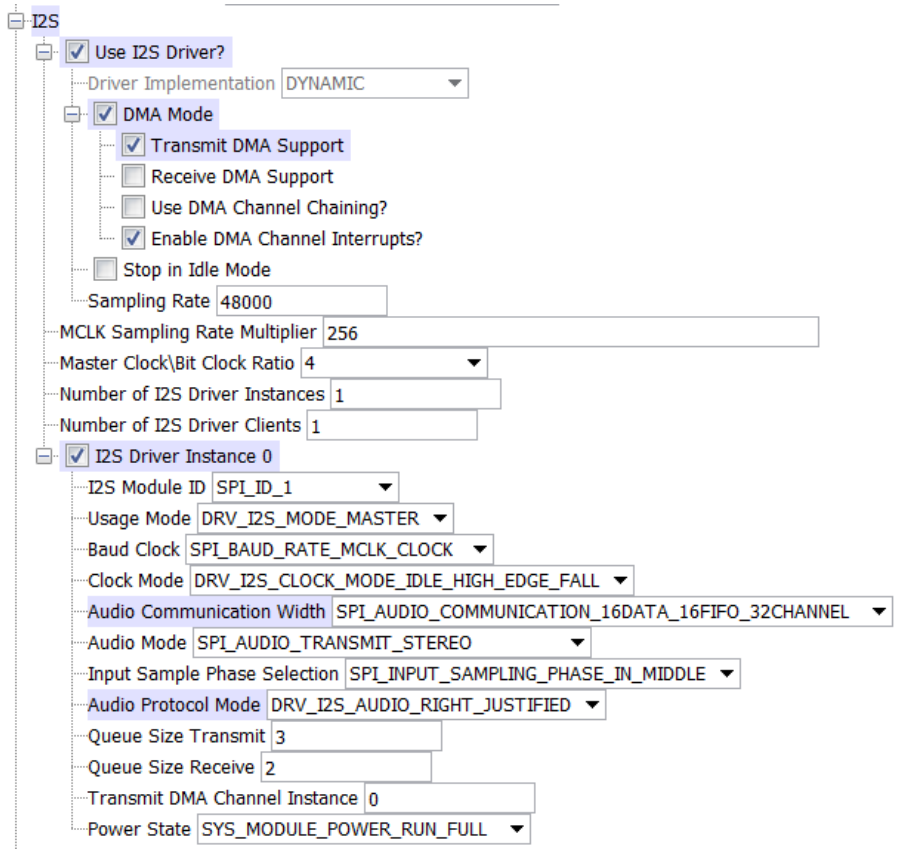
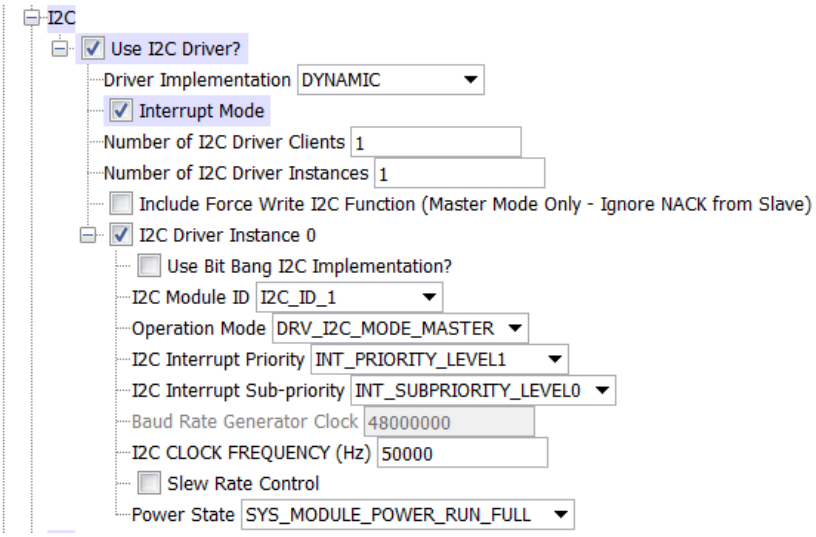


Figure 3: I2C Driver MHC Configuration



Migrating the AK4642 Driver From Earlier Versions of Microchip Harmony

Prior to version 1.08 of MPLAB Harmony, the AK4642 Codec Driver Library used the static I2C driver implementation. Beginning with v1.08 of MPLAB Harmony, applications must use the Dynamic Driver implementation with the MHC configured as shown in Figure 3. In addition, PIC32MZ configurations require the "Include Force Write I2C Function (Master Mode Only - Ignore NACK from Slave)" option to be selected.

Building the Library

This section lists the files that are available in the AK4642 Codec Driver Library.

Description

This section list the files that are available in the `/src` folder of the AK4642 Codec Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/codec/ak4642`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ak4642.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_ak4642.c`	This file contains implementation of the AK4642 Codec Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
There are no optional files for this driver.	N/A

Module Dependencies

The AK4642 Driver Library depends on the following modules:

- I2S Driver Library
- I2C Driver Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_AK4642_Initialize	Initializes hardware and data for the instance of the AK4642 DAC module
	DRV_AK4642_Deinitialize	Deinitializes the specified instance of the AK4642 driver module
	DRV_AK4642_Status	Gets the current status of the AK4642 driver module.
	DRV_AK4642_Tasks	Maintains the driver's control and data interface state machine.

b) Client Setup Functions

	Name	Description
	DRV_AK4642_Open	Opens the specified AK4642 driver instance and returns a handle to it
	DRV_AK4642_Close	Closes an opened-instance of the AK4642 driver

c) Codec Specific Functions

	Name	Description
	DRV_AK4642_MuteOff	This function disables AK4642 output for soft mute.
	DRV_AK4642_MuteOn	This function allows AK4642 output for soft mute on.
	DRV_AK4642_SamplingRateGet	This function gets the sampling rate set on the AK4642. **Implementation:** Dynamic
	DRV_AK4642_SamplingRateSet	This function sets the sampling rate of the media stream.
	DRV_AK4642_VolumeGet	This function gets the volume for AK4642 CODEC.
	DRV_AK4642_VolumeSet	This function sets the volume for AK4642 CODEC.
	DRV_AK4642_IntExtMicSet	This function sets up the codec for the internal or the external microphone use.
	DRV_AK4642_MonoStereoMicSet	This function sets up the codec for the Mono or Stereo microphone mode.

d) Data Transfer Functions

	Name	Description
	DRV_AK4642_BufferAddWrite	Schedule a non-blocking driver write operation.
	DRV_AK4642_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_AK4642_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_AK4642_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

e) Other Functions

	Name	Description
	DRV_AK4642_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.
	DRV_AK4642_VersionGet	This function returns the version of AK4642 driver
	DRV_AK4642_VersionStrGet	This function returns the version of AK4642 driver in string format.

f) Data Types and Constants

	Name	Description
	_DRV_AK4642_H	Include files.
	DRV_AK4642_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4642_COUNT	Number of valid AK4642 driver indices
	DRV_AK4642_INDEX_0	AK4642 driver index definitions
	DRV_AK4642_INDEX_1	This is macro DRV_AK4642_INDEX_1.
	DRV_AK4642_INDEX_2	This is macro DRV_AK4642_INDEX_2.
	DRV_AK4642_INDEX_3	This is macro DRV_AK4642_INDEX_3.
	DRV_AK4642_INDEX_4	This is macro DRV_AK4642_INDEX_4.
	DRV_AK4642_INDEX_5	This is macro DRV_AK4642_INDEX_5.
	DRV_AK4642_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4642_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4642_BUFFER_EVENT_HANDLER	Pointer to a AK4642 Driver Buffer Event handler function
	DRV_AK4642_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4642_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4642_COMMAND_EVENT_HANDLER	Pointer to a AK4642 Driver Command Event Handler Function
	DRV_AK4642_INIT	Defines the data required to initialize or reinitialize the AK4642 driver
	DRV_AK4642_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK4642_MONO_STEREO_MIC	Identifies the Mic input as Mono / Stereo.

Description

This section describes the API functions of the AK4642 Codec Driver library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_AK4642_Initialize Function

Initializes hardware and data for the instance of the AK4642 DAC module

File

drv_ak4642.h

C

```
SYS_MODULE_OBJ DRV_AK4642_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the AK4642 driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized.

Remarks

This routine must be called before any other AK4642 routine is called.

This routine should only be called once during system initialization unless DRV_AK4642_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

DRV_I2S_Initialize must be called before calling this function to initialize the data interface of this CODEC driver. DRV_I2C_Initialize must be called if SPI driver is used for handling the control interface of this CODEC driver.

Example

```
DRV_AK4642_INIT                 init;
SYS_MODULE_OBJ                  objectHandle;

init->inUse                             = true;
init->status                            = SYS_STATUS_BUSY;
init->numClients                        = 0;
init->i2sDriverModuleIndex              = ak4642Init->i2sDriverModuleIndex;
init->i2cDriverModuleIndex              = ak4642Init->i2cDriverModuleIndex;
init->samplingRate                      = DRV_AK4642_AUDIO_SAMPLING_RATE;
init->audioDataFormat                   = DRV_AK4642_AUDIO_DATA_FORMAT_MACRO;

init->isInInterruptContext              = false;

init->commandCompleteCallback = (DRV_AK4642_COMMAND_EVENT_HANDLER)0;
init->commandContextData = 0;
init->mclk_multiplier = DRV_AK4642_MCLK_SAMPLE_FREQ_MULTPLIER;


objectHandle = DRV_AK4642_Initialize(DRV_AK4642_0, (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to be initialized
init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ DRV_AK4642_Initialize
(
const SYS_MODULE_INDEX drvIndex,
const SYS_MODULE_INIT *const init
);

DRV_AK4642_Deinitialize Function

Deinitializes the specified instance of the AK4642 driver module

File

drv_ak4642.h

C

```
void DRV_AK4642_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the AK4642 driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_AK4642_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_AK4642_Initialize
SYS_STATUS          status;


DRV_AK4642_Deinitialize(object);

status = DRV_AK4642_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4642_Initialize routine

Function

void DRV_AK4642_Deinitialize(SYS_MODULE_OBJ object)

DRV_AK4642_Status Function

Gets the current status of the AK4642 driver module.

File

drv_ak4642.h

C

```
SYS_STATUS DRV_AK4642_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized

SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed

SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed

SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This routine provides the current status of the AK4642 driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_AK4642_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4642_Initialize
SYS_STATUS          AK4642Status;
```

```
AK4642Status = DRV_AK4642_Status(object);
if (SYS_STATUS_READY == AK4642Status)
{
    // This means the driver can be opened using the
    // DRV_AK4642_Open() function.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4642_Initialize routine

Function

SYS_STATUS DRV_AK4642_Status(SYS_MODULE_OBJ object)

DRV_AK4642_Tasks Function

Maintains the driver's control and data interface state machine.

File

drv_ak4642.h

C

```
void DRV_AK4642_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal control and data interface state machine and implement its control and data interface implementations. This function should be called from the SYS_Tasks() function.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4642_Initialize

while (true)
{
    DRV_AK4642_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_AK4642_Initialize)

Function

void DRV_AK4642_Tasks(SYS_MODULE_OBJ object);

b) Client Setup Functions

DRV_AK4642_Open Function

Opens the specified AK4642 driver instance and returns a handle to it

File

drv_ak4642.h

C

```
DRV_HANDLE DRV_AK4642_Open(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the number of client objects allocated via DRV_AK4642_CLIENTS_NUMBER is insufficient.
- if the client is trying to open the driver but driver has been opened exclusively by another client.
- if the driver hardware instance being opened is not initialized or is invalid.
- if the ioIntent options passed are not relevant to this driver.

Description

This routine opens the specified AK4642 driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options are not relevant to this driver. All the data transfer functions of this driver are non blocking.

AK4642 can be opened with DRV_IO_INTENT_WRITE, or DRV_IO_INTENT_READ or DRV_IO_INTENT_WRITEREAD io_intent option. This decides whether the driver is used for headphone output, or microphone input or both modes simultaneously.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_AK4642_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

Function DRV_AK4642_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_AK4642_Open(DRV_AK4642_INDEX_0, DRV_IO_INTENT_WRITEREAD | DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

DRV_HANDLE DRV_AK4642_Open

(

const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT ioIntent

)

DRV_AK4642_Close Function

Closes an opened-instance of the AK4642 driver

File

drv_ak4642.h

C

```
void DRV_AK4642_Close(const DRV_HANDLE handle);
```

Returns

- None

Description

This routine closes an opened-instance of the AK4642 driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_AK4642_Open before the caller may use the driver again

Remarks

Usually there is no need for the driver client to verify that the Close operation has completed. The driver will abort any ongoing operations when this routine is called.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_AK4642_Open

DRV_AK4642_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4642_Close(DRV_Handle handle)

c) Codec Specific Functions

DRV_AK4642_MuteOff Function

This function disables AK4642 output for soft mute.

File

drv_ak4642.h

C

```
void DRV_AK4642_MuteOff(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables AK4642 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

    DRV_AK4642_MuteOff(myAK4642Handle); //AK4642 output soft mute disabled
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4642_MuteOff(DRV_HANDLE handle)

DRV_AK4642_MuteOn Function

This function allows AK4642 output for soft mute on.

File

drv_ak4642.h

C

```
void DRV_AK4642_MuteOn(DRV_HANDLE handle);
```

Returns

None.

Description

This function Enables AK4642 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

DRV_AK4642_MuteOn(myAK4642Handle);  //AK4642 output soft muted
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4642_MuteOn(DRV_HANDLE handle);

DRV_AK4642_SamplingRateGet Function

This function gets the sampling rate set on the AK4642.

Implementation: Dynamic

File

drv_ak4642.h

C

```
uint32_t DRV_AK4642_SamplingRateGet(DRV_HANDLE handle);
```

Description

This function gets the sampling rate set on the DAC AK4642.

Remarks

None.

Example

```
uint32_t baudRate;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

baudRate = DRV_AK4642_SamplingRateGet(myAK4642Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_AK4642_SamplingRateGet(DRV_HANDLE handle)

DRV_AK4642_SamplingRateSet Function

This function sets the sampling rate of the media stream.

File

drv_ak4642.h

C

```
void DRV_AK4642_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate);
```

Returns

None.

Description

This function sets the media sampling rate for the client handle.

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

DRV_AK4642_SamplingRateSet(myAK4642Handle, 48000);  //Sets 48000 media sampling rate
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
samplingRate	Sampling frequency in Hz

Function

void DRV_AK4642_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate)

DRV_AK4642_VolumeGet Function

This function gets the volume for AK4642 CODEC.

File

drv_ak4642.h

C

```
uint8_t DRV_AK4642_VolumeGet(DRV_HANDLE handle, DRV_AK4642_CHANNEL channel);
```

Returns

None.

Description

This functions gets the current volume programmed to the CODEC AK4642.

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
uint8_t volume;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

  volume = DRV_AK4642_VolumeGet(myAK4642Handle, DRV_AK4642_CHANNEL_LEFT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
channel	argument indicating Left or Right or Both channel volume to be modified

Function

uint8_t DRV_AK4642_VolumeGet(DRV_HANDLE handle, DRV_AK4642_CHANNEL channel)

DRV_AK4642_VolumeSet Function

This function sets the volume for AK4642 CODEC.

File

drv_ak4642.h

C

```
void DRV_AK4642_VolumeSet(DRV_HANDLE handle, DRV_AK4642_CHANNEL channel, uint8_t volume);
```

Returns

None

Description

This functions sets the volume value from 0-255. The codec has DAC value to volume range mapping as :- 00 H : +12dB FF H : -115dB In order to make the volume value to dB mapping monotonically increasing from 00 to FF, re-mapping is introduced which reverses the volume value to dB mapping as well as normalizes the volume range to a more audible dB range. The current driver implementation assumes that all dB values under -60 dB are inaudible to the human ear. Re-Mapped values 00 H : -60 dB FF H : +12 dB

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

    DRV_AK4642_VolumeSet(myAK4642Handle,DRV_AK4642_CHANNEL_LEFT, 120);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
channel	argument indicating Left or Right or Both channel volume to be modified
volume	volume value specified in the range 0-255 (0x00 to 0xFF)

Function

void DRV_AK4642_VolumeSet(DRV_HANDLE handle, DRV_AK4642_CHANNEL channel, uint8_t volume);

DRV_AK4642_IntExtMicSet Function

This function sets up the codec for the internal or the external microphone use.

File

drv_ak4642.h

C

```
void DRV_AK4642_IntExtMicSet(DRV_HANDLE handle, DRV_AK4642_INT_EXT_MIC micInput);
```

Returns

None

Description

This function sets up the codec for the internal or the external microphone use.

Remarks

None.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
micInput	Internal vs External mic input

Function

void DRV_AK4642_IntExtMicSet(DRV_HANDLE handle);

DRV_AK4642_MonoStereoMicSet Function

This function sets up the codec for the Mono or Stereo microphone mode.

File

drv_ak4642.h

C

```
void DRV_AK4642_MonoStereoMicSet(DRV_HANDLE handle, DRV_AK4642_MONO_STEREO_MIC mono_stereo_mic);
```

Returns

None

Description

This function sets up the codec for the Mono or Stereo microphone mode.

Remarks

Currently the ak4642 codec does not work in the MONO_LEFT_CHANNEL mode. This issue will be followed up with AKM.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
mono_stereo_mic	Mono / Stereo mic setup

Function

void DRV_AK4642_MonoStereoMicSet(DRV_HANDLE handle);

d) Data Transfer Functions

DRV_AK4642_BufferAddWrite Function

Schedule a non-blocking driver write operation.

File

drv_ak4642.h

C

```
void DRV_AK4642_BufferAddWrite(const DRV_HANDLE handle, DRV_AK4642_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4642_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4642_BUFFER_HANDLE_INVALID:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.

- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4642_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4642_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4642 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4642 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 device instance and the DRV_AK4642_Status must have returned SYS_STATUS_READY.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE must have been specified in the DRV_AK4642_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4642_BUFFER_HANDLE bufferHandle;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

// Client registers an event handler with driver

DRV_AK4642_BufferEventHandlerSet(myAK4642Handle,
                APP_AK4642BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4642_BufferAddWrite(myAK4642handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4642_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4642BufferEventHandler(DRV_AK4642_BUFFER_EVENT event,
        DRV_AK4642_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4642_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4642_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the AK4642 instance as return by the DRV_AK4642_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.

bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK4642_BufferAddWrite
(
const DRV_HANDLE handle,
DRV_AK4642_BUFFER_HANDLE *bufferHandle,
void *buffer, size_t size
)

DRV_AK4642_BufferAddRead Function

Schedule a non-blocking driver read operation.

File

drv_ak4642.h

C

```
void DRV_AK4642_BufferAddRead(const DRV_HANDLE handle, DRV_AK4642_BUFFER_HANDLE * bufferHandle, void * buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4642_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking read operation. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance receive queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4642_BUFFER_HANDLE_INVALID

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4642_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4642_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4642 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4642 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 device instance and the DRV_AK4642_Status must have returned SYS_STATUS_READY.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ must have been specified in the DRV_AK4642_Open call.

Parameters

Parameters	Description
handle	Handle of the AK4642 instance as return by the DRV_AK4642_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK4642_BufferAddRead
(
const DRV_HANDLE handle,
DRV_AK4642_BUFFER_HANDLE *bufferHandle,

void *buffer, size_t size

)

DRV_AK4642_BufferAddWriteRead Function

Schedule a non-blocking driver write-read operation.

Implementation: Dynamic

File

drv_ak4642.h

C

```
void DRV_AK4642_BufferAddWriteRead(const DRV_HANDLE handle, DRV_AK4642_BUFFER_HANDLE * bufferHandle, void *
transmitBuffer, void * receiveBuffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4642_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write-read operation. The function returns with a valid buffer handle in the bufferHandle argument if the write-read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4642_BUFFER_EVENT_COMPLETE:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only or write only
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4642_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4642_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4642 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4642 driver instance. It should not otherwise be called directly in an ISR.

This function is useful when there is valid read expected for every AK4642 write. The transmit and receive size must be same.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 device instance and the DRV_AK4642_Status must have returned SYS_STATUS_READY.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READWRITE must have been specified in the DRV_AK4642_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybufferTx[MY_BUFFER_SIZE];
uint8_t mybufferRx[MY_BUFFER_SIZE];
DRV_AK4642_BUFFER_HANDLE bufferHandle;

// myak4642Handle is the handle returned
// by the DRV_AK4642_Open function.

// Client registers an event handler with driver

DRV_AK4642_BufferEventHandlerSet(myak4642Handle,
                APP_AK4642BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4642_BufferAddWriteRead(myak4642handle, &bufferHandle,
                                    mybufferTx,mybufferRx,MY_BUFFER_SIZE);

if(DRV_AK4642_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
```

```
}

// Event is received when
// the buffer is processed.

void APP_AK4642BufferEventHandler(DRV_AK4642_BUFFER_EVENT event,
        DRV_AK4642_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4642_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4642_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the AK4642 instance as returned by the DRV_AK4642_Open function
bufferHandle	Pointer to an argument that will contain the return buffer handle
transmitBuffer	The buffer where the transmit data will be stored
receiveBuffer	The buffer where the received data will be stored
size	Buffer size in bytes

Function

void DRV_AK4642_BufferAddWriteRead
(
const DRV_HANDLE handle,
DRV_AK4642_BUFFER_HANDLE *bufferHandle,
void *transmitBuffer,
void *receiveBuffer,
size_t size
)

DRV_AK4642_BufferEventHandlerSet Function

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

File

drv_ak4642.h

C

```
void DRV_AK4642_BufferEventHandlerSet(DRV_HANDLE handle, const DRV_AK4642_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls DRV_AK4642_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4642_BUFFER_HANDLE bufferHandle;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

// Client registers an event handler with driver

DRV_AK4642_BufferEventHandlerSet(myAK4642Handle,
                APP_AK4642BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4642_BufferAddWrite(myAK4642handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4642_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4642BufferEventHandler(DRV_AK4642_BUFFER_EVENT event,
        DRV_AK4642_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4642_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4642_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4642_BufferEventHandlerSet
(
DRV_HANDLE handle,
const DRV_AK4642_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t contextHandle
)

e) Other Functions

DRV_AK4642_CommandEventHandlerSet Function

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

File

drv_ak4642.h

C

```
void DRV_AK4642_CommandEventHandlerSet(DRV_HANDLE handle, const DRV_AK4642_COMMAND_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

When a client calls DRV_AK4642_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "AK4642 CODEC Specific Client Routines" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the command has completed, it does not need to register a callback.

Preconditions

The DRV_AK4642_Initialize routine must have been called for the specified AK4642 driver instance.

DRV_AK4642_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4642_BUFFER_HANDLE bufferHandle;

// myAK4642Handle is the handle returned
// by the DRV_AK4642_Open function.

// Client registers an event handler with driver

DRV_AK4642_CommandEventHandlerSet(myAK4642Handle,
                APP_AK4642CommandEventHandler, (uintptr_t)&myAppObj);

DRV_AK4642_DeEmphasisFilterSet(myAK4642Handle, DRV_AK4642_DEEMPHASIS_FILTER_44_1KHZ)

// Event is received when
// the buffer is processed.

void APP_AK4642CommandEventHandler(uintptr_t contextHandle)
{
```

```
    // contextHandle points to myAppObj.

    switch(event)
    {
            // Last Submitted command is completed.
            // Perform further processing here
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4642_CommandEventHandlerSet

(

DRV_HANDLE handle,

const DRV_AK4642_COMMAND_EVENT_HANDLER eventHandler,

const uintptr_t contextHandle

)

DRV_AK4642_VersionGet Function

This function returns the version of AK4642 driver

File

drv_ak4642.h

C

```
uint32_t DRV_AK4642_VersionGet();
```

Returns

returns the version of AK4642 driver.

Description

The version number returned from the DRV_AK4642_VersionGet function is an unsigned integer in the following decimal format. * 10000 + * 100 + Where the numbers are represented in decimal and the meaning is the same as above. Note that there is no numerical representation of release type.

Remarks

None.

Preconditions

None.

Example 1

For version "0.03a", return: 0 * 10000 + 3 * 100 + 0 For version "1.00", return: 1 * 100000 + 0 * 100 + 0

Example 2

```
uint32_t AK4642version;
AK4642version = DRV_AK4642_VersionGet();
```

Function

uint32_t DRV_AK4642_VersionGet(void)

DRV_AK4642_VersionStrGet Function

This function returns the version of AK4642 driver in string format.

File

drv_ak4642.h

C

```
int8_t* DRV_AK4642_VersionStrGet();
```

Returns

returns a string containing the version of AK4642 driver.

Description

The DRV_AK4642_VersionStrGet function returns a string in the format: ".[.][]" Where: is the AK4642 driver's version number. is the AK4642 driver's version number. is an optional "patch" or "dot" release number (which is not included in the string if it equals "00"). is an optional release type ("a" for alpha, "b" for beta ? not the entire word spelled out) that is not included if the release is a production version (I.e. Not an alpha or beta).

The String does not contain any spaces. For example, "0.03a" "1.00"

Remarks

None

Preconditions

None.

Example

```
int8_t *AK4642string;
AK4642string = DRV_AK4642_VersionStrGet();
```

Function

int8_t* DRV_AK4642_VersionStrGet(void)

f) Data Types and Constants

_DRV_AK4642_H Macro

File

drv_ak4642.h

C

```
#define _DRV_AK4642_H
```

Description

Include files.

DRV_AK4642_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_ak4642.h

C

```
#define DRV_AK4642_BUFFER_HANDLE_INVALID ((DRV_AK4642_BUFFER_HANDLE)(-1))
```

Description

AK4642 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_AK4642_BufferAddWrite() and the DRV_AK4642_BufferAddRead() function if the buffer add request was not successful.

Remarks

None.

DRV_AK4642_COUNT Macro

Number of valid AK4642 driver indices

File

drv_ak4642.h

C

```
#define DRV_AK4642_COUNT
```

Description

AK4642 Driver Module Count

This constant identifies the maximum number of AK4642 Driver instances that should be defined by the application. Defining more instances than this constant will waste RAM memory space.

This constant can also be used by the application to identify the number of AK4642 instances on this microcontroller.

Remarks

This value is part-specific.

DRV_AK4642_INDEX_0 Macro

AK4642 driver index definitions

File

drv_ak4642.h

C

```
#define DRV_AK4642_INDEX_0 0
```

Description

Driver AK4642 Module Index

These constants provide AK4642 driver index definition.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_AK4642_Initialize and DRV_AK4642_Open routines to identify the driver instance in use.

DRV_AK4642_INDEX_1 Macro

File

drv_ak4642.h

C

```
#define DRV_AK4642_INDEX_1 1
```

Description

This is macro DRV_AK4642_INDEX_1.

DRV_AK4642_INDEX_2 Macro

File

drv_ak4642.h

C

```
#define DRV_AK4642_INDEX_2 2
```

Description

This is macro DRV_AK4642_INDEX_2.

DRV_AK4642_INDEX_3 Macro

File

drv_ak4642.h

C

```
#define DRV_AK4642_INDEX_3 3
```

Description

This is macro DRV_AK4642_INDEX_3.

DRV_AK4642_INDEX_4 Macro

File

drv_ak4642.h

C

```
#define DRV_AK4642_INDEX_4 4
```

Description

This is macro DRV_AK4642_INDEX_4.

DRV_AK4642_INDEX_5 Macro

File

drv_ak4642.h

C

```
#define DRV_AK4642_INDEX_5 5
```

Description

This is macro DRV_AK4642_INDEX_5.

DRV_AK4642_AUDIO_DATA_FORMAT Enumeration

Identifies the Serial Audio data interface format.

File

drv_ak4642.h

C

```
typedef enum {
  DRV_AK4642_AUDIO_DATA_FORMAT_NOT_APPLICABLE = 0,
  DRV_AK4642_AUDIO_DATA_FORMAT_16BITMSB_SDTO_16BITLSB_SDTI,
  DRV_AK4642_AUDIO_DATA_FORMAT_16BITMSB_SDTO_16BITMSB_SDTI,
  DRV_AK4642_AUDIO_DATA_FORMAT_I2S
} DRV_AK4642_AUDIO_DATA_FORMAT;
```

Description

AK4642 Audio data format

This enumeration identifies Serial Audio data interface format.

DRV_AK4642_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_ak4642.h

C

```
typedef enum {
  DRV_AK4642_BUFFER_EVENT_COMPLETE,
  DRV_AK4642_BUFFER_EVENT_ERROR,
  DRV_AK4642_BUFFER_EVENT_ABORT
} DRV_AK4642_BUFFER_EVENT;
```

Members

Members	Description
DRV_AK4642_BUFFER_EVENT_COMPLETE	Data was transferred successfully.
DRV_AK4642_BUFFER_EVENT_ERROR	Error while processing the request
DRV_AK4642_BUFFER_EVENT_ABORT	Data transfer aborted (Applicable in DMA mode)

Description

AK4642 Driver Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_AK4642_BufferAddWrite() or the DRV_AK4642_BufferAddRead() function.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_AK4642_BufferEventHandlerSet function when a buffer transfer request is completed.

DRV_AK4642_BUFFER_EVENT_HANDLER Type

Pointer to a AK4642 Driver Buffer Event handler function

File

drv_ak4642.h

C

```
typedef void (* DRV_AK4642_BUFFER_EVENT_HANDLER)(DRV_AK4642_BUFFER_EVENT event, DRV_AK4642_BUFFER_HANDLE
bufferHandle, uintptr_t contextHandle);
```

Returns

None.

Description

AK4642 Driver Buffer Event Handler Function

This data type defines the required function signature for the AK4642 driver buffer event handling callback function. A client must register a pointer to a buffer event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_AK4642_BUFFER_EVENT_COMPLETE, this means that the data was transferred successfully.

If the event is DRV_AK4642_BUFFER_EVENT_ERROR, this means that the data was not transferred successfully. The bufferHandle parameter contains the buffer handle of the buffer that failed. The DRV_AK4642_BufferProcessedSizeGet() function can be called to find out how many bytes were processed.

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4642_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The buffer handle in bufferHandle expires after this event handler exits. In that the buffer object that was allocated is deallocated by the driver after the event handler exits.

The event handler function executes in the data driver(i2S) peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

DRV_AK4642_BufferAddWrite function can be called in the event handler to add a buffer to the driver queue.

Example

```
void APP_MyBufferEventHandler( DRV_AK4642_BUFFER_EVENT event,
                               DRV_AK4642_BUFFER_HANDLE bufferHandle,
                               uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_AK4642_BUFFER_EVENT_COMPLETE:
            // Handle the completed buffer.
        break;

        case DRV_AK4642_BUFFER_EVENT_ERROR:
        default:
            // Handle error.
        break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the event relates
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4642_BUFFER_HANDLE Type

Handle identifying a write buffer passed to the driver.

File

drv_ak4642.h

C

```
typedef uintptr_t DRV_AK4642_BUFFER_HANDLE;
```

Description

AK4642 Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_AK4642_BufferAddWrite() or DRV_AK4642_BufferAddRead() function. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None

DRV_AK4642_CHANNEL Enumeration

Identifies Left/Right Audio channel

File

drv_ak4642.h

C

```
typedef enum {
  DRV_AK4642_CHANNEL_LEFT,
  DRV_AK4642_CHANNEL_RIGHT,
  DRV_AK4642_CHANNEL_LEFT_RIGHT,
  DRV_AK4642_NUMBER_OF_CHANNELS
} DRV_AK4642_CHANNEL;
```

Description

AK4642 Audio Channel

This enumeration identifies Left/Right Audio channel

Remarks

None.

DRV_AK4642_COMMAND_EVENT_HANDLER Type

Pointer to a AK4642 Driver Command Event Handler Function

File

drv_ak4642.h

C

```
typedef void (* DRV_AK4642_COMMAND_EVENT_HANDLER)(uintptr_t contextHandle);
```

Returns

None.

Description

AK4642 Driver Command Event Handler Function

This data type defines the required function signature for the AK4642 driver command event handling callback function.

A command is a control instruction to the AK4642 CODEC. Example Mute ON/OFF, Zero Detect Enable/Disable etc.

A client must register a pointer to a command event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive command related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

The occurrence of this call back means that the last control command was transferred successfully.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4642_CommandEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The event handler function executes in the control data driver interrupt context. It is recommended of the application to not perform process intensive or blocking operations with in this function.

Example

```
void APP_AK4642CommandEventHandler( uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    // Last Submitted command is completed.
    // Perform further processing here
}
```

Parameters

Parameters	Description
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4642_INIT Structure

Defines the data required to initialize or reinitialize the AK4642 driver

File

drv_ak4642.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX i2sDriverModuleIndex;
```

```
  SYS_MODULE_INDEX i2cDriverModuleIndex;
  uint32_t samplingRate;
  uint8_t volume;
  DRV_AK4642_AUDIO_DATA_FORMAT audioDataFormat;
} DRV_AK4642_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX i2sDriverModuleIndex;	Identifies data module(I2S) driver ID for data interface of CODEC
SYS_MODULE_INDEX i2cDriverModuleIndex;	Identifies data module(I2C) driver ID for control interface of CODEC
uint32_t samplingRate;	Sampling rate
uint8_t volume;	Volume
DRV_AK4642_AUDIO_DATA_FORMAT audioDataFormat;	Identifies the Audio data format

Description

AK4642 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the AK4642 CODEC driver.

Remarks

None.

DRV_AK4642_INT_EXT_MIC Enumeration

Identifies the Mic input source.

File

drv_ak4642.h

C

```
typedef enum {
  INT_MIC,
  EXT_MIC
} DRV_AK4642_INT_EXT_MIC;
```

Description

AK4642 Mic Internal / External Input

This enumeration identifies the Mic input source.

DRV_AK4642_MONO_STEREO_MIC Enumeration

Identifies the Mic input as Mono / Stereo.

File

drv_ak4642.h

C

```
typedef enum {
  ALL_ZEROS,
  MONO_RIGHT_CHANNEL,
  MONO_LEFT_CHANNEL,
  STEREO
} DRV_AK4642_MONO_STEREO_MIC;
```

Description

AK4642 Mic Mono / Stereo Input

This enumeration identifies the Mic input as Mono / Stereo.

Files

Files

Name	Description
drv_ak4642.h	AK4642 CODEC Driver Interface header file
drv_ak4642_config_template.h	AK4642 Codec Driver Configuration Template.

Description

This section lists the source and header files used by the AK4642 Codec Driver Library.

drv_ak4642.h

AK4642 CODEC Driver Interface header file

Enumerations

	Name	Description
	DRV_AK4642_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4642_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4642_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4642_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK4642_MONO_STEREO_MIC	Identifies the Mic input as Mono / Stereo.

Functions

	Name	Description
	DRV_AK4642_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_AK4642_BufferAddWrite	Schedule a non-blocking driver write operation.
	DRV_AK4642_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_AK4642_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_AK4642_Close	Closes an opened-instance of the AK4642 driver
	DRV_AK4642_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.
	DRV_AK4642_Deinitialize	Deinitializes the specified instance of the AK4642 driver module
	DRV_AK4642_Initialize	Initializes hardware and data for the instance of the AK4642 DAC module
	DRV_AK4642_IntExtMicSet	This function sets up the codec for the internal or the external microphone use.
	DRV_AK4642_MonoStereoMicSet	This function sets up the codec for the Mono or Stereo microphone mode.
	DRV_AK4642_MuteOff	This function disables AK4642 output for soft mute.
	DRV_AK4642_MuteOn	This function allows AK4642 output for soft mute on.
	DRV_AK4642_Open	Opens the specified AK4642 driver instance and returns a handle to it
	DRV_AK4642_SamplingRateGet	This function gets the sampling rate set on the AK4642. **Implementation:** Dynamic
	DRV_AK4642_SamplingRateSet	This function sets the sampling rate of the media stream.
	DRV_AK4642_Status	Gets the current status of the AK4642 driver module.
	DRV_AK4642_Tasks	Maintains the driver's control and data interface state machine.
	DRV_AK4642_VersionGet	This function returns the version of AK4642 driver
	DRV_AK4642_VersionStrGet	This function returns the version of AK4642 driver in string format.
	DRV_AK4642_VolumeGet	This function gets the volume for AK4642 CODEC.
	DRV_AK4642_VolumeSet	This function sets the volume for AK4642 CODEC.

Macros

	Name	Description
	_DRV_AK4642_H	Include files.
	DRV_AK4642_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4642_COUNT	Number of valid AK4642 driver indices

	DRV_AK4642_INDEX_0	AK4642 driver index definitions
	DRV_AK4642_INDEX_1	This is macro DRV_AK4642_INDEX_1.
	DRV_AK4642_INDEX_2	This is macro DRV_AK4642_INDEX_2.
	DRV_AK4642_INDEX_3	This is macro DRV_AK4642_INDEX_3.
	DRV_AK4642_INDEX_4	This is macro DRV_AK4642_INDEX_4.
	DRV_AK4642_INDEX_5	This is macro DRV_AK4642_INDEX_5.

Structures

	Name	Description
	DRV_AK4642_INIT	Defines the data required to initialize or reinitialize the AK4642 driver

Types

	Name	Description
	DRV_AK4642_BUFFER_EVENT_HANDLER	Pointer to a AK4642 Driver Buffer Event handler function
	DRV_AK4642_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4642_COMMAND_EVENT_HANDLER	Pointer to a AK4642 Driver Command Event Handler Function

Description

AK4642 CODEC Driver Interface

The AK4642 CODEC device driver interface provides a simple interface to manage the AK4642 16/24-Bit CODEC that can be interfaced Microchip Microcontroller. This file provides the interface definition for the AK4642 CODEC device driver.

File Name

drv_ak4642.h

Company

Microchip Technology Inc.

drv_ak4642_config_template.h

AK4642 Codec Driver Configuration Template.

Macros

	Name	Description
	DRV_AK4642_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4642_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4642_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4642_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4642_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4642_MCLK_SOURCE	Indicate the input clock frequency to generate the MCLK to codec.

Description

AK4642 Codec Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_ak4642_config_template.h

Company

Microchip Technology Inc.

AK4953 Codec Driver Library

This topic describes the AK4953 Codec Driver Library.

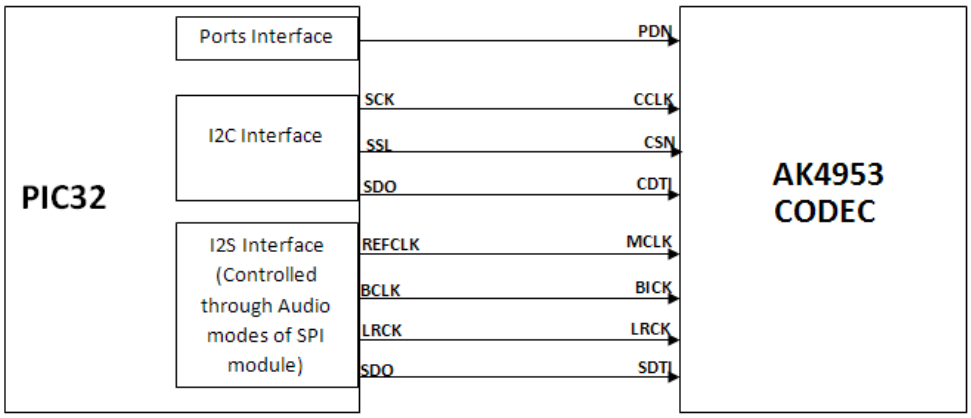
Introduction

This library provides an interface to manage the AK4953 Codec that is serially interfaced to a Microchip microcontroller for providing Audio Solutions.

Description

The AK4953 module is 16/24-bit Audio Codec from Asahi Kasei Microdevices Corporation. The AK4953 can be interfaced to Microchip microcontrollers through I2C and I2S serial interfaces. The I2C interface is used for control command transfer. The I2S interface is used for Audio data output.

A typical interface of AK4953 to a Microchip PIC32 device is provided in the following diagram:



Features

The AK4953 Codec supports the following features:

- Audio Interface Format: MSB first
- ADC: 24-bit MSB justified, 16/24-bit I2S
- DAC: 24-bit MSB justified, 1-6bit LSB justified, 24-bit LSB justified, 16/24-bit I2S
- Sampling Frequency Range: 8 kHz to 192 kHz
- Digital Volume Control: +12dB ~ .115dB, 0.5dB Step
- SoftMute: On and Off
- Master Clock Frequencies: 32 fs/64 fs/128 fs/256 fs

Using the Library

This topic describes the basic architecture of the AK4953 Codec Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_AK4953.h`

The interface to the AK4953 Codec Driver library is defined in the `drv_AK4953.h` header file. Any C language source (`.c`) file that uses the AK4953 Codec Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The AK4953 Codec Driver Library provides an API interface to transfer control commands and digital audio data to the serially interfaced AK4953 DAC module. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the AK4953 Codec Driver Library.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Status Functions	Provides status functions.
Other Functions	Provides driver specific miscellaneous functions such as sampling rate setting, control command functions, etc.

Data Types and Constants	These data types and constants are required while interacting and setting up the AK4953 Codec Driver Library.

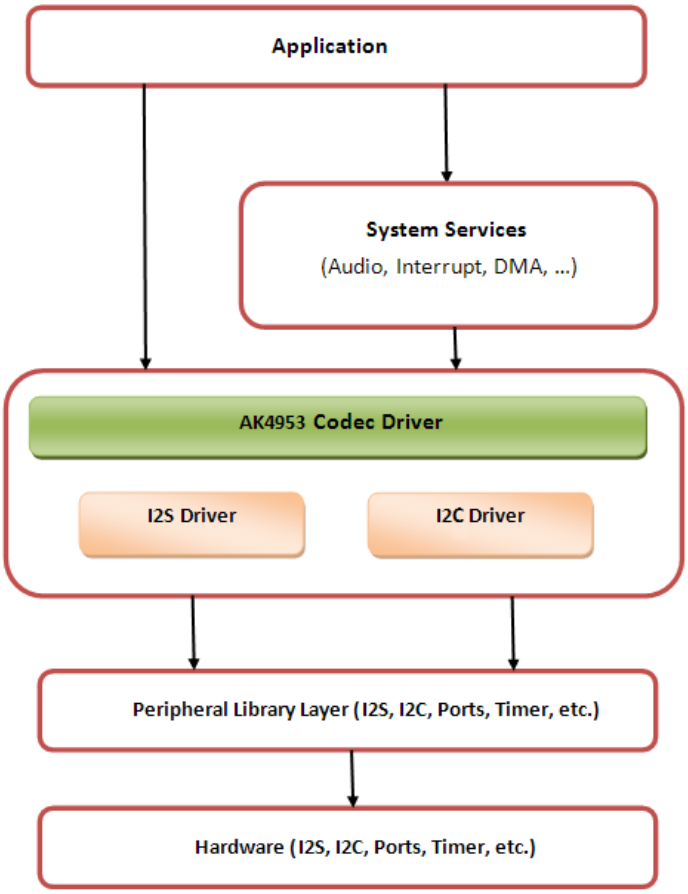
Abstraction Model

This library provides a low-level abstraction of the AK4953 Codec Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The abstraction model shown in the following diagram depicts how the AK4953 Codec Driver is positioned in the MPLAB Harmony framework. The AK4953 Codec Driver uses the SPI and I2S drivers for control and audio data transfers to the AK4953 module.

AK4953 Driver Abstraction Model



How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

System Access

This topic describes system initialization, implementations, and includes a system access code example.

Description

System Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the AK4953 module would be initialized with the following configuration settings (either passed dynamically

at run time using DRV_AK4953_INIT or by using Initialization Overrides) that are supported by the specific AK4953 device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- I2C driver module index. The module index should be same as the one used in initializing the I2C Driver.
- I2S driver module index. The module index should be same as the one used in initializing the I2S Driver.
- Sampling rate
- Audio data format. The audio data format should match with the audio data format settings done in I2S driver initialization
- Power down pin port initialization
- Queue size for the audio data transmit buffer

The DRV_AK4953_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_ AK4953_Deinitialize, DRV_ AK4953_Status and DRV_I2S_Tasks.

Implementations

The AK4953 Codec Driver can has the following implementation:

Description	MPLAB Harmony Components
Dedicated hardware for control (I2C) and data (I2S) interface.	Standard MPLAB Harmony drivers for I2C and I2S interfaces.

Example:

```
DRV_AK4953_INIT drvak4953Codec0InitData =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .i2sDriverModuleIndex = DRV_AK4953_I2S_DRIVER_MODULE_INDEX_IDX0,
    .i2cDriverModuleIndex = DRV_AK4953_I2C_DRIVER_MODULE_INDEX_IDX0,
    .volume = DRV_AK4953_VOLUME,
    .queueSizeTransmit = DRV_AK4953_TRANSMIT_QUEUE_SIZE,
};

// Initialize the I2C driver
   DRV_I2C0_Initialize();

// Initialize the I2S driver. The I2S module index should be same as the one used in initializing
// the I2S driver.
   sysObj.drvI2S0 = DRV_I2S_Initialize(DRV_I2S_INDEX_0, (SYS_MODULE_INIT *)&drvI2S0InitData);

// Initialize the Codec driver
   sysObj.drvak4953Codec0 = DRV_AK4953_Initialize(DRV_AK4953_INDEX_0, (SYS_MODULE_INIT
*)&drvak4953Codec0InitData);

if (SYS_MODULE_OBJ_INVALID == AK4953DevObject)
{
// Handle error
}
```

Task Routine

The DRV_AK4953_Tasks will be called from the System Task Service.

Client Access

For the application to start using an instance of the module, it must call the DRV_AK4953_Open function. The DRV_AK4953_Open provides a driver handle to the AK4953 Codec Driver instance for operations. If the driver is deinitialized using the function DRV_AK4953_Deinitialize, the application must call the DRV_AK4953_Open function again to set up the instance of the driver.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

Client Operations

This topic provides information on client operations and includes a control command and audio buffered data operation flow diagram.

Description

Client operations provide the API interface for control command and audio data transfer to the AK4953 Codec.

The following AK4953 Codec specific control command functions are provided:

- DRV_AK4953_SamplingRateSet
- DRV_AK4953_SamplingRateGet

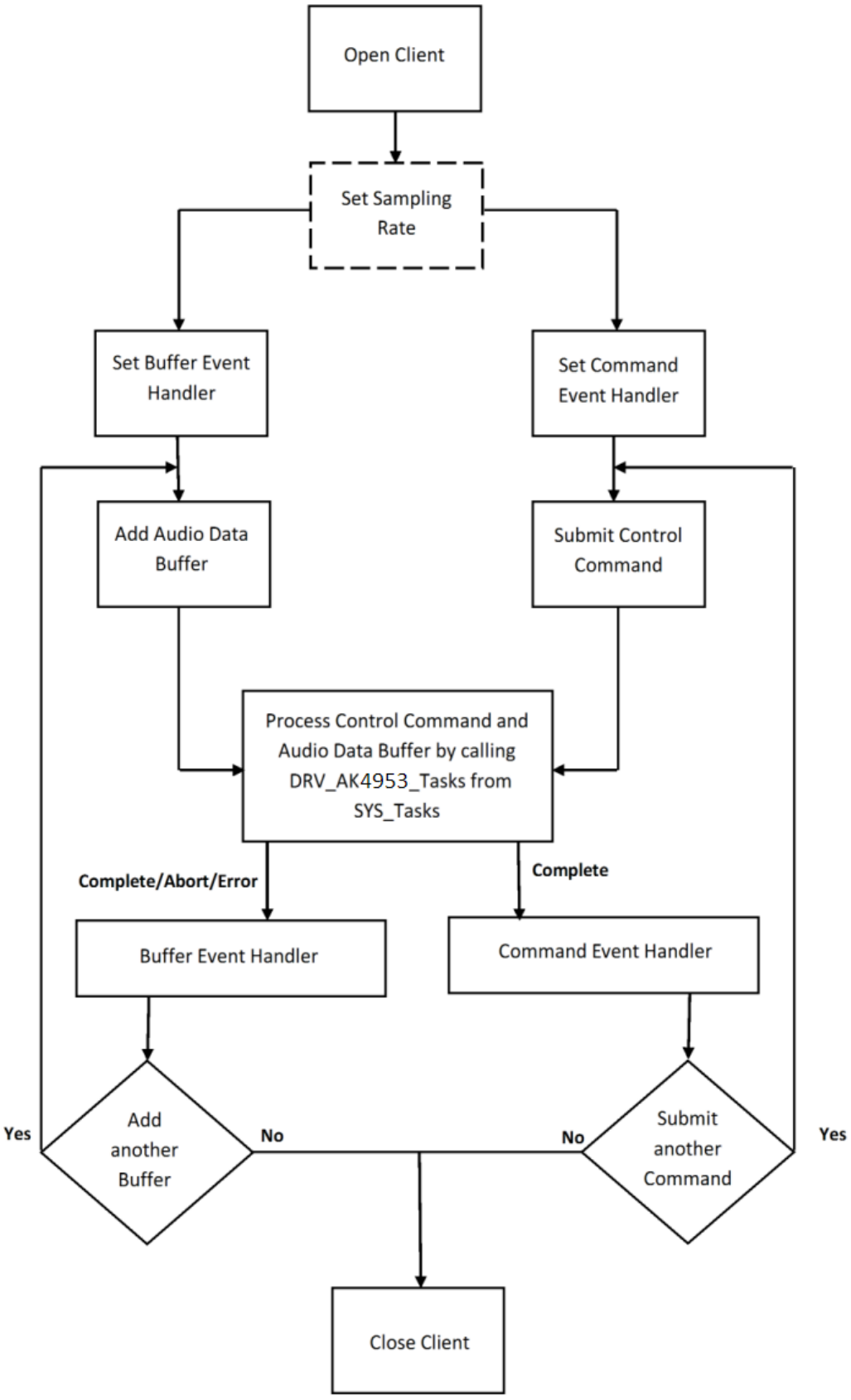
- DRV_AK4953_VolumeSet
- DRV_AK4953_VolumeGet
- DRV_AK4953_MuteOn
- DRV_AK4953_MuteOff
- DRV_AK4953_IntExtMicSet
- DRV_AK4953_MonoStereoMicSet

These functions schedule a non-blocking control command transfer operation. These functions submit the control command request to the AK4953 Codec. These functions submit the control command request to I2C Driver transmit queue, the request is processed immediately if it is the first request, or processed when the previous request is complete.

DRV_AK4953_BufferAddWrite, DRV_AK4953_BufferAddRead, and DRV_AK4953_BufferAddWriteRead are buffered data operation functions.

These functions schedule non-blocking audio data transfer operations. These functions add the request to I2S Driver transmit or receive buffer queue depends on the request type, and are executed immediately if it is the first buffer, or executed later when the previous buffer is complete. The driver notifies the client with DRV_AK4953_BUFFER_EVENT_COMPLETE, DRV_AK4953_BUFFER_EVENT_ERROR, or DRV_AK4953_BUFFER_EVENT_ABORT events.

The following diagram illustrates the control commands and audio buffered data operations.



Note: It is not necessary to close and reopen the client between multiple transfers.

Configuring the Library

Macros

	Name	Description
	DRV_AK4953_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4953_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4953_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4953_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4953_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4953_MCLK_SOURCE	Indicate the input clock frequency to generate the MCLK to codec.
	DRV_AK4953_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.

Description

The configuration of the AK4953 Codec Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the AK4953 Codec Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the AK4953 Codec Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_AK4953_BCLK_BIT_CLK_DIVISOR Macro

Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_BCLK_BIT_CLK_DIVISOR
```

Description

AK4953 BCLK to LRCK Ratio to Generate Audio Stream

Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

Following BCLK to LRCK ratios are supported 16bit data 16 bit channel :- 32fs, hence divisor would be 8 16bit data 32 bit channel :- 64fs, hence divisor would be 4

Remarks

None.

DRV_AK4953_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_CLIENTS_NUMBER DRV_AK4953_INSTANCES_NUMBER
```

Description

AK4953 Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. Typically only one client could be connected to one hardware instance. This value represents the total number of clients to be supported across all hardware instances. Therefore, if there are five AK4953 hardware interfaces, this number will be 5.

Remarks

None.

DRV_AK4953_INPUT_REFCLOCK Macro

Identifies the input REFCLOCK source to generate the MCLK to codec.

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_INPUT_REFCLOCK
```

Description

AK4953 Input reference clock

Identifies the input REFCLOCK source to generate the MCLK to codec.

Remarks

None.

DRV_AK4953_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_INSTANCES_NUMBER
```

Description

AK4953 driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of AK4953 CODEC modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None.

DRV_AK4953_MCLK_SAMPLE_FREQ_MULTPLIER Macro

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_MCLK_SAMPLE_FREQ_MULTPLIER
```

Description

AK4953 MCLK to LRCK Ratio to Generate Audio Stream

Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency I2S sampling frequency

Supported MCLK to Sampling frequency Ratios are as below 256fs, 384fs, 512fs, 768fs or 1152fs

Remarks

None

DRV_AK4953_MCLK_SOURCE Macro

Indicate the input clock frequency to generate the MCLK to codec.

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_MCLK_SOURCE
```

Description

AK4953 Data Interface Master Clock Speed configuration

Indicate the input clock frequency to generate the MCLK to codec.

Remarks

None.

DRV_AK4953_QUEUE_DEPTH_COMBINED Macro

Number of entries of all queues in all instances of the driver.

File

drv_ak4953_config_template.h

C

```
#define DRV_AK4953_QUEUE_DEPTH_COMBINED
```

Description

AK4953 Driver Buffer Queue Entries

This macro defined the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for transmit operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build). The hardware instance transmit buffer queue will queue transmit buffers submitted by the DRV_AK4953_BufferAddWrite function.

A buffer queue will contains buffer queue entries, each related to a BufferAdd request. This configuration macro defines total number of buffer entries that will be available for use between all AK4953 driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking write requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. More the number of buffer entries, greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its transmit buffer queue size.

As an example, consider the case of static single client driver application where full duplex non blocking operation is desired without queuing, the minimum transmit queue depth and minimum receive queue depth should be 1. Hence the total number of buffer entries should be 2.

As an example, consider the case of a dynamic driver (say two instances) where instance one will queue up to three write requests and up to two read requests, and instance two will queue up to two write requests and up to six read requests, the value of this macro should be 13 (2 + 3 + 2 + 6).

Configuring the MHC

Provides examples on how to configure the MPLAB Harmony Configurator (MHC) for a specific driver.

Description

The following three figures show examples of MHC configurations for the AK4953 Codec Driver, I2S Driver, and the I2C Driver.

Figure 1: AK4953 Codec Driver MHC Configuration

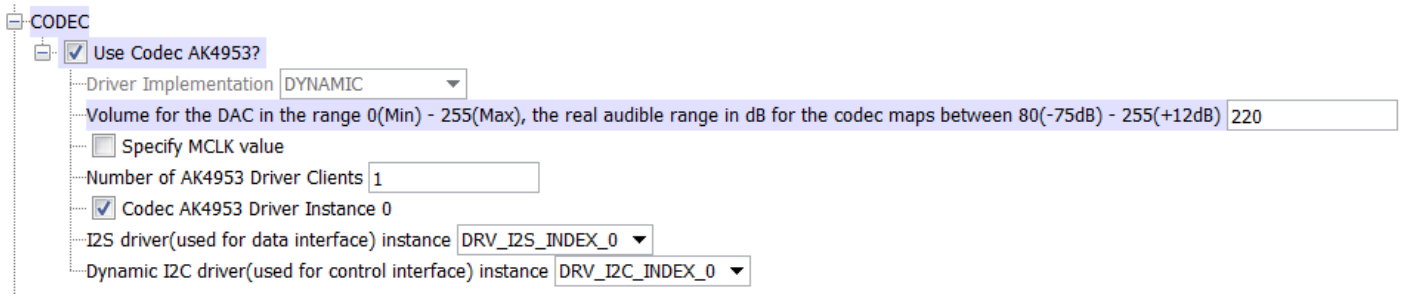


Figure 2: I2S Driver MHC Configuration

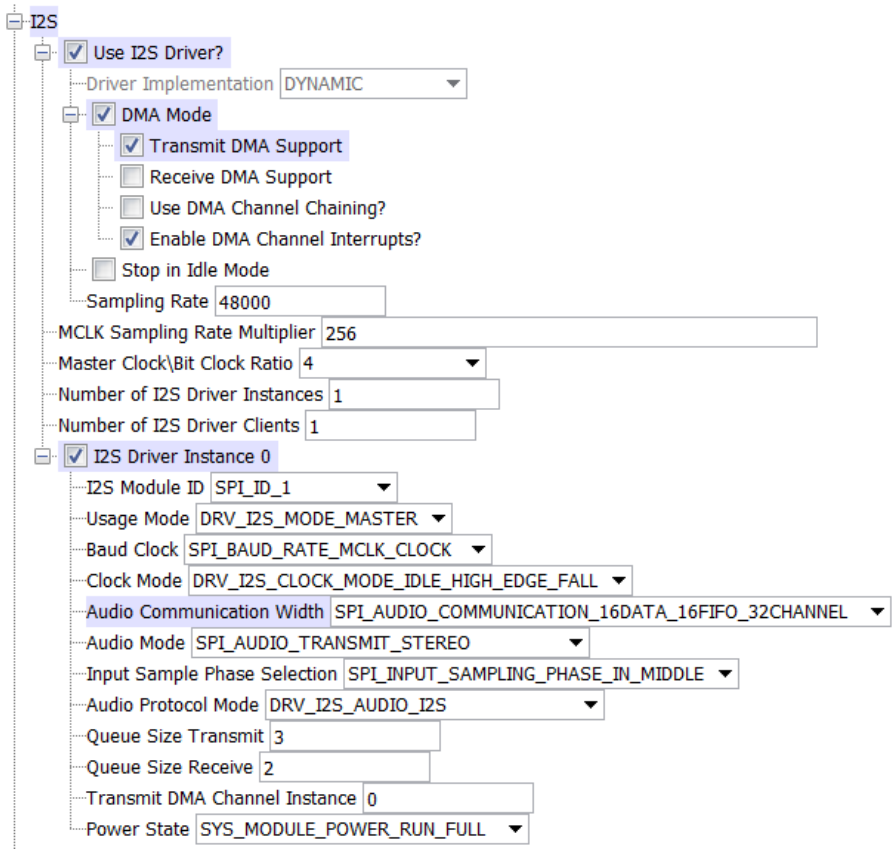
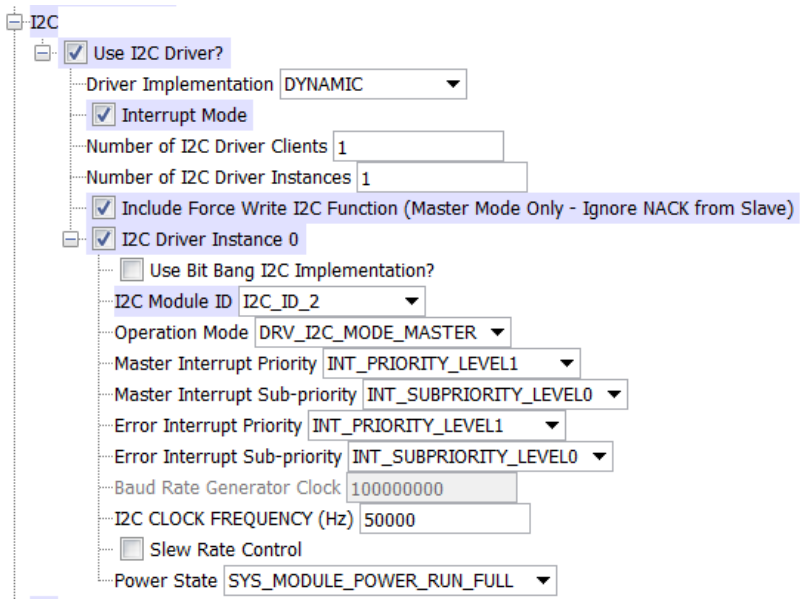


Figure 3: I2C Driver MHC Configuration



Migrating the AK4953 Driver From Earlier Versions of Microchip Harmony

Prior to version 1.08 of MPLAB Harmony, the AK4953 Codec Driver Library used the static I2C driver implementation. Beginning with v1.08 of MPLAB Harmony, applications must use the Dynamic Driver implementation with the MHC configured as shown in Figure 3. In addition, PIC32MZ configurations require the "Include Force Write I2C Function (Master Mode Only - Ignore NACK from Slave)" option to be selected.

Building the Library

This section lists the files that are available in the AK4953 Codec Driver Library.

Description

This section list the files that are available in the `/src` folder of the AK4953 Codec Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/codec/ak4953`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ak4953.h`	Header file that exports the driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_ak4953.c`	This file contains implementation of the AK4953 Codec Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The AK4953 Codec Driver Library depends on the following modules:

- I2S Driver Library
- I2C Driver Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_AK4953_Initialize	Initializes hardware and data for the instance of the AK4953 DAC module. **Implementation:** Dynamic
	DRV_AK4953_Deinitialize	Deinitializes the specified instance of the AK4953 driver module. **Implementation:** Dynamic
	DRV_AK4953_Open	Opens the specified AK4953 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_AK4953_Close	Closes an opened-instance of the AK4953 driver. **Implementation:** Dynamic
	DRV_AK4953_Tasks	Maintains the driver's control and data interface state machine. **Implementation:** Dynamic
	DRV_AK4953_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished. **Implementation:** Dynamic
	DRV_AK4953_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_AK4953_SamplingRateSet	This function sets the sampling rate of the media stream. **Implementation:** Dynamic

b) Status Functions

	Name	Description
	DRV_AK4953_SamplingRateGet	This function gets the sampling rate set on the DAC AK4953. **Implementation:** Dynamic

	DRV_AK4953_Status	Gets the current status of the AK4953 driver module. **Implementation:** Dynamic
	DRV_AK4953_VersionGet	This function returns the version of AK4953 driver. **Implementation:** Dynamic
	DRV_AK4953_VersionStrGet	This function returns the version of AK4953 driver in string format. **Implementation:** Dynamic
	DRV_AK4953_VolumeGet	This function gets the volume for AK4953 CODEC. **Implementation:** Dynamic

c) Other Functions

	Name	Description
	DRV_AK4953_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic
	DRV_AK4953_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_AK4953_MuteOff	This function disables AK4953 output for soft mute. **Implementation:** Dynamic
	DRV_AK4953_MuteOn	This function allows AK4953 output for soft mute on. **Implementation:** Dynamic
	DRV_AK4953_VolumeSet	This function sets the volume for AK4953 CODEC. **Implementation:** Dynamic
	DRV_AK4953_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_AK4953_IntExtMicSet	This function sets up the codec for the internal or the external microphone use.
	DRV_AK4953_MonoStereoMicSet	This function sets up the codec for the Mono or Stereo microphone mode.

d) Data Types and Constants

	Name	Description
	DRV_AK4953_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4953_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4953_BUFFER_EVENT_HANDLER	Pointer to a AK4953 Driver Buffer Event handler function
	DRV_AK4953_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4953_COMMAND_EVENT_HANDLER	Pointer to a AK4953 Driver Command Event Handler Function
	DRV_AK4953_DIGITAL_BLOCK_CONTROL	Identifies Bass-Boost Control function
	DRV_AK4953_INIT	Defines the data required to initialize or reinitialize the AK4953 driver
	_DRV_AK4953_H	Include files.
	DRV_AK4953_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4953_COUNT	Number of valid AK4953 driver indices
	DRV_AK4953_INDEX_0	AK4953 driver index definitions
	DRV_AK4953_INDEX_1	This is macro DRV_AK4953_INDEX_1.
	DRV_AK4953_INDEX_2	This is macro DRV_AK4953_INDEX_2.
	DRV_AK4953_INDEX_3	This is macro DRV_AK4953_INDEX_3.
	DRV_AK4953_INDEX_4	This is macro DRV_AK4953_INDEX_4.
	DRV_AK4953_INDEX_5	This is macro DRV_AK4953_INDEX_5.
	DRV_I2C_INDEX	This is macro DRV_I2C_INDEX.
	DRV_AK4953_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4953_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK4953_MONO_STEREO_MIC	Identifies the Mic input as Mono / Stereo.

Description

This section describes the API functions of the AK4953 Codec Driver library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_AK4953_Initialize Function

Initializes hardware and data for the instance of the AK4953 DAC module.

Implementation: Dynamic

File

drv_ak4953.h

C

```
SYS_MODULE_OBJ DRV_AK4953_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the AK4953 driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized.

Remarks

This routine must be called before any other AK4953 routine is called.

This routine should only be called once during system initialization unless DRV_AK4953_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

DRV_I2S_Initialize must be called before calling this function to initialize the data interface of this CODEC driver. Also DRV_I2C_Initialize must be called before calling this function to initialize the control interface of this CODEC driver.

Example

```
DRV_AK4953_INIT                init;
SYS_MODULE_OBJ                 objectHandle;

init->inUse                          = true;
init->status                         = SYS_STATUS_BUSY;
init->numClients                     = 0;
init->i2sDriverModuleIndex           = ak4953Init->i2sDriverModuleIndex;
init->i2cDriverModuleIndex           = ak4953Init->i2cDriverModuleIndex;
init->samplingRate                   = DRV_AK4953_AUDIO_SAMPLING_RATE;
init->audioDataFormat                = DRV_AK4953_AUDIO_DATA_FORMAT_MACRO;
for(index=0; index < DRV_AK4953_NUMBER_OF_CHANNELS; index++)
{
    init->volume[index] = ak4953Init->volume;
}
init->isInInterruptContext           = false;

init->commandCompleteCallback = (DRV_AK4953_COMMAND_EVENT_HANDLER)0;
init->commandContextData = 0;

init->mclk_multiplier = DRV_AK4953_MCLK_SAMPLE_FREQ_MULTPLIER;


objectHandle = DRV_AK4953_Initialize(DRV_AK4953_0, (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to be initialized
init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

```
    SYS_MODULE_OBJ  DRV_AK4953_Initialize
(
const           SYS_MODULE_INDEX drvIndex,
const           SYS_MODULE_INIT *const init
);
```

DRV_AK4953_Deinitialize Function

Deinitializes the specified instance of the AK4953 driver module.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the AK4953 driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_AK4953_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_AK4953_Initialize
SYS_STATUS          status;


DRV_AK4953_Deinitialize(object);

status = DRV_AK4953_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4953_Initialize routine

Function

void DRV_AK4953_Deinitialize(SYS_MODULE_OBJ object)

DRV_AK4953_Open Function

Opens the specified AK4953 driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_ak4953.h

C

```
DRV_HANDLE DRV_AK4953_Open(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the number of client objects allocated via DRV_AK4953_CLIENTS_NUMBER is insufficient.
- if the client is trying to open the driver but driver has been opened exclusively by another client.
- if the driver hardware instance being opened is not initialized or is invalid.
- if the ioIntent options passed are not relevant to this driver.

Description

This routine opens the specified AK4953 driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options are not relevant to this driver. All the data transfer functions of this driver are non blocking.

AK4953 can be opened with DRV_IO_INTENT_WRITE, or DRV_IO_INTENT_READ or DRV_IO_INTENT_WRITEREAD io_intent option. This decides whether the driver is used for headphone output, or microphone input or both modes simultaneously.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_AK4953_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

Function DRV_AK4953_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_AK4953_Open(DRV_AK4953_INDEX_0, DRV_IO_INTENT_WRITEREAD | DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

DRV_HANDLE DRV_AK4953_Open

(

const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT ioIntent

)

DRV_AK4953_Close Function

Closes an opened-instance of the AK4953 driver.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This routine closes an opened-instance of the AK4953 driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_AK4953_Open before the caller may use the driver again

Remarks

Usually there is no need for the driver client to verify that the Close operation has completed. The driver will abort any ongoing operations when this routine is called.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_AK4953_Open

DRV_AK4953_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4953_Close(DRV_Handle handle)

DRV_AK4953_Tasks Function

Maintains the driver's control and data interface state machine.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal control and data interface state machine and implement its control and data interface implementations. This function should be called from the SYS_Tasks() function.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4953_Initialize

while (true)
{
    DRV_AK4953_Tasks (object);
```

```
    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_AK4953_Initialize)

Function

void DRV_AK4953_Tasks(SYS_MODULE_OBJ object);

DRV_AK4953_CommandEventHandlerSet Function

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_CommandEventHandlerSet(DRV_HANDLE handle, const DRV_AK4953_COMMAND_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

When a client calls DRV_AK4953_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "AK4953 CODEC Specific Client Routines" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the command has completed, it does not need to register a callback.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4953_BUFFER_HANDLE bufferHandle;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

// Client registers an event handler with driver

DRV_AK4953_CommandEventHandlerSet(myAK4953Handle,
                APP_AK4953CommandEventHandler, (uintptr_t)&myAppObj);

DRV_AK4953_DeEmphasisFilterSet(myAK4953Handle, DRV_AK4953_DEEMPHASIS_FILTER_44_1KHZ)

// Event is received when
// the buffer is processed.

void APP_AK4953CommandEventHandler(uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.
```

```
    switch(event)
    {
            // Last Submitted command is completed.
            // Perform further processing here
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4953_CommandEventHandlerSet

(

DRV_HANDLE handle,

const DRV_AK4953_COMMAND_EVENT_HANDLER eventHandler,

const uintptr_t contextHandle

)

DRV_AK4953_BufferEventHandlerSet Function

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

File

drv_ak4953.h

C

```
void DRV_AK4953_BufferEventHandlerSet(DRV_HANDLE handle, const DRV_AK4953_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls DRV_AK4953_BufferAddRead function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4953_BUFFER_HANDLE bufferHandle;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

// Client registers an event handler with driver

DRV_AK4953_BufferEventHandlerSet(myAK4953Handle,
```

```
                APP_AK4953BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4953_BufferAddRead(myAK4953handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4953_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4953BufferEventHandler(DRV_AK4953_BUFFER_EVENT event,
        DRV_AK4953_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4953_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4953_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK4953_BufferEventHandlerSet
(
DRV_HANDLE handle,
const DRV_AK4953_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t contextHandle
)

DRV_AK4953_SamplingRateSet Function

This function sets the sampling rate of the media stream.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate);
```

Returns

None.

Description

This function sets the media sampling rate for the client handle.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

DRV_AK4953_SamplingRateSet(myAK4953Handle, 48000);  //Sets 48000 media sampling rate
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4953_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate)

b) Status Functions

DRV_AK4953_SamplingRateGet Function

This function gets the sampling rate set on the DAC AK4953.

Implementation: Dynamic

File

drv_ak4953.h

C

```
uint32_t DRV_AK4953_SamplingRateGet(DRV_HANDLE handle);
```

Returns

None.

Description

This function gets the sampling rate set on the DAC AK4953.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
uint32_t baudRate;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

baudRate = DRV_AK4953_SamplingRateGet(myAK4953Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_AK4953_SamplingRateGet(DRV_HANDLE handle)

DRV_AK4953_Status Function

Gets the current status of the AK4953 driver module.

Implementation: Dynamic

File

drv_ak4953.h

C

```
SYS_STATUS DRV_AK4953_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized

SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed

SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed

SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This routine provides the current status of the AK4953 driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_AK4953_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK4953_Initialize
SYS_STATUS          AK4953Status;

AK4953Status = DRV_AK4953_Status(object);
if (SYS_STATUS_READY == AK4953Status)
{
    // This means the driver can be opened using the
    // DRV_AK4953_Open() function.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK4953_Initialize routine

Function

SYS_STATUS DRV_AK4953_Status(SYS_MODULE_OBJ object)

DRV_AK4953_VersionGet Function

This function returns the version of AK4953 driver.

Implementation: Dynamic

File

drv_ak4953.h

C

```
uint32_t DRV_AK4953_VersionGet();
```

Returns

returns the version of AK4953 driver.

Description

The version number returned from the DRV_AK4953_VersionGet function is an unsigned integer in the following decimal format. * 10000 + * 100 + Where the numbers are represented in decimal and the meaning is the same as above. Note that there is no numerical representation of release type.

Remarks

None.

Preconditions

None.

Example 1

For version "0.03a", return: 0 * 10000 + 3 * 100 + 0 For version "1.00", return: 1 * 100000 + 0 * 100 + 0

Example 2

```
uint32_t AK4953version;
AK4953version = DRV_AK4953_VersionGet();
```

Function

uint32_t DRV_AK4953_VersionGet(void)

DRV_AK4953_VersionStrGet Function

This function returns the version of AK4953 driver in string format.

Implementation: Dynamic

File

drv_ak4953.h

C

```
int8_t* DRV_AK4953_VersionStrGet();
```

Returns

returns a string containing the version of AK4953 driver.

Description

The DRV_AK4953_VersionStrGet function returns a string in the format: ".[.][]" Where: is the AK4953 driver's version number. is the AK4953 driver's version number. is an optional "patch" or "dot" release number (which is not included in the string if it equals "00"). is an optional release type ("a" for alpha, "b" for beta ? not the entire word spelled out) that is not included if the release is a production version (I.e. Not an alpha or beta).

The String does not contain any spaces.

Remarks

None.

Preconditions

None.

Example 1

"0.03a" "1.00"

Example 2

```
int8_t *AK4953string;
AK4953string = DRV_AK4953_VersionStrGet();
```

Function

int8_t* DRV_AK4953_VersionStrGet(void)

DRV_AK4953_VolumeGet Function

This function gets the volume for AK4953 CODEC.

Implementation: Dynamic

File

drv_ak4953.h

C

```
uint8_t DRV_AK4953_VolumeGet(DRV_HANDLE handle, DRV_AK4953_CHANNEL chan);
```

Returns

None.

Description

This functions gets the current volume programmed to the CODEC AK4953.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
uint8_t volume;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

  volume = DRV_AK4953_VolumeGet(myAK4953Handle,DRV_AK4953_CHANNEL_LEFT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
chan	Audio channel volume to be set

Function

uint8_t DRV_AK4953_VolumeGet(DRV_HANDLE handle, DRV_AK4953_CHANNEL chan)

c) Other Functions

DRV_AK4953_BufferAddWrite Function

Schedule a non-blocking driver write operation.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_BufferAddWrite(const DRV_HANDLE handle, DRV_AK4953_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4953_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4953_BUFFER_HANDLE_INVALID

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4953_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4953_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4953 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4953 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 device instance and the DRV_AK4953_Status must have returned SYS_STATUS_READY.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE must have been specified in the DRV_AK4953_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK4953_BUFFER_HANDLE bufferHandle;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

// Client registers an event handler with driver

DRV_AK4953_BufferEventHandlerSet(myAK4953Handle,
                APP_AK4953BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4953_BufferAddWrite(myAK4953handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK4953_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4953BufferEventHandler(DRV_AK4953_BUFFER_EVENT event,
        DRV_AK4953_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4953_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4953_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
```

```
            break;
        }
    }
```

Parameters

Parameters	Description
handle	Handle of the AK4953 instance as return by the DRV_AK4953_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK4953_BufferAddWrite

(

const DRV_HANDLE handle,

DRV_AK4953_BUFFER_HANDLE *bufferHandle,

void *buffer, size_t size

)

DRV_AK4953_BufferAddWriteRead Function

Schedule a non-blocking driver write-read operation.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_BufferAddWriteRead(const DRV_HANDLE handle, DRV_AK4953_BUFFER_HANDLE * bufferHandle, void *
transmitBuffer, void * receiveBuffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4953_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write-read operation. The function returns with a valid buffer handle in the bufferHandle argument if the write-read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4953_BUFFER_EVENT_COMPLETE:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only or write only
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4953_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4953_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4953 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4953 driver instance. It should not otherwise be called directly in an ISR.

This function is useful when there is valid read expected for every AK4953 write. The transmit and receive size must be same.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 device instance and the DRV_AK4953_Status must have returned SYS_STATUS_READY.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READWRITE must have been specified in the DRV_AK4953_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybufferTx[MY_BUFFER_SIZE];
uint8_t mybufferRx[MY_BUFFER_SIZE];
DRV_AK4953_BUFFER_HANDLE bufferHandle;

// myak4953Handle is the handle returned
// by the DRV_AK4953_Open function.

// Client registers an event handler with driver

DRV_AK4953_BufferEventHandlerSet(myak4953Handle,
                APP_AK4953BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK4953_BufferAddWriteRead(myak4953handle, &bufferHandle,
                                    mybufferTx,mybufferRx,MY_BUFFER_SIZE);

if(DRV_AK4953_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK4953BufferEventHandler(DRV_AK4953_BUFFER_EVENT event,
        DRV_AK4953_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK4953_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK4953_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the AK4953 instance as returned by the DRV_AK4953_Open function
bufferHandle	Pointer to an argument that will contain the return buffer handle
transmitBuffer	The buffer where the transmit data will be stored
receiveBuffer	The buffer where the received data will be stored
size	Buffer size in bytes

Function

void DRV_AK4953_BufferAddWriteRead
(
const DRV_HANDLE handle,
DRV_AK4953_BUFFER_HANDLE *bufferHandle,
void *transmitBuffer,
void *receiveBuffer,
size_t size
)

DRV_AK4953_MuteOff Function

This function disables AK4953 output for soft mute.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_MuteOff(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables AK4953 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

    DRV_AK4953_MuteOff(myAK4953Handle); //AK4953 output soft mute disabled
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4953_MuteOff(DRV_HANDLE handle)

DRV_AK4953_MuteOn Function

This function allows AK4953 output for soft mute on.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_MuteOn(DRV_HANDLE handle);
```

Returns

None.

Description

This function Enables AK4953 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.

DRV_AK4953_MuteOn(myAK4953Handle);  //AK4953 output soft muted
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4953_MuteOn(DRV_HANDLE handle);

DRV_AK4953_VolumeSet Function

This function sets the volume for AK4953 CODEC.

Implementation: Dynamic

File

drv_ak4953.h

C

```
void DRV_AK4953_VolumeSet(DRV_HANDLE handle, DRV_AK4953_CHANNEL channel, uint8_t volume);
```

Returns

None.

Description

This functions sets the volume value from 0-255. The codec has DAC value to volume range mapping as :- 00 H : +12dB FF H : -115dB In order to make the volume value to dB mapping monotonically increasing from 00 to FF, re-mapping is introduced which reverses the volume value to dB mapping as well as normalizes the volume range to a more audible dB range. The current driver implementation assumes that all dB values under -60 dB are inaudible to the human ear. Re-Mapped values 00 H : -60 dB FF H : +12 dB

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK4953Handle is the handle returned
// by the DRV_AK4953_Open function.
```

```
DRV_AK4953_VolumeSet(myAK4953Handle, DRV_AK4953_CHANNEL_LEFT, 120);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
chan	Audio channel volume to be set
volume	volume value specified in the range 0-255 (0x00 to 0xFF)

Function

void DRV_AK4953_VolumeSet(DRV_HANDLE handle, DRV_AK4953_CHANNEL channel, uint8_t volume);

DRV_AK4953_BufferAddRead Function

Schedule a non-blocking driver read operation.

File

drv_ak4953.h

C

```
void DRV_AK4953_BufferAddRead(const DRV_HANDLE handle, DRV_AK4953_BUFFER_HANDLE * bufferHandle, void * buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK4953_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking read operation. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance receive queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK4953_BUFFER_HANDLE_INVALID

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK4953_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK4953_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK4953 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK4953 driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 device instance and the DRV_AK4953_Status must have returned SYS_STATUS_READY.

DRV_AK4953_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ must have been specified in the DRV_AK4953_Open call.

Parameters

Parameters	Description
handle	Handle of the AK4953 instance as return by the DRV_AK4953_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK4953_BufferAddRead

(

const DRV_HANDLE handle,

DRV_AK4953_BUFFER_HANDLE *bufferHandle,
void *buffer, size_t size
)

DRV_AK4953_IntExtMicSet Function

This function sets up the codec for the internal or the external microphone use.

File

drv_ak4953.h

C

```
void DRV_AK4953_IntExtMicSet(DRV_HANDLE handle, DRV_AK4953_INT_EXT_MIC micInput);
```

Returns

None

Description

This function sets up the codec for the internal or the external microphone use.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.
DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
micInput	Internal vs External mic input

Function

void DRV_AK4953_IntExtMicSet(DRV_HANDLE handle);

DRV_AK4953_MonoStereoMicSet Function

This function sets up the codec for the Mono or Stereo microphone mode.

File

drv_ak4953.h

C

```
void DRV_AK4953_MonoStereoMicSet(DRV_HANDLE handle, DRV_AK4953_MONO_STEREO_MIC mono_stereo_mic);
```

Returns

None

Description

This function sets up the codec for the Mono or Stereo microphone mode.

Remarks

None.

Preconditions

The DRV_AK4953_Initialize routine must have been called for the specified AK4953 driver instance.
DRV_AK4953_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_AK4953_MonoStereoMicSet(DRV_HANDLE handle);

d) Data Types and Constants

DRV_AK4953_AUDIO_DATA_FORMAT Enumeration

Identifies the Serial Audio data interface format.

File

drv_ak4953.h

C

```
typedef enum {
  DRV_AK4953_AUDIO_DATA_FORMAT_24BIT_MSB_SDTO_24BIT_LSB_SDTI = 0,
  DRV_AK4953_AUDIO_DATA_FORMAT_24BIT_MSB_SDTO_16BIT_LSB_SDTI,
  DRV_AK4953_AUDIO_DATA_FORMAT_24BIT_MSB_SDTO_24BIT_MSB_SDTI,
  DRV_AK4953_AUDIO_DATA_FORMAT_I2S
} DRV_AK4953_AUDIO_DATA_FORMAT;
```

Description

AK4953 Audio data format

This enumeration identifies Serial Audio data interface format.

DRV_AK4953_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_ak4953.h

C

```
typedef enum {
  DRV_AK4953_BUFFER_EVENT_COMPLETE,
  DRV_AK4953_BUFFER_EVENT_ERROR,
  DRV_AK4953_BUFFER_EVENT_ABORT
} DRV_AK4953_BUFFER_EVENT;
```

Members

Members	Description
DRV_AK4953_BUFFER_EVENT_COMPLETE	Data was transferred successfully.
DRV_AK4953_BUFFER_EVENT_ERROR	Error while processing the request
DRV_AK4953_BUFFER_EVENT_ABORT	Data transfer aborted (Applicable in DMA mode)

Description

AK4953 Driver Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_AK4953_BufferAddWrite() function.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_AK4953_BufferEventHandlerSet function when a buffer transfer request is completed.

DRV_AK4953_BUFFER_EVENT_HANDLER Type

Pointer to a AK4953 Driver Buffer Event handler function

File

drv_ak4953.h

C

```
typedef void (* DRV_AK4953_BUFFER_EVENT_HANDLER)(DRV_AK4953_BUFFER_EVENT event, DRV_AK4953_BUFFER_HANDLE
bufferHandle, uintptr_t contextHandle);
```

Returns

None.

Description

AK4953 Driver Buffer Event Handler Function

This data type defines the required function signature for the AK4953 driver buffer event handling callback function. A client must register a pointer to a buffer event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_AK4953_BUFFER_EVENT_COMPLETE, this means that the data was transferred successfully.

If the event is DRV_AK4953_BUFFER_EVENT_ERROR, this means that the data was not transferred successfully. The bufferHandle parameter contains the buffer handle of the buffer that failed. The DRV_AK4953_BufferProcessedSizeGet() function can be called to find out how many bytes were processed.

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4953_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The buffer handle in bufferHandle expires after this event handler exits. In that the buffer object that was allocated is deallocated by the driver after the event handler exits.

The event handler function executes in the data driver (I2S) peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

DRV_AK4953_BufferAddWrite function can be called in the event handler to add a buffer to the driver queue.

Example

```
void APP_MyBufferEventHandler( DRV_AK4953_BUFFER_EVENT event,
                               DRV_AK4953_BUFFER_HANDLE bufferHandle,
                               uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_AK4953_BUFFER_EVENT_COMPLETE:
            // Handle the completed buffer.
        break;

        case DRV_AK4953_BUFFER_EVENT_ERROR:
        default:
            // Handle error.
        break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the event relates
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4953_BUFFER_HANDLE Type

Handle identifying a write buffer passed to the driver.

File

drv_ak4953.h

C

```
typedef uintptr_t DRV_AK4953_BUFFER_HANDLE;
```

Description

AK4953 Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_AK4953_BufferAddWrite() function. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None

DRV_AK4953_COMMAND_EVENT_HANDLER Type

Pointer to a AK4953 Driver Command Event Handler Function

File

drv_ak4953.h

C

```
typedef void (* DRV_AK4953_COMMAND_EVENT_HANDLER)(uintptr_t contextHandle);
```

Returns

None.

Description

AK4953 Driver Command Event Handler Function

This data type defines the required function signature for the AK4953 driver command event handling callback function.

A command is a control instruction to the AK4953 CODEC. Example Mute ON/OFF, Zero Detect Enable/Disable etc.

A client must register a pointer to a command event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive command related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

The occurrence of this call back means that the last control command was transferred successfully.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK4953_CommandEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The event handler function executes in the control data driver interrupt context. It is recommended of the application to not perform process intensive or blocking operations with in this function.

Example

```
void APP_AK4953CommandEventHandler( uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    // Last Submitted command is completed.
    // Perform further processing here
}
```

Parameters

Parameters	Description
context	Value identifying the context of the application that registered the event handling function.

DRV_AK4953_DIGITAL_BLOCK_CONTROL Enumeration

Identifies Bass-Boost Control function

File

drv_ak4953.h

C

```
typedef enum {
  DRV_AK4953_RECORDING_MODE,
  DRV_AK4953_PLAYBACK_MODE,
  DRV_AK4953_RECORDING_PLAYBACK_2_MODE,
  DRV_AK4953_LOOPBACK_MODE
} DRV_AK4953_DIGITAL_BLOCK_CONTROL;
```

Members

Members	Description
DRV_AK4953_RECORDING_MODE	This is the default setting
DRV_AK4953_PLAYBACK_MODE	Min control
DRV_AK4953_RECORDING_PLAYBACK_2_MODE	Medium control
DRV_AK4953_LOOPBACK_MODE	Maximum control

Description

AK4953 Bass-Boost Control

This enumeration identifies the settings for Bass-Boost Control function.

Remarks

None.

DRV_AK4953_INIT Structure

Defines the data required to initialize or reinitialize the AK4953 driver

File

drv_ak4953.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX i2sDriverModuleIndex;
  SYS_MODULE_INDEX i2cDriverModuleIndex;
  uint32_t samplingRate;
  uint8_t volume;
  DRV_AK4953_AUDIO_DATA_FORMAT audioDataFormat;
} DRV_AK4953_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX i2sDriverModuleIndex;	Identifies data module(I2S) driver ID for data interface of CODEC
SYS_MODULE_INDEX i2cDriverModuleIndex;	Identifies data module(I2C) driver ID for control interface of CODEC
uint32_t samplingRate;	Sampling rate
uint8_t volume;	Volume
DRV_AK4953_AUDIO_DATA_FORMAT audioDataFormat;	Identifies the Audio data format

Description

AK4953 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the AK4953 CODEC driver.

Remarks

None.

_DRV_AK4953_H Macro

File

drv_ak4953.h

C

```
#define _DRV_AK4953_H
```

Description

Include files.

DRV_AK4953_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_ak4953.h

C

```
#define DRV_AK4953_BUFFER_HANDLE_INVALID ((DRV_AK4953_BUFFER_HANDLE)(-1))
```

Description

AK4953 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_AK4953_BufferAddWrite() function if the buffer add request was not successful.

Remarks

None

DRV_AK4953_COUNT Macro

Number of valid AK4953 driver indices

File

drv_ak4953.h

C

```
#define DRV_AK4953_COUNT
```

Description

AK4953 Driver Module Count

This constant identifies the maximum number of AK4953 Driver instances that should be defined by the application. Defining more instances than this constant will waste RAM memory space.

This constant can also be used by the application to identify the number of AK4953 instances on this microcontroller.

Remarks

This value is part-specific.

DRV_AK4953_INDEX_0 Macro

AK4953 driver index definitions

File

drv_ak4953.h

C

```
#define DRV_AK4953_INDEX_0 0
```

Description

Driver AK4953 Module Index

These constants provide AK4953 driver index definition.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_AK4953_Initialize and DRV_AK4953_Open routines to identify the driver instance in use.

DRV_AK4953_INDEX_1 Macro

File

drv_ak4953.h

C

```
#define DRV_AK4953_INDEX_1 1
```

Description

This is macro DRV_AK4953_INDEX_1.

DRV_AK4953_INDEX_2 Macro

File

drv_ak4953.h

C

```
#define DRV_AK4953_INDEX_2 2
```

Description

This is macro DRV_AK4953_INDEX_2.

DRV_AK4953_INDEX_3 Macro

File

drv_ak4953.h

C

```
#define DRV_AK4953_INDEX_3 3
```

Description

This is macro DRV_AK4953_INDEX_3.

DRV_AK4953_INDEX_4 Macro

File

drv_ak4953.h

C

```
#define DRV_AK4953_INDEX_4 4
```

Description

This is macro DRV_AK4953_INDEX_4.

DRV_AK4953_INDEX_5 Macro

File

drv_ak4953.h

C

```
#define DRV_AK4953_INDEX_5 5
```

Description

This is macro DRV_AK4953_INDEX_5.

DRV_I2C_INDEX Macro

File

drv_ak7755.h

C

```
#define DRV_I2C_INDEX DRV_AK7755_I2C_INSTANCES_NUMBER
```

Description

This is macro DRV_I2C_INDEX.

DRV_AK4953_CHANNEL Enumeration

Identifies Left/Right Audio channel

File

drv_ak4953.h

C

```
typedef enum {
  DRV_AK4953_CHANNEL_LEFT,
  DRV_AK4953_CHANNEL_RIGHT,
  DRV_AK4953_CHANNEL_LEFT_RIGHT,
  DRV_AK4953_NUMBER_OF_CHANNELS
} DRV_AK4953_CHANNEL;
```

Description

AK4953 Audio Channel

This enumeration identifies Left/Right Audio channel

Remarks

None.

DRV_AK4953_INT_EXT_MIC Enumeration

Identifies the Mic input source.

File

drv_ak4953.h

C

```
typedef enum {
  INT_MIC,
  EXT_MIC
} DRV_AK4953_INT_EXT_MIC;
```

Description

AK4953 Mic Internal / External Input

This enumeration identifies the Mic input source.

DRV_AK4953_MONO_STEREO_MIC Enumeration

Identifies the Mic input as Mono / Stereo.

File

drv_ak4953.h

C

```
typedef enum {
```

```
  ALL_ZEROS,
  MONO_RIGHT_CHANNEL,
  MONO_LEFT_CHANNEL,
  STEREO
} DRV_AK4953_MONO_STEREO_MIC;
```

Description

AK4953 Mic Mono / Stereo Input

This enumeration identifies the Mic input as Mono / Stereo.

Files

Files

Name	Description
drv_ak4953.h	AK4953 CODEC Driver Interface header file
drv_ak4953_config_template.h	AK4953 Codec Driver Configuration Template.

Description

This section lists the source and header files used by the AK4953Codec Driver Library.

drv_ak4953.h

AK4953 CODEC Driver Interface header file

Enumerations

	Name	Description
	DRV_AK4953_AUDIO_DATA_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK4953_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK4953_CHANNEL	Identifies Left/Right Audio channel
	DRV_AK4953_DIGITAL_BLOCK_CONTROL	Identifies Bass-Boost Control function
	DRV_AK4953_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK4953_MONO_STEREO_MIC	Identifies the Mic input as Mono / Stereo.

Functions

	Name	Description
	DRV_AK4953_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_AK4953_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic
	DRV_AK4953_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_AK4953_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_AK4953_Close	Closes an opened-instance of the AK4953 driver. **Implementation:** Dynamic
	DRV_AK4953_CommandEventHandlerSet	This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished. **Implementation:** Dynamic
	DRV_AK4953_Deinitialize	Deinitializes the specified instance of the AK4953 driver module. **Implementation:** Dynamic
	DRV_AK4953_Initialize	Initializes hardware and data for the instance of the AK4953 DAC module. **Implementation:** Dynamic
	DRV_AK4953_IntExtMicSet	This function sets up the codec for the internal or the external microphone use.
	DRV_AK4953_MonoStereoMicSet	This function sets up the codec for the Mono or Stereo microphone mode.
	DRV_AK4953_MuteOff	This function disables AK4953 output for soft mute. **Implementation:** Dynamic
	DRV_AK4953_MuteOn	This function allows AK4953 output for soft mute on. **Implementation:** Dynamic
	DRV_AK4953_Open	Opens the specified AK4953 driver instance and returns a handle to it. **Implementation:** Dynamic

	DRV_AK4953_SamplingRateGet	This function gets the sampling rate set on the DAC AK4953. **Implementation:** Dynamic
	DRV_AK4953_SamplingRateSet	This function sets the sampling rate of the media stream. **Implementation:** Dynamic
	DRV_AK4953_Status	Gets the current status of the AK4953 driver module. **Implementation:** Dynamic
	DRV_AK4953_Tasks	Maintains the driver's control and data interface state machine. **Implementation:** Dynamic
	DRV_AK4953_VersionGet	This function returns the version of AK4953 driver. **Implementation:** Dynamic
	DRV_AK4953_VersionStrGet	This function returns the version of AK4953 driver in string format. **Implementation:** Dynamic
	DRV_AK4953_VolumeGet	This function gets the volume for AK4953 CODEC. **Implementation:** Dynamic
	DRV_AK4953_VolumeSet	This function sets the volume for AK4953 CODEC. **Implementation:** Dynamic

Macros

	Name	Description
	_DRV_AK4953_H	Include files.
	DRV_AK4953_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK4953_COUNT	Number of valid AK4953 driver indices
	DRV_AK4953_INDEX_0	AK4953 driver index definitions
	DRV_AK4953_INDEX_1	This is macro DRV_AK4953_INDEX_1.
	DRV_AK4953_INDEX_2	This is macro DRV_AK4953_INDEX_2.
	DRV_AK4953_INDEX_3	This is macro DRV_AK4953_INDEX_3.
	DRV_AK4953_INDEX_4	This is macro DRV_AK4953_INDEX_4.
	DRV_AK4953_INDEX_5	This is macro DRV_AK4953_INDEX_5.

Structures

	Name	Description
	DRV_AK4953_INIT	Defines the data required to initialize or reinitialize the AK4953 driver

Types

	Name	Description
	DRV_AK4953_BUFFER_EVENT_HANDLER	Pointer to a AK4953 Driver Buffer Event handler function
	DRV_AK4953_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK4953_COMMAND_EVENT_HANDLER	Pointer to a AK4953 Driver Command Event Handler Function

Description

AK4953 CODEC Driver Interface

The AK4953 CODEC device driver interface provides a simple interface to manage the AK4953 106dB 192kHz 24-Bit DAC that can be interfaced Microchip Microcontroller. This file provides the interface definition for the AK4953 CODEC device driver.

File Name

drv_AK4953.h

Company

Microchip Technology Inc.

drv_ak4953_config_template.h

AK4953 Codec Driver Configuration Template.

Macros

	Name	Description
	DRV_AK4953_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency

	DRV_AK4953_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK4953_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to codec.
	DRV_AK4953_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK4953_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK Ratio to Generate Audio Stream for specified sampling frequency
	DRV_AK4953_MCLK_SOURCE	Indicate the input clock frequency to generate the MCLK to codec.
	DRV_AK4953_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.

Description

AK4953 Codec Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_ak4953_config_template.h

Company

Microchip Technology Inc.

AK7755 Codec Driver Library

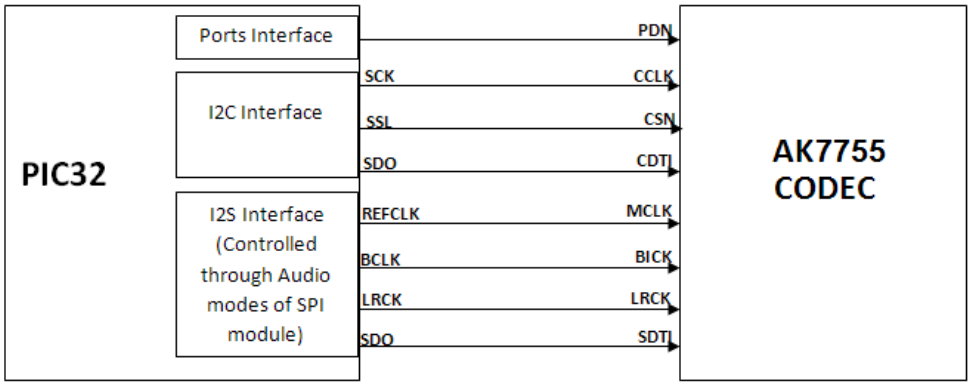
This topic describes the AK7755 Codec Driver Library.

Introduction

This library provides an interface to manage the AK7755 Codec that is serially interfaced to a Microchip microcontroller for providing Audio Solutions.

Description

The AK7755 module is 16/20/24-bit Audio Codec from Asahi Kasei Microdevices Corporation. The AK7755 can be interfaced to Microchip microcontrollers through I2C and I2S serial interfaces. The I2C interface is used for control command transfer. The I2S interface is used for Audio data output. A typical interface of the AK7755 Codec to a Microchip PIC32 device is provided in the following diagram:



Features

The AK7755 Codec supports the following features:

- Two Digital Interfaces (I/F1, I/F2):
- 4-channel/6-channel Digital Signal Input Port: MSB justified 24-bit, LSB justified 24/20/16-bit, I2S
- Short/Long Frame
- 24-bit linear, 8-bit A-law, 8-bit µ-law
- TDM 256 fs (8-channel) MSB Justified and I2S Formats
- SoftMute: On and Off
- Stereo 24-bit ADC:
- Sampling Frequency: fs = 8 kHz ~96 kHz
- ADC Characteristics S/(N+D): 91 dB, DR, S/N: 102 dB
- Two-Channel Analog Input Selector (Differential, Single-ended Input)
- Channel Independent Microphone Analog Gain Amplifier (0 ~18 dB (2 dB Step), 18 ~36 dB (3 dB Step))
- Analog DRC (Dynamic Range Control)

- Channel Independent Digital Volume (24 ~-103 dB, 0.5 dB Step Mute)
- Digital HPF for DC Offset Cancelling
- Mono 24-bit ADC:
- Sampling Frequency: 8 kHz ~ 96 kHz
- ADC Characteristics S/(N+D): 90 dB; DR, S/N: 100 dB
- Line Amplifier: 21 dB ~ -21 dB, 3 dB Step
- Digital Volume (24 dB ~ -103 dB, 0.5 dB step, Mute)
- Digital HPF for DC Offset Cancelling
- Stereo 24-bit DAC:
- Sampling Frequency: fs = 8 kHz ~ 96 kHz
- Digital Volume (12 dB ~ -115 dB, 0.5 step, Mute)
- Digital De-emphasis Filter (tc = 50/15 µs, fs = 32 kHz, 44.1 kHz, 48 kHz)
- Master Clock: 2560 fs (internally generated by PLL from 32, 48, 64, 128, 256 and 384 fs clock)

Using the Library

This topic describes the basic architecture of the AK7755 Codec Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_AK7755.h`

The interface to the AK7755 Codec Driver library is defined in the `drv_AK7755.h` header file. Any C language source (`.c`) file that uses the AK7755 Codec Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

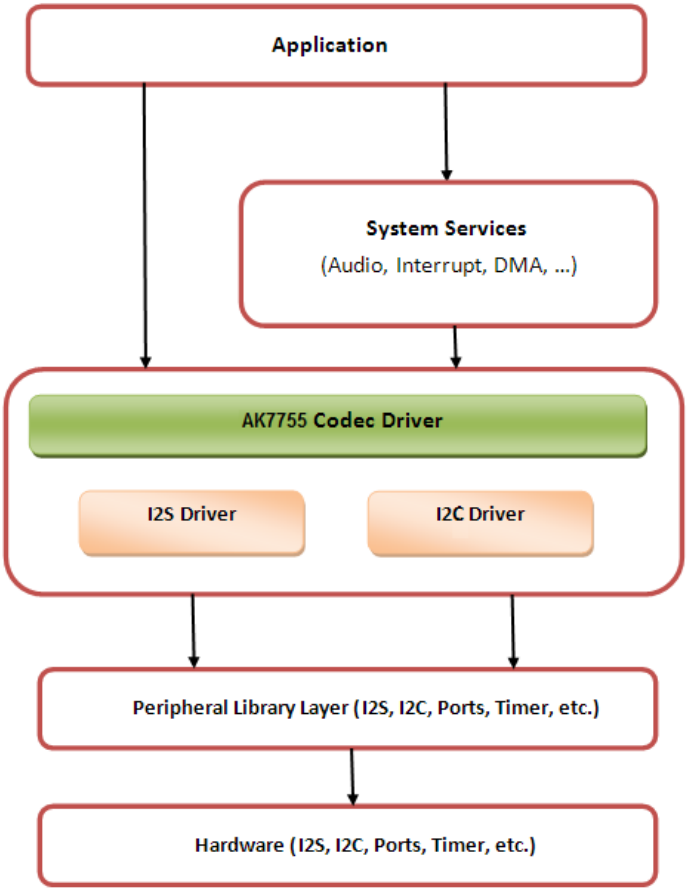
Abstraction Model

This library provides a low-level abstraction of the AK7755 Codec Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The abstraction model shown in the following diagram depicts how the AK7755 Codec Driver is positioned in the MPLAB Harmony framework. The AK7755 Codec Driver uses the SPI and I2S drivers for control and audio data transfers to the AK7755 module.

AK7755 Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The AK7755 Codec Driver Library provides an API interface to transfer control commands and digital audio data to the serially interfaced AK7755 DAC module. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the AK7755 Codec Driver Library.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Status Functions	Provides status functions.
Other Functions	Provides driver specific miscellaneous functions such as sampling rate setting, control command functions, etc.
Data Types and Constants	These data types and constants are required while interacting and setting up the AK7755 Codec Driver Library.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

System Access

This topic describes system initialization, implementations, and includes a system access code example.

Description

System Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the AK7755 module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_AK7755_INIT or by using Initialization Overrides) that are supported by the specific AK7755 device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- I2C driver module index. The module index should be same as the one used in initializing the I2C Driver.
- I2S driver module index. The module index should be same as the one used in initializing the I2S Driver.
- Sampling rate
- Audio data format. The audio data format should match with the audio data format settings done in I2S driver initialization
- Power down pin port initialization
- Queue size for the audio data transmit buffer

The DRV_AK7755_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_ AK7755_Deinitialize, DRV_ AK7755_Status and DRV_I2S_Tasks.

Implementations

The AK7755 Codec Driver can has the following implementation:

Description	MPLAB Harmony Components
Dedicated hardware for control (I2C) and data (I2S) interface.	Standard MPLAB Harmony drivers for I2C and I2S interfaces.

Example:

```
DRV_AK7755_INIT drvak7755Codec0InitData =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .i2sDriverModuleIndex = DRV_AK7755_I2S_DRIVER_MODULE_INDEX_IDX0,
    .i2cDriverModuleIndex = DRV_AK7755_I2C_DRIVER_MODULE_INDEX_IDX0,
    .volume = DRV_AK7755_VOLUME,
    .queueSizeTransmit = DRV_AK7755_TRANSMIT_QUEUE_SIZE,
};

// Initialize the I2C driver
   DRV_I2C0_Initialize();

// Initialize the I2S driver. The I2S module index should be same as the one used in initializing
// the I2S driver.
   sysObj.drvI2S0 = DRV_I2S_Initialize(DRV_I2S_INDEX_0, (SYS_MODULE_INIT *)&drvI2S0InitData);

// Initialize the Codec driver
   sysObj.drvak7755Codec0 = DRV_AK7755_Initialize(DRV_AK7755_INDEX_0, (SYS_MODULE_INIT
*)&drvak7755Codec0InitData);

if (SYS_MODULE_OBJ_INVALID == AK7755DevObject)
{
// Handle error
}
```

Task Routine

The DRV_AK7755_Tasks will be called from the System Task Service.

Client Access

For the application to start using an instance of the module, it must call the DRV_AK7755_Open function. The DRV_AK7755_Open provides a driver handle to the AK7755 Codec Driver instance for operations. If the driver is deinitialized using the function DRV_AK7755_Deinitialize, the application must call the DRV_AK7755_Open function again to set up the instance of the driver.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

Client Operations

This topic provides information on client operations and includes a control command and audio buffered data operation flow diagram.

Description

Client operations provide the API interface for control command and audio data transfer to the AK7755 Codec.

The following AK7755 Codec specific control command functions are provided:

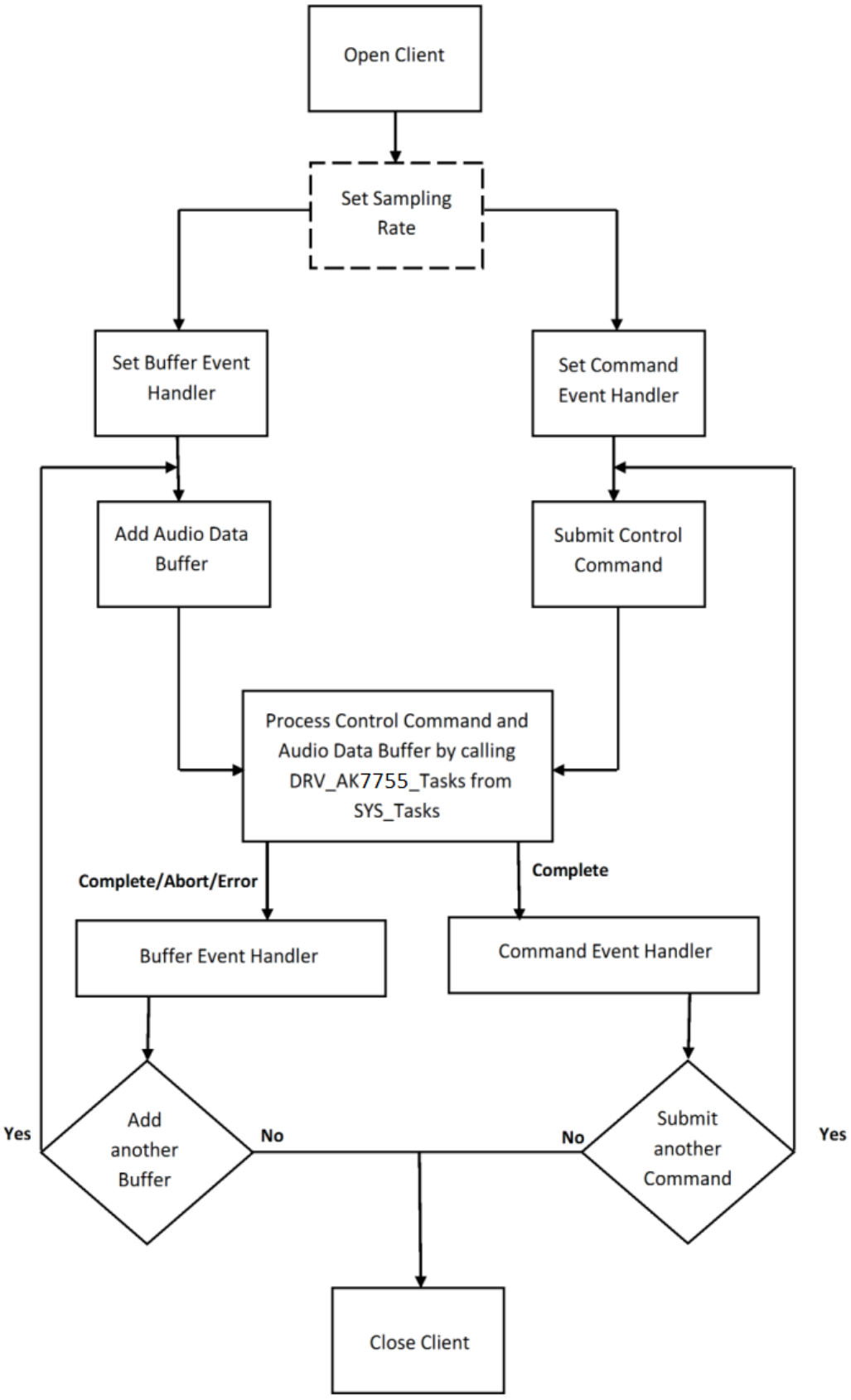
- DRV_AK7755_SamplingRateSet
- DRV_AK7755_SamplingRateGet
- DRV_AK7755_VolumeSet
- DRV_AK7755_VolumeGet
- DRV_AK7755_MuteOn
- DRV_AK7755_MuteOff
- DRV_AK7755_IntExtMicSet
- DRV_AK7755_MonoStereoMicSet

These functions schedule a non-blocking control command transfer operation. These functions submit the control command request to the AK7755 Codec. These functions submit the control command request to I2C Driver transmit queue, the request is processed immediately if it is the first request, or processed when the previous request is complete.

DRV_AK7755_BufferAddWrite, DRV_AK7755_BufferAddRead, and DRV_AK7755_BufferAddWriteRead are buffered data operation functions.

These functions schedule non-blocking audio data transfer operations. These functions add the request to I2S Driver transmit or receive buffer queue depends on the request type, and are executed immediately if it is the first buffer, or executed later when the previous buffer is complete. The driver notifies the client with DRV_AK7755_BUFFER_EVENT_COMPLETE, DRV_AK7755_BUFFER_EVENT_ERROR, or DRV_AK7755_BUFFER_EVENT_ABORT events.

The following diagram illustrates the control commands and audio buffered data operations.



Note: It is not necessary to close and reopen the client between multiple transfers.

Configuring the Library

Macros

	Name	Description
	DRV_AK7755_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.
	DRV_AK7755_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK7755_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to the codec.
	DRV_AK7755_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK7755_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.
	DRV_AK7755_MCLK_SOURCE	Indicates the input clock frequency to generate the MCLK to the codec.

Description

The configuration of the AK7755 Codec Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the AK7755 Codec Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the AK7755 Codec Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_AK7755_BCLK_BIT_CLK_DIVISOR Macro

Sets up the BCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.

File

drv_ak7755_config_template.h

C

```
#define DRV_AK7755_BCLK_BIT_CLK_DIVISOR
```

Description

AK7755 BCLK to LRCK Ratio to Generate Audio Stream

This macro sets up the BCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.

The following BCLK to LRCK ratios are supported:

- 16-bit data 16-bit channel: 32 fs; therefore, the divisor would be 8
- 16-bit data 32-bit channel: 64 fs; therefore, the divisor would be 4

Remarks

None.

DRV_AK7755_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_ak7755_config_template.h

C

```
#define DRV_AK7755_CLIENTS_NUMBER DRV_AK7755_INSTANCES_NUMBER
```

Description

AK7755 Client Count Configuration

This macro sets up the maximum number of clients that can be connected to any hardware instance. Typically only one client could be connected to one hardware instance. This value represents the total number of clients to be supported across all hardware instances. Therefore, if there are five AK7755 hardware interfaces, this number will be 5.

Remarks

None.

DRV_AK7755_INPUT_REFCLOCK Macro

Identifies the input REFCLOCK source to generate the MCLK to the codec.

File

drv_ak7755_config_template.h

C

```
#define DRV_AK7755_INPUT_REFCLOCK
```

Description

AK7755 Input reference clock

This macro identifies the input REFCLOCK source to generate the MCLK to the codec.

Remarks

None.

DRV_AK7755_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_ak7755_config_template.h

C

```
#define DRV_AK7755_INSTANCES_NUMBER
```

Description

AK7755 driver objects configuration

This macro sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of AK7755 Codec modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, the driver will be built statically.

Remarks

None.

DRV_AK7755_MCLK_SAMPLE_FREQ_MULTPLIER Macro

Sets up the MCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.

File

drv_ak7755_config_template.h

C

```
#define DRV_AK7755_MCLK_SAMPLE_FREQ_MULTPLIER
```

Description

AK7755 MCLK to LRCK Ratio to Generate Audio Stream

This macro sets up the MCLK to LRCK ratio to generate the audio stream for the specified I2S sampling frequency.

The supported MCLK to sampling frequency ratios are as follows:

- 256 fs
- 384 fs
- 512 fs
- 768 fs
- 1152 fs

Remarks

None.

DRV_AK7755_MCLK_SOURCE Macro

Indicates the input clock frequency to generate the MCLK to the codec.

File

drv_ak7755_config_template.h

C

```
#define DRV_AK7755_MCLK_SOURCE
```

Description

AK7755 Data Interface Master Clock Speed configuration

This macro indicates the input clock frequency to generate the MCLK to the codec.

Remarks

None.

Configuring the MHC

Description

The following three figures show examples of MHC configurations for the AK7755 Codec Driver, I2S Driver, and the I2C Driver.

Figure 1: AK7755 Codec Driver MHC Configuration

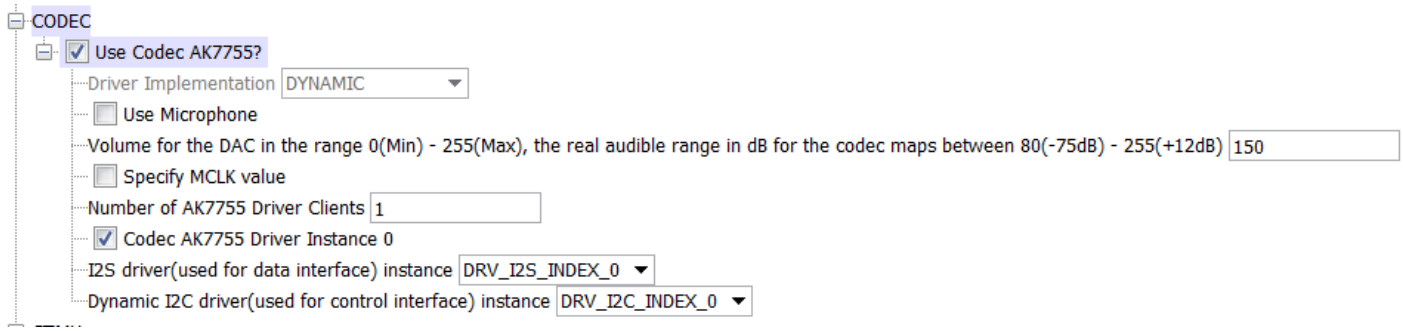


Figure 2: I2S Driver MHC Configuration

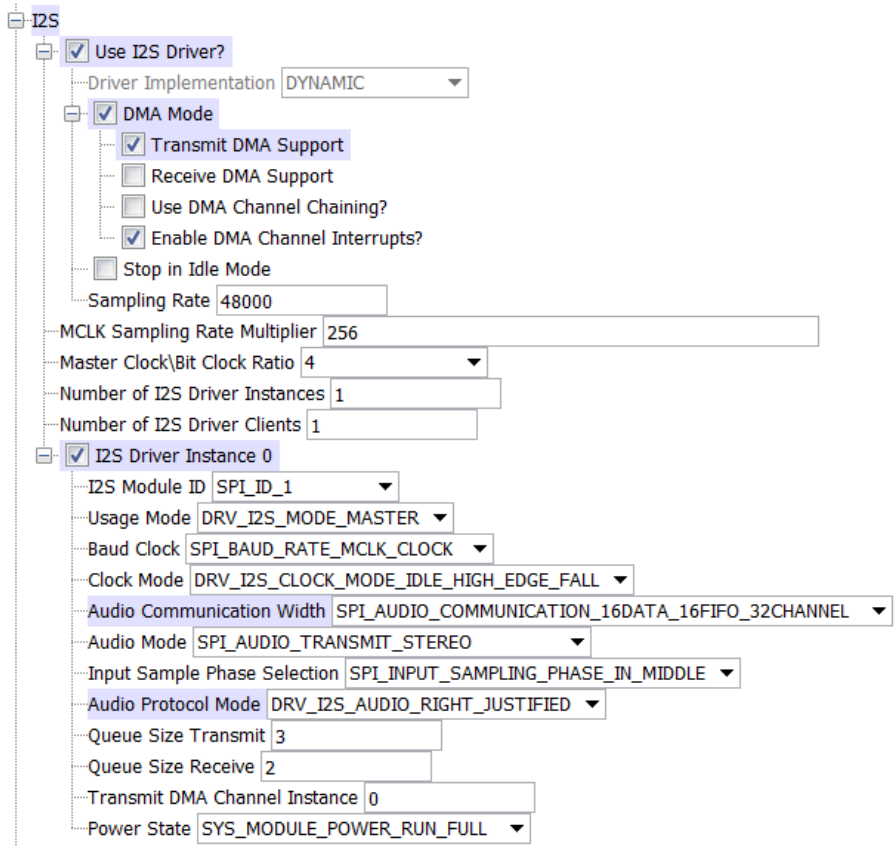
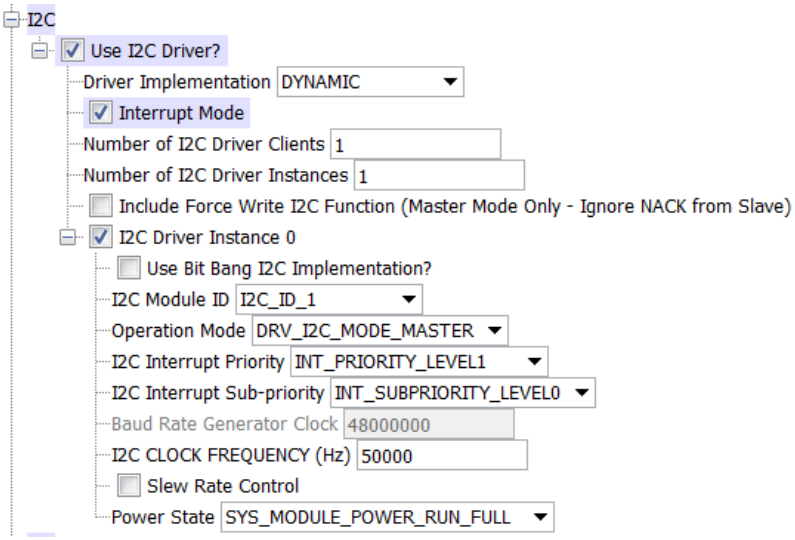


Figure 3: I2C Driver MHC Configuration



Migrating the AK7755 Driver From Earlier Versions of Microchip Harmony

Prior to version 1.08 of MPLAB Harmony, the AK7755 Codec Driver Library used the static I2C driver implementation. Beginning with v1.08 of MPLAB Harmony, applications must use the Dynamic Driver implementation with the MHC configured as shown in Figure 3. In addition, PIC32MZ configurations require the "Include Force Write I2C Function (Master Mode Only - Ignore NACK from Slave)" option to be selected.

Building the Library

This section lists the files that are available in the AK7755 Codec Driver Library.

Description

This section list the files that are available in the `/src` folder of the AK7755 Codec Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/codec/ak7755`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ak7755.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_ak7755.c`	This file contains implementation of the AK7755 Codec Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The AK7755 Codec Driver Library depends on the following modules:

- I2S Driver Library
- I2C Driver Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_AK7755_Close	Closes an opened-instance of the AK7755 Codec Driver.
	DRV_AK7755_Deinitialize	Deinitializes the specified instance of the AK7755 Codec Driver module.
	DRV_AK7755_Initialize	Initializes hardware and data for the instance of the AK7755 DAC module
	DRV_AK7755_Open	Opens the specified AK7755 Codec Driver instance and returns a handle to it
	DRV_AK7755_Tasks	Maintains the driver's control and data interface state machine.
	DRV_AK7755_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_AK7755_CommandEventHandlerSet	Allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.
	DRV_AK7755_SamplingRateSet	This function sets the sampling rate of the media stream.

b) Status Functions

	Name	Description
	DRV_AK7755_SamplingRateGet	This function gets the sampling rate set on the AK7755. **Implementation:** Dynamic
	DRV_AK7755_Status	Gets the current status of the AK7755 Codec Driver module.
	DRV_AK7755_VersionGet	Returns the version of the AK7755 Codec Driver.
	DRV_AK7755_VersionStrGet	This function returns the version of AK7755 Codec Driver in string format.
	DRV_AK7755_VolumeGet	Gets the volume for the AK7755 Codec Driver.

c) Other Functions

	Name	Description
	DRV_AK7755_VolumeSet	This function sets the volume for AK7755 CODEC.
	DRV_AK7755_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_AK7755_BufferAddWrite	Schedule a non-blocking driver write operation.

	DRV_AK7755_BufferAddWriteRead	This is function DRV_AK7755_BufferAddWriteRead.
	DRV_AK7755_IntExtMicSet	Sets up the codec for the internal or the external microphone use.
	DRV_AK7755_MonoStereoMicSet	Sets up the codec for the Mono or Stereo microphone mode.
	DRV_AK7755_MuteOff	Disables AK7755 output for soft mute.
	DRV_AK7755_MuteOn	Allows AK7755 output for soft mute on.

d) Data Types and Constants

	Name	Description
	_DRV_AK7755_H	Include files.
	DRV_AK7755_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK7755_COUNT	Number of valid AK7755 Codec Driver indices
	DRV_AK7755_INDEX_0	AK7755 driver index definitions
	DRV_AK7755_INDEX_1	This is macro DRV_AK7755_INDEX_1.
	DRV_AK7755_INDEX_2	This is macro DRV_AK7755_INDEX_2.
	DRV_AK7755_INDEX_3	This is macro DRV_AK7755_INDEX_3.
	DRV_AK7755_INDEX_4	This is macro DRV_AK7755_INDEX_4.
	DRV_AK7755_INDEX_5	This is macro DRV_AK7755_INDEX_5.
	DRV_AK7755_BICK_FS_FORMAT	This is type DRV_AK7755_BICK_FS_FORMAT.
	DRV_AK7755_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK7755_BUFFER_EVENT_HANDLER	Pointer to a AK7755 Driver Buffer Event handler function.
	DRV_AK7755_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK7755_CHANNEL	Identifies left/right audio channel.
	DRV_AK7755_COMMAND_EVENT_HANDLER	Pointer to a AK7755 Codec Driver command event handler function.
	DRV_AK7755_DAC_INPUT_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK7755_DSP_DIN1_INPUT_FORMAT	This is type DRV_AK7755_DSP_DIN1_INPUT_FORMAT.
	DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT	This is type DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT.
	DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT	This is type DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT.
	DRV_AK7755_DSP_PROGRAM	This is type DRV_AK7755_DSP_PROGRAM.
	DRV_AK7755_INIT	Defines the data required to initialize or reinitialize the AK7755 Codec Driver.
	DRV_AK7755_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK7755_LRCK_IF_FORMAT	This is type DRV_AK7755_LRCK_IF_FORMAT.
	DRV_AK7755_MONO_STEREO_MIC	Identifies the Mic input as Mono/Stereo.

Description

This section describes the API functions of the AK7755 Codec Driver library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_AK7755_Close Function

Closes an opened-instance of the AK7755 Codec Driver.

File

drv_ak7755.h

C

```
void DRV_AK7755_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the AK7755 Codec Driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this function, the handle passed in "handle" must not be used with any of the remaining driver functions. A new handle must be obtained by calling DRV_AK7755_Open before the caller may use the driver again.

Remarks

Usually there is no need for the driver client to verify that the Close operation has completed. The driver will abort any ongoing operations when this function is called.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_AK7755_Open

DRV_AK7755_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_AK7755_Close(DRV_Handle handle)

DRV_AK7755_Deinitialize Function

Deinitializes the specified instance of the AK7755 Codec Driver module.

File

drv_ak7755.h

C

```
void DRV_AK7755_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the specified instance of the AK7755 Codec Driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This function will NEVER block waiting for hardware.

Preconditions

The DRV_AK7755_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_AK7755_Initialize
SYS_STATUS          status;


DRV_AK7755_Deinitialize(object);

status = DRV_AK7755_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK7755_Initialize function

Function

void DRV_AK7755_Deinitialize(SYS_MODULE_OBJ object)

DRV_AK7755_Initialize Function

Initializes hardware and data for the instance of the AK7755 DAC module

File

drv_ak7755.h

C

```
SYS_MODULE_OBJ DRV_AK7755_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the AK7755 Codec Driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized.

Remarks

This function must be called before any other AK7755 function is called.

This function should only be called once during system initialization unless DRV_AK7755_Deinitialize is called to deinitialize the driver instance. This function will NEVER block for hardware access.

Preconditions

DRV_I2S_Initialize must be called before calling this function to initialize the data interface of this codec driver. DRV_SPI_Initialize must be called if SPI driver is used for handling the control interface of this codec driver.

Example

```
DRV_AK7755_INIT                init;
SYS_MODULE_OBJ                 objectHandle;

init->inUse                          = true;
init->status                         = SYS_STATUS_BUSY;
init->numClients                     = 0;
init->i2sDriverModuleIndex           = ak7755Init->i2sDriverModuleIndex;
init->i2cDriverModuleIndex           = ak7755Init->i2cDriverModuleIndex;
init->samplingRate                   = DRV_AK7755_AUDIO_SAMPLING_RATE;
init->audioDataFormat                = DRV_AK7755_AUDIO_DATA_FORMAT_MACRO;

init->isInInterruptContext           = false;

init->commandCompleteCallback = (DRV_AK7755_COMMAND_EVENT_HANDLER)0;
init->commandContextData = 0;
init->mclk_multiplier = DRV_AK7755_MCLK_SAMPLE_FREQ_MULTPLIER;


objectHandle = DRV_AK7755_Initialize(DRV_AK7755_0, (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to be initialized
init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ DRV_AK7755_Initialize

(
const SYS_MODULE_INDEX drvIndex,
const SYS_MODULE_INIT *const init
);

DRV_AK7755_Open Function

Opens the specified AK7755 Codec Driver instance and returns a handle to it

File

drv_ak7755.h

C

```
DRV_HANDLE DRV_AK7755_Open(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the number of client objects allocated via DRV_AK7755_CLIENTS_NUMBER is insufficient.
- if the client is trying to open the driver but driver has been opened exclusively by another client.
- if the driver hardware instance being opened is not initialized or is invalid.
- if the ioIntent options passed are not relevant to this driver.

Description

This function opens the specified AK7755 Codec Driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options are not relevant to this driver. All the data transfer functions of this driver are non blocking.

Only DRV_IO_INTENT_WRITE is a valid ioIntent option as AK7755 is DAC only.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_AK7755_Close function is called. This function will NEVER block waiting for hardware.If the requested intent flags are not supported, the function will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

The DRV_AK7755_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_AK7755_Open(DRV_AK7755_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

DRV_HANDLE DRV_AK7755_Open
(
const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT ioIntent
)

DRV_AK7755_Tasks Function

Maintains the driver's control and data interface state machine.

File

drv_ak7755.h

C

```
void DRV_AK7755_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal control and data interface state machine and implement its control and data interface implementations. This function should be called from the SYS_Tasks function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks function (SYS_Tasks).

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK7755_Initialize

while (true)
{
    DRV_AK7755_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_AK7755_Initialize)

Function

void DRV_AK7755_Tasks(SYS_MODULE_OBJ object);

DRV_AK7755_BufferEventHandlerSet Function

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

File

drv_ak7755.h

C

```
void DRV_AK7755_BufferEventHandlerSet(DRV_HANDLE handle, const DRV_AK7755_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls DRV_AK7755_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK7755_BUFFER_HANDLE bufferHandle;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

// Client registers an event handler with driver

DRV_AK7755_BufferEventHandlerSet(myAK7755Handle,
                APP_AK7755BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK7755_BufferAddWrite(myAK7755handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK7755_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_AK7755BufferEventHandler(DRV_AK7755_BUFFER_EVENT event,
        DRV_AK7755_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK7755_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK7755_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK7755_BufferEventHandlerSet

(

DRV_HANDLE handle,
const DRV_AK7755_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t contextHandle
)

DRV_AK7755_CommandEventHandlerSet Function

Allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

File

drv_ak7755.h

C

```
void DRV_AK7755_CommandEventHandlerSet(DRV_HANDLE handle, const DRV_AK7755_COMMAND_EVENT_HANDLER
eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.

When a client calls DRV_AK7755_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "AK7755 CODEC Specific Client Routines" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the command has completed, it does not need to register a callback.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK7755_BUFFER_HANDLE bufferHandle;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

// Client registers an event handler with driver

DRV_AK7755_CommandEventHandlerSet(myAK7755Handle,
                APP_AK7755CommandEventHandler, (uintptr_t)&myAppObj);

DRV_AK7755_DeEmphasisFilterSet(myAK7755Handle, DRV_AK7755_DEEMPHASIS_FILTER_44_1KHZ)

// Event is received when
// the buffer is processed.

void APP_AK7755CommandEventHandler(uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
            // Last Submitted command is completed.
            // Perform further processing here
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_AK7755_CommandEventHandlerSet
(
DRV_HANDLE handle,
const DRV_AK7755_COMMAND_EVENT_HANDLER eventHandler,
const uintptr_t contextHandle
)

DRV_AK7755_SamplingRateSet Function

This function sets the sampling rate of the media stream.

File

drv_ak7755.h

C

```
void DRV_AK7755_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate);
```

Returns

None.

Description

This function sets the media sampling rate for the client handle.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.
DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

DRV_AK7755_SamplingRateSet(myAK7755Handle, 48000);  //Sets 48000 media sampling rate
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
samplingRate	Sampling frequency in Hz

Function

void DRV_AK7755_SamplingRateSet(DRV_HANDLE handle, uint32_t samplingRate)

b) Status Functions

DRV_AK7755_SamplingRateGet Function

This function gets the sampling rate set on the AK7755.

Implementation: Dynamic

File

drv_ak7755.h

C

```
uint32_t DRV_AK7755_SamplingRateGet(DRV_HANDLE handle);
```

Description

This function gets the sampling rate set on the DAC AK7755.

Remarks

None.

Example

```
uint32_t baudRate;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

baudRate = DRV_AK7755_SamplingRateGet(myAK7755Handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

uint32_t DRV_AK7755_SamplingRateGet(DRV_HANDLE handle)

DRV_AK7755_Status Function

Gets the current status of the AK7755 Codec Driver module.

File

drv_ak7755.h

C

```
SYS_STATUS DRV_AK7755_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized
- SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed
- SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed
- SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This function provides the current status of the AK7755 Codec Driver module.

Remarks

A driver can be opened only when its status is SYS_STATUS_READY.

Preconditions

The DRV_AK7755_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_AK7755_Initialize
SYS_STATUS          AK7755Status;

AK7755Status = DRV_AK7755_Status(object);
if (SYS_STATUS_READY == AK7755Status)
{
    // This means the driver can be opened using the
    // DRV_AK7755_Open function.
```

```
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_AK7755_Initialize function

Function

SYS_STATUS DRV_AK7755_Status(SYS_MODULE_OBJ object)

DRV_AK7755_VersionGet Function

Returns the version of the AK7755 Codec Driver.

File

drv_ak7755.h

C

```
uint32_t DRV_AK7755_VersionGet();
```

Returns

Returns the version of the AK7755 Codec Driver.

Description

The version number returned from the DRV_AK7755_VersionGet function is an unsigned integer in the following decimal format:

- * 10000 + * 100 +

Where the numbers are represented in decimal and the meaning is the same as above. Note that there is no numerical representation of release type.

Remarks

None.

Preconditions

None.

Example 1

- For version "0.03a", return: 0 * 10000 + 3 * 100 + 0
- For version "1.00", return: 1 * 100000 + 0 * 100 + 0

Example 2

```
uint32_t AK7755version;
AK7755version = DRV_AK7755_VersionGet();
```

Function

uint32_t DRV_AK7755_VersionGet(void)

DRV_AK7755_VersionStrGet Function

This function returns the version of AK7755 Codec Driver in string format.

File

drv_ak7755.h

C

```
int8_t* DRV_AK7755_VersionStrGet();
```

Returns

returns a string containing the version of the AK7755 Codec Driver.

Description

The DRV_AK7755_VersionStrGet function returns a string in the format: ".[.][]" Where:

- is the AK7755 Codec Driver's version number.
- is the AK7755 Codec Driver's version number.

- is an optional "patch" or "dot" release number (which is not

included in the string if it equals "00").

- is an optional release type ("a" for alpha, "b" for beta ?

not the entire word spelled out) that is not included if the release is a production version (I.e. Not an alpha or beta).

The String does not contain any spaces. For example, "0.03a" "1.00"

Remarks

None

Preconditions

None.

Example

```
int8_t *AK7755string;
AK7755string = DRV_AK7755_VersionStrGet();
```

Function

int8_t* DRV_AK7755_VersionStrGet(void)

DRV_AK7755_VolumeGet Function

Gets the volume for the AK7755 Codec Driver.

File

drv_ak7755.h

C

```
uint8_t DRV_AK7755_VolumeGet(DRV_HANDLE handle, DRV_AK7755_CHANNEL channel);
```

Returns

None.

Description

This functions gets the current volume programmed to the AK7755 Codec Driver.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
uint8_t volume;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

  volume = DRV_AK7755_VolumeGet(myAK7755Handle, DRV_AK7755_CHANNEL_LEFT);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
channel	argument indicating Left or Right or Both channel volume to be modified

Function

uint8_t DRV_AK7755_VolumeGet(DRV_HANDLE handle, DRV_AK7755_CHANNEL channel)

c) Other Functions

DRV_AK7755_VolumeSet Function

This function sets the volume for AK7755 CODEC.

File

drv_ak7755.h

C

```
void DRV_AK7755_VolumeSet(DRV_HANDLE handle, DRV_AK7755_CHANNEL channel, uint8_t volume);
```

Returns

None

Description

This functions sets the volume value from 0-255, which can attenuate from -115 dB to +12 dB. All decibels below approximately -50 dB are inaudible.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

    DRV_AK7755_VolumeSet(myAK7755Handle,DRV_AK7755_CHANNEL_LEFT, 120);  //Step 120 volume
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's Open function
channel	argument indicating Left or Right or Both channel volume to be modified
volume	Updated volume specified in the range 0-255

Function

void DRV_AK7755_VolumeSet(DRV_HANDLE handle, DRV_AK7755_CHANNEL channel, uint8_t volume);

DRV_AK7755_BufferAddRead Function

Schedule a non-blocking driver read operation.

File

drv_ak7755.h

C

```
void DRV_AK7755_BufferAddRead(const DRV_HANDLE handle, DRV_AK7755_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK7755_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking read operation. The function returns with a valid buffer handle in the bufferHandle argument if the read

request was scheduled successfully. The function adds the request to the hardware instance receive queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK7755_BUFFER_HANDLE_INVALID

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0.
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK7755_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_AK7755_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK7755 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK7755 Codec Driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 device instance and the DRV_AK7755_Status must have returned SYS_STATUS_READY.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ must have been specified in the DRV_AK7755_Open call.

Parameters

Parameters	Description
handle	Handle of the AK7755 instance as return by the DRV_AK7755_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK7755_BufferAddRead

(

const DRV_HANDLE handle,

DRV_AK7755_BUFFER_HANDLE *bufferHandle,

void *buffer, size_t size

)

DRV_AK7755_BufferAddWrite Function

Schedule a non-blocking driver write operation.

File

drv_ak7755.h

C

```
void DRV_AK7755_BufferAddWrite(const DRV_HANDLE handle, DRV_AK7755_BUFFER_HANDLE * bufferHandle, void *
buffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_AK7755_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_AK7755_BUFFER_HANDLE_INVALID:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_AK7755_BUFFER_EVENT_COMPLETE event if

the buffer was processed successfully of DRV_AK7755_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the AK7755 Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another AK7755 Codec Driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 device instance and the DRV_AK7755_Status must have returned SYS_STATUS_READY.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE must have been specified in the DRV_AK7755_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_AK7755_BUFFER_HANDLE bufferHandle;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

// Client registers an event handler with driver

DRV_AK7755_BufferEventHandlerSet(myAK7755Handle,
                APP_AK7755BufferEventHandler, (uintptr_t)&myAppObj);

DRV_AK7755_BufferAddWrite(myAK7755handle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_AK7755_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.

void APP_AK7755BufferEventHandler(DRV_AK7755_BUFFER_EVENT event,
        DRV_AK7755_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_AK7755_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_AK7755_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the AK7755 instance as return by the DRV_AK7755_Open function.
buffer	Data to be transmitted.
size	Buffer size in bytes.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

Function

void DRV_AK7755_BufferAddWrite
(
const DRV_HANDLE handle,
DRV_AK7755_BUFFER_HANDLE *bufferHandle,
void *buffer, size_t size
)

DRV_AK7755_BufferAddWriteRead Function

File

drv_ak7755.h

C

```
void DRV_AK7755_BufferAddWriteRead(const DRV_HANDLE handle, DRV_AK7755_BUFFER_HANDLE * bufferHandle, void *
transmitBuffer, void * receiveBuffer, size_t size);
```

Description

This is function DRV_AK7755_BufferAddWriteRead.

DRV_AK7755_IntExtMicSet Function

Sets up the codec for the internal or the external microphone use.

File

drv_ak7755.h

C

```
void DRV_AK7755_IntExtMicSet(DRV_HANDLE handle, DRV_AK7755_INT_EXT_MIC micInput);
```

Returns

None.

Description

This function sets up the codec for the internal or the external microphone use.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
micInput	Internal vs. External microphone input

Function

void DRV_AK7755_IntExtMicSet(DRV_HANDLE handle);

DRV_AK7755_MonoStereoMicSet Function

Sets up the codec for the Mono or Stereo microphone mode.

File

drv_ak7755.h

C

```
void DRV_AK7755_MonoStereoMicSet(DRV_HANDLE handle, DRV_AK7755_MONO_STEREO_MIC mono_stereo_mic);
```

Returns

None.

Description

This function sets up the codec for the Mono or Stereo microphone mode.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
mono_stereo_mic	Mono/Stereo microphone setup

Function

void DRV_AK7755_MonoStereoMicSet(DRV_HANDLE handle);

DRV_AK7755_MuteOff Function

Disables AK7755 output for soft mute.

File

drv_ak7755.h

C

```
void DRV_AK7755_MuteOff(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables AK7755 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

    DRV_AK7755_MuteOff(myAK7755Handle); //AK7755 output soft mute disabled
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_AK7755_MuteOff(DRV_HANDLE handle)

DRV_AK7755_MuteOn Function

Allows AK7755 output for soft mute on.

File

drv_ak7755.h

C

```
void DRV_AK7755_MuteOn(DRV_HANDLE handle);
```

Returns

None.

Description

This function enables AK7755 output for soft mute.

Remarks

None.

Preconditions

The DRV_AK7755_Initialize function must have been called for the specified AK7755 Codec Driver instance.

DRV_AK7755_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myAK7755Handle is the handle returned
// by the DRV_AK7755_Open function.

DRV_AK7755_MuteOn(myAK7755Handle);  //AK7755 output soft muted
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_AK7755_MuteOn(DRV_HANDLE handle);

d) Data Types and Constants

_DRV_AK7755_H Macro

File

drv_ak7755.h

C

```
#define _DRV_AK7755_H
```

Description

Include files.

DRV_AK7755_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_ak7755.h

C

```
#define DRV_AK7755_BUFFER_HANDLE_INVALID ((DRV_AK7755_BUFFER_HANDLE)(-1))
```

Description

AK7755 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_AK7755_BufferAddWrite and the DRV_AK7755_BufferAddRead function if the buffer add request was not successful.

Remarks

None.

DRV_AK7755_COUNT Macro

Number of valid AK7755 Codec Driver indices

File

drv_ak7755.h

C

```
#define DRV_AK7755_COUNT
```

Description

AK7755 Driver Module Count

This constant identifies the maximum number of AK7755 Codec Driver instances that should be defined by the application. Defining more instances than this constant will waste RAM memory space.

This constant can also be used by the application to identify the number of AK7755 instances on this microcontroller.

Remarks

This value is device-specific.

DRV_AK7755_INDEX_0 Macro

AK7755 driver index definitions

File

drv_ak7755.h

C

```
#define DRV_AK7755_INDEX_0 0
```

Description

Driver AK7755 Module Index

These constants provide AK7755 Codec Driver index definition.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_AK7755_Initialize and DRV_AK7755_Open functions to identify the driver instance in use.

DRV_AK7755_INDEX_1 Macro

File

drv_ak7755.h

C

```
#define DRV_AK7755_INDEX_1 1
```

Description

This is macro DRV_AK7755_INDEX_1.

DRV_AK7755_INDEX_2 Macro

File

drv_ak7755.h

C

```
#define DRV_AK7755_INDEX_2 2
```

Description

This is macro DRV_AK7755_INDEX_2.

DRV_AK7755_INDEX_3 Macro

File

drv_ak7755.h

C

```
#define DRV_AK7755_INDEX_3 3
```

Description

This is macro DRV_AK7755_INDEX_3.

DRV_AK7755_INDEX_4 Macro

File

drv_ak7755.h

C

```
#define DRV_AK7755_INDEX_4 4
```

Description

This is macro DRV_AK7755_INDEX_4.

DRV_AK7755_INDEX_5 Macro

File

drv_ak7755.h

C

```
#define DRV_AK7755_INDEX_5 5
```

Description

This is macro DRV_AK7755_INDEX_5.

DRV_AK7755_BICK_FS_FORMAT Enumeration

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_BICK_64FS,
  DRV_AK7755_BICK_48FS,
  DRV_AK7755_BICK_32FS,
  DRV_AK7755_BICK_256FS
} DRV_AK7755_BICK_FS_FORMAT;
```

Description

This is type DRV_AK7755_BICK_FS_FORMAT.

DRV_AK7755_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_BUFFER_EVENT_COMPLETE,
  DRV_AK7755_BUFFER_EVENT_ERROR,
  DRV_AK7755_BUFFER_EVENT_ABORT
} DRV_AK7755_BUFFER_EVENT;
```

Members

Members	Description
DRV_AK7755_BUFFER_EVENT_COMPLETE	Data was transferred successfully.
DRV_AK7755_BUFFER_EVENT_ERROR	Error while processing the request
DRV_AK7755_BUFFER_EVENT_ABORT	Data transfer aborted (Applicable in DMA mode)

Description

AK7755 Driver Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_AK7755_BufferAddWrite or the DRV_AK7755_BufferAddRead function.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_AK7755_BufferEventHandlerSet function when a buffer transfer request is completed.

DRV_AK7755_BUFFER_EVENT_HANDLER Type

Pointer to a AK7755 Driver Buffer Event handler function.

File

drv_ak7755.h

C

```
typedef void (* DRV_AK7755_BUFFER_EVENT_HANDLER)(DRV_AK7755_BUFFER_EVENT event, DRV_AK7755_BUFFER_HANDLE
bufferHandle, uintptr_t contextHandle);
```

Returns

None.

Description

AK7755 Driver Buffer Event Handler Function

This data type defines the required function signature for the AK7755 Codec Driver buffer event handling callback function. A client must register a

pointer to a buffer event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_AK7755_BUFFER_EVENT_COMPLETE, this means that the data was transferred successfully.

If the event is DRV_AK7755_BUFFER_EVENT_ERROR, this means that the data was not transferred successfully. The bufferHandle parameter contains the buffer handle of the buffer that failed. The DRV_AK7755_BufferProcessedSizeGet function can be called to find out how many bytes were processed.

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK7755_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The buffer handle in bufferHandle expires after this event handler exits. In that the buffer object that was allocated is deallocated by the driver after the event handler exits.

The event handler function executes in the data driver (i.e., I2S) peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

DRV_AK7755_BufferAddWrite function can be called in the event handler to add a buffer to the driver queue.

Example

```
void APP_MyBufferEventHandler( DRV_AK7755_BUFFER_EVENT event,
                               DRV_AK7755_BUFFER_HANDLE bufferHandle,
                               uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_AK7755_BUFFER_EVENT_COMPLETE:
            // Handle the completed buffer.
        break;

        case DRV_AK7755_BUFFER_EVENT_ERROR:
        default:
            // Handle error.
        break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the event relates
context	Value identifying the context of the application that registered the event handling function.

DRV_AK7755_BUFFER_HANDLE Type

Handle identifying a write buffer passed to the driver.

File

drv_ak7755.h

C

```
typedef uintptr_t DRV_AK7755_BUFFER_HANDLE;
```

Description

AK7755 Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_AK7755_BufferAddWrite or DRV_AK7755_BufferAddRead function. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler

function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_AK7755_CHANNEL Enumeration

Identifies left/right audio channel.

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_CHANNEL_LEFT,
  DRV_AK7755_CHANNEL_RIGHT,
  DRV_AK7755_CHANNEL_LEFT_RIGHT,
  DRV_AK7755_NUMBER_OF_CHANNELS
} DRV_AK7755_CHANNEL;
```

Description

AK7755 Audio Channel

This enumeration identifies the left/right audio channel.

Remarks

None.

DRV_AK7755_COMMAND_EVENT_HANDLER Type

Pointer to a AK7755 Codec Driver command event handler function.

File

drv_ak7755.h

C

```
typedef void (* DRV_AK7755_COMMAND_EVENT_HANDLER)(uintptr_t contextHandle);
```

Returns

None.

Description

AK7755 Driver Command Event Handler Function

This data type defines the required function signature for the AK7755 Codec Driver command event handling callback function.

A command is a control instruction to the AK7755 Codec. For example, Mute ON/OFF, Zero Detect Enable/Disable, etc.

A client must register a pointer to a command event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive command related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

The occurrence of this call back means that the last control command was transferred successfully.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_AK7755_CommandEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The event handler function executes in the control data driver interrupt context. It is recommended of the application to not perform process intensive or blocking operations with in this function.

Example

```
void APP_AK7755CommandEventHandler( uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    // Last Submitted command is completed.
```

```
    // Perform further processing here
}
```

Parameters

Parameters	Description
context	Value identifying the context of the application that registered the event handling function.

DRV_AK7755_DAC_INPUT_FORMAT Enumeration

Identifies the Serial Audio data interface format.

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_DAC_INPUT_24BITMSB,
  DRV_AK7755_DAC_INPUT_24BITLSB,
  DRV_AK7755_DAC_INPUT_20BITLSB,
  DRV_AK7755_DAC_INPUT_16BITLSB
} DRV_AK7755_DAC_INPUT_FORMAT;
```

Members

Members	Description
DRV_AK7755_DAC_INPUT_20BITLSB	not supported

Description

AK7755 Audio Data Format

This enumeration identifies the Serial Audio data interface format.

DRV_AK7755_DSP_DIN1_INPUT_FORMAT Enumeration

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_DSP_DIN1_INPUT_24BITMSB,
  DRV_AK7755_DSP_DIN1_INPUT_24BITLSB,
  DRV_AK7755_DSP_DIN1_INPUT_20BITLSB,
  DRV_AK7755_DSP_DIN1_INPUT_16BITLSB
} DRV_AK7755_DSP_DIN1_INPUT_FORMAT;
```

Description

This is type DRV_AK7755_DSP_DIN1_INPUT_FORMAT.

DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT Enumeration

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_DSP_DOUT1_OUTPUT_24BITMSB,
  DRV_AK7755_DSP_DOUT1_OUTPUT_24BITLSB,
  DRV_AK7755_DSP_DOUT1_OUTPUT_20BITLSB,
  DRV_AK7755_DSP_DOUT1_OUTPUT_16BITLSB
} DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT;
```

Description

This is type DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT.

DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT Enumeration

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_DSP_DOUT4_OUTPUT_24BITMSB,
  DRV_AK7755_DSP_DOUT4_OUTPUT_24BITLSB,
  DRV_AK7755_DSP_DOUT4_OUTPUT_20BITLSB,
  DRV_AK7755_DSP_DOUT4_OUTPUT_16BITLSB
} DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT;
```

Description

This is type DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT.

DRV_AK7755_DSP_PROGRAM Enumeration

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_DSP_ECHO_CANCELLATION,
  DRV_AK7755_DSP_REGULAR
} DRV_AK7755_DSP_PROGRAM;
```

Description

This is type DRV_AK7755_DSP_PROGRAM.

DRV_AK7755_INIT Structure

Defines the data required to initialize or reinitialize the AK7755 Codec Driver.

File

drv_ak7755.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX i2sDriverModuleIndex;
  SYS_MODULE_INDEX i2cDriverModuleIndex;
  uint32_t samplingRate;
  uint8_t volume;
} DRV_AK7755_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX i2sDriverModuleIndex;	Identifies data module (I2S) driver ID for data interface of CODEC
SYS_MODULE_INDEX i2cDriverModuleIndex;	Identifies data module (I2C) driver ID for control interface of CODEC
uint32_t samplingRate;	Sampling rate
uint8_t volume;	Volume

Description

AK7755 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the AK7755 Codec Driver.

Remarks

None.

DRV_AK7755_INT_EXT_MIC Enumeration

Identifies the Mic input source.

File

drv_ak7755.h

C

```
typedef enum {
  INT_MIC,
  EXT_MIC
} DRV_AK7755_INT_EXT_MIC;
```

Description

AK7755 Mic Internal / External Input

This enumeration identifies the Mic input source.

Remarks

None.

DRV_AK7755_LRCK_IF_FORMAT Enumeration

File

drv_ak7755.h

C

```
typedef enum {
  DRV_AK7755_LRCK_IF_STANDARD,
  DRV_AK7755_LRCK_IF_I2S_COMPATIBLE,
  DRV_AK7755_LRCK_IF_PCM_SHORT_FRAME,
  DRV_AK7755_LRCK_IF_PCM_LONG_FRAME
} DRV_AK7755_LRCK_IF_FORMAT;
```

Description

This is type DRV_AK7755_LRCK_IF_FORMAT.

DRV_AK7755_MONO_STEREO_MIC Enumeration

Identifies the Mic input as Mono/Stereo.

File

drv_ak7755.h

C

```
typedef enum {
  ALL_ZEROS,
  MONO_RIGHT_CHANNEL,
  MONO_LEFT_CHANNEL,
  STEREO
} DRV_AK7755_MONO_STEREO_MIC;
```

Description

AK7755 Mic Mono/Stereo Input

This enumeration identifies the Mic input as Mono/Stereo.

Remarks

None.

Files

Files

Name	Description
drv_ak7755.h	AK7755 CODEC Driver Interface header file
drv_ak7755_config_template.h	AK7755 Codec Driver configuration template.

Description

This section lists the source and header files used by the AK7755Codec Driver Library.

drv_ak7755.h

AK7755 CODEC Driver Interface header file

Enumerations

	Name	Description
	DRV_AK7755_BICK_FS_FORMAT	This is type DRV_AK7755_BICK_FS_FORMAT.
	DRV_AK7755_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_AK7755_CHANNEL	Identifies left/right audio channel.
	DRV_AK7755_DAC_INPUT_FORMAT	Identifies the Serial Audio data interface format.
	DRV_AK7755_DSP_DIN1_INPUT_FORMAT	This is type DRV_AK7755_DSP_DIN1_INPUT_FORMAT.
	DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT	This is type DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT.
	DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT	This is type DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT.
	DRV_AK7755_DSP_PROGRAM	This is type DRV_AK7755_DSP_PROGRAM.
	DRV_AK7755_INT_EXT_MIC	Identifies the Mic input source.
	DRV_AK7755_LRCK_IF_FORMAT	This is type DRV_AK7755_LRCK_IF_FORMAT.
	DRV_AK7755_MONO_STEREO_MIC	Identifies the Mic input as Mono/Stereo.

Functions

	Name	Description
	DRV_AK7755_BufferAddRead	Schedule a non-blocking driver read operation.
	DRV_AK7755_BufferAddWrite	Schedule a non-blocking driver write operation.
	DRV_AK7755_BufferAddWriteRead	This is function DRV_AK7755_BufferAddWriteRead.
	DRV_AK7755_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_AK7755_Close	Closes an opened-instance of the AK7755 Codec Driver.
	DRV_AK7755_CommandEventHandlerSet	Allows a client to identify a command event handling function for the driver to call back when the last submitted command have finished.
	DRV_AK7755_Deinitialize	Deinitializes the specified instance of the AK7755 Codec Driver module.
	DRV_AK7755_Initialize	Initializes hardware and data for the instance of the AK7755 DAC module
	DRV_AK7755_IntExtMicSet	Sets up the codec for the internal or the external microphone use.
	DRV_AK7755_MonoStereoMicSet	Sets up the codec for the Mono or Stereo microphone mode.
	DRV_AK7755_MuteOff	Disables AK7755 output for soft mute.
	DRV_AK7755_MuteOn	Allows AK7755 output for soft mute on.
	DRV_AK7755_Open	Opens the specified AK7755 Codec Driver instance and returns a handle to it
	DRV_AK7755_SamplingRateGet	This function gets the sampling rate set on the AK7755. **Implementation:** Dynamic
	DRV_AK7755_SamplingRateSet	This function sets the sampling rate of the media stream.
	DRV_AK7755_Status	Gets the current status of the AK7755 Codec Driver module.
	DRV_AK7755_Tasks	Maintains the driver's control and data interface state machine.
	DRV_AK7755_VersionGet	Returns the version of the AK7755 Codec Driver.
	DRV_AK7755_VersionStrGet	This function returns the version of AK7755 Codec Driver in string format.
	DRV_AK7755_VolumeGet	Gets the volume for the AK7755 Codec Driver.
	DRV_AK7755_VolumeSet	This function sets the volume for AK7755 CODEC.

Macros

	Name	Description
	_DRV_AK7755_H	Include files.
	DRV_AK7755_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_AK7755_COUNT	Number of valid AK7755 Codec Driver indices
	DRV_AK7755_INDEX_0	AK7755 driver index definitions
	DRV_AK7755_INDEX_1	This is macro DRV_AK7755_INDEX_1.
	DRV_AK7755_INDEX_2	This is macro DRV_AK7755_INDEX_2.
	DRV_AK7755_INDEX_3	This is macro DRV_AK7755_INDEX_3.
	DRV_AK7755_INDEX_4	This is macro DRV_AK7755_INDEX_4.
	DRV_AK7755_INDEX_5	This is macro DRV_AK7755_INDEX_5.
	DRV_I2C_INDEX	This is macro DRV_I2C_INDEX.

Structures

	Name	Description
	DRV_AK7755_INIT	Defines the data required to initialize or reinitialize the AK7755 Codec Driver.

Types

	Name	Description
	DRV_AK7755_BUFFER_EVENT_HANDLER	Pointer to a AK7755 Driver Buffer Event handler function.
	DRV_AK7755_BUFFER_HANDLE	Handle identifying a write buffer passed to the driver.
	DRV_AK7755_COMMAND_EVENT_HANDLER	Pointer to a AK7755 Codec Driver command event handler function.

Description

AK7755 CODEC Driver Interface

The AK7755 CODEC device driver interface provides a simple interface to manage the AK7755 16/24-Bit Codec that can be interfaced Microchip Microcontroller. This file provides the interface definition for the AK7755 Codec device driver.

File Name

drv_ak7755.h

Company

Microchip Technology Inc.

drv_ak7755_config_template.h

AK7755 Codec Driver configuration template.

Macros

	Name	Description
	DRV_AK7755_BCLK_BIT_CLK_DIVISOR	Sets up the BCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.
	DRV_AK7755_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_AK7755_INPUT_REFCLOCK	Identifies the input REFCLOCK source to generate the MCLK to the codec.
	DRV_AK7755_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_AK7755_MCLK_SAMPLE_FREQ_MULTPLIER	Sets up the MCLK to LRCK ratio to generate the audio stream for the specified sampling frequency.
	DRV_AK7755_MCLK_SOURCE	Indicates the input clock frequency to generate the MCLK to the codec.

Description

AK7755 Codec Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_ak7755_config_template.h

Company

Microchip Technology Inc.

Comparator Driver Library

This section describes the Comparator Driver Library.

Introduction

The Comparator Static Driver provides a high-level interface to manage the Comparator module on the Microchip family of microcontrollers.

Description

Through MHC, this driver provides an API to initialize the Comparator module, as well as reference channels, CVREF, inputs, and interrupts.

Library Interface

Function(s)

	Name	Description
	DRV_CMP_Initialize	Initializes the Comparator instance for the specified driver index. **Implementation:** Static

Description

This section describes the Application Programming Interface (API) functions of the Comparator Driver Library.

Function(s)

DRV_CMP_Initialize Function

Initializes the Comparator instance for the specified driver index.

Implementation: Static

File

help_drv_cmp.h

C

```
void DRV_CMP_Initialize();
```

Returns

None.

Description

This routine initializes the Comparator driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters. The driver instance index is independent of the Comparator module ID. For example, driver instance 0 can be assigned to Comparator 2.

Remarks

This routine must be called before any other Comparator routine is called. This routine should only be called once during system initialization.

Preconditions

None.

Function

void DRV_CMP_Initialize(void)

CPLD XC2C64A Driver Library

This section describes the CPLD XC2C64A Driver Library.

Introduction

This library provides an interface to manage the CPLD XC2C64A devices on Microchip starter kits.

Description

A CPLD is provided on the Multimedia Expansion Board (MEB), which can be used to configure the graphics controller bus interface, SPI channel and Chip Selects used for SPI Flash, the MRF24WBOMA, and the expansion slot. The general I/O inputs are used to change the configuration, which can be done at run-time.

Specific CPLD configuration information is available in the *"Multimedia Expansion Board (MEB) User's Guide"* (DS60001160), which is available from the MEB product page: http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=DM320005

Using the Library

This topic describes the basic architecture of the CPLD XC2C64A Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_xc2c64a.h`

The interface to the CPLD XC2C64A Driver Library is defined in the `drv_xc2c64a.h` header file. Any C language source (`.c`) file that uses the CPLD XC2C64A Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the CPLD XC2C64A Driver.

Library Interface Section	Description
Functions	Provides CPLD XC2C64A initialization and configuration functions.

Configuring the Library

The configuration of the CPLD XC2C64A Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the CPLD XC2C64A Driver. Based on the selections made, the CPLD XC2C64A may support the selected features. These configuration settings will apply to all instances of the CPLD XC2C64A Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the CPLD XC2C64A Driver Library.

Description

This section list the files that are available in the `/src` folder of the CPLD XC2C64A Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/cpld/xc2c64a`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_xc2c64a.h`	Header file that exports the CPLD XC2C64A Driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/drv_xc2c64a.c	Basic CPLD XC2C64A Driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The CPLD XC2C64A Driver Library is not dependent on other modules.

Library Interface

a) Functions

	Name	Description
	CPLDGetDeviceConfiguration	Returns the selected device. **Implementation:** Static
	CPLDGetGraphicsConfiguration	Returns the selected PMP bus, 8 or 16-bit, interface to the graphics controller. **Implementation:** Static
	CPLDGetSPIConfiguration	Returns the selected SPI Channel. **Implementation:** Static
	CPLDInitialize	Initializes the control I/O to the CPLD and places the CPLD in a known state. **Implementation:** Static
	CPLDSetGraphicsConfiguration	Selects the PMP bus, 8 or 16-bit, interface to the graphic controller. **Implementation:** Static
	CPLDSetSPIFlashConfiguration	Selects the SPI Flash device. **Implementation:** Static
	CPLDSetWiFiConfiguration	Selects the Wi-Fi device. **Implementation:** Static
	CPLDSetZigBeeConfiguration	Selects the ZigBee/MiWi device. **Implementation:** Static

b) Data Types and Constants

	Name	Description
	CPLD_DEVICE_CONFIGURATION	CPLD device configuration.
	CPLD_GFX_CONFIGURATION	CPLD graphics controller PMP bus configuration.
	CPLD_SPI_CONFIGURATION	CPLD SPI channel selection.

Description

This section describes the API functions of the CPLD XC2C64A Driver Library.

Refer to each section for a detailed description.

a) Functions

CPLDGetDeviceConfiguration Function

Returns the selected device.

Implementation: Static

File

drv_xc2c64a.h

C

```
CPLD_DEVICE_CONFIGURATION CPLDGetDeviceConfiguration();
```

Returns

- CPLD_DEVICE_SPI_FLASH - SPI Flash.
- CPLD_DEVICE_WiFi - Zero G 802.11 Wi-Fi.
- CPLD_DEVICE_ZIGBEE - ZigBee/MiWi.

Description

This routine returns the selected CPLD device.

Remarks

None.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

```
// Initialize the CPLD
CPLDInitialize();

if(CPLDGetDeviceConfiguration() != CPLD_DEVICE_SPI_FLASH)
{
    // error - not setup as default

}
```

Function

CPLD_DEVICE_CONFIGURATION CPLDGetDeviceConfiguration(void)

CPLDGetGraphicsConfiguration Function

Returns the selected PMP bus, 8 or 16-bit, interface to the graphics controller.

Implementation: Static

File

drv_xc2c64a.h

C

```
CPLD_GFX_CONFIGURATION CPLDGetGraphicsConfiguration();
```

Returns

- CPLD_GFX_CONFIG_8BIT - Graphics controller is configured for 8-bit PMP data bus interface.
- CPLD_GFX_CONFIG_16BIT - Graphics controller is configured for 16-bit PMP data bus interface.

Description

This routine gets the configuration of the PMP, 8 or 16-bit, data bus interface.

Remarks

None.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

```
// Initialize the CPLD
CPLDInitialize();

if(CPLDGetGraphicsConfiguration() != CPLD_GFX_CONFIG_8BIT)
{
    // error - not setup as default
}
```

Function

CPLD_GFX_CONFIGURATION CPLDGetGraphicsConfiguration(void)

CPLDGetSPIConfiguration Function

Returns the selected SPI Channel.

Implementation: Static

File

drv_xc2c64a.h

C

```
CPLD_SPI_CONFIGURATION CPLDGetSPIConfiguration();
```

Returns

- CPLD_SPI2A - SPI Channel 2A with chip select PORT G bit 9 and external interrupt 1 or 3
- CPLD_SPI3A - SPI Channel 3A with chip select PORT F bit 12 and change notice 9
- CPLD_SPI2 - SPI Channel 2 with chip select PORT G bit 9 and external interrupt 1 or 3

Description

This routine returns the selected SPI channel.

Remarks

SPI channels 2 and 2A are located on the same pins. SPI channels 2A and 3A are only available on PIC32MX5xx/6xx/7xx series parts.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

```
// Initialize the CPLD
CPLDInitialize();

if(CPLDGetSPIConfiguration() != CPLD_SPI2A)
{
    // error - not setup as default

}
```

Function

CPLD_SPI_CONFIGURATION CPLDGetSPIConfiguration(void)

CPLDInitialize Function

Initializes the control I/O to the CPLD and places the CPLD in a known state.

Implementation: Static

File

drv_xc2c64a.h

C

```
void CPLDInitialize();
```

Returns

None.

Description

This routine configures the control I/O and places the CPLD in a known state.

- Graphics Controller Bus - 8-bit PMP data interface.
- SPI Channel - SPI2/SPI2A.

- Chip Select - PORT G bit 9.
- External Interrupt 1 or 3
- Device - SPI Flash.

Remarks

None.

Preconditions

None.

Example

```
// Initialize the CPLD
CPLDInitialize();

// CPLD is configured in the default state
```

Function

void CPLDInitialize(void)

CPLDSetGraphicsConfiguration Function

Selects the PMP bus, 8 or 16-bit, interface to the graphic controller.

Implementation: Static

File

drv_xc2c64a.h

C

```
void CPLDSetGraphicsConfiguration(CPLD_GFX_CONFIGURATION configuration);
```

Returns

None.

Description

This routine sets the configuration pins on the graphics controller to select between an 8 or 16-bit data bus interface.

Remarks

The graphics controller interface configuration must be done before initializing the graphics controller.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

Setting the graphics controller to a 16-bit interface

```
// Initialize the CPLD
CPLDInitialize();

// configure the graphics controller for a 16-bit PMP interface.
CPLDSetGraphicsConfiguration(CPLD_GFX_CONFIG_16BIT);
```

Setting the graphics controller to a 8-bit interface

```
// Initialize the CPLD
CPLDInitialize();

// configure the graphics controller for a 8-bit PMP interface.
CPLDSetGraphicsConfiguration(CPLD_GFX_CONFIG_8BIT);
```

Parameters

Parameters	Description
configuration	the type of interface configuration.

Function

void CPLDSetGraphicsConfiguration(CPLD_GFX_CONFIGURATION configuration)

CPLDSetSPIFlashConfiguration Function

Selects the SPI Flash device.

Implementation: Static

File

drv_xc2c64a.h

C

```
void CPLDSetSPIFlashConfiguration(CPLD_SPI_CONFIGURATION configuration);
```

Returns

None.

Description

This routine configures the CPLD to communicate to the SPI Flash device with the selected SPI channel and Chip Select.

Remarks

SPI channels 2 and 2A are located on the same pins. SPI channels 2A and 3A are only available on PIC32MX5xx/6xx/7xx series parts.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

Setting CPLD to SPI Flash using SPI channel 2 and chip select PORT G bit 9

```
// Initialize the CPLD
CPLDInitialize();

// configure the SPI Flash to use SPI channel 2 and chip select PORT G bit 9
CPLDSetSPIFlashConfiguration(CPLD_SPI2);
```

Setting CPLD to SPI Flash using SPI channel 2A and chip select PORT G bit 9

```
// Initialize the CPLD
CPLDInitialize();

// configure the SPI Flash to use SPI channel 2A and chip select PORT G bit 9
CPLDSetSPIFlashConfiguration(CPLD_SPI2A);
```

Setting CPLD to SPI Flash using SPI channel 3A and chip select PORT F bit 12

```
// Initialize the CPLD
CPLDInitialize();

// configure the SPI Flash to use SPI channel 3A and chip select PORT F bit 12
CPLDSetSPIFlashConfiguration(CPLD_SPI3A);
```

Parameters

Parameters	Description
configuration	the type of SPI channel used by the SPI Flash device.

Function

void CPLDSetSPIFlashConfiguration(CPLD_SPI_CONFIGURATION configuration)

CPLDSetWiFiConfiguration Function

Selects the Wi-Fi device.

Implementation: Static

File

drv_xc2c64a.h

C

```
void CPLDSetWiFiConfiguration(CPLD_SPI_CONFIGURATION configuration);
```

Returns

None.

Description

This routine configures the CPLD to communicate to the Wi-Fi device with the selected SPI channel, chip select and external interrupt or change notice.

Remarks

SPI channels 2 and 2A are located on the same pins. SPI channels 2A and 3A are only available on PIC32MX5xx/6xx/7xx series parts.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

Setting CPLD to Wi-Fi using SPI channel 2, chip select PORT G bit 9 and external interrupt 3

```
// Initialize the CPLD
CPLDInitialize();

// configure the Wi-Fi to use SPI channel 2, chip select PORT G bit 9 and external interrupt 3
CPLDSetWiFiConfiguration(CPLD_SPI2);
```

Setting CPLD to Wi-Fi using SPI channel 2A, chip select PORT G bit 9 and external interrupt 3

```
// Initialize the CPLD
CPLDInitialize();

// configure the Wi-Fi to use SPI channel 2A, chip select PORT G bit 9 and external interrupt 3
CPLDSetWiFiConfiguration(CPLD_SPI2A);
```

Setting CPLD to Wi-Fi using SPI channel 3A, chip select PORT F bit 12 and change notice 9

```
// Initialize the CPLD
CPLDInitialize();

// configure the Wi-Fi to use SPI channel 3A, chip select PORT F bit 12 and change notice 9
CPLDSetWiFiConfiguration(CPLD_SPI3A);
```

Parameters

Parameters	Description
configuration	the type of SPI channel used by the Wi-Fi device.

Function

void CPLDSetWiFiConfiguration(CPLD_SPI_CONFIGURATION configuration)

CPLDSetZigBeeConfiguration Function

Selects the ZigBee/MiWi device.

Implementation: Static

File

drv_xc2c64a.h

C

```
void CPLDSetZigBeeConfiguration(CPLD_SPI_CONFIGURATION configuration);
```

Returns

None.

Description

This routine configures the CPLD to communicate to the ZigBee/MiWi device with the selected SPI channel, chip select and external interrupt or change notice.

Remarks

SPI channels 2 and 2A are located on the same pins. SPI channels 2A and 3A are only available on PIC32MX5xx/6xx/7xx series parts.

Preconditions

The initialization routine, CPLDInitialize, must be called.

Example

Setting CPLD to ZigBee/MiWi using SPI channel 2, chip select PORT G bit 9 and external interrupt 3

```
// Initialize the CPLD
CPLDInitialize();

// configure the ZigBee/MiWi to use SPI channel 2, chip select PORT G bit 9 and external interrupt 3
CPLDSetZigBeeConfiguration(CPLD_SPI2);
```

Setting CPLD to ZigBee/MiWi using SPI channel 2A, chip select PORT G bit 9 and external interrupt 3

```
// Initialize the CPLD
CPLDInitialize();

// configure the ZigBee/MiWi to use SPI channel 2A, chip select PORT G bit 9 and external interrupt 3
CPLDSetZigBeeConfiguration(CPLD_SPI2A);
```

Setting CPLD to ZigBee/MiWi using SPI channel 3A, chip select PORT F bit 12 and change notice 9

```
// Initialize the CPLD
CPLDInitialize();

// configure the ZigBee/MiWi to use SPI channel 3A, chip select PORT F bit 12 and change notice 9
CPLDSetZigBeeConfiguration(CPLD_SPI3A);
```

Parameters

Parameters	Description
configuration	the type of SPI channel used by the ZigBee/MiWi device.

Function

void CPLDSetZigBeeConfiguration(CPLD_SPI_CONFIGURATION configuration)

b) Data Types and Constants

CPLD_DEVICE_CONFIGURATION Enumeration

CPLD device configuration.

File

drv_xc2c64a.h

C

```
typedef enum {
  CPLD_DEVICE_SPI_FLASH,
  CPLD_DEVICE_WiFi,
  CPLD_DEVICE_ZIGBEE
} CPLD_DEVICE_CONFIGURATION;
```

Members

Members	Description
CPLD_DEVICE_SPI_FLASH	SPI Flash
CPLD_DEVICE_WiFi	Zero G Wi-Fi
CPLD_DEVICE_ZIGBEE	ZigBee/MiWi

Description

The CPLD can be configured to communicate to three different devices. The application may call routine, CPLDGetDeviceConfiguration, to obtain what device the CPLD is configured to communicate with.

Remarks

None.

Example

```
// select 16-bit PMP data bus
if(CPLDGetDeviceConfiguration() != CPLD_DEVICE_SPI_FLASH)
{
    // error - not default configuration
}
```

CPLD_GFX_CONFIGURATION Enumeration

CPLD graphics controller PMP bus configuration.

File

drv_xc2c64a.h

C

```
typedef enum {
  CPLD_GFX_CONFIG_8BIT,
  CPLD_GFX_CONFIG_16BIT
} CPLD_GFX_CONFIGURATION;
```

Members

Members	Description
CPLD_GFX_CONFIG_8BIT	Configure the Graphics Controller to use 8-bit PMP data bus
CPLD_GFX_CONFIG_16BIT	Configure the Graphics Controller to use 16-bit PMP data bus

Description

The application can select what PMP bus configuration, 8 or 16-bit data bus, when interfacing with the graphics controller.

Remarks

None.

Example

```
// select 16-bit PMP data bus
CPLDSetGraphicsConfiguration(CPLD_GFX_CONFIG_16BIT);
```

CPLD_SPI_CONFIGURATION Enumeration

CPLD SPI channel selection.

File

drv_xc2c64a.h

C

```
typedef enum {
  CPLD_SPI2A,
  CPLD_SPI3A,
  CPLD_SPI2
} CPLD_SPI_CONFIGURATION;
```

Members

Members	Description
CPLD_SPI2A	PIC32 SPI Channel 2A and chip select PORT G bit 9
CPLD_SPI3A	PIC32 SPI Channel 3A and chip select PORT F bit 12
CPLD_SPI2	PIC32 SPI Channel 2 and chip select PORT G bit 9

Description

The application can select what SPI channel will be used as the communication interface. It will also select the Chip Select use for the device.

Remarks

Only one SPI channel can be select for a device. SPI channels 2 and 2A are located on the same pins. SPI channels 2A and 3A are only available on PIC32MX5xx/6xx/7xx series devices.

Example

```
// select SPI channel two for SPI Flash
CPLDSetSPIFlashConfiguration(CPLD_SPI2);
```

Files

Files

Name	Description
drv_xc2c64a.h	This file contains the interface definition for the CUPLD controller.

Description

This section lists the source and header files used by the SPI Flash Driver Library.

drv_xc2c64a.h

This file contains the interface definition for the CUPLD controller.

Enumerations

	Name	Description
	CPLD_DEVICE_CONFIGURATION	CPLD device configuration.
	CPLD_GFX_CONFIGURATION	CPLD graphics controller PMP bus configuration.
	CPLD_SPI_CONFIGURATION	CPLD SPI channel selection.

Functions

	Name	Description
	CPLDGetDeviceConfiguration	Returns the selected device. **Implementation:** Static
	CPLDGetGraphicsConfiguration	Returns the selected PMP bus, 8 or 16-bit, interface to the graphics controller. **Implementation:** Static
	CPLDGetSPIConfiguration	Returns the selected SPI Channel. **Implementation:** Static
	CPLDInitialize	Initializes the control I/O to the CPLD and places the CPLD in a known state. **Implementation:** Static
	CPLDSetGraphicsConfiguration	Selects the PMP bus, 8 or 16-bit, interface to the graphic controller. **Implementation:** Static
	CPLDSetSPIFlashConfiguration	Selects the SPI Flash device. **Implementation:** Static
	CPLDSetWiFiConfiguration	Selects the Wi-Fi device. **Implementation:** Static
	CPLDSetZigBeeConfiguration	Selects the ZigBee/MiWi device. **Implementation:** Static

Description

CUPLD Controller Interface File.

This library provides a low-level abstraction of the CUPLD device. It can be used to simplify low-level access to the device without the necessity of interacting directly with the communication module's registers, thus hiding differences from one serial device variant to another.

File Name

drv_xc2c64a.h

Company

Microchip Technology Inc.

ENC28J60 Driver Library Help

This section provides information on the ENC28J60 Driver Library.

Introduction

This library provides a driver-level abstraction of the ENC28J60 integrated Ethernet MAC and 10Base-T PHY that can be connected to the PIC32. The driver implements the virtual MAC driver model that the MPLAB Harmony TCP/IP Stack requires. Please see the TCP/IP Stack Library MAC Driver Module for details.

The "Host-To-Network"_layer of a TCP/IP stack organization covers the Data Link and Physical Layers of the standard OSI stack. The Ethernet Controller provides the Data Link or Media Access Control Layer, in addition to other functions discussed in this section.

Description

The ENC28J60 External MAC and PHY is an external module to the PIC32 that is connected through a Serial Peripheral Interface (SPI). This driver interfaces with the SPI driver to communicate with the external device to implement a complete Ethernet node in a system.

The following are some of the key features of this module:

- Supports 10 Mbps physical-to-physical layer Ethernet data transfer
- Full-Duplex and Half-Duplex operation
- Broadcast, Multicast and Unicast packets
- Hardware flow control for both Full and Half-Duplex mode
- Fully configurable interrupts
- Configurable receive packet filtering using:
- 64-bit Hash Table
- 64-byte Pattern Match
- Magic Packet™ Filtering
- Supports Packet Payload Checksum calculation
- CRC Check
- Supports SPI interface

Using the Library

This topic describes the basic architecture and functionality of the software driver for the ENC28J60 stand-alone Ethernet Controller with SPI, and is meant for advanced users or TCP/IP stack driver developers.

Description

The user of this driver is the MPLAB Harmony TCP/IP stack. This Ethernet driver is not intended as a system-wide driver that the application or other system modules may use. It is intended for the sole use of the MPLAB Harmony TCP/IP stack and implements the virtual MAC model required by the stack.

Interface Header File: `drv_enc28j60.h`

The interface to the ENC28J60 Driver Library is defined in the `drv_enc28j60.h` header file. Any C language source (`.c`) file that uses the ENC28J60 Driver Library should include `drv_enc28j60.h`.

Library File: The ENC28J60 Driver Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the Understanding MPLAB Harmony section for how the driver interacts with the framework.

Abstraction Model

The ENC28J60 Driver Library provides the low-level abstraction of the communications protocol to communicate to the ENC28J60 external MAC though the SPI peripheral on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the ENC28J60 Driver Library interface.

Description

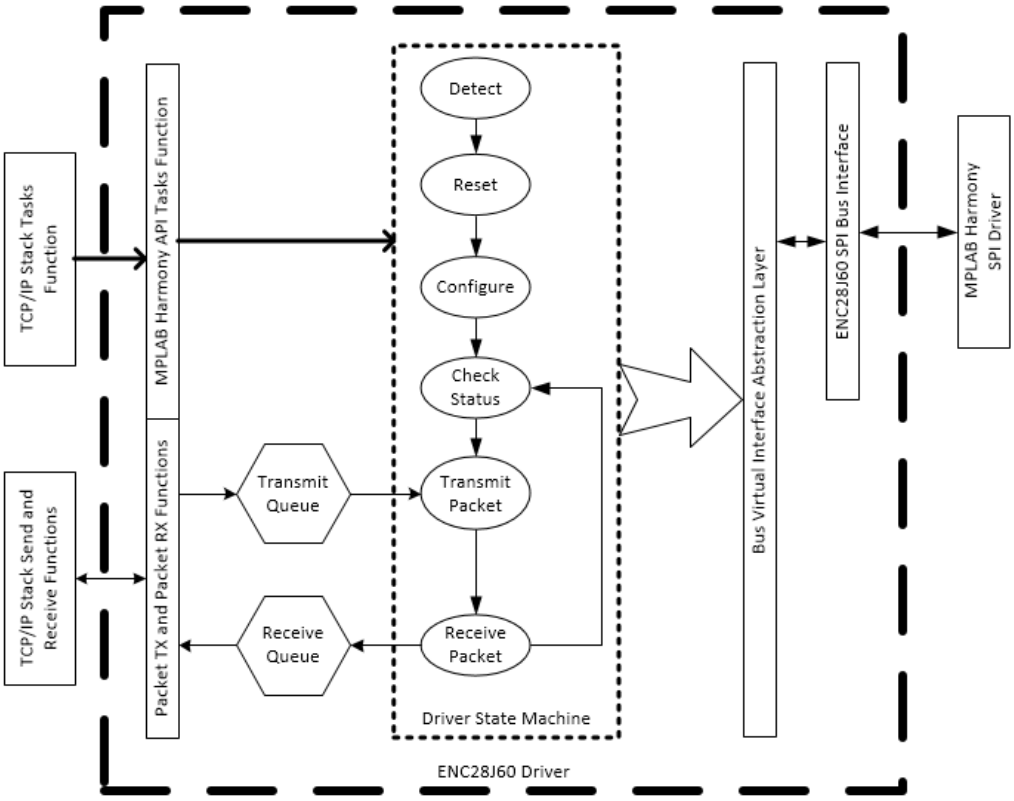
The ENC28J60 Driver library has several different layers to it, as illustrated in the following figure. The interface layer has two main sections that are used the most often: The Tasks function, and the TCP/IP Send and Receive functions.

The Tasks function manages the internal state machine which detects, resets, and then configures the ENC28J60 External MAC. It also handles the monitoring of the hardware status, sending and receiving packets.

The TCP/IP Send and Receive functions interact with the RAM-based queue of packets that are queued to send and packets that have been queued waiting for pick-up by the stack.

The main state machine does not interface directly to the SPI bus, but instead, interfaces to a virtual bus abstraction layer that allows for the replacement of the specific underlying bus implementation.

Abstraction Model



Library Overview

Refer to the section Driver Overview for how the driver operates in a system.

The library interface routines are divided into various sub-sections, each sub-section addresses one of the blocks or the overall operation of the ENC28J60 Driver Library.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Data Transfer Functions	Provides data transfer functions available in the configuration.
Status Functions	Provides status functions.
Miscellaneous Functions	Provides miscellaneous driver functions.

How the Library Works

The library provides interfaces to support the TCP/IP virtual MAC interface.

Configuring the SPI Driver

This section describes the configuration settings for the ENC28J60 Driver Library.

Description

Configuration

The ENC hardware requires a specific configuration of the SPI driver to work correctly. Inside the MHC SPI Driver configuration be sure to select:

- Run the SPI at frequencies of at least 8 MHz
- Clock mode of DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL
- Input phase of SPI_INPUT_SAMPLING_PHASE_AT_END

Recommended Settings

- Interrupt Driver mode
- Enhanced Buffer mode
- DMA mode enabled:
- DMA block transfer size of at least 1600 bytes
- Size of DMA buffer for dummy data of at least 1600 bytes
- Ensure when setting up DMA in interrupt mode that the DMA interrupts are a higher priority than the SPI Driver interrupt

Example:

```
/*** SPI Driver Static Allocation Options ***/
#define DRV_SPI_INSTANCES_NUMBER           1
#define DRV_SPI_CLIENTS_NUMBER             1
#define DRV_SPI_ELEMENTS_PER_QUEUE         30

/*** SPI Driver DMA Options ***/
#define DRV_SPI_DMA_TXFER_SIZE             2048
#define DRV_SPI_DMA_DUMMY_BUFFER_SIZE      2048

/* SPI Driver Instance 0 Configuration */
#define DRV_SPI_SPI_ID_IDX0              SPI_ID_1
#define DRV_SPI_TASK_MODE_IDX0           DRV_SPI_TASK_MODE_ISR
#define DRV_SPI_SPI_MODE_IDX0            DRV_SPI_MODE_MASTER
#define DRV_SPI_ALLOW_IDLE_RUN_IDX0      false
#define DRV_SPI_SPI_PROTOCOL_TYPE_IDX0   DRV_SPI_PROTOCOL_TYPE_STANDARD
#define DRV_SPI_SPI_PROTOCOL_TYPE_IDX0   DRV_SPI_PROTOCOL_TYPE_STANDARD
#define DRV_SPI_COMM_WIDTH_IDX0          SPI_COMMUNICATION_WIDTH_8BITS
#define DRV_SPI_SPI_CLOCK_IDX0           CLK_BUS_PERIPHERAL_2
#define DRV_SPI_BAUD_RATE_IDX0           13333333
#define DRV_SPI_BUFFER_TYPE_IDX0         DRV_SPI_BUFFER_TYPE_ENHANCED
#define DRV_SPI_CLOCK_MODE_IDX0          DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL
#define DRV_SPI_INPUT_PHASE_IDX0         SPI_INPUT_SAMPLING_PHASE_AT_END
#define DRV_SPI_TX_INT_SOURCE_IDX0       INT_SOURCE_SPI_1_TRANSMIT
#define DRV_SPI_RX_INT_SOURCE_IDX0       INT_SOURCE_SPI_1_RECEIVE
#define DRV_SPI_ERROR_INT_SOURCE_IDX0    INT_SOURCE_SPI_1_ERROR
#define DRV_SPI_INT_VECTOR_IDX0          INT_VECTOR_SPI1
#define DRV_SPI_INT_PRIORITY_IDX0        INT_PRIORITY_LEVEL1
#define DRV_SPI_INT_SUB_PRIORITY_IDX0    INT_SUBPRIORITY_LEVEL0
#define DRV_SPI_QUEUE_SIZE_IDX0          30
#define DRV_SPI_RESERVED_JOB_IDX0        1
#define DRV_SPI_TX_DMA_CHANNEL_IDX0      DMA_CHANNEL_1
#define DRV_SPI_TX_DMA_THRESHOLD_IDX0    16
#define DRV_SPI_RX_DMA_CHANNEL_IDX0      DMA_CHANNEL_0
#define DRV_SPI_RX_DMA_THRESHOLD_IDX0    16 Driver Library
```

Configuring the Library

Macros

	Name	Description
	DRV_ENC28J60_CLIENT_INSTANCES	Selects the maximum number of clients.
	DRV_ENC28J60_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.

Description

The configuration of the ENC28J60 Driver Library is based on the file `sys_config.h`.

This header file contains the configuration selection for the ENC28J60 Driver Library. Based on the selections made, the ENC28J60 Driver Library may support the selected features. These configuration settings will apply to all instances of the ENC28J60 Driver Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_ENC28J60_CLIENT_INSTANCES Macro

Selects the maximum number of clients.

File

drv_enc28j60_config_template.h

C

```
#define DRV_ENC28J60_CLIENT_INSTANCES 1
```

Description

enc28j60 maximum number of clients

This definition selects the maximum number of clients that the enc28j60 driver can support at run-time.

Remarks

Mandatory definition.

DRV_ENC28J60_INSTANCES_NUMBER Macro

Selects the maximum number of hardware instances that can be supported by the dynamic driver.

File

drv_enc28j60_config_template.h

C

```
#define DRV_ENC28J60_INSTANCES_NUMBER 1
```

Description

enc28j60 hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver.

Remarks

Mandatory definition.

Building the Library

This section lists the files that are available in the ENC28J60 Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/enc28j60`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source Folder Name	Description
`/drv_enc28j60.h`	This file provides the interface definitions of the ENC28J60 Driver.

Required File(s)

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

All of the required files listed in the following table are automatically loaded into the MPLAB X IDE project by the MHC.

Source Folder Name	Description
`/src/dynamic/drv_drv_enc28j60_api.c`	This file contains the API function implementations.
`/src/dynamic/drv_enc28j60_main_state.c`	This file contains the main state machine functions.
`/src/dynamic/drv_enc28j60_utils.c`	This file contains functions that are used throughout the driver.
`/src/dynamic/bus/spi/drv_enc28j60_spi_bus.c`	This file contains the functions to interface with the SPI bus.

/src/dynamic/closed_state/drv_enc28j60_closed_state.c	This file contains the functions for handling the driver closed state.
/src/dynamic/initialization_state/drv_enc28j60_configure_state.c	This file contains the functions for configuring the ENC hardware.
/src/dynamic/initialization_state/drv_enc28j60_detect_state.c	This file contains the functions for detecting the ENC hardware.
/src/dynamic/initialization_state/drv_enc28j60_initialization_state.c	This file contains the functions for the initialization state machine.
/src/dynamic/initialization_state/drv_enc28j60_reset_state.c	This file contains the functions for resetting the ENC hardware.
/src/dynamic/packet/drv_enc28j60_rx_packet.c	This file contains the functions for receiving a packet from the ENC hardware.
/src/dynamic/packet/drv_enc28j60_tx_packet.c	This file contains the functions for sending a packet to the ENC hardware.
/src/dynamic/running_state/drv_enc28j60_change_duplex_state.c	This file contains the functions for configuring the duplex mode of the ENC hardware.
/src/dynamic/running_state/drv_enc28j60_check_int_state.c	This file contains the functions for checking and processing the ENC hardware interrupts.
/src/dynamic/running_state/drv_enc28j60_check_status_state.c	This file contains the functions for checking the status of the ENC hardware.
/src/dynamic/running_state/drv_enc28j60_check_tx_status_state.c	This file contains the functions for checking the status of a transmitted packet.
/src/dynamic/running_state/drv_enc28j60_running_state.c	This file contains the functions for managing the running state machine.
/src/dynamic/running_state/drv_enc28j60_reset_rx_state.c	This file contains the functions for managing the RX state machine reset requirement during run-time.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source Folder Name	Description
N/A	No optional files exist for this library.

Module Dependencies

The ENC28J60 Driver Library depends on the following modules:

- SPI Driver Library
- TCP/IP Stack Library
- TCP/IP Stack MAC Driver Module

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_ENC28J60_Deinitialize	Deinitializes the ENC28J60 Driver Instance. **Implementation:** Dynamic
	DRV_ENC28J60_Initialize	Initializes the ENC28J60 Driver Instance, with the configuration data. **Implementation:** Dynamic
	DRV_ENC28J60_Process	Additional processing that happens outside the tasks function. **Implementation:** Dynamic
	DRV_ENC28J60_Reinitialize	Reinitializes the instance of the ENC28J60 driver. **Implementation:** Dynamic
	DRV_ENC28J60_SetMacCtrlInfo	This function sets the MAC control information for the driver. **Implementation:** Dynamic
	DRV_ENC28J60_StackInitialize	This function initializes the driver with a TCPIP_MAC_INIT object. **Implementation:** Dynamic

	DRV_ENC28J60_Tasks	Main task function for the driver. **Implementation:** Dynamic

b) Client Level Functions

	Name	Description
	DRV_ENC28J60_Close	Closes a client handle to the driver. **Implementation:** Dynamic
	DRV_ENC28J60_ConfigGet	Gets the current configuration. **Implementation:** Dynamic
	DRV_ENC28J60_LinkCheck	This function returns the status of the link. **Implementation:** Dynamic
	DRV_ENC28J60_Open	This function is called by the client to open a handle to a driver instance. **Implementation:** Dynamic
	DRV_ENC28J60_ParametersGet	Get the parameters of the device. **Implementation:** Dynamic
	DRV_ENC28J60_PowerMode	This function sets the power mode of the device. **Implementation:** Dynamic
	DRV_ENC28J60_RegisterStatisticsGet	Get the register statistics. **Implementation:** Dynamic
	DRV_ENC28J60_StatisticsGet	Retrieve the devices statistics. **Implementation:** Dynamic
	DRV_ENC28J60_Status	Gets the current status of the driver. **Implementation:** Dynamic

c) Receive Functions

	Name	Description
	DRV_ENC28J60_PacketRx	Receive a packet from the driver. **Implementation:** Dynamic
	DRV_ENC28J60_RxFilterHashTableEntrySet	This function adds an entry to the hash table. **Implementation:** Dynamic

d) Transmit Functions

	Name	Description
	DRV_ENC28J60_PacketTx	This function queues a packet for transmission. **Implementation:** Dynamic

e) Event Functions

	Name	Description
	DRV_ENC28J60_EventAcknowledge	Acknowledges an event. **Implementation:** Dynamic
	DRV_ENC28J60_EventMaskSet	Sets the event mask. **Implementation:** Dynamic
	DRV_ENC28J60_EventPendingGet	Gets the current events. **Implementation:** Dynamic

f) Data Types and Constants

	Name	Description
	_DRV_ENC28J60_Configuration	Defines the data required to initialize or reinitialize the ENC28J60 Driver.
	DRV_ENC28J60_Configuration	Defines the data required to initialize or reinitialize the ENC28J60 Driver.
	DRV_ENC28J60_MDIX_TYPE	Defines the enumeration for controlling the MDIX select.
	DRV_ENC28J60_MACObject	ENC28J60 External MAC Virtualization Table

Description

This section describes the Application Programming Interface (API) functions of the ENC28J60 Driver Library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_ENC28J60_Deinitialize Function

Deinitializes the ENC28J60 Driver Instance.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
void DRV_ENC28J60_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

ENC28J60 Deinitialization

This function deallocates any resources allocated by the initialization function.

Preconditions

The driver had to be successfully initialized with DRV_ENC28J60_Initialize.

Parameters

Parameters	Description
Object	the valid object returned from DRV_ENC28J60_Initialize

DRV_ENC28J60_Initialize Function

Initializes the ENC28J60 Driver Instance, with the configuration data.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
SYS_MODULE_OBJ DRV_ENC28J60_Initialize(SYS_MODULE_INDEX index, SYS_MODULE_INIT * init);
```

Returns

- Valid handle to the driver instance - If successful
- SYS_MODULE_OBJ_INVALID - If unsuccessful

Description

ENC28J60 Initialization

This function initializes the ENC28J60 Driver with configuration data passed into it by either the system_init function or by the DRV_ENC28J60_StackInitialize function. Calling this function alone is not enough to initialize the driver, DRV_ENC28J60_SetMacCtrlInfo must be called with valid data before the driver is ready to be opened.

Preconditions

None.

Parameters

Parameters	Description
index	This is the index of the driver instance to be initialized. The definition DRV_ENC28J60_NUM_DRV_INSTANCES controls how many instances are available.
init	This is a pointer to a DRV_ENC28J60_CONFIG structure.

DRV_ENC28J60_Process Function

Additional processing that happens outside the tasks function.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_RES DRV_ENC28J60_Process(DRV_HANDLE hMac);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENC28J60 Process

This function does additional processing that is not done inside the tasks function.

Remarks

This function does nothing in the first release.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

DRV_ENC28J60_Reinitialize Function

Reinitializes the instance of the ENC28J60 driver.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
void DRV_ENC28J60_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None

Description

ENC28J60 Reinitialization

This function will deinitialize and initialize the driver instance. As with DRV_ENC28J60_Initialize DRV_ENC28J60_SetMacCtrlInfo must be called for the driver to be useful.

Remarks

This function is not planned to be implemented for the first release.

Preconditions

The driver had to be successfully initialized with DRV_ENC28J60_Initialize.

DRV_ENC28J60_SetMacCtrlInfo Function

This function sets the MAC control information for the driver.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
void DRV_ENC28J60_SetMacCtrlInfo(SYS_MODULE_OBJ object, TCPIP_MAC_MODULE_CTRL * init);
```

Returns

None.

Description

ENC28J60 Set MAC Control Information

This function is used to pass in the TCPIP_MAC_CONTROL_INIT information that is used for allocation and deallocation of memory, event signaling, etc. This function is needed to be called so that the driver can enter initialization state when the tasks function is called.

Preconditions

The driver had to be successfully initialized with ENC28J60_Initialize.

DRV_ENC28J60_StackInitialize Function

This function initializes the driver with a TCPIP_MAC_INIT object.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
SYS_MODULE_OBJ DRV_ENC28J60_StackInitialize(SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

Returns a valid handle to the driver instance - If successful SYS_MODULE_OBJ_INVALID - If unsuccessful

Description

ENC28J60 Stack Initialization

This function is used by the TCP/IP stack to fully initialize the driver with both the ENC28J60 specific configuration and the MAC control information. With this function there is no need to call DRV_ENC28J60_SetMacCtrlInfo.

Preconditions

None.

Parameters

Parameters	Description
index	This is the index of the driver instance to be initialized. The definition DRV_ENC28J60_NUM_DRV_INSTANCES controls how many instances are available.
init	This is a pointer to a TCPIP_MAC_INIT structure.

DRV_ENC28J60_Tasks Function

Main task function for the driver.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
void DRV_ENC28J60_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

ENC28J60 Tasks

This function will execute the main state machine for the ENC28J60 driver.

Preconditions

The driver had to be successfully initialized with DRV_ENC28J60_Initialize.

Parameters

Parameters	Description
object	The object valid passed back to DRV_ENC28J60_Initialize

b) Client Level Functions

DRV_ENC28J60_Close Function

Closes a client handle to the driver.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
void DRV_ENC28J60_Close(DRV_HANDLE handle);
```

Returns

None.

Description

ENC28J60 Close

This function closes a handle to the driver. If it is the last client open, the driver will send an RX Disable command to the ENC hardware and move to the closed state.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
handle	The successfully opened handle

DRV_ENC28J60_ConfigGet Function

Gets the current configuration.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
size_t DRV_ENC28J60_ConfigGet(DRV_HANDLE hMac, void* configBuff, size_t buffSize, size_t* pConfigSize);
```

Returns

Number of bytes copied to the buffer

Description

ENC28J60 Get Configuration

Gets the current configuration.

Remarks

This function does nothing in the first release.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
configBuff	location to copy the configuration too
buffSize	buffer size
pConfigSize	configuration size needed

DRV_ENC28J60_LinkCheck Function

This function returns the status of the link.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
bool DRV_ENC28J60_LinkCheck(DRV_HANDLE hMac);
```

Returns

- true - if the link is active
- false - all other times

Description

ENC28J60 Link Check

This function checks the status of the link and returns it to the caller.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

DRV_ENC28J60_Open Function

This function is called by the client to open a handle to a driver instance.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
DRV_HANDLE DRV_ENC28J60_Open(SYS_MODULE_INDEX index, DRV_IO_INTENT intent);
```

Returns

Returns a valid handle - If successful INVALID_HANDLE - If unsuccessful

Description

ENC28J60 Open

The client will call this function to open a handle to the driver. When the first instance is opened than the driver will send the RX enabled command to the ENC hardware.

Preconditions

The driver had to be successfully initialized with DRV_ENC28J60_Initialize.

Parameters

Parameters	Description
index	This is the index of the driver instance to be initialized. The definition DRV_ENC28J60_NUM_DRV_INSTANCES controls how many instances are available.

intent	The intent to use when opening the driver. Only exclusive is supported

DRV_ENC28J60_ParametersGet Function

Get the parameters of the device.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_RES DRV_ENC28J60_ParametersGet(DRV_HANDLE hMac, TCPIP_MAC_PARAMETERS* pMacParams);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OK - if the hMac is valid

Description

ENC28J60 Get Parameters

Get the parameters of the device, which includes that it is an Ethernet device and what it's MAC address is. Users of the ENC28J60 must generate a unique MAC address for each controller used.

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pMacParams	pointer to put the parameters

DRV_ENC28J60_PowerMode Function

This function sets the power mode of the device.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
bool DRV_ENC28J60_PowerMode(DRV_HANDLE hMac, TCPIP_MAC_POWER_MODE pwrMode);
```

Returns

- false - This functionality is not supported in this version of the driver

Description

ENC28J60 Power Mode

This function sets the power mode of the ENC28J60.

Remarks

This functionality is not implemented in the first release.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pwrMode	the power mode to set

DRV_ENC28J60_RegisterStatisticsGet Function

Get the register statistics.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_RES DRV_ENC28J60_RegisterStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_STATISTICS_REG_ENTRY*
pRegEntries, int nEntries, int* pHwEntries);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENC28J60 Get Register Statistics

Get the device specific statistics.

Remarks

Statistics are not planned for the first release

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pRegEntries	
nEntries	
pHwEntries	

DRV_ENC28J60_StatisticsGet Function

Retrieve the devices statistics.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_RES DRV_ENC28J60_StatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_RX_STATISTICS* pRxStatistics,
TCPIP_MAC_TX_STATISTICS* pTxStatistics);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENC28J60 Get Statistics

Get the current statistics stored in the driver.

Remarks

Statistics are not planned for the first release.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

DRV_ENC28J60_Status Function

Gets the current status of the driver.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
SYS_STATUS DRV_ENC28J60_Status(SYS_MODULE_OBJ obect);
```

Returns

- SYS_STATUS_ERROR - if an invalid handle has been passed in
- SYS_STATUS_UNINITIALIZED - if the driver has not completed initialization
- SYS_STATUS_BUSY - if the driver is closing and moving to the closed state
- SYS_STATUS_READY - if the driver is ready for client commands

Description

ENC28J60 Status

This function will get the status of the driver instance.

Preconditions

The driver had to be successfully initialized with DRV_ENC28J60_Initialize().

Parameters

Parameters	Description
object	The object valid passed back to DRV_ENC28J60_Initialize()

c) Receive Functions

DRV_ENC28J60_PacketRx Function

Receive a packet from the driver.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_PACKET* DRV_ENC28J60_PacketRx(DRV_HANDLE hMac, TCPIP_MAC_RES* pRes, const
TCPIP_MAC_PACKET_RX_STAT** ppPktStat);
```

Returns

- Pointer to a valid packet - if successful
- NULL - if unsuccessful

Description

ENC28J60 Receive Packet

This function retrieves a packet from the driver. The packet needs to be acknowledged with the linked acknowledge function so it can be reused.

Remarks

ppPktStat is ignored in the first release.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pRes	the result of the operation
ppPktStat	pointer to the receive statistics

DRV_ENC28J60_RxFilterHashTableEntrySet Function

This function adds an entry to the hash table.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_RES DRV_ENC28J60_RxFilterHashTableEntrySet(DRV_HANDLE hMac, TCPIP_MAC_ADDR* DestMACAddr);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENC28J60 Receive Filter Hash Table Entry Set

This function adds to the MAC's hash table for hash table matching.

Remarks

This functionality is not implemented in the first release.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
DestMACAddr	MAC address to add to the hash table

d) Transmit Functions

DRV_ENC28J60_PacketTx Function

This function queues a packet for transmission.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_RES DRV_ENC28J60_PacketTx(DRV_HANDLE hMac, TCPIP_MAC_PACKET * ptrPacket);
```

Returns

- TCPIP_MAC_RES_OP_ERR - if the client handle is invalid
- TCPIP_MAC_RES_IS_BUSY - if the driver is not in the run state
- TCPIP_MAC_RES_QUEUE_TX_FULL - if there are no free descriptors
- TCPIP_MAC_RES_OK - on successful queuing of the packet

Description

ENC28J60 Packet Transmit

This function will take a packet and add it to the queue for transmission. When the packet has finished transmitting the driver will call the packets acknowledge function. When that acknowledge function is complete the driver will forget about the packet.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
ptrPacket	pointer to the packet

e) Event Functions

DRV_ENC28J60_EventAcknowledge Function

Acknowledges an event.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
bool DRV_ENC28J60_EventAcknowledge(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents);
```

Returns

- true - if successful
- false - if not successful

Description

ENC28J60 Acknowledge Event

This function acknowledges an event.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
macEvents	the events to acknowledge

DRV_ENC28J60_EventMaskSet Function

Sets the event mask.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
bool DRV_ENC28J60_EventMaskSet(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents, bool enable);
```

Returns

- true - if the mask could be set
- false - if the mast could not be set

Description

ENC28J60 Set Event Mask

Sets the event mask to what is passed in.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
macEvents	the mask to enable or disable
enable	to enable or disable events

DRV_ENC28J60_EventPendingGet Function

Gets the current events.

Implementation: Dynamic

File

drv_enc28j60.h

C

```
TCPIP_MAC_EVENT DRV_ENC28J60_EventPendingGet(DRV_HANDLE hMac);
```

Returns

- TCPIP_MAC_EV_NONE - Returned on an error
- List of events - Returned on event other than an error

Description

ENC28J60 Get Events

This function gets the current events.

Preconditions

The client had to be successfully opened with DRV_ENC28J60_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

f) Data Types and Constants

DRV_ENC28J60_Configuration Structure

Defines the data required to initialize or reinitialize the ENC28J60 Driver.

File

drv_enc28j60.h

C

```
typedef struct _DRV_ENC28J60_Configuration {
  uint16_t txDescriptors;
  uint16_t rxDescriptors;
  uint16_t rxDescBufferSize;
  SYS_MODULE_INDEX spiDrvIndex;
  uint32_t spiBps;
  uint16_t rxBufferSize;
  uint16_t maxFrameSize;
  PORTS_MODULE_ID spiSSPortModule;
  PORTS_CHANNEL spiSSPortChannel;
```

```
  PORTS_BIT_POS spiSSPortPin;
  bool intEnable;
  PORTS_MODULE_ID intPortModule;
  PORTS_CHANNEL intPortChannel;
  PORTS_BIT_POS intPortPin;
  DRV_ENC28J60_MDIX_TYPE mdixControl;
  PORTS_MODULE_ID mdixPortModule;
  PORTS_CHANNEL mdixPortChannel;
  PORTS_BIT_POS mdixPortPin;
} DRV_ENC28J60_Configuration;
```

Members

Members	Description
uint16_t txDescriptors;	Number of TX Descriptors to Allocate
uint16_t rxDescriptors;	Number of RX Descriptors to Allocate
uint16_t rxDescBufferSize;	Size of the buffer each RX Descriptor will use. Make sure its not smaller than maxFrameSize
SYS_MODULE_INDEX spiDrvIndex;	Index of the SPI driver to use
uint32_t spiBps;	Bus speed to use for the SPI interface. Section 1.0 of the ENC28J60 data sheets says the maximum is 20000000 Hz. It is not recommended to go above this value.
uint16_t rxBufferSize;	The ENC28J60 hardware has a 8 k dram. rxBufferSize defines how much of that memory is used by the rxBuffer
uint16_t maxFrameSize;	The maximum frame size to be supported by the hardware. 1536 is the default
PORTS_MODULE_ID spiSSPortModule;	Port Module of the GPIO pin hooked up to the CS/SS pin of the ENC28J60
PORTS_CHANNEL spiSSPortChannel;	Port Channel of the GPIO pin hooked up to the CS/SS pin of the ENC28J60
PORTS_BIT_POS spiSSPortPin;	Pin position of the GPIO pin hooked up to the CS/SS pin of the ENC28J60
bool intEnable;	Use Interrupts or not.
PORTS_MODULE_ID intPortModule;	Port Module of the GPIO pin hooked up to the INT pin of the ENC28J60
PORTS_CHANNEL intPortChannel;	Port Channel of the GPIO pin hooked up to the INT pin of the ENC28J60
PORTS_BIT_POS intPortPin;	Pin Position of the GPIO pin hooked up to the INT pin of the ENC28J60
DRV_ENC28J60_MDIX_TYPE mdixControl;	To select the control type of the MDIX. This is only needed for hooking up to switches that don't have auto-mdix.
PORTS_MODULE_ID mdixPortModule;	Port Module of the GPIO pin hooked up to the MDIX select pin
PORTS_CHANNEL mdixPortChannel;	Port Channel of the GPIO pin hooked up to the MDIX select pin
PORTS_BIT_POS mdixPortPin;	Pin Position of the GPIO pin hooked up to the MDIX select pin

Description

ENC28J60 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the ENC28J60 driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

DRV_ENC28J60_MDIX_TYPE Enumeration

Defines the enumeration for controlling the MDIX select.

File

drv_enc28j60.h

C

```
typedef enum {
  DRV_ENC28J60_NO_CONTROL = 0,
  DRV_ENC28J60_NORMAL,
  DRV_ENC28J60_REVERSE = 0
} DRV_ENC28J60_MDIX_TYPE;
```

Members

Members	Description
DRV_ENC28J60_NO_CONTROL = 0	No Control
DRV_ENC28J60_NORMAL	Normal MDIX
DRV_ENC28J60_REVERSE = 0	Reverse MDIX

Description

ENC28J60 Driver MDIX Control type

This type defines the enumeration for controlling the MDIX select.

Remarks

None.

DRV_ENC28J60_MACObject Variable

File

drv_enc28j60.h

C

```
const TCPIP_MAC_OBJECT DRV_ENC28J60_MACObject;
```

Description

ENC28J60 External MAC Virtualization Table

Files

Files

Name	Description
drv_enc28j60.h	ENC28J60 Driver interface definition.
drv_enc28j60_config_template.h	enc28j60 Driver configuration definitions template.

Description

drv_enc28j60.h

ENC28J60 Driver interface definition.

Enumerations

	Name	Description
	DRV_ENC28J60_MDIX_TYPE	Defines the enumeration for controlling the MDIX select.

Functions

	Name	Description
	DRV_ENC28J60_Close	Closes a client handle to the driver. **Implementation:** Dynamic
	DRV_ENC28J60_ConfigGet	Gets the current configuration. **Implementation:** Dynamic
	DRV_ENC28J60_Deinitialize	Deinitializes the ENC28J60 Driver Instance. **Implementation:** Dynamic
	DRV_ENC28J60_EventAcknowledge	Acknowledges an event. **Implementation:** Dynamic
	DRV_ENC28J60_EventMaskSet	Sets the event mask. **Implementation:** Dynamic
	DRV_ENC28J60_EventPendingGet	Gets the current events. **Implementation:** Dynamic
	DRV_ENC28J60_Initialize	Initializes the ENC28J60 Driver Instance, with the configuration data. **Implementation:** Dynamic
	DRV_ENC28J60_LinkCheck	This function returns the status of the link. **Implementation:** Dynamic
	DRV_ENC28J60_Open	This function is called by the client to open a handle to a driver instance. **Implementation:** Dynamic

	DRV_ENC28J60_PacketRx	Receive a packet from the driver. **Implementation:** Dynamic
	DRV_ENC28J60_PacketTx	This function queues a packet for transmission. **Implementation:** Dynamic
	DRV_ENC28J60_ParametersGet	Get the parameters of the device. **Implementation:** Dynamic
	DRV_ENC28J60_PowerMode	This function sets the power mode of the device. **Implementation:** Dynamic
	DRV_ENC28J60_Process	Additional processing that happens outside the tasks function. **Implementation:** Dynamic
	DRV_ENC28J60_RegisterStatisticsGet	Get the register statistics. **Implementation:** Dynamic
	DRV_ENC28J60_Reinitialize	Reinitializes the instance of the ENC28J60 driver. **Implementation:** Dynamic
	DRV_ENC28J60_RxFilterHashTableEntrySet	This function adds an entry to the hash table. **Implementation:** Dynamic
	DRV_ENC28J60_SetMacCtrlInfo	This function sets the MAC control information for the driver. **Implementation:** Dynamic
	DRV_ENC28J60_StackInitialize	This function initializes the driver with a TCPIP_MAC_INIT object. **Implementation:** Dynamic
	DRV_ENC28J60_StatisticsGet	Retrieve the devices statistics. **Implementation:** Dynamic
	DRV_ENC28J60_Status	Gets the current status of the driver. **Implementation:** Dynamic
	DRV_ENC28J60_Tasks	Main task function for the driver. **Implementation:** Dynamic

Structures

	Name	Description
	_DRV_ENC28J60_Configuration	Defines the data required to initialize or reinitialize the ENC28J60 Driver.
	DRV_ENC28J60_Configuration	Defines the data required to initialize or reinitialize the ENC28J60 Driver.

Variables

	Name	Description
	DRV_ENC28J60_MACObject	ENC28J60 External MAC Virtualization Table

Description

ENC28J60 Driver Public Interface

This file defines the interface definition for the ENC28J60 Driver.

File Name

drv_enc28j60.h

Company

Microchip Technology Inc.

drv_enc28j60_config_template.h

enc28j60 Driver configuration definitions template.

Macros

	Name	Description
	DRV_ENC28J60_CLIENT_INSTANCES	Selects the maximum number of clients.
	DRV_ENC28J60_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.

Description

enc28j60 Driver Configuration Definitions for the Template Version

These definitions statically define the driver's mode of operation.

File Name

drv_enc28j60_config_template.h

Company

Microchip Technology Inc.

ENCx24J600 Driver Library Help

This section provides information on the ENCx24J600 Driver Library.

Introduction

This library provides a driver-level abstraction of the ENCx24J600 Ethernet MAC that can be connected to the PIC32. The driver implements the virtual MAC driver model that the MPLAB Harmony TCP/IP Stack requires. Please see the TCP/IP Stack Library MAC Driver Module for details.

The "Host-To-Network"_layer of a TCP/IP stack organization covers the Data Link and Physical Layers of the standard OSI stack. The Ethernet Controller provides the Data Link or Media Access Control Layer, in addition to other functions discussed in this section.

Description

The ENCx24J600 External MAC is an external module to the PIC32 that is connected through a SPI or PSP interface. This driver interfaces with the SPI driver to communicate with the external device to implement a complete Ethernet node in a system.

The following are some of the key features of this module:

- Supports 10/100 Ethernet
- Full-Duplex and Half-Duplex operation
- Broadcast, Multicast and Unicast packets
- Manual and automatic flow control
- Supports Auto-MDIX
- Fully configurable interrupts
- Configurable receive packet filtering using:
- 64-bit Hash Table
- 64-byte Pattern Match
- Magic Packet™ Filtering
- Runt Packet Detection and Filtering
- Supports Packet Payload Checksum calculation
- CRC Check
- Supports SPI interface

Using the Library

This topic describes the basic architecture and functionality of the Ethernet MAC driver and is meant for advanced users or TCP/IP stack driver developers.

Description

The user of this driver is the MPLAB Harmony TCP/IP stack. This Ethernet driver is not intended as a system-wide driver that the application or other system modules may use. It is intended for the sole use of the MPLAB Harmony TCP/IP stack and implements the virtual MAC model required by the stack.

Interface Header File: `drv_encx24j600.h`

The interface to the ENCx24J600 Driver Library is defined in the `drv_encx24j600.h` header file. Any C language source (`.c`) file that uses the ENCx24J600 Driver Library should include `drv_encx24j600.h`.

Library File: The ENCx24J600 Driver Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

The ENCx24J600 Driver Library provides the low-level abstraction of the communications protocol to communicate to the ENCx24J600 external MAC though the SPI peripheral on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the ENCx24J600 Driver Library interface.

Description

The ENCx24J600 Driver library has several different layers to it, as illustrated in the following figure. The interface layer has two main sections that

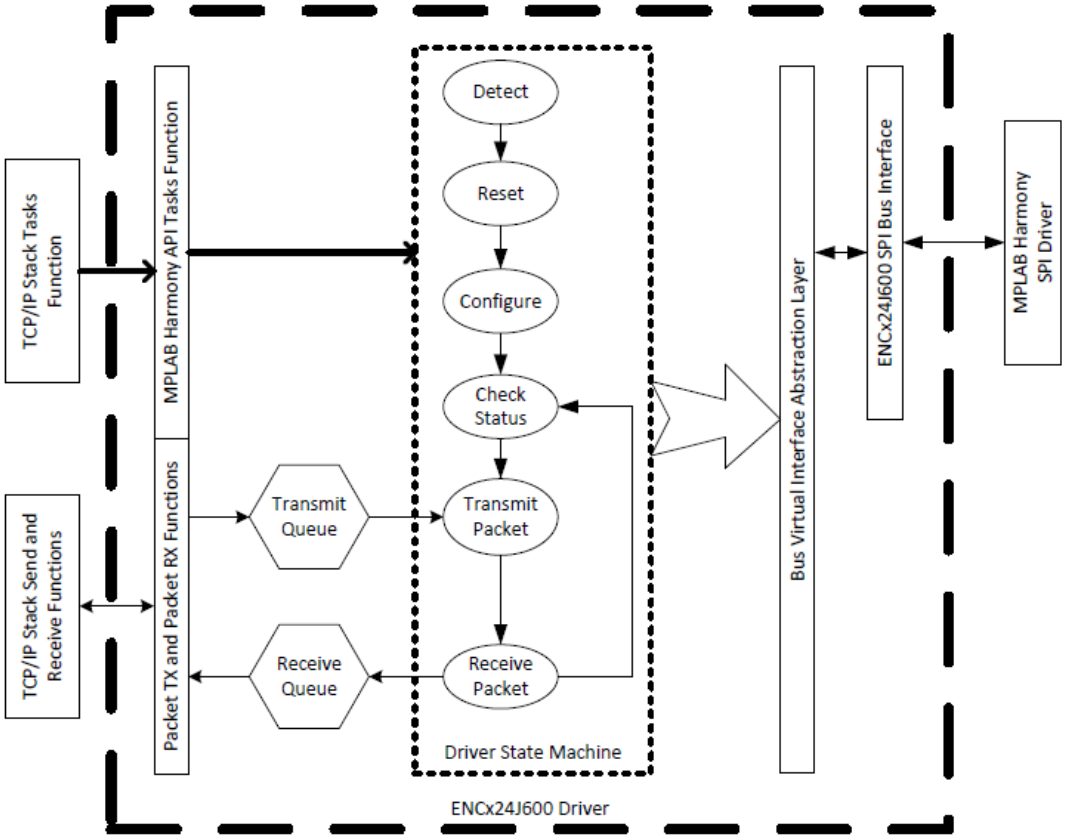
are used the most often: The Tasks function, and the TCP/IP Send and Receive functions.

The Tasks function manages the internal state machine which detects, resets, and then configures the ENCx24J600 External MAC. It also handles the monitoring of the hardware status, sending and receiving packets.

The TCP/IP Send and Receive functions interact with the RAM-based queue of packets that are queued to send and packets that have been queued waiting for pick-up by the stack.

The main state machine does not interface directly to the SPI bus, but instead, interfaces to a virtual bus abstraction layer that allows for the replacement of the specific underlying bus implementation.

Abstraction Model



Library Overview

Refer to the section Driver Overview for how the driver operates in a system.

The library interface routines are divided into various sub-sections, each sub-section addresses one of the blocks or the overall operation of the ENCx24J600 Driver Library.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Data Transfer Functions	Provides data transfer functions available in the configuration.
Status Functions	Provides status functions.
Miscellaneous Functions	Provides miscellaneous driver functions.

How the Library Works

The library provides interfaces to support the TCP/IP virtual MAC interface.

Configuring the SPI Driver

This section describes the configuration settings for the ENCx24J600 Driver Library.

Description

Configuration

The ENC hardware requires a specific configuration of the SPI driver to work correctly. Inside the MHC SPI Driver configuration be sure to select:

- SPI clock rate of 14000000 or less. With a PB clock of 80 MHz, 13333333 is the clock rate.
- Clock mode of DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL
- Input phase of SPI_INPUT_SAMPLING_PHASE_AT_END

Recommended Settings

- Interrupt Driver mode
- Enhanced Buffer mode
- DMA mode enabled:
- DMA block transfer size of at least 1600 bytes
- Size of DMA buffer for dummy data of at least 1600 bytes
- Ensure when setting up DMA in interrupt mode that the DMA interrupts are a higher priority than the SPI Driver interrupt

Example:

```
/*** SPI Driver Static Allocation Options ***/
#define DRV_SPI_INSTANCES_NUMBER           1
#define DRV_SPI_CLIENTS_NUMBER             1
#define DRV_SPI_ELEMENTS_PER_QUEUE         30

/*** SPI Driver DMA Options ***/
#define DRV_SPI_DMA_TXFER_SIZE             2048
#define DRV_SPI_DMA_DUMMY_BUFFER_SIZE      2048

/* SPI Driver Instance 0 Configuration */
#define DRV_SPI_SPI_ID_IDX0              SPI_ID_1
#define DRV_SPI_TASK_MODE_IDX0           DRV_SPI_TASK_MODE_ISR
#define DRV_SPI_SPI_MODE_IDX0            DRV_SPI_MODE_MASTER
#define DRV_SPI_ALLOW_IDLE_RUN_IDX0      false
#define DRV_SPI_SPI_PROTOCOL_TYPE_IDX0   DRV_SPI_PROTOCOL_TYPE_STANDARD
#define DRV_SPI_SPI_PROTOCOL_TYPE_IDX0   DRV_SPI_PROTOCOL_TYPE_STANDARD
#define DRV_SPI_COMM_WIDTH_IDX0          SPI_COMMUNICATION_WIDTH_8BITS
#define DRV_SPI_SPI_CLOCK_IDX0           CLK_BUS_PERIPHERAL_2
#define DRV_SPI_BAUD_RATE_IDX0           13333333
#define DRV_SPI_BUFFER_TYPE_IDX0         DRV_SPI_BUFFER_TYPE_ENHANCED
#define DRV_SPI_CLOCK_MODE_IDX0          DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL
#define DRV_SPI_INPUT_PHASE_IDX0         SPI_INPUT_SAMPLING_PHASE_AT_END
#define DRV_SPI_TX_INT_SOURCE_IDX0       INT_SOURCE_SPI_1_TRANSMIT
#define DRV_SPI_RX_INT_SOURCE_IDX0       INT_SOURCE_SPI_1_RECEIVE
#define DRV_SPI_ERROR_INT_SOURCE_IDX0    INT_SOURCE_SPI_1_ERROR
#define DRV_SPI_INT_VECTOR_IDX0          INT_VECTOR_SPI1
#define DRV_SPI_INT_PRIORITY_IDX0        INT_PRIORITY_LEVEL1
#define DRV_SPI_INT_SUB_PRIORITY_IDX0    INT_SUBPRIORITY_LEVEL0
#define DRV_SPI_QUEUE_SIZE_IDX0          30
#define DRV_SPI_RESERVED_JOB_IDX0        1
#define DRV_SPI_TX_DMA_CHANNEL_IDX0      DMA_CHANNEL_1
#define DRV_SPI_TX_DMA_THRESHOLD_IDX0    16
#define DRV_SPI_RX_DMA_CHANNEL_IDX0      DMA_CHANNEL_0
#define DRV_SPI_RX_DMA_THRESHOLD_IDX0    16 Driver Library
```

Configuring the Library

The configuration of the ENCx24J600 Driver Library is based on the file `sys_config.h`.

This header file contains the configuration selection for the ENCX24J600 Driver Library. Based on the selections made, the ENCx24J600 Driver Library may support the selected features. These configuration settings will apply to all instances of the ENCx24J600 Driver Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the ENCx24J600 Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/encx24j600`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source Folder Name	Description
/drv_encx24j600.h	This file provides the interface definitions of the ENCx24J600 Driver.

Required File(s)

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

Source Folder Name	Description
/src/dynamic/drv_drv_encx24J600_api.c	This file contains the API function implementations.
/src/dynamic/drv_encx24J600_main_state.c	This file contains the main state machine functions.
/src/dynamic/drv_encx24J600_utils.c	This file contains functions that are used throughout the driver.
/src/dynamic/bus/spi/drv_encx24J600_spi_bus.c	This file contains the functions to interface with the SPI bus.
/src/dynamic/closed_state/drv_encx24J600_closed_state.c	This file contains the functions for handling the driver closed state.
/src/dynamic/initialization_state/drv_encx24J600_configure_state.c	This file contains the functions for configuring the ENC hardware.
/src/dynamic/initialization_state/drv_encx24J600_detect_state.c	This file contains the functions for detecting the ENC hardware.
/src/dynamic/initialization_state/drv_encx24J600_initialization_state.c	This file contains the functions for the initialization state machine.
/src/dynamic/initialization_state/drv_encx24J600_reset_state.c	This file contains the functions for resetting the ENC hardware.
/src/dynamic/packet/drv_encx24J600_rx_packet.c	This file contains the functions for receiving a packet from the ENC hardware.
/src/dynamic/packet/drv_encx24J600_tx_packet.c	This file contains the functions for sending a packet to the ENC hardware.
/src/dynamic/running_state/drv_encx24J600_change_duplex_state.c	This file contains the functions for configuring the duplex mode of the ENC hardware.
/src/dynamic/running_state/drv_encx24J600_check_int_state.c	This file contains the functions for checking and processing the ENC hardware interrupts.
/src/dynamic/running_state/drv_encx24J600_check_status_state.c	This file contains the functions for checking the status of the ENC hardware.
/src/dynamic/running_state/drv_encx24J600_check_tx_status_state.c	This file contains the functions for checking the status of a transmitted packet.
/src/dynamic/running_state/drv_encx24J600_running_state.c	This file contains the functions for managing the running state machine.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source Folder Name	Description
N/A	No optional files exist for this library.

Module Dependencies

The ENCx24J600 Driver Library depends on the following modules:

- SPI Driver Library
- TCP/IP Stack Library
- TCP/IP Stack MAC Driver Module

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_ENCX24J600_Deinitialize	Deinitializes the ENCx24J600 Driver Instance. **Implementation:** Dynamic
	DRV_ENCX24J600_Initialize	Initializes the ENCx24J600 Driver Instance, with the configuration data. **Implementation:** Dynamic
	DRV_ENCX24J600_Reinitialize	Reinitializes the instance of the ENCX24J600 driver. **Implementation:** Dynamic
	DRV_ENCX24J600_Tasks	Main task function for the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_SetMacCtrlInfo	This function sets the MAC control information for the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_StackInitialize	This function initializes the driver with a TCPIP_MAC_INIT object. **Implementation:** Dynamic
	DRV_ENCX24J600_Process	Additional processing that happens outside the tasks function. **Implementation:** Dynamic

b) Client Level Functions

	Name	Description
	DRV_ENCX24J600_Close	Closes a client handle to the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_ConfigGet	Gets the current configuration. **Implementation:** Dynamic
	DRV_ENCX24J600_LinkCheck	This function returns the status of the link. **Implementation:** Dynamic
	DRV_ENCX24J600_Open	This function is called by the client to open a handle to a driver instance. **Implementation:** Dynamic
	DRV_ENCX24J600_ParametersGet	Get the parameters of the device. **Implementation:** Dynamic
	DRV_ENCX24J600_PowerMode	This function sets the power mode of the device. **Implementation:** Dynamic
	DRV_ENCX24J600_RegisterStatisticsGet	Get the register statistics. **Implementation:** Dynamic
	DRV_ENCX24J600_StatisticsGet	Retrieve the devices statistics. **Implementation:** Dynamic
	DRV_ENCX24J600_Status	Gets the current status of the driver. **Implementation:** Dynamic

c) Receive Functions

	Name	Description
	DRV_ENCX24J600_PacketRx	Receive a packet from the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_RxFilterHashTableEntrySet	This function adds an entry to the hash table. **Implementation:** Dynamic

d) Transmit Functions

	Name	Description
	DRV_ENCX24J600_PacketTx	This function queues a packet for transmission. **Implementation:** Dynamic

e) Event Functions

	Name	Description
	DRV_ENCX24J600_EventAcknowledge	Acknowledges an event. **Implementation:** Dynamic
	DRV_ENCX24J600_EventMaskSet	Sets the event mask. **Implementation:** Dynamic
	DRV_ENCX24J600_EventPendingGet	Gets the current events. **Implementation:** Dynamic

f) Data Types and Constants

	Name	Description
	_DRV_ENCX24J600_Configuration	Defines the data required to initialize or reinitialize the ENCX24J600 Driver.
	DRV_ENCX24J600_Configuration	Defines the data required to initialize or reinitialize the ENCX24J600 Driver.
	DRV_ENCX24J600_MDIX_TYPE	Defines the enumeration for controlling the MDIX select.

Description

This section describes the Application Programming Interface (API) functions of the ENCx24J600 Driver Library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_ENCX24J600_Deinitialize Function

Deinitializes the ENCx24J600 Driver Instance.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
void DRV_ENCX24J600_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

ENCX24J600 Deinitialization

This function deallocates any resources allocated by the initialization function.

Preconditions

The driver had to be successfully initialized with DRV_ENCX24J600_Initialize.

Parameters

Parameters	Description
Object	the valid object returned from DRV_ENCX24J600_Initialize

DRV_ENCX24J600_Initialize Function

Initializes the ENCx24J600 Driver Instance, with the configuration data.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
SYS_MODULE_OBJ DRV_ENCX24J600_Initialize(SYS_MODULE_INDEX index, SYS_MODULE_INIT * init);
```

Returns

- Valid handle to the driver instance - If successful
- SYS_MODULE_OBJ_INVALID - If unsuccessful

Description

ENCX24J600 Initialization

This function initializes the ENCx24J600 Driver with configuration data passed into it by either the system_init function or by the DRV_ENCX24J600_StackInitialize function. Calling this function alone is not enough to initialize the driver, DRV_ENCX24J600_SetMacCtrlInfo must be called with valid data before the driver is ready to be opened.

Preconditions

None.

Parameters

Parameters	Description
index	This is the index of the driver instance to be initialized. The definition DRV_ENCX24J600_NUM_DRV_INSTANCES controls how many instances are available.
init	This is a pointer to a DRV_ENX24J600_CONFIG structure.

DRV_ENCX24J600_Reinitialize Function

Reinitializes the instance of the ENCX24J600 driver.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
void DRV_ENCX24J600_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None

Description

ENCX24J600 Reinitialization

This function will deinitialize and initialize the driver instance. As with DRV_ENCX24J600_Initialize DRV_ENCX24J600_SetMacCtrlInfo must be called for the driver to be useful.

Remarks

This function is not planned to be implemented for the first release.

Preconditions

The driver had to be successfully initialized with DRV_ENCX24J600_Initialize.

DRV_ENCX24J600_Tasks Function

Main task function for the driver.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
void DRV_ENCX24J600_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

ENCX24J600 Tasks

This function will execute the main state machine for the ENCX24J600 driver.

Preconditions

The driver had to be successfully initialized with DRV_ENCX24J600_Initialize.

Parameters

Parameters	Description
object	The object valid passed back to DRV_ENCX24J600_Initialize

DRV_ENCX24J600_SetMacCtrlInfo Function

This function sets the MAC control information for the driver.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
void DRV_ENCX24J600_SetMacCtrlInfo(SYS_MODULE_OBJ object, TCPIP_MAC_MODULE_CTRL * init);
```

Returns

None.

Description

ENCX24J600 Set MAC Control Information

This function is used to pass in the TCPIP_MAC_CONTROL_INIT information that is used for allocation and deallocation of memory, event signaling, etc. This function is needed to be called so that the driver can enter initialization state when the tasks function is called.

Preconditions

The driver had to be successfully initialized with DRV_ENCX24J600_Initialize.

DRV_ENCX24J600_StackInitialize Function

This function initializes the driver with a TCPIP_MAC_INIT object.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
SYS_MODULE_OBJ DRV_ENCX24J600_StackInitialize(SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

Returns a valid handle to the driver instance - If successful SYS_MODULE_OBJ_INVALID - If unsuccessful

Description

ENCX24J600 Stack Initialization

This function is used by the TCP/IP stack to fully initialize the driver with both the ENCX24J600 specific configuration and the MAC control information. With this function there is no need to call DRV_ENCX24J600_SetMacCtrlInfo.

Preconditions

None.

Parameters

Parameters	Description
index	This is the index of the driver instance to be initialized. The definition DRV_ENCX24J600_NUM_DRV_INSTANCES controls how many instances are available.
init	This is a pointer to a TCPIP_MAC_INIT structure.

DRV_ENCX24J600_Process Function

Additional processing that happens outside the tasks function.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_RES DRV_ENCX24J600_Process(DRV_HANDLE hMac);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENCX24J600 Process

This function does additional processing that is not done inside the tasks function.

Remarks

This function does nothing in the first release.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

b) Client Level Functions

DRV_ENCX24J600_Close Function

Closes a client handle to the driver.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
void DRV_ENCX24J600_Close(DRV_HANDLE handle);
```

Returns

None.

Description

ENCX24J600 Close

This function closes a handle to the driver. If it is the last client open, the driver will send an RX Disable command to the ENC hardware and move to the closed state.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
handle	The successfully opened handle

DRV_ENCX24J600_ConfigGet Function

Gets the current configuration.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
size_t DRV_ENCX24J600_ConfigGet(DRV_HANDLE hMac, void* configBuff, size_t buffSize, size_t* pConfigSize);
```

Returns

Number of bytes copied to the buffer

Description

ENCX24J600 Get Configuration

Gets the current configuration.

Remarks

This function does nothing in the first release.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
configBuff	location to copy the configuration too
buffSize	buffer size
pConfigSize	configuration size needed

DRV_ENCX24J600_LinkCheck Function

This function returns the status of the link.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
bool DRV_ENCX24J600_LinkCheck(DRV_HANDLE hMac);
```

Returns

- true - if the link is active
- false - all other times

Description

ENCX24J600 Link Check

This function checks the status of the link and returns it to the caller.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

DRV_ENCX24J600_Open Function

This function is called by the client to open a handle to a driver instance.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
DRV_HANDLE DRV_ENCX24J600_Open(SYS_MODULE_INDEX index, DRV_IO_INTENT intent);
```

Returns

Returns a valid handle - If successful INVALID_HANDLE - If unsuccessful

Description

ENCX24J600 Open

The client will call this function to open a handle to the driver. When the first instance is opened than the driver will send the RX enabled command to the ENC hardware.

Preconditions

The driver had to be successfully initialized with DRV_ENCX24J600_Initialize.

Parameters

Parameters	Description
index	This is the index of the driver instance to be initialized. The definition DRV_ENCX24J600_NUM_DRV_INSTANCES controls how many instances are available.
intent	The intent to use when opening the driver. Only exclusive is supported

DRV_ENCX24J600_ParametersGet Function

Get the parameters of the device.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_RES DRV_ENCX24J600_ParametersGet(DRV_HANDLE hMac, TCPIP_MAC_PARAMETERS* pMacParams);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OK - if the hMac is valid

Description

ENCX24J600 Get Parameters

Get the parameters of the device, which includes that it is an Ethernet device and what it's MAC address is.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pMacParams	pointer to put the parameters

DRV_ENCX24J600_PowerMode Function

This function sets the power mode of the device.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
bool DRV_ENCX24J600_PowerMode(DRV_HANDLE hMac, TCPIP_MAC_POWER_MODE pwrMode);
```

Returns

- false - This functionality is not supported in this version of the driver

Description

ENCX24J600 Power Mode

This function sets the power mode of the ENCX24J600.

Remarks

This functionality is not implemented in the first release.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pwrMode	the power mode to set

DRV_ENCX24J600_RegisterStatisticsGet Function

Get the register statistics.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_RES DRV_ENCX24J600_RegisterStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_STATISTICS_REG_ENTRY*
pRegEntries, int nEntries, int* pHwEntries);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENCX24J600 Get Register Statistics

Get the device specific statistics.

Remarks

Statistics are not planned for the first release

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pRegEntries	

nEntries	
pHwEntries	

DRV_ENCX24J600_StatisticsGet Function

Retrieve the devices statistics.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_RES DRV_ENCX24J600_StatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_RX_STATISTICS* pRxStatistics,
TCPIP_MAC_TX_STATISTICS* pTxStatistics);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENCX24J600 Get Statistics

Get the current statistics stored in the driver.

Remarks

Statistics are not planned for the first release.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

DRV_ENCX24J600_Status Function

Gets the current status of the driver.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
SYS_STATUS DRV_ENCX24J600_Status(SYS_MODULE_OBJ obect);
```

Returns

- SYS_STATUS_ERROR - if an invalid handle has been passed in
- SYS_STATUS_UNINITIALIZED - if the driver has not completed initialization
- SYS_STATUS_BUSY - if the driver is closing and moving to the closed state
- SYS_STATUS_READY - if the driver is ready for client commands

Description

ENCX24J600 Status

This function will get the status of the driver instance.

Preconditions

The driver had to be successfully initialized with DRV_ENCX24J600_Initialize().

Parameters

Parameters	Description
object	The object valid passed back to DRV_ENCX24J600_Initialize()

c) Receive Functions

DRV_ENCX24J600_PacketRx Function

Receive a packet from the driver.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_PACKET* DRV_ENCX24J600_PacketRx(DRV_HANDLE hMac, TCPIP_MAC_RES* pRes, const
TCPIP_MAC_PACKET_RX_STAT** ppPktStat);
```

Returns

- Pointer to a valid packet - if successful
- NULL - if unsuccessful

Description

ENCX24J600 Receive Packet

This function retrieves a packet from the driver. The packet needs to be acknowledged with the linked acknowledge function so it can be reused.

Remarks

ppPktStat is ignored in the first release.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
pRes	the result of the operation
ppPktStat	pointer to the receive statistics

DRV_ENCX24J600_RxFilterHashTableEntrySet Function

This function adds an entry to the hash table.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_RES DRV_ENCX24J600_RxFilterHashTableEntrySet(DRV_HANDLE hMac, TCPIP_MAC_ADDR* DestMACAddr);
```

Returns

- TCPIP_MAC_RES_TYPE_ERR - if the hMac is invalid
- TCPIP_MAC_RES_OP_ERR - if the hMac is valid

Description

ENCX24J600 Receive Filter Hash Table Entry Set

This function adds to the MAC's hash table for hash table matching.

Remarks

This functionality is not implemented in the first release.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
DestMACAddr	MAC address to add to the hash table

d) Transmit Functions

DRV_ENCX24J600_PacketTx Function

This function queues a packet for transmission.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_RES DRV_ENCX24J600_PacketTx(DRV_HANDLE hMac, TCPIP_MAC_PACKET * ptrPacket);
```

Returns

- TCPIP_MAC_RES_OP_ERR - if the client handle is invalid
- TCPIP_MAC_RES_IS_BUSY - if the driver is not in the run state
- TCPIP_MAC_RES_QUEUE_TX_FULL - if there are no free descriptors
- TCPIP_MAC_RES_OK - on successful queuing of the packet

Description

ENCX24J600 Packet Transmit

This function will take a packet and add it to the queue for transmission. When the packet has finished transmitting the driver will call the packets acknowledge function. When that acknowledge function is complete the driver will forget about the packet.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
ptrPacket	pointer to the packet

e) Event Functions

DRV_ENCX24J600_EventAcknowledge Function

Acknowledges an event.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
bool DRV_ENCX24J600_EventAcknowledge(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents);
```

Returns

- true - if successful
- false - if not successful

Description

ENCX24J600 Acknowledge Event

This function acknowledges an event.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
macEvents	the events to acknowledge

DRV_ENCX24J600_EventMaskSet Function

Sets the event mask.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
bool DRV_ENCX24J600_EventMaskSet(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents, bool enable);
```

Returns

- true - if the mask could be set
- false - if the mast could not be set

Description

ENCX24J600 Set Event Mask

Sets the event mask to what is passed in.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle
macEvents	the mask to enable or disable
enable	to enable or disable events

DRV_ENCX24J600_EventPendingGet Function

Gets the current events.

Implementation: Dynamic

File

drv_encx24j600.h

C

```
TCPIP_MAC_EVENT DRV_ENCX24J600_EventPendingGet(DRV_HANDLE hMac);
```

Returns

- TCPIP_MAC_EV_NONE - Returned on an error
- List of events - Returned on event other than an error

Description

ENCX24J600 Get Events

This function gets the current events.

Preconditions

The client had to be successfully opened with DRV_ENCX24J600_Open.

Parameters

Parameters	Description
hMac	the successfully opened handle

f) Data Types and Constants

DRV_ENCX24J600_Configuration Structure

Defines the data required to initialize or reinitialize the ENCX24J600 Driver.

File

drv_encx24j600.h

C

```
typedef struct _DRV_ENCX24J600_Configuration {
  uint16_t txDescriptors;
  uint16_t rxDescriptors;
  uint16_t rxDescBufferSize;
  SYS_MODULE_INDEX spiDrvIndex;
  uint32_t spiBps;
  uint16_t rxBufferSize;
  uint16_t maxFrameSize;
  PORTS_MODULE_ID spiSSPortModule;
  PORTS_CHANNEL spiSSPortChannel;
  PORTS_BIT_POS spiSSPortPin;
  bool intEnable;
  PORTS_MODULE_ID intPortModule;
  PORTS_CHANNEL intPortChannel;
  PORTS_BIT_POS intPortPin;
  DRV_ENCX24J600_MDIX_TYPE mdixControl;
  PORTS_MODULE_ID mdixPortModule;
  PORTS_CHANNEL mdixPortChannel;
  PORTS_BIT_POS mdixPortPin;
} DRV_ENCX24J600_Configuration;
```

Members

Members	Description
uint16_t txDescriptors;	Number of TX Descriptors to Allocate
uint16_t rxDescriptors;	Number of RX Descriptors to Allocate
uint16_t rxDescBufferSize;	Size of the buffer each RX Descriptor will use. Make sure its not smaller that maxFrameSize
SYS_MODULE_INDEX spiDrvIndex;	Index of the SPI driver to use
uint32_t spiBps;	Bus speed to use for the SPI interface. Section 1.0 of the ENCX24J600 data sheets says the maximum is 14000000 Hz. It is not recommended to go above this value.
uint16_t rxBufferSize;	The ENCX24J600 hardware has a 22 k dram. rxBufferSize defines how much of that memory is used by the rxBuffer
uint16_t maxFrameSize;	The maximum frame size to be supported by the hardware. 1536 is the default
PORTS_MODULE_ID spiSSPortModule;	Port Module of the GPIO pin hooked up to the CS/SS pin of the ENCX24J600
PORTS_CHANNEL spiSSPortChannel;	Port Channel of the GPIO pin hooked up to the CS/SS pin of the ENCX24J600
PORTS_BIT_POS spiSSPortPin;	Pin position of the GPIO pin hooked up to the CS/SS pin of the ENCX24J600
bool intEnable;	Use Interrupts or not.
PORTS_MODULE_ID intPortModule;	Port Module of the GPIO pin hooked up to the INT pin of the ENCX24J600
PORTS_CHANNEL intPortChannel;	Port Channel of the GPIO pin hooked up to the INT pin of the ENCX24J600
PORTS_BIT_POS intPortPin;	Pin Position of the GPIO pin hooked up to the INT pin of the ENCX24J600
DRV_ENCX24J600_MDIX_TYPE mdixControl;	To select the control type of the MDIX. This is only needed for hooking up to switches that don't have auto-mdix.
PORTS_MODULE_ID mdixPortModule;	Port Module of the GPIO pin hooked up to the MDIX select pin
PORTS_CHANNEL mdixPortChannel;	Port Channel of the GPIO pin hooked up to the MDIX select pin
PORTS_BIT_POS mdixPortPin;	Pin Position of the GPIO pin hooked up to the MDIX select pin

Description

ENCX24J600 Driver Initialization Data

This data type defines the data required to initialize or reinitialize the ENCX24J600 driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

DRV_ENCX24J600_MDIX_TYPE Enumeration

Defines the enumeration for controlling the MDIX select.

File

drv_encx24j600.h

C

```
typedef enum {
  DRV_ENCX24J600_NO_CONTROL = 0,
  DRV_ENCX24J600_NORMAL,
  DRV_ENCX24J600_REVERSE = 0
} DRV_ENCX24J600_MDIX_TYPE;
```

Members

Members	Description
DRV_ENCX24J600_NO_CONTROL = 0	No Control
DRV_ENCX24J600_NORMAL	Normal MDIX
DRV_ENCX24J600_REVERSE = 0	Reverse MDIX

Description

ENCX24J600 Driver MDIX Control type

This type defines the enumeration for controlling the MDIX select.

Remarks

None.

Files

Files

Name	Description
drv_encx24j600.h	ENCx24J600 Driver interface definition.

Description

drv_encx24j600.h

ENCx24J600 Driver interface definition.

Enumerations

	Name	Description
	DRV_ENCX24J600_MDIX_TYPE	Defines the enumeration for controlling the MDIX select.

Functions

	Name	Description
	DRV_ENCX24J600_Close	Closes a client handle to the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_ConfigGet	Gets the current configuration. **Implementation:** Dynamic

	DRV_ENCX24J600_Deinitialize	Deinitializes the ENCx24J600 Driver Instance. **Implementation:** Dynamic
	DRV_ENCX24J600_EventAcknowledge	Acknowledges an event. **Implementation:** Dynamic
	DRV_ENCX24J600_EventMaskSet	Sets the event mask. **Implementation:** Dynamic
	DRV_ENCX24J600_EventPendingGet	Gets the current events. **Implementation:** Dynamic
	DRV_ENCX24J600_Initialize	Initializes the ENCx24J600 Driver Instance, with the configuration data. **Implementation:** Dynamic
	DRV_ENCX24J600_LinkCheck	This function returns the status of the link. **Implementation:** Dynamic
	DRV_ENCX24J600_Open	This function is called by the client to open a handle to a driver instance. **Implementation:** Dynamic
	DRV_ENCX24J600_PacketRx	Receive a packet from the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_PacketTx	This function queues a packet for transmission. **Implementation:** Dynamic
	DRV_ENCX24J600_ParametersGet	Get the parameters of the device. **Implementation:** Dynamic
	DRV_ENCX24J600_PowerMode	This function sets the power mode of the device. **Implementation:** Dynamic
	DRV_ENCX24J600_Process	Additional processing that happens outside the tasks function. **Implementation:** Dynamic
	DRV_ENCX24J600_RegisterStatisticsGet	Get the register statistics. **Implementation:** Dynamic
	DRV_ENCX24J600_Reinitialize	Reinitializes the instance of the ENCX24J600 driver. **Implementation:** Dynamic
	DRV_ENCX24J600_RxFilterHashTableEntrySet	This function adds an entry to the hash table. **Implementation:** Dynamic
	DRV_ENCX24J600_SetMacCtrlInfo	This function sets the MAC control information for the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_StackInitialize	This function initializes the driver with a TCPIP_MAC_INIT object. **Implementation:** Dynamic
	DRV_ENCX24J600_StatisticsGet	Retrieve the devices statistics. **Implementation:** Dynamic
	DRV_ENCX24J600_Status	Gets the current status of the driver. **Implementation:** Dynamic
	DRV_ENCX24J600_Tasks	Main task function for the driver. **Implementation:** Dynamic

Structures

	Name	Description
	_DRV_ENCX24J600_Configuration	Defines the data required to initialize or reinitialize the ENCX24J600 Driver.
	DRV_ENCX24J600_Configuration	Defines the data required to initialize or reinitialize the ENCX24J600 Driver.

Description

ENCx24J600 Driver Public Interface

This file defines the interface definition for the ENCx24J600 Driver.

File Name

drv_encx24j600.h

Company

Microchip Technology Inc.

Ethernet MAC Driver Library

This section describes the Ethernet MAC Driver Library.

Introduction

This library provides a driver-level abstraction of the on-chip Ethernet Controller found on many PIC32 devices. The driver implements the virtual MAC driver model that the MPLAB Harmony TCP/IP Stack requires. Please see the TCP/IP Stack Library MAC Driver Module help for details.

The "Host-To-Network" layer of a TCP/IP stack organization covers the Data Link and Physical Layers of the standard OSI stack. The Ethernet Controller provides the Data Link or Media Access Control Layer, in addition to other functions discussed in this section. An external Ethernet "PHY" provides the Physical layer, providing conversion between the digital and analog.

Description

The PIC32 Ethernet Controller is a bus master module that interfaces with an off-chip PHY to implement a complete Ethernet node in a system. The following are some of the key features of this module:

- Supports 10/100 Ethernet
- Full-Duplex and Half-Duplex operation
- Broadcast, Multicast and Unicast packets
- Manual and automatic flow control
- Supports Auto-MDIX enabled PHYs
- Reduced Media Independent Interface (RMII) and Media Independent Interface (MII) PHY data interfaces
- Performance statistics metrics in hardware.
- RAM descriptor based DMA operation for both receive and transmit path
- Fully configurable interrupts
- Configurable receive packet filtering using:
- 64-bit Hash Table
- 64-byte Pattern Match
- Magic Packet™ Filtering
- Runt Packet Detection and Filtering
- Supports Packet Payload Checksum calculation
- CRC Check

Support for the Serial Management Interface (SMI) (also known as the MIIM interface) is provided by the Ethernet PHY Driver Library.

Using the Library

The user of this driver is the MPLAB Harmony TCP/IP stack. This Ethernet driver is not intended as a system wide driver that the application or other system modules may use. It is intended for the sole use of the MPLAB Harmony TCP/IP stack and implements the virtual MAC model required by the stack.

This topic describes the basic architecture and functionality of the Ethernet MAC driver and is meant for advanced users or TCP/IP stack driver developers.

Interface Header File: `drv_ethmac.h`

The interface to the Ethernet MAC library is defined in the `drv_ethmac.h` header file, which is included by the MPLAB Harmony TCP/IP stack.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the Ethernet MAC Driver Library on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

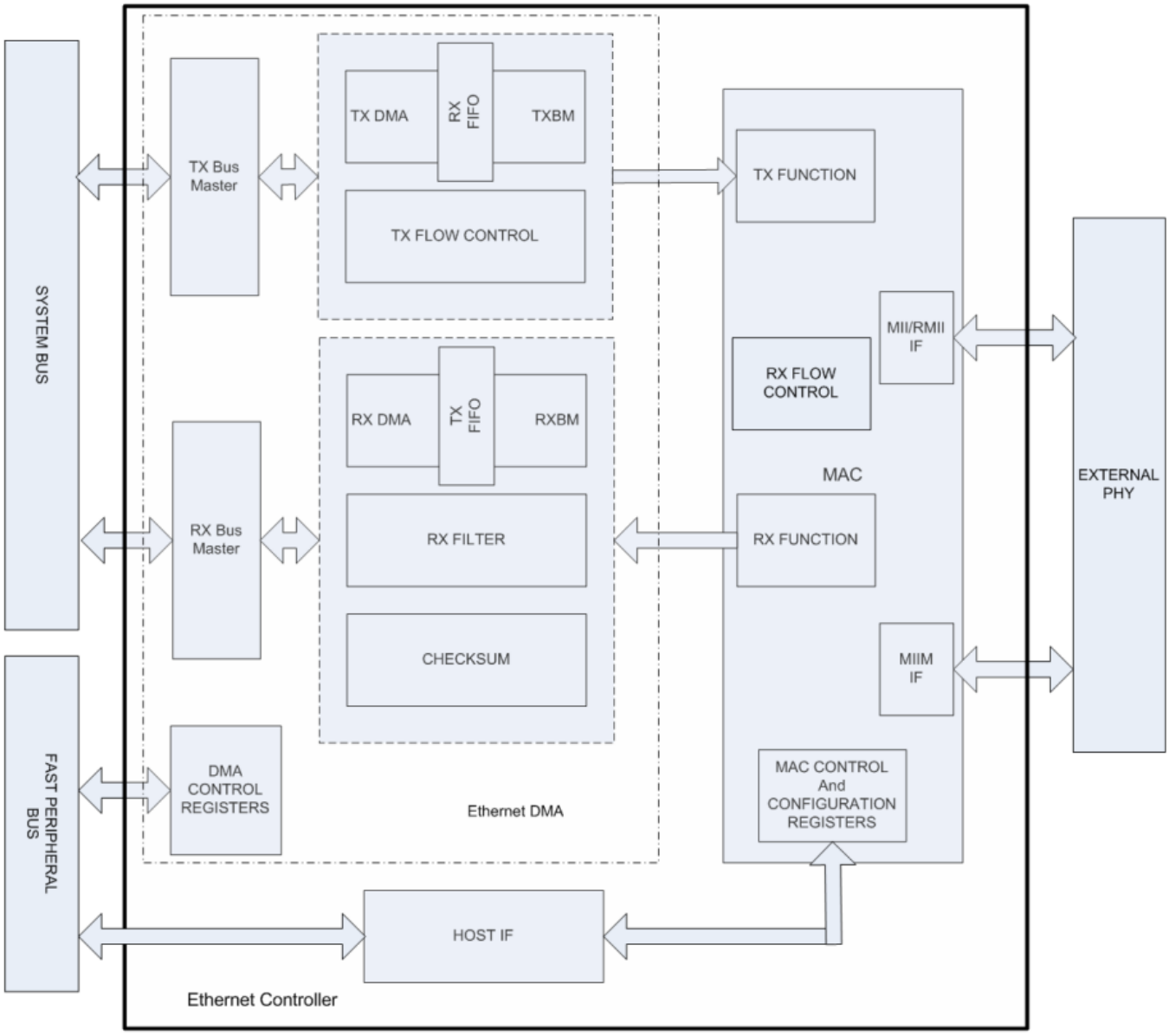
The Ethernet Controller provides the modules needed to implement a 10/100 Mbps Ethernet node using an external Ethernet PHY chip. The PHY chip provides a digital-analog interface as part of the Physical Layer and the controller provides the Media Access Controller (MAC) layer above the PHY.

As shown in Figure 1, the Ethernet Controller consists of the following modules:

- Media Access Control (MAC) block: Responsible for implementing the MAC functions of the Ethernet IEEE 802.3 Specification
- Flow Control (FC) block: Responsible for control of the transmission of PAUSE frames. (Reception of PAUSE frames is handled within the MAC.)
- RX Filter (RXF) block: This module performs filtering on every receive packet to determine whether each packet should be accepted or rejected

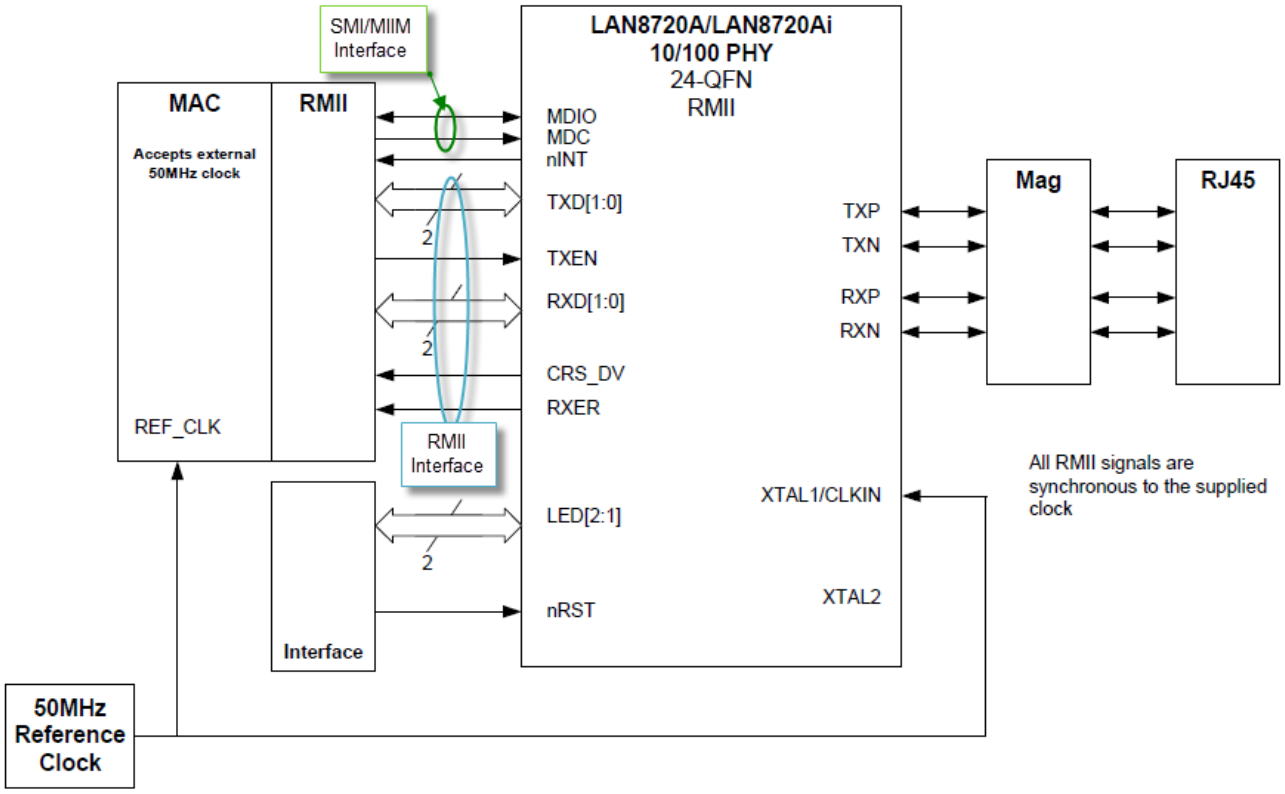
- TX DMA/TX Buffer Management Engine: The TX DMA and TX Buffer Management engines perform data transfers from the memory (using descriptor tables) to the MAC Transmit Interface
- RX DMA/RX Buffer Management Engine: The RX DMA and RX Buffer Management engines transfer receive packets from the MAC to the memory (using descriptor tables)

Figure 1: Ethernet Controller Block Diagram



For completeness, we also need to look at the interface diagram of a representative Ethernet PHY. As shown in Figure 2, the PHY has two interfaces, one for configuring and managing the PHY (SMI/MIIM) and another for transmit and receive data (RMII or MII). The SMI/MIIM interface is the responsibility of the Ethernet PHY Driver Library. When setting up the Ethernet PHY, this Ethernet driver calls primitives from the Ethernet PHY Driver library. The RMII/MII data interface is the responsibility of the Ethernet MAC Driver Library (this library).

Figure 2: Ethernet PHY Interfaces



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system. Refer to the TCP/IP Stack Library MAC Driver Module help for the interface that the Ethernet driver has to implement in a MPLAB Harmony system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Ethernet MAC Driver Library.

Library Interface Section	Description
Client Level Functions	DRV_ETHMAC_PIC32MACOpen, DRV_ETHMAC_PIC32MACClose, and DRV_ETHMAC_PIC32MACSetup to support the TCP/IP Stack. Plus link status and power options.
Receive Functions	Receive routines.
Transmit Functions	Transmit routines.
Event Functions	Ethernet event support routines.
Other Functions	Additional routines.
Data Types and Constants	Typedefs and #defines.

Configuring the Library

Macros

	Name	Description
	DRV_ETHMAC_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ETHMAC_INDEX	Ethernet MAC static index selection.
	DRV_ETHMAC_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_ETHMAC_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_ETHMAC_INTERRUPT_SOURCE	Defines an override of the interrupt source in case of static driver.
	DRV_ETHMAC_PERIPHERAL_ID	Defines an override of the peripheral ID.
	DRV_ETHMAC_POWER_STATE	Defines an override of the power state of the Ethernet MAC driver.

Description

The configuration of the Ethernet MAC driver is done as part of the MPLAB Harmony TCP/IP Stack configuration and is based on the `system_config.h` file, which may include the `tcpip_mac_config.h`. See the TCP/IP Stack Library MAC Driver Module help file for configuration options.

This header file contains the configuration selection for the Ethernet MAC Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_ETHMAC_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_CLIENTS_NUMBER 1
```

Description

Ethernet MAC Maximum Number of Clients

This definition select the maximum number of clients that the Ethernet MAC driver can support at run time.

Remarks

The MAC driver is not a true multi-client driver. Under normal usage, the only client of the MAC driver is the TCP/IP stack. After the MAC driver provided an DRV_HANDLE as a result of an Open operation, any other attempt to call Open will return a invalid handle. Default value should be 1.

However, for allowing other modules to interface directly with the MAC driver while the TCP/IP stack currently uses the the MAC driver this symbol can have a value greater than 1. But the returned handle is the same one as the TCP/IP stack uses.

DRV_ETHMAC_INDEX Macro

Ethernet MAC static index selection.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_INDEX DRV_ETHMAC_INDEX_1
```

Description

Ethernet MAC Static Index Selection

This definition selects the Ethernet MAC static index for the driver object reference

Remarks

This index is required to make a reference to the driver object.

DRV_ETHMAC_INSTANCES_NUMBER Macro

Selects the maximum number of hardware instances that can be supported by the dynamic driver.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_INSTANCES_NUMBER 1
```

Description

Ethernet MAC hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver. Not defining it means using a static driver.

Remarks

None.

DRV_ETHMAC_INTERRUPT_MODE Macro

Controls operation of the driver in the interrupt or polled mode.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_INTERRUPT_MODE true
```

Description

Ethernet MAC Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of timer operation is desired
- false - Select if polling mode of timer operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

DRV_ETHMAC_INTERRUPT_SOURCE Macro

Defines an override of the interrupt source in case of static driver.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_INTERRUPT_SOURCE INT_SOURCE_ETH_1
```

Description

Ethernet MAC Interrupt Source

Defines an override of the interrupt source in case of static driver.

Remarks

Refer to the INT PLIB document for more information on INT_SOURCE enumeration.

DRV_ETHMAC_PERIPHERAL_ID Macro

Defines an override of the peripheral ID.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_PERIPHERAL_ID ETHMAC_ID_1
```

Description

Ethernet MAC Peripheral ID Selection

Defines an override of the peripheral ID, using macros.

Remarks

Some devices also support ETHMAC_ID_0

DRV_ETHMAC_POWER_STATE Macro

Defines an override of the power state of the Ethernet MAC driver.

File

drv_ethmac_config.h

C

```
#define DRV_ETHMAC_POWER_STATE SYS_MODULE_POWER_IDLE_STOP
```

Description

Ethernet MAC power state configuration

Defines an override of the power state of the Ethernet MAC driver.

Remarks

This feature may not be available in the device or the Ethernet MAC module selected.

Building the Library

This section lists the files that are available in the Ethernet MAC Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/ethmac`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ethmac.h`	Header file that exports the driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_ethmac.c`	PIC32 internal Ethernet driver virtual MAC implementation file.
`/src/dynamic/drv_ethmac_lib.c`	PIC32 internal Ethernet driver controller implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The Ethernet MAC Driver Library depends on the following modules:

- Ethernet PHY Driver Library
- Interrupt System Service Library
- Timer System Service Library
- Ethernet Peripheral Library

Library Interface

a) Client Level Functions

	Name	Description
	DRV_ETHMAC_PIC32MACClose	Closes a client instance of the PIC32 MAC Driver. **Implementation:** Dynamic

	DRV_ETHMAC_PIC32MACDeinitialize	Deinitializes the PIC32 Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACInitialize	Initializes the PIC32 Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACLinkCheck	Checks current link status. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACOpen	Opens a client instance of the PIC32 MAC Driver. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACParametersGet	MAC parameter get function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACPowerMode	Selects the current power mode for the Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACProcess	MAC periodic processing function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACStatisticsGet	Gets the current MAC statistics. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACStatus	Provides the current status of the MAC driver module. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACConfigGet	Gets the current MAC driver configuration. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACRegisterStatisticsGet	Gets the current MAC hardware statistics registers. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACReinitialize	Reinitializes the PIC32 Ethernet MAC. **Implementation:** Dynamic

b) Receive Functions

	Name	Description
	DRV_ETHMAC_PIC32MACPacketRx	This is the MAC receive function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet	Sets the current MAC hash table receive filter. **Implementation:** Dynamic

c) Transmit Functions

	Name	Description
	DRV_ETHMAC_PIC32MACPacketTx	MAC driver transmit function. **Implementation:** Dynamic

d) Event Functions

	Name	Description
	DRV_ETHMAC_PIC32MACEventAcknowledge	Acknowledges and re-enables processed events. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACEventMaskSet	Enables/disables the MAC events. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACEventPendingGet	Returns the currently pending events. **Implementation:** Dynamic

e) Other Functions

	Name	Description
	DRV_ETHMAC_Tasks_ISR	Ethernet MAC driver interrupt function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACTasks	Maintains the EThernet MAC driver's state machine. **Implementation:** Dynamic

f) Data Types and Constants

	Name	Description
	DRV_ETHMAC_INDEX_1	This is macro DRV_ETHMAC_INDEX_1.
	DRV_ETHMAC_INDEX_0	Ethernet driver index definitions.
	DRV_ETHMAC_INDEX_COUNT	Number of valid Ethernet driver indices.

Description

This section lists the interface routines, data types, constants and macros for the library.

a) Client Level Functions

DRV_ETHMAC_PIC32MACClose Function

Closes a client instance of the PIC32 MAC Driver.

Implementation: Dynamic

File

drv_ethmac.h

C

```
void DRV_ETHMAC_PIC32MACClose(DRV_HANDLE hMac);
```

Returns

None

Description

This function closes a client instance of the PIC32 MAC Driver.

Remarks

None

Preconditions

DRV_ETHMAC_PIC32MACOpen() should have been called.

Example

Parameters

Parameters	Description
hMac	valid MAC handle, obtained by a call to DRV_ETHMAC_PIC32MACOpen

Function

void DRV_ETHMAC_PIC32MACClose(DRV_HANDLE hMac)

DRV_ETHMAC_PIC32MACDeinitialize Function

Deinitializes the PIC32 Ethernet MAC.

Implementation: Dynamic

File

drv_ethmac.h

C

```
void DRV_ETHMAC_PIC32MACDeinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function supports teardown of the PIC32 Ethernet MAC (opposite of set up). Used by tcpip_module_manager.

Remarks

This function deinitializes the Ethernet controller, the MAC and the associated PHY. It should be called to be release any resources allocated by the initialization and return the MAC and the PHY to the idle/power down state.

Preconditions

DRV_ETHMAC_PIC32MACInitialize must have been called to set up the driver.

Example

Function

void DRV_ETHMAC_PIC32MACDeinitialize(SYS_MODULE_OBJ object);

DRV_ETHMAC_PIC32MACInitialize Function

Initializes the PIC32 Ethernet MAC.

Implementation: Dynamic

File

drv_ethmac.h

C

```
SYS_MODULE_OBJ DRV_ETHMAC_PIC32MACInitialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const
init);
```

Returns

- a valid handle to a driver object, if successful.
- SYS_MODULE_OBJ_INVALID if initialization failed.

Description

This function supports the initialization of the PIC32 Ethernet MAC. Used by tcpip_module_manager.

Remarks

This function initializes the Ethernet controller, the MAC and the associated PHY. It should be called to be able to schedule any Ethernet transmit or receive operation.

Preconditions

None

Example

Function

SYS_MODULE_OBJ DRV_ETHMAC_PIC32MACInitialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);

DRV_ETHMAC_PIC32MACLinkCheck Function

Checks current link status.

Implementation: Dynamic

File

drv_ethmac.h

C

```
bool DRV_ETHMAC_PIC32MACLinkCheck(DRV_HANDLE hMac);
```

Returns

- true - If the link is up
- false - If the link is not up

Description

This function checks the link status of the associated network interface.

Remarks

The function will automatically perform a MAC reconfiguration if the link went up after being down and the PHY auto negotiation is enabled.

Preconditions

DRV_ETHMAC_PIC32MACInitialize must have been called to set up the driver. DRV_ETHMAC_PIC32MACOpen() should have been called to

obtain a valid handle.

Example

Parameters

Parameters	Description
hMac	Ethernet MAC client handle

Function

bool DRV_ETHMAC_PIC32MACLinkCheck(DRV_HANDLE hMac)

DRV_ETHMAC_PIC32MACOpen Function

Opens a client instance of the PIC32 MAC Driver.

Implementation: Dynamic

File

drv_ethmac.h

C

```
DRV_HANDLE DRV_ETHMAC_PIC32MACOpen(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

- DRV_HANDLE - handle (pointer) to MAC client
- 0 if call failed

Description

This function opens a client instance of the PIC32 MAC Driver. Used by tcpip_module_manager.

Remarks

The intent parameter is not used in the current implementation and is maintained only for compatibility with the generic driver Open function signature.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called.

Example

Function

DRV_HANDLE DRV_ETHMAC_PIC32MACOpen(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);

DRV_ETHMAC_PIC32MACParametersGet Function

MAC parameter get function.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACParametersGet(DRV_HANDLE hMac, TCPIP_MAC_PARAMETERS* pMacParams);
```

Returns

- TCPIP_MAC_RES_OK if pMacParams updated properly
- a TCPIP_MAC_RES error code if processing failed for some reason

Description

MAC Parameter Get function TCPIP_MAC_RES DRV_ETHMAC_PIC32MACParametersGet(DRV_HANDLE hMac, TCPIP_MAC_PARAMETERS* pMacParams);

This is a function that returns the run time parameters of the MAC driver.

Remarks

None.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

DRV_ETHMAC_PIC32MACPowerMode Function

Selects the current power mode for the Ethernet MAC.

Implementation: Dynamic

File

drv_ethmac.h

C

```
bool DRV_ETHMAC_PIC32MACPowerMode(DRV_HANDLE hMac, TCPIP_MAC_POWER_MODE pwrMode);
```

Returns

- true if the call succeeded.
- false if the call failed

Description

This function sets the power mode for the Ethernet MAC.

Remarks

This function is not currently supported by the Ethernet MAC and will always return true.

Preconditions

DRV_ETHMAC_PIC32MACInitialize must have been called to set up the driver. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

bool DRV_ETHMAC_PIC32MACPowerMode(DRV_HANDLE hMac, TCPIP_MAC_POWER_MODE pwrMode)

DRV_ETHMAC_PIC32MACProcess Function

MAC periodic processing function.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACProcess(DRV_HANDLE hMac);
```

Returns

- TCPIP_MAC_RES_OK if all processing went on OK
- a TCPIP_MAC_RES error code if processing failed for some reason

Description

This is a function that allows for internal processing by the MAC driver. It is meant for processing that cannot be done from within ISR.

Normally this function will be called in response to an TX and/or RX event signaled by the driver. This is specified by the MAC driver at initialization time using TCPIP_MAC_MODULE_CTRL.

Remarks

- The MAC driver may use the DRV_ETHMAC_PIC32MACProcess() for:
- Processing its pending TX queues
- RX buffers replenishing functionality. If the number of packets in the RX queue falls below a specified limit, the MAC driver may use this

function to allocate some extra RX packets. Similarly, if there are too many allocated RX packets, the MAC driver can free some of them.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Parameters

Parameters	Description
hMac	Ethernet MAC client handle

Function

TCPIP_MAC_RES DRV_ETHMAC_PIC32MACProcess(DRV_HANDLE hMac);

DRV_ETHMAC_PIC32MACStatisticsGet Function

Gets the current MAC statistics.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_RX_STATISTICS* pRxStatistics,
TCPIP_MAC_TX_STATISTICS* pTxStatistics);
```

Returns

- TCPIP_MAC_RES_OK if all processing went on OK.
- TCPIP_MAC_RES_OP_ERR error code if function not supported by the driver.

Description

This function will get the current value of the statistic counters maintained by the MAC driver.

Remarks

- The reported values are info only and change dynamically.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

TCPIP_MAC_RES DRV_ETHMAC_PIC32MACStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_RX_STATISTICS* pRxStatistics, TCPIP_MAC_TX_STATISTICS* pTxStatistics);

DRV_ETHMAC_PIC32MACStatus Function

Provides the current status of the MAC driver module.

Implementation: Dynamic

File

drv_ethmac.h

C

```
SYS_STATUS DRV_ETHMAC_PIC32MACStatus(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed
- SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed
- SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This function provides the current status of the MAC driver module.

Remarks

None.

Preconditions

The DRV_ETHMAC_PIC32MACInitialize function must have been called before calling this function.

Example

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_ETHMAC_PIC32MACInitialize

Function

SYS_STATUS DRV_ETHMAC_PIC32MACStatus (SYS_MODULE_OBJ object)

DRV_ETHMAC_PIC32MACConfigGet Function

Gets the current MAC driver configuration.

Implementation: Dynamic

File

drv_ethmac.h

C

```
size_t DRV_ETHMAC_PIC32MACConfigGet(DRV_HANDLE hMac, void* configBuff, size_t buffSize, size_t*
pConfigSize);
```

Returns

- number of bytes copied into the supplied storage buffer

Description

This function will get the current MAC driver configuration and store it into a supplied buffer.

Remarks

- None

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

size_t DRV_ETHMAC_PIC32MACConfigGet(DRV_HANDLE hMac, void* configBuff, size_t buffSize, size_t* pConfigSize);

DRV_ETHMAC_PIC32MACRegisterStatisticsGet Function

Gets the current MAC hardware statistics registers.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACRegisterStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_STATISTICS_REG_ENTRY*
pRegEntries, int nEntries, int* pHwEntries);
```

Returns

- TCPIP_MAC_RES_OK if all processing went on OK.
- TCPIP_MAC_RES_OP_ERR error code if function not supported by the driver.

Description

This function will get the current value of the statistic registers of the associated MAC controller.

Remarks

- The reported values are info only and change dynamically.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

TCPIP_MAC_RES DRV_ETHMAC_PIC32MACRegisterStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_STATISTICS_REG_ENTRY* pRegEntries, int nEntries, int* pHwEntries);

DRV_ETHMAC_PIC32MACReinitialize Function

Reinitializes the PIC32 Ethernet MAC.

Implementation: Dynamic

File

drv_ethmac.h

C

```
void DRV_ETHMAC_PIC32MACReinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

This function supports re-initialization of the PIC32 Ethernet MAC (opposite of set up).

Remarks

This function is not supported yet.

Preconditions

DRV_ETHMAC_PIC32MACInitialize must have been called to set up the driver.

Example

Function

void DRV_ETHMAC_PIC32MACReinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);

b) Receive Functions

DRV_ETHMAC_PIC32MACPacketRx Function

This is the MAC receive function.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_PACKET* DRV_ETHMAC_PIC32MACPacketRx(DRV_HANDLE hMac, TCPIP_MAC_RES* pRes, const
TCPIP_MAC_PACKET_RX_STAT** ppPktStat);
```

Returns

- a valid pointer to an available RX packet
- 0 if no packet pending/available

Description

This function will return a packet if such a pending packet exists.

Additional information about the packet is available by providing the pRes and ppPktStat fields.

Remarks

- Once a pending packet is available in the MAC driver internal RX queues this function will dequeue the packet and hand it over to the MAC driver's client - i.e., the stack - for further processing.
- The flags for a RX packet are updated by the MAC driver:
- TCPIP_MAC_PKT_FLAG_RX will be set
- TCPIP_MAC_PKT_FLAG_UNICAST is set if that packet is a unicast packet
- TCPIP_MAC_PKT_FLAG_BCAST is set if that packet is a broadcast packet
- TCPIP_MAC_PKT_FLAG_MCAST is set if that packet is a multicast packet
- TCPIP_MAC_PKT_FLAG_QUEUED is set
- TCPIP_MAC_PKT_FLAG_SPLIT is set if the packet has multiple data segments
- The MAC driver dequeues and return to the caller just one single packet. That is the packets are not chained.
- The packet buffers are allocated by the Ethernet MAC driver itself, Once the higher level layers in the stack are done with processing the RX packet, they have to call the corresponding packet acknowledgment function that tells the MAC driver that it can resume control of that packet.
- Once the stack modules are done processing the RX packets and the acknowledge function is called the MAC driver will reuse the RX packets.
- The MAC driver may use the DRV_ETHMAC_PIC32MACProcess() for obtaining new RX packets if needed.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

TCPIP_MAC_PACKET* DRV_ETHMAC_PIC32MACPacketRx (DRV_HANDLE hMac, TCPIP_MAC_RES* pRes, const TCPIP_MAC_PACKET_RX_STAT** ppPktStat);

DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet Function

Sets the current MAC hash table receive filter.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet(DRV_HANDLE hMac, TCPIP_MAC_ADDR* DestMACAddr);
```

Returns

- TCPIP_MAC_RES_OK if success
- a TCPIP_MAC_RES error value if failed

Description

This function sets the MAC hash table filtering to allow packets sent to DestMACAddr to be received. It calculates a CRC-32 using polynomial 0x4C11DB7 over the 6 byte MAC address and then, using bits 28:23 of the CRC, will set the appropriate bits in the hash table filter registers (ETHHT0-ETHHT1).

The function will enable/disable the Hash Table receive filter if needed.

Remarks

- Sets the appropriate bit in the ETHHT0/1 registers to allow packets sent to DestMACAddr to be received and enabled the Hash Table receive filter.
- There is no way to individually remove destination MAC addresses from the hash table since it is possible to have a hash collision and therefore multiple MAC addresses relying on the same hash table bit.
- A workaround is to have the stack store each enabled MAC address and to perform the comparison at run time.

- A call to DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet() using a 00-00-00-00-00-00 destination MAC address, which will clear the entire hash table and disable the hash table filter. This will allow the receive of all packets, regardless of their destination

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

TCPIP_MAC_RES DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet(DRV_HANDLE hMac, TCPIP_MAC_ADDR* DestMACAddr)

c) Transmit Functions

DRV_ETHMAC_PIC32MACPacketTx Function

MAC driver transmit function.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_RES DRV_ETHMAC_PIC32MACPacketTx(DRV_HANDLE hMac, TCPIP_MAC_PACKET * ptrPacket);
```

Returns

- TCPIP_MAC_RES_OK if success
- a TCPIP_MAC_RES error value if failed

Description

This is the MAC transmit function. Using this function a packet is submitted to the MAC driver for transmission.

Remarks

- The MAC driver supports internal queuing. A packet is rejected only if it's not properly formatted. Otherwise it will be scheduled for transmission and queued internally if needed.
- Once the packet is scheduled for transmission the MAC driver will set the TCPIP_MAC_PKT_FLAG_QUEUED flag so that the stack is aware that this packet is under processing and cannot be modified.
- Once the packet is transmitted, the TCPIP_MAC_PKT_FLAG_QUEUED will be cleared, the proper packet acknowledgment result (ackRes) will be set and the packet acknowledgment function (ackFunc) will be called.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

Function

TCPIP_MAC_RES DRV_ETHMAC_PIC32MACPacketTx(DRV_HANDLE hMac, TCPIP_MAC_PACKET * ptrPacket);

d) Event Functions

DRV_ETHMAC_PIC32MACEventAcknowledge Function

Acknowledges and re-enables processed events.

Implementation: Dynamic

File

drv_ethmac.h

C

```
bool DRV_ETHMAC_PIC32MACEventAcknowledge(DRV_HANDLE hMac, TCPIP_MAC_EVENT tcpAckEv);
```

Returns

- true if events acknowledged
- false if no events to be acknowledged

Description

This function acknowledges and re-enables processed events. Multiple events can be ORed together as they are processed together. The events acknowledged by this function should be the events that have been retrieved from the stack by calling DRV_ETHMAC_PIC32MACEventPendingGet() or have been passed to the stack by the driver using the registered notification handler and have been processed and have to be re-enabled.

Remarks

- All events should be acknowledged, in order to be re-enabled.
- Some events are fatal errors and should not be acknowledged (TCPIP_MAC_EV_RX_BUSERR, TCPIP_MAC_EV_TX_BUSERR). Driver/stack re-initialization is needed under such circumstances.
- Some events are just system/application behavior and they are intended only as simple info (TCPIP_MAC_EV_RX_OVFLOW, TCPIP_MAC_EV_RX_BUFNA, TCPIP_MAC_EV_TX_ABORT, TCPIP_MAC_EV_RX_ACT).
- The TCPIP_MAC_EV_RX_FWMARK and TCPIP_MAC_EV_RX_EWMARK events are part of the normal flow control operation (if auto flow control was enabled). They should be enabled alternatively, if needed.
- The events are persistent. They shouldn't be re-enabled unless they have been processed and the condition that generated them was removed. Re-enabling them immediately without proper processing will have dramatic effects on system performance.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

```
DRV_ETHMAC_PIC32MACEventAcknowledge( hMac, stackNewEvents );
```

Function

bool DRV_ETHMAC_PIC32MACEventAcknowledge(DRV_HANDLE hMac, TCPIP_MAC_EVENT tcpAckEv);

DRV_ETHMAC_PIC32MACEventMaskSet Function

Enables/disables the MAC events.

Implementation: Dynamic

File

drv_ethmac.h

C

```
bool DRV_ETHMAC_PIC32MACEventMaskSet(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents, bool enable);
```

Returns

always true, operation succeeded.

Description

This is a function that enables or disables the events to be reported to the Ethernet MAC client (TCP/IP stack).

All events that are to be enabled will be added to the notification process. All events that are to be disabled will be removed from the notification process. The stack has to catch the events that are notified and process them. After that the stack should call DRV_ETHMAC_PIC32MACEventAcknowledge() so that the events can be re-enable

The stack should process at least the following transfer events:

- TCPIP_MAC_EV_RX_PKTPEND
- TCPIP_MAC_EV_RX_DONE
- TCPIP_MAC_EV_TX_DONE

Remarks

- The event notification system enables the user of the TCP/IP stack to call into the stack for processing only when there are relevant events rather than being forced to periodically call from within a loop.

- If the notification events are nil, the interrupt processing will be disabled. Otherwise, the event notification will be enabled and the interrupts relating to the requested events will be enabled.
- Note that once an event has been caught by the stack ISR (and reported if a notification handler is in place) it will be disabled until the DRV_ETHMAC_PIC32MACEventAcknowledge() is called.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

```
DRV_ETHMAC_PIC32MACEventMaskSet( hMac, TCPIP_MAC_EV_RX_OVFLOW | TCPIP_MAC_EV_RX_BUFNA, true );
```

Function

bool DRV_ETHMAC_PIC32MACEventMaskSet(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents, bool enable);

DRV_ETHMAC_PIC32MACEventPendingGet Function

Returns the currently pending events.

Implementation: Dynamic

File

drv_ethmac.h

C

```
TCPIP_MAC_EVENT DRV_ETHMAC_PIC32MACEventPendingGet(DRV_HANDLE hMac);
```

Returns

The currently stack pending events.

Description

This function returns the currently pending Ethernet MAC events. Multiple events will be ORed together as they accumulate. The stack should perform processing whenever a transmission related event (TCPIP_MAC_EV_RX_PKTPEND, TCPIP_MAC_EV_TX_DONE) is present. The other, non critical events, may not be managed by the stack and passed to an user. They will have to be eventually acknowledged if re-enabling is needed.

Remarks

- This is the preferred method to get the current pending MAC events. The stack maintains a proper image of the events from their occurrence to their acknowledgment.
- Even with a notification handler in place it's better to use this function to get the current pending events rather than using the events passed by the notification handler which could be stale.
- The events are persistent. They shouldn't be re-enabled unless they have been processed and the condition that generated them was removed. Re-enabling them immediately without proper processing will have dramatic effects on system performance.
- The returned value is just a momentary value. The pending events can change any time.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. DRV_ETHMAC_PIC32MACOpen() should have been called to obtain a valid handle.

Example

```
TCPIP_MAC_EVENT currEvents = DRV_ETHMAC_PIC32MACEventPendingGet( hMac);
```

Function

TCPIP_MAC_EVENT DRV_ETHMAC_PIC32MACEventPendingGet(DRV_HANDLE hMac)

e) Other Functions

DRV_ETHMAC_Tasks_ISR Function

Ethernet MAC driver interrupt function.

Implementation: Dynamic

File

drv_ethmac.h

C

```
void DRV_ETHMAC_Tasks_ISR(SYS_MODULE_OBJ macIndex);
```

Returns

None.

Description

This is the Ethernet MAC driver interrupt service routine. It processes the Ethernet related interrupts and notifies the events to the driver user (the TCP/IP stack).

Remarks

None.

Preconditions

DRV_ETHMAC_PIC32MACInitialize() should have been called. The TCP/IP stack event notification should be enabled.

Function

void DRV_ETHMAC_Tasks_ISR(SYS_MODULE_OBJ macIndex)

DRV_ETHMAC_PIC32MACTasks Function

Maintains the EThernet MAC driver's state machine.

Implementation: Dynamic

File

drv_ethmac.h

C

```
void DRV_ETHMAC_PIC32MACTasks(SYS_MODULE_OBJ object);
```

Returns

None

Description

This function is used to maintain the driver's internal state machine

Remarks

None.

Preconditions

The DRV_ETHMAC_PIC32MACInitialize routine must have been called for the specified MAC driver instance.

Example

Function

void DRV_ETHMAC_PIC32MACTasks(SYS_MODULE_OBJ object)

f) Data Types and Constants

DRV_ETHMAC_INDEX_1 Macro

File

drv_ethmac.h

C

```
#define DRV_ETHMAC_INDEX_1 1
```

Description

This is macro DRV_ETHMAC_INDEX_1.

DRV_ETHMAC_INDEX_0 Macro

Ethernet driver index definitions.

File

drv_ethmac.h

C

```
#define DRV_ETHMAC_INDEX_0 0
```

Description

Ethernet Driver Module Index Numbers

These constants provide Ethernet driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the MAC initialization routines to identify the driver instance in use.

DRV_ETHMAC_INDEX_COUNT Macro

Number of valid Ethernet driver indices.

File

drv_ethmac.h

C

```
#define DRV_ETHMAC_INDEX_COUNT ETH_NUMBER_OF_MODULES
```

Description

Ethernet Driver Module Index Count

This constant identifies number of valid Ethernet driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from part-specific header files defined as part of the peripheral libraries.

Files

Files

Name	Description
drv_ethmac.h	Ethernet MAC device driver interface file
drv_ethmac_config.h	Ethernet MAC driver configuration definitions template.

Description

This section lists the source and header files used by the Ethernet MAC Driver Library.

drv_ethmac.h

Ethernet MAC device driver interface file

Functions

	Name	Description
	DRV_ETHMAC_PIC32MACClose	Closes a client instance of the PIC32 MAC Driver. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACConfigGet	Gets the current MAC driver configuration. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACDeinitialize	Deinitializes the PIC32 Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACEventAcknowledge	Acknowledges and re-enables processed events. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACEventMaskSet	Enables/disables the MAC events. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACEventPendingGet	Returns the currently pending events. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACInitialize	Initializes the PIC32 Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACLinkCheck	Checks current link status. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACOpen	Opens a client instance of the PIC32 MAC Driver. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACPacketRx	This is the MAC receive function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACPacketTx	MAC driver transmit function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACParametersGet	MAC parameter get function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACPowerMode	Selects the current power mode for the Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACProcess	MAC periodic processing function. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACRegisterStatisticsGet	Gets the current MAC hardware statistics registers. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACReinitialize	Reinitializes the PIC32 Ethernet MAC. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet	Sets the current MAC hash table receive filter. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACStatisticsGet	Gets the current MAC statistics. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACStatus	Provides the current status of the MAC driver module. **Implementation:** Dynamic
	DRV_ETHMAC_PIC32MACTasks	Maintains the EThernet MAC driver's state machine. **Implementation:** Dynamic
	DRV_ETHMAC_Tasks_ISR	Ethernet MAC driver interrupt function. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_ETHMAC_INDEX_0	Ethernet driver index definitions.
	DRV_ETHMAC_INDEX_1	This is macro DRV_ETHMAC_INDEX_1.
	DRV_ETHMAC_INDEX_COUNT	Number of valid Ethernet driver indices.

Description

Ethernet MAC Device Driver Interface

The Ethernet MAC device driver provides a simple interface to manage the Ethernet peripheral. This file defines the interface definitions and prototypes for the Ethernet MAC driver.

File Name

drv_ethmac.h

Company

Microchip Technology Inc.

drv_ethmac_config.h

Ethernet MAC driver configuration definitions template.

Macros

	Name	Description
	DRV_ETHMAC_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ETHMAC_INDEX	Ethernet MAC static index selection.
	DRV_ETHMAC_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_ETHMAC_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_ETHMAC_INTERRUPT_SOURCE	Defines an override of the interrupt source in case of static driver.
	DRV_ETHMAC_PERIPHERAL_ID	Defines an override of the peripheral ID.
	DRV_ETHMAC_POWER_STATE	Defines an override of the power state of the Ethernet MAC driver.

Description

ETHMAC Driver Configuration Definitions for the template version

These definitions statically define the driver's mode of operation.

File Name

drv_ethmac_config.h

Company

Microchip Technology Inc.

Ethernet PHY Driver Library

This section describes the Ethernet PHY Driver Library.

Introduction

This library provides a low-level abstraction of the Ethernet PHY Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

This library provides a software abstraction for configuring external Ethernet PHY devices for use with the on-chip PIC32 Ethernet Controller.

Using the Library

The user of this driver is the MPLAB Harmony TCP/IP Stack through its Ethernet MAC driver. This Ethernet PHY driver is not intended as a system wide driver that the application or other system modules may use. It is intended for the sole use of the MPLAB Harmony TCP/IP stack and implements the PHY driver required by the Ethernet MAC.

This topic describes the basic architecture and functionality of the Ethernet PHY driver and is meant for advanced users or TCP/IP Stack driver developers.

Interface Header File: `drv_ethphy.h`

The interface to the Ethernet PHY library is defined in the `drv_ethphy.h` header file, which is included by the MPLAB Harmony TCP/IP stack.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Abstraction Model

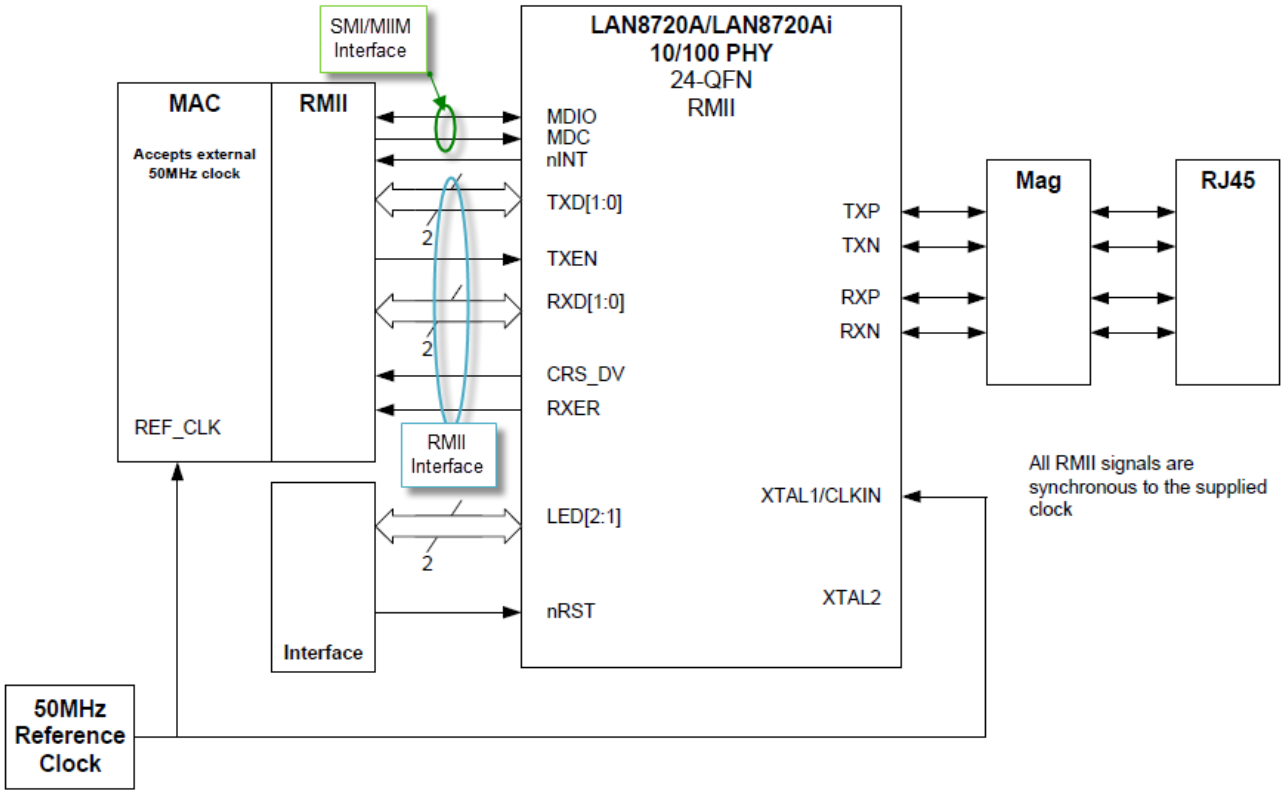
This library provides a low-level abstraction of the Ethernet PHY Driver Library on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

To understand how this library works you must first understand how an external Ethernet PHY interfaces with the Ethernet Controller. As shown in

Figure 1, the PHY has two interfaces, one for managing the PHY, known as the Serial Management Interface (SMI), for configuring the device and a second, known as the Reduced Media Independent Interface (RMII), for transmit and receive data.

Figure 1: Typical External PHY Interface



The block diagram also shows an interrupt signal (nINT) going to a external interrupt pin on the host device and signals going to on-board LEDs to show link state and link activity.

The SMI interface is also known as the MII Management (MIIM) interface. This control interface is standardized for all PHYs by Clause 22 of the 802.3 standard. It provides up to 32 16-bit registers on the PHY. The following table provides a summary of all 32 registers. Consult the data sheet for the PHY device for the specific bit fields in each register.

Register Address	Register Name	Register Type
0	Control	Basic
1	Status	Basic
2, 3	PHY Identifier	Extended
4	Auto-Negotiation Advertisement	Extended
5	Auto-Negotiation Link Partner Base Page Ability	Extended
6	Auto-Negotiation Expansion	Extended
7	Auto-Negotiation Next Page Transmit	Extended
8	Auto-Negotiation Link Partner Received Next Page	Extended
9	MASTER-SLAVE Control Register	Extended
10	MASTER-SLAVE Status Register	Extended
11-14	Reserved	Extended
15	Extended Status	Reserved

16-31	Vendor Specific	Extended

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Ethernet PHY Driver Library

Library Interface Section	Description
System Level Functions	Routines that integrate the driver into the MPLAB Harmony framework.
Client Level Functions	Open, Close, Link Status, Auto Negotiation.
SMI/MIIM Functions	SMI/MIIM Management Interface.
External PHY Support Functions	Provides the API for PHY support routines that the driver will call when setting up the PHY. The driver library provides support for four PHYs.
Other Functions	Functions that provide software version information.
Data Types and Constants	C language typedefs and enums used by this library.

Configuring the Library

Macros

	Name	Description
	DRV_ETHPHY_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ETHPHY_INDEX	Ethernet PHY static index selection.
	DRV_ETHPHY_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_ETHPHY_PERIPHERAL_ID	Defines an override of the peripheral ID.
	DRV_ETHPHY_NEG_DONE_TMO	Value of the PHY negotiation complete time out as per IEEE 802.3 spec.
	DRV_ETHPHY_NEG_INIT_TMO	Value of the PHY negotiation initiation time out as per IEEE 802.3 spec.
	DRV_ETHPHY_RESET_CLR_TMO	Value of the PHY Reset self clear time out as per IEEE 802.3 spec.

Description

The configuration of the Ethernet PHY Driver Library is based on the file `system_config.h`.

This header file contains the configuration selection for the Ethernet PHY Driver Library. Based on the selections made, the Ethernet PHY Driver Library may support the selected features.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_ETHPHY_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_CLIENTS_NUMBER 1
```

Description

Ethernet PHY Maximum Number of Clients This definition select the maximum number of clients that the Ethernet PHY driver can support at run time. Not defining it means using a single client.

Remarks

None.

DRV_ETHPHY_INDEX Macro

Ethernet PHY static index selection.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_INDEX DRV_ETHPHY_INDEX_1
```

Description

Ethernet PHY Static Index Selection

This definition selects the Ethernet PHY static index for the driver object reference.

Remarks

This index is required to make a reference to the driver object.

DRV_ETHPHY_INSTANCES_NUMBER Macro

Selects the maximum number of hardware instances that can be supported by the dynamic driver.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_INSTANCES_NUMBER 1
```

Description

Ethernet PHY hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver. Not defining it means using a static driver.

Remarks

None.

DRV_ETHPHY_PERIPHERAL_ID Macro

Defines an override of the peripheral ID.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_PERIPHERAL_ID ETHPHY_ID_1
```

Description

Ethernet PHY Peripheral ID Selection

Defines an override of the peripheral ID, using macros.

Remarks

Some devices also support ETHPHY_ID_0

DRV_ETHPHY_NEG_DONE_TMO Macro

Value of the PHY negotiation complete time out as per IEEE 802.3 spec.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_NEG_DONE_TMO (2000)
```

Description

Ethernet PHY Negotiation Complete time out

This definition sets the time out of the PHY negotiation complete, in ms.

Remarks

See IEEE 802.3 Clause 28 Table 28-9 autoneg_wait_timer value (max 1s).

DRV_ETHPHY_NEG_INIT_TMO Macro

Value of the PHY negotiation initiation time out as per IEEE 802.3 spec.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_NEG_INIT_TMO (1)
```

Description

Ethernet PHY Negotiation Initiation time out

This definition sets the time out of the PHY negotiation initiation, in ms.

Remarks

None.

DRV_ETHPHY_RESET_CLR_TMO Macro

Value of the PHY Reset self clear time out as per IEEE 802.3 spec.

File

drv_ethphy_config.h

C

```
#define DRV_ETHPHY_RESET_CLR_TMO (500)
```

Description

Ethernet PHY Reset self clear time out

This definition sets the time out of the PHY Reset self clear, in ms.

Remarks

See IEEE 802.3 Clause 22 Table 22-7 and paragraph "22.2.4.1.1 Reset" (max 0.5s)

Building the Library

This section lists the files that are available in the Ethernet PHY Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/ethphy`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_ethphy.h`	Header file that exports the driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/drv_extphy.c	Basic PHY driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
/src/dynamic/drv_extphy_smsc8700.c	SMSC 8700 PHY implementation file.
/src/dynamic/drv_extphy_smsc8720.c	SMSC 8720 PHY implementation file.
/src/dynamic/drv_extphy_smsc8720.c	SMSC 8740 PHY implementation file.
/src/dynamic/drv_extphy_ip101gr.c	IP101GR PHY implementation file.
/src/dynamic/drv_extphy_dp83640.c	National DP83640 PHY implementation file.
/src/dynamic/drv_extphy_dp83848.c	National DP83848 PHY implementation file.

Module Dependencies

The Ethernet MAC Driver Library depends on the following modules:

- Ethernet MAC Driver Library
- Clock System Service Library
- Ports System Service Library
- Timer System Service Library
- Ethernet Peripheral Library

Library Interface

a) System Level Functions

	Name	Description
	DRV_ETHPHY_Initialize	Initializes the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_Deinitialize	Deinitializes the specified instance of the Ethernet PHY driver module. **Implementation:** Dynamic
	DRV_ETHPHY_NegotiationResultGet	Returns the result of a completed negotiation. **Implementation:** Dynamic
	DRV_ETHPHY_PhyAddressGet	Returns the PHY address. **Implementation:** Dynamic
	DRV_ETHPHY_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings. **Implementation:** Dynamic
	DRV_ETHPHY_Status	Provides the current status of the Ethernet PHY driver module. **Implementation:** Dynamic
	DRV_ETHPHY_Tasks	Maintains the driver's state machine and implements its ISR. **Implementation:** Dynamic
	DRV_ETHPHY_HWConfigFlagsGet	Returns the current Ethernet PHY hardware MII/RMII and ALTERNATE/DEFAULT configuration flags. **Implementation:** Dynamic
	DRV_ETHPHY_Setup	Initializes Ethernet PHY configuration and set up procedure. **Implementation:** Dynamic

b) Client Level Functions

	Name	Description
	DRV_ETHPHY_ClientStatus	Gets the current client-specific status the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_Close	Closes an opened instance of the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_NegotiationIsComplete	Returns the results of a previously initiated Ethernet PHY negotiation. **Implementation:** Dynamic
	DRV_ETHPHY_Open	Opens the specified Ethernet PHY driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_ETHPHY_Reset	Immediately resets the Ethernet PHY. **Implementation:** Dynamic
	DRV_ETHPHY_RestartNegotiation	Restarts auto-negotiation of the Ethernet PHY link. **Implementation:** Dynamic
	DRV_ETHPHY_ClientOperationAbort	Aborts a current client operation initiated by the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_ClientOperationResult	Gets the result of a client operation initiated by the Ethernet PHY driver. **Implementation:** Dynamic

c) SMI/MIIM Functions

	Name	Description
	DRV_ETHPHY_SMIScanStatusGet	Gets the status of the SMI/MIIM scan data. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanStop	Stops the scan of a previously requested SMI/MIIM register. **Implementation:** Dynamic
	DRV_ETHPHY_SMIClockSet	Sets the SMI/MIIM interface clock. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanStart	Starts the scan of a requested SMI/MIIM register. **Implementation:** Dynamic
	DRV_ETHPHY_SMIRead	Initiates a SMI/MIIM read transaction. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanDataGet	Gets the latest SMI/MIIM scan data result. **Implementation:** Dynamic
	DRV_ETHPHY_SMIStatus	Returns the current status of the SMI/MIIM interface. **Implementation:** Dynamic
	DRV_ETHPHY_SMIWrite	Initiates a SMI/MIIM write transaction. **Implementation:** Dynamic

d) Vendor Functions

	Name	Description
	DRV_ETHPHY_VendorDataGet	Returns the current value of the vendor data. **Implementation:** Dynamic
	DRV_ETHPHY_VendorDataSet	Returns the current value of the vendor data. **Implementation:** Dynamic
	DRV_ETHPHY_VendorSMIReadResultGet	Reads the result of a previous vendor initiated SMI read transfer with DRV_ETHPHY_VendorSMIReadStart. **Implementation:** Dynamic
	DRV_ETHPHY_VendorSMIReadStart	Starts a vendor SMI read transfer. Data will be available with DRV_ETHPHY_VendorSMIReadResultGet. **Implementation:** Dynamic
	DRV_ETHPHY_VendorSMIWriteStart	Starts a vendor SMI write transfer. **Implementation:** Dynamic

f) Data Types and Constants

	Name	Description
	DRV_ETHPHY_CLIENT_STATUS	Identifies the client-specific status of the Ethernet PHY driver.
	DRV_ETHPHY_INIT	Contains all the data necessary to initialize the Ethernet PHY device.

	DRV_ETHPHY_NEGOTIATION_RESULT	Contains all the data necessary to get the Ethernet PHY negotiation result
	DRV_ETHPHY_SETUP	Contains all the data necessary to set up the Ethernet PHY device.
	DRV_ETHPHY_VENDOR_MDIX_CONFIGURE	Pointer to function that configures the MDIX mode for the Ethernet PHY.
	DRV_ETHPHY_VENDOR_MII_CONFIGURE	Pointer to function to configure the Ethernet PHY in one of the MII/RMII operation modes.
	DRV_ETHPHY_VENDOR_SMI_CLOCK_GET	Pointer to a function to return the SMI/MIIM maximum clock speed in Hz of the Ethernet PHY.
	DRV_ETHPHY_INDEX_0	Ethernet PHY driver index definitions.
	DRV_ETHPHY_INDEX_1	This is macro DRV_ETHPHY_INDEX_1.
	DRV_ETHPHY_INDEX_COUNT	Number of valid Ethernet PHY driver indices.
	DRV_ETHPHY_LinkStatusGet	Returns the current link status. **Implementation:** Dynamic
	DRV_ETHPHY_LINK_STATUS	Defines the possible status flags of PHY Ethernet link.
	DRV_ETHPHY_CONFIG_FLAGS	Defines the possible results of Ethernet operations that can succeed or fail
	DRV_ETHPHY_OBJECT	Identifies the interface of a Ethernet PHY vendor driver.
	DRV_ETHPHY_VENDOR_WOL_CONFIGURE	Pointer to a function to configure the PHY WOL functionality
	DRV_ETHPHY_OBJECT_BASE_TYPE	Identifies the base interface of a Ethernet PHY driver.
	DRV_ETHPHY_OBJECT_BASE	Identifies the base interface of a Ethernet PHY driver.

Description

This section describes the Application Programming Interface (API) functions of the Ethernet PHY Driver Library.

Refer to each section for a detailed description.

a) System Level Functions

DRV_ETHPHY_Initialize Function

Initializes the Ethernet PHY driver.

Implementation: Dynamic

File

drv_ethphy.h

C

```
SYS_MODULE_OBJ DRV_ETHPHY_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

- a valid handle to a driver object, if successful.
- SYS_MODULE_OBJ_INVALID if initialization failed.

Description

This function initializes the Ethernet PHY driver, making it ready for clients to open and use it.

Remarks

- This function must be called before any other Ethernet PHY routine is called.
- This function should only be called once during system initialization unless DRV_ETHPHY_Deinitialize is called to deinitialize the driver instance.
- The returned object must be passed as argument to DRV_ETHPHY_Reinitialize, DRV_ETHPHY_Deinitialize, DRV_ETHPHY_Tasks and DRV_ETHPHY_Status routines.

Preconditions

None.

Example

```
DRV_ETHPHY_INIT    init;
SYS_MODULE_OBJ  objectHandle;

// Populate the Ethernet PHY initialization structure
init.phyId  = ETHPHY_ID_0;
```

```
// Populate the Ethernet PHY initialization structure
init.phyId  = ETHPHY_ID_2;
init.pPhyObject  = &DRV_ETHPHY_OBJECT_SMSC_LAN8720;

// Do something

objectHandle = DRV_ETHPHY_Initialize(DRV_ETHPHY_INDEX_0, (SYS_MODULE_INIT*)&init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Function

SYS_MODULE_OBJ DRV_ETHPHY_Initialize(const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init)

DRV_ETHPHY_Deinitialize Function

Deinitializes the specified instance of the Ethernet PHY driver module.

Implementation: Dynamic

File

drv_ethphy.h

C

```
void DRV_ETHPHY_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the specified instance of the Ethernet PHY driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

- Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Preconditions

The DRV_ETHPHY_Initialize function must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_ETHPHY_Initialize
SYS_STATUS          status;

DRV_ETHPHY_Deinitialize(object);

status = DRV_ETHPHY_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Function

void DRV_ETHPHY_Deinitialize (SYS_MODULE_OBJ object)

DRV_ETHPHY_NegotiationResultGet Function

Returns the result of a completed negotiation.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_NegotiationResultGet(DRV_HANDLE handle, DRV_ETHPHY_NEGOTIATION_RESULT*
pNegResult);
```

Returns

- DRV_ETHPHY_RES_PENDING operation is ongoing
- an DRV_ETHPHY_RESULT error code if the procedure failed.

Description

This function returns the PHY negotiation data gathered after a completed negotiation.

Remarks

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

When operation is completed but negotiation has failed, DRV_ETHPHY_ClientOperationResult will return:

- DRV_ETHPHY_RES_NEGOTIATION_INACTIVE if no negotiation in progress
- DRV_ETHPHY_RES_NEGOTIATION_NOT_STARTED if negotiation not yet started yet (means time out if waitComplete was requested)
- DRV_ETHPHY_RES_NEGOTIATION_ACTIVE if negotiation ongoing

The returned value for the negotiation flags is valid only if the negotiation was completed successfully.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY
- DRV_ETHPHY_RestartNegotiation, and DRV_ETHPHY_NegotiationIsComplete should have been called.

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_NegotiationResultGet(DRV_HANDLE handle, DRV_ETHPHY_NEGOTIATION_RESULT* pNegResult)

DRV_ETHPHY_PhyAddressGet Function

Returns the PHY address.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_PhyAddressGet(DRV_HANDLE handle, int* pPhyAddress);
```

Returns

DRV_ETHPHY_RES_OK - operation successful and the PHY address stored at

DRV_ETHPHY_RES_HANDLE_ERR - passed in handle was invalid pPhyAddress

Description

This function returns the current PHY address as set by the DRV_ETHPHY_Setup procedure.

Remarks

None.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_PhyAddressGet(DRV_HANDLE handle, int* pPhyAddress);

DRV_ETHPHY_Reinitialize Function

Reinitializes the driver and refreshes any associated hardware settings.

Implementation: Dynamic

File

drv_ethphy.h

C

```
void DRV_ETHPHY_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

This function reinitializes the driver and refreshes any associated hardware settings using the initialization data given, but it will not interrupt any ongoing operations.

Remarks

- This function can be called multiple times to reinitialize the module.
- This operation can be used to refresh any supported hardware registers as specified by the initialization data or to change the power state of the module.

Preconditions

The DRV_ETHPHY_Initialize function must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
DRV_ETHPHY_INIT    init;
SYS_MODULE_OBJ  objectHandle;

// Populate the Ethernet PHY initialization structure
init.phyId  = ETHPHY_ID_2;
init.pPhyObject  = &DRV_ETHPHY_OBJECT_SMSC_LAN8720;

DRV_ETHPHY_Reinitialize(objectHandle, (SYS_MODULE_INIT*)&init);

phyStatus = DRV_ETHPHY_Status(objectHandle);
if (SYS_STATUS_BUSY == phyStatus)
{
    // Check again later to ensure the driver is ready
}
else if (SYS_STATUS_ERROR >= phyStatus)
{
    // Handle error
}
```

Function

void DRV_ETHPHY_Reinitialize(SYS_MODULE_OBJ object,
const SYS_MODULE_INIT * const init)

DRV_ETHPHY_Status Function

Provides the current status of the Ethernet PHY driver module.

Implementation: Dynamic

File

drv_ethphy.h

C

```
SYS_STATUS DRV_ETHPHY_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed
- SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed
- SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This function provides the current status of the Ethernet PHY driver module.

Remarks

- Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.
- SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another
- SYS_STATUS_ERROR - Indicates that the driver is in an error state
- Any value less than SYS_STATUS_ERROR is also an error state.
- SYS_MODULE_DEINITIALIZED - Indicates that the driver has been deinitialized
- The this operation can be used to determine when any of the driver's module level operations has completed.
- If the status operation returns SYS_STATUS_BUSY, the a previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.
- The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.
- This function will NEVER block waiting for hardware.
- If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The DRV_ETHPHY_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_ETHPHY_Initialize
SYS_STATUS          status;

status = DRV_ETHPHY_Status(object);
if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_ETHPHY_Initialize

Function

SYS_STATUS DRV_ETHPHY_Status (SYS_MODULE_OBJ object)

DRV_ETHPHY_Tasks Function

Maintains the driver's state machine and implements its ISR.

Implementation: Dynamic

File

drv_ethphy.h

C

```
void DRV_ETHPHY_Tasks(SYS_MODULE_OBJ object);
```

Returns

None

Description

This function is used to maintain the driver's internal state machine and implement its ISR for interrupt-driven implementations.

Remarks

- This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks)
- This function will never block or access any resources that may cause it to block.

Preconditions

The DRV_ETHPHY_Initialize routine must have been called for the specified Ethernet PHY driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_ETHPHY_Initialize

while (true)
{
    DRV_ETHPHY_Tasks (object);

    // Do other tasks
}
```

Function

void DRV_ETHPHY_Tasks(SYS_MODULE_OBJ object)

DRV_ETHPHY_HWConfigFlagsGet Function

Returns the current Ethernet PHY hardware MII/RMII and ALTERNATE/DEFAULT configuration flags.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_HWConfigFlagsGet(DRV_HANDLE handle, DRV_ETHPHY_CONFIG_FLAGS* pFlags);
```

Returns

DRV_ETHPHY_RES_OK - if the configuration flags successfully stored at pFlags DRV_ETHPHY_RESULT error code otherwise

Description

This function returns the current Ethernet PHY hardware MII/RMII and ALTERNATE/DEFAULT configuration flags from the Device Configuration Fuse bits.

Remarks

None.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_ETHPHY_Open)
pFlags	address to store the hardware configuration

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_HWConfigFlagsGet(DRV_HANDLE handle, DRV_ETHPHY_CONFIG_FLAGS* pFlags)

DRV_ETHPHY_Setup Function

Initializes Ethernet PHY configuration and set up procedure.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_Setup(DRV_HANDLE handle, DRV_ETHPHY_SETUP* pSetUp, TCPIP_ETH_OPEN_FLAGS*
pSetupFlags);
```

Returns

- DRV_ETHPHY_RES_PENDING operation has been scheduled successfully
- an DRV_ETHPHY_RESULT error code if the set up procedure failed.

Description

This function initializes the Ethernet PHY communication. It tries to detect the external Ethernet PHY, to read the capabilities and find a match with the requested features. Then, it programs the Ethernet PHY accordingly.

Remarks

PHY configuration may be a lengthy operation due to active negotiation that the PHY has to perform with the link party. The DRV_ETHPHY_ClientStatus will repeatedly return DRV_ETHPHY_CLIENT_STATUS_BUSY until the set up procedure is complete (unless an error detected at which an error code will be returned immediately).

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_Setup(DRV_HANDLE handle, DRV_ETHPHY_SETUP* pSetUp, TCPIP_ETH_OPEN_FLAGS* pSetupFlags)

b) Client Level Functions

DRV_ETHPHY_ClientStatus Function

Gets the current client-specific status the Ethernet PHY driver.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_CLIENT_STATUS DRV_ETHPHY_ClientStatus(DRV_HANDLE handle);
```

Returns

- DRV_ETHPHY_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the Ethernet PHY driver associated with the given handle.

Remarks

This function will not block for hardware access and will immediately return the current status.

This function has to be used to check that a driver operation has completed. It will return DRV_ETHPHY_CLIENT_STATUS_BUSY when an operation is in progress. It will return DRV_ETHPHY_CLIENT_STATUS_READY when the operation has completed.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE phyHandle;  // Returned from DRV_ETHPHY_Open
DRV_ETHPHY_CLIENT_STATUS phyClientStatus;

phyClientStatus = DRV_ETHPHY_ClientStatus(phyHandle);
if(DRV_ETHPHY_CLIENT_STATUS_ERROR >= phyClientStatus)
{
    // Handle the error
}
```

Function

DRV_ETHPHY_CLIENT_STATUS DRV_ETHPHY_ClientStatus(DRV_HANDLE handle)

DRV_ETHPHY_Close Function

Closes an opened instance of the Ethernet PHY driver.

Implementation: Dynamic

File

drv_ethphy.h

C

```
void DRV_ETHPHY_Close(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the Ethernet PHY driver, invalidating the handle.

Remarks

- After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_ETHPHY_Open before the caller may use the driver again.
- Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_ETHPHY_Initialize routine must have been called for the specified Ethernet PHY driver instance.

DRV_ETHPHY_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_ETHPHY_Open

DRV_ETHPHY_Close(handle);
```

Function

void DRV_ETHPHY_Close(DRV_HANDLE handle)

DRV_ETHPHY_NegotiationIsComplete Function

Returns the results of a previously initiated Ethernet PHY negotiation.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_NegotiationIsComplete(DRV_HANDLE handle, bool waitComplete);
```

Returns

- DRV_ETHPHY_RES_PENDING operation is ongoing
- an DRV_ETHPHY_RESULT error code if the procedure failed.

Description

This function returns the results of a previously initiated Ethernet PHY negotiation.

Remarks

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

When operation is completed but negotiation has failed, DRV_ETHPHY_ClientOperationResult will return:

- DRV_ETHPHY_RES_NEGOTIATION_INACTIVE if no negotiation in progress
- DRV_ETHPHY_RES_NEGOTIATION_NOT_STARTED if negotiation not yet started yet (means time out if waitComplete was requested)
- DRV_ETHPHY_RES_NEGOTIATION_ACTIVE if negotiation ongoing

(means time out if waitComplete was requested).

See also DRV_ETHPHY_NegotiationResultGet.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY
- DRV_ETHPHY_RestartNegotiation should have been called.

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_NegotiationIsComplete(DRV_HANDLE handle, bool waitComplete)

DRV_ETHPHY_Open Function

Opens the specified Ethernet PHY driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_HANDLE DRV_ETHPHY_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

- valid open-instance handle if successful (a number identifying both the caller and the module instance).
- DRV_HANDLE_INVALID if an error occurs

Description

This function opens the specified Ethernet PHY driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The handle returned is valid until the DRV_ETHPHY_Close routine is called.

This function will NEVER block waiting for hardware.

The intent parameter is not used. The PHY driver implements a non-blocking behavior.

Preconditions

The DRV_ETHPHY_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_ETHPHY_Open(DRV_ETHPHY_INDEX_0, 0);
```

```
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Function

DRV_HANDLE DRV_ETHPHY_Open(const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT intent)

DRV_ETHPHY_Reset Function

Immediately resets the Ethernet PHY.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_Reset(DRV_HANDLE handle, bool waitComplete);
```

Returns

- DRV_ETHPHY_RES_PENDING for ongoing, in progress operation
- DRV_ETHPHY_RES_OPERATION_ERR - invalid parameter or operation in the current context

Description

This function immediately resets the Ethernet PHY, optionally waiting for a reset to complete.

Remarks

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

When operation is completed but failed, DRV_ETHPHY_ClientOperationResult will return:

- DRV_ETHPHY_RES_DTCT_ERR if the PHY failed to respond

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_Reset(DRV_HANDLE handle, bool waitComplete)

DRV_ETHPHY_RestartNegotiation Function

Restarts auto-negotiation of the Ethernet PHY link.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_RestartNegotiation(DRV_HANDLE handle);
```

Returns

- DRV_ETHPHY_RES_PENDING operation has been scheduled successfully
- an DRV_ETHPHY_RESULT error code if the procedure failed.

Description

This function restarts auto-negotiation of the Ethernet PHY link.

Remarks

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_RestartNegotiation(DRV_HANDLE handle)

DRV_ETHPHY_ClientOperationAbort Function

Aborts a current client operation initiated by the Ethernet PHY driver.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_ClientOperationAbort(DRV_HANDLE handle);
```

Returns

- DRV_ETHPHY_RESULT value describing the current operation result: DRV_ETHPHY_RES_OK for success; operation has been aborted an DRV_ETHPHY_RESULT error code if the operation failed.

Description

Aborts a current client operation initiated by the Ethernet PHY driver.

Remarks

None

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid opened device handle.
- A driver operation was started

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_ClientOperationAbort(DRV_HANDLE handle)

DRV_ETHPHY_ClientOperationResult Function

Gets the result of a client operation initiated by the Ethernet PHY driver.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_ClientOperationResult(DRV_HANDLE handle);
```

Returns

- DRV_ETHPHY_RESULT value describing the current operation result: DRV_ETHPHY_RES_OK for success; operation has been completed successfully DRV_ETHPHY_RES_PENDING operation is in progress an DRV_ETHPHY_RESULT error code if the operation failed.

Description

Returns the result of a client operation initiated by the Ethernet PHY driver.

Remarks

This function will not block for hardware access and will immediately return the current status.

This function returns the result of the last driver operation. It will return DRV_ETHPHY_RES_PENDING if an operation is still in progress. Otherwise a DRV_ETHPHY_RESULT describing the operation outcome.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid opened device handle.
- A driver operation was started and completed

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_ClientOperationResult(DRV_HANDLE handle)

c) SMI/MIIM Functions

DRV_ETHPHY_SMIScanStatusGet Function

Gets the status of the SMI/MIIM scan data.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanStatusGet(DRV_HANDLE handle);
```

Returns

DRV_ETHPHY_RES_OPERATION_ERR - no scan operation currently in progress

DRV_ETHPHY_RES_OK - scan data is available

DRV_ETHPHY_RES_PENDING - scan data is not yet available

< 0 - an error has occurred and the operation could not be completed

Description

This function gets the status of the SMI/MIIM scan data.

Remarks

None.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY
- DRV_ETHPHY_SMIScanStart() has been called.

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanStatusGet(DRV_HANDLE handle)

DRV_ETHPHY_SMIScanStop Function

Stops the scan of a previously requested SMI/MIIM register.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanStop(DRV_HANDLE handle);
```

Returns

DRV_ETHPHY_RES_OPERATION_ERR - no scan operation currently in progress

DRV_ETHPHY_RES_OK - the scan transaction has been stopped successfully < 0 - an error has occurred and the operation could not be completed

Description

This function stops the current scan of a SMI/MIIM register.

Remarks

None.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY
- DRV_ETHPHY_SMIScanStart was called to start a scan

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanStop(DRV_HANDLE handle)

DRV_ETHPHY_SMIClockSet Function

Sets the SMI/MIIM interface clock.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIClockSet(DRV_HANDLE handle, uint32_t hostClock, uint32_t maxSMIClock);
```

Returns

DRV_ETHPHY_RES_HANDLE_ERR - passed in handle was invalid

DRV_ETHPHY_RES_OK - operation successful

Description

This function sets SMI/MIIM interface clock base on host clock and maximum supported SMI/MIIM interface clock speed.

Remarks

None.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIClockSet(DRV_HANDLE handle,

uint32_t hostClock,

uint32_t maxSMIClock)

DRV_ETHPHY_SMIScanStart Function

Starts the scan of a requested SMI/MIIM register.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanStart(DRV_HANDLE handle, unsigned int rIx);
```

Returns

DRV_ETHPHY_RES_PENDING - the scan transaction was initiated and is ongoing < 0 - an error has occurred and the operation could not be completed

Description

This function starts the scan of a requested SMI/MIIM register.

Remarks

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

However, the client status will always be DRV_ETHPHY_CLIENT_STATUS_BUSY and the client result will always show DRV_ETHPHY_RES_PENDING for as long as the scan is active. Use DRV_ETHPHY_SMIScanStop() to stop a scan in progress. Use DRV_ETHPHY_SMIScanStatusGet() to check is there is scan data available. Use DRV_ETHPHY_SMIScanDataGet() to retrieve the scan data.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanStart(DRV_HANDLE handle,

unsigned int rIx)

DRV_ETHPHY_SMIRead Function

Initiates a SMI/MIIM read transaction.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIRead(DRV_HANDLE handle, unsigned int rIx, uint16_t* pSmiRes, int phyAdd);
```

Returns

DRV_ETHPHY_RES_PENDING - the transaction was initiated and is ongoing < 0 - an error has occurred and the operation could not be completed

Description

This function initiates a SMI/MIIM read transaction for a given PHY register.

Remarks

In most situations the PHY address to be used for this function should be the one returned by DRV_ETHPHY_PhyAddressGet(). However this function allows using a different PHY address for advanced operation.

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid opened device handle.

- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIRead(DRV_HANDLE handle, unsigned int rIx, uint16_t* pSmiRes, int phyAdd)

DRV_ETHPHY_SMIScanDataGet Function

Gets the latest SMI/MIIM scan data result.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanDataGet(DRV_HANDLE handle, uint16_t* pScanRes);
```

Returns

DRV_ETHPHY_RES_OPERATION_ERR - no scan operation currently in progress

DRV_ETHPHY_RES_OK - scan data is available and stored at pScanRes DRV_ETHPHY_RES_PENDING - scan data is not yet available

< 0 - an error has occurred and the operation could not be completed

Description

This function gets the latest SMI/MIIM scan data result.

Remarks

None.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY
- DRV_ETHPHY_SMIScanStart() has been called
- Data is available if DRV_ETHPHY_SMIScanStatusGet() previously returned DRV_ETHPHY_RES_OK

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIScanDataGet(DRV_HANDLE handle, uint16_t* pScanRes)

DRV_ETHPHY_SMIStatus Function

Returns the current status of the SMI/MIIM interface.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIStatus(DRV_HANDLE handle);
```

Returns

- DRV_ETHPHY_RES_BUSY - if the SMI/MIIM interface is busy
- DRV_ETHPHY_RES_OK - if the SMI/MIIM is not busy

< 0 - an error has occurred and the operation could not be completed

Description

This function checks if the SMI/MIIM interface is busy with a transaction.

Remarks

This function is info only and returns the momentary status of the SMI bus. Even if the bus is free there is no guarantee it will be free later on especially if the driver is on going some operation.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIStatus(DRV_HANDLE handle)

DRV_ETHPHY_SMIWrite Function

Initiates a SMI/MIIM write transaction.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_SMIWrite(DRV_HANDLE handle, unsigned int rIx, uint16_t wData, int phyAdd, bool waitComplete);
```

Returns

DRV_ETHPHY_RES_OK - the write transaction has been scheduled/completed successfully DRV_ETHPHY_RES_PENDING - the transaction was initiated and is ongoing < 0 - an error has occurred and the operation could not be completed

Description

This function initiates a SMI/MIIM write transaction for a given PHY register.

Remarks

In most situations the PHY address to be used for this function should be the one returned by DRV_ETHPHY_PhyAddressGet(). However this function allows using a different PHY address for advanced operation.

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_SMIWrite(DRV_HANDLE handle, unsigned int rIx, uint16_t wData, int phyAdd, bool waitComplete)

d) Vendor Functions

DRV_ETHPHY_VendorDataGet Function

Returns the current value of the vendor data.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_VendorDataGet(DRV_HANDLE handle, uint32_t* pVendorData);
```

Returns

DRV_ETHPHY_RES_OK - if the vendor data is stored at the pVendorData address

DRV_ETHPHY_RES_HANDLE_ERR - handle error

Description

This function returns the current value of the vendor data. Each DRV_ETHPHY client object maintains data that could be used for vendor specific operations. This routine allows retrieving of the vendor specific data.

Remarks

The PHY driver will clear the vendor specific data before any call to a vendor specific routine. Otherwise the PHY driver functions do not touch this value.

The DRV_ETHPHY_VendorDataSet can be used for writing data into this field.

Currently only a 32 bit value is supported.

The function is intended for implementing vendor specific functions, like DRV_EXTPHY_MIIConfigure and DRV_EXTPHY_MDIXConfigure, that need a way of maintaining their own data and state machine.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_VendorDataGet(DRV_HANDLE handle, uint32_t* pVendorData)

DRV_ETHPHY_VendorDataSet Function

Returns the current value of the vendor data.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_VendorDataSet(DRV_HANDLE handle, uint32_t vendorData);
```

Returns

DRV_ETHPHY_RES_OK - if the vendor data is stored in the client object

DRV_ETHPHY_RES_HANDLE_ERR - handle error

Description

This function returns the current value of the vendor data. Each DRV_ETHPHY client object maintains data that could be used for vendor specific operations. This routine allows retrieving of the vendor specific data.

Remarks

The PHY driver will clear the vendor specific data before any call to a vendor specific routine. Otherwise the PHY driver functions do not touch this value.

The DRV_ETHPHY_VendorDataGet can be used for reading data into this field.

Currently only a 32 bit value is supported.

The function is intended for implementing vendor specific functions, like DRV_EXTPHY_MIIConfigure and DRV_EXTPHY_MDIXConfigure, that need a way of maintaining their own data and state machine.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_VendorDataSet(DRV_HANDLE handle, uint32_t vendorData)

DRV_ETHPHY_VendorSMIReadResultGet Function

Reads the result of a previous vendor initiated SMI read transfer with DRV_ETHPHY_VendorSMIReadStart.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_VendorSMIReadResultGet(DRV_HANDLE handle, uint16_t* pSmiRes);
```

Returns

DRV_ETHPHY_RES_OK - transaction complete and result deposited at pSmiRes.

DRV_ETHPHY_RES_PENDING - if the vendor transaction is still ongoing The call needs to be retried.

< 0 - some error and the DRV_EXTPHY_MIIConfigure/DRV_EXTPHY_MDIXConfigure has to return error to be aborted by the DRV_ETHPHY_Setup

Description

This function will return the data of a SMI read transfer.

Remarks

The function is intended for implementing vendor SMI transfers within DRV_EXTPHY_MIIConfigure and DRV_EXTPHY_MDIXConfigure.

It has to be called from within the DRV_EXTPHY_MIIConfigure or DRV_EXTPHY_MDIXConfigure functions (which are called, in turn, by the DRV_ETHPHY_Setup procedure) otherwise the call will fail.

The DRV_ETHPHY_RES_OK and DRV_ETHPHY_RES_PENDING significance is changed from the general driver API.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup is in progress and configures the PHY
- The vendor implementation of the DRV_EXTPHY_MIIConfigure/DRV_EXTPHY_MDIXConfigure is running and a SMI transfer is needed
- DRV_ETHPHY_VendorSMIReadStart should have been called to initiate a transfer

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_VendorSMIReadResultGet(DRV_HANDLE handle, uint16_t* pSmiRes)

DRV_ETHPHY_VendorSMIReadStart Function

Starts a vendor SMI read transfer. Data will be available with DRV_ETHPHY_VendorSMIReadResultGet.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_VendorSMIReadStart(DRV_HANDLE handle, uint16_t rIx, int phyAddress);
```

Returns

DRV_ETHPHY_RES_OK - the vendor transaction is started DRV_ETHPHY_VendorSMIReadResultGet() needs to be called for the transaction to complete and to retrieve the result

DRV_ETHPHY_RES_PENDING - the SMI bus is busy and the call needs to be retried

< 0 - some error and the DRV_EXTPHY_MIIConfigure/DRV_EXTPHY_MDIXConfigure has to return error to be aborted by the DRV_ETHPHY_Setup

Description

This function will start a SMI read transfer.

Remarks

The function is intended for implementing vendor SMI transfers within DRV_EXTPHY_MIIConfigure and DRV_EXTPHY_MDIXConfigure.

It has to be called from within the DRV_EXTPHY_MIIConfigure or DRV_EXTPHY_MDIXConfigure functions (which are called, in turn, by the DRV_ETHPHY_Setup procedure) otherwise the call will fail.

The DRV_ETHPHY_RES_OK and DRV_ETHPHY_RES_PENDING significance is changed from the general driver API.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup is in progress and configures the PHY
- The vendor implementation of the DRV_EXTPHY_MIIConfigure/DRV_EXTPHY_MDIXConfigure is running and a SMI transfer is needed

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_VendorSMIReadStart(DRV_HANDLE handle, uint16_t rIx, int phyAddress)

DRV_ETHPHY_VendorSMIWriteStart Function

Starts a vendor SMI write transfer.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_VendorSMIWriteStart(DRV_HANDLE handle, uint16_t rIx, uint16_t wData, int phyAddress);
```

Returns

DRV_ETHPHY_RES_OK - if the vendor SMI write transfer is started

DRV_ETHPHY_RES_PENDING - the SMI bus was busy and the call needs to be retried

< 0 - some error and the DRV_EXTPHY_MIIConfigure/DRV_EXTPHY_MDIXConfigure has to return error to be aborted by the DRV_ETHPHY_Setup

Description

This function will start a SMI write transfer.

Remarks

The function is intended for implementing vendor SMI transfers within DRV_EXTPHY_MIIConfigure and DRV_EXTPHY_MDIXConfigure.

It has to be called from within the DRV_EXTPHY_MIIConfigure or DRV_EXTPHY_MDIXConfigure functions (which are called, in turn, by the DRV_ETHPHY_Setup procedure) otherwise the call will fail.

The DRV_ETHPHY_RES_OK and DRV_ETHPHY_RES_PENDING significance is changed from the general driver API.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup is in progress and configures the PHY
- The vendor implementation of the DRV_EXTPHY_MIIConfigure/DRV_EXTPHY_MDIXConfigure is running and a SMI transfer is needed

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_VendorSMIWriteStart(DRV_HANDLE handle, uint16_t rIx, uint16_t wData, int phyAddress)

e) Other Functions

f) Data Types and Constants

DRV_ETHPHY_CLIENT_STATUS Enumeration

Identifies the client-specific status of the Ethernet PHY driver.

File

drv_ethphy.h

C

```
typedef enum {
  DRV_ETHPHY_CLIENT_STATUS_ERROR,
  DRV_ETHPHY_CLIENT_STATUS_CLOSED,
  DRV_ETHPHY_CLIENT_STATUS_BUSY,
  DRV_ETHPHY_CLIENT_STATUS_READY
} DRV_ETHPHY_CLIENT_STATUS;
```

Members

Members	Description
DRV_ETHPHY_CLIENT_STATUS_ERROR	Unspecified error condition
DRV_ETHPHY_CLIENT_STATUS_CLOSED	Client is not open
DRV_ETHPHY_CLIENT_STATUS_BUSY	An operation is currently in progress
DRV_ETHPHY_CLIENT_STATUS_READY	Up and running, no operations running

Description

Ethernet PHY Driver Client Status

This enumeration identifies the client-specific status of the Ethernet PHY driver.

Remarks

None.

DRV_ETHPHY_INIT Structure

Contains all the data necessary to initialize the Ethernet PHY device.

File

drv_ethphy.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  ETH_MODULE_ID ethphyId;
  const DRV_ETHPHY_OBJECT* pPhyObject;
} DRV_ETHPHY_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
ETH_MODULE_ID ethphyId;	Identifies peripheral (PLIB-level) ID
const DRV_ETHPHY_OBJECT* pPhyObject;	Non-volatile pointer to the PHY object providing vendor functions for this PHY

Description

Ethernet PHY Device Driver Initialization Data

This data structure contains all the data necessary to initialize the Ethernet PHY device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_ETHPHY_Initialize routine.

DRV_ETHPHY_NEGOTIATION_RESULT Structure

Contains all the data necessary to get the Ethernet PHY negotiation result

File

drv_ethphy.h

C

```
typedef struct {
  DRV_ETHPHY_LINK_STATUS linkStatus;
  TCPIP_ETH_OPEN_FLAGS linkFlags;
  TCPIP_ETH_PAUSE_TYPE pauseType;
} DRV_ETHPHY_NEGOTIATION_RESULT;
```

Members

Members	Description
DRV_ETHPHY_LINK_STATUS linkStatus;	link status after a completed negotiation
TCPIP_ETH_OPEN_FLAGS linkFlags;	the negotiation result flags
TCPIP_ETH_PAUSE_TYPE pauseType;	pause type supported by the link partner

Description

Ethernet PHY Device Driver Negotiation result Data

Contains all the data necessary to get the Ethernet PHY negotiation result

Remarks

A pointer to a structure of this format must be passed into the DRV_ETHPHY_NegotiationResultGet routine.

DRV_ETHPHY_SETUP Structure

Contains all the data necessary to set up the Ethernet PHY device.

File

drv_ethphy.h

C

```
typedef struct {
  int phyAddress;
  TCPIP_ETH_OPEN_FLAGS openFlags;
  DRV_ETHPHY_CONFIG_FLAGS configFlags;
  TCPIP_ETH_PAUSE_TYPE macPauseType;
} DRV_ETHPHY_SETUP;
```

Members

Members	Description
int phyAddress;	the address the PHY is configured for
TCPIP_ETH_OPEN_FLAGS openFlags;	the capability flags: FD/HD, 100/100Mbps, etc.
DRV_ETHPHY_CONFIG_FLAGS configFlags;	configuration flags: MII/RMII, I/O setup
TCPIP_ETH_PAUSE_TYPE macPauseType;	MAC requested pause type

Description

Ethernet PHY Device Driver Set up Data

This data structure contains all the data necessary to configure the Ethernet PHY device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_ETHPHY_Setup routine.

DRV_ETHPHY_VENDOR_MDIX_CONFIGURE Type

Pointer to function that configures the MDIX mode for the Ethernet PHY.

File

drv_ethphy.h

C

```
typedef DRV_ETHPHY_RESULT (* DRV_ETHPHY_VENDOR_MDIX_CONFIGURE)(const struct DRV_ETHPHY_OBJECT_BASE_TYPE*
pBaseObj, DRV_HANDLE handle, TCPIP_ETH_OPEN_FLAGS oFlags);
```

Returns

- DRV_ETHPHY_RES_OK - if success, operation complete
- DRV_ETHPHY_RES_PENDING - if function needs to be called again

< 0 - on failure: configuration not supported or some other error

Description

Pointer To Function: typedef DRV_ETHPHY_RESULT (* DRV_ETHPHY_VENDOR_MDIX_CONFIGURE) (const struct DRV_ETHPHY_OBJECT_BASE_TYPE* pBaseObj, DRV_HANDLE handle, TCPIP_ETH_OPEN_FLAGS oFlags);

This type describes a pointer to a function that configures the MDIX mode for the Ethernet PHY. This configuration function is PHY specific and every PHY driver has to provide their own implementation.

Remarks

The PHY driver consists of 2 modules:

- the main/base PHY driver which uses standard IEEE PHY registers
- the vendor specific functionality

This function provides vendor specific functionality. Every PHY driver has to expose this vendor specific function as part of its interface.

Traditionally the name used for this function is DRV_EXTPHY_MDIXConfigure but any name can be used.

The function can use all the vendor specific functions to store/retrieve specific data or start SMI transactions (see Vendor Interface Routines).

The function should not block but return DRV_ETHPHY_RES_PENDING if waiting for SMI transactions.

Preconditions

Communication to the PHY should have been established.

DRV_ETHPHY_VENDOR_MII_CONFIGURE Type

Pointer to function to configure the Ethernet PHY in one of the MII/RMII operation modes.

File

drv_ethphy.h

C

```
typedef DRV_ETHPHY_RESULT (* DRV_ETHPHY_VENDOR_MII_CONFIGURE)(const struct DRV_ETHPHY_OBJECT_BASE_TYPE*
pBaseObj, DRV_HANDLE handle, DRV_ETHPHY_CONFIG_FLAGS cFlags);
```

Returns

- DRV_ETHPHY_RES_OK - if success, operation complete
- DRV_ETHPHY_RES_PENDING - if function needs to be called again

< 0 - on failure: configuration not supported or some other error

Description

Pointer To Function: typedef DRV_ETHPHY_RESULT (* DRV_ETHPHY_VENDOR_MII_CONFIGURE) (const struct DRV_ETHPHY_OBJECT_BASE_TYPE* pBaseObj, DRV_HANDLE handle, DRV_ETHPHY_CONFIG_FLAGS cFlags);

This type describes a pointer to a function that configures the Ethernet PHY in one of the MII/RMII operation modes. This configuration function is PHY specific and every PHY driver has to provide their own implementation.

Remarks

The PHY driver consists of 2 modules:

- the main/base PHY driver which uses standard IEEE PHY registers
- the vendor specific functionality

This function provides vendor specific functionality. Every PHY driver has to expose this vendor specific function as part of its interface.

Traditionally the name used for this function is DRV_EXTPHY_MIIConfigure but any name can be used.

The PHY driver will call the vendor set up functions after the communication to the PHY has been established.

The function can use all the vendor specific functions to store/retrieve specific data or start SMI transactions (see Vendor Interface Routines).

The function should not block but return DRV_ETHPHY_RES_PENDING if waiting for SMI transactions.

Preconditions

Communication to the PHY should have been established.

DRV_ETHPHY_VENDOR_SMI_CLOCK_GET Type

Pointer to a function to return the SMI/MIIM maximum clock speed in Hz of the Ethernet PHY.

File

drv_ethphy.h

C

```
typedef unsigned int (* DRV_ETHPHY_VENDOR_SMI_CLOCK_GET)(const struct DRV_ETHPHY_OBJECT_BASE_TYPE*
pBaseObj, DRV_HANDLE handle);
```

Returns

The maximum SMI/MIIM clock speed as an unsigned integer.

Description

Pointer to Function: typedef unsigned int (* DRV_ETHPHY_VENDOR_SMI_CLOCK_GET) (const struct DRV_ETHPHY_OBJECT_BASE_TYPE* pBaseObj, DRV_HANDLE handle);

This type describes a pointer to a function that returns the SMI/MIIM maximum clock speed in Hz of the Ethernet PHY. This configuration function is PHY specific and every PHY driver has to provide their own implementation.

Remarks

The PHY driver consists of 2 modules:

- the main/base PHY driver which uses standard IEEE PHY registers
- the vendor specific functionality

This function provides vendor specific functionality. Every PHY driver has to expose this vendor specific function as part of its interface.

This value is PHY specific. All PHYs are requested to support 2.5 MHz.

Traditionally the name used for this function is DRV_EXTPHY_SMIClockGet but any name can be used.

The PHY driver will call the vendor set up functions after the communication to the PHY has been established.

The function should not block but return immediately. The function cannot start SMI transactions and cannot use the vendor specific functions to store/retrieve specific data (see Vendor Interface Routines).

Preconditions

Communication to the PHY should have been established.

DRV_ETHPHY_INDEX_0 Macro

Ethernet PHY driver index definitions.

File

drv_ethphy.h

C

```
#define DRV_ETHPHY_INDEX_0 0
```

Description

Ethernet PHY Driver Module Index Numbers

These constants provide the Ethernet PHY driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_ETHPHY_Initialize and DRV_ETHPHY_Open routines to identify the driver instance in use.

DRV_ETHPHY_INDEX_1 Macro

File

drv_ethphy.h

C

```
#define DRV_ETHPHY_INDEX_1 1
```

Description

This is macro DRV_ETHPHY_INDEX_1.

DRV_ETHPHY_INDEX_COUNT Macro

Number of valid Ethernet PHY driver indices.

File

drv_ethphy.h

C

```
#define DRV_ETHPHY_INDEX_COUNT 1
```

Description

Ethernet PHY Driver Module Index Count

This constant identifies the number of valid Ethernet PHY driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from part-specific header files defined as part of the peripheral libraries.

DRV_ETHPHY_LinkStatusGet Function

Returns the current link status.

Implementation: Dynamic

File

drv_ethphy.h

C

```
DRV_ETHPHY_RESULT DRV_ETHPHY_LinkStatusGet(DRV_HANDLE handle, DRV_ETHPHY_LINK_STATUS* pLinkStat, bool
refresh);
```

Returns

- DRV_ETHPHY_RES_PENDING for ongoing, in progress operation
- an DRV_ETHPHY_RESULT error code if the link status get procedure failed.

Description

This function returns the current link status.

Remarks

This function reads the Ethernet PHY to get current link status. If refresh is specified then, if the link is down a second read will be performed to return the current link status.

Use DRV_ETHPHY_ClientStatus() and DRV_ETHPHY_ClientOperationResult() to check when the operation was completed and its outcome.

Preconditions

- The DRV_ETHPHY_Initialize routine must have been called.
- DRV_ETHPHY_Open must have been called to obtain a valid device handle.
- DRV_ETHPHY_Setup must have been called to properly configure the PHY

Example

Function

DRV_ETHPHY_RESULT DRV_ETHPHY_LinkStatusGet(DRV_HANDLE handle, DRV_ETHPHY_LINK_STATUS* pLinkStat, bool refresh)

DRV_ETHPHY_LINK_STATUS Enumeration

Defines the possible status flags of PHY Ethernet link.

File

drv_ethphy.h

C

```
typedef enum {
  DRV_ETHPHY_LINK_ST_DOWN,
  DRV_ETHPHY_LINK_ST_UP,
  DRV_ETHPHY_LINK_ST_LP_NEG_UNABLE,
  DRV_ETHPHY_LINK_ST_REMOTE_FAULT,
  DRV_ETHPHY_LINK_ST_PDF,
  DRV_ETHPHY_LINK_ST_LP_PAUSE,
  DRV_ETHPHY_LINK_ST_LP_ASM_DIR,
  DRV_ETHPHY_LINK_ST_NEG_TMO,
  DRV_ETHPHY_LINK_ST_NEG_FATAL_ERR
} DRV_ETHPHY_LINK_STATUS;
```

Members

Members	Description
DRV_ETHPHY_LINK_ST_DOWN	No connection to the LinkPartner
DRV_ETHPHY_LINK_ST_UP	Link is up
DRV_ETHPHY_LINK_ST_LP_NEG_UNABLE	LP non negotiation able
DRV_ETHPHY_LINK_ST_REMOTE_FAULT	LP fault during negotiation
DRV_ETHPHY_LINK_ST_PDF	Parallel Detection Fault encountered (when DRV_ETHPHY_LINK_ST_LP_NEG_UNABLE)
DRV_ETHPHY_LINK_ST_LP_PAUSE	LP supports symmetric pause
DRV_ETHPHY_LINK_ST_LP_ASM_DIR	LP supports asymmetric TX/RX pause operation
DRV_ETHPHY_LINK_ST_NEG_TMO	LP not there
DRV_ETHPHY_LINK_ST_NEG_FATAL_ERR	An unexpected fatal error occurred during the negotiation

Description

Ethernet PHY Device Link Status Codes

This enumeration defines the flags describing the status of the PHY Ethernet link.

Remarks

Multiple flags can be set.

DRV_ETHPHY_CONFIG_FLAGS Enumeration

Defines the possible results of Ethernet operations that can succeed or fail

File

drv_ethphy.h

C

```
typedef enum {
  DRV_ETHPHY_CFG_RMII,
  DRV_ETHPHY_CFG_MII,
  DRV_ETHPHY_CFG_ALTERNATE,
  DRV_ETHPHY_CFG_DEFAULT,
  DRV_ETHPHY_CFG_AUTO
} DRV_ETHPHY_CONFIG_FLAGS;
```

Members

Members	Description
DRV_ETHPHY_CFG_RMII	RMII data interface in configuration fuses.
DRV_ETHPHY_CFG_MII	MII data interface in configuration fuses.
DRV_ETHPHY_CFG_ALTERNATE	Configuration fuses is ALT
DRV_ETHPHY_CFG_DEFAULT	Configuration fuses is DEFAULT

DRV_ETHPHY_CFG_AUTO	Use the fuses configuration to detect if you are RMII/MII and ALT/DEFAULT configuration

Description

Ethernet PHY Driver Operation Result *

PHY Driver Operation Result Codes

This enumeration defines the possible results of any of the PHY driver operations that have the possibility of failing. This result should be checked to ensure that the operation achieved the desired result.

DRV_ETHPHY_OBJECT Structure

Identifies the interface of a Ethernet PHY vendor driver.

File

drv_ethphy.h

C

```
typedef struct {
  DRV_ETHPHY_VENDOR_MII_CONFIGURE miiConfigure;
  DRV_ETHPHY_VENDOR_MDIX_CONFIGURE mdixConfigure;
  DRV_ETHPHY_VENDOR_SMI_CLOCK_GET smiClockGet;
  DRV_ETHPHY_VENDOR_WOL_CONFIGURE wolConfigure;
} DRV_ETHPHY_OBJECT;
```

Members

Members	Description
DRV_ETHPHY_VENDOR_MII_CONFIGURE miiConfigure;	PHY driver function to configure the operation mode: MII/RMII
DRV_ETHPHY_VENDOR_MDIX_CONFIGURE mdixConfigure;	PHY driver function to configure the MDIX mode
DRV_ETHPHY_VENDOR_SMI_CLOCK_GET smiClockGet;	PHY driver function to get the SMI clock rate
DRV_ETHPHY_VENDOR_WOL_CONFIGURE wolConfigure;	PHY driver function to configure the WOL functionality

Description

Ethernet PHY Driver Vendor Object

This data structure identifies the required interface of the Ethernet PHY driver. Any PHY vendor driver has to export this interface.

Remarks

The PHY driver consists of 2 modules:

- the main/base PHY driver which uses standard IEEE PHY registers
- the vendor specific functionality

This object provides vendor specific functionality. Every PHY driver has to expose this vendor specific functionality as part of its interface.

DRV_ETHPHY_VENDOR_WOL_CONFIGURE Type

Pointer to a function to configure the PHY WOL functionality

File

drv_ethphy.h

C

```
typedef void (* DRV_ETHPHY_VENDOR_WOL_CONFIGURE)(const struct DRV_ETHPHY_OBJECT_BASE_TYPE* pBaseObj,
DRV_HANDLE handle, unsigned char bAddr[]);
```

Returns

None

Description

Pointer to Function: typedef void (* DRV_ETHPHY_VENDOR_WOL_CONFIGURE) (const struct DRV_ETHPHY_OBJECT_BASE_TYPE* pBaseObj, DRV_HANDLE handle, unsigned char bAddr[]);

This type describes a pointer to a function that configures the PHY WOL functionality of the Ethernet PHY. Configures the WOL of the PHY with a

Source MAC address or a 6 byte magic packet mac address.

This configuration function is PHY specific and every PHY driver has to provide their own implementation.

Remarks

The PHY driver consists of 2 modules:

- the main/base PHY driver which uses standard IEEE PHY registers
- the vendor specific functionality

This function provides vendor specific functionality. Every PHY driver has to expose this vendor specific function as part of its interface.

Traditionally the name used for this function is DRV_EXTPHY_WOLConfiguration but any name can be used.

The PHY driver will call the vendor set up functions after the communication to the PHY has been established.

The function can use all the vendor specific functions to store/retrieve specific data or start SMI transactions (see Vendor Interface Routines).

The function should not block but return DRV_ETHPHY_RES_PENDING if waiting for SMI transactions.

This feature is not currently supported for all PHYs.

Preconditions

Communication to the PHY should have been established.

DRV_ETHPHY_OBJECT_BASE Structure

Identifies the base interface of a Ethernet PHY driver.

File

drv_ethphy.h

C

```
typedef struct DRV_ETHPHY_OBJECT_BASE_TYPE {
  SYS_MODULE_OBJ (* DRV_ETHPHY_Initialize)(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const
init);
  void (* DRV_ETHPHY_Reinitialize)(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
  void (* DRV_ETHPHY_Deinitialize)(SYS_MODULE_OBJ object);
  SYS_STATUS (* DRV_ETHPHY_Status)(SYS_MODULE_OBJ object);
  void (* DRV_ETHPHY_Tasks)(SYS_MODULE_OBJ object);
  DRV_HANDLE (* DRV_ETHPHY_Open)(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
  void (* DRV_ETHPHY_Close)(DRV_HANDLE handle);
  DRV_ETHPHY_CLIENT_STATUS (* DRV_ETHPHY_ClientStatus)(DRV_HANDLE handle);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_ClientOperationResult)(DRV_HANDLE handle);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_ClientOperationAbort)(DRV_HANDLE handle);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIRead)(DRV_HANDLE handle, unsigned int rIx, uint16_t* pSmiRes, int
phyAdd);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIWrite)(DRV_HANDLE handle, unsigned int rIx, uint16_t wData, int
phyAdd, bool waitComplete);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIScanStart)(DRV_HANDLE handle, unsigned int rIx);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIScanStop)(DRV_HANDLE handle);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIScanStatusGet)(DRV_HANDLE handle);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIScanDataGet)(DRV_HANDLE handle, uint16_t* pScanRes);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIStatus)(DRV_HANDLE handle);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_SMIClockSet)(DRV_HANDLE handle, uint32_t hostClock, uint32_t maxSMIClock);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_PhyAddressGet)(DRV_HANDLE handle, int* pPhyAddress);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_Setup)(DRV_HANDLE handle, DRV_ETHPHY_SETUP* pSetUp, TCPIP_ETH_OPEN_FLAGS*
pSetupFlags);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_RestartNegotiation)(DRV_HANDLE handle);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_HWConfigFlagsGet)(DRV_HANDLE handle, DRV_ETHPHY_CONFIG_FLAGS* pFlags);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_NegotiationIsComplete)(DRV_HANDLE handle, bool waitComplete);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_NegotiationResultGet)(DRV_HANDLE handle, DRV_ETHPHY_NEGOTIATION_RESULT*
pNegResult);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_LinkStatusGet)(DRV_HANDLE handle, DRV_ETHPHY_LINK_STATUS* pLinkStat, bool
refresh);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_Reset)(DRV_HANDLE handle, bool waitComplete);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_VendorDataGet)(DRV_HANDLE handle, uint32_t* pVendorData);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_VendorDataSet)(DRV_HANDLE handle, uint32_t vendorData);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_VendorSMIReadStart)(DRV_HANDLE handle, uint16_t rIx, int phyAddress);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_VendorSMIReadResultGet)(DRV_HANDLE handle, uint16_t* pSmiRes);
  DRV_ETHPHY_RESULT (* DRV_ETHPHY_VendorSMIWriteStart)(DRV_HANDLE handle, uint16_t rIx, uint16_t wData, int
phyAddress);
} DRV_ETHPHY_OBJECT_BASE;
```

Description

Ethernet PHY Driver Base Object

This data structure identifies the required interface of the Ethernet PHY driver. Any dynamic PHY driver has to export this interface.

Remarks

The PHY driver consists of 2 modules:

- the main/base PHY driver which uses standard IEEE PHY registers
- the vendor specific functionality

This object provides the base functionality. Every dynamic PHY driver has to expose this basic functionality as part of its interface.

See above the description of each function that's part of the base PHY driver.

Files

Files

Name	Description
drv_ethphy.h	Ethernet ETHPHY Device Driver Interface File
drv_ethphy_config.h	Ethernet PHY driver configuration definitions template.

Description

This section lists the source and header files used by the Ethernet PHY Driver Library.

drv_ethphy.h

Ethernet ETHPHY Device Driver Interface File

Enumerations

	Name	Description
	DRV_ETHPHY_CLIENT_STATUS	Identifies the client-specific status of the Ethernet PHY driver.
	DRV_ETHPHY_CONFIG_FLAGS	Defines the possible results of Ethernet operations that can succeed or fail
	DRV_ETHPHY_LINK_STATUS	Defines the possible status flags of PHY Ethernet link.

Functions

	Name	Description
	DRV_ETHPHY_ClientOperationAbort	Aborts a current client operation initiated by the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_ClientOperationResult	Gets the result of a client operation initiated by the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_ClientStatus	Gets the current client-specific status the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_Close	Closes an opened instance of the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_Deinitialize	Deinitializes the specified instance of the Ethernet PHY driver module. **Implementation:** Dynamic
	DRV_ETHPHY_HWConfigFlagsGet	Returns the current Ethernet PHY hardware MII/RMII and ALTERNATE/DEFAULT configuration flags. **Implementation:** Dynamic
	DRV_ETHPHY_Initialize	Initializes the Ethernet PHY driver. **Implementation:** Dynamic
	DRV_ETHPHY_LinkStatusGet	Returns the current link status. **Implementation:** Dynamic
	DRV_ETHPHY_NegotiationIsComplete	Returns the results of a previously initiated Ethernet PHY negotiation. **Implementation:** Dynamic
	DRV_ETHPHY_NegotiationResultGet	Returns the result of a completed negotiation. **Implementation:** Dynamic
	DRV_ETHPHY_Open	Opens the specified Ethernet PHY driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_ETHPHY_PhyAddressGet	Returns the PHY address. **Implementation:** Dynamic
	DRV_ETHPHY_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings. **Implementation:** Dynamic

	DRV_ETHPHY_Reset	Immediately resets the Ethernet PHY. **Implementation:** Dynamic
	DRV_ETHPHY_RestartNegotiation	Restarts auto-negotiation of the Ethernet PHY link. **Implementation:** Dynamic
	DRV_ETHPHY_Setup	Initializes Ethernet PHY configuration and set up procedure. **Implementation:** Dynamic
	DRV_ETHPHY_SMIClockSet	Sets the SMI/MIIM interface clock. **Implementation:** Dynamic
	DRV_ETHPHY_SMIRead	Initiates a SMI/MIIM read transaction. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanDataGet	Gets the latest SMI/MIIM scan data result. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanStart	Starts the scan of a requested SMI/MIIM register. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanStatusGet	Gets the status of the SMI/MIIM scan data. **Implementation:** Dynamic
	DRV_ETHPHY_SMIScanStop	Stops the scan of a previously requested SMI/MIIM register. **Implementation:** Dynamic
	DRV_ETHPHY_SMIStatus	Returns the current status of the SMI/MIIM interface. **Implementation:** Dynamic
	DRV_ETHPHY_SMIWrite	Initiates a SMI/MIIM write transaction. **Implementation:** Dynamic
	DRV_ETHPHY_Status	Provides the current status of the Ethernet PHY driver module. **Implementation:** Dynamic
	DRV_ETHPHY_Tasks	Maintains the driver's state machine and implements its ISR. **Implementation:** Dynamic
	DRV_ETHPHY_VendorDataGet	Returns the current value of the vendor data. **Implementation:** Dynamic
	DRV_ETHPHY_VendorDataSet	Returns the current value of the vendor data. **Implementation:** Dynamic
	DRV_ETHPHY_VendorSMIReadResultGet	Reads the result of a previous vendor initiated SMI read transfer with DRV_ETHPHY_VendorSMIReadStart. **Implementation:** Dynamic
	DRV_ETHPHY_VendorSMIReadStart	Starts a vendor SMI read transfer. Data will be available with DRV_ETHPHY_VendorSMIReadResultGet. **Implementation:** Dynamic
	DRV_ETHPHY_VendorSMIWriteStart	Starts a vendor SMI write transfer. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_ETHPHY_INDEX_0	Ethernet PHY driver index definitions.
	DRV_ETHPHY_INDEX_1	This is macro DRV_ETHPHY_INDEX_1.
	DRV_ETHPHY_INDEX_COUNT	Number of valid Ethernet PHY driver indices.

Structures

	Name	Description
	DRV_ETHPHY_OBJECT_BASE_TYPE	Identifies the base interface of a Ethernet PHY driver.
	DRV_ETHPHY_INIT	Contains all the data necessary to initialize the Ethernet PHY device.
	DRV_ETHPHY_NEGOTIATION_RESULT	Contains all the data necessary to get the Ethernet PHY negotiation result
	DRV_ETHPHY_OBJECT	Identifies the interface of a Ethernet PHY vendor driver.
	DRV_ETHPHY_OBJECT_BASE	Identifies the base interface of a Ethernet PHY driver.
	DRV_ETHPHY_SETUP	Contains all the data necessary to set up the Ethernet PHY device.

Types

	Name	Description
	DRV_ETHPHY_VENDOR_MDIX_CONFIGURE	Pointer to function that configures the MDIX mode for the Ethernet PHY.

	DRV_ETHPHY_VENDOR_MII_CONFIGURE	Pointer to function to configure the Ethernet PHY in one of the MII/RMII operation modes.
	DRV_ETHPHY_VENDOR_SMI_CLOCK_GET	Pointer to a function to return the SMI/MIIM maximum clock speed in Hz of the Ethernet PHY.
	DRV_ETHPHY_VENDOR_WOL_CONFIGURE	Pointer to a function to configure the PHY WOL functionality

Description

Ethernet ETHPHY Device Driver Interface

The Ethernet ETHPHY device driver provides a simple interface to manage an Ethernet ETHPHY peripheral using MIIM (or SMI) interface. This file defines the interface definitions and prototypes for the Ethernet ETHPHY driver.

File Name

drv_ethphy.h

Company

Microchip Technology Inc.

drv_ethphy_config.h

Ethernet PHY driver configuration definitions template.

Macros

	Name	Description
	DRV_ETHPHY_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ETHPHY_INDEX	Ethernet PHY static index selection.
	DRV_ETHPHY_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_ETHPHY_NEG_DONE_TMO	Value of the PHY negotiation complete time out as per IEEE 802.3 spec.
	DRV_ETHPHY_NEG_INIT_TMO	Value of the PHY negotiation initiation time out as per IEEE 802.3 spec.
	DRV_ETHPHY_PERIPHERAL_ID	Defines an override of the peripheral ID.
	DRV_ETHPHY_RESET_CLR_TMO	Value of the PHY Reset self clear time out as per IEEE 802.3 spec.

Description

Ethernet PHY Driver Configuration Definitions for the Template Version

These definitions statically define the driver's mode of operation.

File Name

drv_ethphy_config.h

Company

Microchip Technology Inc.

Flash Driver Library

This section describes the Flash Driver Library.

Introduction

The Flash Driver Library provides functions that allow low-level interface with the on-chip Flash.

Description

Through MHC, this driver provides low-level functions for writing and erasing sections of the Flash memory.

Flash Program Memory

The Flash Program Memory is readable, writeable, and erasable during normal operation over the entire operating voltage range.

A read from program memory is executed at one byte/word at a time depending on the width of the data bus.

A write to the program memory is executed in either blocks of specific sizes or a single word depending on the type of processor used.

An erase is performed in blocks. A bulk erase may be performed from user code depending on the type of processor supporting the operation.

Writing or erasing program memory will cease instruction fetches until the operation is complete, restricting memory access, and therefore

preventing code execution. This is controlled by an internal programming timer.

Library Interface

Functions

	Name	Description
	DRV_FLASH_ErasePage	Erases a page of Flash. **Implementation:** Static
	DRV_FLASH_GetPageSize	Returns the size in bytes of a single "Page" which can be erased in the flash. **Implementation:** Static
	DRV_FLASH_GetRowSize	Returns the size in bytes of a single "Row" which can be written to the flash. **Implementation:** Static
	DRV_FLASH_Initialize	Initializes the Flash instance for the specified driver index. **Implementation:** Static
	DRV_FLASH_IsBusy	Returns true if the Flash device is still busy writing or is erasing. **Implementation:** Static
	DRV_FLASH_Open	Initializes a channel to the appropriate flash device.
	DRV_FLASH_WriteQuadWord	Writes four 4-byte words to the Flash at the (word-aligned) flashAddr. **Implementation:** Static
	DRV_FLASH_WriteRow	Writes an DRV_FLASH_ROW_SIZE bytes to the Flash at the (word-aligned) flashAddr. **Implementation:** Static
	DRV_FLASH_WriteWord	Writes a 4-byte Word to the Flash at the (word-aligned) flashAddr. **Implementation:** Static

Data Types and Constants

	Name	Description
	DRV_FLASH_INDEX_0	FLASH driver index definitions
	DRV_FLASH_PAGE_SIZE	Specifies the FLASH Driver Program Page Size in bytes.
	DRV_FLASH_ROW_SIZE	Specifies the FLASH Driver Program Row Size in bytes.

Description

This section describes the Application Programming Interface (API) functions of the Flash Driver Library.

Refer to each section for a detailed description.

Functions

DRV_FLASH_ErasePage Function

Erases a page of Flash.

Implementation: Static

File

drv_flash.h

C

```
void DRV_FLASH_ErasePage(const DRV_HANDLE handle, uint32_t flashAddr);
```

Returns

None.

Description

This function starts the process of erasing a page of Flash. It does not wait for the erase operation to be done. That is left to the user. It does not verify that the erase was successful. That is left to the user. It always erases a single page. The size of a page in bytes will vary by device. It will be available in the DRV_FLASH_PAGE_SIZE parameter.

Remarks

Most devices will be running for code stored in the Flash. This means that any erases of the Flash will necessarily be writes to program space. As such, they will prevent the CPU from reading further instructions until the write is done. However, some devices may have more than one Flash

such that it can run from one while writing to another. Additionally, if the application is small enough, it may run out of a cache. In any case, it is up to the user to wait for an operation to complete and or to decide that such a wait is unnecessary.

Preconditions

The flashAddr is taken as a valid Flash address. No range checking occurs. Any previous Flash operations (write or erase) must be completed or this will fail silently. The Flash must be correctly erased at flashAddr.

Example

```
flashAddr = 0x9d008000;
DRV_FLASH_Erase_Page(handle, flashAddr);
```

Function

void DRV_FLASH_Erase_Page(uint32_t flashAddr);

DRV_FLASH_GetPageSize Function

Returns the size in bytes of a single "Page" which can be erased in the flash.

Implementation: Static

File

drv_flash.h

C

```
uint32_t DRV_FLASH_GetPageSize(const DRV_HANDLE handle);
```

Returns

None.

Description

This function allows the user to get the size of a flash Page.

Remarks

None.

Preconditions

None

Function

uint32_t DRV_FLASH_GetPageSize(const DRV_HANDLE handle)

DRV_FLASH_GetRowSize Function

Returns the size in bytes of a single "Row" which can be written to the flash.

Implementation: Static

File

drv_flash.h

C

```
uint32_t DRV_FLASH_GetRowSize(const DRV_HANDLE handle);
```

Returns

None.

Description

This function allows the user to get the size of a flash Row.

Remarks

None.

Preconditions

None

Function

uint32_t DRV_FLASH_GetRowSize(const DRV_HANDLE handle)

DRV_FLASH_Initialize Function

Initializes the Flash instance for the specified driver index.

Implementation: Static

File

drv_flash.h

C

```
SYS_MODULE_OBJ DRV_FLASH_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the Flash Driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

This function must be called before any other Flash function is called. This function should only be called once during system initialization.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_FLASH_Initialize(
const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init
)

DRV_FLASH_IsBusy Function

Returns true if the Flash device is still busy writing or is erasing.

Implementation: Static

File

drv_flash.h

C

```
bool DRV_FLASH_IsBusy(const DRV_HANDLE handle);
```

Returns

- true - Indicates the Flash is busy
- false - Indicates the Flash is not busy

Description

This function checks whether the process of programming a Word into the Flash is still operating.

Remarks

Most devices will be running for code stored in the Flash. This means that any writes to the Flash will necessarily be writes to program space. As such, they will prevent the CPU from reading further instructions until the write is done. However, some devices may have more than one Flash such that it can run from one while writing to another. Additionally, if the application is small enough, it may run out of a cache. In any case, it is up

to the user to wait for an operation to complete and or to decide that such a wait is unnecessary.

Preconditions

None.

Example

```
flashAddr = 0x9d008000;
sourceData = 0x12345678;
DRV_FLASH_Write_Word(flashAddr,  sourceData);
DRV_FLASH_IsBusy( void );
```

Function

bool DRV_FLASH_IsBusy(void)

DRV_FLASH_Open Function

Initializes a channel to the appropriate flash device.

File

drv_flash.h

C

```
DRV_HANDLE DRV_FLASH_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT ioIntent);
```

Returns

Handle for future calls to the driver's operations.

Preconditions

None

Function

DRV_HANDLE DRV_FLASH_Open(
const SYS_MODULE_INDEX index,
const DRV_IO_INTENT ioIntent
);

DRV_FLASH_WriteQuadWord Function

Writes four 4-byte words to the Flash at the (word-aligned) flashAddr.

Implementation: Static

File

drv_flash.h

C

```
void DRV_FLASH_WriteQuadWord(const DRV_HANDLE handle, uint32_t flashAddr, uint32_t * sourceData);
```

Returns

None.

Description

This function starts the process of programming a word into the Flash. It does not wait for the write operation to be done, which is left to the user. It does not verify that the write was successful, which is left to the user.

Remarks

Most devices will be running for code stored in the Flash. This means that any writes to the Flash will necessarily be writes to program space. As such, they will prevent the CPU from reading further instructions until the write is done. However, some devices may have more than one Flash such that it can run from one while writing to another. Additionally, if the application is small enough, it may run out of a cache. In any case, it is up to the user to wait for an operation to complete and or to decide that such a wait is unnecessary.

Preconditions

The flashAddr is taken as a valid Flash address. No range checking occurs. Any previous Flash operations (write or erase) must be completed or

this will fail silently. The Flash must be correctly erased at flashAddr.

Example

```
flashAddr = 0x9d008000;
sourceData[4] = {0x12345678,0x9ABCDEF0,0x55AAAA55,0x11111111};
DRV_FLASH_WriteQuadWord(handle, flashAddr,  sourceData);
```

Function

void DRV_FLASH_WriteQuadWord(const DRV_HANDLE handle, uint32_t flashAddr, uint32_t sourceData)

DRV_FLASH_WriteRow Function

Writes an DRV_FLASH_ROW_SIZE bytes to the Flash at the (word-aligned) flashAddr.

Implementation: Static

File

drv_flash.h

C

```
void DRV_FLASH_WriteRow(const DRV_HANDLE handle, uint32_t flashAddr, uint32_t sourceData);
```

Returns

None.

Description

This function starts the process of programming a buffer into the Flash. It does not wait for the write operation to be done, which is left to the user. It does not verify that the write was successful, which is left to the user.

Remarks

Most devices will be running for code stored in the Flash. This means that any writes to the Flash will necessarily be writes to program space. As such, they will prevent the CPU from reading further instructions until the write is done. However, some devices may have more than one Flash such that it can run from one while writing to another. Additionally, if the application is small enough, it may run out of a cache. In any case, it is up to the user to wait for an operation to complete and or to decide that such a wait is unnecessary.

Preconditions

The flashAddr is taken as a valid Flash address. No range checking occurs. The memory pointed to by sourceData must be valid memory for at least DRV_FLASH_ROW_SIZE bytes. Any previous Flash operations (write or erase) must be completed or this will fail silently. The Flash must be correctly erased at flashAddr.

Example

```
flashAddr = 0x9d008000;
uint32_t dataStore[DRV_FLASH_ROW_SIZE] = {0,1,2,3,4,5};
DRV_FLASH_Write_Row( const DRV_HANDLE handle, flashAddr,  dataStore);
```

Function

void DRV_FLASH_WriteRow(const DRV_HANDLE handle, uint32_t flashAddr, uint32_t sourceData)

DRV_FLASH_WriteWord Function

Writes a 4-byte Word to the Flash at the (word-aligned) flashAddr.

Implementation: Static

File

drv_flash.h

C

```
void DRV_FLASH_WriteWord(const DRV_HANDLE handle, uint32_t flashAddr, uint32_t sourceData);
```

Returns

None.

Description

This function starts the process of programming a Word into the Flash. It does not wait for the write operation to be done, which is left to the user.

It does not verify that the write was successful, which is left to the user.

Remarks

Most devices will be running for code stored in the Flash. This means that any writes to the Flash will necessarily be writes to program space. As such, they will prevent the CPU from reading further instructions until the write is done. However, some devices may have more than one Flash such that it can run from one while writing to another. Additionally, if the application is small enough, it may run out of a cache. In any case, it is up to the user to wait for an operation to complete and or to decide that such a wait is unnecessary.

Preconditions

The flashAddr is taken as a valid Flash address. No range checking occurs. Any previous Flash operations (write or erase) must be completed or this will fail silently. The Flash must be correctly erased at flashAddr.

Example

```
flashAddr = 0x9d008000;
sourceData = 0x12345678;
DRV_FLASH_WriteWord(handle, flashAddr,  sourceData);
```

Function

void DRV_FLASH_WriteWord(const DRV_HANDLE handle, uint32_t flashAddr, uint32_t sourceData)

Data Types and Constants

DRV_FLASH_INDEX_0 Macro

FLASH driver index definitions

File

drv_flash.h

C

```
#define DRV_FLASH_INDEX_0 0
```

Description

These constants provide FLASH driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_FLASH_Initialize and DRV_FLASH_Open routines to identify the driver instance in use.

Section

Constants

****

****

****

Driver FLASH Module Index

DRV_FLASH_PAGE_SIZE Macro

Specifies the FLASH Driver Program Page Size in bytes.

File

drv_flash.h

C

```
#define DRV_FLASH_PAGE_SIZE (NVM_PAGE_SIZE)
```

Description

FLASH Driver Program Page Size.

This definition specifies the FLASH Driver Program Page Size in bytes. This parameter is device specific and is obtained from the device specific processor header file.

Remarks

None

DRV_FLASH_ROW_SIZE Macro

Specifies the FLASH Driver Program Row Size in bytes.

File

drv_flash.h

C

```
#define DRV_FLASH_ROW_SIZE (NVM_ROW_SIZE)
```

Description

FLASH Driver Program Row Size.

This definition specifies the FLASH Driver Program Row Size in bytes. This parameter is device specific and is obtained from the device specific processor header file. The Program Row Size is the maximum block size that can be programmed in one program operation.

Remarks

None

Files

Files

Name	Description
drv_flash.h	Flash Driver interface declarations for the static single instance driver.

Description

drv_flash.h

Flash Driver interface declarations for the static single instance driver.

Functions

	Name	Description
	DRV_FLASH_ErasePage	Erases a page of Flash. **Implementation:** Static
	DRV_FLASH_GetPageSize	Returns the size in bytes of a single "Page" which can be erased in the flash. **Implementation:** Static
	DRV_FLASH_GetRowSize	Returns the size in bytes of a single "Row" which can be written to the flash. **Implementation:** Static
	DRV_FLASH_Initialize	Initializes the Flash instance for the specified driver index. **Implementation:** Static
	DRV_FLASH_IsBusy	Returns true if the Flash device is still busy writing or is erasing. **Implementation:** Static
	DRV_FLASH_Open	Initializes a channel to the appropriate flash device.
	DRV_FLASH_WriteQuadWord	Writes four 4-byte words to the Flash at the (word-aligned) flashAddr. **Implementation:** Static
	DRV_FLASH_WriteRow	Writes an DRV_FLASH_ROW_SIZE bytes to the Flash at the (word-aligned) flashAddr. **Implementation:** Static
	DRV_FLASH_WriteWord	Writes a 4-byte Word to the Flash at the (word-aligned) flashAddr. **Implementation:** Static

Macros

	Name	Description
	DRV_FLASH_INDEX_0	FLASH driver index definitions
	DRV_FLASH_PAGE_SIZE	Specifies the FLASH Driver Program Page Size in bytes.
	DRV_FLASH_ROW_SIZE	Specifies the FLASH Driver Program Row Size in bytes.

Description

Flash Driver Interface Declarations for Static Single Instance Driver

The Flash device driver provides a simple interface to manage the Flash Controller on Microchip microcontrollers. This file defines the interface Declarations for the Flash driver.

Remarks

Static interfaces incorporate the driver instance number within the names of the routines, eliminating the need for an object ID or object handle.

Static single-open interfaces also eliminate the need for the open handle.

File Name

drv_flash.h

Company

Microchip Technology Inc.

Graphics Driver Library

This section describes the Graphics (GFX) Driver Library.

Introduction

The Graphics (GFX) Driver Layer Library is the GFX library stack available for the Microchip family of microcontrollers.

Description

The Microchip Graphics (GFX) Library is a free, modular library optimized for Microchip Microcontrollers.

- Graphics Object Layer - This layer renders the control objects such as button, list box, progress bar, meter, and so on
- Graphics Primitives Layer - This layer implements the primitive rendering functions
- Graphics Display Driver Layer - This layer is the graphics display driver component that is optimized to the actual display module used

The library comes with features such as alpha blending, gradient fills, and anti-aliased fonts. These features can be enabled or disabled through build configurations. Applications can take advantage of these features to enhance the user experience while delivering performance required by the application.

This help provides information on the Graphics Display Driver Layer. This layer is the interface between the Graphics Core Library and the hardware, which may include a display controller or directly to the display device.

Using the Library

This topic describes the basic architecture of the Graphics Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_gfx_display.h`

The interface to the Graphics Driver Library is defined in the `drv_gfx_display.h` header file.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Overview

This topic provides and overview of the Graphics Driver Library.

Description

The driver library API is described in three groups.

- Initialization - provides system level initialization support
- Operations - provides application level run-time support
- Rendering - provides core library level display drawing support

System and Application functions are accessed by system and application routines without using the graphics core library. Rendering function

(Library Functions) are intended be used by the Graphics Primitive Layer only to render pixel data to the actual display.

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The Library Interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Graphics Driver Library.

Section	Description
Graphics Driver Library Functions	Provides general interface functions specific to the Graphics Driver Library.
GLCD Driver Functions	Provides interface functions specific to the GLCD Graphics Driver Library.
LCC Graphics Driver Functions	Provides interface functions specific to the Low-Cost Controllerless (LCC) Graphics Driver Library.
OTM2201A Graphics Driver Functions	Provides interface functions specific to the OTM2201A Graphics Driver Library.
S1D13517 Graphics Driver Functions	Provides interface functions specific to the S1D13517 Graphics Driver Library.
SSD1289 Graphics Driver Functions	Provides interface functions specific to the SSD1289 Graphics Driver Library.
SSD1926 Graphics Driver Functions	Provides interface functions specific to the SSD1926 Graphics Driver Library.
tft002 Graphics Driver Functions	Provides interface functions specific to the tft002 Graphics Driver Library.

How the Library Works

The library provides interfaces to support:

- Initialization
- Operations
- Rendering

Initialization

This topic provides information for initializing the Graphics Driver Library.

Description

The Graphics Library Driver Layer provides application routines to open and configure an instance of the driver.

The application prototypes are generic to all drivers. The user is required to supply the appropriate configuration in the arguments.

```
DRV_HANDLE DRV_GFX_<driver>_Initialize( const SYS_MODULE_INDEX  drvIndex,
                                        const * SYS_MODULE_INIT drvInit)
```

Operations

This topic provides operations information for the Graphics Driver Library.

Description

The Graphics Library Driver Layer provides application routines to open and configure an instance of the driver.

The application prototypes are generic to all drivers. The user is required to supply the appropriate configuration in the arguments.

```
DRV_HANDLE DRV_GFX_<driver>_Open( const SYS_MODULE_INDEX drvIndex,
                                  const DRV_IO_INTENT ioIntent )
```

Rendering

This topic provides information on rendering functions.

Description

The Graphics library driver layer provides basic rendering routines to draw pixels on a specific display.

More information on these interfaces can be found in the Library Interface section.

Creating a New Graphics Driver

This topic provides information for creating a new graphics driver.

Description

The display driver layer provides a generic programming interface to the Graphics Library code allowing it to draw actual pixels on the display.

This interface is adopted by all supported drivers in the graphics library. Each driver is not required to implement all functions. The interface defines the necessary functions required to support the core library.

The display driver API includes the following basic functions defined in the `drv_gfx_display.h` file:

Return type	Function Name	Description
uint16_t	PixelsPut	Sends pixels to the display
uint16_t	BarFill	Renders a bar to the display
uint16_t*	PixelArrayPut	Sends an array of pixels to the display
uint16_t*	PixelArraryGet	Returns an array of pixels from the display
uint16_t	PixelPut	Sends a pixel to the display.
void	ColorSet	Sets the pixel color.
void	InstanceSet	Sets the instance.
uint16_t	PageSet	Sets a page according to the enum PAGE_TYPE.
uint16_t*	Layer	
uint16_t	PixelGet	Returns a pixel.
uint16_t*	AlphaBlendWindow	Alpha window
GFX_STATUS	Status	Returns the status of the driver.

When creating a new driver, the developer will need to provide to the Graphics Library the set of function by means of the function pointer table structure defined in `gfx_driver_display.h`.

The main application code examples poll the DRV_GFX_XXX_InterfaceSet functions to recall which functions are available from a given driver. For faster throughput, DRV_GFX macros can be created (refer to `gfx_driver_display.h` for examples).

Not all rendering functions are needed to for a graphics controller driver to be usable with the library. The inclusion of the functions are dependant on application requirements. For example, if the controller does not have alpha blending, the primitive layer can perform this task. When the initial driver structure is created these functions need to be set as NULL. Any of the "put" functions need to be defined by the controller driver.

Configuring the Library

LCC Driver Configuration Functions

	Name	Description
	DRV_GFX_CONFIG_LCC_EXTERNAL_MEMORY	Macro that enables external memory framebuffer.
	DRV_GFX_CONFIG_LCC_INTERNAL_MEMORY	Macro that enables internal memory framebuffer.
	DRV_GFX_CONFIG_LCC_PALETTE	Macro that disables internal palette memory framebuffer.
	DRV_GFX_LCC_DMA_CHANNEL_INDEX	Macro that defines the DMA CHANNEL INDEX.

OTM2201A Driver Configuration Functions

	Name	Description
	GFX_CONFIG_OTM2201A_DRIVER_COUNT	Macro sets the number of instances for the driver.

S1D13517 Driver Configuration Functions

	Name	Description
	GFX_CONFIG_S1D13517_DRIVER_COUNT	Macro sets the number of instances for the driver.

SSD1926 Driver Configuration Functions

	Name	Description
	GFX_CONFIG_SSD1926_DRIVER_COUNT	Macro sets the number of instances for the driver.

Description

The configuration of the GFX driver is based on the file `drv_gfx_config.h`.

This header file contains the configuration selection for the Graphics Driver. Based on the selections made, the Graphics Driver may support the selected features. These configuration settings will apply to all instances of the Graphics Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

The configuration of the Graphics Driver layer is mainly dependent on making sure the needed driver has the core driver functions inside of it. The main rendering one being a `PixelPut` type function for the Graphics primitive layer.

Once the Graphics Driver structure (GFX_DRV_DATA) is defined along with its primitive function rendering structure (GFX_DRV_FUNCTIONS) it

can be used by the Graphics Primitive Layer.

GLCD Driver Configuration Functions

The GLCD Driver is a static driver and is established from MHC selections and a generation process. MHC provides the configuration interface for this driver. Refer to the Library Interface section for information on these functions.

LCC Driver Configuration Functions

DRV_GFX_CONFIG_LCC_EXTERNAL_MEMORY Macro

Macro that enables external memory framebuffer.

File

drv_gfx_lcc_config_template.h

C

```
#define DRV_GFX_CONFIG_LCC_EXTERNAL_MEMORY
```

Description

Macro: DRV_GFX_CONFIG_LCC_EXTERNAL_MEMORY

This macro enables the use of off-chip memory for the graphic framebuffer.

To enable external framebuffer, add this macro in the configuration and delete DRV_GFX_CONFIG_LCC_INTERNAL_MEMORY if defined.

Remarks

Mandatory definition if DRV_GFX_CONFIG_LCC_INTERNAL_MEMORY is not set.

DRV_GFX_CONFIG_LCC_INTERNAL_MEMORY Macro

Macro that enables internal memory framebuffer.

File

drv_gfx_lcc_config_template.h

C

```
#define DRV_GFX_CONFIG_LCC_INTERNAL_MEMORY
```

Description

Macro: DRV_GFX_CONFIG_LCC_EXTERNAL_MEMORY

This macro enables the use of on-chip memory for the graphic framebuffer.

To enable internal framebuffer, add this macro in the configuration and delete DRV_GFX_CONFIG_LCC_EXTERNAL_MEMORY if defined.

Remarks

Mandatory definition if DRV_GFX_CONFIG_LCC_EXTERNAL_MEMORY is not set.

DRV_GFX_CONFIG_LCC_PALETTE Macro

Macro that disables internal palette memory framebuffer.

File

drv_gfx_lcc_config_template.h

C

```
#define DRV_GFX_CONFIG_LCC_PALETTE
```

Description

/////////////////// COMPILE OPTIONS ////////////////////

****

Macro: DRV_GFX_LCC_PALETTE_DISABLE

This macro disables the use color lookup palette.

To disable palette internal framebuffer, add this macro in the configuration.

Remarks

Optional definition.

DRV_GFX_LCC_DMA_CHANNEL_INDEX Macro

Macro that defines the DMA CHANNEL INDEX.

File

drv_gfx_lcc_config_template.h

C

```
#define DRV_GFX_LCC_DMA_CHANNEL_INDEX
```

Description

Macro: DRV_GFX_LCC_DMA_CHANNEL_INDEX

This macro defines the DMA CHANNEL to be used for LCC display controller operations.

Remarks

Mandatory definition.

S1D13517 Driver Configuration Functions

GFX_CONFIG_S1D13517_DRIVER_COUNT Macro

Macro sets the number of instances for the driver.

File

drv_gfx_s1d13517_config_template.h

C

```
#define GFX_CONFIG_S1D13517_DRIVER_COUNT
```

Description

////////////////// COMPILE OPTIONS ////////////////////

****

Macro: GFX_CONFIG_S1D13517_DRIVER_COUNT

This macro sets the number of instances the driver will support.

Remarks

None.

SSD1926 Driver Configuration Functions

GFX_CONFIG_SSD1926_DRIVER_COUNT Macro

Macro sets the number of instances for the driver.

File

drv_gfx_ssd1926_config_template.h

C

```
#define GFX_CONFIG_SSD1926_DRIVER_COUNT
```

Description

////////////////// COMPILE OPTIONS ////////////////////

****

Macro: GFX_CONFIG_SSD1926_DRIVER_COUNT

This macro sets the number of instances the driver will support.

Remarks

None.

OTM2201A Driver Configuration Functions

GFX_CONFIG_OTM2201A_DRIVER_COUNT Macro

Macro sets the number of instances for the driver.

File

drv_gfx_otm2201a_config_template.h

C

```
#define GFX_CONFIG_OTM2201A_DRIVER_COUNT
```

Description

////////////////// COMPILE OPTIONS ////////////////////

****

Macro: GFX_CONFIG_OTM2201A_DRIVER_COUNT

This macro sets the number of instances the driver will support.

Remarks

None.

tft002 Driver Configuration Functions

Configuring the Display

Provides display configuration information.

Description

With the exception of the GLCD Graphics Driver, which is configured through MHC, each Graphics Controller Driver can support different LCD displays. Each display has its own configuration parameters to correctly display the rendered data. Configuration parameters are located in the file `<install-dir>/framework/driver/gfx/controller/display>/drv_gfx_<display>.h`, where `<display>` indicates the display type. The `drv_gfx_<display>.h` file needs to be included and the configuration parameters are to be passed to the driver initialization structure DRV_GFX_INIT.

Building the Library

This section lists the files that are available in the Graphics Driver Library.

Description

The Graphics Driver is part of the core functionality of the Graphics Library. Please refer to the Building the Library topic in the Graphics Library section for build information.

Library Interface

GLCD Driver Data Types and Constants

	Name	Description
	GLCD_LAYER_CONTROL	Structure holding the layer control definitions

GLCD Driver Functions

	Name	Description
	DRV_GFX_GLCD_BarFill	Outputs a rectangle filled with the global color. **Implementation:** Static
	DRV_GFX_GLCD_CursorImageSet	Sets the cursor image to the image specified. **Implementation:** Static
	DRV_GFX_GLCD_CursorPaletteSet	Sets the Cursor Palette to the given cursor palette. **Implementation:** Static
	DRV_GFX_GLCD_GammaPaletteSet	Sets the global gamma/palette to the given cursor palette. **Implementation:** Static
	DRV_GFX_GLCD_Initialize	Initializes the GLCD driver instance for the specified module index.
	DRV_GFX_GLCD_InterfaceSet	Sets the driver run-time functions for the specified driver index. **Implementation:** Static
	DRV_GFX_GLCD_MaxXGet	Returns the maximum X coordinate value for the specified layer index. **Implementation:** Static
	DRV_GFX_GLCD_MaxYGet	Returns the maximum Y coordinate value for the specified layer index. **Implementation:** Static
	DRV_GFX_GLCD_Open	Opens a GLCD Layer instance for the specified driver index. **Implementation:** Static
	DRV_GFX_GLCD_PixelArrayPut	Outputs an array of pixels of length count starting at *color **Implementation:** Static
	DRV_GFX_GLCD_PixelPut	Outputs one pixel into the frame buffer at the specified position. **Implementation:** Static
	DRV_GFX_GLCD_SetColor	Sets the global pixel color used for PixelPut operations for the specified layer index. **Implementation:** Static
	DRV_GFX_GLCD_BackgroundColorSet	Sets the background color to the specified RGBA value. **Implementation:** Static
	DRV_GFX_GLCD_Close	Closes the GLCD instance for the specified driver index. **Implementation:** Static
	DRV_GFX_GLCD_CursorSetPosition	Sets the position and enable state of the cursor. **Implementation:** Static
	DRV_GFX_GLCD_LayerFrameBufferSet	Sets the base address of the layer to the specific memory location. **Implementation:** Static
	DRV_GFX_GLCD_LayerModeSet	Sets the mode state for the specified layer index. **Implementation:** Static

Graphics Driver Library Functions

	Name	Description
	DRV_GFX_PaletteSet	returns address to the framebuffer.

LCC Driver Data Types and Constants

	Name	Description
	PIP_BUFFER	This is macro PIP_BUFFER.
	DMA_ISR_TASK	This is type DMA_ISR_TASK.
	DRV_GFX_LCC_INDEX_COUNT	Number of valid LCC driver indices.
	DRV_GFX_LCC_DISPLAY_WRITE_BUS_TYPE	
	DRV_GFX_LCC_FB_WRITE_BUS_TYPE	
	LCC_VSYNC_Callback_FnPtr	function pointer definition for VSYNC callback
	LCC_VSYNC_STATE	This is type LCC_VSYNC_STATE.

LCC Driver Functions

	Name	Description
	GFX_PRIM_SetPIPWindow	returns address to the framebuffer.
	DRV_GFX_LCC_DisplayRefresh	LCD refresh handler
	DRV_GFX_LCC_Close	closes an instance of the graphics controller
	DRV_GFX_LCC_SetColor	Sets the color for the driver instance

	DRV_GFX_LCC_GetBuffer	DOM-IGNORE-END
	DRV_GFX_LCC_Initialize	resets LCD, initializes PMP
	DRV_GFX_LCC_SetPage	DOM-IGNORE-START
	DRV_GFX_LCC_AlphaBlendWindow	DOM-IGNORE-END
	DRV_GFX_LCC_Open	opens an instance of the graphics controller
	DRV_GFX_LCC_Tasks	Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer
	DRV_GFX_LCC_FrameBufferAddressSet	Sets address of the framebuffer
	DRV_GFX_LCC_InterfaceSet	Returns the API of the graphics controller
	DRV_GFX_LCC_MaxXGet	Returns x extent of the display.
	DRV_GFX_LCC_MaxYGet	Returns y extent of the display.
	DRV_GFX_LCC_BarFill	outputs one pixel into the frame buffer at the x,y coordinate given
	DRV_GFX_LCC_PixelArrayGet	gets an array of pixels of length count starting at *color
	DRV_GFX_LCC_PixelArrayPut	outputs an array of pixels of length count starting at *color
	DRV_GFX_LCC_PixelPut	outputs one pixel into the frame buffer at the x,y coordinate given
	DRV_GFX_LCC_VSYNC_CallbackSet	Sets a callback function pointer for notification of VSYNC state
	DRV_GFX_LCC_VSYNC_GetState	Gets the state of the current VSYNC mode

OTM2201A Driver Data Types and Constants

	Name	Description
	OTM2201A_TASK	Enumeration for command type.
	DRV_GFX_OTM2201A_COMMAND	Structure for the commands in the driver queue.
	DRV_GFX_OTM2201A_INDEX_COUNT	Number of valid OTM2201A driver indices.

OTM2201A Driver Functions

	Name	Description
	DRV_GFX_OTM2201A_AddressSet	Sets the start GRAM address where pixel data to be written
	DRV_GFX_OTM2201A_BrightnessSet	Sets the brightness of the display backlight.
	DRV_GFX_OTM2201A_RegGet	Returns graphics controller register value (byte access)
	DRV_GFX_OTM2201A_Busy	Returns non-zero value if LCD controller is busy (previous drawing operation is not completed).
	DRV_GFX_OTM2201A_Close	closes an instance of the graphics controller
	DRV_GFX_OTM2201A_ColorSet	Sets the color for the driver instance
	DRV_GFX_OTM2201A_Initialize	resets LCD, initializes PMP
	DRV_GFX_OTM2201A_InstanceSet	Sets the instance for the driver
	DRV_GFX_OTM2201A_Open	opens an instance of the graphics controller
	DRV_GFX_OTM2201A_RegSet	Updates graphics controller register value (byte access)
	DRV_GFX_OTM2201A_Tasks	Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer
	DRV_GFX_OTM2201A_InterfaceSet	Returns the API of the graphics controller
	DRV_GFX_OTM2201A_MaxXGet	Returns x extent of the display.
	DRV_GFX_OTM2201A_MaxYGet	Returns y extent of the display.
	DRV_GFX_OTM2201A_BarFill	Outputs count number of pixels into the frame buffer from the given x,y coordinate.
	DRV_GFX_OTM2201A_PixelArrayGet	Gets an array of pixels of length count into an array starting at *color
	DRV_GFX_OTM2201A_PixelArrayPut	Outputs an array of pixels of length count starting at *color
	DRV_GFX_OTM2201A_PixelPut	Outputs one pixel into the frame buffer at the x,y coordinate given

S1D13517 Driver Data Types and Constants

	Name	Description
	LAYER_REGISTERS	This structure is used to describe layerS1D13517_REGisters.
	DRV_GFX_S1D13517_INDEX_COUNT	Number of valid S1D13517 driver indices.

S1D13517 Driver Functions

	Name	Description
	DRV_GFX_S1D13517_AlphaBlendWindow	SEE primitive layer alphablendWindow definition
	DRV_GFX_S1D13517_GetReg	returns graphics controllerS1D13517_REGister value (byte access)

	DRV_GFX_S1D13517_SetInstance	Sets the instance for the driver
	DRV_GFX_S1D13517_SetReg	updates graphics controllerS1D13517_REGister value (byte access)
	DRV_GFX_S1D13517_Close	closes an instance of the graphics controller
	DRV_GFX_S1D13517_Layer	Updates a Layer depending on the layer parameters.
	DRV_GFX_S1D13517_Initialize	resets LCD, initializes PMP
	DRV_GFX_S1D13517_Open	opens an instance of the graphics controller
	DRV_GFX_S1D13517_SetColor	Sets the color for the driver instance
	DRV_GFX_S1D13517_Tasks	Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer
	DRV_GFX_S1D13517_SetPage	Sets the page of a certain page type
	DRV_GFX_S1D13517_InterfaceSet	Returns the API of the graphics controller
	DRV_GFX_S1D13517_MaxXGet	Returns x extent of the display.
	DRV_GFX_S1D13517_MaxYGet	Returns y extent of the display.
	DRV_GFX_S1D13517_BarFill	outputs one pixel into the frame buffer at the x,y coordinate given
	DRV_GFX_S1D13517_BrightnessSet	Sets the brightness of the display backlight.
	DRV_GFX_S1D13517_PixelPut	outputs one pixel into the frame buffer at the x,y coordinate given
	DRV_GFX_S1D13517_PixelArrayPut	outputs an array of pixels of length count starting at *color

SSD1289 Driver Functions

	Name	Description
	GFX_TCON_SSD1289Init	Initialize the Solomon Systech SSD1289 Timing Controller.

SSD1926 Driver Data Types and Constants

	Name	Description
	DRV_GFX_SSD1926_INDEX_COUNT	Number of valid SSD1926 driver indices.
	DRV_GFX_SSD1926_COMMAND	Structure for the commands in the driver queue.

SSD1926 Driver Functions

	Name	Description
	DRV_GFX_SSD1926_Busy	Returns non-zero if LCD controller is busy (previous drawing operation is not completed).
	DRV_GFX_SSD1926_GetReg	returns graphics controller register value (byte access)
	DRV_GFX_SSD1926_SetReg	updates graphics controller register value (byte access)
	DRV_GFX_SSD1926_Close	closes an instance of the graphics controller
	DRV_GFX_SSD1926_Initialize	resets LCD, initializes PMP
	DRV_GFX_SSD1926_Open	opens an instance of the graphics controller
	DRV_GFX_SSD1926_SetColor	Sets the color for the driver instance
	DRV_GFX_SSD1926_Status	Provides the current status of the SSD1926 driver module.
	DRV_GFX_SSD1926_SetInstance	Sets the instance for the driver
	DRV_GFX_SSD1926_Tasks	Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer
	DRV_GFX_SSD1926_InterfaceSet	Returns the API of the graphics controller
	DRV_GFX_SSD1926_MaxXGet	Returns x extent of the display.
	DRV_GFX_SSD1926_MaxYGet	Returns y extent of the display.
	DRV_GFX_SSD1926_BarFill	Hardware accelerated barfill function
	DRV_GFX_SSD1926_PixelArrayGet	gets an array of pixels of length count starting at *color
	DRV_GFX_SSD1926_PixelArrayPut	outputs an array of pixels of length count starting at *color
	DRV_GFX_SSD1926_PixelPut	outputs one pixel into the frame buffer at the x,y coordinate given

Description

Graphics Driver Library Application Programming Interface (API).

Graphics Driver Library Functions

DRV_GFX_PaletteSet Function

returns address to the framebuffer.

File

drv_gfx_lcc.h

C

```
uint8_t DRV_GFX_PaletteSet(uint16_t * pPaletteEntry, uint16_t startEntry, uint16_t length);
```

Description

none.

GLCD Driver Functions

DRV_GFX_GLCD_BarFill Function

Outputs a rectangle filled with the global color.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
void DRV_GFX_GLCD_BarFill(const SYS_MODULE_INDEX index, uint16_t left, uint16_t top, uint16_t right,
uint16_t bottom);
```

Returns

None.

Description

This function outputs a rectangle of pixels into the framebuffer at the specified position, width, and height.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Open has been called.

Parameters

Parameters	Description
SYS_MODULE_INDEX index	layer to use
left, top	pixel coordinates
right, bottom	pixel coordinates

Function

void DRV_GFX_GLCD_BarFill(const SYS_MODULE_INDEX index, uint16_t left, uint16_t top,
uint16_t right, uint16_t bottom)

DRV_GFX_GLCD_CursorImageSet Function

Sets the cursor image to the image specified.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
void DRV_GFX_GLCD_CursorImageSet(const SYS_MODULE_INDEX index, uint32_t * cursorImage);
```

Returns

None.

Description

This functions sets the cursor image to the image specified.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Open has been called.

Parameters

Parameters	Description
uint32_t * cursorImage	32x32 palette image

Function

void DRV_GFX_GLCD_CursorImageSet(const SYS_MODULE_INDEX index, uint32_t * cursorImage)

DRV_GFX_GLCD_CursorPaletteSet Function

Sets the Cursor Palette to the given cursor palette.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
void DRV_GFX_GLCD_CursorPaletteSet(const SYS_MODULE_INDEX index, uint32_t * gammaPalette);
```

Returns

None.

Description

This functions sets the cursor image to the image specified.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Open has been called.

Parameters

Parameters	Description
uint32_t * cursorImage	32x32 palette image

Function

void DRV_GFX_GLCD_CursorPaletteSet(uint32_t * cursorPalette)

DRV_GFX_GLCD_GammaPaletteSet Function

Sets the global gamma/palette to the given cursor palette.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
void DRV_GFX_GLCD_GammaPaletteSet(const SYS_MODULE_INDEX index, uint32_t * gammaPalette);
```

Returns

None.

Description

This function sets the global gamma/platform image.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Open has been called.

Parameters

Parameters	Description
uint32_t * cursorImage	32x32 palette image

Function

void DRV_GFX_GLCD_GammaPaletteSet(const SYS_MODULE_INDEX index, uint32_t * cursorPalette)

DRV_GFX_GLCD_Initialize Function

Initializes the GLCD driver instance for the specified module index.

File

help_drv_gfx_glcd.h

C

```
SYS_MODULE_OBJ DRV_GFX_GLCD_Initialize(const SYS_MODULE_INDEX moduleIndex, const SYS_MODULE_INIT * const moduleInit);
```

Returns

None.

Description

This function initializes the GLCD Driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

This function must be called before any other GLCD function is called. This function should only be called once during system initialization.

Preconditions

None.

Function

SYS_MODULE_OBJ DRV_GFX_GLCD_Initialize(const SYS_MODULE_INDEX moduleIndex,
const SYS_MODULE_INIT * const moduleInit)

DRV_GFX_GLCD_InterfaceSet Function

Sets the driver run-time functions for the specified driver index.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
void DRV_GFX_GLCD_InterfaceSet(const SYS_MODULE_INDEX index, DRV_HANDLE handle, DRV_GFX_INTERFACE * interface);
```

Returns

None.

Description

This function sets implementation function function applicable to the Graphics Driver interface.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Initialize has been called.

Parameters

Parameters	Description
SYS_MODULE_INDEX index	layer to use
DRV_HANDLE handle	driver handle
DRV_GFX_INTERFACE * interface	pointer to interface structure

Function

void DRV_GFX_GLCD_InterfaceSet(const SYS_MODULE_INDEX index,
DRV_HANDLE handle,
DRV_GFX_INTERFACE * interface)

DRV_GFX_GLCD_MaxXGet Function

Returns the maximum X coordinate value for the specified layer index.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
uint16_t DRV_GFX_GLCD_MaxXGet(const SYS_MODULE_INDEX index);
```

Returns

- x position - When a message has been received

Description

This function returns the maximum X coordinate value for the specified layer index.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Open has been called.

Parameters

Parameters	Description
SYS_MODULE_INDEX index	layer to use

Function

uint16_t DRV_GFX_GLCD_MaxXGet(const SYS_MODULE_INDEX index)

DRV_GFX_GLCD_MaxYGet Function

Returns the maximum Y coordinate value for the specified layer index.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
uint16_t DRV_GFX_GLCD_MaxYGet(const SYS_MODULE_INDEX index);
```

Returns

- y position - When a message has been received

Description

This function returns the maximum Y coordinate value for the specified layer index.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Open has been called.

Parameters

Parameters	Description
SYS_MODULE_INDEX index	layer to use

Function

uint16_t DRV_GFX_GLCD_MaxYGet(const SYS_MODULE_INDEX index)

DRV_GFX_GLCD_Open Function

Opens a GLCD Layer instance for the specified driver index.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
DRV_HANDLE DRV_GFX_GLCD_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Returns

None.

Description

This function opens the GLCD Driver instance for the specified driver instance, making it ready for clients to use it.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Initialize has been called.

Function

void DRV_GFX_GLCD_Open(const SYS_MODULE_INDEX index,
const DRV_IO_INTENT intent)

DRV_GFX_GLCD_PixelArrayPut Function

Outputs an array of pixels of length count starting at *color

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
void DRV_GFX_GLCD_PixelArrayPut(const SYS_MODULE_INDEX index, GFX_COLOR * color, uint16_t x, uint16_t y,
```

```
uint16_t count, uint16_t lineCount);
```

Returns

None.

Description

This function outputs an array of pixels of length count starting at *color.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Open has been called.

Parameters

Parameters	Description
SYS_MODULE_INDEX index	layer to use
*color	start of the array
x	x coordinate of the start point.
y	y coordinate of the end point.
count	number of pixels
lineCount	number of lines

Function

void DRV_GFX_GLCD_PixelArrayPut(const SYS_MODULE_INDEX index, GFX_COLOR *color,
uint16_t x, uint16_t y, uint16_t count, uint16_t lineCount)

DRV_GFX_GLCD_PixelPut Function

Outputs one pixel into the frame buffer at the specified position.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
void DRV_GFX_GLCD_PixelPut(const SYS_MODULE_INDEX index, uint16_t x, uint16_t y);
```

Returns

None.

Description

This function outputs one pixel into the frame buffer at the specified x,y coordinate.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Open has been called.

Parameters

Parameters	Description
SYS_MODULE_INDEX index	layer to use
uint16_t x,y	pixel coordinates

Function

void DRV_GFX_GLCD_PixelPut(const SYS_MODULE_INDEX index, uint16_t x, uint16_t y)

DRV_GFX_GLCD_SetColor Function

Sets the global pixel color used for PixelPut operations for the specified layer index.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
void DRV_GFX_GLCD_SetColor(const SYS_MODULE_INDEX index, GFX_COLOR color);
```

Returns

None.

Description

This function sets the global pixel color used for PixelPut operations using the specified color.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Open has been called.

Parameters

Parameters	Description
SYS_MODULE_INDEX index	layer to use
GFX_COLOR color	color to use

Function

void DRV_GFX_GLCD_SetColor(const SYS_MODULE_INDEX index, GFX_COLOR color)

DRV_GFX_GLCD_BackgroundColorSet Function

Sets the background color to the specified RGBA value.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
void DRV_GFX_GLCD_BackgroundColorSet(const SYS_MODULE_INDEX index, uint32_t bgColor);
```

Returns

- none

Description

This function sets the background color to the specified RGBA value.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Open has been called.

Parameters

Parameters	Description
bgColor	RGBA color

Function

void void DRV_GFX_GLCD_BackgroundColorSet(const SYS_MODULE_INDEX index,

uint32_t bgColor)

DRV_GFX_GLCD_Close Function

Closes the GLCD instance for the specified driver index.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
void DRV_GFX_GLCD_Close(const SYS_MODULE_INDEX index);
```

Returns

None.

Description

This function closes the GLCD driver instance for the specified driver instance, making it ready for clients to use it.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Initialize has been called.

Function

void DRV_GFX_GLCD_Close(const SYS_MODULE_INDEX index)

DRV_GFX_GLCD_CursorSetPosition Function

Sets the position and enable state of the cursor.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
void DRV_GFX_GLCD_CursorSetPosition(const SYS_MODULE_INDEX index, uint32_t x, uint32_t y, bool enable);
```

Returns

None.

Description

This function sets the x, y position of the cursor and its enable state.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Open has been called.

Parameters

Parameters	Description
x	x coordinate of the start point.
y	y coordinate of the end point.
enable	enable cursor on or off

Function

void DRV_GFX_GLCD_CursorSetPosition(const SYS_MODULE_INDEX index, uint32_t x, uint32_t y, bool enable)

DRV_GFX_GLCD_LayerFrameBufferSet Function

Sets the base address of the layer to the specific memory location.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
void DRV_GFX_GLCD_LayerFrameBufferSet(const SYS_MODULE_INDEX index, uint32_t * frame);
```

Returns

None.

Description

This function sets the base address of the layer to the specific memory location.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Open has been called.

Function

void DRV_GFX_GLCD_LayerFrameBufferSet(const SYS_MODULE_INDEX index, uint32_t * frame)

DRV_GFX_GLCD_LayerModeSet Function

Sets the mode state for the specified layer index.

Implementation: Static

File

help_drv_gfx_glcd.h

C

```
void DRV_GFX_GLCD_LayerModeSet(const SYS_MODULE_INDEX index, uint32_t layerMode);
```

Returns

None.

Description

This function sets the mode state of the layer to either show or not show the layer.

Remarks

None.

Preconditions

DRV_GFX_GLCD_Open has been called.

Function

void DRV_GFX_GLCD_LayerModeSet(const SYS_MODULE_INDEX index, uint32_t layerMode)

GLCD Driver Data Types and Constants

GLCD_LAYER_CONTROL Enumeration

Structure holding the layer control definitions

File

help_drv_gfx_glcd.h

C

```
typedef enum {
  GLCD_LAYER_DISABLE = 0,
  GLCD_LAYER_AHBLOCK = 1<<27,
  GLCD_LAYER_ENABLE = 1<<31,
  GLCD_FORCE_A = 1<<30,
  GLCD_SCALE_NN = 1<<29,
  GLCD_MODULATE_A = 1<<28
} GLCD_LAYER_CONTROL;
```

Description

GLCD Layer Control Definitions

This structure contains the layer control definitions.

LCC Driver Functions

GFX_PRIM_SetPIPWindow Function

returns address to the framebuffer.

File

drv_gfx_lcc.h

C

```
void GFX_PRIM_SetPIPWindow(uint16_t left, uint16_t top, uint16_t hlength, uint16_t vlength, uint16_t pipx,
uint16_t pipy);
```

Description

none.

DRV_GFX_LCC_DisplayRefresh Function

LCD refresh handler

File

drv_gfx_lcc.h

C

```
void DRV_GFX_LCC_DisplayRefresh();
```

Returns

none

Description

This routine is called from the timer interrupt, resulting in a complete LCD update.

Function

void DRV_GFX_LCC_DisplayRefresh(void)

DRV_GFX_LCC_Close Function

closes an instance of the graphics controller

File

drv_gfx_lcc.h

C

```
void DRV_GFX_LCC_Close(DRV_HANDLE handle);
```

Description

none

Function

void DRV_GFX_LCC_Close(DRV_HANDLE handle)

DRV_GFX_LCC_SetColor Function

Sets the color for the driver instance

File

drv_gfx_lcc.h

C

```
void DRV_GFX_LCC_SetColor(GFX_COLOR color);
```

Returns

none

Function

void DRV_GFX_LCC_SetColor(uint8_t instance, GFX_COLOR color)

DRV_GFX_LCC_GetBuffer Function

File

drv_gfx_lcc.h

C

```
unsigned short * DRV_GFX_LCC_GetBuffer();
```

Description

DOM-IGNORE-END

DRV_GFX_LCC_Initialize Function

resets LCD, initializes PMP

File

drv_gfx_lcc.h

C

```
SYS_MODULE_OBJ DRV_GFX_LCC_Initialize(const SYS_MODULE_INDEX moduleIndex, const SYS_MODULE_INIT * const
moduleInit);
```

Returns

1 - call not successful (PMP driver busy) 0 - call successful

Description

none

Parameters

Parameters	Description
instance	driver instance

Function

SYS_MODULE_OBJ DRV_GFX_LCC_Initialize(const SYS_MODULE_INDEX moduleIndex,
const SYS_MODULE_INIT * const moduleInit)

DRV_GFX_LCC_SetPage Function

File

drv_gfx_lcc.h

C

```
uint16_t DRV_GFX_LCC_SetPage(uint8_t pageType, uint8_t page);
```

Description

DOM-IGNORE-START

DRV_GFX_LCC_AlphaBlendWindow Function

File

drv_gfx_lcc.h

C

```
uint16_t* DRV_GFX_LCC_AlphaBlendWindow(GFX_ALPHA_PARAMS* alphaParams, uint16_t width, uint16_t height,
uint8_t alpha);
```

Description

DOM-IGNORE-END

DRV_GFX_LCC_Open Function

opens an instance of the graphics controller

File

drv_gfx_lcc.h

C

```
DRV_HANDLE DRV_GFX_LCC_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Description

none

Function

DRV_GFX_LCC_Open(uint8_t instance)

DRV_GFX_LCC_Tasks Function

Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer

File

drv_gfx_lcc.h

C

```
void DRV_GFX_LCC_Tasks(SYS_MODULE_OBJ object);
```

Function

void DRV_GFX_LCC_Tasks(void)

DRV_GFX_LCC_FrameBufferAddressSet Function

Sets address of the framebuffer

File

drv_gfx_lcc.h

C

```
uint16_t DRV_GFX_LCC_FrameBufferAddressSet(void * address);
```

Returns

Sets address of the framebuffer

Description

none

Function

DRV_GFX_LCC_FrameBufferAddressSet(void * address)

DRV_GFX_LCC_InterfaceSet Function

Returns the API of the graphics controller

File

drv_gfx_lcc.h

C

```
void DRV_GFX_LCC_InterfaceSet(DRV_HANDLE handle, DRV_GFX_INTERFACE * interface);
```

Description

none

Function

DRV_GFX_INTEFACE DRV_GFX_LCC_InterfaceGet(DRV_HANDLE handle)

DRV_GFX_LCC_MaxXGet Function

Returns x extent of the display.

File

drv_gfx_lcc.h

C

```
uint16_t DRV_GFX_LCC_MaxXGet();
```

Example

```
Remarks:
```

Function

void DRV_GFX_LCC_MaxXGet()

DRV_GFX_LCC_MaxYGet Function

Returns y extent of the display.

File

drv_gfx_lcc.h

C

```
uint16_t DRV_GFX_LCC_MaxYGet();
```

Example

```
Remarks:
```

Function

void DRV_GFX_LCC_MaxYGet()

DRV_GFX_LCC_BarFill Function

outputs one pixel into the frame buffer at the x,y coordinate given

File

drv_gfx_lcc.h

C

```
void DRV_GFX_LCC_BarFill(uint16_t left, uint16_t top, uint16_t right, uint16_t bottom);
```

Returns

1 - call not successful (LCC driver busy) 0 - call successful

Description

none

Parameters

Parameters	Description
left,top	pixel coordinates
right, bottom	pixel coordinates

Function

void DRV_GFX_LCC_BarFill(uint16_t left, uint16_t top, uint16_t right, uint16_t bottom)

DRV_GFX_LCC_PixelArrayGet Function

gets an array of pixels of length count starting at *color

File

drv_gfx_lcc.h

C

```
uint16_t* DRV_GFX_LCC_PixelArrayGet(GFX_COLOR * color, uint16_t x, uint16_t y, uint16_t count);
```

Returns

ignore

Description

none

Parameters

Parameters	Description
instance	driver instance
*color	start of the array
x	x coordinate of the start point.
y	y coordinate of the end point.
count	number of pixels

Function

uint16_t* DRV_GFX_LCC_PixelArrayGet(uint16_t *color, uint16_t x, uint16_t y, uint16_t count)

DRV_GFX_LCC_PixelArrayPut Function

outputs an array of pixels of length count starting at *color

File

drv_gfx_lcc.h

C

```
void DRV_GFX_LCC_PixelArrayPut(GFX_COLOR * color, uint16_t x, uint16_t y, uint16_t count, uint16_t
```

```
lineCount);
```

Returns

NULL - call not successful (PMP driver busy) !NULL - address of the display driver queue command

Description

none

Parameters

Parameters	Description
*color	start of the array
x	x coordinate of the start point.
y	y coordinate of the end point.
count	number of pixels
lineCount	number of lines

Function

void DRV_GFX_LCC_PixelArrayPut(uint16_t *color, uint16_t x, uint16_t y, uint16_t count)

DRV_GFX_LCC_PixelPut Function

outputs one pixel into the frame buffer at the x,y coordinate given

File

drv_gfx_lcc.h

C

```
void DRV_GFX_LCC_PixelPut(uint16_t x, uint16_t y);
```

Returns

1 - call not successful (LCC driver busy) 0 - call successful

Description

none

Parameters

Parameters	Description
x,y	pixel coordinates

Function

void DRV_GFX_LCC_PixelPut(uint16_t x, uint16_t y)

DRV_GFX_LCC_VSYNC_CallbackSet Function

Sets a callback function pointer for notification of VSYNC state

File

drv_gfx_lcc.h

C

```
void DRV_GFX_LCC_VSYNC_CallbackSet(LCC_VSYNC_Callback_FnPtr cb);
```

Function

void DRV_GFX_LCC_VSYNC_CallbackSet(LCC_VSYNC_Callback_FnPtr)

DRV_GFX_LCC_VSYNC_GetState Function

Gets the state of the current VSYNC mode

File

drv_gfx_lcc.h

C

```
LCC_VSYNC_STATE DRV_GFX_LCC_VSYNC_GetState();
```

Function

int32_t DRV_GFX_LCC_VSYNC_GetState(void)

LCC Driver Data Types and Constants

PIP_BUFFER Macro

File

drv_gfx_lcc.h

C

```
#define PIP_BUFFER (3)
```

Description

This is macro PIP_BUFFER.

DMA_ISR_TASK Enumeration

File

drv_gfx_lcc.h

C

```
typedef enum {
  ACTIVE_PERIOD = 0,
  BLANKING_PERIOD,
  FINISH_LINE,
  OVERFLOW,
  PIP,
  SCROLL
} DMA_ISR_TASK;
```

Description

This is type DMA_ISR_TASK.

DRV_GFX_LCC_INDEX_COUNT Macro

Number of valid LCC driver indices.

File

drv_gfx_lcc.h

C

```
#define DRV_GFX_LCC_INDEX_COUNT DRV_GFX_LCC_NUMBER_OF_MODULES
```

Description

LCC Driver Module Index Count

This constant identifies LCC driver index definitions.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is device-specific.

DRV_GFX_LCC_DISPLAY_WRITE_BUS_TYPE Enumeration

File

drv_gfx_lcc.h

C

```
typedef enum {
  DRV_GFX_LCC_DISPLAY_WRITE_BUS_TYPE_NONE = 0,
  DRV_GFX_LCC_DISPLAY_WRITE_BUS_TYPE_PMP,
  DRV_GFX_LCC_DISPLAY_WRITE_BUS_TYPE_EBI
} DRV_GFX_LCC_DISPLAY_WRITE_BUS_TYPE;
```

Description

LCC Driver Module Index Count

DRV_GFX_LCC_FB_WRITE_BUS_TYPE Enumeration

File

drv_gfx_lcc.h

C

```
typedef enum {
  DRV_GFX_LCC_FB_WRITE_BUS_TYPE_NONE = 0,
  DRV_GFX_LCC_FB_WRITE_BUS_TYPE_PMP,
  DRV_GFX_LCC_FB_WRITE_BUS_TYPE_EBI
} DRV_GFX_LCC_FB_WRITE_BUS_TYPE;
```

Description

LCC Driver Module Index Count

LCC_VSYNC_Callback_FnPtr Type

File

drv_gfx_lcc.h

C

```
typedef void (* LCC_VSYNC_Callback_FnPtr)(void);
```

Description

function pointer definition for VSYNC callback

LCC_VSYNC_STATE Enumeration

File

drv_gfx_lcc.h

C

```
typedef enum {
  LCC_VSYNC_TRUE = 0,
  LCC_VSYNC_FALSE
} LCC_VSYNC_STATE;
```

Description

This is type LCC_VSYNC_STATE.

S1D13517 Driver Functions

DRV_GFX_S1D13517_AlphaBlendWindow Function

SEE primitive layer alphablendWindow definition

File

drv_gfx_s1d13517.h

C

```
uint16_t* DRV_GFX_S1D13517_AlphaBlendWindow(GFX_ALPHA_PARAMS* alphaParams, uint16_t width, uint16_t height,
uint8_t alpha);
```

Function

void DRV_GFX_S1D13517_AlphaBlendWindow(uint8_t pageType, uint8_t page)

DRV_GFX_S1D13517_GetReg Function

returns graphics controllerS1D13517_REGister value (byte access)

File

drv_gfx_s1d13517.h

C

```
uint8_t DRV_GFX_S1D13517_GetReg(uint8_t index);
```

Returns

0 - when call was passed

Description

none

Function

uint8_t DRV_GFX_S1D13517_GetReg(uint8_t index)

DRV_GFX_S1D13517_SetInstance Function

Sets the instance for the driver

File

drv_gfx_s1d13517.h

C

```
void DRV_GFX_S1D13517_SetInstance(uint8_t instance);
```

Returns

none

Function

void DRV_GFX_S1D13517_SetInstance(uint8_t instance)

DRV_GFX_S1D13517_SetReg Function

updates graphics controllerS1D13517_REGister value (byte access)

File

drv_gfx_s1d13517.h

C

```
uint16_t DRV_GFX_S1D13517_SetReg(uint8_t index, uint8_t value);
```

Returns

1 - call was not passed 0 - call was passed

Description

none

Parameters

Parameters	Description
value	value to write toS1D13517_REGister

Function

uint8_t DRV_GFX_S1D13517_SetReg(uint8_t index, uint8_t value)

DRV_GFX_S1D13517_Close Function

closes an instance of the graphics controller

File

drv_gfx_s1d13517.h

C

```
void DRV_GFX_S1D13517_Close(DRV_HANDLE handle);
```

Returns

0 - instance closed 2 - instance doesn't exist 3 - instance already closed

Description

none

Function

DRV_GFX_S1D13517_Close(uint8_t instance)

DRV_GFX_S1D13517_Layer Function

Updates a Layer depending on the layer parameters.

File

drv_gfx_s1d13517.h

C

```
uint16_t* DRV_GFX_S1D13517_Layer(uint8_t type, GFX_LAYER_PARAMS* layer);
```

Returns

NULL - call not successful (PMP driver busy) !NULL - address of the display driver queue command

Description

none

Parameters

Parameters	Description
type	layer type
layer	parameters for Layer function call

Function

uint16_t* DRV_GFX_S1D13517_Layer(uint8_t type, GFX_LAYER_PARAMS* layer)

DRV_GFX_S1D13517_Initialize Function

resets LCD, initializes PMP

File

drv_gfx_s1d13517.h

C

```
SYS_MODULE_OBJ DRV_GFX_S1D13517_Initialize(const SYS_MODULE_INDEX moduleIndex, const SYS_MODULE_INIT *
const moduleInit);
```

Returns

NULL - call not successful (PMP driver busy) !NULL - address of the display driver queue command

Description

none

Parameters

Parameters	Description
instance	driver instance

Function

uint16_t DRV_GFX_S1D13517_Initialize(uint8_t instance)

DRV_GFX_S1D13517_Open Function

opens an instance of the graphics controller

File

drv_gfx_s1d13517.h

C

```
DRV_HANDLE DRV_GFX_S1D13517_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Returns

1 - driver not initialized 2 - instance doesn't exist 3 - instance already open instance to driver when successful

Description

none

Function

DRV_GFX_S1D13517_Open(uint8_t instance)

DRV_GFX_S1D13517_SetColor Function

Sets the color for the driver instance

File

drv_gfx_s1d13517.h

C

```
void DRV_GFX_S1D13517_SetColor(GFX_COLOR color);
```

Returns

none

Function

void DRV_GFX_S1D13517_SetColor(GFX_COLOR color)

DRV_GFX_S1D13517_Tasks Function

Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer

File

drv_gfx_s1d13517.h

C

```
void DRV_GFX_S1D13517_Tasks(SYS_MODULE_OBJ object);
```

Function

void DRV_GFX_SSD1926_Tasks(void)

DRV_GFX_S1D13517_SetPage Function

Sets the page of a certain page type

File

drv_gfx_s1d13517.h

C

```
uint16_t DRV_GFX_S1D13517_SetPage(uint8_t pageType, uint8_t page);
```

Returns

NULL - call not successful (PMP driver busy) !NULL - address of the display driver queue command

Description

none

Parameters

Parameters	Description
instance	driver instance

Function

void DRV_GFX_S1D13517_SetPage(uint8_t pageType, uint8_t page)

DRV_GFX_S1D13517_InterfaceSet Function

Returns the API of the graphics controller

File

drv_gfx_s1d13517.h

C

```
void DRV_GFX_S1D13517_InterfaceSet(DRV_HANDLE handle, DRV_GFX_INTERFACE * interface);
```

Description

none

Function

void DRV_GFX_S1D13517_InterfaceSet(DRV_HANDLE handle, DRV_GFX_INTERFACE * interface)

DRV_GFX_S1D13517_MaxXGet Function

Returns x extent of the display.

File

drv_gfx_s1d13517.h

C

```
uint16_t DRV_GFX_S1D13517_MaxXGet();
```

Example

```
Remarks:
```

Function

void DRV_GFX_S1D13517_MaxXGet()

DRV_GFX_S1D13517_MaxYGet Function

Returns y extent of the display.

File

drv_gfx_s1d13517.h

C

```
uint16_t DRV_GFX_S1D13517_MaxYGet();
```

Example

```
Remarks:
```

Function

void DRV_GFX_S1D13517_MaxYGet()

DRV_GFX_S1D13517_BarFill Function

outputs one pixel into the frame buffer at the x,y coordinate given

File

drv_gfx_s1d13517.h

C

```
void DRV_GFX_S1D13517_BarFill(uint16_t left, uint16_t top, uint16_t right, uint16_t bottom);
```

Returns

NULL - call not successful (PMP driver busy) !NULL - address of the display driver queue command

Description

none

Parameters

Parameters	Description
left, top	pixel coordinates
right, bottom	pixel coordinates

Function

void DRV_GFX_S1D13517_BarFill(uint16_t left, uint16_t top, uint16_t right, uint16_t bottom)

DRV_GFX_S1D13517_BrightnessSet Function

Sets the brightness of the display backlight.

File

drv_gfx_s1d13517.h

C

```
void DRV_GFX_S1D13517_BrightnessSet(uint16_t level);
```

Returns

none

Description

none

Parameters

Parameters	Description
level	Brightness level. Valid values are 0 to 100. • 0: brightness level is zero or display is turned off • 100: brightness level is maximum

Function

uint8_t DRV_GFX_S1D13517_BrightnessSet(uint16_t level)

DRV_GFX_S1D13517_PixelPut Function

outputs one pixel into the frame buffer at the x,y coordinate given

File

drv_gfx_s1d13517.h

C

```
void DRV_GFX_S1D13517_PixelPut(uint16_t x, uint16_t y);
```

Returns

NULL - call not successful (PMP driver busy) !NULL - address of the display driver queue command

Description

none

Parameters

Parameters	Description
instance	driver instance
color	color to output
x,y	pixel coordinates

Function

uint16_t DRV_GFX_S1D13517_PixelPut(uint16_t x, uint16_t y)

DRV_GFX_S1D13517_PixelArrayPut Function

outputs an array of pixels of length count starting at *color

File

drv_gfx_s1d13517.h

C

```
void DRV_GFX_S1D13517_PixelArrayPut(GFX_COLOR * color, uint16_t x, uint16_t y, uint16_t count, uint16_t
lineCount);
```

Returns

NULL - call not successful (PMP driver busy) !NULL - address to the number of pixels yet to be serviced

Description

none

Parameters

Parameters	Description
*color	start of the array
x	x coordinate of the start point.
y	y coordinate of the end point.
count	number of pixels
lineCount	number of lines

Function

void DRV_GFX_S1D13517_PixelArrayPut(uint16_t *color, uint16_t x, uint16_t y, uint16_t count, uint16_t lineCount)

S1D13517 Driver Data Types and Constants

LAYER_REGISTERS Structure

File

drv_gfx_s1d13517.h

C

```
typedef struct {
  uint8_t XStart;
  uint8_t XEnd;
  uint8_t YStart0;
  uint8_t YStart1;
  uint8_t YEnd0;
  uint8_t YEnd1;
  uint8_t StartAddress0;
  uint8_t StartAddress1;
  uint8_t StartAddress2;
} LAYER_REGISTERS;
```

Description

This structure is used to describe layerS1D13517_REGisters.

DRV_GFX_S1D13517_INDEX_COUNT Macro

Number of valid S1D13517 driver indices.

File

drv_gfx_s1d13517.h

C

```
#define DRV_GFX_S1D13517_INDEX_COUNT DRV_GFX_S1D13517_NUMBER_OF_MODULES
```

Description

This constant identifies S1D13517 driver index definitions.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is device-specific.

Section

Data Types and Constants

****

****

****

SSD1926 Driver Module Index Count

SSD1289 Driver Functions

GFX_TCON_SSD1289Init Function

Initialize the Solomon Systech SSD1289 Timing Controller.

File

drv_gfx_ssd1289.h

C

```
void GFX_TCON_SSD1289Init();
```

Returns

None.

Description

Initialize the I/Os to implement a bit-banged SPI interface to connect to the Timing Controller through the SPI.

Function

void GfxTconInit(void)

SSD1926 Driver Functions

DRV_GFX_SSD1926_Busy Function

Returns non-zero if LCD controller is busy (previous drawing operation is not completed).

File

drv_gfx_ssd1926.h

C

```
uint16_t DRV_GFX_SSD1926_Busy(uint8_t instance);
```

Returns

1 - busy 0 - not busy

Description

none

Parameters

Parameters	Description
instance	driver instance

Function

uint16_t DRV_GFX_SSD1926_Busy(uint8_t instance)

DRV_GFX_SSD1926_GetReg Function

returns graphics controller register value (byte access)

File

drv_gfx_ssd1926.h

C

```
uint8_t DRV_GFX_SSD1926_GetReg(uint16_t index, uint8_t * data);
```

Returns

0 - when call was passed

Description

none

Parameters

Parameters	Description
index	register number
*data	array to store data

Function

uint8_t DRV_GFX_SSD1926_GetReg(uint16_t index, uint8_t *data)

DRV_GFX_SSD1926_SetReg Function

updates graphics controller register value (byte access)

File

drv_gfx_ssd1926.h

C

```
uint16_t DRV_GFX_SSD1926_SetReg(uint16_t index, uint8_t value);
```

Returns

1 - call was not passed 0 - call was passed

Description

none

Parameters

Parameters	Description
index	register number
value	value to write to register

Function

uint8_t DRV_GFX_SSD1926_SetReg(uint16_t index, uint8_t value)

DRV_GFX_SSD1926_Close Function

closes an instance of the graphics controller

File

drv_gfx_ssd1926.h

C

```
void DRV_GFX_SSD1926_Close(DRV_HANDLE handle);
```

Description

none

Function

void DRV_GFX_SSD1926_Close(DRV_HANDLE handle)

DRV_GFX_SSD1926_Initialize Function

resets LCD, initializes PMP

File

drv_gfx_ssd1926.h

C

```
SYS_MODULE_OBJ DRV_GFX_SSD1926_Initialize(const SYS_MODULE_INDEX moduleIndex, const SYS_MODULE_INIT * const moduleInit);
```

Returns

1 - call not successful (PMP driver busy) 0 - call successful

Description

none

Parameters

Parameters	Description
instance	driver instance

Function

SYS_MODULE_OBJ DRV_GFX_SSD1926_Initialize(const SYS_MODULE_INDEX moduleIndex,
const SYS_MODULE_INIT * const moduleInit)

DRV_GFX_SSD1926_Open Function

opens an instance of the graphics controller

File

drv_gfx_ssd1926.h

C

```
DRV_HANDLE DRV_GFX_SSD1926_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Description

none

Function

DRV_GFX_SSD1926_Open(uint8_t instance)

DRV_GFX_SSD1926_SetColor Function

Sets the color for the driver instance

File

drv_gfx_ssd1926.h

C

```
void DRV_GFX_SSD1926_SetColor(GFX_COLOR color);
```

Returns

none

Function

void DRV_GFX_SSD1926_SetColor(GFX_COLOR color)

DRV_GFX_SSD1926_Status Function

Provides the current status of the SSD1926 driver module.

File

drv_gfx_ssd1926.h

C

```
SYS_STATUS DRV_GFX_SSD1926_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Description

This function provides the current status of the SSD1926 driver module.

Preconditions

The DRV_GFX_SSD1926_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_GFX_SSD1926_Initialize
SYS_STATUS          status;

status = DRV_GFX_SSD1926_Status( object );
```

```
if( SYS_STATUS_READY != status )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_GFX_SSD1926_Initialize

Function

SYS_STATUS DRV_GFX_SSD1926_Status (SYS_MODULE_OBJ object)

DRV_GFX_SSD1926_SetInstance Function

Sets the instance for the driver

File

drv_gfx_ssd1926.h

C

```
void DRV_GFX_SSD1926_SetInstance(uint8_t instance);
```

Returns

none

Function

void DRV_GFX_SSD1926_SetInstance(uint8_t instance)

DRV_GFX_SSD1926_Tasks Function

Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer

File

drv_gfx_ssd1926.h

C

```
void DRV_GFX_SSD1926_Tasks(SYS_MODULE_OBJ object);
```

Function

void DRV_GFX_SSD1926_Tasks(void)

DRV_GFX_SSD1926_InterfaceSet Function

Returns the API of the graphics controller

File

drv_gfx_ssd1926.h

C

```
void DRV_GFX_SSD1926_InterfaceSet(DRV_HANDLE handle, DRV_GFX_INTERFACE * interface);
```

Description

none

Function

void DRV_GFX_SSD1926_InterfaceSet(DRV_HANDLE handle, DRV_GFX_INTERFACE * interface)

DRV_GFX_SSD1926_MaxXGet Function

Returns x extent of the display.

File

drv_gfx_ssd1926.h

C

```
uint16_t DRV_GFX_SSD1926_MaxXGet();
```

Example

```
Remarks:
```

Function

void DRV_GFX_SSD1926_MaxXGet()

DRV_GFX_SSD1926_MaxYGet Function

Returns y extent of the display.

File

drv_gfx_ssd1926.h

C

```
uint16_t DRV_GFX_SSD1926_MaxYGet();
```

Example

```
Remarks:
```

Function

void GFX_MaxYGet()

DRV_GFX_SSD1926_BarFill Function

Hardware accelerated barfill function

File

drv_gfx_ssd1926.h

C

```
void DRV_GFX_SSD1926_BarFill(uint16_t left, uint16_t top, uint16_t right, uint16_t bottom);
```

Returns

1 - call not successful (PMP driver busy) 0 - call successful

Description

see primitive BarFill

Function

void DRV_GFX_SSD1926_BarFill(uint16_t left, uint16_t top, uint16_t right, uint16_t bottom)

DRV_GFX_SSD1926_PixelArrayGet Function

gets an array of pixels of length count starting at *color

File

drv_gfx_ssd1926.h

C

```
uint16_t* DRV_GFX_SSD1926_PixelArrayGet(uint16_t * color, uint16_t x, uint16_t y, uint16_t count);
```

Returns

NULL - call not successful !NULL - address of the display driver queue command

Description

none

Parameters

Parameters	Description
instance	driver instance
*color	start of the array
x	x coordinate of the start point.
y	y coordinate of the end point.
count	number of pixels

Function

uint16_t DRV_GFX_SSD1926_PixelArrayGet(uint16_t *color, short x, short y, uint16_t count)

DRV_GFX_SSD1926_PixelArrayPut Function

outputs an array of pixels of length count starting at *color

File

drv_gfx_ssd1926.h

C

```
void DRV_GFX_SSD1926_PixelArrayPut(uint16_t * color, uint16_t x, uint16_t y, uint16_t count, uint16_t
lineCount);
```

Returns

NULL - call not successful !NULL - handle to the number of pixels remaining

Description

none

Parameters

Parameters	Description
*color	start of the array
x	x coordinate of the start point.
y	y coordinate of the end point.
count	number of pixels
lineCount	number of lines

Function

void DRV_GFX_SSD1926_PixelArrayPut(uint16_t *color, uint16_t x, uint16_t y, uint16_t count, uint16_t lineCount)

DRV_GFX_SSD1926_PixelPut Function

outputs one pixel into the frame buffer at the x,y coordinate given

File

drv_gfx_ssd1926.h

C

```
void DRV_GFX_SSD1926_PixelPut(uint16_t x, uint16_t y);
```

Returns

NULL - call not successful !NULL - address of the display driver queue command

Description

none

Parameters

Parameters	Description
x,y	pixel coordinates

Function

uint16_t DRV_GFX_SSD1926_PixelPut(short x, short y)

SSD1926 Driver Data Types and Constants

DRV_GFX_SSD1926_INDEX_COUNT Macro

Number of valid SSD1926 driver indices.

File

drv_gfx_ssd1926.h

C

```
#define DRV_GFX_SSD1926_INDEX_COUNT DRV_GFX_SSD1926_NUMBER_OF_MODULES
```

Description

SSD1926 Driver Module Index Count

This constant identifies SSD1926 driver index definitions.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is device-specific.

DRV_GFX_SSD1926_COMMAND Structure

Structure for the commands in the driver queue.

File

drv_gfx_ssd1926.h

C

```
typedef struct {
  uint32_t address;
  uint16_t * array;
  uint16_t data;
} DRV_GFX_SSD1926_COMMAND;
```

Members

Members	Description
uint32_t address;	whether or not the task is complete

Description

Structure: DRV_GFX_SSD1926_COMMAND

Structure for the commands in the driver queue.

Parameters

Parameters	Description
address	pixel address
array	pointer to array of pixel data
data	pixel color

OTM2201A Driver Functions

DRV_GFX_OTM2201A_AddressSet Function

Sets the start GRAM address where pixel data to be written

File

drv_gfx_otm2201a.h

C

```
uint16_t DRV_GFX_OTM2201A_AddressSet(uint32_t address);
```

Returns

DRV_OTM2201A_ERROR_PMP_WRITE - returns error during PMP Write, DRV_OTM2201A_ERROR_NO_ERROR - returns success without any error.

Description

Address consists of Lower 8 bit at Register REG_RAM_ADDR_LOW and Higher 8 bit at Register REG_RAM_ADDR_HIGH

Parameters

Parameters	Description
address	pixel address

Function

uint16_t DRV_GFX_OTM2201A_AddressSet(uint32_t address)

DRV_GFX_OTM2201A_BrightnessSet Function

Sets the brightness of the display backlight.

File

drv_gfx_otm2201a.h

C

```
void DRV_GFX_OTM2201A_BrightnessSet(uint8_t instance, uint16_t level);
```

Returns

none

Description

Sets the brightness of the display backlight.

Parameters

Parameters	Description
instance	instance of the driver
level	Brightness level. Valid values are 0 to 100. 0 = brightness level is zero or display is turned off. 100 = brightness level is maximum.

Function

uint8_t DRV_GFX_OTM2201A_BrightnessSet(

uint8_t instance,

uint16_t level

)

DRV_GFX_OTM2201A_RegGet Function

Returns graphics controller register value (byte access)

File

drv_gfx_otm2201a.h

C

```
uint8_t DRV_GFX_OTM2201A_RegGet(uint16_t index, uint16_t * data);
```

Returns

DRV_OTM2201A_ERROR_PMP_WRITE - returns error during PMP Write, DRV_OTM2201A_ERROR_PMP_READ - returns error during PMP Read, DRV_OTM2201A_ERROR_NO_ERROR - returns success without any error.

Description

Returns graphics controller register value (byte access)

Parameters

Parameters	Description
index	register number
*data	array to store register data

Function

uint8_t DRV_GFX_OTM2201A_RegGet(

uint16_t index,

uint8_t *data

)

DRV_GFX_OTM2201A_Busy Function

Returns non-zero value if LCD controller is busy (previous drawing operation is not completed).

File

drv_gfx_otm2201a.h

C

```
uint16_t DRV_GFX_OTM2201A_Busy(uint8_t instance);
```

Returns

DRV_OTM2201A_ERROR_DEVICE_BUSY - Device is busy, DRV_OTM2201A_ERROR_NO_ERROR - Success, driver is not busy.

Description

Returns non-zero value if LCD controller is busy (previous drawing operation is not completed).

Parameters

Parameters	Description
instance	driver instance

Function

uint16_t DRV_GFX_OTM2201A_Busy(uint8_t instance)

DRV_GFX_OTM2201A_Close Function

closes an instance of the graphics controller

File

drv_gfx_otm2201a.h

C

```
void DRV_GFX_OTM2201A_Close(DRV_HANDLE handle);
```

Returns

0 - instance closed 2 - instance doesn't exist 3 - instance already closed

Description

none

Function

DRV_GFX_OTM2201A_Close(uint8_t instance)

DRV_GFX_OTM2201A_ColorSet Function

Sets the color for the driver instance

File

drv_gfx_otm2201a.h

C

```
void DRV_GFX_OTM2201A_ColorSet(GFX_COLOR color);
```

Returns

none

Description

Sets the color for the driver instance

Parameters

Parameters	Description
color	16 bit 565 format color value

Function

void DRV_GFX_OTM2201A_ColorSet(GFX_COLOR color)

DRV_GFX_OTM2201A_Initialize Function

resets LCD, initializes PMP

File

drv_gfx_otm2201a.h

C

```
SYS_MODULE_OBJ DRV_GFX_OTM2201A_Initialize(const SYS_MODULE_INDEX moduleIndex, const SYS_MODULE_INIT *
const moduleInit);
```

Returns

NULL - call not successful (PMP driver busy) !NULL - address of the display driver queue command

Description

none

Parameters

Parameters	Description
instance	driver instance

Function

uint16_t DRV_GFX_OTM2201A_Initialize(uint8_t instance)

DRV_GFX_OTM2201A_InstanceSet Function

Sets the instance for the driver

File

drv_gfx_otm2201a.h

C

```
void DRV_GFX_OTM2201A_InstanceSet(uint8_t instance);
```

Returns

none

Description

Sets the instance for the driver

Parameters

Parameters	Description
instance	driver instance

Function

void DRV_GFX_OTM2201A_InstanceSet(uint8_t instance)

DRV_GFX_OTM2201A_Open Function

opens an instance of the graphics controller

File

drv_gfx_otm2201a.h

C

```
DRV_HANDLE DRV_GFX_OTM2201A_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Returns

1 - driver not initialized 2 - instance doesn't exist 3 - instance already open instance to driver when successful

Description

none

Function

DRV_GFX_OTM2201A_Open(uint8_t instance)

DRV_GFX_OTM2201A_RegSet Function

Updates graphics controller register value (byte access)

File

drv_gfx_otm2201a.h

C

```
uint16_t DRV_GFX_OTM2201A_RegSet(uint16_t index, uint16_t value);
```

Returns

DRV_OTM2201A_ERROR_PMP_WRITE - returns error during PMP Write, DRV_OTM2201A_ERROR_NO_ERROR - returns success without any error.

Description

This call can set "value" of the register accessed by its "index" and can repeat the same by number of times mentioned in "repeatCount"

Parameters

Parameters	Description
index	register number
value	value to write to register
repeatCount	repeatCount number of times value is to be written to the register.

Function

uint8_t DRV_GFX_OTM2201A_RegSet(

uint16_t index,

uint8_t value,

uint32_t repeatCount

)

DRV_GFX_OTM2201A_Tasks Function

Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer

File

drv_gfx_otm2201a.h

C

```
void DRV_GFX_OTM2201A_Tasks(SYS_MODULE_OBJ object);
```

Function

void DRV_GFX_OTM2201A_Tasks(void)

DRV_GFX_OTM2201A_InterfaceSet Function

Returns the API of the graphics controller

File

drv_gfx_otm2201a.h

C

```
void DRV_GFX_OTM2201A_InterfaceSet(DRV_HANDLE handle, DRV_GFX_INTERFACE * interface);
```

Description

none

Function

void DRV_GFX_OTM2201A_InterfaceSet(DRV_HANDLE handle, DRV_GFX_INTERFACE * interface)

DRV_GFX_OTM2201A_MaxXGet Function

Returns x extent of the display.

File

drv_gfx_otm2201a.h

C

```
uint16_t DRV_GFX_OTM2201A_MaxXGet();
```

Example

```
Remarks:
```

Function

void DRV_GFX_OTM2201A_MaxXGet(void)

DRV_GFX_OTM2201A_MaxYGet Function

Returns y extent of the display.

File

drv_gfx_otm2201a.h

C

```
uint16_t DRV_GFX_OTM2201A_MaxYGet();
```

Example

```
Remarks:
```

Function

void DRV_GFX_OTM2201A_MaxYGet(void)

DRV_GFX_OTM2201A_BarFill Function

Outputs count number of pixels into the frame buffer from the given x,y coordinate.

File

drv_gfx_otm2201a.h

C

```
void DRV_GFX_OTM2201A_BarFill(uint16_t left, uint16_t top, uint16_t right, uint16_t bottom);
```

Returns

DRV_OTM2201A_ERROR_QUEUE_FULL - OTM2201A command queue is full, DRV_OTM2201A_ERROR_NO_ERROR - Success without any error.

Description

Outputs count number of pixels into the frame buffer from the given x,y coordinate.

Parameters

Parameters	Description
left	pixel coordinate on x axis
top	pixel coordinate on y axis
right	pixel coordinate on x axis
bottom	pixel coordinate on y axis

Function

uint16_t DRV_GFX_OTM2201A_BarFill(

short left,

short top,

short right,

short bottom

)

DRV_GFX_OTM2201A_PixelArrayGet Function

Gets an array of pixels of length count into an array starting at *color

File

drv_gfx_otm2201a.h

C

```
uint16_t* DRV_GFX_OTM2201A_PixelArrayGet(uint16_t * color, uint16_t x, uint16_t y, uint16_t count);
```

Returns

DRV_OTM2201A_ERROR_QUEUE_FULL - OTM2201A command queue is full, DRV_OTM2201A_ERROR_NO_ERROR - Success without any error.

Description

Gets an array of pixels of length count into an array starting at *color

Parameters

Parameters	Description
color	Pointer to array where color data is to be loaded
x	pixel coordinate on x axis
y	pixel coordinate on y axis
count	count number of pixels

Function

uint16_t DRV_GFX_OTM2201A_PixelArrayGet(uint16_t *color,

short x,

short y,
uint16_t count)

DRV_GFX_OTM2201A_PixelArrayPut Function

Outputs an array of pixels of length count starting at *color

File

drv_gfx_otm2201a.h

C

```
void DRV_GFX_OTM2201A_PixelArrayPut(uint16_t * color, uint16_t x, uint16_t y, uint16_t count, uint16_t
lineCount);
```

Returns

handle - handle to the number of pixels remaining, DRV_OTM2201A_ERROR_QUEUE_FULL - OTM2201A command queue is full.

Description

Outputs an array of pixels of length count starting at *color

Parameters

Parameters	Description
color	pointer to array of color of pixels
x	pixel coordinate on x axis.
y	pixel coordinate on y axis.
count	count number of pixels
lineCount	lineCount number of display lines

Function

void DRV_GFX_OTM2201A_PixelArrayPut(
uint16_t *color,
uint16_t x,
uint16_t y,
uint16_t count,
uint16_t lineCount
)

DRV_GFX_OTM2201A_PixelPut Function

Outputs one pixel into the frame buffer at the x,y coordinate given

File

drv_gfx_otm2201a.h

C

```
void DRV_GFX_OTM2201A_PixelPut(uint16_t x, uint16_t y);
```

Returns

DRV_OTM2201A_ERROR_QUEUE_FULL - OTM2201A command queue is full, DRV_OTM2201A_ERROR_NO_ERROR - Success without any error.

Description

Outputs one pixel into the frame buffer at the x,y coordinate given

Parameters

Parameters	Description
x	pixel coordinate on x axis
y	pixel coordinate on y axis

Function

void DRV_GFX_OTM2201A_PixelPut(uint16_t x, uint16_t y)

OTM2201A Driver Data Types and Constants

OTM2201A_TASK Enumeration

Enumeration for command type.

File

drv_gfx_otm2201a.h

C

```
typedef enum {
  INITIALIZE = 0,
  BUSY,
  PUT_ARRAY,
  PUT_PIXELS
} OTM2201A_TASK;
```

Description

Enum: OTM2201A_TASK

Enumeration for command type.

Parameters

Parameters	Description
INITIALIZE	driver initialization task
BUSY	driver busy task
PUT_ARRAY	driver put array task
PUT_PIXELS	driver put pixels task

DRV_GFX_OTM2201A_COMMAND Structure

Structure for the commands in the driver queue.

File

drv_gfx_otm2201a.h

C

```
typedef struct {
  uint8_t instance;
  uint32_t address;
  uint16_t * array;
  uint16_t data;
  uint16_t count;
  uint16_t lineCount;
  OTM2201A_TASK task;
} DRV_GFX_OTM2201A_COMMAND;
```

Description

Structure: DRV_GFX_OTM2201A_COMMAND

Structure for the commands in the driver queue.

Parameters

Parameters	Description
instance	instance of the driver
address	pixel address
array	pointer to array of pixel data
data	pixel color
count	count number of pixels in one line

lineCount	lineCount number of lines of display
task	Type of task (OTM2201A_TASK enum)

DRV_GFX_OTM2201A_INDEX_COUNT Macro

Number of valid OTM2201A driver indices.

File

drv_gfx_otm2201a.h

C

```
#define DRV_GFX_OTM2201A_INDEX_COUNT DRV_GFX_OTM2201A_NUMBER_OF_MODULES
```

Description

This constant identifies OTM2201A driver index definitions.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is device-specific.

Section

Data Types and Constants

****

****

****

OTM2201A Driver Module Index Count

tft002 Driver Functions

tft002 Driver Data Types and Constants

Files

Files

Name	Description
drv_gfx_lcc.h	Interface for the graphics library where the primitives are rendered and sent to the graphics controller either external or internal
drv_gfx_s1d13517.h	Interface for the graphics library where the primitives are rendered and sent to the graphics controller either external or internal
drv_gfx_ssd1289.h	Interface for the graphics library, which initializes the SYSTECH SSD1289 Timing Controller.
drv_gfx_ssd1926.h	Interface for the graphics library where the primitives are rendered and sent to the graphics controller either external or internal
drv_gfx_otm2201a.h	Interface for the graphics library where the primitives are rendered and sent to the graphics controller either external or internal
drv_gfx_tft002.h	This is file drv_gfx_tft002.h.
drv_gfx_lcc_config_template.h	This header file template defines the configurations for Graphics Library Driver LCC.
drv_gfx_s1d13517_config_template.h	This header file template defines the configurations for Graphics Library Driver S1D13517.
drv_gfx_ssd1926_config_template.h	This header file template defines the configurations for Graphics Library Driver SSD1926.
drv_gfx_otm2201a_config_template.h	This header file template defines the configurations for Graphics Library Driver OTM2201A.
drv_gfx_tft002_config_template.h	This is file drv_gfx_tft002_config_template.h.

Description

This section lists the source and header files used by the Graphics Driver Library.

drv_gfx_lcc.h

Interface for the graphics library where the primitives are rendered and sent to the graphics controller either external or internal

Enumerations

	Name	Description
	DMA_ISR_TASK	This is type DMA_ISR_TASK.
	DRV_GFX_LCC_DISPLAY_WRITE_BUS_TYPE	
	DRV_GFX_LCC_FB_WRITE_BUS_TYPE	
	LCC_VSYNC_STATE	This is type LCC_VSYNC_STATE.

Functions

	Name	Description
	DRV_GFX_LCC_AlphaBlendWindow	DOM-IGNORE-END
	DRV_GFX_LCC_BarFill	outputs one pixel into the frame buffer at the x,y coordinate given
	DRV_GFX_LCC_Close	closes an instance of the graphics controller
	DRV_GFX_LCC_DisplayRefresh	LCD refresh handler
	DRV_GFX_LCC_FrameBufferAddressSet	Sets address of the framebuffer
	DRV_GFX_LCC_GetBuffer	DOM-IGNORE-END
	DRV_GFX_LCC_Initialize	resets LCD, initializes PMP
	DRV_GFX_LCC_InterfaceSet	Returns the API of the graphics controller
	DRV_GFX_LCC_MaxXGet	Returns x extent of the display.
	DRV_GFX_LCC_MaxYGet	Returns y extent of the display.
	DRV_GFX_LCC_Open	opens an instance of the graphics controller
	DRV_GFX_LCC_PixelArrayGet	gets an array of pixels of length count starting at *color
	DRV_GFX_LCC_PixelArrayPut	outputs an array of pixels of length count starting at *color
	DRV_GFX_LCC_PixelPut	outputs one pixel into the frame buffer at the x,y coordinate given
	DRV_GFX_LCC_SetColor	Sets the color for the driver instance
	DRV_GFX_LCC_SetPage	DOM-IGNORE-START
	DRV_GFX_LCC_Tasks	Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer
	DRV_GFX_LCC_VSYNC_CallbackSet	Sets a callback function pointer for notification of VSYNC state
	DRV_GFX_LCC_VSYNC_GetState	Gets the state of the current VSYNC mode
	DRV_GFX_PaletteSet	returns address to the framebuffer.
	GFX_PRIM_SetPIPWindow	returns address to the framebuffer.

Macros

	Name	Description
	DRV_GFX_LCC_INDEX_COUNT	Number of valid LCC driver indices.
	PIP_BUFFER	This is macro PIP_BUFFER.

Types

	Name	Description
	LCC_VSYNC_Callback_FnPtr	function pointer definition for VSYNC callback

Description

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Low-Cost Controllerless (LCC) Graphics Controller.

File Name

drv_gfx_lcc.h

Company

Microchip Technology Inc.

drv_gfx_s1d13517.h

Interface for the graphics library where the primitives are rendered and sent to the graphics controller either external or internal

Functions

	Name	Description
	DRV_GFX_S1D13517_AlphaBlendWindow	SEE primitive layer alphablendWindow definition
	DRV_GFX_S1D13517_BarFill	outputs one pixel into the frame buffer at the x,y coordinate given
	DRV_GFX_S1D13517_BrightnessSet	Sets the brightness of the display backlight.
	DRV_GFX_S1D13517_Close	closes an instance of the graphics controller
	DRV_GFX_S1D13517_GetReg	returns graphics controllerS1D13517_REGister value (byte access)
	DRV_GFX_S1D13517_Initialize	resets LCD, initializes PMP
	DRV_GFX_S1D13517_InterfaceSet	Returns the API of the graphics controller
	DRV_GFX_S1D13517_Layer	Updates a Layer depending on the layer parameters.
	DRV_GFX_S1D13517_MaxXGet	Returns x extent of the display.
	DRV_GFX_S1D13517_MaxYGet	Returns y extent of the display.
	DRV_GFX_S1D13517_Open	opens an instance of the graphics controller
	DRV_GFX_S1D13517_PixelArrayPut	outputs an array of pixels of length count starting at *color
	DRV_GFX_S1D13517_PixelPut	outputs one pixel into the frame buffer at the x,y coordinate given
	DRV_GFX_S1D13517_SetColor	Sets the color for the driver instance
	DRV_GFX_S1D13517_SetInstance	Sets the instance for the driver
	DRV_GFX_S1D13517_SetPage	Sets the page of a certain page type
	DRV_GFX_S1D13517_SetReg	updates graphics controllerS1D13517_REGister value (byte access)
	DRV_GFX_S1D13517_Tasks	Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer

Macros

	Name	Description
	DRV_GFX_S1D13517_INDEX_COUNT	Number of valid S1D13517 driver indices.

Structures

	Name	Description
	LAYER_REGISTERS	This structure is used to describe layerS1D13517_REGisters.

Description

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the S1D13517 Graphics Controller.

File Name

drv_gfx_s1d13517.c

Company

Microchip Technology Inc.

drv_gfx_ssd1289.h

Interface for the graphics library, which initializes the SYSTECH SSD1289 Timing Controller.

Functions

	Name	Description
	GFX_TCON_SSD1289Init	Initialize the Solomon Systech SSD1289 Timing Controller.

Description

Module for Microchip Graphics Library

This header file contains the function definition for the interface to the SYSTECH SSD1289 Timing Controller.

File Name

drv_gfx_ssd1289.h

Company

Microchip Technology Inc.

drv_gfx_ssd1926.h

Interface for the graphics library where the primitives are rendered and sent to the graphics controller either external or internal

Functions

	Name	Description
	DRV_GFX_SSD1926_BarFill	Hardware accelerated barfill function
	DRV_GFX_SSD1926_Busy	Returns non-zero if LCD controller is busy (previous drawing operation is not completed).
	DRV_GFX_SSD1926_Close	closes an instance of the graphics controller
	DRV_GFX_SSD1926_GetReg	returns graphics controller register value (byte access)
	DRV_GFX_SSD1926_Initialize	resets LCD, initializes PMP
	DRV_GFX_SSD1926_InterfaceSet	Returns the API of the graphics controller
	DRV_GFX_SSD1926_MaxXGet	Returns x extent of the display.
	DRV_GFX_SSD1926_MaxYGet	Returns y extent of the display.
	DRV_GFX_SSD1926_Open	opens an instance of the graphics controller
	DRV_GFX_SSD1926_PixelArrayGet	gets an array of pixels of length count starting at *color
	DRV_GFX_SSD1926_PixelArrayPut	outputs an array of pixels of length count starting at *color
	DRV_GFX_SSD1926_PixelPut	outputs one pixel into the frame buffer at the x,y coordinate given
	DRV_GFX_SSD1926_SetColor	Sets the color for the driver instance
	DRV_GFX_SSD1926_SetInstance	Sets the instance for the driver
	DRV_GFX_SSD1926_SetReg	updates graphics controller register value (byte access)
	DRV_GFX_SSD1926_Status	Provides the current status of the SSD1926 driver module.
	DRV_GFX_SSD1926_Tasks	Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer

Macros

	Name	Description
	DRV_GFX_SSD1926_INDEX_COUNT	Number of valid SSD1926 driver indices.

Structures

	Name	Description
	DRV_GFX_SSD1926_COMMAND	Structure for the commands in the driver queue.

Description

None

File Name

drv_gfx_ssd1926.h

Company

Microchip Technology Inc.

drv_gfx_otm2201a.h

Interface for the graphics library where the primitives are rendered and sent to the graphics controller either external or internal

Enumerations

	Name	Description
	OTM2201A_TASK	Enumeration for command type.

Functions

	Name	Description
	DRV_GFX_OTM2201A_AddressSet	Sets the start GRAM address where pixel data to be written
	DRV_GFX_OTM2201A_BarFill	Outputs count number of pixels into the frame buffer from the given x,y coordinate.
	DRV_GFX_OTM2201A_BrightnessSet	Sets the brightness of the display backlight.
	DRV_GFX_OTM2201A_Busy	Returns non-zero value if LCD controller is busy (previous drawing operation is not completed).
	DRV_GFX_OTM2201A_Close	closes an instance of the graphics controller
	DRV_GFX_OTM2201A_ColorSet	Sets the color for the driver instance
	DRV_GFX_OTM2201A_Initialize	resets LCD, initializes PMP
	DRV_GFX_OTM2201A_InstanceSet	Sets the instance for the driver
	DRV_GFX_OTM2201A_InterfaceSet	Returns the API of the graphics controller
	DRV_GFX_OTM2201A_MaxXGet	Returns x extent of the display.
	DRV_GFX_OTM2201A_MaxYGet	Returns y extent of the display.
	DRV_GFX_OTM2201A_Open	opens an instance of the graphics controller
	DRV_GFX_OTM2201A_PixelArrayGet	Gets an array of pixels of length count into an array starting at *color
	DRV_GFX_OTM2201A_PixelArrayPut	Outputs an array of pixels of length count starting at *color
	DRV_GFX_OTM2201A_PixelPut	Outputs one pixel into the frame buffer at the x,y coordinate given
	DRV_GFX_OTM2201A_RegGet	Returns graphics controller register value (byte access)
	DRV_GFX_OTM2201A_RegSet	Updates graphics controller register value (byte access)
	DRV_GFX_OTM2201A_Tasks	Task machine that renders the driver calls for the graphics library it must be called periodically to output the contents of its circular buffer

Macros

	Name	Description
	DRV_GFX_OTM2201A_INDEX_COUNT	Number of valid OTM2201A driver indices.

Structures

	Name	Description
	DRV_GFX_OTM2201A_COMMAND	Structure for the commands in the driver queue.

Description

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the OTM2201A Graphics Controller.

File Name

drv_gfx_otm2201a.h

Company

Microchip Technology Inc.

drv_gfx_tft002.h

This is file drv_gfx_tft002.h.

drv_gfx_lcc_config_template.h

This header file template defines the configurations for Graphics Library Driver LCC.

Macros

	Name	Description
	DRV_GFX_CONFIG_LCC_EXTERNAL_MEMORY	Macro that enables external memory framebuffer.
	DRV_GFX_CONFIG_LCC_INTERNAL_MEMORY	Macro that enables internal memory framebuffer.
	DRV_GFX_CONFIG_LCC_PALETTE	Macro that disables internal palette memory framebuffer.
	DRV_GFX_LCC_DMA_CHANNEL_INDEX	Macro that defines the DMA CHANNEL INDEX.

Description

Module for Microchip Graphics Library

File Name

drv_gfx_lcc_config_template.h

Company

Microchip Technology Inc.

drv_gfx_s1d13517_config_template.h

This header file template defines the configurations for Graphics Library Driver S1D13517.

Macros

	Name	Description
	GFX_CONFIG_S1D13517_DRIVER_COUNT	Macro sets the number of instances for the driver.

Description

Module for Microchip Graphics Library

File Name

drv_gfx_s1d13517_config_template.h

Company

Microchip Technology Inc.

drv_gfx_ssd1926_config_template.h

This header file template defines the configurations for Graphics Library Driver SSD1926.

Macros

	Name	Description
	GFX_CONFIG_SSD1926_DRIVER_COUNT	Macro sets the number of instances for the driver.

Description

Module for Microchip Graphics Library

File Name

drv_gfx_ssd1926_config_template.h

Company

Microchip Technology Inc.

drv_gfx_otm2201a_config_template.h

This header file template defines the configurations for Graphics Library Driver OTM2201A.

Macros

	Name	Description
	GFX_CONFIG_OTM2201A_DRIVER_COUNT	Macro sets the number of instances for the driver.

Description

Module for Microchip Graphics Library

File Name

drv_gfx_otm2201A_config_template.h

Company

Microchip Technology Inc.

drv_gfx_tft002_config_template.h

This is file drv_gfx_tft002_config_template.h.

I2C Driver Library Help

This section describes the I2C Driver Library.

Introduction

This library provides an interface to manage the data transfer operations using the I2C module on the Microchip family of microcontrollers.

Description

The driver communicates using the concept of *transactions*. In instances where the I2C operates in Master mode, the driver sends the start signal, followed by a slave device address (including a Read/Write bit), followed by a number of bytes written to or read from the slave. The *transaction* is completed by sending the stop signal. When the driver operates in the Slave mode, it will either read data or write data to the master.

This driver library provides application ready routines to read and write data using the I2C protocol, thus minimizing developer's awareness of the working of the I2C protocol.

- Provides read/write and buffer data transfer models
- Supports interrupt and Polled modes of operation
- Support multi-client and multi-instance operation
- Provides data transfer events
- Supports blocking and non-blocking operation
- Supports baud rate setting

Using the Library

This topic describes the basic architecture of the I2C Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_i2c.h`

The interface to the I2C Driver Library is defined in the `drv_i2c.h` header file. Any C language source (`.c`) file that uses the I2C Driver Library should include `drv_i2c.h`.

Library File: The I2C Driver Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

The I2C Driver Library provides the low-level abstraction of the I2C module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the I2C Driver Library interface.

Description

The I2C Driver Library features routines to perform two functions, driver maintenance and data transfer:

Driver Maintenance

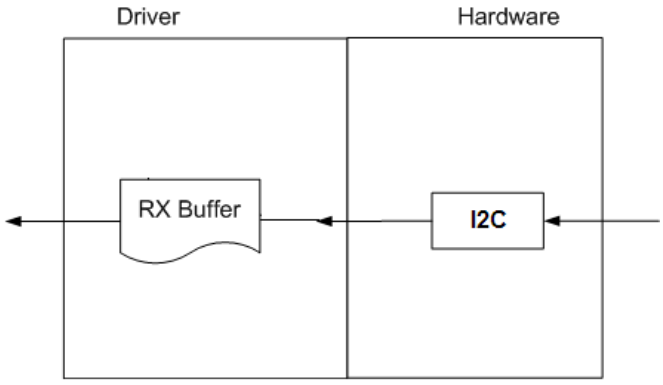
The Driver initialization routines allow the application to initialize the driver. The initialization data configures the I2C module as a Master or a Slave and sets the necessary parameters required for operation in the particular mode. The driver must be initialized before it can be used by the application. After the end of operation, the driver can be deinitialized.

Data Transfer

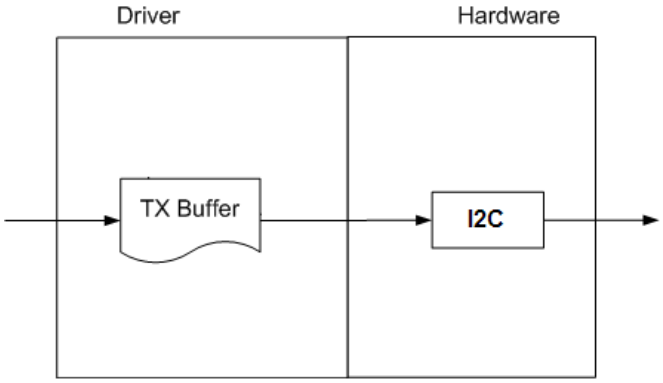
Data transfer is accomplished by separate Write and Read functions through a data buffer. The read and write function makes the user transparent to the internal working of the I2C protocol. The user can use callback mechanisms or use polling to check status of transfer.

The following diagrams illustrate the model used by the I2C Driver for transmitter and receiver.

Receiver Abstraction Model



Transmitter Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the I2C Driver Library.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Setup Functions	Provides open, close, status and other setup functions.
Data Transfer Functions	Provides data transfer functions available in the configuration.
Miscellaneous Functions	Provides miscellaneous driver functions.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

System Access

This section provides information on system access.

Description

System Access

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During

system initialization, each instance of the I2C module would be initialized with the following configuration settings (either passed dynamically at run-time using DRV_I2C_INIT or by using initialization overrides) that are supported by the specific I2C device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- The actual peripheral ID enumerated as the PLIB level module ID (e.g., I2C_ID_2)
- Master or Slave mode of operation and their associated parameters
- Defining the respective interrupt sources for Master, Slave, and Error Interrupt

The DRV_I2C_Initialize API returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the Initialize interface would be used by the other system interfaces like DRV_I2C_Deinitialize, DRV_I2C_Status, and DRV_I2C_Tasks.

Note: The system initialization settings, only affect the instance of the peripheral that is being initialized.

Example:

```
DRV_I2C_INIT        i2c_init_data;
SYS_MODULE_OBJ      objectHandle;

i2c_init_data.i2cId = DRV_I2C_PERIPHERAL_ID_IDX0,
i2c_init_data.i2cMode = DRV_I2C_MODE_MASTER,
OR
i2c_init_data.i2cMode = DRV_I2C_MODE_SLAVE,

/*    Master mode parameters */
i2c_init_data.baudRate = 100000,
i2c_init_data.busspeed = DRV_I2C_SLEW_RATE_CONTROL_IDX0,
i2c_init_data.buslevel = DRV_I2C_SMBus_SPECIFICATION_IDX0,

/*    Master mode parameters */
i2c_init_data.addWidth = DRV_I2C_7BIT_SLAVE,
i2c_init_data.reservedaddenable  = false,
i2c_init_data.generalcalladdress = false,
i2c_init_data.slaveaddvalue = 0x0060,

//interrupt sources
i2c_init_data.mstrInterruptSource = INT_SOURCE_I2C_2_MASTER,
i2c_init_data.slaveInterruptSource = INT_SOURCE_I2C_2_ERROR,
i2c_init_data.errInterruptSource = INT_SOURCE_I2C_2_ERROR,
i2c_init_data.queueSize = 1,

/* callback for Master (Master mode can use callbacks if needed) */
i2c_init_data.operationStarting = NULL,

/* Slave mode callbacks needed */
i2c_init_data.operationStarting = APP_I2CSlaveFunction,

objectHandle = DRV_I2C_Initialize(DRV_I2C_INDEX_0, (SYS_MODULE_INIT *)&drvI2C0InitData)
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
   // Handle error
}
```

Since the I2C bus is controlled by the Master, the Slave should respond to a read or write request whenever the Master makes the request. Thus, the slave does not have driver states like the Master. The operation of the I2C Driver when used in Slave mode is handled using callbacks. The callback, OperationStarting, must be configured during system initialization when in Slave mode. This callback is provided so that the application can respond appropriately when a read or write request is received from the Master.

Client Access

This section provides information on client access.

Description

For the application to start using an instance of the module, it must call the DRV_I2C_Open function. This provides the configuration required to open the I2C instance for operation. If the driver is deinitialized using the function DRV_I2C_Deinitialize, the application must call the DRV_I2C_Open function again to set up the instance of the I2C.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

After a client instance is opened, DRV_I2C_ClientSetup can be called to set up client-specific parameters. In I2C Slave mode, this is used to set-up the IRQ logic so that the slave can toggle this line to request Master to send a Read command.

As during initialization, when the I2C module operates in the Slave mode, only the Master can terminate a transaction with the Slave. In this case,

the driver provides a callback to the application after the reception of each byte from the Master or after transmission of a byte to the Master.

Example:

```
/* I2C Driver Handle  */
DRV_HANDLE  drvI2CHandle;

/* Open the I2C Driver */
appData.drvI2CHandle = DRV_I2C_Open( DRV_I2C_INDEX_0,DRV_IO_INTENT_WRITE );

if (drvI2CHandle != DRV_HANDLE_VALID)
{
     //Client cannot open instance
}
```

Client Transfer

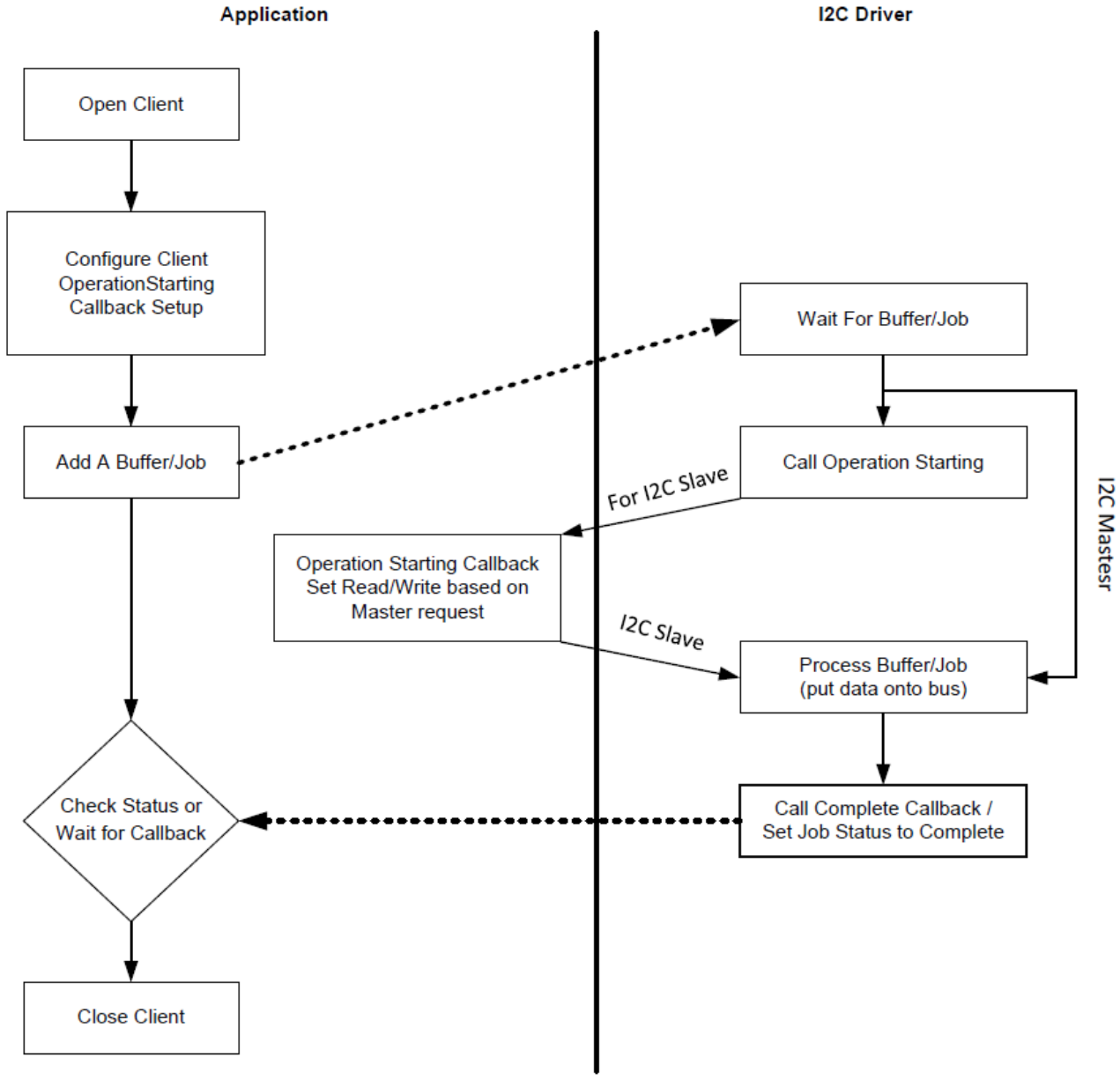
This section provides information on client transfer functionality.

Description

Core Functionality

Client basic functionality provides an extremely basic interface for the driver operation.

The following diagram illustrates the byte/word model used for the data transfer.



Client Data Transfer Functionality

Applications using the I2C driver need to perform the following:

1. The system should have completed necessary initialization and the DRV_I2C_Tasks should either be running in polled environment, or in an interrupt environment.
2. Open the driver using DRV_I2C_Open with the necessary intent.
3. Add a buffer using the DRV_I2C_Receive, DRV_I2C_Transmit, and DRV_I2C_TransmitThenReceive functions. An optional callback can be provided that will be called when the buffer/job is complete using DRV_I2C_BufferEventHandlerSet.
4. Check for the current transfer status using DRV_I2C_TransferStatusGet or wait for the callback to be called with buffer transfer status set to DRV_I2C_BUFFER_EVENT_COMPLETE or DRV_I2C_BUFFER_EVENT_ERROR.
5. Buffer status DRV_I2C_BUFFER_EVENT_COMPLETE implies that the I2C transaction has been completed without any errors. Buffer status DRV_I2C_BUFFER_EVENT_ERROR indicates that the I2C transaction was aborted and the entire contents of the buffer were not transferred.
6. In Master mode, common cases for DRV_I2C_BUFFER_EVENT_ERROR to be set are:

- Slave is non-operational
- Slave is performing an internal operation and cannot accept any more I2C messages from the Master until the operation completes. In such a case, if the Master tries to address the Slave and is attempting to transfer data, the Slave NACKs the transfer. This will result in the Master prematurely terminating the transaction and setting the DRV_I2C_BUFFER_EVENT_FLAG. In the application level, the Master can continuously attempt to send the transaction until transfer status changes from for DRV_I2C_BUFFER_EVENT_ERROR to DRV_I2C_BUFFER_EVENT_COMPLETE. This will in effect perform the so-called "Acknowledge Polling". An example of a Slave device that depicts this behavior is an EEPROM.

7. The client will be able to close the driver using DRV_I2C_Close when required.

Example:

```
/*  This example demonstrates the I2C driver setup of one instance of I2C acting
    as a Master to another instance of the I2C Driver acting as a Slave.
    In the Slave initialization data structure in system_init.c, the member
    operationStarting should be assigned a function pointer. This function will
    be called when the Slave receives an address match. Based on the R/W bit in
    the address, the transmit or receive function will be called by the Slave
    (e.g., .operationStarting = APP_SlaveDataforMaster) */

SYS_MODULE_OBJ i2cMasterObject;

SYS_MODULE_OBJ i2cSlaveObject;


/* function prototype of callback function */
void I2CMasterOpStatusCb ( DRV_I2C_BUFFER_EVENT event,
                           DRV_I2C_BUFFER_HANDLE bufferHandle,
                           uintptr_t context);

int main( void )
{
    while ( 1 )
    {
        appTask ();
    }
}

void appTask ()
{
    #define MY_BUFFER_SIZE             5
    #define RTCC_SLAVE_ADDRESS        0xDE

    /* initialize slave address value */
    unsigned char address             =    RTCC_SLAVE_ADDRESS;

    /*Initialize myBuffer with MY_BUFFER_SIZE bytes of valid data */
    char myBuffer[MY_BUFFER_SIZE] = { 11, 22, 33, 44, 55};
    unsigned int numBytes;

    DRV_HANDLE drvI2CMasterHandle;        //Returned from DRV_I2C_Open for I2C Master
    DRV_I2C_BUFFER_HANDLE bufHandle_M1;   //Returned from calling a Data Transfer function
    uintptr_t i2cOpStatus;                //Operation status of I2C operation returned from callback

    DRV_HANDLE drvI2CSlaveHandle;         //Returned from DRV_I2C_Open for I2C Slave
    DRV_I2C_BUFFER_HANDLE bufHandle_S1;   //Returned from calling a Data Transfer function
    DRV_I2C_BUFFER_HANDLE bufHandle_S2;   //Returned from calling a Data Transfer function



    while( 1 )
    {
        switch( state )
        {
            case APP_STATE_INIT:
            {
                /* Initialize the Master I2C Driver */
                i2cMasterObject = DRV_I2C_Initialize( DRV_I2C_INDEX_0, (SYS_MODULE_INIT *)&drvI2C0InitData
);

                /* Initialize the Slave I2C Driver */
                i2cSlaveObject  = DRV_I2C_Initialize(DRV_I2C_INDEX_1, (SYS_MODULE_INIT *)&drvI2C1InitData);

                /* Check for the System Status */
                if( SYS_STATUS_READY != DRV_I2C_Status( i2cObject ) )
                return 0;

                /* Open the Driver for I2C Master */
                drvI2CMasterHandle = DRV_I2C_Open( DRV_I2C_INDEX_0,DRV_IO_INTENT_WRITE );
```

```
                if ( drvI2CMasterHandle != (DRV_HANDLE)NULL )
                {
                    /* event-handler set up receive callback from DRV_I2C_Tasks */
                    /* Event handler need to be set up only if needed */
                    DRV_I2C_BufferEventHandlerSet(drvI2CMasterHandle, I2CMasterOpStatusCb, i2cOpStatus );

                    /* Update the state to transfer data */
                    state = APP_STATE_DATA_PUT;
                }
                else
                {
                    state = APP_STATE_ERROR;
                }

                /* Open the I2C Driver for Slave on the same device */
                drvI2CSlaveHandle = DRV_I2C_Open( DRV_I2C_INDEX_1,DRV_IO_INTENT_WRITE );

                if ( drvI2CMasterHandle != (DRV_HANDLE)NULL )
                {
                    /* event-handler set up receive callback from DRV_I2C_Tasks */
                    /* Event handler need to be set up only if needed */
                    DRV_I2C_BufferEventHandlerSet(drvI2CMasterHandle, I2CMasterOpStatusCb, i2cOpStatus );

                    /* Update the state to transfer data */
                    state = APP_STATE_DATA_PUT;
                }
                else
                {
                    state = APP_STATE_ERROR;
                }


                break;
            }
            case APP_STATE_DATA_PUT:
            {
                /* I2C master writes data onto I2C bus */
                bufHandle_M1 = DRV_I2C_Transmit ( drvI2CMasterHandle , address, &myBuffer[], 5, NULL );

                /* Update the state to status check */
                state = APP_STATE_DATA_CHECK;
                break;
            }
            case APP_STATE_DATA_CHECK:
            {
                /* Check for the successful data transfer */
                if( DRV_I2C_BUFFER_EVENT_COMPLETE == DRV_I2C_TransferStatusGet
                                                    (drvI2CMasterHandle, bufHandle_M1) )
                {
                    /* Do this repeatedly */
                    state = APP_STATE_DATA_PUT;
                }
                break;
            }
            case APP_STATE_ERROR:
            {

                //include any error handling routines here

                break;
            }
            default:
            {
                break;
            }
        }
    }
```

```
/*****************************************************************************/
//  Function: I2CMasterOpStatusCb
//
//  Callback called in Master mode from the DRV_I2C_Tasks function. This
//  callback is invoked when the Master has to indicate to the application
//  that the BUFFER event is COMPLETE or there was an error in transmission.
//*****************************************************************************/

void I2CMasterOpStatusCb (    DRV_I2C_BUFFER_EVENT event,
                           DRV_I2C_BUFFER_HANDLE bufferHandle,
                           uintptr_t context)
{
    switch(event)
    {
        case DRV_I2C_BUFFER_EVENT_COMPLETE:
            //this indicates that the I2C transaction has completed
            //DRV_I2C_BUFFER_EVENT_COMPLETE can be handled in the callback
            //or by checking for this event using the API DRV_I2C_BufferStatus
            /* include any callback event handling code here if needed */
        break;
        case DRV_I2C_BUFFER_EVENT_ERROR:
            //this indicates that the I2C transaction has completed
            //and a STOP condition has been asserted on the bus.
            //However the slave has NACKED either the address or data
            //byte.
            /* include any callback event handling code here if needed */
        default:
        break;
    }

}

//*****************************************************************************/
//  Function: APP_SlaveDataforMaster
//
//  Callback function from DRV_I2C_Tasks when operating as a Slave. When an
//  address match is received by the Slave, this callback is executed and
//  the buffer event depends on the R/W bit. If R/W = 0, DRV_I2C_Receive is
//  called implying the Slave is going to read data send from the Master.
//  If R/W = 1, DRV_I2C_Transmit is called implying the Slave is going to send
//  data to the Master.
//*****************************************************************************/

void APP_SlaveDataforMaster(DRV_I2C_BUFFER_EVENT event, void * context)
{
    switch (event)
    {
        case DRV_I2C_BUFFER_SLAVE_READ_REQUESTED:
            deviceAddressPIC32 = PIC32_SLAVE_ADDRESS;

            bufHandle_S1 = DRV_I2C_Receive( drvI2CSlaveHandle,
                                            deviceAddressPIC32,
                                            &SlaveRxbuffer[0],
                                            NUMBER_OF_UNKNOWN_BYTES_TO_SLAVE,
                                            NULL );

            break;
        case DRV_I2C_BUFFER_SLAVE_WRITE_REQUESTED:
            deviceAddressPIC32 = PIC32_SLAVE_ADDRESS;

            bufHandle_S2 = DRV_I2C_Transmit ( drvI2CSlaveHandle,
                                              deviceAddressPIC32,
                                              &SlaveTxbuffer[0],
                                              NUMBER_OF_UNKNOWN_BYTES_TO_SLAVE,
                                              NULL );
            break;
        default:
            break;
    }
```

```
}

void __ISR(_I2C_2_VECTOR, ipl4AUTO) _IntHandlerDrvI2CInstance0(void)
{
   DRV_I2C_Tasks(i2cMasterObject);
}

void __ISR(_I2C1_SLAVE_VECTOR, ipl6AUTO) _IntHandlerDrvI2CSlaveInstance1(void)
{
    DRV_I2C_Tasks(i2cSlaveObject);
}
```

Configuring the Library

Macros

	Name	Description
	DRV_DYNAMIC_BUILD	Dynamic driver build, dynamic device instance parameters.
	DRV_I2C_CONFIG_BUILD_TYPE	Selects static or dynamic driver build configuration.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BASIC	Enables the device driver to support basic transfer mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BLOCKING	Enables the device driver to support blocking operations.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_EXCLUSIVE	Enables the device driver to support operation in Exclusive mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_MASTER	Enables the device driver to support operation in Master mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_NON_BLOCKING	Enables the device driver to support non-blocking during operations
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_READ	Enables the device driver to support read operations.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_SLAVE	Enables the device driver to support operation in Slave mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE	Enables the device driver to support write operations.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE_READ	Enables the device driver to support write followed by read.
	DRV_STATIC_BUILD	Static driver build, static device instance parameters.
	DRV_I2C_FORCED_WRITE	Includes function that writes to slave irrespective of whether receiving a ACK or NACK from slave
	I2C_STATIC_DRIVER_MODE	Selects the type of STATIC driver

Description

The configuration of the I2C Driver Library is based on the file `sys_config.h`.

This header file contains the configuration selection for the I2C Driver Library. Based on the selections made, the I2C Driver Library may support the selected features. These configuration settings will apply to all instances of the I2C Driver Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_DYNAMIC_BUILD Macro

Dynamic driver build, dynamic device instance parameters.

File

drv_i2c_config_template.h

C

```
#define DRV_DYNAMIC_BUILD 1
```

Description

Dynamic Driver Build Configuration

This value, if used to identify the build type for a driver, will cause the driver to be built to dynamically, identify the instance of the peripheral at run-time using the parameter passed into its API routines.

DRV_I2C_CONFIG_BUILD_TYPE Macro

Selects static or dynamic driver build configuration.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_BUILD_TYPE DRV_DYNAMIC_BUILD
```

Description

I2C Driver Build Configuration Type

This definition selects if I2C device driver is to be used with static or dynamic build parameters. Must be equated to one of the following values:

- DRV_STATIC_BUILD - Build the driver using static accesses to the peripheral identified by the DRV_I2C_INSTANCE macro
- DRV_DYNAMIC_BUILD - Build the driver using dynamic accesses to the peripherals

Affects all the drv_i2c.h driver functions.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BASIC Macro

Enables the device driver to support basic transfer mode.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BASIC
```

Description

Support Basic Transfer Mode

This definition enables the device driver to support basic transfer mode.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_BufferAddReadRead
- DRV_I2C_BufferAddReadWrite

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BLOCKING Macro

Enables the device driver to support blocking operations.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BLOCKING
```

Description

Support Blocking Operations

This definition enables the device driver to support blocking operations.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_Open
- DRV_I2C_Close
- DRV_I2C_BufferAddRead
- DRV_I2C_BufferAddWrite

- DRV_I2C_BufferAddWriteRead

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_EXCLUSIVE Macro

Enables the device driver to support operation in Exclusive mode.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_EXCLUSIVE
```

Description

Support Exclusive Mode

This definition enables the device driver to support operation in Exclusive mode.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_Open

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_MASTER Macro

Enables the device driver to support operation in Master mode.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_MASTER
```

Description

Support Master Mode

This definition enables the device driver to support operation in Master mode.

Remarks

During the configuration phase, the driver selects a list of operation modes that can be supported. While initializing a hardware instance, the device driver will properly perform the initialization base on the selected modes.

The device driver can support multiple modes within a single build.

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_NON_BLOCKING Macro

Enables the device driver to support non-blocking during operations

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_NON_BLOCKING
```

Description

Support Non-Blocking Operations

This definition enables the device driver to support non-blocking operations.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_Open
- DRV_I2C_Close

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_READ Macro

Enables the device driver to support read operations.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_READ
```

Description

Support Read Mode

This definition enables the device driver to support read operations.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_BufferAddRead

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_SLAVE Macro

Enables the device driver to support operation in Slave mode.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_SLAVE
```

Description

Support Slave Mode

This definition enables the device driver to support operation in Slave mode.

Remarks

During the configuration phase, the driver selects a list of operation modes that can be supported. While initializing a hardware instance, the device driver will properly perform the initialization base on the selected modes.

The device driver can support multiple modes within a single build.

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE Macro

Enables the device driver to support write operations.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE
```

Description

Support Write Mode

This definition enables the device driver to support write operations.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_BufferAddWrite

Refer to the description of each function in the corresponding help file for details.

DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE_READ Macro

Enables the device driver to support write followed by read.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE_READ
```

Description

Support Write followed by a Read using Restart

This definition enables the device driver to support write followed by read without relinquishing control of the bus. Restart is issued instead of Stop at the end of write. Stop is issued after read operation.

Remarks

The device driver can support multiple modes within a single build.

This definition affects the following functions:

- DRV_I2C_BufferAddWriteRead

Refer to the description of each function in the corresponding help file for details.

DRV_STATIC_BUILD Macro

Static driver build, static device instance parameters.

File

drv_i2c_config_template.h

C

```
#define DRV_STATIC_BUILD 0
```

Description

Static Driver Build Configuration

This value, if used to identify the build type for a driver, will cause the driver to be built using a specific statically identified instance of the peripheral.

DRV_I2C_FORCED_WRITE Macro

Includes function that writes to slave irrespective of whether receiving a ACK or NACK from slave

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_FORCED_WRITE true
```

Description

I2C driver objects configuration

When this option is checked, this will include Forced Write function. The Force Write function will send all data bytes to the slave irrespective of receiving ACK or NACK from slave. If writing data to the slave is invoked using DRV_I2C_Transfer, the transaction will be aborted if the Slave NACKs address or any data byte and a STOP condition will be send. This function is typically included for Slaves that require a special reset sequence.

Remarks

None

I2C_STATIC_DRIVER_MODE Macro

Selects the type of STATIC driver

File

drv_i2c_config_template.h

C

```
#define I2C_STATIC_DRIVER_MODE BUFFER_MODEL_STATIC
```

Description

I2C Static Driver type

This selects either the BYTE_MODEL_STATIC or BUFFER_MODEL_STATIC version of I2C driver. The BYTE_MODEL_STATIC version is equivalent to and is referred to as STATIC driver implementation in Harmony Versions 1.06.02 and below. This version of STATIC driver is not recommended for new design and will be deprecated in future release. The BUFFER_MODEL_STATIC supports transfer of buffers and is API compatible with the DYNAMIC implementation of I2C.

Building the Library

This section lists the files that are available in the I2C Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/i2c`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_i2c.h`	This file provides the interface definitions of the I2C driver.
`/drv_i2c_bb.h`	This file provides interface definitions that are transparent to the user when the I2C Driver is used in Bit-bang mode.
`/src/drv_i2c_local.h`	This file provides definitions of the data types that are used in the driver object.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_i2c.c`	This file contains the core implementation of the I2C driver.
`/src/dynamic/drv_i2c_bb.c`	This file implements the I2C Driver in Bit-bang mode.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files exist for this library.

Module Dependencies

The I2C Driver Library depends on the following modules:

- Clock System Service Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_I2C_Deinitialize	Deinitializes the index instance of the I2C module. **Implementation:** Static/Dynamic
	DRV_I2C_Initialize	Initializes hardware and data for the index instance of the I2C module. **Implementation:** Static/Dynamic
	DRV_I2C_Tasks	Maintains the State Machine of the I2C driver and performs all the protocol level actions. **Implementation:** Dynamic

b) Client Setup Functions

	Name	Description
	DRV_I2C_Close	Closes an opened instance of an I2C module driver. **Implementation:** Dynamic
	DRV_I2C_Open	Opens the specified instance of the I2C driver for use and provides an "open-instance" handle. **Implementation:** Dynamic

c) Data Transfer Functions

	Name	Description
	DRV_I2C_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Dynamic
	DRV_I2C_BytesTransferred	Returns the number of bytes transmitted or received in a particular I2C transaction. The transaction is identified by the handle.
	DRV_I2C_Receive	This function reads data written from either Master or Slave. **Implementation:** Dynamic
	DRV_I2C_Transmit	This function writes data to Master or Slave. **Implementation:** Dynamic
	DRV_I2C_TransmitThenReceive	This function writes data to Slave, inserts restart and requests read from slave. **Implementation:** Dynamic
	DRV_I2C_TransmitForced	This function writes data to Master or Slave. **Implementation:** Dynamic

d) Status Functions

	Name	Description
	DRV_I2C_TransferStatusGet	Returns status of data transfer when Master or Slave acts either as a transmitter or a receiver. **Implementation:** Dynamic
	DRV_I2C_Status	Provides the current status of the index instance of the I2C module. **Implementation:** Dynamic

e) Miscellaneous Functions

	Name	Description
	DRV_I2C_QueueFlush	The existing transactions in the queue are voided and the queue pointers are reset to their initial state. This renders the queue empty.

f) Data Types and Constants

	Name	Description
	_DRV_I2C_INIT	Identifies the initialization values that are passed as parameters to the initialize and reinitialize routines of the I2C module.
	DRV_I2C_ADDRESS_WIDTH	Lists the Address Width of the Slave.
	DRV_I2C_BUFFER_HANDLE	Handle to an I2C data buffer.
	DRV_I2C_BUS_LEVEL	Lists the Operational Voltage level of I2C.
	DRV_I2C_BUS_SPEED	Lists the I2C Bus speed mode.
	DRV_I2C_BUFFER_EVENT	Lists the different conditions that happens during a buffer transfer.

	DRV_I2C_INIT	Identifies the initialization values that are passed as parameters to the initialize and reinitialize routines of the I2C module.
	DRV_I2C_BUFFER_EVENT_HANDLER	Points to a callback after completion of an I2C transfer.
	DRV_I2C_MODE	Lists the operation mode of I2C module.
	DRV_I2C_SLAVE_ADDRESS_MASK	Data type for a slave device address bit-mask.
	I2C_DATA_TYPE	This is type I2C_DATA_TYPE.
	DRV_I2C_PERPH	Lists the operation mode of I2C module.
	I2C_SLAVE_ADDRESS_7bit	Data type for a slave device address.
	DRV_I2C_CallBack	Points to a callback to initiate a particular function.
	I2C_SLAVE_ADDRESS_VALUE	This is type I2C_SLAVE_ADDRESS_VALUE.
	DRV_I2C_INDEX_0	Identifies the instance of the I2C driver.
	DRV_I2C_INDEX_1	Identifies the instance of the I2C driver.
	DRV_I2C_INDEX_2	Identifies the instance of the I2C driver.
	DRV_I2C_INDEX_3	Identifies the instance of the I2C driver.
	DRV_I2C_INDEX_4	Identifies the instance of the I2C driver.
	DRV_I2C_INDEX_5	Identifies the instance of the I2C driver.
	DRV_I2C_BUFFER_QUEUE_SUPPORT	Specifies if the Buffer Queue support should be enabled.
	DRV_I2C_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_I2C_INTERRUPT_MODE	Macro controls interrupt based operation of the driver
	DRV_I2C_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.
	DRV_I2C_BB_H	This is macro DRV_I2C_BB_H.
	DRV_I2C_BUS_ERROR_EVENT	This is type DRV_I2C_BUS_ERROR_EVENT.

Description

This section describes the Application Programming Interface (API) functions of the I2C Driver Library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_I2C_Deinitialize Function

Deinitializes the index instance of the I2C module.

Implementation: Static/Dynamic

File

drv_i2c.h

C

```
void DRV_I2C_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the index instance of the I2C module, disabling its operation (and any hardware for driver modules). It deinitializes only the specified module instance. It also resets all the internal data structures and fields for the specified instance to the default settings.

Remarks

If the module instance has to be used again, DRV_I2C_Initialize should be called again to initialize the module instance structures.

This function may block if the driver is running in an OS environment that supports blocking operations and the driver requires system resources access. However, the routine will NEVER block for hardware I2C access. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_I2C_Status operation. The driver client must always use DRV_I2C_Status to find out when the module is in the ready state.

Preconditions

The DRV_I2C_Initialize function should have been called before calling this function.

Example

```
SYS_STATUS   i2c_status;

DRV_I2C_Deinitialize(I2C_ID_1);

i2c_status = DRV_I2C_Status(I2C_ID_1);
if (SYS_STATUS_BUSY == i2c_status)
{
    // Do something else and check back later
}
else if (SYS_STATUS_ERROR >= i2c_status)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Index, identifying the instance of the I2C module to be deinitialized

Function

void DRV_I2C_Deinitialize (SYS_MODULE_OBJ object)

DRV_I2C_Initialize Function

Initializes hardware and data for the index instance of the I2C module.

Implementation: Static/Dynamic

File

drv_i2c.h

C

```
SYS_MODULE_OBJ DRV_I2C_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

This function initializes hardware for the index instance of the I2C module, using the hardware initialization given data. It also initializes any internal driver data structures making the driver ready to be opened.

Remarks

This function must be called before any other I2C function is called.

This function should only be called once during system initialization unless DRV_I2C_Deinitialize is first called to deinitialize the device instance before reinitializing it.

This function may block if the driver is running in an OS environment that supports blocking operations and the driver requires system resources access. However, the routine will NEVER block for hardware I2C access. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_I2C_Status operation. The driver client must always use DRV_I2C_Status to find out when the module is in the ready state.

Whenever a call to DRV_I2C_Initialize is made with a SYS_MODULE_INIT* data == 0 the following default configuration will be used. Adjust this configuration at build time as needed.

Preconditions

None.

Example

```
DRV_I2C_INIT        i2c_init_data;
SYS_MODULE_OBJ      objectHandle;

i2c_init_data.i2cId = DRV_I2C_PERIPHERAL_ID_IDX0,
i2c_init_data.i2cMode = DRV_I2C_MODE_MASTER,

OR
```

```
i2c_init_data.i2cMode = DRV_I2C_MODE_SLAVE,
//Master mode parameters
i2c_init_data.baudRate = 100000,
i2c_init_data.busspeed = DRV_I2C_SLEW_RATE_CONTROL_IDX0,
i2c_init_data.buslevel = DRV_I2C_SMBus_SPECIFICATION_IDX0,

//Slave mode parameters
i2c_init_data.addWidth = DRV_I2C_7BIT_SLAVE,
i2c_init_data.reservedaddenable  = false,
i2c_init_data.generalcalladdress = false,
i2c_init_data.slaveaddvalue = 0x0060,

//interrupt sources
i2c_init_data.mstrInterruptSource = INT_SOURCE_I2C_2_MASTER,
i2c_init_data.slaveInterruptSource = INT_SOURCE_I2C_2_ERROR,
i2c_init_data.errInterruptSource = INT_SOURCE_I2C_2_ERROR,
i2c_init_data.queueSize = 1,

//callback for Master (Master mode can use callbacks if needed)
i2c_init_data.operationStarting = NULL,
// Slave mode callbacks needed
i2c_init_data.operationStarting = APP_I2CSlaveFunction(),
i2c_init_data.operationEnded = NULL

objectHandle = DRV_I2C_Initialize(DRV_I2C_INDEX_0, (SYS_MODULE_INIT *)&drvI2C0InitData)
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Index, identifying the instance of the I2C module to be initialized
data	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and the default initialization is to be used.

Function

void DRV_I2C_Initialize (const I2C_MODULE_ID index,
const SYS_MODULE_INIT *const data)

DRV_I2C_Tasks Function

Maintains the State Machine of the I2C driver and performs all the protocol level actions.

Implementation: Dynamic

File

drv_i2c.h

C

```
void DRV_I2C_Tasks(SYS_MODULE_OBJ object);
```

Description

This functions maintains the internal state machine of the I2C driver. This function acts as the I2C Master or Slave ISR. When used in polling mode, this function needs to be called repeatedly to achieve I2C data transfer. This function implements all the protocol level details like setting the START condition, sending the address with with R/W request, writing data to the SFR, checking for acknowledge and setting the STOP condition.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance.

Example

SYS_MODULE_OBJ object;
while (true) { DRV_I2C_Tasks (object);

Function

void DRV_I2C_Tasks (SYS_MODULE_OBJ object)

b) Client Setup Functions

DRV_I2C_Close Function

Closes an opened instance of an I2C module driver.

Implementation: Dynamic

File

drv_i2c.h

C

```
void DRV_I2C_Close(DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened instance of an I2C module driver, making the specified handle invalid.

Remarks

After calling This function, the handle passed into drvHandle must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_I2C_Open before the caller may use the driver again.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance and the DRV_I2C_Status must have returned SYS_STATUS_READY.

DRV_I2C_Open must have been called to obtain a valid opened device handle.

Example

```
myI2CHandle = DRV_I2C_Open(I2C_ID_1, DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE);

// Perform data transfer operations

DRV_I2C_Close(myI2CHandle);
```

Parameters

Parameters	Description
drvHandle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_I2C_Close (const DRV_HANDLE drvHandle)

DRV_I2C_Open Function

Opens the specified instance of the I2C driver for use and provides an "open-instance" handle.

Implementation: Dynamic

File

drv_i2c.h

C

```
DRV_HANDLE DRV_I2C_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a value identifying both the caller and the module instance). If an error occurs, the returned value is DRV_HANDLE_INVALID.

If an error occurs, the return value is DRV_HANDLE_INVALID. An error can occur when the following is true:

- if the number of client objects allocated via DRV_I2C_INSTANCES_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid

Description

This function opens the specified instance of the I2C module for use and provides a handle that is required to use the remaining driver routines.

This function opens a specified instance of the I2C module driver for use by any client module and provides an "open-instance" handle that must be provided to any of the other I2C driver operations to identify the caller and the instance of the I2C driver/hardware module.

Remarks

The handle returned is valid until the DRV_I2C_Close routine is called.

This function may block if the driver is running in an OS environment that supports blocking operations and the driver requires system resources access. Regarding the hardware I2C access the operation will behave as instructed by the DRV_IO_INTENT parameter.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance and the DRV_I2C_Status must have returned SYS_STATUS_READY.

Example

```
DRV_HANDLE              i2c_handle;

i2c_handle = DRV_I2C_Open(I2C_ID_1, DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE);
if (DRV_HANDLE_INVALID == i2c_handle)
{
    // Handle open error
}

// Close the device when it is no longer needed.
DRV_I2C_Close(i2c_handle);
```

Parameters

Parameters	Description
index	Index, identifying the instance of the I2C module to be opened.
intent	Flags parameter identifying the intended usage and behavior of the driver. Multiple flags may be ORed together to specify the intended usage of the device. See the DRV_IO_INTENT definition.

Function

DRV_HANDLE DRV_I2C_Open (const I2C_MODULE_ID index,
const DRV_IO_INTENT intent)

c) Data Transfer Functions

DRV_I2C_BufferEventHandlerSet Function

Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

Implementation: Dynamic

File

drv_i2c.h

C

```
void DRV_I2C_BufferEventHandlerSet(const DRV_HANDLE handle, const DRV_I2C_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When

a client calls either the DRV_I2C_BufferAddRead, DRV_I2C_BufferAddWrite or DRV_I2C_BufferAddWriteRead function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

When in Master mode, a callback event is registered when to let the application know that the buffer has been transmitted. DRV_I2C_BUFFER_EVENT_COMPLETE is set when the buffer has been transmitted without any errors. DRV_I2C_BUFFER_EVENT_ERROR is set when buffer transmission or reception has been aborted.

When in Slave mode, since the Master controls when a transmit or receive operation is terminated, a callback is registered every time a byte is written or read from the slave.

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C driver instance.

DRV_I2C_Open must have been called to obtain a valid opened device handle.

Example

```
#define MY_BUFFER_SIZE  10

// function prototype of Event Handler Function
void APP_I2CBufferEventFunction ( DRV_I2C_BUFFER_EVENT event,
                                  DRV_I2C_BUFFER_HANDLE bufferHandle,
                                  uintptr_t context );

//Returned from DRV_I2C_Open
DRV_HANDLE drvI2Chandle;

// myAppObj is an application specific state data object.
DRV_I2C_BUFFER_EVENT operationStatus;

uint8_t appBuffer[MY_BUFFER_SIZE];

DRV_I2C_BUFFER_HANDLE drvI2CRDBUFHandle

// Opens an instance of I2C driver
drvI2Chandle = DRV_I2C_Open( DRV_I2C_INDEX_0,DRV_IO_INTENT_WRITE );

// Client registers an event handler with driver. This is done once.
DRV_I2C_BufferEventHandlerSet( drvI2Chandle,
                               APP_I2CBufferEventFunction,
                               operationStatus );

drvI2CRDBUFHandle = DRV_I2C_Receive (   drvI2CHandle,
                                        slaveaddress
                                        &appBuffer[],
                                        MY_BUFFER_SIZE,
                                        NULL );

if(NULL == drvI2CRDBUFHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2CBufferEventFunction( DRV_I2C_BUFFER_EVENT event,
                                 DRV_I2C_BUFFER_HANDLE handle,
                                 uintptr_t context)
{

    switch(event)
    {
        case DRV_I2C_BUFFER_EVENT_COMPLETE:
```

```
            //perform appropriate action
            break;

            case DRV_I2C_BUFFER_EVENT_ERROR:
                // Error handling here.
            break;

            default:
                break;
        }
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_I2C_BufferEventHandlerSet (
const DRV_HANDLE handle,
const DRV_I2C_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t context)

DRV_I2C_BytesTransferred Function

Returns the number of bytes transmitted or received in a particular I2C transaction. The transaction is identified by the handle.

File

drv_i2c.h

C

```
uint32_t DRV_I2C_BytesTransferred(DRV_HANDLE handle, DRV_I2C_BUFFER_HANDLE bufferHandle);
```

Returns

The number of bytes transferred in a particular I2C transaction.
`numOfBytes = DRV_I2C_BytesTransferred (drvI2CHandle_Master,drvBufferHandle);`

Description

This returns the transmitter and receiver transfer status.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
bufferHandle	A valid buffer handle obtained when calling Transmit/Receive/TransmitThenReceive/TransmitForced or BufferAddRead/BufferAddWrite/BufferAddReadWrite function

Function

uint32_t DRV_I2C_BytesTransferred (DRV_I2C_BUFFER_HANDLE bufferHandle)

DRV_I2C_Receive Function

This function reads data written from either Master or Slave.

Implementation: Dynamic

File

drv_i2c.h

C

```
DRV_I2C_BUFFER_HANDLE DRV_I2C_Receive(DRV_HANDLE handle, uint16_t address, void * buffer, size_t size, void
* callbackContext);
```

Returns

A valid BUFFER HANDLE, NULL if the handle is not obtained.

Description

Master calls this function to read data send by Slave. The Slave calls this function to read data send by Master. In case of Master, a START condition is initiated on the I2C bus.

Remarks

The handle that is passed into the function, drvI2CHandle is obtained by calling the DRV_I2C_OPEN function. If the function could not return a valid buffer handle, then a NULL value is returned. If the slave NACKs the address byte, then further read is not attempted. Master asserts STOP condition and DRV_I2C_BUFFER_EVENT_ERROR is set as the buffer-status. If all the requisite number of bytes have been read then DRV_I2C_BUFFER_EVENT_COMPLETE is set as the buffer status.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance and the DRV_I2C_Status must have returned SYS_STATUS_READY.

DRV_I2C_Open must have been called to obtain a valid opened device handle.

Example

```
drvI2CRDBUFHandle = DRV_I2C_Receive( drvI2CHandle,
                                     deviceaddress,
                                     &rxbuffer[0],
                                     num_of_bytes,
                                     NULL );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
address	Device address of slave. If this API is used in Slave mode, then a dummy value can be used
buffer	This buffer holds data is received
size	The number of bytes that the Master expects to read from Slave. This value can be kept as the MAX BUFFER SIZE for slave. This is because the Master controls when the READ operation is terminated.
callbackContext	Not implemented, future expansion

Function

DRV_I2C_BUFFER_HANDLE DRV_I2C_Receive (DRV_HANDLE handle,

uint16_t slaveaddress,

void *rxBuffer,

size_t size,

void * callbackContext)

DRV_I2C_Transmit Function

This function writes data to Master or Slave.

Implementation: Dynamic

File

drv_i2c.h

C

```
DRV_I2C_BUFFER_HANDLE DRV_I2C_Transmit(DRV_HANDLE handle, uint16_t slaveaddress, void * buffer, size_t
size, void * context);
```

Returns

A valid BUFFER HANDLE, NULL if the handle is not obtained.

Description

Master calls this function to write data to Slave. The Slave calls this function to write data to Master.

Remarks

The handle that is passed into the function, drvI2CHandle is obtained by calling the DRV_I2C_OPEN function. If the function could not return a valid buffer handle, then a NULL value is returned. If the slave NACKs the address byte or any data bytes, then further write is not attempted. Master asserts STOP condition and DRV_I2C_BUFFER_EVENT_ERROR is set as the buffer-status. If all the requisite number of bytes have been transmitted to the Slave, then DRV_I2C_BUFFER_EVENT_COMPLETE is set as the buffer status.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance and the DRV_I2C_Status must have returned SYS_STATUS_READY.

DRV_I2C_Open must have been called to obtain a valid opened device handle.

Example

```
drvI2CWRBUFHandle = DRV_I2C_Transmit( drvI2CHandle,
                                      deviceaddress,
                                      &txBuffer[0],
                                      num_of_bytes,
                                      NULL);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
address	Device address of slave. If this API is used in Slave mode, then a dummy value can be used
buffer	Contains data to be transferred
size	The number of bytes that the Master expects to write to Slave. This value can be kept as the MAX BUFFER SIZE for slave. This is because the Master controls when the WRITE operation is terminated.
callbackContext	Not implemented, future expansion

Function

DRV_I2C_BUFFER_HANDLE DRV_I2C_Transmit(DRV_HANDLE handle,
uint16_t slaveaddress,
void *txBuffer,
size_t size,
void *context);

DRV_I2C_TransmitThenReceive Function

This function writes data to Slave, inserts restart and requests read from slave.

Implementation: Dynamic

File

drv_i2c.h

C

```
DRV_I2C_BUFFER_HANDLE DRV_I2C_TransmitThenReceive(DRV_HANDLE handle, uint16_t address, void * writeBuffer,
size_t writeSize, void * readBuffer, size_t readSize, void * callbackContext);
```

Returns

A valid BUFFER HANDLE, NULL if the handle is not obtained.

Description

Master calls this function to send a register address value to the slave and then queries the slave with a read request to read the contents indexed by the register location. The Master sends a restart condition after the initial write before sending the device address with R/W = 1. The restart condition prevents the Master from relinquishing the control of the bus. The slave should not use this function.

Remarks

The handle that is passed into the function, drvI2CHandle is obtained by calling the DRV_I2C_OPEN function. If the function could not return a valid buffer handle, then a NULL value is returned. If there is any error condition during transmission then further transmission or reception is not

attempted and STOP condition is asserted on the bus. In case of error condition, DRV_I2C_BUFFER_EVENT_ERROR is set as the buffer-status. If the I2C bus transaction is completed as requested then the buffer status, is set as DRV_I2C_BUFFER_EVENT_COMPLETE.

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance and the DRV_I2C_Status must have returned SYS_STATUS_READY.

DRV_I2C_Open must have been called to obtain a valid opened device handle.

Example

```
drvI2CRDBUFHandle = DRV_I2C_TransmiThenReceive( appData.drvI2CHandle,
                                                deviceaddress,
                                                &drvI2CTXbuffer[0],
                                                registerbytesize,
                                                rxbuffer,
                                                num_of_bytes,
                                                NULL );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
address	Device address of slave. If this API is used in Slave mode, then a dummy value can be used
writeBuffer	Contains data to be transferred
writeSize	The number of bytes that the Master expects to write to Slave. This value can be kept as the MAX BUFFER SIZE for slave. This is because the Master controls when the WRITE operation is terminated.
readBuffer	This buffer holds data that is send back from slave after read operation.
readSize	The number of bytes the Master expects to be read from the slave
callbackContext	Not implemented, future expansion

Function

DRV_I2C_BUFFER_HANDLE DRV_I2C_TransmiThenReceive (DRV_HANDLE handle,
uint16_t deviceaddress,
void *txBuffer,
size_t writeSize,
void *rxBuffer,
size_t readSize,
void *context)

DRV_I2C_TransmitForced Function

This function writes data to Master or Slave.

Implementation: Dynamic

File

drv_i2c.h

C

```
DRV_I2C_BUFFER_HANDLE DRV_I2C_TransmitForced(DRV_HANDLE handle, uint16_t deviceaddress, void* txBuffer,
size_t txbuflen, DRV_I2C_BUS_ERROR_EVENT eventFlag, void * calbackContext);
```

Returns

A valid BUFFER HANDLE, NULL if the handle is not obtained.

Description

Master calls this function to transmit the entire buffer to the slave even if the slave ACKs or NACKs the address or any of the data bytes. This is typically used for slaves that have to initiate a reset sequence by sending a dummy I2C transaction. Since the slave is still in reset, any or all the bytes can be NACKed. In the normal operation of the driver if the address or data byte is NACKed, then the transmission is aborted and a STOP condition is asserted on the bus.

Remarks

The handle that is passed into the function, drvI2CHandle is obtained by calling the DRV_I2C_OPEN function. If the function could not return a valid buffer handle, then a NULl value is returned. Once all the bytes are transferred the buffer status is set as then

DRV_I2C_BUFFER_EVENT_COMPLETE .

Preconditions

The DRV_I2C_Initialize routine must have been called for the specified I2C device instance and the DRV_I2C_Status must have returned SYS_STATUS_READY.

DRV_I2C_Open must have been called to obtain a valid opened device handle.

Example

```
drvI2CWRBUFHandle = DRV_I2C_TransmitForced ( handle,
                                             deviceaddress,
                                             &txBuffer[0],
                                             txbuflen,
                                             NULL,
                                             NULL)
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
address	Device address of slave. If this API is used in Slave mode, then a dummy value can be used
buffer	Contains data to be transferred
size	The number of bytes that the Master expects to write to Slave. This value can be kept as the MAX BUFFER SIZE for slave. This is because the Master controls when the WRITE operation is terminated.
eventFlag	This field is left for future implementation
callbackContext	Not implemented, future expansion

Function

DRV_I2C_BUFFER_HANDLE DRV_I2C_TransmitForced (DRV_HANDLE handle,

uint16_t deviceaddress,

uint8_t* txBuffer,

uint16_t txbuflen,

DRV_I2C_BUS_ERROR_EVENT eventFlag,

void * calbackContext)

d) Status Functions

DRV_I2C_TransferStatusGet Function

Returns status of data transfer when Master or Slave acts either as a transmitter or a receiver.

Implementation: Dynamic

File

drv_i2c.h

C

```
DRV_I2C_BUFFER_EVENT DRV_I2C_TransferStatusGet(DRV_HANDLE handle, DRV_I2C_BUFFER_HANDLE bufferHandle);
```

Returns

A DRV_I2C_TRANSFER_STATUS value describing the current status of the transfer.

Description

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event. If the event is DRV_I2C_BUFFER_EVENT_COMPLETE, it means that the data was transferred successfully. If the event is DRV_I2C_BUFFER_EVENT_ERROR, it means that the data was not transferred successfully.

Remarks

The handle that is passed into the function, drvI2CBUFHandle is obtained by calling one of the data transfer functions. The drvI2CBUFHandle should be a valid handle and not a NULL value. The DRV_I2C_BufferStatus can be called to check the progress of the data transfer operation. If the buffer is transferred without any error, then DRV_I2C_BUFFER_EVENT_COMPLETE is returned. If an error condition is present, then DRV_I2C_BUFFER_EVENT_ERROR is returned.

Example

```
if(DRV_I2C_BUFFER_EVENT_COMPLETE == DRV_I2C_TransferStatusGet ( handle,
                                                                bufferHandle ))
{
        //perform action
        return true;
}
else
{
        //perform action
        return false;
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
bufferHandle	A valid buffer handle obtained when calling Transmit/Receive/TransmitThenReceive/TransmitForced or BufferAddRead/BufferAddWrite/BufferAddReadWrite function

Function

DRV_I2C_BUFFER_EVENT DRV_I2C_TransferStatusGet (DRV_HANDLE handle,
DRV_I2C_BUFFER_HANDLE bufferHandle)

DRV_I2C_Status Function

Provides the current status of the index instance of the I2C module.

Implementation: Dynamic

File

drv_i2c.h

C

```
SYS_STATUS DRV_I2C_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that any previous module operation for the specified I2Cmodule has completed.
- SYS_STATUS_BUSY - Indicates that a previous module operation for the specified I2C module has not yet completed
- SYS_STATUS_ERROR - Indicates that the specified I2C module is in an error state

Description

This function provides the current status of the index instance of the I2C module.

Remarks

The DRV_I2C_Status operation can be used to determine when any of the I2C module level operations has completed. The value returned by the DRV_I2C_Status routine hast to be checked after calling any of the I2C module operations to find out when they have completed.

If the DRV_I2C_Status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the DRV_I2C_Status operation return SYS_STATUS_READY, any previous operations have completed.

The DRV_I2C_Status function will NEVER block.

If the DRV_I2C_Status operation returns an error value, the error may be cleared by calling the DRV_I2C_Initialize operation. If that fails, the DRV_I2C_Deinitialize operation will need to be called, followed by the DRV_I2C_Initialize operation to return to normal operations.

Preconditions

The DRV_I2C_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;
SYS_STATUS          i2c_status;

i2c_status = DRV_I2C_Status(object);
if (SYS_STATUS_BUSY == i2c_status)
{
```

```
    // Do something else and check back later
}
else if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Index, identifying the instance of the I2C module to get status for.

Function

SYS_STATUS DRV_I2C_Status (SYS_MODULE_OBJ object)

e) Miscellaneous Functions

DRV_I2C_QueueFlush Function

The existing transactions in the queue are voided and the queue pointers are reset to their initial state. This renders the queue empty.

File

drv_i2c.h

C

```
void DRV_I2C_QueueFlush(DRV_HANDLE handle);
```

Returns

None

```
    //Opens an instance of I2C driver
 drvI2Chandle = DRV_I2C_Open( DRV_I2C_INDEX_0,DRV_IO_INTENT_WRITE );

 DRV_I2C_QueueFlush ( drvI2CHandle );
```

Description

The existing transactions in the queue are voided and the queue pointers are reset to their initial state. This renders the queue empty.

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_I2C_QueueFlush (DRV_HANDLE handle)

f) Data Types and Constants

DRV_I2C_ADDRESS_WIDTH Enumeration

Lists the Address Width of the Slave.

File

drv_i2c.h

C

```
typedef enum {
  DRV_I2C_7BIT_SLAVE,
  DRV_I2C_10BIT_SLAVE
} DRV_I2C_ADDRESS_WIDTH;
```

Members

Members	Description
DRV_I2C_7BIT_SLAVE	I2C Slave 7 bit
DRV_I2C_10BIT_SLAVE	I2C Slave 10 bit

Description

I2C Slave Address Width

This enumeration lists if the I2C module is configured as a 7-bit Slave or a 10-bit Slave.

Remarks

None.

DRV_I2C_BUFFER_HANDLE Type

Handle to an I2C data buffer.

File

drv_i2c.h

C

```
typedef uintptr_t DRV_I2C_BUFFER_HANDLE;
```

Description

I2C Buffer Handle

This data type provides a handle to an I2C buffer that has been passed to the driver for processing.

Remarks

This handle is provided by one of the data transfer functions (DRV_I2C_Receive, DRV_I2C_Transmit, DRV_I2C_TransmitThenReceive) and is used by the buffer event-handler (DRV_I2C_BUFFER_EVENT_HANDLER) call-back function and buffer status functions (DRV_I2C_TransferStatusGet, DRV_I2C_BytesTransferred).

DRV_I2C_BUS_LEVEL Enumeration

Lists the Operational Voltage level of I2C.

File

drv_i2c.h

C

```
typedef enum {
  DRV_I2C_OPEN_COLLECTOR_LEVEL,
  DRV_I2C_SMBus_LEVEL
} DRV_I2C_BUS_LEVEL;
```

Members

Members	Description
DRV_I2C_OPEN_COLLECTOR_LEVEL	I2C BUS LEVEL
DRV_I2C_SMBus_LEVEL	SMBus level

Description

I2C Bus Levels

This enumeration lists if the I2C is configured to operate in the traditional I2C mode or the SMBus mode.

Remarks

None.

DRV_I2C_BUS_SPEED Enumeration

Lists the I2C Bus speed mode.

File

drv_i2c.h

C

```
typedef enum {
  DRV_I2C_NORMAL_SPEED,
  DRV_I2C_HIGH_SPEED
} DRV_I2C_BUS_SPEED;
```

Description

I2C Bus Speed

This enumeration lists if the I2C is configured to operate at High-Speed or Normal Speed.

Remarks

None.

DRV_I2C_BUFFER_EVENT Enumeration

Lists the different conditions that happens during a buffer transfer.

File

drv_i2c.h

C

```
typedef enum {
  DRV_I2C_BUFFER_EVENT_PENDING,
  DRV_I2C_BUFFER_EVENT_COMPLETE,
  DRV_I2C_BUFFER_EVENT_ERROR,
  DRV_I2C_SEND_STOP_EVENT,
  DRV_I2C_SEND_RESTART_EVENT,
  DRV_I2C_BUFFER_SLAVE_READ_REQUESTED,
  DRV_I2C_BUFFER_SLAVE_WRITE_REQUESTED,
  DRV_I2C_BUFFER_SLAVE_READ_BYTE,
  DRV_I2C_BUFFER_MASTER_ACK_SEND,
  DRV_I2C_BUFFER_MASTER_NACK_SEND,
  DRV_I2C_BUFFER_SLAVE_WRITE_BYTE
} DRV_I2C_BUFFER_EVENT;
```

Members

Members	Description
DRV_I2C_BUFFER_EVENT_PENDING	Buffer is pending to get processed
DRV_I2C_BUFFER_EVENT_COMPLETE	All data from or to the buffer was transferred successfully.
DRV_I2C_BUFFER_EVENT_ERROR	There was an error while processing the buffer transfer request.
DRV_I2C_SEND_STOP_EVENT	Send Stop by Master
DRV_I2C_SEND_RESTART_EVENT	Send Restart Event by Master
DRV_I2C_BUFFER_SLAVE_READ_REQUESTED	Master sends data to slave
DRV_I2C_BUFFER_SLAVE_WRITE_REQUESTED	Master requests data from slave
DRV_I2C_BUFFER_SLAVE_READ_BYTE	Slave read byte send by Master
DRV_I2C_BUFFER_MASTER_ACK_SEND	sending ACK to Slave for more bytes
DRV_I2C_BUFFER_MASTER_NACK_SEND	sending ACK to Slave for more bytes
DRV_I2C_BUFFER_SLAVE_WRITE_BYTE	Slave send byte to Master

Description

I2C Buffer Events

This enumeration identifies the different conditions that can happen during a buffer transaction. Callbacks can be made with the appropriate buffer condition passed as a parameter to execute the desired action. The application can also poll the BufferStatus flag to check the status of transfer.

The values act like flags and multiple flags can be set.

Remarks

None.

DRV_I2C_INIT Structure

Identifies the initialization values that are passed as parameters to the initialize and reinitialize routines of the I2C module.

File

drv_i2c.h

C

```
typedef struct _DRV_I2C_INIT {
  SYS_MODULE_INIT moduleInit;
  I2C_MODULE_ID i2cId;
  DRV_I2C_MODE i2cMode;
  DRV_I2C_PERPH i2cPerph;
  PORTS_CHANNEL portSCL;
  PORTS_BIT_POS pinSCL;
  PORTS_CHANNEL portSDA;
  PORTS_BIT_POS pinSDA;
  DRV_I2C_ADDRESS_WIDTH addWidth;
  bool reservedaddenable;
  bool generalcalladdress;
  I2C_SLAVE_ADDRESS_VALUE slaveaddvalue;
  uint32_t baudRate;
  DRV_I2C_BUS_LEVEL buslevel;
  DRV_I2C_BUS_SPEED busspeed;
  INT_SOURCE mstrInterruptSource;
  INT_SOURCE slaveInterruptSource;
  INT_SOURCE errInterruptSource;
  unsigned int queueSize;
  DRV_I2C_CallBack operationStarting;
  DRV_I2C_SLAVE_ADDRESS_MASK maskslaveaddress;
} DRV_I2C_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
I2C_MODULE_ID i2cId;	Identifies peripheral (PLIB-level) ID
DRV_I2C_MODE i2cMode;	I2C Usage Mode Type
DRV_I2C_PERPH i2cPerph;	I2C Peripheral or not usage
PORTS_CHANNEL portSCL;	PORT which SCL belongs
PORTS_BIT_POS pinSCL;	Bit position in the port for SCL
PORTS_CHANNEL portSDA;	PORT which SDA belongs
PORTS_BIT_POS pinSDA;	Bit position in the port for SDA
DRV_I2C_ADDRESS_WIDTH addWidth;	Communication Width
bool reservedaddenable;	Reserved Address rule enable
bool generalcalladdress;	General Call address enable
I2C_SLAVE_ADDRESS_VALUE slaveaddvalue;	Slave address value
uint32_t baudRate;	Baud Rate Value
DRV_I2C_BUS_LEVEL buslevel;	I2C Clock mode
DRV_I2C_BUS_SPEED busspeed;	I2C Bus Speed
INT_SOURCE mstrInterruptSource;	Master Interrupt Source for I2C module
INT_SOURCE slaveInterruptSource;	Slave Interrupt Source for I2C module
INT_SOURCE errInterruptSource;	Error Interrupt Source for I2C module
unsigned int queueSize;	This is the buffer queue size. This is the maximum number of transfer requests that driver will queue. For a static build of the driver, this is overridden by the DRV_I2C_QUEUE_SIZE macro in system_config.h
DRV_I2C_CallBack operationStarting;	This callback is fired when an operation needs to be initiated on the I2C bus. This callback is mainly intended when the driver is used in SLAVE mode and is required to send data to the Master. The callback signals the application to prepare data for transfer to Master. This callback may be called from an ISR so should not include OSAL calls. The context parameter is the same one passed into the BufferAddRead, BufferAddWrite, BufferAddWriteRead function.
DRV_I2C_SLAVE_ADDRESS_MASK maskslaveaddress;	Slave address Mask value, the I2C slave address match ignored for the bit fields that the mask is set

Description

I2C Initialization Values

This enumeration identifies the values that are passed as parameters to the initialize and reinitialize routines of the I2C module.

These values specify different I2C module initialization parameters.

DRV_I2C_BUFFER_EVENT_HANDLER Type

Points to a callback after completion of an I2C transfer.

File

drv_i2c.h

C

```
typedef void (* DRV_I2C_BUFFER_EVENT_HANDLER)(DRV_I2C_BUFFER_EVENT event, DRV_I2C_BUFFER_HANDLE
bufferHandle, uintptr_t context);
```

Description

I2C Buffer Event Callback

This type identifies the I2C Buffer Event. It allows the client driver to register a callback using DRV_I2C_BUFFER_EVENT_HANDLER. By using this mechanism, the driver client will be notified at the completion of the corresponding transfer.

Remarks

A transfer can be composed of various transfer segments. Once a transfer is completed the driver will call the client registered transfer callback.

The callback could be called from ISR context and should be kept as short as possible. It is meant for signaling and it should not be blocking.

Parameters

Parameters	Description
DRV_I2C_BUFFER_EVENT	Status of I2C transfer
bufferHandle	Handle that identifies that identifies the particular Buffer Object
context	For future implementation

Function

void (*DRV_I2C_BUFFER_EVENT_HANDLER) (DRV_I2C_BUFFER_EVENT event,
DRV_I2C_BUFFER_HANDLE bufferHandle, uintptr_t context)

DRV_I2C_MODE Enumeration

Lists the operation mode of I2C module.

File

drv_i2c.h

C

```
typedef enum {
  DRV_I2C_MODE_MASTER,
  DRV_I2C_MODE_SLAVE
} DRV_I2C_MODE;
```

Members

Members	Description
DRV_I2C_MODE_MASTER	I2C Mode Master
DRV_I2C_MODE_SLAVE	I2C Mode Slave

Description

I2C Operation Mode

This enumeration lists if the I2C module is configured as a Master or a Slave.

Remarks

None.

DRV_I2C_SLAVE_ADDRESS_MASK Type

Data type for a slave device address bit-mask.

File

drv_i2c.h

C

```
typedef uint8_t DRV_I2C_SLAVE_ADDRESS_MASK;
```

Description

I2C Slave Device Address Mask

This is the data type for an I2C slave device address bit-mask. It can hold both 7-bit and 10-bit slave addresses masks.

Remarks

A slave address mask is used in slave mode to identify "don't care" bits in the base slave address to which the I2C interface will respond..

I2C_DATA_TYPE Type

File

drv_i2c.h

C

```
typedef unsigned char I2C_DATA_TYPE;
```

Description

This is type I2C_DATA_TYPE.

DRV_I2C_PERPH Enumeration

Lists the operation mode of I2C module.

File

drv_i2c.h

C

```
typedef enum {
  DRV_I2C_PERIPHERAL,
  DRV_I2C_BIT_BANG
} DRV_I2C_PERPH;
```

Members

Members	Description
DRV_I2C_PERIPHERAL	Make use of the I2C Peripheral.
DRV_I2C_BIT_BANG	Make use of the I2C Bit Bang Algorithm

Description

I2C Configuration Choice

This enumeration lists if the I2C module is configured to use I2C peripheral or use Bit-Bang implementation.

Remarks

None.

I2C_SLAVE_ADDRESS_7bit Type

Data type for a slave device address.

File

drv_i2c.h

C

```
typedef uint8_t I2C_SLAVE_ADDRESS_7bit;
```

Description

need to type cast 10-bit slave address into 8 bit mode

****

I2C Slave Device Address

This is the data type for an I2C slave device address. It can hold both 7-bit and 10-bit slave addresses.

Remarks

None.

DRV_I2C_CallBack Type

Points to a callback to initiate a particular function.

File

drv_i2c.h

C

```
typedef void (* DRV_I2C_CallBack)(DRV_I2C_BUFFER_EVENT event, void * context);
```

Description

I2C Buffer Event Callback

This type identifies the I2C Buffer Event. It allows the client driver to register a callback using DRV_I2C_BUFFER_EVENT. By using this mechanism, the driver client will can initiate an operation. This is intended to be used for SLAVE mode because the Master drives the I2C bus and through this the Slave can issue a READ or WRITE based on the status of R/W bit received from the Master

Remarks

A transfer can be composed of various transfer segments. Once a transfer is completed the driver will call the client registered transfer callback.

The callback could be called from ISR context and should be kept as short as possible. It is meant for signaling and it should not be blocking.

Parameters

Parameters	Description
DRV_I2C_BUFFER_EVENT	Status of I2C transfer
context	This is left for future implementation

Function

typedef void (*DRV_I2C_CallBack) (DRV_I2C_BUFFER_EVENT event,

void * context)

I2C_SLAVE_ADDRESS_VALUE Type

File

drv_i2c.h

C

```
typedef uint16_t I2C_SLAVE_ADDRESS_VALUE;
```

Description

This is type I2C_SLAVE_ADDRESS_VALUE.

DRV_I2C_INDEX_0 Macro

Identifies the instance of the I2C driver.

File

drv_i2c.h

C

```
#define DRV_I2C_INDEX_0 0
```

Description

Driver Instance Index 0

This value identifies the instance of the I2C driver. It is a zero-based index.

Remarks

This number is not directly related to the peripheral hardware instance. The peripehral hardware instances is associated with the driver instance by the DRV_I2C_Initialize function.

DRV_I2C_INDEX_1 Macro

Identifies the instance of the I2C driver.

File

drv_i2c.h

C

```
#define DRV_I2C_INDEX_1 1
```

Description

Driver Instance Index 1

This value identifies the instance of the I2C driver. It is a zero-based index.

Remarks

This number is not directly related to the peripheral hardware instance. The peripheral hardware instances is associated with the driver instance by the DRV_I2C_Initialize function.

DRV_I2C_INDEX_2 Macro

Identifies the instance of the I2C driver.

File

drv_i2c.h

C

```
#define DRV_I2C_INDEX_2 2
```

Description

Driver Instance Index 2

This value identifies the instance of the I2C driver. It is a zero-based index.

Remarks

This number is not directly related to the peripheral hardware instance. The peripheral hardware instances is associated with the driver instance by the DRV_I2C_Initialize function.

DRV_I2C_INDEX_3 Macro

Identifies the instance of the I2C driver.

File

drv_i2c.h

C

```
#define DRV_I2C_INDEX_3 3
```

Description

Driver Instance Index 3

This value identifies the instance of the I2C driver. It is a zero-based index.

Remarks

This number is not directly related to the peripheral hardware instance. The peripheral hardware instances is associated with the driver instance by the DRV_I2C_Initialize function.

DRV_I2C_INDEX_4 Macro

Identifies the instance of the I2C driver.

File

drv_i2c.h

C

```
#define DRV_I2C_INDEX_4 4
```

Description

Driver Instance Index 4

This value identifies the instance of the I2C driver. It is a zero-based index.

Remarks

This number is not directly related to the peripheral hardware instance. The peripheral hardware instances is associated with the driver instance by the DRV_I2C_Initialize function.

DRV_I2C_INDEX_5 Macro

Identifies the instance of the I2C driver.

File

drv_i2c.h

C

```
#define DRV_I2C_INDEX_5 5
```

Description

Driver Instance Index 5

This value identifies the instance of the I2C driver. It is a zero-based index.

Remarks

This number is not directly related to the peripheral hardware instance. The peripehral hardware instances is associated with the driver instance by the DRV_I2C_Initialize function.

DRV_I2C_BUFFER_QUEUE_SUPPORT Macro

Specifies if the Buffer Queue support should be enabled.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_BUFFER_QUEUE_SUPPORT false
```

Description

I2C Driver Buffer Queue Support

This macro defines if Buffer Queue support should be enabled. Setting this macro to true will enable buffer queue support and all buffer related driver function.

Remarks

None

DRV_I2C_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_INSTANCES_NUMBER 5
```

Description

I2C driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of I2C modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None

DRV_I2C_INTERRUPT_MODE Macro

Macro controls interrupt based operation of the driver

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_INTERRUPT_MODE true
```

Description

I2C Interrupt Mode Operation Control

This macro controls the interrupt based operation of the driver. The possible values it can take are

- true - Enables the interrupt mode
- false - Enables the polling mode

If the macro value is true, then Interrupt Service Routine for the interrupt should be defined in the application. The DRV_I2C_Tasks() routine should be called in the ISR.

Remarks

None

DRV_I2C_QUEUE_DEPTH_COMBINED Macro

Number of entries of all queues in all instances of the driver.

File

drv_i2c_config_template.h

C

```
#define DRV_I2C_QUEUE_DEPTH_COMBINED 7
```

Description

I2C Driver Instance combined queue depth.

This macro defines the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for transmit and receive operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build). The hardware instance transmit buffer queue will queue transmit buffers submitted by the DRV_I2C_BufferAddWrite() function. The hardware instance receive buffer queue will queue receive buffers submitted by the DRV_I2C_BufferAddRead() function.

A buffer queue will contain buffer queue entries, each related to a BufferAdd request. This configuration macro defines total number of buffer entries that will be available for use between all I2C driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking read and write requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. More the number of buffer entries, greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its transmit and receive buffer queue size.

As an example, consider the case of static single client driver application where full duplex non blocking operation is desired without queuing, the

minimum transmit queue depth and minimum receive queue depth should be 1. Hence the total number of buffer entries should be 2.

In the current implementation of I2C driver, queueing of Buffers is not supported. This will be added in a future release.

Remarks

None

DRV_I2C_BB_H Macro

File

drv_i2c_bb.h

C

```
#define DRV_I2C_BB_H
```

Description

This is macro DRV_I2C_BB_H.

DRV_I2C_BUS_ERROR_EVENT Enumeration

File

drv_i2c.h

C

```
typedef enum {
  DRV_I2C_HALT_ON_ERROR = 0x00,
  DRV_I2C_BUS_IGNORE_COLLISION_ERROR = 0x01,
  DRV_I2C_BUS_IGNORE_OVERFLOW_ERROR = 0x02
} DRV_I2C_BUS_ERROR_EVENT;
```

Members

Members	Description
DRV_I2C_HALT_ON_ERROR = 0x00	normal I2C with not ignoring any error
DRV_I2C_BUS_IGNORE_COLLISION_ERROR = 0x01	ignore bus collision error
DRV_I2C_BUS_IGNORE_OVERFLOW_ERROR = 0x02	ignore overflow error

Description

This is type DRV_I2C_BUS_ERROR_EVENT.

Files

Files

Name	Description
drv_i2c.h	I2C module driver interface header.
drv_i2c_bb.h	Contains prototypes for the I2C functions
drv_i2c_config_template.h	I2C device driver configuration file.

Description

drv_i2c.h

I2C module driver interface header.

Enumerations

	Name	Description
	DRV_I2C_ADDRESS_WIDTH	Lists the Address Width of the Slave.
	DRV_I2C_BUFFER_EVENT	Lists the different conditions that happens during a buffer transfer.

	DRV_I2C_BUS_ERROR_EVENT	This is type DRV_I2C_BUS_ERROR_EVENT.
	DRV_I2C_BUS_LEVEL	Lists the Operational Voltage level of I2C.
	DRV_I2C_BUS_SPEED	Lists the I2C Bus speed mode.
	DRV_I2C_MODE	Lists the operation mode of I2C module.
	DRV_I2C_PERPH	Lists the operation mode of I2C module.

Functions

	Name	Description
	DRV_I2C_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Dynamic
	DRV_I2C_BytesTransferred	Returns the number of bytes transmitted or received in a particular I2C transaction. The transaction is identified by the handle.
	DRV_I2C_Close	Closes an opened instance of an I2C module driver. **Implementation:** Dynamic
	DRV_I2C_Deinitialize	Deinitializes the index instance of the I2C module. **Implementation:** Static/Dynamic
	DRV_I2C_Initialize	Initializes hardware and data for the index instance of the I2C module. **Implementation:** Static/Dynamic
	DRV_I2C_Open	Opens the specified instance of the I2C driver for use and provides an "open-instance" handle. **Implementation:** Dynamic
	DRV_I2C_QueueFlush	The existing transactions in the queue are voided and the queue pointers are reset to their initial state. This renders the queue empty.
	DRV_I2C_Receive	This function reads data written from either Master or Slave. **Implementation:** Dynamic
	DRV_I2C_Status	Provides the current status of the index instance of the I2C module. **Implementation:** Dynamic
	DRV_I2C_Tasks	Maintains the State Machine of the I2C driver and performs all the protocol level actions. **Implementation:** Dynamic
	DRV_I2C_TransferStatusGet	Returns status of data transfer when Master or Slave acts either as a transmitter or a receiver. **Implementation:** Dynamic
	DRV_I2C_Transmit	This function writes data to Master or Slave. **Implementation:** Dynamic
	DRV_I2C_TransmitForced	This function writes data to Master or Slave. **Implementation:** Dynamic
	DRV_I2C_TransmitThenReceive	This function writes data to Slave, inserts restart and requests read from slave. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_I2C_INDEX_0	Identifies the instance of the I2C driver.
	DRV_I2C_INDEX_1	Identifies the instance of the I2C driver.
	DRV_I2C_INDEX_2	Identifies the instance of the I2C driver.
	DRV_I2C_INDEX_3	Identifies the instance of the I2C driver.
	DRV_I2C_INDEX_4	Identifies the instance of the I2C driver.
	DRV_I2C_INDEX_5	Identifies the instance of the I2C driver.

Structures

	Name	Description
	_DRV_I2C_INIT	Identifies the initialization values that are passed as parameters to the initialize and reinitialize routines of the I2C module.
	DRV_I2C_INIT	Identifies the initialization values that are passed as parameters to the initialize and reinitialize routines of the I2C module.

Types

	Name	Description
	DRV_I2C_BUFFER_EVENT_HANDLER	Points to a callback after completion of an I2C transfer.

	DRV_I2C_BUFFER_HANDLE	Handle to an I2C data buffer.
	DRV_I2C_CallBack	Points to a callback to initiate a particular function.
	DRV_I2C_SLAVE_ADDRESS_MASK	Data type for a slave device address bit-mask.
	I2C_DATA_TYPE	This is type I2C_DATA_TYPE.
	I2C_SLAVE_ADDRESS_7bit	Data type for a slave device address.
	I2C_SLAVE_ADDRESS_VALUE	This is type I2C_SLAVE_ADDRESS_VALUE.

Description

I2C Device Driver Interface Header File

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the I2C module driver.

File Name

drv_i2c.h

Company

Microchip Technology Inc.

drv_i2c_bb.h

Contains prototypes for the I2C functions

Macros

	Name	Description
	DRV_I2C_BB_H	This is macro DRV_I2C_BB_H.

Description

I2C Bit Bang Functions Header File

File Name

drv_i2c_bb.h

Company

Microchip Technology Inc.

drv_i2c_config_template.h

I2C device driver configuration file.

Macros

	Name	Description
	DRV_DYNAMIC_BUILD	Dynamic driver build, dynamic device instance parameters.
	DRV_I2C_BUFFER_QUEUE_SUPPORT	Specifies if the Buffer Queue support should be enabled.
	DRV_I2C_CONFIG_BUILD_TYPE	Selects static or dynamic driver build configuration.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BASIC	Enables the device driver to support basic transfer mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BLOCKING	Enables the device driver to support blocking operations.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_EXCLUSIVE	Enables the device driver to support operation in Exclusive mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_MASTER	Enables the device driver to support operation in Master mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_NON_BLOCKING	Enables the device driver to support non-blocking during operations
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_READ	Enables the device driver to support read operations.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_SLAVE	Enables the device driver to support operation in Slave mode.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE	Enables the device driver to support write operations.
	DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE_READ	Enables the device driver to support write followed by read.

	DRV_I2C_FORCED_WRITE	Includes function that writes to slave irrespective of whether receiving a ACK or NACK from slave
	DRV_I2C_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_I2C_INTERRUPT_MODE	Macro controls interrupt based operation of the driver
	DRV_I2C_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.
	DRV_STATIC_BUILD	Static driver build, static device instance parameters.
	I2C_STATIC_DRIVER_MODE	Selects the type of STATIC driver

Description

I2C Device Driver Configuration

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_i2c_config.h

Company

Microchip Technology Inc.

I2S Driver Library Help

This section describes the I2S Driver Library.

Introduction

This library provides an interface to manage the Audio Protocol Interface Modes of the Serial Peripheral Interface (SPI) module on the Microchip family of microcontrollers.

Description

The SPI module can be interfaced to most available codec devices to provide microcontroller-based audio solutions. The SPI module provides support to the audio protocol functionality via four standard I/O pins. The four pins that make up the audio protocol interface modes are:

- SDIx: Serial Data Input for receiving sample digital audio data (ADCDAT)
- SDOx: Serial Data Output for transmitting digital audio data (DACDAT)
- SCKx: Serial Clock, also known as bit clock (BCLK)
- /SSx: Left/Right Channel Clock (LRCK)

BCLK provides the clock required to drive the data out or into the module, while LRCK provides the synchronization of the frame based on the protocol mode selected.

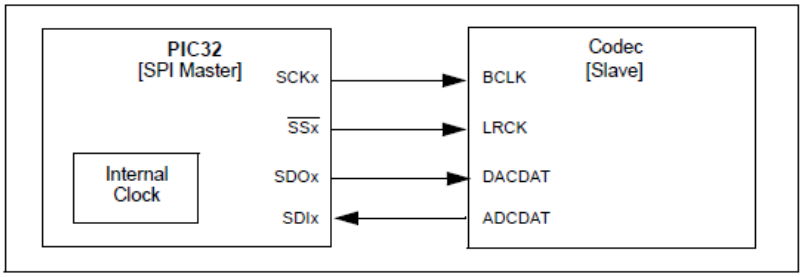
In Master mode, the module generates both the BCLK on the SCKx pin and the LRCK on the /SSx pin. In certain devices, while in Slave mode, the module receives these two clocks from its I2S partner, which is operating in Master mode.

When configured in Master mode, the leading edge of SCK and the LRCK are driven out within one SCK period of starting the audio protocol. Serial data is shifted in or out with timings determined by the protocol mode set.

In Slave mode, the peripheral drives zeros out SDO, but does not transmit the contents of the transmit FIFO until it sees the leading edge of the LRCK, after which time it starts receiving data.

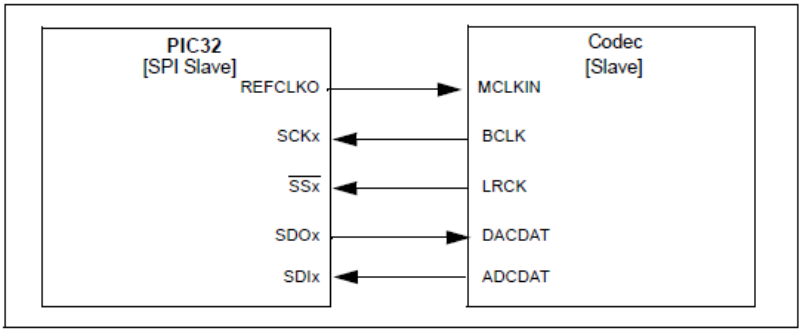
Master Mode

Master Generating its Own Clock – Output BCLK and LRCK



Slave Mode

Codec Device as Master Derives MCLK from PIC32 Reference Clock Out



Audio Protocol Modes

The SPI module supports four audio protocol modes and can be operated in any one of these modes:

- I2S mode
- Left-Justified mode
- Right-Justified mode
- PCM/DSP mode

Using the Library

This topic describes the basic architecture of the I2S Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_i2s.h`

The interface to the I2S Driver Library is defined in the `drv_i2s.h` header file. Any C language source (`.c`) file that uses the I2S Driver Library should include `drv_i2s.h`.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

The SPI Peripheral Library provides the low-level abstraction of the SPI module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the I2S Driver Library interface.

Description

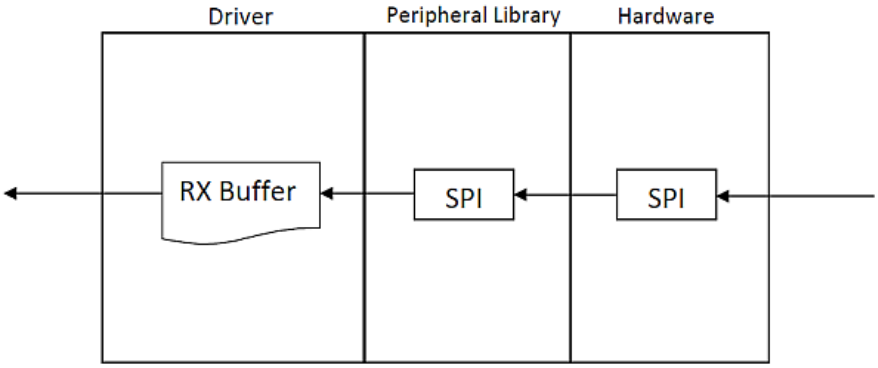
I2S Software Abstraction Block Diagram

Different types of SPIs are available on Microchip microcontrollers. Some have an internal buffer mechanism and some do not. The buffer depth varies across part families. The SPI Peripheral Library provides the ability to access these buffers. The I2S Driver Library abstracts out these differences and provides a unified model for audio data transfer across different types of SPI modules.

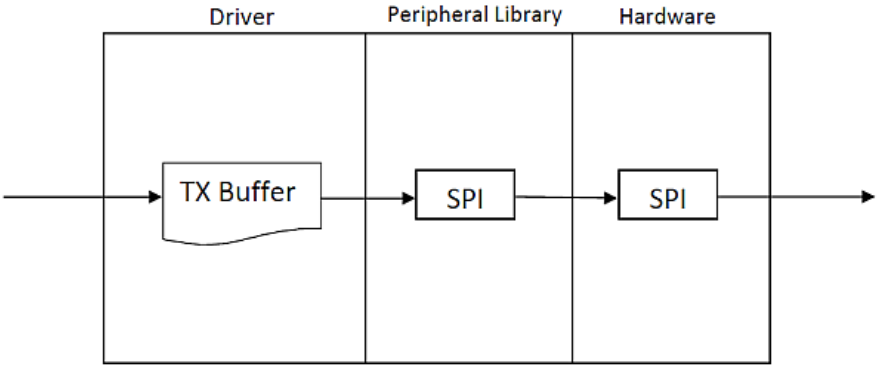
Both the transmitter and receiver provide a buffer in the driver, which transmits and receives data to/from the hardware. The I2S Driver Library provides a set of interfaces to perform the read and the write.

The following diagrams illustrate the model used by the I2S Driver Library for the transmitter and receiver.

Receiver Abstraction Model



Transmitter Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The I2S driver library provides an API interface to transfer/receive digital audio data using supported Audio protocols. The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the I2S Driver Library.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Setup Functions	Provides open and close functions.
Data Transfer Functions	Provides data transfer functions.
Miscellaneous Functions	Provides driver miscellaneous functions such as baud rate setting, get error functions, etc.
Data Types and Constants	These data types and constants are required while interacting and setting up the I2S Driver Library.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

System Access

This section provides information on system access.

Description

System Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization, each instance of the I2S module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_I2S_INIT or by using Initialization Overrides) that are supported by the specific I2S device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- The actual peripheral ID enumerated as the PLIB level module ID (e.g., SPI_ID_2)
- Defining the respective interrupt sources for TX, RX, DMA TX Channel, DMA RX Channel and Error Interrupt

The DRV_I2S_Initialize API returns an object handle of the type SYS_MODULE_OBJ. The object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_I2S_Deinitialize, DRV_I2S_Status, DRV_I2S_Tasks, and DRV_I2S_TasksError.

Notes:

1. The system initialization setting only effect the instance of the peripheral that is being initialized.
2. Configuration of the dynamic driver for DMA mode(uses DMA channel for data transfer) or Non DMA mode can be performed by appropriately setting the 'dmaChannelTransmit' and 'dmaChannelReceive' variables of the DRV_I2S_INIT structure. For example the TX will be in DMA mode when 'dmaChannelTransmit' is initialized to a valid supported channel number from the enum DMA_CHANNEL. TX will be in Non DMA mode when 'dmaChannelTransmit' is initialized to 'DMA_CHANNEL_NONE'.

Example:

```
    DRV_I2S_INIT                  init;
    SYS_MODULE_OBJ              objectHandle;

    init.moduleInit.value           = SYS_MODULE_POWER_RUN_FULL;
    init.spiID                      = SPI_ID_1;
    init.usageMode                  = DRV_I2S_MODE_MASTER;
    init.baudClock                  = SPI_BAUD_RATE_MCLK_CLOCK;
    init.baud                       = 48000;
    init.clockMode                  = DRV_I2S_CLOCK_MODE_IDLE_HIGH_EDGE_FALL;
    init.audioCommWidth             = SPI_AUDIO_COMMUNICATION_24DATA_32FIFO_32CHANNEL;
    init.audioTransmitMode          = SPI_AUDIO_TRANSMIT_STEREO;
    init.inputSamplePhase           = SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE;
    init.protocolMode               = DRV_I2S_AUDIO_I2S;
    init.txInterruptSource          = INT_SOURCE_SPI_1_TRANSMIT;
    init.rxInterruptSource          = INT_SOURCE_SPI_1_RECEIVE;
    init.errorInterruptSource       = INT_SOURCE_SPI_1_ERROR;
    init.queueSizeTransmit          = 3;
    init.queueSizeReceive           = 2;
    init.dmaChannelTransmit         = DMA_CHANNEL_NONE;
    init.dmaChannelReceive          = DMA_CHANNEL_NONE;

  objectHandle = DRV_I2S_Initialize(DRV_I2S_INDEX_1, (SYS_MODULE_INIT*)init);
    if (SYS_MODULE_OBJ_INVALID == objectHandle)
    {
        // Handle error
    }
```

Task Routine

In a polled environment, the system will call DRV_I2S_Tasks and DRV_I2S_TasksError from the System Task Service. In an interrupt-based implementation, DRV_I2S_Tasks and DRV_I2S_TasksError will be called from the Interrupt Service Routine of the I2S. When a DMA channel is used for transmission/reception DRV_I2S_Tasks and DRV_I2S_TasksError will be internally called by the driver from the DMA channel event handler.

Client Access

This section provides information on general client operation.

Description

General Client Operation

For the application to start using an instance of the module, it must call the DRV_I2S_Open function. This provides the settings required to open the I2S instance for operation. If the driver is deinitialized using the function DRV_I2S_Deinitialize, the application must call the DRV_I2S_Open function again to set up the instance of the I2S.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

Example:

```
    DRV_HANDLE handle;
    handle = DRV_I2S_Open(DRV_I2S_INDEX_0, (DRV_IO_INTENT_WRITE | DRV_IO_INTENT_NONBLOCKING));
    if (DRV_HANDLE_INVALID == handle)
    {
        // Unable to open the driver
        // May be the driver is not initialized or the initialization
        // is not complete.
    }
```

Client Operations - Buffered

This section provides information on buffered client operations.

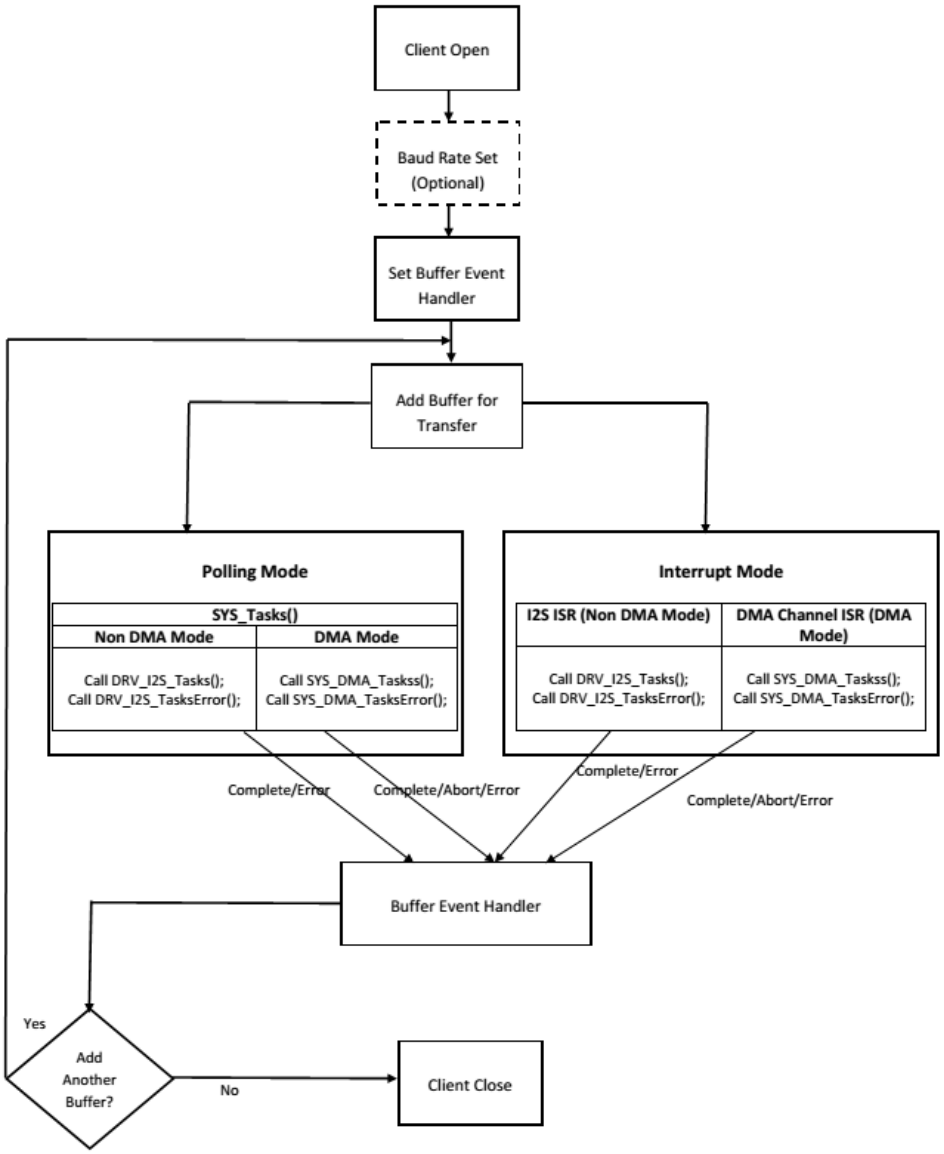
Description

Client Operations - Buffered

Client buffered operations provide a the typical audio interface. The functions DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite, and DRV_I2S_BufferAddWriteRead are the buffered data operation functions. The buffered functions schedules non-blocking operations. The function adds the request to the hardware instance queues and returns a buffer handle. The requesting client also registers a callback event with the driver. The driver notifies the client with DRV_I2S_BUFFER_EVENT_COMPLETE, DRV_I2S_BUFFER_EVENT_ERROR or DRV_I2S_BUFFER_EVENT_ABORT events.

The buffer add requests are processed under DRV_I2S_Tasks, DRV_I2S_TasksError functions. These functions are called from the I2S channel ISR in interrupt mode or from SYS_Tasks routine in Polled mode. When a DMA channel is used for transmission/reception DRV_I2S_Tasks and DRV_I2S_TasksError will be internally called by the driver from the DMA channel event handler.

The following diagram illustrates the buffered data operations



Note: It is not necessary to close and reopen the client between multiple transfers.

An application using the buffered functionality needs to perform the following steps:

1. The system should have completed necessary setup and initializations.
2. If DMA mode is desired, the DMA should be initialized by calling SYS_DMA_Initialize.
3. The necessary ports setup and remapping must be done for I2S lines: ADCDAT, DACDAT, BCLK, LRCK and MCLK (if required).

4. The driver object should have been initialized by calling DRV_I2S_Initialize. If DMA mode is desired, related attributes in the init structure must be set.
5. Open the driver using DRV_I2S_Open with the necessary ioIntent to get a client handle.
6. The necessary BCLK, LRCK, and MCLK should be set up so as to generate the required media bit rate.
7. The necessary Baud rate value should be set up by calling DRV_I2S_BaudrateSet.
8. The Register and event handler for the client handle should be set up by calling DRV_I2S_BufferEventHandlerSet.
9. Add a buffer to initiate the data transfer by calling DRV_I2S_BufferAddWrite/DRV_I2S_BufferAddRead/DRV_I2S_BufferAddWriteRead.
10. Based on polling or interrupt mode service the data processing should be set up by calling DRV_I2S_Tasks, DRV_I2S_TasksError from system tasks or I2S ISR. When a DMA channel is used for transmission/reception system calls SYS_DMA_Tasks(), SYS_DMA_TasksError() from the system tasks or DMA channel ISR, DRV_I2S_Tasks and DRV_I2S_TasksError will be internally called by the driver from the DMA channel event handler.
11. Repeat step 9 through step 10 to handle multiple buffer transmission and reception.
12. When the client is done it can use DRV_I2S_Close to close the client handle.

Example 1:

```
// The following is an example for a Polled mode buffered transmit

#define SYS_I2S_DRIVER_INDEX DRV_I2S_1 // I2S Uses SPI Hardware
#define BUFFER_SIZE 1000
// I2S initialization structure.
// This should be populated with necessary settings.
// attributes dmaChannelTransmit/dmaChannelReceive
// and dmaInterruptTransmitSource/dmaInterruptReceiveSource
// must be set if DMA mode of operation is desired.
DRV_I2S_INIT i2sInit;
SYS_MODULE_OBJ sysObj;      //I2S module object
DRV_HANDLE handle;          //Client handle
uint32_t i2sClock;          //BCLK frequency
uint32_t baudrate;          //baudrate
uint16_t myAudioBuffer[BUFFER_SIZE]; //Audio buffer to be transmitted
DRV_I2S_BUFFER_HANDLE bufferHandle;
APP_DATA_S state;           //Application specific state
uintptr_t contextHandle;

void SYS_Initialize ( void* data )
{
    // The system should have completed necessary setup and initializations.
    // Necessary ports setup and remapping must be done for I2S lines ADCDAT,
    // DACDAT, BCLK, LRCK and MCLK

    sysObj = DRV_I2S_Initialize(SYS_I2S_DRIVER_INDEX, (SYS_MODULE_INIT*)&i2sInit);
    if (SYS_MODULE_OBJ_INVALID == sysObj)
    {
        // Handle error
    }
}


void App_Task(void)
{
        switch(state)
        {
            case APP_STATE_INIT:
            {
                handle = DRV_I2S_Open(SYS_I2S_DRIVER_INDEX, (DRV_IO_INTENT_WRITE |
DRV_IO_INTENT_NONBLOCKING));
                if(handle != DRV_HANDLE_INVALID )
                {
                    /* Update the state */
                    state = APP_STATE_WAIT_FOR_READY;
                }
            }
            break;

            case  APP_STATE_WAIT_FOR_READY:
            {
                // Necessary clock settings must be done to generate
```

```
                // required MCLK, BCLK and LRCK
                DRV_I2S_BaudrateSet(handle, i2sClock, baudrate);

                /* Set the Event handler */
                DRV_I2S_BufferEventHandlerSet(handle,App_BufferEventHandler,
                contextHandle);

                /* Add a buffer to write*/
                DRV_I2S_BufferAddWrite(handle, &bufferHandle
                                       myAudioBuffer, BUFFER_SIZE);
                if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
                {
                    // Error handling here
                }
                state = APP_STATE_IDLE;
            }
            break;

            case APP_STATE_WAIT_FOR_DONE:
                state = APP_STATE_DONE;
            break;

            case  APP_STATE_DONE:
                // Close done
                DRV_I2S_Close(handle);
            break;

            case  APP_STATE_IDLE:
                // Do nothing
            break;

            default:
            break;
        }
}


void App_BufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    uint8_t temp;

    if(DRV_I2S_BUFFER_EVENT_COMPLETE == event)
    {
        // Can set state = APP_STATE_WAIT_FOR_DONE;
        // Take Action as needed
    }
    else if(DRV_I2S_BUFFER_EVENT_ERROR == event)
    {
        // Take Action as needed
    }
    else if(DRV_I2S_BUFFER_EVENT_ABORT == event)
    {
        // Take Action as needed
    }
    else
    {
        // Do nothing
    }
}


void SYS_Tasks ( void )
{
    DRV_I2S_Tasks((SYS_MODULE_OBJ)sysObj);
    DRV_I2S_TasksError((SYS_MODULE_OBJ)sysObj);

    /* Call the application's tasks routine */
    APP_Tasks ( );
```

```
}
```

Example 2:

```
// The following is an example for interrupt mode buffered transmit

#define SYS_I2S_DRIVER_INDEX DRV_I2S_1 // I2S Uses SPI Hardware
#define BUFFER_SIZE 1000
// I2S initialization structure.
// This should be populated with necessary settings.
// attributes dmaChannelTransmit/dmaChannelReceive
// and dmaInterruptTransmitSource/dmaInterruptReceiveSource
// must be set if DMA mode of operation is desired.
DRV_I2S_INIT i2sInit;
SYS_MODULE_OBJ sysObj; //I2S module object
DRV_HANDLE handle;  //Client handle
uint32_t i2sClock;   //BCLK frequency
uint32_t baudrate;   //baudrate
uint16_t myAudioBuffer[BUFFER_SIZE]; //Audio buffer to be transmitted
DRV_I2S_BUFFER_HANDLE bufferHandle;
APP_DATA_S state; //Application specific state
uintptr_t contextHandle;

void SYS_Initialize ( void* data )
{
    // The system should have completed necessary setup and initializations.
    // Necessary ports setup and remapping must be done for I2S lines ADCDAT,
    // DACDAT, BCLK, LRCK and MCLK

    sysObj = DRV_I2S_Initialize(SYS_I2S_DRIVER_INDEX, (SYS_MODULE_INIT*)&i2sInit);
    if (SYS_MODULE_OBJ_INVALID == sysObj)
    {
        // Handle error
    }
}


void App_Task(void)
{
        switch(state)
        {
            case APP_STATE_INIT:
            {
                handle = DRV_I2S_Open(SYS_I2S_DRIVER_INDEX, (DRV_IO_INTENT_WRITE |
DRV_IO_INTENT_NONBLOCKING));
                if(handle != DRV_HANDLE_INVALID )
                {
                    /* Update the state */
                    state = APP_STATE_WAIT_FOR_READY;
                }
            }
            break;

            case  APP_STATE_WAIT_FOR_READY:
            {
                // Necessary clock settings must be done to generate
                // required MCLK, BCLK and LRCK
                DRV_I2S_BaudrateSet(handle, i2sClock, baudrate);

                /* Set the Event handler */
                DRV_I2S_BufferEventHandlerSet(handle,App_BufferEventHandler,
                contextHandle);

                /* Add a buffer to write*/
                DRV_I2S_BufferAddWrite(handle, &bufferHandle
                                       myAudioBuffer, BUFFER_SIZE);
                if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
                {
                    // Error handling here
```

```
                }
                state = APP_STATE_IDLE;
            }
            break;

            case APP_STATE_WAIT_FOR_DONE:

                state = APP_STATE_DONE;
            break;


            case  APP_STATE_DONE:
            {
                // Close done
                DRV_I2S_Close(handle);
            }
            break;

            case  APP_STATE_IDLE:
                // Do nothing
            break;

            default:
            break;
        }
}


void App_BufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    uint8_t temp;

    if(DRV_I2S_BUFFER_EVENT_COMPLETE == event)
    {
        // Can set state = APP_STATE_WAIT_FOR_DONE;
        // Take Action as needed
    }
    else if(DRV_I2S_BUFFER_EVENT_ERROR == event)
    {
        // Take Action as needed
    }
    else if(DRV_I2S_BUFFER_EVENT_ABORT == event)
    {
        // Take Action as needed
    }
    else
    {
        // Do nothing
    }
}


void SYS_Tasks ( void )
{
    /* Call the application's tasks routine */
    APP_Tasks ( );
}


void __ISR ( _SPI1_VECTOR ) _InterruptHandler_I2S1 ( void )
{
    // Call the "tasks" functions for I2S module
    DRV_I2S_Tasks((SYS_MODULE_OBJ)sysObj);
    DRV_I2S_TasksError((SYS_MODULE_OBJ)sysObj);
}


// If DMA Channel 1 was setup during initialization instead of the previous I2S ISR, the following should
```

```
be implemented
void __ISR ( _DMA1_VECTOR ) _InterruptHandler_DMA_CHANNEL_1 ( void )
{
    // Call the DMA system tasks which internally will call the I2S Tasks.
    SYS_DMA_Tasks((SYS_MODULE_OBJ)sysObj);
    SYS_DMA_TasksError((SYS_MODULE_OBJ)sysObj);
}
```

Client Operations - Non-buffered

This section provides information on non-buffered client operations.

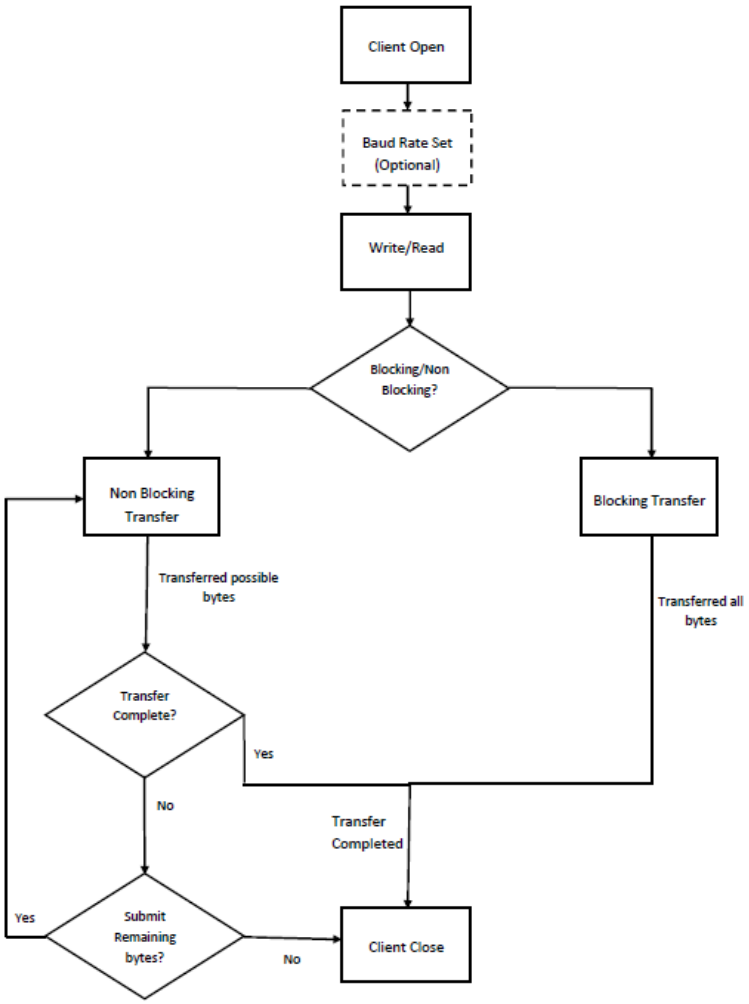
Description

Client Operations - Non-buffered

Client non-buffered operations provide a basic interface for the driver operation. This interface could be used by applications which have do not have buffered data transfer requirements. The functions DRV_I2S_Read and DRV_I2S_Write are the non-buffered data operation functions. The non-buffered functions are blocking/non-blocking depending upon the mode (ioIntent) the client was opened. If the client was opened for blocking mode these functions will only return when (or will block until) the specified data operation is completed or if an error occurred. If the client was opened for non-blocking mode, these functions will return with the number of bytes that were actually accepted for operation. The function will not wait until the data operation has completed.

Note: Non-buffered functions do not support interrupt/DMA mode.

The following diagram illustrates the non-buffered data operations



Note: It is not necessary to close and reopen the client between multiple transfers.

An application using the non-buffered functionality needs to perform the following steps:

1. The system should have completed necessary setup and initializations.
2. The necessary ports setup and remapping must be done for I2S lines: ADCDAT, DACDAT, BCLK, LRCK and MCLK (if required).
3. The driver object should have been initialized by calling DRV_I2S_Initialize.
4. Open the driver using DRV_I2S_Open with the necessary ioIntent to get a client handle.
5. The necessary BCLK, LRCK, and MCLK should be set up so as to generate the required media bit rate.
6. The necessary Baud rate value should be set up by calling DRV_I2S_BaudrateSet.
7. The Transmit/Receive data should be set up by calling DRV_I2S_Write/DRV_I2S_Read.
8. Repeat step 5 through step 7 to handle multiple buffer transmission and reception.
9. When the client is done it can use DRV_I2S_Close to close the client handle.

Example 1:

```
// The following is an example for a blocking transmit
    #define SYS_I2S_DRIVER_INDEX DRV_I2S_1 // I2S Uses SPI Hardware
    #define BUFFER_SIZE 1000
    DRV_I2S_INIT i2sInit; //I2S initialization structure
                          //This should be populated with necessary settings
    SYS_MODULE_OBJ sysObj; //I2S module object
    APP_DATA_S state; //Application specific state
    DRV_HANDLE handle;  //Client handle
    uint32_t i2sClock;   //BCLK frequency
    uint32_t baudrate;   //baudrate
    uint16_t myAudioBuffer[BUFFER_SIZE]; //Audio buffer to be transmitted
    uint32_t count;

    // The system should have completed necessary setup and initializations.
    // Necessary ports setup and remapping must be done for
    // I2S lines ADCDAT, DACDAT, BCLK, LRCK and MCLK

    sysObj = DRV_I2S_Initialize(SYS_I2S_DRIVER_INDEX, (SYS_MODULE_INIT*)&i2sInit);
    if (SYS_MODULE_OBJ_INVALID == sysObj)
    {
        // Handle error
    }
    while(1)
    {
        switch(state)
        {
            case APP_STATE_INIT:
            {
                handle = DRV_I2S_Open(SYS_I2S_DRIVER_INDEX, (DRV_IO_INTENT_WRITE | DRV_IO_INTENT_BLOCKING));
                if(handle != DRV_HANDLE_INVALID )
                {
                    /* Update the state */
                    state = APP_STATE_WAIT_FOR_READY;
                }
            }
            break;
            case  APP_STATE_WAIT_FOR_READY:
            {
                // Necessary clock settings must be done to generate
                // required MCLK, BCLK and LRCK
                DRV_I2S_BaudrateSet(handle, i2sClock, baudrate);
                // Blocks here and transfer the buffer
                count = DRV_I2S_Write(handle, &myAudioBuffer,BUFFER_SIZE);
                if(count == DRV_I2S_WRITE_ERROR)
                {
                    //Handle Error
                } else
                {
                    // Transfer Done
                    state = APP_STATE_DONE;
                }
            }
            break;
            case  APP_STATE_DONE:
            {
                // Close done
```

```
                DRV_I2S_Close(handle);
            }
            break;
            default:
            break;
        }
    }
```

Example 2:

```
// Following is an example for a non blocking transmit
    #define SYS_I2S_DRIVER_INDEX DRV_I2S_1 //I2S Uses SPI Hardware
    #define BUFFER_SIZE 1000
    DRV_I2S_INIT i2sInit; //I2S initialization structure.
                          // This should be populated with necessary settings
    SYS_MODULE_OBJ sysObj; //I2S module object
    APP_DATA_S state; //Application specific state
    DRV_HANDLE handle;  //Client handle
    uint32_t i2sClock;   //BCLK frequency
    uint32_t baudrate;   //baudrate
    uint16_t myAudioBuffer[BUFFER_SIZE]; //Audio buffer to be transmitted
    uint32_t count,total,size;

    total = 0;
    size = BUFFER_SIZE;

    // The system should have completed necessary setup and initializations.
    // Necessary ports setup and remapping must be done for I2S lines ADCDAT,
    // DACDAT, BCLK, LRCK and MCLK

    sysObj = DRV_I2S_Initialize(SYS_I2S_DRIVER_INDEX, (SYS_MODULE_INIT*)&i2sInit);
    if (SYS_MODULE_OBJ_INVALID == sysObj)
    {
        // Handle error
    }

    while(1)
    {
        switch(state)
        {
            case APP_STATE_INIT:
            {
                handle = DRV_I2S_Open(SYS_I2S_DRIVER_INDEX, (DRV_IO_INTENT_WRITE |
DRV_IO_INTENT_NONBLOCKING));
                if(handle != DRV_HANDLE_INVALID )
                {
                    /* Update the state */
                    state = APP_STATE_WAIT_FOR_READY;
                }
            }
            break;
            case  APP_STATE_WAIT_FOR_READY:
            {
                // Necessary clock settings must be done to generate
                // required MCLK, BCLK and LRCK
                DRV_I2S_BaudrateSet(handle, i2sClock, baudrate);
                // Transfer whatever possible number of bytes
                count = DRV_I2S_Write(handle, &myAudioBuffer,size);
                if(count == DRV_I2S_WRITE_ERROR)
                {
                    //Handle Error
                } else
                {
                    // 'count' bytes transferred
                    state = APP_STATE_WAIT_FOR_DONE;
                }
            }
            break;
            case APP_STATE_WAIT_FOR_DONE:
```

```
                // Can perform other Application tasks here
                // ......................
                // ......................
                // ......................
                size = size - count;
                if(size!=0)
                {
                    // Change the state so as to submit
                    // another possible transmission
                    state = APP_STATE_WAIT_FOR_READY;
                }
                else
                {
                    // We are done
                    state = APP_STATE_DONE;
                }
            break;
            case  APP_STATE_DONE:
            {
                if(DRV_I2S_CLOSE_FAILURE == DRV_I2S_Close(handle))
                {
                    // Handle error
                }
                else
                {
                    // Close done
                }
            }
            break;
            default:
            break;
        }
    }
```

Configuring the Library

Client Configuration

	Name	Description
	DRV_I2S_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_I2S_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.

System Configuration

	Name	Description
	DRV_I2S_INDEX	I2S Static Index selection
	DRV_I2S_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_I2S_INTERRUPT_MODE	Macro controls interrupt based operation of the driver
	DRV_I2S_INTERRUPT_SOURCE_ERROR	Defines the interrupt source for the error interrupt
	DRV_I2S_INTERRUPT_SOURCE_RECEIVE	Macro to define the Receive interrupt source in case of static driver
	DRV_I2S_INTERRUPT_SOURCE_TRANSMIT	Macro to define the Transmit interrupt source in case of static driver
	DRV_I2S_PERIPHERAL_ID	Configures the I2S PLIB Module ID
	DRV_I2S_RECEIVE_DMA_CHANNEL	Macro to defines the I2S Driver Receive DMA Channel in case of static driver
	DRV_I2S_STOP_IN_IDLE	Identifies whether the driver should stop operations in stop in Idle mode.
	DRV_I2S_TRANSMIT_DMA_CHANNEL	Macro to defines the I2S Driver Transmit DMA Channel in case of static driver
	DRV_I2S_RECEIVE_DMA_CHAINING_CHANNEL	Macro to defines the I2S Driver Receive DMA Chaining Channel in case of static driver

Description

The configuration of the I2S Driver Library is based on the file `sys_config.h`.

This header file contains the configuration selection for the I2S Driver Library. Based on the selections made, the I2S Driver Library may support the selected features. These configuration settings will apply to all instances of the I2S Driver Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

System Configuration

DRV_I2S_INDEX Macro

I2S Static Index selection

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_INDEX
```

Description

Index - Used for static drivers

I2S Static Index selection for the driver object reference. This macro defines the driver index in case of static and static multi-client build. For example, if this macro is set to DRV_I2S_INDEX_2, then static driver APIs would be DRV_I2S2_Initialize(), DRV_I2S2_Open() etc. When building static drivers, this macro should be different for each static build of the I2S driver that needs to be included in the project.

Remarks

This index is required to make a reference to the driver object

DRV_I2S_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_INSTANCES_NUMBER
```

Description

I2S driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of I2S modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None

DRV_I2S_INTERRUPT_MODE Macro

Macro controls interrupt based operation of the driver

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_INTERRUPT_MODE
```

Description

I2S Interrupt Mode Operation Control

This macro controls the interrupt based operation of the driver. The possible values it can take are

- true - Enables the interrupt mode
- false - Enables the polling mode

If the macro value is true, then Interrupt Service Routine for the interrupt should be defined in the application. The DRV_I2S_Tasks() routine should be called in the ISR.

DRV_I2S_INTERRUPT_SOURCE_ERROR Macro

Defines the interrupt source for the error interrupt

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_INTERRUPT_SOURCE_ERROR
```

Description

Error Interrupt Source

Macro to define the Error interrupt source in case of static driver. The interrupt source defined by this macro will override the errorInterruptSource member of the DRV_I2S_INIT initialization data structure in the driver initialization routine. This value should be set to the I2S module error interrupt enumeration in the Interrupt PLIB for the microcontroller.

DRV_I2S_INTERRUPT_SOURCE_RECEIVE Macro

Macro to define the Receive interrupt source in case of static driver

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_INTERRUPT_SOURCE_RECEIVE
```

Description

Receive Interrupt Source

Macro to define the Receive interrupt source in case of static driver. The interrupt source defined by this macro will override the rxInterruptSource member of the DRV_I2S_INIT initialization data structure in the driver initialization routine. This value should be set to the I2S module receive interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_I2S_INTERRUPT_SOURCE_TRANSMIT Macro

Macro to define the Transmit interrupt source in case of static driver

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_INTERRUPT_SOURCE_TRANSMIT
```

Description

Transmit Interrupt Source

Macro to define the TX interrupt source in case of static driver. The interrupt source defined by this macro will override the txInterruptSource member of the DRV_I2S_INIT initialization data structure in the driver initialization routine. This value should be set to the I2S module transmit interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_I2S_PERIPHERAL_ID Macro

Configures the I2S PLIB Module ID

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_PERIPHERAL_ID
```

Description

I2S Peripheral Library Module ID

This macro configures the PLIB ID if the driver is built statically. This value will override the I2SID member of the DRV_I2S_INIT initialization data structure. In that when the driver is built statically, the I2SID member of the DRV_I2S_INIT data structure will be ignored by the driver initialization routine and this macro will be considered. This should be set to the PLIB ID of I2S module (I2S_ID_1, I2S_ID_2 and so on).

DRV_I2S_RECEIVE_DMA_CHANNEL Macro

Macro to defines the I2S Driver Receive DMA Channel in case of static driver

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_RECEIVE_DMA_CHANNEL
```

Description

I2S Driver Receive DMA Channel

Macro to define the I2S Receive DMA Channel in case of static driver. The DMA channel defined by this macro will override the dmaChannelReceive member of the DRV_I2S_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel in the DMA PLIB for the microcontroller.

Remarks

None.

DRV_I2S_STOP_IN_IDLE Macro

Identifies whether the driver should stop operations in stop in Idle mode.

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_STOP_IN_IDLE
```

Description

I2S driver objects configuration

Identifies whether the driver should stop operations in stop in Idle mode. true - Indicates stop in idle mode. false - Indicates do not stop in Idle mode.

Remarks

None

DRV_I2S_TRANSMIT_DMA_CHANNEL Macro

Macro to defines the I2S Driver Transmit DMA Channel in case of static driver

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_TRANSMIT_DMA_CHANNEL
```

Description

I2S Driver Transmit DMA Channel

Macro to define the I2S Transmit DMA Channel in case of static driver. The DMA channel defined by this macro will override the dmaChannelTransmit member of the DRV_I2S_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel in the DMA PLIB for the microcontroller.

Remarks

None.

DRV_I2S_RECEIVE_DMA_CHAINING_CHANNEL Macro

Macro to defines the I2S Driver Receive DMA Chaining Channel in case of static driver

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_RECEIVE_DMA_CHAINING_CHANNEL
```

Description

I2S Driver Receive DMA Chaining Channel

Macro to define the I2S Receive DMA Chaining Channel in case of static driver. The DMA channel defined by this macro will override the dmaChaningChannelReceive member of the DRV_I2S_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel in the DMA PLIB for the microcontroller.

Remarks

None.

Client Configuration

DRV_I2S_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_CLIENTS_NUMBER
```

Description

I2S Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. This value represents the total number of clients to be supported across all hardware instances. So if I2S1 will be accessed by 2 clients and I2S2 will accessed by 3 clients, then this number should be 5. It is recommended that this be set exactly equal to the number of expected clients. Client support consumes RAM memory space. If this macro is not defined and the DRV_I2S_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - single client operation. If this macro is defined and the DRV_I2S_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - multi client operation.

Remarks

None

DRV_I2S_QUEUE_DEPTH_COMBINED Macro

Number of entries of all queues in all instances of the driver.

File

drv_i2s_config_template.h

C

```
#define DRV_I2S_QUEUE_DEPTH_COMBINED
```

Description

I2S Driver Buffer Queue Entries

This macro defined the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for transmit and receive operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build). The hardware instance transmit buffer queue will queue transmit buffers submitted by the

DRV_I2S_BufferAddWrite() function. The hardware instance receive buffer queue will queue receive buffers submitted by the DRV_I2S_BufferAddRead() function.

A buffer queue will contains buffer queue entries, each related to a BufferAdd request. This configuration macro defines total number of buffer entries that will be available for use between all I2S driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking read and write requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. More the number of buffer entries, greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its transmit and receive buffer queue size.

As an example, consider the case of static single client driver application where full duplex non blocking operation is desired without queuing, the minimum transmit queue depth and minimum receive queue depth should be 1. Hence the total number of buffer entries should be 2.

As an example, consider the case of a dynamic driver (say 2 instances) where instance 1 will queue up to 3 write requests and up to 2 read requests, and instance 2 will queue up to 2 write requests and up to 6 read requests, the value of this macro should be 13 (2 + 3 + 2 + 6).

Remarks

The maximum combined queue depth should not be greater than 0xFFFF(ie 65535)

Building the Library

This section lists the files that are available in the I2S Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/i2s`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_i2s.h`	This file provides the interface definitions of the I2S driver.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_i2s_dma.c`	This file contains the core implementation of the I2S driver with DMA support.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
`/src/dynamic/drv_i2s_dma_advanced.c`	This file contains the implementation of the I2S driver with DMA support using the channel chaining feature.
`/src/dynamic/drv_i2s.c`	This file contains the implementation of the I2S driver without DMA support.
`/src/dynamic/drv_i2s_read_write.c`	This file contains the basic read/write implementation of the I2S driver.

Module Dependencies

The I2S Driver Library depends on the following modules:

- SPI Peripheral Library
- DMA Peripheral Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_I2S_Deinitialize	Deinitializes the specified instance of the I2S driver module. **Implementation:** Dynamic
	DRV_I2S_Initialize	Initializes hardware and data for the instance of the I2S module. **Implementation:** Dynamic
	DRV_I2S_Status	Gets the current status of the I2S driver module. **Implementation:** Dynamic
	DRV_I2S_Tasks	Maintains the driver's receive state machine and implements its ISR. **Implementation:** Dynamic
	DRV_I2S_TasksError	Maintains the driver's error state machine and implements its ISR. **Implementation:** Dynamic

b) Client Setup Functions

	Name	Description
	DRV_I2S_Close	Closes an opened-instance of the I2S driver. **Implementation:** Dynamic
	DRV_I2S_Open	Opens the specified I2S driver instance and returns a handle to it. **Implementation:** Dynamic

c) Data Transfer Functions

	Name	Description
	DRV_I2S_BufferAddRead	Schedule a non-blocking driver read operation. **Implementation:** Dynamic
	DRV_I2S_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic
	DRV_I2S_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_I2S_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Dynamic
	DRV_I2S_BufferProcessedSizeGet	This function returns number of bytes that have been processed for the specified buffer. **Implementation:** Dynamic
	DRV_I2S_BufferCombinedQueueSizeGet	This function returns the number of bytes queued (to be processed) in the buffer queue. **Implementation:** Dynamic
	DRV_I2S_BufferQueueFlush	This function flushes off the buffers associated with the client object. **Implementation:** Dynamic
	DRV_I2S_Read	Reads data from the I2S. **Implementation:** Dynamic
	DRV_I2S_Write	Writes data to the I2S. **Implementation:** Dynamic

d) Miscellaneous Functions

	Name	Description
	DRV_I2S_BaudSet	This function sets the baud. **Implementation:** Dynamic
	DRV_I2S_ErrorGet	This function returns the error(if any) associated with the last client request. **Implementation:** Dynamic
	DRV_I2S_ReceiveErrorIgnore	This function enable/disable ignoring of the receive overflow error. **Implementation:** Dynamic
	DRV_I2S_TransmitErrorIgnore	This function enable/disable ignoring of the transmit underrun error. **Implementation:** Dynamic

e) Data Types and Constants

	Name	Description
	DRV_I2S_AUDIO_PROTOCOL_MODE	Identifies the Audio Protocol Mode of the I2S module.
	DRV_I2S_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_I2S_BUFFER_EVENT_HANDLER	Pointer to a I2S Driver Buffer Event handler function
	DRV_I2S_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.
	DRV_I2S_CLOCK_MODE	Identifies the various clock modes of the I2S module.
	DRV_I2S_DATA16	Defines the left and right channel data for 16-bit audio data
	_DRV_I2S_DATA16	Defines the left and right channel data for 16-bit audio data
	DRV_I2S_DATA24	Defines the left and right channel data for 24-bit audio data
	_DRV_I2S_DATA24	Defines the left and right channel data for 24-bit audio data
	DRV_I2S_DATA32	Defines the left and right channel data for 32-bit audio data
	_DRV_I2S_DATA32	Defines the left and right channel data for 32-bit audio data
	DRV_I2S_ERROR	Defines the possible errors that can occur during driver operation.
	DRV_I2S_INIT	Defines the data required to initialize or reinitialize the I2S driver
	DRV_I2S_MODE	Identifies the usage modes of the I2S module.
	DRV_I2S_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_I2S_COUNT	Number of valid I2S driver indices
	DRV_I2S_READ_ERROR	I2S Driver Read Error.
	DRV_I2S_WRITE_ERROR	I2S Driver Write Error.
	DRV_I2S_INDEX_0	I2S driver index definitions
	DRV_I2S_INDEX_1	This is macro DRV_I2S_INDEX_1.
	DRV_I2S_INDEX_2	This is macro DRV_I2S_INDEX_2.
	DRV_I2S_INDEX_3	This is macro DRV_I2S_INDEX_3.
	DRV_I2S_INDEX_4	This is macro DRV_I2S_INDEX_4.
	DRV_I2S_INDEX_5	This is macro DRV_I2S_INDEX_5.

Description

This section describes the Application Programming Interface (API) functions of the I2S Driver Library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_I2S_Deinitialize Function

Deinitializes the specified instance of the I2S driver module.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the I2S driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_I2S_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_I2S_Initialize
SYS_STATUS          status;


DRV_I2S_Deinitialize(object);

status = DRV_I2S_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_I2S_Initialize routine

Function

void DRV_I2S_Deinitialize(SYS_MODULE_OBJ object)

DRV_I2S_Initialize Function

Initializes hardware and data for the instance of the I2S module.

Implementation: Dynamic

File

drv_i2s.h

C

```
SYS_MODULE_OBJ DRV_I2S_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT *const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the I2S driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the I2S module ID. For example, driver instance 0 can be assigned to I2S2. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_I2S_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other I2S routine is called.

This routine should only be called once during system initialization unless DRV_I2S_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

To Enable the DMA mode of operation the init parameters 'dmaChannelTransmit' /'dmaChannelReceive' must be set to valid DMA channel. When DMA mode of operation is enabled, the normal mode(Usual TX and RX) operation is inhibited. When 'dmaChannelTransmit'/'dmaChannelReceive' are set to valid channel numbers the related DMA interrupt source parameters 'dmaInterruptTransmitSource'/ 'dmaInterruptReceiveSource' must be set with appropriate DMA channel interrupt source.

Preconditions

If DMA mode of operation is intended, SYS_DMA_Initialize should have been called before calling this function.

Example

```
DRV_I2S_INIT                init;
SYS_MODULE_OBJ              objectHandle;

init.moduleInit.value           = SYS_MODULE_POWER_RUN_FULL;
init.spiID                      = SPI_ID_1;
init.usageMode                  = DRV_I2S_MODE_MASTER;
init.baudClock                  = SPI_BAUD_RATE_MCLK_CLOCK;
init.baud                       = 48000;
```

```
init.clockMode                  = DRV_I2S_CLOCK_MODE_IDLE_HIGH_EDGE_FALL;
init.audioCommWidth             = SPI_AUDIO_COMMUNICATION_24DATA_32FIFO_32CHANNEL;
init.audioTransmitMode          = SPI_AUDIO_TRANSMIT_STEREO;
init.inputSamplePhase           = SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE;
init.protocolMode               = DRV_I2S_AUDIO_I2S;
init.txInterruptSource          = INT_SOURCE_SPI_1_TRANSMIT;
init.rxInterruptSource          = INT_SOURCE_SPI_1_RECEIVE;
init.errorInterruptSource       = INT_SOURCE_SPI_1_ERROR;
init.queueSizeTransmit          = 3;
init.queueSizeReceive           = 2;
init.dmaChannelTransmit         = DMA_CHANNEL_NONE;
init.dmaChannelReceive          = DMA_CHANNEL_NONE;

objectHandle = DRV_I2S_Initialize(DRV_I2S_INDEX_1, (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the driver instance to be initialized
init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ DRV_I2S_Initialize (const SYS_MODULE_INDEX drvIndex,
const SYS_MODULE_INIT *const init)

DRV_I2S_Status Function

Gets the current status of the I2S driver module.

Implementation: Dynamic

File

drv_i2s.h

C

```
SYS_STATUS DRV_I2S_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized

SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed

SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed

SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This routine provides the current status of the I2S driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_I2S_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_I2S_Initialize
SYS_STATUS          i2sStatus;

i2sStatus = DRV_I2S_Status(object);
if (SYS_STATUS_READY == i2sStatus)
{
    // This means the driver can be opened using the
    // DRV_I2S_Open() function.
```

```
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_I2S_Initialize routine

Function

SYS_STATUS DRV_I2S_Status(SYS_MODULE_OBJ object)

DRV_I2S_Tasks Function

Maintains the driver's receive state machine and implements its ISR.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal receive state machine and implement its transmit and receive ISR for interrupt-driven implementations. In polling mode, this function should be called from the SYS_Tasks function. In interrupt mode, this function should be called from the interrupt service routine of the I2S that is associated with this I2S driver hardware instance. In DMA mode of operation, this function should be called from the interrupt service routine of the channel associated with the transmission/reception of the I2s driver hardware instance.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_I2S_Initialize

while (true)
{
    DRV_I2S_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_I2S_Initialize)

Function

void DRV_I2S_Tasks(SYS_MODULE_OBJ object)

DRV_I2S_TasksError Function

Maintains the driver's error state machine and implements its ISR.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_TasksError(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal error state machine and implement its error ISR for interrupt-driven implementations. In polling mode, this function should be called from the SYS_Tasks() function. In interrupt mode, this function should be called in the error interrupt service routine of the I2S that is associated with this I2S driver hardware instance. In DMA mode of operation, this function should be called from the interrupt service routine of the channel associated with the transmission/reception of the I2s driver hardware instance.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_I2S_Initialize

while (true)
{
    DRV_I2S_TasksError (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_I2S_Initialize)

Function

void DRV_I2S_TasksError (SYS_MODULE_OBJ object)

b) Client Setup Functions

DRV_I2S_Close Function

Closes an opened-instance of the I2S driver.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_Close(const DRV_HANDLE handle);
```

Returns

- None

Description

This routine closes an opened-instance of the I2S driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_I2S_Open before the caller may use the driver again

Remarks

Usually there is no need for the driver client to verify that the Close operation has completed. The driver will abort any ongoing operations when this routine is called.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_I2S_Open

DRV_I2S_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_I2S_Close(DRV_Handle handle)

DRV_I2S_Open Function

Opens the specified I2S driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_i2s.h

C

```
DRV_HANDLE DRV_I2S_Open(const SYS_MODULE_INDEX iDriver, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the number of client objects allocated via DRV_I2S_CLIENTS_NUMBER is insufficient.
- if the client is trying to open the driver but driver has been opened exclusively by another client.
- if the driver hardware instance being opened is not initialized or is invalid.

Description

This routine opens the specified I2S driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options additionally affect the behavior of the DRV_I2S_Read() and DRV_I2S_Write() functions. If the ioIntent is DRV_IO_INTENT_NONBLOCKING, then these function will not block even if the required amount of data could not be processed. If the ioIntent is DRV_IO_INTENT_BLOCKING, these functions will block until the required amount of data is processed.

If ioIntent is DRV_IO_INTENT_READ, the client will only be read from the driver. If ioIntent is DRV_IO_INTENT_WRITE, the client will only be able to write to the driver. If the ioIntent in DRV_IO_INTENT_READWRITE, the client will be able to do both, read and write.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_I2S_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

Function DRV_I2S_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_I2S_Open(DRV_I2S_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
```

```
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

DRV_HANDLE DRV_I2S_Open(const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT ioIntent)

c) Data Transfer Functions

DRV_I2S_BufferAddRead Function

Schedule a non-blocking driver read operation.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_BufferAddRead(const DRV_HANDLE handle, DRV_I2S_BUFFER_HANDLE * bufferHandle, void * buffer,
size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_I2S_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking read operation. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance receive queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_I2S_BUFFER_HANDLE_INVALID:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for write-only
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_I2S_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_I2S_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the I2S Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another I2S driver instance. It should not otherwise be called directly in an ISR.

This function supports DMA mode of operation.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S device instance and the DRV_I2S_Status must have returned SYS_STATUS_READY.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_I2S_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;
```

```
// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver

DRV_I2S_BufferEventHandlerSet(myI2SHandle,
                APP_I2SBufferEventHandler, (uintptr_t)&myAppObj);

DRV_I2S_BufferAddRead(myI2Shandle, &bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the I2S instance as returned by the DRV_I2S_Open function
buffer	Buffer where the received data will be stored.
size	Buffer size in bytes
bufferHandle	Pointer to an argument that will contain the return buffer handle

Function

void DRV_I2S_BufferAddRead(const DRV_HANDLE handle,
DRV_I2S_BUFFER_HANDLE *bufferHandle,
void * buffer, size_t size)

DRV_I2S_BufferAddWrite Function

Schedule a non-blocking driver write operation.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_BufferAddWrite(const DRV_HANDLE handle, DRV_I2S_BUFFER_HANDLE * bufferHandle, void * buffer,
size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_I2S_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_I2S_BUFFER_HANDLE_INVALID:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read-only
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_I2S_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_I2S_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the I2S Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another I2S driver instance. It should not otherwise be called directly in an ISR.

This function supports DMA mode of operation.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S device instance and the DRV_I2S_Status must have returned SYS_STATUS_READY.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_I2S_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver

DRV_I2S_BufferEventHandlerSet(myI2SHandle,
                APP_I2SBufferEventHandler, (uintptr_t)&myAppObj);

DRV_I2S_BufferAddWrite(myI2Shandle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:
```

```
            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the I2S instance as return by the DRV_I2S_Open function
buffer	Data to be transmitted
size	Buffer size in bytes
bufferHandle	Pointer to an argument that will contain the return buffer handle

Function

void DRV_I2S_BufferAddWrite(const DRV_HANDLE handle,
DRV_I2S_BUFFER_HANDLE *bufferHandle,
void * buffer, size_t size);

DRV_I2S_BufferAddWriteRead Function

Schedule a non-blocking driver write-read operation.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_BufferAddWriteRead(const DRV_HANDLE handle, DRV_I2S_BUFFER_HANDLE * bufferHandle, void *
transmitBuffer, void * receiveBuffer, size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_I2S_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write-read operation. The function returns with a valid buffer handle in the bufferHandle argument if the write-read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_I2S_BUFFER_HANDLE_INVALID:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only or write only
- if the buffer size is 0
- if the queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_I2S_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_I2S_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the I2S Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another I2S driver instance. It should not otherwise be called directly in an ISR.

This function is useful when there is valid read expected for every I2S write. The transmit and receive size must be same.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S device instance and the DRV_I2S_Status must have returned SYS_STATUS_READY.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READWRITE must have been specified in the DRV_I2S_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybufferTx[MY_BUFFER_SIZE];
uint8_t mybufferRx[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver

DRV_I2S_BufferEventHandlerSet(myI2SHandle,
                APP_I2SBufferEventHandler, (uintptr_t)&myAppObj);

DRV_I2S_BufferAddWriteRead(myI2Shandle, &bufferHandle,
                                    mybufferTx,mybufferRx,MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the I2S instance as returned by the DRV_I2S_Open function
bufferHandle	Pointer to an argument that will contain the return buffer handle
transmitBuffer	Buffer where the transmit data will be stored
receiveBuffer	Buffer where the received data will be stored
size	Buffer size in bytes

Function

void DRV_I2S_BufferAddWriteRead(const DRV_HANDLE handle,
DRV_I2S_BUFFER_HANDLE *bufferHandle,
void *transmitBuffer, void *receiveBuffer,
size_t size)

DRV_I2S_BufferEventHandlerSet Function

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_BufferEventHandlerSet(DRV_HANDLE handle, const DRV_I2S_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls either the DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite or DRV_I2S_BufferAddWriteRead function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandler,
                                (uintptr_t)&myAppObj);

DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE handle, uintptr_t contextHandle)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;
```

```
            case DRV_I2S_BUFFER_EVENT_ERROR:

                // Error handling here.
                break;

            default:
                break;
        }
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_I2S_BufferEventHandlerSet(const DRV_HANDLE handle,
DRV_I2S_BUFFER_EVENT_HANDLER eventHandler,
uintptr_t contextHandle)

DRV_I2S_BufferProcessedSizeGet Function

This function returns number of bytes that have been processed for the specified buffer.

Implementation: Dynamic

File

drv_i2s.h

C

```
size_t DRV_I2S_BufferProcessedSizeGet(DRV_I2S_BUFFER_HANDLE bufferHandle);
```

Returns

Returns the number of the bytes that have been processed for this buffer. Returns 0 for an invalid or an expired buffer handle.

Description

This function returns number of bytes that have been processed for the specified buffer. The client can use this function, in a case where the buffer has terminated due to an error, to obtain the number of bytes that have been processed. If this function is called on a invalid buffer handle, or if the buffer handle has expired, the function returns 0.

Remarks

None.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

One of DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite or DRV_I2S_BufferAddWriteRead function must have been called and a valid buffer handle returned.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.
```

```
// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandle,
                                                    (uintptr_t)&myAppObj);

DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:

            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred.

            processedBytes = DRV_I2S_BufferProcessedSizeGet(bufferHandle);

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferhandle	Handle of the buffer of which the processed number of bytes to be obtained.

Function

size_t DRV_I2S_BufferProcessedSizeGet(DRV_I2S_BUFFER_HANDLE bufferHandle)

DRV_I2S_BufferCombinedQueueSizeGet Function

This function returns the number of bytes queued (to be processed) in the buffer queue.

Implementation: Dynamic

File

drv_i2s.h

C

```
size_t DRV_I2S_BufferCombinedQueueSizeGet(DRV_HANDLE handle);
```

Returns

Returns the number of the bytes that have been processed for this buffer. Returns 0 for an invalid or an expired client handle.

Description

This function returns the number of bytes queued (to be processed) in the buffer queue of the driver instance associated with the calling client. The client can use this function to know number of remaining bytes (from the buffers submitted by it)is in the queue to be transmitted.

Remarks

None.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

One of DRV_I2S_BufferAddRead/DRV_I2S_BufferAddWrite function must have been called and buffers should have been queued for transmission.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
size_t bufferQueuedSize;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandle,
                                                    (uintptr_t)&myAppObj);

DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// The data is being processed after adding the buffer to the queue.
// The user can get to know dynamically available data in the queue to be
// transmitted by calling DRV_I2S_BufferCombinedQueueSizeGet
bufferQueuedSize = DRV_I2S_BufferCombinedQueueSizeGet(myI2SHandle);
```

Parameters

Parameters	Description
handle	Opened client handle associated with a driver object.

Function

size_t DRV_I2S_BufferCombinedQueueSizeGet(DRV_HANDLE handle)

DRV_I2S_BufferQueueFlush Function

This function flushes off the buffers associated with the client object.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_BufferQueueFlush(DRV_HANDLE handle);
```

Returns

None.

Description

This function flushes off the buffers associated with the client object and disables the DMA channel used for transmission.

Remarks

None.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

One of DRV_I2S_BufferAddRead/DRV_I2S_BufferAddWrite function must have been called and buffers should have been queued for transmission.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;
size_t bufferQueuedSize;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandle,
                                                    (uintptr_t)&myAppObj);

DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// The data is being processed after adding the buffer to the queue.
// The user can stop the data processing and flushoff the data
// in the queue by calling DRV_I2S_BufferQueueFlush
DRV_I2S_BufferQueueFlush(myI2SHandle);
```

Parameters

Parameters	Description
handle	Opened client handle associated with a driver object.

Function

void DRV_I2S_BufferQueueFlush(DRV_HANDLE handle)

DRV_I2S_Read Function

Reads data from the I2S.

Implementation: Dynamic

File

drv_i2s.h

C

```
size_t DRV_I2S_Read(const DRV_HANDLE handle, uint8_t * buffer, const size_t numBytes);
```

Returns

Number of bytes actually copied into the caller's buffer. Returns DRV_I2S_READ_ERROR in case of an error.

Description

This routine reads data from the I2S. This function is blocking if the driver was opened by the client for blocking operation. This function will not

block if the driver was opened by the client for non blocking operation. If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_BLOCKING, this function will only return when (or will block until) numBytes of bytes have been received or if an error occurred.

If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_NON_BLOCKING, this function will return with the number of bytes that were actually read. The function will not wait until numBytes of bytes have been read.

Remarks

This function is thread safe in a RTOS application. It is recommended that this function not be called in I2S Driver Event Handler due to the potential blocking nature of the function. This function should not be called directly in an ISR. It should not be called in the event handler associated with another I2S driver instance.

This function does not supports DMA mode of operation.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_I2S_Open call.

Example

```
DRV_HANDLE      myI2SHandle;    // Returned from DRV_I2S_Open
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    count;
unsigned int    total;

total = 0;
do
{
    count  = DRV_I2S_Read(myI2SHandle, &myBuffer[total],
                          MY_BUFFER_SIZE - total);
    total += count;
    if(count == DRV_I2S_READ_ERROR)
    {
        // Handle error ...
    }
    else
    {
        // Do what needs to be..
    }
} while( total < MY_BUFFER_SIZE );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
buffer	Buffer into which the data read from the I2S instance will be placed.
numBytes	Total number of bytes that need to be read from the module instance (must be equal to or less than the size of the buffer)

Function

size_t DRV_I2S_Read(const DRV_HANDLE handle, uint8_t *buffer,

const size_t numBytes)

DRV_I2S_Write Function

Writes data to the I2S.

Implementation: Dynamic

File

drv_i2s.h

C

```
size_t DRV_I2S_Write(const DRV_HANDLE handle, uint8_t * buffer, const size_t numBytes);
```

Returns

Number of bytes actually written to the driver. Return DRV_I2S_WRITE_ERROR in case of an error.

Description

This routine writes data to the I2S. This function is blocking if the driver was opened by the client for blocking operation. This function will not block if the driver was opened by the client for non blocking operation. If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_BLOCKING, this function will only return when (or will block until) numbytes of bytes have been transmitted or if an error occurred.

If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_NON_BLOCKING, this function will return with the number of bytes that were actually accepted for transmission. The function will not wait until numBytes of bytes have been transmitted.

Remarks

This function is thread safe in a RTOS application. It is recommended that this function not be called in I2S Driver Event Handler due to the potential blocking nature of the function. This function should not be called directly in an ISR. It should not be called in the event handler associated with another USART driver instance.

This function does not supports DMA mode of operation.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_I2S_Open call.

Example

```
DRV_HANDLE myI2SHandle;    // Returned from DRV_I2S_Open
char myBuffer[MY_BUFFER_SIZE];
int count;
unsigned int total;
total = 0;
do
{
    count  = DRV_I2S_Write(myI2SHandle, &myBuffer[total],
            MY_BUFFER_SIZE - total);
    total += count;
    if(count == DRV_I2S_WRITE_ERROR)
    {
        // Handle error ...
    }
    else
    {
        // Do what needs to be ..
    }
} while( total < MY_BUFFER_SIZE );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
buffer	Buffer containing the data to written.
numbytes	size of the buffer

Function

size_t DRV_I2S_Write(const DRV_HANDLE handle, void * buffer,

const size_t numbytes)

d) Miscellaneous Functions

DRV_I2S_BaudSet Function

This function sets the baud.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_BaudSet(DRV_HANDLE handle, uint32_t spiClock, uint32_t baud);
```

Returns

None

Description

This function sets the baud rate for the I2S operation.

Remarks

None.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_HANDLE handle;
uint32_t baud;
uint32_t clock;


// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandle,
                                                        (uintptr_t)&myAppObj);

// Sets the baud rate to a new value as below
baud = 115200;
clock = 40000000UL;
DRV_I2S_BaudSet(myI2SHandle, clock, baud);

// Further perform the operation needed
DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;
```

```
            case DRV_I2S_BUFFER_EVENT_ERROR:

                // Error handling here.
                // We can find out how many bytes were processed in this
                // buffer before the error occurred.

                processedBytes = DRV_I2S_BufferProcessedSizeGet(bufferHandle);

                break;

            default:
                break;
        }
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
i2sClock	The Source clock frequency to the i2S module.
baud	The baud to be set.

Function

void DRV_I2S_BaudSet(DRV_HANDLE handle, uint32_t spiClock, uint32_t baud)

DRV_I2S_ErrorGet Function

This function returns the error(if any) associated with the last client request.

Implementation: Dynamic

File

drv_i2s.h

C

```
DRV_I2S_ERROR DRV_I2S_ErrorGet(DRV_HANDLE handle);
```

Returns

A DRV_I2S_ERROR type indicating last known error status.

Description

This function returns the error(if any) associated with the last client request. The DRV_I2S_Read() and DRV_I2S_Write() will update the client error status when these functions return DRV_I2S_READ_ERROR and DRV_I2S_WRITE_ERROR, respectively. If the driver send a DRV_I2S_BUFFER_EVENT_ERROR to the client, the client can call this function to know the error cause. The error status will be updated on every operation and should be read frequently (ideally immediately after the driver operation has completed) to know the relevant error status.

Remarks

It is the client's responsibility to make sure that the error status is obtained frequently. The driver will update the client error status regardless of whether this has been examined by the client.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_I2S_BUFFER_HANDLE bufferHandle;

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once
```

```
DRV_I2S_BufferEventHandlerSet( myI2SHandle, APP_I2SBufferEventHandle,
                                (uintptr_t)&myAppObj );

DRV_I2S_BufferAddRead( myI2Shandle,&bufferHandle,
                                     myBuffer, MY_BUFFER_SIZE );

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler( DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle )
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_SUCCESS:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_FAILURE:

            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred. We can also find
            // the error cause.

            processedBytes = DRV_I2S_BufferProcessedSizeGet(bufferHandle);
            if(DRV_I2S_ERROR_RECEIVE_OVERRUN == DRV_I2S_ErrorGet(myI2SHandle))
            {
                // There was an receive over flow error.
                // Do error handling here.
            }

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferhandle	Handle of the buffer of which the processed number of bytes to be obtained.

Function

DRV_I2S_ERROR DRV_I2S_ErrorGet(DRV_HANDLE handle)

DRV_I2S_ReceiveErrorIgnore Function

This function enable/disable ignoring of the receive overflow error.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_ReceiveErrorIgnore(DRV_HANDLE handle, bool errorEnable);
```

Returns

None

Description

A receive overflow is not a critical error; during receive overflow data in the FIFO is not overwritten by receive data. Ignore receive overflow is needed for cases when there is a general performance problem in the system that software must handle properly.

Remarks

None.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_HANDLE handle;
uint32_t baud;
uint32_t baud;*

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandle,
                                                        (uintptr_t)&myAppObj);

// Enable ignoring of receive overflow error
DRV_I2S_ReceiveErrorIgnore(myI2SHandle, true);

// Further perform the operation needed
DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:
```

```
            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred.

            processedBytes = DRV_I2S_BufferProcessedSizeGet(bufferHandle);

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
errorIgnore	When set to 'true' enables ignoring of transmit underrun error. When set to 'false' disables ignoring of transmit underrun error.

Function

void DRV_I2S_ReceiveErrorIgnore(DRV_HANDLE handle, bool errorEnable)

DRV_I2S_TransmitErrorIgnore Function

This function enable/disable ignoring of the transmit underrun error.

Implementation: Dynamic

File

drv_i2s.h

C

```
void DRV_I2S_TransmitErrorIgnore(DRV_HANDLE handle, bool errorIgnore);
```

Returns

None

Description

A Transmit underrun error is not a critical error and zeros are transmitted until the SPIxTXB is not empty. Ignore Transmit underrun error is needed for cases when software does not care or does not need to know about underrun conditions.

Remarks

None.

Preconditions

The DRV_I2S_Initialize routine must have been called for the specified I2S driver instance.

DRV_I2S_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_HANDLE handle;
uint32_t baud;
uint32_t baud;*

// myI2SHandle is the handle returned
// by the DRV_I2S_Open function.

// Client registers an event handler with driver. This is done once

DRV_I2S_BufferEventHandlerSet(myI2SHandle, APP_I2SBufferEventHandle,
                                                    (uintptr_t)&myAppObj);
```

```
// Enable ignoring of transmit underrun error
DRV_I2S_TransmitErrorIgnore(myI2SHandle, true);

// Further perform the operation needed
DRV_I2S_BufferAddRead(myI2Shandle,&bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_I2S_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_I2SBufferEventHandler(DRV_I2S_BUFFER_EVENT event,
        DRV_I2S_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:

            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred.

            processedBytes = DRV_I2S_BufferProcessedSizeGet(bufferHandle);

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
errorIgnore	When set to 'true' enables ignoring of transmit underrun error. When set to 'false' disables ignoring of transmit underrun error.

Function

void DRV_I2S_TransmitErrorIgnore(DRV_HANDLE handle, bool errorIgnore)

e) Data Types and Constants

DRV_I2S_AUDIO_PROTOCOL_MODE Enumeration

Identifies the Audio Protocol Mode of the I2S module.

File

drv_i2s.h

C

```
typedef enum {
```

```
  DRV_I2S_AUDIO_I2S,
  DRV_I2S_AUDIO_LFET_JUSTIFIED,
  DRV_I2S_AUDIO_RIGHT_JUSTIFIED,
  DRV_I2S_AUDIO_PCM_DSP
} DRV_I2S_AUDIO_PROTOCOL_MODE;
```

Members

Members	Description
DRV_I2S_AUDIO_I2S	I2S Audio Protocol
DRV_I2S_AUDIO_LFET_JUSTIFIED	Left Justified Audio Protocol
DRV_I2S_AUDIO_RIGHT_JUSTIFIED	Right Justified Audio Protocol
DRV_I2S_AUDIO_PCM_DSP	PCM/DSP Audio Protocol

Description

I2S Audio Protocol Mode

This enumeration identifies Audio Protocol Mode of the I2S module.

Remarks

None.

DRV_I2S_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_i2s.h

C

```
typedef enum {
  DRV_I2S_BUFFER_EVENT_COMPLETE,
  DRV_I2S_BUFFER_EVENT_ERROR,
  DRV_I2S_BUFFER_EVENT_ABORT
} DRV_I2S_BUFFER_EVENT;
```

Members

Members	Description
DRV_I2S_BUFFER_EVENT_COMPLETE	Data was transferred successfully.
DRV_I2S_BUFFER_EVENT_ERROR	Error while processing the request
DRV_I2S_BUFFER_EVENT_ABORT	Data transfer aborted (Applicable in DMA mode)

Description

I2S Driver Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite or DRV_I2S_BufferAddWriteRead functions.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_I2S_BufferEventHandlerSet function when a buffer transfer request is completed.

DRV_I2S_BUFFER_EVENT_HANDLER Type

Pointer to a I2S Driver Buffer Event handler function

File

drv_i2s.h

C

```
typedef void (* DRV_I2S_BUFFER_EVENT_HANDLER)(DRV_I2S_BUFFER_EVENT event, DRV_I2S_BUFFER_HANDLE
bufferHandle, uintptr_t contextHandle);
```

Returns

None.

Description

I2S Driver Buffer Event Handler Function

This data type defines the required function signature for the I2S driver buffer event handling callback function. A client must register a pointer to a buffer event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_I2S_BUFFER_EVENT_COMPLETE, this means that the data was transferred successfully.

If the event is DRV_I2S_BUFFER_EVENT_ERROR, this means that the data was not transferred successfully. The bufferHandle parameter contains the buffer handle of the buffer that failed. The DRV_I2S_ErrorGet function can be called to know the error. The DRV_I2S_BufferProcessedSizeGet function can be called to find out how many bytes were processed.

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event.

The context parameter contains a handle to the client context, provided at the time the event handling function was registered using the DRV_I2S_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The buffer handle in bufferHandle expires after this event handler exits. In that the buffer object that was allocated is deallocated by the driver after the event handler exits.

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

The DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite and DRV_I2S_BufferAddWriteRead functions can be called in the event handler to add a buffer to the driver queue. These functions can only be called to add buffers to the driver whose event handler is running. For example, buffers cannot be added I2S2 driver in I2S1 driver event handler.

Example

```
void APP_MyBufferEventHandler( DRV_I2S_BUFFER_EVENT event,
                               DRV_I2S_BUFFER_HANDLE bufferHandle,
                               uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_I2S_BUFFER_EVENT_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_I2S_BUFFER_EVENT_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the event relates
context	Value identifying the context of the application that registered the event handling function.

DRV_I2S_BUFFER_HANDLE Type

Handle identifying a read or write buffer passed to the driver.

File

drv_i2s.h

C

```
typedef uintptr_t DRV_I2S_BUFFER_HANDLE;
```

Description

I2S Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite, and DRV_I2S_BufferAddWriteRead functions. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None

DRV_I2S_CLOCK_MODE Enumeration

Identifies the various clock modes of the I2S module.

File

drv_i2s.h

C

```
typedef enum {
  DRV_I2S_CLOCK_MODE_IDLE_LOW_EDGE_RISE,
  DRV_I2S_CLOCK_MODE_IDLE_LOW_EDGE_FALL,
  DRV_I2S_CLOCK_MODE_IDLE_HIGH_EDGE_FALL,
  DRV_I2S_CLOCK_MODE_IDLE_HIGH_EDGE_RISE
} DRV_I2S_CLOCK_MODE;
```

Members

Members	Description
DRV_I2S_CLOCK_MODE_IDLE_LOW_EDGE_RISE	I2S Clock Mode 0 - Idle State Low & Sampling on Rising Edge
DRV_I2S_CLOCK_MODE_IDLE_LOW_EDGE_FALL	I2S Clock Mode 1 - Idle State Low & Sampling on Falling Edge
DRV_I2S_CLOCK_MODE_IDLE_HIGH_EDGE_FALL	I2S Clock Mode 2 - Idle State High & Sampling on Falling Edge
DRV_I2S_CLOCK_MODE_IDLE_HIGH_EDGE_RISE	I2S Clock Mode 3 - Idle State High & Sampling on Rising Edge

Description

I2S Clock Mode Selection

This enumeration identifies the supported clock modes of the I2S module.

Remarks

None.

DRV_I2S_DATA16 Structure

Defines the left and right channel data for 16-bit audio data

File

drv_i2s.h

C

```
typedef struct _DRV_I2S_DATA16 {
  int16_t leftData;
  int16_t rightData;
} DRV_I2S_DATA16;
```

Members

Members	Description
int16_t leftData;	Left channel data
int16_t rightData;	Right channel data

Description

I2S Driver Audio Data 16

Defines the left and right channel data for 16-bit audio data

Remarks

None.

DRV_I2S_DATA24 Structure

Defines the left and right channel data for 24-bit audio data

File

drv_i2s.h

C

```
typedef struct _DRV_I2S_DATA24 {
  int32_t leftData : 24;
  int32_t leftDataPad : 8;
  int32_t rightData : 24;
  int32_t rightDataPad : 8;
} DRV_I2S_DATA24;
```

Members

Members	Description
int32_t leftData : 24;	Left channel data
int32_t leftDataPad : 8;	Left channel data pad
int32_t rightData : 24;	Right channel data
int32_t rightDataPad : 8;	Right channel data pad

Description

I2S Driver Audio Data 24

Defines the left and right channel data for 24-bit audio data

Remarks

None.

DRV_I2S_DATA32 Structure

Defines the left and right channel data for 32-bit audio data

File

drv_i2s.h

C

```
typedef struct _DRV_I2S_DATA32 {
  int32_t leftData;
  int32_t rightDataPad;
} DRV_I2S_DATA32;
```

Members

Members	Description
int32_t leftData;	Left channel data
int32_t rightDataPad;	Right channel data

Description

I2S Driver Audio Data 32

Defines the left and right channel data for 32-bit audio data

Remarks

None.

DRV_I2S_ERROR Enumeration

Defines the possible errors that can occur during driver operation.

File

drv_i2s.h

C

```
typedef enum {
  DRV_I2S_ERROR_NONE,
  DRV_I2S_ERROR_RECEIVE_OVERFLOW,
  DRV_I2S_ERROR_TRANSMIT_UNDERUN,
  DRV_I2S_ERROR_FRAMING,
  DRV_I2S_ERROR_ADDRESS
} DRV_I2S_ERROR;
```

Members

Members	Description
DRV_I2S_ERROR_NONE	Data was transferred successfully.
DRV_I2S_ERROR_RECEIVE_OVERFLOW	Receive overflow error.
DRV_I2S_ERROR_TRANSMIT_UNDERUN	Transmit underrun error.
DRV_I2S_ERROR_FRAMING	Framing error.
DRV_I2S_ERROR_ADDRESS	Channel address error (Applicable in DMA mode)

Description

I2S Driver Error

This data type defines the possible errors that can occur when occur during USART driver operation. These values are returned by DRV_I2S_ErrorGet function.

Remarks

None.

DRV_I2S_INIT Structure

Defines the data required to initialize or reinitialize the I2S driver

File

drv_i2s.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SPI_MODULE_ID spiID;
  DRV_I2S_MODE usageMode;
  SPI_BAUD_RATE_CLOCK baudClock;
  uint32_t baud;
  DRV_I2S_CLOCK_MODE clockMode;
  SPI_AUDIO_COMMUNICATION_WIDTH audioCommWidth;
  SPI_AUDIO_TRANSMIT_MODE audioTransmitMode;
  SPI_INPUT_SAMPLING_PHASE inputSamplePhase;
  DRV_I2S_AUDIO_PROTOCOL_MODE protocolMode;
  INT_SOURCE txInterruptSource;
  INT_SOURCE rxInterruptSource;
  INT_SOURCE errorInterruptSource;
  uint32_t queueSizeTransmit;
  uint32_t queueSizeReceive;
  DMA_CHANNEL dmaChannelTransmit;
  DMA_CHANNEL dmaChaningChannelTransmit;
  DMA_CHANNEL dmaChannelReceive;
  DMA_CHANNEL dmaChaningChannelReceive;
  INT_SOURCE dmaInterruptTransmitSource;
  INT_SOURCE dmaInterruptChainingTransmitSource;
  INT_SOURCE dmaInterruptReceiveSource;
  INT_SOURCE dmaInterruptChainingReceiveSource;
} DRV_I2S_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization

SPI_MODULE_ID spiID;	Identifies I2S hardware module ID(PLIB-level SPI ID) For static build of the driver, this is overridden by DRV_I2S_MODULE_ID macro in system_config.h.
DRV_I2S_MODE usageMode;	Usage Mode Type
SPI_BAUD_RATE_CLOCK baudClock;	Select the clock which generates the baud rate The options available are Peripheral Clock/ Reference clock.
uint32_t baud;	Initial Baud Rate Value
DRV_I2S_CLOCK_MODE clockMode;	Clock mode
SPI_AUDIO_COMMUNICATION_WIDTH audioCommWidth;	Audio communication width
SPI_AUDIO_TRANSMIT_MODE audioTransmitMode;	Audio mono/stereo
SPI_INPUT_SAMPLING_PHASE inputSamplePhase;	Input Sample Phase Selection
DRV_I2S_AUDIO_PROTOCOL_MODE protocolMode;	Audio Protocol Mode
INT_SOURCE txInterruptSource;	Interrupt Source for Transmit Interrupt. For static build of the driver, this is overridden by the DRV_I2S_TRANSMIT_INTERRUPT_SOURCE macro in system_config.h.
INT_SOURCE rxInterruptSource;	Interrupt Source for Receive Interrupt. For static build of the driver, this is overridden by the DRV_I2S_RECEIVE_INTERRUPT_SOURCE macro in system_config.h.
INT_SOURCE errorInterruptSource;	Interrupt Source for Error Interrupt. For static build of the driver, this is overridden by the DRV_I2S_ERROR_INTERRUPT_SOURCE macro in system_config.h.
uint32_t queueSizeTransmit;	This is the transmit buffer queue size. This is the maximum number of write requests that driver will queue. For a static build of the driver, this is overridden by the DRV_I2S_TRANSMIT_QUEUE_SIZE macro in system_config.h.
uint32_t queueSizeReceive;	This is the receive buffer queue size. This is the maximum number of read requests that driver will queue. For a static build of the driver, this is overridden by the DRV_I2S_RECEIVE_QUEUE_SIZE macro in system_config.h.
DMA_CHANNEL dmaChannelTransmit;	This is the transmit DMA channel. A value of DMA_CHANNEL_NONE indicates DMA is not required for Tx. For a static build of the driver this is overridden by the DRV_I2S_TRANSMIT_DMA_CHANNEL macro in system_config.h.
DMA_CHANNEL dmaChaningChannelTransmit;	This is the transmit DMA chaining channel. This channel is needed when DMA is needed to be used channel chaining mode for Tx. Channel Chaining could be used to obtain high quality/resolution audio.
DMA_CHANNEL dmaChannelReceive;	This is the receive DMA channel. A value of DMA_CHANNEL_NONE indicates DMA is not required for Rx. For a static build of the driver this is overridden by the DRV_I2S_RECEIVE_DMA_CHANNEL macro in system_config.h.
DMA_CHANNEL dmaChaningChannelReceive;	This is the receive DMA chaining channel. This channel is needed when DMA is needed to be used channel chaining mode for Rx. Channel Chaining could be used to obtain high quality/resolution audio.
INT_SOURCE dmaInterruptTransmitSource;	This is the transmit DMA channel interrupt. This is applicable only if 'dmaChannelTransmit' has a valid channel number. This takes the interrupt source number for the corresponding DMA channel.
INT_SOURCE dmaInterruptChainingTransmitSource;	This is the transmit DMA chaining channel interrupt. This is applicable only if 'dmaChaningChannelTransmit' has a valid channel number. This takes the interrupt source number for the corresponding DMA chaining channel.
INT_SOURCE dmaInterruptReceiveSource;	This is the receive DMA channel interrupt. This is applicable only if 'dmaChannelReceive' has a valid channel number. This takes the interrupt source number for the corresponding DMA channel.
INT_SOURCE dmaInterruptChainingReceiveSource;	This is the receive DMA chaining channel interrupt. This is applicable only if 'dmaInterruptReceiveSource' has a valid channel number. This takes the interrupt source number for the corresponding DMA chaining channel.

Description

I2S Driver Initialization Data

This data type defines the data required to initialize or reinitialize the I2S driver. If the driver is built statically, some members of this data structure are statically over-ridden by static overrides in the system_config.h file.

Remarks

None.

DRV_I2S_MODE Enumeration

Identifies the usage modes of the I2S module.

File

drv_i2s.h

C

```
typedef enum {
  DRV_I2S_MODE_SLAVE,
  DRV_I2S_MODE_MASTER
} DRV_I2S_MODE;
```

Members

Members	Description
DRV_I2S_MODE_SLAVE	I2S Mode Slave
DRV_I2S_MODE_MASTER	I2S Mode Master

Description

I2S Usage Modes Enumeration

This enumeration identifies the whether the I2S module will be used as a master or slave.

Remarks

None.

DRV_I2S_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_i2s.h

C

```
#define DRV_I2S_BUFFER_HANDLE_INVALID ((DRV_I2S_BUFFER_HANDLE)(-1))
```

Description

I2S Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_I2S_BufferAddRead, DRV_I2S_BufferAddWrite and DRV_I2S_BufferAddWriteRead functions if the buffer add request was not successful.

Remarks

None

DRV_I2S_COUNT Macro

Number of valid I2S driver indices

File

drv_i2s.h

C

```
#define DRV_I2S_COUNT
```

Description

I2S Driver Module Count

This constant identifies the maximum number of I2S Driver instances that should be defined by the application. Defining more instances than this constant will waste RAM memory space.

This constant can also be used by the application to identify the number of I2S instances on this microcontroller.

Remarks

This value is part-specific.

DRV_I2S_READ_ERROR Macro

I2S Driver Read Error.

File

drv_i2s.h

C

```
#define DRV_I2S_READ_ERROR ((size_t)(-1))
```

Description

I2S Driver Read Error

This constant is returned by DRV_I2S_Read function when an error occurs.

Remarks

None.

DRV_I2S_WRITE_ERROR Macro

I2S Driver Write Error.

File

drv_i2s.h

C

```
#define DRV_I2S_WRITE_ERROR ((size_t)(-1))
```

Description

I2S Driver Write Error

This constant is returned by DRV_I2S_Write() function when an error occurs.

Remarks

None.

DRV_I2S_INDEX_0 Macro

I2S driver index definitions

File

drv_i2s.h

C

```
#define DRV_I2S_INDEX_0 0
```

Description

Driver I2S Module Index

These constants provide I2S driver index definition.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_I2S_Initialize and DRV_I2S_Open routines to identify the driver instance in use.

DRV_I2S_INDEX_1 Macro

File

drv_i2s.h

C

```
#define DRV_I2S_INDEX_1 1
```

Description

This is macro DRV_I2S_INDEX_1.

DRV_I2S_INDEX_2 Macro

File

drv_i2s.h

C

```
#define DRV_I2S_INDEX_2 2
```

Description

This is macro DRV_I2S_INDEX_2.

DRV_I2S_INDEX_3 Macro

File

drv_i2s.h

C

```
#define DRV_I2S_INDEX_3 3
```

Description

This is macro DRV_I2S_INDEX_3.

DRV_I2S_INDEX_4 Macro

File

drv_i2s.h

C

```
#define DRV_I2S_INDEX_4 4
```

Description

This is macro DRV_I2S_INDEX_4.

DRV_I2S_INDEX_5 Macro

File

drv_i2s.h

C

```
#define DRV_I2S_INDEX_5 5
```

Description

This is macro DRV_I2S_INDEX_5.

Files

Files

Name	Description
drv_i2s.h	I2S Driver Interface header file
drv_i2s_config_template.h	I2S Driver Configuration Template.

Description

drv_i2s.h

I2S Driver Interface header file

Enumerations

	Name	Description
	DRV_I2S_AUDIO_PROTOCOL_MODE	Identifies the Audio Protocol Mode of the I2S module.
	DRV_I2S_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_I2S_CLOCK_MODE	Identifies the various clock modes of the I2S module.
	DRV_I2S_ERROR	Defines the possible errors that can occur during driver operation.
	DRV_I2S_MODE	Identifies the usage modes of the I2S module.

Functions

	Name	Description
	DRV_I2S_BaudSet	This function sets the baud. **Implementation:** Dynamic
	DRV_I2S_BufferAddRead	Schedule a non-blocking driver read operation. **Implementation:** Dynamic
	DRV_I2S_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Dynamic
	DRV_I2S_BufferAddWriteRead	Schedule a non-blocking driver write-read operation. **Implementation:** Dynamic
	DRV_I2S_BufferCombinedQueueSizeGet	This function returns the number of bytes queued (to be processed) in the buffer queue. **Implementation:** Dynamic
	DRV_I2S_BufferEventHandlerSet	This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Dynamic
	DRV_I2S_BufferProcessedSizeGet	This function returns number of bytes that have been processed for the specified buffer. **Implementation:** Dynamic
	DRV_I2S_BufferQueueFlush	This function flushes off the buffers associated with the client object. **Implementation:** Dynamic
	DRV_I2S_Close	Closes an opened-instance of the I2S driver. **Implementation:** Dynamic
	DRV_I2S_Deinitialize	Deinitializes the specified instance of the I2S driver module. **Implementation:** Dynamic
	DRV_I2S_ErrorGet	This function returns the error(if any) associated with the last client request. **Implementation:** Dynamic
	DRV_I2S_Initialize	Initializes hardware and data for the instance of the I2S module. **Implementation:** Dynamic
	DRV_I2S_Open	Opens the specified I2S driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_I2S_Read	Reads data from the I2S. **Implementation:** Dynamic
	DRV_I2S_ReceiveErrorIgnore	This function enable/disable ignoring of the receive overflow error. **Implementation:** Dynamic
	DRV_I2S_Status	Gets the current status of the I2S driver module. **Implementation:** Dynamic
	DRV_I2S_Tasks	Maintains the driver's receive state machine and implements its ISR. **Implementation:** Dynamic
	DRV_I2S_TasksError	Maintains the driver's error state machine and implements its ISR. **Implementation:** Dynamic
	DRV_I2S_TransmitErrorIgnore	This function enable/disable ignoring of the transmit underrun error. **Implementation:** Dynamic
	DRV_I2S_Write	Writes data to the I2S. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_I2S_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_I2S_COUNT	Number of valid I2S driver indices
	DRV_I2S_INDEX_0	I2S driver index definitions

	DRV_I2S_INDEX_1	This is macro DRV_I2S_INDEX_1.
	DRV_I2S_INDEX_2	This is macro DRV_I2S_INDEX_2.
	DRV_I2S_INDEX_3	This is macro DRV_I2S_INDEX_3.
	DRV_I2S_INDEX_4	This is macro DRV_I2S_INDEX_4.
	DRV_I2S_INDEX_5	This is macro DRV_I2S_INDEX_5.
	DRV_I2S_READ_ERROR	I2S Driver Read Error.
	DRV_I2S_WRITE_ERROR	I2S Driver Write Error.

Structures

	Name	Description
	_DRV_I2S_DATA16	Defines the left and right channel data for 16-bit audio data
	_DRV_I2S_DATA24	Defines the left and right channel data for 24-bit audio data
	_DRV_I2S_DATA32	Defines the left and right channel data for 32-bit audio data
	DRV_I2S_DATA16	Defines the left and right channel data for 16-bit audio data
	DRV_I2S_DATA24	Defines the left and right channel data for 24-bit audio data
	DRV_I2S_DATA32	Defines the left and right channel data for 32-bit audio data
	DRV_I2S_INIT	Defines the data required to initialize or reinitialize the I2S driver

Types

	Name	Description
	DRV_I2S_BUFFER_EVENT_HANDLER	Pointer to a I2S Driver Buffer Event handler function
	DRV_I2S_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.

Description

I2S Driver Interface

The I2S device driver provides a simple interface to manage the I2S module on Microchip microcontrollers. This file provides the interface definition for the I2S driver.

File Name

drv_i2s.h

Company

Microchip Technology Inc.

drv_i2s_config_template.h

I2S Driver Configuration Template.

Macros

	Name	Description
	DRV_I2S_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_I2S_INDEX	I2S Static Index selection
	DRV_I2S_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_I2S_INTERRUPT_MODE	Macro controls interrupt based operation of the driver
	DRV_I2S_INTERRUPT_SOURCE_ERROR	Defines the interrupt source for the error interrupt
	DRV_I2S_INTERRUPT_SOURCE_RECEIVE	Macro to define the Receive interrupt source in case of static driver
	DRV_I2S_INTERRUPT_SOURCE_TRANSMIT	Macro to define the Transmit interrupt source in case of static driver
	DRV_I2S_PERIPHERAL_ID	Configures the I2S PLIB Module ID
	DRV_I2S_QUEUE_DEPTH_COMBINED	Number of entries of all queues in all instances of the driver.
	DRV_I2S_RECEIVE_DMA_CHAINING_CHANNEL	Macro to defines the I2S Driver Receive DMA Chaining Channel in case of static driver
	DRV_I2S_RECEIVE_DMA_CHANNEL	Macro to defines the I2S Driver Receive DMA Channel in case of static driver
	DRV_I2S_STOP_IN_IDLE	Identifies whether the driver should stop operations in stop in Idle mode.
	DRV_I2S_TRANSMIT_DMA_CHANNEL	Macro to defines the I2S Driver Transmit DMA Channel in case of static driver

Description

I2S Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_i2s_config_template.h

Company

Microchip Technology Inc.

Input Capture Driver Library

This section describes the Input Capture Driver Library.

Introduction

The Input Capture Static Driver provides a high-level interface to manage the Input Capture module on the Microchip family of microcontrollers.

Description

Through the MHC, this driver provides APIs for the following:

- Initializing the module
- Starting/Stopping of the capture
- 16/32-bit data reads
- Buffer empty status

Library Interface

Functions

	Name	Description
	DRV_IC_Initialize	Initializes the Input Capture instance for the specified driver index. **Implementation:** Static
	DRV_IC_BufferIsEmpty	Returns the Input Capture instance buffer empty status for the specified driver index. **Implementation:** Static
	DRV_IC_Capture16BitDataRead	Reads the 16-bit Input Capture for the specified driver index. **Implementation:** Static
	DRV_IC_Capture32BitDataRead	Reads the 32-bit Input Capture for the specified driver index. **Implementation:** Static
	DRV_IC_Start	Starts the Input Capture instance for the specified driver index. **Implementation:** Static
	DRV_IC_Stop	Stops the Input Capture instance for the specified driver index. **Implementation:** Static

Description

This section describes the Application Programming Interface (API) functions of the Input Capture Driver Library.

Functions

DRV_IC_Initialize Function

Initializes the Input Capture instance for the specified driver index.

Implementation: Static

File

help_drv_ic.h

C

```
void DRV_IC_Initialize();
```

Returns

None.

Description

This routine initializes the Input Capture driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters. The driver instance index is independent of the Input Capture module ID. For example, driver instance 0 can be assigned to Input Capture 2.

Remarks

This routine must be called before any other Input Capture routine is called. This routine should only be called once during system initialization.

Preconditions

None.

Function

void DRV_IC_Initialize(void)

DRV_IC_BufferIsEmpty Function

Returns the Input Capture instance buffer empty status for the specified driver index.

Implementation: Static

File

help_drv_ic.h

C

```
bool DRV_IC_BufferIsEmpty();
```

Returns

Boolean

- 1 - Buffer is empty
- 0 - Buffer is not empty

Description

Returns the Input Capture instance buffer empty status for the specified driver index. The function should be called to determine whether or not the IC buffer has data.

Remarks

None.

Preconditions

DRV_IC_Initialize has been called.

Function

bool DRV_IC_BufferIsEmpty(void)

DRV_IC_Capture16BitDataRead Function

Reads the 16-bit Input Capture for the specified driver index.

Implementation: Static

File

help_drv_ic.h

C

```
uint16_t DRV_IC_Capture16BitDataRead();
```

Returns

uint16_t value of the data read from the Input Capture.

Description

This routine reads the 16-bit data for the specified driver index.

Remarks

None.

Preconditions

DRV_IC_Initialize has been called.

Function

uint16_t DRV_IC_Capture16BitDataRead(void)

DRV_IC_Capture32BitDataRead Function

Reads the 32-bit Input Capture for the specified driver index.

Implementation: Static

File

help_drv_ic.h

C

```
uint32_t DRV_IC_Capture32BitDataRead();
```

Returns

uint32_t value of the data read from the Input Capture.

Description

This routine reads the 32-bit data for the specified driver index

Remarks

None.

Preconditions

DRV_IC_Initialize has been called.

Function

uint32_t DRV_IC_Capture32BitDataRead(void)

DRV_IC_Start Function

Starts the Input Capture instance for the specified driver index.

Implementation: Static

File

help_drv_ic.h

C

```
void DRV_IC_Start();
```

Returns

None.

Description

This routine starts the Input Capture driver for the specified driver index, starting an input capture.

Remarks

None.

Preconditions

DRV_IC_Initialize has been called.

Function

void DRV_IC_Start(void)

DRV_IC_Stop Function

Stops the Input Capture instance for the specified driver index.

Implementation: Static

File

help_drv_ic.h

C

```
void DRV_IC_Stop();
```

Returns

None.

Description

This routine stops the Input Capture driver for the specified driver index, stopping an input capture.

Remarks

None.

Preconditions

DRV_IC_Initialize has been called.

Function

void DRV_IC_Stop(void)

NVM Driver Library

This section describes the Non-volatile Memory (NVM) Driver Library.

Migrating Applications from v1.03.01 and Earlier Releases of MPLAB Harmony

Provides information on migrating applications from v1.03.01 and earlier releases of MPLAB Harmony to release v1.04 and later.

Description

The NVM Driver Library APIs have changed beginning with the v1.04 release of MPLAB Harmony. Applications that were developed using the earlier version of the MPLAB Harmony NVM Driver (v1.03.01 and earlier) will not build unless the application calls to NVM Driver are updated. While the MHC utility provides an option to continue creating applications using the v1.03.01 and earlier NVM Driver API, it is recommended that existing applications migrate to the latest API to take advantage of the latest features in the NVM Driver. The following sections describe the API changes and other considerations while updating the application for changes in the NVM Driver.

All NVM Driver Demonstration Applications and NVM Driver related documentation have been updated to the latest (new) API. The following sections do not discuss changes in the NVM Driver configuration related code. This code is updated automatically when the project is regenerated using the MHC utility. Only the application related API changes are discussed.

The following table shows the beta API and corresponding v1.04 and Later MPLAB Harmony NVM Driver API.

v1.03.01 and Earlier NVM Driver API	v1.04 and Later NVM Driver API	v1.04 and Later API Notes
DRV_NVM_Initialize	DRV_NVM_Initialize	The init structure now has additional members that allow the NVM media address and geometry to be specified.
DRV_NVM_Deinitialize	DRV_NVM_Deinitialize	No change.
DRV_NVM_Status	DRV_NVM_Status	No change.
DRV_NVM_Open	DRV_NVM_Open	No change.
DRV_NVM_Close	DRV_NVM_Close	No change.
DRV_NVM_Read	DRV_NVM_Read	Parameters have changed: • Returns the command handle associated with the read operation as an output parameter • Data is now read in terms of blocks. The read block size is specified in the NVM Geometry.

DRV_NVM_Write	DRV_NVM_Write	Parameters have changed: • Returns the command handle associated with the write operation as an output parameter • Data is now written in terms of blocks. The write block size is specified in the NVM Geometry.
DRV_NVM_Erase	DRV_NVM_Erase	Parameters have changed: • Returns the command handle associated with the erase operation as an output parameter • NVM Flash is erased in terms of blocks. The erase block size is specified in the NVM Geometry.
DRV_NVM_EraseWrite	DRV_NVM_EraseWrite	Parameters have changed: • Returns the command handle associated with the Erase/Write operation as an output parameter. • Data is now written in terms of blocks. The write block size is specified in the NVM Geometry.
DRV_NVM_BlockEventHandlerSet	DRV_NVM_EventHandlerSet	Function name and parameter type have changed.
DRV_NVM_ClientStatus	Not Available	This API is no longer available.
DRV_NVM_BufferStatus	DRV_NVM_CommandStatus	The DRV_NVM_Read, DRV_NVM_Write, DRV_NVM_Erase, and DRV_NVM_EraseWrite functions now return a command handle associated with the operation. The status of the operation can be checked by passing the command handle to this function.
Not Available	DRV_NVM_GeometryGet	This API gives the following geometrical details of the NVM Flash: • Media Property • Number of Read/Write/Erase regions in the flash device • Number of Blocks and their size in each region of the device
Not Available	DRV_NVM_IsAttached	Returns the physical attach status of the NVM Flash.
Not Available	DRV_NVM_IsWriteProtected	Returns the write protect status of the NVM Flash.
Not Available	DRV_NVM_AddressGet	Returns the NVM Media Start address.

NVM Driver Initialization

DRV_NVM_INIT now takes the following two additional initialization parameters:

- mediaStartAddress - NVM Media Start address. The driver treats this address as the start address for read, write and erase operations.
- nvmMediaGeometry - Indicates the layout of the media in terms of read, write and erase regions.

The following code examples show how the driver initialization was performed with 1.03 APIs and how it is performed with the 1.04 APIs:

Example 1: v1.03 and Earlier Code

```
const DRV_NVM_INIT drvNvmInit =
{
    .moduleInit.sys.powerState = SYS_MODULE_POWER_RUN_FULL,
    .nvmID = NVM_ID_0,
    .interruptSource = INT_SOURCE_FLASH_CONTROL,
};

void SYS_Initialize (void *data)
{
.
.
    // Initialize NVM Driver Layer
    sysObj.drvNvm = DRV_NVM_Initialize(DRV_NVM_INDEX_0, (SYS_MODULE_INIT *)&drvNvmInit);
.
}
```

Example: v1.04 and Later Code

```
/* NVM Geometry structure */
SYS_FS_MEDIA_REGION_GEOMETRY NVMGeometryTable[3] =
{
    {
        .blockSize = 1,
        .numBlocks = (DRV_NVM_MEDIA_SIZE * 1024),
    },
    {
       .blockSize = DRV_NVM_ROW_SIZE,
```

```
        .numBlocks = ((DRV_NVM_MEDIA_SIZE * 1024)/DRV_NVM_ROW_SIZE)
    },
    {
        .blockSize = DRV_NVM_PAGE_SIZE,
        .numBlocks = ((DRV_NVM_MEDIA_SIZE * 1024)/DRV_NVM_PAGE_SIZE)
    }
};

const SYS_FS_MEDIA_GEOMETRY NVMGeometry =
{
    .mediaProperty = SYS_FS_MEDIA_WRITE_IS_BLOCKING,
    .numReadRegions = 1,
    .numWriteRegions = 1,
    .numEraseRegions = 1,
    .geometryTable = (SYS_FS_MEDIA_REGION_GEOMETRY *)&NVMGeometryTable
};

const DRV_NVM_INIT drvNvmInit =
{
    .moduleInit.sys.powerState = SYS_MODULE_POWER_RUN_FULL,
    .nvmID = NVM_ID_0,
    .interruptSource = INT_SOURCE_FLASH_CONTROL,
    .mediaStartAddress = 0x9D010000,
    .nvmMediaGeometry = (SYS_FS_MEDIA_GEOMETRY *)&NVMGeometry
};

void SYS_Initialize (void *data)
{
.
.
    // Initialize NVM Driver Layer
    sysObj.drvNvm = DRV_NVM_Initialize(DRV_NVM_INDEX_0, (SYS_MODULE_INIT *)&drvNvmInit);
.
.
}
```

Addressing in NVM Driver

The v1.03.01 and earlier Read, Write, Erase and EraseWrite APIs took the actual address on which the operation was to be performed. The unit of access was bytes.

In v1.04 the addressing mechanism has been modified. The media start address is set in the DRV_NVM_Initialize. This address is used as the base address for the Read, Write, Erase and EraseWrite APIs. The unit of access is in terms of blocks. The NVM Geometry specifies the media layout in terms of:

- Number of erase, read and write regions
- Block size for erase, read and write operations.
- Number of blocks in erase, read and write regions

For example, in PIC32MZ family devices:

- Read block size = 1 byte
- Write block size = ROW Size = 2048 bytes
- Erase block size = PAGE Size = 16384 bytes

If the size of media is 32 KB then the following table illustrates the address range and number of blocks for the read, write and erase regions:

Region Type	Block Size	Number of blocks	Address range
Read Region	1 Byte	32 KB / Read block size = 32768	0–32767
Write Region	2048 Bytes	32 KB / Write block size = 16 blocks	0–15
Erase Region	16384 Bytes	32 KB / Erase block size = 2 blocks	0–1

Erasing Data on NVM Flash

The NVM Geometry indicates the number of erase blocks and the size of a single erase block. The Erase API takes in the erase block start address and the number of blocks to be erased. The following code examples show how to perform the erase operation in v1.03.01 and earlier and how to perform it with v1.04 and later.

Example: v1.03.01 and Earlier Code

```
DRV_HANDLE      myNVMHandle;    // Returned from DRV_NVM_Open
DRV_NVM_BUFFER_HANDLE bufferHandle;
```

```
bufferHandle = DRV_NVM_Erase(myNVMHandle, (uint8_t*)NVM_BASE_ADDRESS, DRV_NVM_PAGE_SIZE);
if(DRV_NVM_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Do error handling here
}

// Wait until the buffer completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
while(DRV_NVM_BufferStatus(bufferHandle) != DRV_NVM_BUFFER_COMPLETED);
```

Example: v1.04 and Later Code

```
/* This code example shows how to erase NVM Media data */
DRV_HANDLE nvmHandle;
DRV_NVM_COMMAND_HANDLE nvmCommandHandle;
DRV_NVM_COMMAND_STATUS commandStatus;
uint32_t blockAddress;
uint32_t nBlocks;

blockAddress = 0;
nBlocks = 1;

DRV_NVM_Erase(nvmHandle, &nvmCommandHandle, blockAddress, nBlocks);
if(DRV_NVM_COMMAND_HANDLE_INVALID == nvmCommandHandle)
{
    /* Failed to queue the erase request. Handle the error. */
}
// Wait until the command completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.

commandStatus = DRV_NVM_CommandStatus(nvmHandle, nvmCommandHandle);
if(DRV_NVM_COMMAND_COMPLETED == commandStatus)
{
    /* Erase completed */
}
else if (DRV_NVM_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Erase Failed */
}
```

Writing Data to NVM Flash

The NVM Geometry indicates the number of write blocks and the size of a single write block. The Write API takes in the write block start address and the number of blocks to be written. The following code examples show how the write operation was performed in v1.03.01 and earlier and how to perform it with v1.04 and later APIs:

Example : v1.03.01 and Earlier Code

```
DRV_HANDLE myNVMHandle;    // Returned from DRV_NVM_Open
char       myBuffer[2 * DRV_NVM_ROW_SIZE];

// Destination address should be row aligned.
char       *destAddress = (char *)NVM_BASE_ADDRESS_TO_WRITE;

unsigned int    count = 2 * MY_BUFFER_SIZE;
DRV_NVM_BUFFER_HANDLE bufferHandle;

bufferHandle  = DRV_NVM_Write(myNVMHandle, destAddress, &myBuffer[total], count);
if(DRV_NVM_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Do error handling here
}

// Wait until the buffer completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
while(DRV_NVM_BufferStatus(bufferHandle) != DRV_NVM_BUFFER_COMPLETED);
```

Example: v1.04 and Later Code

```
/* This code example shows how to write data to NVM Media */
DRV_HANDLE nvmHandle;
DRV_NVM_COMMAND_HANDLE nvmCommandHandle;
DRV_NVM_COMMAND_STATUS commandStatus;
uint8_t writeBuf[DRV_NVM_ROW_SIZE];
uint32_t blockAddress;
uint32_t nBlocks;

blockAddress = 0;
nBlocks = 1;

DRV_NVM_Write(nvmHandle, &nvmCommandHandle, (uint8_t *)writeBuf, blockAddress, nBlocks);
if(DRV_NVM_COMMAND_HANDLE_INVALID == nvmCommandHandle)
{
    /* Failed to queue the write request. Handle the error. */
}
// Wait until the command completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.

commandStatus = DRV_NVM_CommandStatus(nvmHandle, nvmCommandHandle);
if(DRV_NVM_COMMAND_COMPLETED == commandStatus)
{
    /* Write completed */
}
else if (DRV_NVM_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Write Failed */
}
```

Reading Data from NVM Flash

The NVM Geometry indicates the number of read blocks and the size of a single read block. The Read API takes in the read block start address and the number of blocks to be read. The following code examples show how the read operation was performed with v1.03.01 and earlier APIs and how to perform the same with v1.04 and later APIs:

Example: v1.03.01 and Earlier Code

```
DRV_HANDLE      myNVMHandle;    // Returned from DRV_NVM_Open
char            myBuffer[MY_BUFFER_SIZE];
char            *srcAddress = NVM_BASE_ADDRESS_TO_READ_FROM;
unsigned int    count = MY_BUFFER_SIZE;
DRV_NVM_BUFFER_HANDLE bufferHandle;

bufferHandle  = DRV_NVM_Read(myNVMHandle, &myBuffer[total], srcAddress, count);
if(DRV_NVM_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Do error handling here
}

// Wait until the buffer completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
while(DRV_NVM_BufferStatus(bufferHandle) != DRV_NVM_BUFFER_COMPLETED);
```

Example: v1.04 and Later Code

```
/* This code example shows how to read data from NVM Media */
DRV_HANDLE nvmHandle;
DRV_NVM_COMMAND_HANDLE nvmCommandHandle;
DRV_NVM_COMMAND_STATUS commandStatus;
uint8_t readBuf[DRV_NVM_ROW_SIZE];
uint32_t blockAddress;
uint32_t nBlocks;

blockAddress = 0;
nBlocks = DRV_NVM_ROW_SIZE;

DRV_NVM_Read(nvmHandle, &nvmCommandHandle, (uint8_t *)readBuf, blockAddress, nBlocks);
if(DRV_NVM_COMMAND_HANDLE_INVALID == nvmCommandHandle)
```

```
{
    /* Failed to queue the read request. Handle the error. */
}
// Wait until the command completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.

commandStatus = DRV_NVM_CommandStatus(nvmHandle, nvmCommandHandle);
if(DRV_NVM_COMMAND_COMPLETED == commandStatus)
{
    /* Read completed */
}
else if (DRV_NVM_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Read Failed */
}
```

Introduction

The NVM Driver library provides APIs that can be used to interface with the NVM module (controller plus memory) for memory needs.

Description

The NVM Driver provides APIs for block access of the physical media through NVM Driver APIs. As shown in the NVM Driver Abstraction Model, an application or a client can access the physical media using multiple methods, which eventually are facilitated through the NVM Driver.

Memory Devices for PIC Microcontrollers

Depending on the device, there are two primary forms of on-chip memory: Programmable Flash memory and data EEPROM memory. The access mechanism for both of these types are varied.

Flash Program Memory

The Flash program memory is readable, writeable, and erasable during normal operation over the entire operating voltage range.

A read from program memory is executed at one byte/word at a time depending on the width of the data bus.

A write to the program memory is executed in either blocks of specific sizes or a single word depending on the type of processor used.

An erase is performed in blocks. A bulk erase may be performed from user code depending on the type of processor supporting the operation.

Writing or erasing program memory will cease instruction fetches until the operation is complete, restricting memory access, and therefore preventing code execution. This is controlled by an internal programming timer.

There are three processor dependant methods for program memory modification:

- Run-Time Self-Programming (RTSP)
- In-Circuit Serial Programming (ICSP)
- EJTAG programming

This section describes the RTSP techniques.

Using the Library

This topic describes the basic architecture of the NVM Driver Library and provides information and examples on its use.

Description

Interface Header Files: `drv_nvm.h`

The interface to the NVM Driver Library is defined in the `drv_nvm.h` header file. Any C language source (`.c`) file that uses the NVM Driver library should include `drv_nvm.h`.

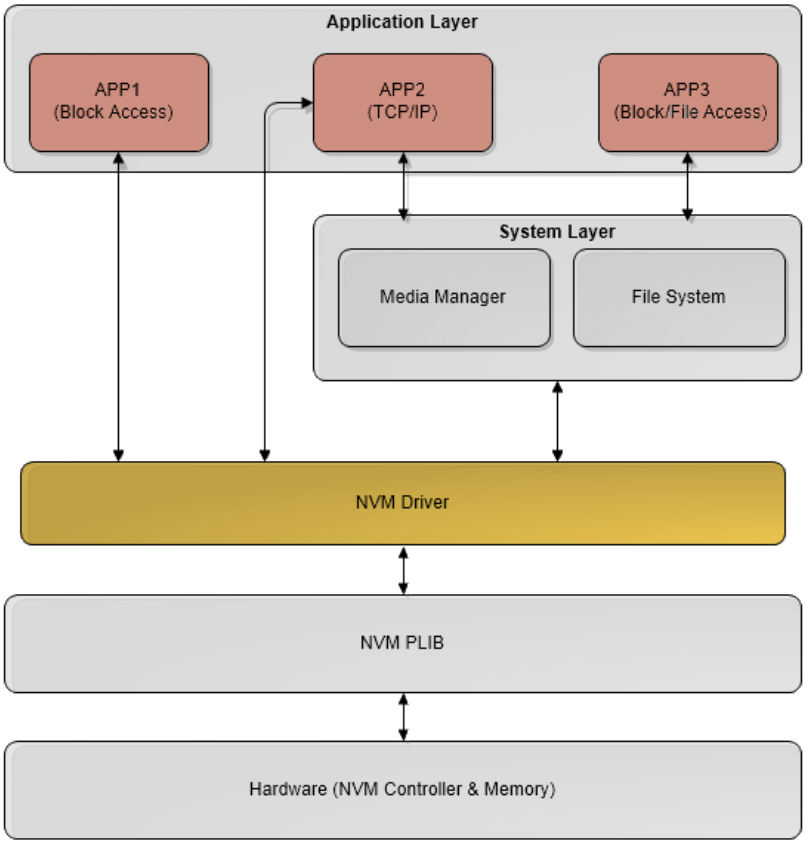
Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the NVM module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

NVM Driver Abstraction Model



Abstraction Model

As shown in the previous diagram, the NVM Driver sits between the Peripheral Libraries and the application or system layer to facilitate block and file access to the NVM media (currently Flash). The application scenarios show how different layers can be accessed by different applications with certain needs. For example, APP1 can access the NVM Driver directly to erase, write, or read NVM with direct addressing. APP2, in this case TCP/IP, can bypass the system layer and access the NVM Driver layer if necessary to fulfill its robust data needs. Finally, APP3 accesses the NVM Driver through the File System Layer using block access methods, so the application does not need to keep track of the physical layout of the media.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the NVM module.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Core Functions	Provides open, close, status and other setup functions.
Client Block Transfer Functions	Provides buffered data operation functions available in the core configuration.
Miscellaneous Functions	Provides driver miscellaneous functions related to versions and others.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality
- Media Functionality

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the modes supported for your device.

NVM Driver Initialization

This section provides information for system initialization and reinitialization.

Description

The system performs the initialization and the reinitialization of the device driver with settings that affect only the instance of the device that is being initialized or reinitialized. During system initialization each instance of the NVM module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_NVM_INIT or by using initialization overrides) that are supported by the specific NVM device hardware:

- Device requested power state: One of the system module power states. For specific details please refer to **Data Types and Constants** in the Library Interface section.
- The actual peripheral ID enumerated as the PLIB level module ID (e.g., NVM_ID_0)
- Defining the respective interrupt sources
- NVM Media Start Address
- NVM Media Geometry

The DRV_NVM_Initialize function returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the initialize interface would be used by the other system interfaces, such as DRV_NVM_Deinitialize, DRV_NVM_Status, and DRV_NVM_Tasks.

Note: The system initialization and the reinitialization settings, only affect the instance of the peripheral that is being initialized or reinitialized.

The SYS_MODULE_INDEX is passed to the DRV_NVM_Initialize function to determine which type of memory is selected using:
DRV_NVM_INDEX_0 - FLASH

Example:

```
const DRV_NVM_INIT drvNvmInit =
{
    .moduleInit.sys.powerState = SYS_MODULE_POWER_RUN_FULL,
    .nvmID = NVM_ID_0,
    .interruptSource = INT_SOURCE_FLASH_CONTROL,
    .mediaStartAddress = 0x9D010000,
    .nvmMediaGeometry = (SYS_FS_MEDIA_GEOMETRY *)&NVMGeometry
};
void SYS_Initialize (void *data)
{
.
.
.

   // Initialize NVM Driver Layer
     sysObj.drvNvm = DRV_NVM_Initialize(DRV_NVM_INDEX_0, (SYS_MODULE_INIT *)&drvNvmInit);
.
.
.
}
```

Tasks Routine

The system will call DRV_NVM_Tasks, from system task service (in a polled environment) or DRV_NVM_Tasks will be called from the Interrupt Service Routine (ISR) of the NVM.

Client Access Operation

This section provides information for general client operation.

Description

General Client Operation

For the application to start using an instance of the module, it must call the DRV_NVM_Open function. This provides the configuration required to open the NVM instance for operation. If the driver is deinitialized using the function DRV_NVM_Deinitialize, the application must call the DRV_NVM_Open function again to set up the instance of the NVM.

For the various options available for I/O INTENT please refer to **Data Types and Constants** in the Library Interface section.

Example:

```
DRV_HANDLE handle;
```

```
handle = DRV_NVM_Open(DRV_NVM_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Client Block Data Operation

This topic provides information on client block data operation.

Description

The NVM Driver provides a block interface to access the NVM media. The interface provides functionality to read, write, erase, and erase-write the NVM media. These interface functions depend on the block sizes and boundaries of the individual devices. The interfaces are responsible for keeping this information transparent from the application.

Erasing Data on the NVM:

The following steps outline the sequence for erasing data on the NVM media:

1. The system should have completed necessary initialization and DRV_NVM_Tasks should either be running in a polled environment, or in an interrupt environment.
2. The driver should have been opened with the necessary intent.
3. Provide the block start address and the number of blocks to be erased and begin the erase process using the DRV_NVM_Erase.
4. The client can check the state of the erase request by invoking the DRV_NVM_CommandStatus and passing the command handle returned by the erase request.
5. The client will be able to close itself by calling the DRV_NVM_Close.

Example:

```
// This code shows how to erase NVM Media data
DRV_HANDLE             nvmHandle;
DRV_NVM_COMMAND_HANDLE nvmCommandHandle;

DRV_NVM_COMMAND_STATUS commandStatus;

uint32_t blockAddress;
uint32_t nBlocks;

blockAddress = 0;
nBlocks = 1;

DRV_NVM_Erase(nvmHandle, &nvmCommandHandle, blockAddress, nBlocks);

if(DRV_NVM_COMMAND_HANDLE_INVALID == nvmCommandHandle)
{
    /* Failed to queue the erase request. Handle the error. */
}

// Wait until the command completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
commandStatus = DRV_NVM_CommandStatus(nvmHandle, nvmCommandHandle);
if(DRV_NVM_COMMAND_COMPLETED == commandStatus)
{
    /* Erase completed */
}
else if (DRV_NVM_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Erase Failed */
}
```

Writing Data to the NVM:

The following steps outline the sequence to be followed for writing data to the NVM Media:

1. The system should have completed necessary initialization and DRV_NVM_Tasks should either be running in a polled environment, or in an interrupt environment.
2. The driver should have been opened with the necessary intent.
3. The client should ensure that blocks of addresses to which write is being performed should be in the erased state.
4. Provide the data to be written, block start address and the number of blocks to be written and begin write using the DRV_NVM_Write.

5. The client can check the state of the write request by invoking the DRV_NVM_CommandStatus and passing the command handle returned by the write request.
6. The client will be able to close itself by calling the DRV_NVM_Close.

Example:

```
// This code shows how to write data to NVM Media
DRV_HANDLE              nvmHandle;
DRV_NVM_COMMAND_HANDLE nvmCommandHandle;

DRV_NVM_COMMAND_STATUS commandStatus;

uint8_t writeBuf[DRV_NVM_ROW_SIZE];
uint32_t blockAddress;
uint32_t nBlocks;

blockAddress = 0;
nBlocks = 1;

DRV_NVM_Write(nvmHandle, &nvmCommandHandle, (uint8_t *)writeBuf, blockAddress, nBlocks);

if(DRV_NVM_COMMAND_HANDLE_INVALID == nvmCommandHandle)
{
    /* Failed to queue the write request. Handle the error. */
}

// Wait until the command completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
commandStatus = DRV_NVM_CommandStatus(nvmHandle, nvmCommandHandle);
if(DRV_NVM_COMMAND_COMPLETED == commandStatus)
{
    /* Write completed */
}
else if (DRV_NVM_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Write Failed */
}
```

Reading Data from the NVM:

The following steps outline the sequence to be followed for reading data from the NVM Media:

1. The system should have completed necessary initialization and DRV_NVM_Tasks should either be running in a polled environment, or in an interrupt environment.
2. The driver should have been opened with the necessary intent.
3. Provide the target buffer, block start address and the number of blocks to be read and begin reading using the DRV_NVM_Read.
4. The client can check the state of the read request by invoking the DRV_NVM_CommandStatus and passing the command handle returned by the read request.
5. The client will be able to close itself by calling the DRV_NVM_Close.

Example:

```
// This code shows how to read data from NVM Media
DRV_HANDLE              nvmHandle;
DRV_NVM_COMMAND_HANDLE nvmCommandHandle;

DRV_NVM_COMMAND_STATUS commandStatus;

uint8_t readBuf[DRV_NVM_ROW_SIZE];
uint32_t blockAddress;
uint32_t nBlocks;

blockAddress = 0;
nBlocks = DRV_NVM_ROW_SIZE;

DRV_NVM_Read(nvmHandle, &nvmCommandHandle, (uint8_t *)readBuf, blockAddress, nBlocks);

if(DRV_NVM_COMMAND_HANDLE_INVALID == nvmCommandHandle)
{
    /* Failed to queue the read request. Handle the error. */
}
```

```
// Wait until the command completes. This should not
// be a while loop if a part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.

commandStatus = DRV_NVM_CommandStatus(nvmHandle, nvmCommandHandle);
if(DRV_NVM_COMMAND_COMPLETED == commandStatus)
{
    /* Read completed */
}
else if (DRV_NVM_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Read Failed */
}
```

Configuring the Library

Macros

	Name	Description
	DRV_NVM_BUFFER_OBJECT_NUMBER	Selects the maximum number of buffer objects
	DRV_NVM_CLIENTS_NUMBER	Selects the maximum number of clients
	DRV_NVM_INSTANCES_NUMBER	Selects the maximum number of Driver instances that can be supported by the dynamic driver.
	DRV_NVM_INTERRUPT_MODE	Macro specifies operation of the driver to be in the interrupt mode or polled mode
	DRV_NVM_ROW_SIZE	Specifies the NVM Driver Program Row Size in bytes.
	DRV_NVM_ERASE_WRITE_ENABLE	Enables support for NVM Driver Erase Write Feature.
	DRV_NVM_PAGE_SIZE	Specifies the NVM Driver Program Page Size in bytes.
	DRV_NVM_DISABLE_ERROR_CHECK	Disables the error checks in the driver.
	DRV_NVM_MEDIA_SIZE	Specifies the NVM Media size.
	DRV_NVM_MEDIA_START_ADDRESS	Specifies the NVM Media start address.
	DRV_NVM_SYS_FS_REGISTER	Register to use with the File system

Description

The configuration of the NVM Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the NVM Driver. Based on the selections made, the NVM Driver may support the selected features. These configuration settings will apply to all instances of the NVM Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_NVM_BUFFER_OBJECT_NUMBER Macro

Selects the maximum number of buffer objects

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_BUFFER_OBJECT_NUMBER 5
```

Description

NVM Driver maximum number of buffer objects

This definition selects the maximum number of buffer objects. This indirectly also specifies the queue depth. The NVM Driver can queue up DRV_NVM_BUFFER_OBJECT_NUMBER of read/write/erase requests before return a DRV_NVM_BUFFER_HANDLE_INVALID due to the queue being full. Buffer objects are shared by all instances of the driver. Increasing this number increases the RAM requirement of the driver.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_NVM_CLIENTS_NUMBER Macro

Selects the maximum number of clients

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_CLIENTS_NUMBER 1
```

Description

NVM maximum number of clients

This definition selects the maximum number of clients that the NVM driver can supported at run time. This constant defines the total number of NVM driver clients that will be available to all instances of the NVM driver.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_NVM_INSTANCES_NUMBER Macro

Selects the maximum number of Driver instances that can be supported by the dynamic driver.

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_INSTANCES_NUMBER 1
```

Description

NVM Driver instance configuration

This definition selects the maximum number of Driver instances that can be supported by the dynamic driver. In case of this driver, multiple instances of the driver could use the same hardware instance.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_NVM_INTERRUPT_MODE Macro

Macro specifies operation of the driver to be in the interrupt mode or polled mode

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_INTERRUPT_MODE true
```

Description

NVM interrupt and polled mode operation control

This macro specifies operation of the driver to be in the interrupt mode or polled mode

- true - Select if interrupt mode of NVM operation is desired
- false - Select if polling mode of NVM operation is desired

Not defining this option to true or false will result in build error.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_NVM_ROW_SIZE Macro

Specifies the NVM Driver Program Row Size in bytes.

File

drv_nvm.h

C

```
#define DRV_NVM_ROW_SIZE (NVM_ROW_SIZE)
```

Description

NVM Driver Program Row Size.

This definition specifies the NVM Driver Program Row Size in bytes. This parameter is device specific and is obtained from the device specific processor header file. The Program Row Size is the minimum block size that can be programmed in one program operation.

Remarks

None

DRV_NVM_ERASE_WRITE_ENABLE Macro

Enables support for NVM Driver Erase Write Feature.

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_ERASE_WRITE_ENABLE
```

Description

NVM Driver Erase Write Feature Enable

Specifying this macro enable row erase write feature. If this macro is specified, the drv_nvm_erasewrite.c file should be added in the project. Support for DRV_NVM_EraseWrite() function then gets enabled.

Remarks

This macro is optional and should be specified only if the DRV_NVM_EraseWrite() function is required.

DRV_NVM_PAGE_SIZE Macro

Specifies the NVM Driver Program Page Size in bytes.

File

drv_nvm.h

C

```
#define DRV_NVM_PAGE_SIZE (NVM_PAGE_SIZE)
```

Description

NVM Driver Program Page Size.

This definition specifies the NVM Driver Program Page Size in bytes. This parameter is device specific and is obtained from the device specific processor header file.

Remarks

None

DRV_NVM_DISABLE_ERROR_CHECK Macro

Disables the error checks in the driver.

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_DISABLE_ERROR_CHECK
```

Description

NVM Driver Disable Error Checks

Specifying this macro disables the error checks in the driver. Error checks like parameter validation, NULL checks etc, will be disabled in the driver in order to optimize the code space.

Remarks

This macro is optional and should be specified only if code space is a constraint.

DRV_NVM_MEDIA_SIZE Macro

Specifies the NVM Media size.

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_MEDIA_SIZE 32
```

Description

NVM Media Size

This definition specifies the NVM Media Size to be used. The size is specified in number of Kilo Bytes. The media size MUST never exceed physical available NVM Memory size. Application code requirements should be kept in mind while defining this parameter.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_NVM_MEDIA_START_ADDRESS Macro

Specifies the NVM Media start address.

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_MEDIA_START_ADDRESS 0x9D010000
```

Description

NVM Media Start Address

This definition specifies the NVM Media Start address parameter.

Remarks

This macro is mandatory when building the driver for dynamic operation.

DRV_NVM_SYS_FS_REGISTER Macro

Register to use with the File system

File

drv_nvm_config_template.h

C

```
#define DRV_NVM_SYS_FS_REGISTER
```

Description

NVM Driver Register with File System

Specifying this macro enables the NVM driver to register its services with the SYS FS.

Remarks

This macro is optional and should be specified only if the NVM driver is to be used with the File System.

Building the Library

This section lists the files that are available in the NVM Driver Library.

Description

This section list the files that are available in the `\src` folder of the NVM Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/nvm`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_nvm.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_nvm.c` `/src/dynamic/drv_nvm_erasewrite.c`	Dynamic NVM Driver implementation file. Dynamic NVM Driver Erase/Write implementation file.
`/src/static/drv_nvm_static.c`	Static NVM Driver implementation file for single clients.
`/src/static_multi/drv_nvm_static_multi.c`	Static NVM Driver implementation file for multiple clients.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The NVM Driver Library depends on the following modules:

- Interrupt System Service Library
- Ports System Service Library

Library Interface

a) System Functions

	Name	Description
	DRV_NVM_Initialize	Initializes the NVM instance for the specified driver index **Implementation:** Static/Dynamic
	DRV_NVM_Deinitialize	Deinitializes the specified instance of the NVM driver module **Implementation:** Static/Dynamic
	DRV_NVM_Status	Gets the current status of the NVM driver module. **Implementation:** Static/Dynamic

b) Client Core Functions

	Name	Description
	DRV_NVM_Open	Opens the specified NVM driver instance and returns a handle to it **Implementation:** Static/Dynamic
	DRV_NVM_Close	Closes an opened-instance of the NVM driver **Implementation:** Static/Dynamic

	DRV_NVM_Read	Reads blocks of data from the specified address in memory. **Implementation:** Static/Dynamic
	DRV_NVM_Write	Writes blocks of data starting from the specified address in flash memory. **Implementation:** Static/Dynamic
	DRV_NVM_Erase	Erase the specified number of blocks of the Flash memory. **Implementation:** Static/Dynamic
	DRV_NVM_EraseWrite	Erase and Write blocks of data starting from a specified address in flash memory. **Implementation:** Static/Dynamic
	DRV_NVM_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Static/Dynamic

c) Client Block Data Functions

	Name	Description
	DRV_NVM_Tasks	Maintains the driver's erase and write state machine and implements its ISR. **Implementation:** Static/Dynamic

d) Status Functions

	Name	Description
	DRV_NVM_AddressGet	Returns the NVM media start address **Implementation:** Static/Dynamic
	DRV_NVM_CommandStatus	Gets the current status of the command. **Implementation:** Static/Dynamic
	DRV_NVM_GeometryGet	Returns the geometry of the device. **Implementation:** Static/Dynamic

e) Miscellaneous Functions

	Name	Description
	DRV_NVM_IsAttached	Returns the physical attach status of the NVM. **Implementation:** Static/Dynamic
	DRV_NVM_IsWriteProtected	Returns the write protect status of the NVM. **Implementation:** Static/Dynamic

f) Data Types and Constants

	Name	Description
	DRV_NVM_INDEX_0	NVM driver index definitions
	DRV_NVM_INIT	Defines the data required to initialize or reinitialize the NVM driver
	DRV_NVM_INDEX_1	This is macro DRV_NVM_INDEX_1.
	DRV_NVM_EVENT	Identifies the possible events that can result from a request.
	DRV_NVM_EVENT_HANDLER	Pointer to a NVM Driver Event handler function
	DRV_NVM_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_NVM_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.
	DRV_NVM_COMMAND_HANDLE_INVALID	This value defines the NVM Driver's Invalid Command Handle.
	DRV_NVM_PROGRAM_UNLOCK_KEY1	Specifies the NVM Driver Program Unlock Key 1
	DRV_NVM_PROGRAM_UNLOCK_KEY2	Specifies the NVM Driver Program Unlock Key 2

Description

This section describes the Application Programming Interface (API) functions of the NVM Driver Library.

Refer to each section for a detailed description.

a) System Functions

DRV_NVM_Initialize Function

Initializes the NVM instance for the specified driver index

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
SYS_MODULE_OBJ DRV_NVM_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the NVM driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This routine must be called before any other NVM routine is called.

This routine should only be called once during system initialization unless DRV_NVM_Deinitialize is called to deinitialize the driver instance.

This routine will NEVER block for hardware access. If the operation requires time to allow the hardware to reinitialize, it will be reported by the DRV_NVM_Status operation. The system must use DRV_NVM_Status to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this routine.

Preconditions

None.

Example

```
// This code snippet shows an example
// of initializing the NVM Driver.

SYS_MODULE_OBJ  objectHandle;

SYS_FS_MEDIA_REGION_GEOMETRY gNvmGeometryTable[3] =
{
    {
        // Read Region Geometry
        .blockSize = 1,
        .numBlocks = (DRV_NVM_MEDIA_SIZE * 1024),
    },
    {
        // Write Region Geometry
        .blockSize = DRV_NVM_ROW_SIZE,
        .numBlocks = ((DRV_NVM_MEDIA_SIZE * 1024)/DRV_NVM_ROW_SIZE)
    },
    {
        // Erase Region Geometry
        .blockSize = DRV_NVM_PAGE_SIZE,
        .numBlocks = ((DRV_NVM_MEDIA_SIZE * 1024)/DRV_NVM_PAGE_SIZE)
    }
};

const SYS_FS_MEDIA_GEOMETRY gNvmGeometry =
{
    .mediaProperty = SYS_FS_MEDIA_WRITE_IS_BLOCKING,

    // Number of read, write and erase entries in the table
    .numReadRegions = 1,
    .numWriteRegions = 1,
    .numEraseRegions = 1,
    .geometryTable = &gNvmGeometryTable
};

// FLASH Driver Initialization Data
const DRV_NVM_INIT drvNvmInit =
{
    .moduleInit.sys.powerState = SYS_MODULE_POWER_RUN_FULL,
    .nvmID = NVM_ID_0,
    .interruptSource = INT_SOURCE_FLASH_CONTROL,
```

```
    .mediaStartAddress = NVM_BASE_ADDRESS,
    .nvmMediaGeometry = &gNvmGeometry
};

//usage of DRV_NVM_INDEX_0 indicates usage of Flash-related APIs
objectHandle = DRV_NVM_Initialize(DRV_NVM_INDEX_0, (SYS_MODULE_INIT*)&drvNVMInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized also the type of memory used
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_NVM_Initialize
(
const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init
);

DRV_NVM_Deinitialize Function

Deinitializes the specified instance of the NVM driver module

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the NVM driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Preconditions

Function DRV_NVM_Initialize should have been called before calling this function.

Parameter: object - Driver object handle, returned from the DRV_NVM_Initialize routine

Example

```
// This code snippet shows an example
// of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_NVM_Initialize
SYS_STATUS          status;

DRV_NVM_Deinitialize(object);

status = DRV_NVM_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Function

void DRV_NVM_Deinitialize
(
SYS_MODULE_OBJ object
);

DRV_NVM_Status Function

Gets the current status of the NVM driver module.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
SYS_STATUS DRV_NVM_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and accept requests for new operations.
SYS_STATUS_UNINITIALIZED - Indicates the driver is not initialized.

Description

This routine provides the current status of the NVM driver module.

Remarks

This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_NVM_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_NVM_Initialize
SYS_STATUS          NVMStatus;

NVMStatus = DRV_NVM_Status(object);
else if (SYS_STATUS_ERROR >= NVMStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_NVM_Initialize routine

Function

SYS_STATUS DRV_NVM_Status
(
SYS_MODULE_OBJ object
);

b) Client Core Functions

DRV_NVM_Open Function

Opens the specified NVM driver instance and returns a handle to it

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
DRV_HANDLE DRV_NVM_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, DRV_HANDLE_INVALID is returned. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_NVM_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.
- if the driver hardware instance being opened is not initialized or is invalid

Description

This routine opens the specified NVM driver instance and provides a handle. This handle must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The handle returned is valid until the DRV_NVM_Close routine is called. This routine will NEVER block waiting for hardware. If the driver has has already been opened, it cannot be opened exclusively.

Preconditions

Function DRV_NVM_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_NVM_Open(DRV_NVM_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

```
DRV_HANDLE DRV_NVM_Open
(
const    SYS_MODULE_INDEX index,
const    DRV_IO_INTENT ioIntent
);
```

DRV_NVM_Close Function

Closes an opened-instance of the NVM driver

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_Close(const DRV_HANDLE handle);
```

Returns

None

Description

This routine closes an opened-instance of the NVM driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_NVM_Open before the caller may use the driver again. Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_NVM_Initialize routine must have been called for the specified NVM driver instance.

DRV_NVM_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_NVM_Open

DRV_NVM_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_NVM_Close

(

const DRV_HANDLE handle

);

DRV_NVM_Read Function

Reads blocks of data from the specified address in memory.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_Read(const DRV_HANDLE handle, DRV_NVM_COMMAND_HANDLE * commandHandle, void * targetBuffer,
uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_NVM_COMMAND_HANDLE_INVALID if the request was not successful.

Description

This routine reads blocks of data from the specified address in memory. This operation is blocking and returns with the required data in the target buffer. If an event handler is registered with the driver the event handler would be invoked from within this function to indicate the status of the operation. This function should not be used to read areas of memory which are queued to be programmed or erased. If required, the program or erase operations should be allowed to complete. The function returns DRV_NVM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the driver handle is invalid
- if the target buffer pointer is NULL
- if the number of blocks to be read is zero or more than the actual number of blocks available
- if a buffer object could not be allocated to the request
- if the client opened the driver in write only mode

Remarks

None.

Preconditions

The DRV_NVM_Initialize routine must have been called for the specified NVM driver instance.

DRV_NVM_Open must have been called with DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE as the ioIntent to obtain a valid opened device handle.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = NVM_BASE_ADDRESS_TO_READ_FROM;
uint32_t    nBlock = 2;
DRV_NVM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myNVMHandle is the handle returned
// by the DRV_NVM_Open function.

DRV_NVM_Read(myNVMHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_NVM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}
else
{
    // Read Successful
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
targetBuffer	Buffer into which the data read from the NVM Flash instance will be placed
blockStart	Start block address in NVM memory from where the read should begin. It can be any address of the flash.
nBlock	Total number of blocks to be read. Each Read block is of 1 byte.

Function

void DRV_NVM_Read

(

const DRV_HANDLE handle,

DRV_NVM_COMMAND_HANDLE * commandHandle,

void * targetBuffer,

uint32_t blockStart,

uint32_t nBlock

);

DRV_NVM_Write Function

Writes blocks of data starting from the specified address in flash memory.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_Write(const DRV_HANDLE handle, DRV_NVM_COMMAND_HANDLE * commandHandle, void * sourceBuffer,
uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_NVM_COMMAND_HANDLE_INVALID if the request was not

successful.

Description

This function schedules a non-blocking write operation for writing blocks of data into flash memory. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_NVM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer object could not be allocated to the request
- if the source buffer pointer is NULL
- if the client opened the driver for read only
- if the number of blocks to be written is either zero or more than the number of blocks actually available
- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_NVM_EVENT_COMMAND_COMPLETE event if the buffer was processed successfully or DRV_NVM_EVENT_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

Performing a flash programming operation while executing (fetching) instructions from program Flash memory, the CPU stalls (waits) until the programming operation is finished. The CPU will not execute any instruction, or respond to interrupts, during this time. If any interrupts occur during the programming cycle, they remain pending until the cycle completes. This makes the NVM write operation blocking in nature.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

DRV_NVM_Open() routine must have been called to obtain a valid opened device handle. DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified as a parameter to this routine.

The flash address location which has to be written, must have be erased before using the DRV_NVM_Erase() routine.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = NVM_BASE_ADDRESS_TO_WRITE_TO;
uint32_t    nBlock = 2;
DRV_NVM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myNVMHandle is the handle returned
// by the DRV_NVM_Open function.

// Client registers an event handler with driver

DRV_NVM_EventHandlerSet(myNVMHandle, APP_NVMEventHandler, (uintptr_t)&myAppObj);

DRV_NVM_Write(myNVMHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_NVM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_NVMEventHandler(DRV_NVM_EVENT event,
        DRV_NVM_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_NVM_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_NVM_EVENT_COMMAND_ERROR:
```

```
            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
sourceBuffer	The source buffer containing data to be programmed into NVM Flash
blockStart	Start block address of NVM Flash where the write should begin. This address should be aligned on a block boundary.
nBlock	Total number of blocks to be written.

Function

void DRV_NVM_Write

(

const DRV_HANDLE handle,

DRV_NVM_COMMAND_HANDLE * commandHandle,

void * sourceBuffer,

uint32_t blockStart,

uint32_t nBlock

);

DRV_NVM_Erase Function

Erase the specified number of blocks of the Flash memory.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_Erase(const DRV_HANDLE handle, DRV_NVM_COMMAND_HANDLE * commandHandle, uint32_t blockStart,
uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It Will be DRV_NVM_COMMAND_HANDLE_INVALID if the request was not queued.

Description

This function schedules a non-blocking erase operation of flash memory. The function returns with a valid erase handle in the commandHandle argument if the erase request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_NVM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer object could not be allocated to the request
- if the client opened the driver for read only
- if the number of blocks to be erased is either zero or more than the number of blocks actually available
- if the erase queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_NVM_EVENT_COMMAND_COMPLETE event if the erase operation was successful or DRV_NVM_EVENT_COMMAND_ERROR event if the erase operation was not successful.

Remarks

Performing a flash erase operation while executing (fetching) instructions from program Flash memory, the CPU stalls (waits) until the erase operation is finished. The CPU will not execute any instruction, or respond to interrupts, during this time. If any interrupts occur during the programming cycle, they remain pending until the cycle completes. This make the NVM erase operation blocking in nature.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() routine must have been called with DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE to obtain a valid opened device handle.

Example

```
// Destination address should be block aligned.
uint32_t blockStart;
uint32_t nBlock;
DRV_NVM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myNVMHandle is the handle returned
// by the DRV_NVM_Open function.

// Client registers an event handler with driver

DRV_NVM_EventHandlerSet(myNVMHandle, APP_NVMEventHandler, (uintptr_t)&myAppObj);

DRV_NVM_Erase( myNVMHandle, &commandHandle, blockStart, nBlock );

if(DRV_NVM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer queue is processed.

void APP_NVMEventHandler(DRV_NVM_EVENT event,
        DRV_NVM_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_NVM_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_NVM_EVENT_COMMAND_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
blockStart	Start block address in NVM memory from where the erase should begin. This should be aligned on a DRV_NVM_PAGE_SIZE byte boundary.
nBlock	Total number of blocks to be erased.

Function

void DRV_NVM_Erase

(

const DRV_HANDLE handle,

DRV_NVM_COMMAND_HANDLE * commandHandle,

uint32_t blockStart,
uint32_t nBlock
);

DRV_NVM_EraseWrite Function

Erase and Write blocks of data starting from a specified address in flash memory.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_EraseWrite(const DRV_HANDLE handle, DRV_NVM_COMMAND_HANDLE * commandHandle, void *
sourceBuffer, uint32_t writeBlockStart, uint32_t nWriteBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It Will be DRV_NVM_COMMAND_HANDLE_INVALID if the request was not queued.

Description

This function combines the step of erasing a page and then writing the row. The application can use this function if it wants to avoid having to explicitly delete a page in order to update the rows contained in the page.

This function schedules a non-blocking operation to erase and write blocks of data into flash memory. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_NVM_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only
- if the buffer size is 0
- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_NVM_EVENT_COMMAND_COMPLETE event if the buffer was processed successfully or DRV_NVM_EVENT_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

In order to use this function, the DRV_NVM_ERASE_WRITE_ENABLE must be defined in system_config.h and the drv_nvm_erasewrite.c file must be included in the project.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() must have been called with DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE as a parameter to obtain a valid opened device handle.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = NVM_BASE_ADDRESS_TO_WRITE_TO;
uint32_t    nBlock = 2;
DRV_NVM_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// myNVMHandle is the handle returned
// by the DRV_NVM_Open function.

// Client registers an event handler with driver

DRV_NVM_EventHandlerSet(myNVMHandle, APP_NVMEventHandler, (uintptr_t)&myAppObj);

DRV_NVM_EraseWrite(myNVMHandle, &commandHandle, &myBuffer, blockStart, nBlock);
```

```
if(DRV_NVM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_NVMEventHandler(DRV_NVM_EVENT event,
        DRV_NVM_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_NVM_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_NVM_EVENT_COMMAND_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle. If NULL, then buffer handle is not returned.
sourceBuffer	The source buffer containing data to be programmed into NVM Flash
writeBlockStart	Start block address of NVM Flash where the write should begin. This address should be aligned on a DRV_NVM_ROW_SIZE byte boundary.
nWriteBlock	Total number of blocks to be written.

Function

void DRV_NVM_EraseWrite
(
const DRV_HANDLE handle,
DRV_NVM_COMMAND_HANDLE * commandHandle,
void * sourceBuffer,
uint32_t writeBlockStart,
uint32_t nWriteBlock
);

DRV_NVM_EventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_EventHandlerSet(const DRV_HANDLE handle, const void * eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client calls a write or erase function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any write or erase operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_NVM_COMMAND_HANDLE commandHandle;

// drvNVMHandle is the handle returned
// by the DRV_NVM_Open function.

// Client registers an event handler with driver. This is done once.

DRV_NVM_EventHandlerSet(drvNVMHandle, APP_NVMEventHandler, (uintptr_t)&myAppObj);

DRV_NVM_Read(drvNVMHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_NVM_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_NVMEventHandler(DRV_NVM_EVENT event,
        DRV_NVM_COMMAND_HANDLE handle, uintptr_t context)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_NVM_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_NVM_EVENT_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_NVM_EventHandlerSet
(
const DRV_HANDLE handle,
const void * eventHandler,
const uintptr_t context
);

c) Client Block Data Functions

DRV_NVM_Tasks Function

Maintains the driver's erase and write state machine and implements its ISR.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
void DRV_NVM_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal write and erase state machine and implement its ISR for interrupt-driven implementations.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_NVM_Initialize routine must have been called for the specified NVM driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_NVM_Initialize

while (true)
{
    DRV_NVM_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_NVM_Initialize)

Function

void DRV_NVM_Tasks

(
SYS_MODULE_OBJ object
);

d) Status Functions

DRV_NVM_AddressGet Function

Returns the NVM media start address

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
uintptr_t DRV_NVM_AddressGet(const DRV_HANDLE handle);
```

Returns

Start address of the NVM Media if the handle is valid otherwise NULL.

Description

This function returns the NVM Media start address.

Remarks

None.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
uintptr_t startAddress;
startAddress = DRV_NVM_AddressGet(drvNVMHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

uintptr_t DRV_NVM_AddressGet
(
const DRV_HANDLE handle
);

DRV_NVM_CommandStatus Function

Gets the current status of the command.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
DRV_NVM_COMMAND_STATUS DRV_NVM_CommandStatus(const DRV_HANDLE handle, const DRV_NVM_COMMAND_HANDLE
commandHandle);
```

Returns

A DRV_NVM_COMMAND_STATUS value describing the current status of the command. Returns DRV_NVM_COMMAND_HANDLE_INVALID if

the client handle or the command handle is not valid.

Description

This routine gets the current status of the command. The application must use this routine where the status of a scheduled command needs to polled on. The function may return DRV_NVM_COMMAND_HANDLE_INVALID in a case where the command handle has expired. A command handle expires when the internal buffer object is re-assigned to another erase or write request. It is recommended that this function be called regularly in order to track the command status correctly.

The application can alternatively register an event handler to receive write or erase operation completion events.

Remarks

This routine will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_NVM_Initialize() routine must have been called.

The DRV_NVM_Open() must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE                  handle;         // Returned from DRV_NVM_Open
DRV_NVM_COMMAND_HANDLE      commandHandle;
DRV_NVM_COMMAND_STATUS      status;

status = DRV_NVM_CommandStatus(handle, commandHandle);
if(status == DRV_NVM_COMMAND_COMPLETED)
{
    // Operation Done
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

DRV_NVM_COMMAND_STATUS DRV_NVM_CommandStatus

(

const DRV_HANDLE handle,

const DRV_NVM_COMMAND_HANDLE commandHandle

);

DRV_NVM_GeometryGet Function

Returns the geometry of the device.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_NVM_GeometryGet(const DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Pointer to structure which holds the media geometry information.

Description

This API gives the following geometrical details of the NVM Flash:

- Media Property
- Number of Read/Write/Erase regions in the flash device
- Number of Blocks and their size in each region of the device

Remarks

None.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
SYS_FS_MEDIA_GEOMETRY * nvmFlashGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalFlashSize;

nvmFlashGeometry = DRV_NVM_GeometryGet(nvmOpenHandle1);

readBlockSize  = nvmFlashGeometry->geometryTable->blockSize;
nReadBlocks = nvmFlashGeometry->geometryTable->numBlocks;
nReadRegions = nvmFlashGeometry->numReadRegions;

writeBlockSize  = (nvmFlashGeometry->geometryTable +1)->blockSize;
eraseBlockSize  = (nvmFlashGeometry->geometryTable +2)->blockSize;

totalFlashSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY * DRV_NVM_GeometryGet

(

const DRV_HANDLE handle

);

e) Miscellaneous Functions

DRV_NVM_IsAttached Function

Returns the physical attach status of the NVM.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
bool DRV_NVM_IsAttached(const DRV_HANDLE handle);
```

Returns

Returns false if the handle is invalid otherwise returns true.

Description

This function returns the physical attach status of the NVM.

Remarks

None.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// The NVM media is always attached and so the below
// always returns true.
```

```
bool isNVMAttached;
isNVMAttached = DRV_NVM_isAttached(drvNVMHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_NVM_IsAttached

(

const DRV_HANDLE handle

);

DRV_NVM_IsWriteProtected Function

Returns the write protect status of the NVM.

Implementation: Static/Dynamic

File

drv_nvm.h

C

```
bool DRV_NVM_IsWriteProtected(const DRV_HANDLE handle);
```

Returns

Always returns false.

Description

This function returns the physical attach status of the NVM. This function always returns false.

Remarks

None.

Preconditions

The DRV_NVM_Initialize() routine must have been called for the specified NVM driver instance.

The DRV_NVM_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// The NVM media is treated as always writeable.
bool isWriteProtected;
isWriteProtected = DRV_NVM_IsWriteProtected(drvNVMHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_NVM_IsWriteProtected

(

const DRV_HANDLE handle

);

f) Data Types and Constants

DRV_NVM_INDEX_0 Macro

NVM driver index definitions

File

drv_nvm.h

C

```
#define DRV_NVM_INDEX_0 0
```

Description

Driver NVM Module Index reference

These constants provide NVM driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_NVM_Initialize and DRV_NVM_Open routines to identify the driver instance in use.

DRV_NVM_INIT Structure

Defines the data required to initialize or reinitialize the NVM driver

File

drv_nvm.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  NVM_MODULE_ID nvmID;
  INT_SOURCE interruptSource;
  uint32_t mediaStartAddress;
  const SYS_FS_MEDIA_GEOMETRY * nvmMediaGeometry;
} DRV_NVM_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
NVM_MODULE_ID nvmID;	Identifies NVM hardware module (PLIB-level) ID
INT_SOURCE interruptSource;	Interrupt Source for Write Interrupt
uint32_t mediaStartAddress;	NVM Media start address. The driver treats this address as • block 0 address for read, write and erase operations.
const SYS_FS_MEDIA_GEOMETRY * nvmMediaGeometry;	NVM Media geometry object.

Description

NVM Driver Initialization Data

This data type defines the data required to initialize or reinitialize the NVM driver.

Remarks

Not all initialization features are available for all devices. Please refer to the specific device data sheet to determine availability.

DRV_NVM_INDEX_1 Macro

File

drv_nvm.h

C

```
#define DRV_NVM_INDEX_1 1
```

Description

This is macro DRV_NVM_INDEX_1.

DRV_NVM_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_nvm.h

C

```
typedef enum {
  DRV_NVM_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_NVM_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR
} DRV_NVM_EVENT;
```

Members

Members	Description
DRV_NVM_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE	Operation has been completed successfully.
DRV_NVM_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR	There was an error during the operation

Description

NVM Driver Events

This enumeration identifies the possible events that can result from a Write or Erase request caused by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_NVM_EventHandlerSet function when a request is completed.

DRV_NVM_EVENT_HANDLER Type

Pointer to a NVM Driver Event handler function

File

drv_nvm.h

C

```
typedef SYS_FS_MEDIA_EVENT_HANDLER DRV_NVM_EVENT_HANDLER;
```

Returns

None.

Description

NVM Driver Event Handler Function Pointer

This data type defines the required function signature for the NVM event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_NVM_EVENT_COMMAND_COMPLETE, it means that the write or a erase operation was completed successfully.

If the event is DRV_NVM_EVENT_COMMAND_ERROR, it means that the scheduled operation was not completed successfully.

The context parameter contains the handle to the client context, provided at the time the event handling function was registered using the DRV_NVM_EventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the read/write/erase request.

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations within this function.

Example

```
void APP_MyNvmEventHandler
(
```

```
    DRV_NVM_EVENT event,
    DRV_NVM_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_NVM_EVENT_COMMAND_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_NVM_EVENT_COMMAND_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle returned from the Read/Write/Erase requests
context	Value identifying the context of the application that registered the event handling function

DRV_NVM_COMMAND_HANDLE Type

Handle identifying commands queued in the driver.

File

drv_nvm.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_NVM_COMMAND_HANDLE;
```

Description

NVM Driver command handle.

A command handle is returned by a call to the Read, Write or Erase functions. This handle allows the application to track the completion of the operation. This command handle is also returned to the client along with the event that has occurred with respect to the command. This allows the application to connect the event to a specific command in case where multiple commands are queued.

The command handle associated with the command request expires when the client has been notified of the completion of the command (after event handler function that notifies the client returns) or after the command has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_NVM_COMMAND_STATUS Enumeration

Specifies the status of the command for the read, write and erase operations.

File

drv_nvm.h

C

```
typedef enum {
  DRV_NVM_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED,
  DRV_NVM_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED,
  DRV_NVM_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS,
  DRV_NVM_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN
} DRV_NVM_COMMAND_STATUS;
```

Members

Members	Description
DRV_NVM_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED	Done OK and ready
DRV_NVM_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED	Scheduled but not started
DRV_NVM_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS	Currently being in transfer
DRV_NVM_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN	Unknown Command

Description

NVM Driver Command Status

NVM Driver command Status

This type specifies the status of the command for the read, write and erase operations.

Remarks

None.

DRV_NVM_COMMAND_HANDLE_INVALID Macro

This value defines the NVM Driver's Invalid Command Handle.

File

drv_nvm.h

C

```
#define DRV_NVM_COMMAND_HANDLE_INVALID SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE_INVALID
```

Description

NVM Driver Invalid Command Handle.

This value defines the NVM Driver Invalid Command Handle. This value is returned by read/write/erase routines when the command request was not accepted.

Remarks

None.

DRV_NVM_PROGRAM_UNLOCK_KEY1 Macro

Specifies the NVM Driver Program Unlock Key 1

File

drv_nvm.h

C

```
#define DRV_NVM_PROGRAM_UNLOCK_KEY1 (NVM_UNLOCK_KEY1)
```

Description

NVM Driver Program Unlock Key 1

This definition specifies the NVM Driver Program Unlock Key 1 parameter is device specific and is obtained from the device specific processor header file.

Remarks

None

DRV_NVM_PROGRAM_UNLOCK_KEY2 Macro

Specifies the NVM Driver Program Unlock Key 2

File

drv_nvm.h

C

```
#define DRV_NVM_PROGRAM_UNLOCK_KEY2 (NVM_UNLOCK_KEY2)
```

Description

NVM Driver Program Unlock Key 2

This definition specifies the NVM Driver Program Unlock Key 2 parameter is device specific and is obtained from the device specific processor header file.

Remarks

None

Files

Files

Name	Description
drv_nvm.h	NVM Driver Interface Definition
drv_nvm_config_template.h	NVM driver configuration definitions.

Description

This section lists the source and header files used by the NVM Driver Library.

drv_nvm.h

NVM Driver Interface Definition

Enumerations

	Name	Description
	DRV_NVM_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.
	DRV_NVM_EVENT	Identifies the possible events that can result from a request.

Functions

	Name	Description
	DRV_NVM_AddressGet	Returns the NVM media start address **Implementation:** Static/Dynamic
	DRV_NVM_Close	Closes an opened-instance of the NVM driver **Implementation:** Static/Dynamic
	DRV_NVM_CommandStatus	Gets the current status of the command. **Implementation:** Static/Dynamic
	DRV_NVM_Deinitialize	Deinitializes the specified instance of the NVM driver module **Implementation:** Static/Dynamic
	DRV_NVM_Erase	Erase the specified number of blocks of the Flash memory. **Implementation:** Static/Dynamic
	DRV_NVM_EraseWrite	Erase and Write blocks of data starting from a specified address in flash memory. **Implementation:** Static/Dynamic
	DRV_NVM_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Static/Dynamic
	DRV_NVM_GeometryGet	Returns the geometry of the device. **Implementation:** Static/Dynamic
	DRV_NVM_Initialize	Initializes the NVM instance for the specified driver index **Implementation:** Static/Dynamic
	DRV_NVM_IsAttached	Returns the physical attach status of the NVM. **Implementation:** Static/Dynamic
	DRV_NVM_IsWriteProtected	Returns the write protect status of the NVM. **Implementation:** Static/Dynamic

	DRV_NVM_Open	Opens the specified NVM driver instance and returns a handle to it **Implementation:** Static/Dynamic
	DRV_NVM_Read	Reads blocks of data from the specified address in memory. **Implementation:** Static/Dynamic
	DRV_NVM_Status	Gets the current status of the NVM driver module. **Implementation:** Static/Dynamic
	DRV_NVM_Tasks	Maintains the driver's erase and write state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_NVM_Write	Writes blocks of data starting from the specified address in flash memory. **Implementation:** Static/Dynamic

Macros

	Name	Description
	DRV_NVM_COMMAND_HANDLE_INVALID	This value defines the NVM Driver's Invalid Command Handle.
	DRV_NVM_INDEX_0	NVM driver index definitions
	DRV_NVM_INDEX_1	This is macro DRV_NVM_INDEX_1.
	DRV_NVM_PAGE_SIZE	Specifies the NVM Driver Program Page Size in bytes.
	DRV_NVM_PROGRAM_UNLOCK_KEY1	Specifies the NVM Driver Program Unlock Key 1
	DRV_NVM_PROGRAM_UNLOCK_KEY2	Specifies the NVM Driver Program Unlock Key 2
	DRV_NVM_ROW_SIZE	Specifies the NVM Driver Program Row Size in bytes.

Structures

	Name	Description
	DRV_NVM_INIT	Defines the data required to initialize or reinitialize the NVM driver

Types

	Name	Description
	DRV_NVM_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_NVM_EVENT_HANDLER	Pointer to a NVM Driver Event handler function

Description

NVM Driver Interface Definition

The NVM driver provides a simple interface to manage the Non Volatile Flash Memory on Microchip microcontrollers. This file defines the interface definition for the NVM driver.

File Name

drv_nvm.h

Company

Microchip Technology Inc.

drv_nvm_config_template.h

NVM driver configuration definitions.

Macros

	Name	Description
	DRV_NVM_BUFFER_OBJECT_NUMBER	Selects the maximum number of buffer objects
	DRV_NVM_CLIENTS_NUMBER	Selects the maximum number of clients
	DRV_NVM_DISABLE_ERROR_CHECK	Disables the error checks in the driver.
	DRV_NVM_ERASE_WRITE_ENABLE	Enables support for NVM Driver Erase Write Feature.
	DRV_NVM_INSTANCES_NUMBER	Selects the maximum number of Driver instances that can be supported by the dynamic driver.
	DRV_NVM_INTERRUPT_MODE	Macro specifies operation of the driver to be in the interrupt mode or polled mode
	DRV_NVM_MEDIA_SIZE	Specifies the NVM Media size.
	DRV_NVM_MEDIA_START_ADDRESS	Specifies the NVM Media start address.
	DRV_NVM_SYS_FS_REGISTER	Register to use with the File system

Description

NVM Driver Configuration Template Header file.

This template file describes all the mandatory and optional configuration macros that are needed for building the NVM driver. Do not include this file in source code.

File Name

drv_nvm_config_template.h

Company

Microchip Technology Inc.

Output Compare Driver Library

This section describes the Output Compare Driver Library.

Introduction

The Output Compare Static Driver provides a high-level interface to manage the Output Compare module on the Microchip family of microcontrollers.

Description

Through the MHC, this driver provides APIs for the following:

- Initializing the module
- Enabling/Disabling of the output compare
- Starting/Stopping of the output compare
- Fault checking

Library Interface

Functions

	Name	Description
	DRV_OC_Disable	Disables the Output Compare instance for the specified driver index. **Implementation:** Static
	DRV_OC_Enable	Enables the Output Compare for the specified driver index. **Implementation:** Static
	DRV_OC_FaultHasOccurred	Checks if a Fault has occurred for the specified driver index. **Implementation:** Static
	DRV_OC_Initialize	Initializes the Comparator instance for the specified driver index. **Implementation:** Static
	DRV_OC_Start	Starts the Comparator instance for the specified driver index. **Implementation:** Static
	DRV_OC_Stop	Stops the Output Compare instance for the specified driver index. **Implementation:** Static

Description

This section describes the Application Programming Interface (API) functions of the Output Compare Driver Library.

Functions

DRV_OC_Disable Function

Disables the Output Compare instance for the specified driver index.

Implementation: Static

File

help_drv_oc.h

C

```
void DRV_OC_Disable();
```

Returns

None.

Description

This routine disables the Output Compare for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

None.

Preconditions

DRV_OC_Initialize has been called.

Function

void DRV_OC_Disable(void)

DRV_OC_Enable Function

Enables the Output Compare for the specified driver index.

Implementation: Static

File

help_drv_oc.h

C

```
void DRV_OC_Enable();
```

Returns

None.

Description

This routine enables the Output Compare for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

None.

Preconditions

DRV_OC_Initialize has been called.

Function

void DRV_OC_Enable(void)

DRV_OC_FaultHasOccurred Function

Checks if a Fault has occurred for the specified driver index.

Implementation: Static

File

help_drv_oc.h

C

```
bool DRV_OC_FaultHasOccurred();
```

Returns

Boolean

- 1 - A Fault has occurred
- 0 - A Fault has not occurred

Description

This routine checks whether or not a Fault has occurred for the specified driver index. The initialization routine is specified by the MHC parameters.

Remarks

None.

Preconditions

DRV_OC_Initialize has been called.

Function

bool DRV_OC_FaultHasOccurred(void)

DRV_OC_Initialize Function

Initializes the Comparator instance for the specified driver index.

Implementation: Static

File

help_drv_oc.h

C

```
void DRV_OC_Initialize();
```

Returns

None.

Description

This routine initializes the Output Compare driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters. The driver instance index is independent of the Output Compare module ID. For example, driver instance 0 can be assigned to Output Compare 1.

Remarks

This routine must be called before any other Comparator routine is called. This routine should only be called once during system initialization.

Preconditions

None.

Function

void DRV_OC_Initialize(void)

DRV_OC_Start Function

Starts the Comparator instance for the specified driver index.

Implementation: Static

File

help_drv_oc.h

C

```
void DRV_OC_Start();
```

Returns

None.

Description

This routine starts the Output Compare for the specified driver instance.

Remarks

None.

Preconditions

DRV_OC_Initialize has been called.

Function

void DRV_OC_Start(void)

DRV_OC_Stop Function

Stops the Output Compare instance for the specified driver index.

Implementation: Static

File

help_drv_oc.h

C

```
void DRV_OC_Stop();
```

Returns

None.

Description

This routine stops the Output Compare for the specified driver instance.

Remarks

None.

Preconditions

DRV_OC_Initialize has been called.

Function

void DRV_OC_Stop(void)

Parallel Master Port (PMP) Driver Library

This section describes the Parallel Master Port Driver Library.

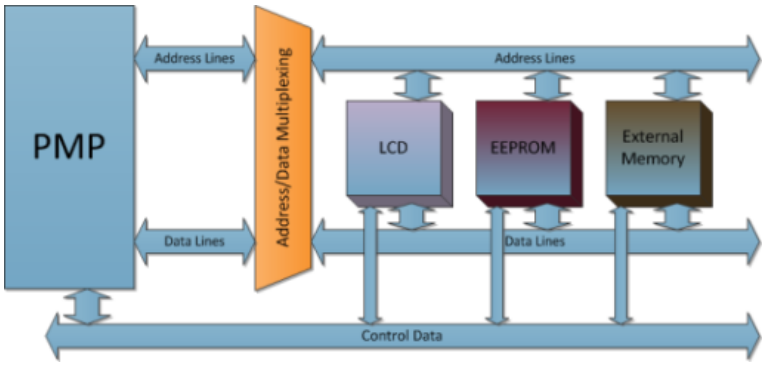
Introduction

This library provides an interface to manage the Parallel Master Port (PMP) module on Microchip family of microcontrollers in different modes of operation.

Description

The Parallel Master Port (PMP) is a parallel 8-bit/16-bit I/O module specifically designed to communicate with a wide variety of parallel devices such as communications peripherals, LCDs, external memory devices and microcontrollers. Because the interfaces to parallel peripherals vary significantly, the PMP module is highly configurable.

The following figure shows a generic block diagram, which illustrates the ways the PMP module can be used:



The PMP module can be used in different modes. Master and Slave are the two modes that can have additional sub-modes, depending on the different microcontroller families.

Master Mode: In Master mode, the PMP module can provide a 8-bit or 16-bit data bus, up to 16 bits of address, and all of the necessary control signals to operate a variety of external parallel devices such as memory devices, peripherals and slave microcontrollers. The PMP master modes provide a simple interface for reading and writing data, but not executing program instructions from external devices, such as SRAM or Flash memories.

Slave Mode: Slave mode only supports 8-bit data and the module control pins are automatically dedicated when this mode is selected.

Using the Library

This topic describes the basic architecture of the PMP Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_pmp.h`

The interface to the PMP Driver library is defined in the `drv_pmp.h` header file. This file is included by the `drv.h` file. Any C language source (`.c`) file that uses the PMP Driver Library should include `drv.h`.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

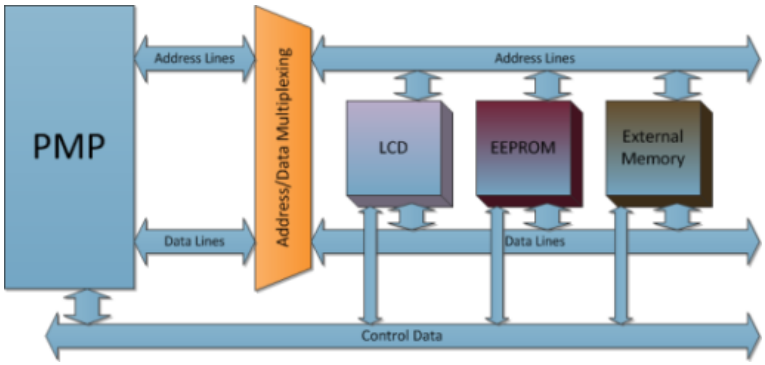
This library provides a low-level abstraction of the Parallel Master Port (PMP) module on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model Description

Depending on the device,the PMP module provides interface routines to interact with external peripherals such as LCD, EEPROM, Flash memory, etc., as shown in the following diagram. The diagram shows the PMP module acting as a master. The PMP module can be easily configured to act as a slave. The address and data lines can be multiplexed to suit the application. The address and data buffers are up to 2-byte (16-bit) buffers for data transmitted or received by the parallel interface to the PMP bus over the data and address lines synchronized with control logic including the read and write strobe.

The desired timing wait states to suit different peripheral timings can also be programmed using the PMP module.



PMP Hardware Abstraction Model Diagram

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the PMP module.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks, and status functions.
Client Interaction Functions	Provides open, close, client status and client mode configuration functions.
Client Transfer Functions	Provides interface for data transfer in master and slave mode.
Miscellaneous	Provides driver miscellaneous functions, version identification functions, etc.

How the Library Works

This section describes how the PMP Driver Library operates.

Description

Before the driver is ready for use, its should be configured (compile time configuration). Refer to the Configuring the Library section for more details on how to configure the driver.

There are few run-time configuration items that are done during initialization of the driver instance, and a few that are client-specific and are done using dedicated functions.

To use the PMP Driver, initialization and client functions should be invoked in a specific sequence to ensure correct operation.

The following is the sequence in which various routines should be called:

1. Call DRV_PMP_Initialize to initialize the PMP Driver. Note that this may be performed by the MPLAB Harmony system module. The DRV_PMP_Status function may be used to check the status of the initialization.
2. Once initialization for a particular driver instance is done, the client wanting to use the driver can open it using DRV_PMP_Open.
3. The DRV_PMP_ModeConfig function should now be called, which will configure the driver for the exact mode of operation required by that client.
4. After configuring the mode, DRV_PMP_Write and/or DRV_PMP_Read can be called by the user application to Write/Read using the PMP module. Calling these functions does not start the PMP transfer immediately in non-interrupt mode. Instead, all of these transfer tasks are queued in an internal queue. Actual transfer starts only when the PMP Task function is called by the system/user. In interrupt mode, although transfer tasks are queued, the actual transfer starts immediately.
5. PMP Write and Read functions return an ID of that particular transfer, which should be saved by user to get the status of that transfer later.
6. The system will either call DRV_PMP_Tasks from the System Task Service (in a polled environment), or it will be called from the ISR of the PMP.
7. At any time status of the transfer can be obtained by using DRV_PMP_TransferStatus.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

System Initialization

This section describes initialization and reinitialization features.

Description

Initialization and Reinitialization

The system performs the initialization and the reinitialization of the device driver with settings that affect only the instance of the device that is being initialized or reinitialized. During system initialization each instance of the PMP device will be initialized with the following configuration settings:

Initialization Member	Description
`moduleInit`	System module initialization of the power state.
`pmpId`	PMP hardware module ID (peripheral library-level ID).
`stopInIdle`	Decide whether or not the module should be stopped in Idle mode.
`muxMode`	To select one of the different multiplexing modes possible for PMP module.
`inputBuffer`	Select the type of Input Buffer (TTL or Schmitt Trigger).
`polarity`	Select polarity of different PMP pins.
`ports`	Set the pins the user wants to use as port or PMP pins.

The DRV_PMP_Initialize function returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the initialize interface would be used by the other system interfaces, such as DRV_PMP_Reinitialize, DRV_PMP_Deinitialize, DRV_PMP_Status, and DRV_PMP_Tasks.

Example for PMP Initialization Through the DRV_PMP_INIT Structure

```
DRV_PMP_INIT     init;
SYS_MODULE_OBJ   object;
SYS_STATUS       pmpStatus;

// populate the PMP init configuration structure
    init.inputBuffer = PMP_INPUT_BUFFER_TTL;
    init.polarity.addressLatchPolarity = PMP_POLARITY_ACTIVE_HIGH;
    init.polarity.rwStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
    init.polarity.writeEnableStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
    init.polarity.chipselect1Polarity = PMP_POLARITY_ACTIVE_HIGH;
    init.polarity.chipselect2Polarity = PMP_POLARITY_ACTIVE_LOW;
    init.ports.addressPortsMask = PMP_PMA0_PORT | PMP_PMA1_PORT | PMP_PMA2_TO_PMA13_PORTS | PMP_PMA14_PORT;
    init.ports.readWriteStrobe = PORT_ENABLE;
```

```
    init.ports.writeEnableStrobe = PORT_ENABLE;
    init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
    init.pmpID            = PMP_ID_0;
    init.stopInIdle       = false;
    init.muxMode          = PMP_MUX_NONE;

object = DRV_PMP_Initialize (DRV_PMP_INDEX_0, (SYS_MODULE_INIT *)&init);

pmpStatus = DRV_PMP_Status(object);

if ( SYS_STATUS_READY != pmpStatus)
{
    // Handle error
}
```

Deinitialization

Once the initialize operation has been called, the deinitialize operation must be called before the initialize operation can be called again. This routine may block if the driver is running in an OS environment that supports blocking operations and the driver requires system resources access. However, the function will never block for hardware PMP access. If the operation requires time to allow the hardware to complete, which will be reported by DRV_PMP_Status.

Status

PMP status is available to query the module state before, during and after initialization, deinitialization, and reinitialization.

Tasks Routine

The DRV_PMP_Tasks function will see the queue status and perform the task of transferring the data accordingly. In the Blocking mode when interrupts are disabled, it will finish one of the tasks completely (that means emptying one space in queue), and then return back. Whereas in Non-Blocking mode, it will return back just after starting one word (8-bit or 16-bit) of transfer (may not be emptying one space in the queue, as that task may not be completely finished).

The DRV_PMP_Tasks function can be called in two ways:

- By the system task service in a polled environment
- By the ISR of the PMP in an interrupt-based system

Example: Polling

```
int main( void )
{
    SYS_MODULE_OBJ object;
    object = DRV_PMP_Initialize( DRV_PMP_INDEX_0, (SYS_MODULE_INIT *) &initConf );

    if( SYS_STATUS_READY != DRV_PMP_Status( object ) )
        return 0;

    while (1)
    {
        DRV_PMP_Tasks (object);
    }
}
```

Example: Interrupt

```
int main( void )
{
    SYS_MODULE_OBJ object;
    object = DRV_PMP_Initialize( DRV_PMP_INDEX_0, (SYS_MODULE_INIT *) &initConf );

    if( SYS_STATUS_READY != DRV_PMP_Status( object ) )
        return 0;

    while (1);
}

/* Sample interrupt routine not specific to any device family */
void ISR PMPInterrupt(void)
{
    //Call the PMP Tasks routine
    DRV_PMP_Tasks(object);
}
```

Note: A PMP transfer in Blocking mode in an interrupt environment is not supported.

Transfer Operation

This section describes transfer operation.

Description

Once the PMP Driver is open and configured for a client, it is set to start Reading/Writing through DRV_PMP_Read and DRV_PMP_Write. However, these functions will not directly start reading or writing. These will just put the relevant information in a queue in non-interrupt mode and return an ID that can be used later for checking the transfer status. In Interrupt mode, the Read/Write functions will trigger the transfer immediately after storing the transfer information in the queue.

The user must use a buffer pointing to character for data values.

The repeatCount parameter allows the user to repeatedly write the same nBytes of data into the slave devices.

Example:

```
unsigned char myReadBuffer[300], myWriteBuffer[100]; // has to be 'char' arrays
uint32_t deviceAddress, nBytes, repeatCount, i;
uint32_t writeID, readID;
DRV_HANDLE handle;

//initialize, open and configure the driver/client
/* ... */

deviceAddress = 0x0206;
nBytes = 100;
repeatCount = 0x01;
for (i=0; i<nBytes; i++)
{
   myWriteBuffer[i]=i*5+7;
}

/* it will write 100 bytes of data in the location starting from 0x0206 and then it will repeat
 writing the same set of data in next 100 location starting from 0x206+100 for 8 bit data mode
 and 50 location starting from 0x206+50 for 16 bit data mode. */
writeID = DRV_PMP_Write ( handle, deviceAddress, &myWriteBuffer[0], nBytes, repeatCount);

// it will read 300 locations starting from 0x0206 into myReadBuffer
readID = DRV_PMP_Read ( handle, deviceAddress, &myReadBuffer[0], nBytes);
```

Transfer Status

The status of the read/write transfers can be obtained using API DRV_PMP_TransferStatus.

Example:

```
DRV_PMP_TRANSFER_STATUS writeStatus, readStatus;
uint32_t writeID, readID;

writeStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, writeID);
readStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, readID);
```

Client Operation

This section describes general client operation.

Description

General Client Operation

For the application to start using an instance of the module, it must call the DRV_PMP_Open function with a specific intent. This provides the configuration required to open the PMP instance for operation. If the driver is deinitialized using the function DRV_PMP_Deinitialize, the application must call the DRV_PMP_Open function again to set up the instance of the PMP. The function DRV_PMP_Open need not be called again if the system is reinitialized using the DRV_PMP_Reinitialize function.

The PMP driver supports DRV_IO_INTENT_NONBLOCKING, DRV_IO_INTENT_BLOCKING, DRV_IO_INTENT_EXCLUSIVE, and DRV_IO_INTENT_SHARED IO.

Example:

```
DRV_HANDLE handle;
```

```
// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

if( handle == DRV_HANDLE_INVALID )
{
    // Client cannot open the instance.
}
```

The function DRV_PMP_Close closes an already opened instance of the PMP driver, invalidating the handle. DRV_PMP_Open must have been called to obtain a valid opened device handle.

Example:

```
DRV_HANDLE handle;

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

/*...*/

DRV_PMP_Close( handle );
```

The client has the option to check the status through the function DRV_PMP_ClientStatus.

Example:

```
DRV_HANDLE handle;

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

if ( DRV_PMP_CLIENT_STATUS_OPEN != DRV_PMP_ClientStatus( handle ) )
   return 0;
```

Client Mode Setting

Any client-specific PMP configuration has to be done using a separate function, DRV_PMP_ModeConfig. This function must be called after the client is open using DRV_PMP_Open.

Following are the client-specific configuration parameters the user can set using this function:

Configuration Parameter	Description
pmpMode	Selects the PMP mode (master or slave) to use.
intMode	Selects the interrupt mode to use.
incrementMode	Sets up address for either auto-increment or decrement mode.
endianMode	Sets Little/Big endian mode.
portSize	Specifies the data width (8-bit or 16-bit).
waitStates	Selects the different wait states.
chipSelect	Selects the Chip Select line.

Example:

```
DRV_HANDLE handle;
DRV_PMP_MODE_CONFIG config;

config.chipSelect = PMCS1_AND_PMCS2_AS_CHIP_SELECT;
config.endianMode = LITTLE_ENDIAN;
config.incrementMode = PMP_ADDRESS_AUTO_INCREMENT;
config.intMode =  PMP_INTERRUPT_NONE;
config.pmpMode = PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT; //Master Mode 2
config.portSize =  PMP_DATA_SIZE_8_BITS;
config.waitStates.dataHoldWait = PMP_DATA_HOLD_2;
config.waitStates.dataWait = PMP_DATA_WAIT_THREE;
config.waitStates.strobeWait = PMP_STROBE_WAIT_5;

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

// Configure the client
DRV_PMP_ModeConfig ( handle, config );
```

Example Code for Complete Operation

A code example of complete operation is provided in this section.

Description

This example code will write 100 bytes of data twice (i.e., repeat once) in the memory location starting from 0x0206, and then it will be read in the buffer, myReadBuffer. The modes selected for this transfer are:

- Non-blocking
- No Interrupt
- PMP Master Mode 2
- Address Auto-increment
- No Address/Data Lines Multiplexing
- 8-bit data

Example:

```
void main(void)
{
DRV_PMP_INIT     init;
SYS_MODULE_OBJ   object;
SYS_STATUS       pmpStatus;
DRV_HANDLE       handle;
DRV_PMP_MODE_CONFIG config;
unsigned char myReadBuffer[300], myWriteBuffer[100];
uint32_t deviceAddress, nBytes, repeatCount, i;
uint32_t writeID, readID;
DRV_PMP_TRANSFER_STATUS writeStatus=0, readStatus=0;


// populate the PMP init configuration structure
    init.inputBuffer = PMP_INPUT_BUFFER_TTL;
    init.polarity.addressLatchPolarity = PMP_POLARITY_ACTIVE_HIGH;
    init.polarity.rwStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
    init.polarity.writeEnableStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
    init.polarity.chipselect1Polarity = PMP_POLARITY_ACTIVE_HIGH;
    init.polarity.chipselect2Polarity = PMP_POLARITY_ACTIVE_LOW;
    init.ports.addressPortsMask = PMP_PMA0_PORT | PMP_PMA1_PORT | PMP_PMA2_TO_PMA13_PORTS;
    init.ports.readWriteStrobe = PORT_ENABLE;
    init.ports.writeEnableStrobe = PORT_ENABLE;
    init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
    init.pmpID           = PMP_ID_0;
    init.stopInIdle      = false;
    init.muxMode         = PMP_MUX_NONE;

object = DRV_PMP_Initialize (DRV_PMP_INDEX_0, (SYS_MODULE_INIT *)&init);

pmpStatus = DRV_PMP_Status(object);

if ( SYS_STATUS_READY != pmpStatus)
{
    // Handle error
}

// Open the instance DRV_PMP_INDEX_0 with Non-blocking and Shared intent
handle = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_SHARED | DRV_IO_INTENT_NONBLOCKING);

if( handle == DRV_HANDLE_INVALID )
{
    // Client cannot open the instance.
}

config.chipSelect = PMCS1_AND_PMCS2_AS_CHIP_SELECT;
config.endianMode = LITTLE_ENDIAN;
config.incrementMode = PMP_ADDRESS_AUTO_INCREMENT;
config.intMode =  PMP_INTERRUPT_NONE;
config.pmpMode = PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT; //Master Mode 2
config.portSize =  PMP_DATA_SIZE_8_BITS;
```

```
config.waitStates.dataHoldWait = PMP_DATA_HOLD_2;
config.waitStates.dataWait = PMP_DATA_WAIT_THREE;
config.waitStates.strobeWait = PMP_STROBE_WAIT_5;

// Configure the client
DRV_PMP_ModeConfig ( handle, config );

deviceAddress = 0x0206;
nBytes = 100;
repeatCount = 0x01;
for (i=0; i<nBytes; i++)
{
   myWriteBuffer[i]=i*5+7;
}

writeID = DRV_PMP_Write ( handle, deviceAddress, &myWriteBuffer[0], nBytes, repeatCount);
readID = DRV_PMP_Read ( handle, deviceAddress, &myReadBuffer[0], nBytes*2);


while(!((writeStatus == PMP_TRANSFER_FINISHED)&&(readStatus == PMP_TRANSFER_FINISHED)))
{
    DRV_PMP_Tasks (object);

    writeStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, writeID);
    readStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, readID);
}

while(1);
}
```

Configuring the Library

Macros

	Name	Description
	DRV_PMP_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_PMP_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_PMP_QUEUE_SIZE	PMP queue size for different instances.

Description

The configuration of the PMP driver is based on the file `drv_pmp_config.h`.

This header file contains the configuration selection for the PMP Driver. Based on the selections made, the PMP Driver may support the selected features. These configuration settings will apply to all instances of the PMP Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_PMP_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_pmp_config.h

C

```
#define DRV_PMP_CLIENTS_NUMBER 2
```

Description

PMP maximum number of clients

This definition select the maximum number of clients that the PMP driver can support at run time.

Remarks

None.

DRV_PMP_INSTANCES_NUMBER Macro

Selects the maximum number of hardware instances that can be supported by the dynamic driver.

File

drv_pmp_config.h

C

```
#define DRV_PMP_INSTANCES_NUMBER 1
```

Description

PMP hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver.

Remarks

None.

DRV_PMP_QUEUE_SIZE Macro

PMP queue size for different instances.

File

drv_pmp_config.h

C

```
#define DRV_PMP_QUEUE_SIZE 8
```

Description

PMP queue size

The PMP queue size for a driver instances should be placed here. If more than one driver instance of PMP is present, then all takes the same queue size.

Remarks

All the transfers (Read/Write) first gets queued and gets completed sequentially when Task API is called in a loop. Therefore, the minimum value of this index should be 1.

Building the Library

This section lists the files that are available in the PMP Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/pmp`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_pmp.h`	This file provides the interface definitions of the PMP driver

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_pmp_dynamic.c`	This file contains the core implementation of the PMP driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The PMP Driver Library depends on the following modules:

- PMP Peripheral Library
- Interrupt System Service Library

Library Interface

a) System Functions

	Name	Description
	DRV_PMP_Deinitialize	Deinitializes the specified instance of the PMP driver module. **Implementation:** Dynamic
	DRV_PMP_Initialize	Initializes the PMP driver. **Implementation:** Static/Dynamic
	DRV_PMP_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings. **Implementation:** Dynamic
	DRV_PMP_Status	Provides the current status of the PMP driver module. **Implementation:** Dynamic
	DRV_PMP_Tasks	Maintains the driver's state machine and implements its ISR. **Implementation:** Dynamic
	DRV_PMP_TimingSet	Sets PMP timing parameters. **Implementation:** Static

b) Client Interaction Functions

	Name	Description
	DRV_PMP_ClientStatus	Gets the current client-specific status of the PMP driver. **Implementation:** Dynamic
	DRV_PMP_Close	Closes an opened instance of the PMP driver. **Implementation:** Dynamic
	DRV_PMP_ModeConfig	Configures the PMP modes. **Implementation:** Static/Dynamic
	DRV_PMP_Open	Opens the specified PMP driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_PMP_Read	Read the data from external device. **Implementation:** Static/Dynamic
	DRV_PMP_Write	Transfers the data from the MCU to the external device. **Implementation:** Static/Dynamic

c) Client Transfer Functions

	Name	Description
	DRV_PMP_TransferStatus	Returns the transfer status. **Implementation:** Dynamic

e) Data Types and Constants

	Name	Description
	DRV_PMP_INDEX_COUNT	Number of valid PMP driver indices.
	DRV_PMP_CHIPX_STROBE_MODE	PMP writeEnable/ReadWrite strobes.
	DRV_PMP_CLIENT_STATUS	PMP client status definitions.
	DRV_PMP_ENDIAN_MODE	PMP Endian modes.
	DRV_PMP_INDEX	PMP driver index definitions.
	DRV_PMP_INIT	Defines the PMP driver initialization data.
	DRV_PMP_MODE_CONFIG	PMP modes configuration.
	DRV_PMP_POLARITY_OBJECT	PMP polarity object.

	DRV_PMP_PORT_CONTROL	PMP port enable/disable definitions.
	DRV_PMP_PORTS	PMP port configuration.
	DRV_PMP_QUEUE_ELEMENT_OBJ	Defines the object for PMP queue element.
	_DRV_PMP_QUEUE_ELEMENT_OBJ	Defines the object for PMP queue element.
	DRV_PMP_TRANSFER_STATUS	Defines the PMP transfer status.
	DRV_PMP_WAIT_STATES	PMP wait states object.
	MAX_NONBUFFERED_BYTE_COUNT	After this number the PMP transfer should be polled to guarantee data transfer
	DRV_PMP_TRANSFER_TYPE	This is type DRV_PMP_TRANSFER_TYPE.
	PMP_QUEUE_ELEMENT_OBJECT	Defines the structure required for maintaining the queue element.

Description

This section describes the Application Programming Interface (API) functions of the PMP Driver.

Refer to each section for a detailed description.

a) System Functions

DRV_PMP_Deinitialize Function

Deinitializes the specified instance of the PMP driver module.

Implementation: Dynamic

File

drv_pmp.h

C

```
void DRV_PMP_Deinitialize(const SYS_MODULE_OBJ pmpDriverObject);
```

Returns

None.

Description

This function deinitializes the specified instance of the PMP driver module, disabling its operation (and any hardware). All internal data is invalidated.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_PMP_Status operation. The system has to use DRV_PMP_Status to find out when the module is in the ready state.

Preconditions

The DRV_PMP_Initialize function must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      pmpDriverObject;     //  Returned from DRV_PMP_Initialize
SYS_STATUS          status;

DRV_PMP_Deinitialize(pmpDriverObject);

status = DRV_PMP_Status(pmpDriverObject);
if (SYS_MODULE_DEINITIALIZED == status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
pmpDriverObject	Driver object handle, returned from the DRV_PMP_Initialize

Function

void DRV_PMP_Deinitialize (SYS_MODULE_OBJ pmpDriverObject)

DRV_PMP_Initialize Function

Initializes the PMP driver.

Implementation: Static/Dynamic

File

drv_pmp.h

C

```
SYS_MODULE_OBJ DRV_PMP_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, it returns a valid handle to a driver object. Otherwise, it returns SYS_MODULE_OBJ_INVALID. The returned object must be passed as argument to DRV_PMP_Reinitialize, DRV_PMP_Deinitialize, DRV_PMP_Tasks and DRV_PMP_Status routines.

Description

This function initializes the PMP driver, making it ready for clients to open and use it.

Remarks

This function must be called before any other PMP function is called.

This function should only be called once during system initialization unless DRV_PMP_Deinitialize is called to deinitialize the driver instance.

This function will NEVER block for hardware access. If the operation requires time to allow the hardware to reinitialize, it will be reported by the DRV_PMP_Status operation. The system must use DRV_PMP_Status to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Preconditions

None.

Example

```
DRV_PMP_INIT    init;
SYS_MODULE_OBJ  objectHandle;

// Populate the initialization structure
init.inputBuffer = PMP_INPUT_BUFFER_TTL;
init.polarity.addressLatchPolarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.rwStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.writeEnableStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.chipselect1Polarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.chipselect2Polarity = PMP_POLARITY_ACTIVE_LOW;
init.ports.addressPortsMask = PMP_PMA0_PORT | PMP_PMA1_PORT | PMP_PMA2_TO_PMA13_PORTS | PMP_PMA14_PORT;
init.ports.readWriteStrobe = PORT_ENABLE;
init.ports.writeEnableStrobe = PORT_ENABLE;
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.pmpID            = PMP_ID_0;
init.stopInIdle       = false;
init.muxMode          = PMP_MUX_NONE;

// Do something

objectHandle = DRV_PMP_Initialize(DRV_PMP_INDEX_0,  (SYS_MODULE_INIT*)&init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Index for the driver instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver

Function

SYS_MODULE_OBJ DRV_PMP_Initialize(const SYS_MODULE_INDEX drvIndex,
const SYS_MODULE_INIT * const init)

DRV_PMP_Reinitialize Function

Reinitializes the driver and refreshes any associated hardware settings.

Implementation: Dynamic

File

drv_pmp.h

C

```
void DRV_PMP_Reinitialize(const SYS_MODULE_OBJ pmpDriverObject, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

This function reinitializes the driver and refreshes any associated hardware settings using the specified initialization data, but it will not interrupt any ongoing operations.

Remarks

This function can be called multiple times to reinitialize the module.

This operation can be used to refresh any supported hardware registers as specified by the initialization data or to change the power state of the module.

This function will NEVER block for hardware access. If the operation requires time to allow the hardware to re-initialize, it will be reported by the DRV_PMP_Status operation. The system must use DRV_PMP_Status to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Preconditions

The DRV_PMP_Initialize function must have been called before calling this function and a valid SYS_MODULE_OBJ must have been returned.

Example

```
DRV_PMP_INIT    init;
SYS_MODULE_OBJ  pmpDriverObject;
SYS_STATUS  pmpStatus;

// Populate the initialization structure
init.inputBuffer = PMP_INPUT_BUFFER_TTL;
init.polarity.addressLatchPolarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.rwStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.writeEnableStrobePolarity = PMP_POLARITY_ACTIVE_LOW;
init.polarity.chipselect1Polarity = PMP_POLARITY_ACTIVE_HIGH;
init.polarity.chipselect2Polarity = PMP_POLARITY_ACTIVE_LOW;
init.ports.addressPortsMask = PMP_PMA0_PORT | PMP_PMA1_PORT | PMP_PMA2_TO_PMA13_PORTS | PMP_PMA14_PORT;
init.ports.readWriteStrobe = PORT_ENABLE;
init.ports.writeEnableStrobe = PORT_ENABLE;
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.pmpID            = PMP_ID_0;
init.stopInIdle       = false;
init.muxMode          = PMP_MUX_NONE;

DRV_PMP_Reinitialize(pmpDriverObject, (SYS_MODULE_INIT*)&init);

pmpStatus = DRV_PMP_Status(pmpDriverObject);
if (SYS_STATUS_BUSY == pmpStatus)
{
    // Check again later to ensure the driver is ready
}
else if (SYS_STATUS_ERROR >= pmpStatus)
```

```
{
    // Handle error
}
```

Parameters

Parameters	Description
pmpDriverObject	Driver object handle, returned from the DRV_PMP_Initialize

Function

void DRV_PMP_Reinitialize (SYS_MODULE_OBJ pmpDriverObject,
const SYS_MODULE_INIT * const init)

init - Pointer to the initialization data structure

DRV_PMP_Status Function

Provides the current status of the PMP driver module.

Implementation: Dynamic

File

drv_pmp.h

C

```
SYS_STATUS DRV_PMP_Status(const SYS_MODULE_OBJ pmpDriverObject);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Description

This function provides the current status of the PMP driver module.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

This operation can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, a previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The DRV_PMP_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      pmpDriverObject;     // Returned from DRV_PMP_Initialize
SYS_STATUS          status;

status = DRV_PMP_Status(pmpDriverObject);
else if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
pmpDriverObject	Driver object handle, returned from the DRV_PMP_Initialize routine

Function

SYS_STATUS DRV_PMP_Status (SYS_MODULE_OBJ pmpDriverObject)

DRV_PMP_Tasks Function

Maintains the driver's state machine and implements its ISR.

Implementation: Dynamic

File

drv_pmp.h

C

```
void DRV_PMP_Tasks(SYS_MODULE_OBJ pmpDriverObject);
```

Returns

None.

Description

This function is used to maintain the queue and execute the tasks stored in the queue. It resides in the ISR of the PMP for interrupt-driven implementations.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This function may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_PMP_Initialize function must have been called for the specified PMP driver instance.

Example

```
SYS_MODULE_OBJ      pmpDriverObject;     // Returned from DRV_PMP_Initialize

while (true)
{
    DRV_PMP_Tasks (pmpDriverObject);

    // Do other tasks
}
```

Parameters

Parameters	Description
pmpDriverObject	Object handle for the specified driver instance (returned from DRV_PMP_Initialize)

Function

void DRV_PMP_Tasks (SYS_MODULE_OBJ pmpDriverObject);

DRV_PMP_TimingSet Function

Sets PMP timing parameters.

Implementation: Static

File

drv_pmp.h

C

```
void DRV_PMP_TimingSet(PMP_DATA_WAIT_STATES dataWait, PMP_STROBE_WAIT_STATES strobeWait,
PMP_DATA_HOLD_STATES dataHold);
```

Returns

None.

Description

This function sets the PMP timing parameters.

Remarks

None.

Preconditions

The DRV_PMP_Initialize function must have been called.

Example

```
DRV_PMP0_TimingSet(PMP_DATA_WAIT_THREE,PMP_STROBE_WAIT_6,PMP_DATA_HOLD_4);
```

Parameters

Parameters	Description
dataWait	Data setup to read/write strobe wait states
strobeWait	Read/write strobe wait states
dataHold	Data hold time after read/write strobe wait states

Function

void DRV_PMP_TimingSet(
PMP_DATA_WAIT_STATES dataWait,
PMP_STROBE_WAIT_STATES strobeWait,
PMP_DATA_HOLD_STATES dataHold
)

b) Client Interaction Functions

DRV_PMP_ClientStatus Function

Gets the current client-specific status of the PMP driver.

Implementation: Dynamic

File

drv_pmp.h

C

```
DRV_PMP_CLIENT_STATUS DRV_PMP_ClientStatus(DRV_HANDLE hClient);
```

Returns

A DRV_PMP_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the PMP driver associated with the specified handle.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_PMP_Initialize routine must have been called.

DRV_PMP_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE hClient;  // Returned from DRV_PMP_Open
DRV_PMP_CLIENT_STATUS pmpClientStatus;

pmpClientStatus = DRV_PMP_ClientStatus(hClient);
```

```
if(DRV_PMP_CLIENT_STATUS_ERROR >= pmpClientStatus)
{
    // Handle the error
}
```

Parameters

Parameters	Description
hClient	A valid open-instance handle, returned from the driver's open routine

Function

DRV_PMP_CLIENT_STATUS DRV_PMP_ClientStatus (DRV_HANDLE hClient)

DRV_PMP_Close Function

Closes an opened instance of the PMP driver.

Implementation: Dynamic

File

drv_pmp.h

C

```
void DRV_PMP_Close(const DRV_HANDLE hClient);
```

Returns

None

Description

This function closes an opened instance of the PMP driver, invalidating the handle.

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_PMP_Open before the caller may use the driver again.

If DRV_IO_INTENT_BLOCKING was requested and the driver was built appropriately to support blocking behavior call may block until the operation is complete.

If DRV_IO_INTENT_NON_BLOCKING request the driver client can call the DRV_PMP_Status operation to find out when the module is in the ready state (the handle is no longer valid).

Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_PMP_Initialize routine must have been called for the specified PMP driver instance.

DRV_PMP_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE hClient;  // Returned from DRV_PMP_Open

DRV_PMP_Close(hClient);
```

Parameters

Parameters	Description
hClient	A valid open instance handle, returned from the driver's open routine

Function

void DRV_PMP_Close (DRV_HANDLE hClient)

DRV_PMP_ModeConfig Function

Configures the PMP modes.

Implementation: Static/Dynamic

File

drv_pmp.h

C

```
void DRV_PMP_ModeConfig(DRV_HANDLE hClient, DRV_PMP_MODE_CONFIG config);
```

Returns

None.

Description

This function configures the modes for client in which it wants to operate. Different master-slave modes, 8/16 data bits selection, address increment/decrement, interrupt mode, wait states, etc., can be configured through this function.

Remarks

This function will NEVER block waiting for hardware. If this API is called more than once for a particular client handle, previous config setting of that client will be overwritten.

Preconditions

Function DRV_PMP_Initialize must have been called. DRV_PMP_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE  hClient;
DRV_PMP_MODE_CONFIG config;

config.chipSelect = PMCS1_AND_PMCS2_AS_CHIP_SELECT;
config.endianMode = LITTLE_ENDIAN;
config.incrementMode = PMP_ADDRESS_AUTO_INCREMENT;
config.intMode =  PMP_INTERRUPT_NONE;
config.pmpMode = PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT;
config.portSize =  PMP_DATA_SIZE_8_BITS;
config.waitStates.dataHoldWait = PMP_DATA_HOLD_2;
config.waitStates.dataWait = PMP_DATA_WAIT_THREE;
config.waitStates.strobeWait = PMP_STROBE_WAIT_5;

DRV_PMP_ModeConfig ( hClient, config );
```

Parameters

Parameters	Description
hClient	Client handle obtained from DRV_PMP_Open API
config	Structure which will have all the required PMP modes configuration

Function

void DRV_PMP_ModeConfig (DRV_HANDLE hClient,
DRV_PMP_MODE_CONFIG config)

DRV_PMP_Open Function

Opens the specified PMP driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_pmp.h

C

```
DRV_HANDLE DRV_PMP_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID.

Description

This function opens the specified PMP driver instance and provides a handle that must be provided to all other client-level operations to identify

the caller and the instance of the driver.

Remarks

The handle returned is valid until the DRV_PMP_Close routine is called.

This function will NEVER block waiting for hardware.

If the DRV_IO_INTENT_BLOCKING is requested and the driver was built appropriately to support blocking behavior, other client-level operations may block waiting on hardware until they are complete.

If DRV_IO_INTENT_NON_BLOCKING is requested the driver client can call the DRV_PMP_ClientStatus operation to find out when the module is in the ready state.

If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID.

Preconditions

The DRV_PMP_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  hClient;

hClient = DRV_PMP_Open(DRV_PMP_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == hClient)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_PMP_Open (const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT intent)

DRV_PMP_Read Function

Read the data from external device.

Implementation: Static/Dynamic

File

drv_pmp.h

C

```
PMP_QUEUE_ELEMENT_OBJECT* DRV_PMP_Read(DRV_HANDLE hClient, uint32_t address, uint16_t* buffer, uint32_t
nBytes);
```

Returns

Returns the position number of the queue, where the data element was stored. Returns '0' when there is no place in the queue to store the data.

Description

This function reads the given number of data bytes from the given address of the external device to the MCU buffer through the selected PMP instance. This function should be used for all the master and slave modes. Proper configuration should be done using DRV_PMP_ModeConfig before calling this function.

Preconditions

The DRV_PMP_Initialize routine must have been called. DRV_PMP_Open must have been called to obtain a valid opened device handle. DRV_PMP_ModeConfig must have been called to configure the desired mode

Example

```
DRV_HANDLE hClient;  // Returned from DRV_PMP_Open
uint32_t deviceAddress;
uint32_t nBytes;
unsigned char myBuffer[nBytes];
uint32_t transferID;
```

```
transferID = DRV_PMP_Read ( hClient, deviceAddress, &myBuffer, nBytes);
```

Parameters

Parameters	Description
hClient	A valid open-instance handle, returned from the driver's open routine
address	Starting address of the slave device from where data has to be read. It does not have any significance for legacy slave mode and buffer mode. In PMP enhanced slave mode i.e. addressable buffer slave mode, this parameter should be the buffer number to be used.
buffer	Pointer to the buffer into which the data read through the PMP instance will be placed. Even if only one word has to be transferred, pointer should be used.
nBytes	Number of bytes that need to be read through the PMP instance

Function

uint32_t DRV_PMP_Read (DRV_HANDLE hClient,

uint32_t address,

unsigned char* buffer,

uint32_t nBytes)

DRV_PMP_Write Function

Transfers the data from the MCU to the external device.

Implementation: Static/Dynamic

File

drv_pmp.h

C

```
PMP_QUEUE_ELEMENT_OBJECT* DRV_PMP_Write(DRV_HANDLE* hClient, bool address, uint32_t * buffer, uint32_t
nBytes, uint32_t repeatCount);
```

Returns

Returns a 32-bit ID with which status of the transfer can be checked later. Returns '0' when there is no place in the queue to store the data.

Description

This function transfer the given number of data bytes from the MCU buffer location to the defined address of the external device through the selected PMP instance. It repeats the operation n (=repeatCount) number of times as well. This function should be used for all the master and slave modes. Proper configuration should be done using DRV_PMP_ModeConfig before calling this function.

Preconditions

The DRV_PMP_Initialize routine must have been called. DRV_PMP_Open must have been called to obtain a valid opened device handle. DRV_PMP_ModeConfig must have been called to configure the desired mode.

Example

```
DRV_HANDLE hClient;  // Returned from DRV_PMP_Open
uint32_t deviceAddress;
uint32_t nBytes;
unsigned char myBuffer[nBytes];
uint32_t repeatCount;
uint32_t transferID;

transferID = DRV_PMP_MasterWrite ( hClient, deviceAddress, &myBuffer, nBytes, repeatCount);
```

Parameters

Parameters	Description
hClient	A valid open-instance handle, returned from the driver's open routine
address	Starting address of the slave device where data has to be written. It does not have any significance for legacy slave mode and buffer mode. In PMP enhanced slave mode (i.e., addressable buffer slave mode), this parameter should be the buffer number to be used.
buffer	Pointer to MCU Buffer from which the data will be written through the PMP instance. even if only one word has to be transferred, pointer should be used.

nBytes	Total number of bytes that need to be written through the PMP instance
repeatCount	Number of times the data set (nBytes of data) to be repeatedly written. This value should be 0 if user does not want any repetition. If repeatCount is greater than 0, then after writing every nBytes of data, the buffer starts pointing to its first element. Ideally, PMP Address should be in auto increment/decrement mode for repeatCount greater than 0.

Function

uint32_t DRV_PMP_Write (DRV_HANDLE hClient,
uint32_t address,
unsigned char* buffer,
uint32_t nBytes,
uint32_t repeatCount)

c) Client Transfer Functions

DRV_PMP_TransferStatus Function

Returns the transfer status.

Implementation: Dynamic

File

drv_pmp.h

C

```
DRV_PMP_TRANSFER_STATUS DRV_PMP_TransferStatus(PMP_QUEUE_ELEMENT_OBJECT* queueObject);
```

Returns

A DRV_PMP_TRANSFER_STATUS value describing the current status of the transfer.

Description

This function returns the status of a particular transfer whose ID has been specified as input.

Example

```
uint32_8 seqID;
DRV_PMP_TRANSFER_STATUS transferStatus;

transferStatus = DRV_PMP_TransferStatus( DRV_PMP_INDEX_0, seqID);
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
seqID	A valid ID returned from read/write transfer functions

Function

DRV_PMP_TRANSFER_STATUS DRV_PMP_TransferStatus(DRV_HANDLE hClient)

d) Miscellaneous Functions

e) Data Types and Constants

DRV_PMP_INDEX_COUNT Macro

Number of valid PMP driver indices.

File

drv_pmp.h

C

```
#define DRV_PMP_INDEX_COUNT _PMP_EXISTS
```

Description

PMP Driver Module Index Count

This constant identifies the number of valid PMP driver indices.

Remarks

The value of "_PMP_EXISTS" is derived from device-specific header files defined as part of the peripheral libraries.

DRV_PMP_CHIPX_STROBE_MODE Enumeration

PMP writeEnable/ReadWrite strobes.

File

drv_pmp.h

C

```
typedef enum {
  PMP_RW_STROBE_WITH_ENABLE_STROBE,
  PMP_READ_AND_WRITE_STROBES
} DRV_PMP_CHIPX_STROBE_MODE;
```

Members

Members	Description
PMP_RW_STROBE_WITH_ENABLE_STROBE	One strobe for read/write and another for enable
PMP_READ_AND_WRITE_STROBES	Separate strobes for read and write operations

Description

PMP writeEnable/ReadWrite strobes

This enumeration provides ReadWrite/WriteEnable Strobe definitions.

DRV_PMP_CLIENT_STATUS Enumeration

PMP client status definitions.

File

drv_pmp.h

C

```
typedef enum {
  DRV_PMP_CLIENT_STATUS_INVALID,
  PMP_CLIENT_STATUS_CLOSED,
  DRV_PMP_CLIENT_STATUS_OPEN
} DRV_PMP_CLIENT_STATUS;
```

Description

PMP Client Status

This enumeration provides various client status possibilities.

DRV_PMP_ENDIAN_MODE Enumeration

PMP Endian modes.

File

drv_pmp.h

C

```
typedef enum {
  LITTLE,
  BIG
} DRV_PMP_ENDIAN_MODE;
```

Members

Members	Description
LITTLE	Little Endian
BIG	Big Endian

Description

PMP Endian modes

This enumeration holds the Endian configuration options.

DRV_PMP_INDEX Enumeration

PMP driver index definitions.

File

drv_pmp.h

C

```
typedef enum {
  DRV_PMP_INDEX_0,
  DRV_PMP_INDEX_1
} DRV_PMP_INDEX;
```

Members

Members	Description
DRV_PMP_INDEX_0	First PMP instance
DRV_PMP_INDEX_1	Second PMP instance (not available for now)

Description

PMP Driver Module Index Numbers

These constants provide PMP driver index definitions.

Remarks

These values should be passed into the DRV_PMP_Initialize and DRV_PMP_Open functions to identify the driver instance in use.

DRV_PMP_INIT Structure

Defines the PMP driver initialization data.

File

drv_pmp.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  PMP_MODULE_ID pmpID;
  bool stopInIdle;
  PMP_MUX_MODE muxMode;
  PMP_INPUT_BUFFER_TYPE inputBuffer;
  DRV_PMP_POLARITY_OBJECT polarity;
  DRV_PMP_PORTS ports;
} DRV_PMP_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	module power state info
PMP_MODULE_ID pmpID;	module PLIB ID
bool stopInIdle;	Stop in Idle enable
PMP_MUX_MODE muxMode;	MUX mode
PMP_INPUT_BUFFER_TYPE inputBuffer;	Input buffer type to be used
DRV_PMP_POLARITY_OBJECT polarity;	Polarity settings

DRV_PMP_PORTS ports;	PMP port settings

Description

PMP Driver Initialize Data

This data type defines data required to initialize or reinitialize the PMP driver.

Remarks

Not all the initialization features are available for all devices.

DRV_PMP_MODE_CONFIG Structure

PMP modes configuration.

File

drv_pmp.h

C

```
typedef struct {
  PMP_OPERATION_MODE pmpMode;
  PMP_INTERRUPT_MODE intMode;
  PMP_INCREMENT_MODE incrementMode;
  DRV_PMP_ENDIAN_MODE endianMode;
  PMP_DATA_SIZE portSize;
  DRV_PMP_WAIT_STATES waitStates;
  PMP_CHIPSELECT_FUNCTION chipSelect;
} DRV_PMP_MODE_CONFIG;
```

Members

Members	Description
PMP_OPERATION_MODE pmpMode;	PMP Usage Mode Type
PMP_INTERRUPT_MODE intMode;	Interrupt mode
PMP_INCREMENT_MODE incrementMode;	should be appropriately selected based on read/write requirements and operation mode setting */ address/buffer increment mode
DRV_PMP_ENDIAN_MODE endianMode;	it does not have any significance in PMP slave mode or 8bit data mode */ Endian modes
PMP_DATA_SIZE portSize;	Data Port Size
DRV_PMP_WAIT_STATES waitStates;	Wait states
PMP_CHIPSELECT_FUNCTION chipSelect;	use this when PLIB is fixed

Description

PMP modes configuration

This data type controls the configuration of PMP modes.

DRV_PMP_POLARITY_OBJECT Structure

PMP polarity object.

File

drv_pmp.h

C

```
typedef struct {
  PMP_POLARITY_LEVEL addressLatchPolarity;
  PMP_POLARITY_LEVEL byteEnablePolarity;
  PMP_POLARITY_LEVEL rwStrobePolarity;
  PMP_POLARITY_LEVEL writeEnableStrobePolarity;
  PMP_POLARITY_LEVEL chipselect1Polarity;
  PMP_POLARITY_LEVEL chipselect2Polarity;
} DRV_PMP_POLARITY_OBJECT;
```

Members

Members	Description
PMP_POLARITY_LEVEL addressLatchPolarity;	Address latch polarity

PMP_POLARITY_LEVEL byteEnablePolarity;	ByteEnable port polarity
PMP_POLARITY_LEVEL rwStrobePolarity;	Read/Write strobe polarity
PMP_POLARITY_LEVEL writeEnableStrobePolarity;	Write/Enable strobe polarity
PMP_POLARITY_LEVEL chipselect1Polarity;	ChipSelect-1 Polarity
PMP_POLARITY_LEVEL chipselect2Polarity;	chipSelect-2 Polarity

Description

PMP polarity object

This structure holds the polarities of different entities to be configured.

DRV_PMP_PORT_CONTROL Enumeration

PMP port enable/disable definitions.

File

drv_pmp.h

C

```
typedef enum {
  PORT_ENABLE,
  PORT_DISABLE
} DRV_PMP_PORT_CONTROL;
```

Members

Members	Description
PORT_ENABLE	Enable the given port
PORT_DISABLE	Disable the given port

Description

PMP port enable/disable.

This enumeration provides port enable/disable values.

DRV_PMP_PORTS Structure

PMP port configuration.

File

drv_pmp.h

C

```
typedef struct {
  PMP_ADDRESS_PORT addressPortsMask;
  PMP_PMBE_PORT byteEnablePort;
  DRV_PMP_PORT_CONTROL readWriteStrobe;
  DRV_PMP_PORT_CONTROL writeEnableStrobe;
} DRV_PMP_PORTS;
```

Members

Members	Description
PMP_ADDRESS_PORT addressPortsMask;	User needs to put the address lines which he wants to use in ORed fashion * Address ports
PMP_PMBE_PORT byteEnablePort;	Byte enable ports
DRV_PMP_PORT_CONTROL readWriteStrobe;	READ/WRITE Strobe PORT
DRV_PMP_PORT_CONTROL writeEnableStrobe;	WRITE/ENABLE strobe port

Description

PMP Ports

This structure holds the ports (including the address ports) to be configured by the application to function as general purpose I/O (GPIO) or part of the PMP.

DRV_PMP_QUEUE_ELEMENT_OBJ Structure

Defines the object for PMP queue element.

File

drv_pmp.h

C

```
typedef struct _DRV_PMP_QUEUE_ELEMENT_OBJ {
  struct _DRV_PMP_CLIENT_OBJ * hClient;
  uint32_t buffer;
  uint16_t* addressBuffer;
  uint32_t nTransfers;
  int32_t nRepeats;
  DRV_PMP_TRANSFER_TYPE type;
} DRV_PMP_QUEUE_ELEMENT_OBJ;
```

Members

Members	Description
struct _DRV_PMP_CLIENT_OBJ * hClient;	handle of the client object returned from open API
uint32_t buffer;	pointer to the buffer holding the transmitted data
uint16_t* addressBuffer;	pointer to the buffer holding the transmitted data
uint32_t nTransfers;	number of bytes to be transferred
int32_t nRepeats;	number of times the data set has to be transferred repeatedly
DRV_PMP_TRANSFER_TYPE type;	PMP Read or Write

Description

PMP Driver Queue Element Object

This defines the object structure for each queue element of PMP. This object gets created for every Read/Write operations APIs.

Remarks

None

DRV_PMP_TRANSFER_STATUS Enumeration

Defines the PMP transfer status.

File

drv_pmp.h

C

```
typedef enum {
  MASTER_8BIT_TRANSFER_IN_PROGRESS = PMP_DATA_SIZE_8_BITS,
  MASTER_16BIT_TRANSFER_IN_PROGRESS = PMP_DATA_SIZE_16_BITS,
  MASTER_8BIT_BUFFER_IN_PROGRESS,
  MASTER_16BIT_BUFFER_IN_PROGRESS,
  MASTER_8BIT_TRANSFER_CONTINUE,
  MASTER_8BIT_BUFFER_CONTINUE,
  QUEUED_BUT_PMP_TRANSFER_NOT_STARTED,
  PMP_TRANSFER_FINISHED
} DRV_PMP_TRANSFER_STATUS;
```

Description

Queue Element Transfer Status

This enumeration defines the PMP transfer status.

DRV_PMP_WAIT_STATES Structure

PMP wait states object.

File

drv_pmp.h

C

```
typedef struct {
  PMP_DATA_HOLD_STATES dataHoldWait;
  PMP_STROBE_WAIT_STATES strobeWait;
  PMP_DATA_WAIT_STATES dataWait;
} DRV_PMP_WAIT_STATES;
```

Members

Members	Description
PMP_DATA_HOLD_STATES dataHoldWait;	data hold wait states
PMP_STROBE_WAIT_STATES strobeWait;	read/write strobe wait states
PMP_DATA_WAIT_STATES dataWait;	data wait strobe wait sates

Description

PMP wait states object

This structure holds the different wait states to be configured. Refer to the PMP PLIB help document for the possible values and meaning of the different wait states.

MAX_NONBUFFERED_BYTE_COUNT Macro

File

drv_pmp.h

C

```
#define MAX_NONBUFFERED_BYTE_COUNT 4 /*************************************************************************
                                      After this number the PMP transfer should be polled to guarantee data
                                      transfer
                                      *************************************************************************

*/
```

Description

After this number the PMP transfer should be polled to guarantee data transfer

DRV_PMP_TRANSFER_TYPE Enumeration

File

drv_pmp.h

C

```
typedef enum {
  ADDRESS,
  READ,
  WRITE,
  BUFFERED_WRITE
} DRV_PMP_TRANSFER_TYPE;
```

Members

Members	Description
ADDRESS	PMP Address needs to be updated
READ	PMP Read Transfer
WRITE	PMP Write Transfer
BUFFERED_WRITE	PMP Array Write Transfer

Description

This is type DRV_PMP_TRANSFER_TYPE.

PMP_QUEUE_ELEMENT_OBJECT Structure

Defines the structure required for maintaining the queue element.

File

drv_pmp.h

C

```
typedef struct {
  DRV_PMP_QUEUE_ELEMENT_OBJ data;
  DRV_PMP_TRANSFER_STATUS eTransferStatus;
  uint32_t nTransfersDone;
} PMP_QUEUE_ELEMENT_OBJECT;
```

Members

Members	Description
DRV_PMP_QUEUE_ELEMENT_OBJ data;	The PMP Q Element
DRV_PMP_TRANSFER_STATUS eTransferStatus;	Flag to indicate that the element is in use
uint32_t nTransfersDone;	sequence id

Description

Queue Element Object

This defines the structure required for maintaining the queue element.

Remarks

None

Files

Files

Name	Description
drv_pmp.h	Parallel Master Port (PMP) device driver interface file.
drv_pmp_config.h	PMP driver configuration definitions template

Description

This section lists the source and header files used by the PMP Driver Library.

drv_pmp.h

Parallel Master Port (PMP) device driver interface file.

Enumerations

	Name	Description
	DRV_PMP_CHIPX_STROBE_MODE	PMP writeEnable/ReadWrite strobes.
	DRV_PMP_CLIENT_STATUS	PMP client status definitions.
	DRV_PMP_ENDIAN_MODE	PMP Endian modes.
	DRV_PMP_INDEX	PMP driver index definitions.
	DRV_PMP_PORT_CONTROL	PMP port enable/disable definitions.
	DRV_PMP_TRANSFER_STATUS	Defines the PMP transfer status.
	DRV_PMP_TRANSFER_TYPE	This is type DRV_PMP_TRANSFER_TYPE.

Functions

	Name	Description
	DRV_PMP_ClientStatus	Gets the current client-specific status of the PMP driver. **Implementation:** Dynamic
	DRV_PMP_Close	Closes an opened instance of the PMP driver. **Implementation:** Dynamic
	DRV_PMP_Deinitialize	Deinitializes the specified instance of the PMP driver module. **Implementation:** Dynamic
	DRV_PMP_Initialize	Initializes the PMP driver. **Implementation:** Static/Dynamic

	DRV_PMP_ModeConfig	Configures the PMP modes. **Implementation:** Static/Dynamic
	DRV_PMP_Open	Opens the specified PMP driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_PMP_Read	Read the data from external device. **Implementation:** Static/Dynamic
	DRV_PMP_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings. **Implementation:** Dynamic
	DRV_PMP_Status	Provides the current status of the PMP driver module. **Implementation:** Dynamic
	DRV_PMP_Tasks	Maintains the driver's state machine and implements its ISR. **Implementation:** Dynamic
	DRV_PMP_TimingSet	Sets PMP timing parameters. **Implementation:** Static
	DRV_PMP_TransferStatus	Returns the transfer status. **Implementation:** Dynamic
	DRV_PMP_Write	Transfers the data from the MCU to the external device. **Implementation:** Static/Dynamic

Macros

	Name	Description
	DRV_PMP_INDEX_COUNT	Number of valid PMP driver indices.
	MAX_NONBUFFERED_BYTE_COUNT	After this number the PMP transfer should be polled to guarantee data transfer

Structures

	Name	Description
	_DRV_PMP_QUEUE_ELEMENT_OBJ	Defines the object for PMP queue element.
	DRV_PMP_INIT	Defines the PMP driver initialization data.
	DRV_PMP_MODE_CONFIG	PMP modes configuration.
	DRV_PMP_POLARITY_OBJECT	PMP polarity object.
	DRV_PMP_PORTS	PMP port configuration.
	DRV_PMP_QUEUE_ELEMENT_OBJ	Defines the object for PMP queue element.
	DRV_PMP_WAIT_STATES	PMP wait states object.
	PMP_QUEUE_ELEMENT_OBJECT	Defines the structure required for maintaining the queue element.

Description

PMP Device Driver Interface

The PMP device driver provides a simple interface to manage the Parallel Master and Slave ports. This file defines the interface definitions and prototypes for the PMP driver.

File Name

drv_pmp.h

Company

Microchip Technology Inc.

drv_pmp_config.h

PMP driver configuration definitions template

Macros

	Name	Description
	DRV_PMP_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_PMP_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_PMP_QUEUE_SIZE	PMP queue size for different instances.

Description

PMP Driver Configuration Definitions for the Template Version

These definitions statically define the driver's mode of operation.

File Name

drv_pmp_config_template.h

Company

Microchip Technology Inc.

RTCC Driver Library

This section describes the RTCC Driver Library.

Introduction

The Real-Time Clock Calendar (RTCC) Static Driver provides a high-level interface to manage the RTCC module on the Microchip family of microcontrollers.

Description

Through the MHC, this driver provides APIs for the following:

- Initializing the module
- Starting/Stopping the RTCC
- Status functions to yield the date/time
- Status functions to yield the alarm date/time
- Clock output control

Library Interface

System Interaction Functions

	Name	Description
	DRV_RTCC_AlarmDateGet	Gets the Alarm Date of the RTCC. **Implementation:** Static
	DRV_RTCC_AlarmTimeGet	Gets the Alarm Time of the RTCC. **Implementation:** Static
	DRV_RTCC_ClockOutput	Enables Clock Output for the RTCC. **Implementation:** Static
	DRV_RTCC_DateGet	Gets the Date of the RTCC. **Implementation:** Static
	DRV_RTCC_Initialize	Initializes the RTCC instance for the specified driver index. **Implementation:** Static
	DRV_RTCC_Start	Starts the RTCC. **Implementation:** Static
	DRV_RTCC_Stop	Stops the RTCC. **Implementation:** Static
	DRV_RTCC_TimeGet	Gets the time of the RTCC. **Implementation:** Static

Description

This section describes the Application Programming Interface (API) functions of the RTCC Driver Library.

System Interaction Functions

DRV_RTCC_AlarmDateGet Function

Gets the Alarm Date of the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
uint32_t DRV_RTCC_AlarmDateGet();
```

Returns

uint32_t alarm date value

Description

This routine gets the RTCC alarm date.

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

uint32_t DRV_RTCC_AlarmDateGet(void)

DRV_RTCC_AlarmTimeGet Function

Gets the Alarm Time of the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
uint32_t DRV_RTCC_AlarmTimeGet();
```

Returns

uint32_t alarm time value

Description

This routine gets the RTCC alarm time.

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

uint32_t DRV_RTCC_AlarmTimeGet(void)

DRV_RTCC_ClockOutput Function

Enables Clock Output for the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
void DRV_RTCC_ClockOutput();
```

Returns

None.

Description

This routine enables the clock output for the RTCC

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

void DRV_RTCC_ClockOutput(void)

DRV_RTCC_DateGet Function

Gets the Date of the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
uint32_t DRV_RTCC_DateGet();
```

Returns

uint32_t date value

Description

This routine gets the RTCC date.

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

uint32_t DRV_RTCC_DateGet(void)

DRV_RTCC_Initialize Function

Initializes the RTCC instance for the specified driver index.

Implementation: Static

File

help_drv_rtcc.h

C

```
void DRV_RTCC_Initialize();
```

Returns

None.

Description

This routine initializes the RTCC driver instance for the specified driver instance, making it ready for clients to use it. The initialization routine is specified by the MHC parameters.

Remarks

This routine must be called before any other RTCC routine is called. This routine should only be called once during system initialization.

Preconditions

None.

Function

void DRV_RTCC_Initialize(void)

DRV_RTCC_Start Function

Starts the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
void DRV_RTCC_Start();
```

Returns

None.

Description

This routine starts the RTCC, making it ready for clients to use it.

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

void DRV_RTCC_Start(void)

DRV_RTCC_Stop Function

Stops the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
void DRV_RTCC_Stop();
```

Returns

None.

Description

This routine stops the RTCC.

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

void DRV_RTCC_Stop(void)

DRV_RTCC_TimeGet Function

Gets the time of the RTCC.

Implementation: Static

File

help_drv_rtcc.h

C

```
uint32_t DRV_RTCC_TimeGet();
```

Returns

uint32_t time value

Description

This routine gets the RTCC time.

Remarks

None.

Preconditions

DRV_RTCC_Initialize has been called.

Function

uint32_t DRV_RTCC_TimeGet(void)

Secure Digital (SD) Card Driver Library

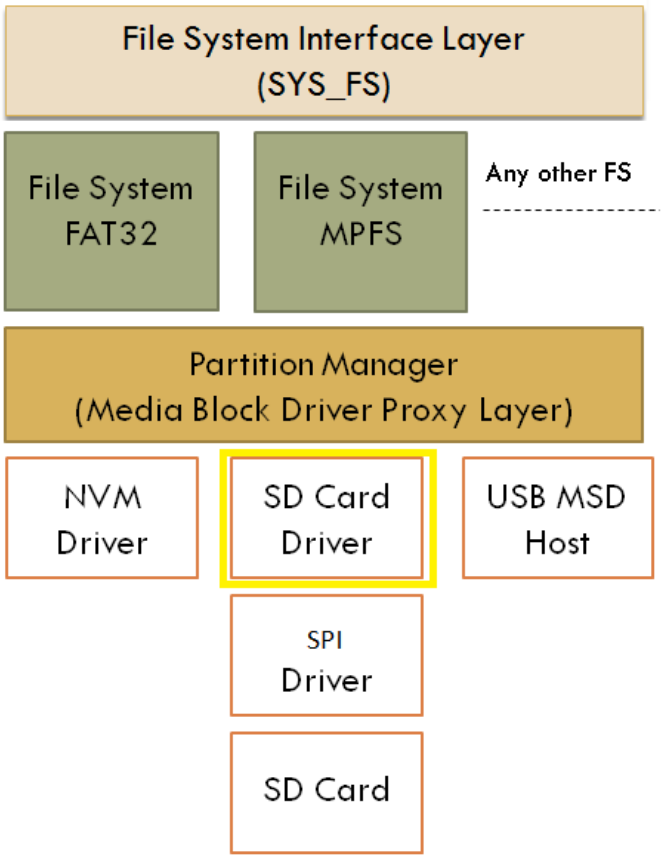
This section describes the Secure Digital (SD) Card Driver Library.

Introduction

The SD Card driver provides the necessary interfaces to interact with an SD card. It provides the necessary abstraction for the higher layer.

Description

A SD Card is a non-volatile memory (Flash memory) card designed to provide high-capacity memory in a small size. Its applications include digital video camcorders, digital cameras, handheld computers, audio players, and mobile phones.



Using the Library

This topic describes the basic architecture of the SD Card Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_sdcard.h`

The interface to the SD Card Driver library is defined in the `drv_sdcard.h` header file. This file is included by the `drv.h` file. Any C language source (`.c`) file that uses the SD Card Driver library should include `drv.h`.

Please refer to the What is MPLAB Harmony? section for how the Driver interacts with the framework.

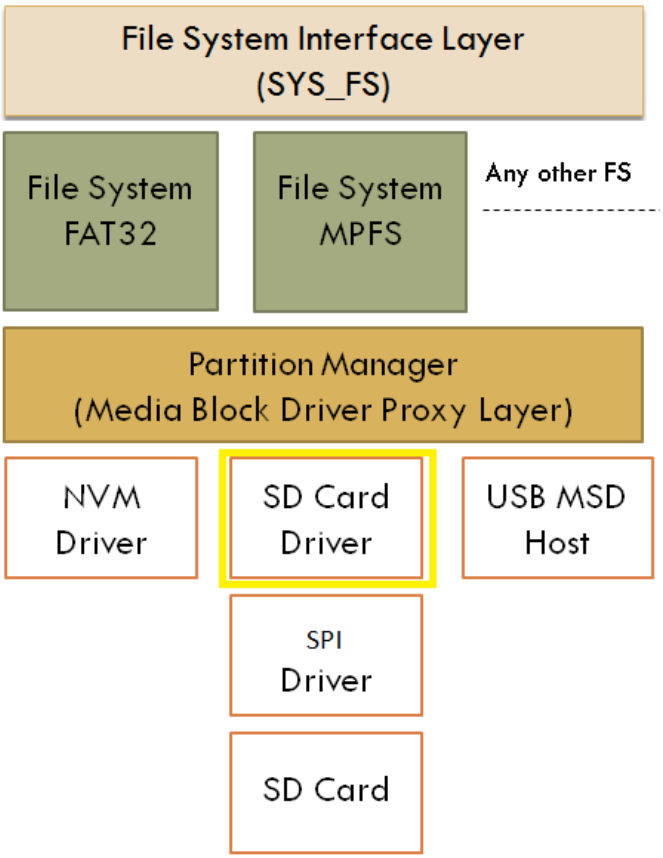
Abstraction Model

This library provides a low-level abstraction of the SD Card Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The SD Card driver comes in the layer below the Partition Manager in the MPLAB Harmony file system architecture and it uses the SPI Driver to interact with the SD card.

SD Card Driver Software Abstraction Block Diagram



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SD Card module.

Library Interface Section	Description
System Level Functions	Includes functions for initialize the module.
Client Level Functions	Functions to open and close a client.
Operation Functions	Functions for read and write operations
Module Information Functions	Functions for information about the module.
Version Information Functions	Functions to get the software version.

How the Library Works

This section describes how the SD Card Driver Library operates.

Description

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

The library provides interfaces that support:

- Driver Initialization Functionality
- Client Block Data Functionality
- Client Access Functionality

SD Card Driver Initialization

This section provides information for system initialization and reinitialization.

Description

The system performs the initialization and the reinitialization of the device driver with settings that affect only the instance of the device that is being initialized or reinitialized. During system initialization each instance of the SD Card module would be initialized with the following configuration settings (either passed dynamically at run time using DRV_SDCARD_INIT or by using initialization overrides) that are supported by the specific SD Card device hardware:

- SPI Peripheral ID: Identifies the SPI Peripheral ID to be used for the SD Card Driver
- SPI Index: SPI Driver Index
- SD Card frequency: SD Card communication speed
- SPI Clock source: Peripheral clock used by the SPI
- Write-Protect Port: Port used to check if the SD Card is write protected
- Write-Protect Pin: Pin used to check if the SD Card is write protected
- Chip Select Port: Port used for the SPI Chip Select
- Chip Select Pin: Pin used for the SPI Chip Select

The DRV_SDCARD_Initialize function returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the initialize interface would be used by the other system interfaces, such as DRV_SDCARD_Deinitialize, DRV_SDCARD_Status, and DRV_SDCARD_Tasks.

Note: The system initialization and the reinitialization settings, only affect the instance of the peripheral that is being initialized or reinitialized.

Example:

```
const DRV_SDCARD_INIT drvSDCardInit =
{
.spiId = SPI_ID_2,
.spiIndex = 0,
.sdcardSpeedHz = 20000000,
.spiClk = CLK_BUS_PERIPHERAL_2,
.writeProtectPort = PORT_CHANNEL_F,
.writeProtectBitPosition = PORTS_BIT_POS_1,
.chipSelectPort = PORT_CHANNEL_B,
.chipSelectBitPosition = PORTS_BIT_POS_14,
};

void SYS_Initialize (void *data)
{
.
.
sysObj.drvSDCard = DRV_SDCARD_Initialize(DRV_SDCARD_INDEX_0,(SYS_MODULE_INIT *)&drvSDCardInit);
.
.
}
```

Tasks Routine

The system will call DRV_SDCARD_Tasks, from system task service to maintain the driver's state machine.

Client Access Operation

This section provides information for general client operation.

Description

General Client Operation

For the application to start using an instance of the module, it must call the DRV_SDCARD_Open function. This provides the configuration required to open the SD Card instance for operation. If the driver is deinitialized using the function DRV_SDCARD_Deinitialize, the application must call the DRV_SDCARD_Open function again to set up the instance of the SDCARD.

For the various options available for I/O INTENT please refer to Data Types and Constants in the Library Interface section.

Example:

```
DRV_HANDLE handle;
handle = DRV_SDCARD_Open(DRV_SDCARD_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
// Unable to open the driver
}
```

Client Block Data Operation

This topic provides information on client block data operation.

Description

The SDCARD Driver provides a block interface to access the SD Card. The interface provides functionality to read from and write to the SD Card.

Reading Data from the SD Card:

The following steps outline the sequence to be followed for reading data from the SD Card:

1. The system should have completed necessary initialization and DRV_SDCARD_Tasks should either be running in a polled environment, or in an interrupt environment.
2. The driver should have been opened with the necessary intent.
3. Invoke the DRV_SDCARD_Read function and pass the pointer where the data is to be stored, block start address and the number of blocks of data to be read.
4. The client should validate the command handle returned by the DRV_SDCARD_Read function. DRV_SDCARD_COMMAND_HANDLE_INVALID value indicates that an error has occurred which the client needs to handle.
5. If the request was successful then the client can check the status of the request by invoking the DRV_SDCARD_CommandStatus and passing the command handle returned by the read request. Alternately the client could use the event handler for notifications from the driver.
6. The client will be able to close itself by calling the DRV_SDCARD_Close.

Example:

```
// This code shows how to read data from the SD Card
DRV_HANDLE sdcardHandle;
DRV_SDCARD_COMMAND_HANDLE sdcardCommandHandle;
DRV_SDCARD_COMMAND_STATUS commandStatus;
uint8_t readBuf[512];
uint32_t blockAddress;
uint32_t nBlocks;

/* Initialize the block start address and the number of blocks to be read */
blockAddress = 0;
nBlocks = 1;

DRV_SDCARD_Read(sdcardHandle, &sdcardCommandHandle, (uint8_t *)readBuf, blockAddress, nBlocks);
if(DRV_SDCARD_COMMAND_HANDLE_INVALID == sdcardCommandHandle)
{
    /* Failed to queue the read request. Handle the error. */
}
// Wait until the command completes. This should not
// be a while loop if part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
commandStatus = DRV_SDCARD_CommandStatus(sdcardHandle, sdcardCommandHandle);
if(DRV_SDCARD_COMMAND_COMPLETED == commandStatus)
{
    /* Read completed */
}
else if (DRV_SDCARD_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Read Failed */
}
```

Writing Data to the SD Card:

The following steps outline the sequence to be followed for writing data to the SD Card:

1. The system should have completed necessary initialization and DRV_SDCARD_Tasks should either be running in a polled environment, or in an interrupt environment.
2. The driver should have been opened with the necessary intent.
3. Invoke the DRV_SDCARD_Write function and pass the pointer to the data to be written, block start address and the number of blocks of data to be written.
4. The client should validate the command handle returned by the DRV_SDCARD_Write function. DRV_SDCARD_COMMAND_HANDLE_INVALID value indicates that an error has occurred which the client needs to handle.
5. If the request was successful then the client can check the status of the request by invoking the DRV_SDCARD_CommandStatus and passing the command handle returned by the write request. Alternately, the client could use the event handler for notifications from the driver.
6. The client will be able to close itself by calling the DRV_SDCARD_Close.

Example:

```
// This code shows how to write data to the SD Card
```

```
DRV_HANDLE sdcardHandle;
DRV_SDCARD_COMMAND_HANDLE sdcardCommandHandle;
DRV_SDCARD_COMMAND_STATUS commandStatus;
uint8_t writeBuf[512];
uint32_t blockAddress;
uint32_t nBlocks;

/* Initialize the block start address and the number of blocks to be written */
blockAddress = 0;
nBlocks = 1;
/* Populate writeBuf with the data to be written */

DRV_SDCARD_Write(sdcardHandle, &sdcardCommandHandle, (uint8_t *)writeBuf, blockAddress, nBlocks);
if(DRV_SDCARD_COMMAND_HANDLE_INVALID == sdcardCommandHandle)
{
    /* Failed to queue the write request. Handle the error. */
}
// Wait until the command completes. This should not
// be a while loop if part of cooperative multi-tasking
// routine. In that case, it should be invoked in task
// state machine.
commandStatus = DRV_SDCARD_CommandStatus(sdcardHandle, sdcardCommandHandle);
if(DRV_SDCARD_COMMAND_COMPLETED == commandStatus)
{
    /* Write completed */
}
else if (DRV_SDCARD_COMMAND_ERROR_UNKNOWN == commandStatus)
{
    /* Write Failed */
}
```

Configuring the Library

Macros

	Name	Description
	DRV_SDCARD_CLIENTS_NUMBER	Selects the miximum number of clients
	DRV_SDCARD_INDEX_MAX	SD Card Static Index selection
	DRV_SDCARD_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver
	DRV_SDCARD_POWER_STATE	Defines an override of the power state of the SD Card driver.
	DRV_SDCARD_SYS_FS_REGISTER	Register to use with the File system
	DRV_SDCARD_ENABLE_WRITE_PROTECT_CHECK	Enable SD Card write protect check.

Description

The configuration of the SD Card Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the SD Card Driver. Based on the selections made, the SD Card Driver may support the selected features. These configuration settings will apply to all instances of the SD Card.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_SDCARD_CLIENTS_NUMBER Macro

Selects the miximum number of clients

File

drv_sdcard_config_template.h

C

```
#define DRV_SDCARD_CLIENTS_NUMBER 1
```

Description

SD Card Maximum Number of Clients

This definition select the maximum number of clients that the SD Card driver can support at run time. Not defining it means using a single client.

Remarks

None.

DRV_SDCARD_INDEX_MAX Macro

SD Card Static Index selection

File

drv_sdcard_config_template.h

C

```
#define DRV_SDCARD_INDEX_MAX 1
```

Description

SD Card Static Index Selection

SD Card Static Index selection for the driver object reference

Remarks

This index is required to make a reference to the driver object

DRV_SDCARD_INSTANCES_NUMBER Macro

Selects the maximum number of hardware instances that can be supported by the dynamic driver

File

drv_sdcard_config_template.h

C

```
#define DRV_SDCARD_INSTANCES_NUMBER 1
```

Description

SD Card hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver. Not defining it means using a static driver.

Remarks

None

DRV_SDCARD_POWER_STATE Macro

Defines an override of the power state of the SD Card driver.

File

drv_sdcard_config_template.h

C

```
#define DRV_SDCARD_POWER_STATE SYS_MODULE_POWER_IDLE_STOP
```

Description

SD Card power state configuration

Defines an override of the power state of the SD Card driver.

Remarks

This feature may not be available in the device or the SD Card module selected.

DRV_SDCARD_SYS_FS_REGISTER Macro

Register to use with the File system

File

drv_sdcard_config_template.h

C

```
#define DRV_SDCARD_SYS_FS_REGISTER
```

Description

SDCARD Driver Register with File System

Specifying this macro enables the SDCARD driver to register its services with the SYS FS.

Remarks

This macro is optional and should be specified only if the SDCARD driver is to be used with the File System.

DRV_SDCARD_ENABLE_WRITE_PROTECT_CHECK Macro

Enable SD Card write protect check.

File

drv_sdcard_config_template.h

C

```
#define DRV_SDCARD_ENABLE_WRITE_PROTECT_CHECK
```

Description

SDCARD Driver Enable Write Protect Check

Specifying this macro enables the SDCARD driver to check whether the SD card is write protected.

Remarks

None

Building the Library

This section lists the files that are available in the SD Card Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/sdcard`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_sdcard.h	This file provides the interface definitions of the SD Card driver

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/drv_sdcard.c	This file contains the core implementation of the SD Card driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The SD Card Driver Library depends on the following modules:

- SPI Driver Library
- Clock System Service Library
- Interrupt System Service Library
- Ports System Service Library
- Timer System Service Library
- Timer Driver Library

Library Interface

a) System Level Functions

	Name	Description
	DRV_SDCARD_Initialize	Initializes the SD Card driver. **Implementation:** Dynamic
	DRV_SDCARD_Deinitialize	Deinitializes the specified instance of the SD Card driver module. **Implementation:** Dynamic
	DRV_SDCARD_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings. **Implementation:** Dynamic
	DRV_SDCARD_Status	Provides the current status of the SD Card driver module. **Implementation:** Dynamic
	DRV_SDCARD_Tasks	Maintains the driver's state machine. **Implementation:** Dynamic

b) Client Level Functions

	Name	Description
	DRV_SDCARD_Close	Closes an opened-instance of the SD Card driver. **Implementation:** Dynamic
	DRV_SDCARD_Open	Opens the specified SD Card driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_SDCARD_Read	Reads blocks of data from the specified block address of the SD Card.
	DRV_SDCARD_Write	Writes blocks of data starting at the specified address of the SD Card.
	DRV_SDCARD_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

c) Status Functions

	Name	Description
	DRV_SDCARD_IsAttached	Returns the physical attach status of the SD Card.
	DRV_SDCARD_IsWriteProtected	Returns the write protect status of the SDCARD.
	DRV_SDCARD_CommandStatus	Gets the current status of the command.
	DRV_SDCARD_GeometryGet	Returns the geometry of the device.

d) Data Types and Constants

	Name	Description
	DRV_SDCARD_INDEX_0	SD Card driver index definitions
	DRV_SDCARD_INDEX_COUNT	Number of valid SD Card driver indices
	DRV_SDCARD_INIT	Contains all the data necessary to initialize the SD Card device
	_DRV_SDCARD_INIT	Contains all the data necessary to initialize the SD Card device
	SDCARD_DETECTION_LOGIC	Defines the different system events
	SDCARD_MAX_LIMIT	Maximum allowed SD card instances
	DRV_SDCARD_INDEX_1	This is macro DRV_SDCARD_INDEX_1.
	DRV_SDCARD_INDEX_2	This is macro DRV_SDCARD_INDEX_2.
	DRV_SDCARD_INDEX_3	This is macro DRV_SDCARD_INDEX_3.
	DRV_SDCARD_COMMAND_HANDLE_INVALID	SDCARD Driver's Invalid Command Handle.
	DRV_SDCARD_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_SDCARD_COMMAND_STATUS	Identifies the possible events that can result from a request.
	DRV_SDCARD_EVENT	Identifies the possible events that can result from a request.

	DRV_SDCARD_EVENT_HANDLER	Pointer to a SDCARDDriver Event handler function

Description

This section describes the Application Programming Interface (API) functions of the SD Card Driver.

Refer to each section for a detailed description.

a) System Level Functions

DRV_SDCARD_Initialize Function

Initializes the SD Card driver.

Implementation: Dynamic

File

drv_sdcard.h

C

```
SYS_MODULE_OBJ DRV_SDCARD_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT *const init);
```

Returns

If successful, returns a valid handle to a driver object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the SD Card driver, making it ready for clients to open and use the driver.

Remarks

This routine must be called before any other SD Card routine is called.

This routine should only be called once during system initialization unless DRV_SDCARD_Deinitialize is called to deinitialize the driver instance.

This routine will NEVER block for hardware access. If the operation requires time to allow the hardware to reinitialize, it will be reported by the DRV_SDCARD_Status operation. The system must use DRV_SDCARD_Status to find out when the driver is in the ready state.

Preconditions

None.

Example

```
DRV_SDCARD_INIT     init;
SYS_MODULE_OBJ      objectHandle;

// Populate the SD Card initialization structure

objectHandle = DRV_SDCARD_Initialize(DRV_SDCARD_INDEX_0, (SYS_MODULE_INIT*)&init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Index for the driver instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

```
SYS_MODULE_OBJ DRV_SDCARD_Initialize
(
const       SYS_MODULE_INDEX index,
const       SYS_MODULE_INIT  * const init
);
```

DRV_SDCARD_Deinitialize Function

Deinitializes the specified instance of the SD Card driver module.

Implementation: Dynamic

File

drv_sdcard.h

C

```
void DRV_SDCARD_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SD Card driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This routine will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_SDCARD_Status operation. The system has to use DRV_SDCARD_Status to check if the de-initialization is complete.

Preconditions

Function DRV_SDCARD_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      objectHandle;     //  Returned from DRV_SDCARD_Initialize
SYS_STATUS          status;

DRV_SDCARD_Deinitialize(objectHandle);

status = DRV_SDCARD_Status(objectHandle);
if (SYS_MODULE_UNINITIALIZED == status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SDCARD_Initialize routine.

Function

void DRV_SDCARD_Deinitialize

(

SYS_MODULE_OBJ object

);

DRV_SDCARD_Reinitialize Function

Reinitializes the driver and refreshes any associated hardware settings.

Implementation: Dynamic

File

drv_sdcard.h

C

```
void DRV_SDCARD_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None

Description

This routine reinitializes the driver and refreshes any associated hardware settings using the given initialization data, but it will not interrupt any ongoing operations.

Remarks

This function can be called multiple times to reinitialize the module.

This operation can be used to refresh any supported hardware registers as specified by the initialization data or to change the power state of the module.

This routine will NEVER block for hardware access. If the operation requires time to allow the hardware to reinitialize, it will be reported by the DRV_SDCARD_Status operation. The system must use DRV_SDCARD_Status to find out when the driver is in the ready state.

Preconditions

Function DRV_SDCARD_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
DRV_SDCARD_INIT     init;
SYS_MODULE_OBJ      objectHandle;  //  Returned from DRV_SDCARD_Initialize

// Update the required fields of the SD Card initialization structure

DRV_SDCARD_Reinitialize (objectHandle, (SYS_MODULE_INIT*)&init);
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SDCARD_Initialize routine
init	Pointer to the initialization data structure

Function

void DRV_SDCARD_Reinitialize

(

SYS_MODULE_OBJ object,

const SYS_MODULE_INIT * const init

);

DRV_SDCARD_Status Function

Provides the current status of the SD Card driver module.

Implementation: Dynamic

File

drv_sdcard.h

C

```
SYS_STATUS DRV_SDCARD_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Note Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Description

This routine provides the current status of the SD Card driver module.

Remarks

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR

This operation can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, then a previous operation has not yet completed. If the status operation returns SYS_STATUS_READY, then it indicates that all previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This routine will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

Function DRV_SDCARD_Initialize must have been called before calling this

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SDCARD_Initialize
SYS_STATUS          status;

status = DRV_SDCARD_Status(object);

if (SYS_MODULE_UNINITIALIZED == status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
else if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SDCARD_Initialize routine

Function

```
SYS_STATUS DRV_SDCARD_Status
(
    SYS_MODULE_OBJ object
);
```

DRV_SDCARD_Tasks Function

Maintains the driver's state machine.

Implementation: Dynamic

File

drv_sdcard.h

C

```
void DRV_SDCARD_Tasks(SYS_MODULE_OBJ object);
```

Returns

None

Description

This routine is used to maintain the driver's internal state machine.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_SDCARD_Initialize routine must have been called for the specified SDCARD driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SDCARD_Initialize
```

```
while (true)
{
    DRV_SDCARD_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_SDCARD_Initialize)

Function

void DRV_SDCARD_Tasks
(
    SYS_MODULE_OBJ object
);

b) Client Level Functions

DRV_SDCARD_Close Function

Closes an opened-instance of the SD Card driver.

Implementation: Dynamic

File

drv_sdcard.h

C

```
void DRV_SDCARD_Close(DRV_HANDLE handle);
```

Returns

None

Description

This routine closes an opened-instance of the SD Card driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_SDCARD_Open before the caller may use the driver again.

If DRV_IO_INTENT_BLOCKING was requested and the driver was built appropriately to support blocking behavior call may block until the operation is complete.

If DRV_IO_INTENT_NON_BLOCKING request the driver client can call the DRV_SDCARD_Status operation to find out when the module is in the ready state (the handle is no longer valid).

Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_SDCARD_Initialize routine must have been called for the specified SD Card driver instance.

DRV_SDCARD_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_SDCARD_Open

DRV_SDCARD_Close (handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_SDCARD_Close
(
DRV_HANDLE handle
);

DRV_SDCARD_Open Function

Opens the specified SD Card driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_sdcard.h

C

```
DRV_HANDLE DRV_SDCARD_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID.

Description

This routine opens the specified SD Card driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The handle returned is valid until the DRV_SDCARD_Close routine is called.

This routine will NEVER block waiting for hardware.

If the DRV_IO_INTENT_BLOCKING is requested and the driver was built appropriately to support blocking behavior, then other client-level operations may block waiting on hardware until they are complete.

If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID.

Preconditions

Function DRV_SDCARD_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_SDCARD_Open (DRV_SDCARD_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);

if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_SDCARD_Open
(
const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT intent
);

DRV_SDCARD_Read Function

Reads blocks of data from the specified block address of the SD Card.

File

drv_sdcard.h

C

```
void DRV_SDCARD_Read(DRV_HANDLE handle, DRV_SDCARD_COMMAND_HANDLE * commandHandle, void * targetBuffer,
uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_SDCARD_COMMAND_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking read operation for reading blocks of data from the SD Card. The function returns with a valid buffer handle in the commandHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SDCARD_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the driver handle is invalid
- if the target buffer pointer is NULL
- if the number of blocks to be read is zero or more than the actual number of blocks available
- if a buffer object could not be allocated to the request
- if the client opened the driver in write only mode

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SDCARD_EVENT_COMMAND_COMPLETE event if the buffer was processed successfully or DRV_SDCARD_EVENT_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

None.

Preconditions

The DRV_SDCARD_Initialize routine must have been called for the specified SDCARD driver instance.

DRV_SDCARD_Open must have been called with DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE as the ioIntent to obtain a valid opened device handle.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = 0x00;
uint32_t    nBlock = 2;
DRV_SDCARD_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySDCARDHandle is the handle returned
// by the DRV_SDCARD_Open function.

DRV_SDCARD_Read(mySDCARDHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_SDCARD_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}
else
{
    // Read Successful
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle

targetBuffer	Buffer into which the data read from the SD Card will be placed
blockStart	Start block address of the SD Card from where the read should begin.
nBlock	Total number of blocks to be read.

Function

void DRV_SDCARD_Read
(
const DRV_HANDLE handle,
DRV_SDCARD_COMMAND_HANDLE * commandHandle,
void * targetBuffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SDCARD_Write Function

Writes blocks of data starting at the specified address of the SD Card.

File

drv_sdcard.h

C

```
void DRV_SDCARD_Write(DRV_HANDLE handle, DRV_SDCARD_COMMAND_HANDLE * commandHandle, void * sourceBuffer,
uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_SDCARD_COMMAND_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking write operation for writing blocks of data to the SD Card. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SDCARD_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer object could not be allocated to the request
- if the source buffer pointer is NULL
- if the client opened the driver for read only
- if the number of blocks to be written is either zero or more than the number of blocks actually available
- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SDCARD_EVENT_COMMAND_COMPLETE event if the buffer was processed successfully or DRV_SDCARD_EVENT_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

None.

Preconditions

The DRV_SDCARD_Initialize() routine must have been called for the specified SDCARD driver instance.

DRV_SDCARD_Open() routine must have been called to obtain a valid opened device handle. DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified as a parameter to this routine.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = 0x00;
uint32_t    nBlock = 2;
DRV_SDCARD_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySDCARDHandle is the handle returned
```

```
// by the DRV_SDCARD_Open function.

// Client registers an event handler with driver

DRV_SDCARD_EventHandlerSet(mySDCARDHandle, APP_SDCARDEventHandler, (uintptr_t)&myAppObj);

DRV_SDCARD_Write(mySDCARDHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_SDCARD_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_SDCARDEventHandler(DRV_SDCARD_EVENT event,
        DRV_SDCARD_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SDCARD_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SDCARD_EVENT_COMMAND_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
sourceBuffer	The source buffer containing data to be programmed to the SD Card.
blockStart	Start block address of SD Card where the writes should begin.
nBlock	Total number of blocks to be written.

Function

```
void DRV_SDCARD_Write
(
const       DRV_HANDLE handle,
    DRV_SDCARD_COMMAND_HANDLE * commandHandle,
void * sourceBuffer,
uint32_t blockStart,
uint32_t nBlock
);
```

DRV_SDCARD_EventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

File

drv_sdcard.h

C

```
void DRV_SDCARD_EventHandlerSet(const DRV_HANDLE handle, const void * eventHandler, const uintptr_t
context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client queues a request for a read or a write operation, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any read or write operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_SDCARD_Initialize() routine must have been called for the specified SDCARD driver instance.

The DRV_SDCARD_Open() routine must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_SDCARD_COMMAND_HANDLE commandHandle;

// drvSDCARDHandle is the handle returned
// by the DRV_SDCARD_Open function.

// Client registers an event handler with driver. This is done once.

DRV_SDCARD_EventHandlerSet(drvSDCARDHandle, APP_SDCARDEventHandler, (uintptr_t)&myAppObj);

DRV_SDCARD_Read(drvSDCARDHandle, &commandHandle, &myBuffer, blockStart, nBlock);

if(DRV_SDCARD_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_SDCARDEventHandler(DRV_SDCARD_EVENT event,
        DRV_SDCARD_COMMAND_HANDLE handle, uintptr_t context)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_SDCARD_EVENT_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SDCARD_EVENT_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
```

```
            break;
        }
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_SDCARD_EventHandlerSet

(

const DRV_HANDLE handle,

const void * eventHandler,

const uintptr_t context

);

c) Status Functions

DRV_SDCARD_IsAttached Function

Returns the physical attach status of the SD Card.

File

drv_sdcard.h

C

```
bool DRV_SDCARD_IsAttached(const DRV_HANDLE handle);
```

Returns

Returns false if the handle is invalid otherwise returns the attach status of the SD Card. Returns true if the SD Card is attached and initialized by the SDCARD driver otherwise returns false.

Description

This function returns the physical attach status of the SD Card.

Remarks

None.

Preconditions

The DRV_SDCARD_Initialize() routine must have been called for the specified SDCARD driver instance.

The DRV_SDCARD_Open() routine must have been called to obtain a valid opened device handle.

Example

```
bool isSDCARDAttached;
isSDCARDAttached = DRV_SDCARD_isAttached(drvSDCARDHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_SDCARD_IsAttached

(

const DRV_HANDLE handle

);

DRV_SDCARD_IsWriteProtected Function

Returns the write protect status of the SDCARD.

File

drv_sdcard.h

C

```
bool DRV_SDCARD_IsWriteProtected(const DRV_HANDLE handle);
```

Returns

Returns true if the attached SD Card is write protected. Returns false if the handle is not valid, or if the SD Card is not write protected.

Description

This function returns true if the SD Card is write protected otherwise it returns false.

Remarks

None.

Preconditions

The DRV_SDCARD_Initialize() routine must have been called for the specified SDCARD driver instance.

The DRV_SDCARD_Open() routine must have been called to obtain a valid opened device handle.

Example

```
bool isWriteProtected;
isWriteProtected = DRV_SDCARD_IsWriteProtected(drvSDCARDHandle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_SDCARD_IsWriteProtected

(

const DRV_HANDLE handle

);

DRV_SDCARD_CommandStatus Function

Gets the current status of the command.

File

drv_sdcard.h

C

```
DRV_SDCARD_COMMAND_STATUS DRV_SDCARD_CommandStatus(const DRV_HANDLE handle, const DRV_SDCARD_COMMAND_HANDLE
commandHandle);
```

Returns

A DRV_SDCARD_COMMAND_STATUS value describing the current status of the command. Returns DRV_SDCARD_COMMAND_HANDLE_INVALID if the client handle or the command handle is not valid.

Description

This routine gets the current status of the command. The application must use this routine where the status of a scheduled command needs to be polled on. The function may return DRV_SDCARD_COMMAND_HANDLE_INVALID in a case where the command handle has expired. A command handle expires when the internal buffer object is re-assigned to another read or write request. It is recommended that this function be called regularly in order to track the command status correctly.

The application can alternatively register an event handler to receive read or write operation completion events.

Remarks

This routine will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_SDCARD_Initialize() routine must have been called.

The DRV_SDCARD_Open() must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE                      handle;         // Returned from DRV_SDCARD_Open
DRV_SDCARD_COMMAND_HANDLE       commandHandle;
DRV_SDCARD_COMMAND_STATUS       status;

status = DRV_SDCARD_CommandStatus(handle, commandHandle);
if(status == DRV_SDCARD_COMMAND_COMPLETED)
{
    // Operation Done
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

DRV_SDCARD_COMMAND_STATUS DRV_SDCARD_CommandStatus

(

const DRV_HANDLE handle,

const DRV_SDCARD_COMMAND_HANDLE commandHandle

);

DRV_SDCARD_GeometryGet Function

Returns the geometry of the device.

File

drv_sdcard.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_SDCARD_GeometryGet(const DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Pointer to structure which holds the media geometry information.

Description

This API gives the following geometrical details of the SD Card.

- Media Property
- Number of Read/Write/Erase regions in the SD Card
- Number of Blocks and their size in each region of the device

Remarks

None.

Preconditions

The DRV_SDCARD_Initialize() routine must have been called for the specified SDCARD driver instance.

The DRV_SDCARD_Open() routine must have been called to obtain a valid opened device handle.

Example

```
SYS_FS_MEDIA_GEOMETRY * SDCARDGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalSize;

SDCARDGeometry = DRV_SDCARD_GeometryGet(SDCARDOpenHandle1);
```

```
readBlockSize  = SDCARDGeometry->geometryTable->blockSize;
nReadBlocks = SDCARDGeometry->geometryTable->numBlocks;
nReadRegions = SDCARDGeometry->numReadRegions;

writeBlockSize  = (SDCARDGeometry->geometryTable +1)->blockSize;
eraseBlockSize  = (SDCARDGeometry->geometryTable +2)->blockSize;

totalSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY * DRV_SDCARD_GeometryGet
(
const DRV_HANDLE handle
);

d) Data Types and Constants

DRV_SDCARD_INDEX_0 Macro

SD Card driver index definitions

File

drv_sdcard.h

C

```
#define DRV_SDCARD_INDEX_0 0
```

Description

SD Card Driver Module Index Numbers

These constants provide SD Card driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_SDCARD_Initialize and DRV_SDCARD_Open routines to identify the driver instance in use.

DRV_SDCARD_INDEX_COUNT Macro

Number of valid SD Card driver indices

File

drv_sdcard.h

C

```
#define DRV_SDCARD_INDEX_COUNT DRV_SDCARD_INDEX_MAX
```

Description

SD Card Driver Module Index Count

This constant identifies number of valid SD Card driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from part-specific header files defined as part of the peripheral libraries.

DRV_SDCARD_INIT Structure

Contains all the data necessary to initialize the SD Card device

File

drv_sdcard.h

C

```
typedef struct _DRV_SDCARD_INIT {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX spiIndex;
  SPI_MODULE_ID spiId;
  CLK_BUSES_PERIPHERAL spiClk;
  uint32_t sdcardSpeedHz;
  SDCARD_DETECTION_LOGIC sdCardPinActiveLogic;
  PORTS_CHANNEL cardDetectPort;
  PORTS_BIT_POS cardDetectBitPosition;
  PORTS_CHANNEL writeProtectPort;
  PORTS_BIT_POS writeProtectBitPosition;
  PORTS_CHANNEL chipSelectPort;
  PORTS_BIT_POS chipSelectBitPosition;
} DRV_SDCARD_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX spiIndex;	SPI driver index
SPI_MODULE_ID spiId;	Identifies peripheral (PLIB-level) ID
CLK_BUSES_PERIPHERAL spiClk;	Peripheral clock used by the SPI
uint32_t sdcardSpeedHz;	SD card communication speed
SDCARD_DETECTION_LOGIC sdCardPinActiveLogic;	SD Card Pin Detection Logic
PORTS_CHANNEL cardDetectPort;	Card detect port
PORTS_BIT_POS cardDetectBitPosition;	Card detect pin
PORTS_CHANNEL writeProtectPort;	Write protect port
PORTS_BIT_POS writeProtectBitPosition;	Write protect pin
PORTS_CHANNEL chipSelectPort;	Chip select port
PORTS_BIT_POS chipSelectBitPosition;	Chip select pin

Description

SD Card Device Driver Initialization Data

This structure contains all the data necessary to initialize the SD Card device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_SDCARD_Initialize routine.

SDCARD_DETECTION_LOGIC Enumeration

Defines the different system events

File

drv_sdcard.h

C

```
typedef enum {
  SDCARD_DETECTION_LOGIC_ACTIVE_LOW,
  SDCARD_DETECTION_LOGIC_ACTIVE_HIGH
} SDCARD_DETECTION_LOGIC;
```

Members

Members	Description
SDCARD_DETECTION_LOGIC_ACTIVE_LOW	The media event is SD Card attach

SDCARD_DETECTION_LOGIC_ACTIVE_HIGH	The media event is SD Card detach

Description

System events

This enum defines different system events.

Remarks

None.

SDCARD_MAX_LIMIT Macro

Maximum allowed SD card instances

File

drv_sdcard.h

C

```
#define SDCARD_MAX_LIMIT 2
```

Description

SD Card Driver Maximum allowed limit

This constant identifies number of valid SD Card driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from part-specific header files defined as part of the peripheral libraries.

DRV_SDCARD_INDEX_1 Macro

File

drv_sdcard.h

C

```
#define DRV_SDCARD_INDEX_1 1
```

Description

This is macro DRV_SDCARD_INDEX_1.

DRV_SDCARD_INDEX_2 Macro

File

drv_sdcard.h

C

```
#define DRV_SDCARD_INDEX_2 2
```

Description

This is macro DRV_SDCARD_INDEX_2.

DRV_SDCARD_INDEX_3 Macro

File

drv_sdcard.h

C

```
#define DRV_SDCARD_INDEX_3 3
```

Description

This is macro DRV_SDCARD_INDEX_3.

DRV_SDCARD_COMMAND_HANDLE_INVALID Macro

SDCARD Driver's Invalid Command Handle.

File

drv_sdcard.h

C

```
#define DRV_SDCARD_COMMAND_HANDLE_INVALID SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE_INVALID
```

Description

SDCARD Driver Invalid Command Handle.

This value defines the SDCARD Driver Invalid Command Handle. This value is returned by read or write routines when the command request was not accepted.

Remarks

None.

DRV_SDCARD_COMMAND_HANDLE Type

Handle identifying commands queued in the driver.

File

drv_sdcard.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_SDCARD_COMMAND_HANDLE;
```

Description

SDCARD Driver command handle.

A command handle is returned by a call to the Read or Write functions. This handle allows the application to track the completion of the operation. This command handle is also returned to the client along with the event that has occurred with respect to the command. This allows the application to connect the event to a specific command in case where multiple commands are queued.

The command handle associated with the command request expires when the client has been notified of the completion of the command (after event handler function that notifies the client returns) or after the command has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_SDCARD_COMMAND_STATUS Enumeration

Identifies the possible events that can result from a request.

File

drv_sdcard.h

C

```
typedef enum {
  DRV_SDCARD_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED,
  DRV_SDCARD_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED,
  DRV_SDCARD_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS,
  DRV_SDCARD_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN
} DRV_SDCARD_COMMAND_STATUS;
```

Members

Members	Description
DRV_SDCARD_COMMAND_COMPLETED = SYS_FS_MEDIA_COMMAND_COMPLETED	Done OK and ready
DRV_SDCARD_COMMAND_QUEUED = SYS_FS_MEDIA_COMMAND_QUEUED	Scheduled but not started

DRV_SDCARD_COMMAND_IN_PROGRESS = SYS_FS_MEDIA_COMMAND_IN_PROGRESS	Currently being in transfer
DRV_SDCARD_COMMAND_ERROR_UNKNOWN = SYS_FS_MEDIA_COMMAND_UNKNOWN	Unknown Command

Description

SDCARD Driver Events

This enumeration identifies the possible events that can result from a read or a write request made by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_SDCARD_EventHandlerSet function when a request is completed.

DRV_SDCARD_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_sdcard.h

C

```
typedef enum {
  DRV_SDCARD_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_SDCARD_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR
} DRV_SDCARD_EVENT;
```

Members

Members	Description
DRV_SDCARD_EVENT_COMMAND_COMPLETE = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE	Operation has been completed successfully.
DRV_SDCARD_EVENT_COMMAND_ERROR = SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR	There was an error during the operation

Description

SDCARD Driver Events

This enumeration identifies the possible events that can result from a read or a write request issued by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_SDCARD_EventHandlerSet function when a request is completed.

DRV_SDCARD_EVENT_HANDLER Type

Pointer to a SDCARDDriver Event handler function

File

drv_sdcard.h

C

```
typedef SYS_FS_MEDIA_EVENT_HANDLER DRV_SDCARD_EVENT_HANDLER;
```

Returns

None.

Description

SDCARD Driver Event Handler Function Pointer

This data type defines the required function signature for the SDCARD event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event calls back from the driver.

The parameters and return values are described here and a partial example implementation is provided.

Remarks

If the event is DRV_SDCARD_EVENT_COMMAND_COMPLETE, it means that the write or a erase operation was completed successfully.

If the event is DRV_SDCARD_EVENT_COMMAND_ERROR, it means that the scheduled operation was not completed successfully.

The context parameter contains the handle to the client context, provided at the time the event handling function was registered using the DRV_SDCARD_EventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the read/write/erase request.

Example

```
void APP_MySDCARDEventHandler
(
    DRV_SDCARD_EVENT event,
    DRV_SDCARD_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_SDCARD_EVENT_COMMAND_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_SDCARD_EVENT_COMMAND_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle returned from the Read/Write requests
context	Value identifying the context of the application that registered the event handling function

Files

Files

Name	Description
drv_sdcard.h	SD Card Device Driver Interface File
drv_sdcard_config_template.h	SD Card driver configuration definitions template

Description

This section lists the source and header files used by the SD Card Driver Library.

drv_sdcard.h

SD Card Device Driver Interface File

Enumerations

	Name	Description
	DRV_SDCARD_COMMAND_STATUS	Identifies the possible events that can result from a request.
	DRV_SDCARD_EVENT	Identifies the possible events that can result from a request.
	SDCARD_DETECTION_LOGIC	Defines the different system events

Functions

	Name	Description
	DRV_SDCARD_Close	Closes an opened-instance of the SD Card driver. **Implementation:** Dynamic
	DRV_SDCARD_CommandStatus	Gets the current status of the command.
	DRV_SDCARD_Deinitialize	Deinitializes the specified instance of the SD Card driver module. **Implementation:** Dynamic
	DRV_SDCARD_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed.
	DRV_SDCARD_GeometryGet	Returns the geometry of the device.
	DRV_SDCARD_Initialize	Initializes the SD Card driver. **Implementation:** Dynamic
	DRV_SDCARD_IsAttached	Returns the physical attach status of the SD Card.
	DRV_SDCARD_IsWriteProtected	Returns the write protect status of the SDCARD.
	DRV_SDCARD_Open	Opens the specified SD Card driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_SDCARD_Read	Reads blocks of data from the specified block address of the SD Card.
	DRV_SDCARD_Reinitialize	Reinitializes the driver and refreshes any associated hardware settings. **Implementation:** Dynamic
	DRV_SDCARD_Status	Provides the current status of the SD Card driver module. **Implementation:** Dynamic
	DRV_SDCARD_Tasks	Maintains the driver's state machine. **Implementation:** Dynamic
	DRV_SDCARD_Write	Writes blocks of data starting at the specified address of the SD Card.

Macros

	Name	Description
	DRV_SDCARD_COMMAND_HANDLE_INVALID	SDCARD Driver's Invalid Command Handle.
	DRV_SDCARD_INDEX_0	SD Card driver index definitions
	DRV_SDCARD_INDEX_1	This is macro DRV_SDCARD_INDEX_1.
	DRV_SDCARD_INDEX_2	This is macro DRV_SDCARD_INDEX_2.
	DRV_SDCARD_INDEX_3	This is macro DRV_SDCARD_INDEX_3.
	DRV_SDCARD_INDEX_COUNT	Number of valid SD Card driver indices
	SDCARD_MAX_LIMIT	Maximum allowed SD card instances

Structures

	Name	Description
	_DRV_SDCARD_INIT	Contains all the data necessary to initialize the SD Card device
	DRV_SDCARD_INIT	Contains all the data necessary to initialize the SD Card device

Types

	Name	Description
	DRV_SDCARD_COMMAND_HANDLE	Handle identifying commands queued in the driver.
	DRV_SDCARD_EVENT_HANDLER	Pointer to a SDCARDDriver Event handler function

Description

SD Card Device Driver Interface

The SD Card device driver provides a simple interface to manage the "SD Card" peripheral. This file defines the interface definitions and prototypes for the SD Card driver.

File Name

drv_sdcard.h

Company

Microchip Technology Inc.

drv_sdcard_config_template.h

SD Card driver configuration definitions template

Macros

	Name	Description
	DRV_SDCARD_CLIENTS_NUMBER	Selects the miximum number of clients
	DRV_SDCARD_ENABLE_WRITE_PROTECT_CHECK	Enable SD Card write protect check.
	DRV_SDCARD_INDEX_MAX	SD Card Static Index selection
	DRV_SDCARD_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver
	DRV_SDCARD_POWER_STATE	Defines an override of the power state of the SD Card driver.
	DRV_SDCARD_SYS_FS_REGISTER	Register to use with the File system

Description

SD Card Driver Configuration Definitions for the template version

These definitions statically define the driver's mode of operation.

File Name

drv_sdcard_config_template.h

Company

Microchip Technology Inc.

SPI Driver Library

This section describes the Serial Peripheral Interface (SPI) Driver Library.

Introduction

This library provides an interface to manage the Serial Peripheral Interface (SPI) module on the Microchip family of microcontrollers in different modes of operation.

Description

The SPI module is a full duplex synchronous serial interface useful for communicating with other peripherals or microcontrollers in master/slave relationship and it can transfer data over short distances at high speeds. The peripheral devices may be serial EEPROMs, shift registers, display drivers, analog-to-digital converters, etc. The SPI module is compatible with Motorola's SPI and SIOP interfaces.

During data transfer devices can work either in master or in Slave mode. The source of synchronization is the system clock, which is generated by the master. The SPI module allows one or more slave devices to be connected to a single master device via the same bus.

The SPI serial interface consists of four pins, which are further sub-divided into data and control lines:

Data Lines:

• MOSI – Master Data Output, Slave Data Input

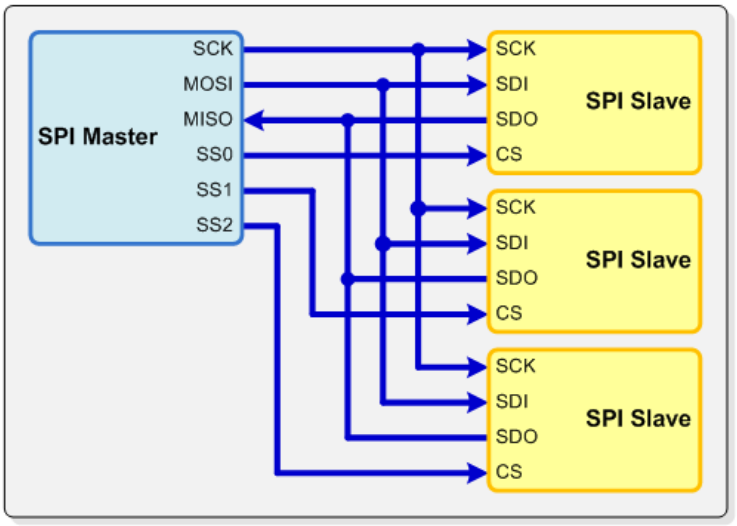
• MISO – Master Data Input, Slave Data Output

Control Lines:

• SCLK – Serial Clock

• /SS – Slave Select (no addressing)

SPI Master-Slave Relationship



The SPI module can be configured to operate using two, three, or four pins. In the 3-pin mode, the Slave Select line is not used. In the 2-pin mode, both the MOSI and /SS lines are not used.

Note: Third-party trademarks are property of their respective owners. Refer to the MPLAB Harmony *Software License Agreement* for complete licensing information. A copy of this agreement is available in the `<install-dir>/doc` folder of your MPLAB Harmony installation.

Using the Library

This topic describes the basic architecture of the SPI Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_spi.h`

The interface to the SPI Driver library is defined in the `drv_spi.h` header file. Any C language source (`.c`) file that uses the SPI Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the Driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the SPI Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

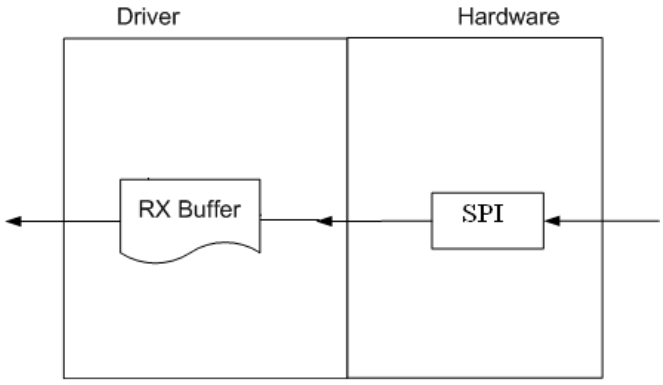
Description

Different types of SPIs are available on Microchip microcontrollers. Some have an internal buffer mechanism and some do not. The buffer depth varies across part families. The SPI driver abstracts out these differences and provides a unified model for data transfer across different types of SPIs available.

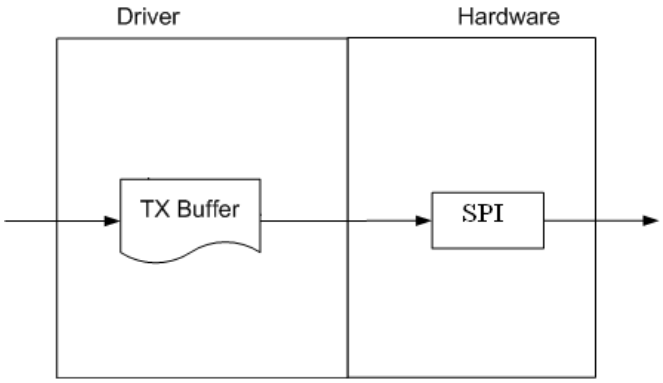
Both transmitter and receiver provides a buffer in the driver which transmits and receives data to/from the hardware. The SPI driver provides a set of interfaces to perform the read and the write.

The following diagrams illustrate the model used by the SPI driver for transmitter and receiver.

Receiver Abstraction Model



Transmitter Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SPI module.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks and status functions.
Client Setup Functions	Provides open, close, status and other setup functions.
Data Transfer Functions	Provides data transfer functions available in the configuration.
Miscellaneous	Provides driver miscellaneous functions, data transfer status function, version identification functions etc.

How the Library Works

The library provides interfaces to support:

- System Functionality
- Client Functionality

Note: Not all modes are available on all devices, please refer to the specific device data sheet to determine the modes that are supported for your device.

System Access

System Initialization and Reinitialization

The system performs the initialization and the reinitialization of the device driver with settings that affect only the instance of the device that is being initialized or reinitialized. During system initialization each instance of the SPI module would be initialized with the following configuration

settings (either passed dynamically at run time using DRV_SPI_INIT or by using Initialization Overrides) that are supported by the specific SPI device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section
- The actual peripheral ID enumerated as the PLIB level module ID (e.g., SPI_ID_2)
- Defining the respective interrupt sources for TX, RX, and Error Interrupt

The DRV_SPI_Initialize API returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the Initialize interface would be used by the other system interfaces like DRV_SPI_Deinitialize, DRV_SPI_Status, and DRV_SPI_Tasks.

Note: The system initialization and the reinitialization settings, only affect the instance of the peripheral that is being initialized or reinitialized.

Example:

```
DRV_SPI_INIT               spiInitData;
SYS_MODULE_OBJ             objectHandle;

    spiInitData.moduleInit.value    = SYS_MODULE_POWER_RUN_FULL;
    spiInitData.spiId               = SPI_ID_1;
    spiInitData.taskMode            = DRV_SPI_TASK_MODE_POLLED;
    spiInitData.spiMode             = DRV_SPI_MODE_MASTER;
    spiInitData.spiProtocolType     = DRV_SPI_PROTOCOL_TYPE_STANDARD;
    spiInitData.commWidth           = SPI_COMMUNICATION_WIDTH_8BITS;
    spiInitData.baudRate            = 5000;
    spiInitData.bufferType          = DRV_SPI_BUFFER_TYPE_STANDARD;
                                      // It is highly recommended to set this to
                                      // DRV_SPI_BUFFER_TYPE_ENHANCED for hardware
                                      // that supports it
    spiInitData.inputSamplePhase    = SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE;
    spiInitData.clockMode           = DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_RISE;
    spiInitData.txInterruptSource   = INT_SOURCE_SPI_1_TRANSMIT;
    spiInitData.rxInterruptSource   = INT_SOURCE_SPI_1_RECEIVE;
    spiInitData.errInterruptSource  = INT_SOURCE_SPI_1_ERROR;
    spiInitData.queueSize = 10;
    spiInitData.jobQueueReserveSize = 1;

    objectHandle = DRV_SPI_Initialize(DRV_SPI_INDEX_1, (SYS_MODULE_INIT*)&spiInitData);
    if (SYS_MODULE_OBJ_INVALID == objectHandle)
   {
    // Handle error
  }
```

Tasks Routine

The system will either call DRV_SPI_Tasks, from System Task Service (in a polled environment) or DRV_SPI_Tasks will be called from the ISR of the SPI.

Client Access

General Client Operation

For the application to start using an instance of the module, it must call the DRV_SPI_Open function. This provides the configuration required to open the SPI instance for operation. If the driver is deinitialized using the function DRV_SPI_Deinitialize, the application must call the DRV_SPI_Open function again to set up the instance of the SPI.

For the various options available for IO_INTENT, please refer to **Data Types and Constants** in the Library Interface section.

After a client instance is opened, DRV_SPI_ClientConfigure can be called to set a client-specific bps, OperationStarting and OperationEnded callbacks. The OperationStarting callback will be called before the first bit is put onto the SPI bus, allowing for the slave select line to be toggled to active. The OperationEnded callback will be called after the last bit is received, allowing for the slave select line to be toggled to inactive. These two callbacks will be called from the ISR, if the SPI driver is operating in ISR mode, care should be taken that they do the minimum needed. For example, OSAL calls make cause exceptions in ISR context.

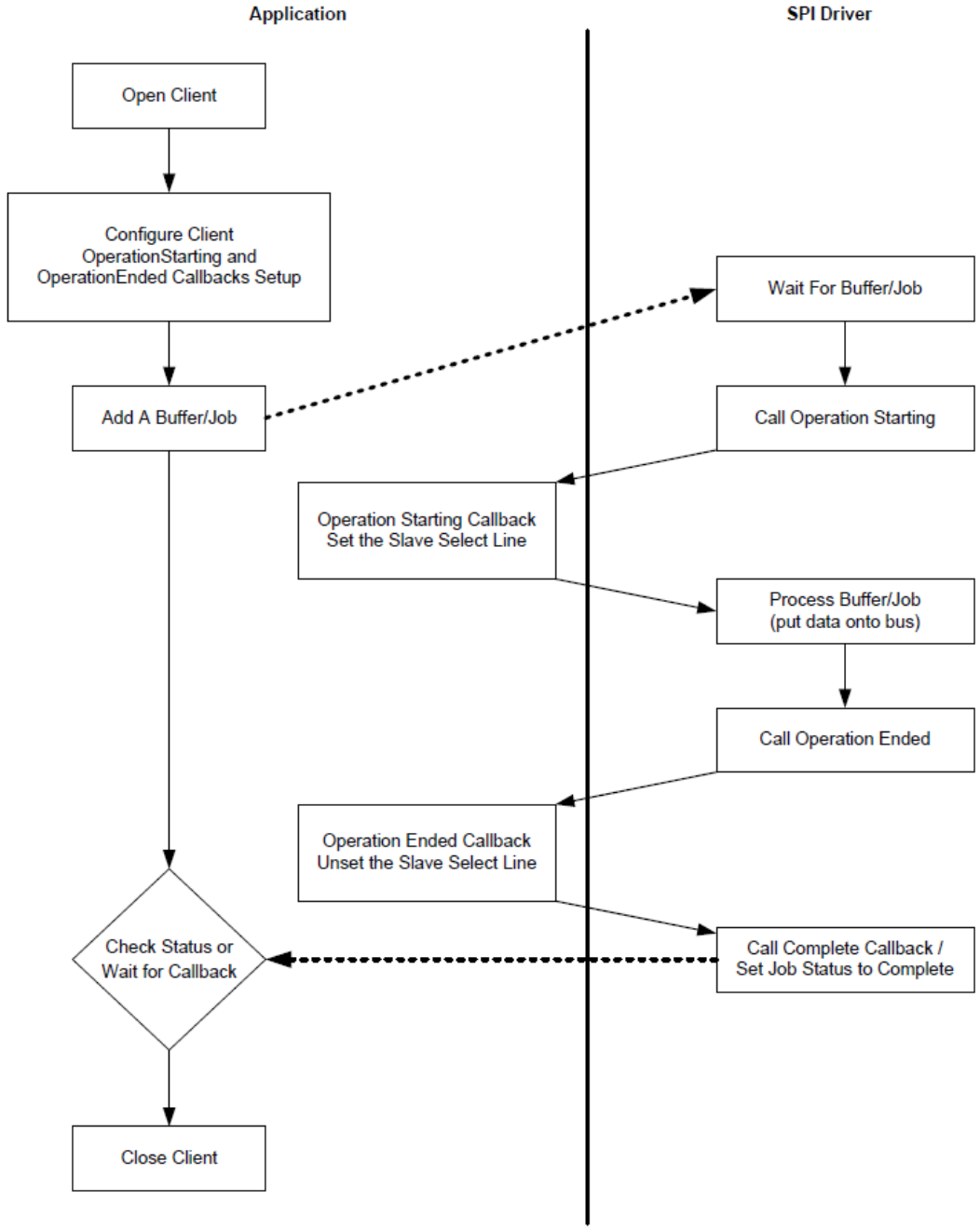
Example:

```
DRV_HANDLE handle;

// Configure the instance DRV_SPI_INDEX_1 with the configuration
handle = DRV_SPI_Open(DRV_SPI_INDEX_1, DRV_IO_INTENT_READWRITE);

if(handle == DRV_HANDLE_INVALID)
{
    // Client cannot open the instance.
}
```

Client Transfer - Core

Client basic functionality provides a extremely basic interface for the driver operation.

The following diagram illustrates the byte/word model used for the data transfer.



Note: It is not necessary to close and reopen the client between multiple transfers.

Client Data Transfer Functionality

Applications using the SPI byte/word functionality, need to perform the following:

1. The system should have completed necessary initialization and the DRV_SPI_Tasks should either be running in polled environment, or in an interrupt environment.
2. Open the driver using DRV_SPI_Open with the necessary intent.
3. Optionally configure the client with DRV_SPI_ClientConfigure to set up OperationStarting and OperationEnded callbacks to handle selecting and deselecting the slave select pin.
4. Add a buffer using the DRV_SPI_BufferAddRead/DRV_SPI_BufferAddWrite/DRV_SPI_BufferAddWriteRead functions. An optional callback can be provided that will be called when the buffer/job is complete.

5. Check for the current transfer status using DRV_SPI_BufferStatus until the transfer progress is DRV_SPI_BUFFER_EVENT_COMPLETE, or wait for the callback to be called. If the SPI driver is configured in Polled more, ensure that DRV_SPI_Tasks is called regularly to handle the buffer/job.
6. The client will be able to close the driver using DRV_SPI_Close when required.

Example:

```
SYS_MODULE_OBJ spiObject;

int main( void )
{
    while ( 1 )
    {
        appTask ();
        DRV_SPI_Tasks(spiObject);
    }
}
void appTask ()
{
    #define MY_BUFFER_SIZE   5
    DRV_HANDLE          handle;    // Returned from DRV_SPI_Open
    char          myBuffer[MY_BUFFER_SIZE] = { 11, 22, 33, 44, 55};
    unsigned int        numBytes;
    DRV_SPI_BUFFER_HANDLE bufHandle;

    // Preinitialize myBuffer with MY_BUFFER_SIZE bytes of valid data.
    while( 1 )
    {
        switch( state )
        {
            case APP_STATE_INIT:
                /* Initialize the SPI Driver */
                spiObject = DRV_SPI_Initialize( DRV_SPI_INDEX_1,
                                                ( SYS_MODULE_INIT * )
                                                  &initConf_1 );

                /* Check for the System Status */
                if( SYS_STATUS_READY != DRV_SPI_Status( spiObject ) )
                    return 0;

                /* Open the Driver */
                handle = DRV_SPI_Open( DRV_SPI_INDEX_1,
                                       DRV_IO_INTENT_EXCLUSIVE );

                /* Enable/Activate the CS */

                /* Update the state to transfer data */
                state = APP_STATE_DATA_PUT;
            break;

            case APP_STATE_DATA_PUT:
                bufHandle = DRV_SPI_BufferAddWrite ( handle, myBuffer,
                                                     5, NULL, NULL );
                /* Update the state to status check */
                state = APP_STATE_DATA_CHECK;
            break;

            case APP_STATE_DATA_CHECK:
                /* Check for the successful data transfer */
                if( DRV_SPI_BUFFER_EVENT_COMPLETE &
                    DRV_SPI_BufferStatus( bufhandle ) )
                {
                    /* Do this repeatedly */
                    state = APP_STATE_DATA_PUT;
                }

            break;
            default:
            break;
        }
    }
```

Configuring the Library

Miscellaneous Configuration

	Name	Description
	DRV_SPI_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver .
	DRV_SPI_CLIENTS_NUMBER	Selects the maximum number of clients.

System Configuration

	Name	Description
	DRV_SPI_16BIT	Controls the compilation of 16 Bit mode
	DRV_SPI_32BIT	Controls the compilation of 32 Bit mode
	DRV_SPI_8BIT	Controls the compilation of 8 Bit mode
	DRV_SPI_DMA	Controls the compilation of DMA support
	DRV_SPI_DMA_DUMMY_BUFFER_SIZE	Controls the size of DMA dummy buffer
	DRV_SPI_DMA_TXFER_SIZE	Controls the size of DMA transfers
	DRV_SPI_EBM	Controls the compilation of Enhanced Buffer Mode mode
	DRV_SPI_ELEMENTS_PER_QUEUE	Controls the number of elements that are allocated.
	DRV_SPI_ISR	Controls the compilation of ISR mode
	DRV_SPI_MASTER	Controls the compilation of master mode
	DRV_SPI_POLLED	Controls the compilation of Polled mode
	DRV_SPI_RM	Controls the compilation of Standard Buffer mode
	DRV_SPI_SLAVE	Controls the compilation of slave mode

Description

The configuration of the SPI driver is based on the file `system_config.h`.

This header file contains the configuration selection for the SPI driver. Based on the selections made, the SPI driver may support the selected features. These configuration settings will apply to all instances of the SPI driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

System Configuration

DRV_SPI_16BIT Macro

Controls the compilation of 16 Bit mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_16BIT 1
```

Description

SPI 16 Bit Mode Enable

This definition controls whether or not 16 Bit mode functionality is built as part of the driver. With it set to 1 then 16 Bit mode will be compiled and commWidth = SPI_COMMUNICATION_WIDTH_16BITS will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert. With this set the BufferAdd functions will only accept buffer sizes of multiples of 2 (16 bit words)

Remarks

Optional definition

DRV_SPI_32BIT Macro

Controls the compilation of 32 Bit mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_32BIT 1
```

Description

SPI 32 Bit Mode Enable

This definition controls whether or not 32 Bit mode functionality is built as part of the driver. With it set to 1 then 32 Bit mode will be compiled and commWidth = SPI_COMMUNICATION_WIDTH_32BITS will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert. With this set the BufferAdd functions will only accept buffer sizes of multiples of 4 (32 bit words)

Remarks

Optional definition

DRV_SPI_8BIT Macro

Controls the compilation of 8 Bit mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_8BIT 1
```

Description

SPI 8 Bit Mode Enable

This definition controls whether or not 8 Bit mode functionality is built as part of the driver. With it set to 1 then 8 Bit mode will be compiled and commWidth = SPI_COMMUNICATION_WIDTH_8BITS will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert.

Remarks

Optional definition

DRV_SPI_DMA Macro

Controls the compilation of DMA support

File

drv_spi_config_template.h

C

```
#define DRV_SPI_DMA 1
```

Description

SPI DMA Enable

This definition controls whether or not DMA functionality is built as part of the driver. With it set to 1 then DMA will be compiled.

Remarks

Optional definition

DRV_SPI_DMA_DUMMY_BUFFER_SIZE Macro

Controls the size of DMA dummy buffer

File

drv_spi_config_template.h

C

```
#define DRV_SPI_DMA_DUMMY_BUFFER_SIZE 256
```

Description

SPI DMA Dummy Buffer Size

This controls the size of the buffer the SPI driver uses to give to the DMA service when it is to send and receive invalid data on the bus. This occurs when the number of bytes to be read are different than the number of bytes transmitted.

Remarks

Optional definition

DRV_SPI_DMA_TXFER_SIZE Macro

Controls the size of DMA transfers

File

drv_spi_config_template.h

C

```
#define DRV_SPI_DMA_TXFER_SIZE 256
```

Description

SPI DMA Transfer Size

This definition controls the maximum number of bytes to transfer per DMA transfer.

Remarks

Optional definition

DRV_SPI_EBM Macro

Controls the compilation of Enhanced Buffer Mode mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_EBM 1
```

Description

SPI Enhanced Buffer Mode Enable (Hardware FIFO)

This definition controls whether or not Enhanced Buffer mode functionality is built as part of the driver. With it set to 1 then enhanced buffer mode will be compiled and bufferType = DRV_SPI_BUFFER_TYPE_ENHANCED will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert. This mode is not available on all PIC32s. Trying to use this mode on PICMX3XX/4XX will cause compile time warnings and errors.

Remarks

Optional definition

DRV_SPI_ELEMENTS_PER_QUEUE Macro

Controls the number of elements that are allocated.

File

drv_spi_config_template.h

C

```
#define DRV_SPI_ELEMENTS_PER_QUEUE 10
```

Description

SPI Buffer Queue Depth

This definition along with DRV_SPI_INSTANCES_NUMBER and DRV_SPI_CLIENT_NUMBER controls how many buffer queue elements are created.

Remarks

Optional definition

DRV_SPI_ISR Macro

Controls the compilation of ISR mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_ISR 1
```

Description

SPI ISR Mode Enable

This definition controls whether or not ISR mode functionality is built as part of the driver. With it set to 1 then ISR mode will be compiled and taskMode = DRV_SPI_TASK_MODE_ISR will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert

Remarks

Optional definition

DRV_SPI_MASTER Macro

Controls the compilation of master mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_MASTER 1
```

Description

SPI Master Mode Enable

This definition controls whether or not master mode functionality is built as part of the driver. With it set to 1 then master mode will be compiled and spiMode = DRV_SPI_MODE_MASTER will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert

Remarks

Optional definition

DRV_SPI_POLLED Macro

Controls the compilation of Polled mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_POLLED 1
```

Description

SPI Polled Mode Enable

This definition controls whether or not polled mode functionality is built as part of the driver. With it set to 1 then polled mode will be compiled and taskMode = DRV_SPI_TASK_MODE_POLLED will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert

Remarks

Optional definition

DRV_SPI_RM Macro

Controls the compilation of Standard Buffer mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_RM 1
```

Description

SPI Standard Buffer Mode Enable

This definition controls whether or not Standard Buffer mode functionality is built as part of the driver. With it set to 1 then standard buffer mode will be compiled and bufferType = DRV_SPI_BUFFER_TYPE_STANDARD will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert. This mode is available on all PIC32s

Remarks

Optional definition

DRV_SPI_SLAVE Macro

Controls the compilation of slave mode

File

drv_spi_config_template.h

C

```
#define DRV_SPI_SLAVE 1
```

Description

SPI Slave Mode Enable

This definition controls whether or not slave mode functionality is built as part of the driver. With it set to 1 then slave mode will be compiled and spiMode = DRV_SPI_MODE_SLAVE will be accepted by SPI_DRV_Initialize(). With it set to 0 SPI_DRV_Initialize() will cause an assert

Remarks

Optional definition

Miscellaneous Configuration

DRV_SPI_INSTANCES_NUMBER Macro

Selects the maximum number of hardware instances that can be supported by the dynamic driver .

File

drv_spi_config_template.h

C

```
#define DRV_SPI_INSTANCES_NUMBER 1
```

Description

SPI hardware instance configuration

This definition selects the maximum number of hardware instances that can be supported by the dynamic driver.

Remarks

Mandatory definition

DRV_SPI_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_spi_config_template.h

C

```
#define DRV_SPI_CLIENTS_NUMBER 1
```

Description

SPI maximum number of clients

This definition selects the maximum number of clients that the SPI driver can support at run time.

Remarks

Mandatory definition

Building the Library

This section lists the files that are available in the SPI Driver Library.

Description

This section list the files that are available in the \src folder of the SPI Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (.h) and source (.c) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/spi`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_spi.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_spi.c`	Basic SPI Driver implementation file.
`/src/dynamic/drv_spi_api.c`	Functions used by the driver API.
`/src/drv_spi_sys_queue_fifo.c`	Queue implementation used by the SPI Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The SPI Driver Library depends on the following modules:

- Clock System Service Library

Optional Dependencies

- DMA System Service Library (used when operating in DMA mode)
- Interrupt System Service Library (used when task is running in Interrupt mode)

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_SPI_Initialize	Initializes the SPI instance for the specified driver index. **Implementation:** Static/Dynamic

	DRV_SPI_Deinitialize	Deinitializes the specified instance of the SPI driver module. **Implementation:** Static/Dynamic
	DRV_SPI_Status	Provides the current status of the SPI driver module. **Implementation:** Static/Dynamic
	DRV_SPI_Tasks	Maintains the driver's state machine and implements its ISR. **Implementation:** Static/Dynamic

b) Client Setup Functions

	Name	Description
	DRV_SPI_Close	Closes an opened instance of the SPI driver. **Implementation:** Static/Dynamic
	DRV_SPI_Open	Opens the specified SPI driver instance and returns a handle to it. **Implementation:** Static/Dynamic
	DRV_SPI_ClientConfigure	Configures a SPI client with specific data. **Implementation:** Static/Dynamic

c) Data Transfer Functions

	Name	Description
	DRV_SPI_BufferStatus	Returns the transmitter and receiver transfer status. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddRead	Registers a buffer for a read operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWrite	Registers a buffer for a write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWriteRead	Registers a buffer for a read and write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddRead2	Registers a buffer for a read operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWrite2	Registers a buffer for a write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWriteRead2	Registers a buffer for a read and write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPIn_ReceiverBufferIsFull	Returns the receive buffer status. 'n' represents the instance of the SPI driver used. **Implementation:** Static
	DRV_SPIn_TransmitterBufferIsFull	Returns the transmit buffer status. 'n' represents the instance of the SPI driver used. **Implementation:** Static

e) Data Types and Constants

	Name	Description
	DRV_SPI_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_SPI_INDEX_0	SPI driver index definitions.
	DRV_SPI_INDEX_COUNT	Number of valid SPI driver indices.
	DRV_SPI_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_SPI_BUFFER_EVENT_HANDLER	Pointer to a SPI Driver Buffer Event handler function
	DRV_SPI_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.
	DRV_SPI_BUFFER_TYPE	Identifies the various buffer types of the SPI module.
	DRV_SPI_CLOCK_MODE	Identifies the various clock modes of the SPI module.
	DRV_SPI_INIT	Defines the data required to initialize or reinitialize the SPI driver
	_DRV_SPI_INIT	Defines the data required to initialize or reinitialize the SPI driver
	DRV_SPI_MODE	Identifies the various usage modes of the SPI module.
	DRV_SPI_PROTOCOL_TYPE	Identifies the various protocols of the SPI module.
	DRV_SPI_INDEX_1	This is macro DRV_SPI_INDEX_1.
	DRV_SPI_INDEX_2	This is macro DRV_SPI_INDEX_2.
	DRV_SPI_INDEX_3	This is macro DRV_SPI_INDEX_3.
	DRV_SPI_INDEX_4	This is macro DRV_SPI_INDEX_4.

	DRV_SPI_INDEX_5	This is macro DRV_SPI_INDEX_5.
	DRV_SPI_TASK_MODE	Identifies the various modes of how the tasks function will be run.
	DRV_SPI_CLIENT_DATA	Defines the data that can be changed per client.
	_DRV_SPI_CLIENT_DATA	Defines the data that can be changed per client.

Description

This section describes the API functions of the SPI Driver library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_SPI_Initialize Function

Initializes the SPI instance for the specified driver index.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
SYS_MODULE_OBJ DRV_SPI_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

- If successful - returns a valid handle to a driver instance object
- If unsuccessful - returns SYS_MODULE_OBJ_INVALID

Description

This routine initializes the SPI driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the 'init' parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the SPI module ID. For example, driver instance 0 can be assigned to SPI2. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_SPI_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other SPI routine is called.

This routine should only be called once during system initialization unless DRV_SPI_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

None.

Example

```
DRV_SPI_INIT        init;
SYS_MODULE_OBJ      objectHandle;

// Populate the SPI initialization structure
init.spiId = SPI_ID_1,
init.taskMode = DRV_SPI_TASK_MODE_ISR,
init.spiMode = DRV_SPI_MODE_MASTER,
init.allowIdleRun = false,
init.spiProtocolType = DRV_SPI_PROTOCOL_TYPE_STANDARD,
init.commWidth = SPI_COMMUNICATION_WIDTH_8BITS,
init.spiClk = CLK_BUS_PERIPHERAL_2,
init.baudRate = 10000000,
init.bufferType = DRV_SPI_BUFFER_TYPE_ENHANCED,
init.clockMode = DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL,
init.inputSamplePhase = SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE,
init.txInterruptSource = INT_SOURCE_SPI_1_TRANSMIT,
init.rxInterruptSource = INT_SOURCE_SPI_1_RECEIVE,
init.errInterruptSource = INT_SOURCE_SPI_1_ERROR,
init.queueSize = 10,
init.jobQueueReserveSize = 1,
```

```
objectHandle = DRV_SPI_Initialize(DRV_SPI_INDEX_1, (SYS_MODULE_INIT*)usartInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized. Please note this is not the SPI id. The hardware SPI id is set in the initialization structure. This is the index of the driver index to use.
init	Pointer to a data structure containing any data necessary to initialize the driver. If this pointer is NULL, the driver uses the static initialization override macros for each member of the initialization data structure.

Function

SYS_MODULE_OBJ DRV_SPI_Initialize(const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init)

DRV_SPI_Deinitialize Function

Deinitializes the specified instance of the SPI driver module.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
void DRV_SPI_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SPI driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_SPI_Status operation. The system has to use DRV_SPI_Status to find out when the module is in the ready state.

Preconditions

Function DRV_SPI_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_SPI_Initialize
SYS_STATUS          status;

DRV_SPI_Deinitialize ( object );

status = DRV_SPI_Status( object );
if( SYS_MODULE_UNINITIALIZED == status )
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_SPI_Initialize

Function

void DRV_SPI_Deinitialize (SYS_MODULE_OBJ object)

DRV_SPI_Status Function

Provides the current status of the SPI driver module.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
SYS_STATUS DRV_SPI_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that the driver is busy with a previous

system level operation and cannot start another

Description

This function provides the current status of the SPI driver module.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_MODULE_UNINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

This function can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The DRV_SPI_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SPI_Initialize
SYS_STATUS          status;

status = DRV_SPI_Status( object );
if( SYS_STATUS_READY != status )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_SPI_Initialize

Function

SYS_STATUS DRV_SPI_Status (SYS_MODULE_OBJ object)

DRV_SPI_Tasks Function

Maintains the driver's state machine and implements its ISR.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
void DRV_SPI_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal state machine and implement its transmit ISR for interrupt-driven implementations. In polling mode, this function should be called from the SYS_Tasks() function. In interrupt mode, this function should be called in the transmit interrupt service routine of the USART that is associated with this USART driver hardware instance.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This function may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SPI_Initialize

while( true )
{
    DRV_SPI_Tasks ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_SPI_Initialize)

Function

void DRV_SPI_Tasks (SYS_MODULE_OBJ object);

b) Client Setup Functions

DRV_SPI_Close Function

Closes an opened instance of the SPI driver.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
void DRV_SPI_Close(DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened instance of the SPI driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_SPI_Open before the caller may use the driver again. This function is thread safe in a RTOS application.

Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_SPI_Open

DRV_SPI_Close ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_SPI_Close (DRV_HANDLE handle)

DRV_SPI_Open Function

Opens the specified SPI driver instance and returns a handle to it.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_HANDLE DRV_SPI_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. An error can occur when the following is true:

- if the number of client objects allocated via DRV_SPI_INSTANCES_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid

Description

This routine opens the specified SPI driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

If ioIntent is DRV_IO_INTENT_READ, the client will only be read from the driver. If ioIntent is DRV_IO_INTENT_WRITE, the client will only be able to write to the driver. If the ioIntent in DRV_IO_INTENT_READWRITE, the client will be able to do both, read and write.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_SPI_Close routine is called. This routine will NEVER block waiting for hardware. If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

The DRV_SPI_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_SPI_Open( DRV_SPI_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

if( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Index of the driver initialized with DRV_SPI_Initialize(). Please note this is not the SPI ID.
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_SPI_Open (const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT ioIntent)

DRV_SPI_ClientConfigure Function

Configures a SPI client with specific data.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
int32_t DRV_SPI_ClientConfigure(DRV_HANDLE handle, const DRV_SPI_CLIENT_DATA * cfgData);
```

Returns

- If successful - the routing will return greater than or equal to zero
- If an error occurs - the return value is negative

Description

This routine takes a DRV_SPI_CLIENT_DATA structure and sets client specific options. Whenever a new SPI job is started these values will be used. Passing in NULL will reset the client back to configuration parameters passed to driver initialization. A zero in any of the structure elements will reset that specific configuration back to the driver default.

Preconditions

The DRV_SPI_Open function must have been called before calling this function.

Parameters

Parameters	Description
handle	handle of the client returned by DRV_SPI_Open.
cfgData	Client-specific configuration data.

Function

int32_t DRV_SPI_ClientConfigure (DRV_HANDLE handle,
const DRV_SPI_CLIENT_DATA * cfgData)

c) Data Transfer Functions

DRV_SPI_BufferStatus Function

Returns the transmitter and receiver transfer status.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_EVENT DRV_SPI_BufferStatus(DRV_SPI_BUFFER_HANDLE bufferHandle);
```

Returns

A DRV_SPI_BUFFER_STATUS value describing the current status of the transfer.

Description

This returns the transmitter and receiver transfer status.

Remarks

The returned status may contain a value with more than one of the bits specified in the DRV_SPI_BUFFER_STATUS enumeration set. The caller should perform an AND with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

DRV_SPI_BufferAddmust have been called to obtain the buffer handle associated with that transfer.

Example

```
// Buffer handle returned from the data transfer function
DRV_SPI_BUFFER_HANDLE bufferHandle;

if( DRV_SPI_BUFFER_STATUS_SUCCESS & DRV_SPI_BufferStatus( bufferHandle ) )
{
    // All transmitter data has been sent.
}
```

Parameters

Parameters	Description
bufferHandle	A valid buffer handle, returned from the driver's data transfer routine

Function

DRV_SPI_BUFFER_EVENT DRV_SPI_BufferStatus (DRV_SPI_BUFFER_HANDLE bufferHandle)

DRV_SPI_BufferAddRead Function

Registers a buffer for a read operation. Actual transfer will happen in the Task function.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddRead(DRV_HANDLE handle, void * rxBuffer, size_t size,
DRV_SPI_BUFFER_EVENT_HANDLER completeCB, void * context);
```

Returns

If the buffer add request is successful, a valid buffer handle is returned. If request is not queued up, DRV_SPI_BUFFER_HANDLE_INVALID is returned.

Description

Registers a buffer for a read operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using DRV_SPI_BufferStatus function. A optional callback can also be provided that will be called when the operation is complete.

Remarks

This API will be deprecated soon, so avoid using it. Use "DRV_SPI_BufferAddRead2" instead of it.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SPI_Open call.

Example

```
DRV_HANDLE      handle;    // Returned from DRV_SPI_Open
char   myBuffer[MY_BUFFER_SIZE], state = 0;
DRV_SPI_BUFFER_HANDLE bufferHandle;

switch ( state )
{
    case 0:
        bufferHandle = DRV_SPI_BufferAddRead( handle, myBuffer, 10, NULL, NULL );
        if(bufferHandle != DRV_SPI_BUFFER_HANDLE_INVALID )
        {
            state++;
        }
```

```
            break;
        case 1:
            if( DRV_SPI_BUFFER_EVENT_COMPLETE & DRV_SPI_BufferStatus( bufferHandle ) )
            {
                state++;
                // All transmitter data has been sent successfully.
            }
            break;
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
rxBuffer	The buffer to which the data should be written to.
size	Number of bytes to be read from the SPI bus.
completeCB	Pointer to a function to be called when this queued operation is complete.
context	unused by the driver but this is passed to the callback when it is called.

Function

DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddRead (DRV_HANDLE handle, void *rxBuffer,
size_t size, DRV_SPI_BUFFER_EVENT_HANDLER completeCB,
void * context)

DRV_SPI_BufferAddWrite Function

Registers a buffer for a write operation. Actual transfer will happen in the Task function.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWrite(DRV_HANDLE handle, void * txBuffer, size_t size,
DRV_SPI_BUFFER_EVENT_HANDLER completeCB, void * context);
```

Returns

If the buffer add request is successful, a valid buffer handle is returned. If request is not queued up, DRV_SPI_BUFFER_HANDLE_INVALID is returned.

Description

Registers a buffer for a write operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using DRV_SPI_BufferStatus function. A optional callback can also be provided that will be called when the operation is complete.

Remarks

This API will be deprecated soon, so avoid using it. Use "DRV_SPI_BufferAddWrite2" instead of it.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SPI_Open call.

Example

```
DRV_HANDLE      handle;    // Returned from DRV_SPI_Open
char   myBuffer[MY_BUFFER_SIZE], state = 0;
DRV_SPI_BUFFER_HANDLE bufferHandle;

switch ( state )
{
    case 0:
        bufferHandle = DRV_SPI_BufferAddWrite( handle, myBuffer, 10, NULL, NULL );
        if(bufferHandle != DRV_SPI_BUFFER_HANDLE_INVALID )
        {
            state++;
```

```
            }
            break;
        case 1:
            if( DRV_SPI_BUFFER_EVENT_COMPLETE & DRV_SPI_BufferStatus( bufferHandle ) )
            {
                state++;
                // All transmitter data has been sent successfully.
            }
            break;
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
txBuffer	The buffer which hold the data.
size	Number of bytes to be written to the SPI bus.
completeCB	Pointer to a function to be called when this queued operation is complete
context	unused by the driver but this is passed to the callback when it is called

Function

DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWrite (DRV_HANDLE handle, void *txBuffer,
size_t size, DRV_SPI_BUFFER_EVENT_HANDLER completeCB,
void * context)

DRV_SPI_BufferAddWriteRead Function

Registers a buffer for a read and write operation. Actual transfer will happen in the Task function.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWriteRead(DRV_HANDLE handle, void * txBuffer, size_t txSize, void *
rxBuffer, size_t rxSize, DRV_SPI_BUFFER_EVENT_HANDLER completeCB, void * context);
```

Returns

If the buffer add request is successful, a valid buffer handle is returned. If request is not queued up, DRV_SPI_BUFFER_HANDLE_INVALID is returned.

Description

Registers a buffer for a read and write operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using DRV_SPI_BufferStatus function. A optional callback can also be provided that will be called when the operation is complete.

Remarks

This API will be deprecated soon, so avoid using it. Use "DRV_SPI_BufferAddWriteRead2" instead of it.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      handle;    // Returned from DRV_SPI_Open
char   myReadBuffer[MY_BUFFER_SIZE], myWriteBuffer[MY_BUFFER_SIZE], state = 0;
DRV_SPI_BUFFER_HANDLE bufferHandle;

switch ( state )
{
    case 0:
        bufferHandle = DRV_SPI_BufferAddWriteRead( handle, myWriteBuffer, 10, myReadBuffer, 10, NULL, NULL
);
        if(bufferHandle != DRV_SPI_BUFFER_HANDLE_INVALID )
        {
```

```
            state++;
        }
        break;
    case 1:
        if( DRV_SPI_BUFFER_EVENT_COMPLETE & DRV_SPI_BufferStatus( bufferHandle ) )
        {
            state++;
            // All transmitter data has been sent successfully.
        }
        break;
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
txBuffer	The buffer which hold the data.
txSize	Number of bytes to be written to the SPI bus.
rxBuffer	The buffer to which the data should be written to.
rxSize	Number of bytes to be read from the SPI bus
completeCB	Pointer to a function to be called when this queued operation is complete
context	unused by the driver but this is passed to the callback when it is called

Function

DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWriteRead(DRV_HANDLE handle,

void *txBuffer, void *rxBuffer, size_t size,)

DRV_SPI_BufferAddRead2 Function

Registers a buffer for a read operation. Actual transfer will happen in the Task function.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddRead2(DRV_HANDLE handle, void * rxBuffer, size_t size,
DRV_SPI_BUFFER_EVENT_HANDLER completeCB, void * context, DRV_SPI_BUFFER_HANDLE * jobHandle);
```

Returns

If the buffer add request is successful, a valid buffer handle is returned. If request is not queued up, DRV_SPI_BUFFER_HANDLE_INVALID is returned.

Description

Registers a buffer for a read operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using DRV_SPI_BufferStatus function. A optional callback can also be provided that will be called when the operation is complete.

Remarks

None.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SPI_Open call.

Example

```
DRV_HANDLE      handle;    // Returned from DRV_SPI_Open
char   myBuffer[MY_BUFFER_SIZE], state = 0;
DRV_SPI_BUFFER_HANDLE bufferHandle, bufferHandle2;

switch ( state )
{
    case 0:
        bufferHandle = DRV_SPI_BufferAddRead2( handle, myBuffer, 10, NULL, NULL, bufferHandle2 );
```

```
            if(bufferHandle2 != DRV_SPI_BUFFER_HANDLE_INVALID )
            {
                state++;
            }
            break;
        case 1:
            if( DRV_SPI_BUFFER_EVENT_COMPLETE & DRV_SPI_BufferStatus( bufferHandle2 ) )
            {
                state++;
                // All transmitter data has been sent successfully.
            }
            break;
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
rxBuffer	The buffer to which the data should be written to.
size	Number of bytes to be read from the SPI bus.
completeCB	Pointer to a function to be called when this queued operation is complete
context	unused by the driver but this is passed to the callback when it is called
jobHandle	pointer to the buffer handle, this will be set before the function returns and can be used in the ISR callback.

Function

DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddRead2 (DRV_HANDLE handle, void *rxBuffer,
size_t size, DRV_SPI_BUFFER_EVENT_HANDLER completeCB,
void * context, DRV_SPI_BUFFER_HANDLE * jobHandle)

DRV_SPI_BufferAddWrite2 Function

Registers a buffer for a write operation. Actual transfer will happen in the Task function.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWrite2(DRV_HANDLE handle, void * txBuffer, size_t size,
DRV_SPI_BUFFER_EVENT_HANDLER completeCB, void * context, DRV_SPI_BUFFER_HANDLE * jobHandle);
```

Returns

If the buffer add request is successful, a valid buffer handle is returned. If request is not queued up, DRV_SPI_BUFFER_HANDLE_INVALID is returned.

Description

Registers a buffer for a write operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using DRV_SPI_BufferStatus function. A optional callback can also be provided that will be called when the operation is complete.

Remarks

None.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SPI_Open call.

Example

```
DRV_HANDLE      handle;    // Returned from DRV_SPI_Open
char   myBuffer[MY_BUFFER_SIZE], state = 0;
DRV_SPI_BUFFER_HANDLE bufferHandle, bufferHandle2;

switch ( state )
```

```
{
    case 0:
        bufferHandle = DRV_SPI_BufferAddWrite2( handle, myBuffer, 10, NULL, NULL, bufferHandle2 );
        if(bufferHandle2 != DRV_SPI_BUFFER_HANDLE_INVALID )
        {
            state++;
        }
        break;
    case 1:
        if( DRV_SPI_BUFFER_EVENT_COMPLETE & DRV_SPI_BufferStatus( bufferHandle2 ) )
        {
            state++;
            // All transmitter data has been sent successfully.
        }
        break;
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
txBuffer	The buffer which hold the data.
size	Number of bytes to be written to the SPI bus.
completeCB	Pointer to a function to be called when this queued operation is complete
context	unused by the driver but this is passed to the callback when it is called
jobHandle	pointer to the buffer handle, this will be set before the function returns and can be used in the ISR callback.

Function

DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWrite2 (DRV_HANDLE handle, void *txBuffer,
size_t size, DRV_SPI_BUFFER_EVENT_HANDLER completeCB,
void * context, DRV_SPI_BUFFER_HANDLE * jobHandle)

DRV_SPI_BufferAddWriteRead2 Function

Registers a buffer for a read and write operation. Actual transfer will happen in the Task function.

Implementation: Static/Dynamic

File

drv_spi.h

C

```
DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWriteRead2(DRV_HANDLE handle, void * txBuffer, size_t txSize, void *
rxBuffer, size_t rxSize, DRV_SPI_BUFFER_EVENT_HANDLER completeCB, void * context, DRV_SPI_BUFFER_HANDLE *
jobHandle);
```

Returns

If the buffer add request is successful, a valid buffer handle is returned. If request is not queued up, DRV_SPI_BUFFER_HANDLE_INVALID is returned.

Description

Registers a buffer for a read and write operation. Actual transfer will happen in the Task function. The status of this operation can be monitored using DRV_SPI_BufferStatus function. A optional callback can also be provided that will be called when the operation is complete.

Remarks

None.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

DRV_SPI_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      handle;    // Returned from DRV_SPI_Open
char   myReadBuffer[MY_BUFFER_SIZE], myWriteBuffer[MY_BUFFER_SIZE], state = 0;
```

```
DRV_SPI_BUFFER_HANDLE bufferHandle, bufferHandle2;

switch ( state )
{
    case 0:
        bufferHandle = DRV_SPI_BufferAddWriteRead2( handle, myWriteBuffer, 10, myReadBuffer, 10, NULL,
NULL, bufferHandle2 );
        if(bufferHandle2 != DRV_SPI_BUFFER_HANDLE_INVALID )
        {
            state++;
        }
        break;
    case 1:
        if( DRV_SPI_BUFFER_EVENT_COMPLETE & DRV_SPI_BufferStatus( bufferHandle2 ) )
        {
            state++;
            // All transmitter data has been sent successfully.
        }
        break;
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
txBuffer	The buffer which hold the data.
txSize	Number of bytes to be written to the SPI bus.
rxBuffer	The buffer to which the data should be written to.
rxSize	Number of bytes to be read from the SPI bus
completeCB	Pointer to a function to be called when this queued operation is complete
context	unused by the driver but this is passed to the callback when it is called
jobHandle	pointer to the buffer handle, this will be set before the function returns and can be used in the ISR callback.

Function

DRV_SPI_BUFFER_HANDLE DRV_SPI_BufferAddWriteRead2(DRV_HANDLE handle,
void *txBuffer, void *rxBuffer, size_t size,
DRV_SPI_BUFFER_EVENT_HANDLER completeCB,
void * context, DRV_SPI_BUFFER_HANDLE * jobHandle)

DRV_SPIn_ReceiverBufferIsFull Function

Returns the receive buffer status. 'n' represents the instance of the SPI driver used.

Implementation: Static

File

drv_spi.h

C

```
bool DRV_SPIn_ReceiverBufferIsFull();
```

Returns

Receive Buffer Status

- 1 - Full
- 0 - Empty

Description

This function returns the receive buffer status (full/empty).

Remarks

None.

Preconditions

None.

Example

```
bool rxBufStat;
// Using instance 1 of SPI driver, that is n = 1
rxBufStat = DRV_SPI1_ReceiverBufferIsFull();

if (rxBufStat)
{
...
}
```

Function

bool DRV_SPIn_ReceiverBufferIsFull(void)

DRV_SPIn_TransmitterBufferIsFull Function

Returns the transmit buffer status. 'n' represents the instance of the SPI driver used.

Implementation: Static

File

drv_spi.h

C

```
bool DRV_SPIn_TransmitterBufferIsFull();
```

Returns

Transmit Buffer Status

- 1 - Full
- 0 - Empty

Description

This function returns the transmit buffer status (full/empty).

Remarks

None.

Preconditions

None.

Example

```
bool txBufStat;
// Using instance 1 of SPI driver, that is n = 1
txBufStat = DRV_SPI1_TransmitterBufferIsFull();

if (txBufStat)
{
...
}
```

Function

bool DRV_SPIn_TransmitterBufferIsFull(void)

d) Miscellaneous Functions

e) Data Types and Constants

DRV_SPI_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_spi.h

C

```
#define DRV_SPI_BUFFER_HANDLE_INVALID ((DRV_SPI_BUFFER_HANDLE)(-1))
```

Description

SPI Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_SPI_BufferAddRead() and DRV_SPI_BufferAddWrite() function if the buffer add request was not successful.

Remarks

None.

DRV_SPI_INDEX_0 Macro

SPI driver index definitions.

File

drv_spi.h

C

```
#define DRV_SPI_INDEX_0 0
```

Description

SPI Driver Module Index Numbers

These constants provide the SPI driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_SPI_Initialize and DRV_SPI_Open functions to identify the driver instance in use.

DRV_SPI_INDEX_COUNT Macro

Number of valid SPI driver indices.

File

drv_spi.h

C

```
#define DRV_SPI_INDEX_COUNT SPI_NUMBER_OF_MODULES
```

Description

SPI Driver Module Index Count

This constant identifies the number of valid SPI driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific header files defined as part of the peripheral libraries.

DRV_SPI_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_spi.h

C

```
typedef enum {
  DRV_SPI_BUFFER_EVENT_PENDING,
  DRV_SPI_BUFFER_EVENT_PROCESSING,
  DRV_SPI_BUFFER_EVENT_COMPLETE,
  DRV_SPI_BUFFER_EVENT_ERROR
} DRV_SPI_BUFFER_EVENT;
```

Members

Members	Description
DRV_SPI_BUFFER_EVENT_PENDING	Buffer is pending to get processed
DRV_SPI_BUFFER_EVENT_PROCESSING	Buffer is being processed
DRV_SPI_BUFFER_EVENT_COMPLETE	All data from or to the buffer was transferred successfully.
DRV_SPI_BUFFER_EVENT_ERROR	There was an error while processing the buffer transfer request.

Description

SPI Driver Buffer Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either DRV_SPI_BufferAddRead2 or DRV_SPI_BufferAddWrite2 or similar functions.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered during buffer add requests.

DRV_SPI_BUFFER_EVENT_HANDLER Type

Pointer to a SPI Driver Buffer Event handler function

File

drv_spi.h

C

```
typedef void (* DRV_SPI_BUFFER_EVENT_HANDLER)(DRV_SPI_BUFFER_EVENT event, DRV_SPI_BUFFER_HANDLE
bufferHandle, void * context);
```

Returns

None.

Description

SPI Driver Buffer Event Handler Function Pointer

This data type defines the required function signature for the SPI driver buffer event handling callback function. A client must register a pointer to a buffer event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values and return value are described here and a partial example implementation is provided.

Remarks

If the event is DRV_SPI_BUFFER_EVENT_COMPLETE, it means that the data was transferred successfully.

If the event is DRV_SPI_BUFFER_EVENT_ERROR, it means that the data was not transferred successfully.

The bufferHandle parameter contains the buffer handle of the buffer that failed.

The context parameter contains the a handle to the client context, provided at the time the event handling function registration. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The event handler function executes in an interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive operations with in this function.

Example

```
void APP_MyBufferEventHandler( DRV_SPI_BUFFER_EVENT event,
                               DRV_SPI_BUFFER_HANDLE bufferHandle,
                               uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
```

```
    {
        case DRV_SPI_BUFFER_EVENT_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_SPI_BUFFER_EVENT_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the vent relates
context	Value identifying the context of the application that registered the event handling function.

DRV_SPI_BUFFER_HANDLE Type

Handle identifying a read or write buffer passed to the driver.

File

drv_spi.h

C

```
typedef uintptr_t DRV_SPI_BUFFER_HANDLE;
```

Description

SPI Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_SPI_BufferAddRead()/ DRV_SPI_BufferAddWrite or DRV_SPI_BufferAddReadWrite() functions. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_SPI_BUFFER_TYPE Enumeration

Identifies the various buffer types of the SPI module.

File

drv_spi.h

C

```
typedef enum {
  DRV_SPI_BUFFER_TYPE_STANDARD,
  DRV_SPI_BUFFER_TYPE_ENHANCED
} DRV_SPI_BUFFER_TYPE;
```

Members

Members	Description
DRV_SPI_BUFFER_TYPE_STANDARD	SPI Buffer Type Standard
DRV_SPI_BUFFER_TYPE_ENHANCED	SPI Enhanced Buffer Type

Description

SPI Buffer Type Selection

This enumeration identifies the various buffer types of the SPI module.

Remarks

None.

DRV_SPI_CLOCK_MODE Enumeration

Identifies the various clock modes of the SPI module.

File

drv_spi.h

C

```
typedef enum {
  DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_RISE,
  DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL,
  DRV_SPI_CLOCK_MODE_IDLE_HIGH_EDGE_FALL,
  DRV_SPI_CLOCK_MODE_IDLE_HIGH_EDGE_RISE
} DRV_SPI_CLOCK_MODE;
```

Members

Members	Description
DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_RISE	SPI Clock Mode 0 - Idle State Low & SDO Sampling on Rising Edge
DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL	SPI Clock Mode 1 - Idle State Low & SDO Sampling on Falling Edge
DRV_SPI_CLOCK_MODE_IDLE_HIGH_EDGE_FALL	SPI Clock Mode 2 - Idle State High & SDO Sampling on Falling Edge
DRV_SPI_CLOCK_MODE_IDLE_HIGH_EDGE_RISE	SPI Clock Mode 3 - Idle State High & SDO Sampling on Rising Edge

Description

SPI Clock Mode Selection

This enumeration identifies the various clock modes of the SPI module.

Remarks

None.

DRV_SPI_INIT Structure

Defines the data required to initialize or reinitialize the SPI driver

File

drv_spi.h

C

```
typedef struct _DRV_SPI_INIT {
  SYS_MODULE_INIT moduleInit;
  SPI_MODULE_ID spiId;
  DRV_SPI_TASK_MODE taskMode;
  DRV_SPI_MODE spiMode;
  bool allowIdleRun;
  DRV_SPI_PROTOCOL_TYPE spiProtocolType;
  bool spiSlaveSSPin;
  SPI_FRAME_SYNC_PULSE frameSyncPulse;
  SPI_FRAME_PULSE_POLARITY framePulsePolarity;
  SPI_FRAME_PULSE_DIRECTION framePulseDirection;
  SPI_FRAME_PULSE_EDGE framePulseEdge;
  SPI_FRAME_PULSE_WIDTH framePulseWidth;
  SPI_AUDIO_TRANSMIT_MODE audioTransmitMode;
  SPI_AUDIO_PROTOCOL audioProtocolMode;
  SPI_COMMUNICATION_WIDTH commWidth;
  CLK_BUSES_PERIPHERAL spiClk;
  uint32_t baudRate;
  DRV_SPI_BUFFER_TYPE bufferType;
  DRV_SPI_CLOCK_MODE clockMode;
  SPI_INPUT_SAMPLING_PHASE inputSamplePhase;
  INT_SOURCE txInterruptSource;
  INT_SOURCE rxInterruptSource;
  INT_SOURCE errInterruptSource;
```

```
    uint8_t numTrfsSmPolled;
    DMA_CHANNEL txDmaChannel;
    uint8_t txDmaThreshold;
    DMA_CHANNEL rxDmaChannel;
    uint8_t rxDmaThreshold;
    uint8_t queueSize;
    uint8_t jobQueueReserveSize;
    DRV_SPI_BUFFER_EVENT_HANDLER operationStarting;
    DRV_SPI_BUFFER_EVENT_HANDLER operationEnded;
  } DRV_SPI_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SPI_MODULE_ID spiId;	Identifies peripheral (PLIB-level) ID
DRV_SPI_TASK_MODE taskMode;	SPI Task Mode Type
DRV_SPI_MODE spiMode;	SPI Usage Mode Type
bool allowIdleRun;	Allow SPI to run when CPU goes to idle mode
DRV_SPI_PROTOCOL_TYPE spiProtocolType;	SPI Protocol Type
bool spiSlaveSSPin;	SPI Slave Mode SSx Pin Select
SPI_FRAME_SYNC_PULSE frameSyncPulse;	Framed mode Sync Pulse
SPI_FRAME_PULSE_POLARITY framePulsePolarity;	Framed Mode pulse polarity
SPI_FRAME_PULSE_DIRECTION framePulseDirection;	Framed Mode Pulse Direction
SPI_FRAME_PULSE_EDGE framePulseEdge;	Framed mode Pulse Edge
SPI_FRAME_PULSE_WIDTH framePulseWidth;	Framed Mode Pulse width
SPI_AUDIO_TRANSMIT_MODE audioTransmitMode;	Audio mode transmit mode
SPI_AUDIO_PROTOCOL audioProtocolMode;	Audio mode protocol mode
SPI_COMMUNICATION_WIDTH commWidth;	Communication Width
CLK_BUSES_PERIPHERAL spiClk;	Peripheral clock used by the SPI driver
uint32_t baudRate;	Baud Rate Value
DRV_SPI_BUFFER_TYPE bufferType;	SPI Buffer Type
DRV_SPI_CLOCK_MODE clockMode;	SPI Clock mode
SPI_INPUT_SAMPLING_PHASE inputSamplePhase;	SPI Input Sample Phase Selection
INT_SOURCE txInterruptSource;	Transmit/Receive or Transmit Interrupt Source for SPI module
INT_SOURCE rxInterruptSource;	Receive Interrupt Source for SPI module
INT_SOURCE errInterruptSource;	Error Interrupt Source for SPI module
uint8_t numTrfsSmPolled;	While using standard buffer and polled mode how many transfers to do before yielding to other tasks
DMA_CHANNEL txDmaChannel;	DMA Channel for the Transmitter
uint8_t txDmaThreshold;	Threshold for the minimum number of bytes to send to use DMA
DMA_CHANNEL rxDmaChannel;	DMA Channel for the Receiver
uint8_t rxDmaThreshold;	Threshold for the minimum number of bytes to receive to use DMA
uint8_t queueSize;	This is the buffer queue size. This is the maximum number of transfer requests that driver will queue.
uint8_t jobQueueReserveSize;	This controls the minimum number of jobs that the driver will be able to accept without running out of memory. The driver will reserve this number of jobs from the global SPI queue so that it will always be available
DRV_SPI_BUFFER_EVENT_HANDLER operationStarting;	This callback is fired when an operation is about to start on the SPI bus. This allows the user to set any pins that need to be set. This callback may be called from an ISR so should not include OSAL calls. The context parameter is the same one passed into the BufferAddRead, BufferAddWrite, BufferAddWriteRead function.
DRV_SPI_BUFFER_EVENT_HANDLER operationEnded;	This callback is fired when an operation has just completed on the SPI bus. This allows the user to set any pins that need to be set. This callback may be called from an ISR so should not include OSAL calls. The context parameter is the same one passed into the BufferAddRead, BufferAddWrite, BufferAddWriteRead function.

Description

SPI Driver Initialization Data

This data type defines the data required to initialize or reinitialize the SPI driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

DRV_SPI_MODE Enumeration

Identifies the various usage modes of the SPI module.

File

drv_spi.h

C

```
typedef enum {
  DRV_SPI_MODE_MASTER,
  DRV_SPI_MODE_SLAVE
} DRV_SPI_MODE;
```

Members

Members	Description
DRV_SPI_MODE_MASTER	SPI Mode Master
DRV_SPI_MODE_SLAVE	SPI Mode Slave

Description

SPI Usage Modes Enumeration

This enumeration identifies the various usage modes of the SPI module.

Remarks

None.

DRV_SPI_PROTOCOL_TYPE Enumeration

Identifies the various protocols of the SPI module.

File

drv_spi.h

C

```
typedef enum {
  DRV_SPI_PROTOCOL_TYPE_STANDARD,
  DRV_SPI_PROTOCOL_TYPE_FRAMED,
  DRV_SPI_PROTOCOL_TYPE_AUDIO
} DRV_SPI_PROTOCOL_TYPE;
```

Members

Members	Description
DRV_SPI_PROTOCOL_TYPE_STANDARD	SPI Protocol Type Standard
DRV_SPI_PROTOCOL_TYPE_FRAMED	SPI Protocol Type Framed
DRV_SPI_PROTOCOL_TYPE_AUDIO	SPI Protocol Type Audio

Description

SPI Protocols Enumeration

This enumeration identifies the various protocols of the SPI module.

Remarks

None.

DRV_SPI_INDEX_1 Macro

File

drv_spi.h

C

```
#define DRV_SPI_INDEX_1 1
```

Description

This is macro DRV_SPI_INDEX_1.

DRV_SPI_INDEX_2 Macro

File

drv_spi.h

C

```
#define DRV_SPI_INDEX_2 2
```

Description

This is macro DRV_SPI_INDEX_2.

DRV_SPI_INDEX_3 Macro

File

drv_spi.h

C

```
#define DRV_SPI_INDEX_3 3
```

Description

This is macro DRV_SPI_INDEX_3.

DRV_SPI_INDEX_4 Macro

File

drv_spi.h

C

```
#define DRV_SPI_INDEX_4 4
```

Description

This is macro DRV_SPI_INDEX_4.

DRV_SPI_INDEX_5 Macro

File

drv_spi.h

C

```
#define DRV_SPI_INDEX_5 5
```

Description

This is macro DRV_SPI_INDEX_5.

DRV_SPI_TASK_MODE Enumeration

Identifies the various modes of how the tasks function will be run.

File

drv_spi.h

C

```
typedef enum {
  DRV_SPI_TASK_MODE_POLLED,
  DRV_SPI_TASK_MODE_ISR
} DRV_SPI_TASK_MODE;
```

Members

Members	Description
DRV_SPI_TASK_MODE_POLLED	Task is configured to run in polled mode
DRV_SPI_TASK_MODE_ISR	Task is configured to run in interrupt mode

Description

SPI Task Modes Enumeration

This enumeration identifies the various tasks mode

Remarks

None.

DRV_SPI_CLIENT_DATA Structure

Defines the data that can be changed per client.

File

drv_spi.h

C

```
typedef struct _DRV_SPI_CLIENT_DATA {
  uint32_t baudRate;
  DRV_SPI_BUFFER_EVENT_HANDLER operationStarting;
  DRV_SPI_BUFFER_EVENT_HANDLER operationEnded;
} DRV_SPI_CLIENT_DATA;
```

Members

Members	Description
uint32_t baudRate;	Baud Rate Value
DRV_SPI_BUFFER_EVENT_HANDLER operationStarting;	This callback is fired when an operation is about to start on the SPI bus. This allows the user to set any pins that need to be set. This callback may be called from an ISR so should not include OSAL calls. The context parameter is the same one passed into the BufferAddRead, BufferAddWrite, BufferAddWriteRead function.
DRV_SPI_BUFFER_EVENT_HANDLER operationEnded;	This callback is fired when an operation has just completed on the SPI bus. This allows the user to set any pins that need to be set. This callback may be called from an ISR so should not include OSAL calls. The context parameter is the same one passed into the BufferAddRead, BufferAddWrite, BufferAddWriteRead function.

Description

SPI Driver Client Specific Configuration

This data type defines the data can be configured per client. This data can be per client, and overrides the configuration data contained inside of DRV_SPI_INIT.

Remarks

None.

Files

Files

Name	Description
drv_spi.h	SPI device driver interface file.
drv_spi_config_template.h	SPI Driver configuration definitions template.

Description

This section lists the source and header files used by the SPI Driver Library.

drv_spi.h

SPI device driver interface file.

Enumerations

	Name	Description
	DRV_SPI_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_SPI_BUFFER_TYPE	Identifies the various buffer types of the SPI module.
	DRV_SPI_CLOCK_MODE	Identifies the various clock modes of the SPI module.
	DRV_SPI_MODE	Identifies the various usage modes of the SPI module.
	DRV_SPI_PROTOCOL_TYPE	Identifies the various protocols of the SPI module.
	DRV_SPI_TASK_MODE	Identifies the various modes of how the tasks function will be run.

Functions

	Name	Description
	DRV_SPI_BufferAddRead	Registers a buffer for a read operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddRead2	Registers a buffer for a read operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWrite	Registers a buffer for a write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWrite2	Registers a buffer for a write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWriteRead	Registers a buffer for a read and write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferAddWriteRead2	Registers a buffer for a read and write operation. Actual transfer will happen in the Task function. **Implementation:** Static/Dynamic
	DRV_SPI_BufferStatus	Returns the transmitter and receiver transfer status. **Implementation:** Static/Dynamic
	DRV_SPI_ClientConfigure	Configures a SPI client with specific data. **Implementation:** Static/Dynamic
	DRV_SPI_Close	Closes an opened instance of the SPI driver. **Implementation:** Static/Dynamic
	DRV_SPI_Deinitialize	Deinitializes the specified instance of the SPI driver module. **Implementation:** Static/Dynamic
	DRV_SPI_Initialize	Initializes the SPI instance for the specified driver index. **Implementation:** Static/Dynamic
	DRV_SPI_Open	Opens the specified SPI driver instance and returns a handle to it. **Implementation:** Static/Dynamic
	DRV_SPI_Status	Provides the current status of the SPI driver module. **Implementation:** Static/Dynamic
	DRV_SPI_Tasks	Maintains the driver's state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_SPIn_ReceiverBufferIsFull	Returns the receive buffer status. 'n' represents the instance of the SPI driver used. **Implementation:** Static

	DRV_SPIn_TransmitterBufferIsFull	Returns the transmit buffer status. 'n' represents the instance of the SPI driver used. **Implementation:** Static

Macros

	Name	Description
	DRV_SPI_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_SPI_INDEX_0	SPI driver index definitions.
	DRV_SPI_INDEX_1	This is macro DRV_SPI_INDEX_1.
	DRV_SPI_INDEX_2	This is macro DRV_SPI_INDEX_2.
	DRV_SPI_INDEX_3	This is macro DRV_SPI_INDEX_3.
	DRV_SPI_INDEX_4	This is macro DRV_SPI_INDEX_4.
	DRV_SPI_INDEX_5	This is macro DRV_SPI_INDEX_5.
	DRV_SPI_INDEX_COUNT	Number of valid SPI driver indices.

Structures

	Name	Description
	_DRV_SPI_CLIENT_DATA	Defines the data that can be changed per client.
	_DRV_SPI_INIT	Defines the data required to initialize or reinitialize the SPI driver
	DRV_SPI_CLIENT_DATA	Defines the data that can be changed per client.
	DRV_SPI_INIT	Defines the data required to initialize or reinitialize the SPI driver

Types

	Name	Description
	DRV_SPI_BUFFER_EVENT_HANDLER	Pointer to a SPI Driver Buffer Event handler function
	DRV_SPI_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.

Description

SPI Driver Interface

The SPI driver provides a simple interface to manage the SPI module. This file defines the interface definitions and prototypes for the SPI driver.

File Name

drv_spi.h

Company

Microchip Technology Inc.

drv_spi_config_template.h

SPI Driver configuration definitions template.

Macros

	Name	Description
	DRV_SPI_16BIT	Controls the compilation of 16 Bit mode
	DRV_SPI_32BIT	Controls the compilation of 32 Bit mode
	DRV_SPI_8BIT	Controls the compilation of 8 Bit mode
	DRV_SPI_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_SPI_DMA	Controls the compilation of DMA support
	DRV_SPI_DMA_DUMMY_BUFFER_SIZE	Controls the size of DMA dummy buffer
	DRV_SPI_DMA_TXFER_SIZE	Controls the size of DMA transfers
	DRV_SPI_EBM	Controls the compilation of Enhanced Buffer Mode mode
	DRV_SPI_ELEMENTS_PER_QUEUE	Controls the number of elements that are allocated.
	DRV_SPI_INSTANCES_NUMBER	Selects the maximum number of hardware instances that can be supported by the dynamic driver .
	DRV_SPI_ISR	Controls the compilation of ISR mode
	DRV_SPI_MASTER	Controls the compilation of master mode
	DRV_SPI_POLLED	Controls the compilation of Polled mode
	DRV_SPI_RM	Controls the compilation of Standard Buffer mode

	DRV_SPI_SLAVE	Controls the compilation of slave mode

Description

SPI Driver Configuration Definitions for the Template Version

These definitions statically define the driver's mode of operation.

File Name

drv_spi_config_template.h

Company

Microchip Technology Inc.

SPI Flash Driver Libraries

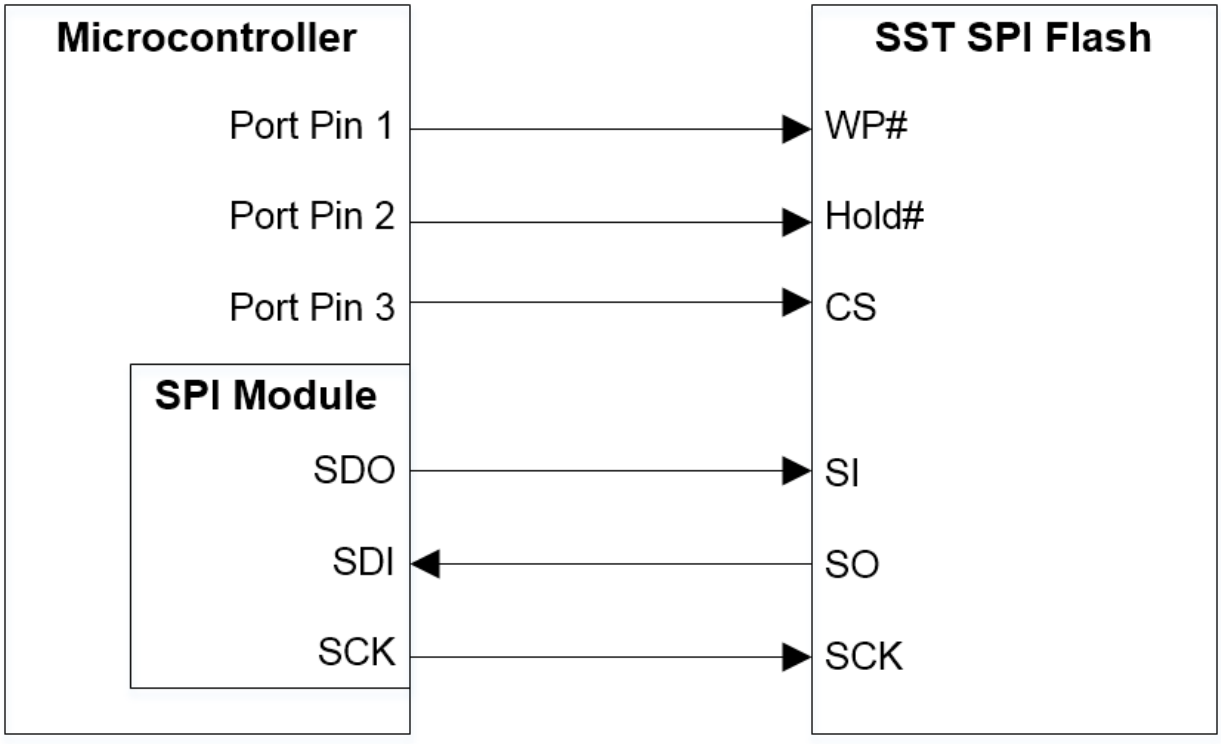
This section describes the Serial Peripheral Interface (SPI) Flash Driver Libraries.

Introduction

This library provides an interface to manage the SST SPI Flash modules (SST25VF020B, SST25VF016B, and SST25VF064C) in different modes of operation.

Description

The SPI Flash Driver uses SPI interface to establish the communication between SST Flash and Microchip microcontrollers. The SPI module of the controller works as a Master device and the Flash module works as a Slave. The following diagram shows the pin connections that are required to make the driver operational:



The SPI Flash Driver is dynamic in nature, so single instance of it can support multiple clients that want to use the same Flash. Multiple instances of the driver can be used when multiple Flash devices are required to be part of the system. The SPI Driver, which is used by the SPI Flash Driver, can be configured for use in either Polled or Interrupt mode.

Using the Library

This topic describes the basic architecture of the SPI Flash Driver Library and provides information and examples on its use.

Description

Interface Header Files: `drv_sst25vf016b.h`, `drv_sst25vf020b.h`, or `drv_sst25vf064c.h`

The interface to the SPI Flash Driver Library is defined in the header file. Any C language source (`.c`) file that uses the SPI Flash Driver library should include this header.

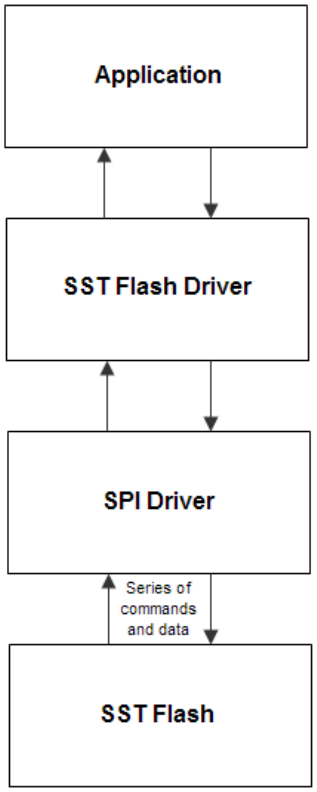
Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the SPI Flash Driver Library with a convenient C language interface. This topic describes how that abstraction is modeled in software.

Description

The SST SPI Flash needs a specific set of commands to be given on its SPI interface along with the required address and data to do different operations. This driver abstracts these requirements and provide simple APIs that can be used to perform Erase, Write, and Read operations. The SPI Driver is used for this purpose. The following layered diagram depicts the communication between different modules.



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SPI Flash module.

Library Interface Section	Description
System Functions	These functions are accessed by the MPLAB Harmony System module and allow the driver to be initialized, deinitialized, and maintained.
Core Client Functions	These functions allow the application client to open and close the driver.
Block Operation Functions	These functions enable the Flash module to be erased, written, and read (to/from).
Media Interface Functions	These functions provide media status and the Flash geometry.

How the Library Works

The library provides interfaces to support:

- System Initialization/Deinitialization

- Opening the Driver
- Block Operations

System Initialization and Deinitialization

Provides information on initializing the system.

Description

System Initialization and Deinitialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization each instance of the SST Flash module would be initialized with the following configuration settings (either passed dynamically at run-time using DRV_SST25VF020B_INIT, DRV_SST25VF016B_INIT, or DRV_SST25VF064C_INIT, or by using Initialization Overrides) that are supported or used by the specific SST Flash device hardware:

- Device requested power state: one of the System Module Power States. For specific details please refer to **Data Types and Constants** in the Library Interface section
- The SPI Driver Module Index which is intended to be used to communicate with SST Flash (e.g., DRV_SPI_INDEX_0)
- Port Pins of the microcontroller to be used for Chip Select, Write Protection, and Hold operations on the SST Flash device
- Maximum Buffer Queue Size for that instance of the SST Flash Driver

Using the SST25VF020B as an example, the DRV_SST25VF020B_Initialize function returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the Initialize interface would be used by the other system interfaces like DRV_SST25VF020B_Deinitialize, DRV_SST25VF020B_Status, and DRV_SST25VF020B_Tasks.

Note: The system initialization and the deinitialization settings, only affect the instance of the peripheral that is being initialized or deinitialized.

Example:

```
// This code example shows the initialization of the SST25VF020B SPI Flash
// Driver. SPI driver index 0 is used for the purpose. Pin numbers 1, 2,
// and 3 of PORTB are configured for the Hold pin, Write Protection pin, and
// the Chip Select pin, respectively. The maximum buffer queue size is set to 5.

   DRV_SST25VF020B_INIT   SST25VF020BInitData;
   SYS_MODULE_OBJ      objectHandle;

   SST25VF020BInitData.moduleInit.value        = SYS_MODULE_POWER_RUN_FULL;
   SST25VF020BInitData.spiDriverModuleIndex    = DRV_SPI_INDEX_0;
   SST25VF020BInitData.holdPortChannel         = PORT_CHANNEL_B;
   SST25VF020BInitData.holdBitPosition         = PORTS_BIT_POS_1;
   SST25VF020BInitData.writeProtectPortChannel = PORT_CHANNEL_B;
   SST25VF020BInitData.writeProtectBitPosition = PORTS_BIT_POS_2;
   SST25VF020BInitData.chipSelectPortChannel   = PORT_CHANNEL_F;
   SST25VF020BInitData.chipSelectBitPosition   = PORTS_BIT_POS_2;
   SST25VF020BInitData.queueSize               = 5;

   objectHandle = DRV_SST25VF020B_Initialize(DRV_SST25VF020B_INDEX_0,
                                (SYS_MODULE_INIT*)SST25VF020BInitData);
   if (SYS_MODULE_OBJ_INVALID == objectHandle)
   {
       // Handle error
   }
```

Tasks Routine

The system will either call DRV_SST25VF020B_Tasks, from SYS_Tasks (in a polled environment) or DRV_SST25VF020B_Tasks will be called from the ISR of the SPI module in use.

Opening the Driver

Provides information on opening the driver.

Description

To use the SST Flash driver, the application must open the driver. Using the SST25VF020B as an example, this is done by calling the DRV_SST25VF020B_Open function. Calling this function with DRV_IO_INTENT_NONBLOCKING will cause the driver to be opened in non blocking mode. Then DRV_SST25VF020B_BlockErase, DRV_SST25VF020B_BlockWrite and DRV_SST25VF020B_BlockRead functions when called by this client will be non-blocking.

The client can also open the driver in Read-only mode (DRV_IO_INTENT_READ), Write-only mode (DRV_IO_INTENT_WRITE), and Exclusive mode (DRV_IO_INTENT_EXCLUSIVE). If the driver has been opened exclusively by a client, it cannot be opened again by another client.

If successful, the DRV_SST25VF020B_Open function will return a handle to the driver. This handle records the association between the client and the driver instance that was opened. The DRV_SST25VF020B_Open function may return DRV_HANDLE_INVALID in the situation where the driver is not ready to be opened. When this occurs, the application can try opening the driver again. Note that the open function may return an invalid handle in other (error) cases as well.

The following code shows an example of the driver being opened in different modes.

```
DRV_HANDLE sstHandle1, sstHandle2;

/* Client 1 opens the SST driver in non blocking mode */
sstHandle1 = DRV_SST25VF020B_Open(DRV_SST25VF020B_INDEX_0, DRV_IO_INTENT_NONBLOCKING);

/* Check if the handle is valid */
if(DRV_HANDLE_INVALID == sstHandle1)
{
    /* The driver was not opened successfully. The client
     * can try opening it again */
}

/* Client 2 opens the SST driver in Exclusive Write only mode */
sstHandle2 = DRV_SST25VF020B_Open(DRV_SST25VF020B_INDEX_0, DRV_IO_INTENT_WRITE | DRV_IO_INTENT_EXCLUSIVE);

/* Check if the handle is valid */
if(DRV_HANDLE_INVALID == sstHandle2)
{
    /* The driver was not opened successfully. The client
     * can try opening it again */
}
```

Block Operations

Provides information on block operations.

Description

This driver provides simple client interfaces to Erase, Write, and Read the SST flash in blocks. A block is the unit to represent minimum amount of data that can be erased, written, or read. Block size may differ for Erase, Write, and Read operations. Using the SST25VF020B as an example, the DRV_SST25VF020B_GeometryGet function can be used to determine the different block sizes for the driver.

The DRV_SST25VF020B_BlockErase, DRV_SST25VF020B_BlockWrite, and DRV_SST25VF020B_BlockRead functions are used to erase, write, and read the data to/from SST SPI Flash. These functions are always non-blocking. All of these functions follow a standard queue model to read, write, and erase. When any of these functions are called (i.e., a block request is made), the request is queued. The size of the queue is determined by the queueSize member of the DRV_SST25VF020B_INIT data structure. All of the requests in the queue are executed by the DRV_SST25VF020B_Tasks function one-by-one.

When the driver adds a request to the queue, it returns a buffer handle. This handle allows the client to track the request as it progresses through the queue. The buffer handle expires when the event associated with the buffer completes. The driver provides driver events (DRV_SST25VF020B_BLOCK_EVENT) that indicate termination of the buffer requests.

The following steps can be performed for a simple Block Data Operation:

1. The system should have completed necessary initialization of the SPI Driver and the SST Flash Driver, and the DRV_SST25VF020B_Tasks function should be running in a polled environment.
2. The DRV_SPI_Tasks function should be running in either a polled environment or an interrupt environment.
3. Open the driver using DRV_SST25VF020B_Open with the necessary intent.
4. Set an event handler callback using the function DRV_SST25VF020B_BlockEventHandlerSet.
5. Request for block operations using the functions, DRV_SST25VF020B_BlockErase, DRV_SST25VF020B_BlockWrite, and DRV_SST25VF020B_BlockRead, with the appropriate parameters.
6. Wait for event handler callback to occur and check the status of the block operation using the callback function parameter of type DRV_SST25VF020B_BLOCK_EVENT.
7. The client will be able to close the driver using the function, DRV_SST25VF020B_Close, when required.

Example:

```
/* This code example shows usage of the block operations
 * on the SPI Flash SST25VF020B device */

DRV_HANDLE sstHandle1;
uint8_t myData1[10], myData2[10];
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE  blockHandle1, blockHandle2, blockHandle3;

/* The driver is opened for read-write in Exclusive mode */
```

```
sstHandle1 = DRV_SST25VF020B_Open(DRV_SST25VF020B_INDEX_0,
                                  DRV_IO_INTENT_READWRITE | DRV_IO_INTENT_EXCLUSIVE);

/* Check if the driver was opened successfully */
if(DRV_HANDLE_INVALID == sstHandle1)
{
    /* The driver could not be opened successfully */
}

/* Register a Buffer Event Handler with SST25VF020B  driver.
 * This event handler function will be called whenever
 * there is a buffer event.  An application defined
 * context can also be specified. This is returned when
 * the event handler is called.
 * */
DRV_SST25VF020B_BlockEventHandlerSet(sstHandle1,
        APP_SSTBufferEventHandler, NULL);

/* Request for all the three block operations one by one */

/* first block API to erase 1 block of the flash starting from address 0x0, each block is of 4kbyte */
DRV_SST25VF020B_BlockErase(sstHandle1, &blockHandle1, 0x0, 1);
/* 2nd block API to write myData1 in the first 10 locations of the flash */
DRV_SST25VF020B_BlockWrite(sstHandle1, &blockHandle2, &myData1[0], 0x0, 10);
/* 3rd block API to read the first 10 locations of the flash into myData2 */
DRV_SST25VF020B_BlockRead(sstHandle1, &blockHandle3, &myData2[0], 0x0, 10);

/* This is the Driver Event Handler */

void APP_SSTBufferEventHandler(DRV_SST25VF020B_BLOCK_EVENT event,
        DRV_SST25VF020B_BLOCK_COMMAND_HANDLE blockHandle, uintptr_t contextHandle)
{
    switch(event)
    {
        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE:
    if ( blockHandle == blockHandle3)
    {
            /* This means the data was read */
    /* Do data verification/processing */
    }
            break;
        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR:
            /* Error handling here. */
            break;
        default:
            break;
    }
}
```

Configuring the Library

SST25VF016B Configuration

	Name	Description
	DRV_SST25VF016B_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_SST25VF016B_HARDWARE_HOLD_ENABLE	Specifies if the hardware hold feature is enabled or not.
	DRV_SST25VF016B_HARDWARE_WRITE_PROTECTION_ENABLE	Specifies if the hardware write protect feature is enabled or not.
	DRV_SST25VF016B_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_SST25VF016B_MODE	Determines whether the driver is implemented as static or dynamic
	DRV_SST25VF016B_QUEUE_DEPTH_COMBINED	Number of entries of queues in all instances of the driver.

SST25VF020B Configuration

	Name	Description
	DRV_SST25VF020B_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_SST25VF020B_HARDWARE_HOLD_ENABLE	Specifies if the hardware hold feature is enabled or not.
	DRV_SST25VF020B_HARDWARE_WRITE_PROTECTION_ENABLE	Specifies if the hardware write protect feature is enabled or not.
	DRV_SST25VF020B_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_SST25VF020B_MODE	Determines whether the driver is implemented as static or dynamic.
	DRV_SST25VF020B_QUEUE_DEPTH_COMBINED	Number of entries of queues in all instances of the driver.

SST25VF064C Configuration

	Name	Description
	DRV_SST25VF064C_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_SST25VF064C_HARDWARE_HOLD_ENABLE	Specifies whether or not the hardware hold feature is enabled.
	DRV_SST25VF064C_HARDWARE_WRITE_PROTECTION_ENABLE	Specifies whether or not the hardware write protect feature is enabled.
	DRV_SST25VF064C_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_SST25VF064C_MODE	Determines whether the driver is implemented as static or dynamic.
	DRV_SST25VF064C_QUEUE_DEPTH_COMBINED	Number of entries of queues in all instances of the driver.

Description

The SST Flash Driver requires the specification of compile-time configuration macros. These macros define resource usage, feature availability, and dynamic behavior of the driver. These configuration macros should be defined in the `system_config.h` file.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

SST25VF016B Configuration

DRV_SST25VF016B_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_sst25vf016b_config_template.h

C

```
#define DRV_SST25VF016B_CLIENTS_NUMBER 4
```

Description

SST25VF016B Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. This value represents the total number of clients to be supported across all hardware instances. So if SST25VF016B-1 will be accessed by 2 clients and SST25VF016B-2 will accessed by 3 clients, then this number should be 5. It is recommended that this be set exactly equal to the number of expected clients. Client support consumes RAM memory space. If this macro is not defined and the DRV_SST25VF016B_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - single client operation. If this macro is defined and the DRV_SST25VF016B_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - multi client operation.

Remarks

None.

DRV_SST25VF016B_HARDWARE_HOLD_ENABLE Macro

Specifies if the hardware hold feature is enabled or not.

File

drv_sst25vf016b_config_template.h

C

```
#define DRV_SST25VF016B_HARDWARE_HOLD_ENABLE false
```

Description

SST25VF016B Hardware HOLD Support

This macro defines if the hardware hold feature is enabled or not. If hardware hold is enabled, then user must provide a port pin corresponding to HOLD pin on the flash

Remarks

None

DRV_SST25VF016B_HARDWARE_WRITE_PROTECTION_ENABLE Macro

Specifies if the hardware write protect feature is enabled or not.

File

drv_sst25vf016b_config_template.h

C

```
#define DRV_SST25VF016B_HARDWARE_WRITE_PROTECTION_ENABLE false
```

Description

SST25VF016B Hardware Write Protect Support

This macro defines if the hardware Write Protect feature is enabled or not. If hardware write protection is enabled, then user must provide a port pin corresponding to WP pin on the flash

Remarks

None.

DRV_SST25VF016B_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported

File

drv_sst25vf016b_config_template.h

C

```
#define DRV_SST25VF016B_INSTANCES_NUMBER 2
```

Description

SST25VF016B driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of SST25VF016B modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None.

DRV_SST25VF016B_MODE Macro

Determines whether the driver is implemented as static or dynamic

File

drv_sst25vf016b_config_template.h

C

```
#define DRV_SST25VF016B_MODE DYNAMIC
```

Description

SST25VF016B mode

Determines whether the driver is implemented as static or dynamic. Static drivers control the peripheral directly with peripheral library routines.

Remarks

None.

DRV_SST25VF016B_QUEUE_DEPTH_COMBINED Macro

Number of entries of queues in all instances of the driver.

File

drv_sst25vf016b_config_template.h

C

```
#define DRV_SST25VF016B_QUEUE_DEPTH_COMBINED 7
```

Description

SST25VF016B Driver Instance combined queue depth.

This macro defines the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for all the read/write/erase operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build).

A buffer queue will contain buffer queue entries, each related to a BufferAdd request. This configuration macro defines total number of buffer entries that will be available for use between all SST25VF016B driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking erase/write/read requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. More the number of buffer entries, greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its buffer queue size.

SST25VF020B Configuration

DRV_SST25VF020B_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_sst25vf020b_config_template.h

C

```
#define DRV_SST25VF020B_CLIENTS_NUMBER 4
```

Description

SST25VF020B Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. This value represents the total number of clients to be supported across all hardware instances. So if SST25VF020B-1 will be accessed by 2 clients and SST25VF020B-2 will accessed by 3 clients, then this number should be 5. It is recommended that this be set exactly equal to the number of expected clients. Client support consumes RAM memory space. If this macro is not defined and the DRV_SST25VF020B_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - single client operation. If this macro is defined and the DRV_SST25VF020B_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - multi client operation.

Remarks

None.

DRV_SST25VF020B_HARDWARE_HOLD_ENABLE Macro

Specifies if the hardware hold feature is enabled or not.

File

drv_sst25vf020b_config_template.h

C

```
#define DRV_SST25VF020B_HARDWARE_HOLD_ENABLE false
```

Description

SST25VF020B Hardware HOLD Support

This macro defines if the hardware hold feature is enabled or not. If hardware hold is enabled, then user must provide a port pin corresponding to HOLD pin on the flash

Remarks

None.

DRV_SST25VF020B_HARDWARE_WRITE_PROTECTION_ENABLE Macro

Specifies if the hardware write protect feature is enabled or not.

File

drv_sst25vf020b_config_template.h

C

```
#define DRV_SST25VF020B_HARDWARE_WRITE_PROTECTION_ENABLE false
```

Description

SST25VF020B Hardware Write Protect Support

This macro defines if the hardware Write Protect feature is enabled or not. If hardware write protection is enabled, then user must provide a port pin corresponding to WP pin on the flash

Remarks

None.

DRV_SST25VF020B_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported.

File

drv_sst25vf020b_config_template.h

C

```
#define DRV_SST25VF020B_INSTANCES_NUMBER 2
```

Description

SST25VF020B driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of SST25VF020B modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None.

DRV_SST25VF020B_MODE Macro

Determines whether the driver is implemented as static or dynamic.

File

drv_sst25vf020b_config_template.h

C

```
#define DRV_SST25VF020B_MODE DYNAMIC
```

Description

SST25VF020B mode

Determines whether the driver is implemented as static or dynamic. Static drivers control the peripheral directly with peripheral library routines.

Remarks

None.

DRV_SST25VF020B_QUEUE_DEPTH_COMBINED Macro

Number of entries of queues in all instances of the driver.

File

drv_sst25vf020b_config_template.h

C

```
#define DRV_SST25VF020B_QUEUE_DEPTH_COMBINED 7
```

Description

SST25VF020B Driver Instance combined queue depth.

This macro defines the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for all the read/write/erase operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build).

A buffer queue will contain buffer queue entries, each related to a BufferAdd request. This configuration macro defines total number of buffer entries that will be available for use between all SST25VF020B driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking erase/write/read requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. More the number of buffer entries, greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its buffer queue size.

SST25VF064C Configuration

DRV_SST25VF064C_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_sst25vf064c_config_template.h

C

```
#define DRV_SST25VF064C_CLIENTS_NUMBER 4
```

Description

SST25VF064C Client Count Configuration

Sets up the maximum number of clients that can be connected to any hardware instance. This value represents the total number of clients to be supported across all hardware instances. So if SST25VF064C-1 will be accessed by 2 clients and SST25VF064C-2 will accessed by 3 clients, then this number should be 5. It is recommended that this be set exactly equal to the number of expected clients. Client support consumes RAM memory space. If this macro is not defined and the DRV_SST25VF064C_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - single client operation. If this macro is defined and the DRV_SST25VF064C_INSTANCES_NUMBER macro is not defined, then the driver will be built for static - multi client operation.

Remarks

None.

DRV_SST25VF064C_HARDWARE_HOLD_ENABLE Macro

Specifies whether or not the hardware hold feature is enabled.

File

drv_sst25vf064c_config_template.h

C

```
#define DRV_SST25VF064C_HARDWARE_HOLD_ENABLE false
```

Description

SST25VF064C Hardware HOLD Support

This macro defines whether or not the hardware hold feature is enabled. If hardware hold is enabled, the user must provide a port pin corresponding to the HOLD pin on the Flash device.

Remarks

None.

DRV_SST25VF064C_HARDWARE_WRITE_PROTECTION_ENABLE Macro

Specifies whether or not the hardware write protect feature is enabled.

File

drv_sst25vf064c_config_template.h

C

```
#define DRV_SST25VF064C_HARDWARE_WRITE_PROTECTION_ENABLE false
```

Description

SST25VF064C Hardware Write Protect Support

This macro defines whether or not the hardware Write Protect feature is enabled. If hardware write protection is enabled, the user must provide a port pin corresponding to the WP pin on the Flash device.

Remarks

None.

DRV_SST25VF064C_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported.

File

drv_sst25vf064c_config_template.h

C

```
#define DRV_SST25VF064C_INSTANCES_NUMBER 2
```

Description

SST25VF064C driver objects configuration

Sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of SST25VF064C modules that are needed by the application. Hardware Instance support consumes RAM memory space. If this macro is not defined, then the driver will be built statically.

Remarks

None.

DRV_SST25VF064C_MODE Macro

Determines whether the driver is implemented as static or dynamic.

File

drv_sst25vf064c_config_template.h

C

```
#define DRV_SST25VF064C_MODE DYNAMIC
```

Description

SST25VF064C mode

Determines whether the driver is implemented as static or dynamic. Static drivers control the peripheral directly with peripheral library routines.

Remarks

None.

DRV_SST25VF064C_QUEUE_DEPTH_COMBINED Macro

Number of entries of queues in all instances of the driver.

File

drv_sst25vf064c_config_template.h

C

```
#define DRV_SST25VF064C_QUEUE_DEPTH_COMBINED 7
```

Description

SST25VF064C Driver Instance combined queue depth.

This macro defines the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for all the read/write/erase operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build).

A buffer queue will contain buffer queue entries, each related to a BufferAdd request. This configuration macro defines total number of buffer entries that will be available for use between all SST25VF064C driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking erase/write/read requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. More the number of buffer entries, greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its buffer queue size.

Building the Library

This section lists the files that are available in the SPI Flash Driver Library.

Description

This section list the files that are available in the `/src` folder of the SPI Flash Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/spi_flash`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`sst25vf016b/drv_sst25vf016b.h`	Header file that exports the SST25VF016B driver API.
`sst25vf020b/drv_sst25vf020b.h`	Header file that exports the SST25VF020B driver API.
`sst25vf064c/drv_sst25vf064c.h`	Header file that exports the SST25VF064C driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`sst25vf016b/src/dynamic/drv_sst25vf016b.c`	Basic SPI Flash Driver SST25VF016B implementation file.

sst25vf020b/src/dynamic/drv_sst25vf020b.c	Basic SPI Flash Driver SST25VF020B implementation file.
sst25vf064c/src/dynamic/drv_sst25vf064c.c	Basic SPI Flash Driver SST25VF064C implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
sst25vf020b/src/dynamic/drv_sst25vf020b_erasewrite.c	This file implements an optional BlockEraseWrite feature for the SST25VF020B driver.

Module Dependencies

The SPI Flash Driver Library depends on the following modules:

- SPI Driver Library
- Ports System Service Library

Library Interface

This section describes the API functions of the SPI Flash Driver Library.

Refer to each section for a detailed description.

SST25FV016B API

a) System Functions

	Name	Description
	DRV_SST25VF016B_Initialize	Initializes the SST25VF016B SPI Flash Driver instance for the specified driver index. **Implementation:** Dynamic
	DRV_SST25VF016B_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module. **Implementation:** Dynamic
	DRV_SST25VF016B_Status	Gets the current status of the SPI Flash Driver module. **Implementation:** Dynamic
	DRV_SST25VF016B_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR. **Implementation:** Dynamic

b) Core Client Functions

	Name	Description
	DRV_SST25VF016B_Close	Closes an opened-instance of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF016B_Open	Opens the specified SPI Flash driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_SST25VF016B_ClientStatus	Gets current client-specific status of the SPI Flash driver. **Implementation:** Dynamic

c) Block Operation Functions

	Name	Description
	DRV_SST25VF016B_BlockErase	Erase the specified number of blocks in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF016B_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Dynamic
	DRV_SST25VF016B_BlockRead	Reads blocks of data starting from the specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF016B_BlockWrite	Write blocks of data starting from a specified address in Flash memory. **Implementation:** Dynamic

d) Media Interface Functions

	Name	Description
	DRV_SST25VF016B_GeometryGet	Returns the geometry of the device. **Implementation:** Dynamic
	DRV_SST25VF016B_MediaIsAttached	Returns the status of the media. **Implementation:** Dynamic

e) Data Types and Constants

	Name	Description
	DRV_SST25VF016B_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.
	DRV_SST25VF016B_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_SST25VF016B_CLIENT_STATUS	Defines the client status. **Implementation:** Dynamic
	DRV_SST25VF016B_EVENT_HANDLER	Pointer to a SST25VF016B SPI Flash Driver Event handler function. **Implementation:** Dynamic
	DRV_SST25VF016B_INIT	Contains all the data necessary to initialize the SPI Flash device. **Implementation:** Dynamic
	DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_SST25VF016B_INDEX_0	SPI Flash driver index definitions
	DRV_SST25VF016B_INDEX_1	This is macro DRV_SST25VF016B_INDEX_1.

Description

This section contains the SST25V016B Flash device API.

a) System Functions

DRV_SST25VF016B_Initialize Function

Initializes the SST25VF016B SPI Flash Driver instance for the specified driver index.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
SYS_MODULE_OBJ DRV_SST25VF016B_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the SPI Flash driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This function must be called before any other SPI Flash function is called.

This function should only be called once during system initialization unless DRV_SST25VF016B_Deinitialize is called to deinitialize the driver instance.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Preconditions

None.

Example

```
// This code snippet shows an example of initializing the SST25VF016B SPI
// Flash Driver. SPI driver index 0 is used for the purpose. Pin numbers 1, 2
// and 3 of port channel B are configured for hold pin, write protection pin
```

```
// and chip select pin respectively. Maximum buffer queue size is set 5.

DRV_SST25VF016B_INIT   SST25VF016BInitData;
SYS_MODULE_OBJ      objectHandle;

SST25VF016BInitData.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
SST25VF016BInitData.spiDriverModuleIndex  = DRV_SPI_INDEX_0;
SST25VF016BInitData.holdPortChannel       = PORT_CHANNEL_B;
SST25VF016BInitData.holdBitPosition       = PORTS_BIT_POS_1;
SST25VF016BInitData.writeProtectPortChannel = PORT_CHANNEL_B;
SST25VF016BInitData.writeProtectBitPosition = PORTS_BIT_POS_2;
SST25VF016BInitData.chipSelectPortChannel = PORT_CHANNEL_F;
SST25VF016BInitData.chipSelectBitPosition = PORTS_BIT_POS_2;
SST25VF016BInitData.queueSize = 5;

objectHandle = DRV_SST25VF016B_Initialize(DRV_SST25VF016B_INDEX_0,
                                (SYS_MODULE_INIT*)SST25VF016BInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_SST25VF016B_Initialize
(
const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init
);

DRV_SST25VF016B_Deinitialize Function

Deinitializes the specified instance of the SPI Flash driver module.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SPI Flash Driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This function will NEVER block waiting for hardware.

Preconditions

Function DRV_SST25VF016B_Initialize should have been called before calling this function.

Example

```
// This code snippet shows an example of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_SST25VF016B_Initialize
SYS_STATUS          status;
```

```
DRV_SST25VF016B_Deinitialize(object);

status = DRV_SST25VF016B_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SST25VF016B_Initialize

Function

void DRV_SST25VF016B_Deinitialize(SYS_MODULE_OBJ object)

DRV_SST25VF016B_Status Function

Gets the current status of the SPI Flash Driver module.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
SYS_STATUS DRV_SST25VF016B_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and accept requests for new operations

SYS_STATUS_UNINITIALIZED - Indicates that the driver is not initialized

Description

This function provides the current status of the SPI Flash Driver module.

Remarks

A driver can only be opened when its status is SYS_STATUS_READY.

Preconditions

Function DRV_SST25VF016B_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST25VF016B_Initialize
SYS_STATUS          SST25VF016BStatus;

SST25VF016BStatus = DRV_SST25VF016B_Status(object);
else if (SYS_STATUS_ERROR >= SST25VF016BStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SST25VF016B_Initialize

Function

SYS_STATUS DRV_SST25VF016B_Status(SYS_MODULE_OBJ object)

DRV_SST25VF016B_Tasks Function

Maintains the driver's read, erase, and write state machine and implements its ISR.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal state machine and should be called from the system's Tasks function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks function (SYS_Tasks).

Preconditions

The DRV_SST25VF016B_Initialize function must have been called for the specified SPI Flash driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST25VF016B_Initialize

while (true)
{
    DRV_SST25VF016B_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_SST25VF016B_Initialize)

Function

void DRV_SST25VF016B_Tasks (SYS_MODULE_OBJ object);

b) Core Client Functions

DRV_SST25VF016B_Close Function

Closes an opened-instance of the SPI Flash driver.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the SPI Flash driver, invalidating the handle.

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_SST25VF016B_Open before the caller may use the driver again.

Usually, there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_SST25VF016B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF016B_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_SST25VF016B_Open

DRV_SST25VF016B_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_SST25VF016B_Close(DRV_Handle handle);

DRV_SST25VF016B_Open Function

Opens the specified SPI Flash driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
DRV_HANDLE DRV_SST25VF016B_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_SST25VF016B_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.
- if the driver status is not ready.

The driver status becomes ready inside "DRV_SST25VF016B_Tasks" function. To make the SST Driver status ready and hence successfully "Open" the driver, "Task" routine need to be called periodically.

Description

This function opens the specified SPI Flash driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The driver will always work in Non-Blocking mode even if IO-intent is selected as blocking.

The handle returned is valid until the DRV_SST25VF016B_Close function is called.

This function will NEVER block waiting for hardware.

Preconditions

Function DRV_SST25VF016B_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_SST25VF016B_Open(DRV_SST25VF016B_INDEX_0,
                                                DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
```

```
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

```
DRV_HANDLE DRV_SST25VF016B_Open
(
const    SYS_MODULE_INDEX drvIndex,
const    DRV_IO_INTENT ioIntent
);
```

DRV_SST25VF016B_ClientStatus Function

Gets current client-specific status of the SPI Flash driver.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
DRV_SST25VF016B_CLIENT_STATUS DRV_SST25VF016B_ClientStatus(const DRV_HANDLE handle);
```

Returns

A DRV_SST25VF016B_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the SPI Flash driver associated with the given handle.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_SST25VF016B_Initialize function must have been called.

DRV_SST25VF016B_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      handle;         // Returned from DRV_SST25VF016B_Open
DRV_SST25VF016B_CLIENT_STATUS     clientStatus;

clientStatus = DRV_SST25VF016B_ClientStatus(handle);
if(DRV_SST25VF016B_CLIENT_STATUS_READY == clientStatus)
{
    // do the tasks
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's open

Function

```
DRV_SST25VF016B_CLIENT_STATUS DRV_SST25VF016B_ClientStatus(DRV_HANDLE handle);
```

c) Block Operation Functions

DRV_SST25VF016B_BlockErase Function

Erase the specified number of blocks in Flash memory.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_BlockErase(const DRV_HANDLE handle, DRV_SST25VF016B_BLOCK_COMMAND_HANDLE *
commandHandle, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It Will be DRV_BUFFER_HANDLE_INVALID if the request was not queued.

Description

This function schedules a non-blocking erase operation in flash memory. The function returns with a valid erase handle in the commandHandle argument if the erase request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the client opened the driver for read only
- if nBlock is 0
- if the queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SST25VF016B_EVENT_ERASE_COMPLETE event if the erase operation was successful or DRV_SST25VF016B_EVENT_ERASE_ERROR event if the erase operation was not successful.

Remarks

Write Protection will be disabled for the complete flash memory region in the beginning by default.

Preconditions

The DRV_SST25VF016B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF016B_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF016B_Open call.

Example

```
// Destination address should be block aligned.
uint32_t blockStart;
uint32_t nBlock;
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF016BHandle is the handle returned
// by the DRV_SST25VF016B_Open function.

// Client registers an event handler with driver

DRV_SST25VF016B_BlockEventHandlerSet(mySST25VF016BHandle,
                APP_SST25VF016BEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF016B_BlockErase( mySST25VF016BHandle, commandHandle,
                                                    blockStart, nBlock );

if(DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer queue is processed.

void APP_SST25VF016BEventHandler(DRV_SST25VF016B_BLOCK_EVENT event,
        DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
```

```
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF016B_EVENT_ERASE_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF016B_EVENT_ERASE_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
blockStart	Start block address in SST25VF016B memory from where the erase should begin. LSBs (A0-A11) of block start address will be ignored to align it with Erase block size boundary.
nBlock	Total number of blocks to be erased. Each Erase block is of size 4 KByte.

Function

void DRV_SST25VF016B_BlockErase
(
const DRV_HANDLE handle,
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE * commandHandle,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SST25VF016B_BlockEventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_BlockEventHandlerSet(const DRV_HANDLE handle, const DRV_SST25VF016B_EVENT_HANDLER
eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client calls any read, write or erase function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any read/write/erase operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_SST25VF016B_Initialize function must have been called for the specified SPI FLash driver instance.

DRV_SST25VF016B_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle;

// mySST25VF016BHandle is the handle returned
// by the DRV_SST25VF016B_Open function.

// Client registers an event handler with driver. This is done once.

DRV_SST25VF016B_BlockEventHandlerSet( mySST25VF016BHandle,
                APP_SST25VF016BEventHandler, (uintptr_t)&myAppObj );

DRV_SST25VF016B_BlockRead( mySST25VF016BHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_SST25VF016BEventHandler(DRV_SST25VF016B_BLOCK_EVENT event,
        DRV_SST25VF016B_BLOCK_COMMAND_HANDLE handle, uintptr_t context)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_SST25VF016B_BlockEventHandlerSet

```
(
const      DRV_HANDLE handle,
const      DRV_SST25VF016B_EVENT_HANDLER eventHandler,
const uintptr_t context
);
```

DRV_SST25VF016B_BlockRead Function

Reads blocks of data starting from the specified address in Flash memory.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_BlockRead(const DRV_HANDLE handle, DRV_SST25VF016B_BLOCK_COMMAND_HANDLE *
commandHandle, uint8_t * targetBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking read operation for reading blocks of data from flash memory. The function returns with a valid handle in the commandHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the target buffer pointer is NULL
- if the client opened the driver for write only
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a
DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully of
DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

The maximum read speed is 33 MHz.

Preconditions

The DRV_SST25VF016B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF016B_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF016B_Open call.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = SST25VF016B_BASE_ADDRESS_TO_READ_FROM;
uint32_t    nBlock = 2;
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF016BHandle is the handle returned
// by the DRV_SST25VF016B_Open function.

// Client registers an event handler with driver

DRV_SST25VF016B_BlockEventHandlerSet(mySST25VF016BHandle,
                APP_SST25VF016BEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF016B_BlockRead( mySST25VF016BHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );
```

```
if(DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.

void APP_SST25VF016BEventHandler(DRV_SST25VF016B_BLOCK_EVENT event,
        DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
*targetBuffer	Buffer into which the data read from the SPI Flash instance will be placed
blockStart	Start block address in SST25VF016B memory from where the read should begin. It can be any address of the flash.
nBlock	Total number of blocks to be read. Each Read block is of 1 byte.

Function

void DRV_SST25VF016B_BlockRead
(
const DRV_HANDLE handle,
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t *targetBuffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SST25VF016B_BlockWrite Function

Write blocks of data starting from a specified address in Flash memory.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
void DRV_SST25VF016B_BlockWrite(DRV_HANDLE handle, DRV_SST25VF016B_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t * sourceBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking write operation for writing blocks of data into flash memory. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only
- if the buffer size is 0
- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a
DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully or
DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

In the case of multi bytes write operation, byte by byte writing will happen instead of Address auto Increment writing.

Write Protection will be disabled for the complete flash memory region in the beginning by default.

Preconditions

The DRV_SST25VF016B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF016B_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF016B_Open call.

The flash address location which has to be written, must be erased before using the API DRV_SST25VF016B_BlockErase().

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = SST25VF016B_BASE_ADDRESS_TO_WRITE_TO;
uint32_t    nBlock = 2;
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF016BHandle is the handle returned
// by the DRV_SST25VF016B_Open function.

// Client registers an event handler with driver

DRV_SST25VF016B_BlockEventHandlerSet(mySST25VF016BHandle,
                APP_SST25VF016BEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF016B_BlockWrite( mySST25VF016BHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_SST25VF016BEventHandler(DRV_SST25VF016B_BLOCK_EVENT event,
        DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE:
```

```
            // This means the data was transferred.
            break;

        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function commandHandle -Pointer to an argument that will contain the return buffer handle
sourceBuffer	The source buffer containing data to be programmed into SPI Flash
blockStart	Start block address of SST25VF016B Flash where the write should begin. It can be any address of the flash.
nBlock	Total number of blocks to be written. Each write block is of 1 byte.

Function

void DRV_SST25VF016B_BlockWrite
(
DRV_HANDLE handle,
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t *sourceBuffer,
uint32_t blockStart,
uint32_t nBlock
);

d) Media Interface Functions

DRV_SST25VF016B_GeometryGet Function

Returns the geometry of the device.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_SST25VF016B_GeometryGet(DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Structure which holds the media geometry information.

Description

This API gives the following geometrical details of the SST25VF016B Flash:

- Media Property
- Number of Read/Write/Erase regions in the flash device
- Number of Blocks and their size in each region of the device

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
SYS_FS_MEDIA_GEOMETRY * sstFlashGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalFlashSize;

sstFlashGeometry = DRV_SST25VF016B_GeometryGet(sstOpenHandle1);

// read block size should be 1 byte
readBlockSize  = sstFlashGeometry->geometryTable->blockSize;
nReadBlocks = sstFlashGeometry->geometryTable->numBlocks;
nReadRegions = sstFlashGeometry->numReadRegions;

// write block size should be 1 byte
writeBlockSize  = (sstFlashGeometry->geometryTable +1)->blockSize;
// erase block size should be 4k byte
eraseBlockSize  = (sstFlashGeometry->geometryTable +2)->blockSize;

// total flash size should be 256k byte
totalFlashSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY DRV_SST25VF016B_GeometryGet(DRV_HANDLE handle);

DRV_SST25VF016B_MediaIsAttached Function

Returns the status of the media.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
bool DRV_SST25VF016B_MediaIsAttached(DRV_HANDLE handle);
```

Returns

- True - Media is attached
- False - Media is not attached

Description

This API tells if the media is attached or not.

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
if (DRV_SST25VF016B_MediaIsAttached(handle))
{
// Do Something
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_SST25VF016B_MediaIsAttached(DRV_HANDLE handle);

e) Data Types and Constants

DRV_SST25VF016B_BLOCK_COMMAND_HANDLE Type

Handle identifying block commands of the driver.

File

drv_sst25vf016b.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_SST25VF016B_BLOCK_COMMAND_HANDLE;
```

Description

SPI Flash Driver Block Command Handle

A block command handle is returned by a call to the Read, Write, or Erase functions. This handle allows the application to track the completion of the operation. The handle is returned back to the client by the "event handler callback" function registered with the driver.

The handle assigned to a client request expires when the client has been notified of the completion of the operation (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_SST25VF016B_BLOCK_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_sst25vf016b.h

C

```
typedef enum {
  DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR
} DRV_SST25VF016B_BLOCK_EVENT;
```

Members

Members	Description
DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE	Block operation has been completed successfully. Read/Write/Erase Complete
DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR	There was an error during the block operation Read/Write/Erase Error

Description

SST25VF016B SPI Flash Driver Events

This enumeration identifies the possible events that can result from a Read, Write, or Erase request caused by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_SST25VF016B_BlockEventHandlerSet function when a block request is completed.

DRV_SST25VF016B_CLIENT_STATUS Enumeration

Defines the client status.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
typedef enum {
  DRV_SST25VF016B_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0,
  DRV_SST25VF016B_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY,
  DRV_SST25VF016B_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED,
  DRV_SST25VF016B_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR
} DRV_SST25VF016B_CLIENT_STATUS;
```

Members

Members	Description
DRV_SST25VF016B_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0	Up and running, ready to start new operations
DRV_SST25VF016B_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY	Operation in progress, unable to start a new one
DRV_SST25VF016B_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED	Client is closed
DRV_SST25VF016B_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR	Client Error

Description

SPI Flash Client Status

Defines the various client status codes.

Remarks

None.

DRV_SST25VF016B_EVENT_HANDLER Type

Pointer to a SST25VF016B SPI Flash Driver Event handler function.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
typedef void (* DRV_SST25VF016B_EVENT_HANDLER)(DRV_SST25VF016B_BLOCK_EVENT event,
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t context);
```

Returns

None.

Description

SST25VF016B SPI Flash Driver Event Handler Function Pointer

This data type defines the required function signature for the SST25VF016B SPI Flash driver event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event calls back from the driver.

The parameters and return values and return value are described here and a partial example implementation is provided.

Remarks

If the event is DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE, it means that the data was transferred successfully.

If the event is DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR, it means that the data was not transferred successfully.

The context parameter contains the a handle to the client context, provided at the time the event handling function was registered using the DRV_SST25VF016B_BlockEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the read/write/erase request.

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

The Read, Write, and Erase functions can be called in the event handler to add a buffer to the driver queue. These functions can only be called to add buffers to the driver whose event handler is running.

Example

```
void APP_MyBufferEventHandler
(
    DRV_SST25VF016B_BLOCK_EVENT event,
    DRV_SST25VF016B_BLOCK_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_SST25VF016B_EVENT_BLOCK_COMMAND_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle returned from the Read/Write/Erase requests
context	Value identifying the context of the application that registered the event handling function

DRV_SST25VF016B_INIT Structure

Contains all the data necessary to initialize the SPI Flash device.

Implementation: Dynamic

File

drv_sst25vf016b.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX spiDriverModuleIndex;
  PORTS_CHANNEL holdPortChannel;
  PORTS_BIT_POS holdBitPosition;
  PORTS_CHANNEL writeProtectPortChannel;
  PORTS_BIT_POS writeProtectBitPosition;
  PORTS_CHANNEL chipSelectPortChannel;
  PORTS_BIT_POS chipSelectBitPosition;
  uint32_t queueSize;
} DRV_SST25VF016B_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX spiDriverModuleIndex;	Identifies the SPI driver to be used
PORTS_CHANNEL holdPortChannel;	HOLD pin port channel
PORTS_BIT_POS holdBitPosition;	HOLD pin port position
PORTS_CHANNEL writeProtectPortChannel;	Write protect pin port channel
PORTS_BIT_POS writeProtectBitPosition;	Write Protect Bit pin position
PORTS_CHANNEL chipSelectPortChannel;	Chip select pin port channel
PORTS_BIT_POS chipSelectBitPosition;	Chip Select Bit pin position

uint32_t queueSize;	This is the buffer queue size. This is the maximum number of requests that this instance of the driver will queue. For a static build of the driver, this is overridden by the DRV_SST25VF016B_QUEUE_SIZE macro in system_config.h

Description

SST SPI Flash Driver Initialization Data

This structure contains all of the data necessary to initialize the SPI Flash device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_SST25VF016B_Initialize function.

DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID Macro

This value defines the SPI Flash Driver Block Command Invalid handle.

File

drv_sst25vf016b.h

C

```
#define DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID
```

Description

SPI Flash Driver Block Event Invalid Handle

This value defines the SPI Flash Driver Block Command Invalid handle. It is returned by read/write/erase routines when the request could not be taken.

Remarks

None.

DRV_SST25VF016B_INDEX_0 Macro

SPI Flash driver index definitions

File

drv_sst25vf016b.h

C

```
#define DRV_SST25VF016B_INDEX_0 0
```

Description

Driver SPI Flash Module Index reference

These constants provide SST25VF016B SPI Flash driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_SST25VF016B_Initialize and DRV_SST25VF016B_Open routines to identify the driver instance in use.

DRV_SST25VF016B_INDEX_1 Macro

File

drv_sst25vf016b.h

C

```
#define DRV_SST25VF016B_INDEX_1 1
```

Description

This is macro DRV_SST25VF016B_INDEX_1.

SST25VF020B API

a) System Functions

	Name	Description
	DRV_SST25VF020B_Initialize	Initializes the SST25VF020B SPI Flash Driver instance for the specified driver index. **Implementation:** Dynamic
	DRV_SST25VF020B_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module. **Implementation:** Dynamic
	DRV_SST25VF020B_Status	Gets the current status of the SPI Flash Driver module. **Implementation:** Dynamic
	DRV_SST25VF020B_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR. **Implementation:** Dynamic

b) Core Client Functions

	Name	Description
	DRV_SST25VF020B_ClientStatus	Gets current client-specific status of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF020B_CommandStatus	Gets the current status of the command.
	DRV_SST25VF020B_Close	Closes an opened-instance of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF020B_Open	Opens the specified SPI Flash driver instance and returns a handle to it. **Implementation:** Dynamic

c) Block Operation Functions

	Name	Description
	DRV_SST25VF020B_BlockErase	Erase the specified number of blocks in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockRead	Reads blocks of data starting from the specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockWrite	Write blocks of data starting from a specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockEraseWrite	Erase and Write blocks of data starting from a specified address in SST flash memory.

d) Media Interface Functions

	Name	Description
	DRV_SST25VF020B_GeometryGet	Returns the geometry of the device. **Implementation:** Dynamic
	DRV_SST25VF020B_MediaIsAttached	Returns the status of the media. **Implementation:** Dynamic

e) Data Types and Constants

	Name	Description
	DRV_SST25VF020B_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.
	DRV_SST25VF020B_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_SST25VF020B_CLIENT_STATUS	Defines the client status.
	DRV_SST25VF020B_EVENT_HANDLER	Pointer to a SST25VF020B SPI Flash Driver Event handler function.
	DRV_SST25VF020B_INIT	Contains all the data necessary to initialize the SPI Flash device.
	DRV_SST25VF020B_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.
	DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_SST25VF020B_INDEX_0	SPI Flash driver index definitions.

	DRV_SST25VF020B_INDEX_1	This is macro DRV_SST25VF020B_INDEX_1.

Description

This section contains the SST25V020B Flash device API.

a) System Functions

DRV_SST25VF020B_Initialize Function

Initializes the SST25VF020B SPI Flash Driver instance for the specified driver index.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
SYS_MODULE_OBJ DRV_SST25VF020B_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the SPI Flash driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This function must be called before any other SPI Flash function is called.

This function should only be called once during system initialization unless DRV_SST25VF020B_Deinitialize is called to deinitialize the driver instance.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Preconditions

None.

Example

```
// This code snippet shows an example of initializing the SST25VF020B SPI
// Flash Driver. SPI driver index 0 is used for the purpose. Pin numbers 1, 2
// and 3 of port channel B are configured for hold pin, write protection pin
// and chip select pin respectively. Maximum buffer queue size is set 5.

DRV_SST25VF020B_INIT   SST25VF020BInitData;
SYS_MODULE_OBJ      objectHandle;

SST25VF020BInitData.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
SST25VF020BInitData.spiDriverModuleIndex  = DRV_SPI_INDEX_0;
SST25VF020BInitData.holdPortChannel       = PORT_CHANNEL_B;
SST25VF020BInitData.holdBitPosition       = PORTS_BIT_POS_1;
SST25VF020BInitData.writeProtectPortChannel = PORT_CHANNEL_B;
SST25VF020BInitData.writeProtectBitPosition = PORTS_BIT_POS_2;
SST25VF020BInitData.chipSelectPortChannel = PORT_CHANNEL_F;
SST25VF020BInitData.chipSelectBitPosition = PORTS_BIT_POS_2;
SST25VF020BInitData.queueSize = 5;

objectHandle = DRV_SST25VF020B_Initialize(DRV_SST25VF020B_INDEX_0,
                            (SYS_MODULE_INIT*)SST25VF020BInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing data necessary to initialize the driver.

Function

```
  SYS_MODULE_OBJ DRV_SST25VF020B_Initialize
(
const    SYS_MODULE_INDEX index,
const    SYS_MODULE_INIT * const init
);
```

DRV_SST25VF020B_Deinitialize Function

Deinitializes the specified instance of the SPI Flash driver module.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SPI Flash Driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This function will NEVER block waiting for hardware.

Preconditions

Function DRV_SST25VF020B_Initialize should have been called before calling this function.

Example

```
// This code snippet shows an example of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_SST25VF020B_Initialize
SYS_STATUS          status;


DRV_SST25VF020B_Deinitialize(object);

status = DRV_SST25VF020B_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SST25VF020B_Initialize

Function

void DRV_SST25VF020B_Deinitialize(SYS_MODULE_OBJ object)

DRV_SST25VF020B_Status Function

Gets the current status of the SPI Flash Driver module.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
SYS_STATUS DRV_SST25VF020B_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and accept requests for new operations

SYS_STATUS_UNINITIALIZED - Indicates that the driver is not initialized

Description

This function provides the current status of the SPI Flash Driver module.

Remarks

A driver can only be opened when its status is SYS_STATUS_READY.

Preconditions

Function DRV_SST25VF020B_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST25VF020B_Initialize
SYS_STATUS          SST25VF020BStatus;

SST25VF020BStatus = DRV_SST25VF020B_Status(object);
else if (SYS_STATUS_ERROR >= SST25VF020BStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SST25VF020B_Initialize

Function

SYS_STATUS DRV_SST25VF020B_Status(SYS_MODULE_OBJ object)

DRV_SST25VF020B_Tasks Function

Maintains the driver's read, erase, and write state machine and implements its ISR.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal state machine and should be called from the system's Tasks function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks function (SYS_Tasks).

Preconditions

The DRV_SST25VF020B_Initialize function must have been called for the specified SPI Flash driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST25VF020B_Initialize

while (true)
{
    DRV_SST25VF020B_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_SST25VF020B_Initialize)

Function

void DRV_SST25VF020B_Tasks (SYS_MODULE_OBJ object);

b) Core Client Functions

DRV_SST25VF020B_ClientStatus Function

Gets current client-specific status of the SPI Flash driver.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
DRV_SST25VF020B_CLIENT_STATUS DRV_SST25VF020B_ClientStatus(const DRV_HANDLE handle);
```

Returns

A DRV_SST25VF020B_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the SPI Flash driver associated with the given handle.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_SST25VF020B_Initialize function must have been called.

DRV_SST25VF020B_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      handle;         // Returned from DRV_SST25VF020B_Open
DRV_SST25VF020B_CLIENT_STATUS     clientStatus;

clientStatus = DRV_SST25VF020B_ClientStatus(handle);
if(DRV_SST25VF020B_CLIENT_STATUS_READY == clientStatus)
{
    // do the tasks
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's open

Function

DRV_SST25VF020B_CLIENT_STATUS DRV_SST25VF020B_ClientStatus(DRV_HANDLE handle);

DRV_SST25VF020B_CommandStatus Function

Gets the current status of the command.

File

drv_sst25vf020b.h

C

```
DRV_SST25VF020B_COMMAND_STATUS DRV_SST25VF020B_CommandStatus(const DRV_HANDLE handle, const
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle);
```

Returns

A DRV_SST25VF020B_COMMAND_STATUS value describing the current status of the buffer. Returns DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID if the client handle or the command handle is not valid.

Description

This routine gets the current status of the buffer. The application must use this routine where the status of a scheduled buffer needs to polled on. The function may return DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID in a case where the buffer handle has expired. A buffer handle expires when the internal buffer object is re-assigned to another erase, read or write request. It is recommended that this function be called regularly in order to track the buffer status correctly.

The application can alternatively register an event handler to receive write, read or erase operation completion events.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

Block command request must have been made using Erase, Read or Write APIs to get a valid command handle.

Example

```
DRV_HANDLE      sstOpenHandle;   // Returned from DRV_SST25VF020B_Open
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE  commandHandle;
DRV_SST25VF020B_BlockErase
                        (
                            sstOpenHandle,
                            &commandHandle,
                            0,
                            1
                        );

if(DRV_SST25VF020B_CommandStatus(sstOpenHandle, commandHandle) == DRV_SST25VF020B_COMMAND_COMPLETED );
{
    // do something
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
commandHandle	A valid command handle, returned from Read/Write/Erase APIs.

Function

DRV_SST25VF020B_COMMAND_STATUS DRV_SST25VF020B_CommandStatus

(

const DRV_HANDLE handle,
const DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle
);

DRV_SST25VF020B_Close Function

Closes an opened-instance of the SPI Flash driver.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the SPI Flash driver, invalidating the handle.

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_SST25VF020B_Open before the caller may use the driver again.

Usually, there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_SST25VF020B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF020B_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_SST25VF020B_Open

DRV_SST25VF020B_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_SST25VF020B_Close(DRV_Handle handle);

DRV_SST25VF020B_Open Function

Opens the specified SPI Flash driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
DRV_HANDLE DRV_SST25VF020B_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_SST25VF020B_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.
- if the driver status is not ready.

The driver status becomes ready inside "DRV_SST25VF020B_Tasks" function. To make the SST Driver status ready and hence successfully "Open" the driver, "Task" routine need to be called periodically.

Description

This function opens the specified SPI Flash driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The driver will always work in Non-Blocking mode even if IO-intent is selected as blocking.

The handle returned is valid until the DRV_SST25VF020B_Close function is called.

This function will NEVER block waiting for hardware.

Preconditions

Function DRV_SST25VF020B_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_SST25VF020B_Open(DRV_SST25VF020B_INDEX_0,
                                                DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_SST25VF020B_Open
(
const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT ioIntent
);

c) Block Operation Functions

DRV_SST25VF020B_BlockErase Function

Erase the specified number of blocks in Flash memory.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_BlockErase(const DRV_HANDLE handle, DRV_SST25VF020B_BLOCK_COMMAND_HANDLE *
commandHandle, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It Will be DRV_BUFFER_HANDLE_INVALID if the request was not queued.

Description

This function schedules a non-blocking erase operation in flash memory. The function returns with a valid erase handle in the commandHandle argument if the erase request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the client opened the driver for read only
- if nBlock is 0
- if the queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SST25VF020B_EVENT_ERASE_COMPLETE event if the erase operation was successful or DRV_SST25VF020B_EVENT_ERASE_ERROR event if the erase operation was not successful.

Remarks

Write Protection will be disabled for the complete flash memory region in the beginning by default.

Preconditions

The DRV_SST25VF020B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF020B_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF020B_Open call.

Example

```
// Destination address should be block aligned.
uint32_t blockStart;
uint32_t nBlock;
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF020BHandle is the handle returned
// by the DRV_SST25VF020B_Open function.

// Client registers an event handler with driver

DRV_SST25VF020B_BlockEventHandlerSet(mySST25VF020BHandle,
                APP_SST25VF020BEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF020B_BlockErase( mySST25VF020BHandle, commandHandle,
                                                    blockStart, nBlock );

if(DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer queue is processed.

void APP_SST25VF020BEventHandler(DRV_SST25VF020B_BLOCK_EVENT event,
        DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF020B_EVENT_ERASE_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF020B_EVENT_ERASE_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
blockStart	Start block address in SST25VF020B memory from where the erase should begin. LSBs (A0-A11) of block start address will be ignored to align it with Erase block size boundary.
nBlock	Total number of blocks to be erased. Each Erase block is of size 4 KByte.

Function

```
void DRV_SST25VF020B_BlockErase
(
const       DRV_HANDLE handle,
    DRV_SST25VF020B_BLOCK_COMMAND_HANDLE * commandHandle,
uint32_t blockStart,
uint32_t nBlock
);
```

DRV_SST25VF020B_BlockEventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_BlockEventHandlerSet(const DRV_HANDLE handle, const DRV_SST25VF020B_EVENT_HANDLER
eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client calls any read, write or erase function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any read/write/erase operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_SST25VF020B_Initialize function must have been called for the specified SPI FLash driver instance.

DRV_SST25VF020B_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle;

// mySST25VF020BHandle is the handle returned
// by the DRV_SST25VF020B_Open function.

// Client registers an event handler with driver. This is done once.

DRV_SST25VF020B_BlockEventHandlerSet( mySST25VF020BHandle,
```

```
                APP_SST25VF020BEventHandler, (uintptr_t)&myAppObj );

DRV_SST25VF020B_BlockRead( mySST25VF020BHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_SST25VF020BEventHandler(DRV_SST25VF020B_BLOCK_EVENT event,
        DRV_SST25VF020B_BLOCK_COMMAND_HANDLE handle, uintptr_t context)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_SST25VF020B_BlockEventHandlerSet

(

const DRV_HANDLE handle,

const DRV_SST25VF020B_EVENT_HANDLER eventHandler,

const uintptr_t context

);

DRV_SST25VF020B_BlockRead Function

Reads blocks of data starting from the specified address in Flash memory.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_BlockRead(const DRV_HANDLE handle, DRV_SST25VF020B_BLOCK_COMMAND_HANDLE *
commandHandle, uint8_t * targetBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking read operation for reading blocks of data from flash memory. The function returns with a valid handle in the commandHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the target buffer pointer is NULL
- if the client opened the driver for write only
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully of DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

The maximum read speed is 33 MHz.

Preconditions

The DRV_SST25VF020B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF020B_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF020B_Open call.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = SST25VF020B_BASE_ADDRESS_TO_READ_FROM;
uint32_t    nBlock = 2;
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF020BHandle is the handle returned
// by the DRV_SST25VF020B_Open function.

// Client registers an event handler with driver

DRV_SST25VF020B_BlockEventHandlerSet(mySST25VF020BHandle,
                APP_SST25VF020BEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF020B_BlockRead( mySST25VF020BHandle, commandHandle,
                                      &myBuffer, blockStart, nBlock );

if(DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.

void APP_SST25VF020BEventHandler(DRV_SST25VF020B_BLOCK_EVENT event,
        DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;
```

```
            case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR:

                // Error handling here.

                break;

            default:
                break;
        }
    }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
*targetBuffer	Buffer into which the data read from the SPI Flash instance will be placed
blockStart	Start block address in SST25VF020B memory from where the read should begin. It can be any address of the flash.
nBlock	Total number of blocks to be read. Each Read block is of 1 byte.

Function

void DRV_SST25VF020B_BlockRead
(
const DRV_HANDLE handle,
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t *targetBuffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SST25VF020B_BlockWrite Function

Write blocks of data starting from a specified address in Flash memory.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_BlockWrite(DRV_HANDLE handle, DRV_SST25VF020B_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t * sourceBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking write operation for writing blocks of data into flash memory. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only
- if the buffer size is 0
- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a
DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully or
DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

In the case of multi bytes write operation, byte by byte writing will happen instead of Address auto Increment writing.

Write Protection will be disabled for the complete flash memory region in the beginning by default.

Preconditions

The DRV_SST25VF020B_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF020B_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF020B_Open call.

The flash address location which has to be written, must be erased before using the API DRV_SST25VF020B_BlockErase().

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = SST25VF020B_BASE_ADDRESS_TO_WRITE_TO;
uint32_t    nBlock = 2;
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF020BHandle is the handle returned
// by the DRV_SST25VF020B_Open function.

// Client registers an event handler with driver

DRV_SST25VF020B_BlockEventHandlerSet(mySST25VF020BHandle,
                APP_SST25VF020BEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF020B_BlockWrite( mySST25VF020BHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_SST25VF020BEventHandler(DRV_SST25VF020B_BLOCK_EVENT event,
        DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function commandHandle -Pointer to an argument that will contain the return buffer handle

sourceBuffer	The source buffer containing data to be programmed into SPI Flash
blockStart	Start block address of SST25VF020B Flash where the write should begin. It can be any address of the flash.
nBlock	Total number of blocks to be written. Each write block is of 1 byte.

Function

void DRV_SST25VF020B_BlockWrite
(
DRV_HANDLE handle,
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t *sourceBuffer,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SST25VF020B_BlockEraseWrite Function

Erase and Write blocks of data starting from a specified address in SST flash memory.

File

drv_sst25vf020b.h

C

```
void DRV_SST25VF020B_BlockEraseWrite(DRV_HANDLE hClient, DRV_SST25VF020B_BLOCK_COMMAND_HANDLE *
commandHandle, uint8_t * sourceBuffer, uint32_t blockStart, uint32_t nBlock);
```

Description

This function combines the step of erasing blocks of SST Flash and then writing the data. The application can use this function if it wants to avoid having to explicitly delete a block in order to update the bytes contained in the block.

This function schedules a non-blocking operation to erase and write blocks of data into SST flash. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read only
- if the buffer size is 0
- if the queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a
DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully or
DRV_SST25VF020B_EVENT_ERASE_ERROR event if the buffer was not processed successfully.

Remarks

Refer to drv_sst25vf020b.h for usage information.

Function

void DRV_SST25VF020B_BlockEraseWrite
(
const DRV_HANDLE handle,
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE * commandHandle,
void * sourceBuffer,
uint32_t writeBlockStart,
uint32_t nWriteBlock
)

d) Media Interface Functions

DRV_SST25VF020B_GeometryGet Function

Returns the geometry of the device.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_SST25VF020B_GeometryGet(DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Structure which holds the media geometry information.

Description

This API gives the following geometrical details of the SST25VF020B Flash:

- Media Property
- Number of Read/Write/Erase regions in the flash device
- Number of Blocks and their size in each region of the device

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
SYS_FS_MEDIA_GEOMETRY * sstFlashGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalFlashSize;

sstFlashGeometry = DRV_SST25VF020B_GeometryGet(sstOpenHandle1);

// read block size should be 1 byte
readBlockSize  = sstFlashGeometry->geometryTable->blockSize;
nReadBlocks = sstFlashGeometry->geometryTable->numBlocks;
nReadRegions = sstFlashGeometry->numReadRegions;

// write block size should be 1 byte
writeBlockSize  = (sstFlashGeometry->geometryTable +1)->blockSize;
// erase block size should be 4k byte
eraseBlockSize  = (sstFlashGeometry->geometryTable +2)->blockSize;

// total flash size should be 256k byte
totalFlashSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY DRV_SST25VF020B_GeometryGet(DRV_HANDLE handle);

DRV_SST25VF020B_MediaIsAttached Function

Returns the status of the media.

Implementation: Dynamic

File

drv_sst25vf020b.h

C

```
bool DRV_SST25VF020B_MediaIsAttached(DRV_HANDLE handle);
```

Returns

- True - Media is attached
- False - Media is not attached

Description

This function determines whether or not the media is attached.

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
if (DRV_SST25VF020B_MediaIsAttached(handle))
{
// Do Something
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_SST25VF020B_MediaIsAttached(DRV_HANDLE handle);

e) Data Types and Constants

DRV_SST25VF020B_BLOCK_COMMAND_HANDLE Type

Handle identifying block commands of the driver.

File

drv_sst25vf020b.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_SST25VF020B_BLOCK_COMMAND_HANDLE;
```

Description

SPI Flash Driver Block Command Handle

A block command handle is returned by a call to the Read, Write, or Erase functions. This handle allows the application to track the completion of the operation. The handle is returned back to the client by the "event handler callback" function registered with the driver.

The handle assigned to a client request expires when the client has been notified of the completion of the operation (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_SST25VF020B_BLOCK_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_sst25vf020b.h

C

```
typedef enum {
```

```
  DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR
} DRV_SST25VF020B_BLOCK_EVENT;
```

Members

Members	Description
DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE	Block operation has been completed successfully. Read/Write/Erase Complete
DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR	There was an error during the block operation Read/Write/Erase Error

Description

SST25VF020B SPI Flash Driver Events

This enumeration identifies the possible events that can result from a Read, Write, or Erase request caused by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_SST25VF020B_BlockEventHandlerSet function when a block request is completed.

DRV_SST25VF020B_CLIENT_STATUS Enumeration

Defines the client status.

File

drv_sst25vf020b.h

C

```
typedef enum {
  DRV_SST25VF020B_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0,
  DRV_SST25VF020B_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY,
  DRV_SST25VF020B_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED,
  DRV_SST25VF020B_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR
} DRV_SST25VF020B_CLIENT_STATUS;
```

Members

Members	Description
DRV_SST25VF020B_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0	Up and running, ready to start new operations
DRV_SST25VF020B_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY	Operation in progress, unable to start a new one
DRV_SST25VF020B_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED	Client is closed
DRV_SST25VF020B_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR	Client Error

Description

SPI Flash Client Status

Defines the various client status codes.

Remarks

None.

DRV_SST25VF020B_EVENT_HANDLER Type

Pointer to a SST25VF020B SPI Flash Driver Event handler function.

File

drv_sst25vf020b.h

C

```
typedef void (* DRV_SST25VF020B_EVENT_HANDLER)(DRV_SST25VF020B_BLOCK_EVENT event,
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t context);
```

Returns

None.

Description

SST25VF020B SPI Flash Driver Event Handler Function Pointer

This data type defines the required function signature for the SST25VF020B SPI Flash driver event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event calls back from the driver.

The parameters and return values and return value are described here and a partial example implementation is provided.

Remarks

If the event is DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE, it means that the data was transferred successfully.

If the event is DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR, it means that the data was not transferred successfully.

The context parameter contains the a handle to the client context, provided at the time the event handling function was registered using the DRV_SST25VF020B_BlockEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the read/write/erase request.

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

The Read, Write, and Erase functions can be called in the event handler to add a buffer to the driver queue. These functions can only be called to add buffers to the driver whose event handler is running.

Example

```
void APP_MyBufferEventHandler
(
    DRV_SST25VF020B_BLOCK_EVENT event,
    DRV_SST25VF020B_BLOCK_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_SST25VF020B_EVENT_BLOCK_COMMAND_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle returned from the Read/Write/Erase requests
context	Value identifying the context of the application that registered the event handling function

DRV_SST25VF020B_INIT Structure

Contains all the data necessary to initialize the SPI Flash device.

File

drv_sst25vf020b.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX spiDriverModuleIndex;
  PORTS_CHANNEL holdPortChannel;
  PORTS_BIT_POS holdBitPosition;
```

```
  PORTS_CHANNEL writeProtectPortChannel;
  PORTS_BIT_POS writeProtectBitPosition;
  PORTS_CHANNEL chipSelectPortChannel;
  PORTS_BIT_POS chipSelectBitPosition;
  uint32_t queueSize;
} DRV_SST25VF020B_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX spiDriverModuleIndex;	Identifies the SPI driver to be used
PORTS_CHANNEL holdPortChannel;	HOLD pin port channel
PORTS_BIT_POS holdBitPosition;	HOLD pin port position
PORTS_CHANNEL writeProtectPortChannel;	Write protect pin port channel
PORTS_BIT_POS writeProtectBitPosition;	Write Protect Bit pin position
PORTS_CHANNEL chipSelectPortChannel;	Chip select pin port channel
PORTS_BIT_POS chipSelectBitPosition;	Chip Select Bit pin position
uint32_t queueSize;	This is the buffer queue size. This is the maximum number of requests that this instance of the driver will queue. For a static build of the driver, this is overridden by the DRV_SST25VF020B_QUEUE_SIZE macro in system_config.h

Description

SST SPI Flash Driver Initialization Data

This structure contains all of the data necessary to initialize the SPI Flash device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_SST25VF020B_Initialize function.

DRV_SST25VF020B_COMMAND_STATUS Enumeration

Specifies the status of the command for the read, write and erase operations.

File

drv_sst25vf020b.h

C

```
typedef enum {
  DRV_SST25VF020B_COMMAND_COMPLETED,
  DRV_SST25VF020B_COMMAND_QUEUED,
  DRV_SST25VF020B_COMMAND_IN_PROGRESS,
  DRV_SST25VF020B_COMMAND_ERROR_UNKNOWN
} DRV_SST25VF020B_COMMAND_STATUS;
```

Members

Members	Description
DRV_SST25VF020B_COMMAND_COMPLETED	Requested operation is completed
DRV_SST25VF020B_COMMAND_QUEUED	Scheduled but not started
DRV_SST25VF020B_COMMAND_IN_PROGRESS	Currently being in transfer
DRV_SST25VF020B_COMMAND_ERROR_UNKNOWN	Unknown Command

Description

SST Flash Driver Command Status

SST Flash Driver command Status

This type specifies the status of the command for the read, write and erase operations.

Remarks

None.

DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID Macro

This value defines the SPI Flash Driver Block Command Invalid handle.

File

drv_sst25vf020b.h

C

```
#define DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID
```

Description

SPI Flash Driver Block Event Invalid Handle

This value defines the SPI Flash Driver Block Command Invalid handle. It is returned by read/write/erase routines when the request could not be taken.

Remarks

None.

DRV_SST25VF020B_INDEX_0 Macro

SPI Flash driver index definitions.

File

drv_sst25vf020b.h

C

```
#define DRV_SST25VF020B_INDEX_0 0
```

Description

Driver SPI Flash Module Index reference

These constants provide SST25VF020B SPI Flash driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_SST25VF020B_Initialize and DRV_SST25VF020B_Open routines to identify the driver instance in use.

DRV_SST25VF020B_INDEX_1 Macro

File

drv_sst25vf020b.h

C

```
#define DRV_SST25VF020B_INDEX_1 1
```

Description

This is macro DRV_SST25VF020B_INDEX_1.

SST25VF064C API

a) System Functions

	Name	Description
	DRV_SST25VF064C_Initialize	Initializes the SST25VF064C SPI Flash Driver instance for the specified driver index.
	DRV_SST25VF064C_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module.
	DRV_SST25VF064C_Status	Gets the current status of the SPI Flash Driver module.
	DRV_SST25VF064C_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR.

b) Core Client Functions

	Name	Description
	DRV_SST25VF064C_ClientStatus	Gets current client-specific status of the SPI Flash driver.
	DRV_SST25VF064C_Close	Closes an opened-instance of the SPI Flash driver.
	DRV_SST25VF064C_CommandStatus	Gets the current status of the command.

	DRV_SST25VF064C_Open	Opens the specified SPI Flash driver instance and returns a handle to it.

c) Block Operation Functions

	Name	Description
	DRV_SST25VF064C_BlockErase	Erase the specified number of blocks in Flash memory.
	DRV_SST25VF064C_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed.
	DRV_SST25VF064C_BlockRead	Reads blocks of data starting from the specified address in Flash memory.
	DRV_SST25VF064C_BlockWrite	Write blocks of data starting from a specified address in Flash memory.

d) Media Interface Functions

	Name	Description
	DRV_SST25VF064C_GeometryGet	Returns the geometry of the device.
	DRV_SST25VF064C_MediaIsAttached	Returns the status of the media.

e) Data Types and Constants

	Name	Description
	DRV_SST25VF064C_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.
	DRV_SST25VF064C_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_SST25VF064C_CLIENT_STATUS	Defines the client status.
	DRV_SST25VF064C_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.
	DRV_SST25VF064C_EVENT_HANDLER	Pointer to a SST25VF064C SPI Flash Driver Event handler function.
	DRV_SST25VF064C_INIT	Contains all the data necessary to initialize the SPI Flash device.
	DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_SST25VF064C_INDEX_0	SPI Flash driver index definitions.
	DRV_SST25VF064C_INDEX_1	This is macro DRV_SST25VF064C_INDEX_1.

Description

a) System Functions

DRV_SST25VF064C_Initialize Function

Initializes the SST25VF064C SPI Flash Driver instance for the specified driver index.

File

drv_sst25vf064c.h

C

```
SYS_MODULE_OBJ DRV_SST25VF064C_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the SPI Flash driver instance for the specified driver index, making it ready for clients to open and use it.

Remarks

This function must be called before any other SPI Flash function is called.

This function should only be called once during system initialization unless DRV_SST25VF064C_Deinitialize is called to deinitialize the driver instance.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Preconditions

None.

Example

```
// This code snippet shows an example of initializing the SST25VF064C SPI
// Flash Driver. SPI driver index 0 is used for the purpose. Pin numbers 1, 2
// and 3 of port channel B are configured for hold pin, write protection pin
// and chip select pin respectively. Maximum buffer queue size is set 5.

DRV_SST25VF064C_INIT   SST25VF064CInitData;
SYS_MODULE_OBJ      objectHandle;

SST25VF064CInitData.moduleInit.value      = SYS_MODULE_POWER_RUN_FULL;
SST25VF064CInitData.spiDriverModuleIndex  = DRV_SPI_INDEX_0;
SST25VF064CInitData.holdPortChannel       = PORT_CHANNEL_B;
SST25VF064CInitData.holdBitPosition       = PORTS_BIT_POS_1;
SST25VF064CInitData.writeProtectPortChannel = PORT_CHANNEL_B;
SST25VF064CInitData.writeProtectBitPosition = PORTS_BIT_POS_2;
SST25VF064CInitData.chipSelectPortChannel = PORT_CHANNEL_F;
SST25VF064CInitData.chipSelectBitPosition = PORTS_BIT_POS_2;
SST25VF064CInitData.queueSize = 5;

objectHandle = DRV_SST25VF064C_Initialize(DRV_SST25VF064C_INDEX_0,
                                (SYS_MODULE_INIT*)SST25VF064CInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing data necessary to initialize the driver.

Function

```
SYS_MODULE_OBJ DRV_SST25VF064C_Initialize
(
const    SYS_MODULE_INDEX index,
const    SYS_MODULE_INIT * const init
);
```

DRV_SST25VF064C_Deinitialize Function

Deinitializes the specified instance of the SPI Flash driver module.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the SPI Flash Driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This function will NEVER block waiting for hardware.

Preconditions

Function DRV_SST25VF064C_Initialize should have been called before calling this function.

Example

```
// This code snippet shows an example of deinitializing the driver.

SYS_MODULE_OBJ      object;     //  Returned from DRV_SST25VF064C_Initialize
SYS_STATUS          status;


DRV_SST25VF064C_Deinitialize(object);

status = DRV_SST25VF064C_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SST25VF064C_Initialize

Function

void DRV_SST25VF064C_Deinitialize(SYS_MODULE_OBJ object)

DRV_SST25VF064C_Status Function

Gets the current status of the SPI Flash Driver module.

File

drv_sst25vf064c.h

C

```
SYS_STATUS DRV_SST25VF064C_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is ready and accept requests for new operations

SYS_STATUS_UNINITIALIZED - Indicates that the driver is not initialized

Description

This function provides the current status of the SPI Flash Driver module.

Remarks

A driver can only be opened when its status is SYS_STATUS_READY.

Preconditions

Function DRV_SST25VF064C_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST25VF064C_Initialize
SYS_STATUS          SST25VF064CStatus;

SST25VF064CStatus = DRV_SST25VF064C_Status(object);
else if (SYS_STATUS_ERROR >= SST25VF064CStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_SST25VF064C_Initialize

Function

SYS_STATUS DRV_SST25VF064C_Status(SYS_MODULE_OBJ object)

DRV_SST25VF064C_Tasks Function

Maintains the driver's read, erase, and write state machine and implements its ISR.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal state machine and should be called from the system's Tasks function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks function (SYS_Tasks).

Preconditions

The DRV_SST25VF064C_Initialize function must have been called for the specified SPI Flash driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_SST25VF064C_Initialize

while (true)
{
    DRV_SST25VF064C_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_SST25VF064C_Initialize)

Function

void DRV_SST25VF064C_Tasks (SYS_MODULE_OBJ object);

b) Core Client Functions

DRV_SST25VF064C_ClientStatus Function

Gets current client-specific status of the SPI Flash driver.

File

drv_sst25vf064c.h

C

```
DRV_SST25VF064C_CLIENT_STATUS DRV_SST25VF064C_ClientStatus(const DRV_HANDLE handle);
```

Returns

A DRV_SST25VF064C_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the SPI Flash driver associated with the given handle.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

The DRV_SST25VF064C_Initialize function must have been called.

DRV_SST25VF064C_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      handle;         // Returned from DRV_SST25VF064C_Open
DRV_SST25VF064C_CLIENT_STATUS     clientStatus;

clientStatus = DRV_SST25VF064C_ClientStatus(handle);
if(DRV_SST25VF064C_CLIENT_STATUS_READY == clientStatus)
{
    // do the tasks
}
```

Parameters

Parameters	Description
handle	A valid open instance handle, returned from the driver's open

Function

DRV_SST25VF064C_CLIENT_STATUS DRV_SST25VF064C_ClientStatus(DRV_HANDLE handle);

DRV_SST25VF064C_Close Function

Closes an opened-instance of the SPI Flash driver.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the SPI Flash driver, invalidating the handle.

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_SST25VF064C_Open before the caller may use the driver again.

Usually, there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_SST25VF064C_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF064C_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_SST25VF064C_Open

DRV_SST25VF064C_Close(handle);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

void DRV_SST25VF064C_Close(DRV_Handle handle);

DRV_SST25VF064C_CommandStatus Function

Gets the current status of the command.

File

drv_sst25vf064c.h

C

```
DRV_SST25VF064C_COMMAND_STATUS DRV_SST25VF064C_CommandStatus(const DRV_HANDLE handle, const
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle);
```

Returns

A DRV_SST25VF064C_COMMAND_STATUS value describing the current status of the buffer. Returns DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID if the client handle or the command handle is not valid.

Description

This routine gets the current status of the buffer. The application must use this routine where the status of a scheduled buffer needs to polled on. The function may return DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID in a case where the buffer handle has expired. A buffer handle expires when the internal buffer object is re-assigned to another erase, read or write request. It is recommended that this function be called regularly in order to track the buffer status correctly.

The application can alternatively register an event handler to receive write, read or erase operation completion events.

Remarks

This function will not block for hardware access and will immediately return the current status.

Preconditions

Block command request must have been made using Erase, Read or Write APIs to get a valid command handle.

Example

```
DRV_HANDLE      sstOpenHandle;   // Returned from DRV_SST25VF064C_Open
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE  commandHandle;
DRV_SST25VF064C_BlockErase
                        (
                            sstOpenHandle,
                            &commandHandle,
                            0,
                            1
                        );

if(DRV_SST25VF064C_CommandStatus(sstOpenHandle, commandHandle) == DRV_SST25VF064C_COMMAND_COMPLETED );
{
    // do something
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
commandHandle	A valid command handle, returned from Read/Write/Erase APIs.

Function

DRV_SST25VF064C_COMMAND_STATUS DRV_SST25VF064C_CommandStatus

(

const DRV_HANDLE handle,

const DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle

);

DRV_SST25VF064C_Open Function

Opens the specified SPI Flash driver instance and returns a handle to it.

File

drv_sst25vf064c.h

C

```
DRV_HANDLE DRV_SST25VF064C_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the function returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Errors can occur under the following circumstances:

- if the number of client objects allocated via DRV_SST25VF064C_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.

Description

This function opens the specified SPI Flash driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The driver will always work in Non-Blocking mode even if IO-intent is selected as blocking.

The handle returned is valid until the DRV_SST25VF064C_Close function is called.

This function will NEVER block waiting for hardware.

Preconditions

Function DRV_SST25VF064C_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_SST25VF064C_Open(DRV_SST25VF064C_INDEX_0,
                                                DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Identifier for the object instance to be opened
ioIntent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_SST25VF064C_Open

(

const SYS_MODULE_INDEX drvIndex,

const DRV_IO_INTENT ioIntent

);

c) Block Operation Functions

DRV_SST25VF064C_BlockErase Function

Erase the specified number of blocks in Flash memory.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_BlockErase(const DRV_HANDLE handle, DRV_SST25VF064C_BLOCK_COMMAND_HANDLE *
commandHandle, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It Will be DRV_BUFFER_HANDLE_INVALID if the request was not queued.

Description

This function schedules a non-blocking erase operation in Flash memory. The function returns with a valid erase handle in the commandHandle argument if the erase request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if the client opened the driver for read only
- if nBlock is 0
- if the queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_SST25VF064C_EVENT_ERASE_COMPLETE event if the erase operation was successful or DRV_SST25VF064C_EVENT_ERASE_ERROR event if the erase operation was not successful.

Remarks

Write Protection will be disabled for the complete Flash memory region in the beginning by default.

Preconditions

The DRV_SST25VF064C_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF064C_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF064C_Open call.

Example

```
// Destination address should be block aligned.
uint32_t blockStart;
uint32_t nBlock;
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF064CHandle is the handle returned
// by the DRV_SST25VF064C_Open function.

// Client registers an event handler with driver

DRV_SST25VF064C_BlockEventHandlerSet(mySST25VF064CHandle,
                APP_SST25VF064CEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF064C_BlockErase( mySST25VF064CHandle, commandHandle,
                                                    blockStart, nBlock );

if(DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer queue is processed.

void APP_SST25VF064CEventHandler(DRV_SST25VF064C_BLOCK_EVENT event,
        DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF064C_EVENT_ERASE_COMPLETE:
```

```
            // This means the data was transferred.
            break;

        case DRV_SST25VF064C_EVENT_ERASE_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
blockStart	Start block address in SST25VF064C memory from where the erase should begin. LSBs (A0-A11) of block start address will be ignored to align it with Erase block size boundary.
nBlock	Total number of blocks to be erased. Each Erase block is of size 4 KByte.

Function

void DRV_SST25VF064C_BlockErase
(
const DRV_HANDLE handle,
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE * commandHandle,
uint32_t blockStart,
uint32_t nBlock
);

DRV_SST25VF064C_BlockEventHandlerSet Function

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_BlockEventHandlerSet(const DRV_HANDLE handle, const DRV_SST25VF064C_EVENT_HANDLER
eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the driver to call back when queued operation has completed. When a client calls any read, write or erase function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the queued operation has completed.

The event handler should be set before the client performs any read/write/erase operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued operation has completed, it does not need to register a callback.

Preconditions

The DRV_SST25VF064C_Initialize function must have been called for the specified SPI FLash driver instance.

DRV_SST25VF064C_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;
```

```
uint8_t myBuffer[MY_BUFFER_SIZE];
uint32_t blockStart, nBlock;
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle;

// mySST25VF064CHandle is the handle returned
// by the DRV_SST25VF064C_Open function.

// Client registers an event handler with driver. This is done once.

DRV_SST25VF064C_BlockEventHandlerSet( mySST25VF064CHandle,
                APP_SST25VF064CEventHandler, (uintptr_t)&myAppObj );

DRV_SST25VF064C_BlockRead( mySST25VF064CHandle, commandHandle,
                                      &myBuffer, blockStart, nBlock );

if(DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when operation is done.

void APP_SST25VF064CEventHandler(DRV_SST25VF064C_BLOCK_EVENT event,
        DRV_SST25VF064C_BLOCK_COMMAND_HANDLE handle, uintptr_t context)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_SST25VF064C_BlockEventHandlerSet

(

const DRV_HANDLE handle,

const DRV_SST25VF064C_EVENT_HANDLER eventHandler,

const uintptr_t context

);

DRV_SST25VF064C_BlockRead Function

Reads blocks of data starting from the specified address in Flash memory.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_BlockRead(const DRV_HANDLE handle, DRV_SST25VF064C_BLOCK_COMMAND_HANDLE *
commandHandle, uint8_t * targetBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking read operation for reading blocks of data from Flash memory. The function returns with a valid handle in the commandHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the target buffer pointer is NULL
- if the client opened the driver for write only
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a
DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully of
DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

The maximum read speed is 33 MHz.

Preconditions

The DRV_SST25VF064C_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF064C_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF064C_Open call.

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = SST25VF064C_BASE_ADDRESS_TO_READ_FROM;
uint32_t    nBlock = 2;
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF064CHandle is the handle returned
// by the DRV_SST25VF064C_Open function.

// Client registers an event handler with driver

DRV_SST25VF064C_BlockEventHandlerSet(mySST25VF064CHandle,
                APP_SST25VF064CEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF064C_BlockRead( mySST25VF064CHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when the buffer is processed.
```

```
void APP_SST25VF064CEventHandler(DRV_SST25VF064C_BLOCK_EVENT event,
        DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function
commandHandle	Pointer to an argument that will contain the return buffer handle
*targetBuffer	Buffer into which the data read from the SPI Flash instance will be placed
blockStart	Start block address in SST25VF064C memory from where the read should begin. It can be any address of the Flash.
nBlock	Total number of blocks to be read. Each Read block is of 1 byte.

Function

void DRV_SST25VF064C_BlockRead

(

const DRV_HANDLE handle,

DRV_SST25VF064C_BLOCK_COMMAND_HANDLE * commandHandle,

uint8_t *targetBuffer,

uint32_t blockStart,

uint32_t nBlock

);

DRV_SST25VF064C_BlockWrite Function

Write blocks of data starting from a specified address in Flash memory.

File

drv_sst25vf064c.h

C

```
void DRV_SST25VF064C_BlockWrite(DRV_HANDLE handle, DRV_SST25VF064C_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t * sourceBuffer, uint32_t blockStart, uint32_t nBlock);
```

Returns

The buffer handle is returned in the commandHandle argument. It will be DRV_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking write operation for writing blocks of data into Flash memory. The function returns with a valid buffer handle in the commandHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID in the commandHandle argument under the following circumstances:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL

- if the client opened the driver for read only
- if the buffer size is 0
- if the write queue size is full or queue depth is insufficient
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a
DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE event if the buffer was processed successfully or
DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR event if the buffer was not processed successfully.

Remarks

In the case of multi bytes write operation, byte by byte writing will happen instead of Address auto Increment writing.

Write Protection will be disabled for the complete Flash memory region in the beginning by default.

Preconditions

The DRV_SST25VF064C_Initialize function must have been called for the specified SPI Flash driver instance.

DRV_SST25VF064C_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_SST25VF064C_Open call.

The Flash address location which has to be written, must be erased before using the API DRV_SST25VF064C_BlockErase().

Example

```
uint8_t myBuffer[MY_BUFFER_SIZE];

// address should be block aligned.
uint32_t blockStart = SST25VF064C_BASE_ADDRESS_TO_WRITE_TO;
uint32_t    nBlock = 2;
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle;
MY_APP_OBJ myAppObj;

// mySST25VF064CHandle is the handle returned
// by the DRV_SST25VF064C_Open function.

// Client registers an event handler with driver

DRV_SST25VF064C_BlockEventHandlerSet(mySST25VF064CHandle,
                APP_SST25VF064CEventHandler, (uintptr_t)&myAppObj);

DRV_SST25VF064C_BlockWrite( mySST25VF064CHandle, commandHandle,
                                        &myBuffer, blockStart, nBlock );

if(DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID == commandHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_SST25VF064CEventHandler(DRV_SST25VF064C_BLOCK_EVENT event,
        DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR:

            // Error handling here.

            break;

        default:
            break;
```

```
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function commandHandle -Pointer to an argument that will contain the return buffer handle
sourceBuffer	The source buffer containing data to be programmed into SPI Flash
blockStart	Start block address of SST25VF064C Flash where the write should begin. It can be any address of the Flash.
nBlock	Total number of blocks to be written. Each write block is of 1 byte.

Function

```
void DRV_SST25VF064C_BlockWrite
(
    DRV_HANDLE handle,
    DRV_SST25VF064C_BLOCK_COMMAND_HANDLE * commandHandle,
uint8_t *sourceBuffer,
uint32_t blockStart,
uint32_t nBlock
);
```

d) Media Interface Functions

DRV_SST25VF064C_GeometryGet Function

Returns the geometry of the device.

File

drv_sst25vf064c.h

C

```
SYS_FS_MEDIA_GEOMETRY * DRV_SST25VF064C_GeometryGet(DRV_HANDLE handle);
```

Returns

SYS_FS_MEDIA_GEOMETRY - Structure which holds the media geometry information.

Description

This API gives the following geometrical details of the SST25VF064C Flash:

- Media Property
- Number of Read/Write/Erase regions in the Flash device
- Number of Blocks and their size in each region of the device

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
SYS_FS_MEDIA_GEOMETRY * sstFlashGeometry;
uint32_t readBlockSize, writeBlockSize, eraseBlockSize;
uint32_t nReadBlocks, nReadRegions, totalFlashSize;

sstFlashGeometry = DRV_SST25VF064C_GeometryGet(sstOpenHandle1);

// read block size should be 1 byte
readBlockSize  = sstFlashGeometry->geometryTable->blockSize;
nReadBlocks = sstFlashGeometry->geometryTable->numBlocks;
nReadRegions = sstFlashGeometry->numReadRegions;
```

```
// write block size should be 1 byte
writeBlockSize  = (sstFlashGeometry->geometryTable +1)->blockSize;
// erase block size should be 4k byte
eraseBlockSize  = (sstFlashGeometry->geometryTable +2)->blockSize;

// total Flash size should be 8 MB
totalFlashSize = readBlockSize * nReadBlocks * nReadRegions;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

SYS_FS_MEDIA_GEOMETRY DRV_SST25VF064C_GeometryGet(DRV_HANDLE handle);

DRV_SST25VF064C_MediaIsAttached Function

Returns the status of the media.

File

drv_sst25vf064c.h

C

```
bool DRV_SST25VF064C_MediaIsAttached(DRV_HANDLE handle);
```

Returns

- True - Media is attached
- False - Media is not attached

Description

This function determines whether or not the media is attached.

Remarks

This function is typically used by File System Media Manager.

Preconditions

None.

Example

```
if (DRV_SST25VF064C_MediaIsAttached(handle))
{
// Do Something
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open function

Function

bool DRV_SST25VF064C_MediaIsAttached(DRV_HANDLE handle);

e) Data Types and Constants

DRV_SST25VF064C_BLOCK_COMMAND_HANDLE Type

Handle identifying block commands of the driver.

File

drv_sst25vf064c.h

C

```
typedef SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE DRV_SST25VF064C_BLOCK_COMMAND_HANDLE;
```

Description

SPI Flash Driver Block Command Handle

A block command handle is returned by a call to the Read, Write, or Erase functions. This handle allows the application to track the completion of the operation. The handle is returned back to the client by the "event handler callback" function registered with the driver.

The handle assigned to a client request expires when the client has been notified of the completion of the operation (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_SST25VF064C_BLOCK_EVENT Enumeration

Identifies the possible events that can result from a request.

File

drv_sst25vf064c.h

C

```
typedef enum {
  DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE,
  DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR
} DRV_SST25VF064C_BLOCK_EVENT;
```

Members

Members	Description
DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE	Block operation has been completed successfully. Read/Write/Erase Complete
DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR	There was an error during the block operation Read/Write/Erase Error

Description

SST25VF064C SPI Flash Driver Events

This enumeration identifies the possible events that can result from a Read, Write, or Erase request caused by the client.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the driver by calling the DRV_SST25VF064C_BlockEventHandlerSet function when a block request is completed.

DRV_SST25VF064C_CLIENT_STATUS Enumeration

Defines the client status.

File

drv_sst25vf064c.h

C

```
typedef enum {
  DRV_SST25VF064C_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0,
  DRV_SST25VF064C_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY,
  DRV_SST25VF064C_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED,
  DRV_SST25VF064C_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR
} DRV_SST25VF064C_CLIENT_STATUS;
```

Members

Members	Description
DRV_SST25VF064C_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY+0	Up and running, ready to start new operations
DRV_SST25VF064C_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY	Operation in progress, unable to start a new one
DRV_SST25VF064C_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED	Client is closed

DRV_SST25VF064C_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR	Client Error

Description

SPI Flash Client Status

Defines the various client status codes.

Remarks

None.

DRV_SST25VF064C_COMMAND_STATUS Enumeration

Specifies the status of the command for the read, write and erase operations.

File

drv_sst25vf064c.h

C

```
typedef enum {
  DRV_SST25VF064C_COMMAND_COMPLETED,
  DRV_SST25VF064C_COMMAND_QUEUED,
  DRV_SST25VF064C_COMMAND_IN_PROGRESS,
  DRV_SST25VF064C_COMMAND_ERROR_UNKNOWN
} DRV_SST25VF064C_COMMAND_STATUS;
```

Members

Members	Description
DRV_SST25VF064C_COMMAND_COMPLETED	Requested operation is completed
DRV_SST25VF064C_COMMAND_QUEUED	Scheduled but not started
DRV_SST25VF064C_COMMAND_IN_PROGRESS	Currently being in transfer
DRV_SST25VF064C_COMMAND_ERROR_UNKNOWN	Unknown Command

Description

SST Flash Driver Command Status

SST Flash Driver command Status. This type specifies the status of the command for the read, write and erase operations.

Remarks

None.

DRV_SST25VF064C_EVENT_HANDLER Type

Pointer to a SST25VF064C SPI Flash Driver Event handler function.

File

drv_sst25vf064c.h

C

```
typedef void (* DRV_SST25VF064C_EVENT_HANDLER)(DRV_SST25VF064C_BLOCK_EVENT event,
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t context);
```

Returns

None.

Description

SST25VF064C SPI Flash Driver Event Handler Function Pointer

This data type defines the required function signature for the SST25VF064C SPI Flash driver event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event calls back from the driver.

The parameters and return values and return value are described here and a partial example implementation is provided.

Remarks

If the event is DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE, it means that the data was transferred successfully.

If the event is DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR, it means that the data was not transferred successfully.

The context parameter contains the a handle to the client context, provided at the time the event handling function was registered using the DRV_SST25VF064C_BlockEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the read/write/erase request.

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

The Read, Write, and Erase functions can be called in the event handler to add a buffer to the driver queue. These functions can only be called to add buffers to the driver whose event handler is running.

Example

```
void APP_MyBufferEventHandler
(
    DRV_SST25VF064C_BLOCK_EVENT event,
    DRV_SST25VF064C_BLOCK_COMMAND_HANDLE commandHandle,
    uintptr_t context
)
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_SST25VF064C_EVENT_BLOCK_COMMAND_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
commandHandle	Handle returned from the Read/Write/Erase requests
context	Value identifying the context of the application that registered the event handling function

DRV_SST25VF064C_INIT Structure

Contains all the data necessary to initialize the SPI Flash device.

File

drv_sst25vf064c.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX spiDriverModuleIndex;
  PORTS_CHANNEL holdPortChannel;
  PORTS_BIT_POS holdBitPosition;
  PORTS_CHANNEL writeProtectPortChannel;
  PORTS_BIT_POS writeProtectBitPosition;
  PORTS_CHANNEL chipSelectPortChannel;
  PORTS_BIT_POS chipSelectBitPosition;
  uint32_t queueSize;
} DRV_SST25VF064C_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX spiDriverModuleIndex;	Identifies the SPI driver to be used

PORTS_CHANNEL holdPortChannel;	HOLD pin port channel
PORTS_BIT_POS holdBitPosition;	HOLD pin port position
PORTS_CHANNEL writeProtectPortChannel;	Write protect pin port channel
PORTS_BIT_POS writeProtectBitPosition;	Write Protect Bit pin position
PORTS_CHANNEL chipSelectPortChannel;	Chip select pin port channel
PORTS_BIT_POS chipSelectBitPosition;	Chip Select Bit pin position
uint32_t queueSize;	This is the buffer queue size. This is the maximum number of requests that this instance of the driver will queue. For a static build of the driver, this is overridden by the DRV_SST25VF064C_QUEUE_SIZE macro in system_config.h

Description

SST SPI Flash Driver Initialization Data

This structure contains all of the data necessary to initialize the SPI Flash device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the DRV_SST25VF064C_Initialize function.

DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID Macro

This value defines the SPI Flash Driver Block Command Invalid handle.

File

drv_sst25vf064c.h

C

```
#define DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID
```

Description

SPI Flash Driver Block Event Invalid Handle

This value defines the SPI Flash Driver Block Command Invalid handle. It is returned by read/write/erase routines when the request could not be taken.

Remarks

None.

DRV_SST25VF064C_INDEX_0 Macro

SPI Flash driver index definitions.

File

drv_sst25vf064c.h

C

```
#define DRV_SST25VF064C_INDEX_0 0
```

Description

Driver SPI Flash Module Index reference

These constants provide SST25VF064C SPI Flash driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_SST25VF064C_Initialize and DRV_SST25VF064C_Open routines to identify the driver instance in use.

DRV_SST25VF064C_INDEX_1 Macro

File

drv_sst25vf064c.h

C

```
#define DRV_SST25VF064C_INDEX_1 1
```

Description

This is macro DRV_SST25VF064C_INDEX_1.

Files

Files

Name	Description
drv_sst25vf016b.h	SPI Flash Driver Interface Definition
drv_sst25vf016b_config_template.h	SST25VF016B Driver Configuration Template.
drv_sst25vf020b.h	SPI Flash Driver Interface Definition
drv_sst25vf020b_config_template.h	SST25VF020B Driver Configuration Template.
drv_sst25vf064c.h	SPI Flash Driver Interface Definition
drv_sst25vf064c_config_template.h	SST25VF064C Driver Configuration Template.

Description

This section lists the source and header files used by the SPI Flash Driver Library.

drv_sst25vf016b.h

SPI Flash Driver Interface Definition

Enumerations

	Name	Description
	DRV_SST25VF016B_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_SST25VF016B_CLIENT_STATUS	Defines the client status. **Implementation:** Dynamic

Functions

	Name	Description
	DRV_SST25VF016B_BlockErase	Erase the specified number of blocks in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF016B_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Dynamic
	DRV_SST25VF016B_BlockRead	Reads blocks of data starting from the specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF016B_BlockWrite	Write blocks of data starting from a specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF016B_ClientStatus	Gets current client-specific status of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF016B_Close	Closes an opened-instance of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF016B_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module. **Implementation:** Dynamic
	DRV_SST25VF016B_GeometryGet	Returns the geometry of the device. **Implementation:** Dynamic
	DRV_SST25VF016B_Initialize	Initializes the SST25VF016B SPI Flash Driver instance for the specified driver index. **Implementation:** Dynamic
	DRV_SST25VF016B_MediaIsAttached	Returns the status of the media. **Implementation:** Dynamic
	DRV_SST25VF016B_Open	Opens the specified SPI Flash driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_SST25VF016B_Status	Gets the current status of the SPI Flash Driver module. **Implementation:** Dynamic

	DRV_SST25VF016B_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_SST25VF016B_INDEX_0	SPI Flash driver index definitions
	DRV_SST25VF016B_INDEX_1	This is macro DRV_SST25VF016B_INDEX_1.

Structures

	Name	Description
	DRV_SST25VF016B_INIT	Contains all the data necessary to initialize the SPI Flash device. **Implementation:** Dynamic

Types

	Name	Description
	DRV_SST25VF016B_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.
	DRV_SST25VF016B_EVENT_HANDLER	Pointer to a SST25VF016B SPI Flash Driver Event handler function. **Implementation:** Dynamic

Description

SPI Flash Driver Interface Definition

The SPI Flash device driver provides a simple interface to manage the SPI Flash modules which are external to Microchip Controllers. This file defines the interface definition for the SPI Flash Driver.

File Name

drv_sst25vf016b.h

Company

Microchip Technology Inc.

drv_sst25vf016b_config_template.h

SST25VF016B Driver Configuration Template.

Macros

	Name	Description
	DRV_SST25VF016B_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_SST25VF016B_HARDWARE_HOLD_ENABLE	Specifies if the hardware hold feature is enabled or not.
	DRV_SST25VF016B_HARDWARE_WRITE_PROTECTION_ENABLE	Specifies if the hardware write protect feature is enabled or not.
	DRV_SST25VF016B_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported
	DRV_SST25VF016B_MODE	Determines whether the driver is implemented as static or dynamic
	DRV_SST25VF016B_QUEUE_DEPTH_COMBINED	Number of entries of queues in all instances of the driver.

Description

SST25VF016B Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_sst25vf016b_config_template.h

Company

Microchip Technology Inc.

drv_sst25vf020b.h

SPI Flash Driver Interface Definition

Enumerations

	Name	Description
	DRV_SST25VF020B_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_SST25VF020B_CLIENT_STATUS	Defines the client status.
	DRV_SST25VF020B_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.

Functions

	Name	Description
	DRV_SST25VF020B_BlockErase	Erase the specified number of blocks in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockEraseWrite	Erase and Write blocks of data starting from a specified address in SST flash memory.
	DRV_SST25VF020B_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockRead	Reads blocks of data starting from the specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF020B_BlockWrite	Write blocks of data starting from a specified address in Flash memory. **Implementation:** Dynamic
	DRV_SST25VF020B_ClientStatus	Gets current client-specific status of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF020B_Close	Closes an opened-instance of the SPI Flash driver. **Implementation:** Dynamic
	DRV_SST25VF020B_CommandStatus	Gets the current status of the command.
	DRV_SST25VF020B_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module. **Implementation:** Dynamic
	DRV_SST25VF020B_GeometryGet	Returns the geometry of the device. **Implementation:** Dynamic
	DRV_SST25VF020B_Initialize	Initializes the SST25VF020B SPI Flash Driver instance for the specified driver index. **Implementation:** Dynamic
	DRV_SST25VF020B_MediaIsAttached	Returns the status of the media. **Implementation:** Dynamic
	DRV_SST25VF020B_Open	Opens the specified SPI Flash driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_SST25VF020B_Status	Gets the current status of the SPI Flash Driver module. **Implementation:** Dynamic
	DRV_SST25VF020B_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_SST25VF020B_INDEX_0	SPI Flash driver index definitions.
	DRV_SST25VF020B_INDEX_1	This is macro DRV_SST25VF020B_INDEX_1.

Structures

	Name	Description
	DRV_SST25VF020B_INIT	Contains all the data necessary to initialize the SPI Flash device.

Types

	Name	Description
	DRV_SST25VF020B_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.

	DRV_SST25VF020B_EVENT_HANDLER	Pointer to a SST25VF020B SPI Flash Driver Event handler function.

Description

SPI Flash Driver Interface Definition

The SPI Flash device driver provides a simple interface to manage the SPI Flash modules which are external to Microchip Controllers. This file defines the interface definition for the SPI Flash Driver.

File Name

drv_sst25vf020b.h

Company

Microchip Technology Inc.

drv_sst25vf020b_config_template.h

SST25VF020B Driver Configuration Template.

Macros

	Name	Description
	DRV_SST25VF020B_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_SST25VF020B_HARDWARE_HOLD_ENABLE	Specifies if the hardware hold feature is enabled or not.
	DRV_SST25VF020B_HARDWARE_WRITE_PROTECTION_ENABLE	Specifies if the hardware write protect feature is enabled or not.
	DRV_SST25VF020B_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_SST25VF020B_MODE	Determines whether the driver is implemented as static or dynamic.
	DRV_SST25VF020B_QUEUE_DEPTH_COMBINED	Number of entries of queues in all instances of the driver.

Description

SST25VF020B Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_sst25vf020b_config_template.h

Company

Microchip Technology Inc.

drv_sst25vf064c.h

SPI Flash Driver Interface Definition

Enumerations

	Name	Description
	DRV_SST25VF064C_BLOCK_EVENT	Identifies the possible events that can result from a request.
	DRV_SST25VF064C_CLIENT_STATUS	Defines the client status.
	DRV_SST25VF064C_COMMAND_STATUS	Specifies the status of the command for the read, write and erase operations.

Functions

	Name	Description
	DRV_SST25VF064C_BlockErase	Erase the specified number of blocks in Flash memory.
	DRV_SST25VF064C_BlockEventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed.
	DRV_SST25VF064C_BlockRead	Reads blocks of data starting from the specified address in Flash memory.
	DRV_SST25VF064C_BlockWrite	Write blocks of data starting from a specified address in Flash memory.
	DRV_SST25VF064C_ClientStatus	Gets current client-specific status of the SPI Flash driver.
	DRV_SST25VF064C_Close	Closes an opened-instance of the SPI Flash driver.

	DRV_SST25VF064C_CommandStatus	Gets the current status of the command.
	DRV_SST25VF064C_Deinitialize	Deinitializes the specified instance of the SPI Flash driver module.
	DRV_SST25VF064C_GeometryGet	Returns the geometry of the device.
	DRV_SST25VF064C_Initialize	Initializes the SST25VF064C SPI Flash Driver instance for the specified driver index.
	DRV_SST25VF064C_MediaIsAttached	Returns the status of the media.
	DRV_SST25VF064C_Open	Opens the specified SPI Flash driver instance and returns a handle to it.
	DRV_SST25VF064C_Status	Gets the current status of the SPI Flash Driver module.
	DRV_SST25VF064C_Tasks	Maintains the driver's read, erase, and write state machine and implements its ISR.

Macros

	Name	Description
	DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID	This value defines the SPI Flash Driver Block Command Invalid handle.
	DRV_SST25VF064C_INDEX_0	SPI Flash driver index definitions.
	DRV_SST25VF064C_INDEX_1	This is macro DRV_SST25VF064C_INDEX_1.

Structures

	Name	Description
	DRV_SST25VF064C_INIT	Contains all the data necessary to initialize the SPI Flash device.

Types

	Name	Description
	DRV_SST25VF064C_BLOCK_COMMAND_HANDLE	Handle identifying block commands of the driver.
	DRV_SST25VF064C_EVENT_HANDLER	Pointer to a SST25VF064C SPI Flash Driver Event handler function.

Description

SPI Flash Driver Interface Definition

The SPI Flash device driver provides a simple interface to manage the SPI Flash modules which are external to Microchip Controllers. This file defines the interface definition for the SPI Flash Driver.

File Name

drv_sst25vf064c.h

Company

Microchip Technology Inc.

drv_sst25vf064c_config_template.h

SST25VF064C Driver Configuration Template.

Macros

	Name	Description
	DRV_SST25VF064C_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_SST25VF064C_HARDWARE_HOLD_ENABLE	Specifies whether or not the hardware hold feature is enabled.
	DRV_SST25VF064C_HARDWARE_WRITE_PROTECTION_ENABLE	Specifies whether or not the hardware write protect feature is enabled.
	DRV_SST25VF064C_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_SST25VF064C_MODE	Determines whether the driver is implemented as static or dynamic.
	DRV_SST25VF064C_QUEUE_DEPTH_COMBINED	Number of entries of queues in all instances of the driver.

Description

SST25VF064C Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_sst25vf064c_config_template.h

Company

Microchip Technology Inc.

Timer Driver Library

This section describes the Timer Driver Library.

Introduction

This library provides an interface to manage the Timer module on the Microchip family of microcontrollers during different modes of operation.

Description

Timers are useful for generating accurate time based periodic interrupts for software application or real time operating systems. Other uses include counting external pulses or accurate timing measurement of external events using the timer's gate functions and accurate hardware delays.

Note: Not all features are available on all devices. Please refer to the specific device data sheet to determine availability.

Using the Library

This topic describes the basic architecture of the Timer Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_tmr.h`

The interface to the Timer Driver Library is defined in the `drv_tmr.h` header file.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

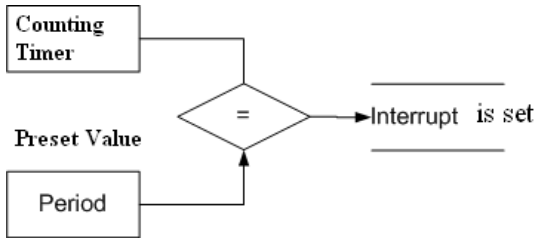
Abstraction Model

The Timer Driver abstracts the hardware by providing the capability to register callback functions to the application.

Description

Abstraction Model

The abstraction model of the Timer Driver is explained in the following diagram:



The core functionality of the Timer allows access to both the counter and the period values.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Timer Driver Library.

Library Interface Section	Description
Configuration	Provides macros for configuring the system. It is required that the system configures the driver to build correctly by choosing appropriate configuration options as listed in this section. These macros enable different features or modes of the timer peripheral.
System Interaction Functions	Provides interfaces to system layer to initialize, deinitialize and reinitialize the module. This section also describes functions to query the status of the module.
Core Functions	Provides interfaces for core functionality of the driver.
Alarm Functions	Provides interfaces to handle alarm features, if alarm functionality is enabled.
Period Functions	Provides interfaces to control the periodicity of the timers.
Counter Control Functions	Provides interfaces to update the counter values.
Miscellaneous Functions	Provides interfaces to get the version information, timer tick and operating frequencies.

How the Library Works

The library provides interfaces to support:

- System Interaction
- Sync Mode Setup
- Period Modification
- Counter Modification
- Client Core Functionality
- Client Alarm Functionality (optional function, enabled using configuration options)
- Other Optional Functionality (enabled using configuration options)

Note: Any code segment pertaining to the driver interfaces will work for both the static or dynamic configurations. It is not necessary to modify the code to move from one configuration to the other (i.e., from static or dynamic or static-multi).

System Interaction

This section describes Timer initialization and reinitialization.

Description

Initialization and Reinitialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized.

The DRV_TMR_Initialize function returns an object handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the Initialize interface would be used by the other system interfaces such as DRV_TMR_Deinitialize and DRV_TMR_Status, DRV_TMR_Tasks.

Example: Timer Initialization

```
DRV_TMR_INIT     init;
SYS_MODULE_OBJ   object;
SYS_STATUS       tmrStatus;

// populate the TMR init configuration structure
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId            = TMR_ID_2;
init.clockSource      = TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK;
init.prescale         = TMR_PRESCALE_VALUE_256;
init.interruptSource  = INT_SOURCE_TIMER_2;
init.mode             = DRV_TMR_OPERATION_MODE_16_BIT;
init.asyncWriteEnable = false;

object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&init);

if (object == SYS_MODULE_OBJ_INVALID)
{
    // Handle error
}
```

Deinitialization

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This routine may block if the driver is running in an OS environment that supports blocking operations and the driver requires system resources access. However, the routine will never block for hardware Timer access.

Status

Timer status is available to query the module state after initialization and reinitialization.

Tasks Routine

The interface DRV_TMR_Tasks needs to be called by the system task service in a polled environment and in an interrupt-based system.

Example: Polling

```
int main( void )
{
    SYS_MODULE_OBJ object;
    object = DRV_TMR_Initialize( DRV_TMR_INDEX_0, (SYS_MODULE_INIT *) &initConf );

    if( SYS_STATUS_READY != DRV_TMR_Status( object ) )
        return 0;

    while (1)
    {
        DRV_TMR_Tasks (object);
    }
}
```

Example: Interrupt

```
int main( void )
{
    SYS_MODULE_OBJ object;
    object = DRV_TMR_Initialize( DRV_TMR_INDEX_0, (SYS_MODULE_INIT *) &initConf );

    if( SYS_STATUS_READY != DRV_TMR_Status( object ) )
        return 0;

    while (1);
}

/* Sample interrupt routine not specific to any device family */
void ISR T1Interrupt(void)
{
    //Call the Timer Tasks routine
    DRV_TMR_Tasks(object);
}
```

Client Interaction

This section describes general client operation.

Description

General Client Operation

For the application to begin using an instance of the Timer module, it must call the DRV_TMR_Open function. This provides the configuration required to open the Timer instance for operation.

The Timer Driver supports only the 'DRV_IO_INTENT_EXCLUSIVE' IO_INTENT.

Example:

```
DRV_HANDLE handle;

// Configure the instance DRV_TMR_INDEX_1 with the configuration
handle = DRV_TMR_Open(DRV_TMR_INDEX_1, DRV_IO_INTENT_EXCLUSIVE);

if( handle == DRV_HANDLE_INVALID )
{
    // Client cannot open the instance.
}
```

The function DRV_TMR_Close closes an already opened instance of the Timer Driver, invalidating the handle. DRV_TMR_Open must have been

called to obtain a valid opened device handle.

Example:
```
DRV_HANDLE handle;

// Configure the instance DRV_TMR_INDEX_1 with the configuration
handle = DRV_TMR_Open(DRV_TMR_INDEX_1, DRV_IO_INTENT_EXCLUSIVE);

/*...*/

DRV_TMR_Close( handle );
```

The client has the option to check the status through the interface DRV_TMR_ClientStatus.

Example:
```
DRV_HANDLE handle;

// Configure the instance DRV_TMR_INDEX_1 with the configuration
handle = DRV_TMR_Open(DRV_TMR_INDEX_1, DRV_IO_INTENT_EXCLUSIVE);

if ( DRV_TMR_CLIENT_STATUS_READY != DRV_TMR_ClientStatus( handle ) )
   return 0;
```

Modification

This section describes Period modification for the different types of Timers (i.e., 16-/32-bit).

Description

These set of functions help modify the Timer periodicity at the client level.

Period Modification

Periodicity of Timer (16/32-bit) can be modified using DRV_TMR_AlarmPeriodSet and the current period can be obtained using DRV_TMR_AlarmPeriodGet.

Example:
```
DRV_HANDLE handle;
/* Open the client */
handle = DRV_TMR_Open( DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

/* ... */

/* Update the new period */
DRV_TMR_AlarmPeriodSet( handle, 0xC350);
```

Counter Modification

This section describes counter modification for the different types of Timers (i.e., 8-/16-/32-bit).

Description

These set of functions help modify the initial value of the Timer counters to help adjust any errors in the periodicity.

Counter Modification

The Timer initial value can be modified using DRV_TMR_CounterValueSet and the current counter value can be obtained using DRV_TMR_CounterValueGet.

Example:
```
DRV_HANDLE handle;
/* Open the client */
handle = DRV_TMR_Open( DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

/* ... */

/* Update the counter value */
/* Following code updates the initial value from 0x0000 to 0x0010
   to cover up any error in the previously set periodicity */

DRV_TMR_CounterValueSet( handle, 0x0010);
```

Core Functionality

This section describes core functionality of the Timer Driver.

Description

Core functionality provides an extremely basic interface for the driver operation.

Applications using the Timer core functionality need to perform the following:

1. The system should have completed the necessary initialization and DRV_TMR_Tasks should be called in a polled/interrupt environment.
2. Open_the driver using DRV_TMR_Open. The Timer Driver only supports exclusive access.
3. The Timer can be updated using DRV_TMR_AlarmPeriodSet. The previously set value can be retrieved using DRV_TMR_AlarmPeriodGet.
4. Start the driver using DRV_TMR_Start.
5. Poll for the elapsed alarm status using DRV_TMR_AlarmHasElapsed.
6. The client will be able to stop the started Timer instance using DRV_TMR_Stop at any time and will be able to close it using DRV_TMR_Close when it is no longer required.

Example:

```
/* Open the client */
handle = DRV_TMR_Open( DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );
DRV_TMR_Start (handle);
unsigned int alarmCount = 0;
while (1)
{
    if (true == DRV_TMR_AlarmHasElapsed (handle))
    {
        alarmCount++;
        // Do something
    }
}
```

Notes:

1. The user needs to stop the Timer before any updates on the counter or period and restart it later.
2. The Timer alarm count gets reset after any call to DRV_TMR_AlarmHasElapsed.
3. The Timer alarm status remains unchanged if the user stops the timer and restarts later.

Alarm Functionality

This section describes the Timer Driver alarm functionality.

Description

The Timer Driver provides alarm functionality.

Applications using the Timer alarm functionality, need to perform the following:

1. The system should have completed the necessary initialization and DRV_TMR_Tasks should be running in either a polled environment or in an interrupt environment.
2. Open the driver using DRV_TMR_Open. The Timer Driver supports exclusive access only.
3. Configure the alarm using DRV_TMR_AlarmRegister.
4. Start the driver using DRV_TMR_Start.
5. If a callback is supplied, the Timer Driver will call the callback function when the alarm expires.
6. The client will be able to stop the started Timer module instance using DRV_TMR_Stop at any time and will be able to close it using DRV_TMR_Close when it is no longer required.
7. The client can deregister the callback by using DRV_TMR_AlarmDeregister.

Example:

```
DRV_HANDLE handle;
/* Open the client */
handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
/* Configure the timer alarm feature */
uint32_t myFreq = 1000; // 1KHz
uint32_t clkFreq = DRV_TMR_CounterFrequencyGet(tmrHandle);   // timer running frequency

// calculate the divider needed
uint32_t divider = clkFreq / myFreq;

// Start the alarm
if(!DRV_TMR_AlarmRegister ( tmrHandle, divider, true, 0, CallBackFreq ))
```

```
{
    // divider value could not be obtain;
    // handle the error
    //
    return;
}

DRV_TMR_Start (handle);

// The driver tasks function calls the client registered callback after the alarm expires.
void CallBackFreq (uintptr_t context, uint32_t alarmCount)
{
    // Do something specific on an alarm event trigger
}
```

Optional Interfaces

This section describes additional/optional client interfaces.

Description

Additional/Optional client interfaces include the following:

Get Operating Frequency

The function DRV_TMR_CounterFrequencyGet provides the client with the information on the Timer operating frequency.

Example:

```
DRV_HANDLE handle;
uint32_t freq;

/* Open the client */
handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);

freq = DRV_TMR_OperatingFrequencyGet (handle);
```

Example Usage of the Timer Driver

This section describes typical usage of the Timer Driver for various Timer modules in polling/interrupt advanced/core modes.

Description

The user can pass NULL to the driver initialize interface. However, the respective configuration parameters need to be configured in the correct manner.

Example:

```
//Polled mode under 32-bit count mode for a PIC32 device using the alarm feature
SYS_MODULE_OBJ object;

// main
DRV_TMR_INIT init;
DRV_HANDLE handle;

init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId            = TMR_ID_2;
init.clockSource      = TMR_CLKSOURCE_INTERNAL;
init.prescale         = TMR_PRESCALE_TX_VALUE_256;
init.interruptSource  = INT_SOURCE_TIMER_3;
init.mode             = DRV_TMR_OPERATION_MODE_16_BIT;init.asyncWriteEnable = false;

object = DRV_TMR_Initialize (DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&init);
if ( SYS_STATUS_READY != DRV_TMR_Status(object))
    return 0;

handle = DRV_TMR_Open (DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if ( DRV_TMR_CLIENT_STATUS_READY != DRV_TMR_ClientStatus(handle))
    return 0;

if(!DRV_TMR_AlarmRegister ( tmrHandle, divider, true, 0, AlarmCallback ))
{
    // divider value could not be obtain;
```

```
    // handle the error
}

DRV_TMR_Start (handle);

while (1)
{
    DRV_TMR_Tasks (object);
}

DRV_TMR_Stop (handle);

DRV_TMR_Close (handle);
if ( DRV_TMR_CLIENT_STATUS_INVALID != DRV_TMR_ClientStatus(handle))
    return 0;

DRV_TMR_Deinitialize (object);
// end main

void AlarmCallback (uintptr_t context, uint32_t alarmCount)
{
  // Do something specific on an alarm event trigger
}
```

Configuring the Library

Macros

	Name	Description
	DRV_TMR_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_TMR_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_TMR_CLOCK_PRESCALER	Sets the default timer driver clock prescaler.
	DRV_TMR_MODE	Sets the default timer driver clock operating mode.
	DRV_TMR_MODULE_ID	Sets the default timer module ID to be used by the timer driver.
	DRV_TMR_MODULE_INIT	Sets the default module init value for the timer driver.
	DRV_TMR_INTERRUPT_SOURCE	Sets the default timer driver clock interrupt source
	DRV_TMR_ASYNC_WRITE_ENABLE	Controls Asynchronous Write mode of the Timer.
	DRV_TMR_CLOCK_SOURCE	Sets the default timer driver clock source.
	DRV_TMR_CLIENTS_NUMBER	Sets up the maximum number of clients that can be supported by an instance of the dynamic driver.

Description

The configuration of the Timer Driver Library is based on the file `system_config.h`.

This header file contains the configuration selection for the Timer Driver Library build. Based on the selections made here and the system setup, the Timer Driver may support the selected features. These configuration settings will apply to all instances of the driver.

This header can be placed anywhere in the application-specific folders and the path of this header needs to be presented to the include search for a successful build. Refer to the Applications Help section for more details.

DRV_TMR_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported by the dynamic driver.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_INSTANCES_NUMBER 5
```

Description

Hardware instances support

This definition sets up the maximum number of hardware instances that can be supported by the dynamic driver.

Remarks

None

DRV_TMR_INTERRUPT_MODE Macro

Controls operation of the driver in the interrupt or polled mode.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_INTERRUPT_MODE true
```

Description

TMR Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of timer operation is desired
- false - Select if polling mode of timer operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

DRV_TMR_CLOCK_PRESCALER Macro

Sets the default timer driver clock prescaler.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_CLOCK_PRESCALER (TMR_PRESCALE_VALUE_256)
```

Description

Default timer driver clock prescaler

This macro sets the default timer driver clock prescaler.

Remarks

This value can be overridden by a run time initialization value.

DRV_TMR_MODE Macro

Sets the default timer driver clock operating mode.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_MODE (DRV_TMR_OPERATION_MODE_16_BIT)
```

Description

Default timer driver clock operating mode

This macro sets the default timer driver clock operating mode.

Remarks

This value can be overridden by a run time initialization value.

DRV_TMR_MODULE_ID Macro

Sets the default timer module ID to be used by the timer driver.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_MODULE_ID (TMR_ID_2)
```

Description

Default timer driver index

This macro sets the default timer module ID to be used by the timer driver.

Remarks

This value can be overridden by a run time initialization value.

DRV_TMR_MODULE_INIT Macro

Sets the default module init value for the timer driver.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_MODULE_INIT (SYS_MODULE_POWER_RUN_FULL)
```

Description

Default module init object configuration

This macro sets the default module init value for the timer driver.

Remarks

This value can be overridden by a run time initialization value.

DRV_TMR_INTERRUPT_SOURCE Macro

Sets the default timer driver clock interrupt source

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_INTERRUPT_SOURCE (INT_SOURCE_TIMER_2)
```

Description

Default timer driver clock interrupt source

This macro sets the default timer driver clock interrupt source

Remarks

This value can be overridden by a run time initialization value.

DRV_TMR_ASYNC_WRITE_ENABLE Macro

Controls Asynchronous Write mode of the Timer.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_ASYNC_WRITE_ENABLE false
```

Description

TMR Asynchronous write mode configuration

This macro controls the Asynchronous Write mode of the Timer. This macro accepts the following values:

- true - Configures the Timer to enable asynchronous write control
- false - Configures the Timer to disable asynchronous write control
- DRV_CONFIG_NOT_SUPPORTED - When the feature is not supported on the instance.

Remarks

This feature is not available in all modules/devices. Refer to the specific device data sheet for more information.

DRV_TMR_CLOCK_SOURCE Macro

Sets the default timer driver clock source.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_CLOCK_SOURCE (DRV_TMR_CLKSOURCE_INTERNAL)
```

Description

Default timer driver clock source

This macro sets the default timer driver clock source.

Remarks

This value can be overridden by a run time initialization value.

DRV_TMR_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be supported by an instance of the dynamic driver.

File

drv_tmr_config_template.h

C

```
#define DRV_TMR_CLIENTS_NUMBER 1
```

Description

Client instances support

This definition sets up the maximum number of clients that can be supported by an instance of the dynamic driver.

Remarks

Currently each client is required to get exclusive access to the timer module. Therfore the DRV_TMR_CLIENTS_NUMBER should always be set to 1.

Building the Library

This section lists the files that are available in the Timer Driver Library.

Description

This section list the files that are available in the `\src` folder of the Timer Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/tmr`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_tmr.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_tmr_dynamic.c`	Basic Timer driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library

Module Dependencies

The Timer Driver Library depends on the following modules:

- Clock System Service Library
- Interrupt System Service Library
- Interrupt Peripheral Library
- Device Control System Service Library

Library Interface

a) System Interaction Functions

	Name	Description
	DRV_TMR_Deinitialize	Deinitializes the specified instance of the Timer driver. **Implementation:** Dynamic
	DRV_TMR_Initialize	Initializes the Timer driver. **Implementation:** Static/Dynamic
	DRV_TMR_Status	Provides the current status of the Timer driver. **Implementation:** Dynamic
	DRV_TMR_Tasks	Maintains the driver's state machine. **Implementation:** Dynamic
	DRV_TMR_ClockSet	Sets the timers clock by selecting the source and prescaler. **Implementation:** Dynamic
	DRV_TMR_GateModeSet	Enables the Gate mode. **Implementation:** Dynamic

b) Core Functions

	Name	Description
	DRV_TMR_ClientStatus	Gets the status of the client operation. **Implementation:** Dynamic
	DRV_TMR_Close	Closes an opened instance of the Timer driver. **Implementation:** Dynamic
	DRV_TMR_Open	Opens the specified Timer driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_TMR_Start	Starts the Timer counting. **Implementation:** Static/Dynamic
	DRV_TMR_Stop	Stops the Timer from counting. **Implementation:** Static/Dynamic

c) Alarm Functions

	Name	Description
	DRV_TMR_Alarm16BitDeregister	Removes a previously set alarm. **Implementation:** Dynamic
	DRV_TMR_Alarm32BitDeregister	Removes a previously set alarm. **Implementation:** Dynamic
	DRV_TMR_AlarmHasElapsed	Provides the status of Timer's period elapse. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriod16BitGet	Provides the 16-bit Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriod16BitSet	Updates the 16-bit Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriod32BitGet	Provides the 32-bit Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriod32BitSet	Updates the 32-bit Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmDisable	Disables an alarm signal. **Implementation:** Dynamic
	DRV_TMR_AlarmEnable	Re-enables an alarm signal. **Implementation:** Dynamic
	DRV_TMR_Alarm16BitRegister	Sets up an alarm. **Implementation:** Dynamic
	DRV_TMR_Alarm32BitRegister	Sets up an alarm. **Implementation:** Dynamic
	DRV_TMR_AlarmDeregister	Removes a previously set alarm. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriodGet	Provides the Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriodSet	Updates the Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmRegister	Sets up an alarm. **Implementation:** Dynamic

d) Counter Control Functions

	Name	Description
	DRV_TMR_CounterFrequencyGet	Provides the Timer input frequency. **Implementation:** Dynamic
	DRV_TMR_CounterValue16BitGet	Reads the 16-bit Timer's counter register. **Implementation:** Dynamic
	DRV_TMR_CounterValue16BitSet	Updates the 16-bit Timer's counter register. **Implementation:** Dynamic
	DRV_TMR_CounterValue32BitGet	Reads the 32-bit Timer's counter register. **Implementation:** Dynamic
	DRV_TMR_CounterValue32BitSet	Updates the 32-bit Timer's counter register. **Implementation:** Dynamic
	DRV_TMR_CounterClear	Clears the Timer's counter register. **Implementation:** Static/Dynamic
	DRV_TMR_CounterValueGet	Reads the Timer's counter register. **Implementation:** Static/Dynamic
	DRV_TMR_CounterValueSet	Updates the Timer's counter register. **Implementation:** Static/Dynamic

e) Miscellaneous Functions

	Name	Description
	DRV_TMR_GateModeClear	Enables the Gate mode. **Implementation:** Dynamic

	DRV_TMR_PrescalerGet	This function gets the currently selected prescaler. **Implementation:** Dynamic
	DRV_TMR_OperationModeGet	This function gets the currently selected operation mode. **Implementation:** Dynamic
	DRV_TMR_DividerRangeGet	Returns the Timer divider values. **Implementation:** Dynamic
	DRV_TMR_Tasks_ISR	Maintains the driver's state machine, processes the events and implements its ISR. **Implementation:** Dynamic

f) Data Types and Constants

	Name	Description
	DRV_TMR_CALLBACK	Pointer to a Timer driver callback function data type.
	DRV_TMR_INIT	Defines the Timer driver initialization data.
	DRV_TMR_CLIENT_STATUS	Identifies the client-specific status of the Timer driver
	DRV_TMR_CLK_SOURCES	Lists the clock sources available for timer driver.
	DRV_TMR_DIVIDER_RANGE	This data structure specifies the divider values that can be obtained by the timer module.
	DRV_TMR_OPERATION_MODE	Lists the operation modes available for timer driver.
	DRV_TMR_INDEX_COUNT	Number of valid Timer driver indices.
	DRV_TMR_INDEX_0	Timer driver index definitions
	DRV_TMR_INDEX_1	This is macro DRV_TMR_INDEX_1.
	DRV_TMR_INDEX_2	This is macro DRV_TMR_INDEX_2.
	DRV_TMR_INDEX_3	This is macro DRV_TMR_INDEX_3.
	DRV_TMR_INDEX_4	This is macro DRV_TMR_INDEX_4.
	DRV_TMR_INDEX_5	This is macro DRV_TMR_INDEX_5.
	DRV_TMR_INDEX_6	This is macro DRV_TMR_INDEX_6.
	DRV_TMR_INDEX_7	This is macro DRV_TMR_INDEX_7.
	DRV_TMR_INDEX_8	This is macro DRV_TMR_INDEX_8.
	DRV_TMR_INDEX_9	This is macro DRV_TMR_INDEX_9.

Description

This section describes the functions of the Timer Driver Library.

Refer to each section for a detailed description.

a) System Interaction Functions

DRV_TMR_Deinitialize Function

Deinitializes the specified instance of the Timer driver.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the Timer driver, disabling its operation (and any hardware). All internal data is invalidated.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_TMR_Status operation. The system has to use DRV_TMR_Status to find out when the module is in the ready state.

Preconditions

The DRV_TMR_Initialize function must have been called before calling this function and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      tmrObject;     //  Returned from DRV_TMR_Initialize
SYS_STATUS          tmrStatus;

DRV_TMR_Deinitialize ( tmrObject );

tmrStatus = DRV_TMR_Status ( tmrObject );

if ( SYS_MODULE_UNINITIALIZED == tmrStatus )
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TMR_Initialize

Function

void DRV_TMR_Deinitialize (SYS_MODULE_OBJ object)

DRV_TMR_Initialize Function

Initializes the Timer driver.

Implementation: Static/Dynamic

File

drv_tmr.h

C

```
SYS_MODULE_OBJ DRV_TMR_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver object. Otherwise, it returns SYS_MODULE_OBJ_INVALID. The returned object must be passed as argument to DRV_TMR_Deinitialize, DRV_TMR_Tasks, DRV_TMR_Tasks_ISR and DRV_TMR_Status functions.

Description

This function initializes the Timer driver, making it ready for clients to open and use it.

Remarks

This function must be called before any other Timer driver function is called.

This function should only be called once during system initialization unless DRV_TMR_Deinitialize is called to deinitialize the driver instance.

This function will NEVER block for hardware access. The system must use DRV_TMR_Status to find out when the driver is in the ready state.

Build configuration options may be used to statically override options in the "init" structure and will take precedence over initialization data passed using this function.

Preconditions

None.

Example

```
DRV_TMR_INIT    init;
SYS_MODULE_OBJ  objectHandle;

// Populate the timer initialization structure
init.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
init.tmrId            = TMR_ID_2;
init.clockSource      = DRV_TMR_CLKSOURCE_INTERNAL;
init.prescale         = TMR_PRESCALE_VALUE_256;
init.interruptSource  = INT_SOURCE_TIMER_2;
```

```
init.mode             = DRV_TMR_OPERATION_MODE_16_BIT;
init.asyncWriteEnable = false;

// Do something

objectHandle = DRV_TMR_Initialize ( DRV_TMR_INDEX_0, (SYS_MODULE_INIT*)&init );

if ( SYS_MODULE_OBJ_INVALID == objectHandle )
{
    // Handle error
}
```

Parameters

Parameters	Description
drvIndex	Index for the driver instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_TMR_Initialize
(
const SYS_MODULE_INDEX drvIndex,
const SYS_MODULE_INIT * const init
)

DRV_TMR_Status Function

Provides the current status of the Timer driver.

Implementation: Dynamic

File

drv_tmr.h

C

```
SYS_STATUS DRV_TMR_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is initialized and ready for operation

Description

This function provides the current status of the Timer driver.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_STATUS_ERROR - Indicates that the driver is in an error state

Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_UNINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

The this operation can be used to determine when any of the driver's module level operations has completed.

Once the status operation returns SYS_STATUS_READY, the driver is ready for operation.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

Preconditions

The DRV_TMR_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_TMR_Initialize
SYS_STATUS          tmrStatus;

tmrStatus = DRV_TMR_Status ( object );
```

```
else if ( SYS_STATUS_ERROR >= tmrStatus )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TMR_Initialize

Function

SYS_STATUS DRV_TMR_Status (SYS_MODULE_OBJ object)

DRV_TMR_Tasks Function

Maintains the driver's state machine.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_Tasks(SYS_MODULE_OBJ object);
```

Returns

None

Description

This function is used to maintain the driver's internal state machine and processes the timer events..

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks)

Preconditions

The DRV_TMR_Initialize function must have been called for the specified Timer driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_TMR_Initialize

while (true)
{
    DRV_TMR_Tasks ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_TMR_Initialize)

Function

void DRV_TMR_Tasks (SYS_MODULE_OBJ object)

DRV_TMR_ClockSet Function

Sets the timers clock by selecting the source and prescaler.

Implementation: Dynamic

File

drv_tmr.h

C

```
bool DRV_TMR_ClockSet(DRV_HANDLE handle, DRV_TMR_CLK_SOURCES clockSource, TMR_PRESCALE preScale);
```

Returns

- true - if the operation is successful
- false - either the handle is invalid or the clockSource and/or prescaler are not supported

Description

This function sets the timer clock by selecting the source and prescaler. The clock sources are device specific, refer device datasheet for supported clock sources. If unsupported clock source is passed then the behaviour of this function is unpredictable.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called. Must have selected 32-Bit timer mode if mode selection is applicable.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE              tmrHandle;  // Returned from DRV_TMR_Open

DRV_TMR_ClockSet ( tmrHandle, DRV_TMR_CLKSOURCE_INTERNAL, TMR_PRESCALE_VALUE_256 );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
clockSource	Clock source of the timer
preScale	Timer's Prescaler divisor

Function

bool DRV_TMR_ClockSet

(

DRV_HANDLE handle,

DRV_TMR_CLK_SOURCES clockSource,

TMR_PRESCALE preScale

)

DRV_TMR_GateModeSet Function

Enables the Gate mode.

Implementation: Dynamic

File

drv_tmr.h

C

```
bool DRV_TMR_GateModeSet(DRV_HANDLE handle);
```

Returns

- true - if the operation is successful
- false - either the handle is invalid or the gate mode is not supported

Description

This function enables the Gated mode of Timer. User can measure the duration of an external signal in this mode. Once the Gate mode is enabled, Timer will start on the raising edge of the external signal. It will keep counting until the next falling edge.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open

DRV_TMR_GateModeSet ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

bool DRV_TMR_GateModeSet (DRV_HANDLE handle)

b) Core Functions

DRV_TMR_ClientStatus Function

Gets the status of the client operation.

Implementation: Dynamic

File

drv_tmr.h

C

```
DRV_TMR_CLIENT_STATUS DRV_TMR_ClientStatus(DRV_HANDLE handle);
```

Returns

None

Description

This function gets the status of the recently completed client level operation.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called for the specified Timer driver instance.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open
DRV_TMR_CLIENT_STATUS tmrDrvStatus;

tmrDrvStatus = DRV_TMR_ClientStatus ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

DRV_TMR_CLIENT_STATUS DRV_TMR_ClientStatus (DRV_HANDLE handle)

DRV_TMR_Close Function

Closes an opened instance of the Timer driver.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_Close(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the Timer driver, invalidating the handle.

Remarks

After calling this function, the handle passed in "handle" must not be used with any of the remaining driver functions. A new handle must be obtained by calling DRV_TMR_Open before the caller may use the driver again.

Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_TMR_Initialize function must have been called for the specified Timer driver instance.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TMR_Open

DRV_TMR_Close ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TMR_Close (DRV_HANDLE handle)

DRV_TMR_Open Function

Opens the specified Timer driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_tmr.h

C

```
DRV_HANDLE DRV_TMR_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
```

Returns

If successful, the function returns a valid open instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID.

Description

This function opens the specified Timer driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. Timer driver does not support multiple clients. If two tasks want to use the timer, one should wait until the other one gets closed.

Remarks

The handle returned is valid until the DRV_TMR_Close function is called.

This function will NEVER block waiting for hardware.

If the requested intent flags are not supported, the function will return DRV_HANDLE_INVALID.

The Timer driver does not support DRV_IO_INTENT_SHARED. Only exclusive access is supported for now.

Preconditions

The DRV_TMR_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_TMR_Open ( DRV_TMR_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

if ( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_TMR_Open
(
const SYS_MODULE_INDEX index,
const DRV_IO_INTENT intent
)

DRV_TMR_Start Function

Starts the Timer counting.

Implementation: Static/Dynamic

File

drv_tmr.h

C

```
bool DRV_TMR_Start(DRV_HANDLE handle);
```

Returns

- true - if the operation succeeded
- false - the supplied handle is invalid or the client doesn't have the needed parameters to run (alarm callback and period)

Description

This function starts the Timer counting.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Timer parameters must have been set by a call to DRV_TMR_AlarmRegister.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open

DRV_TMR_AlarmRegister(tmrHandle, 0x100, true, 0, myTmrCallback);
DRV_TMR_Start ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

bool DRV_TMR_Start (DRV_HANDLE handle)

DRV_TMR_Stop Function

Stops the Timer from counting.

Implementation: Static/Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_Stop(DRV_HANDLE handle);
```

Returns

None.

Description

This function stops the running Timer from counting.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TMR_Open

DRV_TMR_Stop ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TMR_Stop (DRV_HANDLE handle)

c) Alarm Functions

DRV_TMR_Alarm16BitDeregister Function

Removes a previously set alarm.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_Alarm16BitDeregister(DRV_HANDLE handle);
```

Returns

None.

Description

This function removes a previously set alarm. This API is valid only if the 16-bit mode of the timer is selected Otherwise use DRV_TMR_Alarm32BitDeregister function.

Remarks

This function will be deprecated later, so avoid using it as much as possible. Use generic function "DRV_TMR_AlarmDeregister" instead of this.

Preconditions

The DRV_TMR_Initialize function must have been called. Must have selected 16-Bit timer mode if mode selection is applicable.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

DRV_TMR_Alarm16BitRegister function must have been called before.

Example

```
//Example of a key debounce check

static unsigned int lastReadKey, readKey, keyCount, globalKeyState;
DRV_HANDLE              tmrHandle;  // Returned from DRV_TMR_Open
//Do the initialization with 'mode' set to DRV_TMR_OPERATION_MODE_16_BIT

void keyPressDetect ()
{
    // Calculate the count to be passed on from the clock input
    //Periodically toggle LED
    DRV_TMR_Alarm16BitRegister ( tmrHandle, 0xFF00, true,
                                    DebounceCheck );
}

void DebounceCheck ( uintptr_t context )
{
    readKey = AppReadKey();

    if ( readKey != lastReadKey )
    {
        lastReadKey = readKey;
        keyCount = 0;
    }
    else
    {
        if ( keyCount > 20 )
        {
            globalKeyState = readKey;
            DRV_TMR_Alarm16BitDeregister ( tmrHandle );
        }
        keyCount++;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TMR_Alarm16BitDeregister (DRV_HANDLE handle)

DRV_TMR_Alarm32BitDeregister Function

Removes a previously set alarm.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_Alarm32BitDeregister(DRV_HANDLE handle);
```

Returns

None.

Description

This function removes a previously set alarm. This API is valid only if the 32-bit mode of the timer is selected Otherwise use DRV_TMR_Alarm16BitDeregister function.

Remarks

In most of the devices only even numbered instances of timer supports 32-bit mode. This function will be deprecated later, so avoid using it as much as possible. Use generic function "DRV_TMR_AlarmDeregister" instead of this.

Preconditions

The DRV_TMR_Initialize function must have been called. Must have selected 32-Bit timer mode if mode selection is applicable.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

DRV_TMR_Alarm32BitRegister function must have been called before.

Example

```
//Example of a key debounce check

static unsigned int lastReadKey, readKey, keyCount, globalKeyState;
DRV_HANDLE              tmrHandle;  // Returned from DRV_TMR_Open
//Do the initialization with 'mode' set to DRV_TMR_OPERATION_MODE_32_BIT

void keyPressDetect ( void )
{
    // Calculate the count to be passed on from the clock input
    //Periodically check the key status
    DRV_TMR_Alarm32BitRegister ( tmrHandle, 0xFF0FFD20, true, 0,
                                    DebounceCheck );
}

void DebounceCheck ( uintptr_t context )
{
    readKey = AppReadKey();

    if ( readKey != lastReadKey )
    {
        lastReadKey = readKey;
        keyCount = 0;
    }
    else
    {
        if ( keyCount > 20 )
        {
            //Key is stable now
            globalKeyState = readKey;
            DRV_TMR_Alarm32BitDeregister ( tmrHandle );
        }
        keyCount++;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TMR_Alarm32BitDeregister (DRV_HANDLE handle)

DRV_TMR_AlarmHasElapsed Function

Provides the status of Timer's period elapse.

Implementation: Dynamic

File

drv_tmr.h

C

```
uint32_t DRV_TMR_AlarmHasElapsed(DRV_HANDLE handle);
```

Returns

Number of times timer has elapsed since the last call.

Description

This function returns the number of times Timer's period has elapsed since last call to this API has made. On calling this API, the internally maintained counter will be cleared and count will be started again from next elapse.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      tmrHandle;  // Returned from DRV_TMR_Open
bool            elapseStatus;
SYS_MODULE_OBJ  tmrObject // Returned by DRV_TMR_Initialize
unsigned int    appInternalTime = 0;

Sys_Tasks()
{
    //Timer task will be called from ISR

    APP_TimeUpdate_Task();

    //Other Tasks
}

void APP_TimeUpdate_Task ( void )
{
    //We will not miss a count even though we are late
    appInternalTime += DRV_TMR_AlarmHasElapsed ( tmrHandle );
}
```

Parameters

Parameters	Description
handle	A valid handle, returned from the DRV_TMR_Open

Function

unsigned int DRV_TMR_AlarmHasElapsed (DRV_HANDLE handle)

DRV_TMR_AlarmPeriod16BitGet Function

Provides the 16-bit Timer's period.

Implementation: Dynamic

File

drv_tmr.h

C

```
uint16_t DRV_TMR_AlarmPeriod16BitGet(DRV_HANDLE handle);
```

Returns

16-bit timer period value

Description

This function gets the 16-bit Timer's period. This API is valid only if the 16-bit mode of the timer is selected. Otherwise use DRV_TMR_AlarmPeriod32BitGet function.

Remarks

This function will be deprecated later, so avoid using it as much as possible. Use generic function "DRV_TMR_AlarmPeriodGet" instead of this.

Preconditions

The DRV_TMR_Initialize function must have been called. Must have selected 16-Bit timer mode if mode selection is applicable.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open
uint16_t period;

period = DRV_TMR_AlarmPeriod16BitGet ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint16_t DRV_TMR_AlarmPeriod16BitGet (DRV_HANDLE handle)

DRV_TMR_AlarmPeriod16BitSet Function

Updates the 16-bit Timer's period.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_AlarmPeriod16BitSet(DRV_HANDLE handle, uint16_t value);
```

Returns

None.

Description

This function updates the 16-bit Timer's period. This API is valid only if the 16-bit mode of the timer is selected Otherwise use DRV_TMR_AlarmPeriod32BitSet function.

Remarks

This function will be deprecated later, so avoid using it as much as possible. Use generic function "DRV_TMR_AlarmPeriodSet" instead of this.

Preconditions

The DRV_TMR_Initialize function must have been called. Must have selected 16-Bit timer mode if mode selection is applicable.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TMR_Open

DRV_TMR_AlarmPeriod16BitSet ( handle, 0x1000 );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
value	16-bit Period value

Function

void DRV_TMR_AlarmPeriod16BitSet (DRV_HANDLE handle, uint16_t value)

DRV_TMR_AlarmPeriod32BitGet Function

Provides the 32-bit Timer's period.

Implementation: Dynamic

File

drv_tmr.h

C

```
uint32_t DRV_TMR_AlarmPeriod32BitGet(DRV_HANDLE handle);
```

Returns

32-bit Timer period value.

Description

This function gets the 32-bit Timer's period. This API is valid only if the 32-bit mode of the timer is selected Otherwise use DRV_TMR_AlarmPeriod16BitGet function.

Remarks

In most of the devices only even numbered instances of timer supports 32-bit mode. This function will be deprecated later, so avoid using it as much as possible. Use generic function "DRV_TMR_AlarmPeriodGet" instead of this.

Preconditions

The DRV_TMR_Initialize function must have been called. Must have selected 32-Bit timer mode.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TMR_Open
uint32_t period;

period = DRV_TMR_AlarmPeriod32BitGet ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_TMR_AlarmPeriod32BitGet (DRV_HANDLE handle)

DRV_TMR_AlarmPeriod32BitSet Function

Updates the 32-bit Timer's period.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_AlarmPeriod32BitSet(DRV_HANDLE handle, uint32_t period);
```

Returns

None.

Description

This function updates the 32-bit Timer's period. This API is valid only if the 32-bit mode of the timer is selected Otherwise use DRV_TMR_AlarmPeriod16BitSet function.

Remarks

In most of the devices only even numbered instances of timer supports 32-bit mode. This function will be deprecated later, so avoid using it as much as possible. Use generic function "DRV_TMR_AlarmPeriodSet" instead of this.

Preconditions

The DRV_TMR_Initialize unction must have been called. Must have selected 32-Bit timer mode.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TMR_Open

DRV_TMR_AlarmPeriod32BitSet ( handle, 0xFFFFFF00 );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
period	32-bit Period value

Function

void DRV_TMR_AlarmPeriod32BitSet (DRV_HANDLE handle, uint32_t period)

DRV_TMR_AlarmDisable Function

Disables an alarm signal.

Implementation: Dynamic

File

drv_tmr.h

C

```
bool DRV_TMR_AlarmDisable(DRV_HANDLE handle);
```

Returns

The current status of the alarm:

- true if the alarm was currently enabled
- false if the alarm was currently disabled

Description

This function allows the client to disable an alarm generation. Use DRV_TMR_AlarmEnable to re-enable.

Remarks

When the driver operates in interrupts this call resolves to a device interrupt disable.

Do NOT disable the timer except for very short periods of time. If the time that the interrupt is disabled is longer than a wrap around period and the interrupt is missed, the hardware has no means of recovering and the resulting timing will be inaccurate.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

A client alarm must be active.

Example

Parameters

Parameters	Description
handle	A valid handle, returned from DRV_TMR_Open

Function

bool DRV_TMR_AlarmDisable (DRV_HANDLE handle);

DRV_TMR_AlarmEnable Function

Re-enables an alarm signal.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_AlarmEnable(DRV_HANDLE handle, bool enable);
```

Returns

None

Description

This function allows the client to re-enable an alarm after it has been disabled by a DRV_TMR_AlarmDisable call.

Remarks

When the driver operates in interrupts this call resolves to a device interrupt re-enable.

Preconditions

The DRV_TMR_Initialize function must have been called. DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

Parameters

Parameters	Description
handle	A valid handle, returned from DRV_TMR_Open
enable	boolean to enable the current callback

Function

void DRV_TMR_AlarmEnable (DRV_HANDLE handle, bool enable);

DRV_TMR_Alarm16BitRegister Function

Sets up an alarm.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_Alarm16BitRegister(DRV_HANDLE handle, uint16_t period, bool isPeriodic, uintptr_t context,
DRV_TMR_CALLBACK callBack);
```

Returns

None

Description

This function sets up an alarm, allowing the client to receive a callback from the driver when the counter period elapses. Alarms can be one-shot or periodic. This API is valid only if the 16-bit mode of the timer is selected. Otherwise use DRV_TMR_Alarm32BitRegister function.

Remarks

This function will be deprecated later, so avoid using it as much as possible. Use generic function "DRV_TMR_AlarmRegister" instead of this.

Preconditions

The DRV_TMR_Initialize function must have been called. Must have selected 16-Bit timer mode if mode selection is applicable.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
//Do the initialization with 'mode' set to DRV_TMR_OPERATION_MODE_16_BIT

void setupTask ()
{
    DRV_HANDLE              tmrHandle;  // Returned from DRV_TMR_Open

    // Calculate the count to be passed on from the clock input
```

```
    //Periodically toggle LED
    DRV_TMR_Alarm16BitRegister ( tmrHandle, 0xFF40, true, 0,
                                    ToggleLedCallBack );
}

void ToggleLedCallBack ( uintptr_t context )
{
    //Toggle
}
```

Parameters

Parameters	Description
handle	A valid handle, returned from DRV_TMR_Open
period	16-bit period which will be loaded into the Timer hardware register.
isPeriodic	Flag indicating whether the alarm should be one-shot or periodic.
context	A reference, call back function will be called with the same reference.
callBack	A call back function which will be called on period elapse.

Function

void DRV_TMR_Alarm16BitRegister
(
DRV_HANDLE handle,
uint16_t period,
bool isPeriodic,
uintptr_t context,
DRV_TMR_CALLBACK callBack
)

DRV_TMR_Alarm32BitRegister Function

Sets up an alarm.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_Alarm32BitRegister(DRV_HANDLE handle, uint32_t period, bool isPeriodic, uintptr_t context,
DRV_TMR_CALLBACK callBack);
```

Returns

None

Description

This function sets up an alarm, allowing the client to receive a callback from the driver when the counter period elapses. Alarms can be one-shot or periodic. This API is valid only if the 32-bit mode of the timer is selected Otherwise use DRV_TMR_Alarm16BitRegister function.

Remarks

In most of the devices only even numbered instances of timer supports 32-bit mode. This function will be deprecated later, so avoid using it as much as possible. Use generic function "DRV_TMR_AlarmRegister" instead of this.

Preconditions

The DRV_TMR_Initialize function must have been called. Must have selected 32-Bit timer mode.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
//Do the initialization with 'mode' set to DRV_TMR_OPERATION_MODE_32_BIT

void setupTask ()
{
    DRV_HANDLE              tmrHandle;  // Returned from DRV_TMR_Open
```

```
    // Calculate the count to be passed on from the clock input
    //Periodically toggle LED
    DRV_TMR_Alarm32BitRegister ( tmrHandle, 0xFFFFFF00, true, 0,
                                    ToggleLedCallBack );
}

void ToggleLedCallBack ( uintptr_t context )
{
    //Toggle
}
```

Parameters

Parameters	Description
handle	A valid handle, returned from DRV_TMR_Open
period	32-bit period which will be loaded into the Timer hardware register.
isPeriodic	Flag indicating whether the alarm should be one-shot or periodic.
context	A reference, call back function will be called with the same reference.
callBack	A call back function which will be called on period elapse.

Function

void DRV_TMR_Alarm32BitRegister
(
DRV_HANDLE handle,
uint32_t period,
bool isPeriodic,
uintptr_t context,
DRV_TMR_CALLBACK callBack
)

DRV_TMR_AlarmDeregister Function

Removes a previously set alarm.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_AlarmDeregister(DRV_HANDLE handle);
```

Returns

None.

Description

This function removes a previously set alarm.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

DRV_TMR_AlarmRegister function must have been called before.

Example

```
// Example of a key debounce check

static unsigned int lastReadKey, readKey, keyCount, globalKeyState;
DRV_HANDLE              tmrHandle;  // Returned from DRV_TMR_Open

void keyPressDetect ()
```

```
{
    // Calculate the count to be passed on from the clock input
    DRV_TMR_AlarmRegister ( tmrHandle, 0xFF00, true, DebounceCheck );
}

void DebounceCheck ( uintptr_t context )
{
    readKey = AppReadKey();

    if ( readKey != lastReadKey )
    {
        lastReadKey = readKey;
        keyCount = 0;
    }
    else
    {
        if ( keyCount > 20 )
        {
            globalKeyState = readKey;
            DRV_TMR_AlarmDeregister ( tmrHandle );
        }
        keyCount++;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TMR_AlarmDeregister (DRV_HANDLE handle)

DRV_TMR_AlarmPeriodGet Function

Provides the Timer's period.

Implementation: Dynamic

File

drv_tmr.h

C

```
uint32_t DRV_TMR_AlarmPeriodGet(DRV_HANDLE handle);
```

Returns

Timer period value:

- a 16 bit value if the timer is configured in 16 bit mode
- a 32 bit value if the timer is configured in 32 bit mode

Description

This function gets the Timer's period.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open
uint32_t period;

period = DRV_TMR_AlarmPeriodGet ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_TMR_AlarmPeriodGet (DRV_HANDLE handle)

DRV_TMR_AlarmPeriodSet Function

Updates the Timer's period.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_AlarmPeriodSet(DRV_HANDLE handle, uint32_t value);
```

Returns

None.

Description

This function updates the Timer's period.

Remarks

- The period value will be truncated to a 16 bit value if the timer is

configured in 16 bit mode.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TMR_Open

DRV_TMR_AlarmPeriodSet ( handle, 0x1000 );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
value	Period value • a 16 bit value if the timer is configured in 16 bit mode • a 32 bit value if the timer is configured in 32 bit mode

Function

void DRV_TMR_AlarmPeriodSet (DRV_HANDLE handle, uint32_t value)

DRV_TMR_AlarmRegister Function

Sets up an alarm.

Implementation: Dynamic

File

drv_tmr.h

C

```
bool DRV_TMR_AlarmRegister(DRV_HANDLE handle, uint32_t divider, bool isPeriodic, uintptr_t context,
DRV_TMR_CALLBACK callBack);
```

Returns

- true - if the call succeeded
- false - the obtained divider could not be obtained or the passed handle was invalid

Description

This function sets up an alarm, allowing the client to receive a callback from the driver when the timer counter reaches zero. Alarms can be one-shot or periodic. A periodic alarm will reload the timer and generate alarm until stopped. The alarm frequency is: DRV_TMR_CounterFrequencyGet() / divider;

Remarks

The divider value will be truncated to a 16 bit value if the timer is configured in 16 bit mode.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

divider value has to be within the timer divider range (see DRV_TMR_DividerSpecGet).

Example

```
//Do the initialization with 'mode' set to DRV_TMR_OPERATION_MODE_16_BIT

void setupTask ()
{
    DRV_HANDLE              tmrHandle;  // Returned from DRV_TMR_Open

    uint32_t myFreq = 1000; // 1KHz
    uint32_t clkFreq = DRV_TMR_CounterFrequencyGet(tmrHandle);   // timer running frequency

    // calculate the divider needed
    uint32_t divider = clkFreq / myFreq;

    // Start the alarm
    if(!DRV_TMR_AlarmRegister ( tmrHandle, divider, true, 0, CallBackFreq ))
    {
        // divider value could not be obtain;
        // handle the error
        //
    }
}
```

Parameters

Parameters	Description
handle	A valid handle, returned from DRV_TMR_Open
divider	The value to divide the timer clock source to obtain the required alarm frequency. • a 16 bit value if the timer is configured in 16 bit mode • a 32 bit value if the timer is configured in 32 bit mode
isPeriodic	Flag indicating whether the alarm should be one-shot or periodic.
context	A reference, call back function will be called with the same reference.
callBack	A call back function which will be called on time out.

Function

bool DRV_TMR_AlarmRegister

(

DRV_HANDLE handle,

uint32_t divider,

bool isPeriodic,

uintptr_t context,

DRV_TMR_CALLBACK callBack

)

d) Counter Control Functions

DRV_TMR_CounterFrequencyGet Function

Provides the Timer input frequency.

Implementation: Dynamic

File

drv_tmr.h

C

```
uint32_t DRV_TMR_CounterFrequencyGet(DRV_HANDLE handle);
```

Returns

32-bit value corresponding to the running frequency. If Timer clock source is external, then this function returns 0.

Description

This function provides the Timer input frequency. Input frequency is the clock to the Timer register and it is considering the prescaler divisor.

Remarks

On most processors, the Timer's base frequency is the same as the peripheral bus clock.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open
uint32_t   clkFreqHz;

clkFreqHz = DRV_TMR_CounterFrequencyGet ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_TMR_CounterFrequencyGet (DRV_HANDLE handle)

DRV_TMR_CounterValue16BitGet Function

Reads the 16-bit Timer's counter register.

Implementation: Dynamic

File

drv_tmr.h

C

```
uint16_t DRV_TMR_CounterValue16BitGet(DRV_HANDLE handle);
```

Returns

Timer period in 16-bit mode.

Description

This function returns the 16-bit Timer's value in the counter register. This is valid only if the 16-bit mode of the timer is selected. Otherwise use DRV_TMR_CounterValue32BitGet function.

Remarks

This function will be deprecated later, so avoid using it as much as possible. Use generic function "DRV_TMR_CounterValueGet" instead of this.

Preconditions

The DRV_TMR_Initialize function must have been called. Must have selected 16-Bit timer mode if mode selection is applicable.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
//Example to use timer for precision time measurement
//without configuring an alarm (interrupt based)
char appState = 0;
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open

switch ( appState )
{
    case 0:
        //Calculate and set the counter period
        DRV_TMR_CounterValue16BitSet ( tmrHandle, ( 0xFFFF - 0x1000 ) );

        //counter starts
        DRV_TMR_Start ( tmrHandle );

        //Trigger an application operation
        app_trigger_operation();

        //Check for time-out in the next state
        appState++;
    case 1:
        //Overflows and stops at 0 if no alarm is set
        if ( DRV_TMR_CounterValue16BitGet ( tmrHandle ) == 0 )
        {
            //Time-out
            return false;
        }
        else if ( app_operation_isComplete( ) )
        {
            //Operation is complete before time-out
            return true;
        }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint16_t DRV_TMR_CounterValue16BitGet (DRV_HANDLE handle)

DRV_TMR_CounterValue16BitSet Function

Updates the 16-bit Timer's counter register.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_CounterValue16BitSet(DRV_HANDLE handle, uint16_t counterPeriod);
```

Returns

None.

Description

This function updates the 16-bit Timer's value in the counter register. This is valid only if the 16-bit mode of the timer is selected('mode' in the INIT structure is set to DRV_TMR_OPERATION_MODE_16_BIT). Otherwise use DRV_TMR_CounterValue32BitSet function.

Remarks

This function will be deprecated later, so avoid using it as much as possible. Use generic function "DRV_TMR_CounterValueSet" instead of this.

Preconditions

The DRV_TMR_Initialize function must have been called. Must have selected 16-Bit timer mode if mode selection is applicable.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
//Example to use timer for precision time measurement
//without configuring an alarm (interrupt based)
char appState = 0;
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open

switch ( appState )
{
    case 0:
        //Calculate and set the counter period
        DRV_TMR_CounterValue16BitSet ( tmrHandle, ( 0xFFFF - 0x1000 ) );

        //counter starts
        DRV_TMR_Start ( tmrHandle );

        //Trigger an application operation
        app_trigger_operation();

        //Check for time-out in the next state
        appState++;
    case 1:
        //Overflows and stops at 0 if no alarm is set
        if ( DRV_TMR_CounterValue16BitGet ( tmrHandle ) == 0 )
        {
            //Time-out
            return false;
        }
        else if ( app_operation_isComplete( ) )
        {
            //Operation is complete before time-out
            return true;
        }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
counterPeriod	16-bit counter period value

Function

void DRV_TMR_CounterValue16BitSet
(
DRV_HANDLE handle,
uint16_t counterPeriod
)

DRV_TMR_CounterValue32BitGet Function

Reads the 32-bit Timer's counter register.

Implementation: Dynamic

File

drv_tmr.h

C

```
uint32_t DRV_TMR_CounterValue32BitGet(DRV_HANDLE handle);
```

Returns

32-Bit Counter value.

Description

This function returns the 32-bit Timer's value in the counter register. This is valid only if the 32-bit mode of the timer is selected Otherwise use DRV_TMR_CounterValue16BitGet function.

Remarks

In most of the devices only even numbered instances of timer supports 32-bit mode. This function will be deprecated later, so avoid using it as much as possible. Use generic function "DRV_TMR_CounterValueGet" instead of this.

Preconditions

The DRV_TMR_Initialize function must have been called. Must have selected 32-Bit timer mode.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
//Example to use timer for precision time measurement
//without configuring an alarm (interrupt based)
char appState = 0;
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open

switch ( appState )
{
    case 0:
        //Calculate and set the counter period
        DRV_TMR_CounterValue32BitSet ( tmrHandle, ( 0xFFFFFFFF - 0x23321000 ) );

        //counter starts
        DRV_TMR_Start ( tmrHandle );

        //Trigger an application operation
        app_trigger_operation();

        //Check for time-out in the next state
        appState++;
    case 1:
        //Overflows and stops at 0 if no alarm is set
        if ( DRV_TMR_CounterValue32BitGet ( tmrHandle ) == 0 )
        {
            //Time-out
            return false;
        }
        else if ( app_operation_isComplete( ) )
        {
            //Operation is complete before time-out
            return true;
        }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_TMR_CounterValue32BitGet (DRV_HANDLE handle)

DRV_TMR_CounterValue32BitSet Function

Updates the 32-bit Timer's counter register.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_CounterValue32BitSet(DRV_HANDLE handle, uint32_t counterPeriod);
```

Returns

None.

Description

This function updates the 32-bit Timer's value in the counter register. This is valid only if the 32-bit mode of the timer is selected Otherwise use DRV_TMR_CounterValue16BitSet function.

Remarks

In most of the devices only even numbered instances of timer supports 32-bit mode. This function will be deprecated later, so avoid using it as much as possible. Use generic function "DRV_TMR_CounterValueSet" instead of this.

Preconditions

The DRV_TMR_Initialize function must have been called. Must have selected 32-Bit timer mode.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
//Example to use timer for precision time measurement
//without configuring an alarm (interrupt based)
char appState = 0;
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open

switch ( appState )
{
    case 0:
        //Calculate and set the counter period
        DRV_TMR_CounterValue32BitSet ( tmrHandle, ( 0xFFFFFFFF - 0xFF343100 ) );

        //counter starts
        DRV_TMR_Start ( tmrHandle );

        //Trigger an application operation against which we want to use the time-out
        app_trigger_operation();

        //Check for time-out in the next state
        appState++;
    case 1:
        //Overflows and stops at 0 if no alarm is set
        if ( DRV_TMR_CounterValue32BitGet ( tmrHandle ) == 0 )
        {
            //Time-out
            return false;
        }
        else if ( app_operation_isComplete( ) )
        {
            //Operation is complete before time-out
            return true;
        }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
counterPeriod	32-bit counter period value

Function

void DRV_TMR_CounterValue32BitSet

(

DRV_HANDLE handle,

uint32_t counterPeriod

)

DRV_TMR_CounterClear Function

Clears the Timer's counter register.

Implementation: Static/Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_CounterClear(DRV_HANDLE handle);
```

Returns

None.

Description

This function clears the Timer's value in the counter register.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TMR_CounterClear (DRV_HANDLE handle)

DRV_TMR_CounterValueGet Function

Reads the Timer's counter register.

Implementation: Static/Dynamic

File

drv_tmr.h

C

```
uint32_t DRV_TMR_CounterValueGet(DRV_HANDLE handle);
```

Returns

Timer current period:

- a 16 bit value if the timer is configured in 16 bit mode
- a 32 bit value if the timer is configured in 32 bit mode

Description

This function returns the Timer's value in the counter register.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
//Example to use timer for precision time measurement
//without configuring an alarm (interrupt based)
char appState = 0;
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open

switch ( appState )
```

```
{
    case 0:
        //Calculate and set the counter period
        DRV_TMR_CounterValueSet ( tmrHandle, ( 0xFFFF - 0x1000 ) );

        //counter starts
        DRV_TMR_Start ( tmrHandle );

        //Trigger an application operation
        app_trigger_operation();

        //Check for time-out in the next state
        appState++;
    case 1:
        //Overflows and stops at 0 if no alarm is set
        if ( DRV_TMR_CounterValueGet ( tmrHandle ) == 0 )
        {
            //Time-out
            return false;
        }
        else if ( app_operation_isComplete( ) )
        {
            //Operation is complete before time-out
            return true;
        }
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint32_t DRV_TMR_CounterValueGet (DRV_HANDLE handle)

DRV_TMR_CounterValueSet Function

Updates the Timer's counter register.

Implementation: Static/Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_CounterValueSet(DRV_HANDLE handle, uint32_t counterPeriod);
```

Returns

None.

Description

This function updates the Timer's value in the counter register.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

counterPeriod	counter period value • a 16 bit value if the timer is configured in 16 bit mode • a 32 bit value if the timer is configured in 32 bit mode

Function

void DRV_TMR_CounterValueSet (DRV_HANDLE handle, uint32_t counterPeriod)

e) Miscellaneous Functions

DRV_TMR_GateModeClear Function

Enables the Gate mode.

Implementation: Dynamic

File

drv_tmr.h

C

```
bool DRV_TMR_GateModeClear(DRV_HANDLE handle);
```

Returns

- true - if the operation is successful
- false - either the handle is invalid or the gate mode is not supported

Description

This function enables the Gated mode of Timer. User can measure the duration of an external signal in this mode. Once the Gate mode is enabled, Timer will start on the raising edge of the external signal. It will keep counting until the next falling edge.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open

DRV_TMR_GateModeClear ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

bool DRV_TMR_GateModeClear (DRV_HANDLE handle)

DRV_TMR_PrescalerGet Function

This function gets the currently selected prescaler.

Implementation: Dynamic

File

drv_tmr.h

C

```
TMR_PRESCALE DRV_TMR_PrescalerGet(DRV_HANDLE handle);
```

Returns

Timer prescaler.

Description

This function gets the currently selected prescaler.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open
TMR_PRESCALE preScale;

preScale = DRV_TMR_PrescalerGet ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

TMR_PRESCALE DRV_TMR_PrescalerGet (DRV_HANDLE handle)

DRV_TMR_OperationModeGet Function

This function gets the currently selected operation mode.

Implementation: Dynamic

File

drv_tmr.h

C

```
DRV_TMR_OPERATION_MODE DRV_TMR_OperationModeGet(DRV_HANDLE handle);
```

Returns

A DRV_TMR_OPERATION_MODE value showing how the timer is currently configured. DRV_TMR_OPERATION_MODE_NONE is returned for an invalid client handle.

Description

This function gets the currently selected 16/32 bit operation mode.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open
DRV_TMR_OPERATION_MODE operMode;

operMode = DRV_TMR_OperationModeGet ( tmrHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

DRV_TMR_OPERATION_MODE DRV_TMR_OperationModeGet(DRV_HANDLE handle)

DRV_TMR_DividerRangeGet Function

Returns the Timer divider values.

Implementation: Dynamic

File

drv_tmr.h

C

```
DRV_TMR_OPERATION_MODE DRV_TMR_DividerRangeGet(DRV_HANDLE handle, DRV_TMR_DIVIDER_RANGE* pDivRange);
```

Returns

- A DRV_TMR_OPERATION_MODE value showing how the timer is currently configured. The pDivRange is updated with the supported range values.
- DRV_TMR_OPERATION_MODE_NONE for invalid client handle

Description

This function provides the Timer operating mode and divider range.

Remarks

None.

Preconditions

The DRV_TMR_Initialize function must have been called.

DRV_TMR_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE tmrHandle;  // Returned from DRV_TMR_Open
DRV_TMR_OPERATION_MODE  timerMode;
DRV_TMR_DIVIDER_RANGE   timerRange;

DRV_TMR_DividerRangeGet(handle, &timerRange);
uint32_t   clkFreqHz = DRV_TMR_CounterFrequencyGet ( tmrHandle );

uint32_t    maxFreqHz = clkFreqHz / timerRange.dividerMin;
uint32_t    minFreqHz = clkFreqHz / timerRange.dividerMax;
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
pDivRange	Address to store the timer divider range.

Function

```
DRV_TMR_OPERATION_MODE DRV_TMR_DividerRangeGet
(
    DRV_HANDLE handle,
    DRV_TMR_DIVIDER_RANGE* pDivRange
)
```

DRV_TMR_Tasks_ISR Function

Maintains the driver's state machine, processes the events and implements its ISR.

Implementation: Dynamic

File

drv_tmr.h

C

```
void DRV_TMR_Tasks_ISR(SYS_MODULE_OBJ object);
```

Returns

None

Description

This function is used to maintain the driver's internal state machine and processes the timer events in interrupt-driven implementations (DRV_TMR_INTERRUPT_MODE == true).

Remarks

This function is normally not called directly by an application. It is called by the timer driver raw ISR.

This function will execute in an ISR context and will never block or access any resources that may cause it to block.

This function will be deprecated later, so avoid using it as much as possible. Use generic function "DRV_TMR_Tasks" instead of this.

Preconditions

The DRV_TMR_Initialize function must have been called for the specified Timer driver instance.

Example

```
void __ISR(_TIMER_2_VECTOR, ipl4) _InterruptHandler_TMR2(void)
{
    DRV_TMR_Tasks_ISR(appDrvObjects.drvTmrObject);
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_TMR_Initialize)

Function

void DRV_TMR_Tasks_ISR (SYS_MODULE_OBJ object)

f) Data Types and Constants

DRV_TMR_CALLBACK Type

Pointer to a Timer driver callback function data type.

File

drv_tmr.h

C

```
typedef void (* DRV_TMR_CALLBACK)(uintptr_t context, uint32_t alarmCount);
```

Description

Timer Driver Callback Function Pointer

This data type defines a pointer to a Timer driver callback function.

Remarks

Useful only when timer alarm callback support is enabled by defining the DRV_TMR_ALARM_ENABLE configuration option.

DRV_TMR_INIT Structure

Defines the Timer driver initialization data.

File

drv_tmr.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  TMR_MODULE_ID tmrId;
  DRV_TMR_CLK_SOURCES clockSource;
  TMR_PRESCALE prescale;
```

```
  INT_SOURCE interruptSource;
  DRV_TMR_OPERATION_MODE mode;
  bool asyncWriteEnable;
} DRV_TMR_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization.
TMR_MODULE_ID tmrId;	Identifies timer hardware module (PLIB-level) ID
DRV_TMR_CLK_SOURCES clockSource;	Clock Source select.
TMR_PRESCALE prescale;	Prescaler Selection from the processor enumeration
INT_SOURCE interruptSource;	Interrupt Source for TMR module. If 'DRV_TMR_OPERATION_MODE_32_BIT' flag is selected the interrupt will be generated by the 2nd timer of the pair, the odd numbered one.
DRV_TMR_OPERATION_MODE mode;	Select 16/32 bit operation mode. 32 bit mode will combine two 16 bit timer modules to form a 32 bit one. This is usually only necessary for very long delays.
bool asyncWriteEnable;	Asynchronous write enable configuration. If true the asynchronous write is enabled. For timers that do not support this feature the value is ignored

Description

Timer Driver Initialize Data

This data type defines data required to initialize the Timer driver.

Remarks

Not all initialization features are available on all devices.

DRV_TMR_CLIENT_STATUS Enumeration

Identifies the client-specific status of the Timer driver

File

drv_tmr.h

C

```
typedef enum {
  DRV_TMR_CLIENT_STATUS_INVALID,
  DRV_TMR_CLIENT_STATUS_BUSY,
  DRV_TMR_CLIENT_STATUS_READY,
  DRV_TMR_CLIENT_STATUS_RUNNING
} DRV_TMR_CLIENT_STATUS;
```

Members

Members	Description
DRV_TMR_CLIENT_STATUS_INVALID	Driver is invalid (or unopened) state
DRV_TMR_CLIENT_STATUS_BUSY	An operation is currently in progress
DRV_TMR_CLIENT_STATUS_READY	Ready, no operations running
DRV_TMR_CLIENT_STATUS_RUNNING	Timer started and running, processing transactions

Description

Timer Driver Client Status

This enumeration identifies the client-specific status of the Timer driver.

Remarks

None.

DRV_TMR_CLK_SOURCES Enumeration

Lists the clock sources available for timer driver.

File

drv_tmr.h

C

```
typedef enum {
  DRV_TMR_CLKSOURCE_INTERNAL,
  DRV_TMR_CLKSOURCE_EXTERNAL_SYNCHRONOUS,
  DRV_TMR_CLKSOURCE_EXTERNAL_ASYNCHRONOUS,
  DRV_TMR_CLKSOURCE_SECONDARY_OSCILLATOR_SYNCHRONOUS,
  DRV_TMR_CLKSOURCE_SECONDARY_OSCILLATOR_ASYNCHRONOUS,
  DRV_TMR_CLKSOURCE_LPRC_OSCILLATOR_SYNCHRONOUS,
  DRV_TMR_CLKSOURCE_LPRC_OSCILLATOR_ASYNCHRONOUS
} DRV_TMR_CLK_SOURCES;
```

Members

Members	Description
DRV_TMR_CLKSOURCE_INTERNAL	Clock input to the timer module is internal(Peripheral Clock)
DRV_TMR_CLKSOURCE_EXTERNAL_SYNCHRONOUS	Clock input to the timer module is from T1CK pin (external) with clock synchronization enabled
DRV_TMR_CLKSOURCE_EXTERNAL_ASYNCHRONOUS	Clock input to the timer module is from T1CK pin (external) with clock synchronization disabled
DRV_TMR_CLKSOURCE_SECONDARY_OSCILLATOR_SYNCHRONOUS	Clock input to the timer module is from secondary oscillator (external) with clock synchronization enabled
DRV_TMR_CLKSOURCE_SECONDARY_OSCILLATOR_ASYNCHRONOUS	Clock input to the timer module is from secondary oscillator (external) with clock synchronization disabled
DRV_TMR_CLKSOURCE_LPRC_OSCILLATOR_SYNCHRONOUS	Clock input to the timer module is from LPRC with clock synchronization enabled
DRV_TMR_CLKSOURCE_LPRC_OSCILLATOR_ASYNCHRONOUS	Clock input to the timer module is from LPRC with clock synchronization disabled

Description

Timer Driver Clock sources

This enumeration lists all the available clock sources for the timer hardware.

Remarks

Not all modes are available on all devices.

'Synchronization' may not be applicable for all the instances of the timer. The driver discards the Synchronization mode selected if it is not applicable for the selected hardware.

DRV_TMR_DIVIDER_RANGE Structure

This data structure specifies the divider values that can be obtained by the timer module.

File

drv_tmr.h

C

```
typedef struct {
  uint32_t dividerMin;
  uint32_t dividerMax;
  uint32_t dividerStep;
} DRV_TMR_DIVIDER_RANGE;
```

Members

Members	Description
uint32_t dividerMin;	The minimum divider value that the timer module can obtain
uint32_t dividerMax;	The maximum divider value that the timer module can obtain
uint32_t dividerStep;	The divider step value, between 2 divider values Should be 1 for most timers

Description

Timer Driver divider operating specification

This data structure specifies the divider values that can be obtained by the timer hardware.

Remarks

None.

DRV_TMR_OPERATION_MODE Enumeration

Lists the operation modes available for timer driver.

File

drv_tmr.h

C

```
typedef enum {
  DRV_TMR_OPERATION_MODE_NONE,
  DRV_TMR_OPERATION_MODE_16_BIT,
  DRV_TMR_OPERATION_MODE_32_BIT
} DRV_TMR_OPERATION_MODE;
```

Members

Members	Description
DRV_TMR_OPERATION_MODE_NONE	The timer module operating mode none/invalid
DRV_TMR_OPERATION_MODE_16_BIT	The timer module operates in 16 bit mode
DRV_TMR_OPERATION_MODE_32_BIT	The timer module operates in 32 bit mode This will combine two 16 bit timer modules

Description

Timer Driver Operation mode

This enumeration lists all the available operation modes that are valid for the timer hardware.

Remarks

Not all modes are available on all devices.

DRV_TMR_INDEX_COUNT Macro

Number of valid Timer driver indices.

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_COUNT TMR_NUMBER_OF_MODULES
```

Description

Timer Driver Module Index Count

This constant identifies Timer driver index definitions.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is device-specific.

DRV_TMR_INDEX_0 Macro

Timer driver index definitions

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_0 0
```

Description

Timer Driver Module Index Numbers

These constants provide Timer driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_TMR_Initialize and DRV_TMR_Open functions to identify the driver instance in use.

DRV_TMR_INDEX_1 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_1 1
```

Description

This is macro DRV_TMR_INDEX_1.

DRV_TMR_INDEX_2 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_2 2
```

Description

This is macro DRV_TMR_INDEX_2.

DRV_TMR_INDEX_3 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_3 3
```

Description

This is macro DRV_TMR_INDEX_3.

DRV_TMR_INDEX_4 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_4 4
```

Description

This is macro DRV_TMR_INDEX_4.

DRV_TMR_INDEX_5 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_5 5
```

Description

This is macro DRV_TMR_INDEX_5.

DRV_TMR_INDEX_6 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_6 6
```

Description

This is macro DRV_TMR_INDEX_6.

DRV_TMR_INDEX_7 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_7 7
```

Description

This is macro DRV_TMR_INDEX_7.

DRV_TMR_INDEX_8 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_8 8
```

Description

This is macro DRV_TMR_INDEX_8.

DRV_TMR_INDEX_9 Macro

File

drv_tmr.h

C

```
#define DRV_TMR_INDEX_9 9
```

Description

This is macro DRV_TMR_INDEX_9.

Files

Files

Name	Description
drv_tmr.h	Timer device driver interface header file.
drv_tmr_config_template.h	Timer driver configuration definitions for the template version.

Description

This section lists the source and header files used by the Timer Driver Library.

drv_tmr.h

Timer device driver interface header file.

Enumerations

	Name	Description
	DRV_TMR_CLIENT_STATUS	Identifies the client-specific status of the Timer driver
	DRV_TMR_CLK_SOURCES	Lists the clock sources available for timer driver.
	DRV_TMR_OPERATION_MODE	Lists the operation modes available for timer driver.

Functions

	Name	Description
	DRV_TMR_Alarm16BitDeregister	Removes a previously set alarm. **Implementation:** Dynamic
	DRV_TMR_Alarm16BitRegister	Sets up an alarm. **Implementation:** Dynamic
	DRV_TMR_Alarm32BitDeregister	Removes a previously set alarm. **Implementation:** Dynamic
	DRV_TMR_Alarm32BitRegister	Sets up an alarm. **Implementation:** Dynamic
	DRV_TMR_AlarmDeregister	Removes a previously set alarm. **Implementation:** Dynamic
	DRV_TMR_AlarmDisable	Disables an alarm signal. **Implementation:** Dynamic
	DRV_TMR_AlarmEnable	Re-enables an alarm signal. **Implementation:** Dynamic
	DRV_TMR_AlarmHasElapsed	Provides the status of Timer's period elapse. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriod16BitGet	Provides the 16-bit Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriod16BitSet	Updates the 16-bit Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriod32BitGet	Provides the 32-bit Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriod32BitSet	Updates the 32-bit Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriodGet	Provides the Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmPeriodSet	Updates the Timer's period. **Implementation:** Dynamic
	DRV_TMR_AlarmRegister	Sets up an alarm. **Implementation:** Dynamic
	DRV_TMR_ClientStatus	Gets the status of the client operation. **Implementation:** Dynamic
	DRV_TMR_ClockSet	Sets the timers clock by selecting the source and prescaler. **Implementation:** Dynamic
	DRV_TMR_Close	Closes an opened instance of the Timer driver. **Implementation:** Dynamic
	DRV_TMR_CounterClear	Clears the Timer's counter register. **Implementation:** Static/Dynamic
	DRV_TMR_CounterFrequencyGet	Provides the Timer input frequency. **Implementation:** Dynamic
	DRV_TMR_CounterValue16BitGet	Reads the 16-bit Timer's counter register. **Implementation:** Dynamic
	DRV_TMR_CounterValue16BitSet	Updates the 16-bit Timer's counter register. **Implementation:** Dynamic
	DRV_TMR_CounterValue32BitGet	Reads the 32-bit Timer's counter register. **Implementation:** Dynamic
	DRV_TMR_CounterValue32BitSet	Updates the 32-bit Timer's counter register. **Implementation:** Dynamic

	DRV_TMR_CounterValueGet	Reads the Timer's counter register. **Implementation:** Static/Dynamic
	DRV_TMR_CounterValueSet	Updates the Timer's counter register. **Implementation:** Static/Dynamic
	DRV_TMR_Deinitialize	Deinitializes the specified instance of the Timer driver. **Implementation:** Dynamic
	DRV_TMR_DividerRangeGet	Returns the Timer divider values. **Implementation:** Dynamic
	DRV_TMR_GateModeClear	Enables the Gate mode. **Implementation:** Dynamic
	DRV_TMR_GateModeSet	Enables the Gate mode. **Implementation:** Dynamic
	DRV_TMR_Initialize	Initializes the Timer driver. **Implementation:** Static/Dynamic
	DRV_TMR_Open	Opens the specified Timer driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_TMR_OperationModeGet	This function gets the currently selected operation mode. **Implementation:** Dynamic
	DRV_TMR_PrescalerGet	This function gets the currently selected prescaler. **Implementation:** Dynamic
	DRV_TMR_Start	Starts the Timer counting. **Implementation:** Static/Dynamic
	DRV_TMR_Status	Provides the current status of the Timer driver. **Implementation:** Dynamic
	DRV_TMR_Stop	Stops the Timer from counting. **Implementation:** Static/Dynamic
	DRV_TMR_Tasks	Maintains the driver's state machine. **Implementation:** Dynamic
	DRV_TMR_Tasks_ISR	Maintains the driver's state machine, processes the events and implements its ISR. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_TMR_INDEX_0	Timer driver index definitions
	DRV_TMR_INDEX_1	This is macro DRV_TMR_INDEX_1.
	DRV_TMR_INDEX_2	This is macro DRV_TMR_INDEX_2.
	DRV_TMR_INDEX_3	This is macro DRV_TMR_INDEX_3.
	DRV_TMR_INDEX_4	This is macro DRV_TMR_INDEX_4.
	DRV_TMR_INDEX_5	This is macro DRV_TMR_INDEX_5.
	DRV_TMR_INDEX_6	This is macro DRV_TMR_INDEX_6.
	DRV_TMR_INDEX_7	This is macro DRV_TMR_INDEX_7.
	DRV_TMR_INDEX_8	This is macro DRV_TMR_INDEX_8.
	DRV_TMR_INDEX_9	This is macro DRV_TMR_INDEX_9.
	DRV_TMR_INDEX_COUNT	Number of valid Timer driver indices.

Structures

	Name	Description
	DRV_TMR_DIVIDER_RANGE	This data structure specifies the divider values that can be obtained by the timer module.
	DRV_TMR_INIT	Defines the Timer driver initialization data.

Types

	Name	Description
	DRV_TMR_CALLBACK	Pointer to a Timer driver callback function data type.

Description

Timer Device Driver Interface Definition

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Timer device

driver.

File Name

drv_tmr.h

Company

Microchip Technology Inc.

drv_tmr_config_template.h

Timer driver configuration definitions for the template version.

Macros

	Name	Description
	DRV_TMR_ASYNC_WRITE_ENABLE	Controls Asynchronous Write mode of the Timer.
	DRV_TMR_CLIENTS_NUMBER	Sets up the maximum number of clients that can be supported by an instance of the dynamic driver.
	DRV_TMR_CLOCK_PRESCALER	Sets the default timer driver clock prescaler.
	DRV_TMR_CLOCK_SOURCE	Sets the default timer driver clock source.
	DRV_TMR_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported by the dynamic driver.
	DRV_TMR_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_TMR_INTERRUPT_SOURCE	Sets the default timer driver clock interrupt source
	DRV_TMR_MODE	Sets the default timer driver clock operating mode.
	DRV_TMR_MODULE_ID	Sets the default timer module ID to be used by the timer driver.
	DRV_TMR_MODULE_INIT	Sets the default module init value for the timer driver.

Description

Timer Driver Configuration Definitions for the Template Version

These definitions set up the driver for the default mode of operation of the driver.

File Name

drv_tmr_config_template.h

Company

Microchip Technology Inc.

Touch Driver Libraries Help

This section describes the Touch Driver Libraries.

10-bit ADC Touch Driver Library

This topic describes the 10-bit ADC Touch Driver Library.

Introduction

This library provides an interface to manage the 10-bit ADC Touch Driver module on the Microchip family of microcontrollers in different modes of operation.

Description

The MPLAB Harmony 10-bit ADC Touch Driver provides a high-level interface to the 10-bit ADC touch device. This driver provides application routines to read non-gestural single-point touch input data from the touch screen. The 10-bit ADC touch device can notify the availability of touch input data through external interrupt. The 10-bit ADC Touch Driver allows the application to map a controller pin as an external interrupt pin.

Using the Library

This topic describes the basic architecture of the 10-bit ADC Touch Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_adc10bit.h`

The interface to the 10-bit ADC Touch Driver library is defined in the `drv_adc10bit.h` header file. Any C language source (`.c`) file that uses the ADC 10-bit Touch Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the 10-bit ADC Touch Driver module.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, open, close, task, and status functions.

Configuring the Library

Macros

	Name	Description
	DRV_ADC10BIT_CALIBRATION_DELAY	Defines the calibration delay.
	DRV_ADC10BIT_CALIBRATION_INSET	Defines the calibration inset.
	DRV_ADC10BIT_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ADC10BIT_INDEX	ADC10BIT static index selection.
	DRV_ADC10BIT_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_ADC10BIT_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_ADC10BIT_SAMPLE_POINTS	Defines the sample points.
	DRV_ADC10BIT_TOUCH_DIAMETER	Defines the touch diameter.

Description

The configuration of the 10-bit ADC Touch Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the ADC 10-bit Touch Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the 10-bit ADC Touch Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_ADC10BIT_CALIBRATION_DELAY Macro

Defines the calibration delay.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_CALIBRATION_DELAY 300
```

Description

ADC10BIT Calibration Delay

This macro enables the delay between calibration touch points.

Remarks

None.

DRV_ADC10BIT_CALIBRATION_INSET Macro

Defines the calibration inset.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_CALIBRATION_INSET 25
```

Description

ADC10BIT Calibration Inset

This macro defines the calibration inset.

Remarks

None.

DRV_ADC10BIT_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_CLIENTS_NUMBER 1
```

Description

ADC10BIT client number

This macro selects the maximum number of clients.

This definition selected the maximum number of clients that the ADC10BIT driver can support at run-time.

Remarks

None.

DRV_ADC10BIT_INDEX Macro

ADC10BIT static index selection.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_INDEX DRV_ADC10BIT_INDEX_0
```

Description

ADC10BIT Static Index Selection

This macro specifies the static index selection for the driver object reference.

Remarks

This index is required to make a reference to the driver object.

DRV_ADC10BIT_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_INSTANCES_NUMBER 1
```

Description

ADC10BIT hardware instance configuration

This macro sets up the maximum number of hardware instances that can be supported.

Remarks

None.

DRV_ADC10BIT_INTERRUPT_MODE Macro

Controls operation of the driver in the interrupt or polled mode.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_INTERRUPT_MODE false
```

Description

ADC10BIT Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of ADC10BIT operation is desired
- false - Select if polling mode of ADC10BIT operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

DRV_ADC10BIT_SAMPLE_POINTS Macro

Defines the sample points.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_SAMPLE_POINTS 4
```

Description

ADC10BIT Sample Points

This macro defines the sample points.

Remarks

None.

DRV_ADC10BIT_TOUCH_DIAMETER Macro

Defines the touch diameter.

File

drv_adc10bit_config_template.h

C

```
#define DRV_ADC10BIT_TOUCH_DIAMETER 10
```

Description

ADC10BIT Touch Diameter

This macro defines the touch diameter.

Remarks

None.

Building the Library

This section lists the files that are available in the 10-bit ADC Touch Driver Library.

Description

This section list the files that are available in the `\src` folder of the 10-bit ADC Touch Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/touch/adc10bit`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_adc10bit.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/drv_adc10bit.c`	Basic 10-bit ADC Touch Driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The 10-bit ADC Touch Driver Library depends on the following modules:

- Interrupt System Service Library
- Ports System Service Library
- Touch System Service Library
- I2C Driver Library

Library Interface

a) System Functions

	Name	Description
	DRV_TOUCH_ADC10BIT_CalibrationSet	Loads calibration parameters from Non-volatile Memory.
	DRV_TOUCH_ADC10BIT_Close	Closes an opened instance of the 10-bit ADC Driver.
	DRV_TOUCH_ADC10BIT_Deinitialize	Deinitializes the specified instance of the ADC10BIT driver module.
	DRV_TOUCH_ADC10BIT_Initialize	Initializes the 10-bit ADC Driver instance for the specified driver index
	DRV_TOUCH_ADC10BIT_Open	Opens the specified ADC10BIT driver instance and returns a handle to it.
	DRV_TOUCH_ADC10BIT_Status	Provides the current status of the ADC10BIT driver module.
	DRV_TOUCH_ADC10BIT_Tasks	Maintains the driver's state machine and implements its ISR.
	DRV_TOUCH_ADC10BIT_TouchGetRawX	Returns raw x coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchGetRawY	Returns raw y coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchGetX	Returns x coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchStoreCalibration	Stores calibration parameters into Non-volatile Memory.
	DRV_TOUCH_ADC10BIT_PositionDetect	None.
	DRV_TOUCH_ADC10BIT_TouchGetY	Returns y coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchDataRead	Notifies the driver that the current touch data has been read

	DRV_TOUCH_ADC10BIT_TouchStatus	Returns the status of the current touch input.

b) Data Types and Constants

	Name	Description
	_DRV_TOUCH_ADC10BIT_CLIENT_DATA	Defines the data that can be changed per client.
	_DRV_TOUCH_ADC10BIT_INIT	Defines the data required to initialize or reinitialize the 10-bit ADC Driver.
	DRV_ADC10BIT_MODULE_ID	This is type DRV_ADC10BIT_MODULE_ID.
	DRV_TOUCH_ADC10BIT_CLIENT_DATA	Defines the data that can be changed per client.
	DRV_TOUCH_ADC10BIT_HANDLE	Driver handle.
	DRV_TOUCH_ADC10BIT_INIT	Defines the data required to initialize or reinitialize the 10-bit ADC Driver.
	DRV_TOUCH_ADC10BIT_HANDLE_INVALID	Definition of an invalid handle.
	DRV_TOUCH_ADC10BIT_INDEX_0	ADC10BIT driver index definitions.
	DRV_TOUCH_ADC10BIT_INDEX_1	This is macro DRV_TOUCH_ADC10BIT_INDEX_1.
	DRV_TOUCH_ADC10BIT_INDEX_COUNT	Number of valid ADC10BIT driver indices.

Description

This section describes the API functions of the 10-bit ADC Touch Driver library.

Refer to each section for a detailed description.

a) System Functions

DRV_TOUCH_ADC10BIT_CalibrationSet Function

Loads calibration parameters from Non-volatile Memory.

File

drv_adc10bit.h

C

```
void DRV_TOUCH_ADC10BIT_CalibrationSet(DRV_TOUCH_SAMPLE_POINTS * samplePoints);
```

Returns

None.

Description

This function loads calibration parameters from Non-volatile Memory.

Preconditions

The NVM initialization function must be called before calling this function.

Function

void DRV_TOUCH_ADC10BIT_TouchLoadCalibration(void)

DRV_TOUCH_ADC10BIT_Close Function

Closes an opened instance of the 10-bit ADC Driver.

File

drv_adc10bit.h

C

```
void DRV_TOUCH_ADC10BIT_Close(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the 10-bit ADC Driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_TOUCH_ADC10BIT_Open before the caller may use the driver again. This function is thread safe in a RTOS application.

Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

DRV_TOUCH_ADC10BIT_Initialize must have been called for the specified ADC10BIT driver instance.

DRV_TOUCH_ADC10BIT_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TOUCH_ADC10BIT_Open

DRV_TOUCH_ADC10BIT_Close ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TOUCH_ADC10BIT_Close (DRV_HANDLE handle)

DRV_TOUCH_ADC10BIT_Deinitialize Function

Deinitializes the specified instance of the ADC10BIT driver module.

File

drv_adc10bit.h

C

```
void DRV_TOUCH_ADC10BIT_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the specified instance of the 10-bit ADC Driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_TOUCH_ADC10BIT_Status operation. The system has to use DRV_TOUCH_ADC10BIT_Status to determine when the module is in the ready state.

Preconditions

DRV_TOUCH_ADC10BIT_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_TOUCH_ADC10BIT_Initialize
SYS_STATUS          status;

DRV_TOUCH_ADC10BIT_Deinitialize ( object );

status = DRV_TOUCH_ADC10BIT_Status( object );
if( SYS_MODULE_UNINITIALIZED == status )
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TOUCH_ADC10BIT_Initialize

Function

void DRV_TOUCH_ADC10BIT_Deinitialize (SYS_MODULE_OBJ object)

DRV_TOUCH_ADC10BIT_Initialize Function

Initializes the 10-bit ADC Driver instance for the specified driver index

File

drv_adc10bit.h

C

```
SYS_MODULE_OBJ DRV_TOUCH_ADC10BIT_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const
init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the 10-bit ADC Driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the 'init' parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the 10-bit ADC Driver module ID. For example, driver instance 0 can be assigned to ADC10BIT2. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_TOUCH_ADC10BIT_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other ADC10BIT routine is called.

This routine should only be called once during system initialization unless DRV_TOUCH_ADC10BIT_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

None.

Example

```
DRV_TOUCH_ADC10BIT_INIT        init;
SYS_MODULE_OBJ      objectHandle;

// Populate the ADC10BIT initialization structure
init.spiId              = ADC10BIT_ID_1;

objectHandle = DRV_TOUCH_ADC10BIT_Initialize(DRV_TOUCH_ADC10BIT_INDEX_1, (SYS_MODULE_INIT*)usartInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized. Please note this is not the 10-bit ADC Driver ID. The hardware 10-bit ADC Driver ID is set in the initialization structure. This is the index of the driver index to use.
init	Pointer to a data structure containing any data necessary to initialize the driver. If this pointer is NULL, the driver uses the static initialization override macros for each member of the initialization data structure.

Function

SYS_MODULE_OBJ DRV_TOUCH_ADC10BIT_Initialize(const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init)

DRV_TOUCH_ADC10BIT_Open Function

Opens the specified ADC10BIT driver instance and returns a handle to it.

File

drv_adc10bit.h

C

```
DRV_HANDLE DRV_TOUCH_ADC10BIT_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. An error can occur when the following is true:

- if the number of client objects allocated via DRV_TOUCH_ADC10BIT_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid

Description

This function opens the specified USART driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options additionally affect the behavior of the DRV_USART_Read() and DRV_USART_Write() functions. If the ioIntent is DRV_IO_INTENT_NONBLOCKING, then these function will not block even if the required amount of data could not be processed. If the ioIntent is DRV_IO_INTENT_BLOCKING, these functions will block until the required amount of data is processed.

If ioIntent is DRV_IO_INTENT_READ, the client will only be read from the driver. If ioIntent is DRV_IO_INTENT_WRITE, the client will only be able to write to the driver. If the ioIntent in DRV_IO_INTENT_READWRITE, the client will be able to do both, read and write.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_TOUCH_ADC10BIT_Close routine is called. This routine will NEVER block waiting for hardware. If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

DRV_TOUCH_ADC10BIT_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_TOUCH_ADC10BIT_Open( DRV_TOUCH_ADC10BIT_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

if( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Index of the driver initialized with DRV_TOUCH_ADC10BIT_Initialize. Please note this is not the SPI id.
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Function

DRV_HANDLE DRV_TOUCH_ADC10BIT_Open (const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT intent)

DRV_TOUCH_ADC10BIT_Status Function

Provides the current status of the ADC10BIT driver module.

File

drv_adc10bit.h

C

```
SYS_STATUS DRV_TOUCH_ADC10BIT_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Description

This function provides the current status of the ADC10BIT driver module.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_MODULE_UNINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

This function can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

DRV_TOUCH_ADC10BIT_Initialize must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_TOUCH_ADC10BIT_Initialize
SYS_STATUS          status;

status = DRV_TOUCH_ADC10BIT_Status( object );
if( SYS_STATUS_READY != status )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TOUCH_ADC10BIT_Initialize

Function

SYS_STATUS DRV_TOUCH_ADC10BIT_Status (SYS_MODULE_OBJ object)

DRV_TOUCH_ADC10BIT_Tasks Function

Maintains the driver's state machine and implements its ISR.

File

drv_adc10bit.h

C

```
void DRV_TOUCH_ADC10BIT_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal state machine and implement its transmit ISR for interrupt-driven implementations. In polling mode, this function should be called from the SYS_Tasks function. In Interrupt mode, this function should be called in the transmit interrupt service routine of the USART that is associated with this USART driver hardware instance.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR. This function may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

DRV_TOUCH_ADC10BIT_Initialize must have been called for the specified 10-bit ADC Driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_TOUCH_ADC10BIT_Initialize

while( true )
{
    DRV_TOUCH_ADC10BIT_Tasks ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_TOUCH_ADC10BIT_Initialize)

Function

void DRV_TOUCH_ADC10BIT_Tasks (SYS_MODULE_OBJ object);

DRV_TOUCH_ADC10BIT_TouchGetRawX Function

Returns raw x coordinate status when the touch screen is pressed.

File

drv_adc10bit.h

C

```
short DRV_TOUCH_ADC10BIT_TouchGetRawX();
```

Returns

- raw x coordinate - Indicates the touch screen was pressed
- -1 - Indicates the touch screen was not pressed

Description

This function returns the raw x coordinate status when the touch screen is pressed.

Remarks

None.

Preconditions

None.

Function

short DRV_TOUCH_ADC10BIT_TouchGetRawX()

DRV_TOUCH_ADC10BIT_TouchGetRawY Function

Returns raw y coordinate status when the touch screen is pressed.

File

drv_adc10bit.h

C

```
short DRV_TOUCH_ADC10BIT_TouchGetRawY();
```

Returns

- raw y coordinate - Indicates the touch screen was pressed
- -1 - Indicates the touch screen was not pressed

Description

This function returns the raw y coordinate status when the touch screen is pressed.

Remarks

None.

Preconditions

None.

Function

short DRV_TOUCH_ADC10BIT_TouchGetRawY()

DRV_TOUCH_ADC10BIT_TouchGetX Function

Returns x coordinate status when the touch screen is pressed.

File

drv_adc10bit.h

C

```
short DRV_TOUCH_ADC10BIT_TouchGetX(uint8_t touchNumber);
```

Returns

- x coordinate - Indicates the touch screen was pressed
- -1 - Indicates the touch screen was not pressed

Description

This function returns the x coordinate status when the touch screen is pressed.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
touchNumber	touch input index.

Function

short DRV_TOUCH_ADC10BIT_TouchGetX(uint8_t touchNumber)

DRV_TOUCH_ADC10BIT_TouchStoreCalibration Function

Stores calibration parameters into Non-volatile Memory.

File

drv_adc10bit.h

C

```
void DRV_TOUCH_ADC10BIT_TouchStoreCalibration();
```

Returns

None.

Description

This function stores calibration parameters into Non-volatile Memory.

Remarks

This API is deprecated and its funcationality is handled via SYSTEM_INITIALIZATION

Preconditions

The NVM initialization function must be called before calling this function.

Function

void DRV_TOUCH_ADC10BIT_TouchStoreCalibration(void)

DRV_TOUCH_ADC10BIT_PositionDetect Function

None.

File

drv_adc10bit.h

C

```
short DRV_TOUCH_ADC10BIT_PositionDetect();
```

Returns

None.

Description

None.

Preconditions

None.

Function

void DRV_TOUCH_ADC10BIT_TouchLoadCalibration(void)

DRV_TOUCH_ADC10BIT_TouchGetY Function

Returns y coordinate status when the touch screen is pressed.

File

drv_adc10bit.h

C

```
short DRV_TOUCH_ADC10BIT_TouchGetY(uint8_t touchNumber);
```

Returns

- y coordinate - Indicates the touch screen was pressed
- -1 - Indicates the touch screen was not pressed

Description

This function returns the y coordinate status when the touch screen is pressed.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
handle	driver client handle.
touchNumber	touch input index.

Function

short DRV_TOUCH_ADC10BIT_TouchGetY(DRV_HANDLE handle, uint8_t touchNumber)

DRV_TOUCH_ADC10BIT_TouchDataRead Function

Notifies the driver that the current touch data has been read

File

drv_adc10bit.h

C

```
void DRV_TOUCH_ADC10BIT_TouchDataRead(const SYS_MODULE_INDEX index);
```

Returns

None.

Description

Notifies the driver that the current touch data has been read

Function

void DRV_TOUCH_ADC10BIT_TouchDataRead(const SYS_MODULE_INDEX index)

DRV_TOUCH_ADC10BIT_TouchStatus Function

Returns the status of the current touch input.

File

drv_adc10bit.h

C

```
DRV_TOUCH_POSITION_STATUS DRV_TOUCH_ADC10BIT_TouchStatus(const SYS_MODULE_INDEX index);
```

Returns

It returns the status of the current touch input.

Description

It returns the status of the current touch input.

Function

DRV_TOUCH_POSITION_SINGLE DRV_TOUCH_ADC10BIT_TouchStatus(const SYS_MODULE_INDEX index)

b) Data Types and Constants

DRV_ADC10BIT_MODULE_ID Enumeration

File

drv_adc10bit.h

C

```
typedef enum {
  ADC10BIT_ID_1 = 0,
  ADC10BIT_NUMBER_OF_MODULES
} DRV_ADC10BIT_MODULE_ID;
```

Description

This is type DRV_ADC10BIT_MODULE_ID.

DRV_TOUCH_ADC10BIT_CLIENT_DATA Structure

Defines the data that can be changed per client.

File

drv_adc10bit.h

C

```
typedef struct _DRV_TOUCH_ADC10BIT_CLIENT_DATA {
} DRV_TOUCH_ADC10BIT_CLIENT_DATA;
```

Description

Macro: ADC10BIT Driver Client Specific Configuration

This data type defines the data can be configured per client. This data can be per client, and overrides the configuration data contained inside of DRV_TOUCH_ADC10BIT_INIT.

Remarks

None.

DRV_TOUCH_ADC10BIT_HANDLE Type

Driver handle.

File

drv_adc10bit.h

C

```
typedef uintptr_t DRV_TOUCH_ADC10BIT_HANDLE;
```

Description

Macro: ADC10BIT Driver Handle

Touch screen controller interfacing with the 10-bit Analog-to-Digital (ADC) converter device.

Remarks

None

DRV_TOUCH_ADC10BIT_INIT Structure

Defines the data required to initialize or reinitialize the 10-bit ADC Driver.

File

drv_adc10bit.h

C

```
typedef struct _DRV_TOUCH_ADC10BIT_INIT {
  SYS_MODULE_INIT moduleInit;
  DRV_ADC10BIT_MODULE_ID adc10bitId;
} DRV_TOUCH_ADC10BIT_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
DRV_ADC10BIT_MODULE_ID adc10bitId;	Identifies peripheral (PLIB-level) ID

Description

Macro: ADC10BIT Driver Initialization Data

This data type defines the data required to initialize or reinitialize the 10-bit ADC Driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

DRV_TOUCH_ADC10BIT_HANDLE_INVALID Macro

Definition of an invalid handle.

File

drv_adc10bit.h

C

```
#define DRV_TOUCH_ADC10BIT_HANDLE_INVALID ((DRV_TOUCH_ADC10BIT_HANDLE)(-1))
```

Description

Macro: ADC10BIT Driver Invalid Handle

This is the definition of an invalid handle. An invalid handle is returned by DRV_ADC10BIT_RawRead and DRV_ADC10BIT_RawRead functions if the request was not successful.

Remarks

None.

DRV_TOUCH_ADC10BIT_INDEX_0 Macro

ADC10BIT driver index definitions.

File

drv_adc10bit.h

C

```
#define DRV_TOUCH_ADC10BIT_INDEX_0 0
```

Description

Macro: ADC10BIT Driver Module Index Numbers

These constants provide the 10-bit ADC Driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the DRV_ADC10BIT_Initialize and DRV_ADC10BIT_Open functions to identify the driver instance in use.

DRV_TOUCH_ADC10BIT_INDEX_1 Macro

File

drv_adc10bit.h

C

```
#define DRV_TOUCH_ADC10BIT_INDEX_1 1
```

Description

This is macro DRV_TOUCH_ADC10BIT_INDEX_1.

DRV_TOUCH_ADC10BIT_INDEX_COUNT Macro

Number of valid ADC10BIT driver indices.

File

drv_adc10bit.h

C

```
#define DRV_TOUCH_ADC10BIT_INDEX_COUNT 2
```

Description

Macro: ADC10BIT Driver Module Index Count

This constant identifies the number of valid 10-bit ADC Driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific header files defined as part of the peripheral libraries.

Files

Files

Name	Description
drv_adc10bit.h	10-bit ADC Touch Driver interface definitions
drv_adc10bit_config_template.h	10-bit ADC Touch Driver configuration template.

Description

This section lists the source and header files used by the 10-bit ADC Touch Driver Library.

drv_adc10bit.h

10-bit ADC Touch Driver interface definitions

Enumerations

	Name	Description
	DRV_ADC10BIT_MODULE_ID	This is type DRV_ADC10BIT_MODULE_ID.

Functions

	Name	Description
	DRV_TOUCH_ADC10BIT_CalibrationSet	Loads calibration parameters from Non-volatile Memory.
	DRV_TOUCH_ADC10BIT_Close	Closes an opened instance of the 10-bit ADC Driver.
	DRV_TOUCH_ADC10BIT_Deinitialize	Deinitializes the specified instance of the ADC10BIT driver module.
	DRV_TOUCH_ADC10BIT_Initialize	Initializes the 10-bit ADC Driver instance for the specified driver index
	DRV_TOUCH_ADC10BIT_Open	Opens the specified ADC10BIT driver instance and returns a handle to it.
	DRV_TOUCH_ADC10BIT_PositionDetect	None.
	DRV_TOUCH_ADC10BIT_Status	Provides the current status of the ADC10BIT driver module.
	DRV_TOUCH_ADC10BIT_Tasks	Maintains the driver's state machine and implements its ISR.
	DRV_TOUCH_ADC10BIT_TouchDataRead	Notifies the driver that the current touch data has been read
	DRV_TOUCH_ADC10BIT_TouchGetRawX	Returns raw x coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchGetRawY	Returns raw y coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchGetX	Returns x coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchGetY	Returns y coordinate status when the touch screen is pressed.
	DRV_TOUCH_ADC10BIT_TouchStatus	Returns the status of the current touch input.
	DRV_TOUCH_ADC10BIT_TouchStoreCalibration	Stores calibration parameters into Non-volatile Memory.

Macros

	Name	Description
	DRV_TOUCH_ADC10BIT_HANDLE_INVALID	Definition of an invalid handle.
	DRV_TOUCH_ADC10BIT_INDEX_0	ADC10BIT driver index definitions.
	DRV_TOUCH_ADC10BIT_INDEX_1	This is macro DRV_TOUCH_ADC10BIT_INDEX_1.
	DRV_TOUCH_ADC10BIT_INDEX_COUNT	Number of valid ADC10BIT driver indices.

Structures

	Name	Description
	_DRV_TOUCH_ADC10BIT_CLIENT_DATA	Defines the data that can be changed per client.
	_DRV_TOUCH_ADC10BIT_INIT	Defines the data required to initialize or reinitialize the 10-bit ADC Driver.

	DRV_TOUCH_ADC10BIT_CLIENT_DATA	Defines the data that can be changed per client.
	DRV_TOUCH_ADC10BIT_INIT	Defines the data required to initialize or reinitialize the 10-bit ADC Driver.

Types

	Name	Description
	DRV_TOUCH_ADC10BIT_HANDLE	Driver handle.

Description

10-bit ADC Touch Driver Interface Definition

This is a resistive touch screen driver that is using the Microchip Graphics Library. The calibration values are automatically checked (by reading a specific memory location on the non-volatile memory) when initializing the module if the function pointers to the read and write callback functions are initialized. If the read value is invalid calibration will automatically be executed. Otherwise, the calibration values will be loaded and used. The driver assumes that the application side provides the read and write routines to a non-volatile memory. If the callback functions are not initialized, the calibration routine will always be called at start-up to initialize the global calibration values. This driver assumes that the Graphics Library is initialized and will be using the default font of the library.

File Name

drv_adc10bit.h

Company

Microchip Technology Inc.

drv_adc10bit_config_template.h

10-bit ADC Touch Driver configuration template.

Macros

	Name	Description
	DRV_ADC10BIT_CALIBRATION_DELAY	Defines the calibration delay.
	DRV_ADC10BIT_CALIBRATION_INSET	Defines the calibration inset.
	DRV_ADC10BIT_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_ADC10BIT_INDEX	ADC10BIT static index selection.
	DRV_ADC10BIT_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_ADC10BIT_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_ADC10BIT_SAMPLE_POINTS	Defines the sample points.
	DRV_ADC10BIT_TOUCH_DIAMETER	Defines the touch diameter.

Description

10-bit ADC Touch Driver Configuration Template

This header file contains the build-time configuration selections for the 10-bit ADC Touch Driver. This is the template file which give all possible configurations that can be made. This file should not be included in any project.

File Name

drv_adc10bit_config_template.h

Company

Microchip Technology Inc.

AR1021 Touch Driver Library

This topic describes the AR1021 Touch Driver Library.

Introduction

This library provides a low-level abstraction of the AR1021 Touch Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The AR1021 Touch Driver Library, in conjunction with the Microchip AR1021 Resistive Touch Screen Controller module, allows an application to:

- Calibrate touch points
- Receive touch points

The following application services are provided by the AR1021 Touch Driver Library:

- Configuring the AR1021 controller (TouchThreshold, PenUpDelay, PenStateReportDelaylist, SensitivityFilter, etc.)
- Saving touch points to EEPROM

The operational services are not typically accessible to the application as this portion of the code resides within the Touch System Service Library software layer and is used by the Graphics Library stack services to receive touch point data.

Using the Library

This topic describes the basic architecture of the AR1021 Touch Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_ar1021.h`

The interface to the AR1021 Touch Driver library is defined in the `drv_ar1021.h` header file. Any C language source (`.c`) file that uses the AR1021 Touch Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the AR1021 Touch Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

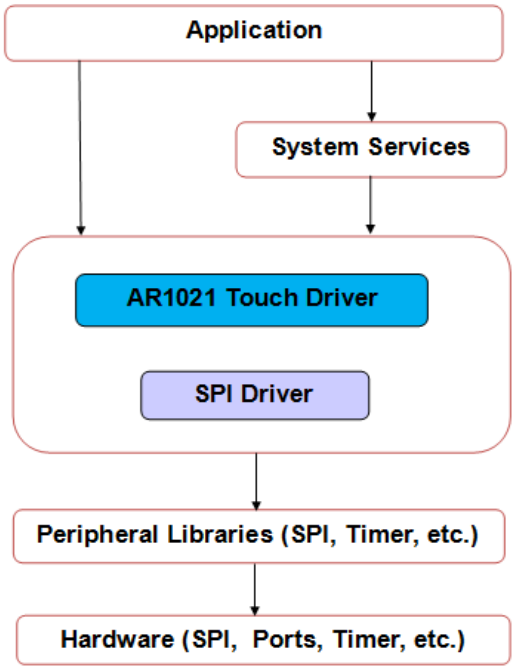
Description

The AR1021 Touch Driver Library provides the following functionality:

- AR1021 library initialization
- AR1021 controller configuration
- AR1021 controller connectivity
- AR1021 polling for pen-down and pen-up touch point events

The abstraction model shown in the following diagram depicts how the AR1021 Touch Driver is positioned in the MPLAB Harmony framework. The AR1021 Touch Driver Library uses the SPI Driver for control and touch data transfers to the AR1021 module.

AR1021 Touch Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the AR1021 Touch Driver module.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, task, and status functions.
Client Functions	Provides functions to open, close, and calibrate the AR1021 Touch Driver.

How the Library Works

The library provides interfaces to support:

- System functions, which provide system module interfaces, device initialization, deinitialization, task, touch, and status functions
- Client functions, which open, close, and calibrate the AR1021 Touch Driver

Initializing the Driver

Before the AR1021 Touch Driver can be opened, it must be configured and initialized. The driver build time configuration is defined by the configuration macros. Refer to the Building the Library section for the location of and more information on the various configuration macros and how these macros should be designed. The driver initialization is configured through the DRV_TOUCH_INIT data structure that is passed to the DRV_TOUCH_AR1021_Initialize function. The initialization parameters include the interrupt source, interrupt pin remap configuration and touch screen resolution. The following code shows an example of initializing the AR1021 Touch Driver.

Example:

```
/* The following code shows an example of designing the
 *  DRV_TOUCH_INIT data structure. It also shows how an example
 *  usage of the DRV_TOUCH_AR1021_Initialize function.
 */

DRV_TOUCH_INIT drvTouchInitData;
SYS_MODULE_OBJ objectHandle;

/* Driver initialization */
objectHandle = DRV_TOUCH_AR1021_Initialize(DRV_TOUCH_INDEX_0,
                                        (SYS_MODULE_INIT*)drvTouchInitData);
    if (SYS_MODULE_OBJ_INVALID == objectHandle)
    {
        // Handle error
    }
```

Opening the Driver

To use the AR1021 Touch Driver, the application must open the driver. This is done by calling the DRV_TOUCH_AR1021_Open function.

If successful, the DRV_TOUCH_AR1021_Open function will return a handle to the driver. This handle records the association between the client and the driver instance that was opened. The DRV_TOUCH_AR1021_Open function may return DRV_HANDLE_INVALID in the situation where the driver is not ready to be opened. When this occurs, the application can try opening the driver again. Note that the open function may return an invalid handle in other (error) cases as well. The following code shows an example of the driver being opened.

```
    DRV_HANDLE  handle;

    handle = DRV_TOUCH_AR1021_Open( DRV_TOUCH_AR1021_INDEX_0,
                                    DRV_IO_INTENT_EXCLUSIVE );

    if( DRV_HANDLE_INVALID == handle )
    {
        // Unable to open the driver
    }
```

Tasks Routine

This routine processes the AR1021 Touch Driver commands from the command queue. If the state of the command is initialize or done it returns. If the read request registration is successful the state of command is to decode input. The tasks routine decodes the input and updates the global variables storing the touch input data in form of x and y coordinates. The AR1021 Touch Driver task routine is to be called from SYS_Tasks. The following code shows an example:

```
SYS_MODULE_OBJ      object;   // Returned from DRV_TOUCH_AR1021_Initialize
```

```
void SYS_Tasks( void )
{
    DRV_TOUCH_AR1021_Tasks ( object );

    // Do other tasks
}
```

Configuring the Library

Macros

	Name	Description
	DRV_AR1021_CALIBRATION_DELAY	Define the calibration delay.
	DRV_AR1021_CALIBRATION_INSET	Define the calibration inset.
	DRV_AR1021_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_AR1021_INDEX	AR1021 static index selection.
	DRV_AR1021_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_AR1021_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_AR1021_SAMPLE_POINTS	Define the sample points.
	DRV_AR1021_TOUCH_DIAMETER	Define the touch diameter.

Description

The configuration of the AR1021 Touch Driver is accomplished through AR1021 Touch Driver selections in the MPLAB Harmony Configurator (MHC). Based on the selections made, a specific AR1021 Touch Driver is established automatically to execute all system configuration, initialization, and steady-state touch acquisitions.

Refer to Volume II: MPLAB Harmony Configurator (MHC) for more details on system configuration. Refer to the Applications Help section for additional information.

DRV_AR1021_CALIBRATION_DELAY Macro

Define the calibration delay.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_CALIBRATION_DELAY 300
```

Description

AR1021 Calibration Delay

This macro enables the delay between calibration touch points.

Remarks

None.

DRV_AR1021_CALIBRATION_INSET Macro

Define the calibration inset.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_CALIBRATION_INSET 25
```

Description

AR1021 Calibration Inset

This macro define the calibration inset.

Remarks

None.

DRV_AR1021_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_CLIENTS_NUMBER 1
```

Description

AR1021 Maximum Number of Clients

This definition selected the maximum number of clients that the AR1021 driver can support at run time.

Remarks

None.

DRV_AR1021_INDEX Macro

AR1021 static index selection.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_INDEX DRV_AR1021_INDEX_0
```

Description

AR1021 Static Index Selection

AR1021 static index selection for the driver object reference.

Remarks

This index is required to make a reference to the driver object.

DRV_AR1021_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_INSTANCES_NUMBER 1
```

Description

AR1021 hardware instance configuration

This macro sets up the maximum number of hardware instances that can be supported.

Remarks

None.

DRV_AR1021_INTERRUPT_MODE Macro

Controls operation of the driver in the interrupt or polled mode.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_INTERRUPT_MODE false
```

Description

AR1021 Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of AR1021 operation is desired
- false - Select if polling mode of AR1021 operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

DRV_AR1021_SAMPLE_POINTS Macro

Define the sample points.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_SAMPLE_POINTS 4
```

Description

AR1021 Sample Points

AR1021 sample points

Remarks

None.

DRV_AR1021_TOUCH_DIAMETER Macro

Define the touch diameter.

File

drv_ar1021_config_template.h

C

```
#define DRV_AR1021_TOUCH_DIAMETER 10
```

Description

AR1021 Touch Diameter

This macro defines the touch diameter

Remarks

None.

Building the Library

This section lists the files that are available in the AR1021 Touch Driver Library.

Description

This section list the files that are available in the `\src` folder of the AR1021 Touch Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/touch/ar1021`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_ar1021.h	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/drv_ar1021.c	Basic AR1021 Touch Driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The AR1021 Touch Driver Library depends on the following modules:

- Interrupt System Service Library
- Ports System Service Library
- Touch System Service Library
- I2C Driver Library

Library Interface

a) System Functions

	Name	Description
	DRV_TOUCH_AR1021_Deinitialize	De-initializes the specified instance of the AR1021 driver module.
	DRV_TOUCH_AR1021_FactoryDefaultSet	Set AR1021 controller to factory default configuration settings.
	DRV_TOUCH_AR1021_Initialize	Initializes the AR1021 instance for the specified driver index
	DRV_TOUCH_AR1021_RegisterConfigWrite	Write a value to the given AR1021 configuration register.
	DRV_TOUCH_AR1021_Status	Provides the current status of the AR1021 driver module.
	DRV_TOUCH_AR1021_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_TouchDataRead	Notifies the driver that the current touch data has been read
	DRV_TOUCH_AR1021_TouchGetX	Returns the x coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_TouchGetY	Returns the y coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_TouchPenGet	Returns the PEN state of the touch event.
	DRV_TOUCH_AR1021_TouchStatus	Returns the status of the current touch input.

b) Client Functions

	Name	Description
	DRV_TOUCH_AR1021_Calibrate	Calibrate the touch screen
	DRV_TOUCH_AR1021_CalibrationSet	Set calibration with pre-defined points..
	DRV_TOUCH_AR1021_Close	Closes an opened instance of the AR1021 driver
	DRV_TOUCH_AR1021_Open	Opens the specified AR1021 driver instance and returns a handle to it. **Implementation:** Dynamic

c) Data Types and Constants

	Name	Description
	DRV_TOUCH_AR1021_CALIBRATION_PROMPT_CALLBACK	Defines the callback functions required to inform the user of touch and release targets.
	DRV_TOUCH_AR1021_HANDLE	Touch screen controller AR1021 driver handle.
	DRV_TOUCH_AR1021_MODULE_ID	This is type DRV_TOUCH_AR1021_MODULE_ID.
	DRV_TOUCH_AR1021_TASK_STATE	Enumeration defining AR1021 touch controller driver task state.
	DRV_TOUCH_AR1021_HANDLE_INVALID	Definition of an invalid handle.
	DRV_TOUCH_AR1021_INDEX_0	AR1021 driver index definitions.
	DRV_TOUCH_AR1021_INDEX_COUNT	Number of valid AR1021 driver indices.

Description

This section describes the API functions of the AR1021 Touch Driver Library.

a) System Functions

DRV_TOUCH_AR1021_Deinitialize Function

De-initializes the specified instance of the AR1021 driver module.

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

De-initializes the specified instance of the AR1021 driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_TOUCH_AR1021_Status operation. The system has to use DRV_TOUCH_AR1021_Status to find out when the module is in the ready state.

Preconditions

Function DRV_TOUCH_AR1021_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_TOUCH_AR1021_Initialize
SYS_STATUS          status;

DRV_TOUCH_AR1021_Deinitialize ( object );

status = DRV_TOUCH_AR1021_Status( object );
if( SYS_MODULE_UNINITIALIZED == status )
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TOUCH_AR1021_Initialize

Function

void DRV_TOUCH_AR1021_Deinitialize (SYS_MODULE_OBJ object)

DRV_TOUCH_AR1021_FactoryDefaultSet Function

Set AR1021 controller to factory default configuration settings.

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_FactoryDefaultSet();
```

Returns

None

Description

This function returns the AR1021 to operate on factory default configuration settings.

Remarks

A power cycle is required to run on the default settings.

Preconditions

The DRV_TOUCH_AR1021_Open routine must have been called for the specified AR1021 driver instance.

Example

```
DRV_TOUCH_AR1021_FactoryDefaultSet ( void );
```

Function

void DRV_TOUCH_AR1021_FactoryDefaultSet(void)

DRV_TOUCH_AR1021_Initialize Function

Initializes the AR1021 instance for the specified driver index

File

drv_ar1021.h

C

```
SYS_MODULE_OBJ DRV_TOUCH_AR1021_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const
init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the AR1021 driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the 'init' parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the AR1021 module ID. For example, driver instance 0 can be assigned to AR10212. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_TOUCH_AR1021_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other AR1021 routine is called.

This routine should only be called once during system initialization unless DRV_TOUCH_AR1021_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

None.

Example

```
DRV_TOUCH_INIT      drvAr1021InitData;
SYS_MODULE_OBJ      objectHandle;

objectHandle = DRV_TOUCH_AR1021_Initialize(DRV_TOUCH_AR1021_INDEX_1, (SYS_MODULE_INIT*)drvAr1021InitData);
```

```
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized. Please note this is not the AR1021 id. The hardware AR1021 id is set in the initialization structure. This is the index of the driver index to use.
init	Pointer to a data structure containing any data necessary to initialize the driver. If this pointer is NULL, the driver uses the static initialization override macros for each member of the initialization data structure.

Function

SYS_MODULE_OBJ DRV_TOUCH_AR1021_Initialize(const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init)

DRV_TOUCH_AR1021_RegisterConfigWrite Function

Write a value to the given AR1021 configuration register.

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_RegisterConfigWrite(uint16_t regOffset, uint8_t Value);
```

Returns

None

Description

This function set a value to the given AR1021 configuration register.

Remarks

none

Preconditions

The DRV_TOUCH_AR1021_Open routine must have been called for the specified AR1021 driver instance.

Example

```
DRV_TOUCH_AR1021_RegisterConfigWrite(uint16_t regOffset, uint8_t Value);
```

Function

void DRV_TOUCH_AR1021_RegisterConfigWrite(uint16_t regOffset, uint8_t Value)

DRV_TOUCH_AR1021_Status Function

Provides the current status of the AR1021 driver module.

File

drv_ar1021.h

C

```
SYS_STATUS DRV_TOUCH_AR1021_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

Description

This function provides the current status of the AR1021 driver module.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_MODULE_UNINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

This function can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The DRV_TOUCH_AR1021_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_TOUCH_AR1021_Initialize
SYS_STATUS          status;

status = DRV_TOUCH_AR1021_Status( object );
if( SYS_STATUS_READY != status )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TOUCH_AR1021_Initialize

Function

SYS_STATUS DRV_TOUCH_AR1021_Status (SYS_MODULE_OBJ object)

DRV_TOUCH_AR1021_Tasks Function

Maintains the driver's state machine and implements its task queue processing.

Implementation: Dynamic

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal state machine and implement its command queue processing. It is always called from SYS_Tasks() function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks)

Preconditions

The DRV_TOUCH_AR1021_Initialize routine must have been called for the specified AR1021 driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_TOUCH_AR1021_Initialize

while( true )
{
    DRV_TOUCH_AR1021_Tasks ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_TOUCH_AR1021_Initialize)

Function

void DRV_TOUCH_AR1021_Tasks (SYS_MODULE_OBJ object);

DRV_TOUCH_AR1021_TouchDataRead Function

Notifies the driver that the current touch data has been read

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_TouchDataRead(const SYS_MODULE_INDEX index);
```

Returns

None.

Description

Notifies the driver that the current touch data has been read

Function

void DRV_TOUCH_AR1021_TouchDataRead(const SYS_MODULE_INDEX index)

DRV_TOUCH_AR1021_TouchGetX Function

Returns the x coordinate of touch input.

Implementation: Dynamic

File

drv_ar1021.h

C

```
short DRV_TOUCH_AR1021_TouchGetX(uint8_t touchNumber);
```

Returns

It returns the x coordinate of the touch input in terms of number of pixels.

Description

It returns the x coordinate in form of number of pixels for a touch input denoted by touchNumber.

Parameters

Parameters	Description
touchNumber	index to the touch input.

Function

short DRV_TOUCH_AR1021_TouchGetX(uint8 touchNumber)

DRV_TOUCH_AR1021_TouchGetY Function

Returns the y coordinate of touch input.

Implementation: Dynamic

File

drv_ar1021.h

C

```
short DRV_TOUCH_AR1021_TouchGetY(uint8_t touchNumber);
```

Returns

It returns the y coordinate of the touch input in terms of number of pixels.

Description

It returns the y coordinate in form of number of pixes for a touch input denoted by touchNumber.

Parameters

Parameters	Description
touchNumber	index to the touch input.

Function

short DRV_TOUCH_AR1021_TouchGetY(uint8 touchNumber)

DRV_TOUCH_AR1021_TouchPenGet Function

Returns the PEN state of the touch event.

File

drv_ar1021.h

C

```
DRV_TOUCH_PEN_STATE DRV_TOUCH_AR1021_TouchPenGet(uint8_t touchNumber);
```

Returns

It returns DRV_TOUCH_PEN_STATE

Description

It returns the PEN state of the last touch event corresponding to the x and y position.

Parameters

Parameters	Description
touchNumber	index to the touch input.

Function

DRV_TOUCH_PEN_STATE DRV_TOUCH_AR1021_TouchPenGet(uint8_t touchNumber)

DRV_TOUCH_AR1021_TouchStatus Function

Returns the status of the current touch input.

File

drv_ar1021.h

C

```
DRV_TOUCH_POSITION_STATUS DRV_TOUCH_AR1021_TouchStatus(const SYS_MODULE_INDEX index);
```

Returns

It returns the status of the current touch input.

Description

It returns the status of the current touch input.

Function

DRV_TOUCH_POSITION_SINGLE DRV_TOUCH_AR1021_TouchStatus(const SYS_MODULE_INDEX index)

b) Client Functions

DRV_TOUCH_AR1021_Calibrate Function

Calibrate the touch screen

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_Calibrate(const DRV_TOUCH_AR1021_CALIBRATION_PROMPT_CALLBACK * prompt);
```

Returns

None

Description

This function display calibration points on the display to enable calibration.

Remarks

None

Preconditions

The DRV_TOUCH_AR1021_Initialize routine must have been called for the specified AR1021 driver instance.

Example

```
DRV_TOUCH_AR1021_Calibrate ( handle );
```

Function

void DRV_TOUCH_AR1021_Calibrate ((const DRV_TOUCH_AR1021_CALIBRATION_PROMPT_CALLBACK * prompt))

DRV_TOUCH_AR1021_CalibrationSet Function

Set calibration with pre-defined points..

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_CalibrationSet(DRV_TOUCH_SAMPLE_POINTS * samplePoints);
```

Returns

None

Description

This function allows for the setting of pre-loaded calibration points.

Remarks

None

Preconditions

The DRV_TOUCH_AR1021_Open routine must have been called for the specified AR1021 driver instance.

Example

```
DRV_TOUCH_AR1021_CalibrationSet ( void );
```

Function

void DRV_TOUCH_AR1021_CalibrationSet(void)

DRV_TOUCH_AR1021_Close Function

Closes an opened instance of the AR1021 driver

File

drv_ar1021.h

C

```
void DRV_TOUCH_AR1021_Close(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the AR1021 driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_TOUCH_AR1021_Open before the caller may use the driver again. This function is thread safe in a RTOS application.

Usually there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_TOUCH_AR1021_Initialize routine must have been called for the specified AR1021 driver instance.

DRV_TOUCH_AR1021_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TOUCH_AR1021_Open

DRV_TOUCH_AR1021_Close ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TOUCH_AR1021_Close (DRV_HANDLE handle)

DRV_TOUCH_AR1021_Open Function

Opens the specified AR1021 driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_ar1021.h

C

```
DRV_HANDLE DRV_TOUCH_AR1021_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. An error can occur when the following is true:

- if the number of client objects allocated via DRV_TOUCH_AR1021_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid

Description

This routine opens the specified AR1021 driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The current version of driver does not support the DRV_IO_INTENT feature. The driver is by default non-blocking. The driver can perform both read and write to the AR1021 device. The driver supports single client only.

Remarks

The handle returned is valid until the DRV_TOUCH_AR1021_Close routine is called. This routine will NEVER block waiting for hardware. If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

The DRV_TOUCH_AR1021_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_TOUCH_AR1021_Open( DRV_TOUCH_AR1021_INDEX_0, DRV_IO_INTENT_EXCLUSIVE );

if( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Index of the driver initialized with DRV_TOUCH_AR1021_Initialize().
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver. The current version of driver does not support the selective IO intent feature.

Function

DRV_HANDLE DRV_TOUCH_AR1021_Open (const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT intent)

c) Data Types and Constants

DRV_TOUCH_AR1021_CALIBRATION_PROMPT_CALLBACK Structure

Defines the callback functions required to inform the user of touch and release targets.

File

drv_ar1021.h

C

```
typedef struct {
  void (* firstPromptCallback)(void);
  void (* secondPromptCallback)(void);
  void (* thirdPromptCallback)(void);
  void (* fourthPromptCallback)(void);
  void (* completeCallback)(void);
} DRV_TOUCH_AR1021_CALIBRATION_PROMPT_CALLBACK;
```

Members

Members	Description
void (* firstPromptCallback)(void);	first calibration target
void (* secondPromptCallback)(void);	second calibration target
void (* thirdPromptCallback)(void);	third calibration target
void (* fourthPromptCallback)(void);	fourth calibration target
void (* completeCallback)(void);	complete calibration

Description

TOUCH Driver Calibration Initialization Data

This data type defines the callback function pointers required to inform of touch and release targets. The driver will invoke each callback in sequential order. The host code can display graphic and/or textual content to direct the user when a where on the LCD display to touch and release.

Remarks

None.

DRV_TOUCH_AR1021_HANDLE Type

Touch screen controller AR1021 driver handle.

File

drv_ar1021.h

C

```
typedef uintptr_t DRV_TOUCH_AR1021_HANDLE;
```

Description

AR1021 Driver Handle

Touch controller AR1021 driver handle is a handle for the driver client object. Each driver with successful open call will return a new handle to the client object.

Remarks

None.

DRV_TOUCH_AR1021_MODULE_ID Enumeration

File

drv_ar1021.h

C

```
typedef enum {
  AR1021_ID_1 = 0,
  AR1021_NUMBER_OF_MODULES
} DRV_TOUCH_AR1021_MODULE_ID;
```

Description

This is type DRV_TOUCH_AR1021_MODULE_ID.

DRV_TOUCH_AR1021_TASK_STATE Enumeration

Enumeration defining AR1021 touch controller driver task state.

File

drv_ar1021.h

C

```
typedef enum {
  DRV_TOUCH_AR1021_TASK_STATE_INIT = 0,
  DRV_TOUCH_AR1021_TASK_STATE_DONE
} DRV_TOUCH_AR1021_TASK_STATE;
```

Members

Members	Description
DRV_TOUCH_AR1021_TASK_STATE_INIT = 0	Task initialize state
DRV_TOUCH_AR1021_TASK_STATE_DONE	Task complete state

Description

AR1021 Touch Controller Driver Task State

This enumeration defines the AR1021 touch controller driver task state. The task state helps to synchronize the operations of initialization the the task, adding the read input task to the task queue once the touch controller notifies the available touch input and a decoding the touch input received.

Remarks

None.

DRV_TOUCH_AR1021_HANDLE_INVALID Macro

Definition of an invalid handle.

File

drv_ar1021.h

C

```
#define DRV_TOUCH_AR1021_HANDLE_INVALID ((DRV_TOUCH_AR1021_HANDLE)(-1))
```

Description

AR1021 Driver Invalid Handle

This is the definition of an invalid handle. An invalid handle is is returned by DRV_TOUCH_AR1021_Open() and DRV_AR1021_Close() functions if the request was not successful.

Remarks

None.

DRV_TOUCH_AR1021_INDEX_0 Macro

AR1021 driver index definitions.

File

drv_ar1021.h

C

```
#define DRV_TOUCH_AR1021_INDEX_0 0
```

Description

AR1021 Driver Module Index Numbers

These constants provide the AR1021 driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_AR1021_Initialize and DRV_AR1021_Open functions to identify the driver instance in use.

DRV_TOUCH_AR1021_INDEX_COUNT Macro

Number of valid AR1021 driver indices.

File

drv_ar1021.h

C

```
#define DRV_TOUCH_AR1021_INDEX_COUNT 1
```

Description

AR1021 Driver Module Index Count

This constant identifies the number of valid AR1021 driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific header files defined as part of the peripheral libraries.

Files

Files

Name	Description
drv_ar1021.h	Touch controller AR1021 driver implementation.

Description

This section lists the source and header files used by the AR1021 Touch Driver Library.

drv_ar1021.h

Touch controller AR1021 driver implementation.

Enumerations

	Name	Description
	DRV_TOUCH_AR1021_MODULE_ID	This is type DRV_TOUCH_AR1021_MODULE_ID.
	DRV_TOUCH_AR1021_TASK_STATE	Enumeration defining AR1021 touch controller driver task state.

Functions

	Name	Description
	DRV_TOUCH_AR1021_Calibrate	Calibrate the touch screen
	DRV_TOUCH_AR1021_CalibrationSet	Set calibration with pre-defined points..
	DRV_TOUCH_AR1021_Close	Closes an opened instance of the AR1021 driver
	DRV_TOUCH_AR1021_Deinitialize	De-initializes the specified instance of the AR1021 driver module.
	DRV_TOUCH_AR1021_FactoryDefaultSet	Set AR1021 controller to factory default configuration settings.
	DRV_TOUCH_AR1021_Initialize	Initializes the AR1021 instance for the specified driver index
	DRV_TOUCH_AR1021_Open	Opens the specified AR1021 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_RegisterConfigWrite	Write a value to the given AR1021 configuration register.
	DRV_TOUCH_AR1021_Status	Provides the current status of the AR1021 driver module.
	DRV_TOUCH_AR1021_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_TouchDataRead	Notifies the driver that the current touch data has been read
	DRV_TOUCH_AR1021_TouchGetX	Returns the x coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_TouchGetY	Returns the y coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_AR1021_TouchPenGet	Returns the PEN state of the touch event.
	DRV_TOUCH_AR1021_TouchStatus	Returns the status of the current touch input.

Macros

	Name	Description
	DRV_TOUCH_AR1021_HANDLE_INVALID	Definition of an invalid handle.
	DRV_TOUCH_AR1021_INDEX_0	AR1021 driver index definitions.
	DRV_TOUCH_AR1021_INDEX_COUNT	Number of valid AR1021 driver indices.

Structures

	Name	Description
	DRV_TOUCH_AR1021_CALIBRATION_PROMPT_CALLBACK	Defines the callback functions required to inform the user of touch and release targets.

Types

	Name	Description
	DRV_TOUCH_AR1021_HANDLE	Touch screen controller AR1021 driver handle.

Description

Touch controller AR1021 driver file

This file consist of touch controller AR1021 driver interfaces. It implements the driver interfaces which read the touch input data from AR1021 through SPI bus.

File Name

drv_ar1021.c

MTCH6301 Touch Driver Library

This topic describes the MTCH6301 Touch Driver Library.

Introduction

This library provides an interface to manage the MTCH6301 Touch Driver module on the Microchip family of microcontrollers in different modes of operation.

Description

The MPLAB Harmony MTCH6301 Touch Driver provides a high-level interface to the MTCH6301 touch controller device. This driver provides application routines to read the touch input data from the touch screen. The MTCH6301 device can notify the availability of touch input data through external interrupt. The MTCH6301 driver allows the application to map a controller pin as an external interrupt pin.

Currently, the MTCH6301 Touch Driver only supports non-gestural single-fingered touch input.

Using the Library

This topic describes the basic architecture of the MTCH6301 Touch Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_mtch6301.h`

The interface to the MTCH6301 Touch Driver library is defined in the `drv_mtch6301.h` header file. Any C language source (`.c`) file that uses the MTCH6301 Touch Driver library should include this header.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the MTCH6301 Touch Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

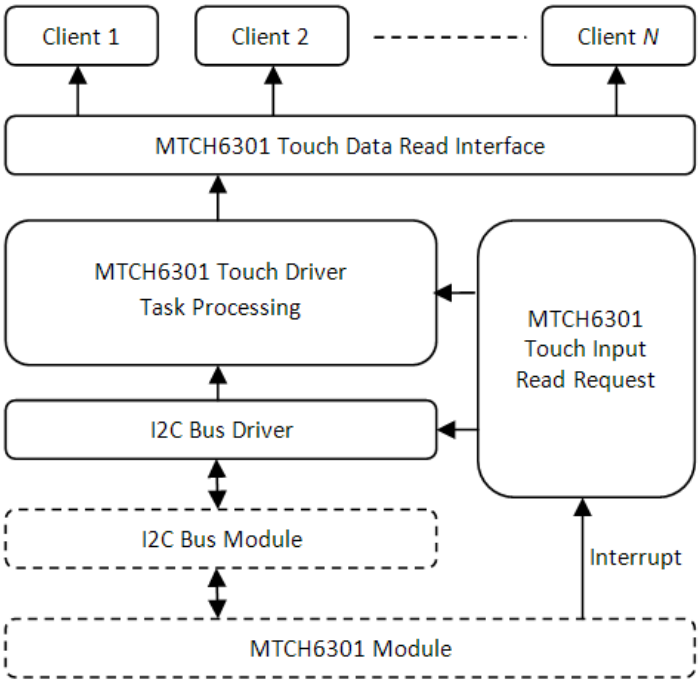
Description

The MTCH6301 Touch Driver has routines to perform the following operations:

- Sending read request
- Reading the touch input data
- Access to the touch input data

The driver initialization routines allow the application to initialize the driver. The driver must be initialized before it can be used by application. Once the driver is initialized the driver open routine allows to retrieve the client handle. Once the touch input is available a touch input read request is sent and input data is retrieved in a buffer. The buffer data is then decoded to get the x and y coordinate of the touch screen in the form of the number of pixels.

MTCH6301 Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the MTCH6301 Touch Driver.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, open, close, task, and status functions.

How the Library Works

The library provides interfaces to support:

- System functions, which provide system module interfaces, device initialization, deinitialization, open, close, task, and status functions.
- Read Request function, which provides Touch input data read request function
- Read Touch Input function, which provides functions retrieving updated Touch input in the form x and y coordinates.

Initializing the Driver

Before the MTCH6301 Touch Driver can be opened, it must be configured and initialized. The driver build time configuration is defined by the configuration macros. Refer to the Building the Library section for the location of and more information on the various configuration macros and how these macros should be designed. The driver initialization is configured through the DRV_TOUCH_INIT data structure that is passed to the DRV_TOUCH_MTCH6301_Initialize function. The initialization parameters include the interrupt source, interrupt pin remap configuration and touch screen resolution. The following code shows an example of initializing the MTCH6301 Touch Driver.

Example:

```
/* The following code shows an example of designing the
 *  DRV_TOUCH_INIT data structure. It also shows how an example
 *  usage of the DRV_TOUCH_MTCH6301_Initialize function.
 */

DRV_TOUCH_INIT drvTouchInitData;
SYS_MODULE_OBJ objectHandle;

/* Touch Module Id*/
drvTouchInitData.touchId                    = DRV_TOUCH_INDEX_0;
```

```
/* I2C Bus driver open */
drvTouchInitData.drvOpen                    = DRV_I2C_Open;

/* Interrupt Source for Touch */
drvTouchInitData.interruptSource            = INT_SOURCE_EXTERNAL_1;

/* Interrupt Pin function mapping */
drvTouchInitData.interruptPort.inputFunction = INPUT_FUNC_INT1;

/* Pin to be mapped as interrupt pin */
drvTouchInitData.interruptPort.inputPin     = INPUT_PIN_RPE8;

/* Analog pin number */
drvTouchInitData.interruptPort.analogPin    = PORTS_ANALOG_PIN_25;

/* Pin Mode of analog pin */
drvTouchInitData.interruptPort.pinMode      = PORTS_PIN_MODE_DIGITAL;

/* Interrupt pin port */
drvTouchInitData.interruptPort.channel      = PORT_CHANNEL_E;

/* Interrupt pin port maskl */
drvTouchInitData.interruptPort.dataMask     = 0x8;

/* Touch screen orientation */
drvTouchInitData.orientation                = DISP_ORIENTATION;

/* Touch screen horizontal resolution */
drvTouchInitData.horizontalResolution           = DISP_HOR_RESOLUTION;

/* Touch screen vertical resolution */
drvTouchInitData.verticalResolution         = DISP_VER_RESOLUTION;

/* Driver initialization */
objectHandle = DRV_TOUCH_MTCH6301_Initialize(DRV_TOUCH_INDEX_0,
                                             (SYS_MODULE_INIT*)drvTouchInitData);
    if (SYS_MODULE_OBJ_INVALID == objectHandle)
    {
        // Handle error
    }
```

Opening the Driver

To use the MTCH6301 Touch Driver, the application must open the driver. This is done by calling the DRV_TOUCH_MTCH6301_Open function.

If successful, the DRV_TOUCH_MTCH6301_Open function will return a handle to the driver. This handle records the association between the client and the driver instance that was opened. The DRV_TOUCH_MTCH6301_Open function may return DRV_HANDLE_INVALID in the situation where the driver is not ready to be opened. When this occurs, the application can try opening the driver again. Note that the open function may return an invalid handle in other (error) cases as well. The following code shows an example of the driver being opened.

```
    DRV_HANDLE  handle;

    handle = DRV_TOUCH_MTCH6301_Open( DRV_TOUCH_MTCH6301_INDEX_0,
                                      DRV_IO_INTENT_EXCLUSIVE );

    if( DRV_HANDLE_INVALID == handle )
    {
        // Unable to open the driver
    }
```

Touch Input Read Request

To read the touch input from the MTCH6301 touch controller device, a read request must be registered. This is done by calling the DRV_TOUCH_MTCH6301_ReadRequest. If successful it registers a buffer read request to the I2C command queue. It also adds a input decode command to the MTCH6301 command queue once the I2C returns with touch input data. It can return error if the driver instance object is invalid or the MTCH6301 command queue is full. The read request is to be called from the MTCH6301 ISR. This ISR is triggered once the touch input is available. The following code shows an example of a MTCH6301 read request registration:

```
    SYS_MODULE_OBJ      object;   // Returned from DRV_TOUCH_MTCH6301_Initialize
```

```
void __ISR(_EXTERNAL_INT_VECTOR, ipl5) _IntHandlerDrvMtch6301(void)
{
    DRV_TOUCH_MTCH6301_ReadRequest ( object );

    // Do other tasks
}
```

Tasks Routine

This routine processes the MTCH6301 commands from the command queue. If the state of the command is initialize or done it returns. If the read request registration is successful the state of command is to decode input. The tasks routine decodes the input and updates the global variables storing the touch input data in form of x and y coordinates. The MTCH6301 Touch Driver task routine is to be called from SYS_Tasks. The following code shows an example:

```
SYS_MODULE_OBJ      object;   // Returned from DRV_TOUCH_MTCH6301_Initialize

void SYS_Tasks( void )
{
    DRV_TOUCH_MTCH6301_Tasks ( object );

    // Do other tasks
}
```

Configuring the Library

Macros

	Name	Description
	DRV_MTCH6301_CALIBRATION_DELAY	Defines the calibration delay.
	DRV_MTCH6301_CALIBRATION_INSET	Defines the calibration inset.
	DRV_MTCH6301_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_MTCH6301_INDEX	MTCH6301 static index selection.
	DRV_MTCH6301_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_MTCH6301_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_MTCH6301_SAMPLE_POINTS	Define the sample points.
	DRV_MTCH6301_TOUCH_DIAMETER	Defines the touch diameter.

Description

The configuration of the MTCH6301 Touch Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the MTCH6301 Touch Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the MTCH6301 Touch Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_MTCH6301_CALIBRATION_DELAY Macro

Defines the calibration delay.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_CALIBRATION_DELAY 300
```

Description

MTCH6301 Calibration Delay

This macro enables the delay between calibration touch points.

Remarks

None.

DRV_MTCH6301_CALIBRATION_INSET Macro

Defines the calibration inset.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_CALIBRATION_INSET 25
```

Description

MTCH6301 Calibration Inset

This macro defines the calibration inset.

Remarks

None.

DRV_MTCH6301_CLIENTS_NUMBER Macro

Selects the maximum number of clients.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_CLIENTS_NUMBER 1
```

Description

MTCH6301 maximum number of clients

This macro selects the maximum number of clients.

This definition selected the maximum number of clients that the MTCH6301 driver can support at run time.

Remarks

None.

DRV_MTCH6301_INDEX Macro

MTCH6301 static index selection.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_INDEX DRV_MTCH6301_INDEX_0
```

Description

MTCH6301 Static Index Selection

This macro specifies the static index selection for the driver object reference.

Remarks

This index is required to make a reference to the driver object.

DRV_MTCH6301_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_INSTANCES_NUMBER 1
```

Description

MTCH6301 hardware instance configuration

This macro sets up the maximum number of hardware instances that can be supported.

Remarks

None.

DRV_MTCH6301_INTERRUPT_MODE Macro

Controls operation of the driver in the interrupt or polled mode.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_INTERRUPT_MODE false
```

Description

MTCH6301 Interrupt And Polled Mode Operation Control

This macro controls the operation of the driver in the interrupt mode of operation. The possible values of this macro are:

- true - Select if interrupt mode of MTCH6301 operation is desired
- false - Select if polling mode of MTCH6301 operation is desired

Not defining this option to true or false will result in a build error.

Remarks

None.

DRV_MTCH6301_SAMPLE_POINTS Macro

Define the sample points.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_SAMPLE_POINTS 4
```

Description

MTCH6301 Sample Points

MTCH6301 sample points

Remarks

None.

DRV_MTCH6301_TOUCH_DIAMETER Macro

Defines the touch diameter.

File

drv_mtch6301_config_template.h

C

```
#define DRV_MTCH6301_TOUCH_DIAMETER 10
```

Description

MTCH6301 Touch Diameter

This macro defines the touch diameter

Remarks

None.

Building the Library

This section lists the files that are available in the MTCH6301 Touch Driver Library.

Description

This section list the files that are available in the \src folder of the MTCH6301 Touch Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/touch/mtch6301`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_mtch6301.h	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/drv_mtch6301.c	Basic MTCH6301 Touch Driver implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The MTCH6301 Touch Driver Library depends on the following modules:

- Interrupt System Service Library
- Ports System Service Library
- Touch System Service Library
- I2C Driver Library

Library Interface

a) System Functions

	Name	Description
	DRV_TOUCH_MTCH6301_Close	Closes an opened instance of the MTCH6301 driver. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Deinitialize	Deinitializes the specified instance of the MTCH6301 driver module. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Initialize	Initializes the MTCH6301 instance for the specified driver index. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Open	Opens the specified MTCH6301 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Status	Provides the current status of the MTCH6301 driver module. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic

	DRV_TOUCH_MTCH6301_ReadRequest	Sends a read request to I2C bus driver and adds the read task to queue. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_TouchGetX	Returns the x coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_TouchGetY	Returns the y coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_TouchDataRead	Notifies the driver that the current touch data has been read
	DRV_TOUCH_MTCH6301_TouchStatus	Returns the status of the current touch input.

b) Data Types and Constants

	Name	Description
	_DRV_MTCH6301_CLIENT_OBJECT	MTCH6301 Driver client object maintaining client data.
	DRV_TOUCH_MTCH6301_HANDLE	Touch screen controller MTCH6301 driver handle.
	DRV_TOUCH_MTCH6301_MODULE_ID	Number of valid MTCH6301 driver indices.
	DRV_TOUCH_MTCH6301_HANDLE_INVALID	Definition of an invalid handle.
	DRV_TOUCH_MTCH6301_I2C_READ_FRAME_SIZE	I2C Frame size for reading MTCH6301 touch input.
	DRV_TOUCH_MTCH6301_CLIENT_OBJECT	MTCH6301 Driver client object maintaining client data.
	DRV_TOUCH_MTCH6301_INDEX_0	MTCH6301 driver index definitions.
	DRV_TOUCH_MTCH6301_INDEX_1	This is macro DRV_TOUCH_MTCH6301_INDEX_1.
	DRV_TOUCH_MTCH6301_INDEX_COUNT	Number of valid Touch controller MTCH6301 driver indices.
	DRV_TOUCH_MTCH6301_OBJECT	Defines the data structure maintaining MTCH6301 driver instance object.
	DRV_TOUCH_MTCH6301_TASK_QUEUE	Defines the MTCH6301 Touch Controller driver task data structure.
	DRV_TOUCH_MTCH6301_TASK_STATE	Enumeration defining MTCH6301 touch controller driver task state.
	DRV_TOUCH_MTCH6301_I2C_MASTER_READ_ID	MTCH6301 input read, I2C address from where master reads touch input data.
	DRV_TOUCH_MTCH6301_I2C_MASTER_WRITE_ID	MTCH6301 command register write, I2C address where master sends the commands.

Description

This section describes the API functions of the MTCH6301 Touch Driver library.

Refer to each section for a detailed description.

a) System Functions

DRV_TOUCH_MTCH6301_Close Function

Closes an opened instance of the MTCH6301 driver.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
void DRV_TOUCH_MTCH6301_Close(DRV_HANDLE handle);
```

Returns

None

Description

This function closes an opened instance of the MTCH6301 driver, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling DRV_TOUCH_MTCH6301_Open before the caller may use the driver again. This function is thread safe in a RTOS application.

Usually, there is no need for the driver client to verify that the Close operation has completed.

Preconditions

The DRV_TOUCH_MTCH6301_Initialize routine must have been called for the specified MTCH6301 driver instance.

DRV_TOUCH_MTCH6301_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_TOUCH_MTCH6301_Open

DRV_TOUCH_MTCH6301_Close ( handle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_TOUCH_MTCH6301_Close (DRV_HANDLE handle)

DRV_TOUCH_MTCH6301_Deinitialize Function

Deinitializes the specified instance of the MTCH6301 driver module.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
void DRV_TOUCH_MTCH6301_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the MTCH6301 driver module, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again.

This function will NEVER block waiting for hardware. If the operation requires time to allow the hardware to complete, this will be reported by the DRV_TOUCH_MTCH6301_Status operation. The system has to use DRV_TOUCH_MTCH6301_Status to determine when the module is in the ready state.

Preconditions

Function DRV_TOUCH_MTCH6301_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Parameter: object - Driver object handle, returned from DRV_TOUCH_MTCH6301_Initialize

Example

```
SYS_MODULE_OBJ      object;    //Returned from DRV_TOUCH_MTCH6301_Initialize
SYS_STATUS          status;

DRV_TOUCH_MTCH6301_Deinitialize ( object );

status = DRV_TOUCH_MTCH6301_Status( object );
if( SYS_MODULE_UNINITIALIZED == status )
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Function

void DRV_TOUCH_MTCH6301_Deinitialize (SYS_MODULE_OBJ object)

DRV_TOUCH_MTCH6301_Initialize Function

Initializes the MTCH6301 instance for the specified driver index.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
SYS_MODULE_OBJ DRV_TOUCH_MTCH6301_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the MTCH6301 driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data is specified by the 'init' parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the MTCH6301 module ID. For example, driver instance 0 can be assigned to MTCH63012. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_TOUCH_MTCH6301_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other MTCH6301 routine is called.

This routine should only be called once during system initialization unless DRV_TOUCH_MTCH6301_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

None.

Example

```
DRV_TOUCH_MTCH6301_INIT        init;
SYS_MODULE_OBJ      objectHandle;

// Populate the MTCH6301 initialization structure
// Touch Module Id
init.touchId                      = DRV_TOUCH_INDEX_0;

// I2C Bus driver open
init.drvOpen                      = DRV_I2C_Open;

// Interrupt Source for Touch
init.interruptSource              = INT_SOURCE_EXTERNAL_1;

// Interrupt Pin function mapping
init.interruptPort.inputFunction  = INPUT_FUNC_INT1;

// Pin to be mapped as interrupt pin
init.interruptPort.inputPin       = INPUT_PIN_RPE8;

// Analog pin number
init.interruptPort.analogPin      = PORTS_ANALOG_PIN_25;

// Pin Mode of analog pin
init.interruptPort.pinMode        = PORTS_PIN_MODE_DIGITAL;

// Interrupt pin port
init.interruptPort.channel        = PORT_CHANNEL_E;

// Interrupt pin port maskl
init.interruptPort.dataMask       = 0x8;

// Touch screen orientation
init.orientation                  = DISP_ORIENTATION;

// Touch screen horizontal resolution
init.horizontalResolution         = DISP_HOR_RESOLUTION;

// Touch screen vertical resolution
init.verticalResolution           = DISP_VER_RESOLUTION;
```

```
objectHandle = DRV_TOUCH_MTCH6301_Initialize(DRV_TOUCH_INDEX_0,
                                             (SYS_MODULE_INIT*)init);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized. Please note this is not the MTCH6301 ID. The hardware MTCH6301 ID is set in the initialization structure. This is the index of the driver index to use.
init	Pointer to a data structure containing any data necessary to initialize the driver. If this pointer is NULL, the driver uses the static initialization override macros for each member of the initialization data structure.

Function

SYS_MODULE_OBJ DRV_TOUCH_MTCH6301_Initialize(const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init)

DRV_TOUCH_MTCH6301_Open Function

Opens the specified MTCH6301 driver instance and returns a handle to it.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
DRV_HANDLE DRV_TOUCH_MTCH6301_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. An error can occur when the following is true:

- if the number of client objects allocated via DRV_TOUCH_MTCH6301_CLIENTS_NUMBER is insufficient
- if the client is trying to open the driver but driver has been opened exclusively by another client
- if the driver hardware instance being opened is not initialized or is invalid

Description

This routine opens the specified MTCH6301 driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The current version of driver does not support the DRV_IO_INTENT feature. The driver is by default non-blocking. The driver can perform both read and write to the MTCH6301 device. The driver supports single client only.

Remarks

The handle returned is valid until the DRV_TOUCH_MTCH6301_Close routine is called. This routine will NEVER block waiting for hardware. If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application. It should not be called in an ISR.

Preconditions

The DRV_TOUCH_MTCH6301_Initialize function must have been called before calling this function.

Example

```
DRV_HANDLE  handle;

handle = DRV_TOUCH_MTCH6301_Open( DRV_TOUCH_MTCH6301_INDEX_0,
                                  DRV_IO_INTENT_EXCLUSIVE );

if( DRV_HANDLE_INVALID == handle )
{
    // Unable to open the driver
}
```

Parameters

Parameters	Description
drvIndex	Index of the driver initialized with DRV_TOUCH_MTCH6301_Initialize().
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver. The current version of driver does not support the selective IO intent feature.

Function

DRV_HANDLE DRV_TOUCH_MTCH6301_Open (const SYS_MODULE_INDEX drvIndex,
const DRV_IO_INTENT intent)

DRV_TOUCH_MTCH6301_Status Function

Provides the current status of the MTCH6301 driver module.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
SYS_STATUS DRV_TOUCH_MTCH6301_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system-level operation and cannot start another

Description

This function provides the current status of the MTCH6301 driver module.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_MODULE_UNINITIALIZED - Indicates that the driver has been deinitialized

This value is less than SYS_STATUS_ERROR.

This function can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The DRV_TOUCH_MTCH6301_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;  // Returned from DRV_TOUCH_MTCH6301_Initialize
SYS_STATUS          status;

status = DRV_TOUCH_MTCH6301_Status( object );
if( SYS_STATUS_READY != status )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from DRV_TOUCH_MTCH6301_Initialize

Function

SYS_STATUS DRV_TOUCH_MTCH6301_Status (SYS_MODULE_OBJ object)

DRV_TOUCH_MTCH6301_Tasks Function

Maintains the driver's state machine and implements its task queue processing.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
void DRV_TOUCH_MTCH6301_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal state machine and implement its command queue processing. It is always called from SYS_Tasks() function. This routine decodes the touch input data available in drvI2CReadFrameData.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks)

Preconditions

The DRV_TOUCH_MTCH6301_Initialize routine must have been called for the specified MTCH6301 driver instance.

Example

```
SYS_MODULE_OBJ      object;   // Returned from DRV_TOUCH_MTCH6301_Initialize

void SYS_Tasks( void )
{
    DRV_TOUCH_MTCH6301_Tasks ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_TOUCH_MTCH6301_Initialize)

Function

void DRV_TOUCH_MTCH6301_Tasks (SYS_MODULE_OBJ object);

DRV_TOUCH_MTCH6301_ReadRequest Function

Sends a read request to I2C bus driver and adds the read task to queue.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
void DRV_TOUCH_MTCH6301_ReadRequest(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to send a touch input read request to the I2C bus driver and adding the input read decode task to the queue. It is always called from MTCH6301 interrupt ISR routine.

Remarks

This function is normally not called directly by an application. It is called by the MTCH6301 ISR routine.

Preconditions

The DRV_TOUCH_MTCH6301_Initialize routine must have been called for the specified MTCH6301 driver instance.

Example

```
SYS_MODULE_OBJ      object;   // Returned from DRV_TOUCH_MTCH6301_Initialize

void __ISR(_EXTERNAL_INT_VECTOR, ipl5) _IntHandlerDrvMtch6301(void)
{
    DRV_TOUCH_MTCH6301_ReadRequest ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_TOUCH_MTCH6301_Initialize)

Function

void DRV_TOUCH_MTCH6301_ReadRequest(SYS_MODULE_OBJ object)

DRV_TOUCH_MTCH6301_TouchGetX Function

Returns the x coordinate of touch input.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
short DRV_TOUCH_MTCH6301_TouchGetX(uint8_t touchNumber);
```

Returns

It returns the x coordinate of the touch input in terms of number of pixels.

Description

It returns the x coordinate in form of number of pixes for a touch input denoted by touchNumber.

Parameters

Parameters	Description
touchNumber	index to the touch input.

Function

short DRV_TOUCH_MTCH6301_TouchGetX(uint8 touchNumber)

DRV_TOUCH_MTCH6301_TouchGetY Function

Returns the y coordinate of touch input.

Implementation: Dynamic

File

drv_mtch6301.h

C

```
short DRV_TOUCH_MTCH6301_TouchGetY(uint8_t touchNumber);
```

Returns

It returns the y coordinate of the touch input in terms of number of pixels.

Description

It returns the y coordinate in form of number of pixes for a touch input denoted by touchNumber.

Parameters

Parameters	Description
touchNumber	index to the touch input.

Function

short DRV_TOUCH_MTCH6301_TouchGetY(uint8 touchNumber)

DRV_TOUCH_MTCH6301_TouchDataRead Function

Notifies the driver that the current touch data has been read

File

drv_mtch6301.h

C

```
void DRV_TOUCH_MTCH6301_TouchDataRead(const SYS_MODULE_INDEX index);
```

Returns

None.

Description

Notifies the driver that the current touch data has been read

Function

void DRV_TOUCH_MTCH6301_TouchDataRead(const SYS_MODULE_INDEX index)

DRV_TOUCH_MTCH6301_TouchStatus Function

Returns the status of the current touch input.

File

drv_mtch6301.h

C

```
DRV_TOUCH_POSITION_STATUS DRV_TOUCH_MTCH6301_TouchStatus(const SYS_MODULE_INDEX index);
```

Returns

It returns the status of the current touch input.

Description

It returns the status of the current touch input.

Function

DRV_TOUCH_POSITION_SINGLE DRV_TOUCH_MTCH6301_TouchStatus(const SYS_MODULE_INDEX index)

b) Data Types and Constants

DRV_TOUCH_MTCH6301_HANDLE Type

Touch screen controller MTCH6301 driver handle.

File

drv_mtch6301.h

C

```
typedef uintptr_t DRV_TOUCH_MTCH6301_HANDLE;
```

Description

MTCH6301 Driver Handle

Touch controller MTCH6301 driver handle is a handle for the driver client object. Each driver with succesful open call will return a new handle to the client object.

Remarks

None.

DRV_TOUCH_MTCH6301_MODULE_ID Enumeration

Number of valid MTCH6301 driver indices.

File

drv_mtch6301.h

C

```
typedef enum {
  MTCH6301_ID_1 = 0,
  MTCH6301_NUMBER_OF_MODULES
} DRV_TOUCH_MTCH6301_MODULE_ID;
```

Description

MTCH6301 Driver Module Index Count

This constant identifies the number of valid MTCH6301 driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific header files defined as part of the peripheral libraries.

DRV_TOUCH_MTCH6301_HANDLE_INVALID Macro

Definition of an invalid handle.

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_HANDLE_INVALID ((DRV_TOUCH_MTCH6301_HANDLE)(-1))
```

Description

MTCH6301 Driver Invalid Handle

This is the definition of an invalid handle. An invalid handle is is returned by DRV_TOUCH_MTCH6301_Open() and DRV_MTCH6301_Close() functions if the request was not successful.

Remarks

None.

DRV_TOUCH_MTCH6301_I2C_READ_FRAME_SIZE Macro

I2C Frame size for reading MTCH6301 touch input.

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_I2C_READ_FRAME_SIZE 7
```

Description

MTCH6301 Driver Module I2C Frame Size

This constant identifies the size of I2C frame required to read from MTCH6301 touch controller. MTCH6301 notifies the availability of input data through interrupt pin.

Remarks

This constant should be used in place of hard-coded numeric literals. This value is derived from device-specific data sheets.

DRV_TOUCH_MTCH6301_CLIENT_OBJECT Structure

MTCH6301 Driver client object maintaining client data.

File

drv_mtch6301.h

C

```
typedef struct _DRV_MTCH6301_CLIENT_OBJECT {
  DRV_TOUCH_MTCH6301_OBJECT* driverObject;
  DRV_IO_INTENT intent;
  struct DRV_TOUCH_MTCH6301_CLIENT_OBJECT* pNext;
} DRV_TOUCH_MTCH6301_CLIENT_OBJECT;
```

Members

Members	Description
DRV_TOUCH_MTCH6301_OBJECT* driverObject;	Driver Object associated with the client
DRV_IO_INTENT intent;	The intent with which the client was opened
struct DRV_TOUCH_MTCH6301_CLIENT_OBJECT* pNext;	Next driver client object

Description

MTCH6301 Driver client object

This defines the object required for the maintenance of the software clients instance. This object exists once per client instance.

Remarks

None.

DRV_TOUCH_MTCH6301_INDEX_0 Macro

MTCH6301 driver index definitions.

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_INDEX_0 0
```

Description

MTCH6301 Driver Module Index Numbers

These constants provide the MTCH6301 driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_MTCH6301_Initialize and DRV_MTCH6301_Open functions to identify the driver instance in use.

DRV_TOUCH_MTCH6301_INDEX_1 Macro

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_INDEX_1 1
```

Description

This is macro DRV_TOUCH_MTCH6301_INDEX_1.

DRV_TOUCH_MTCH6301_INDEX_COUNT Macro

Number of valid Touch controller MTCH6301 driver indices.

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_INDEX_COUNT 2
```

Description

MTCH6301 Driver Module Index Count

This constant identifies the number of valid Touch Controller MTCH6301 driver indices.

Remarks

This constant should be used in place of hard-coded numeric literals. This value is derived from device-specific header files defined as part of the peripheral libraries.

DRV_TOUCH_MTCH6301_OBJECT Structure

Defines the data structure maintaining MTCH6301 driver instance object.

File

drv_mtch6301.h

C

```
typedef struct {
  SYS_STATUS status;
  int touchId;
  SYS_MODULE_INDEX drvIndex;
  bool inUse;
  bool isExclusive;
  uint8_t numClients;
  INT_SOURCE interruptSource;
  uint16_t orientation;
  uint16_t horizontalResolution;
  uint16_t verticalResolution;
  DRV_HANDLE (* drvOpen)(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
  int32_t readRequest;
  DRV_TOUCH_MTCH6301_TASK_QUEUE* taskQueue;
  DRV_HANDLE drvI2CHandle;
  DRV_TOUCH_POSITION_STATUS touchStatus;
} DRV_TOUCH_MTCH6301_OBJECT;
```

Members

Members	Description
SYS_STATUS status;	The status of the driver
int touchId;	The peripheral Id associated with the object
SYS_MODULE_INDEX drvIndex;	Save the index of the driver. Important to know this as we are using reference based accessing
bool inUse;	Flag to indicate instance in use
bool isExclusive;	Flag to indicate module used in exclusive access mode
uint8_t numClients;	Number of clients possible with the hardware instance
INT_SOURCE interruptSource;	Touch input interrupt source
uint16_t orientation;	Orientation of the display (given in degrees of 0,90,180,270)
uint16_t horizontalResolution;	Horizontal Resolution of the displayed orientation in Pixels
uint16_t verticalResolution;	Vertical Resolution of the displayed orientaion in Pixels
DRV_HANDLE (* drvOpen)(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);	Callback for I2C Driver Open call
int32_t readRequest;	Touch Input read request counter
DRV_TOUCH_MTCH6301_TASK_QUEUE* taskQueue;	Head of the task queue

DRV_HANDLE drvI2CHandle;	I2C bus driver handle
DRV_TOUCH_POSITION_STATUS touchStatus;	Touch status

Description

MTCH6301 Driver Instance Object.

This data structure maintains the MTCH6301 driver instance object. The object exists once per hardware instance.

Remarks

None.

DRV_TOUCH_MTCH6301_TASK_QUEUE Structure

Defines the MTCH6301 Touch Controller driver task data structure.

File

drv_mtch6301.h

C

```
typedef struct {
  bool inUse;
  DRV_TOUCH_MTCH6301_TASK_STATE taskState;
  DRV_I2C_BUFFER_HANDLE drvI2CReadBufferHandle;
  uint8_t drvI2CReadFrameData[DRV_TOUCH_MTCH6301_I2C_READ_FRAME_SIZE];
} DRV_TOUCH_MTCH6301_TASK_QUEUE;
```

Members

Members	Description
bool inUse;	Flag denoting the allocation of task
DRV_TOUCH_MTCH6301_TASK_STATE taskState;	Enum maintaining the task state
DRV_I2C_BUFFER_HANDLE drvI2CReadBufferHandle;	I2C Buffer handle
uint8_t drvI2CReadFrameData[DRV_TOUCH_MTCH6301_I2C_READ_FRAME_SIZE];	Response to Read Touch Input Command • Response = { MTCH6301 Read Address, • Input Data Size, • Touch Id, Pen status, • Touch X coordinate (0 to 6), • Touch X coordinate (7 to 11), • Touch Y coordinate (0 to 6), • Touch Y coordinate (7 to 11) }

Description

MTCH6301 Touch Controller driver task data structure.

This data type defines the data structure maintaing task context in the task queue. The inUse flag denotes the task context allocation for a task. The enum variable taskState maintains the current task state. The I2C buffer handle drvI2CReadBufferHandle maintains the I2C driver buffer handle returned by the I2C driver read request. The byte array variable drvI2CReadFrameData maintains the I2C frame data sent by MTCH6301 after a successful read request.

Remarks

None.

DRV_TOUCH_MTCH6301_TASK_STATE Enumeration

Enumeration defining MTCH6301 touch controller driver task state.

File

drv_mtch6301.h

C

```
typedef enum {
  DRV_TOUCH_MTCH6301_TASK_STATE_INIT = 0,
  DRV_TOUCH_MTCH6301_TASK_STATE_READ_INPUT,
```

```
  DRV_TOUCH_MTCH6301_TASK_STATE_DECODE_INPUT,
  DRV_TOUCH_MTCH6301_TASK_STATE_DONE
} DRV_TOUCH_MTCH6301_TASK_STATE;
```

Members

Members	Description
DRV_TOUCH_MTCH6301_TASK_STATE_INIT = 0	Task initialize state
DRV_TOUCH_MTCH6301_TASK_STATE_READ_INPUT	Task read touch input request state
DRV_TOUCH_MTCH6301_TASK_STATE_DECODE_INPUT	Task touch input decode state
DRV_TOUCH_MTCH6301_TASK_STATE_DONE	Task complete state

Description

MTCH6301 Touch Controller Driver Task State

This enumeration defines the MTCH6301 touch controller driver task state. The task state helps to synchronize the operations of initialization the the task, adding the read input task to the task queue once the touch controller notifies the available touch input and a decoding the touch input received.

Remarks

None.

DRV_TOUCH_MTCH6301_I2C_MASTER_READ_ID Macro

MTCH6301 input read, I2C address from where master reads touch input data.

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_I2C_MASTER_READ_ID 0x4B
```

Description

MTCH6301 Driver Module Master Input Read I2C address

This constant defines the MTCH6301 touch input read I2C address. This address is used as I2C address to read Touch input from MTCH6301 Touch controller.

Remarks

This constant should be used in place of hard-coded numeric literals.

This value is derived from device-specific data sheets.

DRV_TOUCH_MTCH6301_I2C_MASTER_WRITE_ID Macro

MTCH6301 command register write, I2C address where master sends the commands.

File

drv_mtch6301.h

C

```
#define DRV_TOUCH_MTCH6301_I2C_MASTER_WRITE_ID 0x4A
```

Description

MTCH6301 Driver Module Master Command Write I2C Address

This constant defines the MTCH6301 command register I2C write address. This address is used as I2C address to write commands into MTCH6301 Touch controller register.

Remarks

This constant should be used in place of hard-coded numeric literals. This value is derived from device-specific data sheets.

Files

Files

Name	Description
drv_mtch6301.h	Touch controller MTCH6301 Driver interface header file.
drv_mtch6301_config_template.h	MTCH6301 Touch Driver configuration template.

Description

This section lists the source and header files used by the MTCH6301 Touch Driver Library.

drv_mtch6301.h

Touch controller MTCH6301 Driver interface header file.

Enumerations

	Name	Description
	DRV_TOUCH_MTCH6301_MODULE_ID	Number of valid MTCH6301 driver indices.
	DRV_TOUCH_MTCH6301_TASK_STATE	Enumeration defining MTCH6301 touch controller driver task state.

Functions

	Name	Description
	DRV_TOUCH_MTCH6301_Close	Closes an opened instance of the MTCH6301 driver. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Deinitialize	Deinitializes the specified instance of the MTCH6301 driver module. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Initialize	Initializes the MTCH6301 instance for the specified driver index. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Open	Opens the specified MTCH6301 driver instance and returns a handle to it. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_ReadRequest	Sends a read request to I2C bus driver and adds the read task to queue. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Status	Provides the current status of the MTCH6301 driver module. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_Tasks	Maintains the driver's state machine and implements its task queue processing. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_TouchDataRead	Notifies the driver that the current touch data has been read
	DRV_TOUCH_MTCH6301_TouchGetX	Returns the x coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_TouchGetY	Returns the y coordinate of touch input. **Implementation:** Dynamic
	DRV_TOUCH_MTCH6301_TouchStatus	Returns the status of the current touch input.

Macros

	Name	Description
	DRV_TOUCH_MTCH6301_HANDLE_INVALID	Definition of an invalid handle.
	DRV_TOUCH_MTCH6301_I2C_MASTER_READ_ID	MTCH6301 input read, I2C address from where master reads touch input data.
	DRV_TOUCH_MTCH6301_I2C_MASTER_WRITE_ID	MTCH6301 command register write, I2C address where master sends the commands.
	DRV_TOUCH_MTCH6301_I2C_READ_FRAME_SIZE	I2C Frame size for reading MTCH6301 touch input.
	DRV_TOUCH_MTCH6301_INDEX_0	MTCH6301 driver index definitions.
	DRV_TOUCH_MTCH6301_INDEX_1	This is macro DRV_TOUCH_MTCH6301_INDEX_1.
	DRV_TOUCH_MTCH6301_INDEX_COUNT	Number of valid Touch controller MTCH6301 driver indices.

Structures

	Name	Description
	_DRV_MTCH6301_CLIENT_OBJECT	MTCH6301 Driver client object maintaining client data.
	DRV_TOUCH_MTCH6301_CLIENT_OBJECT	MTCH6301 Driver client object maintaining client data.
	DRV_TOUCH_MTCH6301_OBJECT	Defines the data structure maintaining MTCH6301 driver instance object.
	DRV_TOUCH_MTCH6301_TASK_QUEUE	Defines the MTCH6301 Touch Controller driver task data structure.

Types

	Name	Description
	DRV_TOUCH_MTCH6301_HANDLE	Touch screen controller MTCH6301 driver handle.

Description

Touch Controller MTCH6301 Driver Interface File

This header file describes the macros, data structure and prototypes of the touch controller MTCH6301 driver interface.

File Name

drv_mtch6301.c

drv_mtch6301_config_template.h

MTCH6301 Touch Driver configuration template.

Macros

	Name	Description
	DRV_MTCH6301_CALIBRATION_DELAY	Defines the calibration delay.
	DRV_MTCH6301_CALIBRATION_INSET	Defines the calibration inset.
	DRV_MTCH6301_CLIENTS_NUMBER	Selects the maximum number of clients.
	DRV_MTCH6301_INDEX	MTCH6301 static index selection.
	DRV_MTCH6301_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_MTCH6301_INTERRUPT_MODE	Controls operation of the driver in the interrupt or polled mode.
	DRV_MTCH6301_SAMPLE_POINTS	Define the sample points.
	DRV_MTCH6301_TOUCH_DIAMETER	Defines the touch diameter.

Description

MTCH6301 Touch Driver Configuration Template

This header file contains the build-time configuration selections for the MTCH6301 Touch Driver. This is the template file which give all possible configurations that can be made. This file should not be included in any project.

File Name

drv_mtch6301_config_template.h

Company

Microchip Technology Inc.

MTCH6303 Touch Driver Library

This topic describes the MTCH6303 Touch Driver Library.

Introduction

This library provides an interface to manage the MTCH6303 Touch Driver module on the Microchip family of microcontrollers in different modes of operation.

Description

The MPLAB Harmony MTCH6303 Touch Driver provides a high-level interface to the MTCH6303 touch controller device. This driver provides application routines to read the touch input data from the touch screen. The MTCH6303 device can notify the availability of touch input data through external interrupt. The MTCH6303 driver allows the application to map a controller pin as an external interrupt pin.

Currently, the MTCH6303 Touch Driver only supports non-gestural single-finger touch screen input.

Using the Library

This topic describes the basic architecture of the MTCH6303 Touch Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_mtch6303_static.h`

The interface to the MTCH6303 Touch Driver Library is defined in the `drv_mtch6303_static.h` header file. This file is generated by the MPLAB Harmony Configurator (MHC) during application code generation. It is included in `system_definitions.h` by MHC during application code generation. Any configuration macros required for MTCH6303 Driver are included in `system_config.h` by MHC during code generation. Any C language source (`.c`) file that uses the MTCH6303 Touch Driver Library should include `system_config.h` and `system_definitions.h`, respectively.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the MTCH6303 Touch Driver Library on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

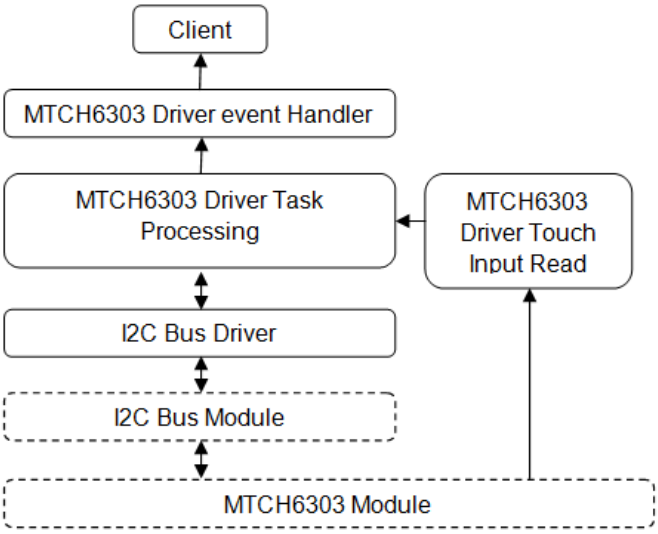
The MTCH6303 Touch Driver has routines to perform the following operations:

- MTCH6303 register read and write
- MTCh6303 message read and write
- MTCH6303 touch input read
- Mapping of the touch input to screen resolution

The driver Initialization routine allows the application to initialize the driver. The driver must be initialized before it can be used by the application. Once the driver is initialized, the driver Open function allows retrieval of the client handle. If the client handle is valid, an event handler routine needs to be registered by the application. The MTCH6303 Touch Driver triggers an interrupt once touch input is available to be read from the MTCH6303 registers. A touch input Read function is called from the interrupt handler to initiate the touch input read task. An Event Handler function is called once the touch input read task is completed. A valid touch input will be available only after the event handler routine is triggered. The touch input must be read inside of the event handler function.

The touch input data is a raw value and needs to be mapped to the target screen resolution. At zero degree orientation, touch input is mapped on the x axis from zero at the left and the maximum value at the right. At zero degree orientation, touch input is mapped on the y axis from zero at the top and the maximum value at the bottom.

MTCH6303 Driver Abstraction Model



Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the MTCH6303 Touch Driver.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization, tasks, and status functions.
Client Setup Functions	Provides open, close, status, and other setup function.
Read and Write Functions	Provides functions to read and write to the MTCH6303 registers, messages, and touch data.
Miscellaneous Functions	Provides miscellaneous functions.

How the Library Works

The library provides interfaces to support:

- System functions, which provide system module interfaces, device initialization, deinitialization, task, and status functions
- Client setup functions, which provide client interfaces such as open, close and event handler registration
- Read and write functions, initiate the touch input or register or message read and write tasks
- Miscellaneous functions such as touch input map function are provided to process the raw touch input data

Configuring the Library

The configuration of the MTCH6303 Touch Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the MTCH6303 Touch Driver. Based on the selections made, the driver may support the selected features. These configuration settings will apply to all instances of the MTCH6303 Touch Driver.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the MTCH6303 Touch Driver Library.

Description

This section list the files that are available in the `/src` folder of the MTCH6303 Touch Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/touch/mtch6303`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_mtch6303_static.h`	Header file that exports the driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/drv_mtch6303_static.c`	Basic MTCH6303 Touch Driver implementation file.
`/src/drv_mtch6303_buffer_queue_i2c_static.c`	MTCH6303 I2C buffer queue implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
`/drv_mtch6303_buffer_queue_touch_static.c`	MTCH6303 message buffer queue implementation file.

Module Dependencies

The MTCH6303 Touch Driver Library depends on the following modules:

- Interrupt System Service Library
- Ports System Service Library
- I2C Driver Library

Library Interface

a) System Functions

	Name	Description
	DRV_MTCH6303_Deinitialize	Deinitializes the instance of the MTCH6303 driver module.
	DRV_MTCH6303_Initialize	Initializes the MTCH6303 static single instance.
	DRV_MTCH6303_Status	Gets the current status of the MTCH6303 driver module.
	DRV_MTCH6303_Tasks	Maintains the driver's register read/write state machine and implements its ISR.

b) Client Setup Functions

	Name	Description
	DRV_MTCH6303_Close	Closes an opened-instance of the MTCH6303 driver.
	DRV_MTCH6303_ErrorGet	This function returns the error associated with the last client request.
	DRV_MTCH6303_Open	Opens the MTCH6303 driver instance and returns a handle to it.

c) Read and Write Functions

	Name	Description
	DRV_MTCH6303_AddRegisterRead	Schedules a non-blocking register read request to read I2C accessible MTCH6303 registers.
	DRV_MTCH6303_AddRegisterWrite	Schedule a non-blocking driver register write operation to write I2C accessible MTCH6303 registers.
	DRV_MTCH6303_TOUCH_AddMessageCommandWrite	Schedule a non-blocking driver command message write operation to write command message to MTCH6303 registers.
	DRV_MTCH6303_TOUCH_AddMessageReportRead	Schedules a non-blocking report message read request to read the report message from MTCH6303 device.
	DRV_MTCH6303_TOUCH_AddTouchInputRead	Schedules a non-blocking read buffer request to read touch input from MTCH6303.
	DRV_MTCH6303_TOUCH_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued message transfers have finished.
	DRV_MTCH6303_TOUCH_Tasks	Maintains the driver's message state machine and implements its ISR.
	DRV_MTCH6303_TouchInputMap	Maps the raw touch input to display resolution.
	DRV_MTCH6303_TouchInputRead	Schedules a non-blocking read buffer request to read touch input from MTCH6303.
	DRV_MTCH6303_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

e) Data Types and Constants

	Name	Description
	DRV_MTCH6303_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_MTCH6303_TOUCH_NUM_INPUTS	Definition of number of touch input packets can be identified by MTCH6303.
	DRV_MTCH6303_BUFFER_EVENT	Lists the different conditions that happens during a buffer transfer.
	DRV_MTCH6303_BUFFER_EVENT_HANDLER	Points to a callback after completion of an register read -write or message stream read - write.
	DRV_MTCH6303_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.
	DRV_MTCH6303_CLIENT_STATUS	Defines the client-specific status of the MTCH6303 driver.
	DRV_MTCH6303_ERROR	Defines the possible errors that can occur during driver operation.
	DRV_MTCH6303_TOUCH_BUFFER_EVENT	Lists the different conditions that happens during a touch message buffer transfer.
	DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER	Points to a callback after completion of an message report read or message command write.

	DRV_MTCH6303_TOUCH_BUFFER_HANDLE	Handle identifying a read or write touch message buffer passed to the driver.
	DRV_MTCH6303_TOUCH_DATA	Defines MTCH6303 I2C Touch Data
	DRV_MTCH6303_TOUCH_INPUT	Defines MTCH6303 Touch Input Packet
	DRV_MTCH6303_TOUCH_MESSAGE	Defines MTCH6303 Touch Message.
	DRV_MTCH6303_TOUCH_MESSAGE_HEADER	Defines Touch Message Header.
	DRV_MTCH6303_TOUCH_NIBBLE_0	Defines the I2C Nibble 0 of MTCH6303 Touch input packet.
	DRV_MTCH6303_TOUCH_STATUS	Defines the I2C touch status register bits
	DRV_TOUCH_MTCH6303_MSG_ID	List of report or command message identification.
	DRV_TOUCH_MTCH6303_I2C_REGISTER_MAP	List of MTCH6303 I2C Accessible Register Identification.

Description

This section describes the API functions of the MTCH6303 Touch Driver library.

Refer to each section for a detailed description.

a) System Functions

DRV_MTCH6303_Deinitialize Function

Deinitializes the instance of the MTCH6303 driver module.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_Deinitialize();
```

Returns

None.

Description

Deinitializes the instance of the MTCH6303 driver module, disabling its operation. Invalidates all the internal data.

Remarks

once the initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. this routine will NEVER block waiting for hardware.

Preconditions

Function DRV_MTCH6303_Initialize should have been called before calling this function.

Example

```
SYS_STATUS    status;

DRV_MTCH6303_Deinitialize();

status = DRV_MTCH6303_Status();
if(SYS_MODULE_DEINITIALIZED != status)
{
    //check again later if you need to know
    //when the driver is deinitialized
}
```

Function

void DRV_MTCH6303_Deinitialize(void)

DRV_MTCH6303_Initialize Function

Initializes the MTCH6303 static single instance.

File

drv_mtch6303.h

C

```
SYS_MODULE_OBJ DRV_MTCH6303_Initialize();
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the MTCH6303 static driver instance. It makes the instance ready for a client to open and use it. The instance parameters are initialized by values set by MPLAB Harmony Configurator.

Preconditions

None.

Example

```
// The following code snippet shows an example MTCH6303 driver initialization.

SYS_MODULE_OBJ    objectHandle;

objectHandle = DRV_MTCH6303_Initialize();
if( SYS_MODULE_OBJ_INVALID == objectHandle )
{
   // Handle error
}
Remarks:   This routine must be called before any other MTCH6303 routine is called.

   This routine should only be called once during system initialization unless
   DRV_MTCH6303_Deinitialize is called to deinitialize the driver instance.
   This routine will NEVER block for hardware access.
```

Function

SYS_MODULE_OBJ DRV_MTCH6303_Initialize (void)

DRV_MTCH6303_Status Function

Gets the current status of the MTCH6303 driver module.

File

drv_mtch6303.h

C

```
SYS_STATUS DRV_MTCH6303_Status();
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another.

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized.

Description

This routine provides the current status of the MTCH6303 driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_MTCH6303_Initialize should have been called before calling this function.

Example

```
SYS_STATUS mtch6303Status;

mtch6303Status = DRV_MTCH6303_Status();
if(SYS_STATUS_READY == mtch6303Status)
```

```
{
    // This means the driver can be opened using the
    // DRV_MTCH6303_Open() function.
}
```

Function

SYS_STATUS DRV_MTCH6303_Status(void)

DRV_MTCH6303_Tasks Function

Maintains the driver's register read/write state machine and implements its ISR.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_Tasks();
```

Returns

None.

Description

This routine is used to maintain the driver's register read/write state machine and implement its ISR for interrupt-driven implementations. In interrupt mode, this function is called in I2C Driver event Handler routine. The I2C Driver event Handler routine is registered by MTCH6303 event Handler register routine.

Remarks

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

Function DRV_MTCH6303_Initialize should have been called before calling this function. It also needs registration of the MTCH6303 Driver event handler routine.

Function

void DRV_MTCH6303_Tasks(void)

b) Client Setup Functions

DRV_MTCH6303_Close Function

Closes an opened-instance of the MTCH6303 driver.

File

drv_mtch6303.h

C

```
DRV_MTCH6303_CLIENT_STATUS DRV_MTCH6303_Close();
```

Returns

DRV_MTCH6303_CLIENT_STATUS_ERROR - if driver fails to remove buffer objects from queue.

DRV_MTCHC6303_CLIENT_STATUS_CLOSED - client is successfully closed

Description

This routine closes an opened-instance of the MTCH6303 driver. Any buffers in the driver queue that were submitted by this client will be removed. DRV_MTCH6303_Open must be called to before using the driver again.

Remarks

The driver will abort any ongoing operations when this routine is called.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called. DRV_MTCH6303_Open must have been called.

Example

```
DRV_MTH6303_CLIENT_STATUS mtch6303Status;

mtch6303Status = DRV_MTCH6303_Close()
if( DRV_MTCH6303_CLIENT_STATUS_ERROR == mtch6303Status )
{
    //retry closing the driver client
}
```

Function

DRV_MTCH6303_CLIENT_STATUS DRV_MTCH6303_Close (void)

DRV_MTCH6303_ErrorGet Function

This function returns the error associated with the last client request.

File

drv_mtch6303.h

C

```
DRV_MTCH6303_ERROR DRV_MTCH6303_ErrorGet();
```

Returns

DRV_MTCH6303_ERROR_NONE - no error

Description

This function returns the error associated with the last client request.

Remarks

This routine always return DRV_MTCH6303_ERROR_NONE the client error is currently not updated by any of the MTCH6303 operations API's.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called. DRV_MTCH6303_Open must have been called to open a device client.

Function

DRV_MTCH6303_ERROR DRV_MTCH6303_ErrorGet (void)

DRV_MTCH6303_Open Function

Opens the MTCH6303 driver instance and returns a handle to it.

File

drv_mtch6303.h

C

```
DRV_HANDLE DRV_MTCH6303_Open();
```

Returns

If successful, the routine returns a valid open-instance handle. If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the driver is not ready to be opened, typically when the initialize routine has not completed execution.
- if the bus driver fails to open
- if the client is trying to open the driver but driver has been opened exclusively by another client.

Description

This routine opens the specified MTCH6303 driver instance and provides a handle.

Remarks

The handle returned is valid until the DRV_MTCH6303_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application.

Preconditions

Function DRV_MTCH6303_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_MTCH6303_Open( );
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Function

DRV_HANDLE DRV_MTCH6303_Open { void }

c) Read and Write Functions

DRV_MTCH6303_AddRegisterRead Function

Schedules a non-blocking register read request to read I2C accessible MTCH6303 registers.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_AddRegisterRead(DRV_MTCH6303_BUFFER_HANDLE * bufferHandle, uint8_t source, size_t nBytes,
uint8_t * destination);
```

Returns

None.

Description

This function schedules a non-blocking register read request to read I2C accessible MTCH6303 registers. The function returns with a valid buffer handle in the bufferHandle argument if the register read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_MTCH6303_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_MTCH6303_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or DRV_MTCH6303_BUFFER_EVENT_ERROR event if the buffer was not processed successfully. The register data is collected into destination and can be read once a buffer event complete is reported. A event handler is called on buffer event complete where the register data must be read from destination.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

DRV_MTCH6303_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t registerData[NUM_REGISTERS];
DRV_MTCH6303_BUFFER_HANDLE bufferHandle;

// Client registers an event handler with driver

DRV_MTCH6303_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                    (uintptr_t)&myAppObj);

DRV_MTCH6303_AddRegisterRead( &bufferHandle,
                              DRV_MTCH6303_REG_TOUCH_STATUS,
                              NUM_REGISTERS,
                              &registerData );
```

```
if(DRV_MTCH6303_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_BUFFER_EVENT event,
                                     DRV_MTCH6303_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
Remarks:   None.
```

Parameters

Parameters	Description
bufferHandle	Handle to the buffer scheduled.
source	Register index.
nBytes	Number of registers to be read, starting from source.
destination	buffer collecting register data.

Function

void DRV_MTCH6303_AddRegisterRead(DRV_MTCH6303_BUFFER_HANDLE * bufferHandle,

uint8_t source,

size_t nBytes,

uint8_t * destination)

DRV_MTCH6303_AddRegisterWrite Function

Schedule a non-blocking driver register write operation to write I2C accessible MTCH6303 registers.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_AddRegisterWrite(DRV_MTCH6303_BUFFER_HANDLE * bufferHandle, uint8_t destination, size_t
nBytes, uint8_t * source);
```

Returns

None.

Description

This function schedules a non-blocking register write request to write I2C accessible MTCH6303 registers. The function returns with a valid buffer handle in the bufferHandle argument if the register write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be

modified. The function returns DRV_MTCH6303_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the write queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_MTCH6303_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or DRV_MTCH6303_BUFFER_EVENT_ERROR event if the buffer was not processed successfully. A event handler is called on buffer event complete where the application data is written to the I2C accessible MTCH6303 Register.

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

DRV_MTCH6303_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
uint8_t registerData[NUM_REGISTERS];
DRV_MTCH6303_BUFFER_HANDLE bufferHandle;

// Client registers an event handler with driver

DRV_MTCH6303_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                    (uintptr_t)&myAppObj);

DRV_MTCH6303_AddRegisterWrite( &bufferHandle,
                               DRV_MTCH6303_REG_TOUCH_STATUS,
                               NUM_REGISTERS,
                               &registerData );

if(DRV_MTCH6303_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_BUFFER_EVENT event,
                                     DRV_MTCH6303_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferHandle	Pointer to an argument that will contain the return buffer handle.
destination	Index to the start of destination register list.
nBytes	number of registers.
source	pointer to the data to be written to the register.

Function

void DRV_MTCH6303_AddRegisterWrite(DRV_MTCH6303_BUFFER_HANDLE * bufferHandle,
uint8_t destination,
size_t nBytes,
uint8_t * source)

DRV_MTCH6303_TOUCH_AddMessageCommandWrite Function

Schedule a non-blocking driver command message write operation to write command message to MTCH6303 registers.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_TOUCH_AddMessageCommandWrite(DRV_MTCH6303_TOUCH_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_MESSAGE * messageCmd, size_t messageSize);
```

Returns

None.

Description

This function schedules a non-blocking command message write request to write command message to MTCH6303. The function returns with a valid buffer handle in the bufferHandle argument if the register command message write request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. While the request is in the queue, the application message buffer is owned by the driver and should not be modified. The function returns DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the message write queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR event if the buffer was not processed successfully. A event handler is called on buffer event complete where the application command message is written to MTCH6303.

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

DRV_MTCH6303_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
DRV_MTCH6303_TOUCH_MESSAGE messageCommand;
DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle;

// Client registers an event handler with driver

DRV_MTCH6303_TOUCH_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                          (uintptr_t)&myAppObj);

DRV_MTCH6303_TOUCH_AddMessageCommandWrite( &bufferHandle,
                                           &messageCommand,
```

```
                                        MY_MESSAGE_SIZE );

if(DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_TOUCH_BUFFER_EVENT event,
                                     DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferHandle	Pointer to an argument that will contain the return buffer handle.
messageCmd	command message to write to MTCH6303.
messageSize	command message size. It includes message header and payload size.

Function

void DRV_MTCH6303_TOUCH_AddMessageCommandWrite
(DRV_MTCH6303_TOUCH_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_MESSAGE * messageCmd,
size_t messageSize)

DRV_MTCH6303_TOUCH_AddMessageReportRead Function

Schedules a non-blocking report message read request to read the report message from MTCH6303 device.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_TOUCH_AddMessageReportRead(DRV_MTCH6303_TOUCH_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_MESSAGE * messageRep, size_t messageSize);
```

Returns

None.

Description

This function schedules a non-blocking report message read request to read the report message from MTCH6303 device. The function returns with a valid buffer handle in the bufferHandle argument if the register read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR event if the buffer was not processed successfully. The register data is collected into destination and can be read once a buffer event complete is reported. A event handler is called on buffer event complete where the register data must be read from destination.

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

DRV_MTCH6303_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
DRV_MTCH6303_TOUCH_MESSAGE messageReport;
DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle;

// Client registers an event handler with driver

DRV_MTCH6303_TOUCH_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                          (uintptr_t)&myAppObj);

DRV_MTCH6303_TOUCH_AddMessageReportRead( &bufferHandle,
                                         &messageReport,
                                         MY_MESSAGE_SIZE );

if(DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_TOUCH_BUFFER_EVENT event,
                                     DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferHandle	Handle to the buffer scheduled.
messageRep	report message buffer.
messageSize	report message size. It includes message header and payload size.

Function

void DRV_MTCH6303_TOUCH_AddMessageReportRead
(DRV_MTCH6303_TOUCH_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_MESSAGE * messageRep,
size_t messageSize)

DRV_MTCH6303_TOUCH_AddTouchInputRead Function

Schedules a non-blocking read buffer request to read touch input from MTCH6303.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_TOUCH_AddTouchInputRead(DRV_MTCH6303_TOUCH_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_DATA * touchData);
```

Returns

None.

Description

This function schedules a non-blocking read buffer request to read touch input from MTCH6303. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR event if the buffer was not processed successfully. The touch data is collected into touchData and can be read once a buffer event complete is reported. A event handler is called on buffer event complete where the touch data must be read from touchData.

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

DRV_MTCH6303_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
DRV_MTCH6303_TOUCH_DATA touchData;
DRV_MTCH6303_BUFFER_HANDLE bufferHandle;

// Client registers an event handler with driver

DRV_MTCH6303_TOUCH_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                          (uintptr_t)&myAppObj);

DRV_MTCH6303_TOUCH_AddTouchInputRead( &bufferHandle, &touchData );

if(DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID == bufferHandle)
{
```

```
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_TOUCH_BUFFER_EVENT event,
                                     DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferHandle	Handle to the buffer scheduled.
touchData	Buffer collecting touch data.

Function

void DRV_MTCH6303_TOUCH_AddTouchInputRead
(DRV_MTCH6303_TOUCH_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_DATA * touchData)

DRV_MTCH6303_TOUCH_BufferEventHandlerSet Function

Allows a client to identify a buffer event handling function for the driver to call back when queued message transfers have finished.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_TOUCH_BufferEventHandlerSet(const DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify a message event handling function for the driver to call back when queued message transfers have finished. When a client calls either the DRV_MTCH6303_TOUCH_AddTouchInputRead, DRV_MTCH6303_TOUCH_AddMessageReportRead or DRV_MTCH6303_TOUCH_AddMessageCommandWrite function, it is provided with a handle identifying the message that was added to the driver's message queue. The driver will pass this handle back to the client by calling "eventHandler" function when the message transfer has completed.

The event handler should be set before the client performs any "message add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;
DRV_MTCH6303_TOUCH_MESSAGE messageReport;
DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle;

// myMTCH6303Handle is the handle returned
// by the DRV_MTCH6303_Open function.

// Client registers an event handler with driver. This is done once

DRV_MTCH6303_TOUCH_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                  (uintptr_t)&myAppObj );

DRV_MTCH6303_TOUCH_AddMessageReportRead( &bufferHandle, &messageReport );

if(DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_TOUCH_BUFFER_EVENT event,
                                     DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_MTCH6303_TOUCH_BufferEventHandlerSet

(

const DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER eventHandler,

const uintptr_t context

)

DRV_MTCH6303_TOUCH_Tasks Function

Maintains the driver's message state machine and implements its ISR.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_TOUCH_Tasks();
```

Returns

None.

Description

This routine is used to maintain the driver's message state machine and implement its ISR for interrupt-driven implementations. In interrupt mode, this function is called in I2C Driver event Handler routine. The I2C Driver event Handler routine is registered by MTCH6303 Touch event Handler register routine.

Remarks

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

Function DRV_MTCH6303_Initialize should have been called before calling this function. It also needs registration of the MTCH6303 Driver Touch event handler routine.

Function

void DRV_MTCH6303_TOUCH_Tasks(void)

DRV_MTCH6303_TouchInputMap Function

Maps the raw touch input to display resolution.

File

drv_mtch6303.h

C

```
inline uint16_t DRV_MTCH6303_TouchInputMap(uint16_t touchValue, uint16_t dispResolution);
```

Returns

This function returns the raw touch input mapped to display resolution in form of number of pixels.

Description

This function maps the raw touch input to display resolution. Raw touch input touchValue is obtained from the individual x or y value of DRV_MTCH6303_TOUCH_DATA. Raw touch value varies from 0 to 0x7FFF. The displayResolution is either horizontal or vertical resolution of the display in pixels. The function returns the raw touch input mapped to display resolution in form of number of pixels.

Remarks

None.

Preconditions

None.

Example

```
// Display with resolution 800 x 480
#define DISP_HOR_RESOUTION  800
#define DISP_VER_RESOLUTION 480

DRV_MTCH6303_TOUCH_DATA touchData;
uint16_t rawTouchX;
uint16_t rawTouchY;
uint16_t touchX;
uint16_t touchY;
```

```
// map 0th touch packet to display resolution
rawTouchX = touchData.touch[0].x;
rawTouchY = touchData.touch[0].y;

// map raw touch input in x direction to display horizontal resolution
touchX = DRV_MTCH6303_TouchInputMap( rawTouchX, DISP_HOR_RESOLUTION );

// map raw touch input in y direction to display vertical resolution
touchY = DRV_MTCH6303_TouchInputMap( rawTouchY, DISP_VER_RESOLUTION );

// use touchX and touchY as input to graphics objects.
```

Parameters

Parameters	Description
touchValue	raw touch input either in x or y direction (0 - 0x7FFF).
dispResolution	display resolution specifying either width or height of the display in pixels.

Function

uint16_t DRV_MTCH6303_TouchInputMap(uint16_t touchValue, uint16_t dispResolution)

DRV_MTCH6303_TouchInputRead Function

Schedules a non-blocking read buffer request to read touch input from MTCH6303.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_TouchInputRead(DRV_MTCH6303_BUFFER_HANDLE * bufferHandle, DRV_MTCH6303_TOUCH_DATA *
touchData);
```

Returns

None.

Description

This function schedules a non-blocking read buffer request to read touch input from MTCH6303. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance queue and returns immediately. The function returns DRV_MTCH6303_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_MTCH6303_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or DRV_MTCH6303_BUFFER_EVENT_ERROR event if the buffer was not processed successfully. The touch data is collected into touchData and can be read once a buffer event complete is reported. A event handler is called on buffer event complete where the touch data must be read from touchData.

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

DRV_MTCH6303_Open must have been called to obtain a valid opened device handle.

Example

```
MY_APP_OBJ myAppObj;
DRV_MTCH6303_TOUCH_DATA touchData;
DRV_MTCH6303_BUFFER_HANDLE bufferHandle;

// Client registers an event handler with driver

DRV_MTCH6303_BufferEventHandlerSet( APP_MTCH6303BufferEventHandler,
                                    (uintptr_t)&myAppObj);
```

```
DRV_MTCH6303_TouchInputRead( &bufferHandle, &touchData );

if(DRV_MTCH6303_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandler( DRV_MTCH6303_BUFFER_EVENT event,
                                     DRV_MTCH6303_BUFFER_HANDLE bufferHandle,
                                     uintptr_t contextHandle )
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferHandle	Handle to the buffer scheduled.
touchData	Buffer collecting touch data.

Function

void DRV_MTCH6303_TouchInputRead(DRV_MTCH6303_BUFFER_HANDLE * bufferHandle,
DRV_MTCH6303_TOUCH_DATA * touchData)

DRV_MTCH6303_BufferEventHandlerSet Function

Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

File

drv_mtch6303.h

C

```
void DRV_MTCH6303_BufferEventHandlerSet(const DRV_MTCH6303_BUFFER_EVENT_HANDLER eventHandler, const
uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls either the DRV_MTCH6303_TouchInputRead, DRV_MTCH6303_AddRegisterRead or DRV_MTCH6303_AddRegisterWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

None.

Preconditions

The DRV_MTCH6303_Initialize routine must have been called and the DRV_MTCH6303_Status must have returned SYS_STATUS_READY.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];

// myMTCH6303Handle is the handle returned
// by the DRV_MTCH6303_Open function.

// Client registers an event handler with driver. This is done once

DRV_MTCH6303_BufferEventHandlerSet( APP_MTCH6303BufferEventHandle,
                                    (uintptr_t)&myAppObj );

DRV_MTCH6303_AddRegisterRead( &bufferHandle
                              DRV_MTCH6303_REG_TOUCH_STATUS,
                              MY_BUFFER_SIZE,
                              &mybuffer);

if(DRV_MTCH6303_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_MTCH6303BufferEventHandle( DRV_MTCH6303_BUFFER_EVENT event,
                                    DRV_MTCH6303_BUFFER_HANDLE handle,
                                    uintptr_t context)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_MTCH6303_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_MTCH6303_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_MTCH6303_BufferEventHandlerSet
(
const DRV_MTCH6303_BUFFER_EVENT_HANDLER eventHandler,
const uintptr_t context
)

d) Miscellaneous Functions

e) Data Types and Constants

DRV_MTCH6303_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_mtch6303.h

C

```
#define DRV_MTCH6303_BUFFER_HANDLE_INVALID
```

Description

MTCH6303 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_MTCH6303_AddRegisterRead, DRV_MTCH6303_AddRegisterWrite or DRV_MTCH6303_TouchInputRead functions if the request was not successful.

Remarks

None

DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_mtch6303.h

C

```
#define DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID
```

Description

MTCH6303 Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_MTCH6303_TOUCH_AddMessageReportRead, DRV_MTCH6303_TOUCH_AddMessageCommandWrite or DRV_MTCH6303_TOUCH_AddTouchInputRead functions if the request was not successful.

Remarks

None

DRV_MTCH6303_TOUCH_NUM_INPUTS Macro

Definition of number of touch input packets can be identified by MTCH6303.

File

drv_mtch6303.h

C

```
#define DRV_MTCH6303_TOUCH_NUM_INPUTS 0xA
```

Description

MTCH6303 Number of touch input packets

MTCH6303 supports multi-touch and can identify upto 10 different touch input packets.

Remarks

None.

DRV_MTCH6303_BUFFER_EVENT Enumeration

Lists the different conditions that happens during a buffer transfer.

File

drv_mtch6303.h

C

```
typedef enum {
  DRV_MTCH6303_BUFFER_EVENT_COMPLETE,
  DRV_MTCH6303_BUFFER_EVENT_ERROR,
  DRV_MTCH6303_BUFFER_EVENT_ABORT
} DRV_MTCH6303_BUFFER_EVENT;
```

Members

Members	Description
DRV_MTCH6303_BUFFER_EVENT_COMPLETE	Event buffer transfer complete
DRV_MTCH6303_BUFFER_EVENT_ERROR	Event buffer transfer error
DRV_MTCH6303_BUFFER_EVENT_ABORT	Event buffer transfer abort

Description

MTCH6303 Buffer Events

This enumeration identifies the different conditions that can happen during a buffer transaction. Callbacks can be made with the appropriate buffer condition passed as a parameter to execute the desired action.

The values act like flags and multiple flags can be set.

Remarks

None.

DRV_MTCH6303_BUFFER_EVENT_HANDLER Type

Points to a callback after completion of an register read -write or message stream read - write.

File

drv_mtch6303.h

C

```
typedef void (* DRV_MTCH6303_BUFFER_EVENT_HANDLER)(DRV_MTCH6303_BUFFER_EVENT event,
DRV_MTCH6303_BUFFER_HANDLE bufferHandle, uintptr_t context);
```

Description

MTCH6303 Buffer Event Callback

This type identifies the MTCH6303 Buffer Event. It allows the client driver to register a callback using DRV_MTCH6303_BUFFER_EVENT_HANDLER. By using this mechanism, the driver client will be notified at the completion of the corresponding transfer.

Remarks

A transfer can be composed of various transfer segments. Once a transfer is completed the driver will call the client registered transfer callback.

The callback could be called from ISR context and should be kept as short as possible. It is meant for signaling and it should not be blocking.

Parameters

Parameters	Description
DRV_MTCH6303_BUFFER_EVENT	Status of MTCH6303 transfer

bufferHandle	Handle that identifies the particular Buffer Object
context	pointer to the object to be processed.

Function

void (*DRV_MTCH6303_BUFFER_EVENT_HANDLER) (DRV_MTCH6303_BUFFER_EVENT event,
DRV_MTCH6303_BUFFER_HANDLE bufferHandle,
uintptr_t context)

DRV_MTCH6303_BUFFER_HANDLE Type

Handle identifying a read or write buffer passed to the driver.

File

drv_mtch6303.h

C

```
typedef uintptr_t DRV_MTCH6303_BUFFER_HANDLE;
```

Description

MTCH6303 Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_MTCH6303_AddRegisterRead, DRV_MTCH6303_AddRegisterWrite or DRV_MTCH6303_TouchInputRead functions. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from these functions is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None

DRV_MTCH6303_CLIENT_STATUS Enumeration

Defines the client-specific status of the MTCH6303 driver.

File

drv_mtch6303.h

C

```
typedef enum {
  DRV_MTCH6303_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR,
  DRV_MTCH6303_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED,
  DRV_MTCH6303_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY,
  DRV_MTCH6303_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY
} DRV_MTCH6303_CLIENT_STATUS;
```

Members

Members	Description
DRV_MTCH6303_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR	An error has occurred.
DRV_MTCH6303_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED	The driver is closed, no operations for this client are ongoing, and/or the given handle is invalid.
DRV_MTCH6303_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY	The driver is currently busy and cannot start additional operations.
DRV_MTCH6303_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY	The module is running and ready for additional operations

Description

MTCH6303 Client-Specific Driver Status

This enumeration defines the client-specific status codes of the MTCH6303 driver.

Remarks

Returned by the DRV_MTCH6303_ClientStatus function.

DRV_MTCH6303_ERROR Enumeration

Defines the possible errors that can occur during driver operation.

File

drv_mtch6303.h

C

```
typedef enum {
} DRV_MTCH6303_ERROR;
```

Description

MTCH6303 Driver Errors.

This data type defines the possible errors that can occur when occur during MTCH6303 driver operation. These values are returned by DRV_MTCH6303_ErrorGet function.

Remarks

None

DRV_MTCH6303_TOUCH_BUFFER_EVENT Enumeration

Lists the different conditions that happens during a touch message buffer transfer.

File

drv_mtch6303.h

C

```
typedef enum {
  DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE,
  DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR,
  DRV_MTCH6303_TOUCH_BUFFER_EVENT_ABORT
} DRV_MTCH6303_TOUCH_BUFFER_EVENT;
```

Members

Members	Description
DRV_MTCH6303_TOUCH_BUFFER_EVENT_COMPLETE	Event touch message buffer transfer complete
DRV_MTCH6303_TOUCH_BUFFER_EVENT_ERROR	Event touch message buffer transfer error
DRV_MTCH6303_TOUCH_BUFFER_EVENT_ABORT	Event touch message buffer transfer abort

Description

MTCH6303 Touch Message Buffer Events

This enumeration identifies the different conditions that can happen during a touch message buffer transaction. Callbacks can be made with the appropriate touch message buffer condition passed as a parameter to execute the desired action.

The values act like flags and multiple flags can be set.

Remarks

None.

DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER Type

Points to a callback after completion of an message report read or message command write.

File

drv_mtch6303.h

C

```
typedef void (* DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER)(DRV_MTCH6303_TOUCH_BUFFER_EVENT event,
DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle, uintptr_t context);
```

Description

MTCH6303 Touch Buffer Event Callback

This type identifies the MTCH6303 Touch Buffer Event. It allows the client driver to register a callback using DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER. By using this mechanism, the driver client will be notified at the completion of the corresponding transfer.

Remarks

A transfer can be composed of various transfer segments. Once a transfer is completed the driver will call the client registered transfer callback.

The callback could be called from ISR context and should be kept as short as possible. It is meant for signaling and it should not be blocking.

Parameters

Parameters	Description
DRV_MTCH6303_TOUCH_BUFFER_EVENT	Status of MTCH6303 touch message transfer
bufferHandle	Handle that identifies the particular Buffer Object
context	pointer to the object to be processed.

Function

void (*DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER) (DRV_MTCH6303_TOUCH_BUFFER_EVENT event,
DRV_MTCH6303_TOUCH_BUFFER_HANDLE bufferHandle,
uintptr_t context)

DRV_MTCH6303_TOUCH_BUFFER_HANDLE Type

Handle identifying a read or write touch message buffer passed to the driver.

File

drv_mtch6303.h

C

```
typedef uintptr_t DRV_MTCH6303_TOUCH_BUFFER_HANDLE;
```

Description

MTCH6303 Driver Touch Message Queue Buffer Handle

A touch message buffer handle value is returned by a call to the DRV_MTCH6303_TOUCH_AddMessageReportRead, DRV_MTCH6303_TOUCH_AddMessageCommandWrite or DRV_MTCH6303_TOUCH_AddTouchInputRead. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from these functions is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None.

DRV_MTCH6303_TOUCH_DATA Structure

Defines MTCH6303 I2C Touch Data

File

drv_mtch6303.h

C

```
typedef struct {
  uint8_t i2cReadAddr;
  DRV_MTCH6303_TOUCH_STATUS status;
  DRV_MTCH6303_TOUCH_INPUT touch[ DRV_MTCH6303_TOUCH_NUM_INPUTS ];
} DRV_MTCH6303_TOUCH_DATA;
```

Members

Members	Description
uint8_t i2cReadAddr;	Dummy I2C Read Address required for bitbang driver
DRV_MTCH6303_TOUCH_STATUS status;	MTCH6303 Touch Status
DRV_MTCH6303_TOUCH_INPUT touch[DRV_MTCH6303_TOUCH_NUM_INPUTS];	MTCH6303 Touch Input array of size DRV_MTCH6303_TOUCH_NUM_INPUTS

Description

MTCH6303 I2C Touch Data

This structure defines MTCH6303 I2C Touch Data. The structure DRV_MTCH6303_TOUCH_DATA is passed to API's DRV_MTCH6303_AddRegisterRead or DRV_MTCH6303_TOUCH_AddTouchInputRead. The API's will update the structure with touch input.

Remarks

It is packed to form structure of size 62 bytes. The structure member i2cReadAddr is only applicable if the I2C driver is of type bitbang. Otherwise the variable required to be commented out.

DRV_MTCH6303_TOUCH_INPUT Structure

Defines MTCH6303 Touch Input Packet

File

drv_mtch6303.h

C

```
typedef struct {
  DRV_MTCH6303_TOUCH_NIBBLE_0 nibble_0;
  uint8_t touchId;
  uint16_t x;
  uint16_t y;
} DRV_MTCH6303_TOUCH_INPUT;
```

Members

Members	Description
DRV_MTCH6303_TOUCH_NIBBLE_0 nibble_0;	MTCH6303 I2C Touch Input Packet Nibble 0
uint8_t touchId;	MTCH6303 I2C Touch Input Packet ID (0 - 16)
uint16_t x;	MTCH6303 I2C Touch Input Packet position x (0 - 0x7FFF)
uint16_t y;	MTCH6303 I2C Touch Input Packet position y (0 - 0x7FFF)

Description

MTCH6303 Touch Input Packet.

This structure defines the MTCH6303 Touch Input Packet.

Remarks

It is part of DRV_MTCH6303_TOUCH_DATA structure. It is packed to form structure of size 6 bytes.

DRV_MTCH6303_TOUCH_MESSAGE Structure

Defines MTCH6303 Touch Message.

File

drv_mtch6303.h

C

```
typedef struct {
  DRV_MTCH6303_TOUCH_MESSAGE_HEADER header;
  uint8_t payload[0x3E];
} DRV_MTCH6303_TOUCH_MESSAGE;
```

Members

Members	Description
DRV_MTCH6303_TOUCH_MESSAGE_HEADER header;	MTCH6303 Touch Message Header
uint8_t payload[0x3E];	MTCH6303 Touch Message payload. First byte of payload is of type DRV_TOUCH_MTCH6303_MSG_ID in case of first fragment of message. Otherwise the first byte acts as a normal payload.

Description

MTCH6303 Touch Message

This structure defines MTCH6303 Touch Message. The variable pointer of type DRV_MTCH6303_TOUCH_MESSAGE is passed to the API's

DRV_MTCH6303_TOUCH_AddMessageReportRead or DRV_MTCH6303_TOUCH_AddMessageCommandWrite.

Remarks

It is packed to form structure of size 63 bytes.

DRV_MTCH6303_TOUCH_MESSAGE_HEADER Structure

Defines Touch Message Header.

File

drv_mtch6303.h

C

```
typedef struct {
  uint32_t msgFragSize : 6;
  uint32_t continued : 1;
  uint32_t moreMessages : 1;
} DRV_MTCH6303_TOUCH_MESSAGE_HEADER;
```

Members

Members	Description
uint32_t msgFragSize : 6;	MTCH6303 Message Fragment Size. If Message Fragment size is 0x3F the Fragment is incomplete and uses up ALL of the parent transport layer packet.
uint32_t continued : 1;	MTCH6303 Message continued from last fragment if set to 1.
uint32_t moreMessages : 1;	MTCH6303 more messages to follow in this block if set to 1.

Description

MTCH6303 Touch Message Header

This structure defines Touch Message Header.

Remarks

It is part of structure DRV_MTCH6303_TOUCH_MESSAGE. It is packed to form structure of size 1 byte.

DRV_MTCH6303_TOUCH_NIBBLE_0 Structure

Defines the I2C Nibble 0 of MTCH6303 Touch input packet.

File

drv_mtch6303.h

C

```
typedef struct {
  uint32_t touchState : 1;
  uint32_t inRange : 1;
  uint32_t reserved : 6;
} DRV_MTCH6303_TOUCH_NIBBLE_0;
```

Members

Members	Description
uint32_t touchState : 1;	Touch packet available
uint32_t inRange : 1;	Touch packet in range
uint32_t reserved : 6;	Reserved bits

Description

MTCH6303 I2C Touch Input Packet Nibble 0

This structure defines the I2C Nibble 0 of MTCH6303 Touch input packet.

Remarks

It is part of DRV_MTCH6303_TOUCH_INPUT structure. It is packed to form structure of size 1 byte.

DRV_MTCH6303_TOUCH_STATUS Structure

Defines the I2C touch status register bits

File

drv_mtch6303.h

C

```
typedef struct {
  uint32_t nTouch : 4;
  uint32_t streamReady : 1;
  uint32_t gestureReady : 1;
  uint32_t gestICData : 1;
  uint32_t reserved : 1;
} DRV_MTCH6303_TOUCH_STATUS;
```

Members

Members	Description
uint32_t nTouch : 4;	Number of available touch packets
uint32_t streamReady : 1;	stream data ready
uint32_t gestureReady : 1;	gesture data ready
uint32_t gestICData : 1;	GestIC data ready
uint32_t reserved : 1;	reserved bit

Description

MTCH6303 I2C touch status

This structure defines the I2C touch status register bits.

Remarks

It is part of DRV_MTCH6303_TOUCH_DATA structure. It is packed to form structure of size 1 byte.

DRV_TOUCH_MTCH6303_MSG_ID Enumeration

List of report or command message identification.

File

drv_mtch6303.h

C

```
typedef enum {
  DRV_TOUCH_MTCH6303_MSG_CMD_QUERY_VERSION
} DRV_TOUCH_MTCH6303_MSG_ID;
```

Members

Members	Description
DRV_TOUCH_MTCH6303_MSG_CMD_QUERY_VERSION	Message sends firmware version query command. Bytes 124:127 = Rev[2].Minor.Major

Description

MTCH6303 Touch message Identification.

This enumeration identifies the different report or command messages supported by MTCH6303. This identifier identifies the type of the message. The identifier is passed in the message DRV_MTCH6303_TOUCH_MESSAGE as first byte of the payload. It is applicable only for first fragment of message. If message involves multiple fragments, the payload of message fragments other than first fragment should start with normal payload byte. The touch message is read or send to MTCH6303 by using DRV_MTCH6303_TOUCH_AddMessageReportRead or DRV_MTCH6303_TOUCH_AddMessageCommandWrite.

Remarks

To be passed as first byte of message payload. Applicable only for first fragment of message.

DRV_TOUCH_MTCH6303_I2C_REGISTER_MAP Enumeration

List of MTCH6303 I2C Accessible Register Identification.

File

drv_mtch6303.h

C

```
typedef enum {
} DRV_TOUCH_MTCH6303_I2C_REGISTER_MAP;
```

Description

MTCH6303 I2C Accessible Register Identification.

This enumeration identifies the different I2C accessible MTCH6303 Registers. The identifier is passed as source to the register read routine or as destination to the register write routine. The MTCH6303 driver routine to read the I2C accessible MTCH6303 registers is DRV_MTCH6303_AddRegisterRead. The MTCH6303 driver routine to write the I2C accessible MTCH6303 registers is DRV_MTCH6303_AddRegisterWrite.

Remarks

To read or write multiple registers, identifier of only first register is sufficient as source or destination respectively.

Files

Files

Name	Description
drv_mtch6303.h	MTCH6303 driver interface declarations for the static single instance driver.

Description

This section lists the source and header files used by the MTCH6303 Touch Driver Library.

drv_mtch6303.h

MTCH6303 driver interface declarations for the static single instance driver.

Enumerations

	Name	Description
	DRV_MTCH6303_BUFFER_EVENT	Lists the different conditions that happens during a buffer transfer.
	DRV_MTCH6303_CLIENT_STATUS	Defines the client-specific status of the MTCH6303 driver.
	DRV_MTCH6303_ERROR	Defines the possible errors that can occur during driver operation.
	DRV_MTCH6303_TOUCH_BUFFER_EVENT	Lists the different conditions that happens during a touch message buffer transfer.
	DRV_TOUCH_MTCH6303_I2C_REGISTER_MAP	List of MTCH6303 I2C Accessible Register Identification.
	DRV_TOUCH_MTCH6303_MSG_ID	List of report or command message identification.

Functions

	Name	Description
	DRV_MTCH6303_AddRegisterRead	Schedules a non-blocking register read request to read I2C accessible MTCH6303 registers.
	DRV_MTCH6303_AddRegisterWrite	Schedule a non-blocking driver register write operation to write I2C accessible MTCH6303 registers.
	DRV_MTCH6303_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.
	DRV_MTCH6303_Close	Closes an opened-instance of the MTCH6303 driver.
	DRV_MTCH6303_Deinitialize	Deinitializes the instance of the MTCH6303 driver module.
	DRV_MTCH6303_ErrorGet	This function returns the error associated with the last client request.
	DRV_MTCH6303_Initialize	Initializes the MTCH6303 static single instance.
	DRV_MTCH6303_Open	Opens the MTCH6303 driver instance and returns a handle to it.
	DRV_MTCH6303_Status	Gets the current status of the MTCH6303 driver module.

	DRV_MTCH6303_Tasks	Maintains the driver's register read/write state machine and implements its ISR.
	DRV_MTCH6303_TOUCH_AddMessageCommandWrite	Schedule a non-blocking driver command message write operation to write command message to MTCH6303 registers.
	DRV_MTCH6303_TOUCH_AddMessageReportRead	Schedules a non-blocking report message read request to read the report message from MTCH6303 device.
	DRV_MTCH6303_TOUCH_AddTouchInputRead	Schedules a non-blocking read buffer request to read touch input from MTCH6303.
	DRV_MTCH6303_TOUCH_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued message transfers have finished.
	DRV_MTCH6303_TOUCH_Tasks	Maintains the driver's message state machine and implements its ISR.
	DRV_MTCH6303_TouchInputMap	Maps the raw touch input to display resolution.
	DRV_MTCH6303_TouchInputRead	Schedules a non-blocking read buffer request to read touch input from MTCH6303.

Macros

	Name	Description
	DRV_MTCH6303_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_MTCH6303_TOUCH_NUM_INPUTS	Definition of number of touch input packets can be identified by MTCH6303.

Structures

	Name	Description
	DRV_MTCH6303_TOUCH_DATA	Defines MTCH6303 I2C Touch Data
	DRV_MTCH6303_TOUCH_INPUT	Defines MTCH6303 Touch Input Packet
	DRV_MTCH6303_TOUCH_MESSAGE	Defines MTCH6303 Touch Message.
	DRV_MTCH6303_TOUCH_MESSAGE_HEADER	Defines Touch Message Header.
	DRV_MTCH6303_TOUCH_NIBBLE_0	Defines the I2C Nibble 0 of MTCH6303 Touch input packet.
	DRV_MTCH6303_TOUCH_STATUS	Defines the I2C touch status register bits

Types

	Name	Description
	DRV_MTCH6303_BUFFER_EVENT_HANDLER	Points to a callback after completion of an register read -write or message stream read - write.
	DRV_MTCH6303_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.
	DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER	Points to a callback after completion of an message report read or message command write.
	DRV_MTCH6303_TOUCH_BUFFER_HANDLE	Handle identifying a read or write touch message buffer passed to the driver.

Description

MTCH6303 Driver Interface Declarations for Static Single Instance Driver

The MTCH6303 device driver provides a simple interface to manage the MTCH6303 module. This file defines the interface Declarations for the MTCH6303 driver.

Remarks

Static single instance driver interface eliminates the need for an object ID or object handle. Static single-open interfaces also eliminate the need for the open handle.

File Name

drv_mtch6303_static.h

Company

Microchip Technology Inc.

USB Driver Libraries

Common Interface

Provides information on the USB Driver interface that is common to all PIC32 devices.

Description

The USB Driver Common Interface definition specifies the functions and their behavior that a USB Driver must implement so that the driver can be used by the MPLAB Harmony USB Host and Device Stack.

Note: The MPLAB Harmony USB Driver for PIC32MX and PIC32MZ devices implements the USB Driver Common Interface.

The USB Driver Common Interface contains functions that are grouped as follows:

- *Driver System Functions* - These functions are called by MPLAB Harmony to initialize and maintain the operational state of the USB Driver. The system functions can vary between different PIC32 device USB Drivers. As such, the USB Driver Common Interface does not require these functions to be of the same type. These functions are not called by the USB Host or Device Stack and therefore are allowed to (and can) vary across different PIC32 device USB Drivers. A description of these functions, along with a description of how to initialize the USB Driver for Host, Device or Dual Role operation, is provided in the specific PIC32 device USB Driver help section (see PIC32MX USB Driver and PIC32MZ USB Driver).
- *Driver General Client Functions* -These functions are called by the USB Host or Device Stack to gain access to the driver
- *Driver Host Mode Client Functions* - These functions are called exclusively by the USB Host Stack to operate and access the USB as a Host
- *Driver Device Mode Client Functions* - These functions are called exclusively by the USB Device Stack to operate and access the USB as a Device

The USB Driver Common Interface is defined in the `<install-dir>\framework\driver\usb\drv_usb.h` file. This file contains the data types and structures that define the interface. Specifically, the DRV_USB_HOST_INTERFACE structure, contained in this file, is the common interface for USB Driver Host mode functions. It is a structure of function pointers, pointing to functions that define the Driver Host mode Client functions. The following code example shows this structure and the function pointer it contains.

```
// *****************************************************************************
/* USB Driver Client Functions Interface (For Host mode)

  Summary:
    Group of function pointers to the USB Driver Host mode Client Functions.

  Description:
    This structure is a group of function pointers pointing to the USB Driver
    Host mode Client routines. The USB Driver should export this group of
    functions so that the Host layer can access the driver functionality.

  Remarks:
    None.
*/

typedef struct
{
    /* This is a pointer to the driver Open function. This function may be
     * called twice in a Dual Role application, once by the Host Stack and then
     * by the Device Stack */
    DRV_HANDLE (*open)(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);

    /* This is pointer to the driver Close function */
    void (*close)(DRV_HANDLE handle);

    /* This is a pointer to the event call back set function */
    void (*eventHandlerSet)(DRV_HANDLE handle, uintptr_t hReferenceData,
           DRV_USB_EVENT_CALLBACK eventHandler);

    /* This is a pointer to the Host IRP submit function */
    USB_ERROR (*hostIRPSubmit)(DRV_USB_HOST_PIPE_HANDLE pipeHandle, USB_HOST_IRP * irp);

    /* This is a pointer to the Host IRP Cancel all function */
    void (*hostIRPCancel)(USB_HOST_IRP * irp);

    /* This is pointer to the Host event disable function */
    bool (*hostEventsDisable)(DRV_HANDLE handle);

    /* This is a pointer to the Host event enable function */
    void (*hostEventsEnable)(DRV_HANDLE handle, bool eventContext);
```

```
    /* This is a pointer to the Host pipe setup function */
    DRV_USB_HOST_PIPE_HANDLE (*hostPipeSetup)
    (
        DRV_HANDLE client,
        uint8_t deviceAddress,
        USB_ENDPOINT endpointAndDirection,
        uint8_t hubAddress,
        uint8_t hubPort,
        USB_TRANSFER_TYPE pipeType,
        uint8_t bInterval,
        uint16_t wMaxPacketSize,
        USB_SPEED speed
    );

    /* This is a pointer to the Host Pipe Close function */
    void (*hostPipeClose)(DRV_USB_HOST_PIPE_HANDLE pipeHandle);

    /* This is a pointer to the Host Root Hub functions */
    DRV_USB_ROOT_HUB_INTERFACE rootHubInterface;

} DRV_USB_HOST_INTERFACE;
```

The DRV_USB_DEVICE_INTERFACE structure, contained in this file, is the common interface for USB Driver Device mode functions. It is a structure of function pointers, pointer to functions that define the Driver Device mode Client functions. The following code example shows this structure and the function pointer it contains.

```
// *****************************************************************************
/* USB Driver Client Functions Interface (For Device Mode)

  Summary:
    Group of function pointers to the USB Driver Device Mode Client Functions.

  Description:
    This structure is a group of function pointers pointing to the USB Driver
    Device Mode Client routines. The USB Driver should export this group of
    functions so that the Device Layer can access the driver functionality.

  Remarks:
    None.
*/

typedef struct
{
    /* This is a pointer to the driver Open function */
    DRV_HANDLE (*open)(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);

    /* This is pointer to the driver Close function */
    void (*close)(DRV_HANDLE handle);

    /* This is a pointer to the event call back set function */
    void (*eventHandlerSet)(DRV_HANDLE handle, uintptr_t hReferenceData,
           DRV_USB_EVENT_CALLBACK eventHandler);

    /* This is a pointer to the device address set function */
    void (*deviceAddressSet)(DRV_HANDLE handle, uint8_t address);

    /* This is a pointer to the device current speed get function */
    USB_SPEED (*deviceCurrentSpeedGet)(DRV_HANDLE handle);

    /* This is a pointer to the SOF Number get function */
    uint16_t (*deviceSOFNumberGet)(DRV_HANDLE handle);

    /* This is a pointer to the device attach function */
    void (*deviceAttach)(DRV_HANDLE handle);

    /* This is a pointer to the device detach function */
    void (*deviceDetach)(DRV_HANDLE handle);

    /* This is a pointer to the device endpoint enable function */
    USB_ERROR (*deviceEndpointEnable)(DRV_HANDLE handle, USB_ENDPOINT endpoint,
                USB_TRANSFER_TYPE transferType, uint16_t endpointSize);
```

```
    /* This is a pointer to the device endpoint disable function */
    USB_ERROR (*deviceEndpointDisable)(DRV_HANDLE handle, USB_ENDPOINT endpoint);

    /* This is a pointer to the device endpoint stall function */
    USB_ERROR (*deviceEndpointStall)(DRV_HANDLE handle, USB_ENDPOINT endpoint);

    /* This is a pointer to the device endpoint stall clear function */
    USB_ERROR (*deviceEndpointStallClear)(DRV_HANDLE handle, USB_ENDPOINT endpoint);

    /* This is pointer to the device endpoint enable status query function */
    bool (*deviceEndpointIsEnabled)(DRV_HANDLE handle, USB_ENDPOINT endpoint);

    /* This is pointer to the device endpoint stall status query function */
    bool (*deviceEndpointIsStalled)(DRV_HANDLE handle, USB_ENDPOINT endpoint);

    /* This is a pointer to the device IRP submit function */
    USB_ERROR (*deviceIRPSubmit)(DRV_HANDLE handle, USB_ENDPOINT endpoint,
                USB_DEVICE_IRP * irp);

    /* This is a pointer to the device IRP Cancel all function */
    USB_ERROR (*deviceIRPCancelAll)(DRV_HANDLE handle, USB_ENDPOINT endpoint);

    /* This is a pointer to the device remote wakeup start function */
    void (*deviceRemoteWakeupStart)(DRV_HANDLE handle);

    /* This is a pointer to the device remote wakeup stop function */
    void (*deviceRemoteWakeupStop)(DRV_HANDLE handle);

    /* This is a pointer to the device Test mode enter function */
    USB_ERROR (*deviceTestModeEnter)(DRV_HANDLE handle, USB_TEST_MODE_SELECTORS testMode);

} DRV_USB_DEVICE_INTERFACE;
```

Both of these structures also contain pointers to General Client functions. The specific PIC32 device USB Driver allocates and initializes such a structure. The following code example shows how the PIC32MX USB Host mode Driver allocates and initializes the DRV_USB_HOST_INTERFACE structure. This code is contained in the <install-dir>\framework\driver\usb\usbhs\src\dynamic\drv_usbfs_host.c file.

```
/*******************************************************
 * This structure is a set of pointer to the USBFS driver
 * functions. It is provided to the Host layer as the
 * interface to the driver.
 * *****************************************************/

DRV_USB_HOST_INTERFACE gDrvUSBFSHostInterface =
{
    .open = DRV_USBFS_Open,
    .close = DRV_USBFS_Close,
    .eventHandlerSet = DRV_USBFS_ClientEventCallBackSet,
    .hostIRPSubmit = DRV_USBFS_HOST_IRPSubmit,
    .hostIRPCancel = DRV_USBFS_HOST_IRPCancel,
    .hostPipeSetup = DRV_USBFS_HOST_PipeSetup,
    .hostPipeClose = DRV_USBFS_HOST_PipeClose,
    .hostEventsDisable = DRV_USBFS_HOST_EventsDisable,
    .hostEventsEnable = DRV_USBFS_HOST_EventsEnable,
    .rootHubInterface.rootHubPortInterface.hubPortReset = DRV_USBFS_HOST_ROOT_HUB_PortReset,
    .rootHubInterface.rootHubPortInterface.hubPortSpeedGet =
                                                DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet,
    .rootHubInterface.rootHubPortInterface.hubPortResetIsComplete =
                                                DRV_USBFS_HOST_ROOT_HUB_PortResetIsComplete,
    .rootHubInterface.rootHubPortInterface.hubPortSuspend = DRV_USBFS_HOST_ROOT_HUB_PortSuspend,
    .rootHubInterface.rootHubPortInterface.hubPortResume = DRV_USBFS_HOST_ROOT_HUB_PortResume,
    .rootHubInterface.rootHubMaxCurrentGet = DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet,
    .rootHubInterface.rootHubPortNumbersGet = DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet,
    .rootHubInterface.rootHubSpeedGet = DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet,
    .rootHubInterface.rootHubInitialize = DRV_USBFS_HOST_ROOT_HUB_Initialize,
    .rootHubInterface.rootHubOperationEnable = DRV_USBFS_HOST_ROOT_HUB_OperationEnable,
    .rootHubInterface.rootHubOperationIsEnabled = DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled,
};
```

Similarly, the PIC32MX USB Device mode Driver allocates and initializes the DRV_USB_DEVICE_INTERFACE structure. This can be reviewed in the <install-dir>\framework\driver\usb\usbhs\src\dynamic\drv_usbfs_device.c file.

```
/*************************************************
 * This structure is a pointer to a set of USB Driver
 * Device mode functions. This set is exported to the
 * Device Layer when the Device Layer must use the
 * PIC32MX USB Controller.
 *************************************************/

DRV_USB_DEVICE_INTERFACE gDrvUSBFSDeviceInterface =
{
    .open = DRV_USBFS_Open,
    .close = DRV_USBFS_Close,
    .eventHandlerSet = DRV_USBFS_ClientEventCallBackSet,
    .deviceAddressSet = DRV_USBFS_DEVICE_AddressSet,
    .deviceCurrentSpeedGet = DRV_USBFS_DEVICE_CurrentSpeedGet,
    .deviceSOFNumberGet = DRV_USBFS_DEVICE_SOFNumberGet,
    .deviceAttach = DRV_USBFS_DEVICE_Attach,
    .deviceDetach = DRV_USBFS_DEVICE_Detach,
    .deviceEndpointEnable = DRV_USBFS_DEVICE_EndpointEnable,
    .deviceEndpointDisable = DRV_USBFS_DEVICE_EndpointDisable,
    .deviceEndpointStall = DRV_USBFS_DEVICE_EndpointStall,
    .deviceEndpointStallClear = DRV_USBFS_DEVICE_EndpointStallClear,
    .deviceEndpointIsEnabled = DRV_USBFS_DEVICE_EndpointIsEnabled,
    .deviceEndpointIsStalled = DRV_USBFS_DEVICE_EndpointIsStalled,
    .deviceIRPSubmit = DRV_USBFS_DEVICE_IRPSubmit,
    .deviceIRPCancelAll = DRV_USBFS_DEVICE_IRPCancelAll,
    .deviceRemoteWakeupStop = DRV_USBFS_DEVICE_RemoteWakeupStop,
    .deviceRemoteWakeupStart = DRV_USBFS_DEVICE_RemoteWakeupStart,
    .deviceTestModeEnter = NULL


};
```

A pointer to the DRV_USB_HOST_INTERFACE structure is passed to the USB Host Stack as part of USB Host Stack initialization. The following code example shows how this is done.

```
/*************************************************************************
 * This is a table of the USB Host mode drivers that this application will
 * support. Also contained in the driver index. In this example, the
 * application will want to use instance 0 of the PIC32MX USB Full-Speed driver.
 * ***********************************************************************/
const USB_HOST_HCD hcdTable =
{
    .drvIndex = DRV_USBFS_INDEX_0,
    .hcdInterface = DRV_USBFS_HOST_INTERFACE
};

/* Here the pointer to the USB Driver Common Interface is provided to the USB
 * Host Layer via the hostControllerDrivers member of the Host Layer
 * Initialization data structure. */
const USB_HOST_INIT usbHostInitData =
{
    .nTPLEntries = 1 ,
    .tplList = (USB_HOST_TPL_ENTRY *)USBTPList,
    .hostControllerDrivers = (USB_HOST_HCD *)&hcdTable

};
```

A pointer to the DRV_USB_DEVICE_INTERFACE structure is passed to the USB Device Stack as part of the USB Device Stack initialization. The Host Stack and Device Stack then access the driver functions through the function pointers contained in these structures.

The Driver General Client, Host mode and Device mode Client functions are described in this section. Any references to a USB Driver Client in the following sections, implies the client is a USB Host Stack and/or the USB Device Stack.

Driver General Client Functions

Provides information on the General Client functions for the USB Driver.

Description

The DRV_USB_HOST_INTERFACE and the DRV_USB_DEVICE_INTERFACE structures contain pointers to the USB Driver's General Client

functions. These functions are not specific to the operation mode (Host, Device, or Dual Role) of the driver. A USB Driver must implement these functions and ensure that the Host or Device Stack can access these functions through the driver's common interface structures. The common interface contains three general client functions:

- Driver Open Function
- Driver Close Function
- Driver Event Handler Set Function

Driver Open Function

The `open` member of the DRV_USB_HOST_INTERFACE and the DRV_USB_DEVICE_INTERFACE structures should point to the USB Driver Open function. The signature of the Open function is as follows:

```
DRV_HANDLE (*open)(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

The USB Driver Open function must match this signature. The Driver Client uses the USB Driver index (drvIndex) to specify the instance of the USB module that Host Stack or the Device Stack should open. The USB Driver should ignore the `intent` parameter. The function should return a driver handle. If the driver is not ready to be opened, it should return an invalid handle (DRV_HANDLE_INVALID). In such a case, the client will continue trying to open the driver by calling the Open function again. The driver may also fail to open for an invalid `index` parameter or if USB module is in an error condition.

When supporting Dual Role operation, both the Host Stack and Device Stack will call the Driver Open function in one application. The USB Driver must support multiple calls to the Open function in the same application. The Open function should be thread-safe.

Driver Close Function

The `close` member of the DRV_USB_HOST_INTERFACE and the DRV_USB_DEVICE_INTERFACE structures should point to the USB Driver Close function. The signature of the Close function is as follows:

```
void (*close)(DRV_HANDLE handle);
```

The USB Driver Close function must match this signature. The Driver Client passes the handle obtained from the Driver Open function as a parameter to the close. The USB Host Stack or USB Device Stack will close the driver only when the stack is deinitialized (which is typically a rare case). The USB Driver should deallocate any client-related resources in the Close function. If the specified driver handle is not valid, the Close function should not have any side effects. The USB Driver expects the Close function to be called from the context of the thread in which the driver was opened; therefore, this function *is not* expected to be thread-safe.

Driver Event Handler Set Function

The `eventHandlerSet` member of the DRV_USB_HOST_INTERFACE and the DRV_USB_DEVICE_INTERFACE structures should point to the USB Driver Event Handler Set function. The signature of the Event Handler Set function is as follows:

```
void (*eventHandlerSet)(DRV_HANDLE handle, uintptr_t hReferenceData, DRV_USB_EVENT_CALLBACK eventHandler);
```

The USB Driver Event Handler Set function must match this signature. The signature of the Client Event Handling function should match DRV_USB_EVENT_CALLBACK. The USB Driver calls this function when it must communicate USB events to the client. The client can set the `eventHandler` parameter to NULL if it does not want to receive USB Driver events. The client will receive Host mode events if the USB Driver is operating in Host mode. It will receive Device mode events if the USB Driver is operating in Device mode. The DRV_USB_EVENT type enumeration contains all the possible events that the USB Driver would generate. The following code example shows the enumeration.

```
// *****************************************************************************
/* USB Driver Events Enumeration

  Summary:
    Identifies the different events that the USB Driver provides.

  Description:
    Identifies the different events that the USB Driver provides. The USB Driver
    should be able to provide these events.

  Remarks:
    None.
*/

typedef enum
{
    /* Bus error occurred and was reported. This event can be generated in both
     * Host and Device mode. */
    DRV_USB_EVENT_ERROR = 1,

    /* Host has issued a device Reset. This event occurs only in Device mode */
    DRV_USB_EVENT_RESET_DETECT,

    /* Resume detected while USB in suspend mode. This event can be generated in
     * both Host and Device mode. In Host mode, the events occurs when a remote
     * wakeup capable device has generated resume signaling. In Device mode,
     * this event will occur when the Host has issued resume signaling. */
```

```
    DRV_USB_EVENT_RESUME_DETECT,

    /* This event is generated in Device mode only. It occurs when the Host
     * suspends the bus and the bus goes idle. */
    DRV_USB_EVENT_IDLE_DETECT,

    /* This event is generated in Host mode and Device mode. In Host mode, this
     * event occurs when the device has stalled the Host. In Device mode, this
     * event occurs when the Host has accessed a stalled endpoint thus
     * triggering the device to send a STALL to the Host. */
    DRV_USB_EVENT_STALL,

    /* This event is generated in Host mode and Device mode. In Device mode,
     * this event occurs when a SOF has been generated by the Host. In Host
     * mode, this event occurs when controller is about to generate an SOF.
     * */
    DRV_USB_EVENT_SOF_DETECT,

    /* This event is generated in Device mode when the VBUS voltage is above
     * VBUS session valid. */
    DRV_USB_EVENT_DEVICE_SESSION_VALID,

    /* This event is generated in Device mode when the VBUS voltage falls
     * below VBUS session valid. */
    DRV_USB_EVENT_DEVICE_SESSION_INVALID,

} DRV_USB_EVENT;
```

This completes the discussion on the Driver General Client Functions.

Driver Host Mode Client Functions

Provides information on the Host mode Client functions for the USB Driver.

Description

The DRV_USB_HOST_INTERFACE structure contains pointers to the USB Driver's Host mode Client functions. These functions are only applicable when the USB module is operating as a USB Host. Along with the function pointers to the driver's Host mode specific functions, the DRV_USB_HOST_INTERFACE structure also contains another structure of function pointers of the type DRV_USB_ROOT_HUB_INTERFACE. This structure contains function pointers to the USB Driver's Root Hub functions. A USB Driver must implement these functions and ensure that the Host Stack can access these functions through the driver's DRV_USB_HOST_INTERFACE structure. The Driver Host mode Client functions in the DRV_USB_HOST_INTERFACE structure are:

- Driver Host Pipe Setup Function
- Driver Host Pipe Close Function
- Driver Host Events Disable Function
- Driver Host Events Enable Function
- Driver Host IRP Submit Function
- Driver Host IRP Cancel Function

Driver Host Pipe Setup Function

The `hostPipeSetup` member of the DRV_USB_HOST_INTERFACE structure should point to the USB Driver Host Pipe Setup function. The signature of the Host Pipe Setup function is as follows:

```
DRV_USB_HOST_PIPE_HANDLE (*hostPipeSetup)  ( DRV_HANDLE client,  uint8_t deviceAddress,
                           USB_ENDPOINT endpointAndDirection, uint8_t hubAddress,
                           uint8_t hubPort, USB_TRANSFER_TYPE pipeType, uint8_t bInterval,
                           uint16_t wMaxPacketSize, USB_SPEED speed);
```

The USB Driver Host mode Pipe Setup function must match this signature. The USB Host Stack calls this function to create a communication pipe to the attached device. The function parameters define the property of this communication pipe. The `driverHandle` parameter is the handle to the driver obtained through the driver Open function. The `deviceAddress` and the `endpointAddress` parameters specify the address of the USB device and the endpoint on this device to which this pipe must connect.

If the device is connected to the Host though a hub, `hubAddress` and `hubPort` must specify the address of the hub and port to which the device is connected. The USB Driver will use these parameters to schedule split transactions if the target device is a Low-Speed or Full-Speed device and is connected to the Host through a high-speed hub. If the device is connected directly to the Host, these parameters should be set to zero ('`0`').

The `pipeType` parameter specifies the type of USB transfers that this pipe would support. The `bInterval` parameter is interpreted as per the USB 2.0 Specification based on the transfer type and the speed of the pipe. The `wMaxPacketSize` parameter defines the maximum size of a transaction that the driver should use while transporting a transfer on the pipe. The Host layer will use the information obtained from the USB device descriptors of the attached device to decide the `wMaxPacketSize` parameter.

The Driver Host Pipe Setup function should be thread-safe, but does not have to be event safe. The Host layer (or the Host Client Drivers) will not, and should not attempt to create a pipe in an interrupt, and therefore, an event context. The function should return DRV_USB_PIPE_HANDLE_INVALID if the driver could not open the pipe. The driver may not be able to open a pipe due to incorrect function parameters or due to lack of resources.

Driver Host Pipe Close Function

The `hostPipeClose` member of the DRV_USB_HOST_INTERFACE structure should point to the USB Driver Host Pipe Close function. The signature of the Host Pipe Close function is as follows:

```
void (*hostPipeClose)(DRV_USB_HOST_PIPE_HANDLE pipeHandle);
```

The USB Driver Host mode Pipe Close function must match this signature. The USB Host Stack calls this function to close communication pipes. The `pipeHandle` parameter is the pipe handle obtained from the Pipe Setup function. The Host Client Driver typically closes pipes when a device detach was detected. The Client Driver may also close pipes when a device configuration needs to change or when the Client Driver is being unloaded by the Host. The Pipe Close function has no side effect if the pipe handle is invalid. Closing the pipe will abort all I/O Request Packets (IRP) that are scheduled on the pipe. Any transaction in progress will complete. The IRP callback functions for each IRP scheduled in the pipe will be called with a USB_HOST_IRP_STATUS_ABORTED status.

The USB Driver Pipe Close function must be thread-safe and event-safe. The latter requirement allows the Pipe Close function to be called in the context of the device detach Interrupt Service Routine.

Driver Host Event Disable Function

The `hostEventsDisable` member of the DRV_USB_HOST_INTERFACE structure should point to the USB Driver Host mode Driver Events Disable function. The signature of the Events Disable function is as follows:

```
bool (*hostEventsDisable)(DRV_HANDLE handle);
```

The USB Driver Host mode Driver Events Disable function must match this signature. The Host Stack will call this function when it wants to execute a section of code that should not be interrupted by the USB Driver. Calling this function should disable USB Driver event generation. The `handle` parameter is set to the driver handle obtained via the driver Open function. The function will return the present state of the event generation, whether it is enabled or disabled. The Host Stack will pass this value to the USB Driver Host mode Driver Events Enable function when it needs to enable the driver events.

Driver Host Events Enable Function

The `hostEventsEnable` member of the DRV_USB_HOST_INTERFACE structure should point to the USB Driver Host mode Driver Events Enable function. The signature of the events enable function is as follows:

```
void (*hostEventsEnable)(DRV_HANDLE handle, bool eventContext);
```

The USB Driver Host mode Driver Events Enable function must match this signature. The USB Host Stack calls this function to re-enable the USB Driver Host mode Events (if they were enabled) after it called the USB Driver Host mode Events Disable function to disable driver events. The `handle` parameter is set to the driver handle obtained via the driver Open function. The `eventContext` parameter is set to the value returned by the Host mode Driver Events Disable function. The USB Driver will use the `eventContext` parameter to restore the event generation status (enabled or disabled) to what it was when the USB Driver Host mode Driver Events Disable function was called.

Driver Host IRP Submit Function

The `hostIRPSubmit` member of the DRV_USB_HOST_INTERFACE structure should point to the USB Driver Host IRP Submit function. The signature of the IRP Submit function is as follows:

```
USB_ERROR (*hostIRPSubmit)(DRV_USB_HOST_PIPE_HANDLE pipeHandle, USB_HOST_IRP * irp);
```

The USB Driver Host IRP Submit function must match this signature. The Host Stack calls this function to submit an IRP to the USB Driver. The USB Driver provides this mechanism to transfer data between the Host Stack and the attached device. The `pipeHandle` parameter should be set to the pipe handle obtained by the Pipe Setup function. The pipe handle specifies the pipe, and therefore, the target device, endpoint, speed and transfer type, on which the I/O must be processed. The `irp` parameter should point to the IRP data structure. The IRP data structure will transport an entire transfer over the pipe. The USB Driver will split up the transfer into transactions based on the parameters specified at the time of pipe creation. This process does not require Host Stack intervention.

The function will return USB_ERROR_HOST_PIPE_INVALID if the pipe handle is not valid. It will return USB_ERROR_OSAL_FUNCTION if an error occurred while performing a RTOS-related operation. It will return USB_ERROR_NONE if the IRP was submitted successfully.

The USB Driver will queue the IRP if there is already an IRP being processed on the pipe. The completion of the IRP processing is indicated by the USB Driver calling the IRP Callback function specified within the IRP. The Host IRP Submit function must be thread-safe and IRP callback-safe. The Host Stack may resubmit the IRP within the IRP Callback function. The IRP Callback function itself executes within an interrupt context. The completion status of the IRP will be available in the `status` member of the IRP when the IRP callback function is invoked.

Driver Host IRP Cancel Function

The `hostIRPCancel` member of the DRV_USB_HOST_INTERFACE structure should point to the USB Driver Host IRP Cancel function. The signature of the IRP Cancel function is as follows

```
void (*hostIRPCancel)(USB_HOST_IRP * irp);
```

The USB Driver Host IRP Cancel function must match this signature. The Host Stack and Host Client Drivers will call this function to cancel an IRP that was submitted. The IRP will be aborted successfully if it is not in progress. If the IRP processing has begun, the on-going transaction will complete and pending transactions in the transfer will be aborted. In either case, the IRP Callback function will be called with the IRP status as USB_HOST_IRP_STATUS_ABORTED.

Driver Host USB Root Hub Port Interface

Provides information on the Root Hub Port interface of the USB Host Driver.

Description

The `rootHubPortInterface` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Port functions. The data type of this member is USB_HUB_INTERFACE. This data type is a structure containing function pointers pointing to the port control functions of the root hub. The USB Driver must assign the function pointers in this structure to the root hub port control functions. These same functions are also exported by a Hub Driver to the USB Host Stack, which allow the Host Stack to control a device regardless of whether it is connected to a root hub or an external hub. The port functions are valid only when a device is attached to the port. The behavior of these functions on a port to which no device is connected is not defined. Descriptions of the port control functions are provided, which include:

- Driver Host Hub Port Reset Function
- Driver Host Hub Port Reset Completion Status Function
- Driver Host Hub Port Suspend Function
- Driver Host Hub Port Resume Function
- Driver Host Hub Port Speed Get Function

Driver Host Hub Port Reset Function

The `hubPortReset` member of the USB_HUB_INTERFACE structure should point to the USB Driver Root Hub Port Reset function. The signature of this function is as follows:

```
USB_ERROR (*hubPortReset)(uintptr_t hubAddress, uint8_t port);
```

The USB Driver Root Hub Port Reset function must follow this signature. This function starts reset signaling on the port. If the device is connected to the root hub, the USB Host Stack will set the `hubAddress` parameter to the driver handle obtained through the driver Open function. The USB Host Stack uses the parent identifier provided by the root hub driver when the USB_HOST_DeviceEnumerate function was called to query the driver handle that is linked to this root hub. If the device is connected to an external hub, the `hubAddress` parameter is directly set to the parent identifier.

For the PIC32MX and PIC32MZ USB Drivers, the `port` parameter is ignored. For an external hub, this must be set to the port to which the device is connected. The function returns USB_ERROR_NONE if the function was successful. If the reset signaling is already in progress on the port, calling this function has no effect. The USB Driver will itself time duration of the reset signal. This does not require USB Host Stack intervention. The USB Host Stack will call the port reset completion status function to check if the reset signaling has completed. Calling this function on a port which exists on an external hub will cause the hub driver to issue a control transfer to start the port reset procedure.

Driver Host Hub Port Reset Completion Status Function

The `hubPortResetIsComplete` member of the USB_HUB_INTERFACE structure should point to the USB Driver Root Hub Port Reset Completion Status function. The signature of this function is as follows:

```
bool (*hubPortResetIsComplete)(uintptr_t hubAddress, uint8_t port);
```

The USB Driver Root Hub Port Reset Completion Status function must follow this signature. The USB Host Stack calls this function to check if the port reset sequence that was started on a port has completed. The function returns true if the reset signaling has completed. If the device is connected to the root hub, the USB Host Stack will set the `hubAddress` parameter to the driver handle obtained through the driver Open function. If the device is connected to an external hub, the `hubAddress` parameter is directly set to the parent identifier.

For the PIC32MX and PIC32MZ USB Drivers, the `port` parameter is ignored. For an external hub, this parameter must be set to the port to which the device is connected.

Driver Host Hub Port Suspend Function

The `hubPortSuspend` member of the USB_HUB_INTERFACE structure should point to the USB Driver Root Hub Port Suspend function. The signature of this function is as follows:

```
USB_ERROR(*hubPortSuspend)(uintptr_t hubAddress, uint8_t port);
```

The USB Driver Root Hub Port Suspend function must follow this signature. The USB Host Stack calls this function to suspend the port. If the device is connected to the root hub, the USB Host Stack will set the `hubAddress` parameter to the driver handle obtained through the driver Open function. If the device is connected to an external hub, the `hubAddress` parameter is directly set to the parent identifier.

For the PIC32MX and PIC32MZ USB Drivers, the `port` parameter is ignored. For an external hub, this parameter must be set to the port to which the device is connected. The function returns USB_ERROR_NONE if the request was successful. Calling this function on a suspended port will not have any effect.

Driver Host Hub Port Resume Function

The `hubPortResume` member of the USB_HUB_INTERFACE structure should point to the USB Driver Root Hub Port Resume function. The signature of this function is as follows:

```
USB_ERROR(*hubPortResume)(uintptr_t hubAddress, uint8_t port);
```

The USB Driver Root Hub Port Resume function must follow this signature. The USB Host Stack calls this function to resume a suspended port. If the device is connected to the root hub, the USB Host Stack will set the `hubAddress` parameter to the driver handle obtained through the driver Open function. If the device is connected to an external hub, the `hubAddress` parameter is directly set to the parent identifier.

For the PIC32MX and PIC32MZ USB Drivers, the `port` parameter is ignored. For an external hub, this parameter must be set to the port to which the device is connected. The function returns USB_ERROR_NONE if the request was successful. Calling this function on a port that is not suspended will not have any effect.

Driver Host Hub Port Speed Get Function

The `hubPortSpeedGet` member of the USB_HUB_INTERFACE structure should point to the USB Driver Root Hub Port Speed Get function. The signature of this function is as follows:

```
USB_SPEED(*hubPortSpeedGet)(uintptr_t hubAddress, uint8_t port);
```

The USB Driver Root Hub Port Speed Get function must follow this signature. The USB Host Stack calls this function to obtain the USB speed of the device that is attached to the port. The Host Stack calls this function only after it has completed reset of the port. If the device is connected to the root hub, the USB Host Stack will set the `hubAddress` parameter to the driver handle obtained through the driver Open function. If the device is connected to an external hub, the `hubAddress` parameter is directly set to the parent identifier.

For the PIC32MX and PIC32MZ USB Drivers, the `port` parameter is ignored. For an external hub, this parameter must be set to the port to which the device is connected. The function returns USB_SPEED_ERROR if the request was not successful. It will return the functional USB speed otherwise.

This concludes the section describing the USB Driver Host mode Client Functions. The USB Driver Device Mode Client Functions are discussed in the next section.

Driver Host Root Hub Interface

Provides information on the Root Hub interface for the USB Host Driver.

Description

The USB Driver Common Interface requires the USB Driver to be operating in Host mode to provide root hub control functions. If the USB peripheral does not contain root hub features in hardware, these features must be emulated in software by the driver. The USB peripheral on PIC32MX and PIC32MZ devices does not contain root hub features; therefore, the USB Driver for these peripherals emulates the root hub functionality in software. The `rootHubInterface` member of the DRV_USB_HOST_INTERFACE structure is a structure of type DRV_USB_ROOT_HUB_INTERFACE. The members of this structure are function pointers to the root hub control functions of the USB Driver.

Along with other Host mode functions, the USB Driver while operating in Host mode must also ensure that the `rootHubInterface` member of DRV_USB_HOST_INTERFACE is set up correctly so that the USB Host Stack can access the root hub functions. Descriptions of the function pointer types in the DRV_USB_ROOT_HUB_INTERFACE include:

- Driver Host Root Hub Speed Get Function
- Driver Host Root Hub Port Numbers Get Function
- Driver Host Root Hub Maximum Current Get Function
- Driver Host Root Hub Operation Enable Function
- Driver Host Root Hub Operation Enable Status Function
- Driver Host Root Hub Initialize Function

Driver Host Root Hub Speed Get Function

The `rootHubSpeedGet` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Speed Get function. The signature of this function is as follows:

```
USB_SPEED (*rootHubSpeedGet)(DRV_HANDLE handle);
```

The USB Driver Root Hub Speed Get function must match this signature. The USB Host Stack calls this function to identify the speed at which the root hub is operating. The `handle` parameter is the handle obtained by calling the USB Driver Open function. The operation speed is configured by the USB Driver initialization and depends on the capability of the USB peripheral. For example, the USB peripheral on PIC32MZ devices supports both Hi-Speed and Full-Speed Host mode operation. It can be configured through initialization to only operate at Full-Speed. The Root Hub Speed Get function must return the USB speed at which the USB peripheral is operating. This should not be confused with the speed of the attached device.

Driver Host Root Hub Port Numbers Get Function

The `rootHubPortNumbersGet` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Port Numbers Get function. The signature of this function is as follows:

```
USB_SPEED (*rootHubSpeedGet)(DRV_HANDLE handle);
```

The USB Driver Root Hub Speed Get function must match this signature. This function should return the number of ports that the root hub contains. On the USB peripheral for both PIC32MZ and PIC32MX devices, this value is always '1'.

Driver Host Root Hub Maximum Current Get Function

The `rootHubMaxCurrentGet` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Maximum Current Get function. The signature of this function is as follows:

```
uint32_t (*rootHubMaxCurrentGet)(DRV_HANDLE handle);
```

The USB Driver Root Hub Maximum Current Get function must match this signature. This function returns the maximum VBUS current that the root hub can provide. The USB Host Stack calls this function to know the maximum current that the root hub VBUS power supply can provide. This

value is then used to determine if the Host can support the current requirements of the attached device. The `handle` parameter is the driver handle obtained by calling the driver Open function.

The PIC32MX and the PIC32MZ USB peripherals cannot supply VBUS. The root hub driver only switches the VBUS supply. The current rating of the VBUS is specified through the USB Driver initialization. The root hub maximum current get function implementation in these drivers returns this value to the Host Stack.

Driver Host Root Hub Operation Enable Function

The `rootHubOperationEnable` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Operation Enable function. The signature of this function is as follows"

```
void (*rootHubOperationEnable)(DRV_HANDLE handle, bool enable);
```

The USB Driver Root Hub Operation Enable function must match this signature. The USB Host Stack calls this function when it ready to receive device attach events from the root hub. Calling this function will cause the USB Driver root hub functionality to enable detection of device attach and detach. The USB Driver will then raise events to the USB Host Stack. The `handle` parameter is the driver handle obtained by calling the driver Open function. Setting the `enable` parameter to true enables the root hub operation. Setting the `enable` parameter to false disables the root hub operation.

Driver Host Root Hub Operation Enable Status Function

The `rootHubOperationIsEnabled` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Operation Enable Status function. The signature of this function is as follows:

```
bool (*rootHubOperationIsEnabled)(DRV_HANDLE handle);
```

The USB Driver Root Hub Operation Enable Status function must match this signature. This USB Host Stack calls this function after calling the operation enable function to check if this has completed. The function returns true if the operation enable function has completed. The USB Host Stack will call this function periodically until it returns true.

Driver Host Root Hub Initialize Function

The `rootHubInitialize` member of the DRV_USB_ROOT_HUB_INTERFACE structure should point to the USB Driver Root Hub Initialize function. The signature of this function is as follows:

```
void (*rootHubInitialize)(DRV_HANDLE handle, USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo);
```

The USB Driver Root Hub Initialize function must match this signature. The USB Host Stack calls this function to assign a device identifier (usbHostDeviceInfo) to the root hub. This function is called before the Host Stack enables the root hub operation. The USB Driver root hub should use this identifier as the parent identifier when it calls the USB_HOST_DeviceEnumerate function to enumerate the attached device. At the time of enumeration, the USB Host Stack will use this parent identifier to identify the parent hub (whether root hub or external hub) of the attached device. The USB Driver root hub should retain the `usbHostDeviceInfo` parameter for the life time of its operation.

Driver Device Mode Client Functions

Provides information on the USB Driver Device mode Client functions.

Description

The DRV_USB_DEVICE_INTERFACE structure contains pointers to the USB Driver's Device mode Client Functions. These functions are only applicable when the USB module is operating as a USB Device. A USB Driver must implement these functions and ensure that the Device Stack can access these functions through the driver's DRV_USB_DEVICE_INTERFACE structure. Descriptions of the Driver Device Mode Client functions in the DRV_USB_DEVICE_INTERFACE structure include:

- Driver Device Address Set Function
- Driver Device Current Speed Get Function
- Driver Device SOF Number Get Function
- Driver Device Attach Function
- Driver Device Detach Function
- Driver Device Endpoint Enable Function
- Driver Device Endpoint Disable Function
- Driver Device Endpoint Stall Function
- Driver Device Endpoint Stall Clear Function
- Driver Device Endpoint Enable Status Function
- Driver Device Endpoint Stall Status Function
- Driver Device IRP Submit Function
- Driver Device IRP Cancel All Function
- Driver Device IRP Cancel Function
- Driver Device Remote Wakeup Start Function
- Driver Device Remote Wakeup Stop Function
- Driver Device Test Mode Enter Function

The PIC32MZ and the PIC32MX USB peripheral drivers implement the Device mode functions and export these functions to the Device Stack

though their respective DRV_USB_DEVICE_INTERFACE structure.

Driver Device Address Set Function

The `deviceAddressSet` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device Address Set function. The signature of this function is as follows:

```
void (*deviceAddressSet)(DRV_HANDLE handle, uint8_t address);
```

The USB Driver Device Address Set Function should match this signature. The USB Device Stack will call this function to set the Device USB Address. The function will be called in an interrupt context and hence the function implementation must be interrupt-safe. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `address` parameter is the address provided by the USB Host through the Set Device Address Standard request.

Driver Device Current Speed Get Function

The `deviceCurrentSpeedGet` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Current Speed Get function. The signature of this function is as follows:

```
USB_SPEED (*deviceCurrentSpeedGet)(DRV_HANDLE handle);
```

The USB Driver Device Current Speed Get function should match this signature. The USB Device Stack will call this function to obtain the speed at which the device has connected to the USB. It will call this function after reset signaling has completed. The `handle` parameter is driver handle obtained from calling the driver Open function. This function is called in an interrupt context and should be interrupt-safe.

Driver Device SOF Number Get Function

The `deviceSOFNumberGet` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Start-Of-Frame Number Get function. The signature of this function is as follows:

```
uint16_t (*deviceSOFNumberGet)(DRV_HANDLE handle);
```

The USB Driver SOF Number Get function should match this signature. The USB Device Stack will call this function to obtain the current SOF number. The USB peripheral uses a 16 bit counter to count the number of SOFs that have occurred since USB reset. This value is returned along with the Device Stack Start of Frame event. This function is called from an interrupt context and should be interrupt-safe. The `handle` parameter is the driver handle obtained from calling the driver Open function.

Driver Device Attach Function

The `deviceAttach` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Attach function. The signature of this function is as follows:

```
uint16_t(*deviceAttach)(DRV_HANDLE handle);
```

The USB Driver Attach function should match this signature. The USB Device Stack will call this function when the Device application calls the USB Device Stack Device Attach function. The USB Driver will enable the required signaling resistors for indicate attach to the Host. The application could call this function in response to a VBUS power available event. This function must be interrupt-safe. The `handle` parameter is the driver handle obtained from calling the driver Open function.

Driver Device Detach Function

The deviceDetach member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Detach function. The signature of this function is as follows:

uint16_t(*deviceDetach)(DRV_HANDLE handle);

The USB Driver Detach function should match this signature. The USB Device Stack will call this function when the Device application calls the USB Device Stack Device Detach function. The USB Driver will disable the required signaling resistors to indicate detach to the Host. The application could call this function in response to a VBUS power not available event. This function should be interrupt-safe. The `handle` parameter is driver handle obtained from calling the driver Open function.

Driver Device Endpoint Enable Function

The `deviceEndpointEnable` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Endpoint Enable function. The signature of this function is as follows:

```
USB_ERROR (*deviceEndpointEnable)(DRV_HANDLE handle, USB_ENDPOINT endpoint,
            USB_TRANSFER_TYPE transferType, uint16_t endpointSize);
```

The USB Driver Endpoint Enable function should match this signature. The USB Device Stack Function Driver will call this function when it is initialized by the USB Device Layer. The Device Layer, on receiving the Set Configuration request from the Host, identifies the function drivers that are required by the configuration and initializes them. The function drivers will call the endpoint enable function to enable the endpoints required for their operation. Enabling the endpoint will cause it reply to transaction requests from the Host and accept transfer requests from the device application.

The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) that should be enabled. The transferType is the type of the USB transfer that this endpoint will handle. The endpointSize is the size of the maximum transaction that the endpoint will handle. This should match the endpoint size communicated to the Host via the device endpoint descriptors.

The function will return USB_ERRROR_NONE if the endpoint was configured successfully. The function will return USB_ERROR_DEVICE_ENDPOINT_INVALID if the specified endpoint is not provisioned in the system configuration. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid.

The endpoint enable function will be called in an interrupt context and should be interrupt-safe. It is not expected to be thread safe. For standard function drivers, the endpoint enable function will be called in the context of the USB Device Layer Client. For vendor USB devices, the vendor application must call the endpoint enable function in response to and within the context of the device configured event. Again this event itself will execute in the context of the Device Layer.

Driver Device Endpoint Disable Function

The `deviceEndpointDisable` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Endpoint Disable function. The signature of this function is as follows:

```
USB_ERROR (*deviceEndpointDisable)(DRV_HANDLE handle, USB_ENDPOINT endpoint);
```

The USB Driver Endpoint Disable function should match this signature. The USB Device Stack Function Driver will call this function when it is deinitialized by the USB Device Layer. The Device Layer will deinitialize function drivers when it receives a USB reset event from the driver or on receiving the Set Configuration request from the Host with configuration parameter 0. Disabling the endpoint will cause it NAK transaction request from the Host and not accept transfer requests from the device application.

The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) that should be disabled.

The function will return USB_ERRROR_NONE if the function executed successfully. The function will return USB_ERROR_DEVICE_ENDPOINT_INVALID if the specified endpoint is not provisioned in the system configuration. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid.

The endpoint disable function will be called in an interrupt context and should be interrupt-safe. It is not expected to be thread safe. For standard function drivers, the endpoint disable function will be called in the context of the USB Device Layer Client. For vendor USB devices, the vendor application must call the endpoint enable function in response to and within the context of the device reset event. Again this event itself will execute in the context of the Device Layer. Disabling the endpoint will not cancel any transfers that have been queued against the endpoint. The function drivers will call the IRP Cancel All function to cancel any pending transfers.

Driver Device Endpoint Stall Function

The `deviceEndpointStall` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Endpoint Stall function. The signature of this function is as follows:

```
USB_ERROR (*deviceEndpointStall)(DRV_HANDLE handle, USB_ENDPOINT endpoint);
```

The USB Driver Endpoint Stall function should match this signature. The USB Device Stack Function Driver will call this function to stall an endpoint. The Device Layer itself will stall endpoint 0 for several reasons including non-support of the Host request or failure while executing the request. A function driver will also stall an endpoint for protocol specific reasons. The driver will stall both, receive and transmit directions when stalling Endpoint 0. The driver will stall the specified direction while stalling a non-zero endpoint.

This function must be thread safe and interrupt safe. Stalling the endpoint will abort all the transfers queued on the endpoint with the completion status set to USB_DEVICE_IRP_STATUS_ABORTED_ENDPOINT_HALT. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) that should be stalled. The function will return USB_ERRROR_NONE if the function executed successfully. The function will return USB_ERROR_DEVICE_ENDPOINT_INVALID if the specified endpoint is not provisioned in the system configuration. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid.

Driver Device Endpoint Stall Clear Function

The `deviceEndpointStallClear` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Endpoint Stall Clear function. The signature of this function is as follows:

```
USB_ERROR (*deviceEndpointStallClear)(DRV_HANDLE handle, USB_ENDPOINT endpoint);
```

The USB Driver Endpoint Stall Clear function should match this signature. The USB Device Stack Function Driver will call this function to clear the stall on a non-zero endpoint. The Device Layer will call this function to clear the stall condition on Endpoint 0. Clearing the stall on a non-zero endpoint will clear all transfers scheduled on the endpoint and transfer completion status will be set to USB_DEVICE_IRP_STATUS_TERMINATED_BY_HOST. When the stall is cleared, the data toggle for non-zero endpoint will be set to DATA0. The data toggle on Endpoint 0 OUT endpoint will be set to DATA1. The USB Driver will clear the Stall condition on an endpoint even if it was not stalled.

This function must be thread safe and interrupt safe. Stalling the endpoint will flush all the transfers queued on the endpoint. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) whose stall condition must be cleared. The function will return USB_ERRROR_NONE if the function executed successfully. The function will return USB_ERROR_DEVICE_ENDPOINT_INVALID if the specified endpoint is not provisioned in the system configuration. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid.

Driver Device Endpoint Enable Status Function

The `deviceEndpointIsEnabled` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Endpoint Enable Status function. The signature of this function is as follows:

```
bool (*deviceEndpointIsEnabled)(DRV_HANDLE handle, USB_ENDPOINT endpoint);
```

The USB Driver Endpoint Enable Status function should match this signature. The USB Device Stack function will call this function to check if an endpoint has been enabled. The function returns true if the endpoint is enabled. The endpoint is enabled through the USB Driver Endpoint Enable function. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) whose enable status needs to be queried.

Driver Device Endpoint Stall Status Function

The `deviceEndpointIsStalled` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Endpoint Stall Status function. The signature of this function is as follows:

```
bool (*deviceEndpointIsStalled)(DRV_HANDLE handle, USB_ENDPOINT endpoint);
```

The USB Driver Endpoint Stall Status function should match this signature. The USB Device Stack function will call this function to check if an endpoint has been stalled. The function returns true if the endpoint is stalled. The endpoint is stalled through the USB Driver Endpoint Stall function. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) whose stall status needs to be queried.

Driver Device IRP Submit Function

The deviceIRPSubmit member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device IRP Submit function. The signature of the IRP submit function is as follows:

```
USB_ERROR (*deviceIRPSubmit)(DRV_HANDLE handle, USB_ENDPOINT endpoint, USB_DEVICE_IRP * irp);
```

The USB Driver Device IRP Submit function must match this signature. The Device Stack (USB Device calls this function to submit an IRP to the USB Driver. The USB Driver provides this mechanism to transfer data between the device and the Host. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter should set to endpoint through which transfer must be processed. The `irp` parameter should point to the Device IRP data structure. The IRP data structure will transport an entire transfer over the endpoint. The USB Driver will split up the transfer into transactions based on the endpoint size specified at the time of enabling the endpoint. This process does not require Device Stack intervention.

The function will return USB_ERRROR_NONE if the function executed successfully. The function will return USB_ERROR_DEVICE_ENDPOINT_INVALID if the specified endpoint is not provisioned in the system configuration. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid. It will return USB_ERROR_DEVICE_IRP_IN_USE if an in progress IRP is resubmitted. It will return USB_ERROR_ENDPOINT_NOT_CONFIGURED if the IRP is submitted to an endpoint that is not enabled.

The USB Driver will queue the IRP if there is already an IRP being processed on the endpoint. The completion of the IRP processing is indicated by the USB Driver calling the IRP callback function specified within the IRP. The Device IRP Submit function must be thread safe and IRP callback safe. The Device Stack may resubmit the IRP within the IRP callback function. The IRP callback function itself executes within an interrupt context. The completion status of the IRP will be available in the status member of the IRP when the IRP callback function is invoked.

Driver Device IRP Cancel All Function

The `deviceIRPCancelAll` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device IRP Cancel All function. The signature of this is as follows:

```
USB_ERROR (*deviceIRPCancelAll)(DRV_HANDLE handle, USB_ENDPOINT endpoint);
```

The USB Driver Device IRP Cancel All function must match this signature. The USB Device Stack will call this function before disabling the endpoint. Calling this function will call all IRPs that are queued on the endpoint to be canceled. The callback of each IRP will be invoked and the IRP completion status will be set to USB_DEVICE_IRP_STATUS_ABORTED. If an IRP is in progress, an ongoing transaction will be allowed to complete and pending transactions will be canceled. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `endpoint` parameter is the USB endpoint (which indicates the direction along with endpoint number) whose queued IRPs must be canceled.

The function is thread safe and interrupt safe and will return USB_ERRROR_NONE if it executed successfully. The function will return USB_ERROR_DEVICE_ENDPOINT_INVALID if the specified endpoint is not provisioned in the system configuration. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid.

Driver Device IRP Cancel Function

The `deviceIRPCancel` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device IRP Cancel function. The signature of this is as follows:

```
USB_ERROR (*deviceIRPCancel)(DRV_HANDLE handle, USB_DEVICE_IRP * IRP);
```

The USB Driver Device IRP Cancel function must match this signature. This function is called by the USB Device Stack function driver to cancel a scheduled IRP. If the IRP is in the queue but it's processing has not started, the IRP will removed from the queue and the IRP callback function will be called from within the cancel function. The callback will be invoked with the IRP completion status set to USB_DEVICE_IRP_STATUS_ABORTED. If an IRP is in progress, an ongoing transaction will be allowed to complete and pending transactions will be canceled. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `irp` parameter is the IRP to be canceled.

The function is thread safe and will return USB_ERRROR_NONE if it executed successfully. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid or if the IRP has status indicates that this IRP is not queued or not in progress. The application should not release the data memory associated with IRP unless the callback has been received.

Driver Device Remote Wakeup Start Function

The `deviceRemoteWakeupStart` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device Remote Wakeup Start function. The signature of this function is as follows:

```
void (*deviceRemoteWakeupStart)(DRV_HANDLE handle);
```

The USB Driver Device Remote Wakeup Start function must match this signature. The USB Device Stack will call the function when the device application wants to start remote wakeup signaling. This would happen if the device supports remote wake-up capability and this has been enabled by the Host. The `handle` parameter is the driver handle obtained from calling the driver Open function.

Driver Device Remote Wakeup Stop Function

The `deviceRemoteWakeupStop` member of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device Remote Wakeup Stop function. The signature of this function is as follows:

```
void (*deviceRemoteWakeupStop)(DRV_HANDLE handle);
```

The USB Driver Device Remote Wakeup Stop function must match this signature. The USB Device Stack will call the function when the device application wants to stop remote wakeup signaling. The application would call after calling the remote wakeup start function. The `handle` parameter is the driver handle obtained from calling the driver Open function.

Driver Device Test Mode Enter Function

The `deviceTestModeEnter` parameter of the DRV_USB_DEVICE_INTERFACE structure should point to the USB Driver Device Test Mode Enter function. The signature of this function is as follows:

```
USB_ERROR (*deviceTestModeEnter)(DRV_HANDLE handle, USB_TEST_MODE_SELECTORS testMode);
```

The USB Driver Device Test Mode Enter function should match this signature. The USB Device Stack calls this driver function to place the driver into test mode. This is required when the USB Host (operating at Hi-Speed) send the Set Feature request with the feature selector test set to test mode. This request also specifies which of the test mode signals, the driver should enable. The `handle` parameter is the driver handle obtained from calling the driver Open function. The `testMode` parameter should be set to one of the test modes as defined in table 9-7 of the USB 2.0 specification.

The test mode enter function is only supported by the PIC32MZ USB Driver as the USB peripheral on this controller supports Hi-Speed operation. The function will return USB_ERRROR_NONE if it executed successfully. It will return USB_ERROR_PARAMETER_INVALID if the driver handle is not valid.

This concludes the discussion on the DRV_USB_DEVICE_INTERFACE structure. The following sections describe using the USB Common Driver.

Opening the Driver

Provides information and examples for opening the driver.

Description

The USB Host Stack and the USB Device Stack must obtain a handle to the USB Driver to access the functionality of the driver. This handle is obtained through the USB Driver Open function. The DRV_USB_DEVICE_INTERFACE structure and DRV_USB_DEVICE_HOST_INTERFACE structure provide access to the USB Driver Open function through the `open` member of these structures. Calling the Open function may not return a valid driver handle the first time the function is called. In fact, the USB Driver will return an invalid driver handle until the driver is ready to be opened. The Host and the Device Stack call the Open function repetitively in a state machine, until the function returns a valid handle.

The USB Host Stack can open the USB Driver but can call its Host mode functions only if the USB Driver was initialized for Host mode or Dual Role operation. The USB Host Stack accesses the driver functions through the DRV_USB_HOST_INTERFACE pointer that was provided to the Host Layer through the Host Stack initialization. The USB Device Stack can open the USB Driver but can call its Device mode functions only if the USB Driver was initialized for Device mode or Dual Role operation. The USB Device Stack accesses the driver functions through the DRV_USB_HOST_INTERFACE pointer that was provided to the Host Layer through the Host Stack initialization

The following code example shows how the USB Host Layer opens the USB Driver.

```
/* This code example shows how the Host Layer open the HCD via the hcdInterface.
 * The driver handle is stored in hcdHandle member of the busObj data structure.
 * The busObj data structure Host Layer local data structure. The Host Layer
 * opens the HCD when the bus is enabled. This operation takes place in the
 * USB_HOST_BUS_STATE_ENABLING state. */

/* Note the Host Layer calls the Open function by accessing the open member of
 * the hcdInterface which is of the type DRV_USB_HOST_INTERFACE. Also note how
 * the function is called repetitively until the Open function returns a valid
 * handle. */

case USB_HOST_BUS_STATE_ENABLING:

    /* The bus is being enabled. Try opening the HCD */
    busObj->hcdHandle = busObj->hcdInterface->open(busObj->hcdIndex, DRV_IO_INTENT_EXCLUSIVE |
            DRV_IO_INTENT_NONBLOCKING | DRV_IO_INTENT_READWRITE );

    /* Validate the Open function status */
    if (DRV_HANDLE_INVALID == busObj->hcdHandle )
    {
        /* The driver may not open the first time. This is okay. We
         * should try opening it again. The state of bus is not
         * changed. */
    }
```

The following code example shows how the USB Device Layer opens the USB Driver.

```
/* This code example shows how the USB Device Layer calls the USBCD open
```

```
    * function to open the USBCD. The Device Layer accesses the USBCD Open function
    * through the driverInterface member of the usbDeviceInstanceState object. The
    * driverInterface member is a DRV_USB_DEVICE_INTERFACE type. The
    * usbDeviceInstanceState is a USB Device Layer local object. */

   /* The Device Layer attempts to open the USBCD when it is initializing. Note how
    * the Device Layer advances to the next state only when the USBCD returns a
    * valid handle.   */

   switch(usbDeviceThisInstance->taskState)
   {
       case USB_DEVICE_TASK_STATE_OPENING_USBCD:

           /* Try to open the driver handle. This could fail if the driver is
            * not ready to be opened. */
           usbDeviceThisInstance->usbCDHandle =
              usbDeviceThisInstance->driverInterface->open( usbDeviceThisInstance->driverIndex,
              DRV_IO_INTENT_EXCLUSIVE|DRV_IO_INTENT_NONBLOCKING|DRV_IO_INTENT_READWRITE);

           /* Check if the driver was opened */
           if(usbDeviceThisInstance->usbCDHandle != DRV_HANDLE_INVALID)
           {
               /* Yes the driver could be opened. */

               /* Advance the state to the next state */
               usbDeviceThisInstance->taskState = USB_DEVICE_TASK_STATE_RUNNING;

               /* Update the USB Device Layer state to indicate that it can be
                * opened */
               usbDeviceThisInstance->usbDeviceInstanceState = SYS_STATUS_READY;
           }

           break;
```

USB Driver Host Mode Operation

Provides information on Host mode operation.

Description

The USB Driver operates or can operate in the Host mode when it is initialized for Host mode or Dual Role operation. When operating in Host mode, the USB Driver is also referred to as the Host Controller Driver (HCD). In Dual Role mode, the USB Driver will switch to Host mode when the USB Driver Host Root Hub Operation Enable function is called.

The USB Driver Client must perform these steps to operate the USB Driver in Host mode.

1. Open the USB Driver to obtain the driver handle.
2. Set the event handler.
3. Call the Root Hub Control function to obtain the speed of the root hub, the number of ports that the root hub supports, and the maximum current that the root hub VBUS can supply.
4. Calls the Root Hub Initialize function with an identifier parameter. This `identifier` parameter allows the Host Stack to uniquely identify the root hub when where there are multiple root hubs.
5. The Driver Client will then enable the root hub operation and will wait until the root hub operation is enabled.
6. The Driver Client can now call the USB Driver Host mode functions.

The following sections explain Steps 2 through 6 in more detail.

Handling Host Mode Driver Events

Currently, the HCD does not provide any events to the client. The client can optionally register an event handler through the eventHandlerSet function pointer in the DRV_USB_HOST_INTERFACE structure. Future releases of the USB Driver may contain features that provide events to the Driver Client. Please refer to the following **Root Hub Operation** section for details on how the driver indicates device attach and detach to the client.

Root Hub Operation

A key feature of the HCD is the Root Hub Driver. The Root Hub Driver emulates hub operation in USB Driver software and provides a hub like interface to the USB Host Layer. The USB Host Layer treats the root hub like an external hub. This simplifies the implementation of USB Host Layer while supporting multiple devices through a hub. In that, the USB Host layer does not have to treat a device connected directly to the USB peripheral differently than a device connected to an external hub. The following code example shows how the USB Host Layer calls the root hub function to obtain information about the root hub.

```
/* This code example shows how the USB Host Layer calls the root hub functions to
 * obtain information about the root. The USB Host Layer first opens the HCD and
 * then accesses the root hub functions through the rootHubInterface member of
 * hcdInterface. rootHubInterface is of the type DRV_USB_ROOT_HUB_INTERFACE and
 * the hcdInterface is of the type of DRV_USB_HOST_INTERFACE. */

/* The code example shows how the Host Layer gets to know the root hub operation
 * speed, number of root hub ports and the maximum amount of current that the
 * root can supply. These function can be called only after HCD was opened and a
 * valid driver handle obtained. */

case USB_HOST_BUS_STATE_ENABLING:

    /* The bus is being enabled. Try opening the HCD */
    busObj->hcdHandle = busObj->hcdInterface->open(busObj->hcdIndex, DRV_IO_INTENT_EXCLUSIVE |
            DRV_IO_INTENT_NONBLOCKING | DRV_IO_INTENT_READWRITE );

    /* Validate the Open function status */
    if (DRV_HANDLE_INVALID == busObj->hcdHandle )
    {
        /* The driver may not open the first time. This is okay. We
         * should try opening it again. The state of bus is not
         * changed. */
    }
    else
    {
        /* Update the bus root hub information with the
         * details of the controller. Get the bus speed, number of
         * ports, the maximum current that the HCD can supply,
         * pointer to the root hub port functions. */

        SYS_DEBUG_PRINT(SYS_ERROR_INFO,
             "\r\nUSB Host Layer: Bus %d Root Hub Driver Opened.",hcCount);

        busObj->rootHubInfo.speed =
             busObj->hcdInterface->rootHubInterface.rootHubSpeedGet(busObj->hcdHandle);

        busObj->rootHubInfo.ports =
             busObj->hcdInterface->rootHubInterface.rootHubPortNumbersGet(busObj->hcdHandle);

        busObj->rootHubInfo.power =
             busObj->hcdInterface->rootHubInterface.rootHubMaxCurrentGet(busObj->hcdHandle);

        busObj->rootHubInfo.rootHubPortInterface =
             busObj->hcdInterface->rootHubInterface.rootHubPortInterface;
```

The USB Host Layer must initialize and enable the operation of the root hub. While initializing the Root Hub Driver, the Host layer will assign a unique identifier to the root hub. The root hub will return this value as the parent identifier while calling the USB_HOST_DeviceEnumerate function. The USB Host Layer must then enable the operation of the root hub driver. This will cause the root hub driver to detect device attach and detach. The following code example shows how the USB Host Layer initializes and enables the root hub driver

```
/* The following code example show how the USB Host Layer initializes the root
 * hub and then enables the root hub operation. The
 * rootHubDevice->deviceIdentifier is a unique identifier that allows the USB
 * Host layer to identify this root hub. It is returned by the root hub driver
 * in the USB_HOST_DeviceEnumerate() function as the parent identifier when the
 * device is connected to the root hub. */

/* The hcdHandle is the driver handle. The hcdInterface pointer is of the type
 * DRV_USB_HOST_INTERFACE and points to the HCD interface. */

busObj->hcdInterface->rootHubInterface.rootHubInitialize( busObj->hcdHandle ,
                                                          rootHubDevice->deviceIdentifier );
busObj->hcdInterface->rootHubInterface.rootHubOperationEnable( busObj->hcdHandle , true );
```

When a device is attached, the Root Hub Driver will implement the required settling attach settling delay and will then call the USB Host Layer's USB_HOST_DeviceEnumerate function to enumerate the device. While calling this function, the root hub driver will provide the identifier that was provided to it in its initialize function. The USB_HOST_DeviceEnumerate function will return an identifier which uniquely identifies the attached device. The root hub driver uses this value to identify the device to the Host when the USB_HOST_DeviceDenumerate function is called on device detach. The following code example shows how the Root Hub driver calls the USB_HOST_DeviceEnumerate and the USB_HOST_DeviceDenumerate functions.

```
/* The following code shows how the root hub driver calls the
 * USB_HOST_DeviceEnumerate() function in the device attach interrupt. As seen
 * here, the root hub returns the identifier that the USB Host Layer assigned to
 * it the rootHubInitialize function call. The pUSBDrvObj->usbHostDeviceInfo
 * variable contains this identifier. */

if(PLIB_USB_InterruptFlagGet(usbID, USB_INT_ATTACH))
{
    /* We can treat this as a valid attach. We then clear the
     * detach flag and enable the detach interrupt. We enable
     * the Transaction interrupt */

    PLIB_USB_InterruptFlagClear(usbID, USB_INT_HOST_DETACH);
    PLIB_USB_InterruptEnable(usbID, USB_INT_HOST_DETACH);
    PLIB_USB_InterruptEnable(usbID, USB_INT_TOKEN_DONE);

    /* Ask the Host layer to enumerate this device. While calling
     * this function, the UHD of the parent device which is the
     * root hub in this case.
     * */
    pUSBDrvObj->attachedDeviceObjHandle = USB_HOST_DeviceEnumerate
                                          (pUSBDrvObj->usbHostDeviceInfo, 0);
}

/* The following code example shows how the root hub driver calls the
 * USB_HOST_DeviceDenumerate() function in the device detach interrupt. Note how
 * the attachedDeviceObjHandle that was assigned at the time of device
 * enumeration is returned to the Host Layer to let the Host know which device
 * is being detached. */

if((usbInterrupts & USB_INT_HOST_DETACH) && (enabledUSBInterrupts & USB_INT_HOST_DETACH))
{
    /* Perform other detach related handling */

    /* Ask the Host Layer to de-enumerate this device. */
    USB_HOST_DeviceDenumerate (pUSBDrvObj->attachedDeviceObjHandle);

    /* Disable the LS Direct Connect. It may have been enabled if the last
     attach was for a Low-Speed device. */
    PLIB_USB_EP0LSDirectConnectDisable(pUSBDrvObj->usbID);

    /* Continue to perform detach handling */
}
```

Root Hub Port Operation

The HCD Root Hub Driver exposes a set of port related functions that allow the USB Host Layer to control the port. The most commonly used functions are the function to reset the port and get the port speed. In this case, this is the speed of the attached device. The following code example shows how the USB Host Layer calls the hubPortReset, hubPortResetIsComplete and hubPortSpeedGet port functions.

```
/* The following code shows an example of how the Host Layer called the
 * hubPortReset function to reset the port to which the device is connected.
 * The code proceeds with the port reset if no device on the bus is in an
 * enumeration state. It will then call the hubPortReset function of the parent
 * hub of the device. The parent hub, hubInterface member of deviceObj points to
 * this driver, can be the root hub or an external hub */

if(!busObj->deviceIsEnumerating)
{
    /* Remember which device is enumerating */
    busObj->enumeratingDeviceIdentifier = deviceObj->deviceIdentifier;

    /* Grab the flag */
    busObj->deviceIsEnumerating = true;

    /* Reset the device */
    deviceObj->hubInterface->hubPortReset( deviceObj->hubHandle, deviceObj->devicePort );
}

/* The following code example shows how the Host checks if the port reset
```

```
 * operation has completed. If the reset operation has completed, the speed of
 * the attached device can be obtained. The reset settling delay can then be
 * started. */

case USB_HOST_DEVICE_STATE_WAITING_FOR_RESET_COMPLETE:

   /* Check if the reset has completed */
   if(deviceObj->hubInterface->hubPortResetIsComplete
               ( deviceObj->hubHandle ,deviceObj->devicePort ))
   {
       /* The reset has completed. We can also obtain the speed of the
        * device. We give a reset recovery delay to the device */

       deviceObj->speed = deviceObj->hubInterface->hubPortSpeedGet
                         (deviceObj->hubHandle, deviceObj->devicePort);

       deviceObj->deviceState = USB_HOST_DEVICE_STATE_START_RESET_SETTLING_DELAY;
   }
```

Opening and Closing a Pipe

The HCD client can open a pipe to the device after resetting the device. The USB Host Layer calls the hostPipeSetup function in the DRV_USB_HOST_INTERFACE structure to open a pipe. The USB Host Layer must open a pipe to communicate to a specific endpoint on a target device. While opening the pipe, the USB Host Layer must specify parameters which specify the address of the target device, the type of the transfer that the pipe must support and the speed of the pipe. If the device is connected to a hub, the address of the hub must be specified. The HCD Pipe Setup function *is not* interrupt-safe. It should not be called in any event handler that executes in an interrupt context.

The Pipe Setup function returns a valid pipe handle if the pipe was opened successfully. Pipe creation may fail if the target device was disconnected or if there are insufficient resources to open the pipe. The pipe handle is then used along with the hostIRPSubmit function to transfer data between the Host and the device. The following code shows example usage of a Pipe Open function.

```
/* The following code example shows how the Host Layer uses the hostPipeSetup
 * function to open a control pipe to the attached device. Most of the
 * parameters that are passed to this function become known when the device is
 * attached. The pipe handle is checked for validity after the hostPipeSetup
 * function call. */

if(busObj->timerExpired)
{
    busObj->busOperationsTimerHandle = SYS_TMR_HANDLE_INVALID;
    /* Settling delay has completed. Now we can open default address
     * pipe and and get the configuration descriptor */

    SYS_DEBUG_PRINT(SYS_ERROR_INFO,
                    "\r\nUSB Host Layer: Bus %d Device Reset Complete.", busIndex);

    deviceObj->controlPipeHandle =
               deviceObj->hcdInterface->hostPipeSetup( deviceObj->hcdHandle,
               USB_HOST_DEFAULT_ADDRESS , 0 /* Endpoint */,
               deviceObj->hubAddress /* Address of the hub */,
               deviceObj->devicePort /* Address of the port */,
               USB_TRANSFER_TYPE_CONTROL, /* Type of pipe to open */
               0 /* bInterval */, 8 /* Endpoint Size */, deviceObj->speed );

    if(DRV_USB_HOST_PIPE_HANDLE_INVALID == deviceObj->controlPipeHandle)
    {
        /* We need a pipe else we cannot proceed */
        SYS_DEBUG_PRINT(SYS_ERROR_DEBUG,
        "\r\nUSB Host Layer: Bus %d Could not open control pipe. Device not supported.", busIndex);
    }
}
```

An open pipe consumes computational and memory resources and must therefore must be closed if it will not be used. This is especially true of pipes to a device that is detached. The Host Layer calls the hostPipeClose function in the DRV_USB_HOST_INTERFACE structure to close the pipe. The pipe to be closed is specified by the pipe handle. The Pipe Close function can be called from an event handler. It is interrupt safe. Closing a pipe will cancel all pending transfers on that pipe. The IRP callback for such canceled transfers will be called with the status USB_HOST_IRP_STATUS_ABORTED. The following code example shows an example of closing the pipe.

```
/* The following code example shows an example of how the Host Layer calls the
 * hostPipeClose function to close an open pipe. Pipe should be closed if it
 * will not used. An open pipe consumes memory resources. In this example, the
 * Host Layer closes the pipe if it was not able successfully submit an IRP to
 * this pipe. */
```

```
/* Submit the IRP */
if(USB_ERROR_NONE != deviceObj->hcdInterface->hostIRPSubmit
                  ( deviceObj->controlPipeHandle, & (deviceObj->controlTransferObj.controlIRP)))
{
    /* We need to be able to send the IRP. We move the device to
     * an error state. Close the pipe and send an event to the
     * application. The assigned address will be released when
     * the device in unplugged. */

    SYS_DEBUG_PRINT(SYS_ERROR_DEBUG,
        "\r\nUSB Host Layer: Bus %d Set Address IRP failed. Device not supported.", busIndex);

    /* Move the device to error state */
    deviceObj->deviceState = USB_HOST_DEVICE_STATE_ERROR;

    /* Close the pipe as we are about mark this device as unsupported. */
    deviceObj->hcdInterface->hostPipeClose(deviceObj->controlPipeHandle);
}
```

Transferring Data to an Attached Device

The USB Host Layer, the HCD client, needs to transfer data to the attached device to understand the device capabilities and to operate the device. The HCD uses a concept of Input Output Request Packet (IRP) to transfer data to and from the attached device. IRPs are transported over pipes which are setup by calling the USB Driver Pipe Setup function.

A Host IRP is a USB_HOST_IRP type data structure. The IRP is created by the Host layer and submitted to the HCD for processing through the hostIRPSubmit function. At the time of submitting the IRP, the pipe over which the IRP must be transported is specified. The data request in the IRP is transported using the attributes of pipe. When an IRP is submitted to the HCD, it is owned by the HCD and cannot be modified by the Host Layer until the HCD issues an IRP callback. The HCD will issue the IRP callback when it has completed or terminated processing of the IRP.

An IRP does not have its own transfer type. It inherits the properties of the pipe to which it is submitted. Hence an IRP becomes a control transfer IRP it was submitted to a control transfer pipe. A pipe allows multiple IRPs to be queued. This allows the Host Layer to submit IRPs to a pipe even while an IRP is being processed on the pipe. The HCD will process an IRP in the order that it was received. The following code example shows the USB_HOST_IRP data structure.

```
/* The following code example shows the USB_HOST_IRP structure. The Host Layer
 * uses this structure to place data transfer requests on a pipe. */

typedef struct _USB_HOST_IRP
{
    /* Points to the 8 byte setup command packet in case this is a IRP is
     * scheduled on a CONTROL pipe. Should be NULL otherwise */
    void * setup;

    /* Pointer to data buffer */
    void * data;

    /* Size of the data buffer */
    unsigned int size;

    /* Status of the IRP */
    USB_HOST_IRP_STATUS status;

    /* Request specific flags */
    USB_HOST_IRP_FLAG flags;

    /* User data */
    uintptr_t userData;

    /* Pointer to function to be called when IRP is terminated. Can be NULL, in
     * which case the function will not be called. */
    void (*callback)(struct _USB_HOST_IRP * irp);

    /*************************************
     * These members of the IRP should not be
     * modified by client
     ************************************/
   uintptr_t privateData[7];

} USB_HOST_IRP;
```

The `setup` member of the USB_HOST_IRP structure must point to the 8 byte setup packet for control transfers. The driver will send this 8 byte data in the Setup phase of the control transfer. It can be NULL for non-control transfers. This member is only considered if the IRP is submitted to a control transfer pipe. It is ignored for non-control transfer pipes. The structure of the setup command should match that specified in the USB 2.0 specification.

The `data` member of the USB_HOST_IRP structure points to a data buffer. This data buffer will contain the data that needs to be sent to the device for data stage of a OUT transfer, or it will contain the data that was received from the device during an IN transfer. Any hardware specific cache coherency and address alignment requirements must be considered while allocating this data buffer. The Driver Client should not modify or examine the contents of the IRP after the IRP has been submitted and is being processed. It can be examined after the driver has released the IRP.

The `size` member of the USB_HOST_IRP structure contains the size of the transfer. for Bulk transfers, the size of the transfer can exceed the size of the transaction (which is equal to size of the endpoint reported by the device). The HCD in such a case will split up the transfer into transactions. This process does not require external intervention. For control transfers, the size of the transfer is specified in the setup packet (pointed to by the `setup` member of the USB_HOST_IRP structure). The driver will itself process the Setup, Data (if required) and Handshake stages of control transfer. This process again does not require external intervention. For interrupt and isochronous transfers, the size of transfer specified in the IRP cannot exceed the size of the transaction. If size is specified as 0, then the driver will send a zero length packet. The `size` parameter of the IRP is updated by the driver when IRP processing is completed. This will contain the size of the completed transfer.

The `status` member of the IRP provides the completion status of the IRP and should be checked only when the IRP processing has completed. This is indicated by the driver calling the IRP callback function. The IRP status is a USB_HOST_IRP_STATUS type. The following code example shows the different possible values of the `status` member and an example of submit a control transfer IRP.

```
/* The following code shows an example of how the Host Layer populates
 * the IRP object and then submits it. IRP_Callback function is called when an
 * IRP has completed processing. The status of the IRP at completion can be
 * checked in the status flag. The size field of the irp will contain the amount
 * of data transferred. */

void IRP_Callback(USB_HOST_IRP * irp)
{
    /* irp is pointing to the IRP for which the callback has occurred. In most
     * cases this function will execute in an interrupt context. The application
     * should not perform any hardware access or interrupt unsafe operations in
     * this function. */

    switch(irp->status)
    {
        case USB_HOST_IRP_STATUS_ERROR_UNKNOWN:
            /* IRP was terminated due to an unknown error */
            break;

        case USB_HOST_IRP_STATUS_ABORTED:
            /* IRP was terminated by the application */
            break;

        case USB_HOST_IRP_STATUS_ERROR_BUS:
            /* IRP was terminated due to a bus error */
            break;

        case USB_HOST_IRP_STATUS_ERROR_DATA:
            /* IRP was terminated due to data error */
            break;

        case USB_HOST_IRP_STATUS_ERROR_NAK_TIMEOUT:
            /* IRP was terminated because of a NAK timeout */
            break;

        case USB_HOST_IRP_STATUS_ERROR_STALL:
            /* IRP was terminated because of a device sent a STALL */
            break;

        case USB_HOST_IRP_STATUS_COMPLETED:
            /* IRP has been completed */
            break;

        case USB_HOST_IRP_STATUS_COMPLETED_SHORT:
            /* IRP has been completed but the amount of data processed was less
             * than requested. */
            break;

        default:
```

```
            break;
        }
    }

    /* In the following code example the a control transfer IRP is submitted to a
     * control pipe. The setup parameter of the IRP points to the Setup command of
     * the control transfer. The direction of the data stage is specified by the
     * Setup packet. */

    USB_HOST_IRP irp;
    USB_ERROR result;
    USB_HOST_PIPE_HANDLE controlPipe;
    USB_SETUP_PACKET setup;
    uint8_t controlTransferData[32];

    irp.setup = setup;
    irp.data = controlTransferData;
    irp.size = 32;
    irp.flags = USB_HOST_IRP_FLAG_NONE ;
    irp.userData = &someApplicationObject;
    irp.callback = IRP_Callback;

    result = DRV_USBFS_HOST_IRPSubmit(controlPipeHandle, &irp);

    switch(result)
    {
        case USB_ERROR_NONE:
            /* The IRP was submitted successfully */
            break;

        case USB_ERROR_HOST_PIPE_INVALID:
            /* The specified pipe handle is not valid */
            break;

        case USB_ERROR_OSAL_FUNCTION:
            /* An error occurred while trying to grab mutex */
            break;

        default:
            break;
    }
```

The `flags` member of the USB_HOST_IRP structure specifies flags which affect the behavior of the IRP. The USB_HOST_IRP_FLAG enumeration specifies the available option. The USB_HOST_IRP_FLAG_SEND_ZLP causes the driver to add a Zero Length Packet (ZLP) to the data stage of the transfer when the transfer size is an exact multiple of the endpoint size. The USB_HOST_IRP_WAIT_FOR_ZLP flag will cause the driver to wait for a ZLP from the device in a case where the size of data received thus far in the transfer is an exact multiple of the endpoint size.

The `callback` member of the USB_HOST_IRP structure points to a function which the driver calls when the IRP processing is completed. The Driver Client must implement this function and assign the pointer to this function to the `callback` member of the IRP. Every IRP can have its own callback function or one common callback function could be used. The callback function will execute in an interrupt context. The Driver Client should not execute interrupt unsafe, blocking, or computationally intensive operations in the callback function. The client can call hostIRPSubmit function in the IRP callback function to submit another IRP or resubmit the same IRP. The client can check the status and size of the IRP in the callback function.

The `userData` member of the USB_HOST_IRP structure can be used by the client to associate a client specific context with the Host. This context can then be used by the client, in the IRP callback function to identify the context in which the IRP was submitted. This member is particularly useful if the client wants to implement one callback function for all IRPs.

The `privateData` member of the IRP is used by the driver and should not be accessed or manipulated by the Driver Client. The following code examples show usage of IRPs to transfer data between the Host and the attached device and along with the different flags.

```
/* The following code shows an example of submitting an IRP to send data
 * to a device. In this example we will request the driver to send a ZLP after
 * sending the last transaction. The driver will send the ZLP only if the size
 * of the transfer is a multiple of the endpoint size. This is not a control
 * transfer IRP. So the setup field of the IRP will be ignored.  */

USB_HOST_IRP irp;
USB_ERROR result;
USB_HOST_PIPE_HANDLE bulkOUTPipeHandle;
uint8_t data[128];
```

```
irp.data = data;
irp.size = 128;
irp.flags = USB_HOST_IRP_FLAG_SEND_ZLP ;
irp.userData = &someApplicationObject;
irp.callback = IRP_Callback;

result = DRV_USBFS_HOST_IPRSubmit( bulkOUTPipeHandle, &irp );


/* The following code shows an example of submitting an IRP to receive
 * data to a device. In this example we will request the driver to wait for a
 * ZLP after receiving the last transaction. The driver will wait for the ZLP
 * only if the size of the transfer is a multiple of the endpoint size. This is
 * not a control transfer IRP. So the setup field of the IRP will be ignored.
 * */

USB_HOST_IRP irp;
USB_ERROR result;
USB_HOST_PIPE_HANDLE bulkINPipeHandle;
uint8_t data[128];

irp.data = data;
irp.size = 128;
irp.flags = USB_HOST_IRP_FLAG_WAIT_FOR_ZLP ;
irp.userData = &someApplicationObject;
irp.callback = IRP_Callback;

result = DRV_USBFS_HOST_IPRSubmit( bulkINPipeHandle, &irp );
```

USB Driver Device Mode Operation

Provides information on Device mode operation.

Description

The USB Driver operates can operate in the Device mode when it is initialized for Device mode or Dual Role operation. When operating in Device mode, the USB Driver is also referred to as the USB Controller Driver (USBCD). In Dual-Role mode, the USB Driver will switch to USBCD mode when the USB Driver Device Attach function is called.

The USB Driver Client must perform these steps to operate the USB Driver in Device mode.

1. Open the USB Driver to obtain the driver handle.
2. Set the event handler.
3. Wait for the application to attach the device to the bus.
4. Enable Endpoint 0 and respond to USB Host Enumeration requests.
5. Allow the application and function drivers to enable other endpoints and communicate with the Host.

The following sections discuss these operations in more detail.

General Device Mode Operations

Provides information on general Device mode operations.

Description

This section describes the USBCD operations such as setting event handlers and attaching and detaching the device.

Handling Device Mode Driver Events

The Device Layer will call the USBCD eventHandlerSet function to register the Device mode event handling function. The USBCD generates various events that indicate different states of the USB. These events are defined by the DRV_USB_EVENT enumeration. The following code example shows how the Device Layer registers the driver event handling function.

```
/* This code example shows the implementation of the USB_DEVICE_Attach and the
 * USB_DEVICE_Detach function. These functions are actually macro that map
 * directly deviceAttach and the deviceDetach function of the driverInterface
 * member of the deviceClient Object (which is the macro parameter) */

#define USB_DEVICE_Attach( x )  ((USB_DEVICE_OBJ *)x)->driverInterface->deviceAttach
                                ( ((USB_DEVICE_OBJ *)(x))->usbCDHandle)
#define USB_DEVICE_Detach( x )  ((USB_DEVICE_OBJ *)x)->driverInterface->deviceDetach
                                ( ((USB_DEVICE_OBJ *)x)->usbCDHandle)
```

If the driver is operating in interrupt mode, the client event handling function will execute in an interrupt context. The client should not call interrupt unsafe, computationally intensive or blocking functions in the event handler. The following code shows a small example of the Device Layer USBCD Event Handler:

```
/* This code example shows a partial implementation of the USB Device Layer
 * event handler. Note how the code type casts the referenceHandle parameter to
 * a USB_DEVICE_OBJ type. This referenceHandle is the same value that the Device
 * Layer passed when the event handler was set. This now easily allows one
 * implementation of the event handling code to be used by multiple Device
 * Layer instances. */

    void _USB_DEVICE_EventHandler
(
    uintptr_t referenceHandle,
    DRV_USB_EVENT eventType,
    void * eventData
)
{
    USB_DEVICE_OBJ* usbDeviceThisInstance;
    USB_DEVICE_MASTER_DESCRIPTOR * ptrMasterDescTable;
    USB_DEVICE_EVENT_DATA_SOF SOFFrameNumber;

    usbDeviceThisInstance = (USB_DEVICE_OBJ *)referenceHandle;

    /* Handle events, only if this instance is in initialized state */
    if( usbDeviceThisInstance->usbDeviceInstanceState <= SYS_STATUS_UNINITIALIZED )
    {
        /* The device should anyway not be attached when the Device Layer is
         * not initialized. If we receive driver event when the Device Layer is
         * not initialized, there is nothing we can do but ignore them. */
        return;
    }

    switch(eventType)
    {
        case DRV_USB_EVENT_RESET_DETECT:

            /* Clear the suspended state */
            usbDeviceThisInstance->usbDeviceStatusStruct.isSuspended = false;

            /* Cancel any IRP already submitted in the RX direction. */
            DRV_USB_DEVICE_IRPCancelAll( usbDeviceThisInstance->usbCDHandle,
                    controlEndpointRx );

            /* Code not shown for the sake of brevity. */

    }
}
```

In the previous code example, the Device Layer (the Driver Client) sets the `hReferenceData` parameter, of the Event Handler Set function, to point to a local object. This pointer is returned to the Device Layer, in the event handler when an event occurs. For multiple instances of USB drivers in one application, this allows the Device Layer to easily associate a Device Layer specific context to the driver instance, thus simplifying implementation of the event handler.

Attaching and Detaching the Device

The USB Device Layer calls the USBCD deviceAttach and deviceDetach functions to attach and detach the device on the USB. The USB Device Layer should be ready to handle events which would occur when the device is attached on the bus. Hence the USB Device Layer should register the USBCD event handler before the attach function is called. The deviceAttach and deviceDetach functions can be called in an interrupt context. These functions are respectively called when the USB Device application detects a valid VBUS voltage and when the VBUS voltage is not valid.

Setting the Device Address

The USB Device Layer will call the USBCD deviceAddressSet function to set the USB address of the device. The Device Layer will do this when it receives the Set Address control request from the Host. The USBCD will reset the device address to '0' when it has received reset signaling from the root hub. The following code example shows how the USB Device Layer calls this function.

```
/* The following code example shows how the USB Device Layer calls the
 * DRV_USB_DEVICE_AddressSet function to set the address. The
 * DRV_USB_DEVICE_AddressSet function is actually a macro that calls the
 * deviceAddressSet function of the driverInterface of usbDeviceThisInstance
 * object. The usbDeviceThisInstance is Device Layer object.
```

```
 *
 * As seen in this code, the Device Layer calls the address set function when
 * the it a pending set address control request from the Host has completed. */


void _USB_DEVICE_Ep0TransmitCompleteCallback(USB_DEVICE_IRP * handle)
{
    USB_DEVICE_IRP * irpHandle = (USB_DEVICE_IRP *)handle;
    USB_DEVICE_OBJ * usbDeviceThisInstance;
    USB_DEVICE_CONTROL_TRANSFER_STRUCT * controlTransfer;

    usbDeviceThisInstance = (USB_DEVICE_OBJ *)irpHandle->userData;
    controlTransfer = &(usbDeviceThisInstance->controlTransfer);

    if(irpHandle->status == USB_DEVICE_IRP_STATUS_ABORTED)
    {
        return;
    }

    if(usbDeviceThisInstance->usbDeviceStatusStruct.setAddressPending)
    {
        DRV_USB_DEVICE_AddressSet(usbDeviceThisInstance->usbCDHandle,
                                  usbDeviceThisInstance->deviceAddress);
        usbDeviceThisInstance->usbDeviceStatusStruct.setAddressPending = false;
    }

    /* Code not shown for the sake of brevity */

}
```

Device Current Speed and SOF Number

The USB Device Layer will call the USBCD deviceCurrentSpeedGet function to know the speed at which the device is attached to the USB. This allows the Device Layer to select the correct endpoint settings at the time of processing the Set Configuration request issued by the Host. The USB Device Layer will call the deviceSOFNumberGet function to return the SOF number at the time of the SOF event.

Device Remote Wake-up

The USB Device Layer will call the USBCD deviceRemoteWakeupStop and deviceRemoteWakeupStart functions to stop and start remote signaling. The Device layer application will call the USB Device Layer Stop and Start Remote Wakeup Signaling functions to remotely let the root hub know that the device is ready to be woken up. The timing of the remote signaling is controlled by the Device Layer. The client should call the remote wakeup function only when the device is suspended by the Host.

Device Endpoint Operations

Provides information on Device Endpoint operations.

Description

The UBSCD Endpoint functions allow the Driver Client to enable, disable, stall and clear the stall condition on an endpoint. The client submits requests to transmit and receive data from the USB Host on an endpoint.

Endpoint Enable and Disable functions

The USBCD client must enable an endpoint it must use the endpoint for communicating with the USB Host. The client will call the USBCD deviceEndpointEnable function to enable the endpoint. While calling this function, the client must specify the endpoint address, the transfer type to be processed on this endpoint and the maximum size of a transaction on this endpoint. This function is thread-safe when called in an RTOS application. The USBCD allows an endpoint to be accessed by one thread only. The USB Device Layer and the device function drivers will enable the endpoint when the Host sets the device configuration. The USBCD deviceEndpointIsEnabled function is available to check if an endpoint is enabled. The following code example shows how the USB Device Layer enables the device endpoint.

```
/* The following code example shows the USB Device Layer enables Endpoint 0 to
 * prepare for the enumeration process after it has received reset signaling
 * from the Host. The Device Layer calls the deviceEndpointEnable function to
 * to enable the endpoint. The driverInterface member of the
 * usbDeviceThisInstance structure points to the USB Device Mode Driver Common
 * Interface. */

void _USB_DEVICE_EventHandler
(
    uintptr_t referenceHandle,
```

```
    DRV_USB_EVENT eventType,
    void * eventData
)
{
    /* Code not shown due to space constraints */

    switch(eventType)
    {
        case DRV_USB_EVENT_RESET_DETECT:

            /* Clear the suspended state */
            usbDeviceThisInstance->usbDeviceStatusStruct.isSuspended = false;

            /* Cancel any IRP already submitted in the RX direction. */
            DRV_USB_DEVICE_IRPCancelAll( usbDeviceThisInstance->usbCDHandle,
                    controlEndpointRx );

            /* Cancel any IRP already submitted in the TX direction. */
            DRV_USB_DEVICE_IRPCancelAll( usbDeviceThisInstance->usbCDHandle,
                    controlEndpointTx );

            /* Deinitialize all function drivers.*/
            _USB_DEVICE_DeInitializeAllFunctionDrivers ( usbDeviceThisInstance );

            /* Disable all endpoints except for EP0.*/
            DRV_USB_DEVICE_EndpointDisableAll(usbDeviceThisInstance->usbCDHandle);

            /* Enable EP0 endpoint in RX direction */
            (void)usbDeviceThisInstance->driverInterface->deviceEndpointEnable
                  (usbDeviceThisInstance->usbCDHandle,
                    controlEndpointTx, USB_TRANSFER_TYPE_CONTROL, USB_DEVICE_EP0_BUFFER_SIZE);

            /* Code not shown due to space constraints */

            break;
    }
}
```

The USB Device Layer and the Function drivers will disable an endpoint when the Host sets a zero-device configuration or when the Host resets the device. The USBCD deviceEndpointDisable function disables an endpoint. When an endpoint is disabled, it does not accept requests for Host communication. Disabling an endpoint does not cancel any communication requests that that have been submitted on the endpoint. These requests must be canceled explicitly.

Device Endpoint Stall and Stall Clear

The USBCD client can call the deviceEndpointStall and deviceEndpointStallClear functions to stall and cleat the stall on an endpoint respectively. The USB Device Layer and function driver may stall endpoint to indicate error or to indicate a protocol state. The endpoint stall condition may be cleared in response to a USB Host Clear Feature request. Stalling or clearing the stall on an endpoint will cause all communication requests on the endpoint to be canceled. The function calls are thread safe and interrupt safe. The deviceEndpointIsStalled function is also available to check if an endpoint is in a stalled state. The following code example shows how the USB Device Layer calls these functions to stall and clear the stall on an endpoint.

```
/* The following code example shows how the USB Device Layer calls the driver
 * endpoint stall function (deviceEndpointStall) to stall an endpoint when the a
 * Host send a Set Feature request with feature selector set to endpoint halt.
 * The endpoint to be halted is identified in the setup packet and is identified
 * in this code example as usbEndpoint. Also shown is how the stall clear
 * (deviceEndpointStallClear) and stall status check (deviceEndpointIsStalled)
 * functions are called. */

/* The driverInterface member of the usbDeviceThisInstance structure is a
 * pointer to the USB Driver Common Interface. */

void _USB_DEVICE_ProcessStandardEndpointRequest
(
    USB_DEVICE_OBJ * usbDeviceThisInstance,
    uint8_t interfaceNumber,
    USB_SETUP_PACKET * setupPkt
)
{
    USB_ENDPOINT usbEndpoint;
```

```
    usbEndpoint = setupPkt->bEPID;

    if( setupPkt->bRequest == USB_REQUEST_GET_STATUS )
    {
        usbDeviceThisInstance->getStatusResponse.status = 0x00;
        usbDeviceThisInstance->getStatusResponse.endPointHalt
            =  usbDeviceThisInstance->driverInterface->deviceEndpointIsStalled
               (usbDeviceThisInstance->usbCDHandle, usbEndpoint );

        USB_DEVICE_ControlSend( (USB_DEVICE_HANDLE)usbDeviceThisInstance,
                (uint8_t *)&usbDeviceThisInstance->getStatusResponse, 2 );
    }
    else if( setupPkt->bRequest == USB_REQUEST_CLEAR_FEATURE )
    {
        if( setupPkt->wValue == USB_FEATURE_SELECTOR_ENDPOINT_HALT )
        {
            usbDeviceThisInstance->driverInterface->deviceEndpointStallClear
            (usbDeviceThisInstance->usbCDHandle, usbEndpoint );
            USB_DEVICE_ControlStatus((USB_DEVICE_HANDLE)usbDeviceThisInstance,
                                     USB_DEVICE_CONTROL_STATUS_OK );
        }
    }
    else if (setupPkt->bRequest == USB_REQUEST_SET_FEATURE )
    {
        if( setupPkt->wValue == USB_FEATURE_SELECTOR_ENDPOINT_HALT )
        {
            usbEndpoint = setupPkt->bEPID;
            usbDeviceThisInstance->driverInterface->deviceEndpointStall
                                 (usbDeviceThisInstance->usbCDHandle, usbEndpoint );
            USB_DEVICE_ControlStatus((USB_DEVICE_HANDLE)usbDeviceThisInstance,
                                     USB_DEVICE_CONTROL_STATUS_OK );
        }
    }

    /* Additional code is not shown due to space constraints */
}
```

Transferring Data to the Host

Provides information on transferring data to the Host.

Description

The USB Device Layer, the USBCD client, needs to transfer data to the Host in response to enumeration requests for general operation on the device. The USB uses a concept of Input Output Request Packet (IRP) to transfer data to and from the Host. IRPs are transported over endpoints which are enabled by calling the USBCD Endpoint Enable function.

A Device IRP is a USB_DEVICE_IRP type data structure. The IRP is created by the Device Layer and submitted to the USBCD for processing through the deviceIRPSubmit function. At the time of submitting the IRP, the endpoint over which the IRP must be transported is specified. The data request in the IRP is transported using the attributes of the endpoint. When an IRP is submitted to the USBCD, it is owned by the USBCD and cannot be modified by the Device Layer until the USBCD issues an IRP callback. The USBCD will issue the IRP callback when it has completed or terminated processing of the IRP.

An IRP does not have its own transfer type. It inherits the properties of the endpoint to which it is submitted. Hence an IRP becomes a control transfer IRP it was submitted to a control endpoint. An endpoint allows multiple IRPs to be queued. This allows the Device Layer to submit IRPs to an endpoint even while an IRP is being processed on the endpoint. The USBCD will process an IRP in the order that it was received. The following code example shows the USB_DEVICE_IRP data structure:

```
/* This code example shows the USB_DEVICE_IPR structure. The Device Layer
 * uses such a structure to transfer data through the driver. A structure of
 * this type is allocated by the Device Layer and the other function drivers and
 * passed to the deviceIRPSubmit function. */

typedef struct _USB_DEVICE_IRP
{
    /* Pointer to the data buffer */
    void * data;

    /* Size of the data buffer */
    unsigned int size;
```

```
    /* Status of the IRP */
    USB_DEVICE_IRP_STATUS status;

    /* IRP Callback. If this is NULL, then there is no callback generated */
    void (*callback)(struct _USB_DEVICE_IRP * irp);

    /* Request specific flags */
    USB_DEVICE_IRP_FLAG flags;

    /* User data */
    uintptr_t userData;

    /***********************************
     * The following members should not
     * be modified by the client
     ***********************************/
    uint32_t privateData[3];

} USB_DEVICE_IRP;
```

The `data` member of the USB_DEVICE_IRP structure points to a data buffer. This data buffer will contain the data that needs to be sent to the Host for the data stage of an IN transfer. For an OUT transfer, it will contain the data that was received from the Host. Any hardware specific cache coherency and address alignment requirements must be considered while allocating this data buffer. The Driver Client should not modify or examine the contents of the IRP after the IRP has been submitted and is being processed. It can be examined after the driver has released the IRP.

The `size` member of the USB_DEVICE_IRP structure specifies the size of the data buffer. The transfer will end when the device has sent or received size number of bytes. While sending data to the Host, the IRP size can exceed the size of the transaction (which is equal to the size of the endpoint). The USBCD in such a case will split up the transfer into transactions. This process does not require external intervention. The driver uses receive and transmit IRPs to process control transfers. When the driver receives a Setup packet, the IRP completion status would be USB_DEVICE_IRP_STATUS. The Driver Client should then use additional receive and transmit IRPs to complete the control transfer.

For interrupt and isochronous transfers, the size of transfer specified in the IRP cannot exceed the size of the transaction. If size is specified as 0, then the driver will send or expect a zero length packet. The `size` parameter of the IRP is updated by the driver when IRP processing is completed. This will contain the size of the completed transfer.

The `status` member of the IRP provides the completion status of the IRP and should be checked only when the IRP processing has completed. This is indicated by the driver calling the IRP callback function. The IRP status is a USB_DEVICE_IRP_STATUS type. The following code example shows the different possible values of the `status` member and example usage of IRPs to transfer data between the device and the Host.

```
/* The followoing code shows example usage of the device IRP. The submit status
 * of the IRP is available when IRP submit function returns. The completion
 * status of the IRP is available when the IRP has terminated and the IRP
 * callback function is invoked. The IRP callback
 * function shown in this example shows the possible complete status of the IRP.
 * The end application may or may not handle all the cases. Multiple IRPs can be
 * queued on an endpoint. */

void IRP_Callback(USB_DEVICE_IRP * irp)
{
    /* irp is pointing to the IRP for which the callback has occurred. In most
     * cases this function will execute in an interrupt context. The application
     * should not perform any hardware access or interrupt unsafe operations in
     * this function. */

    switch(irp->status)
    {
        case USB_DEVICE_IRP_STATUS_TERMINATED_BY_HOST:
            /* The IRP was aborted because the Host cleared the stall on the
             * endpoint */
            break;

        case USB_DEVICE_IRP_STATUS_ABORTED_ENDPOINT_HALT:
            /* IRP was aborted because the endpoint halted */
            break;


        case USB_DEVICE_IRP_STATUS_ABORTED:
            /* USB Device IRP was aborted by the function driver */
            break;

        case USB_DEVICE_IRP_STATUS_ERROR:
            /* An error occurred on the bus when the IRP was being processed */
```

```
            break;

        case USB_DEVICE_IRP_STATUS_COMPLETED:
            /* The IRP was completed */
            break;

        case USB_DEVICE_IRP_STATUS_COMPLETED_SHORT:
            /* The IRP was completed but the amount of data received was less
             * than the requested size */
            break;

        default:
            break;

    }
}

/* In the following example, the IRP is submitted to Endpoint 0x84. This is
 * interpreted as an IN direction endpoint (MSB of 0x84 is 1) and Endpoint 4.
 * The data contained in source will be sent to the USB Host. Assuming
 * the endpoint size is 64, the 130 bytes of data in this case will be sent to
 * the Host in three transaction of 64, 64 and 2 bytes. A transaction completes
 * when the Host polls (sends an IN token) the device.  The callback function
 * will then called indicating the completion status of the IRP. The application
 * should not modify the privateData field of the IRP. If the IRP was submitted
 * successfully, the buffer will be owned by the driver until the IRP callback
 * function has been called. Because the size of the transfer is not a multiple
 * of the endpoint size, the IRP flag must be set
 * USB_DEVICE_IRP_FLAG_DATA_COMPLETE. This directs the driver to not perform any
 * explicit signaling to the Host to indicate end of transfer. The last packet
 * in this case is a short packet and this signals the end of the transfer. */

USB_DEVICE_IRP irp;
USB_ERROR result;
uint8_t source[130];

irp.data = source;
irp.size = 130;
irp.called = IRP_Callback;
flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
userData = &someApplicationObject;

result = DRV_USBFS_DEVICE_IRPSubmit(driverHandle, 0x84, &irp);

switch(result)
{
    case USB_ERROR_PARAMETER_INVALID:
        /* This can happen if the driverHandle is invalid */
        break;

    case USB_ERROR_DEVICE_IRP_IN_USE:
        /* This can happen if the IRP is being resubmitted while it is still in
         * process (it was submitted before but processing has not completed */
        break;

    case USB_ERROR_DEVICE_ENDPOINT_INVALID;
        /* The endpoint to which this IRP is being submitted is not provisioned
         * in the system. This is controller by DRV_USBFS_ENDPOINTS_NUMBER
         * configuration parameter. */
        break;

    case USB_ERROR_ENDPOINT_NOT_CONFIGURED:
        /* The endpoint to which this IRP is being submitted is not enabled. It
         * must be enabled by calling the DRV_USBFS_DEVICE_EndpointEnable()
         * function. */
        break;

    case USB_ERROR_PARAMETER_INVALID:
        /* The USB_DEVICE_IRP_FLAG_DATA_PENDING flag was specified but the
```

```
            * transfer size is not a multiple of the endpoint size. If the IRP was
            * submitted to a receive endpoint, this error can occur if the size is
            * not a multiple of the endpoint size. */
            break;

        case USB_ERROR_OSAL_FUNCTION:
            /* An error occurred while trying to grab a mutex. This is applicable
             * when the driver is running with a RTOS. */
            break;

        case USB_ERROR_NONE:
            /* The IRP was submitted successfully. */
            break;

        default:
            break;
    }

    /* The following code example shows how an IRP is submitted to an OUT endpoint.
     * In this case data will be pointing to a buffer where the received data will
     * be stored. Note that the size of the IRP should be a multiple of the endpoint
     * size. The flags parameter is ignored in the data receive case. The IRP
     * terminates when the specified size of bytes has been received (the Host sends
     * OUT packets) or when a short packet has been received. */

    USB_DEVICE_IRP irp;
    USB_ERROR result;
    uint8_t destination[128];

    irp.data = destination;
    irp.size = 128;
    irp.called = IRP_Callback;
    userData = &someApplicationObject;

    result = DRV_USBFS_DEVICE_IRPSubmit(driverHandle, 0x04, &irp);
```

For IRPs submitted to an Interrupt or Isochronous endpoints, the driver will always send either the less than or equal to the maximum endpoint packet size worth of bytes in a transaction. The application could either submit an IRP per Interrupt/Isochronous polling interval or it could submit one IRP for multiple polling intervals.

The `flags` member of the USB_DEVICE_IRP structure specifies flags which affect the behavior of the IRP. The USB_DEVICE_IRP_FLAG enumeration specifies the available option. The USB_DEVICE_IRP_FLAG_DATA_COMPLETE causes the driver to add a Zero Length Packet (ZLP) to the data stage of the IN transfer when the transfer size is an exact multiple of the endpoint size. If the transfer size is not a multiple of the endpoint size, no ZLP will be sent. The USB_DEVICE_IRP_FLAG_PENDING flag will cause the driver to not send a ZLP in a case where the size of the IN transfer is an exact multiple of the endpoint size. The following code example demonstrates this.

```
    /* In the following code example, the IRP is submitted to an IN endpoint whose size
     * is 64. The transfer size is 128, which is an exact multiple of the endpoint
     * size. The flag is set to USB_DEVICE_IRP_FLAG_DATA_COMPLETE. The driver
     * will send two transactions of 64 bytes each and will then automatically send a
     * Zero Length Packet (ZLP), thus completing the transfer. The IRP callback will
     * be invoked when the ZLP transaction has completed. */

    USB_DEVICE_IRP irp;
    USB_ERROR result;
    uint8_t source[128];

    irp.data = source;
    irp.size = 128;
    irp.called = IRP_Callback;
    flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
    userData = &someApplicationObject;

    result = DRV_USBFS_DEVICE_IRPSubmit(driverHandle, 0x84, &irp);

    /* In the following code example, the IRP is submitted to an IN endpoint whose size
     * is 64. The transfer size is 128, which is an exact multiple of the endpoint
     * size. The flag is set to to USB_DEVICE_IRP_FLAG_DATA_PENDING. The driver will
     * send two transactions of 64 bytes each but will not send a ZLP. The USB Host
     * can then consider that there is more data pending in the transfer. The IRP
     * callback will be invoked when the two transactions have completed. */
```

```
USB_DEVICE_IRP irp;
USB_ERROR result;
uint8_t source[128];

irp.data = source;
irp.size = 128;
irp.called = IRP_Callback;
flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
userData = &someApplicationObject;

result = DRV_USBFS_DEVICE_IRPSubmit(driverHandle, 0x84, &irp);
```

The `callback` member of the USB_DEVICE_IRP structure points to a function which the driver calls when the IRP processing is completed. The Driver Client must implement this function and assign the pointer to this function to the `callback` member of the IRP. Every IRP can have its own callback function or one common callback function could be used. The callback function will execute in an interrupt context. The Driver Client should not execute interrupt unsafe, blocking or computationally intensive operations in the callback function. The client can call deviceIRPSubmit function in the IRP callback function to submit another IRP or resubmit the same IRP. The client can check the status and size of the IRP in the callback function.

The `userData` member of the USB_DEVICE_IRP structure can be used by the client to associate a client specific context with the Host. This context can then be used by the client, in the IRP callback function to identify the context in which the IRP was submitted. This member is particularly useful if the client wants to implement one callback function for all IRPs.

The `privateData` member of the IRP is used by the driver and should not be accessed or manipulated by the Driver Client.

PIC32MX USB Driver

Provides information on the USB Driver specific to PIC32MX devices.

Description

The PIC32MX USB Driver in MPLAB Harmony provides API functions that allow the MPLAB Harmony USB Host and Device Stack to access the USB while operating on a PIC32MX microcontroller. The driver implements the USB Driver Common Interface required by the USB Host and Device Stack. It abstracts the USB module operational details from the Host and Device Stack and provides the stacks with a modular access mechanism to the USB. The PIC32MX USB Driver features the following:

- USB 2.0 Full Speed operation in Peripheral mode
- USB 2.0 Full Speed and Low Speed USB Peripheral Support in Host mode
- Designed for Dual Role Operation
- Capable of operating multiple USB modules
- Features non-blocking function and is interoperable with other MPLAB Harmony modules
- Features thread safe functions when operating within an RTOS
- Capable of operating in Polled and Interrupt modes
- Implements the USB Driver Common Interface required by the MPLAB Harmony USB Host and Device Stack
- Completely configurable through MPLAB Harmony Configurator (MHC) tool
- Implements feature separation (Host and Device mode functions are implemented across different files)

Note: This help section only discusses features that are unique to the PIC32MX USB Driver and are not a part of the USB Driver Common Interface. The driver functions that implement the USB Driver Common Interface are described in the Common Interface Help section.

While the PIC32MX USB module supports USB "On-The-Go" (OTG), this release of the PIC32MX Driver does not implement USB OTG protocol support.

This help section only provides relevant information about the operation of the USB. The reader is encouraged to refer to the USB 2.0 Specification available at www.usb.org for a detailed explanation of USB protocol.

Using the Library

This topic describes the basic architecture of the PIC32MX USB Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_usbfs.h`

The interface to the PIC32MX USB Driver library is defined in the `drv_usbfs.h` header file.

Please refer to the What is MPLAB Harmony? section for how the Driver interacts with the framework.

Library Overview

Provides an overview of the library.

Description

The PIC32MX USB Driver will typically be used by a USB Host and/or Device Stack. The USB Host and Device Stack operate as driver client applications. The driver is initialized as part of the MPLAB Harmony System Initialization. The driver initialization data structure specifies the operation mode (Host, Device, or Dual Role) of the driver. The driver features task routines to be called in the MPLAB Harmony application tasks function (SYS_Tasks function) and the USB Module Interrupt Service Routine (ISR).

The Host and the Device Stack can open the driver only when initialization has completed. It will continue to return an invalid driver handle while the initialization is in progress. Once opened, the Device Mode function can be called if the driver is operating in Device mode. The Host Mode function can be called if the driver is operating in Host mode. In Dual Role operation mode, the driver supports Host and Device operation in the same application. Even then, the driver will either operate as a USB Host or Device. OTG operation is not supported.

The PIC32MX USB Driver features RTOS thread-safe functions. This allows the driver client application to safely call driver functions across different RTOS threads. Not all of the driver functions are interrupt-safe.

In addition to the USB Driver, which implements the USB Driver Common Interface, the PIC32MX USB Driver implements functions which are required for its operation in the MPLAB Harmony framework. The following table lists the different categories of functions in the PIC32MX USB Driver.

Library Interface Section	Description
System Function	These functions are accessed by the MPLAB Harmony System module. They allow the driver to be initialized, deinitialized and maintained. These functions are implemented in the `drv_usbfs.c` source file.
Client Core Functions	These functions allow the USB Host and Device Stack to open, close and perform other general driver operations. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbfs.c` source file.
Device Mode Operation Functions	These functions allow the USB Device Stack to perform USB Device mode specific driver operations. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbfs_device.c` source file
Host Mode Operation Functions	These functions allow the USB Host Stack to perform USB Host mode specific driver operations. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbfs_host.c` source file.
Root Hub Functions	These functions allow the USB Host Stack to access the driver Root hub operation. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbfs_host.c` source file.

Abstraction Model

Provides information on the abstraction model for the library.

Description

The PIC32MX USB Driver implements the abstraction model defined by the USB Driver Common interface. This interface abstracts USB module specific details and provides a module independent interface to the driver client applications.

While operating in Device mode, the driver expects the client application (the USB Device Stack) to enable endpoints and then submit I/O request packet (IRP) requests to the enabled endpoints. Multiple IRPs can be queued on an endpoint. The driver calls the IRP callback function when the IRP is processed. The driver allows the client application to also attach and detach the device on the bus. It generates events which indicate USB states.

While operating in Host mode, the driver expects the client application (the USB Host Stack) to open pipes to endpoints on the connected device. The client application can then submit IRPs to the pipes. Multiple IRPs can be queued on a pipe. The driver calls the IRP callback function when the IRP is processed. The driver will call application defined functions to enumerate and denumerate a device. These functions are called when the driver detect device attach and detach respectively. The driver also exports root hub functions to the client application. This allows the client application to treat the driver as a single port hub

Please refer to the PIC32 USB Driver Common Interface help section for more details on the driver abstraction model.

How the Library Works

Provides information on how the library works.

Description

This section only explains aspects of driver operation which are unique to the PIC32MX USB Driver. Major driver operations are described in the PIC32 USB Driver Common Interface help section.

Driver Initialization

Note: While generating a MPLAB Harmony USB project with MHC, the initialization code for the driver is generated automatically based on selections made in the USB Host stack or Device Stack Configuration trees.

The PIC32MX USB Driver must be initialized so that a client application can open. The client application will not be able to open the driver if the initialization is in progress or has failed. The driver is initialized by calling the DRV_USBFS_Initialize function. This function is called from the SYS_Initialize function in the MPLAB Harmony application project and accepts two input parameters. The index parameter defines the instance of the USB Driver to be initialized. This becomes significant when the PIC32MX microcontroller has more than one USB module. The init parameter is a driver specific data structure of the type DRV_USBFS_INIT. This structure is shown in the following code example.

```
/* This code snippet show the PIC32MX USB Driver Initialization data structure.
 * A structure of this type must be provided to the DRV_USBFS_Initialize()
 * function. */

typedef struct
{
    /* System Module Initialization */
    SYS_MODULE_INIT moduleInit;

    /* Identifies the USB peripheral to be used. This should be the USB PLIB
       module instance identifier. */
    uint8_t usbID;

    /* This should be set to true if the USB module must stop operation in IDLE
       mode */
    bool stopInIdle;

    /* This should be set to true if the USB module must suspend when the CPU
       enters sleep mode. */
    bool suspendInSleep;

    /* Specify the interrupt source for the USB module. This should be Interrupt
       PLIB Interrupt source identifier for the USB module instance specified in
       usbID. */
    INT_SOURCE interruptSource;

    /* Specify the operational speed of the USB module. This should always be
       set to USB_SPEED_FULL. The use of this parameter is deprecated. */
    USB_SPEED operationSpeed;

    /* Specify the operation mode of the USB module. This defines if the USB
     * module will support Device, Host or Dual Role operation */
    DRV_USBFS_OPMODES operationMode;

    /* A pointer to the endpoint descriptor table. This should be aligned at 512
       byte address boundary. The size of the table is equal to the
       DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE times the number of endpoints needed
       in the application. */
    void * endpointTable;

    /* Root hub available current in mA. This specifies the amount of current
       that root hub can provide to the attached device. This should be
       specified in mA. This is required when the driver is required to operate
       in host mode. */
    uint32_t rootHubAvailableCurrent;

    /* When operating in Host mode, the application can specify a Root Hub port
       enable function. This parameter should point to Root Hub port enable
       function. If this parameter is NULL, it implies that the Port is always
       enabled. */
    DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE portPowerEnable;

    /* When operating in Host mode, the application can specify a Root Port
       Indication. This parameter should point to the Root Port Indication
       function. If this parameter is NULL, it implies that Root Port Indication
       is not supported. */
    DRV_USBFS_ROOT_HUB_PORT_INDICATION portIndication;
```

```
    /* When operating is Host mode, the application can specify a Root Port
       Overcurrent detection. This parameter should point to the Root Port
       Indication function. If this parameter is NULL, it implies that
       Overcurrent detection is not supported. */
    DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT portOverCurrentDetect;

} DRV_USBFS_INIT;
```

The operationMode parameter defines by the driver operation mode. parameter in the initialization data structure. This can be set DRV_USBFS_OPMODE_DEVICE, DRV_USBFS_OPMODE_HOST or DRV_USBFS_OPMODE_DUAL_ROLE for device, host and dual role operation respectively.

The `endpointTable` parameter must point to a byte array. The size of the array depends on the maximum number of device endpoints that application needs. A direction of an endpoint counts as one endpoint. Each endpoint requires 32 bytes. Therefore, if the USB Device requires 3 endpoints (Endpoint 0 + Endpoint 1 IN + Endpoint 2 OUT), the size of the array will 96 bytes (32 * 3). The byte array start address must be located on a 512 byte boundary. When operating in host mode, the driver will use only one endpoint and size of the endpoint table array should be set to 32.

The `rootHubAvailableCurrent` parameter should be set to the maximum current that VBUS power supply can provide on the bus. The driver does not use this information directly. It provides this data to the client application while operating in host mode.

The `portPowerEnable` parameter must point to a Port Power Enable function. The driver, while operating in host mode, will call this function to enable the VBUS switch. This function should activate the VBUS switch if the driver calls this function with the enable parameter set to true. It should deactivate the switch if the driver calls this function with the enable parameter set to false. This parameter should be set to NULL if such a switch (of the switch control) is not available in the application.

The `portIndication` parameter must point to a Port Indication function. The driver, while operating in host mode, will call this function to indicate the current state of the port. The driver will call this function with LED color status as defined in the Chapter 11 of the USB 2.0 Specification. This parameter should be set to NULL if such a LED indication is not available in the application.

The `portOverCurrentDetect` parameter must point to a Port Overcurrent Detect function. The driver, while operating in Host mode, will call this function periodically to check if the attached device is overdrawing current. If the function should return true if such a condition exists. This parameter should be set to NULL if such detection is not available in the application.

The following code example shows initialization of the driver for device mode operation.

```
/* This code shows an example of DRV_USBFS_INIT data structure for
 * device mode operation. Here the driver is initialized to work with USB1 USB
 * module. Note how the endPointTable is defined. It should be aligned on a 512
 * byte boundary. */

DRV_USBFS_INIT init;
SYS_MODULE_OBJ usbDriverObj;

uint8_t __attribute__((aligned(512))) endPointTable[DRV_USBFS_ENDPOINTS_NUMBER * 32];

const DRV_USBFS_INIT drvUSBInit =
{
    /* Assign the endpoint table */
    .endpointTable = endPointTable,

    /* Interrupt Source for USB module */
    .interruptSource = INT_SOURCE_USB_1,

    /* System module initialization. */
    .moduleInit = {SYS_MODULE_POWER_RUN_FULL},

    /* Configure the driver for device mode operation. */
    .operationMode = DRV_USBFS_OPMODE_DEVICE,

    /* Configure the driver to operate at full speed. */
    .operationSpeed = USB_SPEED_FULL,

    /* Stop in idle */
    .stopInIdle = false,

    /* Suspend in sleep */
    .suspendInSleep = false,

    /* Identifies peripheral (PLIB-level) ID */
    .usbID = USB_ID_1
};

void SYS_Initialize(void)
{
```

```
    /* Initialize the USB Driver. Note how the init parameter is typecasted to
     * SYS_MODULE_INIT type. The SYS_MODULE_OBJ returned by this function call
     * is passed to the driver tasks routine. DRV_USBFS_INDEX_0 is helper
     * constant defined in drv_usbfs.h */

    usbDriverObj = DRV_USBFS_Initialize(DRV_USBFS_INDEX_0, (SYS_MODULE_INIT *)(drvUSBInit));
}

void SYS_Tasks(void)
{
    /* The polled state of the USB driver is updated by calling the
     * DRV_USBFS_Tasks function in the SYS_Tasks() function. The
     * DRV_USBFS_Tasks() takes the driver module object returned by the
     * DRV_USBFS_Initialize funciton as a parameter. */

    DRV_USBFS_Tasks(usbDriverObj);
}

void __ISR(_USB_1_VECTOR, ipl4AUTO) _IntHandlerUSBInstance0(void)
{
    /* The DRV_USBFS_Tasks_ISR function update the interrupt state of the USB
     * Driver. If the driver is configured for polling mode, this function need
     * not be invoked or included in the project. */

    DRV_USBFS_Tasks_ISR(sysObj.drvUSBObject);
}
```

The following code example shows initialization of the driver for host mode operation.

```
/* This code shows an example of the USBFS driver can be configured for
 * host mode operation. For host mode operation, only one endpoint is needed and
 * hence the size of the endpoint table is 32 bytes (for one endpoint). In this
 * example, the BSP_USBVBUSSwitchOverCurrentDetect function checks for over
 * current condition and the BSP_USBVBUSPowerEnable function enables the VBUS
 * power. The port indication function is not implemented and hence the
 * portIndication member of the initialization data structure is set to NULL. */

/* The implementation of the port over current detect, indication and the VBUS
 * power supply functions is discussed later in this help section. */

uint8_t __attribute__((aligned(512))) endpointTable[32];

DRV_USBFS_INIT drvUSBFSInit =
{
    /* Pointer to the endpoint table */
    .endpointTable = endpointTable,

    /* Interrupt Source for the USB module */
    .interruptSource = INT_SOURCE_USB_1,

    /* This should always be set to SYS_MODULE_POWER_RUN_FULL. */
    .moduleInit = {SYS_MODULE_POWER_RUN_FULL},

    /* Configure for host mode operation. */
    .operationMode = DRV_USBFS_OPMODE_HOST,

    /* The driver should run at full speed. */
    .operationSpeed = USB_SPEED_FULL,

    /* Port indication function is not implemented. */
    .portIndication = NULL,

    /* This is the over current detect function. */
    .portOverCurrentDetect = BSP_USBVBUSSwitchOverCurrentDetect,

    /* This is the VBUS Power enable function */
    .portPowerEnable = BSP_USBVBUSPowerEnable,

    /* Here we state that the VBUS power supply can provide at most 500 mA of
     * current */
    .rootHubAvailableCurrent = 500,
```

```
    /* Module will operate in IDLE. */
    .stopInIdle = false,

    /* Module will not suspend automatically in sleep */
    .suspendInSleep = false,

    /* USB Module ID is 1 */
    .usbID = USB_ID_1
};

void SYS_Initialize(void)
{
    /* Initialize the USB Driver. Note how the init parameter is typecasted to
     * SYS_MODULE_INIT type. The SYS_MODULE_OBJ returned by this function call
     * is passed to the driver tasks routine. DRV_USBFS_INDEX_0 is helper
     * constant defined in drv_usbfs.h */

    usbDriverObj = DRV_USBFS_Initialize(DRV_USBFS_INDEX_0,
                                        (SYS_MODULE_INIT *)(drvUSBInit));
}

void SYS_Tasks(void)
{
    /* The polled state of the USB driver is updated by calling the
     * DRV_USBFS_Tasks function in the SYS_Tasks() function. The
     * DRV_USBFS_Tasks() takes the driver module object returned by the
     * DRV_USBFS_Initialize funciton as a parameter. */

    DRV_USBFS_Tasks(usbDriverObj);
}

void __ISR(_USB_1_VECTOR, ipl4AUTO) _IntHandlerUSBInstance0(void)
{
    /* The DRV_USBFS_Tasks_ISR function update the interrupt state of the USB
     * Driver. If the driver is configured for polling mode, this function need
     * not be invoked or included in the project. */

    DRV_USBFS_Tasks_ISR(sysObj.drvUSBObject);
}
```

The PIC32MX USB Driver requires definition of configuration constants to be available in the `system_config.h` file of the MPLAB Harmony Application Project Configuration. Refer to the Configuring the Library section for details.

Multi-client Operation

The PIC32MX USB Driver supports multi-client operation. In that, it can be opened by two application clients. This is required where Dual Operation is desired. The following should be noted when using multi-client operation:

- The driver should be initialized for Dual Role Operation mode.
- The DRV_USBFS_Open function can be called at the most twice in the application. The driver supports a maximum of two clients.
- A client can access either the host or device functionality of the driver. It cannot do both.
- It is possible for the two clients to operate in two different threads while operating with an RTOS.

Note: The typical the application clients for PIC32MX USB Driver would be the MPLAB Harmony USB Host and Device Stack. The complexity of operating the driver in Dual Role mode is handled by the stack operation. The MHC will configure the driver for Dual Role operation when such operation is selected in USB Stack configuration tree.

USB Driver Common Interface

The PIC32MX USB Driver exports its implementation of the USB Driver Common Interface to the Host and Device Layer via the DRV_USBFS_HOST_INTERFACE and DRV_USBFS_DEVICE_INTERFACE structures. The DRV_USBFS_HOST_INTERFACE structure is defined in the `drv_usbfs_host.c` file. The following code example shows this structure.

```
/*****************************************************
 * This structure is a set of pointer to the USBFS driver
 * functions. It is provided to the host and device layer
 * as the interface to the driver.
 * ****************************************************/

DRV_USB_HOST_INTERFACE gDrvUSBFSHostInterface =
{
```

```
    .open = DRV_USBFS_Open,
    .close = DRV_USBFS_Close,
    .eventHandlerSet = DRV_USBFS_ClientEventCallBackSet,
    .hostIRPSubmit = DRV_USBFS_HOST_IRPSubmit,
    .hostIRPCancel = DRV_USBFS_HOST_IRPCancel,
    .hostPipeSetup = DRV_USBFS_HOST_PipeSetup,
    .hostPipeClose = DRV_USBFS_HOST_PipeClose,
    .hostEventsDisable = DRV_USBFS_HOST_EventsDisable,
    .hostEventsEnable = DRV_USBFS_HOST_EventsEnable,
    .rootHubInterface.rootHubPortInterface.hubPortReset = DRV_USBFS_HOST_ROOT_HUB_PortReset,
    .rootHubInterface.rootHubPortInterface.hubPortSpeedGet = DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet,
    .rootHubInterface.rootHubPortInterface.hubPortResetIsComplete =
DRV_USBFS_HOST_ROOT_HUB_PortResetIsComplete,
    .rootHubInterface.rootHubPortInterface.hubPortSuspend = DRV_USBFS_HOST_ROOT_HUB_PortSuspend,
    .rootHubInterface.rootHubPortInterface.hubPortResume = DRV_USBFS_HOST_ROOT_HUB_PortResume,
    .rootHubInterface.rootHubMaxCurrentGet = DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet,
    .rootHubInterface.rootHubPortNumbersGet = DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet,
    .rootHubInterface.rootHubSpeedGet = DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet,
    .rootHubInterface.rootHubInitialize = DRV_USBFS_HOST_ROOT_HUB_Initialize,
    .rootHubInterface.rootHubOperationEnable = DRV_USBFS_HOST_ROOT_HUB_OperationEnable,
    .rootHubInterface.rootHubOperationIsEnabled = DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled,
};
```

The DRV_USBFS_DEVICE_INTERFACE structure is defined in the `drv_usbfs_device.c` file. The following code example shows this structure. The MPLAB Harmony USB Host and Device stack perform driver independent access through the function pointers contained in these structures.

```
/*************************************************
 * This structure is a pointer to a set of USB Driver
 * Device mode functions. This set is exported to the
 * device layer when the device layer must use the
 * PIC32MX USB Controller.
 *************************************************/

DRV_USB_DEVICE_INTERFACE gDrvUSBFSDeviceInterface =
{
    .open = DRV_USBFS_Open,
    .close = DRV_USBFS_Close,
    .eventHandlerSet = DRV_USBFS_ClientEventCallBackSet,
    .deviceAddressSet = DRV_USBFS_DEVICE_AddressSet,
    .deviceCurrentSpeedGet = DRV_USBFS_DEVICE_CurrentSpeedGet,
    .deviceSOFNumberGet = DRV_USBFS_DEVICE_SOFNumberGet,
    .deviceAttach = DRV_USBFS_DEVICE_Attach,
    .deviceDetach = DRV_USBFS_DEVICE_Detach,
    .deviceEndpointEnable = DRV_USBFS_DEVICE_EndpointEnable,
    .deviceEndpointDisable = DRV_USBFS_DEVICE_EndpointDisable,
    .deviceEndpointStall = DRV_USBFS_DEVICE_EndpointStall,
    .deviceEndpointStallClear = DRV_USBFS_DEVICE_EndpointStallClear,
    .deviceEndpointIsEnabled = DRV_USBFS_DEVICE_EndpointIsEnabled,
    .deviceEndpointIsStalled = DRV_USBFS_DEVICE_EndpointIsStalled,
    .deviceIRPSubmit = DRV_USBFS_DEVICE_IRPSubmit,
    .deviceIRPCancel = DRV_USBFS_DEVICE_IRPCancel,
    .deviceIRPCancelAll = DRV_USBFS_DEVICE_IRPCancelAll,
    .deviceRemoteWakeupStop = DRV_USBFS_DEVICE_RemoteWakeupStop,
    .deviceRemoteWakeupStart = DRV_USBFS_DEVICE_RemoteWakeupStart,
    .deviceTestModeEnter = NULL


};
```

Operation with RTOS

The PIC32MX USB Driver is designed to operate with a RTOS. The driver implementation uses the MPLAB Harmony Operating System Abstraction Layer (OSAL). This allows the driver to function with entire range of RTOSes supported in MPLAB Harmony. The following points must be considered while using the driver with an RTOS.

- The driver can be opened from different threads
- In Device mode, an enabled endpoint should only be accessed from one thread. For example, if an application requires two endpoints, Endpoint 2 and Endpoint 3, the application could contain two threads, one accessing Endpoint 2 and another accessing Endpoint 3. The thread accessing Endpoint 2 cannot access Endpoint 3.
- While operating in Host mode, endpoint pipes can be opened from different threads. A pipe handle to an open pipe cannot be shared across threads.

Host Mode Attach Detach Operation

When the PIC32MX USB Driver operating in Host mode detects a device attach or detach, it implements debouncing before signaling attach detach signal to the USB Host Stack. When the device is attached, the driver waits for DRV_USBFS_HOST_ATTACH_DEBOUNCE_DURATION milliseconds to allow for the mechanical chatter, which occurs when the device is inserted into the host receptacle, to settle. If the device is still attached after the DRV_USBFS_HOST_ATTACH_DEBOUNCE_DURATION expires, the driver calls the USB_HOST_DeviceEnumerate function to let the host stack enumerate the device. It also starts checking for Device Detach.

When the device is detached, the driver waits for DRV_USBFS_POST_DETACH_DELAY milliseconds to allow for the detach operation to settle. If the device is indeed detached after the DRV_USBFS_POST_DETACH_DELAY delay expires, the driver calls USB_HOST_DeviceDenumerate function to let the USB Host stack denumerate the device. It then starts checking for device attach.

Root Hub Operation

The PIC32MX USB Driver implements a Root Hub Driver Interface. This allows the driver to emulate a hub. The USB Host Stack enumerates the Root Hub as a device. The Host Stack then does not differentiate between an external hub and the root hub. While emulating a hub, the PIC32MX USB Driver Root Hub appears as a single port hub.

As a part of the root hub interface, the PIC32MX USB Driver requires the application to supply functions for hub features that it does not implement. These features are:

- Port Overcurrent Detect
- VBUS Switch Control
- Port Indication

A pointer to these functions (if implemented) must be supplied through the driver initialization data (of the type DRV_USBFS_INIT) structure at the time of driver initialization. The application has the option of not implementing these functions. In such a case, the function pointers for the unimplemented function, in the initialization data structure should be set to NULL.

The root hub driver must also be able to communicate the maximum current capability of its port to the USB Host Layer. The PIC32MX USB Controller does not contain built-in (hardware implemented) functionality for controlling the root hub port current. To facilitate this request, the driver will report the current capability that was specified in the `rootHubAvailableCurrent` parameter of the driver initialization data structure. The application must set this parameter to report the current supply capability of the VBUS power supply. The USB Host Layer uses this value to manage the bus current budget. If a connected device reports a configuration that requires more current than what the VBUS power supply can provide, the host will not set the configuration.

Port Overcurrent Detect

The Root Hub operation in PIC32MX USB Driver will periodically call a Port Overcurrent Detect function to detect if an overcurrent condition is active on the port. The application must supply this function if port overcurrent detection is needed. The PIC32MX USB Controller does not contain built-in (hardware implemented) functionality for checking overcurrent condition. The overcurrent condition on the port can occur in a case where the attached device has malfunctioned or when the USB VBUS line has short circuited to ground.

The signature of the function and an example implementation is shown in the following code example. The function must return (and must continue to return) true if an overcurrent condition exists on the port.

```
/* This code shows an example implementation of the
 * portOverCurrentDetect function. The PIC32MX USB Driver will call this
 * function periodically to check if an over current condition exists on the
 * port. In this example, we assume that the over current detect pin from an
 * external circuit in the system, is connected to  port RD0 and the pin logic
 * is active high. The function must return true if an over current condition is
 * present on this pin */


bool BSP_USBVBUSSwitchOverCurrentDetect(uint8_t port)
{
    if(PLIB_PORTS_PinGet(PORTS_ID_0, PORT_CHANNEL_D, 0) == 1)
    {
        return(true);
    }
    else
    {
        return(false);
    }
}
```

VBUS Switch Control

The PIC32MX USB Driver Root Hub operation will attempt to control the VBUS power supply to the port. Because the PIC32MX USB Controller does not contain built-in (hardware implemented) functionality for checking controlling VBUS, such a control function must be supplied by the application. The root hub operation will access this function when the PIC32MX USB Driver will call the portPowerEnable function as a part of the Bus Enable sequence.

The following code shows an example of how this function can be implemented.

```
/* This code shows an example implementation of the VBUS Power Enable
```

```
 * function. The PIC32MX USB Driver will call this function as a part of bus
 * enable function. In this example, it is assumed that system contains an
 * external VBUS power switch and this is control by port RB5.
 */

void BSP_USBVBUSPowerEnable(uint8_t port, bool enable)
{
    if(enable)
    {
        PLIB_PORTS_PinSet(PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_5);
    }
    else
    {
        PLIB_PORTS_PinClear(PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_5);
    }
}
```

Port Indication function

The Root Hub Operation in the PIC32MX USB Driver allows display of Port LED status. If the application requires this indication, it must implement a function which the Root Hub operation would call when a change in the Root Hub port has occurred. The port indication operation is specified in Section 11.5.3 of the USB 2.0 Specification.

```
/* This code shows an example implementation of the port indication
 * function. The PIC32MX USB Driver calls this function when it wants to indicate
 * port status. It is assumed that three function to switch off, blink and
 * switch on an LED are available. It is further assumed that these function
 * accept the color of the LED to operated on. */

void BSP_RootHubPortIndication
(
    uint8_t port,
    USB_HUB_PORT_INDICATOR_COLOR color,
    USB_HUB_PORT_INDICATOR_STATE state
)
{
    /* The color parameter indicates the color of the LED to be affected. The
     * color will be either USB_HUB_PORT_INDICATOR_COLOR_GREEN or
     * USB_HUB_PORT_INDICATOR_COLOR_AMBER. */

    switch (state)
    {
        case USB_HUB_PORT_INDICATOR_STATE_OFF:
            BSP_SwitchLEDOff(color);
            break;
        case USB_HUB_PORT_INDICATOR_STATE_BLINKING:
            BSP_LEDBlink(color);
            break;
        case USB_HUB_PORT_INDICATOR_STATE_ON:
            BSP_SwitchLEDOn(color);
            break;
        default:
            break;
    }
}
```

Configuring the Library

Provides information on the configuring the library.

Macros

	Name	Description
	DRV_USBFS_DEVICE_SUPPORT	Determines if the USB Device Functionality should be enabled.
	DRV_USBFS_ENDPOINTS_NUMBER	Configures the number of endpoints to be provisioned in the driver.
	DRV_USBFS_HOST_ATTACH_DEBOUNCE_DURATION	Configures the time duration (in milliseconds) that the driver will wait to re-confirm a device attach.
	DRV_USBFS_HOST_NAK_LIMIT	Configures the NAK Limit for Host Mode Control Transfers.

	DRV_USBFS_HOST_PIPES_NUMBER	Configures the maximum number of pipes that are can be opened when the driver is operating in Host mode.
	DRV_USBFS_HOST_RESET_DURATION	Configures the time duration (in milliseconds) of the Reset Signal.
	DRV_USBFS_HOST_SUPPORT	Determines if the USB Host Functionality should be enabled.
	DRV_USBFS_INSTANCES_NUMBER	Specifies the number of driver instances to be enabled in the application.
	DRV_USBFS_INTERRUPT_MODE	Configures the driver for interrupt or polling mode operation.

Description

The PIC32MX USB Driver requires the specification of compile-time configuration macros. These macros define resource usage, feature availability, and dynamic behavior of the driver. These configuration macros should be defined in the `system_config.h` file.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_USBFS_DEVICE_SUPPORT Macro

Determines if the USB Device Functionality should be enabled.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_DEVICE_SUPPORT true
```

Description

USB Full Speed Driver Device Mode Support.

This constant should be set to true if USB device support is required in the application. It should be set to false if device support is not required.

Remarks

This constant should always be defined.

DRV_USBFS_ENDPOINTS_NUMBER Macro

Configures the number of endpoints to be provisioned in the driver.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_ENDPOINTS_NUMBER 3
```

Description

USB Full Speed Driver Endpoint Numbers.

This constant configures the number of endpoints that the driver needs to manage. When DRV_USBFS_DEVICE_SUPPORT is enabled, this constant should be set to the total number of endpoints to be enabled in the device. When enabled, a endpoint can be used for communication. Using any direction of an endpoint will require that entire endpoint to be enabled.

Consider the case of a composite USB Device that containing a CDC and MSD function. The CDC function will require 1 Bulk endpoint (OUT and IN directions) and 1 Interrupt endpoint (IN direction). The MSD function will require 1 Bulk endpoint (IN and OUT directions). This design can be implemented by using 4 endpoints. Endpoint 0 is used for the mandatory control interface. Endpoint 1 is used for CDC Bulk interface. Endpoint 2 is used for CDC interrupt interface and endpoint 3 is used for MSD Bulk Interface. The constant should then be set to 4.

For Host mode operation, this constant should be set to 1. Setting this to greater than 1 will result in unused data memory allocation.

Remarks

This constant should always be defined.

DRV_USBFS_HOST_ATTACH_DEBOUNCE_DURATION Macro

Configures the time duration (in milliseconds) that the driver will wait to re-confirm a device attach.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_HOST_ATTACH_DEBOUNCE_DURATION 500
```

Description

USB Full Speed Driver Host Mode Attach Debounce Duration.

This constant configures the time duration (in milliseconds) that driver will wait to re-confirm a device attach. When the driver first detects device attach, it start, it will start a timer for the duration specified by the constant. When the timer expires, the driver will check if the device is still attached. If so, the driver will then signal attach to the host stack. The duration allows for device attach to become electro-mechanically stable.

Remarks

This constant should always be defined when DRV_USBFS_HOST_SUPPORT is set to true.

DRV_USBFS_HOST_NAK_LIMIT Macro

Configures the NAK Limit for Host Mode Control Transfers.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_HOST_NAK_LIMIT 2000
```

Description

USB Full Speed Driver Host Mode Control Transfers NAK Limit.

This constant configures the number of NAKs that the driver can accept from the device in the data stage of a control transfer before aborting the control transfer with a USB_HOST_IRP_STATUS_ERROR_NAK_TIMEOUT. Setting this constant to 0 will disable NAK limit checking. This constant should be adjusted to enable USB host compatibility with USB Devices which require more time to process control transfers.

Remarks

This constant should always be defined when DRV_USBFS_HOST_SUPPORT is set to true.

DRV_USBFS_HOST_PIPES_NUMBER Macro

Configures the maximum number of pipes that are can be opened when the driver is operating in Host mode.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_HOST_PIPES_NUMBER 10
```

Description

USB Full Speed Driver Host Mode Pipes Number.

This constant configures the maximum number of pipes that can be opened when the driver is operating in Host mode. Calling the DRV_USBFS_HOST_PipeSetup function will cause a pipe to be opened. Calling this function when DRV_USBFS_HOST_PIPES_NUMBER number of pipes have already been opened will cause the function to return an Invalid Pipe Handle. This constant should be configured to account for the maximum number of devices and the device types to be supported by the host application.

For example if the USB Host application must support 2 USB Mass Storage devices and 1 CDC device, it must set this constant 9 (4 bulk pipes for 2 Mass Storage devices + 2 bulk pipes and 1 interrupt pipe for 1 CDC device and 2 control pipes for 2 devices). Allocating pipes consumes data memory.

Remarks

This constant should always be defined when DRV_USBFS_HOST_SUPPORT is set to true.

DRV_USBFS_HOST_RESET_DURATION Macro

Configures the time duration (in milliseconds) of the Reset Signal.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_HOST_RESET_DURATION 100
```

Description

USB Full Speed Driver Host Mode Reset Duration.

This constant configures the duration of the reset signal. The driver generates reset signal when the USB Host stack requests for root hub port reset. The driver will generate the reset signal for the duration specified by this constant and will then stop generating the reset signal.

Remarks

This constant should always be defined when DRV_USBFS_HOST_SUPPORT is set to true.

DRV_USBFS_HOST_SUPPORT Macro

Determines if the USB Host Functionality should be enabled.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_HOST_SUPPORT false
```

Description

USB Full Speed Driver Host Mode Support.

This constant should be set to true if USB Host mode support is required in the application. It should be set to false if host support is not required.

Remarks

This constant should always be defined.

DRV_USBFS_INSTANCES_NUMBER Macro

Specifies the number of driver instances to be enabled in the application.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_INSTANCES_NUMBER 1
```

Description

USB Full Speed Driver Instances Number.

This constant defines the number of driver instances to be enabled in the application. This will be typically be the number of USB controllers to be used in the application. On PIC32MX microcontrollers that have one USB controller, this value will always be 1. On PIC32MX microcontrollers which have 2 USB controllers, this value could 1 or 2, depending on whether 1 or 2 USB segments are required. To conserve data memory, this constant should be set to exactly the number of USB controller that are required in the system.

Remarks

This constant should always be defined.

DRV_USBFS_INTERRUPT_MODE Macro

Configures the driver for interrupt or polling mode operation.

File

drv_usbfs_config_template.h

C

```
#define DRV_USBFS_INTERRUPT_MODE true
```

Description

USB Full Speed Driver Interrupt Mode.

This constant configures the driver for interrupt or polling operation. If this flag is set to true, the driver will operate in interrupt mode. If the flag is

set to false, the driver will operate in polled mode. In polled, the driver interrupt state machine gets updated in the SYS_Tasks(). If the driver is configured interrupt mode, the driver interrupt state machine gets updated in the driver interrupt service routine. It is always recommended for the driver to operate in interrupt mode.

Remarks

This constant should always be defined.

Building the Library

This section lists the files that are available in the PIC32MX USB Driver Library.

Description

This section list the files that are available in the `\src` folder of the PIC32MX USB Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/usb/usbfs`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/drv_usbfs.h	This file should be included by any `.c` file which accesses the PIC32MX USB Driver API. This one file contains the prototypes for all driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/drv_usbfs.c	This file should always be included in the project when using the PIC3MX USB Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
/src/dynamic/drv_usbfs_device.c	This file should be included in the project if Device mode operation is required.
/src/dynamic/drv_usbfs_host.c	This file should be included in the project if Host mode operation is required.

Module Dependencies

The PIC32MX USB Driver Library depends on the following modules:

- Interrupt System Service Library

Library Interface

a) System Functions

	Name	Description
	DRV_USBFS_Status	Provides the current status of the USB Driver module.
	DRV_USBFS_Tasks	Maintains the driver's state machine when the driver is configured for Polled mode.
	DRV_USBFS_Tasks_ISR	Maintains the driver's Interrupt state machine and implements its ISR.

b) Client Core Functions

	Name	Description
	DRV_USBFS_ClientEventCallBackSet	This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events.
	DRV_USBFS_Close	Closes an opened-instance of the USB Driver.
	DRV_USBFS_Initialize	Initializes the USB Driver.
	DRV_USBFS_Open	Opens the specified USB Driver instance and returns a handle to it.

c) Device Mode Operation Functions

	Name	Description
	DRV_USBFS_DEVICE_AddressSet	This function will set the USB module address that is obtained from the Host.
	DRV_USBFS_DEVICE_Attach	This function will enable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that a device has been attached on the bus.
	DRV_USBFS_DEVICE_CurrentSpeedGet	This function returns the USB speed at which the device is operating.
	DRV_USBFS_DEVICE_Detach	This function will disable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that the device has detached from the bus.
	DRV_USBFS_DEVICE_EndpointDisable	This function disables an endpoint.
	DRV_USBFS_DEVICE_EndpointDisableAll	This function disables all provisioned endpoints.
	DRV_USBFS_DEVICE_EndpointEnable	This function enables an endpoint for the specified direction and endpoint size.
	DRV_USBFS_DEVICE_EndpointIsEnabled	This function returns the enable/disable status of the specified endpoint and direction.
	DRV_USBFS_DEVICE_EndpointIsStalled	This function returns the stall status of the specified endpoint and direction.
	DRV_USBFS_DEVICE_EndpointStall	This function stalls an endpoint in the specified direction.
	DRV_USBFS_DEVICE_EndpointStallClear	This function clears the stall on an endpoint in the specified direction.
	DRV_USBFS_DEVICE_IRPCancel	This function cancels the specific IRP that are queued and in progress at the specified endpoint.
	DRV_USBFS_DEVICE_IRPCancelAll	This function cancels all IRPs that are queued and in progress at the specified endpoint.
	DRV_USBFS_DEVICE_IRPSubmit	This function submits an I/O Request Packet (IRP) for processing to the Hi-Speed USB Driver.
	DRV_USBFS_DEVICE_RemoteWakeupStart	This function causes the device to start Remote Wakeup Signalling on the bus.
	DRV_USBFS_DEVICE_RemoteWakeupStop	This function causes the device to stop the Remote Wakeup Signalling on the bus.
	DRV_USBFS_DEVICE_SOFNumberGet	This function will return the USB SOF packet number.

d) Host Mode Operation Functions

	Name	Description
	DRV_USBFS_HOST_EventsDisable	Disables Host mode events.
	DRV_USBFS_HOST_EventsEnable	Restores the events to the specified the original value.
	DRV_USBFS_HOST_IRPCancel	Cancels the specified IRP.
	DRV_USBFS_HOST_IRPSubmit	Submits an IRP on a pipe.
	DRV_USBFS_HOST_PipeClose	Closes an open pipe.
	DRV_USBFS_HOST_PipeSetup	Open a pipe with the specified attributes.

e) Root Hub Functions

	Name	Description
	DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet	This function returns the operating speed of the bus to which this root hub is connected.
	DRV_USBFS_HOST_ROOT_HUB_Initialize	This function initializes the root hub driver.
	DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet	Returns the maximum amount of current that this root hub can provide on the bus.
	DRV_USBFS_HOST_ROOT_HUB_OperationEnable	This function enables or disables root hub operation.
	DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled	Returns the operation enabled status of the root hub.
	DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet	Returns the number of ports this root hub contains.
	DRV_USBFS_HOST_ROOT_HUB_PortReset	Resets the specified root hub port.
	DRV_USBFS_HOST_ROOT_HUB_PortResetIsComplete	Returns true if the root hub has completed the port reset operation.
	DRV_USBFS_HOST_ROOT_HUB_PortResume	Resumes the specified root hub port.
	DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet	Returns the speed of at which the port is operating.
	DRV_USBFS_HOST_ROOT_HUB_PortSuspend	Suspends the specified root hub port.

f) Data Types and Constants

	Name	Description
	DRV_USBFS_EVENT	Identifies the different events that the USB Driver provides.
	DRV_USBFS_EVENT_CALLBACK	Type of the USB Driver event callback function.
	DRV_USBFS_HOST_PIPE_HANDLE	Defines the USB Driver Host Pipe Handle type.

	DRV_USBFS_INIT	This type definition defines the Driver Initialization Data Structure.
	DRV_USBFS_OPMODES	Identifies the operating modes supported by the USB Driver.
	DRV_USBFS_ROOT_HUB_PORT_INDICATION	USB Root hub Application Hooks (Port Indication).
	DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT	USB Root hub Application Hooks (Port Overcurrent detection).
	DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE	USB Root hub Application Hooks (Port Power Enable/ Disable).
	DRV_USBFS_DEVICE_INTERFACE	USB Driver Device Mode Interface Functions.
	DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE	USB Driver Endpoint Table Entry Size in bytes.
	DRV_USBFS_HOST_INTERFACE	USB Driver Host Mode Interface Functions.
	DRV_USBFS_HOST_PIPE_HANDLE_INVALID	Value of an Invalid Host Pipe Handle.
	DRV_USBFS_INDEX_0	USB Driver Module Index 0 Definition.
	DRV_USBFS_INDEX_1	USB Driver Module Index 1 Definition.

Description

a) System Functions

DRV_USBFS_Status Function

Provides the current status of the USB Driver module.

File

drv_usbfs.h

C

```
SYS_STATUS DRV_USBFS_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that the driver is ready.
- SYS_STATUS_UNINITIALIZED - Indicates that the driver has never been initialized.

Description

This function provides the current status of the USB Driver module.

Remarks

None.

Preconditions

The DRV_USBFS_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USBFS_Initialize
SYS_STATUS          status;
DRV_USBFS_INIT moduleInit;

uint8_t __attribute__((aligned(512))) endpointTable[DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE * 2];

usbInitData.usbID                = USB_ID_1;
usbInitData.opMode               = DRV_USBFS_OPMODE_DEVICE;
usbInitData.stopInIdle           = false;
usbInitData.suspendInSleep       = false;
usbInitData.operationSpeed       = USB_SPEED_FULL;
usbInitData.interruptSource      = INT_SOURCE_USB;

usbInitData.sysModuleInit.powerState = SYS_MODULE_POWER_RUN_FULL ;

// This is how this data structure is passed to the initialize
// function.

DRV_USBFS_Initialize(DRV_USBFS_INDEX_0, (SYS_MODULE_INIT *) &usbInitData);
```

```
// The status of the driver can be checked.
status = DRV_USBFS_Status(object);
if(SYS_STATUS_READY == status)
{
    // Driver is ready to be opened.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_USBFS_Initialize function.

Function

SYS_STATUS DRV_USBFS_Status (SYS_MODULE_OBJ object)

DRV_USBFS_Tasks Function

Maintains the driver's state machine when the driver is configured for Polled mode.

File

drv_usbfs.h

C

```
void DRV_USBFS_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Maintains the driver's Polled state machine. This function should be called from the SYS_Tasks function.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks). This function will never block.

Preconditions

The DRV_USBFS_Initialize function must have been called for the specified USB Driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USBFS_Initialize

while (true)
{
    DRV_USBFS_Tasks(object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USBFS_Initialize function).

Function

void DRV_USBFS_Tasks(SYS_MODULE_OBJ object)

DRV_USBFS_Tasks_ISR Function

Maintains the driver's Interrupt state machine and implements its ISR.

File

drv_usbfs.h

C

```
void DRV_USBFS_Tasks_ISR(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal Interrupt state machine and implement its ISR for interrupt-driven implementations.

Remarks

This routine should be called from the USB interrupt service routine. In case of multiple USB modules, it should be ensured that the correct USB driver system module object is passed to this routine.

Preconditions

The DRV_USBFS_Initialize function must have been called for the specified USB Driver instance.

Example

```
SYS_MODULE_OBJ object;     // Returned from DRV_USBFS_Initialize

while (true)
{
    DRV_USBFS_Tasks_ISR (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USBFS_Initialize).

Function

void DRV_USBFS_Tasks_ISR(SYS_MODULE_OBJ object)

b) Client Core Functions

DRV_USBFS_ClientEventCallBackSet Function

This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events.

File

drv_usbfs.h

C

```
void DRV_USBFS_ClientEventCallBackSet(DRV_HANDLE handle, uintptr_t hReferenceData, DRV_USB_EVENT_CALLBACK
myEventCallBack);
```

Returns

None.

Description

This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events. The callback is disabled by either not calling this function after the DRV_USBFS_Open function has been called or by setting the myEventCallBack argument as NULL. When the callback function is called, the hReferenceData argument is returned.

Remarks

Typical usage of the USB Driver requires a client to register a callback.

Preconditions

None.

Example

```
// Set the client event callback for the Device Layer.  The
// USBDeviceLayerEventHandler function is the event handler. When this
// event handler is invoked by the driver, the driver returns back the
```

```
// second argument specified in the following function (which in this case
// is the Device Layer data structure). This allows the application
// firmware to identify, as an example, the Device Layer object associated
// with this callback.

DRV_USBFS_ClientEventCallBackSet(myUSBDevice.usbDriverHandle, (uintptr_t)&myUSBDevice,
USBDeviceLayerEventHandler);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
hReferenceData	Object (could be a pointer) that is returned with the callback.
myEventCallBack	Callback function for all USB events.

Function

void DRV_USBFS_ClientEventCallBackSet

(

DRV_HANDLE handle,

uintptr_t hReferenceData,

DRV_USBFS_EVENT_CALLBACK myEventCallBack

);

DRV_USBFS_Close Function

Closes an opened-instance of the USB Driver.

File

drv_usbfs.h

C

```
void DRV_USBFS_Close(DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the USB Driver, invalidating the handle.

Remarks

After calling this function, the handle passed in handle parameter must not be used with any of the other driver functions. A new handle must be obtained by calling DRV_USBFS_Open function before the caller may use the driver again.

Preconditions

The DRV_USBFS_Initialize function must have been called for the specified USB Driver instance. DRV_USBFS_Open function must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_USBFS_Open

DRV_USBFS_Close(handle);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

void DRV_USBFS_Close(DRV_HANDLE handle)

DRV_USBFS_Initialize Function

Initializes the USB Driver.

File

drv_usbfs.h

C

```
SYS_MODULE_OBJ DRV_USBFS_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

- SYS_MODULE_OBJ_INVALID - The driver initialization failed.
- A valid System Module Object - The driver initialization was able to start. It may have not completed and requires the DRV_USBFS_Tasks function to be called periodically. This value will never be the same as SYS_MODULE_OBJ_INVALID.

Description

This function initializes the USB Driver, making it ready for clients to open. The driver initialization does not complete when this function returns. The DRV_USBFS_Tasks function must called periodically to complete the driver initialization. The DRV_USBHS_Open function will fail if the driver was not initialized or if initialization has not completed.

Remarks

This routine must be called before any other USB driver routine is called. This routine should only be called once during system initialization unless DRV_USBFS_Deinitialize is called to deinitialize the driver instance.

Preconditions

None.

Example

```
 // The following code shows an example initialization of the
 // driver. The USB module to be used is USB1.  The module should not
 // automatically suspend when the  microcontroller enters Sleep mode.  The
 // module should continue  operation when the CPU enters Idle mode.  The
 // power state is set to run at full clock speeds. Device Mode operation
 // should be at FULL speed. The size of the endpoint table is set for 2
 // endpoints.

DRV_USBFS_INIT moduleInit;

uint8_t __attribute__((aligned(512))) endpointTable[DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE * 2];

usbInitData.usbID               = USB_ID_1;
usbInitData.opMode              = DRV_USBFS_OPMODE_DEVICE;
usbInitData.stopInIdle          = false;
usbInitData.suspendInSleep      = false;
usbInitData.operationSpeed      = USB_SPEED_FULL;
usbInitData.interruptSource     = INT_SOURCE_USB;

usbInitData.sysModuleInit.powerState = SYS_MODULE_POWER_RUN_FULL ;

// This is how this data structure is passed to the initialize
// function.

DRV_USBFS_Initialize(DRV_USBFS_INDEX_0, (SYS_MODULE_INIT *) &usbInitData);
```

Parameters

Parameters	Description
drvIndex	Ordinal number of driver instance to be initialized. This should be set to DRV_USBFS_INDEX_0 if driver instance 0 needs to be initialized.
init	Pointer to a data structure containing data necessary to initialize the driver. This should be a DRV_USBFS_INIT structure reference typecast to SYS_MODULE_INIT reference.

Function

SYS_MODULE_OBJ DRV_USBHS_Initialize

(

const SYS_MODULE_INDEX drvIndex,

const SYS_MODULE_INIT * const init

)

DRV_USBFS_Open Function

Opens the specified USB Driver instance and returns a handle to it.

File

drv_usbfs.h

C

```
DRV_HANDLE DRV_USBFS_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

- DRV_HANDLE_INVALID - The driver could not be opened successfully.This can happen if the driver initialization was not complete or if an internal error has occurred.
- A Valid Driver Handle - This is an arbitrary value and is returned if the function was successful. This value will never be the same as DRV_HANDLE_INVALID.

Description

This function opens the specified USB Driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The intent flag should always be DRV_IO_INTENT_EXCLUSIVE|DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NON_BLOCKING. Any other setting of the intent flag will return a invalid driver handle. A driver instance can only support one client. Trying to open a driver that has an existing client will result in an unsuccessful function call.

Remarks

The handle returned is valid until the DRV_USBFS_Close function is called. The function will typically return DRV_HANDLE_INVALID if the driver was not initialized. In such a case the client should try to open the driver again.

Preconditions

Function DRV_USBFS_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

// This code assumes that the driver has been initialized.
handle = DRV_USBFS_Open(DRV_USBFS_INDEX_0, DRV_IO_INTENT_EXCLUSIVE| DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NON_BLOCKING);

if(DRV_HANDLE_INVALID == handle)
{
    // The application should try opening the driver again.
}
```

Parameters

Parameters	Description
drvIndex	Identifies the driver instance to be opened. As an example, this value can be set to DRV_USBFS_INDEX_0 if instance 0 of the driver has to be opened.
intent	Should always be (DRV_IO_INTENT_EXCLUSIVE|DRV_IO_INTENT_READWRITE| DRV_IO_INTENT_NON_BLOCKING).

Function

```
DRV_HANDLE DRV_USBFS_Open
(
const    SYS_MODULE_INDEX drvIndex,
const    DRV_IO_INTENT intent
)
```

c) Device Mode Operation Functions

DRV_USBFS_DEVICE_AddressSet Function

This function will set the USB module address that is obtained from the Host.

File

drv_usbfs.h

C

```
void DRV_USBFS_DEVICE_AddressSet(DRV_HANDLE handle, uint8_t address);
```

Returns

None.

Description

This function will set the USB module address that is obtained from the Host in a setup transaction. The address is obtained from the SET_ADDRESS command issued by the Host. The primary (first) client of the driver uses this function to set the module's USB address after decoding the setup transaction from the Host.

Remarks

None.

Preconditions

None.

Example

```
// This function should be called by the first client of the driver,
// which is typically the Device Layer. The address to set is obtained
// from the Host during enumeration.

DRV_USBFS_DEVICE_AddressSet(deviceLayer, 4);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
address	The address of this module on the USB bus.

Function

void DRV_USBFS_DEVICE_AddressSet(DRV_HANDLE handle, uint8_t address);

DRV_USBFS_DEVICE_Attach Function

This function will enable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that a device has been attached on the bus.

File

drv_usbfs.h

C

```
void DRV_USBFS_DEVICE_Attach(DRV_HANDLE handle);
```

Returns

None.

Description

This function enables the pull-up resistors on the D+ or D- lines thus letting the USB Host know that a device has been attached on the bus . This function should be called when the driver client is ready to receive communication from the Host (typically after all initialization is complete). The USB 2.0 specification requires VBUS to be detected before the data line pull-ups are enabled. The application must ensure the same.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// Open the device driver and attach the device to the USB.
handle = DRV_USBFS_Open(DRV_USBFS_INDEX_0, DRV_IO_INTENT_EXCLUSIVE| DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NON_BLOCKING);

// Register a callback
DRV_USBFS_ClientEventCallBackSet(handle, (uintptr_t)&myDeviceLayer, MyDeviceLayerEventCallback);

// The device can be attached when VBUS Session Valid event occurs
void MyDeviceLayerEventCallback(uintptr_t handle, DRV_USBFS_EVENT event, void * hReferenceData)
{
    switch(event)
    {
        case DRV_USBFS_EVENT_DEVICE_SESSION_VALID:
            // A valid VBUS was detected.
            DRV_USBFS_DEVICE_Attach(handle);
            break;

        case DRV_USBFS_EVENT_DEVICE_SESSION_INVALID:
            // VBUS is not valid anymore. The device can be disconnected.
            DRV_USBFS_DEVICE_Detach(handle);
            break;

        default:
            break;
        }
    }
}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

void DRV_USBFS_DEVICE_Attach(DRV_HANDLE handle);

DRV_USBFS_DEVICE_CurrentSpeedGet Function

This function returns the USB speed at which the device is operating.

File

drv_usbfs.h

C

```
USB_SPEED DRV_USBFS_DEVICE_CurrentSpeedGet(DRV_HANDLE handle);
```

Returns

- USB_SPEED_ERROR - The device speed is not valid.
- USB_SPEED_FULL - The device is operating at Full speed.

Description

This function returns the USB speed at which the device is operating.

Remarks

None.

Preconditions

Only valid after the device is attached to the Host and Host has completed reset signaling.

Example

```
// Get the current speed.
```

```
USB_SPEED deviceSpeed;

deviceSpeed = DRV_USBFS_DEVICE_CurrentSpeedGet(deviceLayer);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

USB_SPEED DRV_USBFS_DEVICE_CurrentSpeedGet(DRV_HANDLE handle);

DRV_USBFS_DEVICE_Detach Function

This function will disable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that the device has detached from the bus.

File

drv_usbfs.h

C

```
void DRV_USBFS_DEVICE_Detach(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables the pull-up resistors on the D+ or D- lines. This function should be called when the application wants to disconnect the device from the bus (typically to implement a soft detach or switch to Host mode operation). A self-powered device should be detached from the bus when the VBUS is not valid.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// Open the device driver and attach the device to the USB.
handle = DRV_USBFS_Open(DRV_USBFS_INDEX_0, DRV_IO_INTENT_EXCLUSIVE| DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NON_BLOCKING);

// Register a callback
DRV_USBFS_ClientEventCallBackSet(handle, (uintptr_t)&myDeviceLayer, MyDeviceLayerEventCallback);

// The device can be detached when VBUS Session Invalid event occurs
void MyDeviceLayerEventCallback(uintptr_t handle, DRV_USBFS_EVENT event, void * hReferenceData)
{
    switch(event)
    {
        case DRV_USBFS_EVENT_DEVICE_SESSION_VALID:
            // A valid VBUS was detected.
            DRV_USBFS_DEVICE_Attach(handle);
            break;

        case DRV_USBFS_EVENT_DEVICE_SESSION_INVALID:
            // VBUS is not valid anymore. The device can be disconnected.
            DRV_USBFS_DEVICE_Detach(handle);
            break;

        default:
            break;
        }
    }
}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

void DRV_USBFS_DEVICE_Detach(DRV_HANDLE handle);

DRV_USBFS_DEVICE_EndpointDisable Function

This function disables an endpoint.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_EndpointDisable(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - The endpoint that is being accessed is not a valid endpoint (endpoint was not provisioned through the DRV_USBFS_ENDPOINTS_NUMBER configuration constant) defined for this driver instance.

Description

This function disables an endpoint. If the endpoint type is a control endpoint type, both directions are disabled. For non-control endpoints, the function disables the specified direction only. The direction to be disabled is specified by the Most Significant Bit (MSB) of the endpointAndDirection parameter.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to disable
// a control endpoint. Note that the direction parameter is ignored.
// For a control endpoint, both the directions are disabled.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 0);

DRV_USBFS_DEVICE_EndpointDisable(handle, ep );

// This code shows an example of how to disable a BULK IN
// endpoint

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBFS_DEVICE_EndpointDisable(handle, ep );
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

USB_ERROR DRV_USBFS_DEVICE_EndpointDisable

(

DRV_HANDLE handle,

USB_ENDPOINT endpointAndDirection
)

DRV_USBFS_DEVICE_EndpointDisableAll Function

This function disables all provisioned endpoints.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_EndpointDisableAll(DRV_HANDLE handle);
```

Returns

- USB_ERROR_NONE - The function exited successfully.
- USB_ERROR_PARAMETER_INVALID - The driver handle is invalid.

Description

This function disables all provisioned endpoints in both directions.

Remarks

This function is typically called by the USB Device Layer to disable all endpoints upon detecting a bus reset.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to disable all endpoints.

DRV_USBFS_DEVICE_EndpointDisableAll(handle);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

USB_ERROR DRV_USBFS_DEVICE_EndpointDisableAll(DRV_HANDLE handle)

DRV_USBFS_DEVICE_EndpointEnable Function

This function enables an endpoint for the specified direction and endpoint size.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_EndpointEnable(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection,
USB_TRANSFER_TYPE transferType, uint16_t endpointSize);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is not a valid endpoint defined for this driver instance. The value of DRV_USBFS_ENDPOINTS_NUMBER configuration constant should be adjusted.
- USB_ERROR_PARAMETER_INVALID - The driver handle is invalid.

Description

This function enables an endpoint for the specified direction and endpoint size. The function will enable the endpoint for communication in one direction at a time. It must be called twice if the endpoint is required to communicate in both the directions, with the exception of control endpoints. If the endpoint type is a control endpoint, the endpoint is always bidirectional and the function needs to be called only once.

The size of the endpoint must match the wMaxPacketSize reported in the endpoint descriptor for this endpoint. A transfer that is scheduled over

this endpoint will be scheduled in wMaxPacketSize transactions. The function does not check if the endpoint is already in use. It is the client's responsibility to make sure that a endpoint is not accidentally reused.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to enable Endpoint
// 0 for control transfers. Note that for a control endpoint, the
// direction parameter is ignored. A control endpoint is always
// bidirectional. Endpoint size is 64 bytes.

uint8_t ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 0);

DRV_USBFS_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_CONTROL, 64);

// This code shows an example of how to set up a endpoint
// for BULK IN transfer. For an IN transfer, data moves from device
// to Host. In this example, Endpoint 1 is enabled. The maximum
// packet size is 64.

uint8_t ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBFS_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_BULK, 64);

// If Endpoint 1 must also be set up for BULK OUT, the
// DRV_USBFS_DEVICE_EndpointEnable function must be called again, as shown
// here.

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_HOST_TO_DEVICE, 1);

DRV_USBFS_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_BULK, 64);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.
transferType	Should be USB_TRANSFER_TYPE_CONTROL for control endpoint, USB_TRANSFER_TYPE_BULK for bulk endpoint, USB_TRANSFER_TYPE_INTERRUPT for interrupt endpoint and USB_TRANSFER_TYPE_ISOCHRONOUS for isochronous endpoint.
endpointSize	Maximum size (in bytes) of the endpoint as reported in the endpoint descriptor.

Function

```
USB_ERROR DRV_USBFS_DEVICE_EndpointEnable
(
    DRV_HANDLE handle,
    USB_ENDPOINT endpointAndDirection,
USB_TRANSFER_TYPE transferType,
uint16_t endpointSize
);
```

DRV_USBFS_DEVICE_EndpointIsEnabled Function

This function returns the enable/disable status of the specified endpoint and direction.

File

drv_usbfs.h

C

```
bool DRV_USBFS_DEVICE_EndpointIsEnabled(DRV_HANDLE client, USB_ENDPOINT endpointAndDirection);
```

Returns

- true - The endpoint is enabled.
- false - The endpoint is disabled.

Description

This function returns the enable/disable status of the specified endpoint and direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how the
// DRV_USBFS_DEVICE_EndpointIsEnabled function can be used to obtain the
// status of Endpoint 1 and IN direction.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

if(DRV_USBFS_ENDPOINT_STATE_DISABLED ==
            DRV_USBFS_DEVICE_EndpointIsEnabled(handle, ep))
{
    // Endpoint is disabled. Enable endpoint.

    DRV_USBFS_DEVICE_EndpointEnable(handle, ep, USB_ENDPOINT_TYPE_BULK, 64);

}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

bool DRV_USBFS_DEVICE_EndpointIsEnabled

(

DRV_HANDLE handle,

USB_ENDPOINT endpointAndDirection

)

DRV_USBFS_DEVICE_EndpointIsStalled Function

This function returns the stall status of the specified endpoint and direction.

File

drv_usbfs.h

C

```
bool DRV_USBFS_DEVICE_EndpointIsStalled(DRV_HANDLE client, USB_ENDPOINT endpoint);
```

Returns

- true - The endpoint is stalled.

- false - The endpoint is not stalled.

Description

This function returns the stall status of the specified endpoint and direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how the
// DRV_USBFS_DEVICE_EndpointIsStalled function can be used to obtain the
// stall status of Endpoint 1 and IN direction.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

if(true == DRV_USBFS_DEVICE_EndpointIsStalled (handle, ep))
{
    // Endpoint stall is enabled. Clear the stall.

    DRV_USBFS_DEVICE_EndpointStallClear(handle, ep);

}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

bool DRV_USBFS_DEVICE_EndpointIsStalled

(

DRV_HANDLE handle,

USB_ENDPOINT endpointAndDirection

)

DRV_USBFS_DEVICE_EndpointStall Function

This function stalls an endpoint in the specified direction.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_EndpointStall(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.
- USB_ERROR_OSAL_FUNCTION - An error with an OSAL function called in this function.

Description

This function stalls an endpoint in the specified direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to stall an endpoint. In
// this example, Endpoint 1 IN direction is stalled.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBFS_DEVICE_EndpointStall(handle, ep);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

```
USB_ERROR DRV_USBFS_DEVICE_EndpointStall
(
    DRV_HANDLE handle,
    USB_ENDPOINT endpointAndDirection
)
```

DRV_USBFS_DEVICE_EndpointStallClear Function

This function clears the stall on an endpoint in the specified direction.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_EndpointStallClear(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Description

This function clears the stall on an endpoint in the specified direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to clear a stall. In this
// example, the stall condition on Endpoint 1 IN direction is cleared.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBFS_DEVICE_EndpointStallClear(handle, ep);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

```
  USB_ERROR DRV_USBFS_DEVICE_EndpointStallClear
(
    DRV_HANDLE handle,
    USB_ENDPOINT endpointAndDirection
)
```

DRV_USBFS_DEVICE_IRPCancel Function

This function cancels the specific IRP that are queued and in progress at the specified endpoint.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_IRPCancel(DRV_HANDLE client, USB_DEVICE_IRP * irp);
```

Returns

- USB_ERROR_NONE - The IRP have been canceled successfully.
- USB_ERROR_PARAMETER_INVALID - Invalid parameter or the IRP already has been aborted or completed
- USB_ERROR_OSAL_FUNCTION - An OSAL function called in this function did not execute successfully.

Description

This function attempts to cancel the processing of a queued IRP. An IRP that was in the queue but yet to be processed will be cancelled successfully and the IRP callback function will be called from this function with the USB_DEVICE_IRP_STATUS_ABORTED status. The application can release the data buffer memory used by the IRP when this callback occurs. If the IRP was in progress (a transaction in on the bus) when the cancel function was called, the IRP will be canceled only when an ongoing or the next transaction has completed. The IRP callback function will then be called in an interrupt context. The application should not release the related data buffer unless the IRP callback has occurred.

Remarks

The size returned after the ABORT callback will be always 0 regardless of the amount of data that has been sent or received. The client should not assume any data transaction has happened for an canceled IRP. If the last transaction of the IRP was in progress, the IRP cancel does not have any effect. The first transaction of any ongoing IRP cannot be canceled.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to cancel IRP.  In this example the IRP
// has been scheduled from a device to the Host.

USB_ENDPOINT ep;
USB_DEVICE_IRP irp;

ep.direction = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

irp.data = myDataBufferToSend;
irp.size = 130;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

if (DRV_USBFS_DEVICE_IRPSubmit(handle, ep, &irp) != USB_ERROR_NONE)
{
    // This means there was an error.
}
```

```
else
{
    // Check the status of the IRP.
    if(irp.status != USB_DEVICE_IRP_STATUS_COMPLETED)
    {
        // Cancel the submitted IRP.
        if (DRV_USBFS_DEVICE_IRPCancel(handle, &irp) != USB_ERROR_NONE)
        {
            // The IRP Cancel request submission was successful.
            // IRP cancel status will be notified through the callback
            // function.
        }
        else
        {
            // The IRP may have been completed before IRP cancel operation.
            // could start. No callback notification will be generated.
        }
    }
    else
    {
        // The IRP processing must have been completed before IRP cancel was
        // submitted.
    }
}

void MyIRPCallback(USB_DEVICE_IRP * irp)
{
    // Check if the IRP callback is for a Cancel request
    if(irp->status == USB_DEVICE_IRP_STATUS_ABORTED)
    {
        // IRP cancel completed
    }
 }
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
irp	Pointer to the IRP to cancel.

Function

```
USB_ERROR DRV_USBFS_DEVICE_IRPCancel
(
    DRV_HANDLE client,
    USB_DEVICE_IRP * irp
)
```

DRV_USBFS_DEVICE_IRPCancelAll Function

This function cancels all IRPs that are queued and in progress at the specified endpoint.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_IRPCancelAll(DRV_HANDLE client, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid.
- USB_ERROR_OSAL_FUNCTION - An OSAL function called in this function did not execute successfully.

Description

This function cancels all IRPs that are queued and in progress at the specified endpoint.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to cancel all IRPs.

void MyIRPCallback(USB_DEVICE_IRP * irp)
{
    // Check if this is setup command

    if(irp->status == USB_DEVICE_IRP_STATUS_SETUP)
    {
        if(IsSetupCommandSupported(irp->data) == false)
        {
            // This means that this setup command is not
            // supported. Stall the some related endpoint and cancel all
            // queue IRPs.

            DRV_USBFS_DEVICE_EndpointStall(handle, ep);
            DRV_USBFS_DEVICE_IRPCancelAll(handle, ep);
        }
     }
 }
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

```
USB_ERROR DRV_USBFS_DEVICE_IRPCancelAll
(
    DRV_HANDLE client,
    USB_ENDPOINT endpointAndDirection
);
```

DRV_USBFS_DEVICE_IRPSubmit Function

This function submits an I/O Request Packet (IRP) for processing to the Hi-Speed USB Driver.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_DEVICE_IRPSubmit(DRV_HANDLE client, USB_ENDPOINT endpointAndDirection, USB_DEVICE_IRP *
irp);
```

Returns

- USB_ERROR_NONE - if the IRP was submitted successful.
- USB_ERROR_IRP_SIZE_INVALID - if the size parameter of the IRP is not correct.
- USB_ERROR_PARAMETER_INVALID - If the client handle is not valid.
- USB_ERROR_ENDPOINT_NOT_CONFIGURED - If the endpoint is not enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - The specified endpoint is not valid.
- USB_ERROR_OSAL_FUNCTION - An OSAL call in the function did not complete successfully.

Description

This function submits an I/O Request Packet (IRP) for processing to the USB Driver. The IRP allows a client to send and receive data from the USB Host. The data will be sent or received through the specified endpoint. The direction of the data transfer is indicated by the direction flag in the endpointAndDirection parameter. Submitting an IRP arms the endpoint to either send data to or receive data from the Host. If an IRP is already being processed on the endpoint, the subsequent IRP submit operation will be queued. The contents of the IRP (including the application buffers) should not be changed until the IRP has been processed.

Particular attention should be paid to the size parameter of IRP. The following should be noted:

- The size parameter while sending data to the Host can be less than, greater than, equal to, or be an exact multiple of the maximum packet size for the endpoint. The maximum packet size for the endpoint determines the number of transactions required to process the IRP.
- If the size parameter, while sending data to the Host is less than the maximum packet size, the transfer will complete in one transaction.
- If the size parameter, while sending data to the Host is greater than the maximum packet size, the IRP will be processed in multiple transactions.
- If the size parameter, while sending data to the Host is equal to or an exact multiple of the maximum packet size, the client can optionally ask the driver to send a Zero Length Packet(ZLP) by specifying the USB_DEVICE_IRP_FLAG_DATA_COMPLETE flag as the flag parameter.
- The size parameter, while receiving data from the Host must be an exact multiple of the maximum packet size of the endpoint. If this is not the case, the driver will return a USB_ERROR_IRP_SIZE_INVALID result. If while processing the IRP, the driver receives less than maximum packet size or a ZLP from the Host, the driver considers the IRP as processed. The size parameter at this point contains the actual amount of data received from the Host. The IRP status is returned as USB_DEVICE_IRP_STATUS_COMPLETED_SHORT.
- If a ZLP needs to be sent to Host, the IRP size should be specified as 0 and the flag parameter should be set as USB_DEVICE_IRP_FLAG_DATA_COMPLETE.
- If the IRP size is an exact multiple of the endpoint size, the client can request the driver to not send a ZLP by setting the flag parameter to USB_DEVICE_IRP_FLAG_DATA_PENDING. This flag indicates that there is more data pending in this transfer.
- Specifying a size less than the endpoint size along with the USB_DEVICE_IRP_FLAG_DATA_PENDING flag will cause the driver to return a USB_ERROR_IRP_SIZE_INVALID.
- If the size is greater than but not a multiple of the endpoint size, and the flag is specified as USB_DEVICE_IRP_FLAG_DATA_PENDING, the driver will send multiple of endpoint size number of bytes. For example, if the IRP size is 130 and the endpoint size if 64, the number of bytes sent will 128.

Remarks

This function can be called from the ISR of the USB module to associated with the client.

Preconditions

The Client handle should be valid.

Example

```
// The following code shows an example of how to schedule a IRP to send data
// from a device to the Host. Assume that the max packet size is 64 and
// and this data needs to sent over Endpoint 1. In this example, the
// transfer is processed as three transactions of 64, 64 and 2 bytes.

USB_ENDPOINT ep;
USB_DEVICE_IRP irp;

ep.direction = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

irp.data = myDataBufferToSend;
irp.size = 130;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

if (DRV_USBFS_DEVICE_IRPSubmit(handle, ep, &irp) != USB_ERROR_NONE)
{
    // This means there was an error.
}
else
{
    // The status of the IRP can be checked.
    while(irp.status != USB_DEVICE_IRP_STATUS_COMPLETED)
    {
        // Wait or run a task function.
    }
}

// The following code shows how the client can request
// the driver to send a ZLP when the size is an exact multiple of
```

```
// endpoint size.

irp.data = myDataBufferToSend;
irp.size = 128;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

// Note that while  receiving data from the Host, the size should be an
// exact multiple of the maximum packet size of the endpoint. In the
// following example, the DRV_USBFS_DEVICE_IRPSubmit function will return a
// USB_DEVICE_IRP_SIZE_INVALID value.

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_HOST_TO_DEVICE, 1);

irp.data = myDataBufferToSend;
irp.size = 60; // THIS SIZE IS NOT CORRECT
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).
endpointAndDirection	Specifies the endpoint and direction.
irp	Pointer to the IRP to be added to the queue for processing.

Function

```
USB_ERROR DRV_USBFS_DEVICE_IRPSubmit
(
    DRV_HANDLE client,
    USB_ENDPOINT endpointAndDirection,
    USB_DEVICE_IRP * irp
);
```

DRV_USBFS_DEVICE_RemoteWakeupStart Function

This function causes the device to start Remote Wakeup Signalling on the bus.

File

drv_usbfs.h

C

```
void DRV_USBFS_DEVICE_RemoteWakeupStart(DRV_HANDLE handle);
```

Returns

None.

Description

This function causes the device to start Remote Wakeup Signalling on the bus. This function should be called when the device, presently placed in suspend mode by the Host, wants to be wakeup. Note that the device can do this only when the Host has enabled the device's Remote Wakeup capability.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
DRV_HANDLE handle;
```

```
// If the Host has enabled the Remote Wakeup capability, and if the device
// is in suspend mode, then start Remote Wakeup signaling.

if(deviceIsSuspended && deviceRemoteWakeupEnabled)
{
    DRV_USBFS_DEVICE_RemoteWakeupStart(handle);
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).

Function

void DRV_USBFS_DEVICE_RemoteWakeupStart(DRV_HANDLE handle);

DRV_USBFS_DEVICE_RemoteWakeupStop Function

This function causes the device to stop the Remote Wakeup Signalling on the bus.

File

drv_usbfs.h

C

```
void DRV_USBFS_DEVICE_RemoteWakeupStop(DRV_HANDLE handle);
```

Returns

None.

Description

This function causes the device to stop Remote Wakeup Signalling on the bus. This function should be called after the DRV_USBFS_DEVICE_RemoteWakeupStart function was called to start the Remote Wakeup signaling on the bus.

Remarks

This function should be 1 to 15 milliseconds after the DRV_USBFS_DEVICE_RemoteWakeupStart function was called.

Preconditions

The handle should be valid. The DRV_USBFS_DEVICE_RemoteWakeupStart function was called to start the Remote Wakeup signaling on the bus.

Example

```
DRV_HANDLE handle;

// If the Host has enabled the Remote Wakeup capability, and if the device
// is in suspend mode, then start Remote Wakeup signaling. Wait for 10
// milliseconds and then stop the Remote Wakeup signaling

if(deviceIsSuspended && deviceRemoteWakeupEnabled)
{
    DRV_USBFS_DEVICE_RemoteWakeupStart(handle);
    DelayMilliSeconds(10);
    DRV_USBFS_DEVICE_RemoteWakeupStop(handle);
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).

Function

void DRV_USBFS_DEVICE_RemoteWakeupStop(DRV_HANDLE handle);

DRV_USBFS_DEVICE_SOFNumberGet Function

This function will return the USB SOF packet number.

File

drv_usbfs.h

C

```
uint16_t DRV_USBFS_DEVICE_SOFNumberGet(DRV_HANDLE handle);
```

Returns

The SOF packet number.

Description

This function will return the USB SOF packet number..

Remarks

None.

Preconditions

This function will return a valid value only when the device is attached to the bus. The SOF packet count will not increment if the bus is suspended.

Example

```
// This code shows how the DRV_USBFS_DEVICE_SOFNumberGet function is called
// to read the current SOF number.

DRV_HANDLE handle;
uint16_t sofNumber;

sofNumber = DRV_USBFS_DEVICE_SOFNumberGet(handle);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

uint16_t DRV_USBFS_DEVICE_SOFNumberGet(DRV_HANDLE handle);

d) Host Mode Operation Functions

DRV_USBFS_HOST_EventsDisable Function

Disables Host mode events.

File

drv_usbfs.h

C

```
bool DRV_USBFS_HOST_EventsDisable(DRV_HANDLE handle);
```

Returns

- true - Driver event generation was enabled when this function was called.
- false - Driver event generation was not enabled when this function was called.

Description

This function disables the Host mode events. This function is called by the Host Layer when it wants to execute code atomically.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
// This code shows how the DRV_USBFS_HOST_EventsDisable and
// DRV_USBFS_HOST_EventsEnable function can be called to disable and enable
// events.

DRV_HANDLE driverHandle;
bool eventsWereEnabled;

// Disable the driver events.
eventsWereEnabled = DRV_USBFS_HOST_EventsDisable(driverHandle);

// Code in this region will not be interrupted by driver events.

// Enable the driver events.
DRV_USBFS_HOST_EventsEnable(driverHandle, eventsWereEnabled);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBFS_Open function).

Function

bool DRV_USBFS_HOST_EventsDisable

(

DRV_HANDLE handle

);

DRV_USBFS_HOST_EventsEnable Function

Restores the events to the specified the original value.

File

drv_usbfs.h

C

```
void DRV_USBFS_HOST_EventsEnable(DRV_HANDLE handle, bool eventContext);
```

Returns

None.

Description

This function will restore the enable disable state of the events. The eventRestoreContext parameter should be equal to the value returned by the DRV_USBFS_HOST_EventsDisable function.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
// This code shows how the DRV_USBFS_HOST_EventsDisable and
// DRV_USBFS_HOST_EventsEnable function can be called to disable and enable
// events.

DRV_HANDLE driverHandle;
bool eventsWereEnabled;

// Disable the driver events.
eventsWereEnabled = DRV_USBFS_HOST_EventsDisable(driverHandle);
```

```
// Code in this region will not be interrupted by driver events.

// Enable the driver events.
DRV_USBFS_HOST_EventsEnable(driverHandle, eventsWereEnabled);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
eventRestoreContext	Value returned by the DRV_USBFS_HOST_EventsDisable function.

Function

void DRV_USBFS_HOST_EventsEnable
(
DRV_HANDLE handle
bool eventRestoreContext
);

DRV_USBFS_HOST_IRPCancel Function

Cancels the specified IRP.

File

drv_usbfs.h

C

```
void DRV_USBFS_HOST_IRPCancel(USB_HOST_IRP * inputIRP);
```

Returns

None.

Description

This function attempts to cancel the specified IRP. If the IRP is queued and its processing has not started, it will be cancelled successfully. If the IRP in progress, the ongoing transaction will be allowed to complete.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how a submitted IRP can be cancelled.

USB_HOST_IRP irp;
USB_ERROR result;
USB_HOST_PIPE_HANDLE controlPipe;
USB_SETUP_PACKET setup;
uint8_t controlTransferData[32];

irp.setup = setup;
irp.data = controlTransferData;
irp.size = 32;
irp.flags = USB_HOST_IRP_FLAG_NONE ;
irp.userData = &someApplicationObject;
irp.callback = IRP_Callback;

DRV_USBFS_HOST_IRPSubmit(controlPipeHandle, &irp);

// Additional application logic may come here. This logic may decide to
// cancel the submitted IRP.

DRV_USBFS_HOST_IRPCancel(&irp);
```

Parameters

Parameters	Description
inputIRP	Pointer to the IRP to cancel.

Function

void DRV_USBFS_HOST_IRPCancel(USB_HOST_IRP * inputIRP);

DRV_USBFS_HOST_IRPSubmit Function

Submits an IRP on a pipe.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_HOST_IRPSubmit(DRV_USBFS_HOST_PIPE_HANDLE hPipe, USB_HOST_IRP * pinputIRP);
```

Returns

- USB_ERROR_NONE - The IRP was submitted successfully.
- USB_ERROR_PARAMETER_INVALID - The pipe handle is not valid.
- USB_ERROR_OSAL_FUNCTION - An error occurred in an OSAL function called in this function.

Description

This function submits an IRP on the specified pipe. The IRP will be added to the queue and will be processed in turn. The data will be transferred on the bus based on the USB bus scheduling rules. When the IRP has been processed, the callback function specified in the IRP will be called. The IRP status will be updated to reflect the completion status of the IRP.

Remarks

An IRP can also be submitted in an IRP callback function.

Preconditions

The pipe handle should be valid.

Example

```
// The following code shows an example of how the host layer populates
// the IRP object and then submits it. IRP_Callback function is called when an
// IRP has completed processing. The status of the IRP at completion can be
// checked in the status flag. The size field of the irp will contain the amount
// of data transferred.

void IRP_Callback(USB_HOST_IRP * irp)
{
    // irp is pointing to the IRP for which the callback has occurred. In most
    // cases this function will execute in an interrupt context. The application
    // should not perform any hardware access or interrupt un-safe operations in
    // this function.

    switch(irp->status)
    {
        case USB_HOST_IRP_STATUS_ERROR_UNKNOWN:
            // IRP was terminated due to an unknown error
            break;

        case USB_HOST_IRP_STATUS_ABORTED:
            // IRP was terminated by the application
            break;

        case USB_HOST_IRP_STATUS_ERROR_BUS:
            // IRP was terminated due to a bus error
            break;

        case USB_HOST_IRP_STATUS_ERROR_DATA:
            // IRP was terminated due to data error
```

```
            break;

        case USB_HOST_IRP_STATUS_ERROR_NAK_TIMEOUT:
            // IRP was terminated because of a NAK timeout
            break;

        case USB_HOST_IRP_STATUS_ERROR_STALL:
            // IRP was terminated because of a device sent a STALL
            break;

        case USB_HOST_IRP_STATUS_COMPLETED:
            // IRP has been completed
            break;

        case USB_HOST_IRP_STATUS_COMPLETED_SHORT:
            // IRP has been completed but the amount of data processed was less
            // than requested.
            break;

        default:
            break;
    }
}

// In the following code snippet the a control transfer IRP is submitted to a
// control pipe. The setup parameter of the IRP points to the Setup command of
// the control transfer. The direction of the data stage is specified by the
// Setup packet.

USB_HOST_IRP irp;
USB_ERROR result;
USB_HOST_PIPE_HANDLE controlPipe;
USB_SETUP_PACKET setup;
uint8_t controlTransferData[32];

irp.setup = setup;
irp.data = controlTransferData;
irp.size = 32;
irp.flags = USB_HOST_IRP_FLAG_NONE ;
irp.userData = &someApplicationObject;
irp.callback = IRP_Callback;

result = DRV_USBFS_HOST_IRPSubmit(controlPipeHandle, &irp);
```

Parameters

Parameters	Description
hPipe	Handle to the pipe to which the IRP has to be submitted.
pInputIRP	Pointer to the IRP.

Function

```
  USB_ERROR DRV_USBFS_HOST_IRPSubmit
(
    DRV_USBFS_HOST_PIPE_HANDLE  hPipe,
    USB_HOST_IRP * pInputIRP
);
```

DRV_USBFS_HOST_PipeClose Function

Closes an open pipe.

File

drv_usbfs.h

C

```
void DRV_USBFS_HOST_PipeClose(DRV_USBFS_HOST_PIPE_HANDLE pipeHandle);
```

Returns

None.

Description

This function closes an open pipe. Any IRPs scheduled on the pipe will be aborted and IRP callback functions will be called with the status as DRV_USB_HOST_IRP_STATE_ABORTED. The pipe handle will become invalid and the pipe will not accept IRPs.

Remarks

None.

Preconditions

The pipe handle should be valid.

Example

```
// This code shows how an open Host pipe can be closed.

DRV_HANDLE driverHandle;
DRV_USBFS_HOST_PIPE_HANDLE pipeHandle;

// Close the pipe.
DRV_USBFS_HOST_PipeClose(pipeHandle);
```

Parameters

Parameters	Description
pipeHandle	Handle to the pipe to close.

Function

void DRV_USBFS_HOST_PipeClose

(

DRV_USBFS_HOST_PIPE_HANDLE pipeHandle

);

DRV_USBFS_HOST_PipeSetup Function

Open a pipe with the specified attributes.

File

drv_usbfs.h

C

```
DRV_USBFS_HOST_PIPE_HANDLE DRV_USBFS_HOST_PipeSetup(DRV_HANDLE client, uint8_t deviceAddress, USB_ENDPOINT endpointAndDirection, uint8_t hubAddress, uint8_t hubPort, USB_TRANSFER_TYPE pipeType, uint8_t bInterval, uint16_t wMaxPacketSize, USB_SPEED speed);
```

Returns

- DRV_USB_HOST_PIPE_HANDLE_INVALID - The pipe could not be created.
- A valid Pipe Handle - The pipe was created successfully. This is an arbitrary value and will never be the same as DRV_USB_HOST_PIPE_HANDLE_INVALID.

Description

This function opens a communication pipe between the Host and the device endpoint. The transfer type and other attributes are specified through the function parameters. The driver does not check for available bus bandwidth, which should be done by the application (the USB Host Layer in this case)

Remarks

None.

Preconditions

The driver handle should be valid.

Example

```
// This code shows how the DRV_USBFS_HOST_PipeSetup function is called for
// create a communication pipe. In this example, Bulk pipe is created
// between the Host and a device. The Device address is 2 and the target
// endpoint on this device is 4 . The direction of the data transfer over
// this pipe is from the Host to the device. The device is connected to Port
// 1 of a Hub, whose USB address is 3. The maximum size of a transaction
// on this pipe is 64 bytes. This is a Bulk Pipe and hence the bInterval
// field is set to 0. The target device is operating at Full Speed.

DRV_HANDLE driverHandle;
DRV_USBFS_HOST_PIPE_HANDLE pipeHandle;

pipeHandle = DRV_USBFS_HOST_PipeSetup(driverHandle, 0x02, 0x14, 0x03, 0x01, USB_TRANSFER_TYPE_BULK, 0, 64,
USB_SPEED_FULL);

if(pipeHandle != DRV_USBFS_HOST_PIPE_HANDLE_INVALID)
{
    // The pipe was created successfully.
}
```

Parameters

Parameters	Description
client	Handle to the driver (returned from DRV_USBFS_Open function).
deviceAddress	USB Address of the device to connect to.
endpoint	Endpoint on the device to connect to.
hubAddress	Address of the hub to which this device is connected. If not connected to a hub, this value should be set to 0.
hubPort	Port number of the hub to which this device is connected.
pipeType	Transfer type of the pipe to open.
bInterval	Polling interval for periodic transfers. This should be specified as defined by the USB 2.0 Specification.
wMaxPacketSize	This should be set to the endpoint size reported by the device in its configuration descriptors. This defines the maximum size of the transaction in a transfer on this pipe.
speed	The speed of the pipe. This should match the speed at which the device connected to the Host.

Function

```
  DRV_USBFS_HOST_PIPE_HANDLE DRV_USBFS_HOST_PipeSetup
(
    DRV_HANDLE client,
uint8_t deviceAddress,
    USB_ENDPOINT endpointAndDirection,
uint8_t hubAddress,
uint8_t hubPort,
USB_TRANSFER_TYPE pipeType,
uint8_t bInterval,
uint16_t wMaxPacketSize,
    USB_SPEED speed
);
```

e) Root Hub Functions

DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet Function

This function returns the operating speed of the bus to which this root hub is connected.

File

drv_usbfs.h

C

```
USB_SPEED DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet(DRV_HANDLE handle);
```

Returns

- USB_SPEED_FULL - The Root hub is connected to a bus that is operating at Full Speed.

Description

This function returns the operating speed of the bus to which this root hub is connected.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet function is
// called to know the operating speed of the bus to which this Root hub is
// connected.

DRV_HANDLE driverHandle;
USB_SPEED speed;

speed = DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet(driverHandle);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).

Function

USB_SPEED DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet(DRV_HANDLE handle);

DRV_USBFS_HOST_ROOT_HUB_Initialize Function

This function initializes the root hub driver.

File

drv_usbfs.h

C

```
void DRV_USBFS_HOST_ROOT_HUB_Initialize(DRV_HANDLE handle, USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo);
```

Returns

None.

Description

This function initializes the root hub driver. It is called by the Host Layer at the time of processing the root hub devices. The Host Layer assigns a USB_HOST_DEVICE_INFO reference to this root hub driver. This identifies the relationship between the root hub and the Host Layer.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the USB Host Layer calls the
// DRV_USBFS_HOST_ROOT_HUB_Initialize function. The usbHostDeviceInfo
// parameter is an arbitrary identifier assigned by the USB Host Layer. Its
// interpretation is opaque to the Root hub Driver.

DRV_HANDLE drvHandle;
USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo = 0x10003000;

DRV_USBFS_HOST_ROOT_HUB_Initialize(drvHandle, usbHostDeviceInfo);
```

Parameters

Parameters	Description
handle	Handle to the driver.
usbHostDeviceInfo	Reference provided by the Host.

Function

void DRV_USBFS_HOST_ROOT_HUB_Initialize

(

DRV_HANDLE handle,

USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo,

)

DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet Function

Returns the maximum amount of current that this root hub can provide on the bus.

File

drv_usbfs.h

C

```
uint32_t DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet(DRV_HANDLE handle);
```

Returns

Returns the maximum current (in milliamperes) that the root hub can supply.

Description

This function returns the maximum amount of current that this root hub can provide on the bus.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet
// function is called to obtain the maximum VBUS current that the Root hub
// can supply.

DRV_HANDLE driverHandle;
uint32_t currentMilliAmperes;

currentMilliAmperes = DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet(driverHandle);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).

Function

uint32_t DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet(DRV_HANDLE);

DRV_USBFS_HOST_ROOT_HUB_OperationEnable Function

This function enables or disables root hub operation.

File

drv_usbfs.h

C

```
void DRV_USBFS_HOST_ROOT_HUB_OperationEnable(DRV_HANDLE handle, bool enable);
```

Returns

None.

Description

This function enables or disables root hub operation. When enabled, the root hub will detect devices attached to the port and will request the Host Layer to enumerate the device. This function is called by the Host Layer when it is ready to receive enumeration requests from the Host. If the operation is disabled, the root hub will not detect attached devices.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_OperationEnable and the
// DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled functions are called to enable
// the Root hub operation.

DRV_HANDLE driverHandle;

// Enable Root hub operation.
DRV_USBFS_HOST_ROOT_HUB_OperationEnable(driverHandle);

// Wait till the Root hub operation is enabled.
if(DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled(driverHandle) == false)
{
    // The operation has not completed. Call the
    // DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled function again to check if
    // the operation has completed. Note that the DRV_USBFS_Tasks function
    // must be allowed to run at periodic intervals to allow the enable
    // operation to completed.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
enable	If this is set to true, root hub operation is enabled. If this is set to false, root hub operation is disabled.

Function

void DRV_USBFS_HOST_ROOT_HUB_OperationEnable

(

DRV_HANDLE handle,

bool enable

);

DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled Function

Returns the operation enabled status of the root hub.

File

drv_usbfs.h

C

```
bool DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled(DRV_HANDLE handle);
```

Returns

- true - Root hub operation is enabled.
- false - Root hub operation is not enabled.

Description

This function returns true if the DRV_USBFS_HOST_ROOT_HUB_OperationEnable function has completed enabling the Host.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_OperationEnable and the
// DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled functions are called to enable
// the Root hub operation.

DRV_HANDLE driverHandle;

// Enable Root hub operation.
DRV_USBFS_HOST_ROOT_HUB_OperationEnable(driverHandle);

// Wait till the Root hub operation is enabled.
if(DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled(driverHandle) == false)
{
    // The operation has not completed. Call the
    // DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled function again to check if
    // the operation has completed. Note that the DRV_USBFS_Tasks function
    // must be allowed to run at periodic intervals to allow the enable
    // operation to completed.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).

Function

bool DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled(DRV_HANDLE handle);

DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet Function

Returns the number of ports this root hub contains.

File

drv_usbfs.h

C

```
uint8_t DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet(DRV_HANDLE handle);
```

Returns

This function will always return 1.

Description

This function returns the number of ports that this root hub contains.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet function can
// be called to obtain the number of Root hub ports.

DRV_HANDLE driverHandle;
uint8_t nPorts;

nPorts = DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet(driverHandle);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).

Function

uint8_t DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet(DRV_HANDLE handle);

DRV_USBFS_HOST_ROOT_HUB_PortReset Function

Resets the specified root hub port.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_HOST_ROOT_HUB_PortReset(DRV_HANDLE handle, uint8_t port);
```

Returns

None.

Description

This function resets the root hub port. The reset duration is defined by DRV_USBFS_ROOT_HUB_RESET_DURATION. The status of the reset signaling can be checked using the DRV_USBFS_ROOT_HUB_PortResetIsComplete function.

Remarks

The root hub on the PIC32MZ USB controller contains only one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USB_HOST_ROOT_HUB_PortReset and the
// DRV_USBFS_ROOT_HUB_PortResetIsComplete functions are called to complete a
// port reset sequence.

DRV_HANDLE driverHandle;

// Reset Port 0.
DRV_USB_HOST_ROOT_HUB_PortReset(driverHandle, 0);

// Check if the Reset operation has completed.
if(DRV_USBFS_ROOT_HUB_PortResetIsComplete(driverHandle, 0) == false)
{
    // This means that the Port Reset operation has not completed yet. The
    // DRV_USBFS_ROOT_HUB_PortResetIsComplete function should be called
    // again after some time to check the status.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
port	Port to reset.

Function

void DRV_USBFS_ROOT_HUB_PortReset(DRV_HANDLE handle, uint8_t port);

DRV_USBFS_HOST_ROOT_HUB_PortResetIsComplete Function

Returns true if the root hub has completed the port reset operation.

File

drv_usbfs.h

C

```
bool DRV_USBFS_HOST_ROOT_HUB_PortResetIsComplete(DRV_HANDLE handle, uint8_t port);
```

Returns

- true - The reset signaling has completed.
- false - The reset signaling has not completed.

Description

This function returns true if the port reset operation has completed. It should be called after the DRV_USB_HOST_ROOT_HUB_PortReset function to check if the reset operation has completed.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USB_HOST_ROOT_HUB_PortReset and the
// DRV_USBFS_ROOT_HUB_PortResetIsComplete functions are called to complete a
// port reset sequence.

DRV_HANDLE driverHandle;

// Reset Port 0.
DRV_USB_HOST_ROOT_HUB_PortReset(driverHandle, 0);

// Check if the Reset operation has completed.
if(DRV_USBFS_ROOT_HUB_PortResetIsComplete(driverHandle, 0) == false)
{
    // This means that the Port Reset operation has not completed yet. The
    // DRV_USBFS_ROOT_HUB_PortResetIsComplete function should be called
    // again after some time to check the status.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
port	Port to check

Function

bool DRV_USBFS_ROOT_HUB_PortResetIsComplete

(

DRV_HANDLE handle,

uint8_t port

);

DRV_USBFS_HOST_ROOT_HUB_PortResume Function

Resumes the specified root hub port.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_HOST_ROOT_HUB_PortResume(DRV_HANDLE handle, uint8_t port);
```

Returns

- USB_ERROR_NONE - The function executed successfully.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid or the port number does not exist.

Description

This function resumes the root hub. The resume duration is defined by DRV_USBFS_ROOT_HUB_RESUME_DURATION. The status of the resume signaling can be checked using the DRV_USBFS_ROOT_HUB_PortResumeIsComplete function.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_PortResume function is
// called to resume the specified port.

DRV_HANDLE driverHandle;

// Resume Port 0.
DRV_USBFS_HOST_ROOT_HUB_PortResume(driverHandle, 0);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
port	Port to resume.

Function

USB_ERROR DRV_USBFS_HOST_ROOT_HUB_PortResume

(

DRV_HANDLE handle,

uint8_t port

);

DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet Function

Returns the speed of at which the port is operating.

File

drv_usbfs.h

C

```
USB_SPEED DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet(DRV_HANDLE handle, uint8_t port);
```

Returns

- USB_SPEED_ERROR - This value is returned if the driver handle is not or if the speed information is not available or if the specified port is not valid.
- USB_SPEED_FULL - A Full Speed device has been connected to the port.

- USB_SPEED_LOW - A Low Speed device has been connected to the port.

Description

This function returns the speed at which the port is operating.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet function is
// called to know the operating speed of the port. This also indicates the
// operating speed of the device connected to this port.

DRV_HANDLE driverHandle;
USB_SPEED speed;

speed = DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet(driverHandle, 0);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
port	Port number of the port to be analyzed..

Function

```
  USB_SPEED DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet
(
    DRV_HANDLE handle,
uint8_t port
);
```

DRV_USBFS_HOST_ROOT_HUB_PortSuspend Function

Suspends the specified root hub port.

File

drv_usbfs.h

C

```
USB_ERROR DRV_USBFS_HOST_ROOT_HUB_PortSuspend(DRV_HANDLE handle, uint8_t port);
```

Returns

- USB_ERROR_NONE - The function executed successfully.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid or the port number does not exist.

Description

This function suspends the root hub port.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USBFS_HOST_ROOT_HUB_PortSuspend function is
// called to suspend the specified port.

DRV_HANDLE driverHandle;
```

```
// Suspend Port 0.
DRV_USBFS_HOST_ROOT_HUB_PortSuspend(driverHandle, 0);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBFS_Open function).
port	Port to suspend.

Function

USB_ERROR DRV_USBFS_ROOT_HUB_PortSuspend(DRV_HANDLE handle, uint8_t port);

f) Data Types and Constants

DRV_USBFS_EVENT Enumeration

Identifies the different events that the USB Driver provides.

File

drv_usbfs.h

C

```
typedef enum {
  DRV_USBFS_EVENT_ERROR = DRV_USB_EVENT_ERROR,
  DRV_USBFS_EVENT_RESET_DETECT = DRV_USB_EVENT_RESET_DETECT,
  DRV_USBFS_EVENT_RESUME_DETECT = DRV_USB_EVENT_RESUME_DETECT,
  DRV_USBFS_EVENT_IDLE_DETECT = DRV_USB_EVENT_IDLE_DETECT,
  DRV_USBFS_EVENT_STALL = DRV_USB_EVENT_STALL,
  DRV_USBFS_EVENT_SOF_DETECT = DRV_USB_EVENT_SOF_DETECT,
  DRV_USBFS_EVENT_DEVICE_SESSION_VALID = DRV_USB_EVENT_DEVICE_SESSION_VALID,
  DRV_USBFS_EVENT_DEVICE_SESSION_INVALID = DRV_USB_EVENT_DEVICE_SESSION_INVALID
} DRV_USBFS_EVENT;
```

Members

Members	Description
DRV_USBFS_EVENT_ERROR = DRV_USB_EVENT_ERROR	Bus error occurred and was reported
DRV_USBFS_EVENT_RESET_DETECT = DRV_USB_EVENT_RESET_DETECT	Host has issued a device reset
DRV_USBFS_EVENT_RESUME_DETECT = DRV_USB_EVENT_RESUME_DETECT	Resume detected while USB in suspend mode
DRV_USBFS_EVENT_IDLE_DETECT = DRV_USB_EVENT_IDLE_DETECT	Idle detected
DRV_USBFS_EVENT_STALL = DRV_USB_EVENT_STALL	Stall handshake has occurred
DRV_USBFS_EVENT_SOF_DETECT = DRV_USB_EVENT_SOF_DETECT	Either Device received SOF or SOF threshold was reached in the Host mode operation
DRV_USBFS_EVENT_DEVICE_SESSION_VALID = DRV_USB_EVENT_DEVICE_SESSION_VALID	Session valid
DRV_USBFS_EVENT_DEVICE_SESSION_INVALID = DRV_USB_EVENT_DEVICE_SESSION_INVALID	Session Invalid

Description

USB Driver Events Enumeration.

This enumeration identifies the different events that are generated by the USB Driver.

Remarks

None.

DRV_USBFS_EVENT_CALLBACK Type

Type of the USB Driver event callback function.

File

drv_usbfs.h

C

```
typedef void (* DRV_USBFS_EVENT_CALLBACK)(uintptr_t hClient, DRV_USBFS_EVENT eventType, void * eventData);
```

Returns

None.

Description

Type of the USB Driver Event Callback Function.

Define the type of the USB Driver event callback function. The client should register an event callback function of this type when it intends to receive events from the USB Driver. The event callback function is registered using the DRV_USBFS_ClientEventCallBackSet function.

Remarks

None.

Parameters

Parameters	Description
hClient	Handle to the driver client that registered this callback function.
eventType	This parameter identifies the event that caused the callback function to be called.
eventData	Pointer to a data structure that is related to this event. This value will be NULL if the event has no related data.

DRV_USBFS_HOST_PIPE_HANDLE Type

Defines the USB Driver Host Pipe Handle type.

File

drv_usbfs.h

C

```
typedef uintptr_t DRV_USBFS_HOST_PIPE_HANDLE;
```

Description

USB Driver Host Pipe Handle.

This type definition defines the type of the USB Driver Host Pipe Handle.

Remarks

None.

DRV_USBFS_INIT Structure

This type definition defines the Driver Initialization Data Structure.

File

drv_usbfs.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  USB_MODULE_ID usbID;
  bool stopInIdle;
  bool suspendInSleep;
  INT_SOURCE interruptSource;
  USB_SPEED operationSpeed;
  DRV_USBFS_OPMODES operationMode;
  void * endpointTable;
```

```
  uint32_t rootHubAvailableCurrent;
  DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE portPowerEnable;
  DRV_USBFS_ROOT_HUB_PORT_INDICATION portIndication;
  DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT portOverCurrentDetect;
} DRV_USBFS_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System Module Initialization
USB_MODULE_ID usbID;	Identifies the USB peripheral to be used. This should be the USB PLIB module instance identifier.
bool stopInIdle;	This should be set to true if the USB module must stop operation in IDLE mode
bool suspendInSleep;	This should be set to true if the USB module must suspend when the CPU enters sleep mode.
INT_SOURCE interruptSource;	Specify the interrupt source for the USB module. This should be the interrupt source identifier for the USB module instance specified in usbID.
USB_SPEED operationSpeed;	Specify the operational speed of the USB module. This should always be set to USB_SPEED_FULL.
DRV_USBFS_OPMODES operationMode;	Specify the operation mode of the USB module. This specifies if the USB module should operate as a Device, Host, or both (Dual Role operation).
void * endpointTable;	A pointer to the endpoint descriptor table. This should be aligned at 512 byte address boundary. The size of the table is equal to DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE times the number of endpoints needed in the application.
uint32_t rootHubAvailableCurrent;	Root hub available current in milliamperes. This specifies the amount of current that root hub can provide to the attached device. This should be specified in mA. This is required when the driver is required to operate in host mode.
DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE portPowerEnable;	When operating in Host mode, the application can specify a Root Hub port enable function. This parameter should point to Root Hub port enable function. If this parameter is NULL, it implies that the Port is always enabled.
DRV_USBFS_ROOT_HUB_PORT_INDICATION portIndication;	When operating in Host mode, the application can specify a Root Port Indication. This parameter should point to the Root Port Indication function. If this parameter is NULL, it implies that Root Port Indication is not supported.
DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT portOverCurrentDetect;	When operating is Host mode, the application can specify a Root Port Overcurrent detection. This parameter should point to the Root Port Indication function. If this parameter is NULL, it implies that Overcurrent detection is not supported.

Description

USB Device Driver Initialization Data.

This structure contains all the data necessary to initialize the USB Driver. A pointer to a structure of this type, containing the desired initialization data, must be passed into the DRV_USBFS_Initialize function.

Remarks

None.

DRV_USBFS_OPMODES Enumeration

Identifies the operating modes supported by the USB Driver.

File

drv_usbfs.h

C

```
typedef enum {
  DRV_USBFS_OPMODE_DUAL_ROLE = DRV_USB_OPMODE_DUAL_ROLE,
  DRV_USBFS_OPMODE_DEVICE = DRV_USB_OPMODE_DEVICE,
  DRV_USBFS_OPMODE_HOST = DRV_USB_OPMODE_HOST,
  DRV_USBFS_OPMODE_OTG = DRV_USB_OPMODE_OTG
} DRV_USBFS_OPMODES;
```

Members

Members	Description
DRV_USBFS_OPMODE_DUAL_ROLE = DRV_USB_OPMODE_DUAL_ROLE	The driver should be able to switch between host and device mode
DRV_USBFS_OPMODE_DEVICE = DRV_USB_OPMODE_DEVICE	The driver should support device mode operation only
DRV_USBFS_OPMODE_HOST = DRV_USB_OPMODE_HOST	The driver should support host mode operation only
DRV_USBFS_OPMODE_OTG = DRV_USB_OPMODE_OTG	The driver should support the OTG protocol

Description

USB Operating Modes Enumeration.

This enumeration identifies the operating modes supported by the USB Driver.

Remarks

None.

DRV_USBFS_ROOT_HUB_PORT_INDICATION Type

USB Root hub Application Hooks (Port Indication).

File

drv_usbfs.h

C

```
typedef void (* DRV_USBFS_ROOT_HUB_PORT_INDICATION)(uint8_t port, USB_HUB_PORT_INDICATOR_COLOR color,
USB_HUB_PORT_INDICATOR_STATE state);
```

Description

USB Root hub Application Hooks (Port Indication).

A function of the type defined here should be provided to the driver root to implement Port Indication. The root hub driver calls this function when it needs to update the state of the port indication LEDs. The application can choose to implement the Amber and Green colors as one LED or two different LEDs. The root hub driver specifies the color and the indicator attribute (on, off or blinking) when it calls this function.

Remarks

None.

DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT Type

USB Root hub Application Hooks (Port Overcurrent detection).

File

drv_usbfs.h

C

```
typedef bool (* DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT)(uint8_t port);
```

Description

USB Root hub Application Hooks (Port Overcurrent detection).

A function of the type defined here should be provided to the driver root hub to check for port over current condition. This function will be called periodically by the root hub driver to check the Overcurrent status of the port. It should continue to return true while the Overcurrent condition exists on the port. It should return false when the Overcurrent condition does not exist.

Remarks

None.

DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE Type

USB Root hub Application Hooks (Port Power Enable/ Disable).

File

drv_usbfs.h

C

```
typedef void (* DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE)(uint8_t port, bool control);
```

Description

USB Root hub Application Hooks (Port Power Enable/ Disable).

A function of the type defined here should be provided to the driver root to control port power. The root hub driver will call this function when it needs to enable port power. If the application circuit contains a VBUS switch, the switch should be accessed and controlled by this function. If the enable parameter is true, the switch should be enabled and VBUS should be available on the port. If the enable parameter is false, the switch should be disabled and VBUS should not be available on the port.

Remarks

None.

DRV_USBFS_DEVICE_INTERFACE Macro

USB Driver Device Mode Interface Functions.

File

drv_usbfs.h

C

```
#define DRV_USBFS_DEVICE_INTERFACE
```

Description

USB Driver Device Mode Interface Functions.

The Device Driver interface in the Device Layer Initialization data structure should be set to this value so that Device Layer can access the USB Driver Device Mode functions.

Remarks

None.

DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE Macro

USB Driver Endpoint Table Entry Size in bytes.

File

drv_usbfs.h

C

```
#define DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE 32
```

Description

USB Driver Endpoint Table Entry Size in bytes.

This constant defines the size (in bytes) of an entry in the endpoint table.

Remarks

None.

DRV_USBFS_HOST_INTERFACE Macro

USB Driver Host Mode Interface Functions.

File

drv_usbfs.h

C

```
#define DRV_USBFS_HOST_INTERFACE
```

Description

USB Driver Host Mode Interface Functions.

The Host Controller Driver interface in the Host Layer Initialization data structure should be set to this value so that Host Layer can access the USB Driver Host Mode functions.

Remarks

None.

DRV_USBFS_HOST_PIPE_HANDLE_INVALID Macro

Value of an Invalid Host Pipe Handle.

File

drv_usbfs.h

C

```
#define DRV_USBFS_HOST_PIPE_HANDLE_INVALID ((DRV_USBFS_HOST_PIPE_HANDLE)(-1))
```

Description

USB Driver Invalid Host Pipe Handle.

This constant defines the value of an Invalid Host Pipe Handle.

Remarks

None.

DRV_USBFS_INDEX_0 Macro

USB Driver Module Index 0 Definition.

File

drv_usbfs.h

C

```
#define DRV_USBFS_INDEX_0 0
```

Description

USB Driver Module Index 0 Definition.

This constant defines the value of USB Driver Index 0. The SYS_MODULE_INDEX parameter of the DRV_USBFS_Initialize and DRV_USBFS_Open functions should be set to this value to identify instance 0 of the driver.

Remarks

These constants should be used in place of hard-coded numeric literals and should be passed into the DRV_USBFS_Initialize and DRV_USBFS_Open functions to identify the driver instance in use. These are not indicative of the number of modules that are actually supported by the microcontroller.

DRV_USBFS_INDEX_1 Macro

USB Driver Module Index 1 Definition.

File

drv_usbfs.h

C

```
#define DRV_USBFS_INDEX_1 1
```

Description

USB Driver Module Index 1 Definition.

This constant defines the value of USB Driver Index 1. The SYS_MODULE_INDEX parameter of the DRV_USBFS_Initialize and DRV_USBFS_Open functions should be set to this value to identify instance 1 of the driver.

Remarks

These constants should be used in place of hard-coded numeric literals and should be passed into the DRV_USBFS_Initialize and DRV_USBFS_Open functions to identify the driver instance in use. These are not indicative of the number of modules that are actually supported by the microcontroller.

Files

Files

Name	Description
drv_usbfs.h	PIC32MX USB Module Driver Interface File.
drv_usbfs_config_template.h	USB Full Speed (USBFS) Driver Configuration Template.

Description

drv_usbfs.h

PIC32MX USB Module Driver Interface File.

Enumerations

	Name	Description
	DRV_USBFS_EVENT	Identifies the different events that the USB Driver provides.
	DRV_USBFS_OPMODES	Identifies the operating modes supported by the USB Driver.

Functions

	Name	Description
	DRV_USBFS_ClientEventCallBackSet	This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events.
	DRV_USBFS_Close	Closes an opened-instance of the USB Driver.
	DRV_USBFS_DEVICE_AddressSet	This function will set the USB module address that is obtained from the Host.
	DRV_USBFS_DEVICE_Attach	This function will enable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that a device has been attached on the bus.
	DRV_USBFS_DEVICE_CurrentSpeedGet	This function returns the USB speed at which the device is operating.
	DRV_USBFS_DEVICE_Detach	This function will disable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that the device has detached from the bus.
	DRV_USBFS_DEVICE_EndpointDisable	This function disables an endpoint.
	DRV_USBFS_DEVICE_EndpointDisableAll	This function disables all provisioned endpoints.
	DRV_USBFS_DEVICE_EndpointEnable	This function enables an endpoint for the specified direction and endpoint size.
	DRV_USBFS_DEVICE_EndpointIsEnabled	This function returns the enable/disable status of the specified endpoint and direction.
	DRV_USBFS_DEVICE_EndpointIsStalled	This function returns the stall status of the specified endpoint and direction.
	DRV_USBFS_DEVICE_EndpointStall	This function stalls an endpoint in the specified direction.
	DRV_USBFS_DEVICE_EndpointStallClear	This function clears the stall on an endpoint in the specified direction.
	DRV_USBFS_DEVICE_IRPCancel	This function cancels the specific IRP that are queued and in progress at the specified endpoint.
	DRV_USBFS_DEVICE_IRPCancelAll	This function cancels all IRPs that are queued and in progress at the specified endpoint.
	DRV_USBFS_DEVICE_IRPSubmit	This function submits an I/O Request Packet (IRP) for processing to the Hi-Speed USB Driver.
	DRV_USBFS_DEVICE_RemoteWakeupStart	This function causes the device to start Remote Wakeup Signalling on the bus.
	DRV_USBFS_DEVICE_RemoteWakeupStop	This function causes the device to stop the Remote Wakeup Signalling on the bus.
	DRV_USBFS_DEVICE_SOFNumberGet	This function will return the USB SOF packet number.

	DRV_USBFS_HOST_EventsDisable	Disables Host mode events.
	DRV_USBFS_HOST_EventsEnable	Restores the events to the specified the original value.
	DRV_USBFS_HOST_IRPCancel	Cancels the specified IRP.
	DRV_USBFS_HOST_IRPSubmit	Submits an IRP on a pipe.
	DRV_USBFS_HOST_PipeClose	Closes an open pipe.
	DRV_USBFS_HOST_PipeSetup	Open a pipe with the specified attributes.
	DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet	This function returns the operating speed of the bus to which this root hub is connected.
	DRV_USBFS_HOST_ROOT_HUB_Initialize	This function initializes the root hub driver.
	DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet	Returns the maximum amount of current that this root hub can provide on the bus.
	DRV_USBFS_HOST_ROOT_HUB_OperationEnable	This function enables or disables root hub operation.
	DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled	Returns the operation enabled status of the root hub.
	DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet	Returns the number of ports this root hub contains.
	DRV_USBFS_HOST_ROOT_HUB_PortReset	Resets the specified root hub port.
	DRV_USBFS_HOST_ROOT_HUB_PortResetIsComplete	Returns true if the root hub has completed the port reset operation.
	DRV_USBFS_HOST_ROOT_HUB_PortResume	Resumes the specified root hub port.
	DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet	Returns the speed of at which the port is operating.
	DRV_USBFS_HOST_ROOT_HUB_PortSuspend	Suspends the specified root hub port.
	DRV_USBFS_Initialize	Initializes the USB Driver.
	DRV_USBFS_Open	Opens the specified USB Driver instance and returns a handle to it.
	DRV_USBFS_Status	Provides the current status of the USB Driver module.
	DRV_USBFS_Tasks	Maintains the driver's state machine when the driver is configured for Polled mode.
	DRV_USBFS_Tasks_ISR	Maintains the driver's Interrupt state machine and implements its ISR.

Macros

	Name	Description
	DRV_USBFS_DEVICE_INTERFACE	USB Driver Device Mode Interface Functions.
	DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE	USB Driver Endpoint Table Entry Size in bytes.
	DRV_USBFS_HOST_INTERFACE	USB Driver Host Mode Interface Functions.
	DRV_USBFS_HOST_PIPE_HANDLE_INVALID	Value of an Invalid Host Pipe Handle.
	DRV_USBFS_INDEX_0	USB Driver Module Index 0 Definition.
	DRV_USBFS_INDEX_1	USB Driver Module Index 1 Definition.

Structures

	Name	Description
	DRV_USBFS_INIT	This type definition defines the Driver Initialization Data Structure.

Types

	Name	Description
	DRV_USBFS_EVENT_CALLBACK	Type of the USB Driver event callback function.
	DRV_USBFS_HOST_PIPE_HANDLE	Defines the USB Driver Host Pipe Handle type.
	DRV_USBFS_ROOT_HUB_PORT_INDICATION	USB Root hub Application Hooks (Port Indication).
	DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT	USB Root hub Application Hooks (Port Overcurrent detection).
	DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE	USB Root hub Application Hooks (Port Power Enable/ Disable).

Description

PIC32MX USB Module Driver Interface Header File.

The PIC32MX Full speed USB Module driver provides a simple interface to manage the "USB" peripheral on PIC32MX microcontrollers. This file defines the interface definitions and prototypes for the USB driver. The driver interface meets the requirements of the MPLAB Harmony USB Host and Device Layer.

File Name

drv_usbfs.h

Company

Microchip Technology Inc.

drv_usbfs_config_template.h

USB Full Speed (USBFS) Driver Configuration Template.

Macros

	Name	Description
	DRV_USBFS_DEVICE_SUPPORT	Determines if the USB Device Functionality should be enabled.
	DRV_USBFS_ENDPOINTS_NUMBER	Configures the number of endpoints to be provisioned in the driver.
	DRV_USBFS_HOST_ATTACH_DEBOUNCE_DURATION	Configures the time duration (in milliseconds) that the driver will wait to re-confirm a device attach.
	DRV_USBFS_HOST_NAK_LIMIT	Configures the NAK Limit for Host Mode Control Transfers.
	DRV_USBFS_HOST_PIPES_NUMBER	Configures the maximum number of pipes that are can be opened when the driver is operating in Host mode.
	DRV_USBFS_HOST_RESET_DURATION	Configures the time duration (in milliseconds) of the Reset Signal.
	DRV_USBFS_HOST_SUPPORT	Determines if the USB Host Functionality should be enabled.
	DRV_USBFS_INSTANCES_NUMBER	Specifies the number of driver instances to be enabled in the application.
	DRV_USBFS_INTERRUPT_MODE	Configures the driver for interrupt or polling mode operation.

Description

USB Full Speed Driver Configuration Template.

This file lists all the configurations constants that affect the operation of the USBFS Driver.

File Name

drv_usbfs_config_template.h

Company

Microchip Technology Inc.

PIC32MZ USB Driver

Provides information on the USB Driver specific to PIC32MZ devices.

Description

The PIC32MZ USB Driver in MPLAB Harmony provides API functions that allow the MPLAB Harmony USB Host and Device Stack to access the USB while operating on a PIC32MZ microcontroller. The driver implements the USB Driver Common Interface required by the USB Host and Device Stack. It abstracts the USB module operational details from the Host and Device Stack and provides the stacks with a modular access mechanism to the USB. The PIC32MZ USB Driver features the following:

- USB 2.0 High Speed and Full Speed operation in Peripheral mode
- USB 2.0 High Speed, Full Speed and Low Speed USB Peripheral Support in Host mode
- Designed for Dual Role Operation
- Capable of operating multiple USB modules
- Features non-blocking function and is interoperable with other MPLAB Harmony modules
- Features thread safe functions when operating within an RTOS
- Capable of operating in Polled and Interrupt modes
- Implements the USB Driver Common Interface required by the MPLAB Harmony USB Host and Device Stack
- Completely configurable through the MPLAB Harmony Configurator (MHC)
- Implements feature separation (Host and Device mode functions are implemented across different files)
- Designed to use the module's built-in DMA controller and transfer scheduler

Note: This help section only discusses features that are unique to the PIC32MZ USB Driver and are not a part of the USB Driver Common Interface. The driver functions that implement the USB Driver Common Interface are described in the Common Interface Help section.

While the PIC32MZ USB module supports USB "On-The-Go" (OTG), the PIC32MZ Driver does not currently implement USB OTG protocol support.

This help section only provides relevant information about the operation of the USB. The reader is encouraged to refer to the USB 2.0 Specification available at www.usb.org for a detailed explanation of USB protocol.

Using the Library

This topic describes the basic architecture of the USB PIC32MZ Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_usbhs.h`

The interface to the PIC32MZ USB Driver library is defined in the `drv_usbhs.h` header file.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Overview

Provides an overview of the library.

Description

The PIC32MZ USB Driver will typically be used by a USB Host and/or Device Stack. The USB Host and Device Stack operate as driver client applications. The driver is initialized as part of the MPLAB Harmony System Initialization. The driver initialization data structure specifies the operation mode (Host, Device, or Dual Role) of the driver. The driver features task routines to be called in the MPLAB Harmony application tasks function (SYS_Tasks function) and the USB Module Interrupt Service Routine (ISR).

The Host and the Device Stack can open the driver only when initialization has completed. It will continue to return an invalid driver handle while the initialization is in progress. Once opened, the Device Mode function can be called if the driver is operating in Device mode. The Host Mode function can be called if the driver is operating in Host mode. In Dual Role operation mode, the driver supports Host and Device operation in the same application. Even then, the driver will either operate as a USB Host or Device. OTG operation is not supported.

The PIC32MZ USB Driver features RTOS thread-safe functions. This allows the driver client application to safely call driver functions across different RTOS threads. Not all of the driver functions are interrupt-safe.

In addition to the USB Driver, which implements the USB Driver Common Interface, the PIC32MZ USB Driver implements functions which are required for its operation in the MPLAB Harmony framework. The following table lists the different categories of functions in the PIC32MZ USB Driver.

Library Interface Section	Description
System Function	These functions are accessed by the MPLAB Harmony System module. They allow the driver to be initialized, deinitialized and maintained. These functions are implemented in the `drv_usbhs.c` source file.
Client Core Functions	These functions allow the USB Host and Device Stack to open, close and perform other general driver operations. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbhs.c` source file.
Device Mode Operation Functions	These functions allow the USB Device Stack to perform USB Device mode specific driver operations. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbhs_device.c` source file
Host Mode Operation Functions	These functions allow the USB Host Stack to perform USB Host mode specific driver operations. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbhs_host.c` source file.
Root Hub Functions	These functions allow the USB Host Stack to access the driver Root hub operation. These functions are a part of the USB Driver Common Interface and are implemented in `drv_usbhs_host.c` source file.

Abstraction Model

Provides information on the abstraction model for the library.

Description

The PIC32MZ USB Driver implements the abstraction model defined by the USB Driver Common interface. This interface abstracts USB module specific details and provides a module independent interface to the driver client applications.

While operating in Device mode, the driver expects the client application (the USB Device Stack) to enable endpoints and then submit I/O request packet (IRP) requests to the enabled endpoints. Multiple IRPs can be queued on an endpoint. The driver calls the IRP callback function when the IRP is processed. The driver allows the client application to also attach and detach the device on the bus. It generates events which indicate USB states.

While operating in Host mode, the driver expects the client application (the USB Host Stack) to open pipes to endpoints on the connected device. The client application can then submit IRPs to the pipes. Multiple IRPs can be queued on a pipe. The driver calls the IRP callback function when the IRP is processed. The driver will call application defined functions to enumerate and denumerate a device. These functions are called when the driver detect device attach and detach respectively. The driver also exports root hub functions to the client application. This allows the client application to treat the driver as a single port hub

Please refer to the PIC32 USB Driver Common Interface help section for more details on the driver abstraction model.

How the Library Works

Provides information on how the library works.

Description

This section only explains aspects of driver operation which are unique to the PIC32MZ USB Driver. Major driver operations are described in the PIC32 USB Driver Common Interface help section.

Driver Initialization

Note: While generating a MPLAB Harmony USB project with MHC, the initialization code for the driver is generated automatically based on selections made in the USB Host stack or Device Stack Configuration trees.

The PIC32MZ USB Driver must be initialized so that a client application can open. The client application will not be able to open the driver if the initialization is in progress or has failed. The driver is initialized by calling the DRV_USBHS_Initialize function. This function is called from the SYS_Initialize function in the MPLAB Harmony application project and accepts two input parameters. The `index` parameter defines the instance of the USB Driver to be initialized. This becomes significant when the PIC32MZ microcontroller has more than one USB module. The `init` parameter is a driver-specific data structure of the type DRV_USBHS_INIT. This structure is shown in the following code example.

```
/* This code show the PIC32MZ USB Driver Initialization data structure.
 * A structure of this type must be provided to the DRV_USBHS_Initialize
 * function. */

typedef struct
{
    /* System Module Initialization */
    SYS_MODULE_INIT moduleInit;

    /* Identifies the USB peripheral to be used. This should be the USB PLIB
       module instance identifier. */
    uint8_t usbID;

    /* This should be set to true if the USB module must stop operation in Idle
       mode */
    bool stopInIdle;

    /* This should be set to true if the USB module must suspend when the CPU
       enters Sleep mode. */
    bool suspendInSleep;

    /* Specify the interrupt source for the USB module. This should be Interrupt
       PLIB Interrupt source identifier for the USB module instance specified in
       usbID. */
    INT_SOURCE interruptSource;

    /* Specify the interrupt source for the USB module specific DMA controller.
     * This should be the PLIB Interrupt source identified for the USB
     * module instance specified in usbID. */
    INT_SOURCE interruptSourceUSBDma;

    /* Specify the operational speed of the USB module. This should always be
       set to USB_SPEED_FULL. */
    USB_SPEED operationSpeed;

    /* Specify the operation mode of the USB module. This defines if the USB
     * module will support Device, Host or Dual Role operation */
    DRV_USBHS_OPMODES operationMode;

    /* A pointer to the endpoint descriptor table. This should be aligned at 512
       byte address boundary. The size of the table is equal to the
       DRV_USBHS_ENDPOINT_TABLE_ENTRY_SIZE times the number of endpoints needed
       in the application. */
    void * endpointTable;

    /* Root hub available current in mA. This specifies the amount of current
       that root hub can provide to the attached device. This should be
       specified in mA. This is required when the driver is required to operate
```

```
        in host mode. */
    uint32_t rootHubAvailableCurrent;

    /* When operating in Host mode, the application can specify a Root Hub port
       enable function. This parameter should point to Root Hub port enable
       function. If this parameter is NULL, it implies that the Port is always
       enabled. */
    DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE portPowerEnable;

    /* When operating in Host mode, the application can specify a Root Port
       Indication. This parameter should point to the Root Port Indication
       function. If this parameter is NULL, it implies that Root Port Indication
       is not supported. */
    DRV_USBHS_ROOT_HUB_PORT_INDICATION portIndication;

    /* When operating is Host mode, the application can specify a Root Port
       Overcurrent detection. This parameter should point to the Root Port
       Indication function. If this parameter is NULL, it implies that
       Overcurrent detection is not supported. */
    DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT portOverCurrentDetect;

} DRV_USBHS_INIT;
```

The operationMode parameter defines the driver operation mode. This can be set to DRV_USBFS_OPMODE_DEVICE, DRV_USBFS_OPMODE_HOST, or DRV_USBFS_OPMODE_DUAL_ROLE for Device, Host and Dual Role operation, respectively.

The `rootHubAvailableCurrent` parameter should be set to the maximum current that the VBUS power supply can provide on the bus. The driver does not use this information directly. It provides this data to the client application while operating in Host mode.

The `portPowerEnable` parameter must point to a Port Power Enable function. The driver, while operating in Host mode, will call this function to enable the VBUS switch. This function should activate the VBUS switch if the driver calls this function with the enable parameter set to true. It should deactivate the switch if the driver calls this function with the `enable` parameter set to false. This parameter should be set to NULL if such a switch (of the switch control) is not available in the application.

The `portIndication` parameter must point to a Port Indication function. The driver, while operating in Host mode, will call this function to indicate the current state of the port. The driver will call this function with LED color status as defined in Chapter 11 of the USB 2.0 Specification. This parameter should be set to NULL if such a LED indication is not available in the application.

The `portOverCurrentDetect` parameter must point to a Port Overcurrent Detect function. The driver, while operating in Host mode, will call this function periodically to check if the attached device is overdrawing current. If the function should return true if such a condition exists. This parameter should be set to NULL if such detection is not available in the application.

The following code example shows initialization of the driver for Device mode operation.

```
/* This code shows an example of DRV_USBHS_INIT data structure for
 * Device mode operation. Here the driver is initialized to work with USB0 USB
 * module. */

DRV_USBHS_INIT init;
SYS_MODULE_OBJ usbDriverObj;

const DRV_USBHS_INIT drvUSBInit =
{
    /* Interrupt Source for USB module */
    .interruptSource = INT_SOURCE_USB_1,

    /* DMA Interrupt Source for USB module */
    .interruptSourceUSBDma = INT_SOURCE_USB_1_DMA,

    /* System module initialization */
    .moduleInit = {SYS_MODULE_POWER_RUN_FULL},

    /* Module operate in device mode */
    .operationMode = DRV_USBHS_OPMODE_DEVICE,

    /* Module operated at high speed */
    .operationSpeed = USB_SPEED_HIGH,

    /* Stop in idle */
    .stopInIdle = false,

    /* Suspend in sleep */
    .suspendInSleep = false,
```

```
    /* Identifies peripheral (PLIB-level) ID */
    .usbID = USBHS_ID_0
};

void SYS_Initialize(void)
{
    /* Initialize the USB Driver. Note how the init parameter is typecast to
     * SYS_MODULE_INIT type. The SYS_MODULE_OBJ returned by this function call
     * is passed to the driver tasks routine. DRV_USBHS_INDEX_0 is helper
     * constant defined in drv_usbfs.h */

    usbDriverObj = DRV_USBHS_Initialize(DRV_USBHS_INDEX_0, (SYS_MODULE_INIT *)(drvUSBInit));
}

void SYS_Tasks(void)
{
    /* The polled state of the USB driver is updated by calling the
     * DRV_USBHS_Tasks function in the SYS_Tasks() function. The
     * DRV_USBHS_Tasks() takes the driver module object returned by the
     * DRV_USBHS_Initialize funciton as a parameter. */

    DRV_USBHS_Tasks(usbDriverObj);
}

void __ISR(_USB_VECTOR, ipl4AUTO) _IntHandlerUSBInstance0(void)
{
    /* The DRV_USBHS_Tasks_ISR function update the interrupt state of the USB
     * Driver. If the driver is configured for Polling mode, this function need
     * not be invoked or included in the project. */

    DRV_USBHS_Tasks_ISR(usbDriverObj);
}

void __ISR ( _USB_DMA_VECTOR,ipl4AUTO) _IntHandlerUSBInstance0_USBDMA ( void )
{
    DRV_USBHS_Tasks_ISR_USBDMA(usbDriverObj);
}
```

The following code example shows initialization of the driver for Host mode operation.

```
/* This code shows an example of how the Hi-Speed USB (USBHS) driver can be configured
 * for Host mode operation. In this example, the
 * BSP_USBVBUSSwitchOverCurrentDetect function checks for over current condition
 * and the BSP_USBVBUSPowerEnable function enables the VBUS power. The port
 * indication function is not implemented and hence the portIndication member of
 * the initialization data structure is set to NULL. */

/* The implementation of the port over current detect, indication and the VBUS
 * power supply functions is discussed later in this help section. */

DRV_USBHS_INIT drvUSBHSInit =
{
    /* This should always be set to SYS_MODULE_POWER_RUN_FULL. */
    .moduleInit = {SYS_MODULE_POWER_RUN_FULL},

    /* Interrupt Source for the USB module */
    .interruptSource = INT_SOURCE_USB_1,

    /* Interrupt Source for the USB DMA module */
    .interruptSourceUSBDma = INT_SOURCE_USB_1_DMA,

    /* Configure for host mode operation. */
    .operationMode = DRV_USBHS_OPMODE_HOST,

    /* The driver should run at high speed. */
    .operationSpeed = USB_SPEED_HIGH,

    /* Port indication function is not implemented and is not available */
    .portIndication = NULL,
```

```
    /* This is the VBUS Power enable function */
    .portPowerEnable = BSP_USBVBUSPowerEnable,

    /* This is the over current detect function. */
    .portOverCurrentDetect = BSP_USBVBUSSwitchOverCurrentDetect,

    /* Here we state that the VBUS power supply can provide at most 500 mA of
     * current */
    .rootHubAvailableCurrent = 500,

    /* Moudule will operate in IDLE. */
    .stopInIdle = false,

    /* Module will not suspend automatically in sleep */
    .suspendInSleep = false,

    /* USB Module ID is 1 */
    .usbID = USBHS_ID_0

};

void SYS_Initialize(void)
{
    /* Initialize the USB Driver. Note how the init parameter is typecast to
     * SYS_MODULE_INIT type. The SYS_MODULE_OBJ returned by this function call
     * is passed to the driver tasks routine. DRV_USBHS_INDEX_0 is helper
     * constant defined in drv_usbfs.h */

    usbDriverObj = DRV_USBHS_Initialize(DRV_USBHS_INDEX_0, (SYS_MODULE_INIT *)(drvUSBInit));
}

void SYS_Tasks(void)
{
    /* The polled state of the USB driver is updated by calling the
     * DRV_USBHS_Tasks function in the SYS_Tasks() function. The
     * DRV_USBHS_Tasks takes the driver module object returned by the
     * DRV_USBHS_Initialize funciton as a parameter. */

    DRV_USBHS_Tasks(usbDriverObj);
}

void __ISR( _USB_VECTOR , IPL4AUTO)_IntHandler_USB_stub ( void )
{
    /* The DRV_USBHS_Tasks_ISR function updates the interrupt state of the USB
     * Driver. If the driver is configured for polling mode, this function need
     * not be invoked or included in the project. */

    DRV_USBHS_Tasks_ISR(usbDriverObj);
}

void __ISR ( _USB_DMA_VECTOR, IPL4AUTO)  _IntHandlerUSBInstance0_USBDMA ( void )
{
    /* The DRV_USBHS_Tasks_ISR_USBDMA function update the DMA transfer state of
     * the USB Driver. */

     DRV_USBHS_Tasks_ISR_USBDMA(usbDriverObj);
}
```

The PIC32MX USB Driver requires definition of configuration constants to be available in the `system_config.h` file of the MPLAB Harmony Application Project Configuration. Refer to the Configuring the Library section for details.

Multi-client Operation

The PIC32MZ USB Driver supports multi-client operation. In that, it can be opened by two application clients. This is required where Dual Operation is desired. The following should be noted when using multi-client operation:

- The driver should be initialized for Dual Role Operation mode.
- The DRV_USBHS_Open function can be called at the most twice in the application. The driver supports a maximum of two clients.
- A client can access either the host or device functionality of the driver. It cannot do both.
- It is possible for the two clients to operate in two different threads while operating with an RTOS.

Note: The typical the application clients for PIC32MZ USB Driver would be the MPLAB Harmony USB Host and Device Stack. The complexity of operating the driver in Dual Role mode is handled by the stack operation. The MHC will configure the driver for Dual Role operation when such operation is selected in USB Stack configuration tree.

USB Driver Common Interface

The PIC32MZ USB Driver exports its implementation of the USB Driver Common Interface to the Host and Device Layer via the DRV_USBHS_HOST_INTERFACE and DRV_USBHS_DEVICE_INTERFACE structures. The DRV_USBHS_HOST_INTERFACE structure is defined in the `drv_usbhs_host.c` file. The following code example shows this structure.

```
/*********************************************************
 * This structure is a set of pointer to the USBHS driver
 * functions. It is provided to the host and device layer
 * as the interface to the driver.
 * *******************************************************/

DRV_USB_HOST_INTERFACE gDrvUSBHSHostInterface =
{
    .open = DRV_USBHS_Open,
    .close = DRV_USBHS_Close,
    .eventHandlerSet = DRV_USBHS_ClientEventCallBackSet,
    .hostIRPSubmit = DRV_USBHS_HOST_IRPSubmit,
    .hostIRPCancel = DRV_USBHS_HOST_IRPCancel,
    .hostPipeSetup = DRV_USBHS_HOST_PipeSetup,
    .hostPipeClose = DRV_USBHS_HOST_PipeClose,
    .hostEventsDisable = DRV_USBHS_HOST_EventsDisable,
    .hostEventsEnable = DRV_USBHS_HOST_EventsEnable,
    .rootHubInterface.rootHubPortInterface.hubPortReset = DRV_USBHS_HOST_ROOT_HUB_PortReset,
    .rootHubInterface.rootHubPortInterface.hubPortSpeedGet = DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet,
    .rootHubInterface.rootHubPortInterface.hubPortResetIsComplete =
DRV_USBHS_HOST_ROOT_HUB_PortResetIsComplete,
    .rootHubInterface.rootHubPortInterface.hubPortSuspend = DRV_USBHS_HOST_ROOT_HUB_PortSuspend,
    .rootHubInterface.rootHubPortInterface.hubPortResume = DRV_USBHS_HOST_ROOT_HUB_PortResume,
    .rootHubInterface.rootHubMaxCurrentGet = DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet,
    .rootHubInterface.rootHubPortNumbersGet = DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet,
    .rootHubInterface.rootHubSpeedGet = DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet,
    .rootHubInterface.rootHubInitialize = DRV_USBHS_HOST_ROOT_HUB_Initialize,
    .rootHubInterface.rootHubOperationEnable = DRV_USBHS_HOST_ROOT_HUB_OperationEnable,
    .rootHubInterface.rootHubOperationIsEnabled = DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled,
};
```

The DRV_USBFS_DEVICE_INTERFACE structure is defined in the `drv_usbhs_device.c` file. The following code example shows this structure.

The MPLAB Harmony USB Host and Device stack perform driver independent access through the function pointers contained in these structures.

```
/*****************************************************
 * This structure is a pointer to a set of USB Driver
 * Device mode functions. This set is exported to the
 * device layer when the device layer must use the
 * PIC32MZ USB Controller.
 *****************************************************/

DRV_USB_DEVICE_INTERFACE gDrvUSBHSDeviceInterface =
{
    .open = DRV_USBHS_Open,
    .close = DRV_USBHS_Close,
    .eventHandlerSet = DRV_USBHS_ClientEventCallBackSet,
    .deviceAddressSet = DRV_USBHS_DEVICE_AddressSet,
    .deviceCurrentSpeedGet = DRV_USBHS_DEVICE_CurrentSpeedGet,
    .deviceSOFNumberGet = DRV_USBHS_DEVICE_SOFNumberGet,
    .deviceAttach = DRV_USBHS_DEVICE_Attach,
    .deviceDetach = DRV_USBHS_DEVICE_Detach,
    .deviceEndpointEnable = DRV_USBHS_DEVICE_EndpointEnable,
    .deviceEndpointDisable = DRV_USBHS_DEVICE_EndpointDisable,
    .deviceEndpointStall = DRV_USBHS_DEVICE_EndpointStall,
    .deviceEndpointStallClear = DRV_USBHS_DEVICE_EndpointStallClear,
    .deviceEndpointIsEnabled = DRV_USBHS_DEVICE_EndpointIsEnabled,
    .deviceEndpointIsStalled = DRV_USBHS_DEVICE_EndpointIsStalled,
    .deviceIRPSubmit = DRV_USBHS_DEVICE_IRPSubmit,
    .deviceIRPCancelAll = DRV_USBHS_DEVICE_IRPCancelAll,
    .deviceRemoteWakeupStop = DRV_USBHS_DEVICE_RemoteWakeupStop,
    .deviceRemoteWakeupStart = DRV_USBHS_DEVICE_RemoteWakeupStart,
```

```
    .deviceTestModeEnter = DRV_USBHS_DEVICE_TestModeEnter


};
```

Operation with RTOS

The PIC32MZ USB Driver is designed to operate with a RTOS. The driver implementation uses the MPLAB Harmony Operating System Abstraction Layer (OSAL). This allows the driver to function with entire range of RTOSes supported in MPLAB Harmony. The following points must be considered while using the driver with an RTOS.

- The driver can be opened from different threads
- In Device mode, an enabled endpoint should only be accessed from one thread. For example, if an application requires two endpoints, Endpoint 2 and Endpoint 3, the application could contain two threads, one accessing Endpoint 2 and another accessing Endpoint 3. The thread accessing Endpoint 2 cannot access Endpoint 3.
- While operating in Host mode, endpoint pipes can be opened from different threads. A pipe handle to an open pipe cannot be shared across threads.

USB DMA Operation

The PIC32MZ USB module features a built-in DMA controller. This controller works independently of the PIC32MZ DMA controller. The PIC32MZ USB Driver uses USB DMA controller to expedite transfer of memory from the USB module FIFO to user application memory. The following should be noted for the USB DMA controller:

- If the PIC32MZ USB Driver could not allocate a DMA channel (all channels are busy), it will use the CPU instructions to unload the endpoint FIFOs
- The USB module and the USB DMA controller interrupt priorities should be the same
- The application buffer start address should always be aligned on a 16-byte boundary and should be placed in coherent memory. Refer to the description of the DRV_USBHS_HOST_IRPSubmit and DRV_USBHS_DEVICE_IRPSubmit functions for details on how the user application buffer should be allocated.

Root Hub Operation

The PIC32MZ USB Driver implements a Root Hub Driver Interface. This allows the driver to emulate a hub. The USB Host Stack enumerates the Root Hub as a device. The Host Stack then does not differentiate between an external hub and the root hub. While emulating a hub, the PIC32MZ USB Driver Root Hub appears as a single port hub.

As a part of the Root Hub interface, the PIC32MZ USB Driver requires the application to supply functions for hub features that it does not implement. These features are:

- Port Overcurrent Detect
- VBUS Switch Control
- Port Indication

A pointer to these functions (if implemented) must be supplied through the driver initialization data (of the type DRV_USBHS_INIT) structure at the time of driver initialization. The application has the option of not implementing these functions. In such a case, the function pointers for the unimplemented function, in the initialization data structure should be set to NULL.

The root hub driver must also be able to communicate the maximum current capability of its port to the USB Host Layer. The PIC32MZ USB Controller does not contain built-in (hardware implemented) functionality for controlling the root hub port current. To facilitate this request, the driver will report the current capability that was specified in the `rootHubAvailableCurrent` parameter of the driver initialization data structure. The application must set this parameter to report the current supply capability of the VBUS power supply. The USB Host Layer uses this value to manage the bus current budget. If a connected device reports a configuration that requires more current than what the VBUS power supply can provide, the host will not set the configuration.

Port Overcurrent Detect

The Root Hub operation in PIC32MZ USB Driver will periodically call a Port Overcurrent Detect function to detect if an overcurrent condition is active on the port. The application must supply this function if port overcurrent detection is needed. The PIC32MZ USB Controller does not contain built-in (hardware implemented) functionality for checking overcurrent condition. The overcurrent condition on the port can occur in a case where the attached device has malfunctioned or when the USB VBUS line has short circuited to ground.

The signature of the function and an example implementation is shown in the following code example. The function must return (and must continue to return) true if an overcurrent condition exists on the port.

```
/* This code shows an example implementation of the
 * portOverCurrentDetect function. The PIC32MZ USB Driver will call this
 * function periodically to check if an over current condition exists on the
 * port. In this example, we assume that the over current detect pin from an
 * external circuit in the system, is connected to  port RD0 and the pin logic
 * is active high. The function must return true if an over current condition is
 * present on this pin */


bool BSP_USBVBUSSwitchOverCurrentDetect(uint8_t port)
{
    if(PLIB_PORTS_PinGet(PORTS_ID_0, PORT_CHANNEL_D, 0) == 1)
```

```
    {
        return(true);
    }
    else
    {
        return(false);
    }
}
```

VBUS Switch Control

The PIC32MZ USB Driver Root Hub operation will attempt to control the VBUS power supply to the port. Because the PIC32MZ USB Controller does not contain built-in (hardware implemented) functionality for checking controlling VBUS, such a control function must be supplied by the application. The root hub operation will access this function when the PIC32MX USB Driver will call the portPowerEnable function as a part of the Bus Enable sequence.

The following code shows an example of how this function can be implemented.

```
/* This code shows an example implementation of the VBUS Power Enable
 * function. The PIC32MZ USB Driver will call this function as a part of bus
 * enable function. In this example, it is assumed that system contains an
 * external VBUS power switch and this is control by port RB5.
 */

void BSP_USBVBUSPowerEnable(uint8_t port, bool enable)
{
    if(enable)
    {
        PLIB_PORTS_PinSet(PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_5);
    }
    else
    {
        PLIB_PORTS_PinClear(PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_5);
    }
}
```

Port Indication Function

The Root Hub Operation in the PIC32MZ USB Driver allows display of Port LED status. If the application requires this indication, it must implement a function which the Root Hub operation would call when a change in the Root Hub port has occurred. The port indication operation is specified in Section 11.5.3 of the USB 2.0 Specification.

```
/* This code shows an example implementation of the port indication
 * function. The PIC32MZ USB Driver call this function when it wants to indicate
 * port status. It is assumed that three function to switch off, blink and
 * switch on an LED are available. It is further assumed that these function
 * accept the color of the LED to operated on. */

void BSP_RootHubPortIndication
(
    uint8_t port,
    USB_HUB_PORT_INDICATOR_COLOR color,
    USB_HUB_PORT_INDICATOR_STATE state
)
{
    /* The color parameter indicates the color of the LED to be affected. The
     * color will be either USB_HUB_PORT_INDICATOR_COLOR_GREEN or
     * USB_HUB_PORT_INDICATOR_COLOR_AMBER. */

    switch (state)
    {
        case USB_HUB_PORT_INDICATOR_STATE_OFF:
            BSP_SwitchLEDOff(color);
            break;
        case USB_HUB_PORT_INDICATOR_STATE_BLINKING:
            BSP_LEDBlink(color);
            break;
        case USB_HUB_PORT_INDICATOR_STATE_ON:
            BSP_SwitchLEDOn(color);
            break;
        default:
            break;
    }
```

```
}
```

Configuring the Library

Provides information on the configuring the library.

Macros

	Name	Description
	DRV_USBHS_DEVICE_SUPPORT	Determines if the USB Device Functionality should be enabled.
	DRV_USBHS_ENDPOINTS_NUMBER	Configures the number of endpoints to be provisioned in the driver.
	DRV_USBHS_HOST_ATTACH_DEBOUNCE_DURATION	Configures the time duration (in milliseconds) that the driver will wait to reconfirm a device attach.
	DRV_USBHS_HOST_NAK_LIMIT	Configures the NAK Limit for Host Mode Control Transfers.
	DRV_USBHS_HOST_PIPES_NUMBER	Configures the maximum number of pipes that are can be opened when the driver is operating in Host mode.
	DRV_USBHS_HOST_RESET_DURATION	Configures the time duration (in milliseconds) of the Reset Signal.
	DRV_USBHS_HOST_SUPPORT	Determines if the USB Host Functionality should be enabled.
	DRV_USBHS_INSTANCES_NUMBER	Specifies the number of driver instances to be enabled in the application.
	DRV_USBHS_INTERRUPT_MODE	Configures the driver for interrupt or polling mode operation.

Description

The PIC32MZ USB Driver requires the specification of compile-time configuration macros. These macros define resource usage, feature availability, and dynamic behavior of the driver. These configuration macros should be defined in the `system_config.h` file.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

DRV_USBHS_DEVICE_SUPPORT Macro

Determines if the USB Device Functionality should be enabled.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_DEVICE_SUPPORT true
```

Description

Hi-Speed USB Driver Device Mode Support.

This constant should be set to true if USB device support is required in the application. It should be set to false if device support is not required.

Remarks

This constant should always be defined.

DRV_USBHS_ENDPOINTS_NUMBER Macro

Configures the number of endpoints to be provisioned in the driver.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_ENDPOINTS_NUMBER 3
```

Description

Hi-Speed USB Driver Endpoint Numbers.

This constant configures the number of endpoints that the driver needs to manage. When DRV_USBHS_DEVICE_SUPPORT is enabled, this constant should be set to the total number of endpoints to be enabled in the device. When enabled, an endpoint can be used for communication. Using any direction of an endpoint will require that the entire endpoint to be enabled.

Consider the case of a composite USB Device that contains a CDC and MSD function. The CDC function will require one Bulk endpoint (OUT and IN directions) and one Interrupt endpoint (IN direction). The MSD function will require one Bulk endpoint (IN and OUT directions). This design can

be implemented by using four endpoints. Endpoint 0 is used for the mandatory control interface. Endpoint 1 is used for CDC Bulk interface. Endpoint 2 is used for CDC Interrupt interface and Endpoint 3 is used for MSD Bulk Interface. The constant should then be set to 4.

For Host mode operation, this constant should be set to 1. Setting this value to greater than 1 will result in unused data memory allocation.

Remarks

This constant should always be defined.

DRV_USBHS_HOST_ATTACH_DEBOUNCE_DURATION Macro

Configures the time duration (in milliseconds) that the driver will wait to reconfirm a device attach.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_HOST_ATTACH_DEBOUNCE_DURATION 500
```

Description

Hi-Speed USB Driver Host Mode Attach Debounce Duration.

This constant configures the time duration (in milliseconds) that the driver will wait to reconfirm a device attach. When the driver first detects a device attach, it will start a timer for the duration specified by the constant. When the timer expires, the driver will check if the device is still attached. If so, the driver will then signal an attach event to the host stack. The duration allows for the device attach to become electro-mechanically stable.

Remarks

This constant should always be defined when DRV_USBHS_HOST_SUPPORT is set to true.

DRV_USBHS_HOST_NAK_LIMIT Macro

Configures the NAK Limit for Host Mode Control Transfers.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_HOST_NAK_LIMIT 2000
```

Description

Hi-Speed USB Driver Host Mode Control Transfers NAK Limit.

This constant configures the number of NAKs that the driver can accept from the device in the data stage of a control transfer before aborting the control transfer with a USB_HOST_IRP_STATUS_ERROR_NAK_TIMEOUT. Setting this constant to 0 will disable NAK limit checking. This constant should be adjusted to enable USB host compatibility with USB Devices that require more time to process control transfers.

Remarks

This constant should always be defined when DRV_USBHS_HOST_SUPPORT is set to true.

DRV_USBHS_HOST_PIPES_NUMBER Macro

Configures the maximum number of pipes that are can be opened when the driver is operating in Host mode.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_HOST_PIPES_NUMBER 10
```

Description

Hi-Speed USB Driver Host Mode Pipes Number.

This constant configures the maximum number of pipes that can be opened when the driver is operating in Host mode. Calling the DRV_USBHS_HOST_PipeSetup function will cause a pipe to be opened. Calling this function when DRV_USBHS_HOST_PIPES_NUMBER number of pipes have already been opened will cause the function to return an Invalid Pipe Handle. This constant should be configured to account for the maximum number of devices and the device types to be supported by the host application.

For example, if the USB Host application must support two USB Mass Storage devices and one CDC device. A CDC device requires four pipes and a Mass Storage Device requires three pipes. This constant should therefore be set to a value of 9 (four bulk pipes for two Mass Storage devices + two bulk pipes and one Interrupt pipe for one CDC device and two control pipes for two devices). Allocating pipes consumes data memory.

While enabling support for multiple devices, through a Hub, the application should consider the worst case requirement while configuring this constant. For example, a case where devices with the most number of pipe requirements are connected to the hub. At the same time, setting this constant to more than what is required will consume data memory.

Remarks

This constant should always be defined when DRV_USBHS_HOST_SUPPORT is set to true.

DRV_USBHS_HOST_RESET_DURATION Macro

Configures the time duration (in milliseconds) of the Reset Signal.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_HOST_RESET_DURATION 100
```

Description

Hi-Speed USB Driver Host Mode Reset Duration.

This constant configures the duration of the reset signal. The driver generates a reset signal when the USB Host stack requests for a root hub port reset. The driver will generate the reset signal for the duration specified by this constant and will then stop generating the reset signal.

Remarks

This constant should always be defined when DRV_USBHS_HOST_SUPPORT is set to true.

DRV_USBHS_HOST_SUPPORT Macro

Determines if the USB Host Functionality should be enabled.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_HOST_SUPPORT false
```

Description

Hi-Speed USB Driver Host Mode Support.

This constant should be set to true if USB Host mode support is required in the application. It should be set to false if host support is not required.

Remarks

This constant should always be defined.

DRV_USBHS_INSTANCES_NUMBER Macro

Specifies the number of driver instances to be enabled in the application.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_INSTANCES_NUMBER 1
```

Description

Hi-Speed USB Driver Instances Number.

This constant defines the number of driver instances to be enabled in the application. This will be typically be the number of USB controllers to be used in the application. On PIC32MZ microcontrollers that have one USB controller, this value will always be 1. On PIC32MZ microcontrollers that have two USB controllers, this value could be one or two, depending on whether one or two USB segments are required. To conserve data memory, this constant should be set to exactly the number of USB controllers that are required in the system.

Remarks

This constant should always be defined.

DRV_USBHS_INTERRUPT_MODE Macro

Configures the driver for interrupt or polling mode operation.

File

drv_usbhs_config_template.h

C

```
#define DRV_USBHS_INTERRUPT_MODE true
```

Description

Hi-Speed USB Driver Interrupt Mode.

This constant configures the driver for interrupt or polling operation. If this flag is set to true, the driver will operate in Interrupt mode. If the flag is set to false, the driver will operate in Polled mode. In Polled mode, the driver interrupt state machine gets updated in the SYS_Tasks function. If the driver is configured for Interrupt mode, the driver Interrupt state machine gets updated in the driver Interrupt Service Routine(ISR). It is always recommended for the driver to operate in Interrupt mode.

Remarks

This constant should always be defined.

Building the Library

This section lists the files that are available in the PIC32MZ USB Driver Library.

Description

This section list the files that are available in the `\src` folder of the PIC32MZ USB Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/usb/usbhs`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_usbhs.h`	This file should be included by any `.c` file which accesses the PIC32MZ USB Driver API. This one file contains the prototypes for all driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_usbhs.c`	This file should always be included in the project when using the PIC3MZ USB Driver.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
`/src/dynamic/drv_usbhs_device.c`	This file should be included in the project if Device mode operation is required.
`/src/dynamic/drv_usbhs_host.c`	This file should be included in the project if Host mode operation is required.

Module Dependencies

The PIC32MZ USB Driver Library depends on the following modules:

- Interrupt System Service Library

Library Interface

a) System Functions

	Name	Description
	DRV_USBHS_Initialize	Initializes the Hi-Speed USB Driver.
	DRV_USBHS_Status	Provides the current status of the Hi-Speed USB Driver module.
	DRV_USBHS_Tasks	Maintains the driver's state machine when the driver is configured for Polled mode.
	DRV_USBHS_Tasks_ISR	Maintains the driver's Interrupt state machine and implements its ISR.
	DRV_USBHS_Tasks_ISR_USBDMA	Maintains the driver's DMA Transfer state machine and implements its ISR.

b) Client Core Functions

	Name	Description
	DRV_USBHS_ClientEventCallBackSet	This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events.
	DRV_USBHS_Close	Closes an opened-instance of the Hi-Speed USB Driver.
	DRV_USBHS_Open	Opens the specified Hi-Speed USB Driver instance and returns a handle to it.

c) Device Mode Operation Functions

	Name	Description
	DRV_USBHS_DEVICE_AddressSet	This function will set the USB module address that is obtained from the Host.
	DRV_USBHS_DEVICE_Attach	This function will enable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that a device has been attached on the bus.
	DRV_USBHS_DEVICE_CurrentSpeedGet	This function will return the USB speed at which the device is operating.
	DRV_USBHS_DEVICE_Detach	This function will disable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that the device has detached from the bus.
	DRV_USBHS_DEVICE_EndpointDisable	This function disables an endpoint.
	DRV_USBHS_DEVICE_EndpointDisableAll	This function disables all provisioned endpoints.
	DRV_USBHS_DEVICE_EndpointEnable	This function enables an endpoint for the specified direction and endpoint size.
	DRV_USBHS_DEVICE_EndpointIsEnabled	This function returns the enable/disable status of the specified endpoint and direction.
	DRV_USBHS_DEVICE_EndpointIsStalled	This function returns the stall status of the specified endpoint and direction.
	DRV_USBHS_DEVICE_EndpointStall	This function stalls an endpoint in the specified direction.
	DRV_USBHS_DEVICE_EndpointStallClear	This function clears the stall on an endpoint in the specified direction.
	DRV_USBHS_DEVICE_IRPCancel	This function cancels the specific IRP that are queued and in progress at the specified endpoint.
	DRV_USBHS_DEVICE_IRPCancelAll	This function cancels all IRPs that are queued and in progress at the specified endpoint.
	DRV_USBHS_DEVICE_IRPSubmit	This function submits an I/O Request Packet (IRP) for processing to the Hi-Speed USB Driver.
	DRV_USBHS_DEVICE_RemoteWakeupStart	This function causes the device to start Remote Wakeup Signalling on the bus.
	DRV_USBHS_DEVICE_RemoteWakeupStop	This function causes the device to stop the Remote Wakeup Signalling on the bus.
	DRV_USBHS_DEVICE_SOFNumberGet	This function will return the USB SOF packet number.
	DRV_USBHS_DEVICE_TestModeEnter	This function enables the specified USB 2.0 Test Mode.
	DRV_USBHS_DEVICE_TestModeExit	This function disables the specified USB 2.0 Test Mode.

d) Host Mode Operation Functions

	Name	Description
	DRV_USBHS_HOST_EventsDisable	Disables Host mode events.
	DRV_USBHS_HOST_EventsEnable	Restores the events to the specified the original value.
	DRV_USBHS_HOST_IRPCancel	Cancels the specified IRP.
	DRV_USBHS_HOST_IRPSubmit	Submits an IRP on a pipe.
	DRV_USBHS_HOST_PipeClose	Closes an open pipe.
	DRV_USBHS_HOST_PipeSetup	Open a pipe with the specified attributes.

e) Root Hub Functions

	Name	Description
	DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet	This function returns the operating speed of the bus to which this root hub is connected.
	DRV_USBHS_HOST_ROOT_HUB_Initialize	This function initializes the root hub driver.
	DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet	Returns the maximum amount of current that this root hub can provide on the bus.
	DRV_USBHS_HOST_ROOT_HUB_OperationEnable	This function enables or disables root hub operation.
	DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled	Returns the operation enabled status of the root hub.
	DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet	Returns the number of ports this root hub contains.
	DRV_USBHS_HOST_ROOT_HUB_PortReset	Resets the specified root hub port.
	DRV_USBHS_HOST_ROOT_HUB_PortResetIsComplete	Returns true if the root hub has completed the port reset operation.
	DRV_USBHS_HOST_ROOT_HUB_PortResume	Resumes the specified root hub port.
	DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet	Returns the speed of at which the port is operating.
	DRV_USBHS_HOST_ROOT_HUB_PortSuspend	Suspends the specified root hub port.

f) Data Types and Constants

	Name	Description
	DRV_USBHS_EVENT	Identifies the different events that the Hi-Speed USB Driver provides.
	DRV_USBHS_EVENT_CALLBACK	Type of the Hi-Speed USB Driver event callback function.
	DRV_USBHS_HOST_PIPE_HANDLE	Defines the Hi-Speed USB Driver Host Pipe Handle type.
	DRV_USBHS_INIT	This type definition defines the Driver Initialization Data Structure.
	DRV_USBHS_OPMODES	Identifies the operating modes supported by the Hi-Speed USB Driver.
	DRV_USBHS_ROOT_HUB_PORT_INDICATION	USB Root hub Application Hooks (Port Indication).
	DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT	USB Root hub Application Hooks (Port Overcurrent detection).
	DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE	USB Root hub Application Hooks (Port Power Enable/ Disable).
	DRV_USBHS_DEVICE_INTERFACE	Hi-Speed USB Driver Device Mode Interface Functions.
	DRV_USBHS_HOST_INTERFACE	Hi-Speed USB Driver Host Mode Interface Functions.
	DRV_USBHS_HOST_PIPE_HANDLE_INVALID	Value of an Invalid Host Pipe Handle.
	DRV_USBHS_INDEX_0	Hi-Speed USB Driver Module Index 0 Definition.

Description

a) System Functions

DRV_USBHS_Initialize Function

Initializes the Hi-Speed USB Driver.

File

drv_usbhs.h

C

```
SYS_MODULE_OBJ DRV_USBHS_Initialize(const SYS_MODULE_INDEX drvIndex, const SYS_MODULE_INIT * const init);
```

Returns

- SYS_MODULE_OBJ_INVALID - The driver initialization failed.
- A valid System Module Object - The driver initialization was able to start. It may have not completed and requires the DRV_USBHS_Tasks function to be called periodically. This value will never be the same as SYS_MODULE_OBJ_INVALID.

Description

This function initializes the Hi-Speed USB Driver, making it ready for clients to open. The driver initialization does not complete when this function returns. The DRV_USBHS_Tasks function must called periodically to complete the driver initialization. The DRV_USBHS_Open function will fail if the driver was not initialized or if initialization has not completed.

Remarks

This function must be called before any other Hi-Speed USB Driver function is called. This function should only be called once during system initialization unless DRV_USBHS_Deinitialize is called to deinitialize the driver instance.

Preconditions

None.

Example

```
// The following code shows an example initialization of the
// driver. The USB module to be used is USB1.  The module should not
// automatically suspend when the  microcontroller enters Sleep mode.  The
// module should continue  operation when the module enters Idle mode.  The
// power state is set to run at full clock speeds. Device Mode operation
// should be at FULL speed. The size of the endpoint table is set for two
// endpoints.

DRV_USBHS_INIT moduleInit;

usbInitData.usbID               = USBHS_ID_0;
usbInitData.opMode              = DRV_USBHS_OPMODE_DEVICE;
usbInitData.stopInIdle          = false;
usbInitData.suspendInSleep      = false;
usbInitData.operationSpeed      = USB_SPEED_FULL;
usbInitData.interruptSource     = INT_SOURCE_USB;

usbInitData.sysModuleInit.powerState = SYS_MODULE_POWER_RUN_FULL ;

// This is how this data structure is passed to the initialize
// function.

DRV_USBHS_Initialize(DRV_USBHS_INDEX_0, (SYS_MODULE_INIT *) &usbInitData);
```

Parameters

Parameters	Description
drvIndex	Ordinal number of driver instance to be initialized. This should be set to DRV_USBHS_INDEX_0 if driver instance 0 needs to be initialized.
init	Pointer to a data structure containing data necessary to initialize the driver. This should be a DRV_USBHS_INIT structure reference typecast to SYS_MODULE_INIT reference.

Function

SYS_MODULE_OBJ DRV_USBHS_Initialize
(
const SYS_MODULE_INDEX drvIndex,
const SYS_MODULE_INIT * const init
)

DRV_USBHS_Status Function

Provides the current status of the Hi-Speed USB Driver module.

File

drv_usbhs.h

C

```
SYS_STATUS DRV_USBHS_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that the driver is ready.
- SYS_STATUS_UNINITIALIZED - Indicates that the driver has never been initialized.

Description

This function provides the current status of the Hi-Speed USB Driver module.

Remarks

None.

Preconditions

The DRV_USBHS_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USBHS_Initialize
SYS_STATUS          status;
DRV_USBHS_INIT moduleInit;

usbInitData.usbID               = USBHS_ID_0;
usbInitData.opMode              = DRV_USBHS_OPMODE_DEVICE;
usbInitData.stopInIdle          = false;
usbInitData.suspendInSleep      = false;
usbInitData.operationSpeed      = USB_SPEED_FULL;
usbInitData.interruptSource     = INT_SOURCE_USB;

usbInitData.sysModuleInit.powerState = SYS_MODULE_POWER_RUN_FULL ;

// This is how this data structure is passed to the initialize
// function.

DRV_USBHS_Initialize(DRV_USBHS_INDEX_0, (SYS_MODULE_INIT *) &usbInitData);

// The status of the driver can be checked.
status = DRV_USBHS_Status(object);
if(SYS_STATUS_READY == status)
{
    // Driver is ready to be opened.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_USBHS_Initialize function.

Function

SYS_STATUS DRV_USBHS_Status (SYS_MODULE_OBJ object)

DRV_USBHS_Tasks Function

Maintains the driver's state machine when the driver is configured for Polled mode.

File

drv_usbhs.h

C

```
void DRV_USBHS_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Maintains the driver's Polled state machine. This function should be called from the SYS_Tasks function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks function (SYS_Tasks). This function will never block.

Preconditions

The DRV_USBHS_Initialize function must have been called for the specified Hi-Speed USB Driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USBHS_Initialize
```

```
while (true)
{
    DRV_USBHS_Tasks(object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USBHS_Initialize function).

Function

void DRV_USBHS_Tasks(SYS_MODULE_OBJ object)

DRV_USBHS_Tasks_ISR Function

Maintains the driver's Interrupt state machine and implements its ISR.

File

drv_usbhs.h

C

```
void DRV_USBHS_Tasks_ISR(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal Interrupt state machine and implement its ISR for interrupt-driven implementations.

Remarks

This function should be called from the USB ISR. For multiple USB modules, it should be ensured that the correct Hi-Speed USB Driver system module object is passed to this function.

Preconditions

The DRV_USBHS_Initialize function must have been called for the specified Hi-Speed USB Driver instance.

Example

```
SYS_MODULE_OBJ object;     // Returned from DRV_USBHS_Initialize

while (true)
{
    DRV_USBHS_Tasks_ISR (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USBHS_Initialize).

Function

void DRV_USBHS_Tasks_ISR(SYS_MODULE_OBJ object)

DRV_USBHS_Tasks_ISR_USBDMA Function

Maintains the driver's DMA Transfer state machine and implements its ISR.

File

drv_usbhs.h

C

```
void DRV_USBHS_Tasks_ISR_USBDMA(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal DMA Transfer state machine and implement its ISR for interrupt-driven implementations.

Remarks

This function should be called from the USB DMA ISR. For multiple USB modules, it should be ensured that the correct Hi-Speed USB Driver system module object is passed to this function.

Preconditions

The DRV_USBHS_Initialize function must have been called for the specified Hi-Speed USB Driver instance.

Example

```
SYS_MODULE_OBJ object;     // Returned from DRV_USBHS_Initialize

while (true)
{
    DRV_USBHS_Tasks_ISR_USBDMA (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USBHS_Initialize).

Function

void DRV_USBHS_Tasks_ISR_USBDMA(SYS_MODULE_OBJ object)

b) Client Core Functions

DRV_USBHS_ClientEventCallBackSet Function

This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events.

File

drv_usbhs.h

C

```
void DRV_USBHS_ClientEventCallBackSet(DRV_HANDLE handle, uintptr_t hReferenceData, DRV_USB_EVENT_CALLBACK
myEventCallBack);
```

Returns

None.

Description

This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events. The callback is disabled by either not calling this function after the DRV_USBHS_Open function has been called or by setting the myEventCallBack argument as NULL. When the callback function is called, the hReferenceData argument is returned.

Remarks

Typical usage of the Hi-Speed USB Driver requires a client to register a callback.

Preconditions

None.

Example

```
// Set the client event callback for the Device Layer.  The
// USBDeviceLayerEventHandler function is the event handler. When this
// event handler is invoked by the driver, the driver returns back the
// second argument specified in the following function (which in this case
// is the Device Layer data structure). This allows the application
// firmware to identify, as an example, the Device Layer object associated
// with this callback.

DRV_USBHS_ClientEventCallBackSet(myUSBDevice.usbDriverHandle, (uintptr_t)&myUSBDevice,
USBDeviceLayerEventHandler);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
hReferenceData	Object (could be a pointer) that is returned with the callback.
myEventCallBack	Callback function for all USB events.

Function

void DRV_USBHS_ClientEventCallBackSet

(

DRV_HANDLE handle,

uintptr_t hReferenceData,

DRV_USBHS_EVENT_CALLBACK myEventCallBack

);

DRV_USBHS_Close Function

Closes an opened-instance of the Hi-Speed USB Driver.

File

drv_usbhs.h

C

```
void DRV_USBHS_Close(DRV_HANDLE handle);
```

Returns

None.

Description

This function closes an opened-instance of the Hi-Speed USB Driver, invalidating the handle.

Remarks

After calling this function, the handle passed in handle parameter must not be used with any of the other driver functions. A new handle must be obtained by calling DRV_USBHS_Open function before the caller may use the driver again.

Preconditions

The DRV_USBHS_Initialize function must have been called for the specified Hi-Speed USB Driver instance. DRV_USBHS_Open function must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_USBHS_Open

DRV_USBHS_Close(handle);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

void DRV_USBHS_Close(DRV_HANDLE handle)

DRV_USBHS_Open Function

Opens the specified Hi-Speed USB Driver instance and returns a handle to it.

File

drv_usbhs.h

C

```
DRV_HANDLE DRV_USBHS_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

- DRV_HANDLE_INVALID - The driver could not be opened successfully.This can

happen if the driver initialization was not complete or if an internal error has occurred.

- A Valid Driver Handle - This is an arbitrary value and is returned if the function was successful. This value will never be the same as DRV_HANDLE_INVALID.

Description

This function opens the specified Hi-Speed USB Driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The intent flag should always be DRV_IO_INTENT_EXCLUSIVE|DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NON_BLOCKING. Any other setting of the intent flag will return a invalid driver handle. A driver instance can only support one client. Trying to open a driver that has an existing client will result in an unsuccessful function call.

Remarks

The handle returned is valid until the DRV_USBHS_Close function is called. The function will typically return DRV_HANDLE_INVALID if the driver was not initialized. In such a case the client should try to open the driver again.

Preconditions

Function DRV_USBHS_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

// This code assumes that the driver has been initialized.
handle = DRV_USBHS_Open(DRV_USBHS_INDEX_0, DRV_IO_INTENT_EXCLUSIVE| DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NON_BLOCKING);

if(DRV_HANDLE_INVALID == handle)
{
    // The application should try opening the driver again.
}
```

Parameters

Parameters	Description
drvIndex	Identifies the driver instance to be opened. As an example, this value can be set to DRV_USBHS_INDEX_0 if instance 0 of the driver has to be opened.
intent	Should always be (DRV_IO_INTENT_EXCLUSIVE|DRV_IO_INTENT_READWRITE| DRV_IO_INTENT_NON_BLOCKING).

Function

```
DRV_HANDLE DRV_USBHS_Open
(
const    SYS_MODULE_INDEX drvIndex,
const    DRV_IO_INTENT intent
)
```

c) Device Mode Operation Functions

DRV_USBHS_DEVICE_AddressSet Function

This function will set the USB module address that is obtained from the Host.

File

drv_usbhs.h

C

```
void DRV_USBHS_DEVICE_AddressSet(DRV_HANDLE handle, uint8_t address);
```

Returns

None.

Description

This function will set the USB module address that is obtained from the Host in a setup transaction. The address is obtained from the SET_ADDRESS command issued by the Host. The primary (first) client of the driver uses this function to set the module's USB address after decoding the setup transaction from the Host.

Remarks

None.

Preconditions

None.

Example

```
// This function should be called by the first client of the driver,
// which is typically the Device Layer. The address to set is obtained
// from the Host during enumeration.

DRV_USBHS_DEVICE_AddressSet(deviceLayer, 4);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
address	The address of this module on the USB bus.

Function

void DRV_USBHS_DEVICE_AddressSet(DRV_HANDLE handle, uint8_t address);

DRV_USBHS_DEVICE_Attach Function

This function will enable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that a device has been attached on the bus.

File

drv_usbhs.h

C

```
void DRV_USBHS_DEVICE_Attach(DRV_HANDLE handle);
```

Returns

None.

Description

This function enables the pull-up resistors on the D+ or D- lines thus letting the USB Host know that a device has been attached on the bus . This function should be called when the driver client is ready to receive communication from the Host (typically after all initialization is complete). The USB 2.0 specification requires VBUS to be detected before the data line pull-ups are enabled. The application must ensure the same.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// Open the device driver and attach the device to the USB.
handle = DRV_USBHS_Open(DRV_USBHS_INDEX_0, DRV_IO_INTENT_EXCLUSIVE| DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NON_BLOCKING);

// Register a callback
DRV_USBHS_ClientEventCallBackSet(handle, (uintptr_t)&myDeviceLayer, MyDeviceLayerEventCallback);

// The device can be attached when VBUS Session Valid event occurs
void MyDeviceLayerEventCallback(uintptr_t handle, DRV_USBHS_EVENT event, void * hReferenceData)
{
    switch(event)
    {
        case DRV_USBHS_EVENT_DEVICE_SESSION_VALID:
            // A valid VBUS was detected.
            DRV_USBHS_DEVICE_Attach(handle);
            break;

        case DRV_USBHS_EVENT_DEVICE_SESSION_INVALID:
            // VBUS is not valid anymore. The device can be disconnected.
            DRV_USBHS_DEVICE_Detach(handle);
            break;

        default:
            break;
        }
    }
}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

void DRV_USBHS_DEVICE_Attach(DRV_HANDLE handle);

DRV_USBHS_DEVICE_CurrentSpeedGet Function

This function will return the USB speed at which the device is operating.

File

drv_usbhs.h

C

```
USB_SPEED DRV_USBHS_DEVICE_CurrentSpeedGet(DRV_HANDLE handle);
```

Returns

Returns a member of the USB_SPEED type.

Description

This function will return the USB speed at which the device is operating.

Remarks

None.

Preconditions

Only valid after the device is attached to the Host and Host has completed reset signaling.

Example

```
// Get the current speed.
```

```
USB_SPEED deviceSpeed;

deviceSpeed = DRV_USBHS_DEVICE_CurrentSpeedGet(deviceLayer);

if(deviceLayer == USB_SPEED_HIGH)
{
    // Possibly adjust buffers for higher throughput.
}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

USB_SPEED DRV_USBHS_DEVICE_CurrentSpeedGet(DRV_HANDLE handle);

DRV_USBHS_DEVICE_Detach Function

This function will disable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that the device has detached from the bus.

File

drv_usbhs.h

C

```
void DRV_USBHS_DEVICE_Detach(DRV_HANDLE handle);
```

Returns

None.

Description

This function disables the pull-up resistors on the D+ or D- lines. This function should be called when the application wants to disconnect the device from the bus (typically to implement a soft detach or switch to Host mode operation). A self-powered device should be detached from the bus when the VBUS is not valid.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// Open the device driver and attach the device to the USB.
handle = DRV_USBHS_Open(DRV_USBHS_INDEX_0, DRV_IO_INTENT_EXCLUSIVE| DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NON_BLOCKING);

// Register a callback
DRV_USBHS_ClientEventCallBackSet(handle, (uintptr_t)&myDeviceLayer, MyDeviceLayerEventCallback);

// The device can be detached when VBUS Session Invalid event occurs
void MyDeviceLayerEventCallback(uintptr_t handle, DRV_USBHS_EVENT event, void * hReferenceData)
{
    switch(event)
    {
        case DRV_USBHS_EVENT_DEVICE_SESSION_VALID:
            // A valid VBUS was detected.
            DRV_USBHS_DEVICE_Attach(handle);
            break;

        case DRV_USBHS_EVENT_DEVICE_SESSION_INVALID:
            // VBUS is not valid anymore. The device can be disconnected.
            DRV_USBHS_DEVICE_Detach(handle);
            break;

        default:
```

```
            break;
        }
    }
}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

void DRV_USBHS_DEVICE_Detach(DRV_HANDLE handle);

DRV_USBHS_DEVICE_EndpointDisable Function

This function disables an endpoint.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_EndpointDisable(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - The endpoint that is being accessed is not a valid endpoint (endpoint was not provisioned through the DRV_USBHS_ENDPOINTS_NUMBER configuration constant) defined for this driver instance.

Description

This function disables an endpoint. If the endpoint type is a control endpoint type, both directions are disabled. For non-control endpoints, the function disables the specified direction only. The direction to be disabled is specified by the Most Significant Bit (MSB) of the endpointAndDirection parameter.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to disable
// a control endpoint. Note that the direction parameter is ignored.
// For a control endpoint, both the directions are disabled.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 0);

DRV_USBHS_DEVICE_EndpointDisable(handle, ep );

// This code shows an example of how to disable a BULK IN
// endpoint

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBHS_DEVICE_EndpointDisable(handle, ep );
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

USB_ERROR DRV_USBHS_DEVICE_EndpointDisable
(
DRV_HANDLE handle,
USB_ENDPOINT endpointAndDirection
)

DRV_USBHS_DEVICE_EndpointDisableAll Function

This function disables all provisioned endpoints.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_EndpointDisableAll(DRV_HANDLE handle);
```

Returns

- USB_ERROR_NONE - The function exited successfully.
- USB_ERROR_PARAMETER_INVALID - The driver handle is invalid.

Description

This function disables all provisioned endpoints in both directions.

Remarks

This function is typically called by the USB Device Layer to disable all endpoints upon detecting a bus reset.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to disable all endpoints.

DRV_USBHS_DEVICE_EndpointDisableAll(handle);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

USB_ERROR DRV_USBHS_DEVICE_EndpointDisableAll(DRV_HANDLE handle)

DRV_USBHS_DEVICE_EndpointEnable Function

This function enables an endpoint for the specified direction and endpoint size.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_EndpointEnable(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection,
USB_TRANSFER_TYPE transferType, uint16_t endpointSize);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is not a valid endpoint defined for this driver instance. The value of DRV_USBHS_ENDPOINTS_NUMBER configuration constant should be adjusted.
- USB_ERROR_PARAMETER_INVALID - The driver handle is invalid.

Description

This function enables an endpoint for the specified direction and endpoint size. The function will enable the endpoint for communication in one direction at a time. It must be called twice if the endpoint is required to communicate in both the directions, with the exception of control endpoints. If the endpoint type is a control endpoint, the endpoint is always bidirectional and the function needs to be called only once.

The size of the endpoint must match the wMaxPacketSize reported in the endpoint descriptor for this endpoint. A transfer that is scheduled over this endpoint will be scheduled in wMaxPacketSize transactions. The function does not check if the endpoint is already in use. It is the client's responsibility to make sure that a endpoint is not accidentally reused.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to enable Endpoint
// 0 for control transfers. Note that for a control endpoint, the
// direction parameter is ignored. A control endpoint is always
// bidirectional. Endpoint size is 64 bytes.

uint8_t ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 0);

DRV_USBHS_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_CONTROL, 64);

// This code shows an example of how to set up a endpoint
// for BULK IN transfer. For an IN transfer, data moves from device
// to Host. In this example, Endpoint 1 is enabled. The maximum
// packet size is 64.

uint8_t ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBHS_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_BULK, 64);

// If Endpoint 1 must also be set up for BULK OUT, the
// DRV_USBHS_DEVICE_EndpointEnable function must be called again, as shown
// here.

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_HOST_TO_DEVICE, 1);

DRV_USBHS_DEVICE_EndpointEnable(handle, ep, USB_TRANSFER_TYPE_BULK, 64);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.
transferType	Should be USB_TRANSFER_TYPE_CONTROL for control endpoint, USB_TRANSFER_TYPE_BULK for bulk endpoint, USB_TRANSFER_TYPE_INTERRUPT for interrupt endpoint and USB_TRANSFER_TYPE_ISOCHRONOUS for isochronous endpoint.
endpointSize	Maximum size (in bytes) of the endpoint as reported in the endpoint descriptor.

Function

```
USB_ERROR DRV_USBHS_DEVICE_EndpointEnable
(
    DRV_HANDLE handle,
    USB_ENDPOINT endpointAndDirection,
USB_TRANSFER_TYPE transferType,
uint16_t endpointSize
);
```

DRV_USBHS_DEVICE_EndpointIsEnabled Function

This function returns the enable/disable status of the specified endpoint and direction.

File

drv_usbhs.h

C

```
bool DRV_USBHS_DEVICE_EndpointIsEnabled(DRV_HANDLE client, USB_ENDPOINT endpointAndDirection);
```

Returns

- true - The endpoint is enabled.
- false - The endpoint is disabled.

Description

This function returns the enable/disable status of the specified endpoint and direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how the
// DRV_USBHS_DEVICE_EndpointIsEnabled function can be used to obtain the
// status of Endpoint 1 and IN direction.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

if(DRV_USBHS_ENDPOINT_STATE_DISABLED ==
            DRV_USBHS_DEVICE_EndpointIsEnabled(handle, ep))
{
    // Endpoint is disabled. Enable endpoint.

    DRV_USBHS_DEVICE_EndpointEnable(handle, ep, USB_ENDPOINT_TYPE_BULK, 64);

}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

bool DRV_USBHS_DEVICE_EndpointIsEnabled

(

DRV_HANDLE handle,

USB_ENDPOINT endpointAndDirection

)

DRV_USBHS_DEVICE_EndpointIsStalled Function

This function returns the stall status of the specified endpoint and direction.

File

drv_usbhs.h

C

```
bool DRV_USBHS_DEVICE_EndpointIsStalled(DRV_HANDLE client, USB_ENDPOINT endpoint);
```

Returns

- true - The endpoint is stalled.
- false - The endpoint is not stalled.

Description

This function returns the stall status of the specified endpoint and direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how the
// DRV_USBHS_DEVICE_EndpointIsStalled function can be used to obtain the
// stall status of Endpoint 1 and IN direction.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

if(true == DRV_USBHS_DEVICE_EndpointIsStalled (handle, ep))
{
    // Endpoint stall is enabled. Clear the stall.

    DRV_USBHS_DEVICE_EndpointStallClear(handle, ep);

}
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

bool DRV_USBHS_DEVICE_EndpointIsStalled

(

DRV_HANDLE handle,

USB_ENDPOINT endpointAndDirection

)

DRV_USBHS_DEVICE_EndpointStall Function

This function stalls an endpoint in the specified direction.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_EndpointStall(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

- USB_ERROR_OSAL_FUNCTION - An error with an OSAL function called in this function.

Description

This function stalls an endpoint in the specified direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to stall an endpoint. In
// this example, Endpoint 1 IN direction is stalled.

USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBHS_DEVICE_EndpointStall(handle, ep);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

```
USB_ERROR DRV_USBHS_DEVICE_EndpointStall
(
    DRV_HANDLE handle,
    USB_ENDPOINT endpointAndDirection
)
```

DRV_USBHS_DEVICE_EndpointStallClear Function

This function clears the stall on an endpoint in the specified direction.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_EndpointStallClear(DRV_HANDLE handle, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.

Description

This function clears the stall on an endpoint in the specified direction.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to clear a stall. In this
// example, the stall condition on Endpoint 1 IN direction is cleared.
```

```
USB_ENDPOINT ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

DRV_USBHS_DEVICE_EndpointStallClear(handle, ep);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

```
  USB_ERROR DRV_USBHS_DEVICE_EndpointStallClear
(
    DRV_HANDLE handle,
    USB_ENDPOINT endpointAndDirection
)
```

DRV_USBHS_DEVICE_IRPCancel Function

This function cancels the specific IRP that are queued and in progress at the specified endpoint.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_IRPCancel(DRV_HANDLE client, USB_DEVICE_IRP * irp);
```

Returns

- USB_ERROR_NONE - The IRP have been canceled successfully.
- USB_ERROR_PARAMETER_INVALID - Invalid parameter or the IRP already has been aborted or completed
- USB_ERROR_OSAL_FUNCTION - An OSAL function called in this function did not execute successfully.

Description

This function attempts to cancel the processing of a queued IRP. An IRP that was in the queue but yet to be processed will be cancelled successfully and the IRP callback function will be called from this function with the USB_DEVICE_IRP_STATUS_ABORTED status. The application can release the data buffer memory used by the IRP when this callback occurs. If the IRP was in progress (a transaction in on the bus) when the cancel function was called, the IRP will be canceled only when an ongoing or the next transaction has completed. The IRP callback function will then be called in an interrupt context. The application should not release the related data buffer unless the IRP callback has occurred.

Remarks

The size returned after the ABORT callback will be always 0 regardless of the amount of data that has been sent or received. The client should not assume any data transaction has happened for an canceled IRP. If the last transaction of the IRP was in progress, the IRP cancel does not have any effect. The first transaction of any ongoing IRP cannot be canceled.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to cancel IRP.  In this example the IRP
// has been scheduled from a device to the Host.

USB_ENDPOINT ep;
USB_DEVICE_IRP irp;

ep.direction = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

irp.data = myDataBufferToSend;
irp.size = 130;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
```

```
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

if (DRV_USBHS_DEVICE_IRPSubmit(handle, ep, &irp) != USB_ERROR_NONE)
{
    // This means there was an error.
}
else
{
    // Check the status of the IRP.
    if(irp.status != USB_DEVICE_IRP_STATUS_COMPLETED)
    {
        // Cancel the submitted IRP.
        if (DRV_USBHS_DEVICE_IRPCancel(handle, &irp) != USB_ERROR_NONE)
        {
            // The IRP Cancel request submission was successful.
            // IRP cancel status will be notified through the callback
            // function.
        }
        else
        {
            // The IRP may have been completed before IRP cancel operation.
            // could start. No callback notification will be generated.
        }
    }
    else
    {
        // The IRP processing must have been completed before IRP cancel was
        // submitted.
    }
}

void MyIRPCallback(USB_DEVICE_IRP * irp)
{
    // Check if the IRP callback is for a Cancel request
    if(irp->status == USB_DEVICE_IRP_STATUS_ABORTED)
    {
        // IRP cancel completed
    }
 }
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
irp	Pointer to the IRP to cancel.

Function

USB_ERROR DRV_USBHS_DEVICE_IRPCancel
(
DRV_HANDLE client,
USB_DEVICE_IRP * irp
)

DRV_USBHS_DEVICE_IRPCancelAll Function

This function cancels all IRPs that are queued and in progress at the specified endpoint.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_IRPCancelAll(DRV_HANDLE client, USB_ENDPOINT endpointAndDirection);
```

Returns

- USB_ERROR_NONE - The endpoint was successfully enabled.

- USB_ERROR_DEVICE_ENDPOINT_INVALID - If the endpoint that is being accessed is out of the valid endpoint defined for this driver instance.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid.
- USB_ERROR_OSAL_FUNCTION - An OSAL function called in this function did not execute successfully.

Description

This function cancels all IRPs that are queued and in progress at the specified endpoint.

Remarks

None.

Preconditions

The Client handle should be valid.

Example

```
// This code shows an example of how to cancel all IRPs.

void MyIRPCallback(USB_DEVICE_IRP * irp)
{
    // Check if this is setup command

    if(irp->status == USB_DEVICE_IRP_STATUS_SETUP)
    {
        if(IsSetupCommandSupported(irp->data) == false)
        {
            // This means that this setup command is not
            // supported. Stall the some related endpoint and cancel all
            // queue IRPs.

            DRV_USBHS_DEVICE_EndpointStall(handle, ep);
            DRV_USBHS_DEVICE_IRPCancelAll(handle, ep);
        }
     }
 }
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.

Function

```
USB_ERROR DRV_USBHS_DEVICE_IRPCancelAll
(
    DRV_HANDLE client,
    USB_ENDPOINT endpointAndDirection
);
```

DRV_USBHS_DEVICE_IRPSubmit Function

This function submits an I/O Request Packet (IRP) for processing to the Hi-Speed USB Driver.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_IRPSubmit(DRV_HANDLE client, USB_ENDPOINT endpointAndDirection, USB_DEVICE_IRP *
irp);
```

Returns

- USB_ERROR_NONE - if the IRP was submitted successful.
- USB_ERROR_IRP_SIZE_INVALID - if the size parameter of the IRP is not correct.
- USB_ERROR_PARAMETER_INVALID - If the client handle is not valid.

- USB_ERROR_ENDPOINT_NOT_CONFIGURED - If the endpoint is not enabled.
- USB_ERROR_DEVICE_ENDPOINT_INVALID - The specified endpoint is not valid.
- USB_ERROR_OSAL_FUNCTION - An OSAL call in the function did not complete successfully.

Description

This function submits an I/O Request Packet (IRP) for processing to the USB Driver. The IRP allows a client to send and receive data from the USB Host. The data will be sent or received through the specified endpoint. The direction of the data transfer is indicated by the direction flag in the endpointAndDirection parameter. Submitting an IRP arms the endpoint to either send data to or receive data from the Host. If an IRP is already being processed on the endpoint, the subsequent IRP submit operation will be queued. The contents of the IRP (including the application buffers) should not be changed until the IRP has been processed.

Particular attention should be paid to the size parameter of IRP. The following should be noted:

- The size parameter while sending data to the Host can be less than, greater than, equal to, or be an exact multiple of the maximum packet size for the endpoint. The maximum packet size for the endpoint determines the number of transactions required to process the IRP.
- If the size parameter, while sending data to the Host is less than the maximum packet size, the transfer will complete in one transaction.
- If the size parameter, while sending data to the Host is greater than the maximum packet size, the IRP will be processed in multiple transactions.
- If the size parameter, while sending data to the Host is equal to or an exact multiple of the maximum packet size, the client can optionally ask the driver to send a Zero Length Packet(ZLP) by specifying the USB_DEVICE_IRP_FLAG_DATA_COMPLETE flag as the flag parameter.
- The size parameter, while receiving data from the Host must be an exact multiple of the maximum packet size of the endpoint. If this is not the case, the driver will return a USB_ERROR_IRP_SIZE_INVALID result. If while processing the IRP, the driver receives less than maximum packet size or a ZLP from the Host, the driver considers the IRP as processed. The size parameter at this point contains the actual amount of data received from the Host. The IRP status is returned as USB_DEVICE_IRP_STATUS_COMPLETED_SHORT.
- If a ZLP needs to be sent to Host, the IRP size should be specified as 0 and the flag parameter should be set as USB_DEVICE_IRP_FLAG_DATA_COMPLETE.
- If the IRP size is an exact multiple of the endpoint size, the client can request the driver to not send a ZLP by setting the flag parameter to USB_DEVICE_IRP_FLAG_DATA_PENDING. This flag indicates that there is more data pending in this transfer.
- Specifying a size less than the endpoint size along with the USB_DEVICE_IRP_FLAG_DATA_PENDING flag will cause the driver to return a USB_ERROR_IRP_SIZE_INVALID.
- If the size is greater than but not a multiple of the endpoint size, and the flag is specified as USB_DEVICE_IRP_FLAG_DATA_PENDING, the driver will send multiple of endpoint size number of bytes. For example, if the IRP size is 130 and the endpoint size if 64, the number of bytes sent will 128.

Remarks

This function can be called from the ISR of the USB module to associated with the client.

Preconditions

The Client handle should be valid.

Example

```
// The following code shows an example of how to schedule a IRP to send data
// from a device to the Host. Assume that the max packet size is 64 and
// and this data needs to sent over Endpoint 1. In this example, the
// transfer is processed as three transactions of 64, 64 and 2 bytes.

USB_ENDPOINT ep;
USB_DEVICE_IRP irp;

ep.direction = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

irp.data = myDataBufferToSend;
irp.size = 130;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

if (DRV_USBHS_DEVICE_IRPSubmit(handle, ep, &irp) != USB_ERROR_NONE)
{
    // This means there was an error.
}
else
{
    // The status of the IRP can be checked.
    while(irp.status != USB_DEVICE_IRP_STATUS_COMPLETED)
    {
        // Wait or run a task function.
    }
```

```
}

// The following code shows how the client can request
// the driver to send a ZLP when the size is an exact multiple of
// endpoint size.

irp.data = myDataBufferToSend;
irp.size = 128;
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;

// Note that while  receiving data from the Host, the size should be an
// exact multiple of the maximum packet size of the endpoint. In the
// following example, the DRV_USBHS_DEVICE_IRPSubmit function will return a
// USB_DEVICE_IRP_SIZE_INVALID value.

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_HOST_TO_DEVICE, 1);

irp.data = myDataBufferToSend;
irp.size = 60; // THIS SIZE IS NOT CORRECT
irp.flags = USB_DEVICE_IRP_FLAG_DATA_COMPLETE;
irp.callback = MyIRPCompletionCallback;
irp.referenceData = (uintptr_t)&myDeviceLayerObj;
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).
endpointAndDirection	Specifies the endpoint and direction.
irp	Pointer to the IRP to be added to the queue for processing.

Function

```
USB_ERROR DRV_USBHS_DEVICE_IRPSubmit
(
    DRV_HANDLE client,
    USB_ENDPOINT endpointAndDirection,
    USB_DEVICE_IRP * irp
);
```

DRV_USBHS_DEVICE_RemoteWakeupStart Function

This function causes the device to start Remote Wakeup Signalling on the bus.

File

drv_usbhs.h

C

```
void DRV_USBHS_DEVICE_RemoteWakeupStart(DRV_HANDLE handle);
```

Returns

None.

Description

This function causes the device to start Remote Wakeup Signalling on the bus. This function should be called when the device, presently placed in suspend mode by the Host, wants to be wakeup. Note that the device can do this only when the Host has enabled the device's Remote Wakeup capability.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
DRV_HANDLE handle;

// If the Host has enabled the Remote Wakeup capability, and if the device
// is in suspend mode, then start Remote Wakeup signaling.

if(deviceIsSuspended && deviceRemoteWakeupEnabled)
{
    DRV_USBHS_DEVICE_RemoteWakeupStart(handle);
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).

Function

void DRV_USBHS_DEVICE_RemoteWakeupStart(DRV_HANDLE handle);

DRV_USBHS_DEVICE_RemoteWakeupStop Function

This function causes the device to stop the Remote Wakeup Signalling on the bus.

File

drv_usbhs.h

C

```
void DRV_USBHS_DEVICE_RemoteWakeupStop(DRV_HANDLE handle);
```

Returns

None.

Description

This function causes the device to stop Remote Wakeup Signalling on the bus. This function should be called after the DRV_USBHS_DEVICE_RemoteWakeupStart function was called to start the Remote Wakeup signaling on the bus.

Remarks

This function should be 1 to 15 milliseconds after the DRV_USBHS_DEVICE_RemoteWakeupStart function was called.

Preconditions

The handle should be valid. The DRV_USBHS_DEVICE_RemoteWakeupStart function was called to start the Remote Wakeup signaling on the bus.

Example

```
DRV_HANDLE handle;

// If the Host has enabled the Remote Wakeup capability, and if the device
// is in suspend mode, then start Remote Wakeup signaling. Wait for 10
// milliseconds and then stop the Remote Wakeup signaling

if(deviceIsSuspended && deviceRemoteWakeupEnabled)
{
    DRV_USBHS_DEVICE_RemoteWakeupStart(handle);
    DelayMilliSeconds(10);
    DRV_USBHS_DEVICE_RemoteWakeupStop(handle);
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).

Function

void DRV_USBHS_DEVICE_RemoteWakeupStop(DRV_HANDLE handle);

DRV_USBHS_DEVICE_SOFNumberGet Function

This function will return the USB SOF packet number.

File

drv_usbhs.h

C

```
uint16_t DRV_USBHS_DEVICE_SOFNumberGet(DRV_HANDLE handle);
```

Returns

The SOF packet number.

Description

This function will return the USB SOF packet number..

Remarks

None.

Preconditions

This function will return a valid value only when the device is attached to the bus. The SOF packet count will not increment if the bus is suspended.

Example

```
// This code shows how the DRV_USBHS_DEVICE_SOFNumberGet function is called
// to read the current SOF number.

DRV_HANDLE handle;
uint16_t sofNumber;

sofNumber = DRV_USBHS_DEVICE_SOFNumberGet(handle);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

uint16_t DRV_USBHS_DEVICE_SOFNumberGet(DRV_HANDLE handle);

DRV_USBHS_DEVICE_TestModeEnter Function

This function enables the specified USB 2.0 Test Mode.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_TestModeEnter(DRV_HANDLE handle, USB_TEST_MODE_SELECTORS testMode);
```

Returns

- USB_ERROR_NONE - The function executed successfully.
- USB_ERROR_PARAMETER_INVALID - The handle or the value of testMode parameter is not valid.

Description

This function causes the device to enter the specified USB 2.0 defined test mode. It is called in response to Set Feature command from the host. The wValue field of this command specifies the Test Mode to enter. The USB module will perform the action identified by the testMode parameter.

Remarks

This function should be called only when the USB device has attached to the Host at High speed and only in response to the Set Feature

command from the Host.

Preconditions

The handle should be valid.

Example

```
DRV_HANDLE handle;

// This code shows how the DRV_USBHS_DEVICE_TestModeEnter function is
// called to make the USB device enter the Test_J test mode.

DRV_USBHS_DEVICE_TestModeEnter(handle, USB_TEST_MODE_SELECTOR_TEST_J);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
testMode	This parameter identifies the USB 2.0 specification test mode (see table 9-7 of the USB 2.0 specification).

Function

```
USB_ERROR DRV_USBHS_DEVICE_TestModeEnter
(
    DRV_HANDLE handle,
USB_TEST_MODE_SELECTORS testMode
);
```

DRV_USBHS_DEVICE_TestModeExit Function

This function disables the specified USB 2.0 Test Mode.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_DEVICE_TestModeExit(DRV_HANDLE handle, USB_TEST_MODE_SELECTORS testMode);
```

Returns

- USB_ERROR_NONE - The function executed successfully.
- USB_ERROR_PARAMETER_INVALID - The handle or the value of testMode parameter is not valid.

Description

This function causes the device to stop the specified USB 2.0 defined test mode. This function can be called after calling the DRV_USBHS_DEVICE_TestModeEnter function to stop the test mode execution.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
DRV_HANDLE handle;

// This code shows how the DRV_USBHS_DEVICE_TestModeEnter function is
// called to make the USB device enter the Test_J test mode.

DRV_USBHS_DEVICE_TestModeEnter(handle, USB_TEST_MODE_SELECTOR_TEST_J);

// Now the DRV_USBHS_DEVICE_TestModeExit function is called to stop the
// Test_J test mode.

DRV_USBHS_DEVICE_TestModeExit(handle, USB_TEST_MODE_SELECTOR_TEST_J);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
testMode	This parameter identifies the USB 2.0 specification test mode (see table 9-7 of the USB 2.0 specification).

Function

```
  USB_ERROR DRV_USBHS_DEVICE_TestModeExit
(
    DRV_HANDLE handle,
USB_TEST_MODE_SELECTORS testMode
);
```

d) Host Mode Operation Functions

DRV_USBHS_HOST_EventsDisable Function

Disables Host mode events.

File

drv_usbhs.h

C

```
bool DRV_USBHS_HOST_EventsDisable(DRV_HANDLE handle);
```

Returns

- true - Driver event generation was enabled when this function was called.
- false - Driver event generation was not enabled when this function was called.

Description

This function disables the Host mode events. This function is called by the Host Layer when it wants to execute code atomically.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
// This code shows how the DRV_USBHS_HOST_EventsDisable and
// DRV_USBHS_HOST_EventsEnable function can be called to disable and enable
// events.

DRV_HANDLE driverHandle;
bool eventsWereEnabled;

// Disable the driver events.
eventsWereEnabled = DRV_USBHS_HOST_EventsDisable(driverHandle);

// Code in this region will not be interrupted by driver events.

// Enable the driver events.
DRV_USBHS_HOST_EventsEnable(driverHandle, eventsWereEnabled);
```

Parameters

Parameters	Description
handle	Client's driver handle (returned from DRV_USBHS_Open function).

Function

bool DRV_USBHS_HOST_EventsDisable
(
DRV_HANDLE handle
);

DRV_USBHS_HOST_EventsEnable Function

Restores the events to the specified the original value.

File

drv_usbhs.h

C

```
void DRV_USBHS_HOST_EventsEnable(DRV_HANDLE handle, bool eventContext);
```

Returns

None.

Description

This function will restore the enable disable state of the events. The eventRestoreContext parameter should be equal to the value returned by the DRV_USBHS_HOST_EventsDisable function.

Remarks

None.

Preconditions

The handle should be valid.

Example

```
// This code shows how the DRV_USBHS_HOST_EventsDisable and
// DRV_USBHS_HOST_EventsEnable function can be called to disable and enable
// events.

DRV_HANDLE driverHandle;
bool eventsWereEnabled;

// Disable the driver events.
eventsWereEnabled = DRV_USBHS_HOST_EventsDisable(driverHandle);

// Code in this region will not be interrupted by driver events.

// Enable the driver events.
DRV_USBHS_HOST_EventsEnable(driverHandle, eventsWereEnabled);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
eventRestoreContext	Value returned by the DRV_USBHS_HOST_EventsDisable function.

Function

void DRV_USBHS_HOST_EventsEnable
(
DRV_HANDLE handle
bool eventRestoreContext
);

DRV_USBHS_HOST_IRPCancel Function

Cancels the specified IRP.

File

drv_usbhs.h

C

```
void DRV_USBHS_HOST_IRPCancel(USB_HOST_IRP * inputIRP);
```

Returns

None.

Description

This function attempts to cancel the specified IRP. If the IRP is queued and its processing has not started, it will be cancelled successfully. If the IRP in progress, the ongoing transaction will be allowed to complete.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how a submitted IRP can be cancelled.

USB_HOST_IRP irp;
USB_ERROR result;
USB_HOST_PIPE_HANDLE controlPipe;
USB_SETUP_PACKET setup;
uint8_t controlTransferData[32];

irp.setup = setup;
irp.data = controlTransferData;
irp.size = 32;
irp.flags = USB_HOST_IRP_FLAG_NONE ;
irp.userData = &someApplicationObject;
irp.callback = IRP_Callback;

DRV_USBHS_HOST_IRPSubmit(controlPipeHandle, &irp);

// Additional application logic may come here. This logic may decide to
// cancel the submitted IRP.

DRV_USBHS_HOST_IRPCancel(&irp);
```

Parameters

Parameters	Description
inputIRP	Pointer to the IRP to cancel.

Function

void DRV_USBHS_HOST_IRPCancel(USB_HOST_IRP * inputIRP);

DRV_USBHS_HOST_IRPSubmit Function

Submits an IRP on a pipe.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_HOST_IRPSubmit(DRV_USBHS_HOST_PIPE_HANDLE hPipe, USB_HOST_IRP * pinputIRP);
```

Returns

- USB_ERROR_NONE - The IRP was submitted successfully.
- USB_ERROR_PARAMETER_INVALID - The pipe handle is not valid.
- USB_ERROR_OSAL_FUNCTION - An error occurred in an OSAL function called in this function.

Description

This function submits an IRP on the specified pipe. The IRP will be added to the queue and will be processed in turn. The data will be transferred on the bus based on the USB bus scheduling rules. When the IRP has been processed, the callback function specified in the IRP will be called. The IRP status will be updated to reflect the completion status of the IRP.

Remarks

An IRP can also be submitted in an IRP callback function.

Preconditions

The pipe handle should be valid.

Example

```
// The following code shows an example of how the host layer populates
// the IRP object and then submits it. IRP_Callback function is called when an
// IRP has completed processing. The status of the IRP at completion can be
// checked in the status flag. The size field of the irp will contain the amount
// of data transferred.

void IRP_Callback(USB_HOST_IRP * irp)
{
    // irp is pointing to the IRP for which the callback has occurred. In most
    // cases this function will execute in an interrupt context. The application
    // should not perform any hardware access or interrupt un-safe operations in
    // this function.

    switch(irp->status)
    {
        case USB_HOST_IRP_STATUS_ERROR_UNKNOWN:
            // IRP was terminated due to an unknown error
            break;

        case USB_HOST_IRP_STATUS_ABORTED:
            // IRP was terminated by the application
            break;

        case USB_HOST_IRP_STATUS_ERROR_BUS:
            // IRP was terminated due to a bus error
            break;

        case USB_HOST_IRP_STATUS_ERROR_DATA:
            // IRP was terminated due to data error
            break;

        case USB_HOST_IRP_STATUS_ERROR_NAK_TIMEOUT:
            // IRP was terminated because of a NAK timeout
            break;

        case USB_HOST_IRP_STATUS_ERROR_STALL:
            // IRP was terminated because of a device sent a STALL
            break;

        case USB_HOST_IRP_STATUS_COMPLETED:
            // IRP has been completed
            break;

        case USB_HOST_IRP_STATUS_COMPLETED_SHORT:
            // IRP has been completed but the amount of data processed was less
            // than requested.
            break;

        default:
```

```
            break;
        }
    }

    // In the following code snippet the a control transfer IRP is submitted to a
    // control pipe. The setup parameter of the IRP points to the Setup command of
    // the control transfer. The direction of the data stage is specified by the
    // Setup packet.

    USB_HOST_IRP irp;
    USB_ERROR result;
    USB_HOST_PIPE_HANDLE controlPipe;
    USB_SETUP_PACKET setup;
    uint8_t controlTransferData[32];

    irp.setup = setup;
    irp.data = controlTransferData;
    irp.size = 32;
    irp.flags = USB_HOST_IRP_FLAG_NONE ;
    irp.userData = &someApplicationObject;
    irp.callback = IRP_Callback;

    result = DRV_USBHS_HOST_IRPSubmit(controlPipeHandle, &irp);
```

Parameters

Parameters	Description
hPipe	Handle to the pipe to which the IRP has to be submitted.
pInputIRP	Pointer to the IRP.

Function

```
  USB_ERROR DRV_USBHS_HOST_IRPSubmit
(
    DRV_USBHS_HOST_PIPE_HANDLE  hPipe,
    USB_HOST_IRP * pInputIRP
);
```

DRV_USBHS_HOST_PipeClose Function

Closes an open pipe.

File

drv_usbhs.h

C

```
void DRV_USBHS_HOST_PipeClose(DRV_USBHS_HOST_PIPE_HANDLE pipeHandle);
```

Returns

None.

Description

This function closes an open pipe. Any IRPs scheduled on the pipe will be aborted and IRP callback functions will be called with the status as DRV_USB_HOST_IRP_STATE_ABORTED. The pipe handle will become invalid and the pipe and will not accept IRPs.

Remarks

None.

Preconditions

The pipe handle should be valid.

Example

```
// This code shows how an open Host pipe can be closed.

DRV_HANDLE driverHandle;
```

```
DRV_USBHS_HOST_PIPE_HANDLE pipeHandle;

// Close the pipe.
DRV_USBHS_HOST_PipeClose(pipeHandle);
```

Parameters

Parameters	Description
pipeHandle	Handle to the pipe to close.

Function

void DRV_USBHS_HOST_PipeClose

(

DRV_USBHS_HOST_PIPE_HANDLE pipeHandle

);

DRV_USBHS_HOST_PipeSetup Function

Open a pipe with the specified attributes.

File

drv_usbhs.h

C

```
DRV_USBHS_HOST_PIPE_HANDLE DRV_USBHS_HOST_PipeSetup(DRV_HANDLE client, uint8_t deviceAddress, USB_ENDPOINT
endpointAndDirection, uint8_t hubAddress, uint8_t hubPort, USB_TRANSFER_TYPE pipeType, uint8_t bInterval,
uint16_t wMaxPacketSize, USB_SPEED speed);
```

Returns

- DRV_USB_HOST_PIPE_HANDLE_INVALID - The pipe could not be created.
- A valid Pipe Handle - The pipe was created successfully. This is an arbitrary value and will never be the same as DRV_USB_HOST_PIPE_HANDLE_INVALID.

Description

This function opens a communication pipe between the Host and the device endpoint. The transfer type and other attributes are specified through the function parameters. The driver does not check for available bus bandwidth, which should be done by the application (the USB Host Layer in this case)

Remarks

None.

Preconditions

The driver handle should be valid.

Example

```
// This code shows how the DRV_USBHS_HOST_PipeSetup function is called for
// create a communication pipe. In this example, Bulk pipe is created
// between the Host and a device. The Device address is 2 and the target
// endpoint on this device is 4 . The direction of the data transfer over
// this pipe is from the Host to the device. The device is connected to Port
// 1 of a Hub, whose USB address is 3. The maximum size of a transaction
// on this pipe is 64 bytes. This is a Bulk Pipe and hence the bInterval
// field is set to 0. The target device is operating at Full Speed.

DRV_HANDLE driverHandle;
DRV_USBHS_HOST_PIPE_HANDLE pipeHandle;

pipeHandle = DRV_USBHS_HOST_PipeSetup(driverHandle, 0x02, 0x14, 0x03, 0x01, USB_TRANSFER_TYPE_BULK, 0, 64,
USB_SPEED_FULL);

if(pipeHandle != DRV_USBHS_HOST_PIPE_HANDLE_INVALID)
{
    // The pipe was created successfully.
}
```

Parameters

Parameters	Description
client	Handle to the driver (returned from DRV_USBHS_Open function).
deviceAddress	USB Address of the device to connect to.
endpoint	Endpoint on the device to connect to.
hubAddress	Address of the hub to which this device is connected. If not connected to a hub, this value should be set to 0.
hubPort	Port number of the hub to which this device is connected.
pipeType	Transfer type of the pipe to open.
bInterval	Polling interval for periodic transfers. This should be specified as defined by the USB 2.0 Specification.
wMaxPacketSize	This should be set to the endpoint size reported by the device in its configuration descriptors. This defines the maximum size of the transaction in a transfer on this pipe.
speed	The speed of the pipe. This should match the speed at which the device connected to the Host.

Function

```
    DRV_USBHS_HOST_PIPE_HANDLE DRV_USBHS_HOST_PipeSetup
(
        DRV_HANDLE client,
    uint8_t deviceAddress,
        USB_ENDPOINT endpointAndDirection,
    uint8_t hubAddress,
    uint8_t hubPort,
    USB_TRANSFER_TYPE pipeType,
    uint8_t bInterval,
    uint16_t wMaxPacketSize,
        USB_SPEED speed
);
```

e) Root Hub Functions

DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet Function

This function returns the operating speed of the bus to which this root hub is connected.

File

drv_usbhs.h

C

```
USB_SPEED DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet(DRV_HANDLE handle);
```

Returns

- USB_SPEED_HIGH - The Root hub is connected to a bus that is operating at High Speed.
- USB_SPEED_FULL - The Root hub is connected to a bus that is operating at Full Speed.

Description

This function returns the operating speed of the bus to which this root hub is connected.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet function is
// called to know the operating speed of the bus to which this Root hub is
// connected.

DRV_HANDLE driverHandle;
USB_SPEED speed;

speed = DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet(driverHandle);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).

Function

USB_SPEED DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet(DRV_HANDLE handle);

DRV_USBHS_HOST_ROOT_HUB_Initialize Function

This function initializes the root hub driver.

File

drv_usbhs.h

C

```
void DRV_USBHS_HOST_ROOT_HUB_Initialize(DRV_HANDLE handle, USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo);
```

Returns

None.

Description

This function initializes the root hub driver. It is called by the Host Layer at the time of processing the root hub devices. The Host Layer assigns a USB_HOST_DEVICE_INFO reference to this root hub driver. This identifies the relationship between the root hub and the Host Layer.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the USB Host Layer calls the
// DRV_USBHS_HOST_ROOT_HUB_Initialize function. The usbHostDeviceInfo
// parameter is an arbitrary identifier assigned by the USB Host Layer. Its
// interpretation is opaque to the Root hub Driver.

DRV_HANDLE drvHandle;
USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo = 0x10003000;

DRV_USBHS_HOST_ROOT_HUB_Initialize(drvHandle, usbHostDeviceInfo);
```

Parameters

Parameters	Description
handle	Handle to the driver.
usbHostDeviceInfo	Reference provided by the Host.

Function

void DRV_USBHS_HOST_ROOT_HUB_Initialize

(

DRV_HANDLE handle,

USB_HOST_DEVICE_OBJ_HANDLE usbHostDeviceInfo,
)

DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet Function

Returns the maximum amount of current that this root hub can provide on the bus.

File

drv_usbhs.h

C

```
uint32_t DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet(DRV_HANDLE handle);
```

Returns

Returns the maximum current (in milliamperes) that the root hub can supply.

Description

This function returns the maximum amount of current that this root hub can provide on the bus.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet
// function is called to obtain the maximum VBUS current that the Root hub
// can supply.

DRV_HANDLE driverHandle;
uint32_t currentMilliAmperes;

currentMilliAmperes = DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet(driverHandle);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).

Function

uint32_t DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet(DRV_HANDLE);

DRV_USBHS_HOST_ROOT_HUB_OperationEnable Function

This function enables or disables root hub operation.

File

drv_usbhs.h

C

```
void DRV_USBHS_HOST_ROOT_HUB_OperationEnable(DRV_HANDLE handle, bool enable);
```

Returns

None.

Description

This function enables or disables root hub operation. When enabled, the root hub will detect devices attached to the port and will request the Host Layer to enumerate the device. This function is called by the Host Layer when it is ready to receive enumeration requests from the Host. If the operation is disabled, the root hub will not detect attached devices.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_OperationEnable and the
// DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled functions are called to enable
// the Root hub operation.

DRV_HANDLE driverHandle;

// Enable Root hub operation.
DRV_USBHS_HOST_ROOT_HUB_OperationEnable(driverHandle);

// Wait till the Root hub operation is enabled.
if(DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled(driverHandle) == false)
{
    // The operation has not completed. Call the
    // DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled function again to check if
    // the operation has completed. Note that the DRV_USBHS_Tasks function
    // must be allowed to run at periodic intervals to allow the enable
    // operation to completed.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
enable	If this is set to true, root hub operation is enabled. If this is set to false, root hub operation is disabled.

Function

void DRV_USBHS_HOST_ROOT_HUB_OperationEnable

(

DRV_HANDLE handle,

bool enable

);

DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled Function

Returns the operation enabled status of the root hub.

File

drv_usbhs.h

C

```
bool DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled(DRV_HANDLE handle);
```

Returns

- true - Root hub operation is enabled.
- false - Root hub operation is not enabled.

Description

This function returns true if the DRV_USBHS_HOST_ROOT_HUB_OperationEnable function has completed enabling the Host.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_OperationEnable and the
// DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled functions are called to enable
```

```
// the Root hub operation.

DRV_HANDLE driverHandle;

// Enable Root hub operation.
DRV_USBHS_HOST_ROOT_HUB_OperationEnable(driverHandle);

// Wait till the Root hub operation is enabled.
if(DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled(driverHandle) == false)
{
    // The operation has not completed. Call the
    // DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled function again to check if
    // the operation has completed. Note that the DRV_USBHS_Tasks function
    // must be allowed to run at periodic intervals to allow the enable
    // operation to completed.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).

Function

bool DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled(DRV_HANDLE handle);

DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet Function

Returns the number of ports this root hub contains.

File

drv_usbhs.h

C

```
uint8_t DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet(DRV_HANDLE handle);
```

Returns

This function will always return 1.

Description

This function returns the number of ports that this root hub contains.

Remarks

None.

Preconditions

None.

Example

```
// This code shows how DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet function can
// be called to obtain the number of Root hub ports.

DRV_HANDLE driverHandle;
uint8_t nPorts;

nPorts = DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet(driverHandle);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).

Function

uint8_t DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet(DRV_HANDLE handle);

DRV_USBHS_HOST_ROOT_HUB_PortReset Function

Resets the specified root hub port.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_HOST_ROOT_HUB_PortReset(DRV_HANDLE handle, uint8_t port);
```

Returns

None.

Description

This function resets the root hub port. The reset duration is defined by DRV_USBHS_ROOT_HUB_RESET_DURATION. The status of the reset signaling can be checked using the DRV_USBHS_ROOT_HUB_PortResetIsComplete function.

Remarks

The root hub on the PIC32MZ USB controller contains only one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USB_HOST_ROOT_HUB_PortReset and the
// DRV_USBHS_ROOT_HUB_PortResetIsComplete functions are called to complete a
// port reset sequence.

DRV_HANDLE driverHandle;

// Reset Port 0.
DRV_USB_HOST_ROOT_HUB_PortReset(driverHandle, 0);

// Check if the Reset operation has completed.
if(DRV_USBHS_ROOT_HUB_PortResetIsComplete(driverHandle, 0) == false)
{
    // This means that the Port Reset operation has not completed yet. The
    // DRV_USBHS_ROOT_HUB_PortResetIsComplete function should be called
    // again after some time to check the status.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
port	Port to reset.

Function

void DRV_USBHS_ROOT_HUB_PortReset(DRV_HANDLE handle, uint8_t port);

DRV_USBHS_HOST_ROOT_HUB_PortResetIsComplete Function

Returns true if the root hub has completed the port reset operation.

File

drv_usbhs.h

C

```
bool DRV_USBHS_HOST_ROOT_HUB_PortResetIsComplete(DRV_HANDLE handle, uint8_t port);
```

Returns

- true - The reset signaling has completed.

- false - The reset signaling has not completed.

Description

This function returns true if the port reset operation has completed. It should be called after the DRV_USB_HOST_ROOT_HUB_PortReset function to check if the reset operation has completed.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USB_HOST_ROOT_HUB_PortReset and the
// DRV_USBHS_ROOT_HUB_PortResetIsComplete functions are called to complete a
// port reset sequence.

DRV_HANDLE driverHandle;

// Reset Port 0.
DRV_USB_HOST_ROOT_HUB_PortReset(driverHandle, 0);

// Check if the Reset operation has completed.
if(DRV_USBHS_ROOT_HUB_PortResetIsComplete(driverHandle, 0) == false)
{
    // This means that the Port Reset operation has not completed yet. The
    // DRV_USBHS_ROOT_HUB_PortResetIsComplete function should be called
    // again after some time to check the status.
}
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
port	Port to check

Function

bool DRV_USBHS_ROOT_HUB_PortResetIsComplete

(

DRV_HANDLE handle,

uint8_t port

);

DRV_USBHS_HOST_ROOT_HUB_PortResume Function

Resumes the specified root hub port.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_HOST_ROOT_HUB_PortResume(DRV_HANDLE handle, uint8_t port);
```

Returns

- USB_ERROR_NONE - The function executed successfully.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid or the port number does not exist.

Description

This function resumes the root hub. The resume duration is defined by DRV_USBHS_ROOT_HUB_RESUME_DURATION. The status of the resume signaling can be checked using the DRV_USBHS_ROOT_HUB_PortResumeIsComplete function.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_PortResume function is
// called to resume the specified port.

DRV_HANDLE driverHandle;

// Resume Port 0.
DRV_USBHS_HOST_ROOT_HUB_PortResume(driverHandle, 0);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
port	Port to resume.

Function

```
USB_ERROR DRV_USBHS_HOST_ROOT_HUB_PortResume
(
    DRV_HANDLE handle,
uint8_t port
);
```

DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet Function

Returns the speed of at which the port is operating.

File

drv_usbhs.h

C

```
USB_SPEED DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet(DRV_HANDLE handle, uint8_t port);
```

Returns

- USB_SPEED_ERROR - This value is returned if the driver handle is not or if the speed information is not available or if the specified port is not valid.
- USB_SPEED_HIGH - A High Speed device has been connected to the port.
- USB_SPEED_FULL - A Full Speed device has been connected to the port.
- USB_SPEED_LOW - A Low Speed device has been connected to the port.

Description

This function returns the speed at which the port is operating.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet function is
// called to know the operating speed of the port. This also indicates the
// operating speed of the device connected to this port.

DRV_HANDLE driverHandle;
USB_SPEED speed;

speed = DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet(driverHandle, 0);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
port	Port number of the port to be analyzed..

Function

USB_SPEED DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet
(
DRV_HANDLE handle,
uint8_t port
);

DRV_USBHS_HOST_ROOT_HUB_PortSuspend Function

Suspends the specified root hub port.

File

drv_usbhs.h

C

```
USB_ERROR DRV_USBHS_HOST_ROOT_HUB_PortSuspend(DRV_HANDLE handle, uint8_t port);
```

Returns

- USB_ERROR_NONE - The function executed successfully.
- USB_ERROR_PARAMETER_INVALID - The driver handle is not valid or the port number does not exist.

Description

This function suspends the root hub port.

Remarks

The root hub on this particular hardware only contains one port - port 0.

Preconditions

None.

Example

```
// This code shows how the DRV_USBHS_HOST_ROOT_HUB_PortSuspend function is
// called to suspend the specified port.

DRV_HANDLE driverHandle;

// Suspend Port 0.
DRV_USBHS_HOST_ROOT_HUB_PortSuspend(driverHandle, 0);
```

Parameters

Parameters	Description
handle	Handle to the driver (returned from DRV_USBHS_Open function).
port	Port to suspend.

Function

USB_ERROR DRV_USBHS_ROOT_HUB_PortSuspend(DRV_HANDLE handle, uint8_t port);

f) Data Types and Constants

DRV_USBHS_EVENT Enumeration

Identifies the different events that the Hi-Speed USB Driver provides.

File

drv_usbhs.h

C

```
typedef enum {
  DRV_USBHS_EVENT_ERROR = DRV_USB_EVENT_ERROR,
  DRV_USBHS_EVENT_RESET_DETECT = DRV_USB_EVENT_RESET_DETECT,
  DRV_USBHS_EVENT_RESUME_DETECT = DRV_USB_EVENT_RESUME_DETECT,
  DRV_USBHS_EVENT_IDLE_DETECT = DRV_USB_EVENT_IDLE_DETECT,
  DRV_USBHS_EVENT_STALL = DRV_USB_EVENT_STALL,
  DRV_USBHS_EVENT_SOF_DETECT = DRV_USB_EVENT_SOF_DETECT,
  DRV_USBHS_EVENT_DEVICE_SESSION_VALID = DRV_USB_EVENT_DEVICE_SESSION_VALID,
  DRV_USBHS_EVENT_DEVICE_SESSION_INVALID = DRV_USB_EVENT_DEVICE_SESSION_INVALID
} DRV_USBHS_EVENT;
```

Members

Members	Description
DRV_USBHS_EVENT_ERROR = DRV_USB_EVENT_ERROR	Bus error occurred and was reported
DRV_USBHS_EVENT_RESET_DETECT = DRV_USB_EVENT_RESET_DETECT	Host has issued a device reset
DRV_USBHS_EVENT_RESUME_DETECT = DRV_USB_EVENT_RESUME_DETECT	Resume detected while USB in suspend mode
DRV_USBHS_EVENT_IDLE_DETECT = DRV_USB_EVENT_IDLE_DETECT	Idle detected
DRV_USBHS_EVENT_STALL = DRV_USB_EVENT_STALL	Stall handshake has occurred
DRV_USBHS_EVENT_SOF_DETECT = DRV_USB_EVENT_SOF_DETECT	Device received SOF operation
DRV_USBHS_EVENT_DEVICE_SESSION_VALID = DRV_USB_EVENT_DEVICE_SESSION_VALID	VBUS voltage had Session valid
DRV_USBHS_EVENT_DEVICE_SESSION_INVALID = DRV_USB_EVENT_DEVICE_SESSION_INVALID	Session Invalid

Description

Hi-Speed USB Driver Events Enumeration.

This enumeration identifies the different events that are generated by the Hi-Speed USB Driver.

Remarks

None.

DRV_USBHS_EVENT_CALLBACK Type

Type of the Hi-Speed USB Driver event callback function.

File

drv_usbhs.h

C

```
typedef void (* DRV_USBHS_EVENT_CALLBACK)(uintptr_t hClient, DRV_USBHS_EVENT eventType, void * eventData);
```

Returns

None.

Description

Type of the Hi-Speed USB Driver Event Callback Function.

Define the type of the Hi-Speed USB Driver event callback function. The client should register an event callback function of this type when it intends to receive events from the Hi-Speed USB Driver. The event callback function is registered using the DRV_USBHS_ClientEventCallBackSet function.

Remarks

None.

Parameters

Parameters	Description
hClient	Handle to the driver client that registered this callback function.
eventType	This parameter identifies the event that caused the callback function to be called.
eventData	Pointer to a data structure that is related to this event. This value will be NULL if the event has no related data.

DRV_USBHS_HOST_PIPE_HANDLE Type

Defines the Hi-Speed USB Driver Host Pipe Handle type.

File

drv_usbhs.h

C

```
typedef uintptr_t DRV_USBHS_HOST_PIPE_HANDLE;
```

Description

Hi-Speed USB Driver Host Pipe Handle.

This type definition defines the type of the Hi-Speed USB Driver Host Pipe Handle.

Remarks

None.

DRV_USBHS_INIT Structure

This type definition defines the Driver Initialization Data Structure.

File

drv_usbhs.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  USBHS_MODULE_ID usbID;
  bool stopInIdle;
  bool suspendInSleep;
  INT_SOURCE interruptSource;
  INT_SOURCE interruptSourceUSBDma;
  USB_SPEED operationSpeed;
  DRV_USBHS_OPMODES operationMode;
  void * endpointTable;
  uint32_t rootHubAvailableCurrent;
  DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE portPowerEnable;
  DRV_USBHS_ROOT_HUB_PORT_INDICATION portIndication;
  DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT portOverCurrentDetect;
} DRV_USBHS_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System Module Initialization
USBHS_MODULE_ID usbID;	Identifies the USB peripheral to be used. This should be the USB PLIB module instance identifier.
bool stopInIdle;	This should be set to true if the USB module must stop operation in Idle mode
bool suspendInSleep;	This should be set to true if the USB module must suspend when the CPU enters Sleep mode.
INT_SOURCE interruptSource;	Specify the interrupt source for the USB module. This should be the interrupt source for the USB module instance specified in usbID.

INT_SOURCE interruptSourceUSBDma;	Specify the interrupt source for the USB module specific DMA controller. This should be the USB DMA interrupt source for the USB Module instance specified in usbID.
USB_SPEED operationSpeed;	Specify the operational speed of the USB module. This should always be set to USB_SPEED_FULL.
DRV_USBHS_OPMODES operationMode;	Specify the operation mode of the USB module. This specifies if the USB module should operate as a Device, Host, or both (Dual Role operation).
void * endpointTable;	A pointer to the endpoint descriptor table. This should be aligned at 512 byte address boundary. The size of the table is equal to the DRV_USBHS_ENDPOINT_TABLE_ENTRY_SIZE times the number of endpoints needed in the application.
uint32_t rootHubAvailableCurrent;	Root hub available current in milliamperes. This specifies the amount of current that root hub can provide to the attached device. This should be specified in mA. This is required when the driver is required to operate in Host mode.
DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE portPowerEnable;	When operating in Host mode, the application can specify a Root hub port enable function. This parameter should point to Root hub port enable function. If this parameter is NULL, it implies that the port is always enabled.
DRV_USBHS_ROOT_HUB_PORT_INDICATION portIndication;	When operating in Host mode, the application can specify a Root Port Indication. This parameter should point to the Root Port Indication function. If this parameter is NULL, it implies that Root Port Indication is not supported.
DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT portOverCurrentDetect;	When operating is Host mode, the application can specify a Root Port Overcurrent detection. This parameter should point to the Root Port Indication function. If this parameter is NULL, it implies that Overcurrent detection is not supported.

Description

USB Device Driver Initialization Data.

This structure contains all the data necessary to initialize the Hi-Speed USB Driver. A pointer to a structure of this type, containing the desired initialization data, must be passed into the DRV_USBHS_Initialize function.

Remarks

None.

DRV_USBHS_OPMODES Enumeration

Identifies the operating modes supported by the Hi-Speed USB Driver.

File

drv_usbhs.h

C

```
typedef enum {
  DRV_USBHS_OPMODE_DUAL_ROLE = DRV_USB_OPMODE_DUAL_ROLE,
  DRV_USBHS_OPMODE_DEVICE = DRV_USB_OPMODE_DEVICE,
  DRV_USBHS_OPMODE_HOST = DRV_USB_OPMODE_HOST,
  DRV_USBHS_OPMODE_OTG = DRV_USB_OPMODE_OTG
} DRV_USBHS_OPMODES;
```

Members

Members	Description
DRV_USBHS_OPMODE_DUAL_ROLE = DRV_USB_OPMODE_DUAL_ROLE	The driver should be able to switch between Host and Device mode
DRV_USBHS_OPMODE_DEVICE = DRV_USB_OPMODE_DEVICE	The driver should support Device mode operation only
DRV_USBHS_OPMODE_HOST = DRV_USB_OPMODE_HOST	The driver should support Host mode operation only
DRV_USBHS_OPMODE_OTG = DRV_USB_OPMODE_OTG	The driver should support the OTG protocol

Description

USB Operating Modes Enumeration.

This enumeration identifies the operating modes supported by the Hi-Speed USB Driver.

Remarks

None.

DRV_USBHS_ROOT_HUB_PORT_INDICATION Type

USB Root hub Application Hooks (Port Indication).

File

drv_usbhs.h

C

```
typedef void (* DRV_USBHS_ROOT_HUB_PORT_INDICATION)(uint8_t port, USB_HUB_PORT_INDICATOR_COLOR color,
USB_HUB_PORT_INDICATOR_STATE state);
```

Description

USB Root hub Application Hooks (Port Indication).

A function of the type defined here should be provided to the driver root to implement Port Indication. The root hub driver calls this function when it needs to update the state of the port indication LEDs. The application can choose to implement the Amber and Green colors as one LED or two different LEDs. The root hub driver specifies the color and the indicator attribute (on, off or blinking) when it calls this function.

Remarks

None.

DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT Type

USB Root hub Application Hooks (Port Overcurrent detection).

File

drv_usbhs.h

C

```
typedef bool (* DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT)(uint8_t port);
```

Description

USB Root hub Application Hooks (Port Overcurrent detection).

A function of the type defined here should be provided to the driver root hub to check for port over current condition. This function will be called periodically by the root hub driver to check the Overcurrent status of the port. It should continue to return true while the Overcurrent condition exists on the port. It should return false when the Overcurrent condition does not exist.

Remarks

None.

DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE Type

USB Root hub Application Hooks (Port Power Enable/ Disable).

File

drv_usbhs.h

C

```
typedef void (* DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE)(uint8_t port, bool control);
```

Description

USB Root hub Application Hooks (Port Power Enable/ Disable).

A function of the type defined here should be provided to the driver root to control port power. The root hub driver will call this function when it needs to enable port power. If the application circuit contains a VBUS switch, the switch should be accessed and controlled by this function. If the enable parameter is true, the switch should be enabled and VBUS should be available on the port. If the enable parameter is false, the switch should be disabled and VBUS should not be available on the port.

Remarks

None.

DRV_USBHS_DEVICE_INTERFACE Macro

Hi-Speed USB Driver Device Mode Interface Functions.

File

drv_usbhs.h

C

```
#define DRV_USBHS_DEVICE_INTERFACE
```

Description

Hi-Speed USB Driver Device Mode Interface Functions.

The Device Controller Driver Interface member of the Device Stack Initialization data structure should be set to this value so that the Device Stack can access the Hi-Speed USB Driver Device Mode functions.

Remarks

None.

DRV_USBHS_HOST_INTERFACE Macro

Hi-Speed USB Driver Host Mode Interface Functions.

File

drv_usbhs.h

C

```
#define DRV_USBHS_HOST_INTERFACE
```

Description

Hi-Speed USB Driver Host Mode Interface Functions.

The Host Controller Driver Interface member of the Host Layer Initialization data structure should be set to this value so that the Host Layer can access the Hi-Speed USB Driver Host Mode functions.

Remarks

None.

DRV_USBHS_HOST_PIPE_HANDLE_INVALID Macro

Value of an Invalid Host Pipe Handle.

File

drv_usbhs.h

C

```
#define DRV_USBHS_HOST_PIPE_HANDLE_INVALID
```

Description

Hi-Speed USB Driver Invalid Host Pipe Handle.

This constant defines the value of an Invalid Host Pipe Handle.

Remarks

None.

DRV_USBHS_INDEX_0 Macro

Hi-Speed USB Driver Module Index 0 Definition.

File

drv_usbhs.h

C

```
#define DRV_USBHS_INDEX_0 0
```

Description

Hi-Speed USB Driver Module Index 0 Definition.

This constant defines the value of Hi-Speed USB Driver Index 0. The SYS_MODULE_INDEX parameter of the DRV_USBHS_Initialize and DRV_USBHS_Open functions should be set to this value to identify instance 0 of the driver.

Remarks

These constants should be used in place of hard-coded numeric literals and should be passed into the DRV_USBHS_Initialize and DRV_USBHS_Open functions to identify the driver instance in use. These are not indicative of the number of modules that are actually supported by the microcontroller.

Files

Files

Name	Description
drv_usbhs.h	PIC32MZ USB Module Driver Interface File
drv_usbhs_config_template.h	Hi-Speed USB (USBHS) Driver Configuration Template.

Description

drv_usbhs.h

PIC32MZ USB Module Driver Interface File

Enumerations

	Name	Description
	DRV_USBHS_EVENT	Identifies the different events that the Hi-Speed USB Driver provides.
	DRV_USBHS_OPMODES	Identifies the operating modes supported by the Hi-Speed USB Driver.

Functions

	Name	Description
	DRV_USBHS_ClientEventCallBackSet	This function sets up the event callback function that is invoked by the USB controller driver to notify the client of USB bus events.
	DRV_USBHS_Close	Closes an opened-instance of the Hi-Speed USB Driver.
	DRV_USBHS_DEVICE_AddressSet	This function will set the USB module address that is obtained from the Host.
	DRV_USBHS_DEVICE_Attach	This function will enable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that a device has been attached on the bus.
	DRV_USBHS_DEVICE_CurrentSpeedGet	This function will return the USB speed at which the device is operating.
	DRV_USBHS_DEVICE_Detach	This function will disable the attach signaling resistors on the D+ and D- lines thus letting the USB Host know that the device has detached from the bus.
	DRV_USBHS_DEVICE_EndpointDisable	This function disables an endpoint.
	DRV_USBHS_DEVICE_EndpointDisableAll	This function disables all provisioned endpoints.
	DRV_USBHS_DEVICE_EndpointEnable	This function enables an endpoint for the specified direction and endpoint size.
	DRV_USBHS_DEVICE_EndpointIsEnabled	This function returns the enable/disable status of the specified endpoint and direction.
	DRV_USBHS_DEVICE_EndpointIsStalled	This function returns the stall status of the specified endpoint and direction.
	DRV_USBHS_DEVICE_EndpointStall	This function stalls an endpoint in the specified direction.
	DRV_USBHS_DEVICE_EndpointStallClear	This function clears the stall on an endpoint in the specified direction.
	DRV_USBHS_DEVICE_IRPCancel	This function cancels the specific IRP that are queued and in progress at the specified endpoint.

	DRV_USBHS_DEVICE_IRPCancelAll	This function cancels all IRPs that are queued and in progress at the specified endpoint.
	DRV_USBHS_DEVICE_IRPSubmit	This function submits an I/O Request Packet (IRP) for processing to the Hi-Speed USB Driver.
	DRV_USBHS_DEVICE_RemoteWakeupStart	This function causes the device to start Remote Wakeup Signalling on the bus.
	DRV_USBHS_DEVICE_RemoteWakeupStop	This function causes the device to stop the Remote Wakeup Signalling on the bus.
	DRV_USBHS_DEVICE_SOFNumberGet	This function will return the USB SOF packet number.
	DRV_USBHS_DEVICE_TestModeEnter	This function enables the specified USB 2.0 Test Mode.
	DRV_USBHS_DEVICE_TestModeExit	This function disables the specified USB 2.0 Test Mode.
	DRV_USBHS_HOST_EventsDisable	Disables Host mode events.
	DRV_USBHS_HOST_EventsEnable	Restores the events to the specified the original value.
	DRV_USBHS_HOST_IRPCancel	Cancels the specified IRP.
	DRV_USBHS_HOST_IRPSubmit	Submits an IRP on a pipe.
	DRV_USBHS_HOST_PipeClose	Closes an open pipe.
	DRV_USBHS_HOST_PipeSetup	Open a pipe with the specified attributes.
	DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet	This function returns the operating speed of the bus to which this root hub is connected.
	DRV_USBHS_HOST_ROOT_HUB_Initialize	This function initializes the root hub driver.
	DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet	Returns the maximum amount of current that this root hub can provide on the bus.
	DRV_USBHS_HOST_ROOT_HUB_OperationEnable	This function enables or disables root hub operation.
	DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled	Returns the operation enabled status of the root hub.
	DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet	Returns the number of ports this root hub contains.
	DRV_USBHS_HOST_ROOT_HUB_PortReset	Resets the specified root hub port.
	DRV_USBHS_HOST_ROOT_HUB_PortResetIsComplete	Returns true if the root hub has completed the port reset operation.
	DRV_USBHS_HOST_ROOT_HUB_PortResume	Resumes the specified root hub port.
	DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet	Returns the speed of at which the port is operating.
	DRV_USBHS_HOST_ROOT_HUB_PortSuspend	Suspends the specified root hub port.
	DRV_USBHS_Initialize	Initializes the Hi-Speed USB Driver.
	DRV_USBHS_Open	Opens the specified Hi-Speed USB Driver instance and returns a handle to it.
	DRV_USBHS_Status	Provides the current status of the Hi-Speed USB Driver module.
	DRV_USBHS_Tasks	Maintains the driver's state machine when the driver is configured for Polled mode.
	DRV_USBHS_Tasks_ISR	Maintains the driver's Interrupt state machine and implements its ISR.
	DRV_USBHS_Tasks_ISR_USBDMA	Maintains the driver's DMA Transfer state machine and implements its ISR.

Macros

	Name	Description
	DRV_USBHS_DEVICE_INTERFACE	Hi-Speed USB Driver Device Mode Interface Functions.
	DRV_USBHS_HOST_INTERFACE	Hi-Speed USB Driver Host Mode Interface Functions.
	DRV_USBHS_HOST_PIPE_HANDLE_INVALID	Value of an Invalid Host Pipe Handle.
	DRV_USBHS_INDEX_0	Hi-Speed USB Driver Module Index 0 Definition.

Structures

	Name	Description
	DRV_USBHS_INIT	This type definition defines the Driver Initialization Data Structure.

Types

	Name	Description
	DRV_USBHS_EVENT_CALLBACK	Type of the Hi-Speed USB Driver event callback function.
	DRV_USBHS_HOST_PIPE_HANDLE	Defines the Hi-Speed USB Driver Host Pipe Handle type.
	DRV_USBHS_ROOT_HUB_PORT_INDICATION	USB Root hub Application Hooks (Port Indication).
	DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT	USB Root hub Application Hooks (Port Overcurrent detection).
	DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE	USB Root hub Application Hooks (Port Power Enable/ Disable).

Description

PIC32MZ USB Module Driver Interface Header File

The PIC32MZ Hi-Speed USB Module driver provides a simple interface to manage the "USB" peripheral on the PIC32MZ microcontroller. This file defines the interface definitions and prototypes for the Hi-Speed USB Driver. The driver interface meets the requirements of the MPLAB Harmony USB Host and Device Layer.

File Name

drv_usbhs.h

Company

Microchip Technology Inc.

drv_usbhs_config_template.h

Hi-Speed USB (USBHS) Driver Configuration Template.

Macros

	Name	Description
	DRV_USBHS_DEVICE_SUPPORT	Determines if the USB Device Functionality should be enabled.
	DRV_USBHS_ENDPOINTS_NUMBER	Configures the number of endpoints to be provisioned in the driver.
	DRV_USBHS_HOST_ATTACH_DEBOUNCE_DURATION	Configures the time duration (in milliseconds) that the driver will wait to reconfirm a device attach.
	DRV_USBHS_HOST_NAK_LIMIT	Configures the NAK Limit for Host Mode Control Transfers.
	DRV_USBHS_HOST_PIPES_NUMBER	Configures the maximum number of pipes that are can be opened when the driver is operating in Host mode.
	DRV_USBHS_HOST_RESET_DURATION	Configures the time duration (in milliseconds) of the Reset Signal.
	DRV_USBHS_HOST_SUPPORT	Determines if the USB Host Functionality should be enabled.
	DRV_USBHS_INSTANCES_NUMBER	Specifies the number of driver instances to be enabled in the application.
	DRV_USBHS_INTERRUPT_MODE	Configures the driver for interrupt or polling mode operation.

Description

Hi-Speed USB Driver Configuration Template

This file lists all the configurations constants that affect the operation of the USBHS Driver.

File Name

drv_usbhs_config_template.h

Company

Microchip Technology Inc.

USART Driver Library

This section describes the USART Driver Library.

Introduction

This section introduces the MPLAB Harmony USART Driver.

Description

The MPLAB Harmony USART Driver (also referred to as the USART Driver) provides a high-level interface to the USART and UART peripherals on Microchip's PIC32 microcontrollers. This driver provides application ready routines to read and write data to the UART using common data transfer models, which eliminates the need for the application to implement this code. The USART driver features the following:

- Provides byte transfer, read/write, and buffer queue data transfer models
- Supports Interrupt and Polled modes of operation
- Supports point-to-point data communication
- Supports multi-client and multi-instance operation
- Provides data transfer events
- Supports blocking and non-blocking operation with the read/write data transfer model

- Features thread-safe functions for use in RTOS applications
- Supports DMA transfers
- Supports high baud rate setting
- Major features are implemented in separate source code files and can be included only if needed. This helps optimize overall application code size.

Using the Library

This topic describes the basic architecture of the USART Driver Library and provides information and examples on its use.

Description

Interface Header File: `drv_usart.h`

The interface to the USART library is defined in the `drv_usart.h` header file.

Please refer to the What is MPLAB Harmony? section for how the driver interacts with the framework.

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the USART Driver Library.

Library Interface Section	Description
System Routines	These routines are accessed by the MPLAB Harmony system module. They allow the driver to be initialized, deinitialized and maintained.
Core Client Routines	These routines allow the application client to open and close the driver.
Communication Management Client Routines	These routines allow the client to change the properties of the communication channel (such as baud, parity, etc.).
Buffer Queue Read/Write Client Routines	These routines allow the client to use the buffer queue data transfer model.
File I/O Type Read/Write Routines	These routines allow the client to use the file I/O type read/write routines.
Byte Transfer Routines	These routines allow the client to use the byte data transfer model.

The USART driver must be first initialized. One or more application clients can then open the USART Driver in Blocking or non-Blocking mode. The Open function returns a handle which allows the client to access the driver client functionality. The Driver tasks routines should be invoked regularly from the SYS_Tasks routine in case of Polled mode operation or from USART Driver Interrupt Service Routine (ISR), in case of Interrupt mode.

The driver implementation is split across multiple files to optimize the application project code size. The application project must include the `drv_usart.c` file if the USART driver is needed in the application. If DMA-enabled data transfers are required, the `drv_usart_dma.c` file should be included into the project instead of the `drv_usart.c` file. These files implement the system and core Client routines. Other driver files can be included based on the required driver features.

The USART Driver API, unless otherwise specified, should not be called from an interrupt context. That is, they should not be called from an ISR or from event handlers that are executing within an ISR context.

Abstraction Model

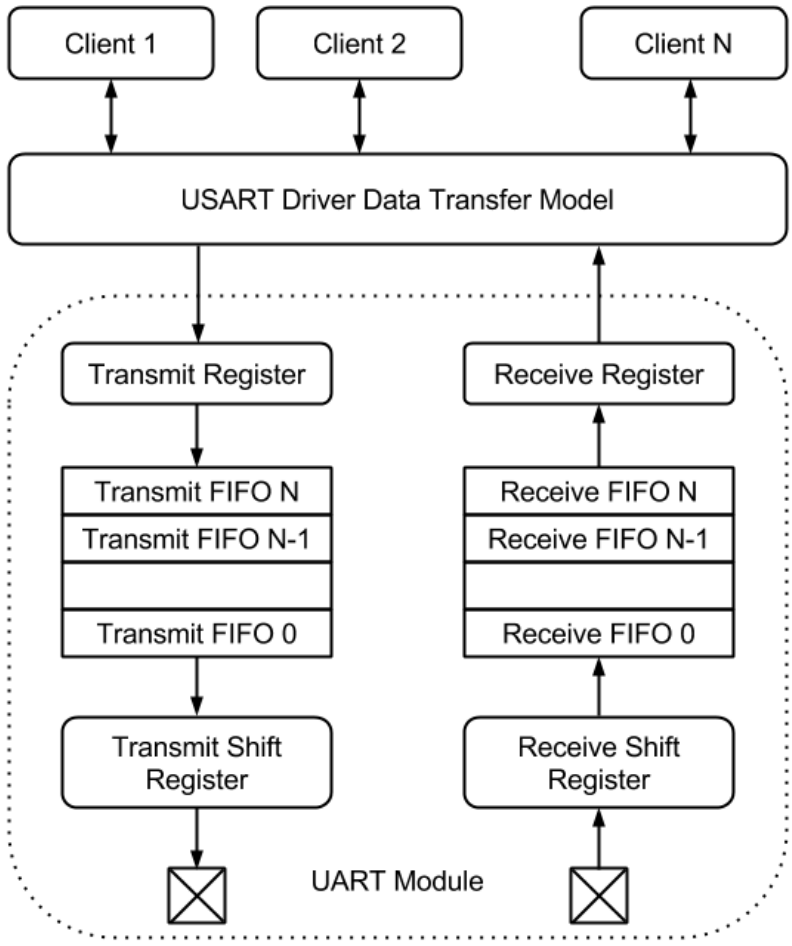
This section describes how the USART Driver abstracts the USART peripheral features.

Description

The USART driver features routines to perform the following functions:

- Driver initialization
- Transfer data
- Manage communication properties of the module

The Driver initialization routines allow the system to initialize the driver. The driver must be initialized before it can be opened by a client. The data transfer routines allow the application to receive and transmit data through the USART. The driver also provides routines to change the communication properties such as USART baud or line control settings.



As seen in the previous figure, the USART driver clients transfer data through the USART Driver Data Transfer model. The driver abstracts out the hardware details of the UART module FIFO mechanism and shift registers, and provides a low overhead data transfer mechanism to the application. The USART driver provides three different data transfer models for transferring data.

- The Byte Transfer Model
- The File I/O Type Read/Write Transfer Model
- Buffer Queue Transfer Model

Byte Transfer Model:

The byte transfer model allows the application to transfer data through USART driver one byte at a time. With this model, the driver reads one byte from the receive FIFO or writes one byte to the transmit FIFO. The application must check if data has been received before reading the data. Similarly, it must check if the transmit FIFO is not full before writing to the FIFO. The byte transfer model places the responsibility of maintaining the USART peripheral on the Application. The driver cannot support other data transfer models if support for this data transfer model is enabled. The byte transfer model is only recommended for simple data transfer applications.

To use the byte transfer model, the `drv_usart_byte_model.c` file must be included in the project and the DRV_USART_BYTE_MODEL_SUPPORT configuration macro should be set to true.

File I/O Type Read/Write Transfer Model:

This data transfer model is similar to file read and write API model in a UNIX operating system application. The application calls the USART driver read and write routines to transfer data through the USART. Unlike the byte transfer model, the read/write data model can process a block of data. Depending on the mode (blocking or non-blocking) in which the client opened the driver, the driver will either block until all of the data is transferred or will immediately return with the number of bytes transferred. The application does not have to check the FIFO status while using this mode. The application can instead use the return status (number of bytes transferred) to maintain its logic and complete the data transfer. The read/write model can be used with the non-DMA buffer queue model. It cannot be used with the byte transfer model and the DMA-enabled buffer queue model in the same application.

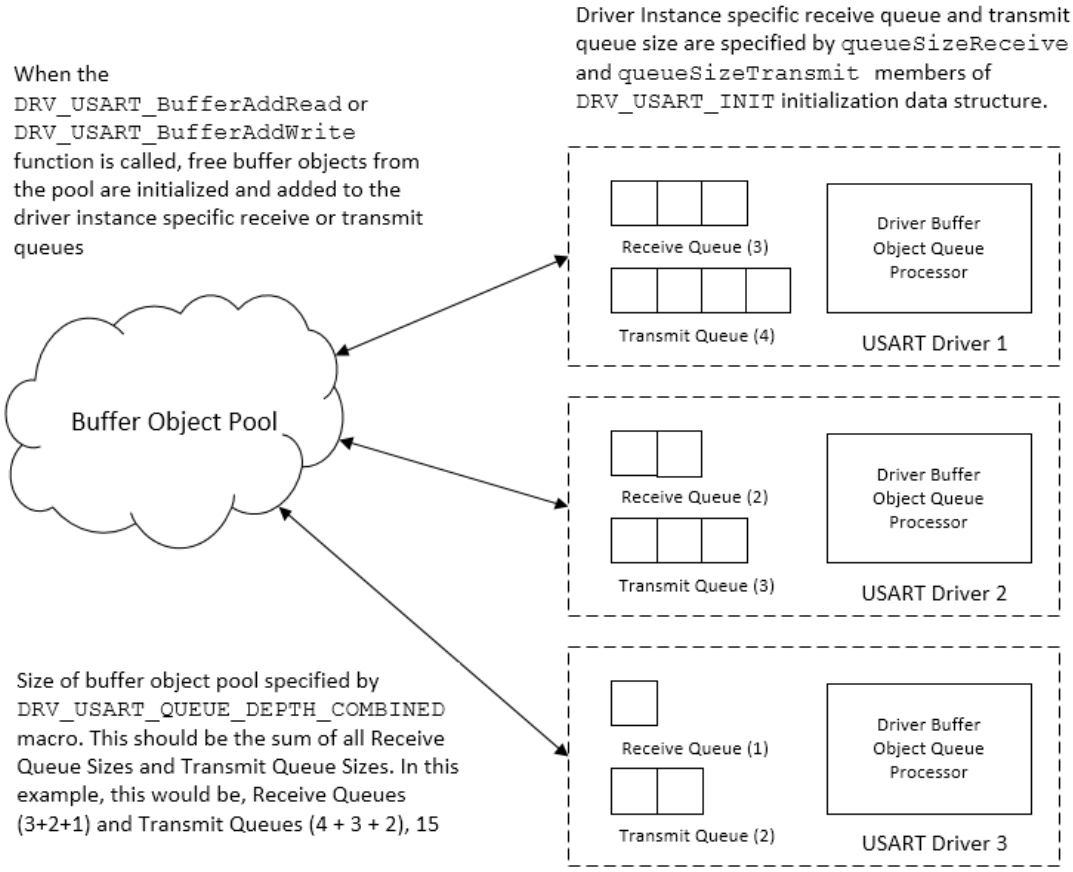
To use the file I/O type read/write data transfer model, the `drv_usart_read_write.c` file must be included in the project and the DRV_USART_READ_WRITE_MODEL_SUPPORT configuration macro should be set to true.

See File I/O Type Read/Write Data Transfer Modef for additional information.

Buffer Queue Transfer Model:

The buffer queue data transfer model allows clients to queue data transfers for processing. This data transfer model is always non-blocking. The USART driver returns a buffer handle for a queued request. The clients can track the completion of a buffer through events and API. If the USART driver is busy processing a data transfer, other data transfer requests are queued. This allows the clients to optimize their application logic and increase throughput. To optimize memory usage, the USART driver implements a shared buffer object pool concept to add a data transfer request to the queue. The following figure shows a conceptual representation of the buffer queue model.

Buffer Queue Transfer Model



As shown in the previous figure, each USART driver hardware instance has a read and write queue. The system designer must configure the sizes of these read and write queues. The USART driver additionally employs a global pool of buffer queue objects. This pool is common to all USART Driver hardware instances and its size is defined by the DRV_USART_QUEUE_DEPTH_COMBINED configuration macro. When a client places a request to add a data transfer, the driver performs the following actions:

- It checks if a buffer object is free in the global pool. If not, the driver rejects the request.
- It then checks if the hardware instance specific queue is full. If not, the buffer object from the global pool is added to the hardware instance specific queue. If the queue is full, the driver rejects the request.

The buffer queue model can be used along with the file I/O type read/write data transfer model.

To use the Buffer Queue Data Transfer model, the `drv_usart_buffer_queue.c` file must be included in the project and DRV_USART_BUFFER_QUEUE_SUPPORT configuration macro should be set to true.

The USART Driver DMA feature is only available while using the Buffer Queue Model. If enabled, the USART Driver uses the DMA module channels to transfer data directly from application memory to USART transmit or receive registers. This reduces CPU resource consumption and improves system performance. To use the buffer queue model with DMA, the `drv_usart_buffer_queue_dma.c` file should be included in the project instead of `drv_usart_buffer_queue.c`.

See Buffer Queue Transfer Model for additional information.

Communication Management

The USART Driver API contains functions to control the USART Driver communication properties. These functions allow the client to change the parity, stop bits, number of data bits and the communication baud rate. A change in the communication setting affects all ongoing communication and all driver clients.

How the Library Works

This section describes how to use the USART Driver.

Description

Prior to using the USART Driver, the application must decide on which USART data transfer models are required. The application project should then include the USART Driver files, required to support the data transfer model into the application project. Additionally, the application design must consider the need for USART Driver to be opened in blocking or non blocking modes. This will also affect the application flow.

Initializing the USART Driver

Describes how to initialize the USART Driver.

Description

The USART Driver must be configured and initialized for clients to be able to open the driver. The driver build time configuration is defined by the configuration macros. Refer to the Building the Library section for the location of and more information on the various configuration macros and how these macros should be designed. The driver initialization is configured through the DRV_USART_INIT data structure that is passed to the DRV_USART_Initialize function. The initialization parameters include the USART baud, the USART Peripheral, USART interrupts and read queue and write queue sizes (which are applicable only when buffer queue data transfer is used). The following code shows an example of initializing the USART driver for 300 bps and uses USART2. If the driver is configured for Interrupt mode of operation, the priority of the USART interrupts needs to be specified.

```
/* The following code shows an example of designing the
 * DRV_USART_INIT data structure. It also shows how an example
 * usage of the DRV_USART_Initialize() function and how Interrupt
 * System Service routines are used to set USART Interrupt
 * priority. */

DRV_USART_INIT usartInit;
SYS_MODULE_OBJ usartModule1;

/* Set the baud to 300 */
usartInit.baud = 300;

/* Auto Baud detection or Stop Idle is not needed */
usartInit.flags = DRV_USART_INIT_FLAG_NONE;

/* Handshaking is not needed */
usartInit.handshake = DRV_USART_HANDSHAKE_NONE;

/* USART Error Interrupt source for this USART
 * driver instance. Note that INT_SOURCE_USART_2_ERROR
 * value is defined by the Interrupt System Service and
 * is the error interrupt for USART 2*/
usartInit.interruptError = INT_SOURCE_USART_2_ERROR;

/* USART Receive Interrupt source for this USART
 * driver instance. Note that INT_SOURCE_USART_2_RECEIVE
 * value is defined by the Interrupt System Service and
 * is the error interrupt for USART 2 */
usartInit.interruptReceive = INT_SOURCE_USART_2_RECEIVE;

/* USART Transmit Interrupt source for this USART
 * driver instance. Note that INT_SOURCE_USART_2_TRANSMIT
 * value is defined by the Interrupt System Service and
 * is the error interrupt for USART 2 */
usartInit.interruptTransmit = INT_SOURCE_USART_2_TRANSMIT;

/* Line control mode */
usartInit.lineControl = DRV_USART_LINE_CONTROL_8NONE1;

/* Operation mode is normal. Loopback or addressed is not
 * needed */
usartInit.mode = DRV_USART_OPERATION_MODE_NORMAL;

/* Peripheral Bus clock frequency at which the USART is
 * operating */
usartInit.brgClock = 80000000;

/* System module power setting. Typically set to
 * SYS_MODULE_POWER_RUN_FULL */
usartInit.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;
```

```
/* Receive buffer queue size. In this case a maximum of 2
 * receive buffers can be queued. Only applicable if the
 * Buffer Queue Data Transfer Model is included in the
 * application. */
usartInit.queueSizeReceive = 2;

/* Transmit buffer queue size. In this case a maximum of 3
 * transmit buffers can be queued. Only applicable if the
 * Buffer Queue Data Transfer Model is included in the
 * application. */
usartInit.queueSizeTransmit = 3;

/* The USART peripheral instance index associated with this
 * driver instance. Note that this value is defined by the
 * USART Peripheral Library */
usartInit.usartID = USART_ID_2;

/* Initialize USART Driver Instance 0 */
usartModule1 = DRV_USART_Initialize(DRV_USART_0, (SYS_MODULE_INIT*)&usartInit);

/* The result of the driver initialization can be checked */
if(SYS_MODULE_OBJ_INVALID == usartModule1)
{
    /* There was an error in initialization. */
}

/* If the USART driver is configured for interrupt mode of
 * operation, the interrupt priorities should be configured.
 * Here the Interrupt System Service is used to set the
 * priority to level 4 */

/* Initialize the interrupt system service */
SYS_INT_Initialize();

/* Set the USART 2 module interrupt priority to 4*/
SYS_INT_VectorPrioritySet(INT_VECTOR_UART2, INT_PRIORITY_LEVEL4);

/* Set the USART 2 module interrupt sub-priority to 0*/
SYS_INT_VectorSubprioritySet(INT_VECTOR_UART2, INT_SUBPRIORITY_LEVEL0);

/* Enable global interrupt */
SYS_INT_Enable();
```

The USART Driver can be configured to transfer data through the DMA. In such a case, the DMA channels to be used for USART transmit and receive need to be specified. The USART Driver depends on the DMA System Service to access the DMA module. The DMA channels to be used for transmit and receive transfers should be specified in the DRV_USART_INIT data structure.

The following code shows an example of using the USART Driver initialization to use DMA for transferring data. The code also shows example initialization of the DMA System Service.

```
/* The following code shows an example of designing the
 * DRV_USART_INIT data structure. It also shows how an example
 * usage of the DRV_USART_Initialize() function and how Interrupt
 * System Service routines are used to set USART Interrupt
 * priority. */

DRV_USART_INIT usartInit;
SYS_DMA_INIT dmaInit;
SYS_MODULE_OBJ usartModule1;
SYS_MODULE_OBJ dmaModule;

/* Set the baud to 300 */
usartInit.baud = 300;

/* Auto Baud detection or Stop Idle is not needed */
usartInit.flags = DRV_USART_INIT_FLAG_NONE;

/* Handshaking is not needed */
usartInit.handshake = DRV_USART_HANDSHAKE_NONE;
```

```
/* USART Error Interrupt source for this USART
 * driver instance. Note that INT_SOURCE_USART_2_ERROR
 * value is defined by the Interrupt System Service and
 * is the error interrupt for USART2*/
usartInit.interruptError = INT_SOURCE_USART_2_ERROR;

/* USART Receive Interrupt source for this USART
 * driver instance. Note that INT_SOURCE_USART_2_RECEIVE
 * value is defined by the Interrupt System Service and
 * is the receive interrupt for USART2 */
usartInit.interruptReceive = INT_SOURCE_USART_2_RECEIVE;

/* USART Transmit Interrupt source for this USART
 * driver instance. Note that INT_SOURCE_USART_2_TRANSMIT
 * value is defined by the Interrupt System Service and
 * is the transmit interrupt for USART2 */
usartInit.interruptTransmit = INT_SOURCE_USART_2_TRANSMIT;

/* Line control mode */
usartInit.lineControl = DRV_USART_LINE_CONTROL_8NONE1;

/* Operation mode is normal. Loopback or addressed is not
 * needed */
usartInit.mode = DRV_USART_OPERATION_MODE_NORMAL;

/* Peripheral Bus clock frequency at which the USART is
 * operating */
usartInit.brgClock = 80000000;

/* System module power setting. Typically set to
 * SYS_MODULE_POWER_RUN_FULL */
usartInit.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;

/* Receive buffer queue size. In this case a maximum of 2
 * receive buffers can be queued. Only applicable if the
 * Buffer Queue Data Transfer Model is included in the
 * application. */
usartInit.queueSizeReceive = 2;

/* Transmit buffer queue size. In this case a maximum of 3
 * transmit buffers can be queued. Only applicable if the
 * Buffer Queue Data Transfer Model is included in the
 * application. */
usartInit.queueSizeTransmit = 3;

/* The USART peripheral instance index associated with this
 * driver instance. Note that this value is defined by the
 * USART Peripheral Library */
usartInit.usartID = USART_ID_2;

/* Use DMA channel 1 for transmit. If transmit via DMA is
 * not required, set this to DMA_CHANNEL_NONE. These values
 * are defined by the DMA System Service. */
usartInit.dmaChannelTransmit = DMA_CHANNEL_1;

/* Use DMA channel 2 for receive. If receive via DMA is
 * not required, set this to DMA_CHANNEL_NONE. These values
 * are defined by the DMA System Service. */
usartInit.dmaChannelReceive = DMA_CHANNEL_2;

/* Set the interrupt source for the Transmit DMA channel.
 * This parameter is ignored if the dmaChannelTransmit
 * parameter is set to DMA_CHANNEL_NONE. */
usartInit.dmaInterruptTransmit = INT_SOURCE_DMA_1;

/* Set the interrupt source for the Receive DMA channel.
 * This parameter is ignored if the dmaChannelReceive
 * parameter is set to DMA_CHANNEL_NONE. */
usartInit.dmaInterruptReceive = INT_SOURCE_DMA_2;
```

```
/********* End of DRV_USART_INIT Initialization *************/


/* If the USART driver is configured for interrupt mode of
 * operation, the interrupt priorities should be configured.
 * Here the Interrupt System Service is used to set the
 * priority to level 4 */

/* Initialize the interrupt system service */
SYS_INT_Initialize();

/* Set the USART 2 module interrupt priority to 4*/
SYS_INT_VectorPrioritySet(INT_VECTOR_UART2, INT_PRIORITY_LEVEL4);

/* Set the USART 2 module interrupt sub-priority to 0*/
SYS_INT_VectorSubprioritySet(INT_VECTOR_UART2, INT_SUBPRIORITY_LEVEL0);

/* Set the DMA 1 channel interrupt priority to 4*/
SYS_INT_VectorPrioritySet(INT_VECTOR_DMA1, INT_PRIORITY_LEVEL4);

/* Set the DMA 1 channel interrupt sub-priority to 0*/
SYS_INT_VectorSubprioritySet(INT_VECTOR_DMA1, INT_SUBPRIORITY_LEVEL0);

/* Set the DMA 2 channel interrupt priority to 4*/
SYS_INT_VectorPrioritySet(INT_VECTOR_DMA2, INT_PRIORITY_LEVEL4);

/* Set the DMA 2 channel interrupt sub-priority to 0*/
SYS_INT_VectorSubprioritySet(INT_VECTOR_DMA2, INT_SUBPRIORITY_LEVEL0);

/* Enable global interrupt */
SYS_INT_Enable();

/* This is the DMA System Service Initialization */
dmaInit.sidl = SYS_DMA_SIDL_DISABLE;
dmaModule = SYS_DMA_Initialize((SYS_MODULE_INIT*)&dmaInit);

/* The result of the DMA System Service initialization can be checked */
if(SYS_MODULE_OBJ_INVALID == dmaModule)
{
    /* DMA System Service initialization was not successful */
}

/* Initialize USART Driver Instance 0 */
usartModule1 = DRV_USART_Initialize(DRV_USART_0, (SYS_MODULE_INIT*)&usartInit);

/* The result of the driver initialization can be checked */
if(SYS_MODULE_OBJ_INVALID == usartModule1)
{
    /* There was an error in initialization. */
}
```

Opening the USART Driver

Describes how to open the USART Driver.

Description

To use the USART driver, the application must open the driver. This is done by calling the DRV_USART_Open function. Calling this function with DRV_IO_INTENT_NONBLOCKING will cause the driver to be opened in non blocking mode. The DRV_USART_Read and DRV_USART_Write functions when called by this client will be non blocking. . Calling this function with DRV_IO_INTENT_BLOCKING will cause the driver to be opened in blocking mode. The DRV_USART_Read and DRV_USART_Write functions when called by this client will be blocking.

If successful, the DRV_USART_Open function will return a valid handle to the driver. This handle records the association between the client and the driver instance that was opened. The DRV_USART_Open function may return DRV_HANDLE_INVALID in the situation where the driver is not ready to be opened. When this occurs, the application can try opening the driver again. Note that the open function may return an invalid handle in other (error) cases as well.

The following code shows an example of the driver being opened in different modes.

```
DRV_HANDLE usartHandle1, usartHandle2;
```

```
/* Client 1 opens the USART driver in non blocking mode */
usartHandle1 = DRV_USART_Open(DRV_USART_0, DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NONBLOCKING);

/* Check if the handle is valid */
if(DRV_HANDLE_INVALID == usartHandle1)
{
    /* The driver was not opened successfully. The client
     * can try opening it again */
}

/* Client 2 opens the USART driver in blocking mode */
usartHandle2 = DRV_USART_Open(DRV_USART_0, DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_BLOCKING);

/* Check if the handle is valid */
if(DRV_HANDLE_INVALID == usartHandle2)
{
    /* The driver was not opened successfully. The client
     * can try opening it again */
}

/* The client can also open the USART driver in read only mode
 * (DRV_IO_INTENT_READ), write only mode (DRV_IO_INTENT_WRITE)
 * and exclusive mode (DRV_IO_INTENT_EXCLUSIVE). If the driver
 * has been opened exclusively by a client, it cannot be opened
 * again by another client */
```

Byte Transfer Model

Describes the USART Driver byte transfer model.

Description

To use the byte transfer model, the DRV_USART_BYTE_MODEL_SUPPORT configuration macro should be true. The `drv_usart_byte_model.c` function should be included in the application project. The application cannot support the read/write and buffer queue data transfer model when the byte model is enabled.

The following code shows an example of how the DRV_USART_WriteByte function and the DRV_USART_ReadByte function are used.

```
 /* Client uses the a byte model API to write a byte*/
if(!DRV_USART_TransmitBufferIsFull(usartHandle1))
{
    byte = '1';
    DRV_USART_WriteByte(usartHandle1,byte);
}

/* Client waits until data is available and then reads
 * byte */
 while(DRV_USART_ReceiverBufferIsEmpty(usartHandle1));
        byte = DRV_USART_ReadByte(usartHandle1);
```

File I/O Type Read/Write Data Transfer Model

This topic describes the file I/O type read/write data transfer model.

Description

To use the file I/O type read/write data transfer model, the DRV_USART_READ_WRITE_MODEL_SUPPORT configuration macro should be 'true'. The file `drv_usart_read_write.c` file should be included in the application project. The driver can support the non-DMA buffer queue data transfer model along with the file I/O type read/write data transfer model. The byte transfer model and DMA buffer queue model cannot be enabled if the file I/O type read/write data transfer model is enabled.

The DRV_USART_Read and DRV_USART_Write function represent the file I/O type read/write data transfer model. The functional behavior of these API is affected by the mode in which the client opened the driver. If the client opened the driver in blocking mode these API will block. In blocking mode, the DRV_USART_Read and DRV_USART_Write functions will not return until the requested number of bytes have been read or written. When operating in a RTOS application, the application thread that has opened driver in blocking mode, will enter a blocked state when it calls DRV_USART_Write or DRV_USART_Read function. This will allow the RTOS scheduler to schedule other threads which are ready to run. If the client opened the driver in non-blocking mode these API will not block. In non-blocking mode, the DRV_USART_Read and DRV_USART_Write functions will return immediately with the amount of data that could be read or written.

Note: Do not open the driver in Blocking mode when the driver is configured for polling operation (DRV_USART_INTERRUPT_MODE is false) in a bare-metal (non-RTOS) application. This will cause the system to enter an infinite loop condition when the DRV_USART_Read or DRV_USART_Write function is called.

The following code shows an example of file I/O type read/write data transfer model usage when the driver is opened in Blocking mode.

```
/* This code shows the functionality of the DRV_USART_Write and
 * DRV_USART_Read function when the driver is opened in blocking mode */

DRV_HANDLE usartHandle1;
uint8_t myData[10];
size_t bytesProcessed;

/* The driver is opened in blocking mode */
usartHandle1 = DRV_USART_Open(DRV_USART_0, DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_BLOCKING);

/* Check if the driver was opened successfully */
if(DRV_HANDLE_INVALID == usartHandle1)
{
    /* The driver could not be opened successfully */
}

/* Transmit 10 bytes from the myData array. Function will not return until 10 bytes
 * have been accepted by the driver. This is because the client opened the driver
 * in blocking mode. */

bytesProcessed = DRV_USART_Write(usartHandle1, myData, 10);

/* Read 10 bytes from the myData array. Function will not return until all 10 bytes
 * have been received by the driver. This is because the client opened the driver
 * in blocking mode. */

bytesProcessed = DRV_USART_Read(usartHandle1, myData, 10);
```

In non-Blocking mode, the driver uses the internal USART hardware FIFO as storage. The DRV_USART_Read function checks if bytes are available in USART receive hardware FIFO. If bytes are available, these are read and the number of bytes read is returned. The DRV_USART_Write function checks if USART transmit hardware FIFO has empty location. If locations are empty, the bytes to be transmitted are queued up in the FIFO and the number of queued bytes is returned. In either case, the number of bytes read or written may be less than the number requested by the client. The client can, in such a case, call the DRV_USART_Read and/or the DRV_USART_Write functions again to process the pending bytes. The following code shows how this can be done.

```
/* This code shows the functionality of the DRV_USART_Write and
 * DRV_USART_Read functions when the driver is opened in non-blocking mode */

DRV_HANDLE usartHandle1;
uint8_t myData[10];
size_t bytesProcessed;

/* The driver is opened in non-blocking mode */
usartHandle1 = DRV_USART_Open(DRV_USART_0,
        DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NONBLOCKING);

/* Check if the driver was opened successfully */
if(DRV_HANDLE_INVALID == usartHandle1)
{
    /* The driver could not be opened successfully */
}

/* The following code call the DRV_USART_Write function
 * multiple times to write 10 bytes completely. Note how the
 * function return value is used to update the location of
 * user source data. */

bytesProcessed = 0;
do
{
    /* Write data to the USART and use the return value to
     * update the source data pointer and pending bytes number. */
    bytesProcessed += DRV_USART_Write(usartHandle1,
            myData + bytesProcessed, (10 - bytesProcessed));
```

```
} while(bytesProcessed < 10);

/* The following code calls the DRV_USART_Read function multiple times to read
 * 10 bytes completely. Note how the function return value is used to update the
 * location of user destination array. */

bytesProcessed = 0;
do
{
    /* Read data from the USART and use the return value to update the
     * destination pointer and pending bytes number. */
    bytesProcessed += DRV_USART_Read(usartHandle1,
            myData + bytesProcessed, (10 - bytesProcessed));

}while (bytesProcessed < 10);
```

Buffer Queue Transfer Model

This topic describes the buffer queue data transfer model.

Description

To use the buffer queue data transfer model, the DRV_USART_BUFFER_QUEUE_SUPPORT configuration macro should be true. The file, `drv_usart_buffer_queue.c`, should be included in the application project. If the DMA-enabled buffer queue model is required, the `drv_usart_buffer_queue_dma.c` file (*and not* the `drv_usart_buffer_queue.c`) should be included in the application project. The DMA and non-DMA buffer queue model API is the same. The driver can support the non-DMA buffer queue data transfer model along with the file I/O type read/write data transfer model. The byte transfer model cannot be enabled if the buffer queue data transfer model is enabled.

The DRV_USART_BufferAddRead and DRV_USART_BufferAddWrite functions represent the buffer queue data transfer model. These functions are always non-blocking. The Buffer Queue Data Transfer Model employs queuing of read and write request. Each driver instance contains a read and write queue. The size of the read queue is determined by the queueSizeRead member of the DRV_USART_INIT data structure. The size of the write queue is determined by the queueSizeWrite member of the DRV_USART_INIT data structure. The driver provides driver events (DRV_USART_BUFFER_EVENT) that indicates termination of the buffer requests.

When the driver is configured for Interrupt mode operation (that is defined and registered by the driver client), the buffer event handler executes in an interrupt context. Calling computationally intensive or hardware polling routines within the event handlers is not recommended. Calling interrupt unsafe functions in the event handler when the driver is configured for Interrupt mode could result in unpredictable system behavior.

When the driver adds request to the queue, it returns a buffer handle. This unique handle allows the client to track the request as it progresses through the queue. The buffer handle is returned with the buffer event and expires when the event associated with the buffer has been generated and the event handler returns. The following code shows an example of using the buffer queue data transfer model.

```
/* This code shows an example of using the
 * Buffer Queue Data Transfer Model. */
DRV_HANDLE usartHandle1;
uint8_t myData1[10], myData2[10];
uint8_t myData3[10], myData4[10];
size_t bytesProcessed;
DRV_USART_BUFFER_HANDLE bufferHandle1, bufferHandle2;
DRV_USART_BUFFER_HANDLE bufferHandle3, bufferHandle4;


/* The driver is opened in non blocking mode */
usartHandle1 = DRV_USART_Open(DRV_USART_0,
        DRV_IO_INTENT_READWRITE|DRV_IO_INTENT_NONBLOCKING);

/* Check if the driver was opened successfully */
if(DRV_HANDLE_INVALID == usartHandle1)
{
    /* The driver could not be opened successfully */
}

/* Register a Buffer Event Handler with USART driver.
 * This event handler function will be called whenever
 * there is a buffer event.  An application defined
 * context can also be specified. This is returned when
 * the event handler is called.
 * */
DRV_USART_BufferEventHandlerSet(usartHandle1,
        APP_USARTBufferEventHandler, NULL);
```

```
/* Queue up two buffers for transmit */
DRV_USART_BufferAddWrite(usartHandle1, &bufferHandle1, myData1, 10);
DRV_USART_BufferAddWrite(usartHandle1, &bufferHandle2, myData2, 10);

/* Queue up two buffers for receive */
DRV_USART_BufferAddRead(usartHandle1, &bufferHandle3, myData3, 10);
DRV_USART_BufferAddRead(usartHandle1, &bufferHandle4, myData4, 10);

/* This is application USART Driver Buffer Event Handler */

void APP_USARTBufferEventHandler(DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            /* This means the data was transferred */
            break;
        case DRV_USART_BUFFER_EVENT_ERROR:
            /* Error handling here. */
            break;
        default:
            break;
    }
}
```

Driver Tasks Routine

This topic describes the Driver "Task" routines.

Description

The USART driver contains three task routines, DRV_USART_TasksTransmit, DRV_USART_TasksReceive and DRV_USART_TasksError. These task routines implement the USART Driver state machines for transmit, receive and error related operations. If the driver is configured for polling operation, the required task routine should be called in SYS_Tasks routine of the system. If the driver is configured for interrupt mode of operation, the task routine should be called from the ISR. The following code shows an example of both.

```
/* The following code shows an example of
 * USART2 Interrupt Service Routine. This function
 * will be called when a USART2 interrupt occurs
 * and the driver is configured for interrupt mode
 * operation */

void __ISR ( _UART_2_VECTOR,ipl4 ) _InterruptHandler_USART ( void )
{
    /* usartModule1 is the System Module Object
     * that was returned by the DRV_USART_Initialize
     * function. */

    DRV_USART_TasksTransmit(usartModule1);
    DRV_USART_TasksReceive(usartModule1);
    DRV_USART_TasksError(usartModule1);

}

/* In case of Polled mode, the tasks routines are
 * invoked from the SYS_Tasks() routine. */

void SYS_Tasks(void)
{
    DRV_USART_TasksTransmit(usartModule1);
    DRV_USART_TasksReceive(usartModule1);
    DRV_USART_TasksError(usartModule1);
}

/* The SYS_Tasks routine is invoked from the main
 * application while(1) loop. */

while(1)
{
```

```
    SYS_Tasks();
}
```

Using the USART Driver with DMA

This topic provides information on using the USART Driver with DMA.

Description

To use the USART Driver with DMA, the following should be noted:

- Include `drv_usart_dma.c` in the project. Do not include `drv_usart.c`.
- Include `drv_usart_buffer_queue_dma.c` in the project. Do not include `drv_usart_buffer_queue.c`.
- Initialize the driver to use DMA. Refer to Initializing the USART Driver for details.
- Refer to the DMA System Service section for details on initializing and using the DMA system service in Polling or Interrupt mode
- The DRV_USART_INTERRUPT_MODE configuration macro should be set to 'true'
- Do not directly invoke the DRV_USART_TasksTransmit and DRV_USART_TasksReceive functions

Configuring the Library

Macros

	Name	Description
	DRV_USART_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_USART_INDEX	USART Static Index selection.
	DRV_USART_INTERRUPT_MODE	Macro controls interrupt based operation of the driver.
	DRV_USART_INTERRUPT_SOURCE_ERROR	Defines the error interrupt source for the static driver.
	DRV_USART_PERIPHERAL_ID	Configures the USART PLIB Module ID.
	DRV_USART_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_USART_BUFFER_QUEUE_SUPPORT	Specifies if the Buffer Queue support should be enabled.
	DRV_USART_BYTE_MODEL_SUPPORT	Specifies if the Byte Model support should be enabled.
	DRV_USART_INTERRUPT_SOURCE_RECEIVE	Defines the Receive interrupt source for the static driver.
	DRV_USART_INTERRUPT_SOURCE_RECEIVE_DMA	Defines the Receive DMA Channel interrupt source for the static driver.
	DRV_USART_INTERRUPT_SOURCE_TRANSMIT	Defines the Transmit interrupt source for the static driver.
	DRV_USART_INTERRUPT_SOURCE_TRANSMIT_DMA	Defines the Transmit DMA Channel interrupt source for the static driver.
	DRV_USART_QUEUE_DEPTH_COMBINED	Defines the number of entries of all queues in all instances of the driver.
	DRV_USART_READ_WRITE_MODEL_SUPPORT	Specifies if Read/Write Model support should be enabled.
	DRV_USART_RECEIVE_DMA	Defines the USART Driver Receive DMA Channel for the static driver.
	DRV_USART_TRANSMIT_DMA	Defines the USART Driver Transmit DMA Channel in case of static driver.
	DRV_USART_BAUD_RATE_IDXn	Specifies the USART Baud at which the USART driver is initialized.
	DRV_USART_BYTE_MODEL_BLOCKING	Enables or Disables DRV_USART_ByteWrite function blocking behavior.
	DRV_USART_BYTE_MODEL_CALLBACK	Enables or Disables Callback Feature of the Byte Transfer Model.
	DRV_USART_RCV_QUEUE_SIZE_IDXn	Sets the USART Driver Receive Queue Size while using the Buffer Queue Data Transfer Model.
	DRV_USART_XMIT_QUEUE_SIZE_IDXn	Sets the USART Driver Transmit Queue Size while using the Buffer Queue Data Transfer Model.

Description

The USART Driver requires the specification of compile-time configuration macros. These macros define resource usage, feature availability, and dynamic behavior of the driver. These configuration macros should be defined in the `system_config.h` file.

This header can be placed anywhere in the application specific folders and the path of this header needs to be presented to the include search for a successful build. Refer to the Applications Help section for more details.

Note: Initialization overrides are not supported in this version.

```
/* In this configuration example, the USART driver
 * must manage only on USART peripheral instance.
 * This macro can be greater than one if more
```

```
 * USART peripherals are needed. Not defining this
 * macro will cause the driver to be built in
 * static mode */
#define DRV_USART_INSTANCES_NUMBER 1

/* There will be 3 different client that use the
 * one instance of the USART peripheral. Note that
 * this macro configures the total (combined) number of clients
 * across all instance of the USART driver. Not defining
 * this macro will cause the driver to be configured
 * for single client operation */
#define DRV_USART_CLIENTS_NUMBER 3

/* USART Driver should be built for interrupt mode.
 * Set this to false for Polled mode operation */
#define DRV_USART_INTERRUPT_MODE true

/* Combined buffer queue depth is 5. Refer to the
 * description of the Buffer Queue data transfer model
 * and the DRV_USART_QUEUE_DEPTH_COMBINED macro
 * for more details on how this is configured. */
#define DRV_USART_QUEUE_DEPTH_COMBINED 5

/* Set this macro to true is Buffer Queue data
 * transfer model is to be enabled. */
#define DRV_USART_BUFFER_QUEUE_SUPPORT true

/* Set this macro to true if Byte by Byte data
 * transfer model is to be enabled. */
#define DRV_USART_BYTE_MODEL_SUPPORT false

/* Set this macro to true File IO type Read Write
 * data transfer model is to be enabled */
#define DRV_USART_READ_WRITE_MODEL_SUPPORT false
```

DRV_USART_CLIENTS_NUMBER Macro

Sets up the maximum number of clients that can be connected to any hardware instance.

File

drv_usart_config_template.h

C

```
#define DRV_USART_CLIENTS_NUMBER 4
```

Description

USART Client Count Configuration

This macro sets up the maximum number of clients that can be connected to any hardware instance. This value represents the total number of clients to be supported across all hardware instances. Therefore, if USART1 will be accessed by two clients and USART2 will accessed by three clients, this number should be 5. It is recommended that this value be set exactly equal to the number of expected clients, as client support consumes RAM memory space. If this macro is not defined and the DRV_USART_INSTANCES_NUMBER macro is not defined, the driver will be built for static - single client operation. If this macro is defined and the DRV_USART_INSTANCES_NUMBER macro is not defined, the driver will be built for static - multi client operation.

Remarks

None.

DRV_USART_INDEX Macro

USART Static Index selection.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INDEX DRV_USART_INDEX_2
```

Description

Index - Used for static drivers

USART Static Index selection for the driver object reference. This macro defines the driver index for static and static multi-client builds. For example, if this macro is set to DRV_USART_INDEX_2, the static driver APIs would be DRV_USART2_Initialize, DRV_USART2_Open, etc. When building static drivers, this macro should be different for each static build of the USART driver that needs to be included in the project.

Remarks

This index is required to make a reference to the driver object

DRV_USART_INTERRUPT_MODE Macro

Macro controls interrupt based operation of the driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INTERRUPT_MODE true
```

Description

USART Interrupt Mode Operation Control

This macro controls the interrupt based operation of the driver. The possible values are:

- true - Enables the interrupt mode
- false - Enables the polling mode

If the macro value is true, the Interrupt Service Routine (ISR) for the interrupt should be defined in the system. The DRV_USART_Tasks routine should be called in the ISR. While using the USART driver with DMA, this flag should always be true.

Remarks

None.

DRV_USART_INTERRUPT_SOURCE_ERROR Macro

Defines the error interrupt source for the static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INTERRUPT_SOURCE_ERROR INT_SOURCE_USART_2_ERROR
```

Description

Error Interrupt Source

This macro defines the Error interrupt source for the static driver. The interrupt source defined by this macro will override the errorInterruptSource member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the USART module error interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_USART_PERIPHERAL_ID Macro

Configures the USART PLIB Module ID.

File

drv_usart_config_template.h

C

```
#define DRV_USART_PERIPHERAL_ID USART_ID_2
```

Description

USART Peripheral Library Module ID

This macro configures the PLIB ID if the driver is built statically. This value will override the usartID member of the DRV_USART_INIT initialization data structure. In that when the driver is built statically, the usartID member of the DRV_USART_INIT data structure will be ignored by the driver initialization routine and this macro will be considered. This should be set to the PLIB ID of USART module (USART_ID_1, USART_ID_2, and so on).

Remarks

None.

DRV_USART_INSTANCES_NUMBER Macro

Sets up the maximum number of hardware instances that can be supported.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INSTANCES_NUMBER 2
```

Description

USART driver objects configuration

This macro sets up the maximum number of hardware instances that can be supported. It is recommended that this number be set exactly equal to the number of USART modules that are needed by the application, as hardware Instance support consumes RAM memory space. If this macro is not defined, the driver will be built statically.

Remarks

None

DRV_USART_BUFFER_QUEUE_SUPPORT Macro

Specifies if the Buffer Queue support should be enabled.

File

drv_usart_config_template.h

C

```
#define DRV_USART_BUFFER_QUEUE_SUPPORT true
```

Description

USART Driver Buffer Queue Support

This macro defines whether or not Buffer Queue support should be enabled. Setting this macro to true will enable buffer queue support and all buffer related driver function. The driver should be built along with the drv_usart_buffer_queue.c file, which contains the functional implementation for buffer queues. If buffer queue operation is enabled, the DRV_USART_BYTE_MODEL_SUPPORT function should not be true. If this macro is set to false, the behavior of the USART Driver Buffer Queue API is not defined. While using the USART driver with DMA, the driver supports Buffer Queue Data transfer model regardless of the value of this configuration macro.

Remarks

None.

DRV_USART_BYTE_MODEL_SUPPORT Macro

Specifies if the Byte Model support should be enabled.

File

drv_usart_config_template.h

C

```
#define DRV_USART_BYTE_MODEL_SUPPORT false
```

Description

USART Driver Byte Model Support

This macro defines whether or Byte Model support should be enabled. Setting this macro to true will enable byte model support and all byte operation related driver functions. The driver should be built along with the drv_usart_byte_model.c file, which contains the functional implementation for byte model operation. If byte model operation is enabled, the driver will not support buffer queue and read write models. The

behavior of the byte mode API when this macro is set to false is not defined.

Remarks

None.

DRV_USART_INTERRUPT_SOURCE_RECEIVE Macro

Defines the Receive interrupt source for the static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INTERRUPT_SOURCE_RECEIVE INT_SOURCE_USART_2_RECEIVE
```

Description

Receive Interrupt Source

This macro defines the Receive interrupt source for the static driver. The interrupt source defined by this macro will override the rxInterruptSource member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the USART module receive interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_USART_INTERRUPT_SOURCE_RECEIVE_DMA Macro

Defines the Receive DMA Channel interrupt source for the static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INTERRUPT_SOURCE_RECEIVE_DMA
```

Description

Receive DMA Channel Interrupt Source

This macro defines the Receive DMA Channel interrupt source for the static driver. The interrupt source defined by this macro will override the dmaInterruptReceive member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_USART_INTERRUPT_SOURCE_TRANSMIT Macro

Defines the Transmit interrupt source for the static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INTERRUPT_SOURCE_TRANSMIT INT_SOURCE_USART_2_TRANSMIT
```

Description

Transmit Interrupt Source

This macro defines the TX interrupt source for the static driver. The interrupt source defined by this macro will override the txInterruptSource member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the USART module transmit interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_USART_INTERRUPT_SOURCE_TRANSMIT_DMA Macro

Defines the Transmit DMA Channel interrupt source for the static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_INTERRUPT_SOURCE_TRANSMIT_DMA
```

Description

Transmit DMA Channel Interrupt Source

This macro defines the TX DMA Channel interrupt source for the static driver. The interrupt source defined by this macro will override the dmaInterruptTransmit member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel interrupt enumeration in the Interrupt PLIB for the microcontroller.

Remarks

None.

DRV_USART_QUEUE_DEPTH_COMBINED Macro

Defines the number of entries of all queues in all instances of the driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_QUEUE_DEPTH_COMBINED 16
```

Description

USART Driver Instance combined queue depth.

This macro defines the number of entries of all queues in all instances of the driver.

Each hardware instance supports a buffer queue for transmit and receive operations. The size of queue is specified either in driver initialization (for dynamic build) or by macros (for static build). The hardware instance transmit buffer queue will queue transmit buffers submitted by the DRV_USART_BufferAddWrite function. The hardware instance receive buffer queue will queue receive buffers submitted by the DRV_USART_BufferAddRead function.

A buffer queue will contain buffer queue entries, with each related to a BufferAdd request. This configuration macro defines the total number of buffer entries that will be available for use between all USART driver hardware instances. The buffer queue entries are allocated to individual hardware instances as requested by hardware instances. Once the request is processed, the buffer queue entry is free for use by other hardware instances.

The total number of buffer entries in the system determines the ability of the driver to service non blocking read and write requests. If a free buffer entry is not available, the driver will not add the request and will return an invalid buffer handle. The greater the number of buffer entries, the greater the ability of the driver to service and add requests to its queue. A hardware instance additionally can queue up as many buffer entries as specified by its transmit and receive buffer queue size.

For example, consider the case of static single client driver application where full duplex non blocking operation is desired without queuing, the minimum transmit queue depth and minimum receive queue depth should be 1. Therefore, the total number of buffer entries should be 2.

As another example, consider the case of a dynamic driver (i.e., two instances) where instance 1 will queue up to three write requests and up to two read requests, and instance 2 will queue up to two write requests and up to six read requests, the value of this macro should be: 13 (2 + 3 + 2 + 6).

Remarks

None.

DRV_USART_READ_WRITE_MODEL_SUPPORT Macro

Specifies if Read/Write Model support should be enabled.

File

drv_usart_config_template.h

C

```
#define DRV_USART_READ_WRITE_MODEL_SUPPORT true
```

Description

USART Driver Read Write Model Support

This macro defines whether or not Read Write Model support should be enabled. Setting this macro to true will enable read write model support and all read/write related driver functions. The driver should be built along with the drv_usart_read_write.c file, which contains the functional implementation for byte model operation. If read/write model operation is enabled, the DRV_USART_BYTE_MODEL_SUPPORT macro should not be true. The behavior of the Read Write Model API when this macro is set to false is not defined.

Remarks

None.

DRV_USART_RECEIVE_DMA Macro

Defines the USART Driver Receive DMA Channel for the static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_RECEIVE_DMA
```

Description

USART Driver Receive DMA Channel

This macro defines the USART Receive DMA Channel for the static driver. The DMA channel defined by this macro will override the dmaReceive member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel in the DMA PLIB for the microcontroller.

Remarks

None.

DRV_USART_TRANSMIT_DMA Macro

Defines the USART Driver Transmit DMA Channel in case of static driver.

File

drv_usart_config_template.h

C

```
#define DRV_USART_TRANSMIT_DMA
```

Description

USART Driver Transmit DMA Channel

This macro defines the USART Transmit DMA Channel for the static driver. The DMA channel defined by this macro will override the dmaTransmit member of the DRV_USB_INIT initialization data structure in the driver initialization routine. This value should be set to the DMA channel in the DMA PLIB for the microcontroller.

Remarks

None.

DRV_USART_BAUD_RATE_IDXn Macro

Specifies the USART Baud at which the USART driver is initialized.

File

drv_usart_config_template.h

C

```
#define DRV_USART_BAUD_RATE_IDXn
```

Description

USART Driver Baud Selection.

This configuration constant specifies the baud rate at which the USART Driver is initialized. This is the baud rate at which the USART module will operate when the driver initialization has completed. The driver client can call the DRV_USART_BaudSet function after opening the driver to change the USART baud rate after initialization has completed.

Remarks

This constant is automatically generated by MHC and its value is set to the value specified in USART Driver Baud Selection field.

DRV_USART_BYTE_MODEL_BLOCKING Macro

Enables or Disables DRV_USART_ByteWrite function blocking behavior.

File

drv_usart_config_template.h

C

```
#define DRV_USART_BYTE_MODEL_BLOCKING
```

Description

USART Driver Byte Write Blocking Behavior

This USART Driver MHC option controls the blocking behavior of the DRV_USART_ByteWrite function and is only applicable when the USART Driver Byte Transfer model is selected. Selecting this option will cause the DRV_USART_ByteWrite function to block until the byte has been written to the USART Transmit FIFO. Blocking behavior is enabled by default (to enable backward compatibility with previous versions of the driver). This option can be used for simple applications where interoperability with other MPLAB Harmony modules is not a design concern.

If the application uses several other MPLAB Harmony modules (Middleware, File System, etc.), it is recommended to disable this option and use the non-blocking DRV_USART_ByteWrite function. This requires the application to call the DRV_USART_TransmitBufferIsFull function to check if the byte can be written to the USART, as shown in the following code example.

```
if(!DRV_USART_TransmitBufferIsFull(usartHandle1))
{
    byte = '1';
    DRV_USART_WriteByte(usartHandle1,byte);
}
```

Using the non-blocking implementation results in improved application interoperability with other MPLAB Harmony modules.

Remarks

The DRV_USART_BYTE_MODEL_BLOCKING constant is specified for documentation purposes only. It does not affect the configuration of the driver.

DRV_USART_BYTE_MODEL_CALLBACK Macro

Enables or Disables Callback Feature of the Byte Transfer Model.

File

drv_usart_config_template.h

C

```
#define DRV_USART_BYTE_MODEL_CALLBACK
```

Description

USART Driver Byte Model Callback Feature.

This USART Driver MHC option controls the Callback feature of the Byte Transfer model. Selecting this option allows an application to register Byte Transfer Event Callback functions with the driver. These callback functions are invoked on the occurrence of Byte Transfer events. Callback functions can be registered to Byte Transmit, Byte Receive, and USART Error events, as shown in the following code example.

```
// This code shows how a callback function is
// registered for the Byte Receive event.
DRV_USART_ByteReceiveCallbackSet(DRV_USART_INDEX_0, APP_USARTReceiveEventHandler);

// Event Processing Technique. Event is received when
// a byte is received.

void APP_USARTReceiveEventHandler(const SYS_MODULE_INDEX index)
{
```

```
        // Byte has been Received. Handle the event.
        // Read byte using DRV_USART_ReadByte.
    }
```

When operating in Interrupt mode, the callback functions are invoked in an interrupt context. If this option is not selected, the application must use the DRV_USART_TransmitBufferIsFull, DRV_USART_ReceiverBufferIsEmpty, and DRV_USART_ErrorGet functions to check the status of Byte transmit or receive.

Remarks

The DRV_USART_BYTE_MODEL_CALLBACK constant is specified for documentation purposes only. It does not affect the configuration of the driver.

DRV_USART_RCV_QUEUE_SIZE_IDXn Macro

Sets the USART Driver Receive Queue Size while using the Buffer Queue Data Transfer Model.

File

drv_usart_config_template.h

C

```
#define DRV_USART_RCV_QUEUE_SIZE_IDXn
```

Description

USART Driver Receive Queue Size Selection.

This constant sets the USART Driver Receive queue size when using the Buffer Queue Data Transfer Model. It affects the queuing capacity of the DRV_USART_BufferAddRead function for the selected driver instance. For example, if this option is set to 5 for USART Driver 0, USART Driver 0 can then queue up to a maximum of five driver client receive buffer requests from any driver clients.

Therefore, if USART Driver 0 has two clients and if client 1 has queued up three buffers for receive, client 2 can only queue up to two buffers. If the client attempts to queue up more buffers, DRV_USART_BufferAddRead will not accept the request and will generate an invalid buffer handle (DRV_USART_BUFFER_HANDLE_INVALID).

Remarks

This constant is automatically generated by MHC and its value is set to the value specified in USART Driver Receive Queue Size field.

DRV_USART_XMIT_QUEUE_SIZE_IDXn Macro

Sets the USART Driver Transmit Queue Size while using the Buffer Queue Data Transfer Model.

File

drv_usart_config_template.h

C

```
#define DRV_USART_XMIT_QUEUE_SIZE_IDXn
```

Description

USART Driver Transmit Queue Size Selection.

This constant sets the USART Driver Transmit queue size when using the Buffer Queue Data Transfer Model. It affects the queuing capacity of the DRV_USART_BufferAddWrite function, for the selected driver instance. For example, if this option is set to 5 for USART Driver 0, USART Driver 0 can then queue up to a maximum of five driver client transmit buffer requests from any driver clients.

Therefore if USART Driver 0 has two clients and if client 1 has queued up three buffers for transmit, client 2 can only queue up to two buffers. If the client attempts to queue up more buffers, DRV_USART_BufferAddWrite will not accept the request and will generate an invalid buffer handle (DRV_USART_BUFFER_HANDLE_INVALID).

Remarks

This constant is automatically generated by MHC and its value is set to the value specified in USART Driver Transmit Queue Size field.

Building the Library

This section lists the files that are available in the USART Driver Library.

Description

This section list the files that are available in the \src folder of the USART Driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/usart`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_usart.h`	This file should be included by any .c file which accesses the USART Driver API. This one file contains the prototypes for all driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/drv_usart.c`	This file should always be included in the project when using the USART Driver.
`/src/dynamic/drv_usart_dma.c`	This file should always be included in the project when using the USART driver with DMA.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
`/src/dynamic/drv_usart_byte_model.c`	This file should be included in the project if the USART Driver Byte Model API is required.
`/src/dynamic/drv_usart_buffer_queue.c`	This file should be included in the project if the USART Driver Buffer Queue Model API (without DMA) is required.
`/src/dynamic/drv_usart_read_write.c`	This file should be included in the project if the USART Driver Read Write Model API is required.
`/src/dynamic/drv_usart_buffer_queue_dma.c`	This file should be included in the project if the USART Driver Buffer Queue Model API with DMA is required.

Module Dependencies

The USART Driver Library depends on the following modules:

- Interrupt System Service Library
- DMA System Service Library (if USART Driver is configured to use DMA)

Library Interface

a) System Functions

	Name	Description
	DRV_USART_Initialize	Initializes the USART instance for the specified driver index. **Implementation:** Static/Dynamic
	DRV_USART_Deinitialize	Deinitializes the specified instance of the USART driver module. **Implementation:** Static/Dynamic
	DRV_USART_Status	Gets the current status of the USART driver module. **Implementation:** Static/Dynamic
	DRV_USART_TasksReceive	Maintains the driver's receive state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_USART_TasksTransmit	Maintains the driver's transmit state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_USART_TasksError	Maintains the driver's error state machine and implements its ISR. **Implementation:** Static/Dynamic

b) Core Client Functions

	Name	Description
	DRV_USART_Open	Opens the specified USART driver instance and returns a handle to it. **Implementation:** Static/Dynamic
	DRV_USART_Close	Closes an opened-instance of the USART driver. **Implementation:** Static/Dynamic
	DRV_USART_ClientStatus	Gets the current client-specific status the USART driver. **Implementation:** Static/Dynamic
	DRV_USART_ErrorGet	This function returns the error(if any) associated with the last client request. **Implementation:** Static/Dynamic

c) Communication Management Client Functions

	Name	Description
	DRV_USART_BaudSet	This function changes the USART module baud to the specified value. **Implementation:** Static/Dynamic
	DRV_USART_LineControlSet	This function changes the USART module line control to the specified value. **Implementation:** Static/Dynamic

d) Buffer Queue Read/Write Client Functions

	Name	Description
	DRV_USART_BufferAddRead	Schedule a non-blocking driver read operation. **Implementation:** Static/Dynamic
	DRV_USART_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Static/Dynamic
	DRV_USART_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Static/Dynamic
	DRV_USART_BufferProcessedSizeGet	This API will be deprecated and not recommended to use. Use DRV_USART_BufferCompletedBytesGet to get the number of bytes processed for the specified buffer.
	DRV_USART_AddressedBufferAddWrite	Schedule a non-blocking addressed driver write operation. **Implementation:** Dynamic
	DRV_USART_BufferCompletedBytesGet	Returns the number of bytes that have been processed for the specified buffer. **Implementation:** Static/Dynamic
	DRV_USART_BufferRemove	Removes a requested buffer from the queue. **Implementation:** Static/Dynamic

e) File I/O Type Read/Write Functions

	Name	Description
	DRV_USART_Read	Reads data from the USART. **Implementation:** Static/Dynamic
	DRV_USART_Write	Writes data to the USART. **Implementation:** Static/Dynamic

f) Byte Transfer Functions

	Name	Description
	DRV_USART_ReadByte	Reads a byte of data from the USART. **Implementation:** Static/Dynamic
	DRV_USART_WriteByte	Writes a byte of data to the USART. **Implementation:** Static/Dynamic
	DRV_USART_TransmitBufferSizeGet	Returns the size of the transmit buffer. **Implementation:** Static/Dynamic
	DRV_USART_ReceiverBufferSizeGet	Returns the size of the receive buffer. **Implementation:** Static/Dynamic
	DRV_USART_TransferStatus	Returns the transmitter and receiver transfer status. **Implementation:** Static/Dynamic
	DRV_USART_TransmitBufferIsFull	Provides the status of the driver's transmit buffer. **Implementation:** Static/Dynamic

	DRV_USART_ReceiverBufferIsEmpty	Provides the status of the driver's receive buffer. **Implementation:** Static/Dynamic
	DRV_USART_ByteErrorCallbackSet	Registers callback to handle for byte error events.
	DRV_USART_ByteReceiveCallbackSet	Registers receive callback function for byte receive event.
	DRV_USART_ByteTransmitCallbackSet	Registers a callback function for byte transmit event.

g) Data Types and Constants

	Name	Description
	DRV_USART_CLIENT_STATUS	Defines the client-specific status of the USART driver.
	DRV_USART_INIT	Defines the data required to initialize or reinitialize the USART driver
	DRV_USART_INIT_FLAGS	Flags identifying features that can be enabled when the driver is initialized.
	DRV_USART_TRANSFER_STATUS	Specifies the status of the receive or transmit
	DRV_USART_INDEX_0	USART driver index definitions
	DRV_USART_INDEX_1	This is macro DRV_USART_INDEX_1.
	DRV_USART_INDEX_2	This is macro DRV_USART_INDEX_2.
	DRV_USART_INDEX_3	This is macro DRV_USART_INDEX_3.
	DRV_USART_INDEX_4	This is macro DRV_USART_INDEX_4.
	DRV_USART_INDEX_5	This is macro DRV_USART_INDEX_5.
	DRV_USART_BAUD_SET_RESULT	Identifies the handshaking modes supported by the USART driver.
	DRV_USART_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_USART_BUFFER_EVENT_HANDLER	Pointer to a USART Driver Buffer Event handler function
	DRV_USART_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.
	DRV_USART_ERROR	Defines the possible errors that can occur during driver operation.
	DRV_USART_LINE_CONTROL_SET_RESULT	Identifies the results of the baud set function.
	DRV_USART_OPERATION_MODE_DATA	Defines the initialization data required for different operation modes of USART.
	DRV_USART_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_USART_COUNT	Number of valid USART drivers
	DRV_USART_READ_ERROR	USART Driver Read Error.
	DRV_USART_WRITE_ERROR	USART Driver Write Error.
	DRV_USART_LINE_CONTROL	Identifies the line control modes supported by the USART driver.
	DRV_USART_OPERATION_MODE	Identifies the modes of the operation of the USART module
	DRV_USART_BYTE_EVENT_HANDLER	Pointer to a USART Driver Byte Event handler function
	DRV_USART_BUFFER_RESULT	Identifies the possible result of the buffer processing.

Description

This section describes the functions of the USART Driver Library.

Refer to each section for a detailed description.

a) System Functions

DRV_USART_Initialize Function

Initializes the USART instance for the specified driver index.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
SYS_MODULE_OBJ DRV_USART_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, returns SYS_MODULE_OBJ_INVALID.

Description

This routine initializes the USART driver instance for the specified driver index, making it ready for clients to open and use it. The initialization data

is specified by the init parameter. The initialization may fail if the number of driver objects allocated are insufficient or if the specified driver instance is already initialized. The driver instance index is independent of the USART module ID. For example, driver instance 0 can be assigned to USART2. If the driver is built statically, then some of the initialization parameters are overridden by configuration macros. Refer to the description of the DRV_USART_INIT data structure for more details on which members on this data structure are overridden.

Remarks

This routine must be called before any other USART routine is called.

This routine should only be called once during system initialization unless DRV_USART_Deinitialize is called to deinitialize the driver instance. This routine will NEVER block for hardware access.

Preconditions

None.

Example

```
// The following code snippet shows an example USART driver initialization.
// The driver is initialized for normal mode and a baud of 300. The
// receive queue size is set to 2 and transmit queue size is set to 3.

DRV_USART_INIT              usartInit;
SYS_MODULE_OBJ              objectHandle;

usartInit.baud  = 300;
usartInit.mode  = DRV_USART_OPERATION_MODE_NORMAL;
usartInit.flags = DRV_USART_INIT_FLAG_NONE;
usartInit.usartID   = USART_ID_2;
usartInit.brgClock  = 80000000;
usartInit.handshake = DRV_USART_HANDSHAKE_NONE;
usartInit.lineControl       = DRV_USART_LINE_CONTROL_8NONE1;
usartInit.interruptError    = INT_SOURCE_USART_2_ERROR;
usartInit.interruptReceive  = INT_SOURCE_USART_2_RECEIVE;
usartInit.queueSizeReceive  = 2;
usartInit.queueSizeTransmit = 3;
usartInit.interruptTransmit = INT_SOURCE_USART_2_TRANSMIT;
usartInit.moduleInit.value  = SYS_MODULE_POWER_RUN_FULL;

objectHandle = DRV_USART_Initialize(DRV_USART_INDEX_1, (SYS_MODULE_INIT*)&usartInitData);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Identifier for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the driver.

Function

SYS_MODULE_OBJ DRV_USART_Initialize

(

const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init

)

DRV_USART_Deinitialize Function

Deinitializes the specified instance of the USART driver module.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the USART driver module, disabling its operation (and any hardware). Invalidates all the internal data.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function DRV_USART_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from DRV_USART_Initialize
SYS_STATUS          status;

DRV_USART_Deinitialize(object);

status = DRV_USART_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_USART_Initialize routine

Function

void DRV_USART_Deinitialize(SYS_MODULE_OBJ object)

DRV_USART_Status Function

Gets the current status of the USART driver module.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
SYS_STATUS DRV_USART_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized

Description

This routine provides the current status of the USART driver module.

Remarks

A driver can opened only when its status is SYS_STATUS_READY.

Preconditions

Function DRV_USART_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USART_Initialize
SYS_STATUS          usartStatus;

usartStatus = DRV_USART _Status(object);
```

```
if (SYS_STATUS_READY == usartStatus)
{
    // This means the driver can be opened using the
    // DRV_USART_Open() function.
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from the DRV_USART_Initialize routine

Function

SYS_STATUS DRV_USART_Status(SYS_MODULE_OBJ object)

DRV_USART_TasksReceive Function

Maintains the driver's receive state machine and implements its ISR.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_TasksReceive(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal receive state machine and implement its receive ISR for interrupt-driven implementations. In polling mode, this function should be called from the SYS_Tasks function. In interrupt mode, this function should be called in the receive interrupt service routine of the USART that is associated with this USART driver hardware instance.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USART_Initialize

while (true)
{
    DRV_USART_TasksReceive (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USART_Initialize)

Function

void DRV_USART_TasksReceive (SYS_MODULE_OBJ object);

DRV_USART_TasksTransmit Function

Maintains the driver's transmit state machine and implements its ISR.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_TasksTransmit(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal transmit state machine and implement its transmit ISR for interrupt-driven implementations. In polling mode, this function should be called from the SYS_Tasks function. In interrupt mode, this function should be called in the transmit interrupt service routine of the USART that is associated with this USART driver hardware instance.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USART_Initialize

while (true)
{
    DRV_USART_TasksTransmit (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USART_Initialize)

Function

void DRV_USART_TasksTransmit (SYS_MODULE_OBJ object);

DRV_USART_TasksError Function

Maintains the driver's error state machine and implements its ISR.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_TasksError(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the driver's internal error state machine and implement its error ISR for interrupt-driven implementations. In polling mode, this function should be called from the SYS_Tasks function. In interrupt mode, this function should be called in the error interrupt service routine of the USART that is associated with this USART driver hardware instance.

Remarks

This routine is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from DRV_USART_Initialize

while (true)
{
    DRV_USART_TasksError (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from DRV_USART_Initialize)

Function

void DRV_USART_TasksError (SYS_MODULE_OBJ object);

b) Core Client Functions

DRV_USART_Open Function

Opens the specified USART driver instance and returns a handle to it.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_HANDLE DRV_USART_Open(const SYS_MODULE_INDEX index, const DRV_IO_INTENT ioIntent);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is DRV_HANDLE_INVALID. Error can occur

- if the number of client objects allocated via DRV_USART_CLIENTS_NUMBER is insufficient.
- if the client is trying to open the driver but driver has been opened exclusively by another client.
- if the driver hardware instance being opened is not initialized or is invalid.
- if the client is trying to open the driver exclusively, but has already been opened in a non exclusive mode by another client.
- if the driver is not ready to be opened, typically when the initialize routine has not completed execution.

Description

This routine opens the specified USART driver instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver. The ioIntent parameter defines how the client interacts with this driver instance.

The DRV_IO_INTENT_BLOCKING and DRV_IO_INTENT_NONBLOCKING ioIntent options additionally affect the behavior of the DRV_USART_Read and DRV_USART_Write functions. If the ioIntent is DRV_IO_INTENT_NONBLOCKING, then these function will not block even if the required amount of data could not be processed. If the ioIntent is DRV_IO_INTENT_BLOCKING, these functions will block until the required amount of data is processed. If the driver is configured for polling and bare-metal operation, it will not support DRV_IO_INTENT_BLOCKING. The driver will operation will always be non-blocking.

If ioIntent is DRV_IO_INTENT_READ, the client will only be able to read from the driver. If ioIntent is DRV_IO_INTENT_WRITE, the client will only be able to write to the driver. If the ioIntent is DRV_IO_INTENT_READWRITE, the client will be able to do both, read and write.

Specifying a DRV_IO_INTENT_EXCLUSIVE will cause the driver to provide exclusive access to this client. The driver cannot be opened by any other client.

Remarks

The handle returned is valid until the DRV_USART_Close routine is called. This routine will NEVER block waiting for hardware.If the requested intent flags are not supported, the routine will return DRV_HANDLE_INVALID. This function is thread safe in a RTOS application.

Preconditions

Function DRV_USART_Initialize must have been called before calling this function.

Example

```
DRV_HANDLE handle;

handle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
if (DRV_HANDLE_INVALID == handle)
{
    // Unable to open the driver
    // May be the driver is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
intent	Zero or more of the values from the enumeration DRV_IO_INTENT "ORed" together to indicate the intended use of the driver. See function description for details.

Function

```
  DRV_HANDLE DRV_USART_Open
(
const    SYS_MODULE_INDEX index,
const    DRV_IO_INTENT ioIntent
)
```

DRV_USART_Close Function

Closes an opened-instance of the USART driver.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_Close(const DRV_HANDLE handle);
```

Returns

None.

Description

This routine closes an opened-instance of the USART driver, invalidating the handle. Any buffers in the driver queue that were submitted by this client will be removed. After calling this routine, the handle passed in "handle" must not be used with any of the remaining driver routines (with one possible exception described in the "Remarks" section). A new handle must be obtained by calling DRV_USART_Open before the caller may use the driver again

Remarks

Usually there is no need for the client to verify that the Close operation has completed. The driver will abort any ongoing operations when this routine is called. However, if it requires additional time to do so in a non-blocking environment, it will still return from the Close operation but the handle is now a zombie handle. The client can only call the DRV_USART_ClientStatus on a zombie handle to track the completion of the Close operation. The DRV_USART_ClientStatus routine will return DRV_CLIENT_STATUS_CLOSED when the close operation has completed.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE handle;  // Returned from DRV_USART_Open

DRV_USART_Close(handle);
```

```
// After this point, the handle cannot be used with any other function
// except the DRV_USART_ClientStatus function, which can be used to query
// the success status of the DRV_USART_Close function.

while(DRV_USART_CLIENT_STATUS_CLOSED != DRV_USART_ClientStatus(handle));
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

void DRV_USART_Close(DRV_Handle handle)

DRV_USART_ClientStatus Function

Gets the current client-specific status the USART driver.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_USART_CLIENT_STATUS DRV_USART_ClientStatus(DRV_HANDLE handle);
```

Returns

A DRV_USART_CLIENT_STATUS value describing the current status of the driver.

Description

This function gets the client-specific status of the USART driver associated with the given handle. This function can be used to check the status of client after the DRV_USART_Close() function has been called.

Remarks

This function will not block for hardware access and will immediately return the current status. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_USART_Initialize function must have been called.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE              handle;  // Returned from DRV_USART_Open
DRV_USART_CLIENT_STATUS   status;

status = DRV_USART_ClientStatus(handle);
if( DRV_USART_CLIENT_STATUS_CLOSED != status )
{
    // The client had not closed.
}
```

Parameters

Parameters	Description
handle	Handle returned from the driver's open function.

Function

DRV_USART_CLIENT_STATUS DRV_USART_ClientStatus(DRV_HANDLE handle)

DRV_USART_ErrorGet Function

This function returns the error(if any) associated with the last client request.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_USART_ERROR DRV_USART_ErrorGet(const DRV_HANDLE client);
```

Returns

A DRV_USART_ERROR type indicating last known error status.

Description

This function returns the error(if any) associated with the last client request. DRV_USART_Read and DRV_USART_Write will update the client error status when these functions return DRV_USART_TRANSFER_ERROR. If the driver send a DRV_USART_BUFFER_EVENT_ERROR to the client, the client can call this function to know the error cause. The error status will be updated on every operation and should be read frequently (ideally immediately after the driver operation has completed) to know the relevant error status.

Remarks

It is the client's responsibility to make sure that the error status is obtained frequently. The driver will update the client error status regardless of whether this has been examined by the client. This function is thread safe when used in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver. This is done once.

DRV_USART_BufferEventHandlerSet( myUSARTHandle, APP_USARTBufferEventHandle,
                                 (uintptr_t)&myAppObj );

bufferHandle = DRV_USART_BufferAddRead( myUSARThandle,
                                        myBuffer, MY_BUFFER_SIZE );

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler( DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle )
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_SUCCESS:

            // This means the data was transferred.
            break;

        case DRV_USART_BUFFER_EVENT_FAILURE:
```

```
            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred. We can also find
            // the error cause.

            processedBytes = DRV_USART_BufferCompletedBytesGet(bufferHandle);
            if(DRV_USART_ERROR_RECEIVE_OVERRUN == DRV_USART_ErrorGet(myUSARTHandle))
            {
                // There was an receive over flow error.
                // Do error handling here.
            }

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferhandle	Handle of the buffer of which the processed number of bytes to be obtained.

Function

DRV_USART_ERROR DRV_USART_ErrorGet(DRV_HANDLE client);

c) Communication Management Client Functions

DRV_USART_BaudSet Function

This function changes the USART module baud to the specified value.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_USART_BAUD_SET_RESULT DRV_USART_BaudSet(const DRV_HANDLE client, uint32_t baud);
```

Returns

None.

Description

This function changes the USART module baud to the specified value. Any queued buffer requests will be processed at the updated baud. The USART driver operates at the baud specified in DRV_USART_Initialize function unless the DRV_USART_BaudSet function is called to change the baud.

Remarks

The implementation of this function, in this release of the driver, changes the baud immediately. This may interrupt on-going data transfer. It is recommended that the driver be opened exclusively if this function is to be called. This function is thread safe when used in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

DRV_USART_BaudSet(myUSARTHandle, 9600);
```

Parameters

Parameters	Description
handle	client handle returned by DRV_USART_Open function.
baud	desired baud.

Function

void DRV_USART_BaudSet(DRV_HANDLE client, uint32_t baud);

DRV_USART_LineControlSet Function

This function changes the USART module line control to the specified value.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_USART_LINE_CONTROL_SET_RESULT DRV_USART_LineControlSet(const DRV_HANDLE client, const
DRV_USART_LINE_CONTROL lineControl);
```

Returns

DRV_USART_LINE_CONTROL_SET_SUCCESS if the function was successful. Returns DRV_HANDLE_INVALID if the client handle is not valid.

Description

This function changes the USART module line control parameters to the specified value. Any queued buffer requests will be processed at the updated line control parameters. The USART driver operates at the line control parameters specified in DRV_USART_Initialize function unless the DRV_USART_LineControlSet function is called to change the line control parameters.

Remarks

The implementation of this function, in this release of the driver, changes the line control immediately. This may interrupt on-going data transfer. It is recommended that the driver be opened exclusively if this function is to be called. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

DRV_USART_LineControlSet(myUSARTHandle, DRV_USART_LINE_CONTROL_8NONE1);
```

Parameters

Parameters	Description
handle	client handle returned by DRV_USART_Open function.
lineControl	line control parameters.

Function

void DRV_USART_LineControlSet

(

DRV_HANDLE client,

DRV_USART_LINE_CONTROL lineControl

);

d) Buffer Queue Read/Write Client Functions

DRV_USART_BufferAddRead Function

Schedule a non-blocking driver read operation.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_BufferAddRead(const DRV_HANDLE handle, DRV_USART_BUFFER_HANDLE * const bufferHandle, void *
buffer, const size_t size);
```

Returns

The buffer handle is returned in the bufferHandle argument. This is DRV_USART_BUFFER_HANDLE_INVALID if the request was not successful.

Description

This function schedules a non-blocking read operation. The function returns with a valid buffer handle in the bufferHandle argument if the read request was scheduled successfully. The function adds the request to the hardware instance receive queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. The function returns DRV_USART_BUFFER_HANDLE_INVALID in the bufferHandle argument:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the buffer size is 0
- if the read queue size is full or queue depth is insufficient.
- if the driver handle is invalid

If the requesting client registered an event callback with the driver, the driver will issue a DRV_USART_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully of DRV_USART_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the USART Driver Buffer Event Handler that is registered by the client. It should not be called in the event handler associated with another USART driver instance. It should not be called directly in an ISR.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART device instance and the DRV_USART_Status must have returned SYS_STATUS_READY.

DRV_USART_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_USART_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_USART_BUFFER_HANDLE bufferHandle;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver

DRV_USART_BufferEventHandlerSet(myUSARTHandle,
                APP_USARTBufferEventHandler, (uintptr_t)&myAppObj);

DRV_USART_BufferAddRead(myUSARThandle, &bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler(DRV_USART_BUFFER_EVENT event,
```

```
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_USART_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the communication channel as returned by the DRV_USART_Open function.
buffer	Buffer where the received data will be stored.
size	Buffer size in bytes.

Function

void DRV_USART_BufferAddRead
(
const DRV_HANDLE handle,
DRV_USART_BUFFER_HANDLE * bufferHandle,
void * buffer,
const size_t size
)

DRV_USART_BufferAddWrite Function

Schedule a non-blocking driver write operation.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_BufferAddWrite(const DRV_HANDLE handle, DRV_USART_BUFFER_HANDLE * bufferHandle, void *
buffer, const size_t size);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_USART_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking write operation. The function returns with a valid buffer handle in the bufferHandle argument if the write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. On returning, the bufferHandle parameter may be DRV_USART_BUFFER_HANDLE_INVALID for the following reasons:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read-only
- if the buffer size is 0

- if the transmit queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_USART_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or a DRV_USART_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the USART Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another USART driver instance. It should not otherwise be called directly in an ISR.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART device instance and the DRV_USART_Status must have returned SYS_STATUS_READY.

DRV_USART_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_USART_Open call.

Example

```
MY_APP_OBJ myAppObj;
uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver

DRV_USART_BufferEventHandlerSet(myUSARTHandle,
                APP_USARTBufferEventHandler, (uintptr_t)&myAppObj);

DRV_USART_BufferAddWrite(myUSARThandle, &bufferHandle,
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler(DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_USART_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the communication channel as return by the DRV_USART_Open function.
bufferHandle	Pointer to an argument that will contain the return buffer handle.

buffer	Data to be transmitted.
size	Buffer size in bytes.

Function

void DRV_USART_BufferAddWrite
(
const DRV_HANDLE handle,
DRV_USART_BUFFER_HANDLE * bufferHandle,
void * buffer,
size_t size
);

DRV_USART_BufferEventHandlerSet Function

Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_BufferEventHandlerSet(const DRV_HANDLE handle, const DRV_USART_BUFFER_EVENT_HANDLER
eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. When a client calls either the DRV_USART_BufferAddRead or DRV_USART_BufferAddWrite function, it is provided with a handle identifying the buffer that was added to the driver's buffer queue. The driver will pass this handle back to the client by calling "eventHandler" function when the buffer transfer has completed.

The event handler should be set before the client performs any "buffer add" operations that could generate events. The event handler once set, persists until the client closes the driver or sets another event handler (which could be a "NULL" pointer to indicate no callback).

Remarks

If the client does not want to be notified when the queued buffer transfer has completed, it does not need to register a callback. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_USART_BUFFER_HANDLE bufferHandle;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver. This is done once

DRV_USART_BufferEventHandlerSet( myUSARTHandle, APP_USARTBufferEventHandle,
                                 (uintptr_t)&myAppObj );

DRV_USART_BufferAddRead(myUSARThandle, &bufferHandle
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
```

```
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler(DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE handle, uintptr_t context)
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) context;

    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_SUCCESS:

            // This means the data was transferred.
            break;

        case DRV_USART_BUFFER_EVENT_FAILURE:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
eventHandler	Pointer to the event handler function.
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void DRV_USART_BufferEventHandlerSet

(

const DRV_HANDLE handle,

const DRV_USART_BUFFER_EVENT_HANDLER eventHandler,

const uintptr_t context

)

DRV_USART_BufferProcessedSizeGet Function

This API will be deprecated and not recommended to use. Use DRV_USART_BufferCompletedBytesGet to get the number of bytes processed for the specified buffer.

File

drv_usart.h

C

```
size_t DRV_USART_BufferProcessedSizeGet(DRV_USART_BUFFER_HANDLE bufferHandle);
```

Returns

None.

Description

None.

Remarks

None.

Preconditions

None.

Example

None.

Function

size_t DRV_USART_BufferProcessedSizeGet

(

DRV_USART_BUFFER_HANDLE bufferHandle

);

DRV_USART_AddressedBufferAddWrite Function

Schedule a non-blocking addressed driver write operation.

Implementation: Dynamic

File

drv_usart.h

C

```
void DRV_USART_AddressedBufferAddWrite(const DRV_HANDLE hClient, DRV_USART_BUFFER_HANDLE * bufferHandle,
uint8_t address, void * source, size_t nWords);
```

Returns

The bufferHandle parameter will contain the return buffer handle. This will be DRV_USART_BUFFER_HANDLE_INVALID if the function was not successful.

Description

This function schedules a non-blocking addressed write operation. The function returns with a valid buffer handle in the bufferHandle argument if the addressed write request was scheduled successfully. The function adds the request to the hardware instance transmit queue and returns immediately. While the request is in the queue, the application buffer is owned by the driver and should not be modified. On returning, the bufferHandle parameter may be DRV_USART_BUFFER_HANDLE_INVALID for the following reasons:

- if a buffer could not be allocated to the request
- if the input buffer pointer is NULL
- if the client opened the driver for read-only
- if the buffer size is 0
- if the transmit queue is full or the queue depth is insufficient

If the requesting client registered an event callback with the driver, the driver will issue a DRV_USART_BUFFER_EVENT_COMPLETE event if the buffer was processed successfully or a DRV_USART_BUFFER_EVENT_ERROR event if the buffer was not processed successfully.

Remarks

This function is thread safe in a RTOS application. It can be called from within the USART Driver Buffer Event Handler that is registered by this client. It should not be called in the event handler associated with another USART driver instance. It should not otherwise be called directly in an ISR.

The source buffer should be a 16-bit word aligned buffer. The 9th bit of the higher byte 16-bit buffer is used to indicate data/address.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART device instance and the DRV_USART_Status must have returned SYS_STATUS_READY.

DRV_USART_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_USART_Open call.

The operation mode of the driver must be DRV_USART_OPERATION_MODE_ADDRESSED.

Example

```
MY_APP_OBJ myAppObj;
uint16_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;
```

```
uint8_t clientAddress;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver

clientAddress = 0x60;
DRV_USART_BufferEventHandlerSet(myUSARTHandle,
                APP_USARTBufferEventHandler, (uintptr_t)&myAppObj);

DRV_USART_AddressedBufferAddWrite(myUSARThandle, &bufferHandle, clientAddress
                                    myBuffer, MY_BUFFER_SIZE);

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler(DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle)
{
    // contextHandle points to myAppObj.

    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case DRV_USART_BUFFER_EVENT_ERROR:

            // Error handling here.

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
hClient	Handle of the communication channel as return by the DRV_USART_Open function.
bufferHandle	Pointer to an argument that will contain the return buffer handle.
address	Address of the receiver client
source	Data to be transmitted.
size	Buffer size in 16-bit words.

Function

void DRV_USART_AddressedBufferAddWrite
(
const DRV_HANDLE hClient,
DRV_USART_BUFFER_HANDLE * bufferHandle,
uint8_t address,
void * source,
size_t nWords
);

DRV_USART_BufferCompletedBytesGet Function

Returns the number of bytes that have been processed for the specified buffer.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
size_t DRV_USART_BufferCompletedBytesGet(DRV_USART_BUFFER_HANDLE bufferHandle);
```

Returns

Returns the number of bytes that have been processed for this buffer.

Returns DRV_USART_BUFFER_HANDLE_INVALID for an invalid or an expired buffer handle.

Description

This function returns number of bytes that have been processed for the specified buffer. The client can use this function, in a case where the buffer has terminated due to an error, to obtain the number of bytes that have been processed. Or in any other use case.

Remarks

This function is thread safe when used in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Either the DRV_USART_BufferAddRead or DRV_USART_BufferAddWrite function must have been called and a valid buffer handle returned.

Example

```
// myAppObj is an application specific object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver. This is done once

DRV_USART_BufferEventHandlerSet( myUSARTHandle, APP_USARTBufferEventHandle,
                                 (uintptr_t)&myAppObj );

bufferHandle = DRV_USART_BufferAddRead( myUSARTHandle,
                                        myBuffer, MY_BUFFER_SIZE );

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler( DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle )
{
    // The context handle was set to an application specific
    // object. It is now retrievable easily in the event handler.
    MY_APP_OBJ myAppObj = (MY_APP_OBJ *) contextHandle;
    size_t processedBytes;

    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_SUCCESS:
```

```
            // This means the data was transferred.
            break;

        case DRV_USART_BUFFER_EVENT_FAILURE:

            // Error handling here.
            // We can find out how many bytes were processed in this
            // buffer before the error occurred.

            processedBytes = DRV_USART_BufferCompletedBytesGet(bufferHandle);

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
bufferhandle	Handle for the buffer of which the processed number of bytes to be obtained.

Function

size_t DRV_USART_BufferCompletedBytesGet

(

DRV_USART_BUFFER_HANDLE bufferHandle

);

DRV_USART_BufferRemove Function

Removes a requested buffer from the queue.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_USART_BUFFER_RESULT DRV_USART_BufferRemove(DRV_USART_BUFFER_HANDLE bufferHandle);
```

Returns

DRV_USART_BUFFER_RESULT_HANDLE_INVALID - Buffer handle is invalid.

DRV_USART_BUFFER_RESULT_HANDLE_EXPIRED - Buffer handle is expired.

DRV_USART_BUFFER_RESULT_REMOVED_SUCCESFULLY - Buffer is removed from the queue successfully.

DRV_USART_BUFFER_RESULT_REMOVAL_FAILED - Failed to remove buffer from the queue because of mutex timeout in RTOS environment.

Description

This function removes a specified buffer from the queue. The client can use this function to delete

1. An unwated stalled buffer.
2. Queued buffers on timeout.

or in any other use case.

Remarks

This function is thread safe when used in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Either the DRV_USART_BufferAddRead or DRV_USART_BufferAddWrite function must have been called and a valid buffer handle returned.

Example

```
// myAppObj is an application specific object.
```

```
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];
DRV_BUFFER_HANDLE bufferHandle;

// myUSARTHandle is the handle returned
// by the DRV_USART_Open function.

// Client registers an event handler with driver. This is done once

DRV_USART_BufferEventHandlerSet( myUSARTHandle, APP_USARTBufferEventHandle,
                                 (uintptr_t)&myAppObj );

bufferHandle = DRV_USART_BufferAddRead( myUSARThandle,
                                        myBuffer, MY_BUFFER_SIZE );

if(DRV_USART_BUFFER_HANDLE_INVALID == bufferHandle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the buffer is processed.

void APP_USARTBufferEventHandler( DRV_USART_BUFFER_EVENT event,
        DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t contextHandle )
{
    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_SUCCESS:

            // This means the data was transferred.
            break;

        case DRV_USART_BUFFER_EVENT_FAILURE:

            // Error handling here.

            break;

        default:
            break;
    }
}

// Timeout function, where remove queued buffer if it still exists.
void APP_TimeOut(void)
{
    DRV_USART_BUFFER_RESULT bufferResult;
    bufferResult = DRV_USART_BufferRemove(bufferHandle);

    if(DRV_USART_BUFFER_RESULT_REMOVED_SUCCESFULLY == bufferResult)
    {
        //Buffer removed succesfully from the queue
    }
    else
    {
        //Either buffer is invalid or expired.
        //Or not able to acquire mutex in RTOS mode.
    }
}
```

Parameters

Parameters	Description
bufferhandle	Handle of the buffer to delete.

Function

DRV_USART_BUFFER_RESULT DRV_USART_BufferRemove(DRV_USART_BUFFER_HANDLE bufferHandle)

e) File I/O Type Read/Write Functions

DRV_USART_Read Function

Reads data from the USART.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
size_t DRV_USART_Read(const DRV_HANDLE handle, void * buffer, const size_t numbytes);
```

Returns

Number of bytes actually copied into the caller's buffer. Returns DRV_USART_READ_ERROR in case of an error.

Description

This routine reads data from the USART. This function is blocking if the driver was opened by the client for blocking operation. This function will not block if the driver was opened by the client for non blocking operation. If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_BLOCKING, this function will only return when (or will block until) numbytes of bytes have been received or if an error occurred. If there are buffers queued for receiving data, these buffers will be serviced first. The function will not return until the requested number of bytes have been read.

If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_NON_BLOCKING, this function will return with the number of bytes that were actually read. The function will not wait until numBytes of bytes have been read. If there are buffer queued for reading data, then the function will not block and will return immediately with 0 bytes read.

Remarks

This function is thread safe in a RTOS application. If the driver is configured for polled operation, this it will not support blocking operation in a bare metal (non-RTOS) application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_READ or DRV_IO_INTENT_READWRITE must have been specified in the DRV_USART_Open call.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    count;
unsigned int    total;

total = 0;
do
{
    count  = DRV_USART_Read(myUSARTHandle, &myBuffer[total], MY_BUFFER_SIZE - total);
    if(count == DRV_USART_READ_ERROR)
    {
       // There was an error. The DRV_USART_ErrorGet() function
       // can be called to find the exact error.
    }
    total += count;

    // Do something else...

} while( total < MY_BUFFER_SIZE );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
buffer	Buffer into which the data read from the USART instance will be placed.

numbytes	Total number of bytes that need to be read from the module instance (must be equal to or less than the size of the buffer)

Function

size_t DRV_USART_Read
(
const DRV_HANDLE handle,
void * buffer,
const size_t numbytes
)

DRV_USART_Write Function

Writes data to the USART.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
size_t DRV_USART_Write(const DRV_HANDLE handle, void * buffer, const size_t numbytes);
```

Returns

Number of bytes actually written to the driver. Return DRV_USART_WRITE_ERROR in case of an error.

Description

This routine writes data to the USART. This function is blocking if the driver was opened by the client for blocking operation. This function will not block if the driver was opened by the client for non blocking operation. If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_BLOCKING, this function will only return when (or will block until) numbytes of bytes have been transmitted or if an error occurred. If there are buffers queued for writing, the function will wait until all the preceding buffers are completed. Ongoing buffer transmit operations will not be affected.

If the ioIntent parameter at the time of opening the driver was DRV_IO_INTENT_NON_BLOCKING, this function will return with the number of bytes that were actually accepted for transmission. The function will not wait until numBytes of bytes have been transmitted. If there a buffers queued for transmit, the function will not wait and will return immediately with 0 bytes.

Remarks

This function is thread safe in a RTOS application. This function is thread safe in a RTOS application. If the driver is configured for polled operation, this it will not support blocking operation in a bare metal (non-RTOS) application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

DRV_IO_INTENT_WRITE or DRV_IO_INTENT_READWRITE must have been specified in the DRV_USART_Open call.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
char            myBuffer[MY_BUFFER_SIZE];
int    count;
unsigned int    total;

total = 0;
do
{
    count  = DRV_USART_Write(myUSARTHandle, &myBuffer[total],
                                            MY_BUFFER_SIZE - total);
    total += count;

    // Do something else...

} while( total < MY_BUFFER_SIZE );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
buffer	Buffer containing the data to written.
numbytes	size of the buffer

Function

size_t DRV_USART_Write
(
const DRV_HANDLE handle,
void * buffer,
const size_t numbytes
)

f) Byte Transfer Functions

DRV_USART_ReadByte Function

Reads a byte of data from the USART.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
uint8_t DRV_USART_ReadByte(const DRV_HANDLE handle);
```

Returns

A data byte received by the driver.

Description

This routine reads a byte of data from the USART.

Remarks

This function is thread safe when called in a RTOS application. Note that DRV_USART_WriteByte and DRV_USART_ReadByte function cannot co-exist with DRV_USART_BufferAddRead, DRV_USART_BufferAddWrite, DRV_USART_Read and DRV_USART_Write functions in a application. Calling the DRV_USART_ReadByte and DRV_USART_WriteByte functions will disrupt the processing of any queued buffers.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

The transfer status should be checked to see if the receiver is not empty before calling this function.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    numBytes;

numBytes = 0;
do
{
    if( DRV_USART_TRANSFER_STATUS_RECEIVER_DATA_PRESENT & DRV_USART_TransferStatus(myUSARTHandle) )
    {
        myBuffer[numBytes++] = DRV_USART_ReadByte(myUSARTHandle);
    }

    // Do something else...

} while( numBytes < MY_BUFFER_SIZE);
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

uint8_t DRV_USART_ReadByte(const DRV_HANDLE handle)

DRV_USART_WriteByte Function

Writes a byte of data to the USART.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
void DRV_USART_WriteByte(const DRV_HANDLE handle, const uint8_t byte);
```

Returns

None.

Description

This routine writes a byte of data to the USART.

Remarks

This function is thread safe when called in a RTOS application. Note that DRV_USART_WriteByte and DRV_USART_ReadByte function cannot co-exist with DRV_USART_BufferAddRead, DRV_USART_BufferAddWrite, DRV_USART_Read and DRV_USART_Write functions in a application. Calling the DRV_USART_ReadByte and DRV_USART_WriteByte function will disrupt the processing of any queued buffers.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

The transfer status should be checked to see if transmitter is not full before calling this function.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    numBytes;

// Preinitialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE );
{
    if( !(DRV_USART_TRANSFER_STATUS_TRANSMIT_FULL & DRV_USART_TransferStatus(myUSARTHandle)) )
    {
        DRV_USART_WriteByte(myUSARTHandle, myBuffer[numBytes++]);
    }

    // Do something else...
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine
byte	Data byte to write to the USART

Function

void DRV_USART_WriteByte(const DRV_HANDLE handle, const uint8_t byte)

DRV_USART_TransmitBufferSizeGet Function

Returns the size of the transmit buffer.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
unsigned int DRV_USART_TransmitBufferSizeGet(const DRV_HANDLE handle);
```

Returns

Size of the driver's transmit buffer, in bytes.

Description

This routine returns the size of the transmit buffer and can be used by the application to determine the number of bytes to write with the DRV_USART_WriteByte function.

Remarks

Does not account for client queued buffers. This function is thread safe when used in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
const uint8_t   writeBuffer[5];
unsigned int    size, numBytes = 0;
unsigned int    writeBufferLen = sizeof(writeBuffer);

size     = DRV_USART_TransmitBufferSizeGet (myUSARTHandle);

// Do something based on the transmitter buffer size
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

unsigned int DRV_USART_TransmitBufferSizeGet (const DRV_HANDLE handle)

DRV_USART_ReceiverBufferSizeGet Function

Returns the size of the receive buffer.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
unsigned int DRV_USART_ReceiverBufferSizeGet(const DRV_HANDLE handle);
```

Returns

Size of the driver's receive buffer, in bytes.

Description

This routine returns the size of the receive buffer.

Remarks

Does not account for client queued buffers. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
const uint8_t   readBuffer[5];
unsigned int    size, numBytes = 0;
unsigned int    readbufferLen = sizeof(readBuffer);

size     = DRV_USART_ReceiverBufferSizeGet(myUSARTHandle);

// Do something based on the receiver buffer size
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

unsigned int DRV_USART_ReceiverBufferSizeGet(const DRV_HANDLE handle)

DRV_USART_TransferStatus Function

Returns the transmitter and receiver transfer status.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
DRV_USART_TRANSFER_STATUS DRV_USART_TransferStatus(const DRV_HANDLE handle);
```

Returns

A DRV_USART_TRANSFER_STATUS value describing the current status of the transfer.

Description

This returns the transmitter and receiver transfer status.

Remarks

The returned status may contain a value with more than one of the bits specified in the DRV_USART_TRANSFER_STATUS enumeration set. The caller should perform an "AND" with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open

if (DRV_USART_TRANSFER_STATUS_RECEIVER_DATA_PRESENT & DRV_USART_TransferStatus(myUSARTHandle))
{
    // Data has been received that can be read
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

DRV_USART_TRANSFER_STATUS DRV_USART_TransferStatus(const DRV_HANDLE handle)

DRV_USART_TransmitBufferIsFull Function

Provides the status of the driver's transmit buffer.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
bool DRV_USART_TransmitBufferIsFull(const DRV_HANDLE handle);
```

Returns

true - if the transmit buffer is full

false - if the transmit buffer is not full

Description

This routine identifies if the driver's transmit buffer is full or not. This function can be used in conjunction with the DRV_USART_Write and DRV_USART_WriteByte functions.

Remarks

Does not account for client queued buffers. This function is thread safe when called in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE      myUSARTHandle;    // Returned from DRV_USART_Open
unsigned int    numBytes;
int             bytesToWrite;
const uint8_t   writeBuffer[35] = "1234567890ABCDEFGHIJKLMNOPn" ;
int             writebufferLen = strlen((char *)writeBuffer);

numBytes = 0;
while( numBytes < writebufferLen )
{
    if (DRV_USART_TransmitBufferisFull())
    {
        // Do something else until there is some room in the driver's Transmit buffer.
    }
    else
    {
        DRV_USART_WriteByte(myUSARTHandle, writeBuffer[numBytes++]);
    }
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

bool DRV_USART_TransmitBufferIsFull(const DRV_HANDLE handle)

DRV_USART_ReceiverBufferIsEmpty Function

Provides the status of the driver's receive buffer.

Implementation: Static/Dynamic

File

drv_usart.h

C

```
bool DRV_USART_ReceiverBufferIsEmpty(const DRV_HANDLE handle);
```

Returns

true - if the driver's receive buffer is empty

false - if the driver's receive buffer is not empty

Description

This routine indicates if the driver's receiver buffer is empty. This function can be used in conjunction with the DRV_USART_Read and DRV_USART_ReadByte functions.

Remarks

Does not account for client queued buffers. This function is safe thread safe when used in a RTOS application.

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

DRV_USART_Open must have been called to obtain a valid opened device handle.

Example

```
DRV_HANDLE               myUSARTHandle;    // Returned from DRV_USART_Open
char                     myBuffer[MY_BUFFER_SIZE];
unsigned int             numBytes;

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE );
{
    if ( !DRV_USART_ReceiverBufferIsEmpty(myUSARTHandle) )
    {
        if( numBytes < MY_BUFFER_SIZE )
        {
            myBuffer[numBytes++] = DRV_USART_ReadByte (myUSARTHandle);
        }
        else
        {
            break;
        }
    }

    // Do something else while more data is received.
}
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from the driver's open routine

Function

bool DRV_USART_ReceiverBufferIsEmpty(const DRV_HANDLE handle)

DRV_USART_ByteErrorCallbackSet Function

Registers callback to handle for byte error events.

File

drv_usart.h

C

```
void DRV_USART_ByteErrorCallbackSet(const SYS_MODULE_INDEX index, const DRV_USART_BYTE_EVENT_HANDLER
eventHandler);
```

Returns

None.

Description

This function allows a callback function to be registered with the driver to handle the error events occurring in the transmit/receive path during byte transfers.

The callback function should be registered as part of the initialization. The callback functionality is available only in the interrupt mode of operation. The driver clears the interrupt after invoking the callback function.

Remarks

None

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];

// myUSARTHandle is the handle returned by the DRV_USART_Open function.
myUSARTHandle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
                                (uintptr_t)&myAppObj );

// Register an event handler with driver. This is done once
DRV_USART_ByteErrorCallbackSet (DRV_USART_INDEX_0, APP_USARTErrorEventHandler);

// Event Processing Technique.
void APP_USARTErrorEventHandler(const SYS_MODULE_INDEX index)
{
    // Error has occurred. Handle the event.
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
eventHandler	Pointer to the event handler function.

Function

void DRV_USART_ByteErrorCallbackSet

(

const SYS_MODULE_INDEX index,

const DRV_USART_BYTE_EVENT_HANDLER eventHandler

)

DRV_USART_ByteReceiveCallbackSet Function

Registers receive callback function for byte receive event.

File

drv_usart.h

C

```
void DRV_USART_ByteReceiveCallbackSet(const SYS_MODULE_INDEX index, const DRV_USART_BYTE_EVENT_HANDLER
eventHandler);
```

Returns

None.

Description

This function allows a receive callback function to be registered with the driver. The callback function is invoked when a byte has been received. The received byte can then be read using DRV_USART_ReadByte() function.

The callback function should be registered with the driver as part of the initialization. The callback functionality is available only in the interrupt mode of operation. The driver clears the interrupt after invoking the callback function.

Remarks

None

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];

// myUSARTHandle is the handle returned by the DRV_USART_Open function.
myUSARTHandle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
                                (uintptr_t)&myAppObj );

// Register an event handler with driver. This is done once
DRV_USART_ByteReceiveCallbackSet(DRV_USART_INDEX_0, APP_USARTReceiveEventHandler);

// Event Processing Technique. Event is received when
// a byte is received.

void APP_USARTReceiveEventHandler(const SYS_MODULE_INDEX index)
{
    // Byte has been Received. Handle the event.
    // Read byte using DRV_USART_ReadByte ()
    // DRV_USART_ReceiverBufferIsEmpty() function can be used to
    // check if the receiver buffer is empty.
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
eventHandler	Pointer to the event handler function.

Function

void DRV_USART_ByteReceiveCallbackSet

(

const SYS_MODULE_INDEX index,

const DRV_USART_BYTE_EVENT_HANDLER eventHandler

)

DRV_USART_ByteTransmitCallbackSet Function

Registers a callback function for byte transmit event.

File

drv_usart.h

C

```
void DRV_USART_ByteTransmitCallbackSet(const SYS_MODULE_INDEX index, const DRV_USART_BYTE_EVENT_HANDLER
eventHandler);
```

Returns

None.

Description

This function allows a transmit callback function to be registered with the driver. The callback function is invoked when a byte has been transmitted using DRV_USART_WriteByte () function.

The callback function should be registered with the driver prior to any writes to the driver. The callback functionality is available only in the interrupt mode of operation. The driver clears the interrupt after invoking the callback function.

Remarks

None

Preconditions

The DRV_USART_Initialize routine must have been called for the specified USART driver instance.

Example

```
// myAppObj is an application specific state data object.
MY_APP_OBJ myAppObj;

uint8_t mybuffer[MY_BUFFER_SIZE];

// myUSARTHandle is the handle returned by the DRV_USART_Open function.
myUSARTHandle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);
                                (uintptr_t)&myAppObj );

// Register an event handler with driver. This is done once
DRV_USART_ByteTransmitCallbackSet (DRV_USART_INDEX_0, APP_USARTTransmitEventHandler);

DRV_USART_WriteByte (myUSARThandle, myBuffer[0]);

// Event Processing Technique. Event is received when
// the byte is transmitted.

void APP_USARTTransmitEventHandler (const SYS_MODULE_INDEX index)
{
    // Byte has been transmitted. Handle the event.
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to be opened
eventHandler	Pointer to the event handler function.

Function

void DRV_USART_ByteTransmitCallbackSet

(

const SYS_MODULE_INDEX index,

const DRV_USART_BYTE_EVENT_HANDLER eventHandler

)

g) Data Types and Constants

DRV_USART_CLIENT_STATUS Enumeration

Defines the client-specific status of the USART driver.

File

drv_usart.h

C

```
typedef enum {
  DRV_USART_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR,
  DRV_USART_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED,
  DRV_USART_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY,
  DRV_USART_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY
} DRV_USART_CLIENT_STATUS;
```

Members

Members	Description
DRV_USART_CLIENT_STATUS_ERROR = DRV_CLIENT_STATUS_ERROR	An error has occurred.
DRV_USART_CLIENT_STATUS_CLOSED = DRV_CLIENT_STATUS_CLOSED	The driver is closed, no operations for this client are ongoing, and/or the given handle is invalid.
DRV_USART_CLIENT_STATUS_BUSY = DRV_CLIENT_STATUS_BUSY	The driver is currently busy and cannot start additional operations.
DRV_USART_CLIENT_STATUS_READY = DRV_CLIENT_STATUS_READY	The module is running and ready for additional operations

Description

USART Client-Specific Driver Status

This enumeration defines the client-specific status codes of the USART driver.

Remarks

Returned by the DRV_USART_ClientStatus function.

DRV_USART_INIT Structure

Defines the data required to initialize or reinitialize the USART driver

File

drv_usart.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  USART_MODULE_ID usartID;
  DRV_USART_OPERATION_MODE mode;
  DRV_USART_OPERATION_MODE_DATA modeData;
  DRV_USART_INIT_FLAGS flags;
  uint32_t brgClock;
  DRV_USART_LINE_CONTROL lineControl;
  uint32_t baud;
  DRV_USART_HANDSHAKE handshake;
  INT_SOURCE interruptTransmit;
  INT_SOURCE interruptReceive;
  INT_SOURCE interruptError;
  unsigned int queueSizeReceive;
  unsigned int queueSizeTransmit;
  DMA_CHANNEL dmaChannelTransmit;
  DMA_CHANNEL dmaChannelReceive;
  INT_SOURCE dmaInterruptTransmit;
  INT_SOURCE dmaInterruptReceive;
} DRV_USART_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization data
USART_MODULE_ID usartID;	Identifies USART hardware module (PLIB-level) ID. For a static build of the driver, this is overridden by DRV_USART_MODULE_ID macro in the system_config.h header file.
DRV_USART_OPERATION_MODE mode;	Identifies the Operation mode of the USART driver. For a static build of the driver, this is overridden by DRV_USART_MODE_SELECT macro in the system_config.h header file.
DRV_USART_OPERATION_MODE_DATA modeData;	Data required by the operation mode of the driver. For a static build of the driver, this is overridden by DRV_USART_MODE_DATA macro in the system_config.h header file.
DRV_USART_INIT_FLAGS flags;	Flags to enable specific features. Refer to the description of DRV_USART_INIT_FLAGS for more details. For a static build of the driver, this is overridden by DRV_USART_FLAGS macro in the system_config.h header file.
uint32_t brgClock;	USART module Baud Rate Generator Clock. This typically the peripheral bus clock frequency. For a static build of the driver, this is overridden by DRV_USART_BRG_CLOCK macro in the system_config.h header file.
DRV_USART_LINE_CONTROL lineControl;	The initial USART line control settings. For a static build of the driver this is overridden by the DRV_USART_LINE_CONTROL_SET macro in the system_config.h header file.

uint32_t baud;	Baud Rate value to be used, if not using auto baud. For a static build of the driver, this is overridden by the DRV_USART_BAUD macro in the system_config.h header file.
DRV_USART_HANDSHAKE handshake;	Handshaking mode. For a static build of the driver, this is overridden by the DRV_USART_HANDSHAKE_SET macro in the system_config.h header file.
INT_SOURCE interruptTransmit;	Interrupt source ID for the transmitter interrupt. For a static build of the driver, this is overridden by the DRV_USART_TRANSMIT_INTERRUPT_SOURCE macro in the system_config.h header.
INT_SOURCE interruptReceive;	Interrupt source ID for the receiver interrupt. For a static build of the driver, this is overridden by the DRV_USART_RECEIVE_INTERRUPT_SOURCE macro in the system_config.h header.
INT_SOURCE interruptError;	Interrupt source ID for the error Interrupt. For a static build of the driver, this is overridden by the DRV_USART_ERROR_INTERRUPT_SOURCE macro in the system_config.h header.
unsigned int queueSizeReceive;	This is the receive buffer queue size. This is the maximum number of read requests that driver will queue. For a static build of the driver, this is overridden by the DRV_USART_RECEIVE_QUEUE_SIZE macro in system_config.h
unsigned int queueSizeTransmit;	This is the transmit buffer queue size. This is the maximum number of write requests that driver will queue. For a static build of the driver, this is overridden by the DRV_USART_TRANSMIT_QUEUE_SIZE macro in system_config.h
DMA_CHANNEL dmaChannelTransmit;	This is the USART transmit DMA channel. This is applicable only if DRV_USART_SUPPORT_TRANSMIT_DMA is defined as true. For a static build of the driver, this is overridden by the DRV_USART_TRANSMIT_DMA_CHANNEL macro in system_config.h
DMA_CHANNEL dmaChannelReceive;	This is the USART receive DMA channel. This is applicable only if DRV_USART_SUPPORT_RECEIVE_DMA is defined as true. For a static build of the driver, this is overridden by the DRV_USART_RECEIVE_DMA_CHANNEL macro in system_config.h
INT_SOURCE dmaInterruptTransmit;	This is the USART transmit DMA channel interrupt. This is applicable only if DRV_USART_SUPPORT_TRANSMIT_DMA is defined as true. For a static build of the driver, this is overridden by the DRV_USART_INTERRUPT_SOURCE_TRANSMIT_DMA_CHANNEL macro in system_config.h
INT_SOURCE dmaInterruptReceive;	This is the USART receive DMA channel interrupt. This is applicable only if DRV_USART_SUPPORT_RECEIVE_DMA is defined as true. For a static build of the driver, this is overridden by the DRV_USART_INTERRUPT_SOURCE_RECEIVE_DMA_CHANNEL macro in system_config.h

Description

USART Driver Initialization Data

This data type defines the data required to initialize or reinitialize the USART driver. If the driver is built statically, the members of this data structure are statically over-ridden by static override definitions in the system_config.h file.

Remarks

None.

DRV_USART_INIT_FLAGS Enumeration

Flags identifying features that can be enabled when the driver is initialized.

File

drv_usart.h

C

```
typedef enum {
  DRV_USART_INIT_FLAG_AUTO_BAUD,
  DRV_USART_INIT_FLAG_STOP_IN_IDLE
} DRV_USART_INIT_FLAGS;
```

Members

Members	Description
DRV_USART_INIT_FLAG_AUTO_BAUD	Flag to enable auto baud detection. If supported and enabled, this feature will allow the USART to automatically detect the baud rate in use.
DRV_USART_INIT_FLAG_STOP_IN_IDLE	Flag to enable stop in idle. If supported and enabled , this feature will allow the USART to stop when the CPU enters Idle mode

Description

USART Initialization flags

This enumeration defines flags identifying features that can be enabled when the driver is initialized.

Remarks

These flags can be logically ORed together. They are passed into the DRV_USART_Initialize function through the "flags" member of the DRV_USART_INIT structure.

DRV_USART_TRANSFER_STATUS Enumeration

Specifies the status of the receive or transmit

File

drv_usart.h

C

```
typedef enum {
  DRV_USART_TRANSFER_STATUS_RECEIVER_DATA_PRESENT,
  DRV_USART_TRANSFER_STATUS_RECEIVER_EMPTY,
  DRV_USART_TRANSFER_STATUS_TRANSMIT_FULL,
  DRV_USART_TRANSFER_STATUS_TRANSMIT_EMPTY
} DRV_USART_TRANSFER_STATUS;
```

Members

Members	Description
DRV_USART_TRANSFER_STATUS_RECEIVER_DATA_PRESENT	Indicates that at least one byte of Data has been received
DRV_USART_TRANSFER_STATUS_RECEIVER_EMPTY	Indicates that the core driver receiver buffer is empty
DRV_USART_TRANSFER_STATUS_TRANSMIT_FULL	Indicates that the core driver transmitter buffer is full
DRV_USART_TRANSFER_STATUS_TRANSMIT_EMPTY	Indicates that the core driver transmitter buffer is empty

Description

USART Driver Transfer Flags

This type specifies the status of the receive or transmit operation.

Remarks

More than one of these values may be OR'd together to create a complete status value. To test a value of this type, the bit of interest must be ANDed with the value and checked to see if the result is non-zero.

DRV_USART_INDEX_0 Macro

USART driver index definitions

File

drv_usart.h

C

```
#define DRV_USART_INDEX_0 0
```

Description

Driver USART Module Index

These constants provide USART driver index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the DRV_USART_Initialize and DRV_USART_Open routines to identify the driver instance in use.

DRV_USART_INDEX_1 Macro

File

drv_usart.h

C

```
#define DRV_USART_INDEX_1 1
```

Description

This is macro DRV_USART_INDEX_1.

DRV_USART_INDEX_2 Macro

File

drv_usart.h

C

```
#define DRV_USART_INDEX_2 2
```

Description

This is macro DRV_USART_INDEX_2.

DRV_USART_INDEX_3 Macro

File

drv_usart.h

C

```
#define DRV_USART_INDEX_3 3
```

Description

This is macro DRV_USART_INDEX_3.

DRV_USART_INDEX_4 Macro

File

drv_usart.h

C

```
#define DRV_USART_INDEX_4 4
```

Description

This is macro DRV_USART_INDEX_4.

DRV_USART_INDEX_5 Macro

File

drv_usart.h

C

```
#define DRV_USART_INDEX_5 5
```

Description

This is macro DRV_USART_INDEX_5.

DRV_USART_BAUD_SET_RESULT Enumeration

Identifies the handshaking modes supported by the USART driver.

File

drv_usart.h

C

```
typedef enum {
} DRV_USART_BAUD_SET_RESULT;
```

Description

USART Handshake Modes

This data type identifies the handshaking modes supported by the USART driver.

Remarks

Not all modes are available on all devices. Refer to the specific device data sheet to determine availability.

DRV_USART_BUFFER_EVENT Enumeration

Identifies the possible events that can result from a buffer add request.

File

drv_usart.h

C

```
typedef enum {
  DRV_USART_BUFFER_EVENT_COMPLETE,
  DRV_USART_BUFFER_EVENT_ERROR,
  DRV_USART_BUFFER_EVENT_ABORT
} DRV_USART_BUFFER_EVENT;
```

Members

Members	Description
DRV_USART_BUFFER_EVENT_COMPLETE	All data from or to the buffer was transferred successfully.
DRV_USART_BUFFER_EVENT_ERROR	There was an error while processing the buffer transfer request.
DRV_USART_BUFFER_EVENT_ABORT	Data transfer aborted (Applicable in DMA mode)

Description

USART Driver Buffer Events

This enumeration identifies the possible events that can result from a buffer add request caused by the client calling either the DRV_USART_BufferAddRead or DRV_USART_BufferAddWrite functions.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that the client registered with the driver by calling the DRV_USART_BufferEventHandlerSet function when a buffer transfer request is completed.

DRV_USART_BUFFER_EVENT_HANDLER Type

Pointer to a USART Driver Buffer Event handler function

File

drv_usart.h

C

```
typedef void (* DRV_USART_BUFFER_EVENT_HANDLER)(DRV_USART_BUFFER_EVENT event, DRV_USART_BUFFER_HANDLE
bufferHandle, uintptr_t context);
```

Returns

None.

Description

USART Driver Buffer Event Handler Function Pointer

This data type defines the required function signature for the USART driver buffer event handling callback function. A client must register a pointer to a buffer event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive buffer related event calls back from the driver.

The parameters and return values and are described here and a partial example implementation is provided.

Remarks

If the event is DRV_USART_BUFFER_EVENT_COMPLETE, it means that the data was transferred successfully.

If the event is DRV_USART_BUFFER_EVENT_ERROR, it means that the data was not transferred successfully. The DRV_USART_ErrorGet function can be called to know the error. The DRV_USART_BufferCompletedBytesGet function can be called to find out how many bytes were

processed.

The bufferHandle parameter contains the buffer handle of the buffer that associated with the event.

The context parameter contains the a handle to the client context, provided at the time the event handling function was registered using the DRV_USART_BufferEventHandlerSet function. This context handle value is passed back to the client as the "context" parameter. It can be any value necessary to identify the client context or instance (such as a pointer to the client's data) instance of the client that made the buffer add request.

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended of the application to not perform process intensive or blocking operations with in this function.

The DRV_USART_BufferAddRead and DRV_USART_BufferAddWrite functions can be called in the event handler to add a buffer to the driver queue. These functions can only be called to add buffers to the driver whose event handler is running. For example, buffers cannot be added USART2 driver in USART1 driver event handler.

Example

```
void APP_MyBufferEventHandler( DRV_USART_BUFFER_EVENT event,
                               DRV_USART_BUFFER_HANDLE bufferHandle,
                               uintptr_t context )
{
    MY_APP_DATA_STRUCT pAppData = (MY_APP_DATA_STRUCT) context;

    switch(event)
    {
        case DRV_USART_BUFFER_EVENT_COMPLETE:

            // Handle the completed buffer.
            break;

        case DRV_USART_BUFFER_EVENT_ERROR:
        default:

            // Handle error.
            break;
    }
}
```

Parameters

Parameters	Description
event	Identifies the type of event
bufferHandle	Handle identifying the buffer to which the vent relates
context	Value identifying the context of the application that registered the event handling function.

DRV_USART_BUFFER_HANDLE Type

Handle identifying a read or write buffer passed to the driver.

File

drv_usart.h

C

```
typedef uintptr_t DRV_USART_BUFFER_HANDLE;
```

Description

USART Driver Buffer Handle

A buffer handle value is returned by a call to the DRV_USART_BufferAddRead or DRV_USART_BufferAddWrite functions. This handle is associated with the buffer passed into the function and it allows the application to track the completion of the data from (or into) that buffer. The buffer handle value returned from the "buffer add" function is returned back to the client by the "event handler callback" function registered with the driver.

The buffer handle assigned to a client request expires when the client has been notified of the completion of the buffer transfer (after event handler function that notifies the client returns) or after the buffer has been retired by the driver if no event handler callback was set.

Remarks

None

DRV_USART_ERROR Enumeration

Defines the possible errors that can occur during driver operation.

File

drv_usart.h

C

```
typedef enum {
  DRV_USART_ERROR_ADDRESS
} DRV_USART_ERROR;
```

Members

Members	Description
DRV_USART_ERROR_ADDRESS	Channel address error (Applicable in DMA mode)

Description

USART Driver Errors.

This data type defines the possible errors that can occur when occur during USART driver operation. These values are returned by DRV_USART_ErrorGet function.

Remarks

None

DRV_USART_LINE_CONTROL_SET_RESULT Enumeration

Identifies the results of the baud set function.

File

drv_usart.h

C

```
typedef enum {
} DRV_USART_LINE_CONTROL_SET_RESULT;
```

Description

USART Line Control Set Result

This data type identifies the results of the DRV_USART_LineControlSet function.

Remarks

None.

DRV_USART_OPERATION_MODE_DATA Union

Defines the initialization data required for different operation modes of USART.

File

drv_usart.h

C

```
typedef union {
  struct {
    uint8_t address;
  } AddressedModeInit;
} DRV_USART_OPERATION_MODE_DATA;
```

Members

Members	Description
struct { uint8_t address; } AddressedModeInit;	Initialization for Addressed mode

uint8_t address;	Address of the device.

Description

Operation Mode Initialization Data

This data type defines the initialization data required for different operation modes of the USART.

Remarks

None

DRV_USART_BUFFER_HANDLE_INVALID Macro

Definition of an invalid buffer handle.

File

drv_usart.h

C

```
#define DRV_USART_BUFFER_HANDLE_INVALID
```

Description

USART Driver Invalid Buffer Handle

This is the definition of an invalid buffer handle. An invalid buffer handle is returned by DRV_USART_BufferAddRead and DRV_USART_BufferAddWrite functions if the buffer add request was not successful.

Remarks

None

DRV_USART_COUNT Macro

Number of valid USART drivers

File

drv_usart.h

C

```
#define DRV_USART_COUNT
```

Description

USART Driver Module Count

This constant identifies the maximum number of USART Driver instances that should be defined in the system. Defining more instances than this constant will waste RAM memory space.

This constant can also be used by the system and application to identify the number of USART instances on this microcontroller.

Remarks

This value is part-specific.

DRV_USART_READ_ERROR Macro

USART Driver Read Error.

File

drv_usart.h

C

```
#define DRV_USART_READ_ERROR
```

Description

USART Driver Read Error

This constant is returned by DRV_USART_Read() function when an error occurs.

Remarks

None.

DRV_USART_WRITE_ERROR Macro

USART Driver Write Error.

File

drv_usart.h

C

```
#define DRV_USART_WRITE_ERROR
```

Description

USART Driver Write Error

This constant is returned by DRV_USART_Write() function when an error occurs.

Remarks

None.

DRV_USART_LINE_CONTROL Enumeration

Identifies the line control modes supported by the USART driver.

File

drv_usart.h

C

```
typedef enum {
} DRV_USART_LINE_CONTROL;
```

Description

USART Line Control Modes

This data type identifies the line control modes supported by the USART driver. Line control modes define the number of data bits, parity mode, and the number of stop bits in a USART transmit and receive frames.

Remarks

The abbreviations used in the labels for the values of this enumeration follow the format , where:is the number of data bitsis either "NONE" (for no parity), "EVEN" for 1 parity bit added to obtain an even number of bits, or "ODD" for one bit added to obtain an odd number of bits.is the number of Stop bits

DRV_USART_OPERATION_MODE Enumeration

Identifies the modes of the operation of the USART module

File

drv_usart.h

C

```
typedef enum {
  DRV_USART_OPERATION_MODE_IRDA,
  DRV_USART_OPERATION_MODE_NORMAL,
  DRV_USART_OPERATION_MODE_ADDRESSED,
  DRV_USART_OPERATION_MODE_LOOPBACK
} DRV_USART_OPERATION_MODE;
```

Members

Members	Description
DRV_USART_OPERATION_MODE_IRDA	USART works in IRDA mode

DRV_USART_OPERATION_MODE_NORMAL	This is the normal point to point communication mode where the USART communicates directly with another USART by connecting it's Transmit signal to the external USART's Receiver signal and vice versa. An external transceiver may be connected to obtain RS-232 signal levels. This type of connection is typically full duplex.
DRV_USART_OPERATION_MODE_ADDRESSED	This is a multi-point bus mode where the USART can communicate with many other USARTS on a bus using an address-based protocol such as RS-485. This mode is typically half duplex and the physical layer may require a transceiver. In this mode every USART on the bus is assigned an address and the number of data bits is 9 bits
DRV_USART_OPERATION_MODE_LOOPBACK	Loopback mode internally connects the Transmit signal to the Receiver signal, looping data transmission back into this USART's own input. It is useful primarily as a test mode.

Description

USART Modes of Operation

This data type identifies the modes of the operation of the USART module.

Remarks

Not all modes are available on all devices. Refer to the specific device data sheet to determine availability.

DRV_USART_BYTE_EVENT_HANDLER Type

Pointer to a USART Driver Byte Event handler function

File

drv_usart.h

C

```
typedef void (* DRV_USART_BYTE_EVENT_HANDLER)(const SYS_MODULE_INDEX index);
```

Returns

None.

Description

USART Driver Byte Event Handler Function Pointer

This data type defines the required function signature for the USART driver byte event handling callback function. A client must register a pointer to a byte event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive byte related event calls back from the driver.

Remarks

The event handler function executes in the driver peripheral's interrupt context when the driver is configured for interrupt mode operation. It is recommended that the application not perform process intensive or blocking operations with in this function.

Example

```
void APP_MyUsartTxEventHandler(void)
{
    // Handle the transmit byte event
}
```

Parameters

Parameters	Description
index	Identifier for the instance

DRV_USART_BUFFER_RESULT Enumeration

Identifies the possible result of the buffer processing.

File

drv_usart.h

C

```
typedef enum {
  DRV_USART_BUFFER_RESULT_HANDLE_INVALID,
  DRV_USART_BUFFER_RESULT_HANDLE_EXPIRED,
  DRV_USART_BUFFER_RESULT_REMOVED_SUCCESFULLY,
```

```
    DRV_USART_BUFFER_RESULT_REMOVAL_FAILED
} DRV_USART_BUFFER_RESULT;
```

Members

Members	Description
DRV_USART_BUFFER_RESULT_HANDLE_INVALID	Buffer handle is not valid
DRV_USART_BUFFER_RESULT_HANDLE_EXPIRED	Buffer handle has expired.
DRV_USART_BUFFER_RESULT_REMOVED_SUCCESFULLY	Buffer is removed from the queue succesfully
DRV_USART_BUFFER_RESULT_REMOVAL_FAILED	Buffer removal failed because of unable to acquire the mutex • This is applicable in RTOS mode only

Description

USART Driver Buffer Result

This enumeration identifies the possible result of the buffer processing.

Remarks

DRV_USART_BUFFER_RESULT_HANDLE_EXPIRED is the state of the buffer which is in the free buffer pool.

Files

Files

Name	Description
drv_usart.h	USART Driver Interface Header File
drv_usart_config_template.h	USART Driver Configuration Template.

Description

This section lists the source and header files used by the USART Driver Library.

drv_usart.h

USART Driver Interface Header File

Enumerations

	Name	Description
	DRV_USART_BAUD_SET_RESULT	Identifies the handshaking modes supported by the USART driver.
	DRV_USART_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
	DRV_USART_BUFFER_RESULT	Identifies the possible result of the buffer processing.
	DRV_USART_CLIENT_STATUS	Defines the client-specific status of the USART driver.
	DRV_USART_ERROR	Defines the possible errors that can occur during driver operation.
	DRV_USART_INIT_FLAGS	Flags identifying features that can be enabled when the driver is initialized.
	DRV_USART_LINE_CONTROL	Identifies the line control modes supported by the USART driver.
	DRV_USART_LINE_CONTROL_SET_RESULT	Identifies the results of the baud set function.
	DRV_USART_OPERATION_MODE	Identifies the modes of the operation of the USART module
	DRV_USART_TRANSFER_STATUS	Specifies the status of the receive or transmit

Functions

	Name	Description
	DRV_USART_AddressedBufferAddWrite	Schedule a non-blocking addressed driver write operation. **Implementation:** Dynamic
	DRV_USART_BaudSet	This function changes the USART module baud to the specified value. **Implementation:** Static/Dynamic
	DRV_USART_BufferAddRead	Schedule a non-blocking driver read operation. **Implementation:** Static/Dynamic
	DRV_USART_BufferAddWrite	Schedule a non-blocking driver write operation. **Implementation:** Static/Dynamic
	DRV_USART_BufferCompletedBytesGet	Returns the number of bytes that have been processed for the specified buffer. **Implementation:** Static/Dynamic

	DRV_USART_BufferEventHandlerSet	Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished. **Implementation:** Static/Dynamic
	DRV_USART_BufferProcessedSizeGet	This API will be deprecated and not recommended to use. Use DRV_USART_BufferCompletedBytesGet to get the number of bytes processed for the specified buffer.
	DRV_USART_BufferRemove	Removes a requested buffer from the queue. **Implementation:** Static/Dynamic
	DRV_USART_ByteErrorCallbackSet	Registers callback to handle for byte error events.
	DRV_USART_ByteReceiveCallbackSet	Registers receive callback function for byte receive event.
	DRV_USART_ByteTransmitCallbackSet	Registers a callback function for byte transmit event.
	DRV_USART_ClientStatus	Gets the current client-specific status the USART driver. **Implementation:** Static/Dynamic
	DRV_USART_Close	Closes an opened-instance of the USART driver. **Implementation:** Static/Dynamic
	DRV_USART_Deinitialize	Deinitializes the specified instance of the USART driver module. **Implementation:** Static/Dynamic
	DRV_USART_ErrorGet	This function returns the error(if any) associated with the last client request. **Implementation:** Static/Dynamic
	DRV_USART_Initialize	Initializes the USART instance for the specified driver index. **Implementation:** Static/Dynamic
	DRV_USART_LineControlSet	This function changes the USART module line control to the specified value. **Implementation:** Static/Dynamic
	DRV_USART_Open	Opens the specified USART driver instance and returns a handle to it. **Implementation:** Static/Dynamic
	DRV_USART_Read	Reads data from the USART. **Implementation:** Static/Dynamic
	DRV_USART_ReadByte	Reads a byte of data from the USART. **Implementation:** Static/Dynamic
	DRV_USART_ReceiverBufferIsEmpty	Provides the status of the driver's receive buffer. **Implementation:** Static/Dynamic
	DRV_USART_ReceiverBufferSizeGet	Returns the size of the receive buffer. **Implementation:** Static/Dynamic
	DRV_USART_Status	Gets the current status of the USART driver module. **Implementation:** Static/Dynamic
	DRV_USART_TasksError	Maintains the driver's error state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_USART_TasksReceive	Maintains the driver's receive state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_USART_TasksTransmit	Maintains the driver's transmit state machine and implements its ISR. **Implementation:** Static/Dynamic
	DRV_USART_TransferStatus	Returns the transmitter and receiver transfer status. **Implementation:** Static/Dynamic
	DRV_USART_TransmitBufferIsFull	Provides the status of the driver's transmit buffer. **Implementation:** Static/Dynamic
	DRV_USART_TransmitBufferSizeGet	Returns the size of the transmit buffer. **Implementation:** Static/Dynamic
	DRV_USART_Write	Writes data to the USART. **Implementation:** Static/Dynamic
	DRV_USART_WriteByte	Writes a byte of data to the USART. **Implementation:** Static/Dynamic

Macros

	Name	Description
	DRV_USART_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
	DRV_USART_COUNT	Number of valid USART drivers
	DRV_USART_INDEX_0	USART driver index definitions
	DRV_USART_INDEX_1	This is macro DRV_USART_INDEX_1.

	DRV_USART_INDEX_2	This is macro DRV_USART_INDEX_2.
	DRV_USART_INDEX_3	This is macro DRV_USART_INDEX_3.
	DRV_USART_INDEX_4	This is macro DRV_USART_INDEX_4.
	DRV_USART_INDEX_5	This is macro DRV_USART_INDEX_5.
	DRV_USART_READ_ERROR	USART Driver Read Error.
	DRV_USART_WRITE_ERROR	USART Driver Write Error.

Structures

	Name	Description
	DRV_USART_INIT	Defines the data required to initialize or reinitialize the USART driver

Types

	Name	Description
	DRV_USART_BUFFER_EVENT_HANDLER	Pointer to a USART Driver Buffer Event handler function
	DRV_USART_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.
	DRV_USART_BYTE_EVENT_HANDLER	Pointer to a USART Driver Byte Event handler function

Unions

	Name	Description
	DRV_USART_OPERATION_MODE_DATA	Defines the initialization data required for different operation modes of USART.

Description

USART Driver Interface Header File

The USART device driver provides a simple interface to manage the USART or UART modules on Microchip microcontrollers. This file provides the interface definition for the USART driver.

File Name

drv_usart.h

Company

Microchip Technology Inc.

drv_usart_config_template.h

USART Driver Configuration Template.

Macros

	Name	Description
	DRV_USART_BAUD_RATE_IDXn	Specifies the USART Baud at which the USART driver is initialized.
	DRV_USART_BUFFER_QUEUE_SUPPORT	Specifies if the Buffer Queue support should be enabled.
	DRV_USART_BYTE_MODEL_BLOCKING	Enables or Disables DRV_USART_ByteWrite function blocking behavior.
	DRV_USART_BYTE_MODEL_CALLBACK	Enables or Disables Callback Feature of the Byte Transfer Model.
	DRV_USART_BYTE_MODEL_SUPPORT	Specifies if the Byte Model support should be enabled.
	DRV_USART_CLIENTS_NUMBER	Sets up the maximum number of clients that can be connected to any hardware instance.
	DRV_USART_INDEX	USART Static Index selection.
	DRV_USART_INSTANCES_NUMBER	Sets up the maximum number of hardware instances that can be supported.
	DRV_USART_INTERRUPT_MODE	Macro controls interrupt based operation of the driver.
	DRV_USART_INTERRUPT_SOURCE_ERROR	Defines the error interrupt source for the static driver.
	DRV_USART_INTERRUPT_SOURCE_RECEIVE	Defines the Receive interrupt source for the static driver.
	DRV_USART_INTERRUPT_SOURCE_RECEIVE_DMA	Defines the Receive DMA Channel interrupt source for the static driver.
	DRV_USART_INTERRUPT_SOURCE_TRANSMIT	Defines the Transmit interrupt source for the static driver.
	DRV_USART_INTERRUPT_SOURCE_TRANSMIT_DMA	Defines the Transmit DMA Channel interrupt source for the static driver.
	DRV_USART_PERIPHERAL_ID	Configures the USART PLIB Module ID.
	DRV_USART_QUEUE_DEPTH_COMBINED	Defines the number of entries of all queues in all instances of the driver.

	DRV_USART_RCV_QUEUE_SIZE_IDXn	Sets the USART Driver Receive Queue Size while using the Buffer Queue Data Transfer Model.
	DRV_USART_READ_WRITE_MODEL_SUPPORT	Specifies if Read/Write Model support should be enabled.
	DRV_USART_RECEIVE_DMA	Defines the USART Driver Receive DMA Channel for the static driver.
	DRV_USART_TRANSMIT_DMA	Defines the USART Driver Transmit DMA Channel in case of static driver.
	DRV_USART_XMIT_QUEUE_SIZE_IDXn	Sets the USART Driver Transmit Queue Size while using the Buffer Queue Data Transfer Model.

Description

USART Driver Configuration Template

These file provides the list of all the configurations that can be used with the driver. This file should not be included in the driver.

File Name

drv_usart_config_template.h

Company

Microchip Technology Inc.

Wi-Fi Driver Libraries

This section describes the Wi-Fi Driver Libraries available in MPLAB Harmony.

Description

Additional information including documentation is available from the Microchip website:

MRF24WG0MA Wi-Fi G PICtail/PICtail Plus Daughter Board: Part number - AC164149

http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=AC164149

MRF24WN0MA Wi-Fi PICtail/PICtail Plus Daughter Board: Part number - AC164153

http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=AC164153

The following table lists the library files available for the Wi-Fi Drivers.

	Target MCU Device		
Wi-Fi Library File Matrix	PIC32MX795F512L	PIC32MZ2048ECH144	PIC32MZ2048EFM144
Wi-Fi Device			
MRF24WG	NA	NA	NA
MRF24WN	wdrvext_mx.a	wdrvext_mz_ec.a	wdrvext_mz_ef.a

Configuring DMA SPI

Provides information on how to enable or disable DMA SPI for the MRF24WG Wi-Fi Driver and MRF24WN Wi-Fi Driver.

Description

The DMA SPI option is configurable through the MPLAB Harmony Configurator for FreeRTOS projects only. DMA SPI is not available for non-FreeRTOS projects.

Enabling DMA SPI

Provides information on how to enable DMA SPI for a Wi-Fi Driver.

Description

The DMA SPI option is configurable through the MPLAB Harmony Configurator only for RTOS projects, and DMA SPI is not available for non-RTOS projects.

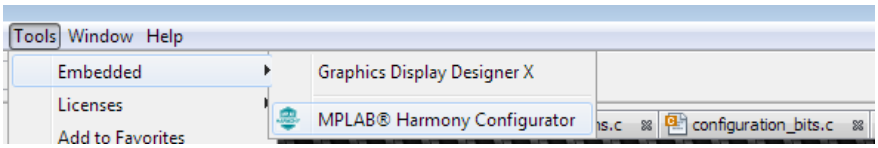
Enabling DMA

Using MHC, perform the following four step process to enable DMA SPI:

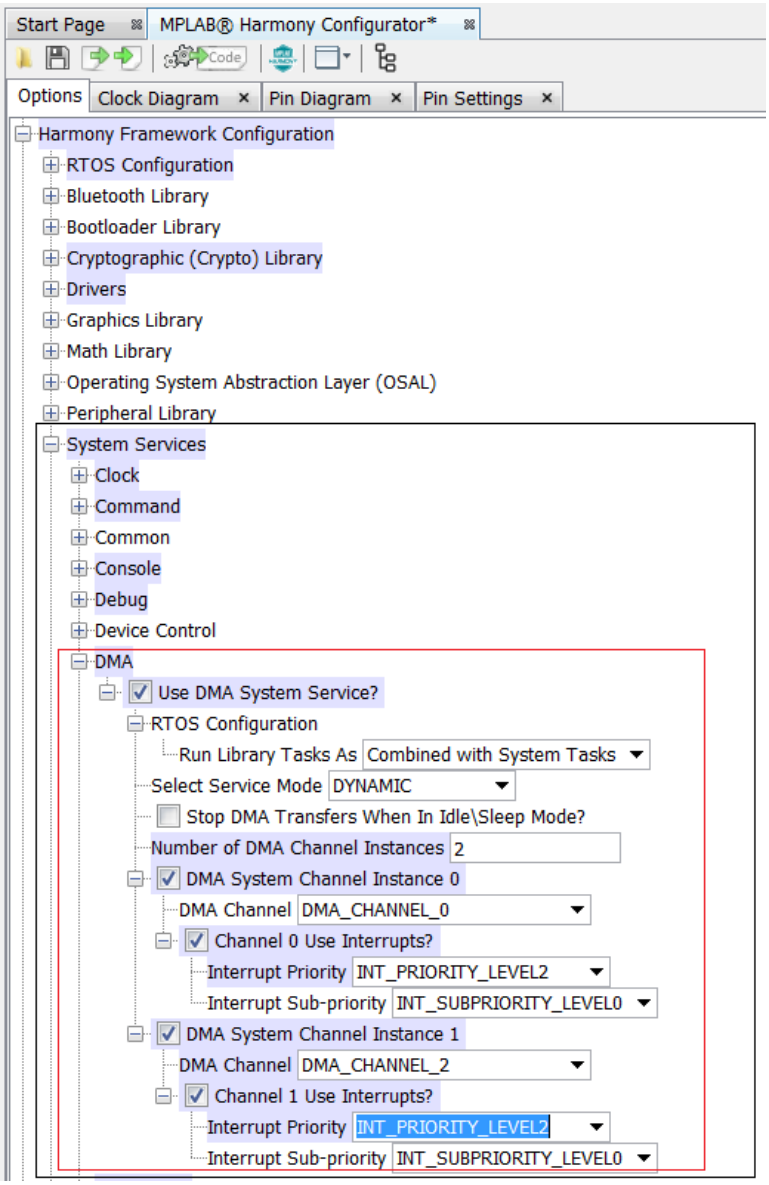
Step 1: Open the MPLAB Harmony Configurator in MPLAB X IDE

1. Open your project in MPLAB X IDE.

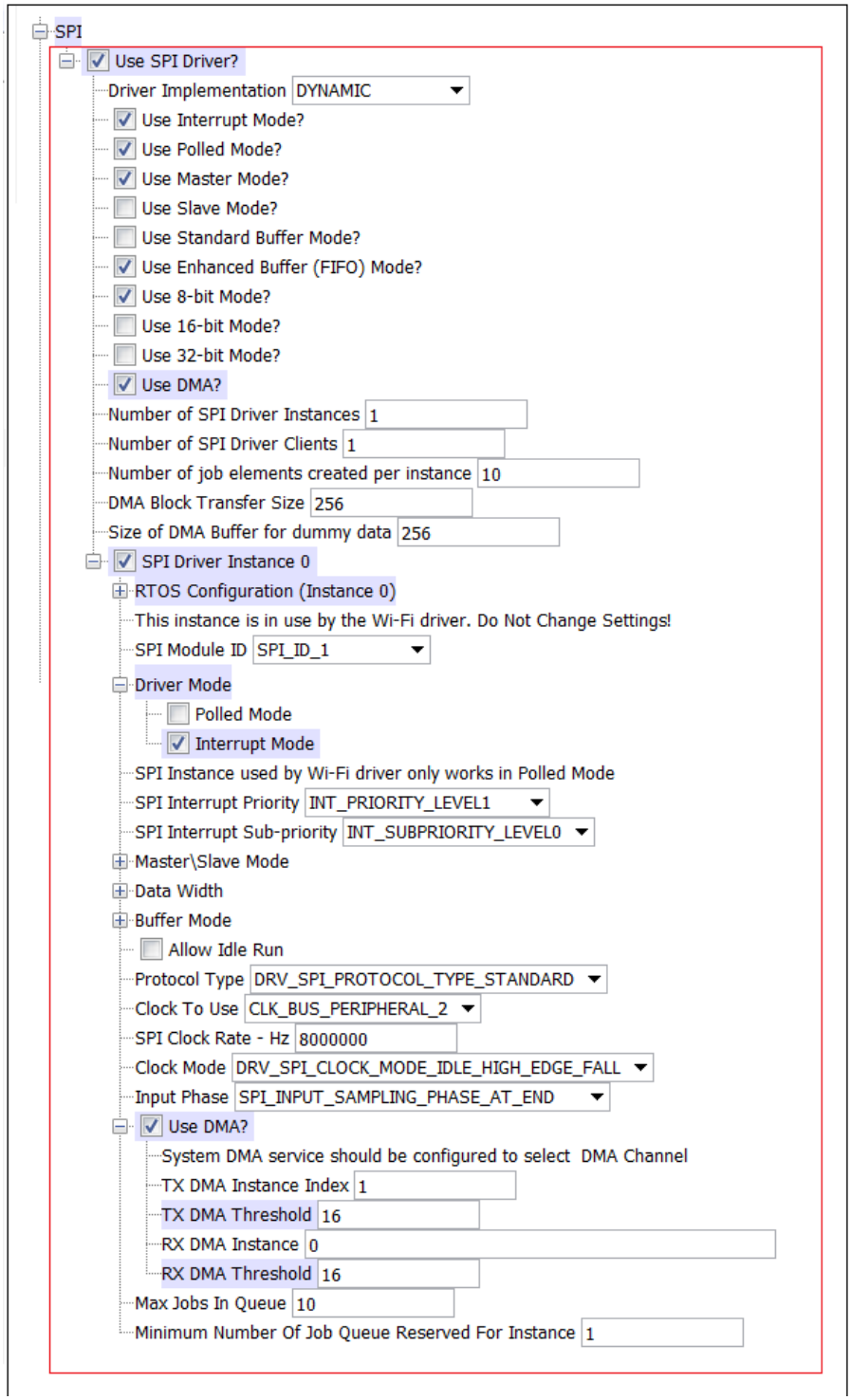
2. Start the MHC by selecting *Tools* > *Embedded* > *MPLAB Harmony Configurator*.



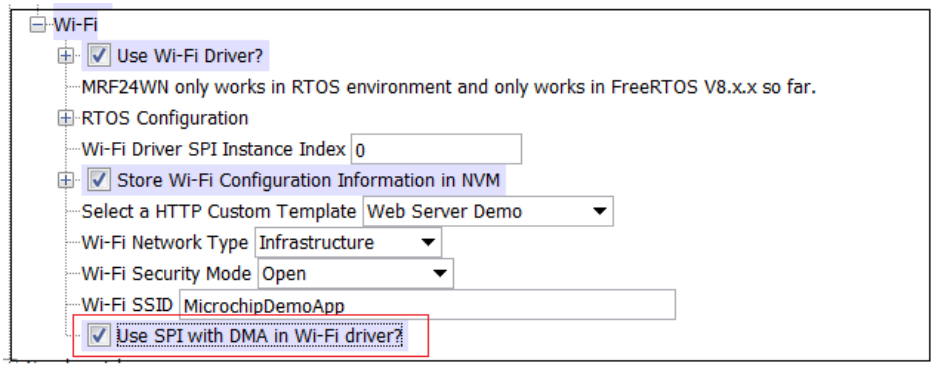
Step 2: Configure the DMA System Service



Step 3: Configure the SPI Driver



Step 4: Configure the Wi-Fi Driver



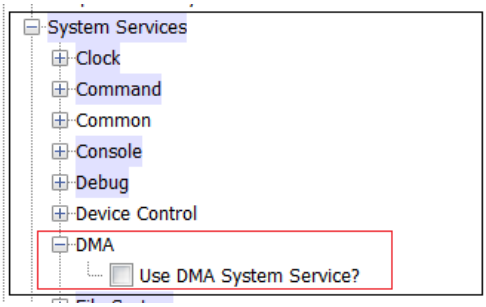
Disabling DMA SPI

Provides information on disabling DMA SPI for a Wi-Fi Driver.

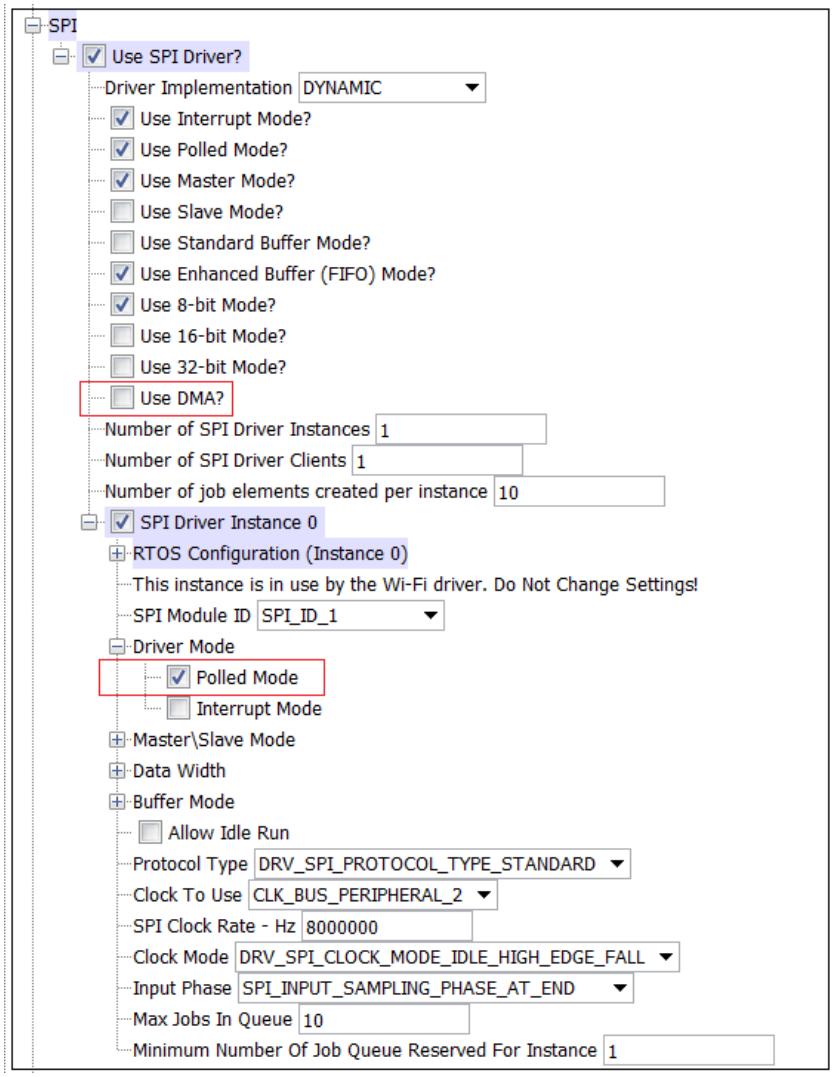
Description

Use the MHC to disable DMA SPI, as follows.

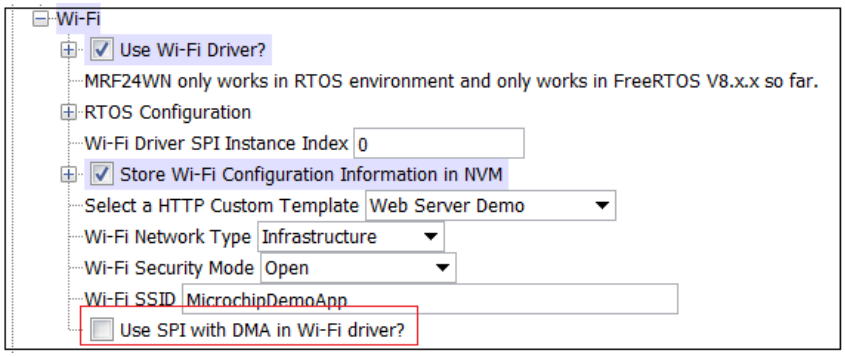
Step 1: Configure the System Services



Step2: Configure the SPI Driver



Step 3: Configure the Wi-Fi Driver



MRF24WG Wi-Fi Driver Library

This topic describes the MRF24WG Wi-Fi Driver Library.

Introduction

This library provides a low-level abstraction of the MRF24WG Wi-Fi Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The Wi-Fi software library, in conjunction with the MRF24WG0MA module, allows an application to:

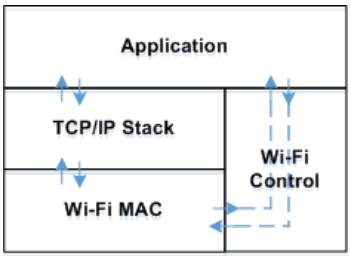
- Join an existing 802.11 Wi-Fi network
- Create a 802.11 Wi-Fi network

The following application services are provided by the Wi-Fi library:

- Configuring Wi-Fi connection (SSID, security mode, channel list, etc.)
- Join an existing network or create an "AdHoc" Wi-Fi network
- Scan for other Wi-Fi devices in the area
- Getting Wi-Fi network status
- Wi-Fi power control

The MAC layer services are not directly accessible to the application; this portion of the code resides under the TCP/IP Stack MAC module software layers and is used by stack services to transmit and receive data over a Wi-Fi network. The following diagram shows the interaction of the primary software blocks in a Wi-Fi application.

Wi-Fi Software Block Diagram



Network Connection Matrix

The following table provides information that includes network mode and security mode support by MRF24WG Wi-Fi Driver.

MRF24WG Network Connection Matrix	Network Mode		
	Infrastructure	Soft AP	Ad Hoc
Security Mode			
Open	YES	YES	YES
WEP40	YES	YES	YES
WEP104	YES	YES	YES
WPA-PSK	YES	NA	NA
WPA2-PSK	YES	NA	NA
WPA-PSK/WPA2-PSK Auto	YES	NA	NA
WPS Push Button	YES	NA	NA
WPS PIN	YES	NA	NA

Using the Library

This topic describes the basic architecture of the MRF24WG Wi-Fi Driver Library and provides information and examples on its use.

Description

Interface Header Files: `drv_wifi.h` and `drv_wifi_iwpriv.h`

The interface to the MRF24WG Wi-Fi Driver Library is defined in the `drv_wifi.h` and `drv_wifi_iwpriv.h` header files.

Please refer to the Understanding MPLAB Harmony section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the MRF24WG Wi-Fi module with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The MRF24WG Wi-Fi Library provides the following functionality:

- Wi-Fi library initialization
- Wi-Fi network configuration
- Wi-Fi network connection
- Scanning for existing Wi-Fi networks
- Wi-Fi event processing

- Wi-Fi status

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The Library Interface functions are divided into various sub-sections, which address one of the blocks or the overall operation of the MRF24WG Wi-FI Driver.

Library Interface Section	Description
Wi-Fi Initialization Functions	This section provides functions that initialize the MRF24WG Wi-Fi Driver Library and allow its API to be used.
Wi-Fi Network General Configuration Functions	This section provides functions that configure the MRF24WG Wi-Fi Driver interface for use.
Wi-Fi Network Security Configuration Functions	This section provides functions that configure the security mode of the MRF24WG Wi-Fi Driver connection.
Wi-Fi Network Power Configuration Functions	This section provides functions that configure max TX power, as well as power-saving modes for a MRF24WG Wi-Fi Driver connection.
Wi-Fi Network Multicast Filter Configuration Functions	This section provides functions that configure the MRF24WG Wi-Fi Driver software multi-cast filter to receive multi-cast address packets.
Wi-Fi Network Connection Functions	This section provides functions that begin the MRF24WG Wi-Fi Driver connection and disconnection processes.
Wi-Fi Network Configuration - Other Functions	This section provides additional miscellaneous functions for configuring the MRF24WG Wi-Fi Driver connection.
Wi-Fi Scanning Functions	This section provides MRF24WG Wi-Fi Driver functions that initiate a Wi-Fi scan and allow retrieval of the scan results.
Wi-Fi Event Processing Functions	This section provides callback functions that inform the application of MRF24WG Wi-Fi Driver related events.
Wi-Fi Data Configuration Functions	This section provides functions that save and restore MRF24WG Wi-Fi Driver configuration data.
Wi-Fi Status Functions	This section provides functions that retrieve the MRF24WG Wi-Fi Driver connection status.

How the Library Works

This section describes how the MRF24WG Wi-Fi Driver Library operates.

Description

Before the driver is ready for use, its should be configured (compile time configuration). Refer to the Configuring the Library section for more details on how to configure the driver.

There are few run-time configuration items that are done during initialization of the driver instance, and a few that are client-specific and are done using dedicated functions.

To use the MRF24WG Wi-Fi Driver, initialization and client functions should be invoked in a specific sequence to ensure correct operation.

System Initialization

This section describes initialization and reinitialization features.

Description

Wi-Fi initialization configures the MRF24WG module and then directs it to join (or create) a Wi-Fi network. The MRF24WG module defaults to open security and scans all channels in the domain. Therefore, to initialize and connect with the minimum function call overhead in an open security network, the following functions can be used:

```
DRV_WIFI_SsidSet("MySsidName");
DRV_WIFI_Connect();           // start the connection process
```

Alternatively, the following functions could be used to achieve the same effect:

```
DRV_WIFI_ChannelListSet(0);   // 0 = all channels in domain
DRV_WIFI_NetworkTypeSet(DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE);
DRV_WIFI_SecurityOpenSet(DRV_WIFI_SECURITY_OPEN);
DRV_WIFI_SsidSet("MySsidName");
DRV_WIFI_Connect();
```

Client Functionality

This section describes core operation.

Description

From the client perspective, once MRF24WG Wi-Fi Driver initialization is complete and the connection process has started, the client responds to Wi-Fi events. The client is notified of events by the callback function DRV_WIFI_ProcessEvent. The parameters into that function are `event` and `eventInfo`, where `event` is the event code and `eventInfo` is additional information about the event.

Wi-Fi Connection Events

Event Type	Description
DRV_WIFI_EVENT_CONNECTION_SUCCESSFUL	A MRF24WG Wi-Fi Driver connection attempt has completed successfully.
DRV_WIFI_EVENT_CONNECTION_FAILED	A MRF24WG Wi-Fi Driver connection attempt has failed.
DRV_WIFI_EVENT_CONNECTION_TEMPORARILY_LOST	An existing MRF24WG Wi-Fi Driver connection has been temporarily lost (and MRF24WG module is attempting to reconnect).
DRV_WIFI_EVENT_CONNECTION_PERMANENTLY_LOST	An existing MRF24WG Wi-Fi Driver connection has been permanently lost.
DRV_WIFI_EVENT_CONNECTION_REESTABLISHED	A MRF24WG Wi-Fi Driver connection that was temporarily lost has been regained.
DRV_WIFI_EVENT_DISCONNECT_DONE	An application MRF24WG Wi-Fi Driver disconnect request has completed.

Scan Events

Event Type	Description
DRV_WIFI_EVENT_SCAN_RESULTS_READY	The MRF24WG Wi-Fi Driver scan is complete and scan results are ready to be read.

Key Events

Event Type	Description
DRV_WIFI_EVENT_KEY_CALCULATION_REQUEST	The MRF24WG Wi-Fi Driver request key calculation is complete and the binary key is ready to send to the MRF24WG module.

Error Events

Event Type	Description
DRV_WIFI_EVENT_ERROR	A MRF24WG Wi-Fi Driver driver error has occurred.

For some of the event types previously listed, there is an `eventInfo` parameter that provides additional information about the event. Not all event types require the `eventInfo` parameter. The tables in DRV_WIFI_CONNECTION_TEMPORARILY_LOST provide additional detail about the `eventInfo` parameter for those events that use it.

DRV_WIFI_CONNECTION_TEMPORARILY_LOST

`eventInfo` is defined as:

15:8	7	6	5:0
Not used	Deauth	Disassoc	Event Data

Bits 15:8 are not used.

If Deauth = 1 or Disassoc = 1, Event Data is:

Table 1: Deauth or Disassoc Event Data

DRV_WIFI_UNSPECIFIED	1
DRV_WIFI_PREV_AUTH_NOT_VALID	2
DRV_WIFI_DISASSOC_DUE_TO_INACTIVITY	3
DRV_WIFI_DISASSOC_AP_BUSY	4
DRV_WIFI_CLASS2_FRAME_FROM_NONAUTH_STA	6
DRV_WIFI_CLASS3_FRAME_FROM_NONASSOC_STA	7
DRV_WIFI_DISASSOC_STA_HAS_LEFT	8

DRV_WIFI_STA_REQ_ASSOC_WITHOUT_AUTH	9
DRV_WIFI_INVALID_IE	13
DRV_WIFI_MIC_FAILURE	14
DRV_WIFI_4WAY_HANDSHAKE_TIMEOUT	15
DRV_WIFI_GROUP_KEY_HANDSHAKE_TIMEOUT	16
DRV_WIFI_IE_DIFFERENT	17
DRV_WIFI_INVALID_GROUP_CIPHER	18
DRV_WIFI_INVALID_PAIRWISE_CIPHER	19
DRV_WIFI_INVALID_AKMP	20
DRV_WIFI_UNSUPP_RSN_VERSION	21
DRV_WIFI_INVALID_RSN_IE_CAP	22
DRV_WIFI_IEEE8021X_FAILED	23
DRV_WIFI_CIPHER_SUITE_REJECTED	24

If both `Deauth` and `Disassoc` equal '0', Event Data is:

Table 2: Connection Lost Event Data

DRV_WIFI_CIPHER_SUITE_REJECTED	**0**
DRV_WIFI_BEACON_TIMEOUT	1
DRV_WIFI_DEAUTH_RECEIVED	2
DRV_WIFI_DISASSOCIATE_RECEIVED	3
DRV_WIFI_TKIP_MIC_FAILURE	4
DRV_WIFI_LINK_DOWN	5

DRV_WIFI_EVENT_CONNECTION_FAILED

`eventInfo` is defined as:

15:8	**7:0**
Status	Reason

Status is:

Table 1: Connection Failures

DRV_WIFI_JOIN_FAILURE	2
DRV_WIFI_AUTHENTICATION_FAILURE	3
DRV_WIFI_ASSOCIATION_FAILURE	4
DRV_WIFI_WEP_HANDSHAKE_FAILURE	5
DRV_WIFI_PSK_CALCULATION_FAILURE	6
DRV_WIFI_PSK_HANDSHAKE_FAILURE	7
DRV_WIFI_ADHOC_JOIN_FAILURE	8
DRV_WIFI_SECURITY_MISMATCH_FAILURE	9
DRV_WIFI_NO_SUITABLE_AP_FOUND_FAILURE	10
DRV_WIFI_RETRY_FOREVER_NOT_SUPPORTED_FAILURE	11
DRV_WIFI_LINK_LOST	12
DRV_WIFI_TKIP_MIC_FAILURE	13
DRV_WIFI_RSN_MIXED_MODE_NOT_SUPPORTED	14
DRV_WIFI_RECV_DEAUTH	15
DRV_WIFI_RECV_DISASSOC	16
DRV_WIFI_WPS_FAILURE	17

DRV_WIFI_P2P_FAILURE	18
DRV_WIFI_LINK_DOWN	19

If Status = DRV_WIFI_RECV_DEAUTH or Status = DRV_WIFI_RECV_DISASSOC, Reason is one of the values from **Table 2: Connection Lost Event Data** in DRV_WIFI_CONNECTION_TEMPORARILY_LOST.

If Status = DRV_WIFI_AUTHENTICATION_FAILURE or Status = DRV_WIFI_ASSOCIATION_FAILURE, Reason is:

Table 2: Authentication and Association Failures

DRV_WIFI_UNSPECIFIED_FAILURE	1
DRV_WIFI_CAPS_UNSUPPORTED	10
DRV_WIFI_REASSOC_NO_ASSOC	11
DRV_WIFI_ASSOC_DENIED_UNSPEC	12
DRV_WIFI_NOT_SUPPORTED_AUTH_ALG	13
DRV_WIFI_UNKNOWN_AUTH_TRANSACTION	14
DRV_WIFI_CHALLENGE_FAIL	15
DRV_WIFI_AUTH_TIMEOUT	16
DRV_WIFI_AP_UNABLE_TO_HANDLE_NEW_STA	17
DRV_WIFI_ASSOC_DENIED_RATES	18
DRV_WIFI_ASSOC_DENIED_NOSHORTPREAMBLE	19
DRV_WIFI_ASSOC_DENIED_NOPBCC	20
DRV_WIFI_ASSOC_DENIED_NOAGILITY	21
DRV_WIFI_ASSOC_DENIED_NOSHORTTIME	25
DRV_WIFI_ASSOC_DENIED_NODSSSOFDM	26
DRV_WIFI_NOT_VALID_IE	40
DRV_WIFI_NOT_VALID_GROUPCIPHER	41
DRV_WIFI_NOT_VALID_PAIRWISE_CIPHER	42
DRV_WIFI_NOT_VALID_AKMP	43
DRV_WIFI_UNSUPPORTED_RSN_VERSION	44
DRV_WIFI_INVALID_RSN_IE_CAP	45
DRV_WIFI_CIPHER_SUITE_REJECTED	46
DRV_WIFI_TIMEOUT	47

If Status = DRV_WIFI_WPS_FAILURE, Reason is defined as:

7:4	3:0
wpsState	wpsConfigErr

Table 3: wpsState

DRV_WIFI_EAPOL_START	1
DRV_WIFI_EAP_REQ_IDENTITY	2
DRV_WIFI_EAP_RSP_IDENTITY	3
DRV_WIFI_EAP_WPS_START	4
DRV_WIFI_EAP_RSP_M1	5
DRV_WIFI_EAP_REQ_M2	6
DRV_WIFI_EAP_RSP_M3	7
DRV_WIFI_EAP_REQ_M4	8
DRV_WIFI_EAP_RSP_M5	9
DRV_WIFI_EAP_REQ_M6	10
DRV_WIFI_EAP_RSP_M7	11

DRV_WIFI_EAP_REQ_M8	12
DRV_WIFI_EAP_RSP_DONE	13
DRV_WIFI_EAP_RSP_DONE	14

Table 4: wpsConfigErr

DRV_WIFI_WPS_NOERR	0
DRV_WIFI_WPS_SESSION_OVERLAPPED	1
DRV_WIFI_WPS_SESSION_OVERLAPPED	2
DRV_WIFI_WPS_24G_NOT_SUPPORTED	3
DRV_WIFI_WPS_24G_NOT_SUPPORTED	4
DRV_WIFI_WPS_INVALID_MSG	5
DRV_WIFI_WPS_AUTH_FAILURE	6
DRV_WIFI_WPS_ASSOC_FAILURE	7
DRV_WIFI_WPS_MSG_TIMEOUT	8
DRV_WIFI_WPS_SESSION_TIMEOUT	9
DRV_WIFI_WPS_DEVPASSWD_AUTH_FAILURE	10
DRV_WIFI_WPS_NO_CONN_TOREG	11
DRV_WIFI_WPS_MULTI_PBC_DETECTED	12
DRV_WIFI_WPS_EAP_FAILURE	13
DRV_WIFI_WPS_DEV_BUSY	14
DRV_WIFI_WPS_SETUP_LOCKED	15

Configuring the Library

The configuration of the MRF24WG Wi-Fi Driver is based on the file `drv_wifi_config.h` and `drv_wifi_iwpriv.h` files.

This header file contains the configuration selection for the MRF24WG Wi-Fi Driver. Based on the selections made, the MRF24WG Wi-Fi Driver may support the selected features. These configuration settings will apply to all instances of the MRF24WG Wi-Fi Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Configuring DMA SPI

Please refer to Configuring DMA SPI for information on enabling and disabling DMA SPI for FreeRTOS projects.

Sample Functionality

Please refer to the Wi-Fi demonstrations for examples of how to use the various features in the MRF24WG Wi-Fi Driver Library.

Building the Library

This section lists the files that are available in the MRF24WG Wi-Fi Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/wifi/mrf24w`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/drv_wifi.h`	Contains all data types, define constants, and function prototypes for interfacing to the MRF24WG Wi-Fi Driver.
`/drv_wifi_iwpriv.h`	Contains optional (private) parameters to configure the MRF24WG Wi-Fi Driver.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
drv_wifi_com.c	MRF24WG Wi-FI Driver com layer. Provides access to the MRF24W Wi-Fi Driver controller.
drv_wifi_commands.c	MRF24WG Wi-Fi Driver commands (based on system commander) implementation. Provides access to the MRF24WG Wi-Fi Driver controller.
drv_wifi_config_data.c	MRF24WG Wi-FI Driver configuration data. Stores and retrieves MRF24WG Wi-Fi Driver configuration information to Non-volatile Memory (NVM).
drv_wifi_connect.c	MRF24WG Wi-Fi Driver connection support. Functions in this module support the connection process for the MRF24WG Wi-Fi Driver.
drv_wifi_connection_algorithm.c	MRF24WG Wi-Fi Driver connection algorithm. Provides access to the MRF24WG Wi-Fi Driver controller.
drv_wifi_connection_manager.c	MRF24WG Wi-FI Driver connection manager. Provides access to the MRF24W Wi-Fi Driver controller.
drv_wifi_connection_profile.c	MRF24WG Wi-Fi Driver connection profile. Provides access to the MRF24W Wi-Fi Driver controller.
drv_wifi_debug_output.c	MRF24WG Driver Wi-Fi debug console. Provides access to the MRF24W Wi-Fi Driver debug console.
drv_wifi_easy_config.c	MRF24WG Wi-Fi Driver EasyConfig functions. Provides MRF24WG Wi-Fi Driver connection support for Ad hoc and SoftAP mode.
drv_wifi_eint.c	MRF24WG Wi-Fi Driver interface functions support for external interrupts.
drv_wifi_event_handler.c	MRF24WG Wi-Fi Driver event handler. Provides access to the MRF24WG Wi-Fi Driver controller.
drv_wifi_events.c	MRF24WG Wi-Fi Driver MAC events implementation. Processes MRF2WG Wi-Fi Driver events via callback functions.
drv_wifi_init.c	MRF24WG W-Fi Driver initialization. Provides access to the MRF24WG Wi-Fi Driver initialization function.
drv_wifi_mac.c	MRF24WG W-Fi Driver Medium Access Control (MAC) layer. Provides access to the MRF24WG Wi-Fi Driver MAC layer.
drv_wifi_mac_pic32.c	MRF24WG Wi-FI Driver Medium Access Control (MAC) layer.
drv_wifi_mgmt_msg.c	MRF24WG Wi-Fi Driver management message functions.
drv_wifi_param_msg.c	MRF24WG Wi-Fi Driver management Set/Get parameter messages.
drv_wifi_pbkdf2.c	MRF24WG Wi-Fi Driver WPA supplicant. SHA1 Hash implementation and interface functions.
drv_wifi_power_save.c	MRF24WG Wi-Fi Driver power-saving functions.
drv_wifi_raw.c	MRF24WG Wi-Fi Driver raw functions for accessing the MAC layer.
drv_wifi_scan.c	MRF24WG Wi-Fi Driver scan functions.
drv_wifi_spi.c	MRF24WG Wi-Fi Driver SPI functions.
drv_wifi_spi_init.c	MRF24WG Driver SPI interface routines. Initializes the SPI hardware used to communicate with the MRF24WG module.
drv_wifi_tx_power.c	MRF24WG Wi-Fi Driver Transmit (TX) Power functions.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files.

Module Dependencies

The MRF24W Wi-Fi Driver Library depends on the following modules:

- SPI Driver Library
- NVM Driver Library
- Operating System Abstraction Layer (OSAL) Library Help

- Clock System Service Library
- System Service Library Introduction
- Console System Service Library
- File System Service Library
- Interrupt System Service Library
- Timer System Service Library
- Debug System Service Library
- Ports System Service Library

Console Commands

This section describes the console commands available for the MRF24WG Wi-Fi Driver.

Description

Both the Web Server and the EasyConfig demonstrations support the followings commands, which enable control over the Wi-Fi settings.

Command: eraseconf

Parameters	Description
None.	Wi-Fi console command to erase saved Wi-Fi configuration in memory.

Command: iwconfig

Parameters	Description
[ssid <name>]	name: Specifies the name of the SSID (1-32 ASCII characters).
[mode <idle | managed>]	idle: Disconnected from the current configuration. managed: Connects in infrastructure mode to the currently set SSID.
[power <enable | disable>]	enable: Enables all Power-Saving features (PS_POLL). Will wake up to check for all types of traffic (unicast, multicast, and broadcast). disable: Disables any Power-Saving features. Will always be in an active power state.
[security <mode>]	mode: open/wep40/wep104/wpa/wpa2/pin/pbc. For example: iwconfig security open iwconfig security wep40 <key> iwconfig security wep104 <key> iwconfig security wpa <key> iwconfig security wpa2 <key> iwconfig security pin <pin> iwconfig security pbc
[scan]	Starts a Wi-Fi scan.
[scanget <scan_index>]	scan_index: Retrieves the scan result after the scan completes (1 - n).

Command: mac

Parameters	Description
None.	Wi-Fi console command to retrieve the MAC address of the MRF24WN module.

Command: readconf

Parameters	Description
None.	Wi-Fi console command to read saved Wi-Fi configuration in memory.

Command: saveconf

Parameters	Description
None.	Wi-Fi console command to save Wi-Fi configuration to memory.

Library Interface

a) Wi-Fi Initialization Functions

	Name	Description
	DRV_WIFI_Initialize	Initializes the MRF24WG Wi-Fi driver. **Implementation:** Dynamic
	DRV_WIFI_Deinitialize	De-initializes the MRF24WG Wi-Fi driver. **Implementation:** Dynamic
	DRV_WIFI_MRF24W_ISR	MRF24WG Wi-Fi driver interrupt service routine. **Implementation:** Dynamic
	DRV_WIFI_Deferred_ISR	Implements MRF24WG Wi-Fi driver deferred ISR. **Implementation:** Dynamic
	DRV_WIFI_DeferredISR_SemGive	Gives semaphore to Wi-Fi deferred ISR. **Implementation:** Dynamic
	DRV_WIFI_InitTask	Implements MRF24WG Wi-Fi driver initialization RTOS task. **Implementation:** Dynamic
	DRV_WIFI_MACTask	Implements MRF24WG Wi-Fi driver MAC process RTOS task. **Implementation:** Dynamic

b) Wi-Fi Network General Configuration Functions

	Name	Description
	DRV_WIFI_ChannelListSet	Sets the channel list. **Implementation:** Dynamic
	DRV_WIFI_NetworkTypeSet	Sets the Wi-Fi network type. **Implementation:** Dynamic
	DRV_WIFI_ReconnectModeSet	Sets the Wi-Fi reconnection mode. **Implementation:** Dynamic
	DRV_WIFI_SsidSet	Sets the SSID. **Implementation:** Dynamic
	DRV_WIFI_AdhocContextSet	Sets the Ad-Hoc context. **Implementation:** Dynamic
	DRV_WIFI_LinkDownThresholdSet	Sets number of consecutive Wi-Fi TX failures before link is considered down. **Implementation:** Dynamic
	DRV_WIFI_ContextLoad	Loads Wi-Fi context to MRF24WG. **Implementation:** Dynamic

c) Wi-Fi Network Security Configuration Functions

	Name	Description
	DRV_WIFI_SetPSK	Sets the binary WPA PSK code in WPS. **Implementation:** Dynamic
	DRV_WIFI_SecurityOpenSet	Sets Wi-Fi security to open (no security). **Implementation:** Dynamic
	DRV_WIFI_SecurityWpaSet	Sets Wi-Fi security to use WPA or WPA2. **Implementation:** Dynamic
	DRV_WIFI_SecurityWpsSet	Sets Wi-Fi security to use WPS. **Implementation:** Dynamic
	DRV_WIFI_SecurityWepSet	Sets Wi-Fi security to use WEP. **Implementation:** Dynamic

d) Wi-Fi Network Power Configuration Functions

	Name	Description
	DRV_WIFI_HibernateEnable	Puts the MRF24WG module into hibernate mode. **Implementation:** Dynamic
	DRV_WIFI_HibernateModeClear	Clears current Hibernate mode. **Implementation:** Dynamic
	DRV_WIFI_PsPollDisable	Disables PS-Poll mode. **Implementation:** Dynamic

	DRV_WIFI_PsPollEnable	Enables PS Poll mode. **Implementation:** Dynamic

e) Wi-Fi Network Multicast Filter Configuration Functions

	Name	Description
	DRV_WIFI_MulticastFilterSet	Sets a multicast address filter using one of the software multicast filters. **Implementation:** Dynamic

f) Wi-Fi Network Connection Functions

	Name	Description
	DRV_WIFI_Connect	Directs the MRF24WG to connect to a Wi-Fi network. **Implementation:** Dynamic
	DRV_WIFI_Disconnect	Directs the MRF24WG to disconnect from a Wi-Fi network. **Implementation:** Dynamic

g) Wi-Fi Network Configuration - Other Functions

	Name	Description
	DRV_WIFI_BssidSet	Sets the Basic Service Set Identifier (BSSID). **Implementation:** Dynamic
	DRV_WIFI_MacAddressSet	Uses a different MAC address for the MRF24WG. **Implementation:** Dynamic
	DRV_WIFI_RtsThresholdSet	Sets the RTS Threshold. **Implementation:** Dynamic
	DRV_WIFI_ScanContextSet	Sets the Wi-Fi scan context. **Implementation:** Dynamic
	DRV_WIFI_TxModeSet	Configures 802.11 TX mode. **Implementation:** Dynamic
	DRV_WIFI_RssiSet	Sets RSSI restrictions when connecting. **Implementation:** Dynamic
	DRV_WIFI_RSSI_Cache_FromRxDataRead	Caches RSSI value from RX data packet. **Implementation:** Dynamic
	DRV_WIFI_RSSI_Get_FromRxDataRead	Reads RSSI value from RX data packet. **Implementation:** Dynamic
	DRV_WIFI_INT_Handle	MRF24WG Wi-Fi driver interrupt handle. **Implementation:** Dynamic
	DRV_WIFI_SpiClose	Closes SPI object for MRF24WG Wi-Fi driver. **Implementation:** Dynamic
	DRV_WIFI_TaskSyncInit	Initializes RTOS Semaphore and Mutex for MRF24WG Wi-Fi driver. **Implementation:** Dynamic

h) Wi-Fi Scanning Functions

	Name	Description
	DRV_WIFI_Scan	Commands the MRF24WG module to start a scan operation. This will generate the WF_EVENT_SCAN_RESULTS_READY event. **Implementation:** Dynamic
	DRV_WIFI_ScanContextGet	Gets the Wi-Fi scan context. **Implementation:** Dynamic

i) Wi-Fi Event Processing Functions

	Name	Description
	DRV_WIFI_ProcessEvent	Processes Wi-Fi event. **Implementation:** Dynamic
	DRV_WIFI_SoftApEventInfoGet	Gets the stored Soft AP event info. **Implementation:** Dynamic
	DRV_WIFI_SoftAPContextSet	Sets the Soft AP context. **Implementation:** Dynamic

j) Wi-Fi Data Configuration Functions

	Name	Description
	DRV_WIFI_ConfigDataLoad	Loads configuration data from the board EEPROM. **Implementation:** Dynamic
	DRV_WIFI_ConfigDataSave	Save configuration data to the board EEPROM. **Implementation:** Dynamic
	DRV_WIFI_ConfigDataDelete	Erases configuration data from the board EEPROM. **Implementation:** Dynamic

k) Wi-Fi Status Functions

	Name	Description
	DRV_WIFI_BssidGet	Gets the BSSID set in DRV_WIFI_BssidSet(). **Implementation:** Dynamic
	DRV_WIFI_ChannelListGet	Gets the channel list. **Implementation:** Dynamic
	DRV_WIFI_ConnectContextGet	Gets the current Wi-Fi connection context. **Implementation:** Dynamic
	DRV_WIFI_ConnectionStateGet	Gets the current Wi-Fi connection state. **Implementation:** Dynamic
	DRV_WIFI_DeviceInfoGet	Retrieves MRF24WG device information. **Implementation:** Dynamic
	DRV_WIFI_MacAddressGet	Retrieves the MRF24WG MAC address. **Implementation:** Dynamic
	DRV_WIFI_MacStatsGet	Gets MAC statistics. **Implementation:** Dynamic
	DRV_WIFI_NetworkTypeGet	Gets the Wi-Fi network type. **Implementation:** Dynamic
	DRV_WIFI_PowerSaveStateGet	Gets the current power-saving state.
	DRV_WIFI_ReconnectModeGet	Gets the Wi-Fi reconnection mode. **Implementation:** Dynamic
	DRV_WIFI_RegionalDomainGet	Retrieves the MRF24WG Regional domain. **Implementation:** Dynamic
	DRV_WIFI_RtsThresholdGet	Gets the RTS Threshold. **Implementation:** Dynamic
	DRV_WIFI_SecurityGet	Gets the current Wi-Fi security setting. **Implementation:** Dynamic
	DRV_WIFI_SsidGet	Gets the SSID. **Implementation:** Dynamic
	DRV_WIFI_TxModeGet	Gets 802.11 TX mode. **Implementation:** Dynamic
	DRV_WIFI_WepKeyTypeGet	Gets the WEP Key type. **Implementation:** Dynamic
	DRV_WIFI_WPSCredentialsGet	Gets the WPS credentials. **Implementation:** Dynamic
	DRV_WIFI_RssiGet	Gets RSSI value set in DRV_WIFI_RssiSet(). **Implementation:** Dynamic
	DRV_WIFI_SecurityTypeGet	Gets the current Wi-Fi security type. **Implementation:** Dynamic
	DRV_WIFI_TxPowerFactoryMaxGet	Retrieves the factory-set max TX power from the MRF24WG module. **Implementation:** Dynamic
	DRV_WIFI_TxPowerMaxGet	Gets the TX max power on the MRF24WG0M. **Implementation:** Dynamic
	DRV_WIFI_TxPowerMaxSet	Sets the TX max power on the MRF24WG0M. **Implementation:** Dynamic
	DRV_WIFI_ScanResultGet	Read selected scan results back from MRF24WG. **Implementation:** Dynamic

	DRV_WIFI_InHibernateMode	Checks if MRF24WG is in hibernate mode. **Implementation:** Dynamic

l) Wi-Fi SPI Functions

	Name	Description
	DRV_WIFI_SpiDmaRx	SPI RX API using DMA. **Implementation:** Dynamic
	DRV_WIFI_SpiDmaTx	SPI TX API using DMA. **Implementation:** Dynamic
	DRV_WIFI_SpiInit	Initializes SPI object for MRF24WG Wi-Fi driver. **Implementation:** Dynamic
	DRV_WIFI_SpiRx	SPI Rx API. **Implementation:** Dynamic
	DRV_WIFI_SpiTx	SPI TX API **Implementation:** Dynamic

m) Private Configuration Functions

	Name	Description
	iwpriv_adhocctx_set	Sets the Ad-Hoc network context information. **Implementation:** Dynamic
	iwpriv_config_read	Reads the Wi-Fi context configuration. **Implementation:** Dynamic

n) Data Types and Constants

	Name	Description
	DRV_WIFI_BSSID_LENGTH	This is macro DRV_WIFI_BSSID_LENGTH.
	DRV_WIFI_DEAUTH_REASONCODE_MASK	This is macro DRV_WIFI_DEAUTH_REASONCODE_MASK.
	DRV_WIFI_DEFAULT_ADHOC_BEACON_PERIOD	ms
	DRV_WIFI_DEFAULT_ADHOC_HIDDEN_SSID	Default values for Wi-Fi Ad-Hoc settings.
	DRV_WIFI_DEFAULT_ADHOC_MODE	This is macro DRV_WIFI_DEFAULT_ADHOC_MODE.
	DRV_WIFI_DEFAULT_PS_DTIM_ENABLED	DTIM wake-up enabled (normally the case)
	DRV_WIFI_DEFAULT_PS_DTIM_INTERVAL	number of beacon periods
	DRV_WIFI_DEFAULT_PS_LISTEN_INTERVAL	100 ms multiplier, e.g., 1 * 100 ms = 100 ms
	DRV_WIFI_DEFAULT_SCAN_COUNT	Default values for Wi-Fi scan context
	DRV_WIFI_DEFAULT_SCAN_MAX_CHANNEL_TIME	ms
	DRV_WIFI_DEFAULT_SCAN_MIN_CHANNEL_TIME	ms
	DRV_WIFI_DEFAULT_SCAN_PROBE_DELAY	us
	DRV_WIFI_DEFAULT_WEP_KEY_TYPE	This is macro DRV_WIFI_DEFAULT_WEP_KEY_TYPE.
	DRV_WIFI_DISABLED	This is macro DRV_WIFI_DISABLED.
	DRV_WIFI_DISASSOC_REASONCODE_MASK	This is macro DRV_WIFI_DISASSOC_REASONCODE_MASK.
	DRV_WIFI_ENABLED	Do not make this an enumerated type!
	DRV_WIFI_MAX_CHANNEL_LIST_LENGTH	This is macro DRV_WIFI_MAX_CHANNEL_LIST_LENGTH.
	DRV_WIFI_MAX_NUM_RATES	This is macro DRV_WIFI_MAX_NUM_RATES.
	DRV_WIFI_MAX_SECURITY_KEY_LENGTH	This is macro DRV_WIFI_MAX_SECURITY_KEY_LENGTH.
	DRV_WIFI_MAX_SSID_LENGTH	This is macro DRV_WIFI_MAX_SSID_LENGTH.
	DRV_WIFI_MAX_WEP_KEY_LENGTH	This is macro DRV_WIFI_MAX_WEP_KEY_LENGTH.
	DRV_WIFI_MAX_WPA_PASS_PHRASE_LENGTH	must exclude string terminator
	DRV_WIFI_MIN_WPA_PASS_PHRASE_LENGTH	must exclude string terminator
	DRV_WIFI_NETWORK_TYPE_ADHOC	This is macro DRV_WIFI_NETWORK_TYPE_ADHOC.
	DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE	Selection of different Wi-Fi network types.
	DRV_WIFI_NETWORK_TYPE_SOFT_AP	This is macro DRV_WIFI_NETWORK_TYPE_SOFT_AP.
	DRV_WIFI_NO_ADDITIONAL_INFO	eventInfo define for DRV_WIFI_ProcessEvent() when no additional info is supplied
	DRV_WIFI_RETRY_ADHOC	This is macro DRV_WIFI_RETRY_ADHOC.
	DRV_WIFI_RETRY_FOREVER	This is macro DRV_WIFI_RETRY_FOREVER.
	DRV_WIFI_RTS_THRESHOLD_MAX	maximum RTS threshold size in bytes

	DRV_WIFI_SECURITY_OPEN	Selection of different Wi-Fi security types
	DRV_WIFI_SECURITY_WEP_104	This is macro DRV_WIFI_SECURITY_WEP_104.
	DRV_WIFI_SECURITY_WEP_40	This is macro DRV_WIFI_SECURITY_WEP_40.
	DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY	This is macro DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY.
	DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE	This is macro DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE.
	DRV_WIFI_SECURITY_WPA_WITH_KEY	This is macro DRV_WIFI_SECURITY_WPA_WITH_KEY.
	DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE	This is macro DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE.
	DRV_WIFI_SECURITY_WPA2_WITH_KEY	This is macro DRV_WIFI_SECURITY_WPA2_WITH_KEY.
	DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE	This is macro DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE.
	DRV_WIFI_SECURITY_WPS_PIN	This is macro DRV_WIFI_SECURITY_WPS_PIN.
	DRV_WIFI_SECURITY_WPS_PUSH_BUTTON	This is macro DRV_WIFI_SECURITY_WPS_PUSH_BUTTON.
	DRV_WIFI_WEP104_KEY_LENGTH	4 keys of 13 bytes each
	DRV_WIFI_WEP40_KEY_LENGTH	4 keys of 5 bytes each
	DRV_WIFI_WPA_KEY_LENGTH	This is macro DRV_WIFI_WPA_KEY_LENGTH.
	DRV_WIFI_WPS_PIN_LENGTH	7 digits + checksum byte
	DRV_WIFI_ADHOC_MODES	Selection of different Ad-Hoc connection modes.
	DRV_WIFI_ADHOC_NETWORK_CONTEXT	Contains data pertaining to Wi-Fi Ad-Hoc context.
	DRV_WIFI_CONNECTION_CONTEXT	Contains data pertaining to MRF24WG connection context.
	DRV_WIFI_CONNECTION_STATES	Wi-Fi Connection States
	DRV_WIFI_DEVICE_INFO	Contains data pertaining to MRF24WG device type and version number.
	DRV_WIFI_DOMAIN_CODES	Wi-Fi Regional Domain Codes
	DRV_WIFI_EVENT_CONN_TEMP_LOST_CODES	Selection of different codes when Wi-Fi connection is temporarily lost.
	DRV_WIFI_EVENT_INFO	Selection of different EventInfo types.
	DRV_WIFI_EVENTS	Selections for events that can occur.
	DRV_WIFI_GENERAL_ERRORS	This is type DRV_WIFI_GENERAL_ERRORS.
	DRV_WIFI_HIBERNATE_STATES	Wi-Fi Hibernate States
	DRV_WIFI_MAC_STATS	Wi-Fi MIB states
	DRV_WIFI_MGMT_ERRORS	Error codes returned when a management message is sent to the MRF24WG module.
	DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT	Contains data pertaining to Wi-Fi Soft AP event.
	DRV_WIFI_MULTICAST_FILTER_IDS	Selections for software Multicast filter IDs.
	DRV_WIFI_MULTICAST_FILTERS	Selections for Software Multicast Filters.
	DRV_WIFI_POWER_SAVE_STATES	Wi-Fi Power-Saving States
	DRV_WIFI_PS_POLL_CONTEXT	Contains data pertaining to Wi-Fi PS-Poll context.
	DRV_WIFI_REASON_CODES	Selection of different codes when a deauthorization or disassociation event has occurred.
	DRV_WIFI_RECONNECT_MODES	Selection of different reconnection modes.
	DRV_WIFI_SCAN_CONTEXT	Contains data pertaining to Wi-Fi scan context.
	DRV_WIFI_SCAN_RESULT	Contains data pertaining to Wi-Fi scan results.
	DRV_WIFI_SCAN_TYPES	Selection of different Wi-Fi scan types.
	DRV_WIFI_SOFT_AP_EVENT_REASON_CODES	Wi-Fi Soft AP Event Reason Codes
	DRV_WIFI_SOFT_AP_STATES	Wi-Fi Soft AP Events
	DRV_WIFI_STATUS_CODES	Selection of different codes when Wi-Fi connection fails due to association or authentication failure.
	DRV_WIFI_TX_MODES	Selections for Wi-Fi TX Mode
	DRV_WIFI_WEP_CONTEXT	Contains data pertaining to Wi-Fi WEP context.
	DRV_WIFI_WEP_KEY_TYPE	Selections for WEP key type when using WEP security.
	DRV_WIFI_WPA_CONTEXT	Contains data pertaining to Wi-Fi WPA.
	DRV_WIFI_WPA_KEY_INFO	Contains data pertaining to Wi-Fi WPA Key.
	DRV_WIFI_WPS_AUTH_TYPES	Selection of WPS Authorization Types
	DRV_WIFI_WPS_CONTEXT	Contains data pertaining to Wi-Fi WPS security.
	DRV_WIFI_WPS_CREDENTIAL	Contains data pertaining to Wi-Fi WPS Credentials.

	DRV_WIFI_WPS_ENCODE_TYPES	Selection of WPS Encoding Types
	adhocMode	Selection of different Ad-Hoc connection modes.
	DRV_WIFI_WPS_ERROR_CONFIG_CODES	Selection of different codes when a WPS connection fails.
	DRV_WIFI_WPS_STATE_CODES	Selection of different codes when a Extensible Authentication Protocol is used.
	DRV_WIFI_DEFAULT_WEP_KEY_INDEX	see DRV_WIFI_SecurityWepSet() and DRV_WIFI_WEP_CONTEXT
	DRV_WIFI_SOFTAP_NETWORK_CONTEXT	Contains data pertaining to Wi-Fi Soft AP context.
	DRV_WIFI_DEFAULT_SOFTAP_HIDDEN_SSID	Default values for Wi-Fi Soft AP settings.
	DRV_WIFI_MULTICAST_CONFIG	Contains data pertaining to Wi-Fi software multicast filter configuration.
	DRV_WIFI_APCONFIG_BIT_PREAMBLE_LONG	This is macro DRV_WIFI_APCONFIG_BIT_PREAMBLE_LONG.
	DRV_WIFI_APCONFIG_BIT_PRIVACY	apConfig bit
	DRV_WIFI_APCONFIG_BIT_WPA	This is macro DRV_WIFI_APCONFIG_BIT_WPA.
	DRV_WIFI_APCONFIG_BIT_WPA2	This is macro DRV_WIFI_APCONFIG_BIT_WPA2.
	DRV_WIFI_DEVICE_TYPE	Codes for MRF Wi-Fi Device Type

Description

This section describes the Application Programming Interface (API) functions of the MRF24WG Wi-Fi Driver.

Refer to each section for a detailed description.

a) Wi-Fi Initialization Functions

DRV_WIFI_Initialize Function

Initializes the MRF24WG Wi-Fi driver.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_Initialize();
```

Returns

None.

Description

This function initializes the MRF24WG driver, making it ready for clients to use.

Remarks

This function must be called before any other Wi-Fi routine is called. The real work of Wi-Fi initialization takes place in an internal state machine, whose state is set to the initial value by this function.

Preconditions

None.

Function

void DRV_WIFI_Initialize(void)

DRV_WIFI_Deinitialize Function

De-initializes the MRF24WG Wi-Fi driver.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_Deinitialize();
```

Returns

None.

Description

This function de-initializes the MRF24WG driver. It also saves the Wi-Fi parameters in non-volatile storage.

Remarks

None.

Preconditions

None.

Function

void DRV_WIFI_Deinitialize(void)

DRV_WIFI_MRF24W_ISR Function

MRF24WG Wi-Fi driver interrupt service routine.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_MRF24W_ISR(SYS_MODULE_OBJ index);
```

Returns

None.

Description

This function is MRF24WG Wi-Fi driver interrupt service routine.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Function

void DRV_WIFI_MRF24W_ISR(SYS_MODULE_OBJ index)

DRV_WIFI_Deferred_ISR Function

Implements MRF24WG Wi-Fi driver deferred ISR.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_Deferred_ISR(void * p_arg);
```

Returns

None.

Description

This function implements MRF24WG Wi-Fi driver deferred ISR.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void DRV_WIFI_Deferred_ISR(void *p_arg)

DRV_WIFI_DeferredISR_SemGive Function

Gives semaphore to Wi-Fi deferred ISR.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_DeferredISR_SemGive();
```

Returns

None.

Description

This function gives semaphore to Wi-Fi Deferred ISR.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void DRV_WIFI_DeferredISR_SemGive(void)

DRV_WIFI_InitTask Function

Implements MRF24WG Wi-Fi driver initialization RTOS task.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_InitTask(void * p_arg);
```

Returns

None.

Description

This function implements MRF24WG Wi-Fi driver initialization RTOS task.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void DRV_WIFI_InitTask(void *p_arg)

DRV_WIFI_MACTask Function

Implements MRF24WG Wi-Fi driver MAC process RTOS task.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_MACTask(void * p_arg);
```

Returns

None.

Description

This function implements MRF24WG Wi-Fi driver MAC process RTOS task.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void DRV_WIFI_MACTask(void *p_arg)

b) Wi-Fi Network General Configuration Functions

DRV_WIFI_ChannelListSet Function

Sets the channel list.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_ChannelListSet(uint8_t * p_channelList, uint8_t numChannels);
```

Returns

None.

Description

This function sets the channel list that the MRF24WG will use when scanning or connecting.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t channelList[1, 6, 11];

DRV_WIFI_ChannelListSet(channelList, sizeof(channelList));
```

Parameters

Parameters	Description
p_channelList	list of channels

numChannels	number of channels in list; if set to 0, then MRF24WG will set its channel list to all valid channels in its regional domain

Function

void DRV_WIFI_ChannelListSet(uint8_t *p_channelList, uint8_t numChannels)

DRV_WIFI_NetworkTypeSet Function

Sets the Wi-Fi network type.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_NetworkTypeSet(uint8_t networkType);
```

Returns

None.

Description

This function selects the Wi-Fi network type.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_NetworkTypeSet(DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE);
```

Parameters

Parameters	Description
networkType	one of the following: DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE DRV_WIFI_NETWORK_TYPE_ADHOC DRV_WIFI_NETWORK_TYPE_SOFT_AP

Function

void DRV_WIFI_NetworkTypeSet(uint8_t networkType)

DRV_WIFI_ReconnectModeSet Function

Sets the Wi-Fi reconnection mode.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_ReconnectModeSet(uint8_t retryCount, uint8_t deauthAction, uint8_t beaconTimeout, uint8_t
beaconTimeoutAction);
```

Returns

None.

Description

This function controls how the MRF24WG behaves when an existing Wi-Fi connection is lost. The MRF24WG can lose an existing connection in one of two ways: 1) Beacon timeout; 2) Deauthorization received from AP.

There are two options with respect to regaining a lost Wi-Fi connection: 1) MRF24WG informs the host that the connection was temporarily lost and then the MRF24WG retries N times (or forever) to regain the connection. 2) MRF24WG simply informs the host application that the connection is lost, and it is up to the host to regain the connection via the API.

Remarks

The retryCount parameter also applies when initially connecting. That is, the retryCount tells the MRF24WG how many time to try to connect to a Wi-Fi network before giving up and generating the DRV_WIFI_EVENT_CONNECTION_FAILED event.

'retryCount' field	Description
0	Do not try to regain a connection (simply report event to host)
1:254	Number of times MRF24WG should try to regain the connection
255	MRF24WG will retry forever (do not use for Ad-Hoc connections)

'deauthAction' field	Description
DRV_WIFI_DO_NOT_ATTEMPT_TO_RECONNECT	Do not attempt to reconnect after a deauth
DRV_WIFI_ATTEMPT_TO_RECONNECT	Attempt to reconnect after a deauth

'beaconTimeout' field	Description
0	MRF24WG will not monitor the beacon timeout condition
1:255	Number of missed beacons before designating the connection as lost.

'beaconTimeoutAction' field	Description
DRV_WIFI_DO_NOT_ATTEMPT_TO_RECONNECT	Do not attempt to reconnect after a beacon timeout
DRV_WIFI_ATTEMPT_TO_RECONNECT	Attempt to reconnect after a beacon timeout

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
// Example 1: MRF24WG should retry forever if either a deauth or beacon
// timeout occurs (beacon timeout is 3 beacon periods).
DRV_WIFI_ReconnectModeSet(WF_RETRY_FOREVER,
                          WF_ATTEMPT_TO_RECONNECT,
                          3,
                          WF_ATTEMPT_TO_RECONNECT);

// Example 2: MRF24WG should not do any connection retries and only report
// deauthorization events to the host.
DRV_WIFI_ReconnectModeSet(0,
                          WF_DO_NOT_ATTEMPT_TO_RECONNECT,
                          0,
                          WF_DO_NOT_ATTEMPT_TO_RECONNECT);

// Example 3: MRF24WG should not do any connection retries, but report deauthorization
// and beacon timeout events to host.  Beacon timeout should be 5 beacon periods.
DRV_WIFI_ReconnectModeSet(0,
                          WF_DO_NOT_ATTEMPT_TO_RECONNECT,
                          5,
                          WF_DO_NOT_ATTEMPT_TO_RECONNECT);

// Example 4: MRF24WG should ignore beacon timeouts, but attempt to
// reconnect 3 times if a deauthorization occurs.
DRV_WIFI_ReconnectModeSet(3,
                          WF_ATTEMPT_TO_RECONNECT,
                          0,
                          WF_DO_NOT_ATTEMPT_TO_RECONNECT);
```

Parameters

Parameters	Description
retryCount	number of times the MRF24WG should try to regain the connection (see description)
deauthAction	in the event of a deauthorization from the AP, the action the MRF24WG should take (see description)
beaconTimeout	number of missed beacons before the MRF24WG designates the connection as lost (see description)
beaconTimeoutAction	in the event of a beacon timeout, the action the MRF24WG should take (see description)

Function

void DRV_WIFI_ReconnectModeSet(uint8_t retryCount, uint8_t deauthAction,

uint8_t beaconTimeout, uint8_t beaconTimeoutAction)

DRV_WIFI_SsidSet Function

Sets the SSID.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_SsidSet(uint8_t * p_ssid, uint8_t ssidLen);
```

Returns

None.

Description

Sets the SSID and SSID Length. Note that an Access Point can have either a visible or hidden SSID. If an Access Point uses a hidden SSID then an active scan must be used.

Remarks

Do not include a string terminator in the SSID length. SSIDs are case-sensitive. SSID length must be less than or equal to DRV_WIFI_MAX_SSID_LENGTH.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t ssid[] = "MySSIDName";
uint8_t ssidLength = strlen(ssid);

DRV_WIFI_SsidSet(ssid, &ssidLen);
```

Parameters

Parameters	Description
p_ssid	pointer to SSID buffer
ssidLength	number of bytes in SSID

Function

void DRV_WIFI_SsidSet(uint8_t *p_ssid, uint8_t ssidLen)

DRV_WIFI_AdhocContextSet Function

Sets the Ad-Hoc context.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_AdhocContextSet(DRV_WIFI_ADHOC_NETWORK_CONTEXT * p_context);
```

Returns

None.

Description

This function sets the Ad-Hoc context. It is only applicable when the DRV_WIFI_NETWORK_TYPE_ADHOC has been selected in DRV_WIFI_NetworkTypeSet.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_ADHOC_NETWORK_CONTEXT adHocContext;

adHocContext.mode = DRV_WIFI_ADHOC_CONNECT_THEN_START;
adHocContext.hiddenSsid = false;
adHocContext.beaconPeriod = DRV_WIFI_DEFAULT_ADHOC_BEACON_PERIOD;

DRV_WIFI_AdhocContextSet(&adHocContext);
```

Parameters

Parameters	Description
p_context	pointer to Ad-Hoc context data; see definition for the DRV_WIFI_ADHOC_NETWORK_CONTEXT structure.

Function

void DRV_WIFI_AdhocContextSet(DRV_WIFI_ADHOC_NETWORK_CONTEXT *p_context)

DRV_WIFI_LinkDownThresholdSet Function

Sets number of consecutive Wi-Fi TX failures before link is considered down.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_LinkDownThresholdSet(uint8_t threshold);
```

Returns

None.

Description

This function allows the application to set the number of MRF24WG consecutive TX failures before the connection failure event (DRV_WIFI_LINK_LOST) is reported to the host application.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_LinkDownThresholdSet(0); // disable link down threshold
```

Parameters

Parameters	Description
threshold	0: disabled (default)
1-255	number of consecutive TX failures before connection failure event is reported

Function

void DRV_WIFI_LinkDownThresholdSet(uint8_t threshold)

DRV_WIFI_ContextLoad Function

Loads Wi-Fi context to MRF24WG.

Implementation: Dynamic

File

drv_wifi.h

C

```
TCPIP_MAC_RES DRV_WIFI_ContextLoad();
```

Returns

TCP/IP stack MAC result.

Description

This function loads Wi-Fi context to MRF24WG.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

TCPIP_MAC_RES DRV_WIFI_ContextLoad(void)

c) Wi-Fi Network Security Configuration Functions

DRV_WIFI_SetPSK Function

Sets the binary WPA PSK code in WPS.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_SetPSK(uint8_t * p_psk);
```

Returns

None.

Description

This function is used in conjunction with DRV_WIFI_YieldPassphraseToHost. It sends the binary key to the MRF24WG after the host has converted an ASCII passphrase to a binary key.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_YieldPassphraseToHost(&info);
```

Parameters

Parameters	Description
p_psk	pointer to the binary key

Function

void DRV_WIFI_SetPSK(uint8_t *p_psk)

DRV_WIFI_SecurityOpenSet Function

Sets Wi-Fi security to open (no security).

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_SecurityOpenSet();
```

Returns

None.

Description

This function sets the Wi-Fi security to open. One can only connect to an AP that is running in open mode.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete. Must be in an unconnected state.

Example

```
DRV_WIFI_SecurityOpenSet();
```

Function

void DRV_WIFI_SecurityOpenSet(void)

DRV_WIFI_SecurityWpaSet Function

Sets Wi-Fi security to use WPA or WPA2.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_SecurityWpaSet(DRV_WIFI_WPA_CONTEXT* p_context);
```

Returns

None.

Description

This function sets the Wi-Fi security to WPA or WPA2. One can only connect to an AP that is running the same WPA mode.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete. Must be in an unconnected state.

Example

```
DRV_WIFI_WPA_CONTEXT context;

context.wpaSecurityType  = DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE
context.keyInfo.key[]    = "MySecretWPAPassPhrase";
context.keyInfo.keyLenth = strlen(context.keyInfo.key);
DRV_WIFI_SecurityWpaSet(&context);
```

Parameters

Parameters	Description
p_context	desired WPA context, see DRV_WIFI_WPA_CONTEXT structure

Function

DRV_WIFI_SecurityWpaSet(DRV_WIFI_WPA_CONTEXT* p_context)

DRV_WIFI_SecurityWpsSet Function

Sets Wi-Fi security to use WPS.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_SecurityWpsSet(DRV_WIFI_WPS_CONTEXT * p_context);
```

Returns

None.

Description

This function sets the Wi-Fi security to WPS. One can only connect to an AP that supports WPS.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete. Must be in an unconnected state.

Example

```
DRV_WIFI_WPS_CONTEXT context;
uint8_t wpsPin[8] = {1, 2, 3, 9, 0, 2, 1, 2};

context.wpsSecurityType = DRV_WIFI_SECURITY_WPS_PUSH_BUTTON;
memcpy(context.wpsPin, wpsPin, sizeof(wpsPin));
context.wpsPinLength = 8;
DRV_WIFI_SecurityWpsSet(&context);
```

Parameters

Parameters	Description
p_context	desired WPA context, see DRV_WIFI_WPS_CONTEXT structure

Function

void DRV_WIFI_SecurityWpsSet(DRV_WIFI_WPS_CONTEXT *p_context)

DRV_WIFI_SecurityWepSet Function

Sets Wi-Fi security to use WEP.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_SecurityWepSet(DRV_WIFI_WEP_CONTEXT * p_context);
```

Returns

None.

Description

This function sets the Wi-Fi security to WEP. One can only connect to an AP that is running the same WEP mode.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete. Must be in an unconnected state.

Example

```
DRV_WIFI_WEP_CONTEXT context;

context.wepSecurityType = DRV_WIFI_SECURITY_WEP_40;
context.wepKey[] = {0x5a, 0xfb, 0x6c, 0x8e, 0x77,
                    0xc1, 0x04, 0x49, 0xfd, 0x4e,
                    0x43, 0x18, 0x2b, 0x33, 0x88,
                    0xb0, 0x73, 0x69, 0xf4, 0x78};

context.wepKeyLength = 20;
context.wepKeyType = DRV_WIFI_SECURITY_WEP_OPENKEY;
DRV_WIFI_SecurityOpenSet(&context);
```

Parameters

Parameters	Description
p_context	desired WEP context, see DRV_WIFI_WEP_CONTEXT structure

Function

void DRV_WIFI_SecurityWepSet(DRV_WIFI_WEP_CONTEXT *p_context)

d) Wi-Fi Network Power Configuration Functions

DRV_WIFI_HibernateEnable Function

Puts the MRF24WG module into hibernate mode.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_HibernateEnable();
```

Returns

None.

Description

This function enables hibernate mode on the MRF24WG module, which effectively turns off the device for maximum power savings.

MRF24WG module state is not maintained when it transitions to hibernate mode.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_HibernateEnable();
```

Function

void DRV_WIFI_HibernateEnable(void)

DRV_WIFI_HibernateModeClear Function

Clears current Hibernate mode.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_HibernateModeClear();
```

Returns

None.

Description

This function clears the current Hibernate mode.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Function

bool DRV_WIFI_HibernateModeClear(void)

DRV_WIFI_PsPollDisable Function

Disables PS-Poll mode.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_PsPollDisable();
```

Returns

None.

Description

Disables PS Poll mode. The MRF24WG module will stay active and not go to sleep.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_PsPollDisable(&context);
```

Function

void DRV_WIFI_PsPollDisable(void)

DRV_WIFI_PsPollEnable Function

Enables PS Poll mode.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_PsPollEnable(DRV_WIFI_PS_POLL_CONTEXT * p_context);
```

Returns

None.

Description

Enables PS Poll mode. PS Poll (Power-Save Poll) is a mode allowing for longer battery life. The MRF24WG module coordinates with the Access Point to go to sleep and wake up at periodic intervals to check for data messages, which the Access Point will buffer. The listenInterval in the Connection Algorithm defines the sleep interval. By default, PS Poll mode is disabled.

When PS Poll is enabled, the Wi-Fi Host Driver will automatically force the MRF24WG module to wake up each time the Host sends TX data or a control message to the module. When the Host message transaction is complete the MRF24WG driver will automatically re-enable PS Poll mode.

When the application is likely to experience a high volume of data traffic then PS Poll mode should be disabled for two reasons:

1. No power savings will be realized in the presence of heavy data traffic.
2. Performance will be impacted adversely as the Wi-Fi Host Driver continually activates and deactivates PS Poll mode via SPI messages.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_PS_POLL_CONTEXT context;

context.listenInterval = DRV_WIFI_DEFAULT_PS_LISTEN_INTERVAL;
context.dtimInterval = DRV_WIFI_DEFAULT_PS_DTIM_INTERVAL;
context.useDtim = DRV_WIFI_DEFAULT_PS_DTIM_ENABLED;

DRV_WIFI_PsPollEnable(&context);
```

Parameters

Parameters	Description
p_context	pointer to PS Poll context, see DRV_WIFI_PS_POLL_CONTEXT structure

Function

void DRV_WIFI_PsPollEnable(DRV_WIFI_PS_POLL_CONTEXT *p_context)

e) Wi-Fi Network Multicast Filter Configuration Functions

DRV_WIFI_MulticastFilterSet Function

Sets a multicast address filter using one of the software multicast filters.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_MulticastFilterSet(DRV_WIFI_MULTICAST_CONFIG * p_config);
```

Returns

None.

Description

This function allows the application to configure up to two Multicast Address Filters on the MRF24WG module. If two active multicast filters are set up they are ORed together - the module will receive and pass to the Host CPU received packets from either multicast address. The allowable values in p_config are:

filterId -- DRV_WIFI_MULTICAST_FILTER_1 through DRV_WIFI_MULTICAST_FILTER_16

action -- DRV_WIFI_MULTICAST_DISABLE_ALL (default) The Multicast Filter discards all received multicast messages - they will not be forwarded to the Host PIC. The remaining fields in this structure are ignored.

DRV_WIFI_MULTICAST_ENABLE_ALL The Multicast Filter forwards all received multicast messages to the Host PIC. The remaining fields in this structure are ignored.

DRV_WIFI_MULTICAST_USE_FILTERS The MAC filter will be used and the remaining fields in this structure configure which Multicast messages are forwarded to the Host PIC.

macBytes -- Array containing the MAC address to filter on (using the destination address of each incoming 802.11 frame). Specific bytes with the MAC address can be designated as "don't care" bytes. See macBitMask. This field in only used if action = DRV_WIFI_MULTICAST_USE_FILTERS.

macBitMask -- A byte where bits 5:0 correspond to macBytes[5:0]. If the bit is zero then the corresponding MAC byte must be an exact match for the frame to be forwarded to the Host PIC. If the bit is one then the corresponding MAC byte is a "don't care" and not used in the Multicast filtering process. This field in only used if action = DRV_WIFI_MULTICAST_USE_FILTERS.

Remarks

Cannot mix hardware and software multicast filters.

Preconditions

MRF24WG Wi-Fi initialization must be complete. DRV_WIFI_MultiCastFilterEnable() must have been called previously.

Example

```
DRV_WIFI_MULTICAST_CONFIG config;
uint8_t macMask[] = {01, 00, 5e, ff, ff, ff};  // (0xff(s) are the don't care bytes)

// configure software multicast filter 1 to filter multicast addresses that
// start with 01:00:5e
config.action = DRV_WIFI_MULTICAST_USE_FILTERS;
config->filterId = DRV_WIFI_MULTICAST_FILTER_1;
memcpy(config->macBytes, macMask, 6);
config->macBitMask = 0x38; // bits 5:3 = 1 (don't care on bytes 3,4,5)
                           // bits 2:0 = 0 (exact match required on bytes 0, 1, 2)
```

Function

void DRV_WIFI_MulticastFilterSet(DRV_WIFI_MULTICAST_CONFIG *p_config)

f) Wi-Fi Network Connection Functions

DRV_WIFI_Connect Function

Directs the MRF24WG to connect to a Wi-Fi network.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_Connect();
```

Returns

None.

Description

This function causes the MRF24WG to connect to a Wi-Fi network. Upon connection, or a failure to connect, an event will be generated.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete and relevant connection parameters must have been set.

Example

```
DRV_WIFI_Connect();
```

Function

void DRV_WIFI_Connect(void)

DRV_WIFI_Disconnect Function

Directs the MRF24WG to disconnect from a Wi-Fi network.

Implementation: Dynamic

File

drv_wifi.h

C

```
uint16_t DRV_WIFI_Disconnect();
```

Returns

DRV_WIFI_SUCCESS or DRV_WIFI_ERROR_DISCONNECT_FAILED.

Description

This function causes the MRF24WG to disconnect from a Wi-Fi network. No event is generated when a connection is terminated via the function call.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete and a connection must be in progress.

Example

```
DRV_WIFI_Disconnect();
```

Function

uint16_t DRV_WIFI_Disconnect(void)

g) Wi-Fi Network Configuration - Other Functions

DRV_WIFI_BssidSet Function

Sets the Basic Service Set Identifier (BSSID).

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_BssidSet(uint8_t * p_bssid);
```

Returns

None.

Description

This sets 6 byte (48-bit) MAC address of the Access Point that is being scanned for. It is optional to use this. Where it is useful is if there are two

APs with the same ID; the BSSID is used to connect to the specified AP. This setting can be used in lieu of the SSID. Set each byte to 0xFF (default) if the BSSID is not being used. Not typically needed.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t bssid[6];

bssid[0] = 0x00;
bssid[1] = 0xe8;
bssid[2] = 0xc0;
bssid[3] = 0x11;
bssid[4] = 0x22;
bssid[5] = 0x33;

DRV_WIFI_BssidSet(bssid);
```

Parameters

Parameters	Description
p_context	pointer to BSSID

Function

void DRV_WIFI_BssidSet(uint8_t *p_bssid)

DRV_WIFI_MacAddressSet Function

Uses a different MAC address for the MRF24WG.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_MacAddressSet(uint8_t * p_mac);
```

Returns

None.

Description

Directs the MRF24WG module to use the input MAC address instead of its factory-default MAC address. This function does not overwrite the factory default, which is in Flash memory.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete. Cannot be called when the MRF24WG is in a connected state.

Example

```
uint8_t mac[6] = {0x00, 0x1e, 0xc0, 0x11, 0x22, 0x33};

DRV_WIFI_MacAddressSet(mac);
```

Parameters

Parameters	Description
p_mac	pointer to 6 byte MAC that will be sent to MRF24WG

Function

void DRV_WIFI_MacAddressSet(uint8_t *p_mac)

DRV_WIFI_RtsThresholdSet Function

Sets the RTS Threshold.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_RtsThresholdSet(uint16_t rtsThreshold);
```

Returns

None.

Description

This function sets the RTS/CTS packet size threshold for when RTS/CTS frame will be sent. The default is 2347 bytes - the maximum for 802.11. It is recommended that the user leave the default at 2347 until they understand the performance and power ramifications of setting it smaller. Valid values are from 0 to DRV_WIFI_RTS_THRESHOLD_MAX (2347).

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_RtsThresholdSet(DRV_WIFI_RTS_THRESHOLD_MAX);
```

Parameters

Parameters	Description
rtsThreshold	Value of the packet size threshold

Function

void DRV_WIFI_RtsThresholdSet(uint16_t rtsThreshold)

DRV_WIFI_ScanContextSet Function

Sets the Wi-Fi scan context.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_ScanContextSet(DRV_WIFI_SCAN_CONTEXT * p_context);
```

Returns

None.

Description

This function sets the Wi-Fi scan context. The MRF24WG defaults are fine for most applications, but they can be changed by this function.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_SCAN_CONTEXT context;
```

```
context.scantype = DRV_WIFI_ACTIVE_SCAN;
context.scanCount      = 1;
context.minChannelTime = 200; // ms
context.maxChannelTime = 400; // ms
context.probeDelay     = 20; // uS

DRV_WIFI_ScanContextSet(&context);
```

Parameters

Parameters	Description
p_context	pointer to scan context, see DRV_WIFI_SCAN_CONTEXT structure.

Function

void DRV_WIFI_ScanContextSet(DRV_WIFI_SCAN_CONTEXT *p_context)

DRV_WIFI_TxModeSet Function

Configures 802.11 TX mode.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_TxModeSet(uint8_t mode);
```

Returns

None.

Description

This function sets the MRF24WG TX mode.

mode	Description
DRV_WIFI_TXMODE_G_RATES	Use all 802.11g rates (default)
DRV_WIFI_TXMODE_B_RATES	Use only 802.11b rates
DRV_WIFI_TXMODE_LEGACY_RATES	Use only 1 and 2 Mbps rates

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_TxModeSet(DRV_WIFI_TXMODE_G_RATES);
```

Parameters

Parameters	Description
mode	TX mode value (see description)

Function

void DRV_WIFI_TxModeSet(uint8_t mode)

DRV_WIFI_RssiSet Function

Sets RSSI restrictions when connecting.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_RssiSet(uint8_t rssi);
```

Returns

None.

Description

This setting is only used if: 1) Neither an SSID or BSSID has been configured or 2) An SSID is defined and multiple APs are discovered with the same SSID.

rssi	Description
0	Connect to first network found
1-254	Only connect to network if the RSSI is greater or equal to this value
255	Connect to the highest RSSI found (default)

Remarks

Rarely needed.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_RssiSet(255);
```

Parameters

Parameters	Description
rssi	see description

Function

void DRV_WIFI_RssiSet(uint8_t rssi)

DRV_WIFI_RSSI_Cache_FromRxDataRead Function

Caches RSSI value from RX data packet.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_RSSI_Cache_FromRxDataRead(uint16_t rssi);
```

Returns

None.

Description

This function caches RSSI value from RX data packet.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Parameters

Parameters	Description
rssi	RSSI value read from RX data packet

Function

void DRV_WIFI_RSSI_Cache_FromRxDataRead(uint16_t rssi)

DRV_WIFI_RSSI_Get_FromRxDataRead Function

Reads RSSI value from RX data packet.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_RSSI_Get_FromRxDataRead(uint16_t * mean, uint16_t * last);
```

Returns

- mean - the calculated mean RSSI
- last - the total count of RSSI values

Description

This function reads RSSI value from RX data packet.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Parameters

Parameters	Description
mean	pointer to where the average mean RSSI value to be stored
last	pointer to where the total count of RSSI values to be stored

Function

void DRV_WIFI_RSSI_Get_FromRxDataRead(uint16_t *mean, uint16_t *last)

DRV_WIFI_INT_Handle Function

MRF24WG Wi-Fi driver interrupt handle.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_INT_Handle();
```

Returns

None.

Description

This function is the interrupt handle of MRF24WG Wi-Fi driver.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Function

void DRV_WIFI_INT_Handle(void)

DRV_WIFI_SpiClose Function

Closes SPI object for MRF24WG Wi-Fi driver.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_SpiClose();
```

Returns

None.

Description

This function closes SPI object for MRF24WG Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void DRV_WIFI_SpiClose(void)

DRV_WIFI_TaskSyncInit Function

Initializes RTOS Semaphore and Mutex for MRF24WG Wi-Fi driver.

Implementation: Dynamic

File

drv_wifi.h

C

```
bool DRV_WIFI_TaskSyncInit();
```

Returns

- true - Initialization was successful
- false - Initialization was not successful

Description

This function initializes RTOS Semaphore and Mutex for MRF24WG Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

bool DRV_WIFI_TaskSyncInit(void)

h) Wi-Fi Scanning Functions

DRV_WIFI_Scan Function

Commands the MRF24WG module to start a scan operation. This will generate the WF_EVENT_SCAN_RESULTS_READY event.

Implementation: Dynamic

File

drv_wifi.h

C

```
uint16_t DRV_WIFI_Scan(bool scanAll);
```

Returns

None.

Description

This function directs the MRF24WG module to initiate a scan operation. The Host Application will be notified that the scan results are ready when it receives the WF_EVENT_SCAN_RESULTS_READY event. The eventInfo field for this event will contain the number of scan results. Once the scan results are ready they can be retrieved with DRV_WIFI_ScanResultGet().

Scan results are retained on the MRF24WG until:

1. Calling DRV_WIFI_Scan() again (after scan results returned from previous call).
2. MRF24WG reset.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_Scan(true);
```

Parameters

Parameters	Description
scanAll	If false: • If SSID defined then only scan results with that SSID are retained. • If SSID not defined then all scanned SSIDs will be retained. • Only scan results from Infrastructure or Ad-Hoc networks are retained. • The channel list that is scanned will be determined from the channels passed in via DRV_WIFI_ChannelListSet().
If true	• All scan results are retained (both Infrastructure and Ad-Hoc networks). • All channels within the MRF24WG's regional domain will be scanned.

Function

uint16_t DRV_WIFI_Scan(bool scanAll)

DRV_WIFI_ScanContextGet Function

Gets the Wi-Fi scan context.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_ScanContextGet(DRV_WIFI_SCAN_CONTEXT * p_context);
```

Returns

None.

Description

This function gets the Wi-Fi scan context.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_SCAN_CONTEXT context;

DRV_WIFI_ScanContextSet(&context);
```

Parameters

Parameters	Description
p_context	pointer to where scan context will be written, see DRV_WIFI_SCAN_CONTEXT structure

Function

void DRV_WIFI_ScanContextGet(DRV_WIFI_SCAN_CONTEXT *p_context)

i) Wi-Fi Event Processing Functions

DRV_WIFI_ProcessEvent Function

Processes Wi-Fi event.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_ProcessEvent(uint16_t event, uint16_t eventInfo);
```

Returns

None.

Description

This function is called to process a Wi-Fi event.

Remarks

None.

Preconditions

TCPIP stack should be initialized.

Parameters

Parameters	Description
event	event code
eventInfo	additional information about the event; not all events have associated info, in which case this value will be set to DRV_WIFI_NO_ADDITIONAL_INFO (0xff)

Function

void DRV_WIFI_ProcessEvent(uint16_t event, uint16_t eventInfo)

DRV_WIFI_SoftApEventInfoGet Function

Gets the stored Soft AP event info.

Implementation: Dynamic

File

drv_wifi.h

C

```
DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT * DRV_WIFI_SoftApEventInfoGet();
```

Returns

None.

Description

This function retrieves the additional event info after a Soft AP event has occurred.

Remarks

None.

Preconditions

Soft AP event must have occurred.

Example

```
DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT info;

DRV_WIFI_WPSCredentialsGet(&info);
```

Parameters

Parameters	Description
p_event	pointer to where event info is written, see DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT structure

Function

DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT *DRV_WIFI_SoftApEventInfoGet(void)

DRV_WIFI_SoftAPContextSet Function

Sets the Soft AP context.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_SoftAPContextSet(DRV_WIFI_SOFTAP_NETWORK_CONTEXT * p_context);
```

Returns

None.

Description

None.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_SOFTAP_NETWORK_CONTEXT SoftAPContext;

//SoftAPContext.mode = DRV_WIFI_ADHOC_CONNECT_THEN_START;
SoftAPContext.hiddenSsid = false;
//SoftAPContext.beaconPeriod = DRV_WIFI_DEFAULT_ADHOC_BEACON_PERIOD;

DRV_WIFI_SoftAPContextSet(&SoftAPContext);
```

Parameters

Parameters	Description
p_context	pointer to Soft AP context data; see definition for the DRV_WIFI_SOFTAP_NETWORK_CONTEXT structure

Function

void DRV_WIFI_SoftAPContextSet(DRV_WIFI_SOFTAP_NETWORK_CONTEXT *p_context)

j) Wi-Fi Data Configuration Functions

DRV_WIFI_ConfigDataLoad Function

Loads configuration data from the board EEPROM.

Implementation: Dynamic

File

drv_wifi.h

C

```
bool DRV_WIFI_ConfigDataLoad();
```

Returns

- true - Configuration data was loaded
- false - Configuration data was not loaded

Description

This function loads configuration data from the board EEPROM. If not present or corrupted then default values will be used.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

bool DRV_WIFI_ConfigDataLoad(void)

DRV_WIFI_ConfigDataSave Function

Save configuration data to the board EEPROM.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_ConfigDataSave();
```

Returns

None.

Description

This function saves configuration data to the board EEPROM.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void DRV_WIFI_ConfigDataSave(void)

DRV_WIFI_ConfigDataDelete Function

Erases configuration data from the board EEPROM.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_ConfigDataDelete();
```

Returns

None.

Description

This function erases configuration data from the board EEPROM.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void DRV_WIFI_ConfigDataDelete(void)

k) Wi-Fi Status Functions

DRV_WIFI_BssidGet Function

Gets the BSSID set in DRV_WIFI_BssidSet().

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_BssidGet(uint8_t * p_bssid);
```

Returns

None.

Description

Retrieves the BSSID set in the previous call to DRV_WIFI_BssidSet().

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t bssid[6];

DRV_WIFI_BssidGet(bssid);
```

Parameters

Parameters	Description
p_context	pointer to where BSSID will be written

Function

void DRV_WIFI_BssidGet(uint8_t *p_bssid)

DRV_WIFI_ChannelListGet Function

Gets the channel list.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_ChannelListGet(uint8_t * p_channelList, uint8_t * p_numChannels);
```

Returns

None.

Description

This function gets the current channel list.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t channelList[DRV_WIFI_MAX_CHANNEL_LIST_LENGTH];
uint8_t numChannels;

DRV_WIFI_ChannelListGet(channelList, &numChannels);
```

Parameters

Parameters	Description
p_channelList	pointer to where channel list will be written
numChannels	pointer to where number of channels in the list will be written

Function

void RV_WIFI_ChannelListGet(uint8_t *p_channelList, uint8_t *p_numChannels)

DRV_WIFI_ConnectContextGet Function

Gets the current Wi-Fi connection context.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_ConnectContextGet(DRV_WIFI_CONNECTION_CONTEXT * p_ctx);
```

Returns

None.

Description

This function gets the current Wi-Fi connection context.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_CONNECTION_CONTEXT ctx;

DRV_WIFI_ConnectContextGet(&ctx);
```

Parameters

Parameters	Description
p_ctx	pointer to where connection context is written, see DRV_WIFI_CONNECTION_CONTEXT structure

Function

void DRV_WIFI_ConnectContextGet(DRV_WIFI_CONNECTION_CONTEXT *p_ctx)

DRV_WIFI_ConnectionStateGet Function

Gets the current Wi-Fi connection state.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_ConnectionStateGet(uint8_t * p_state);
```

Returns

None

Description

This function gets the current Wi-Fi connection state.

*p_state	Description
DRV_WIFI_CSTATE_NOT_CONNECTED	No Wi-Fi connection exists
DRV_WIFI_CSTATE_CONNECTION_IN_PROGRESS	Wi-Fi connection in progress
DRV_WIFI_CSTATE_CONNECTED_INFRASTRUCTURE	Wi-Fi connected in infrastructure mode
DRV_WIFI_CSTATE_CONNECTED_ADHOC	Wi-Fi connected in Ad-Hoc mode
DRV_WIFI_CSTATE_RECONNECTION_IN_PROGRESS	Wi-Fi in process of reconnecting
DRV_WIFI_CSTATE_CONNECTION_PERMANENTLY_LOST	Wi-Fi connection permanently lost

Remarks

None

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t state;

DRV_WIFI_ConnectionStateGet(&state);
```

Parameters

Parameters	Description
p_state	pointer to where state is written (see description)

Function

void DRV_WIFI_ConnectionStateGet(uint8_t *p_state)

DRV_WIFI_DeviceInfoGet Function

Retrieves MRF24WG device information.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_DeviceInfoGet(DRV_WIFI_DEVICE_INFO * p_deviceInfo);
```

Returns

None.

Description

This function retrieves MRF24WG device information. See DRV_WIFI_DEVICE_INFO structure.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_DEVICE_INFO info;

DRV_WIFI_DeviceInfoGet(&info);
```

Parameters

Parameters	Description
p_deviceInfo	pointer where device info will be written

Function

void DRV_WIFI_DeviceInfoGet(DRV_WIFI_DEVICE_INFO *p_deviceInfo)

DRV_WIFI_MacAddressGet Function

Retrieves the MRF24WG MAC address.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_MacAddressGet(uint8_t * p_mac);
```

Returns

None.

Description

This function retrieves the MRF24WG MAC address.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t mac[6];

DRV_WIFI_MacAddressGet(mac);
```

Parameters

Parameters	Description
p_mac	pointer where mac address will be written (must point to a 6 byte buffer)

Function

void DRV_WIFI_MacAddressGet(uint8_t *p_mac)

DRV_WIFI_MacStatsGet Function

Gets MAC statistics.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_MacStatsGet(DRV_WIFI_MAC_STATS * p_macStats);
```

Returns

None.

Description

This function gets the various MAC layer stats as maintained by the MRF24WG.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_MAC_STATS macStats;

DRV_WIFI_MacStatsGet(&macStats);
```

Parameters

Parameters	Description
p_macStats	pointer to where MAC statistics are written, see DRV_WIFI_MAC_STATS structure.

Function

void DRV_WIFI_MacStatsGet(DRV_WIFI_MAC_STATS *p_macStats)

DRV_WIFI_NetworkTypeGet Function

Gets the Wi-Fi network type.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_NetworkTypeGet(uint8_t * p_networkType);
```

Returns

None.

Description

This function gets the Wi-Fi network type.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t networkType;

DRV_WIFI_NetworkTypeGet(&networkType);
```

Parameters

Parameters	Description
p_networkType	pointer to where the network type will be written. One of the following: DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE DRV_WIFI_NETWORK_TYPE_ADHOC DRV_WIFI_NETWORK_TYPE_SOFT_AP

Function

void DRV_WIFI_NetworkTypeGet(uint8_t *p_networkType)

DRV_WIFI_PowerSaveStateGet Function

Gets the current power-saving state.

File

drv_wifi.h

C

```
void DRV_WIFI_PowerSaveStateGet(uint8_t * p_powerSaveState);
```

Returns

None.

Description

This function gets the current MRF24WG power-saving state.

powerSaveState	Definition
DRV_WIFI_PS_HIBERNATE	MRF24WG in hibernate state
DRV_WIFI_PS_PS_POLL_DTIM_ENABLED	MRF24WG in PS-Poll mode with DTIM enabled
DRV_WIFI_PS_PS_POLL_DTIM_DISABLED	MRF24WG in PS-Poll mode with DTIM disabled
DRV_WIFI_PS_POLL_OFF	MRF24WG is not in any power-saving state

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t state;
```

```
DRV_WIFI_PowerSaveStateGet(&state);
```

Parameters

Parameters	Description
p_powerSaveState	pointer to where power state is written (see description)

Function

void DRV_WIFI_PowerSaveStateGet(uint8_t *p_powerSaveState)

DRV_WIFI_ReconnectModeGet Function

Gets the Wi-Fi reconnection mode.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_ReconnectModeGet(uint8_t * p_retryCount, uint8_t * p_deauthAction, uint8_t * p_beaconTimeout,
uint8_t * p_beaconTimeoutAction);
```

Returns

None.

Description

This function gets the reconnection mode parameters set in DRV_WIFI_ReconnectModeGet.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t retryCount, deauthAction, beaconTimeout, beaconTimeoutAction;

DRV_WIFI_ReconnectModeGet(&retryCount,
                          &deauthAction,
                          &beaconTimeout,
                          &beaconTimeoutAction);
```

Parameters

Parameters	Description
p_retryCount	pointer where retry count is written
p_deauthAction	pointer where deauthorization action is written
p_beaconTimeout	pointer where beacon timeout is written
p_beaconTimeoutAction	pointer where beacon timeout action is written

Function

void DRV_WIFI_ReconnectModeGet(uint8_t *p_retryCount, uint8_t *p_deauthAction,
uint8_t *p_beaconTimeout, uint8_t *p_beaconTimeoutAction)

DRV_WIFI_RegionalDomainGet Function

Retrieves the MRF24WG Regional domain.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_RegionalDomainGet(uint8_t * p_regionalDomain);
```

Returns

None.

Description

This function returns the regional domain on the MRF24WG. Values are:

- DRV_WIFI_DOMAIN_FCC
- DRV_WIFI_DOMAIN_ETSI
- DRV_WIFI_DOMAIN_JAPAN
- DRV_WIFI_DOMAIN_OTHER

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t domain;

DRV_WIFI_RegionalDomainGet(&domain);
```

Parameters

Parameters	Description
p_regionalDomain	pointer where the regional domain value will be written

Function

void DRV_WIFI_RegionalDomainGet(uint8_t *p_regionalDomain)

DRV_WIFI_RtsThresholdGet Function

Gets the RTS Threshold.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_RtsThresholdGet(uint16_t * p_rtsThreshold);
```

Returns

None.

Description

This function returns the RTS/CTS packet size threshold.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint16_t threshold;

DRV_WIFI_RtsThresholdGet(&threshold);
```

Parameters

Parameters	Description
p_rtsThreshold	pointer to where RTS threshold is written

Function

void DRV_WIFI_RtsThresholdGet(uint16_t *p_rtsThreshold)

DRV_WIFI_SecurityGet Function

Gets the current Wi-Fi security setting.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_SecurityGet(uint8_t * p_securityType, uint8_t * p_securityKey, uint8_t * p_securityKeyLength);
```

Returns

None.

Description

This function gets the current Wi-Fi security setting.

'securityType' Field	Key	Length
DRV_WIFI_SECURITY_OPEN	N/A	N/A
DRV_WIFI_SECURITY_WEP_40	binary	4 keys, 5 bytes each (total of 20 bytes)
DRV_WIFI_SECURITY_WEP_104	binary	4 keys, 13 bytes each (total of 52 bytes)
DRV_WIFI_SECURITY_WPA_WITH_KEY	binary	32 bytes
DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE	ascii	8-63 ascii characters
DRV_WIFI_SECURITY_WPA2_WITH_KEY	binary	32 bytes
DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE	ascii	8-63 ascii characters
DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY	binary	32 bytes
DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE	ascii	8-63 ascii characters

Remarks

If security was initially set with a passphrase that the MRF24WG used to generate a binary key, this function returns the binary key, not the passphrase.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t securityType;
uint8_t securityKey[DRV_WIFI_MAX_SECURITY_KEY_LENGTH];
uint8_t keyLength;

DRV_WIFI_SecurityGet(&securityType, securityKey, &keyLength);
```

Parameters

Parameters	Description
p_securityType	value corresponding to the security type desired (see description)
p_securityKey	binary key or passphrase (not used if security is DRV_WIFI_SECURITY_OPEN)
p_securityKeyLength	number of bytes in p_securityKey (not used if security is DRV_WIFI_SECURITY_OPEN)

Function

void DRV_WIFI_SecurityGet(uint8_t *p_securityType,
uint8_t *p_securityKey,
uint8_t *p_securityKeyLength)

DRV_WIFI_SsidGet Function

Gets the SSID.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_SsidGet(uint8_t * p_ssid, uint8_t * p_ssidLength);
```

Returns

None.

Description

Gets the SSID and SSID Length.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t ssid[DRV_WIFI_MAX_SSID_LENGTH];
uint8_t ssidLength;

DRV_WIFI_SsidGet(ssid, &ssidLength);
```

Parameters

Parameters	Description
p_ssid	pointer to buffer where SSID will be written
ssidLength	number of bytes in SSID

Function

void DRV_WIFI_SsidGet(uint8_t *p_ssid, uint8_t *p_ssidLength)

DRV_WIFI_TxModeGet Function

Gets 802.11 TX mode.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_TxModeGet(uint8_t * p_mode);
```

Returns

None.

Description

This function gets the MRF24WG TX mode.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t mode;

DRV_WIFI_TxModeGet(&mode);
```

Function

void DRV_WIFI_TxModeGet(uint8_t *p_mode)

DRV_WIFI_WepKeyTypeGet Function

Gets the WEP Key type.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_WepKeyTypeGet(uint8_t * p_wepKeyType);
```

Returns

None.

Description

This function gets the WEP key type:

- DRV_WIFI_SECURITY_WEP_SHAREDKEY
- DRV_WIFI_SECURITY_WEP_OPENKEY

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t wepKeyType;

DRV_WIFI_WepKeyTypeGet(&wepKeyType);
```

Function

void DRV_WIFI_WepKeyTypeGet(uint8_t *p_keyType)

DRV_WIFI_WPSCredentialsGet Function

Gets the WPS credentials.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_WPSCredentialsGet(DRV_WIFI_WPS_CREDENTIAL * p_cred);
```

Returns

None.

Description

This function gets the WPS credentials from the MRF24WG.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_WPS_CREDENTIAL cred;

DRV_WIFI_WPSCredentialsGet(&cred);
```

Function

void DRV_WIFI_WPSCredentialsGet(DRV_WIFI_WPS_CREDENTIAL *p_cred)

DRV_WIFI_RssiGet Function

Gets RSSI value set in DRV_WIFI_RssiSet().

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_RssiGet(uint8_t * p_rssi);
```

Returns

None.

Description

This function retrieves the value set in Gets RSSI value set in DRV_WIFI_RssiSet(). It does not retrieve the current connection RSSI value. The scan result will yield the current RSSI.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t rssi;

DRV_WIFI_RssiGet(&rssi);
```

Parameters

Parameters	Description
p_rssi	pointer where rssi value is written

Function

void DRV_WIFI_RssiGet(uint8_t *p_rssi)

DRV_WIFI_SecurityTypeGet Function

Gets the current Wi-Fi security type.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_SecurityTypeGet(uint8_t * p_securityType);
```

Returns

None.

Description

This function gets the current Wi-Fi security type.

'securityType' Field
DRV_WIFI_SECURITY_OPEN
DRV_WIFI_SECURITY_WEP_40
DRV_WIFI_SECURITY_WEP_104
DRV_WIFI_SECURITY_WPA_WITH_KEY
DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE
DRV_WIFI_SECURITY_WPA2_WITH_KEY
DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE
DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY
DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t securityType;

DRV_WIFI_SecurityTypeGet(&securityType);
```

Parameters

Parameters	Description
p_securityType	value corresponding to the security type desired (see description)

Function

void DRV_WIFI_SecurityTypeGet(uint8_t *p_securityType)

DRV_WIFI_TxPowerFactoryMaxGet Function

Retrieves the factory-set max TX power from the MRF24WG module.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_TxPowerFactoryMaxGet(uint8_t * p_factoryMaxTxPower);
```

Returns

None.

Description

This function retrieves the factory-set max TX power from the MRF24WG.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t maxPower;

DRV_WIFI_TxPowerFactoryMaxGet(&maxPower);
```

Parameters

Parameters	Description
p_factoryMaxTxPower	pointer to where factory max power is written (dbM)

Function

void DRV_WIFI_TxPowerFactoryMaxGet(int8_t *p_factoryMaxTxPower)

DRV_WIFI_TxPowerMaxGet Function

Gets the TX max power on the MRF24WG0M.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_TxPowerMaxGet(uint8_t * p_maxTxPower);
```

Returns

None.

Description

Gets the TX max power setting from the MRF24WG.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
uint8_t maxPower;

DRV_WIFI_TxPowerMaxGet(&maxPower);
```

Parameters

Parameters	Description
p_maxTxPower	pointer to where max power setting is written (dBm)

Function

void DRV_WIFI_TxPowerMaxGet(uint8_t *p_maxTxPower)

DRV_WIFI_TxPowerMaxSet Function

Sets the TX max power on the MRF24WG0M.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_TxPowerMaxSet(uint8_t maxTxPower);
```

Returns

None.

Description

After initialization the MRF24WG0M max TX power is determined by a factory-set value. This function can set a different maximum TX power levels. However, this function can never set a maximum TX power greater than the factory-set value, which can be read via DRV_WIFI_TxPowerFactoryMaxGet.

Remarks

No conversion of units needed, input to MRF24WG0M is in dBm.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
DRV_WIFI_TxPowerMaxSet(8); // set max TX power to 8dBm
```

Parameters

Parameters	Description
maxTxPower	valid range (0 to 17 dBm)

Function

void DRV_WIFI_TxPowerMaxSet(uint8_t maxTxPower)

DRV_WIFI_ScanResultGet Function

Read selected scan results back from MRF24WG.

Implementation: Dynamic

File

drv_wifi.h

C

```
void DRV_WIFI_ScanResultGet(uint8_t listIndex, DRV_WIFI_SCAN_RESULT * p_scanResult);
```

Returns

None.

Description

After a scan has completed this function is used to read one scan result at a time from the MRF24WG.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete. WF_EVENT_SCAN_RESULTS_READY event must have already occurred.

Example

```
DRV_WIFI_SCAN_RESULT scanResult;

DRV_WIFI_ScanResultGet(0, &scanResult); // get first scan result in list
```

Parameters

Parameters	Description
listIndex	index (0 based list) of the scan entry to retrieve
p_scanResult	pointer to where scan result is written, see DRV_WIFI_SCAN_RESULT structure

Function

void DRV_WIFI_ScanResultGet(uint8_t listIndex, t_wfScanResult *p_scanResult)

DRV_WIFI_InHibernateMode Function

Checks if MRF24WG is in hibernate mode.

Implementation: Dynamic

File

drv_wifi.h

C

```
bool DRV_WIFI_InHibernateMode();
```

Returns

- true - MRF24WG is in hibernate mode
- false - MRF24WG is not in hibernate mode

Description

This function checks if the MRF24WG is in hibernate mode.

Remarks

None.

Preconditions

MRF24WG Wi-Fi initialization must be complete.

Example

```
bool flag;

flag = DRV_WIFI_InHibernateMode();
```

Function

bool DRV_WIFI_InHibernateMode(void)

l) Wi-Fi SPI Functions

DRV_WIFI_SpiDmaRx Function

SPI RX API using DMA.

Implementation: Dynamic

File

drv_wifi.h

C

```
bool DRV_WIFI_SpiDmaRx(unsigned char * buf, uint32_t size);
```

Returns

- true - Data was received
- false - Data was not received

Description

This function receives data from the module using DMA over SPI bus.

Remarks

None.

Preconditions

TCP/IP stack should be initialized.

Parameters

Parameters	Description
buf	buffer pointer to the data to be received
size	the data size

Function

DRV_WIFI_SpiDmaRx(unsigned char *buf, uint32_t size)

DRV_WIFI_SpiDmaTx Function

SPI TX API using DMA.

Implementation: Dynamic

File

drv_wifi.h

C

```
bool DRV_WIFI_SpiDmaTx(unsigned char * buf, uint32_t size);
```

Returns

- true - Data was sent
- false - Data was not sent

Description

This function sends data to the module using DMA over SPI bus.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Parameters

Parameters	Description
buf	buffer pointer to the data to be sent
size	the data size

Function

bool DRV_WIFI_SpiDmaTx(unsigned char *buf, uint32_t size)

DRV_WIFI_SpiInit Function

Initializes SPI object for MRF24WG Wi-Fi driver.

Implementation: Dynamic

File

drv_wifi.h

C

```
bool DRV_WIFI_SpiInit(bool (*rx)(unsigned char *buf, uint32_t size), bool (*tx)(unsigned char *buf,
uint32_t size));
```

Returns

- true - The SPI object was initialized
- false - The SPI object was not initialized

Description

This function initializes SPI object for MRF24WG Wi-Fi driver.

Remarks

None.

Preconditions

TCP/IP stack should be initialized.

Parameters

Parameters	Description
rx	function pointer to RX API
tx	function pointer to TX API

Function

bool DRV_WIFI_SpiInit(bool (*rx)(unsigned char *buf, uint32_t size),
bool (*tx)(unsigned char *buf, uint32_t size))

DRV_WIFI_SpiRx Function

SPI Rx API.

Implementation: Dynamic

File

drv_wifi.h

C

```
bool DRV_WIFI_SpiRx(unsigned char * buf, uint32_t size);
```

Returns

- true - Data was received
- false - Data was not received

Description

This function receives data from the module over SPI bus.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Parameters

Parameters	Description
buf	buffer pointer to the data to be received
size	the data size

Function

DRV_WIFI_SpiRx(unsigned char *buf, uint32_t size)

DRV_WIFI_SpiTx Function

SPI TX API

Implementation: Dynamic

File

drv_wifi.h

C

```
bool DRV_WIFI_SpiTx(unsigned char * buf, uint32_t size);
```

Returns

- true - Data was sent

- false - Data was not sent

Description

This function sends data to the module over SPI bus.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Parameters

Parameters	Description
buf	buffer pointer to the data to be sent
size	the data size

Function

bool DRV_WIFI_SpiTx(unsigned char *buf, uint32_t size)

m) Private Configuration Functions

iwpriv_adhocctx_set Function

Sets the Ad-Hoc network context information.

Implementation: Dynamic

File

drv_wifi_iwpriv.h

C

```
void iwpriv_adhocctx_set(DRV_WIFI_ADHOC_NETWORK_CONTEXT * p_context);
```

Returns

None.

Description

This function sets the current Ad-Hoc network context information by reading from a passed pointer.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
p_context	pointer to where the Ad-Hoc network context is stored

Function

void iwpriv_adhocctx_set(DRV_WIFI_ADHOC_NETWORK_CONTEXT *p_context)

iwpriv_config_read Function

Reads the Wi-Fi context configuration.

Implementation: Dynamic

File

drv_wifi_iwpriv.h

C

```
void iwpriv_config_read(DRV_WIFI_CONFIG_DATA * wifi_config);
```

Returns

None.

Description

This function reads the current Wi-Fi context configuration, copies and stores the whole structure to the pointer passed to the function.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
wifi_config	pointer to where the context configuration is written

Function

void iwpriv_config_read(DRV_WIFI_CONFIG_DATA *wifi_config)

n) Data Types and Constants

DRV_WIFI_BSSID_LENGTH Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_BSSID_LENGTH (6)
```

Description

This is macro DRV_WIFI_BSSID_LENGTH.

DRV_WIFI_DEAUTH_REASONCODE_MASK Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_DEAUTH_REASONCODE_MASK ((uint8_t)0x80)
```

Description

This is macro DRV_WIFI_DEAUTH_REASONCODE_MASK.

DRV_WIFI_DEFAULT_ADHOC_BEACON_PERIOD Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_DEFAULT_ADHOC_BEACON_PERIOD (100)   // ms
```

Description

ms

DRV_WIFI_DEFAULT_ADHOC_HIDDEN_SSID Macro

Default values for Wi-Fi Ad-Hoc settings.

File

drv_wifi.h

C

```
#define DRV_WIFI_DEFAULT_ADHOC_HIDDEN_SSID false
```

Description

Wi-Fi Ad-Hoc default settings.

These defines identify various default Wi-Fi Ad-Hoc settings that can be used in the DRV_WIFI_ADHOC_NETWORK_CONTEXT structure.

DRV_WIFI_DEFAULT_ADHOC_MODE Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_DEFAULT_ADHOC_MODE DRV_WIFI_ADHOC_CONNECT_THEN_START
```

Description

This is macro DRV_WIFI_DEFAULT_ADHOC_MODE.

DRV_WIFI_DEFAULT_PS_DTIM_ENABLED Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_DEFAULT_PS_DTIM_ENABLED true            // DTIM wake-up enabled (normally the case)
```

Description

DTIM wake-up enabled (normally the case)

DRV_WIFI_DEFAULT_PS_DTIM_INTERVAL Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_DEFAULT_PS_DTIM_INTERVAL ((uint16_t)2)   // number of beacon periods
```

Description

number of beacon periods

DRV_WIFI_DEFAULT_PS_LISTEN_INTERVAL Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_DEFAULT_PS_LISTEN_INTERVAL ((uint16_t)1)   // 100 ms multiplier, e.g., 1 * 100 ms = 100 ms
```

Description

100 ms multiplier, e.g., 1 * 100 ms = 100 ms

DRV_WIFI_DEFAULT_SCAN_COUNT Macro

Default values for Wi-Fi scan context

File

drv_wifi.h

C

```
#define DRV_WIFI_DEFAULT_SCAN_COUNT (1)
```

Description

Wi-Fi Scan Context default settings

These defines identify the default Wi-Fi scan context values that can be used in the DRV_WIFI_SCAN_CONTEXT structure.

DRV_WIFI_DEFAULT_SCAN_MAX_CHANNEL_TIME Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_DEFAULT_SCAN_MAX_CHANNEL_TIME (400) // ms
```

Description

ms

DRV_WIFI_DEFAULT_SCAN_MIN_CHANNEL_TIME Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_DEFAULT_SCAN_MIN_CHANNEL_TIME (200) // ms
```

Description

ms

DRV_WIFI_DEFAULT_SCAN_PROBE_DELAY Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_DEFAULT_SCAN_PROBE_DELAY (20)  // us
```

Description

us

DRV_WIFI_DEFAULT_WEP_KEY_TYPE Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_DEFAULT_WEP_KEY_TYPE DRV_WIFI_SECURITY_WEP_OPENKEY
```

Description

This is macro DRV_WIFI_DEFAULT_WEP_KEY_TYPE.

DRV_WIFI_DISABLED Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_DISABLED (0)
```

Description

This is macro DRV_WIFI_DISABLED.

DRV_WIFI_DISASSOC_REASONCODE_MASK Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_DISASSOC_REASONCODE_MASK ((uint8_t)0x40)
```

Description

This is macro DRV_WIFI_DISASSOC_REASONCODE_MASK.

DRV_WIFI_ENABLED Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_ENABLED (1)
```

Description

Do not make this an enumerated type!

DRV_WIFI_MAX_CHANNEL_LIST_LENGTH Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_MAX_CHANNEL_LIST_LENGTH (14)
```

Description

This is macro DRV_WIFI_MAX_CHANNEL_LIST_LENGTH.

DRV_WIFI_MAX_NUM_RATES Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_MAX_NUM_RATES (8)
```

Description

This is macro DRV_WIFI_MAX_NUM_RATES.

DRV_WIFI_MAX_SECURITY_KEY_LENGTH Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_MAX_SECURITY_KEY_LENGTH (64)
```

Description

This is macro DRV_WIFI_MAX_SECURITY_KEY_LENGTH.

DRV_WIFI_MAX_SSID_LENGTH Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_MAX_SSID_LENGTH (32)
```

Description

This is macro DRV_WIFI_MAX_SSID_LENGTH.

DRV_WIFI_MAX_WEP_KEY_LENGTH Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_MAX_WEP_KEY_LENGTH (DRV_WIFI_WEP104_KEY_LENGTH)
```

Description

This is macro DRV_WIFI_MAX_WEP_KEY_LENGTH.

DRV_WIFI_MAX_WPA_PASS_PHRASE_LENGTH Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_MAX_WPA_PASS_PHRASE_LENGTH (63)    // must exclude string terminator
```

Description

must exclude string terminator

DRV_WIFI_MIN_WPA_PASS_PHRASE_LENGTH Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_MIN_WPA_PASS_PHRASE_LENGTH (8)     // must exclude string terminator
```

Description

must exclude string terminator

DRV_WIFI_NETWORK_TYPE_ADHOC Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_NETWORK_TYPE_ADHOC (2)
```

Description

This is macro DRV_WIFI_NETWORK_TYPE_ADHOC.

DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE Macro

Selection of different Wi-Fi network types.

File

drv_wifi.h

C

```
#define DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE (1)
```

Description

Wi-Fi Network Types

This enumeration identifies the Wi-Fi network types that can be selected. Do NOT make these an enumerated type as they are used as a compile switch.

DRV_WIFI_NETWORK_TYPE_SOFT_AP Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_NETWORK_TYPE_SOFT_AP (4)
```

Description

This is macro DRV_WIFI_NETWORK_TYPE_SOFT_AP.

DRV_WIFI_NO_ADDITIONAL_INFO Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_NO_ADDITIONAL_INFO ((uint16_t)0xffff)
```

Description

eventInfo define for DRV_WIFI_ProcessEvent() when no additional info is supplied

DRV_WIFI_RETRY_ADHOC Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_RETRY_ADHOC (3)
```

Description

This is macro DRV_WIFI_RETRY_ADHOC.

DRV_WIFI_RETRY_FOREVER Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_RETRY_FOREVER (255)
```

Description

This is macro DRV_WIFI_RETRY_FOREVER.

DRV_WIFI_RTS_THRESHOLD_MAX Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_RTS_THRESHOLD_MAX (2347)  /* maximum RTS threshold size in bytes */
```

Description

maximum RTS threshold size in bytes

DRV_WIFI_SECURITY_OPEN Macro

Selection of different Wi-Fi security types

File

drv_wifi.h

C

```
#define DRV_WIFI_SECURITY_OPEN (0)
```

Description

Wi-Fi Security Types

This enumeration identifies the Wi-Fi security types that can be selected. Do NOT make these an enumerated type as they are used as a compile switch.

DRV_WIFI_SECURITY_WEP_104 Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_SECURITY_WEP_104 (2)
```

Description

This is macro DRV_WIFI_SECURITY_WEP_104.

DRV_WIFI_SECURITY_WEP_40 Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_SECURITY_WEP_40 (1)
```

Description

This is macro DRV_WIFI_SECURITY_WEP_40.

DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY (7)
```

Description

This is macro DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY.

DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE (8)
```

Description

This is macro DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE.

DRV_WIFI_SECURITY_WPA_WITH_KEY Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_SECURITY_WPA_WITH_KEY (3)
```

Description

This is macro DRV_WIFI_SECURITY_WPA_WITH_KEY.

DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE (4)
```

Description

This is macro DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE.

DRV_WIFI_SECURITY_WPA2_WITH_KEY Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_SECURITY_WPA2_WITH_KEY (5)
```

Description

This is macro DRV_WIFI_SECURITY_WPA2_WITH_KEY.

DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE (6)
```

Description

This is macro DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE.

DRV_WIFI_SECURITY_WPS_PIN Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_SECURITY_WPS_PIN (10)
```

Description

This is macro DRV_WIFI_SECURITY_WPS_PIN.

DRV_WIFI_SECURITY_WPS_PUSH_BUTTON Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_SECURITY_WPS_PUSH_BUTTON (9)
```

Description

This is macro DRV_WIFI_SECURITY_WPS_PUSH_BUTTON.

DRV_WIFI_WEP104_KEY_LENGTH Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_WEP104_KEY_LENGTH (52)    // 4 keys of 13 bytes each
```

Description

4 keys of 13 bytes each

DRV_WIFI_WEP40_KEY_LENGTH Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_WEP40_KEY_LENGTH (20)    // 4 keys of 5 bytes each
```

Description

4 keys of 5 bytes each

DRV_WIFI_WPA_KEY_LENGTH Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_WPA_KEY_LENGTH (32)
```

Description

This is macro DRV_WIFI_WPA_KEY_LENGTH.

DRV_WIFI_WPS_PIN_LENGTH Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_WPS_PIN_LENGTH 8       // 7 digits + checksum byte
```

Description

7 digits + checksum byte

DRV_WIFI_ADHOC_MODES Enumeration

Selection of different Ad-Hoc connection modes.

File

drv_wifi.h

C

```
typedef enum adhocMode {
  DRV_WIFI_ADHOC_CONNECT_THEN_START = 0,
  DRV_WIFI_ADHOC_CONNECT_ONLY = 1,
  DRV_WIFI_ADHOC_START_ONLY = 2
} DRV_WIFI_ADHOC_MODES;
```

Members

Members	Description
DRV_WIFI_ADHOC_CONNECT_THEN_START = 0	try to connect existing Ad-Hoc network, if not found then start network
DRV_WIFI_ADHOC_CONNECT_ONLY = 1	only connect to existing Ad-Hoc network
DRV_WIFI_ADHOC_START_ONLY = 2	only start a new Ad-Hoc network

Description

Ad-Hoc Modes

This enumeration identifies the Ad-Hoc modes that can be selected when connecting in Ad-Hoc mode.

DRV_WIFI_ADHOC_NETWORK_CONTEXT Structure

Contains data pertaining to Wi-Fi Ad-Hoc context.

File

drv_wifi.h

C

```
typedef struct {
  uint8_t mode;
  bool hiddenSsid;
  uint16_t beaconPeriod;
} DRV_WIFI_ADHOC_NETWORK_CONTEXT;
```

Members

Members	Description
uint8_t mode;	Defines how to start the Ad-Hoc network. See DRV_WIFI_ADHOC_MODE. Default is DRV_WIFI_ADHOC_CONNECT_THEN_START.
bool hiddenSsid;	When starting an Ad-Hoc network, the SSID can be hidden in the beacons. Set true to hide the SSID, else false. Default is false.
uint16_t beaconPeriod;	Sets the beacon period, in ms. Default is 100 ms.

Description

This structure contains MRF24WG Ad-Hoc context data. See DRV_WIFI_AdhocContextSet.

DRV_WIFI_CONNECTION_CONTEXT Structure

Contains data pertaining to MRF24WG connection context.

File

drv_wifi.h

C

```
typedef struct {
  uint8_t channel;
  uint8_t bssid[6];
} DRV_WIFI_CONNECTION_CONTEXT;
```

Members

Members	Description
uint8_t channel;	channel number of current connection
uint8_t bssid[6];	bssid of connected AP

Description

This structure contains MRF24WG connection context data. See DRV_WIFI_ConnectContextGet.

DRV_WIFI_CONNECTION_STATES Enumeration

Wi-Fi Connection States

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_CSTATE_NOT_CONNECTED = 1,
  DRV_WIFI_CSTATE_CONNECTION_IN_PROGRESS = 2,
  DRV_WIFI_CSTATE_CONNECTED_INFRASTRUCTURE = 3,
  DRV_WIFI_CSTATE_CONNECTED_ADHOC = 4,
  DRV_WIFI_CSTATE_RECONNECTION_IN_PROGRESS = 5,
  DRV_WIFI_CSTATE_CONNECTION_PERMANENTLY_LOST = 6
} DRV_WIFI_CONNECTION_STATES;
```

Members

Members	Description
DRV_WIFI_CSTATE_NOT_CONNECTED = 1	No Wi-Fi connection exists
DRV_WIFI_CSTATE_CONNECTION_IN_PROGRESS = 2	Wi-Fi connection in progress
DRV_WIFI_CSTATE_CONNECTED_INFRASTRUCTURE = 3	Wi-Fi connected in infrastructure mode
DRV_WIFI_CSTATE_CONNECTED_ADHOC = 4	Wi-Fi connected in adHoc mode
DRV_WIFI_CSTATE_RECONNECTION_IN_PROGRESS = 5	Wi-Fi in process of reconnecting
DRV_WIFI_CSTATE_CONNECTION_PERMANENTLY_LOST = 6	Wi-Fi connection permanently lost

Description

This enumeration identifies Wi-Fi Connection states. See DRV_WIFI_ConnectionStateGet.

DRV_WIFI_DEVICE_INFO Structure

Contains data pertaining to MRF24WG device type and version number.

File

drv_wifi.h

C

```
typedef struct {
  uint8_t deviceType;
  uint8_t romVersion;
  uint8_t patchVersion;
} DRV_WIFI_DEVICE_INFO;
```

Members

Members	Description
uint8_t deviceType;	MRF24W device type: DRV_WIFI_DEVICE_TYPE
uint8_t romVersion;	MRF24WG ROM version number
uint8_t patchVersion;	MRF24WG patch version number

Description

This structure contains MRF24WG device type and version number. See DRV_WIFI_DeviceInfoGet.

DRV_WIFI_DOMAIN_CODES Enumeration

Wi-Fi Regional Domain Codes

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_DOMAIN_FCC = 0,
  DRV_WIFI_DOMAIN_ETSI = 2,
  DRV_WIFI_DOMAIN_JAPAN = 7,
  DRV_WIFI_DOMAIN_OTHER = 7
} DRV_WIFI_DOMAIN_CODES;
```

Members

Members	Description
DRV_WIFI_DOMAIN_FCC = 0	FCC, available channels: 1 - 11
DRV_WIFI_DOMAIN_ETSI = 2	ESTI, available Channels: 1 - 13
DRV_WIFI_DOMAIN_JAPAN = 7	Japan, available Channels: 1 - 14
DRV_WIFI_DOMAIN_OTHER = 7	Other, available Channels: 1 - 14

Description

This enumeration identifies Wi-Fi regional domain codes. The regional domain can be determined by calling DRV_WIFI_RegionalDomainGet.

DRV_WIFI_EVENT_CONN_TEMP_LOST_CODES Enumeration

Selection of different codes when Wi-Fi connection is temporarily lost.

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_BEACON_TIMEOUT = 1,
  DRV_WIFI_DEAUTH_RECEIVED = 2,
```

```
  DRV_WIFI_DISASSOCIATE_RECEIVED = 3
} DRV_WIFI_EVENT_CONN_TEMP_LOST_CODES;
```

Members

Members	Description
DRV_WIFI_BEACON_TIMEOUT = 1	connection temporarily lost due to beacon timeout
DRV_WIFI_DEAUTH_RECEIVED = 2	connection temporarily lost due to deauthorization received from AP
DRV_WIFI_DISASSOCIATE_RECEIVED = 3	connection temporarily lost due to disassociation received from AP

Description

'Connection Temporarily Lost' event codes.

This enumeration identifies the codes for a connection temporarily lost. These codes are used in DRV_WIFI_ProcessEvent, case DRV_WIFI_EVENT_CONNECTION_TEMPORARILY_LOST.

DRV_WIFI_EVENT_INFO Enumeration

Selection of different EventInfo types.

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_JOIN_FAILURE = 2,
  DRV_WIFI_AUTHENTICATION_FAILURE = 3,
  DRV_WIFI_ASSOCIATION_FAILURE = 4,
  DRV_WIFI_WEP_HANDSHAKE_FAILURE = 5,
  DRV_WIFI_PSK_CALCULATION_FAILURE = 6,
  DRV_WIFI_PSK_HANDSHAKE_FAILURE = 7,
  DRV_WIFI_ADHOC_JOIN_FAILURE = 8,
  DRV_WIFI_SECURITY_MISMATCH_FAILURE = 9,
  DRV_WIFI_NO_SUITABLE_AP_FOUND_FAILURE = 10,
  DRV_WIFI_RETRY_FOREVER_NOT_SUPPORTED_FAILURE = 11,
  DRV_WIFI_LINK_LOST = 12,
  DRV_WIFI_TKIP_MIC_FAILURE = 13,
  DRV_WIFI_RSN_MIXED_MODE_NOT_SUPPORTED = 14,
  DRV_WIFI_RECV_DEAUTH = 15,
  DRV_WIFI_RECV_DISASSOC = 16,
  DRV_WIFI_WPS_FAILURE = 17,
  DRV_WIFI_LINK_DOWN = 19
} DRV_WIFI_EVENT_INFO;
```

Description

EventInfo Types

This enumeration identifies the eventInfo types used in DRV_WIFI_ProcessEvent, case DRV_WIFI_EVENT_CONNECTION_FAILED.

DRV_WIFI_EVENTS Enumeration

Selections for events that can occur.

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_EVENT_NONE = 0,
  DRV_WIFI_EVENT_CONNECTION_SUCCESSFUL = 1,
  DRV_WIFI_EVENT_CONNECTION_FAILED = 2,
  DRV_WIFI_EVENT_CONNECTION_TEMPORARILY_LOST = 3,
  DRV_WIFI_EVENT_CONNECTION_PERMANENTLY_LOST = 4,
  DRV_WIFI_EVENT_CONNECTION_REESTABLISHED = 5,
  DRV_WIFI_EVENT_FLASH_UPDATE_SUCCESSFUL = 6,
  DRV_WIFI_EVENT_FLASH_UPDATE_FAILED = 7,
  DRV_WIFI_EVENT_KEY_CALCULATION_REQUEST = 8,
  DRV_WIFI_EVENT_INVALID_WPS_PIN = 9,
  DRV_WIFI_EVENT_SCAN_RESULTS_READY = 10,
  DRV_WIFI_EVENT_IE_RESULTS_READY = 11,
```

```
  DRV_WIFI_EVENT_SOFT_AP = 12,
  DRV_WIFI_EVENT_DISCONNECT_DONE = 13,
  DRV_WIFI_EVENT_UPDATE = 14,
  DRV_WIFI_EVENT_ERROR = 15
} DRV_WIFI_EVENTS;
```

Members

Members	Description
DRV_WIFI_EVENT_NONE = 0	No event has occurred
DRV_WIFI_EVENT_CONNECTION_SUCCESSFUL = 1	Connection attempt to network successful
DRV_WIFI_EVENT_CONNECTION_FAILED = 2	Connection attempt failed
DRV_WIFI_EVENT_CONNECTION_TEMPORARILY_LOST = 3	Connection lost; MRF24WG attempting to reconnect
DRV_WIFI_EVENT_CONNECTION_PERMANENTLY_LOST = 4	Connection lost; MRF24WG no longer trying to connect
DRV_WIFI_EVENT_CONNECTION_REESTABLISHED = 5	Connection has been reestablished
DRV_WIFI_EVENT_FLASH_UPDATE_SUCCESSFUL = 6	Update to FLASH successful
DRV_WIFI_EVENT_FLASH_UPDATE_FAILED = 7	Update to FLASH failed
DRV_WIFI_EVENT_KEY_CALCULATION_REQUEST = 8	Key calculation is required
DRV_WIFI_EVENT_INVALID_WPS_PIN = 9	Invalid WPS pin was entered
DRV_WIFI_EVENT_SCAN_RESULTS_READY = 10	Scan results are ready
DRV_WIFI_EVENT_IE_RESULTS_READY = 11	IE data ready
DRV_WIFI_EVENT_SOFT_AP = 12	Client connection events
DRV_WIFI_EVENT_DISCONNECT_DONE = 13	Disconnect done event
DRV_WIFI_EVENT_UPDATE = 14	Wi-Fi update event occurred
DRV_WIFI_EVENT_ERROR = 15	Wi-Fi error event occurred

Description

Wi-Fi Events

This enumeration identifies the Wi-Fi events that can occur and will be sent to DRV_WIFI_ProcessEvent.

DRV_WIFI_GENERAL_ERRORS Enumeration

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_ERROR_IN_HIBERNATE_MODE = 100
} DRV_WIFI_GENERAL_ERRORS;
```

Members

Members	Description
DRV_WIFI_ERROR_IN_HIBERNATE_MODE = 100	invalid operation while MRF24WG is in hibernate mode

Description

This is type DRV_WIFI_GENERAL_ERRORS.

DRV_WIFI_HIBERNATE_STATES Enumeration

Wi-Fi Hibernate States

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_HB_NO_SLEEP = 0,
  DRV_WIFI_HB_ENTER_SLEEP = 1,
```

```
  DRV_WIFI_HB_WAIT_WAKEUP = 2
} DRV_WIFI_HIBERNATE_STATES;
```

Description

This enumeration identifies Wi-Fi hibernate states.

DRV_WIFI_MAC_STATS Structure

Wi-Fi MIB states

File

drv_wifi.h

C

```
typedef struct {
  uint32_t MibWEPExcludeCtr;
  uint32_t MibTxBytesCtr;
  uint32_t MibTxMulticastCtr;
  uint32_t MibTxFailedCtr;
  uint32_t MibTxRtryCtr;
  uint32_t MibTxMultRtryCtr;
  uint32_t MibTxSuccessCtr;
  uint32_t MibRxDupCtr;
  uint32_t MibRxCtsSuccCtr;
  uint32_t MibRxCtsFailCtr;
  uint32_t MibRxAckFailCtr;
  uint32_t MibRxBytesCtr;
  uint32_t MibRxFragCtr;
  uint32_t MibRxMultCtr;
  uint32_t MibRxFCSErrCtr;
  uint32_t MibRxWEPUndecryptCtr;
  uint32_t MibRxFragAgedCtr;
  uint32_t MibRxMICFailureCtr;
} DRV_WIFI_MAC_STATS;
```

Members

Members	Description
uint32_t MibWEPExcludeCtr;	Number of frames received with the Protected Frame sub-field of the Frame Control field set to zero and the value of dot11ExcludeUnencrypted causes that frame to be discarded.
uint32_t MibTxBytesCtr;	Total number of TX bytes that have been transmitted
uint32_t MibTxMulticastCtr;	Number of frames successfully transmitted that had the multicast bit set in the destination MAC address
uint32_t MibTxFailedCtr;	Number of TX frames that failed due to the number of transmits exceeding the retry count
uint32_t MibTxRtryCtr;	Number of times a transmitted frame needed to be retried.
uint32_t MibTxMultRtryCtr;	Number of times a frame was successfully transmitted after more than one retransmission.
uint32_t MibTxSuccessCtr;	Number of TX frames successfully transmitted.
uint32_t MibRxDupCtr;	Number of frames received where the Sequence Control field indicates a duplicate.
uint32_t MibRxCtsSuccCtr;	Number of CTS frames received in response to an RTS frame.
uint32_t MibRxCtsFailCtr;	Number of times an RTS frame was not received in response to a CTS frame.
uint32_t MibRxAckFailCtr;	Number of times an ACK was not received in response to a TX frame.
uint32_t MibRxBytesCtr;	Total number of Rx bytes received.
uint32_t MibRxFragCtr;	Number of successful received frames (management or data).
uint32_t MibRxMultCtr;	Number of frames received with the multicast bit set in the destination MAC address.
uint32_t MibRxFCSErrCtr;	Number of frames received with an invalid Frame Checksum (FCS).
uint32_t MibRxWEPUndecryptCtr;	Number of frames received where the Protected Frame sub-field of the Frame Control Field is set to one and the WEPOn value for the key mapped to the transmitter's MAC address indicates the frame should not have been encrypted.
uint32_t MibRxFragAgedCtr;	Number of times that fragments aged out, or were not received in the allowable time.
uint32_t MibRxMICFailureCtr;	Number of MIC failures that have occurred.

Description

Wi-Fi MIB states

This structure contains all the MIB data returned from the MRF24WG when DRV_WIFI_MacStatsGet() is called.

DRV_WIFI_MGMT_ERRORS Enumeration

Error codes returned when a management message is sent to the MRF24WG module.

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_SUCCESS = 1,
  DRV_WIFI_ERROR_INVALID_SUBTYPE = 2,
  DRV_WIFI_ERROR_OPERATION_CANCELLED = 3,
  DRV_WIFI_ERROR_FRAME_END_OF_LINE_OCCURRED = 4,
  DRV_WIFI_ERROR_FRAME_RETRY_LIMIT_EXCEEDED = 5,
  DRV_WIFI_ERROR_EXPECTED_BSS_VALUE_NOT_IN_FRAME = 6,
  DRV_WIFI_ERROR_FRAME_SIZE_EXCEEDS_BUFFER_SIZE = 7,
  DRV_WIFI_ERROR_FRAME_ENCRYPT_FAILED = 8,
  DRV_WIFI_ERROR_INVALID_PARAM = 9,
  DRV_WIFI_ERROR_AUTH_REQ_ISSUED_WHILE_IN_AUTH_STATE = 10,
  DRV_WIFI_ERROR_ASSOC_REQ_ISSUED_WHILE_IN_ASSOC_STATE = 11,
  DRV_WIFI_ERROR_INSUFFICIENT_RESOURCES = 12,
  DRV_WIFI_ERROR_TIMEOUT_OCCURRED = 13,
  DRV_WIFI_ERROR_BAD_EXCHANGE_ENCOUNTERED_IN_FRAME_RECEPTION = 14,
  DRV_WIFI_ERROR_AUTH_REQUEST_REFUSED = 15,
  DRV_WIFI_ERROR_ASSOCIATION_REQUEST_REFUSED = 16,
  DRV_WIFI_ERROR_PRIOR_MGMT_REQUEST_IN_PROGRESS = 17,
  DRV_WIFI_ERROR_NOT_IN_JOINED_STATE = 18,
  DRV_WIFI_ERROR_NOT_IN_ASSOCIATED_STATE = 19,
  DRV_WIFI_ERROR_NOT_IN_AUTHENTICATED_STATE = 20,
  DRV_WIFI_ERROR_SUPPLICANT_FAILED = 21,
  DRV_WIFI_ERROR_UNSUPPORTED_FEATURE = 22,
  DRV_WIFI_ERROR_REQUEST_OUT_OF_SYNC = 23,
  DRV_WIFI_ERROR_CP_INVALID_ELEMENT_TYPE = 24,
  DRV_WIFI_ERROR_CP_INVALID_PROFILE_ID = 25,
  DRV_WIFI_ERROR_CP_INVALID_DATA_LENGTH = 26,
  DRV_WIFI_ERROR_CP_INVALID_SSID_LENGTH = 27,
  DRV_WIFI_ERROR_CP_INVALID_SECURITY_TYPE = 28,
  DRV_WIFI_ERROR_CP_INVALID_SECURITY_KEY_LENGTH = 29,
  DRV_WIFI_ERROR_CP_INVALID_WEP_KEY_ID = 30,
  DRV_WIFI_ERROR_CP_INVALID_NETWORK_TYPE = 31,
  DRV_WIFI_ERROR_CP_INVALID_ADHOC_MODE = 32,
  DRV_WIFI_ERROR_CP_INVALID_SCAN_TYPE = 33,
  DRV_WIFI_ERROR_CP_INVALID_CP_LIST = 34,
  DRV_WIFI_ERROR_CP_INVALID_CHANNEL_LIST_LENGTH = 35,
  DRV_WIFI_ERROR_NOT_CONNECTED = 36,
  DRV_WIFI_ERROR_ALREADY_CONNECTING = 37,
  DRV_WIFI_ERROR_DISCONNECT_FAILED = 38,
  DRV_WIFI_ERROR_NO_STORED_BSS_DESCRIPTOR = 39,
  DRV_WIFI_ERROR_INVALID_MAX_POWER = 40,
  DRV_WIFI_ERROR_CONNECTION_TERMINATED = 41,
  DRV_WIFI_ERROR_HOST_SCAN_NOT_ALLOWED = 42,
  DRV_WIFI_ERROR_INVALID_WPS_PIN = 44
} DRV_WIFI_MGMT_ERRORS;
```

Members

Members	Description
DRV_WIFI_ERROR_DISCONNECT_FAILED = 38	Disconnect failed. Disconnect is allowed only when module is in connected state.
DRV_WIFI_ERROR_NO_STORED_BSS_DESCRIPTOR = 39	No stored scan results.
DRV_WIFI_ERROR_HOST_SCAN_NOT_ALLOWED = 42	Host Scan Failed. Host scan is allowed only in idle or connected state
DRV_WIFI_ERROR_INVALID_WPS_PIN = 44	WPS pin was invalid.

Description

Management Message Error Codes

This enumeration identifies the errors that can occur when a DRV_WIFI API function call results in a management message being sent, via SPI, to the MRF24WG module.

DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT Structure

Contains data pertaining to Wi-Fi Soft AP event.

File

drv_wifi.h

C

```
typedef struct {
  uint8_t reason;
  uint8_t address[6];
} DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT;
```

Members

Members	Description
uint8_t reason;	reason code
uint8_t address[6];	MAC address

Description

This structure contains data pertaining to Soft AP event. See DRV_WIFI_SoftApEventInfoGet.

DRV_WIFI_MULTICAST_FILTER_IDS Enumeration

Selections for software Multicast filter IDs.

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_MULTICAST_FILTER_1 = 4,
  DRV_WIFI_MULTICAST_FILTER_2 = 5,
  DRV_WIFI_MULTICAST_FILTER_3 = 6,
  DRV_WIFI_MULTICAST_FILTER_4 = 7,
  DRV_WIFI_MULTICAST_FILTER_5 = 8,
  DRV_WIFI_MULTICAST_FILTER_6 = 9,
  DRV_WIFI_MULTICAST_FILTER_7 = 10,
  DRV_WIFI_MULTICAST_FILTER_8 = 11,
  DRV_WIFI_MULTICAST_FILTER_9 = 12,
  DRV_WIFI_MULTICAST_FILTER_10 = 13,
  DRV_WIFI_MULTICAST_FILTER_11 = 14,
  DRV_WIFI_MULTICAST_FILTER_12 = 15,
  DRV_WIFI_MULTICAST_FILTER_13 = 16,
  DRV_WIFI_MULTICAST_FILTER_14 = 17,
  DRV_WIFI_MULTICAST_FILTER_15 = 18,
  DRV_WIFI_MULTICAST_FILTER_16 = 19
} DRV_WIFI_MULTICAST_FILTER_IDS;
```

Description

Multicast Filter IDs

This enumeration identifies the multicast filters that can be selected. See DRV_WIFI_MulticastFilterSet.

DRV_WIFI_MULTICAST_FILTERS Enumeration

Selections for Software Multicast Filters.

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_MULTICAST_DISABLE_ALL = 0,
  DRV_WIFI_MULTICAST_ENABLE_ALL = 1,
  DRV_WIFI_MULTICAST_USE_FILTERS = 2
} DRV_WIFI_MULTICAST_FILTERS;
```

Members

Members	Description
DRV_WIFI_MULTICAST_DISABLE_ALL = 0	Discard all received multicast messages.
DRV_WIFI_MULTICAST_ENABLE_ALL = 1	Forward all multicast messages to host MCU.
DRV_WIFI_MULTICAST_USE_FILTERS = 2	Use the MAC filtering capability for multicast messages.

Description

Multicast Filter Modes

This enumeration identifies the mode of multicast filters that can be selected. See DRV_WIFI_MulticastFilterSet().

DRV_WIFI_POWER_SAVE_STATES Enumeration

Wi-Fi Power-Saving States

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_PS_HIBERNATE = 1,
  DRV_WIFI_PS_PS_POLL_DTIM_ENABLED = 2,
  DRV_WIFI_PS_PS_POLL_DTIM_DISABLED = 3,
  DRV_WIFI_PS_OFF = 4
} DRV_WIFI_POWER_SAVE_STATES;
```

Members

Members	Description
DRV_WIFI_PS_HIBERNATE = 1	enable hibernate mode
DRV_WIFI_PS_PS_POLL_DTIM_ENABLED = 2	enable power-saving mode with DTIM enabled
DRV_WIFI_PS_PS_POLL_DTIM_DISABLED = 3	enable power-saving mode with DTIM disabled
DRV_WIFI_PS_OFF = 4	disable power-saving mode

Description

This enumeration identifies Wi-Fi power-saving states. See DRV_WIFI_PsPollEnable.

DRV_WIFI_PS_POLL_CONTEXT Structure

Contains data pertaining to Wi-Fi PS-Poll context.

File

drv_wifi.h

C

```
typedef struct {
  uint16_t listenInterval;
  uint16_t dtimInterval;
  bool useDtim;
} DRV_WIFI_PS_POLL_CONTEXT;
```

Members

Members	Description
uint16_t listenInterval;	Number of 100ms intervals between instances when the MRF24WG wakes up to received buffered messages from the network. Each count represents 100ms. For example, 1 = 100ms, 2 = 200ms, etc. The default is 1 (100ms).
uint16_t dtimInterval;	Only used if useDtim is true. The DTIM period indicates how often clients serviced by the access point should check for buffered multicast or broadcast messages awaiting pickup on the access point. The DTIM interval is measured in number of beacon periods. Default for DTIM period is 2.
bool useDtim;	True: (default) check for buffered multicast or broadcast messages on the dtimInterval. False: check for buffered multicast or broadcast messages on the listenInterval

Description

This structure contains MRF24WG PS-Poll context data. See DRV_WIFI_PsPollEnable.

DRV_WIFI_REASON_CODES Enumeration

Selection of different codes when a deauthorization or disassociation event has occurred.

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_UNSPECIFIED = 1,
  DRV_WIFI_REASON_PREV_AUTH_NOT_VALID = 2,
  DRV_WIFI_DEAUTH_LEAVING = 3,
  DRV_WIFI_DISASSOC_DUE_TO_INACTIVITY = 4,
  DRV_WIFI_DISASSOC_AP_BUSY = 5,
  DRV_WIFI_CLASS2_FRAME_FROM_NONAUTH_STA = 6,
  DRV_WIFI_CLASS3_FRAME_FROM_NONASSOC_STA = 7,
  DRV_WIFI_DISASSOC_STA_HAS_LEFT = 8,
  DRV_WIFI_STA_REQ_ASSOC_WITHOUT_AUTH = 9,
  DRV_WIFI_INVALID_IE = 13,
  DRV_WIFI_MIC_FAILURE = 14,
  DRV_WIFI_4WAY_HANDSHAKE_TIMEOUT = 15,
  DRV_WIFI_GROUP_KEY_HANDSHAKE_TIMEOUT = 16,
  DRV_WIFI_IE_DIFFERENT = 17,
  DRV_WIFI_INVALID_GROUP_CIPHER = 18,
  DRV_WIFI_INVALID_PAIRWISE_CIPHER = 19,
  DRV_WIFI_INVALID_AKMP = 20,
  DRV_WIFI_UNSUPP_RSN_VERSION = 21,
  DRV_WIFI_INVALID_RSN_IE_CAP = 22,
  DRV_WIFI_IEEE8021X_FAILED = 23,
  DRV_WIFI_CIPHER_SUITE_REJECTED = 24
} DRV_WIFI_REASON_CODES;
```

Description

Deauthorization/Disassociate Reason Codes

This enumeration identifies the reason codes for a connection lost due to a deauthorization or disassociation from the AP.

DRV_WIFI_RECONNECT_MODES Enumeration

Selection of different reconnection modes.

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_DO_NOT_ATTEMPT_TO_RECONNECT = 0,
  DRV_WIFI_ATTEMPT_TO_RECONNECT = 1
} DRV_WIFI_RECONNECT_MODES;
```

Description

Wi-Fi Reconnect Modes

This enumeration identifies the reconnection modes that can be used in DRV_WIFI_ReconnectModeSet.

DRV_WIFI_SCAN_CONTEXT Structure

Contains data pertaining to Wi-Fi scan context.

File

drv_wifi.h

C

```
typedef struct {
```

```
  uint8_t scanType;
  uint8_t scanCount;
  uint16_t minChannelTime;
  uint16_t maxChannelTime;
  uint16_t probeDelay;
} DRV_WIFI_SCAN_CONTEXT;
```

Members

Members	Description
uint8_t scanType;	802.11 allows for active scanning, where the device sends out a broadcast probe request seeking an access point. Also allowed is passive scanning where the device only listens to beacons being broadcast from access points. Set to DRV_WIFI_ACTIVE_SCAN (default) or DRV_WIFI_PASSIVE_SCAN
uint8_t scanCount;	The number of times to scan a channel while attempting to find a particular access point. Default is 1
uint16_t minChannelTime;	The minimum time (in milliseconds) the MRF24WG will wait for a probe response after sending a probe request. If no probe responses are received in minChannelTime, the MRF24WG will go on to the next channel, if any are left to scan, or quit. Default is 200ms.
uint16_t maxChannelTime;	If a probe response is received within minChannelTime, the MRF24WG will continue to collect any additional probe responses up to maxChannelTime before going to the next channel in the channelList. Units are in milliseconds. Default is 400ms.
uint16_t probeDelay;	The number of microseconds to delay before transmitting a probe request following the channel change during scanning. Default is 20uS.

Description

This structure contains MRF24WG scan context data. See DRV_WIFI_ScanContextSet.

DRV_WIFI_SCAN_RESULT Structure

Contains data pertaining to Wi-Fi scan results.

File

drv_wifi.h

C

```
typedef struct {
  uint8_t bssid[DRV_WIFI_BSSID_LENGTH];
  uint8_t ssid[DRV_WIFI_MAX_SSID_LENGTH];
  uint8_t apConfig;
  uint8_t reserved;
  uint16_t beaconPeriod;
  uint16_t atimWindow;
  uint8_t basicRateSet[DRV_WIFI_MAX_NUM_RATES];
  uint8_t rssi;
  uint8_t numRates;
  uint8_t dtimPeriod;
  uint8_t bssType;
  uint8_t channel;
  uint8_t ssidLen;
} DRV_WIFI_SCAN_RESULT;
```

Members

Members	Description
uint8_t bssid[DRV_WIFI_BSSID_LENGTH];	Network BSSID value
uint8_t ssid[DRV_WIFI_MAX_SSID_LENGTH];	Network SSID value
uint8_t apConfig;	Access Point configuration (see description)
uint8_t reserved;	not used
uint16_t beaconPeriod;	Network beacon interval
uint16_t atimWindow;	Only valid if bssType = DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE

uint8_t basicRateSet[DRV_WIFI_MAX_NUM_RATES];	List of Network basic rates. Each rate has the following format: Bit 7 • 0: rate is not part of the basic rates set • 1: rate is part of the basic rates set Bits 6:0 Multiple of 500kbps giving the supported rate. For example, a value of 2 (2 * 500kbps) indicates that 1mbps is a supported rate. A value of 4 in this field indicates a 2mbps rate (4 * 500kbps).
uint8_t rssi;	Signal strength of received frame beacon or probe response. Will range from a low of 43 to a high of 128.
uint8_t numRates;	Number of valid rates in basicRates
uint8_t dtimPeriod;	Part of TIM element
uint8_t bssType;	DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE or DRV_WIFI_NETWORK_TYPE_ADHOC
uint8_t channel;	Channel number
uint8_t ssidLen;	Number of valid characters in ssid

Description

This structure contains the result of Wi-Fi scan operation. See DRV_WIFI_ScanResultGet.

apConfig Bit Mask

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
WPA2	WPA	Preamble	Privacy	Reserved	Reserved	Reserved	IE

IE	1 if AP broadcasting one or more Information Elements, else 0
Privacy	0 : AP is open (no security) 1: AP using security, if neither WPA and WPA2 set then security is WEP.
Preamble	0: AP transmitting with short preamble 1: AP transmitting with long preamble
WPA	Only valid if Privacy is 1. 0: AP does not support WPA 1: AP supports WPA
WPA2	Only valid if Privacy is 1. 0: AP does not support WPA2 1: AP supports WPA2

DRV_WIFI_SCAN_TYPES Enumeration

Selection of different Wi-Fi scan types.

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_ACTIVE_SCAN = 1,
  DRV_WIFI_PASSIVE_SCAN = 2
} DRV_WIFI_SCAN_TYPES;
```

Description

Wi-Fi Scan Types

This enumeration identifies the Wi-Fi scan types that can be selected.

DRV_WIFI_SOFT_AP_EVENT_REASON_CODES Enumeration

Wi-Fi Soft AP Event Reason Codes

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_SOFTAP_EVENT_LINK_LOST = 0,
  DRV_WIFI_SOFTAP_EVENT_RECEIVED_DEAUTH = 1
} DRV_WIFI_SOFT_AP_EVENT_REASON_CODES;
```

Description

This enumeration identifies Wi-Fi Soft AP events.

DRV_WIFI_SOFT_AP_STATES Enumeration

Wi-Fi Soft AP Events

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_SOFTAP_EVENT_CONNECTED = 0,
  DRV_WIFI_SOFTAP_EVENT_DISCONNECTED = 1
} DRV_WIFI_SOFT_AP_STATES;
```

Description

This enumeration identifies Wi-Fi Soft AP events.

DRV_WIFI_STATUS_CODES Enumeration

Selection of different codes when Wi-Fi connection fails due to association or authentication failure.

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_UNSPECIFIED_FAILURE = 1,
  DRV_WIFI_CAPS_UNSUPPORTED = 10,
  DRV_WIFI_REASSOC_NO_ASSOC = 11,
  DRV_WIFI_ASSOC_DENIED_UNSPEC = 12,
  DRV_WIFI_NOT_SUPPORTED_AUTH_ALG = 13,
  DRV_WIFI_UNKNOWN_AUTH_TRANSACTION = 14,
  DRV_WIFI_CHALLENGE_FAIL = 15,
  DRV_WIFI_AUTH_TIMEOUT = 16,
  DRV_WIFI_AP_UNABLE_TO_HANDLE_NEW_STA = 17,
  DRV_WIFI_ASSOC_DENIED_RATES = 18,
  DRV_WIFI_ASSOC_DENIED_NOSHORTPREAMBLE = 19,
  DRV_WIFI_ASSOC_DENIED_NOPBCC = 20,
  DRV_WIFI_ASSOC_DENIED_NOAGILITY = 21,
  DRV_WIFI_ASSOC_DENIED_NOSHORTTIME = 25,
  DRV_WIFI_ASSOC_DENIED_NODSSSOFDM = 26,
  DRV_WIFI_S_INVALID_IE = 40,
  DRV_WIFI_S_INVALID_GROUPCIPHER = 41,
  DRV_WIFI_S_INVALID_PAIRWISE_CIPHER = 42,
  DRV_WIFI_S_INVALID_AKMP = 43,
  DRV_WIFI_UNSUPPORTED_RSN_VERSION = 44,
  DRV_WIFI_S_INVALID_RSN_IE_CAP = 45,
  DRV_WIFI_S_CIPHER_SUITE_REJECTED = 46,
  DRV_WIFI_TIMEOUT = 47
} DRV_WIFI_STATUS_CODES;
```

Description

Status codes for connection for association or authentication failure.

This enumeration identifies the codes for a connection failure due to association or authentication failure. These codes are used in DRV_WIFI_ProcessEvent, case DRV_WIFI_EVENT_CONNECTION_FAILED.

DRV_WIFI_TX_MODES Enumeration

Selections for Wi-Fi TX Mode

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_TXMODE_G_RATES = 0,
  DRV_WIFI_TXMODE_B_RATES = 1,
  DRV_WIFI_TXMODE_LEGACY_RATES = 2
} DRV_WIFI_TX_MODES;
```

Members

Members	Description
DRV_WIFI_TXMODE_G_RATES = 0	Use 802.11 'g' rates
DRV_WIFI_TXMODE_B_RATES = 1	Use only 802.11 'b' rates
DRV_WIFI_TXMODE_LEGACY_RATES = 2	Use only 1 and 2 Mbps rates

Description

TX Modes

This enumeration identifies the choices the MRF24WG TX mode. It is recommended to use the DRV_WIFI_TXMODE_G_RATES for best performance. See DRV_WIFI_TxModeSet.

DRV_WIFI_WEP_CONTEXT Structure

Contains data pertaining to Wi-Fi WEP context.

File

drv_wifi.h

C

```
typedef struct {
  uint8_t wepSecurityType;
  uint8_t wepKey[DRV_WIFI_MAX_WEP_KEY_LENGTH];
  uint8_t wepKeyLength;
  uint8_t wepKeyType;
} DRV_WIFI_WEP_CONTEXT;
```

Members

Members	Description
uint8_t wepSecurityType;	DRV_WIFI_SECURITY_WEP_40 or DRV_WIFI_SECURITY_WEP_104
uint8_t wepKey[DRV_WIFI_MAX_WEP_KEY_LENGTH];	Array containing four WEP binary keys. This will be four, 5-byte keys for WEP-40 or four, thirteen-byte keys for WEP-104.
uint8_t wepKeyLength;	number of bytes pointed to by p_wepKey
uint8_t wepKeyType;	DRV_WIFI_SECURITY_WEP_OPENKEY (default) or DRV_WIFI_SECURITY_WEP_SHAREDKEY

Description

This structure contains MRF24WG WEP context. See DRV_WIFI_SecurityWepSet.

DRV_WIFI_WEP_KEY_TYPE Enumeration

Selections for WEP key type when using WEP security.

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_SECURITY_WEP_SHAREDKEY = 0,
  DRV_WIFI_SECURITY_WEP_OPENKEY = 1
} DRV_WIFI_WEP_KEY_TYPE;
```

Members

Members	Description
DRV_WIFI_SECURITY_WEP_SHAREDKEY = 0	use WEP shared key
DRV_WIFI_SECURITY_WEP_OPENKEY = 1	use WEP open key (default)

Description

WEP Key Types

This enumeration identifies the choices for the WEP key type when using WEP security. The recommended key type (and default) is Open key.

DRV_WIFI_WPA_CONTEXT Structure

Contains data pertaining to Wi-Fi WPA.

File

drv_wifi.h

C

```
typedef struct {
  uint8_t wpaSecurityType;
  DRV_WIFI_WPA_KEY_INFO keyInfo;
} DRV_WIFI_WPA_CONTEXT;
```

Members

Members	Description
uint8_t wpaSecurityType;	desired security type (see description)
DRV_WIFI_WPA_KEY_INFO keyInfo;	see DRV_WIFI_WPA_KEY_INFO

Description

This structure contains MRF24WG WPA context. See DRV_WIFI_SecurityWpaSet.

DRV_WIFI_SECURITY_WPA_WITH_KEY	Select WPA with binary key
DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE	Select WPA with passphrase
DRV_WIFI_SECURITY_WPA2_WITH_KEY	Select WPA2 with binary key
DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE	Select WPA2 with passphrase
DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY	Auto-select between WPA/WPA2 with binary key
DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE	Auto-select between WPA/WPA2 with passphrase

DRV_WIFI_WPA_KEY_INFO Structure

Contains data pertaining to Wi-Fi WPA Key.

File

drv_wifi.h

C

```
typedef struct {
  uint8_t key[DRV_WIFI_MAX_WPA_PASS_PHRASE_LENGTH];
  uint8_t keyLength;
} DRV_WIFI_WPA_KEY_INFO;
```

Members

Members	Description
uint8_t key[DRV_WIFI_MAX_WPA_PASS_PHRASE_LENGTH];	binary key or passphrase
uint8_t keyLength;	number of bytes in binary key (always 32) or passphrase

Description

This structure contains MRF24WG WPA key info. This structure is used in the DRV_WIFI_WPA_CONTEXT and DRV_WIFI_WPS_CONTEXT structures.

DRV_WIFI_WPS_AUTH_TYPES Enumeration

Selection of WPS Authorization Types

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_WPS_AUTH_OPEN = 0x01,
  DRV_WIFI_WPS_AUTH_WPA_PSK = 0x02,
  DRV_WIFI_WPS_AUTH_SHARED = 0x04,
  DRV_WIFI_WPS_AUTH_WPA = 0x08,
  DRV_WIFI_WPS_AUTH_WPA2 = 0x10,
  DRV_WIFI_WPS_AUTH_WPA2_PSK = 0x20
} DRV_WIFI_WPS_AUTH_TYPES;
```

Description

Wi-Fi WPS authorization types.

This enumeration identifies the WPS authorization types.

DRV_WIFI_WPS_CONTEXT Structure

Contains data pertaining to Wi-Fi WPS security.

File

drv_wifi.h

C

```
typedef struct {
  uint8_t wpsSecurityType;
  uint8_t wpsPin[DRV_WIFI_WPS_PIN_LENGTH];
  uint8_t wpsPinLength;
} DRV_WIFI_WPS_CONTEXT;
```

Members

Members	Description
uint8_t wpsSecurityType;	DRV_WIFI_SECURITY_WPS_PUSH_BUTTON or DRV_WIFI_SECURITY_WPS_PIN
uint8_t wpsPin[DRV_WIFI_WPS_PIN_LENGTH];	if using DRV_WIFI_SECURITY_WPS_PIN then pointer to 8-digit pin
uint8_t wpsPinLength;	should always be 8 if used, 0 if not used

Description

This structure contains MRF24WG WPS security context. See DRV_WIFI_SecurityWpsSet.

DRV_WIFI_WPS_CREDENTIAL Structure

Contains data pertaining to Wi-Fi WPS Credentials.

File

drv_wifi.h

C

```
typedef struct {
  uint8_t ssid[DRV_WIFI_MAX_SSID_LENGTH];
  uint8_t netKey[DRV_WIFI_MAX_SECURITY_KEY_LENGTH];
  uint16_t authType;
  uint16_t encType;
  uint8_t netIdx;
  uint8_t ssidLen;
  uint8_t keyIdx;
  uint8_t keyLen;
  uint8_t bssid[DRV_WIFI_BSSID_LENGTH];
} DRV_WIFI_WPS_CREDENTIAL;
```

Members

Members	Description
uint8_t ssid[DRV_WIFI_MAX_SSID_LENGTH];	network SSID

uint8_t netKey[DRV_WIFI_MAX_SECURITY_KEY_LENGTH];	binary security key (not used if security is open)
uint16_t authType;	WPS authorization type (see description)
uint16_t encType;	encoding type (see description)
uint8_t netIdx;	not used
uint8_t ssidLen;	number of bytes in SSID
uint8_t keyIdx;	Only valid encType = WF_ENC_WEP. This is the index of the WEP key being used.
uint8_t keyLen;	number of bytes in netKey
uint8_t bssid[DRV_WIFI_BSSID_LENGTH];	MAC address of AP

Description

This structure contains data pertaining to the configuration of the Wi-Fi WPS credentials.

'authType' Field	Description
DRV_WIFI_WPS_AUTH_OPEN	Open Security
DRV_WIFI_WPS_AUTH_WPA_PSK	WPA with PSK
DRV_WIFI_WPS_AUTH_SHARED	Shared Key
DRV_WIFI_WPS_AUTH_WPA	WPA
DRV_WIFI_WPS_AUTH_WPA2	WPA2
DRV_WIFI_WPS_AUTH_WPA2_PSK	WPA2 with PSK

'encType' Field	Description
DRV_WIFI_WPS_ENC_NONE	No encoding
DRV_WIFI_WPS_ENC_WEP	WEP encoding
DRV_WIFI_WPS_ENC_TKIP	TKIP encoding
DRV_WIFI_ENC_AES	AES encoding

DRV_WIFI_WPS_ENCODE_TYPES Enumeration

Selection of WPS Encoding Types

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_WPS_ENC_NONE = 0x01,
  DRV_WIFI_WPS_ENC_WEP = 0x02,
  DRV_WIFI_WPS_ENC_TKIP = 0x04,
  DRV_WIFI_ENC_AES = 0x08
} DRV_WIFI_WPS_ENCODE_TYPES;
```

Description

Wi-Fi WPS encoding types.

This enumeration identifies the WPS encoding types.

DRV_WIFI_WPS_ERROR_CONFIG_CODES Enumeration

Selection of different codes when a WPS connection fails.

File

drv_wifi.h

C

```
typedef enum {
```

```
  DRV_WIFI_WPS_NOERR = 0,
  DRV_WIFI_WPS_SESSION_OVERLAPPED = 1,
  DRV_WIFI_WPS_DECRYPT_CRC_FAILURE = 2,
  DRV_WIFI_WPS_24G_NOT_SUPPORTED = 3,
  DRV_WIFI_WPS_RETRY_FAILURE = 4,
  DRV_WIFI_WPS_INVALID_MSG = 5,
  DRV_WIFI_WPS_AUTH_FAILURE = 6,
  DRV_WIFI_WPS_ASSOC_FAILURE = 7,
  DRV_WIFI_WPS_MSG_TIMEOUT = 8,
  DRV_WIFI_WPS_SESSION_TIMEOUT = 9,
  DRV_WIFI_WPS_DEVPASSWD_AUTH_FAILURE = 10,
  DRV_WIFI_WPS_NO_CONN_TOREG = 11,
  DRV_WIFI_WPS_MULTI_PBC_DETECTED = 12,
  DRV_WIFI_WPS_EAP_FAILURE = 13,
  DRV_WIFI_WPS_DEV_BUSY = 14,
  DRV_WIFI_WPS_SETUP_LOCKED = 15
} DRV_WIFI_WPS_ERROR_CONFIG_CODES;
```

Description

WPS Config Error Codes

This enumeration identifies the codes that can take place when WPS fails.

DRV_WIFI_WPS_STATE_CODES Enumeration

Selection of different codes when a Extensible Authentication Protocol is used.

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_EAPOL_START = 1,
  DRV_WIFI_EAP_REQ_IDENTITY = 2,
  DRV_WIFI_EAP_RSP_IDENTITY = 3,
  DRV_WIFI_EAP_WPS_START = 4,
  DRV_WIFI_EAP_RSP_M1 = 5,
  DRV_WIFI_EAP_REQ_M2 = 6,
  DRV_WIFI_EAP_RSP_M3 = 7,
  DRV_WIFI_EAP_REQ_M4 = 8,
  DRV_WIFI_EAP_RSP_M5 = 9,
  DRV_WIFI_EAP_REQ_M6 = 10,
  DRV_WIFI_EAP_RSP_M7 = 11,
  DRV_WIFI_EAP_REQ_M8 = 12,
  DRV_WIFI_EAP_RSP_DONE = 13,
  DRV_WIFI_EAP_FAILURE = 14
} DRV_WIFI_WPS_STATE_CODES;
```

Description

WPS State Codes

This enumeration identifies the codes that can take place when using EAPOL.

DRV_WIFI_DEFAULT_WEP_KEY_INDEX Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_DEFAULT_WEP_KEY_INDEX 0
```

Description

see DRV_WIFI_SecurityWepSet() and DRV_WIFI_WEP_CONTEXT

DRV_WIFI_SOFTAP_NETWORK_CONTEXT Structure

Contains data pertaining to Wi-Fi Soft AP context.

File

drv_wifi.h

C

```
typedef struct {
  bool hiddenSsid;
} DRV_WIFI_SOFTAP_NETWORK_CONTEXT;
```

Members

Members	Description
bool hiddenSsid;	When starting an Soft AP network, the SSID can be hidden in the beacons. Set true to hide the SSID, else false. Default is false.

Description

This structure contains MRF24WG Soft AP context data. See DRV_WIFI_SoftAPContextSet.

DRV_WIFI_DEFAULT_SOFTAP_HIDDEN_SSID Macro

Default values for Wi-Fi Soft AP settings.

File

drv_wifi.h

C

```
#define DRV_WIFI_DEFAULT_SOFTAP_HIDDEN_SSID false
```

Description

Wi-Fi Soft AP default settings.

These defines identify various default Wi-Fi Soft AP settings that can be used in the DRV_WIFI_SOFTAP_NETWORK_CONTEXT structure.

DRV_WIFI_MULTICAST_CONFIG Structure

Contains data pertaining to Wi-Fi software multicast filter configuration.

File

drv_wifi.h

C

```
typedef struct {
  uint8_t filterId;
  uint8_t action;
  uint8_t macBytes[6];
  uint8_t macBitMask;
} DRV_WIFI_MULTICAST_CONFIG;
```

Members

Members	Description
uint8_t filterId;	DRV_WIFI_MULTICAST_FILTER_1 through DRV_WIFI_MULTICAST_FILTER_16
uint8_t action;	configures the multicast filter (see description)
uint8_t macBytes[6];	Array containing the MAC address to filter on (using the destination address of each incoming 802.11 frame). Specific bytes within the MAC address can be designated as "don't care" bytes. See macBitMask. This field in only used if action = WF_MULTICAST_USE_FILTERS.
uint8_t macBitMask;	A byte where bits 5:0 correspond to macBytes[5:0]. If the bit is zero then the corresponding MAC byte must be an exact match for the frame to be forwarded to the Host PIC. If the bit is one then the corresponding MAC byte is a "don't care" and not used in the Multicast filtering process. This field in only used if action = WF_MULTICAST_USE_FILTERS.

Description

This structure contains data pertaining to the configuration of the software multicast config filter.

'action' Field	Description
DRV_WIFI_MULTICAST_DISABLE_ALL	Multicast filter discards all received multicast messages.
DRV_WIFI_MULTICAST_ENABLE_ALL	Multicast filter forwards all received multicast messages to host.
DRV_WIFI_MULTICAST_USE_FILTERS	The MAC filter will be used and the remaining fields define the filter.

DRV_WIFI_APCONFIG_BIT_PREAMBLE_LONG Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_APCONFIG_BIT_PREAMBLE_LONG (0x20)
```

Description

This is macro DRV_WIFI_APCONFIG_BIT_PREAMBLE_LONG.

DRV_WIFI_APCONFIG_BIT_PRIVACY Macro

apConfig bit

File

drv_wifi.h

C

```
#define DRV_WIFI_APCONFIG_BIT_PRIVACY (0x10)
```

Description

apConfig bit

DRV_WIFI_APCONFIG_BIT_WPA Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_APCONFIG_BIT_WPA (0x40)
```

Description

This is macro DRV_WIFI_APCONFIG_BIT_WPA.

DRV_WIFI_APCONFIG_BIT_WPA2 Macro

File

drv_wifi.h

C

```
#define DRV_WIFI_APCONFIG_BIT_WPA2 (0x80)
```

Description

This is macro DRV_WIFI_APCONFIG_BIT_WPA2.

DRV_WIFI_DEVICE_TYPE Enumeration

Codes for MRF Wi-Fi Device Type

File

drv_wifi.h

C

```
typedef enum {
  DRV_WIFI_MRF24WB0M_DEVICE = 1,
  DRV_WIFI_MRF24WG0M_DEVICE = 2
} DRV_WIFI_DEVICE_TYPE;
```

Description

MRF Wi-Fi devices type.

This enumeration identifies MRF Wi-Fi device type. The only device supported with this driver is DRV_WIFI_MRF24WG0M_DEVICE.

Files

Files

Name	Description
drv_wifi.h	MRF24WG Wi-Fi Driver Interface File
drv_wifi_iwpriv.h	Configure optional (private) parameters of MRF24WG driver.

Description

This section lists the source and header files used by the MRF24W Wi-Fi Driver Library.

drv_wifi.h

MRF24WG Wi-Fi Driver Interface File

Enumerations

	Name	Description
	adhocMode	Selection of different Ad-Hoc connection modes.
	DRV_WIFI_ADHOC_MODES	Selection of different Ad-Hoc connection modes.
	DRV_WIFI_CONNECTION_STATES	Wi-Fi Connection States
	DRV_WIFI_DEVICE_TYPE	Codes for MRF Wi-Fi Device Type
	DRV_WIFI_DOMAIN_CODES	Wi-Fi Regional Domain Codes
	DRV_WIFI_EVENT_CONN_TEMP_LOST_CODES	Selection of different codes when Wi-Fi connection is temporarily lost.
	DRV_WIFI_EVENT_INFO	Selection of different EventInfo types.
	DRV_WIFI_EVENTS	Selections for events that can occur.
	DRV_WIFI_GENERAL_ERRORS	This is type DRV_WIFI_GENERAL_ERRORS.
	DRV_WIFI_HIBERNATE_STATES	Wi-Fi Hibernate States
	DRV_WIFI_MGMT_ERRORS	Error codes returned when a management message is sent to the MRF24WG module.
	DRV_WIFI_MULTICAST_FILTER_IDS	Selections for software Multicast filter IDs.
	DRV_WIFI_MULTICAST_FILTERS	Selections for Software Multicast Filters.
	DRV_WIFI_POWER_SAVE_STATES	Wi-Fi Power-Saving States
	DRV_WIFI_REASON_CODES	Selection of different codes when a deauthorization or disassociation event has occurred.
	DRV_WIFI_RECONNECT_MODES	Selection of different reconnection modes.
	DRV_WIFI_SCAN_TYPES	Selection of different Wi-Fi scan types.
	DRV_WIFI_SOFT_AP_EVENT_REASON_CODES	Wi-Fi Soft AP Event Reason Codes
	DRV_WIFI_SOFT_AP_STATES	Wi-Fi Soft AP Events
	DRV_WIFI_STATUS_CODES	Selection of different codes when Wi-Fi connection fails due to association or authentication failure.
	DRV_WIFI_TX_MODES	Selections for Wi-Fi TX Mode
	DRV_WIFI_WEP_KEY_TYPE	Selections for WEP key type when using WEP security.
	DRV_WIFI_WPS_AUTH_TYPES	Selection of WPS Authorization Types
	DRV_WIFI_WPS_ENCODE_TYPES	Selection of WPS Encoding Types
	DRV_WIFI_WPS_ERROR_CONFIG_CODES	Selection of different codes when a WPS connection fails.
	DRV_WIFI_WPS_STATE_CODES	Selection of different codes when a Extensible Authentication Protocol is used.

Functions

	Name	Description
	DRV_WIFI_AdhocContextSet	Sets the Ad-Hoc context. **Implementation:** Dynamic
	DRV_WIFI_BssidGet	Gets the BSSID set in DRV_WIFI_BssidSet(). **Implementation:** Dynamic
	DRV_WIFI_BssidSet	Sets the Basic Service Set Identifier (BSSID). **Implementation:** Dynamic
	DRV_WIFI_ChannelListGet	Gets the channel list. **Implementation:** Dynamic
	DRV_WIFI_ChannelListSet	Sets the channel list. **Implementation:** Dynamic
	DRV_WIFI_ConfigDataDelete	Erases configuration data from the board EEPROM. **Implementation:** Dynamic
	DRV_WIFI_ConfigDataLoad	Loads configuration data from the board EEPROM. **Implementation:** Dynamic
	DRV_WIFI_ConfigDataSave	Save configuration data to the board EEPROM. **Implementation:** Dynamic
	DRV_WIFI_Connect	Directs the MRF24WG to connect to a Wi-Fi network. **Implementation:** Dynamic
	DRV_WIFI_ConnectContextGet	Gets the current Wi-Fi connection context. **Implementation:** Dynamic
	DRV_WIFI_ConnectionStateGet	Gets the current Wi-Fi connection state. **Implementation:** Dynamic
	DRV_WIFI_ContextLoad	Loads Wi-Fi context to MRF24WG. **Implementation:** Dynamic
	DRV_WIFI_Deferred_ISR	Implements MRF24WG Wi-Fi driver deferred ISR. **Implementation:** Dynamic
	DRV_WIFI_DeferredISR_SemGive	Gives semaphore to Wi-Fi deferred ISR. **Implementation:** Dynamic
	DRV_WIFI_Deinitialize	De-initializes the MRF24WG Wi-Fi driver. **Implementation:** Dynamic
	DRV_WIFI_DeviceInfoGet	Retrieves MRF24WG device information. **Implementation:** Dynamic
	DRV_WIFI_Disconnect	Directs the MRF24WG to disconnect from a Wi-Fi network. **Implementation:** Dynamic
	DRV_WIFI_HibernateEnable	Puts the MRF24WG module into hibernate mode. **Implementation:** Dynamic
	DRV_WIFI_HibernateModeClear	Clears current Hibernate mode. **Implementation:** Dynamic
	DRV_WIFI_InHibernateMode	Checks if MRF24WG is in hibernate mode. **Implementation:** Dynamic
	DRV_WIFI_Initialize	Initializes the MRF24WG Wi-Fi driver. **Implementation:** Dynamic
	DRV_WIFI_InitTask	Implements MRF24WG Wi-Fi driver initialization RTOS task. **Implementation:** Dynamic
	DRV_WIFI_INT_Handle	MRF24WG Wi-Fi driver interrupt handle. **Implementation:** Dynamic
	DRV_WIFI_LinkDownThresholdSet	Sets number of consecutive Wi-Fi TX failures before link is considered down. **Implementation:** Dynamic
	DRV_WIFI_MacAddressGet	Retrieves the MRF24WG MAC address. **Implementation:** Dynamic
	DRV_WIFI_MacAddressSet	Uses a different MAC address for the MRF24WG. **Implementation:** Dynamic
	DRV_WIFI_MacStatsGet	Gets MAC statistics. **Implementation:** Dynamic

	DRV_WIFI_MACTask	Implements MRF24WG Wi-Fi driver MAC process RTOS task. **Implementation:** Dynamic
	DRV_WIFI_MRF24W_ISR	MRF24WG Wi-Fi driver interrupt service routine. **Implementation:** Dynamic
	DRV_WIFI_MulticastFilterSet	Sets a multicast address filter using one of the software multicast filters. **Implementation:** Dynamic
	DRV_WIFI_NetworkTypeGet	Gets the Wi-Fi network type. **Implementation:** Dynamic
	DRV_WIFI_NetworkTypeSet	Sets the Wi-Fi network type. **Implementation:** Dynamic
	DRV_WIFI_PowerSaveStateGet	Gets the current power-saving state.
	DRV_WIFI_ProcessEvent	Processes Wi-Fi event. **Implementation:** Dynamic
	DRV_WIFI_PsPollDisable	Disables PS-Poll mode. **Implementation:** Dynamic
	DRV_WIFI_PsPollEnable	Enables PS Poll mode. **Implementation:** Dynamic
	DRV_WIFI_ReconnectModeGet	Gets the Wi-Fi reconnection mode. **Implementation:** Dynamic
	DRV_WIFI_ReconnectModeSet	Sets the Wi-Fi reconnection mode. **Implementation:** Dynamic
	DRV_WIFI_RegionalDomainGet	Retrieves the MRF24WG Regional domain. **Implementation:** Dynamic
	DRV_WIFI_RSSI_Cache_FromRxDataRead	Caches RSSI value from RX data packet. **Implementation:** Dynamic
	DRV_WIFI_RSSI_Get_FromRxDataRead	Reads RSSI value from RX data packet. **Implementation:** Dynamic
	DRV_WIFI_RssiGet	Gets RSSI value set in DRV_WIFI_RssiSet(). **Implementation:** Dynamic
	DRV_WIFI_RssiSet	Sets RSSI restrictions when connecting. **Implementation:** Dynamic
	DRV_WIFI_RtsThresholdGet	Gets the RTS Threshold. **Implementation:** Dynamic
	DRV_WIFI_RtsThresholdSet	Sets the RTS Threshold. **Implementation:** Dynamic
	DRV_WIFI_Scan	Commands the MRF24WG module to start a scan operation. This will generate the WF_EVENT_SCAN_RESULTS_READY event. **Implementation:** Dynamic
	DRV_WIFI_ScanContextGet	Gets the Wi-Fi scan context. **Implementation:** Dynamic
	DRV_WIFI_ScanContextSet	Sets the Wi-Fi scan context. **Implementation:** Dynamic
	DRV_WIFI_ScanResultGet	Read selected scan results back from MRF24WG. **Implementation:** Dynamic
	DRV_WIFI_SecurityGet	Gets the current Wi-Fi security setting. **Implementation:** Dynamic
	DRV_WIFI_SecurityOpenSet	Sets Wi-Fi security to open (no security). **Implementation:** Dynamic
	DRV_WIFI_SecurityTypeGet	Gets the current Wi-Fi security type. **Implementation:** Dynamic
	DRV_WIFI_SecurityWepSet	Sets Wi-Fi security to use WEP. **Implementation:** Dynamic
	DRV_WIFI_SecurityWpaSet	Sets Wi-Fi security to use WPA or WPA2. **Implementation:** Dynamic
	DRV_WIFI_SecurityWpsSet	Sets Wi-Fi security to use WPS. **Implementation:** Dynamic
	DRV_WIFI_SetPSK	Sets the binary WPA PSK code in WPS. **Implementation:** Dynamic

	DRV_WIFI_SoftAPContextSet	Sets the Soft AP context. **Implementation:** Dynamic
	DRV_WIFI_SoftApEventInfoGet	Gets the stored Soft AP event info. **Implementation:** Dynamic
	DRV_WIFI_SpiClose	Closes SPI object for MRF24WG Wi-Fi driver. **Implementation:** Dynamic
	DRV_WIFI_SpiDmaRx	SPI RX API using DMA. **Implementation:** Dynamic
	DRV_WIFI_SpiDmaTx	SPI TX API using DMA. **Implementation:** Dynamic
	DRV_WIFI_SpiInit	Initializes SPI object for MRF24WG Wi-Fi driver. **Implementation:** Dynamic
	DRV_WIFI_SpiRx	SPI Rx API. **Implementation:** Dynamic
	DRV_WIFI_SpiTx	SPI TX API **Implementation:** Dynamic
	DRV_WIFI_SsidGet	Gets the SSID. **Implementation:** Dynamic
	DRV_WIFI_SsidSet	Sets the SSID. **Implementation:** Dynamic
	DRV_WIFI_TaskSyncInit	Initializes RTOS Semaphore and Mutex for MRF24WG Wi-Fi driver. **Implementation:** Dynamic
	DRV_WIFI_TxModeGet	Gets 802.11 TX mode. **Implementation:** Dynamic
	DRV_WIFI_TxModeSet	Configures 802.11 TX mode. **Implementation:** Dynamic
	DRV_WIFI_TxPowerFactoryMaxGet	Retrieves the factory-set max TX power from the MRF24WG module. **Implementation:** Dynamic
	DRV_WIFI_TxPowerMaxGet	Gets the TX max power on the MRF24WG0M. **Implementation:** Dynamic
	DRV_WIFI_TxPowerMaxSet	Sets the TX max power on the MRF24WG0M. **Implementation:** Dynamic
	DRV_WIFI_WepKeyTypeGet	Gets the WEP Key type. **Implementation:** Dynamic
	DRV_WIFI_WPSCredentialsGet	Gets the WPS credentials. **Implementation:** Dynamic

Macros

	Name	Description
	DRV_WIFI_APCONFIG_BIT_PREAMBLE_LONG	This is macro DRV_WIFI_APCONFIG_BIT_PREAMBLE_LONG.
	DRV_WIFI_APCONFIG_BIT_PRIVACY	apConfig bit
	DRV_WIFI_APCONFIG_BIT_WPA	This is macro DRV_WIFI_APCONFIG_BIT_WPA.
	DRV_WIFI_APCONFIG_BIT_WPA2	This is macro DRV_WIFI_APCONFIG_BIT_WPA2.
	DRV_WIFI_BSSID_LENGTH	This is macro DRV_WIFI_BSSID_LENGTH.
	DRV_WIFI_DEAUTH_REASONCODE_MASK	This is macro DRV_WIFI_DEAUTH_REASONCODE_MASK.
	DRV_WIFI_DEFAULT_ADHOC_BEACON_PERIOD	ms
	DRV_WIFI_DEFAULT_ADHOC_HIDDEN_SSID	Default values for Wi-Fi Ad-Hoc settings.
	DRV_WIFI_DEFAULT_ADHOC_MODE	This is macro DRV_WIFI_DEFAULT_ADHOC_MODE.
	DRV_WIFI_DEFAULT_PS_DTIM_ENABLED	DTIM wake-up enabled (normally the case)
	DRV_WIFI_DEFAULT_PS_DTIM_INTERVAL	number of beacon periods
	DRV_WIFI_DEFAULT_PS_LISTEN_INTERVAL	100 ms multiplier, e.g., 1 * 100 ms = 100 ms
	DRV_WIFI_DEFAULT_SCAN_COUNT	Default values for Wi-Fi scan context
	DRV_WIFI_DEFAULT_SCAN_MAX_CHANNEL_TIME	ms
	DRV_WIFI_DEFAULT_SCAN_MIN_CHANNEL_TIME	ms
	DRV_WIFI_DEFAULT_SCAN_PROBE_DELAY	us
	DRV_WIFI_DEFAULT_SOFTAP_HIDDEN_SSID	Default values for Wi-Fi Soft AP settings.

	DRV_WIFI_DEFAULT_WEP_KEY_INDEX	see DRV_WIFI_SecurityWepSet() and DRV_WIFI_WEP_CONTEXT
	DRV_WIFI_DEFAULT_WEP_KEY_TYPE	This is macro DRV_WIFI_DEFAULT_WEP_KEY_TYPE.
	DRV_WIFI_DISABLED	This is macro DRV_WIFI_DISABLED.
	DRV_WIFI_DISASSOC_REASONCODE_MASK	This is macro DRV_WIFI_DISASSOC_REASONCODE_MASK.
	DRV_WIFI_ENABLED	Do not make this an enumerated type!
	DRV_WIFI_MAX_CHANNEL_LIST_LENGTH	This is macro DRV_WIFI_MAX_CHANNEL_LIST_LENGTH.
	DRV_WIFI_MAX_NUM_RATES	This is macro DRV_WIFI_MAX_NUM_RATES.
	DRV_WIFI_MAX_SECURITY_KEY_LENGTH	This is macro DRV_WIFI_MAX_SECURITY_KEY_LENGTH.
	DRV_WIFI_MAX_SSID_LENGTH	This is macro DRV_WIFI_MAX_SSID_LENGTH.
	DRV_WIFI_MAX_WEP_KEY_LENGTH	This is macro DRV_WIFI_MAX_WEP_KEY_LENGTH.
	DRV_WIFI_MAX_WPA_PASS_PHRASE_LENGTH	must exclude string terminator
	DRV_WIFI_MIN_WPA_PASS_PHRASE_LENGTH	must exclude string terminator
	DRV_WIFI_NETWORK_TYPE_ADHOC	This is macro DRV_WIFI_NETWORK_TYPE_ADHOC.
	DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE	Selection of different Wi-Fi network types.
	DRV_WIFI_NETWORK_TYPE_SOFT_AP	This is macro DRV_WIFI_NETWORK_TYPE_SOFT_AP.
	DRV_WIFI_NO_ADDITIONAL_INFO	eventInfo define for DRV_WIFI_ProcessEvent() when no additional info is supplied
	DRV_WIFI_RETRY_ADHOC	This is macro DRV_WIFI_RETRY_ADHOC.
	DRV_WIFI_RETRY_FOREVER	This is macro DRV_WIFI_RETRY_FOREVER.
	DRV_WIFI_RTS_THRESHOLD_MAX	maximum RTS threshold size in bytes
	DRV_WIFI_SECURITY_OPEN	Selection of different Wi-Fi security types
	DRV_WIFI_SECURITY_WEP_104	This is macro DRV_WIFI_SECURITY_WEP_104.
	DRV_WIFI_SECURITY_WEP_40	This is macro DRV_WIFI_SECURITY_WEP_40.
	DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY	This is macro DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY.
	DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE	This is macro DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE.
	DRV_WIFI_SECURITY_WPA_WITH_KEY	This is macro DRV_WIFI_SECURITY_WPA_WITH_KEY.
	DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE	This is macro DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE.
	DRV_WIFI_SECURITY_WPA2_WITH_KEY	This is macro DRV_WIFI_SECURITY_WPA2_WITH_KEY.
	DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE	This is macro DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE.
	DRV_WIFI_SECURITY_WPS_PIN	This is macro DRV_WIFI_SECURITY_WPS_PIN.
	DRV_WIFI_SECURITY_WPS_PUSH_BUTTON	This is macro DRV_WIFI_SECURITY_WPS_PUSH_BUTTON.
	DRV_WIFI_WEP104_KEY_LENGTH	4 keys of 13 bytes each
	DRV_WIFI_WEP40_KEY_LENGTH	4 keys of 5 bytes each
	DRV_WIFI_WPA_KEY_LENGTH	This is macro DRV_WIFI_WPA_KEY_LENGTH.
	DRV_WIFI_WPS_PIN_LENGTH	7 digits + checksum byte

Structures

	Name	Description
	DRV_WIFI_ADHOC_NETWORK_CONTEXT	Contains data pertaining to Wi-Fi Ad-Hoc context.
	DRV_WIFI_CONNECTION_CONTEXT	Contains data pertaining to MRF24WG connection context.
	DRV_WIFI_DEVICE_INFO	Contains data pertaining to MRF24WG device type and version number.
	DRV_WIFI_MAC_STATS	Wi-Fi MIB states
	DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT	Contains data pertaining to Wi-Fi Soft AP event.
	DRV_WIFI_MULTICAST_CONFIG	Contains data pertaining to Wi-Fi software multicast filter configuration.
	DRV_WIFI_PS_POLL_CONTEXT	Contains data pertaining to Wi-Fi PS-Poll context.
	DRV_WIFI_SCAN_CONTEXT	Contains data pertaining to Wi-Fi scan context.
	DRV_WIFI_SCAN_RESULT	Contains data pertaining to Wi-Fi scan results.
	DRV_WIFI_SOFTAP_NETWORK_CONTEXT	Contains data pertaining to Wi-Fi Soft AP context.
	DRV_WIFI_WEP_CONTEXT	Contains data pertaining to Wi-Fi WEP context.
	DRV_WIFI_WPA_CONTEXT	Contains data pertaining to Wi-Fi WPA.
	DRV_WIFI_WPA_KEY_INFO	Contains data pertaining to Wi-Fi WPA Key.
	DRV_WIFI_WPS_CONTEXT	Contains data pertaining to Wi-Fi WPS security.

	DRV_WIFI_WPS_CREDENTIAL	Contains data pertaining to Wi-Fi WPS Credentials.

Description

Contains all data types, define constants, and function prototypes for interfacing to the MRF24WG Wi-Fi driver.

File Name

drv_wifi.h

Company

Microchip Technology Inc.

drv_wifi_iwpriv.h

Configure optional (private) parameters of MRF24WG driver.

Functions

	Name	Description
	iwpriv_adhocctx_set	Sets the Ad-Hoc network context information. **Implementation:** Dynamic
	iwpriv_config_read	Reads the Wi-Fi context configuration. **Implementation:** Dynamic

Description

MRF24WG Private Configuration Support

Functions in this module support the connection process for the MRF24WG.

File Name

drv_wifi_iwpriv.h

Company

Microchip Technology Inc.

MRF24WN Wi-Fi Driver Library

This topic describes the MRF24WN Wi-Fi Driver Library.

Description

The following table lists the library files available for the MRF24WN Wi-Fi Driver.

	Target MCU Device		
Wi-Fi Library File Matrix	PIC32MX795F512L	PIC32MZ2048ECH144	PIC32MZ2048EFM144
Wi-Fi Device MRF24WN	wdrvext_mx.a	wdrvext_mz_ec.a	wdrvext_mz_ef.a

Introduction

This library provides a low-level abstraction of the MRF24WN Wi-Fi Driver Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The MRF24WN Wi-Fi Driver Library, in conjunction with the MRF24WN module, allows an application to:

- Join an existing 802.11 Wi-Fi Infrastructure network
- Create a 802.11 Wi-Fi Ad Hoc or Soft AP network

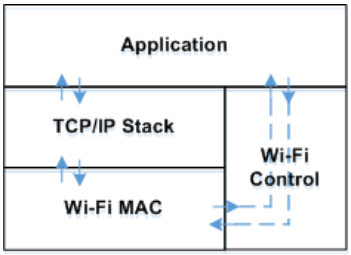
The following application services are provided by the Wi-Fi library:

- Configuring Wi-Fi connection (SSID, security mode, channel list, etc.)
- Join an existing Wi-Fi Infrastructure network
- Create a Wi-Fi Ad Hoc or Soft AP network
- Scan for Wi-Fi Access Point (AP) or Soft AP
- Getting Wi-Fi network status
- Wi-Fi power control

- Wi-Fi console commands

The MAC layer services are not directly accessible to the application; this portion of the code resides under the TCP/IP Stack MAC module software layers and is used by stack services to transmit and receive data over a Wi-Fi network. The following diagram shows the interaction of the primary software blocks in a Wi-Fi application.

Wi-Fi Software Block Diagram



The following table provides information that includes network mode and security mode support by MRF24WN Wi-Fi Driver.

MRF24WN Network Connection Matrix	Network Mode	
	Infrastructure	Soft AP
Security Mode		
Open	YES	YES
WEP40	YES	YES
WEP104	YES	YES
WPA-PSK	YES	YES
WPA2-PSK	YES	YES
WPS Push Button	YES	NA
WPS PIN	YES	NA

Using the Library

This topic describes the basic architecture of the MRF24WN Wi-Fi Driver Library and provides information and examples on its use.

Description

Interface Header Files: `wdrv_mrf24wn_common.h` and `wdrv_mrf24wn_api.h`

The interface to the MRF24WN Wi-Fi Driver Library is defined in the `wdrv_mrf24wn_common.h` and wdrv_mrf24wn_api.h header files.

Please refer to the Understanding MPLAB Harmony section for how the driver interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the MRF24WN Wi-Fi module with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The MRF24WN Wi-Fi Library provides the following functionality:

- Wi-Fi library initialization
- Wi-Fi network configuration
- Wi-Fi network connection
- Scanning for existing Wi-Fi networks
- Wi-Fi event processing
- Wi-Fi status
- Wi-FI console commands

Library Overview

Refer to the Driver Library Overview section for information on how the driver operates in a system.

The Library Interface functions are divided into various sub-sections, which address one of the blocks or the overall operation of the Wi-Fi module.

Library Interface Section	Description
Wi-Fi Initialization Functions	This section provides functions that initialize the Wi-Fi library and allow its API to be used.

Wi-Fi Status Functions	This section provides functions that retrieve the Wi-Fi connection status.
Wi-Fi External Functions	This section provides public functions accessible to TCP/IP applications.
Other Functions	This section provides additional miscellaneous functions for configuring the Wi-Fi connection.

How the Library Works

This section describes how the MRF24WN Wi-Fi Driver Library operates.

Description

Before the driver is ready for use, its should be configured (compile time configuration).

There are few run-time configuration items that are done during initialization of the driver instance, and a few that are client-specific and are done using dedicated functions.

To use the MRF24WN Wi-Fi Driver, initialization and client functions should be invoked in a specific sequence to ensure correct operation.

System Initialization

This section describes initialization and reinitialization features.

Description

Wi-Fi initialization configures the MRF24WN module and then directs it to join (or create) a Wi-Fi network. The MRF24WN module defaults to open security and scans all channels in the domain. Therefore, to initialize and connect with the minimum function call overhead in an open security network, the following functions can be used:

```
WDRV_EXT_CmdSSIDSet("MySsidName",strlen("MySsidName");
WDRV_EXT_CmdConnect();        // start the connection process
```

Alternatively, the following functions could be used to achieve the same effect:

```
WDRV_EXT_CmdNetModeBSSSet();
WDRV_EXT_CmdSecNoneSet();
WDRV_EXT_CmdSSIDSet("MySsidName",strlen("MySsidName");
WDRV_EXT_CmdConnect();
```

Client Functionality

This section describes core operation.

Description

From the client perspective, once Wi-Fi initialization is complete and the connection process has started, the client responds to Wi-Fi events. The client is notified of events by the callback function WDRV_ProcessEvent. The parameters into that function are `event` and `eventInfo`, where `event` is the event code and `eventInfo` is additional information about the event.

Wi-Fi Connection Events

```
/*No Wi-Fi connection exists*/
WDRV_CSTATE_NOT_CONNECTED = 1,

/*Wi-Fi connection in progress*/
WDRV_CSTATE_CONNECTION_IN_PROGRESS = 2,

/*Wi-Fi connected in infrastructure mode*/
WDRV_CSTATE_CONNECTED_INFRASTRUCTURE = 3,

/*Wi-Fi connected in adHoc mode*/
WDRV_CSTATE_CONNECTED_ADHOC = 4,

/*Wi-Fi in process of reconnecting*/
WDRV_CSTATE_RECONNECTION_IN_PROGRESS = 5,

/*Wi-Fi connection temporarily lost*/
WDRV_CSTATE_CONNECTION_TEMPORARY_LOST = 6,
```

```
/*Wi-Fi connection permanently lost*/
WDRV_CSTATE_CONNECTION_PERMANENTLY_LOST = 7
```

Scan Events

```
WDRV_SOFTAP_EVENT_CONNECTED = 0,
WDRV_SOFTAP_EVENT_DISCONNECTED = 1
```

Key Events

```
WDRV_SOFTAP_EVENT_LINK_LOST = 0,
WDRV_SOFTAP_EVENT_RECEIVED_DEAUTH = 1
```

Disconnect Events

```
WDRV_DISCONNECT_REASON_NO_NETWORK_AVAIL = 0x01,
WDRV_DISCONNECT_REASON_LOST_LINK = 0x02,
WDRV_DISCONNECT_REASON_DISCONNECT_CMD = 0x03,
WDRV_DISCONNECT_REASON_BSS_DISCONNECTED = 0x04,
WDRV_DISCONNECT_REASON_AUTH_FAILED = 0x05,
WDRV_DISCONNECT_REASON_ASSOC_FAILED = 0x06,
WDRV_DISCONNECT_REASON_NO_RESOURCES_AVAIL = 0x07,
WDRV_DISCONNECT_REASON_CONNECTION_DENIED = 0x08,
WDRV_DISCONNECT_REASON_INVALID_PROFILE = 0x0A,
WDRV_DISCONNECT_REASON_PROFILE_MISMATCH = 0x0C,
WDRV_DISCONNECT_REASON_CONNECTION_EVICTED = 0x0d
```

Configuring the Library

The configuration of the MRF24WN Wi-Fi Driver is based on the file `system_config.h`.

This header file contains the configuration selection for the Wi-Fi Driver. Based on the selections made, the MRF24WN Wi-Fi Driver may support the selected features. These configuration settings will apply to all instances of the MRF24WN Wi-Fi Driver.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Configuring DMA SPI

Please refer to Configuring DMA SPI for information on enabling and disabling DMA SPI for FreeRTOS projects.

Sample Functionality

The following code provides an example of Wi-Fi Driver configuration.

```
/*** Wi-Fi Driver Configuration ***/

#define WIFI_USE_RTOS

#define WDRV_EXT_INIT_TASK_STACK_SIZE 512u
#define WDRV_EXT_INIT_TASK_PRIO 6u
#define WDRV_EXT_MAIN_TASK_STACK_SIZE 2048u
#define WDRV_EXT_MAIN_TASK_PRIO 7u

#define WDRV_ASSERT(condition, msg) WDRV_Assert(condition, msg, __FILE__, __LINE__)

#define DRV_WIFI_SPI_INDEX 0
#define DRV_WIFI_SPI_INSTANCE sysObj.spiObjectIdx0

#define DRV_WIFI_NVM_SPACE_ENABLE
#define DRV_WIFI_NVM_SPACE_ADDR (48*1024)

#define MRF_INT_SOURCE INT_SOURCE_EXTERNAL_1
#define MRF_INT_VECTOR INT_VECTOR_INT1

// IO mapping for general control pins, including CS, RESET and HIBERNATE
// MRF24W in SPI 1 slot
#define WF_CS_PORT_CHANNEL PORT_CHANNEL_E
#define WF_CS_BIT_POS      9
```

```
#define WF_RESET_PORT_CHANNEL PORT_CHANNEL_F
#define WF_RESET_BIT_POS      0

#define WF_HIBERNATE_PORT_CHANNEL PORT_CHANNEL_F
#define WF_HIBERNATE_BIT_POS      1

#define WF_INT_PRIORITY     3
#define WF_INT_SUBPRIORITY  1
#define WF_INT_PORT_CHANNEL PORT_CHANNEL_E
#define WF_INT_BIT_POS      8

#define WDRV_DEFAULT_NETWORK_TYPE WDRV_NETWORK_TYPE_INFRASTRUCTURE
#define WDRV_DEFAULT_SSID_NAME "MicrochipDemoApp"

#define WDRV_DEFAULT_WIFI_SECURITY_MODE WDRV_SECURITY_OPEN
#define WDRV_DEFAULT_WEP_KEYS_40 "5AFB6C8E77" // default WEP40 key
#define WDRV_DEFAULT_WEP_KEYS_104 "90E96780C739409DA50034FCAA" // default WEP104 key
#define WDRV_DEFAULT_PSK_PHRASE "Microchip 802.11 Secret PSK Password" // default WPA-PSK or WPA2-PSK
passphrase
#define WDRV_DEFAULT_WPS_PIN "12390212" // default WPS PIN

#define WDRV_DEFAULT_CHANNEL 6
#define WDRV_DEFAULT_POWER_SAVE WDRV_FUNC_DISABLED
```

Building the Library

This section lists the files that are available in the MRF24WN Wi-Fi Driver Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/driver/wifi/mrf24wn`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`wdrv_mrf24wn_common.h`	Contains all data types, define constants for the MRF24WN Wi-Fi Driver.
`wdrv_mrf24wn_api.h`	Contains function prototypes for interfacing to the MRF24WN Wi-Fi Driver.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`wdrv_mrf24wn_cli.c`	Provides access to MRF24WN Wi-Fi Driver controller.
`wdrv_mrf24wn_config_data.c`	Stores and retrieves MRF24WN Wi-Fi Driver configuration information in Non-volatile Memory (NVM).
`wdrv_mrf24wn_connmgr.c`	Provides access to MRF24WN Wi-Fi Driver controller for connection manager.
`wdrv_mrf24wn_events.c`	Provides access to MRF24WN Wi-Fi Driver controller for MAC events.
`wdrv_mrf24wn_iwpriv.c`	Provides functions to configure optional (private) parameters of the MRF24WN Wi-Fi Driver.
`wdrv_mrf24wn_main.c`	Module for Microchip TCP/IP Stack PIC32 implementation for multiple Wi-Fi MAC support.
`wdrv_mrf24wn_misc.c`	Miscellaneous support functions and data types for the MRF24WN Wi-Fi Driver.
`wdrv_mrf24wn_osal.c`	RTOS wrapper functions for the MRF24WN Wi-Fi Driver.
`wdrv_mrf24wn_scan_helper.c`	Provides helper functions to access scan results.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	The MRF24WN Wi-Fi Driver controller has no optional files.

Module Dependencies

The MRF24WN Wi-Fi Driver Library depends on the following modules:

- SPI Driver Library
- NVM Driver Library
- UART Driver Library
- USB Driver Library
- Operating System Abstraction Layer (OSAL) Library Help
- Clock System Service Library
- System Service Library Introduction
- Console System Service Library
- File System Service Library
- Interrupt System Service Library
- Timer System Service Library
- Debug System Service Library
- Ports System Service Library
- FreeRTOS Library Help
- Crypto Library
- Peripheral Libraries

Console Commands

This section describes the console commands available for the MRF24WN Wi-Fi Driver.

Description

Both the Web Server and the EasyConfig demonstrations support the followings commands, which enable control over the Wi-Fi settings.

Command: eraseconf

Parameters	Description
None.	Wi-Fi console command to erase saved Wi-Fi configuration in memory.

Command: iwconfig

Parameters	Description
[ssid <name>]	name: Specifies the name of the SSID (1-32 ASCII characters).
[mode <idle | managed>]	idle: Disconnected from the current configuration. managed: Connects in infrastructure mode to the currently set SSID.
[power <enable | disable>]	enable: Enables all Power-Saving features (PS_POLL). Will wake up to check for all types of traffic (unicast, multicast, and broadcast). disable: Disables any Power-Saving features. Will always be in an active power state.

[security <mode>]	mode: open/wep40/wep104/wpa/wpa2/pin/pbc. For example:
	`iwconfig security open`
	`iwconfig security wep40 <key>`
	`iwconfig security wep104 <key>`
	`iwconfig security wpa <key>`
	`iwconfig security wpa2 <key>`
	`iwconfig security pin <pin>`
	`iwconfig security pbc`

[scan]	Starts a Wi-Fi scan.
[scanget <scan_index>]	scan_index: Retrieves the scan result after the scan completes (1 - n).

Command: mac

Parameters	Description
None.	Wi-Fi console command to retrieve the MAC address of the MRF24WN module.

Command: readconf

Parameters	Description
None.	Wi-Fi console command to read saved Wi-Fi configuration in memory.

Command: rftest

Parameters	Description
[rate < packet_count | packet_size | channel header_type >]	Wi-Fi console command to perform regulatory test. rate (in Mbps): • 0 = 1 • 1 = 2 • 2 = 5.5 • 3 = 11 • 4 = 6 • 5 = 9 • 6 = 12 • 7 = 18 • 8 = 24 • 9 = 36 • 10 = 48 • 11 = 54 • 12 = 6.5 • 13 = 13 • 14 = 19.5 • 15 = 26 • 16 = 39 • 17 = 52 • 18 = 58.5 packet_count: Number of transmits (1 through 14). packet_size: Payload size (0 to 1400). channel: 1 through 14. header_type: 0 - Beacon frame; 1 - QoS data frame; 2 through 4 Address data frame.

Command: saveconf

Parameters	Description
None.	Wi-Fi console command to save Wi-Fi configuration to memory.

Library Interface

a) Wi-Fi Initialization Functions

	Name	Description
	WDRV_CLI_Init	Initializes the console CLI interface. **Implementation:** Dynamic
	WDRV_INTR_Deinit	Deinitializes interrupts for Wi-Fi driver. **Implementation:** Dynamic
	WDRV_INTR_Init	Initializes interrupts for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_INTR_SourceDisable	Disables interrupts from the module. **Implementation:** Dynamic

	WDRV_INTR_SourceEnable	Enables interrupts from the module. **Implementation:** Dynamic
	WDRV_SPI_Deinit	Deinitializes the SPI object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_SPI_Init	Initializes the SPI object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_SPI_In	Receives data from the module through the SPI bus. **Implementation:** Dynamic
	WDRV_SPI_Out	Sends data out to the module through the SPI bus. **Implementation:** Dynamic
	WDRV_HibernateDisable	Wakes MRF24WN up from hibernate mode. **Implementation:** Dynamic
	WDRV_HibernateEnable	Hibernates MRF24WN module. **Implementation:** Dynamic
	WDRV_InHibernateMode	Checks if MRF24WN is in hibernate mode. **Implementation:** Dynamic

b) Wi-Fi Status Functions

	Name	Description
	WDRV_EXT_CmdConnectContextChannelGet	Gets the AP channel **Implementation:** Dynamic
	WDRV_EXT_CmdMacAddressGet	Retrieves the MRF24WN MAC address. **Implementation:** Dynamic
	WDRV_EXT_CmdScanGet	Reads the number of scan results from the MRF24WN module. **Implementation:** Dynamic
	WDRV_EXT_CmdSecGet	Gets the current Wi-Fi security type. **Implementation:** Dynamic
	WDRV_EXT_CmdSSIDGet	Gets the SSID. **Implementation:** Dynamic
	WDRV_EXT_CmdConnectContextBssidGet	Gets the BSSID. **Implementation:** Dynamic
	WDRV_EXT_CmdPowerSaveGet	Retrieves current power save status. **Implementation:** Dynamic
	WDRV_EXT_WPSResultsRead	Reads the WPS process results back from the MRF24WN module and updates the configuration data. **Implementation:** Dynamic

c) External Functions

	Name	Description
	WDRV_EXT_CmdConnect	Directs the MRF24WN to connect to a Wi-Fi network. **Implementation:** Dynamic
	WDRV_EXT_CmdDisconnect	Directs the MRF24WN to disconnect from a Wi-Fi network. **Implementation:** Dynamic
	WDRV_EXT_CmdNetModeAPSet	Sets the Wi-Fi network type to SoftAP. **Implementation:** Dynamic
	WDRV_EXT_CmdNetModeBSSSet	Sets the Wi-Fi network type to Infrastructure. **Implementation:** Dynamic
	WDRV_EXT_CmdNetModeIBSSSet	Sets the Wi-Fi network type to Adhoc. **Implementation:** Dynamic
	WDRV_EXT_CmdScanStart	Directs the MRF24WN module to start a scan. **Implementation:** Dynamic
	WDRV_EXT_CmdSecNoneSet	Sets Wi-Fi security to open (no security). **Implementation:** Dynamic
	WDRV_EXT_CmdSecWEPSet	Sets Wi-Fi security to use WEP. **Implementation:** Dynamic
	WDRV_EXT_CmdSecWPA2Set	Sets Wi-Fi security to use WPA2. **Implementation:** Dynamic

	WDRV_EXT_CmdSecWPASet	Sets Wi-Fi security to use WPA. **Implementation:** Dynamic
	WDRV_EXT_CmdSSIDSet	Sets the SSID. **Implementation:** Dynamic
	WDRV_EXT_DataSend	Sends data packets to MRF24WN module. **Implementation:** Dynamic
	WDRV_EXT_Deinitialize	Deinitializes the MRF24WN Wi-Fi driver. **Implementation:** Dynamic
	WDRV_EXT_HWInterruptHandler	Wi-Fi driver (MRF24WN-specific) interrupt service routine. **Implementation:** Dynamic
	WDRV_EXT_PowerUpDown	Powers the MRF24WN module up or down. **Implementation:** Dynamic
	WDRV_EXT_ScanDoneSet	Indicates when a scan has completed. **Implementation:** Dynamic
	WDRV_EXT_CmdChannelSet	Sets the channel on which to operate. **Implementation:** Dynamic
	WDRV_EXT_CmdFWUpdate	Directs the module to start firmware download and upgrade. **Implementation:** Dynamic
	WDRV_EXT_CmdSecWpsSet	Sets Wi-Fi security to use WPS. **Implementation:** Dynamic
	WDRV_EXT_CmdPowerSavePut	Puts the module in IEEE power save mode. **Implementation:** Dynamic
	WDRV_EXT_ScanIsInProgress	Check whether host scan is now in progress or not. **Implementation:** Dynamic
	WDRV_EXT_Misc_Config	Configures miscellaneous parameters. **Implementation:** Dynamic
	WDRV_EXT_CmdRFTest	Performs RF test for regulatory test purposes. **Implementation:** Dynamic
	WDRV_EXT_CmdTxPowerSet	Sets the TX Power. **Implementation:** Dynamic
	WDRV_EXT_ScanResultGet	Reads the selected scan results back from the MRF24WN module. **Implementation:** Dynamic
	WDRV_EXT_Initialize	Initializes the MRF24WN Wi-Fi driver. **Implementation:** Dynamic

d) GPIO Functions

	Name	Description
	WDRV_MRF24WN_ISR	Wi-Fi driver (MRF24WN-specific) interrupt service routine. **Implementation:** Dynamic
	WDRV_GPIO_MRF24WG_Disable	Disables the MRF24WG module on Multimedia Expansion Board II (MEB II). **Implementation:** Dynamic
	WDRV_GPIO_DeInit	Deinitializes the GPIO object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_GPIO_Init	Initializes the GPIO object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_GPIO_OutHigh	Pulls GPIO high. **Implementation:** Dynamic
	WDRV_GPIO_OutLow	Pulls gpio low. **Implementation:** Dynamic
	WDRV_GPIO_PowerDown	Powers down the MRF24WN module. **Implementation:** Dynamic
	WDRV_GPIO_PowerUp	Powers on the MRF24WN module. **Implementation:** Dynamic

e) Private Configuration Functions

	Name	Description
	iwpriv_adhocctx_set	Sets the Ad-Hoc network context information. **Implementation:** Dynamic

	iwpriv_config_read	Reads the Wi-Fi context configuration. **Implementation:** Dynamic
	iwpriv_config_write	Writes to the Wi-Fi context configuration which is currently used by Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_connstatus_get	Gets the Wi-Fi connection status. **Implementation:** Dynamic
	iwpriv_devinfo_get	Gets the device information. **Implementation:** Dynamic
	iwpriv_initialconn_set	Sets the initial connection status of Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_initstatus_get	Gets the initialization status of Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_is_servermode	Checks if the passed Wi-Fi context configuration is operating in server mode. **Implementation:** Dynamic
	iwpriv_leftclient_get	Gets the left client's information. **Implementation:** Dynamic
	iwpriv_mcastfilter_set	Adds a MAC address to the multi-cast filter. **Implementation:** Dynamic
	iwpriv_nettype_get	Gets the current network type. **Implementation:** Dynamic
	iwpriv_nettype_set	Sets the current network type. **Implementation:** Dynamic
	iwpriv_numberofscanresults_get	Gets the number of scan results. **Implementation:** Dynamic
	iwpriv_powersave_config	Enables or disables Power Save mode in Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_prescan_start	Starts prescan. **Implementation:** Dynamic
	iwpriv_scan_start	Starts scan. **Implementation:** Dynamic
	iwpriv_scanresults_display	Outputs all of the scan results. **Implementation:** Dynamic
	iwpriv_scanresult_get	Gets one scan result. **Implementation:** Dynamic
	iwpriv_scanresults_save	Saves all of the scan results. **Implementation:** Dynamic
	iwpriv_scanstatus_get	Gets the prescan status. **Implementation:** Dynamic
	iwpriv_ssid_get	Gets the current SSID. **Implementation:** Dynamic
	iwpriv_ssid_set	Sets the current SSID. **Implementation:** Dynamic
	iwpriv_execute	This is function iwpriv_execute.
	iwpriv_get	This is function iwpriv_get.
	iwpriv_prescan_isfinished	Checks if the prescan is complete. **Implementation:** Dynamic
	iwpriv_prescan_option_get	To see if prescan will run before next connection. **Implementation:** Dynamic
	iwpriv_prescan_option_set	To run prescan or not. **Implementation:** Dynamic
	iwpriv_set	This is function iwpriv_set.

f) Data Types and Constants

	Name	Description
	IWPRIV_CONN_STATUS	This is type IWPRIV_CONN_STATUS.
	IWPRIV_STATUS	This is type IWPRIV_STATUS.
	IWPRIV_CMD	This is type IWPRIV_CMD.
	IWPRIV_EXECUTE_PARAM	This is type IWPRIV_EXECUTE_PARAM.

	IWPRIV_GET_PARAM	This is type IWPRIV_GET_PARAM.
	IWPRIV_PARAM_CLIENTINFO	This is type IWPRIV_PARAM_CLIENTINFO.
	IWPRIV_PARAM_CONTEXT	This is type IWPRIV_PARAM_CONTEXT.
	IWPRIV_PARAM_DEVICEINFO	This is type IWPRIV_PARAM_DEVICEINFO.
	IWPRIV_SCAN_STATUS	This is type IWPRIV_SCAN_STATUS.
	IWPRIV_SET_PARAM	This is type IWPRIV_SET_PARAM.
	IWPRIV_PARAM_CONFIG	This is type IWPRIV_PARAM_CONFIG.
	IWPRIV_PARAM_CONNECT	This is type IWPRIV_PARAM_CONNECT.
	IWPRIV_PARAM_DRIVERSTATUS	This is type IWPRIV_PARAM_DRIVERSTATUS.
	IWPRIV_PARAM_MULTICASTFILTER	This is type IWPRIV_PARAM_MULTICASTFILTER.
	IWPRIV_PARAM_NETWORKTYPE	This is type IWPRIV_PARAM_NETWORKTYPE.
	IWPRIV_PARAM_OPERATIONMODE	This is type IWPRIV_PARAM_OPERATIONMODE.
	IWPRIV_PARAM_POWERSAVE	This is type IWPRIV_PARAM_POWERSAVE.
	IWPRIV_PARAM_SCAN	This is type IWPRIV_PARAM_SCAN.
	IWPRIV_PARAM_SSID	This is type IWPRIV_PARAM_SSID.

Description

This section describes the Application Programming Interface (API) functions of the MRF24WN Wi-Fi Driver.

Refer to each section for a detailed description.

a) Wi-Fi Initialization Functions

WDRV_CLI_Init Function

Initializes the console CLI interface.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
bool WDRV_CLI_Init();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function initializes the console CLI interface.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

bool WDRV_CLI_Init(void)

WDRV_INTR_Deinit Function

Deinitializes interrupts for Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_INTR_Deinit();
```

Returns

None.

Description

This function deinitializes interrupts for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_INTR_Deinit(void)

WDRV_INTR_Init Function

Initializes interrupts for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_INTR_Init();
```

Returns

None.

Description

This function initializes interrupts for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_INTR_Init(void)

WDRV_INTR_SourceDisable Function

Disables interrupts from the module.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_INTR_SourceDisable();
```

Returns

None.

Description

This function disables interrupts from the module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_INTR_SourceDisable(void)

WDRV_INTR_SourceEnable Function

Enables interrupts from the module.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_INTR_SourceEnable();
```

Returns

None.

Description

This function enables interrupts from the module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_INTR_SourceEnable(void)

WDRV_SPI_Deinit Function

Deinitializes the SPI object for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_SPI_Deinit();
```

Returns

None.

Description

This function deinitializes the SPI object for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_SPI_Deinit(void)

WDRV_SPI_Init Function

Initializes the SPI object for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_SPI_Init();
```

Returns

None.

Description

This function initializes the SPI object for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_SPI_Init(void)

WDRV_SPI_In Function

Receives data from the module through the SPI bus.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_SPI_In(uint8_t *const OutBuf, uint16_t OutSize, uint8_t *const InBuf, uint16_t InSize);
```

Returns

None.

Description

This function receives data from the module through the SPI bus.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Parameters

Parameters	Description
bufOut	buffer pointer of output command
OutSize	the command size
InBuf	buffer pointer of input data
InSize	the input data size

Function

void WDRV_SPI_In(uint8_t const *const OutBuf, uint16_t OutSize,
uint8_t *const InBuf, uint16_t InSize)

WDRV_SPI_Out Function

Sends data out to the module through the SPI bus.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_SPI_Out(uint8_t *const bufOut, uint16_t OutSize);
```

Returns

None.

Description

This function sends data out to the module through the SPI bus.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Parameters

Parameters	Description
bufOut	buffer pointer of output data
OutSize	the data size

Function

void WDRV_SPI_Out(uint8_t const *const bufOut, uint16_t OutSize)

WDRV_HibernateDisable Function

Wakes MRF24WN up from hibernate mode.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_HibernateDisable();
```

Returns

None.

Description

This function wakes MRF24WN up from hibernate mode.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_HibernateDisable(void)

WDRV_HibernateEnable Function

Hibernates MRF24WN module.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_HibernateEnable();
```

Returns

None.

Description

This function hibernates the MRF24WN module.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_HibernateEnable(void)

WDRV_InHibernateMode Function

Checks if MRF24WN is in hibernate mode.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
bool WDRV_InHibernateMode();
```

Returns

- 0 - Indicates that MRF24WN is in hibernate mode
- Non-zero value - Indicates that MRF24WN is not in hibernate mode

Description

This function checks if MRF24WN is in hibernate mode.

Remarks

None.

Function

bool WDRV_InHibernateMode(void)

b) Wi-Fi Status Functions

WDRV_EXT_CmdConnectContextChannelGet Function

Gets the AP channel

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdConnectContextChannelGet(uint16_t * bssChannel);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function gets the current AP channel.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
bssChannel	pointer where the current AP channel will be written

Function

uint32_t WDRV_EXT_CmdConnectContextChannelGet(uint16_t *bssChannel)

WDRV_EXT_CmdMacAddressGet Function

Retrieves the MRF24WN MAC address.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdMacAddressGet(uint8_t * MacAddr);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function retrieves the MRF24WN MAC address.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
MacAddr	Pointer where MAC address will be written (must point to a 6 bytes buffer)

Function

uint32_t WDRV_EXT_CmdMacAddressGet(uint8_t *MacAddr)

WDRV_EXT_CmdScanGet Function

Reads the number of scan results from the MRF24WN module.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdScanGet(uint16_t * numOfResults);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function reads the number of scan results from the MRF24WN module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
numOfResults	pointer where the number of scan results will be written

Function

uint32_t WDRV_EXT_CmdScanGet(uint16_t *numOfResults)

WDRV_EXT_CmdSecGet Function

Gets the current Wi-Fi security type.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdSecGet(uint32_t * SecurityType);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function gets the current Wi-Fi security type.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
SecurityType	pointer where the security type will be written

Function

uint32_t WDRV_EXT_CmdSecGet(uint32_t *SecurityType)

WDRV_EXT_CmdSSIDGet Function

Gets the SSID.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdSSIDGet(uint8_t * ssid, uint8_t * length);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function returns the SSID and SSID Length.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
ssid	pointer to buffer where SSID will be written
length	number of bytes in SSID

Function

uint32_t WDRV_EXT_CmdSSIDGet(uint8_t *ssid, uint8_t *length)

WDRV_EXT_CmdConnectContextBssidGet Function

Gets the BSSID.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdConnectContextBssidGet(uint8_t * bssId);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function gets the current AP's BSSID.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
bssId	pointer where the current AP's BSSID will be written

Function

uint32_t WDRV_EXT_CmdConnectContextBssidGet(uint8_t *bssId)

WDRV_EXT_CmdPowerSaveGet Function

Retrieves current power save status.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdPowerSaveGet(bool * enabled);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function retrieves the current power save status.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
enabled	pointer where the current power save status will be written

Function

uint32_t WDRV_EXT_CmdPowerSaveGet(bool *enabled)

WDRV_EXT_WPSResultsRead Function

Reads the WPS process results back from the MRF24WN module and updates the configuration data.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_EXT_WPSResultsRead(WDRV_CONFIG_DATA * config, uint32_t * status);
```

Returns

None.

Description

After the WPS process has completed, this function is used to read the WPS process results from the MRF24WN module and update the configuration data.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
config	pointer to where configuration data will be updated
status	pointer to where WPS process status will be written

Function

void WDRV_EXT_WPSResultsRead(WDRV_CONFIG_DATA *config, uint32_t *status)

c) External Functions

WDRV_EXT_CmdConnect Function

Directs the MRF24WN to connect to a Wi-Fi network.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdConnect();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function causes the MRF24WN to connect to a Wi-Fi network. Upon connection, or a failure to connect, an event will be generated.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete and relevant connection parameters must have been set.

Function

uint32_t WDRV_EXT_CmdConnect(void)

WDRV_EXT_CmdDisconnect Function

Directs the MRF24WN to disconnect from a Wi-Fi network.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdDisconnect();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function causes the MRF24WN to disconnect from a Wi-Fi network.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete and a connection must be in progress.

Function

uint32_t WDRV_EXT_CmdDisconnect(void)

WDRV_EXT_CmdNetModeAPSet Function

Sets the Wi-Fi network type to SoftAP.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdNetModeAPSet();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sets the Wi-Fi network type to SoftAP.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

uint32_t WDRV_EXT_CmdNetModeAPSet(void)

WDRV_EXT_CmdNetModeBSSSet Function

Sets the Wi-Fi network type to Infrastructure.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdNetModeBSSSet();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sets the Wi-Fi network type to Infrastructure.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

uint32_t WDRV_EXT_CmdNetModeBSSSet(void)

WDRV_EXT_CmdNetModeIBSSSet Function

Sets the Wi-Fi network type to Adhoc.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdNetModeIBSSSet();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sets the Wi-Fi network type to Adhoc.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

uint32_t WDRV_EXT_CmdNetModeIBSSSet(void)

WDRV_EXT_CmdScanStart Function

Directs the MRF24WN module to start a scan.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdScanStart();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function directs the MRF24WN module to start a scan.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

uint32_t WDRV_EXT_CmdScanStart(void)

WDRV_EXT_CmdSecNoneSet Function

Sets Wi-Fi security to open (no security).

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdSecNoneSet();
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sets the Wi-Fi security to open. One can only connect to an AP that is running in open mode.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete and in an unconnected state.

Function

uint32_t WDRV_EXT_CmdSecNoneSet(void)

WDRV_EXT_CmdSecWEPSet Function

Sets Wi-Fi security to use WEP.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdSecWEPSet(uint8_t * key, uint16_t len);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sets the Wi-Fi security to WEP. One can only connect to an AP that is running the same WEP mode.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete and in an unconnected state.

Parameters

Parameters	Description
key	pointer to the WEP key buffer
len	WEP key length

Function

uint32_t WDRV_EXT_CmdSecWEPSet(uint8_t *key, uint16_t len)

WDRV_EXT_CmdSecWPA2Set Function

Sets Wi-Fi security to use WPA2.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdSecWPA2Set(uint8_t * key, uint16_t len);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sets the Wi-Fi security to WPA2. One can only connect to an AP that is running the same WPA2 mode.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete and in an unconnected state.

Parameters

Parameters	Description
key	pointer to the WPA2 key buffer
len	WPA2 key length

Function

uint32_t WDRV_EXT_CmdSecWPA2Set(uint8_t *key, uint16_t len)

WDRV_EXT_CmdSecWPASet Function

Sets Wi-Fi security to use WPA.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdSecWPASet(uint8_t * key, uint16_t len);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sets the Wi-Fi security to WPA. One can only connect to an AP that is running the same WPA mode.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete and in an unconnected state.

Parameters

Parameters	Description
key	pointer to the WPA key buffer
len	WPA key length

Function

uint32_t WDRV_EXT_CmdSecWPASet(uint8_t *key, uint16_t len)

WDRV_EXT_CmdSSIDSet Function

Sets the SSID.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdSSIDSet(uint8_t * ssid, uint16_t len);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sets the SSID and SSID length.

Remarks

Do not include a string terminator in the SSID length. SSIDs are case-sensitive. SSID length must be less than or equal to 32.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
ssid	pointer to SSID buffer
len	number of bytes in SSID

Function

uint32_t WDRV_EXT_CmdSSIDSet(uint8_t *ssid, uint16_t len)

WDRV_EXT_DataSend Function

Sends data packets to MRF24WN module.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_DataSend(uint16_t segSize, uint8_t * p_segData);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sends data packets to the MRF24WN module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
seqSize	data size
p_seqData	pointer to the data buffer

Function

uint32_t WDRV_EXT_DataSend(uint16_t segSize, uint8_t *p_segData)

WDRV_EXT_Deinitialize Function

Deinitializes the MRF24WN Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_EXT_Deinitialize();
```

Returns

None.

Description

This function deinitializes the MRF24WN driver.

Remarks

None

Preconditions

None.

Function

void WDRV_EXT_Deinitialize(void)

WDRV_EXT_HWInterruptHandler Function

Wi-Fi driver (MRF24WN-specific) interrupt service routine.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_EXT_HWInterruptHandler(void const * pointer);
```

Returns

None.

Description

This function is the Wi-Fi driver (MRF24WN-specific) interrupt service routine.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_EXT_HWInterruptHandler(void const *pointer)

WDRV_EXT_PowerUpDown Function

Powers the MRF24WN module up or down.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_PowerUpDown(uint32_t up);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function powers the MRF24WN module up or down.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
up	1: power up; 0: power down

Function

uint32_t WDRV_EXT_PowerUpDown(uint32_t up)

WDRV_EXT_ScanDoneSet Function

Indicates when a scan has completed.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_EXT_ScanDoneSet();
```

Returns

None.

Description

This function indicates when a scan has completed.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_EXT_ScanDoneSet(void)

WDRV_EXT_CmdChannelSet Function

Sets the channel on which to operate.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdChannelSet(uint16_t channel);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sets the channel on which to operate.

Remarks

This works only with SoftAP mode. Do not call this in other modes.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
channel	target channel

Function

WDRV_EXT_CmdChannelSet(uint16_t channel)

WDRV_EXT_CmdFWUpdate Function

Directs the module to start firmware download and upgrade.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_EXT_CmdFWUpdate();
```

Returns

None.

Description

This function directs the module to start the firmware download and upgrade.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_EXT_CmdFWUpdate(void)

WDRV_EXT_CmdSecWpsSet Function

Sets Wi-Fi security to use WPS.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdSecWpsSet(bool pinMode, uint8_t * key, uint16_t keyLen);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function sets the Wi-Fi security to WPS. One can only connect to an AP that supports WPS.

Remarks

None

Preconditions

Wi-Fi initialization must be complete and in an unconnected state.

Parameters

Parameters	Description
pinMode	0: PBC mode; 1: PIN mode

key	pointer of the PIN buffer
keyLen	PIN length

Function

int32_t WDRV_EXT_CmdSecWpsSet(bool pinMode, uint8_t *key, uint16_t keyLen)

WDRV_EXT_CmdPowerSavePut Function

Puts the module in IEEE power save mode.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdPowerSavePut(bool enable);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

The function places the module in IEEE power save mode.

Remarks

This works only with Infrastructure mode. Do not call this in other modes.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
enable	true will put the module in IEEE power save mode

Function

uint32_t WDRV_EXT_CmdPowerSavePut(bool enable)

WDRV_EXT_ScanIsInProgress Function

Check whether host scan is now in progress or not.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
bool WDRV_EXT_ScanIsInProgress();
```

Returns

- true - Host scan is in progress
- false - Host scan is not in progress

Description

Check whether host scan is now in progress or not.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_EXT_ScanIsInProgress(void)

WDRV_EXT_Misc_Config Function

Configures miscellaneous parameters.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_EXT_Misc_Config(uint32_t * config);
```

Returns

None.

Description

This function configures miscellaneous parameters.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
config	pointer to the parameter array

Function

void WDRV_EXT_Misc_Config(uint32_t *config)

WDRV_EXT_CmdRFTest Function

Performs RF test for regulatory test purposes.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdRFTest(uint32_t argc, char * argv[]);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

This function performs a RF test for regulatory test purposes.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Example

Use the arguments in this order:

rate packet_count packet_size channel header_type rate: 0 = 1mbps, 1 = 2mbps, 2 = 5.5mbps, 3 = 11mbps, 4 = 6mbps, 5 = 9mbps, 6 = 12mbps, 7 = 18mbps, 8 = 24mbps, 9 = 36mbps, 10 = 48mbps, 11 = 54mbps, 12 = 6.5mbps, 13 = 13mbps, 14 = 19.5mbps, 15 = 26mbps, 16 = 39mbps, 17

= 52mbps, 18 = 58.5mbps packet_count: number of transmits 1 - 14 packet_size: payload size 0 to 1400 channel: 1 - 14 header_type: 0 - beacon frame; 1 - QOS data frame; 2 - 4 address data frame

Parameters

Parameters	Description
argc	argument count
argv	argument vector

Function

int32_t WDRV_EXT_CmdRFTest(int32_t argc, char *argv[])

WDRV_EXT_CmdTxPowerSet Function

Sets the TX Power.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
uint32_t WDRV_EXT_CmdTxPowerSet(uint32_t dbm);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

The function sets the module's TX power.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
dbm	value of tx power

Function

uint32_t WDRV_EXT_CmdTxPowerSet(uint32_t dbm)

WDRV_EXT_ScanResultGet Function

Reads the selected scan results back from the MRF24WN module.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_EXT_ScanResultGet(uint8_t listIndex, WDRV_SCAN_RESULT * p_scanResult);
```

Returns

- 0 - Indicates success
- Non-zero value - Indicates failure

Description

After a scan has completed this function is used to read one scan result at a time from the MRF24WN module.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
listIndex	index (0 based list) of the scan entry to retrieve
p_scanResult	pointer to where scan result is written

Function

void WDRV_EXT_ScanResultGet(uint8_t listIndex, WDRV_SCAN_RESULT *p_scanResult)

WDRV_EXT_Initialize Function

Initializes the MRF24WN Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
int32_t WDRV_EXT_Initialize(WDRV_CALLBACKS const *const CB);
```

Returns

- 0 - Indicates success
- non-zero value - Indicates failure

Description

This function initializes the MRF24WN Wi-Fi driver, making it ready for clients to use.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
CB	pointer to callback functions

Function

int32_t WDRV_EXT_Initialize(WDRV_CALLBACKS const *const CB)

d) GPIO Functions

WDRV_MRF24WN_ISR Function

Wi-Fi driver (MRF24WN-specific) interrupt service routine.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_MRF24WN_ISR(SYS_MODULE_OBJ index);
```

Returns

None.

Description

This function is the Wi-Fi driver (MRF24WN-specific) interrupt service routine.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void WDRV_MRF24WN_ISR(SYS_MODULE_OBJ index)

WDRV_GPIO_MRF24WG_Disable Function

Disables the MRF24WG module on Multimedia Expansion Board II (MEB II).

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_GPIO_MRF24WG_Disable();
```

Returns

None.

Description

This function disables MRF24WG module on MEB II.

Remarks

The MEB II is configured with a built-in MRF24WG module. The MRF24WG module shares the same SPI bus lines with the MRF24WN module, which results in a bus collision. To avoid the collision, it is required to disable the MRF24WG module on the MEB II.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_GPIO_MRF24WG_Disable(void)

WDRV_GPIO_DeInit Function

Deinitializes the GPIO object for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_GPIO_DeInit();
```

Returns

None.

Description

This function deinitializes the GPIO object for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_GPIO_DeInit(void)

WDRV_GPIO_Init Function

Initializes the GPIO object for the Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_GPIO_Init(GPIO_OUTLOW_T L, GPIO_OUTHIGH_T H);
```

Returns

None.

Description

This function initializes the GPIO object for the Wi-Fi driver.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Parameters

Parameters	Description
L	function pointer to the function drives GPIO low
H	function pointer to the function drives GPIO high

Function

void WDRV_GPIO_Init(GPIO_OUTLOW_T L, GPIO_OUTHIGH_T H)

WDRV_GPIO_OutHigh Function

Pulls GPIO high.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_GPIO_OutHigh(uint32_t channel, uint32_t bit_pos);
```

Returns

None.

Description

This function pulls GPIO high.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Parameters

Parameters	Description
channel	port channel
bit_pos	bit position in the channel

Function

void WDRV_GPIO_OutHigh(uint32_t channel, uint32_t bit_pos)

WDRV_GPIO_OutLow Function

Pulls gpio low.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_GPIO_OutLow(uint32_t channel, uint32_t bit_pos);
```

Returns

None.

Description

Pulls gpio low.

Remarks

None.

Preconditions

TCP/IP stack should be initialized.

Parameters

Parameters	Description
channel	port channel
bit_pos	bit position in the channel

Function

void WDRV_GPIO_OutLow(uint32_t channel, uint32_t bit_pos)

WDRV_GPIO_PowerDown Function

Powers down the MRF24WN module.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_GPIO_PowerDown();
```

Returns

None.

Description

This function powers down the MRF24WN module.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_GPIO_PowerDown(void)

WDRV_GPIO_PowerUp Function

Powers on the MRF24WN module.

Implementation: Dynamic

File

wdrv_mrf24wn_api.h

C

```
void WDRV_GPIO_PowerUp();
```

Returns

None.

Description

This function powers on the MRF24WN module.

Remarks

None.

Preconditions

The TCP/IP stack should be initialized.

Function

void WDRV_GPIO_PowerUp(void)

e) Private Configuration Functions

iwpriv_adhocctx_set Function

Sets the Ad-Hoc network context information.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_adhocctx_set(WDRV_ADHOC_NETWORK_CONTEXT * p_context);
```

Returns

None.

Description

This function sets the current Ad-Hoc network context information by reading from a passed pointer.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
p_context	pointer to where the Ad-Hoc network context is stored

Function

void iwpriv_adhocctx_set(DRV_WIFI_ADHOC_NETWORK_CONTEXT *p_context)

iwpriv_config_read Function

Reads the Wi-Fi context configuration.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_config_read(WDRV_CONFIG_DATA * wifi_config);
```

Returns

None.

Description

This function reads the current Wi-Fi context configuration, copies and stores the whole structure to the pointer passed to the function.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
wifi_config	pointer to where the context configuration is written

Function

void iwpriv_config_read(DRV_WIFI_CONFIG_DATA *wifi_config)

iwpriv_config_write Function

Writes to the Wi-Fi context configuration which is currently used by Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_config_write(void * wifi_config);
```

Returns

None.

Description

This function reads from a passed pointer, copies everything from it, and writes to the Wi-Fi context configuration, which is currently used by the Wi-Fi driver.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
wifi_config	pointer to where the context configuration is stored

Function

void iwpriv_config_write(void *wifi_config)

iwpriv_connstatus_get Function

Gets the Wi-Fi connection status.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
IWPRIV_CONN_STATUS iwpriv_connstatus_get();
```

Returns

Status of current Wi-Fi connection. See the definition for the IWPRIV_CONN_STATUS structure.

Description

This function gets the Wi-Fi connection status.

Remarks

IWPRIV_CONNECTION_FAILED does not necessarily mean that the module fails to connect to the network. It stands on the application's perspective, and actually can be customized. For example, in the Web Server demonstrations's use case, WDRV_CSTATE_CONNECTION_PERMANENTLY_LOST is treated as a fail case and will trigger the application to restart.

Preconditions

Wi-Fi initialization must be complete.

Function

IWPRIV_CONN_STATUS iwpriv_connstatus_get(void)

iwpriv_devinfo_get Function

Gets the device information.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_devinfo_get(void * info);
```

Returns

None.

Description

This function returns the device information, which is either MRF24WG or MRF24WN.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
info	pointer to where the device information is written

Function

void iwpriv_devinfo_get(void *info)

iwpriv_initialconn_set Function

Sets the initial connection status of Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_initialconn_set(bool connect);
```

Returns

None.

Description

This function sets the initial connection status of Wi-Fi driver. After Wi-Fi initialization, it decides whether or not to start the Wi-Fi connection.

Remarks

This function is mainly used to implement prescan. It has to be called before Wi-Fi driver's initialization is finished to be effective.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
connect	boolean value which indicates whether or not to start an initial connect

Function

void iwpriv_initialconn_set(bool connect)

iwpriv_initstatus_get Function

Gets the initialization status of Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
uint8_t iwpriv_initstatus_get();
```

Returns

Current initialization status of the Wi-Fi driver (IWPRIV_READY or IWPRIV_IN_PROGRESS).

Description

This function returns the initialization status of the Wi-Fi driver.

Remarks

None.

Preconditions

None.

Function

uint8_t iwpriv_initstatus_get(void)

iwpriv_is_servermode Function

Checks if the passed Wi-Fi context configuration is operating in server mode.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
bool iwpriv_is_servermode();
```

Returns

- true - Wi-Fi context configuration is operating in server mode
- false - Wi-Fi context configuration is not operating in server mode

Description

This function checks if the passed Wi-Fi context configuration is operating in server mode, which includes Ad-Hoc mode and Soft AP mode.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

bool iwpriv_is_servermode(void)

iwpriv_leftclient_get Function

Gets the left client's information.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_leftclient_get(bool * updated, TCPIP_MAC_ADDR * addr);
```

Returns

None.

Description

This function returns the left client's information when the Wi-Fi module works in server mode and has the DHCP Server enabled.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
updated	if the left client's information needs to be updated
addr	MAC address of the left client

Function

void iwpriv_leftclient_get(bool *updated, TCPIP_MAC_ADDR *addr)

iwpriv_mcastfilter_set Function

Adds a MAC address to the multi-cast filter.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
IWPRIV_STATUS iwpriv_mcastfilter_set(uint8_t * addr);
```

Returns

Status of the set operation, IWPRIV_READY or IWPRIV_ERROR. See definition for the IWPRIV_STATUS structure.

Description

This function adds a MAC address to the multi-cast filter.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
addr	pointer to where the MAC address is stored

Function

IWPRIV_STATUS iwpriv_mcastfilter_set(uint8_t *addr)

iwpriv_nettype_get Function

Gets the current network type.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_nettype_get(uint8_t * netType);
```

Returns

None.

Description

This function returns the current network type.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
netType	pointer to where the network type is written

Function

void iwpriv_nettype_get(uint8_t *netType)

iwpriv_nettype_set Function

Sets the current network type.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_nettype_set(uint8_t netType);
```

Returns

None.

Description

This function sets the current network type.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
netType	the network type to set

Function

void iwpriv_nettype_set(uint8_t netType)

iwpriv_numberofscanresults_get Function

Gets the number of scan results.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
uint16_t iwpriv_numberofscanresults_get();
```

Returns

Number of scan results.

Description

This function gets the number of scan results.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

uint16_t iwpriv_numberofscanresults_get(void)

iwpriv_powersave_config Function

Enables or disables Power Save mode in Wi-Fi driver.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_powersave_config(bool enabled);
```

Returns

None.

Description

This function enables or disables Power Save mode in Wi-Fi driver, which depends on the passed boolean value.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
enabled	boolean value which indicates to enable or disable Power Save mode in Wi-Fi driver

Function

void iwpriv_powersave_config(bool enabled)

iwpriv_prescan_start Function

Starts prescan.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_prescan_start();
```

Returns

None.

Description

This function directs the Wi-Fi driver to start a prescan.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void iwpriv_prescan_start(void)

iwpriv_scan_start Function

Starts scan.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_scan_start();
```

Returns

None.

Description

The function starts a Wi-Fi scan.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

void iwpriv_scan_start(void)

iwpriv_scanresults_display Function

Outputs all of the scan results.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
IWPRIV_STATUS iwpriv_scanresults_display();
```

Returns

Status of the output operation, IWPRIV_IN_PROGRESS or IWPRIV_READY. See definition for the IWPRIV_STATUS structure.

Description

This function outputs all of the scan results to the console terminal.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

IWPRIV_STATUS iwpriv_scanresults_display(void)

iwpriv_scanresult_get Function

Gets one scan result.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void * iwpriv_scanresult_get(uint16_t index);
```

Returns

Pointer to where the scan result is written. It uses the g_scanResults[] array to store results, which eliminates the need to allocate additional memory.

Description

This function gets one scan result, which includes the detailed information of an AP.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
index	index of the scan result

Function

void *iwpriv_scanresult_get(uint16_t index)

iwpriv_scanresults_save Function

Saves all of the scan results.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
IWPRIV_STATUS iwpriv_scanresults_save();
```

Returns

Status of the save operation, IWPRIV_IN_PROGRESS or IWPRIV_READY. See definition for the IWPRIV_STATUS structure.

Description

This function saves all of the scan results to the console terminal.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

IWPRIV_STATUS iwpriv_scanresults_save(void)

iwpriv_scanstatus_get Function

Gets the prescan status.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
IWPRIV_SCAN_STATUS iwpriv_scanstatus_get();
```

Returns

Prescan status: IWPRIV_SCAN_IDLE, IWPRIV_SCAN_IN_PROGRESS, IWPRIV_SCAN_NO_AP_FOUND or IWPRIV_SCAN_SUCCESSFUL. See the definition for the IWPRIV_SCAN_STATUS structure.

Description

This function gets the prescan status.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

IWPRIV_SCAN_STATUS iwpriv_scanstatus_get(void)

iwpriv_ssid_get Function

Gets the current SSID.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_ssid_get(uint8_t * ssid, uint8_t * ssidLen);
```

Returns

None.

Description

This function returns the current SSID.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
ssid	pointer to where the SSID is written
ssidLen	pointer to where the SSID length is written

Function

void iwpriv_ssid_get(uint8_t *ssid, uint8_t *ssidLen)

iwpriv_ssid_set Function

Sets the current SSID.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_ssid_set(uint8_t * ssid, uint8_t ssidLen);
```

Returns

None.

Description

This function sets the current SSID.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
ssid	pointer to where the SSID is stored

ssidLen	pointer to where the SSID length is stored

Function

void iwpriv_ssid_set(uint8_t *ssid, uint8_t ssidLen)

iwpriv_execute Function

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_execute(IWPRIV_CMD cmd, IWPRIV_EXECUTE_PARAM * params);
```

Description

This is function iwpriv_execute.

iwpriv_get Function

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_get(IWPRIV_CMD cmd, IWPRIV_GET_PARAM * params);
```

Description

This is function iwpriv_get.

iwpriv_prescan_isfinished Function

Checks if the prescan is complete.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
bool iwpriv_prescan_isfinished();
```

Returns

None.

Description

This function checks if the prescan is complete.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

bool iwpriv_prescan_isfinished(void)

iwpriv_prescan_option_get Function

To see if prescan will run before next connection.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
bool iwpriv_prescan_option_get();
```

Returns

None.

Description

This function checks whether or not the prescan will run before next connection.

Remarks

None.

Preconditions

Wi-Fi initialization must be complete.

Function

bool iwpriv_prescan_option_get(void)

iwpriv_prescan_option_set Function

To run prescan or not.

Implementation: Dynamic

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_prescan_option_set(bool scan);
```

Returns

None.

Description

This function controls whether or not to run prescan.

Remarks

Prescan means the scan runs before the module is connected. It needs to use multiple functions in this file. Please refer to the Easy Configuration demonstration to see the correct usage of prescan.

After the the module is connected, MRF24WN module can also do regular scans. But it cannot perform a scan when the connection is in progress.

Preconditions

Wi-Fi initialization must be complete.

Parameters

Parameters	Description
scan	true: run prescan before next connection
false	do not run prescan before next connection

Function

void iwpriv_prescan_option_set(bool scan)

iwpriv_set Function

File

wdrv_mrf24wn_iwpriv.h

C

```
void iwpriv_set(IWPRIV_CMD cmd, IWPRIV_SET_PARAM * params);
```

Description

This is function iwpriv_set.

f) Data Types and Constants

IWPRIV_CONN_STATUS Enumeration

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef enum {
  IWPRIV_CONNECTION_FAILED = -1,
  IWPRIV_CONNECTION_SUCCESSFUL,
  IWPRIV_CONNECTION_IDLE,
  IWPRIV_CONNECTION_IN_PROGRESS,
  IWPRIV_CONNECTION_REESTABLISHED
} IWPRIV_CONN_STATUS;
```

Description

This is type IWPRIV_CONN_STATUS.

IWPRIV_STATUS Enumeration

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef enum {
  IWPRIV_ERROR = -1,
  IWPRIV_READY,
  IWPRIV_IN_PROGRESS
} IWPRIV_STATUS;
```

Description

This is type IWPRIV_STATUS.

IWPRIV_CMD Enumeration

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef enum {
  PRESCAN_OPTION_GET,
  PRESCAN_OPTION_SET,
  PRESCAN_START,
  PRESCAN_ISFINISHED_GET,
  SCAN_START,
  SCANSTATUS_GET,
  SCANRESULT_GET,
  SCANRESULTS_COUNT_GET,
  SCANRESULTS_DISPLAY,
  SCANRESULTS_SAVE,
  CONFIG_GET,
  CONFIG_SET,
  SSID_GET,
  SSID_SET,
  NETWORKTYPE_GET,
  NETWORKTYPE_SET,
  CONNSTATUS_GET,
  CLIENTINFO_GET,
  DEVICEINFO_GET,
```

```
  INITSTATUS_GET,
  OPERATIONMODE_GET,
  INITCONN_OPTION_SET,
  ADHOCCTX_SET,
  MULTICASTFILTER_SET,
  POWERSAVE_SET
} IWPRIV_CMD;
```

Description

This is type IWPRIV_CMD.

IWPRIV_EXECUTE_PARAM Union

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef union {
  IWPRIV_PARAM_SCAN scan;
} IWPRIV_EXECUTE_PARAM;
```

Description

This is type IWPRIV_EXECUTE_PARAM.

IWPRIV_GET_PARAM Union

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef union {
  IWPRIV_PARAM_SCAN scan;
  IWPRIV_PARAM_CONFIG config;
  IWPRIV_PARAM_SSID ssid;
  IWPRIV_PARAM_NETWORKTYPE netType;
  IWPRIV_PARAM_CONNECT conn;
  IWPRIV_PARAM_CLIENTINFO clientInfo;
  IWPRIV_PARAM_DEVICEINFO devInfo;
  IWPRIV_PARAM_DRIVERSTATUS driverStatus;
  IWPRIV_PARAM_OPERATIONMODE opMode;
} IWPRIV_GET_PARAM;
```

Description

This is type IWPRIV_GET_PARAM.

IWPRIV_PARAM_CLIENTINFO Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  uint8_t * addr;
  bool updated;
} IWPRIV_PARAM_CLIENTINFO;
```

Members

Members	Description
uint8_t * addr;	it usually points to a MAC address, which is an array of 6 uint8_t elements

Description

This is type IWPRIV_PARAM_CLIENTINFO.

IWPRIV_PARAM_CONTEXT Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  void * data;
} IWPRIV_PARAM_CONTEXT;
```

Description

This is type IWPRIV_PARAM_CONTEXT.

IWPRIV_PARAM_DEVICEINFO Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  void * data;
} IWPRIV_PARAM_DEVICEINFO;
```

Description

This is type IWPRIV_PARAM_DEVICEINFO.

IWPRIV_SCAN_STATUS Enumeration

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef enum {
  IWPRIV_SCAN_SUCCESSFUL,
  IWPRIV_SCAN_IDLE,
  IWPRIV_SCAN_IN_PROGRESS,
  IWPRIV_SCAN_NO_AP_FOUND
} IWPRIV_SCAN_STATUS;
```

Description

This is type IWPRIV_SCAN_STATUS.

IWPRIV_SET_PARAM Union

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef union {
  IWPRIV_PARAM_SCAN scan;
  IWPRIV_PARAM_CONFIG config;
  IWPRIV_PARAM_SSID ssid;
  IWPRIV_PARAM_NETWORKTYPE netType;
  IWPRIV_PARAM_CONNECT conn;
  IWPRIV_PARAM_CONTEXT ctx;
  IWPRIV_PARAM_MULTICASTFILTER multicast;
  IWPRIV_PARAM_POWERSAVE powerSave;
} IWPRIV_SET_PARAM;
```

Description

This is type IWPRIV_SET_PARAM.

IWPRIV_PARAM_CONFIG Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  void * data;
} IWPRIV_PARAM_CONFIG;
```

Description

This is type IWPRIV_PARAM_CONFIG.

IWPRIV_PARAM_CONNECT Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  bool initConnAllowed;
  IWPRIV_CONN_STATUS status;
} IWPRIV_PARAM_CONNECT;
```

Description

This is type IWPRIV_PARAM_CONNECT.

IWPRIV_PARAM_DRIVERSTATUS Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  uint8_t initStatus;
} IWPRIV_PARAM_DRIVERSTATUS;
```

Description

This is type IWPRIV_PARAM_DRIVERSTATUS.

IWPRIV_PARAM_MULTICASTFILTER Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  uint8_t * addr;
  IWPRIV_STATUS status;
} IWPRIV_PARAM_MULTICASTFILTER;
```

Members

Members	Description
uint8_t * addr;	it usually points to a MAC address, which is an array of 6 uint8_t elements

Description

This is type IWPRIV_PARAM_MULTICASTFILTER.

IWPRIV_PARAM_NETWORKTYPE Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  uint8_t type;
} IWPRIV_PARAM_NETWORKTYPE;
```

Description

This is type IWPRIV_PARAM_NETWORKTYPE.

IWPRIV_PARAM_OPERATIONMODE Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  bool isServer;
} IWPRIV_PARAM_OPERATIONMODE;
```

Description

This is type IWPRIV_PARAM_OPERATIONMODE.

IWPRIV_PARAM_POWERSAVE Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  bool enabled;
} IWPRIV_PARAM_POWERSAVE;
```

Description

This is type IWPRIV_PARAM_POWERSAVE.

IWPRIV_PARAM_SCAN Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  void * data;
  bool prescanAllowed;
  bool prescanFinished;
  IWPRIV_SCAN_STATUS scanStatus;
  uint16_t numberOfResults;
  uint16_t index;
  IWPRIV_STATUS displayStatus;
  IWPRIV_STATUS saveStatus;
} IWPRIV_PARAM_SCAN;
```

Description

This is type IWPRIV_PARAM_SCAN.

IWPRIV_PARAM_SSID Structure

File

wdrv_mrf24wn_iwpriv.h

C

```
typedef struct {
  uint8_t * ssid;
  uint8_t ssidLen;
} IWPRIV_PARAM_SSID;
```

Description

This is type IWPRIV_PARAM_SSID.

Files

Files

Name	Description
wdrv_mrf24wn_api.h	MRF24WN Interface Functions
wdrv_mrf24wn_iwpriv.h	Configure optional (private) parameters of MRF24WN driver.

Description

This section lists the source and header files used by the MRF24WN Wi-Fi Driver Library.

wdrv_mrf24wn_api.h

MRF24WN Interface Functions

Functions

	Name	Description
	WDRV_CLI_Init	Initializes the console CLI interface. **Implementation:** Dynamic
	WDRV_EXT_CmdChannelSet	Sets the channel on which to operate. **Implementation:** Dynamic
	WDRV_EXT_CmdConnect	Directs the MRF24WN to connect to a Wi-Fi network. **Implementation:** Dynamic
	WDRV_EXT_CmdConnectContextBssidGet	Gets the BSSID. **Implementation:** Dynamic
	WDRV_EXT_CmdConnectContextChannelGet	Gets the AP channel **Implementation:** Dynamic
	WDRV_EXT_CmdDisconnect	Directs the MRF24WN to disconnect from a Wi-Fi network. **Implementation:** Dynamic
	WDRV_EXT_CmdFWUpdate	Directs the module to start firmware download and upgrade. **Implementation:** Dynamic
	WDRV_EXT_CmdMacAddressGet	Retrieves the MRF24WN MAC address. **Implementation:** Dynamic
	WDRV_EXT_CmdNetModeAPSet	Sets the Wi-Fi network type to SoftAP. **Implementation:** Dynamic
	WDRV_EXT_CmdNetModeBSSSet	Sets the Wi-Fi network type to Infrastructure. **Implementation:** Dynamic
	WDRV_EXT_CmdNetModeIBSSSet	Sets the Wi-Fi network type to Adhoc. **Implementation:** Dynamic
	WDRV_EXT_CmdPowerSaveGet	Retrieves current power save status. **Implementation:** Dynamic
	WDRV_EXT_CmdPowerSavePut	Puts the module in IEEE power save mode. **Implementation:** Dynamic

	WDRV_EXT_CmdRFTest	Performs RF test for regulatory test purposes. **Implementation:** Dynamic
	WDRV_EXT_CmdScanGet	Reads the number of scan results from the MRF24WN module. **Implementation:** Dynamic
	WDRV_EXT_CmdScanStart	Directs the MRF24WN module to start a scan. **Implementation:** Dynamic
	WDRV_EXT_CmdSecGet	Gets the current Wi-Fi security type. **Implementation:** Dynamic
	WDRV_EXT_CmdSecNoneSet	Sets Wi-Fi security to open (no security). **Implementation:** Dynamic
	WDRV_EXT_CmdSecWEPSet	Sets Wi-Fi security to use WEP. **Implementation:** Dynamic
	WDRV_EXT_CmdSecWPA2Set	Sets Wi-Fi security to use WPA2. **Implementation:** Dynamic
	WDRV_EXT_CmdSecWPASet	Sets Wi-Fi security to use WPA. **Implementation:** Dynamic
	WDRV_EXT_CmdSecWpsSet	Sets Wi-Fi security to use WPS. **Implementation:** Dynamic
	WDRV_EXT_CmdSSIDGet	Gets the SSID. **Implementation:** Dynamic
	WDRV_EXT_CmdSSIDSet	Sets the SSID. **Implementation:** Dynamic
	WDRV_EXT_CmdTxPowerSet	Sets the TX Power. **Implementation:** Dynamic
	WDRV_EXT_DataSend	Sends data packets to MRF24WN module. **Implementation:** Dynamic
	WDRV_EXT_Deinitialize	Deinitializes the MRF24WN Wi-Fi driver. **Implementation:** Dynamic
	WDRV_EXT_HWInterruptHandler	Wi-Fi driver (MRF24WN-specific) interrupt service routine. **Implementation:** Dynamic
	WDRV_EXT_Initialize	Initializes the MRF24WN Wi-Fi driver. **Implementation:** Dynamic
	WDRV_EXT_Misc_Config	Configures miscellaneous parameters. **Implementation:** Dynamic
	WDRV_EXT_PowerUpDown	Powers the MRF24WN module up or down. **Implementation:** Dynamic
	WDRV_EXT_ScanDoneSet	Indicates when a scan has completed. **Implementation:** Dynamic
	WDRV_EXT_ScanIsInProgress	Check whether host scan is now in progress or not. **Implementation:** Dynamic
	WDRV_EXT_ScanResultGet	Reads the selected scan results back from the MRF24WN module. **Implementation:** Dynamic
	WDRV_EXT_WPSResultsRead	Reads the WPS process results back from the MRF24WN module and updates the configuration data. **Implementation:** Dynamic
	WDRV_GPIO_DeInit	Deinitializes the GPIO object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_GPIO_Init	Initializes the GPIO object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_GPIO_MRF24WG_Disable	Disables the MRF24WG module on Multimedia Expansion Board II (MEB II). **Implementation:** Dynamic
	WDRV_GPIO_OutHigh	Pulls GPIO high. **Implementation:** Dynamic
	WDRV_GPIO_OutLow	Pulls gpio low. **Implementation:** Dynamic
	WDRV_GPIO_PowerDown	Powers down the MRF24WN module. **Implementation:** Dynamic

	WDRV_GPIO_PowerUp	Powers on the MRF24WN module. **Implementation:** Dynamic
	WDRV_HibernateDisable	Wakes MRF24WN up from hibernate mode. **Implementation:** Dynamic
	WDRV_HibernateEnable	Hibernates MRF24WN module. **Implementation:** Dynamic
	WDRV_InHibernateMode	Checks if MRF24WN is in hibernate mode. **Implementation:** Dynamic
	WDRV_INTR_Deinit	Deinitializes interrupts for Wi-Fi driver. **Implementation:** Dynamic
	WDRV_INTR_Init	Initializes interrupts for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_INTR_SourceDisable	Disables interrupts from the module. **Implementation:** Dynamic
	WDRV_INTR_SourceEnable	Enables interrupts from the module. **Implementation:** Dynamic
	WDRV_MRF24WN_ISR	Wi-Fi driver (MRF24WN-specific) interrupt service routine. **Implementation:** Dynamic
	WDRV_SPI_Deinit	Deinitializes the SPI object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_SPI_In	Receives data from the module through the SPI bus. **Implementation:** Dynamic
	WDRV_SPI_Init	Initializes the SPI object for the Wi-Fi driver. **Implementation:** Dynamic
	WDRV_SPI_Out	Sends data out to the module through the SPI bus. **Implementation:** Dynamic

Description

MRF24WN Interface Functions

File Name

wdrv_mrf24wn_api.h

Company

Microchip Technology Inc.

wdrv_mrf24wn_iwpriv.h

Configure optional (private) parameters of MRF24WN driver.

Enumerations

	Name	Description
	IWPRIV_CMD	This is type IWPRIV_CMD.
	IWPRIV_CONN_STATUS	This is type IWPRIV_CONN_STATUS.
	IWPRIV_SCAN_STATUS	This is type IWPRIV_SCAN_STATUS.
	IWPRIV_STATUS	This is type IWPRIV_STATUS.

Functions

	Name	Description
	iwpriv_adhocctx_set	Sets the Ad-Hoc network context information. **Implementation:** Dynamic
	iwpriv_config_read	Reads the Wi-Fi context configuration. **Implementation:** Dynamic
	iwpriv_config_write	Writes to the Wi-Fi context configuration which is currently used by Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_connstatus_get	Gets the Wi-Fi connection status. **Implementation:** Dynamic

	iwpriv_devinfo_get	Gets the device information. **Implementation:** Dynamic
	iwpriv_execute	This is function iwpriv_execute.
	iwpriv_get	This is function iwpriv_get.
	iwpriv_initialconn_set	Sets the initial connection status of Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_initstatus_get	Gets the initialization status of Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_is_servermode	Checks if the passed Wi-Fi context configuration is operating in server mode. **Implementation:** Dynamic
	iwpriv_leftclient_get	Gets the left client's information. **Implementation:** Dynamic
	iwpriv_mcastfilter_set	Adds a MAC address to the multi-cast filter. **Implementation:** Dynamic
	iwpriv_nettype_get	Gets the current network type. **Implementation:** Dynamic
	iwpriv_nettype_set	Sets the current network type. **Implementation:** Dynamic
	iwpriv_numberofscanresults_get	Gets the number of scan results. **Implementation:** Dynamic
	iwpriv_powersave_config	Enables or disables Power Save mode in Wi-Fi driver. **Implementation:** Dynamic
	iwpriv_prescan_isfinished	Checks if the prescan is complete. **Implementation:** Dynamic
	iwpriv_prescan_option_get	To see if prescan will run before next connection. **Implementation:** Dynamic
	iwpriv_prescan_option_set	To run prescan or not. **Implementation:** Dynamic
	iwpriv_prescan_start	Starts prescan. **Implementation:** Dynamic
	iwpriv_scan_start	Starts scan. **Implementation:** Dynamic
	iwpriv_scanresult_get	Gets one scan result. **Implementation:** Dynamic
	iwpriv_scanresults_display	Outputs all of the scan results. **Implementation:** Dynamic
	iwpriv_scanresults_save	Saves all of the scan results. **Implementation:** Dynamic
	iwpriv_scanstatus_get	Gets the prescan status. **Implementation:** Dynamic
	iwpriv_set	This is function iwpriv_set.
	iwpriv_ssid_get	Gets the current SSID. **Implementation:** Dynamic
	iwpriv_ssid_set	Sets the current SSID. **Implementation:** Dynamic

Structures

	Name	Description
	IWPRIV_PARAM_CLIENTINFO	This is type IWPRIV_PARAM_CLIENTINFO.
	IWPRIV_PARAM_CONFIG	This is type IWPRIV_PARAM_CONFIG.
	IWPRIV_PARAM_CONNECT	This is type IWPRIV_PARAM_CONNECT.
	IWPRIV_PARAM_CONTEXT	This is type IWPRIV_PARAM_CONTEXT.
	IWPRIV_PARAM_DEVICEINFO	This is type IWPRIV_PARAM_DEVICEINFO.
	IWPRIV_PARAM_DRIVERSTATUS	This is type IWPRIV_PARAM_DRIVERSTATUS.
	IWPRIV_PARAM_MULTICASTFILTER	This is type IWPRIV_PARAM_MULTICASTFILTER.
	IWPRIV_PARAM_NETWORKTYPE	This is type IWPRIV_PARAM_NETWORKTYPE.
	IWPRIV_PARAM_OPERATIONMODE	This is type IWPRIV_PARAM_OPERATIONMODE.

	IWPRIV_PARAM_POWERSAVE	This is type IWPRIV_PARAM_POWERSAVE.
	IWPRIV_PARAM_SCAN	This is type IWPRIV_PARAM_SCAN.
	IWPRIV_PARAM_SSID	This is type IWPRIV_PARAM_SSID.

Unions

	Name	Description
	IWPRIV_EXECUTE_PARAM	This is type IWPRIV_EXECUTE_PARAM.
	IWPRIV_GET_PARAM	This is type IWPRIV_GET_PARAM.
	IWPRIV_SET_PARAM	This is type IWPRIV_SET_PARAM.

Description

MRF24WN Private Configuration Support

Functions in this module support the connection process for the MRF24WN.

File Name

wdrv_mrf24wn_iwpriv.h

Company

Microchip Technology Inc.

Graphics Libraries Help

This section provides information on the Graphics Libraries that are available in MPLAB Harmony.

MPLAB Harmony Graphics Library

This section provides information on the MPLAB Harmony Graphics Library.

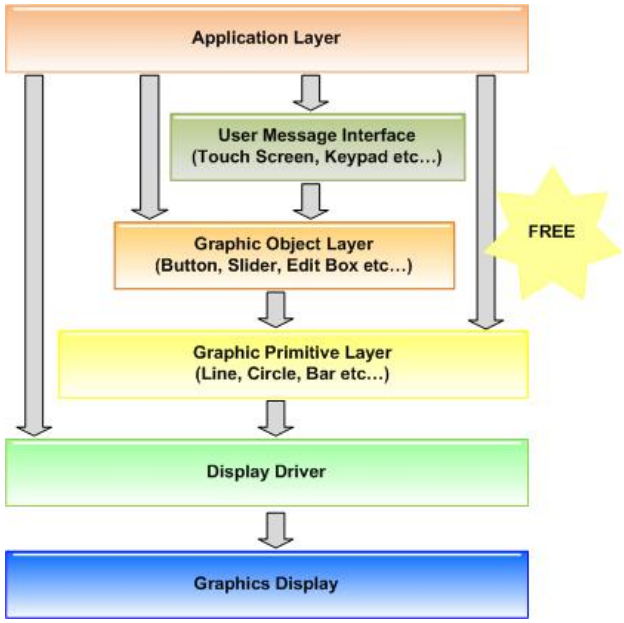
Introduction

This topic provides an overview of the Graphics Library.

Description

The Graphics Library is a free, modular library optimized for Microchip microcontrollers. The library structure is shown in the following figure.

Graphics Library Architecture



- Application Layer - This is the program that utilizes the Graphics Library
- User Message Interface - This is a layer that should be implemented by the user to provide messages for the library
- Graphics Object Layer - This layer renders the control objects such as button, list box, progress bar, meter, and so on
- Graphics Primitives Layer - This layer implements the primitive rendering functions
- Device Display Driver - This layer is the graphics display driver component that is optimized to the actual display module used
- Graphics Display Module - This is the actual display module

The library includes features such as alpha blending, gradient fills, and anti-aliased fonts. These features can be enabled or disabled through build configurations. Applications can take advantage of these features to enhance the user experience while delivering performance required by the application.

Graphics Library Porting Guide

This topic provides porting information for the Graphics Library.

Introduction

This section provides information for porting from the Microchip Library of Applications (MLA) version of the Graphics Library to the Graphics Library within MPLAB Harmony.

Upgrading from the MLA Graphics Library to the MPLAB Harmony Graphics Library

The MPLAB Harmony Graphics Library is a completely new distribution, which is included as part of the MPLAB Harmony installation. No files need to be maintained from an existing MLA Graphics Library installation.

MPLAB Harmony Graphics Library Key Features

This topic provides a description of the key features of the MPLAB Harmony Graphics Library.

Description

Note: The MPLAB Harmony Graphics Library *does not* support any Microchip 16-bit family devices.

The following are the key features of the MPLAB Harmony Graphics Library:

- Multiple instances
- Fully dynamic:
- Stack initialization/deinitialization
- Resource management
- Configurable
- Improved modularity and stack layout
- Interrupt driven operation
- RTOS friendly, with easy RTOS integration

MPLAB Harmony Graphics Library Structure

This topic describes the structure of the MPLAB Harmony Graphics Library.

Description

The MPLAB Harmony Graphics Library consists of a graphic widget and primitive library, a display layer, a graphics driver, and demonstration applications, all of which support the following structure:

- Lowercase file names
- Underscores are used
- System services are used instead of direct access to peripherals
- Interrupts
- Drivers
- Timer Services
- File System
- Board Support Package (BSP)

By default, the Graphics Library and its corresponding display drivers, Board Support Package (BSP) and demonstration applications are placed into the following locations during installation of MPLAB Harmony:

- `/microchip/harmony/<version>/framework/gfx`
- `/microchip/harmony/<version>/framework/driver/gfx`
- `/microchip/harmony/<version>/apps/gfx`

MPLAB Harmony Graphics Library Design

This topic discusses the design of the MPLAB Harmony Graphics Library.

Description

The MPLAB Harmony Graphics Library is designed as a part of a system running other applications, middleware, etc. Therefore, it makes use of the system services that are available to other modules in the system or to the application, such as:

- File System
- System Interrupts
- System Driver Services
- System Timers
- System Device Drivers
- System Command Processor
- System Console
- System Debug Services
- Header files located in `./framework/gfx/src/system` that expose the system-wide available API

MPLAB Harmony Graphics Library API Changes

This topic discusses the reasons behind the API changes to the MPLAB Harmony Graphics Library.

Description

The API is backward compatible in functionality only. All function names and type definitions have changed to reflect the MPLAB Harmony philosophy.

The changes in the library API have been minimized so that the porting process is straightforward and integration with other dependents is made simple. New functionality has been added because of new features, such as dynamic configuration and non-block of the library. Other than these new features, the changes to the API were also made to support commonality between the MPLAB Harmony Graphics Library and the 16-bit Graphics Library.

The API changes are at all layers of the Graphics Library which include the GOL, Primitive, and Driver layers.

MPLAB Harmony Graphics Library Access Changes

This topic describes the stack access updates that were made to the MPLAB Harmony Graphics Library.

Description

The Graphics Library provides the `gfx.h` and `gfx.c` file as the central interface to the library.

The Graphics Library can provide mutually exclusive access to multiple clients. Each client calls the GFX_Initialize function using a client index and appropriate initialization parameters.

The GFX_Initialize function should be called before using the Graphics Object Layer and Primitive Layer.

The Graphics Library no longer initializes the display driver, but requires the application to initialize the driver before calling any Graphics Library functions. For example: GFX_DRV_lcc_Initialize

Note: Direct access to the Graphics Driver rendering functions *is not* supported.

Graphic Primitive Layer Changes

This topic discusses the changes made to the MLA Graphics Library `primitive.h` file to produce the equivalent MPLAB Harmony Graphics Library `gfx_primitive.h` file.

Description

Changes

The changes are as follows:

- *Prefix:*
- `GFX_` is the generic prefix for functions
- `gfx_` is the generic prefix for primitive level files
- Multi-instance support - all re-entrant functions require a client instance index to be passed in as a parameter

Graphic Object Layer (GOL) Changes

This topic discusses the changes made to the MLA Graphics Library `GOL.h` file to produce the equivalent MPLAB Harmony Graphics Library `gfx_gol.h` file.

Description

Changes

The changes are as follows.

- *Prefix:*
- `GFX_GOL_` is the generic prefix for all Graphic Object Level functions
- `gfx_gol_` is the generic prefix for all Graphic Object Level files
- Pre-emption Level support - a three-level self preemption configuration has been added for time-critical rendering
- Multi-instance support - all re-entrant functions require a client instance index to be passed in as a parameter

Graphic Display Driver Layer Changes

This topic discusses the changes made to the `DisplayDriver.h` file.

This topic discusses the changes made to the MLA Graphics Library `DisplayDriver.h` file to produce the equivalent MPLAB Harmony Graphics Library `drv_gfx_display.h` file.

Description

Changes

The changes are as follows:

- *Prefix:*
- `DRV_GFX` is the generic prefix for all Graphics Driver level functions
- `drv_gfx` is the generic prefix for all Graphics Driver level files
- *Common API Interface* – This interface establishes the basic set of driver functions and parameters required by the Graphics Primitive Layer to display graphics content on a Liquid Crystal Display. It also lists the initialization and deinitialization functions and parameters required by the application to establish a driver.

MPLAB Harmony Graphics Library Initialization Changes

This topic describes the initialization updates to the MPLAB Harmony Graphics Library.

Description

The Graphics Library include header files, `gfx_gol.h` and `gfx_primitive.h`, have been updated as follows.

To initialize the library the call is now: `GFX_Initialize`

The default Graphics Library configuration is provided with the library distribution and consists of selecting **Graphics Library** through the MHC Options tab.

New MPLAB Harmony Graphics Library API Functions

This topic describes some of the important new API functions in the MPLAB Harmony Graphics Library.

Description

There are many new functions available that have been introduced to support the MPLAB Harmony philosophy. However, there is no concern for the porting process regarding the new API calls as the previous library implementation did not support this kind of service.

Existing applications will not port without using the new APIs:

- Initialization
- Steady State

MPLAB Harmony Graphics Library Utilities

This topic describes the MPLAB Harmony Graphics Library utilities.

Description

The MPLAB Harmony Graphics Library includes the following utilities:

- Graphics Display Designer X (GDDX) - This utility can be used for designing and building graphical user interfaces (GUIs) from graphics objects
- Graphics Resource Converter (GRC) - This utility can be used for converting data into logically deployable entities that can be used in a graphics application

Porting Applications from the MLA Graphics Library to the MPLAB Harmony Graphics Library

Porting an application from the MLA Graphics Library into the Graphics Library within MPLAB Harmony consists of updating existing Graphics Library function names in your application to the new MPLAB Harmony function names.

MPLAB Harmony Graphics Library File Name Compliance

This topic provides header file name mapping tables that show the MLA Graphics Library file name and the compliant name within the MPLAB Harmony Graphics Library.

Description

The replacement names chosen to make existing Graphics Library functions compliant with MPLAB Harmony are shown in individual tables per header file name.

The files listed in the tables are located in the following two MPLAB Harmony installation folders:

`.\harmony\<version>\framework\gfx`

`.\harmony\<version>\framework\gfx\src`

The following table shows the original and new names used. Note that other than name changes, no other modifications were made to the middleware functions.

MLA Graphics Library File Name	MPLAB Harmony Graphics Library File Name
AnalogClock.h	gfx_gol_analog_clock.h
Button.h	gfx_gol_button.h
Chart.h	gfx_gol_chart.h
CheckBox.h	gfx_gol_check_box.h
DigitalMeter.h	gfx_gol_digital_meter.h
EditBox.h	gfx_gol_edit_box.h
GOL.h	gfx_gol.h
Graphics.h	gfx.h
Grid.h	gfx_gol_grid.h (not supported in this release)
GroupBox.h	gfx_gol_group_box.h
ListBox.h	gfx_gol_list_box.h
Meter.h	gfx_gol_meter.h
Palette.h	gfx_palette.h
Picture.h	gfx_gol_picture.h
Primitive.h	gfx_primitive.h
ProgressBar.h	gfx_gol_progress_bar.h
Radiobutton.h	gfx_gol_radio_button.h
RoundDial.h	gfx_gol_round_dial.h
Slider.h	gfx_gol_scroll_bar.h
StaticText.h	gfx_gol_static_text.h
TextEntry.h	gfx_gol_text_entry.h
Transitions.h	gfx_transitions.h
Window.h	gfx_gol_window.h

MPLAB Harmony Graphics Library Function Name Compliance

This topic provides header file function mapping tables that show the MLA Graphics Library function name and the compliant name within the MPLAB Harmony Graphics Library.

Description

The replacement function names chosen to make the existing MLA Graphics Library function compliant with MPLAB Harmony are shown in individual tables per header file name.

The files listed in the tables are located in the following MPLAB Harmony installation folder: `./harmony/<version>/framework/gfx`.

Header File: `gfx_display_driver.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
InitGraph()	GFX_PixelsPut()
PutPixel	GFX_BarFill()
GetPixel()	GFX_PixelArrayPut()
SetColor()	GFX_PixelArrayGet()
PutPixel	GFX_PixelPut()
SetColor ()	GFX_SetColor()
GetTransparentColor()	GFX_SetInstance()
TransparentColorEnable()	GFX_SetPage()
TransparentColorDisable()	GFX_Layer()
Bar()	GFX_PixelGet()

N/A	GFX_AlphaBlendWindow()

Header File: `gfx_primitive.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
InitGraph()	GFX_Initialize()
PutPixel	GFX_PixelPut()
GetPixel()	GFX_PixelGet()
SetColor()	GFX_ColorSet()
GetColor	GFX_ColorGet()
TransparentColorEnable()	GFX_TransparentColorSet()
GetTransparentColor()	GFX_TransparentColorGet()
TransparentColorEnable()	GFX_TransparentColorEnable()
TransparentColorDisable()	GFX_TransparentColorDisable()
Bar()	GFX_BarDraw()

Header File: `gfx_palette.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
PaletteInit()	GFX_PaletteInit()
EnablePalette()	GFX_PaletteEnable()
DisablePalette()	GFX_PaletteDisable()
IsPaletteEnabled()	GFX_IsPaletteEnabled()
GetPaletteChangeError()	GFX_PaletteChangeErrorGet()
ClearPaletteChangeError()	GFX_PaletteChangeErrorClear()
SetPaletteBpp()	GFX_PaletteBppSet()
RequestPaletteChange()	GFX_PaletteChangeRequest()
RequestEntirePaletteChange()	GFX_EntirePaletteChangeRequest()
SetPalette()	GFX_PaletteSet()
SetEntirePalette()	GFX_EntirePalette()
SetPaletteFlash()	GFX_PaletteFlashSet()

Header File: `gfx_gol.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
GetState()	GFX_GOL_ObjectStateGet()
ClearState()	GFX_GOL_ObjectStateClear()
SetState()	GFX_GOL_ObjectStateSet()
GOLAddObject()	GFX_GOL_ObjectAdd()
GOLFindObject()	GFX_GOL_ObjectFind()
GOLRedraw()	GFX_GOL_ObjectListRedraw()
GOLRedrawRec()	GFX_GOL_ObjectRectangleRedraw()
GOLDraw()	GFX_GOL_ObjectListDraw()
GOLDrawComplete()	GFX_GOL_ObjectDrawComplete()
GOLDrawCallback()	GFX_GOL_DRAW_CALLBACK_FUNC
GOLFree()	GFX_GOL_ObjectListFree()
GetObjType()	GFX_GOL_ObjectTypeGet()
GetObjID	GFX_GOL_ObjectNextGet()
GetObjNext()	GFX_GOL_ObjectNext()
GOLDeleteObject()	GFX_GOL_ObjectDelete()

GOLDeleteObjectByID()	GFX_GOL_ObjectByIDDelete()
GOLNewList()	GFX_GOL_ObjectListNew()
GOLGetList()	GFX_GOL_ObjectListGet()
GOLSetFocus()	GFX_GOL_ObjectFocusSet()
IsObjUpdated()	GFX_GOL_ObjectUpdated()
GOLInit()	GFX_Initialize()
GOLGetFocus()	GFX_GOL_ObjectFocusGet()
GOLCanBeFocused()	GFX_GOL_ObjectCanBeFocused()
GOLGetFocusNext()	GFX_GOL_ObjectFocusNextGet()
GOLGetFocusPrev()	GFX_GOL_ObjectFocusPrevGet()
GOLPanelDraw()	GFX_GOL_PanelParameterSet()
GOLPanelDrawTsk()	GFX_GOL_PanelDraw()
GOLTwoTonePanelDraw()	GFX_GOL_PanelTwoToneDraw()

Header File: `gfx_gol_button.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
BtnCreate()	GFX_GOL_ButtonCreate()
BtnDraw()	GFX_GOL_ButtonDraw()
BtnGetText()	GFX_GOL_ButtonTextGet()
BtnSetText()	GFX_GOL_ButtonTextSet()
BtnGetBitmap()	GFX_GOL_ButtonPressStateImageGet()
BtnSetBitmap()	GFX_GOL_ButtonPressStateImageSet()
BtnMsgDefault()	GFX_GOL_ButtonActionSet()
BtnTranslateMsg()	GFX_GOL_ButtonActionGet()

Header File: `gfx_gol_check_box.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
CbCreate()	GFX_GOL_CheckBoxCreate()
CbDraw()	GFX_GOL_CheckBoxDraw()
CbGetText()	GFX_GOL_CheckBoxTextGet()
CbSetText()	GFX_GOL_CheckBoxTextSet()
CbMsgDefault()	GFX_GOL_CheckBoxActionSet()
CbTranslateMsg()	GFX_GOL_CheckBoxActionGet()

Header File: `gfx_gol_custom_control.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
No equivalent	GFX_GOL_CustomControlGetPos ()
No equivalent	GFX_GOL_CustomControlSetPos ()
No equivalent	GFX_GOL_CustomControlCreate
No equivalent	GFX_GOL_CustomControlActionGet ()
No equivalent	GFX_GOL_CustomControlActionGet ()
No equivalent	GFX_GOL_CustomControlActionSet ()
No equivalent	GFX_GOL_CustomControlDraw ()

Header File: `gfx_gol_digital_meter.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
DmCreate()	GFX_GOL_DigitalMeterCreate()

DmDraw()	GFX_GOL_DigitalMeterDraw()
DmGetValue()	GFX_GOL_DigitalMeterValueGet()
DmSetValue()	GFX_GOL_DigitalMeterValueSet()
DmDecVal()	GFX_GOL_DigitalMeterDecrement()
DmIncVal()	GFX_GOL_DigitalMeterIncrement()
DmTranslateMsg()	GFX_GOL_DigitalMeterActionGet()

Header File: `gfx_gol_edit_ box.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
EbCreate()	GFX_GOL_EditBoxCreate()
EbDraw()	GFX_GOL_EditBoxDraw()
EbGetText()	GFX_GOL_EditBoxTextGet()
EbSetText()	GFX_GOL_EditBoxTextSet()
EbAddChar()	GFX_GOL_EditBoxCharAdd(()
EbDeleteChar()	GFX_GOL_EditBoxCharDelete()
EbMsgDefault()	GFX_GOL_EditBoxActionSet()
EbTranslateMsg()	GFX_GOL_EditBoxActionGet()

Header File: `gfx_gol_group_box.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
GbCreate()	GFX_GOL_GroupBoxCreate()
GbDraw()	GFX_GOL_GroupBoxDraw()
GbGetText()	GFX_GOL_GroupBoxTextGet()
GbSetText()	GFX_GOL_GroupBoxTextGet()
GbTranslateMsg()	GFX_GOL_GroupBoxActionGet()
GbCreate()	GFX_GOL_GroupBoxCreate()
GbDraw()	GFX_GOL_GroupBoxDraw()

Header File: `gfx_gol_list_box.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
LbCreate()	GFX_GOL_ListBoxCreate()
LbDraw()	GFX_GOL_ListBoxDraw()
LbGetItemList()	GFX_GOL_ListBoxItemListGet()
LbAddItem()	GFX_GOL_ListBoxtemAdd()
LbAddItem()	GFX_GOL_ListBoxItemAdd()
LbDelItem()	GFX_GOL_ListBoxItemRemove()
LbChangeSel()	GFX_GOL_ListBoxChangeSelection()
LbSetSel()	GFX_GOL_ListBoxSelectionSet()
LbGetSel()	GFX_GOL_ListBoxSelectionGet()
LbGetFocusedItem()	GFX_GOL_ListBoxFocusedItemGet()
LbSetFocusedItem()	GFX_GOL_ListBoxFocusedItemSet()
LbGetCount(pLb)	GFX_GOL_ListBoxCountGet()
LbGetVisibleCount()	GFX_GOL_ListBoxVisibleCountGet()
LbSetBitmap()	GFX_GOL_ListBoxBitmapSet()
LbGetBitmap(pItem)	GFX_GOL_ListBoxBitmapSet()

Header File: gfx_gol_meter.h

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
MtrCreate()	GFX_GOL_MeterCreate()
MtrDraw()	GFX_GOL_MeterDraw()
MtrSetVa()	GFX_GOL_MeterValueSet()
MtrGetVal()	GFX_GOL_MeterValueGet()
MtrDecVal()	GFX_GOL_MeterDecrementl()
MtrIncVal	GFX_GOL_MeterIncrement()
MtrSetScaleColors	GFX_GOL_MeterScaleColorsSet()
MtrSetTitleFont(pMtr, pNewFont)	GFX_GOL_MeterTitleFontSet()
MtrSetValueFont(pMtr, pNewFont)	GFX_GOL_MeterValueFontSet()
MtrMsgDefault()	GFX_GOL_MeterDefaultActionSet()
MtrTranslateMsg()	GFX_GOL_MeterActionGet()

Header File: `gfx_gol_picture.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
PictCreat()	GFX_GOL_PictureControlCreate()
PictDraw()	GFX_GOL_PictureControlDraw()
PictSetBitmap()	GFX_GOL_PictureControlBitmapSet()
PictGetBitmap()	GFX_GOL_PictureControlBitmapGet()
PictGetScale()	GFX_GOL_PictureControlScaleGet()
PictSetScale()	GFX_GOL_PictureControlScaleGet()
PictTranslateMsg()	GFX_GOL_PictureControlActionGet()

Header File: `gfx_gol_progress_bar.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
PbCreate()	GFX_GOL_ProgressBarCreate()
PbDraw()	GFX_GOL_ProgressBarDraw()
PbSetRange()	GFX_GOL_ProgressBarRangeSet()
PbGetRange()	GFX_GOL_ProgressBarRangeGet()
PbSetPos()	GFX_GOL_ProgressBarPositionSet()
PbGetPos()	GFX_GOL_ProgressBarActionSet()
PbTranslateMsg()	GFX_GOL_ProgressBarActionGet()

Header File: `gfx_gol_radio_button.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
RbCreate()	GFX_GOL_RadioButtonCreate()
RbDraw()	GFX_GOL_RadioButtonDraw()
RbGetCheck()	GFX_GOL_RadioButtonCheckGet()
RbSetCheck()	GFX_GOL_RadioButtonCheckGet()
RbGetText(pRb)	GFX_GOL_RadioButtonTextGet()
RbSetText()	GFX_GOL_RadioButtonTextGet()
RbMsgDefault()	GFX_GOL_RadioButtonActionSet()

Header File: `gfx_gol_scrollbar.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
SldCreate()	GFX_GOL_ScrollBarCreate()
SldDraw()	GFX_GOL_ScrollBarDraw()

SldSetPage()	GFX_GOL_ScrollBarPageSet()
SldGetPage()	GFX_GOL_ScrollBarPageGet()
SldSetPos()	GFX_GOL_ScrollBarPositionSet()
SldGetPos(pSld)	GFX_GOL_ScrollBarPositionGet()
SldSetRange()	GFX_GOL_ScrollBarSetRange()
SldGetRange()	GFX_GOL_ScrollBarRangeGet()
SldIncPos()	GFX_GOL_ScrollBarPositionIncrement()
SldDecPos()	GFX_GOL_ScrollBarPositionDecrement()
SldMsgDefault()	GFX_GOL_ScrollBarActionSet()

Header File: `gfx_gol_static_text.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
StCreate()	GFX_GOL_StaticTextCreate()
StDraw()	GFX_GOL_StaticTextDraw()
StGetText()	GFX_GOL_StaticTextGet()
StSetText()	GFX_GOL_StaticTextSet()
StTranslateMsg()	GFX_GOL_StaticTextActionGet()

Header File: `gfx_gol_text_entry.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
TeCreate()	GFX_GOL_TextEntryCreate()
TeDraw()	GFX_GOL_TextEntryDraw()
TeGetBuffer()	GFX_GOL_TextEntryBufferGet()
TeSetBuffer()	GFX_GOL_TextEntryBufferSet()
TeClearBuffer()	GOL_GOL_TextEntryBufferClear()
TeGetKeyCommand()	GOL_GOL_TextEntryKeyCommandGet()
TeSetKeyCommand()	GOL_GOL_TextEntryKeyCommandSet()
TeCreateKeyMembers()	GOL_GOL_TextEntryKeyMembersCreate()
TeAddChar()	GOL_GOL_TextEntryCharAdd()
TeIsKeyPressed()	GOL_GOL_TextEntryKeyPressed()
TeSpaceChar()	GOL_GOL_TextEntryCharSpace()
TeDelKeyMembers()	GOL_GOL_TextEntryKeyMembersDelete()
TeSetKeyText()	GOL_GOL_TextEntrySetKeyText()
TeMsgDefault()	GOL_GOL_TextEntryActionSet()
TeTranslateMsg()	GOL_GOL_TextEntryActionGet()

Header File: `gfx/gfx_gol_window.h`

MLA Graphics Library Function Name	MPLAB Harmony Graphics Library Function Name
WndCreate()	GFX_GOL_WindowCreate()
WndDraw()	GFX_GOL_WindowDraw()
WndGetText()	GFX_GOL_WindowTextGet()
WndSetText()	GFX_GOL_WindowTextSet()
WndTranslateMsg()	GFX_GOL_WindowActionGet()

Using the Library

This topic describes the architecture of the Graphics Library and the components in the Primitive and Object Layers. Information on how to use the

Primitive Layer and Object Layer is also provided.

Description

The Graphics Library is composed of the following layers:

- Application Layer - This is the program that utilizes the Graphics Library
- User Message Interface - This is a layer should be implemented by user to provide messages for the library
- Graphics Object Layer - This layer renders the control objects such as button, list box, progress bar, meter and so on
- Graphics Primitives Layer - This layer implements the primitive rendering functions
- Device Display Driver - This layer is the graphics display driver component that is optimized to the actual display module used
- Graphics Display Module - This is the actual display module

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Graphics Library.

Library Interface Section	Description
Graphics Primitive Layer Functions	Functions for managing the Primitive Layer of the Graphics Library
Graphics Object Layer Functions	Functions for managing the Object Layer of the Graphics Library
Graphics Functions	Graphics Library configuration functions.

Graphics Objects

This section describes the basics of the Graphics Object Layer.

Description

The Graphics Object Layer is composed of objects that can act as control widgets in an application. There are several predefined objects in the library. Users are not limited to these predefined objects. It is possible to create custom objects. Creating custom objects will require a deeper understanding of the library's architecture. Creating custom objects is described in the Microchip application note, AN1136 *"How to Use Widgets in Microchip Graphics Library*" (DS01136).

Each object will have its own create function. In the create function the following functions are initialized:

- DRAW_FUNC - This is the function that renders the object on the screen
- FREE_FUNC - This is the function that frees the memory used by the object when the object is removed from memory
- ACTIONGET_FUNC - This optional function returns the a translated action of the object. A translated action is actually a planned action that indicates the intended change in the state of the object. The translated action is basically the response of the object to the user interaction on the object. For example, when a user presses on a button object on a screen, a valid planned action of the object is to change its state to a pressed state. Translated actions are specific to each object. See the GFX_GOL_TRANSLATED_ACTION enumeration for the listing of all translated actions of all objects in the library. This list does not include user-defined actions. Translated actions are determined using the messaging scheme of the library. Messaging is discussed in a separate section of this documentation. When this function is not defined, the create function of the object must set the function pointer to NULL.
- ACTIONSET_FUNC - This optional function performs the translated action returned by the ACTIONGET_FUNC. This is the actual function that changes the state of the object. When this function is not defined, the create function of the object must set the function pointer to NULL.

Each of these functions are implemented in each object. The draw function of the object is specific to the object. The draw function should not be called directly by the application. Instead, applications should use the GFX_GOL_ObjectListDraw function so that the management of the object rendering will be done by the Graphics Object Layer.

Object Layer Rendering

This section describes how objects are rendered.

Description

Object List Rendering

Initially, an application will create the objects that it will use by calling the create function of the objects. As the objects are created, a linked list of objects is also created. This linked list of objects is the list that the library will use to manage the rendering of the objects. The application will only need to call the GFX_GOL_ObjectListDraw function to execute the rendering of all the objects. It is also through this list that the library is able to parse all the objects to determine if an object is affected by the user action. This is discussed in the Object Layer Messaging section.

Each object, has a state variable that is composed of state bits. There are two main groups in the state bits:

- Property State bits - define the general shape of the object
- Draw State bits - determine if the object will be redrawn. The redrawing can be a full object redraw or partial object redraw. Depending on the

object, definition, a partial redraw means that only portion of the object is redrawn. This makes the redraw of the object fast and efficient. For example, redrawing the full progress bar object will be longer than just updating the level of the progress bar.

The Property State bits are usually defined once at the object creation. Conversely, the Draw State bits, dynamically change and usually change because of user action on the object. These user actions are encoded as messages. The messages are created in user interface drivers such as touch screen or key pads on the system (see the Object Layer Messaging section for details).

The objects in the library implements the rendering sequence using rendering state machine. The rendering state is not to be confused with the Draw State Bits of the objects. Rendering states are defined in each object's rendering function. An object is basically drawn using combination of primitive calls. Bars, lines, and text can be drawn to represent an object. The purpose of these rendering states is to divide the primitive calls into steps. By doing this, each primitive call is now a step in the process. If one of the primitive call cannot proceed due to busy hardware resource, the object's rendering sequence can be paused. Since, the length of the delay in the rendering cannot be predicted, it is best to pause the rendering of the object and give back control of the CPU to application. Each of the object's rendering function coupled with GFX_GOL_ObjectListDraw, has this capability. This effectively pauses the rendering of the object.

For the rendering to continue, GFX_GOL_ObjectListDraw is called again by the application. When this is called, the specific object rendering function that was called is then resumed and in the object rendering function, the specific step (or primitive call) is executed again. This basically continues the rendering exactly where it left off.

Object Draw Callback Feature

To an allow application to control the rendering of the objects, a callback function is provided (see GFX_GOL_DRAW_CALLBACK_FUNC). The callback function is an application implemented function. The library provides the GFX_GOL_DrawCallbackSet(GFX_GOL_DRAW_CALLBACK_FUNC) API to set the callback function. GFX_GOL_DRAW_CALLBACK_FUNC parameter in this function is the pointer to the defined callback.

The callback allows the application to insert application defined rendering.

For example, a digital signal is sampled on a pin. The value of the signal determines if a scroll bar will increment or decrement its value. To implement the change in the value of a scroll bar, a callback function can be implemented to do that.

```
void App_Tasks()
{
   switch(appData.state)
    {
        case APP_STATE_GFX_READY:
            if ( GFX_Status (gfxObject) == SYS_STATUS_READY )
                appData.state = APP_STATE_GFX_INIT;

            break;

        case APP_STATE_GFX_INIT:

            GFX_GOL_MessageCallbackSet(GFX_INDEX_0, &GFX_GOL_MessageCallback);
            GFX_GOL_DrawCallbackSet(GFX_INDEX_0, &GFX_GOL_DrawCallback);

            appData.state = APP_STATE_RUNNING;
            break;
    }
}

//*****************************************************************
// The draw callback is called within GFX_GOL_ObjectListDraw()
//*****************************************************************
bool APP_ObjectDrawCallback(void)
{
// assume pScrollBar is pointer to the scroll bar
bool signalAssert;
// assume signal is sampled at a pin RC1
if (PORTCbits.RC1 == 1)
signalAssert = true;
else
signalAssert - false;
if (signalAssert == true)
{
// increment the value
GFX_GOL_ScrollBarPositionIncrement(pScrollBar);
}
else
{
// increment the value
GFX_GOL_ScrollBarPositionDecrement(pScrollBar);
}
// redraw only the thumb
GFX_GOL_ObjectStateSet(pScrollBar, GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE);
```

```
return (true);
}
```

Since the callback is called in GFX_GOL_ObjectListDraw and GFX_GOL_ObjectListDraw is called in the main loop, the callback can potentially be executed unnecessarily if the sampled signal frequency of change is slow. To counter that, the callback can be modified to be sampling the signal after some predefined time delay.

```
//*****************************************************************
// The draw callback is called within GFX_GOL_ObjectListDraw()
//*****************************************************************
bool APP_ObjectDrawCallback(void)
{
// assume pScrollBar is pointer to the scrollbar
// assume tick is a system counter, DEFINED_DELAY is
// a predefined delay tuned to the signal frequency
bool signalAssert;
static uint32_t prevtick = 0;
if (tick - prevtick > DEFINED_DELAY)
{
prevtick = tick;
// assume signal is sampled at a pin RC1
if (PORTCbits.RC1 == 1)
signalAssert = true;
else
signalAssert - false;
if (signalAssert == true)
{
// increment the value
GFX_GOL_ScrollBarPositionIncrement(pScrollBar);
}
else
{
// increment the value
GFX_GOL_ScrollBarPositionDecrement(pScrollBar);
}
// redraw only the thumb
GFX_GOL_ObjectStateSet(pScrollBar, GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE);
}
return (true);
```

Object Layer Messaging

This section describes the basics of object messages.

Description

User inputs can also change the state of objects. This is done through messaging. The library passes user action to the objects using the message structure (see GFX_GOL_MESSAGE). The purpose of the messaging is to translate user action (i.e., a user press on a touch screen) to a change in state of the objects in the screen. At the same time, this user action will cause an application function to execute.

For example, a press on a button on the screen will result in an LED being turned on. While pressing on the button, the button on the screen also changes states and is redrawn to a pressed state. When the user releases the button, the button redraws back to an unpressed state and the LED turns off.

The mechanism of messaging and object state change is explained in the next section.

Messaging Structure

The messages are encoded in the GFX_GOL_MESSAGE structure.

```
    typedef struct
    {
        uint8_t type;           // Specifies the type of input device.
        uint8_t uiEvent;        // An event that occurred in the input device.
        int16_t param1;         // Parameter 1, definition and usage is
                                // dependent on the type of input device.
        int16_t param2;         // Parameter 2, definition and usage is
                                // dependent on the type of input device.
    } GFX_GOL_MESSAGE;
```

The type defines the type of input device. See INPUT_DEVICE_TYPE for the currently supported type of devices. The uiEvent is also a set of predefined events. See INPUT_DEVICE_EVENT for a list of supported events. The two parameters, param1 and param2, change in usage depending on the type of input device and event.

The messages are encoded in the drivers for the input devices. The drivers provide functions for the application to call. These functions should

return the message whenever there is a valid user action on the input device.

The following code example shows TouchGetMsg as the function provided by the touch screen driver to retrieve user inputs on the touchscreen.

```
    // GOL message structure
    GFX_GOL_MESSAGE msg;

    ...
    // set the message callback function pointer
    GFX_GOL_MessageCallbackSet(APP_ObjectMessageCallback);
    ...

    uint16_t nQSizes[] = SYS_MSG_BUFFER_SIZES;

    sSysMsgInit.nQSizes = nQSizes;
    hSysMsg = SYS_MSG_Initialize(iSysMsg,(SYS_OBJ_HANDLE)&sSysMsgInit);
    hMsgType = SYS_MSG_TypeCreate(iSysMsg, TYPE_TOUCHSCREEN , 0);
    hMailbox = SYS_MSB_MailboxOpen( iSysMsg, (SYS_MSG_RECEIVE_CALLBACK)&TouchMessageCallback );
    SYS_MSG_MailboxMsgAdd(hMailbox, hMsgType);

void TouchMessageCallback(SYS_MSG_OBJECT *pMsg)
{
    static GFX_GOL_MESSAGE gMsg;

    gMsg.type = pMsg->nMessageTypeID;
    gMsg.uiEvent = pMsg->param0;
    gMsg.param1 = pMsg->param1;
    gMsg.param2 = pMsg->param2;

    GFX_GOL_ObjectMessage(GFX_INDEX_0, &gMsg);
}
```

Graphics Object Message Function

After the input device driver encodes the message, the message is sent by the application to the Graphics Library through the GFX_GOL_ObjectMessage. This function parses the active list of objects and determines if the message affects one or more of the objects. The mechanism to check each object is simple: the GFX_GOL_ObjectMessage calls each of the object's ACTIONGET_FUNC. The ACTIONGET_FUNC of an object is the function that determines if the message affects the object. This function returns a translated action (see the GFX_GOL_TRANSLATED_ACTION enumeration).

If the object is not affected it replies with GFX_GOL_OBJECT_ACTION_INVALID. If the object is affected, it replies with the appropriate translated action.

Object Message Callback Feature

If the object is affected, GFX_GOL_ObjectMessage() then calls the message callback function. The message callback function is an application defined function. This is set initially using the GFX_GOL_MessageCallbackSet(GFX_GOL_MESSAGE_CALLBACK_FUNC) call where GFX_GOL_MESSAGE_CALLBACK_FUNC is a pointer to the defined callback function.

The message callback is an opportunity for the application to respond to user inputs. It is also an opportunity to block the library from performing default state change to objects. See the callback function documentation for details.

The message callback returns true if the application wants to perform the default action set of the objects. If the callback returns false, the application assumes all changes in the states of the object.

After the callback is executed, with a return value of true, the ACTIONSET_FUNC of the affected object is called by GFX_GOL_ObjectMessage. This function performs the state change of the object due to the user input. The object is then redrawn when GFX_GOL_ObjectListDraw is called by the application.

The following code example demonstrates usage of the message callback function. A scroll bar is incremented and decremented by two buttons.

```
//*****************************************************************
// The message callback is called within GFX_GOL_ObjectMessage()
//*****************************************************************

// assume that two buttons and a scroll bar are present in the system
bool APP_MsgCallback(
            uint16_t objMsg,
            GFX_GOL_OBJ_HEADER *pObj,
            GFX_GOL_MESSAGE *pMsg)
{
    uint16_t objectID;
    GFX_GOL_SCROLLBAR *pScrollBar;
    objectID = GFX_GOL_ObjectIDGet(pObj);
    if(objectID == ID_BTN1)
    {
```

```
            // check if button is pressed
            if(objMsg == GFX_GOL_BUTTON_ACTION_PRESSED)
            {
                // find slider pointer
                pScrollBar = (GFX_GOL_SCROLLBAR *)GFX_GOL_ObjectFind(ID_SLD1);
                // decrement the slider position
                GFX_GOL_ScrollBarPositionDecrement(pScrollBar);
                // redraw only the thumb
                GFX_GOL_ObjectStateSet(pScrollBar, GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE);
            }
        }
        if(objectID == ID_BTN2)
        {
            if(objMsg == GFX_GOL_BUTTON_ACTION_PRESSED)
            {
                // find slider pointer
                pScrollBar = (GFX_GOL_SCROLLBAR *)GFX_GOL_ObjectFind(ID_SLD1);
                // increment the slider position
                GFX_GOL_ScrollBarPositionIncrement(pScrollBar);
                // redraw only the thumb
                GFX_GOL_ObjectStateSet(pScrollBar, GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE);
            }
        }
        return (true);
    }
```

Notice that the callback does not check if the scroll bar was affected by the message. Since the callback returns true, the scroll bar ACTIONSET_FUNC will be the one that will perform the state change of the scroll bar if the user directly interacted with the scroll bar.

Graphics Object Action Get and Action Set Functions

Each object has a predefined translated actions (see the GFX_GOL_TRANSLATED_ACTION enumeration). These actions are returned by the object's ACTIONGET_FUNC. The ACTIONGET_FUNC is called by the GFX_GOL_ObjectMessage function.

ACTIONGET_FUNC evaluates the message to determine if the user action affected the object. If the object is affected, it replies with one of the predefined translated action of the object.

For a button object the following translated actions are returned when the button is affected:

- GFX_GOL_BUTTON_ACTION_PRESSED - button is pressed
- GFX_GOL_BUTTON_ACTION_STILLPRESSED - button is continuously pressed
- GFX_GOL_BUTTON_ACTION_RELEASED - button is released
- GFX_GOL_BUTTON_ACTION_CANCELPRESS - button press canceled

When the button is not affected it returns:

- GFX_GOL_OBJECT_ACTION_INVALID - invalid message response

All objects returns GFX_GOL_OBJECT_ACTION_INVALID when they are not affected by the message.

the ACTIONSET_FUNC on the other hand performs the actual state change of the object. The draw state bits of the object are modified to indicate that the object will be redrawn in the next call to the GFX_GOL_ObjectListDraw.

How the Library Works

This section describes how the Graphics Library works and how the Primitive and Object layers are used.

Using the Primitive Layer

This section contains information on how to use the Primitive Layer of the Graphics Library.

Line Rendering

This section describes how to render lines.

Description

When rendering lines, two things need to be set:

- Set the color (GFX_ColorSet)
- Set the line style (GFX_LineStyleSet)

After the color and style is set, GFX_LineDraw can be called to render the line. GFX_LineDraw can be called multiple times to render lines that have the same color and line style. GFX_ColorSet and GFX_LineStyleSet will only be called when the color or the line style needs to be changed. For example, when one of the lines to be rendered has to change color, the GFX_ColorSet function must be called to change the

color.

Note: Alpha blended lines as well as anti-aliased lines are not yet supported.

Polygon Rendering

This section describes how to render polygons.

Unfilled Polygon Rendering

This section describes how to render unfilled polygons.

Description

Unfilled polygons are rectangles and rounded rectangles. Circles is a special case of a rounded rectangle. Rendering unfilled polygons uses the line styles.

Similar to rendering lines, two things need to be set and one action performed:

- Set the color (GFX_ColorSet)
- Set the line style (GFX_LineStyleSet)
- Call the specific polygon function

Filled Polygon Rendering

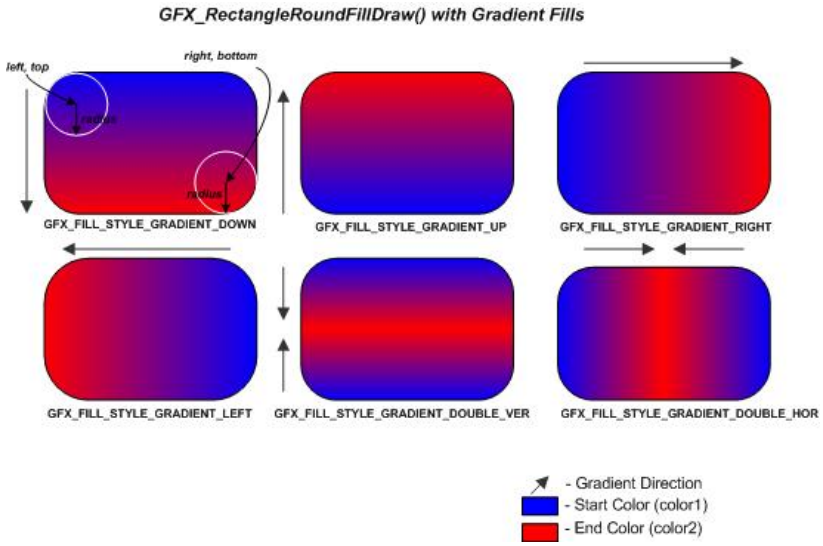
This section describes how to render filled polygons.

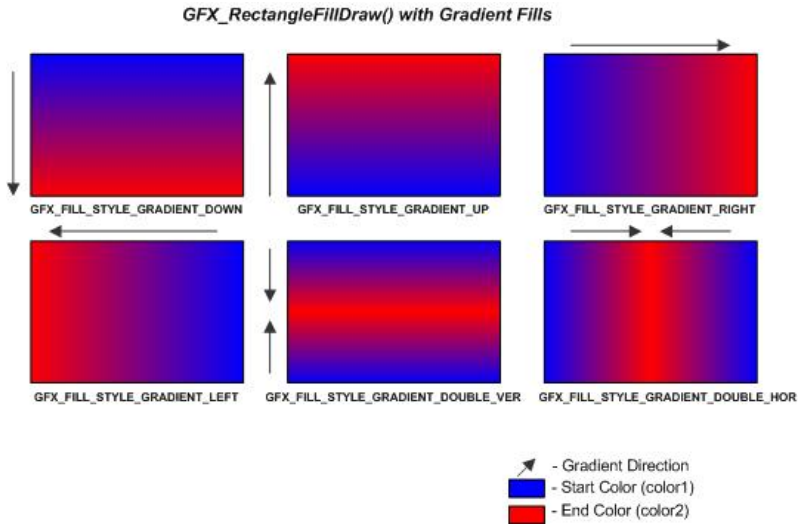
Description

Filled polygons are filled rectangles and filled rounded rectangles. Bar is a special case of a filled rectangle and filled circle is a special case of a filled rounded rectangle.

For filled polygons, the fill styles can be set:

Fill Style	Description
GFX_FILL_STYLE_NONE	The fill will be set to no color.
GFX_FILL_STYLE_ALPHA_COLOR	Sets that the fill style is alpha blended. The alpha blending effect will be dependent on the background type set (see GFX_BackgroundTypeSet). The level of alpha blending will depend on the last GFX_AlphaBlendingValueSet call.
GFX_FILL_STYLE_GRADIENT_DOWN	The effect is a gradient fill in the vertical down direction.
GFX_FILL_STYLE_GRADIENT_UP	The effect is a gradient fill in the vertical up direction.
GFX_FILL_STYLE_GRADIENT_RIGHT	The effect is a gradient fill in the horizontal right direction.
GFX_FILL_STYLE_GRADIENT_LEFT	The effect is a gradient fill in the horizontal left direction.
GFX_FILL_STYLE_GRADIENT_DOUBLE_VER	The effect is a gradient fill in the vertical direction but with two transitions.
GFX_FILL_STYLE_GRADIENT_DOUBLE_HOR	The effect is a gradient fill in the horizontal direction but with two transitions.





To render alpha blended filled polygons:

1. Set the color (GFX_ColorSet).
2. Set the background type (GFX_BackgroundTypeSet).
3. Set the alpha blending value (GFX_AlphaBlendingValueSet).
4. Set the fill style to alpha blending (GFX_FillStyleSet).
5. Call the specific polygon fill function:

- GFX_RectangleFillDraw
- GFX_RectangleRoundFillDraw
- GFX_BarDraw
- GFX_CircleFillDraw

To render gradient filled polygons:

1. Set the color (GFX_ColorSet).
2. Set the two gradient start and end colors (GFX_GradientColorSet).
3. Set the fill style to one of the gradient fill types (GFX_FillStyleSet).
4. Call the specific polygon fill function:

- GFX_RectangleFillDraw
- GFX_RectangleRoundFillDraw
- GFX_BarDraw
- GFX_CircleFillDraw

The following functions renders the same shapes:

- GFX_RectangleRoundDraw(x, y, x, y, radius) is equivalent to GFX_CircleDraw(x, y, radius).
- GFX_RectangleRoundFillDraw(x, y, x, y, radius) is equivalent to GFX_CircleFillDraw(x, y, radius).
- GFX_RectangleFillDraw(x1, y1, x2, y2) is equivalent to GFX_BarDraw(x1, y1, x2, y2).

Notes:

1. Alpha blended unfilled polygons are not yet supported.
2. Anti-aliased unfilled polygons are not yet supported.
3. Alpha blended, gradient fills are not yet supported.
4. Background is not needed when using gradient fills.

Text Rendering and Font Features

This section describes how to render text and strings and other features that are available for font resources.

Text Rendering

This section describes how to render text and strings.

Description

Text rendering in the Graphics Library requires a font table. The font table is considered one of the resources that is used in the library (see GFX_RESOURCE_FONT of the GFX_RESOURCE_HDR structure).

Once the font resource is available, text rendering can be performed by one of the following functions:

- GFX_TextCharDraw - Use to render a single character
- GFX_TextStringDraw - Use to render a string of characters
- GFX_TextStringBoxDraw - Use to render formatted string of characters

Character Rendering

To render a character:

1. Set the color (GFX_ColorSet).
2. Set the font resource to use (GFX_FontSet).
3. Set the location where the character will be rendered (GFX_TextCursorPositionSet).
4. Render the character (GFX_TextCharDraw).

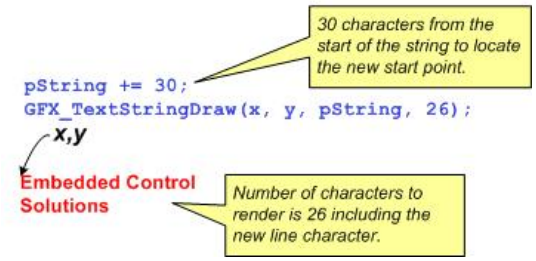
String Rendering

To render a string:

1. Set the color (GFX_ColorSet).
2. Set the font resource to use (GFX_FontSet).
3. Render the string (GFX_TextStringDraw).

GFX_TextStringDraw parameters sets the location where the string will be rendered. The length parameter allows versatility in rendering. When this parameter is set to '0', the rendering will terminate when the string terminator is encountered. When the length is non-zero, the rendering can be performed with the length as the terminating condition. This allows applications to render a portion of a longer string.

```
GFX_XCHAR Text[] =
"Microchip Technology Inc.
The Embedded Control
Solutions Company";
GFX_XCHAR *pString = Text;

GFX_TextStringDraw(x, y, pString, 0);
```



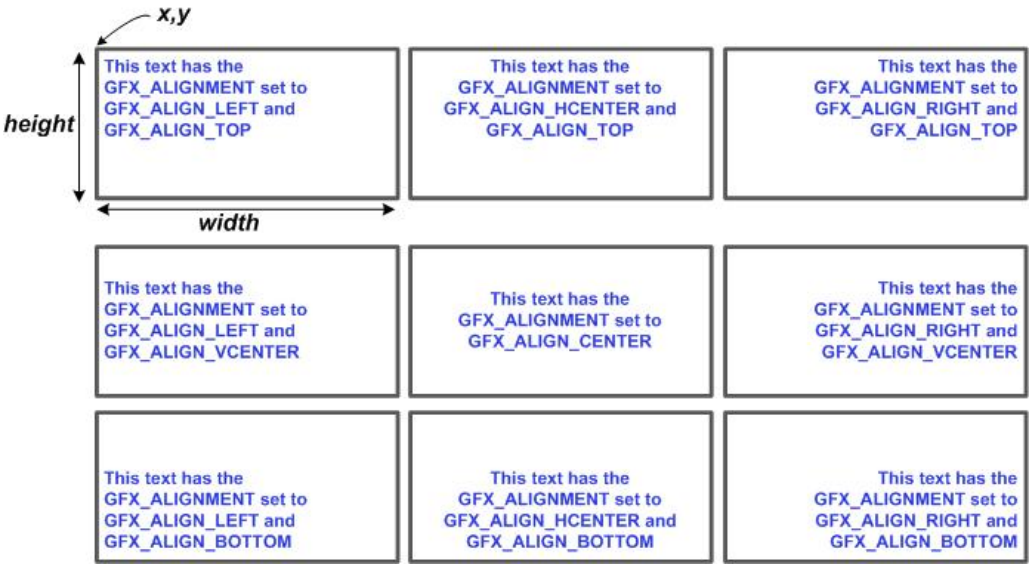
Multiple lines of strings are also supported. The string can contain the new line characters and rendering results in multiple lines of text. The use GFX_TextStringDraw will always render strings left aligned. To render text with alignment use GFX_TextStringBoxDraw.

Formatted String Rendering

To render string with formatting:

1. Set the color (GFX_ColorSet).
2. Set the font resource to use (GFX_FontSet).
3. Render the string (GFX_TextStringBoxDraw).

Parameters of GFX_TextStringBoxDraw determines the rectangular area where the string will be rendered. The alignment (see GFX_ALIGNMENT) specifies how the text will be aligned in the defined text area. The termination of the rendering is the same as the GFX_TextStringDraw where the length parameter can be used to determine how many characters will be rendered.



Added to these rendering functions, support functions are also provided for versatility:

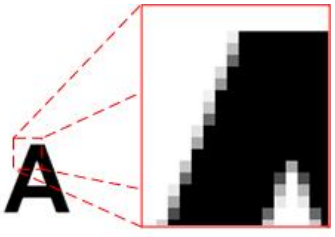
- GFX_TextStringWidthGet - Use to get the length of a string in pixels
- GFX_TextStringHeightGet - Use to get the height of the given font. All characters in a given font resource will have a constant height.
- GFX_FontSet - Use to set the font resource to use
- GFX_FontGet - Use to check the currently used font resource

Anti-aliased Fonts

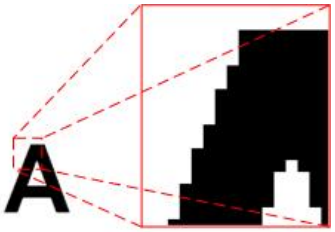
This section describes the anti-aliasing feature of the library.

Description

The Graphics Library supports rendering of anti-aliased font resource. Anti-aliasing is a technique used to make the edges of text appear smooth. This is useful especially with characters like 'A', 'O', etc., which has slant or curved lines. Since the pixels of the display are arranged in rectangular fashion, slant edges cannot be represented smoothly. To make them appear smooth, a pixel adjacent to the pixels is painted with an average of the foreground and background colors as depicted in the following figure.



When anti-aliasing is turned off, the pixels abruptly changes from background color to foreground color shown in the following figure. To implement anti-aliasing, adjacent pixels transitions from background to foreground color using 25% or 75% mid-color values from background to foreground colors. This feature in fonts will require roughly twice the size of memory storage required for font glyphs with no anti-aliasing.



Since the average of foreground and background colors needs to be calculated at runtime, the rendering of anti-aliased fonts take more time than rendering normal fonts.

Anti-aliasing transparency can be set to one of the following using GFX_FontAntiAliasSet (see GFX_FONT_ANTIALIAS_TYPE):

- GFX_FONT_ANTIALIAS_OPAQUE - Mid colors are calculated once while rendering each character which is ideal for rendering text over a constant background. This is the default setting of the library.
- GFX_FONT_ANTIALIAS_TRANSLUCENT - The mid values are calculated for every pixel and this feature is useful while rendering text over an image or on a non-constant color background.

As a result, rendering anti-aliased text takes longer with translucent type than compared to opaque type.

To use anti-aliasing, the font resource must contain glyphs that are anti-aliased enabled. The Graphics Resource Converter (GRC) tool that comes with the installation of the Graphics Library can generate such font resources.

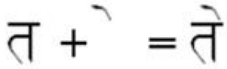
Note: An application may use anti-aliased and non-anti-aliased fonts at the same time.

Extended Glyphs

This section describes the extended glyphs feature of the library.

Description

Extended glyphs are needed to render characters of certain languages which use more than one byte to represent a single character. For example: Asian languages like Thai, Hindi, etc. In these character sets, more than one glyph overlaps each other to form a single character of that language, as shown in the following figure.



Extended glyph fonts can be generated using the Graphics Resource Converter (GRC) utility that is included with the installation of the Graphics Library.

Note: The fonts used with extended glyphs are normal ANSI fonts and not Unicode fonts.

Using the Graphics Object Layer

This section contains information on how to use the Graphics Object Layer of the Graphics Library.

Object Rendering and Style Schemes

This section describes how style schemes are used in rendering the objects of the Graphics Library.

Description

Style Scheme

All of the objects in the library utilize the style scheme to define how the object will be rendered on the screen. The style scheme defines the colors, images, fonts, and text that will be used. Choosing the appropriate colors and settings allows the object to be rendered in many ways.

Style schemes are applied to the objects by assigning the style scheme pointer to the object at creation. When the objects are rendered, the current settings of the style scheme will then be used to render the object. Objects cannot be created without a valid style scheme. This means that it cannot be null or undefined when the object is created. Style scheme can be modified anytime. It can be replaced using the GFX_GOL_ObjectStyleSchemeSet.

Style scheme is divided into these groups (See GFX_GOL_OBJ_SCHEME for details):

Text - The parameter that defines the text used in the object

- pFont - defines the font used in the style of the object

Colors - The parameters that defines the shape of the object

- EmbossDkColor - defines the dark emboss color
- EmbossLtColor - defines the light emboss color
- TextColor0 - defines the first text color option
- TextColor1 - defines the second text color option
- TextColorDisabled - defines the third text color option
- Color0 - defines the first face color option
- Color1 - defines the second face color option
- ColorDisabled - defines the third face color option
- EmbossDkColor - defines the dark emboss color

Background - The parameters that describes the background of the object. The background information defines how the object will be drawn with the background taken into account.

- CommonBkColor - Background color used to hide when the background type (GFX_BACKGROUND_TYPE) is set to GFX_BACKGROUND_COLOR or GFX_BACKGROUND_NONE
- CommonBkLeft - the horizontal starting position of the background
- CommonBkTop - the vertical starting position of the background
- CommonBkType - specifies the type of background to use
- pCommonBkImage - pointer to the background image used when the background type is set to GFX_BACKGROUND_IMAGE

Fill - The parameters that defines how the fill will be performed.

- fillStyle - specifies the fill style to be used (see GFX_FILL_STYLE)
- AlphaValue - alpha value used for alpha blending. This will be used then fillStyle is set to GFX_FILL_STYLE_ALPHA_COLOR.
- gradientStartColor - start color of the gradient fill. This will be used then fillStyle is set to any of the gradient fill styles.
- gradientEndColor - end color of the gradient fill. This will be used then fillStyle is set to any of the gradient fill styles.

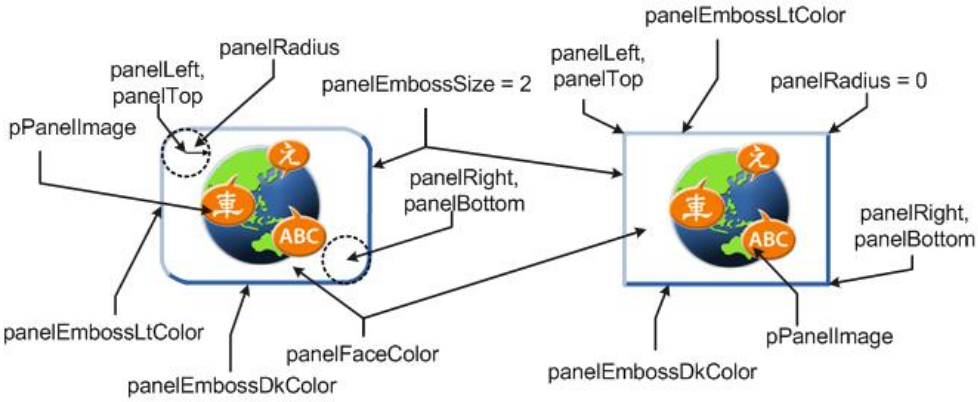
Style - The parameter that defines the 3-D effect of the object.

- EmbossSize - defines the emboss size of the panel for 3-D effect. This parameter should be set to '0' when not used.

Object Panel

Most of the objects in the Graphics Library utilizes a generic panel to define the object's shape. When using the provided objects in the library, the details of how the panel is utilized can be transparent to the users. In cases, where users defines their own objects the following detailed discussion of the panel is important.

A panel is a rectangular shape that is utilized by the objects to define its dimension. The panel is then drawn with the GOL_PANEL_PARAM to define the shape's colors. The colors are taken from the currently assigned style scheme to the object.



Panel Style

The assignment of the panel colors from the style scheme is shown in the following table.

Style Scheme to Panel Assignment

Style Scheme Parameter	Panel Parameter
EmbossDkColor	panelEmbossDkColor
EmbossLtColor	panelEmbossLtColor
Color0, Color1, ColorDisabled	panelFaceColor
fillStyle	panelFillStyle
EmbossSize	panelEmbossSize

TextColor0, TextColor1, TextColorDisabled, pFont, CommonBkColor, CommonBkLeft, CommonBkTop, CommonBkType, pCommonBkImage	Not set in the panel.
AlphaValue,	panelApha - use is dependent on the panelFillStyle setting
gradientStartColor,	panelGradientStartColor - use is dependent on the panelFillStyle setting
gradientEndColor	panelGradientEndColor - use is dependent on the panelFillStyle setting
N/A	panelImage - set by the object that supports images

The text parameters are not handled in the panel. These are handled by each object that supports text.

Refer to Panel Object for the summary of APIs that support panel rendering.

Background Feature

The Primitive Layer provides a global background information variable. This allows users to set a background and easily refresh parts of the screen that the background occupies (see GFX_BACKGROUND for details). The style scheme of the objects also allows association of the object to a background. This feature allows the objects to be easily refreshed on the screen.

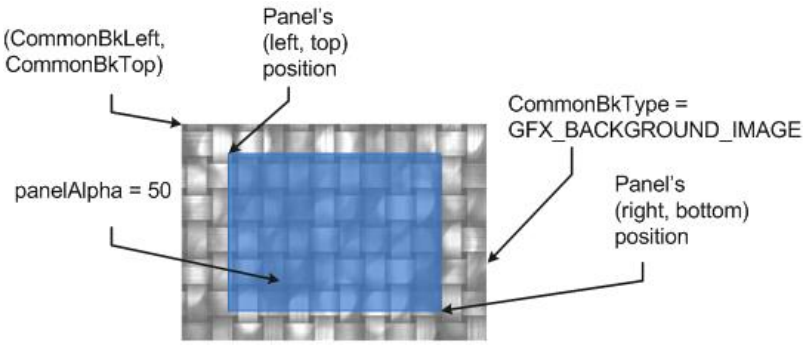
The requirement of the background feature is that the background should encompass the object. In other words, all the pixels of the object should be within the background dimension. Once this requirement is fulfilled, the parameters of the object that defines the location of the object (left, top, right bottom) is used to refresh the background pixels. Because of this Alpha blending, hiding or basic refresh of the object is optimized.

To add background information on the object the following style scheme parameters should be set:

- CommonBkLeft - The left most pixel location of the background
- CommonBkTop - The top most pixel location of the background
- CommonBkType - Should be set to one of the GFX_BACKGROUND_TYPE
- pCommonBkImage - If the CommonBkType is set to GFX_BACKGROUND_IMAGE, this should be set to the location of the image resource

Alpha Blending

When alpha blending is enabled, the object can be alpha blended with the background. Objects can also be removed from the screen easily by redrawing only the areas that the object occupies.



Alpha Blended Panel

Alpha Blending Objects

To implement an alpha blended object the following style scheme parameters should be set:

- fillStyle - Set to GFX_FILL_STYLE_ALPHA_COLOR
- AlphaValue - Set the alpha blending value. Set to 25, 50 or 75 when using primitive layer implementation of alpha blending

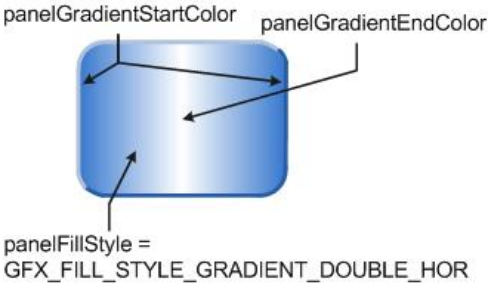
Gradient Fills

Objects can also be rendered with gradient fill on the panels. However, enabling gradient fill that are alpha blended is not supported yet. When AlphaValue is not 0 or 100 and fillStyle set to any of the gradient fill types, the alpha value will be ignored and assumed to be 100.

To implement an gradient filled object the following style scheme parameters should be set:

- fillStyle - Set to any of the gradient fill style (see GFX_FILL_STYLE for details):
- GFX_FILL_STYLE_GRADIENT_DOWN
- GFX_FILL_STYLE_GRADIENT_UP
- GFX_FILL_STYLE_GRADIENT_RIGHT
- GFX_FILL_STYLE_GRADIENT_LEFT
- GFX_FILL_STYLE_GRADIENT_DOUBLE_VER
- GFX_FILL_STYLE_GRADIENT_DOUBLE_HOR

- gradientStartColor - Set the gradient fill start color.
- gradientEndColor - Set the gradient fill end color.



Gradient Filled Panel

Notes:

1. Use of alpha blending on overlapping objects is not supported.
2. Alpha blended gradient fills is not supported.
3. Background feature can only be used if the background encompass the dimension of the object.
4. Certain objects do not support alpha blending and/or gradient fills. Refer to specific object documentation for details.

Double-Buffering Features

This section describes how to enable and use double buffering in the Graphics Primitive Layer.

Description

Manipulating pixels on the screen requires direct writes to the Frame Buffer. While these changes are being executed, the screen is also being refreshed. This means that the changes are displayed immediately as the frame buffer is being updated.

This is not a suitable scheme when the changes are slow, for example, decoding an image or having a large number of widgets on a screen. The display may appear slow in that case. One solution to this problem is to use a double-buffering scheme supported by the Microchip Graphics Library.

The double-buffering scheme does not render directly to the Frame Buffer, instead, rendering is performed to a separate buffer, called the Draw Buffer. After all the changes are made, the Draw Buffer and Frame Buffer are swapped. After the swap, the Draw Buffer becomes the new Frame Buffer and the Frame Buffer becomes the new Draw Buffer. This swapping appears as an instant change in the contents of the screen.

The time it takes to render to the Draw Buffer does not change; however, since rendering occurs in the buffer that is not shown on the screen, the user will not see the pixels being changed. Instead, what a user sees is the instant switch of the screen contents when the two buffers are swapped. The net effect is that the user experience is better since the user does not see the slow rendering.

The next rendering will now occur on the new Draw Buffer while the user views the screen with pixels from the Frame Buffer. The cycle of rendering to the Draw Buffer and displaying the Frame Buffer continues whenever new pixels are being drawn.

The use of the double-buffering feature will double the requirement of the memory for the buffers. Since this feature is dependent on the availability of the memory resources for buffers, the feature can only be used when the Display Driver Layer of the library supports double-buffering features.

The feature is enabled by default in configurations where double-buffering is supported. To disable the feature, define GFX_CONFIG_DOUBLE_BUFFERING_DISABLE in the `graphics_config.h` header file.

The following steps show how the double-buffering feature is enabled and used when rendering primitive shapes. However, for synchronization, there are two ways to perform it, which is differentiated in step 3. There are two ways to use the feature in rendering primitive shapes. Method A - use this method when the driver layer that implements the management of the swapping of buffers, uses some synchronization timing to perform the swap (usually timed to the vertical synchronization period of the display):

1. Enable the feature (GFX_DoubleBufferEnable)
2. Render primitive shapes. When rendering the shapes there are two ways to perform rendering and synchronization.

- Render and register each new area
- Render an area
- Register the rectangular area that was modified (GFX_DoubleBufferAreaMark)
- Render all primitives and rectangular areas and register the whole screen. This method will take more time since more pixels are being synchronized.
- Render shapes and areas
- Register the whole screen to be updated (GFX_DoubleBufferSyncAllStatusSet)

3. Synchronize the buffers so they will contain the same pixel information. There are also two ways to perform this.

- Synchronize using a defined synchronization point. In this method, the driver is set up to perform the point of synchronization. For TFT displays, the vertical synchronization period is the best place to perform this since it will remove any possibility of showing portion of the screen with different data. This will happen when the refresh is in the middle of the screen and the buffers are swapped.
- GFX_DoubleBufferSynchronizeRequest
- Wait until the synchronization has finished using GFX_DoubleBufferSynchronizeStatusGet (i.e.,

```
while(GFX_DoubleBufferSynchronizeStatusGet() == GFX_FEATURE_ENABLED);
```

- Synchronize the buffers immediately
- GFX_DoubleBufferSynchronize

4. Repeat steps 2-3 for more rendering sequences.

If double buffering is disabled during run time, simply call GFX_DoubleBufferDisable. This function will perform one last synchronization to update the screen with recent rendering since the last synchronization.

Multi-instance Support

This section describes how to use two instances of the library.

Description

The application layer is considered a client of the Graphics Library. When GFX_Initialize is called it creates data structures for a single client referenced by the SYSTEM_MODULE_INDEX parameter (GFX client index). Each function of the library distinguishes each client through the use of the client index.

Multi-instance support is helpful when more than one driver and display is required. Multi-instance support is also helpful when two portions of a single display are to be rendered to separately.

Multi-instance support increases the code size of the application. If only one instance is to be used, configure GFX_INSTANCES_NUMBER to '0'. If multiple instances are to be used configure the macro accordingly.

```
#define GFX_INSTANCES_NUMBER 1
GFX_Initialize(GFX_INDEX_0, (SYS_MODULE_INIT*)&gfxInit1);
GFX_Initialize(GFX_INDEX_1, (SYS_MODULE_INIT*)&gfxInit2);
GFX_Tasks(GFX_INDEX_0);
GFX_Tasks(GFX_INDEX_1);
GFX_ColorSet(GFX_INDEX_0, CORAL);
GFX_ColorSet(GFX_INDEX_1, CYAN);
GFX_RectangleFillDraw(GFX_INDEX_0, 0,0,MaxXGet(),MaxYGet()/2);
GFX_RectangleFillDraw(GFX_INDEX_1,0,MaxYGet()/2,MaxXGet(), MaxYGet());
```

Surface Object

This section describes the Surface object.

Description

The Surface object provides a canvas onto which primitives and other objects can be drawn. The canvas can be the extent of the display or a smaller rectangular region. It can have a transparent, alpha-blended, or color background. It supports all draw and render states like all other GOL objects. An application can create multiple surfaces.

Unique to the Surface object is its callback function. The callback function allows the application to render application specific content onto the surface - which includes primitives. Because the Surface object is an Object, its drawing is rendered in sequence with all other objects.

Note: Refer to the Release Notes for any known limitations or restrictions. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

Library Dependencies

This section lists any other modules (by name) that are required to build the Graphics module.

Description

To build the Graphics Library you must include the modules listed in the following table.

Library Name	Location (<install-dir>/framework/)
Interrupt Peripheral Library	peripheral/int
Clock System Service Library	system/clk
DMA System Service Library	system/dma
Common System Service Library	system/common
Timer System Service Library	system/tmr
Interrupt System Service Library	system/int
Ports System Service Library	system/ports
SPI Driver Library	driver/spi

Timer Driver Library	driver/tmr

Configuring the Library

Macros

	Name	Description
	GFX_CONFIG_ALPHABLEND_DISABLE	Macro that disables the Alpha Blending feature.
	GFX_CONFIG_COLOR_DEPTH	Macro that sets the color depth of the application.
	GFX_CONFIG_DOUBLE_BUFFERING_DISABLE	Macro disables the support for double buffering in rendering of pixels to the frame buffer.
	GFX_CONFIG_FOCUS_DISABLE	Macro that disables the focus feature in objects.
	GFX_CONFIG_FONT_ANTIALIASED_DISABLE	Macro that disables the use of anti-aliased fonts.
	GFX_CONFIG_FONT_CHAR_SIZE	Macro that sets the size of the characters used in the fonts.
	GFX_CONFIG_FONT_EXTERNAL_DISABLE	Macro that disables sourcing of font resources from external sources.
	GFX_CONFIG_FONT_FLASH_DISABLE	Macro that disables sourcing of font resources from internal flash memory.
	GFX_CONFIG_FONT_RAM_DISABLE	Macro that disables sourcing of font resources from RAM sources.
	GFX_CONFIG_GRADIENT_DISABLE	Macro that disables the gradient fill feature.
	GFX_CONFIG_IMAGE_EXTERNAL_DISABLE	Macro that disables sourcing of image resources from external sources.
	GFX_CONFIG_IMAGE_PADDING_DISABLE	Macro disables the padding of bits on images converted by the Graphics Resource Converter (GRC).
	GFX_CONFIG_IMAGE_RAM_DISABLE	Macro that disables sourcing of image resources from RAM sources.
	GFX_CONFIG_IPU_DECODE_DISABLE	Macro that disables RLE compression of images.
	GFX_CONFIG_PALETTE_DISABLE	Macro that disables the palette feature.
	GFX_CONFIG_PALETTE_EXTERNAL_DISABLE	Macro that disables the palette feature that are sourced in external resources.
	GFX_CONFIG_RLE_DECODE_DISABLE	Macro that disables RLE compression of images.
	GFX_CONFIG_TRANSPARENT_COLOR_DISABLE	Macro that disables the use of anti-aliased fonts.
	GFX_CONFIG_USE_KEYBOARD_DISABLE	This macro disables the keyboard support in objects.
	GFX_CONFIG_USE_TOUCHSCREEN_DISABLE	This macro disables the resistive touchscreen support in objects.
	GFX_EXTERNAL_FONT_RASTER_BUFFER_SIZE	Macro sets the size of the external font raster buffer.
	GFX_free	Macro that defines the free function for versatility when using Operating Systems.
	GFX_malloc	Macro that defines the malloc function for versatility when using Operating Systems.
	GFX_CONFIG_IMAGE_FLASH_DISABLE	Macro that disables sourcing of image resources from internal flash memory.
	GFX_INSTANCES_NUMBER	Macro to specify the number of possible client modules.

Description

The library can be configured to be used with a specific combination of features. This section enumerates the available configuration options as well as examples of combination of configurations.

GFX_CONFIG_ALPHABLEND_DISABLE Macro

Macro that disables the Alpha Blending feature.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_ALPHABLEND_DISABLE
```

Description

Macro: GFX_CONFIG_ALPHABLEND_DISABLE

Alpha blending feature is available to fill functions.

To disable the alpha blending feature, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_COLOR_DEPTH Macro

Macro that sets the color depth of the application.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_COLOR_DEPTH
```

Description

Macro: GFX_CONFIG_COLOR_DEPTH

This macro sets the color depth used in the application. The library functions are also set to use the set color depth. The valid values for the color depth are the following: 1, 4, 8, 16 and 24 bpp. Usage of each is dependent on the support available on the display driver used. See the specific display driver documentation to verify support of the chosen color depth.

If this macro is not set, a build error will be generated.

Remarks

None.

GFX_CONFIG_DOUBLE_BUFFERING_DISABLE Macro

Macro disables the support for double buffering in rendering of pixels to the frame buffer.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_DOUBLE_BUFFERING_DISABLE
```

Description

Macro: GFX_CONFIG_DOUBLE_BUFFERING_DISABLE

In cases where display drivers has the resources for more than one display buffer, double buffering can be implemented in the driver. This allows application to hide the rendering effects by rendering on a hidden buffer and displaying another one. Once the rendering is done, the buffer are swapped. This gives an instantaneous change in the buffers which makes the change in the screen contents fast. The display driver must support the feature for this mode to work.

In drivers where this feature is not supported, this macro has no effect. In drivers that supports this feature, adding this macro will disable the feature.

Remarks

None.

GFX_CONFIG_FOCUS_DISABLE Macro

Macro that disables the focus feature in objects.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_FOCUS_DISABLE
```

Description

Macro: GFX_CONFIG_FOCUS_DISABLE

This compile option allows keyboard input focus.

- GFX_GOL_ObjectFocusSet()
- GFX_GOL_ObjectFocusGet()

- GFX_GOL_ObjectFocusNextGet()
- GFX_GOL_ObjectFocusPrevGet()

functions will be available. Focus is also changed by touch screen.

To disable the focus feature in objects, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_FONT_ANTIALIASED_DISABLE Macro

Macro that disables the use of anti-aliased fonts.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_FONT_ANTIALIASED_DISABLE
```

Description

Macro: GFX_CONFIG_FONT_ANTIALIASED_DISABLE

Anti-aliased fonts are supported in the library. The fonts must be generated using the Graphics Resource Converter (GRC). The GRC is a utility used in the Graphics Library to generate application resources.

To disable the anti-aliased fonts feature, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_FONT_CHAR_SIZE Macro

Macro that sets the size of the characters used in the fonts.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_FONT_CHAR_SIZE
```

Description

Macro: GFX_CONFIG_FONT_CHAR_SIZE

This configuration sets the text character size used in the library.

To enable support for unicode fonts, GFX_CONFIG_FONT_CHAR_SIZE must be defined as 16 (size is 16 bits). For standard ascii fonts, this can be defined as 8. This changes the GFX_XCHAR definition. See GFX_XCHAR for details.

Set in configuration	GFX_XCHAR	Description
#define GFX_CONFIG_FONT_CHAR_SIZE 16	#define GFX_XCHAR uint16_t	Use multi-byte characters (0-2^16 range).
#define GFX_CONFIG_FONT_CHAR_SIZE 8	#define GFX_XCHAR uint8_t	Use byte characters (0-255 range).

If this macro is not set, the default size used is 8-bits.

```
// example to set characters to use 16 bits
#define GFX_CONFIG_FONT_CHAR_SIZE 16

// example to set characters to use 8 bits
#define GFX_CONFIG_FONT_CHAR_SIZE 8
```

Remarks

None.

GFX_CONFIG_FONT_EXTERNAL_DISABLE Macro

Macro that disables sourcing of font resources from external sources.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_FONT_EXTERNAL_DISABLE
```

Description

Macro: GFX_CONFIG_FONT_EXTERNAL_DISABLE

Font data can be placed in multiple locations. They can be placed in FLASH memory, RAM and external memory. To reduce code size, any one of these locations, when not used, can be disabled by defining the macros at build time.

This macro disables the use of font resources from external memory.

To disable this feature, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_FONT_FLASH_DISABLE Macro

Macro that disables sourcing of font resources from internal flash memory.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_FONT_FLASH_DISABLE
```

Description

Macro: GFX_CONFIG_FONT_FLASH_DISABLE

Font data can be placed in multiple locations. They can be placed in FLASH memory, RAM and external memory. To reduce code size, any one of these locations, when not used, can be disabled by defining the macros at build time.

This macro disables the use of font resources from internal flash memory.

To disable this feature, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_FONT_RAM_DISABLE Macro

Macro that disables sourcing of font resources from RAM sources.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_FONT_RAM_DISABLE
```

Description

Macro: GFX_CONFIG_FONT_RAM_DISABLE

Font data can be placed in multiple locations. They can be placed in FLASH memory, RAM and external memory. To reduce code size, any one of these locations, when not used, can be disabled by defining the macros at build time.

This macro disables the use of font resources from RAM.

To disable this feature, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_GRADIENT_DISABLE Macro

Macro that disables the gradient fill feature.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_GRADIENT_DISABLE
```

Description

Macro: GFX_CONFIG_GRADIENT_DISABLE

Gradient fill is available to fill functions.

To disable the gradient feature, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_IMAGE_EXTERNAL_DISABLE Macro

Macro that disables sourcing of image resources from external sources.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_IMAGE_EXTERNAL_DISABLE
```

Description

Macro: GFX_CONFIG_IMAGE_EXTERNAL_DISABLE

Images can be placed in multiple locations. They can be placed in FLASH memory, RAM and external memory. To reduce code size, any one of these locations, when not used, can be disabled by defining the macros at build time.

This macro disables the use of image resources from external memory.

To disable this feature, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_IMAGE_PADDING_DISABLE Macro

Macro disables the padding of bits on images converted by the Graphics Resource Converter (GRC).

File

gfx_config_template.h

C

```
#define GFX_CONFIG_IMAGE_PADDING_DISABLE
```

Description

Macro: GFX_CONFIG_IMAGE_PADDING_DISABLE

When converting images for use in the Graphics Library, the Graphics Resource Converter has an option to set the images to be padded or not padded. When images are padded, each horizontal line will start on a byte boundary.

Unpadded images allows the least resource space for an image. Unpadded images also allows support for display controllers with windowing and auto-increment features.

The images are created with padding by default. When the images are set to be not padded, application must define the macro in the configuration to correctly process the images.

Padded and unpadded images cannot be combined in one application.

Remarks

None.

GFX_CONFIG_IMAGE_RAM_DISABLE Macro

Macro that disables sourcing of image resources from RAM sources.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_IMAGE_RAM_DISABLE
```

Description

Macro: GFX_CONFIG_IMAGE_RAM_DISABLE

Images can be placed in multiple locations. They can be placed in FLASH memory, RAM and external memory. To reduce code size, any one of these locations, when not used, can be disabled by defining the macros at build time.

This macro disables the use of image resources from RAM.

To disable this feature, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_IPU_DECODE_DISABLE Macro

Macro that disables RLE compression of images.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_IPU_DECODE_DISABLE
```

Description

Macro: GFX_CONFIG_IPU_DECODE_DISABLE

Images can also be compressed using the DEFLATE algorithm. Using the drivers that supports DEFLATE (IPU of PIC24FJ256DA210 Family of devices), GFX_ImageDraw() will be able to render these images. This feature is enabled by default.

To disable this feature, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_PALETTE_DISABLE Macro

Macro that disables the palette feature.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_PALETTE_DISABLE
```

Description

Macro: GFX_CONFIG_PALETTE_DISABLE

The use of a palette is an option to run the application with 256 16, or 2 colors only. When using this feature, the color depth set for the application should be 8, 4 or 1 bpp (see GFX_CONFIG_COLOR_DEPTH for color depth settings).

To disable the palette feature, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_PALETTE_EXTERNAL_DISABLE Macro

Macro that disables the palette feature that are sourced in external resources.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_PALETTE_EXTERNAL_DISABLE
```

Description

Macro: GFX_CONFIG_PALETTE_EXTERNAL_DISABLE

Similar to fonts and images, Palettes are considered resources. The use of palette also allow the application to locate the palette resources in external sources. This macro disables the code that implements externally sourced palettes.

To disable the palette to be sourced from external resources, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_RLE_DECODE_DISABLE Macro

Macro that disables RLE compression of images.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_RLE_DECODE_DISABLE
```

Description

Macro: GFX_CONFIG_RLE_DECODE_DISABLE

Images that have 8bpp or 4 bpp color depth has the option to be RLE compressed. GFX_ImageDraw() will be able to render these images. This feature is enabled by default.

```
    // example to use images that are NOT RLE encoded and sourced from
    // internal flash only
    #define GFX_CONFIG_RLE_DECODE_DISABLE
    #define GFX_CONFIG_IMAGE_EXTERNAL_DISABLE
    #define GFX_CONFIG_IMAGE_RAM_DISABLE
```

To disable this feature, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_TRANSPARENT_COLOR_DISABLE Macro

Macro that disables the use of anti-aliased fonts.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_TRANSPARENT_COLOR_DISABLE
```

Description

Macro: GFX_CONFIG_TRANSPARENT_COLOR_DISABLE

Transparent color is a feature in GFX_ImageDraw() where pixels that equals the color set in the transparent color variable will not be rendered. This is useful in rendering rounded icons or images to the screen with complex background.

To disable the transparent color in images, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_USE_KEYBOARD_DISABLE Macro

This macro disables the keyboard support in objects.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_USE_KEYBOARD_DISABLE
```

Description

Macro: GFX_CONFIG_USE_KEYBOARD_DISABLE

Depending on the input devices used, messages that objects will process are encoded in the input device drivers. The keyboard is one of the input devices that is supported in selected objects.

This support is enabled by default. To disable this feature, add this macro in the configuration.

Remarks

None.

GFX_CONFIG_USE_TOUCHSCREEN_DISABLE Macro

This macro disables the resistive touchscreen support in objects.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_USE_TOUCHSCREEN_DISABLE
```

Description

Macro: GFX_CONFIG_USE_TOUCHSCREEN_DISABLE

Depending on the input devices used, messages that objects will process are encoded in the input device drivers. The resistive touchscreen is one of the input devices that is supported in selected objects.

This support is enabled by default. To disable this feature, add this macro in the configuration.

Remarks

None.

GFX_EXTERNAL_FONT_RASTER_BUFFER_SIZE Macro

Macro sets the size of the external font raster buffer.

File

gfx_config_template.h

C

```
#define GFX_EXTERNAL_FONT_RASTER_BUFFER_SIZE
```

Description

Macro: GFX_EXTERNAL_FONT_RASTER_BUFFER_SIZE

This macro sets the size of the external font raster buffer. This buffer is used to store the character glyph when the glyph is being read from external source. The value that is needed will be calculated by the Graphics Resource Converter (GRC). The value will be shown in the external resource reference c file that the GRC generates.

```
// example to set the buffer size.
// value is taken from the output of the GRC
#define GFX_EXTERNAL_FONT_RASTER_BUFFER_SIZE 100
```

A warning will be issued when building the application when:

- when the configuration do not define this macro
- when the defined value is less than the required size

This macro will have no effect when fonts that are sourced externally is not used.

Remarks

None.

GFX_free Macro

Macro that defines the free function for versatility when using Operating Systems.

File

gfx_config_template.h

C

```
#define GFX_free(pObj) free(pObj)
```

Description

Macro: GFX_free()

This macro definition allows the application to replace the free function to an equivalent function implemented in the operating system used.

When using the library with object layer and without any replacement functions for the malloc function, this macro must be defined as:

```
    #define GFX_free(pObj)      free(pObj)
```

Remarks

None.

GFX_malloc Macro

Macro that defines the malloc function for versatility when using Operating Systems.

File

gfx_config_template.h

C

```
#define GFX_malloc(size) malloc(size)
```

Description

Macro: GFX_malloc()

This macro definition allows the application to replace the malloc function to an equivalent function implemented in the operating system used.

When using the library with object layer and without any replacement functions for the malloc function, this macro must be defined as:

```
    #define GFX_malloc(size)    malloc(size)
```

Remarks

None.

GFX_CONFIG_IMAGE_FLASH_DISABLE Macro

Macro that disables sourcing of image resources from internal flash memory.

File

gfx_config_template.h

C

```
#define GFX_CONFIG_IMAGE_FLASH_DISABLE
```

Description

Macro: GFX_CONFIG_IMAGE_FLASH_DISABLE

Images can be placed in multiple locations. They can be placed in FLASH memory, RAM and external memory. To reduce code size, any one of these locations, when not used, can be disabled by defining the macros at build time.

This macro disables the use of image resources from internal flash memory.

To disable this feature, add this macro in the configuration.

Remarks

None.

GFX_INSTANCES_NUMBER Macro

Macro to specify the number of possible client modules.

File

gfx_config_template.h

C

```
#define GFX_INSTANCES_NUMBER
```

Description

Macro: GFX_INSTANCES_NUMBER

This macro sets the maximum number of possible instances that can be created.

To set the maximum number of clients, add this macro in the configuration and the appropriate number.

Remarks

The number will affect memory usage.

None.

Configuration Examples

This section shows example of configuration combination when using the library.

Example 1

Configuration example 1.

Description

- Non-blocking is enabled on accelerated functions
- Alpha blending
- Gradient fills
- Palette is not used
- Focus on objects that supports it is enabled
- Double buffering is disabled
- Characters size is 16 bits to support UNICODE characters
- Fonts are sourced from two places:
- Internal Flash
- External memory
- Fonts from RAM are disabled
- Fonts that are anti-aliased are enabled
- Fonts sourced from external memory are allocated 51 bytes of buffer
- Images are sourced from two places:
- Internal Flash
- External memory
- Fonts from RAM is disabled
- Images that are RLE encoded are enabled
- Images that are DEFLATE (for IPU module) encoded are enabled
- Images can be rendered with transparent color feature
- Touch screen support is enabled
- Keyboard support is enabled
- GFX_malloc and GFX_free functions are used to manage memory for object layer

```
// Note: Configuration to disable the features are commented out
// to show the specific macros that disables them.

//#define GFX_CONFIG_NONBLOCKING_DISABLE
//#define GFX_CONFIG_ALPHABLEND_DISABLE
//#define GFX_CONFIG_GRADIENT_DISABLE
//#define GFX_CONFIG_FOCUS_DISABLE
#define GFX_CONFIG_PALETTE_DISABLE
#define GFX_CONFIG_PALETTE_EXTERNAL_DISABLE
#define GFX_CONFIG_DOUBLE_BUFFERING_DISABLE

#define GFX_CONFIG_FONT_CHAR_SIZE 16
#define GFX_CONFIG_COLOR_DEPTH 16

//#define GFX_CONFIG_FONT_FLASH_DISABLE
//#define GFX_CONFIG_FONT_EXTERNAL_DISABLE
```

```
#define GFX_CONFIG_FONT_RAM_DISABLE
//#define GFX_CONFIG_FONT_ANTIALIASED_DISABLE

//#define GFX_CONFIG_IMAGE_FLASH_DISABLE
//#define GFX_CONFIG_IMAGE_EXTERNAL_DISABLE
#define GFX_CONFIG_IMAGE_RAM_DISABLE

//#define GFX_CONFIG_RLE_DECODE_DISABLE
//#define GFX_CONFIG_IPU_DECODE_DISABLE

//#define GFX_CONFIG_TRANSPARENT_COLOR_DISABLE
#define GFX_EXTERNAL_FONT_RASTER_BUFFER_SIZE 51

//#define GFX_CONFIG_USE_TOUCHSCREEN_DISABLE
//#define GFX_CONFIG_USE_KEYBOARD_DISABLE

#define GFX_malloc(size) malloc(size)
#define GFX_free(pObj) free(pObj)
```

Example 2

Configuration example 2.

Description

- All primitive functions are blocking
- The following are disabled:
- Alpha blending
- Gradient fills
- Palette
- Focus on objects
- Double buffering
- Anti-aliased fonts
- RLE and DEFLATE encoded images
- Transparent color
- Touchscreen
- Keyboard
- Characters size is 8 bits
- Images are sourced only from internal Flash
- Fonts are sourced only from internal Flash and since fonts are sourced from internal Flash, the external font raster buffer is not needed
- Object layer is not used so GFX_malloc and GFX_free are never referenced

```
// Note: Configuration to disable the features are commented out
//       to show the specific macros that disables them.

#define GFX_CONFIG_NONBLOCKING_DISABLE
#define GFX_CONFIG_ALPHABLEND_DISABLE
#define GFX_CONFIG_GRADIENT_DISABLE
#define GFX_CONFIG_FOCUS_DISABLE
#define GFX_CONFIG_PALETTE_DISABLE
#define GFX_CONFIG_PALETTE_EXTERNAL_DISABLE
#define GFX_CONFIG_DOUBLE_BUFFERING_DISABLE

#define GFX_CONFIG_FONT_CHAR_SIZE 8
#define GFX_CONFIG_COLOR_DEPTH 16

//#define GFX_CONFIG_FONT_FLASH_DISABLE
#define GFX_CONFIG_FONT_EXTERNAL_DISABLE
#define GFX_CONFIG_FONT_RAM_DISABLE
#define GFX_CONFIG_FONT_ANTIALIASED_DISABLE

//#define GFX_CONFIG_IMAGE_FLASH_DISABLE
#define GFX_CONFIG_IMAGE_EXTERNAL_DISABLE
#define GFX_CONFIG_IMAGE_RAM_DISABLE

#define GFX_CONFIG_RLE_DECODE_DISABLE
#define GFX_CONFIG_IPU_DECODE_DISABLE
```

```
#define GFX_CONFIG_TRANSPARENT_COLOR_DISABLE
//#define GFX_EXTERNAL_FONT_RASTER_BUFFER_SIZE 51

#define GFX_CONFIG_USE_TOUCHSCREEN_DISABLE
#define GFX_CONFIG_USE_KEYBOARD_DISABLE

//#define GFX_malloc(size)    malloc(size)
//#define GFX_free(pObj)      free(pObj)
```

Building the Library

This section lists the files that are available in the Graphics Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/gfx`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
gfx.h	Graphics Library initialization header file.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

The following table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build the LCC Library.

Source File Name	Description
/src/drv_gfx_lcc_int.c	LCC Driver Library small buffer internal memory framebuffer driver.
/src/drv_gfx_lcc_int_pe.c	LCC Driver Library large buffer internal memory framebuffer driver.
/src/drv_gfx_lcc_ext.c	LCC Driver Library external memory framebuffer driver for wqvga.
/src/drv_gfx_lcc_ext_wvga.c	LCC Driver Library external memory framebuffer driver for wvga.

the following table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build the Graphics Library.

Source File Name	Description
gfx.h	Graphics Library initialization header file.
/src/gfx.c	Graphics Library initialization implementation.
gfx_primitive.h	Primitive Layer run-time header file.
/src/gfx_primitive.c	Primitive Layer run-time implementation.
gfx_gol.h	Object Layer run-time header.
/src/gfx_gol.c	Object Layer run-time implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included for the Graphics Library if required for the desired implementation.

Source File Name	Description
gfx_colors.h	Default color definitions
gfx_colors_w3.h	W3 color definitions
gfx_colors_x11.h	X11 color definitions
gfx_gol_button.h	Button widget header
/src/gfx_gol_button.c	Button widget implementation

gfx_gol_check_box.h	Check Box widget header
/src/gfx_gol_check_box.c	Check Box widget implementation
gfx_gol_custom_control.h	Custom control widget header
/src/gfx_gol_custom_control.c	Custom control widget implementation
gfx_gol_digital_meter.h	Digital meter widget header
/src/gfx_gol_digital_meter.c	Digital meter widget header
gfx_gol_edit_box.h	Edit Box widget header
/src/gfx_gol_edit_box.c	Edit Box widget implementation
gfx_gol_group_box.h	Group Box widget header
/src/gfx_gol_group_box.c	Group Box widget implementation
gfx_gol_list_box.h	List Box widget implementation
/src/gfx_gol_list_box.c	List Box widget implementation
gfx_gol_meter.h	Analog meter widget header
/src/gfx_gol_meter_c	Analog meter widget implementation
gfx_gol_picture.h	Picture widget header
/src/gfx_gol_picture.c	Picture widget implementation
gfx_gol_progress_bar.h	Progress Bar widget header
/src/gfx_gol_progress_bar.c	Progress Bar widget implementation
gfx_gol_radio_button.h	Radio Button widget header
/src/gfx_gol_radio_button.c	Radio Button widget implementation
gfx_gol_scan_codes.h	Keyboard codes header
gfx_gol_scheme.h	Object style scheme header
/src/gfx_gol_scheme_default.c	Object style scheme implementation
gfx_gol_scroll_bar.h	Scroll bar widget header
/src/gfx_gol_scroll_bar.c	Scroll bar widget implementation
gfx_gol_static_text.h	Static Text widget header
/src/gfx_gol_static_text.c	Static Text widget implementation
/src/gfx_gol_surface.h	Surface object widget header
/src/gfx_gol_surface.c	Surface object widget implementation
/src/gfx_gol_text_entry.c	Text Entry widget header
/src/gfx_gol_text_entry.c	Text Entry widget implementation
gfx_gol_window.h	Window widget header
/src/gfx_gol_window.c	Window widget implementation
gfx_image_decoder.h	Image decoder header
/src/gfx_image_decoder.c	Image decoder implementation
gfx_types_font.h	Font types
gfx_types_image.h	Image types
gfx_types_macros.h	GFX system wide types and macros
gfx_types_palette.h	Palette types
gfx_types_resource.h	Resource types for images fonts palette and binary types

Module Dependencies

The File System Service Library depends on the following modules:

- Graphics Driver Library

Library Interface

This section describes the Application Programming Interface (API) functions of the Graphics Library.

Refer to each section for a detailed description.

Graphics Primitive Layer

This section describes the API and usage of the Graphics Library Primitive Layer.

Graphics Primitive Layer API

Application Programming Interface functions for the Primitive Layer of the Graphics Library.

Initialization Functions

Functions

	Name	Description
	GFX_PRIMITIVE_Initialize	Initialize the GFX PRIMITIVE HAL Library.
	GFX_PRIMITIVE_Deinitialize	Deinitializes the specified instance of the GFX Module.
	GFX_PRIMITIVE_Close	Closes an instance of the primitive layer specified by handle.
	GFX_PRIMITIVE_Open	Opens a client instance of the primitive layer specified by index.
	GFX_ScreenClear	Clears the screen to the currently set color (GFX_ColorSet()).

Description

Graphics Library Primitive Layer initialization API.

GFX_PRIMITIVE_Initialize Function

Initialize the GFX PRIMITIVE HAL Library.

File

gfx_primitive.h

C

```
SYS_MODULE_OBJ GFX_PRIMITIVE_Initialize(const SYS_MODULE_INDEX moduleIndex, const SYS_MODULE_INIT * const moduleInit);
```

Returns

If successful, returns a valid handle to a primitive object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the GFX Library Primitive HAL.

Remarks

This routine must be called before other GFX PRIMITIVE HAL functions.

This function initialize the Graphics Library primitive layer. The following default settings are set when this function is called.

1. font - Set to NULL. GFX_FontSet() must be called prior to any text rendering.
2. line type - Set to GFX_LINE_TYPE_THIN_SOLID (see GFX_LINE_STYLE).
3. fill type - Set to GFX_FILL_TYPE_COLOR (see GFX_FILL_STYLE).
4. text anti-alias type - Set to GFX_FONT_ANTIALIAS_OPAQUE (see GFX_FONT_ANTIALIAS_TYPE). This only affects fonts with anti-aliasing enabled.
5. Set transparent color feature in image draw functions

to be disabled.

6. Set alpha blending value to 100 (or no alpha blending) if

alpha-blending feature is enabled.

7. Set background information to no background.

This function does not clear the screen and does not assign any color to the currently set color. Application should set the color and clear the screen.

Preconditions

None.

Example

```
GFX_PRIMITIVE_INIT gfxInit;
```

```
SYS_MODULE_OBJ primObj;

// GFX Module initialization
primObj = GFX_PRIMITIVE_Initialize(GFX_PRIMITIVE_INDEX_0, &gfxInit);

if (SYS_MODULE_OBJ_INVALID == primObj)
{
    // Handle error
}
```

Parameters

Parameters	Description
moduleIndex	client instance request.
moduleInit	initialization data for the instance.

Function

void GFX_PRIMITIVE_Initialize (const SYS_MODULE_INDEX moduleIndex,
const SYS_MODULE_INIT * const moduleInit);

GFX_PRIMITIVE_Deinitialize Function

Deinitializes the specified instance of the GFX Module.

File

gfx_primitive.h

C

```
void GFX_PRIMITIVE_Deinitialize(SYS_MODULE_OBJ moduleObject);
```

Returns

None.

Description

Deinitializes the specified instance of the GFX Module. All internal data structures will be reset.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This routine will not block waiting for hardware.

Preconditions

Function GFX_GOL_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from GFX_PRIMITIVE_Initialize
SYS_STATUS          status;


GFX_PRIMITIVE_Deinitialize(object);

status = GFX_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	GFX layer object handle, returned from the GFX_PRIMITIVE_Initialize routine

Function

void GFX_PRIMITIVE_Deinitialize(SYS_MODULE_OBJ object)

GFX_PRIMITIVE_Close Function

Closes an instance of the primitive layer specified by handle.

File

gfx_primitive.h

C

```
void GFX_PRIMITIVE_Close(GFX_PRIMITIVE_HANDLE handle);
```

Description

This function closes the client instance.

Parameters

Parameters	Description
handle	primitive handle

Function

void GFX_PRIMITIVE_Close(GFX_PRIMITIVE_HANDLE handle)

GFX_PRIMITIVE_Open Function

Opens a client instance of the primitive layer specified by index.

File

gfx_primitive.h

C

```
GFX_PRIMITIVE_HANDLE GFX_PRIMITIVE_Open(const SYS_MODULE_INDEX index);
```

Returns

GFX_PRIMITIVE HANDLE - handle to the primitive layer.

Description

This function opens a client instance. THe user must have called GFX_PRIMITIVE_Initialize prior to calling this functions. It opens the graphics driver.

Parameters

Parameters	Description
index	instance of the driver

Function

GFX_PRIMITIVE_HANDLE GFX_PRIMITIVE_Open(const SYS_MODULE_INDEX index)

GFX_ScreenClear Function

Clears the screen to the currently set color (GFX_ColorSet()).

File

gfx_primitive.h

C

```
GFX_STATUS GFX_ScreenClear(SYS_MODULE_INDEX gfxIndex);
```

Returns

The status of the screen clearing. GFX_STATUS_SUCCESS - screen was cleared. GFX_STATUS_FAILURE - screen is not yet cleared

Description

This function clears the screen with the current color and sets the line cursor position to (0, 0).

If color is not set, before this function is called, the output is undefined.

If the function returns GFX_STATUS_FAILURE, clearing is not yet finished. Application must call the function again to continue the clearing.

Preconditions

Color must be set by GFX_ColorSet().

None.

Example

```
GFX_ColorSet(GFX_INDEX_0, BLACK);
GFX_ScreenClear(GFX_INDEX_0);
```

Function

GFX_STATUS GFX_ScreenClear(SYS_MODULE_INDEX gfxIndex)

Display Functions

Functions

	Name	Description
	GFX_MaxXGet	This function returns the maximum horizontal pixel coordinate.
	GFX_MaxYGet	This function returns the maximum vertical pixel coordinate.

Description

GFX_MaxXGet Function

This function returns the maximum horizontal pixel coordinate.

File

gfx_primitive.h

C

```
inline uint16_t GFX_MaxXGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

Maximum horizontal pixel coordinate.

Description

This function returns the maximum horizontal pixel coordinate of the chosen display. The value returned is equal to the horizontal dimension minus 1. The horizontal dimension is also dependent on the rotation of the screen. Rotation can be set in the hardware configuration.

Preconditions

None.

Example

None.

****

#if (DISP_ORIENTATION == 90) #define GFX_MaxXGet() (DISP_VER_RESOLUTION - 1) #elif (DISP_ORIENTATION == 270) #define GFX_MaxXGet() (DISP_VER_RESOLUTION - 1) #elif (DISP_ORIENTATION == 10) #define GFX_MaxXGet() (DISP_HOR_RESOLUTION - 1) #elif (DISP_ORIENTATION == 180) #define GFX_MaxXGet() (DISP_HOR_RESOLUTION - 1) #endif

Function

uint16_t GFX_MaxXGet(void)

GFX_MaxYGet Function

This function returns the maximum vertical pixel coordinate.

File

gfx_primitive.h

C

```
inline uint16_t GFX_MaxYGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

Maximum vertical pixel coordinate.

Description

This function returns the maximum vertical pixel coordinate of the chosen display. The value returned is equal to the vertical dimension minus 1. The vertical dimension is also dependent on the rotation of the screen. Rotation can be set in the hardware configuration.

Preconditions

None.

Example

None.

****

#if (DISP_ORIENTATION == 90) #define GFX_MaxYGet() (DISP_HOR_RESOLUTION - 1) #elif (DISP_ORIENTATION == 270) #define GFX_MaxYGet() (DISP_HOR_RESOLUTION - 1) #elif (DISP_ORIENTATION == 10) #define GFX_MaxYGet() (DISP_VER_RESOLUTION - 1) #elif (DISP_ORIENTATION == 180) #define GFX_MaxYGet() (DISP_VER_RESOLUTION - 1) #endif

Function

uint16_t GFX_MaxYGet(void)

Line Rendering Functions

Functions

	Name	Description
	GFX_LineDraw	This function renders a line from x1,y1 to x2,y2 using the currently set line style (see GFX_LineStyleSet()).
	GFX_LinePositionRelativeSet	This function sets the line cursor to a new position relative to the current position.
	GFX_LinePositionSet	This function sets the line cursor to a new position.
	GFX_LinePositionXGet	This function returns the current x position of the line cursor.
	GFX_LinePositionYGet	This function returns the current y position of the line cursor.
	GFX_LineToDraw	This function renders a line from current line cursor position (x,y) to (x2,y2) using the currently set line style (see GFX_LineStyleSet()).
	GFX_LineToRelativeDraw	This function renders a line from current line cursor position (x,y) to (x+dX,y+dY) using the currently set line style (see GFX_LineStyleSet()).

Description

Graphics Library Primitive Layer line rendering API.

GFX_LineDraw Function

This function renders a line from x1,y1 to x2,y2 using the currently set line style (see GFX_LineStyleSet()).

File

gfx_primitive.h

C

```
GFX_STATUS GFX_LineDraw(SYS_MODULE_INDEX gfxIndex, uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2);
```

Returns

Status of the line rendering. GFX_STATUS_SUCCESS - line rendering done. GFX_STATUS_FAILURE - line rendering is not done.

Description

This function renders a line from x1,y1 to x2,y2 using the currently set line style set by GFX_LineStyleSet(). The color used is the color set by the last call to GFX_ColorSet().

If x1,y1 and/or x2,y2 is not on the frame buffer, then the behavior is undefined. If color is not set, before this function is called, the output is undefined.

Preconditions

Color must be set by GFX_ColorSet(). Line style must be set by GFX_LineStyleSet().

Example

```
GFX_ColorSet(GFX_INDEX_0, BRIGHTRED);
```

```
GFX_LineStyleSet(GFX_INDEX_0, GFX_LINE_STYLE_THIN_DOTTED);
GFX_LineDraw(GFX_INDEX_0, 10, 10,100,  10);
```

Parameters

Parameters	Description
x1	x coordinate of the line start point.
y1	y coordinate of the line start point.
x2	x coordinate of the line end point.
y2	y coordinate of the line end point.

Function

GFX_STATUS GFX_LineDraw(
SYS_MODULE_INDEX gfxIndex,
uint16_t x1,
uint16_t y1,
uint16_t x2,
uint16_t y2)

GFX_LinePositionRelativeSet Function

This function sets the line cursor to a new position relative to the current position.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_LinePositionRelativeSet(SYS_MODULE_INDEX gfxIndex, int16_t x, int16_t y);
```

Returns

Status of the relative line position set. GFX_STATUS_SUCCESS - relative line position set done. GFX_STATUS_FAILURE - relative line position set not done.

Description

This function sets the line cursor to a new (x,y) position relative to the current cursor position. The new position is calculated by (x+dX, y + dY).

Line cursor is used as a starting point of the line rendered by the GFX_LineToDraw() and GFX_LineToRelativeDraw() functions. Note that the parameters dX and dY are signed integers. This allows the new line cursor position to be placed to any direction from the current line cursor position.

If (x+dX) and/or (y+dY) results in a position that is not on the frame buffer, then the behavior of GFX_LineToDraw() and GFX_LineToRelativeDraw() functions are undefined. If color is not set, before this function is called, the output is undefined.

Preconditions

None.

Example

See GFX_LineToDraw().

Parameters

Parameters	Description
dX	the offset for the x position that will define the new x-coordinate position of the line cursor.
dY	the offset for the y position that will define the new y-coordinate position of the line cursor.

Function

GFX_STATUS GFX_LinePositionRelativeSet(
SYS_MODULE_INDEX gfxIndex,
int16_t dX,
int16_t dY)

GFX_LinePositionSet Function

This function sets the line cursor to a new position.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_LinePositionSet(SYS_MODULE_INDEX gfxIndex, uint16_t x, uint16_t y);
```

Returns

Status of the line position set. GFX_STATUS_SUCCESS - line position set done. GFX_STATUS_FAILURE - line position set not done.

Description

This function sets the line cursor to a new (x,y) position. Line cursor is used as a starting point of the line rendered by the GFX_LineToDraw() and GFX_LineToRelativeDraw() functions.

If x and/or y does not lie on the frame buffer, then the behavior of GFX_LineToDraw() and GFX_LineToRelativeDraw() functions are undefined.

Preconditions

None.

Example

See GFX_LinePositionXGet().

Parameters

Parameters	Description
x	new x coordinate position of the line cursor.
y	new y coordinate position of the line cursor.

Function

GFX_STATUS GFX_LinePositionSet(
SYS_MODULE_INDEX gfxIndex,
uint16_t x,
uint16_t y)

GFX_LinePositionXGet Function

This function returns the current x position of the line cursor.

File

gfx_primitive.h

C

```
int16_t GFX_LinePositionXGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The current line cursor x position.

Description

This function returns the current x position of the line cursor. Line cursor is used as a starting point of the line rendered by the GFX_LineToDraw() and GFX_LineToRelativeDraw() functions.

Preconditions

None.

Example

```
// implementation of the GFX_LineToRelativeDraw()
GFX_STATUS GFX_LineToRelativeDraw(
                                GFX_INDEX_0,
                                int16_t dX,
                                int16_t dY)
{
    return (GFX_LineDraw(
                            GFX_INDEX_0,
                            GFX_LinePositionXGet(gfxIndex),
                            GFX_LinePositionYGet(gfxIndex),
                            GFX_LinePositionXGet(gfxIndex) + dX,
```

```
                                    GFX_LinePositionYGet(gfxIndex) + dY));
    }
```

Function

int16_t GFX_LinePositionXGet(SYS_MODULE_INDEX gfxIndex)

GFX_LinePositionYGet Function

This function returns the current y position of the line cursor.

File

gfx_primitive.h

C

```
int16_t GFX_LinePositionYGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The current line cursor y position.

Description

This function returns the current y position of the line cursor. Line cursor is used as a starting point of the line rendered by the GFX_LineToDraw() and GFX_LineToRelativeDraw() functions.

Preconditions

None.

Example

See GFX_LinePositionXGet().

Function

int16_t GFX_LinePositionYGet(SYS_MODULE_INDEX gfxIndex)

GFX_LineToDraw Function

This function renders a line from current line cursor position (x,y) to (x2,y2) using the currently set line style (see GFX_LineStyleSet()).

File

gfx_primitive.h

C

```
GFX_STATUS GFX_LineToDraw(SYS_MODULE_INDEX gfxIndex, int16_t x2, int16_t y2);
```

Returns

Status of the line rendering. GFX_STATUS_SUCCESS - line rendering done. GFX_STATUS_FAILURE - line rendering is not done.

Description

This function renders a line from current line cursor position (x,y) to (x2,y2) using the currently set line style set by GFX_LineStyleSet(). The color used is the color set by the last call to GFX_ColorSet().

If x2 and/or y2 does not lie on the frame buffer, then the behavior is undefined. If color is not set, before this function is called, the output is undefined.

Preconditions

Color must be set by GFX_ColorSet(). Line style must be set by GFX_LineStyleSet().

Example

```
    GFX_ColorSet(GFX_INDEX_0, BRIGHTRED);
    GFX_LineStyleSet(GFX_INDEX_0, GFX_LINE_STYLE_THICK_DOTTED);

    GFX_LinePositionSet(GFX_INDEX_0, 100, 100);
    GFX_LineToDraw(GFX_INDEX_0, 100,  10);
    GFX_LinePositionRelativeSet(GFX_INDEX_0, -90,  0);
    GFX_LineToDraw(GFX_INDEX_0, 100,  10);
    GFX_LinePositionRelativeSet(GFX_INDEX_0,   0, 90);
    GFX_LineToDraw(GFX_INDEX_0,  10, 100);
    GFX_LinePositionRelativeSet(GFX_INDEX_0,  90,-90);
    GFX_LineToDraw(GFX_INDEX_0,  10, 100);
```

```
GFX_LinePositionRelativeSet(GFX_INDEX_0,   0,-90);
GFX_LineToDraw(GFX_INDEX_0, 100, 100);
GFX_LinePositionRelativeSet(GFX_INDEX_0,  -90, 0);
GFX_LineToDraw(GFX_INDEX_0, 10,  10);
```

Parameters

Parameters	Description
x2	x coordinate of the line end point.
y2	y coordinate of the line end point.

Function

GFX_STATUS GFX_LineToDraw(
SYS_MODULE_INDEX gfxIndex,
int16_t x2,
int16_t y2)

GFX_LineToRelativeDraw Function

This function renders a line from current line cursor position (x,y) to (x+dX,y+dY) using the currently set line style (see GFX_LineStyleSet()).

File

gfx_primitive.h

C

```
GFX_STATUS GFX_LineToRelativeDraw(SYS_MODULE_INDEX gfxIndex, int16_t dX, int16_t dY);
```

Returns

Status of the line rendering. GFX_STATUS_SUCCESS - line rendering done. GFX_STATUS_FAILURE - line rendering is not done.

Description

This function renders a line from current line cursor position (x,y) to (x+dX,y+dY) using the currently set line style set by GFX_LineStyleSet(). The color used is the color set by the last call to GFX_ColorSet(). Note that the parameters dX and dY are signed integers. This allows the line to be drawn from the line cursor to any direction.

If (x+dX) and/or (y+dY) results in a position that is not on the frame buffer, then the behavior is undefined. If color is not set, before this function is called, the output is undefined.

Preconditions

Color must be set by GFX_ColorSet(). Line style must be set by GFX_LineStyleSet().

Example

```
GFX_ColorSet(GFX_INDEX_0, BRIGHTRED);
GFX_LineStyleSet(GFX_INDEX_0, GFX_LINE_STYLE_THIN_DOTTED);

GFX_LinePositionSet(GFX_INDEX_0, 100, 100);
GFX_LineToRelativeDraw(GFX_INDEX_0,   0, -90);
GFX_LineToRelativeDraw(GFX_INDEX_0, -90,   0);
GFX_LineToRelativeDraw(GFX_INDEX_0,  90,  90);
GFX_LineToRelativeDraw(GFX_INDEX_0, -90,   0);
GFX_LineToRelativeDraw(GFX_INDEX_0,  90, -90);
GFX_LinePositionSet(GFX_INDEX_0, 10, 10);
GFX_LineToRelativeDraw(GFX_INDEX_0,  0,  90);
```

Parameters

Parameters	Description
dX	the offset for the x starting position that will define the x-coordinate of the end of the line.
dY	the offset for the y starting position that will define the y-coordinate of the end of the line.

Function

GFX_STATUS GFX_LineToRelativeDraw(
SYS_MODULE_INDEX gfxIndex,
int16_t dX,
int16_t dY)

Polygon Rendering Functions

Functions

	Name	Description
	GFX_CircleDraw	This function renders a circular shape using the currently set line style and color.
	GFX_PolygonDraw	This function renders a polygon using the currently set line style and color.
	GFX_RectangleDraw	This function renders a rectangular shape using the currently set line style and color.
	GFX_RectangleRoundDraw	This function renders a rounded corner rectangular shape using the currently set line style and color.

Description

Graphics Library Primitive Layer polygon rendering API.

GFX_CircleDraw Function

This function renders a circular shape using the currently set line style and color.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_CircleDraw(SYS_MODULE_INDEX gfxIndex, uint16_t x, uint16_t y, uint16_t radius);
```

Returns

Status of the circle rendering. GFX_STATUS_SUCCESS - circle rendering done. GFX_STATUS_FAILURE - circle rendering is not done.

Description

This function renders a circular shape using the center (x,y) and radius. The shape is rendered using the currently set line style by GFX_LineStyleSet(). The color used is the color set by the last call to GFX_ColorSet().

When x,y falls outside the buffer, the behavior is undefined. When color is not set before this function is called, the behavior is undefined. When any of the following x+radius, x-radius, y+radius and y-radius falls outside the buffer, the behavior is undefined.

Preconditions

Color must be set by GFX_ColorSet(). Line style must be set by GFX_LineStyleSet().

Example

```
// draw a circle using bright red solid line
GFX_LineStyleSet(GFX_INDEX_0, GFX_LINE_STYLE_THIN_SOLID);
GFX_ColorSet(GFX_INDEX_0, BRIGHTRED);
GFX_CircleDraw(GFX_INDEX_0, 50, 50, 40);
```

Parameters

Parameters	Description
x	defines the x-coordinate position of the center of the circle.
y	defines the y-coordinate position of the center of the circle.
radius	defines the radius of the circle.

Function

GFX_STATUS GFX_CircleDraw(
SYS_MODULE_INDEX gfxIndex,
uint16_t x,
uint16_t y,
uint16_t radius)

GFX_PolygonDraw Function

This function renders a polygon using the currently set line style and color.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_PolygonDraw(SYS_MODULE_INDEX gfxIndex, uint16_t numPoints, uint16_t * polyPoints);
```

Returns

Status of the polygon rendering. GFX_STATUS_SUCCESS - polygon rendering done. GFX_STATUS_FAILURE - polygon rendering is not done.

Description

This function renders a polygon using the currently set line style (see GFX_LineStyleSet()) and color (see GFX_ColorSet()). The shape of the polygon is determined by the polygon points (an ordered array of x,y pairs) where the pair count is equal to the parameter sides.

If any of the x,y pairs do not lie on the frame buffer, then the behavior is undefined. If color is not set, before this function is called, the output is undefined.

Preconditions

Color must be set by GFX_ColorSet(). Line style must be set by GFX_LineStyleSet().

Example

```
uint16_t OpenShapeXYPoints[6] = {10, 10, 20, 10, 20, 20};
uint16_t ClosedShapeXYPoints[8] = {10, 10, 20, 10, 20, 20, 10, 10};

GFX_ColorSet(GFX_INDEX_0, WHITE);                        // set color
SetLineType(GFX_INDEX_0, GFX_LINE_STYLE_THIN_DOTTED);    // set line style
GFX_PolygonDraw(GFX_INDEX_0, 3, OpenShapeXYPoints); // draw an open shape
GFX_PolygonDraw(GFX_INDEX_0, 4, ClosedShapeXYPoints);    // draw a  closed shape
```

Parameters

Parameters	Description
sides	the number of sides of the polygon.
pPoints	Pointer to the array of polygon points. The array defines the x,y points of the polygon. The sequence should be x0, y0, x1, y1, x2, y2, ... xn, yn where n is the # of polygon sides.

Function

GFX_STATUS GFX_PolygonDraw(

SYS_MODULE_INDEX gfxIndex,

uint16_t sides,

uint16_t *pPoints)

GFX_RectangleDraw Function

This function renders a rectangular shape using the currently set line style and color.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_RectangleDraw(SYS_MODULE_INDEX gfxIndex, uint16_t left, uint16_t top, uint16_t right,
uint16_t bottom);
```

Returns

Status of the rectangle rendering. GFX_STATUS_SUCCESS - rectangle rendering done. GFX_STATUS_FAILURE - rectangle rendering is not done.

Description

This function renders a rectangular shape using the given left, top, right and bottom parameters to define the shape dimension. The shape is rendered using the currently set line style by GFX_LineStyleSet(). The color used is the color set by the last call to GFX_ColorSet().

The rendering of this shape becomes undefined when any one of the following is true:

- Any of the following pixel locations left,top, right,bottom falls outside the frame buffer.
- Color is not set, before this function is called.
- right < left
- bottom < top

Preconditions

Color must be set by GFX_ColorSet(). Line style must be set by GFX_LineStyleSet().

Example

```
// draw a bright red rectangle
GFX_LineStyleSet(GFX_INDEX_0, GFX_LINE_STYLE_THIN_SOLID);
GFX_ColorSet(GFX_INDEX_0, BRIGHTRED);
GFX_RectangleDraw(GFX_INDEX_0, 30, 30, 88, 88, 15);

// draw a bright blue round rectangle
GFX_LineStyleSet(GFX_INDEX_0, GFX_LINE_STYLE_THIN_DASHED);
GFX_ColorSet(GFX_INDEX_0, BRIGHTBLUE);
GFX_RectangleRoundDraw(GFX_INDEX_0, 130, 30, 188, 88, 15);
```

Parameters

Parameters	Description
left	defines the left most pixel of the shape.
top	defines the top most pixel of the shape.
right	defines the right most pixel of the shape.
bottom	defines the bottom most pixel of the shape.

Function

GFX_STATUS GFX_RectangleDraw(
SYS_MODULE_INDEX gfxIndex,
uint16_t left,
uint16_t top,
uint16_t right,
uint16_t bottom)

GFX_RectangleRoundDraw Function

This function renders a rounded corner rectangular shape using the currently set line style and color.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_RectangleRoundDraw(SYS_MODULE_INDEX gfxIndex, uint16_t left, uint16_t top, uint16_t right,
uint16_t bottom, uint16_t radius);
```

Returns

Status of the rectangle rendering. GFX_STATUS_SUCCESS - rectangle rendering done. GFX_STATUS_FAILURE - rectangle rendering is not done.

Description

This function renders a rectangular shape with rounded corner using the given left, top, right, bottom and radius parameters to define the shape dimension. radius defines the rounded corner shape. The shape is rendered using the currently set line style by GFX_LineStyleSet(). The color used is the color set by the last call to GFX_ColorSet().

Left most pixel location is defined by left - radius. Top most pixel location is defined by top - radius. Right most pixel location is defined by right + radius. Bottom most pixel location is defined by bottom + radius. When radius = 0, there are no rounded corners. In this case (left,top) will define the left, top corner and (right,bottom) will define the right, bottom corner of the shape.

When left = right and top = bottom, with radius > 0, a circular object is drawn. When left < right and top < bottom and radius = 0, a rectangular object is drawn.

The rendering of this shape becomes undefined when any one of the following is true:

- Any of the following pixel locations left-rad , top-rad, right+rad, bottom+rad falls outside the frame buffer.
- Color is not set, before this function is called.
- right < left
- bottom < top

Preconditions

Color must be set by GFX_ColorSet(). Line style must be set by GFX_LineStyleSet().

Example

```
// draw a bright red rectangle
GFX_LineStyleSet(GFX_INDEX_0, GFX_LINE_STYLE_THIN_SOLID);
```

```
GFX_ColorSet(GFX_INDEX_0, BRIGHTRED);
GFX_RectangleDraw(GFX_INDEX_0, 30, 30, 88, 88, 15);

// draw a bright blue round rectangle
GFX_LineStyleSet(GFX_INDEX_0, GFX_LINE_STYLE_THIN_DASHED);
GFX_ColorSet(GFX_INDEX_0, BRIGHTBLUE);
GFX_RectangleRoundDraw(GFX_INDEX_0, 130, 30, 188, 88, 15);
```

Parameters

Parameters	Description
left	Along with rad (left - rad), defines the left most pixel of the shape.
top	Along with rad (top - rad), defines the top most pixel of the shape.
right	Along with rad (right + rad), defines the right most pixel of the shape.
bottom	Along with rad (bottom + rad), defines the bottom most pixel of the shape.
radius	defines the rounded corners. When this is set to zero, there will be no rounded corners.

Function

GFX_STATUS GFX_RectangleRoundDraw(
SYS_MODULE_INDEX gfxIndex,
uint16_t left,
uint16_t top,
uint16_t right,
uint16_t bottom,
uint16_t radius)

Polygon Fill Rendering Functions

Functions

	Name	Description
	GFX_BarDraw	This function renders a bar shape using the currently set fill style and color.
	GFX_CircleFillDraw	This function renders a filled circle shape using the currently set fill style and colors.
	GFX_RectangleFillDraw	This function renders a filled rectangular shape using the currently set fill style and colors.
	GFX_RectangleRoundFillDraw	This function renders a filled rectangular shape with round corners using the currently set fill style and colors.

Description

Graphics Library Primitive Layer polygon fill rendering API.

GFX_BarDraw Function

This function renders a bar shape using the currently set fill style and color.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_BarDraw(SYS_MODULE_INDEX gfxIndex, uint16_t left, uint16_t top, uint16_t right, uint16_t
bottom);
```

Returns

Status of the bar rendering. GFX_STATUS_SUCCESS - bar rendering done. GFX_STATUS_FAILURE - bar rendering failed.

Description

This function renders a bar shape with the currently set fill style (See GFX_FillStyleGet() and GFX_FillStyleSet() for details of fill style):

- solid color - when the fill style is set to GFX_FILL_STYLE_COLOR
- alpha blended fill - when the fill style is set to GFX_FILL_STYLE_ALPHA_COLOR.

Any other selected fill style will be ignored and will assume a solid color fill will be used. The parameters left, top, right bottom will define the shape dimension.

When fill style is set to GFX_FILL_STYLE_ALPHA_COLOR, the bar can also be rendered with an option to select the type of background. GFX_BACKGROUND_NONE - the bar will be rendered with no alpha blending. GFX_BACKGROUND_COLOR - the bar will be alpha blended with the currently set background color. GFX_BACKGROUND_IMAGE - the bar will be alpha blended with the currently set background image. GFX_BACKGROUND_DISPLAY_BUFFER - the bar will be alpha blended with the current contents of the frame buffer.

The background type is set by the GFX_BackgroundTypeSet().

The rendering of this shape becomes undefined when any one of the following is true:

- Any of the following pixel locations left,top or right,bottom falls outside the frame buffer.
- Colors are not set before this function is called.
- When right < left
- When bottom < top
- When pixel locations defined by left, top and/or right, bottom are not on the frame buffer.

Preconditions

Fill style must be set by GFX_FillStyleSet() when alpha blended fill is desired. Color must be set by GFX_ColorSet().

Example

```
// assume RED is a macro that define GFX_COLOR types
GFX_STATUS status;
// assume BackGroundImage is a valid image already draw
// on the screen
GFX_RESOURCE_HDR *pMyBackgroundImage = &BackGroundImage;

// render a RED bar
GFX_FillStyleSet(GFX_INDEX_0, GFX_FILL_STYLE_COLOR);
GFX_ColorSet(GFX_INDEX_0, RED);
status = GFX_BarDraw(GFX_INDEX_0, 10, 110, 100, 200);

// render an alpha blended bar with
// the current contents of the frame buffer
GFX_FillStyleSet(GFX_INDEX_0, GFX_FILL_STYLE_ALPHA_COLOR);
GFX_BackgroundTypeSet(GFX_INDEX_0, GFX_BACKGROUND_DISPLAY_BUFFER);
GFX_ColorSet(GFX_INDEX_0, RED);
status = GFX_BarDraw(GFX_INDEX_0, 10, 110, 100, 200);

// render an alpha blended bar with a background image
GFX_FillStyleSet(GFX_INDEX_0, GFX_FILL_STYLE_ALPHA_COLOR);

// color value here has no effect since the background set
// is type GFX_BACKGROUND_IMAGE
GFX_BackgroundSet(GFX_INDEX_0, 0, 0, pMyBackGroundImage, 0);

GFX_BackgroundTypeSet(GFX_INDEX_0, GFX_BACKGROUND_IMAGE);
GFX_ColorSet(GFX_INDEX_0, RED);
status = GFX_BarDraw(GFX_INDEX_0, 10, 110, 100, 200);
```

Parameters

Parameters	Description
left	defines the left most pixel of the shape.
top	defines the top most pixel of the shape.
right	defines the right most pixel of the shape.
bottom	defines the bottom most pixel of the shape.

Function

GFX_STATUS GFX_BarDraw(
SYS_MODULE_INDEX gfxIndex,
uint16_t left,
uint16_t top,
uint16_t right,
uint16_t bottom)

GFX_CircleFillDraw Function

This function renders a filled circle shape using the currently set fill style and colors.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_CircleFillDraw(SYS_MODULE_INDEX gfxIndex, uint16_t x, uint16_t y, uint16_t radius);
```

Returns

Status of the circle rendering. GFX_STATUS_SUCCESS - circle fill rendering done. GFX_STATUS_FAILURE - circle fill rendering is not done.

Description

This function renders a filled circle shape with the currently set fill style (see GFX_FILL_STYLE) with the given left, top, right and bottom parameters to define the shape dimension. The shape is rendered depending on the fill style. If a flat color is used, color must be set (see GFX_ColorSet()) before calling this function. If gradient color is used, gradient start and end color must be set (see GFX_GradientColorSet()) before calling this function. After the fill style and colors are set, multiple calls to this function can be performed.

The rendering of this shape becomes undefined when any one of the following is true:

- Any of the following pixel locations left,top or right,bottom falls outside the frame buffer.
- Fill style is not set (GFX_FillStyleSet(), before this function is called.
- Colors are not set before this function is called.
- When the center defined by x,y is not on the frame buffer.

Preconditions

Fill style must be set by GFX_FillStyleSet(). Color must be set by GFX_ColorSet().

Example

```
// assume BLUE and RED are macros that define GFX_COLOR types
GFX_STATUS status;

GFX_FillStyleSet(GFX_INDEX_0, GFX_FILL_STYLE_GRADIENT_UP);
GFX_GradientColorSet(GFX_INDEX_0, BLUE, RED);
status = GFX_CircleFillDraw(GFX_INDEX_0, 50, 110, 150, 200, 20);
if (status == GFX_STATUS_SUCCESS)
    // Filled circle was drawn.
else
    // Filled circle is not drawn or not yet
    // finished rendering. To finish the rendering call the
    // function again with the same parameters.
```

Parameters

Parameters	Description
x	defines the x-coordinate position of the center of the circle.
y	defines the y-coordinate position of the center of the circle.
radius	defines the radius of the circle.

Function

GFX_STATUS GFX_CircleFillDraw(
SYS_MODULE_INDEX gfxIndex,
uint16_t x,
uint16_t y,
uint16_t radius)

GFX_RectangleFillDraw Function

This function renders a filled rectangular shape using the currently set fill style and colors.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_RectangleFillDraw(SYS_MODULE_INDEX gfxIndex, uint16_t left, uint16_t top, uint16_t right,
uint16_t bottom);
```

Returns

Status of the rectangle rendering. GFX_STATUS_SUCCESS - rectangle fill rendering done. GFX_STATUS_FAILURE - rectangle fill rendering is not done.

Description

This function renders a filled rectangular shape with the currently set fill style (see GFX_FILL_STYLE) with the given left, top, right, and bottom parameters to define the shape dimension. The shape is rendered depending on the fill style. If a flat color is used, color must be set (see GFX_ColorSet()) before calling this function. If gradient color is used, gradient start and end color must be set (see GFX_GradientColorSet()) before calling this function. After the fill style and colors are set, multiple calls to this function can be performed.

The rendering of this shape becomes undefined when any one of the following is true:

- Any of the following pixel locations left,top or right,bottom falls outside the frame buffer.
- Fill style is not set (GFX_FillStyleSet(), before this function is called.
- Colors are not set before this function is called.
- When right < left
- When bottom < top
- When pixel locations defined by left, top and/or right, bottom are not on the frame buffer.

Preconditions

Fill style must be set by GFX_FillStyleSet(). Color must be set by GFX_ColorSet().

Example

```
// assume BLUE and RED are macros that define GFX_COLOR types
GFX_STATUS status;

// render a rectangle with gradient colors from BLUE to RED
GFX_FillStyleSet(GFX_INDEX_0, GFX_FILL_STYLE_GRADIENT_LEFT);
GFX_GradientColorSet(GFX_INDEX_0, BLUE, RED);
status = GFX_RectangleFillDraw(GFX_INDEX_0, 10, 110, 100, 200);

// render an alpha blended rounded rectangle with
// the current contents of the frame buffer
GFX_FillStyleSet(GFX_INDEX_0, GFX_FILL_STYLE_ALPHA_COLOR);
GFX_AlphaBlendingValueSet(GFX_INDEX_0, 50);
GFX_BackgroundTypeSet(GFX_INDEX_0, GFX_BACKGROUND_DISPLAY_BUFFER);
GFX_ColorSet(GFX_INDEX_0, RED);
status = GFX_RectangleRoundFillDraw(GFX_INDEX_0, 10, 110, 100, 200, 10);

// render an alpha blended rectangle with an image
GFX_FillStyleSet(GFX_INDEX_0, GFX_FILL_STYLE_ALPHA_COLOR);
GFX_AlphaBlendingValueSet(5GFX_INDEX_0, 0);
GFX_BackgroundTypeSet(GFX_INDEX_0, GFX_BACKGROUND_IMAGE);
GFX_ColorSet(GFX_INDEX_0, RED);
status = GFX_RectangleFillDraw(GFX_INDEX_0, 10, 110, 100, 200);

// render a plain RED rounded rectangle
GFX_FillStyleSet(GFX_INDEX_0, GFX_FILL_STYLE_COLOR);
GFX_ColorSet(GFX_INDEX_0, RED);
status = GFX_RectangleFillDraw(GFX_INDEX_0, 10, 110, 100, 200);
```

Parameters

Parameters	Description
left	defines the left most pixel of the shape.
top	defines the top most pixel of the shape.
right	defines the right most pixel of the shape.
bottom	defines the bottom most pixel of the shape.

Function

GFX_STATUS GFX_RectangleFillDraw(
SYS_MODULE_INDEX gfxIndex,
uint16_t left,
uint16_t top,
uint16_t right,
uint16_t bottom)

GFX_RectangleRoundFillDraw Function

This function renders a filled rectangular shape with round corners using the currently set fill style and colors.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_RectangleRoundFillDraw(SYS_MODULE_INDEX gfxIndex, uint16_t left, uint16_t top, uint16_t
right, uint16_t bottom, uint16_t radius);
```

Returns

Status of the rectangle rendering. GFX_STATUS_SUCCESS - rectangle fill rendering done. GFX_STATUS_FAILURE - rectangle fill rendering is not done.

Description

This function renders a filled rounded rectangular shape with the currently set fill style (see GFX_FILL_STYLE) with the given left, top, right, bottom and radius parameters to define the shape dimension. The shape is rendered depending on the fill style. If a flat color is used, color must be set (see GFX_ColorSet()) before calling this function. If gradient color is used, gradient start and end color must be set (see GFX_GradientColorSet()) before calling this function. After the fill style and colors are set, multiple calls to this function can be performed.

When left = right and top = bottom, with radius > 0, a circular object is drawn. When left < right and top < bottom and radius = 0, a rectangular object is drawn.

The rendering of this shape becomes undefined when any one of the following is true:

- Any of the following pixel locations left,top or right,bottom falls outside the frame buffer.
- Fill style is not set (GFX_FillStyleSet(), before this function is called.
- Colors are not set before this function is called.
- When right < left
- When bottom < top
- When pixel locations defined by left, top and/or right, bottom are not on the frame buffer.

Preconditions

Fill style must be set by GFX_FillStyleSet(). Color must be set by GFX_ColorSet().

Example

```
// assume BLUE and RED are macros that define GFX_COLOR types
GFX_STATUS status;

GFX_FillStyleSet(GFX_FILL_STYLE_GRADIENT_DOUBLE_VER);
GFX_GradientColorSet(BLUE, RED);
status = GFX_RectangleFillDraw(50, 110, 150, 200, 20);
if (status == GFX_STATUS_SUCCESS)
    // Filled rounded rectangle shape was drawn.
else
    // Filled rounded rectangle shape is not drawn or not yet
    // finished rendering. To finish the rendering call the
    // function again with the same parameters.
```

Parameters

Parameters	Description
left	defines the left most pixel of the shape.
top	defines the top most pixel of the shape.
right	defines the right most pixel of the shape.
bottom	defines the bottom most pixel of the shape.
radius	defines the radius of the rounded corner. A zero value will result in a rectangular shape drawn.

Function

GFX_STATUS GFX_RectangleRoundFillDraw(
SYS_MODULE_INDEX gfxIndex,
uint16_t left,
uint16_t top,
uint16_t right,
uint16_t bottom,
uint16_t radius)

Image Rendering Functions

Functions

	Name	Description
	GFX_ImageDraw	This function renders an image to the frame buffer.
	GFX_ImageHeaderGet	This function fills the given bitmap header with the image's header information.
	GFX_ImageHeightGet	This function returns the height of the given image.
	GFX_ImagePartialDraw	This function renders a portion of an image to the frame buffer.
	GFX_ImageWidthGet	This function returns the width of the given image.

Description

Graphics Library Primitive Layer image rendering API.

GFX_ImageDraw Function

This function renders an image to the frame buffer.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_ImageDraw(SYS_MODULE_INDEX gfxIndex, uint16_t left, uint16_t top, GFX_RESOURCE_HDR * pImage);
```

Returns

Status of the image rendering. GFX_STATUS_SUCCESS - image rendering done. GFX_STATUS_FAILURE - image rendering failed.

Description

This function renders an image to the frame buffer with the left-top corner of the image located at given left, top parameters.

The rendering of this shape becomes undefined when any one of the following is true:

- left, top pixel position falls outside the frame buffer.
- pointer is not properly initialized to a GFX_RESOURCE_HDR object.

Preconditions

None.

Example

```
// assume the backgroundImage has dimension of 320x240 pixels.
GFX_RESOURCE_HDR *pBackgroundImage;

pBackgroundImage = (GFX_RESOURCE_HDR *)&backgroundImage;

// Render the image starting from (10,10) x,y position

// corner of the image
GFX_ImageDraw(  GFX_INDEX_0, 10, 10,
                pBackgroundImage);
```

Parameters

Parameters	Description
left	Horizontal starting position of the image.
top	Vertical starting position of the image.
pImage	Pointer to the image to be rendered.

Function

```
GFX_STATUS GFX_ImageDraw(
                SYS_MODULE_INDEX gfxIndex,
uint16_t left,
uint16_t top,
                GFX_RESOURCE_HDR *pImage)
```

GFX_ImageHeaderGet Function

This function fills the given bitmap header with the image's header information.

File

gfx_primitive.h

C

```
inline void GFX_ImageHeaderGet(GFX_RESOURCE_HDR * pImage, GFX_MCHP_BITMAP_HEADER * pBitmapHdr);
```

Returns

None.

Description

This function fills the given bitmap header with the image's header information. This function results in an undefined behavior if the pointer to the image is invalid.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
pImage	Pointer to the image.
pBitmap	Pointer to the destination of the header information.

Function

GFX_STATUS GFX_ImageHeaderGet(
GFX_RESOURCE_HDR *pImage,
GFX_MCHP_BITMAP_HEADER *pBitmapHdr)

GFX_ImageHeightGet Function

This function returns the height of the given image.

File

gfx_primitive.h

C

```
inline int16_t GFX_ImageHeightGet(GFX_RESOURCE_HDR * pImage);
```

Returns

The image height in pixels.

Description

This function returns the height of the given image in pixels. This function results in an undefined behavior if the pointer to the image is invalid.

This function return value is undefined if the given pointer does not point to a valid image resource.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
pImage	Pointer to the image.

Function

GFX_STATUS GFX_ImageHeightGet(
GFX_RESOURCE_HDR *pImage)

GFX_ImagePartialDraw Function

This function renders a portion of an image to the frame buffer.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_ImagePartialDraw(SYS_MODULE_INDEX gfxIndex, uint16_t destination_x, uint16_t destination_y,
uint16_t source_x_offset, uint16_t source_y_offset, uint16_t source_width, uint16_t source_height,
GFX_RESOURCE_HDR * pImage);
```

Returns

Status of the image rendering. GFX_STATUS_SUCCESS - image rendering done. GFX_STATUS_FAILURE - image rendering failed.

Description

This function renders an image or a portion of an image to the frame buffer with the top, left corner of the image located at destination_x, destination_y. To render a full image, source_x_offset, source_y_offset, width and height are set to all 0. Using the actual image's width and height will also work if these are known. If they are not known, avoid the extra extra step to get the image's width and height by assigning 0 to the parameters. Another way to render a full image is to use the GFX_ImageDraw(). The image to be rendered is defined by the pointer pImage.

The rendering of this shape becomes undefined when any one of the following is true:

- destination_x, destination_y pixel position falls outside the frame buffer.
- source_x_offset, source_y_offset results in a starting position beyond the image's dimension.
- source_width and/or source_height is larger than the actual image's width and height.
- pointer is not properly initialized to a GFX_RESOURCE_HDR object.

Preconditions

None.

Example

```
// assume the backgroundImage has dimension of 320x240 pixels.
GFX_RESOURCE_HDR *pBackgroundImage;

pBackgroundImage = (GFX_RESOURCE_HDR *)&backgroundImage;

// Render only 1/4 of the image. Render the lower right
// corner of the image
GFX_ImagePartialDraw(   GFX_INDEX_0, 10, 10,
                        320/2,
                        240/2,
                        320/2,
                        240/2,
                        pBackgroundImage);
```

Parameters

Parameters	Description
destination_x	Horizontal starting position of the image.
destination_y	Vertical starting position of the image.
source_x_offset	Horizontal offset in pixels of the starting position of the partial area of the image to be rendered. Set to 0 along with source_y_offset when rendering the full image.
source_y_offset	Vertical offset in pixels of the starting position of the partial area of the image to be rendered. Set to 0 along with source_x_offset when rendering the full image.GFX_ImageDraw
source_width	The width of the partial area of the image to be rendered. Set to 0 along with source_height when rendering the full image.
source_height	The height of the partial area of the image to be rendered. Set to 0 along with source_width when rendering the full image.
pImage	Pointer to the image to be rendered.

Function

GFX_STATUS GFX_ImagePartialDraw(

SYS_MODULE_INDEX gfxIndex,

uint16_t destination_x,

uint16_t destination_y,

uint16_t source_x_offset,
uint16_t source_y_offset,
uint16_t source_width,
uint16_t source_height,
GFX_RESOURCE_HDR *pImage)

GFX_ImageWidthGet Function

This function returns the width of the given image.

File

gfx_primitive.h

C

```
inline int16_t GFX_ImageWidthGet(GFX_RESOURCE_HDR * pImage);
```

Returns

The image width in pixels.

Description

This function returns the width of the given image in pixels. This function results in an undefined behavior if the pointer to the image is invalid.

This function return value is undefined if the given pointer does not point to a valid image resource.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
pImage	Pointer to the image.

Function

GFX_STATUS GFX_ImageWidthGet(
GFX_RESOURCE_HDR *pImage)

Text Rendering Functions

Functions

	Name	Description
	GFX_FontAntiAliasGet	This function returns the current font font anti-aliasing mode.
	GFX_FontAntiAliasSet	This function sets the font anti-aliasing mode.
	GFX_FontSet	This function sets the current font used when rendering strings and characters.
	GFX_FontGet	This function returns the current font used when rendering strings and characters.
	GFX_TextCharDraw	This function renders the given character using the currently set color using the currently set font.
	GFX_TextCursorPositionSet	This function sets the text cursor to a new position.
	GFX_TextCursorPositionXGet	This function returns the current x position of the text cursor.
	GFX_TextCursorPositionYGet	This function returns the current y position of the text cursor.
	GFX_TextStringBoxDraw	This function renders the given string using the currently set color and font into a rectangular area.
	GFX_TextStringDraw	This function renders the given string of character using the currently set color using the currently set font.
	GFX_TextStringHeightGet	This function returns the height of the given font.
	GFX_TextStringWidthGet	This function returns the width of the given string using the given font.
	GFX_TextCursorDraw	This is function GFX_TextCursorDraw.
	GFX_TextCursorTypeGet	This is function GFX_TextCursorTypeGet.
	GFX_TextCursorTypeSet	This is function GFX_TextCursorTypeSet.
	GFX_TextCursorWidthGet	This is function GFX_TextCursorWidthGet.

Description

Graphics Library Primitive Layer text rendering API.

GFX_FontAntiAliasGet Function

This function returns the current font font anti-aliasing mode.

File

gfx_primitive.h

C

```
GFX_FONT_ANTIALIAS_TYPE GFX_FontAntiAliasGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The font anti-aliasing mode used.

Description

This function returns the font anti-aliasing mode used when rendering anti-aliased strings and characters. Default setting at initialization is GFX_FONT_ANTIALIAS_OPAQUE.

Preconditions

None.

Example

```
extern const GFX_RESOURCE_HDR antiAliasedFont25;
uint16_t width, height;
GFX_XCHAR myString[] = "Microchip Technology Inc.";

if (GFX_FontAntiAliasGet(GFX_INDEX_0) != GFX_FONT_ANTIALIAS_OPAQUE)
    GFX_FontAntiAliasSet(GFX_INDEX_0, GFX_FONT_ANTIALIAS_OPAQUE);

GFX_FontSet(GFX_INDEX_0, (GFX_RESOURCE_HDR*) &antiAliasedFont25);
GFX_ColorSet(GFX_INDEX_0, GFX_X11_GREEN);
width = GFX_TextStringWidthGet(myString, (GFX_RESOURCE_HDR*) &Font25);
height = GFX_TextStringHeightGet((GFX_RESOURCE_HDR*) &antiAliasedFont25);

GFX_TextStringDraw( GFX_INDEX_0, (GFX_MaxXGet() - width) >> 1,
                    (GFX_MaxYGet() - height) >> 1,
                    myString,
                    0);
```

Function

GFX_FONT_ANTIALIAS_TYPE GFX_FontAntiAliasGet(SYS_MODULE_INDEX gfxIndex)

GFX_FontAntiAliasSet Function

This function sets the font anti-aliasing mode.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_FontAntiAliasSet(SYS_MODULE_INDEX gfxIndex, GFX_FONT_ANTIALIAS_TYPE type);
```

Returns

The status of the set anti-aliasing mode action.

Description

This function sets the font anti-aliasing mode used when rendering anti-aliased strings and characters.

Preconditions

None.

Example

```
extern const GFX_RESOURCE_HDR antiAliasedFont25;
uint16_t width, height;
```

```
GFX_XCHAR myString[] = "Microchip Technology Inc.";

GFX_FontAntiAliasSet(GFX_INDEX_0, GFX_FONT_ANTIALIAS_TRANSLUCENT);
GFX_FontSet(GFX_INDEX_0, (GFX_RESOURCE_HDR*) &antiAliasedFont25);
GFX_ColorSet(GFX_INDEX_0, GFX_X11_GREEN);
width = GFX_TextStringWidthGet( myString,
                                (GFX_RESOURCE_HDR*) &Font25);
height = GFX_TextStringHeightGet((GFX_RESOURCE_HDR*) &antiAliasedFont25);

GFX_TextStringDraw( GFX_INDEX_0, (GFX_MaxXGet() - width) >> 1,
                    (GFX_MaxYGet() - height) >> 1,
                    myString,
                    0);
```

Parameters

Parameters	Description
type	anti-aliasing mode selected. See GFX_FONT_ANTIALIAS_TYPE for details.

Function

GFX_STATUS GFX_FontAntiAliasSet(
SYS_MODULE_INDEX gfxIndex,
GFX_FONT_ANTIALIAS_TYPE type)

GFX_FontSet Function

This function sets the current font used when rendering strings and characters.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_FontSet(SYS_MODULE_INDEX gfxIndex, GFX_RESOURCE_HDR * pFont);
```

Returns

The status of the set font action.

Description

This function sets the current font used to render strings and characters.

Preconditions

None.

Example

```
extern const GFX_RESOURCE_HDR Font25;
uint16_t width, height;
GFX_XCHAR myString[] = "Microchip Technology Inc.";

GFX_FontSet(GFX_INDEX_0, (GFX_RESOURCE_HDR*) &Font25);
GFX_ColorSet(GFX_INDEX_0, GFX_X11_GREEN);
width = GFX_TextStringWidthGet(myString, (GFX_RESOURCE_HDR*) &Font25);
height = GFX_TextStringHeightGet((GFX_RESOURCE_HDR*) &Font25);

GFX_TextStringDraw( GFX_INDEX_0, (GFX_MaxXGet() - width) >> 1,
                    (GFX_MaxYGet() - height) >> 1,
                    myString,
                    0);
```

Parameters

Parameters	Description
pFont	pointer to the font to be used.

Function

GFX_STATUS GFX_FontSet(
SYS_MODULE_INDEX gfxIndex,
GFX_RESOURCE_HDR *pFont)

GFX_FontGet Function

This function returns the current font used when rendering strings and characters.

File

gfx_primitive.h

C

```
GFX_RESOURCE_HDR* GFX_FontGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The pointer to the current font used in rendering strings and characters.

Description

This function returns the current font used to render strings and characters.

Preconditions

GFX_FontSet() must be called prior to any call to this function. If this function is called first, the returned value is undefined.

Example

None.

Function

GFX_RESOURCE_HDR GFX_FontGet(SYS_MODULE_INDEX gfxIndex)

GFX_TextCharDraw Function

This function renders the given character using the currently set color using the currently set font.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_TextCharDraw(SYS_MODULE_INDEX gfxIndex, GFX_XCHAR ch);
```

Returns

The status of the character rendering. GFX_STATUS_SUCCESS - the character was rendered GFX_STATUS_FAILURE - the character was not rendered, function must be called again to render the character. GFX_STATUS_ERROR - the character ID passed is not a valid or points to the character glyph that does does not exist on the font table.

Description

This function renders the given character using the currently set font, and color to the location defined by the text cursor position. The color is set by GFX_ColorSet() while the font is set by GFX_FontSet(). The text cursor position is set by GFX_TextCursorPositionSet()

The rendering of the character becomes undefined when any one of the following is true:

- Text cursor position is set to locations outside the frame buffer.
- Color is not set, before this function is called.
- Font is not set, before this function is called.

Preconditions

Color must be set by GFX_ColorSet(). Font must be set by GFX_FontSet(). Text cursor position must be set by GFX_TextCursorPositionSet().

Example

```
// assume textString is a string of characters
// assume WHITE is a valid GFX_COLOR data
// assume pMyFont is a valid initialized font resource pointer
static uint16_t counter = 0;
GFX_XCHAR   ch;
GFX_STATUS  status;

GFX_ColorSet(GFX_INDEX_0, WHITE);
GFX_FontSet(GFX_INDEX_0, pMyFont);

// render characters until null character
while((GFX_XCHAR)(ch = *(textString + counter)) != 0)
{
```

```
        status = GFX_TextCharDraw(GFX_INDEX_0, ch);
        if (status != GFX_STATUS_SUCCESS)
            return (GFX_STATUS_FAILURE);
        counter++;
    }
```

Parameters

Parameters	Description
ch	character ID that of the character that will be rendered.

Function

GFX_STATUS GFX_TextCharDraw(
SYS_MODULE_INDEX gfxIndex,
GFX_XCHAR ch)

GFX_TextCursorPositionSet Function

This function sets the text cursor to a new position.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_TextCursorPositionSet(SYS_MODULE_INDEX gfxIndex, int16_t x, int16_t y);
```

Returns

Status of the text cursor position set. GFX_STATUS_SUCCESS - text cursor position set done. GFX_STATUS_FAILURE - text cursor position set is not done.

Description

This function sets the text cursor to a new (x,y) position. Text cursor is used as a starting point of the character rendered by the GFX_TextCharDraw() function.

If x and/or y does not lie on the frame buffer, then the behavior of GFX_TextCharDraw() function is undefined.

Preconditions

None.

Example

```
GFX_TextCursorPositionSet(GFX_INDEX_0, 10,10);
// write letter 'A' starting from 10, 10 position
GFX_TextCharDraw(GFX_INDEX_0, 'A');
```

Parameters

Parameters	Description
x	new x coordinate position of the text cursor.
y	new y coordinate position of the text cursor.

Function

GFX_STATUS GFX_TextCursorPositionSet(
SYS_MODULE_INDEX gfxIndex,
int16_t x,
int16_t y)

GFX_TextCursorPositionXGet Function

This function returns the current x position of the text cursor.

File

gfx_primitive.h

C

```
int16_t GFX_TextCursorPositionXGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The current text cursor x position.

Description

This function returns the current x position of the text cursor. Text cursor is used as a starting point of the line rendered by the GFX_TextCharDraw() function.

Preconditions

None.

Example

None.

Function

int16_t GFX_TextCursorPositionXGet(SYS_MODULE_INDEX gfxIndex)

GFX_TextCursorPositionYGet Function

This function returns the current y position of the text cursor.

File

gfx_primitive.h

C

```
int16_t GFX_TextCursorPositionYGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The current text cursor y position.

Description

This function returns the current y position of the text cursor. Text cursor is used as a starting point of the line rendered by the GFX_TextCharDraw() function.

Preconditions

None.

Example

None.

Function

int16_t GFX_TextCursorPositionYGet(SYS_MODULE_INDEX gfxIndex)

GFX_TextStringBoxDraw Function

This function renders the given string using the currently set color and font into a rectangular area.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_TextStringBoxDraw(SYS_MODULE_INDEX gfxIndex, uint16_t x, uint16_t y, uint16_t width,
uint16_t height, GFX_XCHAR * pString, uint16_t length, GFX_ALIGNMENT align);
```

Returns

The status of the string rendering. GFX_STATUS_SUCCESS - the string was rendered GFX_STATUS_FAILURE - the string was not rendered, or is not yet finished. The function must be called again to render the remaining characters or lines.

Description

This function renders the given string using the currently color and font into the rectangular area defined by the given parameters x,y, width and height. The x,y parameters defines the left, top pixel position of the rectangular area. The width and height defines the size of the rectangular area. The rectangular area will define the area where the text will be rendered. Meaning all pixels EXCLUSIVE of the defined rectangle will be the rendering area for the text. If the given string exceeds the rectangular area, any pixels falling outside and along the defined rectangle (INCLUSIVE of the rectangle) will not be drawn.

The color is set by GFX_ColorSet() while the font is set by GFX_FontSet().

Multiple lines of strings can be rendered. To define string composed of multiple lines, end each line with a new line character (character ID 0x0A). Each line will be rendered according to the text alignment (See GFX_TEXT_ALIGNMENT).

When the length parameter is set to 0, the string will be rendered until the null character is reached. When the length parameter is greater than 0, the string will be rendered until one of the following cases occurs:

- null character is reached before the total rendered characters is less than the value of length
- total rendered characters is equal to the value of length before the null character is reached.

The rendering of the character becomes undefined when any one of the following is true:

- x, y, width and height defines an area partially or fully outside outside the frame buffer.
- Color is not set, before this function is called.
- Font is not set, before this function is called.

Preconditions

Color must be set by GFX_ColorSet(). Font must be set by GFX_FontSet().

Example

```
// assume APP_DEMO_FONT to be a valid font resource
GFX_XCHAR StringMsg[] = "Hello World";

GFX_ColorSet(GFX_INDEX_0, BLACK);
GFX_ScreenClear(GFX_INDEX_0);

GFX_ColorSet(GFX_INDEX_0, WHITE);
GFX_FontSet(GFX_INDEX_0, (GFX_RESOURCE_HDR*)&APP_DEMO_FONT);

// displayed message will be centered on the screen
while(GFX_TextStringBoxDraw( GFX_INDEX_0, 0, 0,
                             GFX_MaxXGet(), GFX_MaxYGet(),
                             StringMsg,
                             0, GFX_ALIGN_CENTER) !=
        GFX_STATUS_SUCCESS);
```

Parameters

Parameters	Description
x	Horizontal starting position of the rectangular area.
y	Vertical position position of the rectangular area.
width	Defines the width of the rectangular area.
height	Defines the height of the rectangular area.
pString	Pointer to the location of the string that will be rendered. String can have multiple lines where each line is terminated by a new line character.
length	Total number of characters to be rendered. When set to 0, the function will terminate until the null character is detected.
align	The alignment of the rendered text.

Function

```
GFX_STATUS FX_TextStringBoxDraw(
                SYS_MODULE_INDEX gfxIndex,
uint16_t x,
uint16_t y,
uint16_t width,
uint16_t height,
                GFX_XCHAR *pString,
uint16_t length,
                GFX_ALIGNMENT align)
```

GFX_TextStringDraw Function

This function renders the given string of character using the currently set color using the currently set font.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_TextStringDraw(SYS_MODULE_INDEX gfxIndex, uint16_t x, uint16_t y, GFX_XCHAR * pString,
uint16_t length);
```

Returns

The status of the string rendering. GFX_STATUS_SUCCESS - the string was rendered GFX_STATUS_FAILURE - the string was not rendered, or is not yet finished. The function must be called again to render the remaining characters.

Description

This function renders the given string of character using the currently set font, and color to the location defined by the given x,y position. The color is set by GFX_ColorSet() while the font is set by GFX_FontSet().

The text rendered by this function is always left aligned. When a newline character is encountered, the next character after the newline character will be rendered left aligned on the next line. The next line position is determined by the height of the font used.

When the length parameter is set to 0, the string will be rendered until the null character is reached. When the length parameter is greater than 0, the string will be rendered until one of the following cases occurs:

- null character is reached before the total rendered characters is less than the value of length
- total rendered characters is equal to the value of length before the null character is reached.

The rendering of the character becomes undefined when any one of the following is true:

- x,y position is set to locations outside the frame buffer.
- Color is not set, before this function is called.
- Font is not set, before this function is called.

Preconditions

Color must be set by GFX_ColorSet(). Font must be set by GFX_FontSet().

Example

```
// assume WHITE is a valid GFX_COLOR data
// assume pMyFont is a valid initialized font resource pointer

static uint16_t counter = 0;
GFX_XCHAR   charArray[] = "Test String";
GFX_XCHAR   pChar = NULL;
GFX_STATUS  status;

GFX_ColorSet(GFX_INDEX_0, WHITE);
GFX_FontSet(GFX_INDEX_0, pMyFont);

// render the whole string
GFX_TextStringDraw(GFX_INDEX_0, 10, 10, charArray, 0);

// render ONLY the "Test" portion of the string
GFX_TextStringDraw(GFX_INDEX_0, 10, 30, charArray, 4);

// render ONLY the "String" portion of the string
pChar = charArray;
pChar += 5;
GFX_TextStringDraw(GFX_INDEX_0, 10, 50, pChar, 0);

// doing this also renders ONLY the "String" portion of the string
GFX_TextStringDraw(GFX_INDEX_0, 10, 70, pChar, 6);
```

Parameters

Parameters	Description
x	Horizontal starting position of the string.
y	Vertical position position of the string.
pString	Pointer to the location of the string that will be rendered.
length	Total number of characters to be rendered. When set to 0, the function will terminate until the null character is detected.

Function

GFX_STATUS GFX_TextStringDraw(
SYS_MODULE_INDEX gfxIndex,

uint16_t x,
uint16_t y,
GFX_XCHAR *pString,
uint16_t length)

GFX_TextStringHeightGet Function

This function returns the height of the given font.

File

gfx_primitive.h

C

```
inline uint16_t GFX_TextStringHeightGet(GFX_RESOURCE_HDR * pFont);
```

Returns

The height of the specified font in pixels.

Description

This function returns the height of the given font in pixels. The given font must be present in the system.

This function return value is undefined if the given pointer does not point to a valid font.

Preconditions

None.

Example

```
// assume pMyFont is a pointer initialized to a valid font
uint16_t height, x, y;

// center the text on the screen

height = GFX_TextStringHeightGet(pMyFont);
width  = GFX_TextStringWidthGet (   (GFX_XCHAR *)"Hello World",
                                     pMyFont
                                 );
y = (GFX_MaxYGet() - height) >> 1;
x = (GFX_MaxXGet() - width) >> 1;
GFX_TextStringDraw(GFX_INDEX_0, x, y, (GFX_XCHAR *)"Hello World");

GFX_TextStringDraw(GFX_INDEX_0, x, y, (GFX_XCHAR *)"Hello World");
```

Parameters

Parameters	Description
pFont	Pointer to the specified font.

Function

uint16_t GFX_TextStringHeightGet(
GFX_RESOURCE_HDR *pFont);

GFX_TextStringWidthGet Function

This function returns the width of the given string using the given font.

File

gfx_primitive.h

C

```
uint16_t GFX_TextStringWidthGet(GFX_XCHAR * textString, GFX_RESOURCE_HDR * pFont);
```

Returns

The width of the specified string using the specified font in pixels.

Description

This function returns the width of the given string using the given font in pixels. The given font must be present in the system.

The pixel length is measured from the first printable character until the last printable character. This means that if the string is composed of

multiple lines, the length returned is only valid for the first line of characters.

This function return value is undefined if the given pointer does not point to a valid font or one or more characters in the given string does not exist on the given font.

Preconditions

None.

Example

```
// assume pMyFont is a pointer initialized to a valid font
uint16_t height, x, y;

// center the text on the screen

height = GFX_TextStringHeightGet(GFX_INDEX_0, pMyFont);
width  = GFX_TextStringWidthGet (GFX_INDEX_0,    (GFX_XCHAR *)"Hello World",
                                      pMyFont
                                );
y = (GFX_MaxYGet() - height) >> 1;
x = (GFX_MaxXGet() - width) >> 1;
GFX_TextStringDraw(GFX_INDEX_0, x, y, (GFX_XCHAR *)"Hello World");
```

Parameters

Parameters	Description
pString	Pointer to the specified string.
pFont	Pointer to the specified font.

Function

uint16_t GFX_TextStringWidthGet(
GFX_XCHAR* pString,
GFX_RESOURCE_HDR *pFont);

GFX_TextCursorDraw Function

File

gfx_primitive.h

C

```
GFX_STATUS GFX_TextCursorDraw(SYS_MODULE_INDEX gfxIndex, uint16_t x, uint16_t y, uint16_t cursorWidth,
uint16_t cursorHeight);
```

Description

This is function GFX_TextCursorDraw.

GFX_TextCursorTypeGet Function

File

gfx_primitive.h

C

```
GFX_TEXT_CURSOR_TYPE GFX_TextCursorTypeGet(SYS_MODULE_INDEX gfxIndex);
```

Description

This is function GFX_TextCursorTypeGet.

GFX_TextCursorTypeSet Function

File

gfx_primitive.h

C

```
GFX_STATUS GFX_TextCursorTypeSet(SYS_MODULE_INDEX gfxIndex, GFX_TEXT_CURSOR_TYPE textCursorType);
```

Description

This is function GFX_TextCursorTypeSet.

GFX_TextCursorWidthGet Function

File

gfx_primitive.h

C

```
int16_t GFX_TextCursorWidthGet(SYS_MODULE_INDEX gfxIndex);
```

Description

This is function GFX_TextCursorWidthGet.

Rendering Style Functions

Functions

	Name	Description
	GFX_FillStyleGet	Return the fill style used when rendering filled shapes.
	GFX_FillStyleSet	Set the fill style to be used when rendering filled shapes.
	GFX_AlphaBlendingValueGet	This function returns the current alpha value set for alpha blending rendering.
	GFX_AlphaBlendingValueSet	This function sets the alpha value for alpha blending rendering.
	GFX_GradientColorSet	Sets the gradient fill start and end color.
	GFX_GradientEndColorGet	Return the gradient end color when rendering gradient filled shapes.
	GFX_GradientStartColorGet	Return the gradient start color when rendering gradient filled shapes.
	GFX_LineStyleGet	Return the line style used when rendering lines.
	GFX_LineStyleSet	Set the line style to be used when rendering lines.

Description

Graphics Library Primitive Layer rendering style API.

GFX_FillStyleGet Function

Return the fill style used when rendering filled shapes.

File

gfx_primitive.h

C

```
GFX_FILL_STYLE GFX_FillStyleGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The current fill style used (see GFX_FILL_STYLE).

Description

This function returns the fill style used (see GFX_FILL_STYLE) when rendering filled shapes.

Preconditions

None.

Example

None.

Function

GFX_FILL_STYLE GFX_FillStyleGet(SYS_MODULE_INDEX gfxIndex)

GFX_FillStyleSet Function

Set the fill style to be used when rendering filled shapes.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_FillStyleSet(SYS_MODULE_INDEX gfxIndex, GFX_FILL_STYLE style);
```

Returns

Status of the fill style set. GFX_STATUS_SUCCESS - fill style set done. GFX_STATUS_FAILURE - fill style set failed.

Description

This function sets the fill style to be used (see GFX_FILL_STYLE) when rendering filled shapes. All calls to the following:

- GFX_RectangleFillDraw()
- GFX_RectangleRoundFillDraw()
- GFX_CircleFillDraw()

will use the fill style set by this function.

Preconditions

None.

Example

```
GFX_GradientColorSet(BRIGHTRED, BRIGHTBLUE);

GFX_FillStyleSet(GFX_INDEX_0, GFX_FILL_STYLE_GRADIENT_DOWN);
GFX_RectangleFillDraw(GFX_INDEX_0,   10, 10,  100,  100);

GFX_FillStyleSet(GFX_INDEX_0, GFX_FILL_STYLE_GRADIENT_UP);
GFX_RectangleFillDraw(GFX_INDEX_0,  110, 10,  200,  100);

GFX_FillStyleSet(GFX_INDEX_0, GFX_FILL_STYLE_GRADIENT_RIGHT);
GFX_RectangleFillDraw( GFX_INDEX_0, 210, 10,  300,  100);

GFX_FillStyleSet(GFX_INDEX_0, GFX_FILL_STYLE_GRADIENT_LEFT);
GFX_RectangleFillDraw(GFX_INDEX_0,   10, 110,  100,  200);
```

Parameters

Parameters	Description
style	The specified fill style to be used.

Function

GFX_STATUS GFX_FillStyleSet(
SYS_MODULE_INDEX gfxIndex,
GFX_FILL_STYLE style)

GFX_AlphaBlendingValueGet Function

This function returns the current alpha value set for alpha blending rendering.

File

gfx_primitive.h

C

```
uint16_t GFX_AlphaBlendingValueGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The alpha blending value.

Description

This function returns the current alpha value set for alpha blending rendering. See GFX_AlphaBlendingValueSet() for details of alpha blending values.

Preconditions

None.

Example

None.

Function

uint16_t GFX_AlphaBlendingValueGet(SYS_MODULE_INDEX gfxIndex)

GFX_AlphaBlendingValueSet Function

This function sets the alpha value for alpha blending rendering.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_AlphaBlendingValueSet(SYS_MODULE_INDEX gfxIndex, uint16_t alpha);
```

Returns

Status of the alpha value set action. Return GFX_STATUS_ERROR when the alpha value set is unsupported.

Description

This function sets the alpha value for alpha blending rendering. Accepted values are dependent on the used alpha blending routines at build time. When using the software routines in the Primitive Layer, accepted values are 0, 25, 50, 75 and 100.

If using a specific implementation in the display driver used, implementation may support the full range (0-100). When this is the case, refer to the driver alpha blending solution for details.

Function operation will ignore unsupported values of alpha.

Preconditions

None.

Example

```
// render a bar with no alpha blending
GFX_AlphaBlendingValueSet(GFX_INDEX_0, 100);
while(GFX_RectangleFillDraw(GFX_INDEX_0, 10,10,30,30) != GFX_STATUS_SUCCESS);

// render a bar with 50% alpha blending to a
// background with RED color
GFX_BackgroundSet(GFX_INDEX_0, x, y, pBgImage, RED);
GFX_BackgroundTypeSet(GFX_INDEX_0, GFX_BACKGROUND_COLOR);

GFX_AlphaBlendingValueSet(GFX_INDEX_0, 50);
while(GFX_RectangleFillDraw(GFX_INDEX_0, 10,40,30,60) != GFX_STATUS_SUCCESS);
```

Parameters

Parameters	Description
alpha	Defines the alpha blending percentage to be used for alpha blending routines. Accepted values are dependent on the alpha blending routines used. For Primitive Layer
implementation accepted values are	• 100 : no alpha blending, last color set by GFX_ColorSet() will replace the pixels. • 75 : 75% of the last color set by GFX_ColorSet() will be alpha blended to the existing pixel. • 50 : 50% of the last color set by GFX_ColorSet() will be alpha blended to the existing pixel. • 25 : 25% of the last color set by GFX_ColorSet() will be alpha blended to the existing pixel. • 0 : 0% of the last color set by GFX_ColorSet() will be alpha blended to the existing pixel. This means the existing pixel color will not change.

Function

GFX_STATUS GFX_AlphaBlendingValueSet(
SYS_MODULE_INDEX gfxIndex, uint16_t alpha)

GFX_GradientColorSet Function

Sets the gradient fill start and end color.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_GradientColorSet(SYS_MODULE_INDEX gfxIndex, GFX_COLOR startColor, GFX_COLOR endColor);
```

Returns

Status of the color set. GFX_STATUS_SUCCESS - gradient color set done. GFX_STATUS_FAILURE - gradient color set failed.

Description

This function sets the gradient fill start and end colors (see GFX_FILL_STYLE) when rendering gradient filled shapes. All subsequent calls to the following using gradient fills:

- GFX_RectangleFillDraw()
- GFX_RectangleRoundFillDraw()
- GFX_CircleFillDraw()

will use the start and end colors set by this function.

Preconditions

None.

Example

See GFX_FillStyleSet().

Parameters

Parameters	Description
startColor	Gradient start color to be used.
endColor	Gradient end color to be used.

Function

GFX_STATUS GFX_GradientColorSet(
SYS_MODULE_INDEX gfxIndex,
GFX_COLOR startColor,
GFX_COLOR endColor)

GFX_GradientEndColorGet Function

Return the gradient end color when rendering gradient filled shapes.

File

gfx_primitive.h

C

```
GFX_COLOR GFX_GradientEndColorGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The current gradient end color used.

Description

This function returns the end color used when rendering gradient filled shapes.

Preconditions

None.

Example

None.

Function

GFX_COLOR GFX_GradientEndColorGet(void)

GFX_GradientStartColorGet Function

Return the gradient start color when rendering gradient filled shapes.

File

gfx_primitive.h

C

```
GFX_COLOR GFX_GradientStartColorGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The current gradient start color used.

Description

This function returns the start color used when rendering gradient filled shapes.

Preconditions

None.

Example

None.

Function

GFX_COLOR GFX_GradientStartColorGet(SYS_MODULE_INDEX gfxIndex)

GFX_LineStyleGet Function

Return the line style used when rendering lines.

File

gfx_primitive.h

C

```
GFX_LINE_STYLE GFX_LineStyleGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The current line style used (see GFX_LINE_STYLE).

Description

This function returns the line style used (see GFX_LINE_STYLE) when rendering lines.

Preconditions

None.

Example

None.

Function

GFX_LINE_STYLE GFX_LineStyleGet(SYS_MODULE_INDEX gfxIndex)

GFX_LineStyleSet Function

Set the line style to be used when rendering lines.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_LineStyleSet(SYS_MODULE_INDEX gfxIndex, GFX_LINE_STYLE style);
```

Returns

Status of the line style set. GFX_STATUS_SUCCESS - line setting done. GFX_STATUS_FAILURE - line setting not done.

Description

This function sets the line style to be used (see GFX_LINE_STYLE) when rendering lines. All calls to the following functions

- GFX_LineDraw()
- GFX_LineToDraw()
- GFX_LineToRelativeDraw()
- GFX_CircleDraw()
- GFX_RectangleDraw()
- GFX_PolygonDraw()

will use the line style set by this function. In addition, all unfilled shapes that does specify the line style to be used will use the line style specified by this function.

Preconditions

None.

Example

```
GFX_ColorSet(GFX_INDEX_0, BRIGHTRED);
GFX_LineStyleSet(GFX_INDEX_0, GFX_LINE_STYLE_THIN_SOLID);
GFX_LineDraw(GFX_INDEX_0, 10, 10,100,  10);
```

Parameters

Parameters	Description
style	The specified line style to be used.

Function

GFX_STATUS GFX_LineStyleSet(
SYS_MODULE_INDEX gfxIndex,
GFX_LINE_STYLE style)

Color Functions

Functions

	Name	Description
	GFX_ColorGet	This function returns the color currently set to render primitive shapes and text.
	GFX_ColorSet	This function sets the color to be used in rendering primitive shapes and text.
	GFX_TransparentColorDisable	This function disables the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.
	GFX_TransparentColorEnable	This function sets the transparent color used in GFX_ImageDraw() functions and enables the transparent color feature.
	GFX_TransparentColorGet	This returns the current transparent color set for the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.
	GFX_TransparentColorStatusGet	This returns the current state of the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.

Description

Graphics Library Primitive Layer color API.

GFX_ColorGet Function

This function returns the color currently set to render primitive shapes and text.

File

gfx_primitive.h

C

```
GFX_COLOR GFX_ColorGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The currently set color.

Description

This function returns the color currently set to render primitive shapes and text (See GFX_ColorSet() for more information).

Preconditions

None.

Example

None.

Function

void GFX_ColorGet(SYS_MODULE_INDEX gfxIndex)

GFX_ColorSet Function

This function sets the color to be used in rendering primitive shapes and text.

File

gfx_primitive.h

C

```
void GFX_ColorSet(SYS_MODULE_INDEX gfxIndex, GFX_COLOR newColor);
```

Returns

None.

Description

This function sets the color to be used to render primitive shapes and text. Any primitive shape that is set to any of the following line style and fill style will be rendering using the set color.

GFX_LINE_STYLE:

- GFX_LINE_STYLE_THIN_SOLID
- GFX_LINE_STYLE_THIN_DOTTED
- GFX_LINE_STYLE_THIN_DASHED
- GFX_LINE_STYLE_THICK_SOLID
- GFX_LINE_STYLE_THICK_DOTTED
- GFX_LINE_STYLE_THICK_DASHED

GFX_FILL_STYLE:

- GFX_FILL_STYLE_COLOR

Rendering text using the following text rendering functions will also use the set color:

- GFX_TextCharDraw()
- GFX_TextStringDraw()
- GFX_TextStringBoxDraw()

Preconditions

None.

Example

None.

Parameters

Parameters	Description
color	the rendering color set to render primitive shapes.

Function

void GFX_ColorSet(SYS_MODULE_INDEX gfxIndex, GFX_COLOR color)

GFX_TransparentColorDisable Function

This function disables the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_TransparentColorDisable(SYS_MODULE_INDEX gfxIndex);
```

Returns

The status of the transparent color disable action.

Description

This function disables the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.

The transparent color feature can only be enabled when the color depth used is 24 or 16 bpp.

Preconditions

Transparent color feature must be enabled at build time. This is enabled by default and can be disabled by defining the macro GFX_CONFIG_TRANSPARENT_COLOR_DISABLE in the system.

Example

```
// assume ScreenBackground and RibbonIcon are valid
// image resources
// assume BLACK is a valid GFX_COLOR value

GFX_TransparentColorEnable(GFX_INDEX_0, BLACK);
GFX_ImageDraw(GFX_INDEX_0, 0,0, (void*)&ScreenBackground);
GFX_ImageDraw(GFX_INDEX_0, 50,50, (void*)&RibbonIcon);

// disable the transparent color feature since the
// next image to render contains black pixels that
// we want to render
GFX_TransparentColorDisable(GFX_INDEX_0);
GFX_ImageDraw(GFX_INDEX_0, 50,50, (void*)&OverlayImage);
```

Function

GFX_STATUS GFX_TransparentColorDisable(SYS_MODULE_INDEX gfxIndex)

GFX_TransparentColorEnable Function

This function sets the transparent color used in GFX_ImageDraw() functions and enables the transparent color feature.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_TransparentColorEnable(SYS_MODULE_INDEX gfxIndex, GFX_COLOR color);
```

Returns

The status of the transparent color set action.

Description

This function sets the transparent color used in GFX_ImageDraw() functions and enables the transparent color feature.

When GFX_ImageDraw() or GFX_ImagePartialDraw() is called, any pixels in the image that matches the color value will not be rendered to the frame buffer.

The transparent color feature can only be enabled when the color depth used is 24 or 16 bpp.

Preconditions

Transparent color feature must be enabled at build time. This is enabled by default and can be disabled by defining the macro GFX_CONFIG_TRANSPARENT_COLOR_DISABLE in the system.

Example

```
// assume ScreenBackground and RibbonIcon are valid
// image resources
// assume BLACK is a valid GFX_COLOR value

GFX_TransparentColorEnable(GFX_INDEX_0, BLACK);
GFX_ImageDraw(GFX_INDEX_0, 0,0, (void*)&ScreenBackground);
GFX_ImageDraw(GFX_INDEX_0, 0,0, (void*)&RibbonIcon);
```

Parameters

Parameters	Description
color	the color value selected as the transparent color.

Function

GFX_STATUS GFX_TransparentColorEnable(
SYS_MODULE_INDEX gfxIndex,
GFX_COLOR color)

GFX_TransparentColorGet Function

This returns the current transparent color set for the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.

File

gfx_primitive.h

C

```
GFX_COLOR GFX_TransparentColorGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The current transparent color used.

Description

This returns the current transparent color set for the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.

The transparent color feature can only be enabled when the color depth used is 24 or 16 bpp.

Preconditions

Transparent color feature must be enabled at build time. This is enabled by default and can be disabled by defining the macro GFX_CONFIG_TRANSPARENT_COLOR_DISABLE in the system.

Example

```
// assume ScreenBackground and RibbonIcon are valid
// image resources
// assume BLACK is a valid GFX_COLOR value

GFX_TransparentColorEnable(GFX_INDEX_0, BLACK);
GFX_ImageDraw(GFX_INDEX_0, 0,0, (void*)&ScreenBackground);
GFX_ImageDraw(GFX_INDEX_0, 50,50, (void*)&RibbonIcon);

// disable the transparent color feature since the
// next image to render contains black pixels that
// we want to render
if (GFX_TransparentColorGet(GFX_INDEX_0) == BLACK)
    GFX_TransparentColorDisable(GFX_INDEX_0);
GFX_ImageDraw(GFX_INDEX_0, 50,50, (void*)&OverlayImage);
```

Function

GFX_COLOR GFX_TransparentColorGet(SYS_MODULE_INDEX gfxIndex)

GFX_TransparentColorStatusGet Function

This returns the current state of the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.

File

gfx_primitive.h

C

```
GFX_FEATURE_STATUS GFX_TransparentColorStatusGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The current transparent feature status (see GFX_FEATURE_STATUS for details).

Description

This returns the current transparent color set for the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.

The transparent color feature can only be enabled when the color depth used is 24 or 16 bpp.

Preconditions

Transparent color feature must be enabled at build time. This is enabled by default and can be disabled by defining the macro GFX_CONFIG_TRANSPARENT_COLOR_DISABLE in the system.

Example

```
// assume ScreenBackground and RibbonIcon are valid
// image resources
// assume BLACK is a valid GFX_COLOR value
```

```
    if (GFX_TransparentColorStatusGet(GFX_INDEX_0) == GFX_FEATURE_DISABLED)
        GFX_TransparentColorEnable(GFX_INDEX_0, BLACK);
    GFX_ImageDraw(GFX_INDEX_0, 0,0, (void*)&ScreenBackground);
    GFX_ImageDraw(GFX_INDEX_0, 50,50, (void*)&RibbonIcon);

    // disable the transparent color feature since the
    // next image to render contains black pixels that
    // we want to render
    if (GFX_TransparentColorGet(GFX_INDEX_0) == BLACK)
        GFX_TransparentColorDisable(GFX_INDEX_0);
    GFX_ImageDraw(GFX_INDEX_0, 50,50, (void*)&OverlayImage);
```

Function

GFX_FEATURE_STATUS GFX_TransparentColorStatusGet(SYS_MODULE_INDEX gfxIndex)

Background Functions

Functions

	Name	Description
	GFX_BackgroundColorGet	This function returns the color used in the current background.
	GFX_BackgroundImageGet	This function returns the image used in the current background.
	GFX_BackgroundImageLeftGet	This function returns the horizontal starting position of the current background.
	GFX_BackgroundImageTopGet	This function returns the vertical starting position of the current background.
	GFX_BackgroundSet	This function sets the background information.
	GFX_BackgroundTypeGet	This function returns the type of the current background.
	GFX_BackgroundTypeSet	This function sets the background type.

Description

Graphics Library Primitive Layer background API.

GFX_BackgroundColorGet Function

This function returns the color used in the current background.

File

gfx_primitive.h

C

```
GFX_COLOR GFX_BackgroundColorGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The color used in the current background.

Description

This function returns the color used in the current background.

Preconditions

None.

Example

None.

Function

GFX_COLOR GFX_BackgroundColorGet(SYS_MODULE_INDEX gfxIndex)

GFX_BackgroundImageGet Function

This function returns the image used in the current background.

File

gfx_primitive.h

C

```
GFX_RESOURCE_HDR * GFX_BackgroundImageGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The pointer to the image used in the current background.

Description

This function returns the pointer to the image used in the current background.

Preconditions

None.

Example

None.

Function

GFX_RESOURCE_HDR *GFX_BackgroundImageGet(SYS_MODULE_INDEX gfxIndex)

GFX_BackgroundImageLeftGet Function

This function returns the horizontal starting position of the current background.

File

gfx_primitive.h

C

```
uint16_t GFX_BackgroundImageLeftGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The horizontal starting position of the current background.

Description

This function returns the horizontal starting position of the current background. This position defines the x position of the upper left corner of the background.

Preconditions

None.

Example

None.

Function

uint16_t GFX_BackgroundImageLeftGet(SYS_MODULE_INDEX gfxIndex)

GFX_BackgroundImageTopGet Function

This function returns the vertical starting position of the current background.

File

gfx_primitive.h

C

```
uint16_t GFX_BackgroundImageTopGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The vertical starting position of the current background.

Description

This function returns the vertical starting position of the current background. This position defines the y position of the upper left corner of the background.

Preconditions

None.

Example

None.

Function

uint16_t GFX_BackgroundImageTopGet(SYS_MODULE_INDEX gfxIndex)

GFX_BackgroundSet Function

This function sets the background information.

File

gfx_primitive.h

C

```
void GFX_BackgroundSet(SYS_MODULE_INDEX gfxIndex, uint16_t left, uint16_t top, GFX_RESOURCE_HDR * pImage,
GFX_COLOR color);
```

Returns

None.

Description

This function sets the background information. Note that the width and height of the background is only needed when the background is an image. When the background is a color (image pointer is NULL), the width and height is not needed here since the purpose of the background information is to record only the color used.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
left	Horizontal starting position of the background.
top	Vertical starting position of the background.
pImage	Pointer to the background image used. Set this to NULL if not used.
color	If the background image is NULL, background is assumed to be this color.

Function

void GFX_BackgroundSet(
SYS_MODULE_INDEX gfxIndex,
uint16_t left,
uint16_t top,
GFX_RESOURCE_HDR *pImage,
GFX_COLOR color);

GFX_BackgroundTypeGet Function

This function returns the type of the current background.

File

gfx_primitive.h

C

```
GFX_BACKGROUND_TYPE GFX_BackgroundTypeGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The type of the current background set.

Description

This function returns the type of the current background.

Preconditions

None.

Example

None.

Function

GFX_BACKGROUND_TYPE GFX_BackgroundTypeGet(SYS_MODULE_INDEX gfxIndex)

GFX_BackgroundTypeSet Function

This function sets the background type.

File

gfx_primitive.h

C

```
void GFX_BackgroundTypeSet(SYS_MODULE_INDEX gfxIndex, GFX_BACKGROUND_TYPE type);
```

Returns

None.

Description

This function sets the background type.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
type	Type of background that will be used.

Function

void GFX_BackgroundTypeSet(SYS_MODULE_INDEX gfxIndex, GFX_BACKGROUND_TYPE type)

Double Buffering Functions

Functions

	Name	Description
	GFX_DoubleBufferAreaMark	This function adds the given rectangular area as an area that will be included in the list of areas for synchronization.
	GFX_DoubleBufferEnable	This function enables the double buffering feature of the graphics library.
	GFX_DoubleBufferDisable	This function disables the double buffering feature of the graphics library.
	GFX_DoubleBufferStatusGet	This function returns the current status of the double buffering feature of the graphics library.
	GFX_DoubleBufferAreaGet	This function returns a rectangular area that needs synchronization specified by the given index.
	GFX_DoubleBufferSyncAllStatusClear	This function clears the synchronize all status.
	GFX_DoubleBufferSyncAllStatusGet	This function returns the status of the synchronize all flag.
	GFX_DoubleBufferSyncAllStatusSet	This function sets the whole draw buffer to be unsynchronized.
	GFX_DoubleBufferSyncAreaCountGet	This function returns the current count of rectangular areas that needs to be synchronized.
	GFX_DoubleBufferSyncAreaCountSet	This function sets the current count of rectangular areas that needs to be synchronized.
	GFX_DoubleBufferSynchronizeRequest	This function requests synchronization of the contents of the draw and frame buffer.
	GFX_DoubleBufferSynchronize	This function synchronizes the contents of the draw and frame buffer immediately.
	GFX_DoubleBufferSynchronizeStatusGet	This function returns the status of the synchronization request of the draw and frame buffer.

	GFX_DrawBufferGet	This function returns the index of the current draw buffer.
	GFX_DrawBufferInitialize	This function initializes the address of the draw buffer specified by the given index.
	GFX_DrawBufferSet	This function sets the draw buffer with the given index number.
	GFX_FrameBufferGet	This function returns the index of the current frame buffer.
	GFX_FrameBufferSet	This function sets the frame buffer with the given index number.
	GFX_DoubleBufferPause	This function pauses the double buffering feature of the graphics library.
	GFX_DoubleBufferResume	This function resumes the double buffering feature of the graphics library.

Description

Graphics Library Primitive Layer double buffering API.

GFX_DoubleBufferAreaMark Function

This function adds the given rectangular area as an area that will be included in the list of areas for synchronization.

File

gfx_primitive.h

C

```
void GFX_DoubleBufferAreaMark(SYS_MODULE_INDEX gfxIndex, uint16_t left, uint16_t top, uint16_t right,
uint16_t bottom);
```

Returns

None.

Description

When double buffering mode is enabled, this function adds the given rectangular area into the list of rectangular areas for synchronization. When this function is called, the given rectangular area is assumed to contain new pixel information and is added into the list of areas to be synchronized.

Synchronization can be scheduled using the GFX_DoubleBufferSynchronizeRequest() or immediately performed using GFX_DoubleBufferSynchronize().

Preconditions

Double buffering feature must be enabled.

Example

None.

Parameters

Parameters	Description
left	defines the left most pixel of the rectangular area.
top	defines the top most pixel of the rectangular area.
right	defines the right most pixel of the rectangular area.
bottom	defines the bottom most pixel of the rectangular area.

Function

void GFX_DoubleBufferAreaMark(

SYS_MODULE_INDEX gfxIndex,

uint16_t left,

uint16_t top,

uint16_t right,

uint16_t bottom)

GFX_DoubleBufferEnable Function

This function enables the double buffering feature of the graphics library.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_DoubleBufferEnable(SYS_MODULE_INDEX gfxIndex);
```

Returns

Status of the double buffering feature enabling. GFX_STATUS_SUCCESS - the double buffering was successfully enabled.
GFX_STATUS_FAILURE - the double buffering was not successfully enabled.

Description

Double buffering is a feature where two buffers are utilized to perform rendering on one buffer while displaying the other buffer. The frame buffer is the buffer that is being displayed while the draw buffer is used for rendering.

When this function is called, the buffer with index number 0 is set as the draw buffer while the buffer with index number 1 is set as the frame buffer. Synchronization of the two buffers is scheduled and sets the count of unsynchronized areas to zero.

Preconditions

None.

Example

None.

Function

GFX_STATUS GFX_DoubleBufferEnable(SYS_MODULE_INDEX gfxIndex)

GFX_DoubleBufferDisable Function

This function disables the double buffering feature of the graphics library.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_DoubleBufferDisable(SYS_MODULE_INDEX gfxIndex);
```

Returns

Status of the double buffering feature disabling. GFX_STATUS_SUCCESS - the double buffering was successfully disabled.
GFX_STATUS_FAILURE - the double buffering was not successfully disabled.

Description

When double buffering is enabled, calling this function disables the double buffering feature of the graphics library.

After this function executes, all rendering will be performed on the frame buffer.

Preconditions

None.

Example

None.

Function

GFX_STATUS GFX_DoubleBufferDisable(SYS_MODULE_INDEX gfxIndex)

GFX_DoubleBufferStatusGet Function

This function returns the current status of the double buffering feature of the graphics library.

File

gfx_primitive.h

C

```
GFX_FEATURE_STATUS GFX_DoubleBufferStatusGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

Status of the double buffering feature disabling. GFX_FEATURE_ENABLED - the double buffering feature is enabled.
GFX_FEATURE_DISABLED - the double buffering feature is not enabled.

Description

If the graphics library double buffering feature is enabled, this function returns if the double buffering mode is activated or not. When activated, the function returns GFX_FEATURE_ENABLED. When deactivated, the function returns GFX_FEATURE_DISABLED.

Preconditions

None.

Example

None.

Function

GFX_FEATURE_STATUS GFX_DoubleBufferStatusGet(SYS_MODULE_INDEX gfxIndex)

GFX_DoubleBufferAreaGet Function

This function returns a rectangular area that needs synchronization specified by the given index.

File

gfx_primitive.h

C

```
GFX_RECTANGULAR_AREA * GFX_DoubleBufferAreaGet(SYS_MODULE_INDEX gfxIndex, uint16_t index);
```

Returns

The location of the rectangular area specified by the structure GFX_RECTANGULAR_AREA.

Description

Double buffering mode maintains an array of these GFX_RECTANGULAR_AREA that needs synchronization. The array serves as a list of pixel areas that needs to be synchronized.

This function returns a rectangular area that needs synchronization specified by the given index. The returned value is a pointer to the structure GFX_RECTANGULAR_AREA that describes the rectangular area of interest.

Preconditions

Double buffering feature must be enabled.

Example

None.

Parameters

Parameters	Description
index	the index of the rectangular area located in the array of areas that needs synchronization.

Function

GFX_RECTANGULAR_AREA GFX_DoubleBufferAreaGet(SYS_MODULE_INDEX gfxIndex, uint16_t index)

GFX_DoubleBufferSyncAllStatusClear Function

This function clears the synchronize all status.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_DoubleBufferSyncAllStatusClear(SYS_MODULE_INDEX gfxIndex);
```

Returns

Status of the synchronize all clear. GFX_STATUS_SUCCESS - the synchronization all clear was successful. GFX_STATUS_FAILURE - the synchronization all clear was not successful.

Description

This function clears the synchronize all status. This function will clear or cancel all synchronization requests. Calling this function while an ongoing synchronization is being performed, will not terminate the synchronization.

Preconditions

Double buffering feature must be enabled.

Example

None.

Function

GFX_STATUS GFX_DoubleBufferSyncAllStatusClear(SYS_MODULE_INDEX gfxIndex)

GFX_DoubleBufferSyncAllStatusGet Function

This function returns the status of the synchronize all flag.

File

gfx_primitive.h

C

```
GFX_FEATURE_STATUS GFX_DoubleBufferSyncAllStatusGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

Status of the synchronize all flag. GFX_FEATURE_ENABLED - the synchronization all is set. GFX_FEATURE_DISABLED - the synchronization all is not set.

Description

This function returns the status of the synchronize all flag.

Preconditions

Double buffering feature must be enabled.

Example

None.

Function

GFX_FEATURE_STATUS GFX_DoubleBufferSyncAllStatusGet(SYS_MODULE_INDEX gfxIndex)

GFX_DoubleBufferSyncAllStatusSet Function

This function sets the whole draw buffer to be unsynchronized.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_DoubleBufferSyncAllStatusSet(SYS_MODULE_INDEX gfxIndex);
```

Returns

Status of the synchronize all set. GFX_STATUS_SUCCESS - the synchronization all was successful. GFX_STATUS_FAILURE - the synchronization all was not successful.

Description

This function sets the whole draw buffer to be unsynchronized. The next synchronization will copy all pixels of the draw buffer to the frame buffer.

Preconditions

Double buffering feature must be enabled.

Example

None.

Function

GFX_STATUS GFX_DoubleBufferSyncAllStatusSet(SYS_MODULE_INDEX gfxIndex)

GFX_DoubleBufferSyncAreaCountGet Function

This function returns the current count of rectangular areas that needs to be synchronized.

File

gfx_primitive.h

C

```
uint16_t GFX_DoubleBufferSyncAreaCountGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The current count of rectangular areas that needs synchronization.

Description

This function returns the current count of rectangular areas that needs to be synchronized.

Preconditions

Double buffering feature must be enabled.

Example

None.

Function

uint16_t GFX_DoubleBufferSyncAreaCountGet(SYS_MODULE_INDEX gfxIndex)

GFX_DoubleBufferSyncAreaCountSet Function

This function sets the current count of rectangular areas that needs to be synchronized.

File

gfx_primitive.h

C

```
void GFX_DoubleBufferSyncAreaCountSet(SYS_MODULE_INDEX gfxIndex, uint16_t count);
```

Returns

None.

Description

This function sets the current count of rectangular areas that needs to be synchronized.

Preconditions

Double buffering feature must be enabled.

Example

None.

Function

void GFX_DoubleBufferSyncAreaCountSet(SYS_MODULE_INDEX gfxIndex, uint16_t count)

GFX_DoubleBufferSynchronizeRequest Function

This function requests synchronization of the contents of the draw and frame buffer.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_DoubleBufferSynchronizeRequest(SYS_MODULE_INDEX gfxIndex);
```

Returns

Status of the double buffer synchronization request. GFX_STATUS_SUCCESS - the synchronization request was successful. GFX_STATUS_FAILURE - the synchronization request was not successful.

Description

This function requests synchronization of the contents of the draw and frame buffer. The contents will be synchronized on the next vertical blanking period.

Preconditions

Double buffering feature must be enabled.

Example

None.

Function

GFX_STATUS GFX_DoubleBufferSynchronizeRequest(SYS_MODULE_INDEX gfxIndex)

GFX_DoubleBufferSynchronize Function

This function synchronizes the contents of the draw and frame buffer immediately.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_DoubleBufferSynchronize(SYS_MODULE_INDEX gfxIndex);
```

Returns

Status of the double buffer synchronization. GFX_STATUS_SUCCESS - the synchronization was successful. GFX_STATUS_FAILURE - the synchronization was not successful.

Description

This function synchronizes the contents of the draw and frame buffer immediately. Contents of both frame and draw buffer will be the same after the function exits.

Preconditions

Double buffering feature must be enabled.

Example

None.

Function

GFX_STATUS GFX_DoubleBufferSynchronize(SYS_MODULE_INDEX gfxIndex)

GFX_DoubleBufferSynchronizeStatusGet Function

This function returns the status of the synchronization request of the draw and frame buffer.

File

gfx_primitive.h

C

```
GFX_FEATURE_STATUS GFX_DoubleBufferSynchronizeStatusGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

Status of the double buffer synchronization request. GFX_FEATURE_ENABLED - the synchronization request is enabled. GFX_FEATURE_DISABLED - the synchronization request is not enabled.

Description

This function returns the status of the synchronization request of the draw and frame buffer. The difference between GFX_DoubleBufferSynchronizeStatusGet() and GFX_DoubleBufferSyncAllStatusGet() is that the GFX_DoubleBufferSynchronizeStatusGet() returns the status of synchronization request. The size of the synchronization may or may not be the full screen synchronization. The GFX_DoubleBufferSyncAllStatusGet() on the other hand returns the status of a full screen synchronization.

Preconditions

Double buffering feature must be enabled.

Example

None.

Function

GFX_FEATURE_STATUS GFX_DoubleBufferSynchronizeStatusGet(SYS_MODULE_INDEX gfxIndex)

GFX_DrawBufferGet Function

This function returns the index of the current draw buffer.

File

gfx_primitive.h

C

```
uint16_t GFX_DrawBufferGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The index of the current draw buffer set.

Description

This function returns the index of the current draw buffer. Draw buffer is the buffer where rendering is performed. For systems with single buffer this function will always return 0.

Preconditions

None.

Example

None.

Function

uint16_t GFX_DrawBufferGet(SYS_MODULE_INDEX gfxIndex)

GFX_DrawBufferInitialize Function

This function initializes the address of the draw buffer specified by the given index.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_DrawBufferInitialize(SYS_MODULE_INDEX gfxIndex, uint16_t index, uint32_t address);
```

Returns

Status of the buffer set. GFX_STATUS_SUCCESS - the buffer was successfully set. GFX_STATUS_FAILURE - the buffer was not successfully set.

Description

**********f**********

For system with multiple buffers, this function is used to initialize the array of buffers. The address of the draw buffer will be associated with the specified index. Use this function to initialize or modify the array of frame buffers in the system at run time.

For systems with single buffer, frame buffer and draw buffer are the same buffer. Calls to this function will have no effect and will always return GFX_STATUS_SUCCESS. The size of the buffer is defined by the dimension of the screen and the color depth used.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
index	specifies the index value of the buffer in the array of buffers. For single buffer systems, the given index will be ignored.
address	specifies the location of the buffer in memory.

Function

GFX_STATUS GFX_DrawBufferInitialize(
SYS_MODULE_INDEX gfxIndex,
uint16_t index,
uint32_t address)

GFX_DrawBufferSet Function

This function sets the draw buffer with the given index number.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_DrawBufferSet(SYS_MODULE_INDEX gfxIndex, uint16_t index);
```

Returns

Status of the draw buffer set. GFX_STATUS_SUCCESS - the draw buffer was successfully set. GFX_STATUS_FAILURE - the draw buffer was not successfully set.

Description

This function sets the draw buffer with the given index number. For systems with single buffer, frame buffer and draw buffer are the same buffer. Calls to this function will have no effect and will always return GFX_STATUS_SUCCESS.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
index	the index of the buffer to be set as the draw buffer.

Function

GFX_STATUS GFX_DrawBufferSet(SYS_MODULE_INDEX gfxIndex, uint16_t index)

GFX_FrameBufferGet Function

This function returns the index of the current frame buffer.

File

gfx_primitive.h

C

```
uint16_t GFX_FrameBufferGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The index of the current frame buffer set.

Description

This function returns the index of the current frame buffer. Frame buffer is the buffer that is currently displayed in the screen. For systems with single buffer this function will always return 0.

Preconditions

None.

Example

None.

Function

uint16_t GFX_FrameBufferGet(void)

GFX_FrameBufferSet Function

This function sets the frame buffer with the given index number.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_FrameBufferSet(SYS_MODULE_INDEX gfxIndex, uint16_t index);
```

Returns

Status of the draw buffer set. GFX_STATUS_SUCCESS - the draw buffer was successfully set. GFX_STATUS_FAILURE - the draw buffer was not successfully set.

Description

This function sets the frame buffer with the given index number. This is the buffer that is displayed on the screen. For systems with single buffer, frame buffer and draw buffer are the same buffer. Calls to this function will have no effect and will always return GFX_STATUS_SUCCESS.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
index	the index of the buffer to be set as the frame buffer.

Function

GFX_STATUS GFX_FrameBufferSet(SYS_MODULE_INDEX gfxIndex, uint16_t index)

GFX_DoubleBufferPause Function

This function pauses the double buffering feature of the graphics library.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_DoubleBufferPause(SYS_MODULE_INDEX gfxIndex);
```

Description

Double buffering is a feature where two buffers are utilized to perform rendering on one buffer while displaying the other buffer. The frame buffer is the buffer that is being displayed while the draw buffer is used for rendering.

When this function is called, the feature is paused and no buffer swap will occur.

Function

GFX_STATUS GFX_DoubleBufferPause(void)

GFX_DoubleBufferResume Function

This function resumes the double buffering feature of the graphics library.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_DoubleBufferResume(SYS_MODULE_INDEX gfxIndex);
```

Description

Double buffering is a feature where two buffers are utilized to perform rendering on one buffer while displaying the other buffer. The frame buffer is the buffer that is being displayed while the draw buffer is used for rendering.

When this function is called, the feature is resumes the previous status of the double buffering feature.

Function

GFX_STATUS GFX_DoubleBufferResume(void)

External Resources Functions

Functions

	Name	Description
	GFX_ExternalResourceCallback	This function performs data fetch from external memory.

Description

Graphics Library Primitive Layer external resources API.

GFX_ExternalResourceCallback Function

This function performs data fetch from external memory.

File

gfx_primitive.h

C

```
GFX_STATUS GFX_ExternalResourceCallback(GFX_RESOURCE_HDR * pResource, uint32_t offset, uint16_t nCount,
void * pBuffer);
```

Returns

Status of the external resource callback. GFX_STATUS_SUCCESS when all the data was successfully retrieved. GFX_STATUS_FAILURE when partial or no data was retrieved.

Description

This function must be implemented in the application. The library will call this function each time when the external memory data will be required.

The application must copy requested byte quantity into the buffer provided. Data start address in external memory is a sum of the address in GFX_RESOURCE_HDR structure and offset.

An example of a situation where external memory will be accessed is when external fonts or images are used. External resources in the library are defined by the type in the GFX_RESOURCE_HDR (see GFX_RESOURCE for details).

Preconditions

None.

Example

None.

Parameters

Parameters	Description
pResource	Pointer to the external memory resource information.
offset	Offset of the data from the location of the resource in external memory.
nCount	Number of bytes to be transferred into the buffer.
buffer	Pointer to the buffer that will hold the retrieved data.

Function

GFX_STATUS GFX_ExternalResourceCallback(
GFX_RESOURCE_HDR *pResource,
uint32_t offset,
uint16_t nCount,
void *pBuffer)

Data Types and Constants

Enumerations

	Name	Description
	GFX_PRIMITIVE_STATES	Defines the various states that can be achieved by the PRIMITIVE HAL operation.

Macros

	Name	Description
	GFX_PRIMITIVE_HANDLE_INVALID	Constant that defines the value of an Invalid Device Handle.
	GFX_TEXT_CURSOR_HEIGHT_DEFAULT	Constant that defines the default value of text cursor height.
	GFX_TEXT_CURSOR_WIDTH_DEFAULT	Constant that defines the default value of text cursor width.

Types

	Name	Description
	GFX_PRIMITIVE_HANDLE	Data type for GFX PRIMITIVE HAL handle.

Description

Graphics Library Primitive Layer data types and constants.

GFX_PRIMITIVE_HANDLE_INVALID Macro

Constant that defines the value of an Invalid Device Handle.

File

gfx_primitive.h

C

```
#define GFX_PRIMITIVE_HANDLE_INVALID
```

Description

GFX PRIMITIVE HAL Invalid Handle

This constant is returned by the GFX_PRIMITIVE_Open() function when the function fails.

Remarks

None.

GFX_PRIMITIVE_HANDLE Type

Data type for GFX PRIMITIVE HAL handle.

File

gfx_primitive.h

C

```
typedef uintptr_t GFX_PRIMITIVE_HANDLE;
```

Description

Data type for GFX PRIMITIVE HAL handle.

The data type of the handle that is returned from GFX_PRIMITIVE_Open function.

Remarks

None.

GFX_PRIMITIVE_STATES Enumeration

Defines the various states that can be achieved by the PRIMITIVE HAL operation.

File

gfx_primitive.h

C

```
typedef enum {
  GFX_PRIMITIVE_STATE_BUSY,
  GFX_PRIMITIVE_STATE_INIT
} GFX_PRIMITIVE_STATES;
```

Members

Members	Description
GFX_PRIMITIVE_STATE_BUSY	Driver state busy
GFX_PRIMITIVE_STATE_INIT	Driver state init

Description

PRIMITIVE HAL State Machine States

This enumeration defines the various states that can be achieved by the HAL operation.

Remarks

None.

GFX_TEXT_CURSOR_HEIGHT_DEFAULT Macro

Constant that defines the default value of text cursor height.

File

gfx_primitive.h

C

```
#define GFX_TEXT_CURSOR_HEIGHT_DEFAULT 10
```

Description

GFX PRIMITIVE HAL Default Text cursor height

This constant is utilized when cursor height is not defined.

Remarks

None.

GFX_TEXT_CURSOR_WIDTH_DEFAULT Macro

Constant that defines the default value of text cursor width.

File

gfx_primitive.h

C

```
#define GFX_TEXT_CURSOR_WIDTH_DEFAULT 6
```

Description

GFX PRIMITIVE HAL Default Text cursor width

This constant is utilized when cursor width is not defined.

Remarks

GFX_TextCursorWidthGet returns this value if a valid cursor width is not available.

Color Definitions

Macros

	Name	Description
	BLACK	Basic colors definitions. The basic colors defined in this section are defined for basic demo requirements. End application can define additional colors or override existing default colors. If overriding an existing default color, define the new color value before the #include of Graphics.h to avoid the compile time warning. When using palette, a different file 'PaletteColorDefines.h' has to be used instead.
	BLUE	This is macro BLUE.
	BRIGHTBLUE	This is macro BRIGHTBLUE.
	BRIGHTCYAN	This is macro BRIGHTCYAN.
	BRIGHTGREEN	This is macro BRIGHTGREEN.
	BRIGHTMAGENTA	This is macro BRIGHTMAGENTA.
	BRIGHTRED	This is macro BRIGHTRED.
	BRIGHTYELLOW	This is macro BRIGHTYELLOW.
	BROWN	This is macro BROWN.
	BURLYWOOD	This is macro BURLYWOOD.
	CYAN	This is macro CYAN.
	DARKGRAY	This is macro DARKGRAY.
	DARKORANGE	This is macro DARKORANGE.

	GFX_W3_ALICEBLUE	This is macro GFX_W3_ALICEBLUE.
	GFX_W3_ANTIQUEWHITE	This is macro GFX_W3_ANTIQUEWHITE.
	GFX_W3_AQUA	This is macro GFX_W3_AQUA.
	GFX_W3_AQUAMARINE	This is macro GFX_W3_AQUAMARINE.
	GFX_W3_AZURE	This is macro GFX_W3_AZURE.
	GFX_W3_BEIGE	This is macro GFX_W3_BEIGE.
	GFX_W3_BISQUE	This is macro GFX_W3_BISQUE.
	GFX_W3_BLACK	This is macro GFX_W3_BLACK.
	GFX_W3_BLANCHEDALMOND	This is macro GFX_W3_BLANCHEDALMOND.
	GFX_W3_BLUE	This is macro GFX_W3_BLUE.
	GFX_W3_BLUEVIOLET	This is macro GFX_W3_BLUEVIOLET.
	GFX_W3_BROWN	This is macro GFX_W3_BROWN.
	GFX_W3_BURLYWOOD	This is macro GFX_W3_BURLYWOOD.
	GFX_W3_CADETBLUE	This is macro GFX_W3_CADETBLUE.
	GFX_W3_CHARTREUSE	This is macro GFX_W3_CHARTREUSE.
	GFX_W3_CHOCOLATE	This is macro GFX_W3_CHOCOLATE.
	GFX_W3_CORAL	This is macro GFX_W3_CORAL.
	GFX_W3_CORNFLOWERBLUE	This is macro GFX_W3_CORNFLOWERBLUE.
	GFX_W3_CORNSILK	This is macro GFX_W3_CORNSILK.
	GFX_W3_CRIMSON	This is macro GFX_W3_CRIMSON.
	GFX_W3_CYAN	This is macro GFX_W3_CYAN.
	GFX_W3_DARKBLUE	This is macro GFX_W3_DARKBLUE.
	GFX_W3_DARKCYAN	This is macro GFX_W3_DARKCYAN.
	GFX_W3_darkgoldenrod	This is macro GFX_W3_darkgoldenrod.
	GFX_W3_DARKGRAY	This is macro GFX_W3_DARKGRAY.
	GFX_W3_DARKGREEN	This is macro GFX_W3_DARKGREEN.
	GFX_W3_DARKGREY	This is macro GFX_W3_DARKGREY.
	GFX_W3_DARKHAKI	This is macro GFX_W3_DARKHAKI.
	GFX_W3_DARKMAGENTA	This is macro GFX_W3_DARKMAGENTA.
	GFX_W3_DARKOLIVEGREEN	This is macro GFX_W3_DARKOLIVEGREEN.
	GFX_W3_DARKORANGE	This is macro GFX_W3_DARKORANGE.
	GFX_W3_DARKORCHID	This is macro GFX_W3_DARKORCHID.
	GFX_W3_DARKRED	This is macro GFX_W3_DARKRED.
	GFX_W3_DARKSALMON	This is macro GFX_W3_DARKSALMON.
	GFX_W3_DARKSEAGREEN	This is macro GFX_W3_DARKSEAGREEN.
	GFX_W3_DARKSLATEBLUE	This is macro GFX_W3_DARKSLATEBLUE.
	GFX_W3_DARKSLATEGRAY	This is macro GFX_W3_DARKSLATEGRAY.
	GFX_W3_DARKSLATEGREY	This is macro GFX_W3_DARKSLATEGREY.
	GFX_W3_DARKTURQUOISE	This is macro GFX_W3_DARKTURQUOISE.
	GFX_W3_DARKVIOLET	This is macro GFX_W3_DARKVIOLET.
	GFX_W3_DEEPPINK	This is macro GFX_W3_DEEPPINK.
	GFX_W3_DEEPSKYBLUE	This is macro GFX_W3_DEEPSKYBLUE.
	GFX_W3_DIMGRAY	This is macro GFX_W3_DIMGRAY.
	GFX_W3_DIMGREY	This is macro GFX_W3_DIMGREY.
	GFX_W3_DODGERBLUE	This is macro GFX_W3_DODGERBLUE.
	GFX_W3_FIREBRICK	This is macro GFX_W3_FIREBRICK.
	GFX_W3_FLORALWHITE	This is macro GFX_W3_FLORALWHITE.
	GFX_W3_FORESTGREEN	This is macro GFX_W3_FORESTGREEN.
	GFX_W3_FUCHSIA	This is macro GFX_W3_FUCHSIA.
	GFX_W3_GAINSBORO	This is macro GFX_W3_GAINSBORO.
	GFX_W3_GHOSTWHITE	This is macro GFX_W3_GHOSTWHITE.
	GFX_W3_GOLD	This is macro GFX_W3_GOLD.
	GFX_W3_GOLDENROD	This is macro GFX_W3_GOLDENROD.
	GFX_W3_GRAY	This is macro GFX_W3_GRAY.
	GFX_W3_GREEN	This is macro GFX_W3_GREEN.

	GFX_W3_GREENYELLOW	This is macro GFX_W3_GREENYELLOW.
	GFX_W3_GREY	This is macro GFX_W3_GREY.
	GFX_W3_HONEYDEW	This is macro GFX_W3_HONEYDEW.
	GFX_W3_HOTPINK	This is macro GFX_W3_HOTPINK.
	GFX_W3_INDIANRED	This is macro GFX_W3_INDIANRED.
	GFX_W3_INDIGO	This is macro GFX_W3_INDIGO.
	GFX_W3_IVORY	This is macro GFX_W3_IVORY.
	GFX_W3_LAVENDER	This is macro GFX_W3_LAVENDER.
	GFX_W3_LAVENDERBLUSH	This is macro GFX_W3_LAVENDERBLUSH.
	GFX_W3_LAWNGREEN	This is macro GFX_W3_LAWNGREEN.
	GFX_W3_LEMONCHIFFON	This is macro GFX_W3_LEMONCHIFFON.
	GFX_W3_LIGHTBLUE	This is macro GFX_W3_LIGHTBLUE.
	GFX_W3_LIGHTCORAL	This is macro GFX_W3_LIGHTCORAL.
	GFX_W3_LIGHTCYAN	This is macro GFX_W3_LIGHTCYAN.
	GFX_W3_LIGHTGOLDENRODYELLOW	This is macro GFX_W3_LIGHTGOLDENRODYELLOW.
	GFX_W3_LIGHTGRAY	This is macro GFX_W3_LIGHTGRAY.
	GFX_W3_LIGHTGREEN	This is macro GFX_W3_LIGHTGREEN.
	GFX_W3_LIGHTGREY	This is macro GFX_W3_LIGHTGREY.
	GFX_W3_LIGHTPINK	This is macro GFX_W3_LIGHTPINK.
	GFX_W3_LIGHTSALMON	This is macro GFX_W3_LIGHTSALMON.
	GFX_W3_LIGHTSKYBLUE	This is macro GFX_W3_LIGHTSKYBLUE.
	GFX_W3_LIGHTSLATEGRAY	This is macro GFX_W3_LIGHTSLATEGRAY.
	GFX_W3_LIGHTSLATEGREY	This is macro GFX_W3_LIGHTSLATEGREY.
	GFX_W3_LIGHTSTEELBLUE	This is macro GFX_W3_LIGHTSTEELBLUE.
	GFX_W3_LIGHTYELLOW	This is macro GFX_W3_LIGHTYELLOW.
	GFX_W3_LIGTHSEAGREEN	This is macro GFX_W3_LIGTHSEAGREEN.
	GFX_W3_LIME	This is macro GFX_W3_LIME.
	GFX_W3_LIMEGREEN	This is macro GFX_W3_LIMEGREEN.
	GFX_W3_LINEN	This is macro GFX_W3_LINEN.
	GFX_W3_MAGENTA	This is macro GFX_W3_MAGENTA.
	GFX_W3_MAROON	This is macro GFX_W3_MAROON.
	GFX_W3_MEDIUMAQUAMARINE	This is macro GFX_W3_MEDIUMAQUAMARINE.
	GFX_W3_MEDIUMBLUE	This is macro GFX_W3_MEDIUMBLUE.
	GFX_W3_MEDIUMORCHID	This is macro GFX_W3_MEDIUMORCHID.
	GFX_W3_MEDIUMPURPLE	This is macro GFX_W3_MEDIUMPURPLE.
	GFX_W3_MEDIUMSEAGREEN	This is macro GFX_W3_MEDIUMSEAGREEN.
	GFX_W3_MEDIUMSLATEBLUE	This is macro GFX_W3_MEDIUMSLATEBLUE.
	GFX_W3_MEDIUMSPRINGGREEN	This is macro GFX_W3_MEDIUMSPRINGGREEN.
	GFX_W3_MEDIUMTURQUOISE	This is macro GFX_W3_MEDIUMTURQUOISE.
	GFX_W3_MEDIUMVIOLETRED	This is macro GFX_W3_MEDIUMVIOLETRED.
	GFX_W3_MIDNIGHTBLUE	This is macro GFX_W3_MIDNIGHTBLUE.
	GFX_W3_MINTCREAM	This is macro GFX_W3_MINTCREAM.
	GFX_W3_MISTYROSE	This is macro GFX_W3_MISTYROSE.
	GFX_W3_MOCCASIN	This is macro GFX_W3_MOCCASIN.
	GFX_W3_NAVAJOWHITE	This is macro GFX_W3_NAVAJOWHITE.
	GFX_W3_NAVY	This is macro GFX_W3_NAVY.
	GFX_W3_OLDLACE	This is macro GFX_W3_OLDLACE.
	GFX_W3_OLIVE	This is macro GFX_W3_OLIVE.
	GFX_W3_OLIVEDRAB	This is macro GFX_W3_OLIVEDRAB.
	GFX_W3_ORANGE	This is macro GFX_W3_ORANGE.
	GFX_W3_ORANGERED	This is macro GFX_W3_ORANGERED.
	GFX_W3_ORCHID	This is macro GFX_W3_ORCHID.
	GFX_W3_PALEGOLDENROD	This is macro GFX_W3_PALEGOLDENROD.
	GFX_W3_PALEGREEN	This is macro GFX_W3_PALEGREEN.
	GFX_W3_PALETURQUOISE	This is macro GFX_W3_PALETURQUOISE.

	GFX_W3_PALEVIOLETRED	This is macro GFX_W3_PALEVIOLETRED.
	GFX_W3_PAPAYAWHIP	This is macro GFX_W3_PAPAYAWHIP.
	GFX_W3_PEACHPUFF	This is macro GFX_W3_PEACHPUFF.
	GFX_W3_PERU	This is macro GFX_W3_PERU.
	GFX_W3_PINK	This is macro GFX_W3_PINK.
	GFX_W3_PLUM	This is macro GFX_W3_PLUM.
	GFX_W3_POWDERBLUE	This is macro GFX_W3_POWDERBLUE.
	GFX_W3_PURPLE	This is macro GFX_W3_PURPLE.
	GFX_W3_RED	This is macro GFX_W3_RED.
	GFX_W3_ROSYBROWN	This is macro GFX_W3_ROSYBROWN.
	GFX_W3_ROYALBLUE	This is macro GFX_W3_ROYALBLUE.
	GFX_W3_SADDLEBROWN	This is macro GFX_W3_SADDLEBROWN.
	GFX_W3_SALMON	This is macro GFX_W3_SALMON.
	GFX_W3_SANDYGREEN	This is macro GFX_W3_SANDYGREEN.
	GFX_W3_SEAGREEN	This is macro GFX_W3_SEAGREEN.
	GFX_W3_SEASHELL	This is macro GFX_W3_SEASHELL.
	GFX_W3_SIENNA	This is macro GFX_W3_SIENNA.
	GFX_W3_SILVER	This is macro GFX_W3_SILVER.
	GFX_W3_SKYBLUE	This is macro GFX_W3_SKYBLUE.
	GFX_W3_SLATEBLUE	This is macro GFX_W3_SLATEBLUE.
	GFX_W3_SLATEGRAY	This is macro GFX_W3_SLATEGRAY.
	GFX_W3_SLATEGREY	This is macro GFX_W3_SLATEGREY.
	GFX_W3_SNOW	This is macro GFX_W3_SNOW.
	GFX_W3_SPRINGGREEN	This is macro GFX_W3_SPRINGGREEN.
	GFX_W3_STEELBLUE	This is macro GFX_W3_STEELBLUE.
	GFX_W3_TAN	This is macro GFX_W3_TAN.
	GFX_W3_TEAL	This is macro GFX_W3_TEAL.
	GFX_W3_THISTLE	This is macro GFX_W3_THISTLE.
	GFX_W3_TOMATO	This is macro GFX_W3_TOMATO.
	GFX_W3_TURQUOISE	This is macro GFX_W3_TURQUOISE.
	GFX_W3_VIOLET	This is macro GFX_W3_VIOLET.
	GFX_W3_WHEAT	This is macro GFX_W3_WHEAT.
	GFX_W3_WHITE	This is macro GFX_W3_WHITE.
	GFX_W3_WHITESMOKE	This is macro GFX_W3_WHITESMOKE.
	GFX_W3_YELLOW	This is macro GFX_W3_YELLOW.
	GFX_W3_YELLOWGREEN	This is macro GFX_W3_YELLOWGREEN.
	GFX_X11_ALICE_BLUE	This is macro GFX_X11_ALICE_BLUE.
	GFX_X11_ANTIQUE_WHITE	This is macro GFX_X11_ANTIQUE_WHITE.
	GFX_X11_AQUA	This is macro GFX_X11_AQUA.
	GFX_X11_AQUAMARINE	This is macro GFX_X11_AQUAMARINE.
	GFX_X11_AZURE	This is macro GFX_X11_AZURE.
	GFX_X11_BEIGE	This is macro GFX_X11_BEIGE.
	GFX_X11_BISQUE	This is macro GFX_X11_BISQUE.
	GFX_X11_BLACK	This is macro GFX_X11_BLACK.
	GFX_X11_BLANCHED_ALMOND	This is macro GFX_X11_BLANCHED_ALMOND.
	GFX_X11_BLUE	This is macro GFX_X11_BLUE.
	GFX_X11_BLUE_VIOLET	This is macro GFX_X11_BLUE_VIOLET.
	GFX_X11_BROWN	This is macro GFX_X11_BROWN.
	GFX_X11_BURLY_WOOD	This is macro GFX_X11_BURLY_WOOD.
	GFX_X11_CADEL_BLUE	This is macro GFX_X11_CADEL_BLUE.
	GFX_X11_CHARTEUSE	This is macro GFX_X11_CHARTEUSE.
	GFX_X11_CHOCOLATE	This is macro GFX_X11_CHOCOLATE.
	GFX_X11_CORAL	This is macro GFX_X11_CORAL.
	GFX_X11_CORNFLOWER_BLUE	This is macro GFX_X11_CORNFLOWER_BLUE.
	GFX_X11_CORNSILK	X11: Brown Colors

	GFX_X11_CRIMSON	This is macro GFX_X11_CRIMSON.
	GFX_X11_CYAN	This is macro GFX_X11_CYAN.
	GFX_X11_DARK_BLUE	This is macro GFX_X11_DARK_BLUE.
	GFX_X11_DARK_CYAN	This is macro GFX_X11_DARK_CYAN.
	GFX_X11_DARK_GOLDENROD	This is macro GFX_X11_DARK_GOLDENROD.
	GFX_X11_DARK_GREEN	This is macro GFX_X11_DARK_GREEN.
	GFX_X11_DARK_GREY	This is macro GFX_X11_DARK_GREY.
	GFX_X11_DARK_KHAKI	This is macro GFX_X11_DARK_KHAKI.
	GFX_X11_DARK_MAGENTA	This is macro GFX_X11_DARK_MAGENTA.
	GFX_X11_DARK_OLIVE_GREEN	X11: Green Colors
	GFX_X11_DARK_ORANGE	This is macro GFX_X11_DARK_ORANGE.
	GFX_X11_DARK_ORCHID	This is macro GFX_X11_DARK_ORCHID.
	GFX_X11_DARK_RED	This is macro GFX_X11_DARK_RED.
	GFX_X11_DARK_SALMON	This is macro GFX_X11_DARK_SALMON.
	GFX_X11_DARK_SEA_GREEN	This is macro GFX_X11_DARK_SEA_GREEN.
	GFX_X11_DARK_SLATE_BLUE	This is macro GFX_X11_DARK_SLATE_BLUE.
	GFX_X11_DARK_SLATE_GREY	This is macro GFX_X11_DARK_SLATE_GREY.
	GFX_X11_DARK_TURQUOISE	This is macro GFX_X11_DARK_TURQUOISE.
	GFX_X11_DARK_VIOLET	This is macro GFX_X11_DARK_VIOLET.
	GFX_X11_DEEP_PINK	This is macro GFX_X11_DEEP_PINK.
	GFX_X11_DEEP_SKY_BLUE	This is macro GFX_X11_DEEP_SKY_BLUE.
	GFX_X11_DIM_GREY	This is macro GFX_X11_DIM_GREY.
	GFX_X11_DODGER_BLUE	This is macro GFX_X11_DODGER_BLUE.
	GFX_X11_FIRE_BRICK	This is macro GFX_X11_FIRE_BRICK.
	GFX_X11_FLORAL_WHITE	This is macro GFX_X11_FLORAL_WHITE.
	GFX_X11_FOREST_GREEN	This is macro GFX_X11_FOREST_GREEN.
	GFX_X11_FUCHSIA	This is macro GFX_X11_FUCHSIA.
	GFX_X11_GAINSBORO	This is macro GFX_X11_GAINSBORO.
	GFX_X11_GHOST_WHITE	This is macro GFX_X11_GHOST_WHITE.
	GFX_X11_GOLD	This is macro GFX_X11_GOLD.
	GFX_X11_GOLDENROD	This is macro GFX_X11_GOLDENROD.
	GFX_X11_GREEN	This is macro GFX_X11_GREEN.
	GFX_X11_GREEN_YELLOW	This is macro GFX_X11_GREEN_YELLOW.
	GFX_X11_GREY	This is macro GFX_X11_GREY.
	GFX_X11_HONEYDEW	This is macro GFX_X11_HONEYDEW.
	GFX_X11_HOT_PINK	This is macro GFX_X11_HOT_PINK.
	GFX_X11_INDIAN_RED	This is macro GFX_X11_INDIAN_RED.
	GFX_X11_INDIGO	This is macro GFX_X11_INDIGO.
	GFX_X11_IVORY	This is macro GFX_X11_IVORY.
	GFX_X11_KHAKI	This is macro GFX_X11_KHAKI.
	GFX_X11_LAVENDER	X11: BLUE COLORS
	GFX_X11_LAVENDOR_BLUSH	This is macro GFX_X11_LAVENDOR_BLUSH.
	GFX_X11_LAWN_GREEN	This is macro GFX_X11_LAWN_GREEN.
	GFX_X11_LEMON_CHIFFON	This is macro GFX_X11_LEMON_CHIFFON.
	GFX_X11_LIGHT_BLUE	This is macro GFX_X11_LIGHT_BLUE.
	GFX_X11_LIGHT_CAROL	This is macro GFX_X11_LIGHT_CAROL.
	GFX_X11_LIGHT_CYAN	This is macro GFX_X11_LIGHT_CYAN.
	GFX_X11_LIGHT_GOLDENROD_YELLOW	This is macro GFX_X11_LIGHT_GOLDENROD_YELLOW.
	GFX_X11_LIGHT_GRAY	This is macro GFX_X11_LIGHT_GRAY.
	GFX_X11_LIGHT_GREEN	This is macro GFX_X11_LIGHT_GREEN.
	GFX_X11_LIGHT_PINK	This is macro GFX_X11_LIGHT_PINK.
	GFX_X11_LIGHT_SALMON	X11: Red Colors
	GFX_X11_LIGHT_SEA_GREEN	This is macro GFX_X11_LIGHT_SEA_GREEN.
	GFX_X11_LIGHT_SKY_BLUE	This is macro GFX_X11_LIGHT_SKY_BLUE.
	GFX_X11_LIGHT_SLATE_GREY	This is macro GFX_X11_LIGHT_SLATE_GREY.

	GFX_X11_LIGHT_STEEL_BLUE	X11: BLUE COLORS
	GFX_X11_LIGHT_YELLOW	This is macro GFX_X11_LIGHT_YELLOW.
	GFX_X11_LIME	This is macro GFX_X11_LIME.
	GFX_X11_LIME_GREEN	This is macro GFX_X11_LIME_GREEN.
	GFX_X11_LINEN	This is macro GFX_X11_LINEN.
	GFX_X11_MAGENTA	This is macro GFX_X11_MAGENTA.
	GFX_X11_MARRON	This is macro GFX_X11_MARRON.
	GFX_X11_MEDIUM_AQUAMARINE	X11: CYAN COLORS
	GFX_X11_MEDIUM_BLUE	This is macro GFX_X11_MEDIUM_BLUE.
	GFX_X11_MEDIUM_ORCHID	This is macro GFX_X11_MEDIUM_ORCHID.
	GFX_X11_MEDIUM_PURPLE	This is macro GFX_X11_MEDIUM_PURPLE.
	GFX_X11_MEDIUM_SEA_GREEN	This is macro GFX_X11_MEDIUM_SEA_GREEN.
	GFX_X11_MEDIUM_SLATE_BLUE	This is macro GFX_X11_MEDIUM_SLATE_BLUE.
	GFX_X11_MEDIUM_SPRING_GREEN	This is macro GFX_X11_MEDIUM_SPRING_GREEN.
	GFX_X11_MEDIUM_TURQUOISE	This is macro GFX_X11_MEDIUM_TURQUOISE.
	GFX_X11_MEDIUM_VIOLET_RED	This is macro GFX_X11_MEDIUM_VIOLET_RED.
	GFX_X11_MIDNIGHT_BLUE	This is macro GFX_X11_MIDNIGHT_BLUE.
	GFX_X11_MINT_CREAM	This is macro GFX_X11_MINT_CREAM.
	GFX_X11_MISTY_ROSE	This is macro GFX_X11_MISTY_ROSE.
	GFX_X11_MOCCASIN	This is macro GFX_X11_MOCCASIN.
	GFX_X11_NAVAJO_WHITE	This is macro GFX_X11_NAVAJO_WHITE.
	GFX_X11_NAVY	This is macro GFX_X11_NAVY.
	GFX_X11_OLD_LACE	This is macro GFX_X11_OLD_LACE.
	GFX_X11_OLIVE	This is macro GFX_X11_OLIVE.
	GFX_X11_OLIVE_DRAB	This is macro GFX_X11_OLIVE_DRAB.
	GFX_X11_ORANGE	This is macro GFX_X11_ORANGE.
	GFX_X11_ORANGE_RED	X11: Orange Colors
	GFX_X11_ORCHID	This is macro GFX_X11_ORCHID.
	GFX_X11_PALE_GOLDENROD	This is macro GFX_X11_PALE_GOLDENROD.
	GFX_X11_PALE_GREEN	This is macro GFX_X11_PALE_GREEN.
	GFX_X11_PALE_TURQUOISE	This is macro GFX_X11_PALE_TURQUOISE.
	GFX_X11_PALE_VIOLET_RED	This is macro GFX_X11_PALE_VIOLET_RED.
	GFX_X11_PAPAYA_WHIP	This is macro GFX_X11_PAPAYA_WHIP.
	GFX_X11_PEACH_PUFF	This is macro GFX_X11_PEACH_PUFF.
	GFX_X11_PERU	This is macro GFX_X11_PERU.
	GFX_X11_PINK	X11: Pink Colors
	GFX_X11_PLUM	This is macro GFX_X11_PLUM.
	GFX_X11_POWDER_BLUE	This is macro GFX_X11_POWDER_BLUE.
	GFX_X11_PURPLE	This is macro GFX_X11_PURPLE.
	GFX_X11_RED	This is macro GFX_X11_RED.
	GFX_X11_ROSY_BROWN	This is macro GFX_X11_ROSY_BROWN.
	GFX_X11_ROYAL_BLUE	This is macro GFX_X11_ROYAL_BLUE.
	GFX_X11_SADDLE_BROWN	This is macro GFX_X11_SADDLE_BROWN.
	GFX_X11_SALMON	This is macro GFX_X11_SALMON.
	GFX_X11_SANDY_BROWN	This is macro GFX_X11_SANDY_BROWN.
	GFX_X11_SEA_GREEN	This is macro GFX_X11_SEA_GREEN.
	GFX_X11_SEASHELL	This is macro GFX_X11_SEASHELL.
	GFX_X11_SIENNA	This is macro GFX_X11_SIENNA.
	GFX_X11_SILVER	This is macro GFX_X11_SILVER.
	GFX_X11_SKY_BLUE	This is macro GFX_X11_SKY_BLUE.
	GFX_X11_SLATE_BLUE	This is macro GFX_X11_SLATE_BLUE.
	GFX_X11_SLATE_GREY	This is macro GFX_X11_SLATE_GREY.
	GFX_X11_SNOW	This is macro GFX_X11_SNOW.
	GFX_X11_SPRING_GREEN	This is macro GFX_X11_SPRING_GREEN.
	GFX_X11_STEEL_BLUE	This is macro GFX_X11_STEEL_BLUE.

	GFX_X11_TAN	This is macro GFX_X11_TAN.
	GFX_X11_TEAL	This is macro GFX_X11_TEAL.
	GFX_X11_THISTLE	This is macro GFX_X11_THISTLE.
	GFX_X11_TOMATO	This is macro GFX_X11_TOMATO.
	GFX_X11_TURQUOISE	This is macro GFX_X11_TURQUOISE.
	GFX_X11_VIOLET	This is macro GFX_X11_VIOLET.
	GFX_X11_WHEAT	This is macro GFX_X11_WHEAT.
	GFX_X11_WHITE	X11: WHITE/GRAY/BLACK COLORS
	GFX_X11_WHITE_SMOKE	This is macro GFX_X11_WHITE_SMOKE.
	GFX_X11_YELLOW	X11: Yellow Colors
	GFX_X11_YELLOW_GREEN	This is macro GFX_X11_YELLOW_GREEN.
	GOLD	This is macro GOLD.
	GRAY000	This is macro GRAY000.
	GRAY001	This is macro GRAY001.
	GRAY002	This is macro GRAY002.
	GRAY003	This is macro GRAY003.
	GRAY004	This is macro GRAY004.
	GRAY005	This is macro GRAY005.
	GRAY006	This is macro GRAY006.
	GRAY007	This is macro GRAY007.
	GRAY008	This is macro GRAY008.
	GRAY009	This is macro GRAY009.
	GRAY010	This is macro GRAY010.
	GRAY011	This is macro GRAY011.
	GRAY012	This is macro GRAY012.
	GRAY013	This is macro GRAY013.
	GRAY014	This is macro GRAY014.
	GRAY015	This is macro GRAY015.
	GRAY032	This is macro GRAY032.
	GRAY096	This is macro GRAY096.
	GRAY128	This is macro GRAY128.
	GRAY160	This is macro GRAY160.
	GRAY192	This is macro GRAY192.
	GRAY204	This is macro GRAY204.
	GRAY224	This is macro GRAY224.
	GRAY229	This is macro GRAY229.
	GRAY242	This is macro GRAY242.
	GREEN	This is macro GREEN.
	KHAKI	This is macro KHAKI.
	LIGHTBLUE	This is macro LIGHTBLUE.
	LIGHTCYAN	This is macro LIGHTCYAN.
	LIGHTGRAY	This is macro LIGHTGRAY.
	LIGHTGREEN	This is macro LIGHTGREEN.
	LIGHTMAGENTA	This is macro LIGHTMAGENTA.
	LIGHTORANGE	This is macro LIGHTORANGE.
	LIGHTRED	This is macro LIGHTRED.
	LIGHTYELLOW	This is macro LIGHTYELLOW.
	MAGENTA	This is macro MAGENTA.
	ORANGE	This is macro ORANGE.
	PERU	This is macro PERU.
	RED	This is macro RED.
	SADDLEBROWN	This is macro SADDLEBROWN.
	SIENNA	This is macro SIENNA.
	TAN	This is macro TAN.
	WHEAT	This is macro WHEAT.

	WHITE	This is macro WHITE.
	YELLOW	This is macro YELLOW.

Description

BLACK Macro

File

gfx_colors.h

C

```
#define BLACK GFX_RGBConvert(0, 0, 0)
```

Description

Basic colors definitions. The basic colors defined in this section are defined for basic demo requirements. End application can define additional colors or override existing default colors. If overriding an existing default color, define the new color value before the #include of Graphics.h to avoid the compile time warning. When using palette, a different file 'PaletteColorDefines.h' has to be used instead.

BLUE Macro

File

gfx_colors.h

C

```
#define BLUE GFX_RGBConvert(0, 0, 128)
```

Description

This is macro BLUE.

BRIGHTBLUE Macro

File

gfx_colors.h

C

```
#define BRIGHTBLUE GFX_RGBConvert(0, 0, 255)
```

Description

This is macro BRIGHTBLUE.

BRIGHTCYAN Macro

File

gfx_colors.h

C

```
#define BRIGHTCYAN GFX_RGBConvert(0, 255, 255)
```

Description

This is macro BRIGHTCYAN.

BRIGHTGREEN Macro

File

gfx_colors.h

C

```
#define BRIGHTGREEN GFX_RGBConvert(0, 255, 0)
```

Description

This is macro BRIGHTGREEN.

BRIGHTMAGENTA Macro

File

gfx_colors.h

C

```
#define BRIGHTMAGENTA GFX_RGBConvert(255, 0, 255)
```

Description

This is macro BRIGHTMAGENTA.

BRIGHTRED Macro

File

gfx_colors.h

C

```
#define BRIGHTRED GFX_RGBConvert(255, 0, 0)
```

Description

This is macro BRIGHTRED.

BRIGHTYELLOW Macro

File

gfx_colors.h

C

```
#define BRIGHTYELLOW GFX_RGBConvert(255, 255, 0)
```

Description

This is macro BRIGHTYELLOW.

BROWN Macro

File

gfx_colors.h

C

```
#define BROWN GFX_RGBConvert(255, 128, 0)
```

Description

This is macro BROWN.

BURLYWOOD Macro

File

gfx_colors.h

C

```
#define BURLYWOOD GFX_RGBConvert(222, 184, 135)
```

Description

This is macro BURLYWOOD.

CYAN Macro

File

gfx_colors.h

C

```
#define CYAN GFX_RGBConvert(0, 128, 128)
```

Description

This is macro CYAN.

DARKGRAY Macro

File

gfx_colors.h

C

```
#define DARKGRAY GFX_RGBConvert(64, 64, 64)
```

Description

This is macro DARKGRAY.

DARKORANGE Macro

File

gfx_colors.h

C

```
#define DARKORANGE GFX_RGBConvert(255, 140, 0)
```

Description

This is macro DARKORANGE.

GFX_W3_ALICEBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_ALICEBLUE GFX_RGBConvert(240, 248, 255)
```

Description

This is macro GFX_W3_ALICEBLUE.

GFX_W3_ANTIQUEWHITE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_ANTIQUEWHITE GFX_RGBConvert(250, 235, 215)
```

Description

This is macro GFX_W3_ANTIQUEWHITE.

GFX_W3_AQUA Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_AQUA GFX_RGBConvert( 0, 255, 255)
```

Description

This is macro GFX_W3_AQUA.

GFX_W3_AQUAMARINE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_AQUAMARINE GFX_RGBConvert(127, 255, 212)
```

Description

This is macro GFX_W3_AQUAMARINE.

GFX_W3_AZURE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_AZURE GFX_RGBConvert(240, 255, 255)
```

Description

This is macro GFX_W3_AZURE.

GFX_W3_BEIGE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_BEIGE GFX_RGBConvert(245, 245, 220)
```

Description

This is macro GFX_W3_BEIGE.

GFX_W3_BISQUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_BISQUE GFX_RGBConvert(255, 228, 196)
```

Description

This is macro GFX_W3_BISQUE.

GFX_W3_BLACK Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_BLACK GFX_RGBConvert( 0, 0, 0)
```

Description

This is macro GFX_W3_BLACK.

GFX_W3_BLANCHEDALMOND Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_BLANCHEDALMOND GFX_RGBConvert(255, 235, 205)
```

Description

This is macro GFX_W3_BLANCHEDALMOND.

GFX_W3_BLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_BLUE GFX_RGBConvert( 0, 0, 255)
```

Description

This is macro GFX_W3_BLUE.

GFX_W3_BLUEVIOLET Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_BLUEVIOLET GFX_RGBConvert(138, 43, 226)
```

Description

This is macro GFX_W3_BLUEVIOLET.

GFX_W3_BROWN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_BROWN GFX_RGBConvert(165, 42, 42)
```

Description

This is macro GFX_W3_BROWN.

GFX_W3_BURLYWOOD Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_BURLYWOOD GFX_RGBConvert(222, 184, 135)
```

Description

This is macro GFX_W3_BURLYWOOD.

GFX_W3_CADETBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_CADETBLUE GFX_RGBConvert( 95, 158, 160)
```

Description

This is macro GFX_W3_CADETBLUE.

GFX_W3_CHARTREUSE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_CHARTREUSE GFX_RGBConvert(127, 255, 0)
```

Description

This is macro GFX_W3_CHARTREUSE.

GFX_W3_CHOCOLATE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_CHOCOLATE GFX_RGBConvert(210, 105, 30)
```

Description

This is macro GFX_W3_CHOCOLATE.

GFX_W3_CORAL Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_CORAL GFX_RGBConvert(255, 127, 80)
```

Description

This is macro GFX_W3_CORAL.

GFX_W3_CORNFLOWERBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_CORNFLOWERBLUE GFX_RGBConvert(100, 149, 237)
```

Description

This is macro GFX_W3_CORNFLOWERBLUE.

GFX_W3_CORNSILK Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_CORNSILK GFX_RGBConvert(255, 248, 220)
```

Description

This is macro GFX_W3_CORNSILK.

GFX_W3_CRIMSON Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_CRIMSON GFX_RGBConvert(220, 20, 60)
```

Description

This is macro GFX_W3_CRIMSON.

GFX_W3_CYAN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_CYAN GFX_RGBConvert( 0, 255, 255)
```

Description

This is macro GFX_W3_CYAN.

GFX_W3_DARKBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKBLUE GFX_RGBConvert( 0, 0, 139)
```

Description

This is macro GFX_W3_DARKBLUE.

GFX_W3_DARKCYAN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKCYAN GFX_RGBConvert( 0, 139, 139)
```

Description

This is macro GFX_W3_DARKCYAN.

GFX_W3_darkgoldenrod Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_darkgoldenrod GFX_RGBConvert(184, 134, 11)
```

Description

This is macro GFX_W3_darkgoldenrod.

GFX_W3_DARKGRAY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKGRAY GFX_RGBConvert(169, 169, 169)
```

Description

This is macro GFX_W3_DARKGRAY.

GFX_W3_DARKGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKGREEN GFX_RGBConvert( 0, 100, 0)
```

Description

This is macro GFX_W3_DARKGREEN.

GFX_W3_DARKGREY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKGREY GFX_RGBConvert(169, 169, 169)
```

Description

This is macro GFX_W3_DARKGREY.

GFX_W3_DARKHAKI Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKHAKI GFX_RGBConvert(189, 183, 107)
```

Description

This is macro GFX_W3_DARKHAKI.

GFX_W3_DARKMAGENTA Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKMAGENTA GFX_RGBConvert(139, 0, 139)
```

Description

This is macro GFX_W3_DARKMAGENTA.

GFX_W3_DARKOLIVEGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKOLIVEGREEN GFX_RGBConvert( 85, 107, 47)
```

Description

This is macro GFX_W3_DARKOLIVEGREEN.

GFX_W3_DARKORANGE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKORANGE GFX_RGBConvert(255, 140, 0)
```

Description

This is macro GFX_W3_DARKORANGE.

GFX_W3_DARKORCHID Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKORCHID GFX_RGBConvert(153, 50, 204)
```

Description

This is macro GFX_W3_DARKORCHID.

GFX_W3_DARKRED Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKRED GFX_RGBConvert(139, 0, 0)
```

Description

This is macro GFX_W3_DARKRED.

GFX_W3_DARKSALMON Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKSALMON GFX_RGBConvert(233, 150, 122)
```

Description

This is macro GFX_W3_DARKSALMON.

GFX_W3_DARKSEAGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKSEAGREEN GFX_RGBConvert(143, 188, 143)
```

Description

This is macro GFX_W3_DARKSEAGREEN.

GFX_W3_DARKSLATEBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKSLATEBLUE GFX_RGBConvert( 72, 61, 139)
```

Description

This is macro GFX_W3_DARKSLATEBLUE.

GFX_W3_DARKSLATEGRAY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKSLATEGRAY GFX_RGBConvert( 47, 79, 79)
```

Description

This is macro GFX_W3_DARKSLATEGRAY.

GFX_W3_DARKSLATEGREY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKSLATEGREY GFX_RGBConvert( 47, 79, 79)
```

Description

This is macro GFX_W3_DARKSLATEGREY.

GFX_W3_DARKTURQUOISE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKTURQUOISE GFX_RGBConvert( 0, 206, 209)
```

Description

This is macro GFX_W3_DARKTURQUOISE.

GFX_W3_DARKVIOLET Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DARKVIOLET GFX_RGBConvert(148, 0, 211)
```

Description

This is macro GFX_W3_DARKVIOLET.

GFX_W3_DEEPPINK Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DEEPPINK GFX_RGBConvert(255, 20, 147)
```

Description

This is macro GFX_W3_DEEPPINK.

GFX_W3_DEEPSKYBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DEEPSKYBLUE GFX_RGBConvert( 0, 191, 255)
```

Description

This is macro GFX_W3_DEEPSKYBLUE.

GFX_W3_DIMGRAY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DIMGRAY GFX_RGBConvert(105, 105, 105)
```

Description

This is macro GFX_W3_DIMGRAY.

GFX_W3_DIMGREY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DIMGREY GFX_RGBConvert(105, 105, 105)
```

Description

This is macro GFX_W3_DIMGREY.

GFX_W3_DODGERBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_DODGERBLUE GFX_RGBConvert( 30, 144, 255)
```

Description

This is macro GFX_W3_DODGERBLUE.

GFX_W3_FIREBRICK Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_FIREBRICK GFX_RGBConvert(178, 34, 34)
```

Description

This is macro GFX_W3_FIREBRICK.

GFX_W3_FLORALWHITE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_FLORALWHITE GFX_RGBConvert(255, 250, 240)
```

Description

This is macro GFX_W3_FLORALWHITE.

GFX_W3_FORESTGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_FORESTGREEN GFX_RGBConvert( 34, 139, 34)
```

Description

This is macro GFX_W3_FORESTGREEN.

GFX_W3_FUCHSIA Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_FUCHSIA GFX_RGBConvert(255, 0, 255)
```

Description

This is macro GFX_W3_FUCHSIA.

GFX_W3_GAINSBORO Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_GAINSBORO GFX_RGBConvert(220, 220, 220)
```

Description

This is macro GFX_W3_GAINSBORO.

GFX_W3_GHOSTWHITE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_GHOSTWHITE GFX_RGBConvert(248, 248, 255)
```

Description

This is macro GFX_W3_GHOSTWHITE.

GFX_W3_GOLD Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_GOLD GFX_RGBConvert(255, 215, 0)
```

Description

This is macro GFX_W3_GOLD.

GFX_W3_GOLDENROD Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_GOLDENROD GFX_RGBConvert(218, 165, 32)
```

Description

This is macro GFX_W3_GOLDENROD.

GFX_W3_GRAY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_GRAY GFX_RGBConvert(128, 128, 128)
```

Description

This is macro GFX_W3_GRAY.

GFX_W3_GREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_GREEN GFX_RGBConvert( 0, 128, 0)
```

Description

This is macro GFX_W3_GREEN.

GFX_W3_GREENYELLOW Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_GREENYELLOW GFX_RGBConvert(173, 255, 47)
```

Description

This is macro GFX_W3_GREENYELLOW.

GFX_W3_GREY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_GREY GFX_RGBConvert(128, 128, 128)
```

Description

This is macro GFX_W3_GREY.

GFX_W3_HONEYDEW Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_HONEYDEW GFX_RGBConvert(240, 255, 240)
```

Description

This is macro GFX_W3_HONEYDEW.

GFX_W3_HOTPINK Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_HOTPINK GFX_RGBConvert(255, 105, 180)
```

Description

This is macro GFX_W3_HOTPINK.

GFX_W3_INDIANRED Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_INDIANRED GFX_RGBConvert(205, 92, 92)
```

Description

This is macro GFX_W3_INDIANRED.

GFX_W3_INDIGO Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_INDIGO GFX_RGBConvert( 75, 0, 130)
```

Description

This is macro GFX_W3_INDIGO.

GFX_W3_IVORY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_IVORY GFX_RGBConvert(255, 255, 240)
```

Description

This is macro GFX_W3_IVORY.

GFX_W3_LAVENDER Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LAVENDER GFX_RGBConvert(230, 230, 250)
```

Description

This is macro GFX_W3_LAVENDER.

GFX_W3_LAVENDERBLUSH Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LAVENDERBLUSH GFX_RGBConvert(255, 240, 245)
```

Description

This is macro GFX_W3_LAVENDERBLUSH.

GFX_W3_LAWNGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LAWNGREEN GFX_RGBConvert(124, 252, 0)
```

Description

This is macro GFX_W3_LAWNGREEN.

GFX_W3_LEMONCHIFFON Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LEMONCHIFFON GFX_RGBConvert(255, 250, 205)
```

Description

This is macro GFX_W3_LEMONCHIFFON.

GFX_W3_LIGHTBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTBLUE GFX_RGBConvert(173, 216, 230)
```

Description

This is macro GFX_W3_LIGHTBLUE.

GFX_W3_LIGHTCORAL Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTCORAL GFX_RGBConvert(240, 128, 128)
```

Description

This is macro GFX_W3_LIGHTCORAL.

GFX_W3_LIGHTCYAN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTCYAN GFX_RGBConvert(224, 255, 255)
```

Description

This is macro GFX_W3_LIGHTCYAN.

GFX_W3_LIGHTGOLDENRODYELLOW Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTGOLDENRODYELLOW GFX_RGBConvert(250, 250, 210)
```

Description

This is macro GFX_W3_LIGHTGOLDENRODYELLOW.

GFX_W3_LIGHTGRAY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTGRAY GFX_RGBConvert(211, 211, 211)
```

Description

This is macro GFX_W3_LIGHTGRAY.

GFX_W3_LIGHTGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTGREEN GFX_RGBConvert(144, 238, 144)
```

Description

This is macro GFX_W3_LIGHTGREEN.

GFX_W3_LIGHTGREY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTGREY GFX_RGBConvert(211, 211, 211)
```

Description

This is macro GFX_W3_LIGHTGREY.

GFX_W3_LIGHTPINK Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTPINK GFX_RGBConvert(255, 182, 193)
```

Description

This is macro GFX_W3_LIGHTPINK.

GFX_W3_LIGHTSALMON Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTSALMON GFX_RGBConvert(255, 160, 122)
```

Description

This is macro GFX_W3_LIGHTSALMON.

GFX_W3_LIGHTSKYBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTSKYBLUE GFX_RGBConvert(135, 206, 250)
```

Description

This is macro GFX_W3_LIGHTSKYBLUE.

GFX_W3_LIGHTSLATEGRAY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTSLATEGRAY GFX_RGBConvert(119, 136, 153)
```

Description

This is macro GFX_W3_LIGHTSLATEGRAY.

GFX_W3_LIGHTSLATEGREY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTSLATEGREY GFX_RGBConvert(119, 136, 153)
```

Description

This is macro GFX_W3_LIGHTSLATEGREY.

GFX_W3_LIGHTSTEELBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTSTEELBLUE GFX_RGBConvert(176, 196, 222)
```

Description

This is macro GFX_W3_LIGHTSTEELBLUE.

GFX_W3_LIGHTYELLOW Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGHTYELLOW GFX_RGBConvert(255, 255, 224)
```

Description

This is macro GFX_W3_LIGHTYELLOW.

GFX_W3_LIGTHSEAGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIGTHSEAGREEN GFX_RGBConvert( 32, 178, 170)
```

Description

This is macro GFX_W3_LIGTHSEAGREEN.

GFX_W3_LIME Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIME GFX_RGBConvert( 0, 255, 0)
```

Description

This is macro GFX_W3_LIME.

GFX_W3_LIMEGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LIMEGREEN GFX_RGBConvert( 50, 205, 50)
```

Description

This is macro GFX_W3_LIMEGREEN.

GFX_W3_LINEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_LINEN GFX_RGBConvert(250, 240, 230)
```

Description

This is macro GFX_W3_LINEN.

GFX_W3_MAGENTA Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MAGENTA GFX_RGBConvert(255, 0, 255)
```

Description

This is macro GFX_W3_MAGENTA.

GFX_W3_MAROON Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MAROON GFX_RGBConvert(128, 0, 0)
```

Description

This is macro GFX_W3_MAROON.

GFX_W3_MEDIUMAQUAMARINE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MEDIUMAQUAMARINE GFX_RGBConvert(102, 205, 170)
```

Description

This is macro GFX_W3_MEDIUMAQUAMARINE.

GFX_W3_MEDIUMBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MEDIUMBLUE GFX_RGBConvert( 0, 0, 205)
```

Description

This is macro GFX_W3_MEDIUMBLUE.

GFX_W3_MEDIUMORCHID Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MEDIUMORCHID GFX_RGBConvert(186, 85, 211)
```

Description

This is macro GFX_W3_MEDIUMORCHID.

GFX_W3_MEDIUMPURPLE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MEDIUMPURPLE GFX_RGBConvert(147, 112, 219)
```

Description

This is macro GFX_W3_MEDIUMPURPLE.

GFX_W3_MEDIUMSEAGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MEDIUMSEAGREEN GFX_RGBConvert( 60, 179, 113)
```

Description

This is macro GFX_W3_MEDIUMSEAGREEN.

GFX_W3_MEDIUMSLATEBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MEDIUMSLATEBLUE GFX_RGBConvert(123, 104, 238)
```

Description

This is macro GFX_W3_MEDIUMSLATEBLUE.

GFX_W3_MEDIUMSPRINGGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MEDIUMSPRINGGREEN GFX_RGBConvert( 0, 250, 154)
```

Description

This is macro GFX_W3_MEDIUMSPRINGGREEN.

GFX_W3_MEDIUMTURQUOISE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MEDIUMTURQUOISE GFX_RGBConvert( 72, 209, 204)
```

Description

This is macro GFX_W3_MEDIUMTURQUOISE.

GFX_W3_MEDIUMVIOLETRED Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MEDIUMVIOLETRED GFX_RGBConvert(199, 21, 133)
```

Description

This is macro GFX_W3_MEDIUMVIOLETRED.

GFX_W3_MIDNIGHTBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MIDNIGHTBLUE GFX_RGBConvert( 25, 25, 112)
```

Description

This is macro GFX_W3_MIDNIGHTBLUE.

GFX_W3_MINTCREAM Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MINTCREAM GFX_RGBConvert(245, 255, 250)
```

Description

This is macro GFX_W3_MINTCREAM.

GFX_W3_MISTYROSE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MISTYROSE GFX_RGBConvert(255, 228, 225)
```

Description

This is macro GFX_W3_MISTYROSE.

GFX_W3_MOCCASIN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_MOCCASIN GFX_RGBConvert(255, 228, 181)
```

Description

This is macro GFX_W3_MOCCASIN.

GFX_W3_NAVAJOWHITE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_NAVAJOWHITE GFX_RGBConvert(255, 222, 173)
```

Description

This is macro GFX_W3_NAVAJOWHITE.

GFX_W3_NAVY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_NAVY GFX_RGBConvert( 0, 0, 128)
```

Description

This is macro GFX_W3_NAVY.

GFX_W3_OLDLACE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_OLDLACE GFX_RGBConvert(253, 245, 230)
```

Description

This is macro GFX_W3_OLDLACE.

GFX_W3_OLIVE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_OLIVE GFX_RGBConvert(128, 128, 0)
```

Description

This is macro GFX_W3_OLIVE.

GFX_W3_OLIVEDRAB Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_OLIVEDRAB GFX_RGBConvert(107, 142, 35)
```

Description

This is macro GFX_W3_OLIVEDRAB.

GFX_W3_ORANGE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_ORANGE GFX_RGBConvert(255, 165, 0)
```

Description

This is macro GFX_W3_ORANGE.

GFX_W3_ORANGERED Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_ORANGERED GFX_RGBConvert(255, 69, 0)
```

Description

This is macro GFX_W3_ORANGERED.

GFX_W3_ORCHID Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_ORCHID GFX_RGBConvert(218, 112, 214)
```

Description

This is macro GFX_W3_ORCHID.

GFX_W3_PALEGOLDENROD Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_PALEGOLDENROD GFX_RGBConvert(238, 232, 170)
```

Description

This is macro GFX_W3_PALEGOLDENROD.

GFX_W3_PALEGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_PALEGREEN GFX_RGBConvert(152, 251, 152)
```

Description

This is macro GFX_W3_PALEGREEN.

GFX_W3_PALETURQUOISE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_PALETURQUOISE GFX_RGBConvert(175, 238, 238)
```

Description

This is macro GFX_W3_PALETURQUOISE.

GFX_W3_PALEVIOLETRED Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_PALEVIOLETRED GFX_RGBConvert(219, 112, 147)
```

Description

This is macro GFX_W3_PALEVIOLETRED.

GFX_W3_PAPAYAWHIP Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_PAPAYAWHIP GFX_RGBConvert(255, 239, 213)
```

Description

This is macro GFX_W3_PAPAYAWHIP.

GFX_W3_PEACHPUFF Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_PEACHPUFF GFX_RGBConvert(255, 218, 185)
```

Description

This is macro GFX_W3_PEACHPUFF.

GFX_W3_PERU Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_PERU GFX_RGBConvert(205, 133, 63)
```

Description

This is macro GFX_W3_PERU.

GFX_W3_PINK Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_PINK GFX_RGBConvert(255, 192, 203)
```

Description

This is macro GFX_W3_PINK.

GFX_W3_PLUM Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_PLUM GFX_RGBConvert(221, 160, 221)
```

Description

This is macro GFX_W3_PLUM.

GFX_W3_POWDERBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_POWDERBLUE GFX_RGBConvert(176, 224, 230)
```

Description

This is macro GFX_W3_POWDERBLUE.

GFX_W3_PURPLE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_PURPLE GFX_RGBConvert(128, 0, 128)
```

Description

This is macro GFX_W3_PURPLE.

GFX_W3_RED Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_RED GFX_RGBConvert(255, 0, 0)
```

Description

This is macro GFX_W3_RED.

GFX_W3_ROSYBROWN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_ROSYBROWN GFX_RGBConvert(188, 143, 143)
```

Description

This is macro GFX_W3_ROSYBROWN.

GFX_W3_ROYALBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_ROYALBLUE GFX_RGBConvert( 65, 105, 225)
```

Description

This is macro GFX_W3_ROYALBLUE.

GFX_W3_SADDLEBROWN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_SADDLEBROWN GFX_RGBConvert(139, 69, 19)
```

Description

This is macro GFX_W3_SADDLEBROWN.

GFX_W3_SALMON Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_SALMON GFX_RGBConvert(250, 128, 114)
```

Description

This is macro GFX_W3_SALMON.

GFX_W3_SANDYGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_SANDYGREEN GFX_RGBConvert(244, 164, 96)
```

Description

This is macro GFX_W3_SANDYGREEN.

GFX_W3_SEAGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_SEAGREEN GFX_RGBConvert( 46, 139, 87)
```

Description

This is macro GFX_W3_SEAGREEN.

GFX_W3_SEASHELL Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_SEASHELL GFX_RGBConvert(255, 245, 238)
```

Description

This is macro GFX_W3_SEASHELL.

GFX_W3_SIENNA Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_SIENNA GFX_RGBConvert(160, 82, 45)
```

Description

This is macro GFX_W3_SIENNA.

GFX_W3_SILVER Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_SILVER GFX_RGBConvert(192, 192, 192)
```

Description

This is macro GFX_W3_SILVER.

GFX_W3_SKYBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_SKYBLUE GFX_RGBConvert(135, 206, 235)
```

Description

This is macro GFX_W3_SKYBLUE.

GFX_W3_SLATEBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_SLATEBLUE GFX_RGBConvert(106, 90, 205)
```

Description

This is macro GFX_W3_SLATEBLUE.

GFX_W3_SLATEGRAY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_SLATEGRAY GFX_RGBConvert(112, 128, 144)
```

Description

This is macro GFX_W3_SLATEGRAY.

GFX_W3_SLATEGREY Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_SLATEGREY GFX_RGBConvert(112, 128, 144)
```

Description

This is macro GFX_W3_SLATEGREY.

GFX_W3_SNOW Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_SNOW GFX_RGBConvert(255, 250, 250)
```

Description

This is macro GFX_W3_SNOW.

GFX_W3_SPRINGGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_SPRINGGREEN GFX_RGBConvert( 0, 255, 127)
```

Description

This is macro GFX_W3_SPRINGGREEN.

GFX_W3_STEELBLUE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_STEELBLUE GFX_RGBConvert( 70, 130, 180)
```

Description

This is macro GFX_W3_STEELBLUE.

GFX_W3_TAN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_TAN GFX_RGBConvert(210, 180, 140)
```

Description

This is macro GFX_W3_TAN.

GFX_W3_TEAL Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_TEAL GFX_RGBConvert( 0, 128, 128)
```

Description

This is macro GFX_W3_TEAL.

GFX_W3_THISTLE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_THISTLE GFX_RGBConvert(216, 191, 216)
```

Description

This is macro GFX_W3_THISTLE.

GFX_W3_TOMATO Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_TOMATO GFX_RGBConvert(255, 99, 71)
```

Description

This is macro GFX_W3_TOMATO.

GFX_W3_TURQUOISE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_TURQUOISE GFX_RGBConvert( 64, 224, 208)
```

Description

This is macro GFX_W3_TURQUOISE.

GFX_W3_VIOLET Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_VIOLET GFX_RGBConvert(238, 130, 238)
```

Description

This is macro GFX_W3_VIOLET.

GFX_W3_WHEAT Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_WHEAT GFX_RGBConvert(245, 222, 179)
```

Description

This is macro GFX_W3_WHEAT.

GFX_W3_WHITE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_WHITE GFX_RGBConvert(255, 255, 255)
```

Description

This is macro GFX_W3_WHITE.

GFX_W3_WHITESMOKE Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_WHITESMOKE GFX_RGBConvert(245, 245, 245)
```

Description

This is macro GFX_W3_WHITESMOKE.

GFX_W3_YELLOW Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_YELLOW GFX_RGBConvert(255, 255, 0)
```

Description

This is macro GFX_W3_YELLOW.

GFX_W3_YELLOWGREEN Macro

File

gfx_colors_w3.h

C

```
#define GFX_W3_YELLOWGREEN GFX_RGBConvert(154, 205, 50)
```

Description

This is macro GFX_W3_YELLOWGREEN.

GFX_X11_ALICE_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_ALICE_BLUE GFX_RGBConvert(0xF0, 0xF8, 0xFF)
```

Description

This is macro GFX_X11_ALICE_BLUE.

GFX_X11_ANTIQUE_WHITE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_ANTIQUE_WHITE GFX_RGBConvert(0xFA, 0xEB, 0xD7)
```

Description

This is macro GFX_X11_ANTIQUE_WHITE.

GFX_X11_AQUA Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_AQUA GFX_RGBConvert(0x00, 0xFF, 0xFF)
```

Description

This is macro GFX_X11_AQUA.

GFX_X11_AQUAMARINE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_AQUAMARINE GFX_RGBConvert(0x7F, 0xFF, 0xD4)
```

Description

This is macro GFX_X11_AQUAMARINE.

GFX_X11_AZURE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_AZURE GFX_RGBConvert(0xF0, 0xFF, 0xFF)
```

Description

This is macro GFX_X11_AZURE.

GFX_X11_BEIGE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_BEIGE GFX_RGBConvert(0xF5, 0xF5, 0xDC)
```

Description

This is macro GFX_X11_BEIGE.

GFX_X11_BISQUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_BISQUE GFX_RGBConvert(0xFF, 0xE4, 0xC4)
```

Description

This is macro GFX_X11_BISQUE.

GFX_X11_BLACK Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_BLACK GFX_RGBConvert(0x00, 0x00, 0x00)
```

Description

This is macro GFX_X11_BLACK.

GFX_X11_BLANCHED_ALMOND Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_BLANCHED_ALMOND GFX_RGBConvert(0xFF, 0xEB, 0xCD)
```

Description

This is macro GFX_X11_BLANCHED_ALMOND.

GFX_X11_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_BLUE GFX_RGBConvert(0x00, 0x00, 0xFF)
```

Description

This is macro GFX_X11_BLUE.

GFX_X11_BLUE_VIOLET Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_BLUE_VIOLET GFX_RGBConvert(0x8A, 0x2B, 0xE2)
```

Description

This is macro GFX_X11_BLUE_VIOLET.

GFX_X11_BROWN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_BROWN GFX_RGBConvert(0xA5, 0x2A, 0x2A)
```

Description

This is macro GFX_X11_BROWN.

GFX_X11_BURLY_WOOD Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_BURLY_WOOD GFX_RGBConvert(0xDE, 0xB8, 0x87)
```

Description

This is macro GFX_X11_BURLY_WOOD.

GFX_X11_CADEL_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_CADEL_BLUE GFX_RGBConvert(0x5F, 0x9E, 0xA0)
```

Description

This is macro GFX_X11_CADEL_BLUE.

GFX_X11_CHARTEUSE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_CHARTEUSE GFX_RGBConvert(0x7F, 0xFF, 0x00)
```

Description

This is macro GFX_X11_CHARTEUSE.

GFX_X11_CHOCOLATE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_CHOCOLATE GFX_RGBConvert(0xD2, 0x69, 0x1E)
```

Description

This is macro GFX_X11_CHOCOLATE.

GFX_X11_CORAL Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_CORAL GFX_RGBConvert(0xFF, 0x7F, 0x50)
```

Description

This is macro GFX_X11_CORAL.

GFX_X11_CORNFLOWER_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_CORNFLOWER_BLUE GFX_RGBConvert(0x64, 0x95, 0xED)
```

Description

This is macro GFX_X11_CORNFLOWER_BLUE.

GFX_X11_CORNSILK Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_CORNSILK GFX_RGBConvert(0xFF, 0xF8, 0xDC)
```

Description

X11: Brown Colors

GFX_X11_CRIMSON Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_CRIMSON GFX_RGBConvert(0xDC, 0x14, 0x3C)
```

Description

This is macro GFX_X11_CRIMSON.

GFX_X11_CYAN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_CYAN GFX_RGBConvert(0x00, 0xFF, 0xFF)
```

Description

This is macro GFX_X11_CYAN.

GFX_X11_DARK_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_BLUE GFX_RGBConvert(0x00, 0x00, 0x8B)
```

Description

This is macro GFX_X11_DARK_BLUE.

GFX_X11_DARK_CYAN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_CYAN GFX_RGBConvert(0x00, 0x8B, 0x8B)
```

Description

This is macro GFX_X11_DARK_CYAN.

GFX_X11_DARK_GOLDENROD Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_GOLDENROD GFX_RGBConvert(0xB8, 0x86, 0x0B)
```

Description

This is macro GFX_X11_DARK_GOLDENROD.

GFX_X11_DARK_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_GREEN GFX_RGBConvert(0x00, 0x64, 0x00)
```

Description

This is macro GFX_X11_DARK_GREEN.

GFX_X11_DARK_GREY Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_GREY GFX_RGBConvert(0xA9, 0xA9, 0xA9)
```

Description

This is macro GFX_X11_DARK_GREY.

GFX_X11_DARK_KHAKI Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_KHAKI GFX_RGBConvert(0xBD, 0xB7, 0x6B)
```

Description

This is macro GFX_X11_DARK_KHAKI.

GFX_X11_DARK_MAGENTA Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_MAGENTA GFX_RGBConvert(0x8B, 0x00, 0x8B)
```

Description

This is macro GFX_X11_DARK_MAGENTA.

GFX_X11_DARK_OLIVE_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_OLIVE_GREEN GFX_RGBConvert(0x55, 0x6B, 0x2F)
```

Description

X11: Green Colors

GFX_X11_DARK_ORANGE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_ORANGE GFX_RGBConvert(0xFF, 0x8C, 0x00)
```

Description

This is macro GFX_X11_DARK_ORANGE.

GFX_X11_DARK_ORCHID Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_ORCHID GFX_RGBConvert(0x99, 0x32, 0xCC)
```

Description

This is macro GFX_X11_DARK_ORCHID.

GFX_X11_DARK_RED Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_RED GFX_RGBConvert(0x8B, 0x00, 0x00)
```

Description

This is macro GFX_X11_DARK_RED.

GFX_X11_DARK_SALMON Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_SALMON GFX_RGBConvert(0xEA, 0x96, 0x7A)
```

Description

This is macro GFX_X11_DARK_SALMON.

GFX_X11_DARK_SEA_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_SEA_GREEN GFX_RGBConvert(0x8F, 0xBC, 0x8F)
```

Description

This is macro GFX_X11_DARK_SEA_GREEN.

GFX_X11_DARK_SLATE_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_SLATE_BLUE GFX_RGBConvert(0x48, 0x3D, 0x8B)
```

Description

This is macro GFX_X11_DARK_SLATE_BLUE.

GFX_X11_DARK_SLATE_GREY Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_SLATE_GREY GFX_RGBConvert(0x2F, 0x4F, 0x4F)
```

Description

This is macro GFX_X11_DARK_SLATE_GREY.

GFX_X11_DARK_TURQUOISE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_TURQUOISE GFX_RGBConvert(0x00, 0xCE, 0xD1)
```

Description

This is macro GFX_X11_DARK_TURQUOISE.

GFX_X11_DARK_VIOLET Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DARK_VIOLET GFX_RGBConvert(0x94, 0x00, 0xD3)
```

Description

This is macro GFX_X11_DARK_VIOLET.

GFX_X11_DEEP_PINK Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DEEP_PINK GFX_RGBConvert(0xFF, 0x14, 0x93)
```

Description

This is macro GFX_X11_DEEP_PINK.

GFX_X11_DEEP_SKY_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DEEP_SKY_BLUE GFX_RGBConvert(0x00, 0xBF, 0xFF)
```

Description

This is macro GFX_X11_DEEP_SKY_BLUE.

GFX_X11_DIM_GREY Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DIM_GREY GFX_RGBConvert(0x69, 0x69, 0x69)
```

Description

This is macro GFX_X11_DIM_GREY.

GFX_X11_DODGER_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_DODGER_BLUE GFX_RGBConvert(0x1E, 0x90, 0xFF)
```

Description

This is macro GFX_X11_DODGER_BLUE.

GFX_X11_FIRE_BRICK Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_FIRE_BRICK GFX_RGBConvert(0xB2, 0x22, 0x22)
```

Description

This is macro GFX_X11_FIRE_BRICK.

GFX_X11_FLORAL_WHITE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_FLORAL_WHITE GFX_RGBConvert(0xFF, 0xFA, 0xF0)
```

Description

This is macro GFX_X11_FLORAL_WHITE.

GFX_X11_FOREST_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_FOREST_GREEN GFX_RGBConvert(0x22, 0x8B, 0x22)
```

Description

This is macro GFX_X11_FOREST_GREEN.

GFX_X11_FUCHSIA Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_FUCHSIA GFX_RGBConvert(0xFF, 0x00, 0xFF)
```

Description

This is macro GFX_X11_FUCHSIA.

GFX_X11_GAINSBORO Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_GAINSBORO GFX_RGBConvert(0xDC, 0xDC, 0xDC)
```

Description

This is macro GFX_X11_GAINSBORO.

GFX_X11_GHOST_WHITE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_GHOST_WHITE GFX_RGBConvert(0xF8, 0xF8, 0xFF)
```

Description

This is macro GFX_X11_GHOST_WHITE.

GFX_X11_GOLD Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_GOLD GFX_RGBConvert(0xFF, 0xD7, 0x00)
```

Description

This is macro GFX_X11_GOLD.

GFX_X11_GOLDENROD Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_GOLDENROD GFX_RGBConvert(0xDA, 0xA5, 0x20)
```

Description

This is macro GFX_X11_GOLDENROD.

GFX_X11_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_GREEN GFX_RGBConvert(0x00, 0x80, 0x00)
```

Description

This is macro GFX_X11_GREEN.

GFX_X11_GREEN_YELLOW Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_GREEN_YELLOW GFX_RGBConvert(0xAD, 0xFF, 0x2F)
```

Description

This is macro GFX_X11_GREEN_YELLOW.

GFX_X11_GREY Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_GREY GFX_RGBConvert(0x80, 0x80, 0x80)
```

Description

This is macro GFX_X11_GREY.

GFX_X11_HONEYDEW Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_HONEYDEW GFX_RGBConvert(0xF0, 0xFF, 0xF0)
```

Description

This is macro GFX_X11_HONEYDEW.

GFX_X11_HOT_PINK Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_HOT_PINK GFX_RGBConvert(0xFF, 0x69, 0xB4)
```

Description

This is macro GFX_X11_HOT_PINK.

GFX_X11_INDIAN_RED Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_INDIAN_RED GFX_RGBConvert(0xCD, 0x5C, 0x5C)
```

Description

This is macro GFX_X11_INDIAN_RED.

GFX_X11_INDIGO Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_INDIGO GFX_RGBConvert(0x4B, 0x00, 0x82)
```

Description

This is macro GFX_X11_INDIGO.

GFX_X11_IVORY Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_IVORY GFX_RGBConvert(0xFF, 0xFF, 0xF0)
```

Description

This is macro GFX_X11_IVORY.

GFX_X11_KHAKI Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_KHAKI GFX_RGBConvert(0xF0, 0xE6, 0x8C)
```

Description

This is macro GFX_X11_KHAKI.

GFX_X11_LAVENDER Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LAVENDER GFX_RGBConvert(0xE6, 0xE6, 0xFA)
```

Description

X11: BLUE COLORS

GFX_X11_LAVENDOR_BLUSH Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LAVENDOR_BLUSH GFX_RGBConvert(0xFF, 0xF0, 0xF5)
```

Description

This is macro GFX_X11_LAVENDOR_BLUSH.

GFX_X11_LAWN_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LAWN_GREEN GFX_RGBConvert(0x7C, 0xFC, 0x00)
```

Description

This is macro GFX_X11_LAWN_GREEN.

GFX_X11_LEMON_CHIFFON Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LEMON_CHIFFON GFX_RGBConvert(0xFF, 0xFA, 0xCD)
```

Description

This is macro GFX_X11_LEMON_CHIFFON.

GFX_X11_LIGHT_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIGHT_BLUE GFX_RGBConvert(0xAD, 0xD8, 0xE6)
```

Description

This is macro GFX_X11_LIGHT_BLUE.

GFX_X11_LIGHT_CAROL Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIGHT_CAROL GFX_RGBConvert(0xF0, 0x80, 0x80)
```

Description

This is macro GFX_X11_LIGHT_CAROL.

GFX_X11_LIGHT_CYAN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIGHT_CYAN GFX_RGBConvert(0xE0, 0xFF, 0xFF)
```

Description

This is macro GFX_X11_LIGHT_CYAN.

GFX_X11_LIGHT_GOLDENROD_YELLOW Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIGHT_GOLDENROD_YELLOW GFX_RGBConvert(0xFA, 0xFA, 0xD2)
```

Description

This is macro GFX_X11_LIGHT_GOLDENROD_YELLOW.

GFX_X11_LIGHT_GRAY Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIGHT_GRAY GFX_RGBConvert(0xD3, 0xD3, 0xD3)
```

Description

This is macro GFX_X11_LIGHT_GRAY.

GFX_X11_LIGHT_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIGHT_GREEN GFX_RGBConvert(0x90, 0xEE, 0x90)
```

Description

This is macro GFX_X11_LIGHT_GREEN.

GFX_X11_LIGHT_PINK Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIGHT_PINK GFX_RGBConvert(0xFF, 0xB6, 0xC1)
```

Description

This is macro GFX_X11_LIGHT_PINK.

GFX_X11_LIGHT_SALMON Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIGHT_SALMON GFX_RGBConvert(0xFF, 0xA0, 0x7A)
```

Description

X11: Red Colors

GFX_X11_LIGHT_SEA_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIGHT_SEA_GREEN GFX_RGBConvert(0x20, 0xB2, 0xAA)
```

Description

This is macro GFX_X11_LIGHT_SEA_GREEN.

GFX_X11_LIGHT_SKY_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIGHT_SKY_BLUE GFX_RGBConvert(0x87, 0xCE, 0xFA)
```

Description

This is macro GFX_X11_LIGHT_SKY_BLUE.

GFX_X11_LIGHT_SLATE_GREY Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIGHT_SLATE_GREY GFX_RGBConvert(0x77, 0x88, 0x99)
```

Description

This is macro GFX_X11_LIGHT_SLATE_GREY.

GFX_X11_LIGHT_STEEL_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIGHT_STEEL_BLUE GFX_RGBConvert(0xB0, 0xC4, 0xDE)
```

Description

X11: BLUE COLORS

GFX_X11_LIGHT_YELLOW Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIGHT_YELLOW GFX_RGBConvert(0xFF, 0xFF, 0xE0)
```

Description

This is macro GFX_X11_LIGHT_YELLOW.

GFX_X11_LIME Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIME GFX_RGBConvert(0x00, 0xFF, 0x00)
```

Description

This is macro GFX_X11_LIME.

GFX_X11_LIME_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LIME_GREEN GFX_RGBConvert(0x32, 0xCD, 0x32)
```

Description

This is macro GFX_X11_LIME_GREEN.

GFX_X11_LINEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_LINEN GFX_RGBConvert(0xFA, 0xF0, 0xE6)
```

Description

This is macro GFX_X11_LINEN.

GFX_X11_MAGENTA Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MAGENTA GFX_RGBConvert(0xFF, 0x00, 0xFF)
```

Description

This is macro GFX_X11_MAGENTA.

GFX_X11_MARRON Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MARRON GFX_RGBConvert(0x80, 0x00, 0x00)
```

Description

This is macro GFX_X11_MARRON.

GFX_X11_MEDIUM_AQUAMARINE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MEDIUM_AQUAMARINE GFX_RGBConvert(0x66, 0xCD, 0xAA)
```

Description

X11: CYAN COLORS

GFX_X11_MEDIUM_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MEDIUM_BLUE GFX_RGBConvert(0x00, 0x00, 0xCD)
```

Description

This is macro GFX_X11_MEDIUM_BLUE.

GFX_X11_MEDIUM_ORCHID Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MEDIUM_ORCHID GFX_RGBConvert(0xBA, 0x55, 0xD3)
```

Description

This is macro GFX_X11_MEDIUM_ORCHID.

GFX_X11_MEDIUM_PURPLE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MEDIUM_PURPLE GFX_RGBConvert(0x93, 0x70, 0xDB)
```

Description

This is macro GFX_X11_MEDIUM_PURPLE.

GFX_X11_MEDIUM_SEA_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MEDIUM_SEA_GREEN GFX_RGBConvert(0x3C, 0xB3, 0x71)
```

Description

This is macro GFX_X11_MEDIUM_SEA_GREEN.

GFX_X11_MEDIUM_SLATE_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MEDIUM_SLATE_BLUE GFX_RGBConvert(0x7B, 0x68, 0xEE)
```

Description

This is macro GFX_X11_MEDIUM_SLATE_BLUE.

GFX_X11_MEDIUM_SPRING_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MEDIUM_SPRING_GREEN GFX_RGBConvert(0x00, 0xFA, 0x9A)
```

Description

This is macro GFX_X11_MEDIUM_SPRING_GREEN.

GFX_X11_MEDIUM_TURQUOISE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MEDIUM_TURQUOISE GFX_RGBConvert(0x48, 0xD1, 0xCC)
```

Description

This is macro GFX_X11_MEDIUM_TURQUOISE.

GFX_X11_MEDIUM_VIOLET_RED Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MEDIUM_VIOLET_RED GFX_RGBConvert(0xC7, 0x15, 0x85)
```

Description

This is macro GFX_X11_MEDIUM_VIOLET_RED.

GFX_X11_MIDNIGHT_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MIDNIGHT_BLUE GFX_RGBConvert(0x19, 0x19, 0x70)
```

Description

This is macro GFX_X11_MIDNIGHT_BLUE.

GFX_X11_MINT_CREAM Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MINT_CREAM GFX_RGBConvert(0xF5, 0xFF, 0xFA)
```

Description

This is macro GFX_X11_MINT_CREAM.

GFX_X11_MISTY_ROSE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MISTY_ROSE GFX_RGBConvert(0xFF, 0xE4, 0xE1)
```

Description

This is macro GFX_X11_MISTY_ROSE.

GFX_X11_MOCCASIN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_MOCCASIN GFX_RGBConvert(0xFF, 0xE4, 0xB5)
```

Description

This is macro GFX_X11_MOCCASIN.

GFX_X11_NAVAJO_WHITE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_NAVAJO_WHITE GFX_RGBConvert(0xFF, 0xDE, 0xAD)
```

Description

This is macro GFX_X11_NAVAJO_WHITE.

GFX_X11_NAVY Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_NAVY GFX_RGBConvert(0x00, 0x00, 0x80)
```

Description

This is macro GFX_X11_NAVY.

GFX_X11_OLD_LACE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_OLD_LACE GFX_RGBConvert(0xFD, 0xF5, 0xE6)
```

Description

This is macro GFX_X11_OLD_LACE.

GFX_X11_OLIVE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_OLIVE GFX_RGBConvert(0x80, 0x80, 0x00)
```

Description

This is macro GFX_X11_OLIVE.

GFX_X11_OLIVE_DRAB Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_OLIVE_DRAB GFX_RGBConvert(0x6B, 0x8E, 0x23)
```

Description

This is macro GFX_X11_OLIVE_DRAB.

GFX_X11_ORANGE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_ORANGE GFX_RGBConvert(0xFF, 0xA5, 0x00)
```

Description

This is macro GFX_X11_ORANGE.

GFX_X11_ORANGE_RED Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_ORANGE_RED GFX_RGBConvert(0xFF, 0x45, 0x00)
```

Description

X11: Orange Colors

GFX_X11_ORCHID Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_ORCHID GFX_RGBConvert(0xDA, 0x70, 0xD6)
```

Description

This is macro GFX_X11_ORCHID.

GFX_X11_PALE_GOLDENROD Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_PALE_GOLDENROD GFX_RGBConvert(0xEE, 0xE8, 0xAA)
```

Description

This is macro GFX_X11_PALE_GOLDENROD.

GFX_X11_PALE_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_PALE_GREEN GFX_RGBConvert(0x98, 0xFB, 0x98)
```

Description

This is macro GFX_X11_PALE_GREEN.

GFX_X11_PALE_TURQUOISE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_PALE_TURQUOISE GFX_RGBConvert(0xAF, 0xEE, 0xEE)
```

Description

This is macro GFX_X11_PALE_TURQUOISE.

GFX_X11_PALE_VIOLET_RED Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_PALE_VIOLET_RED GFX_RGBConvert(0xDB, 0x70, 0x93)
```

Description

This is macro GFX_X11_PALE_VIOLET_RED.

GFX_X11_PAPAYA_WHIP Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_PAPAYA_WHIP GFX_RGBConvert(0xFF, 0xEF, 0xD5)
```

Description

This is macro GFX_X11_PAPAYA_WHIP.

GFX_X11_PEACH_PUFF Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_PEACH_PUFF GFX_RGBConvert(0xFF, 0xDA, 0xB9)
```

Description

This is macro GFX_X11_PEACH_PUFF.

GFX_X11_PERU Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_PERU GFX_RGBConvert(0xCD, 0x85, 0x3F)
```

Description

This is macro GFX_X11_PERU.

GFX_X11_PINK Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_PINK GFX_RGBConvert(0xFF, 0xC0, 0xCB)
```

Description

X11: Pink Colors

GFX_X11_PLUM Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_PLUM GFX_RGBConvert(0xDD, 0xA0, 0xDD)
```

Description

This is macro GFX_X11_PLUM.

GFX_X11_POWDER_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_POWDER_BLUE GFX_RGBConvert(0xB0, 0xE0, 0xE6)
```

Description

This is macro GFX_X11_POWDER_BLUE.

GFX_X11_PURPLE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_PURPLE GFX_RGBConvert(0x80, 0x00, 0x80)
```

Description

This is macro GFX_X11_PURPLE.

GFX_X11_RED Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_RED GFX_RGBConvert(0xFF, 0x00, 0x00)
```

Description

This is macro GFX_X11_RED.

GFX_X11_ROSY_BROWN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_ROSY_BROWN GFX_RGBConvert(0xBc, 0x8F, 0x8F)
```

Description

This is macro GFX_X11_ROSY_BROWN.

GFX_X11_ROYAL_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_ROYAL_BLUE GFX_RGBConvert(0x41, 0x69, 0xE1)
```

Description

This is macro GFX_X11_ROYAL_BLUE.

GFX_X11_SADDLE_BROWN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_SADDLE_BROWN GFX_RGBConvert(0x8B, 0x45, 0x13)
```

Description

This is macro GFX_X11_SADDLE_BROWN.

GFX_X11_SALMON Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_SALMON GFX_RGBConvert(0xFA, 0x80, 0x72)
```

Description

This is macro GFX_X11_SALMON.

GFX_X11_SANDY_BROWN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_SANDY_BROWN GFX_RGBConvert(0xF4, 0xA4, 0x60)
```

Description

This is macro GFX_X11_SANDY_BROWN.

GFX_X11_SEA_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_SEA_GREEN GFX_RGBConvert(0x2E, 0x8B, 0x57)
```

Description

This is macro GFX_X11_SEA_GREEN.

GFX_X11_SEASHELL Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_SEASHELL GFX_RGBConvert(0xFF, 0xF5, 0xEE)
```

Description

This is macro GFX_X11_SEASHELL.

GFX_X11_SIENNA Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_SIENNA GFX_RGBConvert(0xA0, 0x52, 0x2D)
```

Description

This is macro GFX_X11_SIENNA.

GFX_X11_SILVER Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_SILVER GFX_RGBConvert(0xC0, 0xC0, 0xC0)
```

Description

This is macro GFX_X11_SILVER.

GFX_X11_SKY_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_SKY_BLUE GFX_RGBConvert(0x87, 0xCE, 0xEB)
```

Description

This is macro GFX_X11_SKY_BLUE.

GFX_X11_SLATE_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_SLATE_BLUE GFX_RGBConvert(0x6A, 0x5A, 0xCD)
```

Description

This is macro GFX_X11_SLATE_BLUE.

GFX_X11_SLATE_GREY Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_SLATE_GREY GFX_RGBConvert(0x70, 0x80, 0x90)
```

Description

This is macro GFX_X11_SLATE_GREY.

GFX_X11_SNOW Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_SNOW GFX_RGBConvert(0xFF, 0xFA, 0xFA)
```

Description

This is macro GFX_X11_SNOW.

GFX_X11_SPRING_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_SPRING_GREEN GFX_RGBConvert(0x00, 0xFF, 0x7F)
```

Description

This is macro GFX_X11_SPRING_GREEN.

GFX_X11_STEEL_BLUE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_STEEL_BLUE GFX_RGBConvert(0x46, 0x82, 0xB4)
```

Description

This is macro GFX_X11_STEEL_BLUE.

GFX_X11_TAN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_TAN GFX_RGBConvert(0xD2, 0xB4, 0x8C)
```

Description

This is macro GFX_X11_TAN.

GFX_X11_TEAL Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_TEAL GFX_RGBConvert(0x00, 0x80, 0x80)
```

Description

This is macro GFX_X11_TEAL.

GFX_X11_THISTLE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_THISTLE GFX_RGBConvert(0xD8, 0xBF, 0xD8)
```

Description

This is macro GFX_X11_THISTLE.

GFX_X11_TOMATO Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_TOMATO GFX_RGBConvert(0xFF, 0x63, 0x47)
```

Description

This is macro GFX_X11_TOMATO.

GFX_X11_TURQUOISE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_TURQUOISE GFX_RGBConvert(0x40, 0xE0, 0xD0)
```

Description

This is macro GFX_X11_TURQUOISE.

GFX_X11_VIOLET Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_VIOLET GFX_RGBConvert(0xEE, 0x82, 0xEE)
```

Description

This is macro GFX_X11_VIOLET.

GFX_X11_WHEAT Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_WHEAT GFX_RGBConvert(0xF5, 0xDE, 0xB3)
```

Description

This is macro GFX_X11_WHEAT.

GFX_X11_WHITE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_WHITE GFX_RGBConvert(0xFF, 0xFF, 0xFF)
```

Description

X11: WHITE/GRAY/BLACK COLORS

GFX_X11_WHITE_SMOKE Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_WHITE_SMOKE GFX_RGBConvert(0xF5, 0xF5, 0xF5)
```

Description

This is macro GFX_X11_WHITE_SMOKE.

GFX_X11_YELLOW Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_YELLOW GFX_RGBConvert(0xFF, 0xFF, 0x00)
```

Description

X11: Yellow Colors

GFX_X11_YELLOW_GREEN Macro

File

gfx_colors_x11.h

C

```
#define GFX_X11_YELLOW_GREEN GFX_RGBConvert(0x9A, 0xCD, 0x32)
```

Description

This is macro GFX_X11_YELLOW_GREEN.

GOLD Macro

File

gfx_colors.h

C

```
#define GOLD GFX_RGBConvert(255, 215, 0)
```

Description

This is macro GOLD.

GRAY000 Macro

File

gfx_colors.h

C

```
#define GRAY000 0
```

Description

This is macro GRAY000.

GRAY001 Macro

File

gfx_colors.h

C

```
#define GRAY001 1
```

Description

This is macro GRAY001.

GRAY002 Macro

File

gfx_colors.h

C

```
#define GRAY002 2
```

Description

This is macro GRAY002.

GRAY003 Macro

File

gfx_colors.h

C

```
#define GRAY003 3
```

Description

This is macro GRAY003.

GRAY004 Macro

File

gfx_colors.h

C

```
#define GRAY004 4
```

Description

This is macro GRAY004.

GRAY005 Macro

File

gfx_colors.h

C

```
#define GRAY005 5
```

Description

This is macro GRAY005.

GRAY006 Macro

File

gfx_colors.h

C

```
#define GRAY006 6
```

Description

This is macro GRAY006.

GRAY007 Macro

File

gfx_colors.h

C

```
#define GRAY007 7
```

Description

This is macro GRAY007.

GRAY008 Macro

File

gfx_colors.h

C

```
#define GRAY008 8
```

Description

This is macro GRAY008.

GRAY009 Macro

File

gfx_colors.h

C

```
#define GRAY009 9
```

Description

This is macro GRAY009.

GRAY010 Macro

File

gfx_colors.h

C

```
#define GRAY010 GFX_RGBConvert(10, 10, 10)
```

Description

This is macro GRAY010.

GRAY011 Macro

File

gfx_colors.h

C

```
#define GRAY011 11
```

Description

This is macro GRAY011.

GRAY012 Macro

File

gfx_colors.h

C

```
#define GRAY012 12
```

Description

This is macro GRAY012.

GRAY013 Macro

File

gfx_colors.h

C

```
#define GRAY013 13
```

Description

This is macro GRAY013.

GRAY014 Macro

File

gfx_colors.h

C

```
#define GRAY014 14
```

Description

This is macro GRAY014.

GRAY015 Macro

File

gfx_colors.h

C

```
#define GRAY015 15
```

Description

This is macro GRAY015.

GRAY032 Macro

File

gfx_colors.h

C

```
#define GRAY032 GFX_RGBConvert(32, 32, 32)
```

Description

This is macro GRAY032.

GRAY096 Macro

File

gfx_colors.h

C

```
#define GRAY096 GFX_RGBConvert(96, 96, 96)
```

Description

This is macro GRAY096.

GRAY128 Macro

File

gfx_colors.h

C

```
#define GRAY128 GFX_RGBConvert(128, 128, 128)
```

Description

This is macro GRAY128.

GRAY160 Macro

File

gfx_colors.h

C

```
#define GRAY160 GFX_RGBConvert(160, 160, 160)
```

Description

This is macro GRAY160.

GRAY192 Macro

File

gfx_colors.h

C

```
#define GRAY192 GFX_RGBConvert(192, 192, 192)
```

Description

This is macro GRAY192.

GRAY204 Macro

File

gfx_colors.h

C

```
#define GRAY204 GFX_RGBConvert(204, 204, 204)
```

Description

This is macro GRAY204.

GRAY224 Macro

File

gfx_colors.h

C

```
#define GRAY224 GFX_RGBConvert(224, 224, 224)
```

Description

This is macro GRAY224.

GRAY229 Macro

File

gfx_colors.h

C

```
#define GRAY229 GFX_RGBConvert(229, 229, 229)
```

Description

This is macro GRAY229.

GRAY242 Macro

File

gfx_colors.h

C

```
#define GRAY242 GFX_RGBConvert(242, 242, 242)
```

Description

This is macro GRAY242.

GREEN Macro

File

gfx_colors.h

C

```
#define GREEN GFX_RGBConvert(0, 128, 0)
```

Description

This is macro GREEN.

KHAKI Macro

File

gfx_colors_w3.h

C

```
#define KHAKI GFX_RGBConvert(240, 230, 140)
```

Description

This is macro KHAKI.

LIGHTBLUE Macro

File

gfx_colors.h

C

```
#define LIGHTBLUE GFX_RGBConvert(128, 128, 255)
```

Description

This is macro LIGHTBLUE.

LIGHTCYAN Macro

File

gfx_colors.h

C

```
#define LIGHTCYAN GFX_RGBConvert(128, 255, 255)
```

Description

This is macro LIGHTCYAN.

LIGHTGRAY Macro

File

gfx_colors.h

C

```
#define LIGHTGRAY GFX_RGBConvert(128, 128, 128)
```

Description

This is macro LIGHTGRAY.

LIGHTGREEN Macro

File

gfx_colors.h

C

```
#define LIGHTGREEN GFX_RGBConvert(128, 255, 128)
```

Description

This is macro LIGHTGREEN.

LIGHTMAGENTA Macro

File

gfx_colors.h

C

```
#define LIGHTMAGENTA GFX_RGBConvert(255, 128, 255)
```

Description

This is macro LIGHTMAGENTA.

LIGHTORANGE Macro

File

gfx_colors.h

C

```
#define LIGHTORANGE GFX_RGBConvert(255, 200, 0)
```

Description

This is macro LIGHTORANGE.

LIGHTRED Macro

File

gfx_colors.h

C

```
#define LIGHTRED GFX_RGBConvert(255, 128, 128)
```

Description

This is macro LIGHTRED.

LIGHTYELLOW Macro

File

gfx_colors.h

C

```
#define LIGHTYELLOW GFX_RGBConvert(255, 255, 150)
```

Description

This is macro LIGHTYELLOW.

MAGENTA Macro

File

gfx_colors.h

C

```
#define MAGENTA GFX_RGBConvert(128, 0, 128)
```

Description

This is macro MAGENTA.

ORANGE Macro

File

gfx_colors.h

C

```
#define ORANGE GFX_RGBConvert(255, 187, 76)
```

Description

This is macro ORANGE.

PERU Macro

File

gfx_colors.h

C

```
#define PERU GFX_RGBConvert(205, 133, 63)
```

Description

This is macro PERU.

RED Macro

File

gfx_colors.h

C

```
#define RED GFX_RGBConvert(128, 0, 0)
```

Description

This is macro RED.

SADDLEBROWN Macro

File

gfx_colors.h

C

```
#define SADDLEBROWN GFX_RGBConvert(139, 69, 19)
```

Description

This is macro SADDLEBROWN.

SIENNA Macro

File

gfx_colors.h

C

```
#define SIENNA GFX_RGBConvert(160, 82, 45)
```

Description

This is macro SIENNA.

TAN Macro

File

gfx_colors.h

C

```
#define TAN GFX_RGBConvert(210, 180, 140)
```

Description

This is macro TAN.

WHEAT Macro

File

gfx_colors.h

C

```
#define WHEAT GFX_RGBConvert(245, 245, 220)
```

Description

This is macro WHEAT.

WHITE Macro

File

gfx_colors.h

C

```
#define WHITE GFX_RGBConvert(255, 255, 255)
```

Description

This is macro WHITE.

YELLOW Macro

File

gfx_colors.h

C

```
#define YELLOW GFX_RGBConvert(255, 255, 128)
```

Description

This is macro YELLOW.

Graphics Object Layer

This section describes the API and usage of the Graphics Library Object Layer.

Graphics Object Layer API

GOL Objects

Graphics Library Object Layer object API.

Button Object

Button is an object that emulates a press and release effect when operated upon.

Functions

	Name	Description
	GFX_GOL_ButtonActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_ButtonActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_ButtonCreate	This function creates a GFX_GOL_BUTTON object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_ButtonDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_ButtonTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_ButtonTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_ButtonTextSet	This function sets the address of the current text string used by the object.

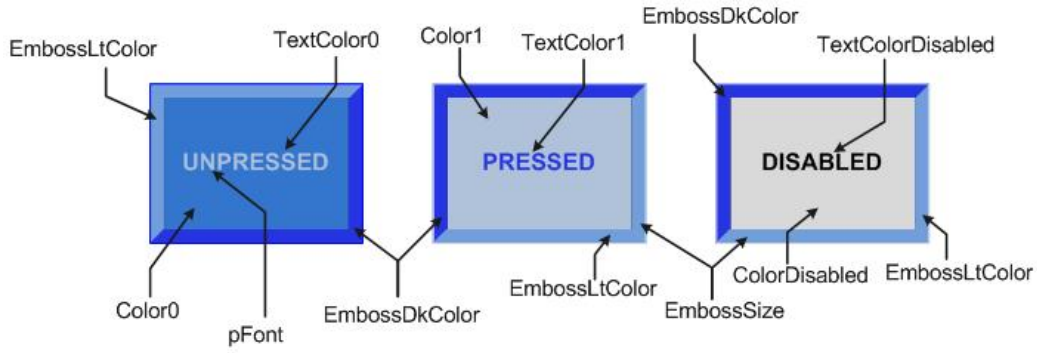
Macros

	Name	Description
	GFX_GOL_ButtonPressStateImageGet	This function gets the image used when in the pressed state.
	GFX_GOL_ButtonPressStateImageSet	This function sets the image to be used when in the pressed state.
	GFX_GOL_ButtonReleaseStateImageGet	This function gets the image used when in the released state.
	GFX_GOL_ButtonReleaseStateImageSet	This function sets the image to be used when in the released state.
	GFX_GOL_ButtonTextGet	This function returns the address of the current text string used by the object.

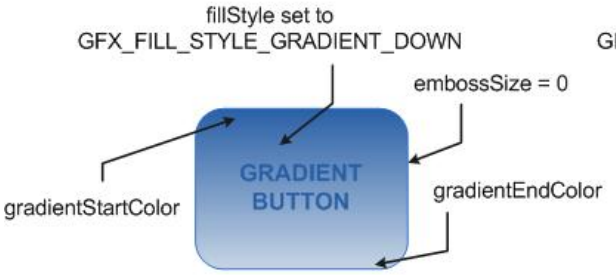
Description

Button supports Keyboard and Touchscreen inputs, replying to their events with the predefined actions (see GFX_GOL_ButtonActionGet() and GFX_GOL_ButtonActionSet() for details).

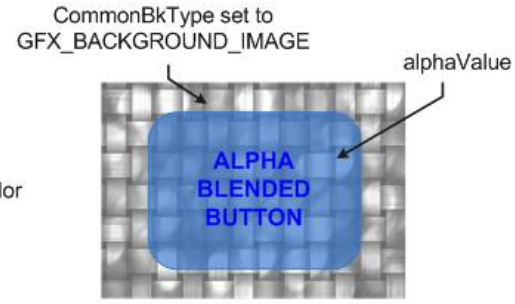
The button object is rendered using the assigned style scheme. The following figure illustrates the usage of the style schemes in the button object.



Button Style Scheme

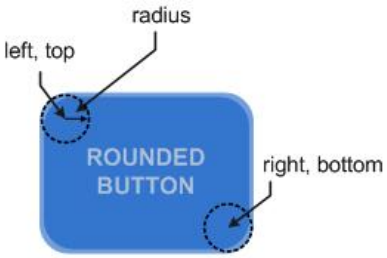


Gradient Fills in Buttons

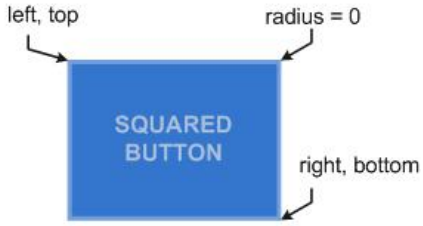


Alpha Blending in Buttons

Aside from the basic color styles, the object can also be drawn with gradient fills and alpha blended fills. Buttons can also be drawn with rectangular or rounded edges.

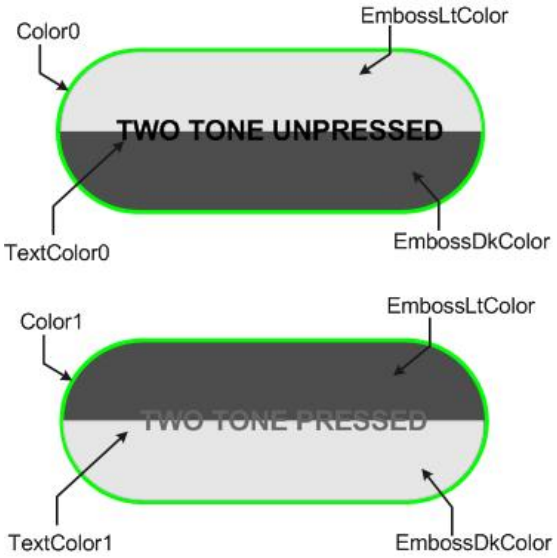


Button with Rounded Corners



Button with Squared Corners

Another variation in the rendering of button object is the two-tones mode.

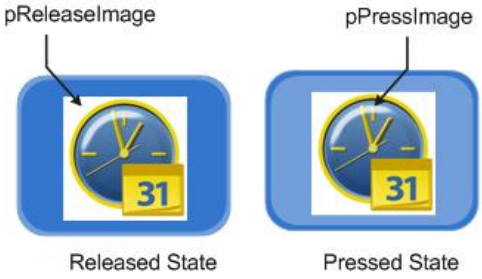


Two Tone Button

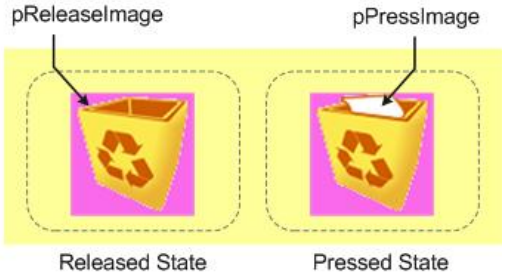
Buttons also allows alignment on the text used. The text alignment behavior of the object is the same as GFX_TextStringBoxDraw().

Instead of using text to name buttons, images or icons can also be used. Since there are two main states of the button (pressed and released state), the object allows usage of two different images. One is assigned to the pressed state and the other assigned to the released state.

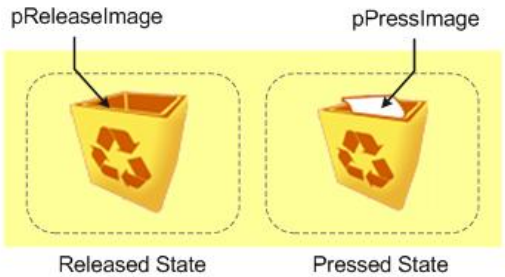
When only one image is used, both presses and released pointers should be assigned to the same image. If one state is set to NULL, then that state will not render any image.



Button with Images



Button with Images and Panel Turned Off



Button with Images, Panel Turned Off and Transparency Feature on Images Enabled

Another feature of the button object is to turn off the panel rendering of the object. This is useful when icons are used to define the buttons. Also to complete the feature set, the transparent color feature of images can also be enabled. This allows applications to use images with the flat background color ignored when rendering.

Note: Alpha blending of gradient fills is not supported by the object.

GFX_GOL_ButtonPressStateImageGet Macro

This function gets the image used when in the pressed state.

File

gfx_gol_button.h

C

```
#define GFX_GOL_ButtonPressStateImageGet(pObject, pImage) \
            (((GFX_GOL_BUTTON *)pObject)->pPressImage)
```

Returns

Pointer to the image resource.

Description

GFX GOL button press state image get.

This function gets the image used when in the pressed state.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_RESOURCE_HDR *GFX_GOL_ButtonPressStateImageGet(
GFX_GOL_BUTTON *pObject)

GFX_GOL_ButtonPressStateImageSet Macro

This function sets the image to be used when in the pressed state.

File

gfx_gol_button.h

C

```
#define GFX_GOL_ButtonPressStateImageSet(pObject, pImage) \
            (((GFX_GOL_BUTTON *)pObject)->pPressImage = pImage)
```

Returns

None.

Description

GFX GOL button press state image set.

This function sets the image to be used when in the pressed state.

Preconditions

Object must exist in memory.

Example

```
// assume ImageIcon is a valid GFX_RESOURCE_HDR

GFX_RESOURCE_HDR *pMyIcon = &ImageIcon;
GFX_GOL_BUTTON *pButton;

GFX_GOL_ButtonPressStateImageSet(pButton , pMyIcon);
```

Parameters

Parameters	Description
pObject	pointer to the object.
pImage	pointer to the image resource.

Function

void GFX_GOL_ButtonPressStateImageSet(
GFX_GOL_BUTTON *pObject,
GFX_RESOURCE_HDR *pImage)

GFX_GOL_ButtonReleaseStateImageGet Macro

This function gets the image used when in the released state.

File

gfx_gol_button.h

C

```
#define GFX_GOL_ButtonReleaseStateImageGet(pObject, pImage) \
            (((GFX_GOL_BUTTON *)pObject)->pReleaseImage)
```

Returns

Pointer to the image resource.

Description

GFX GOL button release state image get.

This function gets the image used when in the released state.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_RESOURCE_HDR *GFX_GOL_ButtonReleaseStateImageGet(
GFX_GOL_BUTTON *pObject)

GFX_GOL_ButtonReleaseStateImageSet Macro

This function sets the image to be used when in the released state.

File

gfx_gol_button.h

C

```
#define GFX_GOL_ButtonReleaseStateImageSet(pObject, pImage) \
            (((GFX_GOL_BUTTON *)pObject)->pReleaseImage = pImage)
```

Returns

None.

Description

GFX GOL button release state image set.

This function sets the image to be used when in the released state.

Preconditions

Object must exist in memory.

Example

```
// assume ImageIcon is a valid GFX_RESOURCE_HDR

GFX_RESOURCE_HDR *pMyIcon = &ImageIcon;
GFX_GOL_BUTTON *pButton;

GFX_GOL_ButtonReleaseStateImageSet(pButton , pMyIcon);
```

Parameters

Parameters	Description
pObject	pointer to the object.
pImage	pointer to the image resource.

Function

void GFX_GOL_ButtonReleaseStateImageSet(
GFX_GOL_BUTTON *pObject,
GFX_RESOURCE_HDR *pImage)

GFX_GOL_ButtonTextGet Macro

This function returns the address of the current text string used by the object.

File

gfx_gol_button.h

C

```
#define GFX_GOL_ButtonTextGet(pObject) (((GFX_GOL_BUTTON *)pObject)->pText)
```

Returns

Pointer to text string.

Description

GFX GOL button text get.

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Example

```
GFX_XCHAR *pChar;
GFX_GOL_BUTTON GFX_GOL_BUTTON[2];

pChar = GFX_GOL_ButtonTextGet(GFX_GOL_BUTTON[0]);
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_XCHAR *GFX_GOL_ButtonTextGet(
GFX_GOL_BUTTON *pObject)

GFX_GOL_ButtonActionSet Function

This function performs the state change of the object based on the translated action.

File

gfx_gol_button.h

C

```
void GFX_GOL_ButtonActionSet(GFX_GOL_TRANSLATED_ACTION translatedMsg, void * pObject, GFX_GOL_MESSAGE *
pMessage);
```

Returns

None.

Description

GFX GOL button action set.

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_BUTTON_ACTION_PRESSED	Touch Screen,	Set GFX_GOL_BUTTON_PRESSED_STATE	Button will be redrawn in the pressed state.
	Keyboard		
GFX_GOL_BUTTON_ACTION_RELEASED	Touch Screen,	Clear GFX_GOL_BUTTON_PRESSED_STATE	Button will be redrawn in the released state.
	Keyboard		
GFX_GOL_BUTTON_ACTION_CANCELPRESS	Touch Screen,	Clear GFX_GOL_BUTTON_PRESSED_STATE	Button will be redrawn in the released state.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Function

void GFX_GOL_ButtonActionSet(

GFX_GOL_TRANSLATED_ACTION translatedMsg,

void *pObject,

GFX_GOL_MESSAGE *pMessage)

GFX_GOL_ButtonActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_button.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_ButtonActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_BUTTON_ACTION_PRESSED - Object is pressed
- GFX_GOL_BUTTON_ACTION_RELEASED - Object is released
- GFX_GOL_BUTTON_ACTION_CANCELPRESS - Object will be released, user cancels press action on the GFX_GOL_BUTTON
- GFX_GOL_OBJECT_ACTION_INVALID - Object is not affected

Description

GFX GOL button action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_BUTTON_ACTION_PRESSED	Touch Screen	EVENT_PRESS, EVENT_MOVE	If events occurs and the x,y position falls in the face of the GFX_GOL_BUTTON while the GFX_GOL_BUTTON is not pressed.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_PRESSED or SCAN_SPACE_PRESSED while the GFX_GOL_BUTTON is not pressed.
GFX_GOL_BUTTON_ACTION_STILLPRESSED	Touch Screen	EVENT_STILLPRESS	If event occurs and the x,y position does not change from the previous press position in the face of the GFX_GOL_BUTTON.
GFX_GOL_BUTTON_ACTION_RELEASED	Touch Screen	EVENT_RELEASE	If the event occurs and the x,y position falls in the face of the GFX_GOL_BUTTON while the GFX_GOL_BUTTON is pressed.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_RELEASED or SCAN_SPACE_RELEASED while the GFX_GOL_BUTTON is pressed.
GFX_GOL_BUTTON_ACTION_CANCELPRESS	Touch Screen	EVENT_MOVE	If the event occurs outside the face of the GFX_GOL_BUTTON and the GFX_GOL_BUTTON is currently pressed.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_ButtonActionGet(
void *pObject,
GFX_GOL_MESSAGE *pMessage);

GFX_GOL_ButtonCreate Function

This function creates a GFX_GOL_BUTTON object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_button.h

C

```
GFX_GOL_BUTTON * GFX_GOL_ButtonCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left, uint16_t top,
uint16_t right, uint16_t bottom, uint16_t radius, uint16_t state, GFX_RESOURCE_HDR * pPressImage,
GFX_RESOURCE_HDR * pReleaseImage, GFX_XCHAR * pText, GFX_ALIGNMENT alignment, GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL button create

This function creates a GFX_GOL_BUTTON object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The object allows setting two images. One for the pressed state and the other for the release state. If no image is to be used for the object set both pointers to NULL.

If only one image is used for both pressed and released state, set both pPressImage and pReleaseImage to the same image.

The behavior of GFX_GOL_ButtonCreate() will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a GFX_GOL_OBJ_SCHEME
- pPressImage and pReleaseImage is not pointing to a GFX_RESOURCE_HDR.
- pText is an unterminated string

Preconditions

None.

Example

```
GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_BUTTON *buttons[3];
GFX_GOL_BUTTON_STATE state;

// assume pScheme is initialized to a scheme in memory.
state = GFX_GOL_BUTTON_DRAW_STATE;

buttons[0] = GFX_GOL_ButtonCreate(
        gfxIndex,
        1, 20, 64, 50, 118, 0,
        state, NULL, NULL, "ON",
        GFX_ALIGN_HCENTER | GFX_ALIGN_VCENTER,
        pScheme);
// check if GFX_GOL_BUTTON 0 is created
if (buttons[0] == NULL)
   return 0;

buttons[1] = GFX_GOL_ButtonCreate(
        gfxIndex,
        2, 52, 64, 82, 118, 0,
        state, NULL, NULL, "OFF",
        GFX_ALIGN_LEFT | GFX_ALIGN_VCENTER,
        pScheme);
// check if GFX_GOL_BUTTON 1 is created
if (buttons[1] == NULL)
   return 0;

buttons[2] = GFX_GOL_ButtonCreate(
        gfxIndex,
        3, 84, 64, 114, 118, 0,
        state, NULL, NULL, "HI",
        GFX_ALIGN_RIGHT | GFX_ALIGN_VCENTER,
        pScheme);
// check if GFX_GOL_BUTTON 2 is created
if (buttons[2] == NULL)
   return 0;

return 1;
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
radius	Radius of the rounded edge. When using gradient buttons and radius != 0, emboss size <= radius. If this is not met, the the GFX_GOL_BUTTON face will not have gradient effect.
state	Sets the initial state of the object.

pPressImage	Pointer to the image used on the face of the object when it is in the pressed state.
pReleaseImage	Pointer to the image used on the face of the object when it is in the pressed state.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Function

```
GFX_GOL_BUTTON  *GFX_GOL_ButtonCreate(
                    SYS_MODULE_INDEX gfxIndex,
uint16_t        ID,
uint16_t        left,
uint16_t        top,
uint16_t        right,
uint16_t        bottom,
uint16_t        radius,
uint16_t        state,
                    GFX_RESOURCE_HDR    *pPressImage,
                    GFX_RESOURCE_HDR    *pReleaseImage,
                    GFX_XCHAR           *pText,
                    GFX_ALIGNMENT       alignment,
                    GFX_GOL_OBJ_SCHEME  *pScheme)
```

GFX_GOL_ButtonDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_button.h

C

```
GFX_STATUS GFX_GOL_ButtonDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL button draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the face of the GFX_GOL_BUTTON is drawn on top of the bitmap. Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_ButtonDraw(void *pObject)

GFX_GOL_ButtonTextAlignmentGet Function

This function returns the text alignment of the text string used by the object.

File

gfx_gol_button.h

C

```
GFX_ALIGNMENT GFX_GOL_ButtonTextAlignmentGet(GFX_GOL_BUTTON * pObject);
```

Returns

The text alignment set in the object. See GFX_ALIGNMENT for more details.

Description

GFX GOL button text alignment get.

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_ALIGNMENT GFX_GOL_ButtonTextAlignmentGet(
GFX_GOL_BUTTON *pObject)

GFX_GOL_ButtonTextAlignmentSet Function

This function sets the text alignment of the text string used by the object.

File

gfx_gol_button.h

C

```
void GFX_GOL_ButtonTextAlignmentSet(GFX_GOL_BUTTON * pObject, GFX_ALIGNMENT align);
```

Returns

None.

Description

GFX GOL button text alignment set.

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Function

void GFX_GOL_ButtonTextAlignmentSet(
GFX_GOL_BUTTON *pObject,

GFX_ALIGNMENT align)

GFX_GOL_ButtonTextSet Function

This function sets the address of the current text string used by the object.

File

gfx_gol_button.h

C

```
void GFX_GOL_ButtonTextSet(GFX_GOL_BUTTON * pObject, GFX_XCHAR * pText);
```

Returns

None.

Description

GFX GOL button text set.

This function sets the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Example

```
GFX_XCHAR Label0[] = â€œONâ€?;
GFX_XCHAR Label1[] = â€œOFFâ€?;
GFX_GOL_BUTTON GFX_GOL_BUTTON[2];

GFX_GOL_ButtonTextSet(GFX_GOL_BUTTON[0], Label0);
GFX_GOL_ButtonTextSet(GFX_GOL_BUTTON[1], Label1);
```

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text string to be used.

Function

GFX_XCHAR *GFX_GOL_ButtonTextSet(
GFX_GOL_BUTTON *pObject,
GFX_XCHAR *pText)

Check Box Object

Check Box is an object that simulates a check box on paper. Usually it is used as an option setting where the checked or filled state means the option is enabled and the unfilled or unchecked state means the option is disabled.

Functions

	Name	Description
	GFX_GOL_CheckBoxActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_CheckBoxActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_CheckBoxCreate	This function creates a GFX_GOL_CHECKBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_CheckBoxDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_CheckBoxTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_CheckBoxTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_CheckBoxTextSet	This function sets the address of the current text string used by the object.

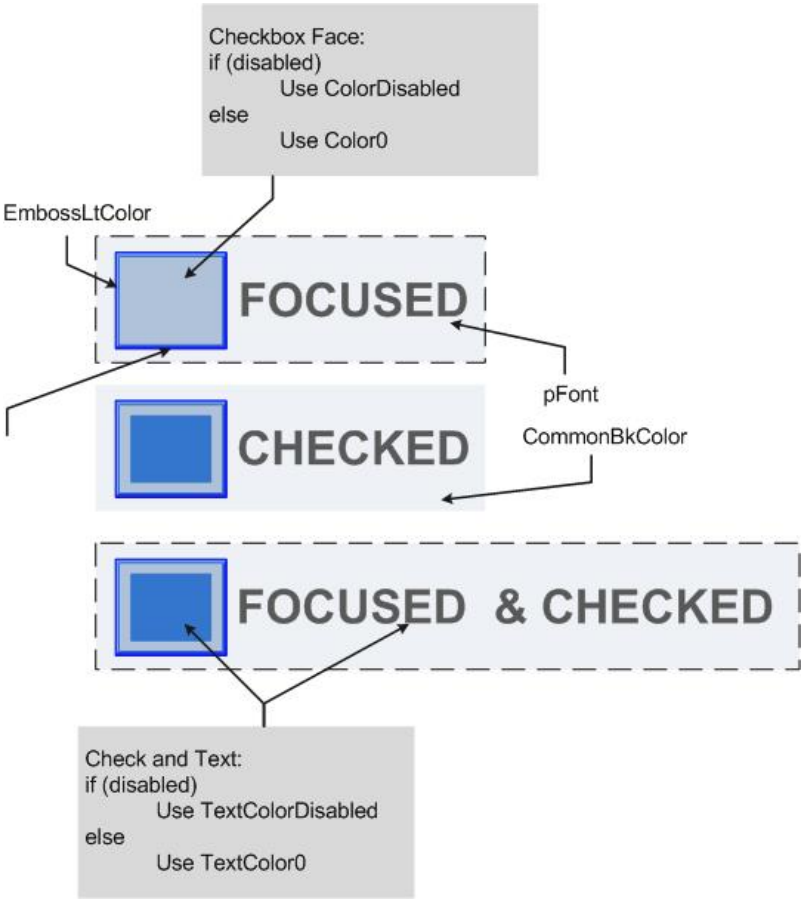
Macros

	Name	Description
	GFX_GOL_CheckBoxTextGet	This function returns the address of the current text string used by the object.

Description

Check Box supports Keyboard and Touchscreen inputs, replying to their events with the predefined actions (see GFX_GOL_CheckBoxActionGet() and GFX_GOL_CheckBoxActionSet() for details).

The Check Box object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



When creating the object, the alignment of the text of the object can be formatted with the same options that GFX_TextStringBoxDraw() allows.

GFX_GOL_CheckBoxTextGet Macro

This function returns the address of the current text string used by the object.

File

gfx_gol_check_box.h

C

```
#define GFX_GOL_CheckBoxTextGet(pObject) (((GFX_GOL_CHECKBOX *)pObject)->pText)
```

Returns

Pointer to text string.

Description

GFX GOL check box text get.

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Example

```
// assume CHECK_BOX_OBJECT is a radio button that exists
GFX_XCHAR *pChar;
GFX_GOL_CHECKBOX *pObject = &CHECK_BOX_OBJECT;

pChar = GFX_GOL_CheckBoxTextGet(pObject);
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_XCHAR *GFX_GOL_CheckBoxTextGet(
GFX_GOL_CHECKBOX *pObject)

GFX_GOL_CheckBoxActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_check_box.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_CheckBoxActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_CHECKBOX_ACTION_CHECKED - Check box is checked
- GFX_GOL_CHECKBOX_ACTION_UNCHECKED - Check box is unchecked
- GFX_GOL_OBJECT_ACTION_INVALID - Object is not affected

Description

GFX GOL checkbox action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_CHECKBOX_ACTION_CHECKED	Touch Screen	EVENT_RELEASE	If events occurs and the x,y position falls in the area of the check box while the check box is unchecked.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_PRESSED or SCAN_SPACE_PRESSED while the check box is unchecked.
GFX_GOL_CHECKBOX_ACTION_UNCHECKED	Touch Screen	EVENT_RELEASE	If events occurs and the x,y position falls in the area of the check box while the check box is checked.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_PRESSED or SCAN_SPACE_PRESSED while the check box is checked.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_CheckBoxActionGet(
void *pObject,
GFX_GOL_MESSAGE *pMessage);

GFX_GOL_CheckBoxActionSet Function

This function performs the state change of the object based on the translated action.

File

gfx_gol_check_box.h

C

```
void GFX_GOL_CheckBoxActionSet(GFX_GOL_TRANSLATED_ACTION translatedMsg, void * pObject, GFX_GOL_MESSAGE *
pMessage);
```

Returns

None.

Description

GFX GOL checkbox action set.

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_CHECKBOX_ACTION_CHECKED	Touch Screen,	Set GFX_GOL_CHECKBOX_CHECKED_STATE	Check Box will be redrawn in checked state.
	Keyboard		
GFX_GOL_CHECKBOX_ACTION_UNCHECKED	Touch Screen,	Clear GFX_GOL_CHECKBOX_CHECKED_STATE	Check Box will be redrawn in un-checked state.
	Keyboard		

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Function

void GFX_GOL_CheckBoxActionSet(
GFX_GOL_TRANSLATED_ACTION translatedMsg,
void *pObject,
GFX_GOL_MESSAGE *pMessage)

GFX_GOL_CheckBoxCreate Function

This function creates a GFX_GOL_CHECKBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_check_box.h

C

```
GFX_GOL_CHECKBOX * GFX_GOL_CheckBoxCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left, uint16_t
top, uint16_t right, uint16_t bottom, uint16_t state, GFX_XCHAR * pText, GFX_ALIGNMENT alignment,
GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL check box create.

This function creates a GFX_GOL_CHECKBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of GFX_GOL_CheckBoxCreate() will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a GFX_GOL_OBJ_SCHEME
- pText is an unterminated string

Preconditions

None.

Example

```
GFX_GOL_OBJ_SCHEME  *pScheme;
GFX_GOL_CHECKBOX    *pCb[2];

pCb[0] = GFX_GOL_CheckBoxCreate(
                    gfxIndex,
                    ID_CHECKBOX1,               // ID
                    20,135,150,175,             // dimension
                    GFX_GOL_CHECKBOX_DRAW_STATE, // Draw the object
                    "Scale",                    // text
                    GFX_ALIGN_CENTER,           // text alignment
                    pScheme);                   // use this scheme

pCb[1] = GFX_GOL_CheckBoxCreate(
                    gfxIndex,
                    ID_CHECKBOX2,               // ID
                    170,135,300,175,            // dimension
                    GFX_GOL_CHECKBOX_DRAW_STATE, // Draw the object
                    "Animate",                  // text
                    GFX_ALIGN_LEFT | GFX_ALIGN_VCENTER, // text alignment
                    pScheme);                   // use this scheme
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Function

```
GFX_GOL_CHECKBOX  *GFX_GOL_CheckBoxCreate(
                    SYS_MODULE_INDEX    gfxIndex,
uint16_t        ID,
uint16_t        left,
uint16_t        top,
uint16_t        right,
uint16_t        bottom,
```

uint16_t state,
GFX_XCHAR *pText,
GFX_ALIGNMENT alignment,
GFX_GOL_OBJ_SCHEME *pScheme)

GFX_GOL_CheckBoxDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_check_box.h

C

```
GFX_STATUS GFX_GOL_CheckBoxDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL checkbox draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the GFX_GOL_CHECKBOX is drawn with the text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_CheckBoxDraw(void *pObject)

GFX_GOL_CheckBoxTextAlignmentGet Function

This function returns the text alignment of the text string used by the object.

File

gfx_gol_check_box.h

C

```
GFX_ALIGNMENT GFX_GOL_CheckBoxTextAlignmentGet(GFX_GOL_CHECKBOX * pObject);
```

Returns

The text alignment set in the object. See GFX_ALIGNMENT for more details.

Description

GFX GOL check box text alignement get.

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_ALIGNMENT GFX_GOL_CheckBoxTextAlignmentGet(
GFX_GOL_CHECKBOX *pObject)

GFX_GOL_CheckBoxTextAlignmentSet Function

This function sets the text alignment of the text string used by the object.

File

gfx_gol_check_box.h

C

```
void GFX_GOL_CheckBoxTextAlignmentSet(GFX_GOL_CHECKBOX * pObject, GFX_ALIGNMENT align);
```

Returns

None.

Description

GFX GOL checkboc text alignment set.

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Function

void GFX_GOL_CheckBoxTextAlignmentSet(
GFX_GOL_CHECKBOX *pObject,
GFX_ALIGNMENT align)

GFX_GOL_CheckBoxTextSet Function

This function sets the address of the current text string used by the object.

File

gfx_gol_check_box.h

C

```
void GFX_GOL_CheckBoxTextSet(GFX_GOL_CHECKBOX * pObject, GFX_XCHAR * pText);
```

Returns

None.

Description

GFX OL check box text set.

This function sets the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Example

```
GFX_XCHAR Label0[] = â€œEnabledâ€?;
GFX_XCHAR Label1[] = â€œDisabledâ€?;
GFX_GOL_CHECKBOX GFX_GOL_CHECKBOX[2];

GFX_GOL_CheckBoxTextSet(GFX_GOL_CHECKBOX[0], Label0);
GFX_GOL_CheckBoxTextSet(GFX_GOL_CHECKBOX[1], Label1);
```

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text string to be used.

Function

GFX_XCHAR *GFX_GOL_CheckBoxTextSet(
GFX_GOL_CHECKBOX *pObject,
GFX_XCHAR *pText)

Custom Control Object

Functions

	Name	Description
	GFX_GOL_CustomControlActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_CustomControlActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_CustomControlCreate	This function creates a GFX_GOL_CUSTOMCONTROL object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_CustomControlDraw	This function renders the object on the screen based on the current state of the object.

Macros

	Name	Description
	GFX_GOL_CustomControlGetPos	This function returns the position of the control.
	GFX_GOL_CustomControlSetPos	This function sets the position of the control.

Description

GFX_GOL_CustomControlGetPos Macro

This function returns the position of the control.

File

gfx_gol_custom_control.h

C

```
#define GFX_GOL_CustomControlGetPos(pCc) pCc->pos
```

Returns

The position of the object.

Description

GFX GOL custom control get position

This function returns the position of the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_ALIGNMENT GFX_GOL_CustomControlGetPos(
GFX_GOL_CUSTOMCONTROL *pObject)

GFX_GOL_CustomControlSetPos Macro

This function sets the position of the control.

File

gfx_gol_custom_control.h

C

```
#define GFX_GOL_CustomControlSetPos(pCc, position) pCc->pos = position
```

Returns

none

Description

GFX GOL custom control set position.

This function sets the position of the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_CustomControlSetPos(
GFX_GOL_CUSTOMCONTROL *pObject, uint16_t pos)

GFX_GOL_CustomControlActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_custom_control.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_CustomControlActionGet(void * pObj, GFX_GOL_MESSAGE * pMsg);
```

Returns

- GFX_GOL_CUSTOMCONTROL_ACTION_SELECTED - Object is selected.
- GFX_GOL_OBJECT_ACTION_INVALID - Object is not affected

Description

GFX GOL custom control action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_CUSTOMCONTROL_ACTION_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE	If events occurs and the x,y position falls in the area of the Digital Meter.

GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_CustomControlActionGet(
void *pObject,
GFX_GOL_MESSAGE *pMessage)

GFX_GOL_CustomControlActionSet Function

This function performs the state change of the object based on the translated action.

File

gfx_gol_custom_control.h

C

```
void GFX_GOL_CustomControlActionSet(GFX_GOL_TRANSLATED_ACTION translatedMsg, void * pObj, GFX_GOL_MESSAGE * pMsg);
```

Returns

None.

Description

GFX GOL custom control action set.

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source Set/Clear State Bit Description
GFX_GOL_CUSTOMCONTROL_ACTION_SELECTED	Touch Screen, Set GFX_GOL_RADIOBUTTON_DRAW_STATE, Depending on the current value of RB_CHECKED Check Box will be redrawn.
	Keyboard Set/Clear GFX_GOL_RADIOBUTTON_CHECKED_STATE

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Function

void GFX_GOL_CustomControlActionSet(
GFX_GOL_TRANSLATED_ACTION translatedMsg,
void *pObject,

GFX_GOL_MESSAGE *pMessage)

GFX_GOL_CustomControlCreate Function

This function creates a GFX_GOL_CUSTOMCONTROL object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_custom_control.h

C

```
GFX_GOL_CUSTOMCONTROL * GFX_GOL_CustomControlCreate(SYS_MODULE_INDEX clientIndex, uint16_t ID, uint16_t
left, uint16_t top, uint16_t right, uint16_t bottom, uint16_t state, GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL custom control create.

This function creates a GFX_GOL_CUSTOMCONTROL object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of GFX_GOL_CustomControlCreate() will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a GFX_GOL_OBJ_SCHEME
- pText is an unterminated string

Preconditions

None.

Example

```
GFX_GOL_OBJ_SCHEME  *pScheme;
GFX_GOL_CUSTOMCONTROL    *pCc[2];

pCc[0] = GFX_GOL_CustomControlCreate(
                    GFX_INDEX_0,
                    ID_CUSTOM1, // ID
                    30,
                    0,
                    150,
                    120, // dimension
                    // will be displayed after creation
                    GFX_GOL_CUSTOMCONTROL_DRAW_STATE,
                    pScheme
 );
```

Parameters

Parameters	Description
clientIndex	client user instance
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Function

```
GFX_GOL_CUSTOMCONTROL *GFX_GOL_CustomControlCreate(
        SYS_MODULE_INDEX clientIndex,
```

uint16_t ID,
uint16_t left,
uint16_t top,
uint16_t right,
uint16_t bottom,
uint16_t state,
GFX_GOL_OBJ_SCHEME * pScheme)

GFX_GOL_CustomControlDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_custom_control.h

C

```
GFX_STATUS GFX_GOL_CustomControlDraw(void * pObj);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL custom control draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_CustomControlDraw(void *pObject)

Digital Meter Object

Digital Meter is an object that can be used to display a value of a sampled variable. This object is ideal when fast refresh of the value is needed. The object refreshes only the digits that needs to change.

Functions

	Name	Description
	GFX_GOL_DigitalMeterActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_DigitalMeterCreate	This function creates a GFX_GOL_DIGITALMETER object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_DigitalMeterDecrement	This function decrements the meter value by the delta value set.
	GFX_GOL_DigitalMeterDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_DigitalMeterIncrement	This function increments the meter value by the delta value set.
	GFX_GOL_DigitalMeterValueSet	This function sets the value of the meter.

Macros

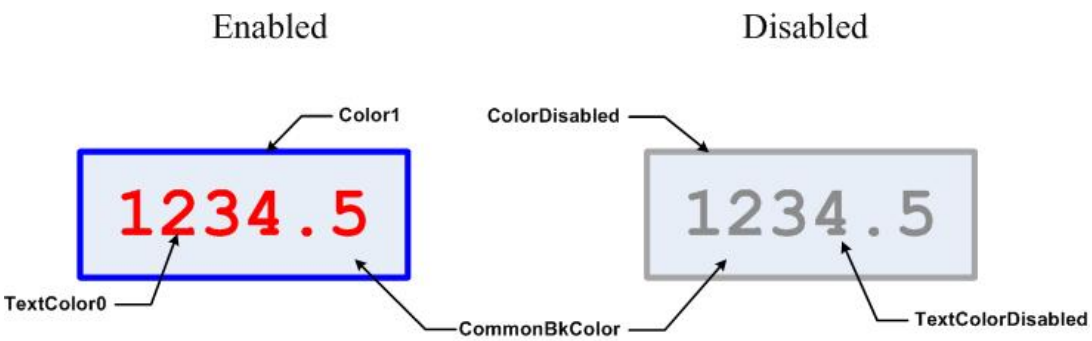
	Name	Description
	GFX_GOL_DigitalMeterTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_DigitalMeterTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_DigitalMeterValueGet	This function returns the current value of the digital meter.

Description

Digital Meter is an object that can be used to display a value of a sampled variable. This object is ideal when fast refresh of the value is needed. The object refreshes only the digits that needs to change.

Digital Meter supports only Touchscreen inputs, replying to the events with the predefined actions (see GFX_GOL_DigitalMeterActionGet() for details). There is no default action set function in this object. Application can create specific responses to the system action in the message callback function.

The DigitalMeter object is rendered using the assigned style scheme. The following figure illustrates the color assignments for the digital meter.



GFX_GOL_DigitalMeterTextAlignmentGet Macro

This function returns the text alignment of the text string used by the object.

File

gfx_gol_digital_meter.h

C

```
#define GFX_GOL_DigitalMeterTextAlignmentGet(pObject) \
                (((GFX_GOL_DIGITALMETER *)pObject)->alignment)
```

Returns

The text alignment set in the object. See GFX_ALIGNMENT for more details.

Description

GFX GOL digital meter text alignment get.

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_ALIGNMENT GFX_GOL_DigitalMeterTextAlignmentGet(
GFX_GOL_DIGITALMETER *pObject)

GFX_GOL_DigitalMeterTextAlignmentSet Macro

This function sets the text alignment of the text string used by the object.

File

gfx_gol_digital_meter.h

C

```
#define GFX_GOL_DigitalMeterTextAlignmentSet(pObject, align) \
                (((GFX_GOL_DIGITALMETER *)pObject)->alignment = align)
```

Returns

None.

Description

GFX GOL digital meter text alignment set.

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Function

void GFX_GOL_DigitalMeterTextAlignmentSet(

GFX_GOL_DIGITALMETER *pObject,

GFX_ALIGNMENT align)

GFX_GOL_DigitalMeterValueGet Macro

This function returns the current value of the digital meter.

File

gfx_gol_digital_meter.h

C

```
#define GFX_GOL_DigitalMeterValueGet(pObject) \
                                (((GFX_GOL_DIGITALMETER*)pObject)->value)
```

Returns

The current value of the object.

Description

GFX GOL digital meter value get.

This function returns the current value of the digital meter.

Preconditions

Object must exist in memory.

Example

```
#define MAXVALUE 100;

GFX_GOL_DIGITALMETER *pMeter;
uint32_t ctr = 0;

// create scroll bar here and initialize parameters
pMeter = GFX_GOL_DigitalMeterCreate(....)
GFX_GOL_ObjectStateSet(pMeter, GFX_GOL_DIGITALMETER_DRAW_STATE);

// draw the scroll bar
GFX_GOL_ObjectListDraw(gfxIndex);
```

```
// a routine that updates the position of the thumb through some
// conditions
while("some condition")
{
    GFX_GOL_DigitalMeterValueSet(pMeter, ctr);
    GFX_GOL_ObjectStateSet( pMeter,
                            GFX_GOL_DIGITALMETER_UPDATE_STATE);

    // update the screen
    GFX_GOL_ObjectListDraw(gfxIndex);

    // update ctr here
    ctr = "some source of value";
}

if (GFX_GOL_DigitalMeterValueGet(pScrollBar) > MAXVALUE)
    return 0;
else
    "do something else"
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

uint16_t GFX_GOL_DigitalMeterValueGet(
GFX_GOL_DIGITALMETER *pObject)

GFX_GOL_DigitalMeterActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_digital_meter.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_DigitalMeterActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_DIGITALMETER_ACTION_SELECTED - Object is selected.
- GFX_GOL_OBJECT_ACTION_INVALID - Object is not affected

Description

GFX GOL Digital meter action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_DIGITALMETER_ACTION_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE	If events occurs and the x,y position falls in the area of the Digital Meter.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_DigitalMeterActionGet(
void *pObject,
GFX_GOL_MESSAGE *pMessage)

GFX_GOL_DigitalMeterCreate Function

This function creates a GFX_GOL_DIGITALMETER object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_digital_meter.h

C

```
GFX_GOL_DIGITALMETER * GFX_GOL_DigitalMeterCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left,
uint16_t top, uint16_t right, uint16_t bottom, uint16_t state, uint32_t value, uint8_t NoOfDigits, uint8_t
DotPos, GFX_ALIGNMENT alignment, GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL digital meter create.

This function creates a GFX_GOL_DIGITALMETER object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

Preconditions

None.

Example

```
GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_DIGITALMETER *pDm;

pScheme = GFX_GOL_SchemeCreate();
state = GFX_GOL_DIGITALMETER_DRAW_STATE |
        GFX_GOL_DIGITALMETER_FRAME_STATE;
GFX_GOL_DigitalMeterCreate(
        gfxIndex,
        ID_DIGITALMETER1,   // ID
        30,80,235,160,      // dimension
        state,              // has frame and center aligned
        789,4,1,            // to display 078.9
        GFX_ALIGN_CENTER,
        pScheme);           // use given scheme

// draw the objects
while(GFX_GOL_ObjectListDraw(gfxIndex) != GFX_STATUS_SUCCESS);
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
instance	Device instance
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.

bottom	Bottom most position of the object.
state	Sets the initial state of the object.
value	Sets the initial value to be displayed
NoOfDigits	Sets the number of digits to be displayed
DotPos	Sets the position of decimal point in the display
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme. Set to NULL if default style scheme is used.

Function

GFX_GOL_DIGITALMETER *GFX_GOL_DigitalMeterCreate(
SYS_MODULE_INDEX gfxIndex,
uint16_t ID,
uint16_t left,
uint16_t top,
uint16_t right,
uint16_t bottom,
uint16_t state,
uint32_t value,
uint8_t NoOfDigits,
uint8_t DotPos,
GFX_ALIGNMENT alignment
GFX_GOL_OBJ_SCHEME *pScheme)

GFX_GOL_DigitalMeterDecrement Function

This function decrements the meter value by the delta value set.

File

gfx_gol_digital_meter.h

C

```
void GFX_GOL_DigitalMeterDecrement(GFX_GOL_DIGITALMETER * pObject, uint16_t delta);
```

Returns

None.

Description

GFX GOL digital meter decrement.

This function decrements the meter value by the given delta value set. If the delta given is less than the minimum value of the meter, the value will remain to be at minimum.

Object must be redrawn after this function is called to reflect the changes to the object.

Preconditions

Object must exist in memory.

Example

See GFX_GOL_DigitalMeterIncrement().

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_DigitalMeterDecrement(
GFX_GOL_DIGITALMETER *pObject,
uint16_t delta)

GFX_GOL_DigitalMeterDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_digital_meter.h

C

```
GFX_STATUS GFX_GOL_DigitalMeterDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL digital meter draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_DigitalMeterDraw(void *pObject)

GFX_GOL_DigitalMeterIncrement Function

This function increments the meter value by the delta value set.

File

gfx_gol_digital_meter.h

C

```
void GFX_GOL_DigitalMeterIncrement(GFX_GOL_DIGITALMETER * pObject, uint16_t delta);
```

Returns

None.

Description

GFX GOL digital meter

This function increments the scroll bar position by the given delta value set. If the delta given exceeds the maximum value of the meter, the value will remain to be at maximum.

Object must be redrawn after this function is called to reflect the changes to the object.

Preconditions

Object must exist in memory.

Example

```
void ControlSpeed(  GFX_GOL_DIGITALMETER* pObj,
                    int setSpeed,
                    int curSpeed)
{
    // set page size to 1
    GFX_GOL_DigitalMeterValueSet(pObj, 1);

    if (setSpeed < curSpeed)
```

```
        {
            while(GFX_GOL_DigitalMeterValueGet(pObj) < SetSpeed)
            {
                // increment by 1
                GFX_GOL_DigitalMeterIncrement(pObj, 1);
            }
        }
        else if (setSpeed > curSpeed)
        {
            while(GFX_GOL_DigitalMeterValueGet(pObj) > SetSpeed)
            {
                // decrement by 1
                GFX_GOL_DigitalMeterDecrement(pObj, 1);
            }
        }
    }
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_DigitalMeterIncrement(
GFX_GOL_DIGITALMETER *pObject,
uint16_t delta)

GFX_GOL_DigitalMeterValueSet Function

This function sets the value of the meter.

File

gfx_gol_digital_meter.h

C

```
void GFX_GOL_DigitalMeterValueSet(GFX_GOL_DIGITALMETER * pObject, int16_t value);
```

Returns

None.

Description

GFX GOL digital meter value set.

This function sets the value of the meter. The value used should be within the range set for the object. The new value is checked to be in the minimum and maximum value range. If the value set is less than the minimum value, the value that will be set is the minimum value. If the value set is more than the maximum value, then the value that will be set is the maximum value.

Preconditions

Object must exist in memory.

Example

```
GFX_GOL_DIGITALMETER *pMeter;
uint16_t ctr = 0;

// create slider here and initialize parameters
GFX_GOL_ObjectStateSet(pMeter, GFX_GOL_DIGITALMETER_DRAW_STATE);
GFX_GOL_ObjectListDraw(gfxIndex);

while("some condition")
{
    GFX_GOL_DigitalMeterValueSet(pMeter, ctr);
    GFX_GOL_ObjectStateSet( pMeter,
                            GFX_GOL_DIGITALMETER_UPDATE_STATE);

    // redraw the scroll bar
    GFX_GOL_ObjectListDraw(gfxIndex);

   // update ctr here
```

```
        ctr = "some source of value";
    }
```

Parameters

Parameters	Description
pObject	pointer to the object.
value	the new value of the object.

Function

void GFX_GOL_DigitalMeterValueSet(
GFX_GOL_DIGITALMETER *pObject,
int16_t value)

Edit Box Object

Edit Box is an object that emulates a cell or a text area that can be edited dynamically.

Functions

	Name	Description
	GFX_GOL_EditBoxActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_EditBoxActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_EditBoxCharAdd	This function adds a character at the end of the text used by the object.
	GFX_GOL_EditBoxCharRemove	This function removes a character at the end of the text used by the object.
	GFX_GOL_EditBoxCreate	This function creates a GFX_GOL_EDITBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_EditBoxDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_EditBoxTextSet	This function sets the text in the object.
	GFX_GOL_EditBoxCursorPositionDecrement	Decrement cursor position.
	GFX_GOL_EditBoxCursorPositionIncrement	Increment cursor position.

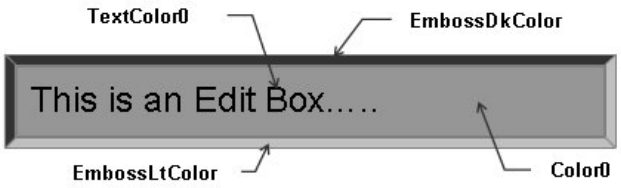
Macros

	Name	Description
	GFX_GOL_EditBoxTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_EditBoxTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_EditBoxTextGet	This function returns the address of the current text string used by the object.

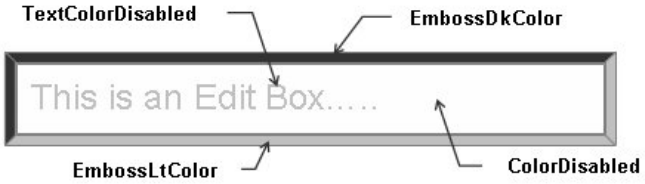
Description

Edit Box supports only keyboard inputs, replying to the events with the predefined actions (see GFX_GOL_EditBoxActionGet() and GFX_GOL_EditBoxActionSet() for details).

The Edit Box object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



Edit Box in enabled state



Edit Box in disabled state

GFX_GOL_EditBoxTextAlignmentGet Macro

This function returns the text alignment of the text string used by the object.

File

gfx_gol_edit_box.h

C

```
#define GFX_GOL_EditBoxTextAlignmentGet(pObject) \
                (((GFX_GOL_EDITBOX *)pObject)->alignment)
```

Returns

The text alignment set in the object. See GFX_ALIGNMENT for more details.

Description

GFX GOL edit box text alignment get.

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_ALIGNMENT GFX_GOL_EditBoxTextAlignmentGet(
GFX_GOL_EDITBOX *pObject)

GFX_GOL_EditBoxTextAlignmentSet Macro

This function sets the text alignment of the text string used by the object.

File

gfx_gol_edit_box.h

C

```
#define GFX_GOL_EditBoxTextAlignmentSet(pObject, align) \
                (((GFX_GOL_EDITBOX *)pObject)->alignment = align)
```

Returns

None.

Description

GFX GOL edit box text alignment set.

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Function

void GFX_GOL_EditBoxTextAlignmentSet(
GFX_GOL_EDITBOX *pObject,
GFX_ALIGNMENT align)

GFX_GOL_EditBoxTextGet Macro

This function returns the address of the current text string used by the object.

File

gfx_gol_edit_box.h

C

```
#define GFX_GOL_EditBoxTextGet(pObject) (((GFX_GOL_EDITBOX *)pObject)->pText)
```

Returns

Pointer to text string.

Description

GFX GOL edit box text get.

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_XCHAR *GFX_GOL_EditBoxTextGet(
GFX_GOL_EDITBOX *pObject)

GFX_GOL_EditBoxActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_edit_box.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_EditBoxActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_EDITBOX_ACTION_ADD_CHAR - New character should be added
- GFX_GOL_EDITBOX_ACTION_DEL_CHAR - Last character should be removed.
- GFX_GOL_EDITBOX_ACTION_TOUCHSCREEN - focus will be drawn when enabled. Caret will be drawn when enabled.
- GFX_GOL_OBJECT_ACTION_INVALID - Object is not affected

Description

GFX GOL edit box action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_EDITBOX_ACTION_ADD_CHAR	Keyboard	EVENT_CHARCODE	New character should be added.
GFX_GOL_EDITBOX_ACTION_DEL_CHAR	Keyboard	EVENT_KEYPRESS	Last character should be removed.
GFX_GOL_EDITBOX_ACTION_TOUCHSCREEN	Touch Screen	GFX_GOL_EDITBOX_FOCUSED_STATE	Focus will be drawn, caret displayed when enabled.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_EditBoxActionGet(

void *pObject,

GFX_GOL_MESSAGE *pMessage);

GFX_GOL_EditBoxActionSet Function

This function performs the state change of the object based on the translated action.

File

gfx_gol_edit_box.h

C

```
void GFX_GOL_EditBoxActionSet(GFX_GOL_TRANSLATED_ACTION translatedMsg, void * pObject, GFX_GOL_MESSAGE *
pMessage);
```

Returns

None.

Description

GFX GOL edit box action set.

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the

application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_EDITBOX_ACTION_ADD_CHAR	Keyboard	Set GFX_GOL_EDITBOX_DRAW_STATE	New character is added and Edit Box will be redrawn.
GFX_GOL_EDITBOX_ACTION_DEL_CHAR	KeyBoard	Set GFX_GOL_EDITBOX_DRAW_STATE	Last character is removed and Edit Box will be redrawn.
GFX_GOL_EDITBOX_ACTION_TOUCHSCREEN	Touch Screen	Set GFX_GOL_EDITBOX_FOCUSED_STATE	When enabled, focus will be redrawn, caret will also be redrawn if enabled.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Function

void GFX_GOL_EditBoxActionSet(
GFX_GOL_TRANSLATED_ACTION translatedMsg,
void *pObject,
GFX_GOL_MESSAGE *pMessage)

GFX_GOL_EditBoxCharAdd Function

This function adds a character at the end of the text used by the object.

File

gfx_gol_edit_box.h

C

```
GFX_STATUS GFX_GOL_EditBoxCharAdd(GFX_GOL_EDITBOX * pObject, GFX_XCHAR ch);
```

Returns

The status of the addition.

- GFX_STATUS_SUCCESS - when the addition was successful.
- GFX_STATUS_FAILURE - when the addition was not successful.

Description

GFX GOL edit box char add.

This function adds a character at the end of the text used by the object. When the object contains the maximum number of characters any addition call to this function will not affect the text in the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.
ch	The character to be added.

Function

GFX_STATUS GFX_GOL_EditBoxCharAdd(
GFX_GOL_EDITBOX *pObject,
GFX_XCHAR ch)

GFX_GOL_EditBoxCharRemove Function

This function removes a character at the end of the text used by the object.

File

gfx_gol_edit_box.h

C

```
GFX_STATUS GFX_GOL_EditBoxCharRemove(GFX_GOL_EDITBOX * pObject);
```

Returns

The status of the addition.

- GFX_STATUS_SUCCESS - when the removal was successful.
- GFX_STATUS_FAILURE - when the removal has no effect and the buffer is empty.

Description

GFX GOL edit box char remove.

This function removes a character at the end of the text used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_EditBoxCharRemove(
GFX_GOL_EDITBOX *pObject)

GFX_GOL_EditBoxCreate Function

This function creates a GFX_GOL_EDITBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_edit_box.h

C

```
GFX_GOL_EDITBOX * GFX_GOL_EditBoxCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left, uint16_t
top, uint16_t right, uint16_t bottom, uint16_t state, GFX_XCHAR * pText, uint16_t charMax, GFX_ALIGNMENT
alignment, GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL edit box create.

This function creates a GFX_GOL_EDITBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of GFX_GOL_ListBoxCreate() will be undefined if one of the following is true:

- left >= right
- top >= bottom

- pScheme is not pointing to a GFX_GOL_OBJ_SCHEME
- pText is an unterminated string

Preconditions

None.

Example

```
// assume pScheme to be initialized with the style scheme

#define MAX_CHAR_COUNT 15

GFX_GOL_OBJ_SCHEME      *pScheme;
GFX_GOL_EDITBOX         *pEb;
GFX_GOL_EDITBOX_STATE   state;

GFX_XCHAR               TextBuffer[MAX_CHAR_COUNT + 1];

// assume pScheme is initialized to a scheme in memory.
state = GFX_GOL_EDITBOX_DRAW_STATE;

pEb = GFX_GOL_EditBoxCreate(
        1, 20, 64, 50, 118,
        state, TextBuffer, MAX_CHAR_COUNT
        GFX_ALIGN_CENTER,
        pScheme);

// check if object is not created
if (pEb == NULL)
   return 0;
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pText	Pointer to the text of the object.
charMax	Defines the maximum number of characters in the edit box.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Function

```
GFX_GOL_EDITBOX  *GFX_GOL_EditBoxCreate(
                    SYS_MODULE_INDEX    gfxIndex,
uint16_t            ID,
uint16_t            left,
uint16_t            top,
uint16_t            right,
uint16_t            bottom,
uint16_t            state,
                    GFX_XCHAR           *pText,
uint16_t            charMax,
                    GFX_ALIGNMENT       alignment,
                    GFX_GOL_OBJ_SCHEME  *pScheme)
```

GFX_GOL_EditBoxDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_edit_box.h

C

```
GFX_STATUS GFX_GOL_EditBoxDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL edit box draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_EditBoxDraw(void *pObject)

GFX_GOL_EditBoxTextSet Function

This function sets the text in the object.

File

gfx_gol_edit_box.h

C

```
void GFX_GOL_EditBoxTextSet(GFX_GOL_EDITBOX * pObject, GFX_XCHAR* pText);
```

Returns

None.

Description

GFX GOL edit box text set.

This function sets the text in the object. This function copies the text located in the address pointed to by pText to the object text buffer.

The string length must be less than or equal to the maximum characters allowed in the object. The object will truncate the string once it reaches the maximum length.

Preconditions

Object must exist in memory.

Example

```
#define MAX_EDITBOX_CHAR_LENGTH 20
GFX_XCHAR Label0[] = "This is really a long string";

// assume pEb is a pointer initialized to an edit box that
// as a buffer with 20 characters allowed.

GFX_GOL_EditBoxTextSet(pEb, Label0);
```

```
// at this point the edit box contains
// This is really a lon" - truncated to only 20 characters.
```

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text string to be copied.

Function

GFX_XCHAR *GFX_GOL_EditBoxTextSet(
GFX_GOL_EDITBOX *pObject,
GFX_XCHAR *pText)

GFX_GOL_EditBoxCursorPositionDecrement Function

Decrement cursor position.

File

gfx_gol_edit_box.h

C

```
GFX_STATUS GFX_GOL_EditBoxCursorPositionDecrement(GFX_GOL_EDITBOX * pObject);
```

Returns

The status of the addition.

- GFX_STATUS_SUCCESS - Cursor position decrement successful.
- GFX_STATUS_FAILURE - Cursor position decrement failed.

Description

GFX GOL edit box cursor position decrement.

This function decrements cursor position.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_EditBoxCursorPositionDecrement(
GFX_GOL_EDITBOX *pObject)

GFX_GOL_EditBoxCursorPositionIncrement Function

Increment cursor position.

File

gfx_gol_edit_box.h

C

```
GFX_STATUS GFX_GOL_EditBoxCursorPositionIncrement(GFX_GOL_EDITBOX * pObject);
```

Returns

The status of the addition.

- GFX_STATUS_SUCCESS - Cursor position increment successful.
- GFX_STATUS_FAILURE - Cursor position increment failed.

Description

GFX GOL edit box cursor position increment.

This function increments cursor position.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_EditBoxCursorPostionIncrement(
GFX_GOL_EDITBOX *pObject)

Group Box Object

Group Box is an object that can be used to group objects together in the screen.

Functions

	Name	Description
	GFX_GOL_GroupboxActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_GroupboxDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_GroupboxCreate	This function creates a GFX_GOL_GROUPBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_GroupboxTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_GroupboxTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_GroupboxTextSet	This function sets the address of the current text string used by the object.

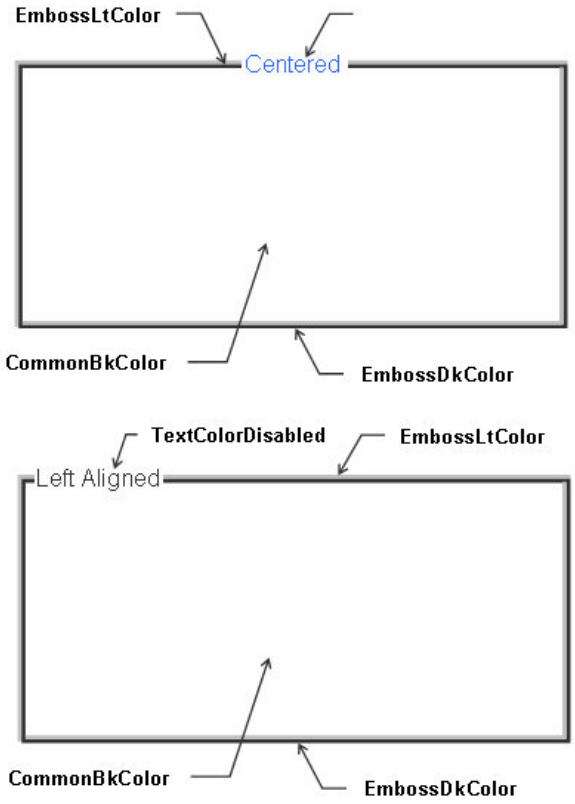
Macros

	Name	Description
	GFX_GOL_GroupboxTextGet	This function returns the address of the current text string used by the object.

Description

Group Box supports only Touchscreen inputs, replying to the events with the predefined actions (see GFX_GOL_GroupBoxActionGet() for details). There is no default action set function in this object. Application can create specific responses to the system action in the message callback function.

The Group Box object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



GFX_GOL_GroupboxTextGet Macro

This function returns the address of the current text string used by the object.

File

gfx_gol_group_box.h

C

```
#define GFX_GOL_GroupboxTextGet(pObject) (((GFX_GOL_GROUPBOX *)pObject)->pText)
```

Returns

Pointer to text string.

Description

GFX GOL group box text get.

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Example

```
GFX_XCHAR *pChar;
GFX_GOL_GROUPBOX pGroupbox;

pChar = GFX_GOL_GroupboxTextGet(pGroupbox);
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_XCHAR *GFX_GOL_GroupboxTextGet(
GFX_GOL_GROUPBOX *pObject)

GFX_GOL_GroupboxActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_group_box.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_GroupboxActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_GROUPBOX_ACTION_SELECTED - Group Box area selected action ID.
- GFX_GOL_OBJECT_ACTION_INVALID â€“ Object is not affected

Description

GFX GOL group box action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Events	Description
GFX_GOL_GROUPBOX_ACTION_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE	If events occurs and the x,y position falls in the area of the group box.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_GroupboxActionGet(

void *pObject,

GFX_GOL_MESSAGE *pMessage);

GFX_GOL_GroupboxDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_group_box.h

C

```
GFX_STATUS GFX_GOL_GroupboxDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL group box draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the face of the GFX_GOL_GROUPBOX is drawn on top of the bitmap. Text alignment based on the alignment parameter set on the

object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_GroupboxDraw(void *pObject)

GFX_GOL_GroupboxCreate Function

This function creates a GFX_GOL_GROUPBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_group_box.h

C

```
GFX_GOL_GROUPBOX * GFX_GOL_GroupboxCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left, uint16_t
top, uint16_t right, uint16_t bottom, uint16_t state, GFX_XCHAR * pText, GFX_ALIGNMENT alignment,
GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL group box create.

This function creates a GFX_GOL_GROUPBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

Preconditions

None.

Example

```
GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_GROUPBOX *pGroupbox;
GFX_GOL_GROUPBOX_STATE state;

// assume pScheme is initialized to a scheme in memory.
state = GFX_GOL_GROUPBOX_DRAW_STATE;

pGroupbox  = GFX_GOL_GroupboxCreate(
       gfxIndex,
       1,                              // ID
       0, 0,
       GFX_Primitive_MaxXGet(),
       GFX_Primitive_MaxYGet(),        // whole screen dimension
       state,                          // set state to draw all
       "Place Title Here.",            // text
       GFX_ALIGN_VCENTER,              // alignment of text
       NULL);                          // use default GOL scheme

if (pGroupbox == NULL)
    return 0;
```

```
    return 1;
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pImage	Pointer to the image used on the face of the object.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Function

```
    GFX_GOL_GROUPBOX  *GFX_GOL_GroupboxCreate(
                        SYS_MODULE_INDEX    gfxIndex,
uint16_t            ID,
uint16_t            left,
uint16_t            top,
uint16_t            right,
uint16_t            bottom,
uint16_t            state,
                        GFX_RESOURCE_HDR    *pImage,
                        GFX_XCHAR           *pText,
                        GFX_ALIGNMENT       alignment,
                        GFX_GOL_OBJ_SCHEME  *pScheme)
```

GFX_GOL_GroupboxTextAlignmentGet Function

This function returns the text alignment of the text string used by the object.

File

gfx_gol_group_box.h

C

```
GFX_ALIGNMENT GFX_GOL_GroupboxTextAlignmentGet(GFX_GOL_GROUPBOX * pObject);
```

Returns

The text alignment set in the object. See GFX_ALIGNMENT for more details.

Description

GFX GOL group box text alignment get.

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

```
    GFX_ALIGNMENT GFX_GOL_GroupboxTextAlignmentGet(
```

GFX_GOL_GROUPBOX *pObject)

GFX_GOL_GroupboxTextAlignmentSet Function

This function sets the text alignment of the text string used by the object.

File

gfx_gol_group_box.h

C

```
void GFX_GOL_GroupboxTextAlignmentSet(GFX_GOL_GROUPBOX * pObject, GFX_ALIGNMENT align);
```

Returns

None.

Description

GFX GOL group box text alignment set.

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Function

void GFX_GOL_GroupboxTextAlignmentSet(
GFX_GOL_GROUPBOX *pObject,
GFX_ALIGNMENT align)

GFX_GOL_GroupboxTextSet Function

This function sets the address of the current text string used by the object.

File

gfx_gol_group_box.h

C

```
void GFX_GOL_GroupboxTextSet(GFX_GOL_GROUPBOX * pObject, GFX_XCHAR * pText);
```

Returns

None.

Description

GFX GOL group box text set.

This function sets the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Example

```
GFX_XCHAR Label0[] = ?Group One?;
GFX_XCHAR Label1[] = ?Group Two?;
GFX_GOL_GROUPBOX pGroupbox;

GFX_GOL_GroupboxTextSet(pGroupbox, Label0);
GFX_GOL_GroupboxTextSet(pGroupbox, Label1);
```

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text string to be used.

Function

```
GFX_XCHAR *GFX_GOL_GroupboxTextSet(
                GFX_GOL_GROUPBOX *pObject,
                GFX_XCHAR *pText)
```

List Box Object

List Box is an object that defines a scrollable area where items are listed. User can select a single item or multiple of items.

Functions

	Name	Description
	GFX_GOL_ListBoxActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_ListBoxActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_ListBoxDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_ListBoxItemAdd	This function adds an item to the list box.
	GFX_GOL_ListBoxItemFocusGet	This function returns the index of the focused item in the list box.
	GFX_GOL_ListBoxItemFocusSet	This function sets the focus of the item with the index number specified by index.
	GFX_GOL_ListBoxItemListRemove	This function removes the entire items list of the list box and free the memory used.
	GFX_GOL_ListBoxItemRemove	This function removes an item from the list box and free the memory used.
	GFX_GOL_ListBoxSelectionChange	This function changes the selection status of the given item in the list box.
	GFX_GOL_ListBoxSelectionGet	This function searches for selected items from the list box.
	GFX_GOL_ListBoxVisibleItemCountGet	This function returns the count of items visible in the list box.
	GFX_GOL_ListBoxCreate	This function creates a GFX_GOL_LISTBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

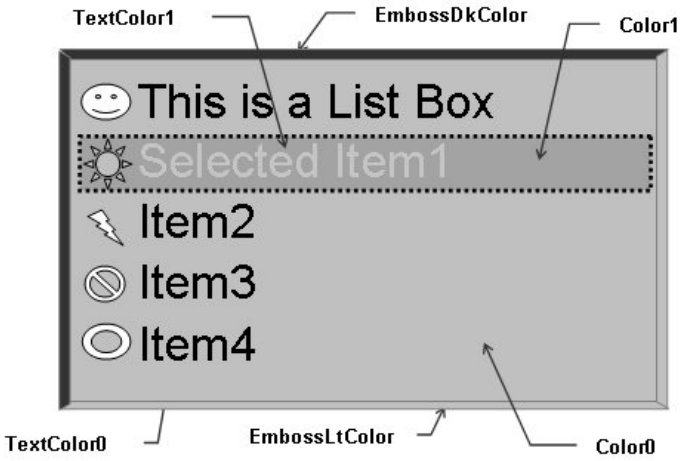
Macros

	Name	Description
	GFX_GOL_ListBoxItemCountGet	This function returns the count of items in the list box.
	GFX_GOL_ListBoxItemImageGet	This function gets the image set for a list box item.
	GFX_GOL_ListBoxItemImageSet	This function sets the image for a list box item.
	GFX_GOL_ListBoxItemListGet	This function returns the pointer to the item list of the list box.
	GFX_GOL_ListBoxItemSelectStatusClear	This function clears the selection status of the item.
	GFX_GOL_ListBoxItemSelectStatusSet	This function sets the selection status of the item to selected.
	GFX_GOL_ListBoxTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_ListBoxTextAlignmentSet	This function sets the text alignment of the text string used by the object.

Description

List Box supports Keyboard and Touchscreen inputs, replying to their events with the predefined actions (see GFX_GOL_ListBoxActionGet() and GFX_GOL_ListBoxActionSet() for details).

The List Box object is rendered using the assigned style scheme. The following figure illustrates the usage of the style schemes in the object.



GFX_GOL_ListBoxItemCountGet Macro

This function returns the count of items in the list box.

File

gfx_gol_list_box.h

C

```
#define GFX_GOL_ListBoxItemCountGet(pObject) \
                (((GFX_GOL_LISTBOX *)pObject)->itemsNumber)
```

Returns

The number of items in the list box.

Description

GFX GOL list box item count get.

This function returns the count of items in the list box.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object.

Function

uint16_t GFX_GOL_ListBoxItemCountGet(
GFX_GOL_LISTBOX *pObject)

GFX_GOL_ListBoxItemImageGet Macro

This function gets the image set for a list box item.

File

gfx_gol_list_box.h

C

```
#define GFX_GOL_ListBoxItemImageGet(pItem) (((GFX_GOL_LISTITEM *)pItem)->pImage)
```

Returns

Pointer to the image used for the item.

Description

GFX GOL list box item image get.

This function gets the image set for a list box item.

Preconditions

Object must exist in memory.

Example

See GFX_GOL_ListBoxItemListGet() example.

Parameters

Parameters	Description
pItem	pointer to the list box item.

Function

GFX_RESOURCE_HDR *GFX_GOL_ListBoxItemImageGet(
GFX_GOL_LISTITEM *pItem)

GFX_GOL_ListBoxItemImageSet Macro

This function sets the image for a list box item.

File

gfx_gol_list_box.h

C

```
#define GFX_GOL_ListBoxItemImageSet(pItem, image) (((GFX_GOL_LISTITEM *)pItem)->pImage = image)
```

Returns

None.

Description

GFX GOL item image set.

This function sets the image for a list box item.

Preconditions

Object must exist in memory.

Example

See GFX_GOL_ListBoxItemAdd() and GFX_GOL_ListBoxItemListGet() example.

Parameters

Parameters	Description
pItem	pointer to the list box item.
pImage	pointer to the image resource.

Function

void GFX_GOL_ListBoxItemImageSet(
GFX_GOL_LISTITEM *pItem,
GFX_RESOURCE_HDR *pImage)

GFX_GOL_ListBoxItemListGet Macro

This function returns the pointer to the item list of the list box.

File

gfx_gol_list_box.h

C

```
#define GFX_GOL_ListBoxItemListGet(pObject) \
                ((GFX_GOL_LISTITEM *)((GFX_GOL_LISTBOX *)pObject)->pItemList)
```

Returns

The pointer to the item list of the list box.

Description

GFX GOL list box item list get.

This function returns the pointer to the item list of the list box.

Preconditions

Object must exist in memory.

Example

```
// assume:
// pLb is a initialized to a list box in memory
// myIcon is a valid image in memory

GFX_GOL_LISTITEM *pList, *pItem;
GFX_RESOURCE_HDR *pMyIcon = &myIcon;

pList = GFX_GOL_ListBoxItemListGet(pLb);
pItem = pList;

// set the image for all the items to myIcon
// set the last item image to NULL

do
{
    GFX_GOL_ListBoxImageSet(pItem, &myIcon);
}
while(pItem->pNextItem != NULL);

// get the last item's image and set to null if not null
if (GFX_GOL_ListBoxItemImageGet(pItem) != NULL)
    GFX_GOL_ListBoxImageSet(pItem, NULL);
```

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_GOL_LISTITEM *GFX_GOL_ListBoxItemListGet(
GFX_GOL_LISTBOX *pObject)

GFX_GOL_ListBoxItemSelectStatusClear Macro

This function clears the selection status of the item.

File

gfx_gol_list_box.h

C

```
#define GFX_GOL_ListBoxItemSelectStatusClear(pObject, pItem) \
                                                             \
                if(pItem->status & GFX_GOL_LISTBOX_ITEM_STATUS_SELECTED)    \
                    GFX_GOL_ListBoxSelectionChange((GFX_GOL_LISTBOX *)pObject, pItem);
```

Returns

None.

Description

GFX GOL box item select status clear.

This function clears the selection status of the item.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object.
pItem	The pointer the item that will have the selected status cleared.

Function

void GFX_GOL_ListBoxItemSelectStatusClear(
GFX_GOL_LISTBOX *pObject,
GFX_GOL_LISTITEM *pItem)

GFX_GOL_ListBoxItemSelectStatusSet Macro

This function sets the selection status of the item to selected.

File

gfx_gol_list_box.h

C

```
#define GFX_GOL_ListBoxItemSelectStatusSet(pObject, pItem) \
                                                           \
                if(!(pItem->status & GFX_GOL_LISTBOX_ITEM_STATUS_SELECTED)) \
                    GFX_GOL_ListBoxSelectionChange((GFX_GOL_LISTBOX *)pObject, pItem);
```

Returns

None.

Description

GFX GOL list box item select status set.

This function sets the selection status of the item to selected.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object.
pItem	The pointer the item that will have the selected status set to selected.

Function

void GFX_GOL_ListBoxItemSelectStatusSet(
GFX_GOL_LISTBOX *pObject,
GFX_GOL_LISTITEM *pItem)

GFX_GOL_ListBoxTextAlignmentGet Macro

This function returns the text alignment of the text string used by the object.

File

gfx_gol_list_box.h

C

```
#define GFX_GOL_ListBoxTextAlignmentGet(pObject) \
                (((GFX_GOL_LISTBOX *)pObject)->alignment)
```

Returns

The text alignment set in the object. See GFX_ALIGNMENT for more details.

Description

GFX GOL list box text alignment get.

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_ALIGNMENT GFX_GOL_ListBoxTextAlignmentGet(
GFX_GOL_LISTBOX *pObject)

GFX_GOL_ListBoxTextAlignmentSet Macro

This function sets the text alignment of the text string used by the object.

File

gfx_gol_list_box.h

C

```
#define GFX_GOL_ListBoxTextAlignmentSet(pObject, align) \
                (((GFX_GOL_LISTBOX *)pObject)->alignment = align)
```

Returns

None.

Description

GFX GOL list box text alignment set.

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Function

void GFX_GOL_ListBoxTextAlignmentSet(
GFX_GOL_LISTBOX *pObject,
GFX_ALIGNMENT align)

GFX_GOL_ListBoxActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_list_box.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_ListBoxActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_LISTBOX_ACTION_TOUCHSCREEN â€“ Item is selected using touch screen
- GFX_GOL_LISTBOX_ACTION_MOVE â€“ Focus is moved to the next item depending on the key pressed (UP or DOWN key).
- GFX_GOL_LISTBOX_ACTION_SELECTED â€“ Selection is set to the currently focused item
- GFX_GOL_OBJECT_ACTION_INVALID â€“ Object is not affected

Description

GFX GOL list box action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_LISTBOX_ACTION_TOUCHSCREEN	Touch Screen	Any	Item is selected using touch screen.
GFX_GOL_LISTBOX_ACTION_MOVE	Keyboard	EVENT_KEYSCAN	Focus is moved to the next item depending on the key pressed (UP or DOWN key).
GFX_GOL_LISTBOX_ACTION_SELECTED	Keyboard	EVENT_KEYSCAN	Selection status (GFX_GOL_LISTBOX_ITEM_STATUS_SELECTED) is set to the currently focused item.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_ListBoxActionGet(
void *pObject,
GFX_GOL_MESSAGE *pMessage);

GFX_GOL_ListBoxActionSet Function

This function performs the state change of the object based on the translated action.

File

gfx_gol_list_box.h

C

```
void GFX_GOL_ListBoxActionSet(GFX_GOL_TRANSLATED_ACTION translatedMsg, void * pObject, GFX_GOL_MESSAGE *
pMessage);
```

Returns

None.

Description

GFX GOL list box action set.

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_LISTBOX_ACTION_TOUCHSCREEN	Touch Screen	Set GFX_GOL_LISTBOX_FOCUSED_STATE,	If focus is enabled, the focus state bit GFX_GOL_LISTBOX_FOCUSED_STATE will be set.
		Set GFX_GOL_LISTBOX_DRAW_FOCUS_STATE	GFX_GOL_LISTBOX_DRAW_FOCUS_STATE draw state bit will force the List Box to be redrawn with focus.
		Set GFX_GOL_LISTBOX_DRAW_ITEMS_STATE	List Box will be redrawn with selected item(s).
GFX_GOL_LISTBOX_ACTION_MOVE	KeyBoard	Set GFX_GOL_LISTBOX_DRAW_ITEMS_STATE	List Box will be redrawn with focus on one item.
GFX_GOL_LISTBOX_ACTION_SELECTED	KeyBoard	Set GFX_GOL_LISTBOX_DRAW_ITEMS_STATE	List Box will be redrawn with selection on the current item focused.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Function

void GFX_GOL_ListBoxActionSet(

GFX_GOL_TRANSLATED_ACTION translatedMsg,

void *pObject,

GFX_GOL_MESSAGE *pMessage)

GFX_GOL_ListBoxDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_list_box.h

C

```
GFX_STATUS GFX_GOL_ListBoxDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL list box draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_ListBoxDraw(void *pObject)

GFX_GOL_ListBoxItemAdd Function

This function adds an item to the list box.

File

gfx_gol_list_box.h

C

```
GFX_GOL_LISTITEM * GFX_GOL_ListBoxItemAdd(GFX_GOL_LISTBOX * pObject, GFX_GOL_LISTITEM * pPrevItem,
GFX_XCHAR * pText, GFX_RESOURCE_HDR * pImage, uint16_t status, uint16_t data);
```

Returns

The pointer to the created item.

Description

GFX GOL list box item add.

This function adds an item to the list box. This function allocates the memory needed for the GFX_GOL_LISTITEM and adds it to the list box. The newly created GFX_GOL_LISTITEM will store the location of pText, pImage and other parameters describing the added item.

Preconditions

Object must exist in memory.

Example

```
const GFX_XCHAR ItemList[] = "Line1n" "Line2n" "Line3n";

extern GFX_RESOURCE_HDR myIcon;
GFX_GOL_LISTBOX         *pLb;
GFX_GOL_LISTITEM        *pItem, *pItemList;
GFX_XCHAR               *pTemp;

// Assume that pLb is pointing to an existing list box in memory
// that is empty (no list).

// Create the list of the list box

// Initialize this to NULL to indicate that items will be added
// at the end of the list if the list exist on the list box or
// start a new list if the list box is empty.
pItem = NULL;
pTemp = ItemList;
pItem = GFX_GOL_ListBoxItemAdd(
                pLb,
                pItem,
                pTemp,
                NULL,
                LB_STS_SELECTED,
                1);
if(pItem == NULL)
    return 0;
GFX_GOL_ListBoxImageSet(pItem, &myIcon);

// Adjust pTemp to point to the next line
```

```
    while((uint16_t)*pTemp++ > (uint16_t)31);

    // add the next item
    pItem = GFX_GOL_ListBoxItemAdd(
                    pLb,
                    pItem,
                    pTemp,
                    NULL,
                    0,
                    2)
    if(pItem == NULL)
        return 0;
    GFX_GOL_ListBoxImageSet(pItem, &myIcon);

    // Adjust pTemp to point to the next line
    while((uint16_t)*pTemp++ > (uint16_t)31);

    // this time insert the next item after the first item on the list
    pItem = LbGetItemList(pLb);
    pItem = GFX_GOL_ListBoxItemAdd(
                    pLb,
                    pItem,
                    pTemp,
                    NULL,
                    0,
                    3)
    if(pItem == NULL)
        return 0;
    GFX_GOL_ListBoxImageSet(pItem, &myIcon);
```

Parameters

Parameters	Description
pObject	Pointer to the object.
pPrevItem	Pointer to the item after which a new item must be inserted, if this pointer is NULL, the item will be appended at the end of the items list.
pText	Pointer to the text that will be inserted. Text must persist in memory for as long as it is referenced by an item in the list box.
pImage	Pointer to the image for the item. Image must persist in memory for as long as it is referenced by the an item in the list box.
status	This parameter specifies if the item being added will be selected or redrawn (LB_STS_SELECTED or LB_STS_REDRAW). Refer to LISTITEM structure for details.
data	User assigned data associated with the item.

Function

```
GFX_GOL_LISTITEM *GFX_GOL_ListBoxItemAdd(
                GFX_GOL_LISTBOX     *pObject,
                GFX_GOL_LISTITEM    *pPrevItem,
                GFX_XCHAR           *pText,
                GFX_RESOURCE_HDR    *pImage,
    uint16_t        status,
    uint16_t        data)
```

GFX_GOL_ListBoxItemFocusGet Function

This function returns the index of the focused item in the list box.

File

gfx_gol_list_box.h

C

```
int16_t GFX_GOL_ListBoxItemFocusGet(GFX_GOL_LISTBOX * pObject);
```

Returns

The index of the focused item in the list box.

Description

GFX GOL list box item focus get.

This function returns the index of the focused item in the list box. First item on the list is always indexed 0. If none of the items in the list is focused, -1 is returned.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object.

Function

int16_t GFX_GOL_ListBoxItemFocusGet(
GFX_GOL_LISTBOX *pObject)

GFX_GOL_ListBoxItemFocusSet Function

This function sets the focus of the item with the index number specified by index.

File

gfx_gol_list_box.h

C

```
void GFX_GOL_ListBoxItemFocusSet(GFX_GOL_LISTBOX * pObject, uint16_t index);
```

Returns

None.

Description

GFX GOL list box item focus set.

This function sets the focus of the item with the index number specified by index. First item on the list is always indexed 0.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object.
index	The index number of the item to be focused.

Function

void GFX_GOL_ListBoxItemFocusSet(
GFX_GOL_LISTBOX *pObject,
uint16_t index)

GFX_GOL_ListBoxItemListRemove Function

This function removes the entire items list of the list box and free the memory used.

File

gfx_gol_list_box.h

C

```
void GFX_GOL_ListBoxItemListRemove(void * pObject);
```

Returns

None.

Description

GFX GOL list box item list remove.

This function removes the entire items list of the list box and free the memory used. The memory freed is the memory used for the list box. The actual text or image used for the item are not removed from memory.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_ListBoxItemListRemove(

void *pObject)

GFX_GOL_ListBoxItemRemove Function

This function removes an item from the list box and free the memory used.

File

gfx_gol_list_box.h

C

```
void GFX_GOL_ListBoxItemRemove(GFX_GOL_LISTBOX * pObject, GFX_GOL_LISTITEM * pItem);
```

Returns

None.

Description

GFX GOL list box item remove.

This function removes an item from the list box and free the memory used. The memory freed is the memory used for the list box. The actual text or image used for the item are not removed from memory.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
pItem	pointer to the list box item that will be removed.

Function

void GFX_GOL_ListBoxItemRemove(

GFX_GOL_LISTBOX *pObject

GFX_GOL_LISTITEM *pItem)

GFX_GOL_ListBoxSelectionChange Function

This function changes the selection status of the given item in the list box.

File

gfx_gol_list_box.h

C

```
void GFX_GOL_ListBoxSelectionChange(GFX_GOL_LISTBOX * pObject, GFX_GOL_LISTITEM * pItem);
```

Returns

None.

Description

GFX GOL list box selection change.

This function changes the selection status of the given item in the list box. There are two cases that this function checks Case 1: The list box is set to multiple selection.

- The item pointed to by pItem will toggle the selection status.

Case 2: The list box is set to single selection.

- If the currently selected item is not the item pointed to by pItem, the currently selected item will toggle to not selected. The item pointed to by pItem will then be set to selected.
- If the currently selected item is the same item pointed to by pItem, the selection status of that item will be set to not selected.

The change in the item's selection status should be redrawn to reflect the changes.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object.
pItem	The pointer to the item that will have the selection status changed.

Function

void GFX_GOL_ListBoxSelectionChange(
GFX_GOL_LISTBOX *pObject,
GFX_GOL_LISTITEM *pItem)

GFX_GOL_ListBoxSelectionGet Function

This function searches for selected items from the list box.

File

gfx_gol_list_box.h

C

```
GFX_GOL_LISTITEM * GFX_GOL_ListBoxSelectionGet(GFX_GOL_LISTBOX * pObject, GFX_GOL_LISTITEM * pFromItem);
```

Returns

The pointer to the first selected item found. NULL if there are no items selected.

Description

GFX GOL list box selection get.

This function searches for selected items from the list box. A starting position can optionally be given. If starting position is set to NULL, search will begin from the first item on the item list.

The function returns the pointer to the first selected item found or NULL if there are no items selected.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object.

pFromItem	The starting point of the search. If this is set to NULL, the search will start at the beginning of the item list.

Function

GFX_GOL_LISTITEM *GFX_GOL_ListBoxSelectionGet(
GFX_GOL_LISTBOX *pObject,
GFX_GOL_LISTITEM *pFromItem)

GFX_GOL_ListBoxVisibleItemCountGet Function

This function returns the count of items visible in the list box.

File

gfx_gol_list_box.h

C

```
uint16_t GFX_GOL_ListBoxVisibleItemCountGet(GFX_GOL_LISTBOX * pObject);
```

Returns

The number of visible items in the list box.

Description

GFX GOL list box visible item count get.

This function returns the count of items visible in the list box.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object.

Function

uint16_t GFX_GOL_ListBoxVisibleItemCountGet(
GFX_GOL_LISTBOX *pObject)

GFX_GOL_ListBoxCreate Function

This function creates a GFX_GOL_LISTBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_list_box.h

C

```
GFX_GOL_LISTBOX * GFX_GOL_ListBoxCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left, uint16_t
top, uint16_t right, uint16_t bottom, uint16_t state, GFX_XCHAR * pText, GFX_ALIGNMENT alignment,
GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL list box create.

This function creates a GFX_GOL_LISTBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of GFX_GOL_ListBoxCreate() will be undefined if one of the following is true:

- left >= right
- top >= bottom

- pScheme is not pointing to a GFX_GOL_OBJ_SCHEME
- pText is an unterminated string

Preconditions

None.

Example

```
#define LISTBOX_ID   10
const XCHAR ItemList[] = "Line1n" "Line2n";

GFX_GOL_OBJ_SCHEME  *pScheme;
GFX_GOL_LISTBOX     *pLb;
GFX_XCHAR           *pTemp;
uint16_t            state, counter;

//assume scheme is a valid style scheme in memory
pScheme = &scheme;
state = GFX_GOL_LISTBOX_DRAW_STATE;

// create an empty list box with default style scheme
pLb = GFX_GOL_ListBoxCreate(    gfxIndex,
                                LISTBOX_ID,     // ID number
                                10,10,150,200,  // dimension
                                state,          // initial state
                                NULL,           // set items to be empty
                                GFX_ALIGN_CENTER,
                                NULL);          // use default style scheme
// check if List box was created
if (pLb == NULL)
   return 0;

// create the list of items to be placed in the list box
// Add items (each line will become one item,
// lines must be separated by 'n' character)
pTemp = ItemList;
counter = 0;

while(*pTemp)
{
    // since each item is appended NULL is assigned to
    // GFX_GOL_LISTITEM pointer.
    if(NULL == GFX_GOL_ListBoxItemAdd(
                pLb,
                NULL,
                pTemp,
                NULL,
                0,
                counter))
    {
        break;
    }

    while(*pTemp++ > (unsigned GFX_XCHAR)31);

    if(*(pTemp-1) == 0)
        break;
    counter++;
}
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.

right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pText	Pointer to the text of the object. This is used for the items of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Function

```
GFX_GOL_LISTBOX *GFX_GOL_ListBoxCreate(
                    SYS_MODULE_INDEX    gfxIndex,
uint16_t            ID,
uint16_t            left,
uint16_t            top,
uint16_t            right,
uint16_t            bottom,
uint16_t            state,
                    GFX_XCHAR           *pText,
                    GFX_ALIGNMENT       alignment,
                    GFX_GOL_OBJ_SCHEME  *pScheme)
```

Meter Object

Meter is an object that can be used to graphically display a sampled input.

Functions

	Name	Description
	GFX_GOL_MeterActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_MeterActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_MeterCreate	This function creates a GFX_GOL_METER object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_MeterDecrement	This function decrements the meter value by the delta value set.
	GFX_GOL_MeterDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_MeterIncrement	This function increments the meter value by the delta value set.
	GFX_GOL_MeterRangeSet	This function sets the range of the meter.
	GFX_GOL_MeterScaleColorsSet	This function sets the arc colors of the object.
	GFX_GOL_MeterValueSet	This function sets the value of the meter.

Macros

	Name	Description
	GFX_GOL_MeterMaximumValueGet	This function returns the maximum value set for the meter.
	GFX_GOL_MeterMinimumValueGet	This function returns the minimum value set for the meter.
	GFX_GOL_MeterTitleFontSet	This function sets the meter title font used.
	GFX_GOL_MeterTypeSet	This function sets the meter draw type.
	GFX_GOL_MeterValueFontSet	This function sets the meter's displayed value font used.
	GFX_GOL_MeterValueGet	This function returns the current value of the meter.

Description

There are three meter types that you can draw:

- MTR_WHOLE_TYPE
- MTR_HALF_TYPE
- MTR_QUARTER_TYPE

Meter supports system inputs, replying to their events with the predefined actions (see GFX_GOL_MeterActionGet() and GFX_GOL_MeterActionSet() for details).

The Meter object is rendered using the assigned style scheme, value range colors (see GFX_GOL_MeterScaleColorSet() for details) and type (see GFX_GOL_METER_DRAW_TYPE). The following figure illustrates the assignments for a half type meter.

Default Half Meter

Half Meter with Arc enabled

Note:
Normal, Critical and Danger Colors are not part of the style scheme struct. They are fields in the METER struct.

GFX_GOL_MeterMaximumValueGet Macro

This function returns the maximum value set for the meter.

File

gfx_gol_meter.h

C

```
#define GFX_GOL_MeterMaximumValueGet(pObject) \
        (((GFX_GOL_METER *)pObject)->maxValue)
```

Returns

The current maximum value set for the object.

Description

GFX GOL meter maximum value get.

This function returns the maximum value set for the meter.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_MeterMaximumValueGet(
GFX_GOL_METER *pObject)

GFX_GOL_MeterMinimumValueGet Macro

This function returns the minimum value set for the meter.

File

gfx_gol_meter.h

C

```
#define GFX_GOL_MeterMinimumValueGet(pObject) \
        (((GFX_GOL_METER *)pObject)->minValue)
```

Returns

The current minimum value set for the object.

Description

GFX GOL meter minimum value get.

This function returns the minimum value set for the meter.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_MeterMinimumValueGet(
GFX_GOL_METER *pObject)

GFX_GOL_MeterTitleFontSet Macro

This function sets the meter title font used.

File

gfx_gol_meter.h

C

```
#define GFX_GOL_MeterTitleFontSet(pObject, pNewFont) \
        (((GFX_GOL_METER *)pObject)->pTitleFont = pNewFont)
```

Returns

None.

Description

GFX GOL meter title font set.

This function sets the meter title font used. Font pointer must be initialized to a valid GFX_RESOURCE_HDR.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
pNewFont	pointer to the selected font.

Function

void GFX_GOL_MeterTitleFontSet(

GFX_GOL_METER *pObject,
GFX_RESOURCE_HDR *pNewFont)

GFX_GOL_MeterTypeSet Macro

This function sets the meter draw type.

File

gfx_gol_meter.h

C

```
#define GFX_GOL_MeterTypeSet(pObject, type) \
        (((GFX_GOL_METER *)pObject)->type = type)
```

Returns

None.

Description

GFX GOL meter type set.

This function sets the meter draw type.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
type	the draw type selected (see GFX_GOL_METER_DRAW_TYPE for more information).

Function

void GFX_GOL_MeterTypeSet(
GFX_GOL_METER *pObject,
GFX_GOL_METER_DRAW_TYPE type)

GFX_GOL_MeterValueFontSet Macro

This function sets the meter's displayed value font used.

File

gfx_gol_meter.h

C

```
#define GFX_GOL_MeterValueFontSet(pObject, pNewFont) \
        (((GFX_GOL_METER *)pObject)->pValueFont = pNewFont)
```

Returns

None.

Description

GFX GOL meter value font set.

This function sets the meter's displayed value font used. Font pointer must be initialized to a valid GFX_RESOURCE_HDR.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

pNewFont	pointer to the selected font.

Function

void GFX_GOL_MeterValueFontSet(
GFX_GOL_METER *pObject,
GFX_RESOURCE_HDR *pNewFont)

GFX_GOL_MeterValueGet Macro

This function returns the current value of the meter.

File

gfx_gol_meter.h

C

```
#define GFX_GOL_MeterValueGet(pMtr) (((GFX_GOL_METER*)pMtr)->value)
```

Returns

The current value of the object.

Description

GFX GOL meter value get.

This function returns the current value of the meter.

Preconditions

Object must exist in memory.

Example

```
#define MAXVALUE 100;

GFX_GOL_METER *pMeter;
uint32_t ctr = 0;

// create scroll bar here and initialize parameters
pMeter = GFX_GOL_MeterCreate(....)
GFX_GOL_ObjectStateSet(pMeter, GFX_GOL_METER_DRAW_STATE);

// draw the scroll bar
GFX_GOL_ObjectListDraw(gfxIndex);

// a routine that updates the position of the thumb through some
// conditions
while("some condition")
{
    GFX_GOL_MeterValueSet(pMeter, ctr);
    GFX_GOL_ObjectStateSet( pMeter,
                            GFX_GOL_METER_UPDATE_DRAW_STATE);

    // update the screen
    GFX_GOL_ObjectListDraw(gfxIndex);

    // update ctr here
    ctr = "some source of value";
}

if (GFX_GOL_MeterValueGet(pScrollBar) > MAXVALUE)
    return 0;
else
    "do something else"
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

uint16_t GFX_GOL_MeterValueGet(
GFX_GOL_METER *pObject)

GFX_GOL_MeterActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_meter.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_MeterActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_METER_ACTION_SET â€“ Meter id is given in parameter 1 for a TYPE_SYSTEM message.
- GFX_GOL_OBJECT_ACTION_INVALID â€“ Object is not affected

Description

GFX GOL meter action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_METER_ACTION_SET	System	EVENT_SET	If event set occurs and the meter id is sent in parameter 1.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_MeterActionGet(
void *pObject,
GFX_GOL_MESSAGE *pMessage)

GFX_GOL_MeterActionSet Function

This function performs the state change of the object based on the translated action.

File

gfx_gol_meter.h

C

```
void GFX_GOL_MeterActionSet(GFX_GOL_TRANSLATED_ACTION translatedMsg, void * pObject, GFX_GOL_MESSAGE *
pMessage);
```

Returns

None.

Description

GFX GOL meter action set.

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_METER_ACTION_SET	System	Set GFX_GOL_METER_DRAW_STATE	Meter will be redrawn to update the needle position and value displayed. The new value is given in the pMessage parameter 2 (param2). While parameter 1 (param1) of the message holds the ID of the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Function

void GFX_GOL_MeterActionSet(

GFX_GOL_TRANSLATED_ACTION translatedMsg,

void *pObject,

GFX_GOL_MESSAGE *pMessage)

GFX_GOL_MeterCreate Function

This function creates a GFX_GOL_METER object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_meter.h

C

```
GFX_GOL_METER * GFX_GOL_MeterCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left, uint16_t top,
uint16_t right, uint16_t bottom, uint16_t state, GFX_GOL_METER_DRAW_TYPE type, int16_t value, int16_t
minValue, int16_t maxValue, GFX_RESOURCE_HDR * pTitleFont, GFX_RESOURCE_HDR * pValueFont, GFX_XCHAR *
pText, GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL meter create.

This function creates a GFX_GOL_METER object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of GFX_GOL_MeterCreate() will be undefined if one of the following is true:

- left >= right
- top >= bottom
- type is not one of the defined types
- pTitleFont and pValueFont is not defined to a valid font GFX_RESOURC_HDR
- pScheme is not pointing to a GFX_GOL_OBJ_SCHEME
- pText is an unterminated string

Preconditions

None.

Example

```
#define ID_METER 101

extern const FONT_FLASH GOLMediumFont;      // medium font
extern const FONT_FLASH GOLSmallFont;       // small font

GFX_GOL_OBJ_SCHEME *pMeterScheme;
GFX_GOL_METER *pMtr;

// assume pMeterScheme is initialized to a scheme in memory.

// draw object after creation
state = GFX_GOL_METER_DRAW_STATE | GFX_GOL_METER_RING_STATE;

pMtr = GFX_GOL_MeterCreate(
            gfxIndex,
            ID_METER,                   // assign ID
            30, 50, 150, 180,           // set dimension
            state,
            GFX_GOL_METER_WHOLE_TYPE,   // type of meter
            0,                          // set initial value
            0, 100,                     // set min and max value
            &GOLMediumFont,             // set title font
            &GOLSmallFont,              // set value font
            "Speed",                    // Text Label
            pMeterScheme);              // style scheme

// check if meter was created
if (pMtr == NULL)
    return 0;

// Change range colors: Normal values to WHITE
//                      Critical values to BLUE
//                      Danger values to RED
// assume that WHITE, GREEN, YELLOW and RED have been defined.
GFX_GOL_MeterScaleColorSet(pMtr, WHITE, WHITE, WHITE,
                                 GREEN, YELLOW, RED);

// use GOLDraw() to draw the meter created
while(!GOLDraw());
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
type	Specifies the type of Meter to be drawn (see GFX_GOL_METER_TYPE).
value	Initial value set to the meter.
minValue	The minimum value the meter will display.
maxValue	The maximum value the meter will display.
pTitleFont	Pointer to the font used for the Title.
pValueFont	Pointer to the font used for the value.
pText	Pointer to the text label of the meter.
pScheme	Pointer to the style scheme used.

Function

```
GFX_GOL_METER *GFX_GOL_MeterCreate(
            SYS_MODULE_INDEX    gfxIndex,
```

```
uint16_t            ID,
uint16_t            left,
uint16_t            top,
uint16_t            right,
uint16_t            bottom,
uint16_t            state,
                    GFX_GOL_METER_DRAW_TYPE type,
int16_t             value,
int16_t             minValue,
int16_t             maxValue,
                    GFX_RESOURCE_HDR       *pTitleFont,
                    GFX_RESOURCE_HDR       *pValueFont,
                    GFX_XCHAR              *pText,
                    GFX_GOL_OBJ_SCHEME     *pScheme)
```

GFX_GOL_MeterDecrement Function

This function decrements the meter value by the delta value set.

File

gfx_gol_meter.h

C

```
void GFX_GOL_MeterDecrement(GFX_GOL_METER * pObject, uint16_t delta);
```

Returns

None.

Description

GFX GOL meter decrement.

This function decrements the meter value by the given delta value set. If the delta given is less than the minimum value of the meter, the value will remain to be at minimum.

Object must be redrawn after this function is called to reflect the changes to the object.

Preconditions

Object must exist in memory.

Example

See GFX_GOL_MeterIncrement().

Parameters

Parameters	Description
pObject	pointer to the object.

Function

```
void GFX_GOL_MeterDecrement(
                    GFX_GOL_METER *pObject,
uint16_t delta)
```

GFX_GOL_MeterDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_meter.h

C

```
GFX_STATUS GFX_GOL_MeterDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished.

Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL meter draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_MeterDraw(void *pObject)

GFX_GOL_MeterIncrement Function

This function increments the meter value by the delta value set.

File

gfx_gol_meter.h

C

```
void GFX_GOL_MeterIncrement(GFX_GOL_METER * pObject, uint16_t delta);
```

Returns

None.

Description

GFX GOL meter increment

This function increments the scroll bar position by the given delta value set. If the delta given exceeds the maximum value of the meter, the value will remain to be at maximum.

Object must be redrawn after this function is called to reflect the changes to the object.

Preconditions

Object must exist in memory.

Example

```
void ControlSpeed(  GFX_GOL_METER* pObj,
                    int setSpeed,
                    int curSpeed)
{
    // set page size to 1
    GFX_GOL_MeterValueSet(pObj, 1);

    if (setSpeed < curSpeed)
    {
        while(GFX_GOL_MeterValueGet(pObj) < SetSpeed)
            GFX_GOL_MeterIncrement(pObj, 1);  // increment by 1
    }
    else if (setSpeed > curSpeed)
    {
        while(GFX_GOL_MeterValueGet(pObj) > SetSpeed)
            GFX_GOL_MeterDecrement(pObj, 1);  // decrement by 1
    }
}
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_MeterIncrement(

GFX_GOL_METER *pObject,

uint16_t delta)

GFX_GOL_MeterRangeSet Function

This function sets the range of the meter.

File

gfx_gol_meter.h

C

```
void GFX_GOL_MeterRangeSet(GFX_GOL_METER * pObject, int16_t minValue, int16_t maxValue);
```

Returns

None.

Description

GFX GOL meter range set.

This function sets the range of the meter. When the range is modified, object must be completely redrawn to reflect the change. minValue should always be less than maxValue.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
minValue	new minimum value of the object.
maxValue	new maximum value of the object.

Function

void GFX_GOL_MeterRangeSet(

GFX_GOL_METER *pObject,

int16_t minValue,

int16_t maxValue)

GFX_GOL_MeterScaleColorsSet Function

This function sets the arc colors of the object.

File

gfx_gol_meter.h

C

```
void GFX_GOL_MeterScaleColorsSet(GFX_GOL_METER * pObject, GFX_COLOR color1, GFX_COLOR color2, GFX_COLOR
color3, GFX_COLOR color4, GFX_COLOR color5, GFX_COLOR color6);
```

Returns

None.

Description

GFX GOL meter scale colors set.

After the object is created, this function must be called to set the arc colors of the object.

Scale colors can be used to highlight values of the meter. User can set these colors to define the arc colors and scale colors. This also sets the color of the meter value when displayed. The color settings are set to the following angles:

Color Boundaries	Type Whole	Type Half	Type Quarter
Arc 6	225 to 180	not used	not used
Arc 5	179 to 135	179 to 135	not used
Arc 4	134 to 90	134 to 90	not used
Arc 3	89 to 45	89 to 45	89 to 45
Arc 2	44 to 0	44 to 0	44 to 0
Arc 1	-45 to -1	not used	not used

As the meter is drawn colors are changed depending on the angle of the scale and label being drawn.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
color1	color assigned to Arc 1 and Scale 1.
color2	color assigned to Arc 2 and Scale 2.
color3	color assigned to Arc 3 and Scale 3.
color4	color assigned to Arc 4 and Scale 4.
color5	color assigned to Arc 5 and Scale 5.
color6	color assigned to Arc 6 and Scale 6.

Function

void GFX_GOL_MeterScaleColorsSet(
GFX_GOL_METER *pObject,
GFX_COLOR color1,
GFX_COLOR color2,
GFX_COLOR color3,
GFX_COLOR color4,
GFX_COLOR color5,
GFX_COLOR color6)

GFX_GOL_MeterValueSet Function

This function sets the value of the meter.

File

gfx_gol_meter.h

C

```
void GFX_GOL_MeterValueSet(GFX_GOL_METER * pObject, int16_t value);
```

Returns

None.

Description

GFX GOL meter value set.

This function sets the value of the meter. The value used should be within the range set for the object. The new value is checked to be in the minimum and maximum value range. If the value set is less than the minimum value, the value that will be set is the minimum value. If the value set is more than the maximum value, then the value that will be set is the maximum value.

Preconditions

Object must exist in memory.

Example

```
GFX_GOL_METER *pMeter;
uint16_t ctr = 0;

// create slider here and initialize parameters
GFX_GOL_ObjectStateSet(pMeter, GFX_GOL_METER_DRAW_STATE);
GFX_GOL_ObjectListDraw(gfxIndex);

while("some condition")
{
    GFX_GOL_MeterValueSet(pMeter, ctr);
    GFX_GOL_ObjectStateSet( pMeter,
                            GFX_GOL_METER_UPDATE_DRAW_STATE);

    // redraw the scroll bar
    GFX_GOL_ObjectListDraw(gfxIndex);

   // update ctr here
   ctr = "some source of value";
}
```

Parameters

Parameters	Description
pObject	pointer to the object.
value	the new value of the object.

Function

void GFX_GOL_MeterValueSet(
GFX_GOL_METER *pObject,
int16_t value)

Panel Object

Functions

	Name	Description
	GFX_GOL_PanelAlphaParameterSet	This function sets the alpha blending value when using alpha blending in panels.
	GFX_GOL_PanelBackgroundSet	This function sets panel background information.
	GFX_GOL_PanelDraw	This function renders the panel.
	GFX_GOL_PanelGradientParameterSet	This function sets the gradient fill start and end colors of a panel.
	GFX_GOL_PanelParameterSet	This function sets the parameters to draw a panel.

Description

GFX_GOL_PanelAlphaParameterSet Function

This function sets the alpha blending value when using alpha blending in panels.

File

gfx_gol.h

C

```
void GFX_GOL_PanelAlphaParameterSet(SYS_MODULE_INDEX gfxIndex, uint16_t alphaValue);
```

Returns

None.

Description

GFX GOL panel alpha parameter set.

This function sets the alpha blending value when using alpha blending in panels. This along with the parameters set by the function GFX_GOL_PanelParameterSet() will determine how the panel will be drawn. The actual drawing of the panel is performed by GFX_GOL_PanelDraw().

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
alphaVlaue	the alpha value used to render alpha blended panels.

Function

void GFX_GOL_PanelAlphaParameterSet(SYS_MODULE_INDEX gfxIndex,
uint16_t alphaValue)

GFX_GOL_PanelBackgroundSet Function

This function sets panel background information.

File

gfx_gol.h

C

```
void GFX_GOL_PanelBackgroundSet(SYS_MODULE_INDEX gfxIndex, GFX_GOL_OBJ_HEADER * pObjectHeader);
```

Returns

None.

Description

GFX GOL panel background set.

This function sets panel background information. This is an internal function and should not be called by the application.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
pObjectHeader	the object header of the object that needs to draw a panel with background.

Function

void GFX_GOL_PanelBackgroundSet(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_OBJ_HEADER *pObjectHeader)

GFX_GOL_PanelDraw Function

This function renders the panel.

File

gfx_gol.h

C

```
GFX_STATUS GFX_GOL_PanelDraw(SYS_MODULE_INDEX gfxIndex);
```

Returns

GFX_STATUS_SUCCESS - when the panel rendering is done. GFX_STATUS_FAILURE - when the panel rendering is not yet done.

Description

GFX GOL panel draw.

This function renders the panel. Panel parameters are set by the GFX_GOL_PanelParameterSet() and GFX_GOL_PanelGradientParameterSet() or GFX_GOL_PanelAlphaParameterSet(). The function returns success (GFX_STATUS_SUCCESS) when the panel is rendered. If the function returned not success this function must be called again until success is returned.

Preconditions

Panel parameters must be set first using GFX_GOL_PanelParameterSet() and GFX_GOL_PanelGradientParameterSet() or GFX_GOL_PanelAlphaParameterSet().

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.

Function

GFX_STATUS GFX_GOL_PanelDraw(SYS_MODULE_INDEX gfxIndex)

GFX_GOL_PanelGradientParameterSet Function

This function sets the gradient fill start and end colors of a panel.

File

gfx_gol.h

C

```
void GFX_GOL_PanelGradientParameterSet(SYS_MODULE_INDEX gfxIndex, GFX_COLOR startColor, GFX_COLOR endColor);
```

Returns

None.

Description

GFX GOL panel gradient parameter set.

This function sets the gradient fill start and end colors of a panel. This along with the parameters set by the function GFX_GOL_PanelParameterSet() will determine how the panel will be drawn. The actual drawing of the panel is performed by GFX_GOL_PanelDraw().

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
startColor	the gradient fill start color.
endColor	the gradient fill end color.

Function

void GFX_GOL_PanelGradientParameterSet(SYS_MODULE_INDEX gfxIndex,

GFX_COLOR startColor,

GFX_COLOR endColor)

GFX_GOL_PanelParameterSet Function

This function sets the parameters to draw a panel.

File

gfx_gol.h

C

```
void GFX_GOL_PanelParameterSet(SYS_MODULE_INDEX gfxIndex, uint16_t left, uint16_t top, uint16_t right,
uint16_t bottom, uint16_t radius, GFX_COLOR faceClr, GFX_COLOR embossLtClr, GFX_COLOR embossDkClr,
```

```
GFX_RESOURCE_HDR * pBitmap, GFX_FILL_STYLE fillStyle, uint16_t embossSize);
```

Returns

None.

Description

GFX GOL panel parameter set.

This function sets the parameters to draw a panel. Panel is not an object. It is a routine to draw a basic component of objects. The actual drawing of the panel is performed by the GFX_GOL_PanelDraw(). After the parameters are set, call GFX_GOL_PanelDraw() to render the panel. The panel is drawn using the following:

1. Panel width is determined by right - left.
2. Panel height is determined by top - bottom.
3. Panel radius - specifies if the panel will have a rounded edge. If zero then the panel will have sharp (cornered) edge.
4. If 2*radius = height = width, the panel is circular.
5. If the panel is drawn with an image, pBitmap should point to an image resource.
6. If the panel face is drawn with the fill style specified by fillStyle. When gradient fill is used, set the gradient colors using GFX_GOL_PanelGradientParameterSet(). When alpha blending fill is used, set the alpha blending value using GFX_GOL_PanelAlphaParameterSet().

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
left	defines the left most pixel of the panel.
top	defines the top most pixel of the panel.
right	defines the right most pixel of the panel.
bottom	defines the bottom most pixel of the panel.
radius	defines the radius of the rounded corner. A zero value will result in a rectangular panel drawn.
faceClr	the color used for the face of the panel.
embossLtClr	the color used for the light emboss color for 3D effect.
embossDkClr	the color used for the dark emboss color for 3D effect.
pBitmap	pointer to the image resource of the panel.
fillStyle	fill style use for the face of the panel.
embossSize	when this is not zero, the embossLtClr and embossDkClr are used to draw the 3D effect. When this is set to zero, there will be no 3D effect.

Function

void GFX_GOL_PanelParameterSet(
SYS_MODULE_INDEX gfxIndex
uint16_t left,
uint16_t top,
uint16_t right,
uint16_t bottom,
uint16_t radius,
GFX_COLOR faceClr,
GFX_COLOR embossLtClr,
GFX_COLOR embossDkClr,
GFX_RESOURCE_HDR *pBitmap,
GFX_FILL_STYLE fillStyle,
uint16_t embossSize)

Picture Control Object

Picture Control is an object that can be used to transform an image to be an object in the screen and have control on the rendering. This object can be used to create animation using a series of bitmaps.

Functions

	Name	Description
	GFX_GOL_PictureControlActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_PictureControlCreate	This function creates a GFX_GOL_PICTURECONTROL object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_PictureControlDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_PictureControlPartialSet	This function sets the partial image parameters to be in the object.
	GFX_GOL_PictureControlScaleSet	Sets the scale factor used to render the image used in the object.

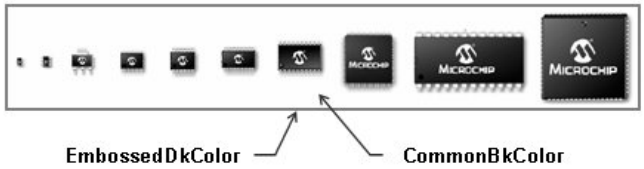
Macros

	Name	Description
	GFX_GOL_PictureControlImageGet	This function gets the image used when in the pressed state.
	GFX_GOL_PictureControlImageSet	This function sets the image to be in the object.

Description

Picture Control object supports Touchscreen inputs, replying to the events with the predefined actions (see GFX_GOL_PictureActionGet() for details).

The Picture object is rendered using the assigned style scheme. The following figure illustrates the usage of the style schemes in the object.



GFX_GOL_PictureControlImageGet Macro

This function gets the image used when in the pressed state.

File

gfx_gol_picture.h

C

```
#define GFX_GOL_PictureControlImageGet(pObject) \
                    (((GFX_GOL_PICTURECONTROL*)pObject)->pImage)
```

Returns

Pointer to the image resource.

Description

GFX GOL picture control image get.

This function gets the image used when in the pressed state.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_RESOURCE_HDR *GFX_GOL_PictureControlImageGet(
GFX_GOL_PICTURECONTROL *pObject)

GFX_GOL_PictureControlImageSet Macro

This function sets the image to be in the object.

File

gfx_gol_picture.h

C

```
#define GFX_GOL_PictureControlImageSet(pObject, image) \
                    (((GFX_GOL_PICTURECONTROL*)pObject)->pImage = (GFX_RESOURCE_HDR *)image)
```

Returns

None.

Description

GFX GOL picture control image set.

This function sets the image to be in the object.

Preconditions

Object must exist in memory.

Example

```
// assume OrigImage and NewImage are valid GFX_RESOURCE_HDR
// pointers for images

GFX_RESOURCE_HDR *pOrigIcon = &OrigImage;
GFX_RESOURCE_HDR *pNewIcon = &NewImage;
GFX_GOL_PICTURECONTROL  *pPicture;

pPicture = GFX_GOL_PictureControlCreate(
                    gfxIndex,
                    10,
                    0, 0,
                    GFX_MaxXGet(), GFX_MaxYGet(),
                    GFX_GOL_PICTURECONTROL_DRAW_STATE,
                    pOrigIcon,
                    NULL);

// change the image used
GFX_GOL_PictureControlImageSet(pPicture , pNewIcon);
```

Parameters

Parameters	Description
pObject	pointer to the object.
pImage	pointer to the image resource.

Function

void GFX_GOL_PictureControlImageSet(
GFX_GOL_PICTURECONTROL *pObject,
GFX_RESOURCE_HDR *pImage)

GFX_GOL_PictureControlActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_picture.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_PictureControlActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_PICTURECONTROL_ACTION_SELECTED â€“ Object is selected
- GFX_GOL_OBJECT_ACTION_INVALID â€“ Object is not affected

Description

GFX GOL picture control action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_PICTURECONTROL_ACTION_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE	If events occurs and the x,y position falls in the area of the picture.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_PictureControlActionGet(

void *pObject,

GFX_GOL_MESSAGE *pMessage);

GFX_GOL_PictureControlCreate Function

This function creates a GFX_GOL_PICTURECONTROL object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_picture.h

C

```
GFX_GOL_PICTURECONTROL * GFX_GOL_PictureControlCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t
left, uint16_t top, uint16_t right, uint16_t bottom, uint16_t state, int8_t scaleFactor, GFX_RESOURCE_HDR *
pImage, GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL picture control create.

This function creates a GFX_GOL_PICTURECONTROL object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The object allows creation with the image set to NULL. In this case, nothing will be drawn when the object is set to be drawn and the frame is not enabled (See GFX_GOL_PICTURECONTROL_FRAME_STATE). If the frame is enabled, then only the frame will be drawn.

When the assigned image's dimension is larger than the dimension of the object, partial image parameters will be set in such a way that the upper left most corner of the image that has the same dimension as the object will be used in the object. This is the default behavior.

The partial parameters can be modified by calling the GFX_GOL_PictureControlPartialSet() function with the desired partial image parameters. See GFX_ImagePartialDraw() for details on the partial image rendering.

The behavior of GFX_GOL_PictureControlCreate() will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a GFX_GOL_OBJ_SCHEME
- pImage is not pointing to a GFX_RESOURCE_HDR.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
scaleFactor	Sets the scaling factor when the image is rendered. This feature is only available in certain builds.
pImage	Pointer to the image by the object
pScheme	Pointer to the style scheme used.

Function

```
    GFX_GOL_PICTURECONTROL  *GFX_GOL_PictureControlCreate(
                SYS_MODULE_INDEX    gfxIndex,
uint16_t        ID,
uint16_t        left,
uint16_t        top,
uint16_t        right,
uint16_t        bottom,
uint16_t        state,
int8_t          scaleFactor,
                GFX_RESOURCE_HDR    *pImage,
                GFX_GOL_OBJ_SCHEME  *pScheme)
```

GFX_GOL_PictureControlDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_picture.h

C

```
GFX_STATUS GFX_GOL_PictureControlDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL picture control draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When the image size The text on the face of the GFX_GOL_PICTURECONTROL is drawn on top of the bitmap. Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to

avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_PictureControlDraw(void *pObject)

GFX_GOL_PictureControlPartialSet Function

This function sets the partial image parameters to be in the object.

File

gfx_gol_picture.h

C

```
void GFX_GOL_PictureControlPartialSet(GFX_GOL_PICTURECONTROL * pObject, uint16_t xOffset, uint16_t yOffset,
uint16_t partialWidth, uint16_t partialHeight);
```

Returns

None.

Description

GFX GOL picture control partial set.

This function sets the partial image parameters to be used in the object. This function allows usage of the object to specify a rectangular area of an image to be drawn as part of the object. This is useful when an image is already included in a larger image. To save memory, a separate image is not necessary for the picture object. The location of the smaller image in the larger image can be specified to show up in the picture object.

This function will result in an undefined behavior when one of the following is true:

- xOffset - value must not be greater than the image width.
- yOffset - value must not be greater than the image height.
- partialWidth - value must not be greater than image width - xoffset + 1. Value must also be less than the actual image width.
- partialHeight - value must not be greater than image height - yoffset + 1. Value must also be less than the actual image height.

Preconditions

Object must exist in memory. The image pointer of the object must be initialized properly.

Example

```
// assume pLargeImage is a valid GFX_RESOURCE_HDR
// assume BigImage has a height and width of 100 pixels.

GFX_RESOURCE_HDR *pLargeImage = &BigImage;
GFX_GOL_PICTURECONTROL  *pPicture;
uint16_t         width, height;
uint16_t         xOffset, yOffset;
uint16_t         objectWidth, objectHeight;

objectWidth  = 60  - 50; // 10 pixels
objectHeight = 120 - 90; // 30 pixels

// -1 is needed since the object dimension is inclusive
pPicture = GFX_GOL_PictureControlCreate(
                    gfxIndex,
                    10,
                    50, 90,
                    50 + objectWidth - 1,
                    90 + objectHeight - 1,
```

```
                        GFX_GOL_PICTURECONTROL_DRAW_STATE,
                        largeImage,
                        NULL);

    // set the parameters of the partial image to be
    // shown on the image

    // get the large image dimensions
    width  = GFX_ImageWidthGet(pLargeImage);
    height = GFX_ImageHeightGet(pLargeImage);

    // get the offset so the middle of the large image with
    // the width and height matching the object will be used.
    xOffset = (width  - objectWidth)  >> 1;
    yOffset = (height - objectHeight) >> 1;

    GFX_GOL_PictureControlPartialSet(  pPicture,
                              xOffset, yOffset,
                              objectWidth, objectHeight);
```

Parameters

Parameters	Description
xOffset	x offset of the smaller portion of a large image
yOffset	y offset of the smaller portion of a large image
partialWidth	width of the selected portion of the image
partialHeight	height of the selected portion of the image

Function

void GFX_GOL_PictureControlPartialSet(

GFX_GOL_PICTURECONTROL *pObject,

uint16_t xOffset,

uint16_t yOffset,

uint16_t partialWidth,

uint16_t partialHeight)

GFX_GOL_PictureControlScaleSet Function

Sets the scale factor used to render the image used in the object.

File

gfx_gol_picture.h

C

```
void GFX_GOL_PictureControlScaleSet(GFX_GOL_PICTURECONTROL * pObject, int8_t scale);
```

Returns

None.

Description

GFX GOL picture control scale set.

This function sets the scale factor used to render the image used in the object using scale.

Remarks

None.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object
scale	Scale factor that will be used to display the image.

Function

void GFX_GOL_PictureControlScaleSet(
GFX_GOL_PICTURECONTROL pObject,
int8_t scale)

Progress Bar Object

Progress Bar is an object that can be used to display the progress of a task such as a data download or transfer.

Functions

	Name	Description
	GFX_GOL_ProgressBarActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_ProgressBarCreate	This function creates a GFX_GOL_PROGRESSBAR object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_ProgressBarDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_ProgressBarPositionSet	This function sets the position of the progress bar.
	GFX_GOL_ProgressBarRangeSet	This function sets the range of the progress bar.

Macros

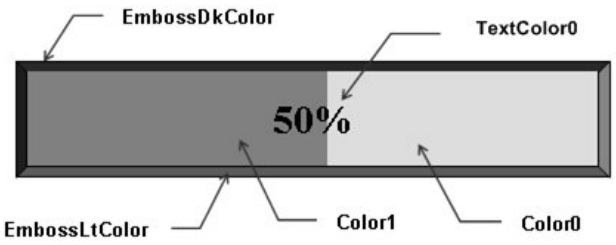
	Name	Description
	GFX_GOL_ProgressBarPositionGet	This function returns the current position of the progress bar.
	GFX_GOL_ProgressBarRangeGet	This function returns the range of the progress bar.

Description

Progress Bar is an object that can be used to display the progress of a task such as a data download or transfer.

Progress Bar supports only Touchscreen inputs, replying to the events with the predefined actions (see GFX_GOL_ProgressBarActionGet() for details).

The Progress Bar object is rendered using the assigned style scheme. The following figure illustrates the usage of the style schemes in the object.



CommonBkColor – used to hide/remove the progress bar from the screen.

GFX_GOL_ProgressBarPositionGet Macro

This function returns the current position of the progress bar.

File

gfx_gol_progress_bar.h

C

```
#define GFX_GOL_ProgressBarPositionGet(pObject) \
                                        (((GFX_GOL_PROGRESSBAR*)pObject)->pos)
```

Returns

The current position of the scroll bar thumb.

Description

GFX GOL progress bar position get.

This function returns the current position of the progress bar.

Preconditions

Object must exist in memory.

Example

See GFX_GOL_ProgressBarPositionSet() example.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

uint16_t GFX_GOL_ProgressBarPositionGet(
GFX_GOL_PROGRESSBAR *pObject)

GFX_GOL_ProgressBarRangeGet Macro

This function returns the range of the progress bar.

File

gfx_gol_progress_bar.h

C

```
#define GFX_GOL_ProgressBarRangeGet(pObject) (pObject->range)
```

Returns

The current range used by the object.

Description

GFX GOL progress bar range get.

This function returns the range of the progress bar.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_ProgressBarRangeGet(
GFX_GOL_PROGRESSBAR *pObject)

GFX_GOL_ProgressBarActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_progress_bar.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_ProgressBarActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_PROGRESSBAR_ACTION_SELECTED â€“ Object is selected
- GFX_GOL_OBJECT_ACTION_INVALID â€“ Object is not affected

Description

GFX GOL progress bar action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit Description
GFX_GOL_PROGRESSBAR_ACTION_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE, If events occurs and the x,y position falls in the area of the object. EVENT_MOVE
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_ProgressBarActionGet(
void *pObject,
GFX_GOL_MESSAGE *pMessage);

GFX_GOL_ProgressBarCreate Function

This function creates a GFX_GOL_PROGRESSBAR object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_progress_bar.h

C

```
GFX_GOL_PROGRESSBAR * GFX_GOL_ProgressBarCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left,
uint16_t top, uint16_t right, uint16_t bottom, uint16_t state, uint16_t pos, uint16_t range,
GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL progress bar create.

This function creates a GFX_GOL_PROGRESSBAR object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of GFX_GOL_RadioButtonCreate() will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pos > range
- range = 0
- pScheme is not pointing to a GFX_GOL_OBJ_SCHEME
- pText is an unterminated string

Preconditions

None.

Example

```
GFX_GOL_PROGRESSBAR *pPBar;
void CreateProgressBar()
{
    pPBar = GFX_GOL_ProgressBarCreate(
                    gfxIndex,
                    ID_PROGRESSBAR1,   // ID
                    50,90,270,140,      // dimension
                    PB_DRAW,            // Draw the object
                    25,                 // position
                    50,                 // set the range
                    NULL);              // use default GOL scheme

}
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pos	Defines the initial position of the progress.
range	This specifies the maximum value of the progress bar when the progress bar is at 100% position.
pScheme	Pointer to the style scheme used.

Function

```
GFX_GOL_PROGRESSBAR *GFX_GOL_ProgressBarCreate(
                SYS_MODULE_INDEX    gfxIndex,
uint16_t        ID,
uint16_t        left,
uint16_t        top,
uint16_t        right,
uint16_t        bottom,
uint16_t        state,
uint16_t        pos,
uint16_t        range,
                GFX_GOL_OBJ_SCHEME *pScheme)
```

GFX_GOL_ProgressBarDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_progress_bar.h

C

```
GFX_STATUS GFX_GOL_ProgressBarDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL progress bar draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_ProgressBarDraw(void *pObject)

GFX_GOL_ProgressBarPositionSet Function

This function sets the position of the progress bar.

File

gfx_gol_progress_bar.h

C

```
void GFX_GOL_ProgressBarPositionSet(GFX_GOL_PROGRESSBAR * pObject, uint16_t position);
```

Returns

None.

Description

GFX GOL progress bar position set.

This function sets the position of the progress bar.

The value used for the position should be within the range set for the object.

Function will have an undefined behavior if the position is outside the range.

Preconditions

Object must exist in memory.

Example

```
        GFX_GOL_PROGRESSBAR *pPb;
        uint8_t  direction = 1;

// this code increments and decrements the progress bar by 1
// assume progress bar was created and initialized before
while (1)
        {
            if(direction)
            {
                if(pPb ->pos == pPb ->range)
                    direction = 0;
                else
                    GFX_GOL_ProgressBarPositionSet(
                        pPb,
                        GFX_GOL_ProgressBarPositionGet(pPb)+1);
            }
            else
            {
                if(pPb ->pos == 0)
                    direction = 1;
                else
                    GFX_GOL_ProgressBarPositionSet(
```

```
                        pPb,
                        GFX_GOL_ProgressBarPositionGet(pPb)-1);
            }
        }
```

Parameters

Parameters	Description
pObject	pointer to the object.
position	the new position of the scroll bar thumb.

Function

void GFX_GOL_ProgressBarPositionSet(

GFX_GOL_PROGRESSBAR *pObject,

uint16_t position)

GFX_GOL_ProgressBarRangeSet Function

This function sets the range of the progress bar.

File

gfx_gol_progress_bar.h

C

```
void GFX_GOL_ProgressBarRangeSet(GFX_GOL_PROGRESSBAR * pObject, uint16_t range);
```

Returns

None.

Description

GFX GOL progress bar range set.

This function sets the range of the progress bar. When the range is modified, object must be completely redrawn to reflect the change.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
range	new range of the object.

Function

void GFX_GOL_ProgressBarRangeSet(

GFX_GOL_PROGRESSBAR *pObject,

uint16_t range)

Radio Button Object

Radio Button is an object that can be used to offer set of choices to the user. Only one of the choices is selectable. Changing selection automatically removes the selection on the previous option.

Functions

	Name	Description
	GFX_GOL_RadioButtonCheckGet	This function returns the ID of the currently checked radio button in the group.
	GFX_GOL_RadioButtonActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_RadioButtonActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_RadioButtonCheckSet	This function sets the ID of the currently checked radio button in the group.
	GFX_GOL_RadioButtonDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_RadioButtonTextAlignmentGet	This function returns the text alignment of the text string used by the object.

	GFX_GOL_RadioButtonTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_RadioButtonTextSet	This function sets the address of the current text string used by the object.
	GFX_GOL_RadioButtonCreate	This function creates a GFX_GOL_RADIOBUTTON object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_RadioButtonListCreate	This is function GFX_GOL_RadioButtonListCreate.

Macros

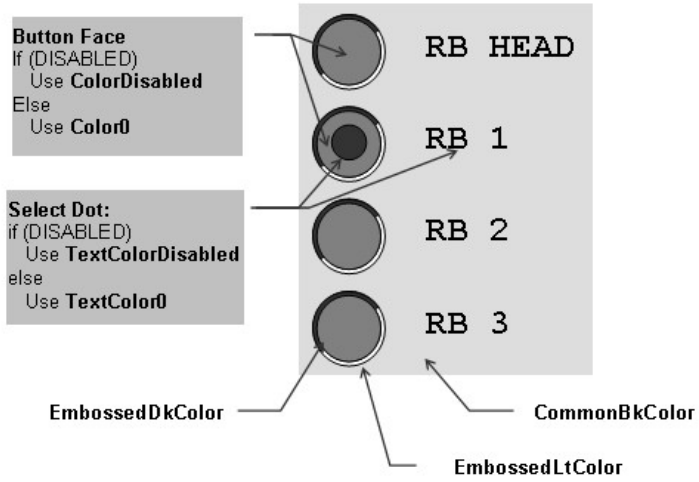
	Name	Description
	GFX_GOL_RadioButtonTextGet	This function returns the address of the current text string used by the object.

Description

Radio Button is an object that can be used to offer set of choices to the user. Only one of the choices is selectable. Changing selection automatically removes the selection on the previous option.

Radio Button supports Keyboard and Touchscreen inputs, replying to their events with the predefined actions (see GFX_GOL_RadioButtonActionGet() and GFX_GOL_RadioButtonActionSet() for details).

The Radio Button object is rendered using the assigned style scheme. The following figure illustrates the usage of the style schemes in the object.



GFX_GOL_RadioButtonTextGet Macro

This function returns the address of the current text string used by the object.

File

gfx_gol_radio_button.h

C

```
#define GFX_GOL_RadioButtonTextGet(pObject) \
                                    (((GFX_GOL_RADIOBUTTON *)pObject)->pText)
```

Returns

Pointer to text string.

Description

GFX GOL radio button text get.

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Example

```
// assume RADIO_BUTTON_OBJECT is a radio button that exists
GFX_XCHAR *pChar;
GFX_GOL_RADIOBUTTON *pRadioButton = &RADIO_BUTTON_OBJECT;
```

```
pChar = GFX_GOL_ButtonRadioTextGet(pRadioButton);
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_XCHAR *GFX_GOL_RadioButtonTextGet(
GFX_GOL_RADIOBUTTON *pObject)

GFX_GOL_RadioButtonCheckGet Function

This function returns the ID of the currently checked radio button in the group.

File

gfx_gol_radio_button.h

C

```
uint16_t GFX_GOL_RadioButtonCheckGet(GFX_GOL_RADIOBUTTON * pObject);
```

Returns

The ID of the member of the group with the check.

Description

GFX GOL radio button check get.

This function returns the ID of the currently checked radio button in the group. When there is only one member of the group, then that member will have the check.

When no member of the group is checked, then the id returned is (-1 or 0xFFFF).

Preconditions

Object must exist in memory.

Example

```
static  GFX_XCHAR   label0[] = "RB1";
static  GFX_XCHAR   label1[] = "RB2";
static  GFX_XCHAR   label2[] = "RB3";
        uint16_t    state;
        GFX_GOL_OBJ_SCHEME  *pScheme;
        RADIOBUTTON *pRb[3];
        uint16_t    ID;

pScheme = GFX_GOL_ObjectSchemeCreate();

// Object will be drawn after creation
// Object will be first button in the group
state = GFX_GOL_RADIOBUTTON_DRAW_STATE |
        GFX_GOL_RADIOBUTTON_CHECKED_STATE;

pRb[0] = GFX_GOL_RadioButtonCreate(
                  gfxIndex,
                  ID_RADIOBUTTON1,
                  255,40,310,80,
                  state,
                  label0,
                  GFX_ALIGN_CENTER,
                  pScheme);

// Object will be drawn after creation
state = GFX_GOL_RADIOBUTTON_DRAW_STATE;

pRb[1] = GFX_GOL_RadioButtonCreate(
                  gfxIndex
                  ID_RADIOBUTTON2,
                  255,85,310,125,
```

```
                    state,
                    label1,
                    GFX_ALIGN_CENTER,
                    pScheme);

    // Object will be drawn after creation
    state = GFX_GOL_RADIOBUTTON_DRAW_STATE;

    pRb[2] = GFX_GOL_RadioButtonCreate(
                    gfxIndex,
                    ID_RADIOBUTTON3,
                    255,130,310,170,
                    state,
                    label2,
                    GFX_ALIGN_CENTER,
                    pScheme);

    // draw the objects
    while(GFX_GOL_ObjectListDraw(gfxIndex) != GFX_STATUS_SUCCESS);

    // can also use pRb[1] or pRb[0] to search the checked
    // radio button of the group. ID here should be ID_RADIOBUTTON1
    ID = GFX_GOL_RadioButtonCheckGet(pRb[2]);

    if (ID == ID_RADIOBUTTON1)
    {
        // do something here then clear the check
GFX_GOL_ObjectStateClear(pRb[0], RB_CHECKED);
// Change the checked object. Pointer used is
        // any of the three. The ID used will find the
        // correct object to be checked
GFX_GOL_RadioButtonCheckSet(pRb[3], ID_RADIOBUTTON2);
    }
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

uint16_t GFX_GOL_RadioButtonCheckGet(
GFX_GOL_RADIOBUTTON *pObject)

GFX_GOL_RadioButtonActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_radio_button.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_RadioButtonActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_RADIOBUTTON_ACTION_CHECKED - Radio Button is checked
- GFX_GOL_OBJECT_ACTION_INVALID - object is not affected

Description

GFX GOL radio button action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_RADIOBUTTON_ACTION_CHECKED	Touch Screen	EVENT_PRESS	If event occurs and the x,y position falls in the area of the Radio Button while the Radio Button is not checked.

	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_CR_PRESSED or SCAN_SPACE_PRESSED while the Radio Button is not checked.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_RadioButtonActionGet(
void *pObject,
GFX_GOL_MESSAGE *pMessage)

GFX_GOL_RadioButtonActionSet Function

This function performs the state change of the object based on the translated action.

File

gfx_gol_radio_button.h

C

```
void GFX_GOL_RadioButtonActionSet(GFX_GOL_TRANSLATED_ACTION translatedMsg, void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

None.

Description

GFX GOL radio button action set.

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit Description
GFX_GOL_RADIOBUTTON_ACTION_CHECKED	Touch Screen,	Set GFX_GOL_RADIOBUTTON_DRAW_STATE, Depending on the current value of RB_CHECKED Check Box will be redrawn.
	Keyboard	Set/Clear GFX_GOL_RADIOBUTTON_CHECKED_STATE

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Function

void GFX_GOL_RadioButtonActionSet(
GFX_GOL_TRANSLATED_ACTION translatedMsg,
void *pObject,
GFX_GOL_MESSAGE *pMessage)

GFX_GOL_RadioButtonCheckSet Function

This function sets the ID of the currently checked radio button in the group.

File

gfx_gol_radio_button.h

C

```
void GFX_GOL_RadioButtonCheckSet(GFX_GOL_RADIOBUTTON * pObject, uint16_t id);
```

Returns

None.

Description

GFX GOL radio button check set.

This function sets the ID of the currently checked radio button in the group. When there is only one member of the group, then that member will have the check.

When the id given does not exist in the group, the function will do nothing.

Preconditions

Object must exist in memory.

Example

See GFX_GOL_RadioButtonCheckGet() example.

Parameters

Parameters	Description
pObject	pointer to the object.
id	id of the member of the group.

Function

GFX_XCHAR GFX_GOL_RadioButtonCheckSet(
GFX_GOL_RADIOBUTTON *pObject,
uint16_t id)

GFX_GOL_RadioButtonDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_radio_button.h

C

```
GFX_STATUS GFX_GOL_RadioButtonDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL radio button draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the GFX_GOL_RADIOBUTTON is drawn with the text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_RadioButtonDraw(void *pObject)

GFX_GOL_RadioButtonTextAlignmentGet Function

This function returns the text alignment of the text string used by the object.

File

gfx_gol_radio_button.h

C

```
GFX_ALIGNMENT GFX_GOL_RadioButtonTextAlignmentGet(GFX_GOL_RADIOBUTTON * pObject);
```

Returns

The text alignment set in the object. See GFX_ALIGNMENT for more details.

Description

GFX GOL radio button text alignment get.

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_ALIGNMENT GFX_GOL_RadioButtonTextAlignmentGet(
GFX_GOL_RADIOBUTTON *pObject)

GFX_GOL_RadioButtonTextAlignmentSet Function

This function sets the text alignment of the text string used by the object.

File

gfx_gol_radio_button.h

C

```
void GFX_GOL_RadioButtonTextAlignmentSet(GFX_GOL_RADIOBUTTON * pObject, GFX_ALIGNMENT align);
```

Returns

None.

Description

GFX GOL radio button text alignment set.

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Function

void GFX_GOL_RadioButtonTextAlignmentSet(
GFX_GOL_RADIOBUTTON *pObject,
GFX_ALIGNMENT align)

GFX_GOL_RadioButtonTextSet Function

This function sets the address of the current text string used by the object.

File

gfx_gol_radio_button.h

C

```
void GFX_GOL_RadioButtonTextSet(GFX_GOL_RADIOBUTTON * pObject, GFX_XCHAR * pText);
```

Returns

None.

Description

GFX GOL radio button text set.

This function sets the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Example

```
GFX_XCHAR Label0[] = â€œONâ€?;
GFX_XCHAR Label1[] = â€œOFFâ€?;
GFX_GOL_RADIOBUTTON GFX_GOL_RADIOBUTTON[2];

GFX_GOL_RadioButtonTextSet(GFX_GOL_RADIOBUTTON[0], Label0);
GFX_GOL_RadioButtonTextSet(GFX_GOL_RADIOBUTTON[1], Label1);
```

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text string to be used.

Function

GFX_XCHAR *GFX_GOL_RadioButtonTextSet(
GFX_GOL_RADIOBUTTON *pObject,
GFX_XCHAR *pText)

GFX_GOL_RadioButtonCreate Function

This function creates a GFX_GOL_RADIOBUTTON object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_radio_button.h

C

```
GFX_GOL_RADIOBUTTON * GFX_GOL_RadioButtonCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left,
uint16_t top, uint16_t right, uint16_t bottom, uint16_t state, GFX_XCHAR * pText, GFX_ALIGNMENT alignment,
GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL Radio button create.

This function creates a GFX_GOL_RADIOBUTTON object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of GFX_GOL_RadioButtonCreate() will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a GFX_GOL_OBJ_SCHEME
- pText is an unterminated string

Preconditions

None.

Example

```
static  GFX_XCHAR   label0[] = "RB1";
static  GFX_XCHAR   label1[] = "RB2";
static  GFX_XCHAR   label2[] = "RB3";
        uint16_t    state;
        GFX_GOL_OBJ_SCHEME  *pScheme;
        RADIOBUTTON *pRb[3];

pScheme = GFX_GOL_ObjectSchemeCreate();

// Object will be drawn after creation
// Object will be first button in the group
state = GFX_GOL_RADIOBUTTON_DRAW_STATE |
        GFX_GOL_RADIOBUTTON_CHECKED_STATE;

pRb[0] = GFX_GOL_RadioButtonCreate(
                  gfxIndex,
                  ID_RADIOBUTTON1,
                  255,40,310,80,
                  state,
                  label0,
                  GFX_ALIGN_CENTER,
                  pScheme);

// Object will be drawn after creation
state = GFX_GOL_RADIOBUTTON_DRAW_STATE;

pRb[1] = GFX_GOL_RadioButtonCreate(
                  gfxIndex,
                  ID_RADIOBUTTON2,
                  255,85,310,125,
                  state,
                  label1,
                  GFX_ALIGN_CENTER,
                  pScheme);

// Object will be drawn after creation
state = GFX_GOL_RADIOBUTTON_DRAW_STATE;

pRb[2] = GFX_GOL_RadioButtonCreate(
                  gfxIndex,
                  ID_RADIOBUTTON3,
                  255,130,310,170,
```

```
                    state,
                    label2,
                    GFX_ALIGN_CENTER,
                    pScheme);

    // draw the objects
    while(GFX_GOL_ObjectListDraw(gfxIndex) != GFX_STATUS_SUCCESS);
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Function

```
GFX_GOL_RADIOBUTTON *GFX_GOL_RadioButtonCreate(
                SYS_MODULE_INDEX    gfxIndex,
uint16_t        ID,
uint16_t        left,
uint16_t        top,
uint16_t        right,
uint16_t        bottom,
uint16_t        state,
                GFX_XCHAR           *pText,
                GFX_ALIGNMENT       alignment,
                GFX_GOL_OBJ_SCHEME  *pScheme)
```

GFX_GOL_RadioButtonListCreate Function

File

gfx_gol_radio_button.h

C

```
GFX_GOL_RADIOBUTTON * GFX_GOL_RadioButtonListCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left,
uint16_t top, uint16_t right, uint16_t bottom, uint16_t state, GFX_XCHAR * pText, GFX_ALIGNMENT alignment,
GFX_GOL_OBJ_SCHEME * pScheme, uint16_t groupNo, uint16_t numOfGroups);
```

Description

This is function GFX_GOL_RadioButtonListCreate.

Scheme Object

Functions

	Name	Description
	GFX_GOL_SchemeBackgroundColorSet	This is function GFX_GOL_SchemeBackgroundColorSet.
	GFX_GOL_SchemeAlphaPrecentSet	This is function GFX_GOL_SchemeAlphaPrecentSet.
	GFX_GOL_SchemeBackgroundColorGet	This is function GFX_GOL_SchemeBackgroundColorGet.
	GFX_GOL_SchemeBackgroundImageSet	This is function GFX_GOL_SchemeBackgroundImageSet.
	GFX_GOL_SchemeBackgroundTypeGet	This is function GFX_GOL_SchemeBackgroundTypeGet.
	GFX_GOL_SchemeBackgroundTypeSet	This is function GFX_GOL_SchemeBackgroundTypeSet.
	GFX_GOL_SchemeColor0Get	This is function GFX_GOL_SchemeColor0Get.
	GFX_GOL_SchemeColor0Set	This is function GFX_GOL_SchemeColor0Set.

	GFX_GOL_SchemeColor1Get	This is function GFX_GOL_SchemeColor1Get.
	GFX_GOL_SchemeColor1Set	This is function GFX_GOL_SchemeColor1Set.
	GFX_GOL_SchemeColorDisabledGet	This is function GFX_GOL_SchemeColorDisabledGet.
	GFX_GOL_SchemeColorDisabledSet	This is function GFX_GOL_SchemeColorDisabledSet.
	GFX_GOL_SchemeColorSet	This is function GFX_GOL_SchemeColorSet.
	GFX_GOL_SchemeEmbossDarkColorGet	This is function GFX_GOL_SchemeEmbossDarkColorGet.
	GFX_GOL_SchemeEmbossDarkColorSet	This is function GFX_GOL_SchemeEmbossDarkColorSet.
	GFX_GOL_SchemeEmbossLightColorGet	This is function GFX_GOL_SchemeEmbossLightColorGet.
	GFX_GOL_SchemeEmbossLightColorSet	This is function GFX_GOL_SchemeEmbossLightColorSet.
	GFX_GOL_SchemeEmbossSet	This is function GFX_GOL_SchemeEmbossSet.
	GFX_GOL_SchemeEmbossSizeGet	This is function GFX_GOL_SchemeEmbossSizeGet.
	GFX_GOL_SchemeEmbossSizeSet	This is function GFX_GOL_SchemeEmbossSizeSet.
	GFX_GOL_SchemeFillStyleGet	This is function GFX_GOL_SchemeFillStyleGet.
	GFX_GOL_SchemeFillStyleSet	This is function GFX_GOL_SchemeFillStyleSet.
	GFX_GOL_SchemeFontGet	This is function GFX_GOL_SchemeFontGet.
	GFX_GOL_SchemeFontSet	This is function GFX_GOL_SchemeFontSet.
	GFX_GOL_SchemeGradientColorSet	This is function GFX_GOL_SchemeGradientColorSet.
	GFX_GOL_SchemeGradientEndColorGet	This is function GFX_GOL_SchemeGradientEndColorGet.
	GFX_GOL_SchemeGradientEndColorSet	This is function GFX_GOL_SchemeGradientEndColorSet.
	GFX_GOL_SchemeGradientStartColorGet	This is function GFX_GOL_SchemeGradientStartColorGet.
	GFX_GOL_SchemeGradientStartColorSet	This is function GFX_GOL_SchemeGradientStartColorSet.
	GFX_GOL_SchemeTextColor0Get	This is function GFX_GOL_SchemeTextColor0Get.
	GFX_GOL_SchemeTextColor0Set	This is function GFX_GOL_SchemeTextColor0Set.
	GFX_GOL_SchemeTextColor1Get	This is function GFX_GOL_SchemeTextColor1Get.
	GFX_GOL_SchemeTextColor1Set	This is function GFX_GOL_SchemeTextColor1Set.
	GFX_GOL_SchemeTextColorDisableGet	This is function GFX_GOL_SchemeTextColorDisableGet.
	GFX_GOL_SchemeTextColorDisableSet	This is function GFX_GOL_SchemeTextColorDisableSet.
	GFX_GOL_SchemeTextColorSet	This is function GFX_GOL_SchemeTextColorSet.

Description

GFX_GOL_SchemeBackgroundColorSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeBackgroundColorSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR color);
```

Description

This is function GFX_GOL_SchemeBackgroundColorSet.

GFX_GOL_SchemeAlphaPrecentSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeAlphaPrecentSet(GFX_GOL_OBJ_SCHEME * pScheme, uint16_t percent);
```

Description

This is function GFX_GOL_SchemeAlphaPrecentSet.

GFX_GOL_SchemeBackgroundColorGet Function

File

gfx_gol_scheme.h

C

```
inline GFX_COLOR GFX_GOL_SchemeBackgroundColorGet(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeBackgroundColorGet.

GFX_GOL_SchemeBackgroundImageSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeBackgroundImageSet(GFX_GOL_OBJ_SCHEME * pScheme, uint16_t left, uint16_t top,
GFX_RESOURCE_HDR * image);
```

Description

This is function GFX_GOL_SchemeBackgroundImageSet.

GFX_GOL_SchemeBackgroundTypeGet Function

File

gfx_gol_scheme.h

C

```
inline GFX_BACKGROUND_TYPE GFX_GOL_SchemeBackgroundTypeGet(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeBackgroundTypeGet.

GFX_GOL_SchemeBackgroundTypeSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeBackgroundTypeSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_BACKGROUND_TYPE type);
```

Description

This is function GFX_GOL_SchemeBackgroundTypeSet.

GFX_GOL_SchemeColor0Get Function

File

gfx_gol_scheme.h

C

```
inline GFX_COLOR GFX_GOL_SchemeColor0Get(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeColor0Get.

GFX_GOL_SchemeColor0Set Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeColor0Set(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR color);
```

Description

This is function GFX_GOL_SchemeColor0Set.

GFX_GOL_SchemeColor1Get Function

File

gfx_gol_scheme.h

C

```
inline GFX_COLOR GFX_GOL_SchemeColor1Get(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeColor1Get.

GFX_GOL_SchemeColor1Set Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeColor1Set(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR color);
```

Description

This is function GFX_GOL_SchemeColor1Set.

GFX_GOL_SchemeColorDisabledGet Function

File

gfx_gol_scheme.h

C

```
inline GFX_COLOR GFX_GOL_SchemeColorDisabledGet(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeColorDisabledGet.

GFX_GOL_SchemeColorDisabledSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeColorDisabledSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR disableColor);
```

Description

This is function GFX_GOL_SchemeColorDisabledSet.

GFX_GOL_SchemeColorSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeColorSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR color0, GFX_COLOR color1,
GFX_COLOR disableColor);
```

Description

This is function GFX_GOL_SchemeColorSet.

GFX_GOL_SchemeEmbossDarkColorGet Function

File

gfx_gol_scheme.h

C

```
inline GFX_COLOR GFX_GOL_SchemeEmbossDarkColorGet(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeEmbossDarkColorGet.

GFX_GOL_SchemeEmbossDarkColorSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeEmbossDarkColorSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR darkColor);
```

Description

This is function GFX_GOL_SchemeEmbossDarkColorSet.

GFX_GOL_SchemeEmbossLightColorGet Function

File

gfx_gol_scheme.h

C

```
inline GFX_COLOR GFX_GOL_SchemeEmbossLightColorGet(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeEmbossLightColorGet.

GFX_GOL_SchemeEmbossLightColorSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeEmbossLightColorSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR lightColor);
```

Description

This is function GFX_GOL_SchemeEmbossLightColorSet.

GFX_GOL_SchemeEmbossSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeEmbossSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR darkColor, GFX_COLOR
lightColor, uint16_t size);
```

Description

This is function GFX_GOL_SchemeEmbossSet.

GFX_GOL_SchemeEmbossSizeGet Function

File

gfx_gol_scheme.h

C

```
inline uint16_t GFX_GOL_SchemeEmbossSizeGet(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeEmbossSizeGet.

GFX_GOL_SchemeEmbossSizeSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeEmbossSizeSet(GFX_GOL_OBJ_SCHEME * pScheme, uint16_t size);
```

Description

This is function GFX_GOL_SchemeEmbossSizeSet.

GFX_GOL_SchemeFillStyleGet Function

File

gfx_gol_scheme.h

C

```
inline GFX_FILL_STYLE GFX_GOL_SchemeFillStyleGet(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeFillStyleGet.

GFX_GOL_SchemeFillStyleSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeFillStyleSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_FILL_STYLE style);
```

Description

This is function GFX_GOL_SchemeFillStyleSet.

GFX_GOL_SchemeFontGet Function

File

gfx_gol_scheme.h

C

```
inline GFX_RESOURCE_HDR* GFX_GOL_SchemeFontGet(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeFontGet.

GFX_GOL_SchemeFontSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeFontSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_RESOURCE_HDR * font);
```

Description

This is function GFX_GOL_SchemeFontSet.

GFX_GOL_SchemeGradientColorSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeGradientColorSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR startColor, GFX_COLOR
endColor);
```

Description

This is function GFX_GOL_SchemeGradientColorSet.

GFX_GOL_SchemeGradientEndColorGet Function

File

gfx_gol_scheme.h

C

```
inline GFX_COLOR GFX_GOL_SchemeGradientEndColorGet(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeGradientEndColorGet.

GFX_GOL_SchemeGradientEndColorSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeGradientEndColorSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR endColor);
```

Description

This is function GFX_GOL_SchemeGradientEndColorSet.

GFX_GOL_SchemeGradientStartColorGet Function

File

gfx_gol_scheme.h

C

```
inline GFX_COLOR GFX_GOL_SchemeGradientStartColorGet(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeGradientStartColorGet.

GFX_GOL_SchemeGradientStartColorSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeGradientStartColorSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR startColor);
```

Description

This is function GFX_GOL_SchemeGradientStartColorSet.

GFX_GOL_SchemeTextColor0Get Function

File

gfx_gol_scheme.h

C

```
inline GFX_COLOR GFX_GOL_SchemeTextColor0Get(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeTextColor0Get.

GFX_GOL_SchemeTextColor0Set Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeTextColor0Set(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR textColor);
```

Description

This is function GFX_GOL_SchemeTextColor0Set.

GFX_GOL_SchemeTextColor1Get Function

File

gfx_gol_scheme.h

C

```
inline GFX_COLOR GFX_GOL_SchemeTextColor1Get(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeTextColor1Get.

GFX_GOL_SchemeTextColor1Set Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeTextColor1Set(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR textColor);
```

Description

This is function GFX_GOL_SchemeTextColor1Set.

GFX_GOL_SchemeTextColorDisableGet Function

File

gfx_gol_scheme.h

C

```
inline GFX_COLOR GFX_GOL_SchemeTextColorDisableGet(GFX_GOL_OBJ_SCHEME * pScheme);
```

Description

This is function GFX_GOL_SchemeTextColorDisableGet.

GFX_GOL_SchemeTextColorDisableSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeTextColorDisableSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR disableColor);
```

Description

This is function GFX_GOL_SchemeTextColorDisableSet.

GFX_GOL_SchemeTextColorSet Function

File

gfx_gol_scheme.h

C

```
inline void GFX_GOL_SchemeTextColorSet(GFX_GOL_OBJ_SCHEME * pScheme, GFX_COLOR textColor0, GFX_COLOR
textColor1, GFX_COLOR disableColor);
```

Description

This is function GFX_GOL_SchemeTextColorSet.

Scroll Bar Object

Scroll Bar is an object that can be used to display a value or scrolling location in a predefined area.

Functions

	Name	Description
	GFX_GOL_ScrollBarActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_ScrollBarActionSet	This function performs the state change of the object based on the translated action.

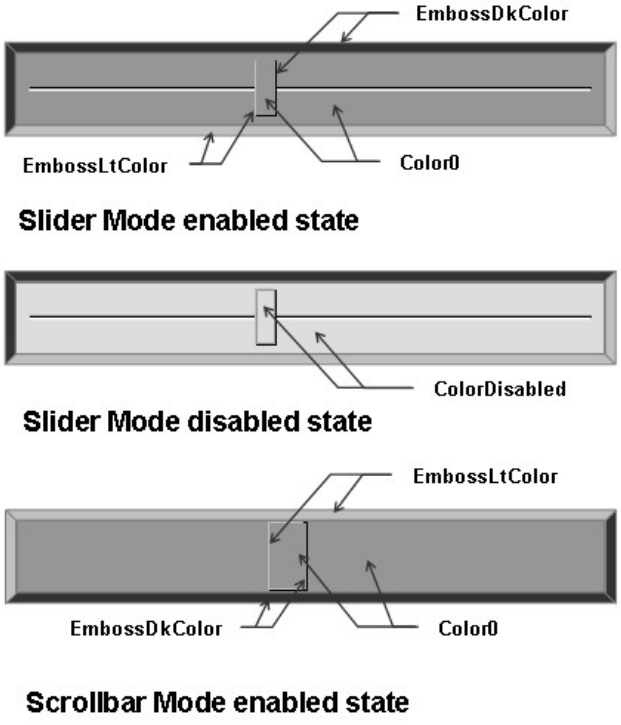
	GFX_GOL_ScrollBarCreate	This function creates a GFX_GOL_SCROLLBAR object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_ScrollBarDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_ScrollBarPositionDecrement	This function decrements the scroll bar position by the delta change (page) value set.
	GFX_GOL_ScrollBarPageGet	This function returns the page size of the object.
	GFX_GOL_ScrollBarPageSet	This function sets the page size of the object.
	GFX_GOL_ScrollBarPositionGet	This function returns the current position of the scroll bar thumb.
	GFX_GOL_ScrollBarPositionIncrement	This function increments the scroll bar position by the delta change (page) value set.
	GFX_GOL_ScrollBarPositionSet	This function sets the position of the scroll bar thumb.
	GFX_GOL_ScrollBarRangeGet	This function returns the range of the thumb of the scroll bar.
	GFX_GOL_ScrollBarRangeSet	This function sets the range of the thumb of the scroll bar.

Description

Scroll Bar is an object that can be used to display a value or scrolling location in a predefined area.

Scroll Bar supports Keyboard and Touchscreen inputs, replying to their events with the predefined actions (see GFX_GOL_ScrollBarActionGet() and GFX_GOL_ScrollBarActionSet() for details).

The Scroll Bar object is rendered using the assigned style scheme. The following figure illustrates the usage of the style schemes in the object.



GFX_GOL_ScrollBarActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_scroll_bar.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_ScrollBarActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_SCROLLBAR_ACTION_INC â€“ Increment scroll bar position.
- GFX_GOL_SCROLLBAR_ACTION_DEC â€“ Decrement scroll bar position.
- GFX_GOL_OBJECT_ACTION_PASSIVE â€“ Object bar is not affected
- GFX_GOL_OBJECT_ACTION_INVALID â€“ Object is not affected

Description

GFX GOL scroll bar action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_SCROLLBAR_ACTION_INC	Touch Screen	EVENT_PRESS, EVENT_MOVE	If events occurs and the x,y position falls in the area of the slider and the slider position is to the LEFT of the x,y position for a horizontal slider or BELOW the x,y position for a vertical slider.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_UP_PRESSED or SCAN_LEFT_PRESSED.
GFX_GOL_SCROLLBAR_ACTION_DEC	Touch Screen	EVENT_PRESS, EVENT_MOVE	If events occurs and the x,y position falls in the area of the slider and the slider position is to the RIGHT of the x,y position for a horizontal slider or ABOVE the x,y position for a vertical slider.
	Keyboard	EVENT_KEYSCAN	If event occurs and parameter1 passed matches the object's ID and parameter 2 passed matches SCAN_DOWN_PRESSED or SCAN_RIGHT_PRESSED.
GFX_GOL_OBJECT_ACTION_PASSIVE	Touch Screen	EVENT_RELEASE	If events occurs and the x,y position falls in the area of the slider.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_ScrollBarActionGet(

void *pObject, GFX_GOL_MESSAGE *pMessage)

GFX_GOL_ScrollBarActionSet Function

This function performs the state change of the object based on the translated action.

File

gfx_gol_scroll_bar.h

C

```
void GFX_GOL_ScrollBarActionSet(GFX_GOL_TRANSLATED_ACTION translatedMsg, void * pObject, GFX_GOL_MESSAGE *
pMessage);
```

Returns

None.

Description

GFX GOL scroll bar action set.

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_SCROLLBAR_ACTION_INC	Touch Screen,	Set GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE	Scroll Bar will be redrawn with thumb in the incremented position.
	Keyboard		
GFX_GOL_SCROLLBAR_ACTION_DEC	Touch Screen,	Set GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE	Scroll Bar will be redrawn with thumb in the decremented position.
	Keyboard		

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Function

void GFX_GOL_ScrollBarActionSet(

GFX_GOL_TRANSLATED_ACTION translatedMsg,

void *pObject,

GFX_GOL_MESSAGE *pMessage)

GFX_GOL_ScrollBarCreate Function

This function creates a GFX_GOL_SCROLLBAR object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_scroll_bar.h

C

```
GFX_GOL_SCROLLBAR * GFX_GOL_ScrollBarCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left, uint16_t top, uint16_t right, uint16_t bottom, uint16_t state, uint16_t range, uint16_t page, uint16_t pos, GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL scroll bar create.

This function creates a GFX_GOL_SCROLLBAR object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The object can be configured as a scroll bar or a slider. Use the state bit GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE to enable the usage of the object as a slider. If this state bit is not enabled, the object is set up as a scroll bar.

The object can also be configured with vertical orientation. Use the state bit GFX_GOL_SCROLLBAR_VERTICAL_STATE to set up the object with vertical orientation. If this state bit is not set, the object is used with horizontal orientation.

The behavior of GFX_GOL_ScrollBarCreate() will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a GFX_GOL_OBJ_SCHEME
- page is set to zero.
- range is set to zero.
- page > range.

Preconditions

None.

Example

```
// assume pScheme is initialized

GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_SCROLLBAR  *srollBarArray[3];
uint16_t           state;

// create a slider with
// range = [0 - 50000]
// delta = 500
// initial position = 25000

state = GFX_GOL_SCROLLBAR_DRAW_STATE;
srollBarArray[0] = GFX_GOL_ScrollBarCreate(
                        gfxIndex, 5,
                        150, 145, 285, 181,
                        state,
                        50000, 500, 25000,
                        pScheme);
if (slider[0] == NULL)
    return 0;

// create a slider with
// range = [0 - 100]
// delta = 20
// initial position = 0

state =  GFX_GOL_SCROLLBAR_DRAW_STATE |
         GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE;
srollBarArray[1] = GFX_GOL_ScrollBarCreate(
                        gfxIndex, 6,
                        150, 190, 285, 220,
                        state,
                        100, 20, 0,
                        pScheme);
if (slider[1] == NULL)
    return 0;

// create a vertical scroll bars with
// range = [0 - 30]
// delta = 2
// initial position = 20

state =  GFX_GOL_SCROLLBAR_DRAW_STATE |
         GFX_GOL_SCROLLBAR_VERTICAL_STATE;
srollBarArray[2] = GFX_GOL_ScrollBarCreate(
                        gfxIndex, 7,
                        120, 145, 140, 220,
                        state,
                        30, 2, 20,
                        pScheme);
if (slider[2] == NULL)
    return 0;

// draw the sliders
while (GFX_GOL_ObjectListDraw() == 0);

return 1;
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.

left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
range	This specifies the maximum value of the slider when the thumb is on the rightmost position for a horizontal orientation and bottom most position for the vertical orientation. Minimum value is always at zero.
page	This is the incremental change of the slider when user action requests to move the slider thumb. This value is added or subtracted to the current position of the thumb.
pos	This defines the initial position of the thumb.
pScheme	The pointer to the style scheme used for the Object. Set to NULL if default style scheme is used.

Function

```
    GFX_GOL_SCROLLBAR  *GFX_GOL_ScrollBarCreate(
                SYS_MODULE_INDEX    gfxIndex,
uint16_t        ID,
uint16_t        left,
uint16_t        top,
uint16_t        right,
uint16_t        bottom,
uint16_t        state,
uint16_t        range,
uint16_t        page,
uint16_t        pos,
                GFX_GOL_OBJ_SCHEME  *pScheme);
```

GFX_GOL_ScrollBarDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_scroll_bar.h

C

```
GFX_STATUS GFX_GOL_ScrollBarDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL scroll bar draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the face of the GFX_GOL_SCROLLBAR is drawn on top of the bitmap. Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_ScrollBarDraw(void *pObject)

GFX_GOL_ScrollBarPositionDecrement Function

This function decrements the scroll bar position by the delta change (page) value set.

File

gfx_gol_scroll_bar.h

C

```
void GFX_GOL_ScrollBarPositionDecrement(GFX_GOL_SCROLLBAR * pObject);
```

Returns

None.

Description

GFX GOL scroll bar position decrement.

This function decrements the scroll bar position by the delta change (page) value set. Object must be redrawn after this function is called to reflect the changes to the object.

Preconditions

Object must exist in memory.

Example

```
void ControlSpeed(  GFX_GOL_SCROLLBAR* pObj,
                    int setSpeed,
                    int curSpeed)
{
    // set page size to 1
    GFX_GOL_ScrollBarPageSet(pObj, 1);

    if (setSpeed < curSpeed)
    {
        while(GFX_GOL_ScrollBarPositionGet(pObj) < SetSpeed)
        {
            // increment by 1
            GFX_GOL_ScrollBarPositionIncrement(pObj);
        }

    }
    else if (setSpeed > curSpeed)
    {
        while(GFX_GOL_ScrollBarPositionGet(pObj) > SetSpeed)
        {
            // decrement by 1
            GFX_GOL_ScrollBarPositionDecrement(pObj);
        }
    }
}
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_ScrollBarPositionDecrement(
GFX_GOL_SCROLLBAR *pObject)

GFX_GOL_ScrollBarPageGet Function

This function returns the page size of the object.

File

gfx_gol_scroll_bar.h

C

```
uint16_t GFX_GOL_ScrollBarPageGet(GFX_GOL_SCROLLBAR * pObject);
```

Returns

The page size of the object.

Description

GFX GOL scroll bar page get.

This function returns the page size of the object. Page size defines the delta change of the thumb position when incremented via GFX_GOL_ScrollBarPositionIncrement() or decremented via GFX_GOL_ScrollBarPositionDecrement(). Page size minimum value is 1. Maximum value is range/2.

Preconditions

Object must exist in memory.

Example

```
uint16_t page;
GFX_GOL_SCROLLBAR *pScrollBar;

// assume pScrollBar is initialized to a scroll bar in memory
 page = GFX_GOL_ScrollBarPageGet(pScrollBar);
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

uint16_t GFX_GOL_ScrollBarPageGet(
GFX_GOL_SCROLLBAR *pObject)

GFX_GOL_ScrollBarPageSet Function

This function sets the page size of the object.

File

gfx_gol_scroll_bar.h

C

```
void GFX_GOL_ScrollBarPageSet(GFX_GOL_SCROLLBAR * pObject, uint16_t page);
```

Returns

None.

Description

GFX GOL scroll bar page set.

This function sets the page size of the object. Page size defines the delta change of the thumb position when incremented via GFX_GOL_ScrollBarPositionIncrement() or decremented via GFX_GOL_ScrollBarPositionDecrement(). Page size minimum value is 1. Maximum value is range/2.

Modifying the page size at run time may require a redraw of the object to show the visual effect of the change.

Preconditions

Object must exist in memory.

Example

See GFX_GOL_ScrollBarPositionIncrement() for code example.

Parameters

Parameters	Description
pObject	pointer to the object.
page	value of the page size of the object.

Function

void GFX_GOL_ScrollBarPageSet(
GFX_GOL_SCROLLBAR *pObject,
uint16_t page)

GFX_GOL_ScrollBarPositionGet Function

This function returns the current position of the scroll bar thumb.

File

gfx_gol_scroll_bar.h

C

```
uint16_t GFX_GOL_ScrollBarPositionGet(GFX_GOL_SCROLLBAR * pObject);
```

Returns

The current position of the scroll bar thumb.

Description

GFX GOL scroll bar position get.

This function returns the current position of the scroll bar thumb. The thumb is the rectangular area that slides left or right (for horizontal orientation) or slides up or down (for vertical orientation).

Preconditions

Object must exist in memory.

Example

```
#define MAXVALUE 100;

GFX_GOL_SCROLLBAR *pScrollBar;
uint32_t ctr = 0;

// create scroll bar here and initialize parameters
pScrollBar = GFX_GOL_ScrollBarCreate(....)
GFX_GOL_ObjectStateSet(pScrollBar, GFX_GOL_SCROLLBAR_DRAW_STATE);

// draw the scroll bar
GFX_GOL_ObjectListDraw(gfxIndex);

// a routine that updates the position of the thumb through some
// conditions
while("some condition")
{
    GFX_GOL_ScrollBarPositionSet(pScrollBar, ctr);
    GFX_GOL_ObjectStateSet( pScrollBar,
                            GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE);

    // update the screen
    GFX_GOL_ObjectListDraw(gfxIndex);

    // update ctr here
    ctr = "some source of value";
}

if (GFX_GOL_ScrollBarPositionGet(pScrollBar) > MAXVALUE)
    return 0;
else
    "do something else"
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

uint16_t GFX_GOL_ScrollBarPositionGet(
GFX_GOL_SCROLLBAR *pObject)

GFX_GOL_ScrollBarPositionIncrement Function

This function increments the scroll bar position by the delta change (page) value set.

File

gfx_gol_scroll_bar.h

C

```
void GFX_GOL_ScrollBarPositionIncrement(GFX_GOL_SCROLLBAR * pObject);
```

Returns

None.

Description

GFX GOL scroll bar position increment.

This function increments the scroll bar position by the delta change (page) value set. Object must be redrawn after this function is called to reflect the changes to the object.

Preconditions

Object must exist in memory.

Example

```
void ControlSpeed(  GFX_GOL_SCROLLBAR* pObj,
                    int setSpeed,
                    int curSpeed)
{
    // set page size to 1
    GFX_GOL_ScrollBarPageSet(pObj, 1);

    if (setSpeed < curSpeed)
    {
        while(GFX_GOL_ScrollBarPositionGet(pObj) < SetSpeed)
            GFX_GOL_ScrollBarPositionIncrement(pObj);  // increment by 1
    }
    else if (setSpeed > curSpeed)
    {
        while(GFX_GOL_ScrollBarPositionGet(pObj) > SetSpeed)
            GFX_GOL_ScrollBarPositionDecrement(pObj);  // decrement by 1
    }
}
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_ScrollBarPositionIncrement(
GFX_GOL_SCROLLBAR *pObject)

GFX_GOL_ScrollBarPositionSet Function

This function sets the position of the scroll bar thumb.

File

gfx_gol_scroll_bar.h

C

```
void GFX_GOL_ScrollBarPositionSet(GFX_GOL_SCROLLBAR * pObject, uint16_t position);
```

Returns

None.

Description

GFX GOL scroll bar position set.

This function sets the position of the scroll bar thumb. The thumb is the rectangular area that slides left or right (for horizontal orientation) or slides up or down (for vertical orientation).

The value used for the position should be within the range set for the object.

Function will have an undefined behavior if the position is outside the range.

Preconditions

Object must exist in memory.

Example

```
GFX_GOL_SCROLLBAR *pScrollBar;
uint16_t ctr = 0;

// create slider here and initialize parameters
GFX_GOL_ObjectStateSet(pScrollBar, GFX_GOL_SCROLLBAR_DRAW_STATE);
GFX_GOL_ObjectListDraw(gfxIndex);

while("some condition")
{
    GFX_GOL_ScrollBarPositionSet(pScrollBar, ctr);
    GFX_GOL_ObjectStateSet( pScrollBar,
                            GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE);

    // redraw the scroll bar
    GFX_GOL_ObjectListDraw(gfxIndex);

   // update ctr here
   ctr = "some source of value";
}
```

Parameters

Parameters	Description
pObject	pointer to the object.
position	the new position of the scroll bar thumb.

Function

void GFX_GOL_ScrollBarPositionSet(

GFX_GOL_SCROLLBAR *pObject,

uint16_t position)

GFX_GOL_ScrollBarRangeGet Function

This function returns the range of the thumb of the scroll bar.

File

gfx_gol_scroll_bar.h

C

```
uint16_t GFX_GOL_ScrollBarRangeGet(GFX_GOL_SCROLLBAR * pObject);
```

Returns

The range of the scroll bar.

Description

GFX GOL scroll bar range get.

This function returns the range of the thumb of the scroll bar.

Preconditions

Object must exist in memory.

Example

```
GFX_GOL_SCROLLBAR *pSld;
uint16_t range;

range = GFX_GOL_ScrollBarRangeGet(pSld);
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

uint16_t GFX_GOL_ScrollBarRangeGet(
GFX_GOL_SCROLLBAR *pObject)

GFX_GOL_ScrollBarRangeSet Function

This function sets the range of the thumb of the scroll bar.

File

gfx_gol_scroll_bar.h

C

```
void GFX_GOL_ScrollBarRangeSet(GFX_GOL_SCROLLBAR * pObject, uint16_t range);
```

Returns

None.

Description

GFX GOL scroll bar range set.

This function sets the range of the thumb of the scroll bar. When the range is modified, object must be completely redrawn to reflect the change.

Preconditions

Object must exist in memory.

Example

```
GFX_GOL_SCROLLBAR *pSld;

GFX_GOL_ScrollBarRangeSet(pSld, 100);

// to completely redraw the object when
// GFX_GOL_ObjectListDraw(gfxIndex) is executed.
GFX_GOL_ObjectStateSet(pSld, SLD_DRAW);
```

Parameters

Parameters	Description
pObject	pointer to the object.
range	new range of the scroll bar.

Function

void GFX_GOL_ScrollBarRangeSet(
GFX_GOL_SCROLLBAR *pObject,
uint16_t range)

Static Text Object

Static Text is an object that can be used to display a single or multi-line string of text in a defined area.

Functions

	Name	Description
	GFX_GOL_StaticTextActionGet	This function sets the text alignment of the text string used by the object.

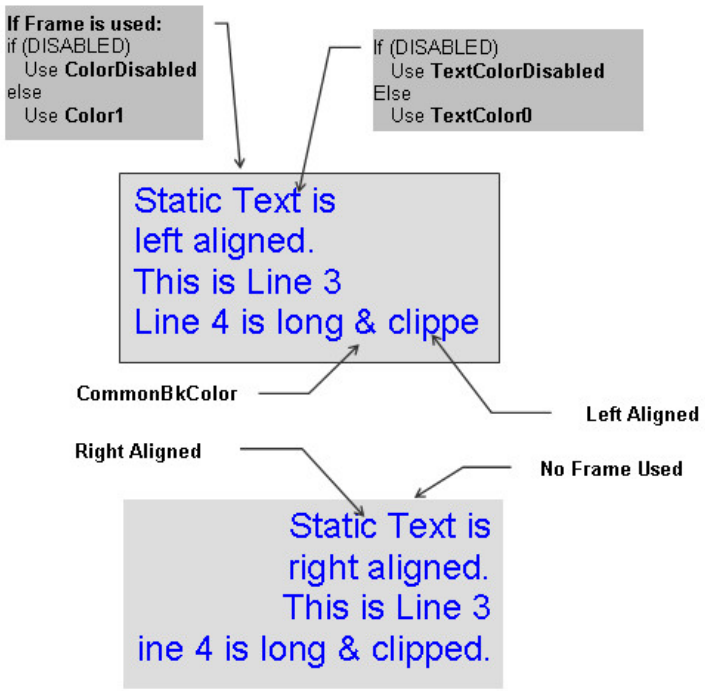
	GFX_GOL_StaticTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_StaticTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_StaticTextCreate	This function creates a GFX_GOL_STATICTEXT object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_StaticTextDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_StaticTextGet	This function returns the address of the current text string used by the object.
	GFX_GOL_StaticTextSet	This function sets the address of the current text string used by the object.

Description

Static Text is an object that can be used to display a single or multi-line string of text in a rectangular area defined by the dimension of the object. The area defined will also serve as the writable region, where any pixels that exceeds the area's dimension will be clipped.

Static Text supports Touchscreen inputs only, replying to the events with the predefined actions (see GFX_GOL_StaticTextActionGet() for details).

The Static Text object is rendered using the assigned style scheme. The following figure illustrates the usage of the style schemes in the object.



GFX_GOL_StaticTextActionGet Function

This function sets the text alignment of the text string used by the object.

File

gfx_gol_static_text.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_StaticTextActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_STATICTEXT_ACTION_SELECTED â€“ Object is selected
- GFX_GOL_OBJECT_ACTION_INVALID â€“ Object is not affected

Description

GFX GOL static text action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen inputs.

Translated Message	Input Source	Events	Description
GFX_GOL_STATICTEXT_ACTION_SELECTED	Touch Screen	EVENT_PRESS, EVENT_RELEASE	If events occurs and the x,y position falls in the area of the static text.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_StaticTextActionGet(
void *pObject,
GFX_GOL_MESSAGE *pMessage);

GFX_GOL_StaticTextAlignmentGet Function

This function returns the text alignment of the text string used by the object.

File

gfx_gol_static_text.h

C

```
GFX_ALIGNMENT GFX_GOL_StaticTextAlignmentGet(GFX_GOL_STATICTEXT * pObject);
```

Returns

The text alignment set in the object. See GFX_ALIGNMENT for more details.

Description

GFX GOL static text alignment get.

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_ALIGNMENT GFX_GOL_StaticTextAlignmentGet(
GFX_GOL_STATICTEXT *pObject)

GFX_GOL_StaticTextAlignmentSet Function

This function sets the text alignment of the text string used by the object.

File

gfx_gol_static_text.h

C

```
void GFX_GOL_StaticTextAlignmentSet(GFX_GOL_STATICTEXT * pObject, GFX_ALIGNMENT align);
```

Returns

None.

Description

GFX GOL static text alignment set.

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.
align	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Function

void GFX_GOL_StaticTextAlignmentSet(

GFX_GOL_STATICTEXT *pObject,

GFX_ALIGNMENT align)

GFX_GOL_StaticTextCreate Function

This function creates a GFX_GOL_STATICTEXT object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_static_text.h

C

```
GFX_GOL_STATICTEXT * GFX_GOL_StaticTextCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left,
uint16_t top, uint16_t right, uint16_t bottom, uint16_t state, GFX_XCHAR * pText, GFX_ALIGNMENT alignment,
GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL static text create.

This function creates a GFX_GOL_STATICTEXT object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The text can be configured to have text aligned. See GFX_ALIGNMENT for details. Text can also have multiple lines by inserting the new line character to the text string supplied to the object. Any string that exceeds the dimension of the object will be clipped.

When the object is used with no background, application must manage the object when text is modified and redrawn. i.e. the previous text must be removed. Use GFX_GOL_STATICTEXT_NOBACKGROUND_STATE state bit to disable the background.

The behavior of GFX_GOL_StaticTextCreate() will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a GFX_GOL_OBJ_SCHEME
- pText is an unterminated string

Preconditions

None.

Example

```
#define ID_STATICTEXT1 0x10

// assume pScheme is initialized
GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_STATICTEXT *pSt;

pSt = GFX_GOL_StaticTextCreate(
    gfxIndex,
    ID_STATICTEXT1,                 // ID
    30,80,235,160,                  // dimension
    GFX_GOL_STATICTEXT_DRAW_STATE,  // draw the object
    "Static Textn Example",        // 2 lines of text
    GFX_ALIGN_CENTER,               // align text on the center
    pScheme);                       // use given scheme
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Function

```
    GFX_GOL_STATICTEXT  *GFX_GOL_StaticTextCreate(
                SYS_MODULE_INDEX    gfxIndex,
uint16_t        ID,
uint16_t        left,
uint16_t        top,
uint16_t        right,
uint16_t        bottom,
uint16_t        state,
                GFX_XCHAR           *pText,
                GFX_ALIGNMENT       alignment,
                GFX_GOL_OBJ_SCHEME  *pScheme)
```

GFX_GOL_StaticTextDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_static_text.h

C

```
GFX_STATUS GFX_GOL_StaticTextDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL static text draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_StaticTextDraw(void *pObject)

GFX_GOL_StaticTextGet Function

This function returns the address of the current text string used by the object.

File

gfx_gol_static_text.h

C

```
GFX_XCHAR * GFX_GOL_StaticTextGet(GFX_GOL_STATICTEXT * pObject);
```

Returns

Pointer to text string.

Description

GFX GOL static text get.

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Example

```
GFX_XCHAR *pChar;
GFX_GOL_STATICTEXT OBJECT_ARRAY[2];

pChar = GFX_GOL_StaticTextGet(OBJECT_ARRAY[0]);
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_XCHAR *GFX_GOL_StaticTextGet(
GFX_GOL_STATICTEXT *pObject)

GFX_GOL_StaticTextSet Function

This function sets the address of the current text string used by the object.

File

gfx_gol_static_text.h

C

```
void GFX_GOL_StaticTextSet(GFX_GOL_STATICTEXT * pObject, GFX_XCHAR * pText);
```

Returns

None.

Description

GFX GOL static text set.

This function sets the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text string to be used.

Function

GFX_XCHAR *GFX_GOL_StaticTextSet(
GFX_GOL_STATICTEXT *pObject,
GFX_XCHAR *pText)

Surface Object

Functions

	Name	Description
	GFX_GOL_SurfaceActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_SurfaceCreate	This function creates a GFX_GOL_SURFACE object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_SurfaceDraw	This function renders the object on the screen based on the current state of the object.

Description

GFX_GOL_SurfaceActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_surface.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_SurfaceActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_SURFACE_ACTION_CLIENT - Surface client area selected action ID.
- GFX_GOL_SURFACE_ACTION_TITLE - Surface title bar selected action ID.

Description

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Events	Description
GFX_GOL_SURFACE_ACTION_TITLE	Touch Screen	EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE	If events occurs and the x,y position falls in the TITLE area of the surface
GFX_GOL_SURFACE_ACTION_CLIENT	Touch Screen	EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE	If events occurs and the x,y position falls in the CLIENT area of the surface
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_SurfaceActionGet(
void *pObject,
GFX_GOL_MESSAGE *pMessage);

GFX_GOL_SurfaceCreate Function

This function creates a GFX_GOL_SURFACE object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_surface.h

C

```
GFX_GOL_SURFACE * GFX_GOL_SurfaceCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left, uint16_t
top, uint16_t right, uint16_t bottom, uint16_t state, GFX_GOL_SURFACE_CALLBACK_FUNC callback,
GFX_RESOURCE_HDR * pImage, GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

This function creates a GFX_GOL_SURFACE object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

Preconditions

None.

Example

```
GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_SURFACE *pSurface;
GFX_GOL_SURFACE_STATE state;

// assume pScheme is initialized to a scheme in memory.
state = GFX_GOL_SURFACE_DRAW_STATE;


pSurface  = GFX_GOL_SurfaceCreate(
       gfxIndex,
       1,           // ID
       0,0,GFX_Primitive_MaxXGet(),GFX_Primitive_MaxYGet(), // whole screen dimension
       state,                   // set state to draw all
       (char*)myIcon,               // icon
       "Place Title Here.",                      // text
       NULL);                   // use default GOL scheme

if (pSurface == NULL)
    return 0;

return 1;
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pImage	Pointer to the image used on the face of the object.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Function

```
    GFX_GOL_SURFACE  *GFX_GOL_SurfaceCreate(
                SYS_MODULE_INDEX    gfxIndex,
uint16_t        ID,
uint16_t        left,
uint16_t        top,
uint16_t        right,
uint16_t        bottom,
uint16_t        state,
                GFX_RESOURCE_HDR    *pImage,
                GFX_XCHAR           *pText,
                GFX_ALIGNMENT       alignment,
                GFX_GOL_OBJ_SCHEME  *pScheme)
```

GFX_GOL_SurfaceDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_surface.h

C

```
GFX_STATUS GFX_GOL_SurfaceDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the face of the GFX_GOL_SURFACE is drawn on top of the bitmap. Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_SurfaceDraw(void *pObject)

Text Entry Object

Text Entry is an object that can be used to emulate a key pad entry with a display area for the entered characters.

Functions

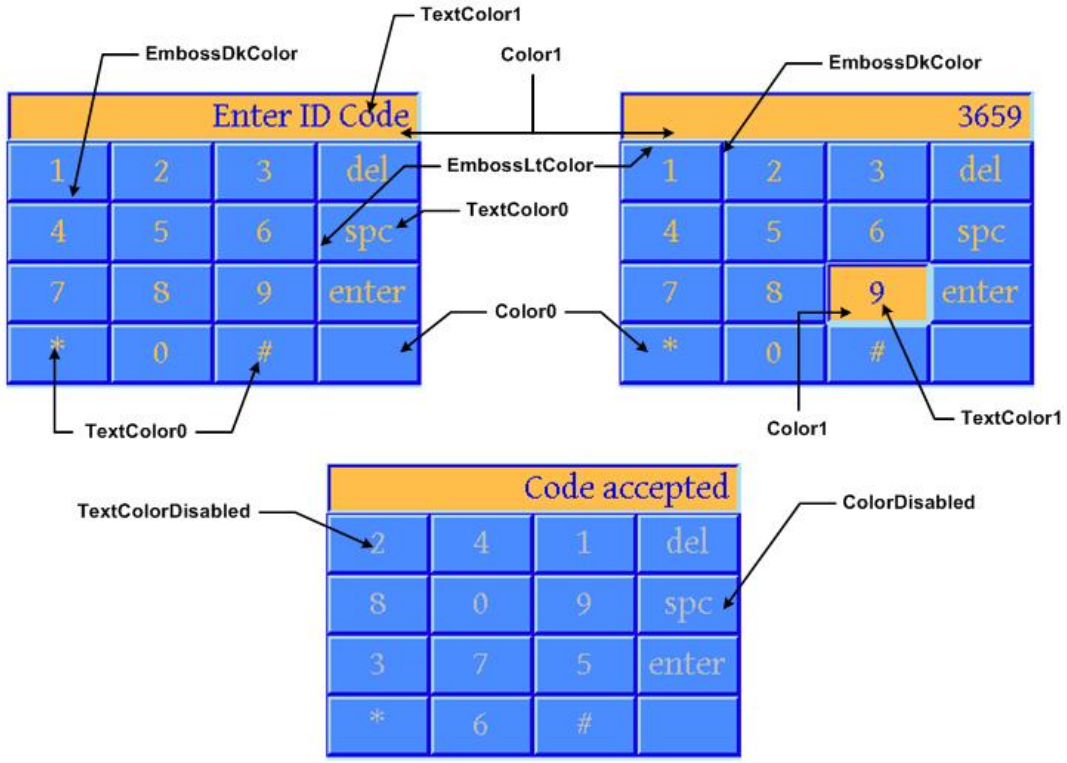
	Name	Description
	GFX_GOL_TextEntryActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_TextEntryActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_TextEntryBufferGet	This function returns the buffer used to display text.
	GFX_GOL_TextEntryBufferClear	This function will clear the data in the display.
	GFX_GOL_TextEntryBufferSet	This function sets the buffer used to display text.
	GFX_GOL_TextEntryCharAdd	This function will insert a character to the end of the buffer.
	GFX_GOL_TextEntryCreate	This function creates a GFX_GOL_TEXTENTRY object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_TextEntryDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_TextEntryKeyCommandGet	This function will return the currently assigned command to the key with the given index.
	GFX_GOL_TextEntryKeyCommandSet	This function will assign a command to a key with the given index.
	GFX_GOL_TextEntryKeyIsPressed	This function will test if a key given by its index in the object is pressed.
	GFX_GOL_TextEntryKeyListCreate	This function will create the list of key members that holds the information on each key.
	GFX_GOL_TextEntryKeyMemberListDelete	This function will delete the key member list assigned to the object.
	GFX_GOL_TextEntryKeyTextSet	This function will set the text assigned to a key with the given index.
	GFX_GOL_TextEntryLastCharDelete	This function will remove the last character of the buffer and replace it with a string terminator.
	GFX_GOL_TextEntrySpaceCharAdd	This function will insert a space character to the end of the buffer.

Description

Text Entry is an object that can be used to emulate a key pad entry with a display area for the entered characters. The object has a feature where you can define a key to reply with a translated message that signifies a command key was pressed. A command key example can be your enter or carriage return key or an escape key. Multiple keys can be assigned command keys. Application can utilize the command key to define the behavior of the program based on a command key press.

Static Text supports Touchscreen inputs only, replying to the events with the predefined actions (see GFX_GOL_TextEntryActionGet() and GFX_GOL_TextEntryActionSet() for details).

The Text Entry object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



When creating the object, the alignment of the text of the object can be formatted with the same options that GFX_TextStringBoxDraw() allows.

GFX_GOL_TextEntryActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_text_entry.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_TextEntryActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_TEXTENTRY_ACTION_PRESS - A key is pressed
- GFX_GOL_TEXTENTRY_ACTION_RELEASED - A key was released (generic for keys with no commands or characters assigned)
- GFX_GOL_TEXTENTRY_ACTION_ADD_CHAR - A key was released with character assigned
- GFX_GOL_TEXTENTRY_ACTION_DELETE - A key was released with delete command assigned
- GFX_GOL_TEXTENTRY_ACTION_SPACE - A key was released with space command assigned
- GFX_GOL_TEXTENTRY_ACTION_ENTER - A key was released with enter command assigned
- GFX_GOL_OBJECT_ACTION_INVALID - Text Entry is not affected

Description

GFX GOL text entry action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Set/Clear State Bit	Description
GFX_GOL_TEXTENTRY_ACTION_PRESS	Touch Screen	EVENT_PRESS, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed.
GFX_GOL_TEXTENTRY_ACTION_RELEASED	Touch Screen	EVENT_MOVE	If the event occurs and the x,y position falls outside the face of one of the keys of the object while the key is pressed.
GFX_GOL_TEXTENTRY_ACTION_RELEASED	Touch Screen	EVENT_RELEASE	If the event occurs and the x,y position falls does not falls inside any of the faces of the keys of the object.

GFX_GOL_TEXTENTRY_ACTION_ADD_CHAR	Touch Screen	EVENT_RELEASE, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed and the key is associated with no commands.
GFX_GOL_TEXTENTRY_ACTION_DELETE	Touch Screen	EVENT_RELEASE, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed and the key is associated with delete command.
GFX_GOL_TEXTENTRY_ACTION_SPACE	Touch Screen	EVENT_RELEASE, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed and the key is associated with space command.
GFX_GOL_TEXTENTRY_ACTION_ENTER	Touch Screen	EVENT_RELEASE, EVENT_MOVE	If the event occurs and the x,y position falls in the face of one of the keys of the object while the key is unpressed and the key is associated with enter command.
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_TextEntryActionGet(
void *pObject,
GFX_GOL_MESSAGE *pMessage);

GFX_GOL_TextEntryActionSet Function

This function performs the state change of the object based on the translated action.

File

gfx_gol_text_entry.h

C

```
void GFX_GOL_TextEntryActionSet(GFX_GOL_TRANSLATED_ACTION translatedMsg, void * pObject, GFX_GOL_MESSAGE *
pMessage);
```

Returns

None.

Description

GFX GOL text entry action set.

This function performs the state change of the object based on the translated action. This change can be overridden by the application using the application defined GFX_GOL_MESSAGE_CALLBACK_FUNC. When the user message is determined to affect the object, application can perform the state change in the message callback function. The following state changes are supported:

Translated Message Input Source	Set/Clear State Bit	Description
GFX_GOL_TEXTENTRY_ACTION_ADD_CHAR Touch Screen,	Set GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE,	Add a character in the buffer and update the text displayed.
GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE,	Clear GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE	
GFX_GOL_TEXTENTRY_ACTION_SPACE Touch Screen,	Set GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE,	Insert a space character in the buffer and update the text displayed.
GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE,	Clear GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE	

GFX_GOL_TEXTENTRY_ACTION_DELETE Touch Screen,	Set GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE,	Delete the most recent character in the buffer and update the text displayed.
GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE,	Clear GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE	
GFX_GOL_TEXTENTRY_ACTION_ENTER Touch Screen,	Set GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE,	User can define the use of this event in the message callback. Object will just update the key.
GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE,	Clear GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE	
GFX_GOL_TEXTENTRY_ACTION_RELEASED Touch Screen,	Clear GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE	A Key in the object will be redrawn in the unpressed state.
	Set GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE	
GFX_GOL_TEXTENTRY_ACTION_PRESSED Touch Screen,	Set GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE	A Key in the object will be redrawn in the pressed state.
GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE		

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
translatedMsg	The action of the object based on the message.
pObject	The pointer to the object whose state will be modified.
pMessage	The pointer to the original message.

Function

void GFX_GOL_TextEntryActionSet(
GFX_GOL_TRANSLATED_ACTION translatedMsg,
void *pObject,
GFX_GOL_MESSAGE *pMessage)

GFX_GOL_TextEntryBufferGet Function

This function returns the buffer used to display text.

File

gfx_gol_text_entry.h

C

```
GFX_XCHAR * GFX_GOL_TextEntryBufferGet(GFX_GOL_TEXTENTRY * pObject);
```

Returns

The pointer to the text buffer used to display text.

Description

GFX GOL text entry buffer get.

This function returns the buffer used to display text.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_XCHAR *GFX_GOL_TextEntryBufferGet(
GFX_GOL_TEXTENTRY *pObject)

GFX_GOL_TextEntryBufferClear Function

This function will clear the data in the display.

File

gfx_gol_text_entry.h

C

```
void GFX_GOL_TextEntryBufferClear(GFX_GOL_TEXTENTRY * pObject);
```

Returns

None.

Description

GFX GOL text entry buffer clear.

This function will clear the data in the display. Object must be redrawn to reflect the change in the buffer. Use the drawing state bit GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE to update the text on the screen.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_TextEntryBufferClear(
GFX_GOL_TEXTENTRY *pObject)

GFX_GOL_TextEntryBufferSet Function

This function sets the buffer used to display text.

File

gfx_gol_text_entry.h

C

```
void GFX_GOL_TextEntryBufferSet(GFX_GOL_TEXTENTRY * pObject, GFX_XCHAR * pText, uint16_t MaxSize);
```

Returns

None.

Description

GFX GOL text entry buffer set.

This function sets the buffer used to display text. If the buffer is initialized with a string, the string must be a null terminated string. If the string length is greater than MaxSize, string will be truncated to MaxSize. pText must point to a valid memory location with size equal to MaxSize. The total number of characters that will be displayed on the object will be MaxSize - 1 since the last character will be the string terminator character.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the new text buffer to be displayed.
maxSize	maximum length of the new buffer to be used.

Function

void GFX_GOL_TextEntryBufferSet(
GFX_GOL_TEXTENTRY *pObject,
GFX_XCHAR *pText,
uint16_t MaxSize)

GFX_GOL_TextEntryCharAdd Function

This function will insert a character to the end of the buffer.

File

gfx_gol_text_entry.h

C

```
void GFX_GOL_TextEntryCharAdd(GFX_GOL_TEXTENTRY * pObject);
```

Returns

None.

Description

GFX GOL text entry char add.

This function will insert a character to the end of the buffer. Drawing states GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE or GFX_GOL_TEXTENTRY_DRAW_STATE must be set to see the effect of the addition.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_TextEntryCharAdd(
GFX_GOL_TEXTENTRY *pObject)

GFX_GOL_TextEntryCreate Function

This function creates a GFX_GOL_TEXTENTRY object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_text_entry.h

C

```
GFX_GOL_TEXTENTRY * GFX_GOL_TextEntryCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left, uint16_t
top, uint16_t right, uint16_t bottom, uint16_t state, uint16_t horizontalKeys, uint16_t verticalKeys,
GFX_XCHAR * pText[], GFX_XCHAR * pBuffer, GFX_ALIGNMENT alignment, uint16_t bufferLength, GFX_RESOURCE_HDR
* pDisplayFont, GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL text entry create.

This function creates a GFX_GOL_TEXTENTRY object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

The behavior of GFX_GOL_TextEntryCreate() will be undefined if one of the following is true:

- left >= right
- top >= bottom
- pScheme is not pointing to a GFX_GOL_OBJ_SCHEME
- horizontal key or vertical key count is 0
- pText is an unterminated string
- pBuffer is initialized to an allocated memory.

Preconditions

None.

Example

```
#define ID_TEXTENTRY 0x20
#define TEBUFFERSIZE 20    // 20 characters

GFX_XCHAR delKey[] = {'d','e','l',0};
GFX_XCHAR spaceKey[] = {'s','p','c',0};
GFX_XCHAR enterKey[] = {'e','n','t','e','r',0};
GFX_XCHAR exitKey[] = {'M','a','i','n',0};
GFX_XCHAR key1[] = {'1',0};
GFX_XCHAR key2[] = {'2',0};
GFX_XCHAR key3[] = {'3',0};
GFX_XCHAR key4[] = {'4',0};
GFX_XCHAR key5[] = {'5',0};
GFX_XCHAR key6[] = {'6',0};
GFX_XCHAR key7[] = {'7',0};
GFX_XCHAR key8[] = {'8',0};
GFX_XCHAR key9[] = {'9',0};
GFX_XCHAR key0[] = {'0',0};
GFX_XCHAR keystar[] = {'*',0};
GFX_XCHAR keypound[] = {'#',0};

GFX_XCHAR *pKeyNames[] = { key1,    key2, key3,     delKey,
                           key4,    key5, key6,     spaceKey,
                           key7,    key8, key9,     enterKey,
                           keystar, key0, keypound, exitKey
                         };

// assume pScheme is initialized
// myFont is a font in memory
GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_TEXTENTRY  *pTe;
GFX_RESOURCE_HDR   *pFont = &myFont;

pTe = GFX_GOL_TextEntryCreate(
    gfxIndex,
    ID_TEXTENTRY,                   // ID
    30,80,235,160,                  // dimension
    GFX_GOL_TEXTENTRY_DRAW_STATE,   // draw the object
    4,                              // number of horizontal keys
    4,                              // number of vertical keys
    pKeyNames,                      // pointer to the array of key names
    "Enter Code",                   // initial text
    GFX_ALIGN_CENTER,               // align text on the center
    TEBUFFERSIZE,                   // size of the buffer for text
    pFont,                          // pointer to the font of the
                                    // displayed text
    pScheme);                       // use given scheme
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
radius	Radius of the rounded edge. When using gradient buttons and radius != 0, emboss size <= radius. If this is not met, the the GFX_GOL_BUTTON face will not have gradient effect.
state	Sets the initial state of the object.
horizontalKeys	Number of horizontal keys
verticalKeys	Number of vertical keys
pText	array of pointer to the custom "text" assigned by the user.
pBuffer	pointer to the buffer that holds the text to be displayed.
alignment	text alignment of the text used in the object.
bufferLength	length of the buffer assigned by the user. The choice of the length should include the string null terminator.
For example	if the bufferLength is set to 3, only two characters can be shown on the object since the last character will be the string terminator character.
pDisplayFont	pointer to the font image to be used on the edit box section of the object.
pScheme	Pointer to the style scheme used.

Function

```
    GFX_GOL_TEXTENTRY *GFX_GOL_TextEntryCreate(
                SYS_MODULE_INDEX    gfxIndex,
uint16_t        ID,
uint16_t        left,
uint16_t        top,
uint16_t        right,
uint16_t        bottom,
uint16_t        state,
uint16_t        horizontalKeys,
uint16_t        verticalKeys,
                GFX_XCHAR           *pText[],
                GFX_XCHAR           *pBuffer,
                GFX_ALIGNMENT       alignment,
uint16_t        bufferLength,
                GFX_RESOURCE_HDR    *pDisplayFont,
                GFX_GOL_OBJ_SCHEME  *pScheme);
```

GFX_GOL_TextEntryDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_text_entry.h

C

```
GFX_STATUS GFX_GOL_TextEntryDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL text entry draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right

and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_TextEntryDraw(void *pObject)

GFX_GOL_TextEntryKeyCommandGet Function

This function will return the currently assigned command to the key with the given index.

File

gfx_gol_text_entry.h

C

```
GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE GFX_GOL_TextEntryKeyCommandGet(GFX_GOL_TEXTENTRY * pObject, uint16_t
index);
```

Returns

Command assigned to the key (See GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE).

Description

GFX GOL text entry key command set.

This function will return the currently assigned command to the key with the given index. (See GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE)

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
index	index or position of the key.

Function

GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE GFX_GOL_TextEntryKeyCommandSet(
GFX_GOL_TEXTENTRY *pObject,
uint16_t index,
GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE command)

GFX_GOL_TextEntryKeyCommandSet Function

This function will assign a command to a key with the given index.

File

gfx_gol_text_entry.h

C

```
bool GFX_GOL_TextEntryKeyCommandSet(GFX_GOL_TEXTENTRY * pObject, uint16_t index,
GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE command);
```

Returns

TRUE - if the assignment was a success. FALSE - if the assignment was not successful.

Description

GFX GOL text entry key command set.

This function will assign a command to a key with the given index. (See GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE)

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
index	index or position of the key.
command	command assigned for the key.

Function

bool GFX_GOL_TextEntryKeyCommandSet(

GFX_GOL_TEXTENTRY *pObject,

uint16_t index,

GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE command)

GFX_GOL_TextEntryKeyIsPressed Function

This function will test if a key given by its index in the object is pressed.

File

gfx_gol_text_entry.h

C

```
bool GFX_GOL_TextEntryKeyIsPressed(GFX_GOL_TEXTENTRY * pObject, uint16_t index);
```

Returns

TRUE - if the key is pressed. FALSE - if the key is not pressed.

Description

GFX GOL text entry key is pressed.

This function will test if a key given by its index in the object is pressed.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
index	index or position of key that is being tested.

Function

bool GFX_GOL_TextEntryKeyIsPressed(

GFX_GOL_TEXTENTRY *pObject,

uint16_t index)

GFX_GOL_TextEntryKeyListCreate Function

This function will create the list of key members that holds the information on each key.

File

gfx_gol_text_entry.h

C

```
GFX_GOL_TEXTENTRY_KEYMEMBER * GFX_GOL_TextEntryKeyListCreate(GFX_GOL_TEXTENTRY * pObject, GFX_XCHAR *
pText[]);
```

Returns

Returns the pointer to the newly created key member list. A NULL is returned if the list is not created successfully.

Description

GFX GOL text entry key list create.

This function will create the list of key members that holds the information on each key. The number of keys is determined by the equation (keycount = verticalKeys*horizontalKeys). The object creates the information holder for each key automatically and assigns each entry in the *pText[] array with the first entry automatically assigned to the key with an index of 1.

The number of entries to *pText[] must be at least equal to keycount. The last key is assigned with an index of keycount-1.

No checking is performed on the length of *pText[] entries to match (verticalKeys*horizontalKeys).

The function behavior is undefined if the pText[] array is less than the keycount value.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text defined by the application.

Function

GFX_GOL_TEXTENTRY_KEYMEMBER *GFX_GOL_TextEntryKeyListCreate(
GFX_GOL_TEXTENTRY *pObject,
GFX_XCHAR *pText[])

GFX_GOL_TextEntryKeyMemberListDelete Function

This function will delete the key member list assigned to the object.

File

gfx_gol_text_entry.h

C

```
void GFX_GOL_TextEntryKeyMemberListDelete(void * pObject);
```

Returns

None.

Description

GFX GOL text entry key member list delete.

This function will delete the key member list assigned to the object from memory. Pointer to the key member list is then initialized to NULL. All memory resources allocated to the key member list is freed.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_TextEntryKeyMemberListDelete(
void *pObject)

GFX_GOL_TextEntryKeyTextSet Function

This function will set the text assigned to a key with the given index.

File

gfx_gol_text_entry.h

C

```
bool GFX_GOL_TextEntryKeyTextSet(GFX_GOL_TEXTENTRY * pObject, uint16_t index, GFX_XCHAR * pText);
```

Returns

TRUE - if the assignment was a success. FALSE - if the assignment was not successful.

Description

GFX GOL text entry key text set.

This function will set the text assigned to a key with the given index.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
index	index or position of the key.
pText	pointer to the text that will be assigned to the key.

Function

bool GFX_GOL_TextEntryKeyTextSet(
GFX_GOL_TEXTENTRY *pObject,
uint16_t index,
GFX_XCHAR *pText)

GFX_GOL_TextEntryLastCharDelete Function

This function will remove the last character of the buffer and replace it with a string terminator.

File

gfx_gol_text_entry.h

C

```
void GFX_GOL_TextEntryLastCharDelete(GFX_GOL_TEXTENTRY * pObject);
```

Returns

None.

Description

GFX GOL text entry last char delete.

This function will remove the last character of the buffer and replace it with a string terminator. Drawing states GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE or GFX_GOL_TEXTENTRY_DRAW_STATE must be set to see the effect of the addition.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_TextEntryLastCharDelete(
GFX_GOL_TEXTENTRY *pObject)

GFX_GOL_TextEntrySpaceCharAdd Function

This function will insert a space character to the end of the buffer.

File

gfx_gol_text_entry.h

C

```
void GFX_GOL_TextEntrySpaceCharAdd(GFX_GOL_TEXTENTRY * pObject);
```

Returns

None.

Description

GFX GOL text entry space char add.

This function will insert a space character to the end of the buffer. Drawing states GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE or GFX_GOL_TEXTENTRY_DRAW_STATE must be set to see the effect of the addition.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

void GFX_GOL_TextEntrySpaceCharAdd(
GFX_GOL_TEXTENTRY *pObject)

Window Object

Window is an object that can be used to encapsulate objects into a group.

Functions

	Name	Description
	GFX_GOL_WindowActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_WindowCreate	This function creates a GFX_GOL_WINDOW object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_WindowDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_WindowTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_WindowTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_WindowTextSet	This function sets the address of the current text string used by the object.

Macros

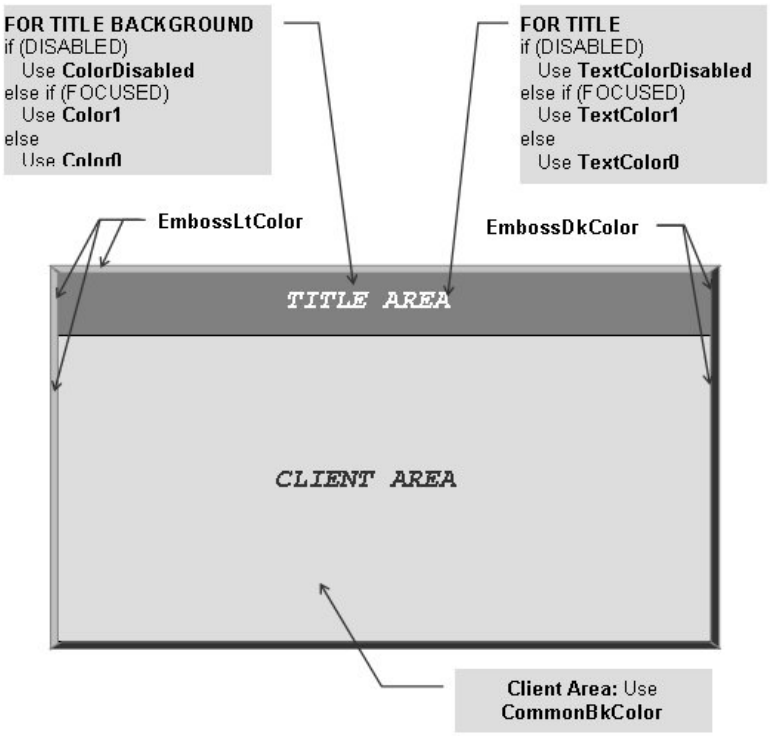
	Name	Description
	GFX_GOL_WindowImageGet	This function gets the image used.
	GFX_GOL_WindowImageSet	This function sets the image used in the object.
	GFX_GOL_WindowTextGet	This function returns the address of the current text string used by the object.

Description

Window is an object that can be used to encapsulate objects into a group. Unlike the Group Box object, the Window object has additional features such as displaying an icon or a small bitmap on its Title Bar. It also has additional controls for both Title Bar and Client Area.

Window supports Touchscreen inputs only, replying to the events with the predefined actions (see GFX_GOL_WindowActionGet() for details).

The Window object is rendered using the assigned style scheme. The following figure illustrates the color assignments.



CommonBkColor – used to hide the window from the screen.

GFX_GOL_WindowImageGet Macro

This function gets the image used.

File

gfx_gol_window.h

C

```
#define GFX_GOL_WindowImageGet(pObject) \
                                (((GFX_GOL_WINDOW *)pObject)->pImage)
```

Returns

Pointer to the image resource.

Description

GFX GOL window image get.

This function gets the image used.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_RESOURCE_HDR *GFX_GOL_WindowImageGet(
GFX_GOL_WINDOW *pObject)

GFX_GOL_WindowImageSet Macro

This function sets the image used in the object.

File

gfx_gol_window.h

C

```
#define GFX_GOL_WindowImageSet(pObject, pImg) \
                                (((GFX_GOL_WINDOW *)pObject)->pImage = pImg)
```

Returns

None.

Description

GFX GOL window image set.

This function sets the image used in the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
pImage	pointer to the image to be set.

Function

void GFX_GOL_WindowImageSet(
GFX_GOL_WINDOW *pObject,
GFX_RESOURCE_HDR *pImage)

GFX_GOL_WindowTextGet Macro

This function returns the address of the current text string used by the object.

File

gfx_gol_window.h

C

```
#define GFX_GOL_WindowTextGet(pObject) (((GFX_GOL_WINDOW *)pObject)->pText)
```

Returns

Pointer to text string.

Description

GFX GOL window text get.

This function returns the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Example

```
GFX_XCHAR *pChar;
GFX_GOL_WINDOW pWindow;

pChar = GFX_GOL_WindowTextGet(pWindow);
```

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_XCHAR *GFX_GOL_WindowTextGet(
GFX_GOL_WINDOW *pObject)

GFX_GOL_WindowActionGet Function

This function evaluates the message from a user if the message will affect the object or not.

File

gfx_gol_window.h

C

```
GFX_GOL_TRANSLATED_ACTION GFX_GOL_WindowActionGet(void * pObject, GFX_GOL_MESSAGE * pMessage);
```

Returns

- GFX_GOL_WINDOW_ACTION_CLIENT - Window client area selected action ID.
- GFX_GOL_WINDOW_ACTION_TITLE - Window title bar selected action ID.

Description

GFX GOL window action get.

This function evaluates the message from a user if the message will affect the object or not. The table below enumerates the action for each event of the touch screen and keyboard inputs.

Translated Message	Input Source	Events	Description
GFX_GOL_WINDOW_ACTION_TITLE	Touch Screen	EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE	If events occurs and the x,y position falls in the TITLE area of the window
GFX_GOL_WINDOW_ACTION_CLIENT	Touch Screen	EVENT_PRESS, EVENT_RELEASE, EVENT_MOVE	If events occurs and the x,y position falls in the CLIENT area of the window
GFX_GOL_OBJECT_ACTION_INVALID	Any	Any	If the message did not affect the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	The pointer to the object where the message will be evaluated to check if the message will affect the object.
pMessage	Pointer to the the message from the user interface.

Function

GFX_GOL_TRANSLATED_ACTION GFX_GOL_WindowActionGet(

void *pObject,

GFX_GOL_MESSAGE *pMessage);

GFX_GOL_WindowCreate Function

This function creates a GFX_GOL_WINDOW object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

File

gfx_gol_window.h

C

```
GFX_GOL_WINDOW * GFX_GOL_WindowCreate(SYS_MODULE_INDEX gfxIndex, uint16_t ID, uint16_t left, uint16_t top,
uint16_t right, uint16_t bottom, uint16_t state, GFX_RESOURCE_HDR * pImage, GFX_XCHAR * pText,
GFX_ALIGNMENT alignment, GFX_GOL_OBJ_SCHEME * pScheme);
```

Returns

Pointer to the newly created object.

Description

GFX GOL window create.

This function creates a GFX_GOL_WINDOW object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.

This function returns the pointer to the newly created object. If the object is not successfully created, it returns NULL.

Preconditions

None.

Example

```
GFX_GOL_OBJ_SCHEME *pScheme;
GFX_GOL_WINDOW *pWindow;
GFX_GOL_WINDOW_STATE state;

// assume pScheme is initialized to a scheme in memory.
state = GFX_GOL_WINDOW_DRAW_STATE;


pWindow  = GFX_GOL_WindowCreate(
       gfxIndex,
       1,           // ID
       0,0,GFX_Primitive_MaxXGet(),GFX_Primitive_MaxYGet(), // whole screen dimension
       state,                   // set state to draw all
       (char*)myIcon,               // icon
       "Place Title Here.",                      // text
       NULL);                   // use default GOL scheme

if (pWindow == NULL)
    return 0;

return 1;
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
ID	Unique user defined ID for the object instance.
left	Left most position of the object.
top	Top most position of the object.
right	Right most position of the object.
bottom	Bottom most position of the object.
state	Sets the initial state of the object.
pImage	Pointer to the image used on the face of the object.
pText	Pointer to the text of the object.
alignment	text alignment of the text used in the object.
pScheme	Pointer to the style scheme used.

Function

```
    GFX_GOL_WINDOW  *GFX_GOL_WindowCreate(
                        SYS_MODULE_INDEX    gfxIndex,
    uint16_t            ID,
    uint16_t            left,
    uint16_t            top,
    uint16_t            right,
    uint16_t            bottom,
    uint16_t            state,
                        GFX_RESOURCE_HDR    *pImage,
                        GFX_XCHAR           *pText,
                        GFX_ALIGNMENT       alignment,
                        GFX_GOL_OBJ_SCHEME  *pScheme)
```

GFX_GOL_WindowDraw Function

This function renders the object on the screen based on the current state of the object.

File

gfx_gol_window.h

C

```
GFX_STATUS GFX_GOL_WindowDraw(void * pObject);
```

Returns

GFX_STATUS_SUCCESS - When the object rendering is finished. GFX_STATUS_FAILURE - When the object rendering is not yet finished. Application needs to call this rendering function again to continue the rendering.

Description

GFX GOL window draw.

This function renders the object on the screen based on the current state of the object. Location of the object is determined by the left, top, right and bottom parameters. The colors used are dependent on the state of the object. The font used is determined by the style scheme set.

The text on the face of the GFX_GOL_WINDOW is drawn on top of the bitmap. Text alignment based on the alignment parameter set on the object.

When rendering objects of the same type, each object must be rendered completely before the rendering of the next object is started. This is to avoid incomplete object rendering.

Normally, application will just call GFX_GOL_ObjectListDraw() to allow the Graphics Library to manage all object rendering. See GFX_GOL_ObjectListDraw() for more information on object rendering.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.

Function

GFX_STATUS GFX_GOL_WindowDraw(void *pObject)

GFX_GOL_WindowTextAlignmentGet Function

This function returns the text alignment of the text string used by the object.

File

gfx_gol_window.h

C

```
GFX_ALIGNMENT GFX_GOL_WindowTextAlignmentGet(GFX_GOL_WINDOW * pObject);
```

Returns

The text alignment set in the object. See GFX_ALIGNMENT for more details.

Description

GFX GOL window text alignment get.

This function returns the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_ALIGNMENT GFX_GOL_WindowTextAlignmentGet(
GFX_GOL_WINDOW *pObject)

GFX_GOL_WindowTextAlignmentSet Function

This function sets the text alignment of the text string used by the object.

File

gfx_gol_window.h

C

```
void GFX_GOL_WindowTextAlignmentSet(GFX_GOL_WINDOW * pObject, GFX_ALIGNMENT alignment);
```

Returns

None.

Description

GFX GOL window text alignment set.

This function sets the text alignment of the text string used by the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	Pointer to the object.
alignment	The alignment set for the text in the object. See GFX_ALIGNMENT for more details.

Function

void GFX_GOL_WindowTextAlignmentSet(
GFX_GOL_WINDOW *pObject,
GFX_ALIGNMENT alignment)

GFX_GOL_WindowTextSet Function

This function sets the address of the current text string used by the object.

File

gfx_gol_window.h

C

```
void GFX_GOL_WindowTextSet(GFX_GOL_WINDOW * pObject, GFX_XCHAR * pText);
```

Returns

None.

Description

GFX GOL window text set.

This function sets the address of the current text string used by the object.

Preconditions

Object must exist in memory.

Example

```
GFX_XCHAR Label0[] = ?ON?;
GFX_XCHAR Label1[] = ?OFF?;
GFX_GOL_WINDOW pWindow;

GFX_GOL_WindowTextSet(pWindow, Label0);
GFX_GOL_WindowTextSet(pWindow, Label1);
```

Parameters

Parameters	Description
pObject	pointer to the object.
pText	pointer to the text string to be used.

Function

void GFX_GOL_WindowTextSet(
GFX_GOL_WINDOW *pObject,
GFX_XCHAR *pText)

GOL Object States

Macros

	Name	Description
	GFX_GOL_ObjectStateClear	This function clears the state bits of the given object.
	GFX_GOL_ObjectStateGet	This function retrieves the current value of the state bits of an object.
	GFX_GOL_ObjectStateSet	This function sets the state bits of the given object.

Description

Graphics Library Object Layer object state API.

GFX_GOL_ObjectStateClear Macro

This function clears the state bits of the given object.

File

gfx_gol.h

C

```
#define GFX_GOL_ObjectStateClear(pObject, stateBits) \
                ((((GFX_GOL_OBJ_HEADER*)pObject)->state) &= (~(stateBits)))
```

Returns

GFX_STATUS_SUCCESS - is returned if the clear was successful. GFX_STATUS_FAILURE - is returned if the clear was not successful.

Description

GFX GOL Object state clear.

This function clears the state bits of the given object. Object must be redrawn to display the changes. It is possible to set several state bits with this function.

Preconditions

None.

Example

See GFX_GOL_ObjectStateSet() for code example.

Parameters

Parameters	Description
pObject	Pointer to the object.
stateBits	Defines which state bits are to be cleared. Please refer to specific objects for object state bits definition for details

Function

GFX_STATUS GFX_GOL_ObjectStateClear(
GFX_GOL_OBJ_HEADER *pObject,
uint16_t stateBits);

GFX_GOL_ObjectStateGet Macro

This function retrieves the current value of the state bits of an object.

File

gfx_gol.h

C

```
#define GFX_GOL_ObjectStateGet(pObject, stateBits) \
                (((GFX_GOL_OBJ_HEADER*)pObject)->state & stateBits)
```

Returns

The current status of the specified state bits.

Description

GFX GOL object state get

This function retrieves the current value of the state bits of an object. It is possible to get several state bits.

Preconditions

None.

Example

```
#define BTN_HIDE 0x8000

GFX_GOL_BUTTON *pB;
// pB is created and initialized
// do something here to set state

// Hide the button (remove from screen)
if (GFX_GOL_ObjectStateGet(pB, GFX_GOL_BUTTON_HIDE_STATE))
{
    GFX_ColorSet(pB->pGolScheme->CommonBkColor);
    GFX_BarDraw(pB->left, pB->top,pB->right,pB->bottom);
}
```

Parameters

Parameters	Description
pObject	Pointer to the object.
stateBits	Defines which state bits are to be retrieved. Please refer to specific objects for object state bits definition for details

Function

uint16_t GFX_GOL_ObjectStateGet(
GFX_GOL_OBJ_HEADER *pObject,
uint16_t stateBits);

GFX_GOL_ObjectStateSet Macro

This function sets the state bits of the given object.

File

gfx_gol.h

C

```
#define GFX_GOL_ObjectStateSet(pObject, stateBits) \
                ((((GFX_GOL_OBJ_HEADER*)pObject)->state) |= stateBits)
```

Returns

GFX_STATUS_SUCCESS - is returned if the set was successful. GFX_STATUS_FAILURE - is returned if the set was not successful.

Description

GFX GOL object state set.

This function sets the state bits of the given object. Object must be redrawn to display the changes. It is possible to set several state bits with this function.

Preconditions

None.

Example

```
void SetMessage(uint16_t msg, GFX_GOL_BUTTON* pB)
{
    switch(msg)
    {
        case GFX_GOL_BUTTON_ACTION_PRESSED:
            // set pressed and redraw
            GFX_GOL_ObjectStateSet(pB, BTN_PRESSED|BTN_DRAW);
            break;
        case GFX_GOL_BUTTON_ACTION_RELEASED:
            // reset pressed
            GFX_GOL_ObjectStateClear(pB, BTN_PRESSED);
            // redraw
            GFX_GOL_ObjectStateSet(pB, BTN_DRAW);
            break;
        default:
            break;
    }
}
```

Parameters

Parameters	Description
pObject	Pointer to the object.
stateBits	Defines which state bits are to be cleared. Please refer to specific objects for object state bits definition for details

Function

GFX_STATUS GFX_GOL_ObjectStateSet(
GFX_GOL_OBJ_HEADER *pObject,
uint16_t stateBits);

GOL Object Management

Functions

	Name	Description
	GFX_GOL_ObjectAdd	This function adds an object to the tail of the currently active list.
	GFX_GOL_ObjectByIDDelete	This function removes an object with the given user defined ID from the currently active list.
	GFX_GOL_ObjectCanBeFocused	Checks if the object can be focused.
	GFX_GOL_ObjectDelete	This function removes an object from the currently active list.
	GFX_GOL_ObjectFind	This function returns the pointer to object in the list with the user defined ID assigned to it.

	GFX_GOL_ObjectFocusGet	This function returns the pointer to the object that is currently receiving keyboard input (or focused).
	GFX_GOL_ObjectFocusNextGet	This function returns the pointer to the next object in the active list of objects which can be focused.
	GFX_GOL_ObjectFocusPrevGet	This function returns the pointer to the previous object in the active list of objects which can be focused.
	GFX_GOL_ObjectFocusSet	This function sets the object to be focused.
	GFX_GOL_ObjectIDGet	This function returns the object ID.
	GFX_GOL_ObjectListFree	This function sets the active list to the new list.
	GFX_GOL_ObjectListGet	This function returns the current active list.
	GFX_GOL_ObjectListNew	This function removes an object with the given user defined ID from the currently active list.
	GFX_GOL_ObjectListSet	This function sets the active list to the new list.
	GFX_GOL_ObjectNextGet	This function returns the pointer to next object in the list after the specified object.
	GFX_GOL_ObjectTypeGet	This function returns the object type.

Description

Graphics Library Object Layer object management API.

GFX_GOL_ObjectAdd Function

This function adds an object to the tail of the currently active list.

File

gfx_gol.h

C

```
void GFX_GOL_ObjectAdd(SYS_MODULE_INDEX gfxIndex, GFX_GOL_OBJ_HEADER * pObject);
```

Returns

None.

Description

GFX GOL object add.

This function adds an object to the tail of the currently active list. The new list tail is set to point to NULL after the new object is added.

Preconditions

None.

Example

```
void MoveObject(    GFX_GOL_OBJ_HEADER *pSrcList,
                    GFX_GOL_OBJ_HEADER *pDstList,
                    GFX_GOL_OBJ_HEADER *pObjtoMove)
{
    GFX_GOL_OBJ_HEADER *pTemp = pSrcList;

    if(pTemp != pObjtoMove)
    {
        while(pTemp->pNxtObj != pObjtoMove)
            pTemp = pTemp->pNxtObj;
    }

    pTemp->pNxtObj = pObjtoMove->pNxt;  // remove object from list
    GFX_GOL_ObjectListSet(pDstList);    // destination as active list
    GFX_GOL_ObjectAdd(gfxIndex, pObjtoMove);      // add object to active list
}
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance. Pointer to the object that will be added to the list.

Function

void GFX_GOL_ObjectAdd(SYS_MODULE_INDEX gfxIndex,

GFX_GOL_OBJ_HEADER *pObject)

GFX_GOL_ObjectByIDDelete Function

This function removes an object with the given user defined ID from the currently active list.

File

gfx_gol.h

C

```
GFX_STATUS GFX_GOL_ObjectByIDDelete(SYS_MODULE_INDEX gfxIndex, uint16_t id);
```

Returns

GFX_STATUS_SUCCESS - is returned if the removal was successful. GFX_STATUS_FAILURE - is returned if the removal was not successful.

Description

GFX GOL object by ID delete.

This function removes an object with the given user defined ID from the currently active list. Aside from the removal of the object from the list, the RAM resources consumed by the object is also freed.

If there is no object with the given ID, the function exits with GFX_STATUS_FAILURE.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance. Pointer to the object that will removed from the list.

Function

GFX_STATUS GFX_GOL_ObjectByIDDelete(SYS_MODULE_INDEX gfxIndex,

uint16_t id)

GFX_GOL_ObjectCanBeFocused Function

Checks if the object can be focused.

File

gfx_gol.h

C

```
GFX_STATUS GFX_GOL_ObjectCanBeFocused(SYS_MODULE_INDEX gfxIndex, GFX_GOL_OBJ_HEADER * pObject);
```

Returns

GFX_STATUS_SUCCESS - when the object can be focused GFX_STATUS_FAILURE - when the object cannot be focused or do not support the focus feature.

Description

GFX GOL object can be focused.

This function checks if the object can be focused or not. If the object can be focused, it returns GFX_STATUS_SUCCESS. If it cannot be focused, it returns GFX_STATUS_FAILURE. Selected objects have the focus feature. Refer to the object documentation for details.

Objects that do not support focus feature will ignore any focus settings.

If the object is disabled it cannot be set to focused state.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
pObject	pointer to the object.

Function

GFX_STATUS GFX_GOL_ObjectCanBeFocused(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_OBJ_HEADER *pObject)

GFX_GOL_ObjectDelete Function

This function removes an object from the currently active list.

File

gfx_gol.h

C

```
GFX_STATUS GFX_GOL_ObjectDelete(SYS_MODULE_INDEX gfxIndex, GFX_GOL_OBJ_HEADER * pObject);
```

Returns

GFX_STATUS_SUCCESS - is returned if the removal was successful. GFX_STATUS_FAILURE - is returned if the removal was not successful.

Description

GFX GOL object delete.

This function removes an object from the currently active list. Aside from the removal of the object from the list, the RAM resources consumed by the object is also freed.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance. Pointer to the object that will removed from the list.

Function

GFX_STATUS GFX_GOL_ObjectDelete(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_OBJ_HEADER *pObject)

GFX_GOL_ObjectFind Function

This function returns the pointer to object in the list with the user defined ID assigned to it.

File

gfx_gol.h

C

```
GFX_GOL_OBJ_HEADER * GFX_GOL_ObjectFind(SYS_MODULE_INDEX gfxIndex, uint16_t ID);
```

Returns

The pointer to the object in the list with the given ID.

Description

GFX GOL object find.

This function returns the pointer to object in the list with the user defined ID assigned to it.

Preconditions

None.

Example

```
void CopyObject(GFX_GOL_OBJ_HEADER *pSrcList,
                GFX_GOL_OBJ_HEADER *pDstList,
                uint16_t ID)
{
    GFX_GOL_OBJ_HEADER *pTemp;

    // find the object
    pTemp = GFX_GOL_ObjectFind(gfxIndex, ID);

    if (pTemp != NULL)
    {
        // destination as active list

        GFX_GOL_ObjectListSet(gfxIndex, pDstList);

        // add object to active list
        GFX_GOL_ObjectAdd(gfxIndex, pTemp);
    }
}
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance. User defined ID of the object of interest.

Function

GFX_GOL_OBJ_HEADER *GFX_GOL_ObjectFind(SYS_MODULE_INDEX gfxIndex, uint16_t ID)

GFX_GOL_ObjectFocusGet Function

This function returns the pointer to the object that is currently receiving keyboard input (or focused).

File

gfx_gol.h

C

```
GFX_GOL_OBJ_HEADER * GFX_GOL_ObjectFocusGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The pointer the currently focused object. Returns NULL if there is no object currently set.

Description

GFX GOL object focus get.

This function returns the pointer to the object that is currently receiving keyboard input (or focused).

If there are no object that can accept keyboard messages, then the function will return NULL.

Objects that can be focused are those objects that can receive keyboard inputs.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.

Function

GFX_GOL_OBJ_HEADER *GFX_GOL_ObjectFocusGet(SYS_MODULE_INDEX gfxIndex)

GFX_GOL_ObjectFocusNextGet Function

This function returns the pointer to the next object in the active list of objects which can be focused.

File

gfx_gol.h

C

```
GFX_GOL_OBJ_HEADER * GFX_GOL_ObjectFocusNextGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The pointer to the object that can be focused.

Description

GFX GOL object focus next get.

This function returns the pointer to the next object in the active list of objects which can be focused.

The reference point is the currently focused object. If there is no currently focused object, the searched starts from the beginning of the active list of objects.

Objects that can be focused are those objects that can receive keyboard inputs.

If there is no object capable of receiving keyboard inputs (i.e. none can be focused) NULL is returned.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.

Function

GFX_GOL_OBJ_HEADER *GFX_GOL_ObjectFocusNextGet(
SYS_MODULE_INDEX gfxIndex)

GFX_GOL_ObjectFocusPrevGet Function

This function returns the pointer to the previous object in the active list of objects which can be focused.

File

gfx_gol.h

C

```
GFX_GOL_OBJ_HEADER * GFX_GOL_ObjectFocusPrevGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The pointer to the object that can be focused.

Description

GFX GOL object focus previous get.

This function returns the pointer to the previous object in the active list of objects which can be focused.

The reference point is the currently focused object. If there is no currently focused object, the searched starts from the beginning of the active list of objects.

Objects that can be focused are those objects that can receive keyboard inputs.

If there is no object capable of receiving keyboard inputs (i.e. none can be focused) NULL is returned.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.

Function

GFX_GOL_OBJ_HEADER *GFX_GOL_ObjectFocusPrevGet(
SYS_MODULE_INDEX gfxIndex)

GFX_GOL_ObjectFocusSet Function

This function sets the object to be focused.

File

gfx_gol.h

C

```
GFX_STATUS GFX_GOL_ObjectFocusSet(SYS_MODULE_INDEX gfxIndex, GFX_GOL_OBJ_HEADER * pObject);
```

Returns

GFX_STATUS_SUCCESS - when the object can be focused GFX_STATUS_FAILURE - when the object cannot be focused or do not support the focus feature.

Description

GFX GOL object focus set.

This function sets the specified object to be focused.

If the object cannot accept keyboard messages, the object will not be set to focused state. If the object can accept keyboard messages, then the focus state will be set and will be marked to be redrawn to show the focus when the focus feature is enabled.

Objects that can be focused are those objects that can receive keyboard inputs.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
pObject	pointer to the object.

Function

GFX_STATUS *GFX_GOL_ObjectFocusSet(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_OBJ_HEADER *pObject)

GFX_GOL_ObjectIDGet Function

This function returns the object ID.

File

gfx_gol.h

C

```
uint16_t GFX_GOL_ObjectIDGet(GFX_GOL_OBJ_HEADER * pObject);
```

Returns

The user defined ID of the object.

Description

GFX GOL object ID get.

This function returns the user defined ID assigned to the object.

Preconditions

None.

Example

```
void ExampleUsageOfGettingID(GFX_GOL_OBJ_HEADER *pObject)
{
```

```
        uint16_t id;

        switch(id = GFX_GOL_ObjectIDGet(pObject))
        {
            case ID_WINDOW1:
                // do something
            case ID_WINDOW2:
                // do something else
            case ID_WINDOW3:
                // do something else
            default:
                // do something else
        }
    }
```

Function

uint16_t GFX_GOL_ObjectIDGet(GFX_GOL_OBJ_HEADER *pObject)

GFX_GOL_ObjectListFree Function

This function sets the active list to the new list.

File

gfx_gol.h

C

```
GFX_STATUS GFX_GOL_ObjectListFree(SYS_MODULE_INDEX gfxIndex);
```

Returns

GFX_STATUS_SUCCESS - is returned if the free was successful. GFX_STATUS_FAILURE - is returned if the free was not successful.

Description

GFX GOL Object List free.

This function frees all the memory used by objects in the active list and initializes the active list pointer to NULL to start a new empty list. This function must be called only inside the GFX_GOL_ObjectDrawCallback() function when using GFX_GOL_ObjectListDraw() and GFX_GOL_ObjectMessage() functions. This requirement assures that primitive rendering settings are not altered by the rendering state machines of the objects.

Preconditions

None.

Example

```
    void DeletePage(GFX_GOL_OBJ_HEADER *pPage)
    {
        GFX_GOL_OBJ_HEADER *pTemp;

        // assuming pPage is different from the current active list
        // save the active list
pTemp = GFX_GOL_ObjectListGet(gfxIndex);

        // set list as active list
        GFX_GOL_ObjectListSet(gfxIndex, pPage);

        // pPage objects are deleted
GFX_GOL_ObjectListFree(gfxIndex);

        // restore the active list
GFX_GOL_ObjectListSet(gfxIndex, pTemp);
    }
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.

Function

GFX_STATUS GFX_GOL_ObjectListFree(SYS_MODULE_INDEX gfxIndex)

GFX_GOL_ObjectListGet Function

This function returns the current active list.

File

gfx_gol.h

C

```
GFX_GOL_OBJ_HEADER * GFX_GOL_ObjectListGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

Pointer (type GFX_GOL_OBJ_HEADER) to the current active list.

Description

GFX GOL object list get

This function returns the pointer to the current active.

Preconditions

None.

Example

See GFX_GOL_ObjectLlistNew() for example code.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.

Function

GFX_GOL_OBJ_HEADER *GFX_GOL_ObjectListGet(SYS_MODULE_INDEX gfxIndex)

GFX_GOL_ObjectListNew Function

This function removes an object with the given user defined ID from the currently active list.

File

gfx_gol.h

C

```
GFX_STATUS GFX_GOL_ObjectListNew(SYS_MODULE_INDEX gfxIndex);
```

Returns

GFX_STATUS_SUCCESS - is returned if the new list start was successful. GFX_STATUS_FAILURE - is returned if the new list start was not successful.

Description

GFX GOL object list new

This function starts a new linked list of objects and resets the keyboard focus to none. This function assigns the current active list and current focused object (receiving keyboard inputs) object pointers to NULL. Any keyboard inputs at this point will be ignored.

This function does not erase the objects in the previous list. Application must save the previous list to another pointer if to be referenced later. If not needed anymore, memory used by that list should be freed by GFX_GOL_ObjectListFree() function. In this case, freeing the list with GFX_GOL_ObjectListFree() function has the same effect as GFX_GOL_ObjectListNew() where the current active list is empty.

Preconditions

None.

Example

```
// assume pointers to objects (pButton, pWindow and pSlider
// are initialized to objects already created
// GFX_GOL_OBJ_HEADER *pButton;
// GFX_GOL_OBJ_HEADER *pWindow;
// GFX_GOL_OBJ_HEADER *pSlider;

GFX_GOL_OBJ_HEADER *pSave;
```

```
// save current list
pSave = GFX_GOL_ObjectListGet(gfxIndex);

// start the new list, after the start of the list, the
// current active list is empty.
GFX_GOL_ObjectListNew(gfxIndex);

// assume that objects are already created
// you can now add objects to the new list
GFX_GOL_ObjectAdd(gfxIndex, pButton);
GFX_GOL_ObjectAdd(gfxIndex, pWindow);
GFX_GOL_ObjectAdd(gfxIndex, pSlider);
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.

Function

GFX_STATUS GFX_GOL_ObjectListNew(SYS_MODULE_INDEX gfxIndex)

GFX_GOL_ObjectListSet Function

This function sets the active list to the new list.

File

gfx_gol.h

C

```
GFX_STATUS GFX_GOL_ObjectListSet(SYS_MODULE_INDEX gfxIndex, GFX_GOL_OBJ_HEADER * pList);
```

Returns

GFX_STATUS_SUCCESS - is returned if the set was successful. GFX_STATUS_FAILURE - is returned if the set was not successful.

Description

GFX GOL object list set.

This function sets the active list to the new list. The previous list will still exist in memory. Application must save the previous list before the set is called if the previous list will be referenced later. If the previous list is not needed anymore, then the list must be removed from memory by GFX_GOL_ObjectListFree() function.

Setting the active list to the new list will reset the focused pointer object to NULL.

Preconditions

None.

Example

```
GFX_GOL_OBJ_HEADER *pSave;

// save current list
pSave = GFX_GOL_ObjectListSet(gfxIndex);

// start the new list
GFX_GOL_ObjectListNew(gfxIndex);

// you can now add objects to the current list
// assume that objects are already created
GFX_GOL_ObjectAdd(gfxIndex, pButton);
GFX_GOL_ObjectAdd(gfxIndex, pWindow);
GFX_GOL_ObjectAdd(gfxIndex, pSlider);

// do something here on the new list

// return the old list
GFX_GOL_ObjectListSet(gfxIndex, pSave);
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance. Pointer to the new list.

Function

GFX_STATUS *GFX_GOL_ObjectListSet(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_OBJ_HEADER *pList)

GFX_GOL_ObjectNextGet Function

This function returns the pointer to next object in the list after the specified object.

File

gfx_gol.h

C

```
GFX_GOL_OBJ_HEADER * GFX_GOL_ObjectNextGet(SYS_MODULE_INDEX gfxIndex, GFX_GOL_OBJ_HEADER * pObject);
```

Returns

The pointer to the next object in the list.

Description

GFX GOL object next get

This function returns the pointer to next object in the list after the specified object.

Preconditions

None.

Example

```
void RedrawButtons(void)
{

    GFX_GOL_OBJ_HEADER *pCurr;

    // get active list
    pCurr = GFX_GOL_ObjectListGet(gfxIndex);

    while(pCurr->pNxtObj != NULL)
    {
        // just select button objects and redraw them
        if (GFX_GOL_ObjectTypeGet(gfxIndex, pCurr) == BUTTON)
        {
            // set to be redrawn
            pCurr->state = BTN_DRAW;
        }
        pCurr = GFX_GOL_ObjectNextGet(gfxIndex, pCurr);
    }
    // redraw all buttons in the active list
    GFX_GOL_ObjectListDraw(gfxIndex);
}
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance. Pointer to the object of interest.

Function

GFX_GOL_OBJ_HEADER *GFX_GOL_ObjectNextGet(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_OBJ_HEADER *pObject)

GFX_GOL_ObjectTypeGet Function

This function returns the object type.

File

gfx_gol.h

C

```
GFX_GOL_OBJ_TYPE GFX_GOL_ObjectTypeGet(SYS_MODULE_INDEX gfxIndex, GFX_GOL_OBJ_HEADER * pObject);
```

Returns

The type of the object. The type is one of the defined enumerated types of GFX_GOL_OBJ_TYPE.

Description

GFX GOL object type get.

This function returns the object type. The object type is one of the defined enumerated types of GFX_GOL_OBJ_TYPE.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance. Pointer to the object of interest.

Function

GFX_GOL_OBJ_TYPE GFX_GOL_ObjectTypeGet(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_OBJ_HEADER *pObject)

GOL Object Rendering

Functions

	Name	Description
	GFX_GOL_DrawCallbackSet	This function sets the draw callback function that the application will use to render application specific rendering.
	GFX_GOL_ObjectDrawEnable	This function sets the object to be redraw.
	GFX_GOL_ObjectDrawDisable	This function resets the drawing state bits of the object.
	GFX_GOL_ObjectIsRedrawSet	This function checks if the object needs to be redrawn or not.
	GFX_GOL_ObjectListDraw	This function redraws all objects in the current active list that has the rendering state bits set.
	GFX_GOL_ObjectListHide	This function marks all objects in the active list to be hidden.
	GFX_GOL_ObjectRectangleRedraw	This function marks all objects in the active list intersected by the given rectangular area to be redrawn.
	GFX_GOL_ObjectHideDraw	This function performs the hiding of an object from the screen.
	GFX_GOL_TwoTonePanelDraw	This function renders the two-tone panel.
	GFX_GOL_ObjectBackGroundSet	This function sets the background information.

Description

Graphics Library Object Layer object rendering API.

GFX_GOL_DrawCallbackSet Function

This function sets the draw callback function that the application will use to render application specific rendering.

File

gfx_gol.h

C

```
void GFX_GOL_DrawCallbackSet(SYS_MODULE_INDEX gfxIndex, GFX_GOL_DRAW_CALLBACK_FUNC pFunc);
```

Returns

None.

Description

GFX GOL draw call back set.

This function sets the draw callback function that the application will use to call primitive function to implement application specific shapes.See GFX_GOL_DRAW_CALLBACK_FUNC definition for details on the draw callback function.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
pFunc	pointer to the draw callback function.

Function

void GFX_GOL_DrawCallbackSet(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_DRAW_CALLBACK_FUNC pFunc)

GFX_GOL_ObjectDrawEnable Function

This function sets the object to be redraw.

File

gfx_gol.h

C

```
void GFX_GOL_ObjectDrawEnable(SYS_MODULE_INDEX gfxIndex, GFX_GOL_OBJ_HEADER * pObject);
```

Returns

None.

Description

GFX GOL object draw enable.

This function sets the object to be redrawn. For the redraw to be effective, the object must be in the current active list. If not, the redraw action will not be performed until the list where the object is currently inserted will be set as the active list.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
pObject	pointer to the object that will be redrawn.

Function

void GFX_GOL_ObjectDrawEnable(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_OBJ_HEADER *pObject)

GFX_GOL_ObjectDrawDisable Function

This function resets the drawing state bits of the object.

File

gfx_gol.h

C

```
void GFX_GOL_ObjectDrawDisable(GFX_GOL_OBJ_HEADER * pObject);
```

Returns

None.

Description

GFX GOL object draw disable.

This function resets the drawing state bits of the object. This function can be called to cancel any drawing state bits that has been set or clears all the drawing state bits after the object has been redrawn.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
pObject	pointer to the object.

Function

void GFX_GOL_ObjectDrawDisable(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_OBJ_HEADER *pObject)

GFX_GOL_ObjectIsRedrawSet Function

This function checks if the object needs to be redrawn or not.

File

gfx_gol.h

C

```
bool GFX_GOL_ObjectIsRedrawSet(GFX_GOL_OBJ_HEADER * pObject);
```

Returns

true - when the object needs to be redrawn. false - when the object does not need to be redrawn.

Description

GFX GOL object is redraw set.

This function checks if the object needs to be redrawn or not. The function returns true if it is to be redrawn or false if it is not to be redrawn.

Preconditions

None.

Example

```
int DrawButtonWindowOnly()
{
    static GFX_GOL_OBJ_HEADER *pCurrentObj = NULL;
    uint16_t done = 0;

    if (pCurrentObj == NULL)
    {
        // get current list
       pCurrentObj = GFX_GOL_ObjectListGet(gfxIndex);
    }

    while(pCurrentObj != NULL)
    {
        if(GFX_GOL_ObjectIsRedrawSet(gfxIndex, pCurrentObj) == true)
        {
            done = pCurrentObj->draw(gfxIndex, pCurrentObj);

            // reset state of object if done
            if (done)
                GOLDrawComplete(pCurrentObj)
```

```
                    // Return if not done. This means that Button Draw function
                    // was not able to finish redrawing the object
                    // and must be called again to finish rendering of
                    // objects in the list that have new states.
                else
                    return 0;
            }
            // go to the next object in the list
            pCurrentObj = pCurrentObj->pNxtObj;
        }
        return 1;
    }
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
pObject	pointer to the object that will be checked.

Function

bool GFX_GOL_ObjectIsRedrawSet(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_OBJ_HEADER *pObject)

GFX_GOL_ObjectListDraw Function

This function redraws all objects in the current active list that has the rendering state bits set.

File

gfx_gol.h

C

```
GFX_STATUS GFX_GOL_ObjectListDraw(SYS_MODULE_INDEX gfxIndex);
```

Returns

GFX_STATUS_SUCCESS - is returned when the active list is completely parsed and redrawn. GFX_STATUS_BUSY - is returned when the active list is not completely parsed and redrawn.

Description

GFX GOL object list draw.

This function loops through the active list and redraws objects that need to be redrawn. Partial redrawing or full redraw is performed depending on the drawing states of the objects.

GFX_GOL_ObjectDrawCallback() function is called by GFX_GOL_ObjectListDraw() when drawing of objects in the active list is completed. GFX_GOL_ObjectDrawCallback() is an application implemented function that allows the application the opportunity to insert application specific rendering using Primitive Layer rendering functions.

The GFX_GOL_ObjectListDraw() function can return with GFX_STATUS_BUSY. In this case, it indicates that the currently redrawn object is not able to continue. Application needs to call GFX_GOL_ObjectListDraw() again to continue the redraw of the objects in the list.

Preconditions

None.

Example

```
    // Assume objects are created & states are set to draw objects
    while(1)
    {
        // parse active list and redraw objects
        // that needs to be redrawn
if( GFX_GOL_ObjectListDraw(gfxIndex) == GFX_STATUS_SUCCESS)
        {
            // at this point drawing is completed
            // it is safe to modify objects states and linked list

            // evaluate messages from touch screen device
            TouchGetMsg(&msg);

            // evaluate each object is affected by the message
            GFX_GOL_ObjectMessage(gfxIndex, &msg);
        }
```

```
}
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.

Function

GFX_STATUS GFX_GOL_ObjectListDraw(SYS_MODULE_INDEX gfxIndex)

GFX_GOL_ObjectListHide Function

This function marks all objects in the active list to be hidden.

File

gfx_gol.h

C

```
void GFX_GOL_ObjectListHide(SYS_MODULE_INDEX gfxIndex);
```

Returns

None.

Description

GFX GOL object list hide.

This function marks all objects in the active list to be hidden.

After calling this function, the next call to GFX_GOL_ObjectListDraw() will hide all objects.

Preconditions

The objects must be using the same background information.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.

Function

void GFX_GOL_ObjectListHide(SYS_MODULE_INDEX gfxIndex)

GFX_GOL_ObjectRectangleRedraw Function

This function marks all objects in the active list intersected by the given rectangular area to be redrawn.

File

gfx_gol.h

C

```
void GFX_GOL_ObjectRectangleRedraw(SYS_MODULE_INDEX gfxIndex, uint16_t left, uint16_t top, uint16_t right,
uint16_t bottom);
```

Returns

None.

Description

GFX GOL object rectangle redraw.

This function marks all objects in the active list intersected by the given rectangular area to be redrawn.

After calling this function, the next call to GFX_GOL_ObjectListDraw() will redraw all objects that are marked for redraw.

Preconditions

None.

Example

```
GFX_GOL_OBJ_HEADER *pTemp;
```

```
GFX_GOL_OBJ_HEADER *pAllObjects;

// assume *pAllObjects points to a list of all existing objects
// created and initialized

// mark all objects inside the rectangle to be redrawn
GFX_GOL_ObjectRectangleRedraw(gfxIndex,10,10,100,100);

// save the current active list
pTemp = pAllObjects;

// reset active list
GFX_GOL_ObjectListNew(gfxIndex);

// build the new active list with only those objects that
// are marked to be redrawn
while(pTemp->pNxtObj != NULL)
{
    if (pTemp->state&0x7C00)
        GFX_GOL_ObjectAdd(gfxIndex, pTemp);

    pTemp = pTemp->pNxtObj;
}

 // redraw active list
GFX_GOL_ObjectListDraw(gfxIndex);
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
left	Defines the left most border of the rectangle area.
top	Defines the top most border of the rectangle area.
right	Defines the right most border of the rectangle area.
bottom	Defines the bottom most border of the rectangle area.

Function

void GFX_GOL_ObjectRectangleRedraw(SYS_MODULE_INDEX gfxIndex,

uint16_t left,

uint16_t top,

uint16_t right,

uint16_t bottom)

GFX_GOL_ObjectHideDraw Function

This function performs the hiding of an object from the screen.

File

gfx_gol.h

C

```
GFX_STATUS GFX_GOL_ObjectHideDraw(SYS_MODULE_INDEX gfxIndex, GFX_GOL_OBJ_HEADER * pObject);
```

Returns

GFX_STATUS_SUCCESS - when the hiding is done. GFX_STATUS_FAILURE - when the hiding is not yet done.

Description

GFX GOL object hide draw.

This function performs the hiding of an object from the screen. If the object's style scheme is set to have a background, the background is taken into account.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
pObject	pointer to the object that will be checked.

Function

GFX_STATUS GFX_GOL_ObjectHideDraw(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_OBJ_HEADER *pObject)

GFX_GOL_TwoTonePanelDraw Function

This function renders the two-tone panel.

File

gfx_gol.h

C

```
GFX_STATUS GFX_GOL_TwoTonePanelDraw(SYS_MODULE_INDEX gfxIndex);
```

Returns

GFX_STATUS_SUCCESS - when the panel rendering is done. GFX_STATUS_FAILURE - when the panel rendering is not yet done.

Description

GFX GOL two tone panel draw.

This function renders the two-tone panel. Panel parameters are set by the GFX_GOL_PanelParameterSet(). The function returns success (GFX_STATUS_SUCCESS) when the panel is rendered. If the function returned not success this function must be called again until success is returned.

Preconditions

Panel parameters must be set first using GFX_GOL_PanelParameterSet().

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.

Function

GFX_STATUS GFX_GOL_TwoTonePanelDraw(SYS_MODULE_INDEX gfxIndex)

GFX_GOL_ObjectBackGroundSet Function

This function sets the background information.

File

gfx_gol.h

C

```
void GFX_GOL_ObjectBackGroundSet(SYS_MODULE_INDEX gfxIndex, GFX_GOL_OBJ_HEADER * pObjectHeader);
```

Returns

None.

Description

GFX GOL object back ground set.

This function sets the background information. This is an internal function and should not be called by the application. This function is used by object's drawing functions to set the background information.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
pObjectHeader	the object header of the object that needs to draw a panel with background.

Function

GFX_STATUS GFX_GOL_ObjectBackGroundSet(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_OBJ_HEADER *pObjectHeader)

GOL Object Messaging

Functions

	Name	Description
	GFX_GOL_ObjectMessage	This function process the received message from the user to determine the affected objects. Depending on the message and the affected objects, object states are modified based on the default or user-defined behavior.
	GFX_GOL_MessageCallbackSet	This function sets the message callback function that the application will use to evaluate user inputs that affects the objects and application behavior.

Description

Graphics Library Object Layer object messaging API.

GFX_GOL_ObjectMessage Function

This function process the received message from the user to determine the affected objects. Depending on the message and the affected objects, object states are modified based on the default or user-defined behavior.

File

gfx_gol.h

C

```
void GFX_GOL_ObjectMessage(SYS_MODULE_INDEX gfxIndex, GFX_GOL_MESSAGE * pMsg);
```

Returns

None.

Description

GFX GOL object message.

This function receives a GFX_GOL_MESSAGE message from user and loops through the active list of objects to check which object is affected by the message. For affected objects the message is translated and GFX_GOL_ObjectMessageCallback() is called. In the call back function, user has the ability to implement action for the message. If the call back function returns non-zero, OBJMsgDefault() is called to process message for the object by default. If zero is returned OBJMsgDefault() is not called. Please refer to GOL Messages section for details.

Preconditions

None.

Example

```
// Assume objects are created & states are set to draw objects
while (1)
{
    if(GOLDraw())
    {
        // GOL drawing is completed here
        // it is safe to change objects

        // from user interface module
```

```
            TouchGetMsg(&msg);
            // process the message
            GFX_GOL_ObjectMessage(gfxIndex, &msg);
        }
    }
```

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
pMessage	Pointer to the message from user.

Function

void GFX_GOL_ObjectMessage(SYS_MODULE_INDEX gfxIndex, GFX_GOL_MESSAGE *pMessage)

GFX_GOL_MessageCallbackSet Function

This function sets the message callback function that the application will use to evaluate user inputs that affects the objects and application behavior.

File

gfx_gol.h

C

```
void GFX_GOL_MessageCallbackSet(SYS_MODULE_INDEX gfxIndex, GFX_GOL_MESSAGE_CALLBACK_FUNC pFunc);
```

Returns

None.

Description

GFX GOL message call back set.

This function sets the message callback function that the application will use to evaluate user inputs that affects the objects and application behavior. The callback function location is specified by the function pointer supplied in the call. See GFX_GOL_MESSAGE_CALLBACK_FUNC definition for details on the message callback function.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.
pFunc	pointer to the message callback function.

Function

void GFX_GOL_MessageCallbackSet(SYS_MODULE_INDEX gfxIndex,
GFX_GOL_MESSAGE_CALLBACK_FUNC pFunc)

GOL Object Style Scheme

Macros

	Name	Description
	GFX_GOL_ObjectStyleSchemeGet	This function returns the style scheme currently set for the object.
	GFX_GOL_ObjectStyleSchemeSet	This function sets the style scheme of the object.

Description

Graphics Library Object Layer object style scheme API.

GFX_GOL_ObjectStyleSchemeGet Macro

This function returns the style scheme currently set for the object.

File

gfx_gol.h

C

```
#define GFX_GOL_ObjectStyleSchemeGet(pObject) \
                (((GFX_GOL_OBJ_HEADER*)pObject)->pGolScheme)
```

Returns

The pointer to the currently set style scheme.

Description

GFX GOL object style scheme get.

This function returns the style scheme currently used by the object. The object must exist when this function is called. This function do not check if the object is valid.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_GOL_OBJ_SCHEME *GFX_GOL_ObjectStyleSchemeGet(
GFX_GOL_OBJ_HEADER *pObject)

GFX_GOL_ObjectStyleSchemeSet Macro

This function sets the style scheme of the object.

File

gfx_gol.h

C

```
#define GFX_GOL_ObjectStyleSchemeSet(pObject, pStyle) \
                ((((GFX_GOL_OBJ_HEADER*)pObject)->pGolScheme) = pStyle)
```

Returns

None.

Description

GFX GOL object style scheme set.

This function sets the style scheme of the object to the given style scheme. The object and style scheme must exist at the time of the assignment. This function do not check if the object or the style is valid.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
pStyle	pointer to the style scheme.

Function

void GFX_GOL_ObjectStyleSchemeSet(
GFX_GOL_OBJ_HEADER *pObject,

GFX_GOL_OBJ_SCHEME *pStyle)

Data Types and Constants

Enumerations

	Name	Description
	GFX_GOL_BUTTON_STATE	Specifies the different states of the Button object.
	GFX_GOL_CHECKBOX_STATE	Specifies the different states of the Check Box object.
	GFX_GOL_DIGITALMETER_STATE	Specifies the different states of the Digital Meter object.
	GFX_GOL_GROUPBOX_STATE	Specifies the different states of the Group Box object.
	GFX_GOL_LISTBOX_ITEM_STATUS	Defines the types used to indicate the status of an item.
	GFX_GOL_LISTBOX_STATE	Specifies the different states of the List Box object.
	GFX_GOL_METER_DRAW_TYPE	Specifies the different types of Meter object.
	GFX_GOL_METER_STATE	Specifies the different states of the Meter object.
	GFX_GOL_OBJ_TYPE	Specifies the different object types used in the library.
	GFX_GOL_PROGRESSBAR_STATE	Specifies the different states of the Progress Bar object.
	GFX_GOL_RADIOBUTTON_STATE	Specifies the different states of the Radio Button object.
	GFX_GOL_SCROLLBAR_STATE	Specifies the different states of the Scroll Bar object.
	GFX_GOL_STATICTEXT_STATE	Specifies the different states of the Static Text object.
	GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE	Specifies the different commands available for command keys of the Text Entry object.
	GFX_GOL_TEXTENTRY_STATE	Specifies the different states of the Text Entry object.
	GFX_GOL_TRANSLATED_ACTION	Specifies the different object actions supported in the library.
	GFX_GOL_WINDOW_STATE	Specifies the different states of the Window object.
	GFX_GOL_EDITBOX_STATE	Specifies the different states of the Edit Box object.
	GFX_GOL_PICTURECONTROLCONTROL_STATE	Specifies the different states of the Picture Control object.
	GFX_GOL_COMMON_STATE_BITS	Common Object States.
	GFX_GOL_CUSTOMCONTROL_STATE	Specifies the different states of the Custom Control object.
	GFX_GOL_CLIENT_STATUS	Enumerated data type that identifies the GFX Module Client Status.
	GFX_GOL_OBJECT_TASK	Lists the different states that GFX GOL task routine can have.
	GFX_GOL_STATES	Defines the various states that can be achieved by the GFX GOL operation.
	GFX_AT_SCAN_CODES	The following macros are the supported AT Keyboard scan codes.
	GFX_GOL_SURFACE_STATE	Specifies the different states of the Surface object.

Macros

	Name	Description
	GFX_GOL_SurfaceImageGet	This function gets the image used.
	GFX_GOL_SurfaceImageSet	This function sets the image used in the object.

Structures

	Name	Description
	GFX_GOL_BUTTON	Defines the structure used for the Button object.
	GFX_GOL_CHECKBOX	Defines the structure used for the Check Box object.
	GFX_GOL_CUSTOMCONTROL	Defines the structure used for the Custom Control object.
	GFX_GOL_DIGITALMETER	Defines the structure used for the Digital Meter object.
	GFX_GOL_EDITBOX	Defines the structure used for the Edit Box object.
	GFX_GOL_GROUPBOX	Defines the structure used for the Group Box object.
	GFX_GOL_LISTBOX	Defines the structure used for the List Box object.
	GFX_GOL_LISTITEM	The structure that defines each item in the list box.
	GFX_GOL_MESSAGE	Specifies message structure used in the library.
	GFX_GOL_METER	Defines the structure used for the Meter object.
	GFX_GOL_OBJ_HEADER	Specifies Graphics Object Layer structure used in objects.
	GFX_GOL_PROGRESSBAR	Defines the structure used for the Progress Bar object.
	GFX_GOL_RADIOBUTTON	Defines the structure used for the Radio Button object.
	GFX_GOL_SCROLLBAR	Defines the structure used for the Scroll Bar object.

	GFX_GOL_STATICTEXT	Defines the structure used for the Static Text object.
	GFX_GOL_TEXTENTRY	Defines the structure used for the Text Entry object.
	GFX_GOL_WINDOW	Defines the structure used for the Window object.
	GOL_PANEL_PARAM	Specifies panel parameters.
	GFX_GOL_PICTURECONTROL	Defines the structure used for the Picture Control object.
	GFX_GOL_TEXTENTRY_KEYMEMBER	Defines the structure used to describe a key in the Text Entry object.
	_GFX_OBJ	Defines the object required for interaction with the GFX Library.
	GFX_GOL_SURFACE	Defines the structure used for the Surface object.
	GFX_GOL_OBJ_SCHEME	This structure specifies the style scheme components of an object.

Types

	Name	Description
	GFX_GOL_DRAW_CALLBACK_FUNC	Draw callback function definition. This application defined function allows the application to perform application specific rendering.
	GFX_GOL_MESSAGE_CALLBACK_FUNC	Message callback function definition. This application defined function allows the application to perform application specific processing of user messages.
	GFX_GOL_SURFACE_CALLBACK_FUNC	Surface callback function definition. This application defined function allows the application to perform application specific rendering.

Description

Graphics Library Object Layer data types and constants.

GFX_GOL_BUTTON Structure

Defines the structure used for the Button object.

File

gfx_gol_button.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  uint16_t radius;
  uint16_t textWidth;
  uint16_t textHeight;
  GFX_XCHAR * pText;
  GFX_ALIGNMENT alignment;
  GFX_RESOURCE_HDR * pPressImage;
  GFX_RESOURCE_HDR * pReleaseImage;
} GFX_GOL_BUTTON;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
uint16_t radius;	Radius for rounded buttons.
uint16_t textWidth;	Computed text width, done at creation.
uint16_t textHeight;	Computed text height, done at creation.
GFX_XCHAR * pText;	Pointer to the text used.
GFX_ALIGNMENT alignment;	text alignment
GFX_RESOURCE_HDR * pPressImage;	Pointer to bitmap used.
GFX_RESOURCE_HDR * pReleaseImage;	Pointer to bitmap used.

Description

GFX GOL button

Typedef: GFX_GOL_BUTTON

Defines the structure used for the Button object.

1. Width is determined by right - left parameter in GFX_GOL_OBJ_HEADER.
2. Height is determined by top - bottom parameter in GFX_GOL_OBJ_HEADER.
3. radius - specifies if the GFX_GOL_BUTTON will have a rounded edge. If zero then the GFX_GOL_BUTTON will have sharp (cornered) edge.
4. If 2*radius = height = width, the GFX_GOL_BUTTON is a circular GFX_GOL_BUTTON.

Remarks

None.

GFX_GOL_BUTTON_STATE Enumeration

Specifies the different states of the Button object.

File

gfx_gol_button.h

C

```
typedef enum {
  GFX_GOL_BUTTON_FOCUSED_STATE,
  GFX_GOL_BUTTON_DISABLED_STATE,
  GFX_GOL_BUTTON_PRESSED_STATE,
  GFX_GOL_BUTTON_TOGGLE_STATE,
  GFX_GOL_BUTTON_TWOTONE_STATE,
  GFX_GOL_BUTTON_NOPANEL_STATE,
  GFX_GOL_BUTTON_DRAW_FOCUS_STATE,
  GFX_GOL_BUTTON_DRAW_STATE,
  GFX_GOL_BUTTON_HIDE_STATE
} GFX_GOL_BUTTON_STATE;
```

Members

Members	Description
GFX_GOL_BUTTON_FOCUSED_STATE	Property bit for focus state.
GFX_GOL_BUTTON_DISABLED_STATE	Property bit for disabled state.
GFX_GOL_BUTTON_PRESSED_STATE	Property bit for press state.
GFX_GOL_BUTTON_TOGGLE_STATE	Property bit to indicate object will have a toggle behavior.
GFX_GOL_BUTTON_TWOTONE_STATE	Property bit to indicate the object is a two tone type.
GFX_GOL_BUTTON_NOPANEL_STATE	Property bit to indicate the object will be drawn without a panel (for faster drawing when the object image used is larger than the object's panel).
GFX_GOL_BUTTON_DRAW_FOCUS_STATE	Draw bit to indicate focus must be redrawn.
GFX_GOL_BUTTON_DRAW_STATE	Draw bit to indicate object must be redrawn.
GFX_GOL_BUTTON_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

GFX GOL button state.

Typedef: GFX_GOL_BUTTON_STATE

This enumeration specifies the different states of the Button object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_CHECKBOX Structure

Defines the structure used for the Check Box object.

File

gfx_gol_check_box.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  uint16_t textHeight;
  GFX_XCHAR * pText;
  GFX_ALIGNMENT alignment;
} GFX_GOL_CHECKBOX;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
uint16_t textHeight;	Pre-computed text height
GFX_XCHAR * pText;	Pointer to text
GFX_ALIGNMENT alignment;	text alignment

Description

GFX GOL checkbox

Typedef: GFX_GOL_CHECKBOX

Defines the structure used for the Check Box object. Object is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_CHECKBOX_STATE Enumeration

Specifies the different states of the Check Box object.

File

gfx_gol_check_box.h

C

```
typedef enum {
  GFX_GOL_CHECKBOX_FOCUSED_STATE,
  GFX_GOL_CHECKBOX_DISABLED_STATE,
  GFX_GOL_CHECKBOX_CHECKED_STATE,
  GFX_GOL_CHECKBOX_DRAW_CHECK_STATE,
  GFX_GOL_CHECKBOX_DRAW_FOCUS_STATE,
  GFX_GOL_CHECKBOX_DRAW_STATE,
  GFX_GOL_CHECKBOX_HIDE_STATE
} GFX_GOL_CHECKBOX_STATE;
```

Members

Members	Description
GFX_GOL_CHECKBOX_FOCUSED_STATE	Property bit for focus state.
GFX_GOL_CHECKBOX_DISABLED_STATE	Property bit to indicate object is disabled.
GFX_GOL_CHECKBOX_CHECKED_STATE	Property bit to indicate object is checked.
GFX_GOL_CHECKBOX_DRAW_CHECK_STATE	Draw bit to indicate the check must be redrawn.
GFX_GOL_CHECKBOX_DRAW_FOCUS_STATE	Draw bit to indicate focus must be redrawn.
GFX_GOL_CHECKBOX_DRAW_STATE	Draw bit to indicate object must be redrawn.
GFX_GOL_CHECKBOX_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

GFX GOL checkbox state.

Typedef: GFX_GOL_CHECKBOX_STATE

This enumeration specifies the different states of the Check Box object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_CUSTOMCONTROL Structure

Defines the structure used for the Custom Control object.

File

gfx_gol_custom_control.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  uint8_t instance;
  uint16_t ID;
  void * pNxtObj;
  GFX_GOL_OBJ_TYPE type;
  uint16_t state;
  uint16_t left;
  uint16_t top;
  uint16_t right;
  uint16_t bottom;
  GFX_GOL_OBJ_SCHEME * pGolScheme;
  DRAW_FUNC draw;
  FREE_FUNC FreeObj;
  ACTIONGET_FUNC actionGet;
  ACTIONSET_FUNC actionSet;
  uint16_t pos;
  uint16_t prevPos;
} GFX_GOL_CUSTOMCONTROL;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_OBJ_HEADER).
uint8_t instance;	instance index
uint16_t ID;	instance unique ID
void * pNxtObj;	a pointer to the next object in the linked list
GFX_GOL_OBJ_TYPE type;	must be set to OBJ_CUSTOM
uint16_t state;	state
uint16_t left;	left border
uint16_t top;	top border
uint16_t right;	right border
uint16_t bottom;	bottom border
GFX_GOL_OBJ_SCHEME * pGolScheme;	the style scheme used
DRAW_FUNC draw;	custom control draw function
FREE_FUNC FreeObj;	function pointer to the object free function
ACTIONGET_FUNC actionGet;	function pointer to the object message function
ACTIONSET_FUNC actionSet;	function pointer to the object default message function
uint16_t pos;	current position
uint16_t prevPos;	previous position

Description

GFX GOL custom control.

Typedef: GFX_GOL_CUSTOMCONTROL

Defines the structure used for the Custom Control object. Object is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_DIGITALMETER Structure

Defines the structure used for the Digital Meter object.

File

gfx_gol_digital_meter.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
```

```
  uint16_t textHeight;
  uint32_t Cvalue;
  uint32_t Pvalue;
  uint8_t NoOfDigits;
  uint8_t DotPos;
  GFX_ALIGNMENT alignment;
} GFX_GOL_DIGITALMETER;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_OBJ_HEADER).
uint16_t textHeight;	Precomputed text height
uint32_t Cvalue;	Current value
uint32_t Pvalue;	Previous value
uint8_t NoOfDigits;	Number of digits to be displayed
uint8_t DotPos;	Position of decimal point
GFX_ALIGNMENT alignment;	text alignment

Description

GFX GOL digital meter.

Typedef: GFX_GOL_DIGITALMETER

Defines the parameters required for a Digital Meter Object. Depending on the type selected the meter is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_DIGITALMETER_STATE Enumeration

Specifies the different states of the Digital Meter object.

File

gfx_gol_digital_meter.h

C

```
typedef enum {
  GFX_GOL_DIGITALMETER_DISABLED_STATE,
  GFX_GOL_DIGITALMETER_FRAME_STATE,
  GFX_GOL_DIGITALMETER_DRAW_STATE,
  GFX_GOL_DIGITALMETER_UPDATE_STATE,
  GFX_GOL_DIGITALMETER_HIDE_STATE
} GFX_GOL_DIGITALMETER_STATE;
```

Members

Members	Description
GFX_GOL_DIGITALMETER_DISABLED_STATE	Property bit for disabled state.
GFX_GOL_DIGITALMETER_FRAME_STATE	Property bit to indicate frame is displayed.
GFX_GOL_DIGITALMETER_DRAW_STATE	Draw bit to indicate object must be redrawn.
GFX_GOL_DIGITALMETER_UPDATE_STATE	Draw bit to indicate that only text must be redrawn.
GFX_GOL_DIGITALMETER_HIDE_STATE	Draw bit to indicate the object must be removed from the screen.

Description

GFX GOL digital meter state.

Typedef: GFX_GOL_DIGITALMETER_STATE

This enumeration specifies the different states of the Digital Meter object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_DRAW_CALLBACK_FUNC Type

Draw callback function definition. This application defined function allows the application to perform application specific rendering.

File

gfx_gol.h

C

```
typedef bool (* GFX_GOL_DRAW_CALLBACK_FUNC)(void);
```

Returns

true - is returned when application rendering is done. false - is returned when application rendering is not yet finished.

Description

GFX GOL draw callback function pointer

This callback function is implemented by the application. This is called inside the GFX_GOL_ObjectListDraw() function when the drawing of objects in the active list is completed.

Any application specific rendering must be performed on this callback function so no object rendering will be affected by the application calls to primitive rendering functions. Application setting the drawing color, line style, fill style, text string cursor position and current font will not affect the object rendering. This is also the safe place to modify the active list.

When the application has performed its own primitive rendering calls, this function must return true to inform the GFX_GOL_ObjectListDraw() that it is done rendering and checking for object drawing or redrawing can continue.

Preconditions

None.

Example

None.

Function

typedef bool (*GFX_GOL_DRAW_CALLBACK_FUNC) (void);

GFX_GOL_EDITBOX Structure

Defines the structure used for the Edit Box object.

File

gfx_gol_edit_box.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  uint16_t textHeight;
  GFX_XCHAR * pText;
  GFX_ALIGNMENT alignment;
  uint16_t charMax;
  uint16_t length;
  uint16_t cursorPosition;
} GFX_GOL_EDITBOX;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
uint16_t textHeight;	Precomputed text height.
GFX_XCHAR * pText;	Pointer to text buffer.
GFX_ALIGNMENT alignment;	text alignment
uint16_t charMax;	Maximum number of characters in the edit box.
uint16_t length;	Current text length.
uint16_t cursorPosition;	cursor position

Description

GFX GOL edit box.

Typedef: GFX_GOL_EDITBOX

Defines the parameters required for a Edit Box Object. Object is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_GROUPBOX Structure

Defines the structure used for the Group Box object.

File

gfx_gol_group_box.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  uint16_t textWidth;
  uint16_t textHeight;
  GFX_XCHAR * pText;
  GFX_ALIGNMENT alignment;
} GFX_GOL_GROUPBOX;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_OBJ_HEADER).
uint16_t textWidth;	Precomputed text width.
uint16_t textHeight;	Precomputed text height.
GFX_XCHAR * pText;	Text string used.
GFX_ALIGNMENT alignment;	Text alignment

Description

GFX GOL group box struct.

Typedef: GFX_GOL_GROUPBOX

Defines the parameters required for a Group Box Object. Object is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_GROUPBOX_STATE Enumeration

Specifies the different states of the Group Box object.

File

gfx_gol_group_box.h

C

```
typedef enum {
  GFX_GOL_GROUPBOX_DISABLED_STATE,
  GFX_GOL_GROUPBOX_DRAW_STATE,
  GFX_GOL_GROUPBOX_HIDE_STATE
} GFX_GOL_GROUPBOX_STATE;
```

Members

Members	Description
GFX_GOL_GROUPBOX_DISABLED_STATE	Property bit to indicate object is disabled.
GFX_GOL_GROUPBOX_DRAW_STATE	Draw bit to indicate group box must be redrawn.
GFX_GOL_GROUPBOX_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

GFX GOL group box state.

Typedef: GFX_GOL_GROUPBOX_STATE

This enumeration specifies the different states of the Group Box object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_LISTBOX Structure

Defines the structure used for the List Box object.

File

gfx_gol_list_box.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  GFX_GOL_LISTITEM * pItemList;
  GFX_GOL_LISTITEM * pFocusItem;
  uint16_t itemsNumber;
  int16_t scrollY;
  int16_t textHeight;
  GFX_ALIGNMENT alignment;
} GFX_GOL_LISTBOX;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
GFX_GOL_LISTITEM * pItemList;	Pointer to the list of items.
GFX_GOL_LISTITEM * pFocusItem;	Pointer to the focused item.
uint16_t itemsNumber;	Number of items in the list box.
int16_t scrollY;	Scroll displacement for the list.
int16_t textHeight;	Precomputed text height.
GFX_ALIGNMENT alignment;	items alignment.

Description

GFX GOL list box structure.

Typedef: GFX_GOL_LISTBOX

Defines the parameters required for a List Box Object. Object is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_LISTBOX_ITEM_STATUS Enumeration

Defines the types used to indicate the status of an item.

File

gfx_gol_list_box.h

C

```
typedef enum {
  GFX_GOL_LISTBOX_ITEM_STATUS_SELECTED,
  GFX_GOL_LISTBOX_ITEM_STATUS_REDRAW
} GFX_GOL_LISTBOX_ITEM_STATUS;
```

Members

Members	Description
GFX_GOL_LISTBOX_ITEM_STATUS_SELECTED	Item is selected.
GFX_GOL_LISTBOX_ITEM_STATUS_REDRAW	Item is to be redrawn.

Description

GFX GOL list box item status.

Typedef: GFX_GOL_LISTBOX_ITEM_STATUS

Defines the types used to indicate the status of an item in the List Box.

Remarks

None.

GFX_GOL_LISTBOX_STATE Enumeration

Specifies the different states of the List Box object.

File

gfx_gol_list_box.h

C

```
typedef enum {
  GFX_GOL_LISTBOX_FOCUSED_STATE,
  GFX_GOL_LISTBOX_DISABLED_STATE,
  GFX_GOL_LISTBOX_SINGLE_SELECT_STATE,
  GFX_GOL_LISTBOX_DRAW_ITEMS_STATE,
  GFX_GOL_LISTBOX_DRAW_FOCUS_STATE,
  GFX_GOL_LISTBOX_DRAW_STATE,
  GFX_GOL_LISTBOX_HIDE_STATE
} GFX_GOL_LISTBOX_STATE;
```

Members

Members	Description
GFX_GOL_LISTBOX_FOCUSED_STATE	Property bit for focus state.
GFX_GOL_LISTBOX_DISABLED_STATE	Property bit to indicate object is disabled.
GFX_GOL_LISTBOX_SINGLE_SELECT_STATE	Property bit to indicate only one item can be selected.
GFX_GOL_LISTBOX_DRAW_ITEMS_STATE	Draw bit to indicate selected items of the object must be redrawn.
GFX_GOL_LISTBOX_DRAW_FOCUS_STATE	Draw bit to indicate focus must be redrawn.
GFX_GOL_LISTBOX_DRAW_STATE	Draw Bit to indicate object must be redrawn.
GFX_GOL_LISTBOX_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

GFX GOL list box state.

Typedef: GFX_GOL_LISTBOX_STATE

This enumeration specifies the different states of the List Box object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_LISTITEM Structure

The structure that defines each item in the list box.

File

gfx_gol_list_box.h

C

```
typedef struct {
  void * pPrevItem;
  void * pNextItem;
  GFX_GOL_LISTBOX_ITEM_STATUS status;
  GFX_XCHAR * pText;
  GFX_RESOURCE_HDR * pImage;
  uint16_t data;
} GFX_GOL_LISTITEM;
```

Members

Members	Description
void * pPrevItem;	Pointer to the next item
void * pNextItem;	Pointer to the next item
GFX_GOL_LISTBOX_ITEM_STATUS status;	Specifies the status of the item.
GFX_XCHAR * pText;	Pointer to the text for the item
GFX_RESOURCE_HDR * pImage;	Pointer to the image used
uint16_t data;	Some data associated with the item

Description

GFX GOL list item structure.

Typedef: GFX_GOL_LISTITEM

Each item in the list box is described by this structure. The items are arranged as a linked list of this type.

Remarks

None.

GFX_GOL_MESSAGE Structure

Specifies message structure used in the library.

File

gfx_gol.h

C

```
typedef struct {
  uint8_t type;
  uint8_t uiEvent;
  int16_t param1;
  int16_t param2;
} GFX_GOL_MESSAGE;
```

Members

Members	Description
uint8_t type;	Specifies the type of input device.
uint8_t uiEvent;	An event that occurred in the input device.
int16_t param1;	Parameter 1, definition and usage is dependent on the type of input device.
int16_t param2;	Parameter 2, definition and usage is dependent on the type of input device.

Description

GFX GOL Message

Typedef: GFX_GOL_MESSAGE

Specifies message structure used in the library.

- The types must be one of the INPUT_DEVICE_TYPE:
- TYPE_UNKNOWN
- TYPE_KEYBOARD
- TYPE_TOUCHSCREEN
- TYPE_MOUSE
- uiEvent must be one of the INPUT_DEVICE_EVENT.
- for touch screen:

- EVENT_INVALID
- EVENT_MOVE
- EVENT_PRESS
- EVENT_STILLPRESS
- EVENT_RELEASE
- for keyboard:
- EVENT_KEYSCAN (param2 contains scan code)
- EVENT_KEYCODE (param2 contains character code)
- param1:
- for touch screen is the x position
- for keyboard ID of object receiving the message
- param2
- for touch screen y position
- for keyboard scan or key code

Remarks

None.

GFX_GOL_METER Structure

Defines the structure used for the Meter object.

File

gfx_gol_meter.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  GFX_XCHAR * pText;
  GFX_GOL_METER_DRAW_TYPE type;
  int16_t value;
  int16_t minValue;
  int16_t maxValue;
  int16_t xCenter;
  int16_t yCenter;
  int16_t radius;
  int16_t xPos;
  int16_t yPos;
  GFX_COLOR color1;
  GFX_COLOR color2;
  GFX_COLOR color3;
  GFX_COLOR color4;
  GFX_COLOR color5;
  GFX_COLOR color6;
  GFX_RESOURCE_HDR * pTitleFont;
  GFX_RESOURCE_HDR * pValueFont;
} GFX_GOL_METER;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER).
GFX_XCHAR * pText;	The text label of the meter.
GFX_GOL_METER_DRAW_TYPE type;	sets the type of the meter
int16_t value;	Current value of the meter.
int16_t minValue;	minimum value the meter can display
int16_t maxValue;	maximum value the meter can display (range is maxValue - minValue)
int16_t xCenter;	The x coordinate center position. This is computed automatically.
int16_t yCenter;	The y coordinate center position. This is computed automatically.
int16_t radius;	Radius of the meter, also defines the needle length.
int16_t xPos;	The current x position of the needle. This is computed automatically.
int16_t yPos;	The current y position of the needle. This is computed automatically.
GFX_COLOR color1;	Arc1 and scale1 color parameter.
GFX_COLOR color2;	Arc2 and scale2 color parameter

GFX_COLOR color3;	Arc3 and scale3 color parameter
GFX_COLOR color4;	Arc4 and scale4 color parameter
GFX_COLOR color5;	Arc5 and scale5 color parameter
GFX_COLOR color6;	Arc6 and scale6 color parameter
GFX_RESOURCE_HDR * pTitleFont;	Pointer to the font used in the title of the meter
GFX_RESOURCE_HDR * pValueFont;	Pointer to the font used in the current reading (if displayed) of the meter

Description

GFX GOL meter structure.

Typedef: GFX_GOL_METER

Defines the parameters required for a Meter Object. Depending on the type selected the meter is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_METER_DRAW_TYPE Enumeration

Specifies the different types of Meter object.

File

gfx_gol_meter.h

C

```
typedef enum {
  GFX_GOL_METER_WHOLE_TYPE,
  GFX_GOL_METER_HALF_TYPE,
  GFX_GOL_METER_QUARTER_TYPE
} GFX_GOL_METER_DRAW_TYPE;
```

Members

Members	Description
GFX_GOL_METER_WHOLE_TYPE	draw circular meter
GFX_GOL_METER_HALF_TYPE	draw semi circle meter
GFX_GOL_METER_QUARTER_TYPE	draw quarter of a circle meter

Description

GFX GOL meter draw type.

Typedef: GFX_GOL_METER_DRAW_TYPE

This enumeration specifies the different typer of the Meter object used in the library.

- GFX_GOL_METER_WHOLE_TYPE - The meter will be drawn using all the arc colors (color1 to color6).
- GFX_GOL_METER_HALF_TYPE - The meter will be drawn using arc colors (color5, color4, color3, color2).
- GFX_GOL_METER_QUARTER_TYPE - The meter will be drawn using arc colors (color3, color2).

Remarks

None.

GFX_GOL_METER_STATE Enumeration

Specifies the different states of the Meter object.

File

gfx_gol_meter.h

C

```
typedef enum {
  GFX_GOL_METER_DISABLED_STATE,
  GFX_GOL_METER_RING_STATE,
  GFX_GOL_METER_ACCURACY_STATE,
  GFX_GOL_METER_UPDATE_DRAW_STATE,
  GFX_GOL_METER_DRAW_STATE,
  GFX_GOL_METER_HIDE_STATE
```

```
} GFX_GOL_METER_STATE;
```

Members

Members	Description
GFX_GOL_METER_DISABLED_STATE	Property bit for disabled state.
GFX_GOL_METER_RING_STATE	Property bit for ring type, scales are drawn over the ring. The default state of this state bit is disabled. Only scales are drawn.
GFX_GOL_METER_ACCURACY_STATE	Property bit when set, the values will have accuracy for 1 decimal place.
GFX_GOL_METER_UPDATE_DRAW_STATE	Draw bit to indicate update of the meter-hand only.
GFX_GOL_METER_DRAW_STATE	Draw bit to indicate the whole object must be drawn.
GFX_GOL_METER_HIDE_STATE	Draw bit to indicate the object must be removed from the screen.

Description

GFX GOL meter state.

Typedef: GFX_GOL_METER_STATE

This enumeration specifies the different states of the Meter object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_OBJ_HEADER Structure

Specifies Graphics Object Layer structure used in objects.

File

gfx_gol.h

C

```
typedef struct {
  SYS_MODULE_INDEX gfxIndex;
  uint16_t ID;
  void * pNxtObj;
  GFX_GOL_OBJ_TYPE type;
  uint16_t state;
  uint16_t left;
  uint16_t top;
  uint16_t right;
  uint16_t bottom;
  GFX_GOL_OBJ_SCHEME * pGolScheme;
  DRAW_FUNC DrawObj;
  FREE_FUNC FreeObj;
  ACTIONGET_FUNC actionGet;
  ACTIONSET_FUNC actionSet;
  uint16_t groupNo;
} GFX_GOL_OBJ_HEADER;
```

Members

Members	Description
SYS_MODULE_INDEX gfxIndex;	GFX instance index
uint16_t ID;	Unique id assigned for referencing.
void * pNxtObj;	A pointer to the next object.
GFX_GOL_OBJ_TYPE type;	Identifies the type of GOL object.
uint16_t state;	State of object.
uint16_t left;	Left position of the Object.
uint16_t top;	Top position of the Object.
uint16_t right;	Right position of the Object.
uint16_t bottom;	Bottom position of the Object.
GFX_GOL_OBJ_SCHEME * pGolScheme;	Pointer to the scheme used.

DRAW_FUNC DrawObj;	function pointer to the object's draw function.
FREE_FUNC FreeObj;	function pointer to the object's free function.
ACTIONGET_FUNC actionGet;	function pointer to the object's action get function.
ACTIONSET_FUNC actionSet;	function pointer to the object's action set function.
uint16_t groupNo;	GroupNo to keep track of group for radiobutton.

Description

GFX GOL object header

Typedef: GFX_GOL_OBJ_HEADER

This structure defines the Graphics Object Layer header used in all objects in the Graphics Library.

Remarks

None.

GFX_GOL_OBJ_TYPE Enumeration

Specifies the different object types used in the library.

File

gfx_gol.h

C

```
typedef enum {
  GFX_GOL_ANALOGCLOCK_TYPE,
  GFX_GOL_BUTTON_TYPE,
  GFX_GOL_CHART_TYPE,
  GFX_GOL_CHECKBOX_TYPE,
  GFX_GOL_DIGITALMETER_TYPE,
  GFX_GOL_EDITBOX_TYPE,
  GFX_GOL_GRID_TYPE,
  GFX_GOL_GROUPBOX_TYPE,
  GFX_GOL_LISTBOX_TYPE,
  GFX_GOL_METER_TYPE,
  GFX_GOL_PICTURECONTROL_TYPE,
  GFX_GOL_PROGRESSBAR_TYPE,
  GFX_GOL_RADIOBUTTON_TYPE,
  GFX_GOL_SCROLLBAR_TYPE,
  GFX_GOL_STATICTEXT_TYPE,
  GFX_GOL_TEXTENTRY_TYPE,
  GFX_GOL_WINDOW_TYPE,
  GFX_GOL_CUSTOM_TYPE,
  GFX_GOL_SURFACE_TYPE,
  GFX_GOL_UNKNOWN_TYPE
} GFX_GOL_OBJ_TYPE;
```

Members

Members	Description
GFX_GOL_ANALOGCLOCK_TYPE	Type defined for Analog Clock Object.
GFX_GOL_BUTTON_TYPE	Type defined for Button Object.
GFX_GOL_CHART_TYPE	Type defined for Chart Object.
GFX_GOL_CHECKBOX_TYPE	Type defined for Check Box Object.
GFX_GOL_DIGITALMETER_TYPE	Type defined for Digital Meter Object.
GFX_GOL_EDITBOX_TYPE	Type defined for Edit Box Object.
GFX_GOL_GRID_TYPE	Type defined for Grid Object.
GFX_GOL_GROUPBOX_TYPE	Type defined for Group Box Object.
GFX_GOL_LISTBOX_TYPE	Type defined for List Box Object.
GFX_GOL_METER_TYPE	Type defined for Meter Object.
GFX_GOL_PICTURECONTROL_TYPE	Type defined for Picture Control Object.
GFX_GOL_PROGRESSBAR_TYPE	Type defined for Progress Bar Object.
GFX_GOL_RADIOBUTTON_TYPE	Type defined for Radio Button Object.
GFX_GOL_SCROLLBAR_TYPE	Type defined for Slider or Scroll Bar Object.
GFX_GOL_STATICTEXT_TYPE	Type defined for Static Text Object.

GFX_GOL_TEXTENTRY_TYPE	Type defined for Text-Entry Object.
GFX_GOL_WINDOW_TYPE	Type defined for Window Object.
GFX_GOL_CUSTOM_TYPE	Type defined for Custom Object.
GFX_GOL_UNKNOWN_TYPE	Type is undefined and not supported by the library.

Description

GFX GOL object type

Typedef: GFX_GOL_OBJ_TYPE

This enumeration specifies the different object types used in the library.

Remarks

None.

GFX_GOL_PROGRESSBAR Structure

Defines the structure used for the Progress Bar object.

File

gfx_gol_progress_bar.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  uint16_t pos;
  uint16_t prevPos;
  uint16_t range;
} GFX_GOL_PROGRESSBAR;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
uint16_t pos;	Current progress position.
uint16_t prevPos;	Previous progress position.
uint16_t range;	Sets the range of the object.

Description

GFX GOL progress bar structure.

Typedef: GFX_GOL_PROGRESSBAR

Defines the parameters required for a Progress Bar object. Object is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_PROGRESSBAR_STATE Enumeration

Specifies the different states of the Progress Bar object.

File

gfx_gol_progress_bar.h

C

```
typedef enum {
  GFX_GOL_PROGRESSBAR_DISABLED_STATE,
  GFX_GOL_PROGRESSBAR_VERTICAL_STATE,
  GFX_GOL_PROGRESSBAR_NOPROGRESS_STATE,
  GFX_GOL_PROGRESSBAR_DRAW_BAR_STATE,
  GFX_GOL_PROGRESSBAR_DRAW_STATE,
  GFX_GOL_PROGRESSBAR_HIDE_STATE
} GFX_GOL_PROGRESSBAR_STATE;
```

Members

Members	Description
GFX_GOL_PROGRESSBAR_DISABLED_STATE	Property bit to indicate object is disabled.
GFX_GOL_PROGRESSBAR_VERTICAL_STATE	Property bit for vertical orientation. • When this state bit is 0 - object is rendered with horizontal orientation. • When this state bit is 1 - object is rendered with vertical orientation.
GFX_GOL_PROGRESSBAR_NOPROGRESS_STATE	Property bit that will suppress rendering of progress in text.
GFX_GOL_PROGRESSBAR_DRAW_BAR_STATE	Draw bit to indicate that the progress bar portion must be redrawn.
GFX_GOL_PROGRESSBAR_DRAW_STATE	Draw bit to indicate object must be redrawn.
GFX_GOL_PROGRESSBAR_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

GFX GOL progress bar state.

Typedef: GFX_GOL_PROGRESSBAR_STATE

This enumeration specifies the different states of the Progress Bar object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_RADIOBUTTON Structure

Defines the structure used for the Radio Button object.

File

gfx_gol_radio_button.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  GFX_GOL_OBJ_HEADER * pHead;
  GFX_GOL_OBJ_HEADER * pNext;
  uint16_t textHeight;
  GFX_XCHAR * pText;
  GFX_ALIGNMENT alignment;
} GFX_GOL_RADIOBUTTON;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER).
GFX_GOL_OBJ_HEADER * pHead;	Pointer to the first Radio Button in the group
GFX_GOL_OBJ_HEADER * pNext;	Pointer to the next Radio Button in the group
uint16_t textHeight;	Precomputed text height
GFX_XCHAR * pText;	Pointer to the text
GFX_ALIGNMENT alignment;	text alignment

Description

GFX GOL Radio button structure.

Typedef: GFX_GOL_RADIOBUTTON

Defines the parameters required for a Radio Button object. Object is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_RADIOBUTTON_STATE Enumeration

Specifies the different states of the Radio Button object.

File

gfx_gol_radio_button.h

C

```
typedef enum {
  GFX_GOL_RADIOBUTTON_FOCUSED_STATE,
  GFX_GOL_RADIOBUTTON_DISABLED_STATE,
  GFX_GOL_RADIOBUTTON_CHECKED_STATE,
  GFX_GOL_RADIOBUTTON_GROUP_STATE,
  GFX_GOL_RADIOBUTTON_DRAW_CHECK_STATE,
  GFX_GOL_RADIOBUTTON_DRAW_FOCUS_STATE,
  GFX_GOL_RADIOBUTTON_DRAW_STATE,
  GFX_GOL_RADIOBUTTON_HIDE_STATE
} GFX_GOL_RADIOBUTTON_STATE;
```

Members

Members	Description
GFX_GOL_RADIOBUTTON_FOCUSED_STATE	Property bit for focus state.
GFX_GOL_RADIOBUTTON_DISABLED_STATE	Property bit for disabled state.
GFX_GOL_RADIOBUTTON_CHECKED_STATE	Property bit to indicate Radio Button is checked.
GFX_GOL_RADIOBUTTON_GROUP_STATE	Bit to indicate the first Radio Button in the group. Each group MUST have this bit set for its first member even for a single member group. This means that any independent or stand alone Radio Button, the GFX_GOL_RADIOBUTTON_GROUP_STATE bit must be always set.
GFX_GOL_RADIOBUTTON_DRAW_CHECK_STATE	Draw bit to indicate check mark should be redrawn.
GFX_GOL_RADIOBUTTON_DRAW_FOCUS_STATE	Draw bit to indicate focus must be redrawn.
GFX_GOL_RADIOBUTTON_DRAW_STATE	Draw bit to indicate object must be redrawn.
GFX_GOL_RADIOBUTTON_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

GFX GOL radio button state.

Typedef: GFX_GOL_RADIOBUTTON_STATE

This enumeration specifies the different states of the Radio Button object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_SCROLLBAR Structure

Defines the structure used for the Scroll Bar object.

File

gfx_gol_scroll_bar.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  int16_t currPos;
  uint16_t prevPos;
  uint16_t range;
  int16_t pos;
  uint16_t page;
  uint16_t thWidth;
  uint16_t thHeight;
} GFX_GOL_SCROLLBAR;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
int16_t currPos;	Position of the slider relative to minimum.
uint16_t prevPos;	Previous position of the slider relative to minimum.
uint16_t range;	User defined range of the slider. Minimum value at 0 and maximum at 0x7FFF.
int16_t pos;	Position of the slider in range domain.
uint16_t page;	User specified resolution to incrementally change the position in range domain.
uint16_t thWidth;	Thumb width. This is computed internally. User must not change this value.
uint16_t thHeight;	Thumb width. This is computed internally. User must not change this value.

Description

GFX GOL scroll bar struct.

Typedef: GFX_GOL_SCROLLBAR

Defines the parameters required for a Scroll Bar object. Object is drawn with the defined shape parameters and values set on the given fields. Depending on the GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE state bit slider or scrollbar mode is set. If GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE is set, mode is slider; if not set mode is scroll bar. For scrollbar mode, focus rectangle is not drawn.

Remarks

None.

GFX_GOL_SCROLLBAR_STATE Enumeration

Specifies the different states of the Scroll Bar object.

File

gfx_gol_scroll_bar.h

C

```
typedef enum {
  GFX_GOL_SCROLLBAR_FOCUSED_STATE,
  GFX_GOL_SCROLLBAR_DISABLED_STATE,
  GFX_GOL_SCROLLBAR_VERTICAL_STATE,
  GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE,
  GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE,
  GFX_GOL_SCROLLBAR_DRAW_FOCUS_STATE,
  GFX_GOL_SCROLLBAR_DRAW_STATE,
  GFX_GOL_SCROLLBAR_HIDE_STATE
} GFX_GOL_SCROLLBAR_STATE;
```

Members

Members	Description
GFX_GOL_SCROLLBAR_FOCUSED_STATE	Property bit for focus state.
GFX_GOL_SCROLLBAR_DISABLED_STATE	Property bit for disabled state.
GFX_GOL_SCROLLBAR_VERTICAL_STATE	Property bit to indicate the scroll bar is drawn with vertical orientation
GFX_GOL_SCROLLBAR_SLIDER_MODE_STATE	Property bit to indicate the scroll bar is in slider mode.
GFX_GOL_SCROLLBAR_DRAW_THUMB_STATE	Draw bit to indicate that only the thumb area will be redrawn.
GFX_GOL_SCROLLBAR_DRAW_FOCUS_STATE	Draw bit to indicate focus must be redrawn.
GFX_GOL_SCROLLBAR_DRAW_STATE	Draw bit to indicate object must be redrawn.
GFX_GOL_SCROLLBAR_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

GFX GOL scroll bar state.

Typedef: GFX_GOL_SCROLLBAR_STATE

This enumeration specifies the different states of the Scroll Bar object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_STATICTEXT Structure

Defines the structure used for the Static Text object.

File

gfx_gol_static_text.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  GFX_XCHAR * pText;
  GFX_ALIGNMENT alignment;
} GFX_GOL_STATICTEXT;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see OBJ_HEADER).
GFX_XCHAR * pText;	The pointer to text used.
GFX_ALIGNMENT alignment;	text alignment

Description

GFX GOL static text structure.

Typedef: GFX_GOL_STATICTEXT

Defines the parameters required for a Static Text object. Object is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_STATICTEXT_STATE Enumeration

Specifies the different states of the Static Text object.

File

gfx_gol_static_text.h

C

```
typedef enum {
  GFX_GOL_STATICTEXT_DISABLED_STATE,
  GFX_GOL_STATICTEXT_FRAME_STATE,
  GFX_GOL_STATICTEXT_NOBACKGROUND_STATE,
  GFX_GOL_STATICTEXT_DRAW_STATE,
  GFX_GOL_STATICTEXT_HIDE_STATE
} GFX_GOL_STATICTEXT_STATE;
```

Members

Members	Description
GFX_GOL_STATICTEXT_DISABLED_STATE	Property bit for disabled state.
GFX_GOL_STATICTEXT_FRAME_STATE	Property bit to indicate frame is enabled.
GFX_GOL_STATICTEXT_NOBACKGROUND_STATE	Property bit to indicate background is enabled.
GFX_GOL_STATICTEXT_DRAW_STATE	Draw bit to indicate object must be redrawn.
GFX_GOL_STATICTEXT_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

GFX GOL static text state.

Typedef: GFX_GOL_STATICTEXT_STATE

This enumeration specifies the different states of the Static Text object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a

complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_TEXTENTRY Structure

Defines the structure used for the Text Entry object.

File

gfx_gol_text_entry.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  uint16_t horizontalKeys;
  uint16_t verticalKeys;
  GFX_XCHAR * pTeOutput;
  GFX_ALIGNMENT alignment;
  uint16_t CurrentLength;
  uint16_t outputLenMax;
  GFX_GOL_TEXTENTRY_KEYMEMBER * pActiveKey;
  GFX_GOL_TEXTENTRY_KEYMEMBER * pHeadOfList;
  GFX_RESOURCE_HDR * pDisplayFont;
} GFX_GOL_TEXTENTRY;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all objects (see GFX_GOL_OBJ_HEADER).
uint16_t horizontalKeys;	Number of horizontal keys.
uint16_t verticalKeys;	Number of vertical keys.
GFX_XCHAR * pTeOutput;	Pointer to the buffer assigned by the user which holds the text shown in the edit box.
GFX_ALIGNMENT alignment;	Defines the text alignment.
uint16_t CurrentLength;	Current length of the string in the buffer. The maximum value of this is equal to outputLenMax. User creates and manages the buffer. Buffer can also be managed using the APIs provided to add a character, delete the last character or clear the buffer.
uint16_t outputLenMax;	Maximum expected length of output buffer. Object will update this parameter when adding, removing characters or clearing the buffer and switching buffers.
GFX_GOL_TEXTENTRY_KEYMEMBER * pActiveKey;	Pointer to the active key. This is only used by the object. User must not change the value of this parameter directly.
GFX_GOL_TEXTENTRY_KEYMEMBER * pHeadOfList;	Pointer to head of the list
GFX_RESOURCE_HDR * pDisplayFont;	Pointer to the font used in displaying the text.

Description

GFX GOL text entry structure.

Typedef: GFX_GOL_TEXTENTRY

Defines the parameters required for a Text Entry object. Object is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE Enumeration

Specifies the different commands available for command keys of the Text Entry object.

File

gfx_gol_text_entry.h

C

```
typedef enum {
  GFX_GOL_TEXTENTRY_NONE_COM,
```

```
  GFX_GOL_TEXTENTRY_DELETE_COM,
  GFX_GOL_TEXTENTRY_SPACE_COM,
  GFX_GOL_TEXTENTRY_ENTER_COM
} GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE;
```

Members

Members	Description
GFX_GOL_TEXTENTRY_NONE_COM	This type is used when the key is not assigned to any command.
GFX_GOL_TEXTENTRY_DELETE_COM	This type is used to assign a "delete" command on a key.
GFX_GOL_TEXTENTRY_SPACE_COM	This type is used to assign an insert "space" command on a key.
GFX_GOL_TEXTENTRY_ENTER_COM	This type is used to assign an "enter" carriage return) command on a key.

Description

GFX GOL text entry key command type.

Typedef: GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE

This enumeration specifies the different commands available for the command keys of the object.

The GFX_GOL_TEXTENTRY_ENTER_COM command can be used to customize application code in the message callback function. Use the returned translated GFX_GOL_TEXTENTRY_ACTION_ENTER to detect the key pressed was assigned the enter command. Refer to GFX_GOL_TextEntryActionGet() for details.

Remarks

None.

GFX_GOL_TEXTENTRY_STATE Enumeration

Specifies the different states of the Text Entry object.

File

gfx_gol_text_entry.h

C

```
typedef enum {
  GFX_GOL_TEXTENTRY_DISABLED_STATE,
  GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE,
  GFX_GOL_TEXTENTRY_ECHO_HIDE_STATE,
  GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE,
  GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE,
  GFX_GOL_TEXTENTRY_DRAW_STATE,
  GFX_GOL_TEXTENTRY_HIDE_STATE
} GFX_GOL_TEXTENTRY_STATE;
```

Members

Members	Description
GFX_GOL_TEXTENTRY_DISABLED_STATE	Property bit for disabled state.
GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE	Property bit for press state.
GFX_GOL_TEXTENTRY_ECHO_HIDE_STATE	Bit to hide the entered characters and instead echo "*" characters to the display.
GFX_GOL_TEXTENTRY_UPDATE_TEXT_STATE	Draw bit to indicate redraw of the text displayed is needed.
GFX_GOL_TEXTENTRY_UPDATE_KEY_STATE	Draw bit to indicate redraw of a key is needed.
GFX_GOL_TEXTENTRY_DRAW_STATE	Draw bit to indicate object must be redrawn.
GFX_GOL_TEXTENTRY_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

GFX GOL text entry state.

Typedef: GFX_GOL_TEXTENTRY_STATE

This enumeration specifies the different states of the Text Entry object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_TRANSLATED_ACTION Enumeration

Specifies the different object actions supported in the library.

File

gfx_gol.h

C

```
typedef enum {
  GFX_GOL_OBJECT_ACTION_INVALID = 0x3500,
  GFX_GOL_OBJECT_ACTION_PASSIVE,
  GFX_GOL_ANALOGCLOCK_ACTION_PRESSED,
  GFX_GOL_ANALOGCLOCK_ACTION_RELEASED,
  GFX_GOL_BUTTON_ACTION_PRESSED,
  GFX_GOL_BUTTON_ACTION_STILLPRESSED,
  GFX_GOL_BUTTON_ACTION_RELEASED,
  GFX_GOL_BUTTON_ACTION_CANCELPRESS,
  GFX_GOL_CHART_ACTION_SELECTED,
  GFX_GOL_CHECKBOX_ACTION_CHECKED,
  GFX_GOL_CHECKBOX_ACTION_UNCHECKED,
  GFX_GOL_CUSTOMCONTROL_ACTION_SELECTED,
  GFX_GOL_DIGITALMETER_ACTION_SELECTED,
  GFX_GOL_EDITBOX_ACTION_ADD_CHAR,
  GFX_GOL_EDITBOX_ACTION_DEL_CHAR,
  GFX_GOL_EDITBOX_ACTION_TOUCHSCREEN,
  GFX_GOL_GRID_ACTION_TOUCHED,
  GFX_GOL_GRID_ACTION_ITEM_SELECTED,
  GFX_GOL_GRID_ACTION_UP,
  GFX_GOL_GRID_ACTION_DOWN,
  GFX_GOL_GRID_ACTION_LEFT,
  GFX_GOL_GRID_ACTION_RIGHT,
  GFX_GOL_GROUPBOX_ACTION_SELECTED,
  GFX_GOL_LISTBOX_ACTION_SELECTED,
  GFX_GOL_LISTBOX_ACTION_MOVE,
  GFX_GOL_LISTBOX_ACTION_TOUCHSCREEN,
  GFX_GOL_METER_ACTION_SET,
  GFX_GOL_PICTURECONTROL_ACTION_SELECTED,
  GFX_GOL_PROGRESSBAR_ACTION_SELECTED,
  GFX_GOL_RADIOBUTTON_ACTION_CHECKED,
  GFX_GOL_SCROLLBAR_ACTION_INC,
  GFX_GOL_SCROLLBAR_ACTION_DEC,
  GFX_GOL_STATICTEXT_ACTION_SELECTED,
  GFX_GOL_TEXTENTRY_ACTION_RELEASED,
  GFX_GOL_TEXTENTRY_ACTION_PRESSED,
  GFX_GOL_TEXTENTRY_ACTION_ADD_CHAR,
  GFX_GOL_TEXTENTRY_ACTION_DELETE,
  GFX_GOL_TEXTENTRY_ACTION_SPACE,
  GFX_GOL_TEXTENTRY_ACTION_ENTER,
  GFX_GOL_WINDOW_ACTION_CLIENT,
  GFX_GOL_WINDOW_ACTION_TITLE
} GFX_GOL_TRANSLATED_ACTION;
```

Members

Members	Description
GFX_GOL_OBJECT_ACTION_INVALID = 0x3500	Invalid message response.
GFX_GOL_OBJECT_ACTION_PASSIVE	Passive message response. No change in object needed.
GFX_GOL_ANALOGCLOCK_ACTION_PRESSED	Analog Clock Pressed Action
GFX_GOL_ANALOGCLOCK_ACTION_RELEASED	Analog Clock Released Action
GFX_GOL_BUTTON_ACTION_PRESSED	Button pressed action ID.
GFX_GOL_BUTTON_ACTION_STILLPRESSED	Button is continuously pressed ID.
GFX_GOL_BUTTON_ACTION_RELEASED	Button released action ID.
GFX_GOL_BUTTON_ACTION_CANCELPRESS	Button released action ID with button press canceled.
GFX_GOL_CHART_ACTION_SELECTED	Chart selected action ID

GFX_GOL_CHECKBOX_ACTION_CHECKED	Check Box check action ID.
GFX_GOL_CHECKBOX_ACTION_UNCHECKED	Check Box uncheck action ID.
GFX_GOL_CUSTOMCONTROL_ACTION_SELECTED	Custom Control selected action ID.
GFX_GOL_DIGITALMETER_ACTION_SELECTED	Digital Meter selected action ID.
GFX_GOL_EDITBOX_ACTION_ADD_CHAR	Edit Box insert character action ID.
GFX_GOL_EDITBOX_ACTION_DEL_CHAR	Edit Box remove character action ID.
GFX_GOL_EDITBOX_ACTION_TOUCHSCREEN	Edit Box touchscreen selected action ID.
GFX_GOL_GRID_ACTION_TOUCHED	Grid item touched action ID.
GFX_GOL_GRID_ACTION_ITEM_SELECTED	Grid item selected action ID.
GFX_GOL_GRID_ACTION_UP	Grid up action ID.
GFX_GOL_GRID_ACTION_DOWN	Grid down action ID.
GFX_GOL_GRID_ACTION_LEFT	Grid left action ID.
GFX_GOL_GRID_ACTION_RIGHT	Grid right action ID.
GFX_GOL_GROUPBOX_ACTION_SELECTED	Group Box selected action ID.
GFX_GOL_LISTBOX_ACTION_SELECTED	List Box item select action ID.
GFX_GOL_LISTBOX_ACTION_MOVE	List Box item move action ID.
GFX_GOL_LISTBOX_ACTION_TOUCHSCREEN	List Box touchscreen selected action ID.
GFX_GOL_METER_ACTION_SET	Meter set value action ID.
GFX_GOL_PICTURECONTROL_ACTION_SELECTED	Picture selected action ID.
GFX_GOL_PROGRESSBAR_ACTION_SELECTED	Progress Bar selected action ID.
GFX_GOL_RADIOBUTTON_ACTION_CHECKED	Radio Button check action ID.
GFX_GOL_SCROLLBAR_ACTION_INC	Slider or Scroll Bar increment action ID.
GFX_GOL_SCROLLBAR_ACTION_DEC	Slider or Scroll Bar decrement action ID.
GFX_GOL_STATICTEXT_ACTION_SELECTED	Static Text selected action ID.
GFX_GOL_TEXTENTRY_ACTION_RELEASED	TextEntry released action ID
GFX_GOL_TEXTENTRY_ACTION_PRESSED	TextEntry pressed action ID
GFX_GOL_TEXTENTRY_ACTION_ADD_CHAR	TextEntry add character action ID
GFX_GOL_TEXTENTRY_ACTION_DELETE	TextEntry delete character action ID
GFX_GOL_TEXTENTRY_ACTION_SPACE	TextEntry add space character action ID
GFX_GOL_TEXTENTRY_ACTION_ENTER	TextEntry enter action ID
GFX_GOL_WINDOW_ACTION_CLIENT	Window client area selected action ID.
GFX_GOL_WINDOW_ACTION_TITLE	Window title bar selected action ID.

Description

GFX GOL Translated Action

Typedef: GFX_GOL_TRANSLATED_ACTION

This enumeration specifies the different object actions supported in the library.

Remarks

None.

GFX_GOL_WINDOW Structure

Defines the structure used for the Window object.

File

gfx_gol_window.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  uint16_t textHeight;
  GFX_XCHAR * pText;
  GFX_ALIGNMENT alignment;
  GFX_RESOURCE_HDR * pImage;
} GFX_GOL_WINDOW;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all objects (see GFX_GOL_OBJ_HEADER).
uint16_t textHeight;	Precomputed text height
GFX_XCHAR * pText;	Pointer to the title text
GFX_ALIGNMENT alignment;	text alignment
GFX_RESOURCE_HDR * pImage;	Pointer to the image for the title bar

Description

GFX GOL window structure.

Typedef: GFX_GOL_WINDOW

Defines the parameters required for a Window object. Object is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_WINDOW_STATE Enumeration

Specifies the different states of the Window object.

File

gfx_gol_window.h

C

```
typedef enum {
  GFX_GOL_WINDOW_FOCUSED_STATE,
  GFX_GOL_WINDOW_DISABLED_STATE,
  GFX_GOL_WINDOW_DRAW_TITLE_STATE,
  GFX_GOL_WINDOW_DRAW_CLIENT_STATE,
  GFX_GOL_WINDOW_DRAW_STATE,
  GFX_GOL_WINDOW_HIDE_STATE
} GFX_GOL_WINDOW_STATE;
```

Members

Members	Description
GFX_GOL_WINDOW_FOCUSED_STATE	Property bit for focus state.
GFX_GOL_WINDOW_DISABLED_STATE	Property bit for disabled state.
GFX_GOL_WINDOW_DRAW_TITLE_STATE	Draw bit to indicate title area must be redrawn.
GFX_GOL_WINDOW_DRAW_CLIENT_STATE	Draw bit to indicate client area must be redrawn.
GFX_GOL_WINDOW_DRAW_STATE	Draw bit to indicate object must be redrawn.
GFX_GOL_WINDOW_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

GFX GOL window state.

Typedef: GFX_GOL_WINDOW_STATE

This enumeration specifies the different states of the Window object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GOL_PANEL_PARAM Structure

Specifies panel parameters.

File

gfx_gol.h

C

```
typedef struct {
  int16_t panelLeft;
  int16_t panelTop;
  int16_t panelRight;
  int16_t panelBottom;
  GFX_COLOR panelFaceColor;
  GFX_COLOR panelEmbossLtColor;
  GFX_COLOR panelEmbossDkColor;
  uint16_t panelEmbossSize;
  uint16_t panelRadius;
  GFX_RESOURCE_HDR * pPanelImage;
  GFX_FILL_STYLE panelFillStyle;
  GFX_COLOR panelGradientStartColor;
  GFX_COLOR panelGradientEndColor;
  uint16_t panelAlpha;
} GOL_PANEL_PARAM;
```

Members

Members	Description
int16_t panelLeft;	Panel left coordinate.
int16_t panelTop;	Panel top coordinate.
int16_t panelRight;	Panel right coordinate.
int16_t panelBottom;	Panel Bottom coordinate.
GFX_COLOR panelFaceColor;	Panel Face color.
GFX_COLOR panelEmbossLtColor;	Panel Emboss light color.
GFX_COLOR panelEmbossDkColor;	Panel Emboss dark color.
uint16_t panelEmbossSize;	Panel Emboss size.
uint16_t panelRadius;	Panel radius.
GFX_RESOURCE_HDR * pPanelImage;	Panel image resource.
GFX_FILL_STYLE panelFillStyle;	Panel fill style.
GFX_COLOR panelGradientStartColor;	Panel gradient start color.
GFX_COLOR panelGradientEndColor;	Panel gradient end color .
uint16_t panelAlpha;	Panel alpha

Description

GOL panel parameter

Typedef: GOL_PANEL_PARAM

This structure defines the panel parameters when rendering a panel.

Remarks

None.

GFX_GOL_EDITBOX_STATE Enumeration

Specifies the different states of the Edit Box object.

File

gfx_gol_edit_box.h

C

```
typedef enum {
  GFX_GOL_EDITBOX_FOCUSED_STATE,
  GFX_GOL_EDITBOX_DISABLED_STATE,
  GFX_GOL_EDITBOX_ENABLE_CARET_STATE,
  GFX_GOL_EDITBOX_ENABLE_BAR_CARET_STATE,
  GFX_GOL_EDITBOX_ENABLE_BLOCK_CARET_STATE,
  GFX_GOL_EDITBOX_ENABLE_UNDERSCORE_CARET_STATE,
  GFX_GOL_EDITBOX_DRAW_CARET_STATE,
  GFX_GOL_EDITBOX_DRAW_FOCUS_STATE,
```

```
  GFX_GOL_EDITBOX_DRAW_STATE,
  GFX_GOL_EDITBOX_HIDE_STATE
} GFX_GOL_EDITBOX_STATE;
```

Members

Members	Description
GFX_GOL_EDITBOX_FOCUSED_STATE	Property bit for focus state. Cursor caret will be drawn when GFX_GOL_EDITBOX_ENABLE_CARET_STATE is also set.
GFX_GOL_EDITBOX_DISABLED_STATE	Property bit to indicate object is disabled.
GFX_GOL_EDITBOX_ENABLE_CARET_STATE	Property bit to indicate cursor caret will always be shown.
GFX_GOL_EDITBOX_ENABLE_BAR_CARET_STATE	property bit to indicate bar type caret
GFX_GOL_EDITBOX_ENABLE_BLOCK_CARET_STATE	property bit to indicate block type caret
GFX_GOL_EDITBOX_ENABLE_UNDERSCORE_CARET_STATE	property bit to indicate underscore type caret
GFX_GOL_EDITBOX_DRAW_CARET_STATE	Draw bit to indicate the cursor caret will be drawn if GFX_GOL_EDITBOX_FOCUSED_STATE state bit is set and erase when GFX_GOL_EDITBOX_FOCUSED_STATE state bit is not set.
GFX_GOL_EDITBOX_DRAW_FOCUS_STATE	Draw bit to indicate focus must be redrawn.
GFX_GOL_EDITBOX_DRAW_STATE	Draw bit to indicate object must be redrawn.
GFX_GOL_EDITBOX_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

GFX GOL edit box state.

Typedef: GFX_GOL_EDITBOX_STATE

This enumeration specifies the different states of the Edit Box object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_PICTURECONTROL Structure

Defines the structure used for the Picture Control object.

File

gfx_gol_picture.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  GFX_RESOURCE_HDR * pImage;
  int8_t scaleFactor;
  uint16_t imageLeft;
  uint16_t imageTop;
  uint16_t imageRight;
  uint32_t * stream;
  uint8_t count;
  uint8_t delay;
  uint16_t imageBottom;
  GFX_PARTIAL_IMAGE_PARAM partial;
} GFX_GOL_PICTURECONTROL;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
GFX_RESOURCE_HDR * pImage;	Pointer to the image
int8_t scaleFactor;	Scale factor for the bitmap
uint16_t imageLeft;	image left position when drawn
uint16_t imageTop;	image top position when drawn
uint16_t imageRight;	image right position when drawn

uint8_t count;	Count for the number of bitmaps to be streamed
uint8_t delay;	Delay in between the streaming of bitmaps
uint16_t imageBottom;	image bottom position when drawn
GFX_PARTIAL_IMAGE_PARAM partial;	structure containing partial image data

Description

GFX GOL picture control structure.

Typedef: GFX_GOL_PICTURECONTROL

Defines the parameters required for a Picture Control object. Object is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_PICTURECONTROLCONTROL_STATE Enumeration

Specifies the different states of the Picture Control object.

File

gfx_gol_picture.h

C

```
typedef enum {
  GFX_GOL_PICTURECONTROL_DISABLED_STATE,
  GFX_GOL_PICTURECONTROL_FRAME_STATE,
  GFX_GOL_PICTURECONTROL_STREAM_STATE,
  GFX_GOL_PICTURECONTROL_DRAW_STATE,
  GFX_GOL_PICTURECONTROL_HIDE_STATE
} GFX_GOL_PICTURECONTROLCONTROL_STATE;
```

Members

Members	Description
GFX_GOL_PICTURECONTROL_DISABLED_STATE	Property bit to indicate object is disabled.
GFX_GOL_PICTURECONTROL_FRAME_STATE	Property bit to indicate that the object will have a frame.
GFX_GOL_PICTURECONTROL_STREAM_STATE	Property bit to indicate Picture is streaming. This feature is available only when the hardware can support streaming of images.
GFX_GOL_PICTURECONTROL_DRAW_STATE	Draw bit to indicate object must be redrawn.
GFX_GOL_PICTURECONTROL_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

GFX GOL picture control, control state.

Typedef: GFX_GOL_PICTURECONTROLCONTROL_STATE

This enumeration specifies the different states of the Picture Control object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_COMMON_STATE_BITS Enumeration

Common Object States.

File

gfx_gol.h

C

```
typedef enum {
  GFX_GOL_FOCUSED,
  GFX_GOL_DISABLED,
```

```
  GFX_GOL_HIDE,
  GFX_GOL_DRAW,
  GFX_GOL_DRAW_FOCUS,
  GFX_GOL_DRAW_UPDATE
} GFX_GOL_COMMON_STATE_BITS;
```

Members

Members	Description
GFX_GOL_FOCUSED	Focus state bit
GFX_GOL_DISABLED	Disabled state bit.
GFX_GOL_HIDE	Object hide state bit. Object will be hidden from the screen by drawing over it the common background color.
GFX_GOL_DRAW	Object redraw state bits. The whole Object must be redrawn.
GFX_GOL_DRAW_FOCUS	Focus redraw state bit. The focus rectangle must be redrawn.
GFX_GOL_DRAW_UPDATE	Partial Object redraw state bits. A part or parts of the Object must be redrawn to show updated state.

Description

GFX GOL common object states

Typedef: Common Object States

The following macros defines the common Object State bits.

Remarks

None.

GFX_GOL_MESSAGE_CALLBACK_FUNC Type

Message callback function definition. This application defined function allows the application to perform application specific processing of user messages.

File

gfx_gol.h

C

```
typedef bool (* GFX_GOL_MESSAGE_CALLBACK_FUNC)(GFX_GOL_TRANSLATED_ACTION, GFX_GOL_OBJ_HEADER *,
GFX_GOL_MESSAGE *);
```

Returns

true - When true is returned, the object will set its state depending on the translated messages. false - When false is returned, the object will not process the translated message and will assume the application has performed necessary action on the message.

Description

GFX GOL message callback function pointer

This application defined function is called by the GFX_GOL_ObjectMessage() function allowing the application the opportunity to process the user messages and customize object behavior as well as application controlled functions.

GFX_GOL_ObjectMessage() calls this function when a valid message for an object in the active list is received. Application implements any action for the message in this callback function. If this callback function returns true, the message for the object will be processed using the default action of the object. If false is returned, the default action will not be performed. In this case, it is assumed that this callback function has performed the appropriate changes to the states of the objects.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
GFX_GOL_TRANSLATED_ACTION	Translated message for the object
GFX_GOL_OBJ_HEADER *	Pointer to the object that processed the message.
GFX_GOL_MESSAGE *	Pointer to the message from user.

Function

typedef bool (*GFX_GOL_MESSAGE_CALLBACK_FUNC)
(
GFX_GOL_TRANSLATED_ACTION,
GFX_GOL_OBJ_HEADER *,
GFX_GOL_MESSAGE *
);

GFX_GOL_TEXTENTRY_KEYMEMBER Structure

Defines the structure used to describe a key in the Text Entry object.

File

gfx_gol_text_entry.h

C

```
typedef struct {
  uint16_t left;
  uint16_t top;
  uint16_t right;
  uint16_t bottom;
  uint16_t index;
  uint16_t state;
  bool update;
  GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE command;
  GFX_XCHAR * pKeyName;
  int16_t textWidth;
  int16_t textHeight;
  void * pNextKey;
} GFX_GOL_TEXTENTRY_KEYMEMBER;
```

Members

Members	Description
uint16_t left;	Left position of the key
uint16_t top;	Top position of the key
uint16_t right;	Right position of the key
uint16_t bottom;	Bottom position of the key
uint16_t index;	Index of the key in the list
uint16_t state;	State of the key. Either Pressed (GFX_GOL_TEXTENTRY_KEY_PRESSED_STATE) or Released (0)
bool update;	flag to indicate key is to be redrawn with the current state
GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE command;	Command of the key. Either GFX_GOL_TEXTENTRY_DELETE_COM, GFX_GOL_TEXTENTRY_SPACE_COM or GFX_GOL_TEXTENTRY_ENTER_COM.
GFX_XCHAR * pKeyName;	Pointer to the custom text assigned to the key. This is displayed over the face of the key.
int16_t textWidth;	Computed text width, done at creation. Used to predict size and position of text on the key face.
int16_t textHeight;	Computed text height, done at creation. Used to predict size and position of text on the key face.
void * pNextKey;	Pointer to the next key parameters.

Description

GFX GOL text entry key member structure.

Typedef: GFX_GOL_TEXTENTRY_KEYMEMBER

Defines the structure used to describe a key in the Text Entry object. Strings displayed on each key is assigned here as well as the commands if key is assigned a command key.

Remarks

None.

GFX_GOL_CUSTOMCONTROL_STATE Enumeration

Specifies the different states of the Custom Control object.

File

gfx_gol_custom_control.h

C

```
typedef enum {
  GFX_GOL_CUSTOMCONTROL_DISABLED_STATE,
  GFX_GOL_CUSTOMCONTROL_DRAW_BAR_STATE,
  GFX_GOL_CUSTOMCONTROL_DRAW_STATE,
  GFX_GOL_CUSTOMCONTROL_HIDE_STATE
} GFX_GOL_CUSTOMCONTROL_STATE;
```

Members

Members	Description
GFX_GOL_CUSTOMCONTROL_DISABLED_STATE	Property bit to indicate object is disabled.
GFX_GOL_CUSTOMCONTROL_DRAW_BAR_STATE	Draw bit to indicate focus must be redrawn.
GFX_GOL_CUSTOMCONTROL_DRAW_STATE	Draw bit to indicate object must be redrawn.
GFX_GOL_CUSTOMCONTROL_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

GFX GOL custom control state.

Typedef: GFX_GOL_CUSTOMCONTROL_STATE

This enumeration specifies the different states of the Custom Control object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_CLIENT_STATUS Enumeration

Enumerated data type that identifies the GFX Module Client Status.

File

gfx_gol.h

C

```
typedef enum {
  GFX_GOL_CLIENT_STATUS_CLOSED,
  GFX_GOL_CLIENT_STATUS_READY
} GFX_GOL_CLIENT_STATUS;
```

Members

Members	Description
GFX_GOL_CLIENT_STATUS_CLOSED	Client is closed or the specified handle is invalid
GFX_GOL_CLIENT_STATUS_READY	Client is ready

Description

GFX GOL Client Status

This enumeration defines the possible status of the GFX Module Client.

Remarks

None.

GFX_GOL_OBJECT_TASK Enumeration

Lists the different states that GFX GOL task routine can have.

File

gfx_gol.h

C

```
typedef enum {
  GFX_GOL_TASK_STATE_OPEN_MODULE,
  GFX_GOL_TASK_STATE_RUNNING
} GFX_GOL_OBJECT_TASK;
```

Members

Members	Description
GFX_GOL_TASK_STATE_OPEN_MODULE	open module for client
GFX_GOL_TASK_STATE_RUNNING	run module for client

Description

GFX GOL task states

This enumeration lists the different states that GFX GOL task routine can have.

Remarks

None.

GFX_GOL_STATES Enumeration

Defines the various states that can be achieved by the GFX GOL operation.

File

gfx_gol.h

C

```
typedef enum {
  GFX_GOL_STATE_BUSY,
  GFX_GOL_STATE_INIT,
  GFX_GOL_STATE_INITIALIZED
} GFX_GOL_STATES;
```

Members

Members	Description
GFX_GOL_STATE_BUSY	Module state busy
GFX_GOL_STATE_INIT	Module state init
GFX_GOL_STATE_INITIALIZED	Module state initialized

Description

GFX GOL Machine States

This enumeration defines the various states that can be achieved by the GOL operation.

Remarks

None.

GFX_AT_SCAN_CODES Enumeration

The following macros are the supported AT Keyboard scan codes.

File

gfx_gol_scan_codes.h

C

```
typedef enum {
  SCAN_CR_PRESSED = 0x1C,
  SCAN_CR_RELEASED = 0x9C,
  SCAN_CRA_PRESSED = 0x2C,
  SCAN_CRA_RELEASED = 0xAC,
  SCAN_DEL_PRESSED = 0x53,
  SCAN_DEL_RELEASED = 0xD3,
  SCAN_BS_PRESSED = 0x0E,
  SCAN_BS_RELEASED = 0x8E,
  SCAN_TAB_PRESSED = 0x0F,
```

```
  SCAN_TAB_RELEASED = 0x8F,
  SCAN_HOME_PRESSED = 0x47,
  SCAN_HOME_RELEASED = 0xC7,
  SCAN_END_PRESSED = 0x4F,
  SCAN_END_RELEASED = 0xCF,
  SCAN_PGUP_PRESSED = 0x49,
  SCAN_PGUP_RELEASED = 0xC9,
  SCAN_PGDOWN_PRESSED = 0x51,
  SCAN_PGDOWN_RELEASED = 0xD1,
  SCAN_UP_PRESSED = 0x48,
  SCAN_UP_RELEASED = 0xC8,
  SCAN_DOWN_PRESSED = 0x50,
  SCAN_DOWN_RELEASED = 0xD0,
  SCAN_LEFT_PRESSED = 0x4B,
  SCAN_LEFT_RELEASED = 0xCB,
  SCAN_RIGHT_PRESSED = 0x4D,
  SCAN_RIGHT_RELEASED = 0xCD,
  SCAN_SPACE_PRESSED = 0x39,
  SCAN_SPACE_RELEASED = 0xB9
} GFX_AT_SCAN_CODES;
```

Members

Members	Description
SCAN_CR_PRESSED = 0x1C	Carriage return pressed.
SCAN_CR_RELEASED = 0x9C	Carriage return released.
SCAN_CRA_PRESSED = 0x2C	Carriage return alternate pressed.
SCAN_CRA_RELEASED = 0xAC	Carriage return alternate released.
SCAN_DEL_PRESSED = 0x53	Delete key pressed.
SCAN_DEL_RELEASED = 0xD3	Delete key released.
SCAN_BS_PRESSED = 0x0E	Back space key pressed.
SCAN_BS_RELEASED = 0x8E	Back space key released.
SCAN_TAB_PRESSED = 0x0F	Tab key pressed.
SCAN_TAB_RELEASED = 0x8F	Tab key released.
SCAN_HOME_PRESSED = 0x47	Home key pressed.
SCAN_HOME_RELEASED = 0xC7	Home key released.
SCAN_END_PRESSED = 0x4F	End key pressed.
SCAN_END_RELEASED = 0xCF	End key released.
SCAN_PGUP_PRESSED = 0x49	Page up key pressed.
SCAN_PGUP_RELEASED = 0xC9	Page up key released.
SCAN_PGDOWN_PRESSED = 0x51	Page down key pressed.
SCAN_PGDOWN_RELEASED = 0xD1	Page down key released.
SCAN_UP_PRESSED = 0x48	Up key pressed.
SCAN_UP_RELEASED = 0xC8	Up key released.
SCAN_DOWN_PRESSED = 0x50	Down key pressed.
SCAN_DOWN_RELEASED = 0xD0	Down key released.
SCAN_LEFT_PRESSED = 0x4B	Left key pressed.
SCAN_LEFT_RELEASED = 0xCB	Left key released.
SCAN_RIGHT_PRESSED = 0x4D	Right key pressed.
SCAN_RIGHT_RELEASED = 0xCD	Right key released.
SCAN_SPACE_PRESSED = 0x39	Space key pressed.
SCAN_SPACE_RELEASED = 0xB9	Space key released.

Description

GFX AT keyboard scan codes.

Typedef: GFX_AT_SCAN_CODES

The following macros are the supported AT Keyboard scan codes.

Remarks

None.

GFX_GOL_SURFACE Structure

Defines the structure used for the Surface object.

File

gfx_gol_surface.h

C

```
typedef struct {
  GFX_GOL_OBJ_HEADER hdr;
  uint16_t textHeight;
  GFX_XCHAR * pText;
  GFX_GOL_SURFACE_CALLBACK_FUNC callback;
  GFX_RESOURCE_HDR * pImage;
} GFX_GOL_SURFACE;
```

Members

Members	Description
GFX_GOL_OBJ_HEADER hdr;	Generic header for all Objects (see GFX_GOL_OBJ_HEADER).
uint16_t textHeight;	Precomputed text height
GFX_XCHAR * pText;	Pointer to the title text
GFX_GOL_SURFACE_CALLBACK_FUNC callback;	text alignment
GFX_RESOURCE_HDR * pImage;	Pointer to the image for the title bar

Description

Typedef: GFX_GOL_SURFACE

Defines the parameters required for a Surface object. Object is drawn with the defined shape parameters and values set on the given fields.

Remarks

None.

GFX_GOL_SURFACE_CALLBACK_FUNC Type

Surface callback function definition. This application defined function allows the application to perform application specific rendering.

File

gfx_gol_surface.h

C

```
typedef GFX_STATUS (* GFX_GOL_SURFACE_CALLBACK_FUNC)(void);
```

Returns

true - is returned when application rendering is done. false - is returned when application rendering is not yet finished.

Description

This callback function is implemented by the application. This is called inside the GFX_GOL_ObjectListDraw() function when the drawing of objects in the active list is completed.

Any application specific rendering must be performed on this callback function so no object rendering will be affected by the application calls to primitive rendering functions. Application setting the drawing color, line style, fill style, text string cursor position and current font will not affect the object rendering. This is also the safe place to modify the active list.

When the application has performed its own primitive rendering calls, this function must return true to inform the GFX_GOL_ObjectListDraw() that it is done rendering and checking for object drawing or redrawing can continue.

Preconditions

None.

Example

None.

Function

typedef GFX_STATUS (*GFX_GOL_SURFACE_CALLBACK_FUNC) (void);

GFX_GOL_SURFACE_STATE Enumeration

Specifies the different states of the Surface object.

File

gfx_gol_surface.h

C

```
typedef enum {
  GFX_GOL_SURFACE_FOCUSED_STATE,
  GFX_GOL_SURFACE_DISABLED_STATE,
  GFX_GOL_SURFACE_DRAW_APP_STATE,
  GFX_GOL_SURFACE_DRAW_CLIENT_STATE,
  GFX_GOL_SURFACE_DRAW_STATE,
  GFX_GOL_SURFACE_HIDE_STATE
} GFX_GOL_SURFACE_STATE;
```

Members

Members	Description
GFX_GOL_SURFACE_FOCUSED_STATE	Property bit for focus state.
GFX_GOL_SURFACE_DISABLED_STATE	Property bit for disabled state.
GFX_GOL_SURFACE_DRAW_APP_STATE	Draw bit to indicate title area must be redrawn.
GFX_GOL_SURFACE_DRAW_CLIENT_STATE	Draw bit to indicate client area must be redrawn.
GFX_GOL_SURFACE_DRAW_STATE	Draw bit to indicate object must be redrawn.
GFX_GOL_SURFACE_HIDE_STATE	Draw bit to indicate object must be removed from screen.

Description

Typedef: GFX_GOL_SURFACE_STATE

This enumeration specifies the different states of the Surface object used in the library.

For the Property State bits, more than one of these values may be OR'd together to create a complete property state.

For the Draw State bits, hide draw bit has higher priority than the draw bit. Any of these two can be combined with other draw bits to create a complete draw state.

To test a value of any of the state types, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

Remarks

None.

GFX_GOL_OBJ_SCHEME Structure

This structure specifies the style scheme components of an object.

File

gfx_gol_scheme.h

C

```
typedef struct {
  GFX_COLOR EmbossDkColor;
  GFX_COLOR EmbossLtColor;
  GFX_COLOR TextColor0;
  GFX_COLOR TextColor1;
  GFX_COLOR TextColorDisabled;
  GFX_COLOR Color0;
  GFX_COLOR Color1;
  GFX_COLOR ColorDisabled;
  GFX_RESOURCE_HDR * pFont;
  GFX_FILL_STYLE fillStyle;
  GFX_COLOR CommonBkColor;
  uint16_t CommonBkLeft;
  uint16_t CommonBkTop;
  GFX_BACKGROUND_TYPE CommonBkType;
  GFX_RESOURCE_HDR * pCommonBkImage;
  uint16_t AlphaValue;
  GFX_COLOR gradientStartColor;
  GFX_COLOR gradientEndColor;
```

```
  uint16_t EmbossSize;
} GFX_GOL_OBJ_SCHEME;
```

Members

Members	Description
GFX_COLOR EmbossDkColor;	Emboss dark color used for 3d effect.
GFX_COLOR EmbossLtColor;	Emboss light color used for 3d effect.
GFX_COLOR TextColor0;	Character color 0 used for objects that supports text.
GFX_COLOR TextColor1;	Character color 1 used for objects that supports text.
GFX_COLOR TextColorDisabled;	Character color used when object is in a disabled state.
GFX_COLOR Color0;	Color 0 usually assigned to an Object state.
GFX_COLOR Color1;	Color 1 usually assigned to an Object state.
GFX_COLOR ColorDisabled;	Color used when an Object is in a disabled state.
GFX_RESOURCE_HDR * pFont;	Font selected for the scheme.
GFX_FILL_STYLE fillStyle;	must be set to a gradient type when using gradient
GFX_COLOR CommonBkColor;	Background color used to hide Objects.
uint16_t CommonBkLeft;	Horizontal starting position of the background.
uint16_t CommonBkTop;	Vertical starting position of the background.
GFX_BACKGROUND_TYPE CommonBkType;	Specifies the type of background to use.
GFX_RESOURCE_HDR * pCommonBkImage;	Pointer to the background image used. Set this
uint16_t AlphaValue;	Alpha value used for alpha blending
GFX_COLOR gradientStartColor;	start color of the gradient fill
GFX_COLOR gradientEndColor;	end color of the gradient fill
uint16_t EmbossSize;	Emboss size of the panel for 3-D effect. Set to zero if not used.

Description

Typedef: GFX_GOL_OBJ_SCHEME

This structure specifies the style scheme components of an object. All objects will use the style scheme when rendering. Refer to specific object documentation on how the style scheme colors are utilized by the object.

The style scheme allows objects to show 3D effects as well as feedback to users. For example, in Button objects, a press and release effect on the Buttons are easily shown by manipulating the colors when the object is drawn with a pressed state and released state. The style scheme also allows effects such as gradients and alpha blending. When using alpha blending, the style scheme requires objects to be associated with background information.

A background can be a flat color background or an image. The usage of a background requires the background dimension to be larger than the object. In other words, the object should be drawn within the background dimension. Multiple objects can share a common background. As long as all the objects are drawn within the dimension of the background they can share a common background.

The supported background types are (See GFX_BACKGROUND_TYPE):

- GFX_BACKGROUND_COLOR - this type will set the common background to be a flat color. The color used is specified by CommonBkType.
- GFX_BACKGROUND_IMAGE - this type will set the common background to be an image. The image used is specified by pCommonBkImage.

When an object is associated with a background, it can be easily hidden or redrawn with some effects (for example alpha blending with the background). The background information allows the redrawing of the background with the object without the need to manually refreshing the background using primitive calls by the application.

Remarks

None.

GFX_GOL_SurfaceImageGet Macro

This function gets the image used.

File

gfx_gol_surface.h

C

```
#define GFX_GOL_SurfaceImageGet(pObject, pImage) \
                                (((GFX_GOL_SURFACE *)pObject)->pImage)
```

Returns

Pointer to the image resource.

Description

This function gets the image used.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.

Function

GFX_RESOURCE_HDR *GFX_GOL_SurfaceImageGet(
GFX_GOL_SURFACE *pObject)

GFX_GOL_SurfaceImageSet Macro

This function sets the image used in the object.

File

gfx_gol_surface.h

C

```
#define GFX_GOL_SurfaceImageSet(pObject, image) \
                                (((GFX_GOL_SURFACE *)pObject)->pImage = image)
```

Returns

None.

Description

This function sets the image used in the object.

Preconditions

Object must exist in memory.

Example

None.

Parameters

Parameters	Description
pObject	pointer to the object.
pImage	pointer to the image to be set.

Function

void GFX_GOL_SurfaceImageSet(
GFX_GOL_SURFACE *pObject,
GFX_RESOURCE_HDR *pImage)

System and Client Interfaces

Functions

	Name	Description
	GFX_Open	Opens the specified GFX library instance and returns a handle to it.
	GFX_Close	Closes a client instance of the GFX library.
	GFX_Initialize	Initialize the GFX Library.
	GFX_Deinitialize	Deinitializes the specified instance of the GFX Module.

	GFX_Status	Gets the current status of the GFX Module.
	GFX_Tasks	Maintains the GFX module state machine. It manages the GFX Module object list items and responds to GFX Module primitive events.
	GFX_GOL_Deinitialize	Deinitializes the specified instance of the GFX Module.
	GFX_GOL_Initialize	Initialize the GFX GOL Library.
	GFX_GOL_Open	opens an instance of the GFX GOL Library.
	GFX_GOL_Tasks	Maintains the GFX GOL module state machine. It manages the GFX Module object list items and responds to GFX Module primitive events.
	GFX_GOL_PreemptionLevelGet	This function returns the current preemption level.
	ImageDecoderInit	This function initializes the global variables to 0 and then initializes the driver. This must be called once before any other function of the library is called
	ImageDecodeTask	This function completes one small part of the image decode function
	ImageDecode	This function decodes and displays the image on the screen
	ImageLoopCallbackRegister	This function registers the loop callback function so that the decoder calls this function in every decoding loop. This can be used by the application program to do maintenance activities such as fetching data, updating the display, etc...
	IMG_vSetboundaries	
	ImageDecoderParceHeader	This is function ImageDecoderParceHeader.
	ImagePixelIntoArray	This is function ImagePixelIntoArray.
	ImagePixelOutput	This is function ImagePixelOutput.
	ImageRegisterFrameCompleteCallback	This is function ImageRegisterFrameCompleteCallback.
	ImageRegisterImageCompleteCallback	Function prototypes
	ImageRegisterImageFailCallback	This is function ImageRegisterImageFailCallback.

Data Types and Constants

	Name	Description
	GFX_FONT_GLYPH_ENTRY	The structure describing the glyph entry in fonts.
	GFX_FONT_GLYPH_ENTRY_EXTENDED	The structure describing the intended glyph entry in fonts.
	GFX_FONT_HEADER	The structure used to define the font header.
	GFX_MCHP_BITMAP_HEADER	The structure used to define the Microchip bitmap header.
	GFX_PARTIAL_IMAGE_PARAM	Partial Image information structure.
	GFX_RECTANGULAR_AREA	A generic rectangular area structure.
	GFX_RESOURCE	Specifies the different resource types in the library.
	GFX_RESOURCE_BINARY	Defines the structure used for the binary type resource.
	GFX_RESOURCE_FONT	Defines the structure used for the font type resource.
	GFX_RESOURCE_HDR	Defines the structure used for the resource types.
	GFX_RESOURCE_IMAGE	Defines the structure used for the image type resource.
	GFX_RESOURCE_PALETTE	Defines the structure used for the palette type resource.
	GFX_STATUS	Rendering status.
	GFX_STATUS_BIT	Additional rendering status.
	int16_gfx_image_prog	This is type int16_gfx_image_prog.
	int32_gfx_image_prog	This is type int32_gfx_image_prog.
	int8_gfx_image_prog	This is type int8_gfx_image_prog.
	uint16_gfx_image_prog	This is type uint16_gfx_image_prog.
	uint32_gfx_image_prog	This is type uint32_gfx_image_prog.
	uint8_gfx_image_prog	GFX Image data Types
	GFX_ALIGNMENT	Summary of the different text alignment supported in the library.
	GFX_BACKGROUND	Background information structure.
	GFX_BACKGROUND_TYPE	Specifies the different background fill types.
	GFX_DOUBLE_BUFFERING_MODE	Structure used for double buffering management.
	GFX_FEATURE_STATUS	States of a feature that can be enabled/disabled at run time.
	GFX_FILL_STYLE	Specifies the available fill styles.
	GFX_FONT_ANTIALIAS_TYPE	Summary of the transparency types in text anti-aliasing.
	GFX_FONT_SPACE	Font space section. The fonts can be located in psv (constant) or program space in PIC24/dsPIC MCUs. This define allows for switching of the pointer type used to access the font structure in memory See: GraphicsConfig.h for the application define.
	GFX_UXCHAR	This is macro GFX_UXCHAR.

	GFX_XCHAR	Configuration for the text character size.
	GRAPHICS_LIBRARY_VERSION	////////////////// GRAPHICS_LIBRARY_VERSION ////////////////////// MSB is version, LSB is subversion
	GFX_COMP_MASK	This is macro GFX_COMP_MASK.
	GFX_MEM_MASK	This is macro GFX_MEM_MASK.
	GFX_TYPE_MASK	This is macro GFX_TYPE_MASK.
	MCHP_BITMAP_NORMAL	no compression, palette is present for color depth = 8, 4 and 1 BPP
	MCHP_BITMAP_PALETTE_STR	palette is provided as a separate object (see PALETTE_HEADER) for color depth = 8, 4, and 1 BPP, ID to the palette is embedded in the bitmap.
	int16_prog	This is type int16_prog.
	int16_prog_pack	This is type int16_prog_pack.
	int32_prog	This is type int32_prog.
	int32_prog_pack	This is type int32_prog_pack.
	int8_prog	This is type int8_prog.
	int8_prog_pack	This is type int8_prog_pack.
	uint16_prog	This is type uint16_prog.
	uint16_prog_pack	This is type uint16_prog_pack.
	uint32_prog	This is type uint32_prog.
	uint32_prog_pack	This is type uint32_prog_pack.
	uint8_prog	General program space data types
	uint8_prog_pack	This is type uint8_prog_pack.
	GFX_CLIENT_OBJ	GFX client object structure.
	GFX_CLIENT_STATUS	Enumerated data type that identifies the GFX Module Client Status.
	GFX_OBJ	Defines the object required for interaction with the GFX Library.
	GFX_OBJECT_TASK	Lists the different states that GFX task routine can have.
	GFX_STATES	Defines the various states that can be achieved by the GFX operation.
	_IMG_FILE_FORMAT	IMG_ImageFileFormat specifies all the supported image file formats
	_IMG_FILE_SYSTEM_API	IMG_FileSystemAPI holds function pointers to the used File System APIs
	_IMG_PIXEL_XY_RGB_888	IMG_PixelRgb holds the RGB information of a pixel in BYTES
	FileFeof	This is type FileFeof.
	FileRead	Typedefs of the used File System APIs
	FileSeek	This is type FileSeek.
	FileTell	This is type FileTell.
	IMG_FILE_FORMAT	IMG_ImageFileFormat specifies all the supported image file formats
	IMG_FILE_SYSTEM_API	IMG_FileSystemAPI holds function pointers to the used File System APIs
	IMG_LOOP_CALLBACK	IMG_LoopCallback is a callback function which is called in every loop of the decoding cycle so that user application can do some maintenance activities
	IMG_PIXEL_OUTPUT	IMG_PixelOutput is a callback function which receives the color information of the output pixel
	IMG_PIXEL_XY_RGB_888	IMG_PixelRgb holds the RGB information of a pixel in BYTES
	__IMAGEDECODER_H__	This is macro __IMAGEDECODER_H__.
	_PutPixel	This is macro _PutPixel.
	ImageAbort	This function sets the Image Decoder's Abort flag so that decoding aborts in the next decoding loop.
	ImageFullScreenDecode	This function decodes and displays the image on the screen in fullscreen mode with center aligned and downscaled if required
	IMG_ALIGN_CENTER	Flags
	IMG_DECODE_ABORTED	This is macro IMG_DECODE_ABORTED.
	IMG_DOWN_SCALE	This is macro IMG_DOWN_SCALE.
	IMG_FEOF	This is macro IMG_FEOF.
	IMG_FILE	This is macro IMG_FILE.
	IMG_FREAD	This is macro IMG_FREAD.
	IMG_FSEEK	This is macro IMG_FSEEK.
	IMG_FTELL	This is macro IMG_FTELL.
	IMG_SCREEN_HEIGHT	This is macro IMG_SCREEN_HEIGHT.
	IMG_SCREEN_WIDTH	The individual image decoders use these defines instead of directly using those provided in the Display driver file

	IMG_vCheckAndAbort	This is macro IMG_vCheckAndAbort.
	IMG_vLoopCallback	This is macro IMG_vLoopCallback.
	IMG_vPutPixel	This is macro IMG_vPutPixel.
	GFX_PALETTE_ENTRY	Defines the union used for each entry in the palette table.
	GFX_TEXT_CURSOR_TYPE	Enumeration defines types of text cursor supported.
	IMG_LOOP_FRAMECOMPLETE	IMG_FrameCompleteCallback is a callback function which is called at completion decoding current frame so that user application can do some activities
	IMG_LOOP_IMAGECOMPLETE	IMG_FrameCompleteCallback is a callback function which is called at completion decoding current frame so that user application can do some activities
	IMG_LOOP_IMAGEFAIL	IMG_FrameCompleteCallback is a callback function which is called at completion decoding current frame so that user application can do some activities
	IMG_vFrameComplete	This is macro IMG_vFrameComplete.
	IMG_vImageComplete	This is macro IMG_vImageComplete.
	IMG_vImageFail	This is macro IMG_vImageFail.
	IMG_vPixelArray	This is macro IMG_vPixelArray.
	IMG_vSetColor	This is macro IMG_vSetColor.

Description

This section describes the Graphics Library System and Client API.

Functions

GFX_Open Function

Opens the specified GFX library instance and returns a handle to it.

File

gfx.h

C

```
GFX_HANDLE GFX_Open(SYS_MODULE_OBJ gfxObject);
```

Returns

If successful, the function returns a valid open instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is GFX_HANDLE_INVALID.

Description

This function opens the specified GFX library instance and provides a handle that must be provided to all other client-level operations to identify the caller and the instance of the driver.

Remarks

The handle returned is valid until the GFX_Close function is called.

Preconditions

The GFX_Initialize function must have been called before calling this function.

Example

```
GFX_HANDLE  handle;

handle = GFX_Open ( gfxObject );

if ( GFX_HANDLE_INVALID == handle )
{
    // Unable to open the library
}
```

Parameters

Parameters	Description
object	Identifier for the object instance to be opened

Function

GFX_HANDLE GFX_Open(SYS_MODULE_OBJ gfxObject)

GFX_Close Function

Closes a client instance of the GFX library.

File

gfx.h

C

```
void GFX_Close(GFX_HANDLE gfxHandle);
```

Returns

None

Description

This routine closes a client instance of the GFX library.

Preconditions

The GFX_Open routine must have been called for the specified handle.

Example

```
GFX_HANDLE handle;  // Returned from GFX_Open

GFX_Close(handle);
```

Parameters

Parameters	Description
handle	A valid handle, returned from the open routine

Function

void GFX_Close(GFX_HANDLE handle)

GFX_Initialize Function

Initialize the GFX Library.

File

gfx.h

C

```
SYS_MODULE_OBJ GFX_Initialize(const SYS_MODULE_INDEX moduleIndex, const SYS_MODULE_INIT * const moduleInit);
```

Returns

If successful, returns a valid handle to a device layer object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initialize the GFX Library primitive layer and Object Layer if enabled. The following default settings are set when this function is called.

1. font - Set to NULL. GFX_FontSet() must be called prior to any text rendering.
2. line type - Set to GFX_LINE_TYPE_THIN_SOLID (see GFX_LINE_STYLE).
3. fill type - Set to GFX_FILL_TYPE_COLOR (see GFX_FILL_STYLE).
4. text anti-alias type - Set to GFX_FONT_ANTIALIAS_OPAQUE (see GFX_FONT_ANTIALIAS_TYPE). This only affects fonts with anti-aliasing enabled.
5. Set transparent color feature in image draw functions to be disabled.
6. Set alpha blending value to 100 (or no alpha blending) if alpha blending feature is enabled.
7. Set background information to no background.

This function does not clear the screen and does not assign any color to the currently set color. Application should set the color and clear the screen.

Remarks

This routine must be called before other GFX library functions.

Preconditions

None.

Example

```
GFX_INIT gfxInit;
SYS_MODULE_OBJ gfxObj;

// GFX Module initialization
gfxObj = GFX_Initialize(GFX_INDEX_0, &gfxInit);

if (SYS_MODULE_OBJ_INVALID == gfxObj)
{
    // Handle error
}
```

Parameters

Parameters	Description
moduleIndex	client instance request.
moduleInit	initialization data for the instance.

Function

void GFX_Initialize (const SYS_MODULE_INDEX moduleIndex,
const SYS_MODULE_INIT * const moduleInit);

GFX_Deinitialize Function

Deinitializes the specified instance of the GFX Module.

File

gfx.h

C

```
void GFX_Deinitialize(SYS_MODULE_OBJ moduleObject);
```

Returns

None.

Description

Deinitializes the specified instance of the GFX Module. This function will also deinitialize the GOL, PRIMITIVE, and driver and conclude all GFX related operations. All internal data structures will be reset.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function GFX_Deinitialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from GFX_Initialize
SYS_STATUS          status;


GFX_Deinitialize(object);

status = GFX_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
```

```
}
```

Parameters

Parameters	Description
object	GFX layer object handle, returned from the GFX_Initialize routine

Function

void GFX_Deinitialize(SYS_MODULE_OBJ object)

GFX_Status Function

Gets the current status of the GFX Module.

File

gfx.h

C

```
SYS_STATUS GFX_Status(SYS_MODULE_OBJ gfxObject);
```

Returns

SYS_STATUS_READY - Indicates that the GFX layer is ready for operations.

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized.

Description

This routine provides the current status of the GFX Module.

Remarks

None.

Preconditions

Function GFX_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_INDEX    index;     // Returned from GFX_Initialize
SYS_STATUS          status;

status = GFX_Status (object);
if (SYS_STATUS_READY == status)
{
    // This means the library can be used, using for example
    // GFX_DrawLine() function.
}
```

Parameters

Parameters	Description
object	GFX object index, returned from the GFX_Initialize routine

Function

SYS_STATUS GFX_Status (SYS_MODULE_OBJ gfxObject)

GFX_Tasks Function

Maintains the GFX module state machine. It manages the GFX Module object list items and responds to GFX Module primitive events.

File

gfx.h

C

```
void GFX_Tasks(SYS_MODULE_OBJ gfxObject);
```

Returns

None.

Description

Maintains the GFX module state machine. It manages the GFX Module object list items and responds to GFX Module primitive events. This function should be called from the SYS_Tasks() function.

Remarks

This routine is normally not called directly by an application.

Preconditions

None.

Example

```
while (true)
{
    GFX_Tasks (GFX_INDEX_0);

    // Do other tasks
}
```

Parameters

Parameters	Description
index	Object index for the specified module instance.

Function

void GFX_Tasks (SYS_MODULE_INDEX index);

GFX_GOL_Deinitialize Function

Deinitializes the specified instance of the GFX Module.

File

gfx_gol.h

C

```
void GFX_GOL_Deinitialize(SYS_MODULE_OBJ moduleObject);
```

Returns

None.

Description

Deinitializes the specified instance of the GFX Module. All internal data structures will be reset.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function GFX_GOL_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from GFX_GOL_Initialize
SYS_STATUS          status;


GFX_GOL_Deinitialize(object);

status = GFX_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	GFX layer object handle, returned from the GFX_GOL_Initialize routine

Function

void GFX_GOL_Deinitialize(SYS_MODULE_OBJ object)

GFX_GOL_Initialize Function

Initialize the GFX GOL Library.

File

gfx_gol.h

C

```
SYS_MODULE_OBJ GFX_GOL_Initialize(const SYS_MODULE_INDEX moduleIndex, const SYS_MODULE_INIT * const moduleInit);
```

Returns

If successful, returns a valid handle to a device layer object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initialize the GFX Library GOL layer. The following default settings are set when this function is called.

1. font -

Remarks

This routine must be called before other GFX library functions.

Preconditions

None.

Example

```
GFX_INIT gfxInit;
SYS_MODULE_OBJ gfxObj;

// GFX Module initialization
gfxObj = GFX_GOL_Initialize(GFX_INDEX_0, &gfxInit);

if (SYS_MODULE_OBJ_INVALID == gfxObj)
{
    // Handle error
}
```

Parameters

Parameters	Description
moduleIndex	client instance request.
moduleInit	initialization data for the instance.

Function

void GFX_GOL_Initialize (const SYS_MODULE_INDEX moduleIndex,
const SYS_MODULE_INIT * const moduleInit);

GFX_GOL_Open Function

opens an instance of the GFX GOL Library.

File

gfx_gol.h

C

```
GFX_HANDLE GFX_GOL_Open(const SYS_MODULE_INDEX index);
```

Function

GFX_HANDLE GFX_GOL_Open(const SYS_MODULE_INDEX index)

GFX_GOL_Tasks Function

Maintains the GFX GOL module state machine. It manages the GFX Module object list items and responds to GFX Module primitive events.

File

gfx_gol.h

C

```
void GFX_GOL_Tasks(SYS_MODULE_INDEX index);
```

Returns

None.

Description

Maintains the GFX module state machine. It manages the GFX Module object list items and responds to GFX Module primitive events. This function should be called from the SYS_Tasks() function.

Remarks

This routine is normally not called directly by an application.

Preconditions

None.

Example

```
while (true)
{
    GFX_GOL_Tasks (GFX_INDEX_0);

    // Do other tasks
}
```

Parameters

Parameters	Description
index	Object index for the specified module instance.

Function

void GFX_GOL_Tasks (SYS_MODULE_INDEX index);

GFX_GOL_PreemptionLevelGet Function

This function returns the current preemption level.

File

gfx_gol.h

C

```
GFX_PREEMPTION_LEVEL GFX_GOL_PreemptionLevelGet(SYS_MODULE_INDEX gfxIndex);
```

Returns

The current preemption level.

Description

GFX GOL Preemption level get

This function returns the current preemption level.

Preconditions

None.

Parameters

Parameters	Description
gfxIndex	Object index for the specified module instance.

Function

GFX_PREEMPTION_LEVEL GFX_GOL_PreemptionLevelGet (
SYS_MODULE_INDEX gfxIndex)

ImageDecoderInit Function

File

gfx_image_decoder.h

C

```
void ImageDecoderInit();
```

Side Effects

The graphics driver will be reset

Returns

None

Description

This function initializes the global variables to 0 and then initializes the driver. This must be called once before any other function of the library is called

Example

```
  void main(void)
{
    ImageInit();
    ...
}
```

Function

void ImageDecoderInit(void)

ImageDecodeTask Function

File

gfx_image_decoder.h

C

```
uint8_t ImageDecodeTask();
```

Side Effects

None

Returns

Status code - '1' means decoding is completed

- '0' means decoding is not yet completed, call this function again

Description

This function completes one small part of the image decode function

Example

```
      IMG_bFullScreenDecode(pImageFile, IMG_JPEG, NULL, NULL);
    while(!ImageDecodeTask());
```

Function

uint8_t ImageDecodeTask(void)

ImageDecode Function

File

gfx_image_decoder.h

C

```
uint8_t ImageDecode(IMG_FILE * pImageFile, IMG_FILE_FORMAT eImgFormat, uint16_t wStartx, uint16_t wStarty,
uint16_t wWidth, uint16_t wHeight, uint16_t wFlags, IMG_FILE_SYSTEM_API * pFileAPIs, IMG_PIXEL_OUTPUT
pPixelOutput);
```

Side Effects

None

Returns

Error code -> 0 means no error

Description

This function decodes and displays the image on the screen

Example

```
void main(void)
{
   IMG_FILE pImageFile;
   IMG_vInitialize();
   pImageFile = IMG_FOPEN("Image.jpg", "r");
   if(pImageFile == NULL)
   {
         <- Error handling ->
   }
   IMG_bDecode(pImageFile, IMG_JPEG, 0, 0, 320, 240, 0, NULL, NULL);
   IMG_FCLOSE(pImageFile);
   while(1);
}
```

Function

uint8_t ImageDecode(IMG_FILE *pImageFile, IMG_FILE_FORMAT eImgFormat, uint16_t wStartx, uint16_t wStarty, uint16_t wWidth, uint16_t wHeight, uint16_t wFlags, IMG_FILE_SYSTEM_API *pFileAPIs, IMG_PIXEL_OUTPUT pPixelOutput)

ImageLoopCallbackRegister Function

File

gfx_image_decoder.h

C

```
void ImageLoopCallbackRegister(IMG_LOOP_CALLBACK pFn);
```

Side Effects

The graphics driver will be reset

Returns

None

Description

This function registers the loop callback function so that the decoder calls this function in every decoding loop. This can be used by the application program to do maintenance activities such as fetching data, updating the display, etc...

Example

```
  void Mainantance(void)
{
```

```
    ...
}

void main(void)
{
    ImageInit();
    ImageLoopCallbackRegister(Mainantance);
    ...
}
```

Function

void ImageLoopCallbackRegister(IMG_LOOP_CALLBACK pLoopCallbackFn)

IMG_vSetboundaries Function

File

gfx_image_decoder.h

C

```
void IMG_vSetboundaries();
```

Description

******* This is not for the user *******

This is used by the individual decoders

ImageDecoderParceHeader Function

File

gfx_image_decoder.h

C

```
int8_t ImageDecoderParceHeader(IMG_FILE * pImageFile, GFX_RESOURCE_HDR* header, GFX_RESOURCE source);
```

Description

This is function ImageDecoderParceHeader.

ImagePixelIntoArray Function

File

gfx_image_decoder.h

C

```
void ImagePixelIntoArray(IMG_PIXEL_XY_RGB_888 * pPix);
```

Description

This is function ImagePixelIntoArray.

ImagePixelOutput Function

File

gfx_image_decoder.h

C

```
void ImagePixelOutput(IMG_PIXEL_XY_RGB_888 * pPix);
```

Description

This is function ImagePixelOutput.

ImageRegisterFrameCompleteCallback Function

File

gfx_image_decoder.h

C

```
void ImageRegisterFrameCompleteCallback(void* callbackFn);
```

Description

This is function ImageRegisterFrameCompleteCallback.

ImageRegisterImageCompleteCallback Function

File

gfx_image_decoder.h

C

```
void ImageRegisterImageCompleteCallback(void* callbackFn);
```

Description

Function prototypes

ImageRegisterImageFailCallback Function

File

gfx_image_decoder.h

C

```
void ImageRegisterImageFailCallback(void* callbackFn);
```

Description

This is function ImageRegisterImageFailCallback.

Data Types and Constants

GFX_FONT_GLYPH_ENTRY Structure

The structure describing the glyph entry in fonts.

File

gfx_types_font.h

C

```
typedef struct {
  uint8_t width;
  uint8_t offsetLSB;
  uint16_t offsetMSB;
} GFX_FONT_GLYPH_ENTRY;
```

Members

Members	Description
uint8_t width;	The width of the glyph.
uint8_t offsetLSB;	The least Significant byte of the glyph location offset.
uint16_t offsetMSB;	The most Significant (2) bytes of the glyph location offset.

Description

Typedef: GFX_FONT_GLYPH_ENTRY

Each character's bitmap (or glyph) in a font is described by its glyph entry structure. The entry will define the width and the offset of the character

glyph from the header of the font table.

Remarks

None.

GFX_FONT_GLYPH_ENTRY_EXTENDED Structure

The structure describing the intended glyph entry in fonts.

File

gfx_types_font.h

C

```
typedef struct {
  uint32_t offset;
  uint16_t cursorAdvance;
  uint16_t glyphWidth;
  int16_t xAdjust;
  int16_t yAdjust;
} GFX_FONT_GLYPH_ENTRY_EXTENDED;
```

Members

Members	Description
uint32_t offset;	The offset of the glyph. The offset order is: LSW_LSB LSW_MSB MSW_MSB MSW_MSB.
uint16_t cursorAdvance;	The x-value by which cursor has to advance after rendering the glyph.
uint16_t glyphWidth;	The width of the glyph.
int16_t xAdjust;	The value that adjusts the x-position.
int16_t yAdjust;	The value that adjusts the y-position.

Description

Typedef: GFX_FONT_GLYPH_ENTRY_EXTENDED

Similar to the GFX_FONT_GLYPH_ENTRY, each character's glyph in a font that supports intended characters is also described by its glyph entry structure. The difference is that the extended glyph entry will contain additional information on how the characters are overlapped.

Remarks

None.

GFX_FONT_HEADER Structure

The structure used to define the font header.

File

gfx_types_font.h

C

```
typedef struct {
  uint8_t fontID;
  uint8_t extendedGlyphEntry : 1;
  uint8_t res1 : 1;
  uint8_t bpp : 2;
  uint8_t orientation : 2;
  uint8_t res2 : 2;
  uint16_t firstChar;
  uint16_t lastChar;
  uint16_t height;
} GFX_FONT_HEADER;
```

Members

Members	Description
uint8_t fontID;	User assigned value
uint8_t extendedGlyphEntry : 1;	Extended Glyph entry flag. When set font has extended glyph feature enabled.
uint8_t res1 : 1;	Reserved for future use (must be set to 0)

uint8_t bpp : 2;	Actual BPP = 2^{bpp} • 0 - 1 BPP • 1 - 2 BPP • 2 - 4 BPP • 3 - 8 BPP
uint8_t orientation : 2;	Orientation of the character glyphs (0,90,180,270 degrees) • 00 - Normal • 01 - Characters rotated 270 degrees clockwise • 10 - Characters rotated 180 degrees • 11 - Characters rotated 90 degrees clockwise
uint8_t res2 : 2;	Reserved for future use (must be set to 0).
uint16_t firstChar;	Character code of first character (e.g. 32).
uint16_t lastChar;	Character code of last character in font (e.g. 3006).
uint16_t height;	Font characters height in pixels.

Description

Typedef: GFX_FONT_HEADER

The structure used to define the font header.

Remarks

None.

GFX_MCHP_BITMAP_HEADER Structure

The structure used to define the Microchip bitmap header.

File

gfx_types_image.h

C

```
typedef struct {
  uint8_t bitmapType;
  uint8_t colorDepth;
  int16_t height;
  int16_t width;
} GFX_MCHP_BITMAP_HEADER;
```

Members

Members	Description
uint8_t bitmapType;	type of image, information on how to render the image 0 - no compression, palette is present for color depth = 8, 4 and 1 BPP 1 - palette is provided as a separate object (see PALETTE_HEADER) for color depth = 8, 4, and 1 BPP, ID to the palette is embedded in the bitmap.
uint8_t colorDepth;	Color depth used
int16_t height;	Image height
int16_t width;	Image width

Description

Typedef: GFX_MCHP_BITMAP_HEADER

The structure used to define the Microchip bitmap header.

Remarks

None.

GFX_PARTIAL_IMAGE_PARAM Structure

Partial Image information structure.

File

gfx_types_macros.h

C

```
typedef struct {
  uint16_t width;
  uint16_t height;
  uint16_t xoffset;
  uint16_t yoffset;
} GFX_PARTIAL_IMAGE_PARAM;
```

Members

Members	Description
uint16_t width;	Partial Image width.
uint16_t height;	Partial Image height.
uint16_t xoffset;	xoffset of the partial image (with respect to the image horizontal origin).
uint16_t yoffset;	yoffset of the partial image (with respect to the image vertical origin).

Description

Typedef: GFX_PARTIAL_IMAGE_PARAM

Structure describing the partial image area to be rendered.

Remarks

None.

GFX_RECTANGULAR_AREA Structure

A generic rectangular area structure.

File

gfx_types_macros.h

C

```
typedef struct {
  uint16_t left;
  uint16_t top;
  uint16_t right;
  uint16_t bottom;
} GFX_RECTANGULAR_AREA;
```

Members

Members	Description
uint16_t left;	left most pixel of the area.
uint16_t top;	top most pixel of the area.
uint16_t right;	right most pixel of the area.
uint16_t bottom;	bottom most pixel of the area.

Description

Typedef: GFX_RECTANGULAR_AREA

Structure describing a generic rectangular area defined by the left,top and right,bottom points on the frame buffer.

Remarks

None.

GFX_RESOURCE Enumeration

Specifies the different resource types in the library.

File

gfx_types_resource.h

C

```
typedef enum {
  GFX_RESOURCE_MEMORY_FLASH,
  GFX_RESOURCE_MEMORY_EXTERNAL,
```

```
  GFX_RESOURCE_MEMORY_FS,
  GFX_RESOURCE_MEMORY_RAM,
  GFX_RESOURCE_MEMORY_EDS_EPMP,
  GFX_RESOURCE_TYPE_MCHP_MBITMAP,
  GFX_RESOURCE_TYPE_JPEG,
  GFX_RESOURCE_TYPE_BINARY,
  GFX_RESOURCE_TYPE_FONT,
  GFX_RESOURCE_TYPE_PALETTE,
  GFX_RESOURCE_TYPE_BMP,
  GFX_RESOURCE_TYPE_GIF,
  GFX_RESOURCE_COMP_NONE,
  GFX_RESOURCE_COMP_RLE,
  GFX_RESOURCE_COMP_IPU,
  GFX_RESOURCE_MCHP_MBITMAP_FLASH_NONE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_MCHP_MBITMAP_FLASH_RLE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_RLE),
  GFX_RESOURCE_MCHP_MBITMAP_FLASH_IPU = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_IPU),
  GFX_RESOURCE_MCHP_MBITMAP_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_EXTERNAL
| GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_MCHP_MBITMAP_EXTERNAL_RLE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_EXTERNAL |
GFX_RESOURCE_COMP_RLE),
  GFX_RESOURCE_MCHP_MBITMAP_EXTERNAL_IPU = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_EXTERNAL |
GFX_RESOURCE_COMP_IPU),
  GFX_RESOURCE_MCHP_MBITMAP_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_EDS_EPMP
| GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_MCHP_MBITMAP_EDS_EPMP_RLE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_EDS_EPMP |
GFX_RESOURCE_COMP_RLE),
  GFX_RESOURCE_MCHP_MBITMAP_EDS_EPMP_IPU = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_EDS_EPMP |
GFX_RESOURCE_COMP_IPU),
  GFX_RESOURCE_JPEG_FLASH_NONE = (GFX_RESOURCE_TYPE_JPEG | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_JPEG_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_JPEG | GFX_RESOURCE_MEMORY_EXTERNAL |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_JPEG_FS_NONE = (GFX_RESOURCE_TYPE_JPEG | GFX_RESOURCE_MEMORY_FS | GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_JPEG_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_JPEG | GFX_RESOURCE_MEMORY_EDS_EPMP |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_BMP_FLASH_NONE = (GFX_RESOURCE_TYPE_BMP | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_BMP_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_BMP | GFX_RESOURCE_MEMORY_EXTERNAL |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_BMP_FS_NONE = (GFX_RESOURCE_TYPE_BMP | GFX_RESOURCE_MEMORY_FS | GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_BMP_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_BMP | GFX_RESOURCE_MEMORY_EDS_EPMP |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_GIF_FLASH_NONE = (GFX_RESOURCE_TYPE_GIF | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_GIF_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_GIF | GFX_RESOURCE_MEMORY_EXTERNAL |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_GIF_FS_NONE = (GFX_RESOURCE_TYPE_GIF | GFX_RESOURCE_MEMORY_FS | GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_GIF_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_GIF | GFX_RESOURCE_MEMORY_EDS_EPMP |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_BINARY_FLASH_NONE = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_BINARY_FLASH_IPU = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_IPU),
  GFX_RESOURCE_BINARY_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_EXTERNAL |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_BINARY_EXTERNAL_IPU = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_EXTERNAL |
GFX_RESOURCE_COMP_IPU),
  GFX_RESOURCE_BINARY_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_EDS_EPMP |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_BINARY_EDS_EPMP_IPU = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_EDS_EPMP |
GFX_RESOURCE_COMP_IPU),
  GFX_RESOURCE_FONT_FLASH_NONE = (GFX_RESOURCE_TYPE_FONT | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_FONT_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_FONT | GFX_RESOURCE_MEMORY_EXTERNAL |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_FONT_RAM_NONE = (GFX_RESOURCE_TYPE_FONT | GFX_RESOURCE_MEMORY_RAM | GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_FONT_EDS_NONE = (GFX_RESOURCE_TYPE_FONT | GFX_RESOURCE_MEMORY_EDS_EPMP |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_PALETTE_FLASH_NONE = (GFX_RESOURCE_TYPE_PALETTE | GFX_RESOURCE_MEMORY_FLASH |
GFX_RESOURCE_COMP_NONE),
  GFX_RESOURCE_PALETTE_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_PALETTE | GFX_RESOURCE_MEMORY_EXTERNAL |
GFX_RESOURCE_COMP_NONE),
```

```
  GFX_RESOURCE_PALETTE_RAM_NONE = (GFX_RESOURCE_TYPE_PALETTE | GFX_RESOURCE_MEMORY_RAM |
GFX_RESOURCE_COMP_NONE)
} GFX_RESOURCE;
```

Members

Members	Description
GFX_RESOURCE_MEMORY_FLASH	A location type: Internal flash memory.
GFX_RESOURCE_MEMORY_EXTERNAL	A location type: External memory.
GFX_RESOURCE_MEMORY_FS	A location type: External memory vis File System.
GFX_RESOURCE_MEMORY_RAM	A location type: Random Access Memory (RAM).
GFX_RESOURCE_MEMORY_EDS_EPMP	A location type: Resource is external and accessed through Enhanced Parallel Master Port (EPMP), memory size and base addresses are are set in the configuration.
GFX_RESOURCE_TYPE_MCHP_MBITMAP	A data type: Microchip bitmap type of resource.
GFX_RESOURCE_TYPE_JPEG	A data type: Image of type JPEG.
GFX_RESOURCE_TYPE_BINARY	A data type: Binary data type.
GFX_RESOURCE_TYPE_FONT	A data type: Font type of data
GFX_RESOURCE_TYPE_PALETTE	A data type: Palette type of data
GFX_RESOURCE_TYPE_BMP	A data type: Image of type BMP.
GFX_RESOURCE_COMP_NONE	A compression type: Data resource has no compression.
GFX_RESOURCE_COMP_RLE	A compression type: Data resource is compressed with RLE.
GFX_RESOURCE_COMP_IPU	A compression type: Data resource compressed with DEFLATE algorithm (for IPU).
GFX_RESOURCE_MCHP_MBITMAP_FLASH_NONE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_FLASH | GFX_RESOURCE_COMP_NONE)	These are common resource combinations used by the graphics library Microchip bitmap image type, located in flash and no compression.
GFX_RESOURCE_MCHP_MBITMAP_FLASH_RLE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_FLASH | GFX_RESOURCE_COMP_RLE)	Microchip bitmap image type, located in flash and RLE compressed.
GFX_RESOURCE_MCHP_MBITMAP_FLASH_IPU = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_FLASH | GFX_RESOURCE_COMP_IPU)	Microchip bitmap image type, located in flash and compressed for IPU
GFX_RESOURCE_MCHP_MBITMAP_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_NONE)	Microchip bitmap image type, located in external memory and no compression.
GFX_RESOURCE_MCHP_MBITMAP_EXTERNAL_RLE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_RLE)	Microchip bitmap image type, located in external memory and RLE compressed.
GFX_RESOURCE_MCHP_MBITMAP_EXTERNAL_IPU = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_IPU)	Microchip bitmap image type, located in external memory and compressed for IPU.
GFX_RESOURCE_MCHP_MBITMAP_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_EDS_EPMP | GFX_RESOURCE_COMP_NONE)	Microchip bitmap image type, located in EDS memory and no compression.
GFX_RESOURCE_MCHP_MBITMAP_EDS_EPMP_RLE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_EDS_EPMP | GFX_RESOURCE_COMP_RLE)	Microchip bitmap image type, located in EDS memory and RLE compressed.
GFX_RESOURCE_MCHP_MBITMAP_EDS_EPMP_IPU = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_EDS_EPMP | GFX_RESOURCE_COMP_IPU)	Microchip bitmap image type, located in EDS memory and compressed for IPU.
GFX_RESOURCE_JPEG_FLASH_NONE = (GFX_RESOURCE_TYPE_JPEG | GFX_RESOURCE_MEMORY_FLASH | GFX_RESOURCE_COMP_NONE)	JPEG image type, located in flash and no compression.

GFX_RESOURCE_JPEG_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_JPEG | GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_NONE)	JPEG image type, located in external memory no compression.
GFX_RESOURCE_JPEG_FS_NONE = (GFX_RESOURCE_TYPE_JPEG | GFX_RESOURCE_MEMORY_FS | GFX_RESOURCE_COMP_NONE)	JPEG image type, located in file system no compression.
GFX_RESOURCE_JPEG_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_JPEG | GFX_RESOURCE_MEMORY_EDS_EPMP | GFX_RESOURCE_COMP_NONE)	JPEG image type, located in EDS memory and no compression.
GFX_RESOURCE_BMP_FLASH_NONE = (GFX_RESOURCE_TYPE_BMP | GFX_RESOURCE_MEMORY_FLASH | GFX_RESOURCE_COMP_NONE)	BMP image type, located in flash and no compression.
GFX_RESOURCE_BMP_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_BMP | GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_NONE)	BMP image type, located in external memory no compression.
GFX_RESOURCE_BMP_FS_NONE = (GFX_RESOURCE_TYPE_BMP | GFX_RESOURCE_MEMORY_FS | GFX_RESOURCE_COMP_NONE)	BMP image type, located in file system no compression.
GFX_RESOURCE_BMP_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_BMP | GFX_RESOURCE_MEMORY_EDS_EPMP | GFX_RESOURCE_COMP_NONE)	BMP image type, located in EDS memory and no compression.
GFX_RESOURCE_GIF_FLASH_NONE = (GFX_RESOURCE_TYPE_GIF | GFX_RESOURCE_MEMORY_FLASH | GFX_RESOURCE_COMP_NONE)	GIF image type, located in flash and no compression.
GFX_RESOURCE_GIF_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_GIF | GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_NONE)	GIF image type, located in external memory no compression.
GFX_RESOURCE_GIF_FS_NONE = (GFX_RESOURCE_TYPE_GIF | GFX_RESOURCE_MEMORY_FS | GFX_RESOURCE_COMP_NONE)	GIF image type, located in file system no compression.
GFX_RESOURCE_GIF_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_GIF | GFX_RESOURCE_MEMORY_EDS_EPMP | GFX_RESOURCE_COMP_NONE)	GIF image type, located in EDS memory and no compression.
GFX_RESOURCE_BINARY_FLASH_NONE = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_FLASH | GFX_RESOURCE_COMP_NONE)	Binary image type, located in flash and no compression.
GFX_RESOURCE_BINARY_FLASH_IPU = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_FLASH | GFX_RESOURCE_COMP_IPU)	Binary image type, located in flash memory and compressed for IPU.
GFX_RESOURCE_BINARY_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_NONE)	Binary image type, located in external memory and no compression.
GFX_RESOURCE_BINARY_EXTERNAL_IPU = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_IPU)	Binary image type, located in external memory and compressed for IPU.
GFX_RESOURCE_BINARY_EDS_EPMP_NONE = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_EDS_EPMP | GFX_RESOURCE_COMP_NONE)	Binary image type, located in EDS memory and no compression.
GFX_RESOURCE_BINARY_EDS_EPMP_IPU = (GFX_RESOURCE_TYPE_BINARY | GFX_RESOURCE_MEMORY_EDS_EPMP | GFX_RESOURCE_COMP_IPU)	Binary image type, located in EDS memory and compressed for IPU.

GFX_RESOURCE_FONT_FLASH_NONE = (GFX_RESOURCE_TYPE_FONT | GFX_RESOURCE_MEMORY_FLASH | GFX_RESOURCE_COMP_NONE)	Font type, located in flash and no compression.
GFX_RESOURCE_FONT_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_FONT | GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_NONE)	Font type, located in external memory and no compression.
GFX_RESOURCE_FONT_RAM_NONE = (GFX_RESOURCE_TYPE_FONT | GFX_RESOURCE_MEMORY_RAM | GFX_RESOURCE_COMP_NONE)	Font type, located in RAM and no compression.
GFX_RESOURCE_FONT_EDS_NONE = (GFX_RESOURCE_TYPE_FONT | GFX_RESOURCE_MEMORY_EDS_EPMP | GFX_RESOURCE_COMP_NONE)	Font type, located in EDS memory and no compression.
GFX_RESOURCE_PALETTE_FLASH_NONE = (GFX_RESOURCE_TYPE_PALETTE | GFX_RESOURCE_MEMORY_FLASH | GFX_RESOURCE_COMP_NONE)	Palette type, located in flash and no compression.
GFX_RESOURCE_PALETTE_EXTERNAL_NONE = (GFX_RESOURCE_TYPE_PALETTE | GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_NONE)	Palette type, located in external memory and no compression.
GFX_RESOURCE_PALETTE_RAM_NONE = (GFX_RESOURCE_TYPE_PALETTE | GFX_RESOURCE_MEMORY_RAM | GFX_RESOURCE_COMP_NONE)	Palette type, located in RAM and no compression.

Description

Typedef: GFX_RESOURCE

This enumeration lists the different types of resources, how they are accessed and how the data of the resource will be interpreted when rendered.

A resource in the library has three major characteristics:

1. Location or source of the resource - The following are the recognized sources of resources
- internal flash memory type
- internal RAM type
- external memory type
- external EDS memory type
- - external via File System Service
2. Data types of the resource - The following are the supported types of resources:
- image
- font
- palette
- binary data
3. Data compression type of the resource - the following are the supported compression of data:
- RLE - Run length encoded compression. This type of compression is only available for 8 and 4 bpp bitmaps.
- IPU - Compressed data is encoded using the DEFLATE algorithm with fixed Huffman codes; dynamic Huffman codes are not supported.

The first four types indicates the location of the resource. The next five types indicates which kind of resource and the next 3 types indicates if the resource data is compressed or not.

By combining these three groups, a resource can be described fully and accessed appropriately in the library. For example: By combining the location, type and compression into one type, the library can pass the resource type parameter when rendering the resource. GFX_RESOURCE_MCHP_MBITMAP_EXTERNAL_RLE = (GFX_RESOURCE_TYPE_MCHP_MBITMAP | GFX_RESOURCE_MEMORY_EXTERNAL | GFX_RESOURCE_COMP_RLE)

Each type will determine how the library will access the resource when rendering.

Remarks

None.

GFX_RESOURCE_BINARY Structure

Defines the structure used for the binary type resource.

File

gfx_types_resource.h

C

```
typedef struct {
  union {
    uint32_t extAddress;
    uint8_gfx_image_prog * progByteAddress;
    uint16_gfx_image_prog * progWordAddress;
    const char * constAddress;
    char * ramAddress;
    __eds__ char * edsAddress;
  } location;
  uint32_t size;
  uint32_t param1;
  uint32_t param2;
} GFX_RESOURCE_BINARY;
```

Members

Members	Description
union { uint32_t extAddress; uint8_gfx_image_prog * progByteAddress; uint16_gfx_image_prog * progWordAddress; const char * constAddress; char * ramAddress; __eds__ char * edsAddress; } location;	This defines the location of the binary resource in memory. Depending on the type, the address location is interpreted accordingly.
uint32_t extAddress;	An external address.
uint8_gfx_image_prog * progByteAddress;	An 8-bit addresses in the program section.
uint16_gfx_image_prog * progWordAddress;	A 16-bit addresses in the program section.
const char * constAddress;	An addresses in constant space in flash.
char * ramAddress;	for addresses in RAM.
__eds__ char * edsAddress;	for addresses in EDS.
uint32_t size;	The size of the binary data in bytes.
uint32_t param1;	Parameters used for the GFX_RESOURCE. Depending on the GFX_RESOURCE type definition of param1 can change. For IPU and RLE compressed images, param1 indicates the compressed size of the image.
uint32_t param2;	Parameters used for the GFX_RESOURCE. Depending on the GFX_RESOURCE type

Description

Typedef: GFX_RESOURCE_BINARY

Defines the structure used for the binary type resource.

Remarks

None.

GFX_RESOURCE_FONT Structure

Defines the structure used for the font type resource.

File

gfx_types_resource.h

C

```
typedef struct {
  union {
    uint32_t extAddress;
    GFX_FONT_SPACE char * progByteAddress;
    char * ramAddress;
    __eds__ char * edsAddress;
  } location;
  GFX_FONT_HEADER header;
} GFX_RESOURCE_FONT;
```

Members

Members	Description
union { uint32_t extAddress; GFX_FONT_SPACE char * progByteAddress; char * ramAddress; __eds__ char * edsAddress; } location;	This defines the location of the font resource in memory. Depending on the type, the address location is interpreted accordingly.
uint32_t extAddress;	An external address.
GFX_FONT_SPACE char * progByteAddress;	An 8-bit addresses in the program section.
char * ramAddress;	An addresses in RAM.
__eds__ char * edsAddress;	An addresses in EDS.
GFX_FONT_HEADER header;	The header that describes the font resource.

Description

Typedef: GFX_RESOURCE_FONT

Defines the structure used for the font type resource.

Remarks

None.

GFX_RESOURCE_HDR Structure

Defines the structure used for the resource types.

File

gfx_types_resource.h

C

```
typedef struct {
  GFX_RESOURCE type;
  uint16_t ID;
  union {
    GFX_RESOURCE_IMAGE image;
    GFX_RESOURCE_FONT font;
    GFX_RESOURCE_BINARY binary;
    GFX_RESOURCE_PALETTE palette;
  } resource;
} GFX_RESOURCE_HDR;
```

Members

Members	Description
GFX_RESOURCE type;	Graphics resource type, determines the type and location of data
uint16_t ID;	memory ID, user defined value to differentiate between graphics resources of the same type When using EDS_EPMP the following ID values are reserved and used by the Microchip display driver 0 - reserved (do not use) 1 - reserved for base address of EPMP CS1 2 - reserved for base address of EPMP CS2
union { GFX_RESOURCE_IMAGE image; GFX_RESOURCE_FONT font; GFX_RESOURCE_BINARY binary; GFX_RESOURCE_PALETTE palette; } resource;	This defines the type of the resource. Depending on the type, the resource is accessed and rendered accordingly.
GFX_RESOURCE_IMAGE image;	Resource is an image.
GFX_RESOURCE_FONT font;	Resource is font.
GFX_RESOURCE_BINARY binary;	Resource is binary.
GFX_RESOURCE_PALETTE palette;	Resource is palette.

Description

Typedef: GFX_RESOURCE_HDR

Defines the common structure used for all the graphics resources.

Remarks

None.

GFX_RESOURCE_IMAGE Structure

Defines the structure used for the image type resource.

File

gfx_types_resource.h

C

```
typedef struct {
  union {
    uint32_t extAddress;
    uint8_gfx_image_prog * progByteAddress;
    uint16_gfx_image_prog * progWordAddress;
    const char * constAddress;
    char * ramAddress;
    __eds__ char * edsAddress;
  } location;
  uint16_t width;
  uint16_t height;
  union {
    uint32_t compressedSize;
    uint32_t reserved;
  } parameter1;
  union {
    uint32_t rawSize;
    uint32_t reserved;
  } parameter2;
  uint8_t colorDepth;
  uint8_t type;
  uint16_t paletteID;
} GFX_RESOURCE_IMAGE;
```

Members

Members	Description
union { uint32_t extAddress; uint8_gfx_image_prog * progByteAddress; uint16_gfx_image_prog * progWordAddress; const char * constAddress; char * ramAddress; __eds__ char * edsAddress; } location;	This defines the location of the image resource in memory. Depending on the type, the address location is interpreted accordingly.
uint32_t extAddress;	An external address.
uint8_gfx_image_prog * progByteAddress;	An 8-bit addresses in the program section.
uint16_gfx_image_prog * progWordAddress;	A 16-bit addresses in the program section.
const char * constAddress;	An addresses in constant space in flash.
char * ramAddress;	An addresses in RAM.
__eds__ char * edsAddress;	An addresses in EDS.
uint16_t width;	The width of the image.
uint16_t height;	The height of the image.
union { uint32_t compressedSize; uint32_t reserved; } parameter1;	A generic parameter 1 that changes in usage depending on the type of the resource.
uint32_t compressedSize;	Parameters used for the GFX_RESOURCE. Depending on the GFX_RESOURCE type definition of param1 can change. For IPU and RLE compressed images, param1 indicates the compressed size of the image.
union { uint32_t rawSize; uint32_t reserved; } parameter2;	A generic parameter 2 that changes in usage depending on the type of the resource.

uint32_t rawSize;	Parameters used for the GFX_RESOURCE. Depending on the GFX_RESOURCE type definition of param2 can change. For IPU and RLE compressed images, param2 indicates the uncompressed size of the image.
uint8_t colorDepth;	The color depth of the image.
uint8_t type;	The type of image, information on how to render the image 0x00 - no compression, palette is present for color depth = 8, 4 and 1 BPP 0x10 - palette is provided as a separate object (see PALETTE_HEADER) for color depth = 8, 4, and 1 BPP, ID to the palette is embedded color depth = 8, 4, and 1 BPP, in the bitmap. 0xYY - reserved
uint16_t paletteID;	The palette ID, if type == MCHP_BITMAP_PALETTE_STR (0x10), this represents the unique ID of the palette being used

Description

Typedef: GFX_RESOURCE_IMAGE

Defines the structure used for the image type resource.

Remarks

None.

GFX_RESOURCE_PALETTE Structure

Defines the structure used for the palette type resource.

File

gfx_types_resource.h

C

```
typedef struct {
  union {
    uint32_t extAddress;
    const GFX_PALETTE_ENTRY * constByteAddress;
    GFX_PALETTE_ENTRY * ramAddress;
  } location;
  uint16_t numberOfEntries;
  uint16_t paletteID;
} GFX_RESOURCE_PALETTE;
```

Members

Members	Description
union { uint32_t extAddress; const GFX_PALETTE_ENTRY * constByteAddress; GFX_PALETTE_ENTRY * ramAddress; } location;	This defines the location of the palette resource in memory. Depending on the type, the address location is interpreted accordingly.
uint32_t extAddress;	An external address.
const GFX_PALETTE_ENTRY * constByteAddress;	An addresses in constant space in flash.
GFX_PALETTE_ENTRY * ramAddress;	An addresses in RAM.
uint16_t numberOfEntries;	The number of color entries in the palette resource.
uint16_t paletteID;	Unique ID of the palette that will match the ID in the GFX_RESOURCE_HDR if the image has palette removed and supplied as a separate object.

Description

Typedef: GFX_RESOURCE_PALETTE

Defines the structure used for the palette type resource.

Remarks

None.

GFX_STATUS Enumeration

Rendering status.

File

gfx_types_macros.h

C

```
typedef enum {
  GFX_STATUS_FAILURE,
  GFX_STATUS_SUCCESS
} GFX_STATUS;
```

Members

Members	Description
GFX_STATUS_FAILURE	Rendering is not yet performed, or has started but not yet finished.
GFX_STATUS_SUCCESS	Rendering has been performed.

Description

Typedef: GFX_STATUS

When rendering into the frame buffer, it is sometimes necessary to return the status of the rendering. This is especially true when using hardware to render primitive shapes or when a FIFO is used to render primitive shapes.

Remarks

None.

GFX_STATUS_BIT Enumeration

Additional rendering status.

File

gfx_types_macros.h

C

```
typedef enum {
  GFX_STATUS_FAILURE_BIT,
  GFX_STATUS_SUCCESS_BIT,
  GFX_STATUS_ERROR_BIT,
  GFX_STATUS_BUSY_BIT,
  GFX_STATUS_READY_BIT
} GFX_STATUS_BIT;
```

Members

Members	Description
GFX_STATUS_FAILURE_BIT	Rendering failed. Depending on the driver used, hardware may or may not recover.
GFX_STATUS_SUCCESS_BIT	Rendering is successful.
GFX_STATUS_ERROR_BIT	Rendering resulted in an error and cannot recover.
GFX_STATUS_BUSY_BIT	Rendering cannot proceed due to a common resource is busy.
GFX_STATUS_READY_BIT	Rendering can proceed.

Description

Typedef: GFX_STATUS_BIT

The following rendering status types are available for a detailed description of the status of rendering.

Remarks

None.

int16_gfx_image_prog Type

File

gfx_types_image.h

C

```
typedef int16_prog_pack int16_gfx_image_prog;
```

Description

This is type int16_gfx_image_prog.

int32_gfx_image_prog Type

File

gfx_types_image.h

C

```
typedef int32_prog_pack int32_gfx_image_prog;
```

Description

This is type int32_gfx_image_prog.

int8_gfx_image_prog Type

File

gfx_types_image.h

C

```
typedef int8_prog_pack int8_gfx_image_prog;
```

Description

This is type int8_gfx_image_prog.

uint16_gfx_image_prog Type

File

gfx_types_image.h

C

```
typedef uint16_prog_pack uint16_gfx_image_prog;
```

Description

This is type uint16_gfx_image_prog.

uint32_gfx_image_prog Type

File

gfx_types_image.h

C

```
typedef uint32_prog_pack uint32_gfx_image_prog;
```

Description

This is type uint32_gfx_image_prog.

uint8_gfx_image_prog Type

File

gfx_types_image.h

C

```
typedef uint8_prog_pack uint8_gfx_image_prog;
```

Description

GFX Image data Types

GFX_ALIGNMENT Enumeration

Summary of the different text alignment supported in the library.

File

gfx_types_macros.h

C

```
typedef enum {
  GFX_ALIGN_LEFT,
  GFX_ALIGN_HCENTER,
  GFX_ALIGN_RIGHT,
  GFX_ALIGN_TOP,
  GFX_ALIGN_VCENTER,
  GFX_ALIGN_BOTTOM,
  GFX_ALIGN_CENTER
} GFX_ALIGNMENT;
```

Members

Members	Description
GFX_ALIGN_LEFT	left aligned text
GFX_ALIGN_HCENTER	horizontal center aligned text
GFX_ALIGN_RIGHT	right aligned text
GFX_ALIGN_TOP	top aligned text
GFX_ALIGN_VCENTER	vertical center aligned text
GFX_ALIGN_BOTTOM	bottom aligned text
GFX_ALIGN_CENTER	vertical & horizontal center aligned text

Description

Typedef: GFX_ALIGNMENT

This lists all the different text alignment supported in the library. The text alignment is used in GFX_TextStringBoxDraw().

Remarks

None.

GFX_BACKGROUND Structure

Background information structure.

File

gfx_types_macros.h

C

```
typedef struct {
  uint16_t left;
  uint16_t top;
  GFX_BACKGROUND_TYPE type;
  GFX_RESOURCE_HDR * pImage;
  GFX_COLOR color;
} GFX_BACKGROUND;
```

Members

Members	Description
uint16_t left;	Horizontal starting position of the background.
uint16_t top;	Horizontal starting position of the background.
GFX_BACKGROUND_TYPE type;	Specifies the type of background to use.
GFX_RESOURCE_HDR * pImage;	Pointer to the background image used. Set this to NULL if not used.
GFX_COLOR color;	If the background image is NULL, background is assumed to be this color.

Description

Typedef: GFX_BACKGROUND

Structure describing the background information. Useful in refreshing the whole screen or an area of the screen. The background information, when used, can be one of the enumerated types in GFX_BACKGROUND_TYPE.

- pBackGroundImage - pointer to a GFX_RESOURCE_HEADER with the type GFX_RESOURCE_TYPE_MCHP_MBITMAP.
- backGroundColor - value of the background color used when pBackgroundImage is NULL.

The pBackGroundImage has the high priority and will assume an image is used as a background when the pointer is initialized. If this is NULL the backgroundColor is assumed to be the background.

Remarks

None.

GFX_BACKGROUND_TYPE Enumeration

Specifies the different background fill types.

File

gfx_types_macros.h

C

```
typedef enum {
  GFX_BACKGROUND_NONE,
  GFX_BACKGROUND_COLOR,
  GFX_BACKGROUND_IMAGE,
  GFX_BACKGROUND_DISPLAY_BUFFER
} GFX_BACKGROUND_TYPE;
```

Members

Members	Description
GFX_BACKGROUND_NONE	No background information set.
GFX_BACKGROUND_COLOR	Background is set to a color.
GFX_BACKGROUND_IMAGE	Background is set to an image.
GFX_BACKGROUND_DISPLAY_BUFFER	Background is set to the current content of the display buffer. This requires support of GFX_PixelArrayGet().

Description

Typedef: GFX_FILL_STYLE

Defines the types of background information. Knowing the background information allows easier refresh of an area on the display buffer.

Remarks

None.

GFX_DOUBLE_BUFFERING_MODE Structure

Structure used for double buffering management.

File

gfx_types_macros.h

C

```
typedef struct {
  GFX_FEATURE_STATUS gfxDoubleBufferFeature;
  GFX_FEATURE_STATUS gfxDoubleBufferFeatureRecentStatus;
  GFX_FEATURE_STATUS gfxDoubleBufferRequestSync;
  GFX_FEATURE_STATUS gfxDoubleBufferFullSync;
  GFX_RECTANGULAR_AREA gfxDoubleBufferAreas[GFX_MAX_INVALIDATE_AREAS];
  uint16_t gfxUnsyncedAreaCount;
} GFX_DOUBLE_BUFFERING_MODE;
```

Members

Members	Description
GFX_FEATURE_STATUS gfxDoubleBufferFeature;	The status of the double buffering feature.
GFX_FEATURE_STATUS gfxDoubleBufferFeatureRecentStatus;	The status of the double buffering feature.

GFX_FEATURE_STATUS gfxDoubleBufferRequestSync;	The status of the request for synchronization.
GFX_FEATURE_STATUS gfxDoubleBufferFullSync;	The status of the full synchronization request.
GFX_RECTANGULAR_AREA gfxDoubleBufferAreas[GFX_MAX_INVALIDATE_AREAS];	The array of rectangular areas that needs synchronization.
uint16_t gfxUnsyncedAreaCount;	The current count of unsynchronized areas.

Description

Typedef: GFX_RECTANGULAR_AREA

Structure describing the double buffering states of the frame and draw buffer. This saves the current states of the feature for management by the graphics library. Double buffering is only available to selected configurations.

Remarks

None.

GFX_FEATURE_STATUS Enumeration

States of a feature that can be enabled/disabled at run time.

File

gfx_types_macros.h

C

```
typedef enum {
  GFX_FEATURE_PAUSED,
  GFX_FEATURE_ENABLED,
  GFX_FEATURE_DISABLED
} GFX_FEATURE_STATUS;
```

Members

Members	Description
GFX_FEATURE_PAUSED	The feature is paused.
GFX_FEATURE_ENABLED	The feature is enabled.
GFX_FEATURE_DISABLED	The feature is disabled.

Description

Typedef: GFX_FEATURE_STATUS

Defines the states of a Graphics Library feature that can be enabled/disabled at run time.

Remarks

None.

GFX_FILL_STYLE Enumeration

Specifies the available fill styles.

File

gfx_types_macros.h

C

```
typedef enum {
  GFX_FILL_STYLE_NONE,
  GFX_FILL_STYLE_COLOR,
  GFX_FILL_STYLE_ALPHA_COLOR,
  GFX_FILL_STYLE_GRADIENT_DOWN,
  GFX_FILL_STYLE_GRADIENT_UP,
  GFX_FILL_STYLE_GRADIENT_RIGHT,
  GFX_FILL_STYLE_GRADIENT_LEFT,
  GFX_FILL_STYLE_GRADIENT_DOUBLE_VER,
  GFX_FILL_STYLE_GRADIENT_DOUBLE_HOR
} GFX_FILL_STYLE;
```

Members

Members	Description
GFX_FILL_STYLE_NONE	no fill
GFX_FILL_STYLE_COLOR	color fill
GFX_FILL_STYLE_ALPHA_COLOR	color fill with alpha blending
GFX_FILL_STYLE_GRADIENT_DOWN	gradient, vertical-down direction
GFX_FILL_STYLE_GRADIENT_UP	gradient, vertical-up direction
GFX_FILL_STYLE_GRADIENT_RIGHT	gradient, horizontal-right direction
GFX_FILL_STYLE_GRADIENT_LEFT	gradient, horizontal-left direction
GFX_FILL_STYLE_GRADIENT_DOUBLE_VER	gradient, vertical-up/down direction
GFX_FILL_STYLE_GRADIENT_DOUBLE_HOR	gradient, horizontal-left/right direction

Description

Typedef: GFX_FILL_STYLE

This enumeration specifies the available fill styles used in the library.

Remarks

None.

GFX_FONT_ANTIALIAS_TYPE Enumeration

Summary of the transparency types in text anti-aliasing.

File

gfx_types_macros.h

C

```
typedef enum {
  GFX_FONT_ANTIALIAS_OPAQUE,
  GFX_FONT_ANTIALIAS_TRANSLUCENT
} GFX_FONT_ANTIALIAS_TYPE;
```

Members

Members	Description
GFX_FONT_ANTIALIAS_OPAQUE	Mid colors are calculated only once while rendering each character. This is ideal for rendering text over a constant background.
GFX_FONT_ANTIALIAS_TRANSLUCENT	Mid colors are calculated for every necessary pixel. This feature is useful when rendering text over an image or when the background is not one flat color.

Description

Typedef: GFX_FONT_ANTIALIAS_TYPE

Transparency type when rendering characters with anti-aliasing.

Remarks

None.

GFX_FONT_SPACE Macro

File

gfx_types_font.h

C

```
#define GFX_FONT_SPACE const
```

Description

Font space section. The fonts can be located in psv (constant) or program space in PIC24/dsPIC MCUs. This define allows for switching of the pointer type used to access the font structure in memory See: GraphicsConfig.h for the application define.

GFX_UXCHAR Macro

File

gfx_types_font.h

C

```
#define GFX_UXCHAR uint8_t
```

Description

This is macro GFX_UXCHAR.

GFX_XCHAR Macro

Configuration for the text character size.

File

gfx_types_font.h

C

```
#define GFX_XCHAR char
```

Description

GFX_CONFIG_FONT_CHAR_SIZE

This configuration sets the text character size used in the library.

To enable support for unicode fonts, GFX_CONFIG_FONT_CHAR_SIZE must be defined as 16 (size is 16 bits). For standard ASCII fonts, this can be defined as 8. This changes the GFX_XCHAR definition. See GFX_XCHAR for details.

Set in configuration	GFX_XCHAR	Description
#define GFX_CONFIG_FONT_CHAR_SIZE 16	#define GFX_XCHAR uint16_t	Use multi-byte characters (0-2^16 range).
#define GFX_CONFIG_FONT_CHAR_SIZE 8	#define GFX_XCHAR uint8_t	Use byte characters (0-255 range).

Remarks

None.

GRAPHICS_LIBRARY_VERSION Macro

File

gfx.h

C

```
#define GRAPHICS_LIBRARY_VERSION 0x0400
```

Description

////////////////// GRAPHICS_LIBRARY_VERSION ////////////////// MSB is version, LSB is subversion

GFX_COMP_MASK Macro

File

gfx_types_resource.h

C

```
#define GFX_COMP_MASK 0xF000
```

Description

This is macro GFX_COMP_MASK.

GFX_MEM_MASK Macro

File

gfx_types_resource.h

C

```
#define GFX_MEM_MASK 0x000F
```

Description

This is macro GFX_MEM_MASK.

GFX_TYPE_MASK Macro

File

gfx_types_resource.h

C

```
#define GFX_TYPE_MASK 0x0F00
```

Description

This is macro GFX_TYPE_MASK.

MCHP_BITMAP_NORMAL Macro

File

gfx_types_resource.h

C

```
#define MCHP_BITMAP_NORMAL 0x00                // no compression, palette is present
```

Description

no compression, palette is present for color depth = 8, 4 and 1 BPP

MCHP_BITMAP_PALETTE_STR Macro

File

gfx_types_resource.h

C

```
#define MCHP_BITMAP_PALETTE_STR 0x10                // palette is provided as a separate
```

Description

palette is provided as a separate object (see PALETTE_HEADER) for color depth = 8, 4, and 1 BPP, ID to the palette is embedded in the bitmap.

int16_prog Type

File

gfx_types_image.h

C

```
typedef __prog__ int16_t int16_prog;
```

Description

This is type int16_prog.

int16_prog_pack Type

File

gfx_types_image.h

C

```
typedef __pack_upper_byte int16_t int16_prog_pack;
```

Description

This is type int16_prog_pack.

int32_prog Type

File

gfx_types_image.h

C

```
typedef __prog__ int32_t int32_prog;
```

Description

This is type int32_prog.

int32_prog_pack Type

File

gfx_types_image.h

C

```
typedef __pack_upper_byte int32_t int32_prog_pack;
```

Description

This is type int32_prog_pack.

int8_prog Type

File

gfx_types_image.h

C

```
typedef __prog__ int8_t int8_prog;
```

Description

This is type int8_prog.

int8_prog_pack Type

File

gfx_types_image.h

C

```
typedef __pack_upper_byte int8_t int8_prog_pack;
```

Description

This is type int8_prog_pack.

uint16_prog Type

File

gfx_types_image.h

C

```
typedef __prog__ uint16_t uint16_prog;
```

Description

This is type uint16_prog.

uint16_prog_pack Type

File

gfx_types_image.h

C

```
typedef __pack_upper_byte uint16_t uint16_prog_pack;
```

Description

This is type uint16_prog_pack.

uint32_prog Type

File

gfx_types_image.h

C

```
typedef __prog__ uint32_t uint32_prog;
```

Description

This is type uint32_prog.

uint32_prog_pack Type

File

gfx_types_image.h

C

```
typedef __pack_upper_byte uint32_t uint32_prog_pack;
```

Description

This is type uint32_prog_pack.

uint8_prog Type

File

gfx_types_image.h

C

```
typedef __prog__ uint8_t uint8_prog;
```

Description

General program space data types

uint8_prog_pack Type

File

gfx_types_image.h

C

```
typedef __pack_upper_byte uint8_t uint8_prog_pack;
```

Description

This is type uint8_prog_pack.

GFX_CLIENT_OBJ Structure

GFX client object structure.

File

gfx.h

C

```
typedef struct {
  GFX_CLIENT_STATUS clientState;
  bool inUse;
  void * obj;
  SYS_MODULE_INDEX halIndex;
  GFX_HANDLE primitiveHandle;
  GFX_HANDLE golHandle;
} GFX_CLIENT_OBJ;
```

Members

Members	Description
GFX_CLIENT_STATUS clientState;	Client status
bool inUse;	Set to true if this object is in use
void * obj;	Object from which the client is derived
SYS_MODULE_INDEX halIndex;	Save the index of the driver

Description

GFX client object structure

GFX client object structure.

Remarks

None.

GFX_CLIENT_STATUS Enumeration

Enumerated data type that identifies the GFX Module Client Status.

File

gfx.h

C

```
typedef enum {
  GFX_CLIENT_STATUS_CLOSED,
  GFX_CLIENT_STATUS_READY
} GFX_CLIENT_STATUS;
```

Members

Members	Description
GFX_CLIENT_STATUS_CLOSED	Client is closed or the specified handle is invalid
GFX_CLIENT_STATUS_READY	Client is ready

Description

GFX Client Status

This enumeration defines the possible status of the GFX Module Client. It is returned by the () function.

Remarks

None.

GFX_OBJ Structure

Defines the object required for interaction with the GFX Library.

File

gfx.h

C

```
typedef struct _GFX_OBJ {
  SYS_STATUS status;
  SYS_MODULE_INDEX index;
  bool inUse;
  GFX_STATES state;
  bool IsExclusive;
  uint8_t numClients;
  GFX_OBJECT_TASK task;
  GFX_INIT * initData;
  GFX_HANDLE gfxHandle;
  SYS_MODULE_OBJ primitiveObject;
  SYS_MODULE_OBJ golObject;
} GFX_OBJ;
```

Members

Members	Description
SYS_STATUS status;	Required : The status of the module
SYS_MODULE_INDEX index;	Object Index
bool inUse;	Required : Flag to indicate in use
GFX_STATES state;	GFX machine state
bool IsExclusive;	Flag to indicate that PMP is used in exclusive access mode
uint8_t numClients;	Optional: number of clients possible with the software instance
GFX_OBJECT_TASK task;	State of the task

Description

GFX Library Instance Object

This defines the object required for interaction with the GFX Library. This object exists once per application instance of the library.

Remarks

None.

GFX_OBJECT_TASK Enumeration

Lists the different states that GFX task routine can have.

File

gfx.h

C

```
typedef enum {
  GFX_TASK_STATE_OPEN_MODULE,
  GFX_TASK_STATE_RUNNING
} GFX_OBJECT_TASK;
```

Members

Members	Description
GFX_TASK_STATE_OPEN_MODULE	open module for client

Description

GFX task states

This enumeration lists the different states that GFX task routine can have.

Remarks

None.

GFX_STATES Enumeration

Defines the various states that can be achieved by the GFX operation.

File

gfx.h

C

```
typedef enum {
  GFX_STATE_BUSY,
  GFX_STATE_INIT,
  GFX_STATE_INITIALIZED
} GFX_STATES;
```

Members

Members	Description
GFX_STATE_BUSY	Module state busy
GFX_STATE_INIT	Module state init

Description

GFX Machine States

This enumeration defines the various states that can be achieved by the HAL operation.

Remarks

None.

FileFeof Type

File

gfx_image_decoder.h

C

```
typedef bool (* FileFeof)(void *stream);
```

Description

This is type FileFeof.

FileRead Type

File

gfx_image_decoder.h

C

```
typedef size_t (* FileRead)(void *ptr, size_t size, size_t n, void *stream);
```

Description

Typedefs of the used File System APIs

FileSeek Type

File

gfx_image_decoder.h

C

```
typedef bool (* FileSeek)(void *stream, long offset, int whence);
```

Description

This is type FileSeek.

FileTell Type

File

gfx_image_decoder.h

C

```
typedef uint32_t (* FileTell)(void *fo);
```

Description

This is type FileTell.

IMG_FILE_FORMAT Enumeration

File

gfx_image_decoder.h

C

```
typedef enum _IMG_FILE_FORMAT {
  IMG_NONE = 0,
  IMG_BMP,
  IMG_JPEG,
  IMG_GIF
} IMG_FILE_FORMAT;
```

Members

Members	Description
IMG_BMP	Bitmap image
IMG_JPEG	JPEG image
IMG_GIF	GIF image

Description

IMG_ImageFileFormat specifies all the supported image file formats

IMG_FILE_SYSTEM_API Structure

File

gfx_image_decoder.h

C

```
typedef struct _IMG_FILE_SYSTEM_API {
  FileRead pFread;
  FileSeek pFseek;
  FileTell pFtell;
  FileFeof pFeof;
} IMG_FILE_SYSTEM_API;
```

Description

IMG_FileSystemAPI holds function pointers to the used File System APIs

IMG_LOOP_CALLBACK Type

File

gfx_image_decoder.h

C

```
typedef void (* IMG_LOOP_CALLBACK)(void);
```

Description

IMG_LoopCallback is a callback function which is called in every loop of the decoding cycle so that user application can do some maintenance activities

IMG_PIXEL_OUTPUT Type

File

gfx_image_decoder.h

C

```
typedef void (* IMG_PIXEL_OUTPUT)(IMG_PIXEL_XY_RGB_888 *pPix);
```

Description

IMG_PixelOutput is a callback function which receives the color information of the output pixel

IMG_PIXEL_XY_RGB_888 Structure

File

gfx_image_decoder.h

C

```
typedef struct _IMG_PIXEL_XY_RGB_888 {
  uint16_t X;
  uint16_t Y;
  uint8_t R;
  uint8_t G;
  uint8_t B;
  uint16_t POS;
} IMG_PIXEL_XY_RGB_888;
```

Description

IMG_PixelRgb holds the RGB information of a pixel in BYTES

__IMAGEDECODER_H__ Macro

File

gfx_image_decoder.h

C

```
#define __IMAGEDECODER_H__
```

Description

This is macro __IMAGEDECODER_H__.

_PutPixel Macro

File

gfx_image_decoder.h

C

```
#define _PutPixel(x, y) if(IMG_pPixelOutput != NULL) { IMG_PixelXYColor.X = x; IMG_PixelXYColor.Y = y;
IMG_pPixelOutput(&IMG_PixelXYColor); }
```

Description

This is macro _PutPixel.

ImageAbort Macro

File

gfx_image_decoder.h

C

```
#define ImageAbort IMG_blAbortImageDecoding = 1;
```

Side Effects

None

Returns

None

Description

This function sets the Image Decoder's Abort flag so that decoding aborts in the next decoding loop.

Example

```
void callback(void);
void main(void)
{
    IMG_FILE pImageFile;
    IMG_vInitialize();
    ImageLoopCallbackRegister(callback);
    pImageFile = IMG_FOPEN("Image.jpg", "r");
    if(pImageFile == NULL)
    {
              <- Error handling ->
    }
    IMG_bFullScreenDecode(pImageFile, IMG_JPEG, NULL, NULL);
    IMG_FCLOSE(pImageFile);
    while(1);
}

void callback(void)
{
    if(<- check for abort condition ->)
    {
        ImageAbort();
    }
}
```

Function

void ImageAbort(void)

ImageFullScreenDecode Macro

File

gfx_image_decoder.h

C

```
#define ImageFullScreenDecode(pImageFile, eImgFormat, pFileAPIs, pPixelOutput) \
        ImageDecode(pImageFile, eImgFormat, 0, 0, IMG_SCREEN_WIDTH, IMG_SCREEN_HEIGHT, (IMG_ALIGN_CENTER |
IMG_DOWN_SCALE), pFileAPIs, pPixelOutput)
```

Side Effects

None

Returns

Error code -> 0 means no error

Description

This function decodes and displays the image on the screen in fullscreen mode with center aligned and downscaled if required

Example

```
  void main(void)
{
    IMG_FILE pImageFile;
    IMG_vInitialize();
    pImageFile = IMG_FOPEN("Image.jpg", "r");
    if(pImageFile == NULL)
    {
              <- Error handling ->
    }
    IMG_bFullScreenDecode(pImageFile, IMG_JPEG, NULL, NULL);
    IMG_FCLOSE(pImageFile);
    while(1);
}
```

Function

uint8_t ImageFullScreenDecode(IMG_FILE *pImageFile, IMG_FILE_FORMAT eImgFormat, IMG_FILE_SYSTEM_API pFileAPIs, IMG_PIXEL_OUTPUT pPixelOutput)

IMG_ALIGN_CENTER Macro

File

gfx_image_decoder.h

C

```
#define IMG_ALIGN_CENTER 0x0001
```

Description

Flags

IMG_DECODE_ABORTED Macro

File

gfx_image_decoder.h

C

```
#define IMG_DECODE_ABORTED 0xFF
```

Description

This is macro IMG_DECODE_ABORTED.

IMG_DOWN_SCALE Macro

File

gfx_image_decoder.h

C

```
#define IMG_DOWN_SCALE 0x0002
```

Description

This is macro IMG_DOWN_SCALE.

IMG_FEOF Macro

File

gfx_image_decoder.h

C

```
#define IMG_FEOF IMG_pFileAPIs->pFeof
```

Description

This is macro IMG_FEOF.

IMG_FILE Macro

File

gfx_image_decoder.h

C

```
#define IMG_FILE void
```

Description

This is macro IMG_FILE.

IMG_FREAD Macro

File

gfx_image_decoder.h

C

```
#define IMG_FREAD IMG_pFileAPIs->pFread
```

Description

This is macro IMG_FREAD.

IMG_FSEEK Macro

File

gfx_image_decoder.h

C

```
#define IMG_FSEEK IMG_pFileAPIs->pFseek
```

Description

This is macro IMG_FSEEK.

IMG_FTELL Macro

File

gfx_image_decoder.h

C

```
#define IMG_FTELL IMG_pFileAPIs->pFtell
```

Description

This is macro IMG_FTELL.

IMG_SCREEN_HEIGHT Macro

File

gfx_image_decoder.h

C

```
#define IMG_SCREEN_HEIGHT (GetMaxY()+1)
```

Description

This is macro IMG_SCREEN_HEIGHT.

IMG_SCREEN_WIDTH Macro

File

gfx_image_decoder.h

C

```
#define IMG_SCREEN_WIDTH (GetMaxX()+1)
```

Description

The individual image decoders use these defines instead of directly using those provided in the Display driver file

IMG_vCheckAndAbort Macro

File

gfx_image_decoder.h

C

```
#define IMG_vCheckAndAbort \
        if(IMG_blAbortImageDecoding == 1)   \
        {                                   \
              IMG_blAbortImageDecoding = 0; \
              return IMG_DECODE_ABORTED;    \
        }
```

Description

This is macro IMG_vCheckAndAbort.

IMG_vLoopCallback Macro

File

gfx_image_decoder.h

C

```
#define IMG_vLoopCallback
```

Description

This is macro IMG_vLoopCallback.

IMG_vPutPixel Macro

File

gfx_image_decoder.h

C

```
#define IMG_vPutPixel(x, y) if(IMG_bDownScalingFactor <= 1)\
                                {                          \
                                    _PutPixel(IMG_wStartX + (x), IMG_wStartY + (y));\
                                }                          \
                                else if((x)%IMG_bDownScalingFactor == 0 && (y)%IMG_bDownScalingFactor ==
0)\
                                {                          \
                                    _PutPixel(IMG_wStartX + ((x)/IMG_bDownScalingFactor), IMG_wStartY +
((y)/IMG_bDownScalingFactor)); \
                                }
```

Description

This is macro IMG_vPutPixel.

GFX_PALETTE_ENTRY Union

Defines the union used for each entry in the palette table.

File

gfx_types_palette.h

C

```
typedef union {
  uint16_t value;
  struct {
    uint16_t r : 5;
    uint16_t g : 6;
    uint16_t b : 5;
  } color;
  struct {
    uint16_t reserved : 12;
    uint16_t luma : 4;
  } monochrome;
} GFX_PALETTE_ENTRY;
```

Members

Members	Description
uint16_t value;	a 16-bit value representing a color or palette entry
struct { uint16_t r : 5; uint16_t g : 6; uint16_t b : 5; } color;	color value in 5-6-5 RGB format
uint16_t r : 5;	represents the RED component
uint16_t g : 6;	represents the GREEN component
uint16_t b : 5;	represents the BLUE component
struct { uint16_t reserved : 12; uint16_t luma : 4; } monochrome;	monochrome LUMA value
uint16_t reserved : 12;	reserved, used as a filler
uint16_t luma : 4;	monochrome LUMA value

Description

Typedef: GFX_PALETTE_ENTRY

Defines the union used for each entry in the palette table specified by GFX_RESOURCE_PALETTE structure.

There are two types of palette:

- For TFT: color is defined as 5-6-5 RGB format where 5-bits for RED, 6-bits for GREEN and 5-bits for BLUE.
- For Monochrome: 4 bits are used to represent the luma.

Remarks

None.

GFX_TEXT_CURSOR_TYPE Enumeration

Enumeration defines types of text cursor supported.

File

gfx_types_macros.h

C

```
typedef enum {
  GFX_TEXT_CURSOR_TYPE_UNDERSCORE
} GFX_TEXT_CURSOR_TYPE;
```

Members

Members	Description
GFX_TEXT_CURSOR_TYPE_UNDERSCORE	Text cursor of type underscore

Description

Typedef: GFX_TEXT_CURSOR_TYPE

This enumeration defines types of cursor can be drawn.

Remarks

None.

IMG_LOOP_FRAMECOMPLETE Type

File

gfx_image_decoder.h

C

```
typedef bool (* IMG_LOOP_FRAMECOMPLETE)(uint16_t frame);
```

Description

IMG_FrameCompleteCallback is a callback function which is called at completion decoding current frame so that user application can do some activities

IMG_LOOP_IMAGECOMPLETE Type

File

gfx_image_decoder.h

C

```
typedef void (* IMG_LOOP_IMAGECOMPLETE)(void);
```

Description

IMG_FrameCompleteCallback is a callback function which is called at completion decoding current frame so that user application can do some activities

IMG_LOOP_IMAGEFAIL Type

File

gfx_image_decoder.h

C

```
typedef void (* IMG_LOOP_IMAGEFAIL)(void);
```

Description

IMG_FrameCompleteCallback is a callback function which is called at completion decoding current frame so that user application can do some activities

IMG_vFrameComplete Macro

File

gfx_image_decoder.h

C

```
#define IMG_vFrameComplete if (IMG_pFrameCompleteCallbackFn != NULL) { IMG_pImageExitRequested =
IMG_pFrameCompleteCallbackFn(++IMG_pCurrentFrame); }
```

Description

This is macro IMG_vFrameComplete.

IMG_vImageComplete Macro

File

gfx_image_decoder.h

C

```
#define IMG_vImageComplete if (IMG_pImageCompleteCallbackFn != NULL) { IMG_pImageCompleteCallbackFn(); }
```

Description

This is macro IMG_vImageComplete.

IMG_vImageFail Macro

File

gfx_image_decoder.h

C

```
#define IMG_vImageFail if (IMG_pImageFailCallbackFn != NULL) { IMG_pImageFailCallbackFn(); }
```

Description

This is macro IMG_vImageFail.

IMG_vPixelArray Macro

File

gfx_image_decoder.h

C

```
#define IMG_vPixelArray(pos) IMG_PixelXYColor.POS = pos;
```

Description

This is macro IMG_vPixelArray.

IMG_vSetColor Macro

File

gfx_image_decoder.h

C

```
#define IMG_vSetColor(r, g, b) IMG_PixelXYColor.R = r; IMG_PixelXYColor.G = g; IMG_PixelXYColor.B = b;
```

Description

This is macro IMG_vSetColor.

Files

Files

Name	Description
gfx.h	The header file joins all header files used in the graphics library and contains compile options and defaults.
gfx_colors.h	The header file contains default color definitions for each color depth.
gfx_common.h	This is file gfx_common.h.
gfx_colors_w3.h	The header file contains W3 color definitions.
gfx_colors_x11.h	The header file contains X11 color definitions.
gfx_gol.h	Graphics Object Layer of the Microchip Graphics Library.
gfx_gol_button.h	This is the header file for the button object of the GOL.
gfx_gol_check_box.h	This is the header file for the check box object of the GOL.

gfx_gol_custom_control.h	This is the header file for the custom control object of the GOL.
gfx_gol_digital_meter.h	This is the header file for the digital meter object of the GOL.
gfx_gol_edit_box.h	This is the header file for the edit box object of the GOL.
gfx_gol_group_box.h	This is the header file for the group box object of the GOL.
gfx_gol_list_box.h	This is the header file for the list box object of the GOL.
gfx_gol_meter.h	This is the header file for the meter object of the GOL.
gfx_gol_picture.h	This is the header file for the picture object of the GOL.
gfx_gol_progress_bar.h	This is the header file for the progress bar object of the GOL.
gfx_gol_radio_button.h	This is the header file for the radio button object of the GOL.
gfx_gol_scan_codes.h	This is the header file for the AT keyboard codes.
gfx_gol_scheme.h	Graphics Object Layer Scheme routines/macros of the Microchip Graphics Library.
gfx_gol_scroll_bar.h	This is the header file for the scroll bar object of the GOL.
gfx_gol_static_text.h	This is the header file for the static text object of the GOL.
gfx_gol_surface.h	This is the header file for the surface object of the GOL.
gfx_gol_text_entry.h	This is the header file for the text entry object of the GOL.
gfx_gol_window.h	This is the header file for the window object of the GOL.
gfx_image_decoder.h	This implements the image decoding of the primitive layer.
gfx_primitive.h	Primitive Layer of the Microchip Graphics Library.
gfx_types_font.h	This header file defines the types used in fonts for the Microchip Graphics Library.
gfx_types_image.h	This header file defines the types used in images for the Microchip Graphics Library.
gfx_types_macros.h	Data types used in Microchip Graphics Library.
gfx_types_palette.h	This header file defines the types used in palettes for the Microchip Graphics Library.
gfx_types_resource.h	This is the definition of the Graphics Resource Types used with the Microchip Graphics Library.
gfx_config_template.h	This header file template defines all the Graphics Library configurations available.

Description

This section lists the source and header files used by the library.

gfx.h

The header file joins all header files used in the graphics library and contains compile options and defaults.

Enumerations

	Name	Description
	GFX_CLIENT_STATUS	Enumerated data type that identifies the GFX Module Client Status.
	GFX_OBJECT_TASK	Lists the different states that GFX task routine can have.
	GFX_STATES	Defines the various states that can be achieved by the GFX operation.

Functions

	Name	Description
	GFX_Close	Closes a client instance of the GFX library.
	GFX_Deinitialize	Deinitializes the specified instance of the GFX Module.
	GFX_Initialize	Initialize the GFX Library.
	GFX_Open	Opens the specified GFX library instance and returns a handle to it.
	GFX_Status	Gets the current status of the GFX Module.
	GFX_Tasks	Maintains the GFX module state machine. It manages the GFX Module object list items and responds to GFX Module primitive events.

Macros

	Name	Description
	GRAPHICS_LIBRARY_VERSION	////////////////// GRAPHICS_LIBRARY_VERSION ////////////////////// MSB is version, LSB is subversion

Structures

	Name	Description
	_GFX_OBJ	Defines the object required for interaction with the GFX Library.

	GFX_CLIENT_OBJ	GFX client object structure.
	GFX_OBJ	Defines the object required for interaction with the GFX Library.

Description

Module for Microchip Graphics Library

This header file includes all the header files required to use the Microchip Graphics Library. Library features and options defined in the Graphics Library configurations will be included in each build.

File Name

gfx.h

Company

Microchip Technology Inc.

gfx_colors.h

The header file contains default color definitions for each color depth.

Macros

	Name	Description
	BLACK	Basic colors definitions. The basic colors defined in this section are defined for basic demo requirements. End application can define additional colors or override existing default colors. If overriding an existing default color, define the new color value before the #include of Graphics.h to avoid the compile time warning. When using palette, a different file 'PaletteColorDefines.h' has to be used instead.
	BLUE	This is macro BLUE.
	BRIGHTBLUE	This is macro BRIGHTBLUE.
	BRIGHTCYAN	This is macro BRIGHTCYAN.
	BRIGHTGREEN	This is macro BRIGHTGREEN.
	BRIGHTMAGENTA	This is macro BRIGHTMAGENTA.
	BRIGHTRED	This is macro BRIGHTRED.
	BRIGHTYELLOW	This is macro BRIGHTYELLOW.
	BROWN	This is macro BROWN.
	BURLYWOOD	This is macro BURLYWOOD.
	CYAN	This is macro CYAN.
	DARKGRAY	This is macro DARKGRAY.
	DARKORANGE	This is macro DARKORANGE.
	GOLD	This is macro GOLD.
	GRAY000	This is macro GRAY000.
	GRAY001	This is macro GRAY001.
	GRAY002	This is macro GRAY002.
	GRAY003	This is macro GRAY003.
	GRAY004	This is macro GRAY004.
	GRAY005	This is macro GRAY005.
	GRAY006	This is macro GRAY006.
	GRAY007	This is macro GRAY007.
	GRAY008	This is macro GRAY008.
	GRAY009	This is macro GRAY009.
	GRAY010	This is macro GRAY010.
	GRAY011	This is macro GRAY011.
	GRAY012	This is macro GRAY012.
	GRAY013	This is macro GRAY013.
	GRAY014	This is macro GRAY014.
	GRAY015	This is macro GRAY015.
	GRAY032	This is macro GRAY032.
	GRAY096	This is macro GRAY096.
	GRAY128	This is macro GRAY128.

	GRAY160	This is macro GRAY160.
	GRAY192	This is macro GRAY192.
	GRAY204	This is macro GRAY204.
	GRAY224	This is macro GRAY224.
	GRAY229	This is macro GRAY229.
	GRAY242	This is macro GRAY242.
	GREEN	This is macro GREEN.
	LIGHTBLUE	This is macro LIGHTBLUE.
	LIGHTCYAN	This is macro LIGHTCYAN.
	LIGHTGRAY	This is macro LIGHTGRAY.
	LIGHTGREEN	This is macro LIGHTGREEN.
	LIGHTMAGENTA	This is macro LIGHTMAGENTA.
	LIGHTORANGE	This is macro LIGHTORANGE.
	LIGHTRED	This is macro LIGHTRED.
	LIGHTYELLOW	This is macro LIGHTYELLOW.
	MAGENTA	This is macro MAGENTA.
	ORANGE	This is macro ORANGE.
	PERU	This is macro PERU.
	RED	This is macro RED.
	SADDLEBROWN	This is macro SADDLEBROWN.
	SIENNA	This is macro SIENNA.
	TAN	This is macro TAN.
	WHEAT	This is macro WHEAT.
	WHITE	This is macro WHITE.
	YELLOW	This is macro YELLOW.

Description

Module for Microchip Graphics Library

This header file provides default color definitions as well as color conversion macros. Application may or may not use this file. Application may replace this file with new color definitions that fits the application needs.

File Name

gfx_colors.h

Company

Microchip Technology Inc.

gfx_common.h

This is file gfx_common.h.

gfx_colors_w3.h

The header file contains W3 color definitions.

Macros

	Name	Description
	GFX_W3_ALICEBLUE	This is macro GFX_W3_ALICEBLUE.
	GFX_W3_ANTIQUEWHITE	This is macro GFX_W3_ANTIQUEWHITE.
	GFX_W3_AQUA	This is macro GFX_W3_AQUA.
	GFX_W3_AQUAMARINE	This is macro GFX_W3_AQUAMARINE.
	GFX_W3_AZURE	This is macro GFX_W3_AZURE.
	GFX_W3_BEIGE	This is macro GFX_W3_BEIGE.
	GFX_W3_BISQUE	This is macro GFX_W3_BISQUE.
	GFX_W3_BLACK	This is macro GFX_W3_BLACK.
	GFX_W3_BLANCHEDALMOND	This is macro GFX_W3_BLANCHEDALMOND.
	GFX_W3_BLUE	This is macro GFX_W3_BLUE.

	GFX_W3_BLUEVIOLET	This is macro GFX_W3_BLUEVIOLET.
	GFX_W3_BROWN	This is macro GFX_W3_BROWN.
	GFX_W3_BURLYWOOD	This is macro GFX_W3_BURLYWOOD.
	GFX_W3_CADETBLUE	This is macro GFX_W3_CADETBLUE.
	GFX_W3_CHARTREUSE	This is macro GFX_W3_CHARTREUSE.
	GFX_W3_CHOCOLATE	This is macro GFX_W3_CHOCOLATE.
	GFX_W3_CORAL	This is macro GFX_W3_CORAL.
	GFX_W3_CORNFLOWERBLUE	This is macro GFX_W3_CORNFLOWERBLUE.
	GFX_W3_CORNSILK	This is macro GFX_W3_CORNSILK.
	GFX_W3_CRIMSON	This is macro GFX_W3_CRIMSON.
	GFX_W3_CYAN	This is macro GFX_W3_CYAN.
	GFX_W3_DARKBLUE	This is macro GFX_W3_DARKBLUE.
	GFX_W3_DARKCYAN	This is macro GFX_W3_DARKCYAN.
	GFX_W3_darkgoldenrod	This is macro GFX_W3_darkgoldenrod.
	GFX_W3_DARKGRAY	This is macro GFX_W3_DARKGRAY.
	GFX_W3_DARKGREEN	This is macro GFX_W3_DARKGREEN.
	GFX_W3_DARKGREY	This is macro GFX_W3_DARKGREY.
	GFX_W3_DARKHAKI	This is macro GFX_W3_DARKHAKI.
	GFX_W3_DARKMAGENTA	This is macro GFX_W3_DARKMAGENTA.
	GFX_W3_DARKOLIVEGREEN	This is macro GFX_W3_DARKOLIVEGREEN.
	GFX_W3_DARKORANGE	This is macro GFX_W3_DARKORANGE.
	GFX_W3_DARKORCHID	This is macro GFX_W3_DARKORCHID.
	GFX_W3_DARKRED	This is macro GFX_W3_DARKRED.
	GFX_W3_DARKSALMON	This is macro GFX_W3_DARKSALMON.
	GFX_W3_DARKSEAGREEN	This is macro GFX_W3_DARKSEAGREEN.
	GFX_W3_DARKSLATEBLUE	This is macro GFX_W3_DARKSLATEBLUE.
	GFX_W3_DARKSLATEGRAY	This is macro GFX_W3_DARKSLATEGRAY.
	GFX_W3_DARKSLATEGREY	This is macro GFX_W3_DARKSLATEGREY.
	GFX_W3_DARKTURQUOISE	This is macro GFX_W3_DARKTURQUOISE.
	GFX_W3_DARKVIOLET	This is macro GFX_W3_DARKVIOLET.
	GFX_W3_DEEPPINK	This is macro GFX_W3_DEEPPINK.
	GFX_W3_DEEPSKYBLUE	This is macro GFX_W3_DEEPSKYBLUE.
	GFX_W3_DIMGRAY	This is macro GFX_W3_DIMGRAY.
	GFX_W3_DIMGREY	This is macro GFX_W3_DIMGREY.
	GFX_W3_DODGERBLUE	This is macro GFX_W3_DODGERBLUE.
	GFX_W3_FIREBRICK	This is macro GFX_W3_FIREBRICK.
	GFX_W3_FLORALWHITE	This is macro GFX_W3_FLORALWHITE.
	GFX_W3_FORESTGREEN	This is macro GFX_W3_FORESTGREEN.
	GFX_W3_FUCHSIA	This is macro GFX_W3_FUCHSIA.
	GFX_W3_GAINSBORO	This is macro GFX_W3_GAINSBORO.
	GFX_W3_GHOSTWHITE	This is macro GFX_W3_GHOSTWHITE.
	GFX_W3_GOLD	This is macro GFX_W3_GOLD.
	GFX_W3_GOLDENROD	This is macro GFX_W3_GOLDENROD.
	GFX_W3_GRAY	This is macro GFX_W3_GRAY.
	GFX_W3_GREEN	This is macro GFX_W3_GREEN.
	GFX_W3_GREENYELLOW	This is macro GFX_W3_GREENYELLOW.
	GFX_W3_GREY	This is macro GFX_W3_GREY.
	GFX_W3_HONEYDEW	This is macro GFX_W3_HONEYDEW.
	GFX_W3_HOTPINK	This is macro GFX_W3_HOTPINK.
	GFX_W3_INDIANRED	This is macro GFX_W3_INDIANRED.
	GFX_W3_INDIGO	This is macro GFX_W3_INDIGO.
	GFX_W3_IVORY	This is macro GFX_W3_IVORY.
	GFX_W3_LAVENDER	This is macro GFX_W3_LAVENDER.
	GFX_W3_LAVENDERBLUSH	This is macro GFX_W3_LAVENDERBLUSH.
	GFX_W3_LAWNGREEN	This is macro GFX_W3_LAWNGREEN.

	GFX_W3_LEMONCHIFFON	This is macro GFX_W3_LEMONCHIFFON.
	GFX_W3_LIGHTBLUE	This is macro GFX_W3_LIGHTBLUE.
	GFX_W3_LIGHTCORAL	This is macro GFX_W3_LIGHTCORAL.
	GFX_W3_LIGHTCYAN	This is macro GFX_W3_LIGHTCYAN.
	GFX_W3_LIGHTGOLDENRODYELLOW	This is macro GFX_W3_LIGHTGOLDENRODYELLOW.
	GFX_W3_LIGHTGRAY	This is macro GFX_W3_LIGHTGRAY.
	GFX_W3_LIGHTGREEN	This is macro GFX_W3_LIGHTGREEN.
	GFX_W3_LIGHTGREY	This is macro GFX_W3_LIGHTGREY.
	GFX_W3_LIGHTPINK	This is macro GFX_W3_LIGHTPINK.
	GFX_W3_LIGHTSALMON	This is macro GFX_W3_LIGHTSALMON.
	GFX_W3_LIGHTSKYBLUE	This is macro GFX_W3_LIGHTSKYBLUE.
	GFX_W3_LIGHTSLATEGRAY	This is macro GFX_W3_LIGHTSLATEGRAY.
	GFX_W3_LIGHTSLATEGREY	This is macro GFX_W3_LIGHTSLATEGREY.
	GFX_W3_LIGHTSTEELBLUE	This is macro GFX_W3_LIGHTSTEELBLUE.
	GFX_W3_LIGHTYELLOW	This is macro GFX_W3_LIGHTYELLOW.
	GFX_W3_LIGTHSEAGREEN	This is macro GFX_W3_LIGTHSEAGREEN.
	GFX_W3_LIME	This is macro GFX_W3_LIME.
	GFX_W3_LIMEGREEN	This is macro GFX_W3_LIMEGREEN.
	GFX_W3_LINEN	This is macro GFX_W3_LINEN.
	GFX_W3_MAGENTA	This is macro GFX_W3_MAGENTA.
	GFX_W3_MAROON	This is macro GFX_W3_MAROON.
	GFX_W3_MEDIUMAQUAMARINE	This is macro GFX_W3_MEDIUMAQUAMARINE.
	GFX_W3_MEDIUMBLUE	This is macro GFX_W3_MEDIUMBLUE.
	GFX_W3_MEDIUMORCHID	This is macro GFX_W3_MEDIUMORCHID.
	GFX_W3_MEDIUMPURPLE	This is macro GFX_W3_MEDIUMPURPLE.
	GFX_W3_MEDIUMSEAGREEN	This is macro GFX_W3_MEDIUMSEAGREEN.
	GFX_W3_MEDIUMSLATEBLUE	This is macro GFX_W3_MEDIUMSLATEBLUE.
	GFX_W3_MEDIUMSPRINGGREEN	This is macro GFX_W3_MEDIUMSPRINGGREEN.
	GFX_W3_MEDIUMTURQUOISE	This is macro GFX_W3_MEDIUMTURQUOISE.
	GFX_W3_MEDIUMVIOLETRED	This is macro GFX_W3_MEDIUMVIOLETRED.
	GFX_W3_MIDNIGHTBLUE	This is macro GFX_W3_MIDNIGHTBLUE.
	GFX_W3_MINTCREAM	This is macro GFX_W3_MINTCREAM.
	GFX_W3_MISTYROSE	This is macro GFX_W3_MISTYROSE.
	GFX_W3_MOCCASIN	This is macro GFX_W3_MOCCASIN.
	GFX_W3_NAVAJOWHITE	This is macro GFX_W3_NAVAJOWHITE.
	GFX_W3_NAVY	This is macro GFX_W3_NAVY.
	GFX_W3_OLDLACE	This is macro GFX_W3_OLDLACE.
	GFX_W3_OLIVE	This is macro GFX_W3_OLIVE.
	GFX_W3_OLIVEDRAB	This is macro GFX_W3_OLIVEDRAB.
	GFX_W3_ORANGE	This is macro GFX_W3_ORANGE.
	GFX_W3_ORANGERED	This is macro GFX_W3_ORANGERED.
	GFX_W3_ORCHID	This is macro GFX_W3_ORCHID.
	GFX_W3_PALEGOLDENROD	This is macro GFX_W3_PALEGOLDENROD.
	GFX_W3_PALEGREEN	This is macro GFX_W3_PALEGREEN.
	GFX_W3_PALETURQUOISE	This is macro GFX_W3_PALETURQUOISE.
	GFX_W3_PALEVIOLETRED	This is macro GFX_W3_PALEVIOLETRED.
	GFX_W3_PAPAYAWHIP	This is macro GFX_W3_PAPAYAWHIP.
	GFX_W3_PEACHPUFF	This is macro GFX_W3_PEACHPUFF.
	GFX_W3_PERU	This is macro GFX_W3_PERU.
	GFX_W3_PINK	This is macro GFX_W3_PINK.
	GFX_W3_PLUM	This is macro GFX_W3_PLUM.
	GFX_W3_POWDERBLUE	This is macro GFX_W3_POWDERBLUE.
	GFX_W3_PURPLE	This is macro GFX_W3_PURPLE.
	GFX_W3_RED	This is macro GFX_W3_RED.
	GFX_W3_ROSYBROWN	This is macro GFX_W3_ROSYBROWN.

	GFX_W3_ROYALBLUE	This is macro GFX_W3_ROYALBLUE.
	GFX_W3_SADDLEBROWN	This is macro GFX_W3_SADDLEBROWN.
	GFX_W3_SALMON	This is macro GFX_W3_SALMON.
	GFX_W3_SANDYGREEN	This is macro GFX_W3_SANDYGREEN.
	GFX_W3_SEAGREEN	This is macro GFX_W3_SEAGREEN.
	GFX_W3_SEASHELL	This is macro GFX_W3_SEASHELL.
	GFX_W3_SIENNA	This is macro GFX_W3_SIENNA.
	GFX_W3_SILVER	This is macro GFX_W3_SILVER.
	GFX_W3_SKYBLUE	This is macro GFX_W3_SKYBLUE.
	GFX_W3_SLATEBLUE	This is macro GFX_W3_SLATEBLUE.
	GFX_W3_SLATEGRAY	This is macro GFX_W3_SLATEGRAY.
	GFX_W3_SLATEGREY	This is macro GFX_W3_SLATEGREY.
	GFX_W3_SNOW	This is macro GFX_W3_SNOW.
	GFX_W3_SPRINGGREEN	This is macro GFX_W3_SPRINGGREEN.
	GFX_W3_STEELBLUE	This is macro GFX_W3_STEELBLUE.
	GFX_W3_TAN	This is macro GFX_W3_TAN.
	GFX_W3_TEAL	This is macro GFX_W3_TEAL.
	GFX_W3_THISTLE	This is macro GFX_W3_THISTLE.
	GFX_W3_TOMATO	This is macro GFX_W3_TOMATO.
	GFX_W3_TURQUOISE	This is macro GFX_W3_TURQUOISE.
	GFX_W3_VIOLET	This is macro GFX_W3_VIOLET.
	GFX_W3_WHEAT	This is macro GFX_W3_WHEAT.
	GFX_W3_WHITE	This is macro GFX_W3_WHITE.
	GFX_W3_WHITESMOKE	This is macro GFX_W3_WHITESMOKE.
	GFX_W3_YELLOW	This is macro GFX_W3_YELLOW.
	GFX_W3_YELLOWGREEN	This is macro GFX_W3_YELLOWGREEN.
	KHAKI	This is macro KHAKI.

Description

Module for Microchip Graphics Library

This header file provides W3 color definitions as an option of color definitions to use.

Remarks

http://www.w3.org/TR/SVG/types.html#ColorKeywords

File Name

gfx_colors_w3.h

Company

Microchip Technology Inc.

gfx_colors_x11.h

The header file contains X11 color definitions.

Macros

	Name	Description
	GFX_X11_ALICE_BLUE	This is macro GFX_X11_ALICE_BLUE.
	GFX_X11_ANTIQUE_WHITE	This is macro GFX_X11_ANTIQUE_WHITE.
	GFX_X11_AQUA	This is macro GFX_X11_AQUA.
	GFX_X11_AQUAMARINE	This is macro GFX_X11_AQUAMARINE.
	GFX_X11_AZURE	This is macro GFX_X11_AZURE.
	GFX_X11_BEIGE	This is macro GFX_X11_BEIGE.
	GFX_X11_BISQUE	This is macro GFX_X11_BISQUE.
	GFX_X11_BLACK	This is macro GFX_X11_BLACK.
	GFX_X11_BLANCHED_ALMOND	This is macro GFX_X11_BLANCHED_ALMOND.
	GFX_X11_BLUE	This is macro GFX_X11_BLUE.

	GFX_X11_BLUE_VIOLET	This is macro GFX_X11_BLUE_VIOLET.
	GFX_X11_BROWN	This is macro GFX_X11_BROWN.
	GFX_X11_BURLY_WOOD	This is macro GFX_X11_BURLY_WOOD.
	GFX_X11_CADEL_BLUE	This is macro GFX_X11_CADEL_BLUE.
	GFX_X11_CHARTEUSE	This is macro GFX_X11_CHARTEUSE.
	GFX_X11_CHOCOLATE	This is macro GFX_X11_CHOCOLATE.
	GFX_X11_CORAL	This is macro GFX_X11_CORAL.
	GFX_X11_CORNFLOWER_BLUE	This is macro GFX_X11_CORNFLOWER_BLUE.
	GFX_X11_CORNSILK	X11: Brown Colors
	GFX_X11_CRIMSON	This is macro GFX_X11_CRIMSON.
	GFX_X11_CYAN	This is macro GFX_X11_CYAN.
	GFX_X11_DARK_BLUE	This is macro GFX_X11_DARK_BLUE.
	GFX_X11_DARK_CYAN	This is macro GFX_X11_DARK_CYAN.
	GFX_X11_DARK_GOLDENROD	This is macro GFX_X11_DARK_GOLDENROD.
	GFX_X11_DARK_GREEN	This is macro GFX_X11_DARK_GREEN.
	GFX_X11_DARK_GREY	This is macro GFX_X11_DARK_GREY.
	GFX_X11_DARK_KHAKI	This is macro GFX_X11_DARK_KHAKI.
	GFX_X11_DARK_MAGENTA	This is macro GFX_X11_DARK_MAGENTA.
	GFX_X11_DARK_OLIVE_GREEN	X11: Green Colors
	GFX_X11_DARK_ORANGE	This is macro GFX_X11_DARK_ORANGE.
	GFX_X11_DARK_ORCHID	This is macro GFX_X11_DARK_ORCHID.
	GFX_X11_DARK_RED	This is macro GFX_X11_DARK_RED.
	GFX_X11_DARK_SALMON	This is macro GFX_X11_DARK_SALMON.
	GFX_X11_DARK_SEA_GREEN	This is macro GFX_X11_DARK_SEA_GREEN.
	GFX_X11_DARK_SLATE_BLUE	This is macro GFX_X11_DARK_SLATE_BLUE.
	GFX_X11_DARK_SLATE_GREY	This is macro GFX_X11_DARK_SLATE_GREY.
	GFX_X11_DARK_TURQUOISE	This is macro GFX_X11_DARK_TURQUOISE.
	GFX_X11_DARK_VIOLET	This is macro GFX_X11_DARK_VIOLET.
	GFX_X11_DEEP_PINK	This is macro GFX_X11_DEEP_PINK.
	GFX_X11_DEEP_SKY_BLUE	This is macro GFX_X11_DEEP_SKY_BLUE.
	GFX_X11_DIM_GREY	This is macro GFX_X11_DIM_GREY.
	GFX_X11_DODGER_BLUE	This is macro GFX_X11_DODGER_BLUE.
	GFX_X11_FIRE_BRICK	This is macro GFX_X11_FIRE_BRICK.
	GFX_X11_FLORAL_WHITE	This is macro GFX_X11_FLORAL_WHITE.
	GFX_X11_FOREST_GREEN	This is macro GFX_X11_FOREST_GREEN.
	GFX_X11_FUCHSIA	This is macro GFX_X11_FUCHSIA.
	GFX_X11_GAINSBORO	This is macro GFX_X11_GAINSBORO.
	GFX_X11_GHOST_WHITE	This is macro GFX_X11_GHOST_WHITE.
	GFX_X11_GOLD	This is macro GFX_X11_GOLD.
	GFX_X11_GOLDENROD	This is macro GFX_X11_GOLDENROD.
	GFX_X11_GREEN	This is macro GFX_X11_GREEN.
	GFX_X11_GREEN_YELLOW	This is macro GFX_X11_GREEN_YELLOW.
	GFX_X11_GREY	This is macro GFX_X11_GREY.
	GFX_X11_HONEYDEW	This is macro GFX_X11_HONEYDEW.
	GFX_X11_HOT_PINK	This is macro GFX_X11_HOT_PINK.
	GFX_X11_INDIAN_RED	This is macro GFX_X11_INDIAN_RED.
	GFX_X11_INDIGO	This is macro GFX_X11_INDIGO.
	GFX_X11_IVORY	This is macro GFX_X11_IVORY.
	GFX_X11_KHAKI	This is macro GFX_X11_KHAKI.
	GFX_X11_LAVENDER	X11: BLUE COLORS
	GFX_X11_LAVENDOR_BLUSH	This is macro GFX_X11_LAVENDOR_BLUSH.
	GFX_X11_LAWN_GREEN	This is macro GFX_X11_LAWN_GREEN.
	GFX_X11_LEMON_CHIFFON	This is macro GFX_X11_LEMON_CHIFFON.
	GFX_X11_LIGHT_BLUE	This is macro GFX_X11_LIGHT_BLUE.
	GFX_X11_LIGHT_CAROL	This is macro GFX_X11_LIGHT_CAROL.

	GFX_X11_LIGHT_CYAN	This is macro GFX_X11_LIGHT_CYAN.
	GFX_X11_LIGHT_GOLDENROD_YELLOW	This is macro GFX_X11_LIGHT_GOLDENROD_YELLOW.
	GFX_X11_LIGHT_GRAY	This is macro GFX_X11_LIGHT_GRAY.
	GFX_X11_LIGHT_GREEN	This is macro GFX_X11_LIGHT_GREEN.
	GFX_X11_LIGHT_PINK	This is macro GFX_X11_LIGHT_PINK.
	GFX_X11_LIGHT_SALMON	X11: Red Colors
	GFX_X11_LIGHT_SEA_GREEN	This is macro GFX_X11_LIGHT_SEA_GREEN.
	GFX_X11_LIGHT_SKY_BLUE	This is macro GFX_X11_LIGHT_SKY_BLUE.
	GFX_X11_LIGHT_SLATE_GREY	This is macro GFX_X11_LIGHT_SLATE_GREY.
	GFX_X11_LIGHT_STEEL_BLUE	X11: BLUE COLORS
	GFX_X11_LIGHT_YELLOW	This is macro GFX_X11_LIGHT_YELLOW.
	GFX_X11_LIME	This is macro GFX_X11_LIME.
	GFX_X11_LIME_GREEN	This is macro GFX_X11_LIME_GREEN.
	GFX_X11_LINEN	This is macro GFX_X11_LINEN.
	GFX_X11_MAGENTA	This is macro GFX_X11_MAGENTA.
	GFX_X11_MARRON	This is macro GFX_X11_MARRON.
	GFX_X11_MEDIUM_AQUAMARINE	X11: CYAN COLORS
	GFX_X11_MEDIUM_BLUE	This is macro GFX_X11_MEDIUM_BLUE.
	GFX_X11_MEDIUM_ORCHID	This is macro GFX_X11_MEDIUM_ORCHID.
	GFX_X11_MEDIUM_PURPLE	This is macro GFX_X11_MEDIUM_PURPLE.
	GFX_X11_MEDIUM_SEA_GREEN	This is macro GFX_X11_MEDIUM_SEA_GREEN.
	GFX_X11_MEDIUM_SLATE_BLUE	This is macro GFX_X11_MEDIUM_SLATE_BLUE.
	GFX_X11_MEDIUM_SPRING_GREEN	This is macro GFX_X11_MEDIUM_SPRING_GREEN.
	GFX_X11_MEDIUM_TURQUOISE	This is macro GFX_X11_MEDIUM_TURQUOISE.
	GFX_X11_MEDIUM_VIOLET_RED	This is macro GFX_X11_MEDIUM_VIOLET_RED.
	GFX_X11_MIDNIGHT_BLUE	This is macro GFX_X11_MIDNIGHT_BLUE.
	GFX_X11_MINT_CREAM	This is macro GFX_X11_MINT_CREAM.
	GFX_X11_MISTY_ROSE	This is macro GFX_X11_MISTY_ROSE.
	GFX_X11_MOCCASIN	This is macro GFX_X11_MOCCASIN.
	GFX_X11_NAVAJO_WHITE	This is macro GFX_X11_NAVAJO_WHITE.
	GFX_X11_NAVY	This is macro GFX_X11_NAVY.
	GFX_X11_OLD_LACE	This is macro GFX_X11_OLD_LACE.
	GFX_X11_OLIVE	This is macro GFX_X11_OLIVE.
	GFX_X11_OLIVE_DRAB	This is macro GFX_X11_OLIVE_DRAB.
	GFX_X11_ORANGE	This is macro GFX_X11_ORANGE.
	GFX_X11_ORANGE_RED	X11: Orange Colors
	GFX_X11_ORCHID	This is macro GFX_X11_ORCHID.
	GFX_X11_PALE_GOLDENROD	This is macro GFX_X11_PALE_GOLDENROD.
	GFX_X11_PALE_GREEN	This is macro GFX_X11_PALE_GREEN.
	GFX_X11_PALE_TURQUOISE	This is macro GFX_X11_PALE_TURQUOISE.
	GFX_X11_PALE_VIOLET_RED	This is macro GFX_X11_PALE_VIOLET_RED.
	GFX_X11_PAPAYA_WHIP	This is macro GFX_X11_PAPAYA_WHIP.
	GFX_X11_PEACH_PUFF	This is macro GFX_X11_PEACH_PUFF.
	GFX_X11_PERU	This is macro GFX_X11_PERU.
	GFX_X11_PINK	X11: Pink Colors
	GFX_X11_PLUM	This is macro GFX_X11_PLUM.
	GFX_X11_POWDER_BLUE	This is macro GFX_X11_POWDER_BLUE.
	GFX_X11_PURPLE	This is macro GFX_X11_PURPLE.
	GFX_X11_RED	This is macro GFX_X11_RED.
	GFX_X11_ROSY_BROWN	This is macro GFX_X11_ROSY_BROWN.
	GFX_X11_ROYAL_BLUE	This is macro GFX_X11_ROYAL_BLUE.
	GFX_X11_SADDLE_BROWN	This is macro GFX_X11_SADDLE_BROWN.
	GFX_X11_SALMON	This is macro GFX_X11_SALMON.
	GFX_X11_SANDY_BROWN	This is macro GFX_X11_SANDY_BROWN.
	GFX_X11_SEA_GREEN	This is macro GFX_X11_SEA_GREEN.

	GFX_X11_SEASHELL	This is macro GFX_X11_SEASHELL.
	GFX_X11_SIENNA	This is macro GFX_X11_SIENNA.
	GFX_X11_SILVER	This is macro GFX_X11_SILVER.
	GFX_X11_SKY_BLUE	This is macro GFX_X11_SKY_BLUE.
	GFX_X11_SLATE_BLUE	This is macro GFX_X11_SLATE_BLUE.
	GFX_X11_SLATE_GREY	This is macro GFX_X11_SLATE_GREY.
	GFX_X11_SNOW	This is macro GFX_X11_SNOW.
	GFX_X11_SPRING_GREEN	This is macro GFX_X11_SPRING_GREEN.
	GFX_X11_STEEL_BLUE	This is macro GFX_X11_STEEL_BLUE.
	GFX_X11_TAN	This is macro GFX_X11_TAN.
	GFX_X11_TEAL	This is macro GFX_X11_TEAL.
	GFX_X11_THISTLE	This is macro GFX_X11_THISTLE.
	GFX_X11_TOMATO	This is macro GFX_X11_TOMATO.
	GFX_X11_TURQUOISE	This is macro GFX_X11_TURQUOISE.
	GFX_X11_VIOLET	This is macro GFX_X11_VIOLET.
	GFX_X11_WHEAT	This is macro GFX_X11_WHEAT.
	GFX_X11_WHITE	X11: WHITE/GRAY/BLACK COLORS
	GFX_X11_WHITE_SMOKE	This is macro GFX_X11_WHITE_SMOKE.
	GFX_X11_YELLOW	X11: Yellow Colors
	GFX_X11_YELLOW_GREEN	This is macro GFX_X11_YELLOW_GREEN.

Description

Module for Microchip Graphics Library

This header file provides X11 color definitions as an option of color definitions to use.

File Name

gfx_colors_x11.h

Company

Microchip Technology Inc.

gfx_gol.h

Graphics Object Layer of the Microchip Graphics Library.

Enumerations

	Name	Description
	GFX_GOL_CLIENT_STATUS	Enumerated data type that identifies the GFX Module Client Status.
	GFX_GOL_COMMON_STATE_BITS	Common Object States.
	GFX_GOL_OBJ_TYPE	Specifies the different object types used in the library.
	GFX_GOL_OBJECT_TASK	Lists the different states that GFX GOL task routine can have.
	GFX_GOL_STATES	Defines the various states that can be achieved by the GFX GOL operation.
	GFX_GOL_TRANSLATED_ACTION	Specifies the different object actions supported in the library.

Functions

	Name	Description
	GFX_GOL_Deinitialize	Deinitializes the specified instance of the GFX Module.
	GFX_GOL_DrawCallbackSet	This function sets the draw callback function that the application will use to render application specific rendering.
	GFX_GOL_Initialize	Initialize the GFX GOL Library.
	GFX_GOL_MessageCallbackSet	This function sets the message callback function that the application will use to evaluate user inputs that affects the objects and application behavior.
	GFX_GOL_ObjectAdd	This function adds an object to the tail of the currently active list.
	GFX_GOL_ObjectBackGroundSet	This function sets the background information.
	GFX_GOL_ObjectByIDDelete	This function removes an object with the given user defined ID from the currently active list.
	GFX_GOL_ObjectCanBeFocused	Checks if the object can be focused.

	GFX_GOL_ObjectDelete	This function removes an object from the currently active list.
	GFX_GOL_ObjectDrawDisable	This function resets the drawing state bits of the object.
	GFX_GOL_ObjectDrawEnable	This function sets the object to be redraw.
	GFX_GOL_ObjectFind	This function returns the pointer to object in the list with the user defined ID assigned to it.
	GFX_GOL_ObjectFocusGet	This function returns the pointer to the object that is currently receiving keyboard input (or focused).
	GFX_GOL_ObjectFocusNextGet	This function returns the pointer to the next object in the active list of objects which can be focused.
	GFX_GOL_ObjectFocusPrevGet	This function returns the pointer to the previous object in the active list of objects which can be focused.
	GFX_GOL_ObjectFocusSet	This function sets the object to be focused.
	GFX_GOL_ObjectHideDraw	This function performs the hiding of an object from the screen.
	GFX_GOL_ObjectIDGet	This function returns the object ID.
	GFX_GOL_ObjectIsRedrawSet	This function checks if the object needs to be redrawn or not.
	GFX_GOL_ObjectListDraw	This function redraws all objects in the current active list that has the rendering state bits set.
	GFX_GOL_ObjectListFree	This function sets the active list to the new list.
	GFX_GOL_ObjectListGet	This function returns the current active list.
	GFX_GOL_ObjectListHide	This function marks all objects in the active list to be hidden.
	GFX_GOL_ObjectListNew	This function removes an object with the given user defined ID from the currently active list.
	GFX_GOL_ObjectListSet	This function sets the active list to the new list.
	GFX_GOL_ObjectMessage	This function process the received message from the user to determine the affected objects. Depending on the message and the affected objects, object states are modified based on the default or user-defined behavior.
	GFX_GOL_ObjectNextGet	This function returns the pointer to next object in the list after the specified object.
	GFX_GOL_ObjectRectangleRedraw	This function marks all objects in the active list intersected by the given rectangular area to be redrawn.
	GFX_GOL_ObjectTypeGet	This function returns the object type.
	GFX_GOL_Open	opens an instance of the GFX GOL Library.
	GFX_GOL_PanelAlphaParameterSet	This function sets the alpha blending value when using alpha blending in panels.
	GFX_GOL_PanelBackgroundSet	This function sets panel background information.
	GFX_GOL_PanelDraw	This function renders the panel.
	GFX_GOL_PanelGradientParameterSet	This function sets the gradient fill start and end colors of a panel.
	GFX_GOL_PanelParameterSet	This function sets the parameters to draw a panel.
	GFX_GOL_PreemptionLevelGet	This function returns the current preemption level.
	GFX_GOL_Tasks	Maintains the GFX GOL module state machine. It manages the GFX Module object list items and responds to GFX Module primitive events.
	GFX_GOL_TwoTonePanelDraw	This function renders the two-tone panel.

Macros

	Name	Description
	GFX_GOL_ObjectStateClear	This function clears the state bits of the given object.
	GFX_GOL_ObjectStateGet	This function retrieves the current value of the state bits of an object.
	GFX_GOL_ObjectStateSet	This function sets the state bits of the given object.
	GFX_GOL_ObjectStyleSchemeGet	This function returns the style scheme currently set for the object.
	GFX_GOL_ObjectStyleSchemeSet	This function sets the style scheme of the object.

Structures

	Name	Description
	GFX_GOL_MESSAGE	Specifies message structure used in the library.
	GFX_GOL_OBJ_HEADER	Specifies Graphics Object Layer structure used in objects.
	GOL_PANEL_PARAM	Specifies panel parameters.

Types

	Name	Description
	GFX_GOL_DRAW_CALLBACK_FUNC	Draw callback function definition. This application defined function allows the application to perform application specific rendering.

	GFX_GOL_MESSAGE_CALLBACK_FUNC	Message callback function definition. This application defined function allows the application to perform application specific processing of user messages.

Description

Module for Microchip Graphics Library - Graphic Object Layer

This module implements the common routines for the Graphics Object Layer of the Microchip Graphics Library. The routines are independent of the Display Driver Layer and should be compatible with any Display Driver that is compliant with the requirements of the Display Driver Layer of the Graphics Library. The module utilizes the Graphics Primitive Layer to render the objects.

File Name

gfx_gol.h

Company

Microchip Technology Inc.

gfx_gol_button.h

This is the header file for the button object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_BUTTON_STATE	Specifies the different states of the Button object.

Functions

	Name	Description
	GFX_GOL_ButtonActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_ButtonActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_ButtonCreate	This function creates a GFX_GOL_BUTTON object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_ButtonDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_ButtonTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_ButtonTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_ButtonTextSet	This function sets the address of the current text string used by the object.

Macros

	Name	Description
	GFX_GOL_ButtonPressStateImageGet	This function gets the image used when in the pressed state.
	GFX_GOL_ButtonPressStateImageSet	This function sets the image to be used when in the pressed state.
	GFX_GOL_ButtonReleaseStateImageGet	This function gets the image used when in the released state.
	GFX_GOL_ButtonReleaseStateImageSet	This function sets the image to be used when in the released state.
	GFX_GOL_ButtonTextGet	This function returns the address of the current text string used by the object.

Structures

	Name	Description
	GFX_GOL_BUTTON	Defines the structure used for the Button object.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the GFX_GOL_BUTTON Object.

File Name

gfx_gol_button.h

Company

Microchip Technology Inc.

gfx_gol_check_box.h

This is the header file for the check box object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_CHECKBOX_STATE	Specifies the different states of the Check Box object.

Functions

	Name	Description
	GFX_GOL_CheckBoxActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_CheckBoxActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_CheckBoxCreate	This function creates a GFX_GOL_CHECKBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_CheckBoxDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_CheckBoxTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_CheckBoxTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_CheckBoxTextSet	This function sets the address of the current text string used by the object.

Macros

	Name	Description
	GFX_GOL_CheckBoxTextGet	This function returns the address of the current text string used by the object.

Structures

	Name	Description
	GFX_GOL_CHECKBOX	Defines the structure used for the Check Box object.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the CheckBox Object.

File Name

gfx_gol_check_box.h

Company

Microchip Technology Inc.

gfx_gol_custom_control.h

This is the header file for the custom control object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_CUSTOMCONTROL_STATE	Specifies the different states of the Custom Control object.

Functions

	Name	Description
	GFX_GOL_CustomControlActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_CustomControlActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_CustomControlCreate	This function creates a GFX_GOL_CUSTOMCONTROL object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_CustomControlDraw	This function renders the object on the screen based on the current state of the object.

Macros

	Name	Description
	GFX_GOL_CustomControlGetPos	This function returns the position of the control.
	GFX_GOL_CustomControlSetPos	This function sets the position of the control.

Structures

	Name	Description
	GFX_GOL_CUSTOMCONTROL	Defines the structure used for the Custom Control object.

Description

Graphics Library Implementation

Refer to Microchip Graphics Library for complete documentation of the CustomControl Object.

File Name

gfx_gol_custom_control.c

Company

Microchip Technology Inc.

gfx_gol_digital_meter.h

This is the header file for the digital meter object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_DIGITALMETER_STATE	Specifies the different states of the Digital Meter object.

Functions

	Name	Description
	GFX_GOL_DigitalMeterActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_DigitalMeterCreate	This function creates a GFX_GOL_DIGITALMETER object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_DigitalMeterDecrement	This function decrements the meter value by the delta value set.
	GFX_GOL_DigitalMeterDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_DigitalMeterIncrement	This function increments the meter value by the delta value set.
	GFX_GOL_DigitalMeterValueSet	This function sets the value of the meter.

Macros

	Name	Description
	GFX_GOL_DigitalMeterTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_DigitalMeterTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_DigitalMeterValueGet	This function returns the current value of the digital meter.

Structures

	Name	Description
	GFX_GOL_DIGITALMETER	Defines the structure used for the Digital Meter object.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Digital Meter Object.

File Name

gfx_gol_digital_meter.h

Company

Microchip Technology Inc.

gfx_gol_edit_box.h

This is the header file for the edit box object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_EDITBOX_STATE	Specifies the different states of the Edit Box object.

Functions

	Name	Description
	GFX_GOL_EditBoxActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_EditBoxActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_EditBoxCharAdd	This function adds a character at the end of the text used by the object.
	GFX_GOL_EditBoxCharRemove	This function removes a character at the end of the text used by the object.
	GFX_GOL_EditBoxCreate	This function creates a GFX_GOL_EDITBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_EditBoxCursorPositionDecrement	Decrement cursor position.
	GFX_GOL_EditBoxCursorPositionIncrement	Increment cursor position.
	GFX_GOL_EditBoxDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_EditBoxTextSet	This function sets the text in the object.

Macros

	Name	Description
	GFX_GOL_EditBoxTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_EditBoxTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_EditBoxTextGet	This function returns the address of the current text string used by the object.

Structures

	Name	Description
	GFX_GOL_EDITBOX	Defines the structure used for the Edit Box object.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Edit Box Object.

File Name

gfx_gol_edit_box.h

Company

Microchip Technology Inc.

gfx_gol_group_box.h

This is the header file for the group box object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_GROUPBOX_STATE	Specifies the different states of the Group Box object.

Functions

	Name	Description
	GFX_GOL_GroupboxActionGet	This function evaluates the message from a user if the message will affect the object or not.

	GFX_GOL_GroupboxCreate	This function creates a GFX_GOL_GROUPBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_GroupboxDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_GroupboxTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_GroupboxTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_GroupboxTextSet	This function sets the address of the current text string used by the object.

Macros

	Name	Description
	GFX_GOL_GroupboxTextGet	This function returns the address of the current text string used by the object.

Structures

	Name	Description
	GFX_GOL_GROUPBOX	Defines the structure used for the Group Box object.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Group Box Object.

File Name

gfx_gol_group_box.h

Company

Microchip Technology Inc.

gfx_gol_list_box.h

This is the header file for the list box object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_LISTBOX_ITEM_STATUS	Defines the types used to indicate the status of an item.
	GFX_GOL_LISTBOX_STATE	Specifies the different states of the List Box object.

Functions

	Name	Description
	GFX_GOL_ListBoxActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_ListBoxActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_ListBoxCreate	This function creates a GFX_GOL_LISTBOX object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_ListBoxDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_ListBoxItemAdd	This function adds an item to the list box.
	GFX_GOL_ListBoxItemFocusGet	This function returns the index of the focused item in the list box.
	GFX_GOL_ListBoxItemFocusSet	This function sets the focus of the item with the index number specified by index.
	GFX_GOL_ListBoxItemListRemove	This function removes the entire items list of the list box and free the memory used.
	GFX_GOL_ListBoxItemRemove	This function removes an item from the list box and free the memory used.
	GFX_GOL_ListBoxSelectionChange	This function changes the selection status of the given item in the list box.
	GFX_GOL_ListBoxSelectionGet	This function searches for selected items from the list box.
	GFX_GOL_ListBoxVisibleItemCountGet	This function returns the count of items visible in the list box.

Macros

	Name	Description
	GFX_GOL_ListBoxItemCountGet	This function returns the count of items in the list box.
	GFX_GOL_ListBoxItemImageGet	This function gets the image set for a list box item.
	GFX_GOL_ListBoxItemImageSet	This function sets the image for a list box item.

	GFX_GOL_ListBoxItemListGet	This function returns the pointer to the item list of the list box.
	GFX_GOL_ListBoxItemSelectStatusClear	This function clears the selection status of the item.
	GFX_GOL_ListBoxItemSelectStatusSet	This function sets the selection status of the item to selected.
	GFX_GOL_ListBoxTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_ListBoxTextAlignmentSet	This function sets the text alignment of the text string used by the object.

Structures

	Name	Description
	GFX_GOL_LISTBOX	Defines the structure used for the List Box object.
	GFX_GOL_LISTITEM	The structure that defines each item in the list box.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the ListBox Object.

File Name

gfx_gol_list_box.h

Company

Microchip Technology Inc.

gfx_gol_meter.h

This is the header file for the meter object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_METER_DRAW_TYPE	Specifies the different types of Meter object.
	GFX_GOL_METER_STATE	Specifies the different states of the Meter object.

Functions

	Name	Description
	GFX_GOL_MeterActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_MeterActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_MeterCreate	This function creates a GFX_GOL_METER object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_MeterDecrement	This function decrements the meter value by the delta value set.
	GFX_GOL_MeterDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_MeterIncrement	This function increments the meter value by the delta value set.
	GFX_GOL_MeterRangeSet	This function sets the range of the meter.
	GFX_GOL_MeterScaleColorsSet	This function sets the arc colors of the object.
	GFX_GOL_MeterValueSet	This function sets the value of the meter.

Macros

	Name	Description
	GFX_GOL_MeterMaximumValueGet	This function returns the maximum value set for the meter.
	GFX_GOL_MeterMinimumValueGet	This function returns the minimum value set for the meter.
	GFX_GOL_MeterTitleFontSet	This function sets the meter title font used.
	GFX_GOL_MeterTypeSet	This function sets the meter draw type.
	GFX_GOL_MeterValueFontSet	This function sets the meter's displayed value font used.
	GFX_GOL_MeterValueGet	This function returns the current value of the meter.

Structures

	Name	Description
	GFX_GOL_METER	Defines the structure used for the Meter object.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Meter Object.

File Name

gfx_gol_meter.h

Company

Microchip Technology Inc.

gfx_gol_picture.h

This is the header file for the picture object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_PICTURECONTROLCONTROL_STATE	Specifies the different states of the Picture Control object.

Functions

	Name	Description
	GFX_GOL_PictureControlActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_PictureControlCreate	This function creates a GFX_GOL_PICTURECONTROL object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_PictureControlDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_PictureControlPartialSet	This function sets the partial image parameters to be in the object.
	GFX_GOL_PictureControlScaleSet	Sets the scale factor used to render the image used in the object.

Macros

	Name	Description
	GFX_GOL_PictureControlImageGet	This function gets the image used when in the pressed state.
	GFX_GOL_PictureControlImageSet	This function sets the image to be in the object.

Structures

	Name	Description
	GFX_GOL_PICTURECONTROL	Defines the structure used for the Picture Control object.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Picture Object.

File Name

gfx_gol_picture.h

Company

Microchip Technology Inc.

gfx_gol_progress_bar.h

This is the header file for the progress bar object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_PROGRESSBAR_STATE	Specifies the different states of the Progress Bar object.

Functions

	Name	Description
	GFX_GOL_ProgressBarActionGet	This function evaluates the message from a user if the message will affect the object or not.

	GFX_GOL_ProgressBarCreate	This function creates a GFX_GOL_PROGRESSBAR object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_ProgressBarDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_ProgressBarPositionSet	This function sets the position of the progress bar.
	GFX_GOL_ProgressBarRangeSet	This function sets the range of the progress bar.

Macros

	Name	Description
	GFX_GOL_ProgressBarPositionGet	This function returns the current position of the progress bar.
	GFX_GOL_ProgressBarRangeGet	This function returns the range of the progress bar.

Structures

	Name	Description
	GFX_GOL_PROGRESSBAR	Defines the structure used for the Progress Bar object.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the ProgressBar Object.

File Name

gfx_gol_progress_bar.h

Company

Microchip Technology Inc.

gfx_gol_radio_button.h

This is the header file for the radio button object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_RADIOBUTTON_STATE	Specifies the different states of the Radio Button object.

Functions

	Name	Description
	GFX_GOL_RadioButtonActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_RadioButtonActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_RadioButtonCheckGet	This function returns the ID of the currently checked radio button in the group.
	GFX_GOL_RadioButtonCheckSet	This function sets the ID of the currently checked radio button in the group.
	GFX_GOL_RadioButtonCreate	This function creates a GFX_GOL_RADIOBUTTON object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_RadioButtonDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_RadioButtonListCreate	This is function GFX_GOL_RadioButtonListCreate.
	GFX_GOL_RadioButtonTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_RadioButtonTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_RadioButtonTextSet	This function sets the address of the current text string used by the object.

Macros

	Name	Description
	GFX_GOL_RadioButtonTextGet	This function returns the address of the current text string used by the object.

Structures

	Name	Description
	GFX_GOL_RADIOBUTTON	Defines the structure used for the Radio Button object.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the RadioButton Object.

File Name

gfx_gol_radio_button.h

Company

Microchip Technology Inc.

gfx_gol_scan_codes.h

This is the header file for the AT keyboard codes.

Enumerations

	Name	Description
	GFX_AT_SCAN_CODES	The following macros are the supported AT Keyboard scan codes.

Description

Module for Microchip Graphics Library - Graphic Object Layer

This header file contains scan codes for the standard AT keyboard that can be used to process user action in GOL messages.

File Name

gfx_gol_scan_codes.h

Company

Microchip Technology Inc.

gfx_gol_scheme.h

Graphics Object Layer Scheme routines/macros of the Microchip Graphics Library.

Functions

	Name	Description
	GFX_GOL_SchemeAlphaPrecentSet	This is function GFX_GOL_SchemeAlphaPrecentSet.
	GFX_GOL_SchemeBackgroundColorGet	This is function GFX_GOL_SchemeBackgroundColorGet.
	GFX_GOL_SchemeBackgroundColorSet	This is function GFX_GOL_SchemeBackgroundColorSet.
	GFX_GOL_SchemeBackgroundImageSet	This is function GFX_GOL_SchemeBackgroundImageSet.
	GFX_GOL_SchemeBackgroundTypeGet	This is function GFX_GOL_SchemeBackgroundTypeGet.
	GFX_GOL_SchemeBackgroundTypeSet	This is function GFX_GOL_SchemeBackgroundTypeSet.
	GFX_GOL_SchemeColor0Get	This is function GFX_GOL_SchemeColor0Get.
	GFX_GOL_SchemeColor0Set	This is function GFX_GOL_SchemeColor0Set.
	GFX_GOL_SchemeColor1Get	This is function GFX_GOL_SchemeColor1Get.
	GFX_GOL_SchemeColor1Set	This is function GFX_GOL_SchemeColor1Set.
	GFX_GOL_SchemeColorDisabledGet	This is function GFX_GOL_SchemeColorDisabledGet.
	GFX_GOL_SchemeColorDisabledSet	This is function GFX_GOL_SchemeColorDisabledSet.
	GFX_GOL_SchemeColorSet	This is function GFX_GOL_SchemeColorSet.
	GFX_GOL_SchemeEmbossDarkColorGet	This is function GFX_GOL_SchemeEmbossDarkColorGet.
	GFX_GOL_SchemeEmbossDarkColorSet	This is function GFX_GOL_SchemeEmbossDarkColorSet.
	GFX_GOL_SchemeEmbossLightColorGet	This is function GFX_GOL_SchemeEmbossLightColorGet.
	GFX_GOL_SchemeEmbossLightColorSet	This is function GFX_GOL_SchemeEmbossLightColorSet.
	GFX_GOL_SchemeEmbossSet	This is function GFX_GOL_SchemeEmbossSet.
	GFX_GOL_SchemeEmbossSizeGet	This is function GFX_GOL_SchemeEmbossSizeGet.
	GFX_GOL_SchemeEmbossSizeSet	This is function GFX_GOL_SchemeEmbossSizeSet.
	GFX_GOL_SchemeFillStyleGet	This is function GFX_GOL_SchemeFillStyleGet.
	GFX_GOL_SchemeFillStyleSet	This is function GFX_GOL_SchemeFillStyleSet.
	GFX_GOL_SchemeFontGet	This is function GFX_GOL_SchemeFontGet.

	GFX_GOL_SchemeFontSet	This is function GFX_GOL_SchemeFontSet.
	GFX_GOL_SchemeGradientColorSet	This is function GFX_GOL_SchemeGradientColorSet.
	GFX_GOL_SchemeGradientEndColorGet	This is function GFX_GOL_SchemeGradientEndColorGet.
	GFX_GOL_SchemeGradientEndColorSet	This is function GFX_GOL_SchemeGradientEndColorSet.
	GFX_GOL_SchemeGradientStartColorGet	This is function GFX_GOL_SchemeGradientStartColorGet.
	GFX_GOL_SchemeGradientStartColorSet	This is function GFX_GOL_SchemeGradientStartColorSet.
	GFX_GOL_SchemeTextColor0Get	This is function GFX_GOL_SchemeTextColor0Get.
	GFX_GOL_SchemeTextColor0Set	This is function GFX_GOL_SchemeTextColor0Set.
	GFX_GOL_SchemeTextColor1Get	This is function GFX_GOL_SchemeTextColor1Get.
	GFX_GOL_SchemeTextColor1Set	This is function GFX_GOL_SchemeTextColor1Set.
	GFX_GOL_SchemeTextColorDisableGet	This is function GFX_GOL_SchemeTextColorDisableGet.
	GFX_GOL_SchemeTextColorDisableSet	This is function GFX_GOL_SchemeTextColorDisableSet.
	GFX_GOL_SchemeTextColorSet	This is function GFX_GOL_SchemeTextColorSet.

Structures

	Name	Description
	GFX_GOL_OBJ_SCHEME	This structure specifies the style scheme components of an object.

Description

Module for Microchip Graphics Library - Graphic Object Layer Scheme

This module implements the common routines for the Graphics Object Layer of the Microchip Graphics Library. The routines are independent of the Display Driver Layer and should be compatible with any Display Driver that is compliant with the requirements of the Display Driver Layer of the Graphics Library. The module utilizes the Graphics Primitive Layer to render the objects.

File Name

gfx_gol_scheme.h

Company

Microchip Technology Inc.

gfx_gol_scroll_bar.h

This is the header file for the scroll bar object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_SCROLLBAR_STATE	Specifies the different states of the Scroll Bar object.

Functions

	Name	Description
	GFX_GOL_ScrollBarActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_ScrollBarActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_ScrollBarCreate	This function creates a GFX_GOL_SCROLLBAR object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_ScrollBarDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_ScrollBarPageGet	This function returns the page size of the object.
	GFX_GOL_ScrollBarPageSet	This function sets the page size of the object.
	GFX_GOL_ScrollBarPositionDecrement	This function decrements the scroll bar position by the delta change (page) value set.
	GFX_GOL_ScrollBarPositionGet	This function returns the current position of the scroll bar thumb.
	GFX_GOL_ScrollBarPositionIncrement	This function increments the scroll bar position by the delta change (page) value set.
	GFX_GOL_ScrollBarPositionSet	This function sets the position of the scroll bar thumb.
	GFX_GOL_ScrollBarRangeGet	This function returns the range of the thumb of the scroll bar.
	GFX_GOL_ScrollBarRangeSet	This function sets the range of the thumb of the scroll bar.

Structures

	Name	Description
	GFX_GOL_SCROLLBAR	Defines the structure used for the Scroll Bar object.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Slider Object.

File Name

gfx_gol_scroll_bar.h

Company

Microchip Technology Inc.

gfx_gol_static_text.h

This is the header file for the static text object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_STATICTEXT_STATE	Specifies the different states of the Static Text object.

Functions

	Name	Description
	GFX_GOL_StaticTextActionGet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_StaticTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_StaticTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_StaticTextCreate	This function creates a GFX_GOL_STATICTEXT object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_StaticTextDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_StaticTextGet	This function returns the address of the current text string used by the object.
	GFX_GOL_StaticTextSet	This function sets the address of the current text string used by the object.

Structures

	Name	Description
	GFX_GOL_STATICTEXT	Defines the structure used for the Static Text object.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Static Text Object.

File Name

gfx_gol_static_text.h

Company

Microchip Technology Inc.

gfx_gol_surface.h

This is the header file for the surface object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_SURFACE_STATE	Specifies the different states of the Surface object.

Functions

	Name	Description
	GFX_GOL_SurfaceActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_SurfaceCreate	This function creates a GFX_GOL_SURFACE object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_SurfaceDraw	This function renders the object on the screen based on the current state of the object.

Macros

	Name	Description
	GFX_GOL_SurfaceImageGet	This function gets the image used.
	GFX_GOL_SurfaceImageSet	This function sets the image used in the object.

Structures

	Name	Description
	GFX_GOL_SURFACE	Defines the structure used for the Surface object.

Types

	Name	Description
	GFX_GOL_SURFACE_CALLBACK_FUNC	Surface callback function definition. This application defined function allows the application to perform application specific rendering.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Surface Object.

File Name

gfx_gol_surface.h

Company

Microchip Technology Inc.

gfx_gol_text_entry.h

This is the header file for the text entry object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE	Specifies the different commands available for command keys of the Text Entry object.
	GFX_GOL_TEXTENTRY_STATE	Specifies the different states of the Text Entry object.

Functions

	Name	Description
	GFX_GOL_TextEntryActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_TextEntryActionSet	This function performs the state change of the object based on the translated action.
	GFX_GOL_TextEntryBufferClear	This function will clear the data in the display.
	GFX_GOL_TextEntryBufferGet	This function returns the buffer used to display text.
	GFX_GOL_TextEntryBufferSet	This function sets the buffer used to display text.
	GFX_GOL_TextEntryCharAdd	This function will insert a character to the end of the buffer.
	GFX_GOL_TextEntryCreate	This function creates a GFX_GOL_TEXTENTRY object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_TextEntryDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_TextEntryKeyCommandGet	This function will return the currently assigned command to the key with the given index.
	GFX_GOL_TextEntryKeyCommandSet	This function will assign a command to a key with the given index.

	GFX_GOL_TextEntryKeyIsPressed	This function will test if a key given by its index in the object is pressed.
	GFX_GOL_TextEntryKeyListCreate	This function will create the list of key members that holds the information on each key.
	GFX_GOL_TextEntryKeyMemberListDelete	This function will delete the key member list assigned to the object.
	GFX_GOL_TextEntryKeyTextSet	This function will set the text assigned to a key with the given index.
	GFX_GOL_TextEntryLastCharDelete	This function will remove the last character of the buffer and replace it with a string terminator.
	GFX_GOL_TextEntrySpaceCharAdd	This function will insert a space character to the end of the buffer.

Structures

	Name	Description
	GFX_GOL_TEXTENTRY	Defines the structure used for the Text Entry object.
	GFX_GOL_TEXTENTRY_KEYMEMBER	Defines the structure used to describe a key in the Text Entry object.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Text Entry Object.

File Name

gfx_gol_text_entry.h

Company

Microchip Technology Inc.

gfx_gol_window.h

This is the header file for the window object of the GOL.

Enumerations

	Name	Description
	GFX_GOL_WINDOW_STATE	Specifies the different states of the Window object.

Functions

	Name	Description
	GFX_GOL_WindowActionGet	This function evaluates the message from a user if the message will affect the object or not.
	GFX_GOL_WindowCreate	This function creates a GFX_GOL_WINDOW object with the parameters given. It automatically attaches the new object into a global linked list of objects and returns the address of the object.
	GFX_GOL_WindowDraw	This function renders the object on the screen based on the current state of the object.
	GFX_GOL_WindowTextAlignmentGet	This function returns the text alignment of the text string used by the object.
	GFX_GOL_WindowTextAlignmentSet	This function sets the text alignment of the text string used by the object.
	GFX_GOL_WindowTextSet	This function sets the address of the current text string used by the object.

Macros

	Name	Description
	GFX_GOL_WindowImageGet	This function gets the image used.
	GFX_GOL_WindowImageSet	This function sets the image used in the object.
	GFX_GOL_WindowTextGet	This function returns the address of the current text string used by the object.

Structures

	Name	Description
	GFX_GOL_WINDOW	Defines the structure used for the Window object.

Description

Module for Microchip Graphics Library - Graphic Object Layer

Refer to Microchip Graphics Library for complete documentation of the Window Object.

File Name

gfx_gol_window.h

Company

Microchip Technology Inc.

gfx_image_decoder.h

This implements the image decoding of the primitive layer.

Enumerations

	Name	Description
	_IMG_FILE_FORMAT	IMG_ImageFileFormat specifies all the supported image file formats
	IMG_FILE_FORMAT	IMG_ImageFileFormat specifies all the supported image file formats

Functions

	Name	Description
	ImageDecode	This function decodes and displays the image on the screen
	ImageDecoderInit	This function initializes the global variables to 0 and then initializes the driver. This must be called once before any other function of the library is called
	ImageDecoderParceHeader	This is function ImageDecoderParceHeader.
	ImageDecodeTask	This function completes one small part of the image decode function
	ImageLoopCallbackRegister	This function registers the loop callback function so that the decoder calls this function in every decoding loop. This can be used by the application program to do maintenance activities such as fetching data, updating the display, etc...
	ImagePixelIntoArray	This is function ImagePixelIntoArray.
	ImagePixelOutput	This is function ImagePixelOutput.
	ImageRegisterFrameCompleteCallback	This is function ImageRegisterFrameCompleteCallback.
	ImageRegisterImageCompleteCallback	Function prototypes
	ImageRegisterImageFailCallback	This is function ImageRegisterImageFailCallback.
	IMG_vSetboundaries	

Macros

	Name	Description
	__IMAGEDECODER_H__	This is macro __IMAGEDECODER_H__.
	_PutPixel	This is macro _PutPixel.
	ImageAbort	This function sets the Image Decoder's Abort flag so that decoding aborts in the next decoding loop.
	ImageFullScreenDecode	This function decodes and displays the image on the screen in fullscreen mode with center aligned and downscaled if required
	IMG_ALIGN_CENTER	Flags
	IMG_DECODE_ABORTED	This is macro IMG_DECODE_ABORTED.
	IMG_DOWN_SCALE	This is macro IMG_DOWN_SCALE.
	IMG_FEOF	This is macro IMG_FEOF.
	IMG_FILE	This is macro IMG_FILE.
	IMG_FREAD	This is macro IMG_FREAD.
	IMG_FSEEK	This is macro IMG_FSEEK.
	IMG_FTELL	This is macro IMG_FTELL.
	IMG_SCREEN_HEIGHT	This is macro IMG_SCREEN_HEIGHT.
	IMG_SCREEN_WIDTH	The individual image decoders use these defines instead of directly using those provided in the Display driver file
	IMG_vCheckAndAbort	This is macro IMG_vCheckAndAbort.
	IMG_vFrameComplete	This is macro IMG_vFrameComplete.
	IMG_vImageComplete	This is macro IMG_vImageComplete.
	IMG_vImageFail	This is macro IMG_vImageFail.
	IMG_vLoopCallback	This is macro IMG_vLoopCallback.
	IMG_vPixelArray	This is macro IMG_vPixelArray.
	IMG_vPutPixel	This is macro IMG_vPutPixel.
	IMG_vSetColor	This is macro IMG_vSetColor.

Structures

	Name	Description
	_IMG_FILE_SYSTEM_API	IMG_FileSystemAPI holds function pointers to the used File System APIs
	_IMG_PIXEL_XY_RGB_888	IMG_PixelRgb holds the RGB information of a pixel in BYTES
	IMG_FILE_SYSTEM_API	IMG_FileSystemAPI holds function pointers to the used File System APIs
	IMG_PIXEL_XY_RGB_888	IMG_PixelRgb holds the RGB information of a pixel in BYTES

Types

	Name	Description
	FileFeof	This is type FileFeof.
	FileRead	Typedefs of the used File System APIs
	FileSeek	This is type FileSeek.
	FileTell	This is type FileTell.
	IMG_LOOP_CALLBACK	IMG_LoopCallback is a callback function which is called in every loop of the decoding cycle so that user application can do some maintenance activities
	IMG_LOOP_FRAMECOMPLETE	IMG_FrameCompleteCallback is a callback function which is called at completion decoding current frame so that user application can do some activities
	IMG_LOOP_IMAGECOMPLETE	IMG_FrameCompleteCallback is a callback function which is called at completion decoding current frame so that user application can do some activities
	IMG_LOOP_IMAGEFAIL	IMG_FrameCompleteCallback is a callback function which is called at completion decoding current frame so that user application can do some activities
	IMG_PIXEL_OUTPUT	IMG_PixelOutput is a callback function which receives the color information of the output pixel

Description

Graphics Image Decoder

Refer to Microchip Graphics Library for complete documentation of the Image Decoder.

File Name

gfx_image_decoder.h

Company

Microchip Technology Inc.

gfx_primitive.h

Primitive Layer of the Microchip Graphics Library.

Enumerations

	Name	Description
	GFX_PRIMITIVE_STATES	Defines the various states that can be achieved by the PRIMITIVE HAL operation.

Functions

	Name	Description
	GFX_AlphaBlendingValueGet	This function returns the current alpha value set for alpha blending rendering.
	GFX_AlphaBlendingValueSet	This function sets the alpha value for alpha blending rendering.
	GFX_BackgroundColorGet	This function returns the color used in the current background.
	GFX_BackgroundImageGet	This function returns the image used in the current background.
	GFX_BackgroundImageLeftGet	This function returns the horizontal starting position of the current background.
	GFX_BackgroundImageTopGet	This function returns the vertical starting position of the current background.
	GFX_BackgroundSet	This function sets the background information.
	GFX_BackgroundTypeGet	This function returns the type of the current background.
	GFX_BackgroundTypeSet	This function sets the background type.
	GFX_BarDraw	This function renders a bar shape using the currently set fill style and color.
	GFX_CircleDraw	This function renders a circular shape using the currently set line style and color.
	GFX_CircleFillDraw	This function renders a filled circle shape using the currently set fill style and colors.
	GFX_ColorGet	This function returns the color currently set to render primitive shapes and text.
	GFX_ColorSet	This function sets the color to be used in rendering primitive shapes and text.

	GFX_DoubleBufferAreaGet	This function returns a rectangular area that needs synchronization specified by the given index.
	GFX_DoubleBufferAreaMark	This function adds the given rectangular area as an area that will be included in the list of areas for synchronization.
	GFX_DoubleBufferDisable	This function disables the double buffering feature of the graphics library.
	GFX_DoubleBufferEnable	This function enables the double buffering feature of the graphics library.
	GFX_DoubleBufferPause	This function pauses the double buffering feature of the graphics library.
	GFX_DoubleBufferResume	This function resumes the double buffering feature of the graphics library.
	GFX_DoubleBufferStatusGet	This function returns the current status of the double buffering feature of the graphics library.
	GFX_DoubleBufferSyncAllStatusClear	This function clears the synchronize all status.
	GFX_DoubleBufferSyncAllStatusGet	This function returns the status of the synchronize all flag.
	GFX_DoubleBufferSyncAllStatusSet	This function sets the whole draw buffer to be unsynchronized.
	GFX_DoubleBufferSyncAreaCountGet	This function returns the current count of rectangular areas that needs to be synchronized.
	GFX_DoubleBufferSyncAreaCountSet	This function sets the current count of rectangular areas that needs to be synchronized.
	GFX_DoubleBufferSynchronize	This function synchronizes the contents of the draw and frame buffer immediately.
	GFX_DoubleBufferSynchronizeRequest	This function requests synchronization of the contents of the draw and frame buffer.
	GFX_DoubleBufferSynchronizeStatusGet	This function returns the status of the synchronization request of the draw and frame buffer.
	GFX_DrawBufferGet	This function returns the index of the current draw buffer.
	GFX_DrawBufferInitialize	This function initializes the address of the draw buffer specified by the given index.
	GFX_DrawBufferSet	This function sets the draw buffer with the given index number.
	GFX_ExternalResourceCallback	This function performs data fetch from external memory.
	GFX_FillStyleGet	Return the fill style used when rendering filled shapes.
	GFX_FillStyleSet	Set the fill style to be used when rendering filled shapes.
	GFX_FontAntiAliasGet	This function returns the current font font anti-aliasing mode.
	GFX_FontAntiAliasSet	This function sets the font anti-aliasing mode.
	GFX_FontGet	This function returns the current font used when rendering strings and characters.
	GFX_FontSet	This function sets the current font used when rendering strings and characters.
	GFX_FrameBufferGet	This function returns the index of the current frame buffer.
	GFX_FrameBufferSet	This function sets the frame buffer with the given index number.
	GFX_GradientColorSet	Sets the gradient fill start and end color.
	GFX_GradientEndColorGet	Return the gradient end color when rendering gradient filled shapes.
	GFX_GradientStartColorGet	Return the gradient start color when rendering gradient filled shapes.
	GFX_ImageDraw	This function renders an image to the frame buffer.
	GFX_ImageHeaderGet	This function fills the given bitmap header with the image's header information.
	GFX_ImageHeightGet	This function returns the height of the given image.
	GFX_ImagePartialDraw	This function renders a portion of an image to the frame buffer.
	GFX_ImageWidthGet	This function returns the width of the given image.
	GFX_LineDraw	This function renders a line from x1,y1 to x2,y2 using the currently set line style (see GFX_LineStyleSet()).
	GFX_LinePositionRelativeSet	This function sets the line cursor to a new position relative to the current position.
	GFX_LinePositionSet	This function sets the line cursor to a new position.
	GFX_LinePositionXGet	This function returns the current x position of the line cursor.
	GFX_LinePositionYGet	This function returns the current y position of the line cursor.
	GFX_LineStyleGet	Return the line style used when rendering lines.
	GFX_LineStyleSet	Set the line style to be used when rendering lines.
	GFX_LineToDraw	This function renders a line from current line cursor position (x,y) to (x2,y2) using the currently set line style (see GFX_LineStyleSet()).
	GFX_LineToRelativeDraw	This function renders a line from current line cursor position (x,y) to (x+dX,y+dY) using the currently set line style (see GFX_LineStyleSet()).
	GFX_MaxXGet	This function returns the maximum horizontal pixel coordinate.
	GFX_MaxYGet	This function returns the maximum vertical pixel coordinate.
	GFX_PolygonDraw	This function renders a polygon using the currently set line style and color.
	GFX_PRIMITIVE_Close	Closes an instance of the primitive layer specified by handle.
	GFX_PRIMITIVE_Deinitialize	Deinitializes the specified instance of the GFX Module.

	GFX_PRIMITIVE_Initialize	Initialize the GFX PRIMITIVE HAL Library.
	GFX_PRIMITIVE_Open	Opens a client instance of the primitive layer specified by index.
	GFX_RectangleDraw	This function renders a rectangular shape using the currently set line style and color.
	GFX_RectangleFillDraw	This function renders a filled rectangular shape using the currently set fill style and colors.
	GFX_RectangleRoundDraw	This function renders a rounded corner rectangular shape using the currently set line style and color.
	GFX_RectangleRoundFillDraw	This function renders a filled rectangular shape with round corners using the currently set fill style and colors.
	GFX_ScreenClear	Clears the screen to the currently set color (GFX_ColorSet()).
	GFX_TextCharDraw	This function renders the given character using the currently set color using the currently set font.
	GFX_TextCursorDraw	This is function GFX_TextCursorDraw.
	GFX_TextCursorPositionSet	This function sets the text cursor to a new position.
	GFX_TextCursorPositionXGet	This function returns the current x position of the text cursor.
	GFX_TextCursorPositionYGet	This function returns the current y position of the text cursor.
	GFX_TextCursorTypeGet	This is function GFX_TextCursorTypeGet.
	GFX_TextCursorTypeSet	This is function GFX_TextCursorTypeSet.
	GFX_TextCursorWidthGet	This is function GFX_TextCursorWidthGet.
	GFX_TextStringBoxDraw	This function renders the given string using the currently set color and font into a rectangular area.
	GFX_TextStringDraw	This function renders the given string of character using the currently set color using the currently set font.
	GFX_TextStringHeightGet	This function returns the height of the given font.
	GFX_TextStringWidthGet	This function returns the width of the given string using the given font.
	GFX_TransparentColorDisable	This function disables the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.
	GFX_TransparentColorEnable	This function sets the transparent color used in GFX_ImageDraw() functions and enables the transparent color feature.
	GFX_TransparentColorGet	This returns the current transparent color set for the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.
	GFX_TransparentColorStatusGet	This returns the current state of the transparent color feature used in GFX_ImageDraw() and GFX_ImagePartialDraw() functions.

Macros

	Name	Description
	GFX_PRIMITIVE_HANDLE_INVALID	Constant that defines the value of an Invalid Device Handle.
	GFX_TEXT_CURSOR_HEIGHT_DEFAULT	Constant that defines the default value of text cursor height.
	GFX_TEXT_CURSOR_WIDTH_DEFAULT	Constant that defines the default value of text cursor width.

Types

	Name	Description
	GFX_PRIMITIVE_HANDLE	Data type for GFX PRIMITIVE HAL handle.

Description

Module for Microchip Graphics Library - Graphic Primitive Layer

This module implements the primitive rendering routines of the Microchip Graphics Library. The routines are software fallbacks for accelerated functions of the display driver and display controller. The API is compatible with any Display Driver that is compliant with the requirements of the Display Driver interface of the Graphics Library.

File Name

gfx_primitive.h

Company

Microchip Technology Inc.

gfx_types_font.h

This header file defines the types used in fonts for the Microchip Graphics Library.

Macros

	Name	Description
	GFX_FONT_SPACE	Font space section. The fonts can be located in psv (constant) or program space in PIC24/dsPIC MCUs. This define allows for switching of the pointer type used to access the font structure in memory See: GraphicsConfig.h for the application define.
	GFX_UXCHAR	This is macro GFX_UXCHAR.
	GFX_XCHAR	Configuration for the text character size.

Structures

	Name	Description
	GFX_FONT_GLYPH_ENTRY	The structure describing the glyph entry in fonts.
	GFX_FONT_GLYPH_ENTRY_EXTENDED	The structure describing the intended glyph entry in fonts.
	GFX_FONT_HEADER	The structure used to define the font header.

Description

Graphics Resource Types Header for Microchip Graphics Library

This header defines the different structures used for fonts resources in the Microchip Graphics Library.

File Name

gfx_types_font.h

Company

Microchip Technology Inc.

gfx_types_image.h

This header file defines the types used in images for the Microchip Graphics Library.

Structures

	Name	Description
	GFX_MCHP_BITMAP_HEADER	The structure used to define the Microchip bitmap header.

Types

	Name	Description
	int16_gfx_image_prog	This is type int16_gfx_image_prog.
	int16_prog	This is type int16_prog.
	int16_prog_pack	This is type int16_prog_pack.
	int32_gfx_image_prog	This is type int32_gfx_image_prog.
	int32_prog	This is type int32_prog.
	int32_prog_pack	This is type int32_prog_pack.
	int8_gfx_image_prog	This is type int8_gfx_image_prog.
	int8_prog	This is type int8_prog.
	int8_prog_pack	This is type int8_prog_pack.
	uint16_gfx_image_prog	This is type uint16_gfx_image_prog.
	uint16_prog	This is type uint16_prog.
	uint16_prog_pack	This is type uint16_prog_pack.
	uint32_gfx_image_prog	This is type uint32_gfx_image_prog.
	uint32_prog	This is type uint32_prog.
	uint32_prog_pack	This is type uint32_prog_pack.
	uint8_gfx_image_prog	GFX Image data Types
	uint8_prog	General program space data types
	uint8_prog_pack	This is type uint8_prog_pack.

Description

Graphics Resource Types Header for Microchip Graphics Library

This header defines the different structures used for images resources in the Microchip Graphics Library.

File Name

gfx_types_image.h

Company

Microchip Technology Inc.

gfx_types_macros.h

Data types used in Microchip Graphics Library.

Enumerations

	Name	Description
	GFX_ALIGNMENT	Summary of the different text alignment supported in the library.
	GFX_BACKGROUND_TYPE	Specifies the different background fill types.
	GFX_FEATURE_STATUS	States of a feature that can be enabled/disabled at run time.
	GFX_FILL_STYLE	Specifies the available fill styles.
	GFX_FONT_ANTIALIAS_TYPE	Summary of the transparency types in text anti-aliasing.
	GFX_STATUS	Rendering status.
	GFX_STATUS_BIT	Additional rendering status.
	GFX_TEXT_CURSOR_TYPE	Enumeration defines types of text cursor supported.

Structures

	Name	Description
	GFX_BACKGROUND	Background information structure.
	GFX_DOUBLE_BUFFERING_MODE	Structure used for double buffering management.
	GFX_PARTIAL_IMAGE_PARAM	Partial Image information structure.
	GFX_RECTANGULAR_AREA	A generic rectangular area structure.

Description

Module for Microchip Graphics Library

This module defines the data types used in the Microchip Graphics Library for use of Primitive and Driver Layers.

File Name

gfx_types_macros.h

Company

Microchip Technology Inc.

gfx_types_palette.h

This header file defines the types used in palettes for the Microchip Graphics Library.

Unions

	Name	Description
	GFX_PALETTE_ENTRY	Defines the union used for each entry in the palette table.

Description

Graphics Resource Types Header for Microchip Graphics Library

This header defines the different structures used for palette resources in the Microchip Graphics Library.

File Name

gfx_types_palette.h

Company

Microchip Technology Inc.

gfx_types_resource.h

This is the definition of the Graphics Resource Types used with the Microchip Graphics Library.

Enumerations

	Name	Description
	GFX_RESOURCE	Specifies the different resource types in the library.

Macros

	Name	Description
	GFX_COMP_MASK	This is macro GFX_COMP_MASK.
	GFX_MEM_MASK	This is macro GFX_MEM_MASK.
	GFX_TYPE_MASK	This is macro GFX_TYPE_MASK.
	MCHP_BITMAP_NORMAL	no compression, palette is present for color depth = 8, 4 and 1 BPP
	MCHP_BITMAP_PALETTE_STR	palette is provided as a separate object (see PALETTE_HEADER) for color depth = 8, 4, and 1 BPP, ID to the palette is embedded in the bitmap.

Structures

	Name	Description
	GFX_RESOURCE_BINARY	Defines the structure used for the binary type resource.
	GFX_RESOURCE_FONT	Defines the structure used for the font type resource.
	GFX_RESOURCE_HDR	Defines the structure used for the resource types.
	GFX_RESOURCE_IMAGE	Defines the structure used for the image type resource.
	GFX_RESOURCE_PALETTE	Defines the structure used for the palette type resource.

Description

Graphics Resource Types Header for Microchip Graphics Library

This header defines the different structures used for the Graphics Resources used in the Graphics Library. This also contains helpful macros as well as data types used.

File Name

gfx_types_resource.h

Company

Microchip Technology Inc.

gfx_config_template.h

This header file template defines all the Graphics Library configurations available.

Macros

	Name	Description
	GFX_CONFIG_ALPHABLEND_DISABLE	Macro that disables the Alpha Blending feature.
	GFX_CONFIG_COLOR_DEPTH	Macro that sets the color depth of the application.
	GFX_CONFIG_DOUBLE_BUFFERING_DISABLE	Macro disables the support for double buffering in rendering of pixels to the frame buffer.
	GFX_CONFIG_FOCUS_DISABLE	Macro that disables the focus feature in objects.
	GFX_CONFIG_FONT_ANTIALIASED_DISABLE	Macro that disables the use of anti-aliased fonts.
	GFX_CONFIG_FONT_CHAR_SIZE	Macro that sets the size of the characters used in the fonts.
	GFX_CONFIG_FONT_EXTERNAL_DISABLE	Macro that disables sourcing of font resources from external sources.
	GFX_CONFIG_FONT_FLASH_DISABLE	Macro that disables sourcing of font resources from internal flash memory.
	GFX_CONFIG_FONT_RAM_DISABLE	Macro that disables sourcing of font resources from RAM sources.
	GFX_CONFIG_GRADIENT_DISABLE	Macro that disables the gradient fill feature.
	GFX_CONFIG_IMAGE_EXTERNAL_DISABLE	Macro that disables sourcing of image resources from external sources.
	GFX_CONFIG_IMAGE_FLASH_DISABLE	Macro that disables sourcing of image resources from internal flash memory.
	GFX_CONFIG_IMAGE_PADDING_DISABLE	Macro disables the padding of bits on images converted by the Graphics Resource Converter (GRC).

	GFX_CONFIG_IMAGE_RAM_DISABLE	Macro that disables sourcing of image resources from RAM sources.
	GFX_CONFIG_IPU_DECODE_DISABLE	Macro that disables RLE compression of images.
	GFX_CONFIG_PALETTE_DISABLE	Macro that disables the palette feature.
	GFX_CONFIG_PALETTE_EXTERNAL_DISABLE	Macro that disables the palette feature that are sourced in external resources.
	GFX_CONFIG_RLE_DECODE_DISABLE	Macro that disables RLE compression of images.
	GFX_CONFIG_TRANSPARENT_COLOR_DISABLE	Macro that disables the use of anti-aliased fonts.
	GFX_CONFIG_USE_KEYBOARD_DISABLE	This macro disables the keyboard support in objects.
	GFX_CONFIG_USE_TOUCHSCREEN_DISABLE	This macro disables the resistive touchscreen support in objects.
	GFX_EXTERNAL_FONT_RASTER_BUFFER_SIZE	Macro sets the size of the external font raster buffer.
	GFX_free	Macro that defines the free function for versatility when using Operating Systems.
	GFX_INSTANCES_NUMBER	Macro to specify the number of possible client modules.
	GFX_malloc	Macro that defines the malloc function for versatility when using Operating Systems.

Description

Module for Microchip Graphics Library

File Name

graphics_config.h

Company

Microchip Technology Inc.

Math Libraries Help

This section provides descriptions of the Math libraries that are available in MPLAB Harmony.

DSP Fixed-Point Math Library

This topic describes the DSP Fixed-Point Math Library.

Introduction

The DSP Fixed-Point Library is available for the PIC32MZ family of microcontrollers. This library was created from optimized assembly routines written specifically for devices with microAptiv™ core features that utilize DSP ASE.

Description

The DSP Fixed-Point Library contains building block functions for developing digital signal processing algorithms. The library supports the Q15 and Q31 fractional data formats. The functions are implemented in efficient assembly specifically targeted at the DSP extensions in this core family. The library makes these functions available in a simple C-callable structure.

Functions included in the DSP Fixed-Point Library include complex math, vector math, matrix math, digital filters, and transforms. In many cases, these functions require specific data structures to operate, which are detailed in the header file and examples.

Using the Library

This topic describes the basic architecture of the DSP Fixed-Point Library and provides information and examples on its use.

Description

Interface Header File: `dsp.h`, `libq.h`

The interface to the DSP Fixed-Point library is defined in the `dsp.h` header file. Any C language source (`.c`) file that uses the DSP Fixed-Point library must include `dsp.h`. This file is automatically added to the project when 'use DSP library' is selected in MHC. In addition, this file name is added to the `#include` section of `system_definitions.h` after the configurations are completed through MHC.

Some functions make special use of the optimized fixed-point math library `libq.h`. For use of those functions, the `libq.h` file must also be included in a project. The `libq.h` file is also installed with MPLAB Harmony. Specific notes within each function will describe if the function is dependent on the LibQ Fixed-Point Math Library.

Library Files: `dsp_pic32mz.a`, `LIBQ_Library.X.a`, and `dsp_pic32mz_ef.a`

The DSP Fixed-Point library archive (`.a`) files are installed with MPLAB Harmony. Although there are two PIC32 DSP files, the linker will only utilize one of these depending on your device usage. If you have a device using a FPU, the `dsp_pic32mz_ef.a` file is used by the linker. Otherwise, the linker will utilize `dsp_pic32mz.a`. In a future release of the tools, the linker in the MPLAB XC32 C/C++ Compiler may be changed to do this automatically. Both of these files are added to a project when the 'use DSP library' option is selected in MHC.

This library is only available in binary form, with prototypes for each function described in the `dsp.h` file.

For functions that are supported by the LibQ Fixed-Point Math library, the `LIBQ_Library.X.a` library must also be installed. This library is also available in binary form and is installed with MPLAB Harmony. In some cases, the linker in the MPLAB XC32 C/C++ Compiler may not acknowledge the LibQ Fixed-Point Math Library file. If the library is included in the project, and the message *undefined reference* is encountered upon compilation, it may be necessary to add the specific assembly file function to your project. The source code for each function can be found in the `<install-dir>/framework/math/libq` folder of your MPLAB Harmony installation.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DSP Fixed-Point Math Library.

Library Interface Section	Description
Complex Math Functions	General mathematical operations using a complex structure with the form (a + bi)
Vector Math Functions	Mathematical operations on a array of numbers or vector
Matrix Math Functions	Mathematical operations on a matrix
Digital Filter Functions	FIR and IIR filtering functions with various architectures
Transform Functions	FFT, Windows and related transform elements
Support Functions	Quick support functions for numerical transform

The DSP Fixed-Point Library uses fixed-point fractional functions to optimize execution speed. These functions limit the accuracy of the

calculations to the bits specified for the function. Due to parallelism in some operations, the 16-bit version of the functions are more efficient than their 32-bit counterparts. In many cases both 16-bit and 32-bit functions are available to give the user the choice of balance between speed and functional resolution.

Fractional representation of a real number is given by:

Q*n.m*

where:

- *n* is the number of data bits to the left of the radix point
- *m* is the number of data bits to the right of the radix point
- a signed bit is implied, and takes one bit of resolution
- Shorthand may eliminate the leading 0, such as in Q0.15, which may be shortened to Q15, and similarly Q0.31, which is shortened to Q31

Qn.m numerical values are used by the library processing data as integers. In this format the n represents the number of integer bits, and the m represents the number of fractional bits. All values assume a sign bit in the most significant bit. Therefore, the range of the numerical value is:

$$-2^{(n-1)} \text{ to } [2^{(n-1)} - 2^{(-m)}]; \text{ with a resolution of } 2^{(-m)}.$$

A Q16 format number (Q15.16) would range from -32768.0 (0x8000 0000) to 32767.99998474 with a precision of 0.000015259 (or 2^{-16}).

For example, a numerical representation of the number 3.14159 in Q2.13 notation would be:

$$3.14159 * 2^{13} = 25735.9 \Rightarrow 0x6488$$

And converting from the Q7.8 format with the value 0x1D89 would be:

$$0x1D89 / 2^{8} = 7561 / 256 \Rightarrow 29.5316, \text{ accurate to } 0.00391$$

A majority of the DSP Fixed-Point Library uses functions with variables in Q15 or Q31 format. Representations of these numbers are given in **Data Types and Constants** in the Library Interface section, and generally are int16_t (for Q15 fractional representation) and int32_t (for Q31 fractional representation). This limits the equivalent numerical range to roughly -1.0 to 0.999999999. It is possible to represent other number ranges, but scaling before and after the function call are necessary.

All library functions will saturate the output if the value exceeds the maximum or is lower than the minimum allowable value for that resolution. Some prescaling may be necessary to prevent unwanted saturation in functions that may otherwise create calculation errors.

Table of Library Functions

Family	Function	Definition
Complex Math	Addition	DSP_ComplexAdd32
	Conjugate	DSP_ComplexConj32
	Dot Product	DSP_ComplexDotProd32
	Multiplication	DSP_ComplexMult32
	Scalar Multiplication	DSP_ComplexScalarMult32
	Subtraction	DSP_ComplexSub32
Digital Filter	FIR	DSP_FilterFIR32
	FIR Decimation	DSP_FilterFIRDecim32
	FIR Interpolation	DSP_FilterFIRInterp32
	FIR LMS	DSP_FilterLMS16
	IIR	DSP_FilterIIR16; DSP_FilterIIRSetup16
	IIR Biquad	DSP_FilterIIRBQ16; DSP_FilterIIRBQ16_fast; DSP_FilterIIRBQ32
	IIR Biquad Cascade	DSP_FilterIIRBQ16_cascade8; DSP_FilterIIRBQ16_cascade8_fast
	IIR Biquad Parallel	DSP_FilterIIRBQ16_parallel8; DSP_FilterIIRBQ16_parallel8_fast
Matrix Math	Addition	DSP_MatrixAdd32
	Equality	DSP_MatrixEqual32
	Initialization	DSP_MatrixInit32
	Multiplication	DSP_MatrixMul32
	Scale	DSP_MatrixScale32
	Subtraction	DSP_MatrixSub32
	Transpose	DSP_MatrixTranspose32
Transform	FFT	DSP_TransformFFT16; DSP_TransformFFT32
	Setup factors	DSP_TransformFFT16_setup; DSP_TransformFFT32_setup

	inverse FFT	DSP_TransformIFFT16
	Windows	DSP_TransformWindow_Bart16; DSP_TransformWindow_Bart32; DSP_TransformWindow_Black16; DSP_TransformWindow_Black32; DSP_TransformWindow_Cosine16; DSP_TransformWindow_Cosine32; DSP_TransformWindow_Hamm16; DSP_TransformWindow_Hamm32; DSP_TransformWindow_Hann16; DSP_TransformWindow_Hann32; DSP_TransformWindow_Kaiser16; DSP_TransformWindow_Kaiser32;
	Window Initialization	DSP_TransformWinInit_Bart16; DSP_TransformWinInit_Bart32; DSP_TransformWinInit_Black16; DSP_TransformWinInit_Black32; DSP_TransformWinInit_Cosine16; DSP_TransformWinInit_Cosine32; DSP_TransformWinInit_Hamm16; DSP_TransformWinInit_Hamm32; DSP_TransformWinInit_Hann16; DSP_TransformWinInit_Hann32; DSP_TransformWinInit_Kaiser16; DSP_TransformWinInit_Kaiser32
Vector Math	Absolute value	DSP_VectorAbs16; DSP_VectorAbs32
	Addition	DSP_VectorAdd16; DSP_VectorAdd32
	Addition w/ constant	DSP_VectorAddc16; DSP_VectorAddc32
	Binary exponent	DSP_VectorBexp16; DSP_VectorBexp32
	Log, Exponent, Square root	DSP_VectorExp; DSP_VectorLog10; DSP_VectorLog2; DSP_VectorLn; DSP_VectorSqrt;
	Multiplication & Division	DSP_VectorMul16; DSP_VectorMul32; DSP_VectorDotp16; DSP_VectorDotp32; DSP_VectorMulc16; DSP_VectorMulc32; DSP_VectorDivC; DSP_VectorRecip
	Subtraction	DSP_VectorSub16; DSP_VectorSub32
	Power (sum of squares)	DSP_VectorSumSquares16; DSP_VectorSumSquares32
	Equality check	DSP_VectorChkEqu32
	Maximum	DSP_VectorMax32; DSP_VectorMaxIndex32
	Minimum	DSP_VectorMin32; DSP_VectorMinIndex32
	Copy & Fill	DSP_VectorCopy; DSP_VectorFill; DSP_VectorZeroPad
	Shift	DSP_VectorShift; DSP_VectorCopyReverse32
	Negate	DSP_VectorNegate
	Statistics	DSP_VectorAutocorr16; DSP_VectorMean32; DSP_VectorRMS16; DSP_VectorStdDev16; DSP_VectorVari16; DSP_VectorVariance

Library Interface

a) Complex Math Functions

	Name	Description
	DSP_ComplexAdd32	Calculates the sum of two complex numbers.
	DSP_ComplexConj16	Calculates the complex conjugate of a complex number.
	DSP_ComplexConj32	Calculates the complex conjugate of a complex number.
	DSP_ComplexDotProd32	Calculates the dot product of two complex numbers.
	DSP_ComplexMult32	Multiplies two complex numbers.
	DSP_ComplexScalarMult32	Multiplies a complex number and a scalar number.
	DSP_ComplexSub32	Calculates the difference of two complex numbers.

b) Digital Filter Functions

	Name	Description
	DSP_FilterFIR32	Performs a Finite Infinite Response (FIR) filter on a vector.
	DSP_FilterFIRDecim32	Performs a decimating FIR filter on the input array.
	DSP_FilterFIRInterp32	Performs an interpolating FIR filter on the input array.
	DSP_FilterIIR16	Performs a single-sample cascaded biquad Infinite Impulse Response (IIR) filter.
	DSP_FilterIIRBQ16	Performs a single-pass IIR Biquad Filter.
	DSP_FilterIIRBQ16_cascade8	Performs a single-sample IIR Biquad Filter as a cascade of 8 series filters.
	DSP_FilterIIRBQ16_cascade8_fast	Performs a single-sample IIR Biquad Filter as a cascade of 8 series filters.
	DSP_FilterIIRBQ16_fast	Performs a single-pass IIR Biquad Filter.

	DSP_FilterIIRBQ16_parallel8	Performs a 8 parallel single-pass IIR Biquad Filters, and sums the result.
	DSP_FilterIIRBQ16_parallel8_fast	Performs a 8 parallel single-pass IIR Biquad Filters, and sums the result.
	DSP_FilterIIRBQ32	Performs a high resolution single-pass IIR Biquad Filter.
	DSP_FilterIIRSetup16	Converts biquad structure to coeffs array to set up IIR filter.
	DSP_FilterLMS16	Performs a single sample Least Mean Squares FIR Filter.

c) Matrix Math Functions

	Name	Description
	DSP_MatrixAdd32	Addition of two matrices C = (A + B).
	DSP_MatrixEqual32	Equality of two matrices C = (A).
	DSP_MatrixInit32	Initializes the first N elements of a Matrix to the value num.
	DSP_MatrixMul32	Multiplication of two matrices C = A x B.
	DSP_MatrixScale32	Scales each element of an input buffer (matrix) by a fixed number.
	DSP_MatrixSub32	Subtraction of two matrices C = (A - B).
	DSP_MatrixTranspose32	Transpose of a Matrix C = A (T).

d) Transform Functions

	Name	Description
	DSP_TransformFFT16	Creates an Fast Fourier Transform (FFT) from a time domain input.
	DSP_TransformFFT16_setup	Creates FFT coefficients for use in the FFT16 function.
	DSP_TransformFFT32	Creates an Fast Fourier Transform (FFT) from a time domain input.
	DSP_TransformFFT32_setup	Creates FFT coefficients for use in the FFT32 function.
	DSP_TransformIFFT16	Creates an Inverse Fast Fourier Transform (FFT) from a frequency domain input.
	DSP_TransformWindow_Bart16	Perform a Bartlett window on a vector.
	DSP_TransformWindow_Bart32	Perform a Bartlett window on a vector.
	DSP_TransformWindow_Black16	Perform a Blackman window on a vector.
	DSP_TransformWindow_Black32	Perform a Blackman window on a vector.
	DSP_TransformWindow_Cosine16	Perform a Cosine (Sine) window on a vector.
	DSP_TransformWindow_Cosine32	Perform a Cosine (Sine) window on a vector.
	DSP_TransformWindow_Hamm16	Perform a Hamming window on a vector.
	DSP_TransformWindow_Hamm32	Perform a Hamming window on a vector.
	DSP_TransformWindow_Hann16	Perform a Hanning window on a vector.
	DSP_TransformWindow_Hann32	Perform a Hanning window on a vector.
	DSP_TransformWindow_Kaiser16	Perform a Kaiser window on a vector.
	DSP_TransformWindow_Kaiser32	Perform a Kaiser window on a vector.
	DSP_TransformWinInit_Bart16	Create a Bartlett window.
	DSP_TransformWinInit_Bart32	Create a Bartlett window.
	DSP_TransformWinInit_Black16	Create a Blackman window.
	DSP_TransformWinInit_Black32	Create a Blackman window.
	DSP_TransformWinInit_Cosine16	Create a Cosine (Sine) window.
	DSP_TransformWinInit_Cosine32	Create a Cosine (Sine) window.
	DSP_TransformWinInit_Hamm16	Create a Hamming window.
	DSP_TransformWinInit_Hamm32	Create a Hamming window.
	DSP_TransformWinInit_Hann16	Create a Hanning window.
	DSP_TransformWinInit_Hann32	Create a Hanning window.
	DSP_TransformWinInit_Kaiser16	Create a Kaiser window.
	DSP_TransformWinInit_Kaiser32	Create a Kaiser window.

e) Vector Math Functions

	Name	Description
	DSP_VectorAbs16	Calculate the absolute value of a vector.
	DSP_VectorAbs32	Calculate the absolute value of a vector.
	DSP_VectorAdd16	Calculate the sum of two vectors.
	DSP_VectorAdd32	Calculate the sum of two vectors.
	DSP_VectorAddc16	Calculate the sum of a vector and a constant.

	DSP_VectorAddc32	Calculate the sum of a vector and a constant.
	DSP_VectorAutocorr16	Computes the Autocorrelation of a Vector.
	DSP_VectorBexp16	Computes the maximum binary exponent of a vector.
	DSP_VectorBexp32	Computes the maximum binary exponent of a vector.
	DSP_VectorChkEqu32	Compares two input vectors, returns an integer '1' if equal, and '0' if not equal.
	DSP_VectorCopy	Copies the elements of one vector to another.
	DSP_VectorCopyReverse32	Reverses the order of elements in one vector and copies them into another.
	DSP_VectorDivC	Divides the first N elements of inVector by a constant divisor, and stores the result in outVector.
	DSP_VectorDotp16	Computes the dot product of two vectors, and scales the output by a binary factor.
	DSP_VectorDotp32	Computes the dot product of two vectors, and scales the output by a binary factor
	DSP_VectorExp	Computes the EXP (e^x) of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorFill	Fills an input vector with scalar data.
	DSP_VectorLn	Computes the Natural Log, Ln(x), of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorLog10	Computes the Log10(x), of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorLog2	Computes the Log2(x) of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorMax32	Returns the maximum value of a vector.
	DSP_VectorMaxIndex32	Returns the index of the maximum value of a vector.
	DSP_VectorMean32	Calculates the mean average of an input vector.
	DSP_VectorMin32	Returns the minimum value of a vector.
	DSP_VectorMinIndex32	Returns the index of the minimum value of a vector.
	DSP_VectorMul16	Multiplication of a series of numbers in one vector to another vector.
	DSP_VectorMul32	Multiplication of a series of numbers in one vector to another vector.
	DSP_VectorMulc16	Multiplication of a series of numbers in one vector to a scalar value.
	DSP_VectorMulc32	Multiplication of a series of numbers in one vector to a scalar value.
	DSP_VectorNegate	Inverses the sign (negates) the elements of a vector.
	DSP_VectorRecip	Computes the reciprocal (1/x) of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorRMS16	Computes the root mean square (RMS) value of a vector.
	DSP_VectorShift	Shifts the data index of an input data vector.
	DSP_VectorSqrt	Computes the square root of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorStdDev16	Computes the Standard Deviation of a Vector.
	DSP_VectorSub16	Calculate the difference of two vectors.
	DSP_VectorSub32	Calculate the difference of two vectors.
	DSP_VectorSumSquares16	Computes the sum of squares of a vector, and scales the output by a binary factor.
	DSP_VectorSumSquares32	Computes the sum of squares of a vector, and scales the output by a binary factor.
	DSP_VectorVari16	Computes the variance of N elements of a Vector.
	DSP_VectorVariance	Computes the variance of N elements of inVector.
	DSP_VectorZeroPad	Fills an input vector with zeros.

f) Support Functions

	Name	Description
	mul16	multiply and shift integer
	mul16r	multiply and shift Q15
	mul32	multiply and shift Q31
	SAT16	saturate both positive and negative Q15
	SAT16N	saturate negative Q15
	SAT16P	saturate positive Q15

g) Data Types and Constants

	Name	Description
	biquad16	Q15 biquad
	int16c	Q15 complex number (a + bi)
	int32c	Q31 complex number (a + bi)

	matrix32	Q31 matrix
	PARM_EQUAL_FILTER	IIR BQ filter structure Q15 data, Q31 storage
	PARM_EQUAL_FILTER_16	IIR BQ filter structure Q15
	PARM_EQUAL_FILTER_32	IIR BQ filter structure Q31
	PARM_FILTER_GAIN	filter gain structure
	_PARM_EQUAL_FILTER	IIR BQ filter structure Q15 data, Q31 storage
	_PARM_EQUAL_FILTER_16	IIR BQ filter structure Q15
	_PARM_EQUAL_FILTER_32	IIR BQ filter structure Q31
	MAX16	maximum Q15
	MAX32	maximum Q31
	MIN16	minimum Q15
	MIN32	minimum Q31

Description

This section describes the Application Programming Interface (API) functions of the DSP Fixed-Point Library.

Refer to each section for a detailed description.

a) Complex Math Functions

DSP_ComplexAdd32 Function

Calculates the sum of two complex numbers.

File

dsp.h

C

```
void DSP_ComplexAdd32(int32c * indata1, int32c * indata2, int32c * Output);
```

Returns

pointer to result complex numbers (int32c)

None.

Description

Function DSP_ComplexAdd32:

void DSP_ComplexAdd32(int32c *indata1, int32c *indata2, int32c *Output);

Calculates the sum of two complex numbers, indata1 and indata2, and stores the complex result in Output. Complex numbers must be in the structural form that includes real and imaginary components. The function saturates the output values if maximum or minimum are exceeded. All values are in Q31 fractional data format. (a + bi) + (c + di) => (a + c) + (b + d)i

Remarks

None.

Preconditions

Complex numbers must be in the int32c format.

Example

```
int32c *res, result;
int32c *input1, *input2;
int32c test_complex_1 = {0x40000000,0x0CCCCCCC};
//                  (0.5 + 0.1i)
int32c test_complex_2 = {0x73333333,0xB3333334};
//                  (0.9 - 0.6i)
res=&result;
input1=&test_complex_1;
input2=&test_complex_2;

DSP_ComplexAdd32(input1, input2, res);

// result = {0x73333333, 0xC0000000} = (0.9 - 0.5i)
```

Parameters

Parameters	Description
indata1	pointer to input complex number (int32c)
indata2	pointer to input complex number (int32c)

DSP_ComplexConj16 Function

Calculates the complex conjugate of a complex number.

File

dsp.h

C

```
void DSP_ComplexConj16(int16c * indata, int16c * Output);
```

Returns

pointer to result complex numbers (int16c)

None.

Description

DSP_ComplexConj16:

void DSP_ComplexConj16(int16c *indata, int16c *Output);

CCalculates the complex conjugate of Indata, and stores the result in Outdata. Both numbers must be in the complex number data structure which includes real and imaginary components. Values are in Q15 fractional data format. The function will saturate the output if maximum or minimum values are exceeded. (a + bi) => (a - bi)

Remarks

None.

Preconditions

Complex numbers must be in the int32c format.

Example

```
int16c *res, result;
int16c *input1;
int16c test_complex_1 = {0x4000,0x0CCC};
//                  (0.5 + 0.1i)

res=&result;
input1=&test_complex_1;

DSP_ComplexConj16(input1, res);

// result = {0x4000, 0xF334} = (0.5 - 0.1i)
```

Parameters

Parameters	Description
indata	pointer to input complex number (int16c)

DSP_ComplexConj32 Function

Calculates the complex conjugate of a complex number.

File

dsp.h

C

```
void DSP_ComplexConj32(int32c * indata, int32c * Output);
```

Returns

pointer to result complex numbers (int32c)

None.

Description

Function DSP_ComplexConj32:

void DSP_ComplexConj32(int32c *indata, int32c *Output);

Calculates the complex conjugate of indata, and stores the result in Output. Both numbers must be in the complex number data structure, which includes real and imaginary components. Values are in Q31 fractional data format. The function will saturate the output if maximum or minimum values are exceeded. (a + bi) => (a - bi)

Remarks

None.

Preconditions

Complex numbers must be in the int32c format.

Example

```
int32c *res, result;
int32c *input1;
int32c test_complex_1 = {0x40000000,0x0CCCCCCC};
//                  (0.5 + 0.1i)

res=&result;
input1=&test_complex_1;

DSP_ComplexConj32(input1, res);

// result = {0x40000000, 0xF3333334} = (0.5 - 0.1i)
```

Parameters

Parameters	Description
indata1	pointer to input complex number (int32c)

DSP_ComplexDotProd32 Function

Calculates the dot product of two complex numbers.

File

dsp.h

C

```
void DSP_ComplexDotProd32(int32c * indata1, int32c * indata2, int32c * Output);
```

Returns

pointer to result complex numbers (int32c)

None.

Description

Function DSP_ComplexDotProd32:

void DSP_ComplexDotProd32(int32c *indata1, int32c *indata2, int32c *Output);

Calculates the dot product of two complex numbers, indata1 and indata2, and stores the result in Output. All numbers must be in complex structural format that includes real and imaginary components, and the numbers are in fractional Q31 format. The function will saturate the output if it exceeds maximum or minimum ratings. The formula for the dot product is as follows: Output(real) = (Input1.re * Input2.re) + (Input1.im * Input2.im); Output(img) = [(Input1.re * Input2.im) - (Input1.im * Input2.re)]i (a + bi) dot (c + di) => (a * c + b * d) + (a * d - b * c)i

Remarks

None.

Preconditions

Complex numbers must be in the int32c format.

Example

```
int32c *res, result;
int32c *input1, *input2;
int32c test_complex_1 = {0x40000000,0x0CCCCCCC};
//                  (0.5 + 0.1i)
int32c test_complex_2 = {0x73333333,0xB3333334};
//                  (0.9 - 0.6i)
res=&result;
input1=&test_complex_1;
input2=&test_complex_2;

DSP_ComplexDotProd32(input1, input2, res);

// result = {0x31EB851E, 0xCE147AE3} = (0.39 - 0.39i)
```

Parameters

Parameters	Description
indata1	pointer to input complex number (int32c)
indata2	pointer to input complex number (int32c)

DSP_ComplexMult32 Function

Multiplies two complex numbers.

File

dsp.h

C

```
void DSP_ComplexMult32(int32c * indata1, int32c * indata2, int32c * Output);
```

Returns

pointer to result complex numbers (int32c)

None.

Description

Function DSP_ComplexMult32:

void DSP_ComplexMult32(int32c *indata1, int32c *indata2, int32c *Output);

Multiplies two complex numbers, indata1 and indata2, and stores the complex result in Output. All numbers must be in the int32c complex data structure. All data is in Q31 fractional format. The function will saturate if maximum or minimum values are exceeded. Output(real) = (Input1.re * Input2.re) - (Input1.im * Input2.im); Output(img) = [(Input1.re * Input2.im) + (Input1.im * Input2.re)]i (a + bi) x (c + di) => (a * c - b * d) + (a * d + b * c)i

Remarks

None.

Preconditions

Complex numbers must be in the int32c format.

Example

```
int32c *res, result;
int32c *input1, *input2;
int32c test_complex_1 = {0x40000000,0x0CCCCCCC};
//                  (0.5 + 0.1i)
int32c test_complex_2 = {0x73333333,0xB3333334};
//                  (0.9 - 0.6i)
res=&result;
input1=&test_complex_1;
input2=&test_complex_2;

DSP_ComplexMult32(input1, input2, res);

// result = {0x4147AE14, 0xE51EB8551} = (0.51 - 0.21i)
```

Parameters

Parameters	Description
indata1	pointer to input complex number (int32c)
indata2	pointer to input complex number (int32c)

DSP_ComplexScalarMult32 Function

Multiplies a complex number and a scalar number.

File

dsp.h

C

```
void DSP_ComplexScalarMult32(int32c * indata, int32_t Scalar, int32c * Output);
```

Returns

pointer to result complex numbers (int32c)

None.

Description

Function DSP_ComplexScalarMult32:

void DSP_ComplexScalarMult32(int32c *indata, int32_t Scalar, int32c *Output);

Multiplies a complex number, indata, by a scalar number, Scalar, and stores the result in Output. indata and Output must be in int32c structure with real and imaginary components. All data must be in the fractional Q31 format. The function will saturate if maximum or minimum values are exceeded. Output(real) = (Input1.re * Scalar); Output(img) = [(Input1.im * Scalar)]i (a + bi) * C => (a * C + b * Ci)

Remarks

None.

Preconditions

Complex numbers must be in the int32c format.

Example

```
int32c *res, result;
int32c *input1;
int32_t  scalarInput = 0x20000000; // 0.25
int32c test_complex_1 = {0x40000000,0x0CCCCCCC};
//                  (0.5 + 0.1i)

res=&result;
input1=&test_complex_1;

DSP_ComplexScalarMult32(input1, scalarInput, res);

// result = {0x10000000, 0x03333333} = (0.125 + 0.025i)
```

Parameters

Parameters	Description
indata	pointer to input complex number (int32c)
Scalar	fractional scalar input value (int32_t)

DSP_ComplexSub32 Function

Calculates the difference of two complex numbers.

File

dsp.h

C

```
void DSP_ComplexSub32(int32c * indata1, int32c * indata2, int32c * Output);
```

Returns

pointer to result complex numbers (int32c)

None.

Description

Function DSP_ComplexSub32:

void DSP_ComplexSub32(int32c *indata1, int32c *indata2, int32c *Output);

Calculates the difference of two complex numbers, indata1 less indata2, and stores the complex result in Output. Both numbers must be in a complex data structure, which includes real and imaginary components. The function saturates the output values if maximum or minimum are exceeded. Real and imaginary components are in the Q31 fractional data format. (a + bi) - (c + di) => (a - c) + (b - d)i

Remarks

None.

Preconditions

Complex numbers must be in the int32c format.

Example

```
int32c *res, result;
int32c *input1, *input2;
int32c test_complex_1 = {0x40000000,0x0CCCCCCC};
//                  (0.5 + 0.1i)
int32c test_complex_2 = {0x73333333,0xB3333334};
//                  (0.9 - 0.6i)
res=&result;
input1=&test_complex_1;
input2=&test_complex_2;

DSP_ComplexSub32(input1, input2, res);

// result = {0xCCCCCCCD, 0x59999998} = (-0.4 + 0.7i)
```

Parameters

Parameters	Description
indata1	pointer to input complex number (int32c)
indata2	pointer to input complex number (int32c)

b) Digital Filter Functions

DSP_FilterFIR32 Function

Performs a Finite Infinite Response (FIR) filter on a vector.

File

dsp.h

C

```
void DSP_FilterFIR32(int32_t * outdata, int32_t * indata, int32_t * coeffs2x, int32_t * delayline, int N,
int K, int scale);
```

Returns

None.

Description

Function DSP_FilterFIR32:

void DSP_FilterFIR32(int32_t *outdata, int32_t *indata, int32_t *coeffs2x, int32_t *delayline, int N, int K, int scale);

Performs an FIR filter on the vector indata, and stores the output in the vector outdata. The number of samples processed in the array is given by N. The number of filter taps is given by K. The values are scaled upon input by the binary scaling factor (right shift), scale. The array of 2*K coefficients is contained in the array coeffs2x, where the values are in order b0, b1, b2... and repeated. Lastly the delayline is an array of K values that are initialized to zero and represent previous values. All values are in fractional Q31 data format. The function will saturate results if minimum

or maximum values are exceeded.

Remarks

Filter coefs must be repeated within the array. The array is twice as large as the number of taps, and the values are repeated in order b0, b1, b2,...bn, b0, b1, b2,... bn. The function updates the delayline array, which must be K elements long. The array should be initialized to zero prior to the first processing. It will contain values for processing cascaded filters within a loop.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four. K must be greater than 2 and a multiple of 2. delayline must have K elements, and be initialized to zero. coeffs2x must have 2*K elements.

Example

```
#define TAPS 4
#define numPOINTS 256

int filterN = numPOINTS;
int filterK = TAPS;
int filterScale = 1;  // scale output by 1/2^1 => output * 0.5
int32_t FilterCoefs[TAPS*2] = {0x40000000, 0x20000000, 0x20000000, 0x20000000,
            0x40000000, 0x20000000, 0x20000000, 0x20000000};
// note repeated filter coefs, A B C D A B C D
//             0.5, 0.25, 0.25, 0.25, 0.5, 0.25, 0.25, 0.25

int32_t outFilterData[numPOINTS]={0};
int32_t inFilterData[numPOINTS];
int filterDelayLine[TAPS]={0};

while(true)
{

//   put some data into input array, inFilterData, here //

DSP_FilterFIR32(outFilterData, inFilterData, FilterCoefs, filterDelayLine,
            filterN, filterK, filterScale);
}
```

Parameters

Parameters	Description
outdata	pointer to destination array of elements (int32_t)
indata	pointer to source array of elements (int32_t)
coeffs2x	pointer to an array of coefficients (int32_t)
delayline	pointer to an array of delay variables (int32_t)
N	number of points in the array to process (int) number of samples (int)
K	number of filter taps
scale	binary scaler divisor (1 / 2^scale) (int)

DSP_FilterFIRDecim32 Function

Performs a decimating FIR filter on the input array.

File

dsp.h

C

```
void DSP_FilterFIRDecim32(int32_t * outdata, int32_t * indata, int32_t * coeffs, int32_t * delayline, int
N, int K, int scale, int rate);
```

Returns

None.

Description

Function DSP_FilterFIRDecim32:

void DSP_FilterFIRDecim32(int32_t *outdata, int32_t *indata, int32_t *coeffs, int32_t *delayline, int N, int K, int scale, int rate);

Compute a FIR decimation filter on the input vector indata, and store the results to the vector outdata. The total number of output elements is set by N, and therefore the outdata array must be at least N in length. The decimation ratio is given by rate. The input is sampled every integer value of rate, skipping every (rate-1) input samples. The input array must therefore be (rate*N) samples long. The amount of filter taps is specified by K. Coeffs specifies the coefficients array. The delayline array holds delay inputs for calculation, and must be initialized to zero prior to calling the filter. Both coeffs and delayline must be K in length. Scale divides the input by a scaling factor by right shifting the number of bits (1/2^scale). All values of input, output, and coeffs are given in Q31 fractional data format. The function will saturate if the output value exceeds the maximum or minimum value.

Y = b0 * X0 + (b1 * X(-1)) + (b2 * X(-2)

Remarks

Coefs are loaded into the array with corresponding to the least delay first (C0, C(-1), C(-2)). K must be greater than rate. Even while decimating the input stream, every input passes through the delayline. So FIR filters of arbitrary length will give the same output as a non-decimating FIR, just with fewer responses.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. delayline must have K elements, and be initialized to zero. coeffs must have K elements. outdata must have N elements indata must have (N*rate) elements

Example

```
#define N 8         // number of output samples
#define TAPS 5
#define SKIP 3

int testFilterN = N;        // number of output elements
int testFilterK = TAPS;     // number of taps
int testFilterRate = SKIP;  // decimation rate R
int32_t outFiltDataDec[N]={0};
int32_t *inTestFilter[N*SKIP];
int filtScaleNum = 1;       // scale output (1 /2^n) => Y * 0.5

int32_t filtDelayTest[8]={0};  // always initialize to zero

// get pointer to input buffer here //
inTestFilter = &inputBuffer;

DSP_FilterFIRDecim32(outFiltDataDec, inTestFilter, inTestCoefs,
        filtDelayTest, testFilterN, testFilterK, filtScaleNum, testFilterRate);
```

Parameters

Parameters	Description
outdata	pointer to output array of elements (int32_t)
indata	pointer to input array of elements (int32_t)
coeffs	pointer to an array of coefficients (int32_t)
delayline	pointer to an array of delay variables (int32_t)
N	number of output elements to be processed (int)
K	number of filter taps and coeffs (int)
scale	binary scaler divisor (1 / 2^scale) (int)
rate	decimation ratio (int)

DSP_FilterFIRInterp32 Function

Performs an interpolating FIR filter on the input array.

File

dsp.h

C

```
void DSP_FilterFIRInterp32(int32_t * outdata, int32_t * indata, int32_t * coeffs, int32_t * delayline, int
N, int K, int scale, int rate);
```

Returns

None.

Description

Function DSP_FilterFIRInterp32:

void DSP_FilterFIRInterp32(int32_t *outdata, int32_t *indata, int32_t *coeffs, int32_t *delayline, int N, int K, int scale, int rate);

Perform an interpolating FIR filter on the first N samples of indata, and stores the result in outdata. The number of output elements is N*rate. The number of filter taps, K, must be an even multiple of N. The coefficients array, Coeffs, must be K elements long. The delay line array, delayline, must be K/R elements long, and be initialized to zero. All data elements must be in Q31 fractional data format. Scaling is performed via binary shift on the input equivalent to (1/2^shift). The function will saturate the output if it exceeds maximum or minimum values. The function creates R output values for each input value processed. The delayline of previous values is processed with R elements of the coefficient array. Numerically:

Y(1,0) = X(0)*C(0) + X(-1)*C(rate) + X(-2)*C(2*rate) ... Y(1,1) = X(0)*C(1) + X(-1)*C(rate+1) + X(-2)*C(2*rate + 1) ... Y(1,rate) = X(0)*C(N) + X(-1)*C(rate+N) + X(-2)*C(2*rate + N) ...

where output Y corresponds to (input,rate) different outputs, input X has (M/rate) sample delays and C is the coefficient array.

Remarks

The function processes each input (rate) times. With each pass, coefficients are offset so that (K/rate) multiply accumulate cycles occur.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. delayline must have (K/R) elements, and be initialized to zero. K (taps) must be an even multiple of R (rate). outdata must have R*N elements.

Example

```
// interpret evenly 1/3 spaced values
#define N 4         // number of output samples
#define TAPS 6
#define INTERP 3

int ifiltN = N;
int ifiltK = TAPS;  // k must be an even multiple of R
int ifiltR = INTERP;

int32_t ifiltOut[N*INTERP]={0};
int32_t ifiltDelay[2]={0};  // must be initialized to zero
int ifiltScale = 0;         // no scaling

int32_t ifiltCoefsThirds[TAPS]={0x2AAAAAA9, 0x55555555,0x7FFFFFFE,
                                0x55555555,0x2AAAAAA9,0x00000000};
//                  0.333333, 0.6666667, 0.99999999, 0.6666667, 0.33333333, 0

int32_t ifiltInput[N]={0x0CCCCCCD, 0x19999999, 0x26666666, 0x33333333};
//                  0.1,  0.2,  0.3,  0.4

DSP_FilterFIRInterp32(ifiltOut, ifiltInput, ifiltCoefsThirds, ifiltDelay,
                                ifiltN, ifiltK, ifiltScale, ifiltR);

// ifiltOut = {0x04444444, 0x08888889, 0x0CCCCCCD, 0x11111111, 0x15555555, 0x19999999,
//          0x1DDDDDDD, 0x22222221, 0x26666665, 0x2AAAAAAA,0x2EEEEEEE, 0x33333332}
// = 0.0333, 0.0667, 0.1, 0.1333, 0.1667, 0.2, 0.2333, 0.2667, 0.3, 0.3333, 0.3667, 0.4
```

Parameters

Parameters	Description
outdata	pointer to output array of elements (int32_t)
indata	pointer to input array of elements (int32_t)
coeffs	pointer to an array of coefficients (int32_t)
delayline	pointer to an array of delay variables (int32_t)
N	number of output elements to be processed (int)
K	number of filter taps and coeffs (int)
scale	binary scaler divisor (1 / 2^scale) (int)
rate	decimation ratio (int)

DSP_FilterIIR16 Function

Performs a single-sample cascaded biquad Infinite Impulse Response (IIR) filter.

File

dsp.h

C

```
int16_t DSP_FilterIIR16(int16_t in, int16_t * coeffs, int16_t * delayline, int B, int scale);
```

Returns

Sample output Y (int16_t)

Description

Function DSP_FilterIIR16:

int16_t DSP_FilterIIR16(int16_t in, int16_t *coeffs, int16_t *delayline, int B, int scale);

Performs a single element cascaded biquad IIR filter on the input, in. The filter contains B number of biquad sections, and cascades the output of one to the input of the next. B must be greater than 2 and a multiple of 2. The int16_t output generated by the function is the computation from the final biquad stage. Delay pipeline array delayline must contain 2*B values and be initialized to zero prior to use. The coefficient array must contain 4*B elements, and must be set up in order of biquad a1, a2, b1, b2. A binary (right shift) factor, scale, will scale the output equivalent to (1/2^scale). All numerical values must be in Q15 fractional data format. The function will saturate values if maximum or minimum values are exceeded.

Y = X0 + (b1 * X(-1)) + (b2 * X(-2) + (a1 * Y(-1)) + (a2 * Y(-2))

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. B must be greater than 2 and a multiple of 2. delayline must have 2*B elements, and be initialized to zero. coeffs must have 4*B elements.

Example

```
#define B 8         // use * biquad filters in cascade

int dataSamples = 256;
int i, j;
biquad16 bquad[B];
int16_t coefs[4*B]= {0};
int16_t delaylines[2*B]= {0};
int16_t Y, X;
int scaleBquad = 1;  // scale output (1 /2^n) => Y * 0.5

// do something to set up coefs, for instance this example //

for (j=0; jRemarks:Filter coefs must be stored within the array as a1, a2, b1, b2, a1, a2, b1, b2, in order
of biquads form input to output. A function to translate the coeffs from biquad structure to coeffs is
available in DSP_FilterIIRSetup16. The function updates the delayline array, which must be 2*B elements
long. The array should be initialized to zero prior to the first processing. It will contain values for
processing cascaded filters within a loop.
```

Parameters

Parameters	Description
in	input data element X (int16_t)
coeffs	pointer to an array of coefficients (int16_t)
delayline	pointer to an array of delay variables (int16_t)
B	number of cascaded biquad filter groups to process (int)
scale	binary scaler divisor (1 / 2^scale) (int)

DSP_FilterIIRBQ16 Function

Performs a single-pass IIR Biquad Filter.

File

dsp.h

C

```
int16_t DSP_FilterIIRBQ16(int16_t Xin, PARM_EQUAL_FILTER * pFilter);
```

Returns

Sample output Y (int16_t)

Description

Function DSP_FilterIIRBQ16:

int16_t DSP_FilterIIRBQ16(int16_t Xin, PARM_EQUAL_FILTER *pFilter);

Calculates a single pass IIR biquad filter on Xin, and delivers the result as a 16-bit output. All math is performed using 32 bit instructions, with results truncated to 16-bits for the output. The delay register is stored as a 32-bit value for subsequent functions. All values are fractional Q15 and Q31, see data structure for specifics.

Y = X(0)*b0 + (b1 * X(-1)) + (b2 * X(-2)) - (a1 * Y(-1)) - (a2 * Y(-2))

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Preconditions

Delay register values should be initialized to zero.

Example

```
PARM_EQUAL_FILTER *ptrFilterEQ;
PARM_EQUAL_FILTER FilterEQ;
uint16_t DataIn, DataOut;
ptrFilterEQ = &FilterEQ;

// 48KHz sampling; 1 KHz bandpass filter; Q=0.9
//  divide by 2 and convert to Q15
//   b0 = 0.06761171785499065
//   b1 = 0
//   b2 = -0.06761171785499065
//   a1 = -1.848823142275648
//   a2 = 0.8647765642900187

// note all coefs are half value of original design, gain handled in algorithm
ptrFiltEQ32->b[0]=0x0453;       // feed forward b0 coef
ptrFiltEQ32->b[1]=0;            // feed forward b1 coef
ptrFiltEQ32->b[2]=0xFBAD;       // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
ptrFiltEQ32->a[0]=0x89AD;       // feedback a1 coef
ptrFiltEQ32->a[1]=0x3758;       // feedback a2 coef

for (i=0;i<256;i++)
    {
    // *** get some input data here
       DataIn32 = three_hundred_hz[i];

       DataOut = DSP_FilterIIRBQ16(DataIn, ptrFilterEQ);

     // *** do something with the DataOut here
     }
```

Parameters

Parameters	Description
Xin	input data element X (int16_t)
pFilter	pointer to filter coef and delay structure

DSP_FilterIIRBQ16_cascade8 Function

Performs a single-sample IIR Biquad Filter as a cascade of 8 series filters.

File

dsp.h

C

```
int16_t DSP_FilterIIRBQ16_cascade8(int16_t Xin, PARM_EQUAL_FILTER * pFilter_Array);
```

Returns

Sample output Y (int16_t)

Description

Function DSP_FilterIIRBQ16_cascade8:

int16_t DSP_FilterIIRBQ16_cascade8(int16_t Xin, PARM_EQUAL_FILTER *pFilter_Array);

Calculates a single pass IIR biquad cascade filter on Xin, and delivers the result as a 16-bit output. The cascade of filters is 8 unique biquad filters arranged in series such that the output of one is provided as the input to the next. A unique filter coefficient set is provided to each, and 32 bit delay lines are maintained for each. All math is performed using 32 bit instructions, which results truncated to 16-bits for the output. Global gain values are available on the output. Fracgain is a Q15 fractional gain value and expgain is a binary shift gain value. The combination of the two can be utilized to normalize the output as desired. All values are fractional Q15 and Q31, see data structure for specifics.

Y = Y7 <- Y6 <- Y5 <- Y4 <- Y3 <- Y2 <- Y1 <- Y0 where each Yn filter element represents a unique IIR biquad: Yn = Y(n-1)*b0 + (b1 * Y(n-2)) + (b2 * Y(n-3)) - (a1 * Yn(-1)) - (a2 * Yn(-2)) and: for Y0; Y(n-1) = Xin(0)

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Preconditions

Delay register values should be initialized to zero.

Example

```
PARM_EQUAL_FILTER filtArray[8];
uint16_t dataY, dataX;

// example to use 2 filter blocks as notch filters
// fill entire Filter Array with coefs
for (i=0;i<8;i++)
{
    filtArray[i].Z[0]=0;
    filtArray[i].Z[1]=0;

    // note all coefs are half value of original design, gain handled in algorithm
    // all pass
    filtArray[i].b[0]=0x4000;
    filtArray[i].b[1]=0;        // feed forward b1 coef
    filtArray[i].b[2]=0;  // feed forward b2 coef

    filtArray[i].a[0]=0;  // feedback a1 coef
    filtArray[i].a[1]=0;  // feedback a2 coef
}
    // Unique filters for example
    // 10KHz notch filter -- divide coefs by 2
    b0 = 0.5883783602332997
    b1 = -0.17124071441396285
    b2 = 0.5883783602332997
    a1 = -0.17124071441396285
    a2 = 0.1767567204665992

    // note all coefs are half value of original design, gain handled in algorithm
    filtArray[3].b[0]=0x25a7;           // feed forward b0 coef
    filtArray[3].b[1]=0xf508;           // feed forward b1 coef
    filtArray[3].b[2]=0x25a7;           // feed forward b2 coef

    // note all coefs are half value of original design, gain handled in algorithm
    // note subtract is handled in algorithm, so coefs go in at actual value
    filtArray[3].a[0]=0xf508;           // feedback a1 coef
    filtArray[3].a[1]=0x0b4f;           // feedback a2 coef
```

```
    // 1 KHz notch filter -- divide coefs by 2
    b0 = 0.9087554064944908
    b1 = -1.7990948352036205
    b2 = 0.9087554064944908
    a1 = -1.7990948352036205
    a2 = 0.8175108129889816

    // note all coefs are half value of original design, gain handled in algorithm
    filtArray[7].b[0]=0x3a29;           // feed forward b0 coef
    filtArray[7].b[1]=0x8cdc;           // feed forward b1 coef
    filtArray[7].b[2]=0x3a29;           // feed forward b2 coef

    // note all coefs are half value of original design, gain handled in algorithm
    // note subtract is handled in algorithm, so coefs go in at actual value
    filtArray[7].a[0]=0x8cdc;           // feedback a1 coef
    filtArray[7].a[1]=0x3452;           // feedback a2 coef

for (i=0;i<256;i++)
    {
     // *** get input data here
     dataX = compound_300_1K_hz16[i];

     dataY = DSP_FilterIIRBQ16_cascade8(dataX, filtArray);

     // *** do something with the DataY here
     }
```

Parameters

Parameters	Description
Xin	input data element X (int16_t)
pFilter	pointer to filter coef and delay structure

DSP_FilterIIRBQ16_cascade8_fast Function

Performs a single-sample IIR Biquad Filter as a cascade of 8 series filters.

File

dsp.h

C

```
int16_t DSP_FilterIIRBQ16_cascade8_fast(int16_t Xin, PARM_EQUAL_FILTER_16 * pFilter_Array);
```

Returns

Sample output Y (int16_t)

Description

Function DSP_FilterIIRBQ16_cascade8_fast:

int16_t DSP_FilterIIRBQ16_cascade8_fast(int16_t Xin, PARM_EQUAL_FILTER_16 *pFilter_Array);

Calculates a single pass IIR biquad cascade filter on Xin, and delivers the result as a 16-bit output. The cascade of filters is 8 unique biquad filters arranged in series such that the output of one is provided as the input to the next. A unique filter coefficient set is provided to each, and 16 bit delay lines are maintained for each. All math is performed using 16 bit instructions, which results rounded to 16-bits for the output. All values are fractional Q15, see data structure for specifics. The function will saturate the output should it exceed maximum or minimum values.

Y = Y7 <- Y6 <- Y5 <- Y4 <- Y3 <- Y2 <- Y1 <- Y0 where each Yn filter element represents a unique IIR biquad: Yn = Y(n-1)*b0 + (b1 * Y(n-2)) + (b2 * Y(n-3)) - (a1 * Yn(-1)) - (a2 * Yn(-2)) and: for Y0; Y(n-1) = Xin(0)

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Preconditions

Delay register values should be initialized to zero.

Example

```
PARM_EQUAL_FILTER_16 filtArray[8];
uint16_t dataY, dataX;

// example to use 2 filter blocks as notch filters
// fill entire Filter Array with coefs
for (i=0;i<8;i++)
{
    filtArray[i].Z[0]=0;
    filtArray[i].Z[1]=0;

    // note all coefs are half value of original design, gain handled in algorithm
    // all pass
    filtArray[i].b[0]=0x4000;
    filtArray[i].b[1]=0;        // feed forward b1 coef
    filtArray[i].b[2]=0;  // feed forward b2 coef

    filtArray[i].a[0]=0;  // feedback a1 coef
    filtArray[i].a[1]=0;  // feedback a2 coef
}
    // Unique filters for example
    // 10KHz notch filter -- divide coefs by 2
    b0 = 0.5883783602332997
    b1 = -0.17124071441396285
    b2 = 0.5883783602332997
    a1 = -0.17124071441396285
    a2 = 0.1767567204665992

    // note all coefs are half value of original design, gain handled in algorithm
    filtArray[3].b[0]=0x25a7;           // feed forward b0 coef
    filtArray[3].b[1]=0xf508;           // feed forward b1 coef
    filtArray[3].b[2]=0x25a7;           // feed forward b2 coef

    // note all coefs are half value of original design, gain handled in algorithm
    // note subtract is handled in algorithm, so coefs go in at actual value
    filtArray[3].a[0]=0xf508;           // feedback a1 coef
    filtArray[3].a[1]=0x0b4f;           // feedback a2 coef

    // 1 KHz notch filter -- divide coefs by 2
    b0 = 0.9087554064944908
    b1 = -1.7990948352036205
    b2 = 0.9087554064944908
    a1 = -1.7990948352036205
    a2 = 0.8175108129889816

    // note all coefs are half value of original design, gain handled in algorithm
    filtArray[7].b[0]=0x3a29;           // feed forward b0 coef
    filtArray[7].b[1]=0x8cdc;           // feed forward b1 coef
    filtArray[7].b[2]=0x3a29;           // feed forward b2 coef

    // note all coefs are half value of original design, gain handled in algorithm
    // note subtract is handled in algorithm, so coefs go in at actual value
    filtArray[7].a[0]=0x8cdc;           // feedback a1 coef
    filtArray[7].a[1]=0x3452;           // feedback a2 coef

for (i=0;i<256;i++)
    {
     // *** get input data here
     dataX = compound_300_1K_hz16[i];

     dataY = DSP_FilterIIRBQ16_cascade8_fast(dataX, filtArray);

     // *** do something with the DataY here
     }
```

Parameters

Parameters	Description
Xin	input data element X (int16_t)
pFilter	pointer to filter coef and delay structure

DSP_FilterIIRBQ16_fast Function

Performs a single-pass IIR Biquad Filter.

File

dsp.h

C

```
int16_t DSP_FilterIIRBQ16_fast(int16_t Xin, PARM_EQUAL_FILTER_16 * pFilter);
```

Returns

Sample output Y (int16_t)

Description

Function DSP_FilterIIRBQ16_fast:

int16_t DSP_FilterIIRBQ16_fast(int16_t Xin, PARM_EQUAL_FILTER_16 *pFilter);

Calculates a single pass IIR biquad filter on Xin, and delivers the result as a 16-bit output. All math is performed using 16 bit instructions, with results rounded to 16-bits for the output. The delay register is stored as a 16-bit value for subsequent functions. The function will saturate the results if maximum or minimum fractional values are exceeded. All values are fractional Q15 format.

Y = X(0)*b0 + (b1 * X(-1)) + (b2 * X(-2)) - (a1 * Y(-1)) - (a2 * Y(-2))

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Preconditions

Delay register values should be initialized to zero.

Example

```
PARM_EQUAL_FILTER_16 *ptrFilterEQ;
PARM_EQUAL_FILTER_16 FilterEQ;
uint16_t DataIn, DataOut;
ptrFilterEQ = &FilterEQ;

// 48KHz sampling; 1 KHz bandpass filter; Q=0.9
//  divide by 2 and convert to Q15
//   b0 = 0.06761171785499065
//   b1 = 0
//   b2 = -0.06761171785499065
//   a1 = -1.848823142275648
//   a2 = 0.8647765642900187

// note all coefs are half value of original design, gain handled in algorithm
ptrFiltEQ32->b[0]=0x0453;       // feed forward b0 coef
ptrFiltEQ32->b[1]=0;            // feed forward b1 coef
ptrFiltEQ32->b[2]=0xFBAD;       // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
ptrFiltEQ32->a[0]=0x89AD;       // feedback a1 coef
ptrFiltEQ32->a[1]=0x3758;       // feedback a2 coef

for (i=0;i<256;i++)
    {
    // *** get some input data here
       DataIn32 = three_hundred_hz[i];
```

```
    DataOut = DSP_FilterIIRBQ16_fast(DataIn, ptrFilterEQ);

  // *** do something with the DataOut here
  }
```

Parameters

Parameters	Description
Xin	input data element X (int16_t)
pFilter	pointer to filter coef and delay structure

DSP_FilterIIRBQ16_parallel8 Function

Performs a 8 parallel single-pass IIR Biquad Filters, and sums the result.

File

dsp.h

C

```
int16_t DSP_FilterIIRBQ16_parallel8(int16_t Xin, PARM_EQUAL_FILTER * pFilter);
```

Returns

Sample output Y (int16_t)

Description

Function DSP_FilterIIRBQ16_parallel8:

int16_t DSP_FilterIIRBQ16_parallel8(int16_t Xin, PARM_EQUAL_FILTER *pFilter);

Calculates a 8 parallel, single pass IIR biquad filters on Xin, sums the result and delivers the result as a 16-bit output. All math is performed using 32 bit instructions, which results truncated to 16-bits for the output. The delay register is stored as a 32-bit value for subsequent functions. Output is tuned by 2 multiplier factors. First each parallel section has a fractional gain (attenuation) that enables individual scaling of that section. Second, a global binary (log2N) gain is applied to the result. The combination of gain factors enable both gain and attenuation. All values are fractional Q15 and Q31, see data structure for specifics.

Y = Y7/8 + Y6/8 + Y5/8 + Y4/8 + Y3/8 + Y2/8 + Y1/8 + Y0/8 where each Yn filter element represents a unique IIR biquad: Yn = X(0)*b0 + (b1 * X(n-1)) + (b2 * X(n-2)) - (a1 * Yn(-1)) - (a2 * Yn(-2))

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Preconditions

Delay register values should be initialized to zero. The sum of all fracgain should be <= 1

Example

```
PARM_EQUAL_FILTER filtArrayPara[8];
uint16_t dataY, dataX;

// fill entire Filter Array with coefs
for (i=0;i<8;i++)
{
    filtArrayPara[i].Z[0]=0;
    filtArrayPara[i].Z[1]=0;

    filtArrayPara[i].G.fracGain = 0x7FFF;       // gain = 1 default
    filtArrayPara[i].G.expGain = 1;             // == 2^N; gain of 2

    // note all coefs are half value of original design, gain handled in algorithm
    // none pass -- default
    filtArrayPara[i].b[0]=0;        // feed forward b0 coef
    filtArrayPara[i].b[1]=0;        // feed forward b1 coef
    filtArrayPara[i].b[2]=0;        // feed forward b2 coef

    // note all coefs are half value of original design, gain handled in algorithm
    // note subtract is handled in algorithm, so coefs go in at actual value
    filtArrayPara[i].a[0]=0;        // feedback a1 coef
```

```
    filtArrayPara[i].a[1]=0;        // feedback a2 coef
}

    // 1K bandpass Q=0.9
    filtArrayPara[7].G.fracGain = 0x4000;   // gain = 0.5 because using 2 outputs
    // note all coefs are half value of original design, gain handled in algorithm
    filtArrayPara[7].b[0]=0x04ad;
    filtArrayPara[7].b[1]=0;                // feed forward b1 coef
    filtArrayPara[7].b[2]=0xfb53;           // feed forward b2 coef

    // note all coefs are half value of original design, gain handled in algorithm
    // note subtract is handled in algorithm, so coefs go in at actual value
    filtArrayPara[7].a[0]=0x8a90;           // feedback a1 coef
    filtArrayPara[7].a[1]=0x36a4;           // feedback a2 coef

    // 300 Hz bandpass    Q=0.9
    filtArrayPara[6].G.fracGain = 0x1000;   // gain = 0.125 as an example
    // note all coefs are half value of original design, gain handled in algorithm
    filtArrayPara[6].b[0]=0x017b;           // feed forward b0 coef
    filtArrayPara[6].b[1]=0;                // feed forward b1 coef
    filtArrayPara[6].b[2]=0xfe85;           // feed forward b2 coef

    // note all coefs are half value of original design, gain handled in algorithm
    // note subtract is handled in algorithm, so coefs go in at actual value
    filtArrayPara[6].a[0]=0x8316;           // feedback a1 coef
    filtArrayPara[6].a[1]=0x3d08;           // feedback a2 coef

for (i=0;i<256;i++)
    {
     // *** get input data here
     dataX = compound_300_1K_hz16[i];

     dataY = DSP_FilterIIRBQ16_cascade8_fast(dataX, filtArray);

     // *** do something with the DataY here
     }
```

Parameters

Parameters	Description
Xin	input data element X (int16_t)
pFilter	pointer to filter coef and delay structure

DSP_FilterIIRBQ16_parallel8_fast Function

Performs a 8 parallel single-pass IIR Biquad Filters, and sums the result.

File

dsp.h

C

```
int16_t DSP_FilterIIRBQ16_parallel8_fast(int16_t Xin, PARM_EQUAL_FILTER_16 * pFilter);
```

Returns

Sample output Y (int16_t)

Description

Function DSP_FilterIIRBQ16_parallel8_fast:

int16_t DSP_FilterIIRBQ16_parallel8_fast(int16_t Xin, PARM_EQUAL_FILTER_16 *pFilter);

Calculates a 8 parallel, single pass IIR biquad filters on Xin, sums the result and delivers the result as a 16-bit output. All math is performed using 16 bit instructions, which results rounded to 16-bits for the output. The delay register is stored as a 16-bit value for subsequent functions. Output is tuned by 2 multiplier factors. First each parallel section has a fractional gain (attenuation) that enables individual scaling of that section. Second, a global binary (log2N) gain is applied to the result. The combination of gain factors enable both gain and attenuation. All values are fractional Q15. The function will round outputs and saturate if maximum or minimum values are exceeded.

Y = Y7/8 + Y6/8 + Y5/8 + Y4/8 + Y3/8 + Y2/8 + Y1/8 + Y0/8 where each Yn filter element represents a unique IIR biquad: Yn = X(0)*b0 + (b1 * X(n-1)) + (b2 * X(n-2)) - (a1 * Yn(-1)) - (a2 * Yn(-2))

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Preconditions

Delay register values should be initialized to zero. The sum of all fracgain should be <= 1

Example

```
PARM_EQUAL_FILTER_16 filtArrayPara[8];      // note change in data structure
uint16_t dataY, dataX;

// fill entire Filter Array with coefs
for (i=0;i<8;i++)
{
    filtArrayPara[i].Z[0]=0;
    filtArrayPara[i].Z[1]=0;

    filtArrayPara[i].G.fracGain = 0x7FFF;       // gain = 1 default
    filtArrayPara[i].G.expGain = 1;             // log2N; gain of 2

    // note all coefs are half value of original design, gain handled in algorithm
    // none pass -- default
    filtArrayPara[i].b[0]=0;        // feed forward b0 coef
    filtArrayPara[i].b[1]=0;        // feed forward b1 coef
    filtArrayPara[i].b[2]=0;        // feed forward b2 coef

    // note all coefs are half value of original design, gain handled in algorithm
    // note subtract is handled in algorithm, so coefs go in at actual value
    filtArrayPara[i].a[0]=0;        // feedback a1 coef
    filtArrayPara[i].a[1]=0;        // feedback a2 coef
 }

    // 1K bandpass Q=0.9
    filtArrayPara[7].G.fracGain = 0x4000;   // gain = 0.5 because using 2 outputs
    // note all coefs are half value of original design, gain handled in algorithm
    filtArrayPara[7].b[0]=0x04ad;
    filtArrayPara[7].b[1]=0;                // feed forward b1 coef
    filtArrayPara[7].b[2]=0xfb53;           // feed forward b2 coef

    // note all coefs are half value of original design, gain handled in algorithm
    // note subtract is handled in algorithm, so coefs go in at actual value
    filtArrayPara[7].a[0]=0x8a90;           // feedback a1 coef
    filtArrayPara[7].a[1]=0x36a4;           // feedback a2 coef

    // 300 Hz bandpass    Q=0.9
    filtArrayPara[6].G.fracGain = 0x1000;   // gain = 0.125 as an example
    // note all coefs are half value of original design, gain handled in algorithm
    filtArrayPara[6].b[0]=0x017b;           // feed forward b0 coef
    filtArrayPara[6].b[1]=0;                // feed forward b1 coef
    filtArrayPara[6].b[2]=0xfe85;           // feed forward b2 coef

    // note all coefs are half value of original design, gain handled in algorithm
    // note subtract is handled in algorithm, so coefs go in at actual value
    filtArrayPara[6].a[0]=0x8316;           // feedback a1 coef
    filtArrayPara[6].a[1]=0x3d08;           // feedback a2 coef

for (i=0;i<256;i++)
    {
     // *** get input data here
     dataX = compound_300_1K_hz16[i];

     dataY = DSP_FilterIIRBQ16_cascade8_fast(dataX, filtArray);

     // *** do something with the DataY here
     }
```

Parameters

Parameters	Description
Xin	input data element X (int16_t)
pFilter	pointer to filter coef and delay structure

DSP_FilterIIRBQ32 Function

Performs a high resolution single-pass IIR Biquad Filter.

File

dsp.h

C

```
int32_t DSP_FilterIIRBQ32(int32_t Xin, PARM_EQUAL_FILTER_32 * pFilter);
```

Returns

Sample output Y (int32_t)

Description

Function DSP_FilterIIRBQ32:

int32_t DSP_FilterIIRBQ32(int32_t Xin, PARM_EQUAL_FILTER_32 *pFilter);

Calculates a single pass IIR biquad filter on Xin, and delivers the result as a 16-bit output. All math is performed using 32 bit instructions, with results truncated to 32-bits for the output. The delay register is stored as a 32-bit value for subsequent functions. All values are fractional Q31, see data structure for specifics.

Y = X(0)*b0 + (b1 * X(-1)) + (b2 * X(-2)) - (a1 * Y(-1)) - (a2 * Y(-2))

Remarks

The delay register values should be initialized to zero prior to the first call to the function, they are updated each pass. A gain of 2 has been hard coded into the function. This implies that all coefs should be input at half value. This is purposeful, since many filter designs need a div2 to have each coef between the required -1

Preconditions

Delay register values should be initialized to zero.

Example

```
PARM_EQUAL_FILTER_32 *ptrFiltEQ32;
PARM_EQUAL_FILTER_32 FilterEQ32;
int32_t DataIn32, DataOut32;

ptrFiltEQ32 = &FilterEQ32;

ptrFiltEQ32->Z[0]=0;
ptrFiltEQ32->Z[1]=0;

//  1000 Hz Q= 0.9 BP filter design, 44.1K sampling
//
//    b0 = 0.07311778239751009  forward
//    b1 = 0
//    b2 = -0.07311778239751009
//    a1 = -1.8349811166056893  back
//    a2 = 0.8537644352049799

// note all coefs are half value of original design, gain handled in algorithm
ptrFiltEQ32->b[0]=0x04ADF635;   // feed forward b0 coef
ptrFiltEQ32->b[1]=0;            // feed forward b1 coef
ptrFiltEQ32->b[2]=0xFB5209CB;   // feed forward b2 coef

// note all coefs are half value of original design, gain handled in algorithm
// note subtract is handled in algorithm, so coefs go in at actual value
ptrFiltEQ32->a[0]=0x8A8FAB5D;  // feedback a1 coef
ptrFiltEQ32->a[1]=0x36A41395;  // feedback a2 coef

for (i=0;i<256;i++)
```

```
{
 // *** get input data here
   DataIn32 = three_hundred_hz[i];

   DataOut32 = DSP_FilterIIRBQ32(DataIn32, ptrFiltEQ32);

 // *** do something with the DataOut32 here
 }
```

Parameters

Parameters	Description
Xin	input data element X (int32_t)
pFilter	pointer to high resolution filter coef and delay structure

DSP_FilterIIRSetup16 Function

Converts biquad structure to coeffs array to set up IIR filter.

File

dsp.h

C

```
void DSP_FilterIIRSetup16(int16_t * coeffs, biquad16 * bq, int B);
```

Returns

None.

Description

Function DSP_FilterIIRSetup16:

void DSP_FilterIIRSetup16(int16_t *coeffs, biquad16 *bq, int B);

Converts an array of biquad coefficients, bq, into an linear array of coefficients, coeffs. The output array must be 4*B elements long. The number of biquads in the resulting factor is given by B. All numerical values must be in Q15 fractional data format.

Remarks

None.

Preconditions

coeffs must have 4*B elements.

Example

```
see DSP_FilterIIR16 for example.
```

Parameters

Parameters	Description
coeffs	pointer to an array of coefficients (int16_t)
bq	pointer to array of biquad structure filter coefs (biquad16)
B	number of cascaded biquad filter groups to process (int)

DSP_FilterLMS16 Function

Performs a single sample Least Mean Squares FIR Filter.

File

dsp.h

C

```
int16_t DSP_FilterLMS16(int16_t in, int16_t ref, int16_t * coeffs, int16_t * delayline, int16_t * error,
int K, int16_t mu);
```

Returns

(int16_t) - FIR filter output

Description

Function DSP_FilterLMS16:

int16_t DSP_FilterLMS16(int16_t in, int16_t ref, int16_t *coeffs, int16_t *delayline, int16_t *error, int K, int16_t mu);

Computes an LMS adaptive filter on the input in. Filter output of the FIR is given as a 16 bit value. The filter target is ref, and the calculation difference between the output and the target is error. The filter adapts its coefficients, coefs, on each pass. The number of coefficients (filter taps) is given by the value K. The delayline array should be initialized to zero prior to calling the filter for the first time, and have K elements. The value mu is the rate at which the filter adapts. All values are Q15 fractional numbers. The function will saturate the output if it exceeds maximum or minimum values. The LMS will adapt its coefs to attempt to drive the output value toward the ref value. The rate of adaption on each pass depends on mu and the error from the previous calculation.

Remarks

Filter coefs may start at random or zero value, but convergence is dependent on the amount of update required.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. delayline must have (K) elements, and be initialized to zero. K (taps) must be a multiple of 4, and >= 8. mu must be positive.

Example

```
#define lmsTAPS 8

int16_t lmsOut;
int lmsTaps = lmsTAPS;

int16_t lmsCoefs[lmsTAPS]={0x5000, 0x4000,0x3000, 0x2000,0x1000, 0x0000,0xF000, 0xE000};
int16_t lmsDelay[lmsTAPS]={0};
int16_t *ptrLMSError;
int16_t lmsError = 0x0200;
int16_t inVal=0;

int16_t refVal = 0x0CCC;  // some target value = 0.1
int16_t lmsAdapt = 0x3000;

ptrLMSError = &lmsError;

for (i=0;i<200;i++)
{
// get some input value here //
 if (i < 100)
 {
     inVal = 0x4233;
 }
 else
 {
     inVal = 0xCF10;
 }

 lmsOut =  DSP_FilterLMS16(inVal, refVal, lmsCoefs, lmsDelay,
       ptrLMSError, lmsTaps, lmsAdapt);
}
```

Parameters

Parameters	Description
in	input data value (int16_t)
ref	target output value (int16_t)
coeffs	pointer to an array of coefficients (int16_t)
delayline	pointer to an array of delay variables (int16_t)
error	output minus reference (int16_t)
K	number of filter taps and coeffs (int)
mu	adaption rate (int16_t)

c) Matrix Math Functions

DSP_MatrixAdd32 Function

Addition of two matrices C = (A + B).

File

dsp.h

C

```
void DSP_MatrixAdd32(matrix32 * resMat, matrix32 * srcMat1, matrix32 * srcMat2);
```

Returns

None.

Description

Function DSP_MatrixAdd32:

void DSP_MatrixAdd32(matrix32 *resMat, matrix32 *srcMat1, matrix32 *srcMat2);

Vector summation of two matrices, where both have 32-bit integer elements. The resulting output will saturate if element addition exceeds MAX32 or MIN32.

Remarks

Execution Time (cycles): 225 cycles + 23 / matrix_element. The function will saturate the output value if it exceeds maximum limits per element.

Preconditions

Both matrices must be equivalent in rows and columns. Both Matrices must be set into structure (ROWS, COLUMNS, vector_pointer).

Example

```
#define ROW 2
#define COL 2

matrix32 *resMat, *srcMat1, *srcMat2;
int32_t result[ROW*COL];

int32_t matA[ROW*COL] = {1,2,3,4};
int32_t matB[ROW*COL] = {2,4,6,8};

matrix32 mat, mat2, mat3;
resMat=&mat;
srcMat1=&mat2;
srcMat2=&mat3;

srcMat1->row=ROW;
srcMat1->col=COL;
srcMat1->pMatrix=matA;

srcMat2->col=COL;
srcMat2->row=ROW;
srcMat2->pMatrix=matB;

resMat->row=ROW;
resMat->col=COL;
resMat->pMatrix=result;

DSP_MatrixAdd32(resMat, srcMat1, srcMat2);

// result[i] = matA[i] + matB[i] = {3,6,9,0xA}
```

Parameters

Parameters	Description
resMat	pointer to new sum Matrix C (*int32_t)
srcMat1	pointer to the Matrix A structure (*int32_t)
srcMat2	pointer to the Matrix B structure (*int32_t)

DSP_MatrixEqual32 Function

Equality of two matrices C = (A).

File

dsp.h

C

```
void DSP_MatrixEqual32(matrix32 * resMat, matrix32 * srcMat);
```

Returns

None.

Description

Function DSP_MatrixEqual32:

void DSP_MatrixEqual32(matrix32 *resMat, matrix32 *srcMat);

Vector copy of all elements from one matrix to another. C is a duplicate of A.

Remarks

Execution Time (cycles): 163 cycles + 12 / matrix_element.

Preconditions

None.

Example

```
#define ROW 2
#define COL 2

matrix32 *resMat, *srcMat1, *srcMat2;
int32_t result[ROW*COL];

int32_t matA[ROW*COL] = {5,2,-3,8};

matrix32 mat, mat2;
resMat=&mat;
srcMat1=&mat2;

srcMat1->row=ROW;
srcMat1->col=COL;
srcMat1->pMatrix=matA;

resMat->row=ROW;
resMat->col=COL;
resMat->pMatrix=result;

DSP_MatrixEqual32(resMat, srcMat1, srcMat2);

// result[i] = matA[i] = {5, 2, -3, 8}
```

Parameters

Parameters	Description
resMat	pointer to completed new Matrix C (*int32_t)
srcMat	pointer to the Matrix A structure (*int32_t)

DSP_MatrixInit32 Function

Initializes the first N elements of a Matrix to the value num.

File

dsp.h

C

```
void DSP_MatrixInit32(int32_t * data_buffer, int N, int32_t num);
```

Returns

None.

Description

Function DSP_MatrixInit32:

void DSP_MatrixInit32(int32_t *data_buffer, int N, int32_t num);

Copy the value num into the first N Matrix elements of data_buffer.

Remarks

None.

Preconditions

data_buffer must be predefined to be equal to or greater than N elements. N must be a factor of four, or it will truncate to the nearest factor of four.

Example

```
#define ROW 3
#define COL 3

int32_t numElements = 4;  // multiple of 4
int valueElements = -1;

int32_t matA[ROW*COL] = {5,2,-3,8,4,2,-6,8,9};

DSP_MatrixInit32(matA, numElements, valueElements);

// matA[i] = {-1,-1,-1,-1,4,2,-6,8,9}
```

Parameters

Parameters	Description
data_buffer	pointer to the Matrix to be initialized (int32_t[M*N])
N	number of elements to be initialized (int32_t)
num	value to be initialized into the matrix (int32_t)

DSP_MatrixMul32 Function

Multiplication of two matrices C = A x B.

File

dsp.h

C

```
void DSP_MatrixMul32(matrix32 * resMat, matrix32 * srcMat1, matrix32 * srcMat2);
```

Returns

None.

Description

Function DSP_MatrixMul32:

void DSP_MatrixMul32(matrix32 *resMat, matrix32 *srcMat1, matrix32 *srcMat2);

Multiplication of two matrices, with inputs and outputs being in fractional Q31 numerical format. The output elements will saturate if the dot product exceeds maximum or minimum fractional values.

Remarks

Execution Time (cycles): 319 cycles + 38 / output matrix_element. The function will saturate the output value if it exceeds maximum limits per element.

Preconditions

Matrices must be aligned such that columns of A = rows of B. resMat must have the format of rows of A, columns of B. All Matrices must be set

into structure (ROWS, COLUMNS, vector_pointer).

Example

```
#define ROW1 3
#define COL1 2
#define ROW2 2
#define COL2 2

matrix32 *resMat, *srcMat1, *srcMat2;
int32_t result[ROW1*COL2];

int32_t test_MatrixA[ROW1*COL1]=
{
    0x40000000,0x20000000,      // 0.5,  0.25
    0xD999999A,0x4CCCCCCC,      // -0.3, 0.6
    0xC0000000,0x0CCCCCCD       // -0.5  0.1
};

int32_t test_MatrixB[ROW2*COL2]=
{
    0x40000000,0x20000000,      // 0.5, 0.25
    0x0CCCCCCD,0xCCCCCCCD       // 0.1, -0.4
};

matrix32 mat, mat2, mat3;
resMat=&mat;
srcMat1=&mat2;
srcMat2=&mat3;

srcMat1->row=ROW1;
srcMat1->col=COL1;
srcMat1->pMatrix=test_MatrixA;

srcMat2->col=COL2;
srcMat2->row=ROW2;
srcMat2->pMatrix=test_MatrixB;

resMat->row=ROW1;  // note resulting matrix MUST have ROW1 & COL2 format
resMat->col=COL2;
resMat->pMatrix=result;

DSP_MatrixMul32(resMat, srcMat1, srcMat2);

// result[] = matA[] x matB[] =
//    { 0x23333333, 0x03333333   // 0.275, 0.025
//      0xF47AE147, 0xD7AE147B   // -0.9, -0.315
//      0xE147AE14, 0xEAE147AE } // -0.24, -0.165
```

Parameters

Parameters	Description
resMat	pointer to different Matrix C structure (*int32_t)
srcMat1	pointer to the Matrix A structure (*int32_t)
srcMat2	pointer to the Matrix B structure (*int32_t)

DSP_MatrixScale32 Function

Scales each element of an input buffer (matrix) by a fixed number.

File

dsp.h

C

```
void DSP_MatrixScale32(int32_t * data_buffer, int N, int32_t num);
```

Returns

None.

Description

Function DSP_MatrixScale32:

void DSP_MatrixScale32(int32_t *data_buffer, int N, int32_t num);

Multiply the first N elements of an input buffer by a fixed scalar num. The resulting value is stored back into the input buffer. N number total samples of the input buffer are processed. All values are in Q31 fractional integer format. The result of calculations is saturated to the MAX32 or MIN32 value if exceeded.

Remarks

Execution time (cycles): 190 + 9 cycles / element, typical.

Preconditions

data_buffer must be predefined to be equal to or greater than N elements. N must be a factor of four, or will truncate to the nearest factor of four.

Example

```
int32_t numScale = 0x40000000;  // 0.5
int valN = 12;
int32_t inputBufScale[12] = {0x40000000, 0x40000000, 0x20000000, 0x20000000,
            0x19999999, 0xCCCCCCCD, 0xF3333333, 0x80000000,
            0x7FFFFFFF, 0x00000000, 0x40000000, 0x70000000 };
// 0.5, 0.5, 0.25, 0.25, 0.25, 0.2, -0.4, -0.1, -1, 1, 0, 0.5, 0.875

DSP_MatrixScale32(inputBufScale,valN,numScale);

// inputBufScale[i] = {0x20000000, 0x20000000, 0x10000000, 0x10000000,
//          0x0CCCCCCC, 0xE6666666, 0xF9999999, 0xC0000000,
//          0x3FFFFFFF, 0x00000000, 0x20000000, 0x38000000}
//  0.25, 0.25, 0.125, 0.125, 0.1, -0.2, -0.05, -0.5, 0.5, 0, 0.25, 0.4375
```

Parameters

Parameters	Description
data_buffer	pointer to the Matrix to be initialized (int32_t[M*N])
N	number of elements to be initialized (int)
num	value to be initialized into the matrix (int32_t)

DSP_MatrixSub32 Function

Subtraction of two matrices C = (A - B).

File

dsp.h

C

```
void DSP_MatrixSub32(matrix32 * resMat, matrix32 * srcMat1, matrix32 * srcMat2);
```

Returns

None.

Description

Function DSP_MatrixSub32:

void DSP_MatrixSub32(matrix32 *resMat, matrix32 *srcMat1, matrix32 *srcMat2);

Vector subtraction of two matrices, where both have 32-bit integer elements. The resulting output will saturate if element addition exceeds MAX32 or MIN32.

Remarks

Execution Time (cycles): 222 cycles + 21 / matrix_element. The function will saturate the output value if it exceeds maximum limits per element.

Preconditions

Both matrices must be equivalent in rows and columns. All Matrices must be set into structure (ROWS, COLUMNS, vector_pointer)

Example

```
#define ROW 2
```

```
#define COL 2

matrix32 *resMat, *srcMat1, *srcMat2;
int32_t result[ROW*COL];

int32_t matA[ROW*COL] = {5,2,-3,8};
int32_t matB[ROW*COL] = {2,2,2,2};

matrix32 mat, mat2, mat3;
resMat=&mat;
srcMat1=&mat2;
srcMat2=&mat3;

srcMat1->row=ROW;
srcMat1->col=COL;
srcMat1->pMatrix=matA;

srcMat2->col=COL;
srcMat2->row=ROW;
srcMat2->pMatrix=matB;

resMat->row=ROW;
resMat->col=COL;
resMat->pMatrix=result;

DSP_MatrixSub32(resMat, srcMat1, srcMat2);

// result[i] = matA[i] - matB[i] = {3,0,-5,6}
```

Parameters

Parameters	Description
resMat	pointer to different Matrix C structure (*int32_t)
srcMat1	pointer to the Matrix A structure (*int32_t)
srcMat2	pointer to the Matrix B structure (*int32_t)

DSP_MatrixTranspose32 Function

Transpose of a Matrix C = A (T).

File

dsp.h

C

```
void DSP_MatrixTranspose32(matrix32 * desMat, matrix32 * srcMat);
```

Returns

None.

Description

Function DSP_MatrixTranspose32:

void DSP_MatrixTranspose32(matrix32 *desMat, matrix32 *srcMat);

Transpose of rows and columns of a matrix.

Remarks

Execution Time (cycles): 210 cycles + 10 / matrix_element.

Preconditions

Matrix definitions for ROWS and COLS must be transposed prior to the function call.

Example

```
#define ROW 3
#define COL 4

matrix32 *resMat, *srcMat1;
```

```
int32_t result[ROW*COL];

int32_t matA[ROW*COL] = { 1,  2,  3,  4,
                          5,  6,  7,  8,
                         -1, -3, -5, -7};

matrix32 mat, mat2;
resMat=&mat;
srcMat1=&mat2;

srcMat1->row=ROW;
srcMat1->col=COL;
srcMat1->pMatrix=matA;

resMat->row=COL;   // note the shift in columns and rows
resMat->col=ROW;
resMat->pMatrix=result;

DSP_MatrixTranspose32(resMat, srcMat1);

// result[] = matA(T)[] = {   1,  5, -1,
//                            2,  6, -3,
//                            3,  7, -5,
//                            4,  8, -7}
```

Parameters

Parameters	Description
desMat	pointer to transposed new Matrix C (*int32_t)
srcMat	pointer to the Matrix A structure (*int32_t)

d) Transform Functions

DSP_TransformFFT16 Function

Creates an Fast Fourier Transform (FFT) from a time domain input.

File

dsp.h

C

```
void DSP_TransformFFT16(int16c * dout, int16c * din, int16c * twiddles, int16c * scratch, int log2N);
```

Returns

None.

Description

Function DSP_TransformFFT16:

void DSP_TransformFFT16(int16c *dout, int16c *din, int16c *twiddles, int16c *scratch, int log2N);

Performs an complex FFT on the input, din, and stores the complex result in dout. Performs 2^log2N point calculation, and the working buffer scratch as well as the input and output must be 2^log2N in length. Coefficient twiddle factors come from twiddles, and may be loaded with the use of DSP_TransformFFT16_setup. All values are 16 bit (Q15) fractional.

Remarks

Scratch must be declared but need not be initialized. Din may be aided with a window function prior to calling the FFT, but is not required. Din is a complex number array, but may be loaded solely with real numbers if no phase information is available.

Preconditions

din, dout, twiddles and scratch must have N elements N is calculated as 2^(log2N) log2N must be >= 3 FFT factors must be calculated in advance, use DSP_TransformFFT16_setup

Example

```
int log2N = 8;  // log2(256) = 8
int fftSamples = 256;
```

```
int16c  *fftDin;
int16c  fftDout[fftSamples];
int16c  scratch[fftSamples];
int16c  fftCoefs[fftSamples];

int16c *fftc;
fftc = &fftCoefs;

DSP_TransformFFT16_setup(fftc, log2N);  // call setup function

while (1)
{
    fftDin = &fftin_8Khz_long_window16;  // get 256 point complex data

    DSP_TransformFFT16(fftDout, fftDin, fftc, scratch, log2N);

    // do something with the output, fftDout
};
```

Parameters

Parameters	Description
dout	pointer to complex output array (int16c)
din	pointer to complex input array (int16c)
twiddles	pointer to an complex array of factors (int16c)
scratch	pointer to a complex scratch pad buffer (int16c)
log2N	binary exponent of number of samples (int)

DSP_TransformFFT16_setup Function

Creates FFT coefficients for use in the FFT16 function.

File

dsp.h

C

```
void DSP_TransformFFT16_setup(int16c * twiddles, int log2N);
```

Returns

None.

Description

Function DSP_TransformFFT16_setup:

void DSP_TransformFFT16_setup(int16c *twiddles, int log2N);

Calculates the N twiddle factors required to operate the FFT16 function. These factors are done in serial fashion, and require considerable processing power. Ideally this function would be run only once prior to an ongoing FFT, and the results held in a buffer.

Remarks

This function is of considerable length and executed in C. It is recommended it only be called once for any given FFT length in time sensitive applications.

Preconditions

twiddles must be N in length N is calculated (2^log2N)

Example

```
see DSP_TransformFFT16 for example.
```

Parameters

Parameters	Description
twiddles	pointer to a complex array of factors (int16c)
log2N	binary exponent of number of data points (int)

DSP_TransformFFT32 Function

Creates an Fast Fourier Transform (FFT) from a time domain input.

File

dsp.h

C

```
void DSP_TransformFFT32(int32c * dout, int32c * din, int32c * twiddles, int32c * scratch, int log2N);
```

Returns

None.

Description

Function DSP_TransformFFT32:

void DSP_TransformFFT32(int32c *dout, int32c *din, int32c *twiddles, int32c *scratch, int log2N);

Performs an complex FFT on the input, din, and stores the complex result in dout. Performs 2^log2N point calculation, and the working buffer scratch as well as the input and output must be 2^log2N in length. Coefficient twiddle factors come from twiddles, and may be loaded with the use of DSP_TransformFFT16_setup. All values are 16 bit (Q31) fractional.

Remarks

Scratch must be declared but need not be initialized. Din may be aided with a window function prior to calling the FFT, but is not required. Din is a complex number array, but may be loaded solely with real numbers if no phase information is available.

Preconditions

din, dout, twiddles and scratch must have N elements N is calculated as 2^(log2N) log2N must be >= 3 FFT factors must be calculated in advance, use DSP_TransformFFT32_setup

Example

```
int log2N = 8;  // log2(256) = 8
int fftSamples = 256;

int32c  *fftDin;
int32c  fftDout[fftSamples];
int32c  scratch[fftSamples];
int32c  fftCoefs[fftSamples];

int32c *fftc;
fftc = &fftCoefs;

DSP_TransformFFT32_setup(fftc, log2N);  // call setup function

while (1)
{
    fftDin = &fftin_5Khz_long_window32;  // get 256 point complex data

    DSP_TransformFFT32(fftDout, fftDin, fftc, scratch, log2N);

    // do something with the output, fftDout
};
```

Parameters

Parameters	Description
dout	pointer to complex output array (int32c)
din	pointer to complex input array (int32c)
twiddles	pointer to an complex array of FFT factors (int32c)
scratch	pointer to a complex scratch pad buffer (int32c)
log2N	binary exponent of number of samples (int)

DSP_TransformFFT32_setup Function

Creates FFT coefficients for use in the FFT32 function.

File

dsp.h

C

```
void DSP_TransformFFT32_setup(int32c * twiddles, int log2N);
```

Returns

None.

Description

Function DSP_TransformFFT32_setup:

void DSP_TransformFFT32_setup(int32c *twiddles, int log2N);

Calculates the N FFT twiddle factors required to operate the FFT32 function. These factors are done in serial fashion, and require considerable processing power. Ideally this function would be run only once prior to an ongoing FFT, and the results held in a buffer.

Remarks

This function is of considerable length and executed in C. It is recommended it only be called once for any given FFT length in time sensitive applications.

Preconditions

twiddles must be N in length N is calculated (2^log2N)

Example

```
see DSP_TransformFFT32 for example.
```

Parameters

Parameters	Description
twiddles	pointer to a complex array of coefficients (int32c)
log2N	binary exponent of number of data points (int)

DSP_TransformIFFT16 Function

Creates an Inverse Fast Fourier Transform (FFT) from a frequency domain input.

File

dsp.h

C

```
void DSP_TransformIFFT16(int16c * dout, int16c * din, int16c * twiddles, int16c * scratch, int log2N);
```

Returns

None.

Description

Function DSP_TransformIFFT16:

void DSP_TransformIFFT16(int16c *dout, int16c *din, int16c *twiddles, int16c *scratch, int log2N);

Performs an complex Inverse FFT on the input, din, and stores the complex result in dout. Performs 2^log2N point calculation, and the working buffer scratch as well as the input and output must be 2^log2N in length. Coefficient twiddle factors come from twiddles, and may be loaded with the use of DSP_TransformFFT16_setup. All values are 16 bit (Q15) fractional.

Remarks

Scratch must be declared but need not be initialized. Din may be aided with a window function prior to calling the FFT, but is not required. A very similar function to the FFT is executed for the inverse FFT. This requires twiddle factors set in advance with the same method as used in the FFT. Complex conjugate and scaling are handled within the algorithm. The output is scaled using binary shifting based on log2N. Since the algorithm reduces the output by a scale factor of log2N, the resolution is reduced proportionally to the number of data points.

Preconditions

din, dout, twiddles and scratch must have N elements N is calculated as 2^(log2N) log2N must be >= 3 FFT factors must be calculated in advance, use DSP_TransformFFT16_setup

Example

```
int ilog2N = 10;  // log2(64) = 6; log2(256) = 8; log2(1024) = 10;
int ifftSamples = pow(2,ilog2N);

int16c  *ifftDin;
int16c ifftDout[ifftSamples];
int16c  iscratch[ifftSamples];
int16c  ifftCoefs[ifftSamples];
int16c  ifftTimeOut[ifftSamples];

// set up twiddle factors, these are used for both FFT and iFFT

int16c *ifftc;
ifftc = &ifftCoefs;
DSP_TransformFFT16_setup( ifftc, ilog2N);  // call to coef setup

// in this example, we take an FFT of an original time domain (sine wave)
// the output of the FFT is used as the input of the iFFT for comparison

ifftDin = &fftin_800hz_verylong16;
DSP_TransformFFT16(ifftDout, ifftDin, ifftc, iscratch, ilog2N);

// ifftDout = frequency domain output, complex number array

DSP_TransformIFFT16(ifftTimeOut, ifftDout, ifftc, iscratch, ilog2N);

// do something with the output, fftTimeOut, time domain
```

Parameters

Parameters	Description
dout	pointer to complex output array (int16c)
din	pointer to complex input array (int16c)
twiddles	pointer to an complex array of factors (int16c)
scratch	pointer to a complex scratch pad buffer (int16c)
log2N	binary exponent of number of samples (int)

DSP_TransformWindow_Bart16 Function

Perform a Bartlett window on a vector.

File

dsp.h

C

```
void DSP_TransformWindow_Bart16(int16_t * OutVector, int16_t * InVector, int N);
```

Returns

None.

Description

Function DSP_TransformWindow_Bart16:

void DSP_TransformWindow_Bart16(int16_t *OutVector, int16_t *InVector, int N);

Compute a Bartlett (Triangle) Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q15 fractional format. The Bartlett Window follows the equation:

Window(n) = 1 - (abs(2*n - N)/N) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

```
int16_t OutVector16[8]={0};
int  WindowN = 8;

for (i=0;i<WindowN;i++)
{
   InVector16[i]= 0x4000;  // constant 0.5 for functional testing
}

   DSP_TransformWindow_Bart16(OutVector16, InVector16, WindowN);
// OutWindow = 0x0000, 0x1000, 0x2000, 0x3000, 0x4000, 0x3000, 0x2000, 0x1000
```

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

DSP_TransformWindow_Bart32 Function

Perform a Bartlett window on a vector.

File

dsp.h

C

```
void DSP_TransformWindow_Bart32(int32_t * OutVector, int32_t * InVector, int N);
```

Returns

None.

Description

Function DSP_TransformWindow_Bart32:

void DSP_TransformWindow_Bart32(int32_t *OutVector, int32_t *InVector, int N);

Compute a Bartlett (Triangle) Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q31 fractional format. The Bartlett Window follows the equation:

Window(n) = 1 - (abs(2*n - N)/N) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

```
int32_t OutVector32[8]={0};
int  WindowN = 8;

for (i=0;i<WindowN;i++)
{
   InVector32[i]= 0x40000000;  // constant 0.5 for functional testing
}

   DSP_TransformWindow_Bart32(OutVector32, InVector32, WindowN);
// OutWindow = 0x0, 0x10000000, 0x20000000, 0x30000000, 0x40000000,
//                      0x30000000, 0x20000000, 0x10000000
```

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

DSP_TransformWindow_Black16 Function

Perform a Blackman window on a vector.

File

dsp.h

C

```
void DSP_TransformWindow_Black16(int16_t * OutVector, int16_t * InVector, int N);
```

Returns

None.

Description

Function DSP_TransformWindow_Black16:

void DSP_TransformWindow_Black16(int16_t *OutVector, int16_t *InVector, int N);

Compute a Blackman Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q15 fractional format. The Blackman Window follows the equation:

Window(n) = 0.42659 - 0.49656 * COS(2*Pi*n/(N-1)) + 0.076849 * COS(4*Pi*n/(N-1)) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

```
int16_t OutVector16[8]={0};
int  WindowN = 8;

for (i=0;i<WindowN;i++)
{
   InVector16[i]= 0x4000;  // constant 0.5 for functional testing
}

   DSP_TransformWindow_Black16(OutVector16, InVector16, WindowN);
// OutWindow = 0x0071, 0x0665, 0x1DF1, 0x3B00, 0x3B00, 0x1DF1, 0x0665, 0x0071
```

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

DSP_TransformWindow_Black32 Function

Perform a Blackman window on a vector.

File

dsp.h

C

```
void DSP_TransformWindow_Black32(int32_t * OutVector, int32_t * InVector, int N);
```

Returns

None.

Description

Function DSP_TransformWindow_Black32:

void DSP_TransformWindow_Black32(int32_t *OutVector, int32_t *InVector, int N);

Compute a Blackman Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q31 fractional format. The Blackman Window follows the equation:

Window(n) = 0.42659 - 0.49656 * COS(2*Pi*n/(N-1)) + 0.076849 * COS(4*Pi*n/(N-1)) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

```
int32_t OutVector32[8]={0};
int  WindowN = 8;

for (i=0;i<WindowN;i++)
{
   InVector32[i]= 0x40000000;  // constant 0.5 for functional testing
}

   DSP_TransformWindow_Black32(OutVector32, InVector32, WindowN);
// OutWindow = 0x0070B490, 0x06649680, 0x1DF13240, 0x3B003D80, 0x3B003D80,
//                       0x1DF13240, 0x06649680, 0x0070B490
```

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

DSP_TransformWindow_Cosine16 Function

Perform a Cosine (Sine) window on a vector.

File

dsp.h

C

```
void DSP_TransformWindow_Cosine16(int16_t * OutVector, int16_t * InVector, int N);
```

Returns

None.

Description

Function DSP_TransformWindow_Cosine16:

void DSP_TransformWindow_Cosine16(int16_t *OutVector, int16_t *InVector, int N);

Compute a Cosine Window on the first N samples of the input vector, InVector. The output is stored in OutWindow. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q15 fractional format. The Cosine Window follows the equation:

Window(n) = SIN(Pi*n/(N-1)) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

```
int16_t OutVector16[8]={0};
int  WindowN = 8;

for (i=0;i<WindowN;i++)
{
   InVector16[i]= 0x4000;  // constant 0.5 for functional testing
}

   DSP_TransformWindow_Cosine16(OutVector16, InVector16, WindowN);
// OutWindow = 0x0000, 0x1BC5, 0x320A, 0x3E65, 0x3E65, 0x320A, 0x1BC5, 0x0071
```

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

DSP_TransformWindow_Cosine32 Function

Perform a Cosine (Sine) window on a vector.

File

dsp.h

C

```
void DSP_TransformWindow_Cosine32(int32_t * OutVector, int32_t * InVector, int N);
```

Returns

None.

Description

Function DSP_TransformWindow_Cosine32:

void DSP_TransformWindow_Cosine32(int32_t *OutVector, int32_t *InVector, int N);

Compute a Cosine Window on the first N samples of the input vector, InVector. The output is stored in OutWindow. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q31 fractional format. The Cosine Window follows the equation:

Window(n) = SIN(Pi*n/(N-1)) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

```
int32_t OutVector32[8]={0};
int  WindowN = 8;

for (i=0;i<WindowN;i++)
{
   InVector32[i]= 0x40000000;  // constant 0.5 for functional testing
}

   DSP_TransformWindow_Cosine32(OutVector32, InVector32, WindowN);
// OutWindow = 0x00000000, 0x1BC4C060, 0x32098700, 0x3E653800, 0x3E653800,
//                       0x32098700, 0x1BC4C060, 0x00000000
```

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

DSP_TransformWindow_Hamm16 Function

Perform a Hamming window on a vector.

File

dsp.h

C

```
void DSP_TransformWindow_Hamm16(int16_t * OutVector, int16_t * InVector, int N);
```

Returns

None.

Description

Function DSP_TransformWindow_Hamm16:

void DSP_TransformWindow_Hamm16(int16_t *OutVector, int16_t *InVector, int N);

Compute a Hamming Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q15 fractional format. The Hamming Window follows the equation:

Window(n) = 0.54 - 0.46 * COS(2*Pi*n/N) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

```
int16_t OutVector16[8]={0};
int  WindowN = 8;

for (i=0;i<WindowN;i++)
{
   InVector16[i]= 0x4000;  // constant 0.5 for functional testing
}

   DSP_TransformWindow_Hamm16(OutVector16, InVector16, WindowN);
// OutWindow = 0x051F, 0x0DBE, 0x228F, 0x3761, 0x4000, 0x3761, 0x228F, 0x0DBE
```

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

DSP_TransformWindow_Hamm32 Function

Perform a Hamming window on a vector.

File

dsp.h

C

```
void DSP_TransformWindow_Hamm32(int32_t * OutVector, int32_t * InVector, int N);
```

Returns

None.

Description

Function DSP_TransformWindow_Hamm32:

void DSP_TransformWindow_Hamm32(int32_t *OutVector, int32_t *InVector, int N);

Compute a Hamming Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q31 fractional format. The Hamming Window follows the equation:

Window(n) = 0.54 - 0.46 * COS(2*Pi*n/N) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

```
int32_t OutVector32[8]={0};
int  WindowN = 8;

for (i=0;i<WindowN;i++)
{
   InVector32[i]= 0x40000000;  // constant 0.5 for functional testing
}

   DSP_TransformWindow_Hamm32(OutVector32, InVector32, WindowN);
// OutWindow = 0x051EB860, 0x0DBE26C0, 0x228F5C40, 0x37609200, 0x40000000,
//                       0x37609200, 0x228F5C40, 0x0DBE26C0
```

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

DSP_TransformWindow_Hann16 Function

Perform a Hanning window on a vector.

File

dsp.h

C

```
void DSP_TransformWindow_Hann16(int16_t * OutVector, int16_t * InVector, int N);
```

Returns

None.

Description

Function DSP_TransformWindow_Hann16:

void DSP_TransformWindow_Hann16(int16_t *OutVector, int16_t *InVector, int N);

Compute a Hanning Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q15 fractional format. The Hanning Window follows the equation:

Window(n) = 0.5 - 0.5 * COS(2*Pi*n/N) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

```
int16_t OutVector16[8]={0};
int  WindowN = 8;

for (i=0;i<WindowN;i++)
{
   InVector16[i]= 0x4000;  // constant 0.5 for functional testing
}

   DSP_TransformWindow_Hann16(OutVector16, InVector16, WindowN);
// OutWindow = 0x0000, 0x095F, 0x2000, 0x36A1, 0x4000, 0x36A1, 0x2000, 0x095F
```

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

DSP_TransformWindow_Hann32 Function

Perform a Hanning window on a vector.

File

dsp.h

C

```
void DSP_TransformWindow_Hann32(int32_t * OutVector, int32_t * InVector, int N);
```

Returns

None.

Description

Function DSP_TransformWindow_Hann32:

void DSP_TransformWindow_Hann32(int32_t *OutVector, int32_t *InVector, int N);

Compute a Hanning Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q31 fractional format. The Hanning Window follows the equation:

Window(n) = 0.5 - 0.5 * COS(2*Pi*n/N) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

```
int32_t OutVector32[8]={0};
int  WindowN = 8;

for (i=0;i<WindowN;i++)
{
   InVector32[i]= 0x40000000;  // constant 0.5 for functional testing
}

   DSP_TransformWindow_Hann32(OutVector32, InVector32, WindowN);
// OutWindow = 0x00000000, 0x095F61C0, 0x20000000, 0x36A09E80, 0x40000000,
//                       0x36A09E80, 0x20000000, 0x095F61C0
```

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

DSP_TransformWindow_Kaiser16 Function

Perform a Kaiser window on a vector.

File

dsp.h

C

```
void DSP_TransformWindow_Kaiser16(int16_t * OutVector, int16_t * InVector, int N);
```

Returns

None.

Description

Function DSP_TransformWindow_Kaiser16:

void DSP_TransformWindow_Kaiser16(int16_t *OutVector, int16_t *InVector, int N);

Compute a Kaiser Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q15 fractional format. The Kaiser Window follows the equation:

Window(n) = 0.402 - 0.498 * COS(2*Pi*n/N) + 0.098 * cos(4*Pi*n/N) + 0.001 * cos(6*Pi*n/N) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

```
int16_t OutVector16[8]={0};
int  WindowN = 8;

for (i=0;i<WindowN;i++)
{
   InVector16[i]= 0x4000;  // constant 0.5 for functional testing
}

   DSP_TransformWindow_Kaiser16(OutVector16, InVector16, WindowN);
// OutWindow = 0x0031, 0x0325, 0x1375, 0x304F, 0x3FCF, 0x304F, 0x1375, 0x0325
```

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

DSP_TransformWindow_Kaiser32 Function

Perform a Kaiser window on a vector.

File

dsp.h

C

```
void DSP_TransformWindow_Kaiser32(int32_t * OutVector, int32_t * InVector, int N);
```

Returns

None.

Description

Function DSP_TransformWindow_Kaiser32:

void DSP_TransformWindow_Kaiser32(int32_t *OutVector, int32_t *InVector, int N);

Compute a Kaiser Window on the first N samples of the input vector, InVector. The output is stored in OutVector. Operations are performed at higher resolution and rounded for the most accuracy possible. Input and output values are in Q31 fractional format. The Kaiser Window follows the equation:

Window(n) = 0.402 - 0.498 * COS(2*Pi*n/N) + 0.098 * cos(4*Pi*n/N) + 0.001 * cos(6*Pi*n/N) where n is the window sample number N is the total number of samples The functional output computes WinVector(n) = Window(n) * InVector(n)

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

```
int32_t OutVector32[8]={0};
int  WindowN = 8;

for (i=0;i<WindowN;i++)
{
   InVector32[i]= 0x40000000;  // constant 0.5 for functional testing
}

   DSP_TransformWindow_Kaiser32(OutVector32, InVector32, WindowN);
// OutWindow = 0x003126F6, 0x032555C8, 0x1374BCA0, 0x304F66C0, 0x3FCED900,
//                       0x304F66C0, 0x1374BCA0, 0x032555C8
```

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

DSP_TransformWinInit_Bart16 Function

Create a Bartlett window.

File

dsp.h

C

```
void DSP_TransformWinInit_Bart16(int16_t * OutWindow, int N);
```

Returns

None.

Description

Function DSP_TransformWinInit_Bart16:

void DSP_TransformWinInit_Bart16(int16_t *OutWindow, int N);

Create a N-element Bartlett (Triangle) Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q15 fractional format. The Bartlett Window follows the equation:

Window(n) = 1 - (abs(2*n - N)/N) where n is the window sample number N is the total number of samples

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

DSP_TransformWinInit_Bart32 Function

Create a Bartlett window.

File

dsp.h

C

```
void DSP_TransformWinInit_Bart32(int32_t * OutWindow, int N);
```

Returns

None.

Description

Function DSP_TransformWinInit_Bart32:

void DSP_TransformWinInit_Bart32(int32_t *OutWindow, int N);

Create a N-element Bartlett (Triangle) Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values Q31 fractional format. The Bartlett Window follows the equation:

Window(n) = 1 - (abs(2*n - N)/N) where n is the window sample number N is the total number of samples

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

DSP_TransformWinInit_Black16 Function

Create a Blackman window.

File

dsp.h

C

```
void DSP_TransformWinInit_Black16(int16_t * OutWindow, int N);
```

Returns

None.

Description

Function DSP_TransformWinInit_Black16:

void DSP_TransformWinInit_Black16(int16_t *OutWindow, int N);

Create a N-element Blackman Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are Q16 fractional format. The Blackman Window follows the equation:

Window(n) = 0.42659 - 0.49656 * COS(2*Pi*n/(N-1)) + 0.076849 * COS(4*Pi*n/(N-1)) where n is the window sample number N is the total number of samples

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

DSP_TransformWinInit_Black32 Function

Create a Blackman window.

File

dsp.h

C

```
void DSP_TransformWinInit_Black32(int32_t * OutWindow, int N);
```

Returns

None.

Description

Function DSP_TransformWinInit_Black32:

void DSP_TransformWinInit_Black32(int32_t *OutWindow, int N);

Create a N-element Blackman Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are Q31 fractional format. The Blackman Window follows the equation:

Window(n) = 0.42659 - 0.49656 * COS(2*Pi*n/(N-1)) + 0.076849 * COS(4*Pi*n/(N-1)) where n is the window sample number N is the total number of samples

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

DSP_TransformWinInit_Cosine16 Function

Create a Cosine (Sine) window.

File

dsp.h

C

```
void DSP_TransformWinInit_Cosine16(int16_t * OutWindow, int N);
```

Returns

None.

Description

Function DSP_TransformWinInit_Cosine16:

void DSP_TransformWinInit_Cosine16(int16_t *OutWindow, int N);

Create a N-element Cosine Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q15 fractional format. The Cosine Window follows the equation:

Window(n) = SIN(Pi*n/(N-1)) where n is the window sample number N is the total number of samples

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

DSP_TransformWinInit_Cosine32 Function

Create a Cosine (Sine) window.

File

dsp.h

C

```
void DSP_TransformWinInit_Cosine32(int32_t * OutWindow, int N);
```

Returns

None.

Description

Function DSP_TransformWinInit_Cosine32:

void DSP_TransformWinInit_Cosine32(int32_t *OutWindow, int N);

Create a N-element Cosine Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q31 fractional format. The Cosine Window follows the equation:

Window(n) = SIN(Pi*n/(N-1)) where n is the window sample number N is the total number of samples

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

DSP_TransformWinInit_Hamm16 Function

Create a Hamming window.

File

dsp.h

C

```
void DSP_TransformWinInit_Hamm16(int16_t * OutWindow, int N);
```

Returns

None.

Description

Function DSP_TransformWinInit_Hamm16:

void DSP_TransformWinInit_Hamm16(int16_t *OutWindow, int N);

Create a N-element Hamming Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q15 fractional format. The Hamming Window follows the equation:

Window(n) = 0.54 - 0.46 * COS(2*Pi*n/N) where n is the window sample number N is the total number of samples

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

DSP_TransformWinInit_Hamm32 Function

Create a Hamming window.

File

dsp.h

C

```
void DSP_TransformWinInit_Hamm32(int32_t * OutWindow, int N);
```

Returns

None.

Description

Function DSP_TransformWinInit_Hamm32:

void DSP_TransformWinInit_Hamm32(int32_t *OutWindow, int N);

Create a N-element Hamming Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q31 fractional format. The Hamming Window follows the equation:

Window(n) = 0.54 - 0.46 * COS(2*Pi*n/N) where n is the window sample number N is the total number of samples

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

DSP_TransformWinInit_Hann16 Function

Create a Hanning window.

File

dsp.h

C

```
void DSP_TransformWinInit_Hann16(int16_t * OutWindow, int N);
```

Returns

None.

Description

Function DSP_TransformWinInit_Hann16:

void DSP_TransformWinInit_Hann16(int16_t *OutWindow, int N);

Create a N-element Hanning Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q15 fractional format. The Hanning Window follows the equation:

Window(n) = 0.5 - 0.5 * COS(2*Pi*n/N) where n is the window sample number N is the total number of samples

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

DSP_TransformWinInit_Hann32 Function

Create a Hanning window.

File

dsp.h

C

```
void DSP_TransformWinInit_Hann32(int32_t * OutWindow, int N);
```

Returns

None.

Description

Function DSP_TransformWinInit_Hann32:

void DSP_TransformWinInit_Hann32(int32_t *OutWindow, int N);

Create a N-element Hanning Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q31 fractional format. The Hanning Window follows the equation:

Window(n) = 0.5 - 0.5 * COS(2*Pi*n/N) where n is the window sample number N is the total number of samples

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

DSP_TransformWinInit_Kaiser16 Function

Create a Kaiser window.

File

dsp.h

C

```
void DSP_TransformWinInit_Kaiser16(int16_t * OutWindow, int N);
```

Returns

None.

Description

Function DSP_TransformWinInit_Kaiser16:

void DSP_TransformWinInit_Kaiser16(int16_t *OutWindow, int N);

Create a N-element Kaiser Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q15 fractional format. The Kaiser Window follows the equation:

Window(n) = 0.402 - 0.498 * COS(2*Pi*n/N) + 0.098 * cos(4*Pi*n/N) + 0.001 * cos(6*Pi*n/N) where n is the window sample number N is the total number of samples

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the

window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int16_t)
N	number of samples (int)

DSP_TransformWinInit_Kaiser32 Function

Create a Kaiser window.

File

dsp.h

C

```
void DSP_TransformWinInit_Kaiser32(int32_t * OutWindow, int N);
```

Returns

None.

Description

Function DSP_TransformWinInit_Kaiser32:

void DSP_TransformWinInit_Kaiser32(int32_t *OutWindow, int N);

Create a N-element Kaiser Window, and store the output to OutWindow. Operations are performed at higher resolution floating point, and rounded for the most accuracy possible. Output values are in Q31 fractional format. The Kaiser Window follows the equation:

Window(n) = 0.402 - 0.498 * COS(2*Pi*n/N) + 0.098 * cos(4*Pi*n/N) + 0.001 * cos(6*Pi*n/N) where n is the window sample number N is the total number of samples

Remarks

This function is performed in C. The function may be optimized for the library. It is dependent on the floating point math library. The functional window is an intermediate result that needs to be multiplied by an input vector prior to FFT processing. Because of significant processing time the window need only be computed once and the multiply of the (window * input) vector done during recurring loop processing.

Preconditions

N must be a positive number. OutWindow must be declared with N elements or larger.

Example

Parameters

Parameters	Description
OutWindow	pointer to output array of elements (int32_t)
N	number of samples (int)

e) Vector Math Functions

DSP_VectorAbs16 Function

Calculate the absolute value of a vector.

File

dsp.h

C

```
void DSP_VectorAbs16(int16_t * outdata, int16_t * indata, int N);
```

Returns

None.

Description

Function DSP_VectorAbs16:

void DSP_VectorAbs16(int16_t *outdata, int16_t *indata, int N);

Computes the absolute value of each element of indata and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q15 fractional format. outdata[i] filled with the absolute value of elements of indata

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Example

```
int16_t *pOutdata;
int16_t outVal[8];
int16_t inBufTest[16] = {-5,2,-3,4,-1,0,-2,-8,-21,21,10,100,200,127,-127,-2};
int Num = 8;

pOutdata = &outVal;

DSP_VectorAbs16(pOutdata, inBufTest, Num);

// outVal[i] = {5,2,3,4,1,0,2,8}
```

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int16_t)
indata	pointer to input array of 16-bit elements (int16_t)
N	number of samples (int)

DSP_VectorAbs32 Function

Calculate the absolute value of a vector.

File

dsp.h

C

```
void DSP_VectorAbs32(int32_t * outdata, int32_t * indata, int N);
```

Returns

None.

Description

Function DSP_VectorAbs32:

void DSP_VectorAbs32(int32_t *outdata, int32_t *indata, int N);

Computes the absolute value of each element of indata and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q31 fractional format. outdata[i] filled with N elements of abs(indata[i])

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Example

```
int16_t *pOutdata;
int32_t outVal[8];
int32_t inBufTest[16] = {-5,2,-3,4,-1,0,-2,-8,-21,21,10,100,200,127,-127,-2};
int Num = 8;

pOutdata = &outVal;

DSP_VectorAbs32(pOutdata, inBufTest, Num);

// outVal[i] = {5,2,3,4,1,0,2,8}
```

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int32_t)
indata	pointer to input array of 16-bit elements (int32_t)
N	number of samples (int)

DSP_VectorAdd16 Function

Calculate the sum of two vectors.

File

dsp.h

C

```
void DSP_VectorAdd16(int16_t * outdata, int16_t * indata1, int16_t * indata2, int N);
```

Returns

None.

Description

Function DSP_VectorAdd16:

void DSP_VectorAdd16(int16_t *outdata, int16_t *indata1, int16_t *indata2, int N);

Computes the sum value of each element of indata1 + indata2 and stores it to outdata. The number of samples to process is given by the parameter N. Data is in the Q15 fractional format. outdata[i] filled with N elements of indata1[i] + indata2[i]

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Example

```
int16_t *pOutdata;
int16_t outVal[8];
int16_t inBufTest[16] = {-5,2,-3,4,-1,0,-2,-8,-21,21,10,100, 200, 127,-127,-2};
int16_t inBuf2[16]=     { 1,2, 3,4, 5,6, 7, 8, 9, 10,-1,-100,-127,127,-7,  0};
int Num = 8;

pOutdata = &outVal;

DSP_VectorAdd16(pOutdata, inBufTest, inBuf2, Num);

// outVal[i] = inBufTest[i] + inBuf2[i] = {-4,4,0,8,4,6,5,0}
```

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int16_t)
indata1	pointer to input array of 16-bit elements (int16_t)
indata2	pointer to input array of 16-bit elements (int16_t)

N	number of samples (int)

DSP_VectorAdd32 Function

Calculate the sum of two vectors.

File

dsp.h

C

```
void DSP_VectorAdd32(int32_t * outdata, int32_t * indata1, int32_t * indata2, int N);
```

Returns

None.

Description

Function DSP_VectorAdd32:

void DSP_VectorAdd32(int32_t *outdata, int32_t *indata1, int32_t *indata2, int N);

Computes the sum value of each element of indata1 + indata2 and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q31 fractional format. outdata[i] filled with N elements of indata1[i] + indata2[i]

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int16_t *pOutdata;
int32_t outVal[8];
int32_t inBufTest[16] = {-5,2,-3,4,-1,0,-2,-8,-21,21,10,100, 200, 127,-127,-2};
int32_t inBuf2[16]=     { 1,2, 3,4, 5,6, 7, 8, 9, 10,-1,-100,-127,127,-7,  0};
int Num = 8;

pOutdata = &outVal;

DSP_VectorAdd32(pOutdata, inBufTest, inBuf2, Num);

// outVal[i] = inBufTest[i] + inBuf2[i] = {-4,4,0,8,4,6,5,0}
```

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int32_t)
indata1	pointer to input array of 16-bit elements (int32_t)
indata2	pointer to input array of 16-bit elements (int32_t)
N	number of samples

DSP_VectorAddc16 Function

Calculate the sum of a vector and a constant.

File

dsp.h

C

```
void DSP_VectorAddc16(int16_t * outdata, int16_t * indata, int16_t c, int N);
```

Returns

None.

Description

Function DSP_VectorAddc16:

void DSP_VectorAddc16(int16_t *outdata, int16_t *indata, int16_t c, int N);

Computes the sum value of each element of (indata + c) and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q15 fractional format. outdata[i] filled with N elements of indata[i] + c

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Example

```
int16_t *pOutdata;
int16_t outVal[8];
int16_t inBufTest[16] = {-5,2,-3,4,-1,0,-2,-8,-21,21,10,100, 200, 127,-127,-2};
int16_t constValue =  3;
int Num = 8;

pOutdata = &outVal;

DSP_VectorAddc16(pOutdata, inBufTest, constValue, Num);

// outVal[i] = inBufTest[i] + constValue = {-2,5,0,7,2,3,1,-5}
```

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int16_t)
indata	pointer to input array of 16-bit elements (int16_t)
c	constant value added to all indata1 elements (int16_t)
N	number of samples (int)

DSP_VectorAddc32 Function

Calculate the sum of a vector and a constant.

File

dsp.h

C

```
void DSP_VectorAddc32(int32_t * outdata, int32_t * indata, int32_t c, int N);
```

Returns

None.

Description

Function DSP_VectorAddc32:

void DSP_VectorAddc32(int32_t *outdata, int32_t *indata, int32_t c, int N);

Computes the sum value of each element of (indata + c) and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q31 fractional format. outdata[i] filled with N elements of indata1[i] + c

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int16_t *pOutdata;
int32_t outVal[8];
int32_t inBufTest[16] = {-5,2,-3,4,-1,0,-2,-8,-21,21,10,100, 200, 127,-127,-2};
```

```
int32_t constValue =  3;
int Num = 8;

pOutdata = &outVal;

DSP_VectorAddc32(pOutdata, inBufTest, constValue, Num);

// outVal[i] = inBufTest[i] + constValue = {-2,5,0,7,2,3,1,-5}
```

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int32_t)
indata	pointer to input array of 16-bit elements (int32_t)
c	constant value added to all indata1 elements (int32_t)
N	number of samples (int)

DSP_VectorAutocorr16 Function

Computes the Autocorrelation of a Vector.

File

dsp.h

C

```
void DSP_VectorAutocorr16(int16_t * outCorr, int16_t * inVector, int N, int K);
```

Returns

None.

Description

Function DSP_VectorAutocorr16:

void DSP_VectorAutocorr16(int16_t *outCorr, int16_t *inVector, int N, int K);

Calculates the autocorrelation, with a lag of 1 to K, on the first N elements of inVector and returns the 16-bit scalar result in outCorr. The autocorrelation is calculated from other statistical equations including mean and variance. While in some cases these equations exist inside the DSP library, the functions are executed in a serial fashion within this code to provide enhanced performance. The unbiased function has the form -

mean (M) = sum[0..N-1](x(n) / N) variance (V) = sum[0..N-1]((x(n) - M)^2) / (N-1) autocovariance (ACV)[k] = sum[0..(N-k)]((x(n) - M) * (x(n+k) - M) / (N-k)) autocorrelation (AC)[k] = CV[k] / V where N is the number of vector elements, n is the index of those elements x(n) is a single element in the input vector M is the mean of the N elements of the vector k is the lag or series index

The output of the function will return K elements, and the outCorr array should be sized to accept those 16-bit results. The outputs correspond to k=1, k=2, .., k=K delay states. The function returns a 16-bit value in rounded, saturated Q15 format.

Input values of the vector and output scalar value is Q15 fractional format. This format has data that ranges from -1 to 1, and has internal saturation limits of those same values. Some care has been taken to reduce the impact of saturation by adding processing steps to effectively complete the processing in blocks. However, in some extreme cases of data variance it is still possible to reach the saturation limits.

Remarks

This function is optimized with microMIPS and M14KCe ASE DSP instructions. This function is dependent on the LibQ library, and the _LIBQ_Q16Div specifically.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. outCorr must be an array with at least K values. N must be greater than or equal to four and a multiple of four.

Example

```
int  autocvN = 16;                    // N value, number of samples in array
int  autocvLag = 4;                   // Lag value, output shifts to observe
int16_t outAC[16]={0};
int16_t CVIn16[16] = {0x1999, 0xD99A, 0x1000, 0x6000, 0x1999, 0x1999, 0x2666,
     0x3333, 0x1000, 0x6000, 0x1999, 0x1999, 0x2666, 0x3333, 0x1999, 0x0CCC};
//  = { .2, -.3, .125, .75, .2, .2, .3, .4, .125, .75, .2, .2, .3, .4, .2, .1};

DSP_VectorAutocorr16(outAC, CVIn16, autocvN, autocvLag);

// outAC = {0xF406, D46C, 0x098F, 0x191A}
```

```
//    = -0.093567, -0.34045, 0.07468, 0.19611
```

Parameters

Parameters	Description
outCorr	pointer to output array (int16_t)
inVector	pointer to source array of elements (int16_t)
N	number of samples (int)
K	lag value, number of output elements (int)

DSP_VectorBexp16 Function

Computes the maximum binary exponent of a vector.

File

dsp.h

C

```
int DSP_VectorBexp16(int16_t * DataIn, int N);
```

Returns

Binary exponent [log2 multiplier] (int)

Description

Function DSP_VectorBexp16:

int DSP_VectorBexp16(int16_t *DataIn, int N);

Calculates the maximum binary exponent on the first N elements of the input vector DataIn, and stores the integer result. The returned value represents the potential binary scaling of the vector, and may be used with other functions that auto scale their output without saturation. Inputs are given in Q15 fractional data format.

Remarks

None.

Preconditions

N must be a multiple of 2 and greater or equal to 2.

Example

```
int valN = 4;
int16_t  dummy16[valN]={0x3004, 0x00CC, 0xFC04, 0xFFF0};
//                          0.375, 0.0062, -0.0311, -0.00049
int answer;

answer =   DSP_VectorBexp16(dummy16, valN);

// answer = 1, maximum binary gain is 2.
```

Parameters

Parameters	Description
DataIn	pointer to input array of 16-bit elements (int16_t)
N	number of samples (int)

DSP_VectorBexp32 Function

Computes the maximum binary exponent of a vector.

File

dsp.h

C

```
int DSP_VectorBexp32(int32_t * DataIn, int N);
```

Returns

Binary exponent [log2 multiplier] (int)

Description

Function DSP_VectorBexp32:

int DSP_VectorBexp32(int32_t *DataIn, int N);

Calculates the maximum binary exponent on the first N elements of the input vector DataIn, and stores the integer result. The returned value represents the potential binary scaling of the vector, and may be used with other functions that auto scale their output without saturation. Inputs are given in Q31 fractional data format.

Remarks

None.

Preconditions

None.

Example

```
int valN=4;
int32_t  datInput32[4]={0xFF000000, 0x07000000,0x000CCCCC, 0x08000000};
//                          -0.007183, 0.054688, 0.0003906, 0.0625
int answer32;

answer32 =   DSP_VectorBexp32(datInput32, valN);

// answer = 3, maximum binary gain is 8.
```

Parameters

Parameters	Description
DataIn	pointer to input array of 16-bit elements (int32_t)
N	number of samples (int)

DSP_VectorChkEqu32 Function

Compares two input vectors, returns an integer '1' if equal, and '0' if not equal.

File

dsp.h

C

```
int DSP_VectorChkEqu32(int32_t * indata1, int32_t * indata2, int N);
```

Returns

(int) - '1' if vectors are equal, '0' if vectors are not equal

Description

Function DSP_VectorChkEqu32:

int DSP_VectorChkEqu32(int32_t *indata1, int32_t *indata2, int N);

Compares the first N values of indata1 to the same elements of indata2. The comparison requires that all numbers be in Q31 fractional data format. Returns the integer value '1' if all numbers are equal, and '0' if they are not equal. N must be greater than or equal to four and a multiple of four, or it will be truncated to the nearest multiple of four.

Remarks

None.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int     outCheck;
int     Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
```

```
                            0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                  1,      -1,       0.9,    0.8,    0.2,    0.5,    1,      -0.6
int32_t inBufTestB[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                            0x80000000, 0x40000000, 0x7FFFFFFF, 0x20000000};
//                  1,      -1,       0.9,    0.8,    -1,    0.5,    1,      0.25

outCheck = DSP_VectorChkEqu32(inBufTestA, inBufTestB, Num);

// outCheck = 1  // true for first 4 numbers of series
```

Parameters

Parameters	Description
indata1	pointer to input array 1 of elements (int32_t)
indata2	pointer to input array 2 of elements (int32_t)
N	number of samples (int)

DSP_VectorCopy Function

Copies the elements of one vector to another.

File

dsp.h

C

```
void DSP_VectorCopy(int32_t * outdata, int32_t * indata, int N);
```

Returns

None.

Description

Function DSP_VectorCopy:

void DSP_VectorCopy(int32_t *outdata, int32_t *indata, int N);

Fills the first N values of an input vector outdata with the elements from indata. N must be a multiple of four and greater than or equal to four or it will be truncated to the nearest multiple of four. The vector result and the scalar value to fill are both Q31 fractional format.

Remarks

None.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int     Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                            0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                  1,      -1,       0.9,    0.8,    0.2,    0.5,    1,      -0.6
int32_t inBufTestB[8]={0x00000000, 0x7FFFFFFF, 0x40000000, 0x0CCCCCCC,
                            0x40000000, 0x60000000, 0x80000000, 0x20000000};
//                  0,      1,       0.5,    0.1,    0.75,    0.5,    -1,      0.25

DSP_VectorCopy(inBufTestA, inBufTestB, Num);

// inBufTestA =  {0x00000000, 0x7FFFFFFF, 0x40000000, 0x0CCCCCCC,
//           0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334} // first 4 values copied
```

Parameters

Parameters	Description
outdata	pointer to destination array of values (int32_t)
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

DSP_VectorCopyReverse32 Function

Reverses the order of elements in one vector and copies them into another.

File

dsp.h

C

```
void DSP_VectorCopyReverse32(int32_t * outdata, int32_t * indata, int N);
```

Returns

None.

Description

Function DSP_VectorCopyReverse32:

void DSP_VectorCopyReverse32(int32_t *outdata, int32_t *indata, int N);

Fills the first N values of an input vector Outdata with the reverse elements from INDATA. N must be a multiple of 4 and greater than 4 or will be truncated to the nearest multiple of 4. The vectors are both Q31 fractional format.

Remarks

None.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int     Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                  1,     -1,      0.9,    0.8,    0.2,   0.5,    1,     -0.6
int32_t inBufTestB[8]={0x00000000, 0x7FFFFFFF, 0x40000000, 0x0CCCCCCC,
                       0x40000000, 0x60000000, 0x80000000, 0x20000000};
//                  0,     1,      0.5,    0.1,    0.75,   0.5,    -1,     0.25

DSP_VectorCopyReverse32(inBufTestA, inBufTestB, Num);

// inBufTestA =  {0x0CCCCCCC, 0x40000000, 0x7FFFFFFF, 0x00000000,
//           0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334}
//   first 4 values copied reverse order
```

Parameters

Parameters	Description
outdata	pointer to destination array of values (int32_t)
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

DSP_VectorDivC Function

Divides the first N elements of inVector by a constant divisor, and stores the result in outVector.

File

dsp.h

C

```
void DSP_VectorDivC(_Q16 * outVector, _Q16 * inVector, _Q16 divisor, int N);
```

Returns

None.

Description

Function DSP_VectorDivC:

void DSP_VectorDivC(_Q16 *outVector, _Q16 *inVector, _Q16 divisor, int N);

Divides each element of the first N elements of inVector by a constant, divisor. The output is stored to outVector. Both vectors and the scalar are _Q16 format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. If values exceed maximum or minimum they will saturate to the maximum or minimum respectively.

Remarks

This function uses the Microchip PIC32MZ LibQ library to function. The user must include that library and header file into the design in order to operate this function. For more information on the Div function see the LibQ documentation for _LIBQ_Q16Div.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. This function uses the Div function from the LibQ library. That library must be compiled as part of the project.

Example

```
int divNum = 4;
_Q16 divScalar = 0x00020000;  // 2.0
_Q16 inDivVec[8] = {0x08000000, 0xfffc0000,0x00024000,0x00100000,0x00038000,
       0x00400000,0xfffe0000,0x00058000};
//     2048, -4, 2.25, 16, 3.5, 64, -2, 5.5
_Q16 outDivVec[8] = {0};

DSP_VectorDivC(outDivVec, inDivVec,  divScalar, divNum);

// outDivVec =  0x04000000, 0xFFFFE0000, 0x00012000, 0x00080000, 0, 0, 0, 0
//                  1024.0,        -2.0,      1.125,       16.0, 0, 0, 0, 0
```

Parameters

Parameters	Description
outdata	pointer to destination array of elements (_Q16)
indata	pointer to source array of elements (_Q16)
divisor	scalar divisor for the input vector (_Q16)
N	number of samples (int)

DSP_VectorDotp16 Function

Computes the dot product of two vectors, and scales the output by a binary factor.

File

dsp.h

C

```
int16_t DSP_VectorDotp16(int16_t * indata1, int16_t * indata2, int N, int scale);
```

Returns

int16_t - scaled output of calculation, Q15 format

Description

Function DSP_VectorDotp16:

int16_t DSP_VectorDotp16(int16_t *indata1, int16_t *indata2, int N, int scale);

Calculates the dot product of two input vectors, and scales the output. Function will saturate if it exceeds maximum or minimum values. Scaling is done by binary shifting, after accumulation in a 32 bit register. All calculations are done in Q15 fractional format. return = 1/(2^scale) * sum(indata1[i] * indata2[i])

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Example

```
int16_t inBufMultA[8]={0x7FFF, 0x8000, 0x7333, 0x6666, 0x1999, 0x4000, 0x7FFF, 0xB334};
//                      1,     -1,      0.9,    0.8,    0.2,    0.5,    1,      -0.6
int16_t inBufMultB[8]={0x0CCD, 0x0CCD, 0x4000, 0xC000, 0xE667, 0x4000, 0x0000, 0x0CCD};
```

```
//                          0.1,    0.1,    0.5,   -0.5,   -0.2,   0.5,    0,      0.1
int Num = 8;
int scaleVal = 2;
int16_t outScalar;

int Num = 8;

outScalar = DSP_VectorDotp16(inBufMultA, inBufMultB, Num, scaleVal);

// outScalar = 1/(2^scaleVal)*(inBufMultA[] dot inBufMultB[]) =
//   (1/4) * (0.1 + -0.1 + 0.45 + -0.4 + -0.04 + 0.25 + 0 + -0.06) = 0.25 * 0.20 = 0.05
//      = (int16_t)0x0666
```

Parameters

Parameters	Description
indata1	pointer to input array of 16-bit elements (int16_t)
indata2	pointer to input array of 16-bit elements (int16_t)
scale	number of bits to shift return right (int)
N	number of samples (int)

DSP_VectorDotp32 Function

Computes the dot product of two vectors, and scales the output by a binary factor

File

dsp.h

C

```
int32_t DSP_VectorDotp32(int32_t * indata1, int32_t * indata2, int N, int scale);
```

Returns

int16_t - scaled output of calculation, Q31 format

Description

Function DSP_VectorDotp32:

int32_t DSP_VectorDotp32(int32_t *indata1, int32_t *indata2, int N, int scale);

Calculates the dot product of two input vectors, and scales the output. Function will saturate if it exceeds maximum or minimum values. Scaling is done by binary shifting, after calculation of the result. All calculations are done in Q31 fractional format. return = 1/(2^scale) * sum(indata1[i] * indata2[i])

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int32_t inBufMultA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                 1,     -1,      0.9,    0.8,    0.2,   0.5,    1,     -0.6
int32_t inBufMultB[8]={0x0CCCCCCD, 0x0CCCCCCD, 0x40000000, 0xC0000000,
                       0xE6666667, 0x40000000, 0x00000000, 0x0CCCCCCD};
//                 0.1,    0.1,    0.5,   -0.5,   -0.2,   0.5,    0,      0.1
int Num = 8;
int scaleVal = 2;
int32_t outScalar;

int Num = 8;

outScalar = DSP_VectorDotp32(inBufMultA, inBufMultB, Num, scaleVal);

// outScalar = 1/(2^scaleVal)*(inBufMultA[] dot inBufMultB[]) =
//   (1/4) * (0.1 + -0.1 + 0.45 + -0.4 + -0.04 + 0.25 + 0 + -0.06) = 0.25 * 0.20 = 0.05
//      = (int32_t)0x06666666
```

Parameters

Parameters	Description
indata1	pointer to input array of 16-bit elements (int32_t)
indata2	pointer to input array of 16-bit elements (int32_t)
scale	number of bits to shift return right (int)
N	number of samples (int)

DSP_VectorExp Function

Computes the EXP (e^x) of the first N elements of inVector, and stores the result in outVector.

File

dsp.h

C

```
void DSP_VectorExp(_Q16 * outVector, _Q16 * inVector, int N);
```

Returns

None.

Description

Function DSP_VectorExp:

void DSP_VectorExp(_Q16 *outVector, _Q16 *inVector, int N);

Computes the Exp value, e to the power of X, on the first N elements of inVector. The output is stored to outVector. Both vectors are _Q16 format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. If values exceed maximum or minimum they will saturate to the maximum or zero respectively.

Remarks

Inclusion of the LibQ header file and library is mandatory to use this function.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. This function uses the Exp and Div functions from the LibQ library. That library must be compiled as part of the project.

Example

```
int expNum = 4;
_Q16 inExpVec[8] = {0x00010000, 0xffff0000,0x00020000,0x00030000,0x00038000,
            0x00040000,0xfffe0000,0x00058000};
//          1.0,   -1.0,   2.0,   3.0,    3.5,    4.0,   -2.0,   5.5
_Q16 outExpVec[8] = {0};

DSP_VectorExp(outExpVec, inExpVec, expNum);

// outExpVec = 0x0002B7E1, 0x00005E2D, 0x00076399, 0x001415E6, 0, 0, 0, 0
//                2.71828,    0.26787,     7.3891,    20.0855, 0, 0, 0, 0
```

Parameters

Parameters	Description
outdata	pointer to destination array of elements (_Q16)
indata	pointer to source array of elements (_Q16)
N	number of samples (int)

DSP_VectorFill Function

Fills an input vector with scalar data.

File

dsp.h

C

```
void DSP_VectorFill(int32_t * indata, int32_t data, int N);
```

Returns

None.

Description

Function DSP_VectorFill:

void DSP_VectorFill(int32_t *indata, int32_t data, int N);

Fills the first N values of an input vector indata with the value data. N must be a multiple of four and greater than or equal to four or it will be truncated to the nearest multiple of four. The vector result and the scalar value to fill are both Q31 fractional format.

Remarks

None.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int32_t fillValue = 0x3FFFFFFF;
int     Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                  1,     -1,      0.9,    0.8,    0.2,   0.5,    1,     -0.6

DSP_VectorFill(inBufTestA, fillValue, Num);

// inBufTestA =  {0x3FFFFFFF, 0x3FFFFFFF, 0x3FFFFFFF, 0x3FFFFFFF,
//           0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334} // first 4 values filled
```

Parameters

Parameters	Description
indata	pointer to source array of elements (int32_t)
data	scalar value to fill the array (int32_t)
N	number of samples (int)

DSP_VectorLn Function

Computes the Natural Log, Ln(x), of the first N elements of inVector, and stores the result in outVector.

File

dsp.h

C

```
void DSP_VectorLn(_Q4_11 * outVector, _Q16 * inVector, int N);
```

Returns

None.

Description

Function DSP_VectorLn:

void DSP_VectorLn(_Q4_11 *outVector, _Q16 *inVector, int N);

Computes the Ln(x) value, on the first N elements of inVector. The output is stored to outVector. Input vector is _Q16 format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. The output vector is reduced resolution Q4.11 format, which is a 16-bit integer format with 11 bits representing the fractional resolution. If values exceed maximum or minimum they will saturate to the maximum or zero respectively.

Remarks

This function uses the Microchip PIC32MZ LibQ library to function. The user must include that library and header file into the design in order to operate this function. For more information on the Ln function see the LibQ documentation for _LIBQ_Q4_11_ln_Q16. A negative number input will return a saturated negative value (0x8000).

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. This function uses the Ln function from the LibQ library. That library must be compiled as part of the project.

Example

```
int lnNum = 4;
_Q16 inLnVal[8] = {0x40000000, 0xffff0000,0x00020000,0x00100000,0x00038000,
       0x00400000,0xfffe0000,0x00058000};
//          16384.0,  -1.0,  2.0,  16.0,   3.5,   64.0,  -2.0,  5.5
_Q4_11 outLnVal[8] = {0};

DSP_VectorLn(outLnVal, inLnVal,  lnNum);

// outLnVal =     0x4DA2,        0x8000,   0x058C,   0x162E, 0, 0, 0, 0
//                 9.704,  sat negative,   0.6934,    2.772, 0, 0, 0, 0
```

Parameters

Parameters	Description
outdata	pointer to destination array of elements (_Q16)
indata	pointer to source array of elements (_Q4_11)
N	number of samples (int)

DSP_VectorLog10 Function

Computes the Log10(x), of the first N elements of inVector, and stores the result in outVector.

File

dsp.h

C

```
void DSP_VectorLog10(_Q3_12 * outVector, _Q16 * inVector, int N);
```

Returns

None.

Description

Function DSP_VectorLog10:

void DSP_VectorLog10(_Q3_12 *outVector, _Q16 *inVector, int N);

Computes the Log10(x) value, on the first N elements of inVector. The output is stored to outVector. Input vector is _Q16 format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. The output vector is reduced resolution Q3.12 format, which is a 16-bit integer format with 12 bits representing the fractional resolution. If values exceed maximum or minimum they will saturate to the maximum or zero respectively.

Remarks

This function uses the Microchip PIC32MZ LibQ library to function. The user must include that library and header file into the design in order to operate this function. For more information on the Log10 function see the LibQ documentation for _LIBQ_Q3_12_log10_Q16. A negative number input will return a saturated negative value (0x8000).

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. This function uses the Log10 function from the LibQ library. That library must be compiled as part of the project.

Example

```
int logNum = 4;
_Q16 inLogVal[8] = {0x40000000, 0xffff0000,0x00020000,0x00100000,0x00038000,
                    0x00400000,0xfffe0000,0x00058000};
//          16384.0,  -1.0,  2.0,  16.0,   3.5,   64.0,  -2.0,  5.5
_Q3_12 outLogVal[8] = {0};

DSP_VectorLog10(outLogVal, inLogVal,  logNum);

// outLogVal =    0x436E,        0x8000,    0x04D1,   0x1344, 0, 0, 0, 0
```

```
//                4.2144,  sat negative,   0.3010,   1.2041, 0, 0, 0, 0
```

Parameters

Parameters	Description
outdata	pointer to destination array of elements (_Q16)
indata	pointer to source array of elements (_Q3_12)
N	number of samples (int)

DSP_VectorLog2 Function

Computes the Log2(x) of the first N elements of inVector, and stores the result in outVector.

File

dsp.h

C

```
void DSP_VectorLog2(_Q5_10 * outVector, _Q16 * inVector, int N);
```

Returns

None.

Description

Function DSP_VectorLog2:

void DSP_VectorLog2(_Q5_10 *outVector, _Q16 *inVector, int N);

Computes the Log2 value, where log2(x) = ln(x) * log2(e), on the first N elements of inVector. The output is stored to outVector. Input vector is _Q16 format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. The output vector is reduced resolution Q5.10 format, which is a 16-bit integer format with 10 bits representing the fractional resolution. If values exceed maximum or minimum they will saturate to the maximum or zero respectively.

Remarks

This function uses the Microchip PIC32MZ LibQ library to function. The user must include that library and header file into the design in order to operate this function. For more information on the Log2 function see the LibQ documentation for _LIBQ_Q5_10_log2_Q16. A negative number input will return a saturated negative value (0x8000).

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. This function uses the Log2 function from the LibQ library. That library must be compiled as part of the project.

Example

```
int log2Num = 4;
_Q16 inLog2Val[8] = {0x40000000, 0xffff0000,0x00020000,0x00030000,0x00038000,
           0x00040000,0xfffe0000,0x00058000};
//          16384.0,  -1.0,  2.0,  3.0,   3.5,   4.0,  -2.0,  5.5
_Q5_10 outLog2Val[8] = {0};

DSP_VectorLog2(outLog2Val, inLog2Val,  log2Num);

// outLog2Val =    0x3800,        0x8000,   0x0400,   0x0657, 0, 0, 0, 0
//                  14.0,  sat negative,      1.0,    1.585, 0, 0, 0, 0
```

Parameters

Parameters	Description
outdata	pointer to destination array of elements (_Q16)
indata	pointer to source array of elements (_Q5_10)
N	number of samples (int)

DSP_VectorMax32 Function

Returns the maximum value of a vector.

File

dsp.h

C

```
int32_t DSP_VectorMax32(int32_t * indata, int N);
```

Returns

(int32_t) - maximum value within the vector, Q31 format

Description

Function DSP_VectorMax32:

int32_t DSP_VectorMax32(int32_t *indata, int N);

Returns the highest value of the first N elements of the vector indata. The comparison requires that all numbers be in Q31 fractional data format. N must be greater than or equal to four and a multiple of four, or it will be truncated to the nearest multiple of four.

Remarks

None.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and multiple of four.

Example

```
int32_t   outCheck;
int     Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                  1,     -1,      0.9,    0.8,    0.2,   0.5,    1,     -0.6

outCheck = DSP_VectorMax32(inBufTestA, Num);

// outCheck = 0x7FFFFFFF  // first 4 values
```

Parameters

Parameters	Description
indata	pointer to input array of elements (int32_t)
N	number of samples (int)

DSP_VectorMaxIndex32 Function

Returns the index of the maximum value of a vector.

File

dsp.h

C

```
int DSP_VectorMaxIndex32(int32_t * indata, int N);
```

Returns

int - index of the position of the maximum array element

Description

Function DSP_VectorMaxIndex32:

int DSP_VectorMaxIndex32(int32_t *indata, int N);

Returns the index of the highest value of the first N elements of the vector indata. The comparison requires that all numbers be in Q31 fractional data format. N must be greater than or equal to four and a multiple of four, or it will be truncated to the nearest multiple of four.

Remarks

Index values range from 0 .. (n-1).

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int     indexValue;
int     Num = 8;
int32_t inBufTestA[8]={0x3FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                  0.5,     -1,      0.9,    0.8,    0.2,   0.5,    1,     -0.6

indexValue = DSP_VectorMaxIndex32(inBufTestA, Num);

// returnValue =  6 (position corresponding to 0x7FFFFFFF)
```

Parameters

Parameters	Description
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

DSP_VectorMean32 Function

Calculates the mean average of an input vector.

File

dsp.h

C

```
int32_t DSP_VectorMean32(int32_t * indata1, int N);
```

Returns

int32_t - mean average value of the vector

Description

Function DSP_VectorMean32:

int32_t DSP_VectorMean32(int32_t *indata, int N);

Calculates the mean average of the first N elements of the vector indata. The values of indata1 are in Q31 fractional format. The value N must be greater than or equal to four and a multiple of four, or it will be truncated to the nearest multiple of four.

Remarks

None.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int32_t   returnValue;
int     Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                  1,     -1,      0.9,    0.8,    0.2,   0.5,    1,     -0.6

returnValue = DSP_VectorMean32(inBufTestA, Num);

// returnValue =  0x36666666 = (1-1+0.9+0.8)/4 = 0.425
```

Parameters

Parameters	Description
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

DSP_VectorMin32 Function

Returns the minimum value of a vector.

File

dsp.h

C

```
int32_t DSP_VectorMin32(int32_t * input, int N);
```

Returns

(int32_t) - minimum value within the vector, Q31 format

Description

Function DSP_VectorMin32:

int32_t DSP_VectorMin32(int32_t *indata, int N);

Returns the lowest value of the first N elements of the vector indata. The comparison requires that all numbers be in Q31 fractional data format. N must be greater than or equal to four and a multiple of four, or it will be truncated to the nearest multiple of four.

Remarks

None.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and multiple of four.

Example

```
int32_t   outCheck;
int     Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                  1,     -1,      0.9,    0.8,    0.2,   0.5,    1,     -0.6

outCheck = DSP_VectorMin32(inBufTestA, Num);

// outCheck = 0x80000000  // first 4 values
```

Parameters

Parameters	Description
indata	pointer to input array of elements (int32_t)
N	number of samples (int)

DSP_VectorMinIndex32 Function

Returns the index of the minimum value of a vector.

File

dsp.h

C

```
int DSP_VectorMinIndex32(int32_t * indata, int N);
```

Returns

int32_t - mean average value of the vector

Description

Function DSP_VectorMinIndex32:

int DSP_VectorMinIndex32(int32_t *indata, int N);

Returns the relative position index of the lowest value of the first N elements of the vector indata. The comparison requires that all numbers be in Q31 fractional data format. N must be greater than or equal to four and a multiple of four, or it will be truncated to the nearest multiple of four.

Remarks

Index values range from 0 .. (n-1).

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int     indexValue;
int     Num = 8;
int32_t inBufTestA[8]={0x3FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                  0.5,     -1,      0.9,    0.8,    0.2,   0.5,    1,     -0.6

indexValue = DSP_VectorMinIndex32(inBufTestA, Num);

// returnValue =  1 (position corresponding to 0x80000000)
```

Parameters

Parameters	Description
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

DSP_VectorMul16 Function

Multiplication of a series of numbers in one vector to another vector.

File

dsp.h

C

```
void DSP_VectorMul16(int16_t * outdata, int16_t * indata1, int16_t * indata2, int N);
```

Returns

None.

Description

Function DSP_VectorMul16:

void DSP_VectorMul16(int16_t *outdata, int16_t *indata1, int16_t *indata2, int N);

Multiples the value of each element of indata1 * indata2 and stores it to outdata. The number of samples to process is given by the parameter N. Data is in the Q15 fractional format. outdata[i] filled with N elements of indata1[i] * indata2[i]

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Example

```
int16_t *pOutdata;
int16_t outVal[8];
int16_t inBufMultA[8]={0x7FFF, 0x8000, 0x7333, 0x6666, 0x1999, 0x4000, 0x7FFF, 0xB334};
//                        1,     -1,      0.9,    0.8,    0.2,   0.5,    1,     -0.6
int16_t inBufMultB[8]={0x0CCD, 0x0CCD, 0x4000, 0xC000, 0xE667, 0x4000, 0x0000, 0x0CCD};
//                        0.1,    0.1,    0.5,   -0.5,   -0.2,   0.5,    0,      0.1
int Num = 8;

pOutdata = &outVal;

DSP_VectorMul16(pOutdata, inBufMultA, inBufMultB, Num);

// outVal[i] = inBufTest[i] * inBuf2[i] =
//    {0x0CCD, 0xF333, 0x399A, 0xCCCD, 0xFAE2, 0x2000, 0x0000, 0xF852}
//      0.1,   -0.1,   0.45,  -0.4,   -0.04,   0.25,    0,     -0.06
```

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int16_t)
indata1	pointer to input array of 16-bit elements (int16_t)

indata2	pointer to input array of 16-bit elements (int16_t)
N	number of samples (int)

DSP_VectorMul32 Function

Multiplication of a series of numbers in one vector to another vector.

File

dsp.h

C

```
void DSP_VectorMul32(int32_t * outdata, int32_t * indata1, int32_t * indata2, int N);
```

Returns

None.

Description

Function DSP_VectorMul32:

void DSP_VectorMul32(int32_t *outdata, int32_t *indata1, int32_t *indata2, int N);

Multiples the value of each element of indata1 * indata2 and stores it to outdata. The number of samples to process is given by the parameter N. Data is in the Q31 fractional format. outdata[i] filled with N elements of indata1[i] * indata2[i]

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int16_t *pOutdata;
int32_t outVal[8];
int32_t inBufMultA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                  1,      -1,       0.9,     0.8,     0.2,    0.5,     1,      -0.6
int32_t inBufMultB[8]={0x0CCCCCCD, 0x0CCCCCCD, 0x40000000, 0xC0000000,
                       0xE6666667, 0x40000000, 0x00000000, 0x0CCCCCCD};
//                  0.1,     0.1,     0.5,    -0.5,    -0.2,    0.5,     0,       0.1
int Num = 8;

pOutdata = &outVal;

DSP_VectorMul32(pOutdata, inBufMultA, inBufMultB, Num);

// outVal[i] = inBufTest[i] * inBuf2[i] =
//    {0x0CCCCCCD, 0xF3333333, 0x3999999A, 0xCCCCCCCD, 0xFAE147AE,
//                                         0x20000000, 0x00000000, 0xF851EB86}
//      0.1,    -0.1,    0.45,   -0.4,    -0.04,    0.25,     0,      -0.06
```

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int32_t)
indata1	pointer to input array of 16-bit elements (int32_t)
indata2	pointer to input array of 16-bit elements (int32_t)
N	number of samples (int)

DSP_VectorMulc16 Function

Multiplication of a series of numbers in one vector to a scalar value.

File

dsp.h

C

```
void DSP_VectorMulc16(int16_t * outdata, int16_t * indata, int16_t c, int N);
```

Returns

None.

Description

Function DSP_VectorMulc16:

void DSP_VectorMulc16(int16_t *outdata, int16_t *indata, int16_t c, int N);

Multiples the value of each element of indata1 * c and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q15 fractional format. outdata[i] filled with N elements of indata[i] * c

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Example

```
int16_t *pOutdata;
int16_t outVal[8];
int16_t inBufMultA[8]={0x7FFF, 0x8000, 0x7333, 0x6666, 0x1999, 0x4000, 0x0000, 0xB334};
//                      1,     -1,      0.9,    0.8,    0.2,   0.5,    0,     -0.6
int16_t constValue = 0x4000;
int Num = 8;

pOutdata = &outVal;

DSP_VectorMulc16(pOutdata, inBufMultA, constValue, Num);

// outVal[i] = inBufTest[i] * constValue =
//    {0x4000, 0xC000, 0x399A, 0x3333, 0x1999, 0x2000, 0x0000, 0xD99A}
//      0.5,   -0.5,   0.45,    0.4,    0.1,   0.25,    0,     -0.3
```

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int16_t)
indata	pointer to input array of 16-bit elements (int16_t)
c	scalar multiplicand (int16_t)
N	number of samples (int)

DSP_VectorMulc32 Function

Multiplication of a series of numbers in one vector to a scalar value.

File

dsp.h

C

```
void DSP_VectorMulc32(int32_t * outdata, int32_t * indata, int32_t c, int N);
```

Returns

None.

Description

Function DSP_VectorMulc32:

void DSP_VectorMulc32(int32_t *outdata, int32_t *indata, int32_t c, int N);

Multiples the value of each element of indata * c and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q31 fractional format. outdata[i] filled with N elements of indata[i] * c

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int16_t *pOutdata;
int32_t outVal[8];
int32_t inBufMultA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x00000000, 0xB3333334};
//                    1,    -1,    0.9,    0.8,    0.2,    0.5,    1,    -0.6
int32_t constValue = 0x4000;
int Num = 8;

pOutdata = &outVal;

DSP_VectorMulc32(pOutdata, inBufMultA, constValue, Num);

// outVal[i] = inBufTest[i] * constValue =
//    {0x40000000, 0xC0000000, 0x3999999A, 0x33333333, 0x19999999,
//                                         0x20000000, 0x00000000, 0xD999999A}
//      0.5,   -0.5,   0.45,    0.4,    0.1,   0.25,    0,     -0.3
```

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int32_t)
indata	pointer to input array of 16-bit elements (int32_t)
c	scalar multiplicand (int32_t)
N	number of samples (int)

DSP_VectorNegate Function

Inverses the sign (negates) the elements of a vector.

File

dsp.h

C

```
void DSP_VectorNegate(int32_t * outdata, int32_t * indata, int N);
```

Returns

None.

Description

Function DSP_VectorNegate:

void DSP_VectorNegate(int32_t *outdata, int32_t *indata, int N);

Sign inversion of the first N values of an indata are assigned to outdata. N must be a multiple of four and greater than or equal to four or it will be truncated to the nearest multiple of four. The vector result and the scalar value to fill are both Q31 fractional format.

Remarks

None.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int     Num = 4;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                  1,     -1,      0.9,    0.8,    0.2,    0.5,    1,     -0.6
int32_t outBufTest[8]={0x0CCCCCCD, 0x0CCCCCCD, 0x40000000, 0xC0000000,
```

```
                            0xE6666667, 0x40000000, 0x00000000, 0x0CCCCCCD};
//                  0.1,    0.1,    0.5,    -0.5,    -0.2,    0.5,    0,      0.1;

DSP_VectorNegate(outBufTest, inBufTestA, Num);

// inBufTestA =  {0x80000000, 0x7FFFFFFF, 0x8CCCCCCD, 0x9999999A,
//            0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334} // first 4 values neg
//               -1,   1,   -0.9,   -0.8,   -0.2,   0.5,   0,   0.1
```

Parameters

Parameters	Description
outdata	pointer to destination array of elements (int32_t)
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

DSP_VectorRecip Function

Computes the reciprocal (1/x) of the first N elements of inVector, and stores the result in outVector.

File

dsp.h

C

```
void DSP_VectorRecip(_Q16 * outVector, _Q16 * inVector, int N);
```

Returns

None.

Description

Function DSP_VectorRecip:

void DSP_VectorRecip(_Q16 *outVector, _Q16 *inVector, int N);

Computes the reciprocal (1/x) on the first N elements of inVector. The output is stored to outVector. Both vectors are _Q16 format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. If values exceed maximum or minimum they will saturate to the maximum or minimum respectively.

Remarks

This function uses the Microchip PIC32MZ LibQ library to function. The user must include that library and header file into the design in order to operate this function. For more information on the Div function see the LibQ documentation for _LIBQ_Q16Div. A value of zero in the array will not cause an error, but will return 0.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. This function uses the Div function from the LibQ library. That library must be compiled as part of the project.

Example

```
int recNum = 4;
_Q16 inRecVec[8] = {0x08000000, 0xfffc0000,0x00020000,0x00100000,0x00038000,
                0x00400000,0xfffe0000,0x00058000};
//            2048.0, -4.0, 2.0, 16.0, 3.5, 64.0, -2.0, 5.5
_Q16 outRecVec[8] = {0};

DSP_VectorRecip(outRecVec, inRecVec,  recNum);

// outRecVec =  0x00000020, 0xFFFFC0000, 0x00008000, 0x00001000, 0, 0, 0, 0
//                0.000488,        -0.25,         0.5,      0.0625, 0, 0, 0, 0
```

Parameters

Parameters	Description
outdata	pointer to destination array of elements (_Q16)
indata	pointer to source array of elements (_Q16)
N	number of samples (int)

DSP_VectorRMS16 Function

Computes the root mean square (RMS) value of a vector.

File

dsp.h

C

```
int16_t DSP_VectorRMS16(int16_t * inVector, int N);
```

Returns

int16_t - RMS function output, Q15 format

Description

Function DSP_VectorRMS16:

int16_t DSP_VectorRMS16(int16_t *inVector, int N);

Computes the root mean square value of the first N values of inVector. Both input and output are Q15 fractional values. The function will saturate if maximum or minimum values are exceeded.

Remarks

This function is optimized with microMIPS and M14KCe ASE DSP instructions. This function is dependent on the LibQ library, and uses the _LIBQ_Q16Sqrt external function call.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and multiple of four.

Example

```
int16_t vecRMSIn[32]={0x1999, 0xD99A, 0x4000, 0x2666,0x1999,0x1999,0x2666, 0x3333};
// 0.2, -0.3, 0.5, 0.3, 0.2, 0.2, 0.3, 0.4
int16_t RMSOut=0;
int Nrms = 8;

RMSOut = DSP_VectorRMS16(vecRMSIn, Nrms);

// RMSOut = 0x287C (= 0.31628)
```

Parameters

Parameters	Description
indata	pointer to input array of 16-bit elements (int16_t)
N	number of samples (int)

DSP_VectorShift Function

Shifts the data index of an input data vector.

File

dsp.h

C

```
void DSP_VectorShift(int32_t * outdata, int32_t * indata, int N, int shift);
```

Returns

None.

Description

Function DSP_VectorShift:

void DSP_VectorShift(int32_t *outdata, int32_t *indata, int N, int shift);

Shifts N data elements of indata to outdata, with an index change of shift. The amount of data shifted includes zero padding for the first (shift) elements if shift is positive. The vector size of indata and outdata need not be the same, however, N must not exceed either array size.

Remarks

Destination array values shift to left (relative to the input vector) when shift is positive (back filled with zeros) and shift to the right when shift is negative. The total amount of values copied to the destination array is the length of N less the shift amount.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must not exceed the amount of elements in the source array. shift must not exceed the number of elements in the destination array.

Example

```
int shiftValue = 3;
int     Num = 8;
int32_t inBufTestA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                  1,     -1,      0.9,    0.8,    0.2,   0.5,    1,     -0.6
int32_t inBufTestB[8]={0x80000000, 0x7FFFFFFF, 0x40000000, 0x0CCCCCCC,
                       0x40000000, 0x60000000, 0x80000000, 0x20000000};
//                  -1,     1,      0.5,    0.1,    0.75,   0.5,    -1,     0.25

DSP_VectorShift(inBufTestA, inBufTestB, Num, shiftValue);

// inBufTestA =  {0x00000000, 0x00000000, 0x00000000, 0x80000000,
//           0x7FFFFFFF, 0x40000000, 0x0CCCCCCC, 0x40000000} // shifted 3 positive
```

Parameters

Parameters	Description
outdata	pointer to destination array of elements (int32_t)
indata	pointer to source array of elements (int32_t)
N	number of samples (int)
shift	number of indexes to shift (int)

DSP_VectorSqrt Function

Computes the square root of the first N elements of inVector, and stores the result in outVector.

File

dsp.h

C

```
void DSP_VectorSqrt(_Q16 * outVector, _Q16 * inVector, int N);
```

Returns

None.

Description

Function DSP_VectorSqrt:

void DSP_VectorSqrt(_Q16 *outVector, _Q16 *inVector, int N);

Computes the Sqrt(x) on the first N elements of inVector. The output is stored to outVector. Both vectors are _Q16 format, which is 32-bit data with 15 bits for the integer and 16 bits for the fractional portion. If values exceed maximum or minimum they will saturate to the maximum or zero respectively.

Remarks

This function uses the Microchip PIC32MZ LibQ library to function. The user must include that library and header file into the design in order to operate this function. For more information on the Sqrt function see the LibQ documentation for _LIBQ_Q16Sqrt. A negative number input will return a saturated value (0x00FFFFxx).

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. This function uses the Sqrt function from the LibQ library. That library must be compiled as part of the project.

Example

```
int sqrtNum = 4;
_Q16 inSqrtVec[8] = {0x40000000, 0xffff0000,0x00020000,0x00100000,0x00038000,
```

```
            0x00400000,0xfffe0000,0x00058000};
//          16384.0,  -1.0,  2.0,  16.0,   3.5,   64.0,  -2.0,  5.5
_Q16 outSqrtVec[8] = {0};

DSP_VectorSqrt(outSqrtVec, inSqrtVec,  sqrtNum);

// outSqrtVec =  0x00800000, 0x00FFFF80, 0x00016A0A, 0x00040000, 0, 0, 0, 0
//                 128.0,  sat negative,   1.41422,      4.0,    0, 0, 0, 0
```

Parameters

Parameters	Description
outdata	pointer to destination array of elements (_Q16)
indata	pointer to source array of elements (_Q16)
N	number of samples (int)

DSP_VectorStdDev16 Function

Computes the Standard Deviation of a Vector.

File

dsp.h

C

```
int16_t DSP_VectorStdDev16(int16_t * inVector, int N);
```

Returns

int16_t - Standard Deviation of N selected elements

Description

Function DSP_VectorStdDev16:

int16_t DSP_VectorStdDev16(int16_t *inVector, int N);

Calculates the standard deviation on the first N elements of inVector and returns the 16-bit scalar result. The standard deviation is the square root of the variance, which is a measure of the delta from mean values. The mean value of the vector is computed in the process. The function has the form -

StdDev = SQRT(sum[0..N]((x(i) - M(N))^2) / (N-1)) where N is the number of vector elements x(i) is a single element in the vector M(N) is the mean of the N elements of the vector

Input values of the vector and output scalar value is Q15 fractional format. This format has data that ranges from -1 to 1, and has internal saturation limits of those same values. Some care has been taken to reduce the impact of saturation by adding processing steps to effectively complete the processing in blocks. However, in some extreme cases of data variance it is still possible to reach the saturation limits.

Remarks

The input vector elements number, N, must be at least 4 and a multiple of 4. This function is optimized with microMIPS and M14KCe ASE DSP instructions. This function is dependent on the LibQ library, and the _LIBQ_Q16Sqrt specifically.

Preconditions

The pointers inVector must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four. Dependent on use of the LibQ library.

Example

```
int16_t vecStdDevIn[32]={0x4000, 0xD99A, 0x1000, 0x6000,0x1999,0x1999,0x2666, 0x3333};
        // .2, -.3, .125, .75, .2, .2, .3, .4
int16_t StDevOut, Var16Out;
int Nstdev = 4;

StDevOut = DSP_VectorStdDev16(vecStdDevIn, Nstdev);
        // StDevOut = 0x3A9E (= 0.45797)
```

Parameters

Parameters	Description
inVector	pointer to source array of elements (int16_t)
N	number of samples (int)

DSP_VectorSub16 Function

Calculate the difference of two vectors.

File

dsp.h

C

```
void DSP_VectorSub16(int16_t * outdata, int16_t * indata1, int16_t * indata2, int N);
```

Returns

None.

Description

Function DSP_VectorSub16:

void DSP_VectorSub16(int16_t *outdata, int16_t *indata1, int16_t *indata2, int N);

Computes the difference value of each element of indata1 - indata2 and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q15 fractional format. outdata[i] filled with N elements of indata1[i] - indata2[i]

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Example

```
int16_t *pOutdata;
int16_t outVal[8];
int16_t inBufTest[16] = {-5,2,-3,4,-1,0,-2,-8,-21,21,10,100, 200, 127,-127,-2};
int16_t inBuf2[16]=     { 1,2, 3,4, 5,6, 7, 8, 9, 10,-1,-100,-127,127,-7,  0};
int Num = 8;

pOutdata = &outVal;

DSP_VectorSub16(pOutdata, inBufTest, inBuf2, Num);

// outVal[i] = inBufTest[i] - inBuf2[i] = {-6,0,-6,0,-6,-6,-9,-16}
```

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int16_t)
indata1	pointer to input array of 16-bit elements (int16_t)
indata2	pointer to input array of 16-bit elements (int16_t)
N	number of samples (int)

DSP_VectorSub32 Function

Calculate the difference of two vectors.

File

dsp.h

C

```
void DSP_VectorSub32(int32_t * outdata, int32_t * indata1, int32_t * indata2, int N);
```

Returns

None.

Description

Function DSP_VectorSub32:

void DSP_VectorSub32(int32_t *outdata, int32_t *indata1, int32_t *indata2, int N);

Computes the difference value of each element of indata1 - indata2 and stores it to outdata. The number of samples to process is given by the parameter N. Data is in a Q31 fractional format. outdata[i] filled with N elements of indata1[i] - indata2[i]

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int16_t *pOutdata;
int32_t outVal[8];
int32_t inBufTest[16] = {-5,2,-3,4,-1,0,-2,-8,-21,21,10,100, 200, 127,-127,-2};
int32_t inBuf2[16]=     { 1,2, 3,4, 5,6, 7, 8, 9, 10,-1,-100,-127,127,-7,  0};
int Num = 8;

pOutdata = &outVal;

DSP_VectorSub32(pOutdata, inBufTest, inBuf2, Num);

// outVal[i] = inBufTest[i] - inBuf2[i] = {-6,0,-6,0,-6,-6,-9,-16}
```

Parameters

Parameters	Description
outdata	pointer to output array of 16-bit elements (int32_t)
indata1	pointer to input array of 16-bit elements (int32_t)
indata2	pointer to input array of 16-bit elements (int32_t)
N	number of samples (int)

DSP_VectorSumSquares16 Function

Computes the sum of squares of a vector, and scales the output by a binary factor.

File

dsp.h

C

```
int16_t DSP_VectorSumSquares16(int16_t * indata, int N, int scale);
```

Returns

int16_t - scaled output of calculation, Q15 format

Description

Function DSP_VectorSumSquares16:

int16_t DSP_VectorSumSquares16(int16_t *indata, int N, int scale);

Calculates the sum of the squares of each element of an input vector, and scales the output. Function will saturate if it exceeds maximum or minimum values. Scaling is done by binary shifting, after accumulation in a 32 bit register. All calculations are done in Q15 fractional format. return = 1/(2^scale) * sum(indata[i]^2)

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to eight and a multiple of eight.

Example

```
int16_t inBufMultA[8]={0x7FFF, 0x8000, 0x7333, 0x6666, 0x1999, 0x4000, 0x7FFF, 0xB334};
//                        1,     -1,      0.9,    0.8,    0.2,   0.5,    1,     -0.6
int Num = 8;
int scaleVal = 3;
int16_t outScalar;

outScalar = DSP_VectorSumSquares16(inBufMultA, Num, scaleVal);
```

```
// outScalar = 1/(2^scaleVal)* sum(inBufMultA[i]^2) =
//   (1/8) * (1 + 1 + 0.81 + 0.64 + 0.04 + 0.25 + 1 + 0.36) = 0.125 * 5.1 = 0.6375
//      = (int16_t)0x5199
```

Parameters

Parameters	Description
indata	pointer to input array of 16-bit elements (int16_t)
scale	number of bits to shift return right (int)
N	number of samples (int)

DSP_VectorSumSquares32 Function

Computes the sum of squares of a vector, and scales the output by a binary factor.

File

dsp.h

C

```
int32_t DSP_VectorSumSquares32(int32_t * indata, int N, int scale);
```

Returns

int32_t - scaled output of calculation, Q15 format

Description

Function DSP_VectorSumSquares32:

int32_t DSP_VectorSumSquares32(int32_t *indata, int N, int scale);

Calculates the sum of the squares of each element of an input vector, and scales the output. The function will saturate if it exceeds maximum or minimum values. Scaling is done by binary shifting, after calculation of the results. All calculations are done in Q31 fractional format. return = 1/(2^scale) * sum(indata[i]^2)

Remarks

This must be assembled with .set microMIPS.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and multiple of four.

Example

```
int32_t inBufMultA[8]={0x7FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x00000000, 0xB3333334};
//                   1,    -1,    0.9,    0.8,    0.2,   0.5,    1,    -0.6
int Num = 8;
int scaleVal = 3;
int32_t outScalar;

outScalar = DSP_VectorSumSquares32(inBufMultA, Num, scaleVal);

// outScalar = 1/(2^scaleVal)* sum(inBufMultA[i]^2) =
//   (1/8) * (1 + 1 + 0.81 + 0.64 + 0.04 + 0.25 + 1 + 0.36) = 0.125 * 5.1 = 0.6375
//      = (int32_t)0x51999999
```

Parameters

Parameters	Description
indata	pointer to input array of 16-bit elements (int32_t)
scale	number of bits to shift return right (int)
N	number of samples (int)

DSP_VectorVari16 Function

Computes the variance of N elements of a Vector.

File

dsp.h

C

```
int16_t DSP_VectorVari16(int16_t * inVector, int N);
```

Returns

int16_t - Variance of N selected elements

Description

Function DSP_VectorVari16:

int16_t DSP_VectorVari16(int16_t *inVector, int N);

Calculates the variance on the first N elements of inVector and returns the 16-bit scalar result. The variance is a measure of the delta from mean values, and the mean value of the vector is computed in the process. The function has the form -

var = sum[0..N]((x(i) - M(N))^2) / (N-1) where N is the number of vector elements x(i) is a single element in the vector M(N) is the mean of the N elements of the vector

Input values of the vector and output scalar value is Q15 fractional format. This format has data that ranges from -1 to 1, and has internal saturation limits of those same values. Some care has been taken to reduce the impact of saturation by adding processing steps to effectively complete the processing in blocks. However, in some extreme cases of data variance it is still possible to reach the saturation limits.

Remarks

The input vector elements number, N, must be at least 4 and a multiple of 4. This function is optimized with microMIPS and M14KCe ASE DSP instructions.

Preconditions

The pointers inVector must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int16_t vecStDevIn[32]={0x4000, 0xD99A, 0x1000, 0x6000,0x1999,0x1999,0x2666, 0x3333};
        // .2, -.3, .125, .75, .2, .2, .3, .4
int16_t Var16Out;
int Nvar = 4;

Var16Out= DSP_VectorVari16(vecStDevIn, Nvar);  // 16-bit variance function
        // Var16Out = 0x1AD8 (= 0.20974)
```

Parameters

Parameters	Description
inVector	pointer to source array of elements (int16_t)
N	number of samples (int)

DSP_VectorVariance Function

Computes the variance of N elements of inVector.

File

dsp.h

C

```
int32_t DSP_VectorVariance(int32_t * inVector, int N);
```

Returns

int32_t - Variance of N selected elements

Description

Function DSP_VectorVariance:

int32_t DSP_VectorVariance(int32_t *inVector, int N);

Calculates the variance on the first N elements of inVector and returns the 32-bit scalar result. The variance is a measure of the delta from mean values, and the mean value of the vector is computed in the process. The function has the form -

var = sum[0..N]((x(i) - M(N))^2) / (N-1) where N is the number of vector elements x(i) is a single element in the vector M(N) is the mean of the N elements of the vector

Input values of the vector and output scalar value is Q31 fractional format. This format has data that ranges from -1 to 1, and has internal saturation limits of those same values. Some care has been taken to reduce the impact of saturation by adding processing steps to effectively complete the processing in blocks. However, in some extreme cases of data variance it is still possible to reach the saturation limits.

Remarks

The input vector elements number, N, must be at least 4 and a multiple of 4. This function is optimized with microMIPS and M14KCe ASE DSP instructions.

Preconditions

The pointers inVector must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int varN = 8;

int32_t inVarVec[8] = {0xE6666667, 0x40000000,0x40000000,0x0CCCCCCC,
          0x00000000,0x59999999,0x20000000,0xC0000000};
//     -0.2, 0.5, 0.5, 0.1, 0, 0.7, 0.25, -0.5
int32_t outVar = 0;

outVar = DSP_VectorVariance(inVarVec, varN);

// outVar == 0x1490D2A6, = 0.1606696
```

Parameters

Parameters	Description
inVector	pointer to source array of elements (int32_t)
N	number of samples (int)

DSP_VectorZeroPad Function

Fills an input vector with zeros.

File

dsp.h

C

```
void DSP_VectorZeroPad(int32_t * indata, int N);
```

Returns

None.

Description

Function DSP_VectorZeroPad:

void DSP_VectorZeroPad(int32_t *indata, int N);

Fills the first N values of an input vector indata with the value zero. N must be a multiple of four and greater than or equal to four or it will be truncated to the nearest multiple of four. The vector result is in Q31 fractional format.

Remarks

None.

Preconditions

The pointers outdata and indata must be aligned on 4-byte boundaries. N must be greater than or equal to four and a multiple of four.

Example

```
int     Num = 4;
int32_t inBufTestA[8]={0x3FFFFFFF, 0x80000000, 0x73333333, 0x66666666,
                       0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
//                 0.5,     -1,      0.9,    0.8,    0.2,   0.5,    1,    -0.6

DSP_VectorZeroPad(inBufTestA, Num);

// inBufTestA =  {0x00000000, 0x00000000, 0x00000000, 0x00000000,
//                            0x19999999, 0x40000000, 0x7FFFFFFF, 0xB3333334};
```

```
// 0, 0, 0, 0, 0.2, 0.5, 1, -0.6
```

Parameters

Parameters	Description
indata	pointer to source array of elements (int32_t)
N	number of samples (int)

f) Support Functions

mul16 Function

File

dsp.h

C

```
static inline int16_t mul16(int16_t a, int16_t b);
```

Description

multiply and shift integer

mul16r Function

File

dsp.h

C

```
static inline int16_t mul16r(int16_t a, int16_t b);
```

Description

multiply and shift Q15

mul32 Function

File

dsp.h

C

```
static inline int32_t mul32(int32_t a, int32_t b);
```

Description

multiply and shift Q31

SAT16 Function

File

dsp.h

C

```
static inline int32_t SAT16(int32_t x);
```

Description

saturate both positive and negative Q15

SAT16N Function

File

dsp.h

C

```
static inline int32_t SAT16N(int32_t x);
```

Description

saturate negative Q15

SAT16P Function

File

dsp.h

C

```
static inline int32_t SAT16P(int32_t x);
```

Description

saturate positive Q15

g) Data Types and Constants

biquad16 Structure

File

dsp.h

C

```
typedef struct {
  int16_t a1;
  int16_t a2;
  int16_t b1;
  int16_t b2;
} biquad16;
```

Members

Members	Description
int16_t a1;	feedback delay 1 coef
int16_t a2;	feedback delay 2 coef
int16_t b1;	feedforward delay 1 coef
int16_t b2;	feedforward delay 1 coef

Description

Q15 biquad

int16c Structure

File

dsp.h

C

```
typedef struct {
  int16_t re;
  int16_t im;
} int16c;
```

Members

Members	Description
int16_t re;	real portion (a)
int16_t im;	imaginary portion (b)

Description

Q15 complex number (a + bi)

int32c Structure

File

dsp.h

C

```
typedef struct {
  int32_t re;
  int32_t im;
} int32c;
```

Members

Members	Description
int32_t re;	real portion (a)
int32_t im;	imaginary portion (b)

Description

Q31 complex number (a + bi)

matrix32 Structure

File

dsp.h

C

```
typedef struct {
  int32_t row;
  int32_t col;
  int32_t * pMatrix;
} matrix32;
```

Members

Members	Description
int32_t row;	matrix rows
int32_t col;	matrix columns
int32_t * pMatrix;	matrix pointer to data

Description

Q31 matrix

PARM_EQUAL_FILTER Structure

File

dsp.h

C

```
typedef struct _PARM_EQUAL_FILTER {
  PARM_FILTER_GAIN G;
  int16_t log2Alpha;
  int16_t b[3];
  int16_t a[2];
  int32_t Z[2];
} PARM_EQUAL_FILTER;
```

Members

Members	Description
PARM_FILTER_GAIN G;	Filter max gain multiplier

int16_t log2Alpha;	coefficient scaling bit shift value
int16_t b[3];	Feedforward Coefficients, Q15 format
int16_t a[2];	Feedback Coefficients, Q15 format
int32_t Z[2];	Filter memory, should be initialized to zero.

Description

IIR BQ filter structure Q15 data, Q31 storage

PARM_EQUAL_FILTER_16 Structure

File

dsp.h

C

```
typedef struct _PARM_EQUAL_FILTER_16 {
  PARM_FILTER_GAIN G;
  int16_t log2Alpha;
  int16_t b[3];
  int16_t a[2];
  int16_t Z[2];
} PARM_EQUAL_FILTER_16;
```

Members

Members	Description
PARM_FILTER_GAIN G;	Filter max gain multiplier
int16_t log2Alpha;	coefficient scaling bit shift value
int16_t b[3];	Feedforward Coefficients, Q15 format
int16_t a[2];	Feedback Coefficients, Q15 format
int16_t Z[2];	Filter memory, should be initialized to zero.

Description

IIR BQ filter structure Q15

PARM_EQUAL_FILTER_32 Structure

File

dsp.h

C

```
typedef struct _PARM_EQUAL_FILTER_32 {
  PARM_FILTER_GAIN G;
  int log2Alpha;
  int32_t b[3];
  int32_t a[2];
  int32_t Z[2];
} PARM_EQUAL_FILTER_32;
```

Members

Members	Description
PARM_FILTER_GAIN G;	Filter max gain multiplier
int log2Alpha;	coefficient scaling bit shift value
int32_t b[3];	Feedforward Coefficients, Q15 format
int32_t a[2];	Feedback Coefficients, Q15 format
int32_t Z[2];	Filter memory, should be initialized to zero.

Description

IIR BQ filter structure Q31

PARM_FILTER_GAIN Structure

File

dsp.h

C

```
typedef struct {
  int16_t fracGain;
  int16_t expGain;
} PARM_FILTER_GAIN;
```

Members

Members	Description
int16_t fracGain;	Q15 fractional filter gain
int16_t expGain;	log2N (binary) filter gain

Description

filter gain structure

MAX16 Macro

File

dsp.h

C

```
#define MAX16 ((int16_t) 0x7FFF)      // maximum Q15
```

Description

maximum Q15

MAX32 Macro

File

dsp.h

C

```
#define MAX32 ((int32_t) 0x7FFFFFFF)  // maximum Q31
```

Description

maximum Q31

MIN16 Macro

File

dsp.h

C

```
#define MIN16 ((int16_t) 0x8000)      // minimum Q15
```

Description

minimum Q15

MIN32 Macro

File

dsp.h

C

```
#define MIN32 ((int32_t) 0x80000000)  // minimum Q31
```

Description

minimum Q31

Files

Files

Name	Description
dsp.h	DSP functions for the PIC32MZ device family.

Description

This section lists the source and header files used by the DSP Fixed-Point Math Library.

dsp.h

DSP functions for the PIC32MZ device family.

Functions

	Name	Description
	DSP_ComplexAdd32	Calculates the sum of two complex numbers.
	DSP_ComplexConj16	Calculates the complex conjugate of a complex number.
	DSP_ComplexConj32	Calculates the complex conjugate of a complex number.
	DSP_ComplexDotProd32	Calculates the dot product of two complex numbers.
	DSP_ComplexMult32	Multiplies two complex numbers.
	DSP_ComplexScalarMult32	Multiplies a complex number and a scalar number.
	DSP_ComplexSub32	Calculates the difference of two complex numbers.
	DSP_FilterFIR32	Performs a Finite Infinite Response (FIR) filter on a vector.
	DSP_FilterFIRDecim32	Performs a decimating FIR filter on the input array.
	DSP_FilterFIRInterp32	Performs an interpolating FIR filter on the input array.
	DSP_FilterIIR16	Performs a single-sample cascaded biquad Infinite Impulse Response (IIR) filter.
	DSP_FilterIIRBQ16	Performs a single-pass IIR Biquad Filter.
	DSP_FilterIIRBQ16_cascade8	Performs a single-sample IIR Biquad Filter as a cascade of 8 series filters.
	DSP_FilterIIRBQ16_cascade8_fast	Performs a single-sample IIR Biquad Filter as a cascade of 8 series filters.
	DSP_FilterIIRBQ16_fast	Performs a single-pass IIR Biquad Filter.
	DSP_FilterIIRBQ16_parallel8	Performs a 8 parallel single-pass IIR Biquad Filters, and sums the result.
	DSP_FilterIIRBQ16_parallel8_fast	Performs a 8 parallel single-pass IIR Biquad Filters, and sums the result.
	DSP_FilterIIRBQ32	Performs a high resolution single-pass IIR Biquad Filter.
	DSP_FilterIIRSetup16	Converts biquad structure to coeffs array to set up IIR filter.
	DSP_FilterLMS16	Performs a single sample Least Mean Squares FIR Filter.
	DSP_MatrixAdd32	Addition of two matrices C = (A + B).
	DSP_MatrixEqual32	Equality of two matrices C = (A).
	DSP_MatrixInit32	Initializes the first N elements of a Matrix to the value num.
	DSP_MatrixMul32	Multiplication of two matrices C = A x B.
	DSP_MatrixScale32	Scales each element of an input buffer (matrix) by a fixed number.
	DSP_MatrixSub32	Subtraction of two matrices C = (A - B).
	DSP_MatrixTranspose32	Transpose of a Matrix C = A (T).
	DSP_TransformFFT16	Creates an Fast Fourier Transform (FFT) from a time domain input.
	DSP_TransformFFT16_setup	Creates FFT coefficients for use in the FFT16 function.
	DSP_TransformFFT32	Creates an Fast Fourier Transform (FFT) from a time domain input.
	DSP_TransformFFT32_setup	Creates FFT coefficients for use in the FFT32 function.
	DSP_TransformIFFT16	Creates an Inverse Fast Fourier Transform (FFT) from a frequency domain input.
	DSP_TransformWindow_Bart16	Perform a Bartlett window on a vector.
	DSP_TransformWindow_Bart32	Perform a Bartlett window on a vector.

	DSP_TransformWindow_Black16	Perform a Blackman window on a vector.
	DSP_TransformWindow_Black32	Perform a Blackman window on a vector.
	DSP_TransformWindow_Cosine16	Perform a Cosine (Sine) window on a vector.
	DSP_TransformWindow_Cosine32	Perform a Cosine (Sine) window on a vector.
	DSP_TransformWindow_Hamm16	Perform a Hamming window on a vector.
	DSP_TransformWindow_Hamm32	Perform a Hamming window on a vector.
	DSP_TransformWindow_Hann16	Perform a Hanning window on a vector.
	DSP_TransformWindow_Hann32	Perform a Hanning window on a vector.
	DSP_TransformWindow_Kaiser16	Perform a Kaiser window on a vector.
	DSP_TransformWindow_Kaiser32	Perform a Kaiser window on a vector.
	DSP_TransformWinInit_Bart16	Create a Bartlett window.
	DSP_TransformWinInit_Bart32	Create a Bartlett window.
	DSP_TransformWinInit_Black16	Create a Blackman window.
	DSP_TransformWinInit_Black32	Create a Blackman window.
	DSP_TransformWinInit_Cosine16	Create a Cosine (Sine) window.
	DSP_TransformWinInit_Cosine32	Create a Cosine (Sine) window.
	DSP_TransformWinInit_Hamm16	Create a Hamming window.
	DSP_TransformWinInit_Hamm32	Create a Hamming window.
	DSP_TransformWinInit_Hann16	Create a Hanning window.
	DSP_TransformWinInit_Hann32	Create a Hanning window.
	DSP_TransformWinInit_Kaiser16	Create a Kaiser window.
	DSP_TransformWinInit_Kaiser32	Create a Kaiser window.
	DSP_VectorAbs16	Calculate the absolute value of a vector.
	DSP_VectorAbs32	Calculate the absolute value of a vector.
	DSP_VectorAdd16	Calculate the sum of two vectors.
	DSP_VectorAdd32	Calculate the sum of two vectors.
	DSP_VectorAddc16	Calculate the sum of a vector and a constant.
	DSP_VectorAddc32	Calculate the sum of a vector and a constant.
	DSP_VectorAutocorr16	Computes the Autocorrelation of a Vector.
	DSP_VectorBexp16	Computes the maximum binary exponent of a vector.
	DSP_VectorBexp32	Computes the maximum binary exponent of a vector.
	DSP_VectorChkEqu32	Compares two input vectors, returns an integer '1' if equal, and '0' if not equal.
	DSP_VectorCopy	Copies the elements of one vector to another.
	DSP_VectorCopyReverse32	Reverses the order of elements in one vector and copies them into another.
	DSP_VectorDivC	Divides the first N elements of inVector by a constant divisor, and stores the result in outVector.
	DSP_VectorDotp16	Computes the dot product of two vectors, and scales the output by a binary factor.
	DSP_VectorDotp32	Computes the dot product of two vectors, and scales the output by a binary factor
	DSP_VectorExp	Computes the EXP (e^x) of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorFill	Fills an input vector with scalar data.
	DSP_VectorLn	Computes the Natural Log, Ln(x), of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorLog10	Computes the Log10(x), of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorLog2	Computes the Log2(x) of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorMax32	Returns the maximum value of a vector.
	DSP_VectorMaxIndex32	Returns the index of the maximum value of a vector.
	DSP_VectorMean32	Calculates the mean average of an input vector.
	DSP_VectorMin32	Returns the minimum value of a vector.
	DSP_VectorMinIndex32	Returns the index of the minimum value of a vector.
	DSP_VectorMul16	Multiplication of a series of numbers in one vector to another vector.
	DSP_VectorMul32	Multiplication of a series of numbers in one vector to another vector.
	DSP_VectorMulc16	Multiplication of a series of numbers in one vector to a scalar value.
	DSP_VectorMulc32	Multiplication of a series of numbers in one vector to a scalar value.
	DSP_VectorNegate	Inverses the sign (negates) the elements of a vector.
	DSP_VectorRecip	Computes the reciprocal (1/x) of the first N elements of inVector, and stores the result in outVector.

	DSP_VectorRMS16	Computes the root mean square (RMS) value of a vector.
	DSP_VectorShift	Shifts the data index of an input data vector.
	DSP_VectorSqrt	Computes the square root of the first N elements of inVector, and stores the result in outVector.
	DSP_VectorStdDev16	Computes the Standard Deviation of a Vector.
	DSP_VectorSub16	Calculate the difference of two vectors.
	DSP_VectorSub32	Calculate the difference of two vectors.
	DSP_VectorSumSquares16	Computes the sum of squares of a vector, and scales the output by a binary factor.
	DSP_VectorSumSquares32	Computes the sum of squares of a vector, and scales the output by a binary factor.
	DSP_VectorVari16	Computes the variance of N elements of a Vector.
	DSP_VectorVariance	Computes the variance of N elements of inVector.
	DSP_VectorZeroPad	Fills an input vector with zeros.
	mul16	multiply and shift integer
	mul16r	multiply and shift Q15
	mul32	multiply and shift Q31
	SAT16	saturate both positive and negative Q15
	SAT16N	saturate negative Q15
	SAT16P	saturate positive Q15

Macros

	Name	Description
	MAX16	maximum Q15
	MAX32	maximum Q31
	MIN16	minimum Q15
	MIN32	minimum Q31

Structures

	Name	Description
	_PARM_EQUAL_FILTER	IIR BQ filter structure Q15 data, Q31 storage
	_PARM_EQUAL_FILTER_16	IIR BQ filter structure Q15
	_PARM_EQUAL_FILTER_32	IIR BQ filter structure Q31
	biquad16	Q15 biquad
	int16c	Q15 complex number (a + bi)
	int32c	Q31 complex number (a + bi)
	matrix32	Q31 matrix
	PARM_EQUAL_FILTER	IIR BQ filter structure Q15 data, Q31 storage
	PARM_EQUAL_FILTER_16	IIR BQ filter structure Q15
	PARM_EQUAL_FILTER_32	IIR BQ filter structure Q31
	PARM_FILTER_GAIN	filter gain structure

Description

Digital Signal Processing (DSP) Library

The DSP Library provides functions that are optimized for performance on the PIC32MZ families of devices that have microAptiv core features and utilize DSP ASE. The library provides advanced mathematical operations for complex numbers, vector and matrix mathematics, digital filtering and transforms.

All functions are implemented in efficient assembly with C-callable prototypes. In some cases both 16-bit and 32-bit functions are supplied to enable the user with a choice of resolution and performance.

For most functions, input and output data is represented by 16-bit fractional numbers in Q15 format, which is the most commonly used data format for signal processing. Some functions use other data formats internally for increased precision of intermediate results.

The Q15 data type used by the DSP functions is specified as int16_t in the C header file that is supplied with the library. Note that within C code, care must be taken to avoid confusing fixed-point values with integers. To the C compiler, objects declared with int16_t type are integers, not fixed-point, and all arithmetic operations performed on those objects in C will be done as integers. Fixed-point values have been declared as int16_t only because the standard C language does not include intrinsic support for fixed-point data types.

Some functions also have versions operating on 32-bit fractional data in Q31 format. These functions operate similarly to their 16-bit counterparts. However, it should be noted that the 32-bit functions do not benefit much from the SIMD capabilities offered by DSP ASE. Thus, the performance of the 32-bit functions is generally reduced compared to the performance of the corresponding 16-bit functions.

Signed fixed point types are defined as follows:

Qn.m where:

- n is the number of data bits to the left of the radix point
- m is the number of data bits to the right of the radix point
- a signed bit is implied

Unique variable types for fractional representation are also defined:

Exact Name # Bits Required Type Q0.15 (Q15) 16 int16_t Q0.31 (Q31) 32 int32_t

Table of DSP Library functions:

Complex Math:

void DSP_ComplexAdd32(int32c *indata1, int32c *indata2, int32c *Output);

void DSP_ComplexConj16(int16c *indata, int16c *Output);

void DSP_ComplexConj32(int32c *indata, int32c *Output);

void DSP_ComplexDotProd32(int32c *indata1, int32c *indata2, int32c *Output);

void DSP_ComplexMult32(int32c *indata1, int32c *indata2, int32c *Output);

void DSP_ComplexScalarMult32(int32c *indata, int32_t Scalar, int32c *Output);

void DSP_ComplexSub32(int32c *indata1, int32c *indata2, int32c *Output);

Filter Functions:

void DSP_FilterFIR32(int32_t *outdata, int32_t *indata, int32_t *coeffs2x, int32_t *delayline, int N, int K, int scale);

void DSP_FilterFIRDecim32(int32_t *outdata, int32_t *indata, int32_t *coeffs, int32_t *delayline, int N, int K, int scale, int rate);

int16_t DSP_FilterIIR16(int16_t in, int16_t *coeffs, int16_t *delayline, int B, int scale);

void DSP_FilterIIRSetup16(int16_t *coeffs, biquad16 *bq, int B);

void DSP_FilterFIRInterp32(int32_t *outdata, int32_t *indata, int32_t *coeffs, int32_t *delayline, int N, int K, int scale, int rate);

int16_t DSP_FilterLMS16(int16_t in, int16_t ref, int16_t *coeffs, int16_t *delayline, int16_t *error, int K, int16_t mu);

int16_t DSP_FilterIIRBQ16_fast(int16_t Xin, PARM_EQUAL_FILTER_16 *pFilter);

int16_t DSP_FilterIIRBQ16(int16_t Xin, PARM_EQUAL_FILTER *pFilter);

int32_t DSP_FilterIIRBQ32(int32_t Xin, PARM_EQUAL_FILTER_32 *pFilter);

int16_t DSP_FilterIIRBQ16_cascade8(int16_t Xin, PARM_EQUAL_FILTER *pFilter_Array);

int16_t DSP_FilterIIRBQ16_cascade8_fast(int16_t Xin, PARM_EQUAL_FILTER_16 *pFilter_Array);

int16_t DSP_FilterIIRBQ16_parallel8(int16_t Xin, PARM_EQUAL_FILTER *pFilter);

int16_t DSP_FilterIIRBQ16_parallel8_fast(int16_t Xin, PARM_EQUAL_FILTER_16 *pFilter);

Matrix Functions:

void DSP_MatrixAdd32(matrix32 *resMat, matrix32 *srcMat1, matrix32 *srcMat2);

void DSP_MatrixEqual32(matrix32 *resMat, matrix32 *srcMat);

void DSP_MatrixInit32(int32_t *data_buffer, int32_t N, int32_t num);

void DSP_MatrixMul32(matrix32 *resMat, matrix32 *srcMat1, matrix32 *srcMat2);

void DSP_MatrixScale32(int32_t *data_buffer, int32_t N, int32_t num);

void DSP_MatrixSub32(matrix32 *resMat, matrix32 *srcMat1, matrix32 *srcMat2);

void DSP_MatrixTranspose32(matrix32 *desMat, matrix32 *srcMat);

Transforms:

void DSP_TransformFFT16(int16c *dout, int16c *din, int16c *twiddles, int16c *scratch, int log2N);

void DSP_TransformIFFT16(int16c *dout, int16c *din, int16c *twiddles, int16c *scratch, int log2N);

void DSP_TransformFFT16_setup(int16c *twiddles, int log2N);

void DSP_TransformFFT32(int32c *dout, int32c *din, int32c *twiddles, int32c *scratch, int log2N);

void DSP_TransformFFT32_setup(int32c *twiddles, int log2N);

void DSP_TransformWindow_Bart16(int16_t *OutVector, int16_t *InVector, int N);

void DSP_TransformWindow_Bart32(int32_t *OutVector, int32_t *InVector, int N);

void DSP_TransformWindow_Black16(int16_t *OutVector, int16_t *InVector, int N);

void DSP_TransformWindow_Black32(int32_t *OutVector, int32_t *InVector, int N);

void DSP_TransformWindow_Cosine16(int16_t *OutVector, int16_t *InVector, int N);

void DSP_TransformWindow_Cosine32(int32_t *OutVector, int32_t *InVector, int N);

void DSP_TransformWindow_Hamm16(int16_t *OutVector, int16_t *InVector, int N);

void DSP_TransformWindow_Hamm32(int32_t *OutVector, int32_t *InVector, int N);

void DSP_TransformWindow_Hann16(int16_t *OutVector, int16_t *InVector, int N);

void DSP_TransformWindow_Hann32(int32_t *OutVector, int32_t *InVector, int N);

void DSP_TransformWindow_Kaiser16(int16_t *OutVector, int16_t *InVector, int N);

void DSP_TransformWindow_Kaiser32(int32_t *OutVector, int32_t *InVector, int N);

void DSP_TransformWinInit_Bart16(int16_t *OutWindow, int N);

void DSP_TransformWinInit_Bart32(int32_t *OutWindow, int N);

void DSP_TransformWinInit_Black16(int16_t *OutWindow, int N);

void DSP_TransformWinInit_Black32(int32_t *OutWindow, int N);
void DSP_TransformWinInit_Cosine16(int16_t *OutWindow, int N);
void DSP_TransformWinInit_Cosine32(int32_t *OutWindow, int N);
void DSP_TransformWinInit_Hamm16(int16_t *OutWindow, int N);
void DSP_TransformWinInit_Hamm32(int32_t *OutWindow, int N);
void DSP_TransformWinInit_Hann16(int16_t *OutWindow, int N);
void DSP_TransformWinInit_Hann32(int32_t *OutWindow, int N);
void DSP_TransformWinInit_Kaiser16(int16_t *OutWindow, int N);
void DSP_TransformWinInit_Kaiser32(int32_t *OutWindow, int N);
Vector Math:
void DSP_VectorAbs16(int16_t *outdata, int16_t *indata, int N);
void DSP_VectorAbs32(int32_t *outdata, int32_t *indata, int N);
void DSP_VectorAdd16(int16_t *outdata, int16_t *indata1, int16_t *indata2, int N);
void DSP_VectorAdd32(int32_t *outdata, int32_t *indata1, int32_t *indata2, int N);
void DSP_VectorAddc16(int16_t *outdata, int16_t *indata, int16_t c, int N);
void DSP_VectorAddc32(int32_t *outdata, int32_t *indata, int32_t c, int N);
void DSP_VectorAutocorr16(int16_t *outCorr, int16_t *inVector, int N, int K);
int DSP_VectorBexp16(int16_t *DataIn, int N);
int DSP_VectorBexp32(int32_t *DataIn, int N);
int DSP_VectorChkEqu32(int32_t* indata1, int32_t *indata2, int N);
void DSP_VectorCopy(int32_t *outdata, int32_t *indata, int N);
void DSP_VectorCopyReverse32(int32_t *outdata, int32_t *indata, int N);
void DSP_VectorDivC(_Q16 *outVector, _Q16 *inVector, _Q16 divisor, int N);
int16_t DSP_VectorDotp16(int16_t *indata1, int16_t *indata2, int N, int scale);
int32_t DSP_VectorDotp32(int32_t *indata1, int32_t *indata2, int N, int scale);
void DSP_VectorExp(_Q16 *outVector, _Q16 *inVector, int N);
void DSP_VectorFill(int32_t *indata, int32_t data, int N);
void DSP_VectorLog10(_Q3_12 *outVector, _Q16 *inVector, int N);
void DSP_VectorLog2(_Q5_10 *outVector, _Q16 *inVector, int N);
void DSP_VectorLn(_Q4_11 *outVector, _Q16 *inVector, int N);
int32_t DSP_VectorMax32(int32_t *indata, int N);
int DSP_VectorMaxIndex32(int32_t *indata, int N);
int32_t DSP_VectorMean32(int32_t *indata, int N);
int32_t DSP_VectorMin32(int32_t *input, int N);
int DSP_VectorMinIndex32(int32_t *indata, int N);
void DSP_VectorMul16(int16_t *outdata, int16_t *indata1, int16_t *indata2, int N);
void DSP_VectorMul32(int32_t *outdata, int32_t *indata1, int32_t *indata2, int N);
void DSP_VectorMulc16(int16_t *outdata, int16_t *indata, int16_t c, int N);
void DSP_VectorMulc32(int32_t *outdata, int32_t *indata, int32_t c, int N);
void DSP_VectorNegate(int32_t *outdata, int32_t *indata, int N);
void DSP_VectorRecip(_Q16 *outVector, _Q16 *inVector, int N);
int16_t DSP_VectorRMS16(int16_t *inVector, int N);
void DSP_VectorShift(int32_t *outdata, int32_t *indata, int N, int shift);
int16_t DSP_VectorStdDev16(int16_t *inVector, int N);
void DSP_VectorSqrt(_Q16 *outVector, _Q16 *inVector, int N);
void DSP_VectorSub16(int16_t *outdata, int16_t *indata1, int16_t *indata2, int N);
void DSP_VectorSub32(int32_t *outdata, int32_t *indata1, int32_t *indata2, int N);
int16_t DSP_VectorSumSquares16(int16_t *indata, int N, int scale);
int32_t DSP_VectorSumSquares32(int32_t *indata, int N, int scale);
int16_t DSP_VectorVari16(int16_t *inVector, int N);
int32_t DSP_VectorVariance(int32_t *inVector, int N);
void DSP_VectorZeroPad(int32_t *indata, int N);

File Name

dsp.h

Company

Microchip Technology Inc.

LibQ Fixed-Point 'C' Math Library

This topic describes the LibQ Fixed-Point 'C' Math Library.

Introduction

The LibQ Fixed-Point 'C' Math Library is available for the PIC32 family of microcontrollers.

Description

The LibQ Fixed-Point 'C' Math Library provides fixed-point math functions written in C for portability between core processors.

The library utilizes signed fixed point types (fractional Q types specified by Qn.m) as follows:

Qndm where:

- n is the number of data bits to the left of the radix point
- m is the number of data bits to the right of the radix point
- a signed bit is implied (unless stated otherwise)

Whereas, the implied fractional scaling of the integer value is given by 2^(-m), i.e. arithmetic left-shift of m bit positions of the fractional part into the value. This necessarily reduces the maximum magnitude of that can be stored in the n+m bit value, since only n bit remain and 1 of those bits is usually the sign..

For convenience, short names are also defined for arbitrary scaled fractional types:

- q15 is signed fractional 16 bit value
- q31 is signed fractional 32 bit value

In addition, A pseudo floating point 32 bit format (FxQFloat32) is defined that consists of 16 bit mantissa and a 16 bit exponent (base 2).

Functions in the library are prefixed with the type of the return value and followed by the argument types, as follows:

libq_: libq_q15_sin_Q2d13

For example, libq_q1d15_Sin_q10d6 returns a Q1.15 value equal to the to the sine of an angle specified as a Q10.6 value (in degrees between 0 and 360). Argument types separated by an underscore.

For arbitrary scaled types (q15 and q31), the scaling of the result will depend on the function and the scaling of the arguments. For instance, libq_q15_Add_q15_q15(a,b) will return a scaled value type that is the same as the two input types (which also must have equivalent Q format).

The library supports the original ETSI family of basic operations as described in:

"ETSI G.729 recommendation Coding of speech at 8 kbit/s using conjugate-structure algebraic-code excited linear prediction (CS-ACELP)".

This includes:

- Saturated arithmetic operations, such as L_add, L_sub, sub, add.
- Multiplication operations, such as mult, mult_r and L_mult.
- Arithmetic shift operations, such as shl and shr.
- Data conversion operations, such as extract_l, extract_h, and round.
- Normalization to maximize resolution, such as norm_l
- Loading of 15 bit values into 32 bit values, such as L_deposit_h
- Signed negation, such as L_negate
- Multiply accumulate, such as L_mac, L_macNs, mac_r
- Multiply subtract, such as L_msu, L_msuNs msu_r
- Divide, such as div_s
- Saturate value, such as L_sat
- Rounding, such as round.
- Left Shift, such as L_shl and shl
- Right Shift, such as L_shr, L_shr_r and shr
- Extraction of 16 bit values, such as extract_l
- Absolute value, such as L_abs

The 'C' Code implementation provides portability of the library, yet the functions easily translate to specific processor assembly code (intrinsic operations) when needed.

Note: The LibQ Fixed-Point 'C' Math Library functions do not correspond to the LibQ Fixed-Point Math Library, which is optimized for the microAptiv core processor and is written in Assembly language.

Using the Library

This topic describes the basic architecture of the LibQ Fixed-Point 'C' Math Library and provides information and examples on its use.

Description

Interface Header File: `libq_C.h`

The interface to the LibQ Fixed-Point 'C' Math Library is defined in the `libq_C.h` header file. Any C language source (`.c`) file that uses the LibQ Fixed-Point Library should include `libq_C.h`.

Library File:

The LibQ Fixed-Point 'C' Math Library archive (`.a`) file is installed with MPLAB Harmony.

Library Interface

Functions

	Name	Description
	libq_q15_Abs_q15	Saturated Absolute value.
	libq_q15_Add_q15_q15	Add two 16-bit 2s-complement fractional values.
	libq_q15_DivisionWithSaturation_q15_q15	Fractional division with saturation.
	libq_q15_ExtractH_q31	Extracts upper 16 bits of input 32-bit fractional value.
	libq_q15_ExtractL_q31	Extracts lower 16-bits of input 32-bit fractional value. Descriptionf Extracts lower 16-bits of input 32-bit fractional value and returns them as 16-bit fractional value. This is a bit-for-bit extraction of the bottom 16-bits of the 32-bit input. This function relates to the ETSI extract_l function.
	libq_q15_Negate_q15	Negate 16-bit 2s-complement fractional value with saturation.
	libq_q15_RoundL_q31	Rounds the lower 16-bits of the 32-bit fractional input.
	libq_q15_ShiftLeft_q15_q15	'Arithmetic' Shift of the 16-bit input argument.
	libq_q15_ShiftRight_q15_q15	'Arithmetic' RIGHT Shift on a 16-bit value.
	libq_q15_ShiftRightRound_q15_q15	Performs an 'Arithmetic' RIGHT Shift on a 16-bit input.
	libq_q1d15_Sin_q10d6	Approximates the sine of an angle.
	libq_q31_Abs_q31	Saturated Absolute value.
	libq_q31_Add_q31_q31	Add two 32-bit 2s-complement fractional values.
	libq_q31_DepositH_q15	Place 16 bits in the upper half of 32 bit word.
	libq_q31_DepositL_q15	Place 16 bits in the lower half of 32 bit word.
	libq_q31_Mult2_q15	fractional multiplication of two 16-bit fractional values.
	libq_q31_Multi_q15_q31	Implement 16 bit by 32 bit multiply.
	libq_q31_Negate_q31	Negate 32-bit 2s-complement fractional value with saturation.
	libq_q31_ShiftRight_q31_q15	'Arithmetic' RIGHT Shift on a 32-bit value.
	libq_q31_ShiftRightRound_q31_q15	'Arithmetic' RIGHT Shift on a 32-bit value
	libq_q20d12_Sin_q20d12	3rd order Polynomial apprx. of a sine function
	libq_q15_ExpAvg_q15_q15_q1d15	Exponential averaging
	libq_q15_Sub_q15_q15	Subtract two 16-bit 2s-complement fractional values
	libq_q1d15_MacR_q1d31_q1d15_q1d15	Multiply accumulate with rounding.
	libq_q1d15_MsuR_q1d31_q1d15_q1d15	Multiply-Subtraction with rounding
	libq_q1d15_MultipyR2_q1d15_q1d15	fractional multiplication of two 16-bit fractional values giving a 16 bit rounded result.
	libq_q1d31_Mac_q1d31_q1d15_q1d15	Multiply-Accumulate function WITH saturation
	libq_q1d31_Msu_q1d31_q1d15_q1d15	L_msu(a,b,c)
	libq_q31_ShiftLeft_q31_i16	'Arithmetic' Shift of the 32-bit value.
	Fx16Norm	Normalize the 16-bit fractional value.
	Fx32Norm	Normalize the 32-bit number.

Data Types and Constants

	Name	Description
	q15	q15 n.n (signed)
	q31	q31 n.n (signed)
	q63	Q63 n.n (signed)
	_LIBQ_C_H_	This is macro _LIBQ_C_H_.
	BITMASKFRACT16	Bit Mask for 16
	BITMASKFRACT32	Bit Mask for 32
	Fl2Fract16	Converts floating point constant value to fractional 16-bit value

	Fl2Fract32	Converts floating point constant value to fractional 32-bit value
	Fl2FxPnt	Converts floating point constant value to fixed/fractional value
	Fl2FxPnt16	Converts floating point constant value to fixed point 16-bit value
	Fl2FxPnt32	Converts floating point constant value to fixed point 32-bit value
	Fl2Int16	Converts floating point constant expression to 16-bit integer value
	Fl2Int32	Converts floating point constant expression to 32-bit integer value
	FrMax	find the maximum of two numbers
	FrMin	find the minimum of two numbers
	LOG102Q5D11	log10(2) scaled to Q5.11 format
	MAXFRACT16	0.999969
	MAXFRACT32	0.9999999995
	MAXINT16	Maximum and minimum values for 16-bit data types.
	MAXINT32	This is macro MAXINT32.
	MAXPFLOAT32	minimum and maximum definitions for FX floating point data types
	MINFRACT16	1.000000
	MINFRACT32	1.0000000000
	MININT16	This is macro MININT16.
	MININT32	This is macro MININT32.
	MINPFLOAT32	This is macro MINPFLOAT32.
	MSBBITFRACT16	16-bit Sign Bit
	MSBBITFRACT32	32-bit Sign Bit
	NINETYQ10D22	Ninety degrees scaled to Q10d22
	NINETYQ10D6	Ninety degrees scaled to Q10d6
	NORMNEGFRACT16	Max -val for 16
	NORMNEGFRACT32	Max -val for 32
	NORMPOSFRACT16	Min +val for 16
	NORMPOSFRACT32	Min +val for 32
	NUMBITSFRACT16	Num of bits 16
	NUMBITSFRACT32	Num of bits 32
	ONEEIGHTYQ10D22	180 degrees scaled to Q10d22
	ONEEIGHTYQ10D6	180 degrees scaled to Q10d6
	ROUNDFRACT32	Rounding value
	THREESIXTYQ10D22	360 degrees scaled to Q10d22
	THREESIXTYQ10D6	360 degrees scaled to Q10d6
	TWOSEVENTYQ10D22	270 degrees scaled to Q10d22
	TWOSEVENTYQ10D6	270 degrees scaled to Q10d6
	UNITYFLOAT	This is macro UNITYFLOAT.
	FxQFloat32	FxQFloat32 pseudo floating point type (limited floating point)
	Exponent16ToQFloat32	Converts a power of 2 16-bit integer to 32-bit floating point value.
	Fl2QFloat32	Converts a decimal floating-point constant expression to pseudo float
	i16	Q16d0

Description

This section describes the Application Programming Interface (API) functions, macros, and types of the LibQ Fixed Point 'C' Math Library. Refer to each section for a detailed description.

Functions

libq_q15_Abs_q15 Function

Saturated Absolute value.

File

libq_C.h

C

```
q15 libq_q15_Abs_q15(q15);
```

Returns

q15 result - abs(input) <= MAXFRACT16

Description

Function libq_q15_Abs_q15:

Creates a saturated Absolute value. It takes the absolute value of the 16-bit 2s-complement fractional input with saturation. The saturation is for handling the case where taking the absolute value of MINFRACT16 is greater than MAXFRACT16, or the allowable range of 16-bit values. This function relates to the ETSI abs function.

Parameters

Parameters	Description
q15 a	input argument

libq_q15_Add_q15_q15 Function

Add two 16-bit 2s-complement fractional values.

File

libq_C.h

C

```
q15 libq_q15_Add_q15_q15(q15, q15);
```

Returns

q15 - a+b on Range: MINFRACT16 <= result <= MAXFRACT16

Description

Function libq_q15_Add_q15_q15: f

Add two 16-bit 2s-complement fractional (op1 + op2) to produce a 16-bit 2s-complement fractional result with saturation. The saturation is for handling the overflow/underflow cases, where the result is set to MAX16 when an overflow occurs and the result is set to MIN16 when an underflow occurs. This function does not produce any status flag to indicate when an overflow or underflow has occured. It is assumed that the binary point is in exactly the same bit position for both 16-bit inputs and the resulting 16-bit output.

libq_q15_DivisionWithSaturation_q15_q15 Function

Fractional division with saturation.

File

libq_C.h

C

```
q15 libq_q15_DivisionWithSaturation_q15_q15(q15, q15);
```

Returns

q15 result - ratio a/b in 16-bit fractional format

Description

Function libq_q15_DivisionWithSaturation_q15_q15():

Performs fractional division with saturation. There are three restrictions that the calling code must satisfy.

1. Both the numerator and denominator must be positive.
2. In order to obtain a non-saturated result, the numerator must be LESS than or equal to the denominator.
3. The denominator must not equal zero.

If 'num' equals 'den', then the result equals MAXINT16.

This function relates to the ETSI div_s function.

Parameters

Parameters	Description
q15 num	16-bit fractional numerator
q15 den	16-bit fractional denumerator

libq_q15_ExtractH_q31 Function

Extracts upper 16 bits of input 32-bit fractional value.

File

libq_C.h

C

```
q15 libq_q15_ExtractH_q31(q31);
```

Returns

q15 result - Upper 16 bits of 32-bit argument a

Description

Function libq_q15_ExtractH_q31:

Extracts upper 16 bits of input 32-bit fractional value and returns them as 16-bit fractional value. This is a bit-for-bit extraction of the top 16-bits of the 32-bit input. This function relates to the ETSI extract_h function.

libq_q15_ExtractL_q31 Function

Extracts lower 16-bits of input 32-bit fractional value.

Descriptionf Extracts lower 16-bits of input 32-bit fractional value and returns them as 16-bit fractional value. This is a bit-for-bit extraction of the bottom 16-bits of the 32-bit input. This function relates to the ETSI extract_l function.

File

libq_C.h

C

```
q15 libq_q15_ExtractL_q31(q31);
```

Returns

q15 - Lower 16 bits of 32-bit argument a

Description

Function libq_q15_ExtractL_q31:

libq_q15_Negate_q15 Function

Negate 16-bit 2s-complement fractional value with saturation.

File

libq_C.h

C

```
q15 libq_q15_Negate_q15(q15);
```

Returns

q15 result on range: MINFRACT16 <= result <= MAXFRACT16

Description

Function libq_q15_Negate_q15:

Negate 16-bit 2s-complement fractional value with saturation. The saturation is for handling the case where negating a MINFRACT16 is greater than MAXFRACT16, or the allowable range of values. This function relates to the ETSI negate function.

libq_q15_RoundL_q31 Function

Rounds the lower 16-bits of the 32-bit fractional input.

File

libq_C.h

C

```
q15 libq_q15_RoundL_q31(q31);
```

Returns

q15 result

Description

Function libq_q15_RoundL_q31:

Rounds the lower 16-bits of the 32-bit fractional input into a 16-bit fractional value with saturation. This converts the 32-bit fractional value to 16-bit fractional value with rounding. This function calls the 'Add' function to perform the 32-bit rounding of the input value and 'ExtractH' function to extract to top 16-bits. This has the effect of rounding positive fractional values up and more positive, and has the effect of rounding negative fractional values up and more positive. This function relates to the ETSI round function.

libq_q15_ShiftLeft_q15_q15 Function

'Arithmetic' Shift of the 16-bit input argument.

File

libq_C.h

C

```
q15 libq_q15_ShiftLeft_q15_q15(q15, int16_t);
```

Returns

q15 result - arithmetically shifted 16-bit signed integer output

Description

Function libq_q15_ShiftLeft_q15_i16:

Performs an 'Arithmetic' Shift of the 16-bit input argument 'a' left by the input argument 'b' bit positions. If 'b' is a positive number, a 16-bit left shift is performed with 'zeros' inserted to the right of the shifted bits. If 'b' is a negative number, a right shift by abs(b) positions with 'sign extention' Saturation is applied if shifting causes an overflow or an underflow.

positive value: # of bits to left shift (zeros inserted at LSB's) {To not always saturate: if 'a=0', then max b=15, else max b=14} negative value: # of bits to right shift (sign extend)

This function relates to the ETSI shl function.

Parameters

Parameters	Description
q15 a	16-bit signed integer value to be shifted.
i16 b	16-bit signed integer shift value

libq_q15_ShiftRight_q15_q15 Function

'Arithmetic' RIGHT Shift on a 16-bit value.

File

libq_C.h

C

```
q15 libq_q15_ShiftRight_q15_q15(q15, int16_t);
```

Returns

q15 result - 16-bit signed shifted output

Description

Function libq_q15_ShiftRight_q15_q15:

Performs an 'Arithmetic' RIGHT Shift on a 16-bit input by 'b' bit positions. For positive shift directions (b>0), 'b' Lsb-bits are shifted out to the right and 'b' sign-extended Msb-bits fill in From the left. For negative shift directions (b<0), 'b' Lsb's are shifted to the LEFT with 0's filling in the empty lsb position. The left shifting causes 'b' Msb-bits to fall off to the left, saturation is applied to any shift left value that overflows. This function calls the left-shift function to perform any 16-bit left shifts. This function does not provide any status-type information to indicate when overflows occur.

positive value: # of bits to right shift (sign extend) { To get all sign bits, b>=15 } negative value: # of bits to left shift (zeros inserted at LSB's)

This function relates to the ETSI shr function.

Parameters

Parameters	Description
q15 a	16-bit signed input value to shift
i16 b	16-bit signed integer shift index

libq_q15_ShiftRightRound_q15_q15 Function

Performs an 'Arithmetic' RIGHT Shift on a 16-bit input.

File

libq_C.h

C

```
q15 libq_q15_ShiftRightRound_q15_q15(q15, int16_t);
```

Returns

q15 result - Arithmetically shifted 16-bit signed integer output

Description

Function libq_q15_ShiftRightRound_q15_q15:

Performs an 'Arithmetic' RIGHT Shift on a 16-bit input by 'b' bits with Rounding applied. The rounding occurs by adding a bit weight of "1/2 Lsb", where the "Lsb" is the Ending (shifted) Lsb. For example: The initial Bit#(b) is after the right shift Bit#(0), so the rounding bit weight is Bit#(b-1). Rounding does not occur on either left shifts or on no shift needed cases. For positive shift directions (b>0), 'b' Lsb-bits are shifted out to the right and 'b' sign-extended Msb-bits fill in From the left. For negative shift directions (b<0), 'b' Lsb's are shifted to the LEFT with 0's filling in the empty lsb position. The left shifting causes 'b' Msb-bits to fall off to the left, saturation is applied to any shift left value that overflows. This function calls the left-shift function to perform the actual 16-bit left shift. This function does not provide any status-type information to indicate when overflows occur.

positive value: # of bits to right shift (sign extend) {b > 15, results in all sign bits} negative value: # of bits to left shift (zeros inserted at LSB's)

This function relates to the ETSI shr_r function.

Parameters

Parameters	Description
q15 a	16-bit signed integer value to be shifted
i16 b	16-bit signed integer shift index

libq_q1d15_Sin_q10d6 Function

Approximates the sine of an angle.

File

libq_C.h

C

```
q15 libq_q1d15_Sin_q10d6(q15 angleQ10d6);
```

Returns

q15 sine(angle) value in Q1.15

Description

Function libq_q1d15_Sin_q10d6:

This function approximates the sine of an angle using the following algorithm: sin(x) = 3.140625x + 0.02026367x^2 - 5.325196x^3 + 0.5446778x^4 + 1.800293x^5. The approximation is accurate for any value of x from 0 degrees to 90 degrees. Because sin(-x) = - sin(x) and sin(x) = sin(180 - x), the sine of any angle can be inferred from an angle in the first quadrant. Therefore, any angle > 90 is converted to an angle between 0 & 90. The coefficients of the algorithm have been scaled by 1/8 to fit a Q1d15 format. So the result is scaled up by 8 to obtain the proper magnitudes. The algorithm expects the angle to be in degrees and represented in Q10.6 format. The computed sine value is returned in Q1.15 format.

Preconditions

none.

Parameters

Parameters	Description
q15 angle	The angle in degrees for which the sine is computed in Q10.6

libq_q31_Abs_q31 Function

Saturated Absolute value.

File

libq_C.h

C

```
q31 libq_q31_Abs_q31(q31);
```

Returns

q31 result - abs(a) <= MAXFRACT32

Description

Function libq_q31_Abs_q31:

Creates a saturated Absolute value. It takes the absolute value of the 32-bit 2s-complement fractional input with saturation. The saturation is for handling the case where taking the absolute value of MINFRACT32 is greater than MAXFRACT32, or the allowable range of 32-bit values.

This function relates to the ETSI L_abs function.

libq_q31_Add_q31_q31 Function

Add two 32-bit 2s-complement fractional values.

File

libq_C.h

C

```
q31 libq_q31_Add_q31_q31(q31, q31);
```

Returns

q31 result a+b on range: MINFRACT32 <= result <= MAXFRACT32

Description

Function libq_q31_Add_q31_q31:

Add two 32-bit 2s-complement fractional (op1 + op2) to produce a 32-bit 2s-complement fractional result with saturation. The saturation is for handling the overflow/underflow cases, where the result is set to MAX32 when an overflow occurs and the result is set to MIN32 when an underflow occurs. This function does not produce any status flag to indicate when an overflow or underflow has occured. It is assumed that the binary point is in exactly the same bit position for both 32-bit inputs and the resulting 32-bit output. This function relates to the ETSI L_add function.

libq_q31_DepositH_q15 Function

Place 16 bits in the upper half of 32 bit word.

File

libq_C.h

C

```
q31 libq_q31_DepositH_q15(q15);
```

Returns

q31 result 16-bits of a in upper MSB's and zeros in the lower LSB's

Description

Function libq_q31_DepositH_q15:

Composes a 32-bit fractional value by placing the input 16-bit fractional value in the composite MSB's and zeros the composite 16-bit LSB's This is a bit-for-bit placement of input 16-bits into the upper part of 32-bit result.

This function relates to the ETSI L_deposit_H function.

libq_q31_DepositL_q15 Function

Place 16 bits in the lower half of 32 bit word.

File

libq_C.h

C

```
q31 libq_q31_DepositL_q15(q15);
```

Returns

q31 result - SignExtended 16-bit MSB's and a Value in lower 16-bit LSB's

Description

Function libq_q31_DepositL_q15:

Composes a 32-bit fractional value by placing the 16-bit Fraction input value into the lower 16-bits of the 32-bit composite value. The 16-bit MSB's of the composite output are sign extended. This is a bit-for-bit placement of input 16-bits into the bottom portion of the composite 32-bit result with sign extention. This function relates to the ETSI L_deposit_l function.

libq_q31_Mult2_q15 Function

fractional multiplication of two 16-bit fractional values.

File

libq_C.h

C

```
q31 libq_q31_Mult2_q15(q15, q15);
```

Returns

q31, a*b

Description

Function libq_q31_Mult2_q15:

Performs fractional multiplication of two 16-bit fractional values and returns the 32-bit fractional scaled result. The function performs the Q15xQ15->Q30 fractional bit multiply. It then shifts the result left by '1', to give a Q31 type result, (the lsb is zero-filled). This automatic shift left is done to get rid of the extra sign bit that occurs in the interpretation of the fractional multiply result. Saturation is applied to any results that overflow, and then the function returns the 32-bit fractional q31 result. This function is for fractional 'Q' data only and it therefore will not give correct results for true integers (because left shift by '1'). For the special case where both inputs equal the MINFRACT16, the function returns a value equal to MAXFACT32, i.e. 0x7fffffff = libq_q15_mult_q15_q31(0x8000,0x8000).

This function relates to the ETSI L_mult function.

Parameters

Parameters	Description
q15 a	multiplicand a
q15 b	multiplicand b

libq_q31_Multi_q15_q31 Function

Implement 16 bit by 32 bit multiply.

File

libq_C.h

C

```
q31 libq_q31_Multi_q15_q31(q15 argAQ1d15, q31 argBQ1d31);
```

Returns

q31 result - a*b rounded

Description

Function libq_q31_Multi_q15_q31():

Implement 16 bit by 32 bit multiply as shown below The 's' and 'u' notation shows the processing of signed and unsigned numbers.

-B1- -B0- s u 2nd argument is 32 bits -A0- s 1st argument is 16 bits

----

A0B0 A0B0 s=s*u 1st 32-bit product is A0*B0 A0B1 A0B1 s=s*s 2nd 32-bit product is A0*B1

-S2- -S1- -S0- s=s+s 48-bit result is the sum of products -P1- -P0- 32-bit return is the most significant bits of sum

The algorithm is implemented entirely with the fractional arithmetic library. The unsigned by signed multiply is implemented by shifting bits 15:1 to bits 14:0 of a 16-bit positive fractional number, which throws away bit 0 of the 32-bit number. Since that affects result bits that are used for rounding, rounding processing is included. Saturation processing is handled implicitly in the fractional arithmetic library, except for the case of maximum negative numbers.

libq_q31_Negate_q31 Function

Negate 32-bit 2s-complement fractional value with saturation.

File

libq_C.h

C

```
q31 libq_q31_Negate_q31(q31);
```

Returns

q31 result on range: MINFRACT32 <= result <= MAXFRACT32

Description

Function libq_q31_Negate_q31:

Negate 32-bit 2s-complement fractional value with saturation. The saturation is for handling the case where negating a MINFRACT32 is greater than MAXFRACT32, or the allowable range of values. This function relates to the ETSI L_negate function.

libq_q31_ShiftRight_q31_q15 Function

'Arithmetic' RIGHT Shift on a 32-bit value.

File

libq_C.h

C

```
q31 libq_q31_ShiftRight_q31_q15(q31, int16_t);
```

Returns

q31 result - Arithmetically shifted 32-bit signed integer output

Description

Function libq_q31_ShiftRight_q31_q15:

Performs an 'Arithmetic' RIGHT Shift on a 32-bit input by 'b' bit positions. For positive shift directions (b>0), 'b' Lsb-bits are shifted out to the right and 'b' sign-extended Msb-bits fill in From the left. For negative shift directions (b<0), 'b' Lsb's are shifted to the LEFT with 0's filling in the empty lsb position. The left shifting causes 'b' Msb-bits to fall off to the left, saturation is applied to any shift left value that overflows. This function calls the left-shift function to perform any 32-bit left shifts. This function does not provide any status-type information to indicate when overflows occur.

positive value: # of bits to right shift (sign extend) negative value: # of bits to left shift (zeros inserted at LSB's)

This function relates to the ETSI L_shr function.

Parameters

Parameters	Description
q31 a	32-bit signed integer value to be shifted
i16 b	16-bit signed integer shift index

libq_q31_ShiftRightRound_q31_q15 Function

'Arithmetic' RIGHT Shift on a 32-bit value

File

libq_C.h

C

```
q31 libq_q31_ShiftRightRound_q31_q15(q31, int16_t);
```

Returns

q31 result - Arithmetically shifted 32-bit signed integer output

Description

Function libq_q31_ShiftRightRound_q31_q15:

Performs an 'Arithmetic' RIGHT Shift on a 32-bit input by 'b' bits with Rounding applied. The rounding occurs before any shift by adding a bit weight of "1/2 Lsb", where the "Lsb" is the Ending (shifted) Lsb. For example: The initial Bit#(i+b) is after the right shift Bit#(i), so the rounding bit weight is Bit#(i+b-1). Rounding does not occur on left shifts, when b is negative. After rounding, this function calls the right-shift function to perform the actual 32-bit right shift. For positive shift directions (b>0), 'b' Lsb-bits are shifted out to the right and 'b' sign-extended Msb-bits fill in From the left. For negative shift directions (b<0), 'b' Lsb's are shifted to the LEFT with 0's filling in the empty lsb position. The left shifting causes 'b' Msb-bits to fall off to the left, saturation is applied to any shift left value that overflows. This function calls the left-shift function to perform the actual 32-bit left shift. This function does not provide any status-type flag to indicate occurence of overflow.

positive value: # of bits to right shift (sign extend) negative value: # of bits to left shift (zeros inserted at LSB's)

This function relates to the ETSI L_shr_r function.

Parameters

Parameters	Description
q31 a	32-bit signed integer value to be shifted
i16 b	16-bit signed integer shift index

libq_q20d12_Sin_q20d12 Function

3rd order Polynomial apprx. of a sine function

File

libq_C.h

C

```
q31 libq_q20d12_Sin_q20d12(q31 angQ20d12);
```

Returns

q31 Sine(angle) value in Q20.12

Description

Function libq_q20d12_Sin_q20d12:

3rd order Polynomial apprx. of a sine function

Preconditions

none.

Parameters

Parameters	Description
q31 angle	The angle in radians for which the sine is computed in Q20.12

libq_q15_ExpAvg_q15_q15_q1d15 Function

Exponential averaging

File

libq_C.h

C

```
q15 libq_q15_ExpAvg_q15_q15_q1d15(q15 prevAvgQ15, q15 newMeasQ15, q15 lamdaQ1d15);
```

Returns

q15 result - S(k+1) = S(k)*L + X(k)*(1-L)

Description

Function libq_q15_ExpAvg_q15_q15_q1d15()

Exponential averaging implements a smoothing function based on the form: avg[i+1] = avg[i] * lamda + new * (1-lamda) In this implementation, is has been optimized as follows. avg[i+1] = (avg[i] - new) * lamda + new

The optimization precludes accurate processing of new numbers that differ from the current average by more than unity. If the difference is greater than unity or less than negative unity, the difference is saturated.

The effect is akin to a smaller lambda, e.g., the new value will have a greater weight than expected. If the smoothing is of data that is entirely positive or entirely negative, then the saturation will not be an issue.

Parameters

Parameters	Description
q15 S(k)	Previous exponential average
q15 X(k)	New value to be averaged in
q15 L	exponential averaging constant in Q1.15

libq_q15_Sub_q15_q15 Function

Subtract two 16-bit 2s-complement fractional values

File

libq_C.h

C

```
q15 libq_q15_Sub_q15_q15(q15, i16);
```

Returns

q15 result a+b on range: MINFRACT16 <= result <= MAXFRACT16

Description

Function libq_q15_Sub_q15_q15:

Subtract two 16-bit 2s-complement fractional (op1 - op2) to produce a 16-bit 2s-complement fractional difference result with saturation. The saturation is for handling the overflow/underflow cases, where the result is set to MAX16 when an overflow occurs and the result is set to MIN16 when an underflow occurs. This function does not produce any status flag to indicate when an overflow or underflow has occured. It is assumed that the binary point is in exactly the same bit position for both 16-bit inputs and the resulting 16-bit output. This function relates to the ETSI sub function.

libq_q1d15_MacR_q1d31_q1d15_q1d15 Function

Multiply accumulate with rounding.

File

libq_C.h

C

```
q15 libq_q1d15_MacR_q1d31_q1d15_q1d15(q31, q15, q15);
```

Returns

q15 result - a+b*c rounded

Description

Function libq_q15_MacR_q31_q15_q15:

This function is multiply-accumulate WITH Rounding applied to the accumulator result before it is saturated and the top 16-bits taken. This function first multiplies the two 16-bit input values 'b x c' which results in a 32-bit value. This result is left shifted by one to account for the extra sign bit inherent in the fractional-type multiply. So, the shifted number now has a '0' in the Lsb. The shifted multiplier output is then added to the 32-bit fractional input 'a'. Then the 32-bits of the accumulator output are rounded by adding '2^15'. This value is then saturated to be within the q15 range. It is assumed that the binary point of the 32-bit input value a is in the same bit position as the shifted multiplier output. This function is for fractional Qtype format data only and it therefore will not give the correct results for true integers. This function relates to the ETSI L_mac_r function.

Parameters

Parameters	Description
q31 a	32-bit accumulator operand
q15 b	16-bit multiplication operand
q15 c	16-bit multiplication operand

libq_q1d15_MsuR_q1d31_q1d15_q1d15 Function

Multiply-Subtraction with rounding

File

libq_C.h

C

```
q15 libq_q1d15_MsuR_q1d31_q1d15_q1d15(q31, q15, q15);
```

Returns

q15 result - a-b*c rounded to Q1.15

Description

Function libq_q15_MsuR_q31_q15_q15:

This function is like Multiply-Subtract but WITH Rounding applied to the subtractor result before it is saturated and the top 16-bits taken. This function first multiplies the two 16-bit input values 'b x c' which results in a 32-bit value. This result is left shifted by one to account for the extra sign bit inherent in the fractional-type multiply. So, the shifted number now has a '0' in the Lsb. The shifted multiplier output is then SUBTRACTED From the 32-bit fractional input 'a'. Then the 32-bits output From this subtraction are rounded by adding '2^15'. This value is then saturated to be within the q15 range. It is assumed that the binary point of the 32-bit input value a is in the same bit position as the shifted multiplier output.

This function is for fractional Qtype format data only and it therefore will not give the correct results for true integers.

This function relates to the ETSI msu_r function.

Parameters

Parameters	Description
q31 a	Value which is subtracted from
q15 b	multiplication operand 1
q15 c	multiplication operand 2

libq_q1d15_MultipyR2_q1d15_q1d15 Function

fractional multiplication of two 16-bit fractional values giving a 16 bit rounded result.

File

libq_C.h

C

```
q15 libq_q1d15_MultipyR2_q1d15_q1d15(q15, q15);
```

Returns

q15 result - a*b rounded 16-bit signed integer (Q1.15) output value

Description

Function libq_q1d15_MultipyR2_q1d15:

Performs fractional multiplication of two 16-bit fractional values and returns a ROUNDED 16-bit fractional result. The function performs a Q15xQ15->Q30 bit multiply with a left shift by '1' to give a Q31 result. This automatic shift left is done to get rid of the extra sign bit that occurs in the interpretation of the fractional multiply result. Saturation is applied to any 32-bit result that overflows. Rounding is applied to the 32-bit SHIFTED result by adding in a weight factor of 2^15, again any overflows are saturated. The TOP 16-bits are extracted and returned. This function is for fractional 'Qtype' data only and it therefore will not give the correct results for true integers (because left shift by '1'). This function assumes that the binary point in the 32-bit shifted multiplier output is between bit_16 and bit_15 when the rounding factor is added. For the special case where both inputs equal the MINFACT16, the function returns a value equal to MAXFACT16, i.e. 0x7fff = FrsssMult(0x8000,0x8000). This function internally calls the libq_q15_mult_q15_q31() routine to perform the actual multiplication and the rounding routine to perform the actual rounding.

This function relates to the ETSI mult_r function.

Parameters

Parameters	Description
q15 a	value in Q1.15
q15 b	value in Q1.15

libq_q1d31_Mac_q1d31_q1d15_q1d15 Function

Multiply-Accumulate function WITH saturation

File

libq_C.h

C

```
q31 libq_q1d31_Mac_q1d31_q1d15_q1d15(q31, q15, q15);
```

Returns

q31 result, a+b*c saturated

Description

Function libq_q31_Mac_q31_q15_q15():

Performs a Multiply-Accumulate function WITH saturation. This routine returns the fully fractional 32-bit result From the accumulator output 'SAT(addOut_Q1d31)=outQ1d15' where 'multOut_Q1d31 + a_Q1d31 = addOut_Q1d31', and 'b_Q1d15 x c_Q1d15 = multOut_Q1d31'. The multiply is performed on the two 16-bit fully fully fractional input values 'b x c' which results in a 32-bit value. This result is left shifted by one to account for the extra sign bit inherent in the fully fully fractional-type multiply. The shifted number represents a Q1d31 number with the lsb set to '0'. This Q1d31 number is added with the 32-bit fully fully fractional input argument 'a'. Saturation is applied on the output of the accumulator to keep the value within the 32-bit fully fractional range and then this value is returned. This function is for fully fractional Qtype format data only and it therefore will not give the correct results for true integers.

This function relates to the ETSI L_mac function.

Parameters

Parameters	Description
q31 a	32-bit accumulator operand 1 in Q1d31
q15 b	16-bit multiplication operand 1 in Q1d15
q15 c	16-bit multiplication operand 2 in Q1d15

libq_q1d31_Msu_q1d31_q1d15_q1d15 Function

File

libq_C.h

C

```
q31 libq_q1d31_Msu_q1d31_q1d15_q1d15(q31, q15, q15);
```

Description

L_msu(a,b,c)

libq_q31_ShiftLeft_q31_i16 Function

'Arithmetic' Shift of the 32-bit value.

File

libq_C.h

C

```
q31 libq_q31_ShiftLeft_q31_i16(q31, i16);
```

Returns

q31 result - arithmetically shifted 32-bit signed integer output

Description

Function libq_q31_ShiftLeft_q31_i16:

Performs an 'Arithmetic' Shift of the 32-bit input argument 'a' left by the input argument 'b' bit positions. If 'b' is a positive number, a 32-bit left shift is performed with 'zeros' inserted to the right of the shifted bits. If 'b' is a negative number, a 32-bit right shift by b bit positions with 'sign extention':

positive value: # of bits to left shift (zeros inserted at LSB's) negative value: # of bits to right shift (sign extend)

Saturation is applied if shifting causes an overflow or an underflow.

This function relates to the ETSI L_shl function.

Parameters

Parameters	Description
q31 a	32-bit signed integer value to be shifted
i16	16-bit signed integer shift index

Fx16Norm Function

Normalize the 16-bit fractional value.

File

libq_C.h

C

```
int16_t Fx16Norm(q15);
```

Returns

i16 result - The number of left shifts required to normalize

Description

Function Fx16Norm:

Produces then number of left shifts needed to Normalize the 16-bit fully fractional input. If the input 'a' is a positive number, it will produce the number of left shifts required to normalized it to the range of a minimum of [(MAXFRACT16+1)/2] to a maximum of [MAXFRACT16]. If the input 'a' is a negative number, it will produce the number of left shifts required to normalized it to the range of a minimum of [MINFRACT16] to a maximum of [MINFRACT16/2]. This function does not actually normalize the input, it just produces the number of left shifts required. To actually normalize the value the left shift function should be used with the value returned From this function.

the 16-bit input on range: 0 => result < 16 (i.e. NUMBITSFRACT16) If a>0: 0x4000 > Normalized Value <= 0x7fff i.e. (MAXFRACT16+1)/2 > aNorm <= MAXFRACT16 If a<0: 0x8000 >= Normalilzed Value < 0xC000 i.e. MINFRACT16>= aNorm < MINFRACT16/2

This function relates to the ETSI norm_s function.

Parameters

Parameters	Description
q15	a in Q1.15

Fx32Norm Function

Normalize the 32-bit number.

File

libq_C.h

C

```
int16_t Fx32Norm(q31);
```

Returns

int16_t result - The number of left shifts required to normalize the

Description

Function Fx32Norm:

Produces then number of left shifts needed to Normalize the 32-bit fractional input. If the input 'a' is a positive number, it will produce the number of left shifts required to normalized it to the range of a minimum of [(MAXFRACT32+1)/2] to a maximum of [MAXFRACT32]. If the input 'a' is a negative number, it will produce the number of left shifts required to normalized it to the range of a minimum of [MINFRACT32] to a maximum of [MINFRACT32/2]. This function does not actually normalize the input, it just produces the number of left shifts required. To actually normalize the value the left-shift function should be used with the value returned From this function.

32-bit input on range: 0 => result < 32 (i.e. NUMBITSFRACT32) If a>0: 0x40000000 > Normalized Value <= 0x7fffffff i.e. (MAXFRACT32+1)/2 > aNorm <= MAXFRACT32 If a<0: 0x80000000 >= Normalized Value < 0xC0000000 i.e. MINFRACT32>= aNorm < MINFRACT32/2

This function relates to the ETSI norm_l function.

Parameters

Parameters	Description
q31 a	32-bit Q1.d31 to be normalized

Data Types and Constants

q15 Type

File

libq_C.h

C

```
typedef int16_t q15;
```

Description

q15 n.n (signed)

q31 Type

File

libq_C.h

C

```
typedef int32_t q31;
```

Description

q31 n.n (signed)

q63 Type

File

libq_C.h

C

```
typedef int64_t q63;
```

Description

Q63 n.n (signed)

_LIBQ_C_H_ Macro

File

libq_C.h

C

```
#define _LIBQ_C_H_
```

Description

This is macro _LIBQ_C_H_.

BITMASKFRACT16 Macro

File

libq_C.h

C

```
#define BITMASKFRACT16 (~((uint16_t)0x0))      /* Bit Mask for 16 */
```

Description

Bit Mask for 16

BITMASKFRACT32 Macro

File

libq_C.h

C

```
#define BITMASKFRACT32 (~((uint32_t)0x0L))     /* Bit Mask for 32 */
```

Description

Bit Mask for 32

Fl2Fract16 Macro

Converts floating point constant value to fractional 16-bit value

File

libq_C.h

C

```
#define Fl2Fract16(value) \
( \
  (q15)Fl2FxPnt((value), 15) \
)
```

Returns

q15 - Equivalent fractional value

Description

Function Fl2Fract16:

Converts floating point constant value to fractional 16-bit value with rounding.

Parameters

Parameters	Description
value	Floating point value constant expression to convert.

Fl2Fract32 Macro

Converts floating point constant value to fractional 32-bit value

File

libq_C.h

C

```
#define Fl2Fract32(value) \
( \
  (q31)Fl2FxPnt((value), 31) \
)
```

Returns

Equivalent fractional value as AtiFract32.

Description

Function Fl2Fract32:

Converts floating point constant value to fractional 32-bit value with rounding.

Parameters

Parameters	Description
value	Floating point value constant expression to convert.

Fl2FxPnt Macro

Converts floating point constant value to fixed/fractional value

File

libq_C.h

C

```
#define Fl2FxPnt(value, bits) \
( \
  ((double)(value)*(double)(1UL<<(bits))) + \
    (((double)((double)(value) >= 0.0)) - ((double)0.5)) \
)
```

Returns

double - Equivalent floating point value

Description

Function Fl2FxPnt:

Converts floating point constant value to fixed/fractional value of specified precision with rounding.

Parameters

Parameters	Description
value	Floating point value constant expression to convert.
int_t bits	Number of fractional bits.

Fl2FxPnt16 Macro

Converts floating point constant value to fixed point 16-bit value

File

libq_C.h

C

```
#define Fl2FxPnt16(value, bits) \
( \
  (q15)Fl2FxPnt((value), (bits)) \
)
```

Returns

q15 - Equivalent fixed point value

Description

Function Fl2FxPnt16:

Converts floating point constant value to fixed point 16-bit value with rounding.

Parameters

Parameters	Description
value	Floating point value constant expression to convert. int_t bits -- Number of fractional bits.

Fl2FxPnt32 Macro

Converts floating point constant value to fixed point 32-bit value

File

libq_C.h

C

```
#define Fl2FxPnt32(value, bits) \
( \
  (q31)Fl2FxPnt((value), (bits)) \
)
```

Returns

q31 - Equivalent fixed point value

Description

Function Fl2FxPnt32:

Converts floating point constant value to fixed point 32-bit value with rounding.

Parameters

Parameters	Description
value	Floating point value constant expression to convert.
int_t bits	Number of fractional bits.

Fl2Int16 Macro

Converts floating point constant expression to 16-bit integer value

File

libq_C.h

C

```
#define Fl2Int16(value) ((int16_t)((double)(value)))
```

Returns

int_t - Calculated value.

Description

Function Fl2Int16:

Converts floating point constant expression to 16-bit integer value Conversion is safe for compilers which assign fractional data type to constants with decimal point

Parameters

Parameters	Description
value	Floating point value constant expression to convert.

Fl2Int32 Macro

Converts floating point constant expression to 32-bit integer value

File

libq_C.h

C

```
#define Fl2Int32(value) ((int32_t)((double)(value)))
```

Returns

int_t - Calculated integer value.

Description

Function Fl2Int32:

Converts floating point constant expression to 32-bit integer value Conversion is safe for compilers which assign fractional data type to constants with decimal point

Parameters

Parameters	Description
value	Floating point value constant expression to convert.

FrMax Macro

File

libq_C.h

C

```
#define FrMax(a,b) ((a)>(b)?(a):(b))
```

Description

find the maximum of two numbers

FrMin Macro

File

libq_C.h

C

```
#define FrMin(a,b) ((a)<(b)?(a):(b))
```

Description

find the minimum of two numbers

LOG102Q5D11 Macro

File

libq_C.h

C

```
#define LOG102Q5D11 Fl2FxPnt16(0.301029996,11)
```

Description

log10(2) scaled to Q5.11 format

MAXFRACT16 Macro

File

libq_C.h

C

```
#define MAXFRACT16 MAXINT16       /* +0.999969 */
```

Description

0.999969

MAXFRACT32 Macro

File

libq_C.h

C

```
#define MAXFRACT32 MAXINT32       /* +0.9999999995 */
```

Description

0.9999999995

MAXINT16 Macro

File

libq_C.h

C

```
#define MAXINT16 ((int16_t) 0x7fff)
```

Description

Maximum and minimum values for 16-bit data types.

MAXINT32 Macro

File

libq_C.h

C

```
#define MAXINT32 ((int32_t)0x7fffffffL)
```

Description

This is macro MAXINT32.

MAXPFLOAT32 Macro

File

libq_C.h

C

```
#define MAXPFLOAT32 {MAXFRACT16, MAXINT16}
```

Description

minimum and maximum definitions for FX floating point data types

MINFRACT16 Macro

File

libq_C.h

C

```
#define MINFRACT16 MININT16       /* -1.000000 */
```

Description

1.000000

MINFRACT32 Macro

File

libq_C.h

C

```
#define MINFRACT32 MININT32       /* -1.0000000000 */
```

Description

1.0000000000

MININT16 Macro

File

libq_C.h

C

```
#define MININT16 ((int16_t) (-MAXINT16 - 1))
```

Description

This is macro MININT16.

MININT32 Macro

File

libq_C.h

C

```
#define MININT32 ((int32_t)(-MAXINT32 - 1))
```

Description

This is macro MININT32.

MINPFLOAT32 Macro

File

libq_C.h

C

```
#define MINPFLOAT32 {MINFRACT16, MAXINT16}
```

Description

This is macro MINPFLOAT32.

MSBBITFRACT16 Macro

File

libq_C.h

C

```
#define MSBBITFRACT16 MININT16                /* 16-bit Sign Bit */
```

Description

16-bit Sign Bit

MSBBITFRACT32 Macro

File

libq_C.h

C

```
#define MSBBITFRACT32 MININT32                /* 32-bit Sign Bit */
```

Description

32-bit Sign Bit

NINETYQ10D22 Macro

File

libq_C.h

C

```
#define NINETYQ10D22 Fl2FxPnt32(90,22)  /* Ninety degrees scaled to Q10d22*/
```

Description

Ninety degrees scaled to Q10d22

NINETYQ10D6 Macro

File

libq_C.h

C

```
#define NINETYQ10D6 Fl2FxPnt16(90,6)   /* Ninety degrees scaled to Q10d6 */
```

Description

Ninety degrees scaled to Q10d6

NORMNEGFRACT16 Macro

File

libq_C.h

C

```
#define NORMNEGFRACT16 ((q15)0xc000)           /* Max -val for 16 */
```

Description

Max -val for 16

NORMNEGFRACT32 Macro

File

libq_C.h

C

```
#define NORMNEGFRACT32 ((q31)0xc0000000L)      /* Max -val for 32 */
```

Description

Max -val for 32

NORMPOSFRACT16 Macro

File

libq_C.h

C

```
#define NORMPOSFRACT16 ((q15)0x4000)           /* Min +val for 16 */
```

Description

Min +val for 16

NORMPOSFRACT32 Macro

File

libq_C.h

C

```
#define NORMPOSFRACT32 ((q31)0x40000000L)      /* Min +val for 32 */
```

Description

Min +val for 32

NUMBITSFRACT16 Macro

File

libq_C.h

C

```
#define NUMBITSFRACT16 ((int16_t)0x010)        /* Num of bits 16  */
```

Description

Num of bits 16

NUMBITSFRACT32 Macro

File

libq_C.h

C

```
#define NUMBITSFRACT32 ((int16_t)0x020)        /* Num of bits 32  */
```

Description

Num of bits 32

ONEEIGHTYQ10D22 Macro

File

libq_C.h

C

```
#define ONEEIGHTYQ10D22 Fl2FxPnt32(180,22) /* 180 degrees scaled to Q10d22   */
```

Description

180 degrees scaled to Q10d22

ONEEIGHTYQ10D6 Macro

File

libq_C.h

C

```
#define ONEEIGHTYQ10D6 Fl2FxPnt16(180,6)  /* 180 degrees scaled to Q10d6    */
```

Description

180 degrees scaled to Q10d6

ROUNDFRACT32 Macro

File

libq_C.h

C

```
#define ROUNDFRACT32 ((q31)0x00008000L)      /* Rounding value  */
```

Description

Rounding value

THREESIXTYQ10D22 Macro

File

libq_C.h

C

```
#define THREESIXTYQ10D22 Fl2FxPnt32(360,22) /* 360 degrees scaled to Q10d22   */
```

Description

360 degrees scaled to Q10d22

THREESIXTYQ10D6 Macro

File

libq_C.h

C

```
#define THREESIXTYQ10D6 Fl2FxPnt16(360,6)  /* 360 degrees scaled to Q10d6    */
```

Description

360 degrees scaled to Q10d6

TWOSEVENTYQ10D22 Macro

File

libq_C.h

C

```
#define TWOSEVENTYQ10D22 Fl2FxPnt32(270,22) /* 270 degrees scaled to Q10d22   */
```

Description

270 degrees scaled to Q10d22

TWOSEVENTYQ10D6 Macro

File

libq_C.h

C

```
#define TWOSEVENTYQ10D6 Fl2FxPnt16(270,6)  /* 270 degrees scaled to Q10d6    */
```

Description

270 degrees scaled to Q10d6

UNITYFLOAT Macro

File

libq_C.h

C

```
#define UNITYFLOAT 1.0
```

Description

This is macro UNITYFLOAT.

FxQFloat32 Variable

File

libq_C.h

C

```
struct {
  q15 man;
  i16 exp;
} FxQFloat32;
```

Description

FxQFloat32 pseudo floating point type (limited floating point)

Remarks

Extended FxQflExt32 used with f2Qfloat32)

Exponent16ToQFloat32 Macro

Converts a power of 2 16-bit integer to 32-bit floating point value.

File

libq_C.h

C

```
#define Exponent16ToQFloat32(value) \
```

```
( \
  (((value) < 0) ? ((float)1.0/(float)(1UL<<(-(value)))): \
   ((float)(1UL<<(value)))) \
)
```

Returns

32-bit floating point value of 2^n.

Description

Function Exponent16ToQFloat32

Converts a power of 2 16-bit integer to 32-bit floating point value.

Parameters

Parameters	Description
int16_t	16-bit integer number for power (expected to be a variable).

Fl2QFloat32 Macro

Converts a decimal floating-point constant expression to pseudo float

File

libq_C.h

C

```
#define Fl2QFloat32(mantissa, exponent) \
  { \
    Fl2Fract16(mantissa), \
    Fl2Int16(exponent) \
  }
```

Returns

Calculated integer value.

Description

Function Fl2QFloat32:

Converts a decimal floating-point constant expression to pseudo float

Parameters

Parameters	Description
mantissa	Floating-point value constant expression to convert to the mantissa portion of the floating-point number.
exponent	Integer value constant expression to convert to the exponent portion of the floating-point number.

i16 Type

File

libq_C.h

C

```
typedef int16_t i16;
```

Description

Q16d0

Files

Files

Name	Description
libq_C.h	C-code fixed point math functions.

Description

This section lists the source and header files used by the LibQ Fixed-Point 'C' Math Library.

libq_C.h

C-code fixed point math functions.

Functions

	Name	Description
	Fx16Norm	Normalize the 16-bit fractional value.
	Fx32Norm	Normalize the 32-bit number.
	libq_q15_Abs_q15	Saturated Absolute value.
	libq_q15_Add_q15_q15	Add two 16-bit 2s-complement fractional values.
	libq_q15_DivisionWithSaturation_q15_q15	Fractional division with saturation.
	libq_q15_ExpAvg_q15_q15_q1d15	Exponential averaging
	libq_q15_ExtractH_q31	Extracts upper 16 bits of input 32-bit fractional value.
	libq_q15_ExtractL_q31	Extracts lower 16-bits of input 32-bit fractional value. Descriptionf Extracts lower 16-bits of input 32-bit fractional value and returns them as 16-bit fractional value. This is a bit-for-bit extraction of the bottom 16-bits of the 32-bit input. This function relates to the ETSI extract_l function.
	libq_q15_Negate_q15	Negate 16-bit 2s-complement fractional value with saturation.
	libq_q15_RoundL_q31	Rounds the lower 16-bits of the 32-bit fractional input.
	libq_q15_ShiftLeft_q15_q15	'Arithmetic' Shift of the 16-bit input argument.
	libq_q15_ShiftRight_q15_q15	'Arithmetic' RIGHT Shift on a 16-bit value.
	libq_q15_ShiftRightRound_q15_q15	Performs an 'Arithmetic' RIGHT Shift on a 16-bit input.
	libq_q15_Sub_q15_q15	Subtract two 16-bit 2s-complement fractional values
	libq_q1d15_MacR_q1d31_q1d15_q1d15	Multiply accumulate with rounding.
	libq_q1d15_MsuR_q1d31_q1d15_q1d15	Multiply-Subtraction with rounding
	libq_q1d15_MultipyR2_q1d15_q1d15	fractional multiplication of two 16-bit fractional values giving a 16 bit rounded result.
	libq_q1d15_Sin_q10d6	Approximates the sine of an angle.
	libq_q1d31_Mac_q1d31_q1d15_q1d15	Multiply-Accumulate function WITH saturation
	libq_q1d31_Msu_q1d31_q1d15_q1d15	L_msu(a,b,c)
	libq_q20d12_Sin_q20d12	3rd order Polynomial apprx. of a sine function
	libq_q31_Abs_q31	Saturated Absolute value.
	libq_q31_Add_q31_q31	Add two 32-bit 2s-complement fractional values.
	libq_q31_DepositH_q15	Place 16 bits in the upper half of 32 bit word.
	libq_q31_DepositL_q15	Place 16 bits in the lower half of 32 bit word.
	libq_q31_Mult2_q15	fractional multiplication of two 16-bit fractional values.
	libq_q31_Multi_q15_q31	Implement 16 bit by 32 bit multiply.
	libq_q31_Negate_q31	Negate 32-bit 2s-complement fractional value with saturation.
	libq_q31_ShiftLeft_q31_i16	'Arithmetic' Shift of the 32-bit value.
	libq_q31_ShiftRight_q31_q15	'Arithmetic' RIGHT Shift on a 32-bit value.
	libq_q31_ShiftRightRound_q31_q15	'Arithmetic' RIGHT Shift on a 32-bit value

Macros

	Name	Description
	_LIBQ_C_H_	This is macro _LIBQ_C_H_.
	BITMASKFRACT16	Bit Mask for 16
	BITMASKFRACT32	Bit Mask for 32
	Exponent16ToQFloat32	Converts a power of 2 16-bit integer to 32-bit floating point value.
	Fl2Fract16	Converts floating point constant value to fractional 16-bit value
	Fl2Fract32	Converts floating point constant value to fractional 32-bit value
	Fl2FxPnt	Converts floating point constant value to fixed/fractional value
	Fl2FxPnt16	Converts floating point constant value to fixed point 16-bit value
	Fl2FxPnt32	Converts floating point constant value to fixed point 32-bit value
	Fl2Int16	Converts floating point constant expression to 16-bit integer value

	Fl2Int32	Converts floating point constant expression to 32-bit integer value
	Fl2QFloat32	Converts a decimal floating-point constant expression to pseudo float
	FrMax	find the maximum of two numbers
	FrMin	find the minimum of two numbers
	LOG102Q5D11	log10(2) scaled to Q5.11 format
	MAXFRACT16	0.999969
	MAXFRACT32	0.9999999995
	MAXINT16	Maximum and minimum values for 16-bit data types.
	MAXINT32	This is macro MAXINT32.
	MAXPFLOAT32	minimum and maximum definitions for FX floating point data types
	MINFRACT16	1.000000
	MINFRACT32	1.0000000000
	MININT16	This is macro MININT16.
	MININT32	This is macro MININT32.
	MINPFLOAT32	This is macro MINPFLOAT32.
	MSBBITFRACT16	16-bit Sign Bit
	MSBBITFRACT32	32-bit Sign Bit
	NINETYQ10D22	Ninety degrees scaled to Q10d22
	NINETYQ10D6	Ninety degrees scaled to Q10d6
	NORMNEGFRACT16	Max -val for 16
	NORMNEGFRACT32	Max -val for 32
	NORMPOSFRACT16	Min +val for 16
	NORMPOSFRACT32	Min +val for 32
	NUMBITSFRACT16	Num of bits 16
	NUMBITSFRACT32	Num of bits 32
	ONEEIGHTYQ10D22	180 degrees scaled to Q10d22
	ONEEIGHTYQ10D6	180 degrees scaled to Q10d6
	ROUNDFRACT32	Rounding value
	THREESIXTYQ10D22	360 degrees scaled to Q10d22
	THREESIXTYQ10D6	360 degrees scaled to Q10d6
	TWOSEVENTYQ10D22	270 degrees scaled to Q10d22
	TWOSEVENTYQ10D6	270 degrees scaled to Q10d6
	UNITYFLOAT	This is macro UNITYFLOAT.

Types

	Name	Description
	i16	Q16d0
	q15	q15 n.n (signed)
	q31	q31 n.n (signed)
	q63	Q63 n.n (signed)

Variables

	Name	Description
	FxQFloat32	FxQFloat32 pseudo floating point type (limited floating point)

Description

The libq_c Fixed-Point Math Library provides fixed-point math functions written in C for portability between core processors.

Signed fixed point types (fractional Q types specified by Qn.m) are named as follows in the library names:

Qndm where:

- n is the number of data bits to the left of the radix point
- m is the number of data bits to the right of the radix point
- a signed bit is implied (unless stated otherwise)

For convenience, short names are also defined for arbitrary scaled fractional types:

q15 is signed fractional 16 bit value q31 is signed fractional 32 bit value i16 is signed integer, i.e. Q16d0

In addition, A pseudo floating point 32 bit format (FxQFloat32) is defined that consists of 16 mantissa and a 16 bit exponant (base 2).

Functions in the library are prefixed with the type of the return value and followed by argument types (in order):

libq___: libq_q15_sin_Q2d13

For example, libq_q1d15_Sin_q10d6 returns a Q1.15 value equal to the to the sine of an angle specified as a Q10.6 value (in degrees between 0 and 360)

Argument types do not always match the return type. Refer to the function prototype for a specification of its arguments.

In some cases, both the return type and the argument type are specified within the function name. For example,

For arbitrary scaled types (q15, q16, q31, and q32) the scaling of the result will depend on the function and the scaling of the arguments. For instance, libq_q15_Add_q15_q15(a,b) will return a scaled value type that is the two input types (which must have equivalent scaled value type).

Remarks

The libq_c functions do not correspond to the libq fixed-point library optimized for the microaptive core processor and written in asm.

Table of LIBQ_C math functions:

Sine: libq_q1d15_Sin_q10d6 libq_q20d12_Sin_q20d12

Abs: libq_q15_Abs_q15 libq_q31_Abs_q31

Negate: libq_q15_Negate_q15 libq_q31_Negate_q31

Round: libq_q15_RoundL_q31

Deposit: libq_q31_DepositH_q15 libq_q31_DepositL_q15

Extract: libq_q15_ExtractH_q31 libq_q15_ExtractL_q31

Add: libq_q15_Add_q15_q15 libq_q31_Add_q31_q31

Subtract: libq_q15_Sub_q15_q15

Shift(Scale): libq_q15_ShiftLeft_q15_q15 libq_q31_ShiftLeft_q31_q15 libq_q15_ShiftRight_q15_q15 libq_q31_ShiftRight_q31_q15 libq_q15_ShiftRightRound_q15_q15 libq_q31_ShiftRightRound_q31_q15

Multiply: libq_q15_Mult_q15_q15) libq_q15_MultipyR2_q15 libq_q31_Multi_q15_q31

Divide: libq_q15_DivisionWithSaturation_q15_q15

Multiply-Accumulate: libq_q31_Mac_q31_q15_q15 libq_q15_MacR_q31_q15_q15

Multiply-Subtract: libq_q31_Msu_q31_q15_q15 libq_q15_MsuR_q31_q15_q15

Exponential-Averaging: libq_q15_ExpAvg_q15_q15_q1d15

Table of LIBQ_C conversion functions:

Normalize Q value: Fx16Norm Fx32Norm

Float-to-Q value: Fl2Fract16 Fl2Fract32 Fl2FxPnt16 Fl2FxPnt32 Fl2FxPnt

Float-To-Integer: Fl2Int16 Fl2Int32

Float-To-FxQFloat32: Fl2QFloat32

Exponent-To-Float: Exponent16ToFloat32

File Name

libq_c.h

Company

Microchip Technology Inc.

LibQ Fixed-Point Math Library

This topic describes the LibQ Fixed-Point Math Library.

Introduction

The LibQ Fixed-Point Math Library is available for the PIC32MZ family of microcontrollers. This library was created from optimized assembly routines written specifically for devices with microAptiv™ core features.

Description

The LibQ Fixed-Point Math Library simplifies writing fixed point algorithms, supporting Q15, Q31 and other 16-bit and 32-bit data formats. Using the simple, C callable functions contained in the library, fast fixed point mathematical operations can be easily executed. Fixed-point mathematical calculations may replace some functions implemented in the floating point library (`math.h`), depending on performance and resolution requirements.

Functions included in the LibQ library include capabilities for trigonometric, power and logarithms, and data conversion. In many cases the functions are identical other than the precision of their operands and the corresponding value that they return.

These functions are implemented in efficient assembly, and generally tuned to optimize performance over code size. In some cases the library breaks out functions that enable one to be optimized for accuracy, while another version is optimized for speed. These functions such as _LIBQ_Q2_29_acos_Q31 and _LIBQ_Q2_29_acos_Q31_Fast are otherwise identical and can be used interchangeably. Each of these functions are typically used in computationally intensive real-time applications where execution time is a critical parameter.

Using the Library

This topic describes the basic architecture of the LibQ Fixed-Point Math Library and provides information and examples on its use.

Description

Interface Header File: `libq.h`

The interface to the LibQ Fixed-Point Math Library is defined in the `libq.h` header file. Any C language source (`.c`) file that uses the LibQ Fixed-Point Library should include `libq.h`.

Library File:

The LibQ Fixed-Point Math Library archive (`.a`) file is installed with MPLAB Harmony.

Library Overview

The LibQ Fixed-Point Math Library contains functions for manipulating Q15, Q31 and other intermediate integer representations of real numbers. The Library Interface section details the operation of the data formats and explains each function in detail.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DSP Fixed-Point Math Library.

Library Interface Section	Description
Divide Functions	_Q16 fixed point divide function.
Square Root Functions	Square root of a positive _Q16 fixed point value function.
Log Functions	Log calculation functions.
Power Functions	Power calculation functions.
Exponential Functions	Exponential calculation functions.
Sine Functions	Sine calculation functions.
Cosine Functions	Cosine calculation functions.
Target Functions	Target calculation functions.
Arcsin Functions	Arcsin calculation functions.
Arccos Functions	Arccos calculation functions.
Arctan2 Functions	Arctan2 calculation functions.
Random Number Functions	_Q15 and _Q31 pseudo-random value functions.
Float Functions	Float conversion functions.
String Functions	ASCII to _Q15 conversions.

Signed fixed-point types are defined as follows:

Q*n*_*m* where:

- *n* is the number of data bits to the left of the radix point
- *m* is the number of data bits to the right of the radix point
- a signed bit is implied

For convenience, short names are also defined:

Exact Name	Number of Bits Required	Short Name
_Q0_15	16	_Q15
_Q15_16	32	_Q16
_Q0_31	32	_Q31

Qn_m numerical values are used by the library processing data as integers. In this format the n represents the number of integer bits, and the m represents the number of fractional bits. All values assume a sign bit in the most significant bit. The range of the numerical value therefore is:

$$-2^{(n-1)} \text{ to } [2^{(n-1)} - 2^{(-m)}]; \text{ with a resolution of } 2^{(-m)}.$$

A _Q16 format number (_Q15_16) would range from -32768.0 (0x8000 0000) to 32767.99998474 with a precision of 0.000015259 (or 2^{-16}).

For example, a numerical representation of the number 3.14159 in _Q2_13 notation would be:

$$3.14159 * 2^{13} = 25735.9 \Rightarrow 0x6488$$

And converting from the _Q7_8 format with the value 0x1D89 would be:

$$0x1D89 / 2^{8} = 7561 / 256 \Rightarrow 29.5316, \text{ accurate to } 0.00391$$

Functions in the library are prefixed with the type of the return value. For example, _LIBQ_Q16Sqrt returns a _Q16 value equal to the square root of its argument. Argument types do not always match the return type. Refer to the function prototype for a specification of its arguments.

In cases where the return value is not a fixed-point type, the argument type is appended to the function name. For example, _LIBQ_ToFloatQ31 accepts a type _Q31 argument.

In some cases, both the return type and the argument type are specified within the function name. For example:

Function Name	Return Type	Argument Type
_LIBQ_Q15_sin_Q2_13	_Q15	_Q2_13
_LIBQ_Q31_sin_Q2_29	_Q31	_Q2_29

Table of Library Functions

Math Function	Function Definition
Divide	_Q16 _LIBQ_Q16Div (_Q16 dividend, _Q16 divisor);
Square Root	_Q16 _LIBQ_Q16Sqrt (_Q16 x);
Exponential	_Q16 _LIBQ_Q16Exp (_Q16 x);
Log	_Q4_11 _LIBQ_Q4_11_ln_Q16 (_Q16 x); _Q3_12 _LIBQ_Q3_12_log10_Q16 (_Q16 x); _Q5_10 _LIBQ_Q5_10_log2_Q16 (_Q16 x);
Power	_Q16 _LIBQ_Q16Power (_Q16 x, _Q16 y);
Sine	_Q15 _LIBQ_Q15_sin_Q2_13 (_Q2_13 x); _Q31 _LIBQ_Q31_sin_Q2_29 (_Q2_29 x);
Cosine	_Q15 _LIBQ_Q15_cos_Q2_13 (_Q2_13 x); _Q31 _LIBQ_Q31_cos_Q2_29 (_Q2_29 x);
Tangent	_Q7_8 _LIBQ_Q7_8_tan_Q2_13 (_Q2_13 x); _Q16 _LIBQ_Q16_tan_Q2_29 (_Q2_29 x);
Arcsin	_Q2_13 _LIBQ_Q2_13_asin_Q15 (_Q15 x); _Q2_29 _LIBQ_Q2_29_asin_Q31 (_Q31 x); _Q2_29 _LIBQ_Q2_29_asin_Q31_Fast (_Q31 x);
Arccos	_Q2_13 _LIBQ_Q2_13_acos_Q15 (_Q15 x); _Q2_29 _LIBQ_Q2_29_acos_Q31 (_Q31 x); _Q2_29 _LIBQ_Q2_29_acos_Q31_Fast (_Q31 x);
Arctan	_Q2_13 _LIBQ_Q2_13_atan_Q7_8 (_Q7_8 x); _Q2_29 _LIBQ_Q2_29_atan_Q16 (_Q16 x);
Arctan2	_Q2_13 _LIBQ_Q2_13_atan2_Q7_8 (_Q7_8 y, _Q7_8 x); _Q2_29 _LIBQ_Q2_29_atan2_Q16 (_Q16 y, _Q16 x);
Random Number	_Q15 _LIBQ_Q15Rand (int64_t *pSeed); _Q31 _LIBQ_Q31Rand (int64_t *pSeed);
Float	float _LIBQ_ToFloatQ31 (_Q31 x); float _LIBQ_ToFloatQ15 (_Q15 x); _Q31 _LIBQ_Q31FromFloat (float x); _Q15 _LIBQ_Q15FromFloat (float x);
String	void _LIBQ_ToStringQ15 (_Q15 x, char *s); _Q15 _LIBQ_Q15FromString (char *s);

Library Interface

a) Divide Functions

	Name	Description
	_LIBQ_Q16Div	_Q16 fixed point divide.

b) Square Root Functions

	Name	Description
	_LIBQ_Q16Sqrt	Square root of a positive _Q16 fixed point value.

c) Log Functions

	Name	Description
	_LIBQ_Q3_12_log10_Q16	Calculates the value of Log10(x).
	_LIBQ_Q4_11_ln_Q16	Calculates the natural logarithm ln(x).
	_LIBQ_Q5_10_log2_Q16	Calculates the value of log2(x).

d) Power Functions

	Name	Description
	_LIBQ_Q16Power	Calculates the value of x raised to the y power (x^y).

e) Exponential Functions

	Name	Description
	_LIBQ_Q16Exp	Calculates the exponential function e^x.

f) Sine Functions

	Name	Description
	_LIBQ_Q15_sin_Q2_13	Calculates the value of sine(x).
	_LIBQ_Q31_sin_Q2_29	Calculates the value of sine(x).

g) Cosine Functions

	Name	Description
	_LIBQ_Q15_cos_Q2_13	Calculates the value of cosine(x).
	_LIBQ_Q31_cos_Q2_29	Calculates the value of cosine(x).

h) Target Functions

	Name	Description
	_LIBQ_Q16_tan_Q2_29	Calculates the value of tan(x).
	_LIBQ_Q7_8_tan_Q2_13	Calculates the value of tan(x).

i) Arcsin Functions

	Name	Description
	_LIBQ_Q2_13_asin_Q15	Calculates the asin value of asin(x).
	_LIBQ_Q2_29_asin_Q31	Calculates the value of asin(x).
	_LIBQ_Q2_29_asin_Q31_Fast	Calculates the value of asin(x). This function executes faster than the _LIBQ_Q2_29_asin_Q31 function, but is less precise.

j) Arccos Functions

	Name	Description
	_LIBQ_Q2_13_acos_Q15	Calculates the value of acos(x).
	_LIBQ_Q2_29_acos_Q31	Calculates the value of acos(x).
	_LIBQ_Q2_29_acos_Q31_Fast	Calculates the value of acos(x). This function executes faster than _LIBQ_Q2_29_acos_Q31 but is less precise.

k) Arctan Functions

	Name	Description
	_LIBQ_Q2_13_atan_Q7_8	Calculates the value of atan(x).
	_LIBQ_Q2_29_atan_Q16	Calculates the value of atan(x).

l) Arctan2 Functions

	Name	Description
	_LIBQ_Q2_13_atan2_Q7_8	Calculates the value of atan2(y, x).
	_LIBQ_Q2_29_atan2_Q16	Calculates the value of atan2(y, x).

m) Random Number Functions

	Name	Description
	_LIBQ_Q15Rand	Generate a _Q15 random number.
	_LIBQ_Q31Rand	Generate a _Q31 random number.

n) Float Functions

	Name	Description
	_LIBQ_Q15FromFloat	Converts a float to a _Q15 value.
	_LIBQ_Q31FromFloat	Converts a float to a _Q31 value.
	_LIBQ_ToFloatQ15	Converts a _Q15 value to a float.
	_LIBQ_ToFloatQ31	Converts a _Q31 value to a float.

o) String Functions

	Name	Description
	_LIBQ_Q15FromString	ASCII to _Q15 conversion.
	_LIBQ_ToStringQ15	_Q15 to ASCII conversion.

p) Data Types and Constants

	Name	Description
	_Q15_MAX	Maximum value of _Q15 (~1.0)
	_Q15_MIN	Minimum value of _Q15 (-1.0)
	_Q16_MAX	Maximum value of _Q16 (~32768.0)
	_Q16_MIN	Minimum value of _Q16 (-32768.0)
	_Q2_13_MAX	Maximum value of _Q2_13 (~4.0)
	_Q2_13_MIN	Minimum value of _Q2_13 (-4.0)
	_Q2_29_MAX	Maximum value of _Q2_29 (~4.0)
	_Q2_29_MIN	Minimum value of _Q2_29 (-4.0)
	_Q3_12_MAX	Maximum value of _Q3_12 (~8.0)
	_Q3_12_MIN	Minimum value of _Q3_12 (-8.0)
	_Q31_MAX	Maximum value of _Q31 (~1.0)
	_Q31_MIN	Minimum value of _Q31 (-1.0)
	_Q4_11_MAX	Maximum value of _Q4_11 (~16.0)
	_Q4_11_MIN	Minimum value of _Q4_11 (-16.0)
	_Q5_10_MAX	Maximum value of _Q5_10 (~32.0)
	_Q5_10_MIN	Minimum value of _Q5_10 (-32.0)
	_Q7_8_MAX	Maximum value of _Q7_8 (~128.0)
	_Q7_8_MIN	Minimum value of _Q7_8 (-128.0)
	_Q0_15	1 sign bit, 15 bits right of radix
	_Q0_31	1 sign bit, 31 bits right of radix
	_Q15	Short name for _Q0_15
	_Q15_16	1 sign bit, 15 bits left of radix, 16 bits right of radix
	_Q16	Short name for _Q15_16
	_Q2_13	1 sign bit, 2 bits left of radix, 13 bits right of radix
	_Q2_29	1 sign bit, 2 bits left of radix, 29 bits right of radix
	_Q3_12	1 sign bit, 3 bits left of radix, 12 bits right of radix
	_Q31	Short name for _Q_0_31
	_Q4_11	1 sign bit, 4 bits left of radix, 11 bits right of radix
	_Q5_10	1 sign bit, 5 bits left of radix, 10 bits right of radix
	_Q7_8	1 sign bit, 7 bits left of radix, 8 bits right of radix
	_LIBQ_H	Guards against multiple inclusion

Description

This section describes the Application Programming Interface (API) functions, macros, and types of the LibQ Fixed Point Math Library.

Refer to each section for a detailed description.

a) Divide Functions

_LIBQ_Q16Div Function

_Q16 fixed point divide.

File

libq.h

C

```
_Q16 _LIBQ_Q16Div(_Q16 dividend, _Q16 divisor);
```

Returns

_Q16 quotient of the divide operation

Description

Function _LIBQ_Q16Div:

_Q16 _LIBQ_Q16Div (_Q16 dividend, _Q16 divisor);

Quotient (_Q16) = Dividend (_Q16) / Divisor (_Q16).

Remarks

The _LIBQ_Q16Div operation saturates its result.

Execution Time (cycles): 143 typical (80 to 244) Program Memory 204 bytes

Error <= 0.000015258789 (accurate to least significant _Q16 bit within the non-saturated range)

Preconditions

Divisor must not equal 0.

Example

```
_Q16  quotient, dividend, divisor;

dividend = (_Q16)0x00010000;  // 1
divisor  = (_Q16)0x00008000;  // 0.5

quotient = _LIBQ_Q16Div (dividend, divisor);

// quotient now equals 2;  i.e.,  (_Q16)0x00020000;
```

Parameters

Parameters	Description
dividend	The divide operation dividend (_Q16)
divisor	The divide operation divisor (_Q16)

b) Square Root Functions

_LIBQ_Q16Sqrt Function

Square root of a positive _Q16 fixed point value.

File

libq.h

C

```
_Q16 _LIBQ_Q16Sqrt(_Q16 x);
```

Returns

_LIBQ_Q16Sqrt returns the _Q16 fixed point value which is the square root of the input parameter.

Description

Function _LIBQ_Q16Sqrt:

_Q16 _LIBQ_Q16Sqrt(_Q16 x);

Calculate the square root of a positive _Q16 fixed point value, and return the _Q16 result.

Remarks

Execution Time (cycles): 240 typical (104 to 258) Program Memory 152 bytes

Error <= 0.000015258789 (accurate to least significant _Q16 bit)

Preconditions

The input value must be positive.

Example

```
_Q16 squareRoot;

squareRoot = _LIBQ_Q16Sqrt((_Q16)0x01000000);  // The square root of 256.0 is 16.0 (0x00100000)

squareRoot = _LIBQ_Q16Sqrt((_Q16)0x00004000);  // The square root of 0.25 is 0.5 (0x00008000)

squareRoot = _LIBQ_Q16Sqrt((_Q16)0x5851f42d);  // The square root of 22609.953125 is 150.366074 (0x00965db7)
```

Parameters

Parameters	Description
x	The _Q16 fixed point value input from which to find the square root.

c) Log Functions

_LIBQ_Q3_12_log10_Q16 Function

Calculates the value of Log10(x).

File

libq.h

C

```
_Q3_12 _LIBQ_Q3_12_log10_Q16(_Q16 x);
```

Returns

_LIBQ_Q3_12_log10_Q16 returns the _Q3_12 fixed point result from the calculation log10(x).

Description

Function _LIBQ_Q3_12_log10_Q16:

_Q3_12 _LIBQ_Q3_12_log10_Q16 (_Q16 x);

Calculates the log10(x), where log10(x) = ln(x) * log10(e). x is of type _Q16 and must be positive. The resulting value is of type _Q3_12.

Remarks

Execution Time (cycles): 301 typical (14 to 346) Program Memory 176 bytes

Error <= 0.000244140625 (accurate to least significant _Q3_12 bit)

Preconditions

The input x must be positive.

Example

```
_Q3_12 resultLog10;

resultLog10 = _LIBQ_Q3_12_log10_Q16 ((_Q16)0x12ed7d91);  // _LIBQ_Q3_12_log10_Q16(4845.490494) = 3.685303
(0x3af7)
```

Parameters

Parameters	Description
x	The input value from which to calculate log10(x).

_LIBQ_Q4_11_ln_Q16 Function

Calculates the natural logarithm ln(x).

File

libq.h

C

```
_Q4_11 _LIBQ_Q4_11_ln_Q16(_Q16 x);
```

Returns

_LIBQ_Q4_11_ln_Q16 returns the _Q4_11 fixed point result from the calculation ln(x).

Description

Function _LIBQ_Q4_11_ln_Q16:

_Q4_11 _LIBQ_Q4_11_ln_Q16 (_Q16 x);

Calculates the natural logarithm ln(x). x is of type _Q16 and must be positive. The resulting value is of type _Q4_11.

Remarks

Execution Time (cycles): 301 typical (14 to 346) Program Memory 176 bytes

Error <= 0.00048828 (accurate to least significant _Q4_11 bit)

Preconditions

The input x must be positive.

Example

```
_Q4_11 resultLN;

resultLN = _LIBQ_Q4_11_ln_Q16 ((_Q16)0x00004000);  // _LIBQ_Q4_11_LN_Q16(0.250000) = -1.386230 (0xf4e9)
```

Parameters

Parameters	Description
x	The input value from which to calculate ln(x).

_LIBQ_Q5_10_log2_Q16 Function

Calculates the value of log2(x).

File

libq.h

C

```
_Q5_10 _LIBQ_Q5_10_log2_Q16(_Q16 x);
```

Returns

_LIBQ_Q5_10_log2_Q16 returns the _Q5_10 fixed point result from the calculation log2(x).

Description

Function _LIBQ_Q5_10_log2_Q16:

_Q5_10 _LIBQ_Q5_10_log2_Q16 (_Q16 x);

Calculates the log2(x), where log2(x) = ln(x) * log2(e). x is of type _Q16 and must be positive. The resulting value is of type _Q5_10.

Remarks

Execution Time (cycles): 227 typical (14 to 268) Program Memory 164 bytes

Error <= 0.0009765625 (accurate to least significant _Q5_10 bit)

Preconditions

The input x must be positive.

Example

```
_Q5_10 resultLog2;

resultLog2 = _LIBQ_Q5_10_log2_Q16 ((_Q16)0x40000000);  // _LIBQ_Q5_10_log2_Q16(16384.000000) = 14.000000
(0x3800)
```

Parameters

Parameters	Description
x	The input value from which to calculate log2(x).

d) Power Functions

_LIBQ_Q16Power Function

Calculates the value of x raised to the y power (x^y).

File

libq.h

C

```
_Q16 _LIBQ_Q16Power(_Q16 x, _Q16 y);
```

Returns

_LIBQ_Q16Power returns the _Q16 fixed point result from the calculation x raised to the y.

Description

Function _LIBQ_Q16Power:

_Q16 _LIBQ_Q16Power (_Q16 x, _Q16 y);

Calculates the x raised to the y power. Both x and y are of type _Q16. x must be positive. The calculation will saturate if the resulting value is outside the range of the _Q16 representation.

Remarks

Execution Time (cycles): 882 typical (586 to 1042) Program Memory 1038 bytes

Error <= 0.000015258789 (accurate to least significant _Q16 bit within the non-saturated range)

Preconditions

x must be positive.

Example

```
_Q16 resultPower;

resultPower = _LIBQ_Q16Power ((_Q16)0x00020000, (_Q16)0xffff0000);  // _LIBQ_Q16Power(2.000000, -1.000000)
= 0.500000 (0x00008000)
```

Parameters

Parameters	Description
x	The _Q16 input value x from which to calculate x raised to the y.
y	The _Q16 input value y from which to calculate x raised to the y.

e) Exponential Functions

_LIBQ_Q16Exp Function

Calculates the exponential function e^x.

File

libq.h

C

```
_Q16 _LIBQ_Q16Exp(_Q16 x);
```

Returns

_LIBQ_Q16Exp returns the _Q16 fixed point result from the calculation e^x.

Description

Function _LIBQ_Q16Exp:

_Q16 _LIBQ_Q16Exp(_Q16 x);

Calculates the exponential function e^x. The calculation will saturate if the resulting value is outside the range of the _Q16 representation. For x > 10.3972015380859375, the resulting value will be saturated to 0x7fffffff. For x < -10.3972015380859375 the resulting value will be saturated to 0.

Remarks

The function _LIBQ_Q16Div is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 170 typical (18 to 292) Program Memory 446 bytes

Error <= 0.000015258789 (accurate to least significant _Q16 bit within the non-saturated range)

Preconditions

None.

Example

```
_Q16 expResult;

expResult = _LIBQ_Q16Exp((_Q16)0x00010000);   // _LIBQ_Q16Exp(1.000000) = 2.718277 (0x0002b7e1)
```

Parameters

Parameters	Description
x	The exponent value

f) Sine Functions

_LIBQ_Q15_sin_Q2_13 Function

Calculates the value of sine(x).

File

libq.h

C

```
_Q15 _LIBQ_Q15_sin_Q2_13(_Q2_13 x);
```

Returns

_LIBQ_Q15_sin_Q2_13 returns the _Q15 fixed point result from the calculation sine(x).

Description

Function _LIBQ_Q15_sin_Q2_13:

_Q15 _LIBQ_Q15_sin_Q2_13 (_Q2_13 x);

Calculates the sine(x), where x is of type _Q2_13 radians and the resulting value is of type _Q15.

Remarks

Execution Time (cycles): 100 typical (100 to 102) Program Memory 220 bytes

Error <= 0.00003052 (accurate to least significant _Q15 bit)

Preconditions

None.

Example

```
_Q15 resultSin;

resultSin = _LIBQ_Q15_sin_Q2_13 ((_Q2_13)0x4093);  // _LIBQ_Q15_sin_Q2_13(2.017944) = 0.901672 (0x736a)
```

Parameters

Parameters	Description
x	The _Q2_13 input value from which to calculate sine(x).

_LIBQ_Q31_sin_Q2_29 Function

Calculates the value of sine(x).

File

libq.h

C

```
_Q31 _LIBQ_Q31_sin_Q2_29(_Q2_29 x);
```

Returns

_LIBQ_Q31_sin_Q2_29 returns the _Q31 fixed point result from the calculation sine(x).

Description

Function _LIBQ_Q31_sin_Q2_29:

_Q31 _LIBQ_Q31_sin_Q2_29 (_Q2_29 x);

Calculates the sine(x), where x is of type _Q2_29 radians and the resulting value is of type _Q31.

Remarks

Execution Time (cycles): 246 typical (244 to 266) Program Memory 598 bytes

Error <= 0.00000000047 (accurate to least significant _Q31 bit)

Preconditions

None.

Example

```
_Q31 resultSin;

resultSin = _LIBQ_Q31_sin_Q2_29 ((_Q2_29)0x5a637cfe);  // _LIBQ_Q31_sin_Q2_29( 2.824644562) =  0.311668121
(0x27e4bdb1)
```

Parameters

Parameters	Description
x	The _Q2_29 input value from which to calculate sine(x).

g) Cosine Functions

_LIBQ_Q15_cos_Q2_13 Function

Calculates the value of cosine(x).

File

libq.h

C

```
_Q15 _LIBQ_Q15_cos_Q2_13(_Q2_13 x);
```

Returns

_LIBQ_Q15_cos_Q2_13 returns the _Q15 fixed point result from the calculation cosine(x).

Description

Function _LIBQ_Q15_cos_Q2_13:

_Q15 _LIBQ_Q15_cos_Q2_13 (_Q2_13 x);

Calculates the cosine(x), where x is of type _Q2_13 radians and the resulting value is of type _Q15.

Remarks

Execution Time (cycles): 102 cycles Program Memory 224 bytes

Error <= 0.00003052 (accurate to least significant _Q15 bit)

Preconditions

None

Example

```
_Q15 resultCos;

resultCos = _LIBQ_Q15_cos_Q2_13 ((_Q2_13)0x2171);  // _LIBQ_Q15_cos_Q2_13(1.045044) = 0.501862 (0x403d)
```

Parameters

Parameters	Description
x	The _Q2_13 input value from which to calculate cosine(x).

_LIBQ_Q31_cos_Q2_29 Function

Calculates the value of cosine(x).

File

libq.h

C

```
_Q31 _LIBQ_Q31_cos_Q2_29(_Q2_29 x);
```

Returns

_LIBQ_Q31_cos_Q2_29 returns the _Q31 fixed point result from the calculation sine(x).

Description

Function _LIBQ_Q31_cos_Q2_29:

_Q31 _LIBQ_Q31_cos_Q2_29 (_Q2_29 x);

Calculates the cosine(x), where x is of type _Q2_29 radians and the resulting value is of type _Q31.

Remarks

Execution Time (cycles): 265 typical (22 to 288) Program Memory 746 bytes

Error <= 0.00000000047 (accurate to least significant _Q31 bit)

Preconditions

None.

Example

```
_Q31 resultCos;

resultCos = _LIBQ_Q31_cos_Q2_29 ((_Q2_29)0x07e2e1c2);  // _LIBQ_Q31_cos_Q2_29( 0.246445540) =  0.969785686
(0x7c21eff7)
```

Parameters

Parameters	Description
x	The _Q2_29 input value from which to calculate cosine(x).

h) Target Functions

_LIBQ_Q16_tan_Q2_29 Function

Calculates the value of tan(x).

File

libq.h

C

```
_Q16 _LIBQ_Q16_tan_Q2_29(_Q2_29 x);
```

Returns

_LIBQ_Q16_tan_Q2_29 returns the _Q16 fixed point result from the calculation tan(x). The resulting value is saturated.

Description

Function _LIBQ_Q16_tan_Q2_29:

_Q16 _LIBQ_Q16_tan_Q2_29 (_Q2_29 x);

Calculates the tan(x), where x is of type _Q2_29 radians and the resulting value is of type _Q16.

Remarks

The functions _LIBQ_Q31_sin_Q2_29, _LIBQ_Q31_cos_Q2_29, and _LIBQ_Q16Div are called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 703 typical (22 to 796) Program Memory 88 bytes

Error <= 0.000015259 (accurate to least significant _Q16 bit for the input range -1.568 .. 1.568) Error rises from 0.0 to 0.065 for the input range -1.568 .. -1.570765808 and 1.568 .. 1.570765808)

Preconditions

None

Example

```
_Q16 resultTan;

resultTan = _LIBQ_Q16_tan_Q2_29 ((_Q2_29)0x16720c36);  // _LIBQ_Q16_tan_Q2_29( 0.701421838) =  0.844726562 (0x0000d840)
```

Parameters

Parameters	Description
x	The _Q2_29 input value from which to calculate tan(x).

_LIBQ_Q7_8_tan_Q2_13 Function

Calculates the value of tan(x).

File

libq.h

C

```
_Q7_8 _LIBQ_Q7_8_tan_Q2_13(_Q2_13 x);
```

Returns

_LIBQ_Q7_8_tan_Q2_13 returns the _Q7_8 fixed point result from the calculation tan(x).

Description

Function _LIBQ_Q7_8_tan_Q2_13:

_Q7_8 _LIBQ_Q7_8_tan_Q2_13 (_Q2_13 x);

Calculates the tan(x), where x is of type _Q2_13 radians and the resulting value is of type _Q7_8.

Remarks

Execution Time (cycles): 288 typical (18 to 346) Program Memory 980 bytes

Error <= 0.00390625 (accurate to least significant _Q7_8 bit)

Preconditions

None

Example

```
_Q7_8 resultTan;

resultTan = _LIBQ_Q7_8_tan_Q2_13 ((_Q2_13)0x2e20);  // _LIBQ_Q7_8_tan_Q2_13(1.441406) = 7.683594 (0x07af)
```

Parameters

Parameters	Description
x	The _Q2_13 input value from which to calculate tan(x).

i) Arcsin Functions

_LIBQ_Q2_13_asin_Q15 Function

Calculates the asin value of asin(x).

File

libq.h

C

```
_Q2_13 _LIBQ_Q2_13_asin_Q15(_Q15 x);
```

Returns

_LIBQ_Q2_13_asin_Q15 returns the _Q2_13 fixed point result from the calculation asin(x).

Description

Function _LIBQ_Q2_13_asin_Q15:

_Q2_13 _LIBQ_Q2_13_asin_Q15 (_Q15 x);

Calculates asin(x), where x is of type _Q15 and the resulting value is of type _Q2_13. The output value will be radians in the range pi >= result >= -pi.

Remarks

The functions _LIBQ_Q16Sqrt and _LIBQ_Q16Div are called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 578 typical (22 to 656) Program Memory 336 bytes

Error <= 0.00012207 (accurate to least significant _Q2_13 bit)

A higher resolution version of this function exists with equivalent performance, see _LIBQ_Q2_29_asin_Q31_Fast

Preconditions

None.

Example

```
_Q2_13 resultAsin;

resultAsin = _LIBQ_Q2_13_asin_Q15 ((_Q15)0x3231);  // _LIBQ_Q2_13_asin_Q15(0.392120) = 0.402954 (0x0ce5)
```

Parameters

Parameters	Description
x	The _Q15 input value from which to calculate asin(x).

_LIBQ_Q2_29_asin_Q31 Function

Calculates the value of asin(x).

File

libq.h

C

```
_Q2_29 _LIBQ_Q2_29_asin_Q31(_Q31 x);
```

Returns

_LIBQ_Q2_29_asin_Q31 returns the _Q2_29 fixed point result from the calculation asin(x).

Description

Function _LIBQ_Q2_29_asin_Q31:

_Q2_29 _LIBQ_Q2_29_asin_Q31 (_Q31 x);

Calculates the asin(x), where x is of type _Q31 and the resulting value is of type _Q2_29. The output value will be in radians the range pi >= result >= -pi.

Remarks

The functions _LIBQ_Q2_29_asin_Q31_Fast and_LIBQ_Q31_sin_Q2_29 are called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 2525 typical (286 to 4330) Program Memory 138 bytes

Error <= 0.0000000019 (accurate to least significant _Q2_29 bit for the range -0.9993..0.9993) Error <= 0.0000000346 (accurate to 5th least significant _Q2_29 bit for the range -1.0 .. -0.9993 and 0.9993 .. 1.0)

A faster version of this function exists with modestly reduced accuracy, see _LIBQ_Q2_29_asin_Q31_Fast

Preconditions

None.

Example

```
_Q2_29 resultAsin;

resultAsin = _LIBQ_Q2_29_asin_Q31 ((_Q31)0x7fe50658);  // _LIBQ_Q2_29_asin_Q31( 0.9991767816) =
1.5302172359 (0x30f78a23)
```

Parameters

Parameters	Description
x	The _Q31 input value from which to calculate asin(x).

_LIBQ_Q2_29_asin_Q31_Fast Function

Calculates the value of asin(x). This function executes faster than the _LIBQ_Q2_29_asin_Q31 function, but is less precise.

File

libq.h

C

```
_Q2_29 _LIBQ_Q2_29_asin_Q31_Fast(_Q31 x);
```

Returns

_LIBQ_Q2_29_asin_Q31_Fast returns the _Q2_29 fixed point result from the calculation asin(x).

Description

Function _LIBQ_Q2_29_asin_Q31_Fast:

_Q2_29 _LIBQ_Q2_29_asin_Q31_Fast (_Q31 x);

Calculates the asin(x), where x is of type _Q31 and the resulting value is of type _Q2_29. The output value will be in radians the range pi >= result >= -pi.

Remarks

Execution Time (cycles): 507 typical (22 to 1300) Program Memory 638 bytes

Error <= 0.000000911 (accurate to 9 least significant _Q2_29 bits)

A higher resolution version of this function exists with reduced performance, see _LIBQ_Q2_29_asin_Q31

Preconditions

None.

Example

```
_Q2_29 resultAsin;

resultAsin = _LIBQ_Q2_29_asin_Q31_Fast ((_Q31)0x7fe50658);  // _LIBQ_Q2_29_asin_Q31_Fast( 0.9991767816) =
1.5302172359 (0x30f78a23)
```

Parameters

Parameters	Description
x	The _Q31 input value from which to calculate asin(x).

j) Arccos Functions

_LIBQ_Q2_13_acos_Q15 Function

Calculates the value of acos(x).

File

libq.h

C

```
_Q2_13 _LIBQ_Q2_13_acos_Q15(_Q15 x);
```

Returns

_LIBQ_Q2_13_acos_Q15 returns the _Q2_13 fixed point result from the calculation acos(x).

Description

Function _LIBQ_Q2_13_acos_Q15:

_Q2_13 _LIBQ_Q2_13_acos_Q15 (_Q15 x);

Calculates the acos(x), where x is of type _Q15 and the resulting value is of type _Q2_13. The output value will be radians in the range pi >= result >= -pi.

Remarks

The function _LIBQ_Q2_13_asin_Q15 is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 588 typical (32 to 666) Program Memory 24 bytes

Error <= 0.00012207 (accurate to least significant _Q2_13 bit)

A higher precision function with equivalent performance exists, see _LIBQ_Q2_29_acos_Q31_Fast

Preconditions

None.

Example

```
_Q2_13 resultAcos;

resultAcos = _LIBQ_Q2_13_acos_Q15((_Q15)0x2993); // _LIBQ_Q2_13_acos_Q15(0.324799) = 1.239990 (0x27ae)
```

Parameters

Parameters	Description
x	The _Q15 input value from which to calculate acos(x).

_LIBQ_Q2_29_acos_Q31 Function

Calculates the value of acos(x).

File

libq.h

C

```
_Q2_29 _LIBQ_Q2_29_acos_Q31(_Q31 x);
```

Returns

_LIBQ_Q2_29_acos_Q31 returns the _Q2_29 fixed point result from the calculation acos(x).

Description

Function _LIBQ_Q2_29_acos_Q31:

_Q2_29 _LIBQ_Q2_29_acos_Q31 (_Q31 x);

Calculates the acos(x), where x is of type _Q31 and the resulting value is of type _Q2_29. The output value will be radians in the range pi >= result >= -pi.

Remarks

The functions _LIBQ_Q2_29_asin_Q31_Fast and _LIBQ_Q31_cos_Q2_29 are called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 3370 typical (70 to 4824) Program Memory 142 bytes

Error <= 0.0000000019 (accurate to least significant _Q2_29 bit for the range -0.9993..0.9993) Error <= 0.0000000355 (accurate to 5th least significant _Q2_29 bit for the range -1.0 .. -0.9993 and 0.9993 .. 1.0)

A similar function with higher performance and reduced precision exists, see _LIBQ_Q2_29_acos_Q31_Fast

Preconditions

None.

Example

```
_Q2_29 resultAcos;

resultAcos = _LIBQ_Q2_29_acos_Q31 ((_Q31)0xee63708c);  // _LIBQ_Q2_29_acos_Q31(-0.1375903431) =
1.7088244837 (0x36aeb0af)
```

Parameters

Parameters	Description
x	The _Q31 input value from which to calculate acos(x).

_LIBQ_Q2_29_acos_Q31_Fast Function

Calculates the value of acos(x). This function executes faster than _LIBQ_Q2_29_acos_Q31 but is less precise.

File

libq.h

C

```
_Q2_29 _LIBQ_Q2_29_acos_Q31_Fast(_Q31 x);
```

Returns

_LIBQ_Q2_29_acos_Q31_Fast returns the _Q2_29 fixed point result from the calculation acos(x).

Description

Function _LIBQ_Q2_29_acos_Q31_Fast:

_Q2_29 _LIBQ_Q2_29_acos_Q31_Fast (_Q31 x);

Calculates the acos(x), where x is of type _Q31 and the resulting value is of type _Q2_29. The output value will be radians in the range pi >= result >= -pi.

Remarks

The function _LIBQ_Q2_29_asin_Q31_Fast is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 517 typical (32 to 1310) Program Memory 28 bytes

Error <= 0.000000911 (accurate to 9 least significant _Q2_29 bits)

A higher precision function with reduced performance exists, see _LIBQ_Q2_29_acos_Q31

Preconditions

None.

Example

```
_Q2_29 resultAcos;
```

```
resultAcos = _LIBQ_Q2_29_acos_Q31_Fast ((_Q31)0xee63708c);  // _LIBQ_Q2_29_acos_Q31_Fast(-0.1375903431) =
1.7088244837 (0x36aeb0af)
```

Parameters

Parameters	Description
x	The _Q31 input value from which to calculate acos(x).

k) Arctan Functions

_LIBQ_Q2_13_atan_Q7_8 Function

Calculates the value of atan(x).

File

libq.h

C

```
_Q2_13 _LIBQ_Q2_13_atan_Q7_8(_Q7_8 x);
```

Returns

_LIBQ_Q2_13_atan_Q7_8 returns the _Q2_13 fixed point result from the calculation atan(x).

Description

Function _LIBQ_Q2_13_atan_Q7_8:

_Q2_13 _LIBQ_Q2_13_atan_Q7_8 (_Q7_8 x);

Calculates the atan(x), where x is of type _Q7_8 and the resulting value is of type _Q2_13. The output value will be radians in the range pi >= result >= -pi.

Remarks

The function _LIBQ_Q2_13_atan2_Q7_8 is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 240 typical (202 to 256) Program Memory 16 bytes

Error <= 0.00012207 (accurate to least significant _Q2_13 bit)

Preconditions

None.

Example

```
_Q2_13 resultAtan;

resultAtan = _LIBQ_Q2_13_atan_Q7_8 ((_Q7_8)0x0097);  // _LIBQ_Q2_13_atan_Q7_8(0.589844) = 0.532959 (0x110e)
```

Parameters

Parameters	Description
x	The _Q7_8 input value from which to calculate atan(x).

_LIBQ_Q2_29_atan_Q16 Function

Calculates the value of atan(x).

File

libq.h

C

```
_Q2_29 _LIBQ_Q2_29_atan_Q16(_Q16 x);
```

Returns

_LIBQ_Q2_29_atan_Q16 returns the _Q2_29 fixed point result from the calculation atan(x).

Description

Function _LIBQ_Q2_29_atan_Q16:

_Q2_29 _LIBQ_Q2_29_atan_Q16 (_Q16 x);

Calculates the atan(x), where x is of type _Q16 and the resulting value is of type _Q2_29. The output value will be radians in the range pi >= result >= -pi.

Remarks

The function _LIBQ_Q2_29_atan2_Q16 is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 354 typical (178 to 360) Program Memory 16 bytes

Error <= 0.000000003 (accurate within one least significant _Q2_29 bit)

Preconditions

None.

Example

```
_Q2_29 resultAtan;

resultAtan = _LIBQ_Q2_29_atan_Q16 ((_Q16)0x00098b31);  // _LIBQ_Q2_29_atan_Q16(9.543716) = 1.466396
(0x2eecb7ee)
```

Parameters

Parameters	Description
x	The _Q16 input value from which to calculate atan(x).

l) Arctan2 Functions

_LIBQ_Q2_13_atan2_Q7_8 Function

Calculates the value of atan2(y, x).

File

libq.h

C

```
_Q2_13 _LIBQ_Q2_13_atan2_Q7_8(_Q7_8 y, _Q7_8 x);
```

Returns

_LIBQ_Q2_13_atan2_Q7_8 returns the _Q2_13 fixed point result from the calculation atan2(y, x).

Description

Function _LIBQ_Q2_13_atan2_Q7_8:

_Q2_13 _LIBQ_Q2_13_atan2_Q7_8 (_Q7_8 y, _Q7_8 x);

Calculates the atan2(y, x), where y and x are of type _Q7_8 and the resulting value is of type _Q2_13. The output value will be radians in the range pi >= result >= -pi.

Remarks

The function _LIBQ_Q16Div is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 220 typical (22 to 250) Program Memory 288 bytes

Error <= 0.00012207 (accurate to least significant _Q2_13 bit)

Preconditions

None.

Example

```
_Q2_13 resultAtan2;

resultAtan2 = _LIBQ_Q2_13_atan2_Q7_8 ((_Q7_8)0x589d, (_Q7_8)0xf878); // _LIBQ_Q2_13_atan2_Q7_8(88.613281,
-7.531250) = 1.655518 (0x34fa)
```

Parameters

Parameters	Description
y	The _Q7_8 input value from which to calculate atan2(y, x).
x	The _Q7_8 input value from which to calculate atan2(y, x).

_LIBQ_Q2_29_atan2_Q16 Function

Calculates the value of atan2(y, x).

File

libq.h

C

```
_Q2_29 _LIBQ_Q2_29_atan2_Q16(_Q16 y, _Q16 x);
```

Returns

_LIBQ_Q2_29_atan2_Q16 returns the _Q2_29 fixed point result from the calculation atan2(y, x).

Description

Function _LIBQ_Q2_29_atan2_Q16:

_Q2_29 _LIBQ_Q2_29_atan2_Q16 (_Q16 y, _Q16 x);

Calculates the atan(y, x), where y and x are of type _Q16 and the resulting value is of type _Q2_29. The output value will be radians in the range pi >= result >= -pi.

Remarks

The C function __divdi3 is called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 348 typical (20 to 376) Program Memory 464 bytes

Error <= 0.000000003 (accurate within one least significant _Q2_29 bit)

Preconditions

None.

Example

```
_Q2_29 resultAtan2;

resultAtan2 = _LIBQ_Q2_29_atan2_Q16 ((_Q16)0xf6276270, x(_Q16)0x34b4b4c0);  //
_LIBQ_Q2_29_atan2_Q16(-2520.615479, 13492.706055) = -0.184684 (0xfa1710c7)
```

Parameters

Parameters	Description
y	The _Q16 input value from which to calculate atan2(y, x).
x	The _Q16 input value from which to calculate atan2(y, x).

m) Random Number Functions

_LIBQ_Q15Rand Function

Generate a _Q15 random number.

File

libq.h

C

```
_Q15 _LIBQ_Q15Rand(int64_t * pSeed);
```

Returns

_LIBQ_Q15Rand returns a random _Q15 value. _LIBQ_Q15Rand also updates the int64_t *pSeed value.

Description

Function _LIBQ_Q15Rand:

_Q15 _LIBQ_Q15Rand (int64_t *pSeed);

Generates a _Q15 pseudo-random value based on the seed supplied as a parameter. The first time this function is called, the seed value must be supplied by the user; this initial seed value can either be constant or random, depending on whether the user wants to generate a repeatable or a non-repeatable pseudo-random sequence.

The function updates the *pSeed value each time it is called. The updated *pSeed value must be passed back to the function with each subsequent call.

Warning: The pseudo-random sequence generated by this function may be insufficient for cryptographic use.

Remarks

Execution Time (cycles): 32 Program Memory 92 bytes

Preconditions

None.

Example

```
// Initialize seed to a constant or random value
static int64_t  randomSeed = 0xA71078BE72D4C1F1;

_Q15           randomValue;

randomValue = _LIBQ_Q15Rand(&randomSeed);
...
randomValue = _LIBQ_Q15Rand(&randomSeed);
```

Parameters

Parameters	Description
pSeed	A pointer to the seed value used by the function to generate a pseudo-random sequence.

_LIBQ_Q31Rand Function

Generate a _Q31 random number.

File

libq.h

C

```
_Q31 _LIBQ_Q31Rand(int64_t * pSeed);
```

Returns

_LIBQ_Q31Rand returns a pseudo-random _Q31 value. _LIBQ_Q31Rand also updates the int64_t *pSeed value.

Description

Function _LIBQ_Q31Rand:

_Q31 _LIBQ_Q31Rand (int64_t *pSeed);

Generates a _Q31 pseudo-random value based on the seed supplied as a parameter. The first time this function is called, the seed value must be supplied by the user; this initial seed value can either be constant or random, depending on whether the user wants to generate a repeatable or a non-repeatable pseudo-random sequence.

The function updates the *pSeed value each time it is called. The updated *pSeed value must be passed back to the function with each subsequent call.

Warning: The pseudo-random sequence generated by this function may be insufficient for cryptographic use.

Remarks

Execution Time (cycles): 32 Program Memory 88 bytes

Preconditions

None.

Example

```
// Initialize seed to a constant or random value
```

```
static int64_t  randomSeed = 0x7F18BA710E72D4C1;

_Q31  randomValue;

randomValue = _LIBQ_Q31Rand(&randomSeed);
...
randomValue = _LIBQ_Q31Rand(&randomSeed);
```

Parameters

Parameters	Description
pSeed	A pointer to the seed value used by the function to generate a pseudo-random sequence.

n) Float Functions

_LIBQ_Q15FromFloat Function

Converts a float to a _Q15 value.

File

libq.h

C

```
_Q15 _LIBQ_Q15FromFloat(float x);
```

Returns

_LIBQ_Q15FromFloat returns the _Q15 fixed point value corresponding to the floating point (float) input value.

Description

Function _LIBQ_Q15FromFloat:

_Q15 _LIBQ_Q15FromFloat(float x);

Converts a floating point value to a _Q15 fixed point representation. The _Q15 fixed point value is returned by the function. The conversion will saturate if the value is outside the range of the _Q15 representation.

Remarks

The C library functions __gesf2, __lesf2, __addsf3, __mulsf3, and __fixsfsi are called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 213 typical (158 to 224) Program Memory 96 bytes

Preconditions

None.

Example

```
_Q15 q15;

q15 = _LIBQ_Q15FromFloat((float)0.5);  // q15 now equals (_Q15)0x4000

q15 = _LIBQ_Q15FromFloat((float)-1.0);  // q15 now equals (_Q15)0x8000

q15 = _LIBQ_Q15FromFloat((float)-0.233828);  // q15 now equals (_Q15)0xe212
```

Parameters

Parameters	Description
x	The float point value to convert to _Q15 fixed point

_LIBQ_Q31FromFloat Function

Converts a float to a _Q31 value.

File

libq.h

C

```
_Q31 _LIBQ_Q31FromFloat(float x);
```

Returns

_LIBQ_Q31FromFloat returns the _Q31 fixed point value corresponding to the floating point (float) input value.

Description

Function _LIBQ_Q31FromFloat:

_Q31 _LIBQ_Q31FromFloat(float x);

Converts a floating point value to a _Q31 fixed point representation. The _Q31 fixed point value is returned by the function. The conversion will saturate if the value is outside the range of the _Q31 representation.

Remarks

The C library functions __gesf2, __lesf2, __addsf3, __mulsf3, and __fixsfsi are called by this routine and thus must be linked into the executable image.

Execution Time (cycles): 210 typical (158 to 214) Program Memory 100 bytes

Preconditions

None.

Example

```
_Q31 q31;

q31 = _LIBQ_Q31FromFloat((float)0.000008);  // q31 now equals (_Q31)0x00004000

q31 = _LIBQ_Q31FromFloat((float)-1.0);  // q31 now equals (_Q31)0x80000000

q31 = _LIBQ_Q31FromFloat((float)0.690001);  // q31 now equals (_Q31)0x5851f400
```

Parameters

Parameters	Description
x	The floating point value to convert to _Q31 fixed point.

_LIBQ_ToFloatQ15 Function

Converts a _Q15 value to a float.

File

libq.h

C

```
float _LIBQ_ToFloatQ15(_Q15 x);
```

Returns

_LIBQ_ToFloatQ15 returns the floating point (float) value corresponding to the _Q15 input value.

Description

Function _LIBQ_ToFloatQ15:

float _LIBQ_ToFloatQ15(_Q15 x);

Converts a _Q15 fixed point value to a floating point representation. The floating point value is returned by the function.

Remarks

The C library functions __floatsisf and __divsf3 are called by this routine and thus must be linked in to the executable image.

Execution Time (cycles): 158 typical (54 to 176) Program Memory 28 bytes

Preconditions

None.

Example

```
float f;
```

```
f = _LIBQ_ToFloatQ15((_Q15)0x4000);  // f now equals 0.5

f = _LIBQ_ToFloatQ15((_Q15)0x8000);  // f now equals -1.0

f = _LIBQ_ToFloatQ15((_Q15)0xb7ff);  // f now equals -0.562531
```

Parameters

Parameters	Description
x	The _Q15 fixed point value to convert to float

_LIBQ_ToFloatQ31 Function

Converts a _Q31 value to a float.

File

libq.h

C

```
float _LIBQ_ToFloatQ31(_Q31 x);
```

Returns

_LIBQ_ToFloatQ31 returns the floating point (float) value corresponding to the _Q31 input value.

Description

Function _LIBQ_ToFloatQ31:

float _LIBQ_ToFloatQ31(_Q31 x);

Converts a _Q31 fixed point value to a floating point representation. The floating point value is returned by the function.

Remarks

The C library functions __floatsisf and __divsf3 are called by this routine and thus must be linked in to the executable image.

Execution Time (cycles): 163 typical (54 to 176) Program Memory 28 bytes

Preconditions

None.

Example

```
float f;

f = _LIBQ_ToFloatQ31((_Q31)0x00004000);  // f now equals 0.000008

f = _LIBQ_ToFloatQ31((_Q31)0x80000000);  // f now equals -1.0

f = _LIBQ_ToFloatQ31((_Q31)0x5851f42d);  // f now equals 0.690001
```

Parameters

Parameters	Description
x	The _Q31 fixed point value to convert to float

o) String Functions

_LIBQ_Q15FromString Function

ASCII to _Q15 conversion.

File

libq.h

C

```
_Q15 _LIBQ_Q15FromString(char * s);
```

Returns

_LIBQ_Q15FromString returns the _Q15 fixed point value represented by the input string.

Description

Function _LIBQ_Q15FromString:

_Q15 _LIBQ_Q15FromString(char *s);

Convert an ASCII string into a _Q15 fixed point value. The ASCII string must be in an -N.NNNNNN format. Leading spaces are ignored. The conversion stops at either the first non-conforming character in the string or the Null string terminator. There must be no spaces within the string value itself.

Remarks

Execution Time (cycles): 296 typical (28 to 346) Program Memory 172 bytes

Preconditions

None.

Example

```
_Q15  x;

x = _LIBQ_Q15FromString("0.125");      // x will equal 0.125 using
                            // an internal value of 0x1000

x = _LIBQ_Q15FromString("-1.0");       // x will equal -1.0 using
                            // an internal value of 0x8000

x = _LIBQ_Q15FromString("0.999969");   // x will equal 0.999969 using
                            // an internal value of 0x7FFF
```

Parameters

Parameters	Description
s	A pointer to the ASCII input string representing the _Q15 fixed point value.

_LIBQ_ToStringQ15 Function

_Q15 to ASCII conversion.

File

libq.h

C

```
void _LIBQ_ToStringQ15(_Q15 x, char * s);
```

Returns

An ASCII string that represents the _Q15 fixed point value in -N.NNNNNN format. The output string will be terminated by a Null (0x00) character.

Description

Function _LIBQ_ToStringQ15:

void _LIBQ_ToStringQ15(_Q15 x, char *s);

Convert a _Q15 fixed point value to an ASCII string representation in a -N.NNNNNN format.

Remarks

Execution Time (cycles): 118 typical (28 to 132) Program Memory 200 bytes

Preconditions

The character string "s" must be at least 10 characters long, including the Null string terminator.

Example

```
char  s[10];

_LIBQ_ToStringQ15((_Q15)0x1000, s);      // s will equal "0.125000"

_LIBQ_ToStringQ15((_Q15)0x8000, s);      // s will equal "-1.000000"
```

```
_LIBQ_ToStringQ15((_Q15)0x7FFF, s);      // s will equal "0.999969"
```

Parameters

Parameters	Description
x	The fixed point value to be converted into an ASCII string (_Q15)
s	A pointer to the output string of at least 10 characters

p) Data Types and Constants

_Q15_MAX Macro

File

libq.h

C

```
#define _Q15_MAX ((_Q15)0x7FFF)        // Maximum value of _Q15 (~1.0)
```

Description

Maximum value of _Q15 (~1.0)

_Q15_MIN Macro

File

libq.h

C

```
#define _Q15_MIN ((_Q15)0x8000)        // Minimum value of _Q15 (-1.0)
```

Description

Minimum value of _Q15 (-1.0)

_Q16_MAX Macro

File

libq.h

C

```
#define _Q16_MAX ((_Q16)0x7FFFFFFF)    // Maximum value of _Q16 (~32768.0)
```

Description

Maximum value of _Q16 (~32768.0)

_Q16_MIN Macro

File

libq.h

C

```
#define _Q16_MIN ((_Q16)0x80000000)    // Minimum value of _Q16 (-32768.0)
```

Description

Minimum value of _Q16 (-32768.0)

_Q2_13_MAX Macro

File

libq.h

C

```
#define _Q2_13_MAX ((_Q2_13)0x7FFF)      // Maximum value of _Q2_13 (~4.0)
```

Description

Maximum value of _Q2_13 (~4.0)

_Q2_13_MIN Macro

File

libq.h

C

```
#define _Q2_13_MIN ((_Q2_13)0x8000)      // Minimum value of _Q2_13 (-4.0)
```

Description

Minimum value of _Q2_13 (-4.0)

_Q2_29_MAX Macro

File

libq.h

C

```
#define _Q2_29_MAX ((_Q2_29)0x7FFFFFFF)  // Maximum value of _Q2_29 (~4.0)
```

Description

Maximum value of _Q2_29 (~4.0)

_Q2_29_MIN Macro

File

libq.h

C

```
#define _Q2_29_MIN ((_Q2_29)0x80000000)  // Minimum value of _Q2_29 (-4.0)
```

Description

Minimum value of _Q2_29 (-4.0)

_Q3_12_MAX Macro

File

libq.h

C

```
#define _Q3_12_MAX ((_Q3_12)0x7FFF)      // Maximum value of _Q3_12 (~8.0)
```

Description

Maximum value of _Q3_12 (~8.0)

_Q3_12_MIN Macro

File

libq.h

C

```
#define _Q3_12_MIN ((_Q3_12)0x8000)      // Minimum value of _Q3_12 (-8.0)
```

Description

Minimum value of _Q3_12 (-8.0)

_Q31_MAX Macro

File

libq.h

C

```
#define _Q31_MAX ((_Q31)0x7FFFFFFF)    // Maximum value of _Q31 (~1.0)
```

Description

Maximum value of _Q31 (~1.0)

_Q31_MIN Macro

File

libq.h

C

```
#define _Q31_MIN ((_Q31)0x80000000)    // Minimum value of _Q31 (-1.0)
```

Description

Minimum value of _Q31 (-1.0)

_Q4_11_MAX Macro

File

libq.h

C

```
#define _Q4_11_MAX ((_Q4_11)0x7FFF)      // Maximum value of _Q4_11 (~16.0)
```

Description

Maximum value of _Q4_11 (~16.0)

_Q4_11_MIN Macro

File

libq.h

C

```
#define _Q4_11_MIN ((_Q4_11)0x8000)      // Minimum value of _Q4_11 (-16.0)
```

Description

Minimum value of _Q4_11 (-16.0)

_Q5_10_MAX Macro

File

libq.h

C

```
#define _Q5_10_MAX ((_Q5_10)0x7FFF)      // Maximum value of _Q5_10 (~32.0)
```

Description

Maximum value of _Q5_10 (~32.0)

_Q5_10_MIN Macro

File

libq.h

C

```
#define _Q5_10_MIN ((_Q5_10)0x8000)      // Minimum value of _Q5_10 (-32.0)
```

Description

Minimum value of _Q5_10 (-32.0)

_Q7_8_MAX Macro

File

libq.h

C

```
#define _Q7_8_MAX ((_Q7_8)0x7FFF)       // Maximum value of _Q7_8 (~128.0)
```

Description

Maximum value of _Q7_8 (~128.0)

_Q7_8_MIN Macro

File

libq.h

C

```
#define _Q7_8_MIN ((_Q7_8)0x8000)       // Minimum value of _Q7_8 (-128.0)
```

Description

Minimum value of _Q7_8 (-128.0)

_Q0_15 Type

File

libq.h

C

```
typedef int16_t _Q0_15;
```

Description

1 sign bit, 15 bits right of radix

_Q0_31 Type

File

libq.h

C

```
typedef int32_t _Q0_31;
```

Description

1 sign bit, 31 bits right of radix

_Q15 Type

File

libq.h

C

```
typedef _Q0_15 _Q15;
```

Description

Short name for _Q0_15

_Q15_16 Type

File

libq.h

C

```
typedef int32_t _Q15_16;
```

Description

1 sign bit, 15 bits left of radix, 16 bits right of radix

_Q16 Type

File

libq.h

C

```
typedef _Q15_16 _Q16;
```

Description

Short name for _Q15_16

_Q2_13 Type

File

libq.h

C

```
typedef int16_t _Q2_13;
```

Description

1 sign bit, 2 bits left of radix, 13 bits right of radix

_Q2_29 Type

File

libq.h

C

```
typedef int32_t _Q2_29;
```

Description

1 sign bit, 2 bits left of radix, 29 bits right of radix

_Q3_12 Type

File

libq.h

C

```
typedef int16_t _Q3_12;
```

Description

1 sign bit, 3 bits left of radix, 12 bits right of radix

_Q31 Type

File

libq.h

C

```
typedef _Q0_31 _Q31;
```

Description

Short name for _Q_0_31

_Q4_11 Type

File

libq.h

C

```
typedef int16_t _Q4_11;
```

Description

1 sign bit, 4 bits left of radix, 11 bits right of radix

_Q5_10 Type

File

libq.h

C

```
typedef int16_t _Q5_10;
```

Description

1 sign bit, 5 bits left of radix, 10 bits right of radix

_Q7_8 Type

File

libq.h

C

```
typedef int16_t _Q7_8;
```

Description

1 sign bit, 7 bits left of radix, 8 bits right of radix

_LIBQ_H Macro

File

libq.h

C

```
#define _LIBQ_H
```

Description

Guards against multiple inclusion

Files

Files

Name	Description
libq.h	Optimized fixed point math functions for the PIC32MZ families of devices with microAptiv core features.

Description

This section lists the source and header files used by the LibQ Fixed-Point Math Library.

libq.h

Optimized fixed point math functions for the PIC32MZ families of devices with microAptiv core features.

Functions

	Name	Description
	_LIBQ_Q15_cos_Q2_13	Calculates the value of cosine(x).
	_LIBQ_Q15_sin_Q2_13	Calculates the value of sine(x).
	_LIBQ_Q15FromFloat	Converts a float to a _Q15 value.
	_LIBQ_Q15FromString	ASCII to _Q15 conversion.
	_LIBQ_Q15Rand	Generate a _Q15 random number.
	_LIBQ_Q16_tan_Q2_29	Calculates the value of tan(x).
	_LIBQ_Q16Div	_Q16 fixed point divide.
	_LIBQ_Q16Exp	Calculates the exponential function e^x.
	_LIBQ_Q16Power	Calculates the value of x raised to the y power (x^y).
	_LIBQ_Q16Sqrt	Square root of a positive _Q16 fixed point value.
	_LIBQ_Q2_13_acos_Q15	Calculates the value of acos(x).
	_LIBQ_Q2_13_asin_Q15	Calculates the asin value of asin(x).
	_LIBQ_Q2_13_atan_Q7_8	Calculates the value of atan(x).
	_LIBQ_Q2_13_atan2_Q7_8	Calculates the value of atan2(y, x).
	_LIBQ_Q2_29_acos_Q31	Calculates the value of acos(x).
	_LIBQ_Q2_29_acos_Q31_Fast	Calculates the value of acos(x). This function executes faster than _LIBQ_Q2_29_acos_Q31 but is less precise.
	_LIBQ_Q2_29_asin_Q31	Calculates the value of asin(x).

	_LIBQ_Q2_29_asin_Q31_Fast	Calculates the value of asin(x). This function executes faster than the _LIBQ_Q2_29_asin_Q31 function, but is less precise.
	_LIBQ_Q2_29_atan_Q16	Calculates the value of atan(x).
	_LIBQ_Q2_29_atan2_Q16	Calculates the value of atan2(y, x).
	_LIBQ_Q3_12_log10_Q16	Calculates the value of Log10(x).
	_LIBQ_Q31_cos_Q2_29	Calculates the value of cosine(x).
	_LIBQ_Q31_sin_Q2_29	Calculates the value of sine(x).
	_LIBQ_Q31FromFloat	Converts a float to a _Q31 value.
	_LIBQ_Q31Rand	Generate a _Q31 random number.
	_LIBQ_Q4_11_ln_Q16	Calculates the natural logarithm ln(x).
	_LIBQ_Q5_10_log2_Q16	Calculates the value of log2(x).
	_LIBQ_Q7_8_tan_Q2_13	Calculates the value of tan(x).
	_LIBQ_ToFloatQ15	Converts a _Q15 value to a float.
	_LIBQ_ToFloatQ31	Converts a _Q31 value to a float.
	_LIBQ_ToStringQ15	_Q15 to ASCII conversion.

Macros

	Name	Description
	_LIBQ_H	Guards against multiple inclusion
	_Q15_MAX	Maximum value of _Q15 (~1.0)
	_Q15_MIN	Minimum value of _Q15 (-1.0)
	_Q16_MAX	Maximum value of _Q16 (~32768.0)
	_Q16_MIN	Minimum value of _Q16 (-32768.0)
	_Q2_13_MAX	Maximum value of _Q2_13 (~4.0)
	_Q2_13_MIN	Minimum value of _Q2_13 (-4.0)
	_Q2_29_MAX	Maximum value of _Q2_29 (~4.0)
	_Q2_29_MIN	Minimum value of _Q2_29 (-4.0)
	_Q3_12_MAX	Maximum value of _Q3_12 (~8.0)
	_Q3_12_MIN	Minimum value of _Q3_12 (-8.0)
	_Q31_MAX	Maximum value of _Q31 (~1.0)
	_Q31_MIN	Minimum value of _Q31 (-1.0)
	_Q4_11_MAX	Maximum value of _Q4_11 (~16.0)
	_Q4_11_MIN	Minimum value of _Q4_11 (-16.0)
	_Q5_10_MAX	Maximum value of _Q5_10 (~32.0)
	_Q5_10_MIN	Minimum value of _Q5_10 (-32.0)
	_Q7_8_MAX	Maximum value of _Q7_8 (~128.0)
	_Q7_8_MIN	Minimum value of _Q7_8 (-128.0)

Types

	Name	Description
	_Q0_15	1 sign bit, 15 bits right of radix
	_Q0_31	1 sign bit, 31 bits right of radix
	_Q15	Short name for _Q0_15
	_Q15_16	1 sign bit, 15 bits left of radix, 16 bits right of radix
	_Q16	Short name for _Q15_16
	_Q2_13	1 sign bit, 2 bits left of radix, 13 bits right of radix
	_Q2_29	1 sign bit, 2 bits left of radix, 29 bits right of radix
	_Q3_12	1 sign bit, 3 bits left of radix, 12 bits right of radix
	_Q31	Short name for _Q_0_31
	_Q4_11	1 sign bit, 4 bits left of radix, 11 bits right of radix
	_Q5_10	1 sign bit, 5 bits left of radix, 10 bits right of radix
	_Q7_8	1 sign bit, 7 bits left of radix, 8 bits right of radix

Description

The LibQ Fixed-Point Math Library provides fixed-point math functions that are optimized for performance on the PIC32MZ families of devices that have microAptiv core features. All functions are optimized for speed. This header file specifies characteristics of each function, including execution

time, memory size, and resolution.

Signed fixed point types are defined as follows:

Qn_m where:

- n is the number of data bits to the left of the radix point
- m is the number of data bits to the right of the radix point
- a signed bit is implied

For convenience, short names are also defined:

Exact Name (& Bits) Required Short Name _Q0_15 (16) _Q15; _Q15_16 (32) _Q16; _Q0_31 (32) _Q31

Functions in the library are prefixed with the type of the return value. For example, _LIBQ_Q16Sqrt returns a Q16 value equal to the square root of its argument.

Argument types do not always match the return type. Refer to the function prototype for a specification of its arguments.

In cases where the return value is not a fixed point type, the argument type is appended to the function name. For example, _LIBQ_ToFloatQ31 accepts a type _Q31 argument.

In some cases, both the return type and the argument type are specified within the function name. For example,

Function Name (Return Type) [Argument Type]: _LIBQ_Q15_sin_Q2_13 (_Q15) [_Q2_13]; _LIBQ_Q31_sin_Q2_29 (_Q31) [_Q2_29]

Table of LIBQ functions:

Divide: _Q16 _LIBQ_Q16Div (_Q16 dividend, _Q16 divisor);

Square root: _Q16 _LIBQ_Q16Sqrt (_Q16 x);

Exponential: _Q16 _LIBQ_Q16Exp (_Q16 x);

Log: _Q4_11 _LIBQ_Q4_11_ln_Q16 (_Q16 x); _Q3_12 _LIBQ_Q3_12_log10_Q16 (_Q16 x); _Q5_10 _LIBQ_Q5_10_log2_Q16 (_Q16 x);

Power: _Q16 _LIBQ_Q16Power (_Q16 x, _Q16 y);

Sine: _Q15 _LIBQ_Q15_sin_Q2_13 (_Q2_13 x); _Q31 _LIBQ_Q31_sin_Q2_29 (_Q2_29 x);

Cosine: _Q15 _LIBQ_Q15_cos_Q2_13 (_Q2_13 x); _Q31 _LIBQ_Q31_cos_Q2_29 (_Q2_29 x);

Tangent: _Q7_8 _LIBQ_Q7_8_tan_Q2_13 (_Q2_13 x); _Q16z _LIBQ_Q16_tan_Q2_29 (_Q2_29 x);

Arcsin: _Q2_13 _LIBQ_Q2_13_asin_Q15 (_Q15 x); _Q2_29 _LIBQ_Q2_29_asin_Q31 (_Q31 x); _Q2_29 _LIBQ_Q2_29_asin_Q31_Fast (_Q31 x);

Arccos: _Q2_13 _LIBQ_Q2_13_acos_Q15 (_Q15 x); _Q2_29 _LIBQ_Q2_29_acos_Q31 (_Q31 x); _Q2_29 _LIBQ_Q2_29_acos_Q31_Fast (_Q31 x);

Arctan: _Q2_13 _LIBQ_Q2_13_atan_Q7_8 (_Q7_8 x); _Q2_29 _LIBQ_Q2_29_atan_Q16 (_Q16 x);

Arctan2: _Q2_13 _LIBQ_Q2_13_atan2_Q7_8 (_Q7_8 y, _Q7_8 x); _Q2_29 _LIBQ_Q2_29_atan2_Q16 (_Q16 y, _Q16 x);

Random number: _Q15 _LIBQ_Q15Rand (int64_t &pSeed); _Q31 _LIBQ_Q31Rand (int64_t &pSeed);

Float: float _LIBQ_ToFloatQ31 (_Q31 x); float _LIBQ_ToFloatQ15 (_Q15 x); _Q31 _LIBQ_Q31FromFloat (float x); _Q15 _LIBQ_Q15FromFloat (float x);

String: void _LIBQ_ToStringQ15 (_Q15 x, char &s); _Q15 _LIBQ_Q15FromString (char &s);

File Name

libq.h

Company

Microchip Technology Inc.

Networking Presentation Layer Help

This section describes the MPLAB Harmony Networking Presentation Layer.

Introduction

This library provides a Networking Presentation Layer that is available on the Microchip family of microcontrollers with a convenient C language interface.

Description

The MPLAB Harmony Networking Presentation Layer is an abstracted middleware layer that provides an encrypted channel. There are three interfaces for the layer:

- Client - this interface starts a network session and reads and writes clear text data
- Provider - this interface encrypts the clear text into cipher text, and decrypts cipher text into clear text
- Transport - this interface transmits and receives the cipher text

The linkage to the provider interface is done through function pointers to isolate it from the rest of MPLAB Harmony. This allows for any provider to be used. Please note that Microchip *does not* supply a commercially licensed TLS provider. wolfSSL (formerly CyaSSL) is supported as a provider for evaluation purposes only.

The transport interface is abstracted to function pointers as well, to allow for any transport to be used. Currently, only the MPLAB Harmony TCP/IP Stack is used as a transport; however, other transports can be created, such as using UART, which can be used instead.

The MPLAB Harmony Networking Presentation Layer has a Tasks function that is used to continuously call the provider's connect and accept functions until they run to success of failure, thereby hiding this job from the application.

Using the Library

This topic describes the basic architecture of the Networking Presentation Layer and provides information and examples on its use.

Description

Interface Header File: `net_pres.h`

The interface to the Networking Presentation Layer is defined in the `net_pres.h` header file. Any C language source (`.c`) file that uses the Networking Presentation Layer should include `net_pres.h`.

Library File:

The Networking Presentation Layer archive (`.a`) file is installed with MPLAB Harmony.

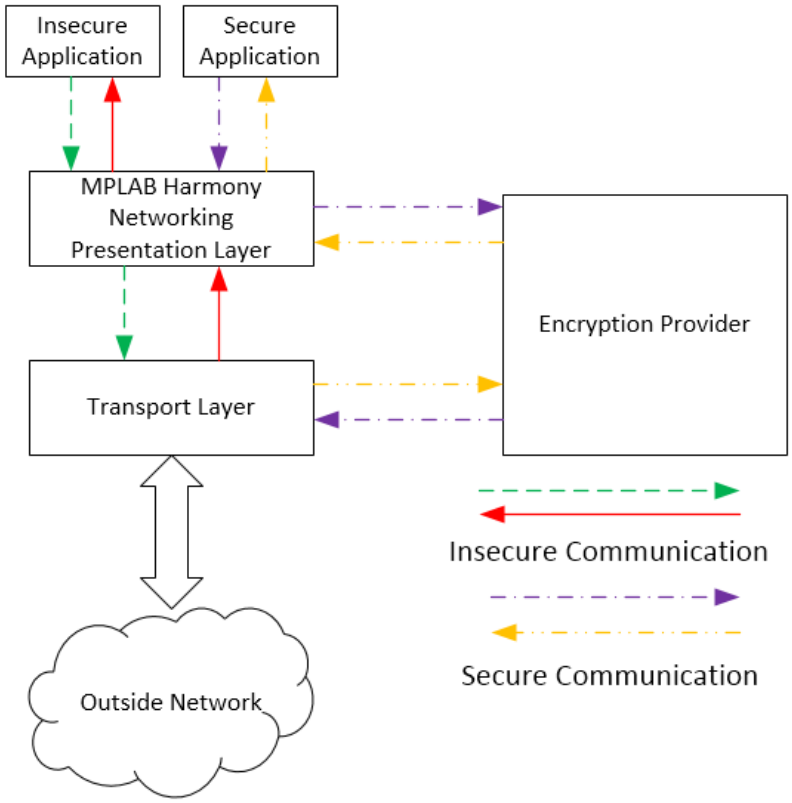
Please refer to the What is MPLAB Harmony? section for how the Networking Presentation Layer interacts with the framework.

Abstraction Model

This library provides the low-level abstraction of the Network Presentation Layer on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Software Abstraction Block Diagram



Library Overview

The Library Interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Networking Presentation Layer.

Library Interface Section	Description
System Functions	Provides "initialize" functions and a "tasks" function for performing general presentation layer tasks.
Socket Functions	Provides various socket-related functions for the purpose of connecting, flushing, and setting options, among others.
Certificate Store Functions	Provides certificate-related functions for the purpose of obtaining certificates and keys.

Configuring the Library

The configuration of the Networking Presentation Layer is based on the file `system_config.h`.

This header file contains the configuration selection for the Networking Presentation Layer. Based on the selections made, the Networking Presentation Layer may support the selected features. These configuration settings will apply to all instances of the Networking Presentation Layer.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Networking Presentation Layer.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/net/pres`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
net_pres.h	This file provides the interface definitions of the Network Presentation Layer.
net_pres_certstore.h	This file describes the common API used to access the certificate store.
net_pres_encryptionproviderapi.h	This file describes the API that an encryption provider must follow to be compatible with the Networking Presentation Layer.
net_pres_socketapi.h	This file describes the API that the application uses to send and receive data though Networking Presentation Layer sockets.
net_pres_transportapi.h	This file descriptions the API that a transport layer must follow to be compatible with the Networking Presentation Layer.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/net_pres.c	This file implements the Networking Presentation Layer API for both the socket layer and the system interface.

Optional File(s)

There are no optional files for the Networking Presentation Layer.

Module Dependencies

The Networking Presentation Layer is not dependent upon other modules.

Library Interface

a) System Functions

	Name	Description
	NET_PRES_Deinitialize	Deinitializes the Network Presentation Layer Instance. **Implementation:** Dynamic
	NET_PRES_Initialize	Initializes the Network Presentation Layer sub-system with the configuration data. **Implementation:** Dynamic
	NET_PRES_Reinitialize	Reinitializes the instance of the presentation layer. **Implementation:** Dynamic
	NET_PRES_Tasks	MPLAB Harmony tasks function used for general presentation layer tasks. **Implementation:** Dynamic
	NET_PRES_Status	Provides the current status of the MPLAB Harmony Networking Presentation Layer. **Implementation:** Dynamic
	NET_PRES_EncProviderReadReady	Defines the read ready function to the provider **Implementation:** Dynamic

b) Socket Functions

	Name	Description
	NET_PRES_SocketIsOpenModeSupported	Checks to see if a mode is supported by open.
	NET_PRES_SocketOpen	Opens a presentation socket.
	NET_PRES_SocketBind	Binds a socket to a local address.
	NET_PRES_SocketClose	Disconnects an open socket and destroys the socket handle, releasing the associated resources.
	NET_PRES_SocketConnect	Connects a client socket.
	NET_PRES_SocketDiscard	Discards any pending data in the RX FIFO.
	NET_PRES_SocketDisconnect	Disconnects an open socket.
	NET_PRES_SocketEncryptSocket	This function turns an insecure socket into a secure socket. Details: This function will turn an unencrypted socket into an encrypted socket and starts encryption negotiation.

	NET_PRES_SocketFlush	Immediately transmits all pending TX data.
	NET_PRES_SocketInfoGet	Obtains information about a currently open socket.
	NET_PRES_SocketIsConnected	Determines whether a socket has an established connection.
	NET_PRES_SocketIsNegotiatingEncryption	This function checks if encryption negotiation is still in progress.
	NET_PRES_SocketIsSecure	This function checks whether a connection is secure.
	NET_PRES_SocketLastError	This function returns the last error code for this socket. Details: This function will return the last error code that was set for this socket and it will clear the current error code. An error code is set whenever a socket operation fails for some missing functionality, bad parameter, etc.
	NET_PRES_SocketOptionsGet	Allows the options for a socket such as, current RX/TX buffer size, etc., to be obtained.
	NET_PRES_SocketOptionsSet	Allows setting options to a socket like adjust RX/TX buffer size, etc.
	NET_PRES_SocketPeek	Reads a specified number of data bytes from the RX buffer/FIFO without removing them from the buffer.
	NET_PRES_SocketRead	Reads an array of data bytes from a socket's RX buffer/FIFO.
	NET_PRES_SocketReadIsReady	Determines how many bytes can be read from the RX buffer.
	NET_PRES_SocketRemoteBind	Binds a socket to a remote local address.
	NET_PRES_SocketSignalHandlerDeregister	Deregisters a previously registered socket signal handler.
	NET_PRES_SocketSignalHandlerRegister	Registers a socket signal handler.
	NET_PRES_SocketWasReset	Self-clearing semaphore indicating socket reset.
	NET_PRES_SocketWrite	Takes a buffer and sends it to the encryption provider.
	NET_PRES_SocketGetTransportHandle	This function returns the transport layer handle. Details: This function returns the transport layer handle for a valid socket
	NET_PRES_SocketWriteIsReady	Determines how much free space is available in the TX buffer.

c) Certificate Store Functions

	Name	Description
	NET_PRES_CertStoreGetCACerts	This function gets the CA certificates from the store, **Implementation:** Dynamic
	NET_PRES_CertStoreGetServerCert	This function gets a server certificate and key from the certificate store. **Implementation:** Dynamic

d) Data Types and Constants

	Name	Description
	_NET_PRES_EncProviderObject	Defines the data that the presentation layer needs from the provider.
	_NET_PRES_TransportObject	Defines the data that the transport layer needs to provide to the Networking Presentation Layer.
	NET_PRES_ADDRESS	This is type NET_PRES_ADDRESS.
	NET_PRES_EncProviderConnect	Connects the function to the provider. **Implementation:** Dynamic
	NET_PRES_EncProviderConnectionClose	Defines the close function to the provider. **Implementation:** Dynamic
	NET_PRES_EncProviderDeinit	Defines the deinitialization function for the provider. **Implementation:** Dynamic
	NET_PRES_EncProviderInit	Defines the initialization function to the encryption provider. **Implementation:** Dynamic
	NET_PRES_EncProviderIsInitialized	Determines whether the encryption provider has been initialized. **Implementation:** Dynamic
	NET_PRES_EncProviderObject	Defines the data that the presentation layer needs from the provider.
	NET_PRES_EncProviderOpen	Defines the open connection function to the provider. **Implementation:** Dynamic
	NET_PRES_EncProviderRead	Defines the read function to the provider **Implementation:** Dynamic
	NET_PRES_EncProviderWrite	Defines the write function to the provider. **Implementation:** Dynamic
	NET_PRES_EncSessionStatus	Defines the enumeration for the state and status of the encrypted portion of a connection.
	NET_PRES_INDEX	Sets the type for the presentation layer index.

	NET_PRES_INIT_DATA	Initializes a Presentation layer.
	NET_PRES_INST_DATA	Initializes a Presentation layer.
	NET_PRES_SIGNAL_FUNCTION	MPLAB Harmony Networking Presentation Layer Signal function.
	NET_PRES_SIGNAL_HANDLE	Sets the type for the presentation layer signal handle.
	NET_PRES_SKT_ADDR_T	This is type NET_PRES_SKT_ADDR_T.
	NET_PRES_SKT_ERROR_T	This is type NET_PRES_SKT_ERROR_T.
	NET_PRES_SKT_HANDLE_T	Sets the type for the presentation layer socket handle.
	NET_PRES_SKT_OPTION_TYPE	This is type NET_PRES_SKT_OPTION_TYPE.
	NET_PRES_SKT_T	This is type NET_PRES_SKT_T.
	NET_PRES_TRANS_ADDR_T	Defines a generic address structure to pass to the transport layer.
	NET_PRES_TRANS_ADDRESS_TYPE	Defines the enumeration for the type of address.
	NET_PRES_TRANS_OPTION_T	Defines the enumeration for the type of options.
	NET_PRES_TransBind	Binds a socket to a local address.
	NET_PRES_TransBool	Generic function prototype for functions that return a bool.
	NET_PRES_TransClose	Function prototype for functions that closes a socket.
	NET_PRES_TransDiscard	Function prototype for functions that clears a socket's RX buffer.
	NET_PRES_TransHandlerRegister	Function prototype that registers a handler with a socket.
	NET_PRES_TransOpen	Opens a presentation socket.
	NET_PRES_TransOption	Sets of gets a socket's options.
	NET_PRES_TransPeek	Function prototype for functions that peeks on the socket's buffer.
	NET_PRES_TransportObject	Defines the data that the transport layer needs to provide to the Networking Presentation Layer.
	NET_PRES_TransRead	Defines the read function provided by the transport layer. **Implementation:** Dynamic
	NET_PRES_TransReady	Defines the ready function provided by the transport layer. **Implementation:** Dynamic
	NET_PRES_TransSignalHandlerDeregister	Function prototype that deregisters a handler with a socket.
	NET_PRES_TransSocketInfoGet	Function prototype for functions that gets the information on a socket.
	NET_PRES_TransWrite	Defines the write function provided by the transport layer. **Implementation:** Dynamic
	NET_PRES_INVALID_SOCKET	Invalid socket indicator macro.
	NET_PRES_SKT_PORT_T	Sets the type for the presentation layer port.
	NET_PRES_TransIsPortDefaultSecured	Checks to see if a port is encrypted by default. **Implementation:** Dynamic
	NET_PRES_EncProviderWriteReady	Defines the write ready function to the provider. **Implementation:** Dynamic

Description

This section describes the Application Programming Interface (API) functions of the Networking Presentation Layer.

Refer to each section for a detailed description.

a) System Functions

NET_PRES_Deinitialize Function

Deinitializes the Network Presentation Layer Instance.

Implementation: Dynamic

File

net_pres.h

C

```
void NET_PRES_Deinitialize(SYS_MODULE_OBJ obj);
```

Returns

None.

Description

Network Presentation Layer Deinitialization

This function deallocates any resources allocated by the initialization function.

Preconditions

The layer must be successfully initialized with NET_PRES_Initialize.

Parameters

Parameters	Description
Object	the valid object returned from NET_PRES_Initialize

NET_PRES_Initialize Function

Initializes the Network Presentation Layer sub-system with the configuration data.

Implementation: Dynamic

File

net_pres.h

C

```
SYS_MODULE_OBJ NET_PRES_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

- Valid handle to the presentation instance - If successful
- SYS_MODULE_OBJ_INVALID - If unsuccessful

Description

Network Presentation Layer Initialization

Initializes the Network Presentation Layer sub-system with the configuration data.

Preconditions

None.

Parameters

Parameters	Description
index	This is the index of the network presentation layer instance to be initialized. Since there is only one network presentation layer, this parameter is ignored.
init	This is a pointer to a NET_PRES_INIT_DATA structure

NET_PRES_Reinitialize Function

Reinitializes the instance of the presentation layer.

Implementation: Dynamic

File

net_pres.h

C

```
void NET_PRES_Reinitialize(SYS_MODULE_OBJ obj, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

Network Presentation Layer Reinitialization

This function will deinitialize and initialize the layer instance.

Preconditions

The layer must be successfully initialized with NET_PRES_Initialize.

Parameters

Parameters	Description
object	The object valid passed back to NET_PRES_Initialize
init	The new initialization structure

NET_PRES_Tasks Function

MPLAB Harmony tasks function used for general presentation layer tasks.

Implementation: Dynamic

File

net_pres.h

C

```
void NET_PRES_Tasks(SYS_MODULE_OBJ obj);
```

Returns

None.

Description

MPLAB Harmony Networking Presentation Layer Tasks

This function is called by the main loop. It is used to pump encryption connections during negotiations.

Preconditions

The layer must be successfully initialized with NET_PRES_Initialize.

Parameters

Parameters	Description
object	The valid object passed back to NET_PRES_Initialize

NET_PRES_Status Function

Provides the current status of the MPLAB Harmony Networking Presentation Layer.

Implementation: Dynamic

File

net_pres.h

C

```
SYS_STATUS NET_PRES_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed
- SYS_STATUS_UNINITIALIZED - Indicates the module has not been initialized
- SYS_STATUS_BUSY - Indicates that the module is busy and can't accept operations
- SYS_STATUS_ERROR - Indicates that there is a fatal error in the module

Description

This function provides the current status of the MPLAB Harmony Net Presentation Layer.

Remarks

None.

Preconditions

The NET_PRES_Initialize function must have been called before calling this function.

Parameters

Parameters	Description
object	Layer object handle, returned from NET_PRES_Initialize

NET_PRES_EncProviderReadReady Type

Defines the read ready function to the provider

Implementation: Dynamic

File

net_pres_encryptionproviderapi.h

C

```
typedef int32_t (* NET_PRES_EncProviderReadReady)(void * providerData);
```

Returns

The number of bytes ready to be read.

Description

Presentation Encryption Provider Read Ready Function Pointer Prototype

This function pointer defines the read ready function. It is called by the presentation layer when the presentation client wants to check whether read data is available from a secured connection.

Preconditions

A connection must have already been created, and be in the open state.

Parameters

Parameters	Description
providerData	A pointer to the buffer for the provider to keep connection specific data.

b) Socket Functions

NET_PRES_SocketIsOpenModeSupported Function

Checks to see if a mode is supported by open.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketIsOpenModeSupported(NET_PRES_INDEX index, NET_PRES_SKT_T socketType);
```

Returns

- true - The mode is supported
- false - The mode is not supported

Description

This function checks to see if a mode is supported by open.

Preconditions

The MPLAB Harmony Networking Presentation Layer is initialized.

Parameters

Parameters	Description
index	Index of the presentation layer.
socketType	The type of socket to mode to be checked.

NET_PRES_SocketOpen Function

Opens a presentation socket.

File

net_pres_socketapi.h

C

```
NET_PRES_SKT_HANDLE_T NET_PRES_SocketOpen(NET_PRES_INDEX index, NET_PRES_SKT_T socketType,
NET_PRES_SKT_ADDR_T addrType, NET_PRES_SKT_PORT_T port, NET_PRES_ADDRESS * addr, NET_PRES_SKT_ERROR_T*
error);
```

Returns

- NET_PRES_INVALID_SOCKET - No sockets of the specified type were available to be opened
- NET_PRES_SKT_HANDLE_T handle - Returned when NET_PRES_INVALID_SOCKET is returned. Save this handle and use it when calling all other presentation socket APIs.

Description

Provides a unified method for opening all presentation sockets types. Sockets are created at the presentation layer module initialization, and can be claimed with this function and freed using NET_PRES_SocketClose. The presentation layer will call the corresponding open function in the transport layer, and if encryption is specified the presentation layer will also handle encryption negotiation.

Preconditions

The MPLAB Harmony Networking Presentation Layer is initialized.

Parameters

Parameters	Description
index	Index of the presentation layer.
socketType	The type of socket to open.
addType	The type of address being used. This is passed unaltered to the transport layer.
port	The port to listen or to send to. This is passed unaltered to the transport layer.
addr	Address to use. This is passed unaltered to the transport layer.
error	The extended error code of the function.

NET_PRES_SocketBind Function

Binds a socket to a local address.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketBind(NET_PRES_SKT_HANDLE_T handle, NET_PRES_SKT_ADDR_T addrType, NET_PRES_SKT_PORT_T
port, NET_PRES_ADDRESS * addr);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This function calls directly to the transport layer's bind function.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The socket to bind.
addType	The type of address being used. This is passed unaltered to the transport layer.
port	The port to use. This is passed unaltered to the transport layer.
addr	The address to bind to. This is passed unaltered to the transport layer.

NET_PRES_SocketClose Function

Disconnects an open socket and destroys the socket handle, releasing the associated resources.

File

net_pres_socketapi.h

C

```
void NET_PRES_SocketClose(NET_PRES_SKT_HANDLE_T handle);
```

Returns

None.

Description

This function calls the encryption provider's close function and then calls the close function of the transport layer for the socket and frees the socket for reuse.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.

NET_PRES_SocketConnect Function

Connects a client socket.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketConnect(NET_PRES_SKT_HANDLE_T handle);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This function calls the transport layer's connect function directly, if it exists.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.

NET_PRES_SocketDiscard Function

Discards any pending data in the RX FIFO.

File

net_pres_socketapi.h

C

```
uint16_t NET_PRES_SocketDiscard(NET_PRES_SKT_HANDLE_T handle);
```

Returns

The number of bytes that have been discarded from the RX buffer.

Description

This function calls the transport layer's discard function, if it exists.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen

Parameters

Parameters	Description
handle	The presentation layer socket handle.

NET_PRES_SocketDisconnect Function

Disconnects an open socket.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketDisconnect(NET_PRES_SKT_HANDLE_T handle);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This function calls the transport layer's disconnect function directly, if it exists.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.

NET_PRES_SocketEncryptSocket Function

This function turns an insecure socket into a secure socket.

Details: This function will turn an unencrypted socket into an encrypted socket and starts encryption negotiation.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketEncryptSocket(NET_PRES_SKT_HANDLE_T handle);
```

Returns

- true - If the call was successful
- false - If the call was unsuccessful

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.

NET_PRES_SocketFlush Function

Immediately transmits all pending TX data.

File

net_pres_socketapi.h

C

```
uint16_t NET_PRES_SocketFlush(NET_PRES_SKT_HANDLE_T handle);
```

Returns

- The number of flushed bytes
- 0, if no flushed bytes or an error occurred

Description

This function calls the transport layer's flush function, if it exists.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.

NET_PRES_SocketInfoGet Function

Obtains information about a currently open socket.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketInfoGet(NET_PRES_SKT_HANDLE_T handle, void * info);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This function calls the transport layer's SocketInfoGet, if it exists.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.
info	The buffer that the information gets written to.

NET_PRES_SocketIsConnected Function

Determines whether a socket has an established connection.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketIsConnected(NET_PRES_SKT_HANDLE_T handle);
```

Description

This function determines whether a socket has an established connection to a remote node. This function calls directly to the transport layer's IsConnected function, if it exists.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.

NET_PRES_SocketIsNegotiatingEncryption Function

This function checks if encryption negotiation is still in progress.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketIsNegotiatingEncryption(NET_PRES_SKT_HANDLE_T handle);
```

Returns

- true - If the encryption negotiation is still ongoing
- false - If there is no ongoing negotiation

Description

This function returns checks to see if an encrypted socket is still undergoing negotiation.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.

NET_PRES_SocketIsSecure Function

This function checks whether a connection is secure.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketIsSecure(NET_PRES_SKT_HANDLE_T handle);
```

Returns

- true - If the communications is secure
- false - If the communications is not secure

Description

This function returns whether or not the connection is secure. It will return true if encryption negotiation was successful .

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.

NET_PRES_SocketLastError Function

This function returns the last error code for this socket.

Details: This function will return the last error code that was set for this socket and it will clear the current error code. An error code is set whenever

a socket operation fails for some missing functionality, bad parameter, etc.

File

net_pres_socketapi.h

C

```
NET_PRES_SKT_ERROR_T NET_PRES_SocketLastError(NET_PRES_SKT_HANDLE_T handle);
```

Returns

A NET_PRES_SKT_ERROR_T representing the last encountered error for this socket.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.

NET_PRES_SocketOptionsGet Function

Allows the options for a socket such as, current RX/TX buffer size, etc., to be obtained.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketOptionsGet(NET_PRES_SKT_HANDLE_T handle, NET_PRES_SKT_OPTION_TYPE option, void*
optParam);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Various options can be obtained at the socket level. This function calls directly to the transport layer's OptionGet function, if it exists.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The socket to set options for.
option	The specific option to set, this is passed unaltered to the transport layer.
optParam	The option value, which is passed unaltered to the transport layer.

NET_PRES_SocketOptionsSet Function

Allows setting options to a socket like adjust RX/TX buffer size, etc.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketOptionsSet(NET_PRES_SKT_HANDLE_T handle, NET_PRES_SKT_OPTION_TYPE option, void*
optParam);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Various options can be set at the socket level. This function calls directly to the transport layer's OptionSet function, if it exists.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The socket to set options for.
option	The specific option to be set, this is passed unaltered to the transport layer.
optParam	The option value, this is passed unaltered to the transport layer.

NET_PRES_SocketPeek Function

Reads a specified number of data bytes from the RX buffer/FIFO without removing them from the buffer.

File

net_pres_socketapi.h

C

```
uint16_t NET_PRES_SocketPeek(NET_PRES_SKT_HANDLE_T handle, void * buffer, uint16_t size);
```

Description

If the socket is encrypted this function will call the encryption provider's peek function. Otherwise this function calls the transport layer's peek function.

Remarks

None

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.
buffer	Destination to write the peeked data bytes.
size	Length of bytes to peek from the RX FIFO and copy to the buffer.

NET_PRES_SocketRead Function

Reads an array of data bytes from a socket's RX buffer/FIFO.

File

net_pres_socketapi.h

C

```
uint16_t NET_PRES_SocketRead(NET_PRES_SKT_HANDLE_T handle, void * buffer, uint16_t size);
```

Returns

The number of bytes read from the socket. If less than len, the RX FIFO buffer became empty or the socket is not connected.

Description

This function reads an array of data bytes from a socket's RX buffer/FIFO. The data is removed from the FIFO in the process. If the connection is encrypted this function calls the encryption provider's read function, otherwise it calls the transport layer's read function.

Remarks

For encrypted connections, a null buffer is an invalid parameter. For non encrypted connections if the supplied buffer is null, the data is simply discarded.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.
buffer	The pointer to the array to store data that was read.
size	The number of bytes to be read.

NET_PRES_SocketReadIsReady Function

Determines how many bytes can be read from the RX buffer.

File

net_pres_socketapi.h

C

```
uint16_t NET_PRES_SocketReadIsReady(NET_PRES_SKT_HANDLE_T handle);
```

Returns

The number of bytes available to be read from the TCP RX buffer.

Description

Call this function to determine how many bytes can be read from the RX buffer. If this function returns zero, the application must return to the main stack loop before continuing in order to wait for more data to arrive. This function calls the transport layer's ReadIsReady function. When using an encrypted connection the number of unencrypted bytes may turn out to be different than what this function returns.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.

NET_PRES_SocketRemoteBind Function

Binds a socket to a remote local address.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketRemoteBind(NET_PRES_SKT_HANDLE_T handle, NET_PRES_SKT_ADDR_T addrType,
NET_PRES_SKT_PORT_T port, NET_PRES_ADDRESS * addr);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This function calls directly to the transport layer's remote bind function.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The socket to bind.
addType	The type of address being used. This is passed unaltered to the transport layer.

port	The port to use. This is passed unaltered to the transport layer.
addr	The address to bind to. This is passed unaltered to the transport layer.

NET_PRES_SocketSignalHandlerDeregister Function

Deregisters a previously registered socket signal handler.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketSignalHandlerDeregister(NET_PRES_SKT_HANDLE_T handle, NET_PRES_SIGNAL_HANDLE hSig);
```

Returns

- true - If the call succeeds
- false - If no such handler is registered

Description

This function calls the transport layer's deregister signal handler function, if it exists

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.
hSig	A handle returned by a previous call to TCPIP_TCP_SignalHandlerRegister.

NET_PRES_SocketSignalHandlerRegister Function

Registers a socket signal handler.

File

net_pres_socketapi.h

C

```
NET_PRES_SIGNAL_HANDLE NET_PRES_SocketSignalHandlerRegister(NET_PRES_SKT_HANDLE_T handle, uint16_t sigMask,
NET_PRES_SIGNAL_FUNCTION handler, const void* hParam);
```

Returns

- valid handle - Indicates the call succeeded
- null handle - Indicates the call failed (null handler, no such socket, existent handler)

Description

This function calls the transport layer's register signal handle function directly, if it exists

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen

Parameters

Parameters	Description
handle	The presentation layer socket handle.
sigMask	The mask of signals to be reported, this parameter is passed to the transport layer directly.
handler	signal handler to be called when an event occurs. This parameter is passed to the transport layer directly.
hParam	Parameter to be used in the handler call. This parameter is passed to the transport layer directly.

NET_PRES_SocketWasReset Function

Self-clearing semaphore indicating socket reset.

File

net_pres_socketapi.h

C

```
bool NET_PRES_SocketWasReset(NET_PRES_SKT_HANDLE_T handle);
```

Description

This function is a self-clearing semaphore indicating whether or not a socket has been disconnected since the previous call. This function calls directly to the transport layer's IsConnected function, if it exists.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.

NET_PRES_SocketWrite Function

Takes a buffer and sends it to the encryption provider.

File

net_pres_socketapi.h

C

```
uint16_t NET_PRES_SocketWrite(NET_PRES_SKT_HANDLE_T handle, const void * buffer, uint16_t size);
```

Returns

The number of bytes written to the socket. If less than len, the buffer became full or the socket is not connected.

Description

This function takes a buffer and sends it to the encryption provider for an encrypted socket, or to the transport layer directly for an unencrypted socket.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.
buffer	The pointer to the array to be written.
size	The number of bytes to be written.

NET_PRES_SocketGetTransportHandle Function

This function returns the transport layer handle.

Details: This function returns the transport layer handle for a valid socket

File

net_pres_socketapi.h

C

```
NET_PRES_SKT_HANDLE_T NET_PRES_SocketGetTransportHandle(NET_PRES_SKT_HANDLE_T handle);
```

Returns

A valid transport layer handle that can be casted into the proper type.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	The presentation layer socket handle.

NET_PRES_SocketWriteIsReady Function

Determines how much free space is available in the TX buffer.

File

net_pres_socketapi.h

C

```
uint16_t NET_PRES_SocketWriteIsReady(NET_PRES_SKT_HANDLE_T handle, uint16_t reqSize, uint16_t minSize);
```

Returns

The number of bytes available in the TX buffer:

- >= reqSize - If the requested space is available in the output buffer
- >= minSize - I there's at least this minimum space (minSize != 0)
- 0 - Requested (minimum) space cannot be granted

Description

This function calls the transport or the encryption layer's WriteIsReady, if it exists.

Preconditions

A socket needs to have been opened by NET_PRES_SocketOpen.

Parameters

Parameters	Description
handle	Presentation layer socket handle.
reqSize	Write size to check for.
minSize	Minimum size that could be guaranteed. Could be '0' if not needed.

c) Certificate Store Functions

NET_PRES_CertStoreGetCACerts Function

This function gets the CA certificates from the store,

Implementation: Dynamic

File

net_pres_certstore.h

C

```
bool NET_PRES_CertStoreGetCACerts(const uint8_t ** certPtr, int32_t * certSize, uint8_t certIndex);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Get CA Certificates function

This function is used by client connections to retrieve the Certificate Authority certificates that are used to validate signatures on server certificates.

Preconditions

None.

Parameters

Parameters	Description
certPtr	A pointer to the CA certificates
certSize	The size of the certificates
certIndex	Most likely '0', but this parameter is provided to select a different set of CA certificates

NET_PRES_CertStoreGetServerCert Function

This function gets a server certificate and key from the certificate store.

Implementation: Dynamic

File

net_pres_certstore.h

C

```
bool NET_PRES_CertStoreGetServerCert(const uint8_t ** serverCertPtr, int32_t * serverCertSize, const
uint8_t ** serverKeyPtr, int32_t * serverKeySize, uint8_t certIndex);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Get Server Certificate and Key function

This function is used by server connections to retrieve their certificate and private key. Multiple server certificates can be stored in the certificate store, for example one for a Web server and one for a mail server.

Preconditions

None.

Parameters

Parameters	Description
serverCertPtr	A pointer to the server certificate
serverCertSize	The size of the server certificate
serverKeyPtr	A pointer to the server private key
serverKeySize	The size of the server private key
certIndex	Most likely '0', but this parameter is provided to select a different server certificate

d) Data Types and Constants

NET_PRES_ADDRESS Structure

File

net_pres_socketapi.h

C

```
typedef struct {
  uint8_t addr[16];
} NET_PRES_ADDRESS;
```

Description

This is type NET_PRES_ADDRESS.

NET_PRES_EncProviderConnect Type

Connects the function to the provider.

Implementation: Dynamic

File

net_pres_encryptionproviderapi.h

C

```
typedef NET_PRES_EncSessionStatus (* NET_PRES_EncProviderConnect)(void * providerData);
```

Returns

- NET_PRES_ENC_SS_CLIENT_NEGOTIATING - Client is still negotiating the connection
- NET_PRES_ENC_SS_SERVER_NEGOTIATING - Server is still negotiating the connection
- NET_PRES_ENC_SS_OPEN - Negotiation is complete and data can be securely transmitted
- NET_PRES_ENC_SS_FAILED - Negotiation failed

Description

Presentation Encryption Provider Connect Prototype

This function is used by the presentation layer to pump the encryption negotiation. While negotiation is ongoing, the presentation layer's task function will continue to call the function until negotiation ends.

Preconditions

A connection must have already been created.

Parameters

Parameters	Description
providerData	A pointer to the buffer that keeps the providerData returned from the Open call.

NET_PRES_EncProviderConnectionClose Type

Defines the close function to the provider.

Implementation: Dynamic

File

net_pres_encryptionproviderapi.h

C

```
typedef NET_PRES_EncSessionStatus (* NET_PRES_EncProviderConnectionClose)(void * providerData);
```

Returns

- NET_PRES_ENC_SS_CLOSING - Connection is closing, function must be called again to pump the close
- NET_PRES_ENC_SS_CLOSED - The connection is closed and can be cleaned up

Description

Presentation Encryption Provider Close Function Pointer Prototype

This function pointer defines the close function. It is called by the Networking Presentation Layer after a connection has been closed by the client.

Preconditions

A connection must have already been created.

Parameters

Parameters	Description
providerData	A pointer to the buffer for the provider to keep connection specific data.

NET_PRES_EncProviderDeinit Type

Defines the deinitialization function for the provider.

Implementation: Dynamic

File

net_pres_encryptionproviderapi.h

C

```
typedef bool (* NET_PRES_EncProviderDeinit)();
```

Returns

- true - Deinitialization succeeded
- false - Deinitialization did not succeed

Description

Presentation Encryption Provider Close Function Pointer Prototype

This function pointer prototype defines the deinitialization function for the provider.

Preconditions

None.

NET_PRES_EncProviderInit Type

Defines the initialization function to the encryption provider.

Implementation: Dynamic

File

net_pres_encryptionproviderapi.h

C

```
typedef bool (* NET_PRES_EncProviderInit)(struct _NET_PRES_TransportObject * transObject);
```

Returns

- true - Initialization succeeded
- false - Initialization did not succeed

Description

Presentation Encryption Provider Initialization Function Pointer Prototype

This function pointer prototype defines the initialization function to the encryption provider.

Preconditions

None.

Parameters

Parameters	Description
transObject	This is a copy of the structure the transport layer provides to the presentation layer to read and write data.

NET_PRES_EncProviderIsInitialized Type

Determines whether the encryption provider has been initialized.

Implementation: Dynamic

File

net_pres_encryptionproviderapi.h

C

```
typedef bool (* NET_PRES_EncProviderIsInitialized)();
```

Returns

- true - The provider has been initialized
- false - The provider has not been initialized

Description

Presentation Encryption Provider Is Initialized Pointer Prototype

This function pointer determines whether the encryption provider has been initialized and informs the presentation layer.

Preconditions

A connection must have already been created, and be in the open state.

NET_PRES_EncProviderObject Structure

Defines the data that the presentation layer needs from the provider.

File

net_pres_encryptionproviderapi.h

C

```
typedef struct _NET_PRES_EncProviderObject {
  NET_PRES_EncProviderInit fpInit;
  NET_PRES_EncProviderDeinit fpDeinit;
  NET_PRES_EncProviderOpen fpOpen;
  NET_PRES_EncProviderConnect fpConnect;
  NET_PRES_EncProviderConnectionClose fpClose;
  NET_PRES_EncProviderWrite fpWrite;
  NET_PRES_EncProviderWriteReady fpWriteReady;
  NET_PRES_EncProviderRead fpRead;
  NET_PRES_EncProviderReadReady fpReadReady;
  NET_PRES_EncProviderRead fpPeek;
  NET_PRES_EncProviderIsInitialized fpIsInited;
} NET_PRES_EncProviderObject;
```

Members

Members	Description
NET_PRES_EncProviderInit fpInit;	Function pointer to open/initialize the provider
NET_PRES_EncProviderDeinit fpDeinit;	Function pointer to close/deinitialize the provider
NET_PRES_EncProviderOpen fpOpen;	Function pointer to create a stream client connection
NET_PRES_EncProviderConnect fpConnect;	Function pointer to connect and pump the negotiation of a stream client connection
NET_PRES_EncProviderConnectionClose fpClose;	Function Pointer to close and clean up a connection
NET_PRES_EncProviderWrite fpWrite;	Function Pointer to write data to a connection
NET_PRES_EncProviderWriteReady fpWriteReady;	Function Pointer to check the connection write space
NET_PRES_EncProviderRead fpRead;	Function pointer to read data from a connection
NET_PRES_EncProviderReadReady fpReadReady;	Function pointer to return the available read data from a connection
NET_PRES_EncProviderRead fpPeek;	Function pointer to peek at data from a connection
NET_PRES_EncProviderIsInitialized fpIsInited;	Function pointer to check to determine if the provider has been initialized

Description

Presentation Encryption Provider Information Structure

This data type is given to the presentation layer during initialization to provide information on the provider, so it can be used during secure communications.

Remarks

None.

NET_PRES_EncProviderOpen Type

Defines the open connection function to the provider.

Implementation: Dynamic

File

net_pres_encryptionproviderapi.h

C

```
typedef bool (* NET_PRES_EncProviderOpen)(uintptr_t transHandle, void * providerData);
```

Returns

- true - Create succeeded
- false - Create did not succeed

Description

Presentation Encryption Provider Open Connection Prototype

This function pointer prototype defines the open connection function to the provider.

Preconditions

None.

Parameters

Parameters	Description
transHandle	The handle from the transport layer to use for this client.
providerData	A pointer to the buffer for the provider to keep connection specific data.

NET_PRES_EncProviderRead Type

Defines the read function to the provider

Implementation: Dynamic

File

net_pres_encryptionproviderapi.h

C

```
typedef int32_t (* NET_PRES_EncProviderRead)(void * providerData, uint8_t * buffer, uint16_t size);
```

Returns

The number of bytes transferred.

Description

Presentation Encryption Provider Read Function Pointer Prototype

This function pointer defines the read function. It is called by the presentation layer when the presentation client wants to read from a secured connection.

Remarks

If the supplied buffer is NULL the operation is ignored.

Preconditions

A connection must have already been created, and be in the open state.

Parameters

Parameters	Description
providerData	A pointer to the buffer for the provider to keep connection specific data.
buffer	A pointer to the buffer that will be read from the provider.
count	Size of the buffer.

NET_PRES_EncProviderWrite Type

Defines the write function to the provider.

Implementation: Dynamic

File

net_pres_encryptionproviderapi.h

C

```
typedef int32_t (* NET_PRES_EncProviderWrite)(void * providerData, const uint8_t * buffer, uint16_t size);
```

Returns

The number of bytes transferred.

Description

Presentation Encryption Provider Write Function Pointer Prototype

This function pointer defines the write function. It is called by the presentation layer when the application wants to write to a secured connection.

Preconditions

A connection must have already been created, and be in the open state.

Parameters

Parameters	Description
providerData	A pointer to the buffer for the provider to keep connection specific data.
buffer	This is a pointer to the buffer that will be sent to the provider.
size	This is the size of the buffer.

NET_PRES_EncSessionStatus Enumeration

Defines the enumeration for the state and status of the encrypted portion of a connection.

File

net_pres_encryptionproviderapi.h

C

```
typedef enum {
  NET_PRES_ENC_SS_UNKNOWN,
  NET_PRES_ENC_SS_WAITING_TO_START_NEGOTIATION,
  NET_PRES_ENC_SS_CLIENT_NEGOTIATING,
  NET_PRES_ENC_SS_SERVER_NEGOTIATING,
  NET_PRES_ENC_SS_OPEN,
  NET_PRES_ENC_SS_FAILED,
  NET_PRES_ENC_SS_CLOSING,
  NET_PRES_ENC_SS_CLOSED
} NET_PRES_EncSessionStatus;
```

Members

Members	Description
NET_PRES_ENC_SS_UNKNOWN	Presentation encryption is in an unknown/default state
NET_PRES_ENC_SS_WAITING_TO_START_NEGOTIATION	Presentation encryption has not started negotiation
NET_PRES_ENC_SS_CLIENT_NEGOTIATING	Presentation encryption client negotiation is in progress
NET_PRES_ENC_SS_SERVER_NEGOTIATING	Presentation encryption server negotiation is in progress
NET_PRES_ENC_SS_OPEN	Presentation encryption negotiation is complete and data can be sent/received
NET_PRES_ENC_SS_FAILED	Presentation encryption negotiation failed or some other failure
NET_PRES_ENC_SS_CLOSING	Presentation encryption is closing, but connection needs to be pumped for final packets
NET_PRES_ENC_SS_CLOSED	Presentation encryption is closed, provider data has been freed

Description

MPLAB Harmony Networking Presentation Layer Encryption status type

This enumeration defines the enumeration for the state and status of the encrypted portion of a connection.

Remarks

None.

NET_PRES_INDEX Type

Sets the type for the presentation layer index.

File

net_pres.h

C

```
typedef uint8_t NET_PRES_INDEX;
```

Description

Net Presentation Index Type

This data type sets the type for the presentation layer index.

Remarks

None.

NET_PRES_INIT_DATA Structure

Initializes a Presentation layer.

File

net_pres.h

C

```
typedef struct {
  uint8_t numLayers;
  const NET_PRES_INST_DATA * pInitData;
} NET_PRES_INIT_DATA;
```

Members

Members	Description
uint8_t numLayers;	Number of presentation layers
const NET_PRES_INST_DATA * pInitData;	Pointer to an array of pointers to presentation layer instance data.

Description

Net Presentation Initialization data

Data type that initializes a Presentation layer.

Remarks

None.

NET_PRES_INST_DATA Structure

Initializes a Presentation layer.

File

net_pres.h

C

```
typedef struct {
  const struct _NET_PRES_TransportObject * pTransObject_ss;
  const struct _NET_PRES_TransportObject * pTransObject_sc;
  const struct _NET_PRES_TransportObject * pTransObject_ds;
  const struct _NET_PRES_TransportObject * pTransObject_dc;
  const struct _NET_PRES_EncProviderObject * pProvObject_ss;
  const struct _NET_PRES_EncProviderObject * pProvObject_sc;
  const struct _NET_PRES_EncProviderObject * pProvObject_ds;
  const struct _NET_PRES_EncProviderObject * pProvObject_dc;
} NET_PRES_INST_DATA;
```

Members

Members	Description
const struct _NET_PRES_TransportObject * pTransObject_ss;	Pointer to the transport object that handles the stream server

const struct _NET_PRES_TransportObject * pTransObject_sc;	Pointer to the transport object that handles the stream client
const struct _NET_PRES_TransportObject * pTransObject_ds;	Pointer to the transport object that handles the datagram server
const struct _NET_PRES_TransportObject * pTransObject_dc;	Pointer to the transport object that handles the datagram client
const struct _NET_PRES_EncProviderObject * pProvObject_ss;	Pointer to the encryption provider object that handles the stream server
const struct _NET_PRES_EncProviderObject * pProvObject_sc;	Pointer to the encryption provider object that handles the stream client
const struct _NET_PRES_EncProviderObject * pProvObject_ds;	Pointer to the encryption provider object that handles the datagram server
const struct _NET_PRES_EncProviderObject * pProvObject_dc;	Pointer to the encryption provider object that handles the datagram client

Description

Net Presentation Instance Initialization data

This data type initializes a Presentation layer.

Remarks

None.

NET_PRES_SIGNAL_FUNCTION Type

MPLAB Harmony Networking Presentation Layer Signal function.

File

net_pres.h

C

```
typedef void (* NET_PRES_SIGNAL_FUNCTION)(NET_PRES_SKT_HANDLE_T handle, NET_PRES_SIGNAL_HANDLE hNet,
uint16_t sigType, const void* param);
```

Description

Type: NET_PRES_SIGNAL_FUNCTION

Prototype of a signal handler. Socket user can register a handler for the socket. Once an event occurs the registered handler will be called.

Remarks

The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

Parameters

Parameters	Description
handle	The presentation socket to be used
hNet	The network interface on which the event has occurred
sigType	The type of signal that has occurred
param	An additional parameter that can has been specified at the handler registration call. Currently not used and it will be null.

NET_PRES_SIGNAL_HANDLE Type

Sets the type for the presentation layer signal handle.

File

net_pres.h

C

```
typedef const void* NET_PRES_SIGNAL_HANDLE;
```

Description

Net Presentation Signal Handle Type

This data type sets the type for the presentation layer signal handle.

Remarks

None.

NET_PRES_SKT_ADDR_T Enumeration

File

net_pres_socketapi.h

C

```
typedef enum {
  NET_PRES_SKT_ADDR_UNKNOWN
} NET_PRES_SKT_ADDR_T;
```

Description

This is type NET_PRES_SKT_ADDR_T.

NET_PRES_SKT_ERROR_T Enumeration

File

net_pres_socketapi.h

C

```
typedef enum {
  NET_PRES_SKT_OK = 0,
  NET_PRES_SKT_OP_NOT_SUPPORTED = -1,
  NET_PRES_SKT_OP_OUT_OF_HANDLES = -2,
  NET_PRES_SKT_OP_INVALID_INDEX = -3,
  NET_PRES_SKT_UNKNOWN_ERROR = -4,
  NET_PRES_SKT_INVALID_SOCKET = -5
} NET_PRES_SKT_ERROR_T;
```

Members

Members	Description
NET_PRES_SKT_OP_NOT_SUPPORTED = -1	Most likely the function is not supported by the socket type

Description

This is type NET_PRES_SKT_ERROR_T.

NET_PRES_SKT_HANDLE_T Type

Sets the type for the presentation layer socket handle.

File

net_pres.h

C

```
typedef int16_t NET_PRES_SKT_HANDLE_T;
```

Description

Net Presentation Socket Handle Type

This data type sets the type for the presentation layer socket handle.

Remarks

None.

NET_PRES_SKT_OPTION_TYPE Enumeration

File

net_pres_socketapi.h

C

```
typedef enum {
  NET_PRES_SKT_OPT_UNKNOWN
} NET_PRES_SKT_OPTION_TYPE;
```

Description

This is type NET_PRES_SKT_OPTION_TYPE.

NET_PRES_SKT_T Enumeration

File

net_pres_socketapi.h

C

```
typedef enum {
  NET_PRES_SKT_CLIENT = 0x0001,
  NET_PRES_SKT_SERVER = 0x0002,
  NET_PRES_SKT_STREAM = 0x0004,
  NET_PRES_SKT_DATAGRAM = 0x0008,
  NET_PRES_SKT_UNENCRYPTED = 0x0010,
  NET_PRES_SKT_ENCRYPTED = 0x0020,
  NET_PRES_SKT_UNENCRYPTED_STREAM_CLIENT = (NET_PRES_SKT_UNENCRYPTED | NET_PRES_SKT_STREAM |
NET_PRES_SKT_CLIENT),
  NET_PRES_SKT_UNENCRYPTED_STREAM_SERVER = (NET_PRES_SKT_UNENCRYPTED | NET_PRES_SKT_STREAM |
NET_PRES_SKT_SERVER),
  NET_PRES_SKT_UNENCRYPTED_DATAGRAM_CLIENT = (NET_PRES_SKT_UNENCRYPTED | NET_PRES_SKT_DATAGRAM |
NET_PRES_SKT_CLIENT),
  NET_PRES_SKT_UNENCRYPTED_DATAGRAM_SERVER = (NET_PRES_SKT_UNENCRYPTED | NET_PRES_SKT_DATAGRAM |
NET_PRES_SKT_SERVER),
  NET_PRES_SKT_ENCRYPTED_STREAM_CLIENT = (NET_PRES_SKT_ENCRYPTED | NET_PRES_SKT_STREAM |
NET_PRES_SKT_CLIENT),
  NET_PRES_SKT_ENCRYPTED_STREAM_SERVER = (NET_PRES_SKT_ENCRYPTED | NET_PRES_SKT_STREAM |
NET_PRES_SKT_SERVER),
  NET_PRES_SKT_ENCRYPTED_DATAGRAM_CLIENT = (NET_PRES_SKT_ENCRYPTED | NET_PRES_SKT_DATAGRAM |
NET_PRES_SKT_CLIENT),
  NET_PRES_SKT_ENCRYPTED_DATAGRAM_SERVER = (NET_PRES_SKT_ENCRYPTED | NET_PRES_SKT_DATAGRAM |
NET_PRES_SKT_SERVER),
  NET_PRES_SKT_DEFAULT_STREAM_CLIENT = (NET_PRES_SKT_STREAM | NET_PRES_SKT_CLIENT),
  NET_PRES_SKT_DEFAULT_STREAM_SERVER = (NET_PRES_SKT_STREAM | NET_PRES_SKT_SERVER),
  NET_PRES_SKT_DEFAULT_DATAGRAM_CLIENT = (NET_PRES_SKT_DATAGRAM | NET_PRES_SKT_CLIENT),
  NET_PRES_SKT_DEFAULT_DATAGRAM_SERVER = (NET_PRES_SKT_DATAGRAM | NET_PRES_SKT_SERVER)
} NET_PRES_SKT_T;
```

Description

This is type NET_PRES_SKT_T.

NET_PRES_TRANS_ADDR_T Structure

Defines a generic address structure to pass to the transport layer.

File

net_pres_transportapi.h

C

```
typedef struct {
  uint8_t addr[16];
} NET_PRES_TRANS_ADDR_T;
```

Members

Members	Description
uint8_t addr[16];	So far biggest for IPv6

Description

MPLAB Harmony Networking Presentation Transport Address Structure

This data type is just a generic address structure. The presentation layer does not do any processing on this data, but instead passes it directly to

the transport.

Remarks

None.

NET_PRES_TRANS_ADDRESS_TYPE Enumeration

Defines the enumeration for the type of address.

File

net_pres_transportapi.h

C

```
typedef enum {
  NET_PRES_ADDRT_UNKNOWN
} NET_PRES_TRANS_ADDRESS_TYPE;
```

Description

MPLAB Harmony Networking Presentation Layer Address type

Defines the enumeration for the type of address. This enumeration is not used directly by the presentation layer and is used to enforce a consistent interface between layers.

Remarks

None.

NET_PRES_TRANS_OPTION_T Enumeration

Defines the enumeration for the type of options.

File

net_pres_transportapi.h

C

```
typedef enum {
  NET_PRES_OPT_UNKNOWN
} NET_PRES_TRANS_OPTION_T;
```

Description

MPLAB Harmony Networking Presentation Layer Option type

Defines the enumeration for the type of options. This enumeration is not used directly by the presentation layer and is used to enforce a consistent interface between layers.

Remarks

None.

NET_PRES_TransBind Type

Binds a socket to a local address.

File

net_pres_transportapi.h

C

```
typedef bool (* NET_PRES_TransBind)(NET_PRES_SKT_HANDLE_T handle, NET_PRES_TRANS_ADDRESS_TYPE addType,
NET_PRES_SKT_PORT_T port, NET_PRES_ADDRESS * address);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Transport Layer Bind Function Pointer Prototype

This function is called by the presentation layer when an application wants to bind a socket.

Preconditions

A socket needs to have been opened by NET_PRES_TransOpen.

Parameters

Parameters	Description
handle	The handle returned from NET_PRES_TransOpen.
addType	The type of address being used. This is passed unaltered to the transport layer.
port	The port to use. This is passed unaltered to the transport layer.
address	The address to bind to. This is passed unaltered to the transport layer.

NET_PRES_TransBool Type

Generic function prototype for functions that return a bool.

File

net_pres_transportapi.h

C

```
typedef bool (* NET_PRES_TransBool)(NET_PRES_SKT_HANDLE_T handle);
```

Returns

The result is passed directly through from the transport layer to the application. The meaning of the return is dependent on the transport function

Description

Transport Layer Boolean Function Pointer Prototype

This function is called by the presentation layer when it accesses a function that takes no parameters apart from the socket handle and returns a boolean.

Preconditions

A socket needs to have been opened by NET_PRES_TransOpen.

Parameters

Parameters	Description
handle	The handle returned from NET_PRES_TransOpen.

NET_PRES_TransClose Type

Function prototype for functions that closes a socket.

File

net_pres_transportapi.h

C

```
typedef void (* NET_PRES_TransClose)(NET_PRES_SKT_HANDLE_T handle);
```

Returns

None.

Description

Transport Layer Close Function Pointer Prototype

This function is called by the presentation layer when the application wants to close a connection.

Preconditions

A socket needs to have been opened by NET_PRES_TransOpen.

Parameters

Parameters	Description
handle	The handle returned from NET_PRES_TransOpen.

NET_PRES_TransDiscard Type

Function prototype for functions that clears a socket's RX buffer.

File

net_pres_transportapi.h

C

```
typedef uint16_t (* NET_PRES_TransDiscard)(NET_PRES_SKT_HANDLE_T handle);
```

Returns

The number of bytes discarded.

Description

Transport Layer Discard Function Pointer Prototype

This function is called by the presentation layer when the application wants to discard the RX buffer in a socket.

Preconditions

A socket needs to have been opened by NET_PRES_TransOpen.

Parameters

Parameters	Description
handle	The handle returned from NET_PRES_TransOpen.

NET_PRES_TransHandlerRegister Type

Function prototype that registers a handler with a socket.

File

net_pres_transportapi.h

C

```
typedef NET_PRES_SIGNAL_HANDLE (* NET_PRES_TransHandlerRegister)(NET_PRES_SKT_HANDLE_T handle, uint16_t
sigMask, NET_PRES_SIGNAL_FUNCTION handler, const void* hParam);
```

Returns

The handle of a signal handler.

Description

Transport Layer Register Handler Function Pointer Prototype

This function is called by the presentation layer when the application wants to register a handler function.

Preconditions

A socket needs to have been opened by NET_PRES_TransOpen.

Parameters

Parameters	Description
handle	The handle returned from NET_PRES_TransOpen.
sigMask	The event mask.
handler	The event handler function.
hParam	Parameters passed to the handler function.

NET_PRES_TransOpen Type

Opens a presentation socket.

File

net_pres_transportapi.h

C

```
typedef NET_PRES_SKT_HANDLE_T (* NET_PRES_TransOpen)(NET_PRES_TRANS_ADDRESS_TYPE addType,
NET_PRES_SKT_PORT_T port, NET_PRES_ADDRESS * address);
```

Returns

- NET_PRES_INVALID_SOCKET - No sockets of the specified type were available to be opened.
- NET_PRES_SKT_HANDLE_T handle - Returned when NET_PRES_INVALID_SOCKET is returned. Save this handle and use it when calling all other presentation socket APIs.

Description

Transport Layer Open Function Pointer Prototype

This function is called by the presentation layer when an application wants to open a socket.

Preconditions

Transport layer must be initialized.

Parameters

Parameters	Description
addType	The type of address being used. This is passed unaltered to the transport layer.
port	The port to listen or to send to. This is passed unaltered to the transport layer.
address	The address to use. This is passed unaltered to the transport layer.

NET_PRES_TransOption Type

Sets of gets a socket's options.

File

net_pres_transportapi.h

C

```
typedef bool (* NET_PRES_TransOption)(NET_PRES_SKT_HANDLE_T handle, NET_PRES_TRANS_OPTION_T option, void *
optParam);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Transport Layer Option Function Pointer Prototype

This function is called by the presentation layer when an application wants to get the current socket options or set them.

Preconditions

A socket needs to have been opened by NET_PRES_TransOpen.

Parameters

Parameters	Description
handle	The handle returned from NET_PRES_TransOpen.
option	The option to set or get.
optParam	The pointer to option specific information.

NET_PRES_TransPeek Type

Function prototype for functions that peeks on the socket's buffer.

File

net_pres_transportapi.h

C

```
typedef bool (* NET_PRES_TransPeek)(NET_PRES_SKT_HANDLE_T handle, uint8_t *vBuffer, uint16_t wLen, uint16_t
wStart);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Transport Layer Peek Function Pointer Prototype

This function is called by the presentation layer when the application wants to peek into the buffer of an unencrypted socket.

Preconditions

A socket needs to have been opened by NET_PRES_TransOpen.

Parameters

Parameters	Description
handle	The handle returned from NET_PRES_TransOpen.
vBuffer	The buffer location to put the information.
wLen	The size of the buffer.
wStart	Where to start peeking into the buffer. This parameter is not used and will always be set to '0'.

NET_PRES_TransportObject Structure

Defines the data that the transport layer needs to provide to the Networking Presentation Layer.

File

net_pres_transportapi.h

C

```
typedef struct _NET_PRES_TransportObject {
  NET_PRES_TransOpen fpOpen;
  NET_PRES_TransBind fpLocalBind;
  NET_PRES_TransBind fpRemoteBind;
  NET_PRES_TransOption fpOptionGet;
  NET_PRES_TransOption fpOptionSet;
  NET_PRES_TransBool fpIsConnected;
  NET_PRES_TransBool fpWasReset;
  NET_PRES_TransBool fpDisconnect;
  NET_PRES_TransBool fpConnect;
  NET_PRES_TransClose fpClose;
  NET_PRES_TransSocketInfoGet fpSocketInfoGet;
  NET_PRES_TransBool fpFlush;
  NET_PRES_TransPeek fpPeek;
  NET_PRES_TransDiscard fpDiscard;
  NET_PRES_TransHandlerRegister fpHandlerRegister;
  NET_PRES_TransSignalHandlerDeregister fpHandlerDeregister;
  NET_PRES_TransRead fpRead;
  NET_PRES_TransWrite fpWrite;
  NET_PRES_TransReady fpReadyToRead;
  NET_PRES_TransReady fpReadyToWrite;
  NET_PRES_TransIsPortDefaultSecured fpIsPortDefaultSecure;
} NET_PRES_TransportObject;
```

Members

Members	Description
NET_PRES_TransOpen fpOpen;	Function pointer to the transport's open call
NET_PRES_TransBind fpLocalBind;	Function pointer to the transport's bind call
NET_PRES_TransBind fpRemoteBind;	Function pointer to the transport's remote bind call
NET_PRES_TransOption fpOptionGet;	Function call to the the transport's option get call
NET_PRES_TransOption fpOptionSet;	Function call to the the transport's option set call
NET_PRES_TransBool fpIsConnected;	Function call to the the transport's is connected call
NET_PRES_TransBool fpWasReset;	Function call to the the transport's was reset call
NET_PRES_TransBool fpDisconnect;	Function call to the the transport's disconnect call
NET_PRES_TransBool fpConnect;	Function call to the the transport's connect call
NET_PRES_TransClose fpClose;	Function call to the the transport's close call

NET_PRES_TransSocketInfoGet fpSocketInfoGet;	Function call to the the transport's get socket info call
NET_PRES_TransBool fpFlush;	Function call to the the transport's flush call
NET_PRES_TransPeek fpPeek;	Function call to the the transport's peek call
NET_PRES_TransDiscard fpDiscard;	Function call to the the transport's discard call
NET_PRES_TransHandlerRegister fpHandlerRegister;	Function call to the the transport's register handler call
NET_PRES_TransSignalHandlerDeregister fpHandlerDeregister;	Function call to the the transport's deregister handler call
NET_PRES_TransRead fpRead;	Function pointer to call when doing a read from a transport layer
NET_PRES_TransWrite fpWrite;	Function pointer to call when doing a write to a transport layer
NET_PRES_TransReady fpReadyToRead;	Function pointer to call when checking to see if there is data available to be read from a transport layer
NET_PRES_TransReady fpReadyToWrite;	Function pointer to call when checking to see if there is space available to be write to a transport layer
NET_PRES_TransIsPortDefaultSecured fpIsPortDefaultSecure;	Function pointer to call when checking to see if a port is secure by default

Description

MPLAB Harmony Networking Presentation Transport Information Structure

This data type defines the data required by the transport layer to effectively work with the Networking Presentation Layer. The data is there to allow the Networking Presentation Layer to configure the provider to effectively use the transport layer.

Remarks

None.

NET_PRES_TransRead Type

Defines the read function provided by the transport layer.

Implementation: Dynamic

File

net_pres_transportapi.h

C

```
typedef uint16_t (* NET_PRES_TransRead)(uintptr_t transHandle, uint8_t* buffer, uint16_t count);
```

Returns

The number of data bytes copied by the transport channel into the buffer.

Description

Presentation Layer Transport Layer Read Function Pointer Prototype

This function prototype is used to define the function that the Networking Presentation Layer will pass to the provider when it is initialized. The provider will use this function when it needs to read from the transport layer.

Preconditions

None.

Parameters

Parameters	Description
transHandle	This is the transport layer handle provided by the transport layer when a communications channel is open.
buffer	This is a pointer to the buffer that the transport layer will copy data to.
count	This is the size of the buffer.

NET_PRES_TransReady Type

Defines the ready function provided by the transport layer.

Implementation: Dynamic

File

net_pres_transportapi.h

C

```
typedef uint16_t (* NET_PRES_TransReady)(uintptr_t transHandle);
```

Returns

- true - The presentation layer can read or write to the transport layer
- false - The transport layer is busy and cannot accept reads or write

Description

Presentation Layer Transport Layer Ready Function

This function prototype is used to define the function that the Networking Presentation Layer will pass to the provider when it is initialized. The provider will use this function when it needs to check if it can read or write to the layer.

Preconditions

None.

Parameters

Parameters	Description
transHandle	This is the transport layer handle provided by the transport layer when a communications channel is open.

NET_PRES_TransSignalHandlerDeregister Type

Function prototype that deregisters a handler with a socket.

File

net_pres_transportapi.h

C

```
typedef bool (* NET_PRES_TransSignalHandlerDeregister)(NET_PRES_SKT_HANDLE_T handle,
NET_PRES_SIGNAL_FUNCTION hSig);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Transport Layer Deregister Handler Function Pointer Prototype

This function is called by the presentation layer when the application wants to deregister a handler function.

Preconditions

A socket needs to have been opened by NET_PRES_TransOpen.

Parameters

Parameters	Description
handle	The handle returned from NET_PRES_TransOpen.
hSig	The handler handle returned from NET_PRES_TransHandlerRegister.

NET_PRES_TransSocketInfoGet Type

Function prototype for functions that gets the information on a socket.

File

net_pres_transportapi.h

C

```
typedef bool (* NET_PRES_TransSocketInfoGet)(NET_PRES_SKT_HANDLE_T handle, void * info);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Transport Layer Get Socket Info Function Pointer Prototype

This function is called by the presentation layer when the application wants to get information on a socket.

Preconditions

A socket needs to have been opened by NET_PRES_TransOpen.

Parameters

Parameters	Description
handle	The handle returned from NET_PRES_TransOpen.
info	The socket information.

NET_PRES_TransWrite Type

Defines the write function provided by the transport layer.

Implementation: Dynamic

File

net_pres_transportapi.h

C

```
typedef uint16_t (* NET_PRES_TransWrite)(uintptr_t transHandle, const uint8_t* buffer, uint16_t count);
```

Returns

The number of data bytes accepted by the transport layer.

Description

Presentation Layer Transport Layer Write Function Pointer Prototype

This function prototype is used to define the function that the Networking Presentation Layer will pass to the provider when it is initialized. The provider will use this function when it needs to write to the transport layer.

Preconditions

None.

Parameters

Parameters	Description
transHandle	This is the transport layer handle provided by the transport layer when a communications channel is open.
buffer	This is a pointer to the buffer contains the data to be passed to the transport layer.
count	This is the size of the buffer.

NET_PRES_INVALID_SOCKET Macro

Invalid socket indicator macro.

File

net_pres.h

C

```
#define NET_PRES_INVALID_SOCKET (-1)
```

Description

Macro: NET_PRES_INVALID_SOCKET

Indicates that the socket is invalid or could not be opened.

NET_PRES_SKT_PORT_T Type

Sets the type for the presentation layer port.

File

net_pres.h

C

```
typedef uint16_t NET_PRES_SKT_PORT_T;
```

Description

Net Presentation Port Type

This data type sets the type for the presentation layer port.

Remarks

None.

NET_PRES_TransIsPortDefaultSecured Type

Checks to see if a port is encrypted by default.

Implementation: Dynamic

File

net_pres_transportapi.h

C

```
typedef bool (* NET_PRES_TransIsPortDefaultSecured)(uint16_t port);
```

Returns

- true - The port is encrypted by default and the presentation layer will start negotiating encryption when it is connected.
- false - The post is not encrypted by default.

Description

Presentation Layer Transport Layer Is Port Encrypted

This function prototype is used by the presentation layer to determine if a port is encrypted by default or not when it is opened.

Preconditions

None.

NET_PRES_EncProviderWriteReady Type

Defines the write ready function to the provider.

Implementation: Dynamic

File

net_pres_encryptionproviderapi.h

C

```
typedef uint16_t (* NET_PRES_EncProviderWriteReady)(void * providerData, uint16_t reqSize, uint16_t
minSize);
```

Returns

The number of bytes available in the output buffer:

- >= reqSize, if the requested space is available in the output buffer
- >= minSize, if there's at least this minimum space (minSize != 0)
- 0, requested (minimum) space cannot be granted

Description

Presentation Encryption Provider Write Ready Function Pointer Prototype

This function pointer defines the write ready function. It is called by the presentation layer when the application wants to check the write space to a

secured connection. The function checks for the requested size. If this is not available, it checks for at least minimum size (if != 0)

Preconditions

A connection must have already been created, and be in the open state.

Parameters

Parameters	Description
providerData	A pointer to the buffer for the provider to keep connection specific data.
reqSize	The requested size to check for.
minSize	Minimum size to check for. Could be 0, if not used.

Files

Files

Name	Description
net_pres.h	Describes the system level interface and common definitions to the MPLAB Harmony presentation layer.
net_pres_certstore.h	This file describes the standard interface for the certificate store
net_pres_socketapi.h	Describes the API for accessing presentation layer sockets.
net_pres_transportapi.h	API descriptions that the transport layers follow for the presentation layer.
net_pres_encryptionproviderapi.h	API descriptions that encryption providers follow for the presentation layer.

Description

This section lists the source and header files used by the Network Presentation Layer.

net_pres.h

Describes the system level interface and common definitions to the MPLAB Harmony presentation layer.

Functions

	Name	Description
	NET_PRES_Deinitialize	Deinitializes the Network Presentation Layer Instance. **Implementation:** Dynamic
	NET_PRES_Initialize	Initializes the Network Presentation Layer sub-system with the configuration data. **Implementation:** Dynamic
	NET_PRES_Reinitialize	Reinitializes the instance of the presentation layer. **Implementation:** Dynamic
	NET_PRES_Status	Provides the current status of the MPLAB Harmony Networking Presentation Layer. **Implementation:** Dynamic
	NET_PRES_Tasks	MPLAB Harmony tasks function used for general presentation layer tasks. **Implementation:** Dynamic

Macros

	Name	Description
	NET_PRES_INVALID_SOCKET	Invalid socket indicator macro.

Structures

	Name	Description
	NET_PRES_INIT_DATA	Initializes a Presentation layer.
	NET_PRES_INST_DATA	Initializes a Presentation layer.

Types

	Name	Description
	NET_PRES_INDEX	Sets the type for the presentation layer index.
	NET_PRES_SIGNAL_FUNCTION	MPLAB Harmony Networking Presentation Layer Signal function.
	NET_PRES_SIGNAL_HANDLE	Sets the type for the presentation layer signal handle.
	NET_PRES_SKT_HANDLE_T	Sets the type for the presentation layer socket handle.

	NET_PRES_SKT_PORT_T	Sets the type for the presentation layer port.

Description

MPLAB Harmony Networking Presentation Layer Header File

This file describes the system interface and common definitions to the MPLAB Harmony Networking Presentation Layer.

Company

Microchip Technology Inc.

Filename: net_pres.h

net_pres_certstore.h

This file describes the standard interface for the certificate store

Functions

	Name	Description
	NET_PRES_CertStoreGetCACerts	This function gets the CA certificates from the store, **Implementation:** Dynamic
	NET_PRES_CertStoreGetServerCert	This function gets a server certificate and key from the certificate store. **Implementation:** Dynamic

Description

MPLAB Harmony Networking Presentation Certificate Storage header file

This file describes the interface that the presentation layer uses to access the certificate store. The MHC can generate a read-only, Flash-based certificate store, or stubs for these functions. If a more complex certificate store is desired, for instance storing multiple certificates or being able to update certificates, the implementation may be provided by the end user.

Company

Microchip Technology Inc.

Filename: net_pres_certstore.h

net_pres_socketapi.h

Describes the API for accessing presentation layer sockets.

Enumerations

	Name	Description
	NET_PRES_SKT_ADDR_T	This is type NET_PRES_SKT_ADDR_T.
	NET_PRES_SKT_ERROR_T	This is type NET_PRES_SKT_ERROR_T.
	NET_PRES_SKT_OPTION_TYPE	This is type NET_PRES_SKT_OPTION_TYPE.
	NET_PRES_SKT_T	This is type NET_PRES_SKT_T.

Functions

	Name	Description
	NET_PRES_SocketBind	Binds a socket to a local address.
	NET_PRES_SocketClose	Disconnects an open socket and destroys the socket handle, releasing the associated resources.
	NET_PRES_SocketConnect	Connects a client socket.
	NET_PRES_SocketDiscard	Discards any pending data in the RX FIFO.
	NET_PRES_SocketDisconnect	Disconnects an open socket.
	NET_PRES_SocketEncryptSocket	This function turns an insecure socket into a secure socket. Details: This function will turn an unencrypted socket into an encrypted socket and starts encryption negotiation.
	NET_PRES_SocketFlush	Immediately transmits all pending TX data.
	NET_PRES_SocketGetTransportHandle	This function returns the transport layer handle. Details: This function returns the transport layer handle for a valid socket
	NET_PRES_SocketInfoGet	Obtains information about a currently open socket.
	NET_PRES_SocketIsConnected	Determines whether a socket has an established connection.

	NET_PRES_SocketIsNegotiatingEncryption	This function checks if encryption negotiation is still in progress.
	NET_PRES_SocketIsOpenModeSupported	Checks to see if a mode is supported by open.
	NET_PRES_SocketIsSecure	This function checks whether a connection is secure.
	NET_PRES_SocketLastError	This function returns the last error code for this socket. Details: This function will return the last error code that was set for this socket and it will clear the current error code. An error code is set whenever a socket operation fails for some missing functionality, bad parameter, etc.
	NET_PRES_SocketOpen	Opens a presentation socket.
	NET_PRES_SocketOptionsGet	Allows the options for a socket such as, current RX/TX buffer size, etc., to be obtained.
	NET_PRES_SocketOptionsSet	Allows setting options to a socket like adjust RX/TX buffer size, etc.
	NET_PRES_SocketPeek	Reads a specified number of data bytes from the RX buffer/FIFO without removing them from the buffer.
	NET_PRES_SocketRead	Reads an array of data bytes from a socket's RX buffer/FIFO.
	NET_PRES_SocketReadIsReady	Determines how many bytes can be read from the RX buffer.
	NET_PRES_SocketRemoteBind	Binds a socket to a remote local address.
	NET_PRES_SocketSignalHandlerDeregister	Deregisters a previously registered socket signal handler.
	NET_PRES_SocketSignalHandlerRegister	Registers a socket signal handler.
	NET_PRES_SocketWasReset	Self-clearing semaphore indicating socket reset.
	NET_PRES_SocketWrite	Takes a buffer and sends it to the encryption provider.
	NET_PRES_SocketWriteIsReady	Determines how much free space is available in the TX buffer.

Structures

	Name	Description
	NET_PRES_ADDRESS	This is type NET_PRES_ADDRESS.

Description

MPLAB Harmony Networking Presentation socket API header file

This file describes the API for accessing Networking Presentation Layer sockets.

Company

Microchip Technology Inc.

Filename: net_pres_socketapi.h

net_pres_transportapi.h

API descriptions that the transport layers follow for the presentation layer.

Enumerations

	Name	Description
	NET_PRES_TRANS_ADDRESS_TYPE	Defines the enumeration for the type of address.
	NET_PRES_TRANS_OPTION_T	Defines the enumeration for the type of options.

Structures

	Name	Description
	_NET_PRES_TransportObject	Defines the data that the transport layer needs to provide to the Networking Presentation Layer.
	NET_PRES_TRANS_ADDR_T	Defines a generic address structure to pass to the transport layer.
	NET_PRES_TransportObject	Defines the data that the transport layer needs to provide to the Networking Presentation Layer.

Types

	Name	Description
	NET_PRES_TransBind	Binds a socket to a local address.
	NET_PRES_TransBool	Generic function prototype for functions that return a bool.
	NET_PRES_TransClose	Function prototype for functions that closes a socket.
	NET_PRES_TransDiscard	Function prototype for functions that clears a socket's RX buffer.
	NET_PRES_TransHandlerRegister	Function prototype that registers a handler with a socket.

	NET_PRES_TransIsPortDefaultSecured	Checks to see if a port is encrypted by default. **Implementation:** Dynamic
	NET_PRES_TransOpen	Opens a presentation socket.
	NET_PRES_TransOption	Sets of gets a socket's options.
	NET_PRES_TransPeek	Function prototype for functions that peeks on the socket's buffer.
	NET_PRES_TransRead	Defines the read function provided by the transport layer. **Implementation:** Dynamic
	NET_PRES_TransReady	Defines the ready function provided by the transport layer. **Implementation:** Dynamic
	NET_PRES_TransSignalHandlerDeregister	Function prototype that deregisters a handler with a socket.
	NET_PRES_TransSocketInfoGet	Function prototype for functions that gets the information on a socket.
	NET_PRES_TransWrite	Defines the write function provided by the transport layer. **Implementation:** Dynamic

Description

MPLAB Harmony Networking Presentation Layer Header File

This file describes the API that transport layers follow for to integrate with MPLAB Harmony's Networking Presentation Layer.

Company

Microchip Technology Inc.

Filename: net_pres_transportapi.h

net_pres_encryptionproviderapi.h

API descriptions that encryption providers follow for the presentation layer.

Enumerations

	Name	Description
	NET_PRES_EncSessionStatus	Defines the enumeration for the state and status of the encrypted portion of a connection.

Structures

	Name	Description
	_NET_PRES_EncProviderObject	Defines the data that the presentation layer needs from the provider.
	NET_PRES_EncProviderObject	Defines the data that the presentation layer needs from the provider.

Types

	Name	Description
	NET_PRES_EncProviderConnect	Connects the function to the provider. **Implementation:** Dynamic
	NET_PRES_EncProviderConnectionClose	Defines the close function to the provider. **Implementation:** Dynamic
	NET_PRES_EncProviderDeinit	Defines the deinitialization function for the provider. **Implementation:** Dynamic
	NET_PRES_EncProviderInit	Defines the initialization function to the encryption provider. **Implementation:** Dynamic
	NET_PRES_EncProviderIsInitialized	Determines whether the encryption provider has been initialized. **Implementation:** Dynamic
	NET_PRES_EncProviderOpen	Defines the open connection function to the provider. **Implementation:** Dynamic
	NET_PRES_EncProviderRead	Defines the read function to the provider **Implementation:** Dynamic
	NET_PRES_EncProviderReadReady	Defines the read ready function to the provider **Implementation:** Dynamic
	NET_PRES_EncProviderWrite	Defines the write function to the provider. **Implementation:** Dynamic
	NET_PRES_EncProviderWriteReady	Defines the write ready function to the provider. **Implementation:** Dynamic

Description

MPLAB Harmony Networking Presentation Layer Encryption Provider Header File

This file describes the API that encryption providers follow for the Networking Presentation Layer.

Company

Microchip Technology Inc.

Filename: net_pres_encryptionproviderapi.h

OSAL Library Help

This section describes the Operating System Abstraction Layer (OSAL) that is available in MPLAB Harmony.

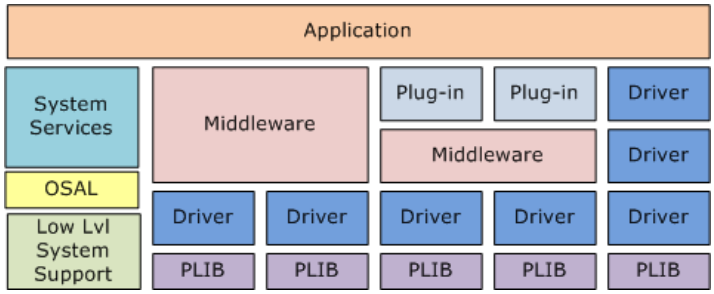
Introduction

The Operating System Abstraction Layer (OSAL) provides a consistent interface to allow MPLAB Harmony-compliant libraries to take advantage of Operating System constructs when running in an OS environment or when operating without one. It is designed to take care of the underlying differences between the available OS Kernels or when no kernel is present.

Description

The OSAL provides the interface to commonly available Real-Time Operating Systems (RTOS) such that MPLAB Harmony libraries may be written using a single interface to a minimal set of OS features needed to provide thread safety. The OSAL interface can be implemented appropriately to support almost any desired RTOS. For systems where no RTOS is available, or desired, a bare version of the OSAL supports either polled or interrupt-driven environments running directly on the hardware. This allows applications designed using MPLAB Harmony libraries to be executed in all three common embedded environments: polled (shared multi-tasking), interrupt-driven, or RTOS-based.

Note: It is possible to make RTOS independent applications using the OSAL. However, as explained in the following section, that is not its purpose. Use and selection of an RTOS is usually determined by the availability of its unique features. And, utilizing those features will, of course, make an application OS-specific.



Scope

By design, the OSAL is a minimal API intended only to enable thread-safe operation for MPLAB Harmony libraries. It only exposes a very small subset of the capabilities of an operating system so that MPLAB Harmony libraries can use semaphores, mutexes, and critical sections (and a few other things) necessary to protect shared resources (data structures, peripheral registers, and other memory objects) from corruption by unsynchronized access by multiple threads. This is done to allow MPLAB Harmony libraries to be made compatible with the largest variety of operating systems, by using a minimal subset of some of the most common OS features. The OSAL is not intended to provide a complete abstraction of an RTOS, which is what you would normally do to implement a complete application. Abstracting an entire operating system is a much more complex task that is roughly equivalent to defining your own RTOS.

The OSAL is not designed to replace a commercial kernel, and therefore, the user is encouraged to use any of the specific features of their chosen RTOS in order to achieve best performance. As such, the OSAL can be considered to be an Operating System Compatibility Layer offering MPLAB Harmony-compliant libraries the required common functions to ensure correct operation in both RTOS and non-RTOS environments.

The common interface presented by the OSAL is designed to offer a set of services typically found on micro-kernel and mini-scheduler systems. Because of this it has no aspirations to provide an equivalent set of capabilities as those found on large multi-tasking systems such as µCLinux™. The common services are designed to allow MPLAB Harmony to implement thread-safe Drivers and Middleware. The design intention is that drivers will use the minimal set of OSAL features necessary to ensure that they can safely operate in a multi-threaded environment yet can also compile and run correctly when no underlying RTOS is present. The range of features used by a driver is typically limited to these OSAL features (see the Library Interface section):

- Semaphore Functions
- Mutex Functions
- Critical Section Functions

Supported RTOS

RTOS	Release Type
FreeRTOS v8.x.x	Production
FreeRTOS v7.x.x	Production
OPENRTOS v8.x.x	Production
OPENRTOS v7.x.x	Production
Micriμm® µC/OS-II™	Beta

Micriμm® μC/OS-III™	Production
Express Logic ThreadX	Beta
SEGGER embOS®	Beta

Using the Library

This topic describes the basic architecture of the OSAL Library and provides information and examples on its use.

Description

Interface Header File: `osal.h`

The interface to the OSAL Library is defined in the `osal.h` header file. Any C language source (`.c`) file that uses the OSAL System Service library should include `osal.h`.

Library File: `osal_<vendor-specified RTOS name>.c` (i.e., freertos, ucos3, etc.)

The OSAL Library consists of a basic implementation and individual ports of the OSAL to target operating systems. The basic implementation is used when the macro OSAL_USE_BASIC is defined and ports of a third-party RTOS, which is used when the macro OSAL_USE_RTOS is defined.

When an RTOS is being used (the OSAL_USE_RTOS configuration) then an external implementation file which provides the required interface wrappers should be added to the project. For instance for the FreeRTOS operating system the file `osal_freertos.c` should be added, while for the Micriμm μC/OS-III operating system the file `osal_ucos3.c` should be added.

The basic implementation and some generic ports are provided with the Library, however, it is the responsibility of third-party vendors to supply an implementation file for operating systems that are not already supported.

Please refer to the What is MPLAB Harmony? section for how the OSAL Library interacts with the framework.

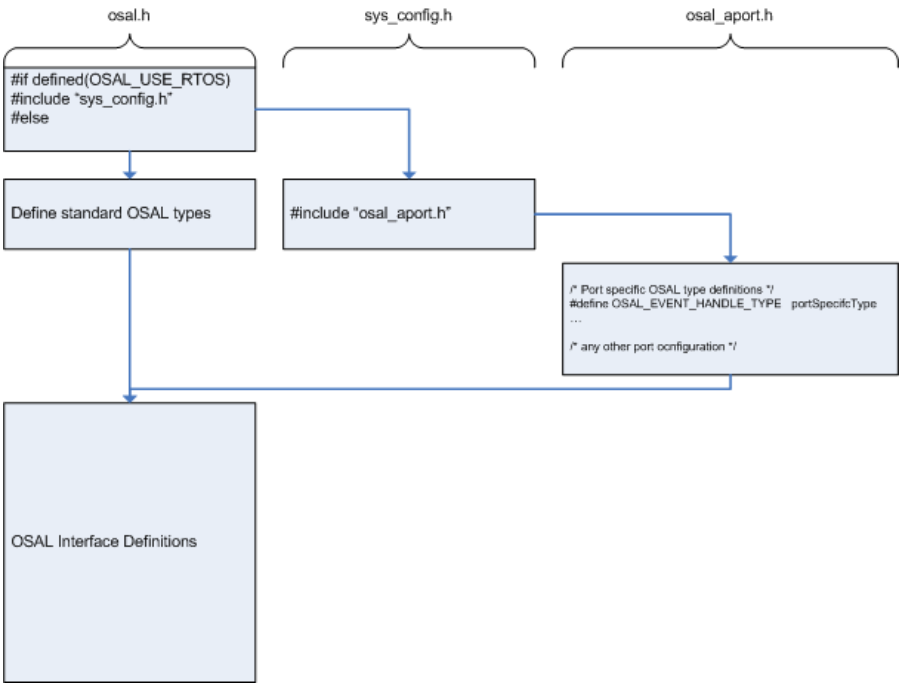
Abstraction Model

The OSAL Library provides a predefined set of functions and types that match common synchronization and communication services that an RTOS will typically provide. It is designed to be a lightweight abstraction model and deliberately excludes the much broader depth and breadth of services that a fully fledged RTOS provides. As such the interface defines only those core functions necessary for the MPLAB Harmony Drivers and middleware to operate in a multi-threaded environment.

Description

The common interface can easily be ported to many host Real-Time Operating Systems (RTOS) by third parties and the set of functions provides a basic level of RTOS compatibility. Where a specific RTOS does not implement a given architectural feature (e.g., events), the OSAL port for that RTOS should endeavor to imitate that feature using the constructs that are available. Although it is recognized that this may have a detrimental effect on the performance of that system it does allow MPLAB Harmony developers the broadest scope for using RTOS features in their designs.

Include Hierarchy of osal.h



Library Overview

This section provides an overview of the OSAL Library.

Description

Refer to the section System Services Introduction for how the system services operates in a system.

The OSAL Library provides a defined interface such that driver and middleware developers will be able to create MPLAB Harmony code that can safely operate in a multi-threaded environment when a supported RTOS is present yet will still compile and function correctly when MPLAB Harmony is being used in a non-RTOS environment with an interrupt or non-interrupt driven application model.

At the application layer, the developer is encouraged to use the specific features of a chosen RTOS once it has been selected since this is likely to provide a more effective and rich programming environment.

The OSAL Library is deliberately designed to be a thin layer over an underlying RTOS, which presents a predefined interface to the common features used by the majority of Real-Time Operating Systems, which includes:

Library Interface Section	Description
Semaphore Functions	Binary and counting semaphores.
Mutex Functions	Thread and resource locking mechanism.
Critical Section Functions	Application and scheduler locking mechanism.
Memory Allocation Functions	Memory allocation primitives or wrappers.
OSAL Control Functions	OSAL initialization.

One of the primary design guidelines is that a host operating system may not be present and so any operations that the OSAL presents are designed to compile out to safe default implementations if no RTOS is present. This can mean the following:

- Implementing a dummy function that mimics typical RTOS behavior
- Implementing a `#define` or inline function that returns a 'safe' generic return value, such as 'true' or a 'call succeeded' status
- Returning a OSAL_RESULT_NOT_IMPLEMENTED value to indicate an unsupported operation
- Throwing an OSAL_ASSERT failure to indicate a terminal error that prevents operation under specific circumstances

How the Library Works

This section provides information on how the OSAL Library works.

Core Functionality

The OSAL Library implements a lightweight abstraction layer on top of a host operating system. The OSAL functionality is designed to allow a programmer access to typical RTOS based functions than can reasonably be expected to exist on a small kernel or micro-scheduler typically found on the target Microchip Microcontrollers.

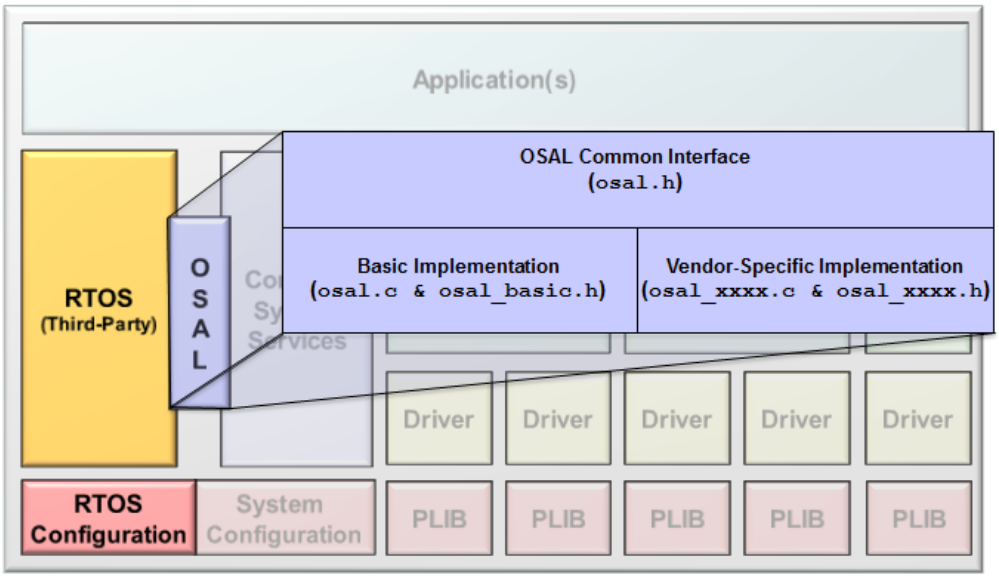
Description

The OSAL is designed to be compiled under three specific mutually exclusive operating modes:

Compilation Mode	Description
OSAL_USE_BASIC	Basic level of support. There is no underlying RTOS however certain facilities are replicated such as Semaphores and Mutexes.
OSAL_USE_RTOS	A full implementation of the OSAL interface is available and there is an underlying RTOS as part of the program.

In the absence of any one of these values being defined or passed into the compiler as a command line parameter then the OSAL will default to the OSAL_USE_BASIC mode of operation.

The following conceptual diagram shows the common interface and how the individual implementations are derived depending upon the configured mode of compilation.



The OSAL interface makes extensive use of object handles when passing RTOS parameters around the various functions. This has been done since depending on the operating mode and underlying RTOS (if any) there may be substantial differences between the C types of the variables. For instance, in the basic implementation mode a semaphore is implemented as a single 8-bit quantity to ensure atomic access on all architectures. On a third party RTOS the same semaphore may be implemented as a 16-bit index into an array of pointers while on another RTOS it could be represented as a size_t pointer (a pointer to the basic word type for that architecture). Because of these differences, the most convenient mechanism to maintain a regular interface is by the use of handles to implementation defined variables.

The majority of the OSAL objects also provide a OSAL_XXX_Delete function. This is provided for completeness however care should be taken in its use. Many RTOS implementations do not provide the ability to delete an object and excessive use of the delete can lead to memory fragmentation and unpredictable system behavior. It is recommended that it is not used at this time, or if used, the return value should be checked for a possible failure to delete the object.

In the following sections the basic behavior of the supported functions are documented. For the purposes of documentation the assumed operation mode will be that a full RTOS is present and so the behavior is that which will be observed in that scenario.

Semaphores

The semaphore implements a method for thread synchronization. This synchronization can be either between one thread and another or between an ISR and a thread. A semaphore once signalled will unblock the highest priority thread currently pending on it.

Description

A semaphore can be used to lock a shared resource, although it is more normal to use a mutex for such an activity. Once obtained a semaphore should be posted back to enable it to be retaken at a later time or in another thread.

```
/* mainline code prior to OS start */
    /* declare a variable of type semaphore handle */
    OSAL_SEM_DECLARE(semSync);
    /* create the semaphore */
    OSAL_SEM_Create(&semSync, OSAL_SEM_TYPE_BINARY, 0, 0);


/* thread one */
    ...
    /* take the semaphore without waiting */
    OSAL_SEM_Pend(semSync, 0);
    ... perform some actions
    /* return the semaphore */
    OSAL_SEM_Post(semSync);
    ...

/* thread two must not execute until thread one has finished its operations*/
    ...
    /* block on the semaphore */
    OSAL_SEM_Pend(semSync, OSAL_WAIT_FOREVER);
    ... perform some more actions
    /* return the semaphore */
    OSAL_SEM_Post(semSync);
```

A semaphore can be signalled multiple times and so provides a method for an ISR to release a thread waiting on it. Even though the blocked

thread never returns the semaphore, because the asynchronous ISR repeatedly posts it the next time the thread wants to pend on the semaphore it will be available. By moving the majority of interrupt service processing from the ISR to a high priority thread the system response time is improved and the eventual processing can take advantage of OSAL features such as mutexes and queues which would normally be harder to implement inside the ISR. This technique is known as deferred interrupt processing.

```
/* an example interrupt handler called from an ISR that performs task synchronization using a semaphore */
void _ISRTasksRX(void) /* N.B. pseudo-code ISR */
{
    ...

    _DRV_USART_InterruptSourceStatusClear(_DRV_USART_GET_INT_SRC_RX(_DRV_USART_OBJ(dObj,
rxInterruptSource)));

    /* Release the receive semaphore unblocking any tasks */
    OSAL_SEM_PostISR(_DRV_USART_OBJ(dObj, rxSemID));

} /* DRV_USART_TasksRX */
```

Mutex Operation

A mutex or mutual exclusion is used to protect a shared resource from access by multiple threads at the same time. A shared resource may be a common data structure in RAM or it may be a hardware peripheral. In either case a mutex can be used to ensure the integrity of the entire resource by only allowing one thread to access it at a time.

Description

The library must be written in such a way that before the shared resources is accessed the mutex has to be obtained. Once obtained the accesses should occur, and once complete the mutex should then be released. While no restrictions are enforced the sequence of operations between the lock and unlock should ideally take as few lines of code as possible to ensure good system performance.

The mutex may be implemented as a form of binary semaphore but an underlying RTOS will often add other features. It is normal to add the restriction that a mutex may only be unlocked from the thread that originally obtained the lock in the first place. The RTOS may also provide features to mitigate priority inversion problems (where a high priority thread blocks on a lower priority one holding a mutex) by providing priority inheritance allowing lower priority threads to be temporarily raised to complete and release a locked mutex.

```
/* perform operations on a shared data structure */
struct myDataStructure {
    uint16_t  x;
    uint8_t y;
} myDataStructure;

...
OSAL_MUTEX_DECLARE(mutexDS);
OSAL_MUTEX_Create(&mutexDS);

...
/* wait 2 seconds to obtain the mutex */
if (OSAL_MUTEX_Lock(mutexDS, 2000) == OSAL_RESULT_TRUE)
{
    /* operate on the data structure */
    myDataStructure.x = 32;
    OSAL_MUTEX_Unlock(mutexDS);
}
```

Critical Section Operation

This section describes how critical sections are used.

Description

Critical sections are used to form sequences of code that must operate in an atomic manner. The interface allows for the possibility of two types of critical section.

- When the critical section is entered all interrupts on the microcontroller are disabled. This prevents the protected sequence of code from being interrupted and ensures the complete atomicity of the operation. This is denoted by the OSAL_CRIT_TYPE_HIGH value
- When the critical section is entered the RTOS scheduler is disabled. In this second case other threads are prevented from running however interrupts can still occur which allows any asynchronous events to still be received and for the temporal accuracy of the RTOS scheduler to be maintained. This is denoted by the OSAL_CRIT_TYPE_LOW value

Since the behavior in the two cases is different the type of critical section must be identified in both the call to enter and leave.

```
/* enter and leave a critical section disabling interrupts */
OSAL_CRIT_Enter(OSAL_CRIT_TYPE_HIGH);
/* perform an atomic sequence of code */
```

```
...
/* leave the critical section */
OSAL_CRIT_Leave(OSAL_CRIT_TYPE_HIGH);
```

The underlying RTOS may not support the second scenario, in which case the OSAL implementation will default to disabling all interrupts.

Memory Operation

This section describes the memory operation using the OSAL Library.

Description

The OSAL Library provides an interface to a memory allocation mechanism. The memory required for dynamic instantiation of variables is normally provided by allocating it from the heap. However the standard C library implementation of `malloc` and free are not considered thread safe and so OSAL specific functions must be used if MPLAB Harmony or the application requires dynamic memory during operation.

When operating without an underlying RTOS the OSAL memory allocators default to using standard malloc and free functions. However, when operating with an RTOS the calls will defer to the specific scheme used by the RTOS. This may involve multiple memory pools or it may simply involve adding a critical section around calls to malloc and free. It is left to the implementation to define the most appropriate scheme.

```
/* allocate a large buffer */
uint8_t* buffer;

buffer = OSAL_Malloc(8000);
if (buffer != NULL)
{
    ... manipulate the buffer
    /* free the buffer */
    OSAL_Free(buffer);
    buffer = NULL;
}
```

OSAL Operation

This section describes OSAL control features.

Description

When the OSAL is using an underlying RTOS it may be necessary to allow the RTOS to perform one-time initialization before any calls to it are made. For instance, the RTOS might implement multiple memory pools for managing queues and semaphores, and it must be given the chance to create these pools before any of the objects are created. For this reason the application program should call OSAL_Initialize early on and certainly before any MPLAB Harmony drivers or middleware is initialized (since these may also create OSAL objects at creation time).

Once the OSAL is initialized and any other remaining parts of the system are configured correctly, the specific RTOS can be started.

Configuring the Library

The configuration of the OSAL Library is based on the file `system_config.h`.

This header file contains the configuration selection for the OSAL Library. Based on the selections made, the OSAL may support the selected features.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Description

To configure the OSAL the compilation mode needs to be selected. Three options are currently supported and these define the implementation details for the OSAL. In all cases the same interface is present; however, the functionality at the varying levels will be different or not supported. Only one of these #define statements should be specified, and this can be either done in a common location (such as `system_config.h`) or on the build tool command line.

Compilation Mode	Description
OSAL_USE_BASIC	Basic level of support. There is no underlying RTOS; however, certain facilities are replicated such as Semaphores and Mutexes.
OSAL_USE_RTOS	A full implementation of the OSAL interface is available and there is an underlying RTOS as part of the program.

Initialization Overrides

This section provides information for overriding OSAL initialization.

Description

The OSAL requires minimal configuration. In the case of the OSAL_USE_BASIC, all of the required configuration is provided by the standard `osal.h` include file. When a particular RTOS is being used then additional OSAL configuration parameters should be placed in `system_config.h`.

The primary requirement is to include the relevant interface implementation for the selected RTOS being used. This file will contain the mappings from OSAL interface types to types relevant for the chosen RTOS.

```
/* included the name of the port specific interface file */
#include "osal_<port name>.h"
```

Examples - Sample Functionality

Example `system_config.h` fragment:

```
// ****************************************************************************
// ****************************************************************************
// Section: Application Configuration
// ****************************************************************************
// ****************************************************************************
/*  These definitions select the configuration options for the application */

/* Include the port specific OS implementation files. The file is required to
 * create an implementation layer for the selected OS. This include will be
 * changed depending on the required OS being used */
#include "osal_freertos.h"
```

Building the Library

This section lists the files that are available in the OSAL Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/osal`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library for the selected RTOS.

Source File Name	Description
`/osal.h`	This file provides the interface definitions of the OSAL Library.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/osal.c` `/src/osal_impl_basic.h`	Source files added to the project if using the "bare-metal" OSAL basic implementation used when no RTOS is present.
`/src/osal_embos.c` `/src/osal_embos.h`	Source files added to the project if using the embOS implementation for compatibility with the embOS operating system from SEGGER.
`/src/osal_freertos.c` `/src/osal_freertos.h`	Source files added to the project if using the FreeRTOS implementation for compatibility with the current version 8 of the FreeRTOS operating system from Real Time Engineers Ltd.
`/src/osal_freertos_v7xx.c` `/src/osal_freertos_v7xx.h`	Source files added to the project if using the FreeRTOS v7 implementation for compatibility with the version 7 of the FreeRTOS operating system from Real Time Engineers Ltd.
`/src/osal_openrtos.c` `/src/osal_openrtos.h`	Source files added to the project if using the OPENRTOS implementation for compatibility with the current version 8 of the OPENRTOS operating system from Wittenstein,
`/src/osal_openrtos_v7xx.c` `/src/osal_openrtos_v7xx.h`	Source files added to the project if using the OPENRTOS implementation for compatibility with the version 7 of the OPENRTOS operating system from Wittenstein.

/src/osal_threadx.c /src/osal_threadx.h	Source files added to the project if using the ThreadX implementation for compatibility with the ThreadX operating system from Express Logic.
/src/osal_ucos2.c /src/osal_ucos2.h	Source files added to the project if using the µC/OS-II implementation for compatibility with the µC/OS-II operating system from Micriµm.
/src/osal_ucos3.c /src/osal/ucos3.h	Source files added to the project if using the µC/OS-III implementation for compatibility with the µC/OS-III operating system from Micriµm.
/src/osal_impl_none.h	This header file should not be used for new designs, as it will be eliminated in a future release.

Optional File(s)

There are no optional files for the OSAL Library.

Module Dependencies

When OSAL_USE_RTOS is defined, the OSAL is dependent on the selected RTOS.

Library Interface

a) Semaphore Functions

	Name	Description
	OSAL_SEM_Create	Creates an OSAL Semaphore.
	OSAL_SEM_Delete	Deletes an OSAL Semaphore.
	OSAL_SEM_GetCount	Returns the current value of a counting semaphore.
	OSAL_SEM_Pend	Waits on a semaphore. Returns true if the semaphore was obtained within the time limit.
	OSAL_SEM_Post	Posts a semaphore or increments a counting semaphore.
	OSAL_SEM_PostISR	Posts a semaphore or increments a counting semaphore from within an Interrupt Service Routine (ISR).

b) Mutex Functions

	Name	Description
	OSAL_MUTEX_Create	Creates a mutex.
	OSAL_MUTEX_Delete	Deletes a mutex.
	OSAL_MUTEX_Lock	Locks a mutex.
	OSAL_MUTEX_Unlock	Unlocks a mutex.

c) Critical Section Functions

	Name	Description
	OSAL_CRIT_Enter	Enters a critical section with the specified severity level.
	OSAL_CRIT_Leave	Leaves a critical section with the specified severity level.

d) Memory Allocation Functions

	Name	Description
	OSAL_Free	Deallocates a block of memory and return to the default pool.
	OSAL_Malloc	Allocates memory using the OSAL default allocator.

e) OSAL Control Functions

	Name	Description
	OSAL_Initialize	Performs OSAL initialization.
	OSAL_Name	Obtains the name of the underlying RTOS.

f) Data Types and Constants

	Name	Description
	OSAL_RESULT	Enumerated type representing the general return value from OSAL functions.
	OSAL_SEM_TYPE	Enumerated type representing the possible types of semaphore.
	OSAL_CRIT_TYPE	Enumerated type representing the possible types of critical section.
	OSAL_MUTEX_DECLARE	Declares an OSAL mutex.
	OSAL_SEM_DECLARE	Declares an OSAL semaphore.

Description

This section describes the APIs of the OSAL Library. Refer to each section for a description.

a) Semaphore Functions

OSAL_SEM_Create Function

Creates an OSAL Semaphore.

File

help_osal.h

C

```
OSAL_RESULT OSAL_SEM_Create(OSAL_SEM_HANDLE_TYPE* semID, OSAL_SEM_TYPE type, uint8_t maxCount, uint8_t
initialCount);
```

Returns

- OSAL_RESULT_TRUE - Semaphore created
- OSAL_RESULT_FALSE - Semaphore creation failed
- semID - Updated with valid semaphore handle if call was successful

Description

This function creates an OSAL binary or counting semaphore. If OSAL_SEM_TYPE_BINARY is specified, the maxcount and initialCount values are ignored.

Remarks

None.

Preconditions

Semaphore must have been declared.

Example

```
OSAL_SEM_Create(&mySemID, OSAL_SEM_TYPE_COUNTING, 10, 5);
```

Parameters

Parameters	Description
semID	Pointer to the Semaphore ID
type	If OSAL_SEM_TYPE_BINARY, create a binary semaphore. If OSAL_SEM_TYPE_COUNTING, create a counting semaphore with the specified count values.
maxCount	Maximum value for a counting semaphore (ignored for a BINARY semaphore). This parameter is ignored for Express Logic ThreadX, SEGGER embOS, and Micrium/OS-III as these RTOS libraries do not support this parameter.
initialCount	Starting count value for the semaphore (ignored for a BINARY semaphore) This should be less than or equal to maxCount when used with a RTOS library that supports the maxCount parameter (i.e., FreeRTOS and OPENRTOS)

Function

OSAL_RESULT OSAL_SEM_Create(OSAL_SEM_HANDLE_TYPE* semID, OSAL_SEM_TYPE type,

uint8_t maxCount, uint8_t initialCount)

OSAL_SEM_Delete Function

Deletes an OSAL Semaphore.

File

help_osal.h

C

```
OSAL_RESULT OSAL_SEM_Delete(OSAL_SEM_HANDLE_TYPE* semID);
```

Returns

- OSAL_RESULT_TRUE - Semaphore deleted
- OSAL_RESULT_FALSE - Semaphore deletion failed

Description

This function deletes an OSAL semaphore.

Remarks

None.

Preconditions

Semaphore must have been created.

Example

```
OSAL_SEM_Delete(&mySemID);
```

Parameters

Parameters	Description
semID	Pointer to the Semaphore ID

Function

OSAL_RESULT OSAL_SEM_Delete(OSAL_SEM_HANDLE_TYPE* semID)

OSAL_SEM_GetCount Function

Returns the current value of a counting semaphore.

File

help_osal.h

C

```
uint8_t OSAL_SEM_GetCount(OSAL_SEM_HANDLE_TYPE* semID);
```

Returns

- 0 - Semaphore is unavailable
- 1-255 - Current value of the counting semaphore

Description

This function returns the current value of a counting semaphore. The value returned is assumed to be a single value ranging from 0-255.

Remarks

None.

Preconditions

Semaphore must have been created.

Example

```
uint8_t semCount;

semCount = OSAL_SEM_GetCount(semUART);

if (semCount > 0)
{
   // obtain the semaphore
    if (OSAL_SEM_Pend(&semUART) == OSAL_RESULT_TRUE)
    {
       // perform processing on the comm channel
       ...
    }
}
else
{
```

```
    // no comm channels available
    ...
}
```

Parameters

Parameters	Description
semID	Pointer to the Semaphore ID

Function

uint8_t OSAL_SEM_GetCount(OSAL_SEM_HANDLE_TYPE* semID)

OSAL_SEM_Pend Function

Waits on a semaphore. Returns true if the semaphore was obtained within the time limit.

File

help_osal.h

C

```
OSAL_RESULT OSAL_SEM_Pend(OSAL_SEM_HANDLE_TYPE* semID, uint16_t waitMS);
```

Returns

- OSAL_RESULT_TRUE - Semaphore obtained
- OSAL_RESULT_FALSE - Semaphore not obtained or time-out occurred

Description

This function is a blocking function call that waits (i.e., pends) on a semaphore. The function will return true if the semaphore has been obtained, or false if it was not available or the time limit was exceeded.

Remarks

None.

Preconditions

Semaphore must have been created.

Example

```
if (OSAL_SEM_Pend(&semUARTRX, 50) == OSAL_RESULT_TRUE)
{
    // character available
    c = DRV_USART_ReadByte(drvID);
    ...
}
else
{
    // character not available, resend prompt
    ...
}
```

Parameters

Parameters	Description
semID	Pointer to the Semaphore ID
waitMS	Time limit to wait in milliseconds: • 0 - do not wait • OSAL_WAIT_FOREVER - return only when semaphore is obtained • Other values - time-out delay

Function

OSAL_RESULT OSAL_SEM_Pend(OSAL_SEM_HANDLE_TYPE* semID, uint16_t waitMS)

OSAL_SEM_Post Function

Posts a semaphore or increments a counting semaphore.

File

help_osal.h

C

```
OSAL_RESULT OSAL_SEM_Post(OSAL_SEM_HANDLE_TYPE* semID);
```

Returns

- OSAL_RESULT_TRUE - Semaphore posted
- OSAL_RESULT_FALSE - Semaphore not posted

Description

This function posts a binary semaphore or increments a counting semaphore. The highest priority task currently blocked on the semaphore will be released and made ready to run.

Remarks

None.

Preconditions

Semaphore must have been created.

Example

```
OSAL_SEM_Post(&semSignal);
```

Parameters

Parameters	Description
semID	The semID

Function

OSAL_RESULT OSAL_SEM_Post(OSAL_SEM_HANDLE_TYPE* semID)

OSAL_SEM_PostISR Function

Posts a semaphore or increments a counting semaphore from within an Interrupt Service Routine (ISR).

File

help_osal.h

C

```
OSAL_RESULT OSAL_SEM_PostISR(OSAL_SEM_HANDLE_TYPE* semID);
```

Returns

- OSAL_RESULT_TRUE - Semaphore posted
- OSAL_RESULT_FALSE - Semaphore not posted

Description

This function posts a binary semaphore or increments a counting semaphore. The highest priority task currently blocked on the semaphore will be released and made ready to run. This form of the post function should be used inside an ISR.

Remarks

This version of the OSAL_SEM_Post function should be used if the program is, or may be, operating inside an ISR. The OSAL will take the necessary steps to ensure correct operation possibly disabling interrupts or entering a critical section. The exact requirements will depend upon the particular RTOS being used.

Preconditions

Semaphore must have been created.

Example

```
void __ISR(UART_2_VECTOR) _UART2RXHandler()
{
   char c;

   // read the character
```

```
    c = U2RXREG;
    // clear the interrupt flag
    IFS1bits.U2IF = 0;
    // post a semaphore indicating a character has been received
    OSAL_SEM_PostISR(&semSignal);

}
```

Parameters

Parameters	Description
semID	Pointer to the Semaphore ID

Function

OSAL_RESULT OSAL_SEM_PostISR(OSAL_SEM_HANDLE_TYPE* semID)

b) Mutex Functions

OSAL_MUTEX_Create Function

Creates a mutex.

File

help_osal.h

C

```
OSAL_RESULT OSAL_MUTEX_Create(OSAL_MUTEX_HANDLE_TYPE* mutexID);
```

Returns

- OSAL_RESULT_TRUE - Mutex successfully created
- OSAL_RESULT_FALSE - Mutex failed to be created

Description

This function creates a mutex, allocating storage if required and placing the mutex handle into the passed parameter.

Remarks

None.

Preconditions

Mutex must have been declared.

Example

```
OSAL_MUTEX_HANDLE_TYPE mutexData;

OSAL_MUTEX_Create(&mutexData);
...
 if (OSAL_MUTEX_Lock(&mutexData, 1000) == OSAL_RESULT_TRUE)
 {
    // manipulate the shared data
    ...
 }
```

Parameters

Parameters	Description
mutexID	Pointer to the mutex handle

Function

OSAL_RESULT OSAL_MUTEX_Create(OSAL_MUTEX_HANDLE_TYPE* mutexID)

OSAL_MUTEX_Delete Function

Deletes a mutex.

File

help_osal.h

C

```
OSAL_RESULT OSAL_MUTEX_Delete(OSAL_MUTEX_HANDLE_TYPE* mutexID);
```

Returns

- OSAL_RESULT_TRUE - Mutex successfully deleted
- OSAL_RESULT_FALSE - Mutex failed to be deleted

Description

This function deletes a mutex and frees associated storage if required.

Remarks

None.

Preconditions

None.

Example

```
OSAL_MUTEX_Delete(&mutexData);
```

Parameters

Parameters	Description
mutexID	Pointer to the mutex handle

Function

OSAL_RESULT OSAL_MUTEX_Delete(OSAL_MUTEX_HANDLE_TYPE* mutexID)

OSAL_MUTEX_Lock Function

Locks a mutex.

File

help_osal.h

C

```
OSAL_RESULT OSAL_MUTEX_Lock(OSAL_MUTEX_HANDLE_TYPE* mutexID, uint16_t waitMS);
```

Returns

- OSAL_RESULT_TRUE - Mutex successfully obtained
- OSAL_RESULT_FALSE - Mutex failed to be obtained or time-out occurred

Description

This function locks a mutex, waiting for the specified time-out. If it cannot be obtained or the time-out period elapses 'false' is returned.

Remarks

None.

Preconditions

Mutex must have been created.

Example

```
...
 if (OSAL_MUTEX_Lock(&mutexData, 1000) == OSAL_RESULT_TRUE)
 {
    // manipulate the shared data
    ...

    // unlock the mutex
    OSAL_MUTEX_Unlock(&mutexData);
 }
```

Parameters

Parameters	Description
mutexID	Pointer to the mutex handle
waitMS	Time-out value in milliseconds: • 0, do not wait return immediately • OSAL_WAIT_FOREVER, wait until mutex is obtained before returning • Other values, time-out delay

Function

OSAL_RESULT OSAL_MUTEX_Lock(OSAL_MUTEX_HANDLE_TYPE* mutexID, uint16_t waitMS)

OSAL_MUTEX_Unlock Function

Unlocks a mutex.

File

help_osal.h

C

```
OSAL_RESULT OSAL_MUTEX_Unlock(OSAL_MUTEX_HANDLE_TYPE* mutexID);
```

Returns

- OSAL_RESULT_TRUE - Mutex released
- OSAL_RESULT_FALSE - Mutex failed to be released or error occurred

Description

This function unlocks a previously obtained mutex.

Remarks

None.

Preconditions

Mutex must have been created.

Example

```
...
if (OSAL_MUTEX_Lock(&mutexData, 1000) == OSAL_RESULT_TRUE)
{
    // manipulate the shared data
    ...

    // unlock the mutex
    OSAL_MUTEX_Unlock(&mutexData);
}
```

Parameters

Parameters	Description
mutexID	Pointer to the mutex handle

Function

OSAL_RESULT OSAL_MUTEX_Unlock(OSAL_MUTEX_HANDLE_TYPE* mutexID)

c) Critical Section Functions

OSAL_CRIT_Enter Function

Enters a critical section with the specified severity level.

File

help_osal.h

C

```
OSAL_CRITSECT_DATA_TYPE OSAL_CRIT_Enter(OSAL_CRIT_TYPE severity);
```

Returns

A data type of OSAL_CRITSECT_DATA_TYPE, this value represents the state of interrupts before entering the critical section.

Description

This function enters a critical section of code. It is assumed that the sequence of operations bounded by the enter and leave critical section operations is treated as one atomic sequence that will not be disturbed. This function should be paired with OSAL_CRIT_Leave().

Remarks

The sequence of operations bounded by the OSAL_CRIT_Enter and OSAL_CRIT_Leave form a critical section. The severity level defines whether the RTOS should perform task locking or completely disable all interrupts.

Preconditions

None.

Example

```
OSAL_CRITSECT_DATA_TYPE IntState;
// prevent other tasks preempting this sequence of code
IntState = OSAL_CRIT_Enter(OSAL_CRIT_TYPE_HIGH);
// modify the peripheral
DRV_USART_Reinitialize( objUSART,  &initData);
OSAL_CRIT_Leave(OSAL_CRIT_TYPE_HIGH, IntState);
```

Parameters

Parameters	Description
severity	OSAL_CRIT_TYPE_LOW, The RTOS should disable all other running tasks effectively locking the scheduling mechanism. OSAL_CRIT_TYPE_HIGH, The RTOS should disable all possible interrupts sources including the scheduler ensuring that the sequence of code operates without interruption. The state of interrupts are returned to the user before they are disabled.

Function

OSAL_CRITSECT_DATA_TYPE void OSAL_CRIT_Enter(OSAL_CRIT_TYPE severity)

OSAL_CRIT_Leave Function

Leaves a critical section with the specified severity level.

File

help_osal.h

C

```
void OSAL_CRIT_Leave(OSAL_CRIT_TYPE severity, OSAL_CRITSECT_DATA_TYPE status);
```

Returns

None.

Description

This function leaves a critical section of code. It is assumed that the sequence of operations bounded by the enter and leave critical section operations is treated as one atomic sequence that will not be disturbed. The severity should match the severity level used in the corresponding OSAL_CRIT_Enter call to ensure that the RTOS carries out the correct action.

Remarks

The sequence of operations bounded by the OSAL_CRIT_Enter and OSAL_CRIT_Leave form a critical section. The severity level defines whether the RTOS should perform task locking or completely disable all interrupts.

Preconditions

None.

Example

```
OSAL_CRITSECT_DATA_TYPE IntState;
// prevent other tasks preempting this sequence of code
intState = OSAL_CRIT_Enter(OSAL_CRIT_TYPE_LOW);
// modify the peripheral
DRV_USART_Reinitialize( objUSART,  &initData);
OSAL_CRIT_Leave(OSAL_CRIT_TYPE_LOW, IntState);
```

Parameters

Parameters	Description
severity	OSAL_CRIT_TYPE_LOW, The scheduler will be unlocked, if no other nested calls to OSAL_CRIT_ENTER have been made. OSAL_CRIT_TYPE_HIGH, Interrupts are returned to the state passed into this function. The state should of been saved by an earlier call to OSAL_CRIT_Enter.
status	The value which will be used to set the state of global interrupts, if OSAL_CRIT_TYPE_HIGH is passed in.

Function

void OSAL_CRIT_Leave(OSAL_CRIT_TYPE severity, OSAL_CRITSECT_DATA_TYPE status)

d) Memory Allocation Functions

OSAL_Free Function

Deallocates a block of memory and return to the default pool.

File

help_osal.h

C

```
void OSAL_Free(void* pData);
```

Returns

None.

Description

This function deallocates memory and returns it to the default pool. In an RTOS-based application, the memory may have been allocated from multiple pools or simply from the heap. In non-RTOS applications, this function calls the C standard function free.

Remarks

None.

Preconditions

None.

Example

```
// create a working array
uint8_t* pData;

 pData = OSAL_Malloc(32);
 if (pData != NULL)
 {
    ...

    // deallocate the memory
    OSAL_Free(pData);
    // and prevent it accidentally being used again
    pData = NULL;
 }
```

Parameters

Parameters	Description
pData	Pointer to the memory block to be set free

Function

void OSAL_Free(void* pData)

OSAL_Malloc Function

Allocates memory using the OSAL default allocator.

File

help_osal.h

C

```
void* OSAL_Malloc(size_t size);
```

Returns

Pointer to the block of allocated memory. NULL is returned if memory could not be allocated.

Description

This function allocates a block of memory from the default allocator from the underlying RTOS. If no RTOS is present, it defaults to malloc. Many operating systems incorporate their own memory allocation scheme, using pools, blocks or by wrapping the standard C library functions in a critical section. Since a MPLAB Harmony application may not know what target OS is being used (if any), this function ensures that the correct thread-safe memory allocator will be used.

Remarks

None.

Preconditions

None.

Example

```
// create a working array
uint8_t* pData;

 pData = OSAL_Malloc(32);
 if (pData != NULL)
 {
    ...
 }
```

Parameters

Parameters	Description
size	Size of the requested memory block in bytes

Function

void* OSAL_Malloc(size_t size)

e) OSAL Control Functions

OSAL_Initialize Function

Performs OSAL initialization.

File

help_osal.h

C

```
OSAL_RESULT OSAL_Initialize();
```

Returns

OSAL_RESULT_TRUE - Initialization completed successfully.

Description

This function performs OSAL initialization .This function should be called near the start of main in an application that will use an underlying RTOS. This permits the RTOS to perform any one time initialization before the application attempts to create drivers or other items that may use the RTOS. Typical actions performed by OSAL_Initialize would be to allocate and prepare any memory pools for later use.

Remarks

None.

Preconditions

None.

Example

```
int main()
{
    OSAL_Initialize();

    App_Init();
    OSAL_Start();
}
```

Function

OSAL_RESULT OSAL_Initialize()

OSAL_Name Function

Obtains the name of the underlying RTOS.

File

help_osal.h

C

```
const char* OSAL_Name();
```

Returns

const char* - Name of the underlying RTOS or NULL

Description

This function returns a const char* to the textual name of the RTOS. The name is a NULL terminated string.

Remarks

None.

Preconditions

None.

Example

```
// get the RTOS name
const char* sName;

sName = OSAL_Name();
sprintf(buff, "RTOS: %s", sName);
```

Function

const char* OSAL_Name()

f) Data Types and Constants

OSAL_RESULT Enumeration

Enumerated type representing the general return value from OSAL functions.

File

help_osal.h

C

```
enum OSAL_RESULT {
  OSAL_RESULT_NOT_IMPLEMENTED = -1,
  OSAL_RESULT_FALSE = 0,
  OSAL_RESULT_TRUE = 1
};
```

Description

OSAL Result type

This enum represents possible return types from OSAL functions.

Remarks

These enum values are the possible return values from OSAL functions where a standard success/fail type response is required. The majority of OSAL functions will return this type with a few exceptions.

OSAL_SEM_TYPE Enumeration

Enumerated type representing the possible types of semaphore.

File

help_osal.h

C

```
enum OSAL_SEM_TYPE {
  OSAL_SEM_TYPE_BINARY,
  OSAL_SEM_TYPE_COUNTING
};
```

Description

OSAL Semaphore Type

This enum represents possible semaphore types.

OSAL_SEM_TYPE_BINARY - Simple binary type that can be taken once

OSAL_SEM_TYPE_COUNTING - Complex type that can be taken set number of times defined at creation time

Remarks

Binary and counting semaphore type.

OSAL_CRIT_TYPE Enumeration

Enumerated type representing the possible types of critical section.

File

help_osal.h

C

```
enum OSAL_CRIT_TYPE {
  OSAL_CRIT_TYPE_LOW,
  OSAL_CRIT_TYPE_HIGH
};
```

Description

OSAL Critical Type

This enum represents possible critical section types.

OSAL_CRIT_TYPE_LOW - Low priority critical section, can be formed by locking the scheduler (if supported by RTOS) OSAL_CRIT_TYPE_HIGH - High priority critical section, will be formed by disabling all interrupts.

Remarks

Critical section types.

OSAL_MUTEX_DECLARE Macro

Declares an OSAL mutex.

File

help_osal.h

C

```
#define OSAL_MUTEX_DECLARE(mutexID) OSAL_MUTEX_HANDLE_TYPE mutexID
```

Description

OSAL_MUTEX_Declare(mutexID)

This function declares a data item of type OSAL_MUTEX_HANDLE_TYPE.

Remarks

None.

// DOM-IGNORE-BEGIN

Place the definition of the OSAL_MUTEX_Declare macro inside each specific implementation file. Not all implementation files use the same definition. The type for each individual RTOS is different, and to handle this properly in the code, each implementation must define its own declaration. **REMOVE THIS NOTE AND THE DOM STATEMENTS IF USING THIS API IN A FILE THAT IS INCLUDED IN THE MPLAB HARMONY INSTALLATION** // DOM-IGNORE-END

OSAL_SEM_DECLARE Macro

Declares an OSAL semaphore.

File

help_osal.h

C

```
#define OSAL_SEM_DECLARE(semID) OSAL_SEM_HANDLE_TYPE   semID
```

Description

OSAL_SEM_Declare(semID)

This function declares a data item of type OSAL_SEM_HANDLE_TYPE.

Remarks

None.

Files

Files

Name	Description
osal.h	Common interface definitions for the Operating System Abstraction Layer (OSAL).

Description

This section lists the source and header files used by the OSAL Library.

osal.h

Common interface definitions for the Operating System Abstraction Layer (OSAL).

Description

Descriptive File Name: Operating System Abstraction Layer

This file defines the common interface to the Operating System Abstraction Layer. It defines the common types used by the OSAL and defines the function prototypes. Depending upon the OSAL mode, a support level specific implementation file is included by this file to give the required level

of compatibility. The available support levels include, OSAL_USE_NONE, OSAL_USE_BASIC, and OSAL_USE_RTOS.

File Name

osal.h

Company

Microchip Technology Inc.

Peripheral Libraries Help

This section provides descriptions of the peripheral libraries that are available in MPLAB Harmony.

Description

The source code for the MPLAB Harmony Peripheral Libraries (PLIBs) is distributed in the installation (`<install-dir>/framework/peripheral`). Each PLIB folder has (at least) two sub-folders: `processor` and `templates`. The files within the `processor` directory are generated mechanically during the MPLAB Harmony development process. These files define register macro translations and translate between the API function prototypes and different variants of the "inline" function implementations that are contained in the `templates` folder.

With a few exceptions PLIBs are implemented as C-language inline functions and translated by the preprocessor at build time. If almost any optimization is used and constants are passed into them, each call will normally compile away to just a few bytes of code. Prebuilt binary (`.a`) files are also provided (built at a -O3 optimization level) for times when no optimization is used or any time the compiler decides to generate a true function call instead of inlining the function. The prebuilt binary (`.a`) libraries are generated by changing the definitions of the PLIB_INLINE and PLIB_INLINE_API macros attached to the PLIB implementations from "inline extern" to just "extern" when the binaries are built. A MPLAB X IDE project is provided in the `<install-dir>/build/framework/peripheral` folder to allow a user to rebuild these binaries, if desired. Be sure to inspect the compiler settings and post-build steps, to see how this is done and to ensure the desired settings are used.

Peripheral Library Overview

This topic provides an overview of the peripheral libraries that are available in MPLAB Harmony.

Introduction

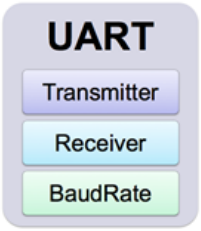
This topic provides an overview of the peripheral libraries in MPLAB Harmony.

Description

Supported PIC32 Devices and Release Type

Note: Refer to the *Release Contents* > *Peripheral Libraries* section in the MPLAB Harmony Release Notes for the list of supported PIC32 devices and their release type. A PDF copy of the release notes is provided in the `<install-dir>/doc` folder of your installation.

MPLAB Harmony peripheral libraries (PLIBs) model the hardware peripheral modules available on Microchip microcontrollers by breaking each peripheral down into a set of individual features. For example, a (simplified) UART peripheral module may have three features, as shown in the following diagram.



Every feature of a peripheral will have one or more primitive operations that can be performed using that feature. These operations are named in a way that identifies the module, feature, and operation and the fact that they are Peripheral Library functions, as shown in the following figure.

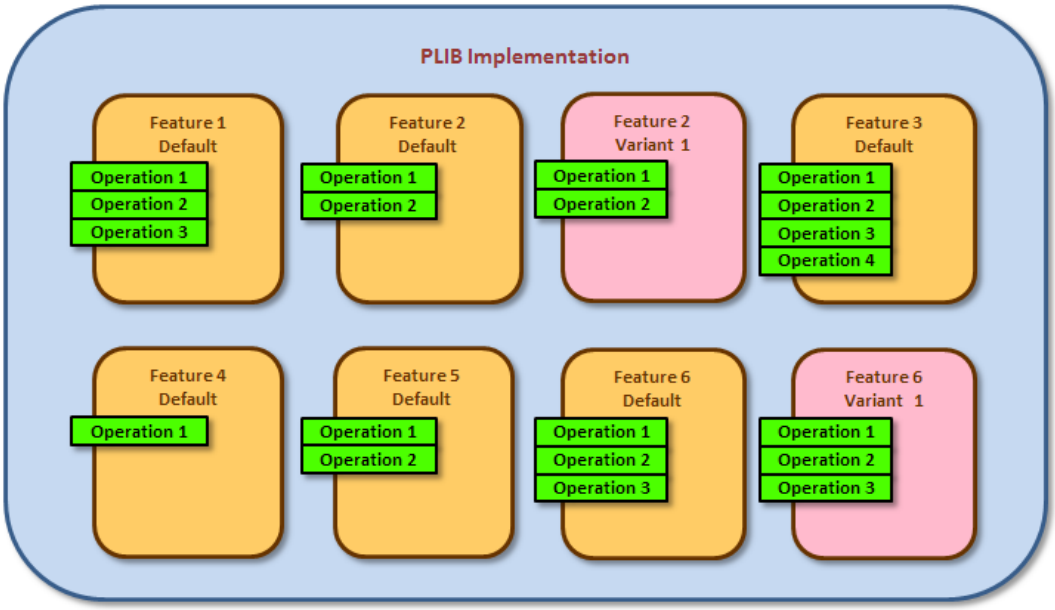
The simplified UART peripheral example has a "baud rate" feature. That feature may have two operations that it can perform. One operation might be a "set" function to set the current baud rate at which the UART will send and receive data and the other might be a "get" operation to find out the baud rate at which the UART is currently transmitting and receiving. Example C-language function signatures of these operations is as follows:

```
void    PLIB_USART_BaudRateSet( USART_MODULE_ID index, uint32_t clockFrequency,
                                uint32_t baudRate );

uint32_t    PLIB_USART_BaudRateGet( USART_MODULE_ID index, int32_t clockFrequency );
```

Notice that each function accepts an "index" parameter, as well as data parameters relevant to the operation itself, such as the input clock frequency and desired baud rate. The "index" parameter allows one set of PLIB functions to support any number of instances of the peripheral on a give microcontroller. Thus, each peripheral library exposes all of the features available on all instances of a given type of peripheral.

The function signature (name, parameters, and return value) for the operations supported by that feature will be the same on all microcontrollers even if the implementations of those functions are different from one microcontroller to another. These are known as "implementation variants". As illustrated in the following diagram, a PLIB implementation is made up of the complete set of default and variant implementations of the operations

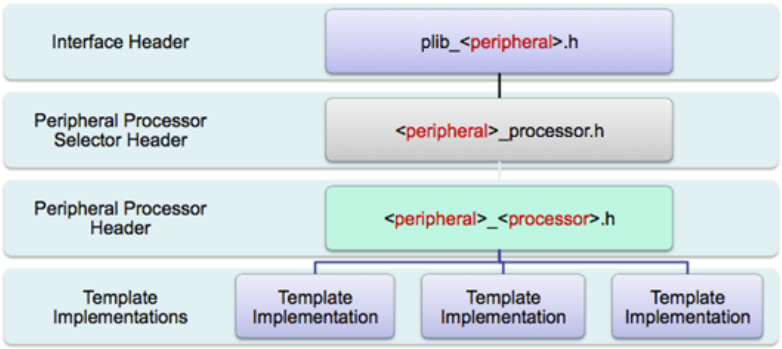
of every feature supported by a specific type of peripheral on a specific microcontroller.



Using this method, MPLAB Harmony Peripheral Libraries provide consistent C-language functional interfaces to access the peripherals available on all supported microcontrollers.

However, if a feature is not supported on all instances of the peripheral on all supported microcontrollers, the peripheral library communicates this to the developer in one of two ways. If a feature is not supported by any instance of the peripheral on a given microcontroller, the primitive functions related to that feature will also be unsupported and they will cause build warnings indicating this if they are called. If a feature is supported on at least one, but not on all of the available instances of a the peripheral on the microcontroller in use, the functions will be supported; however, they will provide a run-time error to facilitate debugging. Which features are supported on each instance of each microcontroller is documented in the help material for each peripheral library.

Peripheral libraries are primarily implemented as C-language "inline" functions. This allows the implementation of each function to be selected at build time, based upon the processor selection. To facilitate this, MPLAB Harmony peripheral libraries are implemented in layers of C-language header (`.h`) files, as shown in the following diagram.



For a given peripheral, "Interface Header" defines set of functions and data types (although, potentially not all of the data types, as discussed in a following section) that make up the interface to that peripheral library. To select the correct implementation for each processor, a "Peripheral Processor Selector Header" includes only the appropriate "Peripheral Processor Header" for the selected processor. This header then acts as a "mapping" header that translates the interface functions defined in the interface header to the correct implementation variant template for each function in the "Template Implementation" files.

This may seem a little confusing at first, but since all functions are implemented in header files as "inline" functions, the compiler can perform all of the mapping at build time, resulting in very small static implementations when static data is passed to the PLIB functions.

One more key concept is that, although peripheral libraries do provide a level of abstraction, they are still very low level. PLIBs combine accesses to multiple registers to implement a single operation when possible. They hiding register details from the caller and provide a consistent "functional" interface to a peripheral that hides differences in implementation variants. But, PLIBs do not manage the state of the peripheral to keep it running and they do not control access to the peripheral to prevent conflicts between different clients. Those jobs belong to the device drivers.

Fundamentally, peripheral libraries are peripheral access libraries. PLIB function calls do not block or call anything outside of the PLIB itself (with the exception of some removable debugging support). A PLIB function does not maintain any state data (outside of the data stored in hardware registers) as it may be called from within the main line of execution, from within an RTOS thread, or from within an ISR. And, PLIB functions are normally generated as "PLIB_INLINE", not as actual function calls (although optimized, prebuilt PLIB libraries are provided to support those times when the compiler does not generate the function "PLIB_INLINE"). A PLIB simply provides the ability to directly access and manipulate the features of a given peripheral using primitive operations. It is the responsibility of the calling module (usually the device driver) to store and appropriately protect any state data necessary to keep the peripheral running.

So, while an application (or any other layer) may directly access the PLIB for any particular peripheral in the system, if it does, it must be the only module in the system that does this and it then becomes responsible for the correct operation that peripheral instance in that system. That is why PLIBs are not normally the recommended way to access peripherals. It is usually better to access a peripheral through a driver or system service that manages the state machine of the peripheral and protects the peripheral so that accesses to it from multiple clients will not interfere with the correct operation of the peripheral.

Configuring a Library

The library is configured for the supported processor when the processor is chosen in the MPLAB X IDE.

Description

Peripheral library interfaces are common across all supported processors. But, their implementations are part specific and are provided in two forms that must normally be used together to avoid the possibility of build errors.

The first form of the PLIB implementation is as a set of C-language "inline" functions found in `.h` header files in the `<install-dir>/framework/peripheral` folder in the MPLAB Harmony installation. The correct implementation template files for the functions supported by each variant of a feature found on a selected processor is selected and included in the header file hierarchy when a PLIB interface header is included in a source file that uses it (either directly or by including `peripheral.h`). The second form in which the peripheral library implementations are provided is as a set of part-specific prebuilt binary library `.a` files (available in the `<install-dir>/bin/framework/peripheral` folder in the MPLAB Harmony installation.

To ensure that your MPLAB Harmony project builds correctly (regardless of the level of optimization selected), you must do two things. You must include the appropriate PLIB interface header file (either by including `peripheral.h` or the specific PLIB interface header) in any C-language file that calls a PLIB function. And, you must include the appropriate prebuilt binary library file (the one whose file name includes the appropriate part name) for the processor you selected in your MPLAB X IDE project.

Both of these steps are necessary because the compiler decides at build time if it will generate an actual function call to each PLIB function or if it will generate the function directly in line with the code that calls it. When the optimization level is high enough, the compiler will directly generate the PLIB function code inline with the calling code and no prebuilt library would be required. But, if the compiler "judges" that the function is too large to inline (or if optimizations are turned off), it will generate an actual function call. When this happens, the prebuilt library is required or the linker will generate an error when it tries to link the function call to an actual implementation. Since the compiler makes this determination for each function call it encounters, the safest choice is to always include the prebuilt binary `.a` file in your project.

Fortunately, the MPLAB Harmony installation provides prebuilt binary `.a` forms of the peripheral libraries that were built using a "-O3" level of optimization so that users of free versions of the compiler, which do not support this advanced level of optimization, can benefit from them. This level of optimization usually provides the best over-all compromise between code size and speed. However, if a different level of optimization is desired, users of the pro version of the compiler can rebuild the peripheral library `.a` file using the MPLAB X IDE project provided in the `<install-dir>/build/framework/peripheral` folder of the MPLAB Harmony installation.

To build the binary `.a` form of the peripheral libraries, the PLIBs require the project to define two macros before any file that includes any PLIB header. These macros are used to enable or disable the "inline" attribute on the peripheral library function definitions so that they can be built into a binary `.a` file that exports the PLIB API function symbols. These macros, their usage, and their default definitions are described as follows.

Macro:

PLIB_INLINE_API

Summary:

Determines if PLIB interface (API) functions are treated by the compiler as inline or extern (called) functions.

Description:

This macro is used as an attribute of every peripheral library interface (API) function. Its default definition allows the compiler to choose to either generate the function directly in line with the calling code or to generate an actual function call which must later be linked to an actual function implementation, based on the size of the code generated with the current optimization settings.

Remarks:

The default definition of this macro is:

```
#ifndef PLIB_INLINE_API
    #define PLIB_INLINE_API extern inline
#endif
```

To build a binary `.a` form of the peripheral libraries, define this macro as shown in the following example to export all PLIB API functions so that calls to them can be linked to the library generated.

```
#define PLIB_INLINE_API  extern
```

Macro:

PLIB_INLINE

Summary:

Determines if PLIB support functions are generated as static inline or extern inline functions.

Description:

This macro is used as an attribute of every peripheral library (PLIB) support function. PLIB support functions are functions that are called by the PLIB interface (API) functions, but are not themselves intended to be exported as PLIB interface functions. This macro's default definition prevents the compiler from generating superfluous error messages that would occur if the PLIB support functions were declared with a different scope (static versus extern) from the PLIB interface functions, which would occur when the PLIB headers are included in calling code.

This superfluous error message would occur because, the support functions do not need to be exported when a binary `.a` file is generated (and, thus should be declared with "static" scope). But, the compiler would detect the mismatch between the scope of the support functions when the API functions are declared with external scope when the PLIBs are directly included in calling code.

Remarks:

The default definition of this macro is:

```
#ifndef PLIB_INLINE
    #define PLIB_INLINE extern inline
#endif
```

To build a binary `.a` form of the peripheral libraries, define this macro as shown in the following example. This prevents the binary `.a` library from exporting all PLIB support functions, dramatically reducing the size of the generated library's symbol table.

```
#define PLIB_INLINE  static inline
```

Peripheral Library Porting Example

Peripheral Library Porting Example Help

Introduction

Provides an example on how to port a legacy (i.e., prior to MPLAB Harmony) USART Peripheral Library (PLIB) demonstration application to a MPLAB Harmony application using the MPLAB Harmony Configurator (MHC).

Description

This section describes the process to port the legacy UART PLIB Interrupt demonstration application (`<compiler-install-dir>/examples/plib_examples/uart/uart_interrupt`) to MPLAB Harmony.

The following assumptions are made:

- The PIC32MX795F512L device will be used; however, the process described in this section is applicable for other PIC32 devices with appropriate changes
- The Explorer 16 Development Board is the hardware used in this example
- For the v1.33 MPLAB XC32 C/C++ Compiler, the `examples` folder is not present. To view the legacy USART PLIB example, refer to v1.31 or earlier of the MPLAB XC32 C/C++ compiler.

Porting Procedure

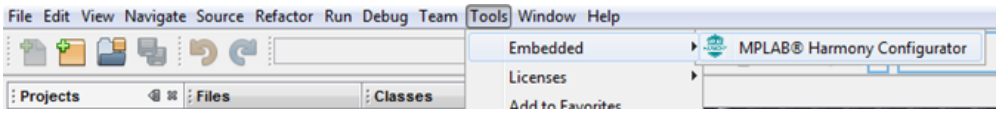
Describes the steps to set up the porting process.

Description

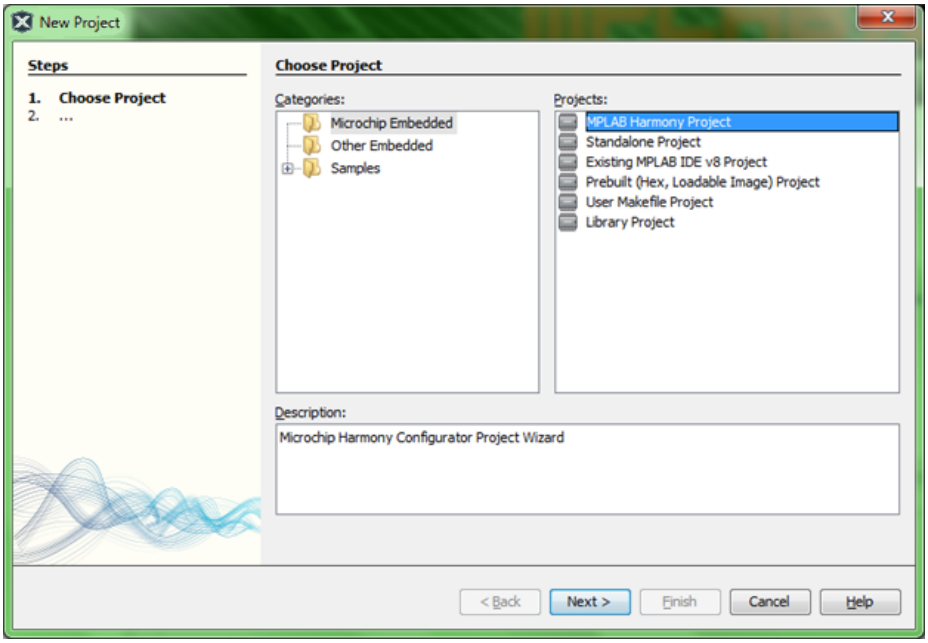
Step 1: Install the latest version of MPLAB Harmony. Throughout this porting guide, it is assumed that MPLAB Harmony is installed in its default location: `C:/microchip/harmony/<version>`. The `<version>` folder is assumed to be the root directory and all further steps will be explained relative to this root folder.

Step 2: Open MPLAB X IDE.

Step 3: Since the project will be created using the MHC, ensure that the MHC plug-in has been installed in MPLAB X IDE. You can verify the installation by selecting Tools > Embedded. If MHC is installed, you will see MPLAB Harmony Configurator is available as an option. Refer to Installing a Plug-in Module for information on installing the plug-in.

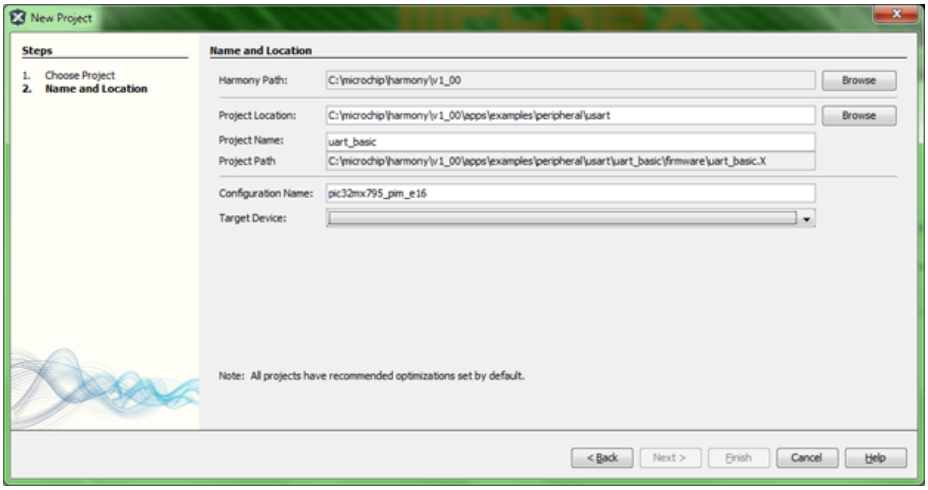


Step 4: Select *File > New Project* or click the New Project icon in MPLAB X IDE. In Categories, select **Microchip Embedded** and in Projects select **MPLAB Harmony Project** from the list of available project templates, and then click **Next** to launch the Microchip Harmony Configurator Project Wizard.

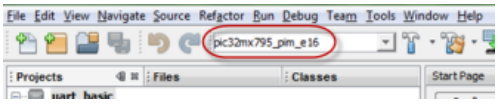


Step 5: Specify the following in the New Project dialog:

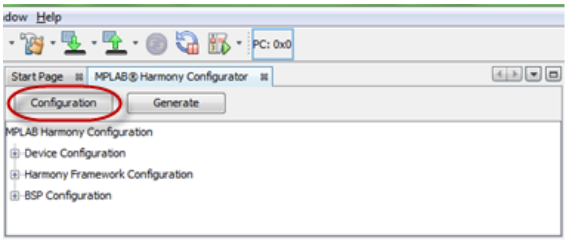
- Project Location: `<install-dir>/apps/examples/peripheral/usart`
- Project Name: uart_basic
- Configuration Name: pic32mx795_pim_e16
- Target Device of PIC32MX795F512L



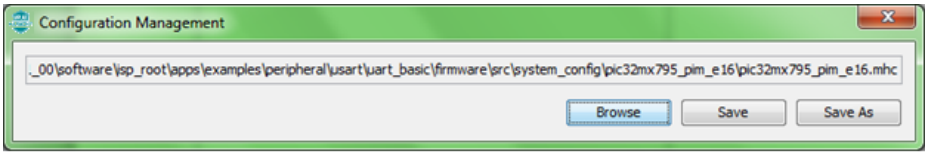
Step 6: Once the project is created, the configuration name will be set as "pic32mx795_pim_e16", as shown in the following figure.



Step 7: In the MPLAB Harmony Configurator tab, click **Configuration**.

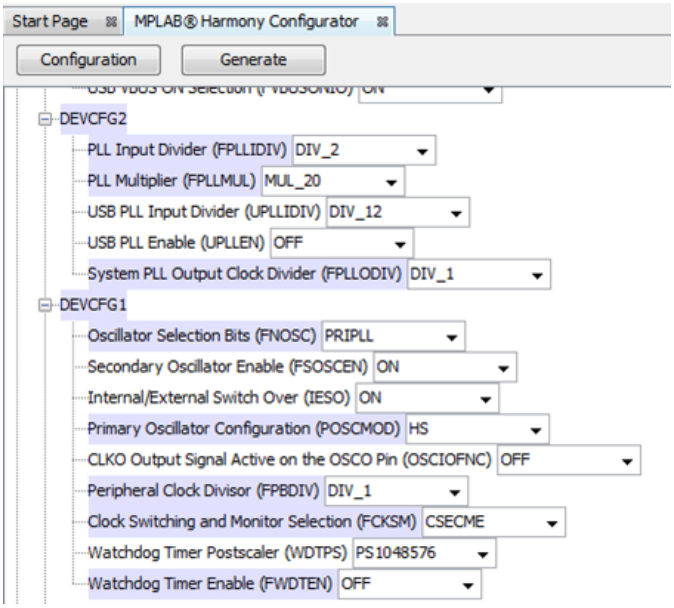


The Configuration Management dialog appears, which lists the folder path where the default configurator file (`.mhc`) will be saved. It is recommended not to modify the default path. Click **Save** to continue.

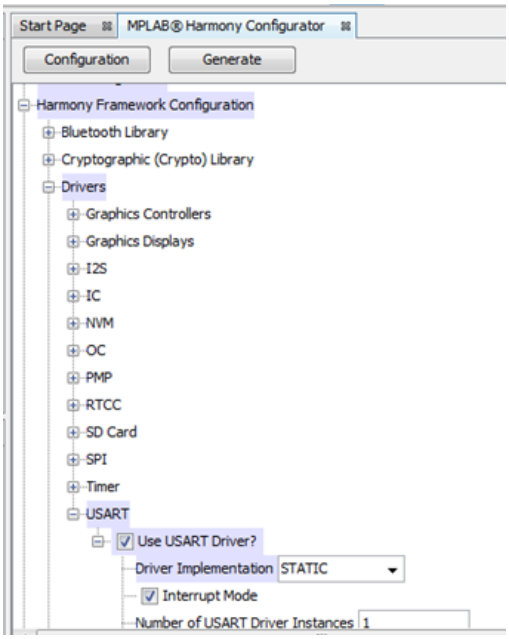


Step 8: In the MPLAB Harmony Configuration tab, configure the Device Configuration, Harmony Framework Configuration and BSP Configuration by selecting appropriate items from each drop down menu.

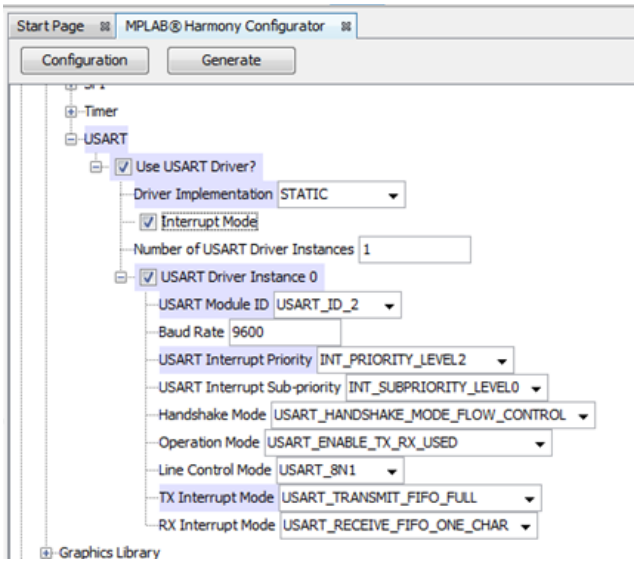
- Device Configuration: Since the controller should be running at 80 MHz with an 8 MHz external crystal, the configuration bits are set from the drop down menu, as shown in the following figure



- Harmony Framework Configuration: To enabler use of the UART Peripheral Library, the UART driver should be selected in the STATIC configuration in Interrupt mode, as shown in the following figure



- UART Configuration: Configure the UART2 with baud rate set to 9600, and interrupt enabled only for RX and Error. Also, set the other UART parameters, as shown in the following figure.

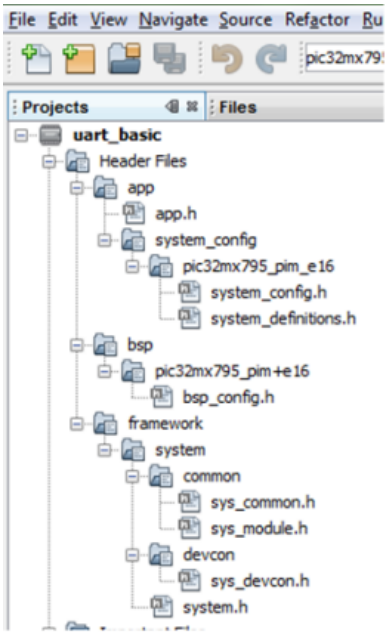


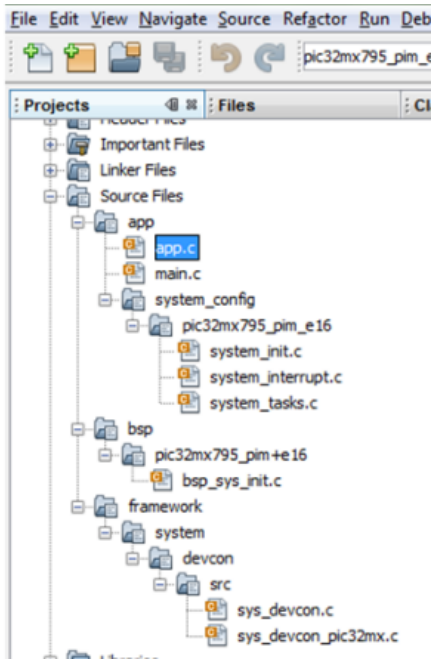
- BSP Configuration: Configure the BSP to use the PIC32MX795F512L and Explorer 16 Development Board, as shown in the following figure

Step 9: Click **Generate**. Since, the configurations were modified from the default values, MHC will ask whether or not the new configuration setting file (`.mhc`) should be saved. Click **Save** to continue.

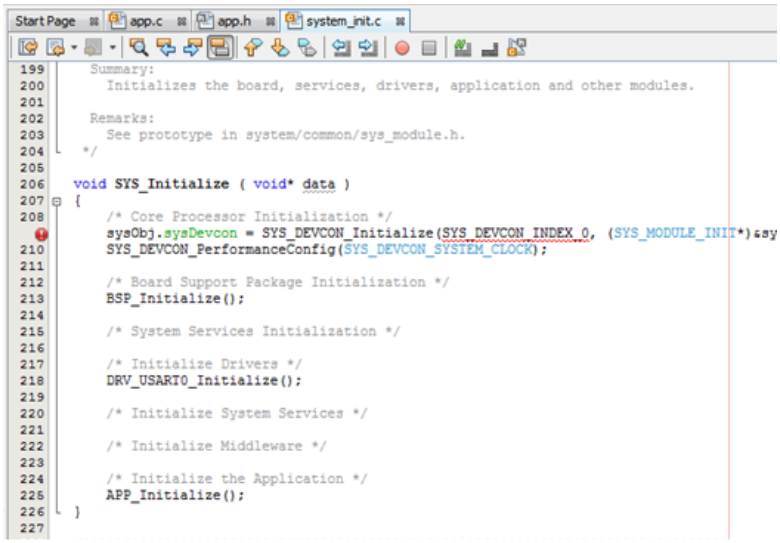
Step 10: Click **Generate**. Once the files are generated, the header, source, and library files will be added to the `uart_basic` project. The explanation of each logical folder is as follows:

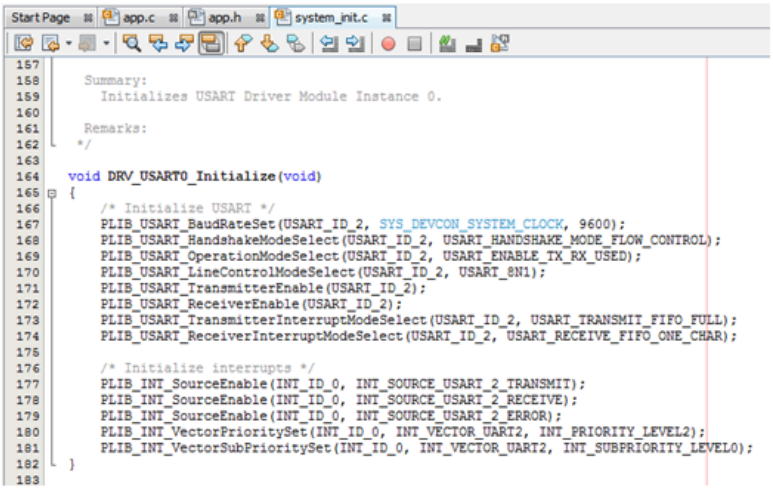
- `app` – Contains all application specific source files
- `bsp` – Contains all board specific source files
- `framework` – Contains all framework files from MPLAB Harmony
- `system_config` – Contains the system and application configuration, initialization, and Interrupt Service Routine (ISR)





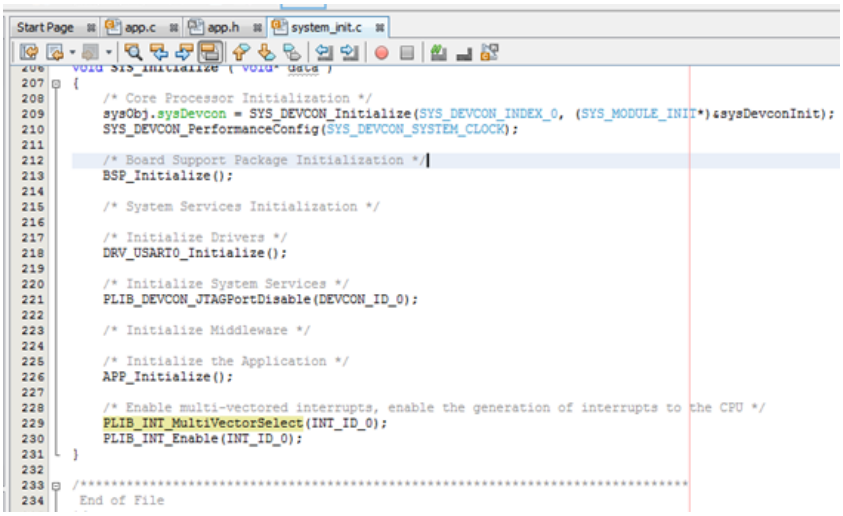
Step 11: At this point in the process, the `system_init.c` file will have functions/code to initialize the device clock and initialize the UART2. The code will be consistent to the settings previously done with MHC in **Step 8**.





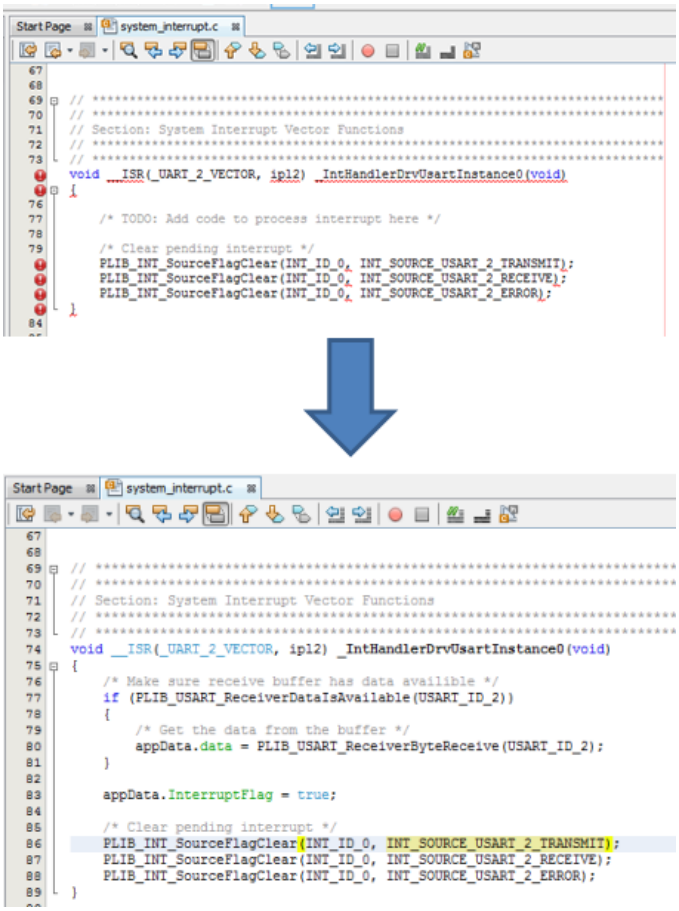
Step 12: Since the LEDs on the Explorer 16 Development Board are controlled by pins of PIC32MX795F512L device, which are shared with the JTAG function, JTAG should be disabled by manually adding the PLIB_DEVCON_JTAGPortDisable function in `system_init.c`. In addition, manually add the functions to set Multi-vector mode (PLIB_INT_MultiVectorSelect) and enable interrupt (PLIB_INT_Enable).

```
/* Initialize System Services */
PLIB_DEVCON_JTAGPortDisable(DEVCON_ID_0);

/* Enable multi-vectored interrupts, enable the generation of interrupts to the CPU */
PLIB_INT_MultiVectorSelect(INT_ID_0);
PLIB_INT_Enable(INT_ID_0);
```



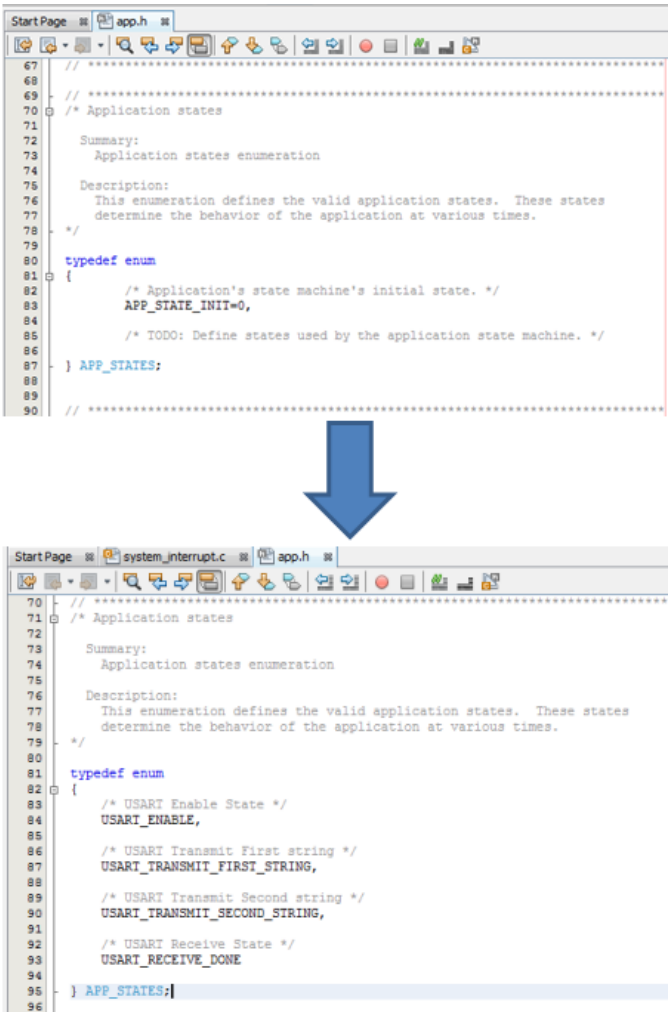
Step 13: In the ISR function of the `system_interrupt.c` file, add code to echo back the received character.

```
 /* Make sure receive buffer has data available */
    if (PLIB_USART_ReceiverDataIsAvailable(USART_ID_2))
    {
        /* Get the data from the buffer */
        appData.data = PLIB_USART_ReceiverByteReceive(USART_ID_2);
    }

    appData.InterruptFlag = true;
;
```



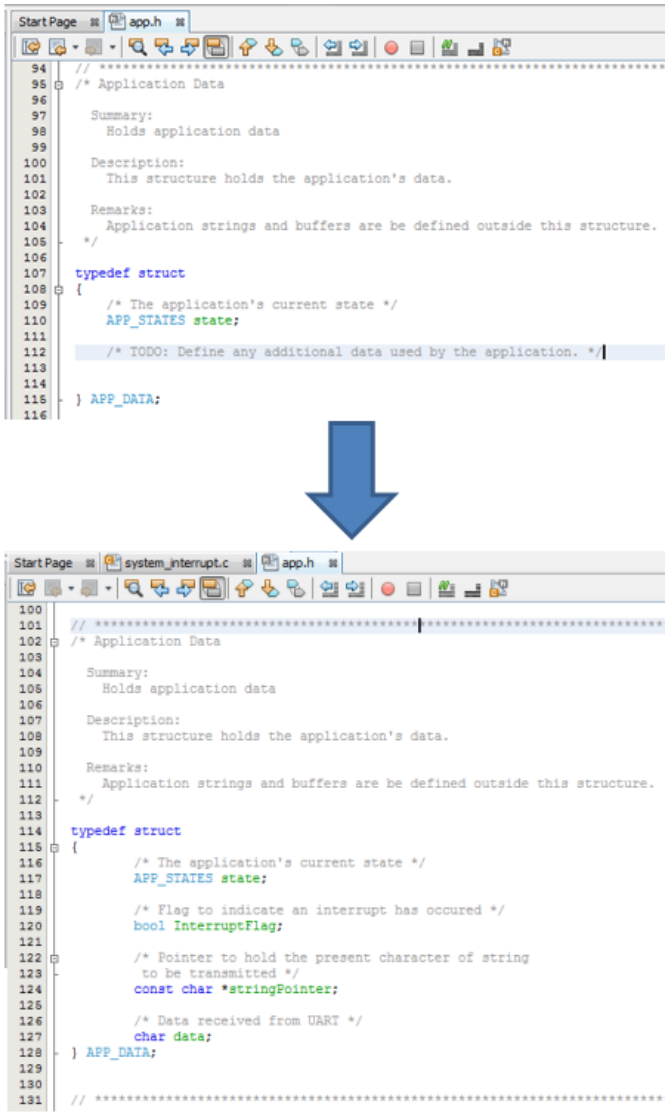
Step 14: In the file `app.h`, define the application-specific members of the APP_STATES enumeration.

```
 /* USART Enable State */
    USART_ENABLE,

 /* USART Transmit First string */
    USART_TRANSMIT_FIRST_STRING,

 /* USART Transmit Second string */
    USART_TRANSMIT_SECOND_STRING,

 /* USART Receive State */
    USART_RECEIVE_DONE
```



Step 15: For APP_DATA structure, define three variables. The first variable is of type Boolean to act as a flag for interrupt indication, the second pointer variable will hold the address of the string that needs to be transmitted to the UART, and the third variable will hold the data received from the UART.

```
const char *stringPointer;
char data;
bool InterruptFlag;
```

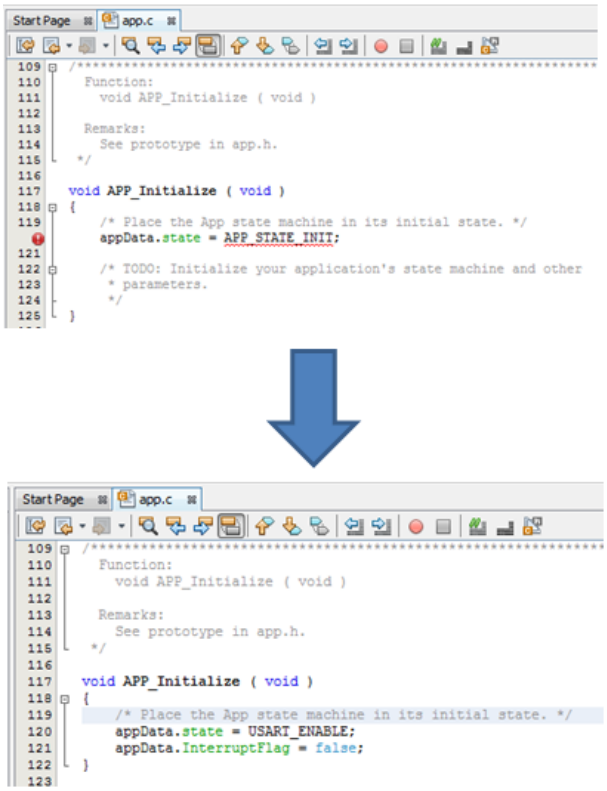


Step 16: Towards the end of the file, insert the prototype declaration for two functions: PutCharacter and WriteString. These functions will later be added to `app.c` (in **Step 20**). Also, declare the `extern APP_DATA appData`, so that the appData variable is available across the project.

```
bool PutCharacter(const char character);

bool WriteString(void);

extern APP_DATA appData;
```

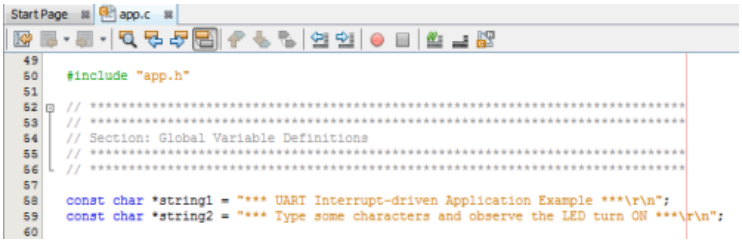
Step 17: In the APP_Initialize of the `app.c` file, add code to set the initial state of application.

```
    /* Place the App state machine in its initial state. */
    appData.state = USART_ENABLE;
    appData.InterruptFlag = false;
```



Step 18: Declare two global strings: string1 and string2.

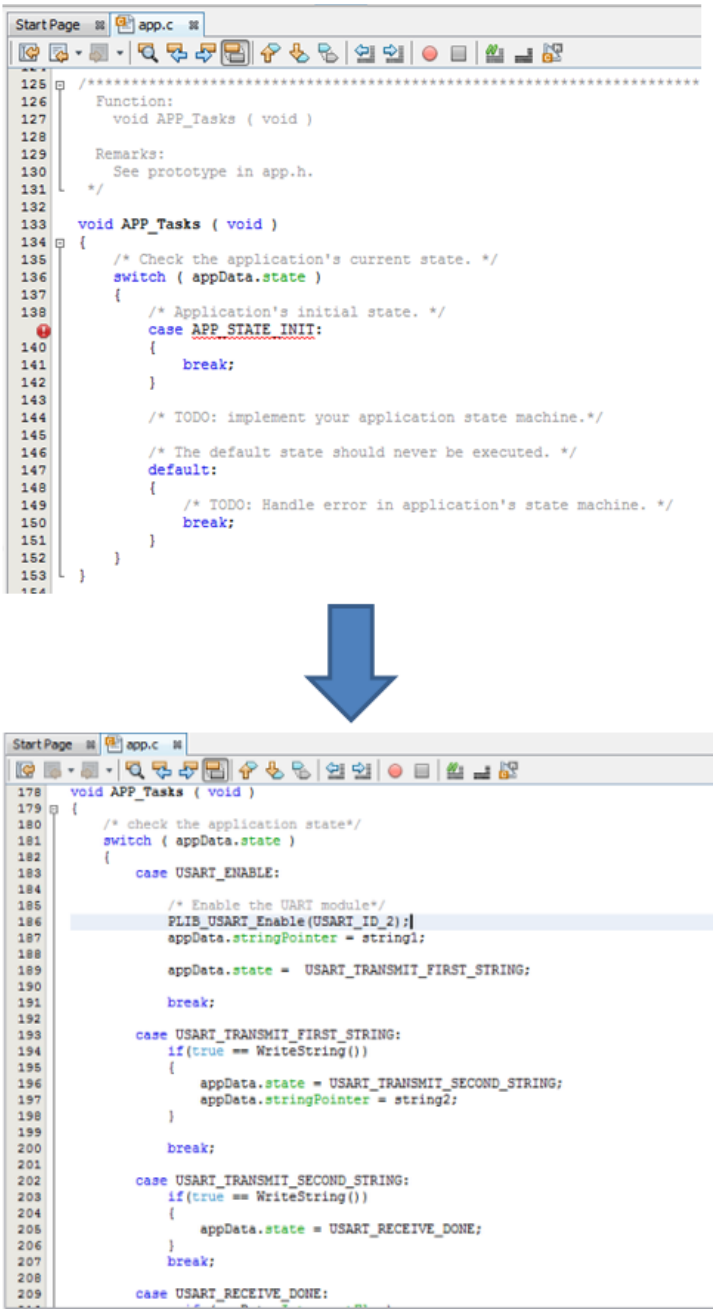
```
const char *string1 = "*** UART Interrupt-driven Application Example ***\r\n";
const char *string2 = "*** Type some characters and observe the LED turn ON ***\r\n";
```



Step 19: Next, add application-specific code to the APP_Tasks function.

```
void APP_Tasks ( void )
{
    /* check the application state*/
    switch ( appData.state )
    {
        case USART_ENABLE:

            /* Enable the UART module*/
            PLIB_USART_Enable(USART_ID_2);
            appData.stringPointer = string1;

            appData.state =  USART_TRANSMIT_FIRST_STRING;

            break;

        case USART_TRANSMIT_FIRST_STRING:
            if(true == WriteString())
            {
                appData.state = USART_TRANSMIT_SECOND_STRING;
                appData.stringPointer = string2;
            }

            break;

        case USART_TRANSMIT_SECOND_STRING:
            if(true == WriteString())
```

```
            {
                appData.state = USART_RECEIVE_DONE;
            }
            break;

        case USART_RECEIVE_DONE:
              if (appData.InterruptFlag)
              {
                  if(true == PutCharacter(appData.data))
                  {
                    BSP_LEDOn(BSP_LED_3);
                    appData.InterruptFlag = false;
                  }
              }
            break;

        default:
            SYS_DEBUG (SYS_ERROR_FATAL,"ERROR! Invalid state\r\n");
            while (1);

    }

}
```



Step 20: Next, add functions for PutCharacter and WriteString.

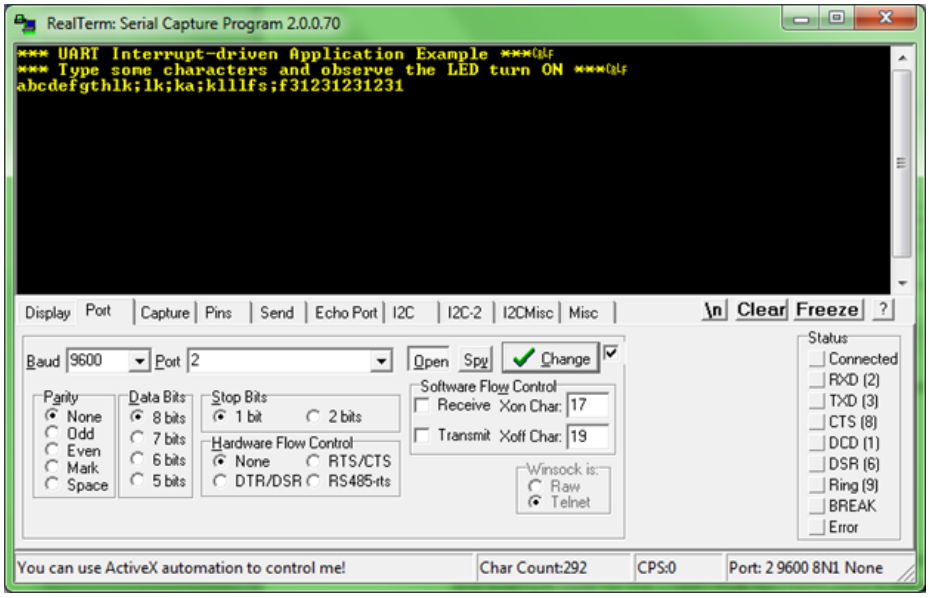
```
bool WriteString(void)
{
    if(*appData.stringPointer == '\0')
    {
        return true;
    }

    /* Write a character at a time, only if transmitter is empty */
    while (PLIB_USART_TransmitterIsEmpty(USART_ID_2))
    {
        /* Send character */
        PLIB_USART_TransmitterByteSend(USART_ID_2, *appData.stringPointer);

        /* Increment to address of next character */
        appData.stringPointer++;

        if(*appData.stringPointer == '\0')
        {
            return true;
```

```
        }
    }
    return false;
}

bool PutCharacter(const char character)
{
    /* Check if buffer is empty for a new transmission */
    if(PLIB_USART_TransmitterIsEmpty(USART_ID_2))
    {
        /* Send character */
        PLIB_USART_TransmitterByteSend(USART_ID_2, character);
        return true;
    }
    else
        return false;
}
```

```
/*******************************************************************************
  Function:
    bool WriteString (void)

  Summary:
    Writes a string to the console
*/
bool WriteString(void)
{
    if(*appData.stringPointer == '\0')
    {
        return true;
    }

    /* Write a character at a time, only if transmitter is empty */
    while (PLIB_USART_TransmitterIsEmpty(USART_ID_2))
    {
        /* Send character */
        PLIB_USART_TransmitterByteSend(USART_ID_2, *appData.stringPointer);

        /* Increment to address of next character */
        appData.stringPointer++;

        if(*appData.stringPointer == '\0')
        {
            return true;
        }
    }
    return false;
}

/*******************************************************************
  Function:
    bool PutCharacter (const char character)

  Summary:
    Sends a character to the console
*/
bool PutCharacter(const char character)
{
    /* Check if buffer is empty for a new transmission */
    if(PLIB_USART_TransmitterIsEmpty(USART_ID_2))
    {
        /* Send character */
        PLIB_USART_TransmitterByteSend(USART_ID_2, character);
        return true;
    }
    else
        return false;
}
```

Step 21: At this point in the process, all of the files have been added to the project with the proper code. Once the project is built, it should build without any errors or warnings.

Step 22: Once the device is programmed and run, a serial cable should be connected to the Explorer 16 Development Board. The appropriate COM port is selected with a baud rate of 9600. As shown in the following RealTerm example, the terminal will echo the typed characters and the LED on the Explorer 16 Development Board will illuminate.



Notes:

1. Refer to the USART Peripheral Library section in the MPLAB Harmony help for a detailed explanation and the example code for the peripheral library functions used in the application.
2. The demonstration code is provided at the same location `<install-dir>/apps/examples/peripheral/usart`.

ADC Peripheral Library

This section describes the Analog-to-Digital Converter (ADC) Peripheral Library.

Introduction

This library provides a low-level abstraction of the Analog-to-Digital Converter (ADC) module, which is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

An ADC is an analog peripheral that converts a continuous quantity to a discrete digital number. An ADC is a signal conversion process that periodically samples and converts a continuously varying signal - analog level into digital values. An ADC might be used to make an isolated measurement. ADCs are also used to quantize time-varying signals by turning them into a sequence of digital samples. This results in the signal being quantized in both time and value. The resolution of a converter indicates the number of discrete values it can produce over the range of analog values.

Using the Library

This topic describes the basic architecture of the ADC Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_adc.h`

The interface to the ADC library is defined in the `plib_adc.h` header file, which is included by the `peripheral.h` peripheral library header file. Any C language source (`.c`) file that uses the ADC library must include `peripheral.h`.

Library File:

The ADC peripheral library is part of the processor-specific peripheral library installed with the compiler. This library is automatically available (in the default search path) for any project built using a Microchip compiler.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

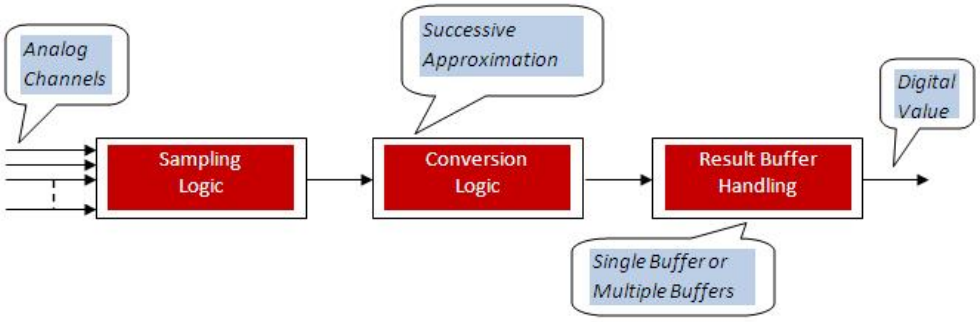
Hardware Abstraction Model

This library provides a low-level abstraction of the ADC module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The ADC module accepts an analog signal at any one instance and converts it to a corresponding 10-bit digital value. It can accommodate a number of analog inputs and separate reference inputs; the actual number available on a particular device depends on the package size.

Hardware Abstraction Block Diagram



A combination of input multiplexers can select the signal to be converted from multiple analog input pins. The entire multiplexer path includes provision for differential analog input, although the number of negative input pins is limited, and the signal difference must remain positive (i.e., unipolar).

Sampling Logic

An internal Sample and Hold (S&H) circuit acquires a sample of an input signal, and then holds that value constant during the conversion process. The purpose of the S&H circuitry is to take a snapshot of the sensor signal and hold the value. The sampled voltage is held and converted to a digital value, which strictly speaking, represents the ratio of that input voltage to a reference voltage. Configuration choices can allow connection of an external reference or use of the device power and ground (AVDD and AVSS).

Conversion Logic

The heart of the ADC is the conversion logic that converts the analog signal value into its equivalent discrete representation. Conversions can be started individually by program control, continuously free-running, or triggered by selected hardware events. A single channel may be repeatedly converted, alternate conversions may be performed on two channels, or any or all of the channels may be sequentially scanned and converted according to a user-defined bit map.

Result Handling

The resulting conversion output is 10-bit digital number in a 16-bit word.

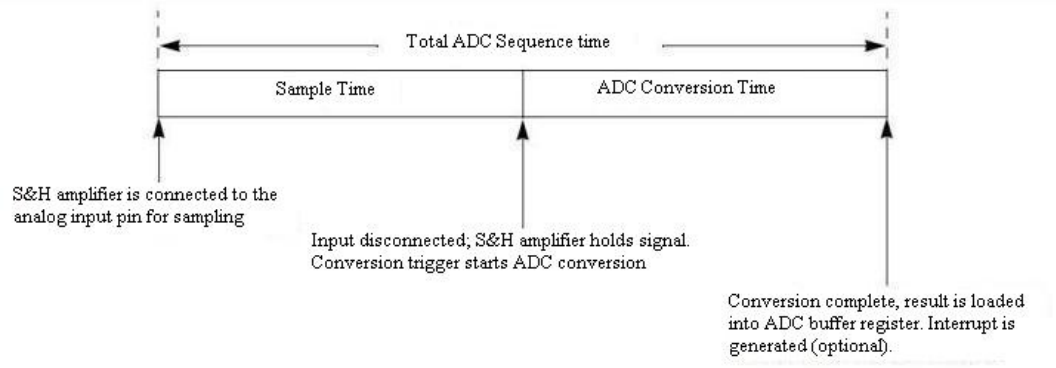
ADC Timing Details

Sample time is the time that the ADC module's S&H circuit is connected to the analog input pin. The sample time may be started and ended automatically by the ADC's hardware or under direct program control. There is a minimum sample time to ensure that the S&H circuit will provide sufficient accuracy for the analog-to-digital conversion.

Conversion time is the time required for the ADC to convert the voltage held by the S&H circuit. The conversion trigger ends the sampling time and begins an analog-to-digital conversion or a repeating sequence. The conversion trigger sources can be taken from a variety of hardware sources or can be controlled directly in software.

Once the conversion is complete, the S&H circuit can be reconnected to the input pin and a CPU interrupt may be generated. The sum of the sample time and the analog-to-digital conversion time provides the total ADC sequence time. The following figure shows the basic conversion sequence and the relationship between intervals.

ADC Sample/Convert Sequence



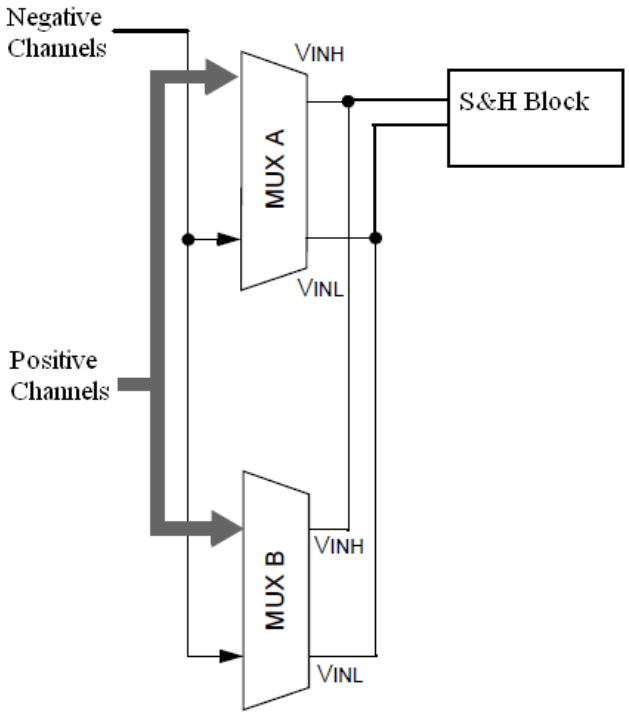
The conversion trigger sources can be taken from a variety of hardware sources, or can be controlled directly by software. One of the conversion trigger options is an auto-conversion, which uses a counter and the ADC clock to set the time between auto-conversions. The Auto-Sample mode and auto-conversion trigger can be used together to provide continuous automatic conversions without software intervention.

A sample/convert sequence that uses multiple S&H channels can be simultaneously sampled or sequentially sampled. Simultaneously sampling multiple signals ensures that the snapshot of the analog inputs occurs at precisely the same time for all inputs. Sequential sampling takes a snapshot of each analog input just before conversion starts on that input. The sampling of multiple inputs is not correlated.

Channel Multiplexers

On some devices, S&H circuits have analog multiplexers on both their non-inverting and inverting inputs to select which analog input(s) are sampled. The ADC of some devices incorporate two independent sets of input multiplexers (MUX A and MUX B), which allow users to choose the analog channels that are to be sampled. Functionally, MUX A and MUX B are very similar to each other. Both multiplexers allow any of the analog input channels to be selected for individual sampling and allow selection of a negative reference source for differential signals. In addition, MUX A can be configured for sequential analog channel scanning. By default the ADC only samples and converts the inputs selected by MUX A. There is also a possibility to alternate between two sets of inputs selected by MUX A and MUX B during successive samples.

MUX Abstraction Model



When using MUX A to select analog inputs, the ADC module has the ability to scan multiple analog channels sequentially.

Input Selection

The ADC module provides a flexible mechanism to select analog inputs for conversion:

- Fixed input selection
- Alternate input selection
- Channel scanning

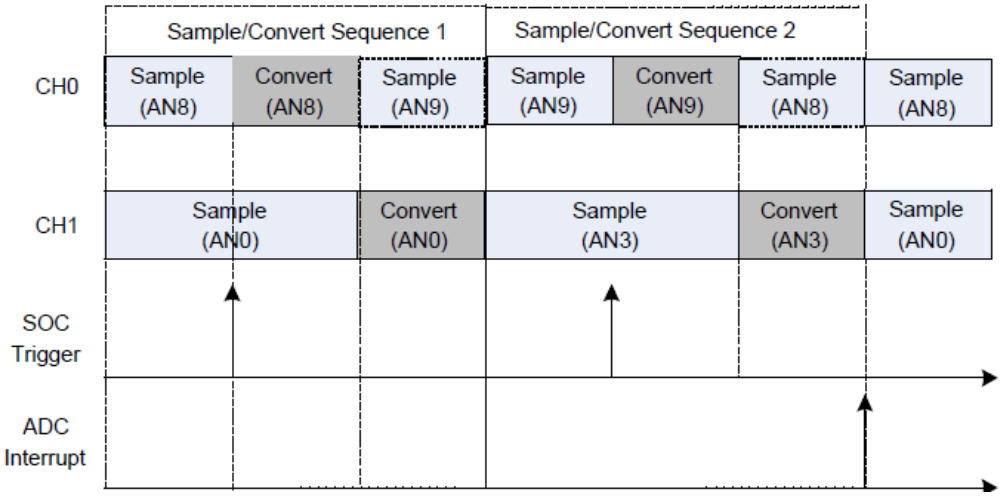
Fixed Input Selection

This is achieved through one or more of the S&H channels available in the device. The S&H channels are connected to the analog input pins through the analog multiplexer.

Alternate Sampling

In an Alternate Input Selection mode, the ADC completes one sweep using the MUX A selection, and then another sweep using the MUX B selection, and then another sweep using the MUX A selection, and so on.

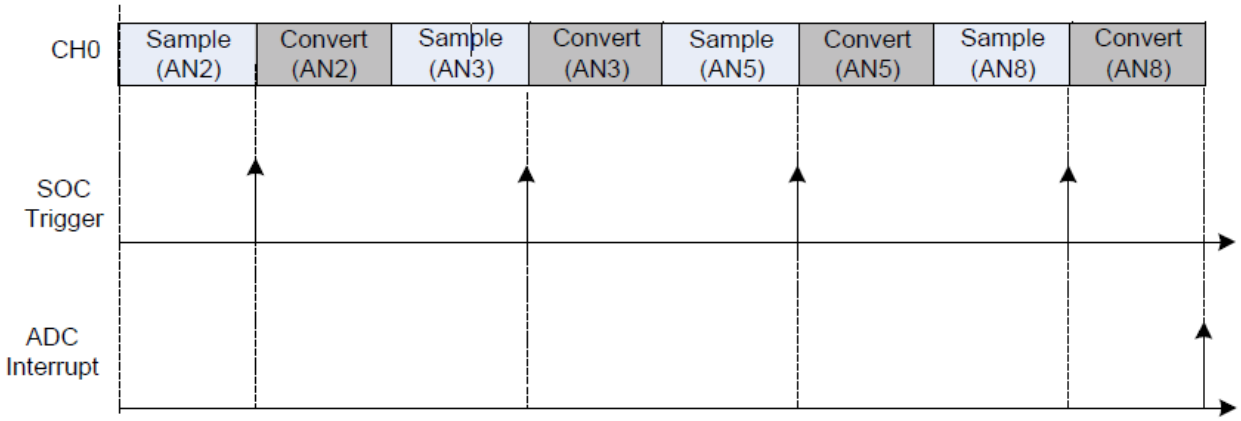
Alternate Input Selection in 2-Channel Sequential Sampling Configuration



Channel Scanning

On some devices, the ADC module supports the Channel Scan mode using S&H Channel 0 (CH0). The number of inputs scanned is software selectable. Any subset of the analog inputs from AN0 to AN31 (depending on the number of analog inputs present on a specific device) can be selected for conversion. The selected inputs are converted in ascending order. For example, if the input selection includes AN4, AN1, and AN3, the conversion sequence is AN1, AN3, and AN4.

Scan Four Analog Inputs Using CH0



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the ADC Peripheral Library.

Library Interface Section	Description
General Configuration	Interface routines for: • Voltage reference selection (positive/negative) • Channel group selection • Positive/negative channel selection • Add/remove channels for scan • Enabling/disabling the ADC module • Enabling/disabling the calibration • Stop in Idle enable/disable • Internal reference channel enable/disable
MUX Selection and Channel Scan	Interface routines for: • Channel 0 positive/negative input selection for MUX A/MUX B • MUX A scan enable/disable

Sample and Hold Control Logic	Interface routines for: • Sampling start/stop, status • Enabling/disabling of sample auto start • Acquisition/auto-sample time selection • Sample per interrupt selection
Conversion Control Logic	Interface routines for: • Conversion start/status • Conversion clock selection set/get • Conversion clock source selection • Conversion trigger source selection • Conversion stop sequence enable/disable
Output Configuration	Interface routines for: • Result format selection • Result buffer fill status • Result buffer mode select • Result based on the buffer index
Feature Existence Functions	These functions determine whether or not a particular feature is supported by the device.

How the Library Works

How the Library Works

The following processes are involved while using an ADC module:

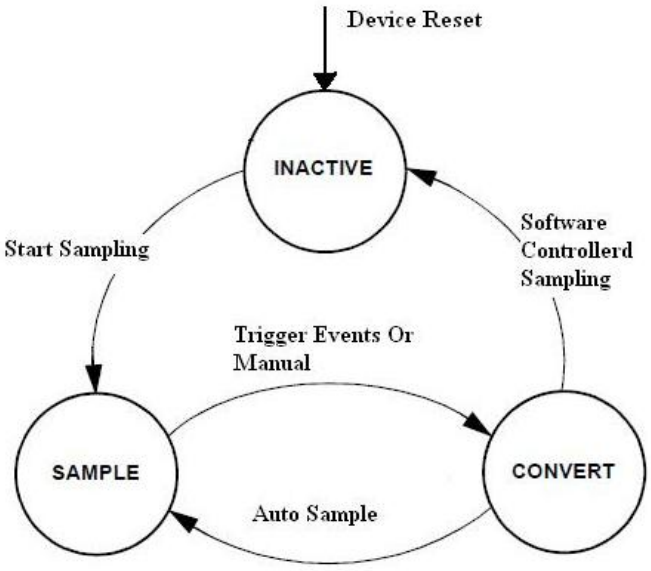
- Initialization
- Controlling the sampling process
- Controlling the conversion process
- Accessing the results buffer

General

The ADC conversion process can be thought of in terms of a finite state machine. The sample state represents the time that the input channel is connected to the S&H circuit and the signal is passed to the converter input. The convert state is transitory; the module enters this state as soon as it exits the sample state and transitions to a different state when that is done. The inactive state is the default state prior to module initialization and following a software-controlled conversion; it can be avoided in operation by using Auto-Sample mode.

Description

ADC Conversion Sequence or State Machine



Initialization

This topic provides information on the different processes needed to perform an analog-to-digital conversion.

Description

The following processes should be followed for performing an analog-to-digital conversion:

Initialize the ADC Module:

Number	Description	Functions associated
1	Selecting the voltage reference source Idle mode control	PLIB_ADC_VoltageReferenceSelect PLIB_ADC_StopInIdleEnable PLIB_ADC_StopInIdleDisable
2	Selecting the ADC conversion clock	PLIB_ADC_ConversionClockSet
3	Input channel selection Configuring MUX A and MUX B inputs, Alternating MUX A and MUX B input selections, Scanning through several inputs	**Scan Mask Selection** PLIB_ADC_InputScanMaskAdd PLIB_ADC_InputScanMaskRemove **Positive Inputs** PLIB_ADC_MuxChannel0InputPositiveSelect **Negative Inputs** PLIB_ADC_MuxChannel0InputNegativeSelect **Scan Mode Selection** PLIB_ADC_MuxAInputScanEnable PLIB_ADC_MuxAInputScanDisable
4	Enabling the ADC module	PLIB_ADC_Enable
5	Determine how sampling will occur	**Sampling Control** PLIB_ADC_SamplingModeSelect PLIB_ADC_SampleAcquisitionTimeSet
6	Selecting Manual or Auto-Sampling	PLIB_ADC_SampleAutoStartEnable PLIB_ADC_SampleAutoStartDisable PLIB_ADC_SamplingStart
7	Select conversion trigger and sampling time	PLIB_ADC_ConversionStart PLIB_ADC_ConversionClockSourceSelect PLIB_ADC_ConversionTriggerSourceSelect PLIB_ADC_ConversionStopSequenceEnable PLIB_ADC_ConversionStopSequenceDisable
8	Select how conversion results are stored in buffer	PLIB_ADC_ResultBufferModeSelect
9	Select the result format	PLIB_ADC_ResultFormatSelect
10	Select the number of readings per interrupt	PLIB_ADC_SamplesPerInterruptSelect

The ADC is configured by the following steps:

1. Select the acquisition time using PLIB_ADC_SampleAcquisitionTimeSet.
2. Select the conversion clock for ADC using PLIB_ADC_ConversionClockSet.
3. The reference for ADC can be set using the function PLIB_ADC_VoltageReferenceSelect.
4. Select the appropriate analog MUX and analog input where the analog voltage is connected.
5. Select the appropriate trigger source using PLIB_ADC_ConversionTriggerSourceSelect.
6. Configure the ADC interrupt (if required):

- Clear the interrupt status flag
- Select the ADC interrupt priority
- Enable ADC interrupt

7. Turn on the ADC module using PLIB_ADC_Enable.

Example Initialization:

```
// Include all channels in scan
PLIB_ADC_InputScanMaskAdd(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Internal Counter triggers conversion
```

```
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Set the interrupt every 16 samples
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_16SAMPLES_PER_INTERRUPT);

// Enable scanning of channels
PLIB_ADC_MuxAInputScanEnable(MY_ADC_INSTANCE);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);
```

Note: Not all functionality is available on all devices. Refer to the **"Analog-to-Digital Converter (ADC)"** chapter in the specific device data sheet for availability.

Controlling the Sampling Process

This topic describes the modes and related sampling functionality for the sampling process.

Description

The sampling process can be set up using the following two modes:

Manual Sampling

Calling PLIB_ADC_SamplingStart causes the ADC to begin sampling. One of several options as discussed in "Controlling the Conversion Process" can be used to end sampling and complete conversions. Sampling does not resume until PLIB_ADC_SamplingStart is called again.

Automatic Sampling

Setting the ADC in the Auto-Sampling mode using PLIB_ADC_SampleAutoStartEnable automatically begins sampling a channel whenever a conversion is not active on that channel. One of several options can be used to end sampling and complete conversions, as discussed in "Controlling the Conversion Process". If the simultaneous sampling is selected using PLIB_ADC_SamplingModeSelect with parameter ADC_SAMPLING_MODE_SIMULTANEOUS, sampling on a channel resumes after the conversion of all channels completes.

Other sampling related functionality:

- **Monitoring Sample Status:** PLIB_ADC_SamplingIsActive obtains the status as sampling or holding for the ADC module.
- **Aborting a Sample:** While in Manual Sampling mode, calling PLIB_ADC_SamplingStop will terminate sampling. If the conversion trigger source is selected as ADC_CONVERSION_TRIGGER_SAMP_CLEAR using PLIB_ADC_ConversionTriggerSourceSelect, this causes a conversion to start automatically. While in Auto-Sampling mode, calling PLIB_ADC_SampleAutoStartEnable does not terminate an outgoing sample/convert sequence; however, sampling will not resume after a subsequent conversion.
- **Sampling Modes:** Different sampling modes can be changed using PLIB_ADC_SamplingModeSelect with the appropriate parameter such as ADC_SAMPLING_MODE_ALTERNATE_INPUT for alternate input mode, ADC_SAMPLING_MODE_SIMULTANEOUS for simultaneous sampling mode, or ADC_SAMPLING_MODE_SEQUENTIAL for the sequential sampling mode. There is a possibility to combine the sampling modes say the Alternate input mode with either the simultaneous or the sequential sampling modes.

Note: Not all functionality is available on all devices. Refer to the **"Analog-to-Digital Converter (ADC)"** chapter in the specific device data sheet for availability.

Controlling the Conversion Process

The conversion trigger source will terminate sampling and start a selected sequence of conversions. It is also possible to obtain the value of the conversion clock, which is obtained by calling PLIB_ADC_ConversionClockGet. 'PLIB_ADC_ConversionTriggerSourceSelect' selects the source of the conversion trigger.

Description

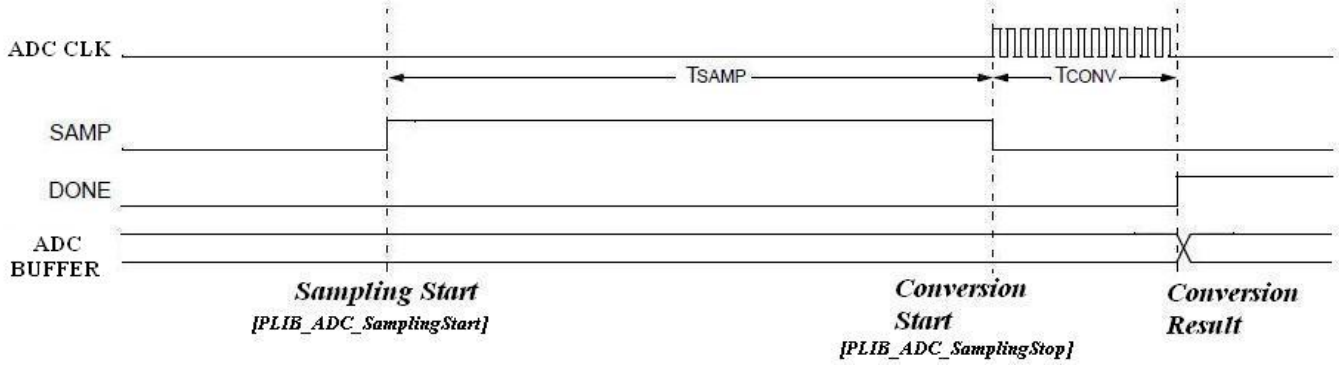
Conversion can be started in one of the following three ways:

Manual Conversion Sequence

Manual Sample Start, Manual Conversion Start

When ADC_CONVERSION_TRIGGER_SAMP_CLEAR is selected using PLIB_ADC_ConversionTriggerSourceSelect, the conversion trigger is under software control. Calls to PLIB_ADC_SamplingStop will stop the sampling and start the conversion sequence. The user must call PLIB_ADC_SamplingStart and PLIB_ADC_SamplingStop in a timed manner, to ensure adequate sampling time.

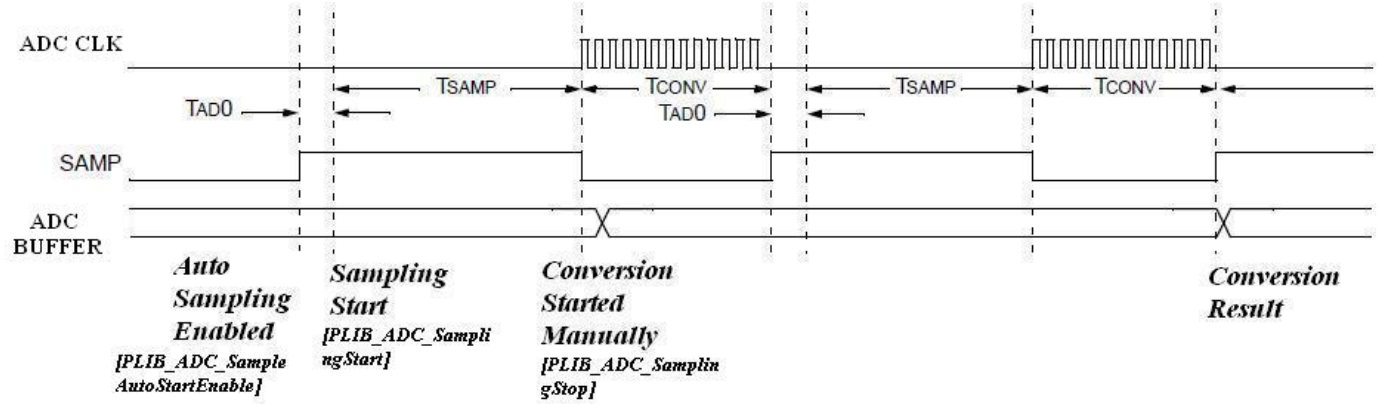
Converting One Channel, Manual Sample Start, Manual Conversion Start



Automatic Sample Start, Manual Conversion Start

Automatic sampling is initiated using PLIB_ADC_SampleAutoStartEnable, calling PLIB_ADC_SamplingStop will terminate sampling and start conversion. After the conversion, the sampling starts again automatically. The user must call PLIB_ADC_SamplingStop in a timed manner, to ensure adequate sampling time. Wait for required acquisition/auto sample time (minimum of 1 TAD), and then call PLIB_ADC_SamplingStop to start the conversion process.

Converting One Channel, Automatic Sample Start, Manual Conversion Start

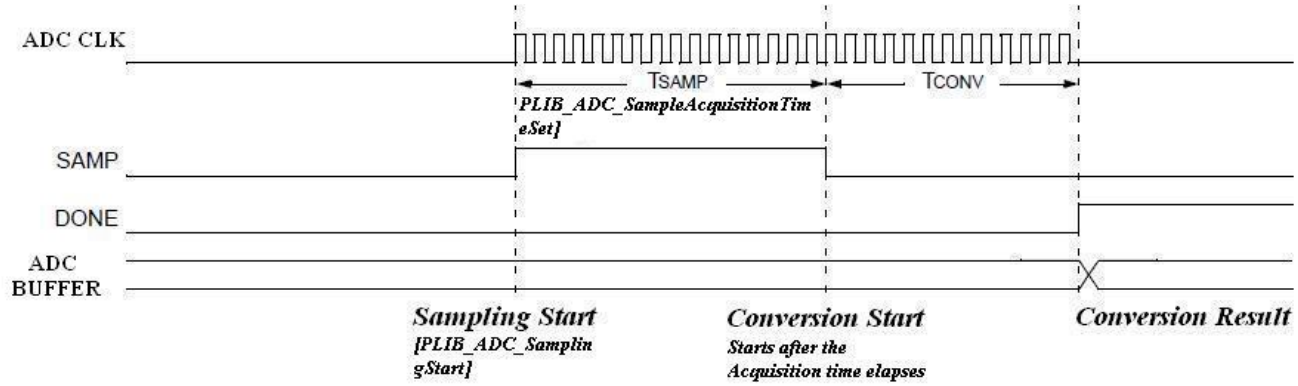


Clocked Conversion Sequence

Manual Sample Start and TAD based Conversion Start

When ADC_CONVERSION_TRIGGER_INTERNAL_COUNT is selected using PLIB_ADC_ConversionTriggerSourceSelect, the conversion trigger is under analog-to-digital clock control. PLIB_ADC_SampleAcquisitionTimeSet selects the TAD clock cycles between the start of sampling and the start of conversion. [Minimum 1 clock cycle has to be selected to ensure the sampling requirements are met]. PLIB_ADC_SamplingStart starts the sampling for the configured acquisition time, and then PLIB_ADC_SamplingStop starts the conversion process.

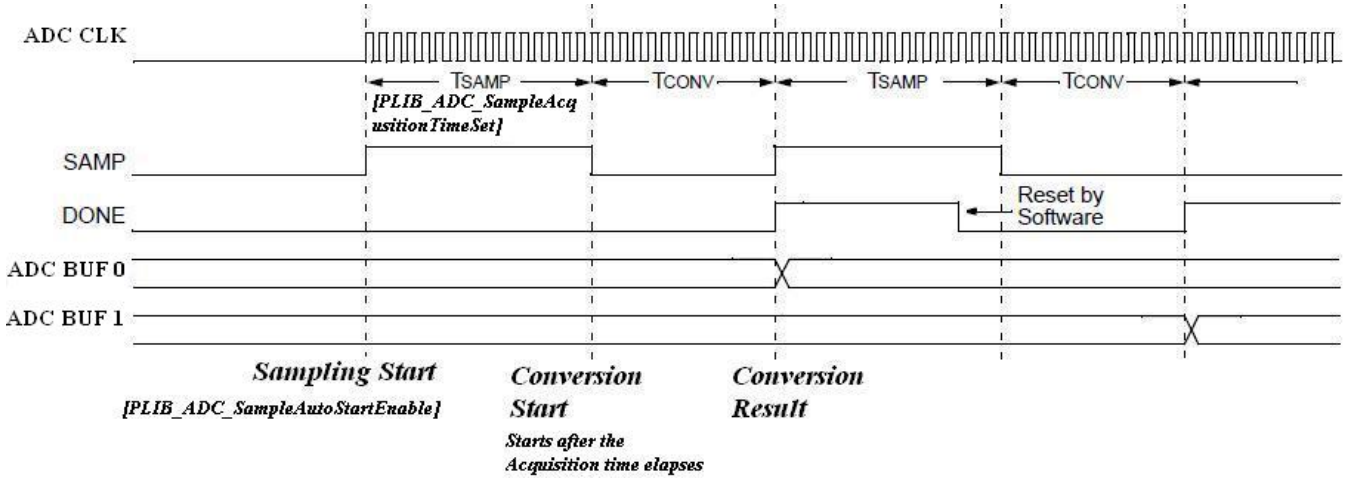
Converting One Channel, Manual Sample Start, TAD-Based Conversion Start



Auto Sample Start and TAD based Conversion Start or Free Running Sample Conversion

With the selection of ADC_CONVERSION_TRIGGER_INTERNAL_COUNT using PLIB_ADC_ConversionTriggerSourceSelect and Automatic sampling initiation using PLIB_ADC_SampleAutoStartEnable allows the ADC module to schedule sample/conversion sequences with no intervention by the user or other device resources.

Converting One Channel, Auto-Sample Start, TAD-Based Conversion Start



Event Trigger Conversion Start

It may be necessary to synchronize the end of sampling and the start of conversion with some other time event. The ADC module may use one of following as conversion trigger sequences:

- **External Pin Trigger:** When ADC_CONVERSION_TRIGGER_INT0_TRANSITION is selected using PLIB_ADC_ConversionTriggerSourceSelect
- **General Purpose Timer Compare Match:** When ADC_CONVERSION_TRIGGER_TMR3_COMPARE_MATCH or ADC_CONVERSION_TRIGGER_TMR5_COMPARE_MATCH is selected using PLIB_ADC_ConversionTriggerSourceSelect

Both of the event trigger conversion modes previously described can be used in combination with auto-sampling (PLIB_ADC_SampleAutoStartEnable) to cause the ADC to synchronize the sample conversion events to the trigger pulse source.

Users should note that some devices have additional conversion trigger sources as part of the enumeration ADC_CONVERSION_TRIGGER_SOURCE.

Other Operations During Conversion Process

- **Monitoring Conversion process:** The status of the conversion can be obtained using PLIB_ADC_ConversionHasCompleted
- **Generating ADC Interrupt:** PLIB_ADC_SamplesPerInterruptSelect controls the generation of the ADC interrupt. To enable the interrupt it is also essential to enable the ADC interrupt.
- **Aborting the Conversion:** Calling PLIB_ADC_Disable will abort the current conversion. The result buffer is not updated with the partially completed ADC conversion sequence.

Timing Details

TAD - The ADC module has a maximum rate at which conversions may be completed. An analog module clock, TAD, controls the conversion timing.

TSAMP - The time required to sample and hold the sampled analog signal, configured through PLIB_ADC_SampleAcquisitionTimeSet.

TCONV - The time required to convert the sampled analog signal, configured through PLIB_ADC_ConversionClockSet. The conversion clock can be verified using PLIB_ADC_ConversionClockGet.

Note: Not all functionality is available on all devices. Refer to the **"Analog-to-Digital Converter (ADC)"** chapter in the specific device data sheet for availability.

Accessing the Result Buffers

The result buffers can be formatted to the desired format using the function PLIB_ADC_ResultFormatSelect.

Description

As the analog-to-digital conversions are completed, the ADC module writes the results of the conversions into the ADC result buffer. This buffer is a RAM array of 16 words, accessed through the Special Function Register (SFR) space.

User software may attempt to read each ADC conversion result as it is generated; however, this might consume too much CPU time. Generally, to minimize software overhead, the ADC module will fill the buffer with results, and then generate an interrupt when the buffer is filled. There are two different modes for accessing the result buffers.

Note: Refer to the **"Analog-to-Digital Converter (ADC)"** chapter in the specific device data sheet to determine the correct mode for accessing the result buffer for your device.

Single Buffer Mode

Conversion results are automatically stored in a dedicated buffer. The module sets its interrupt flag after the conversion is complete, it also marks the conversion status as complete. After the interrupt, the conversion sequence can restart. The converted values are available through PLIB_ADC_ResultGetByIndex.

Multiple Buffer Mode

Conversion results are automatically stored in a dedicated position in the result buffer comprising an array of 16 words, allowing for multiple successive readings to be taken before software service is needed. Successive conversions are placed into sequential buffer positions. Alternatively, the buffer can be split into two arrays of 8 words for simultaneous conversion and read operations. The ADC module sets its interrupt flag after a selectable number of conversions, from 1 to 16, when the all buffer positions can be read. After the interrupt, the sequence restarts at the beginning of the buffer. When the interrupt flag is set, scan selections and the output buffer pointer return to their starting positions. The ADC result buffer is a set of 16 words, accessed through PLIB_ADC_ResultGetByIndex , where buffer index can be can be any value ranging from 0 to 15.

PLIB_ADC_SamplesPerInterruptSelect selects how many analog-to-digital conversions will take place before the CPU is interrupted.

The buffer fill mode can be selected using PLIB_ADC_ResultBufferModeSelect with the parameter of the type ADC_BUFFER_MODE.

When the conversion result buffer is split (ADC_BUFFER_MODE_TWO_8WORD_BUFFERS used as the parameter for PLIB_ADC_ResultBufferModeSelect), PLIB_ADC_ResultBufferStatusGet indicates which half of the buffer is being currently written by the ADC module.

Note: Not all functionality is available on all devices. Refer to the **"Analog-to-Digital Converter (ADC)"** chapter in the specific device data sheet for availability.

Power-Saving Modes

This topic provides information on the power-saving modes available for use with the ADC module.

Description

Operation in Sleep Mode

Operation in Sleep mode requires that the internal RC clock is selected using PLIB_ADC_ConversionClockSourceSelect with the parameter ADC_CLOCK_SOURCE_INTERNAL_RC. If the ADC interrupt is enabled, the device will wake up from Sleep mode on the ADC interrupt. On some microcontrollers, operation in Sleep mode requires that ADC_CONVERSION_CLOCK_FRC is selected using PLIB_ADC_ConversionClockSourceSelect. If the ADC interrupt is enabled, the device will wake up from Sleep mode on the ADC interrupt.

Operation in Idle Mode

PLIB_ADC_StopInIdleEnable and PLIB_ADC_StopInIdleDisable determine if the ADC module stops or continues operation in Idle mode. If PLIB_ADC_StopInIdleDisable is used, the module will continue operation in Idle mode. If the ADC interrupt is enabled, the device will wake up from Idle mode on the ADC interrupt. If PLIB_ADC_StopInIdleEnable is used, the module will stop in Idle mode. If the device enters Idle mode in the middle of conversion, the conversion is aborted.

Operation in other Power-Saving modes

If the ADC module is expected to operate in a power-saving mode, configure the acquisition time and the conversion clock using the functions in accordance with the power-saving mode clock that will be used. After the power-saving mode is entered, an analog-to-digital acquisition can be started. Once the acquisition is started, the device can continue to be clocked by the same power-saving source until the conversion has completed. If desired, the device may be placed in the power-saving Idle mode during conversion.

Note: Not all functionality is available on all devices. Refer to the **"Analog-to-Digital Converter (ADC)"** chapter in the specific device data sheet for availability.

Conversion Sequence Examples

This topic provides examples on how the sampling and conversion will occur in various configurations.

Description

Converting Single Channel, Manual Sample Start, Manual Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

int16_t ADCValue;
```

```
// Enabling the sampling manually
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_SAMP_CLEAR);

// Disabling ADC Input channels for Scan
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Connect AN2 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN2);

// Manual Sample and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 2);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
   // Start Sampling
   PLIB_ADC_SamplingStart(MY_ADC_INSTANCE);

   Delay(); // Ensure, correct sampling time has elapsed before starting conversion.

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   ADCValue = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);

}
```

Converting Single Channel, Manual Sample Start, TAD-based Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

int16_t ADCValue;

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);
// Disabling ADC Input channels for Scan
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);
// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
   // Start Sampling
   PLIB_ADC_SamplingStart(MY_ADC_INSTANCE);

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   ADCValue = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);

}
```

Converting Single Channel, Automatic Sample Start, TAD-based Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

int16_t ADCValue;
```

```
uint8_t index;

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);
// Disabling ADC Input channels for Scan
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000,20000000);

// Set the interrupt every 3 samples
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_3SAMPLES_PER_INTERRUPT);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
    ADCValue = 0;

   // Auto Start Sampling and then go to conversion
   PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   // Yes, stop sample/convert
   PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);

   // Average the 2 ADC values
   for(index = 0; index < 2; index++)
   {
      ADCValue = ADCValue + PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, index);
   }

   ADCValue = ADCValue >> 1;
} // Repeat
```

Converting Single Channel, Automatic Sample Start, Conversion Trigger-based Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

int16_t ADCValue;

// General Purpose Timer 3 compare match ends sampling and starts conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_TMR3_COMPARE_MATCH);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);
// Disabling ADC Input channels for Scan
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSelect(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSelect(MY_ADC_INSTANCE, 1);

// Configure the timer to generate period match as per the acquisition requirements
TMR3  = 0x0000; // set TMR3 to time out every 125 ms
PR3   = 0x3FFF;
T3CON = 0x8010;

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
```

```
{
   // Auto Start Sampling and then go to conversion
   PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   // Yes, stop sample/convert
   PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);

   ADCValue += PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);
}
```

Sampling and Converting a Single Channel Multiple Times

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

int16_t ADCValue;
uint8_t index;

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);
// Disabling ADC Input channels for Scan
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Set the interrupt every 16 samples
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_16SAMPLES_PER_INTERRUPT);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
    ADCValue = 0;

   // Auto Start Sampling and then go to conversion
   PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   // Yes, stop sample/convert
   PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);

   // Average the 16 ADC value
   for(index = 0; index < 16; index++)
   {
      ADCValue = ADCValue  + PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, index);
   }

   ADCValue = ADCValue >> 4;
} // Repeat
```

Sampling and Converting all Channels

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

int ADCValue, index;

// Include all channels in scan
PLIB_ADC_InputScanMaskAdd(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);
```

```
// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Set the interrupt every 16 samples
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_16SAMPLES_PER_INTERRUPT);

// Enable scanning of channels
PLIB_ADC_MuxAInputScanEnable(MY_ADC_INSTANCE);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
    ADCValue = 0;

   // Auto Start Sampling and then go to conversion
   PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   // Yes, stop sample/convert
   PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);

   // Average the 16 ADC value
   for(index = 0; index < 16; index++)
   {
      ADCValue = ADCValue + PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, index);
   }

   ADCValue = ADCValue >> 4;
} // Repeat
```

Wait for Sample, Manual Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

uint8_t ADCValue;

// Internal Counter triggers conversion
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);

// Disabling ADC Input channels for Scan
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_ALL);

// Sample Time = 31TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 30);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
   Delay(); // Ensure, correct sampling time has elapsed before starting conversion.

   // Start Sampling
   PLIB_ADC_SamplingStop(MY_ADC_INSTANCE);

   // Is Conversion Done ??
```

```
    while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

    ADCValue = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);

}
```

Wait for Sample, Triggered Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

uint8_t ADCValue;

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);

// Manual Sample and Conversion Clock
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
  Delay(); // Ensure, correct sampling time has elapsed

  PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INT0_TRANSITION);

  // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

  ADCValue = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 0);
}
```

Manual Sample Start, TAD-based Conversion Start

```
// Where MY_ADC_INSTANCE, is the instance of ADC that the application is using. It is one of the possible
values
// from ADC_MODULE_ID

uint8_t ADCValue;

// Connect AN12 as Positive Input
PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1, ADC_MUX_A, ADC_INPUT_POSITIVE_AN12);

// Sample Time = 1TAD and TAD = 2TCY
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 0);
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);

// Enable the ADC
PLIB_ADC_Enable(MY_ADC_INSTANCE);

while(1)
{
   // Start Sampling
   PLIB_ADC_SamplingStart(MY_ADC_INSTANCE);

   // Is Conversion Done ??
   while(!PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE));

   ADCValue = PLIB_ADC_ResultGet(MY_ADC_INSTANCE);
}
```

Configuring the Library

The library is configured for the supported ADC modules with the multi-buffer interface when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration

	Name	Description
	PLIB_ADC_Enable	ADC module is enabled (turned ON).
	PLIB_ADC_Disable	ADC module is disabled (turned OFF).
	PLIB_ADC_StopInIdleDisable	Continue ADC module operation when the device is in Idle mode.
	PLIB_ADC_StopInIdleEnable	Discontinue ADC module operation when device enters Idle mode.
	PLIB_ADC_VoltageReferenceSelect	Voltage reference configuration.
	PLIB_ADC_CalibrationEnable	Calibration is performed on the next ADC conversion.
	PLIB_ADC_CalibrationDisable	Normal ADC module operation (no calibration is performed).
	PLIB_ADC_InputScanMaskAdd	Select ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskRemove	Omits ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskAddExtended	Select extended ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskRemoveExtended	Omits extended ADC analog channel for input scan.

c) Conversion Control Logic

	Name	Description
	PLIB_ADC_ConversionStart	Starts ADC module manual conversion process.
	PLIB_ADC_ConversionHasCompleted	Provides the conversion completion status of the ADC.
	PLIB_ADC_ConversionTriggerSourceSelect	Selects the conversion trigger source.
	PLIB_ADC_ConversionStopSequenceEnable	Stop conversion sequence (when the first ADC module interrupt is generated).
	PLIB_ADC_ConversionStopSequenceDisable	Normal conversion sequence.
	PLIB_ADC_ConversionClockSet	Sets the ADC module conversion clock.
	PLIB_ADC_ConversionClockGet	Obtains the conversion clock.
	PLIB_ADC_ConversionClockSourceSelect	Selects the ADC module conversion clock source.

d) Sample and Hold Control Logic

	Name	Description
	PLIB_ADC_SampleAutoStartDisable	Sampling auto-start is disabled.
	PLIB_ADC_SampleAutoStartEnable	Sampling auto-start is enabled.
	PLIB_ADC_SamplesPerInterruptSelect	Interrupts at the completion of conversion for each nth sample.
	PLIB_ADC_SamplingIsActive	Provides the ADC sampling status.
	PLIB_ADC_SamplingModeSelect	Enable the selected sampling mode.
	PLIB_ADC_SamplingStart	Sampling is enabled.
	PLIB_ADC_SamplingStop	Holding is enabled.
	PLIB_ADC_SampleAcquisitionTimeSet	Sets the ADC acquisition/auto-sample time in TADs.

f) Output Configuration

	Name	Description
	PLIB_ADC_ResultBufferModeSelect	Selects the result buffer mode.
	PLIB_ADC_ResultBufferStatusGet	Provides the buffer fill status.
	PLIB_ADC_ResultFormatSelect	Selects the result format.
	PLIB_ADC_ResultGetByIndex	Provides the ADC conversion result based on the buffer index.

g) MUX Selection and Channel Scan

	Name	Description
	PLIB_ADC_MuxAInputScanDisable	Do not scan input selections for CH0+ of MUX A.
	PLIB_ADC_MuxAInputScanEnable	Scan input selections for CH0+ of MUX A.
	PLIB_ADC_MuxChannel0InputNegativeSelect	Channel 0 negative input select for multiplexer setting.
	PLIB_ADC_MuxChannel0InputPositiveSelect	Channel 0 positive input select for multiplexer setting.

h) Feature Existence Functions

	Name	Description
	PLIB_ADC_ExistsCalibrationControl	Identifies whether the CalibrationControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionClock	Identifies whether the ConversionClock feature exists on the ADC module
	PLIB_ADC_ExistsConversionClockSource	Identifies whether the ConversionClockSource feature exists on the ADC module
	PLIB_ADC_ExistsConversionControl	Identifies whether the ConversionControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionStatus	Identifies whether the ConversionStatus feature exists on the ADC module
	PLIB_ADC_ExistsConversionStopSequenceControl	Identifies whether the ConversionStopSequenceControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionTriggerSource	Identifies whether the ConversionTriggerSource feature exists on the ADC module
	PLIB_ADC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADC module
	PLIB_ADC_ExistsMuxChannel0NegativeInput	Identifies whether the MuxChannel0NegativeInput feature exists on the ADC module
	PLIB_ADC_ExistsMuxChannel0PositiveInput	Identifies whether the MuxChannel0PositiveInput feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanControl	Identifies whether the MuxInputScanControl feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanSelect	Identifies whether the MuxInputScanSelect feature exists on the ADC module
	PLIB_ADC_ExistsResultBufferFillStatus	Identifies whether the ResultBufferFillStatus feature exists on the ADC module
	PLIB_ADC_ExistsResultBufferMode	Identifies whether the ResultBufferMode feature exists on the ADC module
	PLIB_ADC_ExistsResultFormat	Identifies whether the ResultFormat feature exists on the ADC module
	PLIB_ADC_ExistsResultGetByIndex	Identifies whether the ResultGetByIndex feature exists on the ADC module
	PLIB_ADC_ExistsSamplesPerInterruptSelect	Identifies whether the SamplesPerInterruptSelect feature exists on the ADC module
	PLIB_ADC_ExistsSamplingAcquisitionTime	Identifies whether the SamplingAcquisitionTime feature exists on the ADC module
	PLIB_ADC_ExistsSamplingAutoStart	Identifies whether the SamplingAutoStart feature exists on the ADC module
	PLIB_ADC_ExistsSamplingControl	Identifies whether the SamplingControl feature exists on the ADC module
	PLIB_ADC_ExistsSamplingModeControl	Identifies whether the SamplingModeControl feature exists on the ADC module
	PLIB_ADC_ExistsSamplingStatus	Identifies whether the SamplingStatus feature exists on the ADC module
	PLIB_ADC_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the ADC module
	PLIB_ADC_ExistsVoltageReference	Identifies whether the VoltageReference feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanSelectExtended	Identifies whether the MuxInputScanSelectExtended feature exists on the ADC module

i) Data Types & Constants

	Name	Description
	ADC_MODULE_ID	Identifies the available ADC modules.
	ADC_VOLTAGE_REFERENCE	Defining the different ADC Voltage Reference by which the ADC can be configured.
	ADC_INPUTS_NEGATIVE	Defines the different ADC Negative Input Enumeration.
	ADC_SAMPLES_PER_INTERRUPT	Defining the Samples Per Interrupt Enumeration.
	ADC_CLOCK_SOURCE	Defines the ADC Clock Source Select.
	ADC_CONVERSION_TRIGGER_SOURCE	Defines the ADC Conversion Trigger Source.
	ADC_BUFFER_MODE	Defines the ADC Buffer Mode.
	ADC_RESULT_BUF_STATUS	Defines the ADC Result Buffer Status
	ADC_MUX	Defining the different ADC MUX Enumeration.
	ADC_SAMPLING_MODE	Defines the ADC Sampling Mode Select.
	ADC_INPUTS_SCAN	Defines the ADC Scan inputs.
	ADC_RESULT_FORMAT	Defines the ADC Result Format.
	ADC_INPUTS_POSITIVE	Defines the ADC inputs.
	ADC_ACQUISITION_TIME	Data type defining the different ADC acquisition times by which the ADC can be configured.
	ADC_CONVERSION_CLOCK	Data type defines the different ADC Conversion clock
	ADC_SAMPLE	Data type defining the size of the ADC sample register.

Description

This section describes the Application Programming Interface (API) functions of the Pipelined Analog-to-Digital Converter (ADC) Peripheral Library. Refer to each section for a detailed description.

a) General Configuration

PLIB_ADC_Enable Function

ADC module is enabled (turned ON).

File

plib_adc.h

C

```
void PLIB_ADC_Enable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the selected ADC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_Enable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_Enable(ADC_MODULE_ID index)

PLIB_ADC_Disable Function

ADC module is disabled (turned OFF).

File

plib_adc.h

C

```
void PLIB_ADC_Disable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the selected ADC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_Disable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_Disable(ADC_MODULE_ID index)

PLIB_ADC_StopInIdleDisable Function

Continue ADC module operation when the device is in Idle mode.

File

plib_adc.h

C

```
void PLIB_ADC_StopInIdleDisable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables the ADC module to continue operation when the device is in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_StopInIdleDisable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_StopInIdleDisable(ADC_MODULE_ID index)

PLIB_ADC_StopInIdleEnable Function

Discontinue ADC module operation when device enters Idle mode.

File

plib_adc.h

C

```
void PLIB_ADC_StopInIdleEnable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function discontinues ADC module operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_StopInIdleEnable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_StopInIdleEnable(ADC_MODULE_ID index)

PLIB_ADC_VoltageReferenceSelect Function

Voltage reference configuration.

File

plib_adc.h

C

```
void PLIB_ADC_VoltageReferenceSelect(ADC_MODULE_ID index, ADC_VOLTAGE_REFERENCE configValue);
```

Returns

None.

Description

This function configures the ADC module voltage reference.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsVoltageReference in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_VoltageReferenceSelect(MY_ADC_INSTANCE, ADC_REFERENCE_VREFPLUS_TO_AVSS);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
configValue	One of the possible values from ADC_VOLTAGE_REFERENCE

Function

void PLIB_ADC_VoltageReferenceSelect(ADC_MODULE_ID index,
ADC_VOLTAGE_REFERENCE configValue)

PLIB_ADC_CalibrationEnable Function

Calibration is performed on the next ADC conversion.

File

plib_adc.h

C

```
void PLIB_ADC_CalibrationEnable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables calibration to be performed on the next ADC conversion.

When the Calibration bit is enabled, inputs are disconnected and tied to AVss. This sets the inputs of the ADC to zero. Then, the user can perform a conversion. Use of the Calibration mode is not affected if the ADC line has been configured as analog or digital, nor by channel input selection. Any analog input switches are disconnected from the ADC module in this mode. The conversion result is stored by the user software and is used to compensate subsequent conversions. This can be done by adding the two's complement of the result obtained during calibration to all normal ADC conversions.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsCalibrationControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_CalibrationEnable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_CalibrationEnable(ADC_MODULE_ID index)

PLIB_ADC_CalibrationDisable Function

Normal ADC module operation (no calibration is performed).

File

plib_adc.h

C

```
void PLIB_ADC_CalibrationDisable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables normal ADC module operation without any calibration.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsCalibrationControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_CalibrationDisable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_CalibrationDisable(ADC_MODULE_ID index)

PLIB_ADC_InputScanMaskAdd Function

Select ADC analog channel for input scan.

File

plib_adc.h

C

```
void PLIB_ADC_InputScanMaskAdd(ADC_MODULE_ID index, ADC_INPUTS_SCAN scanInputs);
```

Returns

None.

Description

This function selects the ADC analog channel for input scanning.

Remarks

Multiple channels can be added simultaneously by ORing.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxInputScanSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
// Single channel addition
PLIB_ADC_InputScanMaskAdd(MY_ADC_INSTANCE, ADC_INPUT_SCAN_AN2);
// Multiple channels addition
PLIB_ADC_InputScanMaskAdd(ADC_ID_1, ADC_INPUT_SCAN_AN2 | ADC_INPUT_SCAN_AN2);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
scanInputs	One of the possible values from ADC_INPUTS_SCAN. Inputs are added for scanning.

Function

void PLIB_ADC_InputScanMaskAdd(ADC_MODULE_ID index,
ADC_INPUTS_SCAN scanInputs)

PLIB_ADC_InputScanMaskRemove Function

Omits ADC analog channel for input scan.

File

plib_adc.h

C

```
void PLIB_ADC_InputScanMaskRemove(ADC_MODULE_ID index, ADC_INPUTS_SCAN scanInputs);
```

Returns

None.

Description

This function allows the ADC analog channel to be omitted from input scanning.

Remarks

Multiple channels can be removed simultaneously by ORing.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxInputScanSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
// Single channel removing
PLIB_ADC_InputScanMaskRemove(MY_ADC_INSTANCE, ADC_INPUT_SCAN_AN2);
// Multiple channels removing
PLIB_ADC_InputScanMaskRemove(ADC_ID_1, ADC_INPUT_SCAN_AN2 | ADC_INPUT_SCAN_AN3);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
scanInputs	One of the possible values from ADC_INPUTS_SCAN. Inputs are removed from scanning.

Function

void PLIB_ADC_InputScanMaskRemove(ADC_MODULE_ID index,
ADC_INPUTS_SCAN scanInputs)

PLIB_ADC_InputScanMaskAddExtended Function

Select extended ADC analog channel for input scan.

File

plib_adc.h

C

```
void PLIB_ADC_InputScanMaskAddExtended(ADC_MODULE_ID index, ADC_INPUTS_SCAN_EXTENDED scanInputs);
```

Returns

None.

Description

This function selects the extended ADC analog channel for input scanning.

Remarks

Multiple channels can be added simultaneously by ORing.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxInputScanSelectExtended in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Single channel addition
PLIB_ADC_InputScanMaskAddExtended(ADC_ID_1, ADC_INPUT_SCAN_AN36);
// Multiple channels addition
PLIB_ADC_InputScanMaskAddExtended(ADC_ID_1, ADC_INPUT_SCAN_AN36 | ADC_INPUT_SCAN_AN39);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
scanInputs	One of the possible values from ADC_INPUTS_SCAN_EXTENDED. Inputs are added for scanning.

Function

void PLIB_ADC_InputScanMaskAddExtended(ADC_MODULE_ID index,
ADC_INPUTS_SCAN_EXTENDED scanInputs)

PLIB_ADC_InputScanMaskRemoveExtended Function

Omits extended ADC analog channel for input scan.

File

plib_adc.h

C

```
void PLIB_ADC_InputScanMaskRemoveExtended(ADC_MODULE_ID index, ADC_INPUTS_SCAN_EXTENDED scanInputs);
```

Returns

None.

Description

This function allows the extended ADC analog channel to be omitted from input scanning.

Remarks

Multiple channels can be removed simultaneously by ORing.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxInputScanSelectExtended in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Single channel removing
PLIB_ADC_InputScanMaskRemove(ADC_ID_1, ADC_INPUT_SCAN_AN36);
// Multiple channels removing
PLIB_ADC_InputScanMaskRemove(ADC_ID_1, ADC_INPUT_SCAN_AN36 | ADC_INPUT_SCAN_AN39);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
scanInputs	One of the possible values from ADC_INPUTS_SCAN_EXTENDED. Inputs are removed from scanning.

Function

void PLIB_ADC_InputScanMaskRemove(ADC_MODULE_ID index,
ADC_INPUTS_SCAN_EXTENDED scanInputs)

b) DMA Transactions

c) Conversion Control Logic

PLIB_ADC_ConversionStart Function

Starts ADC module manual conversion process.

File

plib_adc.h

C

```
void PLIB_ADC_ConversionStart(ADC_MODULE_ID index);
```

Returns

None.

Description

This function starts the ADC module manual conversion process.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionControl in your application to determine whether this feature is available.

Preconditions

Automatic sampling must be disabled.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ConversionStart(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_ConversionStart(ADC_MODULE_ID index)

PLIB_ADC_ConversionHasCompleted Function

Provides the conversion completion status of the ADC.

File

plib_adc.h

C

```
bool PLIB_ADC_ConversionHasCompleted(ADC_MODULE_ID index);
```

Returns

Boolean:

- true - ADC conversion is done/completed
- false - ADC conversion is in progress or has not started

Description

This function provides the completion status of analog-to-digital conversion.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
bool my_status = PLIB_ADC_ConversionHasCompleted(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_ADC_ConversionHasCompleted(ADC_MODULE_ID index)

PLIB_ADC_ConversionTriggerSourceSelect Function

Selects the conversion trigger source.

File

plib_adc.h

C

```
void PLIB_ADC_ConversionTriggerSourceSelect(ADC_MODULE_ID index, ADC_CONVERSION_TRIGGER_SOURCE source);
```

Returns

None.

Description

This function selects the ADC module conversion trigger source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionTriggerSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ConversionTriggerSourceSelect(MY_ADC_INSTANCE, ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	One of the possible values from ADC_CONVERSION_TRIGGER_SOURCE

Function

void PLIB_ADC_ConversionTriggerSourceSelect(ADC_MODULE_ID index,
ADC_CONVERSION_TRIGGER_SOURCE source)

PLIB_ADC_ConversionStopSequenceEnable Function

Stop conversion sequence (when the first ADC module interrupt is generated).

File

plib_adc.h

C

```
void PLIB_ADC_ConversionStopSequenceEnable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function stops conversions when the first ADC module interrupt is generated. Hardware clears the Automatic Sampling bit when the ADC interrupt is generated.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionStopSequenceControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ConversionStopSequenceEnable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_ConversionStopSequenceEnable(ADC_MODULE_ID index)

PLIB_ADC_ConversionStopSequenceDisable Function

Normal conversion sequence.

File

plib_adc.h

C

```
void PLIB_ADC_ConversionStopSequenceDisable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables normal operation, wherein the buffer contents will be overwritten by the next conversion sequence.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionStopSequenceControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ConversionStopSequenceDisable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_ConversionStopSequenceDisable(ADC_MODULE_ID index)

PLIB_ADC_ConversionClockSet Function

Sets the ADC module conversion clock.

File

plib_adc.h

C

```
void PLIB_ADC_ConversionClockSet(ADC_MODULE_ID index, uint32_t baseFrequency, ADC_CONVERSION_CLOCK value);
```

Returns

None.

Description

This function sets the ADC module conversion clock prescaler.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionClock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
// Following functions passed the input clock to ADC as 80MHz and
// required conversion clock as 20MHz.
PLIB_ADC_ConversionClockSet(MY_ADC_INSTANCE, 80000000, 20000000);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baseFrequency	Input clock frequency in Hertz (Hz). This is the input clock to ADC module.
value	Unsigned value of type ADC_CONVERSION_CLOCK. This is the required conversion clock of ADC module in Hz.

Function

void PLIB_ADC_ConversionClockSet(ADC_MODULE_ID index,

uint32_t baseFrequency,

ADC_CONVERSION_CLOCK value)

PLIB_ADC_ConversionClockGet Function

Obtains the conversion clock.

File

plib_adc.h

C

```
ADC_CONVERSION_CLOCK PLIB_ADC_ConversionClockGet(ADC_MODULE_ID index, uint32_t baseFrequency);
```

Returns

ADC_CONVERSION_CLOCK - ADC Conversion clock value (in Hz) of type ADC_CONVERSION_CLOCK

Description

This function obtains the conversion clock that is being used by the ADC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionClock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
// baseFrequency is the peripheral input frequency
ADC_CONVERSION_CLOCK conversionClock; // To store the conversion clock value
conversionClock = PLIB_ADC_ConversionClockGet(MY_ADC_INSTANCE, 8000000);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baseFrequencyHz	Input clock frequency to the ADC module in Hz

Function

ADC_CONVERSION_CLOCK PLIB_ADC_ConversionClockGet(ADC_MODULE_ID index,
uint32_t baseFrequencyHz)

PLIB_ADC_ConversionClockSourceSelect Function

Selects the ADC module conversion clock source.

File

plib_adc.h

C

```
void PLIB_ADC_ConversionClockSourceSelect(ADC_MODULE_ID index, ADC_CLOCK_SOURCE source);
```

Returns

None.

Description

This function selects the ADC module conversion clock source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsConversionClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ConversionClockSourceSelect(MY_ADC_INSTANCE, ADC_CLOCK_SOURCE_PERIPHERAL_BUS_CLOCK);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	One of the possible values from ADC_CLOCK_SOURCE

Function

void PLIB_ADC_ConversionClockSourceSelect(ADC_MODULE_ID index,
ADC_CLOCK_SOURCE source)

d) Sample and Hold Control Logic

PLIB_ADC_SampleAutoStartDisable Function

Sampling auto-start is disabled.

File

plib_adc.h

C

```
void PLIB_ADC_SampleAutoStartDisable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function disables auto-sampling and enables manual sampling.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingAutoStart in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SampleAutoStartDisable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_SampleAutoStartDisable(ADC_MODULE_ID index)

PLIB_ADC_SampleAutoStartEnable Function

Sampling auto-start is enabled.

File

plib_adc.h

C

```
void PLIB_ADC_SampleAutoStartEnable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables auto-sampling. Sampling begins immediately after the last conversion is completed.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingAutoStart in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SampleAutoStartEnable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_SampleAutoStartEnable(ADC_MODULE_ID index)

PLIB_ADC_SamplesPerInterruptSelect Function

Interrupts at the completion of conversion for each nth sample.

File

plib_adc.h

C

```
void PLIB_ADC_SamplesPerInterruptSelect(ADC_MODULE_ID index, ADC_SAMPLES_PER_INTERRUPT value);
```

Returns

None.

Description

This function interrupts at the completion of conversion for each nth sample/convert sequence.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplesPerInterruptSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SamplesPerInterruptSelect(MY_ADC_INSTANCE, ADC_16SAMPLES_PER_INTERRUPT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	Possible values from ADC_SAMPLES_PER_INTERRUPT

Function

void PLIB_ADC_SamplesPerInterruptSelect(ADC_MODULE_ID index,
ADC_SAMPLES_PER_INTERRUPT value)

PLIB_ADC_SamplingIsActive Function

Provides the ADC sampling status.

File

plib_adc.h

C

```
bool PLIB_ADC_SamplingIsActive(ADC_MODULE_ID index);
```

Returns

Boolean:

- true - ADC Sample and Hold circuit is sampling
- false - ADC Sample and Hold circuit is holding

Description

This function returns the ADC sampling status on whether the ADC is sampling or holding.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
bool my_status = PLIB_ADC_SamplingIsActive(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_ADC_SamplingIsActive(ADC_MODULE_ID index)

PLIB_ADC_SamplingModeSelect Function

Enable the selected sampling mode.

File

plib_adc.h

C

```
void PLIB_ADC_SamplingModeSelect(ADC_MODULE_ID index, ADC_SAMPLING_MODE mode);
```

Returns

None.

Description

This function selects the sampling mode.

Remarks

Sampling mode could be alternate input or Simultaneous or Sequential mode. Alternate input can be combined with Simultaneous or Sequential modes.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingModeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SamplingModeSelect(MY_ADC_INSTANCE, ADC_SAMPLING_MODE_ALTERNATE_INPUT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the possible values from ADC_SAMPLING_MODE

Function

void PLIB_ADC_SamplingModeSelect(ADC_MODULE_ID index, ADC_SAMPLING_MODE mode)

PLIB_ADC_SamplingStart Function

Sampling is enabled.

File

plib_adc.h

C

```
void PLIB_ADC_SamplingStart(ADC_MODULE_ID index);
```

Returns

None.

Description

This function starts the ADC Sample and Hold circuit to sample the input channel.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingControl in your application to determine whether this feature is available.

Preconditions

Automatic sampling must be disabled.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SamplingStart(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_SamplingStart(ADC_MODULE_ID index)

PLIB_ADC_SamplingStop Function

Holding is enabled.

File

plib_adc.h

C

```
void PLIB_ADC_SamplingStop(ADC_MODULE_ID index);
```

Returns

None.

Description

This function stops the ADC Sample and Hold circuit from sampling and holds the sampled data.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingControl in your application to determine whether this feature is available.

Preconditions

Automatic sampling must be disabled.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
```

```
PLIB_ADC_SamplingStop(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_SamplingStop(ADC_MODULE_ID index)

PLIB_ADC_SampleAcquisitionTimeSet Function

Sets the ADC acquisition/auto-sample time in TADs.

File

plib_adc.h

C

```
void PLIB_ADC_SampleAcquisitionTimeSet(ADC_MODULE_ID index, ADC_ACQUISITION_TIME acqTime);
```

Returns

None.

Description

This function sets the ADC acquisition/auto-sample time in TADs.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsSamplingAcquisitionTime in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_SampleAcquisitionTimeSet(MY_ADC_INSTANCE, 2);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
acqTime	Unsigned value of type ADC_ACQUISITION_TIME

Function

void PLIB_ADC_SampleAcquisitionTimeSet(ADC_MODULE_ID index,
ADC_ACQUISITION_TIME acqTime)

e) Channel Pairs Control

f) Output Configuration

PLIB_ADC_ResultBufferModeSelect Function

Selects the result buffer mode.

File

plib_adc.h

C

```
void PLIB_ADC_ResultBufferModeSelect(ADC_MODULE_ID index, ADC_BUFFER_MODE mode);
```

Returns

None.

Description

This function selects the result buffer mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsResultBufferMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ResultBufferModeSelect(MY_ADC_INSTANCE,
                                ADC_BUFFER_MODE_TWO_8WORD_BUFFERS);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the possible values from ADC_BUFFER_MODE

Function

void PLIB_ADC_ResultBufferModeSelect(ADC_MODULE_ID index,
ADC_BUFFER_MODE mode)

PLIB_ADC_ResultBufferStatusGet Function

Provides the buffer fill status.

File

plib_adc.h

C

```
ADC_RESULT_BUF_STATUS PLIB_ADC_ResultBufferStatusGet(ADC_MODULE_ID index);
```

Returns

Boolean:

- true = ADC is currently filling buffer 08-0F, user should access data in 00-07
- false = ADC is currently filling buffer 00-07, user should access data in 08-0F

Description

This function obtains the buffer fill status.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsResultBufferFillStatus in your application to determine whether this feature is available.

Preconditions

ADC multi-buffer support is available and configured.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
ADC_RESULT_BUF_STATUS my_status;

my_status = PLIB_ADC_ResultBufferStatusGet(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

ADC_RESULT_BUF_STATUS PLIB_ADC_ResultBufferStatusGet(ADC_MODULE_ID index)

PLIB_ADC_ResultFormatSelect Function

Selects the result format.

File

plib_adc.h

C

```
void PLIB_ADC_ResultFormatSelect(ADC_MODULE_ID index, ADC_RESULT_FORMAT resultFormat);
```

Returns

None.

Description

This function selects the result format.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsResultFormat in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_ResultFormatSelect(MY_ADC_INSTANCE, ADC_RESULT_FORMAT_INTEGER_16BIT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
resultFormat	One of the possible values from ADC_RESULT_FORMAT

Function

void PLIB_ADC_ResultFormatSelect(ADC_MODULE_ID index,
ADC_RESULT_FORMAT resultFormat)

PLIB_ADC_ResultGetByIndex Function

Provides the ADC conversion result based on the buffer index.

File

plib_adc.h

C

```
ADC_SAMPLE PLIB_ADC_ResultGetByIndex(ADC_MODULE_ID index, uint8_t bufferIndex);
```

Returns

int16_t - ADC Conversion result at the respective bufferIndex

Description

This function provides the ADC module conversion result based on the buffer index.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsResultGetByIndex in your application to determine whether this feature is available.

Preconditions

ADC multi-buffer support available and configured.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
ADC_SAMPLE my_res = PLIB_ADC_ResultGetByIndex(MY_ADC_INSTANCE, 15);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bufferIndex	Value ranging from 0 to 15

Function

ADC_SAMPLE PLIB_ADC_ResultGetByIndex(ADC_MODULE_ID index,
uint8_t bufferIndex)

g) MUX Selection and Channel Scan

PLIB_ADC_MuxAInputScanDisable Function

Do not scan input selections for CH0+ of MUX A.

File

plib_adc.h

C

```
void PLIB_ADC_MuxAInputScanDisable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function disables scan input for CH0+ of MUX A.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxInputScanControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_MuxAInputScanDisable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_MuxAInputScanDisable(ADC_MODULE_ID index)

PLIB_ADC_MuxAInputScanEnable Function

Scan input selections for CH0+ of MUX A.

File

plib_adc.h

C

```
void PLIB_ADC_MuxAInputScanEnable(ADC_MODULE_ID index);
```

Returns

None.

Description

This function enables scan input for CH0+ of MUX A.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxInputScanControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_MuxAInputScanEnable(MY_ADC_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_ADC_MuxAInputScanEnable(ADC_MODULE_ID index)

PLIB_ADC_MuxChannel0InputNegativeSelect Function

Channel 0 negative input select for multiplexer setting.

File

plib_adc.h

C

```
void PLIB_ADC_MuxChannel0InputNegativeSelect(ADC_MODULE_ID index, ADC_MUX muxType, ADC_INPUTS_NEGATIVE
input);
```

Returns

None.

Description

This function selects the negative input for channel 0 of MUX A or MUX B.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxChannel0NegativeInput in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
```

```
PLIB_ADC_MuxChannel0InputNegativeSelect(MY_ADC_INSTANCE, ADC_MUX_A, ADC_INPUT_NEGATIVE_VREF_MINUS);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
muxType	One of the possible values from ADC_MUX

Function

void PLIB_ADC_MuxChannel0InputNegativeSelect(ADC_MODULE_ID index,
ADC_MUX muxType,
ADC_INPUTS_NEGATIVE input)

PLIB_ADC_MuxChannel0InputPositiveSelect Function

Channel 0 positive input select for multiplexer setting.

File

plib_adc.h

C

```
void PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_MODULE_ID index, ADC_MUX muxType, ADC_INPUTS_POSITIVE
input);
```

Returns

None.

Description

This function selects the positive input for channel 0 of MUX A or MUX B.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_ADC_ExistsMuxChannel0PositiveInput in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_ADC_INSTANCE, is the ADC instance selected for use by the
// application developer.
PLIB_ADC_MuxChannel0InputPositiveSelect(MY_ADC_INSTANCE, ADC_MUX_A, ADC_INPUT_POSITIVE_AN2);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
muxType	One of the possible values from ADC_MUX

Function

void PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_MODULE_ID index,
ADC_MUX muxType,
ADC_INPUTS_POSITIVE input)

h) Feature Existence Functions

PLIB_ADC_ExistsCalibrationControl Function

Identifies whether the CalibrationControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsCalibrationControl(ADC_MODULE_ID index);
```

Returns

- true - The CalibrationControl feature is supported on the device
- false - The CalibrationControl feature is not supported on the device

Description

This function identifies whether the CalibrationControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_CalibrationEnable
- PLIB_ADC_CalibrationDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsCalibrationControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsConversionClock Function

Identifies whether the ConversionClock feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsConversionClock(ADC_MODULE_ID index);
```

Returns

- true - The ConversionClock feature is supported on the device
- false - The ConversionClock feature is not supported on the device

Description

This function identifies whether the ConversionClock feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_ConversionClockSet
- PLIB_ADC_ConversionClockGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsConversionClock(ADC_MODULE_ID index)

PLIB_ADC_ExistsConversionClockSource Function

Identifies whether the ConversionClockSource feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsConversionClockSource(ADC_MODULE_ID index);
```

Returns

- true - The ConversionClockSource feature is supported on the device
- false - The ConversionClockSource feature is not supported on the device

Description

This function identifies whether the ConversionClockSource feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ConversionClockSourceSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsConversionClockSource(ADC_MODULE_ID index)

PLIB_ADC_ExistsConversionControl Function

Identifies whether the ConversionControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsConversionControl(ADC_MODULE_ID index);
```

Returns

- true - The ConversionControl feature is supported on the device
- false - The ConversionControl feature is not supported on the device

Description

This function identifies whether the ConversionControl feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ConversionStart

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsConversionControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsConversionStatus Function

Identifies whether the ConversionStatus feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsConversionStatus(ADC_MODULE_ID index);
```

Returns

- true - The ConversionStatus feature is supported on the device
- false - The ConversionStatus feature is not supported on the device

Description

This function identifies whether the ConversionStatus feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ConversionHasCompleted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsConversionStatus(ADC_MODULE_ID index)

PLIB_ADC_ExistsConversionStopSequenceControl Function

Identifies whether the ConversionStopSequenceControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsConversionStopSequenceControl(ADC_MODULE_ID index);
```

Returns

- true - The ConversionStopSequenceControl feature is supported on the device
- false - The ConversionStopSequenceControl feature is not supported on the device

Description

This function identifies whether the ConversionStopSequenceControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_ConversionStopSequenceEnable
- PLIB_ADC_ConversionStopSequenceDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsConversionStopSequenceControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsConversionTriggerSource Function

Identifies whether the ConversionTriggerSource feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsConversionTriggerSource(ADC_MODULE_ID index);
```

Returns

- true - The ConversionTriggerSource feature is supported on the device
- false - The ConversionTriggerSource feature is not supported on the device

Description

This function identifies whether the ConversionTriggerSource feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ConversionTriggerSourceSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsConversionTriggerSource(ADC_MODULE_ID index)

PLIB_ADC_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsEnableControl(ADC_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_Enable
- PLIB_ADC_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsEnableControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsMuxChannel0NegativeInput Function

Identifies whether the MuxChannel0NegativeInput feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsMuxChannel0NegativeInput(ADC_MODULE_ID index);
```

Returns

- true - The MuxChannel0NegativeInput feature is supported on the device
- false - The MuxChannel0NegativeInput feature is not supported on the device

Description

This function identifies whether the MuxChannel0NegativeInput feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_MuxChannel0InputNegativeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsMuxChannel0NegativeInput(ADC_MODULE_ID index)

PLIB_ADC_ExistsMuxChannel0PositiveInput Function

Identifies whether the MuxChannel0PositiveInput feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsMuxChannel0PositiveInput(ADC_MODULE_ID index);
```

Returns

- true - The MuxChannel0PositiveInput feature is supported on the device
- false - The MuxChannel0PositiveInput feature is not supported on the device

Description

This function identifies whether the MuxChannel0PositiveInput feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_MuxChannel0InputPositiveSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsMuxChannel0PositiveInput(ADC_MODULE_ID index)

PLIB_ADC_ExistsMuxInputScanControl Function

Identifies whether the MuxInputScanControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsMuxInputScanControl(ADC_MODULE_ID index);
```

Returns

- true - The MuxInputScanControl feature is supported on the device
- false - The MuxInputScanControl feature is not supported on the device

Description

This function identifies whether the MuxInputScanControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_MuxAInputScanEnable
- PLIB_ADC_MuxAInputScanDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsMuxInputScanControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsMuxInputScanSelect Function

Identifies whether the MuxInputScanSelect feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsMuxInputScanSelect(ADC_MODULE_ID index);
```

Returns

- true - The MuxInputScanSelect feature is supported on the device
- false - The MuxInputScanSelect feature is not supported on the device

Description

This function identifies whether the MuxInputScanSelect feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_InputScanMaskAdd
- PLIB_ADC_InputScanMaskRemove

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsMuxInputScanSelect(ADC_MODULE_ID index)

PLIB_ADC_ExistsResultBufferFillStatus Function

Identifies whether the ResultBufferFillStatus feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsResultBufferFillStatus(ADC_MODULE_ID index);
```

Returns

- true - The ResultBufferFillStatus feature is supported on the device
- false - The ResultBufferFillStatus feature is not supported on the device

Description

This function identifies whether the ResultBufferFillStatus feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ResultBufferStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsResultBufferFillStatus(ADC_MODULE_ID index)

PLIB_ADC_ExistsResultBufferMode Function

Identifies whether the ResultBufferMode feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsResultBufferMode(ADC_MODULE_ID index);
```

Returns

- true - The ResultBufferMode feature is supported on the device
- false - The ResultBufferMode feature is not supported on the device

Description

This function identifies whether the ResultBufferMode feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ResultBufferModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsResultBufferMode(ADC_MODULE_ID index)

PLIB_ADC_ExistsResultFormat Function

Identifies whether the ResultFormat feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsResultFormat(ADC_MODULE_ID index);
```

Returns

- true - The ResultFormat feature is supported on the device
- false - The ResultFormat feature is not supported on the device

Description

This function identifies whether the ResultFormat feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ResultFormatSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsResultFormat(ADC_MODULE_ID index)

PLIB_ADC_ExistsResultGetByIndex Function

Identifies whether the ResultGetByIndex feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsResultGetByIndex(ADC_MODULE_ID index);
```

Returns

- true - The ResultGetByIndex feature is supported on the device
- false - The ResultGetByIndex feature is not supported on the device

Description

This function identifies whether the ResultGetByIndex feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_ResultGetByIndex

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsResultGetByIndex(ADC_MODULE_ID index)

PLIB_ADC_ExistsSamplesPerInterruptSelect Function

Identifies whether the SamplesPerInterruptSelect feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsSamplesPerInterruptSelect(ADC_MODULE_ID index);
```

Returns

- true - The SamplesPerInterruptSelect feature is supported on the device
- false - The SamplesPerInterruptSelect feature is not supported on the device

Description

This function identifies whether the SamplesPerInterruptSelect feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_SamplesPerInterruptSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsSamplesPerInterruptSelect(ADC_MODULE_ID index)

PLIB_ADC_ExistsSamplingAcquisitionTime Function

Identifies whether the SamplingAcquisitionTime feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsSamplingAcquisitionTime(ADC_MODULE_ID index);
```

Returns

- true - The SamplingAcquisitionTime feature is supported on the device
- false - The SamplingAcquisitionTime feature is not supported on the device

Description

This function identifies whether the SamplingAcquisitionTime feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_SampleAcquisitionTimeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsSamplingAcquisitionTime(ADC_MODULE_ID index)

PLIB_ADC_ExistsSamplingAutoStart Function

Identifies whether the SamplingAutoStart feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsSamplingAutoStart(ADC_MODULE_ID index);
```

Returns

- true - The SamplingAutoStart feature is supported on the device
- false - The SamplingAutoStart feature is not supported on the device

Description

This function identifies whether the SamplingAutoStart feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_SampleAutoStartEnable
- PLIB_ADC_SampleAutoStartDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsSamplingAutoStart(ADC_MODULE_ID index)

PLIB_ADC_ExistsSamplingControl Function

Identifies whether the SamplingControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsSamplingControl(ADC_MODULE_ID index);
```

Returns

- true - The SamplingControl feature is supported on the device
- false - The SamplingControl feature is not supported on the device

Description

This function identifies whether the SamplingControl feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_SamplingStart
- PLIB_ADC_SamplingStop

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsSamplingControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsSamplingModeControl Function

Identifies whether the SamplingModeControl feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsSamplingModeControl(ADC_MODULE_ID index);
```

Returns

- true - The SamplingModeControl feature is supported on the device
- false - The SamplingModeControl feature is not supported on the device

Description

This function identifies whether the SamplingModeControl feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_SamplingModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsSamplingModeControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsSamplingStatus Function

Identifies whether the SamplingStatus feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsSamplingStatus(ADC_MODULE_ID index);
```

Returns

- true - The SamplingStatus feature is supported on the device
- false - The SamplingStatus feature is not supported on the device

Description

This function identifies whether the SamplingStatus feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_SamplingIsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsSamplingStatus(ADC_MODULE_ID index)

PLIB_ADC_ExistsStopInIdleControl Function

Identifies whether the StopInIdle feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsStopInIdleControl(ADC_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_StopInIdleEnable
- PLIB_ADC_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsStopInIdleControl(ADC_MODULE_ID index)

PLIB_ADC_ExistsVoltageReference Function

Identifies whether the VoltageReference feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsVoltageReference(ADC_MODULE_ID index);
```

Returns

- true - The VoltageReference feature is supported on the device
- false - The VoltageReference feature is not supported on the device

Description

This function identifies whether the VoltageReference feature is available on the ADC module. When this function returns true, this function is supported on the device:

- PLIB_ADC_VoltageReferenceSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsVoltageReference(ADC_MODULE_ID index)

PLIB_ADC_ExistsMuxInputScanSelectExtended Function

Identifies whether the MuxInputScanSelectExtended feature exists on the ADC module

File

plib_adc.h

C

```
bool PLIB_ADC_ExistsMuxInputScanSelectExtended(ADC_MODULE_ID index);
```

Returns

- true - The MuxInputScanSelectExtended feature is supported on the device
- false - The MuxInputScanSelectExtended feature is not supported on the device

Description

This function identifies whether the MuxInputScanSelectExtended feature is available on the ADC module. When this function returns true, these functions are supported on the device:

- PLIB_ADC_InputScanMaskAddExtended
- PLIB_ADC_InputScanMaskRemoveExtended

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADC_ExistsMuxInputScanSelectExtended(ADC_MODULE_ID index)

i) Data Types & Constants

ADC_MODULE_ID Enumeration

Identifies the available ADC modules.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_ID_1,
  ADC_ID_2,
  ADC_ID_3,
  ADC_ID_4,
  ADC_ID_5,
  ADC_NUMBER_OF_MODULES
} ADC_MODULE_ID;
```

Members

Members	Description
ADC_ID_1	ADC Module 1 ID
ADC_ID_2	ADC Module 2 ID
ADC_ID_3	ADC Module 3 ID
ADC_ID_4	ADC Module 4 ID
ADC_ID_5	ADC Module 5 ID
ADC_NUMBER_OF_MODULES	Number of available ADC modules.

Description

ADC Module ID

This enumeration identifies the available ADC modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules are available on all devices. Refer to the specific device data sheet to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

ADC_VOLTAGE_REFERENCE Enumeration

Defining the different ADC Voltage Reference by which the ADC can be configured.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_REFERENCE_VDD_TO_AVSS,
  ADC_REFERENCE_VREFPLUS_TO_AVSS,
  ADC_REFERENCE_AVDD_TO_VREF_NEG,
  ADC_REFERENCE_VREFPLUS_TO_VREFNEG
} ADC_VOLTAGE_REFERENCE;
```

Members

Members	Description
ADC_REFERENCE_VDD_TO_AVSS	Positive voltage reference supplied internally by VDD and Negative voltage reference supplied internally through VSS
ADC_REFERENCE_VREFPLUS_TO_AVSS	Positive voltage reference supplied externally through VREF+ pin and Negative voltage reference supplied internally through VSS
ADC_REFERENCE_AVDD_TO_VREF_NEG	Positive voltage reference supplied internally by VDD and Negative voltage reference supplied externally through VREF- pin
ADC_REFERENCE_VREFPLUS_TO_VREFNEG	Positive voltage reference supplied externally through VREF+ pin and Negative voltage reference supplied externally through VREF- pin

Description

ADC Voltage Reference Enumeration

This data type defines the different ADC Voltage Reference by which the ADC can be configured.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_INPUTS_NEGATIVE Enumeration

Defines the different ADC Negative Input Enumeration.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_INPUT_NEGATIVE_VREF_MINUS,
  ADC_INPUT_NEGATIVE_AN1,
  ADC_INPUT_NEGATIVE_AN2,
  ADC_INPUT_NEGATIVE_AN3,
  ADC_INPUT_NEGATIVE_AN4,
  ADC_INPUT_NEGATIVE_AN5,
  ADC_INPUT_NEGATIVE_AN6
} ADC_INPUTS_NEGATIVE;
```

Members

Members	Description
ADC_INPUT_NEGATIVE_VREF_MINUS	Negative input is VREF
ADC_INPUT_NEGATIVE_AN1	Negative input is AN1
ADC_INPUT_NEGATIVE_AN2	Negative input is AN2
ADC_INPUT_NEGATIVE_AN3	Negative input is AN3
ADC_INPUT_NEGATIVE_AN4	Negative input is AN4
ADC_INPUT_NEGATIVE_AN5	Negative input is AN5
ADC_INPUT_NEGATIVE_AN6	Negative input is AN6

Description

ADC Negative Input Enumeration

This data type defines the different ADC Negative Input Enumeration.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_SAMPLES_PER_INTERRUPT Enumeration

Defining the Samples Per Interrupt Enumeration.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_1SAMPLE_PER_INTERRUPT,
  ADC_2SAMPLES_PER_INTERRUPT,
  ADC_3SAMPLES_PER_INTERRUPT,
  ADC_4SAMPLES_PER_INTERRUPT,
  ADC_5SAMPLES_PER_INTERRUPT,
  ADC_6SAMPLES_PER_INTERRUPT,
  ADC_7SAMPLES_PER_INTERRUPT,
  ADC_8SAMPLES_PER_INTERRUPT,
  ADC_9SAMPLES_PER_INTERRUPT,
  ADC_10SAMPLES_PER_INTERRUPT,
  ADC_11SAMPLES_PER_INTERRUPT,
  ADC_12SAMPLES_PER_INTERRUPT,
  ADC_13SAMPLES_PER_INTERRUPT,
  ADC_14SAMPLES_PER_INTERRUPT,
  ADC_15SAMPLES_PER_INTERRUPT,
  ADC_16SAMPLES_PER_INTERRUPT
} ADC_SAMPLES_PER_INTERRUPT;
```

Members

Members	Description
ADC_1SAMPLE_PER_INTERRUPT	Interrupts at the completion of conversion for each sample/convert sequence
ADC_2SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 2nd sample/convert sequence
ADC_3SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 3rd sample/convert sequence
ADC_4SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 4th sample/convert sequence
ADC_5SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 5th sample/convert sequence
ADC_6SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 6th sample/convert sequence
ADC_7SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 7th sample/convert sequence
ADC_8SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 8th sample/convert sequence
ADC_9SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 9th sample/convert sequence
ADC_10SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 10th sample/convert sequence
ADC_11SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 11th sample/convert sequence
ADC_12SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 12th sample/convert sequence
ADC_13SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 13th sample/convert sequence
ADC_14SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 14th sample/convert sequence
ADC_15SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 15th sample/convert sequence
ADC_16SAMPLES_PER_INTERRUPT	Interrupts at the completion of conversion for each 16th sample/convert sequence

Description

ADC samples per Interrupt Enumeration

This data type defines the Samples Per Interrupt Enumeration.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_CLOCK_SOURCE Enumeration

Defines the ADC Clock Source Select.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_CLOCK_SOURCE_PERIPHERAL_BUS_CLOCK,
  ADC_CLOCK_SOURCE_SYSTEM_CLOCK,
  ADC_CLOCK_SOURCE_INTERNAL_RC
} ADC_CLOCK_SOURCE;
```

Members

Members	Description
ADC_CLOCK_SOURCE_PERIPHERAL_BUS_CLOCK	A/D Peripheral clock
ADC_CLOCK_SOURCE_SYSTEM_CLOCK	This option is deprecated, use ADC_CLOCK_SOURCE_PERIPHERAL_BUS_CLOCK
ADC_CLOCK_SOURCE_INTERNAL_RC	A/D internal RC clock

Description

ADC Clock Source Select

This data type defines the ADC Clock Source Select.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_CONVERSION_TRIGGER_SOURCE Enumeration

Defines the ADC Conversion Trigger Source.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_CONVERSION_TRIGGER_SAMP_CLEAR,
  ADC_CONVERSION_TRIGGER_INT0_TRANSITION,
  ADC_CONVERSION_TRIGGER_TMR3_COMPARE_MATCH,
  ADC_CONVERSION_TRIGGER_CTMU_EVENT,
  ADC_CONVERSION_TRIGGER_INTERNAL_COUNT
} ADC_CONVERSION_TRIGGER_SOURCE;
```

Members

Members	Description
ADC_CONVERSION_TRIGGER_SAMP_CLEAR	Clearing SAMP bit (full program control)
ADC_CONVERSION_TRIGGER_INT0_TRANSITION	Active transition on INT0 pin (basic sync convert)
ADC_CONVERSION_TRIGGER_TMR3_COMPARE_MATCH	Timer3 compare match
ADC_CONVERSION_TRIGGER_CTMU_EVENT	CTMU event
ADC_CONVERSION_TRIGGER_INTERNAL_COUNT	Internal counter (auto-convert)

Description

ADC Conversion Trigger Source

This data type defines the ADC Conversion Trigger Source.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_BUFFER_MODE Enumeration

Defines the ADC Buffer Mode.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_BUFFER_MODE_ONE_16WORD_BUFFER,
  ADC_BUFFER_MODE_TWO_8WORD_BUFFERS
} ADC_BUFFER_MODE;
```

Members

Members	Description
ADC_BUFFER_MODE_ONE_16WORD_BUFFER	Buffer configured as one 16-word buffer (ADC1BUF0 to ADC1BUFF)
ADC_BUFFER_MODE_TWO_8WORD_BUFFERS	Buffer configured as two 8-word buffers (ADC1BUF0 to ADC1BUF7 and ADC1BUF8 to ADC1BUFF)

Description

ADC Buffer Mode

This data type defines the ADC Buffer Mode.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_RESULT_BUF_STATUS Enumeration

Defines the ADC Result Buffer Status

File

help_plib_adc.h

C

```
typedef enum {
  ADC_FILLING_BUF_0TO7,
  ADC_FILLING_BUF_8TOF
} ADC_RESULT_BUF_STATUS;
```

Members

Members	Description
ADC_FILLING_BUF_0TO7	Buffers 0x0 to 0x7 are getting filled
ADC_FILLING_BUF_8TOF	Buffers 0x8 to 0xF are getting filled

Description

ADC Result Buffer Status

This data type defines the elements that specify which group of eight buffers the ADC is currently filling.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_MUX Enumeration

Defining the different ADC MUX Enumeration.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_MUX_A,
  ADC_MUX_B
} ADC_MUX;
```

Members

Members	Description
ADC_MUX_A	ADC Mux A Selection
ADC_MUX_B	ADC Mux B Selection

Description

ADC MUX Enumeration

This data type defines the different ADC MUX Enumeration.

Remarks

None.

ADC_SAMPLING_MODE Enumeration

Defines the ADC Sampling Mode Select.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_SAMPLING_MODE_MUXA,
  ADC_SAMPLING_MODE_ALTERNATE_INPUT
} ADC_SAMPLING_MODE;
```

Members

Members	Description
ADC_SAMPLING_MODE_MUXA	Always Mux A Sampling Mode
ADC_SAMPLING_MODE_ALTERNATE_INPUT	Alternate Input Sampling Mode

Description

ADC Sampling Mode Select

This data type defines the ADC Sampling Mode Select.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_INPUTS_SCAN Enumeration

Defines the ADC Scan inputs.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_INPUT_SCAN_AN0,
  ADC_INPUT_SCAN_AN1,
  ADC_INPUT_SCAN_AN2,
  ADC_INPUT_SCAN_AN3,
  ADC_INPUT_SCAN_AN4,
  ADC_INPUT_SCAN_AN5,
  ADC_INPUT_SCAN_AN6,
  ADC_INPUT_SCAN_AN7,
  ADC_INPUT_SCAN_AN8,
  ADC_INPUT_SCAN_AN9,
  ADC_INPUT_SCAN_AN10,
  ADC_INPUT_SCAN_AN11,
  ADC_INPUT_SCAN_AN12,
  ADC_INPUT_SCAN_AN13,
  ADC_INPUT_SCAN_AN14,
  ADC_INPUT_SCAN_AN15,
  ADC_INPUT_SCAN_AN16,
```

```
  ADC_INPUT_SCAN_AN17,
  ADC_INPUT_SCAN_AN18,
  ADC_INPUT_SCAN_AN19,
  ADC_INPUT_SCAN_AN20,
  ADC_INPUT_SCAN_AN21,
  ADC_INPUT_SCAN_AN22,
  ADC_INPUT_SCAN_AN23,
  ADC_INPUT_SCAN_AN24,
  ADC_INPUT_SCAN_AN25,
  ADC_INPUT_SCAN_AN26,
  ADC_INPUT_SCAN_AN27,
  ADC_INPUT_SCAN_AN28,
  ADC_INPUT_SCAN_AN29,
  ADC_INPUT_SCAN_AN30,
  ADC_INPUT_SCAN_AN31,
  ADC_INPUT_SCAN_IVREF,
  ADC_INPUT_SCAN_CTMU,
  ADC_INPUT_SCAN_VSS
} ADC_INPUTS_SCAN;
```

Members

Members	Description
ADC_INPUT_SCAN_AN0	Scan input is AN0
ADC_INPUT_SCAN_AN1	Scan input is AN1
ADC_INPUT_SCAN_AN2	Scan input is AN2
ADC_INPUT_SCAN_AN3	Scan input is AN3
ADC_INPUT_SCAN_AN4	Scan input is AN4
ADC_INPUT_SCAN_AN5	Scan input is AN5
ADC_INPUT_SCAN_AN6	Scan input is AN6
ADC_INPUT_SCAN_AN7	Scan input is AN7
ADC_INPUT_SCAN_AN8	Scan input is AN8
ADC_INPUT_SCAN_AN9	Scan input is AN9
ADC_INPUT_SCAN_AN10	Scan input is AN10
ADC_INPUT_SCAN_AN11	Scan input is AN11
ADC_INPUT_SCAN_AN12	Scan input is AN12
ADC_INPUT_SCAN_AN13	Scan input is AN13
ADC_INPUT_SCAN_AN14	Scan input is AN14
ADC_INPUT_SCAN_AN15	Scan input is AN15
ADC_INPUT_SCAN_AN16	Scan input is AN16
ADC_INPUT_SCAN_AN17	Scan input is AN17
ADC_INPUT_SCAN_AN18	Scan input is AN18
ADC_INPUT_SCAN_AN19	Scan input is AN19
ADC_INPUT_SCAN_AN20	Scan input is AN20
ADC_INPUT_SCAN_AN21	Scan input is AN21
ADC_INPUT_SCAN_AN22	Scan input is AN22
ADC_INPUT_SCAN_AN23	Scan input is AN23
ADC_INPUT_SCAN_AN24	Scan input is AN24
ADC_INPUT_SCAN_AN25	Scan input is AN25
ADC_INPUT_SCAN_AN26	Scan input is AN26
ADC_INPUT_SCAN_AN27	Scan input is AN27
ADC_INPUT_SCAN_AN28	Scan input is AN28
ADC_INPUT_SCAN_AN29	Scan input is AN29
ADC_INPUT_SCAN_AN30	Scan input is AN30
ADC_INPUT_SCAN_AN31	Scan input is AN31
ADC_INPUT_SCAN_IVREF	Scan input is IVref
ADC_INPUT_SCAN_CTMU	Scan input is CTMU input
ADC_INPUT_SCAN_VSS	Scan input is VSS

Description

ADC Scan Inputs Enumeration

This data type defines the ADC Scan inputs.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_RESULT_FORMAT Enumeration

Defines the ADC Result Format.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_RESULT_FORMAT_INTEGER_16BIT,
  ADC_RESULT_FORMAT_SIGNED_INTEGER_16BIT,
  ADC_RESULT_FORMAT_FRACTIONAL_16BIT,
  ADC_RESULT_FORMAT_SIGNED_FRACTIONAL_16BIT,
  ADC_RESULT_FORMAT_INTEGER_32BIT,
  ADC_RESULT_FORMAT_SIGNED_INTEGER_32BIT,
  ADC_RESULT_FORMAT_FRACTIONAL_32BIT,
  ADC_RESULT_FORMAT_SIGNED_FRACTIONAL_32BIT
} ADC_RESULT_FORMAT;
```

Members

Members	Description
ADC_RESULT_FORMAT_INTEGER_16BIT	Integer 16-bit
ADC_RESULT_FORMAT_SIGNED_INTEGER_16BIT	Signed Integer 16-bit
ADC_RESULT_FORMAT_FRACTIONAL_16BIT	Fractional 16-bit
ADC_RESULT_FORMAT_SIGNED_FRACTIONAL_16BIT	Signed Fractional 16-bit
ADC_RESULT_FORMAT_INTEGER_32BIT	Integer 32-bit
ADC_RESULT_FORMAT_SIGNED_INTEGER_32BIT	Signed Integer 32-bit
ADC_RESULT_FORMAT_FRACTIONAL_32BIT	Fractional 32-bit
ADC_RESULT_FORMAT_SIGNED_FRACTIONAL_32BIT	Signed Fractional 32-bit

Description

ADC Result Format

This data type defines the ADC Result Format.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_INPUTS_POSITIVE Enumeration

Defines the ADC inputs.

File

help_plib_adc.h

C

```
typedef enum {
  ADC_INPUT_POSITIVE_AN0,
  ADC_INPUT_POSITIVE_AN1,
  ADC_INPUT_POSITIVE_AN2,
  ADC_INPUT_POSITIVE_AN3,
  ADC_INPUT_POSITIVE_AN4,
  ADC_INPUT_POSITIVE_AN5,
  ADC_INPUT_POSITIVE_AN6,
  ADC_INPUT_POSITIVE_AN7,
  ADC_INPUT_POSITIVE_AN8,
  ADC_INPUT_POSITIVE_AN9,
  ADC_INPUT_POSITIVE_AN10,
  ADC_INPUT_POSITIVE_AN11,
  ADC_INPUT_POSITIVE_AN12,
  ADC_INPUT_POSITIVE_AN13,
  ADC_INPUT_POSITIVE_AN14,
```

```
    ADC_INPUT_POSITIVE_AN15,
    ADC_INPUT_POSITIVE_AN16,
    ADC_INPUT_POSITIVE_AN17,
    ADC_INPUT_POSITIVE_AN18,
    ADC_INPUT_POSITIVE_AN19,
    ADC_INPUT_POSITIVE_AN20,
    ADC_INPUT_POSITIVE_AN21,
    ADC_INPUT_POSITIVE_AN22,
    ADC_INPUT_POSITIVE_AN23,
    ADC_INPUT_POSITIVE_AN24,
    ADC_INPUT_POSITIVE_AN25,
    ADC_INPUT_POSITIVE_AN26,
    ADC_INPUT_POSITIVE_AN27,
    ADC_INPUT_POSITIVE_AN28,
    ADC_INPUT_POSITIVE_AN29,
    ADC_INPUT_POSITIVE_AN30,
    ADC_INPUT_POSITIVE_AN31,
    ADC_INPUT_POSITIVE_AN32,
    ADC_INPUT_POSITIVE_AN33,
    ADC_INPUT_POSITIVE_AN34,
    ADC_INPUT_POSITIVE_AN35,
    ADC_INPUT_POSITIVE_AN36,
    ADC_INPUT_POSITIVE_AN37,
    ADC_INPUT_POSITIVE_AN38,
    ADC_INPUT_POSITIVE_AN39,
    ADC_INPUT_POSITIVE_AN40,
    ADC_INPUT_POSITIVE_AN41,
    ADC_INPUT_POSITIVE_AN42,
    ADC_INPUT_POSITIVE_AN43,
    ADC_INPUT_POSITIVE_AN44,
    ADC_INPUT_POSITIVE_AN45,
    ADC_INPUT_POSITIVE_AN46,
    ADC_INPUT_POSITIVE_AN47,
    ADC_INPUT_POSITIVE_CTMU,
    ADC_INPUT_POSITIVE_IVREF,
    ADC_INPUT_POSITIVE_OPEN
} ADC_INPUTS_POSITIVE;
```

Members

Members	Description
ADC_INPUT_POSITIVE_AN0	Positive input is AN0
ADC_INPUT_POSITIVE_AN1	Positive input is AN1
ADC_INPUT_POSITIVE_AN2	Positive input is AN2
ADC_INPUT_POSITIVE_AN3	Positive input is AN3
ADC_INPUT_POSITIVE_AN4	Positive input is AN4
ADC_INPUT_POSITIVE_AN5	Positive input is AN5
ADC_INPUT_POSITIVE_AN6	Positive input is AN6
ADC_INPUT_POSITIVE_AN7	Positive input is AN7
ADC_INPUT_POSITIVE_AN8	Positive input is AN8
ADC_INPUT_POSITIVE_AN9	Positive input is AN9
ADC_INPUT_POSITIVE_AN10	Positive input is AN10
ADC_INPUT_POSITIVE_AN11	Positive input is AN11
ADC_INPUT_POSITIVE_AN12	Positive input is AN12
ADC_INPUT_POSITIVE_AN13	Positive input is AN13
ADC_INPUT_POSITIVE_AN14	Positive input is AN14
ADC_INPUT_POSITIVE_AN15	Positive input is AN15
ADC_INPUT_POSITIVE_AN16	Positive input is AN16
ADC_INPUT_POSITIVE_AN17	Positive input is AN17
ADC_INPUT_POSITIVE_AN18	Positive input is AN18
ADC_INPUT_POSITIVE_AN19	Positive input is AN19
ADC_INPUT_POSITIVE_AN20	Positive input is AN20
ADC_INPUT_POSITIVE_AN21	Positive input is AN21
ADC_INPUT_POSITIVE_AN22	Positive input is AN22
ADC_INPUT_POSITIVE_AN23	Positive input is AN23
ADC_INPUT_POSITIVE_AN24	Positive input is AN24

ADC_INPUT_POSITIVE_AN25	Positive input is AN25
ADC_INPUT_POSITIVE_AN26	Positive input is AN26
ADC_INPUT_POSITIVE_AN27	Positive input is AN27
ADC_INPUT_POSITIVE_AN28	Positive input is AN28
ADC_INPUT_POSITIVE_AN29	Positive input is AN29
ADC_INPUT_POSITIVE_AN30	Positive input is AN30
ADC_INPUT_POSITIVE_AN31	Positive input is AN31
ADC_INPUT_POSITIVE_AN32	Positive input is AN32
ADC_INPUT_POSITIVE_AN33	Positive input is AN33
ADC_INPUT_POSITIVE_AN34	Positive input is AN34
ADC_INPUT_POSITIVE_AN35	Positive input is AN35
ADC_INPUT_POSITIVE_AN36	Positive input is AN36
ADC_INPUT_POSITIVE_AN37	Positive input is AN37
ADC_INPUT_POSITIVE_AN38	Positive input is AN38
ADC_INPUT_POSITIVE_AN39	Positive input is AN39
ADC_INPUT_POSITIVE_AN40	Positive input is AN40
ADC_INPUT_POSITIVE_AN41	Positive input is AN41
ADC_INPUT_POSITIVE_AN42	Positive input is AN42
ADC_INPUT_POSITIVE_AN43	Positive input is AN43
ADC_INPUT_POSITIVE_AN44	Positive input is AN44
ADC_INPUT_POSITIVE_AN45	Positive input is AN45
ADC_INPUT_POSITIVE_AN46	Positive input is AN46
ADC_INPUT_POSITIVE_AN47	Positive input is AN47
ADC_INPUT_POSITIVE_CTMU	Positive input is from CTMU
ADC_INPUT_POSITIVE_IVREF	Positive input is IVref
ADC_INPUT_POSITIVE_OPEN	Positive input is Open

Description

ADC Inputs Enumeration

This data type defines the ADC inputs.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADC_ACQUISITION_TIME Type

Data type defining the different ADC acquisition times by which the ADC can be configured.

File

plib_adc.h

C

```
typedef uint32_t ADC_ACQUISITION_TIME;
```

Description

ADC Acquisition Time Selection Data Type

This data type defines the different ADC acquisition times by which the ADC can be configured.

Remarks

None.

ADC_CONVERSION_CLOCK Type

Data type defines the different ADC Conversion clock

File

plib_adc.h

C

```
typedef uint32_t ADC_CONVERSION_CLOCK;
```

Description

ADC Conversion Clock Selection Data Type

This data type defines the different ADC Conversion clock

Remarks

None.

ADC_SAMPLE Type

Data type defining the size of the ADC sample register.

File

plib_adc.h

C

```
typedef uint32_t ADC_SAMPLE;
```

Description

ADC Sample size

This data type defines the size of the ADC sample register.

Remarks

None.

Files

Files

Name	Description
plib_adc.h	ADC PLIB interface header for definitions common to the ADC peripheral.
help_plib_adc.h	This is file help_plib_adc.h.

Description

This section lists the source and header files used by the library.

plib_adc.h

ADC PLIB interface header for definitions common to the ADC peripheral.

Functions

	Name	Description
	PLIB_ADC_CalibrationDisable	Normal ADC module operation (no calibration is performed).
	PLIB_ADC_CalibrationEnable	Calibration is performed on the next ADC conversion.
	PLIB_ADC_ConversionClockGet	Obtains the conversion clock.
	PLIB_ADC_ConversionClockSet	Sets the ADC module conversion clock.
	PLIB_ADC_ConversionClockSourceSelect	Selects the ADC module conversion clock source.
	PLIB_ADC_ConversionHasCompleted	Provides the conversion completion status of the ADC.
	PLIB_ADC_ConversionStart	Starts ADC module manual conversion process.
	PLIB_ADC_ConversionStopSequenceDisable	Normal conversion sequence.
	PLIB_ADC_ConversionStopSequenceEnable	Stop conversion sequence (when the first ADC module interrupt is generated).
	PLIB_ADC_ConversionTriggerSourceSelect	Selects the conversion trigger source.
	PLIB_ADC_Disable	ADC module is disabled (turned OFF).
	PLIB_ADC_Enable	ADC module is enabled (turned ON).
	PLIB_ADC_ExistsCalibrationControl	Identifies whether the CalibrationControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionClock	Identifies whether the ConversionClock feature exists on the ADC module

	PLIB_ADC_ExistsConversionClockSource	Identifies whether the ConversionClockSource feature exists on the ADC module
	PLIB_ADC_ExistsConversionControl	Identifies whether the ConversionControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionStatus	Identifies whether the ConversionStatus feature exists on the ADC module
	PLIB_ADC_ExistsConversionStopSequenceControl	Identifies whether the ConversionStopSequenceControl feature exists on the ADC module
	PLIB_ADC_ExistsConversionTriggerSource	Identifies whether the ConversionTriggerSource feature exists on the ADC module
	PLIB_ADC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADC module
	PLIB_ADC_ExistsMuxChannel0NegativeInput	Identifies whether the MuxChannel0NegativeInput feature exists on the ADC module
	PLIB_ADC_ExistsMuxChannel0PositiveInput	Identifies whether the MuxChannel0PositiveInput feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanControl	Identifies whether the MuxInputScanControl feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanSelect	Identifies whether the MuxInputScanSelect feature exists on the ADC module
	PLIB_ADC_ExistsMuxInputScanSelectExtended	Identifies whether the MuxInputScanSelectExtended feature exists on the ADC module
	PLIB_ADC_ExistsResultBufferFillStatus	Identifies whether the ResultBufferFillStatus feature exists on the ADC module
	PLIB_ADC_ExistsResultBufferMode	Identifies whether the ResultBufferMode feature exists on the ADC module
	PLIB_ADC_ExistsResultFormat	Identifies whether the ResultFormat feature exists on the ADC module
	PLIB_ADC_ExistsResultGetByIndex	Identifies whether the ResultGetByIndex feature exists on the ADC module
	PLIB_ADC_ExistsSamplesPerInterruptSelect	Identifies whether the SamplesPerInterruptSelect feature exists on the ADC module
	PLIB_ADC_ExistsSamplingAcquisitionTime	Identifies whether the SamplingAcquisitionTime feature exists on the ADC module
	PLIB_ADC_ExistsSamplingAutoStart	Identifies whether the SamplingAutoStart feature exists on the ADC module
	PLIB_ADC_ExistsSamplingControl	Identifies whether the SamplingControl feature exists on the ADC module
	PLIB_ADC_ExistsSamplingModeControl	Identifies whether the SamplingModeControl feature exists on the ADC module
	PLIB_ADC_ExistsSamplingStatus	Identifies whether the SamplingStatus feature exists on the ADC module
	PLIB_ADC_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the ADC module
	PLIB_ADC_ExistsVoltageReference	Identifies whether the VoltageReference feature exists on the ADC module
	PLIB_ADC_InputScanMaskAdd	Select ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskAddExtended	Select extended ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskRemove	Omits ADC analog channel for input scan.
	PLIB_ADC_InputScanMaskRemoveExtended	Omits extended ADC analog channel for input scan.
	PLIB_ADC_MuxAInputScanDisable	Do not scan input selections for CH0+ of MUX A.
	PLIB_ADC_MuxAInputScanEnable	Scan input selections for CH0+ of MUX A.
	PLIB_ADC_MuxChannel0InputNegativeSelect	Channel 0 negative input select for multiplexer setting.
	PLIB_ADC_MuxChannel0InputPositiveSelect	Channel 0 positive input select for multiplexer setting.
	PLIB_ADC_ResultBufferModeSelect	Selects the result buffer mode.
	PLIB_ADC_ResultBufferStatusGet	Provides the buffer fill status.
	PLIB_ADC_ResultFormatSelect	Selects the result format.
	PLIB_ADC_ResultGetByIndex	Provides the ADC conversion result based on the buffer index.
	PLIB_ADC_SampleAcquisitionTimeSet	Sets the ADC acquisition/auto-sample time in TADs.
	PLIB_ADC_SampleAutoStartDisable	Sampling auto-start is disabled.
	PLIB_ADC_SampleAutoStartEnable	Sampling auto-start is enabled.
	PLIB_ADC_SamplesPerInterruptSelect	Interrupts at the completion of conversion for each nth sample.
	PLIB_ADC_SamplingIsActive	Provides the ADC sampling status.
	PLIB_ADC_SamplingModeSelect	Enable the selected sampling mode.
	PLIB_ADC_SamplingStart	Sampling is enabled.
	PLIB_ADC_SamplingStop	Holding is enabled.
	PLIB_ADC_StopInIdleDisable	Continue ADC module operation when the device is in Idle mode.
	PLIB_ADC_StopInIdleEnable	Discontinue ADC module operation when device enters Idle mode.
	PLIB_ADC_VoltageReferenceSelect	Voltage reference configuration.

Types

	Name	Description
	ADC_ACQUISITION_TIME	Data type defining the different ADC acquisition times by which the ADC can be configured.
	ADC_CONVERSION_CLOCK	Data type defines the different ADC Conversion clock
	ADC_SAMPLE	Data type defining the size of the ADC sample register.

Description

ADC Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the ADC PLIB for all families of Microchip microcontrollers. The definitions in this file are common to the ADC peripheral.

File Name

plib_adc.h

Company

Microchip Technology Inc.

help_plib_adc.h

Enumerations

	Name	Description
	ADC_BUFFER_MODE	Defines the ADC Buffer Mode.
	ADC_CLOCK_SOURCE	Defines the ADC Clock Source Select.
	ADC_CONVERSION_TRIGGER_SOURCE	Defines the ADC Conversion Trigger Source.
	ADC_INPUTS_NEGATIVE	Defines the different ADC Negative Input Enumeration.
	ADC_INPUTS_POSITIVE	Defines the ADC inputs.
	ADC_INPUTS_SCAN	Defines the ADC Scan inputs.
	ADC_MODULE_ID	Identifies the available ADC modules.
	ADC_MUX	Defining the different ADC MUX Enumeration.
	ADC_RESULT_BUF_STATUS	Defines the ADC Result Buffer Status
	ADC_RESULT_FORMAT	Defines the ADC Result Format.
	ADC_SAMPLES_PER_INTERRUPT	Defining the Samples Per Interrupt Enumeration.
	ADC_SAMPLING_MODE	Defines the ADC Sampling Mode Select.
	ADC_VOLTAGE_REFERENCE	Defining the different ADC Voltage Reference by which the ADC can be configured.

Description

This is file help_plib_adc.h.

ADCHS Peripheral Library

This section describes the 12-bit High-Speed Successive Approximation Register (SAR) Analog-to-Digital Converter (ADCHS) Peripheral Library.

Introduction

This library provides a low-level abstraction of the High-Speed SAR ADC (ADCHS) Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the registers, thereby hiding differences from one microcontroller variant to another.

Description

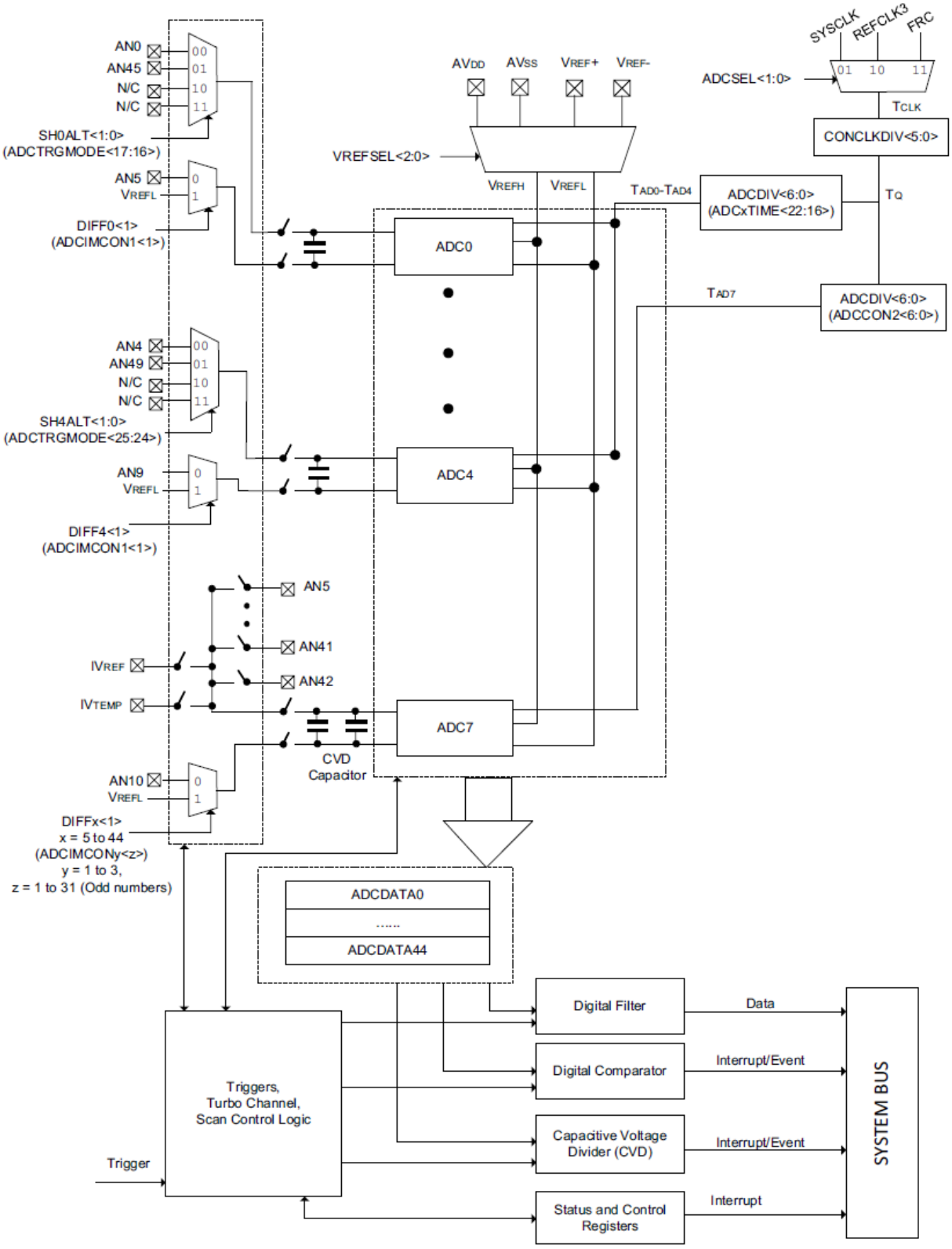
The ADCHS is used to convert an analog input into a digital number that represents that input voltage. When the input is periodically converted, the result is a sequence of digital values that have converted a continuous-time and continuous-amplitude analog signal to a discrete-time and discrete-amplitude digital signal.

This particular ADC architecture consists of multiple SAR channels (maximum of eight). The SAR channels are numbered from Channel 0 to Channel 7. Out of these channels, Channels 0 through 6 have dedicated sample and hold (S&H) circuits connected to a single (selectable) analog input. These SAR channels work independently from each other and are designed to capture time sensitive or transitory analog signals. The remaining channel, Channel 7, has its S&H circuit connected to multiple analog inputs through a multiplexer. Channel 7 is used to convert analog signals that do not need a high conversion rate.

Features of the ADCHS module include:

- Channels 0 through 6 have selectable analog inputs (default, alternate), which can be connected to a S&H circuit
- Channel 7 has multiple analog inputs connected to a S&H circuit
- Analog inputs connected to Channels 0 through 6 are Class 1 analog inputs. Analog inputs connected to Channel 7 are classified as Class 2 and Class 3 analog inputs.
- Class 1 analog inputs have their individual trigger source and are converted by dedicated Channels (0 through 6).
- Class 2 analog inputs have their individual trigger source, but are converted by shared channel-7
- Class 3 analog inputs do not have individual trigger source and are converted by scan mode using Channel 7
- Analog input scan is available, which allows a single trigger to start a sample/conversion sequence of multiple analog inputs using a predefined scan list
- The inputs to a S&H can be configured as unipolar/bipolar and single-ended/differential mode
- For each channel, the converted data resolution can be set as 6, 8, 10, or 12-bits
- Each channel (0 through 7) can have its individual conversion clock and sample time setting
- A voltage reference is required for the ADCHS module. The voltage reference setting is a global setting for the ADCHS module and does not vary across channels.
- Result registers store the conversion results and can be read by the software application. Software is notified of ready results by either polling or through the use of interrupts.
- For Channels 0 through 6, the converted data can be stored in an optional FIFO or directly to RAM using the dedicated DMA interface
- Optional Digital Filters can be used to provide increased resolution at the sacrifice of sample rate. Digital Filters also have an averaging mode.
- Optional Digital Comparators can be used to test conversion results against set high and low limits, which are independent of software, and generate interrupts based on predefined conditions
- Digital Comparators also have Capacitive Voltage Divider (CVD) mode, which can be used to detect a touch event in touch sense application
- Early interrupt generation, providing extremely fast processing of converted data

A block diagram of the ADCHS module is provided in the following figure.



Using the Library

This topic describes the basic architecture of the ADCHS Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_adchs.h`

The interface to the ADCHS Peripheral library is defined in the `plib_adchs.h` header file, which is included by the `peripheral.h` file. Any C

language source (`.c`) file that uses the ADCHS Peripheral library must include `peripheral.h`.

Library File:

The ADCHS Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

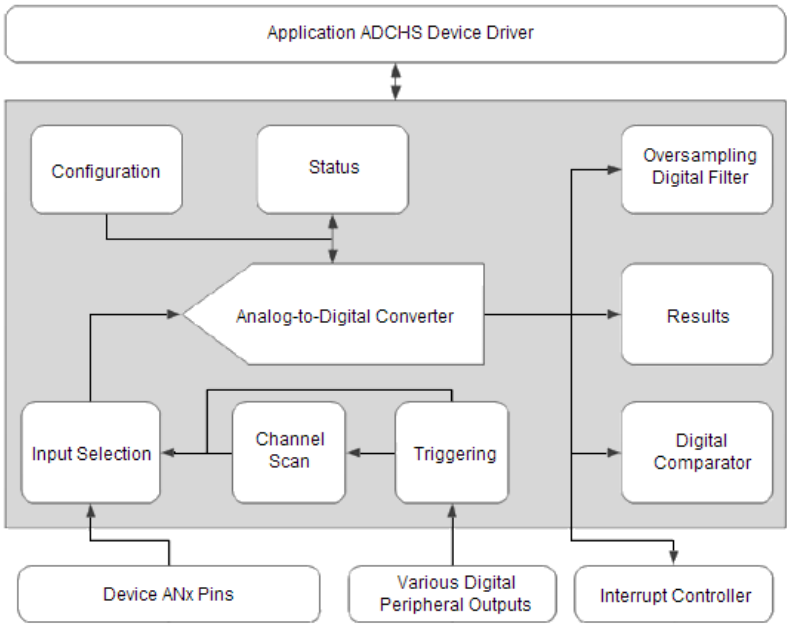
Hardware Abstraction Model

This library provides the low-level abstraction of the ADCHS module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all the functionality of the available ADCHS module on the device. Refer to the **"ADC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each ADCHS module on the device.

Description

High-Speed SAR Analog-to-Digital Converter (ADCHS) Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the ADCHS module.

Library Interface Section	Description
Configuration Functions	These functions are used to configure, enable, and disable the ADCHS.
Analog Input Functions	These functions are used to configure analog input features.
Analog Input Selection Functions	These functions select and configure the ANx inputs to the ADCHS.
CVD Functions	These functions configure the Capacitive Voltage Divider (CVD) mode.
DMA Functions	These functions are used to configure and enable/disable the DMA, enable/ disable interrupts linked to the DMA, and to get the DMA status.
Channel Related Functions	These functions configure Channels 0 through 7, enable/disable the analog and digital circuitry, and control interrupt generation.
Digital Comparator Functions	These functions are used to configure and query the digital comparators.
Digital Filter Functions	These functions are used to configure the Oversampling Digital Filter and fetch results from it.
FIFO Functions	These functions configure the FIFO and control the interrupt generation.
Status Functions	These functions return status information related to the ADCHS.
Interrupt Functions	These functions control interrupt generation.

Turbo Mode Functions	These functions are used to configure, enable, and disable Turbo mode.
Triggering Functions	These functions configure the trigger source for various cores.
Voltage Reference Functions	These functions control the Voltage Reference (VREF) input for the ADC cores and query VREF status and interrupt generation.
Feature Existence Functions	These functions determine whether or not a particular feature is supported by the device.

How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

Functionality

This topic describes the functionality of the ADCHS Peripheral Library.

Description

Analog-to-Digital conversion using the ADCHS involves the following three steps:

1. Sampling of the input signal.
2. Capture of the input signal by the S&H circuit and transfer to the converter.
3. Conversion of the analog signal to its digital representation.

Sampling of the input signal involves charging of the S&H capacitor. The sampling time must be adequate so that the capacitor charges to a value equal to the input voltage. At the appropriate time, the input is disconnected and the capacitor voltage is transferred to the converter. The converter then digitizes the analog signal and provides the result. The converter requires a clock source (common input to individual clock setting of all channels) and a reference voltage. The common input clock is referred to as the control clock and has a period of TQ. The control clock and reference voltage sources are selectable, as well as the clock prescaling that determines the TQ.

The ADCHS has multiple SAR channels. Each channel has independent clocks named TAD0 through TAD7. The analog inputs connected to the SAR channels can be divided as Class 1, Class 2, and Class 3.

- Class 1 inputs are associated with a SAR Channel 0 through 6. Each SAR channel has a single (selectable) Class 1 input associated with it. Each Class 1 input has a unique trigger selection register. Class 1 inputs can also be part of a channel scan list, triggered by the common scan trigger source.
- Class 2 inputs are connected to Channel 7 and are either individually triggered or as part of a channel scan list. When used individually their trigger source is selected by their unique trigger register.
- Class 3 inputs are connected to Channel 7 and are used in channel scan exclusively. They share a common trigger source. When using channel scan it is possible to combine Class 1, Class 2, and Class 3 inputs in the scan list.

Note: For details regarding configuration and triggering options was well as sampling requirements, refer to the **"ADC"** chapter in the specific device data sheet and the family reference manual section specified in that chapter.

Example - Simultaneous Sampling and Conversion of Three Class 1 Inputs

Simultaneous sampling and conversion is used when the application requires capture of more than one signal at the same instance of time. Simultaneous sampling and conversion requires the use of multiple Class 1 inputs where each is assigned the same trigger source. The following example illustrates simultaneous sampling of AN0, AN1 and AN2 using the global software trigger as the trigger source. The respective SAR channels are configured for single-ended input and a unipolar (unsigned) output.

No interrupts are used so the results are polled for ready status.

```
{
    int result[3];                              // storage for results
    // Configure the ADC
    PLIB_ADCHS_Setup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_VREF_AVDD_AVSS,                   // AVDD and AVSS as reference
        ADCHS_CHARGEPUMP_DISABLE,               // No charge pump
        ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL,    // Use fractional format
        true,                                   // Do stop in idle
        ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE,    // Enable Fast synchronous system clock
        ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE, // Disable Fast synchronous peripheral clock
        ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS,  // vector shift unused, interrupt not used
        0,                                      // vector base address unused, interrupt not used
```

```
    ADCHS_CLOCK_SOURCE_SYSCLK,              // SYSCLK is the clock source
    1,                                      // TQ = 1/SYSCLK * 2
    ADCHS_WARMUP_CLOCK_32                   // Warm-up time = 32 clocks
);
// Configure the ADC SAR Channel-0
PLIB_ADCHS_ChannelSetup
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_0,                        // Channel - 0
    ADCHS_DATA_RESOLUTION_12BIT,            // resolution is set to 12bits
    1,                                      // clock divider bit is, TAD0 = 2 * TQ
    1,                                      // Sample time is 3 * TAD
    ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1     // 1 clock early interrupt (ignored, interrupt not used)
);
// Configure the synchronous sampling for Channel-0
if(false == PLIB_ADCHS_ChannelTriggerSampleSelect( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0,                    // channel - 0
        ADCHS_CHANNEL_SYNC_SAMPLING))       // Synchronous sampling selected
        {
            // error has occurred
            while(1);
        }
// Configure the ADC SAR Channel-1
PLIB_ADCHS_ChannelSetup
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_1, // Channel - 1
    ADCHS_DATA_RESOLUTION_12BIT,            // resolution is set to 12bits
    1,                                      // clock divider bit is, TAD1 = 2 * TQ
    1,                                      // Sample time is 3 * TAD
    ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1     // 1 clock early interrupt (ignored, interrupt not used)
);
// Configure the synchronous sampling for Channel-1
if(false == PLIB_ADCHS_ChannelTriggerSampleSelect( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_1,                    // channel - 1
        ADCHS_CHANNEL_SYNC_SAMPLING))       // Synchronous sampling selected
        {
            // error has occurred
            while(1);
        }
// Configure the ADC SAR Channel-2
PLIB_ADCHS_ChannelSetup
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_2,                        // Channel - 2
    ADCHS_DATA_RESOLUTION_12BIT,            // resolution is set to 12bits
    1,                                      // clock divider bit is, TAD2 = 2 * TQ
    1,                                      // Sample time is 3 * TAD
    ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1     // 1 clock early interrupt (ignored, interrupt not used)
);
// Configure the synchronous sampling for Channel-2
if(false == PLIB_ADCHS_ChannelTriggerSampleSelect( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_2,                    // channel - 2
        ADCHS_CHANNEL_SYNC_SAMPLING))       // Synchronous sampling selected
        {
            // error has occurred
            while(1);
        }
// Select inputs for Channel
if(false == PLIB_ADCHS_ChannelInputSelect
            (
                MY_ADCHS_INSTANCE,
                ADCHS_CHANNEL_0,
                ADCHS_CHANNEL_0_DEFAULT_INP_AN0    // Select AN0 for channel-0
            ))
            {
                // error has occurred
                while(1);
            }
```

```
if(false == PLIB_ADCHS_ChannelInputSelect
            (
                MY_ADCHS_INSTANCE,
                ADCHS_CHANNEL_1,
                ADCHS_CHANNEL_0_DEFAULT_INP_AN1    // Select AN1 for channel-1
            ))
            {
                // error has occurred
                while(1);
            }
if(false == PLIB_ADCHS_ChannelInputSelect
            (
                MY_ADCHS_INSTANCE,
                ADCHS_CHANNEL_2,
                ADCHS_CHANNEL_0_DEFAULT_INP_AN2    // Select AN2 for channel-2
            ))
            {
                // error has occurred
                while(1);
            }
// Select input mode for AN0, AN1, AN2
PLIB_ADCHS_AnalogInputModeSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_AN0,
    ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
);
PLIB_ADCHS_AnalogInputModeSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_AN1,
    ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
);
PLIB_ADCHS_AnalogInputModeSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_AN2,
    ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
);
// Select AN0, AN1 and AN2 as edge trigger
PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN0 );
PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN1 );
PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN2 );
// Select AN0, AN1 and AN2 to be software triggered
PLIB_ADCHS_AnalogInputTriggerSourceSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_CLASS12_AN0,
    ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
);
PLIB_ADCHS_AnalogInputTriggerSourceSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_CLASS12_AN1,
    ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
);
PLIB_ADCHS_AnalogInputTriggerSourceSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_CLASS12_AN2,
    ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
);
// Enable ADC
PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
// Check VREF to be ready
While(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));
// Check for VREF Fault
While(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE));
// Enable analog circuit for channel-0, 1 and 2
```

```
    PLIB_ADCHS_ChannelAnalogFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0
    );
    PLIB_ADCHS_ChannelAnalogFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_1
    );
    PLIB_ADCHS_ChannelAnalogFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_2
    );
    // Wait for the channels to be ready
    while(!PLIB_ADCHS_ChannelIsReady
        (
            MY_ADCHS_INSTANCE,
            ADCHS_CHANNEL_0
        )
    );
    while(!PLIB_ADCHS_ChannelIsReady
        (
            MY_ADCHS_INSTANCE,
            ADCHS_CHANNEL_1
        )
    );
    while(!PLIB_ADCHS_ChannelIsReady
        (
            MY_ADCHS_INSTANCE,
            ADCHS_CHANNEL_2
        )
    );
    // Enable digital circuit for channels 0, 1, 2
    PLIB_ADCHS_ChannelDigitalFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0
    );
    PLIB_ADCHS_ChannelDigitalFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_1
    );
    PLIB_ADCHS_ChannelDigitalFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_2
    );
    while(1)
    {
        // Enable global software trigger
        PLIB_ADCHS_GlobalSoftwareTriggerEnable( MY_ADCHS_INSTANCE );
        // Wait for conversion complete for AN0
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN0));
        result[0] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN0 );
        // Wait for conversion complete for AN1
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN1));
        result[1] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN1 );
        // Wait for conversion complete for AN2
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN2));
        result[2] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN2 );
    }
    return(1);
}
```

Example - Channel Scanning

Channel scanning is used in applications where precise timing of the sample to an external source is not needed. Channel scanning allows a large number of analog inputs to be sampled and converted in sequence each time a single trigger occurs.

The following example illustrates how to configure channel scanning of multiple Class 2 and Class 3 inputs using Channel 7. This example uses inputs AN8, AN31 through AN33. AN8 is Class 2 inputs. AN31 to AN33 are Class 3 inputs. The global software trigger is used to initiate the scan. A 3 TAD7 sample time is specified for Channel 7, which is configured for single-ended operation and a unipolar output. No interrupts are used so the results are polled for ready status.

```
{
    int result[4]; // storage for results
    // Configure the ADC
    PLIB_ADCHS_Setup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_VREF_AVDD_AVSS,                    // AVDD and AVSS as reference
        ADCHS_CHARGEPUMP_DISABLE,                // No charge pump
        ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL,     // Use fractional format
        true,                                    // Do stop in idle
        ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE,     // Enable Fast synchronous system clock
        ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE, // Disable Fast synchronous peripheral clock
        ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS,   // vector shift unused, interrupt not used
        0,                                       // vector base address unused, interrupt not used
        ADCHS_CLOCK_SOURCE_SYSCLK,               // SYSCLK is the clock source
        1,                                       // TQ = 1/SYSCLK * 2
        ADCHS_WARMUP_CLOCK_32                    // Warm-up time = 32 clocks
    );
    // Configure the ADC SAR Channel-7
    PLIB_ADCHS_ChannelSetup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7,                         // Channel - 7
        ADCHS_DATA_RESOLUTION_12BIT,             // resolution is set to 12bits
        1,                                       // clock divider bit is, TAD7 = 2 * TQ
        1,                                       // Sample time is 3 * TAD7
        ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1      // 1 clock early interrupt (ignored, interrupt not used)
    );
    // Select analog channel AN8 and scan sequence to be triggered with global scan trigger
    PLIB_ADCHS_AnalogInputScanSetup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN8,
        ADCHS_SCAN_TRIGGER_SENSITIVE_EDGE,
        ADCHS_SCAN_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
    );
    // Now, add further analog inputs, AN31, AN32, AN33 to scan list
    PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN31);
    PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN32);
    PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN33);
    // Select input mode for AN8, AN31, AN32, AN33
    PLIB_ADCHS_AnalogInputModeSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN8,
        ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
    );
    PLIB_ADCHS_AnalogInputModeSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN31,
        ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
    );
    PLIB_ADCHS_AnalogInputModeSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN32,
        ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
    );
```

```
    PLIB_ADCHS_AnalogInputModeSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN33,
        ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
    );
    // Select individual trigger for class-2 channels viz. AN8 to be scan trigger.
    // AN31, AN32 and AN33 are class-3 channels, and do not have individual trigger
    PLIB_ADCHS_AnalogInputTriggerSourceSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CLASS12_AN8,
        ADCHS_TRIGGER_SOURCE_SCAN
    );
    // Select AN8 to be edge trigger (not level trigger).
    // Calling this function is needed for class-1 and 2 analog inputs
    PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN8);
    // Enable ADC
    PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
    // Check VREF to be ready
    While(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));
    // Check for VREF Fault
    While(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE));
    // Enable analog circuit for channel-7
    PLIB_ADCHS_ChannelAnalogFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7
    );
    // Wait for the channel-7 to be ready
    while(!PLIB_ADCHS_ChannelIsReady
        (
            MY_ADCHS_INSTANCE,
            ADCHS_CHANNEL_7
        )
    );
    // Enable digital circuit for channels 7
    PLIB_ADCHS_ChannelDigitalFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7
    );
    while(1)
    {
        // Enable global software trigger
        PLIB_ADCHS_GlobalSoftwareTriggerEnable( MY_ADCHS_INSTANCE );
        // Wait for conversion complete for AN8
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN8));
        result[0] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN8 );
        // Wait for conversion complete for AN31
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN31));
        result[1] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN31 );
        // Wait for conversion complete for AN32
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN32));
        result[2] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN32 );
        // Wait for conversion complete for AN33
        while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN33));
        result[3] = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN33 );
    }
    return(1);
}
```

Example - Digital Filter

The Digital Oversampling filter can be used to increase resolution of the conversion result at the expense of throughput. The following example shows how two extra bits of resolution can be obtained using 16x oversampling (sixteen samples are used to create one higher resolution result). AN0 is used for the input. The sampling time for the retriggers is set to 3 TAD0.

```
{
```

```
int result; // storage for results
bool eventFlag = false; // Digital comparator event flag
// Configure the ADC
PLIB_ADCHS_Setup
(
    MY_ADCHS_INSTANCE,
    ADCHS_VREF_AVDD_AVSS,                    // AVDD and AVSS as reference
    ADCHS_CHARGEPUMP_DISABLE,                // No charge pump
    ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL,     // Use fractional format
    true,                                    // Do stop in idle
    ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE,     // Enable Fast synchronous system clock
    ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE, // Disable Fast synchronous peripheral clock
    ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS,   // vector shift not used since interrupt not used
    0,                                       // vector base address unused, interrupt not used
    ADCHS_CLOCK_SOURCE_SYSCLK,               // SYSCLK is the clock source
    1,                                       // TQ = 1/SYSCLK * 2
    ADCHS_WARMUP_CLOCK_32                    // Warm-up time = 32 clocks
);
// Configure the ADC SAR Channel-0
PLIB_ADCHS_ChannelSetup
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_0,                         // Channel - 0
    ADCHS_DATA_RESOLUTION_12BIT,             // resolution is set to 12bits
    1,                                       // clock divider bit is, TAD0 = 2 * TQ
    1,                                       // Sample time is 3 * TAD
    ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1      // 1 clock early interrupt (ignored, interrupt not used)
);
// Configure the synchronous sampling for Channel-0
if(false == PLIB_ADCHS_ChannelTriggerSampleSelect( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0,                     // channel - 0
        ADCHS_CHANNEL_SYNC_SAMPLING))        // Synchronous sampling selected
        {
            // error has occurred
            while(1);
        }
// Select inputs for Channel
if(false == PLIB_ADCHS_ChannelInputSelect
            (
                MY_ADCHS_INSTANCE,
                ADCHS_CHANNEL_0,
                ADCHS_CHANNEL_0_DEFAULT_INP_AN0 // Select AN0 for channel-0
            ))
            {
                // error has occurred
                while(1);
            }
// Select input mode for AN0
PLIB_ADCHS_AnalogInputModeSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_AN0,
    ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
);
// Select AN0 as edge trigger
PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN0 );
// Select AN0 to be software triggered
PLIB_ADCHS_AnalogInputTriggerSourceSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_CLASS12_AN0,
    ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
);
// Configure digital filter in oversampling mode for 16x sampling
PLIB_ADCHS_DigitalFilterOversamplingModeSetup
(
    MY_ADCHS_INSTANCE, // ADCHS channel ID
    ADCHS_DIGITAL_FILTER_1, // Filter ID
    ADCHS_AN0, // Oversample AN4
```

```
        ADCHS_DIGITAL_FILTER_SIGNIFICANT_ALL_16BITS, // all 16bits significant
        ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_16X, // 16 x oversampling
        false // No Global Int Enable
    );
    // Enable digital filter
    PLIB_ADCHS_DigitalFilterEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
    // Enable ADC
    PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
    // Check VREF to be ready
    While(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));
    // Check for VREF Fault
    While(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE));
    // Enable analog circuit for channel-0
    PLIB_ADCHS_ChannelAnalogFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0
    );
    // Wait for the channels to be ready
    while(!PLIB_ADCHS_ChannelIsReady
        (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0
        )
    );
    // Enable digital circuit for channels 0
    PLIB_ADCHS_ChannelDigitalFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_0
    );
    while(1)
    {
        // Enable global software trigger
        PLIB_ADCHS_GlobalSoftwareTriggerEnable( MY_ADCHS_INSTANCE );
        // Wait for filter result to be ready
        while(!PLIB_ADCHS_DigitalFilterDataIsReady(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1));
        result = PLIB_ADCHS_DigitalFilterDataGet(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
    }
    return(1);
}
```

Example - Digital Comparator

The Digital Comparator is used to automatically evaluate results as they are output by the converter. The following example illustrates an automated test of AN8 for values that are greater than or equal to 80% of full scale or less than 20% of full scale. A count is incremented each time an event occurs.

```
{
    int result; // storage for results
    bool eventFlag = false; // Digital comparator event flag
    // Configure the ADC
    PLIB_ADCHS_Setup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_VREF_AVDD_AVSS,                    // AVDD and AVSS as reference
        ADCHS_CHARGEPUMP_DISABLE,                // No charge pump
        ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL,     // Use fractional format
        true,                                    // Do stop in idle
        ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE,     // Enable Fast synchronous system clock
        ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE, // Disable Fast synchronous peripheral clock
        ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS,   // vector shift not used since interrupt not used
        0,                                       // vector base address unused, interrupt not used
        ADCHS_CLOCK_SOURCE_SYSCLK,               // SYSCLK is the clock source
        1,                                       // TQ = 1/SYSCLK * 2
        ADCHS_WARMUP_CLOCK_32                    // Warm-up time = 32 clocks
    );
    // Configure the ADC SAR Channel-7
    PLIB_ADCHS_ChannelSetup
```

```
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_7,                        // Channel - 7
    ADCHS_DATA_RESOLUTION_12BIT,            // resolution is set to 12bits
    1,                                      // clock divider bit is, TAD7 = 2 * TQ
    1,                                      // Sample time is 3 * TAD7
    ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1     // 1 clock early interrupt (ignored, interrupt not used)
);
// Select input mode for AN8
PLIB_ADCHS_AnalogInputModeSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_AN8,
    ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR
);
// Configure the digital comparator - 1
PLIB_ADCHS_DigitalComparatorSetup
(
    MY_ADCHS_INSTANCE,          // ADCHS channel ID
    ADCHS_DIGITAL_COMPARATOR_1, // Comparator ID
    false,                      // Global Int Enable
    true,                       // test for between low and high
    false,                      // no test for greater than equal to high
    false,                      // no test for less than high
    false,                      // no test for greater than equal to low
    false,                      // no test for less than low
    ADCHS_AN8,                  // select AN8
    0xFFF - (0xFFF/5),          // high limit, 80% of full scale
    (0xFFF/5)                   // low limit, 20% of full scale
);
// Select individual trigger for class-2 channels viz. AN8 to be global software trigger.
PLIB_ADCHS_AnalogInputTriggerSourceSelect
(
    MY_ADCHS_INSTANCE,
    ADCHS_CLASS12_AN8,
    ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
);
// Select edge trigger
PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN8 );
// Enable the digital comparator
PLIB_ADCHS_DigitalComparatorEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_1);
// Enable ADC
PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
// Check VREF to be ready
While(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));
// Check for VREF Fault
While(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE));
// Enable analog circuit for channel-7
PLIB_ADCHS_ChannelAnalogFeatureEnable
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_7
);
// Wait for the channel-7 to be ready
while(!PLIB_ADCHS_ChannelIsReady
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7
    )
);
// Enable digital circuit for channels 7
PLIB_ADCHS_ChannelDigitalFeatureEnable
(
    MY_ADCHS_INSTANCE,
    ADCHS_CHANNEL_7
);
while(1)
{
    // Enable global software trigger
```

```
            PLIB_ADCHS_GlobalSoftwareTriggerEnable( MY_ADCHS_INSTANCE );
            // Wait for conversion complete for AN8
            while(!PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN8));
            result = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN8 );
            // Note: It is not necessary to read the conversion result for digital comparator
            // to work
            // See if we have comparator event
            if(PLIB_ADCHS_DigitalComparatorEventHasOccurred( MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_1 ))
            {
                eventFlag = true;
            }
        }
        return(1);
    }
```

Example - CVD Mode

The CVD mode is used to detect a touch event by measuring the voltage across external capacitor using the capacitive voltage divider method. The Digital Comparator used in conjunction with CVD mode automatically compares the voltage with set limits. Once the voltage is beyond the set limit, a comparator event is generated. In the following example, AN40 is used as an analog input. Physically, the AN40 pin should be connected to a touch sense pad. Once a touch is detected, a comparator event will be generated.

```
{
    int result; // storage for result
    // Configure the ADC
    PLIB_ADCHS_Setup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_VREF_AVDD_AVSS,                    // AVDD and AVSS as reference
        ADCHS_CHARGEPUMP_DISABLE,                // No charge pump
        ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL,     // Use fractional format
        true,                                    // Do stop in idle
        ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE,     // Enable Fast synchronous system clock
        ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE, // Disable Fast synchronous peripheral clock
        ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS,   // vector shift unused, interrupt not used
        0,                                       // vector base address unused, interrupt not used
        ADCHS_CLOCK_SOURCE_SYSCLK,               // SYSCLK is the clock source
        1,                                       // TQ = 1/SYSCLK * 2
        ADCHS_WARMUP_CLOCK_32                    // Warm-up time = 32 clocks
    );
    // Configure the ADC SAR Channel-7
    PLIB_ADCHS_ChannelSetup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7,                    // Channel - 7
        ADCHS_DATA_RESOLUTION_12BIT,        // resolution is set to 12bits
        1,                                  // clock divider bit is, TAD7 = 2 * TQ
        1,                                  // Sample time is 3 * TAD7
        ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1 // 1 clock early interrupt (ignored, interrupt not used)
    );
    // Select analog channels as bipolar and single ended
    // Since CVD measures difference in voltages, the setting should be bipolar
    PLIB_ADCHS_AnalogInputModeSelect
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN40,
        ADCHS_INPUT_MODE_SINGLE_ENDED_BIPOLAR
    );
    // Select AN40 for scanning, as CVD needs scanning to be enabled
    PLIB_ADCHS_AnalogInputScanSetup
    (
        MY_ADCHS_INSTANCE,
        ADCHS_AN40,
        ADCHS_SCAN_TRIGGER_SENSITIVE_EDGE,
        ADCHS_SCAN_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE
    );
    PLIB_ADCHS_CVDSetup
    (
        MY_ADCHS_INSTANCE,
```

```
        ADCHS_CVD_CAPACITOR_5PF,
        false,              // no test for between low and high
        true,               // Once touch event occurs, CVD output will
                            // be higher than the set limits
        false,              // no test for less than high
        false,              // no test for greater than equal to low
        false,              // no test for less than low
        ADCHS_AN40,         // select AN40
        0xFFF - (0xFFF/5), // high limit, 80% of full scale
        (0xFFF/5)           // low limit, 20% of full scale
    );
    // Enable the digital comparator-1, since SVD works with comparator-1
    PLIB_ADCHS_DigitalComparatorEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_1);
    // Enable CVD mode
    PLIB_ADCHS_CVDEnable(MY_ADCHS_INSTANCE);
    // Enable ADC
    PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
    // Check VREF to be ready
    While(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));
    // Check for VREF Fault
    While(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE));
    // Enable analog circuit for channel-7
    PLIB_ADCHS_ChannelAnalogFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7
    );
    // Wait for the channel-7 to be ready
    while(!PLIB_ADCHS_ChannelIsReady
        (
            MY_ADCHS_INSTANCE,
            ADCHS_CHANNEL_7
        )
    );
    // Enable digital circuit for channels 7
    PLIB_ADCHS_ChannelDigitalFeatureEnable
    (
        MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_7
    );
    while(1)
    {
        // Enable global software trigger
        PLIB_ADCHS_GlobalSoftwareTriggerEnable( MY_ADCHS_INSTANCE );
        // Wait, if there is no touch event sensed by CVD + comparator
        while(!PLIB_ADCHS_DigitalComparatorEventHasOccurred
            (
            MY_ADCHS_INSTANCE,
            ADCHS_DIGITAL_COMPARATOR_1
            )
        );
        // Verify of the CVD selected input is AN40
        // This is not required, but provided as an example
        if(ADCHS_AN40 == PLIB_ADCHS_DigitalComparatorAnalogInputGet( MY_ADCHS_INSTANCE,
        ADCHS_DIGITAL_COMPARATOR_1 ))
        {
            result == PLIB_ADCHS_CVDResultGet( MY_ADCHS_INSTANCE );
        }
    }
    return(1);
}
```

Other Functionality

This topic provides information on additional functionality.

Description

The ADCHS also implements the following additional functionalities:

- Digital Filter
- Digital Comparator
- Capacitive Voltage Divider (CVD) mode
- Turbo mode
- FIFO and DMA Interface for dedicated cores

Digital Filter

The Digital Filter consists of an accumulator and a decimator (downsampler), which function together as a low-pass filter. By sampling an analog input at a higher-than-required sample rate (oversampling), and then processing the data through the oversampling digital filter, the number of usable bits of the ADC channel can be increased at the expense of decreased conversion throughput. For example, using 4x oversampling yields one extra usable bit, 16x oversampling yields two extra usable bits, 64x oversampling provides three extra usable bits, and 256x oversampling provides four extra usable bits. Once configured, the application provides a single trigger to the analog input specified for oversampling, and then fetches the result when the process is complete.

Digital Comparator

The ADCHS module features multiple Digital Comparators that can be used to monitor selected analog input conversion results and generate an interrupt when a conversion result matches the user-specified limits. Conversion triggers are still required to initiate conversions. The comparison occurs automatically once the conversion is complete. When using a hardware source as a periodic trigger, it is possible to monitor analog inputs and create an interrupt when the converted level matches specified criteria without any software overhead.

CVD Mode

The CVD mode allows the ADCHS module to detect a touch event by measuring the voltage difference of internal and external capacitors by alternately connecting it to VDD and GND. The CVD mode works in conjunction with the Digital Comparator and can generate an event (interrupt) once the voltage difference is more than the set value (indicating a touch event).

Turbo Mode

The Turbo mode allows two Class 1 analog inputs to work in an interleaved manner to generate converted data at almost double the maximum throughput of a single Class 1 analog input.

FIFO and DMA Interfaces for Channels 0 through 6

The FIFO and DMA interfaces are applicable only for Channel 0 through 6. Using the FIFO, Channel 0 through 6 can save converted data into a FIFO. Using the DMA interface, Channel 0 through 6 can save the converted data directly into RAM.

Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Configuration Functions

	Name	Description
	PLIB_ADCHS_Disable	High-Speed SAR ADC module is disabled (turned OFF).
	PLIB_ADCHS_Enable	Enables the High-Speed SAR ADC module.
	PLIB_ADCHS_Setup	Configures the High-Speed SAR ADC converter.
	PLIB_ADCHS_ControlRegistersCanBeUpdated	Returns the status of update-ready.
	PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable	Disables the update-ready interrupt.
	PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable	Enables the update-ready interrupt.
	PLIB_ADCHS_ExternalConversionRequestDisable	Disables the external conversion request.
	PLIB_ADCHS_ExternalConversionRequestEnable	Enables the external conversion request.
	PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable	Disables the global level software trigger.
	PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable	Initiates a global level software trigger.
	PLIB_ADCHS_GlobalSoftwareTriggerEnable	Initiate a Global Software Trigger.
	PLIB_ADCHS_ScanCompleteInterruptDisable	Disables the end of scan interrupt.
	PLIB_ADCHS_ScanCompleteInterruptEnable	Enables the end of scan interrupt.
	PLIB_ADCHS_SoftwareConversionInputSelect	Selects the analog input of Channel 7 for manual conversion.
	PLIB_ADCHS_SoftwareConversionStart	Triggers the conversion of analog input signal connected to Channel 7.
	PLIB_ADCHS_SoftwareSamplingStart	Enables sampling of analog input for Channel 7.

	PLIB_ADCHS_SoftwareSamplingStop	Disables sampling of analog input for Channel 7, which places Channel 7 into Hold mode.
	PLIB_ADCHS_TriggerSuspendDisable	Disables trigger suspend.
	PLIB_ADCHS_TriggerSuspendEnable	Suspends all trigger for all ADCHS channels.

b) CVD Functions

	Name	Description
	PLIB_ADCHS_CVDDisable	Disables the CVD feature.
	PLIB_ADCHS_CVDEnable	Enables the CVD feature.
	PLIB_ADCHS_CVDResultGet	Returns a CVD result measured by an ADCHS instance.
	PLIB_ADCHS_CVDSetup	Configures the CVD related setting of ADCHS channel.

c) Analog Input Functions

	Name	Description
	PLIB_ADCHS_AnalogInputDataIsReady	Returns the data ready status of analog inputs.
	PLIB_ADCHS_AnalogInputDataReadyInterruptDisable	Disables the data ready interrupt for the selected analog inputs.
	PLIB_ADCHS_AnalogInputDataReadyInterruptEnable	Enables the data ready interrupt for the selected analog input.
	PLIB_ADCHS_AnalogInputEarlyInterruptDisable	Disables the early interrupt for the analog input.
	PLIB_ADCHS_AnalogInputEarlyInterruptEnable	Enables the early interrupt for the analog input.
	PLIB_ADCHS_AnalogInputEarlyInterruptIsReady	Returns the early interrupt ready status of analog input.
	PLIB_ADCHS_AnalogInputIsAvailable	Returns the analog input configuration of ADCHS channel.
	PLIB_ADCHS_AnalogInputModeGet	Returns the mode for the specified analog input.
	PLIB_ADCHS_AnalogInputResultGet	Returns a ADC conversion result.
	PLIB_ADCHS_AnalogInputScanIsComplete	Returns the state of End of scan completion.
	PLIB_ADCHS_AnalogInputScanIsSelected	Returns whether or note an analog input is selected for scanning.
	PLIB_ADCHS_AnalogInputScanRemove	Removes the analog input from scanning selection.

d) Analog Input Selection Functions

	Name	Description
	PLIB_ADCHS_AnalogInputModeSelect	Selects the mode for the specified analog input.
	PLIB_ADCHS_AnalogInputScanSelect	Selects the analog input for scanning.
	PLIB_ADCHS_ChannelTriggerSampleSelect	Selects the trigger and sampling modes for channels of High-Speed SAR ADC
	PLIB_ADCHS_AnalogInputEdgeTriggerSet	Sets the trigger as edge sensitive for selected class 1 and 2 analog input.
	PLIB_ADCHS_AnalogInputLevelTriggerSet	Sets the trigger as level sensitive for selected Class 1 and 2 analog input.
	PLIB_ADCHS_AnalogInputScanSetup	Selects input to include in Analog Input Scan mode.
	PLIB_ADCHS_AnalogInputTriggerSourceSelect	Selects a trigger Source for analog input (Class 1 or Class 2 analog inputs only).

e) Digital Comparator Functions

	Name	Description
	PLIB_ADCHS_DigitalComparatorAnalogInputGet	Returns the analog input ID used by the digital comparator.
	PLIB_ADCHS_DigitalComparatorAnalogInputSelect	Selects analog inputs, whose converted data will be processed by the comparator.
	PLIB_ADCHS_DigitalComparatorDisable	Disables the specified digital comparator.
	PLIB_ADCHS_DigitalComparatorEnable	Enables the specified digital comparator.
	PLIB_ADCHS_DigitalComparatorEventHasOccurred	Returns the status of the selected digital comparator.
	PLIB_ADCHS_DigitalComparatorInterruptDisable	Disables the interrupt for the selected digital comparator.
	PLIB_ADCHS_DigitalComparatorInterruptEnable	Enables the interrupt for the selected digital comparator.
	PLIB_ADCHS_DigitalComparatorLimitSet	Sets the limit for the specified digital comparator.
	PLIB_ADCHS_DigitalComparatorSetup	Configures the Digital Comparator on the High-Speed SAR ADC converter.

f) Digital Filter Functions

	Name	Description
	PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect	Selects the number of samples which are averaged by the Digital Filter.
	PLIB_ADCHS_DigitalFilterAveragingModeSetup	Configures the Digital Filter on the High-Speed SAR ADC converter in Averaging mode.
	PLIB_ADCHS_DigitalFilterDataGet	Used to fetch the data result from the Digital Filter.

	PLIB_ADCHS_DigitalFilterDataIsReady	Used to determine if the Digital Filter has data ready.
	PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable	Disables the interrupt for the selected Digital Filter.
	PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable	Enables the interrupt for the selected Digital Filter.
	PLIB_ADCHS_DigitalFilterDisable	Disables the Digital Filter.
	PLIB_ADCHS_DigitalFilterEnable	Enables the Digital Filter.
	PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect	Selects the oversampling ratio for the Digital Filter.
	PLIB_ADCHS_DigitalFilterOversamplingModeSetup	Configures the Digital Filter on the High-Speed SAR ADC converter in Oversampling mode.

g) DMA Functions

	Name	Description
	PLIB_ADCHS_DMABuffer_A_InterruptDisable	Disables the DMA Buffer A full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_A_InterruptEnable	Enables the DMA Buffer A full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_A_IsFull	Used to determine if the DMA Buffer A is full, for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_B_InterruptDisable	Disables the DMA Buffer B full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_B_InterruptEnable	Enables the DMA Buffer B full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_B_IsFull	Used to determine if the DMA Buffer B is full, for the specified Channel 0 to 6.
	PLIB_ADCHS_DMADisable	Disables the DMA in the High-Speed SAR ADC module.
	PLIB_ADCHS_DMAEnable	Enables the DMA in the High-Speed SAR ADC module.
	PLIB_ADCHS_DMAOverflowErrorHasOccurred	Used to determine if the DMA Buff had an overflow error.
	PLIB_ADCHS_DMASetup	Configures the DMA on the High-Speed SAR ADC.
	PLIB_ADCHS_DMASourceRemove	Disables the DMA for the specified Channel 0 to 6.
	PLIB_ADCHS_DMASourceSelect	Enables the DMA for the specified Channel 0 to 6.

h) Channel Related Functions

	Name	Description
	PLIB_ADCHS_ChannelAnalogFeatureDisable	Disables the analog circuit for channels of High-Speed SAR ADC.
	PLIB_ADCHS_ChannelAnalogFeatureEnable	Enables the analog circuit for High-Speed SAR ADC channels.
	PLIB_ADCHS_ChannelConfigurationGet	Used to get the configuration for the specified channel.
	PLIB_ADCHS_ChannelConfigurationSet	Used to set the configuration for the specified channel.
	PLIB_ADCHS_ChannelDigitalFeatureDisable	Disables the digital circuit for channels of High-Speed SAR ADC.
	PLIB_ADCHS_ChannelDigitalFeatureEnable	Enables (turns ON) the digital circuit for channels.
	PLIB_ADCHS_ChannelIsReady	Returns the state of the channel.
	PLIB_ADCHS_ChannelIsReadyInterruptDisable	Disables the Channel ready interrupt for the specified channel.
	PLIB_ADCHS_ChannelIsReadyInterruptEnable	Enables the Channel ready interrupt for the specified channel.
	PLIB_ADCHS_ChannelSetup	Configures the High-Speed SAR ADC channels.
	PLIB_ADCHS_ChannelInputSelect	Selects the analog input for Channel 0 to 6.

i) FIFO Functions

	Name	Description
	PLIB_ADCHS_FIFODataCountGet	Returns the number of data to be read from FIFO.
	PLIB_ADCHS_FIFODataIsAvailable	Used to determine if the FIFO has data ready.
	PLIB_ADCHS_FIFODataIsNegative	Returns the sign of data stored in FIFO.
	PLIB_ADCHS_FIFODataReadyInterruptDisable	Disables the interrupt for FIFO.
	PLIB_ADCHS_FIFODataReadyInterruptEnable	Enables the interrupt for FIFO.
	PLIB_ADCHS_FIFODisable	Disables the FIFO in the High-Speed SAR ADC.
	PLIB_ADCHS_FIFOEnable	Enables the FIFO in the High-Speed SAR ADC
	PLIB_ADCHS_FIFOErrorHasOccurred	Used to determine if the FIFO has encountered an overflow error.
	PLIB_ADCHS_FIFORead	Used to fetch the data result from the FIFO.
	PLIB_ADCHS_FIFOSourceGet	Returns the channel ID using the FIFO.
	PLIB_ADCHS_FIFOSourceSelect	Sets the Channel 0 to 6 using the FIFO.

j) Interrupt Functions

	Name	Description
	PLIB_ADCHS_EarlyInterruptDisable	Disables the early interrupt for the ADCHS.
	PLIB_ADCHS_EarlyInterruptEnable	Enables the early interrupt for the ADCHS.

k) Turbo Mode Functions

	Name	Description
	PLIB_ADCHS_TurboModeChannelSelect	Configures the channels for Turbo mode.
	PLIB_ADCHS_TurboModeDisable	Disables Turbo mode for High-Speed SAR ADC module.
	PLIB_ADCHS_TurboModeEnable	Enables Turbo mode for the High-Speed SAR ADC module.
	PLIB_ADCHS_TurboModeErrorHasOccurred	Returns the error state of Turbo mode.

l) Voltage Reference Functions

	Name	Description
	PLIB_ADCHS_VREFFaultHasOccurred	Returns the state of VREF fault.
	PLIB_ADCHS_VREFFaultInterruptDisable	Disables the VREF Fault interrupt.
	PLIB_ADCHS_VREFFaultInterruptEnable	Enables the VREF fault interrupt.
	PLIB_ADCHS_VREFIsReady	Returns the state of VREF.
	PLIB_ADCHS_VREFReadyInterruptDisable	Disables the VREF ready interrupt.
	PLIB_ADCHS_VREFReadyInterruptEnable	Enables the VREF ready interrupt.

m) Feature Existence Functions

	Name	Description
	PLIB_ADCHS_ExistsAnalogInputCheck	Identifies whether the System Configuration feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsAnalogInputModeControl	Identifies whether the analog input mode control exists on the ADCHS module.
	PLIB_ADCHS_ExistsAnalogInputScan	Identifies whether the Analog input Scan exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelAnalogControl	Identifies whether the Channel Analog control exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelConfiguration	Identifies whether the Channel Configuration feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelDigitalControl	Identifies whether the Channel Digital control exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelInputSelectControl	Identifies whether the Channel 0 to 6 Input select feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsConfiguration	Identifies whether the Configuration feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsConversionResults	Identifies whether the Conversion Results feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsCVD	Identifies whether the CVD exists on the ADCHS module.
	PLIB_ADCHS_ExistsDigitalComparator	Identifies whether the Digital Comparator feature exists on the ADCHS module .
	PLIB_ADCHS_ExistsDigitalFilter	Identifies whether the Digital Filter feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsDMA	Identifies whether the DMA exists on the ADCHS module.
	PLIB_ADCHS_ExistsEarlyInterruptControl	Identifies whether the Early Interrupt control exists on the ADCHS module.
	PLIB_ADCHS_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADCHS module
	PLIB_ADCHS_ExistsExternalConversionRequestControl	Identifies whether the External Convert feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsFIFO	Identifies whether the FIFO exists on the ADCHS module.
	PLIB_ADCHS_ExistsManualControl	Identifies whether the Manual control exists on the ADCHS module.
	PLIB_ADCHS_ExistsTriggerControl	Identifies whether the Trigger control exists on the ADCHS module.
	PLIB_ADCHS_ExistsTriggerSampleControl	Identifies whether the Trigger Sample control feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsTurboMode	Identifies whether the Turbo mode feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsUpdateReadyControl	Identifies whether the Update Ready feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsVREFControl	Identifies whether the VREF control exists on the ADCHS module.

n) Data Types and Constants

	Name	Description
	ADCHS_AN_INPUT_ID	Type for identifying the available ADC Inputs
	ADCHS_CHARGEPUMP_MODE	Defines the selection for the charge pump.
	ADCHS_CLOCK_SOURCE	Defines the ADCHS Clock Source Select.
	ADCHS_CVD_CAPACITOR	Defines the value of the internal capacitor during CVD mode.
	ADCHS_DATA_RESOLUTION	Identifies the resolution of the ADC output.
	ADCHS_DIGITAL_COMPARATOR_ID	Identifies the supported Digital Comparators.
	ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT	Identifies the Digital Filter averaging sample count.
	ADCHS_DIGITAL_FILTER_ID	Identifies the supported Digital Filters.
	ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO	Identifies the supported Digital Filter oversampling ratios.
	ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS	Data length of digital filter.
	ADCHS_DMA_BUFFER_LENGTH	Defines the length of the DMA buffer length.
	ADCHS_DMA_COUNT	Defines the enable/disable of the count feature for DMA.
	ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK	Defines the number of clocks prior to the arrival of valid data that the associated interrupt is generated.
	ADCHS_FAST_SYNC_PERIPHERAL_CLOCK	Defines the selection of the fast synchronous peripheral clock.
	ADCHS_FAST_SYNC_SYSTEM_CLOCK	Defines the selection of the fast synchronous system clock.
	ADCHS_INPUT_MODE	Defines the available modes for the selected input.
	ADCHS_INTERRUPT_BIT_SHIFT_LEFT	Identifies the bits shift for calculating the interrupt vector.
	ADCHS_OUTPUT_DATA_FORMAT	Defines the selection for the output data format.
	ADCHS_SCAN_TRIGGER_SENSITIVE	Trigger level for scan trigger.
	ADCHS_SCAN_TRIGGER_SOURCE	Defines the ADCHS Channel Scan Trigger Source Selections.
	ADCHS_TRIGGER_SOURCE	Defines the ADCHS Trigger Source Selections.
	ADCHS_VREF_SOURCE	Defines the ADCHS VREF Source Select.
	ADCHS_WARMUP_CLOCK	Identifies the number of clocks before the channel can be ready
	ADCHS_MODULE_ID	Identifies the number of ADC supported.
	ADCHS_CHANNEL_INP_SEL	Defines the alternate input selection for channels 0 to 6.
	ADCHS_CHANNEL_ID	Identifies the ADC channel from 0 to 7.
	ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL	Defines the selection of trigger and sampling for channels 0 to 6.
	ADCHS_CLASS12_AN_INPUT_ID	Type for identifying the available Class 1 and 2 analog Inputs.

Description

This section describes the Application Programming Interface (API) functions of the ADCHS Peripheral Library.

Refer to each section for a detailed description.

a) Configuration Functions

PLIB_ADCHS_Disable Function

High-Speed SAR ADC module is disabled (turned OFF).

File

plib_adchs.h

C

```
void PLIB_ADCHS_Disable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the selected High-Speed SAR ADC module.

Remarks

Not all functionality is available on all devices. Please refer to the specific device data sheet for the list of available features.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_Disable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_Disable(ADCHS_MODULE_ID index)

PLIB_ADCHS_Enable Function

Enables the High-Speed SAR ADC module.

File

plib_adchs.h

C

```
void PLIB_ADCHS_Enable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the selected High-Speed SAR ADC module. There are many ADCHS functionalities which should be set/ selected before the ADCHS module is turned ON. Once the ADCHS is turned ON, the application code should check if the VREF is ready and without any fault. Subsequently, the analog bias circuitry for the required channel should be enabled followed by enabling the digital section.

Remarks

None.

Preconditions

All channels and analog input related selections should be made before enabling the ADCHS module.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_Enable(ADCHS_MODULE_ID index)

PLIB_ADCHS_Setup Function

Configures the High-Speed SAR ADC converter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_Setup(ADCHS_MODULE_ID index, ADCHS_VREF_SOURCE voltageRefSelect, ADCHS_CHARGEPUMP_MODE
chargePump, ADCHS_OUTPUT_DATA_FORMAT outputFormat, bool stopInIdle, ADCHS_FAST_SYNC_SYSTEM_CLOCK sysClk,
```

```
ADCHS_FAST_SYNC_PERIPHERAL_CLOCK periClk, ADCHS_INTERRUPT_BIT_SHIFT_LEFT intVectorShift, uint16_t
intVectorBase, ADCHS_CLOCK_SOURCE adcClockSource, int8_t adcClockDivider, ADCHS_WARMUP_CLOCK warmUpClock);
```

Returns

None.

Description

Configures all ADC parameters which are common to all ADC channels (from 0 to 7). This configuration must occur prior to enabling the ADC and therefore must be called when the ADC is disabled.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The ADCHS module is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the ADC
PLIB_ADCHS_Setup
(
    MY_ADCHS_INSTANCE,
    ADCHS_VREF_AVDD_AVSS,                   // AVDD and AVSS as reference
    ADCHS_CHARGEPUMP_DISABLE,               // Charge pump is disabled
    ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL,    // Use fractional format
    true,                                   // Stop in idle
    ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE,    // Enable Fast synchronous system clock
    ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE, // Disable Fast synchronous peripheral clock
    ADCHS_INTERRUPT_BIT_SHIFT_LEFT_2_BITS,  // vector left shift set to 2
    0x1600,                                 // vector base address set to 0x1600
    ADCHS_CLOCK_SOURCE_SYSCLK,              // SYSCLK is the clock source
    1,                                      // Control clock, TQ = 1/SYSCLK * 2
    ADCHS_WARMUP_CLOCK_32                   // Warm-up up time = 32 clocks
);

// Enable the ADC
PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
voltageRefSelect	VREF Source Selection.
chargePump	Enables/ disables the charge pump. This variable is of ADCHS_CHARGEPUMP_MODE type. Use when
VREFH	VREFL < .64 (AVDD - AVSS). This feature is not available on all devices.
outputFormat	Sets output data format to be integer or fractional. This parameter is of ADCHS_OUTPUT_DATA_FORMAT type.
stopInIdle	Sets ADC to stop when device is in Idle mode if true
sysClk	Sets the Fast synchronous system clock to ADC control clock. This variable is of type ADCHS_FAST_SYNC_SYSTEM_CLOCK.
periClk	Sets the Fast synchronous peripheral clock to ADC control clock. This variable is of type ADCHS_FAST_SYNC_PERIPHERAL_CLOCK.
intVectorShift	Sets the interrupt vector shift left shift. This variable is of ADCHS_INTERRUPT_BIT_SHIFT_LEFT type.
intVectorBase	Sets the interrupt vector base address.
adcClockSource	Clock source selection. This variable is of type ADCHS_CLOCK_SOURCE.
adcClockDivider	Clock source divider. Values range from 0 to 63. This divider determines the input clock frequency which goes as input to all ADC channels (all ADC channels, 0 to 7). This clock is also known as the "control clock" of ADC. The frequency of control
clock for ADC, as per the following equation	Control clock frequency = (frequency of "adcClockSource")/(adcClockDivider * 2) For adcClockDivider = 0, control clock frequency is same as frequency of "adcClockSource".

warmUpClock	This parameter determines the number of channel clock (not control clock) which is required as warm up time for the ADC channel, once the analog feature of that ADC channel is enabled. The variable is of type ADCHS_WARMUP_CLOCK.

Function

```
void PLIB_ADCHS_Setup
(
    ADCHS_MODULE_ID index,
    ADCHS_VREF_SOURCE voltageRefSelect,    // voltage reference
    ADCHS_CHARGEPUMP_MODE chargePump,      // Charge pump enabled/disabled
    ADCHS_OUTPUT_DATA_FORMAT outputFormat,  // result format fractional
bool stopInIdle,                   // stop in idle
    ADCHS_FAST_SYNC_SYSTEM_CLOCK sysClk,   // Fast synchronous system clock
// to ADC control clock
    ADCHS_FAST_SYNC_PERIPHERAL_CLOCK periClk,      // Fast synchronous peripheral
// clock to ADC control clock
    ADCHS_INTERRUPT_BIT_SHIFT_LEFT intVectorShift,  // Interrupt vector shift bits
uint16_t intVectorBase,            // Interrupt vector base address
    ADCHS_CLOCK_SOURCE adcClockSource,     // clock source
int8_t adcClockDivider,            // clock divider
    ADCHS_WARMUP_CLOCK warmUpClock         // Analog channel warm-up clocks.
)
```

PLIB_ADCHS_ControlRegistersCanBeUpdated Function

Returns the status of update-ready.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ControlRegistersCanBeUpdated(ADCHS_MODULE_ID index);
```

Returns

- true - The ADCHS module can be safely updated (settings can be configured)
- false - The ADCHS module is not yet ready to be safely updated

Description

Once the triggers are suspended, the ADCHS raises an update-ready flag indicating that the updates to ADCHS registers can be performed. This function returns the status of update ready.

Remarks

None.

Preconditions

None.

Example

```
// Suspend trigger
PLIB_ADCHS_TriggerSuspendEnable(MY_ADCHS_INSTANCE);

// Wait until the channel registers can be updated
while(!PLIB_ADCHS_ControlRegistersCanBeUpdated(MY_ADCHS_INSTANCE));

// Once the function returns true, update the configurations of ADCHS
//....

// Enable triggers
PLIB_ADCHS_TriggerSuspendDisable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

bool PLIB_ADCHS_ControlRegistersCanBeUpdated(ADCHS_MODULE_ID index)

PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable Function

Disables the update-ready interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables the update-ready interrupt.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable Function

Enables the update-ready interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

Once the triggers are suspended, the ADC channel raises an update ready flag indicating that the updates to ADCHS registers can be performed. This function enables the interrupt which occurs once the flag is raised.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExternalConversionRequestDisable Function

Disables the external conversion request.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ExternalConversionRequestDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the external conversion request. Once disabled, the external sources such as PTG cannot request a conversion.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable external conversion request
PLIB_ADCHS_ExternalConversionRequestDisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_ExternalConversionRequestDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExternalConversionRequestEnable Function

Enables the external conversion request.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ExternalConversionRequestEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the external conversion request. Once enabled, the ADCHS can accept conversion request from external sources

such as the PTG module.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable external conversion request
PLIB_ADCHS_ExternalConversionRequestEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_ExternalConversionRequestEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable Function

Disables the global level software trigger.

File

plib_adchs.h

C

```
void PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables the global level software trigger on the specified channel.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable Function

Initiates a global level software trigger.

File

plib_adchs.h

C

```
void PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function initiates a global level software trigger. All inputs or the scan list configured to trigger on the global level software trigger are triggered. Once initiated, the trigger continues until the global level trigger is disabled using the function PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_GlobalSoftwareTriggerEnable Function

Initiate a Global Software Trigger.

File

plib_adchs.h

C

```
void PLIB_ADCHS_GlobalSoftwareTriggerEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

All inputs or scan list that is configured to trigger on the global software trigger are triggered. Once enabled, the trigger automatically gets disabled (edge sensitive).

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_GlobalSoftwareTriggerEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_GlobalSoftwareTriggerEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_ScanCompleteInterruptDisable Function

Disables the end of scan interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ScanCompleteInterruptDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the end of scan interrupt.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable EOS interrupt
PLIB_ADCHS_ScanCompleteInterruptDisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_ScanCompleteInterruptDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_ScanCompleteInterruptEnable Function

Enables the end of scan interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ScanCompleteInterruptEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the end of scan interrupt. The interrupt is generated when scanning of all of the selected analog inputs has completed.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
```

```
// application developer.

// Enable EOS interrupt
PLIB_ADCHS_ScanCompleteInterruptEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_ScanCompleteInterruptEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_SoftwareConversionInputSelect Function

Selects the analog input of Channel 7 for manual conversion.

File

plib_adchs.h

C

```
void PLIB_ADCHS_SoftwareConversionInputSelect(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID anInput);
```

Returns

None.

Description

This function selects the analog input for Channel 7, which is used for manual conversion (using software control).

Remarks

None.

Preconditions

None.

Example

```
// Select AN24 for manual conversion
PLIB_ADCHS_SoftwareConversionInputSelect(MY_ADCHS_INSTANCE, ADCHS_AN24);

// Place S&H into Sampling mode
PLIB_ADCHS_SoftwareSamplingStart(MY_ADCHS_INSTANCE);

// Wait for the required sampling time
int del;
for(del = 0; del<=20; del++);

// Now, make the S&H into Hold mode
PLIB_ADCHS_SoftwareSamplingStop(MY_ADCHS_INSTANCE);

// Now, start conversion
PLIB_ADCHS_SoftwareConversionStart(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
anInput	Analog input of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_SoftwareConversionInputSelect

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID anInput

)

PLIB_ADCHS_SoftwareConversionStart Function

Triggers the conversion of analog input signal connected to Channel 7.

File

plib_adchs.h

C

```
void PLIB_ADCHS_SoftwareConversionStart(ADCHS_MODULE_ID index);
```

Returns

None.

Description

After a signal is held on the S&H of Channel 7, the conversion can be triggered manually using software by using this function.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_SoftwareConversionStart(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_SoftwareConversionStart(ADCHS_MODULE_ID index)

PLIB_ADCHS_SoftwareSamplingStart Function

Enables sampling of analog input for Channel 7.

File

plib_adchs.h

C

```
void PLIB_ADCHS_SoftwareSamplingStart(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function can be used if Channel 7 needs to be brought to Sampling mode manually (i.e., through software).

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_SoftwareSamplingStart(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_SoftwareSamplingStart(ADCHS_MODULE_ID index)

PLIB_ADCHS_SoftwareSamplingStop Function

Disables sampling of analog input for Channel 7, which places Channel 7 into Hold mode.

File

plib_adchs.h

C

```
void PLIB_ADCHS_SoftwareSamplingStop(ADCHS_MODULE_ID index);
```

Returns

None.

Description

After sampling for a significant time (using the function PLIB_ADCHS_SoftwareSamplingStart), the sampling can be disabled, which will bring the channel to Hold mode.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_SoftwareSamplingStop(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_SoftwareSamplingStop(ADCHS_MODULE_ID index)

PLIB_ADCHS_TriggerSuspendDisable Function

Disables trigger suspend.

File

plib_adchs.h

C

```
void PLIB_ADCHS_TriggerSuspendDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

By calling this function, all triggers are allowed to reach the channel and triggers are not suspended.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_TriggerSuspendDisable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_TriggerSuspendDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_TriggerSuspendEnable Function

Suspends all trigger for all ADCHS channels.

File

plib_adchs.h

C

```
void PLIB_ADCHS_TriggerSuspendEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function suspends all triggers for all channels on the ADCHS module. The trigger needs to be suspended so that the channels could be configured without disabling (and subsequently enabling) the channels.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_TriggerSuspendEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_TriggerSuspendEnable(ADCHS_MODULE_ID index)

b) CVD Functions

PLIB_ADCHS_CVDDisable Function

Disables the CVD feature.

File

plib_adchs.h

C

```
void PLIB_ADCHS_CVDDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables the CVD feature of an ADCHS instance.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_CVDDisable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_CVDDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_CVDEnable Function

Enables the CVD feature.

File

plib_adchs.h

C

```
void PLIB_ADCHS_CVDEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

ADCHS channel supports the Capacitive Voltage Divider (CVD) mode. This function enables the CVD feature of an ADCHS instance.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCHS_CVDEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_CVDEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_CVDResultGet Function

Returns a CVD result measured by an ADCHS instance.

File

plib_adchs.h

C

```
int16_t PLIB_ADCHS_CVDResultGet(ADCHS_MODULE_ID index);
```

Returns

The conversion result expressed as a signed 16-bit integer.

Description

During CVD measurement, the difference between the voltage measured during positive and negative phases is calculated. This function return the difference value.

Remarks

None.

Preconditions

CVD must be enabled and a comparator event linked to CVD should have occurred.

Example

```
int16_t     result;
...
// fetch result for CVD
result = PLIB_ADCHS_CVDResultGet( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

int16_t PLIB_ADCHS_CVDResultGet(ADCHS_MODULE_ID index)

PLIB_ADCHS_CVDSetup Function

Configures the CVD related setting of ADCHS channel.

File

plib_adchs.h

C

```
void PLIB_ADCHS_CVDSetup(ADCHS_MODULE_ID index, ADCHS_CVD_CAPACITOR capSel, bool inBetweenOrEqual, bool
greaterEqualHi, bool lessThanHi, bool greaterEqualLo, bool lessThanLo, ADCHS_AN_INPUT_ID inputEnable,
int16_t hiLimit, int16_t loLimit);
```

Returns

None.

Description

While selecting the CVD mode, the internal capacitor should be similar in magnitude to the external capacitor being sensed. This function configures the internal capacitance. Additionally, the functions sets the limits for CVD measurement and the condition to detect a touch event. Since, CVD is implemented with digital comparator 0, using the CVD functionality also needs that digital comparator is enabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the CVD mode
// Creates an event when the reading of AN3 is between 20% to 80% of the
// full scale 12-bit output. This will be the touch event.

PLIB_ADCHS_CVDSetup(    MY_ADCHS_INSTANCE,
                          ADCHS_CVD_CAPACITOR_5PF,
                          false,            // no test for between low and high
                          true,             // Once touch event occurs, CVD output will
                                            // be higher than the set limits
                          false,            // no test for less than high
```

```
                                    false,          // no test for greater than equal to low
                                    false,          // no test for less than low
                                    ADCHS_AN3,      // enable AN3
                                    0x0CCD,         // high limit, 80% of full scale
                                    0x0333);        // low limit, 20% of full scale
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
capSel	Selection of value of input capacitor of type ADCHS_CVD_CAPACITOR.
inBetweenOrEqual	Event is generated when result is greater than or equal to loLimit and less than hiLimit.
greaterEqualHi	Event is generated when result is greater than or equal to hiLimit.
lessThanHi	Event is generated when result is less than hiLimit.
greaterEqualLo	Event is generated when result is greater than or equal to loLimit.
lessThanLo	Event is generated when result is less than loLimit.
inputEnable	Bit field which determines which analog inputs are tested by this comparator. This parameter is of type ADCHS_AN_INPUT_ID.
hiLimit	High limit in the same format as the conversion result.
loLimit	Low limit in the same format as the conversion result.

Function

```
void PLIB_ADCHS_CVDSetup
(
    ADCHS_MODULE_ID index,
    ADCHS_CVD_CAPACITOR capSel,     // Input capacitor
bool inBetweenOrEqual,          // between low and high
bool greaterEqualHi,            // greater than equal to high
bool lessThanHi,                // less than high
bool greaterEqualLo,            // greater than equal to low
bool lessThanLo,                // less than low
    ADCHS_AN_INPUT_ID inputEnable,  // input enable bit
int16_t  hiLimit,               // high limit
int16_t  loLimit                // low limit
)
```

c) Analog Input Functions

PLIB_ADCHS_AnalogInputDataIsReady Function

Returns the data ready status of analog inputs.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_AnalogInputDataIsReady(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

- true - Converted data is ready for selected analog input
- false - Conversion is not yet complete and data is not ready

Description

This function returns if the converted data is ready for the analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Check if data is ready for AN10
if(PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN10) == true)
{
    // Do further processing
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

bool PLIB_ADCHS_AnalogInputDataIsReady
(
    ADCHS_MODULE_ID index,
    ADCHS_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputDataReadyInterruptDisable Function

Disables the data ready interrupt for the selected analog inputs.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputDataReadyInterruptDisable(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function disables (turns OFF) the data ready interrupt for the selected analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable interrupt for AN10 and AN11
PLIB_ADCHS_AnalogInputDataReadyInterruptDisable(MY_ADCHS_INSTANCE, ADCHS_AN10);
PLIB_ADCHS_AnalogInputDataReadyInterruptDisable(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_AnalogInputDataReadyInterruptDisable
(
ADCHS_MODULE_ID index,
ADCHS_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputDataReadyInterruptEnable Function

Enables the data ready interrupt for the selected analog input.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputDataReadyInterruptEnable(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function enables (turns ON) the data ready interrupt for the selected analog input. The interrupt is generated when the converted data is ready for the selected analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable interrupt for AN10 and AN11
PLIB_ADCHS_AnalogInputDataReadyInterruptEnable(MY_ADCHS_INSTANCE, ADCHS_AN10);
PLIB_ADCHS_AnalogInputDataReadyInterruptEnable(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_AnalogInputDataReadyInterruptEnable
(
ADCHS_MODULE_ID index,
ADCHS_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputEarlyInterruptDisable Function

Disables the early interrupt for the analog input.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputEarlyInterruptDisable(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function disables (turns OFF) the early interrupt for the selected analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable interrupt for AN10 and AN11
PLIB_ADCHS_AnalogInputEarlyInterruptDisable(MY_ADCHS_INSTANCE, ADCHS_AN10);
PLIB_ADCHS_AnalogInputEarlyInterruptDisable(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_AnalogInputEarlyInterruptDisable

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID analogInput

)

PLIB_ADCHS_AnalogInputEarlyInterruptEnable Function

Enables the early interrupt for the analog input.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputEarlyInterruptEnable(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function enables (turns ON) the early interrupt for the selected analog input. The interrupt is generated at a set number of clock prior to conversion complete. The number of clock for early interrupt is set using the "configure" function for a specific channel.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable interrupt for AN10 and AN11
PLIB_ADCHS_AnalogInputEarlyInterruptEnable(MY_ADCHS_INSTANCE, ADCHS_AN10);
```

```
PLIB_ADCHS_AnalogInputEarlyInterruptEnable(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_AnalogInputEarlyInterruptEnable
(
ADCHS_MODULE_ID index,
ADCHS_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputEarlyInterruptIsReady Function

Returns the early interrupt ready status of analog input.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_AnalogInputEarlyInterruptIsReady(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

- true - Early interrupt event is ready
- false - Early interrupt event has not occurred

Description

This function returns the status of early interrupt for the selected analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.


// Check if early interrupt is ready for AN10
if(PLIB_ADCHS_AnalogInputEarlyInterruptIsReady(MY_ADCHS_INSTANCE, ADCHS_AN10))
{
    // Do further processing
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

bool PLIB_ADCHS_AnalogInputEarlyInterruptIsReady
(
ADCHS_MODULE_ID index,
ADCHS_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputIsAvailable Function

Returns the analog input configuration of ADCHS channel.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_AnalogInputIsAvailable(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

- true - Selected analog input is available.
- false - Selected analog input is not available in the ADCHS channel.

Description

This function returns the if the selected analog input is available in the ADCHS channel.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.


// Check if AN10 is available in the ADCHS channel
if(PLIB_ADCHS_AnalogInputIsAvailable(MY_ADCHS_INSTANCE, ADCHS_AN10))
{
    // do further processing
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

bool PLIB_ADCHS_AnalogInputIsAvailable

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID analogInput

)

PLIB_ADCHS_AnalogInputModeGet Function

Returns the mode for the specified analog input.

File

plib_adchs.h

C

```
ADCHS_INPUT_MODE PLIB_ADCHS_AnalogInputModeGet(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

Mode of the selected analog input of type ADCHS_INPUT_MODE type.

Description

This function returns the mode (single ended or differential) and encoding (unipolar or two's compliment) for the specified analog input.

When getting the mode of analog input which is meant to be alternate input for an ADC channel, the default analog input ID should be passed to this function.

For example, for CHANNEL_0, the default analog input is AN0 and alternate analog input is AN45. If the mode for AN45 needs to be obtained, the default input ID (i.e., AN0) should be passed.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Declare a variable of type "ADCHS_INPUT_MODE"
ADCHS_INPUT_MODE mode;

// Get mode for AN10
mode = PLIB_ADCHS_AnalogInputModeGet(MY_ADCHS_INSTANCE, ADCHS_AN10);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID

Function

```
 ADCHS_INPUT_MODE PLIB_ADCHS_AnalogInputModeGet
(
    ADCHS_MODULE_ID index,
    ADCHS_AN_INPUT_ID analogInput
)
```

PLIB_ADCHS_AnalogInputResultGet Function

Returns a ADC conversion result.

File

plib_adchs.h

C

```
uint32_t PLIB_ADCHS_AnalogInputResultGet(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

The conversion result expressed as a 32-bit integer.

Description

This function returns the converted result for the specified analog input.

Remarks

None.

Preconditions

A valid conversion is ready to be fetched.

Example

```
uint32_t        result;
//...
```

```
// Check if data is ready for AN10
if(PLIB_ADCHS_AnalogInputDataIsReady(MY_ADCHS_INSTANCE, ADCHS_AN10))
{
    result = PLIB_ADCHS_AnalogInputResultGet( MY_ADCHS_INSTANCE, ADCHS_AN10 );
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

uint32_t PLIB_ADCHS_AnalogInputResultGet
(
ADCHS_MODULE_ID index,
ADCHS_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputScanIsComplete Function

Returns the state of End of scan completion.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_AnalogInputScanIsComplete(ADCHS_MODULE_ID index);
```

Returns

- true - Analog input scanning is complete for the selected analog inputs
- false - Analog input scanning is not complete

Description

This function returns 'true', if all of the selected Analog Inputs have completed scanning.

Remarks

None.

Preconditions

The ADCHS module should have Analog Inputs Scanning mode selected and enabled for scanning to occur.

Example

```
bool EOSState = PLIB_ADCHS_AnalogInputScanIsComplete(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance

Function

bool PLIB_ADCHS_AnalogInputScanIsComplete(ADCHS_MODULE_ID index)

PLIB_ADCHS_AnalogInputScanIsSelected Function

Returns whether or note an analog input is selected for scanning.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_AnalogInputScanIsSelected(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

- true - Analog input is in selected for scanning
- false - Analog input is not selected for scanning

Description

This function returns whether or not an analog input is included in the scanning queue.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Determine if AN10 is selected for scanning or not
if(PLIB_ADCHS_AnalogInputScanIsSelected(MY_ADCHS_INSTANCE, ADCHS_AN10)  == true)
{
    // Perform further operations
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

bool PLIB_ADCHS_AnalogInputScanIsSelected

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID analogInput

)

PLIB_ADCHS_AnalogInputScanRemove Function

Removes the analog input from scanning selection.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputScanRemove(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function removes the analog input for scanning queue. Therefore, the analog input is no longer used for scanning.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Remove AN10 and AN11 from scanning
PLIB_ADCHS_AnalogInputScanRemove(MY_ADCHS_INSTANCE, ADCHS_AN10);
PLIB_ADCHS_AnalogInputScanRemove(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_AnalogInputScanRemove

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID analogInput

)

d) Analog Input Selection Functions

PLIB_ADCHS_AnalogInputModeSelect Function

Selects the mode for the specified analog input.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputModeSelect(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput,
ADCHS_INPUT_MODE mode);
```

Returns

None.

Description

This function selects the mode (single ended or differential) and encoding (unipolar or two's compliment) for the specified analog input.

When selecting the mode of analog input which is meant to be alternate input for an ADC channel, the default analog input ID should be passed to this function.

For example, for CHANNEL_0, the default analog input is AN0 and the alternate analog input is AN45. If the mode for AN45 needs to be changed, the default input ID (i.e., AN0) should be passed.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Set AN10 and AN11 to use the single ended, two's complement mode
// selection.
PLIB_ADCHS_AnalogInputModeSelect(MY_ADCHS_INSTANCE,
                        ADCHS_AN10,
                        ADCHS_INPUT_MODE_SINGLE_ENDED_TWOS_COMP);

PLIB_ADCHS_AnalogInputModeSelect(MY_ADCHS_INSTANCE,
```

```
                    ADCHS_AN11,
                    ADCHS_INPUT_MODE_SINGLE_ENDED_TWOS_COMP);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance
analogInput	Analog input ID for which the input mode is to be selected. It is of type ADCHS_AN_INPUT_ID.
mode	An ADCHS_INPUT_MODE type indicating the mode selection

Function

void PLIB_ADCHS_AnalogInputModeSelect

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID analogInput,

ADCHS_INPUT_MODE mode

)

PLIB_ADCHS_AnalogInputScanSelect Function

Selects the analog input for scanning.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputScanSelect(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function selects the analog input for scanning.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Select for AN10 and AN11 for scanning
PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN10);
PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.

Function

void PLIB_ADCHS_AnalogInputScanSelect

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID analogInput

)

PLIB_ADCHS_ChannelTriggerSampleSelect Function

Selects the trigger and sampling modes for channels of High-Speed SAR ADC

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ChannelTriggerSampleSelect(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID,
ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL sampSel);
```

Returns

- true - The function successfully selected the sampling and trigger mode
- false - The function could not select the sampling and trigger mode because Channel 7 was selected and Channel 7 does not implement this feature

Description

The channels from 0 through 6 have features to select the trigger as presynchronized or unsynchronized. Also, the sampling can be selected as synchronous. This function provides the functionality to select the trigger and sampling for ADC channels. The valid channels for selecting trigger and sampling is 0 to 6. Channel 7 does not have the feature. Therefore, calling this function for Channel 7 will result in failure. This selection must occur prior to enabling the ADC and therefore must be called when the ADC is disabled. Also, this selection must occur prior to enabling the digital circuit and analog bias circuit for the specified ADC channel.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The channel is disabled when calling this function.

Example

```
bool status = false;

// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the ADC
PLIB_ADCHS_ChannelSetup( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_1,                        // channel - 1
        ADCHS_DATA_RESOLUTION_12BIT,            // resolution is set to 12bits
        1,                                      // channel clock divider bit is, TAD =  2 * TQ
        1,                                      // Sample time is 3 * TAD
        ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1);   // Interrupt, 1 clock early

status = PLIB_ADCHS_ChannelTriggerSampleSelect( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_1,                        // channel - 1
        ADCHS_CHANNEL_SYNC_SAMPLING);           // Synchronous sampling selected

if( false == status)
{
    // error has occurred
    while(1);
}

// Enable the ADC
PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);

// Wait until the reference voltage is ready
while(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));

//Check if there is a fault in reference voltage
if(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE))
{
    // process fault accordingly
    while(1);
}
```

```
// Enable the analog circuit
PLIB_ADCHS_ChannelAnalogFeatureEnable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);

// Wait until the channel wakes up after warm-up
while(!PLIB_ADCHS_ChannelIsReady(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1));

// Enable the digital circuit
PLIB_ADCHS_ChannelDigitalFeatureEnable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);

// ADC will begin conversion now, after a valid trigger
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID from 0 to 6.
sampSel	Sets the presync trigger or sync sampling options. This variable is of type ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL.

Function

bool PLIB_ADCHS_ChannelTriggerSampleSelect

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID,

ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL sampSel

)

PLIB_ADCHS_AnalogInputEdgeTriggerSet Function

Sets the trigger as edge sensitive for selected class 1 and 2 analog input.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputEdgeTriggerSet(ADCHS_MODULE_ID index, ADCHS_CLASS12_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function sets the trigger as edge sensitive for selected Class 1 and 2 analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Set for AN10 and AN11 for edge trigger
PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN10 );
PLIB_ADCHS_AnalogInputEdgeTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN11 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

analogInput	An analog input selection of type ADCHS_AN_INPUT_ID. Since the trigger level can be selected only for Class 1 and 2 analog inputs.

Function

void PLIB_ADCHS_AnalogInputEdgeTriggerSet
(
ADCHS_MODULE_ID index,
ADCHS_CLASS12_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputLevelTriggerSet Function

Sets the trigger as level sensitive for selected Class 1 and 2 analog input.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputLevelTriggerSet(ADCHS_MODULE_ID index, ADCHS_CLASS12_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function sets the trigger as level sensitive for selected Class 1 and 2 analog input.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Set for AN5 and AN6 for level trigger
PLIB_ADCHS_AnalogInputLevelTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN5 );
PLIB_ADCHS_AnalogInputLevelTriggerSet( MY_ADCHS_INSTANCE, ADCHS_CLASS12_AN6 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID. Since the trigger level can be selected only for Class 1 and 2 analog inputs.

Function

void PLIB_ADCHS_AnalogInputLevelTriggerSet
(
ADCHS_MODULE_ID index,
ADCHS_CLASS12_AN_INPUT_ID analogInput
)

PLIB_ADCHS_AnalogInputScanSetup Function

Selects input to include in Analog Input Scan mode.

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputScanSetup(ADCHS_MODULE_ID index, ADCHS_AN_INPUT_ID analogInput,
ADCHS_SCAN_TRIGGER_SENSITIVE trgSensitive, ADCHS_SCAN_TRIGGER_SOURCE triggerSource);
```

Returns

None.

Description

Selects inputs, as determined by the low and high enable scan list for inclusion in the analog input scan sequence. If the input is a Class 1 or Class 2 type, it will also select the trigger source for that input to be the scan trigger, which is required if included in analog input scanning. This function configures the scan functionality with a single ADC input. If more inputs are required to be added for scanning, the PLIB_ADCHS_AnalogInputScanSelect function should be used.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure analog input Scanning
// Analog inputs 10 through 13
// Trigger on Timer 1 match.

PLIB_ADCHS_AnalogInputScanSetup(MY_ADCHS_INSTANCE,
                                ADCHS_AN10,
                                ADCHS_SCAN_TRIGGER_SENSITIVE_EDGE,   // Edge sensitive
                                ADCHS_SCAN_TRIGGER_SOURCE_TMR1_MATCH);

PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN11);
PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN12);
PLIB_ADCHS_AnalogInputScanSelect(MY_ADCHS_INSTANCE, ADCHS_AN13);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID.
trgSensitive	Enables level/edge sensitive scan trigger.
triggerSource	Trigger source used to initiate analog input scan.

Function

void PLIB_ADCHS_AnalogInputScanSetup

(

ADCHS_MODULE_ID index,

ADCHS_AN_INPUT_ID analogInput,

ADCHS_SCAN_TRIGGER_SENSITIVE trgSensitive,

ADCHS_SCAN_TRIGGER_SOURCE triggerSource

)

PLIB_ADCHS_AnalogInputTriggerSourceSelect Function

Selects a trigger Source for analog input (Class 1 or Class 2 analog inputs only).

File

plib_adchs.h

C

```
void PLIB_ADCHS_AnalogInputTriggerSourceSelect(ADCHS_MODULE_ID index, ADCHS_CLASS12_AN_INPUT_ID inputId,
ADCHS_TRIGGER_SOURCE triggerSource);
```

Returns

None.

Description

This function selects the trigger source for Class 1 or Class 2 inputs. A call to this function is not required when the input is being used as part of an analog input scan, as the input scan configure function also configures all trigger sources.

Remarks

None.

Preconditions

The function only applies to Class 1 and Class 2 inputs.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure AN4 for triggering from INT0.
PLIB_ADCHS_AnalogInputTriggerSourceSelect( MY_ADCHS_INSTANCE,
                                   ADCHS_CLASS12_AN4,
                                   ADCHS_TRIGGER_SOURCE_INT0 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
inputId	Class 1 or Class 2 input trigger to be configured.
triggerSource	Trigger source to use for this input of type ADCHS_TRIGGER_SOURCE.

Function

void PLIB_ADCHS_AnalogInputTriggerSourceSelect

(

ADCHS_MODULE_ID index,

ADCHS_CLASS12_AN_INPUT_ID inputId,

ADCHS_TRIGGER_SOURCE triggerSource

)

e) Digital Comparator Functions

PLIB_ADCHS_DigitalComparatorAnalogInputGet Function

Returns the analog input ID used by the digital comparator.

File

plib_adchs.h

C

```
ADCHS_AN_INPUT_ID PLIB_ADCHS_DigitalComparatorAnalogInputGet(ADCHS_MODULE_ID index,
ADCHS_DIGITAL_COMPARATOR_ID digComparator);
```

Returns

An analog input ID of type ADCHS_AN_INPUT_ID.

Description

This function returns the analog input ID, whose converted data is being compared by the specified digital comparator. In a typical application, when a comparator event is recorded, the application will need to know the analog input which caused the event. This function returns the value of type ADCHS_AN_INPUT_ID, indicating the analog input ID.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

ADCHS_AN_INPUT_ID input;

// Get analog input tied to digital comparator 2
input = PLIB_ADCHS_DigitalComparatorAnalogInputGet( MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_2 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
digComparator	Digital comparator ID of type ADCHS_DIGITAL_COMPARATOR_ID.

Function

```
ADCHS_AN_INPUT_ID PLIB_ADCHS_DigitalComparatorAnalogInputGet
(
    ADCHS_MODULE_ID index,
    ADCHS_DIGITAL_COMPARATOR_ID digComparator
)
```

PLIB_ADCHS_DigitalComparatorAnalogInputSelect Function

Selects analog inputs, whose converted data will be processed by the comparator.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorAnalogInputSelect(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID
digComparator, ADCHS_AN_INPUT_ID analogInput);
```

Returns

None.

Description

This function selects the analog input, whose converted data (once available) will be processed by the specified digital comparator. This function should be called only when the comparator is in a disabled state.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Set digital comparator 2 to process the output of AN2
PLIB_ADCHS_DigitalComparatorAnalogInputSelect( MY_ADCHS_INSTANCE,
                                    ADCHS_DIGITAL_COMPARATOR_2, ADCHS_AN2);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digComparator	Digital comparator ID of type ADCHS_DIGITAL_COMPARATOR_ID.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID. The valid input range for "analogInput" is from ADCHS_AN0 to ADCHS_AN31. Values from ADCHS_AN32 and higher will be ignored.

Function

void PLIB_ADCHS_DigitalComparatorAnalogInputSelect

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID digComparator,

ADCHS_AN_INPUT_ID analogInput

)

PLIB_ADCHS_DigitalComparatorDisable Function

Disables the specified digital comparator.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorDisable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID digComparator);
```

Returns

None.

Description

This function disables (turns OFF) the specified digital comparator.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_DigitalComparatorDisable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_2);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digComparator	Digital comparator of type ADCHS_DIGITAL_COMPARATOR_ID.

Function

void PLIB_ADCHS_DigitalComparatorDisable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID digComparator

)

PLIB_ADCHS_DigitalComparatorEnable Function

Enables the specified digital comparator.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorEnable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID digComparator);
```

Returns

None.

Description

This function enables (turns ON) the specified digital comparator.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_DigitalComparatorEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_2);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digComparator	Digital comparator of type ADCHS_DIGITAL_COMPARATOR_ID.

Function

void PLIB_ADCHS_DigitalComparatorEnable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID digComparator

)

PLIB_ADCHS_DigitalComparatorEventHasOccurred Function

Returns the status of the selected digital comparator.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DigitalComparatorEventHasOccurred(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID id);
```

Returns

None.

Description

This function returns the status of the selected digital comparator, whether or not an event related to this comparator has occurred. A comparator event will occur if the analog input selected for the comparator is outside the set limits.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
```

```
// Check, if comparator event has occurred for digital comparator 2
if(PLIB_ADCHS_DigitalComparatorEventHasOccurred( MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_2 ))
{
    // Perform related tasks
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the digital comparator in this device.

Function

bool PLIB_ADCHS_DigitalComparatorEventHasOccurred

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID id

)

PLIB_ADCHS_DigitalComparatorInterruptDisable Function

Disables the interrupt for the selected digital comparator.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorInterruptDisable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID
digComparator);
```

Returns

None.

Description

This function disables (turns OFF) the interrupt for the selected digital comparator. The interrupt will not be generated, once a comparator event occurs.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable interrupt for comparator 2
PLIB_ADCHS_DigitalComparatorInterruptDisable( MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_2 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digComparator	Digital comparator of type ADCHS_DIGITAL_COMPARATOR_ID.

Function

void PLIB_ADCHS_DigitalComparatorInterruptDisable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID digComparator

)

PLIB_ADCHS_DigitalComparatorInterruptEnable Function

Enables the interrupt for the selected digital comparator.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorInterruptEnable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID
digComparator);
```

Returns

None.

Description

This function enables (turns ON) the interrupt for the selected digital comparator. The interrupt is generated when the converted analog data for the related analog input goes out of the specified limits (set for comparator).

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable interrupt for comparator 2
PLIB_ADCHS_DigitalComparatorInterruptEnable( MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_2 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digComparator	Digital comparator of type ADCHS_DIGITAL_COMPARATOR_ID.

Function

void PLIB_ADCHS_DigitalComparatorInterruptEnable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID digComparator

)

PLIB_ADCHS_DigitalComparatorLimitSet Function

Sets the limit for the specified digital comparator.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorLimitSet(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID id, int16_t
hiLimit, int16_t loLimit);
```

Returns

None.

Description

This function sets the high and low limits for the specified digital comparator.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the Digital Comparator
// Creates an event when the reading of AN3 is between 20% to 80% of the
// full scale 12-bit output.
PLIB_ADCHS_DigitalComparatorLimitSet( MY_ADCHS_INSTANCE,    // ADCHS module ID
                      ADCHS_DIGITAL_COMPARATOR_1,           // Comparator ID
                      0x0CCD,                               // high limit, 80% of full scale
                      0x0333);                              // low limit, 20% of full scale
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the digital comparator in this device.
hiLimit	High limit in the same format as the conversion result.
loLimit	Low limit in the same format as the conversion result.

Function

void PLIB_ADCHS_DigitalComparatorLimitSet

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_COMPARATOR_ID id,

int16_t hiLimit,

int16_t loLimit

)

PLIB_ADCHS_DigitalComparatorSetup Function

Configures the Digital Comparator on the High-Speed SAR ADC converter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalComparatorSetup(ADCHS_MODULE_ID index, ADCHS_DIGITAL_COMPARATOR_ID id, bool
intEnable, bool inBetweenOrEqual, bool greaterEqualHi, bool lessThanHi, bool greaterEqualLo, bool
lessThanLo, ADCHS_AN_INPUT_ID analogInput, int16_t hiLimit, int16_t loLimit);
```

Returns

None.

Description

This function configures all parameters for the Digital Comparator of the ADCHS module.

Remarks

This function must be called when the ADC is disabled. The format of hiLimit and loLimit must match the output format of analog input specified in analogInput.

Preconditions

The module is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the Digital Comparator
// Creates an event when the reading of AN3 is between 20% to 80% of the
// full scale 12-bit output.
PLIB_ADCHS_DigitalComparatorSetup( MY_ADCHS_INSTANCE,   // ADCHS module ID
                       ADCHS_DIGITAL_COMPARATOR_1,      // Comparator ID
                       false,                           // No Int Enable
                       true,                            // test for between low and high
                       false,                           // no test for greater than equal to high
                       false,                           // no test for less than high
                       false,                           // no test for greater than equal to low
                       false,                           // no test for less than low
                       ADCHS_AN3,                       // select AN3
                       0x0CCD,                          // high limit, 80% of full scale
                       0x0333);                         // low limit, 20% of full scale

// Enable Digital Comparator 1
PLIB_ADCHS_DigitalComparatorEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_COMPARATOR_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the digital comparator specified.
intEnable	When true, comparator events generates interrupt.
inBetweenOrEqual	Event is generated when result is greater than or equal to loLimit and less than hiLimit.
greaterEqualHi	Event is generated when result is greater than or equal to hiLimit.
lessThanHi	Event is generated when result is less than hiLimit.
greaterEqualLo	Event is generated when result is greater than or equal to loLimit.
lessThanLo	Event is generated when result is less than loLimit.
analogInput	An analog input selection of type ADCHS_AN_INPUT_ID. The valid input range for "analogInput" is from ADCHS_AN0 to ADCHS_AN31. Values from ADCHS_AN32 and higher will be ignored.
hiLimit	High limit in the same format as the conversion result.
loLimit	Low limit in the same format as the conversion result.

Function

```
void PLIB_ADCHS_DigitalComparatorSetup
(
    ADCHS_MODULE_ID index,          // ADCHS channel ID
    ADCHS_DIGITAL_COMPARATOR_ID id,    // Comparator ID
bool intEnable,                 // Int Enable
bool inBetweenOrEqual,          // between low and high
bool greaterEqualHi,            // greater than equal to high
bool lessThanHi,                // less than high
bool greaterEqualLo,            // greater than equal to low
bool lessThanLo,                // less than low
    ADCHS_AN_INPUT_ID analogInput,     // input enable bits
int16_t  hiLimit,               // high limit
int16_t  loLimit                // low limit
)
```

f) Digital Filter Functions

PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect Function

Selects the number of samples which are averaged by the Digital Filter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID
id, ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT count);
```

Returns

None.

Description

This function selects the number of samples which are averaged by the Digital Filter, when the Digital Filter is configured for Averaging mode.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The Digital Filter is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable Digital Filter
PLIB_ADCHS_DigitalFilterDisable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);

// Select the sample count to be 64 samples
PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect( MY_ADCHS_INSTANCE,      // ADCHS module ID
                              ADCHS_DIGITAL_FILTER_1,                          // Filter ID
                              ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_64);   // 64 samples averaged

// Enable Digital Filter-1
PLIB_ADCHS_DigitalFilterEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the Digital Filter in this device.
count	Sets the number of samples which will be averaged by the Digital Filter. This variable is of type. ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT.

Function

```
void PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect
(
    ADCHS_MODULE_ID index,                  // ADCHS module ID
    ADCHS_DIGITAL_FILTER_ID id,             // Filter ID
    ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT count     // Sample count
)
```

PLIB_ADCHS_DigitalFilterAveragingModeSetup Function

Configures the Digital Filter on the High-Speed SAR ADC converter in Averaging mode.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterAveragingModeSetup(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID id,
ADCHS_AN_INPUT_ID analogInput, ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS length,
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT count, bool intEnable);
```

Returns

None.

Description

This function configures the Digital Filter on the High-Speed SAR ADC converter in Averaging mode.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The Digital Filter is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the Digital Filter in Averaging mode
// AN4 is averaged with 64 samples.  No global interrupt is enabled.
PLIB_ADCHS_DigitalFilterAveragingModeSetup( MY_ADCHS_INSTANCE,               // ADCHS module ID
                                ADCHS_DIGITAL_FILTER_1,                     // Filter ID
                                ADCHS_AN4,                                  // Oversample AN4
                                ADCHS_DIGITAL_FILTER_SIGNIFICANT_ALL_16BITS, // all 16bits significant
                                ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_64, // 64 samples averaged
                                false );                                    // No Int Enable

// Enable Digital Filter-1
PLIB_ADCHS_DigitalFilterEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the Digital Filter in this device.
analogInput	Identifier for the analog input to be filtered.
length	Sets the significant date length.
count	Sets the number of samples which will be averaged by Digital Filter. This variable is of type. ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT.
intEnable	When set, Filter events generated interrupt.

Function

```
void PLIB_ADCHS_DigitalFilterAveragingModeSetup
(
    ADCHS_MODULE_ID index,                  // ADCHS module ID
    ADCHS_DIGITAL_FILTER_ID id,               // Filter ID
    ADCHS_AN_INPUT_ID analogInput,              // analog input ID
    ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS length,    // Significant data bit
    ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT count, // Sample count
bool intEnable                          // Int Enable
)
```

PLIB_ADCHS_DigitalFilterDataGet Function

Used to fetch the data result from the Digital Filter.

File

plib_adchs.h

C

```
int16_t PLIB_ADCHS_DigitalFilterDataGet(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID dfltrID);
```

Returns

A 16-bit result in the format specified by the filter's configuration setting.

Description

This function is used to fetch data from the Digital Filter.

Remarks

None.

Preconditions

This function will function only if the Digital Filter was already configured (in Oversampling mode or Averaging mode) and the Digital Filter was enabled.

Example

```
int16_t myDFLTRResult;

if (PLIB_ADCHS_DigitalFilterDataIsReady(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1)) {
    // fetch data
    myDFLTRResult = PLIB_ADCHS_DigitalFilterDataGet(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
    // process result
    ...
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
dfltrID	Identifier for the Digital Filter in this device.

Function

int16_t PLIB_ADCHS_DigitalFilterDataGet

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_FILTER_ID dfltrID

)

PLIB_ADCHS_DigitalFilterDataIsReady Function

Used to determine if the Digital Filter has data ready.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DigitalFilterDataIsReady(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID id);
```

Returns

- true - Digital Filter has filtered data ready to be read
- false - Digital Filter data is not yet ready

Description

This function can be used to determine if the ADCHS Digital Filter has data ready. A 'true' is returned when data is available, which can be fetched using PLIB_ADCHS_DigitalFilterDataGet.

Remarks

None.

Preconditions

None.

Example

```
if (PLIB_ADCHS_DigitalFilterDataIsReady(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1)) {
            // fetch and process data
    }
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the Digital Filter in this device.

Function

bool PLIB_ADCHS_DigitalFilterDataIsReady

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_FILTER_ID id

)

PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable Function

Disables the interrupt for the selected Digital Filter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID
digFilter);
```

Returns

None.

Description

This function disables (turns OFF) the interrupt for the selected Digital Filter. The interrupt will not be generated, once filter data is ready.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable interrupt for filter 2
PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable( MY_ADCHS_INSTANCE, ADCHS_DFLTR2 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digFilter	Digital Filter of type ADCHS_DIGITAL_FILTER_ID.

Function

void PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_FILTER_ID digFilter

)

PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable Function

Enables the interrupt for the selected Digital Filter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID
digFilter);
```

Returns

None.

Description

This function enables (turns ON) the interrupt for the selected Digital Filter. The interrupt is generated when the Digital Filter completes the filtering and data is ready in the register (to be read).

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable interrupt for filter - 2
PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable( MY_ADCHS_INSTANCE, ADCHS_DFLTR2 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
digFilter	Digital Filter of type ADCHS_DIGITAL_FILTER_ID.

Function

void PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_FILTER_ID digFilter

)

PLIB_ADCHS_DigitalFilterDisable Function

Disables the Digital Filter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterDisable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID id);
```

Returns

None.

Description

This function Disables (turns OFF) the selected Digital Filter.

Remarks

None.

Preconditions

None..

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable Digital Filter 1
PLIB_ADCHS_DigitalFilterDisable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the Digital Filter in the ADCHS channel.

Function

void PLIB_ADCHS_DigitalFilterDisable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_FILTER_ID id

)

PLIB_ADCHS_DigitalFilterEnable Function

Enables the Digital Filter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterEnable(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID id);
```

Returns

None.

Description

This function enables (turns ON) the selected Digital Filter.

Remarks

None.

Preconditions

The ADC channel should be configured using the PLIB_ADCHS_Setup function prior to enabling.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable Digital Filter-1
PLIB_ADCHS_DigitalFilterEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the digital Filter in the ADCHS channel.

Function

void PLIB_ADCHS_DigitalFilterEnable

(

ADCHS_MODULE_ID index,

ADCHS_DIGITAL_FILTER_ID id
)

PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect Function

Selects the oversampling ratio for the Digital Filter.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID id,
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO ratio);
```

Returns

None.

Description

This function selects the oversampling ratio for the Digital Filter. This function can be used to change the oversampling ratio of the Digital Filter, when set in Oversampling mode.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The Digital Filter is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable Digital Filter
PLIB_ADCHS_DigitalFilterDisable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);

// Select the oversampling ratio
PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect( MY_ADCHS_INSTANCE,     // ADCHS module ID
                              ADCHS_DIGITAL_FILTER_1,                       // Filter ID
                              ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_16X);  // 16 x oversampling


// Enable Digital Filter-1
PLIB_ADCHS_DigitalFilterEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the Digital Filter in this device.
ratio	Sets the oversampling filter ratio. This variable is of type ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO.

Function

void PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect
(
ADCHS_MODULE_ID index, // ADCHS module ID
ADCHS_DIGITAL_FILTER_ID id, // Filter ID
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO ratio // Oversampling ratio
)

PLIB_ADCHS_DigitalFilterOversamplingModeSetup Function

Configures the Digital Filter on the High-Speed SAR ADC converter in Oversampling mode.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DigitalFilterOversamplingModeSetup(ADCHS_MODULE_ID index, ADCHS_DIGITAL_FILTER_ID id,
ADCHS_AN_INPUT_ID analogInput, ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS length,
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO ratio, bool intEnable);
```

Returns

None.

Description

This function configures the Digital Filter on the High-Speed SAR ADC converter in Oversampling mode.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The Digital Filter channel is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the Digital Filter in Oversampling mode
// AN4 is oversampled at a 16X rate.  No global interrupt is enabled.
PLIB_ADCHS_DigitalFilterOversamplingModeSetup( MY_ADCHS_INSTANCE,           // ADCHS module ID
                               ADCHS_DIGITAL_FILTER_1,                     // Filter ID
                               ADCHS_AN4,                                  // Oversample AN4
                               ADCHS_DIGITAL_FILTER_SIGNIFICANT_ALL_16BITS, // all 16bits significant
                               ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_16X,   // 16 x oversampling
                               false );                                    // No Int Enable

// Enable Digital Filter-1
PLIB_ADCHS_DigitalFilterEnable(MY_ADCHS_INSTANCE, ADCHS_DIGITAL_FILTER_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
id	Identifier for the Digital Filter in this device.
analogInput	Identifier for the analog input to be filtered.
length	Sets the significant data length.
ratio	Sets the oversampling filter ratio. This variable is of type ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO.
intEnable	When set, Filter events generated interrupt.

Function

void PLIB_ADCHS_DigitalFilterOversamplingModeSetup

(

ADCHS_MODULE_ID index, // ADCHS ID

ADCHS_DIGITAL_FILTER_ID id, // Filter ID

ADCHS_AN_INPUT_ID analogInput, // analog input ID

ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS length, // Significant data bit

ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO ratio, // Oversampling ratio

bool intEnable // Int Enable

)

g) DMA Functions

PLIB_ADCHS_DMABuffer_A_InterruptDisable Function

Disables the DMA Buffer A full interrupt for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMABuffer_A_InterruptDisable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully disabled interrupt
- false - The function could not disable the interrupt, as the selected Channel is 7, which does not implement this feature

Description

This function disables (turns OFF) the DMA Buffer A full interrupt for the specified Channel 0 to 6. Since Channel 7 does not implement this feature, passing Channel 7 will result in an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable DMA interrupt
PLIB_ADCHS_DMABuffer_A_InterruptDisable( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

bool PLIB_ADCHS_DMABuffer_A_InterruptDisable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID // Channel ID 0 to 6

)

PLIB_ADCHS_DMABuffer_A_InterruptEnable Function

Enables the DMA Buffer A full interrupt for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMABuffer_A_InterruptEnable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully enabled interrupt
- false - The function could not enable the interrupt, as the selected Channel is 7, which does not implement this feature

Description

This function enables (turns ON) the DMA Buffer A full interrupt for the specified Channel 0 to 6. The interrupt is generated when the DMA buffer A is full. Since Channel 7 does not implement this feature, passing Channel 7 will result in an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable DMA interrupt
PLIB_ADCHS_DMABuffer_A_InterruptEnable( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

bool PLIB_ADCHS_DMABuffer_A_InterruptEnable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID // Channel ID 0 to 6

)

PLIB_ADCHS_DMABuffer_A_IsFull Function

Used to determine if the DMA Buffer A is full, for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
int8_t PLIB_ADCHS_DMABuffer_A_IsFull(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- 1 - Buffer is full
- 0 - Buffer is empty
- -1 - Invalid channel ID passed

Description

This function can be used to determine if the ADCHS DMA Buffer A is full for the specified Channel 0 to 6.

A value of '1' is returned when the Buffer A is full. A value of '0' is returned when the Buffer A is empty.

This feature is not implemented for Channel 7. Therefore, passing Channel 7 returns an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

if (1 == PLIB_ADCHS_DMABuffer_A_IsFull(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 ))
{
    // process data, if DMA Buffer A is full
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID from 0 to 6.

Function

int8_t PLIB_ADCHS_DMABuffer_A_IsFull

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID // channel ID 0 to 6

)

PLIB_ADCHS_DMABuffer_B_InterruptDisable Function

Disables the DMA Buffer B full interrupt for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMABuffer_B_InterruptDisable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully disabled interrupt
- false - The function could not disable the interrupt, as the selected Channel is 7, which does not implement this feature

Description

This function disables (turns OFF) the DMA Buffer B full interrupt for specified Channel 0 to 6. Since Channel 7 does not implement this feature, passing Channel 7 will result in an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable DMA interrupt
PLIB_ADCHS_DMABuffer_B_InterruptDisable( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

bool PLIB_ADCHS_DMABuffer_B_InterruptDisable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID // Channel ID 0 to 6

)

PLIB_ADCHS_DMABuffer_B_InterruptEnable Function

Enables the DMA Buffer B full interrupt for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMABuffer_B_InterruptEnable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully enabled interrupt
- false - The function could not enable the interrupt, as the selected Channel is 7, which does not implement this feature

Description

This function enables (turns ON) the DMA Buffer B full interrupt for specified Channel 0 to 6. The interrupt is generated when the DMA Buffer B is full. Since Channel 7 does not implement this feature, passing Channel 7 will result in an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable DMA interrupt
PLIB_ADCHS_DMABuffer_B_InterruptEnable( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

void PLIB_ADCHS_DMABuffer_B_InterruptEnable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID // Channel ID 0 to 6

)

PLIB_ADCHS_DMABuffer_B_IsFull Function

Used to determine if the DMA Buffer B is full, for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
int8_t PLIB_ADCHS_DMABuffer_B_IsFull(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- 1 - Buffer is full
- 0 - Buffer is empty
- -1 - Invalid channel ID passed

Description

This functino is used to determine if the ADCHS DMA Buffer B is full for the specified Channel 0 to 6.

A value of '1' is returned when Buffer B is full. A value of '0' is returned when Buffer B is empty.

This feature is not implemented for Channel 7. Therefore, passing Channel 7 returns an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

if (1 == PLIB_ADCHS_DMABuffer_B_IsFull(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 ))
{
    // process data, if DMA Buffer B is full
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID from 0 to 6.

Function

int8_t PLIB_ADCHS_DMABuffer_B_IsFull
(
    ADCHS_MODULE_ID index,
    ADCHS_CHANNEL_ID channelID // Channel ID 0 to 6
)

PLIB_ADCHS_DMADisable Function

Disables the DMA in the High-Speed SAR ADC module.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DMADisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the DMA in the High-Speed SAR ADC module.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable DMA
PLIB_ADCHS_DMADisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_DMADisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_DMAEnable Function

Enables the DMA in the High-Speed SAR ADC module.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DMAEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the DMA in the High-Speed SAR ADC module.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable DMA
PLIB_ADCHS_DMAEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_DMAEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_DMAOverflowErrorHasOccurred Function

Used to determine if the DMA Buff had an overflow error.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMAOverflowErrorHasOccurred(ADCHS_MODULE_ID index);
```

Returns

Boolean:

- true - If an overflow error occurred
- false - If an overflow error has not occurred

Description

DMA Buffers are circular in nature. If reading the data from DMA buffer is slower as compared to filling the buffer, the newer data will overwrite the previous data (before the data is read). This function returns 'true' if an overflow error has occurred.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

if (PLIB_ADCHS_DMAOverflowErrorHasOccurred(MY_ADCHS_INSTANCE)
{
    // Error occurred
    while(1);
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance

Function

bool PLIB_ADCHS_DMAOverflowErrorHasOccurred(ADCHS_MODULE_ID index)

PLIB_ADCHS_DMASetup Function

Configures the DMA on the High-Speed SAR ADC.

File

plib_adchs.h

C

```
void PLIB_ADCHS_DMASetup(ADCHS_MODULE_ID index, ADCHS_DMA_BUFFER_LENGTH bufferLengthBytes, uint32_t
baseAddress, ADCHS_DMA_COUNT countMode, uint32_t countBaseAddress);
```

Returns

None.

Description

This functin configures all parameters for the DMA of the High-Speed SAR ADC.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The ADC channel and the Channel 0 to 6 using DMA are disabled.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the DMA
void PLIB_ADCHS_DMASetup(   MY_ADCHS_INSTANCE,              // ADCHS module ID
                            ADCHS_DMA_BUFFER_LENGTH_8BYTES, // buffer length is 8 Bytes
                            0xA002,                         // Base address is 0xA002
                            ADCHS_DMA_COUNT_ENABLE,         // count is enabled
                            0xC400  );                      // Count base address is 0xC400

// Enable DMA
PLIB_ADCHS_DMAEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
bufferLengthBytes	Length of DMA Buffer in number of bytes. Buffer is saved on RAM. The variable is of type ADCHS_DMA_BUFFER_LENGTH.

baseAddress	Address of RAM, which holds the buffer.
countMode	Enable/ Disable the DMA feature to keep count of number of converted data saved by DMA.
countBaseAddress	Address of RAM, which holds the counts.

Function

```
void PLIB_ADCHS_DMASetup
(
    ADCHS_MODULE_ID index,                  // ADCHS module ID
    ADCHS_DMA_BUFFER_LENGTH bufferLengthBytes,    // Buffer length
uint32_t baseAddress,                   // Base address for data
    ADCHS_DMA_COUNT countMode,              // Enable/ Disable count
uint32_t countBaseAddress               // Base address for count
)
```

PLIB_ADCHS_DMASourceRemove Function

Disables the DMA for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMASourceRemove(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully removed the DMA source
- false - The function could not remove the DMA source, as selected Channel is 7, which does not implement this feature

Description

This function disables (turns OFF) the DMA for specified Channel 0 to 6. If Channel 7 is selected, this function returns error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// DMA for Channel 1
if(!PLIB_ADCHS_DMASourceRemove( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 ))
{
    // error
    while(1);
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

```
bool PLIB_ADCHS_DMASourceRemove
(
    ADCHS_MODULE_ID index,
    ADCHS_CHANNEL_ID channelID      // Channel ID 0 to 6
)
```

PLIB_ADCHS_DMASourceSelect Function

Enables the DMA for the specified Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_DMASourceSelect(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully selected the DMA source
- false - The function could not select the DMA source, as selected Channel is 7, which does not implement this feature

Description

This function enables (turns ON) the DMA for specified Channel 0 to 6. If selected Channel is 7, this function return error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// DMA for Channel 1
if(!PLIB_ADCHS_DMASourceSelect( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 ))
{
    // error
    while(1);
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

```
bool PLIB_ADCHS_DMASourceSelect
(
    ADCHS_MODULE_ID index,
    ADCHS_CHANNEL_ID channelID      // Channel ID 0 to 6
)
```

h) Channel Related Functions

PLIB_ADCHS_ChannelAnalogFeatureDisable Function

Disables the analog circuit for channels of High-Speed SAR ADC.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelAnalogFeatureDisable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

None.

Description

This function disables (turns OFF) the analog circuit for channels. The analog circuit for unused channels can be disabled to reduce current consumption.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_ChannelAnalogFeatureDisable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	ID of the required channel.

Function

void PLIB_ADCHS_ChannelAnalogFeatureDisable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID

)

PLIB_ADCHS_ChannelAnalogFeatureEnable Function

Enables the analog circuit for High-Speed SAR ADC channels.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelAnalogFeatureEnable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

None.

Description

This function enables (turns ON) the analog circuit for channels. The analog circuit can be disabled (when not required) to reduce current consumption by the channel. Since, each channel has the feature to individually enable/disable the analog circuit, the unused channel's analog circuit can be disabled. When the analog circuit for a channel is enabled, it needs a minimum warm-up time before which the channel is ready to perform conversion. Once the channel analog feature (circuit) is enabled, its ready status can be check with PLIB_ADCHS_ChannelIsReady function.

Remarks

None.

Preconditions

The ADCHS module should be enabled before calling this function. The ADCHS module can be enabled by calling the PLIB_ADCHS_Enable function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_ChannelAnalogFeatureEnable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);
```

```
// wait until the channel is ready
while(!PLIB_ADCHS_ChannelIsReady(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1));
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	ID of the required channel.

Function

void PLIB_ADCHS_ChannelAnalogFeatureEnable
(
    ADCHS_MODULE_ID index,
    ADCHS_CHANNEL_ID channelID
)

PLIB_ADCHS_ChannelConfigurationGet Function

Used to get the configuration for the specified channel.

File

plib_adchs.h

C

```
uint32_t PLIB_ADCHS_ChannelConfigurationGet(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

32-bit value of the configuration for ADCHS channel.

Description

This functions returns the configuration for the specified channel.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

uint32_t config;

config = PLIB_ADCHS_ChannelConfigurationGet(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID.

Function

uint32_t PLIB_ADCHS_ChannelConfigurationGet
(
    ADCHS_MODULE_ID index,
    ADCHS_CHANNEL_ID channelID // Channel ID
)

PLIB_ADCHS_ChannelConfigurationSet Function

Used to set the configuration for the specified channel.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelConfigurationSet(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID, uint32_t config);
```

Returns

None.

Description

This functions sets the configuration for the specified channel.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

uint32_t config = 0x12345678;

PLIB_ADCHS_ChannelConfigurationSet(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1, config);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID.
config	32-bit value for the configuration of channel.

Function

void PLIB_ADCHS_ChannelConfigurationSet
(
ADCHS_MODULE_ID index,
ADCHS_CHANNEL_ID channelID,
uint32_t config
)

PLIB_ADCHS_ChannelDigitalFeatureDisable Function

Disables the digital circuit for channels of High-Speed SAR ADC.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelDigitalFeatureDisable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

None.

Description

This function disables (turns OFF) the digital circuit for channels. The digital feature (circuit) of unused channels can be disabled to reduce current

consumption. The advantage of disabling the digital feature is that re-enabling the digital feature does not require any warm up time and the channel can be immediately used for conversion.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_ChannelDigitalFeatureDisable( MY_ADCHS_INSTANCE,
                                         ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	ID of the required channel.

Function

void PLIB_ADCHS_ChannelDigitalFeatureDisable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID

)

PLIB_ADCHS_ChannelDigitalFeatureEnable Function

Enables (turns ON) the digital circuit for channels.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelDigitalFeatureEnable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

None.

Description

This function enables (turns ON) the digital circuit for channels. Unlike enabling the analog feature, enabling the digital feature does not require any warm-up time.

Remarks

None.

Preconditions

The ADCHS module should be enabled before calling this function. The ADCHS module can be enabled by calling PLIB_ADCHS_Enable function. Also, for the channel to perform conversion, the analog features of the channel should be enabled using PLIB_ADCHS_ChannelAnalogFeatureEnable, prior to enabling the digital feature.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable analog feature for Channel 1
PLIB_ADCHS_ChannelAnalogFeatureEnable( MY_ADCHS_INSTANCE,
                                       ADCHS_CHANNEL_1 );

// wait until channel is ready
while(!PLIB_ADCHS_ChannelIsReady( MY_ADCHS_INSTANCE,
                                  ADCHS_CHANNEL_1 ));
```

```
// Enable the digital Channel 1
PLIB_ADCHS_ChannelDigitalFeatureEnable( MY_ADCHS_INSTANCE,
                                        ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	ID of the required channel.

Function

void PLIB_ADCHS_ChannelDigitalFeatureEnable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID

)

PLIB_ADCHS_ChannelIsReady Function

Returns the state of the channel.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ChannelIsReady(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - Channel is ready after warm-up time has elapsed
- false - Channel is not ready

Description

This function returns the state, indicating whether the channel is awake and ready after the warm-up time is complete. When a analog feature (circuit) of a channel is enabled, there is a warm-up time required before the channel is ready and can be used for conversion. The ready state of the channel can be acquired using this function.

Remarks

None.

Preconditions

The analog feature for the channel should be enabled using the PLIB_ADCHS_ChannelAnalogFeatureEnable function. Also, the ADCHS module should be enabled before calling this function.

Example

```
// Enable the analog feature for Channel 1
PLIB_ADCHS_ChannelAnalogFeatureEnable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);

// wait until the channel is ready
while(!PLIB_ADCHS_ChannelIsReady(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1));
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured
channelID	ID of the required channel

Function

bool PLIB_ADCHS_ChannelIsReady

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID

)

PLIB_ADCHS_ChannelIsReadyInterruptDisable Function

Disables the Channel ready interrupt for the specified channel.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelIsReadyInterruptDisable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

None.

Description

This function disables (turns OFF) the channel ready interrupt for the specified channel.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable wakeup interrupt for Channel 1
PLIB_ADCHS_ChannelIsReadyInterruptDisable( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID.

Function

void PLIB_ADCHS_ChannelIsReadyInterruptDisable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID // channel ID

)

PLIB_ADCHS_ChannelIsReadyInterruptEnable Function

Enables the Channel ready interrupt for the specified channel.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelIsReadyInterruptEnable(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

None.

Description

This function enables (turns ON) the channel ready interrupt for the specified channel. The interrupt is generated when the channel is ready after wake-up (following warm-up time).

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable wakeup interrupt for Channel 1
PLIB_ADCHS_ChannelIsReadyInterruptEnable( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID.

Function

void PLIB_ADCHS_ChannelIsReadyInterruptEnable

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID // channel ID

)

PLIB_ADCHS_ChannelSetup Function

Configures the High-Speed SAR ADC channels.

File

plib_adchs.h

C

```
void PLIB_ADCHS_ChannelSetup(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID, ADCHS_DATA_RESOLUTION res,
uint8_t channelClockDivider, uint16_t sampleTimeCount, ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK earlyInterruptClk);
```

Returns

None.

Description

This function configures all ADC parameters that are common to the specified ADC Channel 0 to 7.

This configuration must occur prior to enabling the ADC, and therefore, must be called when the ADC is disabled. Also, this configuration must occur prior to enabling the digital circuit and analog bias circuit for the specified ADC Channel 0 to 7.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The channel is disabled when calling this function.

Example

```
bool status = false;

// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the ADC
PLIB_ADCHS_ChannelSetup( MY_ADCHS_INSTANCE,
        ADCHS_CHANNEL_1,                        // channel - 1
        ADCHS_DATA_RESOLUTION_12BIT,            // resolution is set to 12bits
        1,                                      // channel clock divider bit is, TAD =  2 * TQ
        1,                                      // Sample time is 3 * TAD
        ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1);   // Interrupt, 1 clock early

status = PLIB_ADCHS_ChannelTriggerSampleSelect( MY_ADCHS_INSTANCE,
```

```
        ADCHS_CHANNEL_1,                        // channel - 1
        ADCHS_CHANNEL_SYNC_SAMPLING);           // Synchronous sampling selected

if( false == status)
{
    // error has occurred
    while(1);
}
// Enable the ADC
PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);

// Wait until the reference voltage is ready
while(!PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE));

//Check if there is a fault in reference voltage
if(PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE))
{
    // process fault accordingly
    while(1);
}

// Enable the analog circuit
PLIB_ADCHS_ChannelAnalogFeatureEnable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);

// Wait until the channel wakes up after warm-up
while(!PLIB_ADCHS_ChannelIsReady(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1));

// Enable the digital circuit
PLIB_ADCHS_ChannelDigitalFeatureEnable(MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1);

// ADC will begin conversion now, after a valid trigger
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID of the required ADC channel.
res	Sets the output data resolution.
channelClockDivider	Channel Clock source divider. Values range from 1 to 127. This divider determines the channel clock (clock frequency at which the channel 0 to 7 is running). The frequency of
channel clock, as per following equation	Channel clock frequency = (control clock frequency)/(channelClockDivider * 2). Please note that value of "0" for channelClockDivider is not allowed. The minimum value of channelClockDivider is "1", which also means that the highest channel clock frequency is half of control clock frequency.
sampleTimeCount	Sets the sample time for ADC channel, 0 to 7. Values range from
0 to 1023. The sample time is given by the equation	Sample time = ((1/Channel clock frequency) * (sampleTimeCount + 2)).
earlyInterruptClk	Sets the number of channel clocks prior to the availability of actual data that the early interrupt is generated. This variable if of type ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK. The selection of this parameter is dependent on the resolution of output data.

Function

```
void PLIB_ADCHS_ChannelSetup
(
    ADCHS_MODULE_ID index,
    ADCHS_CHANNEL_ID channelID,                 // channel ID 0 to 7
    ADCHS_DATA_RESOLUTION res,                  // Output data resolution
uint8_t channelClockDivider,                // Clock divider
uint16_t sampleTimeCount,                   // Sample time
    ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK earlyInterruptClk // early interrupt clock setting
)
```

PLIB_ADCHS_ChannelInputSelect Function

Selects the analog input for Channel 0 to 6.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ChannelInputSelect(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID, ADCHS_CHANNEL_INP_SEL
sel);
```

Returns

- true - The function successfully selected the analog input for the channel
- false - The function could not select the input, as selected channel is 7, which does not implement this feature. Otherwise, the selected analog input is not available for the selected channel.

Description

Selects the analog input for Channel 0 to 6. The inputs for Channel 0 to 6 can be changed between different inputs. This feature is not available for Channel 7; therefore, calling this function for Channel 7 will result in an error. Also, the input selected should match the ones available for the selected channel. Selecting an input which is not available for the channel will return an error.

Remarks

None.

Preconditions

None.

Example

```
bool status = false;

// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Set Channel-0 to use the first input.
status = PLIB_ADCHS_ChannelInputSelect
                    (
                        MY_ADCHS_INSTANCE,
                        ADCHS_CHANNEL_0,
                        ADCHS_CHANNEL_0_DEFAULT_INP_AN0
                    );

if(false == status)
{
    // error
    while(1);
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
channelID	Channel ID.
sel	Selection of inputs of type ADCHS_CHANNEL_INP_SEL.

Function

bool PLIB_ADCHS_ChannelInputSelect

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID channelID,

ADCHS_CHANNEL_INP_SEL sel

)

i) FIFO Functions

PLIB_ADCHS_FIFODataCountGet Function

Returns the number of data to be read from FIFO.

File

plib_adchs.h

C

```
uint8_t PLIB_ADCHS_FIFODataCountGet(ADCHS_MODULE_ID index);
```

Returns

The number of the data stored in the FIFO.

Description

This function returns the number of data which can be read from FIFO.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// variable to save data
uint32_t convData;

// Check, if FIFO has some data
while(!PLIB_ADCHS_FIFODataIsAvailable(MY_ADCHS_INSTANCE));

// Once data is available, check if the data belongs to channel ID -1
if(PLIB_ADCHS_FIFOSourceGet(MY_ADCHS_INSTANCE) == ADCHS_CHANNEL_1)
{
    // Continue reading FIFO, until the count is zero
    while(PLIB_ADCHS_FIFODataCountGet(MY_ADCHS_INSTANCE) != 0)
    {
        // Read from FIFO. For each read, the count is decremented
        convData = PLIB_ADCHS_FIFORead(MY_ADCHS_INSTANCE);
    }
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

uint8_t PLIB_ADCHS_FIFODataCountGet(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFODataIsAvailable Function

Used to determine if the FIFO has data ready.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_FIFODataIsAvailable(ADCHS_MODULE_ID index);
```

Returns

Boolean:

- true - If data is ready
- false - If data is not ready

Description

Used to determine if the ADCHS FIFO has data ready. A 'true' is returned when data is available, which can be fetched using PLIB_ADCHS_FIFORead.

Remarks

None.

Preconditions

None.

Example

```
if (PLIB_ADCHS_FIFODataIsAvailable(MY_ADCHS_INSTANCE)) {
            // fetch and process data
    }
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

bool PLIB_ADCHS_FIFODataIsAvailable(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFODataIsNegative Function

Returns the sign of data stored in FIFO.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_FIFODataIsNegative(ADCHS_MODULE_ID index);
```

Returns

Boolean:

- true - If sign is negative
- false - If sign is not negative

Description

This function returns the sign of data stored in the FIFO.

Remarks

None.

Preconditions

None.

Example

```
if (PLIB_ADCHS_FIFODataIsNegative(MY_ADCHS_INSTANCE)) {
            //process data
    }
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

bool PLIB_ADCHS_FIFODataIsNegative(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFODataReadyInterruptDisable Function

Disables the interrupt for FIFO.

File

plib_adchs.h

C

```
void PLIB_ADCHS_FIFODataReadyInterruptDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the interrupt for FIFO.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable interrupt for FIFO
PLIB_ADCHS_FIFODataReadyInterruptDisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_FIFODataReadyInterruptDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFODataReadyInterruptEnable Function

Enables the interrupt for FIFO.

File

plib_adchs.h

C

```
void PLIB_ADCHS_FIFODataReadyInterruptEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the interrupt for FIFO. The interrupt is generated when the FIFO has data to be read.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable interrupt for FIFO
PLIB_ADCHS_FIFODataReadyInterruptEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_FIFODataReadyInterruptEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFODisable Function

Disables the FIFO in the High-Speed SAR ADC.

File

plib_adchs.h

C

```
void PLIB_ADCHS_FIFODisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the FIFO.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable FIFO
PLIB_ADCHS_FIFODisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_FIFODisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFOEnable Function

Enables the FIFO in the High-Speed SAR ADC

File

plib_adchs.h

C

```
void PLIB_ADCHS_FIFOEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the FIFO.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable FIFO
PLIB_ADCHS_FIFOEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

void PLIB_ADCHS_FIFOEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFOErrorHasOccurred Function

Used to determine if the FIFO has encountered an overflow error.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_FIFOErrorHasOccurred(ADCHS_MODULE_ID index);
```

Returns

- true - An error has occurred
- false - No error occurred has in the FIFO

Description

This function can be used to determine if the ADCHS FIFO had a data, which was overwritten by the next round of a FIFO write (overflow error). A 'true' is returned when an overflow error has occurred.

Remarks

None.

Preconditions

None.

Example

```
if (!PLIB_ADCHS_FIFOErrorHasOccurred(MY_ADCHS_INSTANCE)) {
            // process data, if overflow error did not occur
    }
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

bool PLIB_ADCHS_FIFOErrorHasOccurred(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFORead Function

Used to fetch the data result from the FIFO.

File

plib_adchs.h

C

```
int32_t PLIB_ADCHS_FIFORead(ADCHS_MODULE_ID index);
```

Returns

A 32-bit result in the format specified by the ADC configuration for the module using the FIFO.

Description

This function is used to fetch data from the FIFO. The FIFO can only be assigned to a Class 1 ADC analog input.

Remarks

None.

Preconditions

The FIFO should have data ready to be read from it.

Example

```
int32_t myFIFOResult;

if (PLIB_ADCHS_FIFODataIsAvailable(MY_ADCHS_INSTANCE)) {
    // fetch data
    myDFLTRResult = PLIB_ADCHS_FIFORead(MY_ADCHS_INSTANCE);
    // process result
    ...
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

int32_t PLIB_ADCHS_FIFORead(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFOSourceGet Function

Returns the channel ID using the FIFO.

File

plib_adchs.h

C

```
ADCHS_CHANNEL_ID PLIB_ADCHS_FIFOSourceGet(ADCHS_MODULE_ID index);
```

Returns

The Channel ID of data type ADCHS_CHANNEL_ID.

Description

This function returns the channel ID of Channel 0 to 6, whose data is stored on the FIFO.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// variable to save data
uint32_t convData;

// Check, if FIFO has some data
while(!PLIB_ADCHS_FIFODataIsAvailable(MY_ADCHS_INSTANCE));

// Once data is available, check if the data belongs to channel ID -1
if(PLIB_ADCHS_FIFOSourceGet(MY_ADCHS_INSTANCE) == ADCHS_CHANNEL_1)
{
    // Continue reading FIFO, until the count is zero
    while(PLIB_ADCHS_FIFODataCountGet(MY_ADCHS_INSTANCE) != 0)
    {
        // Read from FIFO. For each read, the count is decremented
        convData = PLIB_ADCHS_FIFORead(MY_ADCHS_INSTANCE);
    }
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

ADCHS_CHANNEL_ID PLIB_ADCHS_FIFOSourceGet(ADCHS_MODULE_ID index)

PLIB_ADCHS_FIFOSourceSelect Function

Sets the Channel 0 to 6 using the FIFO.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_FIFOSourceSelect(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID channelID);
```

Returns

- true - The function successfully selected the FIFO source
- false - The function could not select the FIFO source, as selected Channel is 7, which does not implement this feature

Description

This function sets the Channel 0 to 6 ID, which would be storing data into FIFO. If this function is called with Channel ID as 7, the function will return an error.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Set Channel 1 to be storing data into FIFO
if(!PLIB_ADCHS_FIFOSourceSelect( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_1 ))
{
    // error has occurred
    while(1);
}
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.
channelID	Channel ID from 0 to 6.

Function

bool PLIB_ADCHS_FIFOSourceSelect
(
ADCHS_MODULE_ID index,
ADCHS_CHANNEL_ID channelID // Channel ID from 0 to 6
)

j) Interrupt Functions

PLIB_ADCHS_EarlyInterruptDisable Function

Disables the early interrupt for the ADCHS.

File

plib_adchs.h

C

```
void PLIB_ADCHS_EarlyInterruptDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function Disables (turns OFF) the early interrupt for the entire ADCHS. Since early interrupt function is disabled, setting early interrupt enable for individual analog input will not work.

Further selection (enabling/disabling) of interrupt feature for individual analog input is possible through the functions PLIB_ADCHS_AnalogInputDataReadyInterruptEnable or PLIB_ADCHS_AnalogInputDataReadyInterruptDisable.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable early interrupt
PLIB_ADCHS_EarlyInterruptDisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_EarlyInterruptDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_EarlyInterruptEnable Function

Enables the early interrupt for the ADCHS.

File

plib_adchs.h

C

```
void PLIB_ADCHS_EarlyInterruptEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the early interrupt for the entire ADCHS. Further selection (enabling/disabling) of early interrupt feature for individual analog input through the functions PLIB_ADCHS_AnalogInputEarlyInterruptEnable or PLIB_ADCHS_AnalogInputEarlyInterruptDisable.

The interrupt is generated at a set number of clock prior to conversion complete. The number of clock for early interrupt is set using the "configure" function for specific channel.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable early interrupt
PLIB_ADCHS_EarlyInterruptEnable( MY_ADCHS_INSTANCE );

// Now, enable the early interrupt for each analog inputs
PLIB_ADCHS_AnalogInputEarlyInterruptEnable(MY_ADCHS_INSTANCE, ADCHS_AN10);
PLIB_ADCHS_AnalogInputEarlyInterruptEnable(MY_ADCHS_INSTANCE, ADCHS_AN11);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_EarlyInterruptEnable(ADCHS_MODULE_ID index)

k) Turbo Mode Functions

PLIB_ADCHS_TurboModeChannelSelect Function

Configures the channels for Turbo mode.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_TurboModeChannelSelect(ADCHS_MODULE_ID index, ADCHS_CHANNEL_ID masterChannelID,
ADCHS_CHANNEL_ID slaveChannelID);
```

Returns

- true - The function successfully selected the channels for Turbo mode
- false - The function could not select the channels for Turbo mode for the following reasons:
- Both the master and slave channels are the same
- Channel 7 was used to set up Turbo mode (either as a master or a slave or both)

Description

While running the High-Speed SAR ADC in Turbo mode, two channels (from 0 to 6) are interleaved to get higher throughput. This function configures the two channels. This configuration must occur prior to enabling the ADC and prior to enabling Turbo mode. Only valid channels from 0 to 6 can be made as master or slave. If Channel 7 is used, the function will return error

Remarks

This function must be called when the ADC is disabled.

Preconditions

The channels are disabled and Turbo mode is disabled when calling this function.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Configure the ADC with Channel 0 as master and Channel 1 as slave
if(!PLIB_ADCHS_TurboModeChannelSelect( MY_ADCHS_INSTANCE, ADCHS_CHANNEL_0, ADCHS_CHANNEL_1))
{
    // Error occurred while setting up Turbo mode
    while(1);
}

// Enable Turbo mode
PLIB_ADCHS_TurboModeEnable(MY_ADCHS_INSTANCE);

// Check for Turbo mode error
if(PLIB_ADCHS_TurboModeErrorHasOccurred(MY_ADCHS_INSTANCE);
{
    // Configuration of Turbo mode has caused an error
    // Process error here or reconfigure Turbo Mode
    while(1);
}

// Enable the ADC
PLIB_ADCHS_Enable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.
masterChannelID	Channel ID of master channel. Valid channel IDs are from 0 to 6.
slaveChannelID	Channel ID of slave channel. Valid channel IDs are from 0 to 6.

Function

bool PLIB_ADCHS_TurboModeChannelSelect

(

ADCHS_MODULE_ID index,

ADCHS_CHANNEL_ID masterChannelID,

ADCHS_CHANNEL_ID slaveChannelID

)

PLIB_ADCHS_TurboModeDisable Function

Disables Turbo mode for High-Speed SAR ADC module.

File

plib_adchs.h

C

```
void PLIB_ADCHS_TurboModeDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) Turbo mode on the High-Speed SAR ADC module.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_TurboModeDisable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_TurboModeDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_TurboModeEnable Function

Enables Turbo mode for the High-Speed SAR ADC module.

File

plib_adchs.h

C

```
void PLIB_ADCHS_TurboModeEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) Turbo mode on the High-Speed SAR ADC module.

Remarks

None.

Preconditions

The master and slave channels must be selected to work in Turbo mode, before enabling Turbo mode.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.
PLIB_ADCHS_TurboModeEnable(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_TurboModeEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_TurboModeErrorHasOccurred Function

Returns the error state of Turbo mode.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_TurboModeErrorHasOccurred(ADCHS_MODULE_ID index);
```

Returns

- true - Turbo mode error has occurred
- false - Turbo mode error has not occurred

Description

Returns the state of error bit for Turbo mode. When Turbo mode is enabled, some parameters of master and slave channels should be same. They are ADC channel clock divisor, ADC resolution, and Sampling time. Also, both master and slave channels should use synchronous sampling. If any or all these conditions are not met, the Turbo mode error will be set, which can be read by calling

Remarks

None.

Preconditions

None.

Example

```
bool errorState = PLIB_ADCHS_TurboModeErrorHasOccurred(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance.

Function

bool PLIB_ADCHS_TurboModeErrorHasOccurred(ADCHS_MODULE_ID index)

l) Voltage Reference Functions

PLIB_ADCHS_VREFFaultHasOccurred Function

Returns the state of VREF fault.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_VREFFaultHasOccurred(ADCHS_MODULE_ID index);
```

Returns

- true - A VREF Fault has occurred
- false - No Fault occurred on VREF

Description

This function returns the Fault state for VREF, Band Gap, or AVDD BOR.

Remarks

None.

Preconditions

The ADCHS module should be enabled before calling this function. The ADCHS module can be enabled by calling the PLIB_ADCHS_Enable function.

Example

```
bool vrefFaultState = PLIB_ADCHS_VREFFaultHasOccurred(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance

Function

bool PLIB_ADCHS_VREFFaultHasOccurred(ADCHS_MODULE_ID index)

PLIB_ADCHS_VREFFaultInterruptDisable Function

Disables the VREF Fault interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_VREFFaultInterruptDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the VREF Fault interrupt.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable VREF fault interrupt
PLIB_ADCHS_VREFFaultInterruptDisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_VREFFaultInterruptDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_VREFFaultInterruptEnable Function

Enables the VREF fault interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_VREFFaultInterruptEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the VREF fault interrupt. The interrupt is generated when a fault is encountered with VREF (mostly by BOR of the analog supply).

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable VREF fault interrupt
PLIB_ADCHS_VREFFaultInterruptEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_VREFFaultInterruptEnable(ADCHS_MODULE_ID index)

PLIB_ADCHS_VREFIsReady Function

Returns the state of VREF.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_VREFIsReady(ADCHS_MODULE_ID index);
```

Returns

- true - Both band gap reference voltage and ADC VREF are ready
- false - Either band gap reference voltage or ADC VREF is not ready

Description

This function returns the VREF state.

Remarks

None.

Preconditions

The ADCHS module should be enabled before checking for VREF status. The ADCHS module can be enabled by calling the PLIB_ADCHS_Enable function.

Example

```
bool vrefState = PLIB_ADCHS_VREFIsReady(MY_ADCHS_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance

Function

bool PLIB_ADCHS_VREFIsReady(ADCHS_MODULE_ID index)

PLIB_ADCHS_VREFReadyInterruptDisable Function

Disables the VREF ready interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_VREFReadyInterruptDisable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the VREF ready interrupt.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Disable VREF interrupt
PLIB_ADCHS_VREFReadyInterruptDisable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_VREFReadyInterruptDisable(ADCHS_MODULE_ID index)

PLIB_ADCHS_VREFReadyInterruptEnable Function

Enables the VREF ready interrupt.

File

plib_adchs.h

C

```
void PLIB_ADCHS_VREFReadyInterruptEnable(ADCHS_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the VREF ready interrupt. The interrupt is generated when the Band gap voltage/VREF is ready to be used by the ADCHS instance.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCHS_INSTANCE, is the ADCHS instance selected for use by the
// application developer.

// Enable VREF ready interrupt
PLIB_ADCHS_VREFReadyInterruptEnable( MY_ADCHS_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the ADCHS instance to be configured.

Function

void PLIB_ADCHS_VREFReadyInterruptEnable(ADCHS_MODULE_ID index)

m) Feature Existence Functions

PLIB_ADCHS_ExistsAnalogInputCheck Function

Identifies whether the System Configuration feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsAnalogInputCheck(ADCHS_MODULE_ID index);
```

Returns

Existence of the System Configuration feature:

- true - The System Configuration feature is supported on the device
- false - The System Configuration feature is not supported on the device

Description

This function identifies whether the System Configuration feature is available on the ADCHS module. When this function returns true, this function is supported on the device:

- PLIB_ADCHS_AnalogInputIsAvailable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsAnalogInputCheck(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsAnalogInputModeControl Function

Identifies whether the analog input mode control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsAnalogInputModeControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the input mode control feature:

- true - The analog input mode control feature is supported on the device
- false - The analog input mode control feature is not supported on the device

Description

This function identifies whether the analog input mode control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_AnalogInputModeSelect
- PLIB_ADCHS_AnalogInputModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsAnalogInputModeControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsAnalogInputScan Function

Identifies whether the Analog input Scan exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsAnalogInputScan(ADCHS_MODULE_ID index);
```

Returns

Existence of the Scan feature:

- true - The Analog Input Scan feature is supported on the device
- false - The Analog Input Scan feature is not supported on the device

Description

This function identifies whether the Analog Input Scan feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_AnalogInputScanIsComplete
- PLIB_ADCHS_AnalogInputScanSelect
- PLIB_ADCHS_AnalogInputScanRemove
- PLIB_ADCHS_AnalogInputScanIsSelected
- PLIB_ADCHS_AnalogInputScanSetup
- PLIB_ADCHS_ScanCompleteInterruptEnable
- PLIB_ADCHS_ScanCompleteInterruptDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsAnalogInputScan(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsChannelAnalogControl Function

Identifies whether the Channel Analog control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsChannelAnalogControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Channel Analog control feature:

- true - The Channel Analog control feature is supported on the device
- false - The Channel Analog control feature is not supported on the device

Description

This function identifies whether the Channel Analog control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_ChannelAnalogFeatureEnable
- PLIB_ADCHS_ChannelAnalogFeatureDisable
- PLIB_ADCHS_ChannelIsReady
- PLIB_ADCHS_ChannelIsReadyInterruptEnable
- PLIB_ADCHS_ChannelIsReadyInterruptDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsChannelAnalogControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsChannelConfiguration Function

Identifies whether the Channel Configuration feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsChannelConfiguration(ADCHS_MODULE_ID index);
```

Returns

Existence of the Channel Configuration feature:

- true - The Channel Configuration feature is supported on the device
- false - The Channel Configuration feature is not supported on the device

Description

This function identifies whether the Channel Configuration feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_ChannelConfigurationGet
- PLIB_ADCHS_ChannelConfigurationSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsChannelConfiguration(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsChannelDigitalControl Function

Identifies whether the Channel Digital control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsChannelDigitalControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Channel Digital control feature:

- true - The Channel Digital control feature is supported on the device
- false - The Channel Digital control feature is not supported on the device

Description

This function identifies whether the Channel Digital control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_ChannelDigitalFeatureEnable
- PLIB_ADCHS_ChannelDigitalFeatureDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsChannelDigitalControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsChannelInputSelectControl Function

Identifies whether the Channel 0 to 6 Input select feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsChannelInputSelectControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Channel 0 to 6 Input feature:

- true - The Channel 0 to 6 Input Select feature is supported on the device
- false - The Channel 0 to 6 Input Select feature is not supported on the device

Description

This function identifies whether the Channel 0 to 6 Input select feature is available on the ADCHS module. When this function returns true, this function is supported on the device:

- PLIB_ADCHS_ChannelInputSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCHS_ExistsChannelInputSelectControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsConfiguration Function

Identifies whether the Configuration feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsConfiguration(ADCHS_MODULE_ID index);
```

Returns

Existence of the Configuration feature:

- true - The Configuration feature is supported on the device
- false - The Configuration feature is not supported on the device

Description

This function identifies whether the Configuration feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_Setup
- PLIB_ADCHS_ChannelSetup

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsConfiguration(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsConversionResults Function

Identifies whether the Conversion Results feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsConversionResults(ADCHS_MODULE_ID index);
```

Returns

Existence of the ConversionResults feature:

- true - The ConversionResults feature is supported on the device
- false - The ConversionResults feature is not supported on the device

Description

This function identifies whether the Conversion Results feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_AnalogInputDataReadyInterruptEnable
- PLIB_ADCHS_AnalogInputDataReadyInterruptDisable
- PLIB_ADCHS_AnalogInputDataIsReady
- PLIB_ADCHS_AnalogInputResultGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsConversionResults(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsCVD Function

Identifies whether the CVD exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsCVD(ADCHS_MODULE_ID index);
```

Returns

Existence of the CVD feature:

- true - The CVD feature is supported on the device
- false - The CVD feature is not supported on the device

Description

This function identifies whether the CVD feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_CVDEnable
- PLIB_ADCHS_CVDDisable
- PLIB_ADCHS_CVDSetup
- PLIB_ADCHS_CVDResultGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsCVD(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsDigitalComparator Function

Identifies whether the Digital Comparator feature exists on the ADCHS module .

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsDigitalComparator(ADCHS_MODULE_ID index);
```

Returns

Existence of the Digital Comparator feature:

- true - The Digital Comparator feature is supported on the device
- false - The Digital Comparator feature is not supported on the device

Description

This function identifies whether the Digital Comparator feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_DigitalComparatorAnalogInputSelect
- PLIB_ADCHS_DigitalComparatorAnalogInputGet
- PLIB_ADCHS_DigitalComparatorEnable
- PLIB_ADCHS_DigitalComparatorDisable
- PLIB_ADCHS_DigitalComparatorInterruptEnable
- PLIB_ADCHS_DigitalComparatorInterruptDisable
- PLIB_ADCHS_DigitalComparatorSetup
- PLIB_ADCHS_DigitalComparatorEventHasOccurred
- PLIB_ADCHS_DigitalComparatorLimitSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsDigitalComparator(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsDigitalFilter Function

Identifies whether the Digital Filter feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsDigitalFilter(ADCHS_MODULE_ID index);
```

Returns

Existence of the Digital Filter feature:

- true - The Digital Filter feature is supported on the device
- false - The Digital Filter feature is not supported on the device

Description

This function identifies whether the Digital Filter feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_DigitalFilterEnable
- PLIB_ADCHS_DigitalFilterDisable
- PLIB_ADCHS_DigitalFilterOversamplingModeSetup
- PLIB_ADCHS_DigitalFilterAveragingModeSetup
- PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect

- PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect
- PLIB_ADCHS_DigitalFilterDataIsReady
- PLIB_ADCHS_DigitalFilterDataGet
- PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable
- PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsDigitalFilter(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsDMA Function

Identifies whether the DMA exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsDMA(ADCHS_MODULE_ID index);
```

Returns

Existence of the DMA feature:

- true - The DMA feature is supported on the device
- false - The DMA feature is not supported on the device

Description

This function identifies whether the DMA feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_DMAEnable
- PLIB_ADCHS_DMADisable
- PLIB_ADCHS_DMASetup
- PLIB_ADCHS_DMASourceSelect
- PLIB_ADCHS_DMASourceRemove
- PLIB_ADCHS_DMABuffer_A_InterruptEnable
- PLIB_ADCHS_DMABuffer_A_InterruptDisable
- PLIB_ADCHS_DMABuffer_B_InterruptEnable
- PLIB_ADCHS_DMABuffer_B_InterruptDisable
- PLIB_ADCHS_DMABuffer_A_IsFull
- PLIB_ADCHS_DMABuffer_B_IsFull
- PLIB_ADCHS_DMAOverflowErrorHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsDMA(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsEarlyInterruptControl Function

Identifies whether the Early Interrupt control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsEarlyInterruptControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Early Interrupt control feature:

- true - The Early Interrupt control feature is supported on the device
- false - The Early Interrupt control feature is not supported on the device

Description

This function identifies whether the Early Interrupt control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_EarlyInterruptEnable
- PLIB_ADCHS_EarlyInterruptDisable
- PLIB_ADCHS_AnalogInputEarlyInterruptEnable
- PLIB_ADCHS_AnalogInputEarlyInterruptDisable
- PLIB_ADCHS_AnalogInputEarlyInterruptIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsEarlyInterruptControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the ADCHS module

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsEnableControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the EnableControl feature:

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_Enable
- PLIB_ADCHS_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsEnableControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsExternalConversionRequestControl Function

Identifies whether the External Convert feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsExternalConversionRequestControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the External Convert feature:

- true - The External Convert feature is supported on the device
- false - The External Convert feature is not supported on the device

Description

This function identifies whether the External Convert feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_ExternalConversionRequestEnable
- PLIB_ADCHS_ExternalConversionRequestDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsExternalConversionRequestControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsFIFO Function

Identifies whether the FIFO exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsFIFO(ADCHS_MODULE_ID index);
```

Returns

Existence of the FIFO feature:

- true - The FIFO feature is supported on the device

- false - The FIFO feature is not supported on the device

Description

This function identifies whether the FIFO feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_FIFORead
- PLIB_ADCHS_FIFODataIsAvailable
- PLIB_ADCHS_FIFODataReadyInterruptEnable
- PLIB_ADCHS_FIFODataReadyInterruptDisable
- PLIB_ADCHS_FIFOEnable
- PLIB_ADCHS_FIFODisable
- PLIB_ADCHS_FIFOSourceSelect
- PLIB_ADCHS_FIFODataCountGet
- PLIB_ADCHS_FIFOSourceGet
- PLIB_ADCHS_FIFOErrorHasOccurred
- PLIB_ADCHS_FIFODataIsNegative

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsFIFO(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsManualControl Function

Identifies whether the Manual control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsManualControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Manual control feature:

- true - The Manual control feature is supported on the device
- false - The Manual control feature is not supported on the device

Description

This function identifies whether the Manual control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_SoftwareSamplingStart
- PLIB_ADCHS_SoftwareSamplingStop
- PLIB_ADCHS_SoftwareConversionStart
- PLIB_ADCHS_SoftwareConversionInputSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsManualControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsTriggerControl Function

Identifies whether the Trigger control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsTriggerControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Trigger control feature:

- true - The Trigger control feature is supported on the device
- false - The Trigger control feature is not supported on the device

Description

This function identifies whether the Trigger control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_AnalogInputLevelTriggerSet
- PLIB_ADCHS_AnalogInputEdgeTriggerSet
- PLIB_ADCHS_AnalogInputTriggerSourceSelect
- PLIB_ADCHS_GlobalSoftwareTriggerEnable
- PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable
- PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable
- PLIB_ADCHS_TriggerSuspendEnable
- PLIB_ADCHS_TriggerSuspendDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsTriggerControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsTriggerSampleControl Function

Identifies whether the Trigger Sample control feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsTriggerSampleControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Trigger Sample control feature:

- true - The Trigger Sample control feature is supported on the device

- false - The Trigger Sample control feature is not supported on the device

Description

This function whether the Trigger Sample control feature exists on the ADCHS module. When this function returns true, this function is supported on the device:

- PLIB_ADCHS_ChannelTriggerSampleSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsTriggerSampleControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsTurboMode Function

Identifies whether the Turbo mode feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsTurboMode(ADCHS_MODULE_ID index);
```

Returns

Existence of the Turbo mode feature:

- true - The Turbo mode feature is supported on the device
- false - The Turbo mode feature is not supported on the device

Description

This function identifies whether the Turbo mode feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_TurboModeEnable
- PLIB_ADCHS_TurboModeDisable
- PLIB_ADCHS_TurboModeErrorHasOccurred
- PLIB_ADCHS_TurboModeChannelSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsTurboMode(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsUpdateReadyControl Function

Identifies whether the Update Ready feature exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsUpdateReadyControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the Update Ready feature:

- true - The Update Ready feature is supported on the device
- false - The Update Ready feature is not supported on the device

Description

This function identifies whether the Update Ready feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable
- PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable
- PLIB_ADCHS_ControlRegistersCanBeUpdated

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsUpdateReadyControl(ADCHS_MODULE_ID index)

PLIB_ADCHS_ExistsVREFControl Function

Identifies whether the VREF control exists on the ADCHS module.

File

plib_adchs.h

C

```
bool PLIB_ADCHS_ExistsVREFControl(ADCHS_MODULE_ID index);
```

Returns

Existence of the VREF control feature:

- true - The VREF control feature is supported on the device
- false - The VREF control feature is not supported on the device

Description

This function identifies whether the VREF control feature is available on the ADCHS module. When this function returns true, these functions are supported on the device:

- PLIB_ADCHS_VREFIsReady
- PLIB_ADCHS_VREFFaultHasOccurred
- PLIB_ADCHS_VREFReadyInterruptEnable
- PLIB_ADCHS_VREFReadyInterruptDisable
- PLIB_ADCHS_VREFFaultInterruptEnable
- PLIB_ADCHS_VREFFaultInterruptDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance.

Function

PLIB_ADCHS_ExistsVREFControl(ADCHS_MODULE_ID index)

n) Data Types and Constants

ADCHS_AN_INPUT_ID Enumeration

Type for identifying the available ADC Inputs

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_AN0,
  ADCHS_AN1,
  ADCHS_AN2,
  ADCHS_AN3,
  ADCHS_AN4,
  ADCHS_AN5,
  ADCHS_AN6,
  ADCHS_AN7,
  ADCHS_AN8,
  ADCHS_AN9,
  ADCHS_AN10,
  ADCHS_AN11,
  ADCHS_AN12,
  ADCHS_AN13,
  ADCHS_AN14,
  ADCHS_AN15,
  ADCHS_AN16,
  ADCHS_AN17,
  ADCHS_AN18,
  ADCHS_AN19,
  ADCHS_AN20,
  ADCHS_AN21,
  ADCHS_AN22,
  ADCHS_AN23,
  ADCHS_AN24,
  ADCHS_AN25,
  ADCHS_AN26,
  ADCHS_AN27,
  ADCHS_AN28,
  ADCHS_AN29,
  ADCHS_AN30,
  ADCHS_AN31,
  ADCHS_AN32,
  ADCHS_AN33,
  ADCHS_AN34,
  ADCHS_AN35,
  ADCHS_AN36,
  ADCHS_AN37,
  ADCHS_AN38,
  ADCHS_AN39,
  ADCHS_AN40,
  ADCHS_AN41,
  ADCHS_AN42,
  ADCHS_AN43,
  ADCHS_AN44,
  ADCHS_AN50,
```

```
    ADCHS_AN51,
    ADCHS_AN52,
    ADCHS_AN53,
    ADCHS_AN54,
    ADCHS_AN55,
    ADCHS_AN56,
    ADCHS_AN57,
    ADCHS_AN58,
    ADCHS_AN59,
    ADCHS_AN60,
    ADCHS_AN61,
    ADCHS_AN62,
    ADCHS_AN63
  } ADCHS_AN_INPUT_ID;
```

Members

Members	Description
ADCHS_AN0	Analog Input AN0
ADCHS_AN1	Analog Input AN1
ADCHS_AN2	Analog Input AN2
ADCHS_AN3	Analog Input AN3
ADCHS_AN4	Analog Input AN4
ADCHS_AN5	Analog Input AN5
ADCHS_AN6	Analog Input AN6
ADCHS_AN7	Analog Input AN7
ADCHS_AN8	Analog Input AN8
ADCHS_AN9	Analog Input AN9
ADCHS_AN10	Analog Input AN10
ADCHS_AN11	Analog Input AN11
ADCHS_AN12	Analog Input AN12
ADCHS_AN13	Analog Input AN13
ADCHS_AN14	Analog Input AN14
ADCHS_AN15	Analog Input AN15
ADCHS_AN16	Analog Input AN16
ADCHS_AN17	Analog Input AN17
ADCHS_AN18	Analog Input AN18
ADCHS_AN19	Analog Input AN19
ADCHS_AN20	Analog Input AN20
ADCHS_AN21	Analog Input AN21
ADCHS_AN22	Analog Input AN22
ADCHS_AN23	Analog Input AN23
ADCHS_AN24	Analog Input AN24
ADCHS_AN25	Analog Input AN25
ADCHS_AN26	Analog Input AN26
ADCHS_AN27	Analog Input AN27
ADCHS_AN28	Analog Input AN28
ADCHS_AN29	Analog Input AN29
ADCHS_AN30	Analog Input AN30
ADCHS_AN31	Analog Input AN31
ADCHS_AN32	Analog Input AN32
ADCHS_AN33	Analog Input AN33
ADCHS_AN34	Analog Input AN34
ADCHS_AN35	Analog Input AN35
ADCHS_AN36	Analog Input AN36
ADCHS_AN37	Analog Input AN37
ADCHS_AN38	Analog Input AN38
ADCHS_AN39	Analog Input AN39
ADCHS_AN40	Analog Input AN40
ADCHS_AN41	Analog Input AN41

ADCHS_AN42	Analog Input AN42
ADCHS_AN43	Analog Input AN43
ADCHS_AN44	Analog Input AN44
ADCHS_AN50	Analog Input AN50
ADCHS_AN51	Analog Input AN51
ADCHS_AN52	Analog Input AN52
ADCHS_AN53	Analog Input AN53
ADCHS_AN54	Analog Input AN54
ADCHS_AN55	Analog Input AN55
ADCHS_AN56	Analog Input AN56
ADCHS_AN57	Analog Input AN57
ADCHS_AN58	Analog Input AN58
ADCHS_AN59	Analog Input AN59
ADCHS_AN60	Analog Input AN60
ADCHS_AN61	Analog Input AN61
ADCHS_AN62	Analog Input AN62
ADCHS_AN63	Analog Input AN63

Description

ADC Analog Input ID

Type for identifying the available ADC Inputs, regardless of Class.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_CHARGEPUMP_MODE Enumeration

Defines the selection for the charge pump.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_CHARGEPUMP_DISABLE,
  ADCHS_CHARGEPUMP_ENABLE
} ADCHS_CHARGEPUMP_MODE;
```

Members

Members	Description
ADCHS_CHARGEPUMP_DISABLE	Charge pump is disabled
ADCHS_CHARGEPUMP_ENABLE	Charge pump is enabled

Description

ADCHS Charge pump setting

This data type defines the state of the charge pump as either enabled or disabled.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_CLOCK_SOURCE Enumeration

Defines the ADCHS Clock Source Select.

File

help_plib_adchs.h

C

```
typedef enum {
```

```
  ADCHS_CLOCK_SOURCE_PBCLK,
  ADCHS_CLOCK_SOURCE_FRC,
  ADCHS_CLOCK_SOURCE_RFCLK3,
  ADCHS_CLOCK_SOURCE_SYSCLK
} ADCHS_CLOCK_SOURCE;
```

Members

Members	Description
ADCHS_CLOCK_SOURCE_PBCLK	TAD clock set to peripheral bus clock (PBCLK)
ADCHS_CLOCK_SOURCE_FRC	TAD clock set to FRC
ADCHS_CLOCK_SOURCE_RFCLK3	TAD clock set to REFCLK3
ADCHS_CLOCK_SOURCE_SYSCLK	TAD clock set to SYSCLK (TCY)

Description

ADCHS Clock source selection

This enumeration data type defines the ADCHS Clock Source Select.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_CVD_CAPACITOR Enumeration

Defines the value of the internal capacitor during CVD mode.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_CVD_CAPACITOR_0PF,
  ADCHS_CVD_CAPACITOR_2_5PF,
  ADCHS_CVD_CAPACITOR_5PF,
  ADCHS_CVD_CAPACITOR_7_5PF,
  ADCHS_CVD_CAPACITOR_10PF,
  ADCHS_CVD_CAPACITOR_12_5PF,
  ADCHS_CVD_CAPACITOR_15PF,
  ADCHS_CVD_CAPACITOR_17_5PF
} ADCHS_CVD_CAPACITOR;
```

Members

Members	Description
ADCHS_CVD_CAPACITOR_0PF	While in CVD mode, internal capacitor is 0pF
ADCHS_CVD_CAPACITOR_2_5PF	While in CVD mode, internal capacitor is 2.5pF
ADCHS_CVD_CAPACITOR_5PF	While in CVD mode, internal capacitor is 5pF
ADCHS_CVD_CAPACITOR_7_5PF	While in CVD mode, internal capacitor is 7.5pF
ADCHS_CVD_CAPACITOR_10PF	While in CVD mode, internal capacitor is 10pF
ADCHS_CVD_CAPACITOR_12_5PF	While in CVD mode, internal capacitor is 12.5pF
ADCHS_CVD_CAPACITOR_15PF	While in CVD mode, internal capacitor is 15pF
ADCHS_CVD_CAPACITOR_17_5PF	While in CVD mode, internal capacitor is 17.5pF

Description

ADCHS CVD capacitor selection

This data type defines the value of the internal capacitor during CVD mode.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DATA_RESOLUTION Enumeration

Identifies the resolution of the ADC output.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DATA_RESOLUTION_6BIT,
  ADCHS_DATA_RESOLUTION_8BIT,
  ADCHS_DATA_RESOLUTION_10BIT,
  ADCHS_DATA_RESOLUTION_12BIT
} ADCHS_DATA_RESOLUTION;
```

Members

Members	Description
ADCHS_DATA_RESOLUTION_6BIT	ADC Output Data resolution is 6 bits
ADCHS_DATA_RESOLUTION_8BIT	ADC Output Data resolution is 8 bits
ADCHS_DATA_RESOLUTION_10BIT	ADC Output Data resolution is 10 bits
ADCHS_DATA_RESOLUTION_12BIT	ADC Output Data resolution is 12 bits

Description

ADC Data Resolution

This data type is used to identify the resolution of the ADC output.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DIGITAL_COMPARATOR_ID Enumeration

Identifies the supported Digital Comparators.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DIGITAL_COMPARATOR_1,
  ADCHS_DIGITAL_COMPARATOR_2,
  ADCHS_DIGITAL_COMPARATOR_3,
  ADCHS_DIGITAL_COMPARATOR_4,
  ADCHS_DIGITAL_COMPARATOR_5,
  ADCHS_DIGITAL_COMPARATOR_6,
  ADCHS_NUMBER_OF_DIGITAL_COMPARATOR
} ADCHS_DIGITAL_COMPARATOR_ID;
```

Members

Members	Description
ADCHS_DIGITAL_COMPARATOR_1	Digital Comparator 1
ADCHS_DIGITAL_COMPARATOR_2	Digital Comparator 2
ADCHS_DIGITAL_COMPARATOR_3	Digital Comparator 3
ADCHS_DIGITAL_COMPARATOR_4	Digital Comparator 4
ADCHS_DIGITAL_COMPARATOR_5	Digital Comparator 5
ADCHS_DIGITAL_COMPARATOR_6	Digital Comparator 6
ADCHS_NUMBER_OF_DIGITAL_COMPARATOR	Number of Digital Comparators

Description

ADCHS Digital Comparator

This enumeration identifies all supported Digital Comparators for this ADCHS channel.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT Enumeration

Identifies the Digital Filter averaging sample count.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_2,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_4,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_8,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_16,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_32,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_64,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_128,
  ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_256
} ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT;
```

Members

Members	Description
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_2	Averaging is performed on 2 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_4	Averaging is performed on 4 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_8	Averaging is performed on 8 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_16	Averaging is performed on 16 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_32	Averaging is performed on 32 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_64	Averaging is performed on 64 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_128	Averaging is performed on 128 samples
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT_256	Averaging is performed on 256 samples

Description

ADCHS Averaging sample

This enumeration identifies all supported digital Filter averaging sample count for this ADCHS channel. Once, the set number of samples are acquired by ADC channel, it starts the averaging process.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DIGITAL_FILTER_ID Enumeration

Identifies the supported Digital Filters.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DIGITAL_FILTER_1,
  ADCHS_DIGITAL_FILTER_2,
  ADCHS_DIGITAL_FILTER_3,
  ADCHS_DIGITAL_FILTER_4,
  ADCHS_DIGITAL_FILTER_5,
  ADCHS_DIGITAL_FILTER_6,
  ADCHS_NUMBER_OF_DIGITAL_FILTER
} ADCHS_DIGITAL_FILTER_ID;
```

Members

Members	Description
ADCHS_DIGITAL_FILTER_1	Digital Filter 1
ADCHS_DIGITAL_FILTER_2	Digital Filter 2
ADCHS_DIGITAL_FILTER_3	Digital Filter 3
ADCHS_DIGITAL_FILTER_4	Digital Filter 4

ADCHS_DIGITAL_FILTER_5	Digital Filter 5
ADCHS_DIGITAL_FILTER_6	Digital Filter 6
ADCHS_NUMBER_OF_DIGITAL_FILTER	Number of Digital Filters

Description

ADCHS Digital Filter

This enumeration identifies all supported digital Filters for this ADCHS channel.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO Enumeration

Identifies the supported Digital Filter oversampling ratios.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_4X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_16X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_64X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_256X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_2X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_8X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_32X,
  ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_128X
} ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO;
```

Members

Members	Description
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_4X	4x oversampling, shift sum 1 bit to right, output data is 13 bits
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_16X	16x oversampling, shift sum 2 bits to right, output data is 14 bits
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_64X	64x oversampling, shift sum 3 bits to right, output data is 15 bits
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_256X	256x oversampling, shift sum 4 bits to right, output data is 16 bits
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_2X	2x oversampling, shift sum 0 bits to right, output data is in 12.1 format
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_8X	8x oversampling, shift sum 1 bit to right, output data is in 13.1 format
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_32X	32x oversampling, shift sum 2 bits to right, output data is in 14.1 format
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO_128X	128x oversampling, shift sum 3 bit to right, output data is in 15.1 format

Description

ADCHS Oversampling Ratio

This enumeration identifies all supported digital Filter oversampling ratios for this ADCHS channel. Oversampling ratios determine the number of samples used to generate a single output and the resulting resolution and format.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS Enumeration

Data length of digital filter.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DIGITAL_FILTER_SIGNIFICANT_FIRST_12BITS,
  ADCHS_DIGITAL_FILTER_SIGNIFICANT_ALL_16BITS
} ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS;
```

Members

Members	Description
ADCHS_DIGITAL_FILTER_SIGNIFICANT_FIRST_12BITS	Output of digital filter has only first 12 bits significant (remaining 4 zeros)
ADCHS_DIGITAL_FILTER_SIGNIFICANT_ALL_16BITS	Output of digital filter has all 16 bits significant

Description

ADCHS Digital filter data length

The output of digital filter can be set as all 16-bit to be significant or only first 12-bit significant followed by 4 zeros.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DMA_BUFFER_LENGTH Enumeration

Defines the length of the DMA buffer length.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DMA_BUFFER_LENGTH_2BYTES,
  ADCHS_DMA_BUFFER_LENGTH_4BYTES,
  ADCHS_DMA_BUFFER_LENGTH_8BYTES,
  ADCHS_DMA_BUFFER_LENGTH_16BYTES,
  ADCHS_DMA_BUFFER_LENGTH_32BYTES,
  ADCHS_DMA_BUFFER_LENGTH_64BYTES,
  ADCHS_DMA_BUFFER_LENGTH_128BYTES,
  ADCHS_DMA_BUFFER_LENGTH_256BYTES
} ADCHS_DMA_BUFFER_LENGTH;
```

Members

Members	Description
ADCHS_DMA_BUFFER_LENGTH_2BYTES	Allocate 2 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_4BYTES	Allocate 4 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_8BYTES	Allocate 8 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_16BYTES	Allocate 16 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_32BYTES	Allocate 32 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_64BYTES	Allocate 64 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_128BYTES	Allocate 128 bytes in RAM for each analog input
ADCHS_DMA_BUFFER_LENGTH_256BYTES	Allocate 256 bytes in RAM for each analog input

Description

ADCHS Class-1 DMA buffer length

This data type defines the length of the DMA buffer (in bytes).

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_DMA_COUNT Enumeration

Defines the enable/disable of the count feature for DMA.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_DMA_COUNT_DISABLE,
  ADCHS_DMA_COUNT_ENABLE
} ADCHS_DMA_COUNT;
```

Members

Members	Description
ADCHS_DMA_COUNT_DISABLE	DMA does not keep count of number of data saved by DMA
ADCHS_DMA_COUNT_ENABLE	DMA keeps count of number of data saved by DMA

Description

ADCHS DMA count enable

This data type defines whether or not the DMA should count the number of samples and store it in DMA RAM location.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK Enumeration

Defines the number of clocks prior to the arrival of valid data that the associated interrupt is generated.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_2,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_3,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_4,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_5,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_6,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_7,
  ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_8
} ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK;
```

Members

Members	Description
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1	The data ready interrupt is generated 1 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_2	The data ready interrupt is generated 2 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_3	The data ready interrupt is generated 3 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_4	The data ready interrupt is generated 4 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_5	The data ready interrupt is generated 5 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_6	The data ready interrupt is generated 6 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_7	The data ready interrupt is generated 7 ADC clock prior to end of conversion
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_8	The data ready interrupt is generated 8 ADC clock prior to end of conversion

Description

ADCHS Early interrupt prior clock selection

This data type defines the number of clocks prior to the arrival of valid data that the associated interrupt is generated. All options are available when the selected resolution for converted data is 12-bit or 10-bit. For a resolution of 8-bit, options from ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1 to ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_6 are valid. For a resolution of 6-bit, options from ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_1 to ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK_4 are valid.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_FAST_SYNC_PERIPHERAL_CLOCK Enumeration

Defines the selection of the fast synchronous peripheral clock.

File

help_plib_adchs.h

C

```
typedef enum {
```

```
  ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE,
  ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_ENABLE
} ADCHS_FAST_SYNC_PERIPHERAL_CLOCK;
```

Members

Members	Description
ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_DISABLE	Peripheral clock is not used for ADCHS
ADCHS_FAST_SYNC_PERIPHERAL_CLOCK_ENABLE	Peripheral clock is used for ADCHS

Description

ADCHS Fast synchronous peripheral clock mode

This data type defines whether the fast synchronous peripheral clock to the ADCHS channel is enabled or disabled.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_FAST_SYNC_SYSTEM_CLOCK Enumeration

Defines the selection of the fast synchronous system clock.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_FAST_SYNC_SYSTEM_CLOCK_DISABLE,
  ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE
} ADCHS_FAST_SYNC_SYSTEM_CLOCK;
```

Members

Members	Description
ADCHS_FAST_SYNC_SYSTEM_CLOCK_DISABLE	System clock is not used for ADCHS
ADCHS_FAST_SYNC_SYSTEM_CLOCK_ENABLE	System clock is used for ADCHS

Description

ADCHS Fast synchronous system clock mode

This data type defines whether the fast synchronous system clock to ADCHS channel is enabled or disabled.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_INPUT_MODE Enumeration

Defines the available modes for the selected input.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR,
  ADCHS_INPUT_MODE_SINGLE_ENDED_TWOS_COMP,
  ADCHS_INPUT_MODE_DIFFERENTIAL_UNIPOLAR,
  ADCHS_INPUT_MODE_DIFFERENTIAL_TWOS_COMP
} ADCHS_INPUT_MODE;
```

Members

Members	Description
ADCHS_INPUT_MODE_SINGLE_ENDED_UNIPOLAR	Single-ended input, Unipolar encoded
ADCHS_INPUT_MODE_SINGLE_ENDED_TWOS_COMP	Single-ended input, two's compliment encoded
ADCHS_INPUT_MODE_DIFFERENTIAL_UNIPOLAR	Differential input, Unipolar encoded
ADCHS_INPUT_MODE_DIFFERENTIAL_TWOS_COMP	Differential input, two's compliment encoded

Description

ADCHS Input Mode

This data type defines the available modes for the selected input.

Remarks

None.

ADCHS_INTERRUPT_BIT_SHIFT_LEFT Enumeration

Identifies the bits shift for calculating the interrupt vector.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_1_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_2_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_3_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_4_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_5_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_6_BITS,
  ADCHS_INTERRUPT_BIT_SHIFT_LEFT_7_BITS
} ADCHS_INTERRUPT_BIT_SHIFT_LEFT;
```

Members

Members	Description
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_0_BITS	Bit shift by 0 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_1_BITS	Bit shift by 1 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_2_BITS	Bit shift by 2 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_3_BITS	Bit shift by 3 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_4_BITS	Bit shift by 4 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_5_BITS	Bit shift by 5 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_6_BITS	Bit shift by 6 bits
ADCHS_INTERRUPT_BIT_SHIFT_LEFT_7_BITS	Bit shift by 7 bits

Description

ADCHS Interrupt vector shift bits

Interrupt vector address, is calculated by (Base_Address + x << vector_shift), where 'x' is the smallest active channel ID from the status register (AD1DSTAT1 or 2) registers (which has highest priority).

This enumeration identifies all the possible bits shift.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_OUTPUT_DATA_FORMAT Enumeration

Defines the selection for the output data format.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_OUTPUT_DATA_FORMAT_INTEGER,
  ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL
} ADCHS_OUTPUT_DATA_FORMAT;
```

Members

Members	Description
ADCHS_OUTPUT_DATA_FORMAT_INTEGER	Output data format is integer
ADCHS_OUTPUT_DATA_FORMAT_FRACTIONAL	Output data format is fractional

Description

ADCHS output data format

The output of ADC can be specified as either integer or fractional. This enum defines the setting of the output data format.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_SCAN_TRIGGER_SENSITIVE Enumeration

Trigger level for scan trigger.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_SCAN_TRIGGER_SENSITIVE_EDGE,
  ADCHS_SCAN_TRIGGER_SENSITIVE_LEVEL
} ADCHS_SCAN_TRIGGER_SENSITIVE;
```

Members

Members	Description
ADCHS_SCAN_TRIGGER_SENSITIVE_EDGE	Scan trigger is edge sensitive
ADCHS_SCAN_TRIGGER_SENSITIVE_LEVEL	Scan trigger is level sensitive

Description

ADCHS scan trigger level

The scan trigger can be configured to be sensitive to level or sensitive to edge. This data type defines the different configurations.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_SCAN_TRIGGER_SOURCE Enumeration

Defines the ADCHS Channel Scan Trigger Source Selections.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_SCAN_TRIGGER_SOURCE_NONE,
  ADCHS_SCAN_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE,
  ADCHS_SCAN_TRIGGER_SOURCE_SOFTWARE_LEVEL,
  ADCHS_SCAN_TRIGGER_SOURCE_INT0,
  ADCHS_SCAN_TRIGGER_SOURCE_TMR1_MATCH,
  ADCHS_SCAN_TRIGGER_SOURCE_TMR3_MATCH,
  ADCHS_SCAN_TRIGGER_SOURCE_TMR5_MATCH,
  ADCHS_SCAN_TRIGGER_SOURCE_OC1,
  ADCHS_SCAN_TRIGGER_SOURCE_OC3,
  ADCHS_SCAN_TRIGGER_SOURCE_OC5,
  ADCHS_SCAN_TRIGGER_SOURCE_COUT1,
  ADCHS_SCAN_TRIGGER_SOURCE_COUT2
} ADCHS_SCAN_TRIGGER_SOURCE;
```

Members

Members	Description
ADCHS_SCAN_TRIGGER_SOURCE_NONE	No scan trigger source is selected
ADCHS_SCAN_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE	Global Software edge trigger selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_SOFTWARE_LEVEL	Level Software trigger selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_INT0	Interrupt 0 (INT0) selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_TMR1_MATCH	Timer 1 Match (TMR1) selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_TMR3_MATCH	Timer 3 Match (TMR3) selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_TMR5_MATCH	Timer 5 Match (TMR5) selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_OC1	Output Compare 1(OC1) period end selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_OC3	Output Compare 3 (OC3) period end selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_OC5	Output Compare 5 (OC5) period end selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_COUT1	Comparator Output 1 selected as scan trigger source
ADCHS_SCAN_TRIGGER_SOURCE_COUT2	Comparator Output 2 selected as scan trigger source

Description

ADCHS Channel Scan Trigger Source Select

This data type defines the ADCHS Channel Scan Trigger Source Selections.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_TRIGGER_SOURCE Enumeration

Defines the ADCHS Trigger Source Selections.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_TRIGGER_SOURCE_NONE,
  ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE,
  ADCHS_TRIGGER_SOURCE_SOFTWARE_LEVEL,
  ADCHS_TRIGGER_SOURCE_STRIG,
  ADCHS_TRIGGER_SOURCE_INT0,
  ADCHS_TRIGGER_SOURCE_TMR1_MATCH,
  ADCHS_TRIGGER_SOURCE_TMR3_MATCH,
  ADCHS_TRIGGER_SOURCE_TMR5_MATCH,
  ADCHS_TRIGGER_SOURCE_OC1,
  ADCHS_TRIGGER_SOURCE_OC3,
  ADCHS_TRIGGER_SOURCE_OC5,
  ADCHS_TRIGGER_SOURCE_COUT1,
  ADCHS_TRIGGER_SOURCE_COUT2
} ADCHS_TRIGGER_SOURCE;
```

Members

Members	Description
ADCHS_TRIGGER_SOURCE_NONE	No trigger source is selected
ADCHS_TRIGGER_SOURCE_GLOBAL_SOFTWARE_EDGE	Global Software edge trigger selected as trigger source
ADCHS_TRIGGER_SOURCE_SOFTWARE_LEVEL	Level Software trigger selected as trigger source
ADCHS_TRIGGER_SOURCE_STRIG	Scan trigger source
ADCHS_TRIGGER_SOURCE_INT0	Interrupt 0 (INT0) selected as trigger source
ADCHS_TRIGGER_SOURCE_TMR1_MATCH	Timer 1 Match (TMR1) selected as trigger source
ADCHS_TRIGGER_SOURCE_TMR3_MATCH	Timer 3 Match (TMR3) selected as trigger source
ADCHS_TRIGGER_SOURCE_TMR5_MATCH	Timer 5 Match (TMR5) selected as trigger source
ADCHS_TRIGGER_SOURCE_OC1	Output Compare 1(OC1) period end selected as trigger source
ADCHS_TRIGGER_SOURCE_OC3	Output Compare 3 (OC3) period end selected as trigger source
ADCHS_TRIGGER_SOURCE_OC5	Output Compare 5 (OC5) period end selected as trigger source
ADCHS_TRIGGER_SOURCE_COUT1	Comparator Output 1 selected as trigger source

ADCHS_TRIGGER_SOURCE_COUT2	Comparator Output 2 selected as trigger source

Description

ADCHS Trigger Source Select

This data type defines the ADCHS Trigger Source Selections.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_VREF_SOURCE Enumeration

Defines the ADCHS VREF Source Select.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_VREF_AVDD_AVSS,
  ADCHS_VREF_EXTVREFP_AVSS,
  ADCHS_VREF_AVDD_EXTVREFN,
  ADCHS_VREF_EXTVREFP_EXTVREFN,
  ADCHS_VREF_INTVREFP_INTVREFN,
  ADCHS_VREF_INTVREFP_EXTVREFN,
  ADCHS_VREF_INTVREFP_AVSS,
  ADCHS_VREF_AVDD_INTVREFN
} ADCHS_VREF_SOURCE;
```

Members

Members	Description
ADCHS_VREF_AVDD_AVSS	Reference voltage set to AVDD and AVSS
ADCHS_VREF_EXTVREFP_AVSS	Reference voltage set to external VREF positive and AVSS
ADCHS_VREF_AVDD_EXTVREFN	Reference voltage set to AVDD and external VREF negative
ADCHS_VREF_EXTVREFP_EXTVREFN	Reference voltage set to external VREF positive and external VREF negative
ADCHS_VREF_INTVREFP_INTVREFN	Reference voltage set to internal VREF positive and internal VREF negative
ADCHS_VREF_INTVREFP_EXTVREFN	Reference voltage set to internal VREF positive and external VREF negative
ADCHS_VREF_INTVREFP_AVSS	Reference voltage set to internal VREF positive and AVSS
ADCHS_VREF_AVDD_INTVREFN	Reference voltage set to AVDD positive and internal VREF negative

Description

ADCHS VREF Source Select

This data type defines the ADCHS Reference Voltage (VREF) Source Select.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_WARMUP_CLOCK Enumeration

Identifies the number of clocks before the channel can be ready

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_WARMUP_CLOCK_16,
  ADCHS_WARMUP_CLOCK_32,
  ADCHS_WARMUP_CLOCK_64,
  ADCHS_WARMUP_CLOCK_128,
  ADCHS_WARMUP_CLOCK_256,
  ADCHS_WARMUP_CLOCK_512,
  ADCHS_WARMUP_CLOCK_1024,
  ADCHS_WARMUP_CLOCK_2048,
```

```
  ADCHS_WARMUP_CLOCK_4096,
  ADCHS_WARMUP_CLOCK_8192,
  ADCHS_WARMUP_CLOCK_16384,
  ADCHS_WARMUP_CLOCK_32768
} ADCHS_WARMUP_CLOCK;
```

Members

Members	Description
ADCHS_WARMUP_CLOCK_16	Warm-up time is 16 ADC clocks
ADCHS_WARMUP_CLOCK_32	Warm-up time is 32 ADC clocks
ADCHS_WARMUP_CLOCK_64	Warm-up time is 64 ADC clocks
ADCHS_WARMUP_CLOCK_128	Warm-up time is 128 ADC clocks
ADCHS_WARMUP_CLOCK_256	Warm-up time is 256 ADC clocks
ADCHS_WARMUP_CLOCK_512	Warm-up time is 512 ADC clocks
ADCHS_WARMUP_CLOCK_1024	Warm-up time is 1024 ADC clocks
ADCHS_WARMUP_CLOCK_2048	Warm-up time is 2048 ADC clocks
ADCHS_WARMUP_CLOCK_4096	Warm-up time is 4096 ADC clocks
ADCHS_WARMUP_CLOCK_8192	Warm-up time is 8192 ADC clocks
ADCHS_WARMUP_CLOCK_16384	Warm-up time is 16384 ADC clocks
ADCHS_WARMUP_CLOCK_32768	Warm-up time is 32768 ADC clocks

Description

ADCHS Wakeup warm-up clocks

When the analog section of an ADC channel is enabled, the channel needs a certain amount of warm-up time before it can be ready to perform conversion. This enumeration contains the various warm-up time in terms of the number of clocks.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_MODULE_ID Enumeration

Identifies the number of ADC supported.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_ID_0,
  ADCHS_NUMBER_OF_MODULES
} ADCHS_MODULE_ID;
```

Members

Members	Description
ADCHS_ID_0	ADC ID 0
ADCHS_NUMBER_OF_MODULES	Number of available ADC.

Description

This enumeration identifies the available ADC.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Refer to the specific device data sheet to determine how many ADC modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

ADCHS_CHANNEL_INP_SEL Enumeration

Defines the alternate input selection for channels 0 to 6.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_CHANNEL_0_DEFAULT_INP_AN0,
  ADCHS_CHANNEL_0_ALTERNATE_INP_AN45,
  ADCHS_CHANNEL_1_DEFAULT_INP_AN1,
  ADCHS_CHANNEL_1_ALTERNATE_INP_AN46,
  ADCHS_CHANNEL_2_DEFAULT_INP_AN2,
  ADCHS_CHANNEL_2_ALTERNATE_INP_AN47,
  ADCHS_CHANNEL_3_DEFAULT_INP_AN3,
  ADCHS_CHANNEL_3_ALTERNATE_INP_AN48,
  ADCHS_CHANNEL_4_DEFAULT_INP_AN4,
  ADCHS_CHANNEL_4_ALTERNATE_INP_AN49
} ADCHS_CHANNEL_INP_SEL;
```

Members

Members	Description
ADCHS_CHANNEL_0_DEFAULT_INP_AN0	AN0 is the default input for Channel 0
ADCHS_CHANNEL_0_ALTERNATE_INP_AN45	AN45 is the alternate input for Channel 0
ADCHS_CHANNEL_1_DEFAULT_INP_AN1	AN1 is the default input for Channel 1
ADCHS_CHANNEL_1_ALTERNATE_INP_AN46	AN46 is the alternate input for Channel 1
ADCHS_CHANNEL_2_DEFAULT_INP_AN2	AN2 is the default input for Channel 2
ADCHS_CHANNEL_2_ALTERNATE_INP_AN47	AN47 is the alternate input for Channel 2
ADCHS_CHANNEL_3_DEFAULT_INP_AN3	AN3 is the default input for Channel 3
ADCHS_CHANNEL_3_ALTERNATE_INP_AN48	AN48 is the alternate input for Channel 3
ADCHS_CHANNEL_4_DEFAULT_INP_AN4	AN4 is the default input for Channel 4
ADCHS_CHANNEL_4_ALTERNATE_INP_AN49	AN49 is the alternate input for Channel 4

Description

ADCHS Alternate input selection for channel- 0 to 6.

This data type defines the possible alternate input selections for channels 0 to 6. For channel 7, alternate selection does not exist. Passing the values for channel 7 will return an error.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_CHANNEL_ID Enumeration

Identifies the ADC channel from 0 to 7.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_CHANNEL_0,
  ADCHS_CHANNEL_1,
  ADCHS_CHANNEL_2,
  ADCHS_CHANNEL_3,
  ADCHS_CHANNEL_4,
  ADCHS_CHANNEL_5,
  ADCHS_CHANNEL_6,
  ADCHS_CHANNEL_7,
  ADCHS_NUMBER_OF_CHANNEL
} ADCHS_CHANNEL_ID;
```

Members

Members	Description
ADCHS_CHANNEL_0	ADCHS channel 0
ADCHS_CHANNEL_1	ADCHS channel 1
ADCHS_CHANNEL_2	ADCHS channel 2
ADCHS_CHANNEL_3	ADCHS channel 3

ADCHS_CHANNEL_4	ADCHS channel 4
ADCHS_CHANNEL_5	ADCHS channel 5
ADCHS_CHANNEL_6	ADCHS channel 6
ADCHS_CHANNEL_7	ADCHS channel 7
ADCHS_NUMBER_OF_CHANNEL	Number of channels

Description

ADCHS channel Select

This enumeration identifies the ADC channels from 0 to 7. Not all devices have all ADCHS channels. Refer to the specific device data sheet for the number of available channels. Some features that are available in channels 0 to 6 may not be available on channel 7.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL Enumeration

Defines the selection of trigger and sampling for channels 0 to 6.

File

help_plib_adchs.h

C

```
typedef enum {
  ADCHS_CHANNEL_UNSYNC_TRIGGER_UNSYNC_SAMPLING,
  ADCHS_CHANNEL_SYNC_SAMPLING,
  ADCHS_CHANNEL_SYNC_TRIGGER_UNSYNC_SAMPLING
} ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL;
```

Members

Members	Description
ADCHS_CHANNEL_UNSYNC_TRIGGER_UNSYNC_SAMPLING	Unsynchronized trigger and unsynchronous sampling
ADCHS_CHANNEL_SYNC_SAMPLING	Synchronous sampling (trigger selection is ignored)
ADCHS_CHANNEL_SYNC_TRIGGER_UNSYNC_SAMPLING	Synchronized trigger and unsynchronous sampling

Description

ADCHS channel trigger and sampling selection

This data type defines the selection of trigger and sampling for ADCHS channels 0 to 6. Passing this value for channel 7 will result in an error.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCHS_CLASS12_AN_INPUT_ID Enumeration

Type for identifying the available Class 1 and 2 analog Inputs.

File

help_plib_adchs.h

C

```
typedef enum {
} ADCHS_CLASS12_AN_INPUT_ID;
```

Description

ADC Analog Input ID of type class-1 and class-2

These enums are passed to functions meant to control or enable/disable "triggers". This is because only Class 1 and Class 2 analog inputs have individual triggers associated with them.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

Files

Files

Name	Description
help_plib_adchs.h	ADCHS Peripheral Library interface header file template.
plib_adchs.h	ADCHS Peripheral Library Interface Header for ADCHS common definitions.

Description

This section lists the source and header files used by the library.

help_plib_adchs.h

ADCHS Peripheral Library interface header file template.

Enumerations

	Name	Description
	ADCHS_AN_INPUT_ID	Type for identifying the available ADC Inputs
	ADCHS_CHANNEL_ID	Identifies the ADC channel from 0 to 7.
	ADCHS_CHANNEL_INP_SEL	Defines the alternate input selection for channels 0 to 6.
	ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL	Defines the selection of trigger and sampling for channels 0 to 6.
	ADCHS_CHARGEPUMP_MODE	Defines the selection for the charge pump.
	ADCHS_CLASS12_AN_INPUT_ID	Type for identifying the available Class 1 and 2 analog Inputs.
	ADCHS_CLOCK_SOURCE	Defines the ADCHS Clock Source Select.
	ADCHS_CVD_CAPACITOR	Defines the value of the internal capacitor during CVD mode.
	ADCHS_DATA_RESOLUTION	Identifies the resolution of the ADC output.
	ADCHS_DIGITAL_COMPARATOR_ID	Identifies the supported Digital Comparators.
	ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT	Identifies the Digital Filter averaging sample count.
	ADCHS_DIGITAL_FILTER_ID	Identifies the supported Digital Filters.
	ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO	Identifies the supported Digital Filter oversampling ratios.
	ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS	Data length of digital filter.
	ADCHS_DMA_BUFFER_LENGTH	Defines the length of the DMA buffer length.
	ADCHS_DMA_COUNT	Defines the enable/disable of the count feature for DMA.
	ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK	Defines the number of clocks prior to the arrival of valid data that the associated interrupt is generated.
	ADCHS_FAST_SYNC_PERIPHERAL_CLOCK	Defines the selection of the fast synchronous peripheral clock.
	ADCHS_FAST_SYNC_SYSTEM_CLOCK	Defines the selection of the fast synchronous system clock.
	ADCHS_INPUT_MODE	Defines the available modes for the selected input.
	ADCHS_INTERRUPT_BIT_SHIFT_LEFT	Identifies the bits shift for calculating the interrupt vector.
	ADCHS_MODULE_ID	Identifies the number of ADC supported.
	ADCHS_OUTPUT_DATA_FORMAT	Defines the selection for the output data format.
	ADCHS_SCAN_TRIGGER_SENSITIVE	Trigger level for scan trigger.
	ADCHS_SCAN_TRIGGER_SOURCE	Defines the ADCHS Channel Scan Trigger Source Selections.
	ADCHS_TRIGGER_SOURCE	Defines the ADCHS Trigger Source Selections.
	ADCHS_VREF_SOURCE	Defines the ADCHS VREF Source Select.
	ADCHS_WARMUP_CLOCK	Identifies the number of clocks before the channel can be ready

Description

ADCHS Peripheral Library Interface Header File Template

This file is provided for documentation purposes.

File Name

help_plib_adchs.h

Company

Microchip Technology Inc.

plib_adchs.h

ADCHS Peripheral Library Interface Header for ADCHS common definitions.

Functions

	Name	Description
	PLIB_ADCHS_AnalogInputDataIsReady	Returns the data ready status of analog inputs.
	PLIB_ADCHS_AnalogInputDataReadyInterruptDisable	Disables the data ready interrupt for the selected analog inputs.
	PLIB_ADCHS_AnalogInputDataReadyInterruptEnable	Enables the data ready interrupt for the selected analog input.
	PLIB_ADCHS_AnalogInputEarlyInterruptDisable	Disables the early interrupt for the analog input.
	PLIB_ADCHS_AnalogInputEarlyInterruptEnable	Enables the early interrupt for the analog input.
	PLIB_ADCHS_AnalogInputEarlyInterruptIsReady	Returns the early interrupt ready status of analog input.
	PLIB_ADCHS_AnalogInputEdgeTriggerSet	Sets the trigger as edge sensitive for selected class 1 and 2 analog input.
	PLIB_ADCHS_AnalogInputIsAvailable	Returns the analog input configuration of ADCHS channel.
	PLIB_ADCHS_AnalogInputLevelTriggerSet	Sets the trigger as level sensitive for selected Class 1 and 2 analog input.
	PLIB_ADCHS_AnalogInputModeGet	Returns the mode for the specified analog input.
	PLIB_ADCHS_AnalogInputModeSelect	Selects the mode for the specified analog input.
	PLIB_ADCHS_AnalogInputResultGet	Returns a ADC conversion result.
	PLIB_ADCHS_AnalogInputScanIsComplete	Returns the state of End of scan completion.
	PLIB_ADCHS_AnalogInputScanIsSelected	Returns whether or note an analog input is selected for scanning.
	PLIB_ADCHS_AnalogInputScanRemove	Removes the analog input from scanning selection.
	PLIB_ADCHS_AnalogInputScanSelect	Selects the analog input for scanning.
	PLIB_ADCHS_AnalogInputScanSetup	Selects input to include in Analog Input Scan mode.
	PLIB_ADCHS_AnalogInputTriggerSourceSelect	Selects a trigger Source for analog input (Class 1 or Class 2 analog inputs only).
	PLIB_ADCHS_ChannelAnalogFeatureDisable	Disables the analog circuit for channels of High-Speed SAR ADC.
	PLIB_ADCHS_ChannelAnalogFeatureEnable	Enables the analog circuit for High-Speed SAR ADC channels.
	PLIB_ADCHS_ChannelConfigurationGet	Used to get the configuration for the specified channel.
	PLIB_ADCHS_ChannelConfigurationSet	Used to set the configuration for the specified channel.
	PLIB_ADCHS_ChannelDigitalFeatureDisable	Disables the digital circuit for channels of High-Speed SAR ADC.
	PLIB_ADCHS_ChannelDigitalFeatureEnable	Enables (turns ON) the digital circuit for channels.
	PLIB_ADCHS_ChannelInputSelect	Selects the analog input for Channel 0 to 6.
	PLIB_ADCHS_ChannelIsReady	Returns the state of the channel.
	PLIB_ADCHS_ChannelIsReadyInterruptDisable	Disables the Channel ready interrupt for the specified channel.
	PLIB_ADCHS_ChannelIsReadyInterruptEnable	Enables the Channel ready interrupt for the specified channel.
	PLIB_ADCHS_ChannelSetup	Configures the High-Speed SAR ADC channels.
	PLIB_ADCHS_ChannelTriggerSampleSelect	Selects the trigger and sampling modes for channels of High-Speed SAR ADC
	PLIB_ADCHS_ControlRegistersCanBeUpdated	Returns the status of update-ready.
	PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable	Disables the update-ready interrupt.
	PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable	Enables the update-ready interrupt.
	PLIB_ADCHS_CVDDisable	Disables the CVD feature.
	PLIB_ADCHS_CVDEnable	Enables the CVD feature.
	PLIB_ADCHS_CVDResultGet	Returns a CVD result measured by an ADCHS instance.
	PLIB_ADCHS_CVDSetup	Configures the CVD related setting of ADCHS channel.
	PLIB_ADCHS_DigitalComparatorAnalogInputGet	Returns the analog input ID used by the digital comparator.
	PLIB_ADCHS_DigitalComparatorAnalogInputSelect	Selects analog inputs, whose converted data will be processed by the comparator.
	PLIB_ADCHS_DigitalComparatorDisable	Disables the specified digital comparator.
	PLIB_ADCHS_DigitalComparatorEnable	Enables the specified digital comparator.
	PLIB_ADCHS_DigitalComparatorEventHasOccurred	Returns the status of the selected digital comparator.
	PLIB_ADCHS_DigitalComparatorInterruptDisable	Disables the interrupt for the selected digital comparator.
	PLIB_ADCHS_DigitalComparatorInterruptEnable	Enables the interrupt for the selected digital comparator.
	PLIB_ADCHS_DigitalComparatorLimitSet	Sets the limit for the specified digital comparator.

	PLIB_ADCHS_DigitalComparatorSetup	Configures the Digital Comparator on the High-Speed SAR ADC converter.
	PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect	Selects the number of samples which are averaged by the Digital Filter.
	PLIB_ADCHS_DigitalFilterAveragingModeSetup	Configures the Digital Filter on the High-Speed SAR ADC converter in Averaging mode.
	PLIB_ADCHS_DigitalFilterDataGet	Used to fetch the data result from the Digital Filter.
	PLIB_ADCHS_DigitalFilterDataIsReady	Used to determine if the Digital Filter has data ready.
	PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable	Disables the interrupt for the selected Digital Filter.
	PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable	Enables the interrupt for the selected Digital Filter.
	PLIB_ADCHS_DigitalFilterDisable	Disables the Digital Filter.
	PLIB_ADCHS_DigitalFilterEnable	Enables the Digital Filter.
	PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect	Selects the oversampling ratio for the Digital Filter.
	PLIB_ADCHS_DigitalFilterOversamplingModeSetup	Configures the Digital Filter on the High-Speed SAR ADC converter in Oversampling mode.
	PLIB_ADCHS_Disable	High-Speed SAR ADC module is disabled (turned OFF).
	PLIB_ADCHS_DMABuffer_A_InterruptDisable	Disables the DMA Buffer A full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_A_InterruptEnable	Enables the DMA Buffer A full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_A_IsFull	Used to determine if the DMA Buffer A is full, for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_B_InterruptDisable	Disables the DMA Buffer B full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_B_InterruptEnable	Enables the DMA Buffer B full interrupt for the specified Channel 0 to 6.
	PLIB_ADCHS_DMABuffer_B_IsFull	Used to determine if the DMA Buffer B is full, for the specified Channel 0 to 6.
	PLIB_ADCHS_DMADisable	Disables the DMA in the High-Speed SAR ADC module.
	PLIB_ADCHS_DMAEnable	Enables the DMA in the High-Speed SAR ADC module.
	PLIB_ADCHS_DMAOverflowErrorHasOccurred	Used to determine if the DMA Buff had an overflow error.
	PLIB_ADCHS_DMASetup	Configures the DMA on the High-Speed SAR ADC.
	PLIB_ADCHS_DMASourceRemove	Disables the DMA for the specified Channel 0 to 6.
	PLIB_ADCHS_DMASourceSelect	Enables the DMA for the specified Channel 0 to 6.
	PLIB_ADCHS_EarlyInterruptDisable	Disables the early interrupt for the ADCHS.
	PLIB_ADCHS_EarlyInterruptEnable	Enables the early interrupt for the ADCHS.
	PLIB_ADCHS_Enable	Enables the High-Speed SAR ADC module.
	PLIB_ADCHS_ExistsAnalogInputCheck	Identifies whether the System Configuration feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsAnalogInputModeControl	Identifies whether the analog input mode control exists on the ADCHS module.
	PLIB_ADCHS_ExistsAnalogInputScan	Identifies whether the Analog input Scan exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelAnalogControl	Identifies whether the Channel Analog control exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelConfiguration	Identifies whether the Channel Configuration feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelDigitalControl	Identifies whether the Channel Digital control exists on the ADCHS module.
	PLIB_ADCHS_ExistsChannelInputSelectControl	Identifies whether the Channel 0 to 6 Input select feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsConfiguration	Identifies whether the Configuration feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsConversionResults	Identifies whether the Conversion Results feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsCVD	Identifies whether the CVD exists on the ADCHS module.
	PLIB_ADCHS_ExistsDigitalComparator	Identifies whether the Digital Comparator feature exists on the ADCHS module .
	PLIB_ADCHS_ExistsDigitalFilter	Identifies whether the Digital Filter feature exists on the ADCHS module.

	PLIB_ADCHS_ExistsDMA	Identifies whether the DMA exists on the ADCHS module.
	PLIB_ADCHS_ExistsEarlyInterruptControl	Identifies whether the Early Interrupt control exists on the ADCHS module.
	PLIB_ADCHS_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADCHS module
	PLIB_ADCHS_ExistsExternalConversionRequestControl	Identifies whether the External Convert feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsFIFO	Identifies whether the FIFO exists on the ADCHS module.
	PLIB_ADCHS_ExistsManualControl	Identifies whether the Manual control exists on the ADCHS module.
	PLIB_ADCHS_ExistsTriggerControl	Identifies whether the Trigger control exists on the ADCHS module.
	PLIB_ADCHS_ExistsTriggerSampleControl	Identifies whether the Trigger Sample control feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsTurboMode	Identifies whether the Turbo mode feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsUpdateReadyControl	Identifies whether the Update Ready feature exists on the ADCHS module.
	PLIB_ADCHS_ExistsVREFControl	Identifies whether the VREF control exists on the ADCHS module.
	PLIB_ADCHS_ExternalConversionRequestDisable	Disables the external conversion request.
	PLIB_ADCHS_ExternalConversionRequestEnable	Enables the external conversion request.
	PLIB_ADCHS_FIFODataCountGet	Returns the number of data to be read from FIFO.
	PLIB_ADCHS_FIFODataIsAvailable	Used to determine if the FIFO has data ready.
	PLIB_ADCHS_FIFODataIsNegative	Returns the sign of data stored in FIFO.
	PLIB_ADCHS_FIFODataReadyInterruptDisable	Disables the interrupt for FIFO.
	PLIB_ADCHS_FIFODataReadyInterruptEnable	Enables the interrupt for FIFO.
	PLIB_ADCHS_FIFODisable	Disables the FIFO in the High-Speed SAR ADC.
	PLIB_ADCHS_FIFOEnable	Enables the FIFO in the High-Speed SAR ADC
	PLIB_ADCHS_FIFOErrorHasOccurred	Used to determine if the FIFO has encountered an overflow error.
	PLIB_ADCHS_FIFORead	Used to fetch the data result from the FIFO.
	PLIB_ADCHS_FIFOSourceGet	Returns the channel ID using the FIFO.
	PLIB_ADCHS_FIFOSourceSelect	Sets the Channel 0 to 6 using the FIFO.
	PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable	Disables the global level software trigger.
	PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable	Initiates a global level software trigger.
	PLIB_ADCHS_GlobalSoftwareTriggerEnable	Initiate a Global Software Trigger.
	PLIB_ADCHS_ScanCompleteInterruptDisable	Disables the end of scan interrupt.
	PLIB_ADCHS_ScanCompleteInterruptEnable	Enables the end of scan interrupt.
	PLIB_ADCHS_Setup	Configures the High-Speed SAR ADC converter.
	PLIB_ADCHS_SoftwareConversionInputSelect	Selects the analog input of Channel 7 for manual conversion.
	PLIB_ADCHS_SoftwareConversionStart	Triggers the conversion of analog input signal connected to Channel 7.
	PLIB_ADCHS_SoftwareSamplingStart	Enables sampling of analog input for Channel 7.
	PLIB_ADCHS_SoftwareSamplingStop	Disables sampling of analog input for Channel 7, which places Channel 7 into Hold mode.
	PLIB_ADCHS_TriggerSuspendDisable	Disables trigger suspend.
	PLIB_ADCHS_TriggerSuspendEnable	Suspends all trigger for all ADCHS channels.
	PLIB_ADCHS_TurboModeChannelSelect	Configures the channels for Turbo mode.
	PLIB_ADCHS_TurboModeDisable	Disables Turbo mode for High-Speed SAR ADC module.
	PLIB_ADCHS_TurboModeEnable	Enables Turbo mode for the High-Speed SAR ADC module.
	PLIB_ADCHS_TurboModeErrorHasOccurred	Returns the error state of Turbo mode.
	PLIB_ADCHS_VREFFaultHasOccurred	Returns the state of VREF fault.
	PLIB_ADCHS_VREFFaultInterruptDisable	Disables the VREF Fault interrupt.
	PLIB_ADCHS_VREFFaultInterruptEnable	Enables the VREF fault interrupt.
	PLIB_ADCHS_VREFIsReady	Returns the state of VREF.
	PLIB_ADCHS_VREFReadyInterruptDisable	Disables the VREF ready interrupt.
	PLIB_ADCHS_VREFReadyInterruptEnable	Enables the VREF ready interrupt.

Description

High-Speed Successive Approximation Register Analog-to-Digital Converter (ADCHS) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the ADCHS PLIB for all families of Microchip microcontrollers.The definitions in this file are common to ADCHS peripheral.

File Name

plib_adchs.h

Company

Microchip Technology Inc.

ADCP Peripheral Library

This section describes the Pipelined Analog-to-Digital Converter (ADCP) Peripheral Library.

Introduction

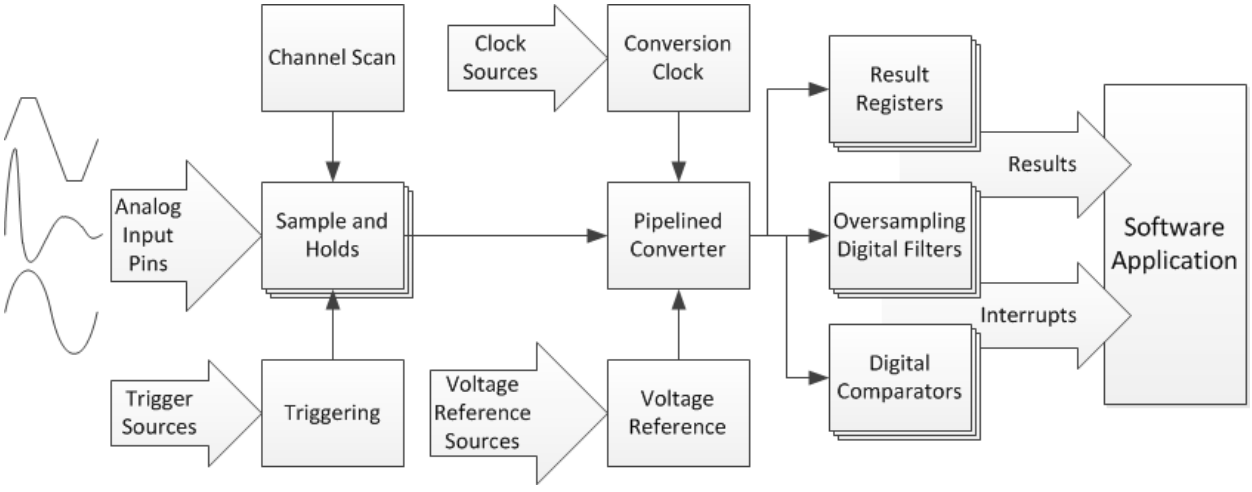
This library provides a low-level abstraction of the Pipelined Analog-to-Digital Converter (ADCP) Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Pipelined Analog-to-Digital Converter (ADCP) peripheral is used to convert an analog input into a digital number that represents that input voltage. When the input is periodically converted, the result is a sequence of digital values that have converted a continuous-time and continuous-amplitude analog signal to a discrete-time and discrete-amplitude digital signal. This particular ADC architecture is pipelined allowing it to simultaneously convert up to six samples, each at a different stage of conversion resulting in a low overall latency of multiple sampled inputs.

The structure of the ADCP is shown in the following diagram.

- Analog input pins connect to the sample and hold (S&H) circuits. These circuits capture the analog signal so that it can be passed to the pipeline converter.
- The S&H circuits are controlled by predefined trigger sources. These trigger sources switch the sample and hold circuit to hold and queue the conversion of the sample.
- Channel scan is available which allows a single trigger to start a sample/conversion sequence of multiple analog inputs using a predefined scan list
- A conversion clock is required for the pipeline converter. This clock determines the conversion speed and latency and affects power consumption.
- A voltage reference is required for the pipeline converter. This voltage reference determines allowable input range of the analog signal.
- Result registers store the conversion results and can be read by the software application. Software is notified of ready results by either polling or through the use of interrupts.
- Optional Oversampling Digital Filters can be used to provide increased measurement accuracy at the sacrifice of sample rate. Software is notified of ready oversampling results by either polling or through the use of interrupts.
- Optional digital comparators can be used to test conversion results independent of software and generate interrupts based on predefined conditions



Using the Library

This topic describes the basic architecture of the Pipelined Analog-to-Digital Converter (ADCP) Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_adcp.h`

The interface to the ADCP Peripheral library is defined in the `plib_adcp.h` header file, which is included by the `peripheral.h` file. Any C language source (`.c`) file that uses the Pipelined Analog-to-Digital Converter (ADCP) Peripheral library must include `peripheral.h`.

Library File:

The ADCP Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

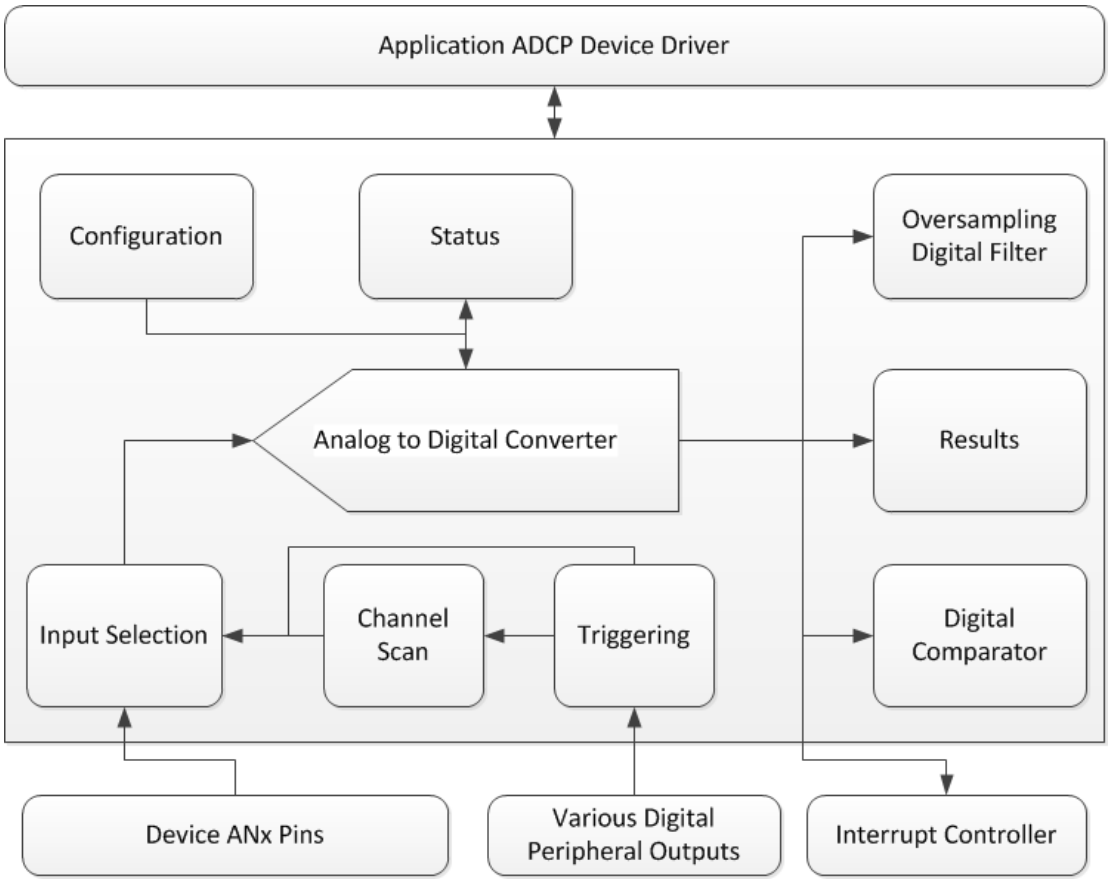
Hardware Abstraction Model

This library provides the low-level abstraction of the ADCP module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a super set of all the functionality of the available Pipelined Analog-to-Digital Converter (ADCP) on the device. Refer to the **"ADC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each ADCP module on the device.

Description

Pipelined Analog-to-Digital Converter (ADCP) Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the ADCP module.

Library Interface Section	Description
Configuration Functions	These library functions are used to configure the ADC, enable and disable it, initiate a calibration and manage low power states.
Status Functions	These library functions return status information related to the ADC.
Input Selection Functions	These library functions select and configure the ANx inputs to the ADC.
Channel Scan Functions	These library functions are used to configure the channel scan feature.
Triggering Functions	These library functions are used to configure and initiate conversion triggers.
Analog Input Conversion Results Functions	These library function are used to retrieve individual conversion results for ANx pins.
Digital Comparator Functions	These library functions are used to configure and query the digital comparators.
Oversampling Digital Filter Functions	These library functions are used to configure the Oversampling Digital Filter and fetch results from it.

How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

Core Functionality

This topic describes the core functionality of the ADCP Peripheral Library.

Description

Analog-to-Digital conversion using the ADCP involves the following three steps:

1. Sampling of the input signal.
2. Capture of the input signal by the sample and hold (S&H) circuit and transfer to the converter.
3. Conversion of the analog signal to its digital representation.

Sampling of the input signal involves charging of the S&H capacitor. The sampling time must be adequate so that the capacitor charges to a value equal to the input voltage. At the appropriate time, the input is disconnected and the capacitor voltage is transferred to the converter. The converter then digitizes the analog signal and provides the result.

The converter requires a clock source and a reference voltage. The clock is referred to as the ADC clock and has a period of TAD. The clock and reference voltage sources are selectable, as well as the clock prescaling that determines the TAD.

The ADCP uses two types of S&H circuits: dedicated and shared. Each ADC implementation includes up to five dedicated S&H circuits and a single shared S&H. Analog inputs connected to the S&H circuits are categorized into three types: Class 1, Class 2, and Class 3.

- Class 1 inputs are associated with a dedicated S&H circuit. Each dedicated S&H has a single Class 1 input associated with it. Each Class 1 input has a unique trigger selection register. Class 1 inputs can also be part of a channel scan list, triggered by the common scan trigger source.
- Class 2 inputs are sampled only on the shared S&H and are either individually triggered or as part of a channel scan list. When used individually their trigger source is selected by their unique trigger register.
- Class 3 inputs are sampled only on the shared S&H and are used in channel scan exclusively. They share a common trigger source. When using channel scan it is possible to combine Class 1, Class 2, and Class 3 inputs in the scan list.

Note: For details regarding configuration and triggering options was well as sampling requirements, refer to the **"ADC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter.

Example - Simultaneous Sampling Three Class 1 Inputs

Simultaneous sampling is used when the application requires capture of more than one signal at the same instance of time. Simultaneous sampling requires the use of multiple Class 1 inputs where each is assigned the same trigger source. The following example illustrates simultaneous sampling of AN0, AN1 and AN2 using the global software trigger as the trigger source. The dedicated Sample and Holds (S&H) circuits are configured for single-ended input and a unipolar (unsigned) output. No interrupts are used so the results are polled for ready status.

Example:

```
int32_t results[3] ;  // storage for results

// This structure is used to simplify testing for a ready channel
AN_READY anReady ;

// Configure the ADC
// AVDD/AVSS reference, ADC clock is system clock (SYSCLK) divided by 8
// No need to specify sample time for class 2/3 inputs or oversampling
```

```
    // sample time as Class 2/3 inputs and oversampling is not used.
    PLIB_ADCP_Configure ( ADCP_ID_1 ,
                          ADCP_VREF_AVDD_AVSS ,     // AVDD and AVSS as reference
                          FALSE ,                   // No VREF boost
                          FALSE ,                   // Do not use fractional format
                          FALSE ,                   // Do not stop in idle
                          ADCP_CLK_SRC_SYSCLK ,     // SYSCLK is the clock source
                          4 ,                       // TAD = 1/SYSCLK * 2 * 4
                                                    // or ADC Clock = SYSCLK / (2 * 4)
                          0 ,                       // Oversampling is not used.
                          0 ,                       // No early interrupt.
                          0 );                      // No class 2 or 3 channels are used

    // Set up the triggers
    // Configure AN0 for triggering from software.
    PLIB_ADCP_Class12TriggerConfigure ( ADCP_ID_1 ,
                                        ADCP_CLASS12_AN0 ,
                                        ADCP_TRG_SRC_SOFTWARE ) ;
    // Configure AN1 for triggering from software.
    PLIB_ADCP_Class12TriggerConfigure ( ADCP_ID_1 ,
                                        ADCP_CLASS12_AN1 ,
                                        ADCP_TRG_SRC_SOFTWARE ) ;
    // Configure AN2 for triggering from software.
    PLIB_ADCP_Class12TriggerConfigure ( ADCP_ID_1 ,
                                        ADCP_CLASS12_AN2 ,
                                        ADCP_TRG_SRC_SOFTWARE ) ;
    // Set S&H modes
    PLIB_ADCP_SHModeSelect ( ADCP_ID_1 , ADCP_SH0 , ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR ) ;
    PLIB_ADCP_SHModeSelect ( ADCP_ID_1 , ADCP_SH1 , ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR ) ;
    PLIB_ADCP_SHModeSelect ( ADCP_ID_1 , ADCP_SH2 , ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR ) ;
    // Enable the ADC Module
    PLIB_ADCP_Enable ( ADCP_ID_1 ) ;

    // Wait for a ready status
    while (!PLIB_ADCP_ModuleIsReady ( ADCP_ID_1 )) ;

    while (1)
    {
        // Initiate a trigger
        PLIB_ADCP_GlobalSoftwareTrigger ( ADCP_ID_1 ) ;

        // Wait for a data results to be ready
        // if lowest priority channel is done then all are done.
        do
        {
            anReady = PLIB_ADCP_ResultReady ( ADCP_ID_1 ) ;
        }
        while (anReady.AN2 == 0) ;

        // fetch AN0
        if (anReady.AN0 == 1)
            results[0] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN0 ) ;

        // fetch AN1
        if (anReady.AN1 == 1)
            results[1] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN1 ) ;

        // fetch AN2
        if (anReady.AN2 == 1)
            results[2] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN2 ) ;
     }
```

Example - Channel Scanning

Channel scanning is used in applications where precise timing of the sample to an external source is not needed. Channel scanning allows a large number of analog inputs to be sampled and converted in sequence each time a single trigger occurs. The following example illustrates how to configure channel scanning of multiple Class 2 and Class 3 inputs using the shared S&H. This example uses inputs AN5, AN11 - AN14. AN5 and AN11 are Class 2 inputs. AN13 and AN 14 are Class 3 inputs. The global software trigger is used to initiate the scan. A 4 TAD sample time is

specified for the shared S&H. The shared S&H is configured for single-ended operation and a unipolar output. No interrupts are used so the results are polled for ready status.

Example:

```
int32_t results[5] ;  // storage for results

// This structure is used to simplify testing of the ready status
AN_READY anReady ;

// This structure is used to simplify specifying channels for scan
// AN5, AN11, AN12, AN13 and AN14 are selected for scanning.
AN_SELECT anSelect ;
anSelect.highWord = 0 ;
anSelect.lowWord = 0 ;
anSelect.AN5 = 1 ;
anSelect.AN11 = 1 ;
anSelect.AN12 = 1 ;
anSelect.AN13 = 1 ;
anSelect.AN14 = 1 ;

// Configure the ADC
// AVDD/AVSS reference, ADC clock is system clock (SYSCLK) divided by 8
// The sample for the shared S&H must be adequate for all inputs.
// In this example it is set to 4 TAD.
// No need to specify sample time for oversampling as it is not used.
PLIB_ADCP_Configure ( ADCP_ID_1 ,
                      ADCP_VREF_AVDD_AVSS ,    // AVDD and AVSS as reference
                      FALSE ,                  // No VREF boost
                      FALSE ,                  // Do not use fractional format
                      FALSE ,                  // Do not stop in idle
                      ADCP_CLK_SRC_SYSCLK ,    // SYSCLK is the clock source
                      4 ,                      // TAD = 1/SYSCLK * 2 * 4
                                               // or ADC Clock = SYSCLK / (2 * 4)
                      0 ,                      // Oversampling is not used.
                      0 ,                      // No early interrupt.
                      3 ) ;                    // 3 + 1 = 4 TAD for Class 2 and 3
                                               // Sample Time.
// Specify the inputs to include in the Channel Scan using the selections made when
// initializing anSelect.  Use the software trigger to initiate the scan.
PLIB_ADCP_ChannelScanConfigure ( ADCP_ID_1 ,
                                 anSelect.lowWord , anSelect.highWord ,
                                 ADCP_SCAN_TRG_SRC_SOFTWARE ) ;

// Set S&H 5 (shared S&H) to use the single ended, unipolar mode
PLIB_ADCP_SHModeSelect ( ADCP_ID_1 , ADCP_SH5 , ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR ) ;

// Enable the ADC Module
PLIB_ADCP_Enable ( ADCP_ID_1 ) ;

// Wait for a ready status
while (!PLIB_ADCP_ModuleIsReady ( ADCP_ID_1 )) ;

while (1)
{
    // Initiate a trigger
    PLIB_ADCP_GlobalSoftwareTrigger ( ADCP_ID_1 ) ;

    // Wait for a data results to be ready
    // if lowest priority channel is done then all are done.
    do
    {
        anReady = PLIB_ADCP_ResultReady ( ADCP_ID_1 ) ;
    }
    while (anReady.AN14 == 0) ;

    // fetch AN5
    if (anReady.AN5 == 1)
        results[0] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN5 ) ;

    // fetch AN11
```

```
    if (anReady.AN11 == 1)
        results[1] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN11 ) ;

    // fetch AN12
    if (anReady.AN12 == 1)
        results[2] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN12 ) ;

    // fetch AN13
    if (anReady.AN13 == 1)
        results[3] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN13 ) ;

    // fetch AN14
    if (anReady.AN14 == 1)
        results[4] = PLIB_ADCP_ResultGet ( ADCP_ID_1 , ADCP_AN14 ) ;
}
```

Other Functionality

This topic provides information on additional functionality.

Description

The ADCP also implements a Digital Oversampling Filter and Digital Comparator.

The oversampling digital filter consists of an accumulator and a decimator (down-sampler), which function together as a low-pass filter. By sampling an analog input at a higher-than-required sample rate, and then processing the data through the oversampling digital filter, the number of usable bits of the ADCP module can be increased at the expense of decreased conversion throughput. For example, using 4x oversampling yields one extra usable bit , 16x oversampling yields two extra usable bits, 64x oversampling provides three extra usable bits, and 256x oversampling provides four extra usable bits. Once configured, the application provides a single trigger to the analog input specified for oversampling, it then fetches the result when the process is complete.

The ADCP module features a digital comparator that can be used to monitor selected analog input conversion results and generate an interrupt when a conversion result matches the user-specified limits. Conversion triggers are still required to initiate conversions. The comparison occurs automatically once the conversion is complete. When using a hardware source as a periodic trigger, it is possible to monitor analog inputs and create an interrupt when the converted level matches specified criteria without any software overhead.

Example - Digital Oversampling Filter

The Digital Oversampling filter can be used to increase resolution of the conversion result at the expense of throughput. The following example shows how two extra bits of resolution can be obtained using 16X oversampling (sixteen samples are used to create one higher resolution result). AN0 is used for the input. The sampling time for the retriggers is set to 3.5 TAD.

Example:

```
int32_t result; // storage for result

// Configure the ADC
// AVDD/AVSS reference, ADC clock is system clock (SYSCLK) divided by 8
// No need to specify sample time for Class 2/3 inputs as they are not used.
// Oversampling sample time is set to 3.5 TAD.
PLIB_ADCP_Configure(ADCP_ID_1,
                    ADCP_VREF_AVDD_AVSS, // AVDD and AVSS as reference
                    FALSE,               // No VREF boost
                    FALSE,               // Do not use fractional format
                    FALSE,               // Do not stop in idle
                    ADCP_CLK_SRC_SYSCLK, // SYSCLK is the clock source
                    4,                   // TAD = 1/SYSCLK * 2 * 4
                                         // or ADC Clock = SYSCLK / (2 * 4)
                    2,                   // 2 + 1.5 = 3.5 TAD between
                                         // oversampling triggers
                    0,                   // No early interrupt.
                    0);                  // No Class 2 and 3 are used.

// Set up the triggers
// Configure AN0 for triggering from software.
PLIB_ADCP_Class12TriggerConfigure(ADCP_ID_1, ADCP_CLASS12_AN0, ADCP_TRG_SRC_SOFTWARE);

// Set S&H modes
// Set S&H 0 to use the single ended, unipolar mode
PLIB_ADCP_SHModeSelect(ADCP_ID_1, ADCP_SH0, ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR);
```

```
// Configure the Oversampling Filter
PLIB_ADCP_OsampDigFilterConfig(ADCP_ID_1,            // ADCP module ID
                               ADCP_ODFLTR1,         // Filter ID
                               ADCP_AN0,             // Oversample AN0
                               ADCP_ODFLTR_16X,      // 16 x oversampling
                               FALSE);               // No Global Int Enable


// Enable ODFLTR0
PLIB_ADCP_OsampDigFilterEnable(ADCP_ID_1, ADCP_ODFLTR1);

// Enable the ADC Module
PLIB_ADCP_Enable(ADCP_ID_1);

// Wait for a ready status
while (!PLIB_ADCP_ModuleIsReady(ADCP_ID_1));

while (1) {
    // Initiate a trigger
    PLIB_ADCP_GlobalSoftwareTrigger(ADCP_ID_1);

    // Wait for data to be ready
    while (PLIB_ADCP_OsampDigFilterDataRdy(ADCP_ID_1, ADCP_ODFLTR1) == FALSE);

    // get the result
    result = PLIB_ADCP_OsampDigFilterDataGet(ADCP_ID_1, ADCP_ODFLTR1);

}
```

Example - Digital Comparator

The Digital Comparator is used to automatically evaluate results as they are output by the converter. The following example illustrates an automated

test of AN5 for values >= 80% of full scale or < 20% of full scale. A count is incremented each time an event occurs.

Example:

```
int32_t result; // storage for result
int32_t count = 0;

/* This structure is used to simplify testing for a ready channel */
AN_READY anReady;

// Configure the ADC
// AVDD/AVSS reference, ADC clock is system clock (SYSCLK) divided by 8
// The sample for the shared S&H must be adequate for all inputs.
// In this example it is set to 4 TAD.
// No need to specify sample time for oversampling as it is not used.
PLIB_ADCP_Configure(ADCP_ID_1,
                    ADCP_VREF_AVDD_AVSS, // AVDD and AVSS as reference
                    FALSE,               // No VREF boost
                    FALSE,               // Do not use fractional format
                    FALSE,               // Do not stop in idle
                    ADCP_CLK_SRC_SYSCLK, // SYSCLK is the clock source
                    4,                   // TAD = 1/SYSCLK * 2 * 4 or
                                         // ADC Clock = SYSCLK / (2 * 4)
                    0,                   // no oversampling
                    0,                   // No early interrupt.
                    3);                  // 3 + 1 = 4 TAD for Class 2 and 3 Sample Time.


// Set up the triggers
// Configure AN5 for triggering from software.
PLIB_ADCP_Class12TriggerConfigure(ADCP_ID_1, ADCP_CLASS12_AN5, ADCP_TRG_SRC_SOFTWARE);

// Set S&H modes
// Set S&H 5 to use the single ended, unipolar mode
PLIB_ADCP_SHModeSelect(ADCP_ID_1, ADCP_SH5, ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR);

// Configure the Digital Comparator
```

```
// Creates an event when the reading of AN5 is less than 20% or
// greater than or equal to 80% of the full scale 12-bit output.
PLIB_ADCP_DigCmpConfig(ADCP_ID_1,           // ADCP module ID
                       ADCP_DCMP1,          // Comparator ID
                       FALSE,               // No Global Int Enable
                       FALSE,               // no test for between low and high
                       TRUE,                // test for greater than or equal to high
                       FALSE,               // no test for less than high
                       FALSE,               // no test for greater than or equal to low
                       TRUE,                // test for less than low
                       1 << 5,              // enable AN5 for comparison
                       0xFFF - (0xFFF / 5), // high limit is 80% of full scale
                       0xFFF / 5);          // low limit is 20% of full scale

// Enable DigCmp1
PLIB_ADCP_DigCmpEnable(ADCP_ID_1, ADCP_DCMP1);

// Enable the ADC Module
PLIB_ADCP_Enable(ADCP_ID_1);

// Wait for a ready status
while (!PLIB_ADCP_ModuleIsReady(ADCP_ID_1));

while (1) {
    // Initiate a trigger
    PLIB_ADCP_GlobalSoftwareTrigger(ADCP_ID_1);

    // wait for conversion to complete
    do {
        anReady = PLIB_ADCP_ResultReady(ADCP_ID_1);
    } while (anReady.AN5 == 0);

    // Check result and increment count if an out of range event
    // occurred on digital comparator 1 for AN5
    if (PLIB_ADCP_DigCmpAIdGet(ADCP_ID_1, ADCP_DCMP1) == 5) {
        count++;
    }
}
```

Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Configuration Functions

	Name	Description
	PLIB_ADCP_Configure	Configures the Pipelined ADC module including the ADC calibration registers.
	PLIB_ADCP_Enable	Enables the Pipelined ADC module.
	PLIB_ADCP_Disable	Pipelined ADC module is disabled (turned OFF).
	PLIB_ADCP_CalibrationStart	Initiates Pipelined ADC Calibration.
	PLIB_ADCP_LowPowerStateEnter	Places the Pipelined ADC module in a low power state.
	PLIB_ADCP_LowPowerStateExit	Takes the Pipelined ADC module out of low power state and puts it into an operational mode.
	PLIB_ADCP_LowPowerStateGet	Returns the state of the low power setting.
	PLIB_ADCP_ExistsCalibration	Identifies whether the Calibration feature exists on the ADCP module.
	PLIB_ADCP_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADCP module.
	PLIB_ADCP_ExistsLowPowerControl	Identifies whether the LowPowerControl feature exists on the ADCP module.
	PLIB_ADCP_ExistsConfiguration	Identifies whether the Configuration feature exists on the ADCP module.

b) Status Functions

	Name	Description
	PLIB_ADCP_ModuleIsReady	Returns the overall ready status of the module.

	PLIB_ADCP_ExistsReadyStatus	Identifies whether the ReadyStatus feature exists on the ADCP module.

c) Input Selection Functions

	Name	Description
	PLIB_ADCP_AlternateInputSelect	Selects the alternate physical input for the specified dedicated (Class 1) S&H.
	PLIB_ADCP_DefaultInputSelect	Selects the default physical input for the specified dedicated (Class 1) S&H.
	PLIB_ADCP_ExistsInputSelect	Identifies whether the InputSelect feature exists on the ADCP module.
	PLIB_ADCP_ExistsModeSelect	Identifies whether the ModeSelect feature exists on the ADCP module.
	PLIB_ADCP_SHModeSelect	Selects the mode for the specified S&H.

d) Channel Scan Functions

	Name	Description
	PLIB_ADCP_ChannelScanConfigure	Selects input to include in Channel Scan Mode
	PLIB_ADCP_ExistsChannelScan	Identifies whether the ChannelScan feature exists on the ADCP module.

e) Triggering Functions

	Name	Description
	PLIB_ADCP_GlobalSoftwareTrigger	Initiates a Global Software Trigger on the specified module.
	PLIB_ADCP_IndividualTrigger	Triggers an individual channel independent of the configured trigger source
	PLIB_ADCP_Class12TriggerConfigure	Configures a Class 1 or Class 2 Trigger Source.
	PLIB_ADCP_ExistsTriggering	Identifies whether the Triggering feature exists on the ADCP module.

f) Individual Analog Input Conversion Results Functions

	Name	Description
	PLIB_ADCP_ResultReady	Returns the ADC conversion result ready bits for the module.
	PLIB_ADCP_ResultGet	Returns a Pipelined ADC conversion result.
	PLIB_ADCP_ResultReadyGrpIntConfigure	Selects input to include in global interrupt.
	PLIB_ADCP_ExistsConversionResults	Identifies whether the ConversionResults feature exists on the ADCP module.

g) Digital Comparator Functions

	Name	Description
	PLIB_ADCP_DigCmpEnable	Enables the Digital Comparator in the Pipelined ADC module.
	PLIB_ADCP_DigCmpDisable	Disables the Digital Comparator in the Pipelined ADC module.
	PLIB_ADCP_DigCmpAIdGet	Returns the Analog Input ID of for the Comparator Event
	PLIB_ADCP_DigCmpConfig	Configures the Digital Comparator on the Pipelined ADC converter.
	PLIB_ADCP_ExistsDigCmp	Identifies whether the DigitalComparator feature exists on the ADCP module.

h) Oversampling Digital Filter Functions

	Name	Description
	PLIB_ADCP_OsampDigFilterEnable	Enables the Oversampling Digital Filter in the Pipelined ADC module
	PLIB_ADCP_OsampDigFilterDisable	Disables the Oversampling Digital Filter in the Pipelined ADC module.
	PLIB_ADCP_OsampDigFilterDataRdy	Determines if the Oversampling Digital Filter has data ready.
	PLIB_ADCP_OsampDigFilterDataGet	Fetches the data result from the Oversampling Digital Filter.
	PLIB_ADCP_ExistsOsampDigFilter	Identifies whether the OsampDigitalFilter feature exists on the ADCP module.
	PLIB_ADCP_OsampDigFilterConfig	Configures the Oversampling Digital Filter on the Pipelined ADC converter.

i) Data Types and Constants

	Name	Description
	ADCP_MODULE_ID	Identifies the number of ADC Modules Supported.
	ADCP_VREF_SOURCE	Defines the ADCP VREF Source Select.
	ADCP_CLOCK_SOURCE	Defines the ADCP Clock Source Select.
	ADCP_CLASS12_INPUT_ID	Identifies the Class 1 and Class 2 ADC Inputs.
	ADCP_SH_ID	Identifies the supported S&H circuits regardless of type.
	ADCP_DSH_ID	Identifies the supported Dedicated Sample and Hold (S&H) circuits.
	ADCP_SH_MODE	Defines the available modes for the S&H.
	ADCP_INPUT_ID	Identifies the available ADC Inputs.

	ADCP_DCMP_ID	Identifies the supported Digital Comparators.
	ADCP_ODFLTR_ID	Identifies the supported Oversampling Digital Filters.
	ADCP_ODFLTR_OSR	Identifies the supported Digital Filter oversampling ratios.
	ADCP_SCAN_TRG_SRC	Defines the ADCP Channel Scan Trigger Source Selections.
	ADCP_TRG_SRC	Defines the ADCP Trigger Source Selections.
	AN_SELECT	This union of structures is provided to simply selection of analog inputs for inclusion in a particular function.
	AN_READY	This union (identical to the AN_SELECT union) is used as the return value by the PLIB_ADCP_Result_Ready function.

Description

This section describes the Application Programming Interface (API) functions of the Pipelined Analog-to-Digital Converter (ADCP) Peripheral Library.

Refer to each section for a detailed description.

a) Configuration Functions

PLIB_ADCP_Configure Function

Configures the Pipelined ADC module including the ADC calibration registers.

File

plib_adcp.h

C

```
void PLIB_ADCP_Configure(ADCP_MODULE_ID index, ADCP_VREF_SOURCE voltageRefSelect, bool boostVref, bool
fractionalOutputOn, bool stopInIdle, ADCP_CLOCK_SOURCE adcClockSource, int8_t adcClockDivider, int8_t
oversampleDigFilterSamTime, int8_t earlyIntEnable, int8_t class2and3SampleTime);
```

Returns

None.

Description

This function configures all ADC parameters common to all inputs. This configuration must occur prior to enabling the ADC and therefore must be called when the ADC is disabled.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The module is disabled when calling this function.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure the ADC
PLIB_ADCP_Configure( MY_ADCP_INSTANCE,
        ADCP_VREF_VREFP_VREFN,  // VREF+ and VREF- as reference
        FALSE,          // No VREF boost
        TRUE,           // Use fractional format
        TRUE,           // Do stop in idle
        ADCP_CLK_SRC_SYSCLK,    // SYSCLK is the clock source
        3,              // TAD = 1/SYSCLK * 2 * 3
                        // or ADC Clock = SYSCLK / (2 * 3)
        2,              // 2 + 1.5 = 3.5 TAD between
                        // oversampling triggers
        0,              // No early interrupt.
        3 );            // 3 + 1 = 4 TAD for Class 2 and 3
                        // Sample Time.

// Enable the ADC
```

```
PLIB_ADCP_Enable(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
voltageRefSelect	VREF Source Selection
boostVref	Enables the VREF boost if TRUE.
Use when VREFH	VREFL < .64 (AVDD - AVSS)
fractionalOutputOn	sets output to a fractional format if TRUE
stopInIdle	sets ADC to stop when device is in idle mode if TRUE
adcClockSource	Clock source selection
adcClockDivider	Clock source divider. Values range from 0 to 254.
oversampleDigFilterSamTime	Sets the delay (sample time) between automatically generated oversampling filter triggers. Values range from 0 to 31 for 1.5 TAD to 32.5 TAD respectively.
earlyIntEnable	Sets the number of clocks prior to the actual arrival of data that the ADRDY bit is set. Range is 0 to 7. Used to generate an early interrupt.
class2and3SampleTime	Set the sample time for Class 2 and Class 3 inputs. Range is 0 to 255 for a sample time of 1 to 256, respectively.

Function

```
void PLIB_ADCP_Configure(    ADCP_MODULE_ID index,
        ADCP_VREF_SOURCE voltageRefSelect,  // voltage reference
bool boostVref,                 // VREF boost
bool fractionalOutputOn,          // result format fractional
bool stopInIdle,                // stop in idle
        ADCP_CLOCK_SOURCE adcClockSource,   // clock source
int8_t adcClockDivider,           // clock divider
int8_t oversampleDigFilterSamTime,  // sample time between digital filter samples
int8_t earlyIntEnable,            // early interrupt enable
int8_t class2and3SampleTime )       // Class 2&3 Sample time
```

PLIB_ADCP_Enable Function

Enables the Pipelined ADC module.

File

plib_adcp.h

C

```
void PLIB_ADCP_Enable(ADCP_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the selected Pipelined ADC module.

Remarks

None

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
PLIB_ADCP_Enable(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance to be configured

Function

void PLIB_ADCP_Enable(ADCP_MODULE_ID index)

PLIB_ADCP_Disable Function

Pipelined ADC module is disabled (turned OFF).

File

plib_adcp.h

C

```
void PLIB_ADCP_Disable(ADCP_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the selected Pipelined ADC module.

Remarks

Not all functionality is available on all devices. Please refer to the specific device data sheet for the list of available features.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
PLIB_ADCP_Disable(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance to be configured

Function

void PLIB_ADCP_Disable(ADCP_MODULE_ID index)

PLIB_ADCP_CalibrationStart Function

Initiates Pipelined ADC Calibration.

File

plib_adcp.h

C

```
void PLIB_ADCP_CalibrationStart(ADCP_MODULE_ID index);
```

Returns

None.

Description

This function initiates calibration of the module. During calibration the module cannot perform conversion.

Remarks

Calibration is complete when PLIB_ADCP_ModuleIsReady() returns a TRUE result.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
PLIB_ADCP_CalibrationStart(MY_ADCP_INSTANCE);
    while (!PLIB_ADCP_ModuleIsRead(MY_ADCP_INSTANCE));
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance to be configured

Function

void PLIB_ADCP_CalibrationStart(ADCP_MODULE_ID index)

PLIB_ADCP_LowPowerStateEnter Function

Places the Pipelined ADC module in a low power state.

File

plib_adcp.h

C

```
void PLIB_ADCP_LowPowerStateEnter(ADCP_MODULE_ID index);
```

Returns

None.

Description

This function places the Pipelined ADC module in a low power state. The feature is used in place of disabling the ADC when power reduction is needed. The Pipelined ADC can come out of low power state and be operational much faster since recalibration is not required.

Remarks

None.

Preconditions

The Pipelined ADC must be enabled.

Example

```
PLIB_ADCP_LowPowerStateEnter(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Function

void PLIB_ADCP_LowPowerStateEnter (ADCP_MODULE_ID index)

PLIB_ADCP_LowPowerStateExit Function

Takes the Pipelined ADC module out of low power state and puts it into an operational mode.

File

plib_adcp.h

C

```
void PLIB_ADCP_LowPowerStateExit(ADCP_MODULE_ID index);
```

Returns

None.

Description

This function takes the Pipelined ADC module out of a low power state and places it into an operational mode.

Remarks

The first five conversions following the exit from ADC Low-power mode may be subject to lower accuracy than specified in the device data sheet.

Preconditions

None.

Example

```
PLIB_ADCP_LowPowerStateExit(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Function

void PLIB_ADCP_LowPowerStateExit(ADCP_MODULE_ID index)

PLIB_ADCP_LowPowerStateGet Function

Returns the state of the low power setting.

File

plib_adcp.h

C

```
bool PLIB_ADCP_LowPowerStateGet(ADCP_MODULE_ID index);
```

Returns

A Boolean indicating the state of the low power setting.

Description

This function returns the state of the low power setting.

Remarks

None.

Preconditions

None.

Example

```
bool lowPowerSate = PLIB_ADCP_LowPowerStateGet(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Function

bool PLIB_ADCP_LowPowerStateGet(ADCP_MODULE_ID index)

PLIB_ADCP_ExistsCalibration Function

Identifies whether the Calibration feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsCalibration(ADCP_MODULE_ID index);
```

Returns

Existence of the Calibration feature:

- true - When Calibration feature is supported on the device
- false - When Calibration feature is not supported on the device

Description

This function identifies whether the Calibration feature is available on the ADCP module. When this function returns true, this function is supported on the device:

- PLIB_ADCP_CalibrationStart

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsCalibration(ADCP_MODULE_ID index)

PLIB_ADCP_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsEnableControl(ADCP_MODULE_ID index);
```

Returns

Existence of the EnableControl feature:

- true - When EnableControl feature is supported on the device
- false - When EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- PLIB_ADCP_Enable
- PLIB_ADCP_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsEnableControl(ADCP_MODULE_ID index)

PLIB_ADCP_ExistsLowPowerControl Function

Identifies whether the LowPowerControl feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsLowPowerControl(ADCP_MODULE_ID index);
```

Returns

Existence of the LowPowerControl feature:

- true - When LowPowerControl feature is supported on the device
- false - When LowPowerControl feature is not supported on the device

Description

This function identifies whether the LowPowerControl feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- PLIB_ADCP_LowPowerStateEnter
- PLIB_ADCP_LowPowerStateExit
- PLIB_ADCP_LowPowerStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsLowPowerControl(ADCP_MODULE_ID index)

PLIB_ADCP_ExistsConfiguration Function

Identifies whether the Configuration feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsConfiguration(ADCP_MODULE_ID index);
```

Returns

Existence of the Configuration feature:

- true - When Configuration feature is supported on the device
- false - When Configuration feature is not supported on the device

Description

This function identifies whether the Configuration feature is available on the ADCP module. When this function returns true, this function is supported on the device:

- PLIB_ADCP_Configure

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsConfiguration(ADCP_MODULE_ID index)

b) Status Functions

PLIB_ADCP_ModuleIsReady Function

Returns the overall ready status of the module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ModuleIsReady(ADCP_MODULE_ID index);
```

Returns

Boolean indicating ready status.

Description

This function returns the ready status of the Pipelined ADC. The ADC must be ready before operation.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Enable the module
PLIB_ADCP_Enable(MY_ADCP_INSTANCE);

// wait for calibration to complete
while (!PLIB_ADCP_ModuleIsReady(MY_ADCP_INSTANCE));

// begin sampling
...
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Function

bool PLIB_ADCP_ModuleIsReady(ADCP_MODULE_ID index)

PLIB_ADCP_ExistsReadyStatus Function

Identifies whether the ReadyStatus feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsReadyStatus(ADCP_MODULE_ID index);
```

Returns

Existence of the ReadyStatus feature:

- true - When ReadyStatus feature is supported on the device
- false - When ReadyStatus feature is not supported on the device

Description

This function identifies whether the ReadyStatus feature is available on the ADCP module. When this function returns true, this function is supported on the device:

- PLIB_ADCP_ModuleIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsReadyStatus(ADCP_MODULE_ID index)

c) Input Selection Functions

PLIB_ADCP_AlternateInputSelect Function

Selects the alternate physical input for the specified dedicated (Class 1) S&H.

File

plib_adcp.h

C

```
void PLIB_ADCP_AlternateInputSelect(ADCP_MODULE_ID index, ADCP_DSH_ID id);
```

Returns

None.

Description

This function selects the alternate physical input for the specified S&H.

Remarks

None.

Preconditions

The function only applies to dedicated S&H circuits (Class 1 Inputs).

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Set S&H 1 to use the alternate input.
PLIB_ADCP_AlternateInputSelect(MY_ADCP_INSTANCE, ADCP_DSH1)
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	An ADCP_DSH_ID type indicating which dedicated Class 1 S&H to configure for its alternate input source.

Function

void PLIB_ADCP_AlternateInputSelect(ADCP_MODULE_ID index, ADCP_DSH_ID id)

PLIB_ADCP_DefaultInputSelect Function

Selects the default physical input for the specified dedicated (Class 1) S&H.

File

plib_adcp.h

C

```
void PLIB_ADCP_DefaultInputSelect(ADCP_MODULE_ID index, ADCP_DSH_ID id);
```

Returns

None.

Description

This function selects the default physical input for the specified S&H.

Remarks

None.

Preconditions

The function only applies to dedicated S&H circuits (Class 1 Inputs).

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Set S&H 1 to use the default input.
PLIB_ADCP_DefaultInputSelect(MY_ADCP_INSTANCE, ADCP_DSH1)
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	An ADCP_DSH_ID type indicating which dedicated Class 1 S&H to configure for its alternate input source.

Function

void PLIB_ADCP_DefaultInputSelect(ADCP_MODULE_ID index, ADCP_DSH_ID id)

PLIB_ADCP_ExistsInputSelect Function

Identifies whether the InputSelect feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsInputSelect(ADCP_MODULE_ID index);
```

Returns

Existence of the InputSelect feature:

- true - When InputSelect feature is supported on the device
- false - When InputSelect feature is not supported on the device

Description

This function identifies whether the InputSelect feature is available on the ADCP module. When this function returns true, this function is supported on the device:

- PLIB_ADCP_DedicatedSHInputSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsInputSelect(ADCP_MODULE_ID index)

PLIB_ADCP_ExistsModeSelect Function

Identifies whether the ModeSelect feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsModeSelect(ADCP_MODULE_ID index);
```

Returns

Existence of the ModeSelect feature:

- true - When ModeSelect feature is supported on the device
- false - When ModeSelect feature is not supported on the device

Description

This function identifies whether the ModeSelect feature is available on the ADCP module. When this function returns true, this function is supported on the device:

- PLIB_ADCP_SHModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsModeSelect(ADCP_MODULE_ID index)

PLIB_ADCP_SHModeSelect Function

Selects the mode for the specified S&H.

File

plib_adcp.h

C

```
void PLIB_ADCP_SHModeSelect(ADCP_MODULE_ID index, ADCP_DSH_ID id, ADCP_SH_MODE mode);
```

Returns

None.

Description

This function selects the mode (single ended or differential) and encoding (unipolar or two's compliment) for the specified S&H.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Set S&H 1 to use the single ended, two's complement mode
// selection.
PLIB_ADCP_SHModeSelect(MY_ADCP_INSTANCE,
                       ADCP_SH1,
                       ADCP_SH_MODE_SINGLE_ENDED_TWOS_COMP)
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	An ADCP_DSH_ID type indicating which dedicated S&H to configure to use its alternate source
mode	An ADCP_SH_MODE type indicating the mode selection

Function

void PLIB_ADCP_SHModeSelect(ADCP_MODULE_ID index,
ADCP_SH_ID id,
ADCP_SH_MODE mode)

d) Channel Scan Functions

PLIB_ADCP_ChannelScanConfigure Function

Selects input to include in Channel Scan Mode

File

plib_adcp.h

C

```
void PLIB_ADCP_ChannelScanConfigure(ADCP_MODULE_ID index, uint32_t lowEnable, uint32_t highEnable,
ADCP_SCAN_TRG_SRC triggerSource);
```

Returns

None.

Description

This function selects inputs, as determined by the low and high enable scan list for inclusion in the channel scan sequence. If the input is a Class 1 or Class 2 type, it will also select the trigger source for that input to be the scan trigger, which is required if included in channel scanning.

Remarks

The type def AN_SELECT can be used to create a variable to simplify selection of the inputs to include in the channel scan. This typedef creates bit field structures for each ANx input that are joined with unions for the low and high 32-bit words. See the code example for Channel scan in the ADCP PLIB help documentation for details on its use.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure Channel Scanning
// Channels 10 through 13
// Trigger on Timer 1 match.
PLIB_ADCP_ChannelScanConfigure(MY_ADCP_INSTANCE,
                               0XF000, //   AN12 - AN15
                               0X0F00, //  AN24 - AN27
                               ADCP_SCAN_TRG_SRC_TMR1_MATCH)
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
lowEnable	bit mask for selecting low order scan channels
highEnable	bit mask for selecting high order scan channels
triggerSource	Trigger source used to initiate channel scan

Function

void PLIB_ADCP_ChannelScanConfigure(ADCP_MODULE_ID index,
uint32_t lowEnable,
uint32_t highEnable,
ADCP_SCAN_TRG_SRC triggerSource)

PLIB_ADCP_ExistsChannelScan Function

Identifies whether the ChannelScan feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsChannelScan(ADCP_MODULE_ID index);
```

Returns

Existence of the ChannelScan feature:

- true - When ChannelScan feature is supported on the device
- false - When ChannelScan feature is not supported on the device

Description

This function identifies whether the ChannelScan feature is available on the ADCP module. When this function returns true, this function is supported on the device:

- PLIB_ADCP_ChannelScanConfigure

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsChannelScan(ADCP_MODULE_ID index)

e) Triggering Functions

PLIB_ADCP_GlobalSoftwareTrigger Function

Initiates a Global Software Trigger on the specified module.

File

plib_adcp.h

C

```
void PLIB_ADCP_GlobalSoftwareTrigger(ADCP_MODULE_ID index);
```

Returns

None.

Description

All inputs or scan list that is configured to trigger on the global software trigger are triggered by this function.

Remarks

None.

Preconditions

None.

Example

```
PLIB_ADCP_GlobalSoftwareTrigger(MY_ADCP_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Function

void PLIB_ADCP_GlobalSoftwareTrigger(ADCP_MODULE_ID index)

PLIB_ADCP_IndividualTrigger Function

Triggers an individual channel independent of the configured trigger source

File

plib_adcp.h

C

```
void PLIB_ADCP_IndividualTrigger(ADCP_MODULE_ID index, ADCP_INPUT_ID inputId);
```

Returns

None.

Description

This function forces a trigger of an individual Class 1 or Class 2 channel independent of its configured trigger source.

Remarks

None.

Preconditions

The function only applies to Class 1 and Class 2 inputs.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
```

```
// Force a trigger of AN5
PLIB_ADCP_IndividualTrigger(MY_ADCP_INSTANCE, ADCP_CLASS12_AN5)
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
inputId	An ADCP_INPUT_ID type indicating which input to trigger.

Function

void PLIB_ADCP_IndividualTrigger(ADCP_MODULE_ID index, ADCP_INPUT_ID inputId)

PLIB_ADCP_Class12TriggerConfigure Function

Configures a Class 1 or Class 2 Trigger Source.

File

plib_adcp.h

C

```
void PLIB_ADCP_Class12TriggerConfigure(ADCP_MODULE_ID index, ADCP_CLASS12_INPUT_ID inputId, ADCP_TRG_SRC
triggerSource);
```

Returns

none.

Description

This function configures the trigger source for Class 1 or Class 2 inputs. A call to this function is not required when the input is being used as part of a channel scan as the channel scan configure function also configures all trigger sources.

Remarks

None.

Preconditions

The function only applies to Class 1 and Class 2 inputs.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure AN0 for triggering from INT0.
PLIB_ADCP_Class12TriggerConfigure(MY_ADCP_INSTANCE,
                                  ADCP_CLASS12_AN0,
                                  ADCP_TRG_SRC_INT0);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
inputId	Class 1 or Class 2 input to configure the trigger for.
triggerSource	Trigger source to use for this input.

Function

void PLIB_ADCP_Class12TriggerConfigure(ADCP_MODULE_ID index,
ADCP_CLASS12_INPUT_ID inputId,
ADCP_TRG_SRC triggerSource)

PLIB_ADCP_ExistsTriggering Function

Identifies whether the Triggering feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsTriggering(ADCP_MODULE_ID index);
```

Returns

Existence of the Triggering feature:

- true - When Triggering feature is supported on the device
- false - When Triggering feature is not supported on the device

Description

This function identifies whether the Triggering feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- PLIB_ADCP_Class12TriggerConfigure
- PLIB_ADCP_GlobalSoftwareTrigger
- PLIB_ADCP_IndividualTrigger

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsTriggering(ADCP_MODULE_ID index)

f) Individual Analog Input Conversion Results Functions

PLIB_ADCP_ResultReady Function

Returns the ADC conversion result ready bits for the module.

File

plib_adcp.h

C

```
AN_READY PLIB_ADCP_ResultReady(ADCP_MODULE_ID index);
```

Returns

A AN_READY type indicating conversion result status.

Description

This function returns a result indicating which analog inputs have conversion results ready.

Remarks

This function returns individual conversion results. It does not return results from the module.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
AN_READY     MyRdyStatus;
```

```
// check for data and process it
if ((MyRdyStatus = PLIB_ADCP_ResultReady(MY_ADCP_INSTANCE)) != 0) {
    // fetch the results and process
}
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance

Function

AN_READY PLIB_ADCP_ResultReady(ADCP_MODULE_ID index)

PLIB_ADCP_ResultGet Function

Returns a Pipelined ADC conversion result.

File

plib_adcp.h

C

```
int32_t PLIB_ADCP_ResultGet(ADCP_MODULE_ID index, ADCP_INPUT_ID inputId);
```

Returns

The conversion result expressed as a 32-bit integer.

Description

This function returns the specified Pipelined ADC analog input conversion result.

Remarks

None.

Preconditions

A valid conversion is ready to be fetched.

Example

```
int32_t     result;
...
// fetch result for AN31
result = PLIB_ADCP_ResultGet( MY_ADCP_INSTANCE, ADCP_AN31 );
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
inputId	Identifier for the input

Function

int32_t PLIB_ADCP_ResultGet(ADCP_MODULE_ID index, ADCP_INPUT_ID inputId)

PLIB_ADCP_ResultReadyGrpIntConfigure Function

Selects input to include in global interrupt.

File

plib_adcp.h

C

```
void PLIB_ADCP_ResultReadyGrpIntConfigure(ADCP_MODULE_ID index, uint32_t lowEnable, uint32_t highEnable);
```

Returns

None.

Description

This function selects inputs, as determined by the input mask channel scan list for inclusion in the global or global interrupt.

Remarks

The type def AN_SELECT can be used to create a variable to simplify selection of the inputs to include in the global interrupt. This typedef creates bit field structures for each ANx input that are joined with unions for the low and high 32-bit words. See the code example for Channel scan in the ADCP PLIB help documentation for details on its use.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure Global Interrupts
// Analog inputs 10 through 13 are included in the global interrupt.
PLIB_ADCP_ResultReadyGrpIntConfigure(MY_ADCP_INSTANCE,
                                     0x0F00,    // inputs AN8 - AN11
                                     0x000F );  // inputs AN16 - AN19
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
lowEnable	bit mask for selecting low order scan channels
highEnable	bit mask for selecting high order scan channels

Function

void PLIB_ADCP_ResultReadyGrpIntConfigure(ADCP_MODULE_ID index,

uint32_t lowEnable,

uint32_t highEnable)

PLIB_ADCP_ExistsConversionResults Function

Identifies whether the ConversionResults feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsConversionResults(ADCP_MODULE_ID index);
```

Returns

Existence of the ConversionResults feature:

- true - When ConversionResults feature is supported on the device
- false - When ConversionResults feature is not supported on the device

Description

This function identifies whether the ConversionResults feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- PLIB_ADCP_ResultReady
- PLIB_ADCP_ResultGet
- PLIB_ADCP_ResultReadyGrpIntConfigure

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsConversionResults(ADCP_MODULE_ID index)

g) Digital Comparator Functions

PLIB_ADCP_DigCmpEnable Function

Enables the Digital Comparator in the Pipelined ADC module.

File

plib_adcp.h

C

```
void PLIB_ADCP_DigCmpEnable(ADCP_MODULE_ID index, ADCP_DCMP_ID id);
```

Returns

None.

Description

This function enables (turns ON) the selected Digital Comparator in the specified Pipelined ADC module.

Remarks

None

Preconditions

The digital comparator should be configured using the PLIB_ADCP_Configure() function prior to enabling.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Enable Digital Comparator 1
PLIB_ADCP_DigCmpEnable(MY_ADCP_INSTANCE, ADCP_DCMP1);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital comparator in the ADCP module.

Function

void PLIB_ADCP_DigCmpEnable(ADCP_MODULE_ID index, ADCP_DCMP_ID id)

PLIB_ADCP_DigCmpDisable Function

Disables the Digital Comparator in the Pipelined ADC module.

File

plib_adcp.h

C

```
void PLIB_ADCP_DigCmpDisable(ADCP_MODULE_ID index, ADCP_DCMP_ID id);
```

Returns

None.

Description

This function Disables (turns OFF) the selected Digital Comparator in the specified Pipelined ADC module.

Remarks

None

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Disable Digital Comparator 1
PLIB_ADCP_DigCmpDisable(MY_ADCP_INSTANCE, ADCP_DCMP1);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital comparator in the ADCP module.

Function

void PLIB_ADCP_DigCmpDisable(ADCP_MODULE_ID index, ADCP_DCMP_ID id)

PLIB_ADCP_DigCmpAIdGet Function

Returns the Analog Input ID of for the Comparator Event

File

plib_adcp.h

C

```
int16_t PLIB_ADCP_DigCmpAIdGet(ADCP_MODULE_ID index, ADCP_DCMP_ID id);
```

Returns

Value 'x' of type int16_t where 'x' is the index to the analog input ANx, which caused the event or -1 if no event has occurred.

Description

This function tests the DigCmp Event Flag and if true, returns the ANx Identifier for the input that caused the event, or -1 if there was no pending DigCmp event. The ID value returned corresponds to numeric portion of the analog input ID, ANx.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.
int16_t     DigCmpResult;

// Get Digital Comparator 1 result.
DigCmpResult = PLIB_ADCP_DigCmpAIdGet(MY_ADCP_INSTANCE, ADCP_DCMP1);
if (DigCmpResult != -1) {
    // process event
}
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance

id	Identifier for the digital comparator in the ADCP module

Function

int16_t PLIB_ADCP_DigCmpAIdGet(ADCP_MODULE_ID index, ADCP_DCMP_ID id)

PLIB_ADCP_DigCmpConfig Function

Configures the Digital Comparator on the Pipelined ADC converter.

File

plib_adcp.h

C

```
void PLIB_ADCP_DigCmpConfig(ADCP_MODULE_ID index, ADCP_DCMP_ID id, bool globalIntEnable, bool
inBetweenOrEqual, bool greaterEqualHi, bool lessThanHi, bool greaterEqualLo, bool lessThanLo, uint32_t
inputEnable, int32_t hiLimit, int32_t loLimit);
```

Returns

None.

Description

This function configures all parameters for the Digital Comparator module of the pipelined ADC.

Remarks

This function must be called when the ADC is disabled. The format of hiLimit and loLimit must match the output format of the channel(s) specified in inputEnable.

Preconditions

The module is disabled when calling this function.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure the Digital Comparator
// Creates an event when the reading of AN0 is between 20% to 80% of the
// full scale 12-bit output.
PLIB_ADCP_DigCmpConfig( MY_ADCP_INSTANCE,   // ADCP module ID
                        ADCP_DCMP1,       // Comparator ID
                        FALSE,            // Global Int Enable
                        TRUE,             // test for between low and high
                        FALSE,            // no test for greater than equal to high
                        FALSE,            // no test for less than high
                        FALSE,            // no test for greater than equal to low
                        FALSE,            // no test for less than low
                        1 << 3,           // enable AN3
                        0x0CCD,           // high limit, 80% of full scale
                        0x0333);          // low limit, 20% of full scale

// Enable Digital Comparator 1
PLIB_ADCP_DigCmpEnable(MY_ADCP_INSTANCE, ADCP_DCMP1);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital comparator in this device
globalIntEnable	When set, comparator events are included in the Global Interrupt
inBetweenOrEqual	Event is generated when result is greater than or equal to loLimit and less than hiLimit
greaterEqualHi	Event is generated when result is greater than or equal to hiLimit
lessThanHi	Event is generated when result is less than hiLimit
greaterEqualLo	Event is generated when result is greater than or equal to loLimit
lessThanLo	Event is generated when result is less than loLimit

inputEnable	Bit field which determines which analog inputs are tested by this comparator module. Bit 0 applies to AN0 and bit 31 to AN31.
hiLimit	High limit in the same format as the conversion result.
loLimit	Low limit in the same format as the conversion result.

Function

```
void PLIB_ADCP_DigCmpConfig(  ADCP_MODULE_ID index,  // ADCP module ID
                              ADCP_DCMP_ID id,       // Comparator ID
bool globalIntEnable,  // Global Int Enable
bool inBetweenOrEqual, // between low and high
bool greaterEqualHi,   // greater than equal to high
bool lessThanHi,       // less than high
bool greaterEqualLo,   // greater than equal to low
bool lessThanLo,       // less than low
uint32_t inputEnable,  // input enable bits
int32_t  hiLimit,      // high limit
int32_t  loLimit )     // low limit
```

PLIB_ADCP_ExistsDigCmp Function

Identifies whether the DigitalComparator feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsDigCmp(ADCP_MODULE_ID index);
```

Returns

Existence of the DigitalComparator feature:

- true - When DigitalComparator feature is supported on the device
- false - When DigitalComparator feature is not supported on the device

Description

This function identifies whether the DigitalComparator feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- PLIB_ADCP_DigCmpConfig
- PLIB_ADCP_DigCmpEnable
- PLIB_ADCP_DigCmpDisable
- PLIB_ADCP_DigCmpAIdGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

```
PLIB_ADCP_ExistsDigCmp(  ADCP_MODULE_ID index )
```

h) Oversampling Digital Filter Functions

PLIB_ADCP_OsampDigFilterEnable Function

Enables the Oversampling Digital Filter in the Pipelined ADC module

File

plib_adcp.h

C

```
void PLIB_ADCP_OsampDigFilterEnable(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id);
```

Returns

None.

Description

This function enables (turns ON) the selected Oversampling Digital Filter in the specified Pipelined ADC module.

Remarks

None.

Preconditions

The Oversampling Digital Filter should be configured using the PLIB_ADCP_Configure() function prior to enabling.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Enable OsampDigFilter1
PLIB_ADCP_OsampDigFilterEnable(MY_ADCP_INSTANCE, ADCP_ODFLTR1);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital Filter in the ADCP module

Function

void PLIB_ADCP_OsampDigFilterEnable(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id)

PLIB_ADCP_OsampDigFilterDisable Function

Disables the Oversampling Digital Filter in the Pipelined ADC module.

File

plib_adcp.h

C

```
void PLIB_ADCP_OsampDigFilterDisable(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id);
```

Returns

None.

Description

This function Disables (turns OFF) the selected Oversampling Digital Filter in the specified Pipelined ADC module.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
```

```
// application developer.

// Disable OsampDigFilter1
PLIB_ADCP_OsampDigFilterDisable(MY_ADCP_INSTANCE, ADCP_ODFLTR1);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital Filter in the ADCP module

Function

void PLIB_ADCP_OsampDigFilterDisable(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id)

PLIB_ADCP_OsampDigFilterDataRdy Function

Determines if the Oversampling Digital Filter has data ready.

File

plib_adcp.h

C

```
bool PLIB_ADCP_OsampDigFilterDataRdy(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id);
```

Returns

Boolean:

- true - Data is ready
- false - Data is not ready

Description

This function can be used to determine if the ADCP digital filter has data ready. A TRUE is returned when data is available, which can be fetched using PLIB_ADCP_OsampDigFilterDataGet.

Remarks

None.

Preconditions

None.

Example

```
if (PLIB_ADCP_OsampDigFilterDataRdy(MY_ADCP_INSTANCE, ADCP_ODFLTR_ID_0)) {
            // fetch and process data
    }
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital Filter in this device

Function

bool PLIB_ADCP_OsampDigFilterDataRdy(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id)

PLIB_ADCP_OsampDigFilterDataGet Function

Fetches the data result from the Oversampling Digital Filter.

File

plib_adcp.h

C

```
int16_t PLIB_ADCP_OsampDigFilterDataGet(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id);
```

Returns

A 16-bit result in the format specified by the filter's oversampling setting.

Description

This function is used to fetch data from the Oversampling Digital Filter. The format of the data is determined by the oversampling setting configuration defined in the call of PLIB_ADCP_OsampDigFilterConfig.

Remarks

None.

Preconditions

None.

Example

```
int16_t myODFLTRResult;

if (PLIB_ADCP_OsampDigFilterDataRdy(MY_ADCP_INSTANCE, ADCP_ODFLTR1)) {
    // fetch data
    myODFLTRResult = PLIB_ADCP_OsampDigFilterDataGet(MY_ADCP_INSTANCE, ADCP_ODFLTR1);
    // process result
    ...
}
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
dfltrID	Identifier for the digital Filter in this device

Function

int16_t PLIB_ADCP_OsampDigFilterDataGet(ADCP_MODULE_ID index, ADCP_ODFLTR_ID dfltrID)

PLIB_ADCP_ExistsOsampDigFilter Function

Identifies whether the OsampDigitalFilter feature exists on the ADCP module.

File

plib_adcp.h

C

```
bool PLIB_ADCP_ExistsOsampDigFilter(ADCP_MODULE_ID index);
```

Returns

Existence of the OsampDigitalFilter feature:

- true - When OsampDigitalFilter feature is supported on the device
- false - When OsampDigitalFilter feature is not supported on the device

Description

This function identifies whether the OsampDigitalFilter feature is available on the ADCP module. When this function returns true, these functions are supported on the device:

- PLIB_ADCP_OsampDigFilterConfig
- PLIB_ADCP_OsampDigFilterEnable
- PLIB_ADCP_OsampDigFilterDisable
- PLIB_ADCP_OsampDigFilterDataRdy
- PLIB_ADCP_OsampDigFilterDataGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ADCP_ExistsOsampDigFilter(ADCP_MODULE_ID index)

PLIB_ADCP_OsampDigFilterConfig Function

Configures the Oversampling Digital Filter on the Pipelined ADC converter.

File

plib_adcp.h

C

```
void PLIB_ADCP_OsampDigFilterConfig(ADCP_MODULE_ID index, ADCP_ODFLTR_ID id, ADCP_INPUT_ID inputId,
ADCP_ODFLTR_SAMPLE_RATIO oversampleRatio, bool globalIntEnable);
```

Returns

None.

Description

Configures all parameters for the Oversampling Digital Filter module of the pipelined ADC.

Remarks

This function must be called when the ADC is disabled.

Preconditions

The Oversampling Digital Filter module is disabled when calling this function.

Example

```
// Where MY_ADCP_INSTANCE, is the ADCP instance selected for use by the
// application developer.

// Configure the Oversampling Digital Filter
// AN4 is oversampled at a 16X rate.  No global interrupt is enabled.
PLIB_ADCP_OsampDigFilterConfig( MY_ADCP_INSTANCE,   // ADCP module ID
                                ADCP_ODFLTR1,       // Filter ID
                                ADCP_AN4,           // Oversample AN4
                                ADCP_ODFLTR_16X,    // 16 x oversampling
                                FALSE );            // No Global Int Enable

// Enable OsampDigFilter1
PLIB_ADCP_OsampDigFilterEnable(MY_ADCP_INSTANCE, ADCP_ODFLTR1);
```

Parameters

Parameters	Description
index	Identifier for the ADCP instance
id	Identifier for the digital Filter in this device
inputId	Identifier for the analog input to be oversampled
oversampleRatio	Enumerator specifying the oversampling ratio
globalIntEnable	When set, Filter events are included in the Global Interrupt

Function

void PLIB_ADCP_OsampDigFilterConfig(ADCP_MODULE_ID index, // ADCP module ID
ADCP_ODFLTR_ID id, // Filter ID
ADCP_INPUT_ID inputId, // Input Id
ADCP_ODFLTR_OSR oversampleRatio, // Oversampling ratio
bool globalIntEnable) // Global Int Enable

i) Data Types and Constants

ADCP_MODULE_ID Enumeration

Identifies the number of ADC Modules Supported.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_ID_1,
  ADC_NUMBER_OF_MODULES
} ADCP_MODULE_ID;
```

Members

Members	Description
ADCP_ID_1	ADC Module 1 ID
ADC_NUMBER_OF_MODULES	Number of available ADC modules.

Description

This enumeration identifies the available ADC modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

ADCP_VREF_SOURCE Enumeration

Defines the ADCP VREF Source Select.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_VREF_AVDD_AVSS,
  ADCP_VREF_VREFP_AVSS,
  ADCP_VREF_AVDD_VREFN,
  ADCP_VREF_VREFP_VREFN
} ADCP_VREF_SOURCE;
```

Members

Members	Description
ADCP_VREF_AVDD_AVSS	Reference voltage set to AVDD and AVSS
ADCP_VREF_VREFP_AVSS	Reference voltage set to VREF positive and AVSS
ADCP_VREF_AVDD_VREFN	Reference voltage set to AVDD and VREF negative
ADCP_VREF_VREFP_VREFN	Reference voltage set to VREF positive and VREF negative

Description

ADCP VREF Source Select

This data type defines the ADCP Reference Voltage (VREF) Source Select.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_CLOCK_SOURCE Enumeration

Defines the ADCP Clock Source Select.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_CLK_SRC_NONE,
  ADCP_CLK_SRC_SYSCLK,
  ADCP_CLK_SRC_RFCLK3,
  ADCP_CLK_SRC_FRC
} ADCP_CLOCK_SOURCE;
```

Members

Members	Description
ADCP_CLK_SRC_NONE	TAD clock disabled
ADCP_CLK_SRC_SYSCLK	TAD clock set to SYSCLK (TCY)
ADCP_CLK_SRC_RFCLK3	TAD clock set to REFCLK3
ADCP_CLK_SRC_FRC	TAD clock set to FRC

Description

This enumeration data type defines the ADCP Clock Source Select.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_CLASS12_INPUT_ID Enumeration

Identifies the Class 1 and Class 2 ADC Inputs.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_CLASS12_AN0,
  ADCP_CLASS12_AN1,
  ADCP_CLASS12_AN2,
  ADCP_CLASS12_AN3,
  ADCP_CLASS12_AN4,
  ADCP_CLASS12_AN5,
  ADCP_CLASS12_AN6,
  ADCP_CLASS12_AN7,
  ADCP_CLASS12_AN8,
  ADCP_CLASS12_AN9,
  ADCP_CLASS12_AN10,
  ADCP_CLASS12_AN11
} ADCP_CLASS12_INPUT_ID;
```

Members

Members	Description
ADCP_CLASS12_AN0	Analog Input AN0
ADCP_CLASS12_AN1	Analog Input AN1
ADCP_CLASS12_AN2	Analog Input AN2
ADCP_CLASS12_AN3	Analog Input AN3
ADCP_CLASS12_AN4	Analog Input AN4
ADCP_CLASS12_AN5	Analog Input AN5
ADCP_CLASS12_AN6	Analog Input AN6
ADCP_CLASS12_AN7	Analog Input AN7
ADCP_CLASS12_AN8	Analog Input AN8

ADCP_CLASS12_AN9	Analog Input AN9
ADCP_CLASS12_AN10	Analog Input AN10
ADCP_CLASS12_AN11	Analog Input AN11

Description

ADC Class 1 and Class 2 Input ID

This data type identifies the Class 1 and Class 2 ADC analog inputs.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_SH_ID Enumeration

Identifies the supported S&H circuits regardless of type.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_SH0,
  ADCP_SH1,
  ADCP_SH2,
  ADCP_SH3,
  ADCP_SH4,
  ADCP_SH5,
  ADCP_NUMBER_OF_SH
} ADCP_SH_ID;
```

Members

Members	Description
ADCP_SH0	S&H 0
ADCP_SH1	S&H 1
ADCP_SH2	S&H 2
ADCP_SH3	S&H 3
ADCP_SH4	S&H 4
ADCP_SH5	S&H 5
ADCP_NUMBER_OF_SH	Number of S&H circuits

Description

ADCP S&H Select

This enumeration identifies all supported S&H circuits.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_DSH_ID Enumeration

Identifies the supported Dedicated Sample and Hold (S&H) circuits.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_DSH0,
  ADCP_DSH1,
  ADCP_DSH2,
  ADCP_DSH3,
  ADCP_DSH4,
  ADCP_NUMBER_OF_DSH
} ADCP_DSH_ID;
```

Members

Members	Description
ADCP_DSH0	Dedicated S&H 0
ADCP_DSH1	Dedicated S&H 1
ADCP_DSH2	Dedicated S&H 2
ADCP_DSH3	Dedicated S&H 3
ADCP_DSH4	Dedicated S&H 4
ADCP_NUMBER_OF_DSH	Number of Dedicated S&H circuits

Description

ADCP Dedicated S&H Select

This enumeration identifies the supported Dedicated S&H circuits.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_SH_MODE Enumeration

Defines the available modes for the S&H.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR,
  ADCP_SH_MODE_SINGLE_ENDED_TWOS_COMP,
  ADCP_SH_MODE_DIFFERENTIAL_UNIPOLAR,
  ADCP_SH_MODE_DIFFERENTIAL_TWOS_COMP
} ADCP_SH_MODE;
```

Members

Members	Description
ADCP_SH_MODE_SINGLE_ENDED_UNIPOLAR	Single-ended input, Unipolar encoded
ADCP_SH_MODE_SINGLE_ENDED_TWOS_COMP	Single-ended input, two's compliment encoded
ADCP_SH_MODE_DIFFERENTIAL_UNIPOLAR	Differential input, Unipolar encoded
ADCP_SH_MODE_DIFFERENTIAL_TWOS_COMP	Differential input, two's compliment encoded

Description

ADCP S&H Mode

This data type defines the available modes for the S&H.

Remarks

None.

ADCP_INPUT_ID Enumeration

Identifies the available ADC Inputs.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_AN0,
  ADCP_AN1,
  ADCP_AN2,
  ADCP_AN3,
  ADCP_AN4,
  ADCP_AN5,
  ADCP_AN6,
```

```
    ADCP_AN7,
    ADCP_AN8,
    ADCP_AN9,
    ADCP_AN10,
    ADCP_AN11,
    ADCP_AN12,
    ADCP_AN13,
    ADCP_AN14,
    ADCP_AN15,
    ADCP_AN16,
    ADCP_AN17,
    ADCP_AN18,
    ADCP_AN19,
    ADCP_AN20,
    ADCP_AN21,
    ADCP_AN22,
    ADCP_AN23,
    ADCP_AN24,
    ADCP_AN25,
    ADCP_AN26,
    ADCP_AN27,
    ADCP_AN28,
    ADCP_AN29,
    ADCP_AN30,
    ADCP_AN31,
    ADCP_AN32,
    ADCP_AN33,
    ADCP_AN34,
    ADCP_AN35,
    ADCP_AN36,
    ADCP_AN37,
    ADCP_AN38,
    ADCP_AN39,
    ADCP_AN40,
    ADCP_AN41,
    ADCP_AN42,
    ADCP_AN43,
    ADCP_AN44,
    ADCP_AN45,
    ADCP_AN46,
    ADCP_AN47,
    ADCP_AN48,
    ADCP_AN49,
    ADCP_AN50,
    ADCP_AN51,
    ADCP_AN52,
    ADCP_AN53,
    ADCP_AN54,
    ADCP_AN55,
    ADCP_AN56,
    ADCP_AN57,
    ADCP_AN58,
    ADCP_AN59,
    ADCP_AN60,
    ADCP_AN61,
    ADCP_AN62,
    ADCP_AN63,
    ADCP_IVREF,
    ADCP_IVTEMP
} ADCP_INPUT_ID;
```

Members

Members	Description
ADCP_AN0	Analog Input AN0
ADCP_AN1	Analog Input AN1
ADCP_AN2	Analog Input AN2
ADCP_AN3	Analog Input AN3
ADCP_AN4	Analog Input AN4
ADCP_AN5	Analog Input AN5
ADCP_AN6	Analog Input AN6
ADCP_AN7	Analog Input AN7

ADCP_AN8	Analog Input AN8
ADCP_AN9	Analog Input AN9
ADCP_AN10	Analog Input AN10
ADCP_AN11	Analog Input AN11
ADCP_AN12	Analog Input AN12
ADCP_AN13	Analog Input AN13
ADCP_AN14	Analog Input AN14
ADCP_AN15	Analog Input AN15
ADCP_AN16	Analog Input AN16
ADCP_AN17	Analog Input AN17
ADCP_AN18	Analog Input AN18
ADCP_AN19	Analog Input AN19
ADCP_AN20	Analog Input AN20
ADCP_AN21	Analog Input AN21
ADCP_AN22	Analog Input AN22
ADCP_AN23	Analog Input AN23
ADCP_AN24	Analog Input AN24
ADCP_AN25	Analog Input AN25
ADCP_AN26	Analog Input AN26
ADCP_AN27	Analog Input AN27
ADCP_AN28	Analog Input AN28
ADCP_AN29	Analog Input AN29
ADCP_AN30	Analog Input AN30
ADCP_AN31	Analog Input AN31
ADCP_AN32	Analog Input AN32
ADCP_AN33	Analog Input AN33
ADCP_AN34	Analog Input AN34
ADCP_AN35	Analog Input AN35
ADCP_AN36	Analog Input AN36
ADCP_AN37	Analog Input AN37
ADCP_AN38	Analog Input AN38
ADCP_AN39	Analog Input AN39
ADCP_AN40	Analog Input AN40
ADCP_AN41	Analog Input AN41
ADCP_AN42	Analog Input AN42
ADCP_AN43	Analog Input AN43
ADCP_AN44	Analog Input AN44
ADCP_AN45	Analog Input AN45
ADCP_AN46	Analog Input AN46
ADCP_AN47	Analog Input AN47
ADCP_AN48	Analog Input AN48
ADCP_AN49	Analog Input AN49
ADCP_AN50	Analog Input AN50
ADCP_AN51	Analog Input AN51
ADCP_AN52	Analog Input AN52
ADCP_AN53	Analog Input AN53
ADCP_AN54	Analog Input AN54
ADCP_AN55	Analog Input AN55
ADCP_AN56	Analog Input AN56
ADCP_AN57	Analog Input AN57
ADCP_AN58	Analog Input AN58
ADCP_AN59	Analog Input AN59
ADCP_AN60	Analog Input AN60
ADCP_AN61	Analog Input AN61
ADCP_AN62	Analog Input AN62

ADCP_AN63	Analog Input AN63
ADCP_IVREF	Analog Input for Internal Voltage Reference
ADCP_IVTEMP	Analog Input for Internal Temperature Sensor

Description

ADC Input ID

This data type identifies the available ADC Inputs, regardless of Class.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all device.

ADCP_DCMP_ID Enumeration

Identifies the supported Digital Comparators.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_DCMP1,
  ADCP_DCMP2,
  ADCP_DCMP3,
  ADCP_DCMP4,
  ADCP_DCMP5,
  ADCP_DCMP6,
  ADCP_NUMBER_OF_DCMP
} ADCP_DCMP_ID;
```

Members

Members	Description
ADCP_DCMP1	DCMP1
ADCP_DCMP2	DCMP2
ADCP_DCMP3	DCMP3
ADCP_DCMP4	DCMP4
ADCP_DCMP5	DCMP5
ADCP_DCMP6	DCMP6
ADCP_NUMBER_OF_DCMP	Number of Digital Comparators

Description

ADCP Digital Comparator

This enumeration identifies all supported Digital Comparators for this ADCP module.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_ODFLTR_ID Enumeration

Identifies the supported Oversampling Digital Filters.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_ODFLTR1,
  ADCP_ODFLTR2,
  ADCP_ODFLTR3,
  ADCP_ODFLTR4,
  ADCP_ODFLTR5,
  ADCP_ODFLTR6,
  ADCP_NUMBER_OF_ODFLTR
} ADCP_ODFLTR_ID;
```

Members

Members	Description
ADCP_ODFLTR1	ODFLTR1
ADCP_ODFLTR2	ODFLTR2
ADCP_ODFLTR3	ODFLTR3
ADCP_ODFLTR4	ODFLTR4
ADCP_ODFLTR5	ODFLTR5
ADCP_ODFLTR6	ODFLTR6
ADCP_NUMBER_OF_ODFLTR	Number of Oversampling Digital Filters

Description

ADCP Oversampling Digital Filter

This enumeration identifies all supported Digital Filters for this ADCP module.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_ODFLTR_OSR Enumeration

Identifies the supported Digital Filter oversampling ratios.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_ODFLTR_4X,
  ADCP_ODFLTR_16X,
  ADCP_ODFLTR_64X,
  ADCP_ODFLTR_256X,
  ADCP_ODFLTR_2X,
  ADCP_ODFLTR_8X,
  ADCP_ODFLTR_32X,
  ADCP_ODFLTR_128X
} ADCP_ODFLTR_OSR;
```

Members

Members	Description
ADCP_ODFLTR_4X	4x oversampling, shift sum 1 bit to right, output data is 13 bits
ADCP_ODFLTR_16X	16x oversampling, shift sum 2 bits to right, output data is 14 bits
ADCP_ODFLTR_64X	64x oversampling, shift sum 3 bits to right, output data is 15 bits
ADCP_ODFLTR_256X	256x oversampling, shift sum 4 bits to right, output data is 16 bits
ADCP_ODFLTR_2X	2x oversampling, shift sum 0 bits to right, output data is in 12.1 format
ADCP_ODFLTR_8X	8x oversampling, shift sum 1 bit to right, output data is in 13.1 format
ADCP_ODFLTR_32X	32x oversampling, shift sum 2 bits to right, output data is in 14.1 format
ADCP_ODFLTR_128X	128x oversampling, shift sum 3 bit to right, output data is in 15.1 format

Description

ADCP Oversampling Ratio

This enumeration identifies all supported Digital Filter oversampling ratios for this ADCP module. Oversampling ratios determine the number of samples used to generate a single output and the resulting resolution and format.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_SCAN_TRG_SRC Enumeration

Defines the ADCP Channel Scan Trigger Source Selections.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_SCAN_TRG_SRC_NONE,
  ADCP_SCAN_TRG_SRC_SOFTWARE,
  ADCP_SCAN_TRG_SRC_INT0,
  ADCP_SCAN_TRG_SRC_TMR1_MATCH,
  ADCP_SCAN_TRG_SRC_TMR3_MATCH,
  ADCP_SCAN_TRG_SRC_TMR5_MATCH,
  ADCP_SCAN_TRG_SRC_OCMP1,
  ADCP_SCAN_TRG_SRC_OCMP3,
  ADCP_SCAN_TRG_SRC_OCMP5,
  ADCP_SCAN_TRG_SRC_COMP1_COUT,
  ADCP_SCAN_TRG_SRC_COMP2_COUT
} ADCP_SCAN_TRG_SRC;
```

Members

Members	Description
ADCP_SCAN_TRG_SRC_NONE	No scan trigger source is selected
ADCP_SCAN_TRG_SRC_SOFTWARE	Global Software trigger selected as scan trigger source
ADCP_SCAN_TRG_SRC_INT0	Interrupt 0 (INT0) selected as scan trigger source
ADCP_SCAN_TRG_SRC_TMR1_MATCH	Timer 1 Match (TMR1) selected as scan trigger source
ADCP_SCAN_TRG_SRC_TMR3_MATCH	Timer 3 Match (TMR3) selected as scan trigger source
ADCP_SCAN_TRG_SRC_TMR5_MATCH	Timer 5 Match (TMR5) selected as scan trigger source
ADCP_SCAN_TRG_SRC_OCMP1	Output Compare 1 (OCMP1) selected as scan trigger source
ADCP_SCAN_TRG_SRC_OCMP3	Output Compare 3 (OCMP3) selected as scan trigger source
ADCP_SCAN_TRG_SRC_OCMP5	Output Compare 5 (OCMP5) selected as scan trigger source
ADCP_SCAN_TRG_SRC_COMP1_COUT	Analog Comparator 1 (COUT) selected as scan trigger source
ADCP_SCAN_TRG_SRC_COMP2_COUT	Analog Comparator 2 (COUT) selected as scan trigger source

Description

ADCP Channel Scan Trigger Source Select

This data type defines the ADCP Channel Scan Trigger Source Selections.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

ADCP_TRG_SRC Enumeration

Defines the ADCP Trigger Source Selections.

File

help_plib_adcp.h

C

```
typedef enum {
  ADCP_TRG_SRC_NONE,
  ADCP_TRG_SRC_SOFTWARE,
  ADCP_TRG_SRC_SCAN_TRG,
  ADCP_TRG_SRC_INT0,
  ADCP_TRG_SRC_TMR1_MATCH,
  ADCP_TRG_SRC_TMR3_MATCH,
  ADCP_TRG_SRC_TMR5_MATCH,
  ADCP_TRG_SRC_OCMP1,
  ADCP_TRG_SRC_OCMP3,
  ADCP_TRG_SRC_OCMP5,
  ADCP_TRG_SRC_COMP1_COUT,
  ADCP_TRG_SRC_COMP2_COUT
} ADCP_TRG_SRC;
```

Members

Members	Description
ADCP_TRG_SRC_NONE	No trigger source is selected
ADCP_TRG_SRC_SOFTWARE	Global Software trigger selected as the trigger source
ADCP_TRG_SRC_SCAN_TRG	Use the Channel Scan Trigger as the trigger source (input is part of ch scan)
ADCP_TRG_SRC_INT0	Interrupt 0 (INT0) selected as the trigger source
ADCP_TRG_SRC_TMR1_MATCH	Timer 1 Match (TMR1) selected as the trigger source
ADCP_TRG_SRC_TMR3_MATCH	Timer 3 Match (TMR3) selected as the trigger source
ADCP_TRG_SRC_TMR5_MATCH	Timer 5 Match (TMR5) selected as the trigger source
ADCP_TRG_SRC_OCMP1	Output Compare 1 (OCMP1) selected as the trigger source
ADCP_TRG_SRC_OCMP3	Output Compare 3 (OCMP3) selected as the trigger source
ADCP_TRG_SRC_OCMP5	Output Compare 5 (OCMP5) selected as the trigger source
ADCP_TRG_SRC_COMP1_COUT	Analog Comparator 1 (COUT) selected as the trigger source
ADCP_TRG_SRC_COMP2_COUT	Analog Comparator 2 (COUT) selected as the trigger source

Description

ADCP Trigger Source Select

This data type defines the ADCP Trigger Source Selections.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

AN_SELECT Union

This union of structures is provided to simply selection of analog inputs for inclusion in a particular function.

File

plib_adcp.h

C

```
union AN_SELECT {
  int ret_val;
  struct {
    uint32_t lowWord;
    uint32_t highWord;
  };
  struct {
    uint32_t AN0 : 1;
    uint32_t AN1 : 1;
    uint32_t AN2 : 1;
    uint32_t AN3 : 1;
    uint32_t AN4 : 1;
    uint32_t AN5 : 1;
    uint32_t AN6 : 1;
    uint32_t AN7 : 1;
    uint32_t AN8 : 1;
    uint32_t AN9 : 1;
    uint32_t AN10 : 1;
    uint32_t AN11 : 1;
    uint32_t AN12 : 1;
    uint32_t AN13 : 1;
    uint32_t AN14 : 1;
    uint32_t AN15 : 1;
    uint32_t AN16 : 1;
    uint32_t AN17 : 1;
    uint32_t AN18 : 1;
    uint32_t AN19 : 1;
    uint32_t AN20 : 1;
    uint32_t AN21 : 1;
    uint32_t AN22 : 1;
    uint32_t AN23 : 1;
    uint32_t AN24 : 1;
    uint32_t AN25 : 1;
    uint32_t AN26 : 1;
    uint32_t AN27 : 1;
```

```
      uint32_t AN28 : 1;
      uint32_t AN29 : 1;
      uint32_t AN30 : 1;
      uint32_t AN31 : 1;
      uint32_t AN32 : 1;
      uint32_t AN33 : 1;
      uint32_t AN34 : 1;
      uint32_t AN35 : 1;
      uint32_t AN36 : 1;
      uint32_t AN37 : 1;
      uint32_t AN38 : 1;
      uint32_t AN39 : 1;
      uint32_t AN40 : 1;
      uint32_t AN41 : 1;
      uint32_t AN42 : 1;
      uint32_t AN43 : 1;
      uint32_t AN44 : 1;
      uint32_t AN45 : 1;
      uint32_t AN46 : 1;
      uint32_t AN47 : 1;
      uint32_t AN48 : 1;
      uint32_t AN49 : 1;
      uint32_t AN50 : 1;
      uint32_t AN51 : 1;
      uint32_t AN52 : 1;
      uint32_t AN53 : 1;
      uint32_t AN54 : 1;
      uint32_t AN55 : 1;
      uint32_t AN56 : 1;
      uint32_t AN57 : 1;
      uint32_t AN58 : 1;
      uint32_t AN59 : 1;
      uint32_t AN60 : 1;
      uint32_t AN61 : 1;
      uint32_t AN62 : 1;
      uint32_t AN63 : 1;
    };
  };
```

Description

ADCP AN Select Union

The structure labels identify the analog inputs associated with each bit when initializing the structure. Unsigned 32-bit integers, lowWord and highWord are used as arguments to the function.

Remarks

See the Channel Scanning example in the ADCP Help documentation regarding use of this type define.

AN_READY Type

This union (identical to the AN_SELECT union) is used as the return value by the PLIB_ADCP_Result_Ready function.

File

plib_adcp.h

C

```
typedef AN_SELECT AN_READY;
```

Description

ADCP AN Ready Union

The structure labels identify the analog inputs associated with each bit for testing ready status of individual inputs. Unsigned 32-bit integers, lowWord and highWord allow for testing ready status of groups of bits using a mask.

Remarks

See the Simultaneous Sampling or Channel Scanning examples in the ADCP Help documentation regarding use of this type define.

Files

Files

Name	Description
plib_adcp.h	ADCP Peripheral Library Interface Header for ADCP common definitions
help_plib_adcp.h	ADCP Peripheral Library interface header file template.

Description

This section lists the source and header files used by the library.

plib_adcp.h

ADCP Peripheral Library Interface Header for ADCP common definitions

Functions

	Name	Description
	PLIB_ADCP_AlternateInputSelect	Selects the alternate physical input for the specified dedicated (Class 1) S&H.
	PLIB_ADCP_CalibrationStart	Initiates Pipelined ADC Calibration.
	PLIB_ADCP_ChannelScanConfigure	Selects input to include in Channel Scan Mode
	PLIB_ADCP_Class12TriggerConfigure	Configures a Class 1 or Class 2 Trigger Source.
	PLIB_ADCP_Configure	Configures the Pipelined ADC module including the ADC calibration registers.
	PLIB_ADCP_DefaultInputSelect	Selects the default physical input for the specified dedicated (Class 1) S&H.
	PLIB_ADCP_DigCmpAIdGet	Returns the Analog Input ID of for the Comparator Event
	PLIB_ADCP_DigCmpConfig	Configures the Digital Comparator on the Pipelined ADC converter.
	PLIB_ADCP_DigCmpDisable	Disables the Digital Comparator in the Pipelined ADC module.
	PLIB_ADCP_DigCmpEnable	Enables the Digital Comparator in the Pipelined ADC module.
	PLIB_ADCP_Disable	Pipelined ADC module is disabled (turned OFF).
	PLIB_ADCP_Enable	Enables the Pipelined ADC module.
	PLIB_ADCP_ExistsCalibration	Identifies whether the Calibration feature exists on the ADCP module.
	PLIB_ADCP_ExistsChannelScan	Identifies whether the ChannelScan feature exists on the ADCP module.
	PLIB_ADCP_ExistsConfiguration	Identifies whether the Configuration feature exists on the ADCP module.
	PLIB_ADCP_ExistsConversionResults	Identifies whether the ConversionResults feature exists on the ADCP module.
	PLIB_ADCP_ExistsDigCmp	Identifies whether the DigitalComparator feature exists on the ADCP module.
	PLIB_ADCP_ExistsEnableControl	Identifies whether the EnableControl feature exists on the ADCP module.
	PLIB_ADCP_ExistsInputSelect	Identifies whether the InputSelect feature exists on the ADCP module.
	PLIB_ADCP_ExistsLowPowerControl	Identifies whether the LowPowerControl feature exists on the ADCP module.
	PLIB_ADCP_ExistsModeSelect	Identifies whether the ModeSelect feature exists on the ADCP module.
	PLIB_ADCP_ExistsOsampDigFilter	Identifies whether the OsampDigitalFilter feature exists on the ADCP module.
	PLIB_ADCP_ExistsReadyStatus	Identifies whether the ReadyStatus feature exists on the ADCP module.
	PLIB_ADCP_ExistsTriggering	Identifies whether the Triggering feature exists on the ADCP module.
	PLIB_ADCP_GlobalSoftwareTrigger	Initiates a Global Software Trigger on the specified module.
	PLIB_ADCP_IndividualTrigger	Triggers an individual channel independent of the configured trigger source
	PLIB_ADCP_LowPowerStateEnter	Places the Pipelined ADC module in a low power state.
	PLIB_ADCP_LowPowerStateExit	Takes the Pipelined ADC module out of low power state and puts it into an operational mode.
	PLIB_ADCP_LowPowerStateGet	Returns the state of the low power setting.
	PLIB_ADCP_ModuleIsReady	Returns the overall ready status of the module.
	PLIB_ADCP_OsampDigFilterConfig	Configures the Oversampling Digital Filter on the Pipelined ADC converter.
	PLIB_ADCP_OsampDigFilterDataGet	Fetches the data result from the Oversampling Digital Filter.
	PLIB_ADCP_OsampDigFilterDataRdy	Determines if the Oversampling Digital Filter has data ready.
	PLIB_ADCP_OsampDigFilterDisable	Disables the Oversampling Digital Filter in the Pipelined ADC module.
	PLIB_ADCP_OsampDigFilterEnable	Enables the Oversampling Digital Filter in the Pipelined ADC module
	PLIB_ADCP_ResultGet	Returns a Pipelined ADC conversion result.
	PLIB_ADCP_ResultReady	Returns the ADC conversion result ready bits for the module.
	PLIB_ADCP_ResultReadyGrpIntConfigure	Selects input to include in global interrupt.

	PLIB_ADCP_SHModeSelect	Selects the mode for the specified S&H.

Types

	Name	Description
	AN_READY	This union (identical to the AN_SELECT union) is used as the return value by the PLIB_ADCP_Result_Ready function.

Unions

	Name	Description
	AN_SELECT	This union of structures is provided to simply selection of analog inputs for inclusion in a particular function.

Description

Pipelined Analog-to-Digital Converter (ADCP) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the ADCP PLIB for all families of Microchip microcontrollers. The definitions in this file are common to the ADCP peripheral.

File Name

plib_adcp.h

Company

Microchip Technology Inc.

help_plib_adcp.h

ADCP Peripheral Library interface header file template.

Enumerations

	Name	Description
	ADCP_CLASS12_INPUT_ID	Identifies the Class 1 and Class 2 ADC Inputs.
	ADCP_CLOCK_SOURCE	Defines the ADCP Clock Source Select.
	ADCP_DCMP_ID	Identifies the supported Digital Comparators.
	ADCP_DSH_ID	Identifies the supported Dedicated Sample and Hold (S&H) circuits.
	ADCP_INPUT_ID	Identifies the available ADC Inputs.
	ADCP_MODULE_ID	Identifies the number of ADC Modules Supported.
	ADCP_ODFLTR_ID	Identifies the supported Oversampling Digital Filters.
	ADCP_ODFLTR_OSR	Identifies the supported Digital Filter oversampling ratios.
	ADCP_SCAN_TRG_SRC	Defines the ADCP Channel Scan Trigger Source Selections.
	ADCP_SH_ID	Identifies the supported S&H circuits regardless of type.
	ADCP_SH_MODE	Defines the available modes for the S&H.
	ADCP_TRG_SRC	Defines the ADCP Trigger Source Selections.
	ADCP_VREF_SOURCE	Defines the ADCP VREF Source Select.

Description

ADCP Peripheral Library Interface Header File Template

This file is used by the DOM project.

File Name

help_plib_adcp.h

Company

Microchip Technology Inc.

BMX Peripheral Library

This section describes the Bus Matrix (BMX) Peripheral Library.

Introduction

This library provides a low-level abstraction of the Bus Matrix (BMX) modules on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The Bus Matrix is essentially a high-speed switch that establishes connections between devices. Devices can be either master (initiator) or slave (target).

The Bus Matrix also performs these other important functions:

- Partition memory - The Bus Matrix provides the ability to partition both RAM and Flash memory into kernel and user areas. It also controls the type of access, program or data, for each memory area.
- Enable/disable program Flash memory cache for Direct Memory Access (DMA)
- Enable/disable bus error exceptions - The Bus Matrix provides the ability to disable bus error exceptions for debugging. They are enabled by default.
- Set Bus Arbitration mode - The Bus Matrix provides the ability to select between arbitration modes. The arbitration modes control the order and priority of initiator access to peripherals.

Using the Library

This topic describes the basic architecture of the BMX Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_bmx.h`

The interface to the BMX library is defined in the `plib_bmx.h` header file, which is included by the `peripheral.h` peripheral library header file. Any C language source (`.c`) file that uses the BMX library must include `peripheral.h`.

Library File:

The BMX Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

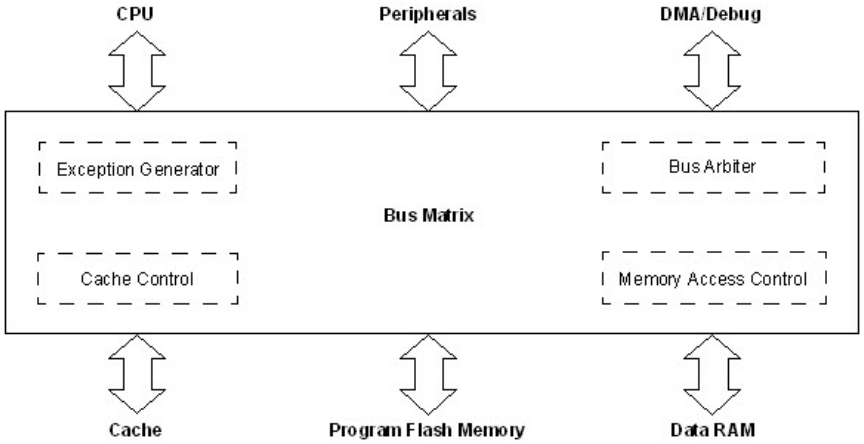
Hardware Abstraction Model

This library provides a low-level abstraction of the BMX module on Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The following figure shows the Bus Matrix hardware abstraction model.

Hardware Abstraction Model



The BMX Peripheral Library provides interface routines to interact with the blocks shown in the previous diagram.

The **Exception Generator** generates a bus error exception for unmapped address accesses from all Bus Matrix initiators:

- Initiator Expansion Interface (IXI) shared bus
- In-Circuit Debug (ICD) unit

- Direct Memory Access (DMA) controller
- CPU data bus
- CPU instruction bus

Each of the initiators can be individually enabled or disabled with Exception Generator peripheral library routines. All are enabled by default.

The **Cache Control** simply enables/disables program Flash memory (PFM) data cacheability for DMA accesses. The default setting is disabled. When enabled, the PCACHE module must have data caching enabled.

The **Bus Arbitrator** assigns priority levels to bus initiators following one of three priority schemes:

- Fixed, with CPU higher than DMA
- Fixed, with DMA higher than CPU (default)
- Rotating between four priority sequences

The **Memory Access Control** partitions program Flash memory and RAM into kernel and user areas. The RAM can be further partitioned into data and program segments for both kernel and user areas. By default, the full PFM and RAM are mapped to Kernel mode.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the BMX module.

Library Interface Section	Description
Bus Exception Routines	Provide an interface to individually enable or disable initiator access to unmapped memory.
Program Flash Cache Routines	Provide an interface to enable or disable caching of DMA accesses to Program Flash Memory.
Arbiter Routines	Provide an interface to set or change the bus initiator priority sequence.
Memory Access Control Routines	Provide an interface to partition Program Flash and Data Memories into user and kernel regions. Provide an interface to read the size of the Boot Flash, Program Flash and Data SRAM memories.
Feature Existence Routines	These functions determine whether or not a particular feature is supported by the device.

How the Library Works

The Bus Matrix is initialized in the start-up code, before an application is invoked. Most applications will not need to modify the Bus Matrix default settings. The BMX Peripheral Library is available for those applications requiring more complex memory partitioning, including operating systems and applications requiring specialized memory access. The ability to disable exceptions for unmapped memory access may be useful for debugging, but is not recommended for normal use. Changes to the arbitration scheme may improve performance for CPU or DMA intensive applications.

Exception Generator

Any unmapped address access by a Bus Matrix initiator will by default generate an exception. These exceptions may be individually enabled or disabled with the peripheral library.

Example: Enable CPU instruction exception

```
// Enable CPU instruction exception
PLIB_BMX_ISBusExceptionEnable();
```

Cache Control

This peripheral library function enables program Flash memory (data) cacheability for DMA accesses. Data caching must be enabled in the Pcache. Hits are still read from the cache, but misses do not update the cache. This function is disabled by default.

Example: Enable PFM Cacheability for DMA access

```
PLIB_BMX_EnablePfmCacheDma()
```

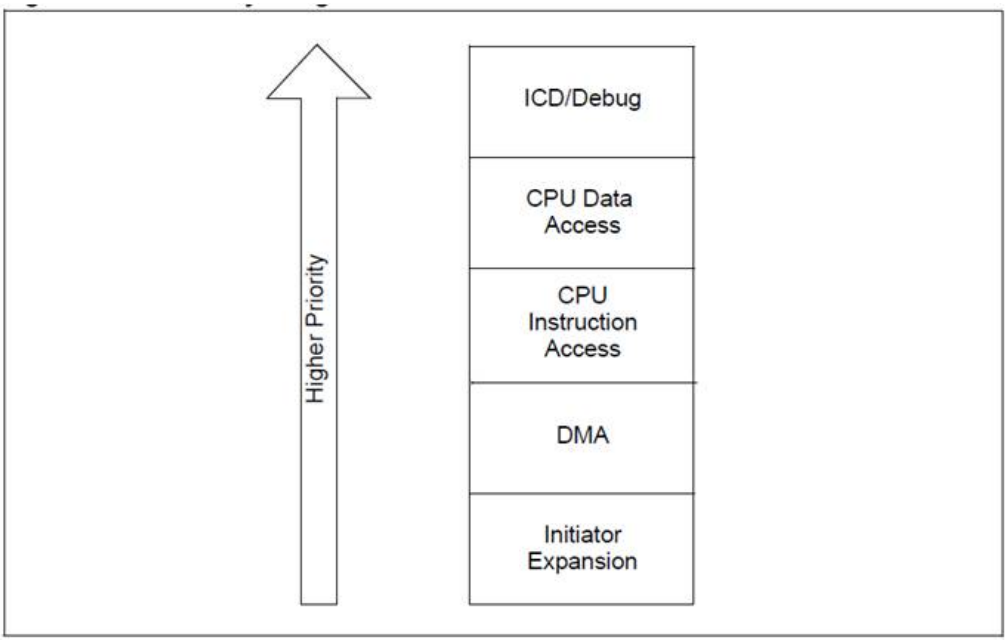
Bus Arbiter

Since there can be more than one initiator attempting to access the same target, an arbitration scheme must be used to control access to the target. The arbitration modes assign priority levels to all the initiators. The initiator with the higher priority level will always win target access over a lower priority initiator. The BMX Bus Arbitrator allows the programmer to select from one of three arbitration modes.

PLIB_BMX_ARB_MODE_INST

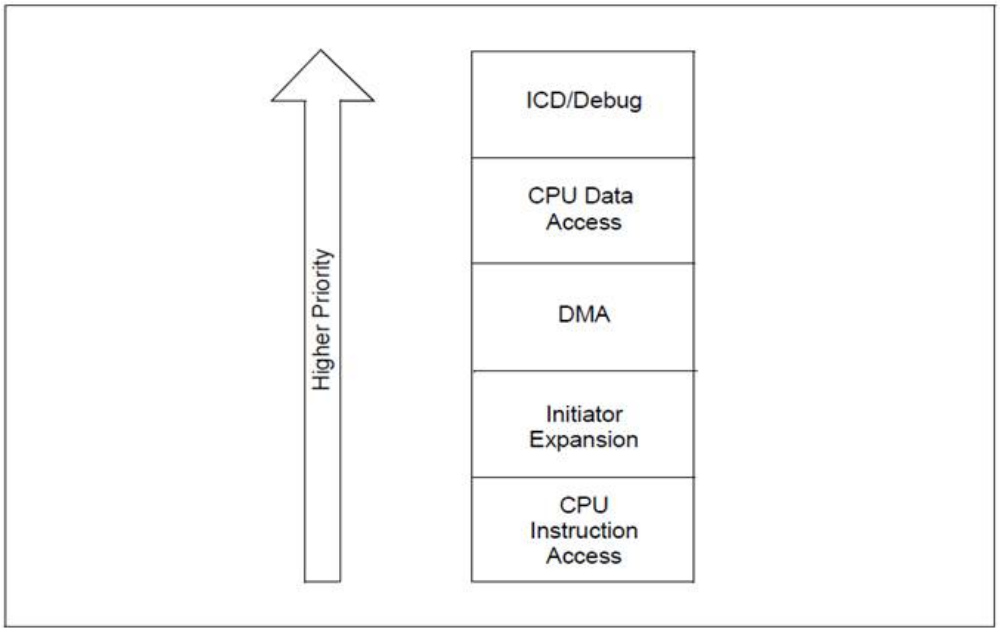
Arbitration mode 0 is a fixed priority scheme, with the CPU given higher priority than DMA. This mode can starve the DMA, so choose this mode

when DMA is not being used.



PLIB_BMX_ARB_MODE_DMA

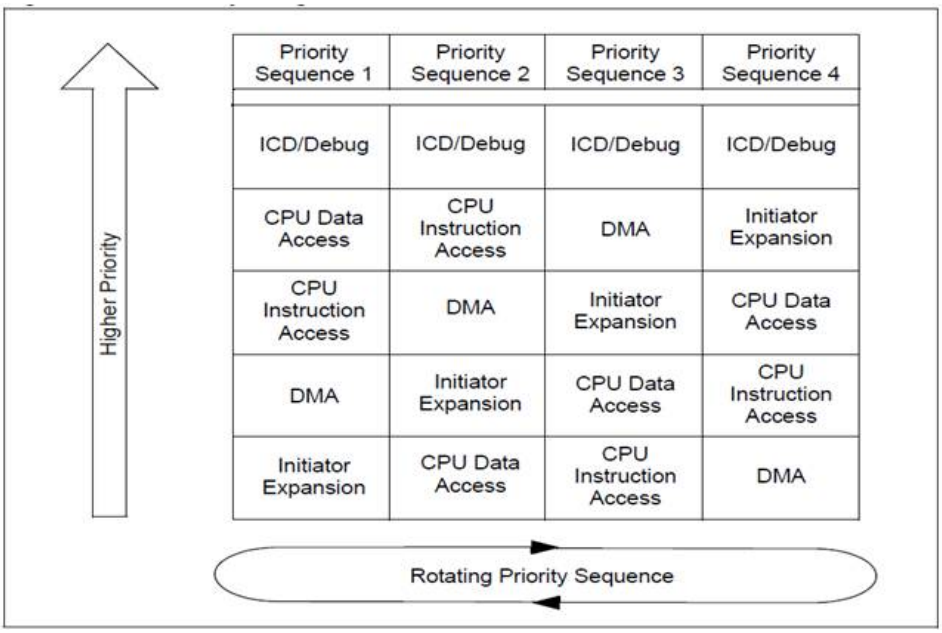
Arbitration Mode 1 is a fixed priority scheme like Mode 0, except that the CPU is always the lowest priority. Mode 1 is the default mode.



PLIB_BMX_ARB_MODE_ROT

Mode 2 arbitration supports rotating priority assignments to all initiators. Instead of a fixed priority assignment, each initiator is assigned the highest priority in a rotating fashion. In this mode, the rotating priority is applied with the following exceptions:

1. CPU data is always selected over CPU instruction.
2. ICD is always the highest priority.
3. When the CPU is processing an exception (EXL = 1) or an error (ERL = 1), the arbitrator temporarily reverts to Mode 0.



Note: Priority Sequence 2 is not selected in the rotation priority scheme if there is a pending CPU data access. In this case, once the data access is complete, Sequence 2 is selected.

Example: Set Bus Arbitration Mode to Rotating Priority

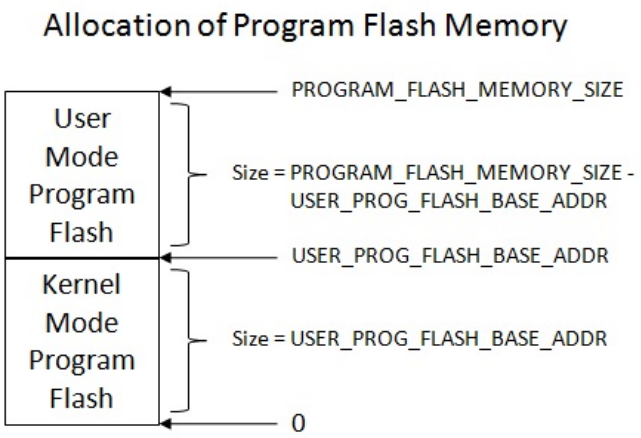
```
PLIB_BMX_SetArbitrationMode(PLIB_BMX_ARB_ROT);
```

Memory Access Control

The Bus Matrix allows the programmer to partition program Flash memory into user and kernel partitions. It also allows the programmer to partition data RAM into user and kernel partitions, each of which can be sub-divided into program and data partitions. At reset, the entire program Flash memory is one kernel partition and the entire data RAM is one kernel data partition.

Program Flash Memory Partitioning

Partitioning of the program Flash memory is accomplished by setting the offset of the user partition within the program Flash memory using the PLIB_BMX_ProgramFlashPartitionSet function. The size of the program Flash memory varies among devices, but can be retrieved with the PLIB_BMX_ProgramFlashMemorySizeGet function. The offset must be a multiple of the program Flash memory block size. The program Flash memory block size may vary among devices. If programmed with a value that is not a multiple of the block size, the value will automatically be truncated to a block size boundary. At reset, the entire program Flash memory is mapped to Kernel mode.



Example: Create a User Mode Partition of 12K in Program Flash Memory

```
size_t pfmSize;
size_t userSize = (6 * PLIB_PCACHE_PFM_BLOCK_SIZE);
size_t userOffset;

// Get size of PFM
pfmSize = PLIB_BMX_ProgramFlashMemorySizeGet();
userOffset = pfmSize - userSize;
```

```
if (userOffset > 0)
{
    PLIB_BMX_ProgramFlashPartitionSet(userOffset);
}
```

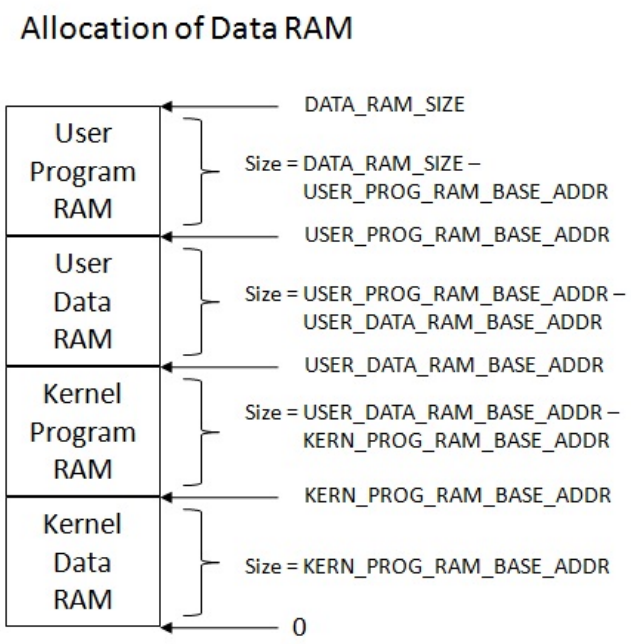
Data RAM Memory Partitioning

The Data RAM Can be partitioned into four sections:

- Kernel Data
- Kernel Program
- User Data
- User Program

Similar to the program Flash memory partitioning, the data RAM partitions are created by setting the base addresses of the various partitions. The BMX Peripheral Library provides a function to set all of the partitions at once. The size of the data RAM varies among devices, but may be retrieved with the PLIB_BMX_DataRAMSizeGet function. The partitions must be a multiple of the Data RAM memory block size. The data RAM memory block size may vary among devices. If programmed with a value that is not a multiple of the block size, the partition will automatically be truncated to a block size boundary. At reset, the entire data RAM is mapped to Kernel mode.

Note: At Reset, the entire data RAM is mapped to Kernel mode and all of the offsets are set to zero. To partition the data RAM, all of the offsets must be programmed. If any of the offsets are set to zero, the entire data RAM is allocated to kernel data.



Example: RAM Partitioned as Kernel Data, Kernel Program, User Data and User Program

```
//Total Data RAM = 32KB.
size_t totalRamSize;
size_t kernProgOffset;
size_t userDataOffset;
size_t userProgOffset;

size_t kernDataRamSize = (3 * PLIB_PCACHE_DRM_BLOCK_SIZE); //Kernel Data RAM = 6KB
size_t kernProgRamSize = (2 * PLIB_PCACHE_DRM_BLOCK_SIZE); //Kernel Program RAM = 4KB
size_t userDataRamSize = (6 * PLIB_PCACHE_DRM_BLOCK_SIZE); //User Data RAM = 12KB
size_t userProgRamSize = (5 * PLIB_PCACHE_DRM_BLOCK_SIZE); //User Program RAM = 10KB

//Get Size of Data RAM
totalRamSize = PLIB_BMX_DataRAMSizeGet();

//Verify our partition sizes fit our RAM
if ((kernDataRamSize + kernProgRamSize + userDataRamSize + userProgRamSize) != totalRamSize)
{
    printf("RAM Partitioning Error\n");
    return -1;
}

kernProgOffset = kernDataRamSize;
```

```
userDataOffset = kernDataRamSize + kernProgRamSize;
userProgOffset = userDataOffset + userDataRamSize;

PLIB_BMX_DataRAMPartitionSet(kernProgOffset, userDataOffset, userProgOffset);
```

Example: Determine the size of Data RAM partitions

```
size_t kernProgOffset;
size_t userDataOffset;
size_t userProgOffset;
size_t kernDataPart;
size_t kernProgPart;
size_t userDataPart;
size_t userProgPart;
size_t totalRamSize;
totalRamSize = PLIB_BMX_DataRAMSizeGet();
kernProgOffset = PLIB_BMX_DataRAMKernelProgramOffsetGet();
userDataOffset = PLIB_BMX_DataRAMUserDataOffsetGet();
userProgOffset = PLIB_BMX_DataRAMUserProgramOffsetGet();

kernDataPart = kernProgOffset;
kernProgPart = userDataOffset - kernProgOffset;
userDataPart = userProgOffset - userDataOffset;
userProgPart = totalRamSize - userProgOffset;
```

Configuring the Library

The library is configured for the supported BMX module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Bus Exception Functions

	Name	Description
	PLIB_BMX_BusExceptionDataDisable	Disables the data bus exception.
	PLIB_BMX_BusExceptionDataEnable	Enables the Data bus exception.
	PLIB_BMX_BusExceptionDMADisable	Disables the DMA bus exception.
	PLIB_BMX_BusExceptionDMAEnable	Enables the DMA bus exception.
	PLIB_BMX_BusExceptionICDDisable	Disables the ICD bus exception.
	PLIB_BMX_BusExceptionICDEnable	Enables the ICD bus exception.
	PLIB_BMX_BusExceptionInstructionDisable	Disables the instruction bus exception.
	PLIB_BMX_BusExceptionInstructionEnable	Enables the instruction bus exception.
	PLIB_BMX_BusExceptionIXIDisable	Disables the IXI bus exception.
	PLIB_BMX_BusExceptionIXIEnable	Enables the IXI bus exception.

b) Program Flash Cache Functions

	Name	Description
	PLIB_BMX_ProgramFlashMemoryCacheDmaDisable	Disables the bus matrix program Flash cacheability for DMA.
	PLIB_BMX_ProgramFlashMemoryCacheDmaEnable	Enables the bus matrix program Flash cacheability for DMA.

c) Arbiter Functions

	Name	Description
	PLIB_BMX_ArbitrationModeGet	Returns the bus matrix arbitration mode.
	PLIB_BMX_ArbitrationModeSet	Sets the bus matrix arbitration mode.

d) Memory Access Control Functions

	Name	Description
	PLIB_BMX_DataRAMKernelProgramOffsetGet	Gets the offset (from start of RAM) of the kernel program partition.
	PLIB_BMX_DataRAMPartitionSet	Sets the size of the kernel and user partitions in data RAM memory.
	PLIB_BMX_DataRAMSizeGet	Gets the size of data RAM memory.
	PLIB_BMX_DataRAMUserDataOffsetGet	Gets the offset (from start of RAM) of the user data partition.

	PLIB_BMX_DataRAMUserProgramOffsetGet	Gets the offset (from start of RAM) of the user program partition.
	PLIB_BMX_DataRamWaitStateGet	Returns the number of data RAM Wait states.
	PLIB_BMX_DataRamWaitStateSet	Sets the number of data RAM Wait states.
	PLIB_BMX_ProgramFlashBootSizeGet	Gets the size of boot program Flash memory.
	PLIB_BMX_ProgramFlashMemorySizeGet	Gets the size of program Flash memory.
	PLIB_BMX_ProgramFlashPartitionGet	Gets the size of the kernel partition in program Flash memory.
	PLIB_BMX_ProgramFlashPartitionSet	Sets the size of the kernel and user partitions in program Flash memory.

e) Feature Existence Functions

	Name	Description
	PLIB_BMX_ExistsArbitrationMode	Identifies that the ArbitrationMode feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionData	Identifies that the BusExceptionData feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionDMA	Identifies that the BusExceptionDMA feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionICD	Identifies that the BusExceptionICD feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionInstruction	Identifies that the BusExceptionInstruction feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionIXI	Identifies that the BusExceptionIXI feature exists on the BMX module.
	PLIB_BMX_ExistsDataRAMPartition	Identifies that the DataRAMPartition feature exists on the BMX module.
	PLIB_BMX_ExistsDataRAMSize	Identifies that the DataRAMSize feature exists on the BMX module.
	PLIB_BMX_ExistsDataRamWaitState	Identifies that the DataRamWaitState feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashBootSize	Identifies that the ProgramFlashBootSize feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashMemoryCacheDma	Identifies that the ProgramFlashMemoryCacheDma feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashMemorySize	Identifies that the ProgramFlashMemorySize feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashPartition	Identifies that the ProgramFlashPartition feature exists on the BMX module.

f) Data Types and Constants

	Name	Description
	BMX_MODULE_ID	Lists the possible Module IDs for the bus matrix.
	PLIB_BMX_ARB_MODE	Lists the possible arbitration modes for the bus matrix.
	PLIB_BMX_DATA_RAM_WAIT_STATES	Defines the number of data RAM wait states for address setup.
	PLIB_BMX_EXCEPTION_SRC	Defines which events trigger a bus exception.
	PLIB_BMX_DRM_BLOCK_SIZE	Defines the minimum partition block size in data RAM memory.
	PLIB_BMX_PFM_BLOCK_SIZE	Defines the minimum partition block size in program Flash memory.

Description

This section describes the Application Programming Interface (API) functions of the BMX Peripheral Library.

Refer to each section for a detailed description.

a) Bus Exception Functions

PLIB_BMX_BusExceptionDataDisable Function

Disables the data bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionDataDisable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function disables the data bus exception.

Remarks

This function implements an operation of the BusExceptionData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionData in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionDataDisable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_BMX_BusExceptionDataDisable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionDataEnable Function

Enables the Data bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionDataEnable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function enables the Data bus exception.

Remarks

This function implements an operation of the BusExceptionData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionData in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionDataEnable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_BMX_BusExceptionDataEnable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionDMADisable Function

Disables the DMA bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionDMADisable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function disables the DMA bus exception.

Remarks

This function implements an operation of the BusExceptionDMA feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionDMA in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionDMADisable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_BMX_BusExceptionDMADisable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionDMAEnable Function

Enables the DMA bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionDMAEnable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function enables the DMA bus exception.

Remarks

This function implements an operation of the BusExceptionDMA feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionDMA in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionDMAEnable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_BMX_BusExceptionDMAEnable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionICDDisable Function

Disables the ICD bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionICDDisable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function disables the ICD bus exception.

Remarks

This function implements an operation of the BusExceptionICD feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionICD in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionICDDisable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_BMX_BusExceptionICDDisable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionICDEnable Function

Enables the ICD bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionICDEnable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function enables the ICD bus exception.

Remarks

This function implements an operation of the BusExceptionICD feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionICD in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionICDEnable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_BMX_BusExceptionICDEnable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionInstructionDisable Function

Disables the instruction bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionInstructionDisable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function disables the instruction bus exception.

Remarks

This function implements an operation of the BusExceptionInstruction feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionInstruction in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionInstructionDisable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_BMX_BusExceptionInstructionDisable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionInstructionEnable Function

Enables the instruction bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionInstructionEnable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function enables the instruction bus exception.

Remarks

This function implements an operation of the BusExceptionInstruction feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionInstruction in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionInstructionEnable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_BMX_BusExceptionInstructionEnable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionIXIDisable Function

Disables the IXI bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionIXIDisable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function disables the IXI bus exception.

Remarks

This function implements an operation of the BusExceptionIXI feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionIXI in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionIXIDisable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_BMX_BusExceptionIXIDisable(BMX_MODULE_ID index)

PLIB_BMX_BusExceptionIXIEnable Function

Enables the IXI bus exception.

File

plib_bmx.h

C

```
void PLIB_BMX_BusExceptionIXIEnable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function enables the IXI bus exception.

Remarks

This function implements an operation of the BusExceptionIXI feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsBusExceptionIXI in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_BusExceptionIXIEnable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_BMX_BusExceptionIXIEnable(BMX_MODULE_ID index)

b) Program Flash Cache Functions

PLIB_BMX_ProgramFlashMemoryCacheDmaDisable Function

Disables the bus matrix program Flash cacheability for DMA.

File

plib_bmx.h

C

```
void PLIB_BMX_ProgramFlashMemoryCacheDmaDisable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function disables the program Flash cacheability for DMA.

Remarks

This function implements an operation of the ProgramFlashMemoryCacheDma feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsProgramFlashMemoryCacheDma in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_ProgramFlashMemoryCacheDmaDisable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_BMX_ProgramFlashMemoryCacheDmaDisable(BMX_MODULE_ID index)

PLIB_BMX_ProgramFlashMemoryCacheDmaEnable Function

Enables the bus matrix program Flash cacheability for DMA.

File

plib_bmx.h

C

```
void PLIB_BMX_ProgramFlashMemoryCacheDmaEnable(BMX_MODULE_ID index);
```

Returns

None.

Description

This function enables the program Flash cacheability for DMA.

Remarks

This function implements an operation of the ProgramFlashMemoryCacheDma feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsProgramFlashMemoryCacheDma in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_ProgramFlashMemoryCacheDmaEnable(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_BMX_ProgramFlashMemoryCacheDmaEnable(BMX_MODULE_ID index)

c) Arbiter Functions

PLIB_BMX_ArbitrationModeGet Function

Returns the bus matrix arbitration mode.

File

plib_bmx.h

C

```
PLIB_BMX_ARB_MODE PLIB_BMX_ArbitrationModeGet(BMX_MODULE_ID index);
```

Returns

PLIB_BMX_ARB_MODE enumerator value representing the arbitration mode.

Description

This function returns the bus matrix arbitration mode.

Remarks

This function implements an operation of the ArbitrationMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsArbitrationMode in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_ARB_MODE mode;
mode = PLIB_BMX_ArbitrationModeGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

PLIB_BMX_ARB_MODE PLIB_BMX_ArbitrationModeGet (BMX_MODULE_ID index)

PLIB_BMX_ArbitrationModeSet Function

Sets the bus matrix arbitration mode.

File

plib_bmx.h

C

```
void PLIB_BMX_ArbitrationModeSet(BMX_MODULE_ID index, PLIB_BMX_ARB_MODE mode);
```

Returns

None.

Description

This function sets the bus matrix arbitration mode.

Remarks

This function implements an operation of the ArbitrationMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsArbitrationMode in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_ArbitrationModeSet(BMX_ID_0, PLIB_BMX_ARB_MODE_DMA);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Identifies the desired arbitration mode

Function

void PLIB_BMX_ArbitrationModeSet (BMX_MODULE_ID index,
PLIB_BMX_ARB_MODE mode)

d) Memory Access Control Functions

PLIB_BMX_DataRAMKernelProgramOffsetGet Function

Gets the offset (from start of RAM) of the kernel program partition.

File

plib_bmx.h

C

```
size_t PLIB_BMX_DataRAMKernelProgramOffsetGet(BMX_MODULE_ID index);
```

Returns

Offset of kernel program partition from base of RAM.

Description

This function returns the offset of start address of the kernel program RAM partition. This represents the size of the kernel data RAM partition.

Remarks

This function implements an operation of the DataRAMPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRAMPartition in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t kernProgOffset;
kernProgOffset = PLIB_BMX_DataRAMKernelProgramOffsetGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

size_t PLIB_BMX_DataRAMKernelProgramOffsetGet(BMX_MODULE_ID index)

PLIB_BMX_DataRAMPartitionSet Function

Sets the size of the kernel and user partitions in data RAM memory.

File

plib_bmx.h

C

```
void PLIB_BMX_DataRAMPartitionSet(BMX_MODULE_ID index, size_t kernProgOffset, size_t userDataOffset, size_t userProgOffset);
```

Returns

None.

Description

This function sets the size of the kernel and user partitions in the data RAM memory (DRM). By default, the entire data RAM is mapped to Kernel mode and all of the offsets are zero. To partition the data RAM, all of the offsets must be set to a minimum of one DRM block size. If any of the offsets are set to zero, the entire data RAM is allocated to kernel data. The partitions must be a multiple of PLIB_BMX_DRM_BLOCK_SIZE. If programmed with a value that is not a multiple of PLIB_BMX_DRM_BLOCK_SIZE, the value will be automatically truncated to PLIB_BMX_DRM_BLOCK_SIZE.

Remarks

This function implements an operation of the DataRAMPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRAMPartition in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
//Total Data RAM = 32 KB.
size_t totalRamSize;
size_t kernDataRamSize = (3 * PLIB_BMX_DRM_BLOCK_SIZE);
size_t kernProgRamSize = (2 * PLIB_BMX_DRM_BLOCK_SIZE);
size_t userDataRamSize = (6 * PLIB_BMX_DRM_BLOCK_SIZE);
size_t userProgRamSize = (5 * PLIB_BMX_DRM_BLOCK_SIZE);

//Get Size of Data RAM
totalRamSize = PLIB_BMX_DataRAMSizeGet(BMX_ID_0);

//Verify our partition sizes fit our RAM
```

```
if ((kernDataRamSize + kernProgRamSize + userDataRamSize +
     userProgRamSize) != totalRamSize)
{
    printf("RAM Partitioning Errorn");
}

size_t kernProgOffset;
size_t userDataOffset;
size_t userProgOffset;

kernProgOffset = kernDataRamSize;
userDataOffset = kernDataRamSize + kernProgRamSize;
userProgOffset = userDataOffset + userDataRamSize;

PLIB_BMX_DataRAMPartitionSet(BMX_ID_0, kernProgOffset,
                             userDataOffset, userProgOffset);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
kernProgOffset	Size of the offset of the Kernel Program partition
userDataOffset	Size of the offset of the User Data partition
userProgOffset	Size of the offset of the User Program partition

Function

void PLIB_BMX_DataRAMPartitionSet(BMX_MODULE_ID index,

size_t kernProgOffset,

size_t userDataOffset,

size_t userProgOffset)

PLIB_BMX_DataRAMSizeGet Function

Gets the size of data RAM memory.

File

plib_bmx.h

C

```
size_t PLIB_BMX_DataRAMSizeGet(BMX_MODULE_ID index);
```

Returns

Size of data RAM memory.

Description

This function returns the size of the data RAM memory.

Remarks

This function implements an operation of the DataRAMSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRAMSize in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t drmSize;
drmSize = PLIB_BMX_DataRAMSizeGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

size_t PLIB_BMX_DataRAMSizeGet (BMX_MODULE_ID index)

PLIB_BMX_DataRAMUserDataOffsetGet Function

Gets the offset (from start of RAM) of the user data partition.

File

plib_bmx.h

C

```
size_t PLIB_BMX_DataRAMUserDataOffsetGet(BMX_MODULE_ID index);
```

Returns

Offset of user data partition from base of RAM.

Description

This function returns the offset of start address of the user data RAM partition. Subtracting the kernel program offset from the user data offset gives the size of the kernel program RAM partition.

Remarks

This function implements an operation of the DataRAMPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRAMPartition in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t userDataOffset;
userDataOffset = PLIB_BMX_DataRAMUserDataOffsetGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

size_t PLIB_BMX_DataRAMUserDataOffsetGet(BMX_MODULE_ID index)

PLIB_BMX_DataRAMUserProgramOffsetGet Function

Gets the offset (from start of RAM) of the user program partition.

File

plib_bmx.h

C

```
size_t PLIB_BMX_DataRAMUserProgramOffsetGet(BMX_MODULE_ID index);
```

Returns

Offset of user data partition from base of RAM.

Description

This function returns the offset of start address of the user program RAM partition. Subtracting the user data offset from the user program offset gives the size of the user data RAM partition.

Remarks

This function implements an operation of the DataRAMPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRAMPartition in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t userProgOffset;
userProgOffset = PLIB_BMX_DataRAMUserProgramOffsetGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

size_t PLIB_BMX_DataRAMUserProgramOffsetGet(BMX_MODULE_ID index)

PLIB_BMX_DataRamWaitStateGet Function

Returns the number of data RAM Wait states.

File

plib_bmx.h

C

```
PLIB_BMX_DATA_RAM_WAIT_STATES PLIB_BMX_DataRamWaitStateGet(BMX_MODULE_ID index);
```

Returns

PLIB_BMX_DATA_RAM_WAIT_STATES enumeration representing the number of wait states.

Description

This function returns the number of data RAM Wait states.

Remarks

This function implements an operation of the DataRamWaitState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRamWaitState in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_DATA_RAM_WAIT_STATES wait;
wait = PLIB_BMX_DataRamWaitStateGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

PLIB_BMX_DATA_RAM_WAIT_STATES PLIB_BMX_DataRamWaitStateGet(BMX_MODULE_ID index)

PLIB_BMX_DataRamWaitStateSet Function

Sets the number of data RAM Wait states.

File

plib_bmx.h

C

```
void PLIB_BMX_DataRamWaitStateSet(BMX_MODULE_ID index, PLIB_BMX_DATA_RAM_WAIT_STATES wait);
```

Returns

None.

Description

This function sets the number of data RAM Wait states.

Remarks

This function implements an operation of the DataRamWaitState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsDataRamWaitState in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_BMX_DataRamWaitStateSet(BMX_ID_0, PLIB_BMX_DATA_RAM_WAIT_ONE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
PLIB_BMX_DATA_RAM_WAIT_STATES	Enumeration representing the number of Wait states

Function

void PLIB_BMX_DataRamWaitStateSet(BMX_MODULE_ID index,
PLIB_BMX_DATA_RAM_WAIT_STATES wait)

PLIB_BMX_ProgramFlashBootSizeGet Function

Gets the size of boot program Flash memory.

File

plib_bmx.h

C

```
size_t PLIB_BMX_ProgramFlashBootSizeGet(BMX_MODULE_ID index);
```

Returns

Size of boot program Flash memory.

Description

This function returns the size of the boot program Flash memory.

Remarks

This function implements an operation of the ProgramFlashBootSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsProgramFlashBootSize in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t bootSize;
bootSize = PLIB_BMX_ProgramFlashBootSizeGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

size_t PLIB_BMX_ProgramFlashBootSizeGet (BMX_MODULE_ID index)

PLIB_BMX_ProgramFlashMemorySizeGet Function

Gets the size of program Flash memory.

File

plib_bmx.h

C

```
size_t PLIB_BMX_ProgramFlashMemorySizeGet(BMX_MODULE_ID index);
```

Returns

Size of program Flash memory.

Description

This function returns the size of the program Flash memory.

Remarks

This function implements an operation of the ProgramFlashMemorySize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsProgramFlashMemorySize in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t pfmSize;
pfmSize = PLIB_BMX_ProgramFlashMemorySizeGet(BMX_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

size_t PLIB_BMX_ProgramFlashMemorySizeGet (BMX_MODULE_ID index)

PLIB_BMX_ProgramFlashPartitionGet Function

Gets the size of the kernel partition in program Flash memory.

File

plib_bmx.h

C

```
size_t PLIB_BMX_ProgramFlashPartitionGet(BMX_MODULE_ID index);
```

Returns

Size of the kernel partition in program Flash memory.

Description

This function gets the size of the kernel partition in the program Flash memory. The remaining Flash is set to user mode, and may be accessed by user programs. On reset, the entire Flash is mapped to Kernel mode, and this function will return zero.

Remarks

This function implements an operation of the ProgramFlashPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsProgramFlashPartition in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t pfmSize;
size_t userPfmSize;
size_t kernPfmSize;

// Get size of PFM
```

```
pfmSize = PLIB_BMX_ProgramFlashMemorySizeGet(BMX_ID_0);
kernPfmSize = PLIB_BMX_ProgramFlashPartitionGet(BMX_ID_0);
userPfmSize = pfmSize - kernPfmSize;
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

size_t PLIB_BMX_ProgramFlashPartitionGet(BMX_MODULE_ID index)

PLIB_BMX_ProgramFlashPartitionSet Function

Sets the size of the kernel and user partitions in program Flash memory.

File

plib_bmx.h

C

```
void PLIB_BMX_ProgramFlashPartitionSet(BMX_MODULE_ID index, size_t user_size);
```

Returns

None.

Description

This function sets the size of the kernel and user partitions in program Flash memory (PFM). Kernel programs may access both partitions, but user programs may not access the kernel partition (including peripheral registers). By default, the entire Flash is mapped to Kernel mode. If called with a non-zero value, a user partition of size user_size is created, and the remaining PFM remains in Kernel mode. The user partition must be a multiple of PLIB_BMX_PFM_BLOCK_SIZE. If programmed with a value that is not a multiple of PLIB_BMX_PFM_BLOCK_SIZE, the value will be truncated to a PLIB_BMX_PFM_BLOCK_SIZE boundary.

Remarks

This function implements an operation of the ProgramFlashPartition feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_BMX_ExistsProgramFlashPartition in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
size_t pfm_size;
size_t userSize = (6 * PLIB_BMX_PFM_BLOCK_SIZE);
size_t userOffset;

// Get size of PFM
size_t pfmSize;
pfmSize = PLIB_BMX_ProgramFlashMemorySizeGet(BMX_ID_0);
userOffset = pfmSize - userSize;
if (userOffset > 0)
{
    PLIB_BMX_ProgramFlashPartitionSet(BMX_ID_0, userOffset);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
userSize	Size of the user partition in PFM

Function

void PLIB_BMX_ProgramFlashPartitionSet(BMX_MODULE_ID index,

size_t userSize)

e) Feature Existence Functions

PLIB_BMX_ExistsArbitrationMode Function

Identifies that the ArbitrationMode feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsArbitrationMode(BMX_MODULE_ID index);
```

Returns

- true = The ArbitrationMode feature is supported on the device
- false = The ArbitrationMode feature is not supported on the device

Description

This interface identifies that the ArbitrationMode feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_ArbitrationModeSet
- PLIB_BMX_ArbitrationModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsArbitrationMode(BMX_MODULE_ID index)

PLIB_BMX_ExistsBusExceptionData Function

Identifies that the BusExceptionData feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsBusExceptionData(BMX_MODULE_ID index);
```

Returns

- true = The BusExceptionData feature is supported on the device
- false = The BusExceptionData feature is not supported on the device

Description

This interface identifies that the BusExceptionData feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_BusExceptionDataEnable
- PLIB_BMX_BusExceptionDataDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsBusExceptionData(BMX_MODULE_ID index)

PLIB_BMX_ExistsBusExceptionDMA Function

Identifies that the BusExceptionDMA feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsBusExceptionDMA(BMX_MODULE_ID index);
```

Returns

- true = The BusExceptionDMA feature is supported on the device
- false = The BusExceptionDMA feature is not supported on the device

Description

This interface identifies that the BusExceptionDMA feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_BusExceptionDMAEnable
- PLIB_BMX_BusExceptionDMADisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsBusExceptionDMA(BMX_MODULE_ID index)

PLIB_BMX_ExistsBusExceptionICD Function

Identifies that the BusExceptionICD feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsBusExceptionICD(BMX_MODULE_ID index);
```

Returns

- true = The BusExceptionICD feature is supported on the device
- false = The BusExceptionICD feature is not supported on the device

Description

This interface identifies that the BusExceptionICD feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_BusExceptionICDEnable
- PLIB_BMX_BusExceptionICDDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsBusExceptionICD(BMX_MODULE_ID index)

PLIB_BMX_ExistsBusExceptionInstruction Function

Identifies that the BusExceptionInstruction feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsBusExceptionInstruction(BMX_MODULE_ID index);
```

Returns

- true = The BusExceptionInstruction feature is supported on the device
- false = The BusExceptionInstruction feature is not supported on the device

Description

This interface identifies that the BusExceptionInstruction feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_BusExceptionInstructionEnable
- PLIB_BMX_BusExceptionInstructionDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsBusExceptionInstruction(BMX_MODULE_ID index)

PLIB_BMX_ExistsBusExceptionIXI Function

Identifies that the BusExceptionIXI feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsBusExceptionIXI(BMX_MODULE_ID index);
```

Returns

true = The BusExceptionIXI feature is supported on the device

- false = The BusExceptionIXI feature is not supported on the device

Description

This interface identifies that the BusExceptionIXI feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_BusExceptionIXIEnable
- PLIB_BMX_BusExceptionIXIDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsBusExceptionIXI(BMX_MODULE_ID index)

PLIB_BMX_ExistsDataRAMPartition Function

Identifies that the DataRAMPartition feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsDataRAMPartition(BMX_MODULE_ID index);
```

Returns

- true = The DataRAMPartition feature is supported on the device
- false = The DataRAMPartition feature is not supported on the device

Description

This interface identifies that the DataRAMPartition feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_DataRAMPartitionSet
- PLIB_BMX_DataRAMKernelProgramOffsetGet
- PLIB_BMX_DataRAMUserDataOffsetGet
- PLIB_BMX_DataRAMUserProgramOffsetGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsDataRAMPartition(BMX_MODULE_ID index)

PLIB_BMX_ExistsDataRAMSize Function

Identifies that the DataRAMSize feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsDataRAMSize(BMX_MODULE_ID index);
```

Returns

- true = The DataRAMSize feature is supported on the device
- false = The DataRAMSize feature is not supported on the device

Description

This interface identifies that the DataRAMSize feature is available on the BMX module. When this interface returns true, this function is supported on the device:

- PLIB_BMX_DataRAMSizeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsDataRAMSize(BMX_MODULE_ID index)

PLIB_BMX_ExistsDataRamWaitState Function

Identifies that the DataRamWaitState feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsDataRamWaitState(BMX_MODULE_ID index);
```

Returns

- true = The DataRamWaitState feature is supported on the device
- false = The DataRamWaitState feature is not supported on the device

Description

This interface identifies that the DataRamWaitState feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_DataRamWaitStateSet
- PLIB_BMX_DataRamWaitStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsDataRamWaitState(BMX_MODULE_ID index)

PLIB_BMX_ExistsProgramFlashBootSize Function

Identifies that the ProgramFlashBootSize feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsProgramFlashBootSize(BMX_MODULE_ID index);
```

Returns

- true = The ProgramFlashBootSize feature is supported on the device
- false = The ProgramFlashBootSize feature is not supported on the device

Description

This interface identifies that the ProgramFlashBootSize feature is available on the BMX module. When this interface returns true, this function is supported on the device:

- PLIB_BMX_ProgramFlashBootSizeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsProgramFlashBootSize(BMX_MODULE_ID index)

PLIB_BMX_ExistsProgramFlashMemoryCacheDma Function

Identifies that the ProgramFlashMemoryCacheDma feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsProgramFlashMemoryCacheDma(BMX_MODULE_ID index);
```

Returns

- true = The ProgramFlashMemoryCacheDma feature is supported on the device
- false = The ProgramFlashMemoryCacheDma feature is not supported on the device

Description

This interface identifies that the ProgramFlashMemoryCacheDma feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_ProgramFlashMemoryCacheDmaEnable
- PLIB_BMX_ProgramFlashMemoryCacheDmaDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsProgramFlashMemoryCacheDma(BMX_MODULE_ID index)

PLIB_BMX_ExistsProgramFlashMemorySize Function

Identifies that the ProgramFlashMemorySize feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsProgramFlashMemorySize(BMX_MODULE_ID index);
```

Returns

- true = The ProgramFlashMemorySize feature is supported on the device
- false = The ProgramFlashMemorySize feature is not supported on the device

Description

This interface identifies that the ProgramFlashMemorySize feature is available on the BMX module. When this interface returns true, this function is supported on the device:

- PLIB_BMX_ProgramFlashMemorySizeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsProgramFlashMemorySize(BMX_MODULE_ID index)

PLIB_BMX_ExistsProgramFlashPartition Function

Identifies that the ProgramFlashPartition feature exists on the BMX module.

File

plib_bmx.h

C

```
bool PLIB_BMX_ExistsProgramFlashPartition(BMX_MODULE_ID index);
```

Returns

- true = The ProgramFlashPartition feature is supported on the device
- false = The ProgramFlashPartition feature is not supported on the device

Description

This interface identifies that the ProgramFlashPartition feature is available on the BMX module. When this interface returns true, these functions are supported on the device:

- PLIB_BMX_ProgramFlashPartitionGet
- PLIB_BMX_ProgramFlashPartitionSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_BMX_ExistsProgramFlashPartition(BMX_MODULE_ID index)

f) Data Types and Constants

BMX_MODULE_ID Enumeration

Lists the possible Module IDs for the bus matrix.

File

help_plib_bmx.h

C

```
typedef enum {
  BMX_ID_0,
  BMX_NUMBER_OF_MODULES
} BMX_MODULE_ID;
```

Description

Module ID

This enumeration lists the possible Module IDs for the bus matrix.

Remarks

Refer to the specific device data sheet to obtain the correct number of modules defined for the desired device.

PLIB_BMX_ARB_MODE Enumeration

Lists the possible arbitration modes for the bus matrix.

File

plib_bmx.h

C

```
typedef enum {
  PLIB_BMX_ARB_MODE_INST,
  PLIB_BMX_ARB_MODE_DMA,
  PLIB_BMX_ARB_MODE_ROT
} PLIB_BMX_ARB_MODE;
```

Members

Members	Description
PLIB_BMX_ARB_MODE_INST	Arbitration Mode 0
PLIB_BMX_ARB_MODE_DMA	Arbitration Mode 1
PLIB_BMX_ARB_MODE_ROT	Arbitration Mode 2

Description

Arbitration Mode

This enumeration lists the possible arbitration modes for the bus matrix.

PLIB_BMX_DATA_RAM_WAIT_STATES Enumeration

Defines the number of data RAM wait states for address setup.

File

plib_bmx.h

C

```
typedef enum {
  PLIB_BMX_DATA_RAM_WAIT_ZERO,
  PLIB_BMX_DATA_RAM_WAIT_ONE
} PLIB_BMX_DATA_RAM_WAIT_STATES;
```

Members

Members	Description
PLIB_BMX_DATA_RAM_WAIT_ZERO	Zero wait states for address setup
PLIB_BMX_DATA_RAM_WAIT_ONE	One wait state for address setup

Description

Wait States

This definition specifies the number of data RAM wait states for address setup.

PLIB_BMX_EXCEPTION_SRC Enumeration

Defines which events trigger a bus exception.

File

plib_bmx.h

C

```
typedef enum {
  PLIB_BMX_ERR_IXI,
  PLIB_BMX_ERR_ICD,
  PLIB_BMX_ERR_DMA,
  PLIB_BMX_ERR_DATA,
  PLIB_BMX_ERR_INST
} PLIB_BMX_EXCEPTION_SRC;
```

Members

Members	Description
PLIB_BMX_ERR_IXI	IXI Shared Bus
PLIB_BMX_ERR_ICD	In-Circuit Debugger
PLIB_BMX_ERR_DMA	DMA Controller
PLIB_BMX_ERR_DATA	CPU Data Bus
PLIB_BMX_ERR_INST	CPU Instruction Bus

Description

Exception Bits

This definition specifies which events trigger a bus exception.

PLIB_BMX_DRM_BLOCK_SIZE Macro

Defines the minimum partition block size in data RAM memory.

File

plib_bmx.h

C

```
#define PLIB_BMX_DRM_BLOCK_SIZE 2048
```

Description

Program Flash Partition Block Size

This definition specifies the minimum partition block size in data RAM memory.

PLIB_BMX_PFM_BLOCK_SIZE Macro

Defines the minimum partition block size in program Flash memory.

File

plib_bmx.h

C

```
#define PLIB_BMX_PFM_BLOCK_SIZE 2048
```

Description

Program Flash Partition Block Size

This definition specifies the minimum partition block size in program Flash memory.

Files

Files

Name	Description
plib_bmx.h	Defines the Bus Matrix (BMX) Peripheral Library interface
help_plib_bmx.h	This file is used for documentation purposes

Description

This section lists the source and header files used by the library.

plib_bmx.h

Defines the Bus Matrix (BMX) Peripheral Library interface

Enumerations

	Name	Description
	PLIB_BMX_ARB_MODE	Lists the possible arbitration modes for the bus matrix.
	PLIB_BMX_DATA_RAM_WAIT_STATES	Defines the number of data RAM wait states for address setup.
	PLIB_BMX_EXCEPTION_SRC	Defines which events trigger a bus exception.

Functions

	Name	Description
	PLIB_BMX_ArbitrationModeGet	Returns the bus matrix arbitration mode.
	PLIB_BMX_ArbitrationModeSet	Sets the bus matrix arbitration mode.
	PLIB_BMX_BusExceptionDataDisable	Disables the data bus exception.
	PLIB_BMX_BusExceptionDataEnable	Enables the Data bus exception.
	PLIB_BMX_BusExceptionDMADisable	Disables the DMA bus exception.
	PLIB_BMX_BusExceptionDMAEnable	Enables the DMA bus exception.
	PLIB_BMX_BusExceptionICDDisable	Disables the ICD bus exception.
	PLIB_BMX_BusExceptionICDEnable	Enables the ICD bus exception.
	PLIB_BMX_BusExceptionInstructionDisable	Disables the instruction bus exception.
	PLIB_BMX_BusExceptionInstructionEnable	Enables the instruction bus exception.
	PLIB_BMX_BusExceptionIXIDisable	Disables the IXI bus exception.
	PLIB_BMX_BusExceptionIXIEnable	Enables the IXI bus exception.
	PLIB_BMX_DataRAMKernelProgramOffsetGet	Gets the offset (from start of RAM) of the kernel program partition.
	PLIB_BMX_DataRAMPartitionSet	Sets the size of the kernel and user partitions in data RAM memory.
	PLIB_BMX_DataRAMSizeGet	Gets the size of data RAM memory.
	PLIB_BMX_DataRAMUserDataOffsetGet	Gets the offset (from start of RAM) of the user data partition.

	PLIB_BMX_DataRAMUserProgramOffsetGet	Gets the offset (from start of RAM) of the user program partition.
	PLIB_BMX_DataRamWaitStateGet	Returns the number of data RAM Wait states.
	PLIB_BMX_DataRamWaitStateSet	Sets the number of data RAM Wait states.
	PLIB_BMX_ExistsArbitrationMode	Identifies that the ArbitrationMode feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionData	Identifies that the BusExceptionData feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionDMA	Identifies that the BusExceptionDMA feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionICD	Identifies that the BusExceptionICD feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionInstruction	Identifies that the BusExceptionInstruction feature exists on the BMX module.
	PLIB_BMX_ExistsBusExceptionIXI	Identifies that the BusExceptionIXI feature exists on the BMX module.
	PLIB_BMX_ExistsDataRAMPartition	Identifies that the DataRAMPartition feature exists on the BMX module.
	PLIB_BMX_ExistsDataRAMSize	Identifies that the DataRAMSize feature exists on the BMX module.
	PLIB_BMX_ExistsDataRamWaitState	Identifies that the DataRamWaitState feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashBootSize	Identifies that the ProgramFlashBootSize feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashMemoryCacheDma	Identifies that the ProgramFlashMemoryCacheDma feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashMemorySize	Identifies that the ProgramFlashMemorySize feature exists on the BMX module.
	PLIB_BMX_ExistsProgramFlashPartition	Identifies that the ProgramFlashPartition feature exists on the BMX module.
	PLIB_BMX_ProgramFlashBootSizeGet	Gets the size of boot program Flash memory.
	PLIB_BMX_ProgramFlashMemoryCacheDmaDisable	Disables the bus matrix program Flash cacheability for DMA.
	PLIB_BMX_ProgramFlashMemoryCacheDmaEnable	Enables the bus matrix program Flash cacheability for DMA.
	PLIB_BMX_ProgramFlashMemorySizeGet	Gets the size of program Flash memory.
	PLIB_BMX_ProgramFlashPartitionGet	Gets the size of the kernel partition in program Flash memory.
	PLIB_BMX_ProgramFlashPartitionSet	Sets the size of the kernel and user partitions in program Flash memory.

Macros

	Name	Description
	PLIB_BMX_DRM_BLOCK_SIZE	Defines the minimum partition block size in data RAM memory.
	PLIB_BMX_PFM_BLOCK_SIZE	Defines the minimum partition block size in program Flash memory.

Description

Bus Matrix Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Bus Matrix (BMX) Peripheral Library (PLIB) for Microchip microcontrollers. The definitions in this file are for the bus matrix controller module.

File Name

plib_bmx.h

Company

Microchip Technology Inc.

help_plib_bmx.h

Enumerations

	Name	Description
	BMX_MODULE_ID	Lists the possible Module IDs for the bus matrix.

Description

This file is used for documentation purposes

CAN Peripheral Library

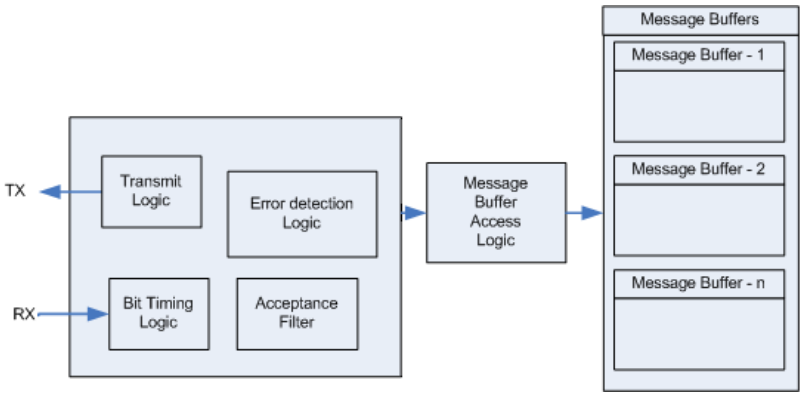
This section describes the CAN Peripheral Library.

Introduction

This library provides a low-level abstraction of the CAN module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The Controller Area Network (CAN) is used primarily in industrial and automotive applications. It is an asynchronous serial data communication protocol that provides reliable communication in an electrically noisy environment.



Note: Trademarks and Intellectual Property are property of their respective owners. Customers are responsible for obtaining appropriate licensing or rights before using this software. Refer to the MPLAB Harmony *Software License Agreement* for complete licensing information. A copy of this agreement is available in the `<install-dir>/doc` folder of your MPLAB Harmony installation.

Using the Library

This topic describes the basic architecture of the CAN Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_can.h`

The interface to the CAN Peripheral library is defined in the `plib_can.h` header file, which is included in the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the CAN Peripheral library must include `peripheral.h`.

Library File:

The CAN Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

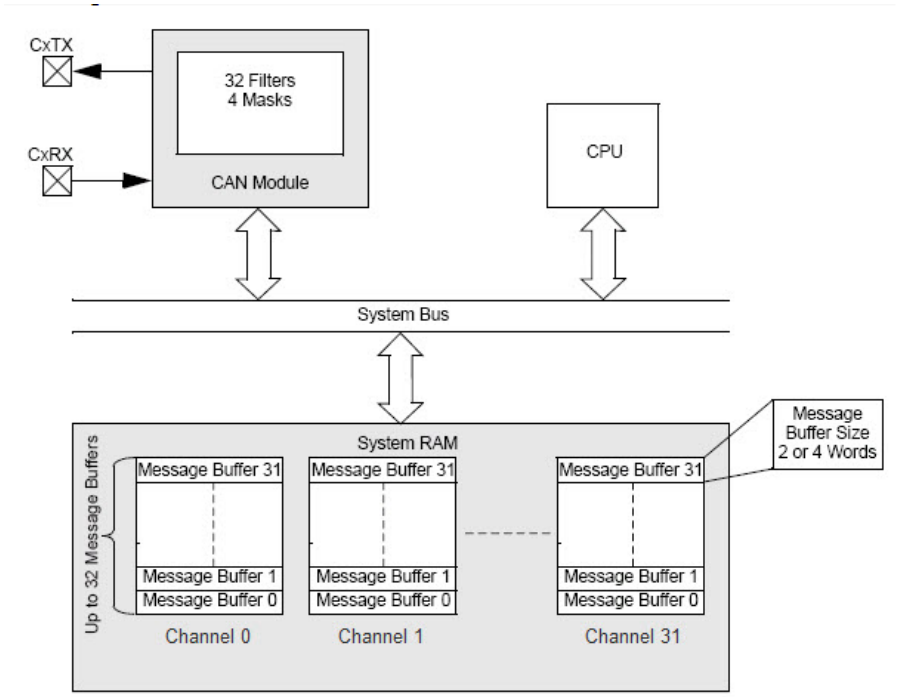
Please refer to the What is MPLAB Harmony? section for how the Peripheral interacts with the framework.

Hardware Abstraction Model

Note: The interface provided is a super set of all of the functionality of the available CAN on the device. Refer to the specific device data sheet or the related family reference manual to determine the set of functions that are supported for each CAN module on the device.

This library provides the low-level abstraction of the CAN module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

CAN Software Abstraction Block Diagram



Software Model:

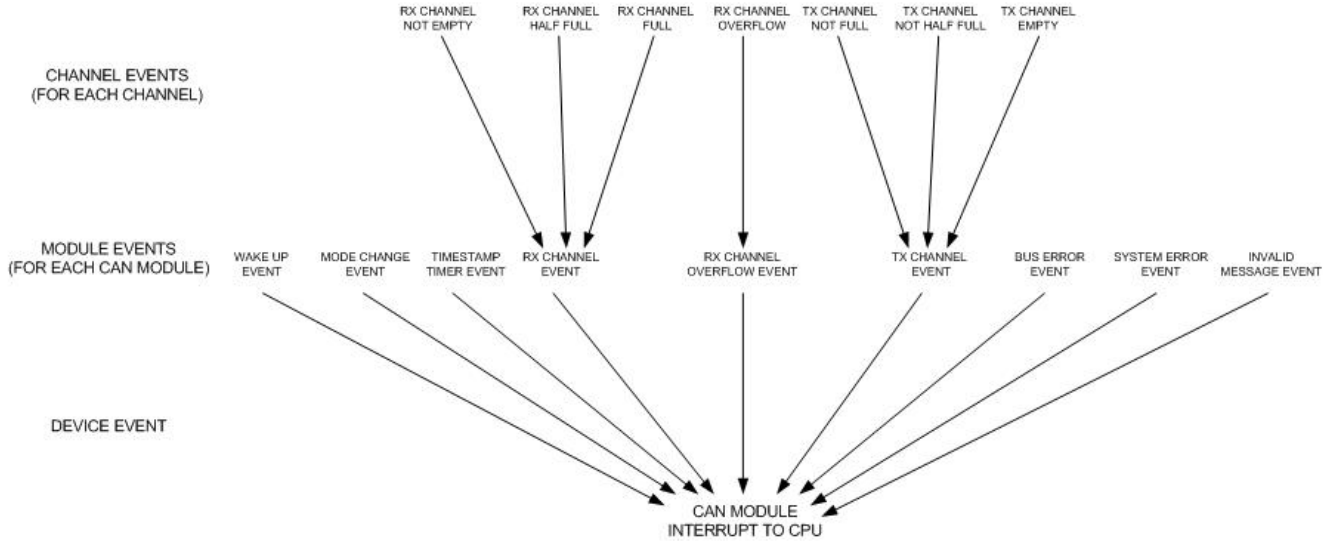
The CAN Peripheral Library provides interface routines to interact with the blocks shown in the previous diagram. A channel can be either a transmit channel or a receive channel. The size of a CAN message is typically 16 bytes (8 bytes for data-only receive messages). The size of a channel (the number of messages it can buffer) is configurable. Each channel has a minimum size of one message buffer. The message buffer area for all channels is assigned at configuration time. The CAN Peripheral Library then provides access to these channels and buffers via the library interface.

The user code must enable and configure message acceptance filters to receive messages. These filters compare the ID field of the incoming message with configured value and accepts the messages if IDs match. The message is then stored in a selectable receive channel. At least one message acceptance filter and one filter mask should be enabled for the CAN module to receive messages. A filter mask allows specified filter bits to be ignored during the comparison process. This allows the filter to accept a message with an ID range.

The CAN Peripheral Library allows the CAN module to generate events. Events can be generated at the channel level and at the module level.

Events:

The CAN module is capable of notifying the user application when specific events occur in the module. The following diagram shows these events and how they are organized.



There are two types of events in the CAN module: Channel events and Module events. Channel events are generated by transmit and receive channels. Module events are generated by various sources (including channels) within the CAN module. Each event can be enabled or disabled. Enabling a channel event will cause the CAN module to generate a module event. An enabled module level event will cause the CAN module to generate an interrupt to the CPU.

The application can also poll channel events to check if these are active. This may be done in cases where a polling scheme (rather than an interrupt scheme) needs to be implemented. In this case, the CAN module provides events status on a channel basis. If a channel event is found active, the application can then use library API calls to inspect which channel condition caused the channel event.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the CAN module.

Library Interface Section	Description
Basic Configuration Functions	APIs that are used for basic control of the module, such as enabling and disabling the module.
Advanced Configuration Functions	These functions enable advanced configuration, such as selection of the operation mode.
Bus Speed Configuration Functions	These functions enable bus speed configuration.
Event Management Functions	These functions can be used to manage the events.
Message Transmit Functions	These functions can be used for message transmission.
Message Receive Functions	These functions are related to data reception.
Error State Tracking Functions	These functions are used for error tracking.
Information Functions	These functions provide information about the module.
Feature Existence Functions	These functions determine whether or not a particular feature is supported by the device.

How the Library Works

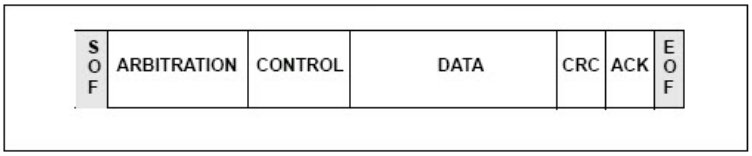
Note: Not all modes are available on all the devices. Please refer to the specific device data sheet to determine the set of modes that are supported for your device.

About CAN Protocol

This section briefly discusses the CAN Protocol briefly. For complete details on the CAN protocol, refer to the CAN 2.0 specification available from Bosch.

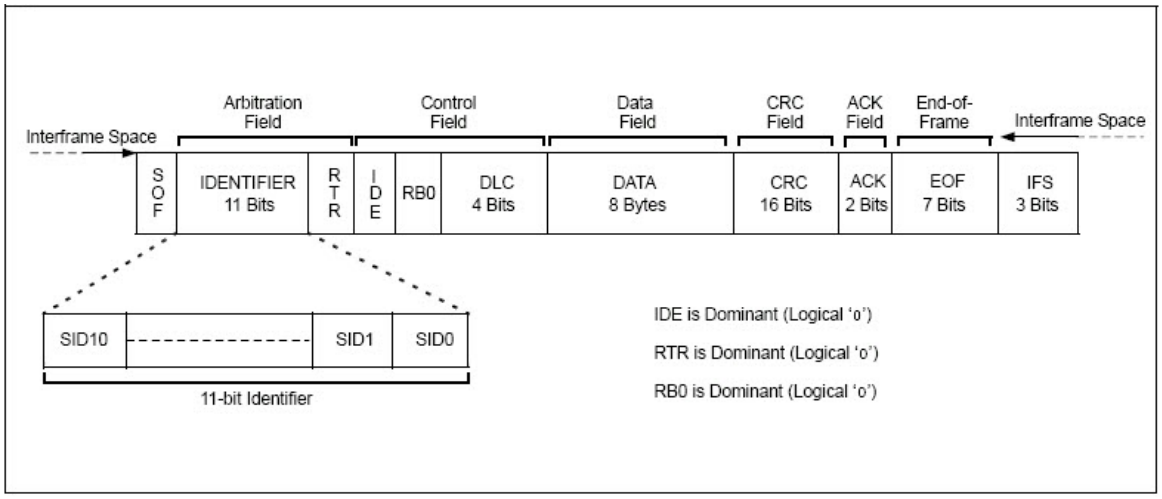
Description

The CAN bus protocol uses asynchronous communication. Information is passed from transmitters to receivers in data frames, which are composed of byte fields that define the contents of the data frame. This is shown in the following figure.



Each frame begins with a Start-of-Frame (SOF) bit and terminates with an End-of-Frame (EOF) bit field. The SOF is followed by arbitration and control fields, which identify the message type, format, length and priority. This information allows each node on the CAN bus to respond appropriately to the message. The data field conveys the message content and is of variable length, ranging from 0 to 8 bytes. Error protection is provided by the Cyclic Redundancy Check (CRC) and acknowledgement (ACK) fields.

Standard ID Message Format



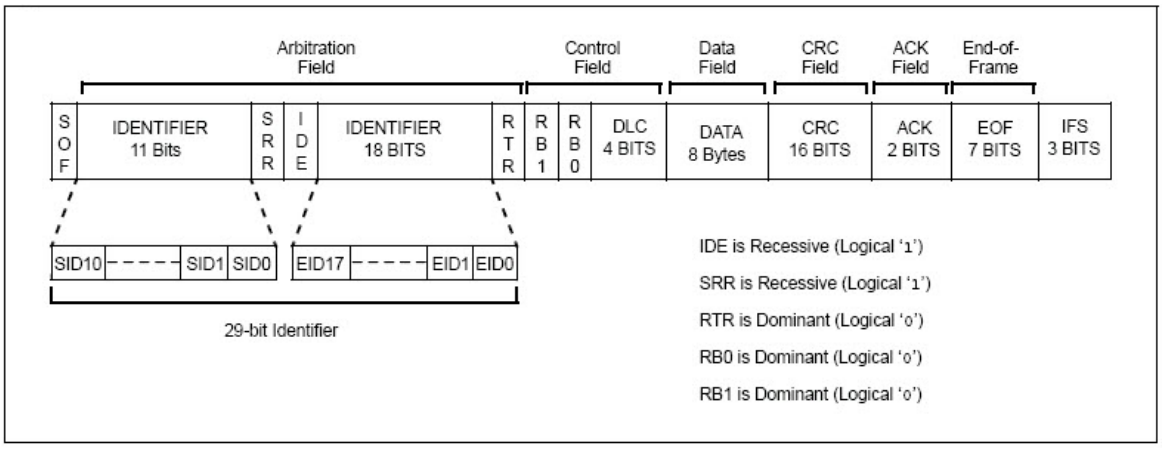
The Standard ID message begins with a Start-of-Frame bit followed by a 12-bit arbitration field. The arbitration field contains an 11-bit identifier and the Remote Transmit Request (RTR) bit. The identifier defines the type of information contained in the message and is used by each receiving node to determine if the message is of interest. The RTR bit distinguishes a data frame from a remote frame. For a standard data frame, the RTR bit is clear.

Following the arbitration field is a 6-bit control field, which provides more information about the contents of the message. The first bit in the control field is an Identifier Extension (IDE) bit, which distinguishes the message as either a standard or extended data frame. A standard data frame is indicated by a dominant state (logic level '0') during transmission of the IDE bit. The second bit in the control field is a reserved (RB0) bit, which is in the dominant state (logic level '0'). The last 4 bits in the control field represent the Data Length Code (DLC), which specifies the number of data bytes present in the message.

The data field follows the control field. This field carries the message data – the actual payload of the data frame. This field is of variable length, ranging from 0 to 8 bytes. The number of bytes is user-selectable.

The data field is followed by the Cyclic Redundancy Check (CRC) field, which is a 15-bit CRC sequence with one delimiter bit. The acknowledgement (ACK) field is sent as a recessive bit (logic level '1') and is overwritten as a dominant bit by any receiver that has received the data correctly. The message is acknowledged by the receiver regardless of the result of the acceptance filter comparison. The last field is the End-of-Frame field, which consists of 7 recessive bits that indicate the end of message.

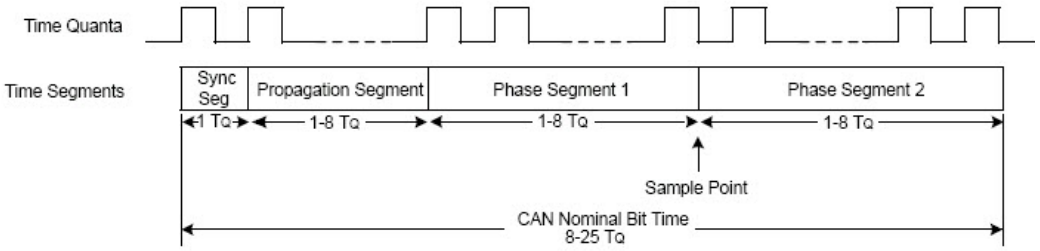
Extended ID Message Format



The Extended ID Message frame begins with an SOF bit followed by a 31-bit arbitration field. The arbitration field for the extended data frame contains 29 identifier bits in two fields separated by a Substitute Remote Request (SRR) bit and an IDE bit. The SRR bit determines if the message is a remote frame. SRR = 1 for extended data frames. The IDE bit indicates the data frame type. For the extended data frame, IDE = 1.

The extended data frame Control field consists of 7 bits. The first bit is the RTR. For the extended data frame, RTR = 0. The next two bits, RB1 and RB0, are reserved bits that are in the dominant state (logic level '0'). The last 4 bits in the Control field are the Data Length Code, which specifies the number of data bytes present in the message. The remaining fields in an extended data frame are identical to a standard data frame.

CAN Bit Time Quanta



The nominal bit rate is the number of bits per second transmitted on the CAN bus.

Nominal bit time = 1 ÷ Nominal Bit Rate.

There are four time segments in a bit time to compensate for any phase shifts due to oscillator drifts or propagation delays. These time segments do not overlap each other and are represented in terms of Time Quantum (TQ). One TQ is a fixed unit of time derived from the oscillator clock. The total number of time quanta in a nominal bit time must be programmed between 8 and 25 TQ.

Each bit transmission time consists of four time segments:

Synchronization Segment – This time segment synchronizes the different nodes connected on the CAN bus. A bit edge is expected to be within this segment. Based on CAN protocol, the Synchronization Segment is assumed to be one Time Quantum.

Propagation Segment – This time segment compensates for any time delay that may occur due to the bus line or due to the various transceivers connected on that bus.

Phase Segment 1 – This time segment compensates for errors that may occur due to phase shift in the edges. The time segment may be lengthened during resynchronization to compensate for the phase shift.

Phase Segment 2 – This time segment compensates for errors that may occur due to phase shift in the edges. The time segment may be shortened during resynchronization to compensate for the phase shift. The Phase Segment 2 time can be configured to be either programmable or specified by the Phase Segment 1 time.

Two types of synchronization are used – Hard Synchronization and Resynchronization. A Hard Synchronization occurs once at the start of a frame. Resynchronization occurs inside a frame.

Hard synchronization takes place on the recessive-to-dominant transition of the start bit. The bit time is restarted from that edge.

Resynchronization takes place when a bit edge does not occur within the Synchronization Segment in a message. One of the Phase Segments is shortened or lengthened by an amount that depends on the phase error in the signal. The maximum amount that can be used is determined by the Synchronization Jump Width parameter.

The length of Phase Segment 1 and Phase Segment 2 can be changed depending on oscillator tolerances of the transmitting and receiving node. Resynchronization compensates for any phase shifts that may occur due to the different oscillators used by the transmitting and receiving nodes.

Bit Lengthening – If the transmitting node in CAN has a slower oscillator than the receiving node, the next falling edge, and therefore, the sample point, can be delayed by lengthening Phase Segment 1 in the bit time.

Bit Shortening – If the transmitting node in CAN has a faster oscillator than the receiving node, the next falling edge, and therefore, the sample point of the next bit, can be reduced by shortening the Phase Segment 2 in the bit time.

Synchronization Jump Width (SJW) – This determines the synchronization jump width by limiting the amount of lengthening or shortening that can be applied to the Phase Segment 1 and Phase Segment 2 time intervals. This segment should not be longer than Phase Segment 2 time. The width can be 1-4 TQ

Initialization of CAN

This section provides information on initializing CAN, including channel, filters, masks, and mode configuration, setting the bus speed, assigning buffer memory, and enabling events.

Mode Configuration

The CAN module must be enabled before configuration. Enabling the CAN module gives it control over the CAN pins of the device. The following code example shows how the PLIB_CAN_Enable function is used to enable the module.

```
/* This code expects the data direction settings to be done from other modules */
/* Switch the CAN module ON */
PLIB_CAN_Enable(CAN_ID_1);
```

The CAN module must be placed in Configuration mode before attempting to configure it. The following features are available in Configuration mode only:

- Configuring the CAN module clock and CAN bus speed
- Configuring the message acceptance filter mask
- Assigning the channel buffer memory

- Configuring channel size and Receive Channel Data-only mode.

The CAN module operating mode can be changed by using the PLIB_CAN_OperationModeSelect function. The caller should wait until the operating mode is set before proceeding. The PLIB_CAN_OperationModeGet function can be used to obtain the current operating mode of the CAN module.

```
/* Switch the CAN module to Configuration mode. Wait until the switch is complete */
PLIB_CAN_Enable(CAN_ID_1);
PLIB_CAN_OperationModeSelect(CAN_ID_1, CAN_CONFIGURATION_MODE);
while(PLIB_CAN_OperationModeGet(CAN_ID_1) != CAN_CONFIGURATION_MODE);
```

The CAN module must be placed in Normal mode to be able to send and receive messages.

The caller should wait until the Normal operating mode is set before proceeding. The PLIB_CAN_OperationModeSelect function can be used to obtain the current operating mode of the CAN module.

```
/* Switch the CAN module to Normal mode. Wait until the switch is complete */
PLIB_CAN_OperationModeSelect(CAN_ID_1, CAN_NORMAL_MODE);
while(PLIB_CAN_OperationModeGet(CAN_ID_1) != CAN_NORMAL_MODE);
```

Setting Bus Speed

The bus speed at which the CAN node operates is determined by the CAN module clock and the number of Bit Time Quanta (TQ) assigned to the CAN bit.

The following code shows an example of setting the CAN module clock and CAN bus speed. In this example, the number of TQ is 11.

```
/*The CPU core frequency is 80000000 Hz and the desired CAN bus speed is 500 kbps.
  Use ECAN Bit Rate Calculator plug-in to get parameter values for 500 kbps Baud rate*/

    #define MY_CAN_ID       CAN_ID_1
    #define PRESCALE        6
    #define SYNCJUMPWIDTH   1
    #define PROPAGATION     2
    #define SEGMENT1        4
    #define SEGMENT2        4
    #define CAN_CLOCK       80000000ul

    uint32_t baudRate;
    bool result;


    if (CAN_CONFIGURATION_MODE == PLIB_CAN_OperationModeGet(MY_CAN_ID))
    {
        result = PLIB_CAN_PrecalculatedBitRateSetup( MY_CAN_ID, PRESCALE, SYNCJUMPWIDTH,
                                            PROPAGATION, SEGMENT1, SEGMENT2 );

        if(false == result)
        {
            // Error occurred, handle accordingly
        }
        else
        {
            baudRate = PLIB_CAN_BaudRateGet(MY_CAN_ID, CAN_CLOCK);
        }

    }
```

Assigning Buffer Memory

The CAN module requires RAM memory to store received messages and queue message that need to be transmitted. The total amount of memory required depends on the size of each channel. While each channel has a minimum size of one message buffer (16 bytes), to simplify usage issues, channels should be used in a sequentially ascending order starting with channel 0.

To better understand the buffer memory requirements of the CAN module, consider the following examples:

Example 1: Two transmit channels (each with eight message buffers) and one full receive channel (with 10 message buffers):

- Two transmit channels with eight message buffers: 2 * 8 * 16 = 256 bytes
- One receive channel with 10 message buffers: 1 * 10 * 16 = 160 bytes
- Total memory required in bytes = 416 bytes

Example 2: Four transmit channels (two with eight message buffers and other two with 10 message buffers) and two full receive channels (one with 10 message buffers and one with 20 message buffers):

- Two transmit channels with eight message buffers: 2 * 8 * 16 = 256 bytes
- Two transmit channels with 10 message buffers: 2 * 10 * 16 = 320 bytes

- One receive channel with 10 message buffers: 1 * 10 * 16 = 160 bytes
- One receive channel with 20 message buffers: 1 * 20 * 16 = 320 bytes
- Total memory required in bytes = 1056 bytes

Example 3: One transmit channel (with maximum message buffers) and two receive Channels (one full receive with 10 message buffers and one data-only channel with maximum message buffers):

- One transmit channel with maximum (32) message buffers: 1 * 32 * 16 = 512 bytes
- One receive channel with 10 message buffers: 1 * 10 * 16 = 160 bytes
- One data-only receive channel with 32 message buffers: 1 * 32 * 8 = 256 bytes
- Total memory required in bytes = 928 bytes

The PLIB_CAN_MemoryBufferAssign function is used to assign a memory buffer to the CAN module, which is shown in the following example.

```
/* Assign the buffer area to the CAN module.
 * In this case assign enough memory for 2
 * channels, each with 8 message buffers.*/

uint8_t MY_CAN_MessageFifoArea[2 * 8 * 16];
PLIB_CAN_MemoryBufferAssign(CAN_ID_1, MY_CAN_MessageFifoArea);
```

Channel Configuration

To transmit and receive CAN messages, the user application must configure channels. The PLIB_CAN_ChannelForTransmitSet function is used to configure a channel for transmit operation. The PLIB_CAN_ChannelForReceiveSet function is used to configure a channel for receive operation.

Note: To receive CAN messages, at least one message acceptance filter with a receive channel must be configured.

A CAN Transmit channel can be enabled to process a Remote Transmit Request (RTR). In this case, the transmit channel will automatically transmit all queued messages when an RTR message is accepted by the CAN module.

The following code shows an example of configuring a CAN Transmit channel.

```
/* Configure CAN1 Channel 0 for Transmit operation. Allocate 8 message buffer,
 * disable the RTR feature and assign low medium priority for transmissions. */

PLIB_CAN_ChannelForTransmitSet(CAN_ID_1, CAN_CHANNEL0, 8, CAN_TX_RTR_DISABLED,
     CAN_LOW_MEDIUM_PRIORITY);
```

A CAN receive channel can be enabled to store the data payload portion of an accepted message. This mode is called the Data-only mode. The CAN module otherwise stores both the ID and data payload portion of the accepted message.

The following code shows an example of configuring a CAN Receive channel.

```
/* Configure CAN1 Channel 1 for Receive operation. Allocate 8 message buffers,
 * disable the Data-only feature (specify full receive mode). */

PLIB_CAN_ChannelForReceiveSet(CAN_ID_1, CAN_CHANNEL1, 8, CAN_RX_FULL_RECEIVE);
```

Filters and Masks Configuration

The CAN message acceptance filter allows an application to accept only selected messages from the CAN bus. The CAN module achieves this by comparing the Standard or Extended ID of a message with the configured filter ID. The message is accepted on an ID match. The accepted message is stored in the configured destination channel.

The CAN message acceptance filter mask directs the CAN module to ignore specified bits in the filter comparison. This allows the application to accept messages within an ID range.

At least one filter and one mask must be configured for the CAN module to be able to receive messages. The destination channel for a filter must be a receive channel unless the channel is a transmit channel with the RTR feature enabled.

The PLIB_CAN_FilterConfigure function is used to configure a message acceptance filter. The PLIB_CAN_FilterMaskConfigure function is used to configure a message acceptance filter mask. The filter and mask must be linked together to a channel. This is done using the PLIB_CAN_FilterToChannelLink function. The CAN filter should be enabled to allow messages to be processed. This is done using the PLIB_CAN_FilterEnable function.

To better understand how the filter and masks can be used to implement filtering, consider the following code examples:

Example 1: Configure a CAN filter to accept a Standard ID message with ID 0x201

```
/* Only one message ID is to be accepted. All bits are to be compared
 * therefore, the mask value should be 0x7FF. The message type is Standard ID.
 * Configure CAN1, Filter 0, Mask 0 and set the destination channel as
 * Channel 1. */

PLIB_CAN_FilterConfigure(CAN_ID_1, CAN_FILTER0, 0x201, CAN_SID);
PLIB_CAN_FilterMaskConfigure(CAN_ID_1, CAN_FILTER_MASK0, 0x7FF, CAN_SID,
    CAN_FILTER_MASK_IDE_TYPE);
PLIB_CAN_FilterToChannelLink(CAN_ID_1, CAN_FILTER0, CAN_FILTER_MASK0,
```

```
    CAN_CHANNEL1);
PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER0);
```

Example 2: Configure a CAN filter to accept an Extended ID message with ID 0x8004001

```
/* Only one message ID is to be accepted. All bits are to be compared
 * therefore, the mask value should be 0x8004001. The message type is Extended ID.
 * Configure CAN1, Filter 0, Mask 0 and set the destination channel as
 * Channel 1. */

PLIB_CAN_FilterConfigure(CAN_ID_1, CAN_FILTER0, 0x8004001, CAN_EID);
PLIB_CAN_FilterMaskConfigure(CAN_ID_1, CAN_FILTER_MASK0, 0x3FFFFFFF, CAN_EID,
    CAN_FILTER_MASK_IDE_TYPE);
PLIB_CAN_FilterToChannelLink(CAN_ID_1, CAN_FILTER0, CAN_FILTER_MASK0,
    CAN_CHANNEL1);
PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER0);
```

Example 3: Configure a CAN filter to accept an Extended ID message within ID range 0x8004000 - 0x8004003

```
/* A message ID range is to be implemented. The 2 least significant bits can be
 * ignored. Therefore, the mask value should be 0x3FFFFFFC. The message type is
 * Extended ID. Configure CAN1, Filter 0, Mask 0 and set the destination channel
 * as Channel 1. */

PLIB_CAN_FilterConfigure(CAN_ID_1, CAN_FILTER0, 0x8004000, CAN_EID);
PLIB_CAN_FilterMaskConfigure(CAN_ID_1, CAN_FILTER_MASK0, 0x3FFFFFFC,
    CAN_EID, CAN_FILTER_MASK_IDE_TYPE);
PLIB_CAN_FilterToChannelLink(CAN_ID_1, CAN_FILTER0, CAN_FILTER_MASK0,
    CAN_CHANNEL1);
PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER0);
```

Note: Refer to About CAN Protocol for more information on Standard and Extended ID addressing.

Enabling Events

The CAN module can generate events that indicate the status of the module. Events can be generated at the channel level and at the module level. Enabling a channel event will cause the CAN module to generate a module level event when the channel event occurs. Enabling a module event will cause the CAN module to interrupt the CPU.

The following code shows an example of using the PLIB_CAN_ChannelEventEnable and PLIB_CAN_ModuleEventEnable functions to enable channel and module events.

```
    /* Enable the Receive channel not empty and Receive channel
     * full events at the channel level. Enable the Receive event
     * at the module level. */
    PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL1,
               (CAN_RX_CHANNEL_NOT_EMPTY|CAN_RX_CHANNEL_FULL));
    PLIB_CAN_ModuleEventEnable (CAN_ID_1, CAN_RX_EVENT);
```

Note: The caller must use the Interrupt Peripheral Library to enable the CAN CPU interrupt.

Transmitting a CAN Message

The caller application can transmit a CAN message by queuing the message in a transmit channel, updating the channel, and then flushing it. Flushing a transmit channel will cause the CAN module to transmit all of the CAN messages that are queued in the channel.

The PLIB_CAN_TransmitBufferGet function should be used to obtain a CAN_TX_MSG_BUFFER handle type to an empty message buffer. The application should then populate this buffer. The PLIB_CAN_ChannelUpdate function must be called when the buffer has been populated. Finally, the PLIB_CAN_TransmitChannelFlush function is called to flush the channel.

The following code shows an example of transmitting an Extended ID message using Channel 0 on the CAN1 module.

```
CAN_TX_MSG_BUFFER * My_Message;

/* Get a pointer to the next buffer in the Transmit channel
 * check if the returned value is null. */
My_Message = PLIB_CAN_TransmitBufferGet(CAN_ID_1, CAN_CHANNEL0);

if(My_Message != NULL)
{
    /* Form a Extended ID CAN message. Start by clearing the buffer.
    Set the IDE bit to 1 to indicate extended ID message. Clear RTR bit
    as this is not a RTR message. */
```

```
    My_Message->messageWord[0] = 0;
    My_Message->messageWord[1] = 0;
    My_Message->messageWord[2] = 0;
    My_Message->messageWord[3] = 0;
    My_Message->msgSID.sid    = 0x200;
    My_Message->msgEID.eid    = 0x4002;
    My_Message->msgEID.ide    = 1;
    My_Message->msgEID.sub_remote_req    = 0;
    My_Message->msgEID.data_length_code  = 1;
    My_Message->data[0]       = 0xAA;

    /* This function lets the CAN module
     know that the message processing is done.*/
    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL0);

    /* Direct the CAN module to flush the
     Transmit channel. This will send any pending
     message in the Transmit channel. */
    PLIB_CAN_TransmitChannelFlush(CAN_ID_1, CAN_CHANNEL0);
}
```

Note: The PLIB_CAN_TransmitBufferGet function will return NULL if the transmit channel is full. In this occurs, the PLIB_CAN_TransmitChannelFlush function should be called to empty the channel.

The following code shows an example of transmitting a Standard ID message using Channel 0 on the CAN1 module.

```
CAN_TX_MSG_BUFFER * myMessage;

/* Get a pointer to the next buffer in the channel
 check if the returned value is null. */
myMessage = PLIB_CAN_TransmitBufferGet(CAN_ID_1, CAN_CHANNEL0);

if(myMessage != NULL)
{

    /* Form a Standard ID CAN message. Start by clearing the buffer.
     Send message to the CAN2 module. IDE = 0 means Standard ID message.
     Send one byte of data.
     */
    myMessage->messageWord[0] = 0;
    myMessage->messageWord[1] = 0;
    myMessage->messageWord[2] = 0;
    myMessage->messageWord[3] = 0;

    myMessage->msgSID.sid     = 0x202;
    myMessage->msgEID.ide     = 0;
    myMessage->msgEID.data_length_code     = 1;
    myMessage->data[0]        = 0x23;

    /* This function lets the CAN module
     know that the message processing is done
     and message is ready to be processed. */
    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL0);

    /* Direct the CAN module to flush the
     Transmit channel. This will send any pending
     message in the Transmit channel. */
    PLIB_CAN_TransmitChannelFlush(CAN_ID_1, CAN_CHANNEL0);
}
```

Receiving a CAN Message

The caller application can use either the interrupt or polling technique to detect when a message has been received.

The following code shows how the PLIB_CAN_ReceivedMessageGet function can be used to read a received message.

```
CAN_RX_MSG_BUFFER * message;

message = (CAN_RX_MSG_BUFFER *)PLIB_CAN_ReceivedMessageGet(CAN_ID_1,
    CAN_CHANNEL1);
if(message != NULL)
```

```
{
    //Process the message received

   /* Call the PLIB_CAN_ChannelUpdate function to let
    the CAN module know that the message processing
    is done. */
    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL1);
}
```

The PLIB_CAN_ReceivedMessageGet function returns an address pointer after converting it from physical address space to virtual address space.

Handling Events

The CAN module can generate events at the channel level and at the module level. An enabled channel event will cause a module event. An enabled module event will cause a CAN CPU interrupt. See Channel Events and Module Events more information on each type of event.

Channel Events

Each channel in the module has its own set of events. The type of events are common to all channels. While a specific channel event may be active and can be inspected, it will cause a module event only if the channel event is enabled.

The following code shows an example of enabling and disabling channel events using the PLIB_CAN_ChannelEventEnable function.

```
/* CAN 1 Channel 1 and 3 are Transmit channels and Channel 2 and 4 are Receive
 channels */
/* Enable Channel 1 empty event and channel not full event. Disable Channel 2 full
 and overflow event */
/* Enable all Transmit events on Channel 2 and disable all RX events on Channel 4 */

PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL1,
    (CAN_TX_CHANNEL_EMPTY | CAN_TX_CHANNEL_NOT_FULL));
PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL2,
    (CAN_RX_CHANNEL_FULL | CAN_RX_CHANNEL_OVERFLOW));
PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL3, CAN_TX_CHANNEL_ANY_EVENT);
PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL4, CAN_RX_CHANNEL_ANY_EVENT);
```

The PLIB_CAN_ModuleEventGet function can be used to check if a specific event is active in a specific channel.

It is only necessary for the event to be active and not enabled for this to work, as shown in the following code example.

```
// Check if RX Channel 2 of CAN module 1 is not empty or if its full.
CAN_CHANNEL_EVENT channelEvent;

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_1, CAN_CHANNEL2);

if((channelEvent & (CAN_RX_CHANNEL_NOT_EMPTY | CAN_RX_CHANNEL_FULL)) != 0)
{
    // This means that either RX Channel 2 is not empty
    // or the Channel is full.
}
```

If a channel event is enabled and is active, it will set a channel event status flag. This flag indicates that some event in the channel is active.

The following code example shows how the PLIB_CAN_AllChannelEventsGet function is used to check if any channel event is active.

```
/* Check if RX Channel 2 of CAN module 1 is not empty or if its full */
CAN_CHANNEL_EVENT channelEvent;

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_1, CAN_CHANNEL2);

if((channelEvent & (CAN_RX_CHANNEL_NOT_EMPTY | CAN_RX_CHANNEL_FULL)) != 0)
{
    // This means that either RX Channel 2 is not empty
    // or the Channel is full.
}
```

A receive channel overflow condition is special channel event and is available as a module event as well. The PLIB_CAN_AllChannelOverflowStatusGet function returns the status of all channels that have an active overflow event. Only channels with enabled overflow events will be reflected.

The following code shows an example of how this function is used.

```
/* Check if Channel 2 or 3 of CAN module 1 have any active events */
CAN_CHANNEL_MASK channelEvents;

channelEvents = PLIB_CAN_AllChannelEventsGet(CAN_ID_1);
```

```
if((channelEvents & (CAN_CHANNEL2_MASK | CAN_CHANNEL3_MASK)) != 0)
{
    // Either Channel 2 or 3 has an active event.
    // The PLIB_CAN_ChannelEventGet function can be
    // used to query the channel for more details.
}
```

Module Events

The CAN module has various sources of events, which includes the channel events. The PLIB_CAN_ModuleEventEnable function can be used to enable and disable module level events. Enabling a module event will cause the CAN module to generate a CPU interrupt, as shown in the following code example.

```
// Enable CAN1 module Transmit Event and Receive Events.
// Disable Receive Overflow event and operation
// mode change event.
PLIB_CAN_ModuleEventEnable(CAN_ID_1, (CAN_TX_EVENT | CAN_RX_EVENT));
PLIB_CAN_ModuleEventDisable(CAN_ID_1,
    (CAN_OPERATION_MODE_CHANGE_EVENT | CAN_RX_OVERFLOW_EVENT));
```

The PLIB_CAN_ModuleEventGet function can be used to check if any module level events are active. This is shown in the following code example.

```
// Check if the CAN Module 1 Receive event
// or if Transmit event is active
if((PLIB_CAN_ModuleEventGet(CAN_ID_1) & (CAN_RX_EVENT | CAN_TX_EVENT)) != 0)
{
    // Handle the Receive or Transmit module Event here.
}
```

The CAN module also provides code to indicate the highest priority pending event in the CAN module. The PLIB_CAN_PendingEventsGet function can be used to read and analyze this code, as shown in the following code example.

```
/* Implement a switch to check and process any active CAN module events */
CAN_EVENT_CODE eventCode;

eventCode = PLIB_CAN_PendingEventsGet(CAN_ID_1);

switch(eventCode)
{
    case CAN_NO_EVENT:
        /* Procedure to handle no CAN events */
        break;
    case CAN_WAKEUP_EVENT:
        /* Procedure to handle device wake-up on CAN bus activity event */
        break;
    default:
        break;
}
```

Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Basic Configuration Functions

	Name	Description
	PLIB_CAN_BusActivityWakeupDisable	Disables the wake-up on bus activity receive line filter.
	PLIB_CAN_BusActivityWakeupEnable	Enables the wake-up on bus activity receive line filter.
	PLIB_CAN_Disable	Disables the specified CAN module.
	PLIB_CAN_Enable	Enables the specified CAN module.
	PLIB_CAN_IsActive	Returns 'true' if the CAN module is active.
	PLIB_CAN_OperationModeGet	Obtains the current CAN operating mode.
	PLIB_CAN_OperationModeSelect	Sets the CAN operating mode.
	PLIB_CAN_StopInIdleDisable	Disables the Stop in Idle feature.
	PLIB_CAN_StopInIdleEnable	Enables the CAN module to stop when the processor enters Idle mode.
	PLIB_CAN_TimeStampDisable	Disables the time stamp feature for the CAN module.

	PLIB_CAN_TimeStampEnable	Enables the time stamp feature for the CAN module.
	PLIB_CAN_ExistsTimeStampPrescaler	Identifies whether the TimeStampPrescaler feature exists on the CAN module.
	PLIB_CAN_TimeStampPrescalerSet	Sets the CAN receive message time stamp timer prescaler.
	PLIB_CAN_MemoryBufferAssign	Assigns buffer memory to the CAN module.

b) Advanced Configuration Functions

	Name	Description
	PLIB_CAN_ChannelResetIsComplete	Returns 'true' if the specified channel reset is complete.
	PLIB_CAN_DeviceNetConfigure	Configures the CAN module DeviceNet(TM) filter feature.
	PLIB_CAN_TimeStampValueGet	Returns the current value of the CAN receive message time stamp timer value.
	PLIB_CAN_TimeStampValueSet	Sets the CAN receive message time stamp timer value.

c) Bus Speed Configuration Functions

	Name	Description
	PLIB_CAN_BusLine3TimesSamplingDisable	Disables the bus line three times sampling.
	PLIB_CAN_BusLine3TimesSamplingEnable	Enables the bus line three times sampling.
	PLIB_CAN_BaudRateGet	Returns the current CAN module Baud rate.
	PLIB_CAN_BaudRatePrescaleSetup	Sets the prescale divisor applied to the CAN module's input clock before it is used to determine the CAN baud rate.
	PLIB_CAN_PrecalculatedBitRateSetup	Sets the desired Baud rate for the respective CAN module.
	PLIB_CAN_BitSamplePhaseSet	Sets the CAN bit-sampling phase parameters.

d) Channel Configuration Functions

	Name	Description
	PLIB_CAN_ChannelForReceiveSet	Configures a CAN channel for receive operation.
	PLIB_CAN_ChannelForTransmitSet	Configures a CAN channel for transmission.
	PLIB_CAN_ChannelReset	Resets a CAN channel.
	PLIB_CAN_ChannelUpdate	Updates the CAN Channel internal pointers.
	PLIB_CAN_ChannelEventDisable	Enables channel level events.

e) Event Management Functions

	Name	Description
	PLIB_CAN_ModuleEventClear	Clears the CAN module level events.
	PLIB_CAN_ModuleEventDisable	Disables the module level events.
	PLIB_CAN_ModuleEventEnable	Enables the module level events.
	PLIB_CAN_ModuleEventGet	Returns the status of the CAN module events.
	PLIB_CAN_AllChannelEventsGet	Returns a value representing the event status of all CAN channels.
	PLIB_CAN_AllChannelOverflowStatusGet	Returns a value representing the receive overflow event status of all CAN channels.
	PLIB_CAN_ChannelEventClear	Clears CAN channel events.
	PLIB_CAN_ChannelEventEnable	Enables channel level events.
	PLIB_CAN_ChannelEventGet	Returns a value representing the event status of a CAN channel.
	PLIB_CAN_PendingEventsGet	Returns a value representing the highest priority active event in the module.

f) Message Transmit Functions

	Name	Description
	PLIB_CAN_PendingTransmissionsAbort	Aborts any pending transmit operations.
	PLIB_CAN_TransmissionIsAborted	Returns 'true' if the transmit abort operation is complete.
	PLIB_CAN_TransmitBufferGet	Returns a pointer to an empty transmit buffer.
	PLIB_CAN_TransmitChannelFlush	Causes all messages in the specified transmit channel to be transmitted.
	PLIB_CAN_TransmitChannelStatusGet	Returns the condition of the transmit channel.
	PLIB_CAN_TransmitErrorCountGet	Returns the CAN transmit error count

g) Message Receive Functions

	Name	Description
	PLIB_CAN_ReceivedMessageGet	Returns a pointer to a message to be read from the CAN channel.

h) Message Filtering Functions

	Name	Description
	PLIB_CAN_FilterConfigure	Configures a CAN message acceptance filter.
	PLIB_CAN_FilterDisable	Disables a CAN message acceptance filter.
	PLIB_CAN_FilterEnable	Enables a CAN message acceptance filter.
	PLIB_CAN_FilterIsDisabled	Returns 'true' if the specified filter is disabled.
	PLIB_CAN_FilterMaskConfigure	Configures a CAN filter mask.
	PLIB_CAN_FilterToChannelLink	Links a filter to a channel.
	PLIB_CAN_LatestFilterMatchGet	Returns the index of the filter that accepted the latest message.

i) Error State Tracking Functions

	Name	Description
	PLIB_CAN_ReceiveErrorCountGet	Returns the CAN receive error count.
	PLIB_CAN_ErrorStateGet	Returns the CAN error status word.

j) Information Functions

	Name	Description
	PLIB_CAN_TotalChannelsGet	Returns the total number of CAN channels per CAN module.
	PLIB_CAN_TotalFiltersGet	Returns the total number of CAN Filters per CAN module.
	PLIB_CAN_TotalMasksGet	Returns the total number of CAN masks per CAN module.

k) Feature Existence Functions

	Name	Description
	PLIB_CAN_ExistsActiveStatus	Identifies whether the ActiveStatus feature exists on the CAN module.
	PLIB_CAN_ExistsAllChannelEvents	Identifies whether the AllChannelEvents feature exists on the CAN module.
	PLIB_CAN_ExistsAllChannelOverflow	Identifies whether the AllChannelOverflow feature exists on the CAN module.
	PLIB_CAN_ExistsBusActivityWakeup	Identifies whether the BusActivityWakeup feature exists on the CAN module.
	PLIB_CAN_ExistsBusLine3TimesSampling	Identifies whether the BusLine3TimesSampling feature exists on the CAN module.
	PLIB_CAN_ExistsChannelEvent	Identifies whether the ChannelEventGet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelEventEnable	Identifies whether the ChannelEventEnable feature exists on the CAN module.
	PLIB_CAN_ExistsChannelForReceiveSet	Identifies whether the ChannelForReceiveSet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelForTransmitSet	Identifies whether the ChannelForTransmitSet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelReset	Identifies whether the ChannelReset feature exists on the CAN module.
	PLIB_CAN_ExistsChannelUpdate	Identifies whether the ChannelUpdate feature exists on the CAN module.
	PLIB_CAN_ExistsDeviceNet	Identifies whether the DeviceNet feature exists on the CAN module.
	PLIB_CAN_ExistsEnableControl	Identifies whether the EnableControl feature exists on the CAN module.
	PLIB_CAN_ExistsErrorState	Identifies whether the ErrorStateGet feature exists on the CAN module.
	PLIB_CAN_ExistsFilterConfigure	Identifies whether the FilterConfigure feature exists on the CAN module.
	PLIB_CAN_ExistsFilterEnable	Identifies whether the FilterEnable feature exists on the CAN module.
	PLIB_CAN_ExistsFilterMaskConfigure	Identifies whether the FilterMaskConfigure feature exists on the CAN module.
	PLIB_CAN_ExistsFilterToChannelLink	Identifies whether the FilterToChannelLink feature exists on the CAN module.
	PLIB_CAN_ExistsLatestFilterMatchGet	Identifies whether the LatestFilterMatchGet feature exists on the CAN module.
	PLIB_CAN_ExistsMemoryBufferAssign	Identifies whether the MemoryBufferAssign feature exists on the CAN module.
	PLIB_CAN_ExistsModuleEventClear	Identifies whether the ModuleEvents feature exists on the CAN module.
	PLIB_CAN_ExistsModuleEventEnable	Identifies whether the ModuleEventEnable feature exists on the CAN module.
	PLIB_CAN_ExistsModuleInfo	Identifies whether the ModuleInformation feature exists on the CAN module.
	PLIB_CAN_ExistsOperationModeRead	Identifies whether the OperationModeRead feature exists on the CAN module.
	PLIB_CAN_ExistsOperationModeWrite	Identifies whether the OperationModeSet feature exists on the CAN module.
	PLIB_CAN_ExistsPendingEventsGet	Identifies whether the PendingEventsGet feature exists on the CAN module.
	PLIB_CAN_ExistsPendingTransmissionsAbort	Identifies whether the PendingTransmissionsAbort feature exists on the CAN module.
	PLIB_CAN_ExistsReceivedMessageGet	Identifies whether the ReceivedMessageGet feature exists on the CAN module.
	PLIB_CAN_ExistsReceiveErrorCount	Identifies whether the ReceiveErrorCount feature exists on the CAN module.
	PLIB_CAN_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CAN module.
	PLIB_CAN_ExistsTimeStampEnable	Identifies whether the TimeStampEnable feature exists on the CAN module

	PLIB_CAN_ExistsTimeStampValue	Identifies whether the TimeStampValue feature exists on the CAN module.
	PLIB_CAN_ExistsTransmissionIsAborted	Identifies whether the TransmissionAbortStatus feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitBufferGet	Identifies whether the TransmitBufferGet feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitChannelFlush	Identifies whether the TransmitChannelFlush feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitChannelStatus	Identifies whether the TransmitChannelStatus feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitErrorCountGet	Identifies whether the TransmitErrorCountGet feature exists on the CAN module.
	PLIB_CAN_ExistsBaudRateGet	Identifies whether the BaudRateGet feature exists on the CAN module.
	PLIB_CAN_ExistsBaudRatePrescaleSetup	Identifies whether the BaudRatePrescaleSetup feature exists on the CAN module.
	PLIB_CAN_ExistsBitSamplePhaseSet	Identifies whether the BitSamplePhaseSet feature exists on the CAN module.
	PLIB_CAN_ExistsPrecalculatedBitRateSetup	Identifies whether the PrecalculatedBitRateSetup feature exists on the CAN module.

l) Data Types and Constants

	Name	Description
	CAN_CHANNEL	Identifies the supported CAN Channels.
	CAN_CHANNEL_EVENT	Identifies all Layer 3 interrupts.
	CAN_CHANNEL_MASK	Lists the series of useful masks.
	CAN_DNET_FILTER_SIZE	Specifies the size of the DeviceNet filter.
	CAN_ERROR_STATE	Specifies the CAN module error states.
	CAN_FILTER	CAN event code returned by the CAN module.
	CAN_FILTER_MASK	Identifies the available CAN filter masks.
	CAN_FILTER_MASK_TYPE	Specifies the CAN filter mask type.
	CAN_ID_TYPE	Specifies the CAN ID type.
	CAN_MODULE_EVENT	Specifies the CAN module events
	CAN_MODULE_ID	Enumeration: CAN_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers. Refer to the specific device data sheet to get the correct number of modules defined for the desired microcontroller.
	CAN_MSG_EID	Defines the structure of the EID word section of the transmit and receive message.
	CAN_OPERATION_MODES	Lists all possible CAN module operational modes.
	CAN_RECEIVE_CHANNEL	Lists all possible CAN module receive channels.
	CAN_RECEIVE_MODES	Lists all possible CAN module receive modes.
	CAN_RX_DATA_MODE	Enables the Data-only Receive mode or Full Receive mode of a CAN receive channel.
	CAN_RX_MSG_BUFFER	Defines the structure of the CAN receive message buffer
	CAN_RX_MSG_SID	Defines the structure of the SID word section of the receive message.
	CAN_TIME_SEGMENT_LENGTH	All possible values for the assignable number of Time Quanta.
	CAN_TX_CHANNEL_STATUS	Identifies possible transmit channel-specific conditions.
	CAN_TX_MSG_BUFFER	Defines the structure of the CAN transmit message buffer.
	CAN_TX_MSG_SID	Defines the structure of the SID word section of the transmit message.
	CAN_TX_RTR	Specifies the status of the CAN Remote Transmit Request (RTR) feature for a CAN transmit channel.
	CAN_TXCHANNEL_PRIORITY	Specifies the priority of a transmit channel.
	CAN_RX_DATA_ONLY_SIZE_BYTES	Used as the size of the CAN data-only receive message buffer in bytes.
	CAN_TX_RX_MESSAGE_SIZE_BYTES	Used as the array size of the CAN transmit and full receive message buffer.

Description

This section describes the Application Programming Interface (API) functions of the CAN Peripheral Library.

Refer to each section for a detailed description.

a) Basic Configuration Functions

PLIB_CAN_BusActivityWakeupDisable Function

Disables the wake-up on bus activity receive line filter.

File

plib_can.h

C

```
void PLIB_CAN_BusActivityWakeupDisable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function disables the Wake up on bus activity receive line filter.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBusActivityWakeup in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_BusActivityWakeupDisable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_BusActivityWakeupDisable (CAN_MODULE_ID index)

PLIB_CAN_BusActivityWakeupEnable Function

Enables the wake-up on bus activity receive line filter.

File

plib_can.h

C

```
void PLIB_CAN_BusActivityWakeupEnable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function enables the wake-up on bus activity receive line filter.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBusActivityWakeup in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_BusActivityWakeupEnable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_BusActivityWakeupEnable (CAN_MODULE_ID index)

PLIB_CAN_Disable Function

Disables the specified CAN module.

File

plib_can.h

C

```
void PLIB_CAN_Disable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function disables the specified CAN module.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

If the user application clears this bit, it may take a number of cycles before the CAN module completes the current transaction and responds to this request. The user application should check this using the PLIB_CAN_IsActive function to verify that the request has been honored.

This function works only if the CAN module is in Configuration mode. Use the PLIB_CAN_OperationModeGet function to get the current mode and the PLIB_CAN_OperationModeSelect function to change the mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_Disable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_Disable (CAN_MODULE_ID index)

PLIB_CAN_Enable Function

Enables the specified CAN module.

File

plib_can.h

C

```
void PLIB_CAN_Enable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function enables the specified CAN module.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

Some of the CAN module settings can be done only when the module is off. Therefore, always call the PLIB_CAN_Enable Function after all other configurations are complete.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

The module should be appropriately configured before being enabled.

Example

```
#define MY_CAN_ID CAN_ID_1

//Do all the other configurations before enabling.

PLIB_CAN_Enable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_Enable (CAN_MODULE_ID index)

PLIB_CAN_IsActive Function

Returns 'true' if the CAN module is active.

File

plib_can.h

C

```
bool PLIB_CAN_IsActive(CAN_MODULE_ID index);
```

Returns

- true = The module is still active
- false = The module is completely disabled

Description

This function returns 'true' if the CAN module is active. This function is typically called after the CAN module is disabled using the PLIB_CAN_Disable function. The CAN module disable request does not complete immediately when requested and depends on the CAN bus status. This function should be called to check whether the module disable is completed.

Remarks

The CAN module disable operation should not be confused with placing the CAN module in the Disable mode using the PLIB_CAN_OperationModeSelect function. The Disable mode is one of the operating modes of the CAN module. The CAN module is still enabled while in Disable mode. The module disable operation switches the CAN module off.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsActiveStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable CAN module 1. Wait until the module is completely disabled.

PLIB_CAN_Disable(CAN_ID_1);
```

```
while(PLIB_CAN_IsActive(CAN_ID_1) == true);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

bool PLIB_CAN_IsActive (CAN_MODULE_ID index)

PLIB_CAN_OperationModeGet Function

Obtains the current CAN operating mode.

File

plib_can.h

C

```
CAN_OPERATION_MODES PLIB_CAN_OperationModeGet(CAN_MODULE_ID index);
```

Returns

The current CAN_OP_MODE type of operation mode of the CAN module.

Description

This function obtains the current CAN operating mode. This function is typically called after the PLIB_CAN_OperationModeSelect function to check if the requested operation mode change is complete.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsOperationModeRead in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check and wait until the CAN module is in Disable Mode.

while(PLIB_CAN_OperationModeGet(CAN_ID_1) != CAN_DISABLE_MODE);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_OPERATION_MODES PLIB_CAN_OperationModeGet (CAN_MODULE_ID index)

PLIB_CAN_OperationModeSelect Function

Sets the CAN operating mode.

File

plib_can.h

C

```
void PLIB_CAN_OperationModeSelect(CAN_MODULE_ID index, CAN_OPERATION_MODES opMode);
```

Returns

None.

Description

This function sets the CAN operating mode. The CAN module requires itself to be in certain modes to gain access to module functionality. For example, the module should be in Normal mode to send and receive messages. Note that after this function is called, it should be checked to determine whether the mode was set by using the PLIB_CAN_OperationModeGet function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsOperationModeWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Set the CAN_ID_1 operating mode to Configuration mode.

PLIB_CAN_OperationModeSelect(CAN_ID_1, CAN_CONFIGURATION_MODE);
```

Parameters

Parameters	Description
module	Identifies the desired CAN module
opMode	Desired CAN_OP_MODE type of the mode to be set

Function

void PLIB_CAN_OperationModeSelect (CAN_MODULE_ID index, CAN_OPERATION_MODES opMode)

PLIB_CAN_StopInIdleDisable Function

Disables the Stop in Idle feature.

File

plib_can.h

C

```
void PLIB_CAN_StopInIdleDisable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function disables the CAN module from stopping operation when the system enters Idle mode. The module continues operation when the system enters Idle mode.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_StopInIdleDisable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_StopInIdleDisable (CAN_MODULE_ID index)

PLIB_CAN_StopInIdleEnable Function

Enables the CAN module to stop when the processor enters Idle mode.

File

plib_can.h

C

```
void PLIB_CAN_StopInIdleEnable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function configures the CAN module to stop operation when the system enters Idle mode.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_StopInIdleEnable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_StopInIdleEnable (CAN_MODULE_ID index)

PLIB_CAN_TimeStampDisable Function

Disables the time stamp feature for the CAN module.

File

plib_can.h

C

```
void PLIB_CAN_TimeStampDisable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function disables time stamp capture feature for the CAN module. This conserves power by stopping the CAN timer.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTimeStampEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_TimeStampDisable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_TimeStampDisable (CAN_MODULE_ID index)

PLIB_CAN_TimeStampEnable Function

Enables the time stamp feature for the CAN module.

File

plib_can.h

C

```
void PLIB_CAN_TimeStampEnable(CAN_MODULE_ID index);
```

Returns

None.

Description

This function enables the time stamp capture feature for the CAN module. The CAN timer value will be stored on valid message reception and is stored with the message.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTimeStampEnable in your application to determine whether this feature is available.

Preconditions

None

Example

```
#define MY_CAN_ID CAN_ID_1

PLIB_CAN_TimeStampEnable(MY_CAN_ID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_TimeStampEnable (CAN_MODULE_ID index)

PLIB_CAN_ExistsTimeStampPrescaler Function

Identifies whether the TimeStampPrescaler feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTimeStampPrescaler(CAN_MODULE_ID index);
```

Returns

- true = The TimeStampPrescaler feature is supported on the device
- false = The TimeStampPrescaler feature is not supported on the device

Description

This function identifies whether the TimeStampPrescaler feature is available on the CAN module. When this function returns true, this function is

supported on the device:

- PLIB_CAN_TimeStampPrescalerSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTimeStampPrescaler(CAN_MODULE_ID index)

PLIB_CAN_TimeStampPrescalerSet Function

Sets the CAN receive message time stamp timer prescaler.

File

plib_can.h

C

```
void PLIB_CAN_TimeStampPrescalerSet(CAN_MODULE_ID index, unsigned preScaler);
```

Returns

None.

Description

This function sets the CAN receive message time stamp timer prescaler. This prescaler determines the rate at the which the time stamp timer is incremented. For example, if the prescaler value is 0xFF, the time stamp timer is incremented by 1 every 255 system clock periods.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTimeStampPrescaler in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Set the CAN_ID_1 Receive Message Time Stamp Timer prescaler to increment
//the Time Stamp Timer every 1024 system clock periods.

PLIB_CAN_TimeStampPrescalerSet(CAN_ID_1, 1024);
```

Parameters

Parameters	Description
module	Identifies the desired CAN module
preScaler	Prescaler for the CAN receive message time stamp timer. This value should be between 0x0 and 0xFFFF.

Function

void PLIB_CAN_TimeStampPrescalerSet(CAN_MODULE_ID index, unsigned preScaler)

PLIB_CAN_MemoryBufferAssign Function

Assigns buffer memory to the CAN module.

File

plib_can.h

C

```
void PLIB_CAN_MemoryBufferAssign(CAN_MODULE_ID index, void * buffer);
```

Returns

None.

Description

This function assigns buffer memory address to the CAN module. The CAN module uses this buffer memory to store messages to be transmitted and received. The size of the memory buffer should be enough to accommodate the required number of message buffers and channels.

Remarks

This API is useful only if the CAN module uses device RAM for message buffers and message FIFO location is software configurable. This should not be used if the device message buffer is part of SFR.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsMemoryBufferAssign in your application to determine whether this feature is available.

Preconditions

The module should be in Configuration mode (using the PLIB_CAN_OperationModeSelect function).

Example

```
// Define and assign a CAN 1 memory buffer for 2 Transmit channels and 1 Receive
// channel, each with 4 message buffers

uint8_t canMemoryBuffer [3 * 4 * CAN_TX_RX_MESSAGE_SIZE_BYTES];

PLIB_CAN_MemoryBufferAssign(CAN_ID_1, canMemoryBuffer);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
buffer	Pointer to the buffer memory area

Function

void PLIB_CAN_MemoryBufferAssign (CAN_MODULE_ID index, void * buffer)

b) Advanced Configuration Functions

PLIB_CAN_ChannelResetIsComplete Function

Returns 'true' if the specified channel reset is complete.

File

plib_can.h

C

```
bool PLIB_CAN_ChannelResetIsComplete(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

- true = Channel reset is complete
- false = Channel reset is in progress

Description

This function returns 'true' if the specified channel reset is complete. This function should preferably be called after calling the PLIB_CAN_ChannelReset function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelReset in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Reset channel 4 of CAN_ID_1 module
// and wait until the reset is complete
PLIB_CAN_ChannelReset(CAN_ID_1,CAN_CHANNEL4);

while(PLIB_CAN_ChannelResetIsComplete(CAN_ID_1,CAN_CHANNEL4) != true);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the CAN Channel to be inspected for reset

Function

bool PLIB_CAN_ChannelResetIsComplete (CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_DeviceNetConfigure Function

Configures the CAN module DeviceNet(TM) filter feature.

File

plib_can.h

C

```
void PLIB_CAN_DeviceNetConfigure(CAN_MODULE_ID index, CAN_DNET_FILTER_SIZE dncnt);
```

Returns

None.

Description

This function configures the CAN module DeviceNet filter feature. DeviceNet filtering allows a portion of the data payload to be used as a filter criteria. This function allows the size of this filter to be configured in bits. The filter can also be disabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsDeviceNet in your application to determine whether this feature is available.

Preconditions

The CAN module should preferably be in Configuration mode (using the PLIB_CAN_OperationModeSelect function).

Example

```
// Disable the CAN_ID_1 DeviceNet filter feature.

PLIB_CAN_DeviceNetConfigure(CAN_ID_1, CAN_DNET_FILTER_DISABLE);

// Set the CAN_ID_2 Device Net Filter Size to 10 bits.

PLIB_CAN_DeviceNetConfigure(CAN_ID_2, CAN_DNET_FILTER_SIZE_10_BIT);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
dncnt	Specifies the CAN_DNET_FILTER_SIZE size of the DeviceNet filter in bits

Function

void PLIB_CAN_DeviceNetConfigure (CAN_MODULE_ID index, CAN_DNET_FILTER_SIZE dncnt)

PLIB_CAN_TimeStampValueGet Function

Returns the current value of the CAN receive message time stamp timer value.

File

plib_can.h

C

```
unsigned PLIB_CAN_TimeStampValueGet(CAN_MODULE_ID index);
```

Returns

The current value of the CAN receive message time stamp timer.

Description

This function gets the current value of the CAN receive message time stamp timer value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTimeStampValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
unsigned int timeStampTimerVal;

// Get the current time stamp timer value.
timeStampTimerVal = PLIB_CAN_TimeStampValueGet(CAN_ID_1);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

unsigned PLIB_CAN_TimeStampValueGet (CAN_MODULE_ID index)

PLIB_CAN_TimeStampValueSet Function

Sets the CAN receive message time stamp timer value.

File

plib_can.h

C

```
void PLIB_CAN_TimeStampValueSet(CAN_MODULE_ID index, unsigned value);
```

Returns

None.

Description

This function sets the CAN receive message time stamp timer value. The timer will then start counting up from this value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTimeStampValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Set start value of CAN_ID_1 Receive Message Time Stamp Timer to 0x100
```

```
PLIB_CAN_TimeStampValueSet(CAN_ID_1, 0x0100);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
value	Start value of the receive message time stamp timer. This value should be between 0x0 and 0xFFFF.

Function

void PLIB_CAN_TimeStampValueSet (CAN_MODULE_ID index, unsigned value)

c) Bus Speed Configuration Functions

PLIB_CAN_BusLine3TimesSamplingDisable Function

Disables the bus line three times sampling.

File

plib_can.h

C

```
void PLIB_CAN_BusLine3TimesSamplingDisable(CAN_MODULE_ID index);
```

Returns

None.

Description

The bus line will be sampled three times at the sampling point. If this is disabled, the bus line will be sampled only once.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This API may not function if the baud rate prescale value is more than a certain limit. Refer to the specific device data sheet to determine the maximum prescale that allows three times sampling.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBusLine3TimesSampling in your application to determine whether this feature is available.

Preconditions

This function works only if the CAN module is in Configuration mode. Use the PLIB_CAN_OperationModeGet function to get the current mode and the PLIB_CAN_OperationModeSelect function to change the mode.

Example

```
#define MY_CAN_ID CAN_ID_1

if (CAN_CONFIGURATION_MODE == PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
PLIB_CAN_BusLine3TimesSamplingDisable(MY_CAN_ID);

}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_BusLine3TimesSamplingDisable (CAN_MODULE_ID index)

PLIB_CAN_BusLine3TimesSamplingEnable Function

Enables the bus line three times sampling.

File

plib_can.h

C

```
void PLIB_CAN_BusLine3TimesSamplingEnable(CAN_MODULE_ID index);
```

Returns

None.

Description

The bus line will be sampled three times at the sampling point. If this is disabled, the bus line will be sampled only once.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This API may not function if the baud rate prescale value is more than a certain limit. Refer to the specific device data sheet to determine the maximum prescale that allows three times sampling.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBusLine3TimesSampling in your application to determine whether this feature is available.

Preconditions

This function works only if the CAN module is in Configuration mode. Use the PLIB_CAN_OperationModeGet function to get the current mode and the PLIB_CAN_OperationModeSelect function to change the mode.

Example

```
#define MY_CAN_ID CAN_ID_1

if (CAN_CONFIGURATION_MODE == PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
PLIB_CAN_BusLine3TimesSamplingEnable(MY_CAN_ID);

}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

void PLIB_CAN_BusLine3TimesSamplingEnable (CAN_MODULE_ID index)

PLIB_CAN_BaudRateGet Function

Returns the current CAN module Baud rate.

File

plib_can.h

C

```
uint32_t PLIB_CAN_BaudRateGet(CAN_MODULE_ID index, uint32_t sysclock);
```

Returns

Current Baud rate value in kbps.

Description

This function returns the current baud rate of the CAN module.

Remarks

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBaudRateGet in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CAN_ID   CAN_ID_1
#define CAN_CLOCK   80000000

uint32_t baudRate;

baudRate = PLIB_CAN_BaudRateGet(MY_CAN_ID, CAN_CLOCK);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
sysclock	CAN input clock frequency

Function

uint32_t PLIB_CAN_BaudRateGet(CAN_MODULE_ID index, uint32_t clock);

PLIB_CAN_BaudRatePrescaleSetup Function

Sets the prescale divisor applied to the CAN module's input clock before it is used to determine the CAN baud rate.

File

plib_can.h

C

```
bool PLIB_CAN_BaudRatePrescaleSetup(CAN_MODULE_ID index, uint16_t prescale);
```

Returns

- true = BaudRate prescale Setup success
- false = BaudRate prescale Setup failure, arguments passed are beyond the possible limits

Description

Sets the prescale divisor applied to the CAN module's input clock before it is used to determine the CAN baud rate.

Remarks

The prescale value is the actual input clock divisor value.

baud rate = input clock / divisor

However, the values must be chosen carefully so that the desired resultant baud rate is an integral number (not fractional).

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBaudRatePrescaleSetup in your application to determine whether this feature is available.

Preconditions

This function works only if the CAN module is in Configuration mode. Use the PLIB_CAN_OperationModeGet function to get the current mode and the PLIB_CAN_OperationModeSelect function to change the mode.

Example

```
#define MY_CAN_ID   CAN_ID_1

#define PRESCALE    7
bool result;
if (CAN_CONFIGURATION_MODE == PLIB_CAN_OperationModeGet(MY_CAN_ID))
{

    result = PLIB_CAN_BaudRatePrescaleSetup(MY_CAN_ID, PRESCALE);

    if(false == result)
    {
        // Error occurred
```

```
    }
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
prescale	Prescale value by which the input clock to the CAN module is divided to determine the CAN baud rate.

Function

bool PLIB_CAN_BaudRatePrescaleSetup(CAN_MODULE_ID index, uint16_t prescale)

PLIB_CAN_PrecalculatedBitRateSetup Function

Sets the desired Baud rate for the respective CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_PrecalculatedBitRateSetup(CAN_MODULE_ID index, uint8_t prescale, uint8_t syncjumpWidth,
uint8_t propagation, uint8_t segment1, uint8_t segment2);
```

Returns

- true = Baud rate setup success
- false = Baud rate setup failure, arguments passed are beyond the possible limits

Description

This function sets the configuration register with the desired Baud rate.

Remarks

This function's parameters must be precalculated for the desired baud rate and properly encoded for the registers of the CAN module that is in use. This function is primarily intended to be used in conjunction with the MPLAB X IDE ECAN Baud Rate calculator or the MPLAB Harmony Configurator (MHC), which precalculates and correctly encodes these values.

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsPrecalculatedBitRateSetup in your application to determine whether this feature is available.

Preconditions

Use the "ECAN Bit Rate Calculator" MPLAB X IDE plug-in to get all of the parameters needed for the function.

This function works only if the CAN module is in Configuration mode. Use the PLIB_CAN_OperationModeGet function to get the current mode and the PLIB_CAN_OperationModeSelect function to change the mode.

Example

```
// Use ECAN Bit Rate Calculator plug-in to get parameter values for 500kbps
// Baud rate.
#define MY_CAN_ID       CAN_ID_1
#define PRESCALE        6
#define SYNCJUMPWIDTH   1
#define PROPAGATION     2
#define SEGMENT1        4
#define SEGMENT2        4
bool result;

if (CAN_CONFIGURATION_MODE == PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
   result = PLIB_CAN_PrecalculatedBitRateSetup( MY_CAN_ID, PRESCALE, SYNCJUMPWIDTH,
                                       PROPAGATION, SEGMENT1, SEGMENT2 );

   if(false == result)
   {
       // Error occurred, handle accordingly
   }
```

```
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
prescale	The CAN module input clock prescale divisor.
syncJumpWidth	The synchronization jump width is the number of time quanta by which the CAN module may adjust phase segment 1 and segment 2 to compensate for jitter in the bit phase.
propagation	Propagation segment length is the number of time quanta allocated to allow for propagation variation in the bit sampling phase.
segment1	Phase segment 1 length(in time quanta) after the propagation segment before sampling of the bit begins.
segment2	Phase segment 2 length (in time quanta) after sampling has begun.

Function

```
bool PLIB_CAN_PrecalculatedBitRateSetup(    CAN_MODULE_ID  index,
uint8_t       prescale,
uint8_t       syncjumpWidth,
uint8_t       propagation,
uint8_t       segment1,
uint8_t       segment2 );
```

PLIB_CAN_BitSamplePhaseSet Function

Sets the CAN bit-sampling phase parameters.

File

plib_can.h

C

```
bool PLIB_CAN_BitSamplePhaseSet(CAN_MODULE_ID index, uint8_t syncJumpWidth, uint8_t propagation, uint8_t
segment1, uint8_t segment2);
```

Returns

- true = BitSamplePhase Setup success
- false = BitSamplePhase Setup failure, arguments passed are beyond the possible limits

Description

This function selects the CAN bit-sampling phase parameters that determine the Baud rate setting.

For a given baud rate setting, a single bit time is divided into a number segments or phases of equal time length called "time quanta". The bit time for a given baud rate (i.e. 1/baud) is divided into a number of time quanta of equal length.

1 time quantum = (one bit time) / (total number of time quanta)

These time quanta are then partitioned into different phases of the bit sample time called synchronization phase (always 1 time quantum long), the propagation phase, and the segment 1 and segment 2 phases. Thus, the total number of time quanta equals the sum of the time quanta allocated to each phase.

total number of time quanta = (1 + propagation + segment1 + segment2)

(Sampling of the bit occurs at the end of the segment 1 phase.)

This function determines the length of each phase (in time quanta) as well as the synchronization jump width, the number of time quanta by which the CAN module may move time from segment 1 to segment 2 (or vice versa) to adjust for jitter in the bit sampling phase timing.

Remarks

The values of the syncJumpWidth, propagation, segment1 and segment2 parameters should passed as a number of time quanta for each. These values will be encoded by this function correctly for the the part in use before they are written to the registers of the CAN peripheral. To pass in precalculated values that are already encoded for the physical registers of the CAN controller (such as the values provided by the ECAN Baud Rate Calculator or the MPLAB Harmony Configurator (MHC) utility), use the PLIB_CAN_PrecalculatedBitRateSetup function instead.

The MY_CAN_ID macro in the example above is used as a place holder for the actual CAN bus ID (as defined by the processor-specific CAN_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsBitSamplePhaseSet in your application to determine whether this feature is available.

Preconditions

This function works only if the CAN module is in Configuration mode. Use the PLIB_CAN_OperationModeGet function to get the current mode and the PLIB_CAN_OperationModeSelect function to change the mode.

Example

```
#define MY_CAN_ID       CAN_ID_1

#define SYNCJUMPWIDTH   1
#define PROPAGATION     2
#define SEGMENT1        4
#define SEGMENT2        4
bool result;
if (CAN_CONFIGURATION_MODE == PLIB_CAN_OperationModeGet(MY_CAN_ID))
{
   result = PLIB_CAN_BitSamplePhaseSet(MY_CAN_ID, SYNCJUMPWIDTH, PROPAGATION,
                                       SEGMENT1, SEGMENT2);

   if(false == result)
   {
       // Error occurred
   }
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
syncJumpWidth	The synchronization jump width is the number of time quanta by which the CAN module may adjust phase segment 1 and segment 2 to compensate for jitter in the bit phase to achieve a valid sampling point.
propagation	Propagation segment length is the number of time quanta allocated to allow for propagation variation in the bit sampling phase.
segment1	Phase segment 1 length(in time quanta) after the propagation segment before sampling of the bit begins.
segment2	Phase segment 2 length (in time quanta) after sampling has begun.

Function

bool PLIB_CAN_BitSamplePhaseSet(CAN_MODULE_ID index,
uint8_t syncJumpWidth,
uint8_t propagation,
uint8_t segment1,
uint8_t segment2)

d) Channel Configuration Functions

PLIB_CAN_ChannelForReceiveSet Function

Configures a CAN channel for receive operation.

File

plib_can.h

C

```
void PLIB_CAN_ChannelForReceiveSet(CAN_MODULE_ID index, CAN_CHANNEL channel, uint32_t channelSize,
CAN_RX_DATA_MODE dataOnly);
```

Returns

None.

Description

This function configures a CAN channel for receive operation. A receive channel can be either a full CAN message receive channel, which receives an entire CAN message (Arbitration field + Data field) or a data-only message channel, which receives only the data payload section of

the message. A receive channel can buffer up to 32 messages. The number of messages to buffer (i.e., the size of the channel) is set by the channelSize parameter.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelForReceiveSet in your application to determine whether this feature is available.

Preconditions

The CAN module should be in Configuration mode. This is achieved by using the PLIB_CAN_OperationModeSelect function.

Example

```
// Configure channel 4 of CAN module for receive operation. Set channel
// size to buffer 10 messages.
// Channel should be data only receive channel

PLIB_CAN_ChannelForReceiveSet(CAN_ID_1, CAN_CHANNEL4, 10, CAN_RX_DATA_ONLY);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Receive channel
channelSize	Specifies the number of received messages that the channel can buffer before it overflows. This should be a value between 1 and 32.
dataOnly	Specifies a full CAN message Receive channel or a data-only message channel. • CAN_RX_DATA_ONLY - Channel will be a data-only message receive channel • CAN_RX_FULL_RECEIVE - Channel will be a full CAN message receive channel

Function

void PLIB_CAN_ChannelForReceiveSet(CAN_MODULE_ID index, CAN_CHANNEL channel,
uint32_t channelSize, CAN_RX_DATA_MODE dataOnly)

PLIB_CAN_ChannelForTransmitSet Function

Configures a CAN channel for transmission.

File

plib_can.h

C

```
void PLIB_CAN_ChannelForTransmitSet(CAN_MODULE_ID index, CAN_CHANNEL channel, uint8_t channelSize,
CAN_TX_RTR rtren, CAN_TXCHANNEL_PRIORITY priority);
```

Returns

None.

Description

This function configures a CAN Channel for transmission. The size of the channel specifies the number of messages that the channel can buffer. The channel is a First In First Out (FIFO) type of buffer. The remote transmit request feature allows the transmit channel to start transmitting when an associated filter has received a message. The transmit channel priority determines the priority as compared to other transmit channels. If two transmit channels have the same priority, the natural channel priority determines which channel transmits first.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelForTransmitSet in your application to determine whether this feature is available.

Preconditions

The CAN Module should be in Configuration mode. This is achieved by using the PLIB_CAN_OperationModeSelect function.

Example

```
// Configure CAN 1 Channel 2 for Transmit operation. Set the channel size to 15
```

```
// and enable remote transmit request. Set the priority to low medium
// priority.

PLIB_CAN_ChannelForTransmitSet (CAN_ID_1, CAN_CHANNEL2, 15,
    CAN_TX_RTR_ENABLED, CAN_LOW_MEDIUM_PRIORITY);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel
channelSize	Size of the channel in messages. This value should be between 1 and 32.
rtren	Enables disables Remote Transmit Request: • CAN_TX_RTR_ENABLED - Remote Transmit Request is enabled • CAN_TX_RTR_DISABLED - Remote Transmit Request is disabled
priority	Specifies the priority of the Transmit channel

Function

void PLIB_CAN_ChannelForTransmitSet (CAN_MODULE_ID index,
CAN_CHANNEL channel, uint8_t channelSize,
CAN_TX_RTR rtren, CAN_TXCHANNEL_PRIORITY priority)

PLIB_CAN_ChannelReset Function

Resets a CAN channel.

File

plib_can.h

C

```
void PLIB_CAN_ChannelReset(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

None.

Description

This function resets a CAN channel. Resetting a CAN channel causes it to discard any unread received messages or any messages that have not been transmitted yet. The reset operation will wait if a message is being currently transmitted or is being received. The PLIB_CAN_ChannelResetIsComplete function can be used to check if the channel reset is complete.

Remarks

Attempting to read from a channel that is reset will return messages that may have already been read or may not have been read at all.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelReset in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Reset channel 4 of CAN_ID_1 module
// and wait until the reset is complete

PLIB_CAN_ChannelReset(CAN_ID_1,CAN_CHANNEL4);

while(PLIB_CAN_ChannelResetIsComplete(CAN_ID_1,CAN_CHANNEL4) != true);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the CAN channel to be reset

Function

void PLIB_CAN_ChannelReset(CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_ChannelUpdate Function

Updates the CAN Channel internal pointers.

File

plib_can.h

C

```
void PLIB_CAN_ChannelUpdate(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

None.

Description

This function updates the CAN Channel internal pointers. This function should be called when a message has been read or processed completely from a CAN receive channel (using the PLIB_CAN_ReceivedMessageGet function). It should be called after a new message has been written to a CAN transmit channel (using the PLIB_CAN_TransmitBufferGet function) and before the PLIB_CAN_TransmitChannelFlush function.

Trying to read a CAN receive channel that has not been updated will cause the PLIB_CAN_ReceivedMessageGet function to return the same message. Writing to a CAN transmit channel that has not been updated will cause the last written message to be overwritten.

Remarks

The PLIB_CAN_ChannelUpdate function should be called on a CAN receive channel only after the message obtained using the PLIB_CAN_ReceivedMessageGet function has been read or processed completely. The CAN module peripheral library does not provide access to older messages once the PLIB_CAN_ChannelUpdate function has been called.

When using the PLIB_CAN_TransmitMessageBuffer function to write to a CAN transmit channel the PLIB_CAN_ChannelUpdate function should be called only when a valid message has been written to the channel. Calling the update function without writing to the CAN channel will cause an incorrect message to be output on the CAN channel when the transmit channel is flushed.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelUpdate in your application to determine whether this feature is available.

Preconditions

This function is effective only when the CAN module is not in Configuration mode or Disable mode.

Example

```
// CAN_ID_1 Channel 1 is a Receive channel and Channel 2 is a Transmit
// channel. Read and update Channel 1. Write a message to Channel 2 and then
// update and flush the channel.

CAN_TX_MSG_BUFFER * TransmitMessage;
CAN_RX_MSG_BUFFER * rxMessage;

rxMessage = PLIB_CAN_ReceivedMessageGet(CAN_ID_1, CAN_CHANNEL1);

if(rxMessage != NULL)
{
    // Process the received message.

    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL1);
}

TransmitMessage = PLIB_CAN_TransmitBufferGet(CAN_ID_1, CAN_CHANNEL2);
if(TransmitMessage != NULL)
{
    // Write a message to buffer

    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL2);
    PLIB_CAN_TransmitChannelFlush(CAN_ID_1, CAN_CHANNEL2);
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the CAN channel to be updated

Function

void PLIB_CAN_ChannelUpdate (CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_ChannelEventDisable Function

Enables channel level events.

File

plib_can.h

C

```
void PLIB_CAN_ChannelEventDisable(CAN_MODULE_ID index, CAN_CHANNEL channel, CAN_CHANNEL_EVENT flags);
```

Returns

None.

Description

This function disables channel level events. Any enabled channel event will cause a CAN module event.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelEventEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CAN_ChannelEventDisable(CAN_ID_1, CAN_CHANNEL1, CAN_TX_CHANNEL_EMPTY);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel
flags	Identifies the CAN channel event(s) to be affected. Several events can be controlled by logically ORed combination of events.

Function

void PLIB_CAN_ChannelEventDisable (CAN_MODULE_ID index, CAN_CHANNEL channel,
CAN_CHANNEL_EVENT flags)

e) Event Management Functions

PLIB_CAN_ModuleEventClear Function

Clears the CAN module level events.

File

plib_can.h

C

```
void PLIB_CAN_ModuleEventClear(CAN_MODULE_ID index, CAN_MODULE_EVENT flags);
```

Returns

None.

Description

This function clears module level events. If the event condition is persistent, clearing the event will have no effect. The event condition itself should be cleared. The CAN_SYSTEM_ERROR_EVENT can only be cleared by disabling the CAN module using the PLIB_CAN_Enable function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleEventClear in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Clear CAN_ID_1 Transmit Event and Receive Events.

PLIB_CAN_ModuleEventClear(CAN_ID_1, (CAN_TX_EVENT | CAN_RX_EVENT));
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
flags	Identifies the CAN module level events to be affected. Several events can be cleared by specifying a logically ORed combination of events.

Function

void PLIB_CAN_ModuleEventClear (CAN_MODULE_ID index, CAN_MODULE_EVENT flags)

PLIB_CAN_ModuleEventDisable Function

Disables the module level events.

File

plib_can.h

C

```
void PLIB_CAN_ModuleEventDisable(CAN_MODULE_ID index, CAN_MODULE_EVENT flags);
```

Returns

None.

Description

This function disables module level events. Any enabled module event will cause the CAN module to generate a CPU interrupt.

Remarks

An event can be active regardless of it being enabled or disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleEventEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable CAN_ID_1 Transmit Event and Receive Events.
// Disable Receive Overflow event and operation
// mode change event.

PLIB_CAN_ModuleEventDisable(CAN_ID_1, (CAN_TX_EVENT | CAN_RX_EVENT));
PLIB_CAN_ModuleEventDisable(CAN_ID_1, (CAN_OPERATION_MODE_CHANGE_EVENT |
    CAN_RX_OVERFLOW_EVENT));
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

flags	Identifies the CAN module level events to be affected. Several events can be controlled by logically ORed combination of events.

Function

void PLIB_CAN_ModuleEventDisable (CAN_MODULE_ID index, CAN_MODULE_EVENT flags)

PLIB_CAN_ModuleEventEnable Function

Enables the module level events.

File

plib_can.h

C

```
void PLIB_CAN_ModuleEventEnable(CAN_MODULE_ID index, CAN_MODULE_EVENT flags);
```

Returns

None.

Description

This function enables module level events. Any enabled module event will cause the CAN module to generate a CPU interrupt.

Remarks

An event can be active regardless of it being enabled or disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleEventEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable CAN_ID_1 Transmit Event and Receive Events.
// Disable Receive Overflow event and operation
// mode change event.

PLIB_CAN_ModuleEventEnable(CAN_ID_1, (CAN_TX_EVENT | CAN_RX_EVENT));
PLIB_CAN_ModuleEventEnable(CAN_ID_1, (CAN_OPERATION_MODE_CHANGE_EVENT |
    CAN_RX_OVERFLOW_EVENT));
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
flags	Identifies the CAN module level events to be affected. Several events can be controlled by logically ORed combination of events.

Function

void PLIB_CAN_ModuleEventEnable (CAN_MODULE_ID index, CAN_MODULE_EVENT flags)

PLIB_CAN_ModuleEventGet Function

Returns the status of the CAN module events.

File

plib_can.h

C

```
CAN_MODULE_EVENT PLIB_CAN_ModuleEventGet(CAN_MODULE_ID index);
```

Returns

A status word representing the status of the CAN module events.

Description

This function returns the status of the CAN module events. The routine returns a status word. This word should be logically ANDed with the desired CAN_MODULE_EVENT event mask. A non-zero result of such an operation would mean that the events specified in the event mask are active. An event mask can contain one event or can be a logical OR combination of multiple events.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleEventClear in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if the CAN_ID_1 Module Receive event
// or if Transmit event is active

if((PLIB_CAN_ModuleEventGet(CAN_ID_1) & (CAN_RX_EVENT | CAN_TX_EVENT)) != 0)
{
    // Handle the Receive or Transmit module Event here.

}

// Check if the CAN_ID_2 System Error Event
// is active

if((PLIB_CAN_ModuleEventGet(CAN_ID_2) & CAN_SYSTEM_ERROR_EVENT) != 0)
{
    // CAN_ID_2 System error event is active.

}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_MODULE_EVENT PLIB_CAN_ModuleEventGet(CAN_MODULE_ID index)

PLIB_CAN_AllChannelEventsGet Function

Returns a value representing the event status of all CAN channels.

File

plib_can.h

C

```
CAN_CHANNEL_MASK PLIB_CAN_AllChannelEventsGet(CAN_MODULE_ID index);
```

Returns

Returns a value that can be logically ANDed with a CAN_CHANNEL_MASK mask value to check if any event on a channel is active.

Description

This function returns a value representing the event status of all of the CAN channels. The return value can be logically ANDed with a CAN_CHANNEL_MASK type to check whether the channel has any active events. Only an enabled channel event will cause the bit to be updated.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsAllChannelEvents in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if Channel 2 or 3 of CAN_ID_1 module
// have any active events

CAN_CHANNEL_MASK channelEvents;

channelEvents = PLIB_CAN_AllChannelEventsGet(CAN_ID_1);

if((channelEvents & (CAN_CHANNEL2_MASK | CAN_CHANNEL3_MASK)) != 0)
{
    // Either Channel 2 or 3 has an active event.
    // The PLIB_CAN_ChannelEventGet function can be
    // used to query the channel for more details.
}

// Check if Channel 31 of CAN_ID_2 module
// has an any active events

channelEvents = PLIB_CAN_AllChannelEventsGet(CAN_ID_2);

if((channelEvents & CAN_CHANNEL31_MASK) != 0)
{
    // Channel 31 of CAN_ID_2 module
    // has an active event.
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_CHANNEL_MASK PLIB_CAN_AllChannelEventsGet(CAN_MODULE_ID index)

PLIB_CAN_AllChannelOverflowStatusGet Function

Returns a value representing the receive overflow event status of all CAN channels.

File

plib_can.h

C

```
CAN_CHANNEL_MASK PLIB_CAN_AllChannelOverflowStatusGet(CAN_MODULE_ID index);
```

Returns

Returns a value that can be logically ANDed with a CAN_CHANNEL_MASK mask value to check if a receive channel overflow event is active.

Description

This function returns a value representing the Receive overflow event status of all the CAN Channels. The return value can be logically ANDed with a CAN_CHANNEL_MASK type to check whether a channel has any active receive overflow events. The return value will reflect the channel status only if the receive overflow event of the channel is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsAllChannelOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if Receive Channel 2 or 3 of CAN_ID_1 module
// have overflowed

CAN_CHANNEL_MASK channelOverflowEvent;
```

```
channelOverflowEvent = PLIB_CAN_AllChannelOverflowStatusGet(CAN_ID_1);

if((channelOverflowEvent & (CAN_CHANNEL2_MASK | CAN_CHANNEL3_MASK)) != 0)
{
    // Either Receive Channel 2 or 3 has  overflowed.
    // The PLIB_CAN_ChannelEventGet function can be
    // used to query the channel for more details.
}

// Check if Receive Channel 31 of CAN_ID_2 module
// has overflowed

channelOverflowEvent = PLIB_CAN_AllChannelOverflowStatusGet(CAN_ID_2);

if((channelOverflowEvent & CAN_CHANNEL31_MASK) != 0)
{
    // Channel 31 of CAN_ID_2 module
    // has overflowed.
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_CHANNEL_MASK PLIB_CAN_AllChannelOverflowStatusGet (CAN_MODULE_ID index)

PLIB_CAN_ChannelEventClear Function

Clears CAN channel events.

File

plib_can.h

C

```
void PLIB_CAN_ChannelEventClear(CAN_MODULE_ID index, CAN_CHANNEL channel, CAN_CHANNEL_EVENT events);
```

Returns

None.

Description

This function clears channel events. The events to be cleared are specified as mask. Note that only the receive overflow event is clearable. Attempting to clear other events will have no effect since these events are persistent and clear only when the event condition is cleared.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelEvent in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Clear CAN_ID_2 Receive Channel 3 overflow event

PLIB_CAN_ChannelEventClear(CAN_ID_2, CAN_CHANNEL3, CAN_RX_CHANNEL_OVERFLOW);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel
events	Mask specifying the events to be cleared

Function

void PLIB_CAN_ChannelEventClear (CAN_MODULE_ID index, CAN_CHANNEL channel, CAN_CHANNEL_EVENT events)

PLIB_CAN_ChannelEventEnable Function

Enables channel level events.

File

plib_can.h

C

```
void PLIB_CAN_ChannelEventEnable(CAN_MODULE_ID index, CAN_CHANNEL channel, CAN_CHANNEL_EVENT flags);
```

Returns

None.

Description

This function enables channel level events. Any enabled channel event will cause a CAN module event. An event can be active regardless of it being enabled or disabled. Enabling a transmit type of event for a receive channel will have no any effect.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelEventEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// CAN_ID_1 Channel 1 and 3 are Transmit channels and Channel 2  and 4 are
// Receive channels. Enable Channel 1 empty event and channel not full
// event. Disable Channel 2 full and overflow event.
// Enable all Transmit events on Channel 2 and disable all Receive events on
// on Channel 4.

PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL1, (CAN_TX_CHANNEL_EMPTY |
    CAN_TX_CHANNEL_NOT_FULL));

PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL2, (CAN_RX_CHANNEL_FULL |
    CAN_RX_CHANNEL_OVERFLOW));

PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL3, CAN_TX_CHANNEL_ANY_EVENT);
PLIB_CAN_ChannelEventEnable(CAN_ID_1, CAN_CHANNEL4, CAN_RX_CHANNEL_ANY_EVENT);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel
flags	Identifies the CAN channel event(s) to be affected. Several events can be controlled by logically ORed combination of events.

Function

void PLIB_CAN_ChannelEventEnable (CAN_MODULE_ID index, CAN_CHANNEL channel,
CAN_CHANNEL_EVENT flags)

PLIB_CAN_ChannelEventGet Function

Returns a value representing the event status of a CAN channel.

File

plib_can.h

C

```
CAN_CHANNEL_EVENT PLIB_CAN_ChannelEventGet(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

Returns a value that can be logically ANDed with a CAN_CHANNEL_EVENT type to check if specific CAN channel events are active.

Description

This function returns a value representing the event status of a CAN channel. The return value can be logically ANDed with CAN_CHANNEL_EVENT type to check for a specific event(s). Channels events are affected regardless of whether the event itself is enabled or disabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsChannelEvent in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if Receive Channel 2 of CAN_ID_1 module
// is not empty or if its full.

CAN_CHANNEL_EVENT channelEvent;

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_1,CAN_CHANNEL2);

if((channelEvent & (CAN_RX_CHANNEL_NOT_EMPTY | CAN_RX_CHANNEL_FULL)) != 0)
{
    // This means that either Receive Channel 2 is not empty
    // or the Channel is full.


}

// Check if Transmit Channel 3 of CAN_ID_2 module
// has any active events

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_2, CAN_CHANNEL3);

if((channelEvent & CAN_TX_CHANNEL_ANY_EVENT) != 0)
{
    // This means that some event is active

}

// Check if Transmit Channel 6 of CAN_ID_2 module is not full

channelEvent = PLIB_CAN_ChannelEventGet(CAN_ID_2, CAN_CHANNEL6);

if((channelEvent & CAN_TX_CHANNEL_NOT_FULL) != 0)
{
    // This means the Channel is not full.

}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel

Function

CAN_CHANNEL_EVENT PLIB_CAN_ChannelEventGet (CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_PendingEventsGet Function

Returns a value representing the highest priority active event in the module.

File

plib_can.h

C

```
CAN_EVENT_CODE PLIB_CAN_PendingEventsGet(CAN_MODULE_ID index);
```

Returns

Returns a CAN_EVENT_CODE type representing the highest priority active event in the module.

Description

This function returns a value representing the highest priority active event in the module. The return value is a CAN_EVENT_CODE type.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsPendingEventsGet in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Implement a switch to check and process
// any active CAN module events.

CAN_EVENT_CODE eventCode;

eventCode = PLIB_CAN_PendingEventsGet(CAN_ID_1);

switch(eventCode)
    {
        case CAN_NO_EVENT:
            // Procedure to handle no CAN events
            break;

        case CAN_WAKEUP_EVENT:
            // Procedure to handle device wake-up
            // on CAN bus activity event
            break;

        default:
            break;
    }
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_EVENT_CODE PLIB_CAN_PendingEventsGet (CAN_MODULE_ID index)

f) Message Transmit Functions

PLIB_CAN_PendingTransmissionsAbort Function

Aborts any pending transmit operations.

File

plib_can.h

C

```
void PLIB_CAN_PendingTransmissionsAbort(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

None.

Description

This function aborts any pending transmit operations. Any messages that are yet to be transmitted will not be transmitted. The messages will still be present in the respective channel. Any message that is in the process of being transmitted will be transmitted completely. Either one channel or all channels can be specified.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsPendingTransmissionsAbort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Abort any pending transmissions on CAN_ID_1 Channel 4 and
// Channel 5.
PLIB_CAN_PendingTransmissionsAbort(CAN_ID_1, CAN_CHANNEL4);
PLIB_CAN_PendingTransmissionsAbort(CAN_ID_1, CAN_CHANNEL5);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel. By specifying CAN_ALL_CHANNELS, transmission on all transmit channels will be aborted.

Function

void PLIB_CAN_PendingTransmissionsAbort (CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_TransmissionIsAborted Function

Returns 'true' if the transmit abort operation is complete.

File

plib_can.h

C

```
bool PLIB_CAN_TransmissionIsAborted(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

- true - If channel = CAN_ALL_CHANNELS, Transmit Abort is complete. If channel = CAN_CHANNELx, Transmit Abort was successful.
- false - If channel = CAN_ALL_CHANNELS, Transmit Abort is in progress. If channel = CAN_CHANNELx, Transmit Abort was not successful.

Description

This function returns 'true' if the transmit abort operation is complete. Either individual channels can be specified or all channels can be specified.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTransmissionIsAborted in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Abort any pending transmissions on CAN_ID_1 Channel 4 and
// check if the current message transmission was aborted.

PLIB_CAN_PendingTransmissionsAbort(CAN_ID_1, CAN_CHANNEL4);
```

```
if(PLIB_CAN_TransmissionIsAborted(CAN_ID_1, CAN_CHANNEL4) == false)
{
    // The message was not aborted.
}
else
{
    // The message was aborted.
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel. By specifying CAN_ALL_CHANNELS the status of transmit abort on all transmit channels will be returned.

Function

bool PLIB_CAN_TransmissionIsAborted (CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_TransmitBufferGet Function

Returns a pointer to an empty transmit buffer.

File

plib_can.h

C

```
CAN_TX_MSG_BUFFER * PLIB_CAN_TransmitBufferGet(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

Returns a CAN_TX_MSG_BUFFER type pointer to an empty message buffer in the Transmit channel. Returns NULL if the channel is full and there aren't any empty message buffers.

Description

This function returns a pointer to an empty transmit buffer. The routine will return a NULL pointer if there aren't any empty transmit buffers. In such a case, the application should flush the channel and wait until the transmit channel has at least one empty buffer. In order to function correctly, it is essential that the PLIB_CAN_ChannelUpdate function is called in the proper sequence for the PLIB_CAN_TransmitBufferGet function to return a pointer to an empty buffer.

Remarks

Calling the PLIB_CAN_TransmitBufferGet function on a channel that has not been updated after a message was written to the channel, will cause the function to return a pointer to the written message in case of transmit buffer FIFO. Therefore, a non-transmitted message could get overwritten.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTransmitBufferGet in your application to determine whether this feature is available.

Preconditions

PLIB_CAN_MemoryBufferAssign must be called if the 'transmit buffer' should be in the device RAM. It is not required if the 'transmit buffer' is in SFR space.

Example

```
// Transmit a message through CAN_ID_1 Channel 4

CAN_TX_MSG_BUFFER * msgBuffer;

msgBuffer = PLIB_CAN_TransmitBufferGet(CAN_ID_1, CAN_CHANNEL4);

if(msgBuffer != NULL)
{
    // Load the message here

}
else
{
    // No space in the channel
    // wait until a message
```

```
    // buffer is free.

    while((PLIB_CAN_ChannelEventGet(CAN_ID_1, CAN_CHANNEL4) &
        CAN_TX_CHANNEL_NOT_FULL) == 0);
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN Channel

Function

CAN_TX_MSG_BUFFER * PLIB_CAN_TransmitBufferGet(CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_TransmitChannelFlush Function

Causes all messages in the specified transmit channel to be transmitted.

File

plib_can.h

C

```
void PLIB_CAN_TransmitChannelFlush(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

None.

Description

This function causes all messages in the specified transmit channel to be transmitted. All messages in the channel at the time of the flush operation will be transmitted. The transmit channel flush operation should preferably be called immediately after the PLIB_CAN_ChannelUpdate function. This will ensure that messages are transmitted promptly.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTransmitChannelFlush in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Flush CAN_ID_1 Transmit Channel 4.

PLIB_CAN_TransmitChannelFlush(CAN_ID_1, CAN_CHANNEL4);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel

Function

void PLIB_CAN_TransmitChannelFlush (CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_TransmitChannelStatusGet Function

Returns the condition of the transmit channel.

File

plib_can.h

C

```
CAN_TX_CHANNEL_STATUS PLIB_CAN_TransmitChannelStatusGet(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

Returns a status word that can be logically ANDed with the CAN_TX_CHANNEL_STATUS type to check whether a condition exists.

Description

This function returns the condition of the transmit channel. The return value can be logically ANDed with CAN_TX_CHANNEL_STATUS type values.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTransmitChannelStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if CAN_ID_1 Transmit Channel 4
// is still transmitting

CAN_TX_CHANNEL_STATUS status;

status = PLIB_CAN_TransmitChannelStatusGet(CAN_ID_1, CAN_CHANNEL4);

if((status & CAN_TX_CHANNEL_TRANSMITTING) != 0)
{
    // This means that channel is still
    // transmitting.

}

// Check if CAN_ID_2 Transmit Channel 5 has lost arbitration
// or other Transmit errors.

status = PLIB_CAN_TransmitChannelStatusGet(CAN_ID_2, CAN_CHANNEL5);

if((status & (CAN_TX_CHANNEL_ARBITRATION_LOST | CAN_TX_CHANNEL_ERROR)) != 0)
{
    // This means that the Transmit channel has either
    // lost arbitration or a Transmit error has occurred.
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
channel	Identifies the desired CAN channel

Function

CAN_TX_CHANNEL_STATUS PLIB_CAN_TransmitChannelStatusGet(CAN_MODULE_ID index, CAN_CHANNEL channel)

PLIB_CAN_TransmitErrorCountGet Function

Returns the CAN transmit error count

File

plib_can.h

C

```
int PLIB_CAN_TransmitErrorCountGet(CAN_MODULE_ID index);
```

Returns

Returns the CAN transmit error count.

Description

This function returns the CAN transmit error count. Refer to CAN 2.0B specification for more details on the CAN transmit error count and its

significance.

Remarks

There are multiple bus conditions, which could cause the transmit error count to increase. Please refer to the CAN 2.0B specification for details.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsTransmitErrorCountGet in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if the CAN_ID_1 Transmit error count is more than 200
int errorCount;

errorCount = PLIB_CAN_TransmitErrorCountGet(CAN_ID_1);
if(errorCount > 200)
{
    // This error count is high.
    // Do some diagnostics.
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

int PLIB_CAN_TransmitErrorCountGet(CAN_MODULE_ID index)

g) Message Receive Functions

PLIB_CAN_ReceivedMessageGet Function

Returns a pointer to a message to be read from the CAN channel.

File

plib_can.h

C

```
CAN_RX_MSG_BUFFER * PLIB_CAN_ReceivedMessageGet(CAN_MODULE_ID index, CAN_CHANNEL channel);
```

Returns

Returns a pointer to a message to be read from the CAN channel; returns a CAN_RX_MSG_BUFFER type pointer to a received CAN message. If the receive channel is a full CAN message receive channel, the caller should use the msgSID, msgEID and data members of the CAN_RX_MSG_BUFFER data structure to access the received CAN message. If the receive channel is a data-only channel, the message will only contain 8 payload data bytes (even if the message was placed on the bus with less than 8 bytes). The caller in this case should use the dataOnlyMsgData member of the CAN_RX_MSG_BUFFER data structure to read the data contained in the received CAN message.

Description

This function returns a CAN_RX_MSG_BUFFER pointer to a message to be read from the CAN channel. The PLIB_CAN_ChannelUpdate routine should be called after the received message has been processed.

Remarks

The CAN receive channel is configured as a full CAN message receive channel or a data-Only channel while configuring the channel using the PLIB_CAN_ChannelForReceiveSet function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsReceivedMessageGet in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Read a message from the CAN_ID_1 Channel 8
```

```
// which is configured as full CAN message
// receive channel.

CAN_RX_MSG_BUFFER * receivedMsg;

receivedMsg = (CAN_RX_MSG_BUFFER *)PLIB_CAN_ReceivedMessageGet(CAN_ID_1,
    CAN_CHANNEL8);

if(receivedMsg != NULL)
{
    // rxMsg is pointing to
    // a CAN message. Process
    // the message and then update
    // the CAN channel.


    PLIB_CAN_ChannelUpdate(CAN_ID_1, CAN_CHANNEL8);
}

// Read a message from the CAN_ID_2 Channel 9
// which is configured as data only message
// receive channel. Access the message
// as bytes;

CAN_RX_MSG_BUFFER * rxMsg;

rxMsg = PLIB_CAN_ReceivedMessageGet(CAN_ID_2, CAN_CHANNEL9);

if(rxMsg != NULL)
{
    // rxMsg is pointing to
    // a CAN message. Process
    // the message and then update
    // the CAN channel.

    // rxMsg->dataOnlyMsgData[0] is the first byte of message
    // data payload. rxMsg->dataOnlyMsgData[0] is the second
    // byte of message data payload and so on.


    PLIB_CAN_ChannelUpdate(CAN_ID_2, CAN_CHANNEL9);
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module.
channel	Identifies the desired CAN Receive channel.

Function

CAN_RX_MSG_BUFFER * PLIB_CAN_ReceivedMessageGet(CAN_MODULE_ID index, CAN_CHANNEL channel)

h) Message Filtering Functions

PLIB_CAN_FilterConfigure Function

Configures a CAN message acceptance filter.

File

plib_can.h

C

```
void PLIB_CAN_FilterConfigure(CAN_MODULE_ID index, CAN_FILTER filter, uint32_t id, CAN_ID_TYPE filterType);
```

Returns

None.

Description

This function configures a CAN message acceptance filter. The ID field of the incoming message must match the filter bits for the CAN module to accept the message. A filter can be a EID type filter, which filters EID messages or a SID filter, which in turn filters SID messages. The filter mask bits (configured using the PLIB_CAN_FilterMaskConfigure function) additionally allow specified message ID bits to be ignored in the filtering process.

Remarks

A CAN message acceptance filter can be configured in Normal operation mode. The filter must be disabled before this is done.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsFilterConfigure in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Configure CAN_ID_2 filter 4 to accept standard ID messages
// with SID 0x100

PLIB_CAN_FilterConfigure(CAN_ID_2, CAN_FILTER4, 0x100, CAN_SID);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN receive filter
id	A value in the range 0x0 to 0x7FF for SID filter type or 0x0 to 0x1FFFFFFF for EID filter type.
filterType	Specifies the type of filter • CAN_EID - Filter is an extended ID message filter • CAN_SID - Filter is an standard ID message filter

Function

void PLIB_CAN_FilterConfigure (CAN_MODULE_ID index, CAN_FILTER filter,
uint32_t id, CAN_ID_TYPE filterType)

PLIB_CAN_FilterDisable Function

Disables a CAN message acceptance filter.

File

plib_can.h

C

```
void PLIB_CAN_FilterDisable(CAN_MODULE_ID index, CAN_FILTER filter);
```

Returns

None.

Description

This function disables a CAN message acceptance filter. At least one filter must be enabled for the CAN module to receive messages. A receive channel associated with a filter will not receive messages unless the filter is enabled. After a filter is disabled, the PLIB_CAN_FilterIsDisabled function should be called to verify that the filter is disabled. A filter must be disabled before it can be configured.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsFilterEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable filter 6 of CAN_ID_2

PLIB_CAN_FilterDisable (CAN_ID_2, CAN_FILTER6);

while(PLIB_CAN_FilterIsDisabled(CAN_ID_1, CAN_FILTER4));
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN Message Acceptance Filter

Function

void PLIB_CAN_FilterDisable(CAN_MODULE_ID index, CAN_FILTER filter)

PLIB_CAN_FilterEnable Function

Enables a CAN message acceptance filter.

File

plib_can.h

C

```
void PLIB_CAN_FilterEnable(CAN_MODULE_ID index, CAN_FILTER filter);
```

Returns

None.

Description

This function enables a CAN message acceptance filter. At least one filter must be enabled for the CAN module to receive messages. A receive channel associated with a filter will not receive messages unless the filter is enabled. After a filter is disabled, the PLIB_CAN_FilterIsDisabled function should be called to verify that the filter is disabled. A filter must be disabled before it can be configured.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsFilterEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable filter 6 of CAN_ID_2

PLIB_CAN_FilterEnable (CAN_ID_2, CAN_FILTER6);


while(!PLIB_CAN_FilterIsDisabled(CAN_ID_1, CAN_FILTER4));
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN message acceptance filter

Function

void PLIB_CAN_FilterEnable(CAN_MODULE_ID index, CAN_FILTER filter)

PLIB_CAN_FilterIsDisabled Function

Returns 'true' if the specified filter is disabled.

File

plib_can.h

C

```
bool PLIB_CAN_FilterIsDisabled(CAN_MODULE_ID index, CAN_FILTER filter);
```

Returns

- true = The filter is disabled
- false = The filter is enabled

Description

Returns 'true' if the specified filter is disabled. This function should be called to check if the filter is disabled before calling the PLIB_CAN_FilterConfigure function and PLIB_CAN_FilterToChannelLink function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsFilterEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable CAN_ID_1 filter 3 and wait until the filter is disabled.
PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER3);

while(PLIB_CAN_FilterIsDisabled(CAN_ID_1, CAN_FILTER3) == false);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN filter

Function

bool PLIB_CAN_FilterIsDisabled(CAN_MODULE_ID index, CAN_FILTER filter)

PLIB_CAN_FilterMaskConfigure Function

Configures a CAN filter mask.

File

plib_can.h

C

```
void PLIB_CAN_FilterMaskConfigure(CAN_MODULE_ID index, CAN_FILTER_MASK mask, uint32_t maskBits, CAN_ID_TYPE
idType, CAN_FILTER_MASK_TYPE mide);
```

Returns

None.

Description

This function configures a CAN filter mask. The mask bits determine which message ID bits are ignored and compared during the filtering process.

Remarks

A mask bit value of zero essentially means that all messages with any ID are accepted. The mode and idType input parameters are still relevant in such a case.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsFilterMaskConfigure in your application to determine whether this feature is available.

Preconditions

The CAN module should be in Configuration mode. This is achieved by using the PLIB_CAN_OperationModeSelect function.

Example

```
// Configure CAN_ID_1 Filter Mask 2 to accept
// extended ID messages in the range 0x4F1DE8 - 0x4F1DEC.
// On analyzing this address range it can be seen that only
// the last two bits of the incoming CAN message should ignored.
// Therefore the mask value should 0x1FFFFFFC.
// This mask will be used with a extended ID filter.
// Set the masking option to filter IDE type.

PLIB_CAN_FilterMaskConfigure(CAN_ID_1, CAN_FILTER_MASK2, 0x1FFFFFFC,
                             CAN_EID, CAN_FILTER_MASK_IDE_TYPE);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
mask	Identifies the desired CAN receive filter mask.
maskBits	A value in the range 0x0 to 0x7FF for SID range or 0x0 to 0x1FFFFFFF for the EID range. Each set bit will mean that the corresponding bit in the filter will be compared to the corresponding bit in the message ID. A clear mask bit means the corresponding bit in the incoming message ID field will be ignored.
idType	Specifies the value range of maskBits parameter. • CAN_EID - Value range of maskBits parameter is 0x0 (ignore all 29 bits of the incoming message ID) to 0x1FFFFFFF (compare all 29 bits of the incoming message ID). • CAN_SID - Value range of maskBits parameter is 0x0 (ignore all 11 bits of the incoming message ID) to 0x7FF (compare all 11 bits of the incoming message ID).
mide	Specifies ID masking options • CAN_FILTER_MASK_IDE_TYPE - Masking will be performed by filter type only. If the filter is set up for SID messages, the mask will decline all incoming EID messages. If the filter is set up for EID messages, the mask will decline all incoming SID messages • CAN_FILTER_MASK_ANY_TYPE - Masking will be performed regardless of the incoming message ID type. The message will be accepted on a Filter and Message SID match or a filter and message SID/EID match.

Function

void PLIB_CAN_FilterMaskConfigure(CAN_MODULE_ID index, CAN_FILTER_MASK mask,
uint32_t maskBits, CAN_ID_TYPE idType, CAN_FILTER_MASK_TYPE mide)

PLIB_CAN_FilterToChannelLink Function

Links a filter to a channel.

File

plib_can.h

C

```
void PLIB_CAN_FilterToChannelLink(CAN_MODULE_ID index, CAN_FILTER filter, CAN_FILTER_MASK mask, CAN_CHANNEL channel);
```

Returns

None.

Description

This function links a filter to a channel. A filter is typically linked to a receive channel. This allows the channel to receive messages accepted by the filter. A filter can also be linked to a transmit channel if the transmit channel is configured for remote request transmit. In this case, a message accepted by the filter will automatically cause the linked transmit channel to transmit CAN messages that are buffered in the channel. Note that a filter should be enabled using the PLIB_CAN_FilterEnable function after the filter has been linked to the desired channel.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsFilterToChannelLink in your application to determine whether this feature is available.

Preconditions

Filter should have been disabled using the PLIB_CAN_FilterDisable function.

Example

```
// Configure CAN_ID_1 filter 3 to accept extended ID messages
// with EID 0x1D400 and link the filter to CAN_ID_1 Channel 5.
// Note the sequence in which the steps are performed.
// Disable the filter and check if its disabled.
// Configure the filter. Link it to the Channel and then
// enable the filter.

PLIB_CAN_FilterDisable(CAN_ID_1, CAN_FILTER3);

while(PLIB_CAN_FilterIsDisabled(CAN_ID_1, CAN_FILTER3) == false);

PLIB_CAN_FilterConfigure(CAN_ID_1, CAN_FILTER3, 0x1D400, CAN_EID);

PLIB_CAN_FilterToChannelLink(CAN_ID_1, CAN_FILTER3, CAN_FILTER_MASK0,
    CAN_CHANNEL5);

PLIB_CAN_FilterEnable(CAN_ID_1, CAN_FILTER3);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module
filter	Identifies the desired CAN Filter
mask	Identifies the mask to be used with this filter
channel	Identifies the channel to be linked to this filter. If a transmit channel is linked, the transmit channel should have its remote transmit request feature enabled.

Function

void PLIB_CAN_FilterToChannelLink(CAN_MODULE_ID index, CAN_FILTER filter,
CAN_FILTER_MASK mask, CAN_CHANNEL channel)

PLIB_CAN_LatestFilterMatchGet Function

Returns the index of the filter that accepted the latest message.

File

plib_can.h

C

```
CAN_FILTER PLIB_CAN_LatestFilterMatchGet(CAN_MODULE_ID index);
```

Returns

Index of the filter that accepted the latest message

Description

This function returns the index of the filter that accepted the latest message.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsLatestFilterMatchGet in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if CAN_ID_2 module Receive Buffer event
// is active and if so check which filter
// accepted the message.

CAN_FILTER filter;

if((PLIB_CAN_ModuleEventGet(CAN_ID_1) & CAN_RX_EVENT) != 0)
{
```

```
    // Check which filter accepted the message

    filter = PLIB_CAN_LatestFilterMatchGet(CAN_ID_1);
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_FILTER PLIB_CAN_LatestFilterMatchGet(CAN_MODULE_ID index)

i) Error State Tracking Functions

PLIB_CAN_ReceiveErrorCountGet Function

Returns the CAN receive error count.

File

plib_can.h

C

```
int PLIB_CAN_ReceiveErrorCountGet(CAN_MODULE_ID index);
```

Returns

Returns the CAN receive error count.

Description

This function returns the CAN receive error count. Refer to the CAN 2.0B specification for more details on the CAN receive error count and its significance.

Remarks

There are multiple bus conditions, which could cause the receive error count to increase. Please refer to the CAN 2.0b specification for details.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsReceiveErrorCount in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if CAN_ID_1 Receive error count is more than 200.
int errCount;

errCount = PLIB_CAN_ReceiveErrorCountGet(CAN_ID_1);
if(errCount > 200)
{
    // This error count is high.
    // Do some diagnostics.
}
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

int PLIB_CAN_ReceiveErrorCountGet(CAN_MODULE_ID index)

PLIB_CAN_ErrorStateGet Function

Returns the CAN error status word.

File

plib_can.h

C

```
CAN_ERROR_STATE PLIB_CAN_ErrorStateGet(CAN_MODULE_ID index);
```

Returns

Returns the CAN_ERROR_STATE word, which can be logically ANDed with the desired CAN_ERROR_STATE member to check whether the CAN module is in a specific error state.

Description

This function returns the CAN error status word. The return word can be logically ANDed with the desired CAN_ERROR_STATE member to check if the CAN module is in a specific error state.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsErrorState in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if CAN_ID_1 is in the Transmit or Receive warning state.
CAN_ERROR_STATE errorState;

errorState = PLIB_CAN_ErrorStateGet(CAN_ID_1);

if((errorState & CAN_TX_RX_WARNING_STATE) != 0)
{
    // CAN_ID_1 is in either  Transmit or Receive warning state.
}
// Check if CAN_ID_2 is in the Receive Bus Passive or Transmit Bus passive
// state.
errorState = PLIB_CAN_ErrorStateGet(CAN_ID_2);

if((errorState & (CAN_TX_BUS_PASSIVE_STATE | CAN_RX_BUS_PASSIVE_STATE)) != 0)
{
    // This means that the CAN module is in Transmit Bus Passive
    // or Receive Bus Passive state.
 }
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

CAN_ERROR_STATE PLIB_CAN_ErrorStateGet(CAN_MODULE_ID index)

j) Information Functions

PLIB_CAN_TotalChannelsGet Function

Returns the total number of CAN channels per CAN module.

File

plib_can.h

C

```
char PLIB_CAN_TotalChannelsGet(CAN_MODULE_ID index);
```

Returns

The total number of CAN channels per CAN module.

Description

This function returns the total number of CAN channels per CAN module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleInfo in your application to determine whether this feature is available.

Preconditions

None.

Example

```
char totalChannels;

totalChannels = PLIB_CAN_TotalChannelsGet(CAN_ID_1);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

char PLIB_CAN_TotalChannelsGet(CAN_MODULE_ID index)

PLIB_CAN_TotalFiltersGet Function

Returns the total number of CAN Filters per CAN module.

File

plib_can.h

C

```
char PLIB_CAN_TotalFiltersGet(CAN_MODULE_ID index);
```

Returns

The total number of CAN Filters per CAN module.

Description

This function returns the total number of CAN filters per CAN module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleInfo in your application to determine whether this feature is available.

Preconditions

None.

Example

```
char totalFilters;

totalFilters = PLIB_CAN_TotalFiltersGet(CAN_ID_1);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

char PLIB_CAN_TotalFiltersGet(CAN_MODULE_ID index)

PLIB_CAN_TotalMasksGet Function

Returns the total number of CAN masks per CAN module.

File

plib_can.h

C

```
char PLIB_CAN_TotalMasksGet(CAN_MODULE_ID index);
```

Returns

The total number of CAN Masks per CAN module.

Description

This function returns the total number of CAN masks per CAN module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CAN_ExistsModuleInfo in your application to determine whether this feature is available.

Preconditions

None.

Example

```
char totalMasks;

totalMasks = PLIB_CAN_TotalMasksGet(CAN_ID_1);
```

Parameters

Parameters	Description
index	Identifies the desired CAN module

Function

char PLIB_CAN_TotalMasksGet(CAN_MODULE_ID index)

k) Feature Existence Functions

PLIB_CAN_ExistsActiveStatus Function

Identifies whether the ActiveStatus feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsActiveStatus(CAN_MODULE_ID index);
```

Returns

- true = The ActiveStatus feature is supported on the device
- false = The ActiveStatus feature is not supported on the device

Description

This function identifies whether the ActiveStatus feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_IsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsActiveStatus(CAN_MODULE_ID index)

PLIB_CAN_ExistsAllChannelEvents Function

Identifies whether the AllChannelEvents feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsAllChannelEvents(CAN_MODULE_ID index);
```

Returns

- true = The AllChannelEvents feature is supported on the device
- false = The AllChannelEvents feature is not supported on the device

Description

This function identifies whether the AllChannelEvents feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_AllChannelEventsGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsAllChannelEvents(CAN_MODULE_ID index)

PLIB_CAN_ExistsAllChannelOverflow Function

Identifies whether the AllChannelOverflow feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsAllChannelOverflow(CAN_MODULE_ID index);
```

Returns

- true = The AllChannelOverflow feature is supported on the device
- false = The AllChannelOverflow feature is not supported on the device

Description

This function identifies whether the AllChannelOverflow feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_AllChannelOverflowStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsAllChannelOverflow(CAN_MODULE_ID index)

PLIB_CAN_ExistsBusActivityWakeup Function

Identifies whether the BusActivityWakeup feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsBusActivityWakeup(CAN_MODULE_ID index);
```

Returns

- true = The BusActivityWakeup feature is supported on the device
- false = The BusActivityWakeup feature is not supported on the device

Description

This function identifies whether the BusActivityWakeup feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_BusActivityWakeupEnable
- PLIB_CAN_BusActivityWakeupDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsBusActivityWakeup(CAN_MODULE_ID index)

PLIB_CAN_ExistsBusLine3TimesSampling Function

Identifies whether the BusLine3TimesSampling feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsBusLine3TimesSampling(CAN_MODULE_ID index);
```

Returns

- true = The BusLine3TimesSampling feature is supported on the device
- false = The BusLine3TimesSampling feature is not supported on the device

Description

This function identifies whether the BusLine3TimesSampling feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_BusLine3TimesSamplingEnable
- PLIB_CAN_BusLine3TimesSamplingDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsBusLine3TimesSampling(CAN_MODULE_ID index)

PLIB_CAN_ExistsChannelEvent Function

Identifies whether the ChannelEventGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsChannelEvent(CAN_MODULE_ID index);
```

Returns

- true = The ChannelEventGet feature is supported on the device
- false = The ChannelEventGet feature is not supported on the device

Description

This function identifies whether the ChannelEventGet feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_ChannelEventGet
- PLIB_CAN_ChannelEventClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsChannelEvent(CAN_MODULE_ID index)

PLIB_CAN_ExistsChannelEventEnable Function

Identifies whether the ChannelEventEnable feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsChannelEventEnable(CAN_MODULE_ID index);
```

Returns

- true = The ChannelEventEnable feature is supported on the device

- false = The ChannelEventEnable feature is not supported on the device

Description

This function identifies whether the ChannelEventEnable feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_ChannelEventEnable
- PLIB_CAN_ChannelEventDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsChannelEventEnable(CAN_MODULE_ID index)

PLIB_CAN_ExistsChannelForReceiveSet Function

Identifies whether the ChannelForReceiveSet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsChannelForReceiveSet(CAN_MODULE_ID index);
```

Returns

- true = The ChannelForReceiveSet feature is supported on the device
- false = The ChannelForReceiveSet feature is not supported on the device

Description

This function identifies whether the ChannelForReceiveSet feature is available on the CAN module. When this function returns true, this function is are supported on the device:

- PLIB_CAN_ChannelForReceiveSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsChannelForReceiveSet(CAN_MODULE_ID index)

PLIB_CAN_ExistsChannelForTransmitSet Function

Identifies whether the ChannelForTransmitSet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsChannelForTransmitSet(CAN_MODULE_ID index);
```

Returns

- true = The ChannelForTransmitSet feature is supported on the device
- false = The ChannelForTransmitSet feature is not supported on the device

Description

This function identifies whether the ChannelForTransmitSet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_ChannelForTransmitSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsChannelForTransmitSet(CAN_MODULE_ID index)

PLIB_CAN_ExistsChannelReset Function

Identifies whether the ChannelReset feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsChannelReset(CAN_MODULE_ID index);
```

Returns

- true = The ChannelReset feature is supported on the device
- false = The ChannelReset feature is not supported on the device

Description

This function identifies whether the ChannelReset feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_ChannelReset
- PLIB_CAN_ChannelResetIsComplete

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsChannelReset(CAN_MODULE_ID index)

PLIB_CAN_ExistsChannelUpdate Function

Identifies whether the ChannelUpdate feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsChannelUpdate(CAN_MODULE_ID index);
```

Returns

- true = The ChannelUpdate feature is supported on the device
- false = The ChannelUpdate feature is not supported on the device

Description

This function identifies whether the ChannelUpdate feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_ChannelUpdate

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsChannelUpdate(CAN_MODULE_ID index)

PLIB_CAN_ExistsDeviceNet Function

Identifies whether the DeviceNet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsDeviceNet(CAN_MODULE_ID index);
```

Returns

- true = The DeviceNet feature is supported on the device
- false = The DeviceNet feature is not supported on the device

Description

This function identifies whether the DeviceNet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_DeviceNetConfigure

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsDeviceNet(CAN_MODULE_ID index)

PLIB_CAN_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsEnableControl(CAN_MODULE_ID index);
```

Returns

- true = The EnableControl feature is supported on the device
- false = The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_Enable
- PLIB_CAN_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsEnableControl(CAN_MODULE_ID index)

PLIB_CAN_ExistsErrorState Function

Identifies whether the ErrorStateGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsErrorState(CAN_MODULE_ID index);
```

Returns

- true = The ErrorStateGet feature is supported on the device
- false = The ErrorStateGet feature is not supported on the device

Description

This function identifies whether the ErrorStateGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_ErrorStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsErrorState(CAN_MODULE_ID index)

PLIB_CAN_ExistsFilterConfigure Function

Identifies whether the FilterConfigure feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsFilterConfigure(CAN_MODULE_ID index);
```

Returns

- true = The FilterConfigure feature is supported on the device
- false = The FilterConfigure feature is not supported on the device

Description

This function identifies whether the FilterConfigure feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_FilterConfigure

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsFilterConfigure(CAN_MODULE_ID index)

PLIB_CAN_ExistsFilterEnable Function

Identifies whether the FilterEnable feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsFilterEnable(CAN_MODULE_ID index);
```

Returns

- true = The FilterEnable feature is supported on the device
- false = The FilterEnable feature is not supported on the device

Description

This function identifies whether the FilterEnable feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_FilterEnable
- PLIB_CAN_FilterDisable
- PLIB_CAN_FilterIsDisabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsFilterEnable(CAN_MODULE_ID index)

PLIB_CAN_ExistsFilterMaskConfigure Function

Identifies whether the FilterMaskConfigure feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsFilterMaskConfigure(CAN_MODULE_ID index);
```

Returns

- true = The FilterMaskConfigure feature is supported on the device
- false = The FilterMaskConfigure feature is not supported on the device

Description

This function identifies whether the FilterMaskConfigure feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_FilterMaskConfigure

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsFilterMaskConfigure(CAN_MODULE_ID index)

PLIB_CAN_ExistsFilterToChannelLink Function

Identifies whether the FilterToChannelLink feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsFilterToChannelLink(CAN_MODULE_ID index);
```

Returns

- true = The FilterToChannelLink feature is supported on the device
- false = The FilterToChannelLink feature is not supported on the device

Description

This function identifies whether the FilterToChannelLink feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_FilterToChannelLink

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsFilterToChannelLink(CAN_MODULE_ID index)

PLIB_CAN_ExistsLatestFilterMatchGet Function

Identifies whether the LatestFilterMatchGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsLatestFilterMatchGet(CAN_MODULE_ID index);
```

Returns

- true = The LatestFilterMatchGet feature is supported on the device
- false = The LatestFilterMatchGet feature is not supported on the device

Description

This function identifies whether the LatestFilterMatchGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_LatestFilterMatchGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsLatestFilterMatchGet(CAN_MODULE_ID index)

PLIB_CAN_ExistsMemoryBufferAssign Function

Identifies whether the MemoryBufferAssign feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsMemoryBufferAssign(CAN_MODULE_ID index);
```

Returns

- true = The MemoryBufferAssign feature is supported on the device
- false = The MemoryBufferAssign feature is not supported on the device

Description

This function identifies whether the MemoryBufferAssign feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_MemoryBufferAssign

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsMemoryBufferAssign(CAN_MODULE_ID index)

PLIB_CAN_ExistsModuleEventClear Function

Identifies whether the ModuleEvents feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsModuleEventClear(CAN_MODULE_ID index);
```

Returns

- true = The ModuleEvents feature is supported on the device
- false = The ModuleEvents feature is not supported on the device

Description

This function identifies whether the ModuleEvents feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_ModuleEventClear
- PLIB_CAN_ModuleEventGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsModuleEventClear(CAN_MODULE_ID index)

PLIB_CAN_ExistsModuleEventEnable Function

Identifies whether the ModuleEventEnable feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsModuleEventEnable(CAN_MODULE_ID index);
```

Returns

- true = The ModuleEventEnable feature is supported on the device
- false = The ModuleEventEnable feature is not supported on the device

Description

This function identifies whether the ModuleEventEnable feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_ModuleEventEnable
- PLIB_CAN_ModuleEventDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsModuleEventEnable(CAN_MODULE_ID index)

PLIB_CAN_ExistsModuleInfo Function

Identifies whether the ModuleInformation feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsModuleInfo(CAN_MODULE_ID index);
```

Returns

- true = The ModuleInformation feature is supported on the device
- false = The ModuleInformation feature is not supported on the device

Description

This function identifies whether the ModuleInformation feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_TotalChannelsGet
- PLIB_CAN_TotalFiltersGet
- PLIB_CAN_TotalMasksGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsModuleInfo(CAN_MODULE_ID index)

PLIB_CAN_ExistsOperationModeRead Function

Identifies whether the OperationModeRead feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsOperationModeRead(CAN_MODULE_ID index);
```

Returns

- true = The OperationModeRead feature is supported on the device
- false = The OperationModeRead feature is not supported on the device

Description

This function identifies whether the OperationModeRead feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_OperationModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsOperationModeRead(CAN_MODULE_ID index)

PLIB_CAN_ExistsOperationModeWrite Function

Identifies whether the OperationModeSet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsOperationModeWrite(CAN_MODULE_ID index);
```

Returns

- true = The OperationModeSet feature is supported on the device
- false = The OperationModeSet feature is not supported on the device

Description

This function identifies whether the OperationModeSet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_OperationModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsOperationModeWrite(CAN_MODULE_ID index)

PLIB_CAN_ExistsPendingEventsGet Function

Identifies whether the PendingEventsGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsPendingEventsGet(CAN_MODULE_ID index);
```

Returns

- true = The PendingEventsGet feature is supported on the device
- false = The PendingEventsGet feature is not supported on the device

Description

This function identifies whether the PendingEventsGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_PendingEventsGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsPendingEventsGet(CAN_MODULE_ID index)

PLIB_CAN_ExistsPendingTransmissionsAbort Function

Identifies whether the PendingTransmissionsAbort feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsPendingTransmissionsAbort(CAN_MODULE_ID index);
```

Returns

- true = The PendingTransmissionsAbort feature is supported on the device
- false = The PendingTransmissionsAbort feature is not supported on the device

Description

This function identifies whether the PendingTransmissionsAbort feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_PendingTransmissionsAbort

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsPendingTransmissionsAbort(CAN_MODULE_ID index)

PLIB_CAN_ExistsReceivedMessageGet Function

Identifies whether the ReceivedMessageGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsReceivedMessageGet(CAN_MODULE_ID index);
```

Returns

- true = The ReceivedMessageGet feature is supported on the device
- false = The ReceivedMessageGet feature is not supported on the device

Description

This function identifies whether the ReceivedMessageGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_ReceivedMessageGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsReceivedMessageGet(CAN_MODULE_ID index)

PLIB_CAN_ExistsReceiveErrorCount Function

Identifies whether the ReceiveErrorCount feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsReceiveErrorCount(CAN_MODULE_ID index);
```

Returns

- true = The ReceiveErrorCount feature is supported on the device
- false = The ReceiveErrorCount feature is not supported on the device

Description

This function identifies whether the ReceiveErrorCount feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_ReceiveErrorCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsReceiveErrorCount(CAN_MODULE_ID index)

PLIB_CAN_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsStopInIdle(CAN_MODULE_ID index);
```

Returns

- true = The StopInIdle feature is supported on the device
- false = The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_StopInIdleEnable
- PLIB_CAN_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsStopInIdle(CAN_MODULE_ID index)

PLIB_CAN_ExistsTimeStampEnable Function

Identifies whether the TimeStampEnable feature exists on the CAN module

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTimeStampEnable(CAN_MODULE_ID index);
```

Returns

- true = The TimeStampEnable feature is supported on the device
- false = The TimeStampEnable feature is not supported on the device

Description

This function identifies whether the TimeStampEnable feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_TimeStampEnable
- PLIB_CAN_TimeStampDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTimeStampEnable(CAN_MODULE_ID index)

PLIB_CAN_ExistsTimeStampValue Function

Identifies whether the TimeStampValue feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTimeStampValue(CAN_MODULE_ID index);
```

Returns

- true = The TimeStampValue feature is supported on the device
- false = The TimeStampValue feature is not supported on the device

Description

This function identifies whether the TimeStampValue feature is available on the CAN module. When this function returns true, these functions are supported on the device:

- PLIB_CAN_TimeStampValueSet
- PLIB_CAN_TimeStampValueGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTimeStampValue(CAN_MODULE_ID index)

PLIB_CAN_ExistsTransmissionIsAborted Function

Identifies whether the TransmissionAbortStatus feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTransmissionIsAborted(CAN_MODULE_ID index);
```

Returns

- true = The TransmissionAbortStatus feature is supported on the device
- false = The TransmissionAbortStatus feature is not supported on the device

Description

This function identifies whether the TransmissionAbortStatus feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_TransmissionIsAborted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTransmissionIsAborted(CAN_MODULE_ID index)

PLIB_CAN_ExistsTransmitBufferGet Function

Identifies whether the TransmitBufferGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTransmitBufferGet(CAN_MODULE_ID index);
```

Returns

- true = The TransmitBufferGet feature is supported on the device
- false = The TransmitBufferGet feature is not supported on the device

Description

This function identifies whether the TransmitBufferGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_TransmitBufferGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTransmitBufferGet(CAN_MODULE_ID index)

PLIB_CAN_ExistsTransmitChannelFlush Function

Identifies whether the TransmitChannelFlush feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTransmitChannelFlush(CAN_MODULE_ID index);
```

Returns

- true = The TransmitChannelFlush feature is supported on the device
- false = The TransmitChannelFlush feature is not supported on the device

Description

This function identifies whether the TransmitChannelFlush feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_TransmitChannelFlush

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTransmitChannelFlush(CAN_MODULE_ID index)

PLIB_CAN_ExistsTransmitChannelStatus Function

Identifies whether the TransmitChannelStatus feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTransmitChannelStatus(CAN_MODULE_ID index);
```

Returns

- true = The TransmitChannelStatus feature is supported on the device
- false = The TransmitChannelStatus feature is not supported on the device

Description

This function identifies whether the TransmitChannelStatus feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_TransmitChannelStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTransmitChannelStatus(CAN_MODULE_ID index)

PLIB_CAN_ExistsTransmitErrorCountGet Function

Identifies whether the TransmitErrorCountGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsTransmitErrorCountGet(CAN_MODULE_ID index);
```

Returns

- true = The TransmitErrorCountGet feature is supported on the device
- false = The TransmitErrorCountGet feature is not supported on the device

Description

This function identifies whether the TransmitErrorCountGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_TransmitErrorCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsTransmitErrorCountGet(CAN_MODULE_ID index)

PLIB_CAN_ExistsBaudRateGet Function

Identifies whether the BaudRateGet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsBaudRateGet(CAN_MODULE_ID index);
```

Returns

- true = The BaudRateGet feature is supported on the device.
- false = The BaudRateGet feature is not supported on the device.

Description

This function identifies whether the BaudRateGet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_BaudRateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsBaudRateGet(CAN_MODULE_ID index)

PLIB_CAN_ExistsBaudRatePrescaleSetup Function

Identifies whether the BaudRatePrescaleSetup feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsBaudRatePrescaleSetup(CAN_MODULE_ID index);
```

Returns

- true = The BaudRatePrescaleSetup feature is supported on the device.
- false = The BaudRatePrescaleSetup feature is not supported on the device.

Description

This function identifies whether the BaudRatePrescaleSetup feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_BaudRatePrescaleSetup

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsBaudRatePrescaleSetup(CAN_MODULE_ID index)

PLIB_CAN_ExistsBitSamplePhaseSet Function

Identifies whether the BitSamplePhaseSet feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsBitSamplePhaseSet(CAN_MODULE_ID index);
```

Returns

- true = The BitSamplePhaseSet feature is supported on the device.
- false = The BitSamplePhaseSet feature is not supported on the device.

Description

This function identifies whether the BitSamplePhaseSet feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_BitSamplePhaseSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsBitSamplePhaseSet(CAN_MODULE_ID index)

PLIB_CAN_ExistsPrecalculatedBitRateSetup Function

Identifies whether the PrecalculatedBitRateSetup feature exists on the CAN module.

File

plib_can.h

C

```
bool PLIB_CAN_ExistsPrecalculatedBitRateSetup(CAN_MODULE_ID index);
```

Returns

- true = The PrecalculatedBitRateSetup feature is supported on the device.
- false = The PrecalculatedBitRateSetup feature is not supported on the device.

Description

This function identifies whether the PrecalculatedBitRateSetup feature is available on the CAN module. When this function returns true, this function is supported on the device:

- PLIB_CAN_PrecalculatedBitRateSetup

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CAN_ExistsPrecalculatedBitRateSetup(CAN_MODULE_ID index)

I) Data Types and Constants

CAN_CHANNEL Enumeration

Identifies the supported CAN Channels.

File

plib_can_help.h

C

```
typedef enum {
  CAN_CHANNEL0,
  CAN_CHANNEL1,
  CAN_CHANNEL2,
  CAN_CHANNEL3,
  CAN_CHANNEL4,
  CAN_CHANNEL5,
  CAN_CHANNEL6,
  CAN_CHANNEL7,
  CAN_CHANNEL8,
  CAN_CHANNEL9,
  CAN_CHANNEL10,
```

```
  CAN_CHANNEL11,
  CAN_CHANNEL12,
  CAN_CHANNEL13,
  CAN_CHANNEL14,
  CAN_CHANNEL15,
  CAN_CHANNEL16,
  CAN_CHANNEL17,
  CAN_CHANNEL18,
  CAN_CHANNEL19,
  CAN_CHANNEL20,
  CAN_CHANNEL21,
  CAN_CHANNEL22,
  CAN_CHANNEL23,
  CAN_CHANNEL24,
  CAN_CHANNEL25,
  CAN_CHANNEL26,
  CAN_CHANNEL27,
  CAN_CHANNEL28,
  CAN_CHANNEL29,
  CAN_CHANNEL30,
  CAN_CHANNEL31
} CAN_CHANNEL;
```

Members

Members	Description
CAN_CHANNEL0	Channel 0 ID
CAN_CHANNEL1	Channel 1 ID
CAN_CHANNEL2	Channel 2 ID
CAN_CHANNEL3	Channel 3 ID
CAN_CHANNEL4	Channel 4 ID
CAN_CHANNEL5	Channel 5 ID
CAN_CHANNEL6	Channel 6 ID
CAN_CHANNEL7	Channel 7 ID
CAN_CHANNEL8	Channel 8 ID
CAN_CHANNEL9	Channel 9 ID
CAN_CHANNEL10	Channel 10 ID
CAN_CHANNEL11	Channel 11 ID
CAN_CHANNEL12	Channel 12 ID
CAN_CHANNEL13	Channel 13 ID
CAN_CHANNEL14	Channel 14 ID
CAN_CHANNEL15	Channel 15 ID
CAN_CHANNEL16	Channel 16 ID
CAN_CHANNEL17	Channel 17 ID
CAN_CHANNEL18	Channel 18 ID
CAN_CHANNEL19	Channel 19 ID
CAN_CHANNEL20	Channel 20 ID
CAN_CHANNEL21	Channel 21 ID
CAN_CHANNEL22	Channel 22 ID
CAN_CHANNEL23	Channel 23 ID
CAN_CHANNEL24	Channel 24 ID
CAN_CHANNEL25	Channel 25 ID
CAN_CHANNEL26	Channel 26 ID
CAN_CHANNEL27	Channel 27 ID
CAN_CHANNEL28	Channel 28 ID
CAN_CHANNEL29	Channel 29 ID
CAN_CHANNEL30	Channel 30 ID
CAN_CHANNEL31	Channel 31 ID

Description

CAN Channel

This enumeration identifies the available CAN channels.

CAN_CHANNEL_EVENT Enumeration

Identifies all Layer 3 interrupts.

File

plib_can_help.h

C

```
typedef enum {
  CAN_RX_CHANNEL_NOT_EMPTY,
  CAN_RX_CHANNEL_HALF_FULL,
  CAN_RX_CHANNEL_FULL,
  CAN_RX_CHANNEL_OVERFLOW,
  CAN_RX_CHANNEL_ANY_EVENT,
  CAN_TX_CHANNEL_EMPTY,
  CAN_TX_CHANNEL_HALF_EMPTY,
  CAN_TX_CHANNEL_NOT_FULL,
  CAN_TX_CHANNEL_ANY_EVENT
} CAN_CHANNEL_EVENT;
```

Members

Members	Description
CAN_RX_CHANNEL_NOT_EMPTY	CAN Receive Channel Not Empty Event Mask
CAN_RX_CHANNEL_HALF_FULL	CAN Receive Channel Half Full Event Mask
CAN_RX_CHANNEL_FULL	CAN Receive Channel Full Event Mask
CAN_RX_CHANNEL_OVERFLOW	CAN Receive Channel Overflow Event Mask
CAN_RX_CHANNEL_ANY_EVENT	CAN Receive Channel Any Event Mask
CAN_TX_CHANNEL_EMPTY	CAN Transmit Channel Empty Event Mask
CAN_TX_CHANNEL_HALF_EMPTY	CAN Transmit Channel Half Empty Event Mask
CAN_TX_CHANNEL_NOT_FULL	CAN Transmit Channel Not Full Event Mask
CAN_TX_CHANNEL_ANY_EVENT	CAN Transmit Channel Any Event Mask

Description

CAN Layer 3 Interrupts

This enumerates all the leading interrupts generated due to CAN transmit and receive events. This enumeration can be used to enable or disable channel events and as a mask to check if a channel event is active.

A single or a combination of the interrupts can be logically ORed to specify the interrupt(s) to be enabled disabled or events to check.

This enumeration is used by PLIB_CAN_ChannelEventEnable, PLIB_CAN_ChannelEventDisable, PLIB_CAN_ChannelEventClear, and PLIB_CAN_ChannelEventGet functions.

CAN_CHANNEL_MASK Enumeration

Lists the series of useful masks.

File

plib_can_help.h

C

```
typedef enum {
  CAN_CHANNEL0_MASK,
  CAN_CHANNEL1_MASK,
  CAN_CHANNEL2_MASK,
  CAN_CHANNEL3_MASK,
  CAN_CHANNEL4_MASK,
  CAN_CHANNEL5_MASK,
  CAN_CHANNEL6_MASK,
  CAN_CHANNEL7_MASK,
  CAN_CHANNEL8_MASK,
  CAN_CHANNEL9_MASK,
  CAN_CHANNEL10_MASK,
  CAN_CHANNEL11_MASK,
  CAN_CHANNEL12_MASK,
  CAN_CHANNEL13_MASK,
  CAN_CHANNEL14_MASK,
```

```
    CAN_CHANNEL15_MASK,
    CAN_CHANNEL16_MASK,
    CAN_CHANNEL17_MASK,
    CAN_CHANNEL18_MASK,
    CAN_CHANNEL19_MASK,
    CAN_CHANNEL20_MASK,
    CAN_CHANNEL21_MASK,
    CAN_CHANNEL22_MASK,
    CAN_CHANNEL23_MASK,
    CAN_CHANNEL24_MASK,
    CAN_CHANNEL25_MASK,
    CAN_CHANNEL26_MASK,
    CAN_CHANNEL27_MASK,
    CAN_CHANNEL28_MASK,
    CAN_CHANNEL29_MASK,
    CAN_CHANNEL30_MASK,
    CAN_CHANNEL31_MASK,
    CAN_ANYCHANNEL_MASK
} CAN_CHANNEL_MASK;
```

Members

Members	Description
CAN_CHANNEL0_MASK	Channel 0 Mask
CAN_CHANNEL1_MASK	Channel 1 Mask
CAN_CHANNEL2_MASK	Channel 2 Mask
CAN_CHANNEL3_MASK	Channel 3 Mask
CAN_CHANNEL4_MASK	Channel 4 Mask
CAN_CHANNEL5_MASK	Channel 5 Mask
CAN_CHANNEL6_MASK	Channel 6 Mask
CAN_CHANNEL7_MASK	Channel 7 Mask
CAN_CHANNEL8_MASK	Channel 8 Mask
CAN_CHANNEL9_MASK	Channel 9 Mask
CAN_CHANNEL10_MASK	Channel 10 Mask
CAN_CHANNEL11_MASK	Channel 11 Mask
CAN_CHANNEL12_MASK	Channel 12 Mask
CAN_CHANNEL13_MASK	Channel 13 Mask
CAN_CHANNEL14_MASK	Channel 14 Mask
CAN_CHANNEL15_MASK	Channel 15 Mask
CAN_CHANNEL16_MASK	Channel 16 Mask
CAN_CHANNEL17_MASK	Channel 17 Mask
CAN_CHANNEL18_MASK	Channel 18 Mask
CAN_CHANNEL19_MASK	Channel 19 Mask
CAN_CHANNEL20_MASK	Channel 20 Mask
CAN_CHANNEL21_MASK	Channel 21 Mask
CAN_CHANNEL22_MASK	Channel 22 Mask
CAN_CHANNEL23_MASK	Channel 23 Mask
CAN_CHANNEL24_MASK	Channel 24 Mask
CAN_CHANNEL25_MASK	Channel 25 Mask
CAN_CHANNEL26_MASK	Channel 26 Mask
CAN_CHANNEL27_MASK	Channel 27 Mask
CAN_CHANNEL28_MASK	Channel 28 Mask
CAN_CHANNEL29_MASK	Channel 29 Mask
CAN_CHANNEL30_MASK	Channel 30 Mask
CAN_CHANNEL31_MASK	Channel 31 Mask
CAN_ANYCHANNEL_MASK	Channel any channel Mask

Description

CAN Channel Masks

This enumeration identifies a series of useful masks. Each mask represents a CAN channel. These masks are used with the PLIB_CAN_AllChannelEventsGet and PLIB_CAN_AllChannelOverflowStatusGet functions. The value returned by these functions can be logically ANDed with any of these masks to check if an event or overflow event for that channel is active.

CAN_DNET_FILTER_SIZE Enumeration

Specifies the size of the DeviceNet filter.

File

plib_can_help.h

C

```
typedef enum {
  CAN_DNET_FILTER_SIZE_18_BIT,
  CAN_DNET_FILTER_SIZE_17_BIT,
  CAN_DNET_FILTER_SIZE_16_BIT,
  CAN_DNET_FILTER_SIZE_15_BIT,
  CAN_DNET_FILTER_SIZE_14_BIT,
  CAN_DNET_FILTER_SIZE_13_BIT,
  CAN_DNET_FILTER_SIZE_12_BIT,
  CAN_DNET_FILTER_SIZE_11_BIT,
  CAN_DNET_FILTER_SIZE_10_BIT,
  CAN_DNET_FILTER_SIZE_9_BIT,
  CAN_DNET_FILTER_SIZE_8_BIT,
  CAN_DNET_FILTER_SIZE_7_BIT,
  CAN_DNET_FILTER_SIZE_6_BIT,
  CAN_DNET_FILTER_SIZE_5_BIT,
  CAN_DNET_FILTER_SIZE_4_BIT,
  CAN_DNET_FILTER_SIZE_3_BIT,
  CAN_DNET_FILTER_SIZE_2_BIT,
  CAN_DNET_FILTER_SIZE_1_BIT,
  CAN_DNET_FILTER_DISABLE
} CAN_DNET_FILTER_SIZE;
```

Members

Members	Description
CAN_DNET_FILTER_SIZE_18_BIT	Compare up to data byte 2 bit 6 with filter mask bit 17
CAN_DNET_FILTER_SIZE_17_BIT	Compare up to data byte 2 bit 7 with filter mask bit 16
CAN_DNET_FILTER_SIZE_16_BIT	Compare up to data byte 1 bit 0 with filter mask bit 15
CAN_DNET_FILTER_SIZE_15_BIT	Compare up to data byte 1 bit 1 with filter mask bit 14
CAN_DNET_FILTER_SIZE_14_BIT	Compare up to data byte 1 bit 2 with filter mask bit 13
CAN_DNET_FILTER_SIZE_13_BIT	Compare up to data byte 1 bit 3 with filter mask bit 12
CAN_DNET_FILTER_SIZE_12_BIT	Compare up to data byte 1 bit 4 with filter mask bit 11
CAN_DNET_FILTER_SIZE_11_BIT	Compare up to data byte 1 bit 5 with filter mask bit 10
CAN_DNET_FILTER_SIZE_10_BIT	Compare up to data byte 1 bit 6 with filter mask bit 9
CAN_DNET_FILTER_SIZE_9_BIT	Compare up to data byte 1 bit 7 with filter mask bit 8
CAN_DNET_FILTER_SIZE_8_BIT	Compare up to data byte 0 bit 0 with filter mask bit 7
CAN_DNET_FILTER_SIZE_7_BIT	Compare up to data byte 0 bit 1 with filter mask bit 6
CAN_DNET_FILTER_SIZE_6_BIT	Compare up to data byte 0 bit 2 with filter mask bit 5
CAN_DNET_FILTER_SIZE_5_BIT	Compare up to data byte 0 bit 3 with filter mask bit 4
CAN_DNET_FILTER_SIZE_4_BIT	Compare up to data byte 0 bit 4 with filter mask bit 3
CAN_DNET_FILTER_SIZE_3_BIT	Compare up to data byte 0 bit 5 with filter mask bit 2
CAN_DNET_FILTER_SIZE_2_BIT	Compare up to data byte 0 bit 6 with filter mask bit 1
CAN_DNET_FILTER_SIZE_1_BIT	Compare up to data byte 0 bit 7 with filter mask bit 0
CAN_DNET_FILTER_DISABLE	Do not compare data bytes, Device Net Filtering is disabled

Description

CAN DeviceNet filter bit numbers.

This enumeration identifies the size of the DeviceNet filter in bits. If the size of the DeviceNet filter is "n", the "n" most significant bits of the data payload are compared with the EID field of enabled filters.

CAN_ERROR_STATE Enumeration

Specifies the CAN module error states.

File

plib_can_help.h

C

```
typedef enum {
  CAN_TX_RX_WARNING_STATE,
  CAN_RX_WARNING_STATE,
  CAN_TX_WARNING_STATE,
  CAN_RX_BUS_PASSIVE_STATE,
  CAN_TX_BUS_PASSIVE_STATE,
  CAN_TX_BUS_OFF_STATE
} CAN_ERROR_STATE;
```

Members

Members	Description
CAN_TX_RX_WARNING_STATE	CAN Module is in a Transmit or Receive warning state.
CAN_RX_WARNING_STATE	CAN Module is in a Receive warning state.
CAN_TX_WARNING_STATE	CAN Module is in a Transmit warning state.
CAN_RX_BUS_PASSIVE_STATE	CAN Receive is in a Bus Passive state.
CAN_TX_BUS_PASSIVE_STATE	CAN Transmit is in a Bus Passive state.
CAN_TX_BUS_OFF_STATE	CAN Transmit is in Bus Off state.

Description

CAN Error States

This enumeration identifies all of the CAN module error states events. The CAN module enters or exits an error state as the transmit and receive error counter values change. Refer to the CAN 2.0B specification for more details on the error states.

This enumeration is used with the PLIB_CAN_ErrorStateGet function.

CAN_FILTER Enumeration

CAN event code returned by the CAN module.

File

plib_can_help.h

C

```
typedef enum {
  CAN_CHANNEL0_EVENT,
  CAN_CHANNEL1_EVENT,
  CAN_CHANNEL2_EVENT,
  CAN_CHANNEL3_EVENT,
  CAN_CHANNEL4_EVENT,
  CAN_CHANNEL5_EVENT,
  CAN_CHANNEL6_EVENT,
  CAN_CHANNEL7_EVENT,
  CAN_CHANNEL8_EVENT,
  CAN_CHANNEL9_EVENT,
  CAN_CHANNEL10_EVENT,
  CAN_CHANNEL11_EVENT,
  CAN_CHANNEL12_EVENT,
  CAN_CHANNEL13_EVENT,
  CAN_CHANNEL14_EVENT,
  CAN_CHANNEL15_EVENT,
  CAN_CHANNEL16_EVENT,
  CAN_CHANNEL17_EVENT,
  CAN_CHANNEL18_EVENT,
  CAN_CHANNEL19_EVENT,
  CAN_CHANNEL20_EVENT,
  CAN_CHANNEL21_EVENT,
  CAN_CHANNEL22_EVENT,
  CAN_CHANNEL23_EVENT,
  CAN_CHANNEL24_EVENT,
  CAN_CHANNEL25_EVENT,
  CAN_CHANNEL26_EVENT,
  CAN_CHANNEL27_EVENT,
  CAN_CHANNEL28_EVENT,
```

```
    CAN_CHANNEL29_EVENT,
    CAN_CHANNEL30_EVENT,
    CAN_CHANNEL31_EVENT,
    CAN_NO_EVENT,
    CAN_ERROR_EVENT,
    CAN_WAKEUP_EVENT,
    CAN_RX_CHANNEL_OVERFLOW_EVENT,
    CAN_ADDRESS_ERROR_EVENT,
    CAN_BUS_BANDWIDTH_ERROR,
    CAN_TIMESTAMP_TIMER_EVENT,
    CAN_MODE_CHANGE_EVENT,
    CAN_FILTER0,
    CAN_FILTER1,
    CAN_FILTER2,
    CAN_FILTER3,
    CAN_FILTER4,
    CAN_FILTER5,
    CAN_FILTER6,
    CAN_FILTER7,
    CAN_FILTER8,
    CAN_FILTER9,
    CAN_FILTER10,
    CAN_FILTER11,
    CAN_FILTER12,
    CAN_FILTER13,
    CAN_FILTER14,
    CAN_FILTER15,
    CAN_FILTER16,
    CAN_FILTER17,
    CAN_FILTER18,
    CAN_FILTER19,
    CAN_FILTER20,
    CAN_FILTER21,
    CAN_FILTER22,
    CAN_FILTER23,
    CAN_FILTER24,
    CAN_FILTER25,
    CAN_FILTER26,
    CAN_FILTER27,
    CAN_FILTER28,
    CAN_FILTER29,
    CAN_FILTER30,
    CAN_FILTER31
} CAN_FILTER;
```

Members

Members	Description
CAN_CHANNEL0_EVENT	An event on Channel 0 is active
CAN_CHANNEL1_EVENT	An event on Channel 1 is active
CAN_CHANNEL2_EVENT	An event on Channel 2 is active
CAN_CHANNEL3_EVENT	An event on Channel 3 is active
CAN_CHANNEL4_EVENT	An event on Channel 4 is active
CAN_CHANNEL5_EVENT	An event on Channel 5 is active
CAN_CHANNEL6_EVENT	An event on Channel 6 is active
CAN_CHANNEL7_EVENT	An event on Channel 7 is active
CAN_CHANNEL8_EVENT	An event on Channel 8 is active
CAN_CHANNEL9_EVENT	An event on Channel 9 is active
CAN_CHANNEL10_EVENT	An event on Channel 10 is active
CAN_CHANNEL11_EVENT	An event on Channel 11 is active
CAN_CHANNEL12_EVENT	An event on Channel 12 is active
CAN_CHANNEL13_EVENT	An event on Channel 13 is active
CAN_CHANNEL14_EVENT	An event on Channel 14 is active
CAN_CHANNEL15_EVENT	An event on Channel 15 is active
CAN_CHANNEL16_EVENT	An event on Channel 16 is active
CAN_CHANNEL17_EVENT	An event on Channel 17 is active
CAN_CHANNEL18_EVENT	An event on Channel 18 is active
CAN_CHANNEL19_EVENT	An event on Channel 19 is active

CAN_CHANNEL20_EVENT	An event on Channel 20 is active
CAN_CHANNEL21_EVENT	An event on Channel 21 is active
CAN_CHANNEL22_EVENT	An event on Channel 22 is active
CAN_CHANNEL23_EVENT	An event on Channel 23 is active
CAN_CHANNEL24_EVENT	An event on Channel 24 is active
CAN_CHANNEL25_EVENT	An event on Channel 25 is active
CAN_CHANNEL26_EVENT	An event on Channel 26 is active
CAN_CHANNEL27_EVENT	An event on Channel 27 is active
CAN_CHANNEL28_EVENT	An event on Channel 28 is active
CAN_CHANNEL29_EVENT	An event on Channel 29 is active
CAN_CHANNEL30_EVENT	An event on Channel 30 is active
CAN_CHANNEL31_EVENT	An event on Channel 31 is active
CAN_NO_EVENT	No Event is active
CAN_ERROR_EVENT	CAN Bus Error Event is active
CAN_WAKEUP_EVENT	CAN Bus Wakeup Event is active
CAN_RX_CHANNEL_OVERFLOW_EVENT	CAN Receive Channel Overflow Event is active
CAN_ADDRESS_ERROR_EVENT	CAN Address Error Event is active
CAN_BUS_BANDWIDTH_ERROR	CAN Bus Bandwidth Error is active
CAN_TIMESTAMP_TIMER_EVENT	CAN Time Stamp Timer Overflow event is active
CAN_MODE_CHANGE_EVENT	CAN Module Mode Change is active

Description

CAN Event Code

This enumeration identifies all of the event codes returned by the PLIB_CAN_PendingEventsGet function.

CAN_FILTER_MASK Enumeration

Identifies the available CAN filter masks.

File

plib_can_help.h

C

```
typedef enum {
  CAN_FILTER_MASK0,
  CAN_FILTER_MASK1,
  CAN_FILTER_MASK2,
  CAN_FILTER_MASK3,
  CAN_NUMBER_OF_FILTER_MASKS
} CAN_FILTER_MASK;
```

Members

Members	Description
CAN_FILTER_MASK0	CAN Filter Mask 0
CAN_FILTER_MASK1	CAN Filter Mask 1
CAN_FILTER_MASK2	CAN Filter Mask 2
CAN_FILTER_MASK3	CAN Filter Mask 3
CAN_NUMBER_OF_FILTER_MASKS	Total number of filter masks in the module

Description

CAN Filter Masks

This enumeration identifies the available filters masks in each CAN module.

CAN_FILTER_MASK_TYPE Enumeration

Specifies the CAN filter mask type.

File

plib_can_help.h

C

```
typedef enum {
  CAN_FILTER_MASK_IDE_TYPE,
  CAN_FILTER_MASK_ANY_TYPE
} CAN_FILTER_MASK_TYPE;
```

Members

Members	Description
CAN_FILTER_MASK_IDE_TYPE	Mask processes only Filter type of message
CAN_FILTER_MASK_ANY_TYPE	Mask processes any type (SID or EID) of message

Description

CAN Filter Mask Type

This enumeration specifies the filter mask type. The filter mask can either process messages with any type of ID (extended or standard) or can only process by ID specified in the filter configuration. For example, if the filter associated with the mask only accepts EID type messages and if the mask is configured to process by ID type, then SID messages will not be accepted. If the mask is configured to process any type of message, both SID and EID messages will be accepted on a filter match.

CAN_ID_TYPE Enumeration

Specifies the CAN ID type.

File

plib_can_help.h

C

```
typedef enum {
  CAN_EID,
  CAN_SID
} CAN_ID_TYPE;
```

Members

Members	Description
CAN_EID	CAN Extended ID
CAN_SID	CAN Standard ID

Description

CAN ID Type

This enumeration specifies the two CAN ID types: Standard and Extended. The Standard Type ID is 11 bits long and the Extended ID is 29 bits long. This enumeration then specifies the type of the ID specified while configuring filters and filter masks.

CAN_MODULE_EVENT Enumeration

Specifies the CAN module events

File

plib_can_help.h

C

```
typedef enum {
  CAN_TX_EVENT,
  CAN_RX_EVENT,
  CAN_TIMESTAMP_TIMER_OVERFLOW_EVENT,
  CAN_OPERATION_MODE_CHANGE_EVENT,
  CAN_RX_OVERFLOW_EVENT,
  CAN_SYSTEM_ERROR_EVENT,
  CAN_BUS_ERROR_EVENT,
  CAN_BUS_ACTIVITY_WAKEUP_EVENT,
  CAN_INVALID_RX_MESSAGE_EVENT
} CAN_MODULE_EVENT;
```

Members

Members	Description
CAN_TX_EVENT	Transmit channel event. This event will occur if any of the Transmit Channel events are active.
CAN_RX_EVENT	Receive channel event. This event will occur if any of the Receive Channel events are active.
CAN_TIMESTAMP_TIMER_OVERFLOW_EVENT	CAN Time Stamp Timer Overflow event occurs. This event occurs when the Time Stamp Timer has overflowed.
CAN_OPERATION_MODE_CHANGE_EVENT	CAN Operation Mode Change Event. This event occurs when the CAN module has changed its operating mode successfully.
CAN_RX_OVERFLOW_EVENT	CAN Receive Channel Overflow Event. This event occurs when any of the Receive Channel has overflowed.
CAN_SYSTEM_ERROR_EVENT	CAN System Error Event. This event occurs when CAN module tries to access an invalid Device RAM location.
CAN_BUS_ERROR_EVENT	CAN Bus Error Event. This event occurs when the CAN module cannot access the system bus.
CAN_BUS_ACTIVITY_WAKEUP_EVENT	CAN Bus Activity Wakeup. This event occurs when the device is in Sleep mode and bus activity is detected on the CAN bus.
CAN_INVALID_RX_MESSAGE_EVENT	CAN Bus Invalid Receive Message Event. This event occurs when the CAN module receives an Invalid message.

Description

CAN Module Events

This enumeration identifies all of the CAN module events. A combination of listed events can be logically ORed to enable or disable module level events. Similarly, a combination of events can be checked if active.

This enumeration is used with the PLIB_CAN_ModuleEventEnable function and PLIB_CAN_ModuleEventGet function.

CAN_MODULE_ID Enumeration

File

plib_can_help.h

C

```
typedef enum {
  CAN_ID_1,
  CAN_ID_2,
  CAN_ID_3,
  CAN_NUMBER_OF_MODULES
} CAN_MODULE_ID;
```

Members

Members	Description
CAN_NUMBER_OF_MODULES	The total number of modules available.

Description

Enumeration: CAN_MODULE_ID

This enumeration defines the number of modules that are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers.

Refer to the specific device data sheet to get the correct number of modules defined for the desired microcontroller.

CAN_MSG_EID Structure

Defines the structure of the EID word section of the transmit and receive message.

File

plib_can.h

C

```
typedef struct {
  unsigned data_length_code : 4;
  unsigned reserved0 : 1;
  unsigned unimplemented1 : 3;
  unsigned reserved1 : 1;
```

```
  unsigned remote_request : 1;
  unsigned eid : 18;
  unsigned ide : 1;
  unsigned sub_remote_req : 1;
  unsigned unimplemented2 : 2;
} CAN_MSG_EID;
```

Members

Members	Description
unsigned data_length_code : 4;	Data Length Control. Specifies the size of the data payload section of the CAN packet. Valid values are 0x0 - 0x8
unsigned reserved0 : 1;	Reserved bit. Should be always 0.
unsigned unimplemented1 : 3;	Unimplemented bit. Should be always 0.
unsigned reserved1 : 1;	Reserved bit. Should be always 0.
unsigned remote_request : 1;	Remote Transmit Request bit. Should be set for RTR messages, clear otherwise.
unsigned eid : 18;	CAN Transmit and Receive Extended ID field. Valid values are in range 0x0 - 0x3FFFF
unsigned ide : 1;	Identifier bit. If 0 means that message is SID. If 1 means that message is EID type.
unsigned sub_remote_req : 1;	Substitute Remote request bit. This bit should always be clear for an EID message. It is ignored for an SID message.
unsigned unimplemented2 : 2;	Unimplemented bit. Should be always 0.

Description

CAN Message EID Word

This data structure represents the EID section of the CAN transmit and receive message. The data structure is an element of the CAN_TX_MSG_BUFFER and CAN_RX_MSG_BUFFER data structure.

CAN_OPERATION_MODES Enumeration

Lists all possible CAN module operational modes.

File

plib_can_help.h

C

```
typedef enum {
  CAN_LISTEN_ALL_MESSAGES_MODE,
  CAN_CONFIGURATION_MODE,
  CAN_LISTEN_ONLY_MODE,
  CAN_LOOPBACK_MODE,
  CAN_DISABLE_MODE,
  CAN_NORMAL_MODE
} CAN_OPERATION_MODES;
```

Members

Members	Description
CAN_LISTEN_ALL_MESSAGES_MODE	CAN Listen All Message Mode. The CAN module listens to all messages, regardless of errors
CAN_CONFIGURATION_MODE	CAN Configuration Mode. Various CAN module settings can be configured in this mode
CAN_LISTEN_ONLY_MODE	CAN Listen Only Mode. In this mode, the CAN module will not acknowledge signal and will not participate in error signaling. All messages are captured
CAN_LOOPBACK_MODE	CAN Loopback Mode. In this mode, the CAN module Transmit is internally connected to the Receive line. This mode is useful for debugging operation
CAN_DISABLE_MODE	CAN Disable Mode. The CAN module does not transmit or receive messages in this mode
CAN_NORMAL_MODE	CAN Normal Operation Mode. The CAN module transmits and receives messages in this mode

Description

CAN Operation Modes

This enumerates all operating modes of the CAN module. The application should set the desired mode using the PLIB_CAN_OperationModeSelect function, and should then use the PLIB_CAN_OperationModeGet function to check if the CAN module has entered the requested mode.

CAN_RECEIVE_CHANNEL Enumeration

Lists all possible CAN module receive channels.

File

plib_can_help.h

C

```
typedef enum {
  CAN_RECEIVE_CHANNEL0,
  CAN_RECEIVE_CHANNEL1
} CAN_RECEIVE_CHANNEL;
```

Members

Members	Description
CAN_RECEIVE_CHANNEL0	CAN receive channel 0
CAN_RECEIVE_CHANNEL1	CAN receive channel 1

Description

CAN Receive Channels

This enumerates all CAN module receive channels(channels that are not allowed to be configured for transmit).

CAN_RECEIVE_MODES Enumeration

Lists all possible CAN module receive modes.

File

plib_can_help.h

C

```
typedef enum {
  CAN_RECEIVE_ALL_MSG_MODE,
  CAN_RECEIVE_VALID_EID_MODE,
  CAN_RECEIVE_VALID_STD_MODE,
  CAN_RECEIVE_VALID_MODE
} CAN_RECEIVE_MODES;
```

Members

Members	Description
CAN_RECEIVE_ALL_MSG_MODE	Receive all messages(Including with errors). The message filters will not be considered
CAN_RECEIVE_VALID_EID_MODE	Receive only valid messages(with error will be discarded, filter will be considered) with extended identifier. This is applicable only for functional mode 0
CAN_RECEIVE_VALID_STD_MODE	Receive only valid messages(with error will be discarded, filter will be considered) with standard identifier. This is applicable only for functional mode 0
CAN_RECEIVE_VALID_MODE	Receive only valid messages(with error will be discarded, filter will be considered) with extended identifier/standard identifier Based on the configuration

Description

CAN Receive Modes

This enumerates all operating modes of the CAN module. The application should set the desired mode using the PLIB_CAN_ReceiveModeSelect function, and should then use the PLIB_CAN_ReceiveModeGet function to check if the CAN module has entered the requested mode.

CAN_RX_DATA_MODE Enumeration

Enables the Data-only Receive mode or Full Receive mode of a CAN receive channel.

File

plib_can_help.h

C

```
typedef enum {
```

```
  CAN_RX_DATA_ONLY,
  CAN_RX_FULL_RECEIVE
} CAN_RX_DATA_MODE;
```

Members

Members	Description
CAN_RX_DATA_ONLY	CAN Receive Channel Data Only Mode is enabled
CAN_RX_FULL_RECEIVE	CAN Receive Channel Full Receive Mode is enabled

Description

CAN Receive Channel Data Only Mode

This enumeration specifies the status of the CAN receive channel data-only feature. If the feature is enabled, the CAN module will store only the data payload portion of the received CAN message. If the Full Receive mode is specified, the CAN module stores the entire CAN message (ID field plus data payload). The receive channel can be either in Data-Only mode or Full Receive mode.

CAN_RX_MSG_BUFFER Union

Defines the structure of the CAN receive message buffer

File

plib_can.h

C

```
typedef union {
  struct {
    CAN_RX_MSG_SID msgSID;
    CAN_MSG_EID msgEID;
    uint8_t data[8];
  }
  uint8_t dataOnlyMsgData[8];
  uint32_t messageWord[4];
} CAN_RX_MSG_BUFFER;
```

Members

Members	Description
CAN_RX_MSG_SID msgSID;	This is SID portion of the CAN Receive message
CAN_MSG_EID msgEID;	This is EID portion of the CAN Receive message
uint8_t data[8];	This is the data payload section of the received message
uint8_t dataOnlyMsgData[8];	This can be used if the message buffer is to be read from a Data-Only type of CAN Receive Channel
uint32_t messageWord[4];	This is CAN Receive message organized as a set of 32-bit words

Description

CAN Receive Message Buffer

This data structure represents the CAN receive message buffer. Received messages could be either full-receive messages or data-only messages. A full-receive message contains the message header and data payload section. For a full-receive CAN receive channel, the caller should use the msgSID, msgEID and data members. A data-only message contains only an 8-byte data payload. While using this data structure with a data-only type of CAN receive channel, the caller should use the dataOnlyMsgData member of the structure and should read only 8 bytes of data.

CAN_RX_MSG_SID Structure

Defines the structure of the SID word section of the receive message.

File

plib_can.h

C

```
typedef struct {
  unsigned sid : 11;
  unsigned filter_hit : 5;
  unsigned msg_timestamp : 16;
} CAN_RX_MSG_SID;
```

Members

Members	Description
unsigned sid : 11;	SID of the Received CAN Message
unsigned filter_hit : 5;	Filter which accepted this message
unsigned msg_timestamp : 16;	Time stamp of the received message. This is valid only if the Timestamping is enabled

Description

CAN Receive Message SID Word

This data structure represents the SID section of the CAN receive message. The data structure is an element of the CAN_RX_MSG_BUFFER data structure.

CAN_TIME_SEGMENT_LENGTH Enumeration

All possible values for the assignable number of Time Quanta.

File

plib_can_help.h

C

```
typedef enum {
  CAN_TIME_SEGMENT_LEN_8TQ,
  CAN_TIME_SEGMENT_LEN_7TQ,
  CAN_TIME_SEGMENT_LEN_6TQ,
  CAN_TIME_SEGMENT_LEN_5TQ,
  CAN_TIME_SEGMENT_LEN_4TQ,
  CAN_TIME_SEGMENT_LEN_3TQ,
  CAN_TIME_SEGMENT_LEN_2TQ,
  CAN_TIME_SEGMENT_LEN_1TQ
} CAN_TIME_SEGMENT_LENGTH;
```

Members

Members	Description
CAN_TIME_SEGMENT_LEN_8TQ	Segment length is 8 x TQ
CAN_TIME_SEGMENT_LEN_7TQ	Segment length is 7 x TQ
CAN_TIME_SEGMENT_LEN_6TQ	Segment length is 6 x TQ
CAN_TIME_SEGMENT_LEN_5TQ	Segment length is 5 x TQ
CAN_TIME_SEGMENT_LEN_4TQ	Segment length is 4 x TQ
CAN_TIME_SEGMENT_LEN_3TQ	Segment length is 3 x TQ
CAN_TIME_SEGMENT_LEN_2TQ	Segment length is 2 x TQ
CAN_TIME_SEGMENT_LEN_1TQ	Segment length is 1 x TQ

Description

CAN Segment length

This enumeration defines values that can be assigned while defining the number of Time Quanta in a bit.

CAN_TX_CHANNEL_STATUS Enumeration

Identifies possible transmit channel-specific conditions.

File

plib_can_help.h

C

```
typedef enum {
  CAN_TX_CHANNEL_TRANSMITTING,
  CAN_TX_CHANNEL_ERROR,
  CAN_TX_CHANNEL_ARBITRATION_LOST
} CAN_TX_CHANNEL_STATUS;
```

Members

Members	Description
CAN_TX_CHANNEL_TRANSMITTING	CAN Transmit Channel is currently Transmitting.
CAN_TX_CHANNEL_ERROR	CAN Transmit Channel Error has occurred.
CAN_TX_CHANNEL_ARBITRATION_LOST	CAN Transmit Channel lost arbitration.

Description

CAN Transmit Channel Condition

This enumeration identifies the possible transmit channel condition. These masks can be logically ANDed with values returned by the PLIB_CAN_TransmitChannelStatusGet function to check if a condition is active.

CAN_TX_MSG_BUFFER Union

Defines the structure of the CAN transmit message buffer.

File

plib_can.h

C

```
typedef union {
  struct {
    CAN_TX_MSG_SID msgSID;
    CAN_MSG_EID msgEID;
    unsigned char data[8];
  }
  uint32_t messageWord[4];
} CAN_TX_MSG_BUFFER;
```

Members

Members	Description
CAN_TX_MSG_SID msgSID;	This is SID portion of the CAN Transmit message
CAN_MSG_EID msgEID;	This is EID portion of the CAN Transmit message
unsigned char data[8];	This is the data portion of the CAN Transmit message
uint32_t messageWord[4];	This is CAN Transmit message organized as a set of 32 bit words

Description

CAN Transmit Message Buffer

This data structure represents the CAN transmit message buffer. This should be used for writing data to a CAN transmit channel and transmitting data. A pointer to this type of data structure is returned by the PLIB_CAN_TransmitBufferGet function.

The data structure allows the transmit message buffer to be accessed as fields of a CAN transmit message and also as an array of four 32-bit words. The latter allows for quick initialization of the message buffer.

CAN_TX_MSG_SID Structure

Defines the structure of the SID word section of the transmit message.

File

plib_can.h

C

```
typedef struct {
  unsigned sid : 11;
} CAN_TX_MSG_SID;
```

Members

Members	Description
unsigned sid : 11;	CAN Transmit Message Standard ID. This value should be between 0x0 - 0x7FF

Description

CAN Transmit Message SID Word

This data structure represents the SID section of the CAN transmit message. The data structure is an element of the CAN_TX_MSG_BUFFER data structure.

CAN_TX_RTR Enumeration

Specifies the status of the CAN Remote Transmit Request (RTR) feature for a CAN transmit channel.

File

plib_can_help.h

C

```
typedef enum {
  CAN_TX_RTR_ENABLED,
  CAN_TX_RTR_DISABLED
} CAN_TX_RTR;
```

Members

Members	Description
CAN_TX_RTR_ENABLED	CAN Transmit Channel RTR Feature is enabled
CAN_TX_RTR_DISABLED	CAN Transmit Channel RTR Feature is disabled

Description

CAN Transmit Channel Remote Transmit Request (RTR)

This enumeration specifies the status of the CAN RTR feature for a CAN transmit channel. The RTR feature allows a node on the CAN Bus to request a transmission from another node on the bus. The responding node in this case should have a RTR enabled Transmit Channel in order to be able to respond to this request.

CAN_TXCHANNEL_PRIORITY Enumeration

Specifies the priority of a transmit channel.

File

plib_can_help.h

C

```
typedef enum {
  CAN_LOWEST_PRIORITY,
  CAN_LOW_MEDIUM_PRIORITY,
  CAN_HIGH_MEDIUM_PRIORITY,
  CAN_HIGHEST_PRIORITY
} CAN_TXCHANNEL_PRIORITY;
```

Members

Members	Description
CAN_LOWEST_PRIORITY	CAN lowest priority
CAN_LOW_MEDIUM_PRIORITY	CAN low medium priority
CAN_HIGH_MEDIUM_PRIORITY	CAN high medium priority
CAN_HIGHEST_PRIORITY	CAN highest priority

Description

CAN Transmit Channel Priority

This enumeration identifies the available transmit channel priorities. A transmit channel has its own natural priority order, which determines priority when two or more transmit channels are assigned the same priority level. Channel 1 has higher natural priority than channel 0 and channel 2 has a natural priority than channel 1, and so on.

CAN_RX_DATA_ONLY_SIZE_BYTES Macro

Used as the size of the CAN data-only receive message buffer in bytes.

File

plib_can.h

C

```
#define CAN_RX_DATA_ONLY_SIZE_BYTES 8
```

Description

CAN Data-Only Receive Message Buffer

This constant is used as the size of the CAN data-only receive message buffer in bytes.

CAN_TX_RX_MESSAGE_SIZE_BYTES Macro

Used as the array size of the CAN transmit and full receive message buffer.

File

plib_can.h

C

```
#define CAN_TX_RX_MESSAGE_SIZE_BYTES 16
```

Description

CAN Transmit and Receive Message Buffer size

This constant is used as the array size of the CAN transmit and full receive message buffer.

Files

Files

Name	Description
plib_can.h	This file contains the interface definition for the CAN Peripheral Library.
plib_can_help.h	This file contains the enumerations for the CAN Peripheral Library help file.

Description

This section lists the source and header files used by the library.

plib_can.h

This file contains the interface definition for the CAN Peripheral Library.

Functions

	Name	Description
	PLIB_CAN_AllChannelEventsGet	Returns a value representing the event status of all CAN channels.
	PLIB_CAN_AllChannelOverflowStatusGet	Returns a value representing the receive overflow event status of all CAN channels.
	PLIB_CAN_BaudRateGet	Returns the current CAN module Baud rate.
	PLIB_CAN_BaudRatePrescaleSetup	Sets the prescale divisor applied to the CAN module's input clock before it is used to determine the CAN baud rate.
	PLIB_CAN_BitSamplePhaseSet	Sets the CAN bit-sampling phase parameters.
	PLIB_CAN_BusActivityWakeupDisable	Disables the wake-up on bus activity receive line filter.
	PLIB_CAN_BusActivityWakeupEnable	Enables the wake-up on bus activity receive line filter.
	PLIB_CAN_BusLine3TimesSamplingDisable	Disables the bus line three times sampling.
	PLIB_CAN_BusLine3TimesSamplingEnable	Enables the bus line three times sampling.
	PLIB_CAN_ChannelEventClear	Clears CAN channel events.
	PLIB_CAN_ChannelEventDisable	Enables channel level events.
	PLIB_CAN_ChannelEventEnable	Enables channel level events.
	PLIB_CAN_ChannelEventGet	Returns a value representing the event status of a CAN channel.
	PLIB_CAN_ChannelForReceiveSet	Configures a CAN channel for receive operation.
	PLIB_CAN_ChannelForTransmitSet	Configures a CAN channel for transmission.
	PLIB_CAN_ChannelReset	Resets a CAN channel.
	PLIB_CAN_ChannelResetIsComplete	Returns 'true' if the specified channel reset is complete.
	PLIB_CAN_ChannelUpdate	Updates the CAN Channel internal pointers.
	PLIB_CAN_DeviceNetConfigure	Configures the CAN module DeviceNet(TM) filter feature.

	PLIB_CAN_Disable	Disables the specified CAN module.
	PLIB_CAN_Enable	Enables the specified CAN module.
	PLIB_CAN_ErrorStateGet	Returns the CAN error status word.
	PLIB_CAN_ExistsActiveStatus	Identifies whether the ActiveStatus feature exists on the CAN module.
	PLIB_CAN_ExistsAllChannelEvents	Identifies whether the AllChannelEvents feature exists on the CAN module.
	PLIB_CAN_ExistsAllChannelOverflow	Identifies whether the AllChannelOverflow feature exists on the CAN module.
	PLIB_CAN_ExistsBaudRateGet	Identifies whether the BaudRateGet feature exists on the CAN module.
	PLIB_CAN_ExistsBaudRatePrescaleSetup	Identifies whether the BaudRatePrescaleSetup feature exists on the CAN module.
	PLIB_CAN_ExistsBitSamplePhaseSet	Identifies whether the BitSamplePhaseSet feature exists on the CAN module.
	PLIB_CAN_ExistsBusActivityWakeup	Identifies whether the BusActivityWakeup feature exists on the CAN module.
	PLIB_CAN_ExistsBusLine3TimesSampling	Identifies whether the BusLine3TimesSampling feature exists on the CAN module.
	PLIB_CAN_ExistsChannelEvent	Identifies whether the ChannelEventGet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelEventEnable	Identifies whether the ChannelEventEnable feature exists on the CAN module.
	PLIB_CAN_ExistsChannelForReceiveSet	Identifies whether the ChannelForReceiveSet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelForTransmitSet	Identifies whether the ChannelForTransmitSet feature exists on the CAN module.
	PLIB_CAN_ExistsChannelReset	Identifies whether the ChannelReset feature exists on the CAN module.
	PLIB_CAN_ExistsChannelUpdate	Identifies whether the ChannelUpdate feature exists on the CAN module.
	PLIB_CAN_ExistsDeviceNet	Identifies whether the DeviceNet feature exists on the CAN module.
	PLIB_CAN_ExistsEnableControl	Identifies whether the EnableControl feature exists on the CAN module.
	PLIB_CAN_ExistsErrorState	Identifies whether the ErrorStateGet feature exists on the CAN module.
	PLIB_CAN_ExistsFilterConfigure	Identifies whether the FilterConfigure feature exists on the CAN module.
	PLIB_CAN_ExistsFilterEnable	Identifies whether the FilterEnable feature exists on the CAN module.
	PLIB_CAN_ExistsFilterMaskConfigure	Identifies whether the FilterMaskConfigure feature exists on the CAN module.
	PLIB_CAN_ExistsFilterToChannelLink	Identifies whether the FilterToChannelLink feature exists on the CAN module.
	PLIB_CAN_ExistsLatestFilterMatchGet	Identifies whether the LatestFilterMatchGet feature exists on the CAN module.
	PLIB_CAN_ExistsMemoryBufferAssign	Identifies whether the MemoryBufferAssign feature exists on the CAN module.
	PLIB_CAN_ExistsModuleEventClear	Identifies whether the ModuleEvents feature exists on the CAN module.
	PLIB_CAN_ExistsModuleEventEnable	Identifies whether the ModuleEventEnable feature exists on the CAN module.
	PLIB_CAN_ExistsModuleInfo	Identifies whether the ModuleInformation feature exists on the CAN module.
	PLIB_CAN_ExistsOperationModeRead	Identifies whether the OperationModeRead feature exists on the CAN module.
	PLIB_CAN_ExistsOperationModeWrite	Identifies whether the OperationModeSet feature exists on the CAN module.
	PLIB_CAN_ExistsPendingEventsGet	Identifies whether the PendingEventsGet feature exists on the CAN module.
	PLIB_CAN_ExistsPendingTransmissionsAbort	Identifies whether the PendingTransmissionsAbort feature exists on the CAN module.
	PLIB_CAN_ExistsPrecalculatedBitRateSetup	Identifies whether the PrecalculatedBitRateSetup feature exists on the CAN module.
	PLIB_CAN_ExistsReceivedMessageGet	Identifies whether the ReceivedMessageGet feature exists on the CAN module.
	PLIB_CAN_ExistsReceiveErrorCount	Identifies whether the ReceiveErrorCount feature exists on the CAN module.
	PLIB_CAN_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CAN module.
	PLIB_CAN_ExistsTimeStampEnable	Identifies whether the TimeStampEnable feature exists on the CAN module
	PLIB_CAN_ExistsTimeStampPrescaler	Identifies whether the TimeStampPrescaler feature exists on the CAN module.
	PLIB_CAN_ExistsTimeStampValue	Identifies whether the TimeStampValue feature exists on the CAN module.
	PLIB_CAN_ExistsTransmissionIsAborted	Identifies whether the TransmissionAbortStatus feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitBufferGet	Identifies whether the TransmitBufferGet feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitChannelFlush	Identifies whether the TransmitChannelFlush feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitChannelStatus	Identifies whether the TransmitChannelStatus feature exists on the CAN module.
	PLIB_CAN_ExistsTransmitErrorCountGet	Identifies whether the TransmitErrorCountGet feature exists on the CAN module.
	PLIB_CAN_FilterConfigure	Configures a CAN message acceptance filter.
	PLIB_CAN_FilterDisable	Disables a CAN message acceptance filter.
	PLIB_CAN_FilterEnable	Enables a CAN message acceptance filter.
	PLIB_CAN_FilterIsDisabled	Returns 'true' if the specified filter is disabled.
	PLIB_CAN_FilterMaskConfigure	Configures a CAN filter mask.
	PLIB_CAN_FilterToChannelLink	Links a filter to a channel.
	PLIB_CAN_IsActive	Returns 'true' if the CAN module is active.
	PLIB_CAN_LatestFilterMatchGet	Returns the index of the filter that accepted the latest message.

	PLIB_CAN_MemoryBufferAssign	Assigns buffer memory to the CAN module.
	PLIB_CAN_ModuleEventClear	Clears the CAN module level events.
	PLIB_CAN_ModuleEventDisable	Disables the module level events.
	PLIB_CAN_ModuleEventEnable	Enables the module level events.
	PLIB_CAN_ModuleEventGet	Returns the status of the CAN module events.
	PLIB_CAN_OperationModeGet	Obtains the current CAN operating mode.
	PLIB_CAN_OperationModeSelect	Sets the CAN operating mode.
	PLIB_CAN_PendingEventsGet	Returns a value representing the highest priority active event in the module.
	PLIB_CAN_PendingTransmissionsAbort	Aborts any pending transmit operations.
	PLIB_CAN_PrecalculatedBitRateSetup	Sets the desired Baud rate for the respective CAN module.
	PLIB_CAN_ReceivedMessageGet	Returns a pointer to a message to be read from the CAN channel.
	PLIB_CAN_ReceiveErrorCountGet	Returns the CAN receive error count.
	PLIB_CAN_StopInIdleDisable	Disables the Stop in Idle feature.
	PLIB_CAN_StopInIdleEnable	Enables the CAN module to stop when the processor enters Idle mode.
	PLIB_CAN_TimeStampDisable	Disables the time stamp feature for the CAN module.
	PLIB_CAN_TimeStampEnable	Enables the time stamp feature for the CAN module.
	PLIB_CAN_TimeStampPrescalerSet	Sets the CAN receive message time stamp timer prescaler.
	PLIB_CAN_TimeStampValueGet	Returns the current value of the CAN receive message time stamp timer value.
	PLIB_CAN_TimeStampValueSet	Sets the CAN receive message time stamp timer value.
	PLIB_CAN_TotalChannelsGet	Returns the total number of CAN channels per CAN module.
	PLIB_CAN_TotalFiltersGet	Returns the total number of CAN Filters per CAN module.
	PLIB_CAN_TotalMasksGet	Returns the total number of CAN masks per CAN module.
	PLIB_CAN_TransmissionIsAborted	Returns 'true' if the transmit abort operation is complete.
	PLIB_CAN_TransmitBufferGet	Returns a pointer to an empty transmit buffer.
	PLIB_CAN_TransmitChannelFlush	Causes all messages in the specified transmit channel to be transmitted.
	PLIB_CAN_TransmitChannelStatusGet	Returns the condition of the transmit channel.
	PLIB_CAN_TransmitErrorCountGet	Returns the CAN transmit error count

Macros

	Name	Description
	CAN_RX_DATA_ONLY_SIZE_BYTES	Used as the size of the CAN data-only receive message buffer in bytes.
	CAN_TX_RX_MESSAGE_SIZE_BYTES	Used as the array size of the CAN transmit and full receive message buffer.

Structures

	Name	Description
	CAN_MSG_EID	Defines the structure of the EID word section of the transmit and receive message.
	CAN_RX_MSG_SID	Defines the structure of the SID word section of the receive message.
	CAN_TX_MSG_SID	Defines the structure of the SID word section of the transmit message.

Unions

	Name	Description
	CAN_RX_MSG_BUFFER	Defines the structure of the CAN receive message buffer
	CAN_TX_MSG_BUFFER	Defines the structure of the CAN transmit message buffer.

Description

CAN Peripheral Library Interface Header

This library provides a low-level abstraction of the Controller Area Network (CAN) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

File Name

plib_can.h

Company

Microchip Technology Inc.

plib_can_help.h

This file contains the enumerations for the CAN Peripheral Library help file.

Enumerations

	Name	Description
	CAN_CHANNEL	Identifies the supported CAN Channels.
	CAN_CHANNEL_EVENT	Identifies all Layer 3 interrupts.
	CAN_CHANNEL_MASK	Lists the series of useful masks.
	CAN_DNET_FILTER_SIZE	Specifies the size of the DeviceNet filter.
	CAN_ERROR_STATE	Specifies the CAN module error states.
	CAN_FILTER	CAN event code returned by the CAN module.
	CAN_FILTER_MASK	Identifies the available CAN filter masks.
	CAN_FILTER_MASK_TYPE	Specifies the CAN filter mask type.
	CAN_ID_TYPE	Specifies the CAN ID type.
	CAN_MODULE_EVENT	Specifies the CAN module events
	CAN_MODULE_ID	Enumeration: CAN_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers. Refer to the specific device data sheet to get the correct number of modules defined for the desired microcontroller.
	CAN_OPERATION_MODES	Lists all possible CAN module operational modes.
	CAN_RECEIVE_CHANNEL	Lists all possible CAN module receive channels.
	CAN_RECEIVE_MODES	Lists all possible CAN module receive modes.
	CAN_RX_DATA_MODE	Enables the Data-only Receive mode or Full Receive mode of a CAN receive channel.
	CAN_TIME_SEGMENT_LENGTH	All possible values for the assignable number of Time Quanta.
	CAN_TX_CHANNEL_STATUS	Identifies possible transmit channel-specific conditions.
	CAN_TX_RTR	Specifies the status of the CAN Remote Transmit Request (RTR) feature for a CAN transmit channel.
	CAN_TXCHANNEL_PRIORITY	Specifies the priority of a transmit channel.

Description

CAN Peripheral Library help file

This file contains the enumerations for the CAN Peripheral Library help file.

File Name

plib_can_help.h

Company

Microchip Technology Inc.

Comparator Peripheral Library

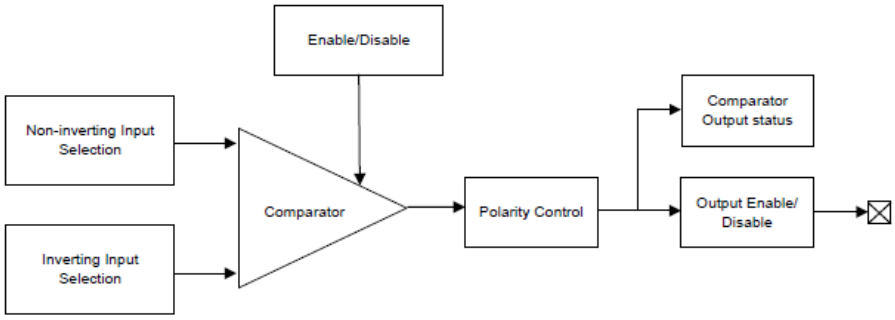
This section describes the Comparator Peripheral Library.

Introduction

The Comparator module is comprised of several identical analog comparator blocks, each complete with its own supporting input selectors and output logic. Each Comparator can be configured in a variety of ways, independent of the other comparators. The input voltage to the module can be either fed externally or it can be configured to use the internal voltage.

Description

This library provides a low-level abstraction of the Comparator module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby abstracting hardware differences from one microcontroller variant to another.



Using the Library

This topic describes the basic architecture of the Comparator Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_cmp.h`

The interface to the Comparator Peripheral Library is defined in the `plib_cmp.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Comparator Peripheral Library must include `peripheral.h`.

Library File:

The Comparator Peripheral Library is part of the processor-specific peripheral library installed with the compiler. This library is automatically available (in the default search path) for any project built using the Microchip compilers.

Peripheral Module IDs

Peripheral libraries are indexed to allow a single library to control any number of instances of a peripheral in a single microcontroller. The first parameter to each operation in a peripheral library is the module instance ID. The module instance ID is defined by an enumeration that is defined in the processor-specific header files (included by the library's interface header). For the Comparator Peripheral Library, the module instance IDs are defined by the Comparator module enumeration.

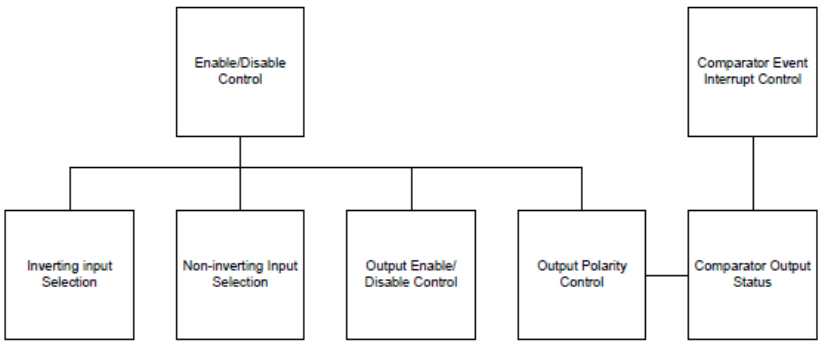
Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

Hardware Abstraction Model

This library provides a low-level abstraction of the Comparator (CMP) module on Microchip PIC microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model Diagram



Hardware Abstraction Model Description

The Comparator Peripheral Library provides interface routines to interact with the blocks shown in the previous diagram.

The Comparator VIN+ and VIN- block selects the Comparator non-inverting and inverting inputs, which can be either external or internal inputs.

The Comparator module block compares the voltage level on the inputs and produce an output. If the input VIN+ > VIN-, the output will be high. The output will be low when VIN+ < VIN-.

The Comparator Output Polarity block selects the Comparator output polarity.

The Comparator Output Control block enables/disables the Comparator output pin.

The Comparator Interrupt Logic Control block decide the comparator interrupt generation. The interrupt can happen at Low-to-high, High-to-Low, and for both transitions of the Comparator output.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Comparator module

Library Interface Section	Description
General Configuration	Provides the configuration routines for: • Enabling and disabling the module • Enabling and disabling Comparator output • Selecting Comparator input • Comparator status
Comparator Feature Configuration	Provides the configuration routines for various features in the Comparator: • Enabling and disabling Comparator Stop in Idle mode • Selecting Comparator speed/power • Enabling and disabling Comparator hysteresis • Enabling and disabling Comparator filters
Feature Existence Functions	These functions determine whether or not a particular feature is supported by the device.

How the Library Works

The following processes are involved while using a Comparator module:

- CVREF Setup
- Initialization
- Configuring the Comparator interrupt (optional)
- Power-Saving modes
- Turning on the Comparator module

CVREF Setup

This topic describes how to configure the Comparator Voltage Reference (CVREF) if it is to be used as one of the inputs for the Comparator module.

Description

The CVREF is one of the internal voltage references available for the Comparator module, which can be configured for the required voltage value.

Do the following to set up the CVREF:

1. Select the voltage source for CVREF using PLIB_CMP_CVREF_SourceVoltageSelect.
2. Select the CVREF voltage range using PLIB_CMP_CVREF_WideRangeEnable and/or PLIB_CMP_CVREF_WideRangeDisable.
3. Select the CVREF value using PLIB_CMP_CVREF_ValueSelect.
4. Enable or disable the CVREF pin output using PLIB_CMP_CVREF_OutputEnable or PLIB_CMP_CVREF_OutputDisable.
5. Turn on the CVREF module using PLIB_CMP_CVREF_Enable.

Example

```
#define MY_CMP_ID          CMP_ID_1

//Select the CVREF source
PLIB_CMP_CVREF_SourceVoltageSelect(MY_CMP_ID, CMP_CVREF_VOLTAGE_SOURCE_VDD);

//Select the CVREF for wide range
PLIB_CMP_CVREF_WideRangeEnable(MY_CMP_ID);

//Select the CVREF value
PLIB_CMP_CVREF_ValueSelect(MY_CMP_ID, CMP_CVREF_VALUE_15);

//Enable the CVREF output pin
PLIB_CMP_CVREF_OutputEnable(MY_CMP_ID);
```

```
//Turn on the CVREF module
PLIB_CMP_CVREF_Enable(MY_CMP_ID);
```

Initialization

This section describes Comparator module initialization.

Description

Do the following when using a Comparator module:

1. Select the Comparator inverting input using PLIB_CMP_InvertingInputChannelSelect.
2. Select the Comparator non-inverting input using PLIB_CMP_NonInvertingInputChannelSelect.
3. Select the Comparator output polarity using PLIB_CMP_OutputInvertEnable and/or PLIB_CMP_OutputInvertDisable.
4. Select the Comparator output pin using PLIB_CMP_OutputEnable and/or PLIB_CMP_OutputDisable.
5. Turn on the Comparator module using PLIB_CMP_Enable.

Example

```
#define MY_CMP_ID          CMP_ID_1

//Select the comparator inverting input
PLIB_CMP_InvertingInputChannelSelect(MY_CMP_INSTANCE,CMP_INVERTING_INPUT_CHANNEL_B);

//Select the comparator non-inverting input
PLIB_CMP_NonInvertingInputChannelSelect(MY_CMP_INSTANCE,CMP_NON_INVERTING_INPUT_A);

//Select the comparator output polarity
 PLIB_CMP_OutputInvertDisable(MY_CMP_INSTANCE);

//Enable the comparator output pin
PLIB_CMP_OutputEnable(MY_CMP_INSTANCE);

//Turn on the comparator module
PLIB_CMP_Enable(MY_CMP_INSTANCE);
```

Configuring the Comparator Interrupts

This section describes the configuration of Comparator module interrupts.

Description

The comparator interrupt can happen for three events. The user must configure the event-detection logic to report changes to the output state of the Comparator, which can in turn be used for event or interrupt generation. The options include event generation on rising state changes (low-to-high), falling state changes (high-to-low), and all state changes. When a configured event occurs, it is flagged by the event detection flag. The user must clear this flag for the next interrupt generation.

The following step is involved while configuring a Comparator interrupt:

1. Select the Comparator interrupt event using PLIB_CMP_InterruptEventSelect.

Example

```
#define MY_CMP_ID          CMP_ID_1

//This code expects the comparator interrupt
//enabled in the interrupt module

//Select the comparator interrupt event
PLIB_CMP_InterruptEventSelect(MY_CMP_INSTANCE, CMP_LOW_TO_HIGH);

// If the event meets, interrupt will occur. Do the respective
   operation in the ISR and clear the interrupt flag.
```

Note: Not all functionality is available on all devices. Please refer to the specific device data sheet for availability.

Power-Saving Modes

This section provides information on the Power-Saving modes for the Comparator module.

Description

Operation in Sleep mode: If the Comparator interrupt is enabled, the device will wake up from Sleep mode on the Comparator interrupt. If the Comparator interrupt is disabled,the comparator will work, but it will not wake up the device from sleep.

Operation in Idle mode: The functions PLIB_CMP_StopInIdleModeEnable and PLIB_CMP_StopInIdleModeDisable determine if the Comparator module stops or continues operation in Idle mode. If the function PLIB_CMP_StopInIdleModeDisable is used, the module will continue operation in the Idle mode. If the Comparator interrupt is enabled, the device will wake up from Idle mode on the Comparator interrupt. If the function PLIB_CMP_StopInIdleModeEnable is used, the module will work in Idle mode but the interrupt will not generate if it is enabled.

#	Description	Function Associated
1	Enable Comparator Stop in Idle mode.	PLIB_CMP_StopInIdleModeEnable
2	Disable Comparator Stop in Idle mode.	PLIB_CMP_StopInIdleModeDisable

Example

```
//Select comparator stop in idle mode
PLIB_CMP_StopInIdleModeEnable(MY_CMP_INSTANCE);
```

Note: Not all functionality is available on all devices. Please refer to the specific device data sheet for availability.

Configuring the Library

The library is configured for the supported Comparator module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_CMP_CVREF_BandGapReferenceSourceSelect	Selects the band gap reference voltage source.
	PLIB_CMP_CVREF_Enable	Enables the voltage reference of the Comparator module.
	PLIB_CMP_CVREF_OutputDisable	Disables the output voltage.
	PLIB_CMP_CVREF_OutputEnable	Enables the voltage output.
	PLIB_CMP_CVREF_ReferenceVoltageSelect	Selects the voltage reference value, CVref.
	PLIB_CMP_CVREF_SourceNegativeInputSelect	Configures the Comparator module to use the selected input as a negative reference.
	PLIB_CMP_CVREF_SourceVoltageSelect	Connects the Comparator module to the selected voltage source.
	PLIB_CMP_CVREF_ValueSelect	Selects the voltage reference value.
	PLIB_CMP_CVREF_WideRangeDisable	Disables the wide range.
	PLIB_CMP_CVREF_WideRangeEnable	Enables the wide range.
	PLIB_CMP_CVREF_WideRangeIsEnabled	Returns whether the wide range is selected for the reference voltage.
	PLIB_CMP_CVREF_Disable	Disables the voltage reference of the Comparator module.
	PLIB_CMP_Disable	Disables the Comparator module.
	PLIB_CMP_Enable	Enables the Comparator module.
	PLIB_CMP_InterruptEventSelect	Comparator interrupt event select.
	PLIB_CMP_InvertingInputChannelSelect	Comparator inverting input channel select.
	PLIB_CMP_NonInvertingInputChannelSelect	Comparator input channel select.
	PLIB_CMP_OutputDisable	Disables the Comparator output.
	PLIB_CMP_OutputEnable	Enables the Comparator output.
	PLIB_CMP_OutputStatusGet	Comparator output status.
	PLIB_CMP_OutputInvertDisable	Comparator output is non-inverted.
	PLIB_CMP_OutputInvertEnable	Comparator output is inverted.
	PLIB_CMP_OutputLogicHigh	Comparator output bit will give a 'logic high' on satisfying the input condition.
	PLIB_CMP_OutputLogicLow	Comparator will be set to give 'logic low' on satisfying the input condition.

b) Comparator Feature Configuration Functions

	Name	Description
	PLIB_CMP_StopInIdleModeDisable	Disables Stop in Idle mode.

	PLIB_CMP_StopInIdleModeEnable	Enables Stop in Idle mode.

c) Feature Existence Functions

	Name	Description
	PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect	Identifies whether the CVREFBGRefVoltageRangeSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFEnableControl	Identifies whether the CVREFEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFOutputEnableControl	Identifies whether the CVREFOutputEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFRefVoltageRangeSelect	Identifies whether the CVREFRefVoltageRangeSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFValueSelect	Identifies whether the CVREFValueSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFVoltageRangeSelect	Identifies whether the CVREFVoltageRangeSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFWideRangeControl	Identifies whether the CVREFWideRangeControl feature exists on the CMP module.
	PLIB_CMP_ExistsEnableControl	Identifies whether the ComparatorEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsInterruptEventSelect	Identifies whether the InterruptEventSelect feature exists on the CMP module.
	PLIB_CMP_ExistsInvertingInputSelect	Identifies whether the InvertingInputSelect feature exists on the CMP module.
	PLIB_CMP_ExistsInvertOutputControl	Identifies whether the InvertOutputSelectControl feature exists on the CMP module.
	PLIB_CMP_ExistsNonInvertingInputSelect	Identifies whether the NonInvertingInputSelect feature exists on the CMP module.
	PLIB_CMP_ExistsOutputEnableControl	Identifies whether the ComparatorOutputEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsOutputLevelControl	Identifies whether the OutputLevelSelectControl feature exists on the CMP module.
	PLIB_CMP_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CMP module.
	PLIB_CMP_ExistsOutputStatusGet	Identifies whether the OutputStatusGet feature exists on the CMP module.

d) Data Types and Constants

	Name	Description
	CMP_CLOCK_DIVIDE	Defines Comparator Filter Clock Divide.
	CMP_CVREF_BANDGAP_SELECT	Lists the band gap selection options.
	CMP_CVREF_REFERENCE_SELECT	Lists the reference selection options.
	CMP_CVREF_VALUE	Lists the voltage reference value selection options.
	CMP_CVREF_VOLTAGE_SOURCE	Lists the Voltage source options.
	CMP_FILTER_CLOCK	Defines Comparator filter input clock
	CMP_INTERRUPT_EVENT	Defines Comparator interrupt events.
	CMP_INVERTING_INPUT	Defines the list of Comparator inverting Input.
	CMP_MASK_A	Defines Comparator Mask A Input.
	CMP_MASK_B	Defines Comparator Mask B Input.
	CMP_MASK_C	Defines Comparator Mask C Input.
	CMP_MODULE_ID	Identifies the Comparator modules supported
	CMP_NON_INVERTING_INPUT	Defines the list of Comparator non-inverting Input.
	CMP_SPEED_POWER	Defines the Speed/Power of the Comparator

Description

This section describes the Application Programming Interface (API) functions of the Comparator Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_CMP_CVREF_BandGapReferenceSourceSelect Function

Selects the band gap reference voltage source.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_BandGapReferenceSourceSelect(CMP_MODULE_ID index, CMP_CVREF_BANDGAP_SELECT select);
```

Returns

None.

Description

This function selects the band gap reference voltage source from the available options from CMP_CVREF_BANDGAP_SELECT.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_BandGapReferenceSourceSelect ( CMP_ID_1,  CMP_CVREF_BANDGAP_0_6V );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
select	Select a band gap reference source from CMP_CVREF_BANDGAP_SELECT

Function

void PLIB_CMP_CVREF_BandGapReferenceSourceSelect (CMP_MODULE_ID index,
CMP_CVREF_BANDGAP_SELECT bandGap);

PLIB_CMP_CVREF_Enable Function

Enables the voltage reference of the Comparator module.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_Enable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the voltage reference of the Comparator module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFEnableControl in your application to determine whether this feature is available.

Preconditions

The Comparator module should be appropriately configured before being enabled.

Example

```
PLIB_CMP_CVREF_Enable ( CMP_ID_1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_CVREF_Enable (CMP_CVREF_MOD index)

PLIB_CMP_CVREF_OutputDisable Function

Disables the output voltage.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_OutputDisable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the reference voltage output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFOutputEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_OutputDisable ( CMP_ID_1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_CVREF_OutputDisable (CMP_MODULE_ID index)

PLIB_CMP_CVREF_OutputEnable Function

Enables the voltage output.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_OutputEnable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the voltage output

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFOutputEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_OutputEnable(CMP_ID_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_CVREF_OutputEnable (CMP_MODULE_ID index)

PLIB_CMP_CVREF_ReferenceVoltageSelect Function

Selects the voltage reference value, CVref.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_ReferenceVoltageSelect(CMP_MODULE_ID index, CMP_CVREF_REFERENCE_SELECT reference);
```

Returns

None.

Description

This function selects the voltage reference value, CVref. This value decides which voltage source should be taken as reference voltage from the set CMP_CVREF_REFERENCE_SELECT.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsCVREFRefVoltageRangeSelect in your application to determine whether this feature is available.

Preconditions

Determine the correct value that should be passed.

Example

```
PLIB_CMP_CVREF_ReferenceVoltageSelect ( CMP_ID_1,  CMP_CVREF_RESISTOR_LADDER_VOLTAGE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	Select value from CMP_CVREF_REFERENCE_SELECT

Function

void PLIB_CMP_CVREF_ReferenceVoltageSelect (CMP_MODULE_ID index,
CMP_CVREF_REFERENCE_SELECT reference);

PLIB_CMP_CVREF_SourceNegativeInputSelect Function

Configures the Comparator module to use the selected input as a negative reference.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_SourceNegativeInputSelect(CMP_MODULE_ID index, CMP_CVREF_VOLTAGE_SOURCE_NEG_REFERENCE
negInput);
```

Returns

None.

Description

This function configures the Comparator module to use the selected input as a negative reference for the voltage source.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability. For such devices, selecting the positive source will automatically select the negative input.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_SourceNegativeInputSelect ( CMP_ID_1, CMP_CVREF_VOLTAGE_SOURCE_NEG_REF_GROUND );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
negInput	Select the voltage source negative reference from CMP_CVREF_VOLTAGE_SOURCE_NEG_REFERENCE

Function

void PLIB_CMP_CVREF_SourceNegativeInputSelect (CMP_MODULE_ID index,
CMP_CVREF_VOLTAGE_SOURCE_NEG_REFERENCE negInput)

PLIB_CMP_CVREF_SourceVoltageSelect Function

Connects the Comparator module to the selected voltage source.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_SourceVoltageSelect(CMP_MODULE_ID index, CMP_CVREF_VOLTAGE_SOURCE source);
```

Returns

None.

Description

This function connects the Comparator module to the selected voltage source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFVoltageRangeSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_SourceVoltageSelect ( CMP_ID_1, CMP_CVREF_VOLTAGE_SOURCE_INTERNAL );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Select the voltage source from CMP_CVREF_VOLTAGE_SOURCE

Function

void PLIB_CMP_CVREF_SourceVoltageSelect (CMP_MODULE_ID index, CMP_CVREF_VOLTAGE_SOURCE source)

PLIB_CMP_CVREF_ValueSelect Function

Selects the voltage reference value.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_ValueSelect(CMP_MODULE_ID index, CMP_CVREF_VALUE value);
```

Returns

None.

Description

This function selects the voltage reference value. This value decides how many resistance units will be added and therefore, decides the output voltage.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFValueSelect in your application to determine whether this feature is available.

Preconditions

Determine the correct value that should be passed.

Example

```
PLIB_CMP_CVREF_ValueSelect ( CMP_ID_1,  CMP_CVREF_VALUE_13 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	Select value from CMP_CVREF_VALUE

Function

void PLIB_CMP_CVREF_ValueSelect (CMP_MODULE_ID index,
CMP_CVREF_VALUE value);

PLIB_CMP_CVREF_WideRangeDisable Function

Disables the wide range.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_WideRangeDisable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the wide range for reference voltage. The range of possible voltages will become narrower, and finer voltage options can be achieved in this case.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFWideRangeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_WideRangeDisable ( CMP_ID_1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_CVREF_WideRangeDisable (CMP_MODULE_ID index)

PLIB_CMP_CVREF_WideRangeEnable Function

Enables the wide range.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_WideRangeEnable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the wide range for reference voltage. The voltage range starts from zero if the wide range is selected.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFWideRangeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_WideRangeEnable(CMP_ID_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_CVREF_WideRangeEnable (CMP_MODULE_ID index)

PLIB_CMP_CVREF_WideRangeIsEnabled Function

Returns whether the wide range is selected for the reference voltage.

File

plib_cmp.h

C

```
bool PLIB_CMP_CVREF_WideRangeIsEnabled(CMP_MODULE_ID index);
```

Returns

- true = The wide range is enabled
- false = The wide range is not enabled

Description

This function returns whether the wide range is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFWideRangeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool range;

range = PLIB_CMP_CVREF_WideRangeIsEnabled ( MY_CMP_CVREF_ID );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_CMP_CVREF_WideRangeIsEnabled (CMP_MODULE_ID index);

PLIB_CMP_CVREF_Disable Function

Disables the voltage reference of the Comparator module.

File

plib_cmp.h

C

```
void PLIB_CMP_CVREF_Disable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the voltage reference of the Comparator module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_CVREF_ExistsCVREFEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CMP_CVREF_Disable ( CMP_ID_1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_CVREF_Disable (CMP_MODULE_ID index)

PLIB_CMP_Disable Function

Disables the Comparator module.

File

plib_cmp.h

C

```
void PLIB_CMP_Disable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the selected Comparator module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_Disable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_Disable (CMP_MODULE_ID index)

PLIB_CMP_Enable Function

Enables the Comparator module.

File

plib_cmp.h

C

```
void PLIB_CMP_Enable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the selected Comparator module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_Enable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_Enable (CMP_MODULE_ID index)

PLIB_CMP_InterruptEventSelect Function

Comparator interrupt event select.

File

plib_cmp.h

C

```
void PLIB_CMP_InterruptEventSelect(CMP_MODULE_ID index, CMP_INTERRUPT_EVENT event);
```

Returns

None.

Description

This function will select when the Comparator interrupt should occur.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsInterruptEventSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
// CMP_INTERRUPT_EVENT - CMP_LOW_TO_HIGH
PLIB_CMP_InterruptEventSelect ( MY_CMP_INSTANCE,
                        CMP_INTERRUPT_GENERATION_LOW_TO_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
event	One of the possible values from CMP_INTERRUPT_EVENT

Function

void PLIB_CMP_InterruptEventSelect (CMP_MODULE_ID index, CMP_INTERRUPT_EVENT event)

PLIB_CMP_InvertingInputChannelSelect Function

Comparator inverting input channel select.

File

plib_cmp.h

C

```
void PLIB_CMP_InvertingInputChannelSelect(CMP_MODULE_ID index, CMP_INVERTING_INPUT channel);
```

Returns

None.

Description

This function will select the inverting input channels for the Comparator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_CMP_ExistsInvertingInputSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
 PLIB_CMP_InvertingInputChannelSelect ( MY_CMP_INSTANCE,
                            CMP_INVERTING_INPUT_IVREF );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	One of the possible values from CMP_INVERTING_INPUT

Function

void PLIB_CMP_InvertingInputChannelSelect (CMP_MODULE_ID index,
CMP_INVERTING_INPUT channel)

PLIB_CMP_NonInvertingInputChannelSelect Function

Comparator input channel select.

File

plib_cmp.h

C

```
void PLIB_CMP_NonInvertingInputChannelSelect(CMP_MODULE_ID index, CMP_NON_INVERTING_INPUT input);
```

Returns

None.

Description

This function will select the non-inverting input channels for the Comparator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsNonInvertingInputSelect in your application to determine whether this feature is available.s

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_NonInvertingInputChannelSelect ( MY_CMP_INSTANCE, CMP_NON_INVERTING_INPUT_CVREF );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	One of the possible values from CMP_NON_INVERTING_INPUT

Function

void PLIB_CMP_NonInvertingInputChannelSelect (CMP_MODULE_ID index,
CMP_NON_INVERTING_INPUT input)

PLIB_CMP_OutputDisable Function

Disables the Comparator output.

File

plib_cmp.h

C

```
void PLIB_CMP_OutputDisable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the Comparator output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsOutputEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputDisable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_OutputDisable (CMP_MODULE_ID index)

PLIB_CMP_OutputEnable Function

Enables the Comparator output.

File

plib_cmp.h

C

```
void PLIB_CMP_OutputEnable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the Comparator output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsOutputEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputEnable( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_OutputEnable (CMP_MODULE_ID index)

PLIB_CMP_OutputStatusGet Function

Comparator output status.

File

plib_cmp.h

C

```
bool PLIB_CMP_OutputStatusGet(CMP_MODULE_ID index);
```

Returns

- true - The status flag is set
- false - The status flag is clear

Description

This function will return the current status of the Comparator output.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1
bool cmp_status;

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
cmp_status=PLIB_CMP_OutputStatusGet ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_CMP_OutputStatusGet (CMP_MODULE_ID index)

PLIB_CMP_OutputInvertDisable Function

Comparator output is non-inverted.

File

plib_cmp.h

C

```
void PLIB_CMP_OutputInvertDisable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function will select the non-inverted comparator output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsOutputEnableControlExistsInvertOutputControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputInvertDisable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_OutputInvertDisable (CMP_MODULE_ID index)

PLIB_CMP_OutputInvertEnable Function

Comparator output is inverted.

File

plib_cmp.h

C

```
void PLIB_CMP_OutputInvertEnable(CMP_MODULE_ID index);
```

Returns

None.

Description

Calling this function will set the comparator to make its output inverted.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsOutputEnableControlExistsInvertOutputControl in your application to determine whether this feature is available.

Setting this bit will invert the signal to the comparator interrupt generator as well. This will result in an interrupt being generated on the opposite edge from the one selected by PLIB_CMP_InterruptEventSelect function.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputDisable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_OutputInvertEnable (CMP_MODULE_ID index)

PLIB_CMP_OutputLogicHigh Function

Comparator output bit will give a 'logic high' on satisfying the input condition.

File

plib_cmp.h

C

```
void PLIB_CMP_OutputLogicHigh(CMP_MODULE_ID index);
```

Returns

None.

Description

Calling this API will set the Comparator module to output a 'logic high' on satisfying the input condition.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsOutputLevelControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputLogicHigh ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_OutputLogicHigh (CMP_MODULE_ID index)

PLIB_CMP_OutputLogicLow Function

Comparator will be set to give 'logic low' on satisfying the input condition.

File

plib_cmp.h

C

```
void PLIB_CMP_OutputLogicLow(CMP_MODULE_ID index);
```

Returns

None.

Description

This function will set the Comparator to give 'logic low' on satisfying the input condition.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsOutputLevelControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_OutputLogicLow( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_OutputLogicLow (CMP_MODULE_ID index)

b) Comparator Feature Configuration Functions

PLIB_CMP_StopInIdleModeDisable Function

Disables Stop in Idle mode.

File

plib_cmp.h

C

```
void PLIB_CMP_StopInIdleModeDisable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function will continue operation of all enabled comparators when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_CMP_INSTANCE   CMP_ID_1

// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_StopInIdleModeDisable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_StopInIdleModeDisable (CMP_MODULE_ID index)

PLIB_CMP_StopInIdleModeEnable Function

Enables Stop in Idle mode.

File

plib_cmp.h

C

```
void PLIB_CMP_StopInIdleModeEnable(CMP_MODULE_ID index);
```

Returns

None.

Description

This function will discontinue operation of all comparators when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CMP_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_CMP_INSTANCE, is the Comparator instance selected for use by the
// application developer.
PLIB_CMP_StopInIdleModeEnable ( MY_CMP_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_CMP_StopInIdleModeEnable (CMP_MODULE_ID index)

c) Feature Existence Functions

PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect Function

Identifies whether the CVREFBGRefVoltageRangeSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect(CMP_MODULE_ID index);
```

Returns

- true - The CVREFBGRefVoltageRangeSelect feature is supported on the device
- false - The CVREFBGRefVoltageRangeSelect feature is not supported on the device

Description

This function identifies whether the CVREFBGRefVoltageRangeSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_CVREF_BandGapReferenceSourceSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsCVREFEnableControl Function

Identifies whether the CVREFEnableControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFEnableControl(CMP_MODULE_ID index);
```

Returns

- true - The CVREFEnableControl feature is supported on the device
- false - The CVREFEnableControl feature is not supported on the device

Description

This function identifies whether the CVREFEnableControl feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_CVREF_Enable
- PLIB_CMP_CVREF_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFEnableControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsCVREFOutputEnableControl Function

Identifies whether the CVREFOutputEnableControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFOutputEnableControl(CMP_MODULE_ID index);
```

Returns

- true - The CVREFOutputEnableControl feature is supported on the device
- false - The CVREFOutputEnableControl feature is not supported on the device

Description

This function identifies whether the CVREFOutputEnableControl feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_CVREF_OutputEnable
- PLIB_CMP_CVREF_OutputDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFOutputEnableControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsCVREFRefVoltageRangeSelect Function

Identifies whether the CVREFRefVoltageRangeSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFRefVoltageRangeSelect(CMP_MODULE_ID index);
```

Returns

- true - The CVREFRefVoltageRangeSelect feature is supported on the device
- false - The CVREFRefVoltageRangeSelect feature is not supported on the device

Description

This function identifies whether the CVREFRefVoltageRangeSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_CVREF_ReferenceVoltageSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFRefVoltageRangeSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsCVREFValueSelect Function

Identifies whether the CVREFValueSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFValueSelect(CMP_MODULE_ID index);
```

Returns

- true - The CVREFValueSelect feature is supported on the device
- false - The CVREFValueSelect feature is not supported on the device

Description

This function identifies whether the CVREFValueSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_CVREF_ValueSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFValueSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsCVREFVoltageRangeSelect Function

Identifies whether the CVREFVoltageRangeSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFVoltageRangeSelect(CMP_MODULE_ID index);
```

Returns

- true - The CVREFVoltageRangeSelect feature is supported on the device
- false - The CVREFVoltageRangeSelect feature is not supported on the device

Description

This function identifies whether the CVREFVoltageRangeSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_CVREF_SourceVoltageSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFVoltageRangeSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsCVREFWideRangeControl Function

Identifies whether the CVREFWideRangeControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsCVREFWideRangeControl(CMP_MODULE_ID index);
```

Returns

- true - The CVREFWideRangeControl feature is supported on the device
- false - The CVREFWideRangeControl feature is not supported on the device

Description

This function identifies whether the CVREFWideRangeControl feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_CVREF_WideRangeEnable
- PLIB_CMP_CVREF_WideRangeDisable
- PLIB_CMP_CVREF_WideRangeIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsCVREFWideRangeControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsEnableControl Function

Identifies whether the ComparatorEnableControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsEnableControl(CMP_MODULE_ID index);
```

Returns

- true - The ComparatorEnableControl feature is supported on the device
- false - The ComparatorEnableControl feature is not supported on the device

Description

This function identifies whether the ComparatorEnableControl feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_Enable
- PLIB_CMP_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsEnableControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsInterruptEventSelect Function

Identifies whether the InterruptEventSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsInterruptEventSelect(CMP_MODULE_ID index);
```

Returns

- true - The InterruptEventSelect feature is supported on the device
- false - The InterruptEventSelect feature is not supported on the device

Description

This function identifies whether the InterruptEventSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_InterruptEventSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsInterruptEventSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsInvertingInputSelect Function

Identifies whether the InvertingInputSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsInvertingInputSelect(CMP_MODULE_ID index);
```

Returns

- true - The InvertingInputSelect feature is supported on the device
- false - The InvertingInputSelect feature is not supported on the device

Description

This function identifies whether the InvertingInputSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_InvertingInputChannelSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsInvertingInputSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsInvertOutputControl Function

Identifies whether the InvertOutputSelectControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsInvertOutputControl(CMP_MODULE_ID index);
```

Returns

- true - The InvertOutputSelectControl feature is supported on the device
- false - The InvertOutputSelectControl feature is not supported on the device

Description

This function identifies whether the InvertOutputSelectControl feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_OutputInvertEnable
- PLIB_CMP_OutputInvertDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsInvertOutputControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsNonInvertingInputSelect Function

Identifies whether the NonInvertingInputSelect feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsNonInvertingInputSelect(CMP_MODULE_ID index);
```

Returns

- true - The NonInvertingInputSelect feature is supported on the device
- false - The NonInvertingInputSelect feature is not supported on the device

Description

This function identifies whether the NonInvertingInputSelect feature is available on the CMP module. When this function returns true, this function is supported on the device:

- PLIB_CMP_NonInvertingInputChannelSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsNonInvertingInputSelect(CMP_MODULE_ID index)

PLIB_CMP_ExistsOutputEnableControl Function

Identifies whether the ComparatorOutputEnableControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsOutputEnableControl(CMP_MODULE_ID index);
```

Returns

- true - The ComparatorOutputEnableControl feature is supported on the device
- false - The ComparatorOutputEnableControl feature is not supported on the device

Description

This function identifies whether the ComparatorOutputEnableControl feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_OutputEnable
- PLIB_CMP_OutputDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsOutputEnableControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsOutputLevelControl Function

Identifies whether the OutputLevelSelectControl feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsOutputLevelControl(CMP_MODULE_ID index);
```

Returns

- true - The OutputLevelSelectControl feature is supported on the device
- false - The OutputLevelSelectControl feature is not supported on the device

Description

This function identifies whether the OutputLevelSelectControl feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_OutputLogicHigh
- PLIB_CMP_OutputLogicLow

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsOutputLevelControl(CMP_MODULE_ID index)

PLIB_CMP_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsStopInIdle(CMP_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_StopInIdleModeEnable
- PLIB_CMP_StopInIdleModeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsStopInIdle(CMP_MODULE_ID index)

PLIB_CMP_ExistsOutputStatusGet Function

Identifies whether the OutputStatusGet feature exists on the CMP module.

File

plib_cmp.h

C

```
bool PLIB_CMP_ExistsOutputStatusGet(CMP_MODULE_ID index);
```

Returns

- true - The OutputStatusGet feature is supported on the device
- false - The OutputStatusGet feature is not supported on the device

Description

This function identifies whether the OutputStatusGet feature is available on the CMP module. When this function returns true, these functions are supported on the device:

- PLIB_CMP_OutputStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CMP_ExistsOutputStatusGet(CMP_MODULE_ID index)

d) Data Types and Constants

CMP_CLOCK_DIVIDE Enumeration

Defines Comparator Filter Clock Divide.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_FILTER_CLOCK_DIVIDE_1_1,
  CMP_FILTER_CLOCK_DIVIDE_1_2,
  CMP_FILTER_CLOCK_DIVIDE_1_4,
  CMP_FILTER_CLOCK_DIVIDE_1_8,
  CMP_FILTER_CLOCK_DIVIDE_1_16,
  CMP_FILTER_CLOCK_DIVIDE_1_32,
  CMP_FILTER_CLOCK_DIVIDE_1_64,
  CMP_FILTER_CLOCK_DIVIDE_1_128
} CMP_CLOCK_DIVIDE;
```

Members

Members	Description
CMP_FILTER_CLOCK_DIVIDE_1_1	Comparator Filter Clock Division is 1:1
CMP_FILTER_CLOCK_DIVIDE_1_2	Comparator Filter Clock Division is 1:2
CMP_FILTER_CLOCK_DIVIDE_1_4	Comparator Filter Clock Division is 1:4
CMP_FILTER_CLOCK_DIVIDE_1_8	Comparator Filter Clock Division is 1:8
CMP_FILTER_CLOCK_DIVIDE_1_16	Comparator Filter Clock Division is 1:16
CMP_FILTER_CLOCK_DIVIDE_1_32	Comparator Filter Clock Division is 1:32
CMP_FILTER_CLOCK_DIVIDE_1_64	Comparator Filter Clock Division is 1:64
CMP_FILTER_CLOCK_DIVIDE_1_128	Comparator Filter Clock Division is 1:128

Description

Comparator Filter Clock Divide Select

This macro defines the Comparator filter clock divide.

CMP_CVREF_BANDGAP_SELECT Enumeration

Lists the band gap selection options.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_CVREF_BANDGAP_1_2V,
  CMP_CVREF_BANDGAP_0_6V,
  CMP_CVREF_BANDGAP_VREFPLUS
} CMP_CVREF_BANDGAP_SELECT;
```

Members

Members	Description
CMP_CVREF_BANDGAP_1_2V	Select the Band Gap Reference Source as 1.2 V
CMP_CVREF_BANDGAP_0_6V	Select the Band Gap Reference Source as 0.6 V
CMP_CVREF_BANDGAP_VREFPLUS	Select the Band Gap Reference Source as VREF

Description

CVREF band gap select enumeration

This enumeration lists the possible band gap selection options.

Remarks

None.

CMP_CVREF_REFERENCE_SELECT Enumeration

Lists the reference selection options.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_CVREF_RESISTOR_LADDER_VOLTAGE,
  CMP_CVREF_POSITIVE_REFERENCE_VOLTAGE
} CMP_CVREF_REFERENCE_SELECT;
```

Members

Members	Description
CMP_CVREF_RESISTOR_LADDER_VOLTAGE	Select the Band Gap Reference Source as 1.2 V
CMP_CVREF_POSITIVE_REFERENCE_VOLTAGE	Select the Band Gap Reference Source as 0.6 V

Description

CVREF reference select enumeration

This enumeration lists the possible reference selection options.

Remarks

None.

CMP_CVREF_VALUE Enumeration

Lists the voltage reference value selection options.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_CVREF_VALUE_0,
  CMP_CVREF_VALUE_1,
  CMP_CVREF_VALUE_2,
  CMP_CVREF_VALUE_3,
  CMP_CVREF_VALUE_4,
```

```
  CMP_CVREF_VALUE_5,
  CMP_CVREF_VALUE_6,
  CMP_CVREF_VALUE_7,
  CMP_CVREF_VALUE_8,
  CMP_CVREF_VALUE_9,
  CMP_CVREF_VALUE_10,
  CMP_CVREF_VALUE_11,
  CMP_CVREF_VALUE_12,
  CMP_CVREF_VALUE_13,
  CMP_CVREF_VALUE_14,
  CMP_CVREF_VALUE_15
} CMP_CVREF_VALUE;
```

Members

Members	Description
CMP_CVREF_VALUE_0	Voltage reference value 0
CMP_CVREF_VALUE_1	Voltage reference value 1
CMP_CVREF_VALUE_2	Voltage reference value 2
CMP_CVREF_VALUE_3	Voltage reference value 3
CMP_CVREF_VALUE_4	Voltage reference value 4
CMP_CVREF_VALUE_5	Voltage reference value 5
CMP_CVREF_VALUE_6	Voltage reference value 6
CMP_CVREF_VALUE_7	Voltage reference value 7
CMP_CVREF_VALUE_8	Voltage reference value 8
CMP_CVREF_VALUE_9	Voltage reference value 9
CMP_CVREF_VALUE_10	Voltage reference value 10
CMP_CVREF_VALUE_11	Voltage reference value 11
CMP_CVREF_VALUE_12	Voltage reference value 12
CMP_CVREF_VALUE_13	Voltage reference value 13
CMP_CVREF_VALUE_14	Voltage reference value 14
CMP_CVREF_VALUE_15	Voltage reference value 15

Description

CVREF voltage reference value selection enumeration

This enumeration lists the possible Voltage reference value selection options.

Remarks

None.

CMP_CVREF_VOLTAGE_SOURCE Enumeration

Lists the Voltage source options.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_CVREF_VOLTAGE_SOURCE_INTERNAL,
  CMP_CVREF_VOLTAGE_SOURCE_EXTERNAL
} CMP_CVREF_VOLTAGE_SOURCE;
```

Members

Members	Description
CMP_CVREF_VOLTAGE_SOURCE_INTERNAL	Select VDD/VSS source
CMP_CVREF_VOLTAGE_SOURCE_EXTERNAL	Select external voltage source

Description

CVREF Voltage source selection enumeration

This enumeration lists the possible Voltage source selection options.

Remarks

None.

CMP_FILTER_CLOCK Enumeration

Defines Comparator filter input clock

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_FILTER_CLOCK_FP,
  CMP_FILTER_CLOCK_FOSC,
  CMP_FILTER_CLOCK_SYNCO1,
  CMP_FILTER_CLOCK_T2CLK,
  CMP_FILTER_CLOCK_T3CLK,
  CMP_FILTER_CLOCK_T4CLK,
  CMP_FILTER_CLOCK_T5CLK
} CMP_FILTER_CLOCK;
```

Members

Members	Description
CMP_FILTER_CLOCK_FP	FP will be the Comparator filter input clock
CMP_FILTER_CLOCK_FOSC	FOSC will be the Comparator filter input clock
CMP_FILTER_CLOCK_SYNCO1	SYNCO1 will be the Comparator filter input clock
CMP_FILTER_CLOCK_T2CLK	T2CLK will be the Comparator filter input clock
CMP_FILTER_CLOCK_T3CLK	T3CLK will be the Comparator filter input clock
CMP_FILTER_CLOCK_T4CLK	T4CLK will be the Comparator filter input clock
CMP_FILTER_CLOCK_T5CLK	T5CLK will be the Comparator filter input clock

Description

Comparator Filter Input Clock Select

This macro defines the Comparator filter input clock

CMP_INTERRUPT_EVENT Enumeration

Defines Comparator interrupt events.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_INTERRUPT_GENERATION_DISABLED
} CMP_INTERRUPT_EVENT;
```

Members

Members	Description
CMP_INTERRUPT_GENERATION_DISABLED	Select VDD/VSS source

Description

Comparator interrupt event Select

This macro defines the Comparator interrupt events.

CMP_INVERTING_INPUT Enumeration

Defines the list of Comparator inverting Input.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_INVERTING_INPUT_EXTERNAL_PIN_B,
  CMP_INVERTING_INPUT_EXTERNAL_PIN_C,
  CMP_INVERTING_INPUT_EXTERNAL_PIN_D,
  CMP_INVERTING_INPUT_EXTERNAL_NEGATIVE_PIN,
  CMP_INVERTING_INPUT_EXTERNAL_POSITIVE_PIN,
  CMP_INVERTING_INPUT_OTHER_MODULE_EXTERNAL_POSITIVE_PIN,
  CMP_INVERTING_INPUT_IVREF,
  CMP_INPUT_C2IN_NEGATIVE,
  CMP_INPUT_C2IN_POSITIVE,
  CMP_INPUT_C1IN_POSITIVE,
  CMP_INPUT_IVREF
} CMP_INVERTING_INPUT;
```

Members

Members	Description
CMP_INVERTING_INPUT_EXTERNAL_PIN_B	Select external voltage source at pin CxINB as inverting input. If using CMP_ID_1 , this pin is C1INB
CMP_INVERTING_INPUT_EXTERNAL_PIN_C	Select external voltage source at pin CxINC as inverting input
CMP_INVERTING_INPUT_EXTERNAL_PIN_D	Select external voltage source at pin CxIND as inverting input
CMP_INVERTING_INPUT_EXTERNAL_NEGATIVE_PIN	Select external voltage source at pin CxIN- as inverting input
CMP_INVERTING_INPUT_EXTERNAL_POSITIVE_PIN	Select external voltage source at pin CxIN+ as inverting input. If using CMP_ID_1 , this pin is C1IN+
CMP_INVERTING_INPUT_OTHER_MODULE_EXTERNAL_POSITIVE_PIN	Select external voltage source at pin CyIN+ as inverting input. If using CMP_ID_1 , this pin is C2IN+. If using CMP_ID_2 , this pin is C1IN+
CMP_INVERTING_INPUT_IVREF	Select internal voltage ,VDD/VSS source as inverting input
CMP_INPUT_C2IN_NEGATIVE	The member is obsolete and maintained here only for the backward compatibility.
CMP_INPUT_C2IN_POSITIVE	The member is obsolete and maintained here only for the backward compatibility.
CMP_INPUT_C1IN_POSITIVE	The member is obsolete and maintained here only for the backward compatibility.
CMP_INPUT_IVREF	The member is obsolete and maintained here only for the backward compatibility.

Description

Comparator inverting input channel select

This macro defines the super set of Comparator inverting Inputs. Not all options will be available on all microcontrollers. Refer to the processor header for the specific controller in use to determine which options are supported.

CMP_MASK_A Enumeration

Defines Comparator Mask A Input.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_MASK_A_PWM1L,
  CMP_MASK_A_PWM1H,
  CMP_MASK_A_PWM2L,
  CMP_MASK_A_PWM2H,
  CMP_MASK_A_PWM3L,
  CMP_MASK_A_PWM3H,
  CMP_MASK_A_PTGO18,
  CMP_MASK_A_PTGO19,
  CMP_MASK_A_FLT2,
  CMP_MASK_A_FLT4
} CMP_MASK_A;
```

Members

Members	Description
CMP_MASK_A_PWM1L	PWM1L will be the Comparator Mask A input
CMP_MASK_A_PWM1H	PWM1H will be the Comparator Mask A input
CMP_MASK_A_PWM2L	PWM2L will be the Comparator Mask A input
CMP_MASK_A_PWM2H	PWM2H will be the Comparator Mask A input
CMP_MASK_A_PWM3L	PWM3L will be the Comparator Mask A input
CMP_MASK_A_PWM3H	PWM3H will be the Comparator Mask A input
CMP_MASK_A_PTGO18	PTGO18 will be the Comparator Mask A input
CMP_MASK_A_PTGO19	PTGO19 will be the Comparator Mask A input
CMP_MASK_A_FLT2	FLT2 will be the Comparator Mask A input
CMP_MASK_A_FLT4	FLT4 will be the Comparator Mask A input

Description

Comparator Mask A Input Select

This macro defines the Comparator Mask A Input.

CMP_MASK_B Enumeration

Defines Comparator Mask B Input.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_MASK_B_PWM1L,
  CMP_MASK_B_PWM1H,
  CMP_MASK_B_PWM2L,
  CMP_MASK_B_PWM2H,
  CMP_MASK_B_PWM3L,
  CMP_MASK_B_PWM3H,
  CMP_MASK_B_PTGO18,
  CMP_MASK_B_PTGO19,
  CMP_MASK_B_FLT2,
  CMP_MASK_B_FLT4
} CMP_MASK_B;
```

Members

Members	Description
CMP_MASK_B_PWM1L	PWM1L will be the Comparator Mask B input
CMP_MASK_B_PWM1H	PWM1H will be the Comparator Mask B input
CMP_MASK_B_PWM2L	PWM2L will be the Comparator Mask B input
CMP_MASK_B_PWM2H	PWM2H will be the Comparator Mask B input
CMP_MASK_B_PWM3L	PWM3L will be the Comparator Mask B input
CMP_MASK_B_PWM3H	PWM3H will be the Comparator Mask B input
CMP_MASK_B_PTGO18	PTGO18 will be the Comparator Mask B input
CMP_MASK_B_PTGO19	PTGO19 will be the Comparator Mask B input
CMP_MASK_B_FLT2	FLT2 will be the Comparator Mask B input
CMP_MASK_B_FLT4	FLT4 will be the Comparator Mask B input

Description

Comparator Mask B Input Select

This macro defines the Comparator Mask B Input.

CMP_MASK_C Enumeration

Defines Comparator Mask C Input.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_MASK_C_PWM1L,
  CMP_MASK_C_PWM1H,
  CMP_MASK_C_PWM2L,
  CMP_MASK_C_PWM2H,
  CMP_MASK_C_PWM3L,
  CMP_MASK_C_PWM3H,
  CMP_MASK_C_PTGO18,
  CMP_MASK_C_PTGO19,
  CMP_MASK_C_FLT2,
  CMP_MASK_C_FLT4
} CMP_MASK_C;
```

Members

Members	Description
CMP_MASK_C_PWM1L	PWM1L will be the Comparator mask C input
CMP_MASK_C_PWM1H	PWM1H will be the Comparator mask C input
CMP_MASK_C_PWM2L	PWM2L will be the Comparator mask C input
CMP_MASK_C_PWM2H	PWM2H will be the Comparator mask C input
CMP_MASK_C_PWM3L	PWM3L will be the Comparator mask C input
CMP_MASK_C_PWM3H	PWM3H will be the Comparator mask C input
CMP_MASK_C_PTGO18	PTGO18 will be the Comparator mask C input
CMP_MASK_C_PTGO19	PTGO19 will be the Comparator mask C input
CMP_MASK_C_FLT2	FLT2 will be the Comparator mask C input
CMP_MASK_C_FLT4	FLT4 will be the Comparator mask C input

Description

Comparator Mask C Input Select

This macro defines the Comparator Mask C Input.

CMP_MODULE_ID Enumeration

Identifies the Comparator modules supported

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_ID_1,
  CMP_ID_2,
  CMP_ID_3
} CMP_MODULE_ID;
```

Members

Members	Description
CMP_ID_1	Comparator Module 1 ID
CMP_ID_2	Comparator Module 2 ID
CMP_ID_3	Comparator Module 3 ID

Description

Comparator Module ID

This enumeration identifies the Comparator modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on the Microchip microcontrollers.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

CMP_NON_INVERTING_INPUT Enumeration

Defines the list of Comparator non-inverting Input.

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_NON_INVERTING_INPUT_EXTERNAL_PIN_A,
  CMP_NON_INVERTING_INPUT_EXTERNAL_POSITIVE_PIN,
  CMP_NON_INVERTING_INPUT_CVREF,
  CMP_INPUT_C2IN_POSITIVE,
  CMP_INPUT_INTERNAL_CVREF
} CMP_NON_INVERTING_INPUT;
```

Members

Members	Description
CMP_NON_INVERTING_INPUT_EXTERNAL_PIN_A	Select external voltage source at pin CxINA as non-inverting input
CMP_NON_INVERTING_INPUT_EXTERNAL_POSITIVE_PIN	Select external voltage source at pin CxIN+ as non-inverting input
CMP_NON_INVERTING_INPUT_CVREF	Select internal voltage source CVREF as non-inverting input
CMP_INPUT_C2IN_POSITIVE	The member is obsolete and maintained here only for the backward compatibility.
CMP_INPUT_INTERNAL_CVREF	The member is obsolete and maintained here only for the backward compatibility.

Description

Comparator Non inverting input channel select

This macro defines the super set of Comparator non-inverting Inputs. Not all options will be available on all microcontrollers. Refer to the processor header for the specific controller in use to determine which options are supported.

CMP_SPEED_POWER Enumeration

Defines the Speed/Power of the Comparator

File

help_plib_cmp.h

C

```
typedef enum {
  CMP_SPEED_POWER_LOWER,
  CMP_SPEED_POWER_HIGHER
} CMP_SPEED_POWER;
```

Members

Members	Description
CMP_SPEED_POWER_LOWER	Comparator low-power, low-speed
CMP_SPEED_POWER_HIGHER	Comparator normal power, higher speed

Description

Comparator Speed/Power Select

This macro defines the Speed/Power of the Comparator.

Files

Files

Name	Description
plib_cmp.h	Comparator Peripheral Library Interface Header for Comparator module definitions.
help_plib_cmp.h	

Description

This section lists the source and header files used by the library.

plib_cmp.h

Comparator Peripheral Library Interface Header for Comparator module definitions.

Functions

	Name	Description
	PLIB_CMP_CVREF_BandGapReferenceSourceSelect	Selects the band gap reference voltage source.
	PLIB_CMP_CVREF_Disable	Disables the voltage reference of the Comparator module.
	PLIB_CMP_CVREF_Enable	Enables the voltage reference of the Comparator module.
	PLIB_CMP_CVREF_OutputDisable	Disables the output voltage.
	PLIB_CMP_CVREF_OutputEnable	Enables the voltage output.
	PLIB_CMP_CVREF_ReferenceVoltageSelect	Selects the voltage reference value, CVref.
	PLIB_CMP_CVREF_SourceNegativeInputSelect	Configures the Comparator module to use the selected input as a negative reference.
	PLIB_CMP_CVREF_SourceVoltageSelect	Connects the Comparator module to the selected voltage source.
	PLIB_CMP_CVREF_ValueSelect	Selects the voltage reference value.
	PLIB_CMP_CVREF_WideRangeDisable	Disables the wide range.
	PLIB_CMP_CVREF_WideRangeEnable	Enables the wide range.
	PLIB_CMP_CVREF_WideRangeIsEnabled	Returns whether the wide range is selected for the reference voltage.
	PLIB_CMP_Disable	Disables the Comparator module.
	PLIB_CMP_Enable	Enables the Comparator module.
	PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect	Identifies whether the CVREFBGRefVoltageRangeSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFEnableControl	Identifies whether the CVREFEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFOutputEnableControl	Identifies whether the CVREFOutputEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFRefVoltageRangeSelect	Identifies whether the CVREFRefVoltageRangeSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFValueSelect	Identifies whether the CVREFValueSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFVoltageRangeSelect	Identifies whether the CVREFVoltageRangeSelect feature exists on the CMP module.
	PLIB_CMP_ExistsCVREFWideRangeControl	Identifies whether the CVREFWideRangeControl feature exists on the CMP module.
	PLIB_CMP_ExistsEnableControl	Identifies whether the ComparatorEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsInterruptEventSelect	Identifies whether the InterruptEventSelect feature exists on the CMP module.
	PLIB_CMP_ExistsInvertingInputSelect	Identifies whether the InvertingInputSelect feature exists on the CMP module.
	PLIB_CMP_ExistsInvertOutputControl	Identifies whether the InvertOutputSelectControl feature exists on the CMP module.
	PLIB_CMP_ExistsNonInvertingInputSelect	Identifies whether the NonInvertingInputSelect feature exists on the CMP module.
	PLIB_CMP_ExistsOutputEnableControl	Identifies whether the ComparatorOutputEnableControl feature exists on the CMP module.
	PLIB_CMP_ExistsOutputLevelControl	Identifies whether the OutputLevelSelectControl feature exists on the CMP module.
	PLIB_CMP_ExistsOutputStatusGet	Identifies whether the OutputStatusGet feature exists on the CMP module.
	PLIB_CMP_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CMP module.
	PLIB_CMP_InterruptEventSelect	Comparator interrupt event select.
	PLIB_CMP_InvertingInputChannelSelect	Comparator inverting input channel select.
	PLIB_CMP_NonInvertingInputChannelSelect	Comparator input channel select.
	PLIB_CMP_OutputDisable	Disables the Comparator output.
	PLIB_CMP_OutputEnable	Enables the Comparator output.

	PLIB_CMP_OutputInvertDisable	Comparator output is non-inverted.
	PLIB_CMP_OutputInvertEnable	Comparator output is inverted.
	PLIB_CMP_OutputLogicHigh	Comparator output bit will give a 'logic high' on satisfying the input condition.
	PLIB_CMP_OutputLogicLow	Comparator will be set to give 'logic low' on satisfying the input condition.
	PLIB_CMP_OutputStatusGet	Comparator output status.
	PLIB_CMP_StopInIdleModeDisable	Disables Stop in Idle mode.
	PLIB_CMP_StopInIdleModeEnable	Enables Stop in Idle mode.

Description

Comparator Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Comparator Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the Comparator peripheral.

File Name

plib_cmp.h

Company

Microchip Technology Inc.

help_plib_cmp.h

Enumerations

	Name	Description
	CMP_CLOCK_DIVIDE	Defines Comparator Filter Clock Divide.
	CMP_CVREF_BANDGAP_SELECT	Lists the band gap selection options.
	CMP_CVREF_REFERENCE_SELECT	Lists the reference selection options.
	CMP_CVREF_VALUE	Lists the voltage reference value selection options.
	CMP_CVREF_VOLTAGE_SOURCE	Lists the Voltage source options.
	CMP_FILTER_CLOCK	Defines Comparator filter input clock
	CMP_INTERRUPT_EVENT	Defines Comparator interrupt events.
	CMP_INVERTING_INPUT	Defines the list of Comparator inverting Input.
	CMP_MASK_A	Defines Comparator Mask A Input.
	CMP_MASK_B	Defines Comparator Mask B Input.
	CMP_MASK_C	Defines Comparator Mask C Input.
	CMP_MODULE_ID	Identifies the Comparator modules supported
	CMP_NON_INVERTING_INPUT	Defines the list of Comparator non-inverting Input.
	CMP_SPEED_POWER	Defines the Speed/Power of the Comparator

CTMU Peripheral Library

This section describes the Charge Time Measurement Unit (CTMU) Library.

Introduction

Charge Time Measurement Unit (CTMU) for Microchip Microcontrollers

This library provides a low-level abstraction of Charge Time Measurement Unit (CTMU) features. It provides a convenient C language interface, allowing the easy construction of CTMU code that can function across the full range of Microchip devices with CTMU modules.

Description

The CTMU is a flexible analog module that has a configurable current source with a digital configuration circuit built around it. Using its high precision current source the CTMU can support:

- Pulse timing measurements - pulse to pulse, high pulse width, and low pulse width - using an on-chip ADC
- Pulse regeneration with delays
- Temperature measurement (on devices with a temperature diode) using an on-chip ADC
- Capacitance measurement - capacitive touch, and capacitive sensor support - using an on-chip ADC

The CTMU module has the following features:

- On-chip precision current source with four ranges and trimmable output
- Current source controllable by software or external triggers
- Up to 32 input channels supported (ADC inputs)
- Up to 16 sources for triggering
- Level sensitive triggering with edge triggering available on some devices
- Control of triggering polarity and direction
- Two trigger channels with control of trigger sequence
- Time measurement resolution of 1 nanosecond
- Provides ADC capture strobe to automatically initiate ADC voltage measurements

Using the Library

This topic describes the basic architecture of the CTMU Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_can.h`

The interface to the CTMU Peripheral library is defined in the `plib_can.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the CTMU Peripheral library must include `peripheral.h`.

Library File:

The CTMU Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

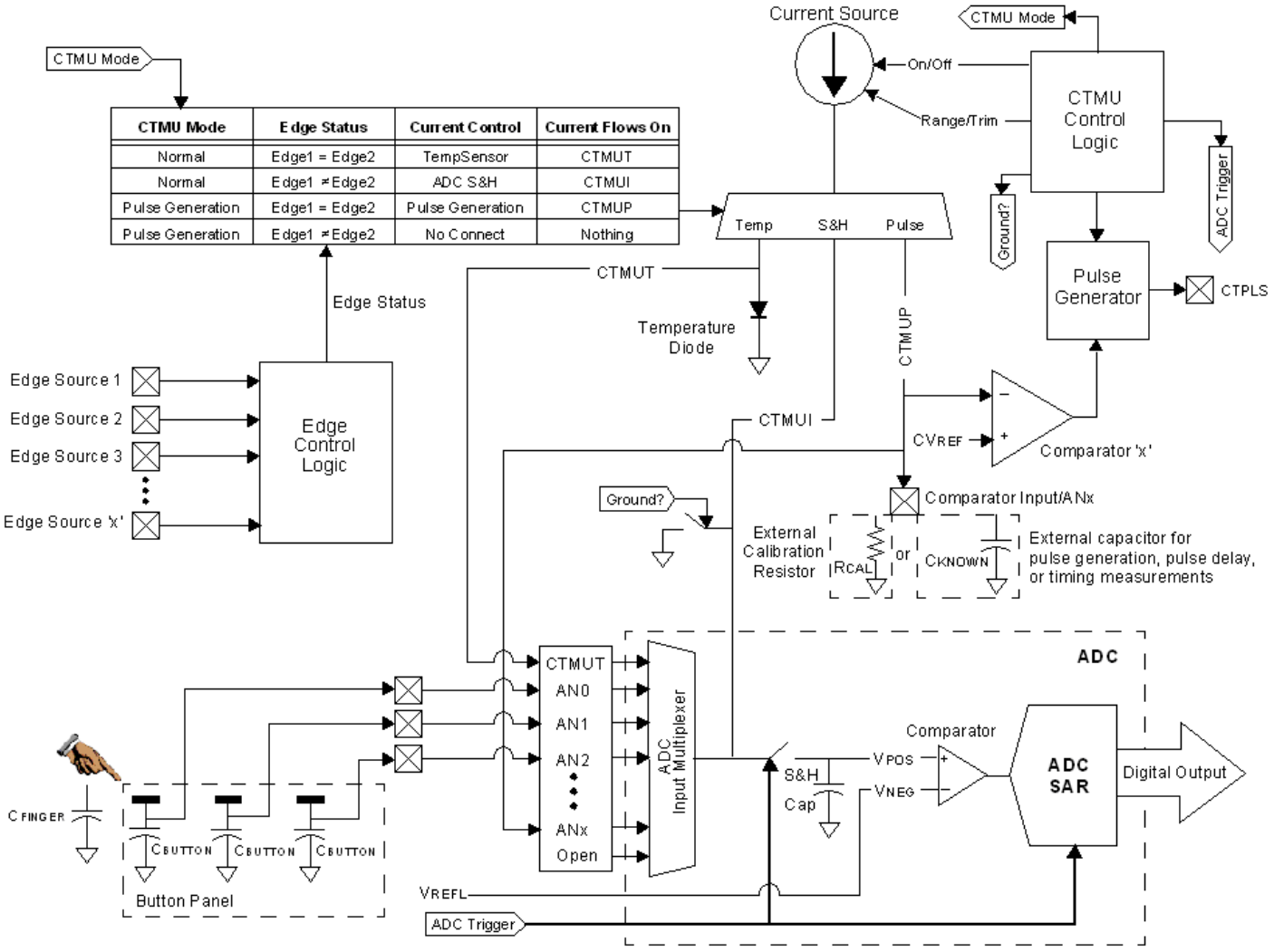
Please refer to the What is MPLAB Harmony? section for how the peripheral interacts with the framework.

Hardware Abstraction Model

This library provides a low-level abstraction of the CTMU module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The following figure shows a block diagram of the CTMU module, focusing on CTMU function rather than the bits in the interface:



CTMU Mode	Edge Status	Current Control	Current Flows On
Normal	Edge1 = Edge2	TempSensor	CTMUT
Normal	Edge1 ≠ Edge2	ADC S&H	CTMUI
Pulse Generation	Edge1 = Edge2	Pulse Generation	CTMUP
Pulse Generation	Edge1 ≠ Edge2	No Connect	Nothing

The CTMU is built around a high precision current source (i.e., charge pump) and is closely integrated with the device's on-chip Analog-to-Digital Converter (ADC) module and an on-chip Comparator module. The ADC converts the voltage carried by the Sample and Hold (S&H) capacitor into a digital number representing ADC counts. These counts can be converted into a voltage and the voltage used by the embedded application. The Comparator module associated with the CTMU can compare the voltage on an external capacitor with a known voltage and output when the voltages match. See the Example Applications section, for details of these applications.

In the previous block diagram, an analog multiplexer (MUX) controls where the charge pump sends current. This multiplexer is controlled by the CTMU's mode (Normal versus Pulse Generation) and edge status, as shown by the look-up table. In Normal mode, current is switched between the on-chip temperature sensor (**CTMUT** path) and the ADC's S&H capacitor (**CTMUI** path). In the Pulse Generation mode, current is switched on and off over the **CTMUP** path, with the "on" current going to the negative input of the associated comparator and its special input pin, labeled "Comparator Input/ANx" in the diagram.

External triggers can be used to start and stop the charge pump using up to 16 edge sources. The charge pump is also controlled by manipulating two edge status bits in software.

The block labeled Edge Control Logic determines the edge status based on the selected edge sources and edge control settings. Basically, current is off if the edges are equal, and on if the edges are unequal. In other words, Current (on/off) = Edge 1 XOR Edge 2. Sources can trigger an edge in the control logic based on levels or transitions and can trigger going high-low or low-high. The order of edge events can be controlled, with Edge 1 always occurring before Edge 2, or edge events can occur in any order.

The *Ground?* control signal created by the CTMU Control Logic block enables the grounding of the **CTMUI** path, connecting the charge pump and the ADC's S&H capacitor. This is done before starting any capacitance or timing measurement to make sure there is no residual charge on the ADC's Sample and Hold capacitor and on any external capacitor, such as a button. To discharge the ADC's Sample and Hold capacitor the ADC must be sampling the input voltage (i.e., the switch before the S&H capacitor is closed). This is normally done by manually starting ADC sampling using the "Sampling" bit in one the ADC's control registers.

In Normal mode, current flows from the charge pump to the ADC via the **CTMUI** path. Current flows to external capacitors (such as button capacitors) through the ADC's analog input multiplexer.

The ADC input multiplexer has a special input signal, named "Open", which does not connect anything to the ADC's input. This minimizes stray capacitance and provides the smallest internal capacitance for very fast time measurements.

In Pulse Generation mode, current flows directly to a special comparator/analog input pin via the **CTMUP** path, bypassing the ADC's input

multiplexer and its approximately 2500 ohm series resistance. This extra series resistance is why the charge pump should only be calibrated in Pulse Generation mode, as shown in the block diagram. Except in the lowest current settings, this series resistance causes an error that is too large in the overall calibration resistance value to provide useable charge pump current measurements.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the CTMU module.

Library Interface Section	Description
CTMU Control Functions	These functions can be used to enable, disable, and set CTMU features.
Edge Control and Status Functions	These functions can be used to configure CTMU edge control and status.
Feature Existence Functions	These functions can be used to determine whether or not a particular feature is available on the device.

How the Library Works

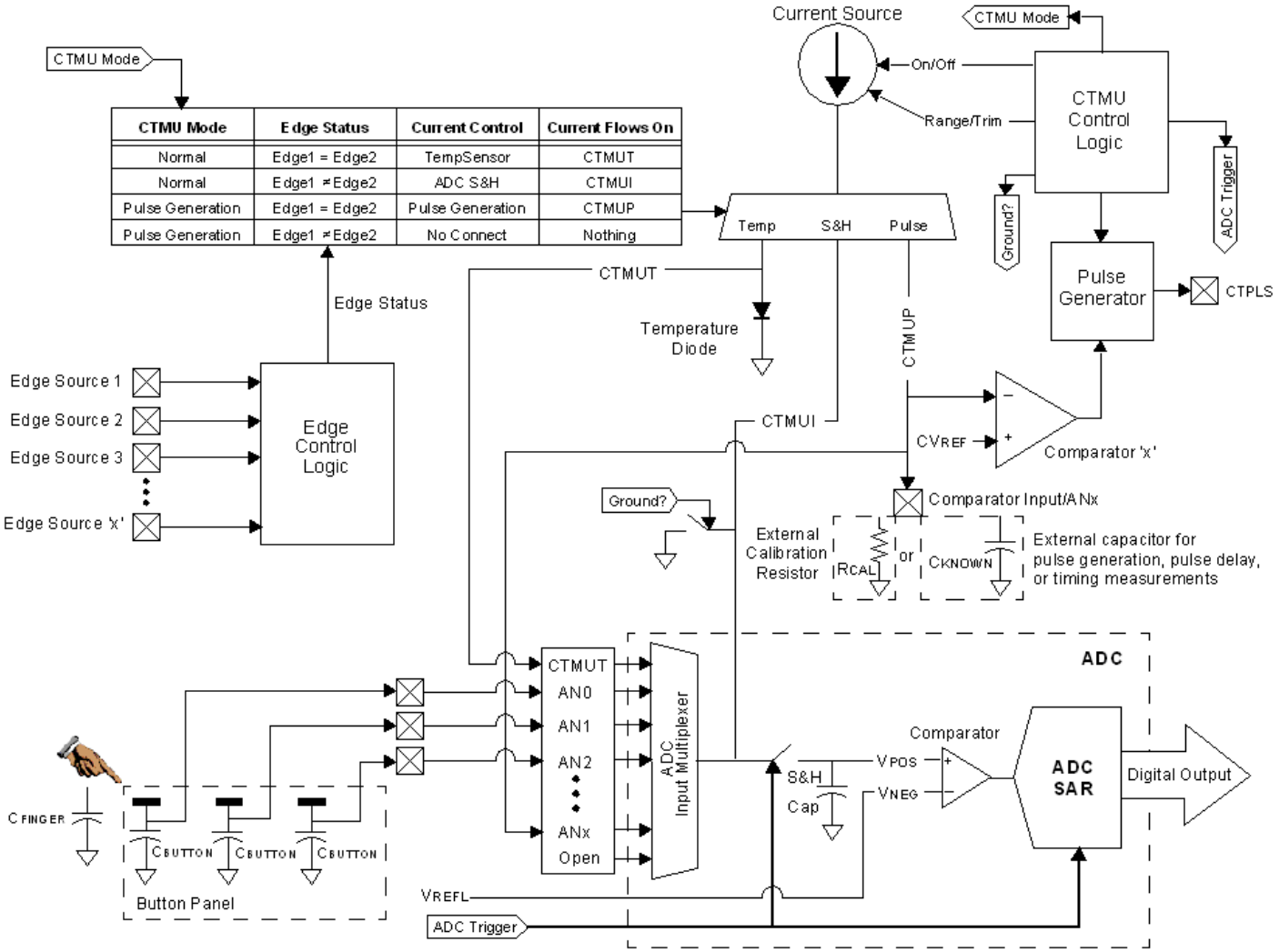
This section describes how the CTMU Peripheral Library works.

Description

The CTMU is a simple analog module, built around a high precision charge pump.

The focus of the CTMU peripheral library is controlling the output of the charge pump and interfacing with the CTMU's associated on-chip ADC.

The block diagram of the CTMU module is repeated here for reference. Next, we will discuss library primitives by functional group, starting with simple things such as turning the module on or off, and then proceeding to more complicated topics such as edge control.



CTMU On/Off and Reset

These features are controlled by these functions:

```
void PLIB_CTMU_Enable ( CTMU_MODULE_ID ctmu_id )
void PLIB_CTMU_Disable ( CTMU_MODULE_ID ctmu_id )
void PLIB_CTMU_PORDefaultsRestore ( CTMU_MODULE_ID ctmu_id )
```

The parameter ctmu_id identifies which CTMU module is referenced. Since all currently available devices only have a single CTMU, this parameter is primarily in support of future expansion.

Operation in Idle Mode

The CTMU can stop operating in Idle mode or operate when the chip is in Idle mode depending on:

```
void PLIB_CTMU_StopInIdleEnable( CTMU_MODULE_ID ctmu_id )
void PLIB_CTMU_StopInIdleDisable( CTMU_MODULE_ID ctmu_id )
```

Current Source

Charge pump output is controlled by a current range, and then a trim off of the nominal current of each of four current ranges.

These functions control current range and current trim:

```
void PLIB_CTMU_CurrentTrimSet( CTMU_MODULE_ID ctmu_id, signed short current_trim )
void PLIB_CTMU_CurrentRangeSet( CTMU_MODULE_ID ctmu_id, CTMU_CURRENT_RANGE current_range )
```

Current is shut off by grounding the charge pump or manipulating the status of two trigger edges, as shown previously. Edge status can be directly controlled in software or determined by two input signals.

The charge pump is grounded or not grounded using:

```
void PLIB_CTMU_CurrentSourceGrounded( CTMU_MODULE_ID ctmu_id )
void PLIB_CTMU_CurrentSourceNotGrounded( CTMU_MODULE_ID ctmu_id )
```

Edge Control

The CTMU can use from four to 16 different sources to provide on and off triggers for the charge pump.

How a source changes edge status is completely controlled. Edges can trigger on source signals going high-to-low or low-to-high. On some devices, source signal transitions can also trigger an edge. On these devices triggers can occur on for both high-low transitions and low-high transitions.

The CTMU can require an edge sequence, with Edge 1 triggering before Edge 2 triggers, or can respond to either Edge 1 or Edge 2 in any order.

After edge sources have caused the charge pump to start and then stop, edge status must be reset by software before the CTMU can respond to edge sources again.

All CTMU edge control functions start with "PLIB_CTMU_Edge". Each type of control function is discussed in the following section.

Control of the CTMU charge pump by the edges is enabled/disabled by:

```
 void PLIB_CTMU_EdgeEnable(CTMU_MODULE_ID ctmu_id)
 void PLIB_CTMU_EdgeDisable(CTMU_MODULE_ID ctmu_id)
```

Even with edges disabled (the default setting after POR) the charge pump can be controlled in software by setting and clearing the module's edge status bits using:

```
 void PLIB_CTMU_EdgeStatusGotEdgeSet(CTMU_MODULE_ID ctmu_id, CTMU_EDGE_NUM edge_number)
 void PLIB_CTMU_EdgeStatusNoEdgeSet(CTMU_MODULE_ID ctmu_id, CTMU_EDGE_NUM edge_number)
```

Since there are two edges and the charge pump is turned on when the edge status is unequal, there are two ways of turning the charge pump on/off in software. The first way is to leave one edge set to its POR default (off) and to manipulate the other status. Typically, Edge 1 is modified and Edge 2 is untouched, as shown in this code example:

```
PLIB_CTMU_EdgeStatusGotEdgeSet( CTMU_ID_1, CTMU_Edge1 ); // Start charging button
// TO DO: Charge Delay
PLIB_CTMU_EdgeStatusNoEdgeSet( CTMU_ID_1, CTMU_Edge1 ); // Stop charging button
```

The other way of turning the charge pump on/off in software mimics what happens when using external sources for edge signals. One edge is set to turn the charge pump on and then the other edge is set to turn the charge pump off:

```
PLIB_CTMU_EdgeStatusGotEdgeSet( CTMU_ID_1, CTMU_Edge1 ); // Start charging button
// TO DO: Charge Delay
PLIB_CTMU_EdgeStatusGotEdgeSet( CTMU_ID_1, CTMU_Edge2 ); // Stop charging button
```

To reset edge status for the next measurement you must clear both edges at the same time:

```
 void PLIB_CTMU_EdgeStatusNoEdgesAllSet(CTMU_MODULE_ID ctmu_id)
```

(To use two calls to PLIB_CTMU_EdgeStatusSetNoEdge would cause the charge pump to pulse between the first call and the second call because in between the status would not be equal, thus enabling the charge pump again.)

Edges can fire based on edges (i.e., transitions) or levels (POR default):

```
 void PLIB_CTMU_EdgeModeEdgeSensitiveSet(CTMU_MODULE_ID ctmu_id, CTMU_EDGE_NUM edge_number)
 void PLIB_CTMU_EdgeModeLevelSensitiveSet(CTMU_MODULE_ID ctmu_id, CTMU_EDGE_NUM edge_number)
```

You can also determine whether a low-to-high change triggers an edge:

```
 void PLIB_CTMU_EdgePolarityPositiveSet(CTMU_MODULE_ID ctmu_id, CTMU_EDGE_NUM edge_number)
```

or whether a high-to-low change triggers an edge:
```
void PLIB_CTMU_EdgePolarityNegativeSet(CTMU_MODULE_ID ctmu_id, CTMU_EDGE_NUM edge_number)
```
If Edge 1 turns on the charge pump and Edge 2 turns it off, you must call:
```
void PLIB_CTMU_EdgeSequenceEnable(CTMU_MODULE_ID ctmu_id)
```
in setting up the CTMU. This feature can be disabled to reinstate the POR default behavior using:
```
void PLIB_CTMU_EdgeSequenceDisable(CTMU_MODULE_ID ctmu_id)
```
With edge sequencing disabled, either edge can be used to control the charge pump.

Finally, the choice of signals for both edges is controlled by:
```
void PLIB_CTMU_EdgeSourceSelect(CTMU_MODULE_ID ctmu_id,
CTMU_EDGE_NUM edge_number,
CTMU_EDGE_SOURCE edge_source)
```

Current Output Control

As shown in the previous block diagram, where the charge pump current goes is controlled by he CTMU mode as well as the edge status.

Whether the CTMU is in capacitance/time or time pulse generation mode is controlled by:
```
void PLIB_CTMU_TimePulseGenEnable( CTMU_MODULE_ID ctmu_id )
void PLIB_CTMU_TimePulseGenDisable( CTMU_MODULE_ID ctmu_id )
```
As shown in the previous block diagram, when the CTMU is in capacitance/time (normal) mode the charge pump runs when Edge 1 Status != Edge 2 Status and stops when Edge 1 Status = Edge 2 Status. (Actually what happens is the charge pump switches from the ADC S&H capacitor to the on-chip temperature diode (if available), but the effect is the same, charge stops flowing.) In pulse generation mode, the charge pump switches to a "no connect" when Edge 1 Status = Edge 2 Status and is connected to CTMUP when Edge 1 Status != Edge 2 Status.

The CTMU charge pump can be manually turned on by the call:
```
PLIB_CTMU_EdgeStatusGotEdgeSet( CTMU_ID_1, PLIB_CTMU_Edge1 );
```
and turned off by the call:
```
PLIB_CTMU_EdgeStatusNoEdgeSet( CTMU_ID_1, PLIB_CTMU_Edge1 );
```
assuming that Edge 2 status is "NoEdge".

Interface with the ADC

Absolute and relative capacitance measurements, as well as pulse timing measurements, require using the associated on-chip ADC to measure the resulting voltage across an ADC input pin. ADC conversion can be triggered by a special ADC trigger strobe signal between the CTMU and ADC modules. The ADC trigger strobe fires when the charge pump changes from running to stopped. Temperature measurements by the ADC involve switching the ADC to a special (internal) input pin attached to the voltage diode.

The ADC trigger strobe is controlled by:
```
void PLIB_CTMU_ADCTriggerEnable( CTMU_MODULE_ID ctmu_id )
void PLIB_CTMU_ADCTriggerDisable( CTMU_MODULE_ID ctmu_id )
```

Power On Reset (POR) Status

The CTMU after a power on reset (POR) has its features configured as follows:
```
PLIB_CTMU_Disable(CTMU_ID_1);              // Turn CTMU off
PLIB_CTMU_EdgeModeLevelSensitiveSet(CTMU_ID_1,CTMU_Edge1);
PLIB_CTMU_EdgeModeLevelSensitiveSet(CTMU_ID_1,CTMU_Edge2);
PLIB_CTMU_EdgePolarityNegativeSet(CTMU_ID_1,CTMU_Edge1);
PLIB_CTMU_EdgePolarityNegativeSet(CTMU_ID_1,CTMU_Edge2);
PLIB_CTMU_EdgeSourceSelect(CTMU_ID_1,CTMU_Edge1,CTMU_EdgeSource_0);
PLIB_CTMU_EdgeSourceSelect(CTMU_ID_1,CTMU_Edge2,CTMU_EdgeSource_0);
PLIB_CTMU_EdgeStatusNoEdgesAllSet(CTMU_ID_1);
PLIB_CTMU_StopInIdleDisable(CTMU_ID_1);   // Continue CTMU operation in Idle mode
PLIB_CTMU_TimePulseGenDisable(CTMU_ID_1); // Capacitance/time measurement
PLIB_CTMU_EdgeDisable(CTMU_ID_1);         // Edges are blocked
PLIB_CTMU_EdgeSequenceDisable(CTMU_ID_1); // Edges can fire in any order
PLIB_CTMU_CurrentSourceNotGrounded(CTMU_ID_1);
PLIB_CTMU_ADCTriggerDisable(CTMU_ID_1);
PLIB_CTMU_CurrentTrimSet(CTMU_ID_1,0);
PLIB_CTMU_CurrentRangeSet(CTMU_ID_1,CTMU_CurrentRangeBase_1000xBase);
All of these functions can be accomplished by a single call to
PLIB_CTMU_PORDefaultsRestore(CTMU_ID_1);
```

Device-to-Device Variations

Some devices do not have an on-chip temperature diode.

Some devices only support edge triggering based on source signal levels rather than transitions. For these devices the function PLIB_CTMU_EdgeModeSetEdgeSensitive does not apply.

The number of possible edge sources varies from device to device. Some have only four possible sources for Edge 1 and Edge 2. Other devices support up to 16 possible sources.

Refer to the **"Charge Time Measurement Unit (CTMU)**" chapter in the specific device data sheet for more information.

CTMU Setup

This section provides the CTMU setup sequence.

Description

The following sequence is a general guideline used to initialize the CTMU module:

1. Reset the CTMU back to POR defaults. (This step isn't necessary if the CTMU is not being reconfigured.)
2. Select the current source range.
3. Adjust the current source trim.
4. Select the operating mode (Normal or Pulse Generation). (If you are using the Comparator Input/ANx pin, you must use the Pulse Generation Mode; otherwise, use the Normal mode).
5. Clear the Edge Status bits.
6. Optional setups:

- Clear the CTMU interrupt flag.
- Configure the ADC to trigger on the CTMU's strobe.
- Configure button/key circuit pins to be ADC inputs.
- Configure the ADC input multiplexer to point to the first button/key pin.
- Turn on the ADC module.

7. Turn on the CTMU module and wait 50 microseconds for the charge pump to stabilize.
8. Discharge the connected circuit by grounding the charge pump. (If using the ADC, enable ADC sampling to connect S&H capacitor to ground.)

```
// 1. Reset CTMU back to POR defaults
    PLIB_CTMU_RestorePORDefaults(CTMU_ID_1);  // POR defaults, turn CTMU off

// 2. Select the current source range.
    PLIB_CTMU_CurrentRangeSet(CTMU_ID_1,CTMU_CurrentRangeBase_Base);

// 3. Adjust the current source trim.
    PLIB_CTMU_CurrentTrimSet(CTMU_ID_1,10); // increase by 20%

// 4. Select the operating mode (Normal or Pulse Generation)
    PLIB_CTMU_TimePulseGenDisable(CTMU_ID_1);

// 5. Clear the Edge Status bits
    PLIB_CTMU_EdgeStatusSetAllNoEdges(CTMU_ID_1);

// 6. Optional setups
//  Clear CTMU interrupt flag
//  Configure ADC to trigger on CTMU strobe
//  Configure button/key circuit pins to be ADC inputs
//  Configure ADC input multiplexer to point to first button/key pin
//  Turn on the ADC

// 7. Turn on the CTMU module, wait 50 us for charge pump to stabilize
    PLIB_CTMU_Enable(CTMU_ID_1);
    // TO DO: Wait 50 microseconds

// 8. Discharge the connected circuit by grounding the charge pump.
    PLIB_CTMU_CurrentSourceGrounded(CTMU_ID_1);
    // TO DO: Enable ADC sampling
    // TO DO: Wait for discharge to occur
```

General Setup

The following sequence is a general guideline used to initialize the CTMU module:

1. Reset CTMU back to POR defaults. (This step isn't necessary if the CTMU is not being reconfigured.)
2. Select the current source range.
3. Adjust the current source trim.
4. Select the operating mode (Normal or Pulse Generation). (If you are using the Comparator Input/AN*x* pin then you must use the Pulse Generation Mode, otherwise, use the Normal mode).
5. Clear the Edge Status bits.
6. Optional setups:

- Clear CTMU interrupt flag.
- Configure ADC to trigger on CTMU strobe.
- Configure button/key circuit pins to be ADC inputs.
- Configure ADC input multiplexer to point to first button/key pin.
- Turn on the ADC module.

7. Turn on the CTMU module, wait 50 microseconds for charge pump to stabilize.
8. Discharge the connected circuit by grounding the charge pump. (If using the ADC, enable ADC sampling to connect S&H capacitor to ground.)

```
// 1. Reset CTMU back to POR defaults
    PLIB_CTMU_PORDefaultsRestore(CTMU_ID_1);  // POR defaults, turn CTMU off

// 2. Select the current source range.
    PLIB_CTMU_CurrentRangeSet(CTMU_ID_1,CTMU_CurrentRangeBase_Base);

// 3. Adjust the current source trim.
    PLIB_CTMU_CurrentTrimSet(CTMU_ID_1,10); // increase by 20%

// 4. Select the operating mode (Normal or Pulse Generation)
    PLIB_CTMU_TimePulseGenDisable(CTMU_ID_1);

// 5. Clear the Edge Status bits
    PLIB_CTMU_EdgeStatusNoEdgesAllSet(CTMU_ID_1);

// 6. Optional setups
//  Clear CTMU interrupt flag
//  Configure ADC to trigger on CTMU strobe
//  Configure button/key circuit pins to be ADC inputs
//  Configure ADC input multiplexer to point to first button/key pin
//  Turn on the ADC

// 7. Turn on the CTMU module, wait 50 us for charge pump to stabilize
    PLIB_CTMU_Enable(CTMU_ID_1);
    // TO DO: Wait 50 microseconds

// 8. Discharge the connected circuit by grounding the charge pump.
    PLIB_CTMU_CurrentSourceGrounded(CTMU_ID_1);
    // TO DO: Enable ADC sampling
    // TO DO: Wait for discharge to occur
```

Measuring Time Between Pulses

This section describes the process for measuring time between pulses.

Description

To measure the time between rising edges of two input signals, first calibrate the CTMU charge pump, using a calibration resistor on the Comparator/ANx input pin. Then replace the resistor with a calibrated capacitor of the correct size for the expected time delay and desired charge pump settings (charge range and trim). Do the following to set up the CTMU for these measurements:

1. Reset the CTMU back to its POR defaults. (This step isn't necessary if the CTMU is not being reconfigured.)
2. Select the current source range.
3. Adjust the current source trim.
4. Select the Pulse Generation generation.
5. Clear the Edge Status bits.
6. Select the input source for each edge.
7. Configure each edge to trigger on a rising edge.
8. Enable edge sequencing so Edge 1 occurs before Edge 2.
9. Enable edges to control the charge pump.
10. Turn on the CTMU module then wait 50 us for charge pump to stabilize.
11. Discharge the connected circuit by grounding the charge pump. (If using the ADC, enable ADC sampling to connect sample and hold capacitor to ground.)

```
// 1. Reset CTMU back to POR defaults
    PLIB_CTMU_PORDefaultsRestore(CTMU_ID_1);  // POR defaults, turn CTMU off

// 2. Select the current source range.
    PLIB_CTMU_CurrentRangeSet(CTMU_ID_1,CTMU_CurrentRangeBase_Base);
```

```
// 3. Adjust the current source trim.
    PLIB_CTMU_CurrentTrimSet(CTMU_ID_1,+10); // increase by 20% = +10 * 2%

// 4. Select the operating mode (Pulse Generation)
    PLIB_CTMU_TimePulseGenEnable(CTMU_ID_1); // Use Comparator Input/ANx pin

// 5. Clear the Edge Status bits
    PLIB_CTMU_EdgeStatusNoEdgesAllSet(CTMU_ID_1);

// 6. Configure the edge input sources for Edge 1 and Edge 2.
    PLIB_CTMU_EdgeSourceSelect(CTMU_ID_1,CTMU_Edge1,CTMU_EdgeSource_0);
    PLIB_CTMU_EdgeSourceSelect(CTMU_ID_1,CTMU_Edge2,CTMU_EdgeSource_1);

// 7. Configure the input polarities:
    PLIB_CTMU_EdgePolarityPositiveSet(CTMU_ID_1,CTMU_Edge1); // Trigger on Rising Edge
    PLIB_CTMU_EdgePolarityPositiveSet(CTMU_ID_1,CTMU_Edge2); // Trigger on Rising Edge

// 8. Enable edge sequencing so Edge1 (rising) occurs before Edge 2 (rising)
    PLIB_CTMU_EdgeSequenceEnable(CTMU_ID_1);

// 9. Enable edges to control charge pump
    PLIB_CTMU_EdgeEnable(CTMU_ID_1);

// 10. Turn on the CTMU module, wait 50 us for charge pump to stabilize
    PLIB_CTMU_Enable(CTMU_ID_1);
    // TO DO: Wait 50 microseconds

// 11. Discharge the connected circuit by grounding the charge pump.
    PLIB_CTMU_CurrentSourceGrounded(CTMU_ID_1);
    // TO DO: Enable ADC sampling
    // TO DO: Wait for discharge to occur
```

The time between the rising edge of each pulse is given by:

```
Time = ( Capacitance * (VDD * ADC_Count / MaxADCCount) ) / Current
```

Measuring Pulse Width

This section describes the process for measuring pulse width.

Description

Measuring the pulse width of a single input is similar to measuring the time between two pulses. The only difference is that the same signal is used for both edges and Edge 2 is triggered on a falling edge rather than a rising edge.

```
// 1. Reset CTMU back to POR defaults
    PLIB_CTMU_PORDefaultsRestore(CTMU_ID_1);  // POR defaults, turn CTMU off

// 2. Select the current source range.
    PLIB_CTMU_CurrentRangeSet(CTMU_ID_1,CTMU_CurrentRangeBase_Base);

// 3. Adjust the current source trim.
    PLIB_CTMU_CurrentTrimSet(CTMU_ID_1,+10); // increase by 20% = +10 * 2%

// 4. Select the operating mode (Pulse Generation)
    PLIB_CTMU_TimePulseGenEnable(CTMU_ID_1); // Use Comparator Input/ANx pin

// 5. Clear the Edge Status bits
    PLIB_CTMU_EdgeStatusSetAllNoEdges(CTMU_ID_1);

// 6. Use the same input source for both edges.
    PLIB_CTMU_EdgeSourceSelect(CTMU_ID_1,CTMU_Edge1,CTMU_EdgeSource_0);
    PLIB_CTMU_EdgeSourceSelect(CTMU_ID_1,CTMU_Edge2,CTMU_EdgeSource_0);

// 7. Configure the input polarities:
    PLIB_CTMU_EdgePolarityPositiveSet(CTMU_ID_1,CTMU_Edge1); // Trigger on Rising Edge
    PLIB_CTMU_EdgePolarityPositiveSet(CTMU_ID_1,CTMU_Edge2); // Trigger on falling Edge

// 8. Enable edge sequencing so Edge 1 (rising) occurs before Edge 2 (falling)
    PLIB_CTMU_EdgeSequenceEnable(CTMU_ID_1);

// 9. Enable edges to control charge pump
```

```
    PLIB_CTMU_EdgeEnable(CTMU_ID_1);

// 10. Turn on the CTMU module, wait 50 us for charge pump to stabilize
    PLIB_CTMU_Enable(CTMU_ID_1);
    // TO DO: Wait 50 microseconds

// 11. Discharge the connected circuit by grounding the charge pump.
    PLIB_CTMU_CurrentSourceGrounded(CTMU_ID_1);
    // TO DO: Enable ADC sampling
    // TO DO: Wait for discharge to occur
```

Again, the time between the rising edge of the pulse and the falling edge is given by:

Time = (Capacitance * (VDD * ADC_Count / MaxADCCount)) / Current

Capacitive Touch Measurement

This section describes how to measure the capacitance of a button.

Description

Assuming that the CTMU has been configured as previously described, the following code example shows how to measure the capacitance of a button.

```
#define MAX_ADC_CHANNELS  13
// Mapping of port (A,B,C) and pin (0-15) for each ADC Channel
   unsigned short int PortPinADC[MAX_ADC_CHANNELS] =
                                      { 0xA0, 0xA1, 0xB0, 0xB1,
                                        0xB2, 0xB3, 0xC0, 0xC1,
                                        0xC2, 0xBF, 0xBE, 0xBD,
                                        0xC3 };
    unsigned short int Vpos;  // ADC voltage measurement
    unsigned short int iChan, // ADC channel assigned to each button
                       PortPinChan, // Entry in PortPinADC coding matrix for each button
                       iPort, // Port for each button (0xA,0xB,0xC for ports A,B,C)
                       iPin;  // Pin for each ADC channel (0-15)

    iChan = ScanChannels[button_set_number]; // button_set_number = button to be scanned
    PortPinChan = PortPinADC[iChan];
    iPort   = (PortPinChan>>4) & 0x0F;
    iPin    =  PortPinChan & 0x0F;

    // TO DO: Switch ADC to sensor pin given by iChan

    switch ( iPort )  // Make sure pin is setup for input
    {
        case 0xA:
            // TO DO: Tri-state iPin for Port A
            break;
        case 0xB:
            // TO DO: Tri-state iPin for Port B
            break;
        case 0xC:
            // TO DO: Tri-state iPin for Port C
            break;
    }

    // TO DO: Start ADC sampling to connect sample and hold cap to charge pump
    PLIB_CTMU_CurrentSourceGrounded(CTMU_ID_1);    // Ground charge pump
    // TO DO: Wait for discharge of any residual charge from ADC S&H cap and button cap

    PLIB_CTMU_CurrentSourceNotGrounded(CTMU_ID_1); // Unground charge pump
    // TO DO: Disable interrupts to prevent ISRs from corrupting charge timing
    PLIB_CTMU_EdgeStatusGotEdgeSet( CTMU_ID_1, CTMU_Edge1 ); // Start charging button
    // TO DO: Wait for charge of button capacitor
    PLIB_CTMU_EdgeStatusNoEdgeSet( CTMU_ID_1, CTMU_Edge1 );  // Stop charging button

 // Start A/D conversion
    // TO DO: Stop ADC sampling to initiate ADC conversion
    // TO DO: Enable interrupts, since timing is no longer critical
    // TO DO: Wait until ADC conversion complete
    PLIB_CTMU_CurrentSourceGrounded(CTMU_ID_1);  // Ground charge pump
```

```
//Read new Vpos
    // TO DO: Read ADC results buffer for Vpos
```

Having measured VPOS, the software can now compare it to a limit. If VPOS is less than limit, the button is being pressed by the user.

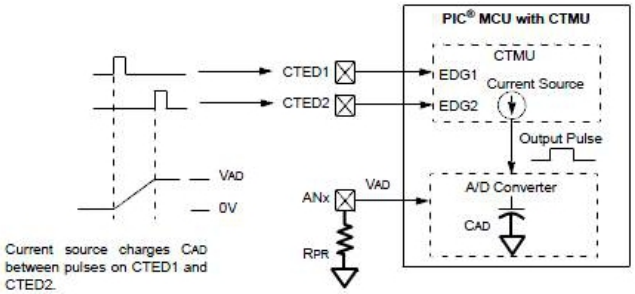
Example Applications

Examples of CTMU applications are discussed.

Description

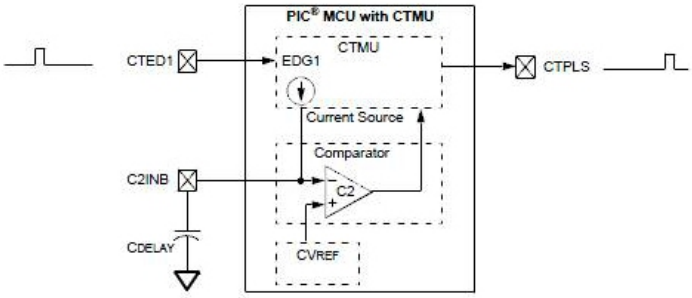
In most applications, the CTMU module provides the current to a circuit and the ADC module measures the resulting voltage. Depending on the setup of the CTMU and the ADC modules, this voltage can represent many things, including:

- On-chip temperature using the built-in temperature diode
- Finger press or swipe on a button panel by measuring relative capacitance
- True or absolute capacitance by measuring capacitance after calibrating the charge pump
- Pulse timing by turning the charge pump on/off with the pulse and then measuring the charge collected by a known capacitor



The CTMU External Edge Input pins, CTED1 and CTED2, are used for connecting two pulse signals to the CTMU. A calibrated capacitor, CKNOWN, is connected to an ADC analog input pin, ANx. The voltage captured by the ADC measures the time difference between the rising edge of the signal on CTED1, which turns on the CTMU's current source, and the rising edge of the signal on CTED2, which turns off the current source.

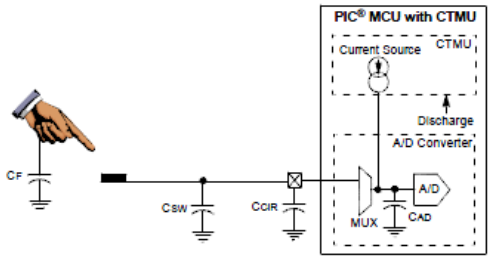
The CTMU and Comparator can be used, without the ADC's help, to delay an input pulse by a variable amount of time, using a known delay capacitor and the charge pumps settings.



In this example, an analog pulse on CTED1 is delayed by a fixed time using a special comparator associated with the CTMU (Comparator 2). Using a calibrated capacitor attached to a special comparator input pin, C2INB, a fixed delay can be added to the analog pulse based on the current and trim settings of the CTMU's charge pump. The time delay is based on the time required to bring the charge voltage on CDELAY up to CVREF. Once there, the comparator's output triggers the start of the output pulse.

Capacitive Touch

Capacitive touch is easy to implement with a CTMU module and its associated on-chip ADC. Since button assertion is a *change* in capacitance, no calibration is required. The software must merely detect the difference between an unpressed button and a button that is pressed. When the user's finger presses a capacitive touch button, the additional capacitance of the finger causes the ADC to see a lower voltage. So software interprets a button voltage drop as a button press by the user.



Configuring the Library

The library is configured for the supported CTMU module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) CTMU Control Functions

	Name	Description
	PLIB_CTMU_CurrentRangeSet	Selects the current source range.
	PLIB_CTMU_Disable	Disables the CTMU module.
	PLIB_CTMU_Enable	Enables the CTMU module.
	PLIB_CTMU_StopInIdleDisable	Sets the CTMU module to continue running when the device is in Idle mode.
	PLIB_CTMU_StopInIdleEnable	Sets the CTMU module to not operate when the device is in Idle mode.
	PLIB_CTMU_AutomaticADCTriggerDisable	Disables the module to automatically trigger an analog-to-digital conversion.
	PLIB_CTMU_AutomaticADCTriggerEnable	Enables the module to automatically trigger an analog-to-digital conversion when the second edge event has occurred.
	PLIB_CTMU_CurrentDischargeDisable	Disables the Current discharge by not connecting the current source output to ground.
	PLIB_CTMU_CurrentDischargeEnable	Enables the Current discharge by connecting the current source output to ground.
	PLIB_CTMU_CurrentTrimSet	Trims current source off of the nominal value.
	PLIB_CTMU_TimePulseGenerationDisable	Disables generation of time pulses using Comparator 2.
	PLIB_CTMU_TimePulseGenerationEnable	Enables the generation of time pulses.

b) Edge Control and Status Functions

	Name	Description
	PLIB_CTMU_EdgeDisable	Disables the edges of the CTMU.
	PLIB_CTMU_EdgeEnable	Enables the edges of the CTMU.
	PLIB_CTMU_EdgeSequenceDisable	Disables the edge sequence of the CTMU.
	PLIB_CTMU_EdgeSequenceEnable	Enables the edge sequence of the CTMU.
	PLIB_CTMU_EdgeIsSet	Gets the status of a CTMU edge.
	PLIB_CTMU_EdgePolaritySet	Sets a CTMU edge to fire on an edge/level with the selected polarity of signal.
	PLIB_CTMU_EdgeSensitivitySet	Sets CTMU sensitivity for the selected edge.
	PLIB_CTMU_EdgeSet	Sets the status of a CTMU edge.
	PLIB_CTMU_EdgeTriggerSourceSelect	Assigns a trigger source for an edge.

c) Feature Existence Functions

	Name	Description
	PLIB_CTMU_ExistsAutomaticADCTrigger	Identifies whether the AutomaticADCTrigger feature exists on the CTMU module.
	PLIB_CTMU_ExistsCurrentDischarge	Identifies whether the CurrentDischarge feature exists on the CTMU module.
	PLIB_CTMU_ExistsCurrentRange	Identifies whether the CurrentRange feature exists on the CTMU module.
	PLIB_CTMU_ExistsCurrentTrim	Identifies whether the CurrentTrim feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeEnable	Identifies whether the EdgeEnable feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgePolarity	Identifies whether the EdgePolarity feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeSensitivity	Identifies whether the EdgeSensitivity feature exists on the CTMU module,
	PLIB_CTMU_ExistsEdgeSequencing	Identifies whether the EdgeSequencing feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeStatus	Identifies whether the EdgeStatus feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeTriggerSource	Identifies whether the EdgeTriggerSource feature exists on the CTMU module.
	PLIB_CTMU_ExistsModuleEnable	Identifies whether the ModuleEnable feature exists on the CTMU module.
	PLIB_CTMU_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CTMU module.
	PLIB_CTMU_ExistsTimePulseGeneration	Identifies whether the TimePulseGeneration feature exists on the CTMU module.

Description

This section describes the Application Programming Interface (API) functions of the CTMU Peripheral Library.

Refer to each section for a detailed description.

a) CTMU Control Functions

PLIB_CTMU_CurrentRangeSet Function

Selects the current source range.

File

plib_ctmu.h

C

```
void PLIB_CTMU_CurrentRangeSet(CTMU_MODULE_ID index, CTMU_CURRENT_RANGE currentRange);
```

Returns

None.

Description

This function selects the current source range.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsCurrentRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Select current range
PLIB_CTMU_CurrentRangeSet ( CTMU_ID_0, CTMU_CURRENT_RANGE_BASE );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module
currentRange	Charge pump current range selected, one of the possible enumeration values from CTMU_CURRENT_RANGE enum

Function

void PLIB_CTMU_CurrentRangeSet (CTMU_MODULE_ID index, CTMU_CURRENT_RANGE currentRange)

PLIB_CTMU_Disable Function

Disables the CTMU module.

File

plib_ctmu.h

C

```
void PLIB_CTMU_Disable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function disables (turns OFF) the CTMU module. Typically, this function is called before reconfiguring the CTMU module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsModuleEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Turn off (disable) CTMU
PLIB_CTMU_Disable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_Disable (CTMU_MODULE_ID index)

PLIB_CTMU_Enable Function

Enables the CTMU module.

File

plib_ctmu.h

C

```
void PLIB_CTMU_Enable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function enables (turns ON) the CTMU module. This function is called after configuring the CTMU module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsModuleEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Turn on (enable) CTMU Module
PLIB_CTMU_Enable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_Enable (CTMU_MODULE_ID index)

PLIB_CTMU_StopInIdleDisable Function

Sets the CTMU module to continue running when the device is in Idle mode.

File

plib_ctmu.h

C

```
void PLIB_CTMU_StopInIdleDisable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function sets the CTMU module to continue running when the device is in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable CTMU to run when CPU is in Idle Mode
PLIB_CTMU_StopInIdleDisable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_StopInIdleDisable (CTMU_MODULE_ID index)

PLIB_CTMU_StopInIdleEnable Function

Sets the CTMU module to not operate when the device is in Idle mode.

File

plib_ctmu.h

C

```
void PLIB_CTMU_StopInIdleEnable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function sets the CTMU module to not operate when the device is in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Do not run in Idle mode
PLIB_CTMU_StopInIdleEnable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_StopInIdleEnable (CTMU_MODULE_ID index)

PLIB_CTMU_AutomaticADCTriggerDisable Function

Disables the module to automatically trigger an analog-to-digital conversion.

File

plib_ctmu.h

C

```
void PLIB_CTMU_AutomaticADCTriggerDisable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function disables the module to automatically trigger an analog-to-digital conversion.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsAutomaticADCTrigger in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable ADC trigger from CTMU
PLIB_CTMU_AutomaticADCTriggerDisable ( CTMU_ID_0 );

//To Do: Tell ADC to use CTMU trigger
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_AutomaticADCTriggerDisable (CTMU_MODULE_ID index)

PLIB_CTMU_AutomaticADCTriggerEnable Function

Enables the module to automatically trigger an analog-to-digital conversion when the second edge event has occurred.

File

plib_ctmu.h

C

```
void PLIB_CTMU_AutomaticADCTriggerEnable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function enables the module to automatically trigger an analog-to-digital conversion when the second edge event has occurred.

The ADC trigger is asserted whenever the CTMU current source switches from ON to OFF. The ADC can then end sampling and start converting the measured voltage into bits.

Remarks

The ADC must be configured to use the CTMU trigger signal generated.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsAutomaticADCTrigger in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable ADC trigger from CTMU
PLIB_CTMU_AutomaticADCTriggerEnable ( CTMU_ID_0 );

//To Do: Tell ADC to use CTMU trigger
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_AutomaticADCTriggerEnable (CTMU_MODULE_ID index)

PLIB_CTMU_CurrentDischargeDisable Function

Disables the Current discharge by not connecting the current source output to ground.

File

plib_ctmu.h

C

```
void PLIB_CTMU_CurrentDischargeDisable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function disables the Current discharge by not connecting the current source output to ground.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsCurrentDischarge in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Measure button voltage
// TO DO: Start ADC sampling to connect S&H capacitor to charge pump
PLIB_CTMU_CurrentDischargeDisable ( CTMU_ID_0 );    // Ground charge pump
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_CurrentDischargeDisable (CTMU_MODULE_ID index)

PLIB_CTMU_CurrentDischargeEnable Function

Enables the Current discharge by connecting the current source output to ground.

File

plib_ctmu.h

C

```
void PLIB_CTMU_CurrentDischargeEnable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function enables the Current discharge by connecting the current source output to ground.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsCurrentDischarge in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Measure button voltage
// TO DO: Start ADC sampling to connect S&H capacitor to charge pump
PLIB_CTMU_CurrentDischargeEnable ( CTMU_ID_0 );    // Ground charge pump
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_CurrentDischargeEnable (CTMU_MODULE_ID index)

PLIB_CTMU_CurrentTrimSet Function

Trims current source off of the nominal value.

File

plib_ctmu.h

C

```
void PLIB_CTMU_CurrentTrimSet(CTMU_MODULE_ID index, int16_t currentTrim);
```

Description

This function trims current source off of the nominal value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsCurrentTrim in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Set current trim, adjust from nominal by -10 x 2% = -20%
PLIB_CTMU_CurrentTrimSet ( CTMU_ID_0, -10 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module
currentTrim	Current trim index, from -31 to 31

Function

void PLIB_CTMU_CurrentTrimSet (CTMU_MODULE_ID index, int16_t currentTrim)

PLIB_CTMU_TimePulseGenerationDisable Function

Disables generation of time pulses using Comparator 2.

File

plib_ctmu.h

C

```
void PLIB_CTMU_TimePulseGenerationDisable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function disables generation of time pulses using Comparator 2.

Current source is made available for other measurements, including capacitance, time, or temperature (if temperature sensor is available).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsTimePulseGeneration in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Return to button relative capacitance measurements
PLIB_CTMU_TimePulseGenerationDisable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_TimePulseGenerationDisable (CTMU_MODULE_ID index)

PLIB_CTMU_TimePulseGenerationEnable Function

Enables the generation of time pulses.

File

plib_ctmu.h

C

```
void PLIB_CTMU_TimePulseGenerationEnable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function enables generation of time pulses using Comparator 2.

Current source will not be available for capacitance, time, or temperature measurements.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsTimePulseGeneration in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Send current to ADC S&H without going through ADC input MUX.
PLIB_CTMU_TimePulseGenerationEnable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_TimePulseGenerationEnable (CTMU_MODULE_ID index)

b) Edge Control and Status Functions

PLIB_CTMU_EdgeDisable Function

Disables the edges of the CTMU.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeDisable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function disables the edges of the CTMU (makes them blocked).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable edges, now can only turn CTMU on/off by modifying edge status
PLIB_CTMU_EdgeDisable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_EdgeDisable (CTMU_MODULE_ID index)

PLIB_CTMU_EdgeEnable Function

Enables the edges of the CTMU.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeEnable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function enables the edges of the CTMU (makes it not blocked).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable edges so that input signals can turn the CTMU charge pump on and off
PLIB_CTMU_EdgeEnable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_EdgeEnable (CTMU_MODULE_ID index)

PLIB_CTMU_EdgeSequenceDisable Function

Disables the edge sequence of the CTMU.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeSequenceDisable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function disables the edge sequence of the CTMU.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeSequencing in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Trigger on Edge1 or Edge2, which ever one occurs first
PLIB_CTMU_EdgeSequenceDisable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_EdgeSequenceDisable (CTMU_MODULE_ID index)

PLIB_CTMU_EdgeSequenceEnable Function

Enables the edge sequence of the CTMU.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeSequenceEnable(CTMU_MODULE_ID index);
```

Returns

None.

Description

This function enables the edge sequence of the CTMU. Edge 1 must occur before Edge 2 can occur.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeSequencing in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Trigger on Edge1 then Edge2, i.e., don't trigger on Edge 2
//until Edge 1 has occurred
PLIB_CTMU_EdgeSequenceEnable ( CTMU_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module

Function

void PLIB_CTMU_EdgeSequenceEnable (CTMU_MODULE_ID index)

PLIB_CTMU_EdgeIsSet Function

Gets the status of a CTMU edge.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_EdgeIsSet(CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber);
```

Returns

None.

Description

This function gets the status of the selected CTMU edge (given by edgeNumber).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool edgeStatus;

edgeStatus = PLIB_CTMU_EdgeIsSet ( CTMU_ID_0, CTMU_EDGE1 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module.
edgeNumber	CTMU Edge for which the trigger source to be set, one of the possible elements of the CTMU_EDGE_SELECT enum.

Function

bool PLIB_CTMU_EdgeIsSet (CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber)

PLIB_CTMU_EdgePolaritySet Function

Sets a CTMU edge to fire on an edge/level with the selected polarity of signal.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgePolaritySet(CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber, CTMU_EDGE_POLARITY
edgePolarity);
```

Returns

None.

Description

This function sets a CTMU edge to fire on an edge/level with the selected polarity of signal.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgePolarity in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Example to set both edges to trigger on positive level/edge.
PLIB_CTMU_EdgePolaritySet ( CTMU_ID_0, CTMU_EDGE1, CTMU_EDGE_X_POSITIVE_EDGE );
PLIB_CTMU_EdgePolaritySet ( CTMU_ID_0, CTMU_EDGE2, CTMU_EDGE_X_POSITIVE_EDGE );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module.
edgeNumber	CTMU Edge for which the polarity to be set, one of the possible elements of CTMU_EDGE_SELECT enum.
edgeSense	CTMU Edge polarity, one of the possible elements of the CTMU_EDGE_POLARITY enum.

Function

void PLIB_CTMU_EdgePolaritySet (CTMU_MODULE_ID index,CTMU_EDGE_SELECT edgeNumber,

CTMU_EDGE_POLARITY edgePolarity);

PLIB_CTMU_EdgeSensitivitySet Function

Sets CTMU sensitivity for the selected edge.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeSensitivitySet(CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber, CTMU_EDGE_SENSITIVITY
edgeSense);
```

Returns

None.

Description

This function can be used to set the sensitivity of a particular edge to level-sensitive or edge sensitive.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeSensitivity in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Example to set both edges to trigger on edges rather than levels
PLIB_CTMU_EdgeSensitivitySet ( CTMU_ID_0, CTMU_EDGE1, CTMU_EDGE_X_EDGE_SENSITIVE );
PLIB_CTMU_EdgeSensitivitySet ( CTMU_ID_0, CTMU_EDGE2, CTMU_EDGE_X_EDGE_SENSITIVE );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module.

edgeNumber	CTMU edge for which the sensitivity to be set, one of the possible elements of the CTMU_EDGE_SELECT enum.
edgeSense	CTMU Edge sensitivity, one of the possible elements of the CTMU_EDGE_SENSITIVITY enum.

Function

void PLIB_CTMU_EdgeSensitivitySet (CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber,
CTMU_EDGE_SENSITIVITY edgeSense)

PLIB_CTMU_EdgeSet Function

Sets the status of a CTMU edge.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeSet(CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber);
```

Returns

None.

Description

This function sets the status of the selected CTMU edge (given by edge_number).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_CTMU_EdgeSet ( CTMU_ID_0, CTMU_EDGE1 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module.
edgeNumber	CTMU Edge for which the trigger source to be set, one of the possible elements of the CTMU_EDGE_SELECT enum.

Function

void PLIB_CTMU_EdgeSet (CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber)

PLIB_CTMU_EdgeTriggerSourceSelect Function

Assigns a trigger source for an edge.

File

plib_ctmu.h

C

```
void PLIB_CTMU_EdgeTriggerSourceSelect(CTMU_MODULE_ID index, CTMU_EDGE_SELECT edgeNumber,
CTMU_TRIGGER_SOURCES triggerSource);
```

Returns

None.

Description

This function will enable the CTMU edge (given by edge_number) to trigger on the signal provided by the trigger source selected.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_CTMU_ExistsEdgeTriggerSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Set source 0 as the trigger source for both the edges
PLIB_CTMU_EdgeTriggerSourceSelect ( CTMU_ID_0, CTMU_EDGE1, CTMU_TRIGGERSOURCE_0 );
PLIB_CTMU_EdgeTriggerSourceSelect ( CTMU_ID_0, CTMU_EDGE2, CTMU_TRIGGERSOURCE_0 );
```

Parameters

Parameters	Description
index	Identifier for the desired CTMU module.
edgeNumber	CTMU Edge for which the trigger source to be set, one of the possible elements of the CTMU_EDGE_SELECT enum.
triggerSource	CTMU Edge trigger source, one of the possible elements of the CTMU_TRIGGER_SOURCES enum.

Function

void PLIB_CTMU_EdgeTriggerSourceSelect (CTMU_MODULE_ID index,

CTMU_EDGE_SELECT edgeNumber,

CTMU_TRIGGER_SOURCES triggerSource)

c) Feature Existence Functions

PLIB_CTMU_ExistsAutomaticADCTrigger Function

Identifies whether the AutomaticADCTrigger feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsAutomaticADCTrigger(CTMU_MODULE_ID index);
```

Returns

- true - The AutomaticADCTrigger feature is supported on the device
- false - The AutomaticADCTrigger feature is not supported on the device

Description

This function identifies whether the AutomaticADCTrigger feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_AutomaticADCTriggerDisable
- PLIB_CTMU_AutomaticADCTriggerEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsAutomaticADCTrigger(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsCurrentDischarge Function

Identifies whether the CurrentDischarge feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsCurrentDischarge(CTMU_MODULE_ID index);
```

Returns

- true - The CurrentDischarge feature is supported on the device
- false - The CurrentDischarge feature is not supported on the device

Description

This function identifies whether the CurrentDischarge feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_CurrentDischargeEnable
- PLIB_CTMU_CurrentDischargeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsCurrentDischarge(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsCurrentRange Function

Identifies whether the CurrentRange feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsCurrentRange(CTMU_MODULE_ID index);
```

Returns

- true - The CurrentRange feature is supported on the device
- false - The CurrentRange feature is not supported on the device

Description

This function identifies whether the CurrentRange feature is available on the CTMU module. When this function returns true, this function is supported on the device:

- PLIB_CTMU_CurrentRangeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsCurrentRange(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsCurrentTrim Function

Identifies whether the CurrentTrim feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsCurrentTrim(CTMU_MODULE_ID index);
```

Returns

- true - The CurrentTrim feature is supported on the device
- false - The CurrentTrim feature is not supported on the device

Description

This function identifies whether the CurrentTrim feature is available on the CTMU module. When this function returns true, this function is supported on the device:

- PLIB_CTMU_CurrentTrimSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsCurrentTrim(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsEdgeEnable Function

Identifies whether the EdgeEnable feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsEdgeEnable(CTMU_MODULE_ID index);
```

Returns

- true - The EdgeEnable feature is supported on the device
- false - The EdgeEnable feature is not supported on the device

Description

This function identifies whether the EdgeEnable feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_EdgeDisable
- PLIB_CTMU_EdgeEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsEdgeEnable(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsEdgePolarity Function

Identifies whether the EdgePolarity feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsEdgePolarity(CTMU_MODULE_ID index);
```

Returns

- true - The EdgePolarity feature is supported on the device
- false - The EdgePolarity feature is not supported on the device

Description

This function identifies whether the EdgePolarity feature is available on the CTMU module. When this function returns true, this function is supported on the device:

- PLIB_CTMU_EdgePolaritySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsEdgePolarity(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsEdgeSensitivity Function

Identifies whether the EdgeSensitivity feature exists on the CTMU module,

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsEdgeSensitivity(CTMU_MODULE_ID index);
```

Returns

- true - The EdgeSensitivity feature is supported on the device
- false - The EdgeSensitivity feature is not supported on the device

Description

This function identifies whether the EdgeSensitivity feature is available on the CTMU module. When this function returns true, this function is supported on the device:

- PLIB_CTMU_EdgeSensitivitySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsEdgeSensitivity(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsEdgeSequencing Function

Identifies whether the EdgeSequencing feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsEdgeSequencing(CTMU_MODULE_ID index);
```

Returns

- true - The EdgeSequencing feature is supported on the device
- false - The EdgeSequencing feature is not supported on the device

Description

This function identifies whether the EdgeSequencing feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_EdgeSequenceDisable
- PLIB_CTMU_EdgeSequenceEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsEdgeSequencing(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsEdgeStatus Function

Identifies whether the EdgeStatus feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsEdgeStatus(CTMU_MODULE_ID index);
```

Returns

- true - The EdgeStatus feature is supported on the device
- false - The EdgeStatus feature is not supported on the device

Description

This function identifies whether the EdgeStatus feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_EdgeIsSet
- PLIB_CTMU_EdgeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsEdgeStatus(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsEdgeTriggerSource Function

Identifies whether the EdgeTriggerSource feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsEdgeTriggerSource(CTMU_MODULE_ID index);
```

Returns

- true - The EdgeTriggerSource feature is supported on the device
- false - The EdgeTriggerSource feature is not supported on the device

Description

This function identifies whether the EdgeTriggerSource feature is available on the CTMU module. When this function returns true, this function is supported on the device:

- PLIB_CTMU_EdgeTriggerSourceSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsEdgeTriggerSource(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsModuleEnable Function

Identifies whether the ModuleEnable feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsModuleEnable(CTMU_MODULE_ID index);
```

Returns

- true - The ModuleEnable feature is supported on the device
- false - The ModuleEnable feature is not supported on the device

Description

This function identifies whether the ModuleEnable feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_Disable
- PLIB_CTMU_Enable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsModuleEnable(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsStopInIdle(CTMU_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_StopInIdleDisable
- PLIB_CTMU_StopInIdleEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsStopInIdle(CTMU_MODULE_ID index)

PLIB_CTMU_ExistsTimePulseGeneration Function

Identifies whether the TimePulseGeneration feature exists on the CTMU module.

File

plib_ctmu.h

C

```
bool PLIB_CTMU_ExistsTimePulseGeneration(CTMU_MODULE_ID index);
```

Returns

- true - The TimePulseGeneration feature is supported on the device
- false - The TimePulseGeneration feature is not supported on the device

Description

This function identifies whether the TimePulseGeneration feature is available on the CTMU module. When this function returns true, these functions are supported on the device:

- PLIB_CTMU_TimePulseGenerationDisable
- PLIB_CTMU_TimePulseGenerationEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_CTMU_ExistsTimePulseGeneration(CTMU_MODULE_ID index)

Files

Files

Name	Description
plib_ctmu.h	This file contains the interface definition for the CTMU Peripheral Library.

Description

This section lists the source and header files used by the library.

plib_ctmu.h

This file contains the interface definition for the CTMU Peripheral Library.

Functions

	Name	Description
	PLIB_CTMU_AutomaticADCTriggerDisable	Disables the module to automatically trigger an analog-to-digital conversion.
	PLIB_CTMU_AutomaticADCTriggerEnable	Enables the module to automatically trigger an analog-to-digital conversion when the second edge event has occurred.
	PLIB_CTMU_CurrentDischargeDisable	Disables the Current discharge by not connecting the current source output to ground.
	PLIB_CTMU_CurrentDischargeEnable	Enables the Current discharge by connecting the current source output to ground.
	PLIB_CTMU_CurrentRangeSet	Selects the current source range.
	PLIB_CTMU_CurrentTrimSet	Trims current source off of the nominal value.
	PLIB_CTMU_Disable	Disables the CTMU module.
	PLIB_CTMU_EdgeDisable	Disables the edges of the CTMU.
	PLIB_CTMU_EdgeEnable	Enables the edges of the CTMU.

	PLIB_CTMU_EdgeIsSet	Gets the status of a CTMU edge.
	PLIB_CTMU_EdgePolaritySet	Sets a CTMU edge to fire on an edge/level with the selected polarity of signal.
	PLIB_CTMU_EdgeSensitivitySet	Sets CTMU sensitivity for the selected edge.
	PLIB_CTMU_EdgeSequenceDisable	Disables the edge sequence of the CTMU.
	PLIB_CTMU_EdgeSequenceEnable	Enables the edge sequence of the CTMU.
	PLIB_CTMU_EdgeSet	Sets the status of a CTMU edge.
	PLIB_CTMU_EdgeTriggerSourceSelect	Assigns a trigger source for an edge.
	PLIB_CTMU_Enable	Enables the CTMU module.
	PLIB_CTMU_ExistsAutomaticADCTrigger	Identifies whether the AutomaticADCTrigger feature exists on the CTMU module.
	PLIB_CTMU_ExistsCurrentDischarge	Identifies whether the CurrentDischarge feature exists on the CTMU module.
	PLIB_CTMU_ExistsCurrentRange	Identifies whether the CurrentRange feature exists on the CTMU module.
	PLIB_CTMU_ExistsCurrentTrim	Identifies whether the CurrentTrim feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeEnable	Identifies whether the EdgeEnable feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgePolarity	Identifies whether the EdgePolarity feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeSensitivity	Identifies whether the EdgeSensitivity feature exists on the CTMU module,
	PLIB_CTMU_ExistsEdgeSequencing	Identifies whether the EdgeSequencing feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeStatus	Identifies whether the EdgeStatus feature exists on the CTMU module.
	PLIB_CTMU_ExistsEdgeTriggerSource	Identifies whether the EdgeTriggerSource feature exists on the CTMU module.
	PLIB_CTMU_ExistsModuleEnable	Identifies whether the ModuleEnable feature exists on the CTMU module.
	PLIB_CTMU_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the CTMU module.
	PLIB_CTMU_ExistsTimePulseGeneration	Identifies whether the TimePulseGeneration feature exists on the CTMU module.
	PLIB_CTMU_StopInIdleDisable	Sets the CTMU module to continue running when the device is in Idle mode.
	PLIB_CTMU_StopInIdleEnable	Sets the CTMU module to not operate when the device is in Idle mode.
	PLIB_CTMU_TimePulseGenerationDisable	Disables generation of time pulses using Comparator 2.
	PLIB_CTMU_TimePulseGenerationEnable	Enables the generation of time pulses.

Description

Charge Time Measurement Unit (CTMU) Peripheral Library Header

This library provides a low-level abstraction of the Charge Time Measurement Unit(CTMU) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

File Name

plib_ctmu.h

Company

Microchip Technology Inc.

Deadman Timer Peripheral Library

This section describes the Deadman Timer (DMT) Peripheral Library.

Introduction

This library provides a low-level abstraction of the DMT Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by abstracting differences from one microcontroller variant to another.

Description

The primary function of the Deadman Timer (DMT) is to reset the processor in the event of a software malfunction.

Using the Library

This topic describes the basic architecture of the DMT Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_dmt.h`

The interface to the DMT Peripheral Library is defined in the `plib_dmt.h` header file, which is included by the peripheral library header file,

`peripheral.h`. Any C language source (`.c`) file that uses the DMT Peripheral Library must include `peripheral.h`.

Library File:

The DMT Peripheral Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Hardware Abstraction Model

This library provides the low-level abstraction of the Deadman Timer module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

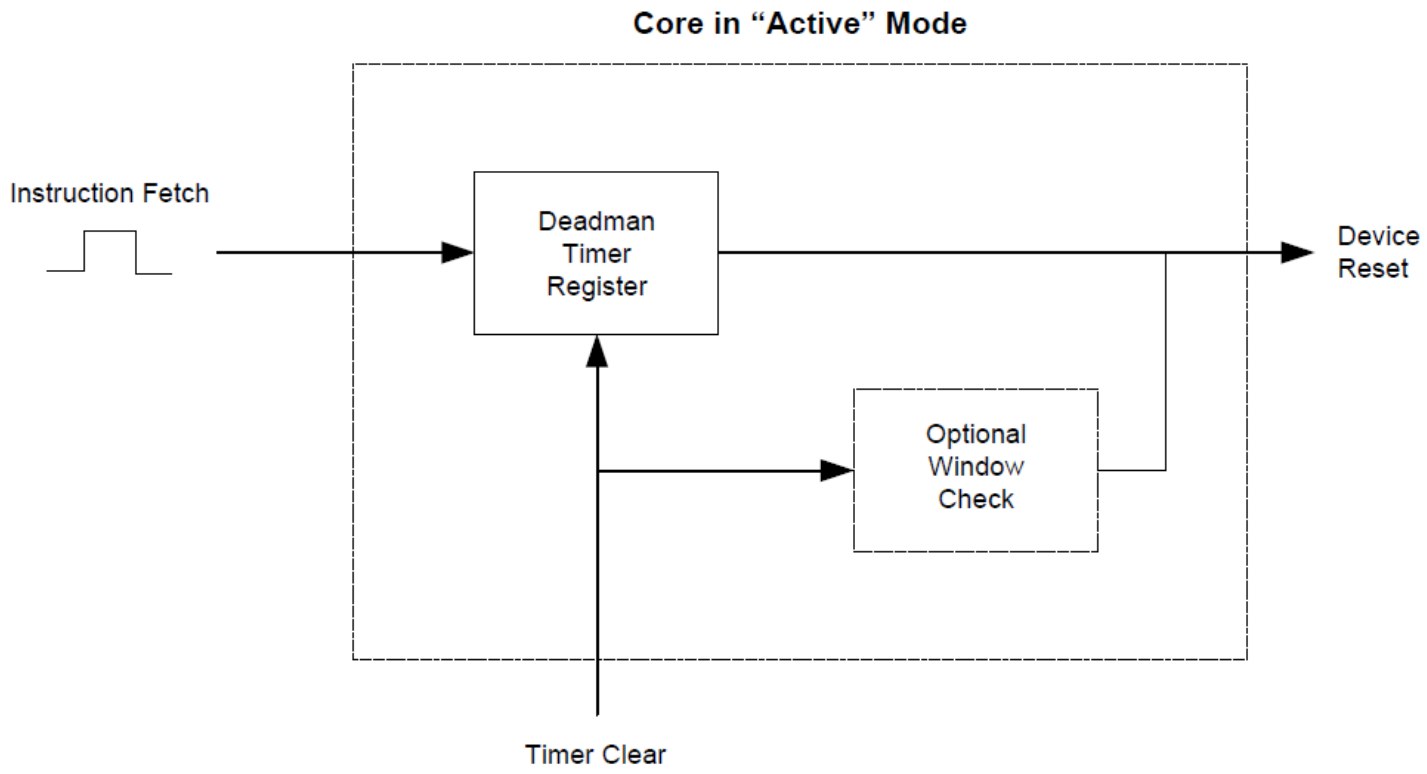
The Deadman Timer is a free-running instruction fetch timer, which is clocked whenever an instruction fetch occurs.

The Deadman Timer is active in the normal mode only. Since the Deadman Timer is only incremented by instruction fetches, the count value will not change when the core is inactive. So, the Deadman Timer remains inactive in Sleep and Idle modes.

The Deadman Timer module uses the Deadman register as a timer. If there is no reset signal from the software and if the counter overflows, this results in the device Reset.

For Windowed mode, resetting the counter when the count is not in the specified window will also lead to the device Reset.

Deadman Timer Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Deadman Timer module.

Library Interface Section	Description
General Configuration Functions	Includes module enable and disable functions.
General Status Functions	Status routines for the DMT.
Feature Existence Functions	Functions that determine existence of certain features.

How the Library Works

This section provides information on how the Deadman Timer Peripheral Library works.

Configuring the Library

The library is configured for the supported Deadman Timer module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_DMT_ClearStep1	Resets the DMT module.
	PLIB_DMT_ClearStep2	Resets the DMT module.
	PLIB_DMT_Disable	Disables the DMT module.
	PLIB_DMT_Enable	Enables the DMT module.

b) General Status Functions

	Name	Description
	PLIB_DMT_BAD1Get	Returns BAD1 flag from the DMT Status Register.
	PLIB_DMT_BAD2Get	Returns BAD2 flag from the DMT Status Register.
	PLIB_DMT_CounterGet	Returns the DMT counter value.
	PLIB_DMT_IsEnabled	Returns the Deadman Timer on/off(enable/disable) status.
	PLIB_DMT_PostscalerIntervalGet	Returns the DMT postscaler interval value.
	PLIB_DMT_PostscalerValueGet	Returns the DMT postscaler value.
	PLIB_DMT_WindowIsOpen	Returns Window is Open flag from the DMT Status Register.
	PLIB_DMT_EventOccurred	Returns Event flag from the DMT Status Register.

c) Feature Existence Functions

	Name	Description
	PLIB_DMT_ExistsCounter	Identifies whether the Counter feature exists on the DMT module.
	PLIB_DMT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the DMT module.
	PLIB_DMT_ExistsPostscalerInterval	Identifies whether the Postscaler Interval feature exists on the DMT module.
	PLIB_DMT_ExistsPostscalerValue	Identifies whether the Postscaler Value feature exists on the DMT module.
	PLIB_DMT_ExistsStatus	Identifies whether the Status feature exists on the DMT module.
	PLIB_DMT_ExistsStep1	Identifies whether the STEP1 Clear feature exists on the DMT module.
	PLIB_DMT_ExistsStep2	Identifies whether the STEP2 Clear feature exists on the DMT module.

d) Data Types and Constants

	Name	Description
	DMT_MODULE_ID	Identifies the supported DMT modules.

Description

This section describes the Application Programming Interface (API) functions of the Deadman Timer Peripheral Library.
Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_DMT_ClearStep1 Function

Resets the DMT module.

File

plib_dmt.h

C

```
void PLIB_DMT_ClearStep1(DMT_MODULE_ID index);
```

Returns

None.

Description

This function performs the STEP1 Clearing of the DMT module. The DMT module should be cleared in two steps, within the interval determined by the Count Window before the DMT forces an NMI or device reset.

Remarks

Resetting the device before the count reaches the window will cause a reset in Windowed mode.

The example code does not include the settings that should be done through the Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsTimerClear in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );

//Application loop
while(1)
{
    PLIB_DMT_ClearStep1 ( DMT_ID_0 );
    //user code
    PLIB_DMT_ClearStep2 ( DMT_ID_0 );
    //user code
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

void PLIB_DMT_ClearStep1 (DMT_MODULE_ID index)

PLIB_DMT_ClearStep2 Function

Resets the DMT module.

File

plib_dmt.h

C

```
void PLIB_DMT_ClearStep2(DMT_MODULE_ID index);
```

Returns

None.

Description

This function performs the STEP2 Clearing of the DMT module. The DMT module should be cleared in two steps, within the interval determined by the Count Window before the DMT forces an NMI or device reset.

Remarks

Resetting the device before the count reaches the window will cause a reset in Windowed mode.

The example code doesn't include the settings that should be done through the Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsTimerClear in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );

//Application loop
while(1)
{
    PLIB_DMT_ClearStep1 ( DMT_ID_0 );
    //user code
    PLIB_DMT_ClearStep2 ( DMT_ID_0 );
    //user code
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

void PLIB_DMT_ClearStep2 (DMT_MODULE_ID index)

PLIB_DMT_Disable Function

Disables the DMT module.

File

plib_dmt.h

C

```
void PLIB_DMT_Disable(DMT_MODULE_ID index);
```

Returns

None.

Description

This function disables the DMT module if it is enabled in software.

Remarks

This function will not disable the DMT module if it is enabled through its Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

The DMT module must have been enabled through software.

Example

```
PLIB_DMT_Disable ( DMT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

void PLIB_DMT_Disable (DMT_MODULE_ID index)

PLIB_DMT_Enable Function

Enables the DMT module.

File

plib_dmt.h

C

```
void PLIB_DMT_Enable(DMT_MODULE_ID index);
```

Returns

None.

Description

This function enables the DMT module. If it is already enabled through the Configuration bits, it will keep it enabled.

Remarks

Calling this function is not necessary if it is enabled through its Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

void PLIB_DMT_Enable (DMT_MODULE_ID index)

b) General Status Functions

PLIB_DMT_BAD1Get Function

Returns BAD1 flag from the DMT Status Register.

File

plib_dmt.h

C

```
bool PLIB_DMT_BAD1Get(DMT_MODULE_ID index);
```

Returns

The flag value, true or false.

Description

This function returns the DMT BAD1 Flag. This flag is set when there is an incorrect write to STEP1, such as the wrong value, writing too early, or writing too late.

Remarks

The flag returned will indicated if a STEP1 write was not successful.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );
//user code
```

```
PLIB_DMT_ClearStep1();
if( PLIB_DMT_BAD1Get ( DMT_ID_0 ))
{
   //user code
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

bool PLIB_DMT_BAD1Get (DMT_MODULE_ID index)

PLIB_DMT_BAD2Get Function

Returns BAD2 flag from the DMT Status Register.

File

plib_dmt.h

C

```
bool PLIB_DMT_BAD2Get(DMT_MODULE_ID index);
```

Returns

The flag value, true or false.

Description

This function returns the DMT BAD2 Flag. This flag is set when there is an incorrect write to STEP2, such as the wrong value, writing too early, or writing too late.

Remarks

The flag returned will indicated if a STEP2 write was not successful.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );
//user code
PLIB_DMT_ClearStep1();
//user code
PLIB_DMT_ClearStep2();
if( PLIB_DMT_BAD2Get ( DMT_ID_0 ))
{
   //user code
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

bool PLIB_DMT_BAD2Get (DMT_MODULE_ID index)

PLIB_DMT_CounterGet Function

Returns the DMT counter value.

File

plib_dmt.h

C

```
uint32_t PLIB_DMT_CounterGet(DMT_MODULE_ID index);
```

Returns

The counter value.

Description

This function returns the DMT counter value. The value is the number of instructions counted since the count was last cleared.

Remarks

The value returned will be right-aligned.

Refer the data sheet of the specific device to get the division factor corresponding to the value.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsCounterValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t value;

PLIB_DMT_Enable ( DMT_ID_0 );
value = PLIB_DMT_CounterGet ( DMT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

uint32_t PLIB_DMT_CounterGet (DMT_MODULE_ID index)

PLIB_DMT_IsEnabled Function

Returns the Deadman Timer on/off(enable/disable) status.

File

plib_dmt.h

C

```
bool PLIB_DMT_IsEnabled(DMT_MODULE_ID index);
```

Returns

true - If the Deadman Timer is on false - If the Deadman Timer is off

Description

Returns the 'true', if the Deadman Timer is already ON. Otherwise returns 'false'.

Remarks

This function returns 'true' if the device is enabled either though the Configuration bits or in the software.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_DMT_IsEnabled ( DMT_ID_0 ) )
{
    //Do some action
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

bool PLIB_DMT_IsEnabled (DMT_MODULE_ID index)

PLIB_DMT_PostscalerIntervalGet Function

Returns the DMT postscaler interval value.

File

plib_dmt.h

C

```
uint32_t PLIB_DMT_PostscalerIntervalGet(DMT_MODULE_ID index);
```

Returns

The postscaler interval.

Description

This function returns the DMT postscaler interval. The value will correspond to a total number of instructions for DMT window, a value determined by configuration bits.

Remarks

The value returned will be right-aligned.

Refer the data sheet of the specific device to get the association of the configuration bits corresponding to this postscaler value.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsPostscalerValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t value;

PLIB_DMT_Enable ( DMT_ID_0 );
value = PLIB_DMT_PostscalerIntervalGet ( DMT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

uint32_t PLIB_DMT_PostscalerIntervalGet (DMT_MODULE_ID index)

PLIB_DMT_PostscalerValueGet Function

Returns the DMT postscaler value.

File

plib_dmt.h

C

```
uint32_t PLIB_DMT_PostscalerValueGet(DMT_MODULE_ID index);
```

Returns

The postscaler value.

Description

This function returns the DMT postscaler value. The value will correspond to a total number of instructions for DMT timeout, a value determined by

configuration bits.

Remarks

The value returned will be right-aligned.

Refer to the specific device data sheet to get the association of the configuration bits corresponding to this postscaler value.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsPostscalerValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t value;

PLIB_DMT_Enable ( DMT_ID_0 );
value = PLIB_DMT_PostscalerValueGet ( DMT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

uint32_t PLIB_DMT_PostscalerValueGet (DMT_MODULE_ID index)

PLIB_DMT_WindowIsOpen Function

Returns Window is Open flag from the DMT Status Register.

File

plib_dmt.h

C

```
bool PLIB_DMT_WindowIsOpen(DMT_MODULE_ID index);
```

Returns

The flag value, true or false.

Description

This function returns the flag indicating the DMT Window is open.

Remarks

The flag returned will indicated if the DMT window is open.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );
//user code
PLIB_DMT_ClearStep1();
//user code
if( PLIB_DMT_WindowIsOpen( DMT_ID_0 ))
{
   PLIB_DMT_ClearStep2();
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

bool PLIB_DMT_WindowIsOpen(DMT_MODULE_ID index)

PLIB_DMT_EventOccurred Function

Returns Event flag from the DMT Status Register.

File

plib_dmt.h

C

```
bool PLIB_DMT_EventOccurred(DMT_MODULE_ID index);
```

Returns

The flag value, true or false.

Description

This function returns the flag indicating a DMT event has happened, such as a Window Open, or a Bad Flag is set.

Remarks

The flag returned will indicate if a DMT event has happened.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_DMT_ExistsStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMT_Enable ( DMT_ID_0 );
//user code
PLIB_DMT_ClearStep1();
//user code
if( PLIB_DMT_EventOccurred( DMT_ID_0 ))
{
   //user code
}
```

Parameters

Parameters	Description
index	Identifies the desired DMT module

Function

bool PLIB_DMT_EventOccurred(DMT_MODULE_ID index)

c) Feature Existence Functions

PLIB_DMT_ExistsCounter Function

Identifies whether the Counter feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsCounter(DMT_MODULE_ID index);
```

Returns

- true - The Counter feature is supported on the device
- false - The Counter feature is not supported on the device

Description

This function identifies whether the Counter feature is available on the DMT module. When this function returns true, this function is supported on the device:

- PLIB_DMT_CounterGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsCounter(DMT_MODULE_ID index)

PLIB_DMT_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsEnableControl(DMT_MODULE_ID index);
```

Returns

Existence of the EnableControl feature:

- true - When EnableControl feature is supported on the device
- false - When EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the DMT module. When this function returns true, these functions are supported on the device:

- PLIB_DMT_Enable
- PLIB_DMT_Disable
- PLIB_DMT_IsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsEnableControl(DMT_MODULE_ID index)

PLIB_DMT_ExistsPostscalerInterval Function

Identifies whether the Postscaler Interval feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsPostscalerInterval(DMT_MODULE_ID index);
```

Returns

- true - The Postscaler Interval feature is supported on the device
- false - The Postscaler Interval feature is not supported on the device

Description

This function identifies whether the Postscaler Interval feature is available on the DMT module. When this function returns true, this function is supported on the device:

- PLIB_DMT_PostscalerIntervalGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsInterval(DMT_MODULE_ID index)

PLIB_DMT_ExistsPostscalerValue Function

Identifies whether the Postscaler Value feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsPostscalerValue(DMT_MODULE_ID index);
```

Returns

- true - The Postscaler Value feature is supported on the device
- false - The Postscaler Value feature is not supported on the device

Description

This function identifies whether the PostscalerValue feature is available on the DMT module. When this function returns true, this function is supported on the device:

- PLIB_DMT_PostscalerValueGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsPostscalerValue(DMT_MODULE_ID index)

PLIB_DMT_ExistsStatus Function

Identifies whether the Status feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsStatus(DMT_MODULE_ID index);
```

Returns

Existence of the Status feature:

- true - When Status feature is supported on the device
- false - When Status feature is not supported on the device

Description

This function identifies whether the Status feature is available on the DMT module. When this function returns true, these functions are supported on the device:

- PLIB_DMT_WindowIsOpen
- PLIB_DMT_EventOccurred
- PLIB_DMT_BAD1Get
- PLIB_DMT_BAD2Get

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsStatus(DMT_MODULE_ID index)

PLIB_DMT_ExistsStep1 Function

Identifies whether the STEP1 Clear feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsStep1(DMT_MODULE_ID index);
```

Returns

- true - The STEP1 Clear feature is supported on the device
- false - The STEP1 Clear feature is not supported on the device

Description

This function identifies whether the Step 1 Clear feature is available on the DMT module. When this function returns true, this function is supported on the device:

- PLIB_DMT_ClearStep1

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsStep1(DMT_MODULE_ID index)

PLIB_DMT_ExistsStep2 Function

Identifies whether the STEP2 Clear feature exists on the DMT module.

File

plib_dmt.h

C

```
bool PLIB_DMT_ExistsStep2(DMT_MODULE_ID index);
```

Returns

- true - The STEP2 Clear feature is supported on the device
- false - The STEP2 Clear feature is not supported on the device

Description

This function identifies whether the STEP2 Clear feature is available on the DMT module. When this function returns true, this function is supported on the device:

- PLIB_DMT_ClearStep2

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMT_ExistsStep2(DMT_MODULE_ID index)

d) Data Types and Constants

DMT_MODULE_ID Enumeration

Identifies the supported DMT modules.

File

plib_dmt_help.h

C

```
typedef enum {
  DMT_ID_0,
  DMT_NUMBER_OF_MODULES
} DMT_MODULE_ID;
```

Members

Members	Description
DMT_ID_0	DMT Module 0 ID
DMT_NUMBER_OF_MODULES	Number of available WDT modules.

Description

DMT Module ID

This enumeration identifies the available DMT modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Files

Files

Name	Description
plib_dmt.h	Deadman Timer (DMT) Peripheral Library interface header for Deadman Timer common definitions.
plib_dmt_help.h	

Description

This section lists the source and header files used by the library.

plib_dmt.h

Deadman Timer (DMT) Peripheral Library interface header for Deadman Timer common definitions.

Functions

	Name	Description
	PLIB_DMT_BAD1Get	Returns BAD1 flag from the DMT Status Register.
	PLIB_DMT_BAD2Get	Returns BAD2 flag from the DMT Status Register.
	PLIB_DMT_ClearStep1	Resets the DMT module.
	PLIB_DMT_ClearStep2	Resets the DMT module.
	PLIB_DMT_CounterGet	Returns the DMT counter value.
	PLIB_DMT_Disable	Disables the DMT module.
	PLIB_DMT_Enable	Enables the DMT module.
	PLIB_DMT_EventOccurred	Returns Event flag from the DMT Status Register.
	PLIB_DMT_ExistsCounter	Identifies whether the Counter feature exists on the DMT module.
	PLIB_DMT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the DMT module.
	PLIB_DMT_ExistsPostscalerInterval	Identifies whether the Postscaler Interval feature exists on the DMT module.
	PLIB_DMT_ExistsPostscalerValue	Identifies whether the Postscaler Value feature exists on the DMT module.
	PLIB_DMT_ExistsStatus	Identifies whether the Status feature exists on the DMT module.
	PLIB_DMT_ExistsStep1	Identifies whether the STEP1 Clear feature exists on the DMT module.
	PLIB_DMT_ExistsStep2	Identifies whether the STEP2 Clear feature exists on the DMT module.
	PLIB_DMT_IsEnabled	Returns the Deadman Timer on/off(enable/disable) status.
	PLIB_DMT_PostscalerIntervalGet	Returns the DMT postscaler interval value.
	PLIB_DMT_PostscalerValueGet	Returns the DMT postscaler value.
	PLIB_DMT_WindowIsOpen	Returns Window is Open flag from the DMT Status Register.

Description

Deadman Timer (DMT) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Deadman Timer Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the Deadman Timer peripheral.

File Name

plib_dmt.h

Company

Microchip Technology Inc.

plib_dmt_help.h

Enumerations

	Name	Description
	DMT_MODULE_ID	Identifies the supported DMT modules.

Section

Data Types & Constants

Device Control Peripheral Library

This section describes the Device Control Peripheral Library.

Introduction

Device Control Peripheral Library for Microchip microcontrollers.

Description

This library provides a low-level abstraction of the Device Control module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby abstracting hardware differences from one microcontroller variant to another.

Using the Library

This topic describes the basic architecture of the Device Control Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_devcon.h`

The interface to the Device Control Peripheral Library is defined in the `plib_devcon.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Device Control Peripheral Library must include `peripheral.h`.

Library File:

The Device Control Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

Hardware Abstraction Model

This library provides a low-level abstraction of the Device Control module on Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The Device Control (DEVCON) peripheral library is a collection of functions that may be used by many different modules to perform tasks not specific to any given module. These functions include:

- System Locking and Unlocking
- JTAG, and Trace enable/disable
- Reading of device ID and version
- Other general device configuration settings

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Device Control module.

Library Interface Section	Description
System Functions	APIs that are used for general configuration of the module, such as enabling and disabling the Device Control Peripheral Library.
Data Types and Constants	These data types and constants are required while interacting and configuring the Device Control Peripheral Library.
Feature Existence Functions	These functions identify whether a particular feature exists on the Device Control Peripheral Library.

How the Library Works

Provides information on how the library works.

Description

System Locking an Unlocking

Many operations require the system to be unlocked prior to the operation being performed. This is to prevent catastrophic failure due to an inadvertent register write. These operations include but are not limited to:

- Software reset
- Clock, oscillator and peripheral bus frequency changes
- System bus access permissions
- PPS and PMD control register locking

The following functions provide simple APIs to perform system lock/unlock:

- PLIB_DEVCON_SystemLock, PLIB_DEVCON_SystemUnlock
- PLIB_DEVCON_DeviceRegistersLock, PLIB_DEVCON_DeviceRegistersUnlock (PPS/PMD)

JTAG and Trace

JTAG and Trace are enabled/disabled using the following functions:

- PLIB_DEVCON_JTAGPortEnable, PLIB_DEVCON_JTAGPortEnable
- PLIB_DEVCON_2WireJTAGEnableTDO, PLIB_DEVCON_2WireJTAGDisableTDO
- PLIB_DEVCON_TraceOutputEnable, PLIB_DEVCON_TraceOutputDisable

Device ID and Version Number

The device ID and version number are obtained using the following functions:

- PLIB_DEVCON_DeviceIdGet
- PLIB_DEVCON_DeviceVersionGet

General Device Configuration Settings

The following functions provide general device configuration settings:

- PLIB_DEVCON_AlternateClockEnable, PLIB_DEVCON_AlternateClockDisable
- PLIB_DEVCON_FlashErrCorrectionModeSet
- PLIB_DEVCON_IgnoreDebugFreezeEnable, PLIB_DEVCON_IgnoreDebugFreezeDisable

Configuring the Library

The library is configured for the supported Device Control module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) System Functions

	Name	Description
	PLIB_DEVCON_2WireJTAGDisableTDO	Disables 2-Wire JTAG protocol use of TDO.
	PLIB_DEVCON_2WireJTAGEnableTDO	Enables 2-Wire JTAG protocol to use TDO.
	PLIB_DEVCON_AlternateClockDisable	Disables the alternate clock source for Input Capture or Output Compare modules, The primary clock source will be used instead.
	PLIB_DEVCON_AlternateClockEnable	Selects the alternate clock source for Input Capture or Output Compare modules.
	PLIB_DEVCON_DeviceIdGet	Gets the device identifier.
	PLIB_DEVCON_DeviceRegistersLock	Locks device module registers, preventing modifications to the registers.
	PLIB_DEVCON_DeviceRegistersUnlock	Unlocks device module registers, allowing modifications to the registers.
	PLIB_DEVCON_DeviceVersionGet	Gets the device version.
	PLIB_DEVCON_FlashErrCorrectionModeSet	Sets Flash error correction operation.
	PLIB_DEVCON_IgnoreDebugFreezeDisable	Module stops when commanded by debugger.
	PLIB_DEVCON_IgnoreDebugFreezeEnable	Allows module to ignore FREEZE command from debugger and continue running.
	PLIB_DEVCON_JTAGPortDisable	Disables the JTAG port on the device.
	PLIB_DEVCON_JTAGPortEnable	Enables the JTAG port on the device.
	PLIB_DEVCON_SystemLock	Executes the system lock sequence.
	PLIB_DEVCON_SystemUnlock	Executes the system unlock sequence.

	PLIB_DEVCON_TraceOutputDisable	Disables trace outputs and the stop trace clock.
	PLIB_DEVCON_TraceOutputEnable	Enables trace outputs and the start trace clock.
	PLIB_DEVCON_USBPHYSleepModeSet	Selects USB PHY clocking during Sleep mode.
	PLIB_DEVCON_AnalogIOChargePumpDisable	Disables the I/O Analog Charge Pump on the device.
	PLIB_DEVCON_AnalogIOChargePumpEnable	Enables the I/O Analog Charge Pump on the device.
	PLIB_DEVCON_ExistsAnalogChargePumpControl	Identifies whether the I/O Analog Charge Pump feature exists on the DEVCON module.
	PLIB_DEVCON_BootExtSelect	Routes SPI0 pins to the PIC32WK pads.
	PLIB_DEVCON_BootIPFSelect	Routes SPI0 pins to In-Package Flash.
	PLIB_DEVCON_HSUARTDisable	Disables High Speed UART.
	PLIB_DEVCON_HSUARTEnable	Enables High Speed UART.
	PLIB_DEVCON_OTPConfigLock	Locks or unlocks the configuration registers.
	PLIB_DEVCON_OTPConfigUnlock	unlocks the configuration registers.

b) MPLL Functions

	Name	Description
	PLIB_DEVCON_MPLLDisable	Disables the MPLL.
	PLIB_DEVCON_MPLLEnable	Enables the MPLL.
	PLIB_DEVCON_MPLLInputDivSet	Sets the MPLL Input Divider bits.
	PLIB_DEVCON_MPLLIsReady	Reads MPLL status.
	PLIB_DEVCON_MPLLMultiplierSet	Sets the MPLL Multiplier bits.
	PLIB_DEVCON_MPLLODiv1Set	Sets the MPLL output divider 1 bits.
	PLIB_DEVCON_MPLLODiv2Set	Sets the MPLL output divider 2 bits.
	PLIB_DEVCON_MPLLVrefSet	Sets the VREF control setting.
	PLIB_DEVCON_MPLLVregDisable	Disables the MPLL VREG.
	PLIB_DEVCON_MPLLVregEnable	Enables the MPLL VREG.
	PLIB_DEVCON_MPLLVregIsReady	Reads the MPLL VREG status.

c) Feature Existence Functions

	Name	Description
	PLIB_DEVCON_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsDeviceRegsLockUnlock	Identifies whether the DeviceRegsLockUnlock feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsDeviceVerAndId	Identifies whether the DeviceVerAndId feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsECCModes	Identifies whether the ECCModes feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsIgnoreDebugFreeze	Identifies whether the IgnoreDebugFreeze feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsJTAGEnable	Identifies whether the JTAGEnable feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsJTAGUsesTDO	Identifies whether the JTAGUsesTDO feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsSystemLockUnlock	Identifies whether the SysLockUnlock feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsTraceOutput	Identifies whether the TraceOutput feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsUSB_PHY_SleepModeSet	Identifies whether the USB_PHY_SleepModeSet feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsMPLL	Identifies whether the MPLL feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsBootSelection	Identifies whether the BootSelection feature exists on the DEVCON module
	PLIB_DEVCON_ExistsHSUARTControl	Identifies whether the HSUARTControl feature exists on the DEVCON module
	PLIB_DEVCON_ExistsOTPConfigLockUnlock	Identifies whether the OTPConfigLockUnlock feature exists on the DEVCON module

d) Data Types and Constants

	Name	Description
	DEVCON_ALT_CLOCK_TARGET	Selects Input Capture or Output Compare modules.
	DEVCON_DEBUG_PERIPHERAL	Sets modules to ignore debugger's freeze command.
	DEVCON_ECC_CONFIG	Selects how ECC is applied to Flash memory.
	DEVCON_REGISTER_SET	Selects module registers for write lock or unlock.
	DEVCON_USB_SLEEP_MODE	Selects whether the USB clock operates in Sleep mode.
	DEVCON_MODULE_ID	Identifies the supported DEVCON modules.

	DEVCON_MPLL_OUTPUT_DIVIDER	Specifies the MPLL Output divider bits.
	DEVCON_MPLL_VREF_CONTROL	VREF control.

Description

This section describes the Application Programming Interface (API) functions of the Device Control Peripheral Library.
Refer to each section for a detailed description.

a) System Functions

PLIB_DEVCON_2WireJTAGDisableTDO Function

Disables 2-Wire JTAG protocol use of TDO.

File

plib_devcon.h

C

```
void PLIB_DEVCON_2WireJTAGDisableTDO(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables 2-Wire JTAG protocol use of TDO.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_2WireJTAGDisableTDO(DEVCON_MODULE_ID index)

PLIB_DEVCON_2WireJTAGEnableTDO Function

Enables 2-Wire JTAG protocol to use TDO.

File

plib_devcon.h

C

```
void PLIB_DEVCON_2WireJTAGEnableTDO(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables 2-Wire JTAG protocol to use TDO.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_2WireJTAGEnableTDO(DEVCON_MODULE_ID index)

PLIB_DEVCON_AlternateClockDisable Function

Disables the alternate clock source for Input Capture or Output Compare modules, The primary clock source will be used instead.

File

plib_devcon.h

C

```
void PLIB_DEVCON_AlternateClockDisable(DEVCON_MODULE_ID index, DEVCON_ALT_CLOCK_TARGET targetAltClock);
```

Returns

None.

Description

This function disables the alternate clock source for the Input Capture or Output Compare modules. The primary clock source will be used instead.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed.

Preconditions

None.

Example

```
// Call system service to unlock oscillator
PLIB_DEVCON_AlternateClockDisable(DEVCON_ID, DEVCON_ALT_CLOCK_IC || DEVCON_ALT_CLOCK_OC );
```

Parameters

Parameters	Description
index	Always DEVCON_ID
targetAltClock	DEVCON_ALT_CLOCK_IC or DEVCON_ALT_CLOCK_OC

Function

void PLIB_DEVCON_AlternateClockDisable(DEVCON_MODULE_ID index,
DEVCON_ALT_CLOCK_TARGET targetAltClock)

PLIB_DEVCON_AlternateClockEnable Function

Selects the alternate clock source for Input Capture or Output Compare modules.

File

plib_devcon.h

C

```
void PLIB_DEVCON_AlternateClockEnable(DEVCON_MODULE_ID index, DEVCON_ALT_CLOCK_TARGET targetAltClock);
```

Returns

None.

Description

This function selects the alternate clock source for the Input Capture or Output Compare modules.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed.

Preconditions

None.

Example

```
// Call system service to unlock oscillator
PLIB_DEVCON_AlternateClockEnable(DEVCON_ID, DEVCON_ALT_CLOCK_IC || DEVCON_ALT_CLOCK_OC );
```

Parameters

Parameters	Description
index	Always DEVCON_ID
targetAltClock	DEVCON_ALT_CLOCK_IC or DEVCON_ALT_CLOCK_OC

Function

void PLIB_DEVCON_AlternateClockEnable(DEVCON_MODULE_ID index,
DEVCON_ALT_CLOCK_TARGET targetAltClock)

PLIB_DEVCON_DeviceIdGet Function

Gets the device identifier.

File

plib_devcon.h

C

```
uint32_t PLIB_DEVCON_DeviceIdGet(DEVCON_MODULE_ID index);
```

Returns

The version of the device.

Description

This function returns the device identifier.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

uint32_t PLIB_DEVCON_DeviceIdGet(DEVCON_MODULE_ID index)

PLIB_DEVCON_DeviceRegistersLock Function

Locks device module registers, preventing modifications to the registers.

File

plib_devcon.h

C

```
void PLIB_DEVCON_DeviceRegistersLock(DEVCON_MODULE_ID index, DEVCON_REGISTER_SET registersToLock);
```

Returns

None.

Description

This function locks device module registers, preventing modifications to the registers.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed.

Preconditions

None.

Example

```
// Call system service to unlock oscillator
PLIB_DEVCON_DeviceRegistersLock(DEVCON_ID,  DEVCON_ALL_REGISTERS);
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID,DEVCON_PPS_REGISTERS);
```

Parameters

Parameters	Description
index	Always DEVCON_ID
registersToLock	element from DEVCON_REGISTER_SET, which can be ORed together

Function

void PLIB_DEVCON_DeviceRegistersLock(DEVCON_MODULE_ID index,
DEVCON_REGISTER_SET registersToLock)

PLIB_DEVCON_DeviceRegistersUnlock Function

Unlocks device module registers, allowing modifications to the registers.

File

plib_devcon.h

C

```
void PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_MODULE_ID index, DEVCON_REGISTER_SET registersToLock);
```

Returns

None.

Description

This function unlocks device module registers, allowing modifications to the registers.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed.

Preconditions

None.

Example

```
// Call system service to unlock oscillator
PLIB_DEVCON_DeviceRegistersLock(DEVCON_ID,  DEVCON_ALL_REGISTERS);
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID,DEVCON_PPS_REGISTERS);
```

Parameters

Parameters	Description
index	Always DEVCON_ID
registersToLock	element from DEVCON_REGISTER_SET, which can be ORed together

Function

void PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_MODULE_ID index,
DEVCON_REGISTER_SET registersToLock)

PLIB_DEVCON_DeviceVersionGet Function

Gets the device version.

File

plib_devcon.h

C

```
uint8_t PLIB_DEVCON_DeviceVersionGet(DEVCON_MODULE_ID index);
```

Returns

The version of the device.

Description

This functions returns the device version.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

uint8_t PLIB_DEVCON_DeviceVersionGet(DEVCON_MODULE_ID index)

PLIB_DEVCON_FlashErrCorrectionModeSet Function

Sets Flash error correction operation.

File

plib_devcon.h

C

```
void PLIB_DEVCON_FlashErrCorrectionModeSet(DEVCON_MODULE_ID index, DEVCON_ECC_CONFIG flashECC);
```

Returns

None.

Description

This function sets Flash error correction operation.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. Once ECC has been locked, it cannot be unlocked except through a system reset.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Always DEVCON_ID

flashECC	DEVCON_ECC_DISABLED, DEVCON_ECC_DISABLED_LOCKED, DEVCON_DYN_ECC_ENABLED_LOCKED, or DEVCON_FLASH_ECC_ENABLED_LOCKED

Function

void PLIB_DEVCON_FlashErrCorrectionModeSet(DEVCON_MODULE_ID index,
DEVCON_ECC_CONFIG flashECC)

PLIB_DEVCON_IgnoreDebugFreezeDisable Function

Module stops when commanded by debugger.

File

plib_devcon.h

C

```
void PLIB_DEVCON_IgnoreDebugFreezeDisable(DEVCON_MODULE_ID index, DEVCON_DEBUG_PERIPHERAL myPeripheral);
```

Returns

None.

Description

This function stops the module when commanded by the debugger.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. Peripherals can be ORed together.

Preconditions

None.

Example

```
PLIB_DEVCON_DebugIgnoreFreezeEnable(DEVCON_ID,DEVCON_DEBUG_ALL);
PLIB_DEVCON_DebugIgnoreFreezeDisable(DEVCON_ID,DEVCON_DEBUG_SPI1);
```

Parameters

Parameters	Description
index	Always DEVCON_ID
myPeripheral	DEVCON_DEBUG_USB, DEVCON_DEBUG_UART1, DEVCON_DEBUG_UART2, DEVCON_DEBUG_SPI1, or DEVCON_DEBUG_ALL (for all modules)

Function

void PLIB_DEVCON_IgnoreDebugFreezeDisable(DEVCON_MODULE_ID index,
DEVCON_DEBUG_PERIPHERAL myPeripheral)

PLIB_DEVCON_IgnoreDebugFreezeEnable Function

Allows module to ignore FREEZE command from debugger and continue running.

File

plib_devcon.h

C

```
void PLIB_DEVCON_IgnoreDebugFreezeEnable(DEVCON_MODULE_ID index, DEVCON_DEBUG_PERIPHERAL myPeripheral);
```

Returns

None.

Description

This function allows the module to ignore the FREEZE command from the debugger and continue running.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. Peripherals can be ORed together.

Preconditions

None.

Example

```
PLIB_DEVCON_DebugIgnoreFreezeEnable(DEVCON_ID,DEVCON_DEBUG_ALL);
PLIB_DEVCON_DebugIgnoreFreezeDisable(DEVCON_ID,DEVCON_DEBUG_SPI1);
```

Parameters

Parameters	Description
index	Always DEVCON_ID
myPeripheral	DEVCON_DEBUG_USB, DEVCON_DEBUG_UART1, DEVCON_DEBUG_UART2, DEVCON_DEBUG_SPI1, or DEVCON_DEBUG_ALL (for all modules)

Function

void PLIB_DEVCON_IgnoreDebugFreezeEnable(DEVCON_MODULE_ID index,
DEVCON_DEBUG_PERIPHERAL myPeripheral)

PLIB_DEVCON_JTAGPortDisable Function

Disables the JTAG port on the device.

File

plib_devcon.h

C

```
void PLIB_DEVCON_JTAGPortDisable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables the JTAG port on the device.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_JTAGPortDisable(DEVCON_MODULE_ID index)

PLIB_DEVCON_JTAGPortEnable Function

Enables the JTAG port on the device.

File

plib_devcon.h

C

```
void PLIB_DEVCON_JTAGPortEnable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables the JTAG port on the device.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_JTAGPortEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_SystemLock Function

Executes the system lock sequence.

File

plib_devcon.h

C

```
void PLIB_DEVCON_SystemLock(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function executes the system lock sequence.

Remarks

Should only be called after PLIB_DEVCON_SystemUnlock and the action that required the unlock have been performed.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_SystemLock(DEVCON_MODULE_ID index)

PLIB_DEVCON_SystemUnlock Function

Executes the system unlock sequence.

File

plib_devcon.h

C

```
void PLIB_DEVCON_SystemUnlock(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function executes the system unlock sequence.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_SystemUnlock(DEVCON_MODULE_ID index)

PLIB_DEVCON_TraceOutputDisable Function

Disables trace outputs and the stop trace clock.

File

plib_devcon.h

C

```
void PLIB_DEVCON_TraceOutputDisable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables trace outputs and the stop trace clock.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_TraceOutputDisable(DEVCON_MODULE_ID index)

PLIB_DEVCON_TraceOutputEnable Function

Enables trace outputs and the start trace clock.

File

plib_devcon.h

C

```
void PLIB_DEVCON_TraceOutputEnable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables trace outputs and the start trace clock (trace probe must be present).

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_TraceOutputEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_USBPHYSleepModeSet Function

Selects USB PHY clocking during Sleep mode.

File

plib_devcon.h

C

```
void PLIB_DEVCON_USBPHYSleepModeSet(DEVCON_MODULE_ID index, DEVCON_USB_SLEEP_MODE sleepOrRun);
```

Returns

None.

Description

This function selects USB PHY clocking during Sleep mode.

Remarks

The feature is not supported on all devices. Refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed.

Preconditions

None.

Example

```
// Call system service to unlock oscillator
PLIB_DEVCON_USBPHYSleepModeSet(DEVCON_ID,DEVCON_USB_NO_CLOCK_IN_SLEEP)
```

Parameters

Parameters	Description
index	Always DEVCON_ID
sleepOrRun	DEVCON_USB_NO_CLOCK_IN_SLEEP or DEVCON_USB_CLOCK_IN_SLEEP

Function

void PLIB_DEVCON_USBPHYSleepModeSet(DEVCON_MODULE_ID index,
DEVCON_USB_SLEEP_MODE sleepOrRun)

PLIB_DEVCON_AnalogIOChargePumpDisable Function

Disables the I/O Analog Charge Pump on the device.

File

plib_devcon.h

C

```
void PLIB_DEVCON_AnalogIOChargePumpDisable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables the I/O Analog Charge Pump on the device.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_AnalogIOChargePumpDisable(DEVCON_MODULE_ID index)

PLIB_DEVCON_AnalogIOChargePumpEnable Function

Enables the I/O Analog Charge Pump on the device.

File

plib_devcon.h

C

```
void PLIB_DEVCON_AnalogIOChargePumpEnable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables the I/O Analog Charge Pump on the device.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Always DEVCON_ID

Function

void PLIB_DEVCON_AnalogIOChargePumpEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsAnalogChargePumpControl Function

Identifies whether the I/O Analog Charge Pump feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsAnalogChargePumpControl(DEVCON_MODULE_ID index);
```

Returns

- true - The I/O Analog Charge Pump feature feature is supported on the device
- false - The I/O Analog Charge Pump feature feature is not supported on the device

Description

This function identifies whether the I/O Analog Charge Pump feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_AnalogIOChargePumpEnable

- PLIB_DEVCON_AnalogIOChargePumpDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsAnalogChargePumpControl(DEVCON_MODULE_ID index)

PLIB_DEVCON_BootExtSelect Function

Routes SPI0 pins to the PIC32WK pads.

File

plib_devcon.h

C

```
void PLIB_DEVCON_BootExtSelect(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function routes the SPI0 pins to the PIC32WK pads.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_DEVCON_BootExtSelect(DEVCON_MODULE_ID index)

PLIB_DEVCON_BootIPFSelect Function

Routes SPI0 pins to In-Package Flash.

File

plib_devcon.h

C

```
void PLIB_DEVCON_BootIPFSelect(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function routes the SPI0 pins to In-Package Flash.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_DEVCON_BootIPFSelect(DEVCON_MODULE_ID index)

PLIB_DEVCON_HSUARTDisable Function

Disables High Speed UART.

File

plib_devcon.h

C

```
void PLIB_DEVCON_HSUARTDisable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables the high speed UART and UART 1 can be configured with PPS.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_DEVCON_HSUARTDisable(DEVCON_MODULE_ID index)

PLIB_DEVCON_HSUARTEnable Function

Enables High Speed UART.

File

plib_devcon.h

C

```
void PLIB_DEVCON_HSUARTEnable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables the high speed UART and UART 1 will be using dedicated UART pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_DEVCON_HSUARTEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_OTPConfigLock Function

Locks or unlocks the configuration registers.

File

plib_devcon.h

C

```
void PLIB_DEVCON_OTPConfigLock(DEVCON_MODULE_ID index, DEVCON_CFGLOCK lockType);
```

Returns

None.

Description

This function locks or unlocks the configuration register as per the locktype specified.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
lockType	Enumerated config lock value

Function

void PLIB_DEVCON_OTPConfigLock(DEVCON_MODULE_ID index, DEVCON_CFGLOCK lockType)

PLIB_DEVCON_OTPConfigUnlock Function

unlocks the configuration registers.

File

plib_devcon.h

C

```
void PLIB_DEVCON_OTPConfigUnlock(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function unlocks the configuration register provided, the CFGLOCK field is not to a value to locked until the reset.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_DEVCON_OTPConfigUnlock(DEVCON_MODULE_ID index)

b) MPLL Functions

PLIB_DEVCON_MPLLDisable Function

Disables the MPLL.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLDisable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables the MPLL.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_MPLLDisable(DEVCON_MODULE_ID index)

PLIB_DEVCON_MPLLEnable Function

Enables the MPLL.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLEnable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables the MPLL.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_MPLLEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_MPLLInputDivSet Function

Sets the MPLL Input Divider bits.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLInputDivSet(DEVCON_MODULE_ID index, uint8_t value);
```

Returns

None.

Description

This function sets the MPLL Input Divider bits.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
value	Multiplier value (between 1 & 63)

Function

PLIB_DEVCON_MPLLInputDivSet(DEVCON_MODULE_ID index, uint8_t value)

PLIB_DEVCON_MPLLIsReady Function

Reads MPLL status.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_MPLLIsReady(DEVCON_MODULE_ID index);
```

Returns

- true - MPLL clock is stable
- false - MPLL clock is not stable

Description

This function reads MPLL status.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_MPLLIsReady(DEVCON_MODULE_ID index)

PLIB_DEVCON_MPLLMultiplierSet Function

Sets the MPLL Multiplier bits.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLMultiplierSet(DEVCON_MODULE_ID index, uint8_t value);
```

Returns

None.

Description

This function sets the MPLL Multiplier bits.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
value	Multiplier value (between 16 & 160)

Function

PLIB_DEVCON_MPLLMultiplierSet(DEVCON_MODULE_ID index, uint8_t value)

PLIB_DEVCON_MPLLODiv1Set Function

Sets the MPLL output divider 1 bits.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLODiv1Set(DEVCON_MODULE_ID index, DEVCON_MPLL_OUTPUT_DIVIDER bits);
```

Returns

None.

Description

This function sets the MPLL output divider 1 bits.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
bits	Enumerated divider value

Function

PLIB_DEVCON_MPLLODiv1Set(DEVCON_MODULE_ID index, DEVCON_MPLL_OUTPUT_DIVIDER bits)

PLIB_DEVCON_MPLLODiv2Set Function

Sets the MPLL output divider 2 bits.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLODiv2Set(DEVCON_MODULE_ID index, DEVCON_MPLL_OUTPUT_DIVIDER bits);
```

Returns

None.

Description

This function sets the MPLL output divider 2 bits.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
bits	Enumerated divider value

Function

PLIB_DEVCON_MPLLODiv2Set(DEVCON_MODULE_ID index, DEVCON_MPLL_OUTPUT_DIVIDER bits)

PLIB_DEVCON_MPLLVrefSet Function

Sets the VREF control setting.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLVrefSet(DEVCON_MODULE_ID index, DEVCON_MPLL_VREF_CONTROL vref);
```

Returns

None.

Description

This function sets the VREF control setting.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
value	Enumerated VREF Control setting

Function

PLIB_DEVCON_MPLLVrefSet(DEVCON_MODULE_ID index, DEVCON_MPLL_VREF_CONTROL vref)

PLIB_DEVCON_MPLLVregDisable Function

Disables the MPLL VREG.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLVregDisable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function disables the MPLL VREG.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_MPLLVregDisable(DEVCON_MODULE_ID index)

PLIB_DEVCON_MPLLVregEnable Function

Enables the MPLL VREG.

File

plib_devcon.h

C

```
void PLIB_DEVCON_MPLLVregEnable(DEVCON_MODULE_ID index);
```

Returns

None.

Description

This function enables the MPLL VREG.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_MPLLVregEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_MPLLVregIsReady Function

Reads the MPLL VREG status.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_MPLLVregIsReady(DEVCON_MODULE_ID index);
```

Returns

- true - MPLL VREG is ready
- false - MPLL VREG is not ready

Description

This function reads the MPLL VREG status.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_MPLLVregIsReady(DEVCON_MODULE_ID index)

c) Feature Existence Functions

PLIB_DEVCON_ExistsAlternateClock Function

Identifies whether the AlternateClock feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsAlternateClock(DEVCON_MODULE_ID index);
```

Returns

- true - The AlternateClock feature is supported on the device
- false - The AlternateClock feature is not supported on the device

Description

This function identifies whether the AlternateClock feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_AlternateClockEnable
- PLIB_DEVCON_AlternateClockDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsAlternateClock(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsDeviceRegsLockUnlock Function

Identifies whether the DeviceRegsLockUnlock feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsDeviceRegsLockUnlock(DEVCON_MODULE_ID index);
```

Returns

- true - The DeviceRegsLockUnlock feature is supported on the device
- false - The DeviceRegsLockUnlock feature is not supported on the device

Description

This function identifies whether the DeviceRegsLockUnlock feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_DeviceRegistersLock
- PLIB_DEVCON_DeviceRegistersUnlock

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsDeviceRegsLockUnlock(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsDeviceVerAndId Function

Identifies whether the DeviceVerAndId feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsDeviceVerAndId(DEVCON_MODULE_ID index);
```

Returns

- true - The DeviceVerAndId feature is supported on the device
- false - The DeviceVerAndId feature is not supported on the device

Description

This function identifies whether the DeviceVerAndId feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_DeviceVersionGet
- PLIB_DEVCON_DeviceIdGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsDeviceVerAndId(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsECCModes Function

Identifies whether the ECCModes feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsECCModes(DEVCON_MODULE_ID index);
```

Returns

- true - The ECCModes feature is supported on the device
- false - The ECCModes feature is not supported on the device

Description

This function identifies whether the ECCModes feature is available on the DEVCON module. When this function returns true, this function is supported on the device:

- PLIB_DEVCON_FlashErrCorrectionModeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsECCModes(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsIgnoreDebugFreeze Function

Identifies whether the IgnoreDebugFreeze feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsIgnoreDebugFreeze(DEVCON_MODULE_ID index);
```

Returns

- true - The IgnoreDebugFreeze feature is supported on the device
- false - The IgnoreDebugFreeze feature is not supported on the device

Description

This function identifies whether the IgnoreDebugFreeze feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_DebugIgnoreFreezeEnable
- PLIB_DEVCON_IgnoreDebugFreezeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsIgnoreDebugFreeze(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsJTAGEnable Function

Identifies whether the JTAGEnable feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsJTAGEnable(DEVCON_MODULE_ID index);
```

Returns

- true - The JTAGEnable feature is supported on the device
- false - The JTAGEnable feature is not supported on the device

Description

This function identifies whether the JTAGEnable feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_JTAGPortEnable
- PLIB_DEVCON_JTAGPortDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsJTAGEnable(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsJTAGUsesTDO Function

Identifies whether the JTAGUsesTDO feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsJTAGUsesTDO(DEVCON_MODULE_ID index);
```

Returns

- true - The JTAGUsesTDO feature is supported on the device
- false - The JTAGUsesTDO feature is not supported on the device

Description

This function identifies whether the JTAGUsesTDO feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_2WireJTAGEnableTDO
- PLIB_DEVCON_2WireJTAGDisableTDO

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsJTAGUsesTDO(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsSystemLockUnlock Function

Identifies whether the SysLockUnlock feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsSystemLockUnlock(DEVCON_MODULE_ID index);
```

Returns

- true - The SysLockUnlock feature is supported on the device
- false - The SysLockUnlock feature is not supported on the device

Description

This function identifies whether the SysLockUnlock feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_SystemUnlock
- PLIB_DEVCON_SystemLock

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsSystemLockUnlock(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsTraceOutput Function

Identifies whether the TraceOutput feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsTraceOutput(DEVCON_MODULE_ID index);
```

Returns

- true - The TraceOutput feature is supported on the device
- false - The TraceOutput feature is not supported on the device

Description

This function identifies whether the TraceOutput feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_TraceOutputEnable
- PLIB_DEVCON_TraceOutputDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsTraceOutput(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsUSB_PHY_SleepModeSet Function

Identifies whether the USB_PHY_SleepModeSet feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsUSB_PHY_SleepModeSet(DEVCON_MODULE_ID index);
```

Returns

- true - The USB_PHY_SleepModeSet feature is supported on the device
- false - The USB_PHY_SleepModeSet feature is not supported on the device

Description

This function identifies whether the USB_PHY_SleepModeSet feature is available on the DEVCON module. When this function returns true, this function is supported on the device:

- PLIB_DEVCON_USBPHYSleepModeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsUSB_PHY_SleepModeSet(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsMPLL Function

Identifies whether the MPLL feature exists on the DEVCON module.

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsMPLL(DEVCON_MODULE_ID index);
```

Returns

- true - The MPLL feature is supported on the device
- false - The MPLL feature is not supported on the device

Description

This function identifies whether the MPLL feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_MPLLIsReady
- PLIB_DEVCON_MPLLEnable
- PLIB_DEVCON_MPLLDisable
- PLIB_DEVCON_MPLLODiv1Set
- PLIB_DEVCON_MPLLODiv2Set
- PLIB_DEVCON_MPLLVregIsReady
- PLIB_DEVCON_MPLLVregEnable
- PLIB_DEVCON_MPLLVregDisable
- PLIB_DEVCON_MPLLMultiplierSet
- PLIB_DEVCON_MPLLInputDivSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsMPLL(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsBootSelection Function

Identifies whether the BootSelection feature exists on the DEVCON module

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsBootSelection(DEVCON_MODULE_ID index);
```

Returns

- true - The BootSelection feature is supported on the device

- false - The BootSelection feature is not supported on the device

Description

This function identifies whether the BootSelection feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_BootIPFSelect
- PLIB_DEVCON_BootExtSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsBootSelection(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsHSUARTControl Function

Identifies whether the HSUARTControl feature exists on the DEVCON module

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsHSUARTControl(DEVCON_MODULE_ID index);
```

Returns

- true - The HSUARTControl feature is supported on the device
- false - The HSUARTControl feature is not supported on the device

Description

This function identifies whether the HSUARTControl feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_HSUARTEnable
- PLIB_DEVCON_HSUARTDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsHSUARTControl(DEVCON_MODULE_ID index)

PLIB_DEVCON_ExistsOTPConfigLockUnlock Function

Identifies whether the OTPConfigLockUnlock feature exists on the DEVCON module

File

plib_devcon.h

C

```
bool PLIB_DEVCON_ExistsOTPConfigLockUnlock(DEVCON_MODULE_ID index);
```

Returns

- true - The OTPConfigLockUnlock feature is supported on the device
- false - The OTPConfigLockUnlock feature is not supported on the device

Description

This function identifies whether the OTPConfigLockUnlock feature is available on the DEVCON module. When this function returns true, these functions are supported on the device:

- PLIB_DEVCON_OTPConfigLock
- PLIB_DEVCON_OTPConfigUnlock

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DEVCON_ExistsOTPConfigLockUnlock(DEVCON_MODULE_ID index)

d) Data Types and Constants

DEVCON_ALT_CLOCK_TARGET Enumeration

Selects Input Capture or Output Compare modules.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_ALT_CLOCK_IC,
  DEVCON_ALT_CLOCK_OC
} DEVCON_ALT_CLOCK_TARGET;
```

Members

Members	Description
DEVCON_ALT_CLOCK_IC	Input Capture
DEVCON_ALT_CLOCK_OC	Output Compare

Description

Alternate Clock Targets Enumeration

This enumeration selects the Input Capture or Output Compare module for enabling or disabling the alternate clock.

Remarks

None.

DEVCON_DEBUG_PERIPHERAL Enumeration

Sets modules to ignore debugger's freeze command.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_DEBUG_USB,
  DEVCON_DEBUG_UART1,
  DEVCON_DEBUG_UART2,
  DEVCON_DEBUG_SPI1,
  DEVCON_DEBUG_ALL
} DEVCON_DEBUG_PERIPHERAL;
```

Members

Members	Description
DEVCON_DEBUG_USB	USB module
DEVCON_DEBUG_UART1	UART 1
DEVCON_DEBUG_UART2	UART 2
DEVCON_DEBUG_SPI1	SPI 1
DEVCON_DEBUG_ALL	USB, UART 1, UART 2, and SPI 1

Description

Ignore Debugger Freeze for Peripheral Module(s) enumeration

This enumeration sets modules to ignore the debugger's freeze command.

Remarks

None.

DEVCON_ECC_CONFIG Enumeration

Selects how ECC is applied to Flash memory.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_ECC_DISABLED,
  DEVCON_ECC_DISABLED_LOCKED,
  DEVCON_DYN_ECC_ENABLED_LOCKED,
  DEVCON_FLASH_ECC_ENABLED_LOCKED
} DEVCON_ECC_CONFIG;
```

Members

Members	Description
DEVCON_ECC_DISABLED	ECC and dynamic ECC are disabled
DEVCON_ECC_DISABLED_LOCKED	ECC and dynamic ECC are disabled and the configuration locked
DEVCON_DYN_ECC_ENABLED_LOCKED	Dynamic Flash ECC is enabled and the configuration is locked
DEVCON_FLASH_ECC_ENABLED_LOCKED	Flash ECC is enabled, the configuration is locked, word Flash writes disabled

Description

Flash Error Correcting Code (ECC) Configuration enumeration

This enumeration selects how ECC is applied to Flash memory.

Remarks

None.

DEVCON_REGISTER_SET Enumeration

Selects module registers for write lock or unlock.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_PPS_REGISTERS,
  DEVCON_PMD_REGISTERS,
  DEVCON_PERMISSION_GROUP_REGISTERS,
  DEVCON_ALL_REGISTERS
} DEVCON_REGISTER_SET;
```

Members

Members	Description
DEVCON_PPS_REGISTERS	Peripheral Pin Select registers
DEVCON_PMD_REGISTERS	Peripheral Module Disable registers
DEVCON_PERMISSION_GROUP_REGISTERS	Permission Group registers
DEVCON_ALL_REGISTERS	All lockable registers

Description

Module Registers for Lock/Unlock enumeration.

This enumeration selects the module registers for write lock or unlock.

Remarks

Can be ORed together.

DEVCON_USB_SLEEP_MODE Enumeration

Selects whether the USB clock operates in Sleep mode.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_USB_NO_CLOCK_IN_SLEEP,
  DEVCON_USB_CLOCK_IN_SLEEP
} DEVCON_USB_SLEEP_MODE;
```

Members

Members	Description
DEVCON_USB_NO_CLOCK_IN_SLEEP	USB PHY clock is shut down when Sleep mode is active.
DEVCON_USB_CLOCK_IN_SLEEP	USP PHY clock continues to run when Sleep mode is active

Description

USB Clock In Sleep Mode Enumeration

This enumeration selects whether the USB clock operates in Sleep mode.

Remarks

None.

DEVCON_MODULE_ID Enumeration

Identifies the supported DEVCON modules.

File

help_plib_devcon.h

C

```
typedef enum {
  DEVCON_ID_0,
  DEVCON_NUMBER_OF_MODULES
} DEVCON_MODULE_ID;
```

Members

Members	Description
DEVCON_ID_0	DEVCON Module 0 ID
DEVCON_NUMBER_OF_MODULES	Number of available DEVCON modules.

Description

DEVCON Module ID

This enumeration identifies the DEVCON modules that are available on the microcontroller. This is the super set of all the possible instances that might be available on the Microchip microcontrollers.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

DEVCON_MPLL_OUTPUT_DIVIDER Enumeration

Specifies the MPLL Output divider bits.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_MPLL_ODIV_1,
  DEVCON_MPLL_ODIV_2,
  DEVCON_MPLL_ODIV_3,
  DEVCON_MPLL_ODIV_4,
  DEVCON_MPLL_ODIV_5,
  DEVCON_MPLL_ODIV_6,
  DEVCON_MPLL_ODIV_7
} DEVCON_MPLL_OUTPUT_DIVIDER;
```

Description

MPLL Output Divider bits enumeration

This enumeration specifies the MPLL Output divider bits.

Remarks

None.

DEVCON_MPLL_VREF_CONTROL Enumeration

VREF control.

File

plib_devcon.h

C

```
typedef enum {
  DEVCON_MPLL_VREF_EXT,
  DEVCON_MPLL_VREF_VDD,
  DEVCON_MPLL_VREF_VSS,
  DEVCON_MPLL_VREF_INT
} DEVCON_MPLL_VREF_CONTROL;
```

Description

MPLL VREF Control enumeration

This enumeration provides VREF control.

Remarks

None.

Files

Files

Name	Description
plib_devcon.h	Defines the Device Control Peripheral Library Interface.
help_plib_devcon.h	This is file help_plib_devcon.h.

Description

This section lists the source and header files used by the library.

plib_devcon.h

Defines the Device Control Peripheral Library Interface.

Enumerations

	Name	Description
	DEVCON_ALT_CLOCK_TARGET	Selects Input Capture or Output Compare modules.
	DEVCON_DEBUG_PERIPHERAL	Sets modules to ignore debugger's freeze command.
	DEVCON_ECC_CONFIG	Selects how ECC is applied to Flash memory.
	DEVCON_MPLL_OUTPUT_DIVIDER	Specifies the MPLL Output divider bits.
	DEVCON_MPLL_VREF_CONTROL	VREF control.
	DEVCON_REGISTER_SET	Selects module registers for write lock or unlock.
	DEVCON_USB_SLEEP_MODE	Selects whether the USB clock operates in Sleep mode.

Functions

	Name	Description
	PLIB_DEVCON_2WireJTAGDisableTDO	Disables 2-Wire JTAG protocol use of TDO.
	PLIB_DEVCON_2WireJTAGEnableTDO	Enables 2-Wire JTAG protocol to use TDO.
	PLIB_DEVCON_AlternateClockDisable	Disables the alternate clock source for Input Capture or Output Compare modules, The primary clock source will be used instead.
	PLIB_DEVCON_AlternateClockEnable	Selects the alternate clock source for Input Capture or Output Compare modules.
	PLIB_DEVCON_AnalogIOChargePumpDisable	Disables the I/O Analog Charge Pump on the device.
	PLIB_DEVCON_AnalogIOChargePumpEnable	Enables the I/O Analog Charge Pump on the device.
	PLIB_DEVCON_BootExtSelect	Routes SPI0 pins to the PIC32WK pads.
	PLIB_DEVCON_BootIPFSelect	Routes SPI0 pins to In-Package Flash.
	PLIB_DEVCON_DeviceIdGet	Gets the device identifier.
	PLIB_DEVCON_DeviceRegistersLock	Locks device module registers, preventing modifications to the registers.
	PLIB_DEVCON_DeviceRegistersUnlock	Unlocks device module registers, allowing modifications to the registers.
	PLIB_DEVCON_DeviceVersionGet	Gets the device version.
	PLIB_DEVCON_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsAnalogChargePumpControl	Identifies whether the I/O Analog Charge Pump feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsBootSelection	Identifies whether the BootSelection feature exists on the DEVCON module
	PLIB_DEVCON_ExistsDeviceRegsLockUnlock	Identifies whether the DeviceRegsLockUnlock feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsDeviceVerAndId	Identifies whether the DeviceVerAndId feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsECCModes	Identifies whether the ECCModes feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsHSUARTControl	Identifies whether the HSUARTControl feature exists on the DEVCON module
	PLIB_DEVCON_ExistsIgnoreDebugFreeze	Identifies whether the IgnoreDebugFreeze feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsJTAGEnable	Identifies whether the JTAGEnable feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsJTAGUsesTDO	Identifies whether the JTAGUsesTDO feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsMPLL	Identifies whether the MPLL feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsOTPConfigLockUnlock	Identifies whether the OTPConfigLockUnlock feature exists on the DEVCON module

	PLIB_DEVCON_ExistsSystemLockUnlock	Identifies whether the SysLockUnlock feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsTraceOutput	Identifies whether the TraceOutput feature exists on the DEVCON module.
	PLIB_DEVCON_ExistsUSB_PHY_SleepModeSet	Identifies whether the USB_PHY_SleepModeSet feature exists on the DEVCON module.
	PLIB_DEVCON_FlashErrCorrectionModeSet	Sets Flash error correction operation.
	PLIB_DEVCON_HSUARTDisable	Disables High Speed UART.
	PLIB_DEVCON_HSUARTEnable	Enables High Speed UART.
	PLIB_DEVCON_IgnoreDebugFreezeDisable	Module stops when commanded by debugger.
	PLIB_DEVCON_IgnoreDebugFreezeEnable	Allows module to ignore FREEZE command from debugger and continue running.
	PLIB_DEVCON_JTAGPortDisable	Disables the JTAG port on the device.
	PLIB_DEVCON_JTAGPortEnable	Enables the JTAG port on the device.
	PLIB_DEVCON_MPLLDisable	Disables the MPLL.
	PLIB_DEVCON_MPLLEnable	Enables the MPLL.
	PLIB_DEVCON_MPLLInputDivSet	Sets the MPLL Input Divider bits.
	PLIB_DEVCON_MPLLIsReady	Reads MPLL status.
	PLIB_DEVCON_MPLLMultiplierSet	Sets the MPLL Multiplier bits.
	PLIB_DEVCON_MPLLODiv1Set	Sets the MPLL output divider 1 bits.
	PLIB_DEVCON_MPLLODiv2Set	Sets the MPLL output divider 2 bits.
	PLIB_DEVCON_MPLLVrefSet	Sets the VREF control setting.
	PLIB_DEVCON_MPLLVregDisable	Disables the MPLL VREG.
	PLIB_DEVCON_MPLLVregEnable	Enables the MPLL VREG.
	PLIB_DEVCON_MPLLVregIsReady	Reads the MPLL VREG status.
	PLIB_DEVCON_OTPConfigLock	Locks or unlocks the configuration registers.
	PLIB_DEVCON_OTPConfigUnlock	unlocks the configuration registers.
	PLIB_DEVCON_SystemLock	Executes the system lock sequence.
	PLIB_DEVCON_SystemUnlock	Executes the system unlock sequence.
	PLIB_DEVCON_TraceOutputDisable	Disables trace outputs and the stop trace clock.
	PLIB_DEVCON_TraceOutputEnable	Enables trace outputs and the start trace clock.
	PLIB_DEVCON_USBPHYSleepModeSet	Selects USB PHY clocking during Sleep mode.

Description

Device Control (DEVCON) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Device Control Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Device Control module.

File Name

plib_devcon.h

Company

Microchip Technology Inc.

help_plib_devcon.h

Enumerations

	Name	Description
	DEVCON_MODULE_ID	Identifies the supported DEVCON modules.

Description

This is file help_plib_devcon.h.

DMA Peripheral Library Help

This section describes the Direct Memory Access (DMA) Peripheral Library.

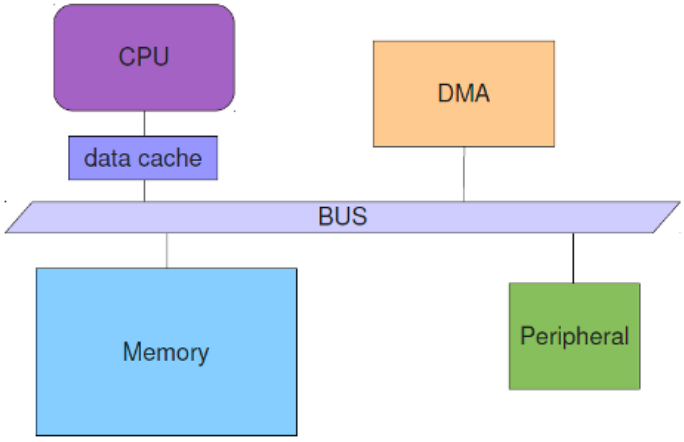
Introduction

Direct Memory Access (DMA) is a feature (sub-system) of the microcontroller that allows certain hardware sub-systems within the microcontroller to access the system memory independently of the processor (or the CPU).

Description

DMA also assists in accessing peripheral memory along with the system memory (data RAM). The DMA sub-system in the PIC family of microcontrollers is optimized for high-performance, real-time, embedded applications where determinism and system latency are priorities.

DMA Hardware Abstraction Model



Using the Library

This topic describes the basic architecture of the DMA Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_dma.h`

The interface to the DMA Peripheral Library is defined in the `plib_dma.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the DMA Peripheral Library must include `peripheral.h`.

Library File:

The DMA Peripheral Library (.a) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony?What is MPLAB Harmony? section for how the peripheral interacts with the framework.

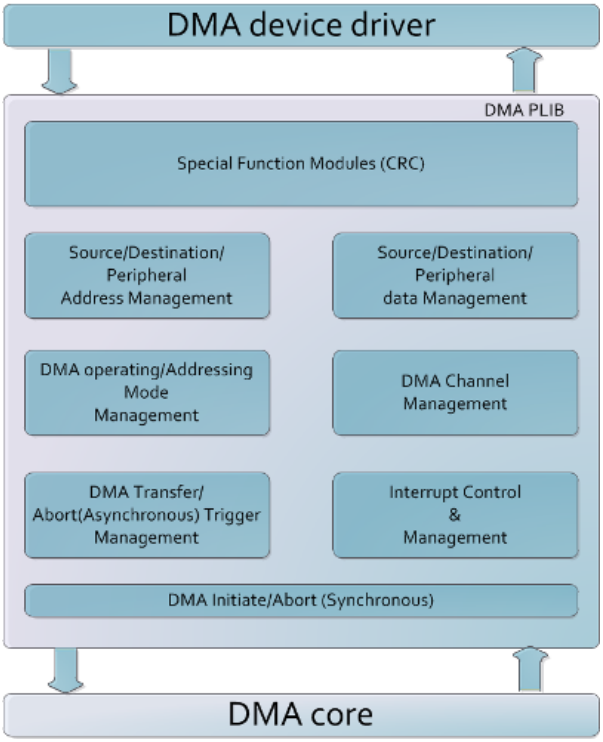
Hardware Abstraction Model

This library provides the low-level abstraction of the DMA module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

The DMA Peripheral Library software is depicted using the following block diagram.

DMA Software Abstraction Block Diagram



The DMA Peripheral Library takes requests from application software or the DMA device driver and controls the DMA core (the DMA hardware) as per the inputs passed to the library.

The major components of the DMA Peripheral Library are:

- Source/Destination/Peripheral Address Management
- Source/Destination/Peripheral Data Management
- DMA Operating/Addressing Mode Management
- DMA Channel Management
- Interrupt Control and Management

The topic Library Overview explains in detail each block of the DMA Peripheral Library software architecture.

Note: The interface provided is a superset of all the functionality of the available DMA module on the device. Refer to the **"DMA"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each DMA module on your device.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DMA module.

Library Interface Section	Description
Transfer/Abort (Synchronous)	Handles the synchronous DMA transfer start and End. This force request is controlled fully by the software.
Transfer/Abort (Asynchronous) Trigger Management	Each DMA transaction can be asynchronously initiated or aborted by an interrupt. This feature handles the required configuration.
Interrupt Control and Management	Each DMA channel can be configured to respond to events such as address error (upper address error, lower address error, etc.), transfer count half done, transfer count done, source data completely transferred, etc.
Operating/Addressing Mode Management	Before initiating a DMA transaction, the addressing and operating modes of the DMA module need to be configured (per channel basis). As an example, the DMA module can be configured for one shot mode with peripheral indirect addressing mode for a DMA transaction.
Channel Management	Each DMA channel can be individually configured for a DMA transaction. Priorities can be assigned, a channel can be enabled or disabled, channel chaining can be used, etc.
Source/Destination/Peripheral Address Management	For each DMA transaction to start, the DMA core needs to know the source and destination addresses. This feature handles this type of configuration.

Source/Destination/Peripheral Data Management	For each DMA transaction, the amount of data to be transferred, the source and destination size, the transaction type (byte/word), and transaction size can be configured by this feature.
Special Function Modules (CRC)	Certain high-end controllers support this CRC generator. This feature can be assigned to any of the available DMA channels. This feature handles the configuration of these aspects.
Status (Including Channel)	This feature specifies the application to access the status information of the DMA and each channel.
Feature Existence Functions	Lists the interface routines that determine whether or not the feature is supported by the device.

How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine which modes are supported.

General Configuration

DMA General configuration includes enabling, disabling DMA module globally. It also involves DMA suspend control and behavior in Idle mode.

Enable/Disable DMA

Enabling and disabling the DMA are the basic routines that may be called during initialization and exit routines of the application.

DMA Enable/Disable Example:

```
// Enable the DMA controller
PLIB_DMA_Enable ( DMA_ID_0 );
// Disable the DMA
PLIB_DMA_Disable ( DMA_ID_0 );
// Check the enable status of the DMA controller.
if(PLIB_DMA_IsEnabled( DMA_ID_0 ))
{
// application code
}
```

Transfer/Abort (Synchronous)

A DMA transaction can be initiated/aborted asynchronously by configuring an interrupt for a channel. Refer to Transfer/Abort (Asynchronous) Trigger Management for details.

In many instances, an application or a device driver needs to initiate a DMA transfer or abort an ongoing DMA transfer for reasons not necessarily known. To handle such cases, the DMA core supports a forceful DMA start and abort.

Note: The DMA abort is not supported by all devices. Refer to the specific device data sheet to determine availability of this feature.

When a DMA force start request is sent and the channel is already processing/pending to process an interrupt-based DMA transfer, the force transfer request will be ignored. In certain devices, a flag is raised to indicate a collision of DMA requests. Refer to Status (Including Channel) for details. In certain devices, an interrupt is issued to indicate that the active channel may not be able to keep up with the demands from the peripheral module being serviced. For more details refer to Interrupt Control and Management.

A DMA force transfer is initiated by the PLIB_DMA_StartTransferSet function.

A DMA force abort is initiated by the PLIB_DMA_AbortTransferSet function.

DMA Transfer/Abort (Synchronous) Configuration

A DMA transfer can be initiated or aborted forcefully by the software.

Example:

```
// Initiate a forced transfer on channel-3
PLIB_DMA_StartTransferSet ( DMA_ID_0, DMA_CHANNEL_3 );
// Abort a transfer on channel-3
PLIB_DMA_AbortTransferSet ( DMA_ID_0, DMA_CHANNEL_3 );
```

Transfer/Abort (Asynchronous) Trigger Management

DMA Start IRQ

Each channel can be configured to initiate a DMA transaction based on the interrupt from a source. This can be enabled/disable per channel basis. To initiate a DMA transfer upon an interrupt trigger, the IRQ number needs to be configured as the start IRQ for this channel. Which source can act as the start IRQ is device-dependent. In certain devices, any of the available IRQs can be configured as the starting IRQ, while in other devices, there is restriction on which interrupt source can act as the start IRQ. Refer to the specific device data sheet to determine feature availability.

The starting IRQ is enabled by the PLIB_DMA_ChannelXTriggerEnable function. A DMA channel can be configured to start a DMA transfer based on the event on an IRQ using the PLIB_DMA_ChannelXStartIRQSet function.

DMA Abort IRQ

Each channel can be configured to abort a DMA transaction based on the interrupt from a source. This can be enabled/disabled per channel basis. To abort a DMA transfer upon an interrupt trigger, the IRQ number needs to be configured as the abort IRQ for this channel.

The abort IRQ is enabled by the PLIB_DMA_ChannelXTriggerEnable function. A DMA channel can be configured to abort a DMA transfer based on the event on an IRQ using the ChannelXAbortIRQSet function.

DMA Pattern Matching Termination

Pattern Match Termination mode allows the user to end a transfer if a byte of data written during a transaction matches a specific pattern. This feature is useful in applications where a variable data size is required and eases the setup of the DMA channel. The UART module is a good example of where this can be effectively used. For details about the pattern data configuration refer to Channel Management**.**

DMA Pattern match mode is enabled using the PLIB_DMA_ChannelXTriggerEnable function**.** It can also be disabled using the PLIB_DMA_ChannelXTriggerDisable function**.**

DMA Transfer/Abort (Asynchronous) Configuration

A DMA transfer can also be triggered/aborted asynchronously by using an interrupt event. The interrupt event required to initiate/abort a DMA transfer for a channel needs to be set. Prior to this, the start IRQ and abort IRQ both need to be enabled.

Example:

```
// Enable the start IRQ. An interrupt event can trigger a DMA transfer.
PLIB_DMA_ChannelXTriggerEnable ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_TRIGGER_TRANSFER_START );
// Enable the abort IRQ. An interrupt event can abort a DMA transfer on channel-3
PLIB_DMA_ChannelXTriggerEnable ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_TRIGGER_TRANSFER_ABORT );
// Set the start IRQ to UART-2 RX Interrupt
PLIB_DMA_ChannelXStartIRQSet ( DMA_ID_0, DMA_CHANNEL_3, DMA_TRIGGER_USART_2_RECEIVE );
// Set the abort IRQ to UART-2 error Interrupt
PLIB_DMA_ChannelXAbortIRQSet ( DMA_ID_0, DMA_CHANNEL_3, DMA_TRIGGER_USART_2_ERROR );
```

In some devices, abort IRQ trigger is not supported. Refer to the specific device data sheet to determine availability.

DMA Transfer Pattern Match Abort Configuration

The DMA transfer can also be aborted on a pattern match trigger (supported on PIC32 devices). Refer to Transfer/Abort (Asynchronous) Trigger Management for more information on pattern match abort. To use this feature, it must first be enabled. The pattern data needs to be programmed for the required channel. Once this is done, when a DMA transfer is in progress on that particular channel, if the DMA controller comes across the data matching the programmed data the transfer is automatically aborted.

Example:

```
// Enable the pattern match abort for channel-3
PLIB_DMA_ChannelXTriggerEnable ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_TRIGGER_PATTERN_MATCH_ABORT );
// Set a pattern data to match
PLIB_DMA_ChannelXPatternDataSet ( DMA_ID_0, DMA_CHANNEL_3, 0xEE );
```

Interrupt Control and Management

The DMA module has the ability to generate interrupts reflecting the events that occur during the channel's data transfer.

The following are all of the possible interrupts generated during DMA operation. This topic provides information for configuring these interrupts.

Error Interrupts

This type of event is signalled when an address error has occurred during the channel transfer operation. An address error also can occur if a DMA operation accesses an address beyond configured address limits (low and high).

Abort Interrupts

This type of event is signalled when a DMA transfer is aborted because of a system event matching the abort IRQ configured for the current channel.

DMA Completion Interrupts (Block Completion Interrupts)

This event occurs when a channel transfers a block of configured data.

Call Complete Interrupts

This event occurs when a channel completes a cell sized data transfer.

DMA Midpoint Interrupts

This event occurs when the channel is at the midpoint of completing the data transfer.

Overrun Interrupts

When a DMA channel receives a trigger (software or hardware) while it is servicing another request, an overrun condition occurs. This indicates that the channel is being requested before its current transaction is finished. An overrun condition will not result in termination of the current transaction.

Source Address Pointer Activity Interrupts

This event occurs when the channel source pointer reached the end of the source or when it reaches the midpoint of the source depending on the way this interrupt is configured.

Destination Address Pointer Activity Interrupts

This event occurs when the channel destination pointer reached the end of the destination or when it reaches the midpoint of the destination depending on the way this interrupt is configured.

This feature helps in configuring the previously listed interrupts. Depending on the device selected, all or some of the interrupts can be individually enabled and disabled. In addition, this feature obtains the status of the interrupts previously mentioned.

All of the interrupts belonging to the DMA channel map to the corresponding channel interrupt vector. In general, each channel of the DMA has a dedicated vectored interrupt. Therefore, the ISR for each channel should internally check for the status of the previously mentioned interrupts to respond to the corresponding events.

DMA Interrupt Control and Management

Each DMA channel has a vectored interrupt. However, each DMA channel internally generates various interrupts, which will invoke the same channel interrupt based on various events. This feature helps to enable/disable them and is also used to obtain the interrupt status.

Example:

```
// Enable the channel-3 source done interrupt
PLIB_DMA_ChannelXINTSourceEnable ( DMA_ID_0, DMA_CHANNEL_3, DMA_INT_SOURCE_DONE );
// Get the status of address error on channel-3
if(PLIB_DMA_ChannelXINTSourceFlagGet ( DMA_ID_0, DMA_CHANNEL_3, DMA_INT_ADDRESS_ERROR ))
{
// application code for error handling
}
// Disable the channel abort interrupt
PLIB_DMA_ChannelXINTSourceDisable (DMA_ID_0, DMA_CHANNEL_3, DMA_INT_TRANSFER_ABORT);
```

Operating/Addressing Mode Management

DMA Operating Modes

The various operating/transfer modes supported are:

- One-Shot mode
- Repeated One-Shot mode
- Continuous mode
- Repeated Continuous mode
- Ping-Pong mode
- Null Data Write mode

One-Shot Mode

In One-Shot mode, a single transfer is performed for each trigger event. The trigger event can be synchronous (refer to Transfer/Abort (Synchronous) or asynchronous (refer to Transfer/Abort (Asynchronous) Trigger Management). When a single one-shot transfer occurs and the DMA count is decremented to zero, this disables the channel and requires the channel to be re-enabled to perform the next transaction.

Repeated One-Shot Mode

In Repeated One-Shot mode, single transfers occurs repeatedly so long as triggers are being provided. When the DMA count reaches zero, the

channel is not disabled as in One-Shot mode, but the count is reloaded and transfers begin again.

Continuous Mode

In Continuous mode, a single trigger starts a sequence of back-to-back transfers. These continue with each transfer decrementing the DMA count until it reaches zero. At this point, and like one-shot mode, the channel is disabled.

Repeated Continuous Mode

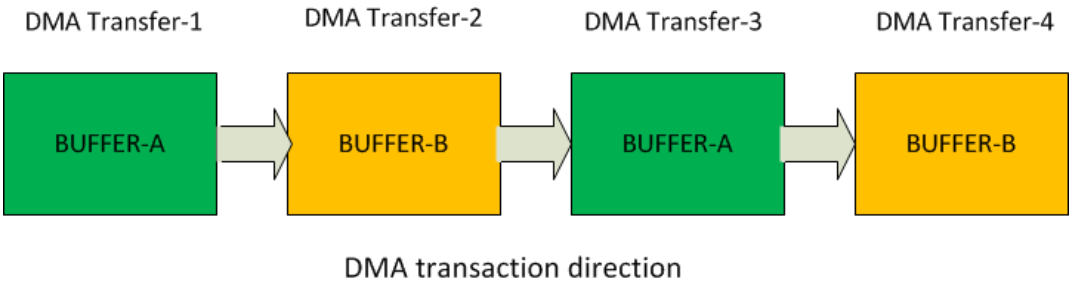
This mode can be seen as a combination of Continuous and Repeated One-Shot modes. Data transfers keep occurring so as long as triggers are provided and multiple transfers can occur with each trigger. When the DMA count reaches zero, the address and data registers are reloaded and the process is repeated.

Ping-Pong Mode

Certain devices support Ping-Pong mode, which allows the CPU to process one buffer while a second buffer operates with the DMA channel. The net result is that the CPU has the entire DMA block transfer time to process the buffer that is currently not being used by the DMA channel.

The following figure demonstrates the data flow for four DMA transactions using Ping-Pong mode.

Ping-Pong Mode



When the DMA is filling buffer-B, the CPU can process buffer-A and vice-versa.

Ping-Pong mode can be combined with One-Shot and Continuous modes (previously discussed) to create other combinations.

Null Data Write Mode

A DMA transfer operates only in one direction from the source address to the destination address. However, some communication protocols require symmetrical buffer accesses, meaning that for every read operation performed on a buffer, there must be an accompanying write operation. The Null Write mode is designed to satisfy this requirement. This mode works by transferring data from the source to the destination like any other DMA operation. Once this is done, the transferred data still with the DMA buffer is written back to the source.

DMA Addressing Modes

The addressing modes can be broadly classified into the following categories:

- Register Indirect with Post Increment Addressing mode
- Register Indirect without Post Increment Addressing mode
- Source/destination increment based on size
- Source/destination decrement based on size
- Peripheral Indirect Addressing mode

Register Indirect With Post Increment Addressing Mode

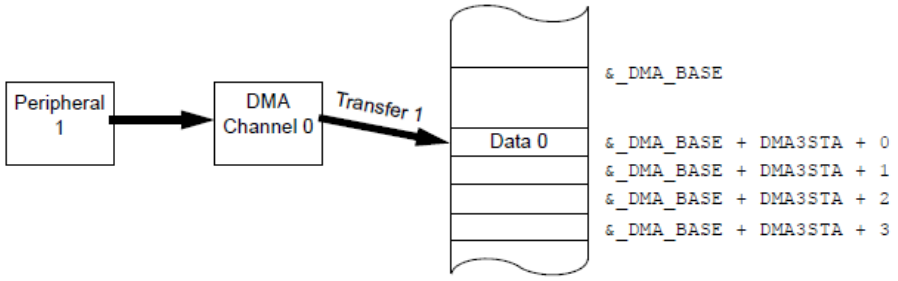
In this mode, the source data address is incremented by one location after each DMA transaction. The starting address is provided by a start address offset register.

Register Indirect Without Post Increment Addressing Mode

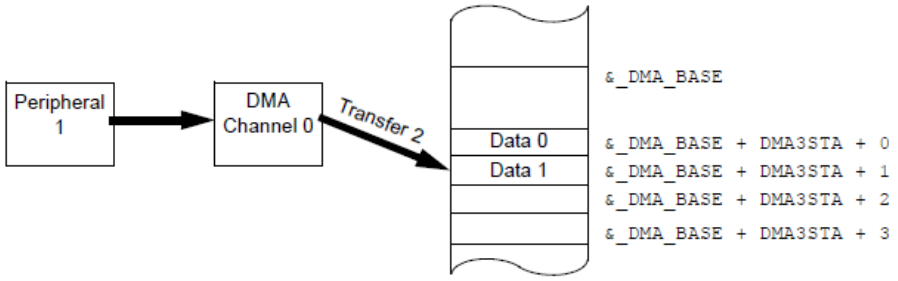
In this mode, the source data address is not updated after each DMA transaction. Therefore, the next DMA transfer is initiated from the same RAM address. The starting address is provided by a start address offset register.

Register Indirect Addressing With and Without Post-increment

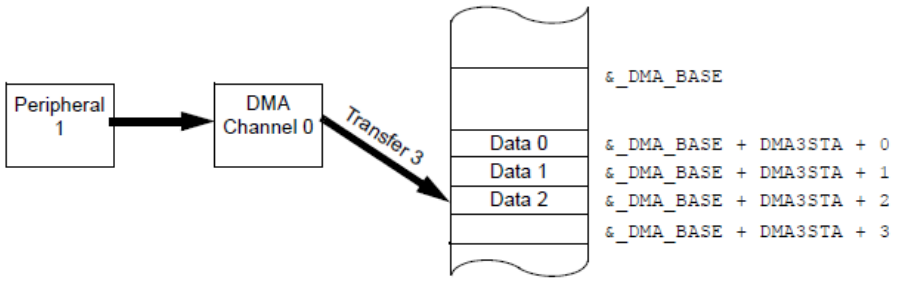
A **DMA Channel 0, First Transfer (with Post-Increment Addressing)**



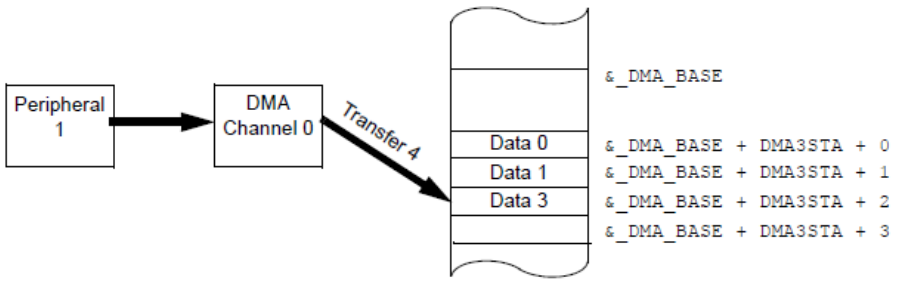
B **DMA Channel 0, Second Transfer (with Post-Increment Addressing)**



C **DMA Channel 0, Third Transfer (mode changed to "Without Post-Increment" Addressing)**



C **DMA Channel 0, Fourth Transfer (without Post-Increment Addressing)**



Source/Destination Increment Based on Size

After each DMA transaction, the source/destination (a separate configuration exist for each) address increments by the size of data (word/byte) configured.

Source/Destination Decrement Based on Size

After each DMA transaction, the source/destination (a separate configuration exist for each) address decrements by the size of data (word/byte) configured.

Peripheral Indirect Addressing Mode

Peripheral Indirect Addressing (PIA) is a special auto-incrementing mode for the transfer of data to and from a multi-level peripheral buffer. This mode is only available for specific peripherals designed with its use in mind. Refer to the specific device data sheet to determine availability of this mode.

PIA-capable peripherals - When selected, the PIA-enabled peripheral generates a short Indirect Address (IA) (size defined by the peripheral) to the DMA channel. The IA is logically ORed with either the contents of the source address or the destination address to define a specific address for the peripheral inside the DMA address space.

DMA Operating/Transfer Modes:

Refer to Operating/Addressing Mode Management for descriptions of the available operating modes.

The following code demonstrates usage of Continuous mode.

Example:

```
// Enable the DMA
PLIB_DMA_Enable ( DMA_ID_0 );
// disable the selected channel-3
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_3 );
// program the source and destination addresses
// Set the source starting address to 0x1000
PLIB_DMA_ChannelXSourceStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x1000 );
// Set the destination starting address to 0x2000
PLIB_DMA_ChannelXDestinationStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x2000 );
// Set the transfer count to 100.
PLIB_DMA_ChannelXTransferCountSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
// set the data transfer mode to one-shot
PLIB_DMA_ChannelXOperatingTransferModeSelect ( DMA_ID_0, DMA_CHANNEL_3, DMA_MODE_CONTINUOUS );
// select the source addressing mode
PLIB_DMA_ChannelXSourceAddressModeSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE );
// select the destination addressing mode
PLIB_DMA_ChannelXDestinationAddressModeSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_ADDRESSING_DESTINATION_INCREMENT_BASED_ON_SIZE );
// Enable the channel
PLIB_DMA_ChannelXEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// Initiate the transfer
PLIB_DMA_StartTransferSet ( DMA_ID_0, DMA_CHANNEL_3 );
```

DMA Addressing Modes:

DMA addressing modes can be selected by using following code example.

DMA Addressing mode Example:

```
// set the source address to increment upon each transfer
PLIB_DMA_ChannelXSourceAddressModeSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE );
// Set the destination address to decrement upon each transfer
PLIB_DMA_ChannelXDestinationAddressModeSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_ADDRESSING_DESTINATION_DECREMENT_BASED_ON_SIZE );
```

Channel Management

Each Channel in the DMA module is can be independently configured for a specific transaction with different properties.

The following aspects of the channel can be configured using this feature.

DMA Channel Enable/Disable

Each channel can be enabled or disabled to control the transactions on the channel. A DMA transfer can only occur when the channel is enabled. A DMA channel can be enabled using the PLIB_DMA_ChannelXEnable function. When a channel is disabled by calling the PLIB_DMA_ChannelXDisable function, no data transfers can occur on the channel.

DMA Channel Priority

The DMA module has a natural priority associated with each of the channels. Channel 0 has the highest priority. Each DMA channel can be assigned priority. This priority scheme changes depending on the device selected. There are three ways in which priorities are assigned in PIC microcontrollers. In the first method, two bits are allocated for each channel for priority. Therefore, priorities 0 to 3 can be assigned to channels. The second is called round-robin. In this method, the module starts by giving Channel 0 preference in any data transfer conflicts. For each successive transfer conflict, the next higher channel receives preference, continuing as a cycle through all the channels. The third method is called fixed scheme, which always gives priority to the lowest requesting channel number. The channel priority can be set using the PLIB_DMA_ChannelPrioritySelect function or the PLIB_DMA_ChannelXPrioritySelect function depending on the device selected.

DMA Channel Chaining

Channel chaining is an enhancement to the DMA channel operation. A channel (slave channel) can be chained to an adjacent channel (master channel). The slave channel will be enabled when a block transfer of the master channel completes. At this point, any event on the slave channel

will initiate a DMA transfer. If the channel has an event pending, a DMA transfer will begin immediately. The channel chaining can be enabled or disabled using the PLIB_DMA_ChannelXChainEnable function and the PLIB_DMA_ChannelXChainDisable function, respectively. The channel chaining higher/lower channel number can be configured using the PLIB_DMA_ChannelXChainToHigher function and the PLIB_DMA_ChannelXChainToLower function.

DMA Channel Transfer Direction

Each DMA transfer will have a direction. It can be memory to peripheral, peripheral to memory, or memory to memory. This can be set using the PLIB_DMA_ChannelXTransferDirectionSelect function.

DMA Channel Pattern Data

On supported devices, a DMA transfer can be aborted by a pattern match (refer to Transfer/Abort (Asynchronous) Trigger Management). The pattern data is per channel basis. It can be set using the PLIB_DMA_ChannelXPatternDataSet function.

Channel Auto Enable

This feature can be used to keep the channel active even if a block transfer or pattern match occurs. This mode is useful for applications that do repeated pattern matching. To enable this feature use the PLIB_DMA_ChannelXAutoEnable function. To disable this feature use the PLIB_DMA_ChannelXAutoDisable function.

DMA Channel Configuration

The following example configures DMA Channel 3 by setting its priority to two, enabling the auto-enable feature (refer to Channel Management section), enabling the channel chain feature and chaining Channel 3 and Channel 2, and finally enabling Channel 3.

Example:

```
// Disable the channel-3
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_3 );
// set the channel priority to 2
PLIB_DMA_ChannelXPrioritySelect  ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_PRIORITY_2 );
// Configure the channel auto enable feature
PLIB_DMA_ChannelXAutoEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// Enable the channel chain feature
PLIB_DMA_ChannelXChainEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// chain channel 3 to channel 2
PLIB_DMA_ChannelXChainToLower ( DMA_ID_0, DMA_CHANNEL_3 );
// Enable channel-3
PLIB_DMA_ChannelXEnable ( DMA_ID_0, DMA_CHANNEL_3 );
```

In some devices it is required to select the DMA transfer direction before initiating DMA transfer. This can be done using the following code.

Example:

```
// Disable channel-3
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_3 );
// Set the data transfer direction from memory to peripheral
PLIB_DMA_ChannelXTransferDirectionSelect ( DMA_ID_0, DMA_CHANNEL_3,
DMA_READ_FROM_MEMORY_WRITE_TO_PERIPHERAL );
// we can also check the current channels transfer direction
// check the data transfer direction
if ( DMA_READ_FROM_PERIPHERAL_WRITE_TO_MEMORY == PLIB_DMA_ChannelXTransferDirectionGet ( DMA_ID_0,
DMA_CHANNEL_3 ) )
{

// application code

}
```

Source/Destination/Peripheral Address Management

This topic describes configuring the source/destination/peripheral addresses involved in the DMA transfer.

Source/Destination Start Address

Before initiating a DMA transfer, the channel should be programmed with the source and the destination addresses. In supported devices, this can be done using the PLIB_DMA_ChannelXSourceStartAddressSet Function and the PLIB_DMA_ChannelXDestinationStartAddressSet Function, respectively. At any point, the programmed addresses can be read using the PLIB_DMA_ChannelXSourceStartAddressGet Function and the PLIB_DMA_ChannelXDestinationStartAddressGet Function.

Source/Destination Start Address Offset

In some devices, a dedicated region of RAM is allocated for DMA transfers typically called DMA RAM. For any DMA transfer, the source or destination address needs to be defined as an offset from the starting of the DMA RAM for such devices. Use the

PLIB_DMA_ChannelXStartAddressOffsetSet Function to set the offset addresses of buffers A and B. Buffers A and B are used in Ping-Pong mode. To learn more, refer to Operating/Addressing Mode Management.

Peripheral Address

When DMA is used for a peripheral to memory or a memory to peripheral transfer, the peripheral address within a register must be set. This register needs to be programmed with the peripheral location where data transfers takes place. For example this can be the I2C buffer Special Function Register (SFR) location or the UART Receive register, etc. To set the peripheral address use the PLIB_DMA_ChannelXPeripheralAddressSet Function.

DMA Source/Destination/Peripheral Address Control

The first step in initiating a DMA transfer is to set the source and destination addresses.

The following example sets the source address and the destination address for Channel 3.

Example:

```
// Set the source starting address to 0x1000
PLIB_DMA_ChannelXSourceStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x1000 );
// Set the destination starting address to 0x2000
PLIB_DMA_ChannelXDestinationStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x2000 );
```

On certain devices, the source starting address needs to be given as an offset from the start of the DMA RAM address (in such devices, only memory to peripheral transfers and vice-versa are possible). In such devices, the source start address is set using the PLIB_DMA_ChannelXStartAddressOffsetSet function, as shown in the following code.

Example:

```
// Set the offset-A register to 0x1000
PLIB_DMA_ChannelXStartAddressOffsetSet ( DMA_ID_0, DMA_CHANNEL_3, 0x1000, DMA_ADDRESS_OFFSET_PRIMARY);
// If ping-pong mode is enabled we will make use of buffer-B, and in that case
// we need to set an address for that also
// Set the offset-B to 0x2000
PLIB_DMA_ChannelXStartAddressOffsetSet ( DMA_ID_0, DMA_CHANNEL_3, 0x2000, DMA_ADDRESS_OFFSET_SECONDARY);
```

The following example sets the peripheral address when the DMA needs to be initiated between memory and peripheral.

DMA Peripheral Address Set Example:

```
// Set the peripheral address to ADC1 buffer for channel-3 transfers.
PLIB_DMA_ChannelXPeripheralAddressSet ( DMA_ID_0, DMA_CHANNEL_3, (volatile uint16_t)(&ADC1BUF0));
```

There are also equivalent functions available within this library that read the source/destination/peripheral address set.

Source/Destination/Peripheral Data Management

This topic describes the data configuration.

Source/Destination Transfer Size Configuration

In supported devices, each channel can be programmed to read a particular sized data from a source, and write in a particular size to a destination. Both the source and destination sizes are configurable. Use the PLIB_DMA_ChannelXSourceSizeSet Function and the PLIB_DMA_ChannelXDestinationSizeSet Function for read size and write size configuration, respectively. For example, the source size can be 200 bytes and the destination size can be 100 bytes. However, it is logical to have both set to the same value. In some devices, the amount of data that needs to be transferred is independent of the source and destination, and is called transfer count. In such devices use the PLIB_DMA_ChannelXTransferCountSet Function to set the total amount of data (block transfer data).

Cell Size Configuration

In supported devices, when DMA transfer is initiated, the amount of data configured for a cell transfer will be delivered to the destination per trigger. This can be configured using the PLIB_DMA_ChannelXCellSizeSet Function.

Channel Data Size

The DMA controller can handle both byte and word (16-bit) transactions. Each DMA channel is individually configurable for the data size using the PLIB_DMA_ChannelXDataSizeSelect Function.

Source/Destination Pointer Read

In supported devices, the application can read the byte in source/destination to which the DMA core is currently pointing. This can be done using the PLIB_DMA_ChannelXSourcePointerGet Function and the PLIB_DMA_ChannelXDestinationPointerGet Function for source and destination, respectively.

DMA Source/Destination/Peripheral Data Control

After setting the source/destination address the amount of data that needs to be transferred is set in a DMA operation.

However, prior to that the data transaction type (byte/word) needs to be configured on certain devices.

Example:
```
// Set channel-3 transactions as byte transactions
PLIB_DMA_ChannelXDataSizeSelect ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_DATA_8 );
// Set channel-3 transactions as 16-bit transactions
PLIB_DMA_ChannelXDataSizeSelect ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_DATA_16 );
// Get the current data size configured
if( DMA_CHANNEL_BYTE == PLIB_DMA_ChannelXDataSizeGet ( DMA_ID_0, DMA_CHANNEL_3 ))
{
// application code
}
```

Example:
```
// Set the transfer count to 100.
PLIB_DMA_ChannelXTransferCountSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
```

Some high-end microcontrollers (especially PIC32 devices) use the method of cell transfer and block transfer. A cell transfer is the amount of data a DMA channel transfers per an event. In such microcontrollers the application can configure the sizes of the source and destination (per channel basis). A block transfer is defined as the number of bytes transferred when a channel is enabled. A block transfer is comprised of several cell transfers. For example, if the source and destination size is 100 and the cell size is 10, it will take 10 cell transfers to complete the block transfer, which is 100 bytes (larger of source or destination sizes).

DMA cell transfer configuration Example:
```
// Set the source size to be 100
PLIB_DMA_ChannelXSourceSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
// Set the destination size to be 100
PLIB_DMA_ChannelXDestinationSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
// Set the cell size to be 10
PLIB_DMA_ChannelXCellSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 10 );
```

Special Function Modules (CRC)

Selected high-end devices support a special feature called the CRC generator.

Depending on the device, the is a highly configurable, 16-bit or 32-bit CRC generator. The CRC can be assigned to any available DMA channel using PLIB_DMA_CRCChannelSelect Function. The CRC is enabled using the PLIB_DMA_CRCEnable Function.

Optionally, the data from the source can be subjected to byte reordering using the PLIB_DMA_CRCByteOrderSelect Function. The data is then optionally passed to the Linear Shift Feedback Register (LFSR) CRC or the IP header checksum blocks using the PLIB_DMA_CRCTypeSet Function.

The CRC feature is attached to a channel in one of two possible modes, Background or Append.

Background Mode

The CRC is calculated in the background, with normal DMA behavior maintained. This can be selected by disabling the Append mode.

Append Mode

Data read from the source is not written to the destination, but the CRC data is accumulated in the CRC data register. The accumulated CRC is written to the location specified by the destination address when a block transfer completes. This mode is enabled by the PLIB_DMA_CRCAppendModeEnable Function.

The following code shows DMA CRC calculation in Append mode.

Example:
```
// CRC of the Flash block
uint32_t blockCRC;
bool error = false;
// disable the channel 0 interrupts using the Interrupts Peripheral Library
// Enable the DMA controller
PLIB_DMA_Enable ( DMA_ID_0 );
// disable the selected channel-3
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_3 );
// set the channel priority to 2
PLIB_DMA_ChannelXPrioritySelect  ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_PRIORITY_2 );
// Configure the channel auto enable feature
PLIB_DMA_ChannelXAutoEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// Disable the channel chain feature
PLIB_DMA_ChannelXChainDisable ( DMA_ID_0, DMA_CHANNEL_3 );
// Disable the start IRQ.
PLIB_DMA_ChannelXTriggerDisable ( DMA_ID_0, DMA_CHANNEL_3, DMA_CHANNEL_TRIGGER_TRANSFER_START );
// Seed the CRC generator
PLIB_DMA_CRCDataWrite ( DMA_ID_0, 0xFFFF );
// use the standard CCITT CRC 16 polynomial : X^16+X^12+X^5+1
PLIB_DMA_CRCXOREnableSet ( DMA_ID_0, 0x1021 );
```

```
// set polynomial length to 16
PLIB_DMA_CRCPolynomialLengthSet (DMA_ID_0, 16);
// Enable append mode
PLIB_DMA_CRCAppendModeEnable ( DMA_ID_0 );
// Attach CRC to DMA channel-3
PLIB_DMA_CRCChannelSelect (DMA_ID_0, DMA_CHANNEL_3);
// Enable the CRC
PLIB_DMA_CRCEnable ( DMA_ID_0 );
// Set the source starting address to 0x1000
PLIB_DMA_ChannelXSourceStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x1000 );
// Set the destination starting address to 0x2000
PLIB_DMA_ChannelXDestinationStartAddressSet ( DMA_ID_0, DMA_CHANNEL_3, 0x2000 );
// set source and destination size to 100
PLIB_DMA_ChannelXSourceSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
PLIB_DMA_ChannelXDestinationSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
// Set cell size to 100
PLIB_DMA_ChannelXCellSizeSet ( DMA_ID_0, DMA_CHANNEL_3, 100 );
//Enable channel-3
PLIB_DMA_ChannelXEnable ( DMA_ID_0, DMA_CHANNEL_3 );
// Force initiate a transfer
PLIB_DMA_StartTransferSet ( DMA_ID_0, DMA_CHANNEL_3 );
```

Status (including Channel)

This module gives application access to various DMA and channel status information.

Configuring the Library

The library is configured for the supported DMA module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_DMA_BusyActiveReset	Resets the BUSY bit of the DMA controller.
	PLIB_DMA_BusyActiveSet	Sets the BUSY bit of the DMA controller.
	PLIB_DMA_Disable	DMA module is disabled.
	PLIB_DMA_Enable	DMA module is enabled.
	PLIB_DMA_StopInIdleDisable	DMA transfers continue during Idle mode.
	PLIB_DMA_StopInIdleEnable	DMA transfers are halted during Idle mode.
	PLIB_DMA_SuspendDisable	DMA suspend is disabled and the DMA module operates normally.
	PLIB_DMA_SuspendEnable	DMA transfers are suspended to allow uninterrupted access by the CPU to the data bus.

b) Transfer/Abort (Synchronous) Functions

	Name	Description
	PLIB_DMA_AbortTransferSet	Aborts transfer on the specified channel.
	PLIB_DMA_StartTransferSet	Initiates transfer on the specified channel.

c) Transfer/Abort (Asynchronous) Trigger Management Functions

	Name	Description
	PLIB_DMA_ChannelXAbortIRQSet	Sets the IRQ to abort the DMA transfer on the specified channel.
	PLIB_DMA_ChannelXStartIRQSet	Sets the IRQ to initiate the DMA transfer on the specified channel.
	PLIB_DMA_ChannelXTriggerDisable	Disables the DMA transfer abort via a matching interrupt (specified by the IRQ).
	PLIB_DMA_ChannelXTriggerEnable	Enables the specified DMA channel trigger.
	PLIB_DMA_ChannelXTriggerIsEnabled	Returns the enable status of the specified DMA transfer/abort trigger.
	PLIB_DMA_ChannelXTriggerSourceNumberGet	Gets the source number for the DMA channel interrupts.

d) Interrupt Control and Management Functions

	Name	Description
	PLIB_DMA_ChannelXINTSourceDisable	Disables the specified interrupt source for the specified channel.

	PLIB_DMA_ChannelXINTSourceEnable	Enables the specified interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagClear	Clears the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagGet	Returns the status of the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagSet	Sets the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceIsEnabled	Returns the enable status of the specified interrupt source for the specified channel.

e) Operating/Addressing Mode Configuration Functions

	Name	Description
	PLIB_DMA_ChannelXAddressModeGet	Gets the channel address mode.
	PLIB_DMA_ChannelXAddressModeSelect	Sets the channel address mode.
	PLIB_DMA_ChannelXDestinationAddressModeGet	Gets the source address mode of the specified channel.
	PLIB_DMA_ChannelXDestinationAddressModeSelect	Sets the source address mode of the specified channel.
	PLIB_DMA_ChannelXNullWriteModeDisable	Disables the Null Write mode.
	PLIB_DMA_ChannelXNullWriteModeEnable	Enables the Null Write mode.
	PLIB_DMA_ChannelXOperatingTransferModeGet	Returns the current transfer/operating mode for the specified channel.
	PLIB_DMA_ChannelXOperatingTransferModeSelect	Selects the transfer/operating mode for the specified channel.
	PLIB_DMA_ChannelXReloadDisable	Disables reloading of the address and count registers.
	PLIB_DMA_ChannelXReloadEnable	Enables reloading of the address and count registers.
	PLIB_DMA_ChannelXSourceAddressModeGet	Gets the source address mode of the specified channel.
	PLIB_DMA_ChannelXSourceAddressModeSelect	Sets the source address mode of the specified channel.
	PLIB_DMA_ChannelXReloadIsEnabled	Returns the address and count registers reload enable status.

f) Source/Destination/Peripheral Address Control Interface Functions

	Name	Description
	PLIB_DMA_ChannelXDestinationStartAddressGet	Reads the destination start address configured for the specified channel.
	PLIB_DMA_ChannelXDestinationStartAddressSet	Writes the specified destination start address into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXPeripheralAddressGet	Gets the peripheral address configured for the specified channel.
	PLIB_DMA_ChannelXPeripheralAddressSet	Sets the peripheral address for the specified channel.
	PLIB_DMA_ChannelXSourceStartAddressGet	Reads the source start address configured for the specified channel.
	PLIB_DMA_ChannelXSourceStartAddressSet	Writes the specified source start address into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXStartAddressOffsetGet	Gets the primary/secondary start address (DPSRAM) offset.
	PLIB_DMA_ChannelXStartAddressOffsetSet	Sets the primary/secondary start address (DPSRAM) offset to the value specified depending on the offset type specified for the specified channel.

g) Source/Destination/Peripheral Data Management Functions

	Name	Description
	PLIB_DMA_ChannelXCellProgressPointerGet	Returns the number of bytes transferred since the last event.
	PLIB_DMA_ChannelXCellSizeGet	Reads the cell size (in bytes) configured for the specified channel.
	PLIB_DMA_ChannelXCellSizeSet	Writes the specified cell size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXDataSizeGet	Returns the current data size for the specified channel.
	PLIB_DMA_ChannelXDataSizeSelect	Selects the data size for the specified channel.
	PLIB_DMA_ChannelXDestinationPointerGet	Reads the current byte of the destination being pointed to for the specified channel.
	PLIB_DMA_ChannelXDestinationSizeGet	Reads the destination size configured for the specified channel.
	PLIB_DMA_ChannelXDestinationSizeSet	Writes the specified destination size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXPatternDataGet	Returns the pattern matching (for DMA abort) data programmed for the specified channel.
	PLIB_DMA_ChannelXPatternDataSet	Writes the specified pattern matching data (for DMA abort) into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXSourcePointerGet	Reads the current byte of the source being pointed to for the specified channel.
	PLIB_DMA_ChannelXSourceSizeGet	Reads the source size configured for the specified channel.
	PLIB_DMA_ChannelXSourceSizeSet	Writes the specified source size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXTransferCountGet	Gets the DMA data transfer count that is programmed for the specified channel.

	PLIB_DMA_ChannelXTransferCountSet	Sets the DMA data transfer count for the specified channel.

h) Channel Configuration Functions

	Name	Description
	PLIB_DMA_ChannelPrioritySelect	Sets the priority scheme of the DMA channels.
	PLIB_DMA_ChannelPriorityGet	Gets the priority scheme of the DMA channels.
	PLIB_DMA_ChannelXAutoDisable	Channel is disabled after a block transfer is complete.
	PLIB_DMA_ChannelXAutoEnable	Channel is continuously enabled.
	PLIB_DMA_ChannelXBusyActiveSet	Sets the Busy bit to active.
	PLIB_DMA_ChannelXBusyInActiveSet	Sets the Busy bit to inactive.
	PLIB_DMA_ChannelXChainDisable	Disables the channel chaining for the specified DMA channel.
	PLIB_DMA_ChannelXChainEnable	Channel chain feature is enabled.
	PLIB_DMA_ChannelXChainToHigher	Chains the specified channel to a channel higher in natural priority.
	PLIB_DMA_ChannelXChainToLower	Chains the specified channel to a channel lower in natural priority.
	PLIB_DMA_ChannelXDisable	Disable the specified channel.
	PLIB_DMA_ChannelXEnable	Enable the specified channel.
	PLIB_DMA_ChannelXPriorityGet	Gets the priority of the specified channel.
	PLIB_DMA_ChannelXPrioritySelect	Sets the priority of the specified channel.
	PLIB_DMA_ChannelXTransferDirectionGet	Returns the data transfer direction of the specified channel.
	PLIB_DMA_ChannelXTransferDirectionSelect	Selects the data transfer direction of the specified channel.
	PLIB_DMA_ChannelXDisabledDisablesEvents	Channel start/abort events will be ignored even if the channel is disabled.
	PLIB_DMA_ChannelXDisabledEnablesEvents	Channel start/abort events will be registered even if the channel is disabled.
	PLIB_DMA_ChannelXPatternIgnoreByteDisable	Disables the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreByteEnable	Enables the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled	Returns the state of the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreGet	Returns the pattern match ignore value.
	PLIB_DMA_ChannelXPatternIgnoreSet	Sets the pattern match ignore value.
	PLIB_DMA_ChannelXPatternLengthGet	Returns the pattern match length.
	PLIB_DMA_ChannelXPatternLengthSet	Sets the pattern match length.

i) CRC Module Interface Functions

	Name	Description
	PLIB_DMA_CRCAppendModeDisable	Disables the CRC append mode.
	PLIB_DMA_CRCAppendModeEnable	Enables the CRC append mode.
	PLIB_DMA_CRCAppendModeIsEnabled	Gets the enable status of the CRC append mode.
	PLIB_DMA_CRCBitOrderSelect	Selects the bit order for checksum calculation.
	PLIB_DMA_CRCByteOrderGet	Gets the current byte order selected by the DMA module CRC feature.
	PLIB_DMA_CRCByteOrderSelect	Selects the byte order.
	PLIB_DMA_CRCChannelGet	Returns the current DMA channel to which the CRC is assigned.
	PLIB_DMA_CRCChannelSelect	Assigns the CRC to the specified DMA channel.
	PLIB_DMA_CRCDataRead	Reads the contents of the DMA CRC data register.
	PLIB_DMA_CRCDataWrite	Writes the contents of the DMA CRC data register with the specified data.
	PLIB_DMA_CRCDisable	Disables the DMA module CRC feature.
	PLIB_DMA_CRCEnable	Enables the DMA module CRC feature.
	PLIB_DMA_CRCPolynomialLengthGet	Gets the current polynomial length.
	PLIB_DMA_CRCPolynomialLengthSet	Selects the polynomial length.
	PLIB_DMA_CRCTypeGet	Gets the current DMA module CRC feature type.
	PLIB_DMA_CRCTypeSet	Selects the DMA module CRC feature type.
	PLIB_DMA_CRCWriteByteOrderAlter	The source data is written to the destination reordered as defined by the BYTO<1:0> bits.
	PLIB_DMA_CRCWriteByteOrderMaintain	The source data is written to the destination unaltered.
	PLIB_DMA_CRCXOREnableGet	Reads the CRC XOR register.
	PLIB_DMA_CRCXOREnableSet	Writes to the CRC XOR enable register as per the specified enable mask.

j) Global Status Functions

	Name	Description
	PLIB_DMA_CRCIsEnabled	Gets the enable status of the CRC feature.
	PLIB_DMA_IsBusy	Gets the BUSY bit of the DMA controller.
	PLIB_DMA_IsEnabled	Returns the DMA module enable status.
	PLIB_DMA_LastBusAccessIsRead	Returns true if the last DMA bus access was a read.
	PLIB_DMA_LastBusAccessIsWrite	Returns true if the last DMA bus access was a write.
	PLIB_DMA_RecentAddressAccessed	Returns the address of the most recent DMA access.
	PLIB_DMA_SuspendIsEnabled	Returns the DMA suspend status.

k) Channel Status Functions

	Name	Description
	PLIB_DMA_ChannelXAutoIsEnabled	Returns the channel automatic enable status.
	PLIB_DMA_ChannelXBufferedDataIsWritten	Returns the buffered data write status for the specified channel.
	PLIB_DMA_ChannelXBusyIsBusy	Returns the busy status of the specified channel.
	PLIB_DMA_ChannelXChainIsEnabled	Returns the chain status of the specified channel.
	PLIB_DMA_ChannelXCollisionStatus	Returns the status of the specified collision type for the specified channel.
	PLIB_DMA_ChannelXEventIsDetected	Returns the event status on the specified channel.
	PLIB_DMA_ChannelXIsEnabled	Return the enable status of the specified channel.
	PLIB_DMA_ChannelXNullWriteModeIsEnabled	Returns the enable status of the Null Write mode for the specified channel.
	PLIB_DMA_ChannelXPingPongModeGet	Returns the Ping-Pong mode status for the specified channel.
	PLIB_DMA_ChannelBitsGet	Returns the DMA channel bits.

l) Feature Existence Functions

	Name	Description
	PLIB_DMA_ExistsAbortTransfer	Identifies whether the AbortTransfer feature exists on the DMA module.
	PLIB_DMA_ExistsBusy	Identifies whether the Busy feature exists on the DMA module.
	PLIB_DMA_ExistsChannelBits	Identifies whether the ChannelBits feature exists on the DMA module.
	PLIB_DMA_ExistsChannelX	Identifies whether the ChannelX feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXAbortIRQ	Identifies whether the ChannelXAbortIRQ feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXAuto	Identifies whether the ChannelXAuto feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXBusy	Identifies whether the ChannelXBusy feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXCellProgressPointer	Identifies whether the ChannelXCellProgressPointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXCellSize	Identifies whether the ChannelXCellSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXChain	Identifies whether the ChannelXChain feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXChainEnbl	Identifies whether the ChannelXChainEnbl feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationPointer	Identifies whether the ChannelXDestinationPointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationSize	Identifies whether the ChannelXDestinationSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationStartAddress	Identifies whether the ChannelXDestinationStartAddress feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDisabled	Identifies whether the ChannelXDisabled feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXEvent	Identifies whether the ChannelXEvent feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXINTSource	Identifies whether the ChannelXINTSource feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXINTSourceFlag	Identifies whether the ChannelXINTSourceFlag feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternData	Identifies whether the ChannelXPatternData feature exists on the DMA module
	PLIB_DMA_ExistsChannelXPatternIgnore	Identifies whether the ChannelXPatternIgnore feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternIgnoreByte	Identifies whether the ChannelXPatternIgnoreByte feature exists on the DMA module.

	PLIB_DMA_ExistsChannelXPatternLength	Identifies whether the ChannelXPatternLength feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPriority	Identifies whether the ChannelXPriority feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourcePointer	Identifies whether the ChannelXSourcePointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourceSize	Identifies whether the ChannelXSourceSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourceStartAddress	Identifies whether the ChannelXSourceStartAddress feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXStartIRQ	Identifies whether the ChannelXStartIRQ feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXTrigger	Identifies whether the ChannelXTrigger feature exists on the DMA module.
	PLIB_DMA_ExistsCRC	Identifies whether the CRC feature exists on the DMA module.
	PLIB_DMA_ExistsCRCAppendMode	Identifies whether the CRCAppendMode feature exists on the DMA module.
	PLIB_DMA_ExistsCRCBitOrder	Identifies whether the CRCBitOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCByteOrder	Identifies whether the CRCByteOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCChannel	Identifies whether the CRCChannel feature exists on the DMA module.
	PLIB_DMA_ExistsCRCData	Identifies whether the CRCData feature exists on the DMA module.
	PLIB_DMA_ExistsCRCPolynomialLength	Identifies whether the CRCPolynomialLength feature exists on the DMA module.
	PLIB_DMA_ExistsCRCType	Identifies whether the CRCType feature exists on the DMA module.
	PLIB_DMA_ExistsCRCWriteByteOrder	Identifies whether the CRCWriteByteOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCXOREnable	Identifies whether the CRCXOREnable feature exists on the DMA module.
	PLIB_DMA_ExistsEnableControl	Identifies whether the EnableControl feature exists on the DMA module.
	PLIB_DMA_ExistsLastBusAccess	Identifies whether the LastBusAccess feature exists on the DMA module.
	PLIB_DMA_ExistsRecentAddress	Identifies whether the RecentAddress feature exists on the DMA module.
	PLIB_DMA_ExistsStartTransfer	Identifies whether the StartTransfer feature exists on the DMA module.
	PLIB_DMA_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the DMA module.
	PLIB_DMA_ExistsSuspend	Identifies whether the Suspend feature exists on the DMA module.

m) Data Types and Constants

	Name	Description
	DMA_ADDRESS_OFFSET_TYPE	Lists the possible DMA address offsets.
	DMA_CHANNEL	Lists the possible DMA channels available.
	DMA_CHANNEL_ADDRESSING_MODE	Lists the possible channel addressing modes.
	DMA_CHANNEL_COLLISION	Lists the possible channel collision types.
	DMA_CHANNEL_DATA_SIZE	Lists the possible data sizes for the DMA channel.
	DMA_CHANNEL_PRIORITY	Lists the possible channel priorities.
	DMA_CHANNEL_TRANSFER_DIRECTION	Lists the possible data transfer directions.
	DMA_CHANNEL_TRIGGER_TYPE	Lists the possible DMA channel triggers.
	DMA_CRC_BIT_ORDER	Lists the possible CRC calculation bit orders
	DMA_CRC_BYTE_ORDER	Lists the possible byte orders.
	DMA_CRC_TYPE	Lists the possible checksums.
	DMA_DESTINATION_ADDRESSING_MODE	Lists the possible destination addressing modes.
	DMA_INT_TYPE	Lists the possible Interrupt types for a particular channel-X
	DMA_MODULE_ID	Possible instances of the DMA module.
	DMA_PATTERN_LENGTH	Gives the DMA pattern length
	DMA_PING_PONG_MODE	Lists the possible ping-pong modes.
	DMA_SOURCE_ADDRESSING_MODE	Lists the possible source addressing modes.
	DMA_TRANSFER_MODE	Lists the possible DMA transfer/operating modes.
	DMA_TRIGGER_SOURCE	Lists the possible DMA trigger sources.
	DMA_CHANNEL_INT_SOURCE	Lists the Interrupt Source number for the available DMA channels.

Description

This section describes the Application Programming Interface (API) functions of the DMA Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_DMA_BusyActiveReset Function

Resets the BUSY bit of the DMA controller.

File

plib_dma.h

C

```
void PLIB_DMA_BusyActiveReset(DMA_MODULE_ID index);
```

Returns

None.

Description

This function resets the BUSY bit of the DMA controller. The DMA module is disabled and is not actively transferring data.

Remarks

This function implements an operation of the Busy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsBusy function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_BusyActiveReset( DMA_ID_0 );
```

Function

void PLIB_DMA_BusyActiveReset (DMA_MODULE_ID index)

PLIB_DMA_BusyActiveSet Function

Sets the BUSY bit of the DMA controller.

File

plib_dma.h

C

```
void PLIB_DMA_BusyActiveSet(DMA_MODULE_ID index);
```

Returns

None.

Description

This function sets the BUSY bit of the DMA controller. The DMA module is active.

Remarks

This function implements an operation of the Busy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsBusy function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_BusyActiveSet( DMA_ID_0 );
```

Function

void PLIB_DMA_BusyActiveSet (DMA_MODULE_ID index)

PLIB_DMA_Disable Function

DMA module is disabled.

File

plib_dma.h

C

```
void PLIB_DMA_Disable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function disables the DMA module.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsEnableControl function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_Disable( DMA_ID_0 );
```

Function

void PLIB_DMA_Disable (DMA_MODULE_ID index)

PLIB_DMA_Enable Function

DMA module is enabled.

File

plib_dma.h

C

```
void PLIB_DMA_Enable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function enables the DMA module.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsEnableControl function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_Enable( DMA_ID_0 );
```

Function

void PLIB_DMA_Enable (DMA_MODULE_ID index)

PLIB_DMA_StopInIdleDisable Function

DMA transfers continue during Idle mode.

File

plib_dma.h

C

```
void PLIB_DMA_StopInIdleDisable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function causes DMA transfers to continue during Idle mode.

Remarks

This function implements an operation of the StopInIdle feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsStopInIdle function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_StopInIdleDisable( DMA_ID_0 );
```

Function

void PLIB_DMA_StopInIdleDisable (DMA_MODULE_ID index)

PLIB_DMA_StopInIdleEnable Function

DMA transfers are halted during Idle mode.

File

plib_dma.h

C

```
void PLIB_DMA_StopInIdleEnable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function halts DMA transfers during Idle mode.

Remarks

This function implements an operation of the StopInIdle feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsStopInIdle function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_StopInIdleEnable( DMA_ID_0 );
```

Function

void PLIB_DMA_StopInIdleEnable (DMA_MODULE_ID index)

PLIB_DMA_SuspendDisable Function

DMA suspend is disabled and the DMA module operates normally.

File

plib_dma.h

C

```
void PLIB_DMA_SuspendDisable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function disables the DMA suspend. The DMA module continues to operate normally.

Remarks

This function implements an operation of the Suspend feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsSuspend function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_SuspendDisable( DMA_ID_0 );
```

Function

void PLIB_DMA_SuspendDisable (DMA_MODULE_ID index)

PLIB_DMA_SuspendEnable Function

DMA transfers are suspended to allow uninterrupted access by the CPU to the data bus.

File

plib_dma.h

C

```
void PLIB_DMA_SuspendEnable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function suspends the DMA transfers to allow uninterrupted access by the CPU to the data bus.

Remarks

This function implements an operation of the Suspend feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsSuspend function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_SuspendEnable( DMA_ID_0 );
```

Function

void PLIB_DMA_SuspendEnable (DMA_MODULE_ID index)

b) Transfer/Abort (Synchronous) Functions

PLIB_DMA_AbortTransferSet Function

Aborts transfer on the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_AbortTransferSet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function aborts transfer on the specified channel. This is a forced abort controlled via software.

Remarks

This function implements an operation of the AbortTransfer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsAbortTransfer function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_AbortTransferSet (    DMA_ID_0,
                               DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_AbortTransferSet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_StartTransferSet Function

Initiates transfer on the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_StartTransferSet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function initiates transfer on the specified channel. This is a forced transfer controlled via software.

Remarks

This function implements an operation of the StartTransfer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsStartTransfer function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_StartTransferSet (    DMA_ID_0,
                               DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_StartTransferSet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

c) Transfer/Abort (Asynchronous) Trigger Management Functions

PLIB_DMA_ChannelXAbortIRQSet Function

Sets the IRQ to abort the DMA transfer on the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXAbortIRQSet(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_TRIGGER_SOURCE IRQ);
```

Returns

None.

Description

This function sets the IRQ to abort the DMA transfer on the specified channel. (IRQ to start the channel transfer.)

Remarks

This function implements an operation of the ChannelXAbortIRQ feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXAbortIRQ function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_TRIGGER_SOURCE irq = DMA_TRIGGER_TIMER_CORE;
PLIB_DMA_ChannelXAbortIRQSet (  DMA_ID_0,
                                DMA_CHANNEL_4,
                                irq );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
IRQnum	The IRQ number of the trigger source of type DMA_TRIGGER_SOURCE

Function

void PLIB_DMA_ChannelXAbortIRQSet (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_TRIGGER_SOURCE IRQ)

PLIB_DMA_ChannelXStartIRQSet Function

Sets the IRQ to initiate the DMA transfer on the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXStartIRQSet(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_TRIGGER_SOURCE IRQnum);
```

Returns

None.

Description

This function sets the IRQ to initiate the DMA transfer on the specified channel. (IRQ to start the channel transfer.)

Remarks

This function implements an operation of the ChannelXStartIRQ feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXStartIRQ function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_TRIGGER_SOURCE irq = DMA_TRIGGER_OUTPUT_COMPARE_1;
PLIB_DMA_ChannelXStartIRQSet (  DMA_ID_0,
                                DMA_CHANNEL_4,
                                irq );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
IRQnum	The IRQ number of the trigger source of type DMA_TRIGGER_SOURCE

Function

void PLIB_DMA_ChannelXStartIRQSet (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_TRIGGER_SOURCE IRQnum)

PLIB_DMA_ChannelXTriggerDisable Function

Disables the DMA transfer abort via a matching interrupt (specified by the IRQ).

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXTriggerDisable(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_CHANNEL_TRIGGER_TYPE
trigger);
```

Returns

None.

Description

This function disables the DMA transfer abort via a matching interrupt (specified by the IRQ). The interrupt number IRQ is ignored and does not terminate a transfer.

Remarks

This function implements an operation of the ChannelXTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXTrigger function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXTriggerDisable (   DMA_ID_0,
                                    DMA_CHANNEL_4,
                                    DMA_CHANNEL_TRIGGER_PATTERN_MATCH_ABORT );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
trigger	Type of trigger (transfer start/abort/pattern match abort)

Function

void PLIB_DMA_ChannelXTriggerDisable (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_TRIGGER_TYPE trigger)

PLIB_DMA_ChannelXTriggerEnable Function

Enables the specified DMA channel trigger.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXTriggerEnable(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_CHANNEL_TRIGGER_TYPE
trigger);
```

Returns

None.

Description

This function enables the specified DMA channel trigger.

Remarks

This function implements an operation of the ChannelXTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXTrigger function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXTriggerEnable (    DMA_ID_0,
                                    DMA_CHANNEL_4,
                                    DMA_CHANNEL_TRIGGER_TRANSFER_ABORT );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
trigger	Type of trigger (transfer start/abort/pattern match abort)

Function

void PLIB_DMA_ChannelXTriggerEnable (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_TRIGGER_TYPE trigger)

PLIB_DMA_ChannelXTriggerIsEnabled Function

Returns the enable status of the specified DMA transfer/abort trigger.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXTriggerIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_CHANNEL_TRIGGER_TYPE
trigger);
```

Returns

- true - The specified trigger is enabled
- false - The specified trigger is disabled

Description

This function returns the enable status of the specified DMA transfer/abort trigger.

Remarks

This function implements an operation of the ChannelXTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXTrigger function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool startTriggerstatus;
startTriggerstatus = PLIB_DMA_ChannelXTriggerIsEnabled (
                                        DMA_ID_0,
                                        DMA_CHANNEL_4,
                                        DMA_CHANNEL_TRIGGER_TRANSFER_START );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
trigger	Type of trigger (transfer start/abort/pattern match abort)

Function

bool PLIB_DMA_ChannelXTriggerIsEnabled(DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_TRIGGER_TYPE trigger)

PLIB_DMA_ChannelXTriggerSourceNumberGet Function

Gets the source number for the DMA channel interrupts.

File

plib_dma.h

C

```
DMA_CHANNEL_INT_SOURCE PLIB_DMA_ChannelXTriggerSourceNumberGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function returns the interrupt source number for the specified DMA channel.

Remarks

This function implements an operation of the ChannelXTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXTrigger function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXTriggerSourceNumberGet (   DMA_ID_0,
                                            DMA_CHANNEL_4);
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXTriggerSourceNumberGet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

d) Interrupt Control and Management Functions

PLIB_DMA_ChannelXINTSourceDisable Function

Disables the specified interrupt source for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXINTSourceDisable(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_INT_TYPE
dmaINTSource);
```

Returns

None.

Description

This function disables the specified interrupt source for the specified channel.

Remarks

This function implements an operation of the ChannelXINTSource feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXINTSource function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXINTSourceDisable ( DMA_ID_0,
                                    spiDMAChannel,
                                    DMA_INT_ADDRESS_ERROR );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Function

void PLIB_DMA_ChannelXINTSourceDisable (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
DMA_INT_TYPE dmaINTSource)

PLIB_DMA_ChannelXINTSourceEnable Function

Enables the specified interrupt source for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXINTSourceEnable(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_INT_TYPE
dmaINTSource);
```

Returns

None.

Description

This function enables the specified interrupt source for the specified channel.

Remarks

This function implements an operation of the ChannelXINTSource feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXINTSource function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXINTSourceEnable (  DMA_ID_0,
                                    spiDMAChannel,
                                    DMA_INT_ADDRESS_ERROR );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Function

void PLIB_DMA_ChannelXINTSourceEnable (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
DMA_INT_TYPE dmaINTSource)

PLIB_DMA_ChannelXINTSourceFlagClear Function

Clears the interrupt flag of the specified DMA interrupt source for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXINTSourceFlagClear(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_INT_TYPE
dmaINTSource);
```

Returns

None.

Description

This function clears the interrupt flag of the specified DMA interrupt source for the specified channel.

Remarks

This function implements an operation of the ChannelXINTSourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXINTSourceFlag function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXINTSourceFlagClear (   DMA_ID_0,
                                        spiDMAChannel,
                                        DMA_INT_ADDRESS_ERROR );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Function

void PLIB_DMA_ChannelXINTSourceFlagClear (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
DMA_INT_TYPE dmaINTSource)

PLIB_DMA_ChannelXINTSourceFlagGet Function

Returns the status of the interrupt flag of the specified DMA interrupt source for the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXINTSourceFlagGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_INT_TYPE
dmaINTSource);
```

Returns

- true - The interrupt flag is set
- false - The interrupt flag is not set

Description

This function returns the status of the interrupt flag of the specified DMA interrupt source for the specified channel.

Remarks

This function implements an operation of the ChannelXINTSourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXINTSourceFlag function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
bool AddressErrorINTStatus;
AddressErrorINTStatus = PLIB_DMA_ChannelXINTSourceFlagGet (
                                                DMA_ID_0,
                                                spiDMAChannel,
                                                DMA_INT_ADDRESS_ERROR );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Function

bool PLIB_DMA_ChannelXINTSourceFlagGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
DMA_INT_TYPE dmaINTSource)

PLIB_DMA_ChannelXINTSourceFlagSet Function

Sets the interrupt flag of the specified DMA interrupt source for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXINTSourceFlagSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_INT_TYPE
dmaINTSource);
```

Returns

None.

Description

This function sets the interrupt flag of the specified DMA interrupt source for the specified channel.

Remarks

This function implements an operation of the ChannelXINTSourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXINTSourceFlag function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXINTSourceFlagSet ( DMA_ID_0,
                                    spiDMAChannel,
                                    DMA_INT_ADDRESS_ERROR );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Function

void PLIB_DMA_ChannelXINTSourceFlagSet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
DMA_INT_TYPE dmaINTSource)

PLIB_DMA_ChannelXINTSourceIsEnabled Function

Returns the enable status of the specified interrupt source for the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXINTSourceIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_INT_TYPE
dmaINTSource);
```

Returns

- true - The interrupt is enabled
- false - The interrupt is not enabled

Description

This function returns the enable status of the specified interrupt source for the specified channel.

Remarks

This function implements an operation of the ChannelXINTSource feature. This feature may not be available on all devices. Please refer to the

specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXINTSource function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
bool dmaINTSourceEnableStatus;
dmaINTSourceEnableStatus = PLIB_DMA_ChannelXINTSourceIsEnabled (
                                                    DMA_ID_0,
                                                    spiDMAChannel,
                                                    DMA_INT_ADDRESS_ERROR );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
dmaINTSource	One of the DMA interrupt sources specified by DMA_INT_TYPE

Function

bool PLIB_DMA_ChannelXINTSourceIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
DMA_INT_TYPE dmaINTSource)

e) Operating/Addressing Mode Configuration Functions

PLIB_DMA_ChannelXAddressModeGet Function

Gets the channel address mode.

File

plib_dma.h

C

```
DMA_CHANNEL_ADDRESSING_MODE PLIB_DMA_ChannelXAddressModeGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- channelAddressMode - One of the possible source addressing modes listed by DMA_CHANNEL_ADDRESSING_MODE

Description

This function gets the channel address mode.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_CHANNEL_ADDRESSING_MODE channelAddressMode;
channelAddressMode = PLIB_DMA_ChannelXAddressModeGet (  DMA_ID_0,
                                                        DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_CHANNEL_ADDRESSING_MODE PLIB_DMA_ChannelXAddressModeGet (
DMA_MODULE_ID index,

DMA_CHANNEL channel)

PLIB_DMA_ChannelXAddressModeSelect Function

Sets the channel address mode.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXAddressModeSelect(DMA_MODULE_ID index, DMA_CHANNEL channel,
DMA_CHANNEL_ADDRESSING_MODE channelAddressMode);
```

Returns

None.

Description

This function sets the channel address mode.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXAddressModeSelect (
                    DMA_ID_0,
                    DMA_CHANNEL_4,
                    DMA_ADDRESSING_REGISTER_INDIRECT_WITH_POST_INCREMENT );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
channelAddressMode	One of the possible channel addressing modes listed by DMA_CHANNEL_ADDRESSING_MODE

Function

void PLIB_DMA_ChannelXAddressModeSelect (

DMA_MODULE_ID index,

DMA_CHANNEL channel,

DMA_CHANNEL_ADDRESSING_MODE channelAddressMode)

PLIB_DMA_ChannelXDestinationAddressModeGet Function

Gets the source address mode of the specified channel.

File

plib_dma.h

C

```
DMA_DESTINATION_ADDRESSING_MODE PLIB_DMA_ChannelXDestinationAddressModeGet(DMA_MODULE_ID index, DMA_CHANNEL
channel);
```

Returns

- destinationAddressMode - One of the possible source addressing modes listed by DMA_DESTINATION_ADDRESSING_MODE

Description

This function gets the source address mode of the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_SOURCE_ADDRESSING_MODE destinationAddressMode;
destinationAddressMode = PLIB_DMA_ChannelXDestinationAddressModeGet (
                                                        DMA_ID_0,
                                                        DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_DESTINATION_ADDRESSING_MODE PLIB_DMA_ChannelXDestinationAddressModeGet (
DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXDestinationAddressModeSelect Function

Sets the source address mode of the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDestinationAddressModeSelect(DMA_MODULE_ID index, DMA_CHANNEL channel,
DMA_DESTINATION_ADDRESSING_MODE destinationAddressMode);
```

Returns

None.

Description

This function Sets the source address mode of the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXDestinationAddressModeSelect (
                        DMA_ID_0,
                DMA_CHANNEL_4,
                        DMA_ADDRESSING_DESTINATION_INCREMENT_BASED_ON_SIZE );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
destinationAddressMode	One of the possible source addressing modes listed by DMA_DESTINATION_ADDRESSING_MODE

Function

void PLIB_DMA_ChannelXDestinationAddressModeSelect (
DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_DESTINATION_ADDRESSING_MODE destinationAddressMode)

PLIB_DMA_ChannelXNullWriteModeDisable Function

Disables the Null Write mode.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXNullWriteModeDisable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function disables the Null Write mode. No dummy write is initiated.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXNullWriteModeDisable ( DMA_ID_0,
                                        DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXNullWriteModeDisable(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXNullWriteModeEnable Function

Enables the Null Write mode.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXNullWriteModeEnable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function enables the Null Write mode. A dummy write is initiated to the source address for every write to the destination address.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXNullWriteModeEnable (  DMA_ID_0,
                                        DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXNullWriteModeEnable (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXOperatingTransferModeGet Function

Returns the current transfer/operating mode for the specified channel.

File

plib_dma.h

C

```
DMA_TRANSFER_MODE PLIB_DMA_ChannelXOperatingTransferModeGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- channeltransferMode - One of the possible operating/transfer modes listed by DMA_TRANSFER_MODE

Description

This function returns the current transfer/operating mode for the specified channel. (Transfer mode and operating mode are used interchangeably.)

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_TRANSFER_MODE channeltransferMode;
channeltransferMode = PLIB_DMA_ChannelXOperatingTransferModeGet (
                                                        DMA_ID_0,
                                                        DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_TRANSFER_MODE PLIB_DMA_ChannelXOperatingTransferModeGet (
DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXOperatingTransferModeSelect Function

Selects the transfer/operating mode for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXOperatingTransferModeSelect(DMA_MODULE_ID index, DMA_CHANNEL channel,
DMA_TRANSFER_MODE channeltransferMode);
```

Returns

None.

Description

This function selects the transfer/operating mode for the specified channel. (Transfer mode and operating mode are used interchangeably.)

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_TRANSFER_MODE channeltransferMode = DMA_MODE_REPEATED_CONTINUOUS;
PLIB_DMA_ChannelXOperatingTransferModeSelect (  DMA_ID_0,
                                                DMA_CHANNEL_4,
                                                channeltransferMode );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
channeltransferMode	One of the possible operating/transfer modes listed by DMA_TRANSFER_MODE

Function

void PLIB_DMA_ChannelXOperatingTransferModeSelect (
DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_TRANSFER_MODE channeltransferMode)

PLIB_DMA_ChannelXReloadDisable Function

Disables reloading of the address and count registers.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXReloadDisable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function disables reloading of the address and count registers. The source, destination address, and the DMA count registers are not reloaded to their previous values upon the start of the next operation.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXReloadDisable (    DMA_ID_0,
                                    DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXReloadDisable (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXReloadEnable Function

Enables reloading of the address and count registers.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXReloadEnable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function enables reloading of the address and count registers. The source, destination address, and the DMA count registers are reloaded to their previous values upon the start of the next operation.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXReloadEnable ( DMA_ID_0,
                                DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXReloadEnable(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXSourceAddressModeGet Function

Gets the source address mode of the specified channel.

File

plib_dma.h

C

```
DMA_SOURCE_ADDRESSING_MODE PLIB_DMA_ChannelXSourceAddressModeGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- DMA_SOURCE_ADDRESSING_MODE - One of the possible source addressing modes listed by DMA_SOURCE_ADDRESSING_MODE

Description

This function gets the source address mode of the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_SOURCE_ADDRESSING_MODE sourceAddressMode;
sourceAddressMode = PLIB_DMA_ChannelXSourceAddressModeGet ( DMA_ID_0,
                                                            DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_SOURCE_ADDRESSING_MODE PLIB_DMA_ChannelXSourceAddressModeGet (
DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXSourceAddressModeSelect Function

Sets the source address mode of the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXSourceAddressModeSelect(DMA_MODULE_ID index, DMA_CHANNEL channel,
DMA_SOURCE_ADDRESSING_MODE sourceAddressMode);
```

Returns

None.

Description

This function sets the source address mode of the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXSourceAddressModeSelect (
                            DMA_ID_0,
                            DMA_CHANNEL_4,
                            DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
sourceAddressMode	One of the possible source addressing modes listed by DMA_SOURCE_ADDRESSING_MODE

Function

void PLIB_DMA_ChannelXSourceAddressModeSelect (
DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_SOURCE_ADDRESSING_MODE sourceAddressMode)

PLIB_DMA_ChannelXReloadIsEnabled Function

Returns the address and count registers reload enable status.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXReloadIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - The address and count registers reload is enabled
- false - The address and count registers reload is disabled

Description

This function returns the address and count registers reload enable status.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
bool chAddressCountReloadStatus;
chAddressCountReloadStatus = PLIB_DMA_ChannelXReloadIsEnabled (
                                                              DMA_ID_0,
                                                              DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXReloadIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL channel)

f) Source/Destination/Peripheral Address Control Interface Functions

PLIB_DMA_ChannelXDestinationStartAddressGet Function

Reads the destination start address configured for the specified channel.

File

plib_dma.h

C

```
uint32_t PLIB_DMA_ChannelXDestinationStartAddressGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint32_t - The destination start address configured for this channel

Description

This function reads the destination start address configured for the specified channel.

Remarks

This function implements an operation of the ChannelXDestinationStartAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDestinationStartAddress function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint32_t DestinationStartAddress;
DestinationStartAddress = PLIB_DMA_ChannelXDestinationStartAddressGet (
                                                              DMA_ID_0,
                                                              spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint32_t PLIB_DMA_ChannelXDestinationStartAddressGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXDestinationStartAddressSet Function

Writes the specified destination start address into the register corresponding to the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDestinationStartAddressSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, uint32_t
destinationStartAddress);
```

Returns

None.

Description

This function writes the specified destination start address into the register corresponding to the specified channel.

Remarks

This function implements an operation of the ChannelXDestinationStartAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDestinationStartAddress function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint32_t destinationStartAddress = 0x00FDEA00;
PLIB_DMA_ChannelXDestinationStartAddressSet(    DMA_ID_0,
                                                spiDMAChannel,
                                                destinationStartAddress );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
destinationStartAddress	The destination start address

Function

void PLIB_DMA_ChannelXDestinationStartAddressSet (
DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
uint32_t destinationStartAddress)

PLIB_DMA_ChannelXPeripheralAddressGet Function

Gets the peripheral address configured for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXPeripheralAddressGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- uint16_t - The peripheral address configured for the specified channel

Description

This function gets the peripheral address configured for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint16_t peripheraladdress;
peripheraladdress = PLIB_DMA_ChannelXPeripheralAddressGet ( DMA_ID_0,
                                                            DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXPeripheralAddressGet(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXPeripheralAddressSet Function

Sets the peripheral address for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPeripheralAddressSet(DMA_MODULE_ID index, DMA_CHANNEL channel, uint16_t
peripheraladdress);
```

Returns

None.

Description

This function sets the peripheral address for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint16_t peripheraladdress = 0x100;
PLIB_DMA_ChannelXPeripheralAddressSet ( DMA_ID_0,
                                        DMA_CHANNEL_4,
                                        peripheraladdress );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
peripheraladdress	The peripheral address for the specified channel

Function

void PLIB_DMA_ChannelXPeripheralAddressSet(DMA_MODULE_ID index,

DMA_CHANNEL channel ,
uint16_t peripheraladdress)

PLIB_DMA_ChannelXSourceStartAddressGet Function

Reads the source start address configured for the specified channel.

File

plib_dma.h

C

```
uint32_t PLIB_DMA_ChannelXSourceStartAddressGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint32_t - The source start address configured for this channel

Description

This function reads the source start address configured for the specified channel.

Remarks

This function implements an operation of the ChannelXSourceStartAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXSourceStartAddress function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint32_t SourceStartAddress;
SourceStartAddress = PLIB_DMA_ChannelXSourceStartAddressGet(DMA_ID_0,
                                                            spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint32_t PLIB_DMA_ChannelXSourceStartAddressGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXSourceStartAddressSet Function

Writes the specified source start address into the register corresponding to the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXSourceStartAddressSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, uint32_t
sourceStartAddress);
```

Returns

None.

Description

This function writes the specified Source start address into the register corresponding to the specified channel.

Remarks

This function implements an operation of the ChannelXSourceStartAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXSourceStartAddress function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint32_t sourceStartAddress = 0x00FDEA00;
PLIB_DMA_ChannelXSourceStartAddressSet (    DMA_ID_0,
                                            spiDMAChannel,
                                            sourceStartAddress );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
sourceStartAddress	The source start address

Function

void PLIB_DMA_ChannelXSourceStartAddressSet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
uint32_t sourceStartAddress)

PLIB_DMA_ChannelXStartAddressOffsetGet Function

Gets the primary/secondary start address (DPSRAM) offset.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXStartAddressOffsetGet(DMA_MODULE_ID index, DMA_CHANNEL channel,
DMA_ADDRESS_OFFSET_TYPE offset);
```

Returns

- uint16_t - The primary/secondary DPSRAM start address offset

Description

This function gets the primary/secondary start address (DPSRAM) offset.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint16_t addressOffsetA;
addressOffsetA = PLIB_DMA_ChannelXStartAddressOffsetGet (
                                                DMA_ID_0,
                                                DMA_CHANNEL_4,
                                                address,
                                                DMA_ADDRESS_OFFSET_PRIMARY );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
offset	The type of the address offset (primary/secondary)

Function

uint16_t PLIB_DMA_ChannelXStartAddressOffsetGet (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_ADDRESS_OFFSET_TYPE offset)

PLIB_DMA_ChannelXStartAddressOffsetSet Function

Sets the primary/secondary start address (DPSRAM) offset to the value specified depending on the offset type specified for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXStartAddressOffsetSet(DMA_MODULE_ID index, DMA_CHANNEL channel, uint16_t address,
DMA_ADDRESS_OFFSET_TYPE offset);
```

Returns

None.

Description

This function sets the primary/secondary start address (DPSRAM) offset to the value specified depending on the offset type specified for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint16_t address = 0x100;
PLIB_DMA_ChannelXStartAddressOffsetSet (    DMA_ID_0,
                                            DMA_CHANNEL_4,
                                            address,
                                            DMA_ADDRESS_OFFSET_PRIMARY );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
address	The primary/secondary DPSRAM start address offset
offset	The type of the address offset (primary/secondary)

Function

void PLIB_DMA_ChannelXStartAddressOffsetSet(DMA_MODULE_ID index,
DMA_CHANNEL channel ,
uint16_t address,
DMA_ADDRESS_OFFSET_TYPE offset)

g) Source/Destination/Peripheral Data Management Functions

PLIB_DMA_ChannelXCellProgressPointerGet Function

Returns the number of bytes transferred since the last event.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXCellProgressPointerGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The number of bytes transferred since the last event.

The cell progress pointer (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function returns the number of bytes transferred since the last event.

Remarks

This function implements an operation of the ChannelXCellProgressPointer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXCellProgressPointer function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t CellProgress;
CellProgress = PLIB_DMA_ChannelXCellProgressPointerGet (   DMA_ID_0,
                                                           spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXCellProgressPointerGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXCellSizeGet Function

Reads the cell size (in bytes) configured for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXCellSizeGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The cell size configured (in bytes) for this channel

The cell size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the cell size (in bytes) configured for the specified channel.

Remarks

This function implements an operation of the ChannelXCellSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXCellSize function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t cellSize;
cellSize = PLIB_DMA_ChannelXCellSizeGet (   DMA_ID_0,
                                            spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXCellSizeGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXCellSizeSet Function

Writes the specified cell size into the register corresponding to the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXCellSizeSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, uint16_t CellSize);
```

Returns

None.

Description

This function writes the specified cell size into the register corresponding to the specified channel.

Remarks

This function implements an operation of the ChannelXCellSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXCellSize function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t cellSize = 0x10;
PLIB_DMA_ChannelXCellSizeSet ( DMA_ID_0, spiDMAChannel, cellSize );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
cellSize	The cell size in bytes. The cell size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Function

void PLIB_DMA_ChannelXCellSizeSet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
uint16_t cellSize)

PLIB_DMA_ChannelXDataSizeGet Function

Returns the current data size for the specified channel.

File

plib_dma.h

C

```
DMA_CHANNEL_DATA_SIZE PLIB_DMA_ChannelXDataSizeGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- channelDataSize - One of the possible data sizes listed by DMA_CHANNEL_DATA_SIZE

Description

This function returns the current data size for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_CHANNEL_DATA_SIZE channelDataSize;
channelDataSize = PLIB_DMA_ChannelXDataSizeGet( DMA_ID_0,
                                                DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_CHANNEL_DATA_SIZE PLIB_DMA_ChannelXDataSizeGet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXDataSizeSelect Function

Selects the data size for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDataSizeSelect(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_CHANNEL_DATA_SIZE
channelDataSize);
```

Returns

None.

Description

This function selects the data size for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_CHANNEL_DATA_SIZE channelDataSize = DMA_CHANNEL_DATA_8;
PLIB_DMA_ChannelXDataSizeSelect (   DMA_ID_0,
                                    DMA_CHANNEL_4,
                                    channelDataSize );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
channelDataSize	One of the possible data sizes listed by DMA_CHANNEL_DATA_SIZE

Function

void PLIB_DMA_ChannelXDataSizeSelect (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_DATA_SIZE channelDataSize)

PLIB_DMA_ChannelXDestinationPointerGet Function

Reads the current byte of the destination being pointed to for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXDestinationPointerGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The destination byte being pointed to for this channel.

The destination pointer (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the current byte of the destination being pointed to for the specified channel.

Remarks

This function implements an operation of the ChannelXDestinationPointer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDestinationPointer function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t destinationbyte;
destinationbyte = PLIB_DMA_ChannelXDestinationPointerGet ( DMA_ID_0,
                                                           spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXDestinationPointerGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXDestinationSizeGet Function

Reads the destination size configured for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXDestinationSizeGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The destination size configured (in bytes) for this channel.

The destination size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the destination size configured for the specified channel.

Remarks

This function implements an operation of the ChannelXDestinationSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDestinationSize function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t DestinationSize;
DestinationSize = PLIB_DMA_ChannelXDestinationSizeGet ( DMA_ID_0,
                                                        spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXDestinationSizeGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXDestinationSizeSet Function

Writes the specified destination size into the register corresponding to the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDestinationSizeSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, uint16_t
destinationSize);
```

Returns

None.

Description

This function writes the specified destination size into the register corresponding to the specified channel.

Remarks

This function implements an operation of the ChannelXDestinationSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDestinationSize function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t destinationSize = 0xA00;
PLIB_DMA_ChannelXDestinationSizeSet( DMA_ID_0, spiDMAChannel, destinationSize );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
destinationSize	The destination size. The destination size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Function

void PLIB_DMA_ChannelXDestinationSizeSet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
uint16_t destinationSize)

PLIB_DMA_ChannelXPatternDataGet Function

Returns the pattern matching (for DMA abort) data programmed for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXPatternDataGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The pattern matching data programmed for the current channel.

The size of pattern matching data(8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function returns pattern matching (for DMA abort) data programmed for the specified channel. (The size of pattern matching data(8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Remarks

This function implements an operation of the ChannelXPatternData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternData function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t patternData;
patternData = PLIB_DMA_ChannelXPatternDataGet ( DMA_ID_0, spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXPatternDataGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXPatternDataSet Function

Writes the specified pattern matching data (for DMA abort) into the register corresponding to the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPatternDataSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, uint16_t patternData);
```

Returns

None.

Description

This function writes the specified pattern matching data (for DMA abort) into the register corresponding to the specified channel. (The size of pattern matching data(8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Remarks

This function implements an operation of the ChannelXPatternData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternData function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
```

```
uint16_t patternData = '0';
PLIB_DMA_ChannelXPatternDataSet ( DMA_ID_0, spiDMAChannel, patternData );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
patternData	The pattern matching DATA programmed for the current channel (The size of pattern matching data(8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.)

Function

void PLIB_DMA_ChannelXPatternDataSet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
uint16_t patternData)

PLIB_DMA_ChannelXSourcePointerGet Function

Reads the current byte of the source being pointed to for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXSourcePointerGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The source byte being pointed to for this channel.

The source pointer (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the current byte of the source being pointed to for the specified channel.

Remarks

This function implements an operation of the ChannelXSourcePointer feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXSourcePointer function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t sourcebyte;
sourcebyte = PLIB_DMA_ChannelXSourcePointerGet ( DMA_ID_0, spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXSourcePointerGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXSourceSizeGet Function

Reads the source size configured for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXSourceSizeGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- uint16_t - The Source size configured (in bytes) for this channel.

The source size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the source size configured for the specified channel.

Remarks

This function implements an operation of the ChannelXSourceSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXSourceSize function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t sourceSize;
sourceSize = PLIB_DMA_ChannelXSourceSizeGet (    DMA_ID_0,
                                                 spiDMAChannel );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXSourceSizeGet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel)

PLIB_DMA_ChannelXSourceSizeSet Function

Writes the specified source size into the register corresponding to the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXSourceSizeSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, uint16_t sourceSize);
```

Returns

None.

Description

This function writes the specified source size into the register corresponding to the specified channel.

Remarks

This function implements an operation of the ChannelXSourceSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXSourceSize function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL spiDMAChannel = DMA_CHANNEL_2;
uint16_t sourceSize = 0xA00;
PLIB_DMA_ChannelXSourceSizeSet (    DMA_ID_0,
                                    spiDMAChannel,
                                    sourceSize );
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
sourceSize	The source size. The source size (8-bit, 16-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Function

void PLIB_DMA_ChannelXSourceSizeSet (DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,
uint16_t sourceSize)

PLIB_DMA_ChannelXTransferCountGet Function

Gets the DMA data transfer count that is programmed for the specified channel.

File

plib_dma.h

C

```
uint16_t PLIB_DMA_ChannelXTransferCountGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- uint16_t - The DMA transfer count for the channel

Description

This function gets the DMA data transfer count that is programmed for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint16_t transferCount;
transferCount = PLIB_DMA_ChannelXTransferCountGet ( DMA_ID_0,
                                                    DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint16_t PLIB_DMA_ChannelXTransferCountGet(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXTransferCountSet Function

Sets the DMA data transfer count for the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXTransferCountSet(DMA_MODULE_ID index, DMA_CHANNEL channel, uint16_t transferCount);
```

Returns

None.

Description

This function sets the DMA data transfer count for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint16_t transferCount = 0x10;
PLIB_DMA_ChannelXTransferCountSet ( DMA_ID_0,
                                    DMA_CHANNEL_4,
                                    transferCount );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
transferCount	The DMA transfer count for the channel

Function

void PLIB_DMA_ChannelXTransferCountSet(DMA_MODULE_ID index,
DMA_CHANNEL channel ,
uint16_t transferCount)

h) Channel Configuration Functions

PLIB_DMA_ChannelPrioritySelect Function

Sets the priority scheme of the DMA channels.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelPrioritySelect(DMA_MODULE_ID index, DMA_CHANNEL_PRIORITY channelPriority);
```

Returns

None.

Description

This function sets the priority scheme of the DMA channels at the global level. This function is used in devices that do not have the per channel priority feature.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_CHANNEL_PRIORITY channelPriority = DMA_CHANNEL_ROUND_ROBIN;
PLIB_DMA_ChannelPrioritySelect( DMA_ID_0, channelPriority );
```

Parameters

Parameters	Description
channelPriority	One of the supported priorities listed by DMA_CHANNEL_PRIORITY

Function

void PLIB_DMA_ChannelPrioritySelect (DMA_MODULE_ID index,
DMA_CHANNEL_PRIORITY channelPriority)

PLIB_DMA_ChannelPriorityGet Function

Gets the priority scheme of the DMA channels.

File

plib_dma.h

C

```
DMA_CHANNEL_PRIORITY PLIB_DMA_ChannelPriorityGet(DMA_MODULE_ID index);
```

Returns

- channelPriority - One of the supported priorities listed by DMA_CHANNEL_PRIORITY

Description

This function gets the priority scheme of the DMA channels at the global level. This function is used in devices that do not have the per channel priority feature.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_CHANNEL_PRIORITY channelPriority;
channelPriority = PLIB_DMA_ChannelPriorityGet( DMA_ID_0 );
```

Function

DMA_CHANNEL_PRIORITY PLIB_DMA_ChannelPriorityGet (DMA_MODULE_ID index)

PLIB_DMA_ChannelXAutoDisable Function

Channel is disabled after a block transfer is complete.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXAutoDisable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function disables a channel after a block transfer is complete.

Remarks

This function implements an operation of the ChannelXAuto feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXAuto function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXAutoDisable( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXAutoDisable (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXAutoEnable Function

Channel is continuously enabled.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXAutoEnable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function enables the channel continuously. The channel is not automatically disabled after a block transfer is complete.

Remarks

This function implements an operation of the ChannelXAuto feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXAuto function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXAutoEnable( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXAutoEnable (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXBusyActiveSet Function

Sets the Busy bit to active.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXBusyActiveSet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function sets the Busy bit to active, indicating the channel is active or has been enabled.

Remarks

This function implements an operation of the ChannelXBusy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXBusy function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXBusyActiveSet ( DMA_ID_0, DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXBusyActiveSet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXBusyInActiveSet Function

Sets the Busy bit to inactive.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXBusyInActiveSet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function sets the Busy bit to inactive, indicating the channel is inactive or has been disabled.

Remarks

This function implements an operation of the ChannelXBusy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXBusy function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXBusyInActiveSet ( DMA_ID_0, DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXBusyInActiveSet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXChainDisable Function

Disables the channel chaining for the specified DMA channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXChainDisable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function disables the channel chaining for the specified DMA channel.

Remarks

This function implements an operation of the ChannelXChainEnbl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXChainEnbl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXChainDisable( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXChainDisable(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXChainEnable Function

Channel chain feature is enabled.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXChainEnable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function enables the channel chain feature.

Remarks

This function implements an operation of the ChannelXChainEnbl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXChainEnbl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXChainEnable( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXChainEnable (DMA_MODULE_ID index,

DMA_CHANNEL channel)

PLIB_DMA_ChannelXChainToHigher Function

Chains the specified channel to a channel higher in natural priority.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXChainToHigher(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function will chain the specified channel to a channel higher in natural priority. CH5 will be enabled by a CH4 transfer complete.

Remarks

This function implements an operation of the ChannelXChain feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXChain function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXChainToHigher ( DMA_ID_0, DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXChainToHigher (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXChainToLower Function

Chains the specified channel to a channel lower in natural priority.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXChainToLower(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function will chain the specified channel to a channel lower in natural priority. CH3 will be enabled by a CH4 transfer complete.

Remarks

This function implements an operation of the ChannelXChain feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXChain function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXChainToLower ( DMA_ID_0, DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXChainToLower(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXDisable Function

Disable the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDisable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function will disable the specified channel.

Remarks

This function implements an operation of the ChannelX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelX function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXDisable ( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXDisable (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXEnable Function

Enable the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXEnable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function will enable the specified channel.

Remarks

This function implements an operation of the ChannelX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelX function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXEnable ( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXEnable (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXPriorityGet Function

Gets the priority of the specified channel.

File

plib_dma.h

C

```
DMA_CHANNEL_PRIORITY PLIB_DMA_ChannelXPriorityGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- channelPriority - One of the supported priorities listed by DMA_CHANNEL_PRIORITY

Description

This function gets the priority of the specified channel.

Remarks

This function implements an operation of the ChannelXPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL channel = DMA_CHANNEL_0;
DMA_CHANNEL_PRIORITY channelPriority;
channelPriority = PLIB_DMA_ChannelXPriorityGet( DMA_ID_0, channel );
```

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL

Function

DMA_CHANNEL_PRIORITY PLIB_DMA_ChannelXPriorityGet (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXPrioritySelect Function

Sets the priority of the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPrioritySelect(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_CHANNEL_PRIORITY
channelPriority);
```

Returns

None.

Description

This function sets the priority of the specified channel.

Remarks

This function implements an operation of the ChannelXPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL channel = DMA_CHANNEL_0;
DMA_CHANNEL_PRIORITY channelPriority = DMA_CHANNEL_PRIORITY_3;
PLIB_DMA_ChannelXPrioritySelect( DMA_ID_0, channel, channelPriority );
```

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL
channelPriority	One of the supported priorities listed by DMA_CHANNEL_PRIORITY

Function

void PLIB_DMA_ChannelXPrioritySelect (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_PRIORITY channelPriority)

PLIB_DMA_ChannelXTransferDirectionGet Function

Returns the data transfer direction of the specified channel.

File

plib_dma.h

C

```
DMA_CHANNEL_TRANSFER_DIRECTION PLIB_DMA_ChannelXTransferDirectionGet(DMA_MODULE_ID index, DMA_CHANNEL
channel);
```

Returns

- DMA_CHANNEL_TRANSFER_DIRECTION - The transfer direction indicated by the DMA_CHANNEL_TRANSFER_DIRECTION

Description

This function returns the data transfer direction of the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_CHANNEL_TRANSFER_DIRECTION chTransferDirection;
chTransferDirection = PLIB_DMA_ChannelXTransferDirectionGet (
                                                              DMA_ID_0,
                                                              DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_CHANNEL_TRANSFER_DIRECTION PLIB_DMA_ChannelXTransferDirectionGet (
DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXTransferDirectionSelect Function

Selects the data transfer direction of the specified channel.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXTransferDirectionSelect(DMA_MODULE_ID index, DMA_CHANNEL channel,
DMA_CHANNEL_TRANSFER_DIRECTION chTransferDirection);
```

Returns

None.

Description

This function selects the data transfer direction of the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXTransferDirectionSelect (
                              DMA_ID_0,
                              DMA_CHANNEL_4,
                              DMA_READ_FROM_MEMORY_WRITE_TO_PERIPHERAL );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL
chTransferDirection	The transfer direction indicated by DMA_CHANNEL_TRANSFER_DIRECTION

Function

void PLIB_DMA_ChannelXTransferDirectionSelect (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_TRANSFER_DIRECTION chTransferDirection)

PLIB_DMA_ChannelXDisabledDisablesEvents Function

Channel start/abort events will be ignored even if the channel is disabled.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDisabledDisablesEvents(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function will allow the channel start/abort events to be ignored even if the channel is disabled.

Remarks

This function implements an operation of the ChannelXDisabled feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDisabled function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXDisabledDisablesEvents ( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXDisabledDisablesEvents (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXDisabledEnablesEvents Function

Channel start/abort events will be registered even if the channel is disabled.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXDisabledEnablesEvents(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function will allow the channel register start/abort events even if the channel is disabled.

Remarks

This function implements an operation of the ChannelXDisabled feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXDisabled function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_ChannelXDisabledEnablesEvents ( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXDisabledEnablesEvents (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXPatternIgnoreByteDisable Function

Disables the pattern match ignore byte.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPatternIgnoreByteDisable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function disables the pattern match ignore byte.

Remarks

This function implements an operation of the ChannelXPatternIgnoreByte feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternIgnoreByte function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXPatternIgnoreByteDisable(DMA_ID_0, dmaChannel);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXPatternIgnoreByteDisable (DMA_MODULE_ID index,
DMA_CHANNEL channel);

PLIB_DMA_ChannelXPatternIgnoreByteEnable Function

Enables the pattern match ignore byte.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPatternIgnoreByteEnable(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function enables the pattern match ignore byte.

Remarks

This function implements an operation of the ChannelXPatternIgnoreByte feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternIgnoreByte function in your application to

automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXPatternIgnoreByteEnable(DMA_ID_0, dmaChannel);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_ChannelXPatternIgnoreByteEnable (DMA_MODULE_ID index,
DMA_CHANNEL channel);

PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled Function

Returns the state of the pattern match ignore byte.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - The pattern match ignore byte is enabled
- false - The pattern match ignore byte is disabled

Description

This function returns the state (enabled or disabled) of the pattern match ignore byte.

Remarks

This function implements an operation of the ChannelXPatternIgnoreByte feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternIgnoreByte function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool patternIsEnabled;
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
patternIsEnabled = PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled(DMA_ID_0, dmaChannel);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL channel);

PLIB_DMA_ChannelXPatternIgnoreGet Function

Returns the pattern match ignore value.

File

plib_dma.h

C

```
uint8_t PLIB_DMA_ChannelXPatternIgnoreGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- uint8_t - Pattern match ignore value

Description

This function returns the value of the pattern match ignore.

Remarks

This function implements an operation of the ChannelXPatternIgnore feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternIgnore function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t patternMatch;
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
patternMatch = PLIB_DMA_ChannelXPatternIgnoreGet ( DMA_ID_0, dmaChannel);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

uint8_t PLIB_DMA_ChannelXPatternIgnoreGet(DMA_MODULE_ID index,
DMA_CHANNEL channel);

PLIB_DMA_ChannelXPatternIgnoreSet Function

Sets the pattern match ignore value.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPatternIgnoreSet(DMA_MODULE_ID index, DMA_CHANNEL channel, uint8_t pattern);
```

Returns

None.

Description

This function sets the value of the pattern match ignore.

Remarks

This function implements an operation of the ChannelXPatternIgnore feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternIgnore function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t patternMatch = 0x8;
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
PLIB_DMA_ChannelXPatternIgnoreSet ( DMA_ID_0, dmaChannel,patternMatch);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
pattern	Pattern match ignore value

Function

void PLIB_DMA_ChannelXPatternIgnoreSet (DMA_MODULE_ID index,
DMA_CHANNEL channel, uint8_t pattern);

PLIB_DMA_ChannelXPatternLengthGet Function

Returns the pattern match length.

File

plib_dma.h

C

```
DMA_PATTERN_LENGTH PLIB_DMA_ChannelXPatternLengthGet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel);
```

Returns

- patternLen - Length of pattern match (either 1 or 2)

Description

This function returns the length of the byte matching the CHPIGN bits during a pattern match that may be ignored during the pattern match determination when the CHPIGNEN bit is set.

Remarks

This function implements an operation of the ChannelXPatternLength feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternLength function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
DMA_PATTERN_LENGTH patternLen;
patternLen = PLIB_DMA_ChannelXPatternLengthGet(DMA_ID_0, dmaChannel);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL

Function

DMA_PATTERN_LENGTH PLIB_DMA_ChannelXPatternLengthGet(DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel);

PLIB_DMA_ChannelXPatternLengthSet Function

Sets the pattern match length.

File

plib_dma.h

C

```
void PLIB_DMA_ChannelXPatternLengthSet(DMA_MODULE_ID index, DMA_CHANNEL dmaChannel, DMA_PATTERN_LENGTH patternLen);
```

Returns

None.

Description

This function sets the length of the pattern match ignore to 1 or 2 bytes.

Remarks

This function implements an operation of the ChannelXPatternLength feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXPatternLength function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL dmaChannel = DMA_CHANNEL_2;
DMA_PATTERN_LENGTH patternLen;
patternLen = DMA_PATTERN_MATCH_LENGTH_1BYTE;
PLIB_DMA_ChannelXPatternLengthSet(DMA_ID_0, dmaChannel, patternLen);
```

Parameters

Parameters	Description
dmaChannel	One of the possible DMA channels listed by DMA_CHANNEL
patternLen	Length of pattern match (either 1 or 2)

Function

void PLIB_DMA_ChannelXPatternLengthSet(DMA_MODULE_ID index,
DMA_CHANNEL dmaChannel,DMA_PATTERN_LENGTH patternLen)

i) CRC Module Interface Functions

PLIB_DMA_CRCAppendModeDisable Function

Disables the CRC append mode.

File

plib_dma.h

C

```
void PLIB_DMA_CRCAppendModeDisable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function disables the CRC append mode. The DMA transfers data from the source through the CRC obeying WBO (DMA_MODULE_ID index, write byte order) as it writes the data to the destination.

Remarks

This function implements an operation of the CRCAppendMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCAppendMode function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCAppendModeDisable( DMA_ID_0 );
```

Function

void PLIB_DMA_CRCAppendModeDisable (DMA_MODULE_ID index)

PLIB_DMA_CRCAppendModeEnable Function

Enables the CRC append mode.

File

plib_dma.h

C

```
void PLIB_DMA_CRCAppendModeEnable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function enables the CRC append mode. The DMA transfers data from the source into the CRC, but not to the destination. When a block transfer completes, the DMA writes the calculated CRC value to the location specified by the CHxDSA register.

Remarks

This function implements an operation of the CRCAppendMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCAppendMode function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCAppendModeEnable( DMA_ID_0 );
```

Function

void PLIB_DMA_CRCAppendModeEnable (DMA_MODULE_ID index)

PLIB_DMA_CRCAppendModeIsEnabled Function

Gets the enable status of the CRC append mode.

File

plib_dma.h

C

```
bool PLIB_DMA_CRCAppendModeIsEnabled(DMA_MODULE_ID index);
```

Returns

- true - CRC append mode is enabled
- false - CRC append mode is disabled

Description

This function gets the enable status of the CRC append mode.

Remarks

This function implements an operation of the CRCAppendMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCAppendMode function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool DMAcrcAppendMode;
DMAcrcAppendMode = PLIB_DMA_CRCAppendModeIsEnabled( DMA_ID_0 );
```

Function

bool PLIB_DMA_CRCAppendModeIsEnabled (DMA_MODULE_ID index)

PLIB_DMA_CRCBitOrderSelect Function

Selects the bit order for checksum calculation.

File

plib_dma.h

C

```
void PLIB_DMA_CRCBitOrderSelect(DMA_MODULE_ID index, DMA_CRC_BIT_ORDER bitOrder);
```

Returns

None.

Description

This function selects the bit order for checksum calculation.

Remarks

This function implements an operation of the CRCBitOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCBitOrder function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCBitOrderSelect ( DMA_ID_0,DMA_CRC_BIT_ORDER_LSB );
```

Parameters

Parameters	Description
bitOrder	Specifies the bit order for CRC calculation

Function

void PLIB_DMA_CRCBitOrderSelect (DMA_MODULE_ID index,
DMA_CRC_BIT_ORDER bitOrder)

PLIB_DMA_CRCByteOrderGet Function

Gets the current byte order selected by the DMA module CRC feature.

File

plib_dma.h

C

```
DMA_CRC_BYTE_ORDER PLIB_DMA_CRCByteOrderGet(DMA_MODULE_ID index);
```

Returns

- byteOrder - One of the possible byte orders specified by DMA_CRC_BYTE_ORDER

Description

This function gets the current byte order selected by the DMA module CRC feature.

Remarks

This function implements an operation of the CRCByteOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCByteOrder function in your application to automatically determine whether this feature is available.

Preconditions

The WBO bit must be set to use this function.

Example

```
DMA_CRC_BYTE_ORDER byteOrder;
```

```
byteOrder = PLIB_DMA_CRCByteOrderGet ( DMA_ID_0 );
```

Function

DMA_CRC_BYTE_ORDER PLIB_DMA_CRCByteOrderGet (DMA_MODULE_ID index)

PLIB_DMA_CRCByteOrderSelect Function

Selects the byte order.

File

plib_dma.h

C

```
void PLIB_DMA_CRCByteOrderSelect(DMA_MODULE_ID index, DMA_CRC_BYTE_ORDER byteOrder);
```

Returns

None.

Description

This function selects the byte order.

Remarks

This function implements an operation of the CRCByteOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCByteOrder function in your application to automatically determine whether this feature is available.

Preconditions

The WBO bit must be set to use this function.

Example

```
PLIB_DMA_CRCByteOrderSelect (   DMA_ID_0,
                                DMA_CRC_SWAP_HALF_WORD_ON_WORD_BOUNDARY );
```

Parameters

Parameters	Description
byteOrder	One of the possible byte orders specified by DMA_CRC_BYTE_ORDER

Function

void PLIB_DMA_CRCByteOrderSelect (DMA_MODULE_ID index,
DMA_CRC_BYTE_ORDER byteOrder)

PLIB_DMA_CRCChannelGet Function

Returns the current DMA channel to which the CRC is assigned.

File

plib_dma.h

C

```
DMA_CHANNEL PLIB_DMA_CRCChannelGet(DMA_MODULE_ID index);
```

Returns

crcChannel - One of the possible DMA channels listed by DMA_CHANNEL

Description

This function returns the current DMA channel to which the CRC is assigned.

Remarks

This function implements an operation of the CRCChannel feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCChannel function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CHANNEL crcChannel;
crcChannel = PLIB_DMA_CRCChannelGet( DMA_ID_0 );
```

Function

DMA_CHANNEL PLIB_DMA_CRCChannelGet (DMA_MODULE_ID index)

PLIB_DMA_CRCChannelSelect Function

Assigns the CRC to the specified DMA channel.

File

plib_dma.h

C

```
void PLIB_DMA_CRCChannelSelect(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

None.

Description

This function assigns the CRC feature to the specified channel.

Remarks

This function implements an operation of the CRCChannel feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCChannel function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCChannelSelect(  DMA_ID_0,
                            DMA_CHANNEL_5 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

void PLIB_DMA_CRCChannelSelect(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_CRCDataRead Function

Reads the contents of the DMA CRC data register.

File

plib_dma.h

C

```
uint32_t PLIB_DMA_CRCDataRead(DMA_MODULE_ID index);
```

Returns

- uint32_t - 32-bit CRC data

The CRC data (16-bit, 32-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the contents of the DMA CRC data register.

Remarks

This function implements an operation of the CRCData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCData function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t DMACRCdata;
DMACRCdata = PLIB_DMA_CRCDataRead ( DMA_ID_0 );
```

Function

uint32_t PLIB_DMA_CRCDataRead (DMA_MODULE_ID index)

PLIB_DMA_CRCDataWrite Function

Writes the contents of the DMA CRC data register with the specified data.

File

plib_dma.h

C

```
void PLIB_DMA_CRCDataWrite(DMA_MODULE_ID index, uint32_t DMACRCdata);
```

Returns

None.

Description

This function writes the contents of the DMA CRC data register.

Remarks

This function implements an operation of the CRCData feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCData function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t DMACRCdata = 0x0E0E0E;
PLIB_DMA_CRCDataWrite ( DMA_ID_0, DMACRCdata );
```

Function

void PLIB_DMA_CRCDataWrite (DMA_MODULE_ID index,
uint32_t DMACRCdata)

PLIB_DMA_CRCDisable Function

Disables the DMA module CRC feature.

File

plib_dma.h

C

```
void PLIB_DMA_CRCDisable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function disables the DMA module CRC feature. The channel transfers proceed normally.

Remarks

This function implements an operation of the CRC feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRC function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCDisable( DMA_ID_0 );
```

Function

void PLIB_DMA_CRCDisable (DMA_MODULE_ID index)

PLIB_DMA_CRCEnable Function

Enables the DMA module CRC feature.

File

plib_dma.h

C

```
void PLIB_DMA_CRCEnable(DMA_MODULE_ID index);
```

Returns

None.

Description

This function enables the DMA module CRC feature. The channel transfers are routed through the CRC.

Remarks

This function implements an operation of the CRC feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRC function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCEnable( DMA_ID_0 );
```

Function

void PLIB_DMA_CRCEnable (DMA_MODULE_ID index)

PLIB_DMA_CRCPolynomialLengthGet Function

Gets the current polynomial length.

File

plib_dma.h

C

```
uint8_t PLIB_DMA_CRCPolynomialLengthGet(DMA_MODULE_ID index);
```

Returns

- uint8_t - Polynomial length

Description

This function gets the current polynomial length.

Remarks

This function implements an operation of the CRCPolynomialLength feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCPolynomialLength function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t polyLength;
polyLength = PLIB_DMA_CRCPolynomialLengthGet( DMA_ID_0 );
```

Function

uint8_t PLIB_DMA_CRCPolynomialLengthGet (DMA_MODULE_ID index)

PLIB_DMA_CRCPolynomialLengthSet Function

Selects the polynomial length.

File

plib_dma.h

C

```
void PLIB_DMA_CRCPolynomialLengthSet(DMA_MODULE_ID index, uint8_t polyLength);
```

Returns

None.

Description

This function Selects the polynomial length.

Remarks

This function implements an operation of the CRCPolynomialLength feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCPolynomialLength function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t polyLength = 0x2;
PLIB_DMA_CRCPolynomialLengthSet( DMA_ID_0, polyLength );
```

Parameters

Parameters	Description
polyLength	Polynomial length

Function

void PLIB_DMA_CRCPolynomialLengthSet (DMA_MODULE_ID index,

uint8_t polyLength)

PLIB_DMA_CRCTypeGet Function

Gets the current DMA module CRC feature type.

File

plib_dma.h

C

```
DMA_CRC_TYPE PLIB_DMA_CRCTypeGet(DMA_MODULE_ID index);
```

Returns

- CRCType - One of the possible CRC checksums listed by DMA_CRC_TYPE.

Description

This function gets the DMA module CRC feature type. The CRC feature will compute either the IP header checksum or the Linear Shift Feedback Register (LFSR) checksum.

Remarks

This function implements an operation of the CRCType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCType function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
DMA_CRC_TYPE CRCType;
CRCType = PLIB_DMA_CRCTypeGet( DMA_ID_0 );
```

Function

DMA_CRC_TYPE PLIB_DMA_CRCTypeGet (DMA_MODULE_ID index)

PLIB_DMA_CRCTypeSet Function

Selects the DMA module CRC feature type.

File

plib_dma.h

C

```
void PLIB_DMA_CRCTypeSet(DMA_MODULE_ID index, DMA_CRC_TYPE CRCType);
```

Returns

None.

Description

This function selects the DMA module CRC feature type. The CRC feature will compute either the IP header checksum or the Linear Shift Feedback Register (LFSR) checksum.

Remarks

This function implements an operation of the CRCType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCType function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCTypeSet(DMA_ID_0,
                    DMA_CRC_IP_HEADER );
```

Parameters

Parameters	Description
CRCType	One of the possible CRC checksums listed by DMA_CRC_TYPE

Function

void PLIB_DMA_CRCTypeSet (DMA_MODULE_ID index,
DMA_CRC_TYPE CRCType)

PLIB_DMA_CRCWriteByteOrderAlter Function

The source data is written to the destination reordered as defined by the BYTO<1:0> bits.

File

plib_dma.h

C

```
void PLIB_DMA_CRCWriteByteOrderAlter(DMA_MODULE_ID index);
```

Returns

None.

Description

This function enables byte order alteration as specified by the BYTO<1:0> bits. The source data is written to the destination reordered as defined by the BYTO<1:0> bits.

Remarks

This function implements an operation of the CRCWriteByteOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCWriteByteOrder function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCWriteByteOrderAlter ( DMA_ID_0 );
```

Function

void PLIB_DMA_CRCWriteByteOrderAlter (DMA_MODULE_ID index)

PLIB_DMA_CRCWriteByteOrderMaintain Function

The source data is written to the destination unaltered.

File

plib_dma.h

C

```
void PLIB_DMA_CRCWriteByteOrderMaintain(DMA_MODULE_ID index);
```

Returns

None.

Description

This function disables byte order alteration. The source data is written to the destination unaltered.

Remarks

This function implements an operation of the CRCWriteByteOrder feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCWriteByteOrder function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_CRCWriteByteOrderMaintain ( DMA_ID_0 );
```

Function

void PLIB_DMA_CRCWriteByteOrderMaintain (DMA_MODULE_ID index)

PLIB_DMA_CRCXOREnableGet Function

Reads the CRC XOR register.

File

plib_dma.h

C

```
uint32_t PLIB_DMA_CRCXOREnableGet(DMA_MODULE_ID index);
```

Returns

- uint32_t - 32-bit CRC XOR enable mask data

The CRC data (16-bit, 32-bit) is device-specific. Please refer to the specific device data sheet to determine availability.

Description

This function reads the CRC XOR register.

Remarks

This function implements an operation of the CRCXOREnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCXOREnable function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t DMACRCXORdata;
DMACRCXORdata = PLIB_DMA_CRCXOREnableGet ( DMA_ID_0 );
```

Function

uint32_t PLIB_DMA_CRCXOREnableGet (DMA_MODULE_ID index)

PLIB_DMA_CRCXOREnableSet Function

Writes to the CRC XOR enable register as per the specified enable mask.

File

plib_dma.h

C

```
void PLIB_DMA_CRCXOREnableSet(DMA_MODULE_ID index, uint32_t DMACRCXOREnableMask);
```

Returns

None.

Description

This function writes to the CRC XOR enable register as per the specified enable mask. Each enabled bit will be taken as input to the shift register.

Remarks

This function implements an operation of the CRCXOREnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRCXOREnable function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t DMACRCXOREnableMask = 0x05EFFFFF;
PLIB_DMA_CRCXOREnableSet ( DMA_ID_0, DMACRCXOREnableMask );
```

Function

void PLIB_DMA_CRCXOREnableSet (DMA_MODULE_ID index,

uint32_t DMACRCXOREnableMask)

j) Global Status Functions

PLIB_DMA_CRCIsEnabled Function

Gets the enable status of the CRC feature.

File

plib_dma.h

C

```
bool PLIB_DMA_CRCIsEnabled(DMA_MODULE_ID index);
```

Returns

- true - The CRC feature is enabled
- false - The CRC feature is disabled

Description

This function gets the enable status of the CRC feature.

Remarks

This function implements an operation of the CRC feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsCRC function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool DMAcrcStatus;
DMAcrcStatus = PLIB_DMA_CRCIsEnabled( DMA_ID_0 );
```

Function

bool PLIB_DMA_CRCIsEnabled (DMA_MODULE_ID index)

PLIB_DMA_IsBusy Function

Gets the BUSY bit of the DMA controller.

File

plib_dma.h

C

```
bool PLIB_DMA_IsBusy(DMA_MODULE_ID index);
```

Returns

- true - DMA module is active
- false - DMA module is disabled and is not actively transferring data

Description

This function gets the BUSY bit of the DMA controller.

Remarks

This function implements an operation of the Busy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsBusy function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool dmaBusyStatus;
dmaBusyStatus = PLIB_DMA_IsBusy( DMA_ID_0 );
```

Function

bool PLIB_DMA_IsBusy (DMA_MODULE_ID index)

PLIB_DMA_IsEnabled Function

Returns the DMA module enable status.

File

plib_dma.h

C

```
bool PLIB_DMA_IsEnabled(DMA_MODULE_ID index);
```

Returns

- true - The DMA is enabled
- false - The DMA is disabled

Description

This function returns the DMA module enable status.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsEnableControl function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_DMA_IsEnabled( DMA_ID_0 );
```

Function

bool PLIB_DMA_IsEnabled (DMA_MODULE_ID index)

PLIB_DMA_LastBusAccessIsRead Function

Returns true if the last DMA bus access was a read.

File

plib_dma.h

C

```
bool PLIB_DMA_LastBusAccessIsRead(DMA_MODULE_ID index);
```

Returns

- true - The last bus access was a read
- false - The last bus access was not a read

Description

This function returns true if the last DMA bus access was a read.

Remarks

This function implements an operation of the LastBusAccess feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsLastBusAccess function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool dmaLastBusAccessType;
dmaLastBusAccessType = PLIB_DMA_LastBusAccessIsRead( DMA_ID_0 );
```

Function

bool PLIB_DMA_LastBusAccessIsRead (DMA_MODULE_ID index)

PLIB_DMA_LastBusAccessIsWrite Function

Returns true if the last DMA bus access was a write.

File

plib_dma.h

C

```
bool PLIB_DMA_LastBusAccessIsWrite(DMA_MODULE_ID index);
```

Returns

- true - The last bus access was a write operation
- false - The last bus access was not a write operation

Description

This function returns true if the last DMA bus access was a write operation.

Remarks

This function implements an operation of the LastBusAccess feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsLastBusAccess function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool dmaLastBusAccessType;
dmaLastBusAccessType = PLIB_DMA_LastBusAccessIsWrite( DMA_ID_0 );
```

Function

bool PLIB_DMA_LastBusAccessIsWrite (DMA_MODULE_ID index)

PLIB_DMA_RecentAddressAccessed Function

Returns the address of the most recent DMA access.

File

plib_dma.h

C

```
uint32_t PLIB_DMA_RecentAddressAccessed(DMA_MODULE_ID index);
```

Returns

- uint32_t - The most recent address accessed by the DMA

Description

This function returns the address of the most recent DMA access.

Remarks

This function implements an operation of the RecentAddress feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsRecentAddress function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t dmaLastAddressAccessed;
dmaLastAddressAccessed = PLIB_DMA_RecentAddressAccessed( DMA_ID_0 );
```

Function

uint32_t PLIB_DMA_RecentAddressAccessed (DMA_MODULE_ID index)

PLIB_DMA_SuspendIsEnabled Function

Returns the DMA suspend status.

File

plib_dma.h

C

```
bool PLIB_DMA_SuspendIsEnabled(DMA_MODULE_ID index);
```

Returns

- true - The DMA transfers are suspended
- false - The DMA operates normally

Description

This function returns the DMA suspend status.

Remarks

This function implements an operation of the Suspend feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsSuspend function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool dmaSuspendStatus;
dmaSuspendStatus = PLIB_DMA_SuspendIsEnabled( DMA_ID_0 );
```

Function

bool PLIB_DMA_SuspendIsEnabled (DMA_MODULE_ID index)

k) Channel Status Functions

PLIB_DMA_ChannelXAutoIsEnabled Function

Returns the channel automatic enable status.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXAutoIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - Channel automatic enable is on
- false - Channel automatic enable is off

Description

This function returns the channel automatic enable status.

Remarks

This function implements an operation of the ChannelXAuto feature. This feature may not be available on all devices. Please refer to the specific

device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXAuto function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool ChAutoEnableStatus;
ChAutoEnableStatus = PLIB_DMA_ChannelXAutoIsEnabled(DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXAutoIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXBufferedDataIsWritten Function

Returns the buffered data write status for the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXBufferedDataIsWritten(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - The content of the DMA buffer has not been written to the location specified in the destination/source address or in Null Write mode
- false - The content of the DMA buffer has been written to the location specified in the destination/source address or in Null Write mode

Description

This function returns the buffered data write status for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
bool chBuffWriteStatus;
chBuffWriteStatus = PLIB_DMA_ChannelXBufferedDataIsWritten( DMA_ID_0,
                                                            DMA_CHANNEL_3 );
```

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXBufferedDataIsWritten (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXBusyIsBusy Function

Returns the busy status of the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXBusyIsBusy(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - The channel is active or has been enabled
- false - The channel is inactive or has been disabled

Description

This function returns the busy status of the specified channel.

Remarks

This function implements an operation of the ChannelXBusy feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXBusy function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool chBusyStatus;
chBusyStatus = PLIB_DMA_ChannelXBusyIsBusy ( DMA_ID_0, DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXBusyIsBusy (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXChainIsEnabled Function

Returns the chain status of the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXChainIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - The channel chain is on for this channel
- false - The channel chain is off for this channel

Description

This function returns the chain status of the specified channel.

Remarks

This function implements an operation of the ChannelXChainEnbl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelXChainEnbl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool ChchainStatus;
ChchainStatus = PLIB_DMA_ChannelXChainIsEnabled( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXChainIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXCollisionStatus Function

Returns the status of the specified collision type for the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXCollisionStatus(DMA_MODULE_ID index, DMA_CHANNEL channel, DMA_CHANNEL_COLLISION collisonType);
```

Returns

- true - A collision specified by collisonType was detected
- false - No collision of type collisonType was detected

Description

This function returns the status of the specified collision type for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
bool memWriteCollisionStatus;
memWriteCollisionStatus = PLIB_DMA_ChannelXMemoryWriteCollisionStatus(
                                        DMA_ID_0,
                                        DMA_CHANNEL_3,
                                        DMA_CHANNEL_COLLISION_MEMORY );
```

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL
collisonType	Collision type listed by DMA_CHANNEL_COLLISION

Function

bool PLIB_DMA_ChannelXCollisionStatus (DMA_MODULE_ID index,
DMA_CHANNEL channel,
DMA_CHANNEL_COLLISION collisonType)

PLIB_DMA_ChannelXEventIsDetected Function

Returns the event status on the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXEventIsDetected(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - An event was detected
- false - No events were detected

Description

This function returns the event status on the specified channel.

Remarks

This function implements an operation of the ChannelXEvent feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_DMA_ExistsChannelXEvent function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool channeleventStatus;
channeleventStatus = PLIB_DMA_ChannelXEventIsDetected(  DMA_ID_0,
                                                        DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXEventIsDetected (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXIsEnabled Function

Return the enable status of the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - The specified DMA channel is enabled
- false - The specified DMA channel is disabled

Description

This function will return the enable status of the specified channel.

Remarks

This function implements an operation of the ChannelX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelX function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool chEnableStatus;
chEnableStatus = PLIB_DMA_ChannelXIsEnabled ( DMA_ID_0, DMA_CHANNEL_2 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXNullWriteModeIsEnabled Function

Returns the enable status of the Null Write mode for the specified channel.

File

plib_dma.h

C

```
bool PLIB_DMA_ChannelXNullWriteModeIsEnabled(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

- true - Null write mode is enabled for this channel
- false - Null write mode is disabled for this channel

Description

This function returns the enable status of the Null Write mode for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
bool chNullWriteStatus;
chNullWriteStatus = PLIB_DMA_ChannelXNullWriteModeIsEnabled (
                                                        DMA_ID_0,
                                                        DMA_CHANNEL_4 );
```

Parameters

Parameters	Description
channel	One of the possible DMA channels listed by DMA_CHANNEL

Function

bool PLIB_DMA_ChannelXNullWriteModeIsEnabled (DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelXPingPongModeGet Function

Returns the Ping-Pong mode status for the specified channel.

File

plib_dma.h

C

```
DMA_PING_PONG_MODE PLIB_DMA_ChannelXPingPongModeGet(DMA_MODULE_ID index, DMA_CHANNEL channel);
```

Returns

mode - One of the possible Ping-Pong modes

Description

This function returns the Ping-Pong mode status for the specified channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
DMA_PING_PONG_MODE chPingPongStatus;
chPingPongStatus = PLIB_DMA_ChannelXPingPongModeGet(DMA_ID_0,
                                                    DMA_CHANNEL_3 );
if (DMA_PING_PONG_SECONDARY == chPingPongStatus)
    {
        \Application
    }
```

Parameters

Parameters	Description
channel	One of the existing DMA channels listed by DMA_CHANNEL

Function

DMA_PING_PONG_MODE PLIB_DMA_ChannelXPingPongModeGet(DMA_MODULE_ID index,
DMA_CHANNEL channel)

PLIB_DMA_ChannelBitsGet Function

Returns the DMA channel bits.

File

plib_dma.h

C

```
uint8_t PLIB_DMA_ChannelBitsGet(DMA_MODULE_ID index);
```

Returns

- uint8_t - DMA channel bits

Description

This function returns the channel bits.

Remarks

This function implements an operation of the ChannelBits feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_DMA_ExistsChannelBits function in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t dmaChBits;
dmaChBits = PLIB_DMA_ChannelBitsGet( DMA_ID_0 );
```

Function

uint8_t PLIB_DMA_ChannelBitsGet (DMA_MODULE_ID index)

l) Feature Existence Functions

PLIB_DMA_ExistsAbortTransfer Function

Identifies whether the AbortTransfer feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsAbortTransfer(DMA_MODULE_ID index);
```

Returns

- true - The AbortTransfer feature is supported on the device
- false - The AbortTransfer feature is not supported on the device

Description

This function identifies whether the AbortTransfer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_AbortTransferSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsAbortTransfer(DMA_MODULE_ID index)

PLIB_DMA_ExistsBusy Function

Identifies whether the Busy feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsBusy(DMA_MODULE_ID index);
```

Returns

- true - The Busy feature is supported on the device
- false - The Busy feature is not supported on the device

Description

This function identifies whether the Busy feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_BusyActiveSet
- PLIB_DMA_BusyActiveReset
- PLIB_DMA_IsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsBusy(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelBits Function

Identifies whether the ChannelBits feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelBits(DMA_MODULE_ID index);
```

Returns

- true - The ChannelBits feature is supported on the device
- false - The ChannelBits feature is not supported on the device

Description

This function identifies whether the ChannelBits feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelBitsGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelBits(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelX Function

Identifies whether the ChannelX feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelX(DMA_MODULE_ID index);
```

Returns

- true - The ChannelX feature is supported on the device
- false - The ChannelX feature is not supported on the device

Description

This function identifies whether the ChannelX feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXEnable
- PLIB_DMA_ChannelXIsEnabled
- PLIB_DMA_ChannelXDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelX(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXAbortIRQ Function

Identifies whether the ChannelXAbortIRQ feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXAbortIRQ(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXAbortIRQ feature is supported on the device
- false - The ChannelXAbortIRQ feature is not supported on the device

Description

This function identifies whether the ChannelXAbortIRQ feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelXAbortIRQSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXAbortIRQ(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXAuto Function

Identifies whether the ChannelXAuto feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXAuto(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXAuto feature is supported on the device
- false - The ChannelXAuto feature is not supported on the device

Description

This function identifies whether the ChannelXAuto feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXAutoEnable
- PLIB_DMA_ChannelXAutoDisable
- PLIB_DMA_ChannelXAutoIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXAuto(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXBusy Function

Identifies whether the ChannelXBusy feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXBusy(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXBusy feature is supported on the device
- false - The ChannelXBusy feature is not supported on the device

Description

This function identifies whether the ChannelXBusy feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXBusyActiveSet
- PLIB_DMA_ChannelXBusyInActiveSet
- PLIB_DMA_ChannelXBusyIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXBusy(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXCellProgressPointer Function

Identifies whether the ChannelXCellProgressPointer feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXCellProgressPointer(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXCellProgressPointer feature is supported on the device
- false - The ChannelXCellProgressPointer feature is not supported on the device

Description

This function identifies whether the ChannelXCellProgressPointer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelXCellProgressPointerGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXCellProgressPointer(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXCellSize Function

Identifies whether the ChannelXCellSize feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXCellSize(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXCellSize feature is supported on the device
- false - The ChannelXCellSize feature is not supported on the device

Description

This function identifies whether the ChannelXCellSize feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXCellSizeGet
- PLIB_DMA_ChannelXCellSizeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXCellSize(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXChain Function

Identifies whether the ChannelXChain feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXChain(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXChain feature is supported on the device
- false - The ChannelXChain feature is not supported on the device

Description

This function identifies whether the ChannelXChain feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXChainToLower
- PLIB_DMA_ChannelXChainToHigher

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXChain(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXChainEnbl Function

Identifies whether the ChannelXChainEnbl feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXChainEnbl(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXChainEnbl feature is supported on the device
- false - The ChannelXChainEnbl feature is not supported on the device

Description

This function identifies whether the ChannelXChainEnbl feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXChainEnable
- PLIB_DMA_ChannelXChainDisable
- PLIB_DMA_ChannelXChainIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXChainEnbl(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXDestinationPointer Function

Identifies whether the ChannelXDestinationPointer feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXDestinationPointer(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXDestinationPointer feature is supported on the device
- false - The ChannelXDestinationPointer feature is not supported on the device

Description

This function identifies whether the ChannelXDestinationPointer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelXDestinationPointerGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXDestinationPointer(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXDestinationSize Function

Identifies whether the ChannelXDestinationSize feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXDestinationSize(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXDestinationSize feature is supported on the device
- false - The ChannelXDestinationSize feature is not supported on the device

Description

This function identifies whether the ChannelXDestinationSize feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXDestinationSizeGet
- PLIB_DMA_ChannelXDestinationSizeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXDestinationSize(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXDestinationStartAddress Function

Identifies whether the ChannelXDestinationStartAddress feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXDestinationStartAddress(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXDestinationStartAddress feature is supported on the device
- false - The ChannelXDestinationStartAddress feature is not supported on the device

Description

This function identifies whether the ChannelXDestinationStartAddress feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXDestinationStartAddressGet
- PLIB_DMA_ChannelXDestinationStartAddressSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXDestinationStartAddress(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXDisabled Function

Identifies whether the ChannelXDisabled feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXDisabled(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXDisabled feature is supported on the device
- false - The ChannelXDisabled feature is not supported on the device

Description

This function identifies whether the ChannelXDisabled feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXDisabledEnablesEvents
- PLIB_DMA_ChannelXDisabledDisablesEvents

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXDisabled(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXEvent Function

Identifies whether the ChannelXEvent feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXEvent(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXEvent feature is supported on the device
- false - The ChannelXEvent feature is not supported on the device

Description

This function identifies whether the ChannelXEvent feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelXEventIsDetected

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXEvent(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXINTSource Function

Identifies whether the ChannelXINTSource feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXINTSource(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXINTSource feature is supported on the device
- false - The ChannelXINTSource feature is not supported on the device

Description

This function identifies whether the ChannelXINTSource feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXINTSourceEnable
- PLIB_DMA_ChannelXINTSourceDisable
- PLIB_DMA_ChannelXINTSourceIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXINTSource(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXINTSourceFlag Function

Identifies whether the ChannelXINTSourceFlag feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXINTSourceFlag(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXINTSourceFlag feature is supported on the device
- false - The ChannelXINTSourceFlag feature is not supported on the device

Description

This function identifies whether the ChannelXINTSourceFlag feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXINTSourceFlagGet
- PLIB_DMA_ChannelXINTSourceFlagSet
- PLIB_DMA_ChannelXINTSourceFlagClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXINTSourceFlag(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXPatternData Function

Identifies whether the ChannelXPatternData feature exists on the DMA module

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXPatternData(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXPatternData feature is supported on the device
- false - The ChannelXPatternData feature is not supported on the device

Description

This function identifies whether the ChannelXPatternData feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXPatternDataGet
- PLIB_DMA_ChannelXPatternDataSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXPatternData(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXPatternIgnore Function

Identifies whether the ChannelXPatternIgnore feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXPatternIgnore(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXPatternIgnore feature is supported on the device
- false - The ChannelXPatternIgnore feature is not supported on the device

Description

This function identifies whether the ChannelXPatternIgnore feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXPatternIgnoreSet
- PLIB_DMA_ChannelXPatternIgnoreGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXPatternIgnore(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXPatternIgnoreByte Function

Identifies whether the ChannelXPatternIgnoreByte feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXPatternIgnoreByte(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXPatternIgnoreByte feature is supported on the device
- false - The ChannelXPatternIgnoreByte feature is not supported on the device

Description

This function identifies whether the ChannelXPatternIgnoreByte feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXPatternIgnoreByteEnable
- PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled
- PLIB_DMA_ChannelXPatternIgnoreByteDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXPatternIgnoreByte(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXPatternLength Function

Identifies whether the ChannelXPatternLength feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXPatternLength(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXPatternLength feature is supported on the device
- false - The ChannelXPatternLength feature is not supported on the device

Description

This function identifies whether the ChannelXPatternLength feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXPatternLengthSet
- PLIB_DMA_ChannelXPatternLengthGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXPatternLength(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXPriority Function

Identifies whether the ChannelXPriority feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXPriority(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXPriority feature is supported on the device
- false - The ChannelXPriority feature is not supported on the device

Description

This function identifies whether the ChannelXPriority feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXPrioritySelect
- PLIB_DMA_ChannelXPriorityGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXPriority(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXSourcePointer Function

Identifies whether the ChannelXSourcePointer feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXSourcePointer(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXSourcePointer feature is supported on the device
- false - The ChannelXSourcePointer feature is not supported on the device

Description

This function identifies whether the ChannelXSourcePointer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelXSourcePointerGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXSourcePointer(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXSourceSize Function

Identifies whether the ChannelXSourceSize feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXSourceSize(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXSourceSize feature is supported on the device
- false - The ChannelXSourceSize feature is not supported on the device

Description

This function identifies whether the ChannelXSourceSize feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXSourceSizeGet
- PLIB_DMA_ChannelXSourceSizeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXSourceSize(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXSourceStartAddress Function

Identifies whether the ChannelXSourceStartAddress feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXSourceStartAddress(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXSourceStartAddress feature is supported on the device
- false - The ChannelXSourceStartAddress feature is not supported on the device

Description

This function identifies whether the ChannelXSourceStartAddress feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXSourceStartAddressGet
- PLIB_DMA_ChannelXSourceStartAddressSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXSourceStartAddress(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXStartIRQ Function

Identifies whether the ChannelXStartIRQ feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXStartIRQ(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXStartIRQ feature is supported on the device
- false - The ChannelXStartIRQ feature is not supported on the device

Description

This function identifies whether the ChannelXStartIRQ feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_ChannelXStartIRQSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXStartIRQ(DMA_MODULE_ID index)

PLIB_DMA_ExistsChannelXTrigger Function

Identifies whether the ChannelXTrigger feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsChannelXTrigger(DMA_MODULE_ID index);
```

Returns

- true - The ChannelXTrigger feature is supported on the device
- false - The ChannelXTrigger feature is not supported on the device

Description

This function identifies whether the ChannelXTrigger feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_ChannelXTriggerEnable
- PLIB_DMA_ChannelXTriggerIsEnabled
- PLIB_DMA_ChannelXTriggerDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsChannelXTrigger(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRC Function

Identifies whether the CRC feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRC(DMA_MODULE_ID index);
```

Returns

- true - The CRC feature is supported on the device
- false - The CRC feature is not supported on the device

Description

This function identifies whether the CRC feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCEnable
- PLIB_DMA_CRCDisable
- PLIB_DMA_CRCIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRC(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCAppendMode Function

Identifies whether the CRCAppendMode feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCAppendMode(DMA_MODULE_ID index);
```

Returns

- true - The CRCAppendMode feature is supported on the device
- false - The CRCAppendMode feature is not supported on the device

Description

This function identifies whether the CRCAppendMode feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCAppendModeEnable
- PLIB_DMA_CRCAppendModeDisable
- PLIB_DMA_CRCAppendModeIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCAppendMode(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCBitOrder Function

Identifies whether the CRCBitOrder feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCBitOrder(DMA_MODULE_ID index);
```

Returns

- true - The CRCBitOrder feature is supported on the device
- false - The CRCBitOrder feature is not supported on the device

Description

This function identifies whether the CRCBitOrder feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_CRCBitOrderSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCBitOrder(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCByteOrder Function

Identifies whether the CRCByteOrder feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCByteOrder(DMA_MODULE_ID index);
```

Returns

- true - The CRCByteOrder feature is supported on the device
- false - The CRCByteOrder feature is not supported on the device

Description

This function identifies whether the CRCByteOrder feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCByteOrderSelect
- PLIB_DMA_CRCByteOrderGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCByteOrder(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCChannel Function

Identifies whether the CRCChannel feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCChannel(DMA_MODULE_ID index);
```

Returns

- true - The CRCChannel feature is supported on the device
- false - The CRCChannel feature is not supported on the device

Description

This function identifies whether the CRCChannel feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCChannelSelect
- PLIB_DMA_CRCChannelGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCChannel(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCData Function

Identifies whether the CRCData feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCData(DMA_MODULE_ID index);
```

Returns

- true - The CRCData feature is supported on the device
- false - The CRCData feature is not supported on the device

Description

This function identifies whether the CRCData feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCDataRead
- PLIB_DMA_CRCDataWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCData(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCPolynomialLength Function

Identifies whether the CRCPolynomialLength feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCPolynomialLength(DMA_MODULE_ID index);
```

Returns

- true - The CRCPolynomialLength feature is supported on the device
- false - The CRCPolynomialLength feature is not supported on the device

Description

This function identifies whether the CRCPolynomialLength feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCPolynomialLengthSet
- PLIB_DMA_CRCPolynomialLengthGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCPolynomialLength(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCType Function

Identifies whether the CRCType feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCType(DMA_MODULE_ID index);
```

Returns

- true - The CRCType feature is supported on the device
- false - The CRCType feature is not supported on the device

Description

This function identifies whether the CRCType feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCTypeGet
- PLIB_DMA_CRCTypeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCType(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCWriteByteOrder Function

Identifies whether the CRCWriteByteOrder feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCWriteByteOrder(DMA_MODULE_ID index);
```

Returns

- true - The CRCWriteByteOrder feature is supported on the device
- false - The CRCWriteByteOrder feature is not supported on the device

Description

This function identifies whether the CRCWriteByteOrder feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCWriteByteOrderAlter
- PLIB_DMA_CRCWriteByteOrderMaintain

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCWriteByteOrder(DMA_MODULE_ID index)

PLIB_DMA_ExistsCRCXOREnable Function

Identifies whether the CRCXOREnable feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsCRCXOREnable(DMA_MODULE_ID index);
```

Returns

- true - The CRCXOREnable feature is supported on the device
- false - The CRCXOREnable feature is not supported on the device

Description

This function identifies whether the CRCXOREnable feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_CRCXOREnableSet
- PLIB_DMA_CRCXOREnableGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsCRCXOREnable(DMA_MODULE_ID index)

PLIB_DMA_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsEnableControl(DMA_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_Enable
- PLIB_DMA_Disable
- PLIB_DMA_IsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsEnableControl(DMA_MODULE_ID index)

PLIB_DMA_ExistsLastBusAccess Function

Identifies whether the LastBusAccess feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsLastBusAccess(DMA_MODULE_ID index);
```

Returns

- true - The LastBusAccess feature is supported on the device
- false - The LastBusAccess feature is not supported on the device

Description

This function identifies whether the LastBusAccess feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_LastBusAccessIsRead
- PLIB_DMA_LastBusAccessIsWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsLastBusAccess(DMA_MODULE_ID index)

PLIB_DMA_ExistsRecentAddress Function

Identifies whether the RecentAddress feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsRecentAddress(DMA_MODULE_ID index);
```

Returns

- true - The RecentAddress feature is supported on the device
- false - The RecentAddress feature is not supported on the device

Description

This function identifies whether the RecentAddress feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_RecentAddressAccessed

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsRecentAddress(DMA_MODULE_ID index)

PLIB_DMA_ExistsStartTransfer Function

Identifies whether the StartTransfer feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsStartTransfer(DMA_MODULE_ID index);
```

Returns

- true - The StartTransfer feature is supported on the device
- false - The StartTransfer feature is not supported on the device

Description

This function identifies whether the StartTransfer feature is available on the DMA module. When this function returns true, this function is supported on the device:

- PLIB_DMA_StartTransferSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsStartTransfer(DMA_MODULE_ID index)

PLIB_DMA_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsStopInIdle(DMA_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_StopInIdleEnable
- PLIB_DMA_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsStopInIdle(DMA_MODULE_ID index)

PLIB_DMA_ExistsSuspend Function

Identifies whether the Suspend feature exists on the DMA module.

File

plib_dma.h

C

```
bool PLIB_DMA_ExistsSuspend(DMA_MODULE_ID index);
```

Returns

- true - The Suspend feature is supported on the device
- false - The Suspend feature is not supported on the device

Description

This function identifies whether the Suspend feature is available on the DMA module. When this function returns true, these functions are supported on the device:

- PLIB_DMA_SuspendEnable
- PLIB_DMA_SuspendDisable
- PLIB_DMA_SuspendIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DMA_ExistsSuspend(DMA_MODULE_ID index)

m) Data Types and Constants

DMA_ADDRESS_OFFSET_TYPE Enumeration

Lists the possible DMA address offsets.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_ADDRESS_OFFSET_PRIMARY,
  DMA_ADDRESS_OFFSET_SECONDARY
} DMA_ADDRESS_OFFSET_TYPE;
```

Members

Members	Description
DMA_ADDRESS_OFFSET_PRIMARY	address offset-A
DMA_ADDRESS_OFFSET_SECONDARY	address offset-B

Description

DMA address offsets

This enumeration lists all the possible DMA address offsets.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL Enumeration

Lists the possible DMA channels available.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CHANNEL_0,
  DMA_CHANNEL_1,
  DMA_CHANNEL_2,
  DMA_CHANNEL_3,
  DMA_CHANNEL_4,
  DMA_CHANNEL_5,
  DMA_CHANNEL_6,
  DMA_CHANNEL_7
} DMA_CHANNEL;
```

Members

Members	Description
DMA_CHANNEL_0	DMA Channel 0
DMA_CHANNEL_1	DMA Channel 1

DMA_CHANNEL_2	DMA Channel 2
DMA_CHANNEL_3	DMA Channel 3
DMA_CHANNEL_4	DMA Channel 4
DMA_CHANNEL_5	DMA Channel 5
DMA_CHANNEL_6	DMA Channel 6
DMA_CHANNEL_7	DMA Channel 7

Description

DMA channel

This enumeration lists all the possible DMA channels available.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL_ADDRESSING_MODE Enumeration

Lists the possible channel addressing modes.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_ADDRESSING_REGISTER_INDIRECT_WITH_POST_INCREMENT,
  DMA_ADDRESSING_REGISTER_INDIRECT_WITHOUT_POST_INCREMENT,
  DMA_ADDRESSING_PERIPHERAL_INDIRECT
} DMA_CHANNEL_ADDRESSING_MODE;
```

Members

Members	Description
DMA_ADDRESSING_REGISTER_INDIRECT_WITH_POST_INCREMENT	register indirect with post increment
DMA_ADDRESSING_REGISTER_INDIRECT_WITHOUT_POST_INCREMENT	register indirect without post increment
DMA_ADDRESSING_PERIPHERAL_INDIRECT	peripheral indirect addressing

Description

DMA channel addressing modes

This enumeration lists all the possible channel addressing modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL_COLLISION Enumeration

Lists the possible channel collision types.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CHANNEL_COLLISION_MEMORY,
  DMA_CHANNEL_COLLISION_PERIPHERAL,
  DMA_CHANNEL_COLLISION_TRANSFER_REQUEST
} DMA_CHANNEL_COLLISION;
```

Members

Members	Description
DMA_CHANNEL_COLLISION_MEMORY	Memory Write Collision
DMA_CHANNEL_COLLISION_PERIPHERAL	Peripheral Write collision
DMA_CHANNEL_COLLISION_TRANSFER_REQUEST	Transfer request collision

Description

DMA channel collision types

This enumeration lists all the possible channel collision types.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

DMA_CHANNEL_DATA_SIZE Enumeration

Lists the possible data sizes for the DMA channel.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CHANNEL_DATA_8,
  DMA_CHANNEL_DATA_16
} DMA_CHANNEL_DATA_SIZE;
```

Members

Members	Description
DMA_CHANNEL_DATA_8	8 bit data size
DMA_CHANNEL_DATA_16	16 bit data size

Description

DMA channel data size

This enumeration lists all the possible data sizes for the DMA channels.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL_PRIORITY Enumeration

Lists the possible channel priorities.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CHANNEL_PRIORITY_0,
  DMA_CHANNEL_PRIORITY_1,
  DMA_CHANNEL_PRIORITY_2,
  DMA_CHANNEL_PRIORITY_3,
  DMA_CHANNEL_ROUND_ROBIN,
  DMA_CHANNEL_FIXED_PRIORITY
} DMA_CHANNEL_PRIORITY;
```

Members

Members	Description
DMA_CHANNEL_PRIORITY_0	priority 0
DMA_CHANNEL_PRIORITY_1	priority 1
DMA_CHANNEL_PRIORITY_2	priority 2
DMA_CHANNEL_PRIORITY_3	priority 3
DMA_CHANNEL_ROUND_ROBIN	round-robin scheme
DMA_CHANNEL_FIXED_PRIORITY	fixed priority scheme

Description

DMA channel priority types

This enumeration lists all the possible channel priorities.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL_TRANSFER_DIRECTION Enumeration

Lists the possible data transfer directions.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_READ_FROM_MEMORY_WRITE_TO_PERIPHERAL,
  DMA_READ_FROM_PERIPHERAL_WRITE_TO_MEMORY,
  DMA_READ_FROM_MEMORY_WRITE_TO_MEMORY,
  DMA_READ_FROM_PERIPHERAL_WRITE_TO_PERIPHERAL
} DMA_CHANNEL_TRANSFER_DIRECTION;
```

Members

Members	Description
DMA_READ_FROM_MEMORY_WRITE_TO_PERIPHERAL	DMA transfer happens from the memory to peripheral
DMA_READ_FROM_PERIPHERAL_WRITE_TO_MEMORY	DMA transfer happens from the peripheral to memory
DMA_READ_FROM_MEMORY_WRITE_TO_MEMORY	DMA transfer happens from the memory to memory
DMA_READ_FROM_PERIPHERAL_WRITE_TO_PERIPHERAL	DMA transfer happens from the peripheral to peripheral

Description

DMA channel data transfer direction

This enumeration lists all the possible data transfer directions.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL_TRIGGER_TYPE Enumeration

Lists the possible DMA channel triggers.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CHANNEL_TRIGGER_TRANSFER_START,
  DMA_CHANNEL_TRIGGER_TRANSFER_ABORT,
  DMA_CHANNEL_TRIGGER_PATTERN_MATCH_ABORT
} DMA_CHANNEL_TRIGGER_TYPE;
```

Members

Members	Description
DMA_CHANNEL_TRIGGER_TRANSFER_START	Trigger for DMA transfer start
DMA_CHANNEL_TRIGGER_TRANSFER_ABORT	Trigger for DMA transfer abort
DMA_CHANNEL_TRIGGER_PATTERN_MATCH_ABORT	Trigger for DMA transfer abort via pattern match

Description

DMA trigger types

This enumeration lists all the possible DMA channel triggers.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CRC_BIT_ORDER Enumeration

Lists the possible CRC calculation bit orders

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CRC_BIT_ORDER_LSB,
  DMA_CRC_BIT_ORDER_MSB
} DMA_CRC_BIT_ORDER;
```

Members

Members	Description
DMA_CRC_BIT_ORDER_LSB	CRC is calculated least significant bit first
DMA_CRC_BIT_ORDER_MSB	CRC is calculated most significant bit first

Description

DMA CRC calculation bit order

This enumeration lists all the possible CRC calculation bit orders

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CRC_BYTE_ORDER Enumeration

Lists the possible byte orders.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CRC_BYTEORDER_NO_SWAPPING,
  DMA_CRC_SWAP_BYTE_ON_WORD_BOUNDARY,
  DMA_CRC_SWAP_HALF_WORD_ON_WORD_BOUNDARY,
  DMA_CRC_SWAP_BYTE_ON_HALF_WORD_BOUNDARY
} DMA_CRC_BYTE_ORDER;
```

Members

Members	Description
DMA_CRC_BYTEORDER_NO_SWAPPING	No swapping, Source byte order
DMA_CRC_SWAP_BYTE_ON_WORD_BOUNDARY	Endian byte swap on word boundaries (reverse source byte order)
DMA_CRC_SWAP_HALF_WORD_ON_WORD_BOUNDARY	Swap half-words on word boundaries (reverse source half-word order with source byte order per half-word)
DMA_CRC_SWAP_BYTE_ON_HALF_WORD_BOUNDARY	Endian byte swap on half-word boundaries (reverse source half-word order with reverse source byte order per half-word)

Description

CRC Byte order type

This enumeration lists all the possible byte orders handled by CRC module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CRC_TYPE Enumeration

Lists the possible checksums.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CRC_IP_HEADER,
  DMA_CRC_LFSR
} DMA_CRC_TYPE;
```

Members

Members	Description
DMA_CRC_IP_HEADER	CRC module will calculate IP header checksum
DMA_CRC_LFSR	CRC module will calculate Linear Feedback Shift Registers checksum

Description

CRC type

This enumeration lists all the possible checksums handled by CRC module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_DESTINATION_ADDRESSING_MODE Enumeration

Lists the possible destination addressing modes.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_ADDRESSING_DESTINATION_UNCHANGED,
  DMA_ADDRESSING_DESTINATION_INCREMENT_BASED_ON_SIZE,
  DMA_ADDRESSING_DESTINATION_DECREMENT_BASED_ON_SIZE,
  DMA_ADDRESSING_DESTINATION_PERIPHERAL_INDIRECT
} DMA_DESTINATION_ADDRESSING_MODE;
```

Members

Members	Description
DMA_ADDRESSING_DESTINATION_UNCHANGED	DMADST remains unchanged after a transfer completion
DMA_ADDRESSING_DESTINATION_INCREMENT_BASED_ON_SIZE	DMADST is incremented based on SIZE bit after a transfer completion
DMA_ADDRESSING_DESTINATION_DECREMENT_BASED_ON_SIZE	DMADST is decremented based on SIZE bit after a transfer completion
DMA_ADDRESSING_DESTINATION_PERIPHERAL_INDIRECT	DMADST is used in peripheral indirect addressing and remains unchanged

Description

DMA destination addressing modes

This enumeration lists all the possible destination addressing modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_INT_TYPE Enumeration

Lists the possible Interrupt types for a particular channel-X

File

help_plib_dma.h

C

```
typedef enum {
  DMA_INT_ADDRESS_ERROR,
```

```
  DMA_INT_TRANSFER_ABORT,
  DMA_INT_CELL_TRANSFER_COMPLETE,
  DMA_INT_BLOCK_TRANSFER_COMPLETE,
  DMA_INT_DESTINATION_HALF_FULL,
  DMA_INT_DESTINATION_DONE,
  DMA_INT_SOURCE_HALF_EMPTY,
  DMA_INT_SOURCE_DONE,
  DMA_INT_COUNT_HALF_DONE,
  DMA_INT_CHANNEL_OVERRUN,
  DMA_INT_COMPLETE,
  DMA_INT_LOW_ADDRESS_LIMIT,
  DMA_INT_HIGH_ADDRESS_LIMIT
} DMA_INT_TYPE;
```

Members

Members	Description
DMA_INT_ADDRESS_ERROR	Channel address error interrupt
DMA_INT_TRANSFER_ABORT	channel transfer abort interrupt
DMA_INT_CELL_TRANSFER_COMPLETE	channel cell transfer complete
DMA_INT_BLOCK_TRANSFER_COMPLETE	Channel block transfer complete
DMA_INT_DESTINATION_HALF_FULL	Channel destination half full interrupt
DMA_INT_DESTINATION_DONE	Channel destination done interrupt
DMA_INT_SOURCE_HALF_EMPTY	Channel source half empty interrupt
DMA_INT_SOURCE_DONE	Channel source done interrupt
DMA_INT_COUNT_HALF_DONE	DMA count has reached its halfway
DMA_INT_CHANNEL_OVERRUN	channel overrun. channel is triggered while it is still completing the operation based on the previous trigger
DMA_INT_COMPLETE	DMA complete operation interrupt
DMA_INT_LOW_ADDRESS_LIMIT	DMA low address limit interrupt
DMA_INT_HIGH_ADDRESS_LIMIT	DMA high address limit interrupt

Description

DMA channel X interrupt types

This enumeration lists all the possible interrupt types for a channel. Not to be confused with the DMA trigger/abort interrupt sources (supported only for few parts).

Remarks

Not to be confused with the DMA trigger/abort interrupt sources (supported only for few parts). This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_MODULE_ID Enumeration

Possible instances of the DMA module.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_ID_0
} DMA_MODULE_ID;
```

Members

Members	Description
DMA_ID_0	first instance of the DMA

Description

DMA module ID

This data type defines the possible instances of the DMA module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_PATTERN_LENGTH Enumeration

Gives the DMA pattern length

File

help_plib_dma.h

C

```
typedef enum {
  DMA_PATTERN_MATCH_LENGTH_1BYTE,
  DMA_PATTERN_MATCH_LENGTH_2BYTES
} DMA_PATTERN_LENGTH;
```

Members

Members	Description
DMA_PATTERN_MATCH_LENGTH_1BYTE	The Length of matching pattern with CHPIGN<7:0> in DCHxCON<31:24> is 1 BYTE
DMA_PATTERN_MATCH_LENGTH_2BYTES	The Length of matching pattern with CHPIGN<7:0> in DCHxCON<31:24> is 2 BYTES

Description

DMA pattern length

This enumeration gives the length of the pattern to be matched with CHPIGN<7:0> in DCHxCON<31:24.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_PING_PONG_MODE Enumeration

Lists the possible ping-pong modes.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_PING_PONG_PRIMARY,
  DMA_PING_PONG_SECONDARY
} DMA_PING_PONG_MODE;
```

Members

Members	Description
DMA_PING_PONG_PRIMARY	DMAxSTA is selected in the ping-pong mode
DMA_PING_PONG_SECONDARY	DMAxSTB is selected in the ping-pong mode

Description

DMA PING-PONG modes

This enumeration lists all the possible ping-pong modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_SOURCE_ADDRESSING_MODE Enumeration

Lists the possible source addressing modes.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_ADDRESSING_SOURCE_UNCHANGED,
  DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE,
```

```
  DMA_ADDRESSING_SOURCE_DECREMENT_BASED_ON_SIZE,
  DMA_ADDRESSING_SOURCE_PERIPHERAL_INDIRECT
} DMA_SOURCE_ADDRESSING_MODE;
```

Members

Members	Description
DMA_ADDRESSING_SOURCE_UNCHANGED	DMASRC remains unchanged after a transfer completion
DMA_ADDRESSING_SOURCE_INCREMENT_BASED_ON_SIZE	DMASRC is incremented based on SIZE bit after a transfer completion
DMA_ADDRESSING_SOURCE_DECREMENT_BASED_ON_SIZE	DMASRC is decremented based on SIZE bit after a transfer completion
DMA_ADDRESSING_SOURCE_PERIPHERAL_INDIRECT	DMASRC is used in peripheral indirect addressing and remains unchanged

Description

DMA source addressing modes

This enumeration lists all the possible source addressing modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_TRANSFER_MODE Enumeration

Lists the possible DMA transfer/operating modes.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_MODE_ONE_SHOT,
  DMA_MODE_REPEATED_ONE_SHOT,
  DMA_MODE_CONTINUOUS,
  DMA_MODE_REPEATED_CONTINUOUS,
  DMA_MODE_ONE_SHOT_PING_PONG_ENABLED,
  DMA_MODE_ONE_SHOT_PING_PONG_DISABLED,
  DMA_MODE_CONTINUOUS_PING_PONG_DISABLED,
  DMA_MODE_CONTINUOUS_PING_PONG_ENABLED
} DMA_TRANSFER_MODE;
```

Members

Members	Description
DMA_MODE_ONE_SHOT	one-shot transfer
DMA_MODE_REPEATED_ONE_SHOT	repeated one-shot transfer
DMA_MODE_CONTINUOUS	continuous transfer
DMA_MODE_REPEATED_CONTINUOUS	repeated continuous transfer
DMA_MODE_ONE_SHOT_PING_PONG_ENABLED	one-shot transfer with ping-pong buffers
DMA_MODE_ONE_SHOT_PING_PONG_DISABLED	one-shot transfer without the ping-pong buffers
DMA_MODE_CONTINUOUS_PING_PONG_DISABLED	continuous transfer without the ping-pong buffers
DMA_MODE_CONTINUOUS_PING_PONG_ENABLED	continuous transfer with ping-pong buffers

Description

DMA transfer/operating mode

This enumeration lists the possible DMA transfer/operating modes.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

DMA_TRIGGER_SOURCE Enumeration

Lists the possible DMA trigger sources.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_TRIGGER_TIMER_CORE,
  DMA_TRIGGER_SOFTWARE_0,
  DMA_TRIGGER_SOFTWARE_1,
  DMA_TRIGGER_EXTERNAL_0,
  DMA_TRIGGER_EXTERNAL_1,
  DMA_TRIGGER_EXTERNAL_2,
  DMA_TRIGGER_EXTERNAL_3,
  DMA_TRIGGER_EXTERNAL_4,
  DMA_TRIGGER_TIMER_1,
  DMA_TRIGGER_TIMER_2,
  DMA_TRIGGER_TIMER_3,
  DMA_TRIGGER_TIMER_4,
  DMA_TRIGGER_TIMER_5,
  DMA_TRIGGER_INPUT_CAPTURE_1,
  DMA_TRIGGER_INPUT_CAPTURE_2,
  DMA_TRIGGER_INPUT_CAPTURE_3,
  DMA_TRIGGER_INPUT_CAPTURE_4,
  DMA_TRIGGER_INPUT_CAPTURE_5,
  DMA_TRIGGER_INPUT_CAPTURE_1_ERROR,
  DMA_TRIGGER_INPUT_CAPTURE_2_ERROR,
  DMA_TRIGGER_INPUT_CAPTURE_3_ERROR,
  DMA_TRIGGER_INPUT_CAPTURE_4_ERROR,
  DMA_TRIGGER_INPUT_CAPTURE_5_ERROR,
  DMA_TRIGGER_OUTPUT_COMPARE_1,
  DMA_TRIGGER_OUTPUT_COMPARE_2,
  DMA_TRIGGER_OUTPUT_COMPARE_3,
  DMA_TRIGGER_OUTPUT_COMPARE_4,
  DMA_TRIGGER_OUTPUT_COMPARE_5,
  DMA_TRIGGER_SPI_1_ERROR,
  DMA_TRIGGER_SPI_1_RECEIVE,
  DMA_TRIGGER_SPI_1_TRANSMIT,
  DMA_TRIGGER_SPI_1A_ERROR,
  DMA_TRIGGER_SPI_1A_RECEIVE,
  DMA_TRIGGER_SPI_1A_TRANSMIT,
  DMA_TRIGGER_SPI_2_ERROR,
  DMA_TRIGGER_SPI_2_RECEIVE,
  DMA_TRIGGER_SPI_2_TRANSMIT,
  DMA_TRIGGER_SPI_2A_ERROR,
  DMA_TRIGGER_SPI_2A_RECEIVE,
  DMA_TRIGGER_SPI_2A_TRANSMIT,
  DMA_TRIGGER_SPI_3_ERROR,
  DMA_TRIGGER_SPI_3_RECEIVE,
  DMA_TRIGGER_SPI_3_TRANSMIT,
  DMA_TRIGGER_SPI_3A_ERROR,
  DMA_TRIGGER_SPI_3A_RECEIVE,
  DMA_TRIGGER_SPI_3A_TRANSMIT,
  DMA_TRIGGER_SPI_4_ERROR,
  DMA_TRIGGER_SPI_4_RECEIVE,
  DMA_TRIGGER_SPI_4_TRANSMIT,
  DMA_TRIGGER_I2C_1_ERROR,
  DMA_TRIGGER_I2C_1_SLAVE,
  DMA_TRIGGER_I2C_1_MASTER,
  DMA_TRIGGER_I2C_1A_ERROR,
  DMA_TRIGGER_I2C_1A_SLAVE,
  DMA_TRIGGER_I2C_1A_MASTER,
  DMA_TRIGGER_I2C_2_ERROR,
  DMA_TRIGGER_I2C_2_SLAVE,
  DMA_TRIGGER_I2C_2_MASTER,
  DMA_TRIGGER_I2C_2A_ERROR,
  DMA_TRIGGER_I2C_2A_SLAVE,
  DMA_TRIGGER_I2C_2A_MASTER,
  DMA_TRIGGER_I2C_3_ERROR,
  DMA_TRIGGER_I2C_3_SLAVE,
  DMA_TRIGGER_I2C_3_MASTER,
  DMA_TRIGGER_I2C_3A_ERROR,
  DMA_TRIGGER_I2C_3A_SLAVE,
  DMA_TRIGGER_I2C_3A_MASTER,
  DMA_TRIGGER_I2C_4_ERROR,
  DMA_TRIGGER_I2C_4_SLAVE,
  DMA_TRIGGER_I2C_4_MASTER,
  DMA_TRIGGER_I2C_5_ERROR,
  DMA_TRIGGER_I2C_5_SLAVE,
```

```
    DMA_TRIGGER_I2C_5_MASTER,
    DMA_TRIGGER_USART_1_ERROR,
    DMA_TRIGGER_USART_1_RECEIVE,
    DMA_TRIGGER_USART_1_TRANSMIT,
    DMA_TRIGGER_USART_1A_ERROR,
    DMA_TRIGGER_USART_1A_RECEIVE,
    DMA_TRIGGER_USART_1A_TRANSMIT,
    DMA_TRIGGER_USART_1B_ERROR,
    DMA_TRIGGER_USART_1B_RECEIVE,
    DMA_TRIGGER_USART_1B_TRANSMIT,
    DMA_TRIGGER_USART_2_ERROR,
    DMA_TRIGGER_USART_2_RECEIVE,
    DMA_TRIGGER_USART_2_TRANSMIT,
    DMA_TRIGGER_USART_2A_ERROR,
    DMA_TRIGGER_USART_2A_RECEIVE,
    DMA_TRIGGER_USART_2A_TRANSMIT,
    DMA_TRIGGER_USART_2B_ERROR,
    DMA_TRIGGER_USART_2B_RECEIVE,
    DMA_TRIGGER_USART_2B_TRANSMIT,
    DMA_TRIGGER_USART_3_ERROR,
    DMA_TRIGGER_USART_3_RECEIVE,
    DMA_TRIGGER_USART_3_TRANSMIT,
    DMA_TRIGGER_USART_3A_ERROR,
    DMA_TRIGGER_USART_3A_RECEIVE,
    DMA_TRIGGER_USART_3A_TRANSMIT,
    DMA_TRIGGER_USART_3B_ERROR,
    DMA_TRIGGER_USART_3B_RECEIVE,
    DMA_TRIGGER_USART_3B_TRANSMIT,
    DMA_TRIGGER_USART_4_ERROR,
    DMA_TRIGGER_USART_4_RECEIVE,
    DMA_TRIGGER_USART_4_TRANSMIT,
    DMA_TRIGGER_USART_5_ERROR,
    DMA_TRIGGER_USART_5_RECEIVE,
    DMA_TRIGGER_USART_5_TRANSMIT,
    DMA_TRIGGER_USART_6_ERROR,
    DMA_TRIGGER_USART_6_RECEIVE,
    DMA_TRIGGER_USART_6_TRANSMIT,
    DMA_TRIGGER_CHANGE_NOTICE,
    DMA_TRIGGER_CHANGE_NOTICE_A,
    DMA_TRIGGER_CHANGE_NOTICE_B,
    DMA_TRIGGER_CHANGE_NOTICE_C,
    DMA_TRIGGER_CHANGE_NOTICE_D,
    DMA_TRIGGER_CHANGE_NOTICE_E,
    DMA_TRIGGER_CHANGE_NOTICE_F,
    DMA_TRIGGER_CHANGE_NOTICE_G,
    DMA_TRIGGER_DMA_0,
    DMA_TRIGGER_DMA_1,
    DMA_TRIGGER_DMA_2,
    DMA_TRIGGER_DMA_3,
    DMA_TRIGGER_DMA_4,
    DMA_TRIGGER_DMA_5,
    DMA_TRIGGER_DMA_6,
    DMA_TRIGGER_DMA_7,
    DMA_TRIGGER_COMPARATOR_1,
    DMA_TRIGGER_COMPARATOR_2,
    DMA_TRIGGER_COMPARATOR_3,
    DMA_TRIGGER_ADC_1,
    DMA_TRIGGER_PARALLEL_PORT,
    DMA_TRIGGER_CAN_1,
    DMA_TRIGGER_CAN_2,
    DMA_TRIGGER_CLOCK_MONITOR,
    DMA_TRIGGER_RTCC,
    DMA_TRIGGER_FLASH_CONTROL,
    DMA_TRIGGER_USB_1,
    DMA_TRIGGER_ETH_1,
    DMA_TRIGGER_PARALLEL_PORT_ERROR,
    DMA_TRIGGER_CTMU
} DMA_TRIGGER_SOURCE;
```

Members

Members	Description
DMA_TRIGGER_TIMER_CORE	Core timer
DMA_TRIGGER_SOFTWARE_0	Software 0

DMA_TRIGGER_SOFTWARE_1	Software 1
DMA_TRIGGER_EXTERNAL_0	External 0
DMA_TRIGGER_EXTERNAL_1	External 1
DMA_TRIGGER_EXTERNAL_2	External 2
DMA_TRIGGER_EXTERNAL_3	External 3
DMA_TRIGGER_EXTERNAL_4	External 4
DMA_TRIGGER_TIMER_1	Timer1
DMA_TRIGGER_TIMER_2	Timer2
DMA_TRIGGER_TIMER_3	Timer3
DMA_TRIGGER_TIMER_4	Timer4
DMA_TRIGGER_TIMER_5	Timer5
DMA_TRIGGER_INPUT_CAPTURE_1	Input Capture 1
DMA_TRIGGER_INPUT_CAPTURE_2	Input Capture 2
DMA_TRIGGER_INPUT_CAPTURE_3	Input Capture 3
DMA_TRIGGER_INPUT_CAPTURE_4	Input Capture 4
DMA_TRIGGER_INPUT_CAPTURE_5	Input Capture 5
DMA_TRIGGER_INPUT_CAPTURE_1_ERROR	Input Capture 1 error
DMA_TRIGGER_INPUT_CAPTURE_2_ERROR	Input Capture 2 error
DMA_TRIGGER_INPUT_CAPTURE_3_ERROR	Input Capture 3 error
DMA_TRIGGER_INPUT_CAPTURE_4_ERROR	Input Capture 4 error
DMA_TRIGGER_INPUT_CAPTURE_5_ERROR	Input Capture 5 error
DMA_TRIGGER_OUTPUT_COMPARE_1	Output Compare 1
DMA_TRIGGER_OUTPUT_COMPARE_2	Output Compare 2
DMA_TRIGGER_OUTPUT_COMPARE_3	Output Compare 3
DMA_TRIGGER_OUTPUT_COMPARE_4	Output Compare 4
DMA_TRIGGER_OUTPUT_COMPARE_5	Output Compare 5
DMA_TRIGGER_SPI_1_ERROR	SPI1 error
DMA_TRIGGER_SPI_1_RECEIVE	SPI1 receive
DMA_TRIGGER_SPI_1_TRANSMIT	SPI1 transmit
DMA_TRIGGER_SPI_1A_ERROR	SPI1A error
DMA_TRIGGER_SPI_1A_RECEIVE	SPI1A receive
DMA_TRIGGER_SPI_1A_TRANSMIT	SPI1A transmit
DMA_TRIGGER_SPI_2_ERROR	SPI2 error
DMA_TRIGGER_SPI_2_RECEIVE	SPI2 receive
DMA_TRIGGER_SPI_2_TRANSMIT	SPI2 transmit
DMA_TRIGGER_SPI_2A_ERROR	SPI2A receive
DMA_TRIGGER_SPI_2A_RECEIVE	SPI2A receive
DMA_TRIGGER_SPI_2A_TRANSMIT	SPI2A transmit
DMA_TRIGGER_SPI_3_ERROR	SPI3 error
DMA_TRIGGER_SPI_3_RECEIVE	SPI3 receive
DMA_TRIGGER_SPI_3_TRANSMIT	SPI3 transmit
DMA_TRIGGER_SPI_3A_ERROR	SPI3A receive
DMA_TRIGGER_SPI_3A_RECEIVE	SPI3A receive
DMA_TRIGGER_SPI_3A_TRANSMIT	SPI3A receive
DMA_TRIGGER_SPI_4_ERROR	SPI4 error
DMA_TRIGGER_SPI_4_RECEIVE	SPI4 receive
DMA_TRIGGER_SPI_4_TRANSMIT	SPI4 transmit
DMA_TRIGGER_I2C_1_ERROR	I2C1 error
DMA_TRIGGER_I2C_1_SLAVE	I2C1 slave
DMA_TRIGGER_I2C_1_MASTER	I2C1 master
DMA_TRIGGER_I2C_1A_ERROR	I2C1A error
DMA_TRIGGER_I2C_1A_SLAVE	I2C1A slave
DMA_TRIGGER_I2C_1A_MASTER	I2C1A master
DMA_TRIGGER_I2C_2_ERROR	I2C2 error
DMA_TRIGGER_I2C_2_SLAVE	I2C2 slave

DMA_TRIGGER_I2C_2_MASTER	I2C2 master
DMA_TRIGGER_I2C_2A_ERROR	I2C2A error
DMA_TRIGGER_I2C_2A_SLAVE	I2C2A slave
DMA_TRIGGER_I2C_2A_MASTER	I2C2A master
DMA_TRIGGER_I2C_3_ERROR	I2C3 error
DMA_TRIGGER_I2C_3_SLAVE	I2C3 slave
DMA_TRIGGER_I2C_3_MASTER	I2C3 master
DMA_TRIGGER_I2C_3A_ERROR	I2C3A error
DMA_TRIGGER_I2C_3A_SLAVE	I2C3A slave
DMA_TRIGGER_I2C_3A_MASTER	I2C3A master
DMA_TRIGGER_I2C_4_ERROR	I2C4 error
DMA_TRIGGER_I2C_4_SLAVE	I2C4 slave
DMA_TRIGGER_I2C_4_MASTER	I2C4 master
DMA_TRIGGER_I2C_5_ERROR	I2C5 error
DMA_TRIGGER_I2C_5_SLAVE	I2C5 slave
DMA_TRIGGER_I2C_5_MASTER	I2C5 master
DMA_TRIGGER_USART_1_ERROR	USART1 error
DMA_TRIGGER_USART_1_RECEIVE	USART1 receive
DMA_TRIGGER_USART_1_TRANSMIT	USART1 transmit
DMA_TRIGGER_USART_1A_ERROR	USART1A error
DMA_TRIGGER_USART_1A_RECEIVE	USART1A receive
DMA_TRIGGER_USART_1A_TRANSMIT	USART1A transmit
DMA_TRIGGER_USART_1B_ERROR	USART1B error
DMA_TRIGGER_USART_1B_RECEIVE	USART1B receive
DMA_TRIGGER_USART_1B_TRANSMIT	USART1B transmit
DMA_TRIGGER_USART_2_ERROR	USART2 error
DMA_TRIGGER_USART_2_RECEIVE	USART2 receive
DMA_TRIGGER_USART_2_TRANSMIT	USART2 transmit
DMA_TRIGGER_USART_2A_ERROR	USART2A error
DMA_TRIGGER_USART_2A_RECEIVE	USART2A receive
DMA_TRIGGER_USART_2A_TRANSMIT	USART2A transmit
DMA_TRIGGER_USART_2B_ERROR	USART2B error
DMA_TRIGGER_USART_2B_RECEIVE	USART2B receive
DMA_TRIGGER_USART_2B_TRANSMIT	USART2B transmit
DMA_TRIGGER_USART_3_ERROR	USART3 error
DMA_TRIGGER_USART_3_RECEIVE	USART3 receive
DMA_TRIGGER_USART_3_TRANSMIT	USART3 transmit
DMA_TRIGGER_USART_3A_ERROR	USART3A error
DMA_TRIGGER_USART_3A_RECEIVE	USART3A receive
DMA_TRIGGER_USART_3A_TRANSMIT	USART3A transmit
DMA_TRIGGER_USART_3B_ERROR	USART3B error
DMA_TRIGGER_USART_3B_RECEIVE	USART3B receive
DMA_TRIGGER_USART_3B_TRANSMIT	USART3B transmit
DMA_TRIGGER_USART_4_ERROR	USART4 error
DMA_TRIGGER_USART_4_RECEIVE	USART3B receive
DMA_TRIGGER_USART_4_TRANSMIT	USART4 transmit
DMA_TRIGGER_USART_5_ERROR	USART5 error
DMA_TRIGGER_USART_5_RECEIVE	USART5 receive
DMA_TRIGGER_USART_5_TRANSMIT	USART5 transmit
DMA_TRIGGER_USART_6_ERROR	USART6 error
DMA_TRIGGER_USART_6_RECEIVE	USART6 receive
DMA_TRIGGER_USART_6_TRANSMIT	USART6 transmit
DMA_TRIGGER_CHANGE_NOTICE	Change notice
DMA_TRIGGER_CHANGE_NOTICE_A	Change notice A
DMA_TRIGGER_CHANGE_NOTICE_B	Change notice B

DMA_TRIGGER_CHANGE_NOTICE_C	Change notice C
DMA_TRIGGER_CHANGE_NOTICE_D	Change notice D
DMA_TRIGGER_CHANGE_NOTICE_E	Change notice E
DMA_TRIGGER_CHANGE_NOTICE_F	Change notice F
DMA_TRIGGER_CHANGE_NOTICE_G	Change notice G
DMA_TRIGGER_DMA_0	DMA Channel 0
DMA_TRIGGER_DMA_1	DMA Channel 1
DMA_TRIGGER_DMA_2	DMA Channel 2
DMA_TRIGGER_DMA_3	DMA Channel 3
DMA_TRIGGER_DMA_4	DMA Channel 4
DMA_TRIGGER_DMA_5	DMA Channel 5
DMA_TRIGGER_DMA_6	DMA Channel 6
DMA_TRIGGER_DMA_7	DMA Channel 7
DMA_TRIGGER_COMPARATOR_1	Comparator 1
DMA_TRIGGER_COMPARATOR_2	DMA Channel 2
DMA_TRIGGER_COMPARATOR_3	DMA Channel 3
DMA_TRIGGER_ADC_1	ADC1
DMA_TRIGGER_PARALLEL_PORT	Parallel port
DMA_TRIGGER_CAN_1	CAN1
DMA_TRIGGER_CAN_2	CAN2
DMA_TRIGGER_CLOCK_MONITOR	Clock monitor
DMA_TRIGGER_RTCC	RTCC
DMA_TRIGGER_FLASH_CONTROL	Flash control
DMA_TRIGGER_USB_1	USB1
DMA_TRIGGER_ETH_1	ETH1
DMA_TRIGGER_PARALLEL_PORT_ERROR	Parallel port error
DMA_TRIGGER_CTMU	CTMU

Description

DMA trigger sources

This enumeration lists all the possible DMA trigger sources.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

DMA_CHANNEL_INT_SOURCE Enumeration

Lists the Interrupt Source number for the available DMA channels.

File

help_plib_dma.h

C

```
typedef enum {
  DMA_CHANNEL_0_INT_SOURCE,
  DMA_CHANNEL_1_INT_SOURCE,
  DMA_CHANNEL_2_INT_SOURCE,
  DMA_CHANNEL_3_INT_SOURCE,
  DMA_CHANNEL_4_INT_SOURCE,
  DMA_CHANNEL_5_INT_SOURCE,
  DMA_CHANNEL_6_INT_SOURCE,
  DMA_CHANNEL_7_INT_SOURCE
} DMA_CHANNEL_INT_SOURCE;
```

Members

Members	Description
DMA_CHANNEL_0_INT_SOURCE	DMA Channel 0 Interrupt Source
DMA_CHANNEL_1_INT_SOURCE	DMA Channel 1 Interrupt Source
DMA_CHANNEL_2_INT_SOURCE	DMA Channel 2 Interrupt Source

DMA_CHANNEL_3_INT_SOURCE	DMA Channel 3 Interrupt Source
DMA_CHANNEL_4_INT_SOURCE	DMA Channel 4 Interrupt Source
DMA_CHANNEL_5_INT_SOURCE	DMA Channel 5 Interrupt Source
DMA_CHANNEL_6_INT_SOURCE	DMA Channel 6 Interrupt Source
DMA_CHANNEL_7_INT_SOURCE	DMA Channel 7 Interrupt Source

Description

DMA channel Interrupt source number

This enumeration lists the Interrupt Source number for all of the available DMA channels.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Files

Files

Name	Description
plib_dma.h	Defines the DMA Peripheral Library interface functions.
help_plib_dma.h	Defines the DMA Peripheral Library data types.

Description

This section lists the source and header files used by the library.

plib_dma.h

Defines the DMA Peripheral Library interface functions.

Functions

	Name	Description
	PLIB_DMA_AbortTransferSet	Aborts transfer on the specified channel.
	PLIB_DMA_BusyActiveReset	Resets the BUSY bit of the DMA controller.
	PLIB_DMA_BusyActiveSet	Sets the BUSY bit of the DMA controller.
	PLIB_DMA_ChannelBitsGet	Returns the DMA channel bits.
	PLIB_DMA_ChannelPriorityGet	Gets the priority scheme of the DMA channels.
	PLIB_DMA_ChannelPrioritySelect	Sets the priority scheme of the DMA channels.
	PLIB_DMA_ChannelXAbortIRQSet	Sets the IRQ to abort the DMA transfer on the specified channel.
	PLIB_DMA_ChannelXAddressModeGet	Gets the channel address mode.
	PLIB_DMA_ChannelXAddressModeSelect	Sets the channel address mode.
	PLIB_DMA_ChannelXAutoDisable	Channel is disabled after a block transfer is complete.
	PLIB_DMA_ChannelXAutoEnable	Channel is continuously enabled.
	PLIB_DMA_ChannelXAutoIsEnabled	Returns the channel automatic enable status.
	PLIB_DMA_ChannelXBufferedDataIsWritten	Returns the buffered data write status for the specified channel.
	PLIB_DMA_ChannelXBusyActiveSet	Sets the Busy bit to active.
	PLIB_DMA_ChannelXBusyInActiveSet	Sets the Busy bit to inactive.
	PLIB_DMA_ChannelXBusyIsBusy	Returns the busy status of the specified channel.
	PLIB_DMA_ChannelXCellProgressPointerGet	Returns the number of bytes transferred since the last event.
	PLIB_DMA_ChannelXCellSizeGet	Reads the cell size (in bytes) configured for the specified channel.
	PLIB_DMA_ChannelXCellSizeSet	Writes the specified cell size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXChainDisable	Disables the channel chaining for the specified DMA channel.
	PLIB_DMA_ChannelXChainEnable	Channel chain feature is enabled.
	PLIB_DMA_ChannelXChainIsEnabled	Returns the chain status of the specified channel.
	PLIB_DMA_ChannelXChainToHigher	Chains the specified channel to a channel higher in natural priority.
	PLIB_DMA_ChannelXChainToLower	Chains the specified channel to a channel lower in natural priority.
	PLIB_DMA_ChannelXCollisionStatus	Returns the status of the specified collision type for the specified channel.

	PLIB_DMA_ChannelXDataSizeGet	Returns the current data size for the specified channel.
	PLIB_DMA_ChannelXDataSizeSelect	Selects the data size for the specified channel.
	PLIB_DMA_ChannelXDestinationAddressModeGet	Gets the source address mode of the specified channel.
	PLIB_DMA_ChannelXDestinationAddressModeSelect	Sets the source address mode of the specified channel.
	PLIB_DMA_ChannelXDestinationPointerGet	Reads the current byte of the destination being pointed to for the specified channel.
	PLIB_DMA_ChannelXDestinationSizeGet	Reads the destination size configured for the specified channel.
	PLIB_DMA_ChannelXDestinationSizeSet	Writes the specified destination size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXDestinationStartAddressGet	Reads the destination start address configured for the specified channel.
	PLIB_DMA_ChannelXDestinationStartAddressSet	Writes the specified destination start address into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXDisable	Disable the specified channel.
	PLIB_DMA_ChannelXDisabledDisablesEvents	Channel start/abort events will be ignored even if the channel is disabled.
	PLIB_DMA_ChannelXDisabledEnablesEvents	Channel start/abort events will be registered even if the channel is disabled.
	PLIB_DMA_ChannelXEnable	Enable the specified channel.
	PLIB_DMA_ChannelXEventIsDetected	Returns the event status on the specified channel.
	PLIB_DMA_ChannelXINTSourceDisable	Disables the specified interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceEnable	Enables the specified interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagClear	Clears the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagGet	Returns the status of the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceFlagSet	Sets the interrupt flag of the specified DMA interrupt source for the specified channel.
	PLIB_DMA_ChannelXINTSourceIsEnabled	Returns the enable status of the specified interrupt source for the specified channel.
	PLIB_DMA_ChannelXIsEnabled	Return the enable status of the specified channel.
	PLIB_DMA_ChannelXNullWriteModeDisable	Disables the Null Write mode.
	PLIB_DMA_ChannelXNullWriteModeEnable	Enables the Null Write mode.
	PLIB_DMA_ChannelXNullWriteModeIsEnabled	Returns the enable status of the Null Write mode for the specified channel.
	PLIB_DMA_ChannelXOperatingTransferModeGet	Returns the current transfer/operating mode for the specified channel.
	PLIB_DMA_ChannelXOperatingTransferModeSelect	Selects the transfer/operating mode for the specified channel.
	PLIB_DMA_ChannelXPatternDataGet	Returns the pattern matching (for DMA abort) data programmed for the specified channel.
	PLIB_DMA_ChannelXPatternDataSet	Writes the specified pattern matching data (for DMA abort) into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXPatternIgnoreByteDisable	Disables the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreByteEnable	Enables the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled	Returns the state of the pattern match ignore byte.
	PLIB_DMA_ChannelXPatternIgnoreGet	Returns the pattern match ignore value.
	PLIB_DMA_ChannelXPatternIgnoreSet	Sets the pattern match ignore value.
	PLIB_DMA_ChannelXPatternLengthGet	Returns the pattern match length.
	PLIB_DMA_ChannelXPatternLengthSet	Sets the pattern match length.
	PLIB_DMA_ChannelXPeripheralAddressGet	Gets the peripheral address configured for the specified channel.
	PLIB_DMA_ChannelXPeripheralAddressSet	Sets the peripheral address for the specified channel.
	PLIB_DMA_ChannelXPingPongModeGet	Returns the Ping-Pong mode status for the specified channel.
	PLIB_DMA_ChannelXPriorityGet	Gets the priority of the specified channel.
	PLIB_DMA_ChannelXPrioritySelect	Sets the priority of the specified channel.
	PLIB_DMA_ChannelXReloadDisable	Disables reloading of the address and count registers.
	PLIB_DMA_ChannelXReloadEnable	Enables reloading of the address and count registers.
	PLIB_DMA_ChannelXReloadIsEnabled	Returns the address and count registers reload enable status.
	PLIB_DMA_ChannelXSourceAddressModeGet	Gets the source address mode of the specified channel.
	PLIB_DMA_ChannelXSourceAddressModeSelect	Sets the source address mode of the specified channel.
	PLIB_DMA_ChannelXSourcePointerGet	Reads the current byte of the source being pointed to for the specified channel.
	PLIB_DMA_ChannelXSourceSizeGet	Reads the source size configured for the specified channel.

	PLIB_DMA_ChannelXSourceSizeSet	Writes the specified source size into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXSourceStartAddressGet	Reads the source start address configured for the specified channel.
	PLIB_DMA_ChannelXSourceStartAddressSet	Writes the specified source start address into the register corresponding to the specified channel.
	PLIB_DMA_ChannelXStartAddressOffsetGet	Gets the primary/secondary start address (DPSRAM) offset.
	PLIB_DMA_ChannelXStartAddressOffsetSet	Sets the primary/secondary start address (DPSRAM) offset to the value specified depending on the offset type specified for the specified channel.
	PLIB_DMA_ChannelXStartIRQSet	Sets the IRQ to initiate the DMA transfer on the specified channel.
	PLIB_DMA_ChannelXTransferCountGet	Gets the DMA data transfer count that is programmed for the specified channel.
	PLIB_DMA_ChannelXTransferCountSet	Sets the DMA data transfer count for the specified channel.
	PLIB_DMA_ChannelXTransferDirectionGet	Returns the data transfer direction of the specified channel.
	PLIB_DMA_ChannelXTransferDirectionSelect	Selects the data transfer direction of the specified channel.
	PLIB_DMA_ChannelXTriggerDisable	Disables the DMA transfer abort via a matching interrupt (specified by the IRQ).
	PLIB_DMA_ChannelXTriggerEnable	Enables the specified DMA channel trigger.
	PLIB_DMA_ChannelXTriggerIsEnabled	Returns the enable status of the specified DMA transfer/abort trigger.
	PLIB_DMA_ChannelXTriggerSourceNumberGet	Gets the source number for the DMA channel interrupts.
	PLIB_DMA_CRCAppendModeDisable	Disables the CRC append mode.
	PLIB_DMA_CRCAppendModeEnable	Enables the CRC append mode.
	PLIB_DMA_CRCAppendModeIsEnabled	Gets the enable status of the CRC append mode.
	PLIB_DMA_CRCBitOrderSelect	Selects the bit order for checksum calculation.
	PLIB_DMA_CRCByteOrderGet	Gets the current byte order selected by the DMA module CRC feature.
	PLIB_DMA_CRCByteOrderSelect	Selects the byte order.
	PLIB_DMA_CRCChannelGet	Returns the current DMA channel to which the CRC is assigned.
	PLIB_DMA_CRCChannelSelect	Assigns the CRC to the specified DMA channel.
	PLIB_DMA_CRCDataRead	Reads the contents of the DMA CRC data register.
	PLIB_DMA_CRCDataWrite	Writes the contents of the DMA CRC data register with the specified data.
	PLIB_DMA_CRCDisable	Disables the DMA module CRC feature.
	PLIB_DMA_CRCEnable	Enables the DMA module CRC feature.
	PLIB_DMA_CRCIsEnabled	Gets the enable status of the CRC feature.
	PLIB_DMA_CRCPolynomialLengthGet	Gets the current polynomial length.
	PLIB_DMA_CRCPolynomialLengthSet	Selects the polynomial length.
	PLIB_DMA_CRCTypeGet	Gets the current DMA module CRC feature type.
	PLIB_DMA_CRCTypeSet	Selects the DMA module CRC feature type.
	PLIB_DMA_CRCWriteByteOrderAlter	The source data is written to the destination reordered as defined by the BYTO<1:0> bits.
	PLIB_DMA_CRCWriteByteOrderMaintain	The source data is written to the destination unaltered.
	PLIB_DMA_CRCXOREnableGet	Reads the CRC XOR register.
	PLIB_DMA_CRCXOREnableSet	Writes to the CRC XOR enable register as per the specified enable mask.
	PLIB_DMA_Disable	DMA module is disabled.
	PLIB_DMA_Enable	DMA module is enabled.
	PLIB_DMA_ExistsAbortTransfer	Identifies whether the AbortTransfer feature exists on the DMA module.
	PLIB_DMA_ExistsBusy	Identifies whether the Busy feature exists on the DMA module.
	PLIB_DMA_ExistsChannelBits	Identifies whether the ChannelBits feature exists on the DMA module.
	PLIB_DMA_ExistsChannelX	Identifies whether the ChannelX feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXAbortIRQ	Identifies whether the ChannelXAbortIRQ feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXAuto	Identifies whether the ChannelXAuto feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXBusy	Identifies whether the ChannelXBusy feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXCellProgressPointer	Identifies whether the ChannelXCellProgressPointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXCellSize	Identifies whether the ChannelXCellSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXChain	Identifies whether the ChannelXChain feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXChainEnbl	Identifies whether the ChannelXChainEnbl feature exists on the DMA module.

	PLIB_DMA_ExistsChannelXDestinationPointer	Identifies whether the ChannelXDestinationPointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationSize	Identifies whether the ChannelXDestinationSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDestinationStartAddress	Identifies whether the ChannelXDestinationStartAddress feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXDisabled	Identifies whether the ChannelXDisabled feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXEvent	Identifies whether the ChannelXEvent feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXINTSource	Identifies whether the ChannelXINTSource feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXINTSourceFlag	Identifies whether the ChannelXINTSourceFlag feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternData	Identifies whether the ChannelXPatternData feature exists on the DMA module
	PLIB_DMA_ExistsChannelXPatternIgnore	Identifies whether the ChannelXPatternIgnore feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternIgnoreByte	Identifies whether the ChannelXPatternIgnoreByte feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPatternLength	Identifies whether the ChannelXPatternLength feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXPriority	Identifies whether the ChannelXPriority feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourcePointer	Identifies whether the ChannelXSourcePointer feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourceSize	Identifies whether the ChannelXSourceSize feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXSourceStartAddress	Identifies whether the ChannelXSourceStartAddress feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXStartIRQ	Identifies whether the ChannelXStartIRQ feature exists on the DMA module.
	PLIB_DMA_ExistsChannelXTrigger	Identifies whether the ChannelXTrigger feature exists on the DMA module.
	PLIB_DMA_ExistsCRC	Identifies whether the CRC feature exists on the DMA module.
	PLIB_DMA_ExistsCRCAppendMode	Identifies whether the CRCAppendMode feature exists on the DMA module.
	PLIB_DMA_ExistsCRCBitOrder	Identifies whether the CRCBitOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCByteOrder	Identifies whether the CRCByteOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCChannel	Identifies whether the CRCChannel feature exists on the DMA module.
	PLIB_DMA_ExistsCRCData	Identifies whether the CRCData feature exists on the DMA module.
	PLIB_DMA_ExistsCRCPolynomialLength	Identifies whether the CRCPolynomialLength feature exists on the DMA module.
	PLIB_DMA_ExistsCRCType	Identifies whether the CRCType feature exists on the DMA module.
	PLIB_DMA_ExistsCRCWriteByteOrder	Identifies whether the CRCWriteByteOrder feature exists on the DMA module.
	PLIB_DMA_ExistsCRCXOREnable	Identifies whether the CRCXOREnable feature exists on the DMA module.
	PLIB_DMA_ExistsEnableControl	Identifies whether the EnableControl feature exists on the DMA module.
	PLIB_DMA_ExistsLastBusAccess	Identifies whether the LastBusAccess feature exists on the DMA module.
	PLIB_DMA_ExistsRecentAddress	Identifies whether the RecentAddress feature exists on the DMA module.
	PLIB_DMA_ExistsStartTransfer	Identifies whether the StartTransfer feature exists on the DMA module.
	PLIB_DMA_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the DMA module.
	PLIB_DMA_ExistsSuspend	Identifies whether the Suspend feature exists on the DMA module.
	PLIB_DMA_IsBusy	Gets the BUSY bit of the DMA controller.
	PLIB_DMA_IsEnabled	Returns the DMA module enable status.
	PLIB_DMA_LastBusAccessIsRead	Returns true if the last DMA bus access was a read.
	PLIB_DMA_LastBusAccessIsWrite	Returns true if the last DMA bus access was a write.
	PLIB_DMA_RecentAddressAccessed	Returns the address of the most recent DMA access.
	PLIB_DMA_StartTransferSet	Initiates transfer on the specified channel.
	PLIB_DMA_StopInIdleDisable	DMA transfers continue during Idle mode.
	PLIB_DMA_StopInIdleEnable	DMA transfers are halted during Idle mode.
	PLIB_DMA_SuspendDisable	DMA suspend is disabled and the DMA module operates normally.
	PLIB_DMA_SuspendEnable	DMA transfers are suspended to allow uninterrupted access by the CPU to the data bus.
	PLIB_DMA_SuspendIsEnabled	Returns the DMA suspend status.

Description

DMA Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Direct Memory Access (DMA) Peripheral Library for Microchip microcontrollers. The definitions in this file are for the DMA module.

File Name

plib_dma.h

Company

Microchip Technology Inc.

help_plib_dma.h

Defines the DMA Peripheral Library data types.

Enumerations

	Name	Description
	DMA_ADDRESS_OFFSET_TYPE	Lists the possible DMA address offsets.
	DMA_CHANNEL	Lists the possible DMA channels available.
	DMA_CHANNEL_ADDRESSING_MODE	Lists the possible channel addressing modes.
	DMA_CHANNEL_COLLISION	Lists the possible channel collision types.
	DMA_CHANNEL_DATA_SIZE	Lists the possible data sizes for the DMA channel.
	DMA_CHANNEL_INT_SOURCE	Lists the Interrupt Source number for the available DMA channels.
	DMA_CHANNEL_PRIORITY	Lists the possible channel priorities.
	DMA_CHANNEL_TRANSFER_DIRECTION	Lists the possible data transfer directions.
	DMA_CHANNEL_TRIGGER_TYPE	Lists the possible DMA channel triggers.
	DMA_CRC_BIT_ORDER	Lists the possible CRC calculation bit orders
	DMA_CRC_BYTE_ORDER	Lists the possible byte orders.
	DMA_CRC_TYPE	Lists the possible checksums.
	DMA_DESTINATION_ADDRESSING_MODE	Lists the possible destination addressing modes.
	DMA_INT_TYPE	Lists the possible Interrupt types for a particular channel-X
	DMA_MODULE_ID	Possible instances of the DMA module.
	DMA_PATTERN_LENGTH	Gives the DMA pattern length
	DMA_PING_PONG_MODE	Lists the possible ping-pong modes.
	DMA_SOURCE_ADDRESSING_MODE	Lists the possible source addressing modes.
	DMA_TRANSFER_MODE	Lists the possible DMA transfer/operating modes.
	DMA_TRIGGER_SOURCE	Lists the possible DMA trigger sources.

Description

DMA Peripheral Library data types.

This header file contains data types and constants that make up the Interface to the direct memory access(DMA) peripheral library (PLIB) for Microchip microcontrollers. The definitions in this file are for DMA module.

File Name

help_plib_dma.h

Company

Microchip Technology Inc.

Dual Data Rate (DDR) SDRAM Peripheral Library

This topic describes the Dual Data Rate (DDR) SDRAM Peripheral Library.

Introduction

This library provides a low-level abstraction of the DDR SDRAM Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by abstracting differences from one microcontroller variant to another.

Description

The DDR SDRAM controller and associated physical interface (PHY) provide access to external SDRAM from the CPU and other targets within the device. The specific targets may vary between devices, but typically include graphics controller(s), DMA, and other peripherals. External DRAM expands the volatile memory available for the device, and is useful for operating systems, graphics controllers/accelerators, and other operations requiring large amounts of memory.

The SDRAM and controller must be initialized before the SDRAM can be accessed. The initialization is specific to the timing constraints and addressing of the particular SDRAM in use, as well as the clocks to the controller and PHY. This library provides functions for initializing the controller, PHY, and external SDRAM.

Using the Library

This topic describes the basic architecture of the DDR SDRAM Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_ddr.h`

The interface to the DDR SDRAM Peripheral Library is defined in the `plib_ddr.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the DDR SDRAM Peripheral Library must include `peripheral.h`.

Library File:

The DDR SDRAM Peripheral Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the Understanding MPLAB Harmony section for how the library interacts with the framework.

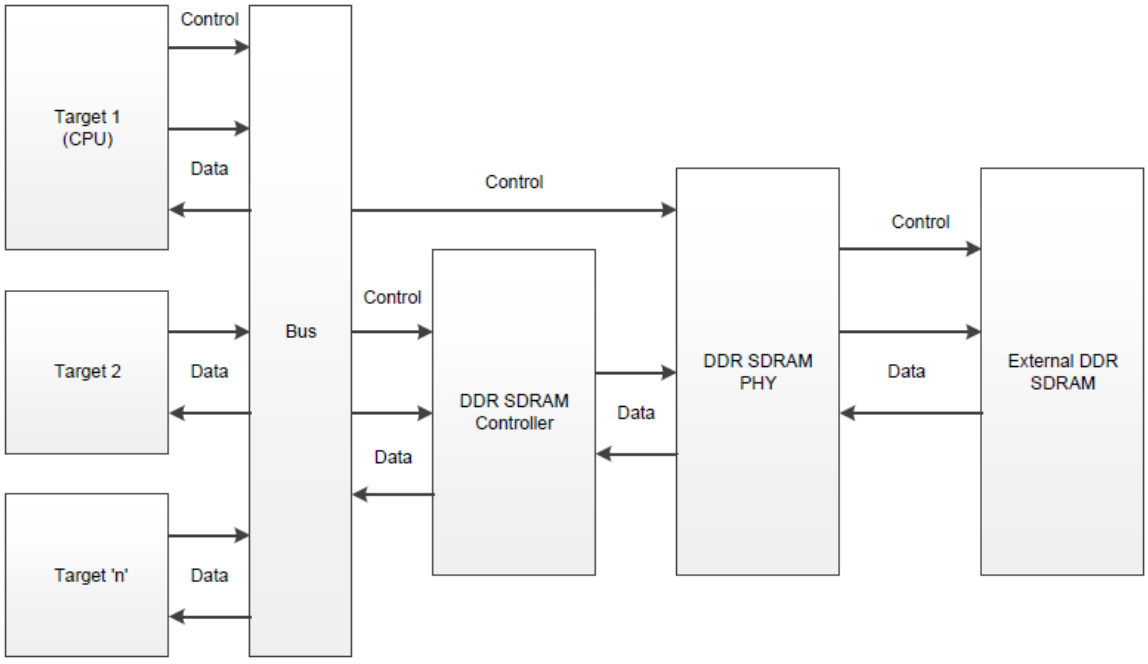
Abstraction Model

This library provides the low-level abstraction of the Dual Data Rate (DDR) SDRAM module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

The DDR SDRAM controller and PHY must be initialized prior to accessing SDRAM. The SDRAM itself must also be initialized. This library provides interface routines to perform these initializations.

DDR SDRAM Peripheral Library Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various subsections, each of the sub-sections addresses one of the blocks or the overall operation of the DDR SDRAM module.

Library Interface Section	Description
General Configuration Functions	The SDRAM controller must be configured for the system and device in use. General configuration options include DRAM type, operation mode and On-Die-Termination (ODT) settings.
Target Arbitration Functions	Arbitration parameters for each target can be programmed individually. The arbitration parameters determine the relative bandwidth assigned to each target. Adjusting target arbitration parameters can improve overall system performance by maximizing bandwidth utilization among targets.
SDRAM Addressing Functions	DDR SDRAM is addressed in terms of banks, rows and columns. The CPU address space is translated to SDRAM addresses by the controller. The addressing parameters are determined by the size and geometry of the SDRAM, and must be initialized in the controller to access the SDRAM correctly.
SDRAM Timing Functions	To operate correctly, timing delays must be inserted for various SDRAM operations. The timing delays are device-specific, and are specified in the SDRAM data sheet. These delays are used to initialize the controller for correct operation.
SDRAM Initialization Functions	The SDRAM must be initialized prior to use. Initialization consists of writing system-specific values to registers within the SDRAM itself, using the Host Command Interface. The registers and initialization sequence are standardized across SDRAM devices.
PHY Initialization Functions	Like the SDRAM controller, the PHY must be initialized prior to use. PHY parameters include ODT and drive-strength settings, DLL settings, and Self-Calibration-Logic (SCL) settings.
Feature Existence Functions	Interface routines that can be used to determine whether or not the feature is supported by the device.

How the Library Works

This section provides information on how the DDR SDRAM Peripheral Library works.

Description

General Configuration

General configuration of the SDRAM controller includes the following:

- On-Die-Termination (ODT) settings for data reads and writes
- Endianness
- DDR type (DDR2 or DDR3)
- Rate mode (full or half)

Target Arbitration

Initializing the arbitration parameters for each target is a two-step procedure. First, the minimum number of uninterrupted bursts (without interference from another target) is programmed, using the PLIB_DDR_MinLimit function. Then the number of bursts within a specific time-out period is set using the PLIB_DDR_ReqPeriod and PLIB_DDR_MinCommand functions. The relative time-outs and number of bursts between targets determine the total bandwidth allocation.

For details, refer to the **"DDR SDRAM"** chapter in the specific device data sheet and **Section 55. "DDR2 SDRAM Controller"** (DS60001321) in the *"PIC32 Family Reference Manual"*.

SDRAM Addressing

The address used by the CPU to access the SDRAM must be converted into a format understood by the SDRAM. The SDRAM uses bank, row, column and chip select bits to access data within the memory. The SDRAM addressing functions use shift and mask operations to translate the CPU address into bank, row, column and chip-select. There are four addressing functions, one each for bank, row, column and chip select. Each function takes mask and shift arguments. The mask parameter determines the number of bits that make up the address component, and the shift parameter determines where in the 32-bit CPU address the component is located. Refer to the DDR SDRAM Family Reference Manual for more details and examples.

PHY Initialization

The PHY controls the physical interface from the PIC32 to SDRAM. ODT is provided by the pads, and can be enabled or disabled using the PHY initialization functions. Setting ODT correctly for the platform can improve signal integrity by minimizing signal reflection. Enabling/disabling ODT and setting the impedance is done using the following functions:

- PLIB_DDR_PHY_OdtEnable
- PLIB_DDR_PHY_OdtDisable

Similarly, the pad drive strength can be set using the PLIB_DDR_PHY_DriveStrengthSet function.

Fine tuning of the ODT impedance and drive strength can be performed using the calibration functions PLIB_DDR_PHY_OdtCal and PLIB_DDR_PHY_DrvStrgthCal.

The SDRAM PHY contains Self-Calibration-Logic (SCL) that is run automatically during initialization. Various SCL parameters can be set using the following functions:

- PLIB_DDR_PHY_SCLEnable - enables SCL for a specific Chip Select
- PLIB_DDR_PHY_ReadCASLatencySet - sets the read CAS latency during SCL
- PLIB_DDR_PHY_WriteCASLatencySet - sets the write CAS latency during SCL
- PLIB_DDR_PHY_OdtCSEnable - enables ODT on Chip Select during SCL
- PLIB_DDR_PHY_SCLDelay - sets the SCL delay

SDRAM Timing

Several timing parameters must be initialized for the SDRAM interface to operate correctly. These include:

- Refresh interval and delay
- Read and write delays
- Precharge delays
- Addressing delays
- Bank activate delays

Timing parameters for specific SDRAMs are provided in the SDRAM data sheet. These are passed to the SDRAM timing initialization functions along with the controller clock period. The initialization functions then calculate the delays and set the controller timing registers to the correct values. Refer to **Section 55. "DDR2 SDRAM Controller"** (DS60001321)in the *"PIC32 Family Reference Manual"* for details.

SDRAM Initialization

The SDRAM is initialized by writing a series of commands to internal registers of the SDRAM itself. The register command interface is encoded on the control and address lines to the SDRAM. The SDRAM controller includes a series of registers that are initialized and transmitted to the SDRAM using the SDRAM initialization functions.

The sequence is as follows:

1. Write the initialization words to the host command registers using the PLIB_DDR_CmdDataWrite function.
2. Set the number of commands with the PLIB_DDR_NumHostCmdsSet function.
3. Write the commands to the SDRAM using PLIB_DDR_CmdDataSend.
4. Wait for the SDRAM to acknowledge valid initialization using PLIB_DDR_CmdDataIsComplete.
5. Enable the SDRAM for normal operation with the PLIB_DDR_ControllerEnable function.

The initialization command sequence is specified in the data sheet of the SDRAM device. An example initialization sequence is provided in **Section 55. "DDR2 SDRAM Controller"** (DS60001321) of the *"PIC32 Family Reference Manual"*.

Configuring the Library

The library is configured for the supported processor when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_DDR_BigEndianSet	Sets the DDR data endianness to big.
	PLIB_DDR_FullRateSet	Sets the DDR controller to Full-rate mode.
	PLIB_DDR_HalfRateSet	Sets the DDR controller to Half-rate mode.
	PLIB_DDR_LittleEndianSet	Sets the DDR data endianness to little.
	PLIB_DDR_MaxPendingRefSet	Initializes the DDR controller refresh configuration.
	PLIB_DDR_OdtReadDisable	Selects which Chip Select to disable ODT for data reads.
	PLIB_DDR_OdtReadEnable	Selects which Chip Select to enable ODT for data reads.
	PLIB_DDR_OdtWriteDisable	Selects which Chip Select to disable ODT for data writes.
	PLIB_DDR_OdtWriteEnable	Selects which Chip Select to enable ODT for data writes.
	PLIB_DDR_AutoPchrgDisable	Prevents the DDR controller from issuing an auto-precharge command to close the bank at the end of every user command.
	PLIB_DDR_AutoPowerDownDisable	Prevents the DDR controller from automatically entering Power-down mode.
	PLIB_DDR_AutoPchrgPowerDownDisable	Prevents the DDR controller from automatically entering Precharge Power-down mode.
	PLIB_DDR_AutoSelfRefreshDisable	Prevents the DDR controller from automatically entering Self-refresh mode.
	PLIB_DDR_AutoPchrgPowerDownEnable	Enables the DDR controller to automatically enter Precharge Power-down mode.

	PLIB_DDR_AutoPowerDownEnable	Enables the DDR controller to automatically enter Power-down mode.
	PLIB_DDR_AutoSelfRefreshEnable	Enables the DDR controller to automatically enter Self-refresh mode.
	PLIB_DDR_AutoPchrgEnable	Enables the DDR controller to issue an auto-precharge command to close the bank at the end of every user command.
	PLIB_DDR_TfawDelaySet	Initializes the DDR controller with the four-bank activation window needed for the specific DDR in use.

b) Target Arbitration Functions

	Name	Description
	PLIB_DDR_MinCommand	Sets the minimum number of bursts to be serviced for a DDR target within (REQPER * 4) number of clocks.
	PLIB_DDR_MinLimit	Sets the minimum number of bursts for a DDR target.
	PLIB_DDR_ReqPeriod	Sets the timeout for MINCMD number of bursts to be serviced for a DDR target.

c) SDRAM Initialization Functions

	Name	Description
	PLIB_DDR_CmdDataIsComplete	Returns the value of the valid bit in the DDR2CMDISSUE register. This bit is cleared by hardware, when the SDRAM initialization data has been transmitted.
	PLIB_DDR_CmdDataSend	Transmits the data in the command registers to the SDRAM.
	PLIB_DDR_CmdDataValid	Indicates to the controller that the data in the command registers is valid.
	PLIB_DDR_ControllerEnable	Enables the controller for normal operations after SDRAM is initialized.
	PLIB_DDR_MaxCmdBrstCntSet	Sets the maximum number of commands that can be written to the controller in Burst mode.
	PLIB_DDR_NumHostCmdsSet	Sets the number of commands to be transmitted to the SDRAM.
	PLIB_DDR_CmdDataWrite	Writes an SDRAM command word to a command register in the controller.

d) SDRAM Addressing Functions

	Name	Description
	PLIB_DDR_BankAddressSet	Initializes the DDR controller memory configuration registers for bank address.
	PLIB_DDR_ChipSelectAddressSet	Initializes the DDR controller memory configuration registers for Chip Select address.
	PLIB_DDR_ColumnAddressSet	Initializes the DDR controller memory configuration registers for the column address.
	PLIB_DDR_RowAddressSet	Initializes the DDR controller memory configuration registers for row address.

e) SDRAM Timing Functions

	Name	Description
	PLIB_DDR_OdtReadParamSet	Sets the ODT parameters for data reads.
	PLIB_DDR_OdtWriteParamSet	Sets the ODT parameters for data writes.
	PLIB_DDR_DataDelaySet	Initializes the DDR controller with the data delays needed for the specific DDR in use.
	PLIB_DDR_PowerDownDelaySet	Initializes the DDR controller with the power down delays needed for the specific DDR in use.
	PLIB_DDR_SelfRefreshDelaySet	Initializes the DDR controller with the self-refresh delays needed for the specific DDR in use.
	PLIB_DDR_PrechargAllBanksSet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_ReadWriteDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.
	PLIB_DDR_RefreshTimingSet	Initializes the DDR controller refresh configuration.
	PLIB_DDR_WriteWriteDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.
	PLIB_DDR_PrechargeToRASDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_RASToCASDelaySet	Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.
	PLIB_DDR_RASToPrechargeDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_RASToRASBankDelaySet	Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.
	PLIB_DDR_RASToRASDelaySet	Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.

	PLIB_DDR_ReadToPrechargeDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_WriteToPrechargeDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_ReadReadDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.
	PLIB_DDR_WriteReadDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.

f) PHY Initialization Functions

	Name	Description
	PLIB_DDR_PHY_DDRTypeSet	Sets the DRAM type for the PHY.
	PLIB_DDR_PHY_DllMasterDelayStartSet	Sets the start value of the digital DLL master delay line.
	PLIB_DDR_PHY_DllRecalibDisable	Disables periodic recalibration of the internal digital DLL.
	PLIB_DDR_PHY_DllRecalibEnable	Enables periodic recalibration of the internal digital DLL, and sets the recalibration period.
	PLIB_DDR_PHY_DriveStrengthSet	Disables On Die Termination.
	PLIB_DDR_PHY_DrvStrgthCal	Calibrates the pad NFET and PFET output impedance.
	PLIB_DDR_PHY_ExternalDllEnable	Enables the use of an external DLL.
	PLIB_DDR_PHY_ExtraClockDisable	Does not enable the drive pad for an extra clock cycle after a write burst.
	PLIB_DDR_PHY_ExtraClockEnable	Enables the drive pad for an extra clock cycle after a write burst.
	PLIB_DDR_PHY_InternalDllEnable	Enables the use of the internal digital DLL.
	PLIB_DDR_PHY_OdtCal	Calibrates the pull-up and pull-down ODT impedance.
	PLIB_DDR_PHY_OdtCSDisable	Disables ODT on Chip Select while running SCL test.
	PLIB_DDR_PHY_OdtCSEnable	Enables ODT on Chip Select while running SCL test.
	PLIB_DDR_PHY_OdtDisable	Disables On Die Termination.
	PLIB_DDR_PHY_OdtEnable	Enables On Die Termination and sets the value.
	PLIB_DDR_PHY_PadReceiveEnable	Enables pad receivers on bidirectional I/O.
	PLIB_DDR_PHY_PreambleDlySet	Sets the length of the preamble for data writes.
	PLIB_DDR_PHY_ReadCASLatencySet	Sets the read CAS latency while running SCL test.
	PLIB_DDR_PHY_SCLDelay	Disables ODT on Chip Select while running SCL test.
	PLIB_DDR_PHY_SCLEnable	Enables SCL on the Chip Select 'cs'.
	PLIB_DDR_PHY_SCLTestBurstModeSet	Sets the burst mode of the DRAM while running SCL test.
	PLIB_DDR_PHY_WriteCASLatencySet	Sets the write CAS latency while running SCL test.
	PLIB_DDR_PHY_PadReceiveDisable	Disables pad receivers on bidirectional I/O.
	PLIB_DDR_PHY_SCLCapClkDelaySet	Sets capture clock delay during SCL.
	PLIB_DDR_PHY_SCLDDRClkDelaySet	Sets DDR clock delay during SCL.
	PLIB_DDR_PHY_HalfRateSet	Sets the PHY to half rate mode.
	PLIB_DDR_PHY_SCLStart	Runs PHY Self Calibration.
	PLIB_DDR_PHY_SCLStatus	Checks status of PHY Self Calibration.
	PLIB_DDR_PHY_AddCtlDriveStrengthSet	Sets the drive strength for the PHY address and control pads.
	PLIB_DDR_PHY_DataDriveStrengthSet	Sets the drive strength for the PHY data pads.
	PLIB_DDR_PHY_WriteCmdDelayDisable	Disables write command delay.
	PLIB_DDR_PHY_WriteCmdDelayEnable	Enables write command delay. The extra delay is needed for devices with even write latency (WL).

g) Feature Existence Functions

	Name	Description
	PLIB_DDR_ExistsAddressMapping	Identifies whether the AddressMapping feature exists on the DDR module.
	PLIB_DDR_ExistsArbitrationControl	Identifies whether the ArbitrationControl feature exists on the DDR module.
	PLIB_DDR_ExistsAutoPowerDown	Identifies whether the AutoPowerDown feature exists on the DDR module.
	PLIB_DDR_ExistsAutoPrecharge	Identifies whether the AutoPrecharge feature exists on the DDR module.
	PLIB_DDR_ExistsAutoSelfRefresh	Identifies whether the AutoSelfRefresh feature exists on the DDR module.
	PLIB_DDR_ExistsDDRCommands	Identifies whether the DDRCommands feature exists on the DDR module.
	PLIB_DDR_ExistsDDRControllerConfig	Identifies whether the DDRControllerConfig feature exists on the DDR module.
	PLIB_DDR_ExistsODTConfig	Identifies whether the ODTConfig feature exists on the DDR module.
	PLIB_DDR_ExistsPHY_DLLCalibration	Identifies whether the PHY_DLLCalibration feature exists on the DDR module.

	PLIB_DDR_ExistsPHY_PadConfig	Identifies whether the PHY_PadConfig feature exists on the DDR module.
	PLIB_DDR_ExistsPHY_SCLConfig	Identifies whether the PHY_SCLConfig feature exists on the DDR module
	PLIB_DDR_ExistsRefreshConfig	Identifies whether the RefreshConfig feature exists on the DDR module.
	PLIB_DDR_ExistsTimingDelays	Identifies whether the TimingDelays feature exists on the DDR module

h) Data Types and Constants

	Name	Description
	DDR_CMD_IDLE_NOP	
	DDR_PHY_DDR_TYPE	Defines the DDR type.
	DDR_CMD_LOAD_MODE	This is macro DDR_CMD_LOAD_MODE.
	DDR_PHY_DRIVE_STRENGTH	Defines the PHY Pad drive strength value.
	DDR_CMD_PRECHARGE_ALL	This is macro DDR_CMD_PRECHARGE_ALL.
	DDR_PHY_ODT	Defines the ODT termination value.
	DDR_CMD_REFRESH	This is macro DDR_CMD_REFRESH.
	DDR_PHY_PREAMBLE_DLY	Defines the PHY preamble delay value.
	DDR_PHY_SCL_BURST_MODE	Defines the burst mode for SCL.
	DDR_PHY_SCL_DELAY	Defines the SCL delay.
	DDR_HOST_CMD_REG	Defines the DDR host command register.
	DDR_MODULE_ID	Enumeration: DDR_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on Microchip microcontrollers. Refer to the data sheet to get the correct number of modules defined for desired microcontroller.
	DDR_TARGET	Defines the different targets to which the DDR controller is connected.

Description

This section describes the Application Programming Interface (API) functions of the DDR SDRAM Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_DDR_BigEndianSet Function

Sets the DDR data endianness to big.

File

plib_ddr.h

C

```
void PLIB_DDR_BigEndianSet(DDR_MODULE_ID index);
```

Returns

None.

Description

This function sets the DDR data endianness to big.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_BigEndianSet(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_BigEndianSet(DDR_MODULE_ID index);

PLIB_DDR_FullRateSet Function

Sets the DDR controller to Full-rate mode.

File

plib_ddr.h

C

```
void PLIB_DDR_FullRateSet(DDR_MODULE_ID index);
```

Returns

None.

Description

This function sets the DDR controller to Full-rate mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_FullRateSet(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_FullRateSet(DDR_MODULE_ID index);

PLIB_DDR_HalfRateSet Function

Sets the DDR controller to Half-rate mode.

File

plib_ddr.h

C

```
void PLIB_DDR_HalfRateSet(DDR_MODULE_ID index);
```

Returns

None.

Description

This function sets the DDR controller to Half-rate mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_HalfRateSet(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_HalfRateSet(DDR_MODULE_ID index);

PLIB_DDR_LittleEndianSet Function

Sets the DDR data endianness to little.

File

plib_ddr.h

C

```
void PLIB_DDR_LittleEndianSet(DDR_MODULE_ID index);
```

Returns

None.

Description

This function sets the DDR data endianness to little.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_LittleEndianSet(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_LittleEndianSet(DDR_MODULE_ID index);

PLIB_DDR_MaxPendingRefSet Function

Initializes the DDR controller refresh configuration.

File

plib_ddr.h

C

```
void PLIB_DDR_MaxPendingRefSet(DDR_MODULE_ID index, uint8_t maxRefs);
```

Returns

None.

Description

This function initializes the DDR controller refresh configuration and programs the number of refreshes that may be pending at one time. The following register field is programmed with this function:

- MAXREFS<4:0> - DDRREFCFG<24:26>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define MAX_PEND_REF 7

PLIB_DDR_MaxPendingRefSet(DDR_ID_0, MAX_PEND_REF);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
maxRefs	Maximum number of refreshes that may be pending at one time

Function

void PLIB_DDR_MaxPendingRefSet(DDR_MODULE_ID index, uint8_t maxRefs);

PLIB_DDR_OdtReadDisable Function

Selects which Chip Select to disable ODT for data reads.

File

plib_ddr.h

C

```
void PLIB_DDR_OdtReadDisable(DDR_MODULE_ID index, uint8_t odtCS);
```

Returns

None.

Description

This function selects which Chip Select to disable ODT for data reads.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_OdtReadDisable(DDR_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
odtCS	Chip select to program ODT control value

Function

void PLIB_DDR_OdtReadDisable(DDR_MODULE_ID index, uint8_t odtCS);

PLIB_DDR_OdtReadEnable Function

Selects which Chip Select to enable ODT for data reads.

File

plib_ddr.h

C

```
void PLIB_DDR_OdtReadEnable(DDR_MODULE_ID index, uint8_t odtCS);
```

Returns

None.

Description

This function selects which Chip Select to enable ODT for data reads.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_OdtReadEnable(DDR_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
odtCS	Chip select to program ODT control value

Function

void PLIB_DDR_OdtReadEnable(DDR_MODULE_ID index, uint8_t odtCS);

PLIB_DDR_OdtWriteDisable Function

Selects which Chip Select to disable ODT for data writes.

File

plib_ddr.h

C

```
void PLIB_DDR_OdtWriteDisable(DDR_MODULE_ID index, uint8_t odtCS);
```

Returns

None.

Description

This function selects which Chip Select to disable ODT for data writes.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_OdtWriteDisable(DDR_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
odtCS	Chip select to program ODT control value

Function

void PLIB_DDR_OdtWriteDisable(DDR_MODULE_ID index, uint8_t odtCS);

PLIB_DDR_OdtWriteEnable Function

Selects which Chip Select to enable ODT for data writes.

File

plib_ddr.h

C

```
void PLIB_DDR_OdtWriteEnable(DDR_MODULE_ID index, uint8_t odtCS);
```

Returns

None.

Description

This function selects which Chip Select to enable ODT for data writes.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_OdtWriteEnable(DDR_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
odtCS	Chip select to program ODT control value

Function

void PLIB_DDR_OdtWriteEnable(DDR_MODULE_ID index, uint8_t odtCS);

PLIB_DDR_AutoPchrgDisable Function

Prevents the DDR controller from issuing an auto-precharge command to close the bank at the end of every user command.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoPchrgDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function prevents the DDR controller from issuing a auto-precharge command to close the bank at the end of every user command.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoPchrgDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoPchrgDisable(DDR_MODULE_ID index);

PLIB_DDR_AutoPowerDownDisable Function

Prevents the DDR controller from automatically entering Power-down mode.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoPowerDownDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function prevents the DDR controller from automatically entering Power-down mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoPowerDownDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoPowerDownDisable(DDR_MODULE_ID index);

PLIB_DDR_AutoPchrgPowerDownDisable Function

Prevents the DDR controller from automatically entering Precharge Power-down mode.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoPchrgPowerDownDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function prevents the DDR controller from automatically entering Precharge Power-down mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoPchrgPowerDownDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoPchrgPowerDownDisable(DDR_MODULE_ID index);

PLIB_DDR_AutoSelfRefreshDisable Function

Prevents the DDR controller from automatically entering Self-refresh mode.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoSelfRefreshDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function prevents the DDR controller from automatically entering Self-refresh mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoSelfRefreshDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoSelfRefreshDisable(DDR_MODULE_ID index);

PLIB_DDR_AutoPchrgPowerDownEnable Function

Enables the DDR controller to automatically enter Precharge Power-down mode.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoPchrgPowerDownEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the DDR controller to automatically enter Precharge Power-down mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoPchrgPowerDownEnable(DDR_ID_0);
```

I

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoPchrgPowerDownEnable(DDR_MODULE_ID index);

PLIB_DDR_AutoPowerDownEnable Function

Enables the DDR controller to automatically enter Power-down mode.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoPowerDownEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the DDR controller to automatically enter Power-down mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoPowerDownEnable(DDR_ID_0, pdd);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoPowerDownEnable(DDR_MODULE_ID index);

PLIB_DDR_AutoSelfRefreshEnable Function

Enables the DDR controller to automatically enter Self-refresh mode.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoSelfRefreshEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the DDR controller to automatically enter Self-refresh mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoSelfRefreshEnable(DDR_ID_0, srd);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoSelfRefreshEnable(DDR_MODULE_ID index);

PLIB_DDR_AutoPchrgEnable Function

Enables the DDR controller to issue an auto-precharge command to close the bank at the end of every user command.

File

plib_ddr.h

C

```
void PLIB_DDR_AutoPchrgEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the DDR controller to issue a auto-precharge command to close the bank at the end of every user command.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_AutoPchrgEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_AutoPchrgEnable(DDR_MODULE_ID index);

PLIB_DDR_TfawDelaySet Function

Initializes the DDR controller with the four-bank activation window needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_TfawDelaySet(DDR_MODULE_ID index, uint32_t tFaw, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the four-bank activation window for the DDR in use. The following register field is programmed with this function:

- FAWTDLY<5:0> - DDRDLYCFG3<21:16>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tFAW            35000   // psec

PLIB_DDR_TfawDelaySet(DDR_ID_0, tFAW, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tFaw	Four bank activation window (psec)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_TfawDelaySet(DDR_MODULE_ID index, uint32_t tFaw, uint32_t ctrlClkPer);

b) Target Arbitration Functions

PLIB_DDR_MinCommand Function

Sets the minimum number of bursts to be serviced for a DDR target within (REQPER * 4) number of clocks.

File

plib_ddr.h

C

```
void PLIB_DDR_MinCommand(DDR_MODULE_ID index, uint8_t minCmd, DDR_TARGET target);
```

Returns

None.

Description

This function sets the minimum number of bursts to be serviced for a DDR target within (REQPER * 4) number of clocks. Used with PLIB_DDR_ReqPeriod to determine the total bandwidth allocated to a target.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

The target must be selected with PLIB_DDR_TargetSelect() prior to using this function.

Example

```
PLIB_DDR_MinCommand(DDR_ID_0, 83, DDR_TARGET_CPU);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
minCmd	Number of bursts
target	Target for minCmd setting

Function

void PLIB_DDR_MinCommand(DDR_MODULE_ID index, uint8_t minCmd, DDR_TARGET target)

PLIB_DDR_MinLimit Function

Sets the minimum number of bursts for a DDR target.

File

plib_ddr.h

C

```
void PLIB_DDR_MinLimit(DDR_MODULE_ID index, uint8_t minLim, DDR_TARGET target);
```

Returns

None.

Description

This function sets the minimum number of bursts (two cycles per burst) that a target must have uninterrupted access to without interference from another target.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

The target must be selected with PLIB_DDR_TargetSelect prior to using this function.

Example

```
PLIB_DDR_MinLimit(DDR_ID_0, 1, DDR_TARGET_CPU);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
minLim	Number of bursts
target	Target for minLim setting

Function

void PLIB_DDR_MinLimit(DDR_MODULE_ID index, uint8_t minLim, DDR_TARGET target)

PLIB_DDR_ReqPeriod Function

Sets the timeout for MINCMD number of bursts to be serviced for a DDR target.

File

plib_ddr.h

C

```
void PLIB_DDR_ReqPeriod(DDR_MODULE_ID index, uint8_t reqPer, DDR_TARGET target);
```

Returns

None.

Description

This function sets the timeout for MINCMD number of bursts to be serviced for a DDR target. Used with PLIB_DDR_MinCommand to determine the total bandwidth allocated to a target.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

The target must be selected with PLIB_DDR_TargetSelect prior to using this function.

Example

```
PLIB_DDR_ReqPeriod(DDR_ID_0, 167, DDR_TARGET_CPU);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
reqPer	Number of clocks (multiplied by 4)
target	Target for reqPer setting

Function

void PLIB_DDR_ReqPeriod(DDR_MODULE_ID index, uint8_t reqPer, DDR_TARGET target)

c) SDRAM Initialization Functions

PLIB_DDR_CmdDataIsComplete Function

Returns the value of the valid bit in the DDR2CMDISSUE register. This bit is cleared by hardware, when the SDRAM initialization data has been transmitted.

File

plib_ddr.h

C

```
bool PLIB_DDR_CmdDataIsComplete(DDR_MODULE_ID index);
```

Returns

None.

Description

This function returns the value of the valid bit in the DDR2CMDISSUE register. This bit is cleared by hardware, when the SDRAM initialization data has been transmitted.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
while (PLIB_DDR_CmdDataIsComplete(DDR_ID_0));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_DDR_CmdDataIsComplete(DDR_MODULE_ID index);

PLIB_DDR_CmdDataSend Function

Transmits the data in the command registers to the SDRAM.

File

plib_ddr.h

C

```
void PLIB_DDR_CmdDataSend(DDR_MODULE_ID index);
```

Returns

None.

Description

This function transmits the data in the command registers to the SDRAM.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_CmdDataSend(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_CmdDataSend(DDR_MODULE_ID index);

PLIB_DDR_CmdDataValid Function

Indicates to the controller that the data in the command registers is valid.

File

plib_ddr.h

C

```
void PLIB_DDR_CmdDataValid(DDR_MODULE_ID index);
```

Returns

None.

Description

This function indicates to the controller that the data in the command registers is valid.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_CmdDataValid(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_CmdDataValid(DDR_MODULE_ID index);

PLIB_DDR_ControllerEnable Function

Enables the controller for normal operations after SDRAM is initialized.

File

plib_ddr.h

C

```
void PLIB_DDR_ControllerEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the controller for normal operations after SDRAM is initialized.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_ControllerEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_ControllerEnable(DDR_MODULE_ID index);

PLIB_DDR_MaxCmdBrstCntSet Function

Sets the maximum number of commands that can be written to the controller in Burst mode.

File

plib_ddr.h

C

```
void PLIB_DDR_MaxCmdBrstCntSet(DDR_MODULE_ID index, int8_t maxCmds);
```

Returns

None.

Description

This function sets the maximum number of commands that can be written to the controller in Burst mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_MaxCmdBrstCntSet(DDR_ID_0, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

maxCmds	Maximum number of commands that can be written to the controller in burst mode

Function

void PLIB_DDR_MaxCmdBrstCntSet(DDR_MODULE_ID index, int8_t maxCmds);

PLIB_DDR_NumHostCmdsSet Function

Sets the number of commands to be transmitted to the SDRAM.

File

plib_ddr.h

C

```
void PLIB_DDR_NumHostCmdsSet(DDR_MODULE_ID index, int8_t numCmds);
```

Returns

None.

Description

This function sets the number of commands to be transmitted to the SDRAM.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_NumHostCmdsSet(DDR_ID_0, 0x0B);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
numCmds	Number of commands to be transmitted to the SDRAM

Function

void PLIB_DDR_NumHostCmdsSet(DDR_MODULE_ID index, int8_t numCmds);

PLIB_DDR_CmdDataWrite Function

Writes an SDRAM command word to a command register in the controller.

File

plib_ddr.h

C

```
void PLIB_DDR_CmdDataWrite(DDR_MODULE_ID index, DDR_HOST_CMD_REG cmdReg, uint32_t cmdData);
```

Returns

None.

Description

This function writes an SDRAM command word to a command register in the controller.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define DDR_CMD_IDLE_NOP 0x00FFFFFF
PLIB_DDR_CmdDataWrite(DDR_ID_0, DDR_HOST_CMD_REG_100, DDR_CMD_IDLE_NOP);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cmdData	32-bit command data word
cmdReg	Command register

Function

void PLIB_DDR_CmdDataWrite(DDR_MODULE_ID index, DDR_HOST_CMD_REG cmdReg, uint32_t cmdData);

d) SDRAM Addressing Functions

PLIB_DDR_BankAddressSet Function

Initializes the DDR controller memory configuration registers for bank address.

File

plib_ddr.h

C

```
void PLIB_DDR_BankAddressSet(DDR_MODULE_ID index, uint32_t bnkShft, uint32_t bnkMsk);
```

Returns

None.

Description

This function initializes the DDR controller memory configuration registers for the bank address. The following register fields are programmed with this

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_BankAddressSet(DDR_ID_0, 9, 0x03);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bnkShft	Number of bits to shift the bank address
bnkMsk	Bank address bit mask

Function

void PLIB_DDR_BankAddressSet(DDR_MODULE_ID index, uint32_t bnkShft, uint32_t bnkMsk);

- BNKADDR<4:0> - DDRMEMCFG0<12:8>
- BNKADDRMSK<2:0> - DDRMEMCFG4<2:0>

PLIB_DDR_ChipSelectAddressSet Function

Initializes the DDR controller memory configuration registers for Chip Select address.

File

plib_ddr.h

C

```
void PLIB_DDR_ChipSelectAddressSet(DDR_MODULE_ID index, uint32_t csShft, uint32_t csMsk);
```

Returns

None.

Description

This function initializes the DDR controller memory configuration registers for the Chip Select address. The following register fields are programmed with this

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_ChipSelectAddressSet(DDR_ID_0, 18, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
csShft	Number of bits to shift the Chip Select address
csMsk	Chip select address bit mask

Function

void PLIB_DDR_ChipSelectAddressSet(DDR_MODULE_ID index, uint32_t csShft, uint32_t csMsk);

- CSADDR<4:0> - DDRMEMCFG0<20:16>
- CSADDRMSK<2:0> - DDRMEMCFG4<8:6>

PLIB_DDR_ColumnAddressSet Function

Initializes the DDR controller memory configuration registers for the column address.

File

plib_ddr.h

C

```
void PLIB_DDR_ColumnAddressSet(DDR_MODULE_ID index, uint32_t colShft, uint32_t colMskLo, uint32_t colMskHi);
```

Returns

None.

Description

This function initializes the DDR controller memory configuration registers for the column address. The following register fields are programmed with this

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_ColumnAddressSet(DDR_ID_0, 0, 0x1FF, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
colShft	Number of bits to shift the column address

colMaskLo	Column address bit mask low
colMaskHi	Column address bit mask high

Function

void PLIB_DDR_ColumnAddressSet(DDR_MODULE_ID index, uint32_t colShft, uint32_t colMskLo, uint32_t colMskHi);

- CLHADDR<4:0> - DDRMEMCFG0<28:24>
- CLADDRHMSK<12:0> - DDRMEMCFG2<12:0>
- CLADDRLMSK<12:0> - DDRMEMCFG3<12:0>

PLIB_DDR_RowAddressSet Function

Initializes the DDR controller memory configuration registers for row address.

File

plib_ddr.h

C

```
void PLIB_DDR_RowAddressSet(DDR_MODULE_ID index, uint32_t rowShft, uint32_t rowMsk);
```

Returns

None.

Description

this function initializes the DDR controller memory configuration registers for the row address. The following register fields are programmed with this function:

- RWADDR<4:0> - DDRMEMCFG0<4:0>
- RWADDRMASK<12:0> - DDRMEMCFG1<12:0>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_RowAddressSet(DDR_ID_0, 12, 0x1FFF);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
rowShft	Number of bits to shift the row address
rowMask	Row address bit mask

Function

void PLIB_DDR_RowAddressSet(DDR_MODULE_ID index, uint32_t rowShft, uint32_t rowMsk);

e) SDRAM Timing Functions

PLIB_DDR_OdtReadParamSet Function

Sets the ODT parameters for data reads.

File

plib_ddr.h

C

```
void PLIB_DDR_OdtReadParamSet(DDR_MODULE_ID index, uint8_t rLen, uint8_t rDly);
```

Returns

None.

Description

This function sets the ODT parameters for data reads.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_OdtReadParamSet(DDR_ID_0, 2, 4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
rLen	The number of clocks the ODT is turned on for reads
rDly	The number of clocks after a read command before turning on ODT to the DDR

Function

void PLIB_DDR_OdtReadParamSet(DDR_MODULE_ID index, uint8_t rLen, uint8_t rDly);

PLIB_DDR_OdtWriteParamSet Function

Sets the ODT parameters for data writes.

File

plib_ddr.h

C

```
void PLIB_DDR_OdtWriteParamSet(DDR_MODULE_ID index, uint8_t wLen, uint8_t wDly);
```

Returns

None.

Description

This function sets the ODT parameters for data writes.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_OdtWriteParamSet(DDR_ID_0, 3, 1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
wLen	The number of clocks the ODT is turned on for writes
wDly	The number of clocks after a write command before turning on ODT to the DDR

Function

void PLIB_DDR_OdtWriteParamSet(DDR_MODULE_ID index, uint8_t wLen, uint8_t wDly);

PLIB_DDR_DataDelaySet Function

Initializes the DDR controller with the data delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_DataDelaySet(DDR_MODULE_ID index, uint8_t rLat, uint8_t wLat);
```

Returns

None.

Description

This function initializes the DDR controller with the data delays for the DDR in use. The following register fields are programmed with this function:

- RBENDDLY<3:0> - DDRDLYCFG2<31:28>
- NXTDATRQDLY<3:0> - DDRXFERCFG<3:0>
- NXTDATAVDLY<4:0> - DDRXFERCFG<7:4>, DDRDLYCFG1<28>
- RDATENDLY<3:0> - DDRXFERCFG<19:16>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_DataDelaySet(DDR_ID_0, 5, 4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
rLat	Read (CAS) latency in cycles (RL)
wLat	Write latency in cycles (WL)

Function

void PLIB_DDR_DataDelaySet(DDR_MODULE_ID index, uint8_t rLat, uint8_t wLat);

PLIB_DDR_PowerDownDelaySet Function

Initializes the DDR controller with the power down delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_PowerDownDelaySet(DDR_MODULE_ID index, uint32_t pwrDnDly, uint32_t tCKE, uint32_t tXP);
```

Returns

None.

Description

This function initializes the DDR controller with the power down delays for the DDR in use. The folPowerDownDelaySetlowing register fields are programmed with this function:

- PWRDNDLY<7:0> - DDRPWRCFG<11:4>
- PWRDNMINDLY<3:0> - DDR2DLYCFG1<19:16>
- PWRDNEXDLY<5:0> - DDR2DLYCFG1<25:20>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define CLK_PERIOD          2500 // psec
#define CTRL_CLK_PERIOD     (CLK_PERIOD * 2)
#define PWR_DN_DLY          8
#define tCKE                3
#define tXP                 2

PLIB_DDR_PowerDownDelaySet(DDR_ID_0, PWR_DN_DLY, tCKE, tXP);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pwrDnDly	Power down delay (entry) in cycles
tCKE	Power down minimum delay in cycles (tCKE)
tXP	Power down exit delay in cycles (tXP)

Function

void PLIB_DDR_PowerDownDelaySet(DDR_MODULE_ID index, uint32_t pwrDnDly,

uint32_t tCKE, uint32_t tXP);

PLIB_DDR_SelfRefreshDelaySet Function

Initializes the DDR controller with the self-refresh delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_SelfRefreshDelaySet(DDR_MODULE_ID index, uint32_t slfRefDly, uint32_t tCKE, uint32_t tDLLK);
```

Returns

None.

Description

This function initializes the DDR controller with the self-refresh delays for the specific DDR in use. The following register fields are programmed with this function:

- SLFREFDLY<9:0> - DDRPWRCFG<21:12>
- SLFREFMINDLY<7:0> - DDRDLYCFG1<7:0>
- SLFREFEXDLY<8:0> - DDRDLYCFG1<15:8>, DDR2DLYCFG1<30>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define SELF_REF_DLY        17
#define tCKE                3
PLIB_DDR_SelfRefreshDelaySet(DDR_ID_0, SELF_REF_DLY, tCKE, 85);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
slfRefDly	Self refresh delay in number of clock cycles divided by 1024 (e.g., 1 = 1024 clocks, 2 = 2048 clocks, etc.)
tCKE	Power down minimum delay in cycles (tCKE)

tDLLK	DLL lock delay time in cycles (tDLLK)

Function

void PLIB_DDR_SelfRefreshDelaySet(DDR_MODULE_ID index, uint32_t slfRefDly,
uint32_t tCKE, uint32_t tDLLK);

PLIB_DDR_PrechargAllBanksSet Function

Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_PrechargAllBanksSet(DDR_MODULE_ID index, uint32_t tRP, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write precharge delays for the DDR in use. The following register fields are programmed with this function:

- PCHRGALLDLY<3:0> - DDRDLYCFG2<3:0>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRP             12500   // psec

PLIB_DDR_PrechargAllBanksSet(DDR_ID_0, tRP, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRP	Precharge to active delay (psec)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_PrechargAllBanksSet(DDR_MODULE_ID index, uint32_t tRP, uint32_t ctrlClkPer);

PLIB_DDR_ReadWriteDelaySet Function

Initializes the DDR controller with the read/write delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_ReadWriteDelaySet(DDR_MODULE_ID index, uint8_t bLen, uint8_t wLat, uint8_t rLat);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write delays for the specific DDR in use. The following register fields are programmed with

this function:

- R2WDLY<3:0> - DDRDLYCFG0<27:24>
- RMWDLY<3:0> - DDRDLYCFG0<31:28>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define BL  2
#define WL  4
#define RL  5
PLIB_DDR_ReadWriteDelaySet(DDR_ID_0 BL, WL, RL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bLen	Burst length in cycles (BL)
wLat	Write latency in cycles (WL)
rLat	Read latency in cycles (RL)

Function

void PLIB_DDR_ReadWriteDelaySet(DDR_MODULE_ID index, uint8_t bLen,

uint8_t wLat, uint8_t rLat);

PLIB_DDR_RefreshTimingSet Function

Initializes the DDR controller refresh configuration.

File

plib_ddr.h

C

```
void PLIB_DDR_RefreshTimingSet(DDR_MODULE_ID index, uint32_t tRFC, uint32_t tRFI, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller refresh configuration and programs the refresh interval and delay between refreshes. The following register fields are programmed with this function:

- REFCNT<15:0> - DDRREFCFG<15:0>
- REFDLY<5:0> - DDRREFCFG<23:16>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)

#define tRFC            75000   // psec
#define tRFI            7800000 // psec

PLIB_DDR_RefreshConfig(DDR_ID_0, tRFC, tRFI, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRFC	Refresh Cycle Time in clock cycles
tRFI	Refresh Interval in clock cycles
ctrlClkPer	DDR Controller clock period

Function

void PLIB_DDR_RefreshTimingSet(DDR_MODULE_ID index, uint32_t tRFC, uint32_t tRFI,
uint32_t ctrlClkPer);

PLIB_DDR_WriteWriteDelaySet Function

Initializes the DDR controller with the read/write delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_WriteWriteDelaySet(DDR_MODULE_ID index, uint8_t bLen);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write delays for the specific DDR in use. The following register fields are programmed with this function:

- W2WDLY<3:0> - DDRDLYCFG0<19:16>
- W2WCSDLY<3:0> - DDRDLYCFG0<23:20>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define BL  2
PLIB_DDR_WriteWriteDelaySet(DDR_ID_0, BL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bLen	Burst length in cycles (BL)

Function

void PLIB_DDR_WriteWriteDelaySet(DDR_MODULE_ID index, uint8_t bLen);

PLIB_DDR_PrechargeToRASDelaySet Function

Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_PrechargeToRASDelaySet(DDR_MODULE_ID index, uint32_t tRP, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write precharge delays for the DDR in use. The following register fields are programmed with this function:

- PCHRG2RASDLY<3:0> - DDRDLYCFG2<24:27>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRP             12500   // psec

PLIB_DDR_PrechargeToRASDelaySet(DDR_ID_0, tRP, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRP	Precharge to active delay (psec)
ctrlClkPer	DDR Controller clock period

Function

void PLIB_DDR_PrechargeToRASDelaySet(DDR_MODULE_ID index, uint32_t tRP,

uint32_t ctrlClkPer);

PLIB_DDR_RASToCASDelaySet Function

Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_RASToCASDelaySet(DDR_MODULE_ID index, uint32_t tRCD, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the row/column address delays for the DDR in use. The following register fields are programmed with this function:

- RAS2CASDLY<3:0> - DDRDLYCFG2<23:20>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRCD            12500   // psec

PLIB_DDR_RASToCASDelaySet(DDR_ID_0, tRCD, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRCD	RAS to CAS delay (psec)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_RASToCASDelaySet(DDR_MODULE_ID index, uint32_t tRCD,

uint32_t ctrlClkPer);

PLIB_DDR_RASToPrechargeDelaySet Function

Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_RASToPrechargeDelaySet(DDR_MODULE_ID index, uint32_t tRAS, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write precharge delays for the DDR in use. The following register fields are programmed with this function:

- RAS2PCHRGDLY<4:0> - DDRDLYCFG3<4:0>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRAS            45000   // psec

PLIB_DDR_RASToPrechargeDelaySet(DDR_ID_0, tRAS, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRAS	Active to Precharge delay (psec)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_RASToPrechargeDelaySet(DDR_MODULE_ID index, uint32_t tRAS, uint32_t ctrlClkPer);

PLIB_DDR_RASToRASBankDelaySet Function

Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_RASToRASBankDelaySet(DDR_MODULE_ID index, uint32_t tRC, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the row/column address delays for the DDR in use. The following register fields are programmed with this function:

- RAS2RASSBNKDLY<5:0> - DDRDLYCFG3<13:8>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRC             57500   // psec

PLIB_DDR_RASToRASBankDelaySet(DDR_ID_0, tRC, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRC	Row cycle time (psec)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_RASToRASBankDelaySet(DDR_MODULE_ID index, uint32_t tRC, uint32_t ctrlClkPer);

PLIB_DDR_RASToRASDelaySet Function

Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_RASToRASDelaySet(DDR_MODULE_ID index, uint32_t tRRD, uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the row/column address delays for the DDR in use. The following register fields are programmed with this function:

- RAS2RASDLY<3:0> - DDRDLYCFG2<19:16>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRRD            7500    // psec

PLIB_DDR_RASToRASDelaySet(DDR_ID_0, tRRD, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRRD	RAS to RAS delay (psec)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_RASToRASDelaySet(DDR_MODULE_ID index, uint32_t tRRD, uint32_t ctrlClkPer);

PLIB_DDR_ReadToPrechargeDelaySet Function

Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_ReadToPrechargeDelaySet(DDR_MODULE_ID index, uint32_t tRTP, uint8_t bLen, uint32_t
ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write precharge delays for the DDR in use. The following register fields are programmed with this function:

- R2PCHRGDLY<3:0> - DDRDLYCFG2<11:8>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRTP            7500    // psec
#define BL              2

PLIB_DDR_ReadToPrechargeDelaySet(DDR_ID_0, tRTP, BL, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tRTP	Read to precharge delay (psec)
bLen	Burst length in cycles (BL)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_ReadToPrechargeDelaySet(DDR_MODULE_ID index, uint32_t tRTP,

uint8_t bLen, uint32_t ctrlClkPer);

PLIB_DDR_WriteToPrechargeDelaySet Function

Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_WriteToPrechargeDelaySet(DDR_MODULE_ID index, uint32_t tWR, uint8_t bLen, uint8_t wLat,
uint32_t ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write precharge delays for the DDR in use. The following register fields are programmed with this function:

- W2PCHRGDLY<4:0> - DDRDLYCFG2<15:12>, DDRDLYCFG1<26>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define clock_period    2500    // psec
#define CTRL_CLK_PERIOD (clock_period * 2)
#define tRTP            7500    // psec
#define BL              2
#define wLat            4

PLIB_DDR_WriteToPrechargeDelaySet(DDR_ID_0, tRTP, BL, wLat, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tWR	Write recovery delay (psec)
bLen	Burst length in cycles (BL)
wLat	Write latency in cycles (WL)
ctrlClkPer	DDR Controller clock period (psec)

Function

void PLIB_DDR_WriteToPrechargeDelaySet(DDR_MODULE_ID index, uint32_t tWR,

uint8_t bLen, uint8_t wLat, uint32_t ctrlClkPer);

PLIB_DDR_ReadReadDelaySet Function

Initializes the DDR controller with the read/write delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_ReadReadDelaySet(DDR_MODULE_ID index, uint8_t bLen);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write delays for the specific DDR in use. The following register fields are programmed with this function:

- R2RDLY<3:0> - DDRDLYCFG0<11:8>
- R2RCSDLY<3:0> - DDRDLYCFG0<15:12>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define BL  2
PLIB_DDR_ReadReadDelaySet(DDR_ID_0, BL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bLen	Burst length in cycles (BL)

Function

void PLIB_DDR_ReadReadDelaySet(DDR_MODULE_ID index, uint8_t bLen);

PLIB_DDR_WriteReadDelaySet Function

Initializes the DDR controller with the read/write delays needed for the specific DDR in use.

File

plib_ddr.h

C

```
void PLIB_DDR_WriteReadDelaySet(DDR_MODULE_ID index, uint32_t tWTR, uint8_t bLen, uint8_t wLat, uint32_t
ctrlClkPer);
```

Returns

None.

Description

This function initializes the DDR controller with the read/write delays for the specific DDR in use. The following register fields are programmed with this function:

- W2RDLY<4:0> - DDRDLYCFG0<3:0>, DDRDLYCFG1<27>
- W2RCSDLY<4:0> - DDRDLYCFG0<7:4>, DDRDLYCFG1<28>

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
#define CLK_PERIOD          2500 // psec
#define CTRL_CLK_PERIOD     (CLK_PERIOD * 2)
#define tWTR                7500

PLIB_DDR_WriteReadDelaySet(DDR_ID_0, tWTR, 2, 4, CTRL_CLK_PERIOD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
tWTR	Write to read latency (in psecs)
bLen	Burst length in cycles (BL)
wLat	Write latency in cycles (WL)
ctrlClkPer	Control clock period (psec)

Function

void PLIB_DDR_WriteReadDelaySet(DDR_MODULE_ID index, uint32_t tWTR,

uint8_t bLen, uint8_t wLat, uint32_t ctrlClkPer);

f) PHY Initialization Functions

PLIB_DDR_PHY_DDRTypeSet Function

Sets the DRAM type for the PHY.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DDRTypeSet(DDR_MODULE_ID index, DDR_PHY_DDR_TYPE ddrType);
```

Returns

None.

Description

This function sets the DRAM type for the PHY.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DDRTypeSet(DDR_ID_0, DDR_PHY_DDR_TYPE_DDR2);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
ddrType	DDR type (DDR2 or DDR3)

Function

void PLIB_DDR_PHY_DDRTypeSet(DDR_MODULE_ID index, DDR_PHY_DDR_TYPE ddrType);

PLIB_DDR_PHY_DllMasterDelayStartSet Function

Sets the start value of the digital DLL master delay line.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DllMasterDelayStartSet(DDR_MODULE_ID index, uint8_t dlyStart);
```

Returns

None.

Description

This function sets the start value of the digital DLL master delay line.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DllMasterDelayStartSet(DDR_ID_0, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dlyStart	

Function

void PLIB_DDR_PHY_DllMasterDelayStartSet(DDR_MODULE_ID index, uint8_t dlyStart);

PLIB_DDR_PHY_DllRecalibDisable Function

Disables periodic recalibration of the internal digital DLL.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DllRecalibDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

this function disables periodic recalibration of the internal digital DLL.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DllRecalibDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_DllRecalibDisable(DDR_MODULE_ID index);

PLIB_DDR_PHY_DllRecalibEnable Function

Enables periodic recalibration of the internal digital DLL, and sets the recalibration period.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DllRecalibEnable(DDR_MODULE_ID index, uint32_t recalibCnt);
```

Returns

None.

Description

This function enables periodic recalibration of the internal digital DLL, and sets the recalibration period.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DllRecalibEnable(DDR_ID_0, 100);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
recalibCnt	Period of DLL recalibration in units of (PHY clock * 256)

Function

void PLIB_DDR_PHY_DllRecalibEnable(DDR_MODULE_ID index, uint32_t recalibCnt);

PLIB_DDR_PHY_DriveStrengthSet Function

Disables On Die Termination.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DriveStrengthSet(DDR_MODULE_ID index, DDR_PHY_DRIVE_STRENGTH drvStr);
```

Returns

None.

Description

This function disables On Die Termination (ODT).

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DriveStrengthSet(DDR_ID_0, DDR_PHY_DRIVE_STRENGTH_FULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
drvStr	Drive strength

Function

void PLIB_DDR_PHY_DriveStrengthSet(DDR_MODULE_ID index, DDR_PHY_DRIVE_STRENGTH drvStr);

PLIB_DDR_PHY_DrvStrgthCal Function

Calibrates the pad NFET and PFET output impedance.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DrvStrgthCal(DDR_MODULE_ID index, uint8_t nFet, uint8_t pFet);
```

Returns

None.

Description

This function calibrates the pad NFET and PFET output impedance.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DrvStrgthCal(DDR_ID_0, 0xF, 0xF);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
nFet	NFET impedance calibration value
pFet	PFET impedance calibration value

Function

void PLIB_DDR_PHY_DrvStrgthCal(DDR_MODULE_ID index, uint8_t nFet, uint8_t pFet);

PLIB_DDR_PHY_ExternalDllEnable Function

Enables the use of an external DLL.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_ExternalDllEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the use of an external DLL.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_ExternalDllEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_ExternalDllEnable(DDR_MODULE_ID index);

PLIB_DDR_PHY_ExtraClockDisable Function

Does not enable the drive pad for an extra clock cycle after a write burst.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_ExtraClockDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function does not enable the drive pad for an extra clock cycle after a write burst.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_ExtraClockDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_ExtraClockDisable(DDR_MODULE_ID index);

PLIB_DDR_PHY_ExtraClockEnable Function

Enables the drive pad for an extra clock cycle after a write burst.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_ExtraClockEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the drive pad for an extra clock cycle after a write burst.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_ExtraClockEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_ExtraClockEnable(DDR_MODULE_ID index);

PLIB_DDR_PHY_InternalDllEnable Function

Enables the use of the internal digital DLL.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_InternalDllEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables the use of the internal digital DLL.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_InternalDllEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_InternalDllEnable(DDR_MODULE_ID index);

PLIB_DDR_PHY_OdtCal Function

Calibrates the pull-up and pull-down ODT impedance.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_OdtCal(DDR_MODULE_ID index, uint8_t puCal, uint8_t pdCal);
```

Returns

None.

Description

This function calibrates the pull-up and pull-down ODT impedance.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_OdtCal(DDR_ID_0, 2, 2);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
puCal	Pull-up calibration value
pdCal	Pull-down calibration value

Function

void PLIB_DDR_PHY_OdtCal(DDR_MODULE_ID index, uint8_t puCal, uint8_t pdCal);

PLIB_DDR_PHY_OdtCSDisable Function

Disables ODT on Chip Select while running SCL test.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_OdtCSDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function disables ODT on Chip Select while running SCL test.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_OdtCSDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_OdtCSDisable(DDR_MODULE_ID index);

PLIB_DDR_PHY_OdtCSEnable Function

Enables ODT on Chip Select while running SCL test.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_OdtCSEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables ODT on Chip Select while running SCL test.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_OdtCSEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_OdtCSEnable(DDR_MODULE_ID index);

PLIB_DDR_PHY_OdtDisable Function

Disables On Die Termination.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_OdtDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function disables On Die Termination.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_OdtDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_OdtDisable(DDR_MODULE_ID index);

PLIB_DDR_PHY_OdtEnable Function

Enables On Die Termination and sets the value.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_OdtEnable(DDR_MODULE_ID index, DDR_PHY_ODT odtVal);
```

Returns

None.

Description

This function enables On Die Termination (ODT) and sets the value.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_OdtEnable(DDR_ID_0, DDR_PHY_ODT_75_OHM);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
odtVal	The ODT value

Function

void PLIB_DDR_PHY_OdtEnable(DDR_MODULE_ID index, DDR_PHY_ODT odtVal);

PLIB_DDR_PHY_PadReceiveEnable Function

Enables pad receivers on bidirectional I/O.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_PadReceiveEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function enables pad receivers on bidirectional I/O.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_PadReceiveEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_PadReceiveEnable(DDR_MODULE_ID index);

PLIB_DDR_PHY_PreambleDlySet Function

Sets the length of the preamble for data writes.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_PreambleDlySet(DDR_MODULE_ID index, DDR_PHY_PREAMBLE_DLY preDly);
```

Returns

None.

Description

This function sets the length of the preamble for data writes.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_PreambleDlySet(DDR_ID_0, DDR_PHY_PREAMBLE_DLY_1_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
preDly	Preamble delay

Function

void PLIB_DDR_PHY_PreambleDlySet(DDR_MODULE_ID index, DDR_PHY_PREAMBLE_DLY preDly);

PLIB_DDR_PHY_ReadCASLatencySet Function

Sets the read CAS latency while running SCL test.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_ReadCASLatencySet(DDR_MODULE_ID index, uint8_t rLat);
```

Returns

None.

Description

This function sets the read CAS latency while running SCL test.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_ReadCASLatencySet(DDR_ID_0, 5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
rLat	Read CAS latency (RL)

Function

void PLIB_DDR_PHY_ReadCASLatencySet(DDR_MODULE_ID index, uint8_t rLat);

PLIB_DDR_PHY_SCLDelay Function

Disables ODT on Chip Select while running SCL test.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_SCLDelay(DDR_MODULE_ID index, DDR_PHY_SCL_DELAY sclDly);
```

Returns

None.

Description

This function disables ODT on Chip Select while running SCL test.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLDelay(DDR_ID_0, DDR_PHY_SCL_DELAY_SINGLE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
sclDly	SCL delay

Function

void PLIB_DDR_PHY_SCLDelay(DDR_MODULE_ID index, DDR_PHY_SCL_DELAY sclDly);

PLIB_DDR_PHY_SCLEnable Function

Enables SCL on the Chip Select 'cs'.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_SCLEnable(DDR_MODULE_ID index, uint8_t cs);
```

Returns

None.

Description

This function enables SCL on the Chip Select 'cs'.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLEnable(DDR_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cs	Chip select

Function

void PLIB_DDR_PHY_SCLEnable(DDR_MODULE_ID index, uint8_t cs);

PLIB_DDR_PHY_SCLTestBurstModeSet Function

Sets the burst mode of the DRAM while running SCL test.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_SCLTestBurstModeSet(DDR_MODULE_ID index, DDR_PHY_SCL_BURST_MODE brstMode);
```

Returns

None.

Description

This function sets the burst mode of the DRAM while running SCL test.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLTestBurstModeSet(DDR_ID_0, DDR_PHY_SCL_BURST_MODE_8);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
brstMode	Burst mode of the DRAM

Function

void PLIB_DDR_PHY_SCLTestBurstModeSet(DDR_MODULE_ID index, DDR_PHY_SCL_BURST_MODE brstMode);

PLIB_DDR_PHY_WriteCASLatencySet Function

Sets the write CAS latency while running SCL test.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_WriteCASLatencySet(DDR_MODULE_ID index, uint8_t wLat);
```

Returns

None.

Description

This function sets the write CAS latency while running SCL test.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_WriteCASLatencySet(DDR_ID_0, 4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
wLat	Write CAS latency (WL)

Function

void PLIB_DDR_PHY_WriteCASLatencySet(DDR_MODULE_ID index, uint8_t wLat);

PLIB_DDR_PHY_PadReceiveDisable Function

Disables pad receivers on bidirectional I/O.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_PadReceiveDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

This function disables pad receivers on bidirectional I/O.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_PadReceiveDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_PadReceiveDisable(DDR_MODULE_ID index);

PLIB_DDR_PHY_SCLCapClkDelaySet Function

Sets capture clock delay during SCL.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_SCLCapClkDelaySet(DDR_MODULE_ID index, uint8_t capDly);
```

Returns

None.

Description

This function sets capture clock delay during SCL.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLCapClkDelaySet(DDR_ID_0, 4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
capDly	Capture Clock delay

Function

void PLIB_DDR_PHY_SCLCapClkDelaySet(DDR_MODULE_ID index, uint8_t capDly);

PLIB_DDR_PHY_SCLDDRClkDelaySet Function

Sets DDR clock delay during SCL.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_SCLDDRClkDelaySet(DDR_MODULE_ID index, uint8_t ddrDly);
```

Returns

None.

Description

This function sets DDR clock delay during SCL.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLDDRClkDelaySet(DDR_ID_0, 4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
ddrDly	DDR Clock delay

Function

void PLIB_DDR_PHY_SCLDDRClkDelaySet(DDR_MODULE_ID index, uint8_t ddrDly);

PLIB_DDR_PHY_HalfRateSet Function

Sets the PHY to half rate mode.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_HalfRateSet(DDR_MODULE_ID index);
```

Returns

None.

Description

This function sets the PHY to half rate mode.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_HalfRateSet(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_HalfRateSet(DDR_MODULE_ID index);

PLIB_DDR_PHY_SCLStart Function

Runs PHY Self Calibration.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_SCLStart(DDR_MODULE_ID index);
```

Returns

None.

Description

This function runs PHY Self Calibration.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLStart(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_SCLStart(DDR_MODULE_ID index);

PLIB_DDR_PHY_SCLStatus Function

Checks status of PHY Self Calibration.

File

plib_ddr.h

C

```
bool PLIB_DDR_PHY_SCLStatus(DDR_MODULE_ID index);
```

Returns

- true - The PHY SCL has passed
- false - The PHY SCL has not passed

Description

This function checks status of PHY Self Calibration.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_SCLStatus(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_DDR_PHY_SCLStatus(DDR_MODULE_ID index);

PLIB_DDR_PHY_AddCtlDriveStrengthSet Function

Sets the drive strength for the PHY address and control pads.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_AddCtlDriveStrengthSet(DDR_MODULE_ID index, DDR_PHY_DRIVE_STRENGTH drvStr);
```

Returns

None.

Description

Sets the drive strength for the PHY address and control pads.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_AddCtlDriveStrengthSet(DDR_ID_0, DDR_PHY_DRIVE_STRENGTH_FULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
drvStr	Drive strength

Function

void PLIB_DDR_PHY_AddCtlDriveStrengthSet(DDR_MODULE_ID index, DDR_PHY_DRIVE_STRENGTH drvStr);

PLIB_DDR_PHY_DataDriveStrengthSet Function

Sets the drive strength for the PHY data pads.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_DataDriveStrengthSet(DDR_MODULE_ID index, DDR_PHY_DRIVE_STRENGTH drvStr);
```

Returns

None.

Description

Sets the drive strength for the PHY data pads.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_DataDriveStrengthSet(DDR_ID_0, DDR_PHY_DRIVE_STRENGTH_FULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
drvStr	Drive strength

Function

void PLIB_DDR_PHY_DataDriveStrengthSet(DDR_MODULE_ID index, DDR_PHY_DRIVE_STRENGTH drvStr);

PLIB_DDR_PHY_WriteCmdDelayDisable Function

Disables write command delay.

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_WriteCmdDelayDisable(DDR_MODULE_ID index);
```

Returns

None.

Description

Disables write command delay.

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_WriteCmdDelayDisable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_WriteCmdDelayDisable(DDR_MODULE_ID index);

PLIB_DDR_PHY_WriteCmdDelayEnable Function

Enables write command delay. The extra delay is needed for devices with even write latency (WL).

File

plib_ddr.h

C

```
void PLIB_DDR_PHY_WriteCmdDelayEnable(DDR_MODULE_ID index);
```

Returns

None.

Description

Enables write command delay. The extra delay is needed for devices with even write latency (WL).

Remarks

This feature is not available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_DDR_PHY_WriteCmdDelayEnable(DDR_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_DDR_PHY_WriteCmdDelayEnable(DDR_MODULE_ID index);

g) Feature Existence Functions

PLIB_DDR_ExistsAddressMapping Function

Identifies whether the AddressMapping feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsAddressMapping(DDR_MODULE_ID index);
```

Returns

- true - The AddressMapping feature is supported on the device
- false - The AddressMapping feature is not supported on the device

Description

This function identifies whether the AddressMapping feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_RowAddressSet
- PLIB_DDR_ColumnAddressSet
- PLIB_DDR_BankAddressSet
- PLIB_DDR_ChipSelectAddressSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsAddressMapping(DDR_MODULE_ID index)

PLIB_DDR_ExistsArbitrationControl Function

Identifies whether the ArbitrationControl feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsArbitrationControl(DDR_MODULE_ID index);
```

Returns

- true - The ArbitrationControl feature is supported on the device
- false - The ArbitrationControl feature is not supported on the device

Description

This function identifies whether the ArbitrationControl feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_TargetSelect
- PLIB_DDR_MinLimit
- PLIB_DDR_ReqPeriod
- PLIB_DDR_MinCommand

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsArbitrationControl(DDR_MODULE_ID index)

PLIB_DDR_ExistsAutoPowerDown Function

Identifies whether the AutoPowerDown feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsAutoPowerDown(DDR_MODULE_ID index);
```

Returns

- true - The AutoPowerDown feature is supported on the device
- false - The AutoPowerDown feature is not supported on the device

Description

This function identifies whether the AutoPowerDown feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_AutoPowerDownEnable
- PLIB_DDR_AutoPowerDownDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsAutoPowerDown(DDR_MODULE_ID index)

PLIB_DDR_ExistsAutoPrecharge Function

Identifies whether the AutoPrecharge feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsAutoPrecharge(DDR_MODULE_ID index);
```

Returns

- true - The AutoPrecharge feature is supported on the device
- false - The AutoPrecharge feature is not supported on the device

Description

This function identifies whether the AutoPrecharge feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_AutoPchrgEnable
- PLIB_DDR_AutoPchrgDisable
- PLIB_DDR_AutoPchrgPowerDownEnable
- PLIB_DDR_AutoPchrgPowerDownDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsAutoPrecharge(DDR_MODULE_ID index)

PLIB_DDR_ExistsAutoSelfRefresh Function

Identifies whether the AutoSelfRefresh feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsAutoSelfRefresh(DDR_MODULE_ID index);
```

Returns

- true - The AutoSelfRefresh feature is supported on the device
- false - The AutoSelfRefresh feature is not supported on the device

Description

This function identifies whether the AutoSelfRefresh feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_AutoSelfRefreshEnable
- PLIB_DDR_AutoSelfRefreshDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsAutoSelfRefresh(DDR_MODULE_ID index)

PLIB_DDR_ExistsDDRCommands Function

Identifies whether the DDRCommands feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsDDRCommands(DDR_MODULE_ID index);
```

Returns

- true - The DDRCommands feature is supported on the device
- false - The DDRCommands feature is not supported on the device

Description

This function identifies whether the DDRCommands feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_MaxCmdBrstCntSet
- PLIB_DDR_NumHostCmdsSet
- PLIB_DDR_CmdDataWrite
- PLIB_DDR_CmdDataValid
- PLIB_DDR_CmdDataSend
- PLIB_DDR_CmdDataIsComplete
- PLIB_DDR_ControllerEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsDDRCommands(DDR_MODULE_ID index)

PLIB_DDR_ExistsDDRControllerConfig Function

Identifies whether the DDRControllerConfig feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsDDRControllerConfig(DDR_MODULE_ID index);
```

Returns

- true - The DDRControllerConfig feature is supported on the device
- false - The DDRControllerConfig feature is not supported on the device

Description

This function identifies whether the DDRControllerConfig feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_BigEndianSet
- PLIB_DDR_LittleEndianSet
- PLIB_DDR_FullRateSet
- PLIB_DDR_HalfRateSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsDDRControllerConfig(DDR_MODULE_ID index)

PLIB_DDR_ExistsODTConfig Function

Identifies whether the ODTConfig feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsODTConfig(DDR_MODULE_ID index);
```

Returns

- true - The ODTConfig feature is supported on the device
- false - The ODTConfig feature is not supported on the device

Description

This function identifies whether the ODTConfig feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_OdtReadEnable
- PLIB_DDR_OdtReadDisable
- PLIB_DDR_OdtWriteEnable
- PLIB_DDR_OdtWriteDisable
- PLIB_DDR_OdtWriteParamSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsODTConfig(DDR_MODULE_ID index)

PLIB_DDR_ExistsPHY_DLLCalibration Function

Identifies whether the PHY_DLLCalibration feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsPHY_DLLCalibration(DDR_MODULE_ID index);
```

Returns

- true - The PHY_DLLCalibration feature is supported on the device
- false - The PHY_DLLCalibration feature is not supported on the device

Description

This function identifies whether the PHY_DLLCalibration feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_PHY_DllRecalibEnable
- PLIB_DDR_PHY_DllRecalibDisable
- PLIB_DDR_PHY_DllMasterDelayStartSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsPHY_DLLCalibration(DDR_MODULE_ID index)

PLIB_DDR_ExistsPHY_PadConfig Function

Identifies whether the PHY_PadConfig feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsPHY_PadConfig(DDR_MODULE_ID index);
```

Returns

- true - The PHY_PadConfig feature is supported on the device
- false - The PHY_PadConfig feature is not supported on the device

Description

This function identifies whether the PHY_PadConfig feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_PHY_OdtEnable
- PLIB_DDR_PHY_OdtDisable
- PLIB_DDR_PHY_DriveStrengthSet
- PLIB_DDR_PHY_OdtCal
- PLIB_DDR_PHY_DrvStrgthCal
- PLIB_DDR_PHY_ExtraClockEnable
- PLIB_DDR_PHY_ExtraClockDisable
- PLIB_DDR_PHY_InternalDllEnable
- PLIB_DDR_PHY_ExternalDllEnable
- PLIB_DDR_PHY_PadReceiveEnable
- PLIB_DDR_PHY_PreambleDlySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsPHY_PadConfig(DDR_MODULE_ID index)

PLIB_DDR_ExistsPHY_SCLConfig Function

Identifies whether the PHY_SCLConfig feature exists on the DDR module

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsPHY_SCLConfig(DDR_MODULE_ID index);
```

Returns

- true - The PHY_SCLConfig feature is supported on the device
- false - The PHY_SCLConfig feature is not supported on the device

Description

This function identifies whether the PHY_SCLConfig feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_PHY_SCLTestBurstModeSet
- PLIB_DDR_PHY_DDRTypeSet
- PLIB_DDR_PHY_ReadCASLatencySet
- PLIB_DDR_PHY_WriteCASLatencySet
- PLIB_DDR_PHY_OdtCSEnable
- PLIB_DDR_PHY_OdtCSDisable
- PLIB_DDR_PHY_SCLDelay
- PLIB_DDR_PHY_SCLEnable
- PLIB_DDR_PHY_SCLDDRClkDelaySet
- PLIB_DDR_PHY_SCLCapClkDelaySet
- PLIB_DDR_PHY_SCLStart
- PLIB_DDR_PHY_SCLStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsPHY_SCLConfig(DDR_MODULE_ID index)

PLIB_DDR_ExistsRefreshConfig Function

Identifies whether the RefreshConfig feature exists on the DDR module.

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsRefreshConfig(DDR_MODULE_ID index);
```

Returns

- true - The RefreshConfig feature is supported on the device
- false - The RefreshConfig feature is not supported on the device

Description

This function identifies whether the RefreshConfig feature is available on the DDR module. When this function returns true, this function is supported on the device:

- PLIB_DDR_RefreshConfig

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsRefreshConfig(DDR_MODULE_ID index)

PLIB_DDR_ExistsTimingDelays Function

Identifies whether the TimingDelays feature exists on the DDR module

File

plib_ddr.h

C

```
bool PLIB_DDR_ExistsTimingDelays(DDR_MODULE_ID index);
```

Returns

- true - The TimingDelays feature is supported on the device
- false - The TimingDelays feature is not supported on the device

Description

This function identifies whether the TimingDelays feature is available on the DDR module. When this function returns true, these functions are supported on the device:

- PLIB_DDR_ReadWriteDelaySet
- PLIB_DDR_SelfRefreshDelaySet
- PLIB_DDR_PowerDownDelaySet
- PLIB_DDR_PchrgDelaySet
- PLIB_DDR_AddressDelaySet
- PLIB_DDR_DataDelaySet
- PLIB_DDR_TfawDelaySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_DDR_ExistsTimingDelays(DDR_MODULE_ID index)

h) Data Types and Constants

DDR_CMD_IDLE_NOP Macro

File

plib_ddr.h

C

```
#define DDR_CMD_IDLE_NOP 0x00FFFFFF
```

Section

Constants and Data Types

DDR_PHY_DDR_TYPE Enumeration

Defines the DDR type.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_PHY_DDR_TYPE_DDR2,
  DDR_PHY_DDR_TYPE_DDR3
} DDR_PHY_DDR_TYPE;
```

Description

DDR Type

This data type defines the DDR type.

Remarks

The actual definition of this enumeration is device-specific.

DDR_CMD_LOAD_MODE Macro

File

plib_ddr.h

C

```
#define DDR_CMD_LOAD_MODE 0x00FFF001
```

Description

This is macro DDR_CMD_LOAD_MODE.

DDR_PHY_DRIVE_STRENGTH Enumeration

Defines the PHY Pad drive strength value.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_PHY_DRIVE_STRENGTH_60,
  DDR_PHY_DRIVE_STRENGTH_FULL
} DDR_PHY_DRIVE_STRENGTH;
```

Description

PHY Pad Drive Strength Value

This data type defines the PHY Pad drive strength value.

Remarks

The actual definition of this enumeration is device-specific.

DDR_CMD_PRECHARGE_ALL Macro

File

plib_ddr.h

C

```
#define DDR_CMD_PRECHARGE_ALL 0x00FFF401
```

Description

This is macro DDR_CMD_PRECHARGE_ALL.

DDR_PHY_ODT Enumeration

Defines the ODT termination value.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_PHY_ODT_75_OHM,
  DDR_PHY_ODT_150_OHM
} DDR_PHY_ODT;
```

Description

On-Die-Termination Value

This data type defines the ODT termination value.

Remarks

The actual definition of this enumeration is device-specific.

DDR_CMD_REFRESH Macro

File

plib_ddr.h

C

```
#define DDR_CMD_REFRESH 0x00FFF801
```

Description

This is macro DDR_CMD_REFRESH.

DDR_PHY_PREAMBLE_DLY Enumeration

Defines the PHY preamble delay value.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_PHY_PREAMBLE_DLY_2_0,
  DDR_PHY_PREAMBLE_DLY_1_5,
  DDR_PHY_PREAMBLE_DLY_1_0
} DDR_PHY_PREAMBLE_DLY;
```

Description

PHY Preamble Delay

This data type defines the PHY preamble delay value.

Remarks

The actual definition of this enumeration is device-specific.

DDR_PHY_SCL_BURST_MODE Enumeration

Defines the burst mode for SCL.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_PHY_SCL_BURST_MODE_4,
  DDR_PHY_SCL_BURST_MODE_8
} DDR_PHY_SCL_BURST_MODE;
```

Description

PHY Preamble Delay

This data type defines the burst mode for SCL.

Remarks

The actual definition of this enumeration is device-specific.

DDR_PHY_SCL_DELAY Enumeration

Defines the SCL delay.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_PHY_SCL_DELAY_SINGLE,
  DDR_PHY_SCL_DELAY_DOUBLE
} DDR_PHY_SCL_DELAY;
```

Description

DDR Type

This data type defines the SCL delay.

Remarks

The actual definition of this enumeration is device-specific.

DDR_HOST_CMD_REG Enumeration

Defines the DDR host command register.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_HOST_CMD_REG_100,
  DDR_HOST_CMD_REG_101,
  DDR_HOST_CMD_REG_102,
  DDR_HOST_CMD_REG_103,
  DDR_HOST_CMD_REG_104,
  DDR_HOST_CMD_REG_105,
  DDR_HOST_CMD_REG_106,
  DDR_HOST_CMD_REG_107,
  DDR_HOST_CMD_REG_108,
  DDR_HOST_CMD_REG_109,
  DDR_HOST_CMD_REG_110,
  DDR_HOST_CMD_REG_111,
  DDR_HOST_CMD_REG_112,
  DDR_HOST_CMD_REG_113,
  DDR_HOST_CMD_REG_114,
  DDR_HOST_CMD_REG_115,
  DDR_HOST_CMD_REG_200,
  DDR_HOST_CMD_REG_201,
  DDR_HOST_CMD_REG_202,
  DDR_HOST_CMD_REG_203,
  DDR_HOST_CMD_REG_204,
  DDR_HOST_CMD_REG_205,
  DDR_HOST_CMD_REG_206,
  DDR_HOST_CMD_REG_207,
  DDR_HOST_CMD_REG_208,
  DDR_HOST_CMD_REG_209,
  DDR_HOST_CMD_REG_210,
  DDR_HOST_CMD_REG_211,
  DDR_HOST_CMD_REG_212,
  DDR_HOST_CMD_REG_213,
  DDR_HOST_CMD_REG_214,
  DDR_HOST_CMD_REG_215
} DDR_HOST_CMD_REG;
```

Description

DDR Host Command Register

this data type defines the DDR host command register.

Remarks

The actual definition of this enumeration is device-specific.

DDR_MODULE_ID Enumeration

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_ID_1,
  DDR_NUMBER_OF_MODULES
} DDR_MODULE_ID;
```

Members

Members	Description
DDR_ID_1	DDR 1
DDR_NUMBER_OF_MODULES	Total number of DDR modules available

Description

Enumeration: DDR_MODULE_ID

This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on Microchip microcontrollers.

Refer to the data sheet to get the correct number of modules defined for desired microcontroller.

DDR_TARGET Enumeration

Defines the different targets to which the DDR controller is connected.

File

plib_ddr_help.h

C

```
typedef enum {
  DDR_TARGET_CPU,
  DDR_TARGET_GC_IN,
  DDR_TARGET_GC_OUT,
  DDR_TARGET_GPU_IN,
  DDR_TARGET_GPU_OUT
} DDR_TARGET;
```

Description

Targets

This data type defines the different targets to which the DDR controller is connected.

Remarks

The actual definition of this enumeration is device-specific.

Files

Files

Name	Description
plib_ddr.h	Defines the DDR Peripheral Library Interface.

Description

This section lists the source and header files used by the library.

plib_ddr.h

Defines the DDR Peripheral Library Interface.

Functions

	Name	Description
	PLIB_DDR_AutoPchrgDisable	Prevents the DDR controller from issuing an auto-precharge command to close the bank at the end of every user command.
	PLIB_DDR_AutoPchrgEnable	Enables the DDR controller to issue an auto-precharge command to close the bank at the end of every user command.
	PLIB_DDR_AutoPchrgPowerDownDisable	Prevents the DDR controller from automatically entering Precharge Power-down mode.
	PLIB_DDR_AutoPchrgPowerDownEnable	Enables the DDR controller to automatically enter Precharge Power-down mode.
	PLIB_DDR_AutoPowerDownDisable	Prevents the DDR controller from automatically entering Power-down mode.
	PLIB_DDR_AutoPowerDownEnable	Enables the DDR controller to automatically enter Power-down mode.
	PLIB_DDR_AutoSelfRefreshDisable	Prevents the DDR controller from automatically entering Self-refresh mode.
	PLIB_DDR_AutoSelfRefreshEnable	Enables the DDR controller to automatically enter Self-refresh mode.
	PLIB_DDR_BankAddressSet	Initializes the DDR controller memory configuration registers for bank address.
	PLIB_DDR_BigEndianSet	Sets the DDR data endianness to big.
	PLIB_DDR_ChipSelectAddressSet	Initializes the DDR controller memory configuration registers for Chip Select address.
	PLIB_DDR_CmdDataIsComplete	Returns the value of the valid bit in the DDR2CMDISSUE register. This bit is cleared by hardware, when the SDRAM initialization data has been transmitted.
	PLIB_DDR_CmdDataSend	Transmits the data in the command registers to the SDRAM.
	PLIB_DDR_CmdDataValid	Indicates to the controller that the data in the command registers is valid.
	PLIB_DDR_CmdDataWrite	Writes an SDRAM command word to a command register in the controller.
	PLIB_DDR_ColumnAddressSet	Initializes the DDR controller memory configuration registers for the column address.
	PLIB_DDR_ControllerEnable	Enables the controller for normal operations after SDRAM is initialized.
	PLIB_DDR_DataDelaySet	Initializes the DDR controller with the data delays needed for the specific DDR in use.
	PLIB_DDR_ExistsAddressMapping	Identifies whether the AddressMapping feature exists on the DDR module.
	PLIB_DDR_ExistsArbitrationControl	Identifies whether the ArbitrationControl feature exists on the DDR module.
	PLIB_DDR_ExistsAutoPowerDown	Identifies whether the AutoPowerDown feature exists on the DDR module.
	PLIB_DDR_ExistsAutoPrecharge	Identifies whether the AutoPrecharge feature exists on the DDR module.
	PLIB_DDR_ExistsAutoSelfRefresh	Identifies whether the AutoSelfRefresh feature exists on the DDR module.
	PLIB_DDR_ExistsDDRCommands	Identifies whether the DDRCommands feature exists on the DDR module.
	PLIB_DDR_ExistsDDRControllerConfig	Identifies whether the DDRControllerConfig feature exists on the DDR module.
	PLIB_DDR_ExistsODTConfig	Identifies whether the ODTConfig feature exists on the DDR module.
	PLIB_DDR_ExistsPHY_DLLCalibration	Identifies whether the PHY_DLLCalibration feature exists on the DDR module.
	PLIB_DDR_ExistsPHY_PadConfig	Identifies whether the PHY_PadConfig feature exists on the DDR module.
	PLIB_DDR_ExistsPHY_SCLConfig	Identifies whether the PHY_SCLConfig feature exists on the DDR module
	PLIB_DDR_ExistsRefreshConfig	Identifies whether the RefreshConfig feature exists on the DDR module.
	PLIB_DDR_ExistsTimingDelays	Identifies whether the TimingDelays feature exists on the DDR module
	PLIB_DDR_FullRateSet	Sets the DDR controller to Full-rate mode.
	PLIB_DDR_HalfRateSet	Sets the DDR controller to Half-rate mode.
	PLIB_DDR_LittleEndianSet	Sets the DDR data endianness to little.
	PLIB_DDR_MaxCmdBrstCntSet	Sets the maximum number of commands that can be written to the controller in Burst mode.
	PLIB_DDR_MaxPendingRefSet	Initializes the DDR controller refresh configuration.
	PLIB_DDR_MinCommand	Sets the minimum number of bursts to be serviced for a DDR target within (REQPER * 4) number of clocks.
	PLIB_DDR_MinLimit	Sets the minimum number of bursts for a DDR target.
	PLIB_DDR_NumHostCmdsSet	Sets the number of commands to be transmitted to the SDRAM.
	PLIB_DDR_OdtReadDisable	Selects which Chip Select to disable ODT for data reads.
	PLIB_DDR_OdtReadEnable	Selects which Chip Select to enable ODT for data reads.
	PLIB_DDR_OdtReadParamSet	Sets the ODT parameters for data reads.
	PLIB_DDR_OdtWriteDisable	Selects which Chip Select to disable ODT for data writes.
	PLIB_DDR_OdtWriteEnable	Selects which Chip Select to enable ODT for data writes.
	PLIB_DDR_OdtWriteParamSet	Sets the ODT parameters for data writes.
	PLIB_DDR_PHY_AddCtlDriveStrengthSet	Sets the drive strength for the PHY address and control pads.
	PLIB_DDR_PHY_DataDriveStrengthSet	Sets the drive strength for the PHY data pads.
	PLIB_DDR_PHY_DDRTypeSet	Sets the DRAM type for the PHY.

	PLIB_DDR_PHY_DllMasterDelayStartSet	Sets the start value of the digital DLL master delay line.
	PLIB_DDR_PHY_DllRecalibDisable	Disables periodic recalibration of the internal digital DLL.
	PLIB_DDR_PHY_DllRecalibEnable	Enables periodic recalibration of the internal digital DLL, and sets the recalibration period.
	PLIB_DDR_PHY_DriveStrengthSet	Disables On Die Termination.
	PLIB_DDR_PHY_DrvStrgthCal	Calibrates the pad NFET and PFET output impedance.
	PLIB_DDR_PHY_ExternalDllEnable	Enables the use of an external DLL.
	PLIB_DDR_PHY_ExtraClockDisable	Does not enable the drive pad for an extra clock cycle after a write burst.
	PLIB_DDR_PHY_ExtraClockEnable	Enables the drive pad for an extra clock cycle after a write burst.
	PLIB_DDR_PHY_HalfRateSet	Sets the PHY to half rate mode.
	PLIB_DDR_PHY_InternalDllEnable	Enables the use of the internal digital DLL.
	PLIB_DDR_PHY_OdtCal	Calibrates the pull-up and pull-down ODT impedance.
	PLIB_DDR_PHY_OdtCSDisable	Disables ODT on Chip Select while running SCL test.
	PLIB_DDR_PHY_OdtCSEnable	Enables ODT on Chip Select while running SCL test.
	PLIB_DDR_PHY_OdtDisable	Disables On Die Termination.
	PLIB_DDR_PHY_OdtEnable	Enables On Die Termination and sets the value.
	PLIB_DDR_PHY_PadReceiveDisable	Disables pad receivers on bidirectional I/O.
	PLIB_DDR_PHY_PadReceiveEnable	Enables pad receivers on bidirectional I/O.
	PLIB_DDR_PHY_PreambleDlySet	Sets the length of the preamble for data writes.
	PLIB_DDR_PHY_ReadCASLatencySet	Sets the read CAS latency while running SCL test.
	PLIB_DDR_PHY_SCLCapClkDelaySet	Sets capture clock delay during SCL.
	PLIB_DDR_PHY_SCLDDRClkDelaySet	Sets DDR clock delay during SCL.
	PLIB_DDR_PHY_SCLDelay	Disables ODT on Chip Select while running SCL test.
	PLIB_DDR_PHY_SCLEnable	Enables SCL on the Chip Select 'cs'.
	PLIB_DDR_PHY_SCLStart	Runs PHY Self Calibration.
	PLIB_DDR_PHY_SCLStatus	Checks status of PHY Self Calibration.
	PLIB_DDR_PHY_SCLTestBurstModeSet	Sets the burst mode of the DRAM while running SCL test.
	PLIB_DDR_PHY_WriteCASLatencySet	Sets the write CAS latency while running SCL test.
	PLIB_DDR_PHY_WriteCmdDelayDisable	Disables write command delay.
	PLIB_DDR_PHY_WriteCmdDelayEnable	Enables write command delay. The extra delay is needed for devices with even write latency (WL).
	PLIB_DDR_PowerDownDelaySet	Initializes the DDR controller with the power down delays needed for the specific DDR in use.
	PLIB_DDR_PrechargAllBanksSet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_PrechargeToRASDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_RASToCASDelaySet	Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.
	PLIB_DDR_RASToPrechargeDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_RASToRASBankDelaySet	Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.
	PLIB_DDR_RASToRASDelaySet	Initializes the DDR controller with the row/column address delays needed for the specific DDR in use.
	PLIB_DDR_ReadReadDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.
	PLIB_DDR_ReadToPrechargeDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_ReadWriteDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.
	PLIB_DDR_RefreshTimingSet	Initializes the DDR controller refresh configuration.
	PLIB_DDR_ReqPeriod	Sets the timeout for MINCMD number of bursts to be serviced for a DDR target.
	PLIB_DDR_RowAddressSet	Initializes the DDR controller memory configuration registers for row address.
	PLIB_DDR_SelfRefreshDelaySet	Initializes the DDR controller with the self-refresh delays needed for the specific DDR in use.
	PLIB_DDR_TfawDelaySet	Initializes the DDR controller with the four-bank activation window needed for the specific DDR in use.

	PLIB_DDR_WriteReadDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.
	PLIB_DDR_WriteToPrechargeDelaySet	Initializes the DDR controller with the read/write precharge delays needed for the specific DDR in use.
	PLIB_DDR_WriteWriteDelaySet	Initializes the DDR controller with the read/write delays needed for the specific DDR in use.

Macros

	Name	Description
	DDR_CMD_IDLE_NOP	
	DDR_CMD_LOAD_MODE	This is macro DDR_CMD_LOAD_MODE.
	DDR_CMD_PRECHARGE_ALL	This is macro DDR_CMD_PRECHARGE_ALL.
	DDR_CMD_REFRESH	This is macro DDR_CMD_REFRESH.

Description

Dual Data Rate (DDR) SDRAM Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the DDR Peripheral Library for Microchip microcontrollers. The definitions in this file are for the DDR module.

File Name

plib_ddr.h

Company

Microchip Technology Inc.

EBI Peripheral Library

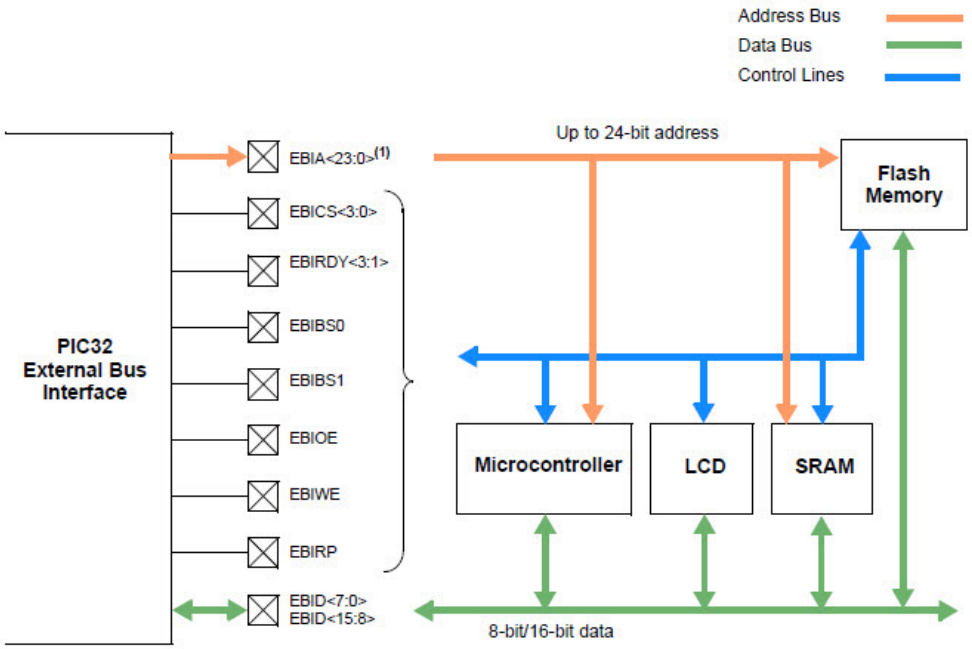
This section describes the External Bus Interface (EBI) Peripheral Library.

Introduction

This library provides a low-level abstraction of the EBI Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The External Bus Interface (EBI) allows for external physical memory to be mapped into the kernels virtual memory map. The External Bus Interface (EBI) module provides a high-speed, convenient way to interface external parallel memory devices to the PIC32 family device. With the EBI module, it is possible to connect asynchronous SRAM and NOR Flash devices, as well as non-memory devices such as camera sensors. The module also supports Low-Cost Controllerless (LCC) Graphics devices.



Note 1: No EBIA address pins are available on 64-pin devices and the EBIA<23:20> address pins are only available on 144-pin devices.

Using the Library

This topic describes the basic architecture of the EBI Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_ebi.h`

The interface to the EBI Peripheral library is defined in the plib_ebi.h header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the EBI Peripheral library must include `peripheral.h`.

Library File:

The EBI Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the Peripheral interacts with the framework.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the EBI module.

Library Interface Section	Description
Get Functions	These functions can be used to return the status of a feature.
Set Functions	These functions can be used to configure features.
Feature Existence Functions	Lists the interface routines that describe whether or not the feature is supported by the device.

How the Library Works

Before using the EBI module, it must be enabled by the application developer(s). To work correctly, the device characteristics of the attached Memory/Flash must be set with the functions in this library. The Characteristics of the Memory/Flash will be found in the specific device data sheet.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

Configuring the Library

The library is configured for the supported EBI module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Get Functions

	Name	Description
	PLIB_EBI_AddressHoldTimeGet	Returns the address hold time.
	PLIB_EBI_AddressSetupTimeGet	Returns the setup hold time.
	PLIB_EBI_BaseAddressGet	Returns the base address set for each Chip Select.
	PLIB_EBI_ControlEnableGet	Returns the status of the EBI enable bit.
	PLIB_EBI_DataTurnAroundTimeGet	Returns the data turn-around time set for the EBI bus.
	PLIB_EBI_FlashPowerDownModeGet	Returns the set power-down state.
	PLIB_EBI_FlashResetPinGet	Sets the control use of Flash Reset pin.
	PLIB_EBI_PageModeGet	Returns the Paging mode settings.
	PLIB_EBI_PageReadCycleTimeGet	Returns the cycle time for a page read.
	PLIB_EBI_ReadCycleTimeGet	Returns the read time in the number of clock cycles.
	PLIB_EBI_ReadyModeGet	Returns whether or not Ready mode was set.
	PLIB_EBI_StaticMemoryWidthRegisterGet	Returns the set width of the data bus.
	PLIB_EBI_WritePulseWidthGet	Returns the set hold time in clock cycles.
	PLIB_EBI_MemoryPageSizeGet	Returns the Paging mode settings.
	PLIB_EBI_FlashTimingGet	Returns the set Flash timing for external Flash.

b) Set Functions

	Name	Description
	PLIB_EBI_AddressPinEnableBitsSet	Sets the address pins used for EBI.
	PLIB_EBI_BaseAddressSet	Sets the base address for physical memory at each Chip Select pin.
	PLIB_EBI_ByteSelectPinSet	Sets the data Byte Select High <15:8> and Low <7:0> enable pins for use.
	PLIB_EBI_ChipSelectEnableSet	Sets the Chip Select pins for use with the EBI or GPIO.
	PLIB_EBI_DataEnableSet	Sets the use of Data Byte Select Pins, High and Low, for control with EBI or GPIO.
	PLIB_EBI_FlashPowerDownModeSet	Sets the pin state for Flash devices on Power-down and Reset.
	PLIB_EBI_FlashResetPinSet	Sets the control use of the Flash Reset pin.
	PLIB_EBI_FlashTimingSet	Sets the timing for hold in reset for external Flash.
	PLIB_EBI_MemoryCharacteristicsSet	Sets the characteristics for memory or attached devices attached to the specified pin.
	PLIB_EBI_MemoryPagingSet	Sets the size of the memory page if paging is used.
	PLIB_EBI_MemoryTimingConfigSet	Sets the cycle time for page reading.
	PLIB_EBI_ReadyPin1ConfigSet	Sets the control use of ReadyPin1 and inverts the status.
	PLIB_EBI_ReadyPin2ConfigSet	Sets the control use of ReadyPin2 and inverts the status.
	PLIB_EBI_ReadyPin3ConfigSet	Sets the control use of ReadyPin3 and inverts the status.
	PLIB_EBI_ReadyPinSensSet	Sets the sensitivity of the Ready pin.
	PLIB_EBI_StaticMemoryWidthRegisterSet	Sets the data width of static memory.
	PLIB_EBI_WriteOutputControlSet	Sets the Write Enable and Output Enable control pins.
	PLIB_EBI_ControlEnableSet	Sets the EBI bus ON/OFF enable bit.
	PLIB_EBI_ReadyModeSet	Sets the use of Ready mode for each pin.

c) Feature Existence Functions

	Name	Description
	PLIB_EBI_ExistsAddressHoldTime	Identifies whether the AddressHoldTime feature exists on the EBI module.
	PLIB_EBI_ExistsAddressPinEnableBits	Identifies whether the AddressPinEnableBits feature exists on the EBI module.
	PLIB_EBI_ExistsAddressSetupTime	Identifies whether the AddressSetupTime feature exists on the EBI module.
	PLIB_EBI_ExistsBaseAddress	Identifies whether the Base_Address feature exists on the EBI module.
	PLIB_EBI_ExistsByteSelectPin	Identifies whether the ByteSelectPin feature exists on the EBI module.
	PLIB_EBI_ExistsChipSelectEnable	Identifies whether the ChipSelectEnable feature exists on the EBI module.
	PLIB_EBI_ExistsControlEnable	Identifies whether the ControlEnable feature exists on the EBI module.
	PLIB_EBI_ExistsDataEnable	Identifies whether the DataEnable feature exists on the EBI module.
	PLIB_EBI_ExistsDataTurnAroundTime	Identifies whether the DataTurnAroundTime feature exists on the EBI module.
	PLIB_EBI_ExistsFlashPowerDownMode	Identifies whether the FlashPowerDownMode feature exists on the EBI module.

	PLIB_EBI_ExistsFlashResetPin	Identifies whether the FlashResetPin feature exists on the EBI module.
	PLIB_EBI_ExistsFlashTiming	Identifies whether the FlashTiming feature exists on the EBI module.
	PLIB_EBI_ExistsMemoryCharacteristics	Identifies whether the MemoryCharacteristics feature exists on the EBI module.
	PLIB_EBI_ExistsMemoryPaging	Identifies whether the MemoryPaging feature exists on the EBI module.
	PLIB_EBI_ExistsMemoryTimingConfig	Identifies whether the MemoryTimingConfig feature exists on the EBI module.
	PLIB_EBI_ExistsPageMode	Identifies whether the PageMode feature exists on the EBI module.
	PLIB_EBI_ExistsPageReadTime	Identifies whether the PageReadTime feature exists on the EBI module.
	PLIB_EBI_ExistsReadCycleTime	Identifies whether the ReadCycleTime feature exists on the EBI module.
	PLIB_EBI_ExistsReadyMode	Identifies whether the ReadyMode feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPin1Config	Identifies whether the ReadyPin1Config feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPin2Config	Identifies whether the ReadyPin2Config feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPin3Config	Identifies whether the ReadyPin3Config feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPinSens	Identifies whether the ReadyPinSens feature exists on the EBI module
	PLIB_EBI_ExistsStaticMemoryWidthRegister	Identifies whether the StaticMemoryWidthRegister feature exists on the EBI module.
	PLIB_EBI_ExistsWriteOutputControl	Identifies whether the WriteOutputControl feature exists on the EBI module.
	PLIB_EBI_ExistsWritePulseWidth	Identifies whether the WritePulseWidth feature exists on the EBI module.

d) Data Types and Constants

	Name	Description
	EBI_ADDRESS_PORT	Selects the EBI address port.
	EBI_CS_TIMING	Selects the timing register set for Chip Select.
	EBI_MEMORY_SIZE	Selects the memory size for Chip Select.
	EBI_MEMORY_TYPE	Selects the memory type for Chip Select.
	EBI_MODULE_ID	Identifies the supported EBI modules.
	EBI_PAGE_SIZE	Selects the page size for the page mode device.
	EBI_STATIC_MEMORY_WIDTH	Selects the static memory width for the register EBISMTx.

Description

This section describes the Application Programming Interface (API) functions of the EBI Peripheral Library.

Refer to each section for a detailed description.

a) Get Functions

PLIB_EBI_AddressHoldTimeGet Function

Returns the address hold time.

File

plib_ebi.h

C

```
int PLIB_EBI_AddressHoldTimeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an integer, which is the Address and Data Hold time in the number of clocks.

Description

This function returns the address and data hold time.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_AddressHoldTimeGet (EBI_MODULE_ID index,int ChipSelectNumber)

PLIB_EBI_AddressSetupTimeGet Function

Returns the setup hold time.

File

plib_ebi.h

C

```
int PLIB_EBI_AddressSetupTimeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an integer, which is the address setup time in the number of clocks.

Description

This function returns the setup hold time. A value of '0' is only valid in the case of SSRAM.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_AddressSetupTimeGet (EBI_MODULE_ID index,int ChipSelectNumber)

PLIB_EBI_BaseAddressGet Function

Returns the base address set for each Chip Select.

File

plib_ebi.h

C

```
uint32_t PLIB_EBI_BaseAddressGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an unsigned integer that is the physical address of the attached device.

Description

This function returns the base address for each Chip Select pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies which Chip Select address pin address is being read

Function

PLIB_EBI_BaseAddressGet (EBI_MODULE_ID index, int ChipSelectNumber)

PLIB_EBI_ControlEnableGet Function

Returns the status of the EBI enable bit.

File

plib_ebi.h

C

```
bool PLIB_EBI_ControlEnableGet(EBI_MODULE_ID index);
```

Returns

Boolean:

- 1 = The EBI module is ON
- 0 = The EBI module is OFF

Description

This function returns the status of the EBI enable bit.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ControlEnableGet (EBI_MODULE_ID index)

PLIB_EBI_DataTurnAroundTimeGet Function

Returns the data turn-around time set for the EBI bus.

File

plib_ebi.h

C

```
int PLIB_EBI_DataTurnAroundTimeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an integer, which is the number of clock cycles that is set in the turn-around register.

Description

This function returns the data turn-around time set for the EBI Bus. Returns the clock cycles (0-7) for static memory between read-to-write, write-to-read, and read-to-read when Chip Select changes.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_DataTurnAroundTimeGet (EBI_MODULE_ID index,int ChipSelectNumber)

PLIB_EBI_FlashPowerDownModeGet Function

Returns the set power-down state.

File

plib_ebi.h

C

```
bool PLIB_EBI_FlashPowerDownModeGet(EBI_MODULE_ID index);
```

Returns

A bool, depending on the setting of PLIB_EBI_FlashPowerDownModeSet.

Description

This function returns the set power-down state.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_FlashPowerDownModeGet (EBI_MODULE_ID index)

PLIB_EBI_FlashResetPinGet Function

Sets the control use of Flash Reset pin.

File

plib_ebi.h

C

```
bool PLIB_EBI_FlashResetPinGet(EBI_MODULE_ID index);
```

Returns

Returns a Boolean.

Description

This function set the control use of the Flash Reset pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_FlashResetPinGet (EBI_MODULE_ID index)

PLIB_EBI_PageModeGet Function

Returns the Paging mode settings.

File

plib_ebi.h

C

```
bool PLIB_EBI_PageModeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

- 1 = Page Mode is used
- 0 = Page Mode is not used

Description

This function returns the state of the register PAGEMODE.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_PageModeGet (EBI_MODULE_ID index,int ChipSelectNumber)

PLIB_EBI_PageReadCycleTimeGet Function

Returns the cycle time for a page read.

File

plib_ebi.h

C

```
int PLIB_EBI_PageReadCycleTimeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an integer, which is the number of clock cycles used to read a memory page.

Description

This function returns the cycle time for a page read. Read cycle time = TPRC + 1cycle. The value set and return are the same, the controller performs the read on the next clock, which is why there is a +1.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_PageReadCycleTimeGet (EBI_MODULE_ID index,int ChipSelectNumber)

PLIB_EBI_ReadCycleTimeGet Function

Returns the read time in the number of clock cycles.

File

plib_ebi.h

C

```
int PLIB_EBI_ReadCycleTimeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an integer, which is the read cycle time in system clocks.

Description

This function returns the read time in number of clock cycles. Read Cycle time = TRC +1 clock cycle. The controller will always take one extra clock (the next clock) for a Read Cycle. The return will be the time set by the user.

Setting a Read Cycle time to a '0' will return a '0' when read, but the controller will add '1' to that value for the next available clock.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies which Chip Select number to which the device is attached

Function

PLIB_EBI_ReadCycleTimeGet (EBI_MODULE_ID index, int ChipSelectNumber)

PLIB_EBI_ReadyModeGet Function

Returns whether or not Ready mode was set.

File

plib_ebi.h

C

```
bool PLIB_EBI_ReadyModeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

This function returns a bool.

- 1 = Ready input is used
- 0 = Ready input is not used

Description

This function returns the state of the register RDYMODE.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_ReadyModeGet (EBI_MODULE_ID index, int ChipSelectNumber)

PLIB_EBI_StaticMemoryWidthRegisterGet Function

Returns the set width of the data bus.

File

plib_ebi.h

C

```
EBI_STATIC_MEMORY_WIDTH PLIB_EBI_StaticMemoryWidthRegisterGet(EBI_MODULE_ID index, int RegisterNumber);
```

Returns

An enum type, which is the bus mode in use.

Description

This function returns the set width of the data bus.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_StaticMemoryWidthRegisterGet (EBI_MODULE_ID index,
int RegisterNumber)

PLIB_EBI_WritePulseWidthGet Function

Returns the set hold time in clock cycles.

File

plib_ebi.h

C

```
int PLIB_EBI_WritePulseWidthGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Returns an integer, which is the number of clock cycles to which the write pulse width is set.

Description

This function returns the set hold time in clock cycles. Write Pulse with = TWP + 1 clock cycle.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_WritePulseWidthGet (EBI_MODULE_ID index,int ChipSelectNumber)

PLIB_EBI_MemoryPageSizeGet Function

Returns the Paging mode settings.

File

plib_ebi.h

C

```
EBI_PAGE_SIZE PLIB_EBI_MemoryPageSizeGet(EBI_MODULE_ID index, int ChipSelectNumber);
```

Returns

Size of the memory page.

Description

This function returns the state of the register PAGESIZE.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies the specific Chip Select pin

Function

PLIB_EBI_MemoryPageSizeGet (EBI_MODULE_ID index, int ChipSelectNumber)

PLIB_EBI_FlashTimingGet Function

Returns the set Flash timing for external Flash.

File

plib_ebi.h

C

```
int PLIB_EBI_FlashTimingGet(EBI_MODULE_ID index);
```

Returns

Returns an integer, which is the number of clock cycles.

Description

This function returns the set number of clock cycles to hold the external Flash memory in reset.

Remarks

None.

Preconditions

MemoryType must be set to Flash.

Function

PLIB_EBI_FlashTimingGet (EBI_MODULE_ID index)

b) Set Functions

PLIB_EBI_AddressPinEnableBitsSet Function

Sets the address pins used for EBI.

File

plib_ebi.h

C

```
void PLIB_EBI_AddressPinEnableBitsSet(EBI_MODULE_ID index, EBI_ADDRESS_PORT AddressPort);
```

Returns

None.

Description

This function sets the address pins used for EBI.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
AddressPort	Identifies how many pins are used for addressing on the EBI bus

Function

PLIB_EBI_AddressPinEnableBitsSet (EBI_MODULE_ID index,
EBI_ADDRESS_PORT AddressPort)

PLIB_EBI_BaseAddressSet Function

Sets the base address for physical memory at each Chip Select pin.

File

plib_ebi.h

C

```
void PLIB_EBI_BaseAddressSet(EBI_MODULE_ID index, int ChipSelectNumber, uint32_t BaseAddress);
```

Returns

None.

Description

This function sets the base address for physical memory at each Chip Select pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies which Chip Select address pin is being assigned an address
BaseAddress	A physical address for memory

Function

PLIB_EBI_BaseAddressSet(EBI_MODULE_ID index, int ChipSelectNumber,
uint32_t BaseAddress)

PLIB_EBI_ByteSelectPinSet Function

Sets the data Byte Select High <15:8> and Low <7:0> enable pins for use.

File

plib_ebi.h

C

```
void PLIB_EBI_ByteSelectPinSet(EBI_MODULE_ID index, bool ByteSelect0, bool ByteSelect1);
```

Returns

None.

Description

This function sets the data Byte Select High <15:8> and Low <7:0> enable pins for use. If the system uses Byte Select High/Low pins for the data, the pins must be enabled for use in the EBI controller.

Remarks

None.

Preconditions

The EBIDEN0 and EBIDEN1 registers must first be enabled, which is done using the function PLIB_EBI_DataEnableSet.

Parameters

Parameters	Description
index	Identifier for the device instance
ByteSelect0	Identifies the Lower Byte Select Pin for enabling

ByteSelect1	Identifies the Upper Byte Select Pin for enabling

Function

PLIB_EBI_ByteSelectPinSet (EBI_MODULE_ID index, bool ByteSelect0, bool ByteSelect1)

PLIB_EBI_ChipSelectEnableSet Function

Sets the Chip Select pins for use with the EBI or GPIO.

File

plib_ebi.h

C

```
void PLIB_EBI_ChipSelectEnableSet(EBI_MODULE_ID index, bool ChipSelect0, bool ChipSelect1, bool
ChipSelect2, bool ChipSelect3);
```

Returns

None.

Description

This functions sets which Chip Select pins to use with the EBI or GPIO.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelect0	Identifies control of Chip Select 0 for enabling
ChipSelect1	Identifies control of Chip Select 1 for enabling
ChipSelect2	Identifies control of Chip Select 2 for enabling
ChipSelect3	Identifies control of Chip Select 3 for enabling

Function

PLIB_EBI_ChipSelectEnableSet (EBI_MODULE_ID index, bool ChipSelect0,
bool ChipSelect1, bool ChipSelect2, bool ChipSelect3)

PLIB_EBI_DataEnableSet Function

Sets the use of Data Byte Select Pins, High and Low, for control with EBI or GPIO.

File

plib_ebi.h

C

```
void PLIB_EBI_DataEnableSet(EBI_MODULE_ID index, bool DataUpperByte, bool DataLowerByte);
```

Returns

None.

Description

This function sets the use of the Data Byte Select Pins, High and Low, for control with EBI or GPIO.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
DataUpperByte	Identifies control of Upper Data Byte for enabling
DataLowerByte	Identifies control of Lower Data Byte for enabling

Function

PLIB_EBI_DataEnableSet (EBI_MODULE_ID index, bool DataUpperByte, bool DataLowerByte)

PLIB_EBI_FlashPowerDownModeSet Function

Sets the pin state for Flash devices on Power-down and Reset.

File

plib_ebi.h

C

```
void PLIB_EBI_FlashPowerDownModeSet(EBI_MODULE_ID index, bool FlashPowerDownMode);
```

Returns

None.

Description

This function sets the pin state for Flash devices upon a Power-down and Reset.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
FlashPowerDownMode	A bool, which sets the power-down state for Flash memory

Function

PLIB_EBI_FlashPowerDownModeSet (EBI_MODULE_ID index, bool FlashPowerDownMode)

PLIB_EBI_FlashResetPinSet Function

Sets the control use of the Flash Reset pin.

File

plib_ebi.h

C

```
void PLIB_EBI_FlashResetPinSet(EBI_MODULE_ID index, bool FlashResetPin);
```

Returns

None.

Description

This function sets the control use of the Flash Reset pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
FlashReadPin	Flash Reset pin is used

Function

PLIB_EBI_FlashResetPinSet (EBI_MODULE_ID index, bool FlashReadPin)

PLIB_EBI_FlashTimingSet Function

Sets the timing for hold in reset for external Flash.

File

plib_ebi.h

C

```
void PLIB_EBI_FlashTimingSet(EBI_MODULE_ID index, int FlashTiming);
```

Returns

None.

Description

This function sets the number of clock cycles to hold the external Flash memory in reset.

Remarks

None.

Preconditions

NOR Flash must be used with EBI instead of SRAM.

Parameters

Parameters	Description
index	Identifier for the device instance
FlashTiming	An integer, which is the number of clock cycles

Function

PLIB_EBI_FlashTimingSet (EBI_MODULE_ID index, int FlashTiming)

PLIB_EBI_MemoryCharacteristicsSet Function

Sets the characteristics for memory or attached devices attached to the specified pin.

File

plib_ebi.h

C

```
void PLIB_EBI_MemoryCharacteristicsSet(EBI_MODULE_ID index, int ChipSelectNumber, EBI_MEMORY_TYPE
MemoryType, EBI_MEMORY_SIZE MemorySize, EBI_CS_TIMING TimingReg);
```

Returns

None.

Description

This function sets the characteristics for memory or attached devices attached to the specified pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Identifies which Chip Select pin is used
MemoryType	Identifies which memory is used
MemorySize	An enum type, which sets the size of the attached memory device
TimingRegSet	Identifies the timing register

Function

PLIB_EBI_MemoryCharacteristicsSet(EBI_MODULE_ID index, int ChipSelectNumber,
EBI_MEMORY_TYPE MemoryType,
EBI_MEMORY_SIZE MemorySize,
EBI_CS_TIMING TimingReg)

PLIB_EBI_MemoryPagingSet Function

Sets the size of the memory page if paging is used.

File

plib_ebi.h

C

```
void PLIB_EBI_MemoryPagingSet(EBI_MODULE_ID index, int ChipSelectNumber, bool PageMode, EBI_PAGE_SIZE
MemoryPageSize);
```

Returns

- 1 = Device supports Page mode
- 0 = Device does not support Page mode

Description

This function sets the size of the memory page if paging is used.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ChipSelectNumber	Sets for that Chip Select pin <3:0>
PageMode	Enable or disable Page mode
MemoryPageSize	Size of memory pages

Function

PLIB_EBI_MemoryPagingSet (EBI_MODULE_ID index, int ChipSelectNumber,
bool PageMode, EBI_PAGE_SIZE MemoryPageSize)

PLIB_EBI_MemoryTimingConfigSet Function

Sets the cycle time for page reading.

File

plib_ebi.h

C

```
void PLIB_EBI_MemoryTimingConfigSet(EBI_MODULE_ID index, int CS_Timing_Reg, int PageReadTime, int
DataTurnAroundTime, int WritePulseWidth, int AddressHoldTime, int AddressSetupTime, int ReadCycleTime);
```

Returns

None.

Description

This function sets the cycle time for page reading.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
CS_Timing_Reg	Identifies which Chip Select Timing register to use EBISMT<0:2>
PageReadTime	The clock cycle needed for a Memory Page read
DataTurnAroundTime	Time between read-to-write, write-to-read, and read-to-read when Chip Select changes state.
WritePulseWidth	The clock cycles needed for a memory write
AddressHoldTime	The clock time needed for the address bus to hold
AddressSetupTime	The time needed for the address to settle
ReadCycleTime	_nt_

Function

PLIB_EBI_MemoryTimingConfigSet(EBI_MODULE_ID index, int CS_Timing_Reg,
int PageReadTime,
int DataTurnAroundTime,
int WritePulseWidth,
int AddressHoldTime,
int AddressSetupTime,
int ReadCycleTime)

PLIB_EBI_ReadyPin1ConfigSet Function

Sets the control use of ReadyPin1 and inverts the status.

File

plib_ebi.h

C

```
void PLIB_EBI_ReadyPin1ConfigSet(EBI_MODULE_ID index, bool ReadyPin1Enable, bool ReadyPin1Invert);
```

Returns

None.

Description

This function sets the control use of ReadyPin1, and then inverts the status.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ReadyPin1Invert	Identifies the ON/OFF for invert
ReadyPin1Enable	Identifies if this pin is used by the control part

Function

PLIB_EBI_ReadyPin1ConfigSet (EBI_MODULE_ID index, bool ReadyPin1Enable,
bool ReadyPin1Invert)

PLIB_EBI_ReadyPin2ConfigSet Function

Sets the control use of ReadyPin2 and inverts the status.

File

plib_ebi.h

C

```
void PLIB_EBI_ReadyPin2ConfigSet(EBI_MODULE_ID index, bool ReadyPin2Enable, bool ReadyPin2Invert);
```

Returns

None.

Description

This function sets the control use of ReadyPin2 and inverts the status.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ReadyPin2Invert	Identifies the ON/OFF for invert
ReadyPin2Enable	Identifies if this pin is used by the control part

Function

PLIB_EBI_ReadyPin2ConfigSet (EBI_MODULE_ID index, bool ReadyPin2Enable,
bool ReadyPin2Invert)

PLIB_EBI_ReadyPin3ConfigSet Function

Sets the control use of ReadyPin3 and inverts the status.

File

plib_ebi.h

C

```
void PLIB_EBI_ReadyPin3ConfigSet(EBI_MODULE_ID index, bool ReadyPin3Enable, bool ReadyPin3Invert);
```

Returns

None.

Description

This function sets the control use of ReadyPin3, and then inverts the status.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
ReadyPin3Invert	Identifies the ON/OFF for invert
ReadyPin3Enable	Identifies if this pin is used by the control part

Function

PLIB_EBI_ReadyPin3ConfigSet (EBI_MODULE_ID index, bool ReadyPin3Enable,
bool ReadyPin3Invert)

PLIB_EBI_ReadyPinSensSet Function

Sets the sensitivity of the Ready pin.

File

plib_ebi.h

C

```
void PLIB_EBI_ReadyPinSensSet(EBI_MODULE_ID index, bool SensitivityControl);
```

Returns

None.

Description

This function sets the sensitivity of the Ready pin.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
SensitivityControl	Selection edge or level detection

Function

PLIB_EBI_ReadyPinSensSet (EBI_MODULE_ID index, bool SensitivityControl)

PLIB_EBI_StaticMemoryWidthRegisterSet Function

Sets the data width of static memory.

File

plib_ebi.h

C

```
void PLIB_EBI_StaticMemoryWidthRegisterSet(EBI_MODULE_ID index, int RegisterNumber, EBI_STATIC_MEMORY_WIDTH
StaticMemoryWidthRegister);
```

Returns

None.

Description

This function sets the data width of static memory.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
RegisterNumber	The ID for the register being set
StaticMemoryWidthRegister	Identifies a bus width of either 8 bits or 16 bits

Function

PLIB_EBI_StaticMemoryWidthRegisterSet (EBI_MODULE_ID index, int RegisterNumber,
EBI_STATIC_MEMORY_WIDTH StaticMemoryWidthRegister)

PLIB_EBI_WriteOutputControlSet Function

Sets the Write Enable and Output Enable control pins.

File

plib_ebi.h

C

```
void PLIB_EBI_WriteOutputControlSet(EBI_MODULE_ID index, bool WriteEnable, bool OutputEnable);
```

Returns

None.

Description

This function sets the Write Enable and Output Enable control pins.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
WriteEnable	Used for enabling Write Enable
OutputEnable	Used for enabling Output Enable

Function

PLIB_EBI_WriteOutputControlSet (EBI_MODULE_ID index, bool WriteEnable,

bool OutputEnable)

PLIB_EBI_ControlEnableSet Function

Sets the EBI bus ON/OFF enable bit.

File

plib_ebi.h

C

```
void PLIB_EBI_ControlEnableSet(EBI_MODULE_ID index, bool EnableBit);
```

Returns

None.

Description

This function sets the EBI bus ON/OFF enable bit.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance
EnableBit	Identifies the enable bit

Function

PLIB_EBI_ControlEnableSet (EBI_MODULE_ID index, bool EnableBit)

PLIB_EBI_ReadyModeSet Function

Sets the use of Ready mode for each pin.

File

plib_ebi.h

C

```
void PLIB_EBI_ReadyModeSet(EBI_MODULE_ID index, bool ReadyPin0, bool ReadyPin1, bool ReadyPin2);
```

Returns

None.

Description

This function sets the use of Ready mode for each pin. The attached device will either pull the ready pin high or low.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance; ReadyPin0, ReadyPin1, ReadyPin2
ReadyPin(x)	Identifies the ready pin (1-3)

Function

PLIB_EBI_ReadyModeSet (EBI_MODULE_ID index, bool ReadyPin0, bool ReadyPin1,
bool ReadyPin2)

c) Feature Existence Functions

PLIB_EBI_ExistsAddressHoldTime Function

Identifies whether the AddressHoldTime feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsAddressHoldTime(EBI_MODULE_ID index);
```

Returns

- true - The AddressHoldTime feature is supported on the device
- false - The AddressHoldTime feature is not supported on the device

Description

This function identifies whether the AddressHoldTime feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_AddressHoldTimeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsAddressHoldTime(EBI_MODULE_ID index)

PLIB_EBI_ExistsAddressPinEnableBits Function

Identifies whether the AddressPinEnableBits feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsAddressPinEnableBits(EBI_MODULE_ID index);
```

Returns

- true - The AddressPinEnableBits feature is supported on the device
- false - The AddressPinEnableBits feature is not supported on the device

Description

This function identifies whether the AddressPinEnableBits feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_AddressPinEnableBitsSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsAddressPinEnableBits(EBI_MODULE_ID index)

PLIB_EBI_ExistsAddressSetupTime Function

Identifies whether the AddressSetupTime feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsAddressSetupTime(EBI_MODULE_ID index);
```

Returns

- true - The AddressSetupTime feature is supported on the device
- false - The AddressSetupTime feature is not supported on the device

Description

This function identifies whether the AddressSetupTime feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_AddressSetupTimeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsAddressSetupTime(EBI_MODULE_ID index)

PLIB_EBI_ExistsBaseAddress Function

Identifies whether the Base_Address feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsBaseAddress(EBI_MODULE_ID index);
```

Returns

- true - The Base_Address feature is supported on the device
- false - The Base_Address feature is not supported on the device

Description

This function identifies whether the Base_Address feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_BaseAddressSet
- PLIB_EBI_BaseAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsBaseAddress(EBI_MODULE_ID index)

PLIB_EBI_ExistsByteSelectPin Function

Identifies whether the ByteSelectPin feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsByteSelectPin(EBI_MODULE_ID index);
```

Returns

- true - The ByteSelectPin feature is supported on the device
- false - The ByteSelectPin feature is not supported on the device

Description

This function identifies whether the ByteSelectPin feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ByteSelectPinSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsByteSelectPin(EBI_MODULE_ID index)

PLIB_EBI_ExistsChipSelectEnable Function

Identifies whether the ChipSelectEnable feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsChipSelectEnable(EBI_MODULE_ID index);
```

Returns

- true - The ChipSelectEnable feature is supported on the device
- false - The ChipSelectEnable feature is not supported on the device

Description

This function identifies whether the ChipSelectEnable feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ChipSelectEnableSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsChipSelectEnable(EBI_MODULE_ID index)

PLIB_EBI_ExistsControlEnable Function

Identifies whether the ControlEnable feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsControlEnable(EBI_MODULE_ID index);
```

Returns

- true - The ControlEnable feature is supported on the device
- false - The ControlEnable feature is not supported on the device

Description

This function identifies whether the ControlEnable feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_ControlEnableSet
- PLIB_EBI_ControlEnableGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsControlEnable(EBI_MODULE_ID index)

PLIB_EBI_ExistsDataEnable Function

Identifies whether the DataEnable feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsDataEnable(EBI_MODULE_ID index);
```

Returns

- true - The DataEnable feature is supported on the device
- false - The DataEnable feature is not supported on the device

Description

This function identifies whether the DataEnable feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_DataEnableSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsDataEnable(EBI_MODULE_ID index)

PLIB_EBI_ExistsDataTurnAroundTime Function

Identifies whether the DataTurnAroundTime feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsDataTurnAroundTime(EBI_MODULE_ID index);
```

Returns

- true - The DataTurnAroundTime feature is supported on the device
- false - The DataTurnAroundTime feature is not supported on the device

Description

This function identifies whether the DataTurnAroundTime feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_DataTurnAroundTimeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsDataTurnAroundTime(EBI_MODULE_ID index)

PLIB_EBI_ExistsFlashPowerDownMode Function

Identifies whether the FlashPowerDownMode feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsFlashPowerDownMode(EBI_MODULE_ID index);
```

Returns

- true - The FlashPowerDownMode feature is supported on the device
- false - The FlashPowerDownMode feature is not supported on the device

Description

This function identifies whether the FlashPowerDownMode feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_FlashPowerDownModeSet
- PLIB_EBI_FlashPowerDownModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsFlashPowerDownMode(EBI_MODULE_ID index)

PLIB_EBI_ExistsFlashResetPin Function

Identifies whether the FlashResetPin feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsFlashResetPin(EBI_MODULE_ID index);
```

Returns

- true - The FlashResetPin feature is supported on the device
- false - The FlashResetPin feature is not supported on the device

Description

This function identifies whether the FlashResetPin feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_FlashResetPinSet
- PLIB_EBI_FlashResetPinGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsFlashResetPin(EBI_MODULE_ID index)

PLIB_EBI_ExistsFlashTiming Function

Identifies whether the FlashTiming feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsFlashTiming(EBI_MODULE_ID index);
```

Returns

- true - The FlashTiming feature is supported on the device
- false - The FlashTiming feature is not supported on the device

Description

This function identifies whether the FlashTiming feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_FlashTimingSet
- PLIB_EBI_FlashTimingGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsFlashTiming(EBI_MODULE_ID index)

PLIB_EBI_ExistsMemoryCharacteristics Function

Identifies whether the MemoryCharacteristics feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsMemoryCharacteristics(EBI_MODULE_ID index);
```

Returns

- true - The MemoryCharacteristics feature is supported on the device
- false - The MemoryCharacteristics feature is not supported on the device

Description

This function identifies whether the MemoryCharacteristics feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_MemoryCharacteristicsSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsMemoryCharacteristics(EBI_MODULE_ID index)

PLIB_EBI_ExistsMemoryPaging Function

Identifies whether the MemoryPaging feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsMemoryPaging(EBI_MODULE_ID index);
```

Returns

- true - The MemoryPaging feature is supported on the device
- false - The MemoryPaging feature is not supported on the device

Description

This function identifies whether the MemoryPaging feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_MemoryPagingSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsMemoryPaging(EBI_MODULE_ID index)

PLIB_EBI_ExistsMemoryTimingConfig Function

Identifies whether the MemoryTimingConfig feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsMemoryTimingConfig(EBI_MODULE_ID index);
```

Returns

- true - The MemoryTimingConfig feature is supported on the device
- false - The MemoryTimingConfig feature is not supported on the device

Description

This function identifies whether the MemoryTimingConfig feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_MemoryTimingConfigSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsMemoryTimingConfig(EBI_MODULE_ID index)

PLIB_EBI_ExistsPageMode Function

Identifies whether the PageMode feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsPageMode(EBI_MODULE_ID index);
```

Returns

- true - The PageMode feature is supported on the device
- false - The PageMode feature is not supported on the device

Description

This function identifies whether the PageMode feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_PageModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsPageMode(EBI_MODULE_ID index)

PLIB_EBI_ExistsPageReadTime Function

Identifies whether the PageReadTime feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsPageReadTime(EBI_MODULE_ID index);
```

Returns

- true - The PageReadTime feature is supported on the device
- false - The PageReadTime feature is not supported on the device

Description

This function identifies whether the PageReadTime feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_PageReadCycleTimeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsPageReadTime(EBI_MODULE_ID index)

PLIB_EBI_ExistsReadCycleTime Function

Identifies whether the ReadCycleTime feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsReadCycleTime(EBI_MODULE_ID index);
```

Returns

- true - The ReadCycleTime feature is supported on the device
- false - The ReadCycleTime feature is not supported on the device

Description

This function identifies whether the ReadCycleTime feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ReadCycleTimeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsReadCycleTime(EBI_MODULE_ID index)

PLIB_EBI_ExistsReadyMode Function

Identifies whether the ReadyMode feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsReadyMode(EBI_MODULE_ID index);
```

Returns

- true - The ReadyMode feature is supported on the device
- false - The ReadyMode feature is not supported on the device

Description

This function identifies whether the ReadyMode feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_ReadyModeSet
- PLIB_EBI_ReadyModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsReadyMode(EBI_MODULE_ID index)

PLIB_EBI_ExistsReadyPin1Config Function

Identifies whether the ReadyPin1Config feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsReadyPin1Config(EBI_MODULE_ID index);
```

Returns

- true - The ReadyPin1Config feature is supported on the device
- false - The ReadyPin1Config feature is not supported on the device

Description

This function identifies whether the ReadyPin1Config feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ReadyPin1ConfigSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsReadyPin1Config(EBI_MODULE_ID index)

PLIB_EBI_ExistsReadyPin2Config Function

Identifies whether the ReadyPin2Config feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsReadyPin2Config(EBI_MODULE_ID index);
```

Returns

- true - The ReadyPin2Config feature is supported on the device
- false - The ReadyPin2Config feature is not supported on the device

Description

This function identifies whether the ReadyPin2Config feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ReadyPin2ConfigSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsReadyPin2Config(EBI_MODULE_ID index)

PLIB_EBI_ExistsReadyPin3Config Function

Identifies whether the ReadyPin3Config feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsReadyPin3Config(EBI_MODULE_ID index);
```

Returns

- true - The ReadyPin3Config feature is supported on the device
- false - The ReadyPin3Config feature is not supported on the device

Description

This function identifies whether the ReadyPin3Config feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ReadyPin3ConfigSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsReadyPin3Config(EBI_MODULE_ID index)

PLIB_EBI_ExistsReadyPinSens Function

Identifies whether the ReadyPinSens feature exists on the EBI module

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsReadyPinSens(EBI_MODULE_ID index);
```

Returns

- true - The ReadyPinSens feature is supported on the device
- false - The ReadyPinSens feature is not supported on the device

Description

This function identifies whether the ReadyPinSens feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_ReadyPinSensSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsReadyPinSens(EBI_MODULE_ID index)

PLIB_EBI_ExistsStaticMemoryWidthRegister Function

Identifies whether the StaticMemoryWidthRegister feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsStaticMemoryWidthRegister(EBI_MODULE_ID index);
```

Returns

- true - The StaticMemoryWidthRegister feature is supported on the device
- false - The StaticMemoryWidthRegister feature is not supported on the device

Description

This function identifies whether the StaticMemoryWidthRegister feature is available on the EBI module. When this function returns true, these functions are supported on the device:

- PLIB_EBI_StaticMemoryWidthRegisterSet
- PLIB_EBI_StaticMemoryWidthRegisterGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsStaticMemoryWidthRegister(EBI_MODULE_ID index)

PLIB_EBI_ExistsWriteOutputControl Function

Identifies whether the WriteOutputControl feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsWriteOutputControl(EBI_MODULE_ID index);
```

Returns

- true - The WriteOutputControl feature is supported on the device
- false - The WriteOutputControl feature is not supported on the device

Description

This function identifies whether the WriteOutputControl feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_WriteOutputControlSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsWriteOutputControl(EBI_MODULE_ID index)

PLIB_EBI_ExistsWritePulseWidth Function

Identifies whether the WritePulseWidth feature exists on the EBI module.

File

plib_ebi.h

C

```
bool PLIB_EBI_ExistsWritePulseWidth(EBI_MODULE_ID index);
```

Returns

- true - The WritePulseWidth feature is supported on the device
- false - The WritePulseWidth feature is not supported on the device

Description

This function identifies whether the WritePulseWidth feature is available on the EBI module. When this function returns true, this function is supported on the device:

- PLIB_EBI_WritePulseWidthGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_EBI_ExistsWritePulseWidth(EBI_MODULE_ID index)

d) Data Types and Constants

EBI_ADDRESS_PORT Enumeration

Selects the EBI address port.

File

plib_ebi_help.h

C

```
typedef enum {
  EBI_EBIADDR_PIN0,
  EBI_EBIADDR_PIN1,
  EBI_EBIADDR_PIN2,
  EBI_EBIADDR_PIN3,
  EBI_EBIADDR_PIN4,
  EBI_EBIADDR_PIN5,
  EBI_EBIADDR_PIN6,
  EBI_EBIADDR_PIN7,
  EBI_EBIADDR_PIN8,
  EBI_EBIADDR_PIN9,
  EBI_EBIADDR_PIN10,
  EBI_EBIADDR_PIN11,
  EBI_EBIADDR_PIN12,
  EBI_EBIADDR_PIN13,
  EBI_EBIADDR_PIN14,
  EBI_EBIADDR_PIN15,
  EBI_EBIADDR_PIN16,
  EBI_EBIADDR_PIN17,
  EBI_EBIADDR_PIN18,
  EBI_EBIADDR_PIN19,
  EBI_EBIADDR_PIN20,
  EBI_EBIADDR_PIN21,
  EBI_EBIADDR_PIN22,
  EBI_EBIADDR_PIN23,
  EBI_ADDR_PRESET8,
  EBI_ADDR_PRESET12,
  EBI_ADDR_PRESET16,
  EBI_ADDR_PRESET_ALL
} EBI_ADDRESS_PORT;
```

Members

Members	Description
EBI_EBIADDR_PIN0	PIN 0
EBI_EBIADDR_PIN1	PIN 1
EBI_EBIADDR_PIN2	PIN 2
EBI_EBIADDR_PIN3	PIN 3
EBI_EBIADDR_PIN4	PIN 4
EBI_EBIADDR_PIN5	PIN 5
EBI_EBIADDR_PIN6	PIN 6
EBI_EBIADDR_PIN7	PIN 7
EBI_EBIADDR_PIN8	PIN 8
EBI_EBIADDR_PIN9	PIN 9

EBI_EBIADDR_PIN10	PIN 10
EBI_EBIADDR_PIN11	PIN 11
EBI_EBIADDR_PIN12	PIN 12
EBI_EBIADDR_PIN13	PIN 13
EBI_EBIADDR_PIN14	PIN 14
EBI_EBIADDR_PIN15	PIN 15
EBI_EBIADDR_PIN16	PIN 16
EBI_EBIADDR_PIN17	PIN 17
EBI_EBIADDR_PIN18	PIN 18
EBI_EBIADDR_PIN19	PIN 19
EBI_EBIADDR_PIN20	PIN 20
EBI_EBIADDR_PIN21	PIN 21
EBI_EBIADDR_PIN22	PIN 22
EBI_EBIADDR_PIN23	PIN 23
EBI_ADDR_PRESET8	Preset 8
EBI_ADDR_PRESET12	Preset 12
EBI_ADDR_PRESET16	Preset 16
EBI_ADDR_PRESET_ALL	Preset All

Description

EBI Address Port

This enumeration selects the EBI address port.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

EBI_CS_TIMING Enumeration

Selects the timing register set for Chip Select.

File

plib_ebi_help.h

C

```
typedef enum {
  CS_TIMING_0,
  CS_TIMING_1,
  CS_TIMING_2
} EBI_CS_TIMING;
```

Members

Members	Description
CS_TIMING_0	Use EBISMT0
CS_TIMING_1	Use EBISMT1
CS_TIMING_2	Use EBISMT2

Description

Timing Register for Chip Select

This enumeration selects the timing register set for Chip Select

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

EBI_MEMORY_SIZE Enumeration

Selects the memory size for Chip Select.

File

plib_ebi_help.h

C

```
typedef enum {
  CSNOTUSED,
  MEMORY_SIZE_64KB,
  MEMORY_SIZE_128KB,
  MEMORY_SIZE_256KB,
  MEMORY_SIZE_512KB,
  MEMORY_SIZE_1MB,
  MEMORY_SIZE_2MB,
  MEMORY_SIZE_4MB,
  MEMORY_SIZE_8MB,
  MEMORY_SIZE_16MB
} EBI_MEMORY_SIZE;
```

Members

Members	Description
CSNOTUSED	Chip Select Not Used
MEMORY_SIZE_64KB	64 KB (smaller memories alias within this range)
MEMORY_SIZE_128KB	128 KB
MEMORY_SIZE_256KB	256 KB
MEMORY_SIZE_512KB	512 KB
MEMORY_SIZE_1MB	1 MB
MEMORY_SIZE_2MB	2 MB
MEMORY_SIZE_4MB	4 MB
MEMORY_SIZE_8MB	8 MB
MEMORY_SIZE_16MB	16 MB

Description

Memory Size for Chip Select

This enumeration selects memory size for Chip Select.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

EBI_MEMORY_TYPE Enumeration

Selects the memory type for Chip Select.

File

plib_ebi_help.h

C

```
typedef enum {
  SRAM,
  NORFLASH
} EBI_MEMORY_TYPE;
```

Members

Members	Description
SRAM	SRAM
NORFLASH	NOR-Flash

Description

Memory Type for Chip Select

This enumeration selects the memory type for Chip Select.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

EBI_MODULE_ID Enumeration

Identifies the supported EBI modules.

File

plib_ebi_help.h

C

```
typedef enum {
  EBI_ID_0,
  EBI_NUMBER_OF_MODULES
} EBI_MODULE_ID;
```

Members

Members	Description
EBI_ID_0	EBI Module 0 ID
EBI_NUMBER_OF_MODULES	Number of available WDT modules.

Description

EBI Module ID

This enumeration identifies the available EBI modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

EBI_PAGE_SIZE Enumeration

Selects the page size for the page mode device.

File

plib_ebi_help.h

C

```
typedef enum {
  PAGE_WORD4,
  PAGE_WORD8,
  PAGE_WORD16,
  PAGE_WORD32
} EBI_PAGE_SIZE;
```

Members

Members	Description
PAGE_WORD4	4-word page
PAGE_WORD8	8-word page
PAGE_WORD16	16-word page
PAGE_WORD32	32-word page

Description

Page Size for page mode device

This enumeration selects the page size for the page mode device.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

EBI_STATIC_MEMORY_WIDTH Enumeration

Selects the static memory width for the register EBISMTx.

File

plib_ebi_help.h

C

```
typedef enum {
  MEMORY_WIDTH_8BIT,
  MEMORY_WIDTH_16BIT
} EBI_STATIC_MEMORY_WIDTH;
```

Members

Members	Description
MEMORY_WIDTH_8BIT	8 bits
MEMORY_WIDTH_16BIT	16 bits

Description

Memory Width

This enumeration selects the static memory width for the register EBISMTx.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Files

Files

Name	Description
plib_ebi.h	EBI Peripheral Library Interface Header.
plib_ebi_help.h	

Description

This section lists the source and header files used by the library.

plib_ebi.h

EBI Peripheral Library Interface Header.

Functions

	Name	Description
	PLIB_EBI_AddressHoldTimeGet	Returns the address hold time.
	PLIB_EBI_AddressPinEnableBitsSet	Sets the address pins used for EBI.
	PLIB_EBI_AddressSetupTimeGet	Returns the setup hold time.
	PLIB_EBI_BaseAddressGet	Returns the base address set for each Chip Select.
	PLIB_EBI_BaseAddressSet	Sets the base address for physical memory at each Chip Select pin.
	PLIB_EBI_ByteSelectPinSet	Sets the data Byte Select High <15:8> and Low <7:0> enable pins for use.
	PLIB_EBI_ChipSelectEnableSet	Sets the Chip Select pins for use with the EBI or GPIO.
	PLIB_EBI_ControlEnableGet	Returns the status of the EBI enable bit.
	PLIB_EBI_ControlEnableSet	Sets the EBI bus ON/OFF enable bit.
	PLIB_EBI_DataEnableSet	Sets the use of Data Byte Select Pins, High and Low, for control with EBI or GPIO.
	PLIB_EBI_DataTurnAroundTimeGet	Returns the data turn-around time set for the EBI bus.
	PLIB_EBI_ExistsAddressHoldTime	Identifies whether the AddressHoldTime feature exists on the EBI module.
	PLIB_EBI_ExistsAddressPinEnableBits	Identifies whether the AddressPinEnableBits feature exists on the EBI module.
	PLIB_EBI_ExistsAddressSetupTime	Identifies whether the AddressSetupTime feature exists on the EBI module.
	PLIB_EBI_ExistsBaseAddress	Identifies whether the Base_Address feature exists on the EBI module.
	PLIB_EBI_ExistsByteSelectPin	Identifies whether the ByteSelectPin feature exists on the EBI module.
	PLIB_EBI_ExistsChipSelectEnable	Identifies whether the ChipSelectEnable feature exists on the EBI module.
	PLIB_EBI_ExistsControlEnable	Identifies whether the ControlEnable feature exists on the EBI module.
	PLIB_EBI_ExistsDataEnable	Identifies whether the DataEnable feature exists on the EBI module.
	PLIB_EBI_ExistsDataTurnAroundTime	Identifies whether the DataTurnAroundTime feature exists on the EBI module.
	PLIB_EBI_ExistsFlashPowerDownMode	Identifies whether the FlashPowerDownMode feature exists on the EBI module.
	PLIB_EBI_ExistsFlashResetPin	Identifies whether the FlashResetPin feature exists on the EBI module.

	PLIB_EBI_ExistsFlashTiming	Identifies whether the FlashTiming feature exists on the EBI module.
	PLIB_EBI_ExistsMemoryCharacteristics	Identifies whether the MemoryCharacteristics feature exists on the EBI module.
	PLIB_EBI_ExistsMemoryPaging	Identifies whether the MemoryPaging feature exists on the EBI module.
	PLIB_EBI_ExistsMemoryTimingConfig	Identifies whether the MemoryTimingConfig feature exists on the EBI module.
	PLIB_EBI_ExistsPageMode	Identifies whether the PageMode feature exists on the EBI module.
	PLIB_EBI_ExistsPageReadTime	Identifies whether the PageReadTime feature exists on the EBI module.
	PLIB_EBI_ExistsReadCycleTime	Identifies whether the ReadCycleTime feature exists on the EBI module.
	PLIB_EBI_ExistsReadyMode	Identifies whether the ReadyMode feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPin1Config	Identifies whether the ReadyPin1Config feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPin2Config	Identifies whether the ReadyPin2Config feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPin3Config	Identifies whether the ReadyPin3Config feature exists on the EBI module.
	PLIB_EBI_ExistsReadyPinSens	Identifies whether the ReadyPinSens feature exists on the EBI module
	PLIB_EBI_ExistsStaticMemoryWidthRegister	Identifies whether the StaticMemoryWidthRegister feature exists on the EBI module.
	PLIB_EBI_ExistsWriteOutputControl	Identifies whether the WriteOutputControl feature exists on the EBI module.
	PLIB_EBI_ExistsWritePulseWidth	Identifies whether the WritePulseWidth feature exists on the EBI module.
	PLIB_EBI_FlashPowerDownModeGet	Returns the set power-down state.
	PLIB_EBI_FlashPowerDownModeSet	Sets the pin state for Flash devices on Power-down and Reset.
	PLIB_EBI_FlashResetPinGet	Sets the control use of Flash Reset pin.
	PLIB_EBI_FlashResetPinSet	Sets the control use of the Flash Reset pin.
	PLIB_EBI_FlashTimingGet	Returns the set Flash timing for external Flash.
	PLIB_EBI_FlashTimingSet	Sets the timing for hold in reset for external Flash.
	PLIB_EBI_MemoryCharacteristicsSet	Sets the characteristics for memory or attached devices attached to the specified pin.
	PLIB_EBI_MemoryPageSizeGet	Returns the Paging mode settings.
	PLIB_EBI_MemoryPagingSet	Sets the size of the memory page if paging is used.
	PLIB_EBI_MemoryTimingConfigSet	Sets the cycle time for page reading.
	PLIB_EBI_PageModeGet	Returns the Paging mode settings.
	PLIB_EBI_PageReadCycleTimeGet	Returns the cycle time for a page read.
	PLIB_EBI_ReadCycleTimeGet	Returns the read time in the number of clock cycles.
	PLIB_EBI_ReadyModeGet	Returns whether or not Ready mode was set.
	PLIB_EBI_ReadyModeSet	Sets the use of Ready mode for each pin.
	PLIB_EBI_ReadyPin1ConfigSet	Sets the control use of ReadyPin1 and inverts the status.
	PLIB_EBI_ReadyPin2ConfigSet	Sets the control use of ReadyPin2 and inverts the status.
	PLIB_EBI_ReadyPin3ConfigSet	Sets the control use of ReadyPin3 and inverts the status.
	PLIB_EBI_ReadyPinSensSet	Sets the sensitivity of the Ready pin.
	PLIB_EBI_StaticMemoryWidthRegisterGet	Returns the set width of the data bus.
	PLIB_EBI_StaticMemoryWidthRegisterSet	Sets the data width of static memory.
	PLIB_EBI_WriteOutputControlSet	Sets the Write Enable and Output Enable control pins.
	PLIB_EBI_WritePulseWidthGet	Returns the set hold time in clock cycles.

Description

External Bus Interface (EBI) Peripheral Library Interface

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the EBI Peripheral Library.

File Name

plib_ebi.h

Company

Microchip Technology Inc.

plib_ebi_help.h

Enumerations

	Name	Description
	EBI_ADDRESS_PORT	Selects the EBI address port.
	EBI_CS_TIMING	Selects the timing register set for Chip Select.

	EBI_MEMORY_SIZE	Selects the memory size for Chip Select.
	EBI_MEMORY_TYPE	Selects the memory type for Chip Select.
	EBI_MODULE_ID	Identifies the supported EBI modules.
	EBI_PAGE_SIZE	Selects the page size for the page mode device.
	EBI_STATIC_MEMORY_WIDTH	Selects the static memory width for the register EBISMTx.

Section

Data Types & Constants

Ethernet Peripheral Library

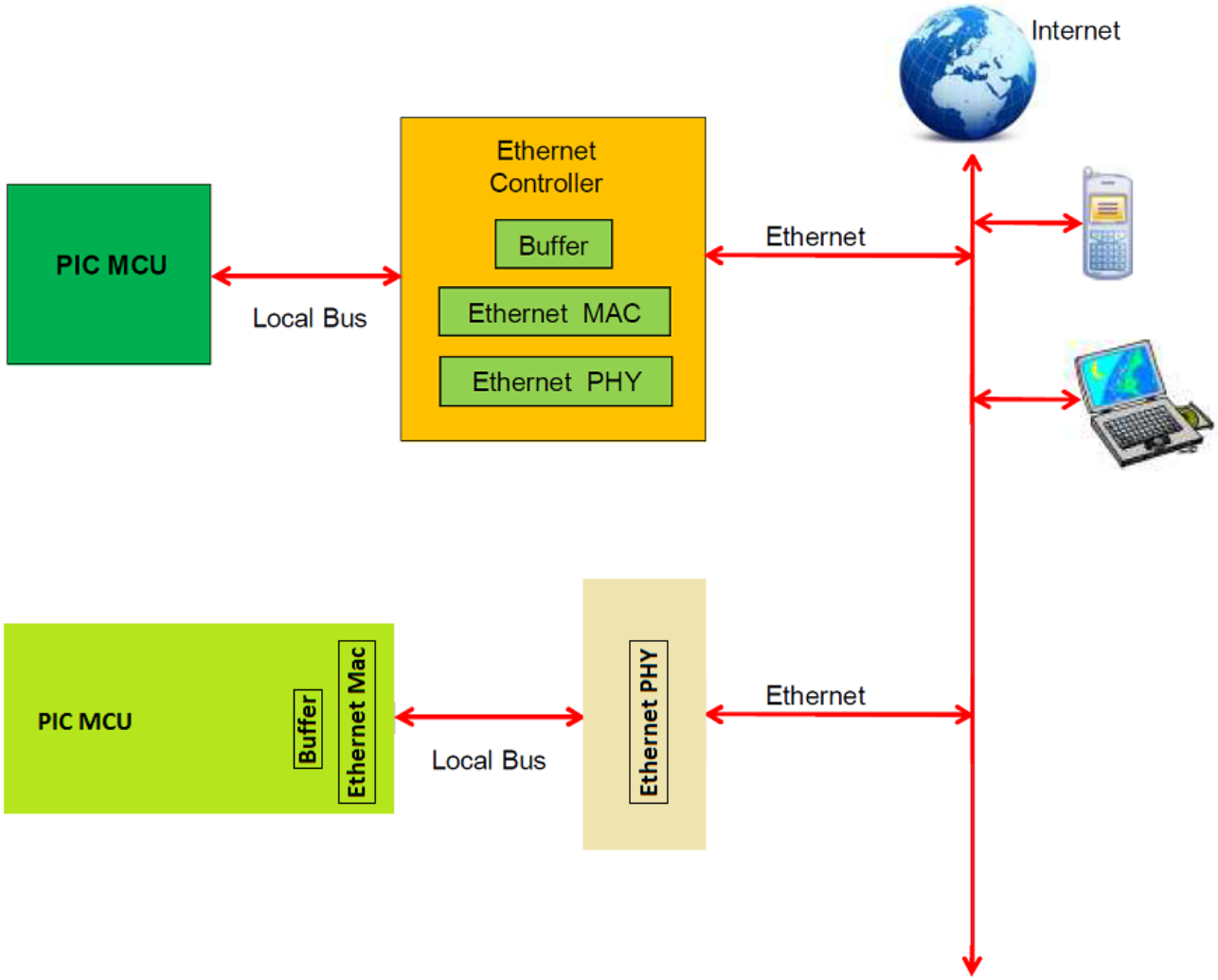
This section describes the Ethernet Peripheral Library.

Introduction

The Ethernet Peripheral Library provides a low-level abstraction of the Ethernet module in the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without interacting directly with the module's registers, thereby hiding the differences in microcontroller variations.

Description

Ethernet is the most widely deployed network in offices and homes. Ethernet’s infrastructure, interoperability, and scalability serve to ease development and facilitate communications to the device. Once equipment is connected to an Ethernet network, it can be monitored or controlled through the internet with low latency – "real time" remote delivery.



The Ethernet Controller is a bus master module that interfaces with an off-chip PHY to implement a complete Ethernet node in a system.

Following are some of the key features of this module:

- Supports 10/100 Mbps data transfer rates
- Supports Full-Duplex and Half-Duplex operation
- Supports Reduced Media Independent Interface (RMII) and Media Independent Interface (MII) PHY interface
- Supports MII Management (MIIM) PHY Management interface
- Supports both manual and automatic flow control
- Supports Auto-MDIX and enabled PHYs
- RAM descriptor based DMA operation for both receive and transmit path
- Fully configurable interrupts
- Configurable receive packet filtering:
- CRC Check
- 64-byte Pattern Match
- Broadcast, Multicast and Unicast packets
- Magic Packet™
- 64-bit Hash Table
- Runt Packet
- Supports Packet Payload Checksum calculation

- Supports various hardware statistics counters

Using the Library

This topic describes the basic architecture of the Ethernet Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_eth.h`

The interface to the Ethernet Peripheral Library is defined in the `plib_eth.h` header file., which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Ethernet Peripheral Library should include `peripheral.h`.

Library File:

The Ethernet Peripheral Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the Peripheral interacts with the framework.

Hardware Abstraction Model

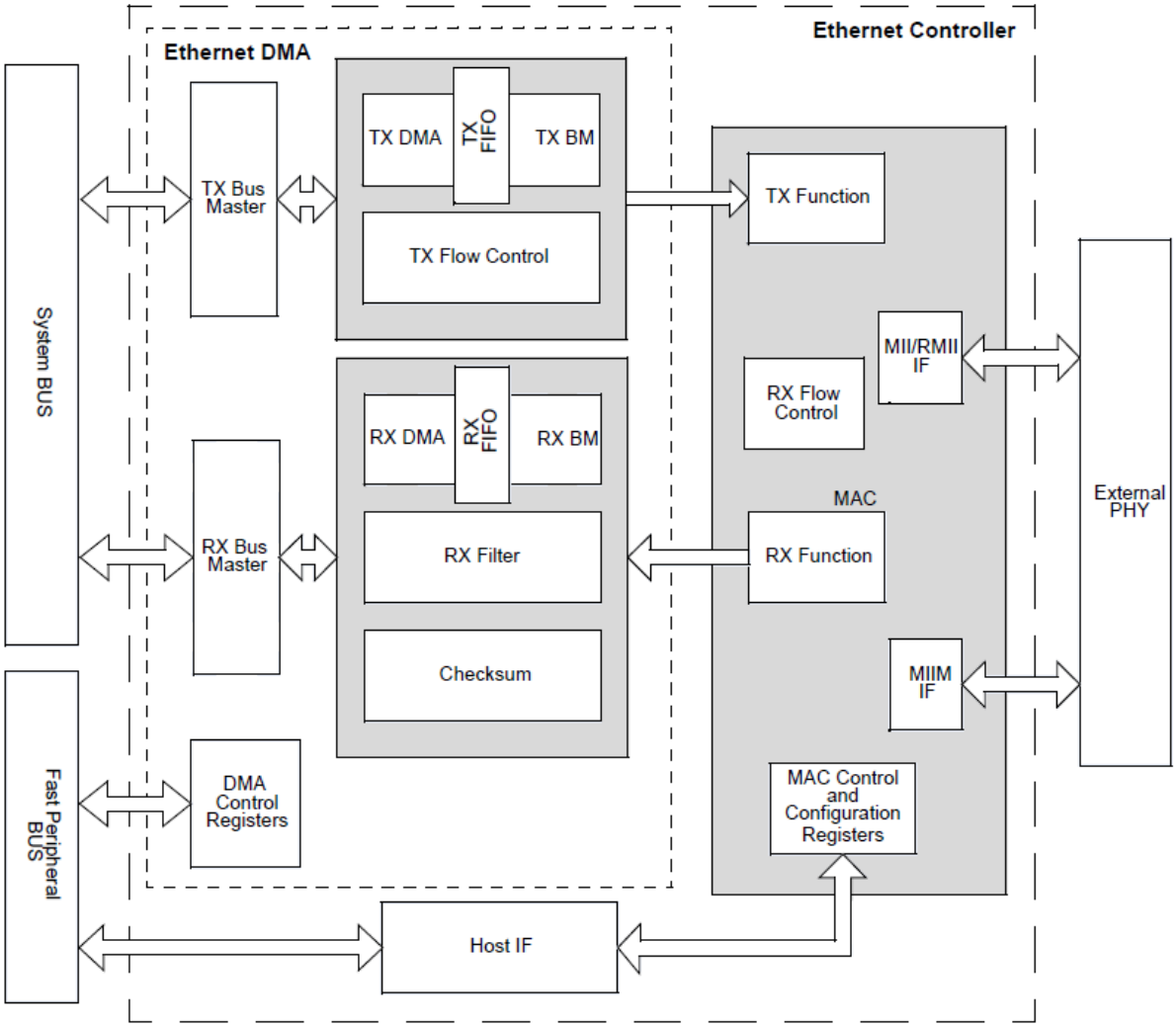
This section describes how the low-level abstraction is modeled in the software and introduces the library interface. The interface provided is a superset of all the functionality of the available Ethernet on the device. Refer to the specific data sheet or family reference manual to determine the set of functions that are supported for each Ethernet module on your device.

Description

Hardware Model

This library provides the low-level abstraction of the Ethernet module on Microchip family of microcontrollers with a convenient C language interface.

Ethernet Controller Block Diagram



Note: For information on off-chip options to provide Ethernet functionality, refer to the specific device data sheet.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Ethernet module.

Library Interface Section	Description
Initialization Functions	Provides setup and configuration routines.
Control Functions	Provides control routines.
Status Functions	Provides status routines.
Filtering Functions	Provides filtering routines.
Flow Control Functions	Provides flow control routines.
Interrupt Functions	Provides interrupt routines.
Statistics Functions	Provides statistics routines.
Feature Existence Functions	These functions can be used to determine whether or not a particular feature exists on the device.

How the Library Works

This section provides information on the operation of the Ethernet Peripheral Library.

Description

The Peripheral Library is an abstraction of hardware which can be used to control hardware and obtain status. It is intended to allow the programmer access to hardware registers without knowing the exact bit or register names which may vary on individual devices. For example, instead of turning on the Ethernet module using:

ETHCON1bits.EON = 1; the programmer can turn on the Ethernet module using: PLIB_ETH_Enable(moduleId);. This allows the programmer to use the same command for any device containing an Ethernet module without having to learn the individual device register names.

Basic Overview

The Ethernet Controller provides the modules needed to implement a 10/100 Mbps Ethernet node using an external PHY chip. To off-load the CPU from moving packet data to and from the module, internal descriptor-based DMA engines are included in the controller.

The Ethernet Controller consists of the following modules:

- Media Access Control (MAC) block: Responsible for implementing the MAC functions of the Ethernet IEEE 802.3 Specification
- Flow Control (FC) block, which provides:
- Manual flow control
- Automatic flow control
- Transmission of PAUSE frames
- Reception of PAUSE frame is handled within the MAC
- RX Filter (RXF) block: This module performs filtering on every receive packet to determine whether each packet should be accepted or rejected and includes:
- CRC Check
- 64-byte Pattern Match
- Broadcast, Multicast and Unicast packets
- Magic Packet™
- 64-bit Hash Table
- Runt Packet
- TX DMA/TX BM Engine: The TX DMA and TX Buffer Management engines perform data transfers from the memory (using descriptor tables) to the MAC Transmit Interface
- RX DMA/RX BM Engine: The RX DMA and RX Buffer Management engines transfer receive packets from the MAC to the memory (using descriptor tables)

Basic Ethernet Controller Operation

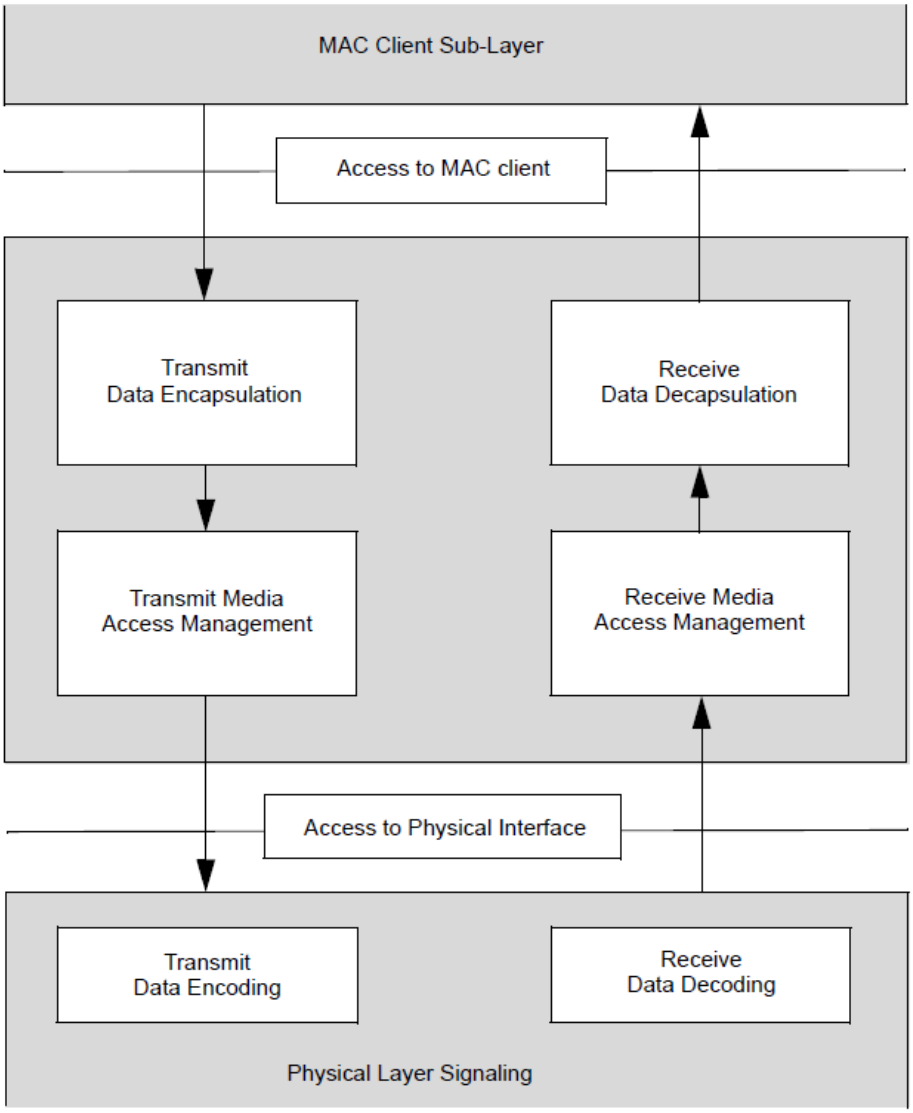
The Ethernet Controller is enabled using PLIB_ETH_Enable and is disabled using PLIB_ETH_Disable. The Ethernet Controller is disabled by default after any Reset. If the Ethernet Controller is disabled, all of the I/O pins used for the MII/RMII and MIIM interfaces operate as port pins and are under the control of the respective PORTx latch bit and TRISx bit.

Disabling the controller resets the internal DMA state machines and all transmit and receive operations are aborted. The Special Function Registers (SFRs) are still accessible and their values are preserved.

Clearing the ON bit while the Ethernet Controller is active will abort all pending operations and reset the peripheral as previously defined.

Re-enabling the ON bit will restart the Ethernet Controller in its clean reset state while preserving the SFR values.

Ethernet Controller Section Block Diagram



MAC Overview

The MAC sub-layer is part of the functionality described in the OSI model for the Data Link Layer. It defines a medium-independent facility, built on the medium-dependent physical facility provided by the Physical Layer, and under the access-layer-independent LLC sub-layer or other MAC client. It is applicable to a general class of local area broadcast media suitable for use with the Carrier Sense Multiple Access with Collision Detection (CSMA/CD).

The CSMA/CD MAC sub-layer provides services to the MAC client required for the transmission and reception of frames. The CSMA/CD MAC sub-layer makes a best effort to acquire the medium and transfer a serial stream of bits to the Physical Layer. Although certain errors are reported to the client, error recovery is not provided by the MAC.

The following is a summary of the functional capabilities of the CSMA/CD MAC sub-layer as shown in the previous diagram.

- For Frame Transmission:
- Accepts data from the MAC client and constructs a frame
- Presents a bit-serial data stream to the Physical Layer for transmission on the medium
- In half-duplex mode, defers transmission of a bit-serial stream whenever the physical medium is busy
- It can append proper Frame Check Sequence (FCS) value to outgoing frames and verifies full octet boundary alignment
- Delays transmission of frame bit stream for specified inter-frame gap period
- In half-duplex mode, halts transmission when collision is detected
- In half-duplex mode, schedules retransmission after a collision until a specified retry limit is reached
- In half-duplex mode, enforces collision to ensure propagation throughout network by sending jam message
- Prepends preamble and Start Frame Delimiter and appends FCS to all frame Appends PAD field for frames whose data length is less than a minimum value
- For Frame Reception:

- Receives a bit-serial data stream from the Physical Layer
- Presents the received frames to the MAC client (broadcast, multicast, unicast frames, etc.)
- Checks incoming frames for transmission errors by way of FCS and verifies octet boundary alignment
- Removes preamble, Start Frame Delimiter and PAD field (if necessary) from received frames
- Implements the MII Management block that provides control/status connection to the external MII PHY

Note: Not all modes are available on all devices, please refer to the specific device data sheet to determine the set of modes supported for your device. In addition, refer to **Section 35. "Ethernet Controller"** (DS60001155) of the *"PIC32 Family Reference Manual"* or the Ethernet MAC Driver Library for more information.

Ethernet Frame Overview

Ethernet Frame Overview

IEEE 802.3-compliant Ethernet frames (packets) are between 64 and 1518 bytes long (Preamble and Start -of-Frame Delimiter not included). Frames containing less than 64 bytes are known as "runt" frames, while frames containing more than 1518 bytes are known as "huge" frames.

An Ethernet frame is made up of the following fields:

- Start-of-Stream/Preamble
- Start-of-Frame Delimiter (SFD)
- Destination MAC address (DA)
- Source MAC address (SA)
- Type/Length field
- Data Payload
- Optional Padding field
- Frame Check Sequence (FCS)
- Traffic on the actual physical cable

Please refer to the IEEE 802.3 Specification for detailed information about the Ethernet protocol.

Ethernet Controller Operation

Basic Ethernet Controller Operation

The Ethernet Controller is enabled using PLIB_ETH_Enable and is disabled using PLIB_ETH_Disable. The Ethernet Controller is disabled by default after any Reset. If the Ethernet Controller is disabled, all of the I/O pins used for the MII/RMII and MIIM interfaces operate as port pins and are under the control of the respective PORT latch bit and TRIS bit.

Disabling the controller resets the internal DMA state machines and all transmit and receive operations are aborted. The SFRs are still accessible and their values preserved.

Clearing the ON bit while the Ethernet Controller is active will abort all pending operations and reset the peripheral as previously defined.

Re-enabling the ON bit will restart the Ethernet Controller in its clean reset state while preserving the SFR values.

Media Independent Interface (MII)

This section describes the Media Independent Interface (MII).

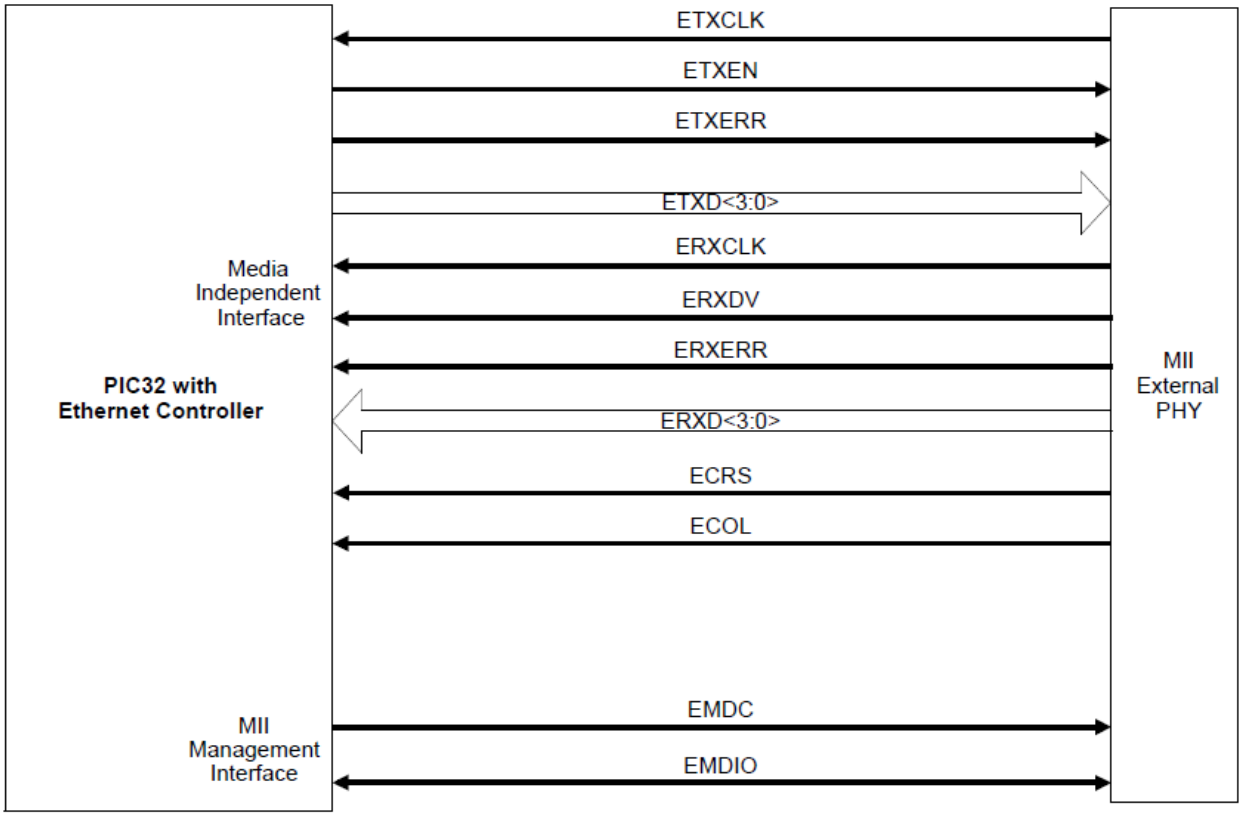
Description

Media Independent Interface (MII)

The Media Independent Interface (MII) is a standard interconnection between the MAC and the PHY for communicating TX and RX frame data. This MII has the following important characteristics:

- Capable of supporting 10/100 Mbps rates for data transfer, and offers support for management functions
- Provides independent four bit wide transmit and receive data paths
- Uses TTL signal levels, compatible with common digital CMOS processes
- Provides full-duplex operation

In 10 Mbps mode, the MII runs at 2.5 MHz; in 100 Mbps mode it runs at 25 MHz. PHYs that provide MII are not required to support both data rates, and may support either one or both.



Reduced Media Independent Interface (RMII)

This section describes the Reduced Media Independent Interface (RMII).

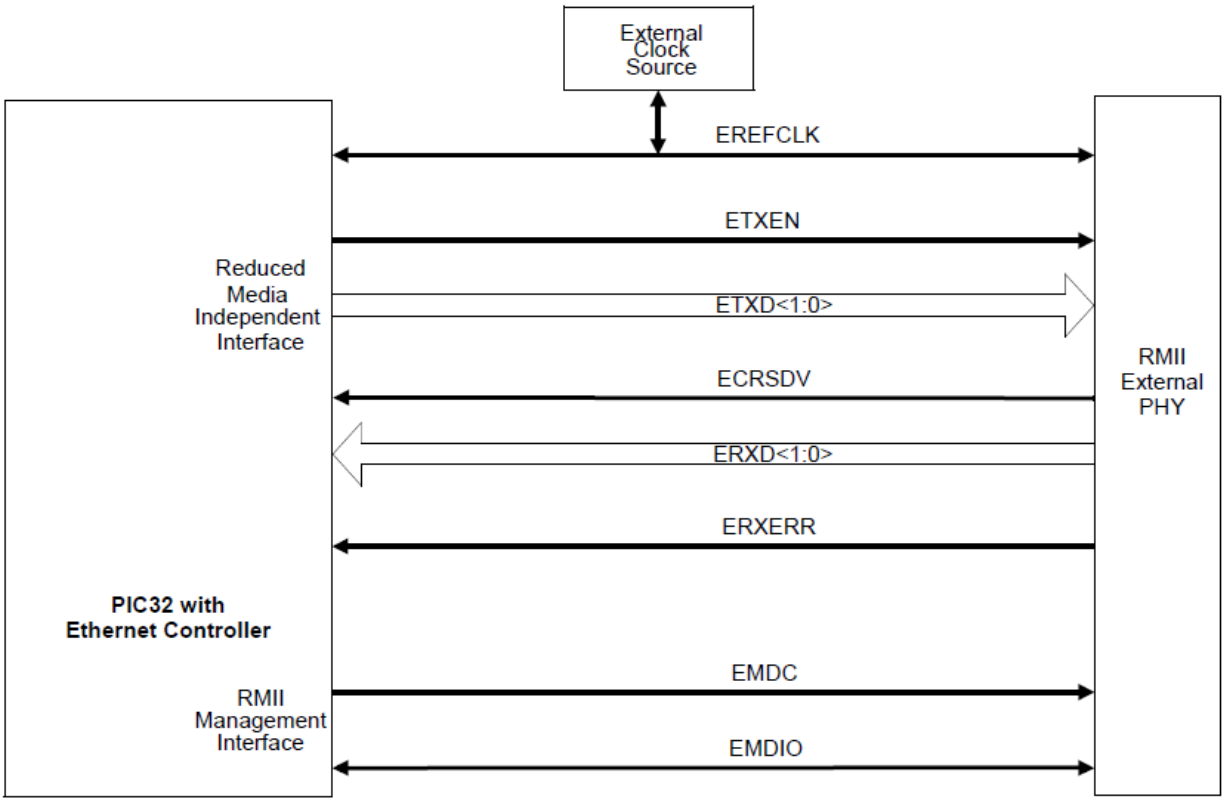
Description

Reduced Media Independent Interface (RMII)

The management interface (MDIO/MDC) is assumed to be identical to that defined in MII.

The RMII has the following characteristics:

- Capable of supporting 10 Mbps and 100 Mbps data rates
- Single clock reference for both MAC and the PHY (can be sourced from the MAC or from an external source)
- Provides independent 2 bit wide transmit and receive data paths
- Uses TTL signal levels, compatible with common digital CMOS processes
- Provides full-duplex operation
- The interface runs at 50 MHz



Flow Control Overview

Flow Control Overview

Ethernet flow control consists of the ability to send and receive PAUSE frames, which cause the receiving node to stop transmitting for a specific amount of time.

On the transmit side, the Flow Control (FC) block handles the hardware handshaking between the MAC and the CPU when the transmit flow control is enabled. Flow control for the received packets is part of the MAC functionality.

The PIC32 MAC supports both Symmetric PAUSE and Asymmetric PAUSE as described in Clause 28, Table 28B-2, and Clause 31 and Annex 31B of the IEEE 802.3 Specification.

The FC block supports two modes of operation: manual and automatic. In addition, the mode of transmission (Full-Duplex or Half-Duplex) programmed into the MAC registers, is used by the FC block.

Receive Filtering Overview

Receive Filtering Overview

The Receive Filter (RXF) block examines all incoming receive packets and accepts or rejects the packet, based on user-selectable filters. The following RX filters are supported:

- CRC Error Acceptance Filter
- Runt Error Acceptance Filter
- CRC Check Rejection Filter
- Runt Rejection Filter
- Unicast Acceptance Filter
- Not Me Unicast Acceptance Filter
- Multicast Acceptance Filter
- Broadcast Acceptance Filter
- Hash Table Acceptance Filter
- Magic Packet Acceptance Filter
- Pattern Match Acceptance Filter with logical inversion

Ethernet DMA and Buffer Management Engine

Ethernet DMA and Buffer Management Engines

To reduce the overhead on the CPU to move the packet data between data memory and the Ethernet controller, internal receive (RX) and transmit (TX) DMA engines are integrated into the module. The DMA engines are responsible for transferring data from system memory to the MAC for transmit packets and for transferring data from the MAC to system memory for receive packets. The DMA engines each have access to the system memory by acting as two different bus masters, one bus master for transmit and one for receive.

The DMA engines use separate Ethernet Descriptor Tables (EDTs) for TX and RX operations to determine where the TX/RX packet buffer resides in the system memory.

Both transmit and receive descriptors, called Ethernet Descriptors (EDs), used by the DMA engines have a similar format, with only the status word formats being different.

Refer to the **"Ethernet"** section in the family reference manual for more information.

Sample Code

This section provides information for initialization, transmit, and receive using code samples.

Description

Sample Code

The basic functionality of the Ethernet Peripheral Library is grouped into these sections:

- Initialization
- Transmit
- Receive

For a properly functioning Ethernet port, development of a data queue and use of interrupts will be necessary.

For this additional functionality, see the **"Ethernet"** section in the family reference manual or the Ethernet Driver Library for more information.

Initialization Sequence

The Ethernet Controller is enabled using PLIB_ETH_Enable and is disabled using PLIB_ETH_Disable. The Ethernet Controller is disabled by default after any Reset.

After being Enabled, disabling the controller resets the internal DMA state machines and all transmit and receive operations are aborted. All registers remain accessible and their values are preserved. It is important to reset the PHY and MII Management registers to ensure the PHY is in a known initialized state.

Before enabling the module for transmitting or receiving packets, there are several registers that must be initialized. To initialize the Ethernet Controller to receive and transmit, Microchip recommends the following sequence:

1. Begin with the module.
- Disable the CPU Ethernet interrupt
- Turn the Ethernet Controller off
- Disable receive
- Disable transmit
- Wait until the Ethernet module is no longer busy
- Disable all Ethernet Interrupts
- Clear the Ethernet Interrupt Pending Flag(s)
- Clear the transmit and receive start addresses

2. MAC initialize.
- Reset the MAC
- A configuration fuse (FETHIO) setting shows which pins will be used for connection to the PHY
- A configuration fuse (FMIIEN) settings shows whether MII or RMII will be used

3. PHY Initialize is PHY dependent. Common procedures (which may vary by PHY vender) include the following:
- Reset PHY (using Control Register 0)
- Set the MII/RMII operation mode
- Set the normal, swapped, or auto (preferred)MDIX
- Check the PY capabilities (using Status Register 1)
- Preferably the auto-negotiation should be selected if the PHY supports it.
- Expose the supported capabilities (extended Register 4)
- Start the negotiation (using Control Register 0)
- wait for the negotiation to complete and get the link partner capabilities (extended Register 5)
- negotiation result (vendor-specific register)

- If auto-negotiation is not supported (or not selected)
- update the PHY Duplex and speed settings (use Control Register 0 or vender-specific register)

4. MAC Configuration

- Enable MAC receive and pause both transmit and receive direction control frames.
- Select the desired auto-padding, duplex, huge frame, and CRC capabilities.
- Program back-to-back inter-packet gap and non-back-to-back interpacket gap.
- Program collision window and maximum retransmissions.
- Set Maximum frame length.
- Optionally set the station MAC address(es) (or let them default to the factory value).

5. Continue Ethernet Controller initialization:

- If planning to turn on flow control, update the Pause Timer Value
- If using auto-flow control, set up the full/empty watermarks and enable it
- Set the receive filters
- Set receive buffer size - using packets that are too small impacts performance
- Prepare the list/ring of transmit descriptors - refer to the device data sheet for more information
- Prepare the list of receive descriptors populated with valid buffers for messages to be received - refer to the device data sheet for more information
- Update the transmit buffer start address and the receive buffer start address
- Enable the Ethernet Controller and wait until the module is ready
- Enable Ethernet Receive.

Receive Loop

1. Wait until the receive buffer count has been incremented.
2. Inspect the receive descriptor to see whether it is still owned by the Ethernet Module.
3. If the receive descriptor is still owned by the Ethernet module, nothing has been received. Loop to 1.
4. Otherwise, a message was received and software has control of the memory where it is stored.
5. Extract a message and do something with it.
6. Reinitialize buffer descriptor and return receive buffer to list and decrement the receive packet buffer count.
7. Loop.

Transmit Loop

1. Software writes the packet data to packet data memory. A pointer to the packet data is inserted into the transmit buffer descriptor.
2. Update the necessary number of transmit descriptors, starting from the head of the list - setting the data buffer address (large packet fragmentation will impact performance).
3. Update the byte count for each descriptor with the number of bytes contained in each buffer.
4. Set the flag in each transmit descriptor that belongs to the packet to give the descriptor to the Ethernet module for transmission.
5. Enable the transmission.
6. Wait until the Ethernet module returns the transmit buffer descriptor ownership to software.
7. Insect the results of the transmission.
8. Loop.

The packet transmit process is as follows:

1. Software writes the packet data to a packet buffer in data memory and programs a buffer descriptor entry to point to the packet data buffer. The buffer descriptor is used to define the Transmission Packet data buffer location and length, and its address is written to the Ethernet Controller.
2. Software then calls the TransmitRequestToSendEnable function, which starts the TX DMA and TXBM logic.
3. After the TX DMA engine reads the DATA_BUFFER_ADDRESS value, the engine will begin reading the packet data from the location read from the descriptor.
4. The TXBM indicates the start-of-frame to the MAC and transmits the entire frame data from the transmit buffer, until the end address is reached. The TXBM simply transmits from the start address until the specified number of bytes has been transmitted to the MAC transmit interface.
5. The MAC can retry a transmission due to an early collision.
6. The MAC can abort a transmission due to a late collision, excessive collisions or excessive defers. This condition is signaled by TXABORT bit (ETHIRQ<2>).
7. Once transmission has completed, the TX DMA engine stores the relevant bits of the TSV into the first descriptor entry of the packet.
8. After the packet has been transmitted, all the descriptors used for the transmission are released to the software via the EOWN bits being cleared.
9. If more valid TX descriptors are available (EOWN = 1), the DMA engine will go back to step 3 to begin the next packet's transmission. Otherwise, if the next descriptor is still owned by hardware (EOWN = 0), transmission will halt and wait for the software to set the TXRTS bit again.

Note that any collision that occurs within the CWINDOW bit (EMAC1CLRT<8:14>) boundary is an early collision and results in a Retry operation. A collision that happens beyond the CWINDOW boundary will be treated as a late collision and will cause an abort. The CWINDOW bit in the EMAC1CLRT register is typically set to 64 bytes. An abort condition can also result from reaching the maximum collision count RETX bit

(EMAC1CLRT<0:3>) for a packet trying to be sent.

See the **"Ethernet"** section in the family reference manual or the Ethernet Driver Library for more information.

Initialization

```
#include "plib_eth.h"

/*****************************************************************************
*/
void my_ETH_Initialize(ETH_MODULE_ID id)
{

    //Disable System (CPU) Interrupt associated with Ethernet Peripheral
    // using the Interrupt Peripheral Library
    // Example: PLIB_INT_SourceDisable(PLIB_INT_SOURCE_ETH_1);

    //Disable Ethernet Peripheral in case it was previously enabled
    PLIB_ETH_Disable(ETH_MODULE_ID1);

    //Disable any Ethernet Controller interrupt generation:
    PLIB_ETH_InterruptSourceDisable(ETH_MODULE_ID1, ETH_ALL_INTERRUPT_SOURCES);

    PLIB_ETH_RxDisable(ETH_MODULE_ID1);
    PLIB_ETH_TxRTSDisable(ETH_MODULE_ID1);

    while (PLIB_ETH_EthernetIsBusy(ETH_MODULE_ID1)) {/*wait*/ }

    //Clear the System Interrupt Flag associated with Ethernet Peripheral
    // using the Interrupt Peripheral Library
    // Example:     PLIB_INT_SourceFlagClear(PLIB_INT_SOURCE_ETH_1);

    ////////////////////////////////////////////////
    //MAC Initialize
    PLIB_ETH_MIIResetEnable(ETH_MODULE_ID1);
    PLIB_ETH_MIIResetDisable(ETH_MODULE_ID1);

    //A configuration fuse (FETHIO) setting shows which digital pins need to be configured
    if (CONFIGURATION_FUSE_PRIMARY_ETH_PINS==1)
    {
        ConfigurePrimaryEthernetPinsAsDigital();
    }
    else //(CONFIGURATION_FUSE_PRIMARY_ETH_PINS==0)
    {
        ConfigureAlternateEthernetPinsAsDigital();
    }

    //A configuration fuse (FMIIEN) settings shows which digital pins need to be configured
    if (CONFIGURATION_FUSE_MII_STYLE==RMII)
    {
        PLIB_ETH_RMIIResetEnable(ETH_MODULE_ID1);
        PLIB_ETH_RMIIResetDisable(ETH_MODULE_ID1);
        PLIB_ETH_RMIISpeedSet(ETH_MODULE_ID1, ETH_RMII_100Mps);
    }
    PLIB_ETH_MIIResetEnable(ETH_MODULE_ID1);
    PLIB_ETH_MIIResetDisable(ETH_MODULE_ID1);
    PLIB_ETH_MIIMClockSet(ETH_MODULE_ID1, ETH_MII_SYSCLK_DIV_BY_10);


    ////////////////////////////////////////////////
    //PHY Initialize is PHY dependent.
    // Common procedures (which may vary by PHY vender) include the following:
    //
    //Reset PHY (using Control Register 0)
    //Set the MII/RMII operation mode
    //Set the normal, swapped, or auto (preferred)MDIX
    //Check the PY capabilities (using Status Register 1)
    //Preferably the auto-negotiation should be selected if the PHY supports it.
    //Expose the supported capabilities (extended Register 4)
```

```
    //Start the negotiation (using Control Register 0)
    //wait for the negotiation to complete and get the link partner capabilities (extended Register 5)
    //negotiation result (vendor-specific register)
    //If auto-negotiation is not supported (or not selected)
    //update the PHY Duplex and speed settings (use Control Register 0 or vender-specific register)

    ////////////////////////////////////////////////
    //MAC Configuration
    PLIB_ETH_RxEnable(ETH_MODULE_ID1);
    PLIB_ETH_TxPauseEnable(ETH_MODULE_ID1);
    PLIB_ETH_RxPauseEnable(ETH_MODULE_ID1);

    //Select the desired auto-padding, duplex, huge frame, and CRC capabilities
    PLIB_ETH_AutoDetecPadSet(ETH_MODULE_ID1, ETH_AUTOPAD_BY_PKT_TYPE_ADD_CRC );
    PLIB_ETH_FullDuplexEnable(ETH_MODULE_ID1); //or disable for half-duplex
    PLIB_ETH_CRCEnable(ETH_MODULE_ID1); //or disable if EMAC does not add CRC when no padding is necessary
    PLIB_ETH_HugeFrameDisable(ETH_MODULE_ID1); //or enable to receive huge frames

    //Program back-to-back inter-packet gap and non-back-to-back interpacket gap
    PLIB_ETH_BackToBackIPGpSet(ETH_MODULE_ID1, 0x15 /*backToBackGapValue*/);
    PLIB_ETH_NonBackToBackIPG1Set(ETH_MODULE_ID1, 0x0C /*nonBackToBackGap1Value*/);
    PLIB_ETH_NonBackToBackIPG2Set(ETH_MODULE_ID1, 0x12 /*nonBackToBackGap2Value*/);

    //Set Maximum frame length
    PLIB_ETH_MaxFrameLengthSet(ETH_MODULE_ID1, 1527 /*maxFrameLenValue*/);

    //Optionally set the station MAC address(es) (or let them default to the factory value)
    PLIB_ETH_StationAddressSet(ETH_MODULE_ID1, 1, 0x13 /*stationAddr1Value*/);


    ////////////////////////////////////////////////
    //Continue Ethernet Controller Initialization

    //Set receive buffer size - using packets that are too small impacts performance
    PLIB_ETH_ReceiveBufferSizeSet(ETH_MODULE_ID1, 200 /*receiveBufSizeValue*/);

    //Prepare the list/ring of transmit descriptors - refer to the device data sheet for more information
    //Prepare the list of receive descriptors populated with valid buffers for messages to be received -
refer
    //to the device data sheet for more information
    //Update the transmit buffer start address and the receive buffer start address
    PLIB_ETH_TxPacketDescAddrSet(ETH_MODULE_ID1, (uint8_t *)0xadd1add0/*txPktDescStartAddrValue*/);
    PLIB_ETH_RxPacketDescAddrSet(ETH_MODULE_ID1, (uint8_t *)0xadd3add2/*rxPktDescStartAddrValue*/);

    //If using auto flow control, set up the full/empty watermarks and enable it
    PLIB_ETH_PauseTimerSet(ETH_MODULE_ID1, 0x64 /*pauseTimerValue*/);
    PLIB_ETH_RxEmptyWmarkSet(ETH_MODULE_ID1, 0x3F/*emptyRxWatermarkValue*/);
    PLIB_ETH_RxFullWmarkSet(ETH_MODULE_ID1, 0x10 /*fullRxWatermarkValue*/);
    PLIB_ETH_AutoFlowControlEnable(ETH_MODULE_ID1);

    PLIB_ETH_ReceiveFilterEnable(ETH_MODULE_ID1,
                                 ETH_MULTICAST_FILT |   // enable to accept multicast address packets
                                 ETH_BROADCAST_FILTER );// enable to accept all broadcast address packets

    PLIB_ETH_InterruptSourceDisable(ETH_MODULE_ID1, ETH_ALL_INTERRUPT_SOURCES);

    //Enable the Ethernet Controller and wait until the module is ready
    PLIB_ETH_Enable(ETH_MODULE_ID1);
    while (PLIB_ETH_EthernetIsBusy(ETH_MODULE_ID1))
    {
        //Wait
    }

    PLIB_ETH_RxEnable(ETH_MODULE_ID1);

    return (EXIT_SUCCESS);
}
```

Descriptor Table Example

```
typedef volatile struct
{
    union
    { //Common to both Receive and Transmit
        struct
        {
            unsigned: 7;
            unsigned EOWN: 1; //0=Software owns, 1=Ethernet Controller owns
            unsigned NPV: 1;  //0=Next Desc follows this, 1=Next Desc pointed to by NEXT_ED
            unsigned: 7;
            unsigned bCount: 11;
            unsigned: 3;
            unsigned EOP: 1;
            unsigned SOP: 1;
        };
        uint32_t w;
    };                      // descriptor header

    //Common to both Receive and Transmit
    uint8_t* pEDBuff;     // pointer to data buffer

    union
    { //This portion differs from Receive to Transmit
        uint64_t RX_STATUS;    //Receive Status
        struct
        {
            unsigned RX_PKT_CHECKSUM: 16;
            unsigned: 8;
            unsigned RXF_RSV: 8;
            unsigned RSV: 32;
        };

        uint64_t TX_STATUS;    // Transmit Status
        struct //Transmit Struct
        {
            unsigned TSV_TRANSMIT_BYTE_COUNT: 16;
            unsigned TSV_TX_COLLISION_COUNT: 4;
            unsigned TSV_TX_CRC_ERROR: 1;
            unsigned TSV_TX_LENGTH_CHECK_ERROR: 1;
            unsigned TSV_TX_DONE: 1;
            unsigned TSV_TX_MULTICAST: 1;
            unsigned TSV_TX_BROADCAST: 1;
            unsigned TSV_TX_PACKET_DEFER: 1;
            unsigned TSV_TX_EXCESSIVE_DEFER: 1;
            unsigned TSV_TX_MAXIMUM_COLLISION: 1;
            unsigned TSV_TX_LATE_COLLISION: 1;
            unsigned TSV_TX_GIANT: 1;
            unsigned TSV_TX_UNDER_RUN: 1;
            unsigned TSV_TX_STATUS_VECTOR:1;

            unsigned TSV_TOTAL_BYTES_TX_ON_WIRE: 16;
            unsigned TSV_TX_CONTROL_FRAME: 1;
            unsigned TSV_TX_PAUSE_CONTROL_FRAME: 1;
            unsigned TSV_TX_BACKPRESSURE_APPLIED: 1;
            unsigned TSV_TX_VLAN_TAGGED_FRAME: 1;
            unsigned TSV_IGNORED: 4;
            unsigned TSV_USER_DEFINED: 8;
        };
    };                  // descriptor header

    //Common to both Receive and Transmit
    uint8_t* next_ed;       // next descriptor (NPV==1);

}__attribute__((__packed__)) eth_buffer_descriptor_t; // hardware RX descriptor (linked).
```

For more information on this structure, refer to the **"Ethernet"** section in the family reference manual.

Transmit

C

Description

```
#include "plib_eth.h"

/*********************************************************************
//
*/

void my_ETH_Transmit(ETH_MODULE_ID id, eth_buffer_descriptor_t *pArrDcpt, uint8_t *pArrBuff,
                     uint8_t nArrayItems, uint16_t *pArrSize)
{
    //Input Parameters:
    //   *pArrDcpt : pointer to an array that holds free descriptors
    //   *pArrBuff: pointer to an array that holds buffers to be transmitted
    //   *pArrSize: pointer to an array that holds the sizes of the buffers to be transmitted
    //   nArrayItems: number of buffers to be transmitted
    //
    // Before attempting to transmit, the data must be placed into a packet
    // buffer (list) and the point(s) must be placed into transmit descriptor(s)
    // Refer to the DRV_ETH or the device family reference manual for more information

    extern void* VA_TO_PA(char* pBuff); // extern function that returns the physical
                                        // address of the input virtual address parameter

    uint16_t ix;                        // loop index
    eth_buffer_descriptor_t *pEDcpt;    // current Ethernet descriptor
    eth_buffer_descriptor_t *tailDcpt; // last Ethernet descriptor
    uint8_t *pBuff;                     // current data buffer to be transmitted
    uint16_t *pSize;                    // current data buffer size

    pEDcpt=pArrDcpt;
    pBuff=pArrBuff;
    pSize=pArrSize;
    tailDcpt=0;

    for(ix=0; ix< nArrayItems; ix++, pEDcpt++, pBuff++, pSize++)
    {
        // pass the descriptor to hw, use linked descriptors, set proper size
        pEDcpt->pEDBuff = (uint8_t *)VA_TO_PA(pBuff); // set buffer
        pEDcpt->w=0;                                  // clear all the fields
        pEDcpt->NPV=1;                                // set next pointer valid
        pEDcpt->EOWN=1;                               // set hw ownership
        pEDcpt->bCount = *pSize;                      // set proper size
        if(tailDcpt)
        {
            tailDcpt->next_ed = VA_TO_PA(&pEDcpt);
        }
        tailDcpt=pEDcpt;
    }
    // at this moment pEDcpt is an extra descriptor we use to end the descriptors list
    pEDcpt->w=0;                                                  // software ownership
    tailDcpt->next_ed= VA_TO_PA(&pEDcpt);
    pArrDcpt[0].SOP=1;                                            // start of packet
    pArrDcpt[nArrayItems-1].EOP=1;                                // end of packet

    PLIB_ETH_TransmitPacketDescStartAddrSet(id, VA_TO_PA(pArrDcpt)); // set the transmit address
    PLIB_ETH_TransmitRequestToSendIsEnabled(id);                     // set the TXRTS

    // the ETHC will transmit the buffers we just programmed
    /* do something else in between */
    while (PLIB_ETH_TxRTSIsEnabled(id))
    {
       //Wait until transmit is not busy
```

```
    }

    // check the ETHSTAT register to see the transfer result
 }
```

Receive

```
#include "plib_eth.h"

/*****************************************************************
//
*/
void my_ETH_Receive(ETH_MODULE_ID id, eth_buffer_descriptor_t *pArrDcpt, uint8_t *pArrBuff,
                    uint8_t nArrayItems, uint16_t rxBuffSize)
{
   extern void* VA_TO_PA(char* pBuff); // extern function that returns the physical
                                       // address of the input virtual address parameter
    int ix;// index
    eth_buffer_descriptor_t *pEDcpt;   // current Ethernet descriptor
    eth_buffer_descriptor_t *tailDcpt; // last Ethernet descriptor
    char* pBuff;// current data buffer to be transmitted

    pEDcpt=pArrDcpt;
    pBuff=pArrBuff;
    tailDcpt=0;

    //Prepare to receive
    PLIB_ETH_ReceiveBufferSizeSet(id, (rxBuffSize/16));

    for(ix=0; ix< nArrayItems; ix++, pEDcpt++, pBuff++)
    {// pass the descriptor to hw, use linked descriptors, set proper size
        pEDcpt->pEDBuff=(unsigned char*)VA_TO_PA(pBuff);// set buffer
        pEDcpt->w=0;// clear all the fields
        pEDcpt->NPV= 1;// set next pointer valid
        pEDcpt->EOWN= 1;// set hw ownership

        if(tailDcpt)
        {
            tailDcpt->next_ed=VA_TO_PA(&pEDcpt);
        }
        tailDcpt=pEDcpt;
    }
    // at this moment pEDcpt is an extra descriptor we use to end the descriptors list
    pEDcpt->w=0;// software ownership
    tailDcpt->next_ed= VA_TO_PA(&pEDcpt);

    PLIB_ETH_RxPacketDescStartAddrSet(id, VA_TO_PA(pArrDcpt));
    // set the address of the first RX descriptor
    PLIB_ETH_ReceiveEnable(id);
    // the Ethernet Controller will receive frames and place them in the receive buffers
    // we just programmed
    /* do something else in between */

    while(PLIB_ETH_RxPacketCountGet(id)==0)
    {
        //Wait until something is received.
        // Packets are received as set by the receive filter
    }

    //Receive the packet
    //doSomethingWithReceivedPacket()
    // After the received packet has been processed, restore EOWN to 1 and
    PLIB_ETH_RxBufferCountDecrement(id);
}
```

Support for Legacy "Ethernet Controller Library"

These routines (PLIB_ETH_Legacy*) provide MPLAB Harmony support for applications that use the "Ethernet Controller Library for Microchip

PIC32MX Microcontrollers" that is installed with XC32 compilers. Refer to the related compiled Help, `hlpETH.chm`, which is located in the `../Microchip/xc32/<version>/docs/pic32-lib-help` folder of your XC32 compiler installation for legacy information.

An additional argument, an index value from the ETH_MODULE_ID enumeration, has been added to allow these functions to work from within the context of this MPLAB Harmony Peripheral Library.

Legacy Controller Library	MPLAB Harmony
EthClose	DRV_ETHMAC_LegacyClose
EthDescriptorGetBuffer	DRV_ETHMAC_LegacyDescriptorGetBuffer
EthDescriptorsPoolAdd	DRV_ETHMAC_LegacyDescriptorsPoolAdd
EthDescriptorsPoolCleanUp	DRV_ETHMAC_LegacyDescriptorsPoolCleanUp
EthDescriptorsPoolRemove	DRV_ETHMAC_LegacyDescriptorsPoolRemove
EthEventsClr	PLIB_ETH_EventsClr
EthEventsEnableClr	PLIB_ETH_EventsEnableClr
EthEventsEnableGet	PLIB_ETH_EventsEnableGet
EthEventsEnableSet	PLIB_ETH_EventsEnableSet
EthEventsEnableWrite	PLIB_ETH_EventsEnableWrite
EthEventsGet	PLIB_ETH_EventsGet
EthInit	DRV_ETHMAC_LegacyInit
EthMACGetAddress	PLIB_ETH_MACGetAddress
EthMACOpen	DRV_ETHMAC_LegacyMACOpen
EthMACSetAddress	PLIB_ETH_MACSetAddress
EthMACSetMaxFrame	PLIB_ETH_MACSetMaxFrame
EthRxAcknowledgeBuffer	DRV_ETHMAC_LegacyRxAcknowledgeBuffer
EthRxAcknowledgePacket	DRV_ETHMAC_LegacyRxAcknowledgePacket
EthRxBuffersAppend	DRV_ETHMAC_LegacyRxBuffersAppend
EthRxFiltersClr	PLIB_ETH_RxFiltersClr
EthRxFiltersHTSet	PLIB_ETH_RxFiltersHTSet
EthRxFiltersPMClr	PLIB_ETH_RxFiltersPMClr
EthRxFiltersPMSet	PLIB_ETH_RxFiltersPMSet
EthRxFiltersSet	PLIB_ETH_RxFiltersSet
EthRxFiltersWrite	PLIB_ETH_RxFiltersWrite
EthRxGetBuffer	DRV_ETHMAC_LegacyRxGetBuffer
EthRxGetPacket	DRV_ETHMAC_LegacyRxGetPacket
EthRxSetBufferSize	PLIB_ETH_RxSetBufferSize
EthStatRxAlgnErrCnt	PLIB_ETH_StatRxAlgnErrCnt
EthStatRxFcsErrCnt	PLIB_ETH_StatRxFcsErrCnt
EthStatRxOkCnt	PLIB_ETH_StatRxOkCnt
EthStatRxOvflCnt	PLIB_ETH_StatRxOvflCnt
EthStatTxMColCnt	PLIB_ETH_StatTxMColCnt
EthStatTxOkCnt	PLIB_ETH_StatTxOkCnt
EthStatTxSColCnt	PLIB_ETH_StatTxSColCnt
EthTxAcknowledgeBuffer	DRV_ETHMAC_LegacyTxAcknowledgeBuffer
EthTxAcknowledgePacket	DRV_ETHMAC_LegacyTxAcknowledgePacket
EthTxGetBufferStatus	DRV_ETHMAC_LegacyTxGetBufferStatus
EthTxGetPacketStatus	DRV_ETHMAC_LegacyTxGetPacketStatus
EthTxSendBuffer	DRV_ETHMAC_LegacyTxSendBuffer
EthTxSendPacket	DRV_ETHMAC_LegacyTxSendPacket

Ethernet Peripheral Library	Legacy Controller Library
PLIB_ETH_EventsClr	EthEventsClr
PLIB_ETH_EventsEnableClr	EthEventsEnableClr
PLIB_ETH_EventsEnableGet	EthEventsEnableGet
PLIB_ETH_EventsEnableSet	EthEventsEnableSet
PLIB_ETH_EventsEnableWrite	EthEventsEnableWrite
PLIB_ETH_EventsGet	EthEventsGet
PLIB_ETH_MACGetAddress	EthMACGetAddress
PLIB_ETH_MACSetAddress	EthMACSetAddress
PLIB_ETH_MACSetMaxFrame	EthMACSetMaxFrame
PLIB_ETH_RxFiltersClr	EthRxFiltersClr
PLIB_ETH_RxFiltersHTSet	EthRxFiltersHTSet
PLIB_ETH_RxFiltersPMClr	EthRxFiltersPMClr
PLIB_ETH_RxFiltersPMSet	EthRxFiltersPMSet
PLIB_ETH_RxFiltersSet	EthRxFiltersSet
PLIB_ETH_RxFiltersWrite	EthRxFiltersWrite
PLIB_ETH_RxSetBufferSize	EthRxSetBufferSize
PLIB_ETH_StatRxAlgnErrCnt	EthStatRxAlgnErrCnt
PLIB_ETH_StatRxFcsErrCnt	EthStatRxFcsErrCnt
PLIB_ETH_StatRxOkCnt	EthStatRxOkCnt
PLIB_ETH_StatRxOvflCnt	EthStatRxOvflCnt
PLIB_ETH_StatTxMColCnt	EthStatTxMColCnt
PLIB_ETH_StatTxOkCnt	EthStatTxOkCnt
PLIB_ETH_StatTxSColCnt	EthStatTxSColCnt

Configuring the Library

The library is configured for the supported Ethernet module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Initialization Functions

	Name	Description
	PLIB_ETH_AutoDetectPadClear	Clears the EMAC Auto-Pad option.
	PLIB_ETH_AutoDetectPadGet	Gets the current EMAC Auto-Pad setting.
	PLIB_ETH_AutoDetectPadSet	Sets the EMAC Auto-Pad option.
	PLIB_ETH_BackToBackIPGGet	Gets the EMAC back-to-back interpacket gap.
	PLIB_ETH_BackToBackIPGSet	Sets the EMAC back-to-back interpacket gap.
	PLIB_ETH_CollisionWindowGet	Gets the EMAC collision window.
	PLIB_ETH_CollisionWindowSet	Sets the EMAC collision window.
	PLIB_ETH_DelayedCRCDisable	Disables EMAC delayed CRC.
	PLIB_ETH_DelayedCRCEnable	Enables EMAC delayed CRC.
	PLIB_ETH_DelayedCRCIsEnabled	Gets the EMAC delayed CRC enable status.
	PLIB_ETH_Disable	Disables the Ethernet module.
	PLIB_ETH_Enable	Enables the Ethernet module.
	PLIB_ETH_ExcessDeferDisable	Disables EMAC excess defer.
	PLIB_ETH_ExcessDeferEnable	Enables EMAC excess defer.
	PLIB_ETH_ExcessDeferIsEnabled	Gets the EMAC excess defer enable status.
	PLIB_ETH_FrameLengthCheckDisable	Disables the EMAC frame length check.
	PLIB_ETH_FrameLengthCheckEnable	Enables the EMAC frame length check.
	PLIB_ETH_FrameLengthCheckIsEnabled	Gets the EMAC frame length check status.
	PLIB_ETH_FullDuplexDisable	Disables the EMAC full duplex operation.
	PLIB_ETH_FullDuplexEnable	Enables the EMAC full duplex operation.
	PLIB_ETH_FullDuplexIsEnabled	Gets the EMAC full duplex enable status.

	PLIB_ETH_HugeFrameDisable	Disables the EMAC from receiving huge frames.
	PLIB_ETH_HugeFrameEnable	Enables the EMAC to receive huge frames.
	PLIB_ETH_HugeFrameIsEnabled	Gets the EMAC huge frame enable status.
	PLIB_ETH_IsEnabled	Gets the Ethernet module enable status.
	PLIB_ETH_LongPreambleDisable	Disables EMAC long preamble enforcement.
	PLIB_ETH_LongPreambleEnable	Enables EMAC long preamble enforcement.
	PLIB_ETH_LongPreambleIsEnabled	Gets the EMAC long preamble enforcement enable status.
	PLIB_ETH_LoopbackDisable	Disables the EMAC loopback logic.
	PLIB_ETH_LoopbackEnable	Enables the EMAC loopback logic.
	PLIB_ETH_LoopbackIsEnabled	Gets the EMAC Loopback interface enable status.
	PLIB_ETH_MaxFrameLengthGet	Gets the EMAC maximum frame length.
	PLIB_ETH_MaxFrameLengthSet	Sets the EMAC maximum frame length.
	PLIB_ETH_MIIMClockGet	Gets the current EMAC MIIM clock selection.
	PLIB_ETH_MIIMClockSet	Sets the EMAC MIM clock selection.
	PLIB_ETH_MIIMNoPreDisable	Disables EMAC No Preamble (allows preamble).
	PLIB_ETH_MIIMNoPreEnable	Enables EMAC MIIM No Preamble (suppresses preamble).
	PLIB_ETH_MIIMNoPreIsEnabled	Gets the EMAC MIIM No Preamble enable status.
	PLIB_ETH_NonBackToBackIPG1Get	Gets the EMAC non-back-to-back interpacket gap register 1.
	PLIB_ETH_NonBackToBackIPG1Set	Sets the EMAC non-back-to-back interpacket gap register 1.
	PLIB_ETH_NonBackToBackIPG2Get	Gets the EMAC non-back-to-back interpacket gap register 2.
	PLIB_ETH_NonBackToBackIPG2Set	Sets the EMAC non-back-to-back interpacket gap register 2.
	PLIB_ETH_PauseTimerGet	Gets the Pause Timer value used for flow control.
	PLIB_ETH_PauseTimerSet	Sets the Pause Timer value used for flow control.
	PLIB_ETH_ReceiveBufferSizeGet	Gets the Ethernet module receive buffer size.
	PLIB_ETH_ReceiveBufferSizeSet	Sets the Ethernet module receive buffer size.
	PLIB_ETH_ReceiveDisable	Disables the EMAC from receiving frames.
	PLIB_ETH_ReceiveEnable	Enables the EMAC to receive the frames.
	PLIB_ETH_ReceiveIsEnabled	Gets the Ethernet EMAC receive enable status.
	PLIB_ETH_ReTxMaxGet	Gets the EMAC maximum retransmissions.
	PLIB_ETH_ReTxMaxSet	Sets the EMAC maximum retransmissions.
	PLIB_ETH_RMIISpeedGet	Gets the current EMAC RMII speed.
	PLIB_ETH_RMIISpeedSet	Sets EMAC RMII Speed.
	PLIB_ETH_StopInIdleDisable	Ethernet module operation will continue when the device enters Idle mode.
	PLIB_ETH_StopInIdleEnable	Ethernet module operation will stop when the device enters Idle mode.
	PLIB_ETH_StopInIdleIsEnabled	Gets the Ethernet module Stop in Idle mode enable status.

b) Control Functions

	Name	Description
	PLIB_ETH_MCSRxResetDisable	Disables the reset EMAC Control Sub-layer/Receive domain logic.
	PLIB_ETH_MCSRxResetEnable	Enables the reset EMAC Control Sub-layer/Receive domain logic.
	PLIB_ETH_MCSRxResetIsEnabled	Gets the EMAC Control Sub-layer/Receive domain logic reset status.
	PLIB_ETH_MCSTxResetDisable	Disables the reset EMAC Control Sub-layer/Transmit domain logic.
	PLIB_ETH_MCSTxResetEnable	Enables the reset of EMAC Control Sub-layer/Transmit domain logic.
	PLIB_ETH_MCSTxResetIsEnabled	Gets the EMAC Control Sub-layer/Transmit domain logic reset status.
	PLIB_ETH_MIIMReadDataGet	Gets EMAC MIIM management read data after a MII read cycle has completed.
	PLIB_ETH_MIIMReadStart	Initiates an MII management read command.
	PLIB_ETH_MIIMResetDisable	Disables EMAC Reset MII Management.
	PLIB_ETH_MIIMResetEnable	Enables EMAC Reset Media Independent Interface (MII) Management.
	PLIB_ETH_MIIMResetIsEnabled	Gets the EMAC Reset MII Management enable status.
	PLIB_ETH_MIIMScanIncrEnable	Enables EMAC MIIM Scan Increment.
	PLIB_ETH_MIIMScanIncrDisable	Disables the EMAC MIIM Scan Increment.
	PLIB_ETH_MIIMScanIncrIsEnabled	Gets the EMAC MIIM scan increment enable status.
	PLIB_ETH_MIIMScanModeDisable	Disables MIIM scan mode.
	PLIB_ETH_MIIMScanModeEnable	Enables MIIM scan mode.
	PLIB_ETH_MIIMScanModeIsEnabled	Gets the MII management scan enable status.

	PLIB_ETH_MIIMWriteDataSet	Sets the EMAC MIIM write data before initiating an MII write cycle.
	PLIB_ETH_MIIMWriteStart	Initiates an MII management write command.
	PLIB_ETH_MIIResetDisable	Disables the EMAC Soft reset.
	PLIB_ETH_MIIResetEnable	Enables the EMAC MIIM Soft reset.
	PLIB_ETH_MIIResetIsEnabled	Gets EMAC MIIM Soft Reset enable status.
	PLIB_ETH_PHYAddressGet	Gets the EMAC MIIM management PHY address.
	PLIB_ETH_PHYAddressSet	Sets the EMAC MIIM PHY address.
	PLIB_ETH_RegisterAddressGet	Gets the EMAC MIIM management register address.
	PLIB_ETH_RegisterAddressSet	Sets EMAC MIIM register address.
	PLIB_ETH_RMIIResetDisable	Disables EMAC Reset RMII.
	PLIB_ETH_RMIIResetEnable	Enables EMAC Reset RMII.
	PLIB_ETH_RMIIResetIsEnabled	Gets the EMAC Reset RMII enable status.
	PLIB_ETH_RxBufferCountDecrement	Causes the Receive Descriptor Buffer Counter to decrement by 1.
	PLIB_ETH_RxDisable	Disables the Ethernet module receive logic.
	PLIB_ETH_RxEnable	Enables the Ethernet receive logic.
	PLIB_ETH_RxFuncResetDisable	Disables the EMAC reset receive function logic.
	PLIB_ETH_RxFuncResetEnable	Enables the EMAC reset receive function logic.
	PLIB_ETH_RxFuncResetIsEnabled	Gets the EMAC reset receive function status.
	PLIB_ETH_RxIsEnabled	Gets the Ethernet module receive enable status.
	PLIB_ETH_RxPacketDescAddrGet	Gets the address of the next receive descriptor.
	PLIB_ETH_RxPacketDescAddrSet	Sets the Ethernet module receive packet descriptor start address.
	PLIB_ETH_TestPauseDisable	Disables EMAC test pause.
	PLIB_ETH_TestPauseEnable	Enables EMAC test pause.
	PLIB_ETH_TestPauseIsEnabled	Gets the EMAC test pause enable status.
	PLIB_ETH_TxFuncResetDisable	Disables the EMAC Transmit function reset.
	PLIB_ETH_TxFuncResetEnable	Enables the EMAC transmit function reset.
	PLIB_ETH_TxFuncResetIsEnabled	Gets the EMAC Transmit function reset status.
	PLIB_ETH_TxPacketDescAddrGet	Gets the address of the next descriptor to be transmitted.
	PLIB_ETH_TxPacketDescAddrSet	Sets the Ethernet module transmit packet descriptor start address.
	PLIB_ETH_TxRTSDisable	Aborts an Ethernet module transmission.
	PLIB_ETH_TxRTSEnable	Enables the Ethernet transmit request to send.
	PLIB_ETH_TxRTSIsEnabled	Gets the Ethernet module transmit request to send status.
	PLIB_ETH_CRCDisable	Disables EMAC CRC.
	PLIB_ETH_CRCEnable	Enables EMAC CRC.
	PLIB_ETH_CRCIsEnabled	Gets the EMAC CRC enable status.

c) Status Functions

	Name	Description
	PLIB_ETH_DataNotValid	Gets the MII management read data not valid status.
	PLIB_ETH_EthernetIsBusy	Gets the status value of the Ethernet logic busy.
	PLIB_ETH_LinkHasFailed	Gets the MII management link fail status.
	PLIB_ETH_MIIMIsBusy	Gets the MII management busy status.
	PLIB_ETH_MIIMIsScanning	Gets the MII Management scanning status.
	PLIB_ETH_ReceiveIsBusy	Gets the Ethernet receive logic busy status.
	PLIB_ETH_TransmitIsBusy	Gets the status value of the Ethernet transmit logic busy status

d) Filtering Functions

	Name	Description
	PLIB_ETH_HashTableGet	Gets the value of the Hash table.
	PLIB_ETH_HashTableSet	Sets the Ethernet module Hash table with the new value.
	PLIB_ETH_PassAllDisable	Disables the EMAC PassAll.
	PLIB_ETH_PassAllEnable	Enables the EMAC PassAll.
	PLIB_ETH_PassAllIsEnabled	Gets the EMAC PassAll enable status.
	PLIB_ETH_PatternMatchChecksumGet	Gets the value of the pattern match checksum register.
	PLIB_ETH_PatternMatchChecksumSet	Sets the Ethernet module pattern match checksum register with the new value.

	PLIB_ETH_PatternMatchGet	Gets the value of the selected pattern match mask register.
	PLIB_ETH_PatternMatchModeGet	Gets the value of the selected pattern match mask register.
	PLIB_ETH_PatternMatchModeSet	Sets the Ethernet module pattern match mode.
	PLIB_ETH_PatternMatchOffsetGet	Gets the value of the patter match offset register.
	PLIB_ETH_PatternMatchOffsetSet	Sets the Ethernet module patter match offset register with the new value.
	PLIB_ETH_PatternMatchSet	Sets the Ethernet module pattern match mask register with the new value.
	PLIB_ETH_PurePreambleDisable	Disables EMAC pure preamble enforcement.
	PLIB_ETH_PurePreambleEnable	Enables EMAC pure preamble enforcement.
	PLIB_ETH_PurePreambleIsEnabled	Gets EMAC pure preamble enforcement enable status.
	PLIB_ETH_ReceiveFilterDisable	Disables the specified receive filter.
	PLIB_ETH_ReceiveFilterEnable	Enables the specified receive filter.
	PLIB_ETH_ReceiveFilterIsEnable	Disables the specified receive filter.
	PLIB_ETH_StationAddressGet	Gets the selected EMAC station address.
	PLIB_ETH_StationAddressSet	Sets the selected EMAC Station Address.

e) Flow Control Functions

	Name	Description
	PLIB_ETH_AutoFlowControlDisable	Disables the Ethernet module Automatic Flow Control logic.
	PLIB_ETH_AutoFlowControlEnable	Enables the Ethernet Automatic Flow Control logic.
	PLIB_ETH_AutoFlowControlIsEnabled	Gets the Ethernet module Automatic Flow Control status.
	PLIB_ETH_BackPresNoBackoffDisable	Disables EMAC backpressure/no back-off.
	PLIB_ETH_BackPresNoBackoffEnable	Enables EMAC back pressure/no back-off.
	PLIB_ETH_BackPresNoBackoffIsEnabled	Gets the EMAC backpressure/no back-off enable status.
	PLIB_ETH_ManualFlowControlDisable	Disable Ethernet module Manual Flow Control logic.
	PLIB_ETH_ManualFlowControlEnable	Enables the Ethernet Manual Flow Control logic.
	PLIB_ETH_ManualFlowControlIsEnabled	Gets the Ethernet module Manual Flow Control enable status.
	PLIB_ETH_NoBackoffDisable	Disables EMAC no back-offs.
	PLIB_ETH_NoBackoffEnable	Enables EMAC no back-off.
	PLIB_ETH_NoBackoffIsEnabled	Gets the EMAC no back-off enable status.
	PLIB_ETH_RxEmptyWmarkGet	Gets the Ethernet module receive empty watermark.
	PLIB_ETH_RxEmptyWmarkSet	Sets the Ethernet module receive empty water mark.
	PLIB_ETH_RxFullWmarkGet	Gets the Ethernet module to receive a full watermark.
	PLIB_ETH_RxFullWmarkSet	Sets the Ethernet module to receive a full watermark.
	PLIB_ETH_RxPauseDisable	Disables the EMAC receive flow control.
	PLIB_ETH_RxPauseEnable	Enables the EMAC receive flow control.
	PLIB_ETH_RxPauseIsEnabled	Gets the EMAC receive flow pause enable status.
	PLIB_ETH_ShortcutQuantaDisable	Disables EMAC shortcut pause quanta.
	PLIB_ETH_ShortcutQuantaEnable	Enables EMAC shortcut pause quanta.
	PLIB_ETH_ShortcutQuantaIsEnabled	Gets EMAC shortcut pause quanta enable status.
	PLIB_ETH_SimResetDisable	Disables the EMAC simulation reset.
	PLIB_ETH_SimResetEnable	Enables the EMAC simulation reset.
	PLIB_ETH_SimResetIsEnabled	Gets the EMAC simulation reset status.
	PLIB_ETH_TestBackPressDisable	Disables EMAC Test backpressure.
	PLIB_ETH_TestBackPressEnable	Enables EMAC Test backpressure.
	PLIB_ETH_TestBackPressIsEnabled	Gets the EMAC test backpressure enable status.
	PLIB_ETH_TxPauseDisable	Disables the transmission of Pause frames.
	PLIB_ETH_TxPauseEnable	Enables the transmission Pause frames.
	PLIB_ETH_TxPauseIsEnabled	Gets the Ethernet module enable status.

f) Interrupt Functions

	Name	Description
	PLIB_ETH_InterruptClear	Clears the Ethernet module interrupt source status as a group, using a mask.
	PLIB_ETH_InterruptSet	Sets the Ethernet module interrupt source status as a group, using a mask.
	PLIB_ETH_InterruptsGet	Gets the Ethernet module Interrupt Flag register as a group.
	PLIB_ETH_InterruptSourceDisable	Clears the Ethernet module Interrupt Enable register as a group, using a mask.

	PLIB_ETH_InterruptSourceEnable	Sets the Ethernet module Interrupt Enable register in a group, using a mask.
	PLIB_ETH_InterruptSourceIsEnabled	Gets the Ethernet module Interrupt Enable register singularly or as a group.
	PLIB_ETH_InterruptSourcesGet	Returns the entire interrupt enable register.
	PLIB_ETH_InterruptStatusGet	Gets the Ethernet module Interrupt Flag register as a group, using a mask.

g) Statistics Functions

	Name	Description
	PLIB_ETH_AlignErrorCountClear	Sets the count of Ethernet alignment errors to zero.
	PLIB_ETH_AlignErrorCountGet	Gets the count of Ethernet alignment errors.
	PLIB_ETH_FCSErrorCountClear	Sets the value of the Ethernet frame check sequence error to zero.
	PLIB_ETH_FCSErrorCountGet	Gets the count of the frame check sequence error.
	PLIB_ETH_FramesRxdOkCountClear	Sets the value of the Ethernet received frames 'OK' count to zero.
	PLIB_ETH_FramesRxdOkCountGet	Gets the count of the frames frames received successfully.
	PLIB_ETH_FramesTxdOkCountClear	Sets the count of Ethernet Control frames transmitted to zero.
	PLIB_ETH_FramesTxdOkCountGet	Gets the count of Ethernet Control Frames transmitted successfully.
	PLIB_ETH_MultipleCollisionCountClear	Sets the value of the Ethernet multiple collision frame count to zero.
	PLIB_ETH_MultipleCollisionCountGet	Gets the count of the frames transmitted successfully after there was more than one collision.
	PLIB_ETH_RxOverflowCountClear	Sets the value of the Ethernet receive overflow count to zero.
	PLIB_ETH_RxOverflowCountGet	Gets the count of the dropped Ethernet receive frames.
	PLIB_ETH_RxPacketCountGet	Gets the value of the receive packet buffer count.
	PLIB_ETH_SingleCollisionCountClear	Sets the value of the Ethernet single collision frame count to zero.
	PLIB_ETH_SingleCollisionCountGet	Gets the count of the frames transmitted successfully on a second attempt.

h) Feature Existence Functions

	Name	Description
	PLIB_ETH_ExistsAlignmentErrorCount	Identifies whether the AlignmentErrorCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsAutoFlowControl	Identifies whether the AutoFlowControl feature exists on the Ethernet module.
	PLIB_ETH_ExistsCollisionCounts	Identifies whether the CollisionCounts feature exists on the Ethernet module.
	PLIB_ETH_ExistsCollisionWindow	Identifies whether the CollisionWindow feature exists on the Ethernet module.
	PLIB_ETH_ExistsEnable	Identifies whether the Enable feature exists on the Ethernet module.
	PLIB_ETH_ExistsEthernetControllerStatus	Identifies whether the EthernetControllerStatus feature exists on the Ethernet module.
	PLIB_ETH_ExistsFCSErrorCount	Identifies whether the FCSErrorCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsFramesTransmittedOK	Identifies whether the FramesTransmittedOK feature exists on the Ethernet module.
	PLIB_ETH_ExistsFramexReceivedOK	Identifies whether the FramexReceivedOK feature exists on the Ethernet module.
	PLIB_ETH_ExistsHashTable	Identifies whether the HashTable feature exists on the Ethernet module.
	PLIB_ETH_ExistsInterPacketGaps	Identifies whether the InterPacketGaps feature exists on the Ethernet module.
	PLIB_ETH_ExistsInterrupt	Identifies whether the Interrupt feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Configuration	Identifies whether the MAC_Configuration feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Resets	Identifies whether the MAC_Resets feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Testing	Identifies whether the MAC_Testing feature exists on the Ethernet module.
	PLIB_ETH_ExistsManualFlowControl	Identifies whether the ManualFlowControl feature exists on the Ethernet module.
	PLIB_ETH_ExistsMaxFrameLength	Identifies whether the MaxFrameLength feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIM_Config	Identifies whether the MIIM_Config feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIM_Indicators	Identifies whether the MIIM_Indicators feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMAddresses	Identifies whether the MIIMAddresses feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMReadWrite	Identifies whether the MIIMReadWrite feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMScanMode	Identifies whether the MIIMScanMode feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIWriteReadData	Identifies whether the MIIWriteReadData feature exists on the Ethernet module.
	PLIB_ETH_ExistsPatternMatch	Identifies whether the PatternMatch feature exists on the Ethernet module.
	PLIB_ETH_ExistsPauseTimer	Identifies whether the PauseTimer feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveBufferSize	Identifies whether the ReceiveBufferSize feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveFilters	Identifies whether the ReceiveFilters feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveOverflowCount	Identifies whether the ReceiveOverflowCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveWmarks	Identifies whether the ReceiveWmarks feature exists on the Ethernet module.

	PLIB_ETH_ExistsRetransmissionMaximum	Identifies whether the RetransmissionMaximum feature exists on the Ethernet module.
	PLIB_ETH_ExistsRMII_Support	Identifies whether the RMII_Support feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxBufferCountDecrement	Identifies whether the RxBufferCountDecrement feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxEnable	Identifies whether the ReceiveEnable feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxPacketDescriptorAddress	Identifies whether the RxPacketDescriptorAddress feature exists on the Ethernet module.
	PLIB_ETH_ExistsStationAddress	Identifies whether the StationAddress feature exists on the Ethernet module.
	PLIB_ETH_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the Ethernet module.
	PLIB_ETH_ExistsTransmitRTS	Identifies whether the TransmitRTS feature exists on the Ethernet module.
	PLIB_ETH_ExistsTxPacketDescriptorAddress	Identifies whether the TxPacketDescriptorAddress feature exists on the Ethernet module.

i) Data Types and Constants

	Name	Description
	ETH_AUTOPAD_OPTION	Lists the possible automatic padding configurations.
	ETH_INTERRUPT_SOURCES	Lists the possible values of ETH_INTERRUPT_SOURCES.
	ETH_MIIM_CLK	Lists the possible speed selection for the Reduced Media Independent Interface (RMII).
	ETH_PATTERN_MATCH_MODE	Lists the possible pattern match values.
	_ETH_PATTERN_MATCH_MODE_	Lists the possible pattern match values.
	ETH_RECEIVE_FILTER	Lists the possible values of the receive filter.
	ETH_RMII_SPEED	Lists the possible speed selection for the Reduced Media Independent Interface (RMII).

Description

This section describes the Application Programming Interface (API) functions of the Ethernet Peripheral Library.

Refer to each section for a detailed description.

a) Initialization Functions

PLIB_ETH_AutoDetectPadClear Function

Clears the EMAC Auto-Pad option.

File

plib_eth.h

C

```
void PLIB_ETH_AutoDetectPadClear(ETH_MODULE_ID index, ETH_AUTOPAD_OPTION option);
```

Returns

None.

Description

This function sets the EMAC Auto-Pad option. If any auto-pad option other than ETH_AUTOPAD_DISABLED_CHECK_CRC is selected, the CRC must be enabled by PLIB_ETH_CRCEnable.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_AutoDetectPadClear(MY_ETH_INSTANCE, PAD_OPTION);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_AutoDetectPadClear(ETH_MODULE_ID index, ETH_AUTOPAD_OPTION option);

PLIB_ETH_AutoDetectPadGet Function

Gets the current EMAC Auto-Pad setting.

File

plib_eth.h

C

```
ETH_AUTOPAD_OPTION PLIB_ETH_AutoDetectPadGet(ETH_MODULE_ID index);
```

Returns

ETH_AUTOPAD_OPTION.

Description

This function gets the current EMAC Auto-Pad setting.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
autoPadOption = PLIB_ETH_AutoDetectPadGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

ETH_AUTOPAD_OPTION PLIB_ETH_AutoDetectPadGet(ETH_MODULE_ID index);

PLIB_ETH_AutoDetectPadSet Function

Sets the EMAC Auto-Pad option.

File

plib_eth.h

C

```
void PLIB_ETH_AutoDetectPadSet(ETH_MODULE_ID index, ETH_AUTOPAD_OPTION option);
```

Returns

None.

Description

This function sets the EMAC Auto-Pad option. If any auto-pad option other than ETH_AUTOPAD_DISABLED_CHECK_CRC is selected, the CRC must be enabled by PLIB_ETH_CRCEnable.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_AutoDetectPadSet(MY_ETH_INSTANCE, PAD_OPTION);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_AutoDetectPadSet(ETH_MODULE_ID index, ETH_AUTOPAD_OPTION option);

PLIB_ETH_BackToBackIPGGet Function

Gets the EMAC back-to-back interpacket gap.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_BackToBackIPGGet(ETH_MODULE_ID index);
```

Returns

- backToBackIPGValue - Back-to-back interpacket gap (7 bits)

Description

This function gets the EMAC back-to-back interpacket gap.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_BackToBackIPGGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_BackToBackIPGGet(ETH_MODULE_ID index)

PLIB_ETH_BackToBackIPGSet Function

Sets the EMAC back-to-back interpacket gap.

File

plib_eth.h

C

```
void PLIB_ETH_BackToBackIPGSet(ETH_MODULE_ID index, uint8_t backToBackIPGValue);
```

Returns

None.

Description

This function sets the EMAC back-to-back interpacket gap.

Remarks

In Full-Duplex mode, the register value should be the desired period in nibble times minus 3. In Full-Duplex, the recommended setting is 0x15.

In Half-Duplex mode, the register value should be the desired period in nibble times minus 6. n Half-Duplex, the recommended setting is 0x12.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_BackToBackIPGSet(MY_ETH_INSTANCE, backToBackIPGValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
backToBackIPGValue	Back-to-back interpacket gap

Function

void PLIB_ETH_BackToBackIPGSet(ETH_MODULE_ID index, uint8_t backToBackIPGValue)

PLIB_ETH_CollisionWindowGet Function

Gets the EMAC collision window.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_CollisionWindowGet(ETH_MODULE_ID index);
```

Returns

- collisionWindowValue - A uint8_t (6-bit) value.

Description

This function gets the EMAC collision window.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_CollisionWindowGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_CollisionWindowGet(ETH_MODULE_ID index)

PLIB_ETH_CollisionWindowSet Function

Sets the EMAC collision window.

File

plib_eth.h

C

```
void PLIB_ETH_CollisionWindowSet(ETH_MODULE_ID index, uint8_t collisionWindowValue);
```

Returns

None.

Description

This function sets the EMAC collision window.

Remarks

This is a programmable field representing the slot time or collision window during which collisions occur in properly configured networks. Since the collision window starts at the beginning of transmission, the preamble and SFD is included. Its default of 0x37 (55d) corresponds to the count of frame bytes at the end of the window.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_CollisionWindowSet(MY_ETH_INSTANCE, collisionWindowValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
collisionWindowValue	Collision window

Function

void PLIB_ETH_CollisionWindowSet(ETH_MODULE_ID index, uint8_t collisionWindowValue)

PLIB_ETH_DelayedCRCDisable Function

Disables EMAC delayed CRC.

File

plib_eth.h

C

```
void PLIB_ETH_DelayedCRCDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC delayed CRC. No proprietary header exists.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_DelayedCRCDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_DelayedCRCDisable(ETH_MODULE_ID index)

PLIB_ETH_DelayedCRCEnable Function

Enables EMAC delayed CRC.

File

plib_eth.h

C

```
void PLIB_ETH_DelayedCRCEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC delayed CRC. A proprietary packet header exists. Four bytes of the packet header are ignored by the CRC function.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_DelayedCRCEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_DelayedCRCEnable(ETH_MODULE_ID index)

PLIB_ETH_DelayedCRCIsEnabled Function

Gets the EMAC delayed CRC enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_DelayedCRCIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - A proprietary header exists. Four bytes of packet header are ignored by the CRC function.
- false - No proprietary header

Description

This function returns the EMAC delayed CRC enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_DelayedCRCIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_DelayedCRCIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_Disable Function

Disables the Ethernet module.

File

plib_eth.h

C

```
void PLIB_ETH_Disable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the Ethernet module.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_Disable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_Disable(ETH_MODULE_ID index)

PLIB_ETH_Enable Function

Enables the Ethernet module.

File

plib_eth.h

C

```
void PLIB_ETH_Enable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the Ethernet module.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_Enable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_Enable(ETH_MODULE_ID index)

PLIB_ETH_ExcessDeferDisable Function

Disables EMAC excess defer.

File

plib_eth.h

C

```
void PLIB_ETH_ExcessDeferDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC excess defer. The EMAC will abort when the excessive deferral limit is reached.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ExcessDeferDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ExcessDeferDisable(ETH_MODULE_ID index)

PLIB_ETH_ExcessDeferEnable Function

Enables EMAC excess defer.

File

plib_eth.h

C

```
void PLIB_ETH_ExcessDeferEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC excess defer. The EMAC will defer to the carrier indefinitely.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ExcessDeferEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ExcessDeferEnable(ETH_MODULE_ID index)

PLIB_ETH_ExcessDeferIsEnabled Function

Gets the EMAC excess defer enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_ExcessDeferIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC will defer to carrier indefinitely as per the Standard
- false - The EMAC will abort when the excessive deferral limit is reached

Description

This function gets the EMAC excess defer enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_ExcessDeferIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_ExcessDeferIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_FrameLengthCheckDisable Function

Disables the EMAC frame length check.

File

plib_eth.h

C

```
void PLIB_ETH_FrameLengthCheckDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC frame length check.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FrameLengthCheckDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_FrameLengthCheckDisable(ETH_MODULE_ID index)

PLIB_ETH_FrameLengthCheckEnable Function

Enables the EMAC frame length check.

File

plib_eth.h

C

```
void PLIB_ETH_FrameLengthCheckEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC frame length check. Both transmit and receive frame lengths are compared to the Length/Type field. If the Length/Type field represents a length, the check is performed. Mismatches are reported on the transmit/receive statistics vector.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FrameLengthCheckEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_FrameLengthCheckEnable(ETH_MODULE_ID index)

PLIB_ETH_FrameLengthCheckIsEnabled Function

Gets the EMAC frame length check status.

File

plib_eth.h

C

```
bool PLIB_ETH_FrameLengthCheckIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Both transmit and receive frame lengths are compared to the Length/Type field. If the Length/Type field represents a length then the check is performed. Mismatches are reported on the transmit/receive statistics vector.
- false - Length/Type field check is not performed

Description

This function returns the EMAC frame length check status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_FrameLengthCheckIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_FrameLengthCheckIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_FullDuplexDisable Function

Disables the EMAC full duplex operation.

File

plib_eth.h

C

```
void PLIB_ETH_FullDuplexDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC full-duplex operation, which results in the EMAC operating in half-duplex.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FullDuplexDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_FullDuplexDisable(ETH_MODULE_ID index)

PLIB_ETH_FullDuplexEnable Function

Enables the EMAC full duplex operation.

File

plib_eth.h

C

```
void PLIB_ETH_FullDuplexEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC full duplex operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FullDuplexEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_FullDuplexEnable(ETH_MODULE_ID index)

PLIB_ETH_FullDuplexIsEnabled Function

Gets the EMAC full duplex enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_FullDuplexIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC operates in Full-Duplex mode
- false - The EMAC operates in Half-Duplex mode

Description

This function returns the EMAC full duplex enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_FullDuplexIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_FullDuplexIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_HugeFrameDisable Function

Disables the EMAC from receiving huge frames.

File

plib_eth.h

C

```
void PLIB_ETH_HugeFrameDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC from receiving huge frames.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_HugeFrameDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_HugeFrameDisable(ETH_MODULE_ID index)

PLIB_ETH_HugeFrameEnable Function

Enables the EMAC to receive huge frames.

File

plib_eth.h

C

```
void PLIB_ETH_HugeFrameEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC receiving the frames of any length.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_HugeFrameEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_HugeFrameEnable(ETH_MODULE_ID index)

PLIB_ETH_HugeFrameIsEnabled Function

Gets the EMAC huge frame enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_HugeFrameIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Frames of any length are transmitted and received
- false - Huge frames are not allowed for receive or transmit

Description

This function returns the EMAC huge frame enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_HugeFrameIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_HugeFrameIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_IsEnabled Function

Gets the Ethernet module enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_IsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Ethernet module is enabled
- false - Ethernet module is disabled

Description

This function returns the Ethernet module enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_IsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_IsEnabled(ETH_MODULE_ID index)

PLIB_ETH_LongPreambleDisable Function

Disables EMAC long preamble enforcement.

File

plib_eth.h

C

```
void PLIB_ETH_LongPreambleDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC long preamble enforcement. The EMAC allows any length preamble.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_LongPreambleDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_LongPreambleDisable(ETH_MODULE_ID index)

PLIB_ETH_LongPreambleEnable Function

Enables EMAC long preamble enforcement.

File

plib_eth.h

C

```
void PLIB_ETH_LongPreambleEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC long preamble enforcement. The EMAC only allows receive packets which contain preamble fields less than 12 bytes in length.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_LongPreambleEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_LongPreambleEnable(ETH_MODULE_ID index)

PLIB_ETH_LongPreambleIsEnabled Function

Gets the EMAC long preamble enforcement enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_LongPreambleIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC only allows receive packets, which contain preamble fields less than 12 bytes in length
- false - The EMAC allows any length preamble

Description

This function gets the EMAC long preamble enforcement enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_LongPreambleIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_LongPreambleIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_LoopbackDisable Function

Disables the EMAC loopback logic.

File

plib_eth.h

C

```
void PLIB_ETH_LoopbackDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC loopback. EMAC resumes normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_LoopbackDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_LoopbackDisable(ETH_MODULE_ID index)

PLIB_ETH_LoopbackEnable Function

Enables the EMAC loopback logic.

File

plib_eth.h

C

```
void PLIB_ETH_LoopbackEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC loopback logic. The EMAC transmit interface is looped back to the EMAC receive interface.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_LoopbackEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_LoopbackEnable(ETH_MODULE_ID index)

PLIB_ETH_LoopbackIsEnabled Function

Gets the EMAC Loopback interface enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_LoopbackIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - EMAC transmit interface is looped back to the EMAC receive interface

- false - EMAC normal operation

Description

This function gets the EMAC Loopback interface enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_LoopbackIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_LoopbackIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_MaxFrameLengthGet Function

Gets the EMAC maximum frame length.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_MaxFrameLengthGet(ETH_MODULE_ID index);
```

Returns

- MaxFrameLength - Maximum frame length

Description

This function gets the EMAC maximum frame length.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_MaxFrameLengthGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_MaxFrameLengthGet(ETH_MODULE_ID index)

PLIB_ETH_MaxFrameLengthSet Function

Sets the EMAC maximum frame length.

File

plib_eth.h

C

```
void PLIB_ETH_MaxFrameLengthSet(ETH_MODULE_ID index, uint16_t MaxFrameLength);
```

Returns

None.

Description

This function sets the EMAC maximum frame length.

Remarks

This field resets to 0x05EE, which represents a maximum receive frame of 1518 bytes. An untagged maximum size Ethernet frame is 1518 bytes. A tagged frame adds four octets for a total of 1522 bytes. If a shorter/longer maximum length restriction is desired, program this 16-bit field.

If a proprietary header is allowed, this field should be adjusted accordingly. For example, if 4-byte headers are prepended to frames, MACMAXF could be set to 1527 bytes. This would allow the maximum VLAN tagged frame plus the 4-byte header.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MaxFrameLengthSet(MY_ETH_INSTANCE, MaxFrameLength);
```

Parameters

Parameters	Description
index	Identifier for the device instance
MaxFrameLength	Maximum frame length

Function

void PLIB_ETH_MaxFrameLengthSet(ETH_MODULE_ID index, uint16_t MaxFrameLength)

PLIB_ETH_MIIMClockGet Function

Gets the current EMAC MIIM clock selection.

File

plib_eth.h

C

```
ETH_MIIM_CLK PLIB_ETH_MIIMClockGet(ETH_MODULE_ID index);
```

Returns

- MIIMClock - Of type PLIB_ETH_MIIM_CLK

Description

This function returns the current EMAC MIIM clock selection.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
clkstat = PLIB_ETH_MIIMClockGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

ETH_MII_CLK PLIB_ETH_MIIMClockGet(ETH_MODULE_ID index)

PLIB_ETH_MIIMClockSet Function

Sets the EMAC MIM clock selection.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMClockSet(ETH_MODULE_ID index, ETH_MIIM_CLK MIIMClock);
```

Returns

None.

Description

This function sets the EMAC MIIM clock selection.

Remarks

This field is used by the clock divide logic in creating the MII Management Clock (MDC), which the IEEE 802.3 Specification defines to be no faster than 2.5 MHz. Some PHYs support clock rates up to 12.5 MHz.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMClockSet(MY_ETH_INSTANCE, MIIMClock);
```

Parameters

Parameters	Description
index	Identifier for the device instance
MIIMClock	of type ETH_MIIM_CLK - the system clock divisor for MII

Function

void PLIB_ETH_MIIMClockSet(ETH_MODULE_ID index, ETH_MIIM_CLK MIIMClock)

PLIB_ETH_MIIMNoPreDisable Function

Disables EMAC No Preamble (allows preamble).

File

plib_eth.h

C

```
void PLIB_ETH_MIIMNoPreDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC No preamble (allows preamble). Normal read/write cycles are performed.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMNoPreDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMNoPreDisable(ETH_MODULE_ID index)

PLIB_ETH_MIIMNoPreEnable Function

Enables EMAC MIIM No Preamble (suppresses preamble).

File

plib_eth.h

C

```
void PLIB_ETH_MIIMNoPreEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC MIIM No Preamble (suppresses preamble). The MII Management will perform read/write cycles without the 32-bit preamble field. Some PHYs support suppressed preamble.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMNoPreEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMNoPreEnable(ETH_MODULE_ID index)

PLIB_ETH_MIIMNoPreIsEnabled Function

Gets the EMAC MIIM No Preamble enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIMNoPreIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The MII Management will perform read/write cycles without the 32-bit preamble field. Some PHYs support suppressed preamble.
- false - Normal read/write cycles are performed

Description

This function gets the EMAC MIIM No Preamble enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MIIMNoPreIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIMNoPreIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_NonBackToBackIPG1Get Function

Gets the EMAC non-back-to-back interpacket gap register 1.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_NonBackToBackIPG1Get(ETH_MODULE_ID index);
```

Returns

- nonBackToBackIPGValue - Non-back-to-back interpacket gap (7 bits)

Description

This function gets the EMAC non-back-to-back interpacket gap register 1.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_NonBackToBackIPG1Get(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_NonBackToBackIPG1Get(ETH_MODULE_ID index)

PLIB_ETH_NonBackToBackIPG1Set Function

Sets the EMAC non-back-to-back interpacket gap register 1.

File

plib_eth.h

C

```
void PLIB_ETH_NonBackToBackIPG1Set(ETH_MODULE_ID index, uint8_t nonBackToBackIPGValue);
```

Returns

None.

Description

This function sets the EMAC non-back-to-back interpacket gap register 1. A value of 0x0C is recommended.

Remarks

This is a programmable field representing the optional carrierSense window referenced in the IEEE 802.3 Specification. If a carrier is detected during the timing of IPGR1, the MAC defers to the carrier. If, however, the carrier comes after IPGR1, the MAC continues timing IPGR2 and transmits, knowingly causing a collision, thereby ensuring fair access to the medium. Its range of values is 0x0 to IPGR2. Its recommended value is 0xC (12d).

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_NonBackToBackIPG1Set(MY_ETH_INSTANCE, nonBackToBackIPGValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
nonBackToBackIPGValue	Non-back-to-back interpacket gap

Function

void PLIB_ETH_NonBackToBackIPG1Set(ETH_MODULE_ID index, uint8_t nonBackToBackIPGValue)

PLIB_ETH_NonBackToBackIPG2Get Function

Gets the EMAC non-back-to-back interpacket gap register 2.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_NonBackToBackIPG2Get(ETH_MODULE_ID index);
```

Returns

- nonBackToBackIPGValue - Non-back-to-back interpacket gap (7 bits)

Description

This function gets the EMAC non-back-to-back interpacket gap register 2.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_NonBackToBackIPG2Get(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_NonBackToBackIPG2Get(ETH_MODULE_ID index)

PLIB_ETH_NonBackToBackIPG2Set Function

Sets the EMAC non-back-to-back interpacket gap register 2.

File

plib_eth.h

C

```
void PLIB_ETH_NonBackToBackIPG2Set(ETH_MODULE_ID index, uint8_t nonBackToBackIPGValue);
```

Returns

None.

Description

This function sets the EMAC non-back-to-back interpacket gap register 2. A value of 0x12 is recommended.

Remarks

This is a programmable field representing the non-back-to-back Inter-Packet-Gap. Its recommended value is 0x12 (18d), which represents the minimum IPG of 0.96 us (in 100 Mbps) or 9.6 us (in 10 Mbps).

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_NonBackToBackIPG2Set(MY_ETH_INSTANCE, nonBackToBackIPGValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
nonBackToBackIPGValue	Non-back-to-back interpacket gap

Function

void PLIB_ETH_NonBackToBackIPG2Set(ETH_MODULE_ID index, uint8_t nonBackToBackIPGValue)

PLIB_ETH_PauseTimerGet Function

Gets the Pause Timer value used for flow control.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_PauseTimerGet(ETH_MODULE_ID index);
```

Returns

- PauseTimerValue - Pause Timer Value

Description

This function gets the Pause Timer value used for flow control.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PauseTimerValue = PLIB_ETH_PauseTimerGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_PauseTimerGet(ETH_MODULE_ID index)

PLIB_ETH_PauseTimerSet Function

Sets the Pause Timer value used for flow control.

File

plib_eth.h

C

```
void PLIB_ETH_PauseTimerSet(ETH_MODULE_ID index, uint16_t PauseTimerValue);
```

Returns

None.

Description

This function sets the Pause Timer value used for flow control.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Write to the Pause Timer register before enabling the receiver. Call this function before calling PLIB_ETH_ReceiveEnable.

Example

```
PLIB_ETH_PauseTimerSet(MY_ETH_INSTANCE, PauseTimerValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
PauseTimerValue	used for Flow Control

Function

void PLIB_ETH_PauseTimerSet(ETH_MODULE_ID index, uint16_t PauseTimerValue)

PLIB_ETH_ReceiveBufferSizeGet Function

Gets the Ethernet module receive buffer size.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_ReceiveBufferSizeGet(ETH_MODULE_ID index);
```

Returns

ReceiveBufferSize - receive buffer size divided by 16

Description

This function gets the Ethernet receive buffer size. The buffer size is set from 16 bytes up to 2032 bytes in increments of 16 bytes each.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
size = PLIB_ETH_ReceiveBufferSizeGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_ReceiveBufferSizeGet(ETH_MODULE_ID index)

PLIB_ETH_ReceiveBufferSizeSet Function

Sets the Ethernet module receive buffer size.

File

plib_eth.h

C

```
void PLIB_ETH_ReceiveBufferSizeSet(ETH_MODULE_ID index, uint8_t ReceiveBufferSize);
```

Returns

None.

Description

This function sets the Ethernet receive buffer size. The buffer size is set from 16 bytes up to 2032 bytes in increments of 16 bytes each.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ReceiveBufferSizeSet(MY_ETH_INSTANCE, ReceiveBufferSize/16 );
```

Parameters

Parameters	Description
index	Identifier for the device instance
ReceiveBufferSize	In units of 16 bytes, must be a value of 0x01 to 0x7F, 0x00 is invalid

Function

void PLIB_ETH_ReceiveBufferSizeSet(ETH_MODULE_ID index, uint8_t ReceiveBufferSize)

PLIB_ETH_ReceiveDisable Function

Disables the EMAC from receiving frames.

File

plib_eth.h

C

```
void PLIB_ETH_ReceiveDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC from receiving frames.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ReceiveDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ReceiveDisable(ETH_MODULE_ID index)

PLIB_ETH_ReceiveEnable Function

Enables the EMAC to receive the frames.

File

plib_eth.h

C

```
void PLIB_ETH_ReceiveEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC to receive the frames.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ReceiveEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ReceiveEnable(ETH_MODULE_ID index)

PLIB_ETH_ReceiveIsEnabled Function

Gets the Ethernet EMAC receive enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_ReceiveIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Enable the EMAC to receive frames
- false - Disable the EMAC to receive frames

Description

This function gets the Ethernet EMAC receive enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_ReceiveIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_ReceiveIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_ReTxMaxGet Function

Gets the EMAC maximum retransmissions.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_ReTxMaxGet(ETH_MODULE_ID index);
```

Returns

- retransmitMax - Maximum number of retransmissions

Description

This function gets the EMAC maximum retransmissions.

Remarks

The maximum number of attempts is limited to 0x0F.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_ReTxMaxGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_ReTxMaxGet(ETH_MODULE_ID index)

PLIB_ETH_ReTxMaxSet Function

Sets the EMAC maximum retransmissions.

File

plib_eth.h

C

```
void PLIB_ETH_ReTxMaxSet(ETH_MODULE_ID index, uint16_t retransmitMax);
```

Returns

None.

Description

This function sets the EMAC maximum retransmissions.

Remarks

This is a programmable field specifying the number of retransmission attempts following a collision before aborting the packet due to excessive collisions. The IEEE 802.3 Specification standard specifies the maximum number of attempts (attemptLimit) to be 0xF (15d). Its default is 000.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ReTxMaxSet(retransmitMax);
```

Parameters

Parameters	Description
index	Identifier for the device instance
retransmitMax	Maximum number of retransmissions

Function

void PLIB_ETH_ReTxMaxSet(ETH_MODULE_ID index, uint8_t retransmitMax)

PLIB_ETH_RMIISpeedGet Function

Gets the current EMAC RMII speed.

File

plib_eth.h

C

```
ETH_RMII_SPEED PLIB_ETH_RMIISpeedGet(ETH_MODULE_ID index);
```

Returns

- RMII_100Mbps - RMII running at 100 Mbps
- RMII_10Mbps - RMII running at 10 Mbps

Description

This function gets the current EMAC RMII speed.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RMIISpeedGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

ETH_RMII_SPEED PLIB_ETH_RMIISpeedGet(ETH_MODULE_ID index)

PLIB_ETH_RMIISpeedSet Function

Sets EMAC RMII Speed.

File

plib_eth.h

C

```
void PLIB_ETH_RMIISpeedSet(ETH_MODULE_ID index, ETH_RMII_SPEED RMIISpeed);
```

Returns

None.

Description

This function sets EMAC RMII speed. RMII speed can be either RMII_100Mbps or RMII_10Mbps.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RMIISpeedSet(MY_ETH_INSTANCE, ETH_RMII_SPEED RMIISpeed);
```

Parameters

Parameters	Description
index	Identifier for the device instance
RMIISpeed	RMII_100Mbps or RMII_10Mbps of type ETH_RMII_SPEED

Function

void PLIB_ETH_RMIISpeedSet(ETH_MODULE_ID index, ETH_RMII_SPEED RMIISpeed)

PLIB_ETH_StopInIdleDisable Function

Ethernet module operation will continue when the device enters Idle mode.

File

plib_eth.h

C

```
void PLIB_ETH_StopInIdleDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function directs the Ethernet module to continue operation when the device enters Idle mode.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_StopInIdleDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_StopInIdleDisable(ETH_MODULE_ID index)

PLIB_ETH_StopInIdleEnable Function

Ethernet module operation will stop when the device enters Idle mode.

File

plib_eth.h

C

```
void PLIB_ETH_StopInIdleEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function directs the Ethernet module to stop operation when the device enters Idle mode.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_StopInIdleEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_StopInIdleEnable(ETH_MODULE_ID index)

PLIB_ETH_StopInIdleIsEnabled Function

Gets the Ethernet module Stop in Idle mode enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_StopInIdleIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Ethernet module transfers stop during Idle mode
- false - Ethernet module transfers continue to run during Idle mode

Description

This function returns the Ethernet module Stop in Idle mode enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_StopInIdleIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_StopInIdleIsEnabled(ETH_MODULE_ID index)

b) Control Functions

PLIB_ETH_MCSRxResetDisable Function

Disables the reset EMAC Control Sub-layer/Receive domain logic.

File

plib_eth.h

C

```
void PLIB_ETH_MCSRxResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the reset EMAC Control Sub-layer/Receive domain logic. The MCS/Receive domain logic is released from reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MCSRxResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MCSRxResetDisable(ETH_MODULE_ID index)

PLIB_ETH_MCSRxResetEnable Function

Enables the reset EMAC Control Sub-layer/Receive domain logic.

File

plib_eth.h

C

```
void PLIB_ETH_MCSRxResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the reset EMAC Control Sub-layer/Receive domain logic. The MCS/Receive domain logic is held in reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MCSRxResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MCSRxResetEnable(ETH_MODULE_ID index)

PLIB_ETH_MCSRxResetIsEnabled Function

Gets the EMAC Control Sub-layer/Receive domain logic reset status.

File

plib_eth.h

C

```
bool PLIB_ETH_MCSRxResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - EMAC Control Sub-layer/Receive domain logic is in reset
- false - EMAC Control Sub-layer/Receive domain logic is not in reset

Description

This function gets the EMAC Control Sub-layer/Receive domain logic reset status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MCSRxResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MCSRxResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_MCSTxResetDisable Function

Disables the reset EMAC Control Sub-layer/Transmit domain logic.

File

plib_eth.h

C

```
void PLIB_ETH_MCSTxResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the reset EMAC Control Sub-layer/Transmit domain logic. The MCS/Transmit domain logic is released from reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MCSTxResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MCSTxResetDisable(ETH_MODULE_ID index)

PLIB_ETH_MCSTxResetEnable Function

Enables the reset of EMAC Control Sub-layer/Transmit domain logic.

File

plib_eth.h

C

```
void PLIB_ETH_MCSTxResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function resets the EMAC Control Sub-layer/Transmit domain logic. The MCS/Transmit domain logic is held in reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MCSTxResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MCSTxResetEnable(ETH_MODULE_ID index)

PLIB_ETH_MCSTxResetIsEnabled Function

Gets the EMAC Control Sub-layer/Transmit domain logic reset status.

File

plib_eth.h

C

```
bool PLIB_ETH_MCSTxResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - EMAC Control Sub-layer/Transmit domain logic is held in reset
- false - EMAC Control Sub-layer/Transmit domain logic is not in reset

Description

This function gets the EMAC Control Sub-layer/Transmit domain logic reset status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MCSTxResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MCSTxResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_MIIMReadDataGet Function

Gets EMAC MIIM management read data after a MII read cycle has completed.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_MIIMReadDataGet(ETH_MODULE_ID index);
```

Returns

- readData - MII read data

Description

This function gets EMAC MIIM management read data after a MII read cycle has completed.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
readData = PLIB_ETH_MIIMReadDataGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_MIIMReadDataGet(ETH_MODULE_ID index)

PLIB_ETH_MIIMReadStart Function

Initiates an MII management read command.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMReadStart(ETH_MODULE_ID index);
```

Returns

None.

Description

This function initiates an MII read command. The MII Management module will perform a single read cycle. To get data, use PLIB_ETH_MIIReadDataGet.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
while (PLIB_ETH_MIIMIsBusy(MY_ETH_INSTANCE))
{
    //Wait until MII Link is not busy
}
PLIB_ETH_MIIReadStart(MY_ETH_INSTANCE);
while (PLIB_ETH_MIIReadIsDataNotValid(MY_ETH_INSTANCE))
{
    //Wait until read is complete
}
data = PLIB_ETH_MIIReadDataGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMReadStart(ETH_MODULE_ID index)

PLIB_ETH_MIIMResetDisable Function

Disables EMAC Reset MII Management.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC Reset MII Management. EMAC will resume normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMResetDisable(ETH_MODULE_ID index)

PLIB_ETH_MIIMResetEnable Function

Enables EMAC Reset Media Independent Interface (MII) Management.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC Reset MII Management and holds the MII Management module in reset while enabled.

Remarks

MII Management held in Reset after This function is called. Disable ResetMIIanagement to return to normal operation.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMResetEnable(ETH_MODULE_ID index)

PLIB_ETH_MIIMResetIsEnabled Function

Gets the EMAC Reset MII Management enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIMResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Reset the MII Management module
- false - Normal operation

Description

This function gets the EMAC Reset MII Management enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MIIMResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIMResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_MIIMScanIncrEnable Function

Enables EMAC MIIM Scan Increment.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMScanIncrEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC MIIM Scan Increment. The MII Management module will perform read cycles across a range of PHYs. The read cycles will start from address 1 through the value set in the PHY address register.

Remarks

The read cycles will start at PHY address 1 and continue through the value set for as the PHY address register.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMScanIncrEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMScanIncrEnable(ETH_MODULE_ID index)

PLIB_ETH_MIIMScanIncrDisable Function

Disables the EMAC MIIM Scan Increment.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMScanIncrDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC MIIM Scan Increment. Allows continuous reads of the same PHY.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMScanIncrDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMScanIncrDisable(ETH_MODULE_ID index)

PLIB_ETH_MIIMScanIncrIsEnabled Function

Gets the EMAC MIIM scan increment enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIMScanIncrIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The MII Management module will perform read cycles across a range of PHYs. The read cycles will start from address 1 through the value set in the PHY address register.
- false - Continuous reads of the same PHY

Description

This function gets the EMAC MIIM scan increment enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
ScanIncrement = PLIB_ETH_MIIMScanIncrIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIMScanIncrIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_MIIMScanModeDisable Function

Disables MIIM scan mode.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMScanModeDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables MIIM scan mode. Scan is disabled for Normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMScanModeDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMScanModeDisable(ETH_MODULE_ID index)

PLIB_ETH_MIIMScanModeEnable Function

Enables MIIM scan mode.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMScanModeEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables MIIM scan mode. The MII Management module will perform read cycles continuously. (Useful for monitoring the Link Fail.)

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIMScanModeEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMScanModeEnable(ETH_MODULE_ID index)

PLIB_ETH_MIIMScanModeIsEnabled Function

Gets the MII management scan enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIMScanModeIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The MII Management module will perform read cycles continuously (for example, useful for monitoring the Link Fail)
- false - Normal operation

Description

This function returns the MII management scan enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MIIMScanModeIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIMScanModeIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_MIIMWriteDataSet Function

Sets the EMAC MIIM write data before initiating an MII write cycle.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMWriteDataSet(ETH_MODULE_ID index, uint16_t writeData);
```

Returns

None.

Description

This function sets the EMAC MIIM write data before initiating write cycle.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Prior to a call to this routine, the PHY and Register addresses should be set using PLIB_ETH_MIIPHYAddressSet and PLIB_ETH_MIIRegisterAddressSet.

Example

```
PLIB_ETH_MIIMWriteDataSet(MY_ETH_INSTANCE, WriteData);
```

Parameters

Parameters	Description
index	Identifier for the device instance
writeData	MII write data

Function

void PLIB_ETH_MIIMWriteDataSet(ETH_MODULE_ID index, uint16_t writeData)

PLIB_ETH_MIIMWriteStart Function

Initiates an MII management write command.

File

plib_eth.h

C

```
void PLIB_ETH_MIIMWriteStart(ETH_MODULE_ID index);
```

Returns

None.

Description

This function initiates an MII management read command. The MII Management module will perform a write cycle.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

The PHY address and MII register address must be configured before a write using PLIB_ETH_MIIPHYAddressSet(MY_ETH_INSTANCE, phyAddr) and PLIB_ETH_MIIRegisterAddressSet(MY_ETH_INSTANCE, regAddr)

Data to be written must be first loaded into the MII write register using PLIB_ETH_MIIMWriteDataSet(MY_ETH_INSTANCE, writeData)

Example

```
PLIB_ETH_MIIMWriteDataSet(MY_ETH_INSTANCE, dataToWrite);
PLIB_ETH_MIIMWriteStart(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIMWriteStart(ETH_MODULE_ID index)

PLIB_ETH_MIIResetDisable Function

Disables the EMAC Soft reset.

File

plib_eth.h

C

```
void PLIB_ETH_MIIResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC MIIM Soft reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIResetDisable(ETH_MODULE_ID index)

PLIB_ETH_MIIResetEnable Function

Enables the EMAC MIIM Soft reset.

File

plib_eth.h

C

```
void PLIB_ETH_MIIResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC MIIM soft reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MIIResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_MIIResetEnable(ETH_MODULE_ID index)

PLIB_ETH_MIIResetIsEnabled Function

Gets EMAC MIIM Soft Reset enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - MAC MII is in reset

- false - MAC MII is not in reset

Description

This function gets EMAC MIIM Soft reset enable status. By default this bit is set to '1'.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MIIResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_PHYAddressGet Function

Gets the EMAC MIIM management PHY address.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_PHYAddressGet(ETH_MODULE_ID index);
```

Returns

- phyAddr - A 5-bit address of the PHY

Description

This function gets the EMAC MIIM management PHY address.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
phyAddr = PLIB_ETH_PHYAddressGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_PHYAddressGet(ETH_MODULE_ID index)

PLIB_ETH_PHYAddressSet Function

Sets the EMAC MIIM PHY address.

File

plib_eth.h

C

```
void PLIB_ETH_PHYAddressSet(ETH_MODULE_ID index, uint8_t phyAddr);
```

Returns

None.

Description

This function sets the EMAC MIIM PHY address. This field represents the 5-bit PHY Address field of Management cycles. Up to 31 PHYs can be addressed (0 is reserved).

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_PHYAddressSet(MY_ETH_INSTANCE, PhyAddr);
```

Parameters

Parameters	Description
index	Identifier for the device instance
phyAddr	A 5-bit address of the PHY

Function

void PLIB_ETH_PHYAddressSet(ETH_MODULE_ID index, uint8_t phyAddr)

PLIB_ETH_RegisterAddressGet Function

Gets the EMAC MIIM management register address.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_RegisterAddressGet(ETH_MODULE_ID index);
```

Returns

- regAddr - The (5-bit) address of the MII Registers

Description

This function gets the EMAC MIIM management register address.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
regAddr = PLIB_ETH_RegisterAddressGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_RegisterAddressGet(ETH_MODULE_ID index)

PLIB_ETH_RegisterAddressSet Function

Sets EMAC MIIM register address.

File

plib_eth.h

C

```
void PLIB_ETH_RegisterAddressSet(ETH_MODULE_ID index, uint8_t regAddr);
```

Returns

None.

Description

This function sets the EMAC MIIM register address. Up to 32 registers may be accessed.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RegisterAddressSet(MY_ETH_INSTANCE, regAddr);
```

Parameters

Parameters	Description
index	Identifier for the device instance
regAddr	The (5-bit) address of the MII Registers.

Function

void PLIB_ETH_RegisterAddressSet(ETH_MODULE_ID index, uint8_t regAddr)

PLIB_ETH_RMIIResetDisable Function

Disables EMAC Reset RMII.

File

plib_eth.h

C

```
void PLIB_ETH_RMIIResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC Reset RMII for normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RMIIResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RMIIResetDisable(ETH_MODULE_ID index)

PLIB_ETH_RMIIResetEnable Function

Enables EMAC Reset RMII.

File

plib_eth.h

C

```
void PLIB_ETH_RMIIResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC Reset RMII.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RMIIResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RMIIResetEnable(ETH_MODULE_ID index)

PLIB_ETH_RMIIResetIsEnabled Function

Gets the EMAC Reset RMII enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_RMIIResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Reset the EMAC RMII module
- false - Normal operation

Description

This function gets the EMAC Reset RMII enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_RMIIResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_RMIIResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_RxBufferCountDecrement Function

Causes the Receive Descriptor Buffer Counter to decrement by 1.

File

plib_eth.h

C

```
void PLIB_ETH_RxBufferCountDecrement(ETH_MODULE_ID index);
```

Returns

None.

Description

This function causes the Receive Descriptor Buffer Counter to decrement by 1.

Remarks

Hardware increments the receive buffer counter and software decrements it. If the receive buffer counter is incremented by the receive logic at the same time, the receive buffer counter will appear unchanged.

Always reads as '0', so there is no get value routine.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxBufferCountDecrement(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxBufferCountDecrement(ETH_MODULE_ID index)

PLIB_ETH_RxDisable Function

Disables the Ethernet module receive logic.

File

plib_eth.h

C

```
void PLIB_ETH_RxDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the Ethernet receive logic.

Remarks

Disabling Ethernet receive is not recommended for making changes to any receive-related registers. After the receiver has been enabled, the

Ethernet module must be reinitialized to implement changes.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxDisable(ETH_MODULE_ID index)

PLIB_ETH_RxEnable Function

Enables the Ethernet receive logic.

File

plib_eth.h

C

```
void PLIB_ETH_RxEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the Ethernet receive logic. Packets are received and stored in the receive buffer as controlled by the filter configuration.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

All receive registers must be configured before calling this function.

Example

```
PLIB_ETH_RxEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxEnable(ETH_MODULE_ID index)

PLIB_ETH_RxFuncResetDisable Function

Disables the EMAC reset receive function logic.

File

plib_eth.h

C

```
void PLIB_ETH_RxFuncResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC reset receive function logic. The reset receive function logic is released from reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxFuncResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxFuncResetDisable(ETH_MODULE_ID index)

PLIB_ETH_RxFuncResetEnable Function

Enables the EMAC reset receive function logic.

File

plib_eth.h

C

```
void PLIB_ETH_RxFuncResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC reset receive function logic. The receive function logic is held in reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxFuncResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxFuncResetEnable(ETH_MODULE_ID index)

PLIB_ETH_RxFuncResetIsEnabled Function

Gets the EMAC reset receive function status.

File

plib_eth.h

C

```
bool PLIB_ETH_RxFuncResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - EMAC Receive function logic is held in reset
- false - EMAC Receive function logic is not in reset

Description

This function gets the EMAC reset receive function status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_RxFuncResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_RxFuncResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_RxIsEnabled Function

Gets the Ethernet module receive enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_RxIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Ethernet module receive is enabled
- false - Ethernet module receive is disabled

Description

This function returns the Ethernet module receive enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_ReceiveIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_RxIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_RxPacketDescAddrGet Function

Gets the address of the next receive descriptor.

File

plib_eth.h

C

```
uint8_t * PLIB_ETH_RxPacketDescAddrGet(ETH_MODULE_ID index);
```

Returns

- ReceivePacketDescriptorAddress - receive packet descriptor address

Description

This function gets the address of the next receive descriptor to be used.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
RxPacketDescAddr = PLIB_ETH_RxPacketDescAddrGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t *PLIB_ETH_RxPacketDescAddrGet(ETH_MODULE_ID index)

PLIB_ETH_RxPacketDescAddrSet Function

Sets the Ethernet module receive packet descriptor start address.

File

plib_eth.h

C

```
void PLIB_ETH_RxPacketDescAddrSet(ETH_MODULE_ID index, uint8_t * rxPacketDescStartAddr);
```

Returns

None.

Description

This function sets the Ethernet receive packet descriptor start address.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

No transmit, receive, or DMA operations should be in progress when this function is called. Call this function before enabling transmit and receive.

Example

```
PLIB_ETH_RxPacketDescAddrSet(MY_ETH_INSTANCE, rxPacketDescStartAddr)
```

Parameters

Parameters	Description
index	Identifier for the device instance
rxPacketDescStartAddr	This address must be 4-byte aligned. (The least significant 2 bits must be '00'.)

Function

void PLIB_ETH_RxPacketDescAddrSet(ETH_MODULE_ID index, uint8_t *rxPacketDescStartAddr)

PLIB_ETH_TestPauseDisable Function

Disables EMAC test pause.

File

plib_eth.h

C

```
void PLIB_ETH_TestPauseDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC Test Pause. The EMAC will resume normal operation.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TestPauseDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TestPauseDisable(ETH_MODULE_ID index)

PLIB_ETH_TestPauseEnable Function

Enables EMAC test pause.

File

plib_eth.h

C

```
void PLIB_ETH_TestPauseEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC test pause. The EMAC Control sub-layer will inhibit transmissions, just as if a Pause Receive Control frame with a non-zero pause time parameter was received.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TestPauseEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TestPauseEnable(ETH_MODULE_ID index)

PLIB_ETH_TestPauseIsEnabled Function

Gets the EMAC test pause enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_TestPauseIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC Control sub-layer will inhibit transmissions, just as if a Pause Receive Control frame with a non-zero pause time parameter was received
- false - Normal operation

Description

This function returns the EMAC test pause enable status.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_TestPauseIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_TestPauseIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_TxFuncResetDisable Function

Disables the EMAC Transmit function reset.

File

plib_eth.h

C

```
void PLIB_ETH_TxFuncResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC Transmit function reset. The EMAC transmit function is released from reset for normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TxFuncResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TxFuncResetDisable(ETH_MODULE_ID index)

PLIB_ETH_TxFuncResetEnable Function

Enables the EMAC transmit function reset.

File

plib_eth.h

C

```
void PLIB_ETH_TxFuncResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC transmit function reset. The transmit function is held in reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TxFuncResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TxFuncResetEnable(ETH_MODULE_ID index)

PLIB_ETH_TxFuncResetIsEnabled Function

Gets the EMAC Transmit function reset status.

File

plib_eth.h

C

```
bool PLIB_ETH_TxFuncResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - EMAC transmit function logic is held in reset
- false - EMAC transmit function logic is not in reset

Description

This function gets the EMAC Transmit function reset status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TxFuncResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_TxFuncResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_TxPacketDescAddrGet Function

Gets the address of the next descriptor to be transmitted.

File

plib_eth.h

C

```
uint8_t * PLIB_ETH_TxPacketDescAddrGet(ETH_MODULE_ID index);
```

Returns

Transmit Packet Descriptor Start Address.

Description

This function gets the address of the next transmit descriptor.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
txPacketDescAddr = PLIB_ETH_TxPacketDescAddrGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t *PLIB_ETH_TxPacketDescAddrGet(ETH_MODULE_ID index)

PLIB_ETH_TxPacketDescAddrSet Function

Sets the Ethernet module transmit packet descriptor start address.

File

plib_eth.h

C

```
void PLIB_ETH_TxPacketDescAddrSet(ETH_MODULE_ID index, uint8_t * txPacketDescStartAddr);
```

Returns

None.

Description

This function sets the Ethernet transmit packet descriptor start address.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

No transmit, receive, or DMA operations should be in progress when this function is called. Call this function before enabling transmit and receive.

Example

```
PLIB_ETH_TxPacketDescAddrSet(MY_ETH_INSTANCE, txPacketDescStartAddr)
```

Parameters

Parameters	Description
index	Identifier for the device instance
txPacketDescStartAddr	This address must be 4-byte aligned. (The least significant 2 bits must be '00'.)

Function

void PLIB_ETH_TxPacketDescAddrSet(ETH_MODULE_ID index, uint8_t *txPacketDescStartAddr)

PLIB_ETH_TxRTSDisable Function

Aborts an Ethernet module transmission.

File

plib_eth.h

C

```
void PLIB_ETH_TxRTSDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function aborts an Ethernet module transmission and disables the transmitter after the current packet has completed.

Remarks

When disabled by software, transmission stops after the current packet has been completed.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TxRTSDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TxRTSDisable(ETH_MODULE_ID index)

PLIB_ETH_TxRTSEnable Function

Enables the Ethernet transmit request to send.

File

plib_eth.h

C

```
void PLIB_ETH_TxRTSEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the Ethernet request to send. Transmit logic is activated and any packets defined in the Ethernet descriptor table are transmitted.

Remarks

This status is cleared by hardware when the transmission is complete.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

The TX descriptor list and TX DMA must be initialized using PLIB_ETH_TransmitPacketDescStartAddrSet.

Example

```
PLIB_ETH_TxRTSEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TxRTSEnable(ETH_MODULE_ID index)

PLIB_ETH_TxRTSIsEnabled Function

Gets the Ethernet module transmit request to send status.

File

plib_eth.h

C

```
bool PLIB_ETH_TxRTSIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Ethernet module transmit is active
- false - Ethernet module transmission has stopped or has completed

Description

This function returns the Ethernet module transmit request to send status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_TransmitRTSIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_TxRTSIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_CRCDisable Function

Disables EMAC CRC.

File

plib_eth.h

C

```
void PLIB_ETH_CRCDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC CRC.

Remarks

The frames presented to the EMAC have a valid CRC

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_CRCDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_CRCDisable(ETH_MODULE_ID index)

PLIB_ETH_CRCEnable Function

Enables EMAC CRC.

File

plib_eth.h

C

```
void PLIB_ETH_CRCEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC CRC. The EMAC will append CRC whether padding is required or not. This must be enabled if auto-padding is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_CRCEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_CRCEnable(ETH_MODULE_ID index)

PLIB_ETH_CRCIsEnabled Function

Gets the EMAC CRC enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_CRCIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC will append a CRC to every frame whether or not padding was required. Must be set if auto-padding is enabled.
- false - The frames presented to the EMAC have a valid CRC

Description

This function returns the EMAC CRC enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_CRCIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_CRCIsEnabled(ETH_MODULE_ID index)

c) Status Functions

PLIB_ETH_DataNotValid Function

Gets the MII management read data not valid status.

File

plib_eth.h

C

```
bool PLIB_ETH_DataNotValid(ETH_MODULE_ID index);
```

Returns

- true - The MII Management read cycle has not completed and the read data is not yet valid
- false - The MII Management read cycle is complete and the read data is valid

Description

This function gets the MII management read data not valid status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_DataNotValid(MY_ETH_INSTANCE);
```

Also see the example for function: PLIB_ETH_MIIReadStart().

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_DataNotValid(ETH_MODULE_ID index)

PLIB_ETH_EthernetIsBusy Function

Gets the status value of the Ethernet logic busy.

File

plib_eth.h

C

```
bool PLIB_ETH_EthernetIsBusy(ETH_MODULE_ID index);
```

Returns

- true - Ethernet logic has been turned ON or is completing a transaction
- false - Ethernet logic is idle

Description

This function sets the value of the Ethernet logic busy. A request indicates that the module has just been turned ON or is completing a transaction after being turned OFF.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_EthernetIsBusy(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_EthernetIsBusy(ETH_MODULE_ID index)

PLIB_ETH_LinkHasFailed Function

Gets the MII management link fail status.

File

plib_eth.h

C

```
bool PLIB_ETH_LinkHasFailed(ETH_MODULE_ID index);
```

Returns

- true - The MII Management module link fail has occurred
- false - The MII Management module link fail has not occurred

Description

This function returns the MII management link fail status. This value reflects the last read from the PHY status register.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_LinkHasFailed(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_LinkHasFailed(ETH_MODULE_ID index)

PLIB_ETH_MIIMIsBusy Function

Gets the MII management busy status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIMIsBusy(ETH_MODULE_ID index);
```

Returns

- true - The MII Management module is currently performing an MII Management read or write cycle
- false - The MII Management is free

Description

This function returns the MII management busy status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MIIMIsBusy(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIMIsBusy(ETH_MODULE_ID index)

PLIB_ETH_MIIMIsScanning Function

Gets the MII Management scanning status.

File

plib_eth.h

C

```
bool PLIB_ETH_MIIMIsScanning(ETH_MODULE_ID index);
```

Returns

- true - The MII Management module scan operation (continuous MII Management Read cycles) is in progress
- false - The MII Management scan operation is not in progress

Description

This function gets the MII Management scanning status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_MIIMIsScanning(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_MIIMIsScanning(ETH_MODULE_ID index)

PLIB_ETH_ReceiveIsBusy Function

Gets the Ethernet receive logic busy status.

File

plib_eth.h

C

```
bool PLIB_ETH_ReceiveIsBusy(ETH_MODULE_ID index);
```

Returns

- true - Receive logic is receiving data
- false - Receive logic is idle

Description

This function gets the Ethernet receive logic busy status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_ReceiveIsBusy(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_ReceiveIsBusy(ETH_MODULE_ID index)

PLIB_ETH_TransmitIsBusy Function

Gets the status value of the Ethernet transmit logic busy status

File

plib_eth.h

C

```
bool PLIB_ETH_TransmitIsBusy(ETH_MODULE_ID index);
```

Returns

- true - Transmit logic is sending data
- false - Transmit logic is idle

Description

This function gets the value of the Ethernet transmit logic busy status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_TransmitIsBusy(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_TransmitIsBusy(ETH_MODULE_ID index)

d) Filtering Functions

PLIB_ETH_HashTableGet Function

Gets the value of the Hash table.

File

plib_eth.h

C

```
uint32_t PLIB_ETH_HashTableGet(ETH_MODULE_ID index);
```

Returns

- hashTable - Hash table value (64-bits)

Description

This function gets the value of the Hash table.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_HashTableGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint64_t PLIB_ETH_HashTableGet(ETH_MODULE_ID index)

PLIB_ETH_HashTableSet Function

Sets the Ethernet module Hash table with the new value.

File

plib_eth.h

C

```
void PLIB_ETH_HashTableSet(ETH_MODULE_ID index, uint64_t hashTableValue);
```

Returns

None.

Description

This function sets the Hash table with the new value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Call PLIB_ETH_HashTableFilterDisable to disable the Hash table filter before changing the hash table, or set the Hash Table prior to calling PLIB_ETH_ReceiveEnable.

Example

```
PLIB_ETH_HashTableSet(MY_ETH_INSTANCE, hashTableValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
hashTableValue	hash table value (64-bits)

Function

void PLIB_ETH_HashTableSet(ETH_MODULE_ID index, uint64_t hashTableValue)

PLIB_ETH_PassAllDisable Function

Disables the EMAC PassAll.

File

plib_eth.h

C

```
void PLIB_ETH_PassAllDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC PassAll. Control frames are ignored.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_PassAllDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_PassAllDisable(ETH_MODULE_ID index)

PLIB_ETH_PassAllEnable Function

Enables the EMAC PassAll.

File

plib_eth.h

C

```
void PLIB_ETH_PassAllEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC PassAll, which enables both the normal and the control frames to be passed.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_PassAllEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_PassAllEnable(ETH_MODULE_ID index)

PLIB_ETH_PassAllIsEnabled Function

Gets the EMAC PassAll enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_PassAllIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC will accept all frames regardless of type (normal vs. Control)
- false - The received Control frames are ignored

Description

This function gets the EMAC PassAll enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_PassAllIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_PassAllIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_PatternMatchChecksumGet Function

Gets the value of the pattern match checksum register.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_PatternMatchChecksumGet(ETH_MODULE_ID index);
```

Returns

The pattern match checksum.

Description

This function gets the value of the patter match checksum register.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_PatternMatchChecksumGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_PatternMatchChecksumGet(ETH_MODULE_ID index)

PLIB_ETH_PatternMatchChecksumSet Function

Sets the Ethernet module pattern match checksum register with the new value.

File

plib_eth.h

C

```
void PLIB_ETH_PatternMatchChecksumSet(ETH_MODULE_ID index, uint16_t PatternMatchChecksumValue);
```

Returns

None.

Description

This function sets the pattern match checksum register with the new value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Call PLIB_ETH_PatternMatchModeSet(ETH_PATTERN_MATCH_DISABLED) to disable PatternMatch before changing the pattern match mask, or set the value before calling PLIB_ETH_ReceiveEnable.

Example

```
PLIB_ETH_PatternMatchChecksumSet(MY_ETH_INSTANCE, PatternMatchChecksumValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
PatternMatchChecksumValue	Pattern match checksum value

Function

void PLIB_ETH_PatternMatchChecksumSet(ETH_MODULE_ID index, uint16_t PatternMatchChecksumValue)

PLIB_ETH_PatternMatchGet Function

Gets the value of the selected pattern match mask register.

File

plib_eth.h

C

```
uint64_t PLIB_ETH_PatternMatchGet(ETH_MODULE_ID index);
```

Returns

- patternMatchMaskValue - Pattern Match Mask Values

Description

This function gets the selected value of the patter match mask register.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_PatternMatchGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint64_t PLIB_ETH_PatternMatchGet(ETH_MODULE_ID index)

PLIB_ETH_PatternMatchModeGet Function

Gets the value of the selected pattern match mask register.

File

plib_eth.h

C

```
ETH_PATTERN_MATCH_MODE PLIB_ETH_PatternMatchModeGet(ETH_MODULE_ID index);
```

Returns

- modeSel - The method of pattern matching from ETH_PATTERN_MATCH_DISABLED

Description

This function gets the selected value of the patter match mask register.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_PatternMatchModeGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

ETH_PATTERN_MATCH_MODE PLIB_ETH_PatternMatchModeGet(ETH_MODULE_ID index)

PLIB_ETH_PatternMatchModeSet Function

Sets the Ethernet module pattern match mode.

File

plib_eth.h

C

```
void PLIB_ETH_PatternMatchModeSet(ETH_MODULE_ID index, ETH_PATTERN_MATCH_MODE modeSel);
```

Returns

None.

Description

This function sets the pattern match mode.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Set the value before calling PLIB_ETH_ReceiveEnable().

Example

```
PLIB_ETH_PatternMatchModeSet(MY_ETH_INSTANCE, ETH_PATTERN_MATCH_DISABLED);
```

Parameters

Parameters	Description
index	Identifier for the device instance
modeSel	The method of pattern matching from ETH_PATTERN_MATCH_DISABLED

Function

void PLIB_ETH_PatternMatchModeSet(ETH_MODULE_ID index, ETH_PATTERN_MATCH_MODE modeSel)

PLIB_ETH_PatternMatchOffsetGet Function

Gets the value of the patter match offset register.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_PatternMatchOffsetGet(ETH_MODULE_ID index);
```

Returns

- PatternMatchOffsetValue - Pattern match offset value

Description

This function gets the value of the patter match offset register.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
value = PLIB_ETH_PatternMatchOffsetGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_PatternMatchOffsetGet(ETH_MODULE_ID index)

PLIB_ETH_PatternMatchOffsetSet Function

Sets the Ethernet module patter match offset register with the new value.

File

plib_eth.h

C

```
void PLIB_ETH_PatternMatchOffsetSet(ETH_MODULE_ID index, uint16_t PatternMatchOffsetValue);
```

Returns

None.

Description

This function sets the pattern match offset register with the new value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Call PLIB_ETH_PatternMatchModeSet(ETH_PATTERN_MATCH_DISABLED) to disable PatternMatch before changing the pattern match mask, or set the value before calling PLIB_ETH_ReceiveEnable.

Example

```
PLIB_ETH_PatternMatchOffsetSet(MY_ETH_INSTANCE, PatternMatchOffsetValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance

PatternMatchOffsetValue	Pattern match offset value

Function

void PLIB_ETH_PatternMatchOffsetSet(ETH_MODULE_ID index, uint16_t PatternMatchOffsetValue)

PLIB_ETH_PatternMatchSet Function

Sets the Ethernet module pattern match mask register with the new value.

File

plib_eth.h

C

```
void PLIB_ETH_PatternMatchSet(ETH_MODULE_ID index, uint64_t patternMatchMaskValue);
```

Returns

None.

Description

This function sets the pattern match mask register with the new value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

Call PLIB_ETH_PatternMatchModeSet(index,ETH_PATTERN_MATCH_DISABLED) to disable PatternMatch before changing the pattern match mask, or set the value before calling PLIB_ETH_ReceiveEnable.

Example

```
PLIB_ETH_PatternMatchSet(MY_ETH_INSTANCE, patternMatchMaskValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
patternMatchMaskValue	Pattern Match Mask Values (64-bits)

Function

void PLIB_ETH_PatternMatchSet(ETH_MODULE_ID index, uint64_t patternMatchMaskValue)

PLIB_ETH_PurePreambleDisable Function

Disables EMAC pure preamble enforcement.

File

plib_eth.h

C

```
void PLIB_ETH_PurePreambleDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC pure preamble enforcement. The EMAC does not perform any preamble checking.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_PurePreambleDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_PurePreambleDisable(ETH_MODULE_ID index)

PLIB_ETH_PurePreambleEnable Function

Enables EMAC pure preamble enforcement.

File

plib_eth.h

C

```
void PLIB_ETH_PurePreambleEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC pure preamble enforcement. The EMAC will verify the contents of the preamble and discard packets with errors in the preamble.

Remarks

The EMAC will verify the content of the preamble to ensure it contains 0x55 and is error-free. A packet with errors in its preamble is discarded.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_PurePreambleEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_PurePreambleEnable(ETH_MODULE_ID index)

PLIB_ETH_PurePreambleIsEnabled Function

Gets EMAC pure preamble enforcement enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_PurePreambleIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC will verify the content of the preamble to ensure it contains 0x55 and is error-free. A packet with errors in its preamble is discarded.
- false - The EMAC does not perform any preamble checking

Description

This function gets the EMAC pure preamble enforcement enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_PurePreambleIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_PurePreambleIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_ReceiveFilterDisable Function

Disables the specified receive filter.

File

plib_eth.h

C

```
void PLIB_ETH_ReceiveFilterDisable(ETH_MODULE_ID index, ETH_RECEIVE_FILTER filter);
```

Returns

None.

Description

This function disables the specified receive filter.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ReceiveFilterDisable(index,
                              ETH_CRC_OK_FILTER      |
                              ETH_RUNT_ENABLE_FILTER |
                              ETH_UNICAST_FILTER     );
```

Parameters

Parameters	Description
index	Identifier for the device instance
filter	The selection of receive filters to be disabled from the enumerated selection ETH_RECEIVE_FILTER

Function

void PLIB_ETH_ReceiveFilterDisable(ETH_MODULE_ID index, ETH_RECEIVE_FILTER filter)

PLIB_ETH_ReceiveFilterEnable Function

Enables the specified receive filter.

File

plib_eth.h

C

```
void PLIB_ETH_ReceiveFilterEnable(ETH_MODULE_ID index, ETH_RECEIVE_FILTER filter);
```

Returns

None.

Description

This function enables the specified receive filter.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ReceiveFilterEnable(index,
                             ETH_CRC_OK_FILTER      |
                             ETH_RUNT_ENABLE_FILTER |
                             ETH_UNICAST_FILTER     );
```

Parameters

Parameters	Description
index	Identifier for the device instance
filter	The selection of receive filters to be enabled from the enumerated selection ETH_RECEIVE_FILTER

Function

void PLIB_ETH_ReceiveFilterEnable(ETH_MODULE_ID index, ETH_RECEIVE_FILTER filter)

PLIB_ETH_ReceiveFilterIsEnable Function

Disables the specified receive filter.

File

plib_eth.h

C

```
bool PLIB_ETH_ReceiveFilterIsEnable(ETH_MODULE_ID index, ETH_RECEIVE_FILTER filter);
```

Returns

- true - If at least one of the specified filters is enabled
- false - If no specified filter is enabled

Description

This function disables the specified receive filter.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if (PLIB_ETH_ReceiveFilterIsEnable(index, ETH_UNICAST_FILTER))
{
    PLIB_ETH_ReceiveFilterDisable(MY_MODULE_ID, ETH_UNICAST_FILTER);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance
filter	The selection of receive filters to be disabled from the enumerated selection ETH_RECEIVE_FILTER

Function

bool PLIB_ETH_ReceiveFilterIsEnable(ETH_MODULE_ID index, ETH_RECEIVE_FILTER filter)

PLIB_ETH_StationAddressGet Function

Gets the selected EMAC station address.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_StationAddressGet(ETH_MODULE_ID index, uint8_t which);
```

Returns

- stationAddress1 - Station address

Description

This function gets the selected EMAC station address.

Remarks

On a reset, this register is loaded with the factory preprogrammed station address.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stationAddr1 = PLIB_ETH_StationAddress1Get(MY_ETH_INSTANCE, which);
```

Parameters

Parameters	Description
index	Identifier for the device instance
which	Select station address to change. Valid values are 1,2,3,4,5,6.

Function

uint8_t PLIB_ETH_StationAddressGet(ETH_MODULE_ID index, uint8_t which)

PLIB_ETH_StationAddressSet Function

Sets the selected EMAC Station Address.

File

plib_eth.h

C

```
void PLIB_ETH_StationAddressSet(ETH_MODULE_ID index, uint8_t which, uint8_t stationAddress);
```

Returns

None.

Description

This function sets the selected EMAC Station Address.

Remarks

On a reset, this register is loaded with the factory preprogrammed station address.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_StationAddressSet(MY_ETH_INSTANCE, which, stationAddress);
```

Parameters

Parameters	Description
index	Identifier for the device instance
which	Select station address to change. Valid values are 1,2,3,4,5,6.
stationAddress	Station Address.

Function

void PLIB_ETH_StationAddressSet(ETH_MODULE_ID index, uint8_t which, uint8_t stationAddress)

e) Flow Control Functions

PLIB_ETH_AutoFlowControlDisable Function

Disables the Ethernet module Automatic Flow Control logic.

File

plib_eth.h

C

```
void PLIB_ETH_AutoFlowControlDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the Ethernet Automatic Flow Control logic.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_AutoFlowControlDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_AutoFlowControlDisable(ETH_MODULE_ID index)

PLIB_ETH_AutoFlowControlEnable Function

Enables the Ethernet Automatic Flow Control logic.

File

plib_eth.h

C

```
void PLIB_ETH_AutoFlowControlEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables Ethernet Automatic Flow Control logic.

Remarks

The full and empty watermarks are used to automatically enable and disable flow control, respectively. When the number of received buffers rises to the full watermark, flow control is automatically enabled. When the receive buffer count falls to the empty watermark, flow control is automatically disabled.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_AutoFlowControlEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_AutoFlowControlEnable(ETH_MODULE_ID index)

PLIB_ETH_AutoFlowControlIsEnabled Function

Gets the Ethernet module Automatic Flow Control status.

File

plib_eth.h

C

```
bool PLIB_ETH_AutoFlowControlIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Ethernet module Automatic Flow Control is enabled
- false - Ethernet module Automatic Flow Control is disabled

Description

This function gets the Ethernet Automatic Flow Control enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_AutoFlowControlIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_AutoFlowControlIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_BackPresNoBackoffDisable Function

Disables EMAC backpressure/no back-off.

File

plib_eth.h

C

```
void PLIB_ETH_BackPresNoBackoffDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC backpressure/no back-off. The EMAC will back-off when there is backpressure and collisions occur.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_BackPresNoBackoffDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_BackPresNoBackoffDisable(ETH_MODULE_ID index)

PLIB_ETH_BackPresNoBackoffEnable Function

Enables EMAC back pressure/no back-off.

File

plib_eth.h

C

```
void PLIB_ETH_BackPresNoBackoffEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC backpressure/no back-off. The EMAC will not back-off when there is backpressure and collisions occur.

Remarks

The EMAC after incidentally causing a collision during backpressure will immediately retransmit without back-off reducing the chance of further collisions and ensuring transmit packets get sent.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_BackPresNoBackoffEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_BackPresNoBackoffEnable(ETH_MODULE_ID index)

PLIB_ETH_BackPresNoBackoffIsEnabled Function

Gets the EMAC backpressure/no back-off enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_BackPresNoBackoffIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC after incidentally causing a collision during backpressure will immediately retransmit without back-off reducing the chance of further collisions and ensuring transmit packets get sent
- false - The EMAC will not remove the back-off

Description

This function gets the EMAC backpressure/no back-off enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_BackPresNoBackoffIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_BackPresNoBackoffIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_ManualFlowControlDisable Function

Disable Ethernet module Manual Flow Control logic.

File

plib_eth.h

C

```
void PLIB_ETH_ManualFlowControlDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the Ethernet Manual Flow Control logic and automatically sends a Pause frame with a 0x0000 Pause Timer value. This function affects both transmit and receive operations.

Remarks

Disabling Manual Flow Control will automatically send a Pause frame with a 0x0000 Pause Timer value to disable flow control.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ManualFlowControlDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ManualFlowControlDisable(ETH_MODULE_ID index)

PLIB_ETH_ManualFlowControlEnable Function

Enables the Ethernet Manual Flow Control logic.

File

plib_eth.h

C

```
void PLIB_ETH_ManualFlowControlEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the Ethernet Manual Flow Control logic. Enabling Manual Flow Control will send a Pause frame using the Pause Timer value. While enabled, a Pause frame is repeated every 128 x (Pause timer value)/2 transmit clock cycles.

Affects both transmit and receive operations.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ManualFlowControlEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ManualFlowControlEnable(ETH_MODULE_ID index)

PLIB_ETH_ManualFlowControlIsEnabled Function

Gets the Ethernet module Manual Flow Control enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_ManualFlowControlIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Ethernet module Manual Flow Control is enabled
- false - Ethernet module Manual Flow Control is disabled

Description

This function returns the Ethernet module Manual Flow Control enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_ManualFlowControlIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_ManualFlowControlIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_NoBackoffDisable Function

Disables EMAC no back-offs.

File

plib_eth.h

C

```
void PLIB_ETH_NoBackoffDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC no back-off. The EMAC will back-off when a collision occurs.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_NoBackoffDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_NoBackoffDisable(ETH_MODULE_ID index)

PLIB_ETH_NoBackoffEnable Function

Enables EMAC no back-off.

File

plib_eth.h

C

```
void PLIB_ETH_NoBackoffEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC no back-off. The EMAC will not back-off when a collision occurs.

Remarks

Following a collision, the EMAC will immediately retransmit rather than using the Binary Exponential Back-off algorithm as specified in the Standard.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_NoBackoffEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_NoBackoffEnable(ETH_MODULE_ID index)

PLIB_ETH_NoBackoffIsEnabled Function

Gets the EMAC no back-off enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_NoBackoffIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Following a collision, the EMAC will immediately retransmit
- false - Following a collision, the EMAC will use the Binary Exponential Back-off algorithm

Description

This function gets the EMAC no back-off enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_NoBackoffIsEnabled(MY_ETsH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_NoBackoffIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_RxEmptyWmarkGet Function

Gets the Ethernet module receive empty watermark.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_RxEmptyWmarkGet(ETH_MODULE_ID index);
```

Returns

- receiveEmptyWmark - Empty watermark value

Description

This function gets the Ethernet receive empty watermark.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
watermarkValue = PLIB_ETH_RxEmptyWmarkGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_RxEmptyWmarkGet(ETH_MODULE_ID index)

PLIB_ETH_RxEmptyWmarkSet Function

Sets the Ethernet module receive empty water mark.

File

plib_eth.h

C

```
void PLIB_ETH_RxEmptyWmarkSet(ETH_MODULE_ID index, uint8_t watermarkValue);
```

Returns

None.

Description

This function sets the Ethernet receive empty water mark with a new value. The software controlled Receive Buffer Empty Wmark is compared against the receive buffer count to determine the empty watermark condition for the empty watermark interrupt and for disabling flow control if Auto Flow Control is enabled.

The Full Wmark value should always be greater than the Empty Wmark value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxEmptyWmarkSet(MY_ETH_INSTANCE, watermarkValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
watermarkValue	Empty watermark value

Function

void PLIB_ETH_RxEmptyWmarkSet(ETH_MODULE_ID index, uint8_t watermarkValue)

PLIB_ETH_RxFullWmarkGet Function

Gets the Ethernet module to receive a full watermark.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_RxFullWmarkGet(ETH_MODULE_ID index);
```

Returns

- receiveFullWmark - Full watermark value

Description

This function gets the Ethernet to receive a full watermark.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
watermarkValue = PLIB_ETH_RxFullWmarkGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_RxFullWmarkGet(ETH_MODULE_ID index)

PLIB_ETH_RxFullWmarkSet Function

Sets the Ethernet module to receive a full watermark.

File

plib_eth.h

C

```
void PLIB_ETH_RxFullWmarkSet(ETH_MODULE_ID index, uint8_t watermarkValue);
```

Returns

None.

Description

This function sets the Ethernet to receive a full watermark with a new value.

The software controlled RX Buffer Full Watermark (Wmark) Pointer is compared against the receive buffer count to determine the full watermark condition for the full watermark interrupt and for enabling flow control if Auto Flow Control is enabled.

The Full Wmark value should always be greater than the Empty Wmark value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxFullWmarkSet(MY_ETH_INSTANCE, watermarkValue);
```

Parameters

Parameters	Description
index	Identifier for the device instance
watermarkValue	Full watermark value

Function

void PLIB_ETH_RxFullWmarkSet(ETH_MODULE_ID index, uint8_t watermarkValue)

PLIB_ETH_RxPauseDisable Function

Disables the EMAC receive flow control.

File

plib_eth.h

C

```
void PLIB_ETH_RxPauseDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC receive flow control. The received Pause Flow control frames are ignored.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxPauseDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxPauseDisable(ETH_MODULE_ID index)

PLIB_ETH_RxPauseEnable Function

Enables the EMAC receive flow control.

File

plib_eth.h

C

```
void PLIB_ETH_RxPauseEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC receive flow control. The EMAC will act upon the received Pause frames.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxPauseEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxPauseEnable(ETH_MODULE_ID index)

PLIB_ETH_RxPauseIsEnabled Function

Gets the EMAC receive flow pause enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_RxPauseIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC acts upon received Pause Flow Control frames
- false - Received Pause Flow Control frames are ignored

Description

This function gets the EMAC receive flow pause enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
status = PLIB_ETH_RxPauseIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_RxPauseIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_ShortcutQuantaDisable Function

Disables EMAC shortcut pause quanta.

File

plib_eth.h

C

```
void PLIB_ETH_ShortcutQuantaDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC shortcut pause quanta. The EMAC will resume normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ShortcutQuantaDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ShortcutQuantaDisable(ETH_MODULE_ID index)

PLIB_ETH_ShortcutQuantaEnable Function

Enables EMAC shortcut pause quanta.

File

plib_eth.h

C

```
void PLIB_ETH_ShortcutQuantaEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC shortcut pause quanta. The EMAC reduces the effective pause quanta from 64 byte-times to 1 byte-time.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_ShortcutQuantaEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_ShortcutQuantaEnable(ETH_MODULE_ID index)

PLIB_ETH_ShortcutQuantaIsEnabled Function

Gets EMAC shortcut pause quanta enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_ShortcutQuantaIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC reduces the effective Pause Quanta from 64 byte-times to 1 byte-time
- false - Normal operation

Description

This function returns EMAC shortcut pause quanta enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_ShortcutQuantaIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_ShortcutQuantaIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_SimResetDisable Function

Disables the EMAC simulation reset.

File

plib_eth.h

C

```
void PLIB_ETH_SimResetDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the EMAC simulation reset.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_SimResetDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_SimResetDisable(ETH_MODULE_ID index)

PLIB_ETH_SimResetEnable Function

Enables the EMAC simulation reset.

File

plib_eth.h

C

```
void PLIB_ETH_SimResetEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the EMAC simulation reset and resets the random number generator within the transmit (TX) function.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_SimResetEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_SimResetEnable(ETH_MODULE_ID index)

PLIB_ETH_SimResetIsEnabled Function

Gets the EMAC simulation reset status.

File

plib_eth.h

C

```
bool PLIB_ETH_SimResetIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Simulation Reset is enabled
- false - Simulation Reset is disabled

Description

This function returns the EMAC simulation reset status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
stat = PLIB_ETH_SimResetIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_SimResetIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_TestBackPressDisable Function

Disables EMAC Test backpressure.

File

plib_eth.h

C

```
void PLIB_ETH_TestBackPressDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables EMAC Test backpressure. The EMAC will resume normal operation.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TestBackPressDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TestBackPressDisable(ETH_MODULE_ID index)

PLIB_ETH_TestBackPressEnable Function

Enables EMAC Test backpressure.

File

plib_eth.h

C

```
void PLIB_ETH_TestBackPressEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables EMAC Test backpressure. The EMAC will assert backpressure on the link. Backpressure causes preamble to be transmitted, raising carrier sense. A transmit packet from the system will be sent during backpressure.

Remarks

This functionality is intended for testing only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TestBackPressEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TestBackPressEnable(ETH_MODULE_ID index)

PLIB_ETH_TestBackPressIsEnabled Function

Gets the EMAC test backpressure enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_TestBackPressIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - The EMAC will assert backpressure on the link. Backpressure causes the preamble to be transmitted, raising the carrier sense. A transmit packet from the system will be sent during backpressure.
- false - Normal operation

Description

This function gets the EMAC test backpressure enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TestBackPressIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_TestBackPressIsEnabled(ETH_MODULE_ID index)

PLIB_ETH_TxPauseDisable Function

Disables the transmission of Pause frames.

File

plib_eth.h

C

```
void PLIB_ETH_TxPauseDisable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function disables the transmit Pause frames. The Pause frames are blocked from being transmitted.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TxPauseDisable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TxPauseDisable(ETH_MODULE_ID index)

PLIB_ETH_TxPauseEnable Function

Enables the transmission Pause frames.

File

plib_eth.h

C

```
void PLIB_ETH_TxPauseEnable(ETH_MODULE_ID index);
```

Returns

None.

Description

This function enables the transmission Pause frames. The Pause frames are allowed for transmission.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_TxPauseEnable(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_TxPauseEnable(ETH_MODULE_ID index)

PLIB_ETH_TxPauseIsEnabled Function

Gets the Ethernet module enable status.

File

plib_eth.h

C

```
bool PLIB_ETH_TxPauseIsEnabled(ETH_MODULE_ID index);
```

Returns

- true - Pause Flow Control frames are allowed to be transmitted
- false - Pause Flow Control frames are blocked

Description

This function gets the Ethernet module enable status.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
status = PLIB_ETH_TxPauseIsEnabled(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_ETH_TxPauseIsEnabled(ETH_MODULE_ID index)

f) Interrupt Functions

PLIB_ETH_InterruptClear Function

Clears the Ethernet module interrupt source status as a group, using a mask.

File

plib_eth.h

C

```
void PLIB_ETH_InterruptClear(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);
```

Returns

None.

Description

This function clears the Ethernet module interrupt source status using a mask. Logically 'OR' the masks together. Any masks not ORed with the others will be disabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// To clear several interrupts with one command, execute the following:
PLIB_ETH_InterruptClear(MY_ETH_INSTANCE,
                                    ETH_EMPTY_WATERMARK_INTERRUPT      |
                                    ETH_FULL_WATERMARK_INTERRUPT       |
                                    ETH_RX_FIFO_OVERFLOW_ERROR_INTERRUPT );
```

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Function

void PLIB_ETH_InterruptClear(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask)

PLIB_ETH_InterruptSet Function

Sets the Ethernet module interrupt source status as a group, using a mask.

File

plib_eth.h

C

```
void PLIB_ETH_InterruptSet(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);
```

Returns

None.

Description

This function sets the Ethernet module interrupt source status using a mask. Logically 'OR' the masks together. Any masks not OR'd with the others will be disabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// To set several interrupt flags with one command, execute the following:
PLIB_ETH_InterruptSet(MY_ETH_INSTANCE,
                                    ETH_EMPTY_WATERMARK_INTERRUPT      |
                                    ETH_FULL_WATERMARK_INTERRUPT       |
                                    ETH_RX_FIFO_OVERFLOW_ERROR_INTERRUPT );
```

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Function

void PLIB_ETH_InterruptSet(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask)

PLIB_ETH_InterruptsGet Function

Gets the Ethernet module Interrupt Flag register as a group.

File

plib_eth.h

C

```
ETH_INTERRUPT_SOURCES PLIB_ETH_InterruptsGet(ETH_MODULE_ID index);
```

Returns

Interrupt bit map, as defined in ETH_INTERRUPT_SOURCES, showing which interrupts have fired.

Description

Gets the Ethernet module Interrupt Flag register as a group.

Remarks

none.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Identifier for the device instance

Function

ETH_INTERRUPT_SOURCES PLIB_ETH_InterruptsGet(ETH_MODULE_ID index)

PLIB_ETH_InterruptSourceDisable Function

Clears the Ethernet module Interrupt Enable register as a group, using a mask.

File

plib_eth.h

C

```
void PLIB_ETH_InterruptSourceDisable(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);
```

Returns

None.

Description

This function disables the Ethernet module interrupts by using a mask to select one or multiple interrupts to disable. Logically 'OR' elements of ETH_INTERRUPT_SOURCES together.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// To disable some interrupts with one command, execute the following:
PLIB_ETH_InterruptSourceDisble(MY_ETH_INSTANCE,
                                ETH_TRANSMIT_DONE_INTERRUPT                |
                                ETH_TRANSMIT_ABORT_INTERRUPT               |
                                ETH_RECEIVE_BUFFER_NOT_AVAILABLE_INTERRUPT |
                                ETH_RECEIVE_FIFO_OVERFLOW_ERROR_INTERRUPT );

// Or to disable one interrupt with one command, execute the following:
PLIB_ETH_InterruptSourceDisble(MY_ETH_INSTANCE, ETH_TRANSMIT_DONE_INTERRUPT);

// Or to disable all interrupts with one command, execute the following:
PLIB_ETH_InterruptSourceDisble(MY_ETH_INSTANCE, ETH_ALL_ETH_INTERRUPTS);
```

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Function

void PLIB_ETH_InterruptSourceDisable(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask)

PLIB_ETH_InterruptSourceEnable Function

Sets the Ethernet module Interrupt Enable register in a group, using a mask.

File

plib_eth.h

C

```
void PLIB_ETH_InterruptSourceEnable(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);
```

Returns

None.

Description

This function enables the Ethernet module interrupts by using a mask to enable one or multiple interrupt enables. Logically 'OR' elements of ETH_INTERRUPT_SOURCES together.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// To enable some interrupts with one command, execute the following:
PLIB_ETH_InterruptSourceEnable(MY_ETH_INSTANCE,
                                    ETH_TRANSMIT_DONE_INTERRUPT                |
                                    ETH_TRANSMIT_ABORT_INTERRUPT               |
                                    ETH_RECEIVE_BUFFER_NOT_AVAILABLE_INTERRUPT |
                                    ETH_RECEIVE_FIFO_OVERFLOW_ERROR_INTERRUPT);

// Or to enable one interrupt with one command, execute the following:
PLIB_ETH_InterruptSourceEnable(MY_ETH_INSTANCE, ETH_TRANSMIT_DONE_INTERRUPT);

// Or to enable all interrupts with one command, execute the following:
PLIB_ETH_InterruptSourceEnable(MY_ETH_INSTANCE, ETH_ALL_ETH_INTERRUPTS);
```

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Function

void PLIB_ETH_InterruptSourceEnable(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask)

PLIB_ETH_InterruptSourceIsEnabled Function

Gets the Ethernet module Interrupt Enable register singularly or as a group.

File

plib_eth.h

C

```
bool PLIB_ETH_InterruptSourceIsEnabled(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);
```

Returns

- true - If any of the specified sources are enabled
- false - If none of the specified sources are enabled

Description

This function returns a true if any of the specified interrupt sources are enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_ETH_InterruptSourceIsEnabled(index, ETH_TRANSMIT_DONE_INTERRUPT))
{
    //If the interrupt is enabled, disable it...
    PLIB_ETH_InterruptSourceDisable(index, ETH_TRANSMIT_DONE_INTERRUPT);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance
mask	The interrupt mask(s) to be checked

Function

bool PLIB_ETH_InterruptSourceIsEnabled(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask)

PLIB_ETH_InterruptSourcesGet Function

Returns the entire interrupt enable register.

File

plib_eth.h

C

```
ETH_INTERRUPT_SOURCES PLIB_ETH_InterruptSourcesGet(ETH_MODULE_ID index);
```

Returns

Bit map of interrupt sources, all ORed together, as defined by ETH_INTERRUPT_SOURCES.

Description

This function returns the entire interrupt enable register.

Remarks

None.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Identifier for the device instance

Function

ETH_INTERRUPT_SOURCES PLIB_ETH_InterruptSourcesGet(ETH_MODULE_ID index)

PLIB_ETH_InterruptStatusGet Function

Gets the Ethernet module Interrupt Flag register as a group, using a mask.

File

plib_eth.h

C

```
bool PLIB_ETH_InterruptStatusGet(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);
```

Returns

- true - If any of the specified source statuses are enabled

- false - If none of the specified source statuses are enabled

Description

This function gets the Ethernet module Interrupt Flag register using a mask.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if (PLIB_ETH_InterruptStatusGet(index, ETH_TRANSMIT_DONE_INTERRUPT))
{
    PLIB_ETH_InterruptClear(index, ETH_TRANSMIT_DONE_INTERRUPT);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance
intmask	Members of ETH_INTERRUPT_SOURCES logically ORed together

Function

bool PLIB_ETH_InterruptStatusGet(ETH_MODULE_ID index, ETH_INTERRUPT_SOURCES intmask);

g) Statistics Functions

PLIB_ETH_AlignErrorCountClear Function

Sets the count of Ethernet alignment errors to zero.

File

plib_eth.h

C

```
void PLIB_ETH_AlignErrorCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the count of Ethernet alignment errors to zero.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_AlignErrorCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance
value	count of alignment errors

Function

void PLIB_ETH_AlignErrorCountClear(ETH_MODULE_ID index)

PLIB_ETH_AlignErrorCountGet Function

Gets the count of Ethernet alignment errors.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_AlignErrorCountGet(ETH_MODULE_ID index);
```

Returns

- value - Count of alignment errors

Description

This function gets the count of Ethernet alignment errors.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_AlignErrorCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_AlignErrorCountGet(ETH_MODULE_ID index)

PLIB_ETH_FCSErrorCountClear Function

Sets the value of the Ethernet frame check sequence error to zero.

File

plib_eth.h

C

```
void PLIB_ETH_FCSErrorCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the value of the Ethernet frame check sequence error to zero.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FCSErrorCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of FCS Errors

Function

void PLIB_ETH_FCSErrorCountClear(ETH_MODULE_ID index)

PLIB_ETH_FCSErrorCountGet Function

Gets the count of the frame check sequence error.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_FCSErrorCountGet(ETH_MODULE_ID index);
```

Returns

- value - Count of FCS Errors

Description

This function gets the count of the frame check sequence error.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_FCSErrorCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_FCSErrorCountGet(ETH_MODULE_ID index)

PLIB_ETH_FramesRxdOkCountClear Function

Sets the value of the Ethernet received frames 'OK' count to zero.

File

plib_eth.h

C

```
void PLIB_ETH_FramesRxdOkCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the value of the Ethernet frames 'OK' count to zero.

Remarks

Changing the value of this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FramesRxdOkCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of frames received correctly

Function

void PLIB_ETH_FramesRxdOkCountClear(ETH_MODULE_ID index)

PLIB_ETH_FramesRxdOkCountGet Function

Gets the count of the frames frames received successfully.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_FramesRxdOkCountGet(ETH_MODULE_ID index);
```

Returns

- value - Count of frames received correctly

Description

This function gets the count of the frames received successfully. Increment count for frames received successfully by the RX Filter. This count will not be incremented if there is a Frame Check Sequence (FCS) or Alignment error.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_FramesRxdOkCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_FramesRxdOkCountGet(ETH_MODULE_ID index)

PLIB_ETH_FramesTxdOkCountClear Function

Sets the count of Ethernet Control frames transmitted to zero.

File

plib_eth.h

C

```
void PLIB_ETH_FramesTxdOkCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the count of Ethernet Control frames transmitted to zero.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_FramesTxdOkCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of control frames transmitted correctly

Function

void PLIB_ETH_FramesTxdOkCountClear(ETH_MODULE_ID index)

PLIB_ETH_FramesTxdOkCountGet Function

Gets the count of Ethernet Control Frames transmitted successfully.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_FramesTxdOkCountGet(ETH_MODULE_ID index);
```

Returns

- count - count of control frames transmitted correctly

Description

This function gets the count of Ethernet Control Frames transmitted successfully.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_FramesTxdOkCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_FramesTxdOkCountGet(ETH_MODULE_ID index)

PLIB_ETH_MultipleCollisionCountClear Function

Sets the value of the Ethernet multiple collision frame count to zero.

File

plib_eth.h

C

```
void PLIB_ETH_MultipleCollisionCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the value of the Ethernet multiple collision frame count to zero.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_MultipleCollisionCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of multiple collision frames

Function

void PLIB_ETH_MultipleCollisionCountClear(ETH_MODULE_ID index)

PLIB_ETH_MultipleCollisionCountGet Function

Gets the count of the frames transmitted successfully after there was more than one collision.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_MultipleCollisionCountGet(ETH_MODULE_ID index);
```

Returns

- count - Count of multiple collision frames

Description

This function gets the count of the frames transmitted successfully after there was more than one collision.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_MultipleCollisionCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_MultipleCollisionCountGet(ETH_MODULE_ID index)

PLIB_ETH_RxOverflowCountClear Function

Sets the value of the Ethernet receive overflow count to zero.

File

plib_eth.h

C

```
void PLIB_ETH_RxOverflowCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the value of the Ethernet receive overflow count to zero. Increment counter for frames accepted by the RX filter and subsequently dropped due to internal receive error. This event also sets the receive overflow interrupt flag.

Remarks

Setting or clearing any bits in this register should be done for debug/test purposes only.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_RxOverflowCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

void PLIB_ETH_RxOverflowCountClear(ETH_MODULE_ID index)

PLIB_ETH_RxOverflowCountGet Function

Gets the count of the dropped Ethernet receive frames.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_RxOverflowCountGet(ETH_MODULE_ID index);
```

Returns

- count - uint16_t receiver overflow counts

Description

This function gets the count of the dropped Ethernet receive frames.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_RxOverflowCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_RxOverflowCountGet(ETH_MODULE_ID index)

PLIB_ETH_RxPacketCountGet Function

Gets the value of the receive packet buffer count.

File

plib_eth.h

C

```
uint8_t PLIB_ETH_RxPacketCountGet(ETH_MODULE_ID index);
```

Returns

- RxPacketCount - Number of received packets in memory

Description

This function gets the value of the receive packet buffer count. When a packet has been successfully received, this value is incremented by hardware. Software decrements the counter after a packet has been read. Call PLIB_ETH_ReceiveBufferCountDecrement(ETH_MODULE_ID index) to decrement.

Remarks

Receive Packet Buffer Counter cannot be decremented below 0x00 Cleared when the Ethernet module is reset.

The Receive Packet Buffer Count is not set to '0' when the Ethernet module is turned OFF. This allows software to continue to utilize and decrement the count.

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_RxPacketCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_ETH_RxPacketCountGet(ETH_MODULE_ID index)

PLIB_ETH_SingleCollisionCountClear Function

Sets the value of the Ethernet single collision frame count to zero.

File

plib_eth.h

C

```
void PLIB_ETH_SingleCollisionCountClear(ETH_MODULE_ID index);
```

Returns

None.

Description

This function sets the value of the Ethernet single collision frame count to zero.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_ETH_SingleCollisionCountClear(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance
value	Count of single collision frames

Function

void PLIB_ETH_SingleCollisionCountClear(ETH_MODULE_ID index)

PLIB_ETH_SingleCollisionCountGet Function

Gets the count of the frames transmitted successfully on a second attempt.

File

plib_eth.h

C

```
uint16_t PLIB_ETH_SingleCollisionCountGet(ETH_MODULE_ID index);
```

Returns

- count - Count of frames transmitted successfully on second attempt

Description

This function gets the count of the frames transmitted successfully on a second attempt.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
count = PLIB_ETH_SingleCollisionCountGet(MY_ETH_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint16_t PLIB_ETH_SingleCollisionCountGet(ETH_MODULE_ID index)

h) Feature Existence Functions

PLIB_ETH_ExistsAlignmentErrorCount Function

Identifies whether the AlignmentErrorCount feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsAlignmentErrorCount(ETH_MODULE_ID index);
```

Returns

- true - The AlignmentErrorCount feature is supported on the device
- false - The AlignmentErrorCount feature is not supported on the device

Description

This function identifies whether the AlignmentErrorCount feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_AlignErrorCountClear
- PLIB_ETH_AlignErrorCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsAlignmentErrorCount(ETH_MODULE_ID index)

PLIB_ETH_ExistsAutoFlowControl Function

Identifies whether the AutoFlowControl feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsAutoFlowControl(ETH_MODULE_ID index);
```

Returns

- true - The AutoFlowControl feature is supported on the device
- false - The AutoFlowControl feature is not supported on the device

Description

This function identifies whether the AutoFlowControl feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_AutoFlowControlEnable
- PLIB_ETH_AutoFlowControlDisable
- PLIB_ETH_AutoFlowControlIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsAutoFlowControl(ETH_MODULE_ID index)

PLIB_ETH_ExistsCollisionCounts Function

Identifies whether the CollisionCounts feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsCollisionCounts(ETH_MODULE_ID index);
```

Returns

- true - The CollisionCounts feature is supported on the device
- false - The CollisionCounts feature is not supported on the device

Description

This function identifies whether the CollisionCounts feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_SingleCollisionCountClear
- PLIB_ETH_SingleCollisionCountGet
- PLIB_ETH_MultipleCollisionCountClear
- PLIB_ETH_MultipleCollisionCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsCollisionCounts(ETH_MODULE_ID index)

PLIB_ETH_ExistsCollisionWindow Function

Identifies whether the CollisionWindow feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsCollisionWindow(ETH_MODULE_ID index);
```

Returns

- true - The CollisionWindow feature is supported on the device
- false - The CollisionWindow feature is not supported on the device

Description

This function identifies whether the CollisionWindow feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_CollisionWindowGet
- PLIB_ETH_CollisionWindowSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsCollisionWindow(ETH_MODULE_ID index)

PLIB_ETH_ExistsEnable Function

Identifies whether the Enable feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsEnable(ETH_MODULE_ID index);
```

Returns

- true - The Enable feature is supported on the device
- false - The Enable feature is not supported on the device

Description

This function identifies whether the Enable feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_Enable
- PLIB_ETH_Disable
- PLIB_ETH_IsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsEnable(ETH_MODULE_ID index)

PLIB_ETH_ExistsEthernetControllerStatus Function

Identifies whether the EthernetControllerStatus feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsEthernetControllerStatus(ETH_MODULE_ID index);
```

Returns

- true - The EthernetControllerStatus feature is supported on the device
- false - The EthernetControllerStatus feature is not supported on the device

Description

This function identifies whether the EthernetControllerStatus feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_RxPacketCountGet
- PLIB_ETH_EthernetIsBus

- PLIB_ETH_TransmitIsBusy
- PLIB_ETH_ReceiveIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsEthernetControllerStatus(ETH_MODULE_ID index)

PLIB_ETH_ExistsFCSErrorCount Function

Identifies whether the FCSErrorCount feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsFCSErrorCount(ETH_MODULE_ID index);
```

Returns

- true - The FCSErrorCount feature is supported on the device
- false - The FCSErrorCount feature is not supported on the device

Description

This function identifies whether the FCSErrorCount feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_FCSErrorCountClear
- PLIB_ETH_FCSErrorCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsFCSErrorCount(ETH_MODULE_ID index)

PLIB_ETH_ExistsFramesTransmittedOK Function

Identifies whether the FramesTransmittedOK feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsFramesTransmittedOK(ETH_MODULE_ID index);
```

Returns

- true - The FramesTransmittedOK feature is supported on the device

- false - The FramesTransmittedOK feature is not supported on the device

Description

This function identifies whether the FramesTransmittedOK feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_FramesTxdOkCountClear
- PLIB_ETH_FramesTxdOkCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsFramesTransmittedOK(ETH_MODULE_ID index)

PLIB_ETH_ExistsFramexReceivedOK Function

Identifies whether the FramexReceivedOK feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsFramexReceivedOK(ETH_MODULE_ID index);
```

Returns

- true - The FramexReceivedOK feature is supported on the device
- false - The FramexReceivedOK feature is not supported on the device

Description

This function identifies whether the FramexReceivedOK feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_FramesRxdOkCountClear
- PLIB_ETH_FramesRxdOkCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsFramexReceivedOK(ETH_MODULE_ID index)

PLIB_ETH_ExistsHashTable Function

Identifies whether the HashTable feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsHashTable(ETH_MODULE_ID index);
```

Returns

- true - The HashTable feature is supported on the device
- false - The HashTable feature is not supported on the device

Description

This function identifies whether the HashTable feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_HashTableSet
- PLIB_ETH_HashTableGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsHashTable(ETH_MODULE_ID index)

PLIB_ETH_ExistsInterPacketGaps Function

Identifies whether the InterPacketGaps feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsInterPacketGaps(ETH_MODULE_ID index);
```

Returns

- true - The InterPacketGaps feature is supported on the device
- false - The InterPacketGaps feature is not supported on the device

Description

This function identifies whether the InterPacketGaps feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_BackToBackIPGGet
- PLIB_ETH_BackToBackIPGSet
- PLIB_ETH_NonBackToBackIPG1Get
- PLIB_ETH_NonBackToBackIPG1Set
- PLIB_ETH_NonBackToBackIPG2Get
- PLIB_ETH_NonBackToBackIPG2Set

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsInterPacketGaps(ETH_MODULE_ID index)

PLIB_ETH_ExistsInterrupt Function

Identifies whether the Interrupt feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsInterrupt(ETH_MODULE_ID index);
```

Returns

- true - The Interrupt feature is supported on the device
- false - The Interrupt feature is not supported on the device

Description

This function identifies whether the Interrupt feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_InterruptSourceEnable
- PLIB_ETH_InterruptSourceDisable
- PLIB_ETH_InterruptSourceIsEnabled
- PLIB_ETH_InterruptSet
- PLIB_ETH_InterruptClear
- PLIB_ETH_InterruptStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsInterrupt(ETH_MODULE_ID index)

PLIB_ETH_ExistsMAC_Configuration Function

Identifies whether the MAC_Configuration feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMAC_Configuration(ETH_MODULE_ID index);
```

Returns

- true - The MAC_Configuration feature is supported on the device
- false - The MAC_Configuration feature is not supported on the device

Description

This function identifies whether the MAC_Configuration feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_LoopbackEnable
- PLIB_ETH_LoopbackDisable
- PLIB_ETH_LoopbackIsEnabled

- PLIB_ETH_TxPauseEnable
- PLIB_ETH_TxPauseDisable
- PLIB_ETH_TxPauseIsEnabled
- PLIB_ETH_RxPauseEnable
- PLIB_ETH_RxPauseDisable
- PLIB_ETH_RxPauseIsEnabled
- PLIB_ETH_PassAllEnable
- PLIB_ETH_PassAllDisable
- PLIB_ETH_PassAllIsEnabled
- PLIB_ETH_ReceiveEnable
- PLIB_ETH_ReceiveDisable
- PLIB_ETH_ReceiveIsEnabled
- PLIB_ETH_ExcessDeferEnable
- PLIB_ETH_ExcessDeferDisable
- PLIB_ETH_ExcessDeferIsEnabled
- PLIB_ETH_BackPresNoBackoffEnable
- PLIB_ETH_BackPresNoBackoffDisable
- PLIB_ETH_BackPresNoBackoffIsEnabled
- PLIB_ETH_NoBackoffEnable
- PLIB_ETH_NoBackoffDisable
- PLIB_ETH_NoBackoffIsEnabled
- PLIB_ETH_LongPreambleEnable
- PLIB_ETH_LongPreambleDisable
- PLIB_ETH_LongPreambleIsEnabled
- PLIB_ETH_PurePreambleEnable
- PLIB_ETH_PurePreambleDisable
- PLIB_ETH_PurePreambleIsEnabled
- PLIB_ETH_AutoDetectPadGet
- PLIB_ETH_AutoDetectPadSet
- PLIB_ETH_AutoDetectPadClear
- PLIB_ETH_CRCEnable
- PLIB_ETH_CRCDisable
- PLIB_ETH_CRCIsEnabled
- PLIB_ETH_DelayedCRCEnable
- PLIB_ETH_DelayedCRCDisable
- PLIB_ETH_DelayedCRCIsEnabled
- PLIB_ETH_HugeFrameEnable
- PLIB_ETH_HugeFrameDisable
- PLIB_ETH_HugeFrameIsEnabled
- PLIB_ETH_FrameLengthCheckEnable
- PLIB_ETH_FrameLengthCheckDisable
- PLIB_ETH_FrameLengthCheckIsEnabled
- PLIB_ETH_FullDuplexEnable
- PLIB_ETH_FullDuplexDisable
- PLIB_ETH_FullDuplexIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMAC_Configuration(ETH_MODULE_ID index)

PLIB_ETH_ExistsMAC_Resets Function

Identifies whether the MAC_Resets feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMAC_Resets(ETH_MODULE_ID index);
```

Returns

- true - The MAC_Resets feature is supported on the device
- false - The MAC_Resets feature is not supported on the device

Description

This function identifies whether the MAC_Resets feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_MIIResetEnable
- PLIB_ETH_MIIResetDisable
- PLIB_ETH_MIIResetIsEnabled
- PLIB_ETH_SimResetEnable
- PLIB_ETH_SimResetDisable
- PLIB_ETH_SimResetIsEnabled
- PLIB_ETH_MCSRxResetEnable
- PLIB_ETH_MCSRxResetDisable
- PLIB_ETH_MCSRxResetIsEnabled
- PLIB_ETH_RxFuncResetEnable
- PLIB_ETH_RxFuncResetDisable
- PLIB_ETH_RxFuncResetIsEnabled
- PLIB_ETH_MCSTxResetEnable
- PLIB_ETH_MCSTxResetDisable
- PLIB_ETH_MCSTxResetIsEnabled
- PLIB_ETH_TxFuncResetEnable
- PLIB_ETH_TxFuncResetDisable
- PLIB_ETH_TxFuncResetIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMAC_Resets(ETH_MODULE_ID index)

PLIB_ETH_ExistsMAC_Testing Function

Identifies whether the MAC_Testing feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMAC_Testing(ETH_MODULE_ID index);
```

Returns

- true - The MAC_Testing feature is supported on the device
- false - The MAC_Testing feature is not supported on the device

Description

This function identifies whether the MAC_Testing feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_TestBackPressEnable
- PLIB_ETH_TestBackPressDisable
- PLIB_ETH_TestBackPressIsEnabled
- PLIB_ETH_TestPauseEnable
- PLIB_ETH_TestPauseDisable
- PLIB_ETH_TestPauseIsEnabled
- PLIB_ETH_ShortcutQuantaEnable
- PLIB_ETH_ShortcutQuantaDisable
- PLIB_ETH_ShortcutQuantaIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMAC_Testing(ETH_MODULE_ID index)

PLIB_ETH_ExistsManualFlowControl Function

Identifies whether the ManualFlowControl feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsManualFlowControl(ETH_MODULE_ID index);
```

Returns

- true - The ManualFlowControl feature is supported on the device
- false - The ManualFlowControl feature is not supported on the device

Description

This function identifies whether the ManualFlowControl feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_ManualFlowControlEnable
- PLIB_ETH_ManualFlowControlDisable
- PLIB_ETH_ManualFlowControlIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsManualFlowControl(ETH_MODULE_ID index)

PLIB_ETH_ExistsMaxFrameLength Function

Identifies whether the MaxFrameLength feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMaxFrameLength(ETH_MODULE_ID index);
```

Returns

- true - The MaxFrameLength feature is supported on the device
- false - The MaxFrameLength feature is not supported on the device

Description

This function identifies whether the MaxFrameLength feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_MaxFrameLengthGet
- PLIB_ETH_MaxFrameLengthSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMaxFrameLength(ETH_MODULE_ID index)

PLIB_ETH_ExistsMIIM_Config Function

Identifies whether the MIIM_Config feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMIIM_Config(ETH_MODULE_ID index);
```

Returns

- true - The MIIM_Config feature is supported on the device
- false - The MIIM_Config feature is not supported on the device

Description

This function identifies whether the MIIM_Config feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_MIIMResetEnable
- PLIB_ETH_MIIMResetDisable
- PLIB_ETH_MIIMResetIsEnabled
- PLIB_ETH_MIIMClockGet
- PLIB_ETH_MIIMClockSet
- PLIB_ETH_MIIMNoPreEnable
- PLIB_ETH_MIIMNoPreDisable

- PLIB_ETH_MIIMNoPreIsEnabled
- PLIB_ETH_MIIMScanIncrEnable
- PLIB_ETH_MIIMScanIncrDisable
- PLIB_ETH_MIIMScanIncrIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMIIM_Config(ETH_MODULE_ID index)

PLIB_ETH_ExistsMIIM_Indicators Function

Identifies whether the MIIM_Indicators feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMIIM_Indicators(ETH_MODULE_ID index);
```

Returns

- true - The MIIM_Indicators feature is supported on the device
- false - The MIIM_Indicators feature is not supported on the device

Description

This function identifies whether the MIIM_Indicators feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_LinkHasFailed
- PLIB_ETH_DataNotValid
- PLIB_ETH_MIIMIsScanning
- PLIB_ETH_MIIMIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMIIM_Indicators(ETH_MODULE_ID index)

PLIB_ETH_ExistsMIIMAddresses Function

Identifies whether the MIIMAddresses feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMIIMAddresses(ETH_MODULE_ID index);
```

Returns

- true - The MIIMAddresses feature is supported on the device
- false - The MIIMAddresses feature is not supported on the device

Description

This function identifies whether the MIIMAddresses feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_PHYAddressGet
- PLIB_ETH_PHYAddressSet
- PLIB_ETH_RegisterAddressGet
- PLIB_ETH_RegisterAddressSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMIIMAddresses(ETH_MODULE_ID index)

PLIB_ETH_ExistsMIIMReadWrite Function

Identifies whether the MIIMReadWrite feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMIIMReadWrite(ETH_MODULE_ID index);
```

Returns

- true - The MIIMReadWrite feature is supported on the device
- false - The MIIMReadWrite feature is not supported on the device

Description

This function identifies whether the MIIMReadWrite feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_MIIMReadStart
- PLIB_ETH_MIIMWriteStart

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMIIMReadWrite(ETH_MODULE_ID index)

PLIB_ETH_ExistsMIIMScanMode Function

Identifies whether the MIIMScanMode feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMIIMScanMode(ETH_MODULE_ID index);
```

Returns

- true - The MIIMScanMode feature is supported on the device
- false - The MIIMScanMode feature is not supported on the device

Description

This function identifies whether the MIIMScanMode feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_MIIMScanModeEnable
- PLIB_ETH_MIIMScanModeDisable
- PLIB_ETH_MIIMScanModeIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMIIMScanMode(ETH_MODULE_ID index)

PLIB_ETH_ExistsMIIWriteReadData Function

Identifies whether the MIIWriteReadData feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsMIIWriteReadData(ETH_MODULE_ID index);
```

Returns

- true - The MIIWriteReadData feature is supported on the device
- false - The MIIWriteReadData feature is not supported on the device

Description

This function identifies whether the MIIWriteReadData feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_MIIMWriteDataSet
- PLIB_ETH_MIIMReadDataGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsMIIWriteReadData(ETH_MODULE_ID index)

PLIB_ETH_ExistsPatternMatch Function

Identifies whether the PatternMatch feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsPatternMatch(ETH_MODULE_ID index);
```

Returns

- true - The PatternMatch feature is supported on the device
- false - The PatternMatch feature is not supported on the device

Description

This function identifies whether the PatternMatch feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_PatternMatchSet
- PLIB_ETH_PatternMatchGet
- PLIB_ETH_PatternMatchChecksumSet
- PLIB_ETH_PatternMatchChecksumGet
- PLIB_ETH_PatternMatchOffsetSet
- PLIB_ETH_PatternMatchOffsetGet
- PLIB_ETH_PatternMatchModeSet
- PLIB_ETH_PatternMatchModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsPatternMatch(ETH_MODULE_ID index)

PLIB_ETH_ExistsPauseTimer Function

Identifies whether the PauseTimer feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsPauseTimer(ETH_MODULE_ID index);
```

Returns

- true - The PauseTimer feature is supported on the device
- false - The PauseTimer feature is not supported on the device

Description

This function identifies whether the PauseTimer feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_PauseTimerSet
- PLIB_ETH_PauseTimerGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsPauseTimer(ETH_MODULE_ID index)

PLIB_ETH_ExistsReceiveBufferSize Function

Identifies whether the ReceiveBufferSize feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsReceiveBufferSize(ETH_MODULE_ID index);
```

Returns

- true - The ReceiveBufferSize feature is supported on the device
- false - The ReceiveBufferSize feature is not supported on the device

Description

This function identifies whether the ReceiveBufferSize feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_ReceiveBufferSizeGet
- PLIB_ETH_ReceiveBufferSizeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsReceiveBufferSize(ETH_MODULE_ID index)

PLIB_ETH_ExistsReceiveFilters Function

Identifies whether the ReceiveFilters feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsReceiveFilters(ETH_MODULE_ID index);
```

Returns

- true - The ReceiveFilters feature is supported on the device
- false - The ReceiveFilters feature is not supported on the device

Description

This function identifies whether the ReceiveFilters feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_ReceiveFilterEnable
- PLIB_ETH_ReceiveFilterDisable
- PLIB_ETH_ReceiveFilterIsEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsReceiveFilters(ETH_MODULE_ID index)

PLIB_ETH_ExistsReceiveOverflowCount Function

Identifies whether the ReceiveOverflowCount feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsReceiveOverflowCount(ETH_MODULE_ID index);
```

Returns

- true - The ReceiveOverflowCount feature is supported on the device
- false - The ReceiveOverflowCount feature is not supported on the device

Description

This function identifies whether the ReceiveOverflowCount feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_RxOverflowCountClear
- PLIB_ETH_RxOverflowCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsReceiveOverflowCount(ETH_MODULE_ID index)

PLIB_ETH_ExistsReceiveWmarks Function

Identifies whether the ReceiveWmarks feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsReceiveWmarks(ETH_MODULE_ID index);
```

Returns

- true - The ReceiveWmarks feature is supported on the device
- false - The ReceiveWmarks feature is not supported on the device

Description

This function identifies whether the ReceiveWmarks feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_RxFullWmarkSet
- PLIB_ETH_RxFullWmarkGet
- PLIB_ETH_RxEmptyWmarkSet
- PLIB_ETH_RxEmptyWmarkGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsReceiveWmarks(ETH_MODULE_ID index)

PLIB_ETH_ExistsRetransmissionMaximum Function

Identifies whether the RetransmissionMaximum feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsRetransmissionMaximum(ETH_MODULE_ID index);
```

Returns

- true - The RetransmissionMaximum feature is supported on the device
- false - The RetransmissionMaximum feature is not supported on the device

Description

This function identifies whether the RetransmissionMaximum feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_ReTxMaxGet
- PLIB_ETH_ReTxMaxSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsRetransmissionMaximum(ETH_MODULE_ID index)

PLIB_ETH_ExistsRMII_Support Function

Identifies whether the RMII_Support feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsRMII_Support(ETH_MODULE_ID index);
```

Returns

- true - The RMII_Support feature is supported on the device
- false - The RMII_Support feature is not supported on the device

Description

This function identifies whether the RMII_Support feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_RMIIResetEnable
- PLIB_ETH_RMIIResetDisable
- PLIB_ETH_RMIIResetIsEnabled
- PLIB_ETH_RMIISpeedGet
- PLIB_ETH_RMIISpeedSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsRMII_Support(ETH_MODULE_ID index)

PLIB_ETH_ExistsRxBufferCountDecrement Function

Identifies whether the RxBufferCountDecrement feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsRxBufferCountDecrement(ETH_MODULE_ID index);
```

Returns

- true - The RxBufferCountDecrement feature is supported on the device
- false - The RxBufferCountDecrement feature is not supported on the device

Description

This function identifies whether the RxBufferCountDecrement feature is available on the Ethernet module. When this function returns true, this function is supported on the device:

- PLIB_ETH_RxBufferCountDecrement

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsRxBufferCountDecrement(ETH_MODULE_ID index)

PLIB_ETH_ExistsRxEnable Function

Identifies whether the ReceiveEnable feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsRxEnable(ETH_MODULE_ID index);
```

Returns

- true - The ReceiveEnable feature is supported on the device
- false - The ReceiveEnable feature is not supported on the device

Description

This function identifies whether the ReceiveEnable feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_RxEnable
- PLIB_ETH_RxDisable
- PLIB_ETH_RxIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsRxEnable(ETH_MODULE_ID index)

PLIB_ETH_ExistsRxPacketDescriptorAddress Function

Identifies whether the RxPacketDescriptorAddress feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsRxPacketDescriptorAddress(ETH_MODULE_ID index);
```

Returns

- true - The RxPacketDescriptorAddress feature is supported on the device

- false - The RxPacketDescriptorAddress feature is not supported on the device

Description

This function identifies whether the RxPacketDescriptorAddress feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_RxPacketDescAddrSet
- PLIB_ETH_RxPacketDescAddrGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsRxPacketDescriptorAddress(ETH_MODULE_ID index)

PLIB_ETH_ExistsStationAddress Function

Identifies whether the StationAddress feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsStationAddress(ETH_MODULE_ID index);
```

Returns

- true - The StationAddress feature is supported on the device
- false - The StationAddress feature is not supported on the device

Description

This function identifies whether the StationAddress feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_StationAddressGet
- PLIB_ETH_StationAddressSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsStationAddress(ETH_MODULE_ID index)

PLIB_ETH_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsStopInIdle(ETH_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_StopInIdleEnable
- PLIB_ETH_StopInIdleDisable
- PLIB_ETH_StopInIdleIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsStopInIdle(ETH_MODULE_ID index)

PLIB_ETH_ExistsTransmitRTS Function

Identifies whether the TransmitRTS feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsTransmitRTS(ETH_MODULE_ID index);
```

Returns

- true - The TransmitRTS feature is supported on the device
- false - The TransmitRTS feature is not supported on the device

Description

This function identifies whether the TransmitRTS feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_TxRTSEnable
- PLIB_ETH_TxRTSDisable
- PLIB_ETH_TxRTSIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsTransmitRTS(ETH_MODULE_ID index)

PLIB_ETH_ExistsTxPacketDescriptorAddress Function

Identifies whether the TxPacketDescriptorAddress feature exists on the Ethernet module.

File

plib_eth.h

C

```
bool PLIB_ETH_ExistsTxPacketDescriptorAddress(ETH_MODULE_ID index);
```

Returns

- true - The TxPacketDescriptorAddress feature is supported on the device
- false - The TxPacketDescriptorAddress feature is not supported on the device

Description

This function identifies whether the TxPacketDescriptorAddress feature is available on the Ethernet module. When this function returns true, these functions are supported on the device:

- PLIB_ETH_TxPacketDescAddrSet
- PLIB_ETH_TxPacketDescAddrGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_ETH_ExistsTxPacketDescriptorAddress(ETH_MODULE_ID index)

i) Data Types and Constants

ETH_AUTOPAD_OPTION Enumeration

Lists the possible automatic padding configurations.

File

plib_eth.h

C

```
typedef enum {
  ETH_AUTOPAD_DISABLED_CHECK_CRC,
  ETH_AUTOPAD_TO_60BYTES_ADD_CRC,
  ETH_AUTOPAD_TO_64BYTES_ADD_CRC,
  ETH_AUTOPAD_BY_PKT_TYPE_ADD_CRC
} ETH_AUTOPAD_OPTION;
```

Description

AutoDetectPad Selection

This enumeration lists the possible automatic padding configurations.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

ETH_INTERRUPT_SOURCES Enumeration

Lists the possible values of ETH_INTERRUPT_SOURCES.

File

plib_eth.h

C

```
typedef enum {
  ETH_TRANSMIT_BVCI_BUS_ERROR_INTERRUPT,
  ETH_RECEIVE_BVCI_BUS_ERROR_INTERRUPT,
  ETH_EMPTY_WATERMARK_INTERRUPT,
  ETH_FULL_WATERMARK_INTERRUPT,
  ETH_RECEIVE_DONE_INTERRUPT,
  ETH_PACKET_PENDING_INTERRUPT,
  ETH_RECEIVE_ACTIVITY_INTERRUPT,
  ETH_TRANSMIT_DONE_INTERRUPT,
  ETH_TRANSMIT_ABORT_INTERRUPT,
  ETH_RECEIVE_BUFFER_NOT_AVAILABLE_INTERRUPT,
  ETH_RECEIVE_FIFO_OVERFLOW_ERROR_INTERRUPT,
  ETH_ALL_INTERRUPT_SOURCES
} ETH_INTERRUPT_SOURCES;
```

Description

ETH Module Interrupt Mask

This enumeration lists the possible values of ETH_INTERRUPT_SOURCES.

Remarks

See also PLIB_ETH_EVENTS in plib_eth_lib.h.

ETH_MIIM_CLK Enumeration

Lists the possible speed selection for the Reduced Media Independent Interface (RMII).

File

plib_eth.h

C

```
typedef enum {
  ETH_MIIM_SYSCLK_DIV_BY_4,
  ETH_MIIM_SYSCLK_DIV_RSVD,
  ETH_MIIM_SYSCLK_DIV_BY_6,
  ETH_MIIM_SYSCLK_DIV_BY_8,
  ETH_MIIM_SYSCLK_DIV_BY_10,
  ETH_MIIM_SYSCLK_DIV_BY_14,
  ETH_MIIM_SYSCLK_DIV_BY_20,
  ETH_MIIM_SYSCLK_DIV_BY_28,
  ETH_MIIM_SYSCLK_DIV_BY_40
} ETH_MIIM_CLK;
```

Description

MII Clock Selection

This enumeration lists the possible speed selection for RMII. The body contains only two states: RMII_10Mbps or RMII_100Mbps.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

ETH_PATTERN_MATCH_MODE Enumeration

Lists the possible pattern match values.

File

plib_eth.h

C

```
typedef enum _ETH_PATTERN_MATCH_MODE_ {
  ETH_PATTERN_MATCH_DISABLED,
  ETH_PATTERN_MATCH_MODE_CHECKSUM_MATCH,
  ETH_PATTERN_MATCH_MODE_STATION_ADDR,
```

```
  ETH_PATTERN_MATCH_MODE_UNICAST_ADDR,
  ETH_PATTERN_MATCH_MODE_BROADCAST_ADDR,
  ETH_PATTERN_MATCH_MODE_HASH_MATCH,
  ETH_PATTERN_MATCH_MODE_MAGIC_PACKET
} ETH_PATTERN_MATCH_MODE;
```

Members

Members	Description
ETH_PATTERN_MATCH_DISABLED	Pattern Match is disabled; pattern match is always unsuccessful
ETH_PATTERN_MATCH_MODE_CHECKSUM_MATCH	Pattern match if (NOTPM = 1 XOR Pattern Match Checksum matches)
ETH_PATTERN_MATCH_MODE_STATION_ADDR	Pattern match if (NOTPM = 1 XOR Pattern Match Checksum matches) AND (Destination Address = Station Address)
ETH_PATTERN_MATCH_MODE_UNICAST_ADDR	Pattern match if (NOTPM = 1 XOR Pattern Match Checksum matches) AND (Destination Address = Unicast Address)
ETH_PATTERN_MATCH_MODE_BROADCAST_ADDR	Pattern match if (NOTPM = 1 XOR Pattern Match Checksum matches) AND (Destination Address = Broadcast Address)
ETH_PATTERN_MATCH_MODE_HASH_MATCH	Pattern match if (NOTPM = 1 XOR Pattern Match Checksum matches) AND (Hash Table Filter match)
ETH_PATTERN_MATCH_MODE_MAGIC_PACKET	Pattern match if (NOTPM = 1 XOR Pattern Match Checksum matches) AND (Packet = Magic Packet)

Description

Pattern Match Modes

This enumeration lists the pattern match mode values.

Remarks

In all pattern match enabled cases, it is the pattern match inversion XOR pattern match checksum AND .

Some states were omitted from the enumeration since they were duplicates.

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

ETH_RECEIVE_FILTER Enumeration

Lists the possible values of the receive filter.

File

plib_eth.h

C

```
typedef enum {
  ETH_HASH_FILTER,
  ETH_MAGIC_PACKET_FILTER,
  ETH_PATTERN_MATCH_INVERSION,
  ETH_CRC_ERROR_FILTER,
  ETH_CRC_OK_FILTER,
  ETH_RUNT_ERROR_FILTER,
  ETH_RUNT_ENABLE_FILTER,
  ETH_UNICAST_FILTER,
  ETH_NOT_ME_UNICAST_FILTER,
  ETH_MULTICAST_FILTER,
  ETH_BROADCAST_FILTER,
  ETH_ENABLE_ALL_FILTER
} ETH_RECEIVE_FILTER;
```

Members

Members	Description
ETH_HASH_FILTER	Enable: Hash Table Filtering.
ETH_MAGIC_PACKET_FILTER	Enable: Magic Packet Filtering.
ETH_PATTERN_MATCH_INVERSION	Enable: Pattern Match Checksum must not match to be accepted.
ETH_CRC_ERROR_FILTER	Enable: Received packet CRC must be invalid to be accepted.
ETH_CRC_OK_FILTER	Enable: Received packet CRC must be valid to be accepted.
ETH_RUNT_ERROR_FILTER	Enable: Received packet must be runt packet to be accepted.
ETH_RUNT_ENABLE_FILTER	Enable: Received packet must not be a runt packet to be accepted.

ETH_UNICAST_FILTER	Enable: Unicast Packets whose destination address matches the station address are accepted.
ETH_NOT_ME_UNICAST_FILTER	Enable: Unicast Packets whose destination address do NOT match the station address are accepted.
ETH_MULTICAST_FILTER	Enable: Multicast Address Packets are accepted.
ETH_BROADCAST_FILTER	Enable: Broadcast Address Packets are accepted.
ETH_ENABLE_ALL_FILTER	Enable: All of the above packets are accepted.

Description

ETH Module Receive Filter Mask

This enumeration lists the possible values of the receive filter.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

ETH_RMII_SPEED Enumeration

Lists the possible speed selection for the Reduced Media Independent Interface (RMII).

File

plib_eth.h

C

```
typedef enum {
  ETH_RMII_10Mbps,
  ETH_RMII_100Mps
} ETH_RMII_SPEED;
```

Description

RMII Speed Selection

This enumeration lists the possible speed selection for RMII. The body contains only two states: RMII_10Mbps or RMII_100Mbps.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files (see processor.h).

Files

Files

Name	Description
plib_eth.h	Defines the Ethernet Peripheral Library Interface.

Description

This section lists the source and header files used by the library.

plib_eth.h

Defines the Ethernet Peripheral Library Interface.

Enumerations

	Name	Description
	_ETH_PATTERN_MATCH_MODE_	Lists the possible pattern match values.
	ETH_AUTOPAD_OPTION	Lists the possible automatic padding configurations.
	ETH_INTERRUPT_SOURCES	Lists the possible values of ETH_INTERRUPT_SOURCES.
	ETH_MIIM_CLK	Lists the possible speed selection for the Reduced Media Independent Interface (RMII).
	ETH_PATTERN_MATCH_MODE	Lists the possible pattern match values.
	ETH_RECEIVE_FILTER	Lists the possible values of the receive filter.
	ETH_RMII_SPEED	Lists the possible speed selection for the Reduced Media Independent Interface (RMII).

Functions

	Name	Description
	PLIB_ETH_AlignErrorCountClear	Sets the count of Ethernet alignment errors to zero.
	PLIB_ETH_AlignErrorCountGet	Gets the count of Ethernet alignment errors.
	PLIB_ETH_AutoDetectPadClear	Clears the EMAC Auto-Pad option.
	PLIB_ETH_AutoDetectPadGet	Gets the current EMAC Auto-Pad setting.
	PLIB_ETH_AutoDetectPadSet	Sets the EMAC Auto-Pad option.
	PLIB_ETH_AutoFlowControlDisable	Disables the Ethernet module Automatic Flow Control logic.
	PLIB_ETH_AutoFlowControlEnable	Enables the Ethernet Automatic Flow Control logic.
	PLIB_ETH_AutoFlowControlIsEnabled	Gets the Ethernet module Automatic Flow Control status.
	PLIB_ETH_BackPresNoBackoffDisable	Disables EMAC backpressure/no back-off.
	PLIB_ETH_BackPresNoBackoffEnable	Enables EMAC back pressure/no back-off.
	PLIB_ETH_BackPresNoBackoffIsEnabled	Gets the EMAC backpressure/no back-off enable status.
	PLIB_ETH_BackToBackIPGGet	Gets the EMAC back-to-back interpacket gap.
	PLIB_ETH_BackToBackIPGSet	Sets the EMAC back-to-back interpacket gap.
	PLIB_ETH_CollisionWindowGet	Gets the EMAC collision window.
	PLIB_ETH_CollisionWindowSet	Sets the EMAC collision window.
	PLIB_ETH_CRCDisable	Disables EMAC CRC.
	PLIB_ETH_CRCEnable	Enables EMAC CRC.
	PLIB_ETH_CRCIsEnabled	Gets the EMAC CRC enable status.
	PLIB_ETH_DataNotValid	Gets the MII management read data not valid status.
	PLIB_ETH_DelayedCRCDisable	Disables EMAC delayed CRC.
	PLIB_ETH_DelayedCRCEnable	Enables EMAC delayed CRC.
	PLIB_ETH_DelayedCRCIsEnabled	Gets the EMAC delayed CRC enable status.
	PLIB_ETH_Disable	Disables the Ethernet module.
	PLIB_ETH_Enable	Enables the Ethernet module.
	PLIB_ETH_EthernetIsBusy	Gets the status value of the Ethernet logic busy.
	PLIB_ETH_ExcessDeferDisable	Disables EMAC excess defer.
	PLIB_ETH_ExcessDeferEnable	Enables EMAC excess defer.
	PLIB_ETH_ExcessDeferIsEnabled	Gets the EMAC excess defer enable status.
	PLIB_ETH_ExistsAlignmentErrorCount	Identifies whether the AlignmentErrorCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsAutoFlowControl	Identifies whether the AutoFlowControl feature exists on the Ethernet module.
	PLIB_ETH_ExistsCollisionCounts	Identifies whether the CollisionCounts feature exists on the Ethernet module.
	PLIB_ETH_ExistsCollisionWindow	Identifies whether the CollisionWindow feature exists on the Ethernet module.
	PLIB_ETH_ExistsEnable	Identifies whether the Enable feature exists on the Ethernet module.
	PLIB_ETH_ExistsEthernetControllerStatus	Identifies whether the EthernetControllerStatus feature exists on the Ethernet module.
	PLIB_ETH_ExistsFCSErrorCount	Identifies whether the FCSErrorCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsFramesTransmittedOK	Identifies whether the FramesTransmittedOK feature exists on the Ethernet module.
	PLIB_ETH_ExistsFramexReceivedOK	Identifies whether the FramexReceivedOK feature exists on the Ethernet module.
	PLIB_ETH_ExistsHashTable	Identifies whether the HashTable feature exists on the Ethernet module.
	PLIB_ETH_ExistsInterPacketGaps	Identifies whether the InterPacketGaps feature exists on the Ethernet module.
	PLIB_ETH_ExistsInterrupt	Identifies whether the Interrupt feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Configuration	Identifies whether the MAC_Configuration feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Resets	Identifies whether the MAC_Resets feature exists on the Ethernet module.
	PLIB_ETH_ExistsMAC_Testing	Identifies whether the MAC_Testing feature exists on the Ethernet module.
	PLIB_ETH_ExistsManualFlowControl	Identifies whether the ManualFlowControl feature exists on the Ethernet module.
	PLIB_ETH_ExistsMaxFrameLength	Identifies whether the MaxFrameLength feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIM_Config	Identifies whether the MIIM_Config feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIM_Indicators	Identifies whether the MIIM_Indicators feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMAddresses	Identifies whether the MIIMAddresses feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMReadWrite	Identifies whether the MIIMReadWrite feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIMScanMode	Identifies whether the MIIMScanMode feature exists on the Ethernet module.
	PLIB_ETH_ExistsMIIWriteReadData	Identifies whether the MIIWriteReadData feature exists on the Ethernet module.

	PLIB_ETH_ExistsPatternMatch	Identifies whether the PatternMatch feature exists on the Ethernet module.
	PLIB_ETH_ExistsPauseTimer	Identifies whether the PauseTimer feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveBufferSize	Identifies whether the ReceiveBufferSize feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveFilters	Identifies whether the ReceiveFilters feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveOverflowCount	Identifies whether the ReceiveOverflowCount feature exists on the Ethernet module.
	PLIB_ETH_ExistsReceiveWmarks	Identifies whether the ReceiveWmarks feature exists on the Ethernet module.
	PLIB_ETH_ExistsRetransmissionMaximum	Identifies whether the RetransmissionMaximum feature exists on the Ethernet module.
	PLIB_ETH_ExistsRMII_Support	Identifies whether the RMII_Support feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxBufferCountDecrement	Identifies whether the RxBufferCountDecrement feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxEnable	Identifies whether the ReceiveEnable feature exists on the Ethernet module.
	PLIB_ETH_ExistsRxPacketDescriptorAddress	Identifies whether the RxPacketDescriptorAddress feature exists on the Ethernet module.
	PLIB_ETH_ExistsStationAddress	Identifies whether the StationAddress feature exists on the Ethernet module.
	PLIB_ETH_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the Ethernet module.
	PLIB_ETH_ExistsTransmitRTS	Identifies whether the TransmitRTS feature exists on the Ethernet module.
	PLIB_ETH_ExistsTxPacketDescriptorAddress	Identifies whether the TxPacketDescriptorAddress feature exists on the Ethernet module.
	PLIB_ETH_FCSErrorCountClear	Sets the value of the Ethernet frame check sequence error to zero.
	PLIB_ETH_FCSErrorCountGet	Gets the count of the frame check sequence error.
	PLIB_ETH_FrameLengthCheckDisable	Disables the EMAC frame length check.
	PLIB_ETH_FrameLengthCheckEnable	Enables the EMAC frame length check.
	PLIB_ETH_FrameLengthCheckIsEnabled	Gets the EMAC frame length check status.
	PLIB_ETH_FramesRxdOkCountClear	Sets the value of the Ethernet received frames 'OK' count to zero.
	PLIB_ETH_FramesRxdOkCountGet	Gets the count of the frames frames received successfully.
	PLIB_ETH_FramesTxdOkCountClear	Sets the count of Ethernet Control frames transmitted to zero.
	PLIB_ETH_FramesTxdOkCountGet	Gets the count of Ethernet Control Frames transmitted successfully.
	PLIB_ETH_FullDuplexDisable	Disables the EMAC full duplex operation.
	PLIB_ETH_FullDuplexEnable	Enables the EMAC full duplex operation.
	PLIB_ETH_FullDuplexIsEnabled	Gets the EMAC full duplex enable status.
	PLIB_ETH_HashTableGet	Gets the value of the Hash table.
	PLIB_ETH_HashTableSet	Sets the Ethernet module Hash table with the new value.
	PLIB_ETH_HugeFrameDisable	Disables the EMAC from receiving huge frames.
	PLIB_ETH_HugeFrameEnable	Enables the EMAC to receive huge frames.
	PLIB_ETH_HugeFrameIsEnabled	Gets the EMAC huge frame enable status.
	PLIB_ETH_InterruptClear	Clears the Ethernet module interrupt source status as a group, using a mask.
	PLIB_ETH_InterruptSet	Sets the Ethernet module interrupt source status as a group, using a mask.
	PLIB_ETH_InterruptsGet	Gets the Ethernet module Interrupt Flag register as a group.
	PLIB_ETH_InterruptSourceDisable	Clears the Ethernet module Interrupt Enable register as a group, using a mask.
	PLIB_ETH_InterruptSourceEnable	Sets the Ethernet module Interrupt Enable register in a group, using a mask.
	PLIB_ETH_InterruptSourceIsEnabled	Gets the Ethernet module Interrupt Enable register singularly or as a group.
	PLIB_ETH_InterruptSourcesGet	Returns the entire interrupt enable register.
	PLIB_ETH_InterruptStatusGet	Gets the Ethernet module Interrupt Flag register as a group, using a mask.
	PLIB_ETH_IsEnabled	Gets the Ethernet module enable status.
	PLIB_ETH_LinkHasFailed	Gets the MII management link fail status.
	PLIB_ETH_LongPreambleDisable	Disables EMAC long preamble enforcement.
	PLIB_ETH_LongPreambleEnable	Enables EMAC long preamble enforcement.
	PLIB_ETH_LongPreambleIsEnabled	Gets the EMAC long preamble enforcement enable status.
	PLIB_ETH_LoopbackDisable	Disables the EMAC loopback logic.
	PLIB_ETH_LoopbackEnable	Enables the EMAC loopback logic.
	PLIB_ETH_LoopbackIsEnabled	Gets the EMAC Loopback interface enable status.
	PLIB_ETH_ManualFlowControlDisable	Disable Ethernet module Manual Flow Control logic.
	PLIB_ETH_ManualFlowControlEnable	Enables the Ethernet Manual Flow Control logic.
	PLIB_ETH_ManualFlowControlIsEnabled	Gets the Ethernet module Manual Flow Control enable status.
	PLIB_ETH_MaxFrameLengthGet	Gets the EMAC maximum frame length.

	PLIB_ETH_MaxFrameLengthSet	Sets the EMAC maximum frame length.
	PLIB_ETH_MCSRxResetDisable	Disables the reset EMAC Control Sub-layer/Receive domain logic.
	PLIB_ETH_MCSRxResetEnable	Enables the reset EMAC Control Sub-layer/Receive domain logic.
	PLIB_ETH_MCSRxResetIsEnabled	Gets the EMAC Control Sub-layer/Receive domain logic reset status.
	PLIB_ETH_MCSTxResetDisable	Disables the reset EMAC Control Sub-layer/Transmit domain logic.
	PLIB_ETH_MCSTxResetEnable	Enables the reset of EMAC Control Sub-layer/Transmit domain logic.
	PLIB_ETH_MCSTxResetIsEnabled	Gets the EMAC Control Sub-layer/Transmit domain logic reset status.
	PLIB_ETH_MIIMClockGet	Gets the current EMAC MIIM clock selection.
	PLIB_ETH_MIIMClockSet	Sets the EMAC MIM clock selection.
	PLIB_ETH_MIIMIsBusy	Gets the MII management busy status.
	PLIB_ETH_MIIMIsScanning	Gets the MII Management scanning status.
	PLIB_ETH_MIIMNoPreDisable	Disables EMAC No Preamble (allows preamble).
	PLIB_ETH_MIIMNoPreEnable	Enables EMAC MIIM No Preamble (suppresses preamble).
	PLIB_ETH_MIIMNoPreIsEnabled	Gets the EMAC MIIM No Preamble enable status.
	PLIB_ETH_MIIMReadDataGet	Gets EMAC MIIM management read data after a MII read cycle has completed.
	PLIB_ETH_MIIMReadStart	Initiates an MII management read command.
	PLIB_ETH_MIIMResetDisable	Disables EMAC Reset MII Management.
	PLIB_ETH_MIIMResetEnable	Enables EMAC Reset Media Independent Interface (MII) Management.
	PLIB_ETH_MIIMResetIsEnabled	Gets the EMAC Reset MII Management enable status.
	PLIB_ETH_MIIMScanIncrDisable	Disables the EMAC MIIM Scan Increment.
	PLIB_ETH_MIIMScanIncrEnable	Enables EMAC MIIM Scan Increment.
	PLIB_ETH_MIIMScanIncrIsEnabled	Gets the EMAC MIIM scan increment enable status.
	PLIB_ETH_MIIMScanModeDisable	Disables MIIM scan mode.
	PLIB_ETH_MIIMScanModeEnable	Enables MIIM scan mode.
	PLIB_ETH_MIIMScanModeIsEnabled	Gets the MII management scan enable status.
	PLIB_ETH_MIIMWriteDataSet	Sets the EMAC MIIM write data before initiating an MII write cycle.
	PLIB_ETH_MIIMWriteStart	Initiates an MII management write command.
	PLIB_ETH_MIIResetDisable	Disables the EMAC Soft reset.
	PLIB_ETH_MIIResetEnable	Enables the EMAC MIIM Soft reset.
	PLIB_ETH_MIIResetIsEnabled	Gets EMAC MIIM Soft Reset enable status.
	PLIB_ETH_MultipleCollisionCountClear	Sets the value of the Ethernet multiple collision frame count to zero.
	PLIB_ETH_MultipleCollisionCountGet	Gets the count of the frames transmitted successfully after there was more than one collision.
	PLIB_ETH_NoBackoffDisable	Disables EMAC no back-offs.
	PLIB_ETH_NoBackoffEnable	Enables EMAC no back-off.
	PLIB_ETH_NoBackoffIsEnabled	Gets the EMAC no back-off enable status.
	PLIB_ETH_NonBackToBackIPG1Get	Gets the EMAC non-back-to-back interpacket gap register 1.
	PLIB_ETH_NonBackToBackIPG1Set	Sets the EMAC non-back-to-back interpacket gap register 1.
	PLIB_ETH_NonBackToBackIPG2Get	Gets the EMAC non-back-to-back interpacket gap register 2.
	PLIB_ETH_NonBackToBackIPG2Set	Sets the EMAC non-back-to-back interpacket gap register 2.
	PLIB_ETH_PassAllDisable	Disables the EMAC PassAll.
	PLIB_ETH_PassAllEnable	Enables the EMAC PassAll.
	PLIB_ETH_PassAllIsEnabled	Gets the EMAC PassAll enable status.
	PLIB_ETH_PatternMatchChecksumGet	Gets the value of the pattern match checksum register.
	PLIB_ETH_PatternMatchChecksumSet	Sets the Ethernet module pattern match checksum register with the new value.
	PLIB_ETH_PatternMatchGet	Gets the value of the selected pattern match mask register.
	PLIB_ETH_PatternMatchModeGet	Gets the value of the selected pattern match mask register.
	PLIB_ETH_PatternMatchModeSet	Sets the Ethernet module pattern match mode.
	PLIB_ETH_PatternMatchOffsetGet	Gets the value of the patter match offset register.
	PLIB_ETH_PatternMatchOffsetSet	Sets the Ethernet module patter match offset register with the new value.
	PLIB_ETH_PatternMatchSet	Sets the Ethernet module pattern match mask register with the new value.
	PLIB_ETH_PauseTimerGet	Gets the Pause Timer value used for flow control.
	PLIB_ETH_PauseTimerSet	Sets the Pause Timer value used for flow control.
	PLIB_ETH_PHYAddressGet	Gets the EMAC MIIM management PHY address.
	PLIB_ETH_PHYAddressSet	Sets the EMAC MIIM PHY address.

	PLIB_ETH_PurePreambleDisable	Disables EMAC pure preamble enforcement.
	PLIB_ETH_PurePreambleEnable	Enables EMAC pure preamble enforcement.
	PLIB_ETH_PurePreambleIsEnabled	Gets EMAC pure preamble enforcement enable status.
	PLIB_ETH_ReceiveBufferSizeGet	Gets the Ethernet module receive buffer size.
	PLIB_ETH_ReceiveBufferSizeSet	Sets the Ethernet module receive buffer size.
	PLIB_ETH_ReceiveDisable	Disables the EMAC from receiving frames.
	PLIB_ETH_ReceiveEnable	Enables the EMAC to receive the frames.
	PLIB_ETH_ReceiveFilterDisable	Disables the specified receive filter.
	PLIB_ETH_ReceiveFilterEnable	Enables the specified receive filter.
	PLIB_ETH_ReceiveFilterIsEnable	Disables the specified receive filter.
	PLIB_ETH_ReceiveIsBusy	Gets the Ethernet receive logic busy status.
	PLIB_ETH_ReceiveIsEnabled	Gets the Ethernet EMAC receive enable status.
	PLIB_ETH_RegisterAddressGet	Gets the EMAC MIIM management register address.
	PLIB_ETH_RegisterAddressSet	Sets EMAC MIIM register address.
	PLIB_ETH_ReTxMaxGet	Gets the EMAC maximum retransmissions.
	PLIB_ETH_ReTxMaxSet	Sets the EMAC maximum retransmissions.
	PLIB_ETH_RMIIResetDisable	Disables EMAC Reset RMII.
	PLIB_ETH_RMIIResetEnable	Enables EMAC Reset RMII.
	PLIB_ETH_RMIIResetIsEnabled	Gets the EMAC Reset RMII enable status.
	PLIB_ETH_RMIISpeedGet	Gets the current EMAC RMII speed.
	PLIB_ETH_RMIISpeedSet	Sets EMAC RMII Speed.
	PLIB_ETH_RxBufferCountDecrement	Causes the Receive Descriptor Buffer Counter to decrement by 1.
	PLIB_ETH_RxDisable	Disables the Ethernet module receive logic.
	PLIB_ETH_RxEmptyWmarkGet	Gets the Ethernet module receive empty watermark.
	PLIB_ETH_RxEmptyWmarkSet	Sets the Ethernet module receive empty water mark.
	PLIB_ETH_RxEnable	Enables the Ethernet receive logic.
	PLIB_ETH_RxFullWmarkGet	Gets the Ethernet module to receive a full watermark.
	PLIB_ETH_RxFullWmarkSet	Sets the Ethernet module to receive a full watermark.
	PLIB_ETH_RxFuncResetDisable	Disables the EMAC reset receive function logic.
	PLIB_ETH_RxFuncResetEnable	Enables the EMAC reset receive function logic.
	PLIB_ETH_RxFuncResetIsEnabled	Gets the EMAC reset receive function status.
	PLIB_ETH_RxIsEnabled	Gets the Ethernet module receive enable status.
	PLIB_ETH_RxOverflowCountClear	Sets the value of the Ethernet receive overflow count to zero.
	PLIB_ETH_RxOverflowCountGet	Gets the count of the dropped Ethernet receive frames.
	PLIB_ETH_RxPacketCountGet	Gets the value of the receive packet buffer count.
	PLIB_ETH_RxPacketDescAddrGet	Gets the address of the next receive descriptor.
	PLIB_ETH_RxPacketDescAddrSet	Sets the Ethernet module receive packet descriptor start address.
	PLIB_ETH_RxPauseDisable	Disables the EMAC receive flow control.
	PLIB_ETH_RxPauseEnable	Enables the EMAC receive flow control.
	PLIB_ETH_RxPauseIsEnabled	Gets the EMAC receive flow pause enable status.
	PLIB_ETH_ShortcutQuantaDisable	Disables EMAC shortcut pause quanta.
	PLIB_ETH_ShortcutQuantaEnable	Enables EMAC shortcut pause quanta.
	PLIB_ETH_ShortcutQuantaIsEnabled	Gets EMAC shortcut pause quanta enable status.
	PLIB_ETH_SimResetDisable	Disables the EMAC simulation reset.
	PLIB_ETH_SimResetEnable	Enables the EMAC simulation reset.
	PLIB_ETH_SimResetIsEnabled	Gets the EMAC simulation reset status.
	PLIB_ETH_SingleCollisionCountClear	Sets the value of the Ethernet single collision frame count to zero.
	PLIB_ETH_SingleCollisionCountGet	Gets the count of the frames transmitted successfully on a second attempt.
	PLIB_ETH_StationAddressGet	Gets the selected EMAC station address.
	PLIB_ETH_StationAddressSet	Sets the selected EMAC Station Address.
	PLIB_ETH_StopInIdleDisable	Ethernet module operation will continue when the device enters Idle mode.
	PLIB_ETH_StopInIdleEnable	Ethernet module operation will stop when the device enters Idle mode.
	PLIB_ETH_StopInIdleIsEnabled	Gets the Ethernet module Stop in Idle mode enable status.
	PLIB_ETH_TestBackPressDisable	Disables EMAC Test backpressure.
	PLIB_ETH_TestBackPressEnable	Enables EMAC Test backpressure.

	PLIB_ETH_TestBackPressIsEnabled	Gets the EMAC test backpressure enable status.
	PLIB_ETH_TestPauseDisable	Disables EMAC test pause.
	PLIB_ETH_TestPauseEnable	Enables EMAC test pause.
	PLIB_ETH_TestPauseIsEnabled	Gets the EMAC test pause enable status.
	PLIB_ETH_TransmitIsBusy	Gets the status value of the Ethernet transmit logic busy status
	PLIB_ETH_TxFuncResetDisable	Disables the EMAC Transmit function reset.
	PLIB_ETH_TxFuncResetEnable	Enables the EMAC transmit function reset.
	PLIB_ETH_TxFuncResetIsEnabled	Gets the EMAC Transmit function reset status.
	PLIB_ETH_TxPacketDescAddrGet	Gets the address of the next descriptor to be transmitted.
	PLIB_ETH_TxPacketDescAddrSet	Sets the Ethernet module transmit packet descriptor start address.
	PLIB_ETH_TxPauseDisable	Disables the transmission of Pause frames.
	PLIB_ETH_TxPauseEnable	Enables the transmission Pause frames.
	PLIB_ETH_TxPauseIsEnabled	Gets the Ethernet module enable status.
	PLIB_ETH_TxRTSDisable	Aborts an Ethernet module transmission.
	PLIB_ETH_TxRTSEnable	Enables the Ethernet transmit request to send.
	PLIB_ETH_TxRTSIsEnabled	Gets the Ethernet module transmit request to send status.

Description

Ethernet Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Ethernet Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Ethernet module.

File Name

plib_eth.h

Company

Microchip Technology Inc.

GLCD Controller Peripheral Library

This section describes the GLCD Controller (GLCD Controller) Peripheral Library.

Introduction

This library provides a low-level abstraction of the GLCD Controller Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by abstracting differences from one microcontroller variant to another.

Description

The primary function of the Graphics LCD (GLCD) controller is to directly interface with display glasses and to control pixels on a screen. The GLCD controller transfers display data from a memory device and formats it for a display device. It also provides accelerated on-the-fly rendering of vertical and horizontal lines, rectangles, and copying of a rectangular area between different locations on the screen.

After initialization, the GLCD controller will perform rendering through DMA, thereby off-loading the CPU.

The GLCD controller can support three rendering layers. Each layer supports the configurable stride and alpha blending feature. Each layer supports different input pixel formats such as RGBA8888, ARGB8888, RGB888, RGB565, RGBA5551, YUYV, RGB232, LUT8 and grayscale. These layers can be combined together into a single layer with a configurable output format. The output format can be RGB888, RGB666, RGB323, YUYV, Serial 3-beat (RGB), Serial 4-beat (RGBA), Two-phase 12-bit mode, and BT656. For the LUT8 input format, the Look-up Table (LUT) resides within the device and is configurable.

The device supports a programmable cursor and the cursor data can be set in the device memory. The polarity of signals HSYNC, VSYNC, DE, and PCLK is configurable.

Using the Library

This topic describes the basic architecture of the GLCD Controller Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_glcd.h`

The interface to the GLCD Controller Peripheral Library is defined in the `plib_glcd.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the GLCD Controller Peripheral Library must include `peripheral.h`.

Library File:

The GLCD Controller Peripheral Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

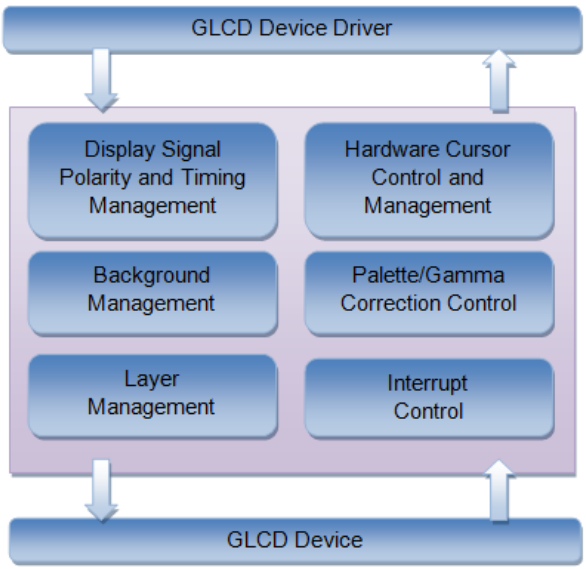
Hardware Abstraction Model

This library provides the low-level abstraction of the GLCD Controller module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

The GLCD Peripheral Library software is depicted by the following block diagram:

GLCD Controller Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Deadman Timer module.

Library Interface Section	Description
General Configuration Functions	Functions that enable/disable the GLCD and set the RGB Sequential mode.
Display Signal Polarity and Timing Management Functions	Functions that handles polarity and timing of display signals such as VSYNC, HSYNC, DE, and PCLK.
Background Management Functions	Functions that handle setting the background color.
Layer Management	Functions that manage layer configuration and overlay.
Hardware Cursor Control and Management Functions	Functions that handle cursor data settings and cursor drawing position control.
Palette and Gamma Correction Control Functions	Functions that control hardware color palette/gamma table data settings.
Interrupt Control Functions	Functions that control device interrupt settings.
Miscellaneous Control Functions	Functions that control miscellaneous GLCD features.
Feature Existence Functions	Functions that determine existence of certain features.

How the Library Works

This section provides information on how the GLCD Controller Peripheral Library works.

General Configuration

Provides general configuration information.

Description

General configuration of the GLCD Controller includes enabling and disabling the GLCD module globally.

Enable/Disable GLCD

Enabling and disabling the GLCD are the basic routines that may be called during initialization and exit routines of the application.

Example:

```
// Enable the GLCD controller
PLIB_GLCD_Enable ( GLCD_ID_0 );
// Disable the GLCD
PLIB_GLCD_Disable ( GLCD_ID_0 );
// Check the enable status of the GLCD controller.
if(PLIB_GLCD_IsEnabled( GLCD_ID_0 ))
{
// application code
}
```

GLCD RGB Sequential Mode Set

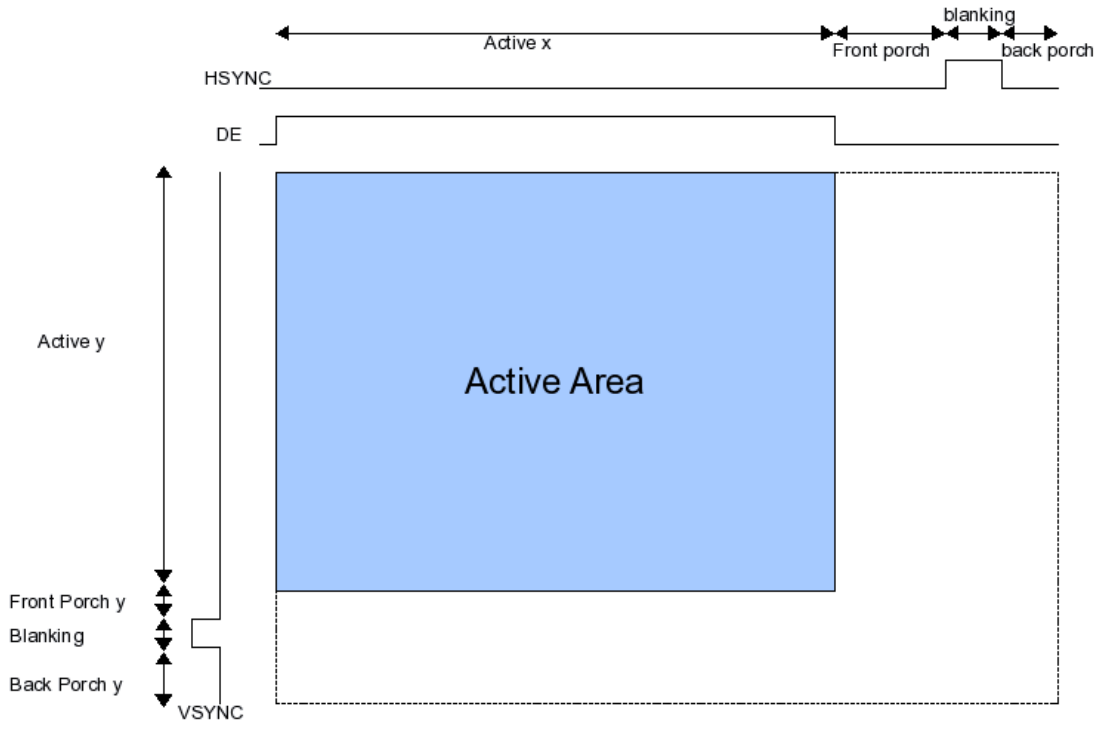
The GLCD RGB Sequential mode controls the GLCD output pins. The function used to set the RGB sequential mode is PLIB_GLCD_RGBSequentialModeSet. This basic function may be called during initialization and exit routines of the application.

Display Signal Polarity and Timing Management

Provides information on display signal polarity.

Description

A GLCD controller peripheral is connected the display using the following pins: HSYNC, VSYNC, GCLK, GEN and GD<23:0>. The HSYNC, VSYNC, GCLK and GEN polarity can be set using the PLIB_GLCD_SignalPolaritySet function. The signal polarity can be either positive or negative based on the display requirements. The GLCD source clock and pixel clock frequency can be controlled using the PLIB_GLCD_FormattingClockDivideEnable and PLIB_GLCD_ClockDividerSet functions. The timing of the signals, HSYNC, VSYNC, and GEN, is based on the display requirements and can be programmed to suit different display resolutions. The functions used to modify the HSYNC, VSYNC and GEN timing parameters are PLIB_GLCD_ResolutionXYSet, PLIB_GLCD_BlankingXYSet PLIB_GLCD_FrontPorchXYSet, and PLIB_GLCD_BackPorchXYSet.

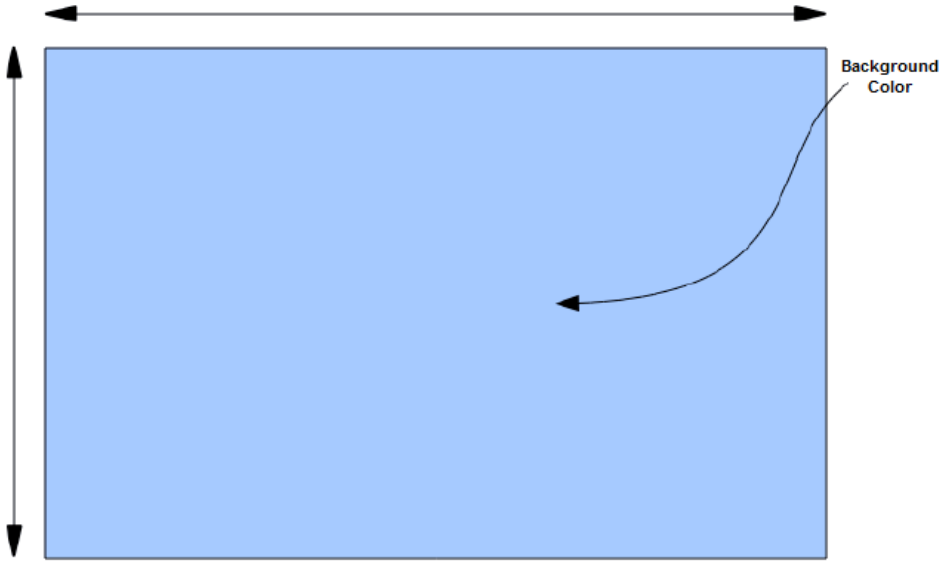


Display Background Management

Provides information on display background management.

Description

The GLCD Controller supports three layers and a background layer. A background layer is a basic layer that can be filled with one color of format RGBA8888 using the PLIB_GLCD_BackgroundColorSet function. The other layers will be applied on top of the background layer with a requested blending mode.

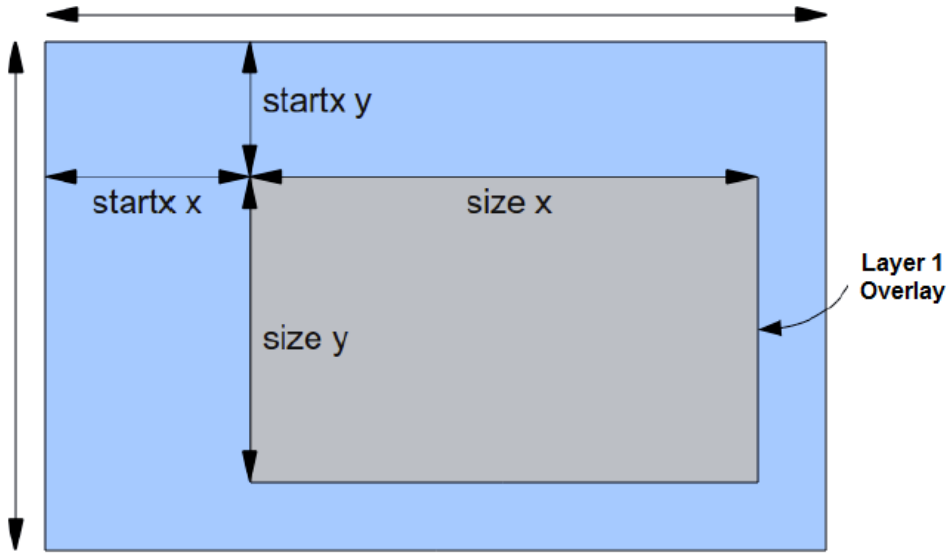


Layer Management

Provides information on layer management.

Description

The GLCD Controller supports three drawing layers. Each layer can be enabled or disabled using the PLIB_GLCD_LayerEnable or PLIB_GLCD_LayerDisable functions, respectively. Each layer's start location, start (x, y), can be set using the PLIB_GLCD_LayerStartXYSet function and the size, size(x, y), can be set using the PLIB_GLCD_LayerSizeXYSet function.

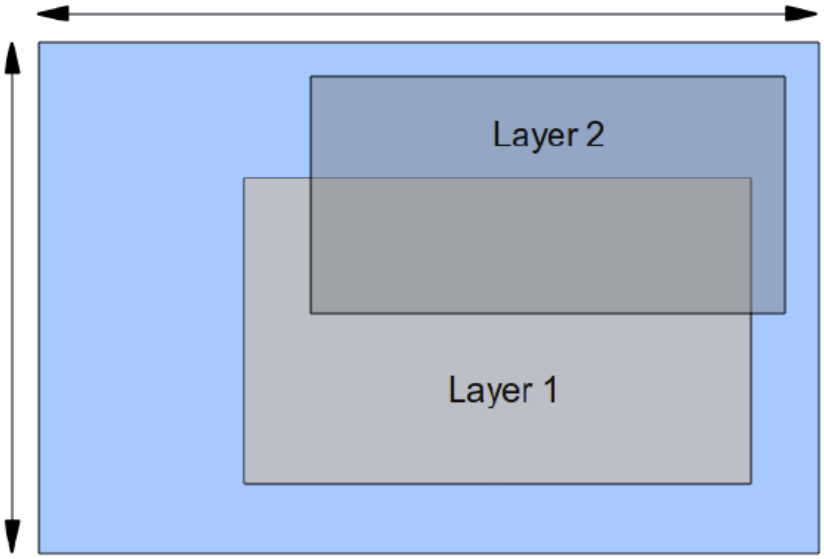


Layer resolution and stride can be set using the PLIB_GLCD_LayerResXYSet and PLIB_GLCD_LayerStrideSet functions, respectively. A bi-linear filter can be applied to each layer using the PLIB_GLCD_LayerBilinearFilterEnable function.

Each layer supports different color formats and the color format can be set using the PLIB_GLCD_LayerColorModeSet function. Each Layer can

have a different global alpha value, which can be set using the PLIB_GLCD_LayerGlobalAlphaSet function. Each Layer alpha can be premultiplied by global alpha using the PLIB_GLCD_LayerPreMultiplyImageAlphaEnable function; otherwise, the PLIB_GLCD_LayerPreMultiplyImageAlphaDisable function is used. To force global alpha value as the layer alpha the PLIB_GLCD_LayerForceWithGlobalAlphaEnable function is called; otherwise, the PLIB_GLCD_LayerForceWithGlobalAlphaDisble function is called.

The next layer is overlayed on a previous layer. Two layers can be blended based on source and destination layer blending functions. The source and destination blending functions can be set using the PLIB_GLCD_LayerSrcBlendFuncSet and PLIB_GLCD_LayerDestBlendFuncSet functions, respectively. The layer overlay and blending is not restricted by the layer color format. Two layers with a different color format can be blended together.



With the exception of the background layer, the other three layers have no memory space allocated on its own into GLCD device. The application must allocate GLCD accessible memory into the controller RAM and pass the address of the allocated memory to the GLCD. The address of the allocated memory for the layer can be set into the GLCD using the PLIB_GLCD_LayerBaseAddressSet function.

Hardware Cursor Control and Management

Provides information on hardware cursor control and management.

Description

The GLCD Controller supports a fully programmable 32 x 32 16-color hardware cursor. The cursor can be enabled or disabled using the PLIB_GLCD_CursorEnable or PLIB_GLCD_CursorDisbale functions, respectively.

The position of the cursor is set using the PLIB_GLCD_CursorXYSet function. Each pixel value of the cursor bitmap is represented by 4 bits. The first 32-bit word contains 8 pixels of the cursor bitmap data starting from pixel position (0, 7) to pixel (0, 0). The remainder of the cursor bitmap data continues similarly to the first 32-bit word. The complete cursor bitmap data is represented by 128 x 32 bit words.

This cursor data is set using the PLIB_GLCD_CursorDataSet function. The 4-bit value representing the pixel in the cursor bitmap data is used as the index of the cursor Color Look-up Table (CLUT). The cursor CLUT is set using the PLIB_GLCD_CursorLUTSet function. The color format of the cursor CLUT is XRGB, where 'X' = unimplemented. The size of cursor CLUT is 16 colors.

Palette and Gamma Correction Control

Provides information on palette/gamma correction control.

Description

For a layer with a source format set to 8-bit color palette look-up table (LUT8), the global CLUT can be set in the GLCD using the PLIB_GLCD_GlobalColorLUTSet function. The format of each color in the CLUT is XRGB8888, where 'X' = unimplemented. The size of the CLUT is 256 colors. The same global CLUT is used to map RGB values to new RGB values for the purpose of gamma correction. The PLIB_GLCD_PaletteGammaRampEnable or PLIB_GLCD_PaletteGammaRampDisable function can be used to enable or disable the Palette/Gamma correction feature.

Interrupt Control

Provides information interrupt control

Description

The GLCD Controller can trigger an interrupt on HSYNC and VSYNC. To enable or disable interrupt on HSYNC the PLIB_GLCD_HSyncInterruptEnable or PLIB_GLCD_HSyncInterruptDisable functions is used, respectively. To enable or disable interrupt on VSYNC the PLIB_GLCD_VSyncInterruptEnable or PLIB_GLCD_VSyncInterruptDisable function is used, respectively. The interrupt trigger can be set to either level or edge triggered using the PLIB_GLCD_IRQTriggerControlSet function.

Miscellaneous Control

Provides information on miscellaneous control features.

Description

The other GLCD feature controls that were not covered in the previous sections are: dithering, VSYNC for single cycle per line, force output blank, and YUV output.

The dithering feature can be enabled or disabled using the PLIB_GLCD_DitheringEnable or PLIB_GLCD_DitheringDisable functions, respectively.

VSYNC for single cycle per line is enabled or disabled using the PLIB_GLCD_SingleCyclePerLineVsyncEnable or PLIB_GLCD_SingleCyclePerLineVsyncDisable functions, respectively.

The forcing of GLCD output to blank is enabled or disabled using the PLIB_GLCD_ForceOutputBlankEnable or PLIB_GLCD_ForceOutputBlankDisable functions, respectively.

If the GLCD RGB Sequential mode is set to YUYU or BT656 format, the function, PLIB_GLCD_YUVOutputEnable, must be called; otherwise, the PLIB_GLCD_YUVOutputDisable function must be called.

The GLCD Peripheral Library also provides access to various GLCD status flags. To verify if the last row is currently displayed, the function, PLIB_GLCD_IsLastRowm is called. To determine the signal level of DE, HSYNC and VSYNC, the functions, PLIB_GLCD_DESignalLevelGet, PLIB_GLCD_HSyncSignalLevelGet, and PLIB_GLCD_VSyncSignalLevelGet are called, respectively. To know whether GLCD is in active vertical blanking state the PLIB_GLCD_IsVerticalBlankingActive function is called.

Configuring the Library

The library is configured for the supported GLCD Controller module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_GLCD_Disable	Disables the Graphics LCD Controller.
	PLIB_GLCD_Enable	Enables the Graphics LCD Controller.
	PLIB_GLCD_RGBSequentialModeSet	Sets the RGB output sequential mode.

b) Display Signal Polarity and Timing Management Functions

	Name	Description
	PLIB_GLCD_SignalPolarityGet	Gets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.
	PLIB_GLCD_SignalPolaritySet	Sets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.
	PLIB_GLCD_BackPorchXGet	Gets X Axis Back Porch.
	PLIB_GLCD_BackPorchXYSet	Sets the back porch on the x and y axis for the Graphics LCD Controller.
	PLIB_GLCD_BackPorchYGet	Gets Y Axis Back Porch.
	PLIB_GLCD_BlankingXYSet	Sets the blanking period on the x and y axis of the Graphics LCD Controller.
	PLIB_GLCD_FormattingClockDivideDisable	Disbale the Clock Formatting feature.
	PLIB_GLCD_FormattingClockDivideEnable	Enable the Clock Formatting feature.
	PLIB_GLCD_FormattingClockDivideIsEnabled	Verify whether Clock Formatting feature is enabled.
	PLIB_GLCD_FrontPorchXGet	Gets X Axis Front Porch.
	PLIB_GLCD_FrontPorchXYSet	Sets the front porch on the x and y axis for the Graphics LCD Controller.
	PLIB_GLCD_FrontPorchYGet	Gets Y Axis Front Porch.
	PLIB_GLCD_LinesPrefetchGet	Gets value of lines to be prefetched.
	PLIB_GLCD_ResolutionXGet	Gets X Axis Resolution.
	PLIB_GLCD_ResolutionYGet	Gets Y Axis Resolution.
	PLIB_GLCD_BlankingXGet	Gets the X Axis Blanking Period.

	PLIB_GLCD_BlankingYGet	Gets the Y Axis Blanking Period.
	PLIB_GLCD_LinesPrefetchSet	Sets the clock controls of the Graphics LCD Controller.
	PLIB_GLCD_ClockDividerSet	Sets clock controls of the Graphics LCD Controller.
	PLIB_GLCD_ClockDividerGet	Gets Clock Divider value.
	PLIB_GLCD_ResolutionXYSet	Sets the display resolution for the Graphics LCD Controller.

c) Background Management Functions

	Name	Description
	PLIB_GLCD_BackgroundColorGet	Gets the value of the Background Color.
	PLIB_GLCD_BackgroundColorSet	Sets the background color.

d) Layer Management Functions

	Name	Description
	PLIB_GLCD_LayerBaseAddressGet	Gets the Layer Base Address.
	PLIB_GLCD_LayerBaseAddressSet	Sets the base address of layer surface of the Graphics LCD Controller.
	PLIB_GLCD_LayerBilinearFilterDisable	Disables the layer Bilinear filter of the Graphics LCD Controller.
	PLIB_GLCD_LayerBilinearFilterEnable	Enables the layer Bilinear filter of the Graphics LCD Controller.
	PLIB_GLCD_LayerBilinearFilterIsEnabled	Verify whether Layer Bilinear Filter is enabled.
	PLIB_GLCD_LayerColorModeGet	Gets the Layer Color Mode.
	PLIB_GLCD_LayerColorModeSet	Sets the layer color mode of the Graphics LCD Controller.
	PLIB_GLCD_LayerDestBlendFuncGet	Gets the Layer Destination Blend Function.
	PLIB_GLCD_LayerDestBlendFuncSet	Sets the layer destination blend function of the Graphics LCD Controller.
	PLIB_GLCD_LayerDisable	Disables the layer of the Graphics LCD Controller.
	PLIB_GLCD_LayerEnable	Enables the layer of the Graphics LCD Controller.
	PLIB_GLCD_LayerForceWithGlobalAlphaDisable	Disables the Layer Force Global Alpha feature.
	PLIB_GLCD_LayerForceWithGlobalAlphaEnable	Enable the Layer Force Global Alpha feature.
	PLIB_GLCD_LayerForceWithGlobalAlphaIsEnabled	Verify whether Layer Force with Global Alpha Feature is enabled.
	PLIB_GLCD_LayerGlobalAlphaGet	Gets the Layer Global Alpha value.
	PLIB_GLCD_LayerGlobalAlphaSet	Sets the layer global alpha of the Graphics LCD Controller.
	PLIB_GLCD_LayerIsEnabled	Verify whether Layer is enabled.
	PLIB_GLCD_LayerPreMultiplyImageAlphaDisable	Disable Layer Pre-Multiply Image Alpha Feature.
	PLIB_GLCD_LayerPreMultiplyImageAlphaEnable	Enable Layer Pre-Multiply Image Alpha Feature.
	PLIB_GLCD_LayerPreMultiplyImageAlphaIsEnabled	Verify whether the Layer Pre-Multiply Image Alpha Feature is enabled.
	PLIB_GLCD_LayerResXGet	Gets the Layer X Axis Resolution.
	PLIB_GLCD_LayerResXYSet	Sets the layer resolution in pixels.
	PLIB_GLCD_LayerResYGet	Gets the Layer Y Axis Resolution.
	PLIB_GLCD_LayerSizeXGet	Gets the Layer X Axis Size.
	PLIB_GLCD_LayerSizeXYSet	Sets the layer size x and size y of the Graphics LCD Controller.
	PLIB_GLCD_LayerSizeYGet	Gets the Layer Y Axis Size.
	PLIB_GLCD_LayerSrcBlendFuncGet	Gets the Layer Source Blend Function.
	PLIB_GLCD_LayerSrcBlendFuncSet	Sets the layer source blend function of the Graphics LCD Controller.
	PLIB_GLCD_LayerStartXGet	Gets the Layer X Axis Start Position.
	PLIB_GLCD_LayerStartXYSet	Sets the layer start x and start y of the Graphics LCD Controller.
	PLIB_GLCD_LayerStartYGet	Gets the Layer Y Axis Start Position.
	PLIB_GLCD_LayerStrideGet	Gets the Layer Stride value.
	PLIB_GLCD_LayerStrideSet	Sets the layer surface stride of the Graphics LCD Controller.

e) Hardware Cursor Control and Management Functions

	Name	Description
	PLIB_GLCD_CursorDataGet	Gets the Cursor Data at given Index.
	PLIB_GLCD_CursorDataSet	Sets the cursor image data.
	PLIB_GLCD_CursorDisable	Disables the cursor of the Graphics LCD Controller.
	PLIB_GLCD_CursorEnable	Enables the cursor of the Graphics LCD Controller.
	PLIB_GLCD_CursorIsEnabled	Verifies whether the cursor is enabled.
	PLIB_GLCD_CursorLUTGet	Gets the color from Cursor LUT at given Index.

	PLIB_GLCD_CursorLUTSet	Sets the cursor color look-up table (LUT) in XRGB8888 format.
	PLIB_GLCD_CursorXGet	Gets the cursor X Axis Position.
	PLIB_GLCD_CursorXYSet	Sets the x and y coordinates of the Graphics LCD Controller cursor.
	PLIB_GLCD_CursorYGet	Gets the Cursor Y Axis Position.

f) Palette and Gamma Correction Control Functions

	Name	Description
	PLIB_GLCD_GlobalColorLUTGet	Gets the Color from Global Color LUT at given Index.
	PLIB_GLCD_GlobalColorLUTSet	Set Global Color LUT.
	PLIB_GLCD_PaletteGammaRampDisable	Disables the palette / gamma ramp of the Graphics LCD Controller.
	PLIB_GLCD_PaletteGammaRampEnable	Enables the palette / gamma ramp of the Graphics LCD Controller.
	PLIB_GLCD_PaletteGammaRampIsEnabled	Verify whether Palette / Gamma Ramp feature is enabled.

g) Interrupt Control Functions

	Name	Description
	PLIB_GLCD_HSyncInterruptDisable	Disables interrupts at Hsync.
	PLIB_GLCD_HSyncInterruptEnable	Enables interrupts at Hsync.
	PLIB_GLCD_HSyncInterruptIsEnabled	Verify whether HSYNC Interrupt is enabled.
	PLIB_GLCD_HSyncSignalLevelGet	Gets the Hsync signal level.
	PLIB_GLCD_IRQTriggerControlGet	Gets the IRQ Trigger Control value.
	PLIB_GLCD_IRQTriggerControlSet	Sets the IRQ trigger control.
	PLIB_GLCD_VSyncInterruptDisable	Disables interrupts at Vsync.
	PLIB_GLCD_VSyncInterruptEnable	Enables interrupts at Vsync.
	PLIB_GLCD_VSyncInterruptIsEnabled	Verify whether VSYNC Interrupt is enabled.
	PLIB_GLCD_VSyncSignalLevelGet	Gets the Vsync signal level.

h) Miscellaneous Control Functions

	Name	Description
	PLIB_GLCD_DitheringDisable	Disables the Dithering feature of the Graphics LCD Controller.
	PLIB_GLCD_DitheringEnable	Enables the Dithering feature of the Graphics LCD Controller.
	PLIB_GLCD_ForceOutputBlankDisable	Disable Force Output Blank feature.
	PLIB_GLCD_ForceOutputBlankEnable	Enable Force Output Blank feature.
	PLIB_GLCD_SingleCyclePerLineVsyncDisable	Clears VSYNC on single cycle per line.
	PLIB_GLCD_SingleCyclePerLineVsyncEnable	Sets VSYNC on single cycle per line.
	PLIB_GLCD_SingleCyclePerLineVsyncIsEnabled	Verify whether VSYNC on Single Cycle Per Line feature is enabled.
	PLIB_GLCD_YUVOutputDisable	Disables the output of the Graphics LCD Controller in YUV format.
	PLIB_GLCD_YUVOutputEnable	Enables the output of the Graphics LCD Controller in YUV format.
	PLIB_GLCD_DESignalLevelGet	Gets the display enable signal level.
	PLIB_GLCD_DitheringIsEnabled	Verifies whether Dithering is enabled.
	PLIB_GLCD_ForceOutputBlankIsEnabled	Verify whether the Force Output Blank feature is enabled.
	PLIB_GLCD_IsEnabled	Verifies whether the Graphics LCD Controller is Enabled.
	PLIB_GLCD_IsLastRow	Gets the status indicating whether a last row is currently displayed by the Graphics Display Controller.
	PLIB_GLCD_IsVerticalBlankingActive	Get the active status.
	PLIB_GLCD_RGBSequentialModeGet	Get the RGB Sequential Mode already set.
	PLIB_GLCD_YUVOutputIsEnabled	Verify whether the YUV Output feature is enabled.

i) Feature Existence Functions

	Name	Description
	PLIB_GLCD_ExistsBackgroundColor	Verify whether the Background Color Feature is supported.
	PLIB_GLCD_ExistsBackPorchXY	Verify whether Back Porch Feature is supported.
	PLIB_GLCD_ExistsBlankingXY	Verify whether Blanking Period Feature is supported.
	PLIB_GLCD_ExistsClockDivider	Verify whether the Clock Divider feature is supported.
	PLIB_GLCD_ExistsCursor	Verifies whether the cursor feature exists.
	PLIB_GLCD_ExistsCursorData	
	PLIB_GLCD_ExistsCursorLUT	Verify whether Cursor LUT feature is supported.

	PLIB_GLCD_ExistsCursorXY	Verify whether Cursor XY Position feature is supported.
	PLIB_GLCD_ExistsDESignalLevel	Verify whether DE Signal Level feature is supported.
	PLIB_GLCD_ExistsDithering	Verifies whether the Dithering feature exists.
	PLIB_GLCD_ExistsEnable	Identifies whether the GLCD Enable feature exists on the GLCD module.
	PLIB_GLCD_ExistsForceOutputBlank	Verify whether Force Output Blank feature is supported.
	PLIB_GLCD_ExistsFormattingClockDivide	Verify whether Clock Formatting feature is available.
	PLIB_GLCD_ExistsFrontPorchXY	Verify whether Front Porch Feature is supported.
	PLIB_GLCD_ExistsGlobalColorLUT	Verify whether Global Color LUT feature is supported.
	PLIB_GLCD_ExistsHSyncInterruptEnable	Verify whether the HSYNC Interrupt Enable feature is supported.
	PLIB_GLCD_ExistsHSyncSignalLevel	Verify whether HSYNC Signal Level feature is supported.
	PLIB_GLCD_ExistsIRQTriggerControl	Verify whether the IRQ Trigger Control feature is supported.
	PLIB_GLCD_ExistsIsLastRow	Verify whether Is Last Row Feature is supported.
	PLIB_GLCD_ExistsIsVerticalBlankingActive	Verify whether Is Vertical Blanking Active feature is supported.
	PLIB_GLCD_ExistsLayerBaseAddress	Verify whether the Layer Base Address feature is supported.
	PLIB_GLCD_ExistsLayerBilinearFilterEnable	Enable Layer Bilinear Filter Feature.
	PLIB_GLCD_ExistsLayerColorMode	Verify whether Layer Color Mode is supported.
	PLIB_GLCD_ExistsLayerDestBlendFunc	Verify whether Layer Destination Blend Function Feature is supported.
	PLIB_GLCD_ExistsLayerEnable	Verify whether Layer Enable Feature is supported.
	PLIB_GLCD_ExistsLayerForceWithGlobalAlpha	Verify whether Layer Force with Global Alpha Feature is supported.
	PLIB_GLCD_ExistsLayerGlobalAlpha	Verify whether Layer Global Alpha Feature is supported.
	PLIB_GLCD_ExistsLayerPreMultiplyImageAlpha	Verify whether Layer Pre-Multiply Image Alpha Feature is supported.
	PLIB_GLCD_ExistsLayerResXY	
	PLIB_GLCD_ExistsLayerSizeXY	Verify whether Layer X Axis and Y Axis Size feature is supported.
	PLIB_GLCD_ExistsLayerSrcBlendFunc	Verify whether Layer Source Blend Function Feature is supported.
	PLIB_GLCD_ExistsLayerStartXY	Verify whether Layer Start XY Position feature is supported.
	PLIB_GLCD_ExistsLayerStride	Verify whether the Layer Stride Feature is supported.
	PLIB_GLCD_ExistsLinesPrefetch	Verify whether Lines Prefetch Set Feature supported.
	PLIB_GLCD_ExistsPaletteGammaRamp	Verifies whether the palette / gamma ramp feature is supported.
	PLIB_GLCD_ExistsResolutionXY	Verify whether YUV Output feature is supported.
	PLIB_GLCD_ExistsRGBSequentialMode	Verify whether RGB Sequential Mode feature is supported.
	PLIB_GLCD_ExistsSignalPolarity	Verifies whether the Signal Polarity Selection feature exists.
	PLIB_GLCD_ExistsSingleCyclePerLineVsync	Verifies whether VSYNC on Single cycle Per Line Feature exists.
	PLIB_GLCD_ExistsVSyncInterruptEnable	Verify whether VSYNC Interrupt Enable feature is supported.
	PLIB_GLCD_ExistsVSyncSignalLevel	Verify whether VSYNC Signal Level feature is supported.
	PLIB_GLCD_ExistsYUVOutput	Verify whether YUV output feature is supported.

j) Data Types and Constants

	Name	Description
	GLCD_IRQ_TRIGGER_CONTROL	Possible values of GLCD Interrupt Trigger Mode.
	GLCD_LAYER_COLOR_MODE	Possible values of GLCD color Mode.
	GLCD_LAYER_DEST_BLEND_FUNC	Possible values of GLCD Layer Destination Blend Functions.
	GLCD_LAYER_ID	Possible values of GLCD Layer Ids
	GLCD_LAYER_SRC_BLEND_FUNC	Possible values of GLCD Layer Source Blend Functions.
	GLCD_MODULE_ID	Possible instances of the GLCD module
	GLCD_RGB_MODE	GLCD RGB Sequential Mode
	GLCD_SIGNAL_POLARITY	Polarity values of different output signals.

Description

This section describes the Application Programming Interface (API) functions of the GLCD Controller Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_GLCD_Disable Function

Disables the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_Disable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_Disable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_Disable(GLCD_MODULE_ID index)

PLIB_GLCD_Enable Function

Enables the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_Enable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_Enable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_Enable(GLCD_MODULE_ID index)

PLIB_GLCD_RGBSequentialModeSet Function

Sets the RGB output sequential mode.

File

plib_glcd.h

C

```
void PLIB_GLCD_RGBSequentialModeSet(GLCD_MODULE_ID index, GLCD_RGB_MODE mode);
```

Returns

None.

Description

This function sets the RGB output sequential mode.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_RGBSequentialModeSet( GLCD_MODULE_ID_0,
                                GLCD_RGB_MODE_PARALLEL );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
mode	RGB sequential mode defined by GLCD_RGB_MODE.

Function

void PLIB_GLCD_RGBSequentialModeSet(GLCD_MODULE_ID index, GLCD_RGB_MODE mode)

b) Display Signal Polarity and Timing Management Functions

PLIB_GLCD_SignalPolarityGet Function

Gets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.

File

plib_glcd.h

C

```
GLCD_SIGNAL_POLARITY PLIB_GLCD_SignalPolarityGet(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function gets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t signalPolarity = GLCD_POLARITY_POSITIVE;

signalPolarity = PLIB_GLCD_VSyncPolarityGet( GLCD_MODULE_ID_0 );

if( signalPolarity & GLCD_DE_POLARITY_NEGATIVE )
{
    // DE Polarity Negative
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

GLCD_SIGNAL_POLARITY PLIB_GLCD_SignalPolarityGet(GLCD_MODULE_ID index)

PLIB_GLCD_SignalPolaritySet Function

Sets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_SignalPolaritySet(GLCD_MODULE_ID index, GLCD_SIGNAL_POLARITY polarity);
```

Returns

None.

Description

This function sets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_SignalPolaritySet( GLCD_MODULE_ID_0,
                             GLCD_POLARITY_POSITIVE |
                             GLCD_DE_POLARITY_NEGATIVE );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
polarity	Enum variable specifying polarity.

Function

void PLIB_GLCD_SignalPolaritySet(GLCD_MODULE_ID index,
GLCD_SIGNAL_POLARITY polarity)

PLIB_GLCD_BackPorchXGet Function

Gets X Axis Back Porch.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_BackPorchXGet(GLCD_MODULE_ID index);
```

Returns

- value of X Axis Back Porch.

Description

This function gets X Axis Back Porch.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t backPorchX;

backPorchX = PLIB_GLCD_BackPorchXGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_BackPorchXGet(GLCD_MODULE_ID index)

PLIB_GLCD_BackPorchXYSet Function

Sets the back porch on the x and y axis for the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_BackPorchXYSet(GLCD_MODULE_ID index, uint32_t backPorchX, uint32_t backPorchY);
```

Returns

None.

Description

This function sets the back porch on the x and y axis for the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// set back porch as: on x axis: 40 and on y axis: 20
PLIB_GLCD_BackPorchXYSet( GLCD_MODULE_ID_0, 40, 20 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
backPorchX	Back porch pulse width on x axis.
backPorchY	Back porch pulse width on y axis.

Function

void PLIB_GLCD_BackPorchXYSet(GLCD_MODULE_ID index,
uint32_t backPorchX
uint32_t backPorchY)

PLIB_GLCD_BackPorchYGet Function

Gets Y Axis Back Porch.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_BackPorchYGet(GLCD_MODULE_ID index);
```

Returns

- value of Y Axis Back Porch.

Description

This function gets Y Axis Back Porch.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t backPorchY;

backPorchY = PLIB_GLCD_BackPorchYGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_BackPorchYGet(GLCD_MODULE_ID index)

PLIB_GLCD_BlankingXYSet Function

Sets the blanking period on the x and y axis of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_BlankingXYSet(GLCD_MODULE_ID index, uint32_t blankingX, uint32_t blankingY);
```

Returns

None.

Description

This function sets the blanking period on the x and y axis of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// Set blanking period: on x axis: 10 and on y axis: 10
PLIB_GLCD_BlankingXY_Set( GLCD_MODULE_ID_0, 10, 10 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
blankingX	Blanking period on x axis.
blankingY	Blanking period on y axis.

Function

void PLIB_GLCD_BlankingXYSet(GLCD_MODULE_ID index,
uint32_t blankingX
uint32_t blankingY)

PLIB_GLCD_FormattingClockDivideDisable Function

Disbale the Clock Formatting feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_FormattingClockDivideDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disbales the Clock Formatting feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_FormattingClockDivideDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_FormattingClockDivideDisable(GLCD_MODULE_ID index)

PLIB_GLCD_FormattingClockDivideEnable Function

Enable the Clock Formatting feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_FormattingClockDivideEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function Enables the Clock Formatting feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_FormattingClockDivideEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_FormattingClockDivideEnable(GLCD_MODULE_ID index)

PLIB_GLCD_FormattingClockDivideIsEnabled Function

Verify whether Clock Formatting feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_FormattingClockDivideIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The Clock Formatting feature is enabled.
- false - The Clock Formatting feature is disabled.

Description

This function verifies if the Clock Formatting feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_FormattingClockDivideIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_FormattingClockDivideDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_FormattingClockDivideIsEnabled(GLCD_MODULE_ID index)

PLIB_GLCD_FrontPorchXGet Function

Gets X Axis Front Porch.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_FrontPorchXGet(GLCD_MODULE_ID index);
```

Returns

- value of X Axis Front Porch.

Description

This Function gets X Axis Front Porch.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t porchX;

porchX = PLIB_GLCD_FrontPorchXGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_FrontPorchXGet(GLCD_MODULE_ID index)

PLIB_GLCD_FrontPorchXYSet Function

Sets the front porch on the x and y axis for the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_FrontPorchXYSet(GLCD_MODULE_ID index, uint32_t frontPorchX, uint32_t frontPorchY);
```

Returns

None.

Description

This function sets the front porch on the x and y axis for the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// set front porch as: on x axis: 40 and on y axis: 20
PLIB_GLCD_FrontPorchXYSet( GLCD_MODULE_ID_0, 40, 20 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
frontPorchX	Front porch pulse width on x axis.
frontPorchY	Front porch pulse width on y axis.

Function

void PLIB_GLCD_FrontPorchXYSet(GLCD_MODULE_ID index,

uint32_t frontPorchX,
uint32_t frontPorchY)

PLIB_GLCD_FrontPorchYGet Function

Gets Y Axis Front Porch.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_FrontPorchYGet(GLCD_MODULE_ID index);
```

Returns

- value of y axis front porch.

Description

This function gets Y Axis Front Porch.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t porchY;

porchY = PLIB_GLCD_FrontPorchYGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_FrontPorchYGet(GLCD_MODULE_ID index)

PLIB_GLCD_LinesPrefetchGet Function

Gets value of lines to be prefetched.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LinesPrefetchGet(GLCD_MODULE_ID index);
```

Returns

- value of lines to be prefetched.

Description

This function gets the value of lines to be prefetched.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t linesPrefetch;
```

```
linesPrefetch = PLIB_GLCD_LinesPrefetchGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_LinesPrefetchGet(GLCD_MODULE_ID index)

PLIB_GLCD_ResolutionXGet Function

Gets X Axis Resolution.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_ResolutionXGet(GLCD_MODULE_ID index);
```

Returns

- value of x axis resolution.

Description

This function gets the X Axis resolution.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t resolutionX;

resolutionX = PLIB_GLCD_ResolutionXGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_ResolutionXGet(GLCD_MODULE_ID index)

PLIB_GLCD_ResolutionYGet Function

Gets Y Axis Resolution.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_ResolutionYGet(GLCD_MODULE_ID index);
```

Returns

- Value of Y Axis Resolution.

Description

This function gets the Y Axis Resolution.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t resolutionY;

resolutionY = PLIB_GLCD_ResolutionYGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_ResolutionYGet(GLCD_MODULE_ID index)

PLIB_GLCD_BlankingXGet Function

Gets the X Axis Blanking Period.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_BlankingXGet(GLCD_MODULE_ID index);
```

Returns

- value of X Axis Blanking Period.

Description

This function gets the X Axis Blanking Period.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t blankingX;

blankingX = PLIB_GLCD_BlankingXGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_BlankingXGet(GLCD_MODULE_ID index)

PLIB_GLCD_BlankingYGet Function

Gets the Y Axis Blanking Period.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_BlankingYGet(GLCD_MODULE_ID index);
```

Returns

- value of Y Axis Blanking Period.

Description

This function gets the Y Axis Blanking Period.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t blankingY;

blankingY = PLIB_GLCD_BlankingYGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_BlankingYGet(GLCD_MODULE_ID index)

PLIB_GLCD_LinesPrefetchSet Function

Sets the clock controls of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LinesPrefetchSet(GLCD_MODULE_ID index, uint32_t linesPrefetch);
```

Returns

None.

Description

This function sets the clock controls of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LinesPrefetchSet( GLCD_MODULE_ID_0,
                            MY_LINES_PREFETCH_VALUE );
```

Parameters

Parameters	Description
index	Identifier of the device instance
linesPrefetch	clock lines prefetch

Function

void PLIB_GLCD_LinesPrefetchSet(GLCD_MODULE_ID index, uint32_t linesPrefetch)

PLIB_GLCD_ClockDividerSet Function

Sets clock controls of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_ClockDividerSet(GLCD_MODULE_ID index, uint32_t clockDivider);
```

Returns

None.

Description

This function sets clock controls of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_ClockDividerSet( GLCD_MODULE_ID_0, MY_CLOCK_DIVIDER_VALUE );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
clockDivider	Factor dividing the GLCD clock.

Function

void PLIB_GLCD_ClockDividerSet(GLCD_MODULE_ID index, uint32_t clockDivider)

PLIB_GLCD_ClockDividerGet Function

Gets Clock Divider value.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_ClockDividerGet(GLCD_MODULE_ID index);
```

Returns

- value of clock divider.

Description

This function gets the Clock Divider value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t dividerValue;

dividerValue = PLIB_GLCD_ClockDividerGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_ClockDividerGet(GLCD_MODULE_ID index)

PLIB_GLCD_ResolutionXYSet Function

Sets the display resolution for the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_ResolutionXYSet(GLCD_MODULE_ID index, uint32_t resolutionX, uint32_t resolutionY);
```

Returns

None.

Description

This function sets the display resolution for the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// set display resolution as: width: 640 pixels and height: 480 pixels
PLIB_GLCD_ResolutionXYSet( GLCD_MODULE_ID_0, 640, 480 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
resolutionX	Display resolution on x axis in terms of number of pixels.
resolutionY	Display resolution on y axis in terms of number of pixels.

Function

void PLIB_GLCD_ResolutionXYSet(GLCD_MODULE_ID index,
uint32_t resolutionX,
uint32_t resolutionY)

c) Background Management Functions

PLIB_GLCD_BackgroundColorGet Function

Gets the value of the Background Color.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_BackgroundColorGet(GLCD_MODULE_ID index);
```

Returns

- value of the Background Color.

Description

This function gets the value of the Background Color.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t backgroundColor;

backgroundColor = PLIB_GLCD_BackgroundColorGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_BackgroundColorGet(GLCD_MODULE_ID index);

PLIB_GLCD_BackgroundColorSet Function

Sets the background color.

File

plib_glcd.h

C

```
void PLIB_GLCD_BackgroundColorSet(GLCD_MODULE_ID index, uint32_t bgColor);
```

Returns

None.

Description

This function sets the background color of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_BGColorSet( GLCD_MODULE_ID_0, 0x000000FF );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
bgColor	Background color.

Function

void ExistsBackgroundColor(GLCD_MODULE_ID index)

d) Layer Management Functions

PLIB_GLCD_LayerBaseAddressGet Function

Gets the Layer Base Address.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerBaseAddressGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- Layer base address value.

Description

This function sets the Layer Base Address.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t layerBaseAddress;

layerBaseAddress = PLIB_GLCD_LayerBaseAddressGet( GLCD_MODULE_ID_0
                                                  GLCD_LAYER_0  );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

uint32_t PLIB_GLCD_LayerBaseAddressGet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerBaseAddressSet Function

Sets the base address of layer surface of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerBaseAddressSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, uint32_t baseAddress);
```

Returns

None.

Description

This function sets the layer base address of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// Set layer 0 surface base address as 0x20000000
PLIB_GLCD_LayerBaseAddressSet( GLCD_MODULE_ID_0, GLCD_LAYER_0, 0x20000000 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
baseAddress	baseAddress of layer surface.

Function

void PLIB_GLCD_LayerBaseAddressSet(GLCD_MODULE_ID index,

GLCD_LAYER_ID layerId,
uint32_t baseAddress)

PLIB_GLCD_LayerBilinearFilterDisable Function

Disables the layer Bilinear filter of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerBilinearFilterDisable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function disables the layer Bilinear filter of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerBilinearFilterDisable( GLCD_MODULE_ID_0, GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerBilinearFilterDisable (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerBilinearFilterEnable Function

Enables the layer Bilinear filter of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerBilinearFilterEnable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function enables the layer Bilinear filter of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerBilinearFilterEnable( GLCD_MODULE_ID_0, GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerBilinearFilterEnable (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerBilinearFilterIsEnabled Function

Verify whether Layer Bilinear Filter is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_LayerBilinearFilterIsEnabled(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- true - The Layer Bilinear feature is enabled.
- false - The Layer Bilinear feature is disabled.

Description

This function verifies whether Layer Bilinear Filter is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_LayerBilinearFilterIsEnabled( GLCD_MODULE_ID_0,
                                             GLCD_LAYER_0 ) )
{
    PLIB_GLCD_LayerBilinearFilterDisable( GLCD_MODULE_ID_0,
                                          GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

bool PLIB_GLCD_LayerBilinearFilterIsEnabled (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerColorModeGet Function

Gets the Layer Color Mode.

File

plib_glcd.h

C

```
GLCD_LAYER_COLOR_MODE PLIB_GLCD_LayerColorModeGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- GLCD_LAYER_COLOR_MODE_LUT8 - Layer Color Mode set to LUT8
- GLCD_LAYER_COLOR_MODE_RGBA5551 - Layer Color Mode set to RGBA5551
- GLCD_LAYER_COLOR_MODE_RGBA8888 - Layer Color Mode set to RGBA8888
- GLCD_LAYER_COLOR_MODE_RGB332 - Layer Color Mode set to RGB 332
- GLCD_LAYER_COLOR_MODE_RGB565 - Layer Color Mode set to RGB565
- GLCD_LAYER_COLOR_MODE_ARGB8888 - Layer Color Mode set to ARGB8888
- GLCD_LAYER_COLOR_MODE_L8 - Layer Color Mode set to L8
- GLCD_LAYER_COLOR_MODE_L1 - Layer Color Mode set to L1
- GLCD_LAYER_COLOR_MODE_L4 - Layer Color Mode set to L4
- GLCD_LAYER_COLOR_MODE_YUYV - Layer Color Mode set to YUYV
- GLCD_LAYER_COLOR_MODE_RGB888 - Layer Color Mode set to RGB888

Description

This function gets the Layer Color Mode

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t colorMode;

colorMode = PLIB_GLCD_LayerColorModeGet( GLCD_MODULE_ID_0
                                         GLCD_LAYER_0   );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

```
GLCD_LAYER_COLOR_MODE PLIB_GLCD_LayerColorModeGet( GLCD_MODULE_ID index,
                                                   GLCD_LAYER_ID  layerId );
```

PLIB_GLCD_LayerColorModeSet Function

Sets the layer color mode of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerColorModeSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, GLCD_LAYER_COLOR_MODE
colorMode);
```

Returns

None.

Description

This function sets the layer color mode of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// Set layer color mode to RGB232
PLIB_GLCD_LayerColorModeSet( GLCD_MODULE_ID_0, GLCD_LAYER_0,
                             GLCD_LAYER_COLOR_MODE_RGB232 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
colorMode	Color mode defined by GLCD_LAYER_COLOR_MODE.

Function

```
void PLIB_GLCD_LayerColorModeSet(  GLCD_MODULE_ID       index,
                                   GLCD_LAYER_ID        layerId,
                                   GLCD_LAYER_COLOR_MODE colorMode )
```

PLIB_GLCD_LayerDestBlendFuncGet Function

Gets the Layer Destination Blend Function.

File

plib_glcd.h

C

```
GLCD_LAYER_DEST_BLEND_FUNC PLIB_GLCD_LayerDestBlendFuncGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- GLCD_LAYER_DEST_BLEND_BLACK - Destination Blend function set to Black = 0.
- GLCD_LAYER_DEST_BLEND_WHITE - Destination Blend function set to White = 255.
- GLCD_LAYER_DEST_BLEND_ALPHA_SRC - Destination Blend function set to Source Alpha.
- GLCD_LAYER_DEST_BLEND_ALPHA_GBL - Destination Blend function set to Global Alpha.
- GLCD_LAYER_DEST_BLEND_ALPHA_SRCGBL - Destination Blend function set to (Source * Global Alpha)
- GLCD_LAYER_DEST_BLEND_INV_SRC - Destination Blend function set to (1 - Source Alpha)
- GLCD_LAYER_DEST_BLEND_INV_GBL - Destination Blend function set to (1 - Global Alpha)
- GLCD_LAYER_DEST_BLEND_INV_SRCGBL - Destination Blend function set to (1 - (Source Alpha * Global Alpha)).
- GLCD_LAYER_DEST_BLEND_ALPHA_DST - Destination Blend function set to Destination Alpha.
- GLCD_LAYER_DEST_BLEND_INV_DST - Destination Blend function set to (1 - Destination Alpha)

Description

This function gets the Layer Destination Blend Function.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t destBlendFunc;

destBlendFunc = PLIB_GLCD_LayerDestBlendFuncGet( GLCD_MODULE_ID_0
                                                 GLCD_LAYER_0    );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

```
GLCD_LAYER_DEST_BLEND_FUNC PLIB_GLCD_LayerDestBlendFuncGet( GLCD_MODULE_ID index,
```

GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerDestBlendFuncSet Function

Sets the layer destination blend function of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerDestBlendFuncSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId,
GLCD_LAYER_DEST_BLEND_FUNC blendFunc);
```

Returns

None.

Description

This function sets the layer destination blend function of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerDestBlendFuncSet( GLCD_MODULE_ID_0, GLCD_LAYER_0,
                                 GLCD_BLEND_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
blendFunc	Blend function defined by GLCD_LAYER_BLEND_FUNCTION.

Function

void PLIB_GLCD_LayerDestBlendFuncSet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId,
GLCD_LAYER_DEST_BLEND_FUNC blendFunc)

PLIB_GLCD_LayerDisable Function

Disables the layer of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerDisable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function disables the layer of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerDisable( GLCD_MODULE_ID_0, GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerDisable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerEnable Function

Enables the layer of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerEnable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function enables the layer of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerEnable( GLCD_MODULE_ID_0, GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerEnable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerForceWithGlobalAlphaDisable Function

Disables the Layer Force Global Alpha feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerForceWithGlobalAlphaDisable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function disables the Layer Force Global Alpha feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerForceWithGlobalAlphaDisable ( GLCD_MODULE_ID_0,
                                             GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerForceWithGlobalAlphaDisable (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerForceWithGlobalAlphaEnable Function

Enable the Layer Force Global Alpha feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerForceWithGlobalAlphaEnable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function enables the Layer Force Global Alpha feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerForceWithGlobalAlphaEnable ( GLCD_MODULE_ID_0,
                                            GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerForceWithGlobalAlphaEnable (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerForceWithGlobalAlphaIsEnabled Function

Verify whether Layer Force with Global Alpha Feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_LayerForceWithGlobalAlphaIsEnabled(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- true - The Layer Force Global Alpha feature is enabled.
- false - The Layer Force Global Alpha feature is disabled.

Description

This function verifies whether Layer Force with Global Alpha Feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_LayerForceWithGlobalAlphaIsEnabled( GLCD_MODULE_ID_0,
                                                   GLCD_LAYER_0 ) )
{
    PLIB_GLCD_LayerForceWithGlobalAlphaDisable( GLCD_MODULE_ID_0,
                                                GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

bool PLIB_GLCD_LayerForceWithGlobalAlphaIsEnabled (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerGlobalAlphaGet Function

Gets the Layer Global Alpha value.

File

plib_glcd.h

C

```
uint8_t PLIB_GLCD_LayerGlobalAlphaGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- Layer Alpha value

Description

This function gets the Layer Global Alpha value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint8_t layerAlpha;

layerAlpha = PLIB_GLCD_LayerGlobalAlphaGet( GLCD_MODULE_ID_0
                                            GLCD_LAYER_0    );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

uint8_t PLIB_GLCD_LayerGlobalAlphaGet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerGlobalAlphaSet Function

Sets the layer global alpha of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerGlobalAlphaSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, uint8_t value);
```

Returns

None.

Description

This function sets the layer global alpha of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// set global alpha value of layer 0 to 255
PLIB_GLCD_Layer_GlobalAlpha_Set( GLCD_MODULE_ID_0, GLCD_LAYER_0, 255 )
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
value	Global alpha value.

Function

void PLIB_GLCD_LayerGlobalAlphaSet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId,
uint8_t value)

PLIB_GLCD_LayerIsEnabled Function

Verify whether Layer is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_LayerIsEnabled(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- true - The given Layer is enabled.
- false - The given Layer is disabled.

Description

This function verifies whether a given Layer is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_LayerIsEnabled( GLCD_MODULE_ID_0, GLCD_LAYER_0 ) )
{
    PLIB_GLCD_LayerDisable( GLCD_MODULE_ID_0, GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_LayerIsEnabled (GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerPreMultiplyImageAlphaDisable Function

Disable Layer Pre-Multiply Image Alpha Feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerPreMultiplyImageAlphaDisable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function disables Layer Pre-Multiply Image Alpha Feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerPreMultiplyImageAlphaDisable( GLCD_MODULE_ID_0,
                                             GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerPreMultiplyImageAlphaDisable (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerPreMultiplyImageAlphaEnable Function

Enable Layer Pre-Multiply Image Alpha Feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerPreMultiplyImageAlphaEnable(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

None.

Description

This function enables the Layer Pre-Multiply Image Alpha Feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerPreMultiplyImageAlphaEnable( GLCD_MODULE_ID_0,
                                            GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

void PLIB_GLCD_LayerPreMultiplyImageAlphaEnable (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerPreMultiplyImageAlphaIsEnabled Function

Verify whether the Layer Pre-Multiply Image Alpha Feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_LayerPreMultiplyImageAlphaIsEnabled(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- true - The Layer Pre-Multiply Image Alpha feature is enabled
- false - The Layer Pre-Multiply Image Alpha feature is disabled.

Description

This function verifies whether the Layer Pre-Multiply Image Alpha Feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_LayerPreMultiplyImageAlphaIsEnabled( GLCD_MODULE_ID_0,
                                                    GLCD_LAYER_0 ) )
{
    PLIB_GLCD_LayerPreMultiplyImageAlphaDisable( GLCD_MODULE_ID_0,
                                                 GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

bool PLIB_GLCD_LayerPreMultiplyImageAlphaIsEnabled (GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerResXGet Function

Gets the Layer X Axis Resolution.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerResXGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- value of layer x axis resolution.

Description

This function gets the Layer X Axis Resolution.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t resolutionX;

resolutionX = PLIB_GLCD_LayerResXGet( GLCD_MODULE_ID_0
                                      GLCD_LAYER_0  );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

uint32_t PLIB_GLCD_LayerResXGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerResXYSet Function

Sets the layer resolution in pixels.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerResXYSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, uint32_t resolutionX, uint32_t
resolutionY);
```

Returns

None.

Description

This function sets the layer resolution in pixels. The resolution is defined as the number of pixels on the x axis and the number of pixels on the y

axis.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// Layer resolution set to x = 320 pixels and y = 240 pixels
PLIB_GLCD_Layer_ResXY_Set( GLCD_MODULE_ID_0, GLCD_LAYER_0, 320, 240);
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
resolutionX	resolution on x axis in terms of pixels.
resolutionY	resolution on y axis in terms of pixels.

Function

void PLIB_GLCD_LayerResXYSet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId,
uint32_t resolutionX,
uint32_t resolutionY)

PLIB_GLCD_LayerResYGet Function

Gets the Layer Y Axis Resolution.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerResYGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- value of layer y axis resolution.

Description

This function gets the Layer Y Axis Resolution.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t resolutionY;

resolutionY = PLIB_GLCD_LayerResYGet( GLCD_MODULE_ID_0
                                      GLCD_LAYER_0  );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

uint32_t PLIB_GLCD_LayerResYGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerSizeXGet Function

Gets the Layer X Axis Size.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerSizeXGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- Layer Size X value.

Description

This function gets the Layer X Axis Size.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t sizeX;

startX  = PLIB_GLCD_LayerSizeXGet( GLCD_MODULE_ID_0
                                   GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

uint32_t PLIB_GLCD_LayerSizeXGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerSizeXYSet Function

Sets the layer size x and size y of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerSizeXYSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, uint32_t sizeX, uint32_t sizeY);
```

Returns

None.

Description

This function sets the layer start x and start y of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// set layer size to (640, 480)
PLIB_GLCD_LayerSizeXYSet( GLCD_MODULE_ID_0, GLCD_LAYER_0, 640, 480);
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
sizeX	Layer size on x axis in terms of pixels.
sizeY	Layer size on y axis in terms of pixels.

Function

```
void PLIB_GLCD_LayerSizeXYSet(  GLCD_MODULE_ID index,
                                GLCD_LAYER_ID  layerId,
uint32_t      sizeX,
uint32_t      sizeY )
```

PLIB_GLCD_LayerSizeYGet Function

Gets the Layer Y Axis Size.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerSizeYGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- Layer Size Y value.

Description

This function gets the Layer Y Axis Size.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t sizeY;

sizeY   = PLIB_GLCD_LayerSizeYGet( GLCD_MODULE_ID_0
                                   GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

```
uint32_t PLIB_GLCD_LayerSizeYGet(  GLCD_MODULE_ID index, GLCD_LAYER_ID  layerId )
```

PLIB_GLCD_LayerSrcBlendFuncGet Function

Gets the Layer Source Blend Function.

File

plib_glcd.h

C

```
GLCD_LAYER_SRC_BLEND_FUNC PLIB_GLCD_LayerSrcBlendFuncGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- GLCD_LAYER_SRC_BLEND_BLACK - Source Blend Function set to Black = 0.
- GLCD_LAYER_SRC_BLEND_WHITE - Source Blend Function set to White = 255.
- GLCD_LAYER_SRC_BLEND_ALPHA_SRC - Source Blend Function set to Source Alpha.
- GLCD_LAYER_SRC_BLEND_ALPHA_GBL - Source Blend Function set to Global Alpha.
- GLCD_LAYER_SRC_BLEND_ALPHA_SRCGBL - Source Blend Function set to (Source Alpha * Global Alpha).
- GLCD_LAYER_SRC_BLEND_INV_SRC - Source Blend Function set to (1 - Source Alpha).
- GLCD_LAYER_SRC_BLEND_INV_GBL - Source Blend Function set to (1 - Global Alpha).
- GLCD_LAYER_SRC_BLEND_INV_SRCGBL - Source Blend Function set to (1 - (Source Alpha * Global Alpha))
- GLCD_LAYER_SRC_BLEND_ALPHA_DST - Source Blend Function set to Destination Alpha
- GLCD_LAYER_SRC_BLEND_INV_DST - Source Blend Function set to (1 - Destination Alpha)

Description

This function gets the Layer Source Blend Function.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t srcBlendFunc;

srcBlendFunc = PLIB_GLCD_LayerSrcBlendFuncGet( GLCD_MODULE_ID_0
                                               GLCD_LAYER_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

GLCD_LAYER_SRC_BLEND_FUNC PLIB_GLCD_LayerSrcBlendFuncGet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerSrcBlendFuncSet Function

Sets the layer source blend function of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerSrcBlendFuncSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, GLCD_LAYER_SRC_BLEND_FUNC
blendFunc);
```

Returns

None.

Description

This function sets the layer source blend function of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_LayerSrcBlendFuncSet( GLCD_MODULE_ID_0, GLCD_LAYER_0,
```

```
                                    GLCD_BLEND_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
blendFunc	Blend function defined by GLCD_LAYER_BLEND_FUNCTION.

Function

```
void PLIB_GLCD_LayerSrcBlendFuncSet(  GLCD_MODULE_ID          index,
                                      GLCD_LAYER_ID           layerId,
                                      GLCD_LAYER_SRC_BLEND_FUNC blendFunc )
```

PLIB_GLCD_LayerStartXGet Function

Gets the Layer X Axis Start Position.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerStartXGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- Layer Start X value.

Description

This function gets the Layer X Axis Start Position.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t startX;

startX  = PLIB_GLCD_LayerStartXGet( GLCD_MODULE_ID_0
                                    GLCD_LAYER_0    );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

```
uint32_t PLIB_GLCD_LayerStartXGet(  GLCD_MODULE_ID index,
                                    GLCD_LAYER_ID  layerId );
```

PLIB_GLCD_LayerStartXYSet Function

Sets the layer start x and start y of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerStartXYSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, uint32_t startX, uint32_t
startY);
```

Returns

None.

Description

This function sets the layer start x and start y of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None

Example

```
// set layer start to (20, 20)
PLIB_GLCD_LayerStartXYSet( GLCD_MODULE_ID_0, GLCD_LAYER_0, 20, 20);
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
startX	Layer start on x axis in terms of pixels.
startY	Layer start on y axis in terms of pixels.

Function

```
void PLIB_GLCD_LayerStartXYSet(    GLCD_MODULE_ID index,
                                   GLCD_LAYER_ID  layerId,
uint32_t      startX,
uint32_t      startY )
```

PLIB_GLCD_LayerStartYGet Function

Gets the Layer Y Axis Start Position.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerStartYGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- Layer Start Y value.

Description

This function gets the Layer Y Axis Start Position.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t startY;

startY  = PLIB_GLCD_LayerStartYGet( GLCD_MODULE_ID_0
                                    GLCD_LAYER_0    );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

layerId	Identifier of the Graphics rendering layer.

Function

uint32_t PLIB_GLCD_LayerStartYGet(GLCD_MODULE_ID index,
GLCD_LAYER_ID layerId);

PLIB_GLCD_LayerStrideGet Function

Gets the Layer Stride value.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_LayerStrideGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId);
```

Returns

- value of the stride.

Description

This function gets the Layer Stride value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t layerStride;

layerStride = PLIB_GLCD_LayerStrideGet( GLCD_MODULE_ID_0
                                        GLCD_LAYER_0    );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.

Function

uint32_t PLIB_GLCD_LayerStrideGet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId)

PLIB_GLCD_LayerStrideSet Function

Sets the layer surface stride of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_LayerStrideSet(GLCD_MODULE_ID index, GLCD_LAYER_ID layerId, uint32_t stride);
```

Returns

None.

Description

This function sets the layer surface stride of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t imageStride = IMAGE_WIDTH * IMAGE_BYTES_PER_PIXEL;

// To make stride 4 byte aligned
imageStride += ( imageStride % 4 );

PLIB_GLCD_LayerStrideSet( GLCD_MODULE_ID_0, GLCD_LAYER_0,
                          imageStride );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
layerId	Identifier of the Graphics rendering layer.
stride	line width in bytes including padding.

Function

```
void PLIB_GLCD_LayerStrideSet(  GLCD_MODULE_ID index,
                        GLCD_LAYER_ID  layerId,
uint32_t      stride )
```

e) Hardware Cursor Control and Management Functions

PLIB_GLCD_CursorDataGet Function

Gets the Cursor Data at given Index.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_CursorDataGet(GLCD_MODULE_ID index, uint32_t dataIndex);
```

Returns

- value of the 8 pixels of the cursor at given index.

Description

This function gets the Cursor Data at given Index.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t cursorData;

cursorData = PLIB_GLCD_CursorDataGet( GLCD_MODULE_ID_0,
                                      MY_CURSOR_DATA_INDEX );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
dataIndex	Index of the Cursor data.

Function

uint32_t PLIB_GLCD_CursorDataGet(GLCD_MODULE_ID index, uint32_t dataIndex)

PLIB_GLCD_CursorDataSet Function

Sets the cursor image data.

File

plib_glcd.h

C

```
void PLIB_GLCD_CursorDataSet(GLCD_MODULE_ID index, uint32_t * cursorData);
```

Returns

None.

Description

This function sets the cursor image data. The image data format is 4-bit black, white and 50 percent gray color.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t *cursorData = cursorDataAddress;

// Set cursor data in the format: 4-bit CLUT Index
PLIB_GLCD_CursorDataSet( GLCD_MODULE_ID_0, cursorData );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
cursorData	pointer to cursor data in the format 4-bit CLUT Index.

Function

void PLIB_GLCD_CursorDataSet(GLCD_MODULE_ID index, uint32_t * cursorData)

PLIB_GLCD_CursorDisable Function

Disables the cursor of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_CursorDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables the cursor of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ExistsCursor( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_CursorDisable(GLCD_MODULE_ID index)

PLIB_GLCD_CursorEnable Function

Enables the cursor of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_CursorEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables the cursor of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ExistsCursor( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_CursorEnable(GLCD_MODULE_ID index)

PLIB_GLCD_CursorIsEnabled Function

Verifies whether the cursor is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_CursorIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - Cursor is enabled.
- false - Cursor is disabled.

Description

This function verifies whether the cursor is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_CursorIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_CursorIsEnabled (GLCD_MODULE_ID index)

PLIB_GLCD_CursorLUTGet Function

Gets the color from Cursor LUT at given Index.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_CursorLUTGet(GLCD_MODULE_ID index, uint32_t lutIndex);
```

Returns

- value of color from color LUT at given index.

Description

This function gets the color from Cursor LUT at given Index.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t cursorLUTColor;

cursorLUTColor = PLIB_GLCD_CursorLUTGet( GLCD_MODULE_ID_0,
                                         MY_CURSOR_CLUT_INDEX );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
lutIndex	Cursor Color LUT index.

Function

uint32_t PLIB_GLCD_CursorLUTGet(GLCD_MODULE_ID index, uint32_t lutIndex)

PLIB_GLCD_CursorLUTSet Function

Sets the cursor color look-up table (LUT) in XRGB8888 format.

File

plib_glcd.h

C

```
void PLIB_GLCD_CursorLUTSet(GLCD_MODULE_ID index, uint32_t * cursorLUT);
```

Returns

None.

Description

This function sets the cursor color LUT of the Graphics LCD Controller in XRGB8888 format.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t * cursorLUTColors = cursorLUTAddress;

PLIB_GLCD_CursorLUTSet( GLCD_MODULE_ID_0, cursorLUTColors );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
cursorLUT	Pointer to the cursor color LUT in XRGB8888 format.

Function

void PLIB_GLCD_CursorLUTSet(GLCD_MODULE_ID index, uint32_t * cursorLUT)

PLIB_GLCD_CursorXGet Function

Gets the cursor X Axis Position.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_CursorXGet(GLCD_MODULE_ID index);
```

Returns

- Cursor X Axis Position.

Description

This function gets the cursor X Axis Position.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t cursorX;

cursorX = PLIB_GLCD_CursorXGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_CursorXGet(GLCD_MODULE_ID index)

PLIB_GLCD_CursorXYSet Function

Sets the x and y coordinates of the Graphics LCD Controller cursor.

File

plib_glcd.h

C

```
void PLIB_GLCD_CursorXYSet(GLCD_MODULE_ID index, uint32_t cursorX, uint32_t cursorY);
```

Returns

None.

Description

This function sets the x and y coordinates of the Graphics Display Controller cursor.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// update cursor position at x:20 pixels and y:20 pixels
PLIB_GLCD_CursorXYSet( GLCD_MODULE_ID_0, 20, 20 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
cursorX	cursor coordinate on x axis in terms of pixels.
cursorY	cursor coordinate on y axis in terms of pixels.

Function

void PLIB_GLCD_CursorXYSet(GLCD_MODULE_ID index,
uint32_t cursorX,
uint32_t cursorY)

PLIB_GLCD_CursorYGet Function

Gets the Cursor Y Axis Position.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_CursorYGet(GLCD_MODULE_ID index);
```

Returns

- Cursor Y Axis Position.

Description

This function gets the Cursor Y Axis Position.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t cursorY;

cursorY = PLIB_GLCD_CursorYGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

uint32_t PLIB_GLCD_CursorYGet(GLCD_MODULE_ID index)

f) Palette and Gamma Correction Control Functions

PLIB_GLCD_GlobalColorLUTGet Function

Gets the Color from Global Color LUT at given Index.

File

plib_glcd.h

C

```
uint32_t PLIB_GLCD_GlobalColorLUTGet(GLCD_MODULE_ID index, uint32_t lutIndex);
```

Returns

- value of Color from the Color Lookup Table.

Description

This function gets the Color from Global Color LUT at given Index.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t lutColor;

lutColor = PLIB_GLCD_GlobalColorLUTGet( GLCD_MODULE_ID_0,
                                        MY_LUT_COLOR_INDEX );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
lutIndex	Color Lookup table index.

Function

uint32_t PLIB_GLCD_GlobalColorLUTGet(GLCD_MODULE_ID index, uint32_t lutIndex)

PLIB_GLCD_GlobalColorLUTSet Function

Set Global Color LUT.

File

plib_glcd.h

C

```
void PLIB_GLCD_GlobalColorLUTSet(GLCD_MODULE_ID index, uint32_t * globalLUT);
```

Returns

None.

Description

This function sets the Global Color LUT.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t globalColorLUT[256] = { // Initial values
                               };
PLIB_GLCD_GlobalColorLUTSet( GLCD_MODULE_ID_0, globalColorLUT );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
globalLUT	pointer of the Global Color Lookup table data array.

Function

void PLIB_GLCD_GlobalColorLUTSet(GLCD_MODULE_ID index, uint32_t *globalLUT)

PLIB_GLCD_PaletteGammaRampDisable Function

Disables the palette / gamma ramp of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_PaletteGammaRampDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables the palette / gamma ramp of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_PaletteGammaRampDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_PaletteGammaRampDisable(GLCD_MODULE_ID index)

PLIB_GLCD_PaletteGammaRampEnable Function

Enables the palette / gamma ramp of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_PaletteGammaRampEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables the palette / gamma ramp of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_PaletteGammaRampEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_PaletteGammaRampEnable(GLCD_MODULE_ID index)

PLIB_GLCD_PaletteGammaRampIsEnabled Function

Verify whether Palette / Gamma Ramp feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_PaletteGammaRampIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The palette / gamma ramp feature is enabled.
- false - The palette / gamma ramp feature is disabled.

Description

This function verifies whether Palette / Gamma Ramp feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_PaletteGammaRampIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_PaletteGammaRampDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_PaletteGammaRampIsEnabled(GLCD_MODULE_ID index)

g) Interrupt Control Functions

PLIB_GLCD_HSyncInterruptDisable Function

Disables interrupts at Hsync.

File

plib_glcd.h

C

```
void PLIB_GLCD_HSyncInterruptDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables interrupts at Hsync.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_HsyncInterruptDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_HSyncInterruptDisable(GLCD_MODULE_ID index)

PLIB_GLCD_HSyncInterruptEnable Function

Enables interrupts at Hsync.

File

plib_glcd.h

C

```
void PLIB_GLCD_HSyncInterruptEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables interrupts at Hsync.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_HSyncInterruptEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_HSyncInterruptEnable(GLCD_MODULE_ID index)

PLIB_GLCD_HSyncInterruptIsEnabled Function

Verify whether HSYNC Interrupt is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_HSyncInterruptIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The HSYNC Interrupt feature is enabled.
- false - The HSYNC Interrupt feature is disabled.

Description

This function verifies whether HSYNC Interrupt is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_HSyncInterruptIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_HSyncInterruptDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_HSyncInterruptIsEnabled(GLCD_MODULE_ID index)

PLIB_GLCD_HSyncSignalLevelGet Function

Gets the Hsync signal level.

File

plib_glcd.h

C

```
bool PLIB_GLCD_HSyncSignalLevelGet(GLCD_MODULE_ID index);
```

Returns

- true - The Hsync signal level is high.

- false - The Hsync signal level is low.

Description

This function returns the Hsync signal level.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(!PLIB_GLCD_HSyncSignalLevelGet( GLCD_MODULE_ID_0 ));
{
    //Hsync signal level is low
    return;
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_HSyncSignalLevelGet(GLCD_MODULE_ID index)

PLIB_GLCD_IRQTriggerControlGet Function

Gets the IRQ Trigger Control value.

File

plib_glcd.h

C

```
GLCD_IRQ_TRIGGER_CONTROL PLIB_GLCD_IRQTriggerControlGet(GLCD_MODULE_ID index);
```

Returns

- GLCD_IRQ_TRIGGER_LEVEL - IRQ is Level Triggered.
- GLCD_IRQ_TRIGGER_EDGE - IRQ is edge Triggered.

Description

This function gets the IRQ Trigger Control value.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t irqTrigger;

irqTrigger = PLIB_GLCD_IRQTriggerControlGet( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

GLCD_IRQ_TRIGGER_CONTROL PLIB_GLCD_IRQTriggerControlGet(GLCD_MODULE_ID index)

PLIB_GLCD_IRQTriggerControlSet Function

Sets the IRQ trigger control.

File

plib_glcd.h

C

```
void PLIB_GLCD_IRQTriggerControlSet(GLCD_MODULE_ID index, GLCD_IRQ_TRIGGER_CONTROL irqControl);
```

Returns

None.

Description

This function sets the IRQ trigger control. IRQ trigger detection is defined at the edge or level of the trigger pulse.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
// Set trigger detection at edge
PLIB_GLCD_IRQTriggerControlSet( GLCD_MODULE_ID_0, GLCD_IRQ_TRIGGER_EDGE );
```

Parameters

Parameters	Description
index	Identifier of the device instance.
control	Trigger control defined by GLCD_IRQ_TRIGGER_CONTROL.

Function

void PLIB_GLCD_IRQTriggerControlSet(GLCD_MODULE_ID index,
GLCD_IRQ_TRIGGER_CONTROL irqControl)

PLIB_GLCD_VSyncInterruptDisable Function

Disables interrupts at Vsync.

File

plib_glcd.h

C

```
void PLIB_GLCD_VSyncInterruptDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables interrupts at Vsync.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_VsyncInterruptDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_VSyncInterruptDisable(GLCD_MODULE_ID index)

PLIB_GLCD_VSyncInterruptEnable Function

Enables interrupts at Vsync.

File

plib_glcd.h

C

```
void PLIB_GLCD_VSyncInterruptEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables interrupts at Vsync.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None

Example

```
PLIB_GLCD_VSyncInterruptEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_VSyncInterruptEnable(GLCD_MODULE_ID index)

PLIB_GLCD_VSyncInterruptIsEnabled Function

Verify whether VSYNC Interrupt is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_VSyncInterruptIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The VSYNC Interrupt feature is enabled.
- false - The VSYNC Interrupt feature is disabled.

Description

This function verifies whether the VSYNC Interrupt is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_VSyncInterruptIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_VSyncInterruptDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_VSyncInterruptIsEnabled(GLCD_MODULE_ID index)

PLIB_GLCD_VSyncSignalLevelGet Function

Gets the Vsync signal level.

File

plib_glcd.h

C

```
bool PLIB_GLCD_VSyncSignalLevelGet(GLCD_MODULE_ID index);
```

Returns

- true - The Vsync signal level is high.
- false - The Vsync signal level is low.

Description

This function returns the Vsync signal level.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(!PLIB_GLCD_VSyncSignalLevelGet( GLCD_MODULE_ID_0 ));
{
    //Vsync signal level is low
    return;
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_VSyncSignalLevelGet(GLCD_MODULE_ID index)

h) Miscellaneous Control Functions

PLIB_GLCD_DitheringDisable Function

Disables the Dithering feature of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_DitheringDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables the Dithering feature of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_DitheringDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_DitheringDisable(GLCD_MODULE_ID index)

PLIB_GLCD_DitheringEnable Function

Enables the Dithering feature of the Graphics LCD Controller.

File

plib_glcd.h

C

```
void PLIB_GLCD_DitheringEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables the Dithering feature of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_DitheringEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_DitheringEnable(GLCD_MODULE_ID index)

PLIB_GLCD_ForceOutputBlankDisable Function

Disable Force Output Blank feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_ForceOutputBlankDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function disables the Force Output Blank feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_ForceOutputBlankDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_ForceOutputBlankDisable(GLCD_MODULE_ID index)

PLIB_GLCD_ForceOutputBlankEnable Function

Enable Force Output Blank feature.

File

plib_glcd.h

C

```
void PLIB_GLCD_ForceOutputBlankEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables the Force Output Blank feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_ForceOutputBlankEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_ForceOutputBlankEnable(GLCD_MODULE_ID index)

PLIB_GLCD_SingleCyclePerLineVsyncDisable Function

Clears VSYNC on single cycle per line.

File

plib_glcd.h

C

```
void PLIB_GLCD_SingleCyclePerLineVsyncDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function clears VSYNC on single cycle per line.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_SingleCyclePerLineVsyncClear( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_SingleCyclePerLineVsyncClear(GLCD_MODULE_ID index)

PLIB_GLCD_SingleCyclePerLineVsyncEnable Function

Sets VSYNC on single cycle per line.

File

plib_glcd.h

C

```
void PLIB_GLCD_SingleCyclePerLineVsyncEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function sets VSYNC on single cycle per line.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_SingleCyclePerLineVsyncEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_SingleCyclePerLineVsyncSet(GLCD_MODULE_ID index)

PLIB_GLCD_SingleCyclePerLineVsyncIsEnabled Function

Verify whether VSYNC on Single Cycle Per Line feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_SingleCyclePerLineVsyncIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - VSYNC on Single Cycle Per Line is enabled.
- false - VSYNC on Single Cycle Per Line is disabled.

Description

This function verifies whether VSYNC on Single Cycle Per Line feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(PLIB_GLCD_SingleCyclePerLineVsyncIsEnabled(GLCD_MODULE_ID_0))
{
    PLIB_GLCD_SingleCyclePerLineVsyncDisable(GLCD_MODULE_ID_0);
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_SingleCyclePerLineVsyncIsEnabled(GLCD_MODULE_ID index)

PLIB_GLCD_YUVOutputDisable Function

Disables the output of the Graphics LCD Controller in YUV format.

File

plib_glcd.h

C

```
void PLIB_GLCD_YUVOutputDisable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function Disables the output of the Graphics LCD Controller in YUV format.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_YUVOutputDisable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_YUVOutputDisable(GLCD_MODULE_ID index)

PLIB_GLCD_YUVOutputEnable Function

Enables the output of the Graphics LCD Controller in YUV format.

File

plib_glcd.h

C

```
void PLIB_GLCD_YUVOutputEnable(GLCD_MODULE_ID index);
```

Returns

None.

Description

This function enables the output of the Graphics LCD Controller in YUV format.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
PLIB_GLCD_YUVOutputEnable( GLCD_MODULE_ID_0 );
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

void PLIB_GLCD_YUVOutputEnable(GLCD_MODULE_ID index)

PLIB_GLCD_DESignalLevelGet Function

Gets the display enable signal level.

File

plib_glcd.h

C

```
bool PLIB_GLCD_DESignalLevelGet(GLCD_MODULE_ID index);
```

Returns

- true - The DE signal level high.
- false - The DE signal level low.

Description

This function returns the display enable (DE) signal of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(!PLIB_GLCD_DESignalLevelGet( GLCD_MODULE_ID_0 ));
{
    //DE signal low
    return;
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_DESignalLevelGet(GLCD_MODULE_ID index)

PLIB_GLCD_DitheringIsEnabled Function

Verifies whether Dithering is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_DitheringIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The Dithering feature is enabled.
- false - The Dithering feature is disabled.

Description

This function verifies whether Dithering is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(PLIB_GLCD_DitheringIsEnabled(GLCD_MODULE_ID_0 ))
{
    PLIB_GLCD_DitheringDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_DitheringIsEnabled(GLCD_MODULE_ID index)

PLIB_GLCD_ForceOutputBlankIsEnabled Function

Verify whether the Force Output Blank feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ForceOutputBlankIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The Force Output Blank feature is enabled.
- false - The Force Output Blank feature is disabled.

Description

This function verifies whether the Force Output Blank feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ForceOutputBlankIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_ForceOutputBlankDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ForceOutputBlankIsEnabled(GLCD_MODULE_ID index)

PLIB_GLCD_IsEnabled Function

Verifies whether the Graphics LCD Controller is Enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_IsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - Graphics LCD Controller is Enabled
- false - Graphics LCD Controller is Disabled

Description

This function verifies whether the Graphics LCD Controller is Enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_IsEnabled( GLCD_MODULE_ID_0 ) )
```

```
{
    PLIB_GLCD_Disable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_IsEnabled (GLCD_MODULE_ID index)

PLIB_GLCD_IsLastRow Function

Gets the status indicating whether a last row is currently displayed by the Graphics Display Controller.

File

plib_glcd.h

C

```
bool PLIB_GLCD_IsLastRow(GLCD_MODULE_ID index);
```

Returns

- true - The last row is currently displayed.
- false - The last row is not currently displayed.

Description

This function returns the status indicating whether a last row is currently displayed by the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(!PLIB_GLCD_IsLastRow( GLCD_MODULE_ID_0 ));
{
    // The last row is not currently displayed
    return;
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_IsLastRow(GLCD_MODULE_ID index)

PLIB_GLCD_IsVerticalBlankingActive Function

Get the active status.

File

plib_glcd.h

C

```
bool PLIB_GLCD_IsVerticalBlankingActive(GLCD_MODULE_ID index);
```

Returns

- true - The Vertical Blanking is active.
- false - The Vertical Blanking is inactive.

Description

This function returns the active status of the Graphics LCD Controller.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_IsVerticalBlankingActive( GLCD_MODULE_ID_0 ) )
{
    //GLCD Vertical Blanking active
    return;
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_IsVerticalBlankingActive(GLCD_MODULE_ID index)

PLIB_GLCD_RGBSequentialModeGet Function

Get the RGB Sequential Mode already set.

File

plib_glcd.h

C

```
GLCD_RGB_MODE PLIB_GLCD_RGBSequentialModeGet(GLCD_MODULE_ID index);
```

Returns

- GLCD_RGB_MODE_PARALLEL_RGB565 - Parallel RGB 565 Mode
- GLCD_RGB_MODE_PARALLEL_RGB888 - Parallel RGB 888 Mode
- GLCD_RGB_MODE_SERIAL_RGB_3 - Byte Serial RGB 3 Mode
- GLCD_RGB_MODE_SERIAL_RGBA_4 - Byte Serial RGB 4 Mode
- GLCD_RGB_MODE_SERIAL_12BIT - Byte Two-Phase 12-bit Mode
- GLCD_RGB_MODE_YUYV_16BIT - YUYV 16 bit Mode
- GLCD_RGB_MODE_BT_656 - BT 656 Mode

Description

This function gets the RGB Sequential Mode already set.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
uint32_t rgbMode;

rgbMode = PLIB_GLCD_RGBSequentialModeGet( GLCD_MODULE_ID_0 );

if( rgbMode == GLCD_RGB_MODE_PARALLEL_RGB565 )
{
    // RGB Mode set to RGB 565 format
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

GLCD_RGB_MODE PLIB_GLCD_RGBSequentialModeGet(GLCD_MODULE_ID index)

PLIB_GLCD_YUVOutputIsEnabled Function

Verify whether the YUV Output feature is enabled.

File

plib_glcd.h

C

```
bool PLIB_GLCD_YUVOutputIsEnabled(GLCD_MODULE_ID index);
```

Returns

- true - The YUV output feature is enabled.
- false - The YUV output feature is disabled.

Description

This function verifies whether the YUV Output feature is enabled.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_YUVOutputIsEnabled( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_YUVOutputDisable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_YUVOutputIsEnabled(GLCD_MODULE_ID index)

i) Feature Existence Functions

PLIB_GLCD_ExistsBackgroundColor Function

Verify whether the Background Color Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsBackgroundColor(GLCD_MODULE_ID index);
```

Returns

- true - The Background Color feature is supported.
- false - The Background Color feature is not supported.

Description

This function verifies whether the Background Color Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsBackgroundColor( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_BackgroundColorSet( GLCD_MODULE_ID_0,
                                  MY_BACKGROUND_COLOR_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsBackgroundColor(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsBackPorchXY Function

Verify whether Back Porch Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsBackPorchXY(GLCD_MODULE_ID index);
```

Returns

- true - The Back Porch feature is supported.
- false - The Back Porch feature is not supported.

Description

This function verifies whether Back Porch Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsBackPorchXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_BackPorchXYSet( GLCD_MODULE_ID_0,
                              MY_BACK_PORCH_X_VALUE,
                              MY_BACK_PORCH_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsBackPorchXY(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsBlankingXY Function

Verify whether Blanking Period Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsBlankingXY(GLCD_MODULE_ID index);
```

Returns

- true - The Blanking feature is supported.
- false - The Blanking feature is not supported.

Description

This function verifies whether Blanking Period Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsBlankingXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_BlankingXYSet( GLCD_MODULE_ID_0,
                             MY_BLANKING_PERIOD_X_VALUE,
                             MY_BLANKING_PERIOD_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsBlankingXY(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsClockDivider Function

Verify whether the Clock Divider feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsClockDivider(GLCD_MODULE_ID index);
```

Returns

- true - The Clock divide feature is supported.
- false - The Clock divide feature is not supported.

Description

This function verifies whether the Clock Divider feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsClockDivider( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_ClockDividerSet( GLCD_MODULE_ID_0, MY_CLOCK_DIVIDER_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsClockDivider(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsCursor Function

Verifies whether the cursor feature exists.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsCursor(GLCD_MODULE_ID index);
```

Returns

- true - Cursor feature exists.
- false - Cursor feature does not exits.

Description

This function verifies whether the cursor feature exists.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ExistsCursor( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsCursor(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsCursorData Function

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsCursorData(GLCD_MODULE_ID index);
```

Returns

- true - The Cursor Data feature is supported.
- false - The Cursor Data feature is not supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsCursorData( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorDataSet( GLCD_MODULE_ID_0,
                             MY_CURSOR_DATA_POINTER_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsCursorData(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsCursorLUT Function

Verify whether Cursor LUT feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsCursorLUT(GLCD_MODULE_ID index);
```

Returns

- true - The Cursor LUT feature is supported.
- false - The Cursor LUT feature is not supported.

Description

This function verifies whether Cursor LUT feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsCursorLUT( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorLUTSet( GLCD_MODULE_ID_0,
                            MY_CURSOR_LUT_DATA_POINTER_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsCursorLUT(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsCursorXY Function

Verify whether Cursor XY Position feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsCursorXY(GLCD_MODULE_ID index);
```

Returns

- true - The Cursor Position feature is supported.
- false - The Cursor Position feature is not supported.

Description

This function verifies whether Cursor XY Position feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsCursorXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_CursorXYSet( GLCD_MODULE_ID_0,
                           MY_CURSOR_POSITION_X_VALUE,
                           MY_CURSOR_POSITION_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsCursorXY(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsDESignalLevel Function

Verify whether DE Signal Level feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsDESignalLevel(GLCD_MODULE_ID index);
```

Returns

- true - The DE Signal Level feature is supported.
- false - The DE Signal Level feature is not supported.

Description

This function verifies whether DE Signal Level feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsDESignalLevel( GLCD_MODULE_ID_0 ) )
{
    if( PLIB_GLCD_DESignalLevelGet( GLCD_MODULE_ID_0 ) )
    {
```

```
            // DE Signal Level High
        }
    }
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsDESignalLevel(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsDithering Function

Verifies whether the Dithering feature exists.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsDithering(GLCD_MODULE_ID index);
```

Returns

- true - The Dithering feature is supported on the device
- false - The Dithering feature is not supported on the device

Description

This function verifies whether the Dithering feature exists.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ExistsDithering( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_DitheringEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsDithering(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsEnable Function

Identifies whether the GLCD Enable feature exists on the GLCD module.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsEnable(GLCD_MODULE_ID index);
```

Returns

- true - The GLCD Enable feature is supported on the device
- false - The GLCD Enable feature is not supported on the device

Description

This function identifies whether the GLCD Enable feature is available on the GLCD module. When this function returns true, these functions are supported on the device:

- PLIB_GLCD_Enable
- PLIB_GLCD_Disable

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ExistsEnable( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_Enable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_GLCD_ExistsEnable (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsForceOutputBlank Function

Verify whether Force Output Blank feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsForceOutputBlank(GLCD_MODULE_ID index);
```

Returns

- true - The force output blank feature supported
- false - The force output blank feature not supported

Description

This function verifies whether Force Output Blank feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsForceOutputBlank( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_ForceOutputBlankEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsForceOutputBlank(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsFormattingClockDivide Function

Verify whether Clock Formatting feature is available.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsFormattingClockDivide(GLCD_MODULE_ID index);
```

Returns

- true - The Clock formatting feature is supported.
- false - The Clock formatting feature is not supported.

Description

This function verifies whether Clock formatting feature is available.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsFormattingClockDivide( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_FormattingClockDivideEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsFormattingClockDivide(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsFrontPorchXY Function

Verify whether Front Porch Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsFrontPorchXY(GLCD_MODULE_ID index);
```

Returns

- true - The Front Porch feature is supported.
- false - The Front Porch feature is not supported.

Description

This funtion verifies whether Front Porch Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsFrontPorchXY( GLCD_MODULE_ID_0 ) )
```

```
{
    PLIB_GLCD_FrontPorchXYSet( GLCD_MODULE_ID_0,
                               MY_FRONT_PORCH_X_VALUE,
                               MY_FRONT_PORCH_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsFrontPorchXY(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsGlobalColorLUT Function

Verify whether Global Color LUT feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsGlobalColorLUT(GLCD_MODULE_ID index);
```

Returns

- true - The Global Color LUT feature is supported.
- false - The Global Color LUT feature is not supported.

Description

This function verifies whether Global Color LUT feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsGlobalColorLUT( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_GlobalColorLUTSet( GLCD_MODULE_ID_0,
                                 MY_GLOBAL_LUT_POINTER_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsGlobalColorLUT(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsHSyncInterruptEnable Function

Verify whether the HSYNC Interrupt Enable feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsHSyncInterruptEnable(GLCD_MODULE_ID index);
```

Returns

- true - The HSYNC Interrupt Enable feature is supported.

- false - The HSYNC Interrupt Enable feature is not supported.

Description

This function verifies whether the HSYNC Interrupt Enable feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsHSyncInterruptEnable( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_HSyncInterruptEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsHSyncInterruptEnable(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsHSyncSignalLevel Function

Verify whether HSYNC Signal Level feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsHSyncSignalLevel(GLCD_MODULE_ID index);
```

Returns

- true - The HSYNC Signal Level feature is supported.
- false - The HSYNC Signal Level feature is not supported.

Description

This function verifies whether the HSYNC Signal Level feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsHSyncSignalLevel( GLCD_MODULE_ID_0 ) )
{
    if( PLIB_GLCD_HSyncSignalLevelGet( GLCD_MODULE_ID_0 ) )
    {
       // HSYNC Signal Level High
    }
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsHSyncSignalLevel(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsIRQTriggerControl Function

Verify whether the IRQ Trigger Control feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsIRQTriggerControl(GLCD_MODULE_ID index);
```

Returns

- true - The IRQ Trigger Control feature is supported.
- false - The IRQ Trigger Control feature is not supported.

Description

This function verifies whether the IRQ Trigger Control feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsIRQTriggerControl( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_IRQTriggerControlSet( GLCD_MODULE_ID_0,
                                    MY_IRQ_TRIGGER_CONTROL );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsIRQTriggerControl(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsIsLastRow Function

Verify whether Is Last Row Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsIsLastRow(GLCD_MODULE_ID index);
```

Returns

- true - The Is Last Row feature is supported.
- false - The Is Last Row feature is not supported.

Description

This function verifies whether Is Last Row Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsIsLastRow( GLCD_MODULE_ID_0 ) )
{
    if( PLIB_GLCD_IsLastRow( GLCD_MODULE_ID_0 ) )
    {
       // Last row currently displayed
    }
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsIsLastRow(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsIsVerticalBlankingActive Function

Verify whether Is Vertical Blanking Active feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsIsVerticalBlankingActive(GLCD_MODULE_ID index);
```

Returns

- true - The Is Vertical Blanking feature is supported.
- false - The Is Vertical Blanking feature is not supported.

Description

This function verifies whether Is Vertical Blanking Active feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsIsVerticalBlankingActive( GLCD_MODULE_ID_0 ) )
{
    if( PLIB_GLCD_IsVerticalBlankingActive( GLCD_MODULE_ID_0 ) )
    {
       // Vertical Blanking Active
    }
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsIsVerticalBlankingActive(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerBaseAddress Function

Verify whether the Layer Base Address feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerBaseAddress(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Base Address feature is supported.
- false - The Layer Base Address feature is not supported.

Description

This function verifies whether the Layer Base Address feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerBaseAddress( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerBaseAddressSet( GLCD_MODULE_ID_0,
                                   GLCD_LAYER_0,
                                   MY_LAYER_BASE_ADDRESS_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerBaseAddress (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerBilinearFilterEnable Function

Enable Layer Bilinear Filter Feature.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerBilinearFilterEnable(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Bilinear Filter feature is supported.
- false - The Layer Bilinear Filter feature is not supported.

Description

This function enables the Layer Bilinear Filter Feature.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerBilinearFilterEnable( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerBilinearFilterEnable( GLCD_MODULE_ID_0,
                                         GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerBilinearFilterEnable (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerColorMode Function

Verify whether Layer Color Mode is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerColorMode(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Color Mode feature is supported.
- false - The Layer Color Mode feature is not supported.

Description

This function verifies whether Layer Color Mode is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerColorMode( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerColorModeSet( GLCD_MODULE_ID_0,
                                 GLCD_LAYER_0,
                                 MY_LAYER_COLOR_MODE_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerColorMode (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerDestBlendFunc Function

Verify whether Layer Destination Blend Function Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerDestBlendFunc(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Destination Blend Function feature is supported.
- false - The Layer Destination Blend Function feature is not supported.

Description

This function verifies whether Layer Destination Blend Function Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerDestBlendFunc( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerDestBlendFuncSet( GLCD_MODULE_ID_0,
                                     GLCD_LAYER_0,
                                     MY_LAYER_DEST_BLEND_FUNC_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerDestBlendFunc (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerEnable Function

Verify whether Layer Enable Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerEnable(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Enable feature is supported.
- false - The Layer Enable feature is not supported.

Description

This function verifies whether Layer enable Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerEnable( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerEnable( GLCD_MODULE_ID_0, GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerEnable (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerForceWithGlobalAlpha Function

Verify whether Layer Force with Global Alpha Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerForceWithGlobalAlpha(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Force Global Alpha feature is supported.
- false - The Layer Force Global Alpha feature is not supported.

Description

This function verifies whether Layer Force with Global Alpha Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerForceWithGlobalAlpha( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerForceWithGlobalAlphaEnable( GLCD_MODULE_ID_0,
                                               GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerForceWithGlobalAlpha (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerGlobalAlpha Function

Verify whether Layer Global Alpha Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerGlobalAlpha(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Global Alpha feature is supported.
- false - The Layer Global Alpha feature is not supported.

Description

This function verifies whether Layer Global Alpha Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerGlobalAlpha( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerGlobalAlphaSet( GLCD_MODULE_ID_0,
                                   GLCD_LAYER_0,
                                   MY_LAYER_GLOBAL_ALPHA_VALUE );
```

```
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerGlobalAlpha(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerPreMultiplyImageAlpha Function

Verify whether Layer Pre-Multiply Image Alpha Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerPreMultiplyImageAlpha(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Pre-Multiply Image Alpha feature is supported.
- false - The Layer Pre-Multiply Image Alpha feature is not supported.

Description

This function verifies whether Layer Pre-Multiply Image Alpha Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerPreMultiplyImageAlpha( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerPreMultiplyImageAlphaEnable( GLCD_MODULE_ID_0,
                                                GLCD_LAYER_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerPreMultiplyImageAlpha (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerResXY Function

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerResXY(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Resolution feature is supported.
- false - The Layer Resolution feature is not supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerResXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerResXYSet( GLCD_MODULE_ID_0,
                             GLCD_LAYER_0,
                             MY_LAYER_RES_X_VALUE,
                             MY_LAYER_RES_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerResXY(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerSizeXY Function

Verify whether Layer X Axis and Y Axis Size feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerSizeXY(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Size feature is supported.
- false - The Layer Size feature is not supported.

Description

This function verifies whether Layer X Axis and Y Axis Size feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerSizeXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerSizeXYSet( GLCD_MODULE_ID_0,
                              GLCD_LAYER_0,
                              MY_LAYER_SIZE_X_VALUE,
                              MY_LAYER_SIZE_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerSizeXY (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerSrcBlendFunc Function

Verify whether Layer Source Blend Function Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerSrcBlendFunc(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Source Blend Function feature is supported.
- false - The Layer Source Blend Function feature is not supported.

Description

This function verifies whether Layer Source Blend Function Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerSrcBlendFunc( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerSrcBlendFuncSet( GLCD_MODULE_ID_0,
                                    GLCD_LAYER_0,
                                    MY_LAYER_SRC_BLEND_FUNC_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerSrcBlendFunc (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerStartXY Function

Verify whether Layer Start XY Position feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerStartXY(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Start feature is supported.
- false - The Layer Start feature is not supported.

Description

This function verifies whether Layer Start XY Position feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLayerStartXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LayerStartXYSet( GLCD_MODULE_ID_0,
                               GLCD_LAYER_0,
```

```
                                MY_LAYER_START_X_VALUE,
                                MY_LAYER_START_Y_VALUE );
    }
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerStartXY (GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLayerStride Function

Verify whether the Layer Stride Feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLayerStride(GLCD_MODULE_ID index);
```

Returns

- true - The Layer Stride feature is supported.
- false - The Layer Stride feature is not supported.

Description

This function verifies whether the Layer Stride Feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
    if( PLIB_GLCD_ExistsLayerStride( GLCD_MODULE_ID_0 ) )
    {
        PLIB_GLCD_LayerStrideSet( GLCD_MODULE_ID_0,
                                  GLCD_LAYER_0,
                                  MY_LAYER_STRIDE_VALUE );
    }
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLayerStride(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsLinesPrefetch Function

Verify whether Lines Prefetch Set Feature supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsLinesPrefetch(GLCD_MODULE_ID index);
```

Returns

- true - The Lines Prefetch feature is supported.
- false - The Lines Prefetch feature is not supported.

Description

This function verifies whether Lines Prefetch Set Feature supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsLinesPrefetch( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_LinesPrefetchSet( GLCD_MODULE_ID_0,
                                MY_LINES_PREFETCH_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsLinesPrefetch(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsPaletteGammaRamp Function

Verifies whether the palette / gamma ramp feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsPaletteGammaRamp(GLCD_MODULE_ID index);
```

Returns

- true - The palette / gamma ramp enable feature supported.
- false - The palette / gamma ramp enable feature not supported.

Description

This function verifies whether the palette / gamma ramp feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsPaletteGammaRamp( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_PaletteGammaRampEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsPaletteGammaRamp(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsResolutionXY Function

Verify whether YUV Output feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsResolutionXY(GLCD_MODULE_ID index);
```

Returns

- true - The Resolution feature is supported.
- false - The Resolution feature is not supported.

Description

This function verifies whether YUV Output feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsResolutionXY( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_ResolutionXYSet( GLCD_MODULE_ID_0,
                               MY_RESOLUTION_X_VALUE,
                               MY_RESOLUTION_Y_VALUE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsYUVOutput(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsRGBSequentialMode Function

Verify whether RGB Sequential Mode feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsRGBSequentialMode(GLCD_MODULE_ID index);
```

Returns

- true - The RGB Sequential Mode feature is supported.
- false - The RGB Sequential Mode feature is not supported.

Description

This function verifies whether RGB Sequential Mode feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsRGBSequentialMode( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_RGBSequentialModeSet( GLCD_MODULE_ID_0,
                                    GLCD_RGB_MODE_PARALLEL );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsRGBSequentialMode(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsSignalPolarity Function

Verifies whether the Signal Polarity Selection feature exists.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsSignalPolarity(GLCD_MODULE_ID index);
```

Returns

- true - Feature exists
- false - Feature does not exist

Description

This function verifies whether the Signal Polarity Selection feature exists.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if ( PLIB_GLCD_ExistsSignalPolarity( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_SignalPolaritySet( GLCD_MODULE_ID_0,
                                 GLCD_POLARITY_POSITIVE |
                                 GLCD_DE_POLARITY_NEGATIVE );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsSignalPolarity(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsSingleCyclePerLineVsync Function

Verifies whether VSYNC on Single cycle Per Line Feature exists.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsSingleCyclePerLineVsync(GLCD_MODULE_ID index);
```

Returns

- true - The feature: VSYNC on Single Cycle Per Line is supported on the device.
- false - The feature: VSYNC on Single Cycle Per Line is not supported on the device.

Description

This function verifies whether VSYNC on Single cycle Per Line Feature exists.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if(PLIB_GLCD_ExistsSingleCyclePerLineVsync( GLCD_MODULE_ID_0))
{
    PLIB_GLCD_SingleCyclePerLineVsyncEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsSingleCycleVsync(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsVSyncInterruptEnable Function

Verify whether VSYNC Interrupt Enable feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsVSyncInterruptEnable(GLCD_MODULE_ID index);
```

Returns

- true - The VSYNC Interrupt Enable feature is supported.
- false - The VSYNC Interrupt Enable feature is not supported.

Description

This function verifies whether VSYNC Interrupt Enable feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsVSyncInterruptEnable( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_VSyncInterruptEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsVSyncInterruptEnable(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsVSyncSignalLevel Function

Verify whether VSYNC Signal Level feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsVSyncSignalLevel(GLCD_MODULE_ID index);
```

Returns

- true - The VSYNC Signal Level feature is supported.
- false - The VSYNC Signal Level feature is not supported.

Description

This function verifies whether VSYNC Signal Level feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsVSyncSignalLevel( GLCD_MODULE_ID_0 ) )
{
    if( PLIB_GLCD_VSyncSignalLevelGet( GLCD_MODULE_ID_0 ) )
    {
       // VSYNC Signal Level High
    }
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsVSyncSignalLevel(GLCD_MODULE_ID index)

PLIB_GLCD_ExistsYUVOutput Function

Verify whether YUV output feature is supported.

File

plib_glcd.h

C

```
bool PLIB_GLCD_ExistsYUVOutput(GLCD_MODULE_ID index);
```

Returns

- true - The YUV output feature is supported.
- false - The YUV output feature is not supported.

Description

This function verifies whether YUV output feature is supported.

Remarks

This functionality is not available on all devices. Please refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
if( PLIB_GLCD_ExistsYUVOutput( GLCD_MODULE_ID_0 ) )
{
    PLIB_GLCD_YUVOutputEnable( GLCD_MODULE_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier of the device instance.

Function

bool PLIB_GLCD_ExistsYUVOutput(GLCD_MODULE_ID index)

j) Data Types and Constants

GLCD_IRQ_TRIGGER_CONTROL Enumeration

Possible values of GLCD Interrupt Trigger Mode.

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_IRQ_TRIGGER_LEVEL = 0x00000000,
  GLCD_IRQ_TRIGGER_EDGE = 0x80000000
} GLCD_IRQ_TRIGGER_CONTROL;
```

Members

Members	Description
GLCD_IRQ_TRIGGER_LEVEL = 0x00000000	Interrupt Level Triggered
GLCD_IRQ_TRIGGER_EDGE = 0x80000000	Interrupt Edge Triggered

Description

GLCD Interrupt Trigger Mode

This data type defines the possible values of GLCD Interrupt Trigger modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_LAYER_COLOR_MODE Enumeration

Possible values of GLCD color Mode.

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_LAYER_COLOR_MODE_LUT8 = 0x0,
  GLCD_LAYER_COLOR_MODE_RGBA5551 = 0x1,
  GLCD_LAYER_COLOR_MODE_RGBA8888 = 0x2,
  GLCD_LAYER_COLOR_MODE_RGB332 = 0x4,
  GLCD_LAYER_COLOR_MODE_RGB565 = 0x5,
  GLCD_LAYER_COLOR_MODE_ARGB8888 = 0x6,
  GLCD_LAYER_COLOR_MODE_L8 = 0x7,
  GLCD_LAYER_COLOR_MODE_L1 = 0x8,
  GLCD_LAYER_COLOR_MODE_L4 = 0x9,
  GLCD_LAYER_COLOR_MODE_YUYV = 0xA,
  GLCD_LAYER_COLOR_MODE_RGB888 = 0xB
} GLCD_LAYER_COLOR_MODE;
```

Members

Members	Description
GLCD_LAYER_COLOR_MODE_LUT8 = 0x0	GLCD Layer color mode LUT 8
GLCD_LAYER_COLOR_MODE_RGBA5551 = 0x1	GLCD Layer color mode RGBA 5551
GLCD_LAYER_COLOR_MODE_RGBA8888 = 0x2	GLCD Layer color mode RGBA 8888
GLCD_LAYER_COLOR_MODE_RGB332 = 0x4	GLCD Layer color mode RGB 332
GLCD_LAYER_COLOR_MODE_RGB565 = 0x5	GLCD Layer color mode RGB 565
GLCD_LAYER_COLOR_MODE_ARGB8888 = 0x6	GLCD Layer color mode ARGB 8888
GLCD_LAYER_COLOR_MODE_L8 = 0x7	GLCD Layer color mode L8
GLCD_LAYER_COLOR_MODE_L1 = 0x8	GLCD Layer color mode L1
GLCD_LAYER_COLOR_MODE_L4 = 0x9	GLCD Layer color mode L4
GLCD_LAYER_COLOR_MODE_YUYV = 0xA	GLCD Layer color mode YUYV
GLCD_LAYER_COLOR_MODE_RGB888 = 0xB	GLCD Layer color mode RGB888

Description

GLCD Layer color Mode

This data type defines the possible values of GLCD Layer color modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_LAYER_DEST_BLEND_FUNC Enumeration

Possible values of GLCD Layer Destination Blend Functions.

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_LAYER_DEST_BLEND_BLACK = 0x0000,
  GLCD_LAYER_DEST_BLEND_WHITE = 0x1000,
  GLCD_LAYER_DEST_BLEND_ALPHA_SRC = 0x2000,
  GLCD_LAYER_DEST_BLEND_ALPHA_GBL = 0x3000,
  GLCD_LAYER_DEST_BLEND_ALPHA_SRCGBL = 0x4000,
  GLCD_LAYER_DEST_BLEND_INV_SRC = 0x5000,
  GLCD_LAYER_DEST_BLEND_INV_GBL = 0x6000,
  GLCD_LAYER_DEST_BLEND_INV_SRCGBL = 0x7000,
  GLCD_LAYER_DEST_BLEND_ALPHA_DST = 0xA000,
  GLCD_LAYER_DEST_BLEND_INV_DST = 0xD000
} GLCD_LAYER_DEST_BLEND_FUNC;
```

Members

Members	Description
GLCD_LAYER_DEST_BLEND_BLACK = 0x0000	Layer Destination Blend Function: Black, Destination Alpha = 0
GLCD_LAYER_DEST_BLEND_WHITE = 0x1000	Layer Destination Blend Function: White, Destination Alpha = 255
GLCD_LAYER_DEST_BLEND_ALPHA_SRC = 0x2000	Layer Destination Blend Function: Source Alpha
GLCD_LAYER_DEST_BLEND_ALPHA_GBL = 0x3000	Layer Destination Blend Function: Global Alpha
GLCD_LAYER_DEST_BLEND_ALPHA_SRCGBL = 0x4000	Layer Destination Blend Function: (Source Alpha * Global Alpha)
GLCD_LAYER_DEST_BLEND_INV_SRC = 0x5000	Layer Destination Blend Function: (1 - Source Alpha)
GLCD_LAYER_DEST_BLEND_INV_GBL = 0x6000	Layer Destination Blend Function: (1 - Global Alpha)
GLCD_LAYER_DEST_BLEND_INV_SRCGBL = 0x7000	Layer Destination Blend Function: (1 - (Source Alpha * Global Alpha))

GLCD_LAYER_DEST_BLEND_ALPHA_DST = 0xA000	Layer Destination Blend Function: Destination Alpha
GLCD_LAYER_DEST_BLEND_INV_DST = 0xD000	Layer Destination Blend Function: (1 - Destination Alpha)

Description

GLCD Layer Destination Blend Function

This data type defines the possible values of GLCD Layer Destination Blend Functions.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_LAYER_ID Enumeration

Possible values of GLCD Layer Ids

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_LAYER_ID_0 = 0x0,
  GLCD_LAYER_ID_1 = 0x1,
  GLCD_LAYER_ID_2 = 0x2,
  GLCD_LAYER_ID_MAX = 0x3
} GLCD_LAYER_ID;
```

Members

Members	Description
GLCD_LAYER_ID_0 = 0x0	GLCD Layer 0 Id
GLCD_LAYER_ID_1 = 0x1	GLCD Layer 1 Id
GLCD_LAYER_ID_2 = 0x2	GLCD Layer 2 Id
GLCD_LAYER_ID_MAX = 0x3	Maximum number of Layers supported by GLCD

Description

GLCD Layer Ids

This data type defines the possible values of GLCD Layer Ids.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_LAYER_SRC_BLEND_FUNC Enumeration

Possible values of GLCD Layer Source Blend Functions.

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_LAYER_SRC_BLEND_BLACK = 0x0000,
  GLCD_LAYER_SRC_BLEND_WHITE = 0x0100,
  GLCD_LAYER_SRC_BLEND_ALPHA_SRC = 0x0200,
  GLCD_LAYER_SRC_BLEND_ALPHA_GBL = 0x0300,
  GLCD_LAYER_SRC_BLEND_ALPHA_SRCGBL = 0x0400,
  GLCD_LAYER_SRC_BLEND_INV_SRC = 0x0500,
  GLCD_LAYER_SRC_BLEND_INV_GBL = 0x0600,
  GLCD_LAYER_SRC_BLEND_INV_SRCGBL = 0x0700,
  GLCD_LAYER_SRC_BLEND_ALPHA_DST = 0x0A00,
  GLCD_LAYER_SRC_BLEND_INV_DST = 0x0D00
} GLCD_LAYER_SRC_BLEND_FUNC;
```

Members

Members	Description
GLCD_LAYER_SRC_BLEND_BLACK = 0x0000	Layer Source Blend Function: Black, Source Alpha = 0
GLCD_LAYER_SRC_BLEND_WHITE = 0x0100	Layer Source Blend Function: White, Source Alpha = 255
GLCD_LAYER_SRC_BLEND_ALPHA_SRC = 0x0200	Layer Source Blend Function: Source Alpha
GLCD_LAYER_SRC_BLEND_ALPHA_GBL = 0x0300	Layer Source Blend Function: Global Alpha
GLCD_LAYER_SRC_BLEND_ALPHA_SRCGBL = 0x0400	Layer Source Blend Function: (Source Alpha * Global Alpha)
GLCD_LAYER_SRC_BLEND_INV_SRC = 0x0500	Layer Source Blend Function: (1 - Source Alpha)
GLCD_LAYER_SRC_BLEND_INV_GBL = 0x0600	Layer Source Blend Function: (1 - Global Alpha)
GLCD_LAYER_SRC_BLEND_INV_SRCGBL = 0x0700	Layer Source Blend Function: (1 - (Source Alpha * Global Alpha))
GLCD_LAYER_SRC_BLEND_ALPHA_DST = 0x0A00	Layer Source Blend Function: Destination Alpha
GLCD_LAYER_SRC_BLEND_INV_DST = 0x0D00	Layer Source Blend Function: (1 - Destination Alpha)

Description

GLCD Layer Source Blend Function

This data type defines the possible values of GLCD Layer Source Blend Functions.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_MODULE_ID Enumeration

Possible instances of the GLCD module

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_ID_0 = 0,
  GLCD_NUMBER_OF_MODULES
} GLCD_MODULE_ID;
```

Members

Members	Description
GLCD_ID_0 = 0	first instance of the GLCD
GLCD_NUMBER_OF_MODULES	number of GLCD modules

Description

GLCD module ID

This data type defines the possible instances of the GLCD module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_RGB_MODE Enumeration

GLCD RGB Sequential Mode

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_RGB_MODE_PARALLEL_RGB565 = 0x00000000,
  GLCD_RGB_MODE_PARALLEL_RGB888 = 0x00000001,
  GLCD_RGB_MODE_SERIAL_RGB_3 = 0x00000002,
  GLCD_RGB_MODE_SERIAL_RGBA_4 = 0x00000003,
  GLCD_RGB_MODE_SERIAL_12BIT = 0x00000004,
  GLCD_RGB_MODE_YUYV_16BIT = 0x00000005,
  GLCD_RGB_MODE_BT_656 = 0x00000006
} GLCD_RGB_MODE;
```

Members

Members	Description
GLCD_RGB_MODE_PARALLEL_RGB565 = 0x00000000	Parallel RGB 565 Mode
GLCD_RGB_MODE_PARALLEL_RGB888 = 0x00000001	Parallel RGB 888 Mode
GLCD_RGB_MODE_SERIAL_RGB_3 = 0x00000002	Byte Serial RGB 3 Mode
GLCD_RGB_MODE_SERIAL_RGBA_4 = 0x00000003	Byte Serial RGB 4 Mode
GLCD_RGB_MODE_SERIAL_12BIT = 0x00000004	Byte Two-Phase 12-bit Mode
GLCD_RGB_MODE_YUYV_16BIT = 0x00000005	YUYV 16 bit Mode
GLCD_RGB_MODE_BT_656 = 0x00000006	BT 656 Mode

Description

GLCD RGB Sequential Mode

This data type defines the possible RGB Sequential Mode of GLCD.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

GLCD_SIGNAL_POLARITY Enumeration

Polarity values of different output signals.

File

plib_glcd_help.h

C

```
typedef enum {
  GLCD_POLARITY_POSITIVE = 0x00000000,
  GLCD_PIXEL_CLOCK_POLARITY_NEGATIVE = 0x00400000,
  GLCD_DE_POLARITY_NEGATIVE = 0x04000000,
  GLCD_HSYNC_POLARITY_NEGATIVE = 0x08000000,
  GLCD_VSYNC_POLARITY_NEGATIVE = 0x10000000
} GLCD_SIGNAL_POLARITY;
```

Members

Members	Description
GLCD_POLARITY_POSITIVE = 0x00000000	Positive polarity of PCLK, DE, HSYNC and VSYNC pins
GLCD_PIXEL_CLOCK_POLARITY_NEGATIVE = 0x00400000	Negative Pixel Clock Pin Polarity
GLCD_DE_POLARITY_NEGATIVE = 0x04000000	Negative DE Pin Polarity
GLCD_HSYNC_POLARITY_NEGATIVE = 0x08000000	Negative HSYNC Pin Polarity
GLCD_VSYNC_POLARITY_NEGATIVE = 0x10000000	Negative VSYNC Pin Polarity

Description

GLCD Output Signal Polarity

This data type defines the polarity values of different output signals.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

Files

Files

Name	Description
plib_glcd.h	Defines the Graphics LCD Controller (GLCD) Peripheral Library Interface.

Description

This section lists the source and header files used by the library.

plib_glcd.h

Defines the Graphics LCD Controller (GLCD) Peripheral Library Interface.

Functions

	Name	Description
	PLIB_GLCD_BackgroundColorGet	Gets the value of the Background Color.
	PLIB_GLCD_BackgroundColorSet	Sets the background color.
	PLIB_GLCD_BackPorchXGet	Gets X Axis Back Porch.
	PLIB_GLCD_BackPorchXYSet	Sets the back porch on the x and y axis for the Graphics LCD Controller.
	PLIB_GLCD_BackPorchYGet	Gets Y Axis Back Porch.
	PLIB_GLCD_BlankingXGet	Gets the X Axis Blanking Period.
	PLIB_GLCD_BlankingXYSet	Sets the blanking period on the x and y axis of the Graphics LCD Controller.
	PLIB_GLCD_BlankingYGet	Gets the Y Axis Blanking Period.
	PLIB_GLCD_ClockDividerGet	Gets Clock Divider value.
	PLIB_GLCD_ClockDividerSet	Sets clock controls of the Graphics LCD Controller.
	PLIB_GLCD_CursorDataGet	Gets the Cursor Data at given Index.
	PLIB_GLCD_CursorDataSet	Sets the cursor image data.
	PLIB_GLCD_CursorDisable	Disables the cursor of the Graphics LCD Controller.
	PLIB_GLCD_CursorEnable	Enables the cursor of the Graphics LCD Controller.
	PLIB_GLCD_CursorIsEnabled	Verifies whether the cursor is enabled.
	PLIB_GLCD_CursorLUTGet	Gets the color from Cursor LUT at given Index.
	PLIB_GLCD_CursorLUTSet	Sets the cursor color look-up table (LUT) in XRGB8888 format.
	PLIB_GLCD_CursorXGet	Gets the cursor X Axis Position.
	PLIB_GLCD_CursorXYSet	Sets the x and y coordinates of the Graphics LCD Controller cursor.
	PLIB_GLCD_CursorYGet	Gets the Cursor Y Axis Position.
	PLIB_GLCD_DESignalLevelGet	Gets the display enable signal level.
	PLIB_GLCD_Disable	Disables the Graphics LCD Controller.
	PLIB_GLCD_DitheringDisable	Disables the Dithering feature of the Graphics LCD Controller.
	PLIB_GLCD_DitheringEnable	Enables the Dithering feature of the Graphics LCD Controller.
	PLIB_GLCD_DitheringIsEnabled	Verifies whether Dithering is enabled.
	PLIB_GLCD_Enable	Enables the Graphics LCD Controller.
	PLIB_GLCD_ExistsBackgroundColor	Verify whether the Background Color Feature is supported.
	PLIB_GLCD_ExistsBackPorchXY	Verify whether Back Porch Feature is supported.
	PLIB_GLCD_ExistsBlankingXY	Verify whether Blanking Period Feature is supported.
	PLIB_GLCD_ExistsClockDivider	Verify whether the Clock Divider feature is supported.
	PLIB_GLCD_ExistsCursor	Verifies whether the cursor feature exists.
	PLIB_GLCD_ExistsCursorData	
	PLIB_GLCD_ExistsCursorLUT	Verify whether Cursor LUT feature is supported.
	PLIB_GLCD_ExistsCursorXY	Verify whether Cursor XY Position feature is supported.
	PLIB_GLCD_ExistsDESignalLevel	Verify whether DE Signal Level feature is supported.
	PLIB_GLCD_ExistsDithering	Verifies whether the Dithering feature exists.

	PLIB_GLCD_ExistsEnable	Identifies whether the GLCD Enable feature exists on the GLCD module.
	PLIB_GLCD_ExistsForceOutputBlank	Verify whether Force Output Blank feature is supported.
	PLIB_GLCD_ExistsFormattingClockDivide	Verify whether Clock Formatting feature is available.
	PLIB_GLCD_ExistsFrontPorchXY	Verify whether Front Porch Feature is supported.
	PLIB_GLCD_ExistsGlobalColorLUT	Verify whether Global Color LUT feature is supported.
	PLIB_GLCD_ExistsHSyncInterruptEnable	Verify whether the HSYNC Interrupt Enable feature is supported.
	PLIB_GLCD_ExistsHSyncSignalLevel	Verify whether HSYNC Signal Level feature is supported.
	PLIB_GLCD_ExistsIRQTriggerControl	Verify whether the IRQ Trigger Control feature is supported.
	PLIB_GLCD_ExistsIsLastRow	Verify whether Is Last Row Feature is supported.
	PLIB_GLCD_ExistsIsVerticalBlankingActive	Verify whether Is Vertical Blanking Active feature is supported.
	PLIB_GLCD_ExistsLayerBaseAddress	Verify whether the Layer Base Address feature is supported.
	PLIB_GLCD_ExistsLayerBilinearFilterEnable	Enable Layer Bilinear Filter Feature.
	PLIB_GLCD_ExistsLayerColorMode	Verify whether Layer Color Mode is supported.
	PLIB_GLCD_ExistsLayerDestBlendFunc	Verify whether Layer Destination Blend Function Feature is supported.
	PLIB_GLCD_ExistsLayerEnable	Verify whether Layer Enable Feature is supported.
	PLIB_GLCD_ExistsLayerForceWithGlobalAlpha	Verify whether Layer Force with Global Alpha Feature is supported.
	PLIB_GLCD_ExistsLayerGlobalAlpha	Verify whether Layer Global Alpha Feature is supported.
	PLIB_GLCD_ExistsLayerPreMultiplyImageAlpha	Verify whether Layer Pre-Multiply Image Alpha Feature is supported.
	PLIB_GLCD_ExistsLayerResXY	
	PLIB_GLCD_ExistsLayerSizeXY	Verify whether Layer X Axis and Y Axis Size feature is supported.
	PLIB_GLCD_ExistsLayerSrcBlendFunc	Verify whether Layer Source Blend Function Feature is supported.
	PLIB_GLCD_ExistsLayerStartXY	Verify whether Layer Start XY Position feature is supported.
	PLIB_GLCD_ExistsLayerStride	Verify whether the Layer Stride Feature is supported.
	PLIB_GLCD_ExistsLinesPrefetch	Verify whether Lines Prefetch Set Feature supported.
	PLIB_GLCD_ExistsPaletteGammaRamp	Verifies whether the palette / gamma ramp feature is supported.
	PLIB_GLCD_ExistsResolutionXY	Verify whether YUV Output feature is supported.
	PLIB_GLCD_ExistsRGBSequentialMode	Verify whether RGB Sequential Mode feature is supported.
	PLIB_GLCD_ExistsSignalPolarity	Verifies whether the Signal Polarity Selection feature exists.
	PLIB_GLCD_ExistsSingleCyclePerLineVsync	Verifies whether VSYNC on Single cycle Per Line Feature exists.
	PLIB_GLCD_ExistsVSyncInterruptEnable	Verify whether VSYNC Interrupt Enable feature is supported.
	PLIB_GLCD_ExistsVSyncSignalLevel	Verify whether VSYNC Signal Level feature is supported.
	PLIB_GLCD_ExistsYUVOutput	Verify whether YUV output feature is supported.
	PLIB_GLCD_ForceOutputBlankDisable	Disable Force Output Blank feature.
	PLIB_GLCD_ForceOutputBlankEnable	Enable Force Output Blank feature.
	PLIB_GLCD_ForceOutputBlankIsEnabled	Verify whether the Force Output Blank feature is enabled.
	PLIB_GLCD_FormattingClockDivideDisable	Disbale the Clock Formatting feature.
	PLIB_GLCD_FormattingClockDivideEnable	Enable the Clock Formatting feature.
	PLIB_GLCD_FormattingClockDivideIsEnabled	Verify whether Clock Formatting feature is enabled.
	PLIB_GLCD_FrontPorchXGet	Gets X Axis Front Porch.
	PLIB_GLCD_FrontPorchXYSet	Sets the front porch on the x and y axis for the Graphics LCD Controller.
	PLIB_GLCD_FrontPorchYGet	Gets Y Axis Front Porch.
	PLIB_GLCD_GlobalColorLUTGet	Gets the Color from Global Color LUT at given Index.
	PLIB_GLCD_GlobalColorLUTSet	Set Global Color LUT.
	PLIB_GLCD_HSyncInterruptDisable	Disables interrupts at Hsync.
	PLIB_GLCD_HSyncInterruptEnable	Enables interrupts at Hsync.
	PLIB_GLCD_HSyncInterruptIsEnabled	Verify whether HSYNC Interrupt is enabled.
	PLIB_GLCD_HSyncSignalLevelGet	Gets the Hsync signal level.
	PLIB_GLCD_IRQTriggerControlGet	Gets the IRQ Trigger Control value.
	PLIB_GLCD_IRQTriggerControlSet	Sets the IRQ trigger control.
	PLIB_GLCD_IsEnabled	Verifies whether the Graphics LCD Controller is Enabled.
	PLIB_GLCD_IsLastRow	Gets the status indicating whether a last row is currently displayed by the Graphics Display Controller.
	PLIB_GLCD_IsVerticalBlankingActive	Get the active status.
	PLIB_GLCD_LayerBaseAddressGet	Gets the Layer Base Address.
	PLIB_GLCD_LayerBaseAddressSet	Sets the base address of layer surface of the Graphics LCD Controller.

	PLIB_GLCD_LayerBilinearFilterDisable	Disables the layer Bilinear filter of the Graphics LCD Controller.
	PLIB_GLCD_LayerBilinearFilterEnable	Enables the layer Bilinear filter of the Graphics LCD Controller.
	PLIB_GLCD_LayerBilinearFilterIsEnabled	Verify whether Layer Bilinear Filter is enabled.
	PLIB_GLCD_LayerColorModeGet	Gets the Layer Color Mode.
	PLIB_GLCD_LayerColorModeSet	Sets the layer color mode of the Graphics LCD Controller.
	PLIB_GLCD_LayerDestBlendFuncGet	Gets the Layer Destination Blend Function.
	PLIB_GLCD_LayerDestBlendFuncSet	Sets the layer destination blend function of the Graphics LCD Controller.
	PLIB_GLCD_LayerDisable	Disables the layer of the Graphics LCD Controller.
	PLIB_GLCD_LayerEnable	Enables the layer of the Graphics LCD Controller.
	PLIB_GLCD_LayerForceWithGlobalAlphaDisable	Disables the Layer Force Global Alpha feature.
	PLIB_GLCD_LayerForceWithGlobalAlphaEnable	Enable the Layer Force Global Alpha feature.
	PLIB_GLCD_LayerForceWithGlobalAlphaIsEnabled	Verify whether Layer Force with Global Alpha Feature is enabled.
	PLIB_GLCD_LayerGlobalAlphaGet	Gets the Layer Global Alpha value.
	PLIB_GLCD_LayerGlobalAlphaSet	Sets the layer global alpha of the Graphics LCD Controller.
	PLIB_GLCD_LayerIsEnabled	Verify whether Layer is enabled.
	PLIB_GLCD_LayerPreMultiplyImageAlphaDisable	Disable Layer Pre-Multiply Image Alpha Feature.
	PLIB_GLCD_LayerPreMultiplyImageAlphaEnable	Enable Layer Pre-Multiply Image Alpha Feature.
	PLIB_GLCD_LayerPreMultiplyImageAlphaIsEnabled	Verify whether the Layer Pre-Multiply Image Alpha Feature is enabled.
	PLIB_GLCD_LayerResXGet	Gets the Layer X Axis Resolution.
	PLIB_GLCD_LayerResXYSet	Sets the layer resolution in pixels.
	PLIB_GLCD_LayerResYGet	Gets the Layer Y Axis Resolution.
	PLIB_GLCD_LayerSizeXGet	Gets the Layer X Axis Size.
	PLIB_GLCD_LayerSizeXYSet	Sets the layer size x and size y of the Graphics LCD Controller.
	PLIB_GLCD_LayerSizeYGet	Gets the Layer Y Axis Size.
	PLIB_GLCD_LayerSrcBlendFuncGet	Gets the Layer Source Blend Function.
	PLIB_GLCD_LayerSrcBlendFuncSet	Sets the layer source blend function of the Graphics LCD Controller.
	PLIB_GLCD_LayerStartXGet	Gets the Layer X Axis Start Position.
	PLIB_GLCD_LayerStartXYSet	Sets the layer start x and start y of the Graphics LCD Controller.
	PLIB_GLCD_LayerStartYGet	Gets the Layer Y Axis Start Position.
	PLIB_GLCD_LayerStrideGet	Gets the Layer Stride value.
	PLIB_GLCD_LayerStrideSet	Sets the layer surface stride of the Graphics LCD Controller.
	PLIB_GLCD_LinesPrefetchGet	Gets value of lines to be prefetched.
	PLIB_GLCD_LinesPrefetchSet	Sets the clock controls of the Graphics LCD Controller.
	PLIB_GLCD_PaletteGammaRampDisable	Disables the palette / gamma ramp of the Graphics LCD Controller.
	PLIB_GLCD_PaletteGammaRampEnable	Enables the palette / gamma ramp of the Graphics LCD Controller.
	PLIB_GLCD_PaletteGammaRampIsEnabled	Verify whether Palette / Gamma Ramp feature is enabled.
	PLIB_GLCD_ResolutionXGet	Gets X Axis Resolution.
	PLIB_GLCD_ResolutionXYSet	Sets the display resolution for the Graphics LCD Controller.
	PLIB_GLCD_ResolutionYGet	Gets Y Axis Resolution.
	PLIB_GLCD_RGBSequentialModeGet	Get the RGB Sequential Mode already set.
	PLIB_GLCD_RGBSequentialModeSet	Sets the RGB output sequential mode.
	PLIB_GLCD_SignalPolarityGet	Gets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.
	PLIB_GLCD_SignalPolaritySet	Sets the polarity of the pixel clock, DE, HSync and VSync pin of the Graphics LCD Controller.
	PLIB_GLCD_SingleCyclePerLineVsyncDisable	Clears VSYNC on single cycle per line.
	PLIB_GLCD_SingleCyclePerLineVsyncEnable	Sets VSYNC on single cycle per line.
	PLIB_GLCD_SingleCyclePerLineVsyncIsEnabled	Verify whether VSYNC on Single Cycle Per Line feature is enabled.
	PLIB_GLCD_VSyncInterruptDisable	Disables interrupts at Vsync.
	PLIB_GLCD_VSyncInterruptEnable	Enables interrupts at Vsync.
	PLIB_GLCD_VSyncInterruptIsEnabled	Verify whether VSYNC Interrupt is enabled.
	PLIB_GLCD_VSyncSignalLevelGet	Gets the Vsync signal level.
	PLIB_GLCD_YUVOutputDisable	Disables the output of the Graphics LCD Controller in YUV format.
	PLIB_GLCD_YUVOutputEnable	Enables the output of the Graphics LCD Controller in YUV format.
	PLIB_GLCD_YUVOutputIsEnabled	Verify whether the YUV Output feature is enabled.

Description

Graphics LCD Controller (GLCD) Peripheral Library Interface Header.

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Graphics LCD Controller Peripheral Library for Microchip micro-controllers. The definitions in this file are only for the directly addressable memory mapped Graphics LCD Controller module.

File Name

plib_glcd.h

Company

Microchip Technology Inc.

I2C Peripheral Library

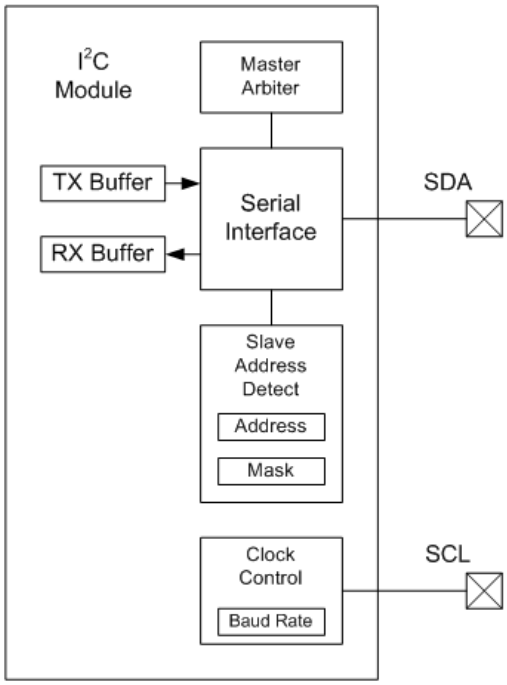
This section describes the Inter-Integrated Circuit (I2C) Peripheral Library.

Introduction

This library provides a low-level abstraction of the Inter-Integrated Circuit (I2C) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The I2C module is a serial interface useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, display drivers, analog-to-digital converters, etc.



Please refer to the I2C-Bus Specification (v2.1), available from Philips Semiconductor (see the following **Note**) for complete details on the operation and requirements for the I2C bus.

Note: Trademarks and Intellectual Property are property of their respective owners. Customers are responsible for obtaining appropriate licensing or rights before using this software. Refer to the MPLAB Harmony *Software License Agreement* for complete licensing information. A copy of this agreement is available in the `<install-dir>/doc` folder of your MPLAB Harmony installation.

Using the Library

This topic describes the basic architecture of the I2C Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_i2c.h`

The interface to the I2C library is defined in the `plib_i2c.h` header file, which is included by the peripheral library header file,`peripheral.h`. Any C language source (`.c`) file that uses the I2C library must include `peripheral.h`.

Peripheral Module IDs

Peripheral libraries are indexed to allow a single library to control any number of instances of a peripheral in a single microcontroller. To support this, the first parameter to each operation in a peripheral library is the module instance ID. The module instance ID is defined by an enumeration that is defined in the processor-specific header files (included by the library's interface header). For the I2C Peripheral Library, the module instance IDs are defined by the I2C_MODULE_ID enumeration. Not all microcontrollers will have all instances of the module listed in this enumeration. Please refer to the specific device data sheet for more information.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

Hardware Abstraction Model

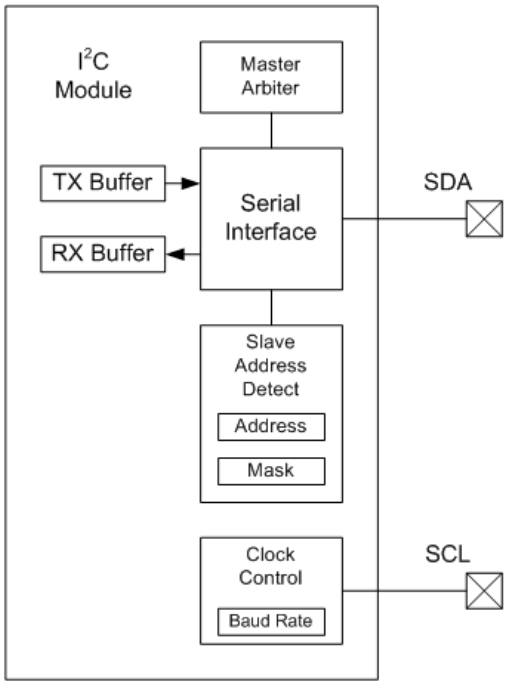
This library provides a low-level abstraction of the Inter-Integrated Circuit (I2C) module on Microchip PIC microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model Description

The I2C Peripheral Library provides interface routines to interact with the blocks shown in the following diagram.

Hardware Abstraction Model Diagram



The **TX Buffer** and **RX Buffer** are 1-byte (8-bit) buffers for data transmitted (TX) or received (RX) by the **Serial Interface** to the I2C bus over the **SDA** line synchronized with the **SCL** line by the **Clock Control** logic.

The desired baud rate frequency can be programmed in the **Clock Control** logic (within I2C bus specifications of 0-100 kHz for Low-to-Fast I2C mode and up to 3.4 MHz for High-Speed I2C mode) by setting the **Baud Rate** value. However, the nature of the I2C bus allows slower targets to stretch the clock to slower rates as required, in real time.

The I2C bus is a transfer-based multi-master bus, allowing multiple masters to initiate transfers at any time the bus is not currently busy. The **Master Arbiter** logic monitors transfers and determines if arbitration has been lost. There is no loss of data for the winning master. However, the losing master must retry the entire transfer again later, when the bus is idle. Slave devices are identified by either 7- or-10 bit addresses. The I2C module can act as either a bus master that can initiate transfers or as a slave, responding to transfers initiated by other masters on the bus.

The **Slave Address Detect** logic monitors transfers initiated by other masters to recognize when other masters have addressed this module when it is acting as a slave. The Slave **Address** is programmed into the I2C module by the software along with an (optional) address **Mask**, specifying bits in the address that can be ignored. This ability to "mask off" certain bits in the address allows the I2C module to identify multiple addresses and respond as if it were more than one slave device.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the I2C module.

Library Interface Section	Description
General Initialization Functions	Provides configuration routines for: • Slave Mode Clock Stretching • General Call and Other Reserved Address Support • System Management Bus (SMBus) support • High Frequency Support • Stop-in-Idle Control • Module Enable/Disable Control
General Status Functions	Provides status routines for I2C bus conditions.
Baud Rate Generator Control Functions	Provides configuration and access routines for baud rate control.
Slave Address Control Functions	Provides configuration, access, and status routines for control of the slave address to which the I2C module will respond when operating in Slave mode.
Slave Mask Control Functions	Provides configuration and access routines for control of the "ignore" bit mask for slave addressing. The "ignore" mask determines which bits in the incoming transfer's address are ignored, allowing the module to respond to multiple I2C addresses when operating in slave mode. **Note:** This feature is not available on all devices. Refer to the specific device data sheet determine whether slave address masking is supported for your device.
Slave Control Functions	Provides status and control routines for operating in slave mode (either as a receiver or as a transmitter).
Master Control Functions	Provides routines for control of the module when operating in master mode (either as a receiver or as a transmitter). **Note:** The necessary status routines are either part of the general, transmitter, or receiver groups depending on their purpose.
Transmitter Control Functions	Provides control, status, and data transfer routines for transmitting data (either as a master or a slave).
Receiver Control Functions	Provides control, status, and data transfer routines for receiving data (either as a master or a slave).
Feature Existence Functions	Determine whether particular features exist on a device.

How the Library Works

Provides information on how the library works.

Description

Before enabling the I2C module the caller must initialize basic configuration, clock frequency, and Slave address recognition features (see Initializing the I2C).

After the module has been enabled, it can begin monitoring the bus as a slave device waiting to be addressed by an external master (see Operating as a Slave Receiver). A slave device only transfers data on the bus when it has been addressed by a master (see Operating as a Slave Transmitter). If the module is used to start a transfer, it is operating as a master. A master addresses a slave and controls the transfer of data by providing the clock (see Operating as a Master Transmitter and Operating as a Master Receiver).

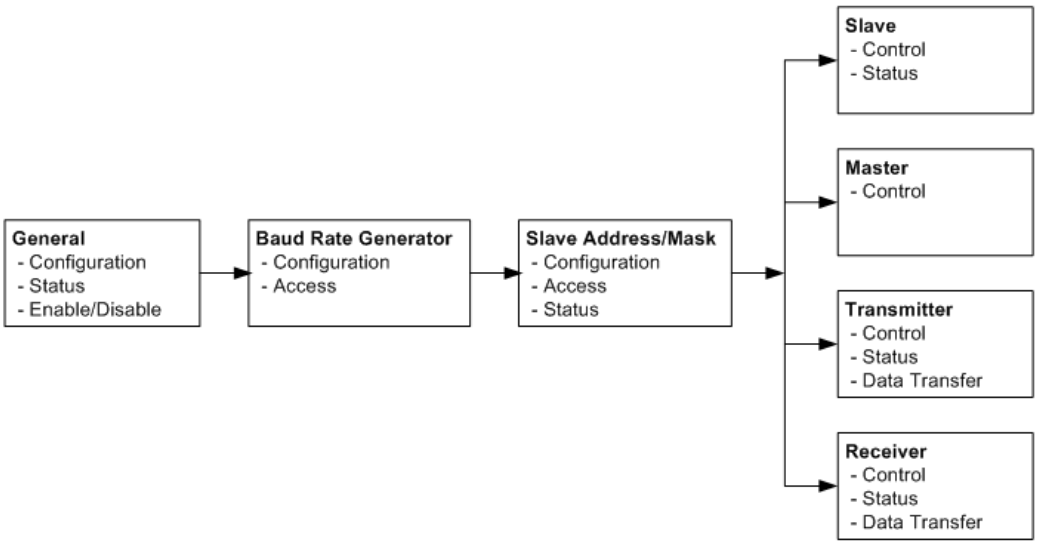
Some operations in the I2C library initiate actions on the I2C bus that require time to complete. Also, some events occur asynchronously on the bus. In each of these cases, the module causes either a "master", "slave", or "error" interrupt. Interrupt State Machine describes the different states that can cause an interrupt and show what causes the transition from one state to another. Handling Errors describes the various errors that can occur and how to deal with them.

Data transfers on the I2C bus must follow specific formats defined by the I2C bus protocol. (Refer to Forming Transfers for details)

Note: The I2C peripheral library does not directly respond to interrupts. The client software (usually a driver, middleware layer, or application) should implement the Interrupt Service Routine (ISR) and call the I2C library's interface routines from that ISR to manage the state of the I2C module.

Usage Model

The following diagram illustrates the library usage model for the I2C Peripheral Library.



Initializing the I2C

To initialize the I2C module, perform the following sequence:

1. Select the desired general configuration options using General Initialization Functions (see the Library Interface section) to select the desired operation of the following features:

- Slave Clock Stretching
- General Call Address Recognition
- System Management Bus (SMBus) Support
- High Frequency Operation
- Reserved Address Protection
- Stop-in-Idle Operation

2. Program the Baud Rate Generator to set the desired baud rate using the PLIB_I2C_BaudRateSet function.
3. If operating in slave mode, set the desired slave address using the PLIB_I2C_SlaveAddress7BitSet function (for 7-bit mode operation) or PLIB_I2C_SlaveAddress10BitSet function (for 10-bit mode operation).
4. If running in an interrupt-driven environment, clear any pending interrupts and enable the appropriate (master, slave, and error) I2C interrupts.

Note: Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for more information.

5. Enable the module for operation using the PLIB_I2C_Enable function.

Example

```
#define MY_I2C_ID  I2C_ID_1
uint32_t  actualClock;

// Configure General I2C Options
PLIB_I2C_SlaveClockStretchingEnable(MY_I2C_ID);
PLIB_I2C_GeneralCallEnable(MY_I2C_ID);
PLIB_I2C_SMBDisable(MY_I2C_ID);
PLIB_I2C_HighFrequencyDisable(MY_I2C_ID);
PLIB_I2C_ReservedAddressProtectEnable(MY_I2C_ID);
PLIB_I2C_StopInIdleEnable(MY_I2C_ID);

// Set Desired baud rate
PLIB_I2C_BaudRateSet ( MY_I2C_ID, MY_PERIPHEAL_CLOCK_FREQ, MY_I2C_BAUD_RATE);

// Set the appropriate slave address (slave only)
PLIB_I2C_SlaveAddress7BitSet(MY_I2C_ID, MY_SLAVE_ADDRESS);

// Optional:  Clear and enable interrupts before enabling the I2C module.

// Enable the module
```

```
PLIB_I2C_Enable(MY_I2C_ID);
```

Note: Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section referenced by that chapter for more information.

The previous example uses the following constants:

Symbol	Description
I2C_ID_1	Defined as the selected I2C module value from the I2C Module enumeration. Example: `#define MY_I2C_ID I2C_ID_1`
MY_I2C_PERIPHERAL_CLOCK	Defined as the peripheral clock frequency (in Hz) of the I2C module. Example: `#define MY_I2C_PERIPHERAL_CLOCK 80000000`
MY_I2C_BAUD_RATE	Defined as desired I2C baud rate. Example: #define MY_I2C_BAUD_RATE 100000
MY_SLAVE_ADDRESS	Defined as the desired 7-bit I2C slave address. Example: `#define 0x1A // Slave address 0011010` in binary **Note:** If '0' is passed as the mask value, the I2C module will only respond to single slave address passed in the second parameter.

Operating as a Slave Receiver

Before enabling the module, the module's "local" slave address and mask should be programmed using the PLIB_I2C_SlaveAddress7BitSet function or the PLIB_I2C_SlaveAddress10BitSet function, depending on the desired address width. If slave address masking is supported on the chosen processor the "don't care bits" mask can be set using the PLIB_I2C_SlaveMask7BitSet function or the PLIB_I2C_SlaveMask10BitSet function, also depending on the desired address width. Once the I2C module has been enabled, it will begin waiting for an I2C master to address it. When the programmed local slave address is recognized (ignoring any bits in the programmed mask value, if supported), the module will ACK the address and notify software that it has received an address byte. If the R/W bit of the address is 0, the module enters receive mode automatically. As additional bytes are received, the module will automatically ACK them if the previous byte has been read from the RX buffer before the new byte has been completely received and notify software of the newly available byte. Both address and data bytes received must be read from the RX buffer by software.

Preconditions

1. The module must have had its basic options and clock frequency initialized and have been enabled (see Initializing the I2C).
2. An external master must have started a transfer.
3. The address of the transfer must have matched with the address (ignoring any bits in the mask) programmed into the module by software.

Process

Software must follow this sequence:

1. Ensure that a byte has been received either by polling the PLIB_I2C_ReceivedByteIsAvailable function or by waiting for a slave interrupt (at which time, this function should still be checked).
2. Identify if the byte is a data byte or an address byte using the PLIB_I2C_SlaveAddressIsDetected function.
3. If the byte is an address byte, software must identify if it is starting a read or write operation using the PLIB_I2C_SlaveReadIsRequested function.
4. Software must read the byte from the RX buffer using the PLIB_I2C_ReceivedByteGet function, even if it was an address byte.
5. If clock stretching was enabled during initialization, software must call the PLIB_I2C_SlaveClockRelease function to release the SCL line.

Example

```
#define MY_I2C_ID  I2C_ID_1
uint8_t data;

if ( PLIB_I2C_ReceivedByteIsAvailable(MY_I2C_ID) )
{
    // Check to see if the byte is data or an address
    if (PLIB_I2C_SlaveAddressWasDetected(MY_I2C_ID))
    {
        if ( PLIB_I2C_SlaveReadWasRequested(MY_I2C_ID) )
        {
            // Beginning a slave receive sequence
        }
        else
        {
            // Beginning a slave transmit sequence
        }
    }
    else
    {
```

```
        // Data Byte Received
    }
    // Read the data or address from the RX buffer
    data = PLIB_I2C_ReceivedByteGet(MY_I2C_ID);

    // Free the SCL line (only if clock stretching was enabled)
    PLIB_I2C_SlaveClockRelease(MY_I2C_ID);
}
```

The previous example uses the following constants and variables:

Symbol	Description
MY_I2C_ID	Defined as the selected I2C module value from the I2C_MODULE_ID enumeration. Example: `#define MY_I2C_ID I2C_ID_1`
data	This is the variable used to hold the byte received.

Operating as a Slave Transmitter

Once an external master has addressed the I2C module (see Operating as a Slave Receiver) with the read-write bit set (as determined by calling the PLIB_I2C_SlaveReadIsRequested function), software can begin transmitting data to the master.

Preconditions

1. The module must have had its basic options and clock frequency initialized and have been enabled(see Initializing the I2C)
2. An external master must have started a transfer.
3. The address of the transfer must have matched with the address slave programmed into the I2C module (see Managing Slave Addresses).
4. A slave read must have been requested (as previously described).

Note: When the address has been received and matched the programmed local slave address, a slave interrupt will occur if it has been enabled.

Transmitting a Byte

When operating as a slave transmitter on the I2C bus, the software must use the following sequence to transmit data bytes.

1. Ensure that the TX buffer is currently ready to accept a byte to be transmitted using the PLIB_I2C_TransmitterIsReady function.
2. Write the byte to be transmitted to the TX buffer using the PLIB_I2C_TransmitterByteSend function.
3. If clock stretching was enabled (using the PLIB_I2C_SlaveClockStretchingEnable function) and the previous byte was ACKed, the PLIB_I2C_SlaveClockRelease function must be called to release the SCL line (it is safe to call this even if the previous byte was NAKed.)

Example

```
#define MY_I2C_ID  I2C_ID_1
uint8_t         data;

// Assign the desired byte value into the "data" variable

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    // Write the byte to the TX buffer
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, data);

     // Free the SCL line (only if clock stretching was enabled)
    PLIB_I2C_SlaveClockRelease(MY_I2C_ID);
}
```

Checking Acknowledgement

After a byte has been transmitted, the module latches the ACK/NAK response from the master so that software may use the following sequence to verify it and causes another slave interrupt (if it has been enabled).

1. Ensure that the byte has been completely transmitted using the PLIB_I2C_TransmitterByteHasCompleted function (and/or by responding to the slave interrupt).
2. Determine if the byte was ACKed or NAKed using the PLIB_I2C_TransmitterByteWasAcknowledged function.

Example

```
#define MY_I2C_ID  I2C_ID_1

if (PLIB_I2C_TransmitterByteHasCompleted(MY_I2C_ID))
{
    if (PLIB_I2C_TransmitterByteWasAcknowledged(MY_I2C_ID))
    {
        // Transmission successful
    }
```

```
    else
    {
        // Transmission was not successful
    }
}
```

The previous examples use the following constants and variables:

Symbol	Description
MY_I2C_ID	Defined as the selected I2C module value from the I2C_MODULE_ID enumeration. Example: `#define MY_I2C_ID I2C_ID_1`
data	This is the buffer containing byte of data to transmit.

Operating as a Master Transmitter

Whenever the I2C module initiates a transfer on the bus, it is acting as master. A transfer can be initiated any time the bus is idle. If two masters attempt to initiate transfers at nearly the same time, one of them will lose arbitration at some point during the transfer and must retry the transfer later when the bus becomes idle. The winner of arbitration can continue, without any data loss.

An I2C bus transfer always begins with a Start condition, followed by one or two properly formatted bytes containing the 7-bit or 10-bit target slave address (address of the slave device that is the target of the transfer). The direction of the remainder of transfers (after the address has been sent) is indicated by bit 0 of the address (0 = write/transmitting, 1 = read/receiving, from the point of view of the master). After that, any amount of data may be transferred and a repeated Start condition (followed by another properly formatted address) can occur to change the direction of the transfer or even the target slave device. See Forming Transfers for details of the possible transfer formats. When the master is finished, an I2C bus transfer always ends with a Stop condition. These steps are summarized as follows:

Summary of Steps

1. Send a Start condition.
2. Send a 7-bit or 10-bit address (one or two bytes with a write transfer indication in bit 0).
3. Send data bytes.
4. Send a Stop condition.

Each of these steps making up a transfer will take some time to complete. The completion of each step will cause an I2C master interrupt. The software must always keep track of which state it was in, check the I2C status, and respond to the interrupt by performing the next appropriate step.

Notes:

1. Arbitration loss can occur after each step has completed. Software should check for arbitration loss by calling the PLIB_I2C_ArbitrationLossHasOccurred function. If arbitration loss occurs, it must be cleared by calling the PLIB_I2C_ArbitrationLossClear function.
2. After any step completes, but before sending the Stop condition (see step 4), the master may send a repeated Start condition and start over at step 2 to change slave targets and/or transfer direction.

Sending a Start Condition

To start a master transfer, the software must do the following:

Preconditions

The module must have had its basic options and clock frequency initialized and have been enabled (see Initializing the I2C).

Process

1. Verify that the bus is idle using the PLIB_I2C_BusIsIdle function.
2. Send the start signal using the PLIB_I2C_MasterStart function.
3. Check for arbitration loss after the start condition has completed using the PLIB_I2C_ArbitrationLossHasOccurred function.

Example: Sending a Start Condition

```
#define MY_I2C_ID  I2C_ID_1

if (PLIB_I2C_BusIsIdle(MY_I2C_ID))
{
    PLIB_I2C_MasterStart(MY_I2C_ID);
}
```

The Start condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Start condition has been initiated. Even if the bus is verified to be idle before sending the start condition, a bus collision can still occur while the Start condition is being transmitted. To verify that arbitration has not been lost during transmission of the Start condition, the software should check the PLIB_I2C_ArbitrationLossHasOccurred function after the Start condition has completed.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID  I2C_ID_1

if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
```

```
    // Abort transfer, retry later

    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}
```

Sending a 7-bit Address

To send a 7-bit address, the software must do the following:

Preconditions

The Start condition must have been sent and completed as previously described.

Process

1. Write the address byte to the TX buffer using the PLIB_I2C_TransmitterByteSend function. The address byte that is supplied as the parameter to PLIB_I2C_TransmitterByteSend should also include the R/W bit.

Note: A master interrupt will occur once the address byte has been transmitted and ACKed or NAKed by the targeted slave device. (If the slave device does not answer, an implicit NAK will occur.)

2. Once the address byte has been ACKed or NAKed by the slave target, software must verify the result and respond appropriately. The Acknowledgement and Arbitration process is described in a subsequent section, just before the description of how to send a Stop condition.)

Example: Sending a 7-bit Address

```
#define MY_I2C_ID I2C_ID_1
#define SLAVE_TARGET_ADDRESS 0x22
uint8_t slaveAddress = 0;

// Format the 7-bit address byte with the write indication in bit 0
slaveAddress = SLAVE_TARGET_ADDRESS;

// Send the address byte
if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, slaveAddress);
}
```

Note: Bits 7-1 contain the 7-bit target slave address. Bit 0 indicates read or write direction for the remainder of the transfer.

Sending a 10-bit Target Slave Address

To send a 10-bit address, the software must do the following:

Preconditions

The Start condition must have been sent and completed as previously described.

Process

1. Format the two address bytes correctly with the 10-bit address and the read/write bit appropriately set (for a read transfer) or cleared (for a write transfer).
2. Ensure that the TX buffer is currently ready to accept a byte to be transmitted using the PLIB_I2C_TransmitterIsReady function.
3. Write the first address byte to the TX buffer using the PLIB_I2C_TransmitterByteSend function.

Note: After the first byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or you can poll the PLIB_I2C_TransmitterByteHasCompleted function).

4. Ensure that the first address byte was ACKed by the slave target (to do this, follow the Acknowledgement process described in a following section.
5. Write the second address byte to the TX buffer using the PLIB_I2C_TransmitterIsReady function and the PLIB_I2C_TransmitterByteSend function.

Note: After the second byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or you can poll the PLIB_I2C_TransmitterByteHasCompleted function).

6. Ensure that the second address byte was ACKed by the slave target (to do this, follow the process in the Acknowledgement and Arbitration, which is described later in this section).

Example: Formatting the 10-bit Address in Two Bytes

```
/* first byte of 10-bit address which is inclusive of R/W bit */
#define SLAVE_TARGET_10BIT_ADDRESS_BYTE1 0xF6

/* second byte of 10-bit address */
#define SLAVE_TARGET_10BIT_ADDRESS_BYTE2 0x22

uint8_t slaveAddressByte1 = 0;
uint8_t slaveAddressByte2 = 0;
```

```
/* include the R/W in bit 0 of the first byte */
slaveAddressByte1 = SLAVE_TARGET_10BIT_ADDRESS_BYTE1;
slaveAddressByte2 = SLAVE_TARGET_10BIT_ADDRESS_BYTE2;
```

Example: Sending the First Byte of a 10-bit Address

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, slaveAddressByte1);
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Example: Sending the Second Byte of a 10-bit Address

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, slaveAddressByte2);
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Sending a Data Byte

The following sequence can be used to send a data byte and repeated to send an arbitrary number of data bytes.

Preconditions

- The Start condition must have been sent and completed as previously described
- The target slave address has been fully transmitted (with the write indication in bit 0) and ACKed by the targeted slave device

Process

1. Ensure that the TX buffer is currently ready to accept a byte to be transmitted using the PLIB_I2C_TransmitterIsReady Function.
2. Write the byte to be transmitted to the TX buffer using the PLIB_I2C_TransmitterByteSend Function.

Note: After the byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or poll the PLIB_I2C_TransmitterByteHasCompleted function).

3. Check for arbitration loss or an ACK or NAK from the slave as described in the next section, Acknowledgement and Arbitration.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t         data;

// Assign the data value into the "data" variable

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, data);
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Acknowledgement and Arbitration

After sending a data or address byte (as previously described), the master transmitter must check to see if arbitration was lost and if the byte was ACKed or NAKed. If the byte was ACKed, the slave receiver is ready to receive more bytes. If the byte was NAKed, the slave receiver cannot accept any more bytes at this time.

Preconditions

- The start condition must have been sent and completed as previously described
- A byte (address or data) must have been transmitted, using one of the previous processes

Process

1. Ensure that the byte has been completely transmitted using the PLIB_I2C_TransmitterByteHasCompleted function (and/or by waiting for the master interrupt)
2. Determine if arbitration loss has occurred using the PLIB_I2C_ArbitrationLossHasOccurred function.
3. Determine if the byte was ACKed or NAKed using the PLIB_I2C_TransmitterByteWasAcknowledged function.

Example

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_TransmitterByteHasCompleted(MY_I2C_ID))
{
    if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
    {
```

```
        // Handle arbitration loss

        PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
    }
    else
    {
        if (PLIB_I2C_TransmitterByteWasAcknowledged(MY_I2C_ID))
        {
            // Transmission successful
        }
        else
        {
            // Transmission was not successful
        }
    }
}
```

To handle arbitration loss, the master transmitter must stop the transfer process and retry the entire transfer again at a later time when the bus is not busy.

Sending a Repeated Start Condition

If the master wants to change slave targets or transfer directions, it can send a repeated Start condition followed by a new address at any time after the previous target slave address has been acknowledged (and before the Stop condition has been sent).

Preconditions

- The Start condition must have been sent and completed as previously described
- A target slave address must have been transmitted and ACKed
- Zero or more data bytes may also have been sent and ACKed

Process

Send the repeated Start condition using the PLIB_I2C_MasterStartRepeat function.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStartRepeat(MY_I2C_ID);
```

The Start condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Start condition has been initiated.

The repeated Start must be followed by another target slave address (either 7-bit or 10-bit, as appropriate).

Sending a Stop Condition

When the master transmitter has finished sending data, it must end the transfer with a stop condition.

Preconditions

- The Start condition must have been sent and completed as previously described
- A target slave address must have been transmitted and ACKed
- Zero or more data bytes may also have been sent and ACKed

Note: At least one address byte must be transmitted between the Start and Stop conditions. Following a Start condition immediately with a Stop condition is a protocol error and can cause some slave devices to lock up and stop responding to the bus.

Process

Send a stop condition using the PLIB_I2C_MasterStop function.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStop(MY_I2C_ID);
```

The Stop condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Stop condition has been initiated.

After a Stop condition, the bus should be idle unless a master immediately starts a new transfer. Strictly speaking, a bus collision can still occur while the Stop condition is being transmitted. Although the transfer is already completed when the Stop condition has been sent, software does not have to abort the transfer; however, it should still check for and clear the arbitration loss condition if it occurs.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}
```

The previous examples use the following constants and variables:

Symbol	Description
`MY_I2C_ID`	Defined as the selected I^2C module value from the I2C Module enumeration. Example: `#define MY_I2C_ID I2C_ID_1`
`data`	This is the buffer containing byte of data to transmit.
`slaveTargetAddress7Bit`	This variable is used to demonstrate how to properly format and access a 7-bit target slave address
`SLAVE_TARGET_7BIT_ADDRESS`	This constant is used as a placeholder for the actual 7-bit target slave address that must be defined by the client code.
`slaveAddress10Bit`	This variable is used to demonstrate how to properly format and access a 10-bit slave address.
`SLAVE_TARGET_10BIT_ADDRESS`	This constant is used as a placeholder for the actual 10-bit slave address that must be defined by the client code.

Operating as a Master Receiver

Any time the I2C module initiates a transfer on the bus, it is acting as a master. A transfer can be initiated any time the bus is idle. If two masters attempt to initiate transfers at nearly the same time, one of them will lose arbitration and must retry the transfer later when the bus becomes idle. The winner of arbitration can continue, without any data loss.

An I2C bus transfer always begins with a Start condition, followed by a 7-bit or 10-bit (1-byte or 2-byte) target slave address. Therefore, every master transfer starts off transmitting at least one byte. The direction of transfer of subsequent bytes (after the first byte has been sent) is indicated by bit 0 of the first address byte (0 = write/transmitting, 1 = read/receiving, from the point of view of the master). If 10-bit addressing is used or additional data (such as an internal address) is to be transmitted after the first address byte, the first address byte must indicate a write transfer. To change the direction after transmitting more than just a single address byte, a repeated Start condition (followed by another single address byte) must be sent (see Forming Transfers for details of the possible transfer formats). When the master is finished, an I2C bus transfer always ends with a Stop condition. This results in two basic formats for receiving data as a master, summarized as follows:

Summary of Steps: Read Only

1. Send a Start condition.
2. Send a 7-bit address with a read transfer indication in bit 0.
3. Receive data bytes.
4. Send a Stop condition.

Summary of Steps: Combined Read-Write

1. Send a Start condition.
2. Send a 7-bit address (or the first byte of a 10-bit address) with a write transfer indication in bit 0.
3. Send the second byte of a 10-bit address (and/or any additional data to be transmitted).
4. Send a repeated Start condition.
5. Resend the first byte of the 10-bit address (or a 7-bit address) with a read indication in bit 0.
6. Receive data bytes.
7. Send a Stop condition.

Each of these steps making up a transfer will take some time to complete. The completion of each step will cause an I2C master interrupt. The software must always keep track of which state it was in and respond to the interrupt by performing the next appropriate step.

Sending a Start Condition

To start a master transfer, the software must do the following:

Preconditions

The module must have had its basic options and clock frequency initialized and have been enabled (see Initializing the I2C).

Process

1. Verify that the bus is idle using the PLIB_I2C_BusIsIdle function.
2. Send the start signal using the PLIB_I2C_MasterStart function.

Example

```
#define MY_I2C_ID  I2C_ID_1
if (PLIB_I2C_BusIsIdle(MY_I2C_ID))
{
    PLIB_I2C_MasterStart(MY_I2C_ID);
}
```

The Start condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Start condition has been initiated. Even if the bus is verified to be idle before sending the Start condition, a bus collision can still occur while the Start condition is being transmitted. To verify that arbitration has not been lost during transmission of the Start condition, the software should check the PLIB_I2C_ArbitrationLossHasOccurred function after the Start condition has completed.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID  I2C_ID_1
```

```
if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    // Abort transfer, retry later
    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}
```

Sending a 7-bit Address

To send a 7-bit address (or the first byte of a 10-bit address), the software must do the following:

Preconditions

The Start condition must have been sent and completed as previously described.

Process

1. Ensure that the TX buffer is currently ready to accept a byte to be transmitted using the PLIB_I2C_TransmitterIsReady function.
2. Format the address byte correctly with the 7-bit address and the read/write bit appropriately set (for a read transfer) or cleared (for a write transfer).
3. Write the address byte to the TX buffer using the PLIB_I2C_TransmitterByteSend function.

Note: A master interrupt will occur once the address byte has been transmitted and ACKed or NAKed by the targeted slave device (or if arbitration was lost). If the slave device does not answer, an implicit NAK will occur.

4. Once the address byte has been ACKed or NAKed by the slave target (or if arbitration was lost), software must verify the result and respond appropriately. The Acknowledgement and Arbitration process is described in a subsequent section, just before the description of how to send a Stop condition.)

Example: Sending a 7-bit Address

```
#define MY_I2C_ID  I2C_ID_1
#define SLAVE_TARGET_ADDRESS 0x22
uint8_t slaveAddress = 0;

// Format the 7-bit address byte with the read indication in bit 0
slaveAddress = (SLAVE_TARGET_ADDRESS | 0x01);

// Send the address byte
if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, slaveAddress);
}
```

Note: Bits 7-1 contain the 7-bit target slave address. Bit 0 indicates read or write direction for the remainder of the transfer.

Sending a 10-bit Target Slave Address

To send a 10-bit address, the software must do the following:

Preconditions

The Start condition must have been sent and completed as previously described.

Process

1. Format the two address bytes correctly with the 10-bit address and the read/write bit cleared, indicating a write transfer.
2. Ensure that the TX buffer is currently ready to accept a byte to be transmitted using the PLIB_I2C_TransmitterIsReady function.
3. Write the first address byte to the TX buffer using the PLIB_I2C_TransmitterByteSend function.

Note: After the first byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or poll the PLIB_I2C_TransmitterByteHasCompleted function).

4. Ensure that the first address byte was ACKed by the slave target and check for arbitration loss. To do this, use the Acknowledgement and Arbitration process, which described in the next section.
5. Write the second address byte to the TX buffer using the PLIB_I2C_TransmitterIsReady function and the PLIB_I2C_TransmitterByteSend function.

Note: After the second byte has been sent and ACKed or NAKed by the slave, a master interrupt will occur (or you can poll the PLIB_I2C_TransmitterByteHasCompleted function).

6. Ensure that the second address byte was ACKed by the slave target and that arbitration was not lost. To do this, use the Acknowledgement and Arbitration process, which described in the next section.

Example: Formatting the 10-bit Address in Two Bytes

```
#define MY_I2C_ID I2C_ID_1

/* first byte of 10-bit address which is inclusive of R/W bit */
#define SLAVE_TARGET_10BIT_ADDRESS_BYTE1 0xF6
```

```
/* second byte of 10-bit address */
#define SLAVE_TARGET_10BIT_ADDRESS_BYTE2 0x22

uint8_t slaveAddressByte1 = 0;
uint8_t slaveAddressByte2 = 0;

/* including the R/W bit in byte-1 of 10-bit address */
slaveAddressByte1 = (SLAVE_TARGET_10BIT_ADDRESS_BYTE1 | 0x01);

/* the second byte is included as is */
slaveAddressByte2 = SLAVE_TARGET_10BIT_ADDRESS_BYTE2;
```

Example: Sending the First Byte of a 10-bit Address

```
#define MY_I2C_ID I2C_ID_1
if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, slaveAddressByte1);
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Example: Sending the Second Byte of a 10-bit Address

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_TransmitterIsReady(MY_I2C_ID))
{
    PLIB_I2C_TransmitterByteSend(MY_I2C_ID, slaveAddressByte2);
}
```

Wait for transmission to complete and check for acknowledgement and/or arbitration loss.

Acknowledgement and Arbitration

After sending a data or address byte (as previously described), the master transmitter must check to see if arbitration was lost and if the byte was ACKed or NAKed. If the byte was ACKed, the slave receiver is ready to receive more bytes. If the byte was NAKed, the slave receiver cannot accept any more bytes at this time.

Preconditions

- The Start condition must have been sent and completed as previously described
- A byte (address or data) must have been transmitted, using one of the previous processes

Process

1. Ensure that the byte has been completely transmitted using the PLIB_I2C_TransmitterByteHasCompleted function (and/or by waiting for the master interrupt).
2. Determine if arbitration loss has occurred using the PLIB_I2C_ArbitrationLossHasOccurred function.
3. Determine if the byte was ACKed or NAKed using the PLIB_I2C_TransmitterByteWasAcknowledged function.

```
#define MY_I2C_ID I2C_ID_1
if (PLIB_I2C_TransmitterByteHasCompleted(MY_I2C_ID))
{
    if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
    {
        // Handle arbitration loss
        PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
    }
    else
    {
        if (PLIB_I2C_TransmitterByteWasAcknowledged(MY_I2C_ID))
        {
            // Transmission successful
        }
        else
        {
            // Transmission was not successful
        }
    }
}
```

To handle arbitration loss, the master must stop the transfer process and retry the entire transfer again at a later time when the bus is not busy.

Sending a Repeated Start Condition

If the master wants to change slave targets or transfer directions, it must send a repeated Start condition followed by a new address at any time after the previous target slave address has been acknowledged (and before the Stop condition has been sent).

Preconditions

- The Start condition must have been sent and completed as previously described
- A target slave address must have been transmitted and ACKed.
- Zero or more data bytes may also have been sent or received and ACKed as appropriate.

Process

Send the repeated Start condition using the PLIB_I2C_MasterStartRepeat function.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStartRepeat(MY_I2C_ID);
```

The repeated Start condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Start condition has been initiated. It must then be followed by another target slave address (either 7-bit or 10-bit, as appropriate). Strictly speaking, a bus collision can still occur while the repeated start condition is being transmitted. To verify that arbitration has not been lost during transmission of the repeated Start condition, the software should check the PLIB_I2C_ArbitrationLossHasOccurred function after the repeated Start condition has completed.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    // Abort transfer, retry later

    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}
```

Receiving a Data Byte

The following sequence can be used to receive a data byte and repeated to receive an arbitrary number of data bytes.

Preconditions

- The Start condition must have been sent and completed as previously described
- The target slave address must have been fully transmitted (with the read indication in bit 0) and ACKed by the targeted slave device

Process

1. Start the I2C module cycling the clock line to receive one byte by calling the PLIB_I2C_MasterReceiverClock1Byte function. The I2C module will generate a master interrupt once the eighth clock cycle has completed, but before the ninth cycle has started.
2. Ensure that a byte was received and is available in the RX buffer using the PLIB_I2C_ReceivedByteIsAvailable function.
3. If a byte is available, get it from the RX buffer using the PLIB_I2C_ReceivedByteGet function.
4. ACK or NAK the byte (as determined by software) using the PLIB_I2C_ReceivedByteAcknowledge function. The master receiver indicates to the slave transmitter that it wants to terminate the transfer by NAKing the last byte. All other bytes should be ACKed after reception.
5. Ensure that the ACK/NAK signal has completed using the PLIB_I2C_ReceiverByteAcknowledgeHasCompleted function or by waiting for the master interrupt.

Note: Arbitration loss cannot occur during a master reception (only during transmission).

Example: Enabling the Receiver

```
#define MY_I2C_ID I2C_ID_1
PLIB_I2C_MasterReceiverClock1Byte(MY_I2C_ID);
```

Wait for reception to complete before attempting to get the byte from the receiver.

Example: Reading a Byte from the RX Buffer

```
#define MY_I2C_ID I2C_ID_1
#define MY_MAX_COUNT  10
uint8_t data;
int     count;

if ( PLIB_I2C_ReceivedByteIsAvailable(MY_I2C_ID) )
{
    data = PLIB_I2C_ReceivedByteGet(MY_I2C_ID);
    count++;

    // Determine if the data should be ACKed or NAKed
    if (count < MY_MAX_COUNT)
    {
        PLIB_I2C_ReceivedByteAcknowledge(MY_I2C_ID, true);
    }
    else
    {
        PLIB_I2C_ReceivedByteAcknowledge(MY_I2C_ID, false);
    }
```

```
}
```

Wait for the ACK/NAK to complete by waiting for the next master interrupt.

Example: Verifying ACK/NAK Signal is Complete

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(MY_I2C_ID))
{
    // Byte reception sequence complete
}
```

Sending a Stop Condition

When the master transmitter has finished sending data, it must end the transfer with a Stop condition.

Preconditions

- The Start condition must have been sent and completed as previously described
- A target slave address must have been transmitted and ACKed.
- Zero or more data bytes may also have been sent or received and ACKed as appropriate

Note: At least one address byte must be transmitted between the Start and Stop conditions. Following a Start condition immediately with a Stop condition is a protocol error and can cause some slave devices to lock up and stop responding to the bus.

Process

Send a Stop condition using the PLIB_I2C_MasterStop function.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStop(MY_I2C_ID);
```

The Stop condition must be allowed to complete before any additional steps can be taken. The only way to determine this is to wait for a master interrupt to occur after the Stop condition has been initiated.

After a Stop condition, the bus should be idle unless a master immediately starts a new transfer. Strictly speaking, a bus collision can still occur while the stop condition is being transmitted. Although the transfer is already completed when the Stop condition has been sent, software does not have to abort the transfer; however, it should still check for and clear the arbitration loss condition if it occurs.

Example: Checking for Arbitration Loss

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_ArbitrationLossHasOccurred(MY_I2C_ID))
{
    PLIB_I2C_ArbitrationLossClear(MY_I2C_ID);
}
```

The previous example uses the following constants and variables:

Symbol	Description
MY_I2C_ID	Defined as the selected I2C module value from the I2C Module enumeration. Example: `#define MY_I2C_ID I2C_ID_1`
data	This is the buffer to hold byte of data received.
`slaveTargetAddress7Bit`	This variable is used to demonstrate how to properly format and access a 7-bit target slave address
`SLAVE_TARGET_7BIT_ADDRESS`	This constant is used as a place holder for the actual 7-bit target slave address that must be defined by the client code.
`slaveAddress10Bit`	This variable is used to demonstrate how to properly format and access a 10-bit target slave address
`SLAVE_TARGET_10BIT_ADDRESS`	This constant is used as a place holder for the actual 10-bit target slave address that must be defined by the client code.

Interrupt State Machine

The I2C state machine can be used in either a polled or an interrupt-driven manner. However, even when polled, software must check the state of the master, slave, and error interrupt flags to identify when a state transition occurs.

The I2C module can cause three different interrupts:

- Slave Interrupt - A byte was sent or received in Slave mode
- Master Interrupt - A bus action has completed in Master mode
- Error Interrupt - An error has occurred in any mode (see Handling Errors)

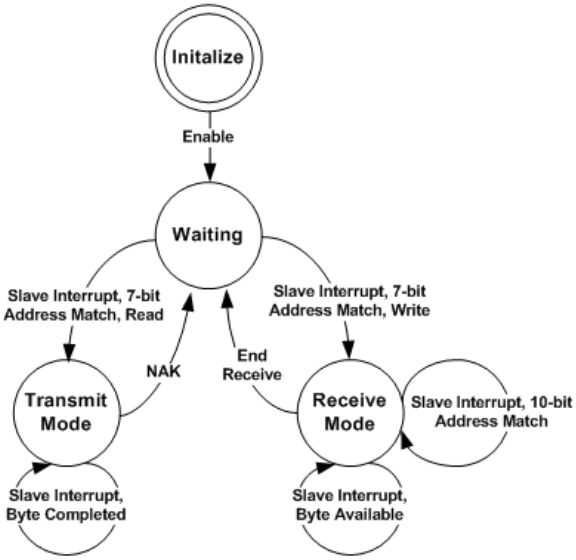
Note: Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for more information.

The I2C module will not start generating interrupts (setting the I2C interrupt flags, even if interrupts are disabled) until it is properly configured and enabled. After that, interrupts are generated as described in the following section. Software has to respond appropriately once the interrupt has occurred (the flag has been set) to allow the state machine to advance to the next state. These actions are described following the state machine diagram to which they refer.

Notes:

1. These state machine diagrams show the normal operations. Error and exception cases are explained in the Handling Errors section and are not shown in the diagrams.
2. The transition labels sometimes represent hardware interrupts (or the setting of an interrupt flag) and sometimes represent software actions. Refer to the transition tables to understand the transitions.
3. The Wait state is shared between all state machines (i.e., while in the Wait state, the I2C module can start a new master-mode transfer or be addressed by an external master).
4. The Start state is shared between both read and write master transfer state machines (i.e., when starting a new transfer, software can choose start either a read or a write transfer).

Slave Mode State Machine



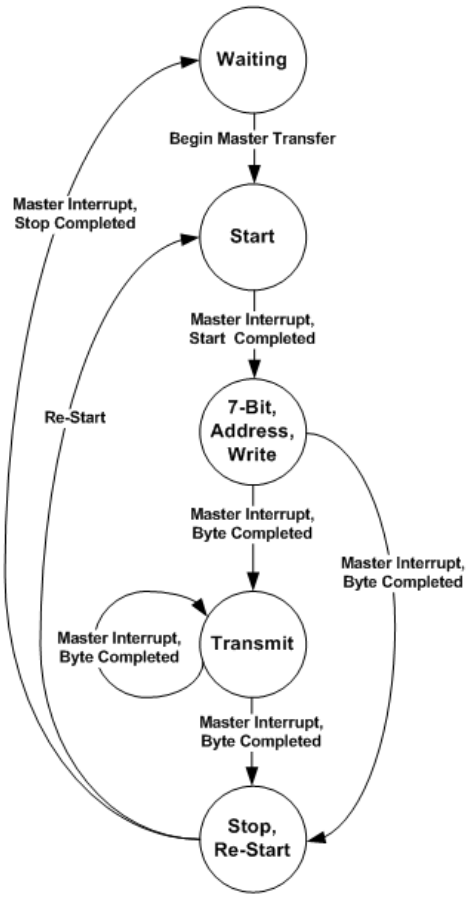
Slave Mode State Transitions

In Slave mode, state transitions occur under hardware control automatically or in response to an interrupt. Software must respond appropriately to ensure that the module continues to operate correctly.

Transition	From State	To State	Software Actions
Enable	Initialize	Wait	Enable the module, using the PLIB_I2C_Enable function
Slave Interrupt, 7-bit Address Match, Write	Wait	Receive mode	Software should call the PLIB_I2C_ReceivedByteIsAvailable function to verify that a byte has been received and the PLIB_I2C_SlaveAddressWasDetected function to verify that a 7-bit address (or the first byte of a 10-bit address) has been received. Then, software must get the address byte from the receiver buffer (the byte is automatically ACKed), using the PLIB_I2C_ReceivedByteGet function. Software must also release the clock by calling PLIB_I2C_SlaveClockRelease function to allow reception of the next byte. **Note:** This transition occurs on a write transfer with either a 7-bit or 10-bit address (for the first of two address bytes).
Slave Interrupt, Byte Received	Receive mode	Receive mode	Software should call the PLIB_I2C_ReceivedByteIsAvailable function to verify that a byte has been received and, if one has, get the byte from the receiver buffer by calling PLIB_I2C_ReceivedByteGet function. Then, software must release the clock by calling PLIB_I2C_SlaveClockRelease function to allow reception of the next byte.
Slave Interrupt, 10-bit Address Match	Receive mode	Receive mode	Software should call the PLIB_I2C_ReceivedByteIsAvailable function to verify that a byte has been received and (if 10-bit addressing is being used) call the PLIB_I2C_SlaveAddress10BitWasDetected function to verify that the second byte of the 10-bit address matched the low-order 8-bits of the local-slave 10-bit address. Next, software must get the address byte received from the receiver buffer (the byte is automatically ACKed), using the PLIB_I2C_ReceivedByteGet function. **Note:** This transition occurs on write transfer when using 10-bit (2-byte) addressing when the second address byte is received.

End Receive	Receive mode	Wait	None. Software is not aware of this transition. A receive transfer can end with a stop condition, a repeated start, or if the second-half of a 10-bit slave address does not match the programmed local slave address (and possibly the optional mask).
Slave Interrupt, 7-bit Address Match, Read	Wait	Transmit mode	Software should call the PLIB_I2C_ReceivedByteIsAvailable function to verify that a byte has been received and the PLIB_I2C_SlaveAddressWasDetected function to verify that an address has been received. Then, software must get the address byte from the receiver buffer (the byte is automatically ACKed), using the PLIB_I2C_ReceivedByteGet function. **Notes:** 1. This transition occurs on a read transfer with a 7-bit (1-byte) address or after a repeated-start with a 10-bit slave address. 2. The second byte of a 10-bit address is not sent after a repeated-start condition, only the first byte. The 10-bit slave device must remember if it was addressed at the beginning of this transfer. 3. After handling the address byte received, software should call the PLIB_I2C_TransmitterIsReady function to make sure that the transmitter buffer is able to accept a byte and, if it is, call the PLIB_I2C_TransmitterByteSend function to send the requested byte.
Slave Interrupt, Byte Completed	Transmit mode	Transmit mode	Software must call the PLIB_I2C_TransmitterByteWasAcknowledged function to check to see if the byte was ACKed or NAKed. If the byte was ACKed, software must send the next byte by calling the PLIB_I2C_TransmitterIsReady function to make sure that the transmitter buffer is able to accept a byte and, if it is, call the PLIB_I2C_TransmitterByteSend function to send the next byte.
NAK	Transmit mode	Wait	Software must call the PLIB_I2C_TransmitterByteWasAcknowledged function to check to see if the byte was ACKed or NAKed. The master will NAK the last byte it wants to receive. At that point, the transfer is effectively over and software must not attempt to send any more bytes. The module automatically transitions back to the Wait state when the Stop condition occurs, but software does not receive an interrupt at that point.

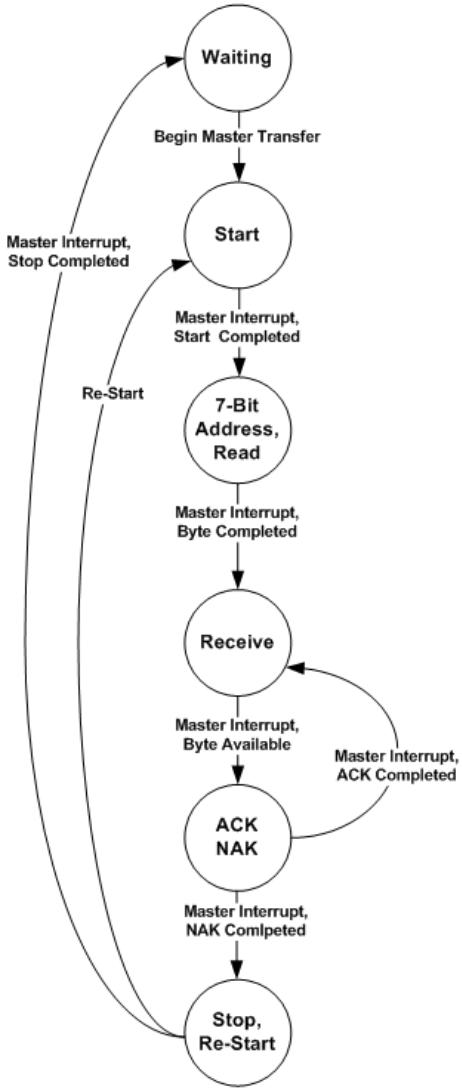
Master Mode Write Transfer State Machine



Master-Mode Write-Transfer State Transitions

Transition	From State	To State	Software Actions
Begin Master Transfer	Wait	Start	This state transition occurs when software initiates transmission of the start condition. Software should call the PLIB_I2C_BusIsIdle function to verify that the bus is not currently in use by another master. Then it must call the PLIB_I2C_MasterStart function (or PLIB_I2C_MasterStartRepeat function) to send the start (or repeated start) condition.
Master Interrupt, Start Completed	Start	7-Bit Address, Write	After the start condition has completed, software should check for a possible collision by calling the PLIB_I2C_ArbitrationLossHasOccurred function. Then, it must send an appropriately formatted 7-bit address byte (or the first byte of the 10-bit address) to the I^2C bus to address a slave target. To do this, it should first call the PLIB_I2C_TransmitterIsReadyfunction to verify that the transmitter buffer is ready to accept a byte and then call the PLIB_I2C_TransmitterByteSend function to send the byte.
Master Interrupt, Byte Completed	7-bit Address, Write	Transmit	When the master interrupt occurs (or the PLIB_I2C_TransmitterByteHasCompleted function returns true) to indicate that the last byte sent has completed, software must check to ensure that the slave device ACKed the byte by calling the PLIB_I2C_TransmitterByteWasAcknowledged function. Software should also call the PLIB_I2C_ArbitrationLossHasOccurred function to determine if a collision occurred. If all is well, software may then send another byte to the slave using the PLIB_I2C_TransmitterIsReady function and the PLIB_I2C_TransmitterByteSend function. If 10-bit addressing is being used for this transfer, the start condition and first byte of the 10-bit address must be followed by the second byte of the 10-bit address. Multiple slaves may respond to the first byte of the 10-bit address. It requires the second byte to uniquely identify the 10-bit slave. However, this is sent using the same process as if it were a data byte.
Master Interrupt, Byte Completed	Transmit	Transmit	Software must call the PLIB_I2C_TransmitterByteWasAcknowledged function check to ensure that the slave device ACKed the byte and the PLIB_I2C_ArbitrationLossHasOccurred function to determine if a collision occurred. If all is well, software may call the PLIB_I2C_TransmitterIsReady function and PLIB_I2C_TransmitterByteSendfunction to send another byte to the slave and remain in the Transmit state. If not, software should abort the transfer by sending a Stop condition and transition to the Stop, repeated Start state.
Master Interrupt, Byte Completed	7-bit Address, Write	Stop, repeated Start	If the slave NAKed any byte sent or if software wants to end or restart the transfer (perhaps to change direction) it can do so by sending a stop condition (using the PLIB_I2C_MasterStop function) or repeated Start condition (using the PLIB_I2C_MasterStartRepeat function).
Master Interrupt, Byte Completed	Transmit	Stop, repeated Start	Software must call the PLIB_I2C_TransmitterByteWasAcknowledged function check to check to see if the slave ACKed or NAKed the byte. If the slave NAKed the byte or if software wants to end or restart the transfer (perhaps to change direction) it can do so by sending a Stop condition (using the PLIB_I2C_MasterStop function) or repeated Start condition (using the PLIB_I2C_MasterStartRepeat function).
Restart	Stop, repeated Start	Start	Immediately after initiating the repeated Start condition, software transitions back to the Start state where it must decide what sort of transfer it will start next. **Note:** Read transfers will transition to the Start state in the Master Mode Read Transfer state machine.
Master Interrupt, Stop Completed	Stop, repeated Start	Wait	When the master interrupt occurs, indicating that the Stop condition has completed, software transitions back to the Wait state (which is actually part of the Slave Mode state machine).

Master-Mode Read-Transfer State Machine



Master Mode Read Transfer State Transitions

In Master mode, transitions are initiated under software control and interrupts occur (the master interrupt flag is set, even when interrupts are disabled) when the action initiated has completed.

If at any point, an address or data byte is NAKed by the slave or an error occurs, usually the safest thing for the master to do is to end the transfer with a Stop condition and release the bus.

Transition	From State	To State	Software Actions
Begin Master Transfer	Wait	Start	This state transition occurs when software initiates transmission of the start condition. Software should call the PLIB_I2C_BusIsIdle function to verify that the bus is not currently in use by another master. Then it must call the PLIB_I2C_MasterStart function (or PLIB_I2C_MasterStartRepeat function) to send the start (or repeated Start) condition.
Master Interrupt, Start Completed	Start	7-Bit Address, Read	After the start condition has completed, software should check for a possible collision by calling the PLIB_I2C_ArbitrationLossHasOccurred function. Then, it must send an appropriately formatted 7-bit address byte (or the first byte of the 10-bit address) to the I2C bus to address a slave target. To do this, it should first call the PLIB_I2C_TransmitterIsReady function to verify that the transmitter buffer is ready to accept a byte and then call the PLIB_I2C_TransmitterByteSend function to send the byte. **Note:** If using 10-bit addressing, this is a repeated transmission of the first byte of the 10-bit address since all 10-bit transfers must start off as write transfers (so both address bytes can be sent) then be restarted to become read transfers.
Master Interrupt, Byte Completed	7-Bit Address, Read	Receive	Software must initiate the "Clock1Byte" operation (using the PLIB_I2C_MasterReceiverClock1Byte function) so that the I2C module will cycle the SCL line eight times causing the slave to send the requested byte of data.

Master Interrupt, Byte Available	Receive	ACK NAK	When a byte has been received (which software can verify using the PLIB_I2C_ReceivedByteIsAvailable function), software must get the received byte from the RX buffer (using the PLIB_I2C_ReceivedByteGet function) and initiate transmission of the ACK/NAK bit (using the PLIB_I2C_ReceivedByteAcknowledge function). If it intends to receive another byte, software must ACK the byte just received. If it does not intend to receive another byte, software must NAK the byte just received.
Master Interrupt, ACK Completed	ACK, NAK	Receive	Software can verify that the ACK signal has completed using the PLIB_I2C_ReceiverByteAcknowledgeHasCompleted function. If software ACKed the previously received byte, it must initiate the "Clock1Byte" operation (using the PLIB_I2C_MasterReceiverClock1Byte function) so that the I2C module will cycle the SCL line eight times so that the slave can send the requested byte. Otherwise, software must have NAKed the previous byte to end the transfer.
Master Interrupt, NAK Completed	ACK, NAK	Stop, Restart	Software can verify that the NAK signal has completed using the PLIB_I2C_ReceiverByteAcknowledgeHasCompleted function. If software intends to continue the transfer without allowing the bus to go idle, it must initiate a repeated-start (using the PLIB_I2C_MasterStartRepeat function) and transition immediately to the Start state. If software intends to end the transfer and allow the bus to go idle, it must initiate a Stop condition using the PLIB_I2C_MasterStop function.
Restart	Stop, repeated Start	Start	Immediately after initiating the repeated Start condition, software automatically transitions back to the "Start" state where it must decide if it will restart with a read or write transfer. **Note:** Write transfers will transition to the Start state in the Master Mode Write Transfer state machine.
Master Interrupt, Stop Completed	Stop, repeated Start	Wait	When the master interrupt occurs, indicating that the Stop condition has completed, software transitions back to the waiting state (which is actually part of the Slave Mode state machine).

Managing Slave Addresses

An I2C address identifies which slave device is being targeted by a master device on a transfer-by-transfer basis. I2C addresses can be either 7-bits or 10-bits, as shown in the following sections, 7-Bit Slave Addresses and 10-Bit Slave Addresses.

Addresses that are 7-bits long are sent in a single byte where bit 0 identifies the direction of the remaining data in the transfer. Addresses that are 10-bits long are sent in two bytes using a reserved range in the 7-bit address space (addresses 0x78 through 0x7C) to encode the two high-order bits (bits A9 and A8) in the first byte. The second byte encodes the eight low-order bites (A7 through A0). These formats are in the following two sections.

7-Bit Slave Addresses

When using 7-bit slave addressing, the address byte is the first byte sent by any I2C master device immediately following a "Start" condition. The 7-bit address is shifted left by 1 bit to fill bits 7-to-1 in the address byte. Bit 0 in the 7-bit address byte indicates the transfer direction, as follows.

7-Bit Address Format

A6	A5	A4	A3	A2	A1	A0	R/$\overline{W}$

Bits	Usage
A6:A0	Bits 7 through 1 of the address byte contain the 7-bit slave address.
R/W	Bit 0 of the address byte is the read/write bit (0 = Read, 1 = Write).

See Operating as a Master Transmitter and Operating as a Master Receiver for examples.

10-Bit Slave Addresses

When using 10-bit slave addresses, the first address byte must be sent immediately following a start or repeated Start condition. The second byte must immediately follow the first byte when performing a write transfer. Read transfers must begin as write transfers so that both bytes may be sent. Immediately after sending the second address byte, the master will issue a repeated Start condition and resend the first address byte only (with the read/write bit cleared, indicating a read transfer. The second byte is not retransmitted. Since the transfer has been restarted, all listening 10-bit slave devices know that they will not be address again until the transfer has been ended with a Stop condition. However, devices that respond to 7-bit addresses might respond, so the first byte of the 10-bit address must be repeated to prevent that. The format of the 10-bit address is shown as follows.

10-Bit Address Format

1st Byte								2nd Byte							
1	1	1	1	0	A9	A8	R/$\overline{W}$	A7	A6	A5	A4	A3	A2	A1	A0

Bits	Usage
11110	Bits 7 through 3 of the first byte are constants. This guarantees that the first byte of the 10-bit address will fall within the reserved range of the 7-bit addresses.
A9:A8	Bits 2 through 1 of the first byte are the two high-order bits of the 10-bit address.

R/W	Bit 0 of the first byte is the Read/Write bit (0 = Read, 1 = Write).
A7:A0	The second byte contains the 8 low-order bits of the 10-bit address.

See Operating as a Master Transmitter and Operating as a Master Receiver for examples.

Operating in Slave Mode

In Slave mode, the "slave address" refers to the address (and optional mask) programmed into the I2C module to determine to which address or addresses the module will respond when addressed by an external master. The "raw" (unformatted) address is passed to or returned from the Managing Slave Addresses routines (without being formatted as previously shown). Likewise, the mask passed to or returned from the Slave Mask Control Functions (see the Library Interface section) is unformatted.

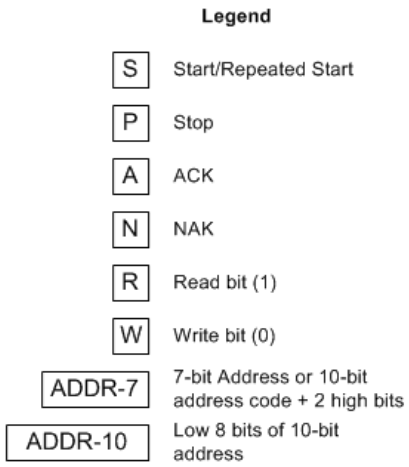
Operating in Master Mode

In Master mode, the "slave address" refers to the address transmitted on the I2C bus to identify the slave target of the transfer. In these cases, the slave address must be correctly formatted (as previously shown) and sent on the bus one byte at a time.

Forming Transfers

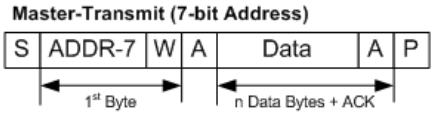
Forming Transfers

Transfers are formed by chaining the basic "building blocks" shown in the following transfer format legend.

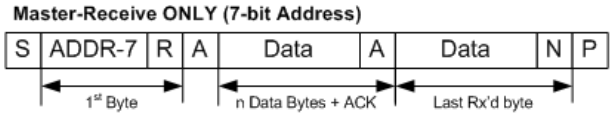


The following are all valid I2C transfer formats:

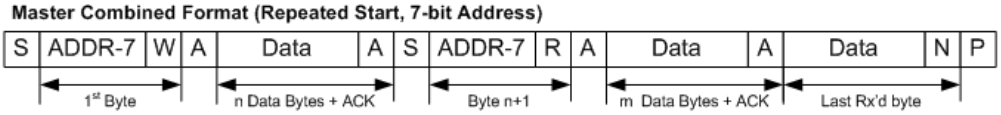
- This format allows a master to transmit *n* bytes of data to an I2C slave supporting a 7-bit address.



- This format allows a master to receive *n*+1 bytes of data from a "streaming" slave. Such a slave has no internal addressing capability. That is, it has no internal registers or addresses. It can only "stream" data to a master that reads from it.



- This format allows a master to write *n* bytes of data to a slave, followed immediately by reading *n*+1 bytes of data from a slave by sending a repeated Start condition after the first transfer. This format is useful for specifying an internal address to a device, and then reading data from that address. A repeated Start condition can also be used to chain transfers together without letting the bus go idle and potentially losing arbitration.

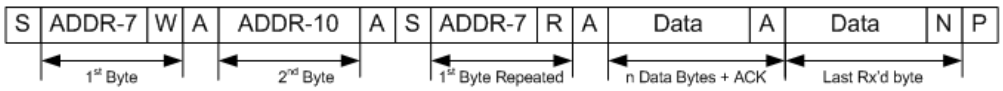


- The following format allows a master to send *n* bytes of data to a target supporting a 10-bit address.

Master Transmit (10-bit Address)

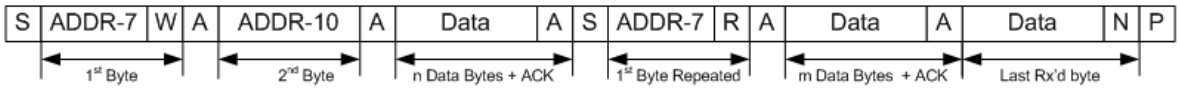
- This format is used to receive *n*+1 bytes of data from a slave supporting a 10-bit address. The repeated Start condition must be used because the second byte of the 10-bit address must be sent after the first byte of the 10-bit address. Then, the direction of the transfer must be switched to a "read". To do this, after the repeated Start, the master repeats the first byte of the 10-bit address with a "read" indication in the low-order bit and 10-bit slave devices must "remember" if they were being addressed once the transfer has started.

Master Receive (10-bit Address)



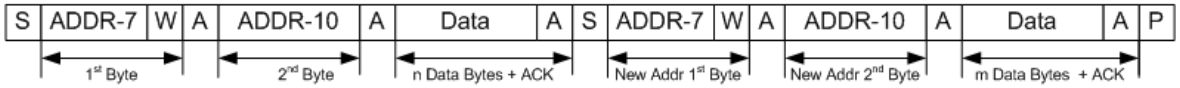
- This format allows a master to send *n* bytes of data to a slave device supporting a 10-bit address and immediately read *n*+1 bytes of data from the same device using the repeated Start and resending the first byte of the 10-bit address technique as shown in the previous format. This format is useful when the master needs to read data from a specific address within a device and it needs to send that address before performing the read.

Master Combined Format (10-bit write, 10-bit read)



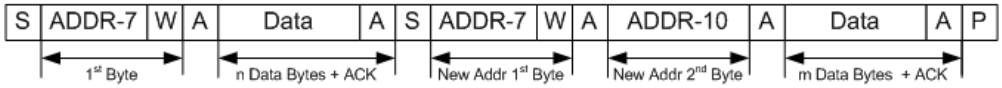
- This format allows a master to chain writes to multiple slaves supporting 10-bit addresses.

Master Combined Format (10-bit write, 10-bit write)



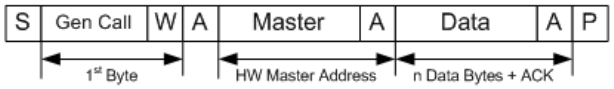
- This format allows a master to chain multiple writes to different slaves. Note that only the first write can be to a slave supporting a 7-bit address. The second (and any subsequent) writes must be to slaves with 10-bit addresses once any 10-bit device has been addressed.

Master Combined Format (7-bit write, 10-bit write)



- This format can be used by a "hardware" master that does not have a programmable address. This allows the hardware master to broadcast to all slaves on the bus using the "General Call" address, followed by the master's address. (The master address is generally the same as the device's slave address if it can also act as a slave device.) This format is not normally used by a microcontroller acting as a master. However, it may be received by a microcontroller if such a hardware master exists on the bus.

HW Master Transmit



Handling Errors

There are three basic types of errors that can occur during various I2C operations:

- Transmitter Overflow
- Receiver Overflow
- Arbitration Loss

Handling Transmitter Overflow Errors

A transmitter overflow error occurs when the software attempts to write to the TX buffer (by calling the PLIB_I2C_TransmitterByteSend function) while the transmitter is busy. When this occurs, the write is not allowed and the transmitter overflow status bit is set. This can be identified by calling the PLIB_I2C_TransmitterOverflowHasOccurred function. Additional attempts to write to the TX buffer will not be allowed until the transmitter overflow error is cleared by calling the PLIB_I2C_TransmitterOverflowClear function. This sort of error should be either avoided by checking the PLIB_I2C_TransmitterIsReady function before attempting to send a byte or by checking for the error after attempting to send a byte.

Interrupts: The transmitter overflow error does not trigger an interrupt.

Handling Receiver Overflow Errors

A receiver overflow error occurs when an incoming byte has been completely received and is ready to be transferred to the RX buffer, but software has not yet gotten the previous byte from the RX buffer. When this happens the receiver overflow status bit is set and the incoming byte will be lost. This can happen when software does not respond quickly enough to a slave or master interrupt when receiving data. Software should check for this error before calling the PLIB_I2C_ReceivedByteGet function to get a byte from the RX buffer to determine if any data has been lost. This error can be identified by calling the PLIB_I2C_ReceiverOverflowHasOccurred function. This error must be cleared by calling the PLIB_I2C_ReceiverOverflowClear function or no additional data will be received.

Interrupts: The receiver overflow error does not trigger an interrupt (although the incoming data byte does, even though it is lost.)

Handling Arbitration Loss Errors

Strictly speaking, arbitration loss is not an error. Rather it is a normal part of bus operation in a multi-master environment. Arbitration loss occurs when more than one master attempts to transmit on the bus at the same (or nearly the same) time. When this happens, at some point one of the masters will attempt to transmit a binary one ('1') when the other attempts to transmit a binary zero ('0'). At that point, the master transmitting the zero will win the transmission, because the zero transmission will dominate on the bus. The master detecting that the bus signals a zero when it attempted to transmit a one will lose arbitration and must immediately stop transmitting on the bus (which happens automatically).

Arbitration loss can be detected during any of the following actions:

- Transmission of a Start condition
- Transmission of a repeated Start condition
- Transmission of an address or data byte
- Transmission of an acknowledge bit
- Transmission of a Stop condition

Arbitration loss can be identified by calling the PLIB_I2C_ArbitrationLossHasOccurred function. The arbitration loss condition can be cleared by calling the PLIB_I2C_ArbitrationLossClear function. Once arbitration loss occurs, the losing master must wait until the bus is once again idle (at which time a master interrupt will occur if arbitration loss has occurred) and retransmit the transfer from the beginning.

Interrupts: The I2C module will trigger an error interrupt when arbitration loss occurs.

Other Features

Other Features

Note: The following features are not available on all devices. Please refer to the specific device data sheet to determine which features are supported by your device.

General Call Address Support

The I2C General Call address (reserved address 0) allows a master to address all slave devices on the bus. During slave operation the module can be made to respond to or ignore the general call address using the PLIB_I2C_GeneralCallEnable function or the PLIB_I2C_GeneralCallDisable function. When a slave address byte is received, software can identify if it was the general call address using the PLIB_I2C_SlaveAddressIsGeneralCall function (or by checking the value of the address byte received).

Note: The general call address is always 7-bits.

SMBus Support

The I2C module can support communication on a System Management Bus (SMBus) by calling the PLIB_I2C_SMBEnable function to adjust the electrical characteristics of the SMBus.

Operation During Sleep Mode

When the device enters Sleep mode, all clock sources supplied to the I2C modules are shut down. If the device enters Sleep mode in the middle of an I2C master transmission or reception operation, the operation is aborted. If the device enters Sleep mode when the module is operating in slave mode, the external master's clock will continue run the slave state machine. If a slave address match occurs, an interrupt will be provided to wake up the device.

Note: Indeterminate results can occur if an error occurs during Sleep mode.

Operation During Idle Mode

When the device enters Idle mode, the system clock sources remain functional and the CPU stops code execution.

I2C operation during Idle mode can be controlled using the PLIB_I2C_StopInIdleEnable function or the PLIB_I2C_StopInIdleDisable function.

By default, the I2C module continues to operate in Idle mode and provide full functionality.

Operation with DMA

Software activity must occur between the transfer of each byte, making DMA inappropriate for use with the I2C module.

Configuring the Library

This library is appropriate for microcontrollers with a dedicated I2C module (not an SSP or MSSP module). The library is configured for the supported I2C module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Initialization Functions

	Name	Description
	PLIB_I2C_Disable	Disables the specified I2C module.
	PLIB_I2C_Enable	Enables the specified I2C module.
	PLIB_I2C_GeneralCallDisable	Disables the I2C module from recognizing the general call address.
	PLIB_I2C_GeneralCallEnable	Enables the I2C module to recognize the general call address.
	PLIB_I2C_HighFrequencyDisable	Disables the I2C module from using high frequency (400 kHz or 1 MHz) signaling.
	PLIB_I2C_HighFrequencyEnable	Enables the I2C module to use high frequency (400 kHz or 1 MHz) signaling.
	PLIB_I2C_IPMIDisable	Disables the I2C module's support for the IPMI specification
	PLIB_I2C_IPMIEnable	Enables the I2C module to support the Intelligent Platform Management Interface (IPMI) specification (see Remarks).
	PLIB_I2C_ReservedAddressProtectDisable	Disables the I2C module from protecting reserved addresses, allowing it to respond to them.
	PLIB_I2C_ReservedAddressProtectEnable	Enables the I2C module to protect (not respond to) reserved addresses.
	PLIB_I2C_SlaveClockStretchingDisable	Disables the I2C module from stretching the slave clock.
	PLIB_I2C_SlaveClockStretchingEnable	Enables the I2C module to stretch the slave clock.
	PLIB_I2C_SMBDisable	Disable the I2C module support for SMBus electrical signaling levels.
	PLIB_I2C_SMBEnable	Enables the I2C module to support System Management Bus (SMBus) electrical signaling levels.
	PLIB_I2C_StopInIdleDisable	Disables the Stop-in-Idle feature.
	PLIB_I2C_StopInIdleEnable	Enables the I2C module to stop when the processor enters Idle mode

b) General Status Functions

	Name	Description
	PLIB_I2C_ArbitrationLossClear	Clears the arbitration loss status flag
	PLIB_I2C_ArbitrationLossHasOccurred	Identifies if bus arbitration has been lost.
	PLIB_I2C_BusIsIdle	Determines whether the I2C bus is idle or busy.
	PLIB_I2C_StartClear	Clears the start status flag
	PLIB_I2C_StartWasDetected	Identifies when a Start condition has been detected.
	PLIB_I2C_StopClear	Clears the stop status flag
	PLIB_I2C_StopWasDetected	Identifies when a Stop condition has been detected

c) Baud Rate Generator Control Functions

	Name	Description
	PLIB_I2C_BaudRateGet	Calculates the I2C module's current SCL clock frequency.
	PLIB_I2C_BaudRateSet	Sets the desired baud rate.

d) Slave Address Control Functions

	Name	Description
	PLIB_I2C_SlaveAddress10BitGet	Identifies the current 10-bit Slave mode address.
	PLIB_I2C_SlaveAddress10BitSet	Selects 10-bit Slave mode addressing and sets the address value.
	PLIB_I2C_SlaveAddress10BitWasDetected	Detects reception of the second byte of a 10-bit slave address.
	PLIB_I2C_SlaveAddress7BitGet	Identifies the current 7-bit Slave mode address.
	PLIB_I2C_SlaveAddress7BitSet	Selects 7-bit Slave mode addressing and sets the slave address value.
	PLIB_I2C_SlaveAddressHoldDisable	Disables Address holding.
	PLIB_I2C_SlaveAddressHoldEnable	Enables address holding.
	PLIB_I2C_SlaveAddressIsDetected	Detects if the most recent byte received is a data or an address byte.
	PLIB_I2C_SlaveAddressIsGeneralCall	Identifies if the current slave address received is the general call address.

	PLIB_I2C_SlaveAddressModeIs10Bits	Identifies if the current slave address mode is 7-bits or 10-bits.
	PLIB_I2C_SlaveDataIsDetected	Detects if the most recent byte received is a data or an address byte.
	PLIB_I2C_SlaveReadIsRequested	Detects if the request from the master was a read or write.

e) Slave Mask Control Functions

	Name	Description
	PLIB_I2C_SlaveMask10BitGet	Identifies the current 10-bit Slave mode address mask.
	PLIB_I2C_SlaveMask10BitSet	Sets the 10-bit mask for Slave mode addressing.
	PLIB_I2C_SlaveMask7BitGet	Identifies the current 7-bit Slave mode address mask.
	PLIB_I2C_SlaveMask7BitSet	Sets the 7-bit mask for Slave mode addressing .

f) Slave Control Functions

	Name	Description
	PLIB_I2C_AcksequenceIsInProgress	Checks whether the acknowledge sequence is in progress or it is completed.
	PLIB_I2C_DataLineHoldTimeSet	Sets the data line hold time.
	PLIB_I2C_SlaveBufferOverwriteDisable	Disables buffer overwrite.
	PLIB_I2C_SlaveBufferOverwriteEnable	Enables buffer overwrite.
	PLIB_I2C_SlaveBusCollisionDetectDisable	Disables bus collision detect interrupts.
	PLIB_I2C_SlaveBusCollisionDetectEnable	Enables slave bus collision interrupts.
	PLIB_I2C_SlaveClockHold	Holds the SCL clock line low after receiving a byte.
	PLIB_I2C_SlaveClockRelease	Releases a previously held SCL clock line.
	PLIB_I2C_SlaveDataHoldDisable	Disables data holding.
	PLIB_I2C_SlaveDataHoldEnable	Enables data holding.
	PLIB_I2C_SlaveInterruptOnStartDisable	Disables the feature of generating interrupt on start condition.
	PLIB_I2C_SlaveInterruptOnStartEnable	Enables the feature of generating interrupt on start condition.
	PLIB_I2C_SlaveInterruptOnStopDisable	Disables the feature of generating interrupt on stop condition.
	PLIB_I2C_SlaveInterruptOnStopEnable	Enables the feature of generating interrupt on stop condition.

g) Master Control Functions

	Name	Description
	PLIB_I2C_MasterReceiverClock1Byte	Drives the bus clock to receive 1 byte of data from a slave device.
	PLIB_I2C_MasterStart	Sends an I2C Start condition on the I2C bus in Master mode.
	PLIB_I2C_MasterStartRepeat	Sends a repeated Start condition during an ongoing transfer in Master mode.
	PLIB_I2C_MasterStop	Sends an I2C Stop condition to terminate a transfer in Master mode.

h) Transmitter Control Functions

	Name	Description
	PLIB_I2C_TransmitterByteHasCompleted	Detects whether the module has finished transmitting the most recent byte.
	PLIB_I2C_TransmitterByteSend	Sends a byte of data on the I2C bus.
	PLIB_I2C_TransmitterByteWasAcknowledged	Determines whether the most recently sent byte was acknowledged.
	PLIB_I2C_TransmitterIsBusy	Identifies if the transmitter of the specified I2C module is currently busy (unable to accept more data).
	PLIB_I2C_TransmitterIsReady	Detects if the transmitter is ready to accept data to transmit.
	PLIB_I2C_TransmitterOverflowClear	Clears the transmitter overflow status flag.
	PLIB_I2C_TransmitterOverflowHasOccurred	Identifies if a transmitter overflow error has occurred.

i) Receiver Control Functions

	Name	Description
	PLIB_I2C_ReceivedByteAcknowledge	Allows a receiver to acknowledge a that a byte of data has been received.
	PLIB_I2C_ReceivedByteGet	Gets a byte of data received from the I2C bus interface.
	PLIB_I2C_ReceivedByteIsAvailable	Detects whether the receiver has data available.
	PLIB_I2C_ReceiverByteAcknowledgeHasCompleted	Determines if the previous acknowledge has completed.
	PLIB_I2C_ReceiverOverflowClear	Clears the receiver overflow status flag.
	PLIB_I2C_ReceiverOverflowHasOccurred	Identifies if a receiver overflow error has occurred.
	PLIB_I2C_MasterReceiverReadyToAcknowledge	Checks whether the hardware is ready to acknowledge.

j) Feature Existence Functions

	Name	Description
	PLIB_I2C_ExistsAcksequenceProgress	Identifies whether the AcksequenceIsInProgress feature exists on the I2C module.
	PLIB_I2C_ExistsArbitrationLoss	Identifies whether the ArbitrationLoss feature exists on the I2C module.
	PLIB_I2C_ExistsBaudRate	Identifies whether the BaudRate feature exists on the I2C module.
	PLIB_I2C_ExistsBusIsIdle	Identifies whether the BusIdle feature exists on the I2C module.
	PLIB_I2C_ExistsClockStretching	Identifies whether the ClockStretching feature exists on the I2C module.
	PLIB_I2C_ExistsDataLineHoldTime	Identifies whether the DataLineHoldTime feature exists on the I2C module.
	PLIB_I2C_ExistsGeneralCall	Identifies whether the GeneralCall feature exists on the I2C module.
	PLIB_I2C_ExistsGeneralCallAddressDetect	Identifies whether the GeneralCallAddressDetect feature exists on the I2C module.
	PLIB_I2C_ExistsHighFrequency	Identifies whether the HighFrequency feature exists on the I2C module.
	PLIB_I2C_ExistsIPMI	Identifies whether the IPMI feature exists on the I2C module.
	PLIB_I2C_ExistsMasterReceiverClock1Byte	Identifies whether the MasterReceiverClock1Byte feature exists on the I2C module.
	PLIB_I2C_ExistsMasterStart	Identifies whether the MasterStart feature exists on the I2C module.
	PLIB_I2C_ExistsMasterStartRepeat	Identifies whether the MasterStartRepeat feature exists on the I2C module.
	PLIB_I2C_ExistsMasterStop	Identifies whether the MasterStop feature exists on the I2C module.
	PLIB_I2C_ExistsModuleEnable	Identifies whether the ModuleEnable feature exists on the I2C module.
	PLIB_I2C_ExistsReceivedByteAcknowledge	Identifies whether the ReceivedByteAcknowledge feature exists on the I2C module.
	PLIB_I2C_ExistsReceivedByteAvailable	Identifies whether the ReceivedByteAvailable feature exists on the I2C module
	PLIB_I2C_ExistsReceivedByteGet	Identifies whether the ReceivedByteGet feature exists on the I2C module
	PLIB_I2C_ExistsReceiverOverflow	Identifies whether the ReceiverOverflow feature exists on the I2C module.
	PLIB_I2C_ExistsReservedAddressProtect	Identifies whether the ReservedAddressProtect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddress10Bit	Identifies whether the SlaveAddress10Bit feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddress7Bit	Identifies whether the SlaveAddress7Bit feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddressDetect	Identifies whether the SlaveAddressDetect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddressHoldEnable	Identifies whether the SlaveAddressHoldEnable feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveBufferOverwrite	Identifies whether the SlaveBufferOverwrite feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveBusCollisionDetect	Identifies whether the SlaveBusCollisionDetect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveClockHold	Identifies whether the SlaveClockHold feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveDataDetect	Identifies whether the SlaveDataDetect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveInterruptOnStart	Identifies whether the SlaveInterruptOnStart feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveInterruptOnStop	Identifies whether the SlaveInterruptOnStop feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveMask	Identifies whether the SlaveMask feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveReadRequest	Identifies whether the SlaveReadRequest feature exists on the I2C module.
	PLIB_I2C_ExistsSMBus	Identifies whether the SMBus feature exists on the I2C module.
	PLIB_I2C_ExistsStartDetect	Identifies whether the StartDetect feature exists on the I2C module.
	PLIB_I2C_ExistsStopDetect	Identifies whether the StopDetect feature exists on the I2C module.
	PLIB_I2C_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterByteAcknowledge	Identifies whether the TransmitterByteAcknowledge feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterByteComplete	Identifies whether the TransmitterByteComplete feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterByteSend	Identifies whether the TransmitterByteSend feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterIsBusy	Identifies whether the TransmitterBusy feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterOverflow	Identifies whether the TransmitterOverflow feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveDataHoldEnable	Identifies whether the SlaveDataHoldEnable feature exists on the I2C module.

k) Data Types and Constants

	Name	Description
	I2C_MODULE_ID	This is type I2C_MODULE_ID.

Description

This section describes the Application Programming Interface (API) functions of the I2C Peripheral Library.

Refer to each section for a detailed description.

a) General Initialization Functions

PLIB_I2C_Disable Function

Disables the specified I2C module.

File

plib_i2c.h

C

```
void PLIB_I2C_Disable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the specified I2C module.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsModuleEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_Disable( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_Disable(I2C_MODULE_ID index)

PLIB_I2C_Enable Function

Enables the specified I2C module.

File

plib_i2c.h

C

```
void PLIB_I2C_Enable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the specified I2C module.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsModuleEnable in your application to determine whether this feature is available.

Preconditions

The module should be appropriately configured before being enabled.

Example

```
#define MY_I2C_ID               I2C_ID_1
#define MY_CLOCK_FREQUENCY      80000000
#define MY_BAUD_RATE            10000
#define MY_SLAVE_ADDRESS        0x23

PLIB_I2C_SlaveClockStretchingEnable ( MY_I2C_ID );
PLIB_I2C_SMBDisable ( MY_I2C_ID );
PLIB_I2C_HighFrequencyDisable ( MY_I2C_ID );
PLIB_I2C_ReservedAddressProtectEnable ( MY_I2C_ID );
PLIB_I2C_StopInIdleDisable ( MY_I2C_ID );
PLIB_I2C_BaudRateSet ( MY_I2C_ID, MY_CLOCK_FREQUENCY, MY_BAUD_RATE );

PLIB_I2C_SlaveAddress7BitSet( MY_I2C_ID, MY_SLAVE_ADDRESS );
PLIB_I2C_Enable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_Enable(I2C_MODULE_ID index)

PLIB_I2C_GeneralCallDisable Function

Disables the I2C module from recognizing the general call address.

File

plib_i2c.h

C

```
void PLIB_I2C_GeneralCallDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the I2C module from recognizing the general call address when operating as a slave receiver.

Remarks

The General-call feature can be re-enabled by calling the PLIB_I2C_GeneralCallEnable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsGeneralCall in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_GeneralCallDisable(MY_I2C_ID);
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_GeneralCallDisable (I2C_MODULE_ID index)

PLIB_I2C_GeneralCallEnable Function

Enables the I2C module to recognize the general call address.

File

plib_i2c.h

C

```
void PLIB_I2C_GeneralCallEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to recognize the general call address when operating as a slave receiver.

Remarks

The General-call feature can be disabled by calling the PLIB_I2C_GeneralCallDisable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsGeneralCall in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_GeneralCallEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_GeneralCallEnable (I2C_MODULE_ID index)

PLIB_I2C_HighFrequencyDisable Function

Disables the I2C module from using high frequency (400 kHz or 1 MHz) signaling.

File

plib_i2c.h

C

```
void PLIB_I2C_HighFrequencyDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the I2C module from using high-frequency signaling, preventing it from using the 400 kHz and 1 MHz signaling rates.

Remarks

The high-frequency feature can be re-enabled by calling the PLIB_I2C_HighFrequencyEnable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsHighFrequency in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_HighFrequencyDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_HighFrequencyDisable (I2C_MODULE_ID index)

PLIB_I2C_HighFrequencyEnable Function

Enables the I2C module to use high frequency (400 kHz or 1 MHz) signaling.

File

plib_i2c.h

C

```
void PLIB_I2C_HighFrequencyEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to use high-frequency signaling, allowing it to use the 400 kHz and 1 MHz signaling rates.

Remarks

The high-speed feature can be disabled by calling the PLIB_I2C_HighSpeedEnable function.

This feature must be enabled if frequencies higher than 100 kHz programmed using the PLIB_I2C_BaudRateSet function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsHighFrequency in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_HighFrequencyEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_HighFrequencyEnable (I2C_MODULE_ID index)

PLIB_I2C_IPMIDisable Function

Disables the I2C module's support for the IPMI specification

File

plib_i2c.h

C

```
void PLIB_I2C_IPMIDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the I2C module's support for the IPMI specification.

Remarks

Please refer to the IPMI specification for details of the Intelligent Platform Management Interface. The IPMI specification is the property of Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Inc.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsIPMI in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_IPMIDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_IPMIDisable (I2C_MODULE_ID index)

PLIB_I2C_IPMIEnable Function

Enables the I2C module to support the Intelligent Platform Management Interface (IPMI) specification (see Remarks).

File

plib_i2c.h

C

```
void PLIB_I2C_IPMIEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to support the IPMI specification.

Remarks

Please refer to the IPMI specification for details of the Intelligent Platform Management Interface. The IPMI specification is the property of Intel Corporation, Hewlett-Packard Company, NEC Corporation, and Dell Inc.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsIPMI in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1
```

```
PLIB_I2C_IPMIEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_IPMIEnable (I2C_MODULE_ID index)

PLIB_I2C_ReservedAddressProtectDisable Function

Disables the I2C module from protecting reserved addresses, allowing it to respond to them.

File

plib_i2c.h

C

```
void PLIB_I2C_ReservedAddressProtectDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the I2C module from protecting reserved addresses, allowing it to respond to them when they match the module's slave address and mask.

Remarks

The reserved-address-protect feature can be re-enabled by calling the PLIB_I2C_ReservedAddressProtectEnable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReservedAddressProtect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_ReservedAddressProtectDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_ReservedAddressProtectDisable (I2C_MODULE_ID index)

PLIB_I2C_ReservedAddressProtectEnable Function

Enables the I2C module to protect (not respond to) reserved addresses.

File

plib_i2c.h

C

```
void PLIB_I2C_ReservedAddressProtectEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to protect reserved addresses by not responding to them, even if they match the module's slave address and mask.

Remarks

The reserved-address-protect feature can be disabled by calling the PLIB_I2C_ReservedAddressProtectDisable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReservedAddressProtect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_ReservedAddressProtectEnable(MY_I2C_ID);
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_ReservedAddressProtectEnable (I2C_MODULE_ID index)

PLIB_I2C_SlaveClockStretchingDisable Function

Disables the I2C module from stretching the slave clock.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveClockStretchingDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the I2C module from stretching the slave clock to allow time for software to respond to bytes received from the master.

Remarks

The clock stretching feature can be re-enabled by calling the PLIB_I2C_SlaveClockStretchingEnable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsClockStretching in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveClockStretchingDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveClockStretchingDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveClockStretchingEnable Function

Enables the I2C module to stretch the slave clock.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveClockStretchingEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to stretch the slave clock to allow time for software to respond to bytes received from the master.

Remarks

The clock stretching feature can be disabled by calling the PLIB_I2C_SlaveClockStretchingDisable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsClockStretching in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveClockStretchingEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveClockStretchingEnable (I2C_MODULE_ID index)

PLIB_I2C_SMBDisable Function

Disable the I2C module support for SMBus electrical signaling levels.

File

plib_i2c.h

C

```
void PLIB_I2C_SMBDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the I2C module support for SMBus electrical signaling levels.

Remarks

The SMB feature can be re-enabled by calling the PLIB_I2C_SMBEnable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSMBus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SMBDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SMBDisable (I2C_MODULE_ID index)

PLIB_I2C_SMBEnable Function

Enables the I2C module to support System Management Bus (SMBus) electrical signaling levels.

File

plib_i2c.h

C

```
void PLIB_I2C_SMBEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to support SMBus electrical signaling levels.

Remarks

The SMB feature can be disabled by calling the PLIB_I2C_SMBDisable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSMBus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SMBEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SMBEnable (I2C_MODULE_ID index)

PLIB_I2C_StopInIdleDisable Function

Disables the Stop-in-Idle feature.

File

plib_i2c.h

C

```
void PLIB_I2C_StopInIdleDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the Stop-in-Idle feature, preventing the I2C module from stopping when the processor enters Idle mode.

Remarks

The Stop-in-Idle feature can be re-enabled by calling the PLIB_I2C_StopInIdleEnable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_StopInIdleDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_StopInIdleDisable (I2C_MODULE_ID index)

PLIB_I2C_StopInIdleEnable Function

Enables the I2C module to stop when the processor enters Idle mode

File

plib_i2c.h

C

```
void PLIB_I2C_StopInIdleEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the I2C module to stop when the processor enters Idle mode.

Remarks

The Stop-in-Idle feature can be disabled by calling the PLIB_I2C_StopInIdleDisable function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_StopInIdleEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_StopInIdleEnable (I2C_MODULE_ID index)

b) General Status Functions

PLIB_I2C_ArbitrationLossClear Function

Clears the arbitration loss status flag

File

plib_i2c.h

C

```
void PLIB_I2C_ArbitrationLossClear(I2C_MODULE_ID index);
```

Returns

None.

Description

This function clears the arbitration loss status flag.

Remarks

This flag is set by hardware when bus arbitration loss occurs. Its status can be tested using the PLIB_I2C_ArbitrationLossHasOccurred function. This flag should be cleared by software after the arbitration loss has been handled. To handle the error, the entire transmission (from the Start condition to the Stop or restart condition) must be retried later when the bus becomes idle.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsArbitrationLoss in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ArbitrationLossHasOccurred ( MY_I2C_ID ) )
{
    // Handle bus collision

    PLIB_I2C_ArbitrationLossClear ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_ArbitrationLossClear (I2C_MODULE_ID index)

PLIB_I2C_ArbitrationLossHasOccurred Function

Identifies if bus arbitration has been lost.

File

plib_i2c.h

C

```
bool PLIB_I2C_ArbitrationLossHasOccurred(I2C_MODULE_ID index);
```

Returns

- true - If software if a bus collision occurred, resulting in loss of bus arbitration
- false - If no bus collision occurred

Description

This function identifies if bus arbitration has been lost.

Remarks

The arbitration status should be checked after any Master mode transmission or if an error interrupt occurs. If a bus collision occurs, the entire transmission (from the Start condition to the Stop or restart condition) must be retried later when the bus becomes idle.

This flag should be cleared by software using the PLIB_I2C_ArbitrationLossClear function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsArbitrationLoss in your application to determine whether this feature is available.

Preconditions

Bus collisions can occur during any master-mode transmission including:

- Sending a Start condition
- Sending a repeated Start condition
- Sending an address or data byte
- sending an ACK/NAK bit
- Sending a Stop condition

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ArbitrationLossHasOccurred ( MY_I2C_ID ) )
{
    // Handle bus collision

    PLIB_I2C_ArbitrationLossClear( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_ArbitrationLossHasOccurred (I2C_MODULE_ID index)

PLIB_I2C_BusIsIdle Function

Determines whether the I2C bus is idle or busy.

File

plib_i2c.h

C

```
bool PLIB_I2C_BusIsIdle(I2C_MODULE_ID index);
```

Returns

- true - The bus is currently idle. It is safe to start a master transfer.
- false - The bus is currently busy. Do not start a master transfer.

Description

This function checks to see if the I2C bus is currently idle or if there is some activity currently taking place.

Remarks

When this function returns true it does not guarantee that a bus arbitration loss cannot occur. Two or more masters can start a transfer within the minimum start signal hold time. (Refer to the I2C specification for a definition of the minimum Start condition hold time.)

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsBusIsIdle in your application to determine whether this feature is available.

Preconditions

The module must be configured appropriately and enabled.

Example

```
#define MY_I2C_ID I2C_ID_1

if (PLIB_I2C_BusIsIdle ( MY_I2C_ID ))
{
    PLIB_I2C_MasterStart ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_BusIsIdle (I2C_MODULE_ID index)

PLIB_I2C_StartClear Function

Clears the start status flag

File

plib_i2c.h

C

```
void PLIB_I2C_StartClear(I2C_MODULE_ID index);
```

Returns

None.

Description

This function clears the start status flag.

Remarks

This flag is cleared automatically by the hardware when a Stop condition is detected, but it can also be cleared by software.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsStartDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_StartWasDetected( MY_I2C_ID ) )
{
    // Handle Start condition

    PLIB_I2C_StartClear(MY_I2C_ID);
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_StartClear (I2C_MODULE_ID index)

PLIB_I2C_StartWasDetected Function

Identifies when a Start condition has been detected.

File

plib_i2c.h

C

```
bool PLIB_I2C_StartWasDetected(I2C_MODULE_ID index);
```

Returns

- true - A Start Condition has been detected
- false - No Start condition has been detected since the last time a Stop condition was detected (or the module was initialized)

Description

This function identifies when a Start condition has been detected.

Remarks

This flag is cleared automatically by the hardware when a stop condition is detected, but it can also be cleared by software using PLIB_I2C_StartClear function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsStartDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_StartWasDetected ( MY_I2C_ID ) )
{
    // Handle Start condition
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_StartWasDetected (I2C_MODULE_ID index)

PLIB_I2C_StopClear Function

Clears the stop status flag

File

plib_i2c.h

C

```
void PLIB_I2C_StopClear(I2C_MODULE_ID index);
```

Returns

None.

Description

This function clears the stop status flag.

Remarks

This flag is cleared automatically by the hardware when a Start condition is detected, but it can also be cleared by software.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsStopDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_StopWasDetected ( MY_I2C_ID ) )
{
    // Handle stop condition

    PLIB_I2C_StopClear ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_StopClear (I2C_MODULE_ID index)

PLIB_I2C_StopWasDetected Function

Identifies when a Stop condition has been detected

File

plib_i2c.h

C

```
bool PLIB_I2C_StopWasDetected(I2C_MODULE_ID index);
```

Returns

- true - A Stop condition has been detected
- false - No Stop condition has been detected since the last time a Start condition was detected (or the module was initialized)

Description

This function identifies when a Stop condition has been detected.

Remarks

This flag is cleared automatically by the hardware when a Start condition is detected, but it can also be cleared by software using the PLIB_I2C_StopClear function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsStopDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_StopWasDetected ( MY_I2C_ID ) )
{
    // Handle stop condition
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_StopWasDetected (I2C_MODULE_ID index)

c) Baud Rate Generator Control Functions

PLIB_I2C_BaudRateGet Function

Calculates the I2C module's current SCL clock frequency.

File

plib_i2c.h

C

```
I2C_BAUD_RATE PLIB_I2C_BaudRateGet(I2C_MODULE_ID index, uint32_t clockFrequencyHz);
```

Returns

SCL frequency currently used

Description

This function calculates the I2C module's current SCL clock frequency.

Remarks

The actual frequency provided may be slightly different than the frequency requested due to truncation errors. The actual frequency observed on the SCL line may be lower due to clock stretching.

The MY_CLOCK_FREQUENCY macro in the example is used as placeholder for the actual clock frequency.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsBaudRate in your application to determine whether this feature is available.

Preconditions

The returned value may not be valid if PLIB_I2C_BaudRateSet has not been previously called to set the SCL clock frequency.

Example

```
#define MY_I2C_ID I2C_ID_1
#define MY_CLOCK_FREQ_INPUT     80000000
uint32_t myBaudRate;

myBaudRate = PLIB_I2C_BaudRateGet ( MY_I2C_ID, MY_CLOCK_FREQ_INPUT );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
clockFrequencyHz	Clock Frequency (Hz) provided for the I2C module

Function

I2C_BAUD_RATE PLIB_I2C_BaudRateGet (I2C_MODULE_ID index, uint32_t clockFrequencyHz)

PLIB_I2C_BaudRateSet Function

Sets the desired baud rate.

File

plib_i2c.h

C

```
void PLIB_I2C_BaudRateSet(I2C_MODULE_ID index, uint32_t clockFrequencyHz, I2C_BAUD_RATE baudRate);
```

Returns

None.

Description

This function sets the desired baud rate so that the I2C module will operate at the desired clock frequency (on the SCL line of the bus.)

Remarks

IMPORTANT: The I2C module's high-frequency mode must be enabled for frequencies higher than 100 kHz using the PLIB_I2C_HighFrequencyEnable function. Otherwise, the high-frequency mode must be disabled using the PLIB_I2C_HighFrequencyDisable function.

The actual frequency selected may be slightly different than the frequency requested due to truncation errors. Use the PLIB_I2C_BaudRateGet function to obtain the actual baud rate value that has been programmed.

The actual frequency observed on the SCL line may be lower due to clock stretching.

If the system clock is changed during an active receive operation, a receiver error or data loss may result. To avoid this issue, verify that no receptions are in progress before changing the system clock.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsBaudRate in your application to determine whether this feature is available.

Preconditions

The source clock, being sent to the I2C module (internal to the microcontroller) must be operating at the frequency passed.

Example

```
#define MY_I2C_ID I2C_ID_1
#define MY_CLOCK_FREQUENCY_INPUT 80000000

PLIB_I2C_HighFrequencyDisable ( MY_I2C_ID );
PLIB_I2C_BaudRateSet ( MY_I2C_ID, MY_CLOCK_FREQUENCY_INPUT, 40000 );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
clockFrequencyHz	Clock Frequency (Hz)
baudRate	The desired baud rate value. This should be an appropriate value for the frequency and microcontroller in use.

Function

void PLIB_I2C_BaudRateSet (I2C_MODULE_ID index, uint32_t clockFrequencyHz,

I2C_BAUD_RATE baudRate)

d) Slave Address Control Functions

PLIB_I2C_SlaveAddress10BitGet Function

Identifies the current 10-bit Slave mode address.

File

plib_i2c.h

C

```
uint16_t PLIB_I2C_SlaveAddress10BitGet(I2C_MODULE_ID index);
```

Returns

The 10-bit slave address to which the module is currently set to respond.

Description

This function identifies the 10-bit slave address to which the module will currently respond.

Remarks

The 16-bit address will be right-aligned in the 16-bit return value.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveAddress10Bit in your application to determine whether this feature is available.

Preconditions

The address provided may not be valid if it has not been previously set.

Example

```
#define MY_I2C_ID I2C_ID_1
uint16_t slaveAddress;

if (PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slaveAddress = PLIB_I2C_SlaveAddress10BitGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

uint16_t PLIB_I2C_SlaveAddress10BitGet (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddress10BitSet Function

Selects 10-bit Slave mode addressing and sets the address value.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveAddress10BitSet(I2C_MODULE_ID index, uint16_t address);
```

Returns

None.

Description

This function selects 10-bit Slave mode addressing sets the slave address to which the module will respond when operating in Slave mode.

Remarks

I2C modules on some microcontroller families may also support an address mask (to allow the module to respond to multiple addresses. When using these microcontrollers, the PLIB_I2C_SlaveAddress10BitSetMasked may be used to support the mask feature. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_SLAVE_ADDRESS_10_BIT macro in the example is used as a placeholder for the actual desired slave address.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveAddress10Bit in your application to determine whether this feature is available.

Preconditions

None

Example

```
#define MY_I2C_ID I2C_ID_1
#define MY_SLAVE_ADDR_10_BIT    0x23

PLIB_I2C_SlaveAddress10BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_10_BIT );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
address	The 10-bit slave address to which the module will respond (The address should be right-aligned in the 16-bit parameter, without any read/write bit in the 0 position)

Function

void PLIB_I2C_SlaveAddress10BitSet (I2C_MODULE_ID index, uint16_t address)

PLIB_I2C_SlaveAddress10BitWasDetected Function

Detects reception of the second byte of a 10-bit slave address.

File

plib_i2c.h

C

```
bool PLIB_I2C_SlaveAddress10BitWasDetected(I2C_MODULE_ID index);
```

Returns

- true - If the second byte of the local 10-bit address has been received
- false - If the second byte of the local 10-bit address has not been received

Description

This function detects if the second byte of a 10-bit slave address (containing the low-order 8 bits) matching the local address has been received.

Remarks

This function should only be used by slave receivers.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveAddressDetect in your application to determine whether this feature is available.

Preconditions

The module must have been configured for 10-bit addressing in Slave mode, enabled, and the first byte of the 10-bit local slave address must have already been received and matched.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t  i2cReadData;

if ( PLIB_I2C_SlaveAddress10BitWasDetected ( MY_I2C_ID ) )
{
    i2cReadData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_SlaveAddress10BitWasDetected (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddress7BitGet Function

Identifies the current 7-bit Slave mode address.

File

plib_i2c.h

C

```
uint8_t PLIB_I2C_SlaveAddress7BitGet(I2C_MODULE_ID index);
```

Returns

The 7-bit slave address to which the module is currently set to respond.

Description

This function identifies the 7-bit slave address to which the module will currently respond.

Remarks

The 7-bit address will be right-aligned in the 8-bit return value.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveAddress7Bit in your application to determine whether this feature is available.

Preconditions

The address provided may not be valid if it has not been previously set.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t slave_address7bit;

if (!PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slave_address7bit = PLIB_I2C_SlaveAddress7BitGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

uint8_t PLIB_I2C_SlaveAddress7BitGet (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddress7BitSet Function

Selects 7-bit Slave mode addressing and sets the slave address value.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveAddress7BitSet(I2C_MODULE_ID index, uint8_t address);
```

Returns

None.

Description

This function selects 7-bit Slave mode addressing sets the address to which the module will respond when operating in Slave mode.

Remarks

I2C modules on some microcontroller families may also support an address mask (to allow the module to respond to multiple addresses. When using these microcontrollers, the PLIB_I2C_SlaveAddress7BitSetMasked may be used to support the mask feature. Refer to the specific device

data sheet to determine whether this feature is supported for your device.

The MY_SLAVE_ADDRESS_7_BIT macro in the example is used as a placeholder for the actual desired slave address.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveAddress7Bit in your application to determine whether this feature is available.

Preconditions

None

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveAddress7BitSet ( MY_I2C_ID, MY_SLAVE_ADDRESS_7_BIT );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
address	The 7-bit slave address to which the module will respond (The address should be right-aligned in the 8-bit parameter, without any read/write bit in the 0 position)

Function

void PLIB_I2C_SlaveAddress7BitSet (I2C_MODULE_ID index, uint8_t address)

PLIB_I2C_SlaveAddressHoldDisable Function

Disables Address holding.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveAddressHoldDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables address holding.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_SlaveAddressHoldEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveAddressHoldDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveAddressHoldDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddressHoldEnable Function

Enables address holding.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveAddressHoldEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables address holding. If address byte is received, following the 8th falling edge of clock for a clock line will be held low.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_SlaveAddressHoldEnable in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveAddressHoldEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveAddressHoldEnable (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddressIsDetected Function

Detects if the most recent byte received is a data or an address byte.

File

plib_i2c.h

C

```
bool PLIB_I2C_SlaveAddressIsDetected(I2C_MODULE_ID index);
```

Returns

- true - If the byte received is an address byte
- false - If the byte received is a data byte

Description

This function identifies if the most recent byte received was a data byte or an address byte.

Remarks

This function should only be used by slave receivers.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveReadRequest in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t getData;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_SlaveAddressIsDetected ( MY_I2C_ID ) )
    {
        if ( PLIB_I2C_SlaveReadIsRequested ( MY_I2C_ID ) )
        {
            // Begin slave transmit mode
        }
        else
        {
            // Begin slave receive mode
        }
    }

    // Read the data and release the bus
    getData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
    PLIB_I2C_SlaveClockRelease ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_SlaveAddressIsDetected (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddressIsGeneralCall Function

Identifies if the current slave address received is the general call address.

File

plib_i2c.h

C

```
bool PLIB_I2C_SlaveAddressIsGeneralCall(I2C_MODULE_ID index);
```

Returns

- true - If the slave address received is the general call address
- false - if the slave address received is not the general call address

Description

This function identifies if the current slave address received is the general call address.

Remarks

This bit is set when the general call address has been received.

This bit is automatically cleared by hardware when a Stop condition occurs and cannot be cleared by software.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsGeneralCallAddressDetect in your application to determine whether this feature is available.

Preconditions

A slave address must have been received.

Example

```
#define MY_I2C_ID I2C_ID_1
```

```
uint8_t  slaveAddress7Bit;

if ( !PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_SlaveAddressIsGeneralCall( MY_I2C_ID ) )
    {
        // Handle general call address
    }

    slaveAddress7Bit = PLIB_I2C_SlaveAddress7BitGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_SlaveAddressIsGeneralCall (I2C_MODULE_ID index)

PLIB_I2C_SlaveAddressModeIs10Bits Function

Identifies if the current slave address mode is 7-bits or 10-bits.

File

plib_i2c.h

C

```
bool PLIB_I2C_SlaveAddressModeIs10Bits(I2C_MODULE_ID index);
```

Returns

- true - If the current slave addressing mode is 10-bits
- false - if the current slave addressing mode is 7-bits

Description

This function identifies if the current slave addressing mode is 7-bits or 10-bits.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveAddress10Bit in your application to determine whether this feature is available.

Preconditions

The mode provided may not be valid if the address mode has not been previously set.

Example

```
uint8_t  slave_address7Bit;
uint16_t slave_address10Bit;

if ( PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slave_address10Bit = PLIB_I2C_SlaveAddress10BitGet ( MY_I2C_ID );
}
else
{
    slave_address7Bit = PLIB_I2C_SlaveAddress7BitGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_SlaveAddressModeIs10Bits (I2C_MODULE_ID index)

PLIB_I2C_SlaveDataIsDetected Function

Detects if the most recent byte received is a data or an address byte.

File

plib_i2c.h

C

```
bool PLIB_I2C_SlaveDataIsDetected(I2C_MODULE_ID index);
```

Returns

- true - If the byte received is a data byte
- false - If the byte received is an address byte

Description

This function identifies if the most recent byte received was a data byte or an address byte.

Remarks

This function should only be used by slave receivers.

This function returns the opposite of the PLIB_I2C_SlaveAddressIsDetected function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveDataDetect in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t dataReceived;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_SlaveDataIsDetected( MY_I2C_ID ) )
    {
        // Read the data and release the bus
        dataReceived = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
        PLIB_I2C_SlaveClockRelease ( MY_I2C_ID );
    }
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module.

Function

bool PLIB_I2C_SlaveDataIsDetected (I2C_MODULE_ID index)

PLIB_I2C_SlaveReadIsRequested Function

Detects if the request from the master was a read or write.

File

plib_i2c.h

C

```
bool PLIB_I2C_SlaveReadIsRequested(I2C_MODULE_ID index);
```

Returns

- true - If an external master is requesting data (slave read/transmit)

- false - If an external master is sending data (slave write/receive)

Description

This function identifies if a slave read (transmit) or a slave write (receive) was requested by the master that addressed the module.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveReadRequest in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t slaveTxData;
uint8_t slaveRxData;

if ( PLIB_I2C_SlaveReadIsRequested ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_TransmitterIsReady ( MY_I2C_ID ) )
    {
        PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, slaveTxData );
    }
}
else
{
    slaveRxData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_SlaveReadIsRequested (I2C_MODULE_ID index)

e) Slave Mask Control Functions

PLIB_I2C_SlaveMask10BitGet Function

Identifies the current 10-bit Slave mode address mask.

File

plib_i2c.h

C

```
uint16_t PLIB_I2C_SlaveMask10BitGet(I2C_MODULE_ID index);
```

Returns

The 10-bit slave address mask that the module is currently using to determine to which addresses the module will respond.

Description

This function identifies the 10-bit slave address mask that is currently being used to determine which slave addresses the module will respond.

Remarks

The 10-bit address mask will be Right-aligned in the 16-bit return value.

This function is not supported on microcontrollers with I2C modules that do not support the slave address mask feature. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveMask in your application to determine whether this feature is available.

Preconditions

The address mask provided may not be valid if it has not been previously been set.

Example

```
#define MY_I2C_ID I2C_ID_1

uint16_t slave_address;
uint16_t slave_addressMask;

if ( PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slave_address     = PLIB_I2C_SlaveAddress10BitGet ( MY_I2C_ID );
    slave_addressMask = PLIB_I2C_SlaveMask10BitGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

uint16_t PLIB_I2C_SlaveMask10BitGet (I2C_MODULE_ID index)

PLIB_I2C_SlaveMask10BitSet Function

Sets the 10-bit mask for Slave mode addressing.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveMask10BitSet(I2C_MODULE_ID index, uint16_t mask);
```

Returns

None.

Description

This function sets the 10-bit mask for Slave mode addressing. The address mask (if supported) is used, along with the slave address to identify to which addresses the module will respond when operating in Slave mode. It acts as an "ignore" mask, allowing the module to ignore bits within the slave address and thus respond to multiple slave addresses on microcontrollers with I2C modules that support the mask feature.

Remarks

I2C modules on some microcontroller families may not support the mask feature, in which case the PLIB_I2C_SlaveMask10BitSet function will not be supported. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_SLAVE_ADDRESS_10_BIT and MY_SLAVE_ADDRESS_MASK_10_BIT macros in the example code are used as placeholders for the actual desired slave address and mask that must be filled in by the caller.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveMask in your application to determine whether this feature is available.

Preconditions

None

Example

```
#define MY_I2C_ID                       I2C_ID_1
#define MY_SLAVE_ADDR_10_BIT            0x23
#define MY_SLAVE_ADDR_MASK_10_BIT       0x12

PLIB_I2C_SlaveAddress10BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_10_BIT );
PLIB_I2C_SlaveMask10BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_MASK_10_BIT );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
mask	This parameter identifies bits in the address that are "don't care" bits. These bits will be ignored when attempting to match the address, effectively allowing the module to recognize multiple slave addresses. (To match an address exactly, this
value must be zero (0).) (This value must also be right	aligned in the 16-bit parameter.)

Function

void PLIB_I2C_SlaveMask10BitSet (I2C_MODULE_ID index, uint16_t mask)

PLIB_I2C_SlaveMask7BitGet Function

Identifies the current 7-bit Slave mode address mask.

File

plib_i2c.h

C

```
uint8_t PLIB_I2C_SlaveMask7BitGet(I2C_MODULE_ID index);
```

Returns

The 7-bit Slave mode address mask that the module is currently using to determine to which addresses the module will respond.

Description

This function identifies the 7-bit Slave mode address mask that is currently being used to determine which slave addresses the module will respond.

Remarks

The 7-bit address mask will be right-aligned in the 8-bit return value.

This function is not supported on microcontrollers with I2C modules that do not support the slave address mask feature. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveMask in your application to determine whether this feature is available.

Preconditions

The address mask provided may not be valid if it has not been previously been set.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t slaveAddr;
uint8_t slaveAddressMsk;

if ( !PLIB_I2C_SlaveAddressModeIs10Bits ( MY_I2C_ID ) )
{
    slaveAddr     = PLIB_I2C_SlaveAddress7BitGet ( MY_I2C_ID );
    slaveAddressMsk = PLIB_I2C_SlaveMask7BitGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

uint8_t PLIB_I2C_SlaveMask7BitGet (I2C_MODULE_ID index)

PLIB_I2C_SlaveMask7BitSet Function

Sets the 7-bit mask for Slave mode addressing .

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveMask7BitSet(I2C_MODULE_ID index, uint8_t mask);
```

Returns

None.

Description

This function sets the 7-bit mask for Slave mode addressing. The address mask (if supported) is used, along with the slave address to identify to which addresses the module will respond when operating in Slave mode. It acts as an "ignore" mask, allowing the module to ignore bits within the slave address and will respond to multiple slave addresses on microcontrollers with I2C modules that support the mask feature.

Remarks

I2C modules on some microcontroller families may not support the mask feature, in which case the PLIB_I2C_SlaveMask7BitSet function will not be supported. Refer to the specific device data sheet to determine whether this feature is supported for your device.

The MY_SLAVE_ADDRESS_7_BIT and MY_SLAVE_ADDRESS_MASK_7_BIT macros in the example code are used as placeholders for the actual desired slave address and mask that must be filled in by the caller.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveMask in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID                   I2C_ID_1
#define MY_SLAVE_ADDR_7_BIT         0x23
#define MY_SLAVE_ADDR_MASK_7_BIT    0x12

PLIB_I2C_SlaveAddress7BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_7_BIT );
PLIB_I2C_SlaveMask7BitSet ( MY_I2C_ID, MY_SLAVE_ADDR_MASK_7_BIT );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
mask	This parameter identifies bits in the address that are "don't care" bits. These bits will be ignored when attempting to match the address, effectively allowing the module to recognize multiple slave addresses. (To match an address exactly, this parameter must be zero (0).)

Function

void PLIB_I2C_SlaveMask7BitSet (I2C_MODULE_ID index, uint8_t mask)

f) Slave Control Functions

PLIB_I2C_AcksequenceIsInProgress Function

Checks whether the acknowledge sequence is in progress or it is completed.

File

plib_i2c.h

C

```
bool PLIB_I2C_AcksequenceIsInProgress(I2C_MODULE_ID index);
```

Returns

- true - Acknowledge sequence is in progress.
- false - Acknowledge sequence is not started or completed.

Description

This function checks whether the acknowledge sequence is in progress or it is completed.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsAcksequenceProgress in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_AcksequenceIsInProgress ( MY_I2C_ID ) )
{
  //Transmission completed
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_AcksequenceIsInProgress (I2C_MODULE_ID index)

PLIB_I2C_DataLineHoldTimeSet Function

Sets the data line hold time.

File

plib_i2c.h

C

```
void PLIB_I2C_DataLineHoldTimeSet(I2C_MODULE_ID index, I2C_SDA_MIN_HOLD_TIME sdaHoldTimeNs);
```

Returns

None.

Description

This function sets the data line hold time.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsDataLineHoldTime in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_PLIB_I2C_DataLineHoldTimeSet ( MY_I2C_ID, I2C_SDA_MIN_HOLD_TIME_300NS );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module.
sdaHoldTimeNs	SDA hold time in nanoseconds.

Function

void PLIB_I2C_DataLineHoldTimeSet (I2C_MODULE_ID index, I2C_SDA_MIN_HOLD_TIME sdaHoldTimeNs)

PLIB_I2C_SlaveBufferOverwriteDisable Function

Disables buffer overwrite.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveBufferOverwriteDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables buffer overwrite. If the buffer overwrite is disabled, on data receive, when the previous data is not read, the buffer will not be overwritten.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveBufferOverwrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveBufferOverwriteDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveBufferOverwriteDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveBufferOverwriteEnable Function

Enables buffer overwrite.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveBufferOverwriteEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables buffer overwrite. If the buffer overwrite is enabled, on data receive, even if the previous data is not read, the buffer will be

filled with new data and acknowledge will be generated.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveBufferOverwrite in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveBufferOverwriteEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveBufferOverwriteEnable (I2C_MODULE_ID index)

PLIB_I2C_SlaveBusCollisionDetectDisable Function

Disables bus collision detect interrupts.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveBusCollisionDetectDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables bus collision detect interrupts.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_SlaveBusCollisionDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveBusCollisionDetectDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_SlaveBusCollisionDetectDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveBusCollisionDetectEnable Function

Enables slave bus collision interrupts.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveBusCollisionDetectEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables bus collision interrupts.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_SlaveBusCollisionDetect in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveAddressHoldEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveBusCollisionDetectEnable (I2C_MODULE_ID index)

PLIB_I2C_SlaveClockHold Function

Holds the SCL clock line low after receiving a byte.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveClockHold(I2C_MODULE_ID index);
```

Returns

None.

Description

This function allows an I2C slave to stretch the SCL clock line, holding it low to throttle data transfer from a master transmitter.

Remarks

This function will cause the SCL line to be forced low, after the current byte has been fully received.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveClockHold in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started by an external master.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveClockHold ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveClockHold (I2C_MODULE_ID index)

PLIB_I2C_SlaveClockRelease Function

Releases a previously held SCL clock line.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveClockRelease(I2C_MODULE_ID index);
```

Returns

None.

Description

This function allows a slave receiver to release a previously held SCL clock line, allowing it to go high and allowing data transfer to continue.

Remarks

Calling this function when the clock has not been held will not cause any problems.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveClockHold in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started by an external master, and the SCL clock line should have been previously held (forced low) by the I2C module.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveClockRelease ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveClockRelease (I2C_MODULE_ID index)

PLIB_I2C_SlaveDataHoldDisable Function

Disables data holding.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveDataHoldDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables data holding.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_SlaveDataHoldEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveDataHoldDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveDataHoldDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveDataHoldEnable Function

Enables data holding.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveDataHoldEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables data holding. If a data byte is received, following the 8th falling edge of clock for a clock line will be held low.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_SlaveDataHoldEnable in your application to determine whether this feature is available.

This API is supported only for the slave mode of I2C.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveDataHoldEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveDataHoldEnable (I2C_MODULE_ID index)

PLIB_I2C_SlaveInterruptOnStartDisable Function

Disables the feature of generating interrupt on start condition.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveInterruptOnStartDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the feature of generating interrupt on start condition.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveInterruptOnStart in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveInterruptOnStartDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveInterruptOnStartDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveInterruptOnStartEnable Function

Enables the feature of generating interrupt on start condition.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveInterruptOnStartEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the feature of generating interrupt on start condition.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveInterruptOnStart in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveInterruptOnStartEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveInterruptOnStartEnable (I2C_MODULE_ID index)

PLIB_I2C_SlaveInterruptOnStopDisable Function

Disables the feature of generating interrupt on stop condition.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveInterruptOnStopDisable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function disables the feature of generating interrupt on stop condition.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveInterruptOnStop in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveInterruptOnStopDisable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveInterruptOnStopDisable (I2C_MODULE_ID index)

PLIB_I2C_SlaveInterruptOnStopEnable Function

Enables the feature of generating interrupt on stop condition.

File

plib_i2c.h

C

```
void PLIB_I2C_SlaveInterruptOnStopEnable(I2C_MODULE_ID index);
```

Returns

None.

Description

This function enables the feature of generating interrupt on stop condition.

Remarks

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsSlaveInterruptOnStop in your application to determine whether this feature is available.

This API is applicable only in I2C slave mode.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_SlaveInterruptOnStopEnable ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_SlaveInterruptOnStopEnable (I2C_MODULE_ID index)

g) Master Control Functions

PLIB_I2C_MasterReceiverClock1Byte Function

Drives the bus clock to receive 1 byte of data from a slave device.

File

plib_i2c.h

C

```
void PLIB_I2C_MasterReceiverClock1Byte(I2C_MODULE_ID index);
```

Returns

None.

Description

This function drives the bus clock to receive 1 byte of data from a slave device.

Remarks

The module stops driving the bus clock after the reception of a single byte of data and this function must be called again to receive another byte.

After the module has finished receiving a data byte (determined by responding to an I2C master interrupt and/or by checking the

PLIB_I2C_ReceivedByteIsAvailable function), software should check the PLIB_I2C_ReceiverOverflowHasOccurred function to ensure that no data was lost because the receiver buffer was full when the byte was completely received and ready to be placed into the receiver buffer.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsMasterReceiverClock1Byte in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and the module must be the intended receiver of the next byte of data.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterReceiverClock1Byte ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_MasterReceiverClock1Byte (I2C_MODULE_ID index)

PLIB_I2C_MasterStart Function

Sends an I2C Start condition on the I2C bus in Master mode.

File

plib_i2c.h

C

```
void PLIB_I2C_MasterStart(I2C_MODULE_ID index);
```

Returns

None.

Description

This function sends the start signal (a falling edge on SDA while SCL is high) to start a transfer on the I2C bus when acting in Master mode.

Remarks

Only an I2C master can start a transfer on the bus. The bus is considered to be busy after a Start condition.

After the Start condition has completed (detected by responding to the I2C master interrupt), software should check for arbitration loss by calling the PLIB_I2C_ArbitrationLossHasOccurred function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsMasterStart in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled and the I2C bus must currently be idle.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_BusIsIdle ( MY_I2C_ID ) )
{
    PLIB_I2C_MasterStart ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_MasterStart (I2C_MODULE_ID index)

PLIB_I2C_MasterStartRepeat Function

Sends a repeated Start condition during an ongoing transfer in Master mode.

File

plib_i2c.h

C

```
void PLIB_I2C_MasterStartRepeat(I2C_MODULE_ID index);
```

Returns

None.

Description

This function supports sending a repeated Start condition to change slave targets or transfer direction to support certain I2C transfer formats in Master mode.

Remarks

Only an I2C master that has already started a transfer can send a repeated Start condition.

After the repeated-start condition has completed (detected by responding to the I2C master interrupt), software should check for arbitration loss by the PLIB_I2C_ArbitrationLossHasOccurred function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsMasterStartRepeat in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, and a transfer must have been previously started.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStartRepeat ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_MasterStartRepeat (I2C_MODULE_ID index)

PLIB_I2C_MasterStop Function

Sends an I2C Stop condition to terminate a transfer in Master mode.

File

plib_i2c.h

C

```
void PLIB_I2C_MasterStop(I2C_MODULE_ID index);
```

Returns

None.

Description

This function sends the stop signal (a rising edge on SDA while SCL is high) on the I2C bus, to end a transfer on the I2C bus when in Master mode.

Remarks

Only an I2C master that has already started a transfer can send a Stop condition.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsMasterStop in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately, enabled, and a previously started transfer must be completed.

Example

```
#define MY_I2C_ID I2C_ID_1

PLIB_I2C_MasterStop ( MY_I2C_ID );
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_MasterStop (I2C_MODULE_ID index)

h) Transmitter Control Functions

PLIB_I2C_TransmitterByteHasCompleted Function

Detects whether the module has finished transmitting the most recent byte.

File

plib_i2c.h

C

```
bool PLIB_I2C_TransmitterByteHasCompleted(I2C_MODULE_ID index);
```

Returns

- true - If the transmitter has completed sending the data byte
- false - If the transmitter is still busy sending the data byte

Description

This function determines if the transmitter has finished sending the most recently sent byte on the I2C bus.

Remarks

This function should be used by both master and slave transmitters.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterByteComplete in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and a data or address byte must have been sent.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_TransmitterByteHasCompleted ( MY_I2C_ID ) )
{
  //Transmission completed
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_TransmitterByteHasCompleted (I2C_MODULE_ID index)

PLIB_I2C_TransmitterByteSend Function

Sends a byte of data on the I2C bus.

File

plib_i2c.h

C

```
void PLIB_I2C_TransmitterByteSend(I2C_MODULE_ID index, uint8_t data);
```

Returns

None.

Description

This function allows the caller to send a byte of data on the I2C bus.

Remarks

This function should be used by both master and slave transmitters.

The caller must either first call the PLIB_I2C_TransmitterIsReady function before calling this function to ensure that the transmitter is ready to receive a new byte to transmit or call the PLIB_I2C_TransmitterOverflowHasOccurred function immediately after calling this function to ensure that a transmitter write collision did not occur.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterByteSend in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and the I2C module's transmitter must be ready to accept a byte of data to send.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t transmitData;

//set 'transmitData' variable

if ( PLIB_I2C_TransmitterIsReady ( MY_I2C_ID ) )
{
    PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, transmitData );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
data	Data byte to send on the I2C bus

Function

void PLIB_I2C_TransmitterByteSend (I2C_MODULE_ID index, uint8_t data)

PLIB_I2C_TransmitterByteWasAcknowledged Function

Determines whether the most recently sent byte was acknowledged.

File

plib_i2c.h

C

```
bool PLIB_I2C_TransmitterByteWasAcknowledged(I2C_MODULE_ID index);
```

Returns

- true - If the receiver ACKed the byte
- false - If the receiver NAKed the byte

Description

This function allows a transmitter to determine if the byte just sent was positively acknowledged (ACKed) or negatively acknowledged (NAKed) by the receiver.

Remarks

This function can be used by both master or slave transmitters.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterByteAcknowledge in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, a byte of data must have been sent on the I2C bus, and the transmission must have completed.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_TransmitterByteHasCompleted ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_TransmitterByteWasAcknowledged ( MY_I2C_ID ) )
    {
        // transmission successful
    }
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_TransmitterByteWasAcknowledged (I2C_MODULE_ID index)

PLIB_I2C_TransmitterIsBusy Function

Identifies if the transmitter of the specified I2C module is currently busy (unable to accept more data).

File

plib_i2c.h

C

```
bool PLIB_I2C_TransmitterIsBusy(I2C_MODULE_ID index);
```

Returns

- true - The transmitter is busy (unable to accept new data)
- false - The transmitter is ready (able to accept new data)

Description

This function identifies if the transmitter of the specified I2C module is currently busy (unable to accept more data).

Remarks

This function returns the inverse of the PLIB_I2C_TransmitterIsReady function.

This flag is cleared automatically by the hardware when the transmitter is ready. It cannot be cleared by software.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterIsBusy in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t transData;

//set 'transData' variable

if ( !PLIB_I2C_TransmitterIsBusy( MY_I2C_ID ) )
{
    PLIB_I2C_TransmitterByteSend( MY_I2C_ID, transData );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_TransmitterIsBusy (I2C_MODULE_ID index)

PLIB_I2C_TransmitterIsReady Function

Detects if the transmitter is ready to accept data to transmit.

File

plib_i2c.h

C

```
bool PLIB_I2C_TransmitterIsReady(I2C_MODULE_ID index);
```

Returns

- true - If the transmitter is ready to accept more data
- false - If the transmitter is not ready to accept more data

Description

This function determines if the transmitter is ready to accept more data to be transmitted on the I2C bus.

Remarks

This function should be used by both master and slave transmitters.

This function returns the inverse of the PLIB_I2C_TransmitterIsBusy function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterIsBusy in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately, enabled, and a transfer must have been previously started.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t sendData;

//set 'sendData' variable

if ( PLIB_I2C_TransmitterIsReady ( MY_I2C_ID ) )
{
```

```
    PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, sendData );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_TransmitterIsReady (I2C_MODULE_ID index)

PLIB_I2C_TransmitterOverflowClear Function

Clears the transmitter overflow status flag.

File

plib_i2c.h

C

```
void PLIB_I2C_TransmitterOverflowClear(I2C_MODULE_ID index);
```

Returns

None.

Description

This function clears the transmitter overflow status flag.

Remarks

This flag is set by hardware when an overflow error occurs. Its status can be tested using the PLIB_I2C_TransmitterOverflowHasOccurred function. This flag must be cleared be cleared by software after the overflow error has been handled. To handle the error, software must retry the write later after the PLIB_I2C_TransmitterByteSend function returns TRUE.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t my_data;

//set 'my_data' variable
PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, my_data );
if ( PLIB_I2C_TransmitterOverflowHasOccurred ( MY_I2C_ID ) )
{
    // Handle overflow error

    PLIB_I2C_TransmitterOverflowClear ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_TransmitterOverflowClear (I2C_MODULE_ID index)

PLIB_I2C_TransmitterOverflowHasOccurred Function

Identifies if a transmitter overflow error has occurred.

File

plib_i2c.h

C

```
bool PLIB_I2C_TransmitterOverflowHasOccurred(I2C_MODULE_ID index);
```

Returns

- true - If software attempted to write a byte to the transmitter buffer while the transmitter was busy and unable to accept a new byte (i.e., the write will not occur)
- false - If no transmitter overflow occurred when software attempted to write a byte to the transmitter buffer (i.e., the write occurred successfully)

Description

This function identifies if a transmitter overflow error has occurred.

Remarks

This flag must be cleared be cleared by software using the PLIB_I2C_TransmitterOverflowClear function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsTransmitterOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t txData;

//set 'txData' variable

PLIB_I2C_TransmitterByteSend ( MY_I2C_ID, txData );
if ( PLIB_I2C_TransmitterOverflowHasOccurred ( MY_I2C_ID ) )
{
    // Handle overflow error

    PLIB_I2C_TransmitterOverflowClear ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_TransmitterOverflowHasOccurred (I2C_MODULE_ID index)

i) Receiver Control Functions

PLIB_I2C_ReceivedByteAcknowledge Function

Allows a receiver to acknowledge a that a byte of data has been received.

File

plib_i2c.h

C

```
void PLIB_I2C_ReceivedByteAcknowledge(I2C_MODULE_ID index, bool ack);
```

Returns

None.

Description

This function allows a receiver to positively acknowledge (ACK) or negatively acknowledge (NAK) a byte of data that has been received from the I2C bus.

Remarks

This function can only be used by master receivers. Slave receivers automatically ACK or NAK bytes.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceivedByteAcknowledge in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and a byte of data must have been received from the I2C bus.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_MasterReceiverReadyToAcknowledge ( MY_I2C_ID ) )
    {
        PLIB_I2C_ReceivedByteAcknowledge ( MY_I2C_ID, true );
        data = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
    }
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module
ack	Determines how the byte should be acknowledged: • If true, positively acknowledges (ACK) the byte of data received • If false, negatively acknowledges (NAK) the byte of data received

Function

void PLIB_I2C_ReceivedByteAcknowledge (I2C_MODULE_ID index, bool ack)

PLIB_I2C_ReceivedByteGet Function

Gets a byte of data received from the I2C bus interface.

File

plib_i2c.h

C

```
uint8_t PLIB_I2C_ReceivedByteGet(I2C_MODULE_ID index);
```

Returns

A byte of data received from the I2C bus.

Description

This function allows the caller to read a byte of data received from the I2C bus.

Remarks

This function should be used by both master and slave receivers.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceivedByteGet in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, and a byte of data must have been received from the I2C bus.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t receivedData;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    PLIB_I2C_ReceivedByteAcknowledge ( MY_I2C_ID, true );
    receivedData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

uint8_t PLIB_I2C_ReceivedByteGet (I2C_MODULE_ID index)

PLIB_I2C_ReceivedByteIsAvailable Function

Detects whether the receiver has data available.

File

plib_i2c.h

C

```
bool PLIB_I2C_ReceivedByteIsAvailable(I2C_MODULE_ID index);
```

Returns

- true - If the receiver has data available
- false - If the receiver does not have data available

Description

This function determines if the receiver has data available to be read by software.

Remarks

This function should be used by both master and slave receivers.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceivedByteAvailable in your application to determine whether this feature is available.

Preconditions

The I2C module must have been configured appropriately and enabled, and a transfer must have been previously started.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t readData;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    readData = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_ReceivedByteIsAvailable (I2C_MODULE_ID index)

PLIB_I2C_ReceiverByteAcknowledgeHasCompleted Function

Determines if the previous acknowledge has completed.

File

plib_i2c.h

C

```
bool PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_MODULE_ID index);
```

Returns

- true - If the acknowledgment has completed
- false - If the acknowledgment has not completed

Description

This function allows the receiver to determine if the acknowledgment signal has completed.

Remarks

This function can only be used by master receivers. Slave receivers automatically ACK or NAK bytes.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceivedByteAcknowledge in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started, a byte of data must have been received on the I2C bus, and the acknowledgment must have been started (by calling the PLIB_I2C_ReceivedByteAcknowledge function).

Example

```
PLIB_I2C_ReceivedByteAcknowledge ( MY_I2C_ID, true );

if ( PLIB_I2C_ReceiverByteAcknowledgeHasCompleted ( MY_I2C_ID ) )
{
    // acknowledgment completed
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_ReceiverByteAcknowledgeHasCompleted (I2C_MODULE_ID index)

PLIB_I2C_ReceiverOverflowClear Function

Clears the receiver overflow status flag.

File

plib_i2c.h

C

```
void PLIB_I2C_ReceiverOverflowClear(I2C_MODULE_ID index);
```

Returns

None.

Description

This function clears the receiver overflow status flag.

Remarks

This flag is set by hardware when an overflow error occurs. Its status can be tested using the PLIB_I2C_ReceiverOverflowHasOccurred function. This flag should be cleared be cleared by software after the overflow error has been handled by reading the byte in the receiver buffer.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceiverOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ReceiverOverflowHasOccurred ( MY_I2C_ID ) )
{
    // Handle overflow error

    PLIB_I2C_ReceiverOverflowClear ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

void PLIB_I2C_ReceiverOverflowClear (I2C_MODULE_ID index)

PLIB_I2C_ReceiverOverflowHasOccurred Function

Identifies if a receiver overflow error has occurred.

File

plib_i2c.h

C

```
bool PLIB_I2C_ReceiverOverflowHasOccurred(I2C_MODULE_ID index);
```

Returns

- true - If a byte was received while the receiver buffer still held a previously received byte (The new byte will be lost)
- false - If no receiver overflow occurred when a byte has been received

Description

This function identifies if a receiver overflow error has occurred.

Remarks

This flag should be cleared be cleared by software using the PLIB_I2C_ReceiverOverflowClear function.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceiverOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_I2C_ID I2C_ID_1

if ( PLIB_I2C_ReceiverOverflowHasOccurred ( MY_I2C_ID ) )
{
    // Handle overflow error
```

```
    PLIB_I2C_ReceiverOverflowClear ( MY_I2C_ID );
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_ReceiverOverflowHasOccurred (I2C_MODULE_ID index)

PLIB_I2C_MasterReceiverReadyToAcknowledge Function

Checks whether the hardware is ready to acknowledge.

File

plib_i2c.h

C

```
bool PLIB_I2C_MasterReceiverReadyToAcknowledge(I2C_MODULE_ID index);
```

Returns

- true - If the hardware status is ready to acknowledge
- false - If the hardware is not ready to acknowledge

Description

This function checks for preconditions before acknowledging a slave.

Remarks

This function can only be used by master receivers.

The MY_I2C_ID macro in the example above is used as a placeholder for the actual I2C bus ID (as defined by the processor-specific I2C_MODULE_ID enum).

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_I2C_ExistsReceivedByteAcknowledge in your application to determine whether this feature is available.

Preconditions

The module must have been configured appropriately and enabled, a transfer must have been previously started and a byte of data must have been received from the I2C bus.

Example

```
#define MY_I2C_ID I2C_ID_1
uint8_t data;

if ( PLIB_I2C_ReceivedByteIsAvailable ( MY_I2C_ID ) )
{
    if ( PLIB_I2C_MasterReceiverReadyToAcknowledge ( MY_I2C_ID ) )
    {
        PLIB_I2C_ReceivedByteAcknowledge ( MY_I2C_ID, true );
        data = PLIB_I2C_ReceivedByteGet ( MY_I2C_ID );
    }
}
```

Parameters

Parameters	Description
index	Identifies the desired I2C module

Function

bool PLIB_I2C_MasterReceiverReadyToAcknowledge (I2C_MODULE_ID index)

j) Feature Existence Functions

PLIB_I2C_ExistsAcksequenceProgress Function

Identifies whether the AcksequenceIsInProgress feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsAcksequenceProgress(I2C_MODULE_ID index);
```

Returns

- true - If the AcksequenceIsInProgress feature is supported on the device
- false - If the AcksequenceIsInProgress feature is not supported on the device

Description

This function identifies whether the AcksequenceIsInProgress feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_AcksequenceIsInProgress

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsAcksequenceProgress(I2C_MODULE_ID index)

PLIB_I2C_ExistsArbitrationLoss Function

Identifies whether the ArbitrationLoss feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsArbitrationLoss(I2C_MODULE_ID index);
```

Returns

- true - If the ArbitrationLoss feature is supported on the device
- false - If the ArbitrationLoss feature is not supported on the device

Description

This function identifies whether the ArbitrationLoss feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_ArbitrationLossHasOccurred
- PLIB_I2C_ArbitrationLossClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsArbitrationLoss(I2C_MODULE_ID index)

PLIB_I2C_ExistsBaudRate Function

Identifies whether the BaudRate feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsBaudRate(I2C_MODULE_ID index);
```

Returns

- true - If the BaudRate feature is supported on the device
- false - If the BaudRate feature is not supported on the device

Description

This function identifies whether the BaudRate feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_BaudRateSet
- PLIB_I2C_BaudRateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsBaudRate(I2C_MODULE_ID index)

PLIB_I2C_ExistsBusIsIdle Function

Identifies whether the BusIdle feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsBusIsIdle(I2C_MODULE_ID index);
```

Returns

- true - If the BusIdle feature is supported on the device
- false - If the BusIdle feature is not supported on the device

Description

This function identifies whether the BusIdle feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_BusIsIdle

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsBusIsIdle(I2C_MODULE_ID index)

PLIB_I2C_ExistsClockStretching Function

Identifies whether the ClockStretching feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsClockStretching(I2C_MODULE_ID index);
```

Returns

- true - If the ClockStretching feature is supported on the device
- false - If the ClockStretching feature is not supported on the device

Description

This function identifies whether the ClockStretching feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveClockStretchingEnable
- PLIB_I2C_SlaveClockStretchingDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsClockStretching(I2C_MODULE_ID index)

PLIB_I2C_ExistsDataLineHoldTime Function

Identifies whether the DataLineHoldTime feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsDataLineHoldTime(I2C_MODULE_ID index);
```

Returns

- true - If the DataLineHoldTime feature is supported on the device
- false - If the DataLineHoldTime feature is not supported on the device

Description

This function identifies whether the DataLineHoldTime feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_DataLineHoldTimeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsDataLineHoldTime(I2C_MODULE_ID index)

PLIB_I2C_ExistsGeneralCall Function

Identifies whether the GeneralCall feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsGeneralCall(I2C_MODULE_ID index);
```

Returns

- true - If the GeneralCall feature is supported on the device
- false - If the GeneralCall feature is not supported on the device

Description

This function identifies whether the GeneralCall feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_GeneralCallEnable
- PLIB_I2C_GeneralCallDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsGeneralCall(I2C_MODULE_ID index)

PLIB_I2C_ExistsGeneralCallAddressDetect Function

Identifies whether the GeneralCallAddressDetect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsGeneralCallAddressDetect(I2C_MODULE_ID index);
```

Returns

- true - If the GeneralCallAddressDetect feature is supported on the device
- false - If the GeneralCallAddressDetect feature is not supported on the device

Description

This function identifies whether the GeneralCallAddressDetect feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_SlaveAddressIsGeneralCall

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsGeneralCallAddressDetect(I2C_MODULE_ID index)

PLIB_I2C_ExistsHighFrequency Function

Identifies whether the HighFrequency feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsHighFrequency(I2C_MODULE_ID index);
```

Returns

- true - If the HighFrequency feature is supported on the device
- false - If the HighFrequency feature is not supported on the device

Description

This function identifies whether the HighFrequency feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_HighFrequencyEnable
- PLIB_I2C_HighFrequencyDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsHighFrequency(I2C_MODULE_ID index)

PLIB_I2C_ExistsIPMI Function

Identifies whether the IPMI feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsIPMI(I2C_MODULE_ID index);
```

Returns

- true - If the IPMI feature is supported on the device
- false - If the IPMI feature is not supported on the device

Description

This function identifies whether the IPMI feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_IPMIEnable
- PLIB_I2C_IPMIDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsIPMI(I2C_MODULE_ID index)

PLIB_I2C_ExistsMasterReceiverClock1Byte Function

Identifies whether the MasterReceiverClock1Byte feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsMasterReceiverClock1Byte(I2C_MODULE_ID index);
```

Returns

- true - If the MasterReceiverClock1Byte feature is supported on the device
- false - If the MasterReceiverClock1Byte feature is not supported on the device

Description

This function identifies whether the MasterReceiverClock1Byte feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_MasterReceiverClock1Byte

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsMasterReceiverClock1Byte(I2C_MODULE_ID index)

PLIB_I2C_ExistsMasterStart Function

Identifies whether the MasterStart feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsMasterStart(I2C_MODULE_ID index);
```

Returns

- true - If the MasterStart feature is supported on the device
- false - If the MasterStart feature is not supported on the device

Description

This function identifies whether the MasterStart feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_MasterStart

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsMasterStart(I2C_MODULE_ID index)

PLIB_I2C_ExistsMasterStartRepeat Function

Identifies whether the MasterStartRepeat feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsMasterStartRepeat(I2C_MODULE_ID index);
```

Returns

- true - If the MasterStartRepeat feature is supported on the device
- false - If the MasterStartRepeat feature is not supported on the device

Description

This function identifies whether the MasterStartRepeat feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_MasterStartRepeat

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsMasterStartRepeat(I2C_MODULE_ID index)

PLIB_I2C_ExistsMasterStop Function

Identifies whether the MasterStop feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsMasterStop(I2C_MODULE_ID index);
```

Returns

- true - If the MasterStop feature is supported on the device
- false - If the MasterStop feature is not supported on the device

Description

This function identifies whether the MasterStop feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_MasterStop

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsMasterStop(I2C_MODULE_ID index)

PLIB_I2C_ExistsModuleEnable Function

Identifies whether the ModuleEnable feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsModuleEnable(I2C_MODULE_ID index);
```

Returns

- true - If the ModuleEnable feature is supported on the device
- false - If the ModuleEnable feature is not supported on the device

Description

This function identifies whether the ModuleEnable feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_Enable
- PLIB_I2C_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsModuleEnable(I2C_MODULE_ID index)

PLIB_I2C_ExistsReceivedByteAcknowledge Function

Identifies whether the ReceivedByteAcknowledge feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsReceivedByteAcknowledge(I2C_MODULE_ID index);
```

Returns

- true - If the ReceivedByteAcknowledge feature is supported on the device
- false - If the ReceivedByteAcknowledge feature is not supported on the device

Description

This function identifies whether the ReceivedByteAcknowledge feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_ReceivedByteAcknowledge
- PLIB_I2C_ReceiverByteAcknowledgeHasCompleted
- PLIB_I2C_MasterReceiverReadyToAcknowledge

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsReceivedByteAcknowledge(I2C_MODULE_ID index)

PLIB_I2C_ExistsReceivedByteAvailable Function

Identifies whether the ReceivedByteAvailable feature exists on the I2C module

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsReceivedByteAvailable(I2C_MODULE_ID index);
```

Returns

- true - If the ReceivedByteAvailable feature is supported on the device
- false - If the ReceivedByteAvailable feature is not supported on the device

Description

This function identifies whether the ReceivedByteAvailable feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_ReceivedByteIsAvailable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsReceivedByteAvailable(I2C_MODULE_ID index)

PLIB_I2C_ExistsReceivedByteGet Function

Identifies whether the ReceivedByteGet feature exists on the I2C module

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsReceivedByteGet(I2C_MODULE_ID index);
```

Returns

- true - If the ReceivedByteGet feature is supported on the device
- false - If the ReceivedByteGet feature is not supported on the device

Description

This function identifies whether the ReceivedByteGet feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_ReceivedByteGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsReceivedByteGet(I2C_MODULE_ID index)

PLIB_I2C_ExistsReceiverOverflow Function

Identifies whether the ReceiverOverflow feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsReceiverOverflow(I2C_MODULE_ID index);
```

Returns

- true - If the ReceiverOverflow feature is supported on the device
- false - If the ReceiverOverflow feature is not supported on the device

Description

This function identifies whether the ReceiverOverflow feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_ReceiverOverflowHasOccurred
- PLIB_I2C_ReceiverOverflowClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsReceiverOverflow(I2C_MODULE_ID index)

PLIB_I2C_ExistsReservedAddressProtect Function

Identifies whether the ReservedAddressProtect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsReservedAddressProtect(I2C_MODULE_ID index);
```

Returns

- true - If the ReservedAddressProtect feature is supported on the device
- false - If the ReservedAddressProtect feature is not supported on the device

Description

This function identifies whether the ReservedAddressProtect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_ReservedAddressProtectEnable
- PLIB_I2C_ReservedAddressProtectDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsReservedAddressProtect(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveAddress10Bit Function

Identifies whether the SlaveAddress10Bit feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveAddress10Bit(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveAddress10Bit feature is supported on the device
- false - If the SlaveAddress10Bit feature is not supported on the device

Description

This function identifies whether the SlaveAddress10Bit feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveAddress10BitSet
- PLIB_I2C_SlaveAddress10BitGet
- PLIB_I2C_SlaveAddressModeIs10Bits

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveAddress10Bit(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveAddress7Bit Function

Identifies whether the SlaveAddress7Bit feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveAddress7Bit(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveAddress7Bit feature is supported on the device
- false - If the SlaveAddress7Bit feature is not supported on the device

Description

This function identifies whether the SlaveAddress7Bit feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveAddress7BitSet
- PLIB_I2C_SlaveAddress7BitGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveAddress7Bit(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveAddressDetect Function

Identifies whether the SlaveAddressDetect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveAddressDetect(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveAddressDetect feature is supported on the device
- false - If the SlaveAddressDetect feature is not supported on the device

Description

This function identifies whether the SlaveAddressDetect feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_SlaveAddress10BitWasDetected

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveAddressDetect(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveAddressHoldEnable Function

Identifies whether the SlaveAddressHoldEnable feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveAddressHoldEnable(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveAddressHoldEnable feature is supported on the device
- false - If the SlaveAddressHoldEnable feature is not supported on the device

Description

This function identifies whether the SlaveAddressHoldEnable feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveAddressHoldEnable
- PLIB_I2C_SlaveAddressHoldDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveAddressHoldEnable(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveBufferOverwrite Function

Identifies whether the SlaveBufferOverwrite feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveBufferOverwrite(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveBufferOverwrite feature is supported on the device
- false - If the SlaveBufferOverwrite feature is not supported on the device

Description

This function identifies whether the SlaveBufferOverwrite feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveBufferOverwriteEnable
- PLIB_I2C_SlaveBufferOverwriteDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveBufferOverwrite(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveBusCollisionDetect Function

Identifies whether the SlaveBusCollisionDetect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveBusCollisionDetect(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveBusCollisionDetect feature is supported on the device
- false - If the SlaveBusCollisionDetect feature is not supported on the device

Description

This function identifies whether the SlaveBusCollisionDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveBusCollisionDetectEnable
- PLIB_I2C_SlaveBusCollisionDetectDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveBusCollisionDetect(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveClockHold Function

Identifies whether the SlaveClockHold feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveClockHold(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveClockHold feature is supported on the device
- false - If the SlaveClockHold feature is not supported on the device

Description

This function identifies whether the SlaveClockHold feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveClockHold
- PLIB_I2C_SlaveClockRelease

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveClockHold(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveDataDetect Function

Identifies whether the SlaveDataDetect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveDataDetect(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveDataDetect feature is supported on the device
- false - If the SlaveDataDetect feature is not supported on the device

Description

This function identifies whether the SlaveDataDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveDataIsDetected
- PLIB_I2C_SlaveAddressIsDetected

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveDataDetect(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveInterruptOnStart Function

Identifies whether the SlaveInterruptOnStart feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveInterruptOnStart(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveInterruptOnStart feature is supported on the device
- false - If the SlaveInterruptOnStart feature is not supported on the device

Description

This function identifies whether the SlaveInterruptOnStart feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveInterruptOnStartEnable
- PLIB_I2C_SlaveInterruptOnStartDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveInterruptOnStart(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveInterruptOnStop Function

Identifies whether the SlaveInterruptOnStop feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveInterruptOnStop(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveInterruptOnStop feature is supported on the device
- false - If the SlaveInterruptOnStop feature is not supported on the device

Description

This function identifies whether the SlaveInterruptOnStop feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveInterruptOnStopEnable
- PLIB_I2C_SlaveInterruptOnStopDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveInterruptOnStop(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveMask Function

Identifies whether the SlaveMask feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveMask(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveMask feature is supported on the device
- false - If the SlaveMask feature is not supported on the device

Description

This function identifies whether the SlaveMask feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveMask7BitSet
- PLIB_I2C_SlaveMask7BitGet
- PLIB_I2C_SlaveMask10BitSet
- PLIB_I2C_SlaveMask10BitGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveMask(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveReadRequest Function

Identifies whether the SlaveReadRequest feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveReadRequest(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveReadRequest feature is supported on the device
- false - If the SlaveReadRequest feature is not supported on the device

Description

This function identifies whether the SlaveReadRequest feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_SlaveReadIsRequested

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveReadRequest(I2C_MODULE_ID index)

PLIB_I2C_ExistsSMBus Function

Identifies whether the SMBus feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSMBus(I2C_MODULE_ID index);
```

Returns

- true - If the SMBus feature is supported on the device
- false - If the SMBus feature is not supported on the device

Description

This function identifies whether the SMBus feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SMBEnable
- PLIB_I2C_SMBDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSMBus(I2C_MODULE_ID index)

PLIB_I2C_ExistsStartDetect Function

Identifies whether the StartDetect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsStartDetect(I2C_MODULE_ID index);
```

Returns

- true - If the StartDetect feature is supported on the device
- false - If the StartDetect feature is not supported on the device

Description

This function identifies whether the StartDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_StartWasDetected
- PLIB_I2C_StartClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsStartDetect(I2C_MODULE_ID index)

PLIB_I2C_ExistsStopDetect Function

Identifies whether the StopDetect feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsStopDetect(I2C_MODULE_ID index);
```

Returns

- true - If the StopDetect feature is supported on the device
- false - If the StopDetect feature is not supported on the device

Description

This function identifies whether the StopDetect feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_StopWasDetected
- PLIB_I2C_StopClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsStopDetect(I2C_MODULE_ID index)

PLIB_I2C_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsStopInIdle(I2C_MODULE_ID index);
```

Returns

- true - If the StopInIdle feature is supported on the device
- false - If the StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_StopInIdleEnable
- PLIB_I2C_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsStopInIdle(I2C_MODULE_ID index)

PLIB_I2C_ExistsTransmitterByteAcknowledge Function

Identifies whether the TransmitterByteAcknowledge feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsTransmitterByteAcknowledge(I2C_MODULE_ID index);
```

Returns

- true - If the TransmitterByteAcknowledge feature is supported on the device
- false - If the TransmitterByteAcknowledge feature is not supported on the device

Description

This function identifies whether the TransmitterByteAcknowledge feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_TransmitterByteWasAcknowledged

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsTransmitterByteAcknowledge(I2C_MODULE_ID index)

PLIB_I2C_ExistsTransmitterByteComplete Function

Identifies whether the TransmitterByteComplete feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsTransmitterByteComplete(I2C_MODULE_ID index);
```

Returns

- true - If the TransmitterByteComplete feature is supported on the device
- false - If the TransmitterByteComplete feature is not supported on the device

Description

This function identifies whether the TransmitterByteComplete feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_TransmitterByteHasCompleted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsTransmitterByteComplete(I2C_MODULE_ID index)

PLIB_I2C_ExistsTransmitterByteSend Function

Identifies whether the TransmitterByteSend feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsTransmitterByteSend(I2C_MODULE_ID index);
```

Returns

- true - If the TransmitterByteSend feature is supported on the device
- false - If the TransmitterByteSend feature is not supported on the device

Description

This function identifies whether the TransmitterByteSend feature is available on the I2C module. When this function returns true, this function is supported on the device:

- PLIB_I2C_TransmitterByteSend

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsTransmitterByteSend(I2C_MODULE_ID index)

PLIB_I2C_ExistsTransmitterIsBusy Function

Identifies whether the TransmitterBusy feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsTransmitterIsBusy(I2C_MODULE_ID index);
```

Returns

- true - If the TransmitterBusy feature is supported on the device
- false - If the TransmitterBusy feature is not supported on the device

Description

This function identifies whether the TransmitterBusy feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_TransmitterIsBusy
- PLIB_I2C_TransmitterIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsTransmitterIsBusy(I2C_MODULE_ID index)

PLIB_I2C_ExistsTransmitterOverflow Function

Identifies whether the TransmitterOverflow feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsTransmitterOverflow(I2C_MODULE_ID index);
```

Returns

- true - If the TransmitterOverflow feature is supported on the device
- false - If the TransmitterOverflow feature is not supported on the device

Description

This function identifies whether the TransmitterOverflow feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_TransmitterOverflowHasOccurred
- PLIB_I2C_TransmitterOverflowClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsTransmitterOverflow(I2C_MODULE_ID index)

PLIB_I2C_ExistsSlaveDataHoldEnable Function

Identifies whether the SlaveDataHoldEnable feature exists on the I2C module.

File

plib_i2c.h

C

```
bool PLIB_I2C_ExistsSlaveDataHoldEnable(I2C_MODULE_ID index);
```

Returns

- true - If the SlaveDataHoldEnable feature is supported on the device
- false - If the SlaveDataHoldEnable feature is not supported on the device

Description

This function identifies whether the SlaveDataHoldEnable feature is available on the I2C module. When this function returns true, these functions are supported on the device:

- PLIB_I2C_SlaveDataHoldEnable
- PLIB_I2C_SlaveDataHoldDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_I2C_ExistsSlaveDataHoldEnable(I2C_MODULE_ID index)

k) Data Types and Constants

I2C_MODULE_ID Enumeration

File

plib_i2c_help.h

C

```
typedef enum {
  I2C_ID_1,
  I2C_ID_2,
  I2C_ID_3,
  I2C_ID_4,
  I2C_ID_5,
  I2C_NUMBER_OF_MODULES
} I2C_MODULE_ID;
```

Members

Members	Description
I2C_ID_1	I2C Module 1 ID
I2C_ID_2	I2C Module 2 ID
I2C_ID_3	I2C Module 3 ID
I2C_ID_4	I2C Module 4 ID
I2C_ID_5	I2C Module 5 ID
I2C_NUMBER_OF_MODULES	Number of available I2C modules.

Description

This is type I2C_MODULE_ID.

Files

Files

Name	Description
plib_i2c.h	This file contains the interface definition for the I2C Peripheral Library.
plib_i2c_help.h	Identifies the supported I2C modules.

Description

This section lists the source and header files used by the library.

plib_i2c.h

This file contains the interface definition for the I2C Peripheral Library.

Functions

	Name	Description
	PLIB_I2C_AcksequenceIsInProgress	Checks whether the acknowledge sequence is in progress or it is completed.
	PLIB_I2C_ArbitrationLossClear	Clears the arbitration loss status flag
	PLIB_I2C_ArbitrationLossHasOccurred	Identifies if bus arbitration has been lost.
	PLIB_I2C_BaudRateGet	Calculates the I2C module's current SCL clock frequency.
	PLIB_I2C_BaudRateSet	Sets the desired baud rate.
	PLIB_I2C_BusIsIdle	Determines whether the I2C bus is idle or busy.
	PLIB_I2C_DataLineHoldTimeSet	Sets the data line hold time.
	PLIB_I2C_Disable	Disables the specified I2C module.
	PLIB_I2C_Enable	Enables the specified I2C module.
	PLIB_I2C_ExistsAcksequenceProgress	Identifies whether the AcksequenceIsInProgress feature exists on the I2C module.

	PLIB_I2C_ExistsArbitrationLoss	Identifies whether the ArbitrationLoss feature exists on the I2C module.
	PLIB_I2C_ExistsBaudRate	Identifies whether the BaudRate feature exists on the I2C module.
	PLIB_I2C_ExistsBusIsIdle	Identifies whether the BusIdle feature exists on the I2C module.
	PLIB_I2C_ExistsClockStretching	Identifies whether the ClockStretching feature exists on the I2C module.
	PLIB_I2C_ExistsDataLineHoldTime	Identifies whether the DataLineHoldTime feature exists on the I2C module.
	PLIB_I2C_ExistsGeneralCall	Identifies whether the GeneralCall feature exists on the I2C module.
	PLIB_I2C_ExistsGeneralCallAddressDetect	Identifies whether the GeneralCallAddressDetect feature exists on the I2C module.
	PLIB_I2C_ExistsHighFrequency	Identifies whether the HighFrequency feature exists on the I2C module.
	PLIB_I2C_ExistsIPMI	Identifies whether the IPMI feature exists on the I2C module.
	PLIB_I2C_ExistsMasterReceiverClock1Byte	Identifies whether the MasterReceiverClock1Byte feature exists on the I2C module.
	PLIB_I2C_ExistsMasterStart	Identifies whether the MasterStart feature exists on the I2C module.
	PLIB_I2C_ExistsMasterStartRepeat	Identifies whether the MasterStartRepeat feature exists on the I2C module.
	PLIB_I2C_ExistsMasterStop	Identifies whether the MasterStop feature exists on the I2C module.
	PLIB_I2C_ExistsModuleEnable	Identifies whether the ModuleEnable feature exists on the I2C module.
	PLIB_I2C_ExistsReceivedByteAcknowledge	Identifies whether the ReceivedByteAcknowledge feature exists on the I2C module.
	PLIB_I2C_ExistsReceivedByteAvailable	Identifies whether the ReceivedByteAvailable feature exists on the I2C module
	PLIB_I2C_ExistsReceivedByteGet	Identifies whether the ReceivedByteGet feature exists on the I2C module
	PLIB_I2C_ExistsReceiverOverflow	Identifies whether the ReceiverOverflow feature exists on the I2C module.
	PLIB_I2C_ExistsReservedAddressProtect	Identifies whether the ReservedAddressProtect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddress10Bit	Identifies whether the SlaveAddress10Bit feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddress7Bit	Identifies whether the SlaveAddress7Bit feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddressDetect	Identifies whether the SlaveAddressDetect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveAddressHoldEnable	Identifies whether the SlaveAddressHoldEnable feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveBufferOverwrite	Identifies whether the SlaveBufferOverwrite feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveBusCollisionDetect	Identifies whether the SlaveBusCollisionDetect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveClockHold	Identifies whether the SlaveClockHold feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveDataDetect	Identifies whether the SlaveDataDetect feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveDataHoldEnable	Identifies whether the SlaveDataHoldEnable feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveInterruptOnStart	Identifies whether the SlaveInterruptOnStart feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveInterruptOnStop	Identifies whether the SlaveInterruptOnStop feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveMask	Identifies whether the SlaveMask feature exists on the I2C module.
	PLIB_I2C_ExistsSlaveReadRequest	Identifies whether the SlaveReadRequest feature exists on the I2C module.
	PLIB_I2C_ExistsSMBus	Identifies whether the SMBus feature exists on the I2C module.
	PLIB_I2C_ExistsStartDetect	Identifies whether the StartDetect feature exists on the I2C module.
	PLIB_I2C_ExistsStopDetect	Identifies whether the StopDetect feature exists on the I2C module.
	PLIB_I2C_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterByteAcknowledge	Identifies whether the TransmitterByteAcknowledge feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterByteComplete	Identifies whether the TransmitterByteComplete feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterByteSend	Identifies whether the TransmitterByteSend feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterIsBusy	Identifies whether the TransmitterBusy feature exists on the I2C module.
	PLIB_I2C_ExistsTransmitterOverflow	Identifies whether the TransmitterOverflow feature exists on the I2C module.
	PLIB_I2C_GeneralCallDisable	Disables the I2C module from recognizing the general call address.
	PLIB_I2C_GeneralCallEnable	Enables the I2C module to recognize the general call address.
	PLIB_I2C_HighFrequencyDisable	Disables the I2C module from using high frequency (400 kHz or 1 MHz) signaling.
	PLIB_I2C_HighFrequencyEnable	Enables the I2C module to use high frequency (400 kHz or 1 MHz) signaling.
	PLIB_I2C_IPMIDisable	Disables the I2C module's support for the IPMI specification

	PLIB_I2C_IPMIEnable	Enables the I2C module to support the Intelligent Platform Management Interface (IPMI) specification (see Remarks).
	PLIB_I2C_MasterReceiverClock1Byte	Drives the bus clock to receive 1 byte of data from a slave device.
	PLIB_I2C_MasterReceiverReadyToAcknowledge	Checks whether the hardware is ready to acknowledge.
	PLIB_I2C_MasterStart	Sends an I2C Start condition on the I2C bus in Master mode.
	PLIB_I2C_MasterStartRepeat	Sends a repeated Start condition during an ongoing transfer in Master mode.
	PLIB_I2C_MasterStop	Sends an I2C Stop condition to terminate a transfer in Master mode.
	PLIB_I2C_ReceivedByteAcknowledge	Allows a receiver to acknowledge a that a byte of data has been received.
	PLIB_I2C_ReceivedByteGet	Gets a byte of data received from the I2C bus interface.
	PLIB_I2C_ReceivedByteIsAvailable	Detects whether the receiver has data available.
	PLIB_I2C_ReceiverByteAcknowledgeHasCompleted	Determines if the previous acknowledge has completed.
	PLIB_I2C_ReceiverOverflowClear	Clears the receiver overflow status flag.
	PLIB_I2C_ReceiverOverflowHasOccurred	Identifies if a receiver overflow error has occurred.
	PLIB_I2C_ReservedAddressProtectDisable	Disables the I2C module from protecting reserved addresses, allowing it to respond to them.
	PLIB_I2C_ReservedAddressProtectEnable	Enables the I2C module to protect (not respond to) reserved addresses.
	PLIB_I2C_SlaveAddress10BitGet	Identifies the current 10-bit Slave mode address.
	PLIB_I2C_SlaveAddress10BitSet	Selects 10-bit Slave mode addressing and sets the address value.
	PLIB_I2C_SlaveAddress10BitWasDetected	Detects reception of the second byte of a 10-bit slave address.
	PLIB_I2C_SlaveAddress7BitGet	Identifies the current 7-bit Slave mode address.
	PLIB_I2C_SlaveAddress7BitSet	Selects 7-bit Slave mode addressing and sets the slave address value.
	PLIB_I2C_SlaveAddressHoldDisable	Disables Address holding.
	PLIB_I2C_SlaveAddressHoldEnable	Enables address holding.
	PLIB_I2C_SlaveAddressIsDetected	Detects if the most recent byte received is a data or an address byte.
	PLIB_I2C_SlaveAddressIsGeneralCall	Identifies if the current slave address received is the general call address.
	PLIB_I2C_SlaveAddressModeIs10Bits	Identifies if the current slave address mode is 7-bits or 10-bits.
	PLIB_I2C_SlaveBufferOverwriteDisable	Disables buffer overwrite.
	PLIB_I2C_SlaveBufferOverwriteEnable	Enables buffer overwrite.
	PLIB_I2C_SlaveBusCollisionDetectDisable	Disables bus collision detect interrupts.
	PLIB_I2C_SlaveBusCollisionDetectEnable	Enables slave bus collision interrupts.
	PLIB_I2C_SlaveClockHold	Holds the SCL clock line low after receiving a byte.
	PLIB_I2C_SlaveClockRelease	Releases a previously held SCL clock line.
	PLIB_I2C_SlaveClockStretchingDisable	Disables the I2C module from stretching the slave clock.
	PLIB_I2C_SlaveClockStretchingEnable	Enables the I2C module to stretch the slave clock.
	PLIB_I2C_SlaveDataHoldDisable	Disables data holding.
	PLIB_I2C_SlaveDataHoldEnable	Enables data holding.
	PLIB_I2C_SlaveDataIsDetected	Detects if the most recent byte received is a data or an address byte.
	PLIB_I2C_SlaveInterruptOnStartDisable	Disables the feature of generating interrupt on start condition.
	PLIB_I2C_SlaveInterruptOnStartEnable	Enables the feature of generating interrupt on start condition.
	PLIB_I2C_SlaveInterruptOnStopDisable	Disables the feature of generating interrupt on stop condition.
	PLIB_I2C_SlaveInterruptOnStopEnable	Enables the feature of generating interrupt on stop condition.
	PLIB_I2C_SlaveMask10BitGet	Identifies the current 10-bit Slave mode address mask.
	PLIB_I2C_SlaveMask10BitSet	Sets the 10-bit mask for Slave mode addressing.
	PLIB_I2C_SlaveMask7BitGet	Identifies the current 7-bit Slave mode address mask.
	PLIB_I2C_SlaveMask7BitSet	Sets the 7-bit mask for Slave mode addressing .
	PLIB_I2C_SlaveReadIsRequested	Detects if the request from the master was a read or write.
	PLIB_I2C_SMBDisable	Disable the I2C module support for SMBus electrical signaling levels.
	PLIB_I2C_SMBEnable	Enables the I2C module to support System Management Bus (SMBus) electrical signaling levels.
	PLIB_I2C_StartClear	Clears the start status flag
	PLIB_I2C_StartWasDetected	Identifies when a Start condition has been detected.
	PLIB_I2C_StopClear	Clears the stop status flag
	PLIB_I2C_StopInIdleDisable	Disables the Stop-in-Idle feature.
	PLIB_I2C_StopInIdleEnable	Enables the I2C module to stop when the processor enters Idle mode
	PLIB_I2C_StopWasDetected	Identifies when a Stop condition has been detected
	PLIB_I2C_TransmitterByteHasCompleted	Detects whether the module has finished transmitting the most recent byte.

	PLIB_I2C_TransmitterByteSend	Sends a byte of data on the I2C bus.
	PLIB_I2C_TransmitterByteWasAcknowledged	Determines whether the most recently sent byte was acknowledged.
	PLIB_I2C_TransmitterIsBusy	Identifies if the transmitter of the specified I2C module is currently busy (unable to accept more data).
	PLIB_I2C_TransmitterIsReady	Detects if the transmitter is ready to accept data to transmit.
	PLIB_I2C_TransmitterOverflowClear	Clears the transmitter overflow status flag.
	PLIB_I2C_TransmitterOverflowHasOccurred	Identifies if a transmitter overflow error has occurred.

Description

I2C Peripheral Library Interface Header

This library provides a low-level abstraction of the Inter-Integrated Circuit (I2C) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

File Name

plib_i2c.h

Company

Microchip Technology Inc.

plib_i2c_help.h

Identifies the supported I2C modules.

Enumerations

	Name	Description
	I2C_MODULE_ID	This is type I2C_MODULE_ID.

Description

I2C Module ID

This enumeration identifies the available I2C modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers specified in the data sheet.

Input Capture Peripheral Library

This section describes the Input Capture Peripheral Library.

Introduction

This library provides a low-level abstraction of the Input Capture (IC) Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The Input Capture (IC) module is used to capture a timer value on the occurrence of an event on an input pin. The IC module plays a major role in applications requiring frequency measurement.

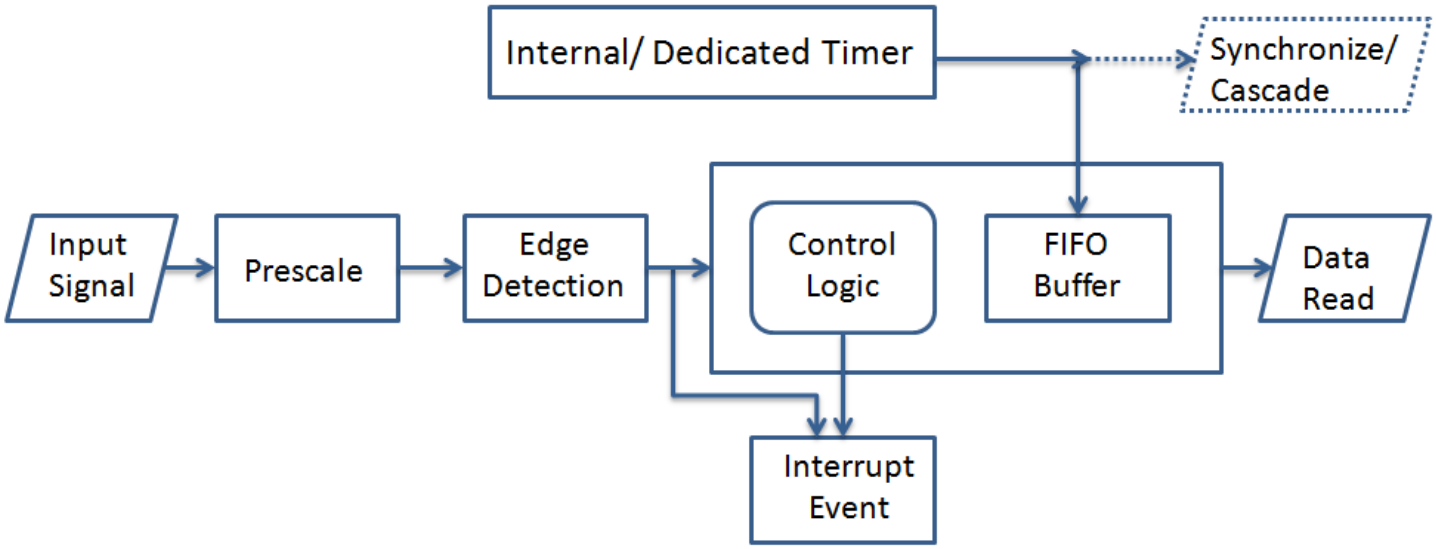


Figure 1: Input Capture Module

Figure 1 shows the functionality of an Input Capture module. An input signal is provided to the input of this module. The input may be prescaled. An edge detection block detects if the input is undergoing a rising or falling transition. Control logic undertakes the capture of the timer value depending upon the operating mode selected.

Some devices have Input Capture modules with dedicated timers that can be used to synchronize multiple IC modules together or cascade 16-bit timers to obtain a 32-bit timer. The timer may be triggered by an external, asynchronous input to start counting as well.

The IC module has multiple modes of operation which can be used for different applications. Operating modes include the following:

- Timer value capture at falling edge of input signal
- Timer value capture at rising edge of input signal
- Timer value capture at both edges of input signal
- Timer value capture on a specified edge of the input signal and every edge thereafter
- Timer value capture on every fourth or sixteenth rising edge of the input signal
- Interrupt-only mode in which Input Capture module acts as a source of external interrupt. This mode is functional only in device Sleep and Idle modes.

The IC module also incorporates a First In First Out (FIFO) buffer, which temporarily holds the captured value until it is read by the user application. The IC module is capable of generating interrupts when a set number of captures have taken place.

Using the Library

This topic describes the basic architecture of the Input Capture Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_ic.h`

The interface to the Input Capture Peripheral library is defined in the `plib_ic.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Input Capture Peripheral library must include `peripheral.h`.

Library File: The IC Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the Peripheral interacts with the framework.

Hardware Abstraction Model

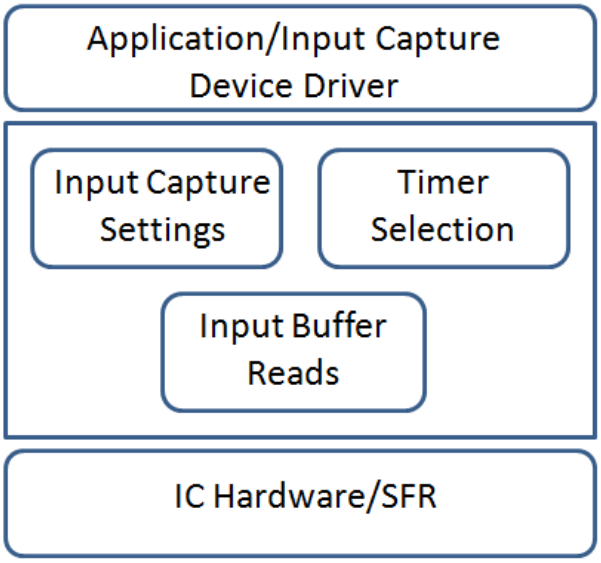
This library provides the low-level abstraction of the Input Capture (IC) module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all of the functionality of the available IC module on the device. Refer to the **"Input Capture"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each IC module on your device.

Description

The IC Peripheral Library is used to simplify low-level access to the IC module without having the need to directly interact with the module registers. The names of the functions are generic and lead to easier access to the IC module on different device variants.

Input Capture module Software Abstraction Block Diagram



The IC module can be described from a software standpoint as having functions to configure the module itself and select an intended mode of operation. Separate functions exist to select the timer whose value is captured by the IC module. When an external input event occurs, the IC module captures the timer value and stores it in a First In First Out (FIFO) buffer. The captured value can be read from the FIFO buffer by performing an input buffer read by calling the respective function.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Input Capture module.

Library Interface Section	Description
General Setup Functions	Provides setup, configuration, and status interface routines for the overall operation of the Input Capture module.
Timer Functions	Provides timer interface routines.
Input Capture Buffer Functions	Provides buffer interface routines.
Operation Modes Functions	Provides operation mode routines.
Power-Saving Modes Functions	Provides Power-Saving Modes routines.
Feature Existence Functions	Determine whether particular features exist on a device.

How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

All of the interface functions in this library are classified according to:

- General setup functions: These functions deal with general setup of the IC module
- Timer functions: These routines are used to select the timer, input for the timer, and configure synchronous or triggered operation for the timer
- Operation Modes routine: The PLIB_IC_ModeSelect function is used to select an operation mode of IC module
- Input capture buffer routines: These routines deal with reading the capture value stored in the IC buffer and performing status checks on the IC buffer
- Power-Saving modes:
- In Sleep mode, the IC module can function only as an external interrupt source. A rising edge input wakes the device from sleep. An interrupt will be generated if it is enabled for that module.
- In Idle mode, the IC module can be configured to either stop its operation or continue its operation
- Exists functions: These functions determine whether or not a particular feature is present on a device

Subsequent sections provide examples that depict the use of interface routines to perform general tasks such as IC module setup and reading captured values.

Input Capture Module Setup

This section shows the steps required to initialize and configure the IC module. The IC module is configured such that a capture event occurs at the first falling edge of the input signal, and then at every subsequent edge of the input signal.

Example:

```
/* This example shows how to initialize and configure IC module */
/* Disable IC module */
PLIB_IC_Disable(MY_IC_ID);

/* Disable any peripherals multiplexed with Input Capture pin */

/* Configure I/O pin as input pin. Refer PORTS Peripheral Library */

/* Stop in Idle mode is disabled for IC module. IC mode functions even in Idle mode */
PLIB_IC_StopInIdleDisable(MY_IC_ID);

/* Every Edge Capture mode of operation is selected for IC module */
PLIB_IC_ModeSelect(MY_IC_ID, IC_INPUT_CAPTURE_EVERY_EDGE_MODE);

/* Timer2 is selected as a clock source for the IC module  */
PLIB_IC_TimerSelect(MY_IC_ID, IC_TIMER_TMR2);

/* 32-bit timer is selected*/
PLIB_IC_BufferSizeSelect(MY_IC_ID, IC_BUFFER_SIZE_32BIT);

/* IC module is enabled */
PLIB_IC_Enable(MY_IC_ID);
```

Reading a Capture Value

This section shows how to read a capture value from the input capture buffer.

Example:

```
/* This example shows how to read a Capture value */
uint32_t bufferVal;

/* Reading a value from an empty Capture Buffer leads to indeterminate data*/
/* Read value from a non-empty buffer */
if(!PLIB_IC_BufferIsEmpty(MY_IC_ID))
{
     /* Read the buffer value */
     bufferVal = PLIB_IC_Buffer32BitGet(MY_IC_ID);
}

/* Buffer Overflow condition */
while(PLIB_IC_BufferOverflowHasOccurred)
{
     /*Read buffer values until buffer is empty and overflow condition ceases*/
     bufferVal = PLIB_IC_Buffer32BitGet(MY_IC_ID);

}
```

Configuring the Library

The library is configured for the supported Input Capture module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Setup Functions

	Name	Description
	PLIB_IC_Disable	Disables the IC module.
	PLIB_IC_Enable	Enables the IC module.

	PLIB_IC_EventsPerInterruptSelect	Selects the number of capture events before an interrupt is issued.
	PLIB_IC_FirstCaptureEdgeSelect	Captures the timer count value at the first selected edge of input signal.

b) Timer Functions

	Name	Description
	PLIB_IC_TimerSelect	Selects the clock source for the IC module.
	PLIB_IC_AlternateClockDisable	Selects Timer2 and Timer3, instead of the alternate clock source.
	PLIB_IC_AlternateClockEnable	Selects the alternate clock source.
	PLIB_IC_AlternateTimerSelect	Selects an alternate timer as a clock source for the IC module.
	PLIB_IC_ExistsAlternateTimerSelect	Identifies whether the AlternateTimerSelect feature exists on the IC module.

c) Input Capture Buffer Functions

	Name	Description
	PLIB_IC_Buffer16BitGet	Obtains the 16-bit input capture buffer value.
	PLIB_IC_Buffer32BitGet	Obtains the 32-bit input capture buffer value.
	PLIB_IC_BufferIsEmpty	Checks whether the input capture buffer is empty.
	PLIB_IC_BufferOverflowHasOccurred	Checks whether an input capture buffer overflow has occurred.
	PLIB_IC_BufferSizeSelect	Sets the input capture buffer size.

d) Operation Modes Functions

	Name	Description
	PLIB_IC_ModeSelect	Selects the input capture mode for IC module.

e) Power-Saving Modes Functions

	Name	Description
	PLIB_IC_StopInIdleDisable	Continues module operation when the device enters Idle mode
	PLIB_IC_StopInIdleEnable	Discontinues IC module operation when the device enters Idle mode.

f) Feature Existence Functions

	Name	Description
	PLIB_IC_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the IC module.
	PLIB_IC_ExistsBufferIsEmptyStatus	Identifies whether the BufferIsEmptyStatus feature exists on the IC module
	PLIB_IC_ExistsBufferOverflowStatus	Identifies whether the BufferOverflowStatus feature exists on the IC module.
	PLIB_IC_ExistsBufferSize	Identifies whether the BufferSize feature exists on the IC module.
	PLIB_IC_ExistsBufferValue	Identifies whether the BufferValue feature exists on the IC module.
	PLIB_IC_ExistsCaptureMode	Identifies whether the CaptureMode feature exists on the IC module.
	PLIB_IC_ExistsEdgeCapture	Identifies whether the EdgeCapture feature exists on the IC module.
	PLIB_IC_ExistsEnable	Identifies whether the EnableControl feature exists on the IC module
	PLIB_IC_ExistsEventsPerInterruptSelect	Identifies whether the EventsPerInterruptSelect feature exists on the IC module.
	PLIB_IC_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the IC module.
	PLIB_IC_ExistsTimerSelect	Identifies whether the TimerSelect feature exists on the IC module.

g) Data Types and Constants

	Name	Description
	IC_BUFFER_SIZE	Selects the IC Buffer size.
	IC_EDGE_TYPES	Lists the options to select a rising or falling edge for input capture.
	IC_EVENTS_PER_INTERRUPT	Lists the number of input capture events for an interrupt to occur.
	IC_INPUT_CAPTURE_MODES	Lists all of the input capture modes for the IC module.
	IC_MODULE_ID	IC_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on a Microchip microcontroller. Refer to the specific device data sheet to determine the correct number of modules defined for the device.
	IC_TIMERS	Lists the different 16-bit timers for the IC module.
	IC_ALT_TIMERS	Lists the different 16-bit alternate timers for the IC module.

Description

This section describes the Application Programming Interface (API) functions of the Input Capture Peripheral Library.

Refer to each section for a detailed description.

a) General Setup Functions

PLIB_IC_Disable Function

Disables the IC module.

File

plib_ic.h

C

```
void PLIB_IC_Disable(IC_MODULE_ID index);
```

Returns

None.

Description

This function disables the IC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_IC_ID IC_ID_1

PLIB_IC_Disable(MY_IC_ID);
```

Parameters

Parameters	Description
index	Identifies the IC module

Function

void PLIB_IC_Disable(IC_MODULE_ID index)

PLIB_IC_Enable Function

Enables the IC module.

File

plib_ic.h

C

```
void PLIB_IC_Enable(IC_MODULE_ID index);
```

Returns

None.

Description

This function enables the IC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsEnable in your application to determine whether this feature is available.

Preconditions

The module should be appropriately configured before being enabled.

Example

```
#define MY_IC_ID IC_ID_1

//Do all the other configurations before enabling.

PLIB_IC_Enable(MY_IC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

void PLIB_IC_Enable(IC_MODULE_ID index)

PLIB_IC_EventsPerInterruptSelect Function

Selects the number of capture events before an interrupt is issued.

File

plib_ic.h

C

```
void PLIB_IC_EventsPerInterruptSelect(IC_MODULE_ID index, IC_EVENTS_PER_INTERRUPT event);
```

Returns

None.

Description

The IC module can be configured to generate interrupts depending upon the occurrence of a certain number of capture events. The number of capture events before an interrupt is generated is set by this function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsEventsPerInterruptSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_IC_ID IC_ID_1
// IC module is configured to generate interrupt on every capture event
PLIB_IC_EventsPerInterruptSelect(MY_IC_ID, IC_INTERRUPT_ON_EVERY_CAPTURE_EVENT );
```

Parameters

Parameters	Description
index	Identifies the desired IC module
event	Identifies the interrupt control mode

Function

void PLIB_IC_EventsPerInterruptSelect(IC_MODULE_ID index,IC_EVENTS_PER_INTERRUPT event)

PLIB_IC_FirstCaptureEdgeSelect Function

Captures the timer count value at the first selected edge of input signal.

File

plib_ic.h

C

```
void PLIB_IC_FirstCaptureEdgeSelect(IC_MODULE_ID index, IC_EDGE_TYPES edgeType);
```

Returns

None.

Description

This function captures the timer count value at the first selected edge of the input signal.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsEdgeCapture in your application to determine whether this feature is available.

Preconditions

This setting applies only when the Every Edge Capture mode is set. The capture mode is set by the PLIB_IC_ModeSelect function.

Example

```
#define MY_IC_ID IC_ID_1

PLIB_IC_FirstCaptureEdgeSelect(MY_IC_ID,IC_EDGE_FALLING);
```

Parameters

Parameters	Description
index	Identifies the desired IC module
edgeType	Identifies the edge type (i.e., whether rising or falling edge)

Function

void PLIB_IC_FirstCaptureEdgeSelect(IC_MODULE_ID index, IC_EDGE_TYPES edgeType)

b) Timer Functions

PLIB_IC_TimerSelect Function

Selects the clock source for the IC module.

File

plib_ic.h

C

```
void PLIB_IC_TimerSelect(IC_MODULE_ID index, IC_TIMERS tmr);
```

Returns

None.

Description

This function selects the 16-bit timer source for the IC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsTimerSelect in your application to determine whether this feature is available.

Preconditions

The 16-bit time base needs to be set.

Example

```
#define MY_IC_ID IC_ID_1
// 16-bit Timer-2 is selected
PLIB_IC_TimerSelect(MY_IC_ID, IC_TIMER_TMR2);
```

Parameters

Parameters	Description
index	Identifies the desired IC module
tmr	Identifies the 16-bit timer

Function

void PLIB_IC_TimerSelect(IC_MODULE_ID index, IC_TIMERS tmr)

PLIB_IC_AlternateClockDisable Function

Selects Timer2 and Timer3, instead of the alternate clock source.

File

plib_ic.h

C

```
void PLIB_IC_AlternateClockDisable(IC_MODULE_ID index);
```

Returns

None.

Description

Selects Timer2 and Timer3, instead of the alternate clock source.

Remarks

The feature is not supported on all devices. Please refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed. This function applies to all input capture modules, regardless of the IC_MODULE_ID passed in the call.

Preconditions

A system unlock PLIB_DEVCON_SystemUnlock must be performed before this function can be executed.

Example

```
// Call system service to unlock oscillator
#define MY_IC_ID IC_ID_1
PLIB_IC_AlternateClockDisable( MY_IC_ID );
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

void PLIB_IC_AlternateClockDisable(IC_MODULE_ID index)

PLIB_IC_AlternateClockEnable Function

Selects the alternate clock source.

File

plib_ic.h

C

```
void PLIB_IC_AlternateClockEnable(IC_MODULE_ID index);
```

Returns

None.

Description

Selects the alternate clock source instead of Timer2/3.

Remarks

The feature is not supported on all devices. Please refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed. This function applies to all input capture modules, regardless of the IC_MODULE_ID passed in the call.

Preconditions

A system unlock PLIB_DEVCON_SystemUnlock must be performed before this function can be executed.

Example

```
// Call system service to unlock oscillator
#define MY_IC_ID IC_ID_1
PLIB_IC_AlternateClockEnable( MY_IC_ID );
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

void PLIB_IC_AlternateClockEnable(IC_MODULE_ID index)

PLIB_IC_AlternateTimerSelect Function

Selects an alternate timer as a clock source for the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_AlternateTimerSelect(IC_MODULE_ID index, IC_ALT_TIMERS tmr);
```

Returns

Boolean

- true - Alternate timer selected successfully.
- false - Alternate timer selection failure, select appropriate alternate timer for the IC module index.

Description

This function selects an alternate timer as a clock source for the IC module. IC_ID_1,IC_ID_2,IC_ID_3 : Can use Timer4 or Timer5 as alternate timers. IC_ID_4,IC_ID_5,IC_ID_6 : Can use Timer2 or Timer3 as alternate timers. IC_ID_7,IC_ID_8,IC_ID_9 : Can use Timer6 or Timer7 as alternate timers.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsAlternateTimerSelect in your application to determine whether this feature is available.

Preconditions

The 16-bit time base needs to be set. PLIB_IC_AlternateClockEnable API should be called for the IC module to enable the alternate clock selection.

Example

```
#define MY_IC_ID   IC_ID_1
bool result;

//Enabling alternate timer selection
// Call system service to unlock oscillator
PLIB_IC_AlternateClockEnable( MY_IC_ID );

// 16-bit Timer4 is selected as the clock source for IC module 1
result = PLIB_IC_AlternateTimerSelect(MY_IC_ID, IC_ALT_TIMER_TMR4);

if(false == result)
{
    //Selected alternate timer does not available for the desired IC module.
    //Select appropriate alternate timer.
```

```
}
```

Parameters

Parameters	Description
index	Identifies the desired IC module
tmr	Identifies the alternate timer

Function

bool PLIB_IC_AlternateTimerSelect(IC_MODULE_ID index, IC_ALT_TIMERS tmr)

PLIB_IC_ExistsAlternateTimerSelect Function

Identifies whether the AlternateTimerSelect feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsAlternateTimerSelect(IC_MODULE_ID index);
```

Returns

- true - If the AlternateTimerSelect feature is supported on the device
- false - If the AlternateTimerSelect feature is not supported on the device

Description

This function identifies whether the AlternateTimerSelect feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_AlternateTimerSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_IC_ExistsAlternateTimerSelect(IC_MODULE_ID index)

c) Input Capture Buffer Functions

PLIB_IC_Buffer16BitGet Function

Obtains the 16-bit input capture buffer value.

File

plib_ic.h

C

```
uint16_t PLIB_IC_Buffer16BitGet(IC_MODULE_ID index);
```

Returns

None.

Description

This function reads the 16-bit value of the IC buffer to obtain the captured timer value. This function is used when the buffer size is set to 16-bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsBufferValue in your application to determine whether this feature is available.

Preconditions

The buffer size should be set to 16 bits (i.e., PLIB_IC_BufferSizeSet(MY_IC_ID, IC_BUFFER_SIZE_16BIT).

Example

```
#define MY_IC_ID IC_ID_1
uint16_t bufferVal;
// Read input capture buffer value and store it in 'bufferVal'
bufferVal = PLIB_IC_Buffer16BitGet(MY_IC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

uint16_t PLIB_IC_Buffer16BitGet(IC_MODULE_ID index)

PLIB_IC_Buffer32BitGet Function

Obtains the 32-bit input capture buffer value.

File

plib_ic.h

C

```
uint32_t PLIB_IC_Buffer32BitGet(IC_MODULE_ID index);
```

Returns

None.

Description

This function reads the 32-bit value of the IC buffer to obtain the captured timer value. This function is used when the buffer size is set to 32 bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsBufferValue in your application to determine whether this feature is available.

Preconditions

The buffer size should be set to 32 bits (i.e., PLIB_IC_BufferSizeSet(MY_IC_ID, IC_BUFFER_SIZE_32BIT).

Example

```
#define MY_IC_ID IC_ID_1
uint32_t bufferVal;
// Read input capture buffer value and store it in 'bufferVal'
bufferVal = PLIB_IC_Buffer32BitGet(MY_IC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

uint32_t PLIB_IC_Buffer32BitGet(IC_MODULE_ID index)

PLIB_IC_BufferIsEmpty Function

Checks whether the input capture buffer is empty.

File

plib_ic.h

C

```
bool PLIB_IC_BufferIsEmpty(IC_MODULE_ID index);
```

Returns

Boolean

- true - The input capture buffer is empty
- false - The input capture buffer is not empty

Description

This function returns 'true' if the input capture buffer is empty and 'false' if the buffer is not empty.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsBufferIsEmptyStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_IC_ID IC_ID_1
if(!PLIB_IC_BufferIsEmpty(MY_IC_ID))
{
    // Do some operation
};
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

bool PLIB_IC_BufferIsEmpty(IC_MODULE_ID index)

PLIB_IC_BufferOverflowHasOccurred Function

Checks whether an input capture buffer overflow has occurred.

File

plib_ic.h

C

```
bool PLIB_IC_BufferOverflowHasOccurred(IC_MODULE_ID index);
```

Returns

Boolean

- true - An overflow of the input capture buffer has occurred
- false - An overflow of the input capture buffer has not occurred

Description

This function returns 'true' if an input capture buffer has overflowed and 'false' if the buffer has not overflowed.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsBufferOverflowStatus in your application to determine whether this feature is available.

Preconditions

This function only applies when not in Edge Detect mode.

Example

```
#define MY_IC_ID IC_ID_1
if(!PLIB_IC_BufferOverflowHasOccurred(MY_IC_ID))
{
    // Do some operation
};
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

bool PLIB_IC_BufferOverflowHasOccurred(IC_MODULE_ID index)

PLIB_IC_BufferSizeSelect Function

Sets the input capture buffer size.

File

plib_ic.h

C

```
void PLIB_IC_BufferSizeSelect(IC_MODULE_ID index, IC_BUFFER_SIZE bufSize);
```

Returns

None.

Description

This function sets the input capture buffer size for IC module. The buffer size can be set to 16 bits or 32 bits. The buffer size should be consistent with the timer selected. A 32-bit timer requires a 32-bit buffer and a 16-bit timer requires a 16-bit buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsBufferSize in your application to determine whether this feature is available.

Preconditions

The buffer size should be consistent with the timer selected.

Example

```
#define MY_IC_ID IC_ID_1
// 32-bit timer resource is selected
PLIB_IC_BufferSizeSelect(MY_IC_ID, IC_BUFFER_SIZE_32BIT);
```

Parameters

Parameters	Description
index	Identifies the desired IC module
bufSize	Sets the buffer size to 16 bits or 32 bits

Function

void PLIB_IC_BufferSizeSelect(IC_MODULE_ID index, IC_BUFFER_SIZE bufSize)

d) Operation Modes Functions

PLIB_IC_ModeSelect Function

Selects the input capture mode for IC module.

File

plib_ic.h

C

```
void PLIB_IC_ModeSelect(IC_MODULE_ID index, IC_INPUT_CAPTURE_MODES modeSel);
```

Returns

None.

Description

This function selects the input capture mode for the IC module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsCaptureMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_IC_ID IC_ID_1
// Every Edge Mode is selected
PLIB_IC_ModeSelect(MY_IC_ID, IC_INPUT_CAPTURE_EVERY_EDGE_MODE);
```

Parameters

Parameters	Description
index	Identifies the desired IC module
modeSel	Identifies the timer type required

Function

void PLIB_IC_ModeSelect(IC_MODULE_ID index, IC_INPUT_CAPTURE_MODES modeSel)

e) Power-Saving Modes Functions

PLIB_IC_StopInIdleDisable Function

Continues module operation when the device enters Idle mode

File

plib_ic.h

C

```
void PLIB_IC_StopInIdleDisable(IC_MODULE_ID index);
```

Returns

None.

Description

The function continues operation of the IC module when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_IC_ID IC_ID_1

PLIB_IC_StopInIdleDisable(MY_IC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

void PLIB_IC_StopInIdleDisable(IC_MODULE_ID index)

PLIB_IC_StopInIdleEnable Function

Discontinues IC module operation when the device enters Idle mode.

File

plib_ic.h

C

```
void PLIB_IC_StopInIdleEnable(IC_MODULE_ID index);
```

Returns

None.

Description

This function discontinues IC module operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_IC_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_IC_ID IC_ID_1

PLIB_IC_StopInIdleEnable(MY_IC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired IC module

Function

void PLIB_IC_StopInIdleEnable(IC_MODULE_ID index)

f) Feature Existence Functions

PLIB_IC_ExistsAlternateClock Function

Identifies whether the AlternateClock feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsAlternateClock(IC_MODULE_ID index);
```

Returns

- true - The AlternateClock feature is supported on the device
- false - The AlternateClock feature is not supported on the device

Description

This function identifies whether the AlternateClock feature is available on the IC module. When this function returns true, these functions are supported on the device:

- PLIB_IC_AlternateClockEnable
- PLIB_IC_AlternateClockDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsAlternateClock(IC_MODULE_ID index)

PLIB_IC_ExistsBufferIsEmptyStatus Function

Identifies whether the BufferIsEmptyStatus feature exists on the IC module

File

plib_ic.h

C

```
bool PLIB_IC_ExistsBufferIsEmptyStatus(IC_MODULE_ID index);
```

Returns

- true - The BufferIsEmptyStatus feature is supported on the device
- false - The BufferIsEmptyStatus feature is not supported on the device

Description

This function identifies whether the BufferIsEmptyStatus feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_BufferIsEmpty

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsBufferIsEmptyStatus(IC_MODULE_ID index)

PLIB_IC_ExistsBufferOverflowStatus Function

Identifies whether the BufferOverflowStatus feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsBufferOverflowStatus(IC_MODULE_ID index);
```

Returns

- true - The BufferOverflowStatus feature is supported on the device
- false - The BufferOverflowStatus feature is not supported on the device

Description

This function identifies whether the BufferOverflowStatus feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_BufferOverflowHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsBufferOverflowStatus(IC_MODULE_ID index)

PLIB_IC_ExistsBufferSize Function

Identifies whether the BufferSize feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsBufferSize(IC_MODULE_ID index);
```

Returns

- true - The BufferSize feature is supported on the device
- false - The BufferSize feature is not supported on the device

Description

This function identifies whether the BufferSize feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_BufferSizeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsBufferSize(IC_MODULE_ID index)

PLIB_IC_ExistsBufferValue Function

Identifies whether the BufferValue feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsBufferValue(IC_MODULE_ID index);
```

Returns

- true - The BufferValue feature is supported on the device
- false - The BufferValue feature is not supported on the device

Description

This function identifies whether the BufferValue feature is available on the IC module. When this function returns true, these functions are supported on the device:

- PLIB_IC_Buffer32BitGet
- PLIB_IC_Buffer16BitGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsBufferValue(IC_MODULE_ID index)

PLIB_IC_ExistsCaptureMode Function

Identifies whether the CaptureMode feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsCaptureMode(IC_MODULE_ID index);
```

Returns

- true - The CaptureMode feature is supported on the device
- false - The CaptureMode feature is not supported on the device

Description

This function identifies whether the CaptureMode feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_ModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsCaptureMode(IC_MODULE_ID index)

PLIB_IC_ExistsEdgeCapture Function

Identifies whether the EdgeCapture feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsEdgeCapture(IC_MODULE_ID index);
```

Returns

- true - The EdgeCapture feature is supported on the device
- false - The EdgeCapture feature is not supported on the device

Description

This function identifies whether the EdgeCapture feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_FirstCaptureEdgeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsEdgeCapture(IC_MODULE_ID index)

PLIB_IC_ExistsEnable Function

Identifies whether the EnableControl feature exists on the IC module

File

plib_ic.h

C

```
bool PLIB_IC_ExistsEnable(IC_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the IC module. When this function returns true, these functions are supported on the device:

- PLIB_IC_Enable
- PLIB_IC_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsEnable(IC_MODULE_ID index)

PLIB_IC_ExistsEventsPerInterruptSelect Function

Identifies whether the EventsPerInterruptSelect feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsEventsPerInterruptSelect(IC_MODULE_ID index);
```

Returns

- true - The EventsPerInterruptSelect feature is supported on the device
- false - The EventsPerInterruptSelect feature is not supported on the device

Description

This function identifies whether the EventsPerInterruptSelect feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_EventsPerInterruptSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsEventsPerInterruptSelect(IC_MODULE_ID index)

PLIB_IC_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsStopInIdle(IC_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the IC module. When this function returns true, these functions are supported on the device:

- PLIB_IC_StopInIdleEnable
- PLIB_IC_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsStopInIdle(IC_MODULE_ID index)

PLIB_IC_ExistsTimerSelect Function

Identifies whether the TimerSelect feature exists on the IC module.

File

plib_ic.h

C

```
bool PLIB_IC_ExistsTimerSelect(IC_MODULE_ID index);
```

Returns

- true - The TimerSelect feature is supported on the device
- false - The TimerSelect feature is not supported on the device

Description

This function identifies whether the TimerSelect feature is available on the IC module. When this function returns true, this function is supported on the device:

- PLIB_IC_TimerSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_IC_ExistsTimerSelect(IC_MODULE_ID index)

g) Data Types and Constants

IC_BUFFER_SIZE Enumeration

Selects the IC Buffer size.

File

plib_ic_help.h

C

```
typedef enum {
  IC_BUFFER_SIZE_32BIT,
  IC_BUFFER_SIZE_16BIT
} IC_BUFFER_SIZE;
```

Members

Members	Description
IC_BUFFER_SIZE_32BIT	32-bit timer source select
IC_BUFFER_SIZE_16BIT	16-bit timer source select

Description

IC Buffer Size

The IC buffer can be set to 16 bits or 32 bits. The buffer size is set by the PLIB_IC_BufferSizeSelect function.

IC_EDGE_TYPES Enumeration

Lists the options to select a rising or falling edge for input capture.

File

plib_ic_help.h

C

```
typedef enum {
  IC_EDGE_FALLING,
  IC_EDGE_RISING
} IC_EDGE_TYPES;
```

Members

Members	Description
IC_EDGE_FALLING	Select Falling Edge
IC_EDGE_RISING	Select Rising Edge

Description

IC Edge Types Select

This enumeration lists the options to select a rising edge input capture or a falling edge input capture. The selection is made by the PLIB_IC_FirstCaptureEdgeSelect function.

IC_EVENTS_PER_INTERRUPT Enumeration

Lists the number of input capture events for an interrupt to occur.

File

plib_ic_help.h

C

```
typedef enum {
  IC_INTERRUPT_ON_EVERY_4TH_CAPTURE_EVENT,
  IC_INTERRUPT_ON_EVERY_3RD_CAPTURE_EVENT,
  IC_INTERRUPT_ON_EVERY_2ND_CAPTURE_EVENT,
  IC_INTERRUPT_ON_EVERY_CAPTURE_EVENT
} IC_EVENTS_PER_INTERRUPT;
```

Members

Members	Description
IC_INTERRUPT_ON_EVERY_4TH_CAPTURE_EVENT	Interrupt on every 4th capture event
IC_INTERRUPT_ON_EVERY_3RD_CAPTURE_EVENT	Interrupt on every 3rd capture event
IC_INTERRUPT_ON_EVERY_2ND_CAPTURE_EVENT	Interrupt on every 2nd capture event
IC_INTERRUPT_ON_EVERY_CAPTURE_EVENT	Interrupt on every capture event

Description

IC Capture Events Per Interrupt Select

This enumeration lists the number of events that should occur before an interrupt is generated. An interrupt can occur on every capture event, and every second, third, or fourth capture event.

IC_INPUT_CAPTURE_MODES Enumeration

Lists all of the input capture modes for the IC module.

File

plib_ic_help.h

C

```
typedef enum {
  IC_INPUT_CAPTURE_INTERRUPT_MODE,
  IC_INPUT_CAPTURE_EVERY_EDGE_MODE,
  IC_INPUT_CAPTURE_EVERY_16TH_EDGE_MODE,
  IC_INPUT_CAPTURE_EVERY_4TH_EDGE_MODE,
  IC_INPUT_CAPTURE_RISING_EDGE_MODE,
  IC_INPUT_CAPTURE_FALLING_EDGE_MODE,
  IC_INPUT_CAPTURE_EDGE_DETECT_MODE,
  IC_INPUT_CAPTURE_DISABLE_MODE
} IC_INPUT_CAPTURE_MODES;
```

Members

Members	Description
IC_INPUT_CAPTURE_INTERRUPT_MODE	Interrupt only mode: Rising edge on input triggers an interrupt. This mode is used only when device is in Sleep/Idle mode
IC_INPUT_CAPTURE_EVERY_EDGE_MODE	Every Edge Capture mode: The first edge of the input signal is specified via PLIB_IC_RisingEdgeCaptureSet() or PLIB_IC_FallingEdgeCaptureSet() routines. Subsequently, timer count value is captured on every rising and falling of the input signal
IC_INPUT_CAPTURE_EVERY_16TH_EDGE_MODE	Prescaled Capture mode: Timer count value is captured every 16th rising edge of input signal
IC_INPUT_CAPTURE_EVERY_4TH_EDGE_MODE	Prescaled Capture mode: Timer count value is captured every 4th rising edge of input signal
IC_INPUT_CAPTURE_RISING_EDGE_MODE	Rising Edge mode: Timer count value is captured on every rising edge of input signal
IC_INPUT_CAPTURE_FALLING_EDGE_MODE	Falling Edge mode: Timer count value is captured on every falling edge of input signal
IC_INPUT_CAPTURE_EDGE_DETECT_MODE	Edge Detect mode: Timer count value is captured on every rising and falling of the input signal Interrupt control bits are ignored and an interrupt event is generated for every capture. Overflow status is not updated.
IC_INPUT_CAPTURE_DISABLE_MODE	Capture module is disabled.Input signal is ignored and no capture events or interrupts are generated

Description

IC Input Capture Mode Select

This enumeration lists all of the input capture modes for IC module. The capture mode is set via the PLIB_IC_ModeSelect function.

IC_MODULE_ID Enumeration

File

plib_ic_help.h

C

```
typedef enum {
  IC_ID_1,
  IC_ID_2,
  IC_ID_3,
  IC_ID_4,
  IC_ID_5,
  IC_ID_6,
  IC_ID_7,
  IC_ID_8,
  IC_ID_9,
  IC_NUMBER_OF_MODULES
} IC_MODULE_ID;
```

Members

Members	Description
IC_NUMBER_OF_MODULES	The total number of modules available.

Description

IC_MODULE_ID

This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on a Microchip microcontroller.

Refer to the specific device data sheet to determine the correct number of modules defined for the device.

IC_TIMERS Enumeration

Lists the different 16-bit timers for the IC module.

File

plib_ic_help.h

C

```
typedef enum {
  IC_TIMER_TMR3,
  IC_TIMER_TMR2
} IC_TIMERS;
```

Members

Members	Description
IC_TIMER_TMR3	Timer3 select
IC_TIMER_TMR2	Timer2 select

Description

IC 16-bit Timer Select

This enumeration lists the different 16-bit timers for the IC time base when the IC module is configured with a 16-bit timer resource.

IC_ALT_TIMERS Enumeration

Lists the different 16-bit alternate timers for the IC module.

File

plib_ic_help.h

C

```
typedef enum {
  IC_ALT_TIMER_TMR3,
  IC_ALT_TIMER_TMR2,
  IC_ALT_TIMER_TMR5,
  IC_ALT_TIMER_TMR4,
  IC_ALT_TIMER_TMR7,
  IC_ALT_TIMER_TMR6
} IC_ALT_TIMERS;
```

Members

Members	Description
IC_ALT_TIMER_TMR3	Select Timer3 as the alternate clock source for IC module
IC_ALT_TIMER_TMR2	Select Timer2 as the alternate clock source for IC module
IC_ALT_TIMER_TMR5	Select Timer5 as the alternate clock source for IC module
IC_ALT_TIMER_TMR4	Select Timer4 as the alternate clock source for IC module
IC_ALT_TIMER_TMR7	Select Timer7 as the alternate clock source for IC module
IC_ALT_TIMER_TMR6	Select Timer6 as the alternate clock source for IC module

Description

IC 16-bit alternate Timer Select

This enumeration lists the different 16-bit timers for the IC time base when the IC module is configured with a 16-bit alternate timer resource.

Files

Files

Name	Description
plib_ic.h	This file contains the interface definition for the Input Capture (IC) peripheral library.
plib_ic_help.h	This is file plib_ic_help.h.

Description

This section lists the source and header files used by the library.

plib_ic.h

This file contains the interface definition for the Input Capture (IC) peripheral library.

Functions

	Name	Description
	PLIB_IC_AlternateClockDisable	Selects Timer2 and Timer3, instead of the alternate clock source.
	PLIB_IC_AlternateClockEnable	Selects the alternate clock source.
	PLIB_IC_AlternateTimerSelect	Selects an alternate timer as a clock source for the IC module.
	PLIB_IC_Buffer16BitGet	Obtains the 16-bit input capture buffer value.
	PLIB_IC_Buffer32BitGet	Obtains the 32-bit input capture buffer value.
	PLIB_IC_BufferIsEmpty	Checks whether the input capture buffer is empty.
	PLIB_IC_BufferOverflowHasOccurred	Checks whether an input capture buffer overflow has occurred.
	PLIB_IC_BufferSizeSelect	Sets the input capture buffer size.
	PLIB_IC_Disable	Disables the IC module.
	PLIB_IC_Enable	Enables the IC module.
	PLIB_IC_EventsPerInterruptSelect	Selects the number of capture events before an interrupt is issued.
	PLIB_IC_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the IC module.
	PLIB_IC_ExistsAlternateTimerSelect	Identifies whether the AlternateTimerSelect feature exists on the IC module.
	PLIB_IC_ExistsBufferIsEmptyStatus	Identifies whether the BufferIsEmptyStatus feature exists on the IC module
	PLIB_IC_ExistsBufferOverflowStatus	Identifies whether the BufferOverflowStatus feature exists on the IC module.
	PLIB_IC_ExistsBufferSize	Identifies whether the BufferSize feature exists on the IC module.
	PLIB_IC_ExistsBufferValue	Identifies whether the BufferValue feature exists on the IC module.
	PLIB_IC_ExistsCaptureMode	Identifies whether the CaptureMode feature exists on the IC module.
	PLIB_IC_ExistsEdgeCapture	Identifies whether the EdgeCapture feature exists on the IC module.
	PLIB_IC_ExistsEnable	Identifies whether the EnableControl feature exists on the IC module
	PLIB_IC_ExistsEventsPerInterruptSelect	Identifies whether the EventsPerInterruptSelect feature exists on the IC module.
	PLIB_IC_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the IC module.
	PLIB_IC_ExistsTimerSelect	Identifies whether the TimerSelect feature exists on the IC module.
	PLIB_IC_FirstCaptureEdgeSelect	Captures the timer count value at the first selected edge of input signal.
	PLIB_IC_ModeSelect	Selects the input capture mode for IC module.
	PLIB_IC_StopInIdleDisable	Continues module operation when the device enters Idle mode
	PLIB_IC_StopInIdleEnable	Discontinues IC module operation when the device enters Idle mode.
	PLIB_IC_TimerSelect	Selects the clock source for the IC module.

Description

IC Module Peripheral Library Interface Header

This library provides a low-level abstraction of the Input Capture (IC) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

File Name

plib_ic.h

Company

Microchip Technology Inc.

plib_ic_help.h

Enumerations

	Name	Description
	IC_ALT_TIMERS	Lists the different 16-bit alternate timers for the IC module.
	IC_BUFFER_SIZE	Selects the IC Buffer size.
	IC_EDGE_TYPES	Lists the options to select a rising or falling edge for input capture.
	IC_EVENTS_PER_INTERRUPT	Lists the number of input capture events for an interrupt to occur.
	IC_INPUT_CAPTURE_MODES	Lists all of the input capture modes for the IC module.
	IC_MODULE_ID	IC_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on a Microchip microcontroller. Refer to the specific device data sheet to determine the correct number of modules defined for the device.
	IC_TIMERS	Lists the different 16-bit timers for the IC module.

Description

This is file plib_ic_help.h.

Interrupt Peripheral Library

This section describes the Interrupt Peripheral Library.

Introduction

This library provides a low-level abstraction of the Interrupt Controller module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The Interrupt Controller is a key component of a microcontroller. The Interrupt Controller reduces the numerous peripheral interrupt request signals to a single interrupt request signal to the microcontroller core.

The key features present on an Interrupt Controller are:

- **Interrupt source configuration**: Provides the ability to enable, disable, and prioritize a source that may interrupt the core
- **Interrupt status flags:** Identify the source of an interrupt request or exception, usually generated by a peripheral

The interrupt will be sent to the core, only if the source is enabled to generate an interrupt and interrupts have been enabled (if supported). Sources will be prioritized so that the highest priority interrupt will be sent to the core if multiple sources cause interrupts at the same time.

Many interrupt controllers also include, other optional features:

- **Global enable configuration**: Controls the generation of any interrupt to the core from the Interrupt Controller
- **Choice of edge transition**: Controls the edge transition (high-to-low or low-to-high) on which the interrupts are generated
- **Priority configuration**: Associates a programmable priority with the interrupt source

Interrupts can be handled inside the core in the following ways:

- **Vectored**: Each interrupt has an associated interrupt handler routine
- **Non-vectored**: All interrupts have a single (shared) interrupt handler routine. The software determines the interrupt source based on the interrupt status-flags register.
- **Combination of vectored and non-vectored**: Interrupts combined in groups. Each group has a single common interrupt vector or handler, but there are multiple groups (i.e., multiple vectors). Within a specific vector group, software determines which one in the group sources caused the interrupt.

A microcontroller can support one, or more than one of these modes.

Note: Please refer to the specific device data sheet to determine availability of these features for your device.

Using the Library

This topic describes the basic architecture of the Interrupt Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_int.h`

The interface to the Interrupt Peripheral Library is defined in the `plib_int.h` header file, which is included in the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Interrupt Peripheral Library must include `peripheral.h`.

Library File:

The Interrupt Peripheral Library (`.a`) file is installed with MPLAB Harmony.

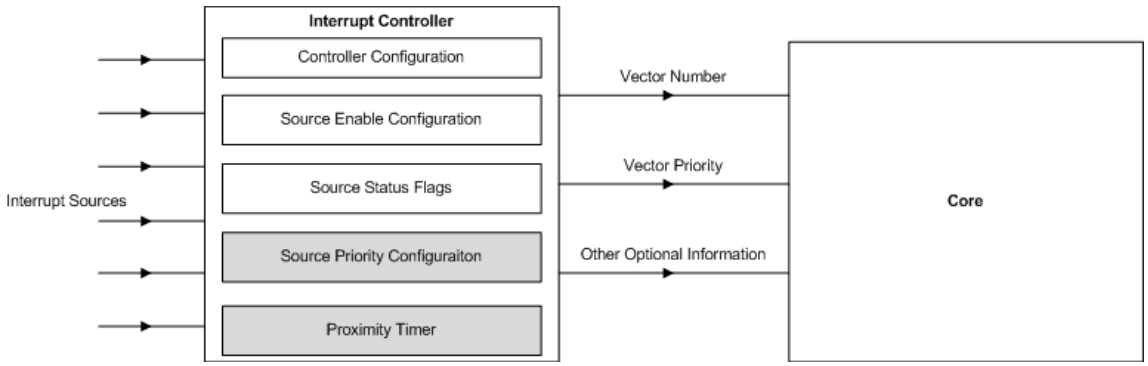
Please refer to the What is MPLAB Harmony? section for how the peripheral interacts with the framework.

Hardware Abstraction Model

This library provides a low-level abstraction of the Interrupt Controller module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model



The **Interrupt Controller** receives requests from multiple **Interrupt Sources**. The **Interrupt Controller Configuration** controls the overall operation of the interrupt controller. Not all microcontrollers support all the following features listed:

- Interrupt nesting configuration
- Edge selection of the external interrupts
- Vector mode selection
- Priority level configuration
- Current CPU interrupt and priority status information

Source Enable Configuration enables or disables a particular source. On some microcontrollers other CPU exceptions or traps can also be controlled. **Source Status Flags** give the status of the interrupt request from a specific interrupt source.

Other optional features include the **Vector Priority Configuration**, which is available on some micro controllers and the **Proximity Timer**. Priority configuration allows a priority level to be associated with each interrupt vector. The proximity timer creates a temporal window in which a group of interrupts of the same, or lower, priority will be held off. This provides an opportunity to queue these interrupt requests and process them using tail-chaining of multiple IRQs and allows higher priority interrupts to pass through during the hold-off window.

Traps and Exceptions

Some of the things that can interrupt the normal flow of instruction execution of the CPU core are not actual interrupt sources. These are things such as a division-by-zero error or attempting to address unimplemented memory. These errors are either internal to the CPU core itself or are so catastrophic that they are non-maskable and are trapped separately. This library refers to them in general as "traps" and there are interface elements provided to allow software to handle them.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Interrupt module.

Library Interface Section	Description
General Configuration Functions	Provides setup, configuration, and status interface routines for the overall operation of the Interrupt Controller module.

Interrupt Source Control Functions	Provides setup and status routines for: • Setting up the source enable configuration • Setting up the vector priority configuration • Querying a source's interrupt status flag
Other Status and Control Functions	Provides setup and status routines for: • Global interrupt enable status • Vector number and priority of the current ISR • Proximity timer • Trap handling
Feature Existence Functions	Determine whether certain features are available.

How the Library Works

This section provides information on initialization and interrupt source setup and handling.

Description

Initialization

The steps that are required to initialize the Interrupt Controller module vary for different microcontrollers.

Note: Refer to the specific device data sheet for the correct initialization sequence for your device.

Interrupt Source Setup

An interrupt Source needs to be enabled in order for it to be able to generate a interrupt to the core. Use the function PLIB_INT_SourceEnable to enable the interrupt source. INT_SOURCE is a list of the possible interrupt sources

On some microcontrollers, each interrupt vector can be associated with a priority. Use the function PLIB_INT_VectorPrioritySet to set the interrupt vector priority. INT_VECTOR is a list of the possible interrupt vectors and INT_PRIORITY_LEVEL is the list of possible priority levels.

On some microcontrollers, each interrupt vector can be associated with a sub-priority. Use the function PLIB_INT_VectorSubPrioritySet to set the interrupt source sub-priority. INT_SUBPRIORITY_LEVEL is a list of possible sub-priority levels.

On most microcontrollers, it will be necessary to enable interrupt generation to the core by calling the PLIB_INT_Enable function.

Interrupt Handling

The status of the interrupt indicates whether or not the interrupt request was generated by the source. The status of the trap source can be obtained using the function PLIB_INT_TrapSourceFlagGet and the status of the interrupt source can be obtained using the function PLIB_INT_SourceFlagGet. The interrupt status flags can be used to indicate the interrupt status even if the interrupt is not enabled for the source. When the interrupt is not enabled for the source, no interrupt is generated to the core even though the interrupt controller still recognizes that the source requested an interrupt. INT_TRAP_SOURCE is a list of possible trap events, and INT_SOURCE is a list of possible interrupt sources.

Before the Interrupt Service Routine (ISR) ends, the status of the interrupt needs to be cleared to avoid entering the ISR repeatedly for the same interrupt event. A trap source status can be cleared using the function PLIB_INT_TrapSourceFlagClear, and an interrupt source status can be cleared using the function PLIB_INT_SourceFlagClear. INT_TRAP_SOURCE is a list of possible trap events, and INT_SOURCE is a list of possible interrupt sources.

Configuring the Library

The library is configured for the supported Interrupt module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_INT_Disable	Disables the generation of interrupts to the CPU.
	PLIB_INT_Enable	Enables the generation of interrupts to the CPU.
	PLIB_INT_ExternalFallingEdgeSelect	Selects the falling edge as the edge polarity of the external interrupt.
	PLIB_INT_ExternalRisingEdgeSelect	Selects the rising edge as the edge polarity of the external interrupt.
	PLIB_INT_IsEnabled	Identifies if interrupts are currently enabled or disabled at the top level.
	PLIB_INT_MultiVectorSelect	Configures the Interrupt Controller for Multiple Vector mode.

	PLIB_INT_PriorityGet	Returns the priority level of the latest interrupt presented to the CPU.
	PLIB_INT_ProximityTimerDisable	Disables the interrupt temporal-proximity timer.
	PLIB_INT_ProximityTimerEnable	Enables the interrupt temporal-proximity timer and selects the priority levels that start the timer.
	PLIB_INT_SingleVectorSelect	Configures the Interrupt Controller for Single Vector mode.
	PLIB_INT_SingleVectorShadowSetDisable	Disables the Shadow Register Set in Single Vector mode.
	PLIB_INT_SingleVectorShadowSetEnable	Enables the Shadow Register Set in Single Vector mode.
	PLIB_INT_SetState	Restores the status of CPU interrupts based on the value passed into the function.
	PLIB_INT_ShadowRegisterAssign	Assigns a shadow register set for an interrupt priority level.
	PLIB_INT_VariableVectorOffsetSet	Sets the offset specific to an interrupt source number.
	PLIB_INT_SoftwareNMITrigger	Triggers software NMI.

b) Interrupt Source Control Functions

	Name	Description
	PLIB_INT_SourceDisable	Disables the interrupt source
	PLIB_INT_SourceEnable	Enables the interrupt source.
	PLIB_INT_SourceFlagClear	Clears the status of the interrupt flag for the selected source.
	PLIB_INT_SourceFlagGet	Returns the status of the interrupt flag for the selected source.
	PLIB_INT_SourceFlagSet	Sets the status of the interrupt flag for the selected source.
	PLIB_INT_SourceIsEnabled	Gets the enable state of the interrupt source.
	PLIB_INT_VectorPriorityGet	Gets the priority of the interrupt vector.
	PLIB_INT_VectorPrioritySet	Sets the priority of the interrupt vector.
	PLIB_INT_VectorSubPriorityGet	Gets the sub-priority of the interrupt vector.
	PLIB_INT_VectorSubPrioritySet	Sets the sub-priority of the interrupt vector.

c) Other Status and Control Functions

	Name	Description
	PLIB_INT_CPUCurrentPriorityLevelGet	Gets the interrupt priority level at which the CPU is currently operating.
	PLIB_INT_ProximityTimerGet	Returns the value used by the temporal proximity timer as a reload value when the temporal proximity timer is triggered by an interrupt event.
	PLIB_INT_ProximityTimerSet	Sets the temporal proximity timer reload value. This value is used to reload the proximity timer after it has been triggered by an interrupt event.
	PLIB_INT_VectorGet	Returns the interrupt vector that is presented to the CPU.
	PLIB_INT_GetStateAndDisable	Disables the generation of interrupts to the CPU.
	PLIB_INT_ShadowRegisterGet	Gets the shadow register set assigned for an interrupt priority level.
	PLIB_INT_VariableVectorOffsetGet	Gets the offset specific to an interrupt source number.

d) Feature Existence Functions

	Name	Description
	PLIB_INT_ExistsCPUCurrentPriorityLevel	Identifies whether the CPUCurrentPriorityLevel feature exists on the Interrupts module.
	PLIB_INT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Interrupt module.
	PLIB_INT_ExistsExternalINTEdgeSelect	Identifies whether the ExternalINTEdgeSelect feature exists on the Interrupts module.
	PLIB_INT_ExistsINTCPUPriority	Identifies whether the INTCPUPriority feature exists on the Interrupt module.
	PLIB_INT_ExistsINTCPUVector	Identifies whether the INTCPUVector feature exists on the Interrupt module.
	PLIB_INT_ExistsProximityTimerControl	Identifies whether the ProximityTimerControl feature exists on the Interrupts module.
	PLIB_INT_ExistsProximityTimerEnable	Identifies whether the ProximityTimerEnable feature exists on the Interrupts module.
	PLIB_INT_ExistsSingleVectorShadowSet	Identifies whether the SingleVectorShadowSet feature exists on the Interrupt module.
	PLIB_INT_ExistsSourceControl	Identifies whether the SourceControl feature exists on the Interrupt module.
	PLIB_INT_ExistsSourceFlag	Identifies whether the SourceFlag feature exists on the Interrupt module.
	PLIB_INT_ExistsVectorPriority	Identifies whether the VectorPriority feature exists on the Interrupt module.
	PLIB_INT_ExistsVectorSelect	Identifies whether the VectorSelect feature exists on the Interrupt module.
	PLIB_INT_ExistsShadowRegisterAssign	Identifies whether the ShadowRegisterAssign feature exists on the Interrupts module.
	PLIB_INT_ExistsVariableOffset	Identifies whether the VariableOffset feature exists on the Interrupt module.
	PLIB_INT_ExistsSoftwareNMI	Identifies whether the SoftwareNMI feature exists on the Interrupt module.

e) Data Types and Constants

	Name	Description
	INT_EXTERNAL_SOURCES	Lists the possible external sources that can cause an interrupt to occur.
	INT_PRIORITY_LEVEL	Lists the possible interrupt priority levels.
	INT_SOURCE	Lists the possible sources that can cause an interrupt to occur.
	INT_SUBPRIORITY_LEVEL	Lists the possible interrupt sub priority levels.
	INT_VECTOR	Lists the possible interrupt vectors.
	INT_STATE_GLOBAL	Data type defining the global interrupt state.
	INT_SHADOW_REGISTER	Lists the possible shadow register sets.
	INT_VECTOR_SPACING	Lists the possible interrupt vector spacing.

Description

This section describes the Application Programming Interface (API) functions of the Interrupt Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_INT_Disable Function

Disables the generation of interrupts to the CPU.

File

plib_int.h

C

```
void PLIB_INT_Disable(INT_MODULE_ID index);
```

Returns

None.

Description

This function disables the generation of interrupts to the CPU.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsEnableControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_Disable( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_Disable (INT_MODULE_ID index)

PLIB_INT_Enable Function

Enables the generation of interrupts to the CPU.

File

plib_int.h

C

```
void PLIB_INT_Enable(INT_MODULE_ID index);
```

Returns

None.

Description

This function enables the generation of interrupts to the CPU.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsEnableControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_Enable( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_Enable (INT_MODULE_ID index)

PLIB_INT_ExternalFallingEdgeSelect Function

Selects the falling edge as the edge polarity of the external interrupt.

File

plib_int.h

C

```
void PLIB_INT_ExternalFallingEdgeSelect(INT_MODULE_ID index, INT_EXTERNAL_SOURCES source);
```

Returns

None.

Description

This function selects the falling edge as the edge polarity of the external interrupt.

Remarks

This function implements an operation of the ExternalINTEdgeSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsExternalINTEdgeSelect function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_ExternalFallingEdgeSelect( INT_ID_0,
                                    (INT_EXTERNAL_INT_SOURCE0 |
                                    INT_EXTERNAL_INT_SOURCE1) );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

source	One or more of the values from INT_EXTERNAL_SOURCES. Values can be combined using a bitwise "OR" operation.

Function

void PLIB_INT_ExternalFallingEdgeSelect (INT_MODULE_ID index, INT_EXTERNAL_SOURCES source)

PLIB_INT_ExternalRisingEdgeSelect Function

Selects the rising edge as the edge polarity of the external interrupt.

File

plib_int.h

C

```
void PLIB_INT_ExternalRisingEdgeSelect(INT_MODULE_ID index, INT_EXTERNAL_SOURCES source);
```

Returns

None.

Description

This function selects the rising edge as the edge polarity of the external interrupt.

Remarks

This function implements an operation of the ExternalINTEdgeSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsExternalINTEdgeSelect function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_ExternalRisingEdgeSelect( INT_ID_0,
                                   ( INT_EXTERNAL_INT_SOURCE0 |
                                       INT_EXTERNAL_INT_SOURCE1 ) );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One or more of the values from INT_EXTERNAL_SOURCES. Values can be combined using a bitwise "OR" operation.

Function

void PLIB_INT_ExternalRisingEdgeSelect (INT_MODULE_ID index,
INT_EXTERNAL_SOURCES source)

PLIB_INT_IsEnabled Function

Identifies if interrupts are currently enabled or disabled at the top level.

File

plib_int.h

C

```
bool PLIB_INT_IsEnabled(INT_MODULE_ID index);
```

Returns

- true - If the interrupts are currently enabled
- false - If the interrupts are currently disabled

Description

This function identifies if interrupts are enabled or disabled at the top level.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsEnableControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
status = PLIB_INT_IsEnabled( INT_ID_0 );
```

Function

bool PLIB_INT_IsEnabled (INT_MODULE_ID index)

PLIB_INT_MultiVectorSelect Function

Configures the Interrupt Controller for Multiple Vector mode.

File

plib_int.h

C

```
void PLIB_INT_MultiVectorSelect(INT_MODULE_ID index);
```

Returns

None.

Description

This function configures the Interrupt Controller for Multiple Vector mode. Interrupt requests will serviced at the calculated vector addresses. The CPU vectors to the unique address for each vector number.

Remarks

While the user can, during run-time, reconfigure the interrupt controller from Single Vector mode to Multiple Vector mode, such action is strongly discouraged, as it may result in undefined behavior.

This function implements an operation of the VectorSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVectorSelect function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_MultiVectorSelect( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_MultiVectorSelect(INT_MODULE_ID index)

PLIB_INT_PriorityGet Function

Returns the priority level of the latest interrupt presented to the CPU.

File

plib_int.h

C

```
INT_PRIORITY_LEVEL PLIB_INT_PriorityGet(INT_MODULE_ID index);
```

Returns

One of the possible values from INT_PRIORITY_LEVEL.

Description

This function returns the priority level of the latest interrupt presented to the CPU.

Remarks

This value should only be used when the interrupt controller is configured for single vector mode using the function PLIB_INT_SingleVectorSelect.

This function implements an operation of the INTCPUPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsINTCPUPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_PRIORITY_LEVEL level = PLIB_INT_RequestedPriorityLevelGet( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

INT_PRIORITY_LEVEL PLIB_INT_PriorityGet (INT_MODULE_ID index)

PLIB_INT_ProximityTimerDisable Function

Disables the interrupt temporal-proximity timer.

File

plib_int.h

C

```
void PLIB_INT_ProximityTimerDisable(INT_MODULE_ID index);
```

Returns

None.

Description

This function disables the interrupt temporal-proximity timer.

Remarks

This function implements an operation of the ProximityTimerEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsProximityTimerEnable function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_ProximityTimerDisable(INT_PRIORITY_LEVEL3);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_ProximityTimerDisable(INT_MODULE_ID index)

PLIB_INT_ProximityTimerEnable Function

Enables the interrupt temporal-proximity timer and selects the priority levels that start the timer.

File

plib_int.h

C

```
void PLIB_INT_ProximityTimerEnable(INT_MODULE_ID index, INT_PRIORITY_LEVEL priority);
```

Returns

None.

Description

This function enables the interrupt temporal-proximity timer and selects the priority levels that start the timer. One of the possible values from PLIB_INT_PRIORITY_LEVEL can be selected. Interrupts of that priority and lower start the temporal proximity timer.

Remarks

This function implements an operation of the ProximityTimerEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsProximityTimerEnable function in your application to determine whether this feature is available.

Choosing INT_PRIORITY_0 disables the proximity timer (exactly the same as if PLIB_INT_ProximityTimerDisable had been called.

Preconditions

None.

Example

```
//Interrupts of group priority 3 or lower start the temporal proximity timer
PLIB_INT_ProximityTimerEnable( INT_ID_0, INT_PRIORITY_LEVEL3 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
priority	One of possible values from INT_PRIORITY_LEVEL.

Function

void PLIB_INT_ProximityTimerEnable (INT_MODULE_ID index,
INT_PRIORITY_LEVEL priority)

PLIB_INT_SingleVectorSelect Function

Configures the Interrupt Controller for Single Vector mode.

File

plib_int.h

C

```
void PLIB_INT_SingleVectorSelect(INT_MODULE_ID index);
```

Returns

None.

Description

The function configures the Interrupt Controller for Single Vector mode. All interrupt requests will serviced at one vector address. The CPU vectors to the same address for all interrupt sources.

Remarks

While the user can, during run-time, reconfigure the Interrupt Controller from Single Vector mode to Multiple Vector mode, such action is strongly discouraged, as it may result in undefined behavior.

This function implements an operation of the VectorSelect feature. This feature may not be available on all devices. Please refer to the specific

device data sheet to determine availability or use the PLIB_INT_ExistsVectorSelect function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SingleVectorSelect( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_SingleVectorSelect (INT_MODULE_ID index)

PLIB_INT_SingleVectorShadowSetDisable Function

Disables the Shadow Register Set in Single Vector mode.

File

plib_int.h

C

```
void PLIB_INT_SingleVectorShadowSetDisable(INT_MODULE_ID index);
```

Returns

None.

Description

This function disables usage of the "shadow" set of processor registers when operating in Single Vector mode.

Remarks

This function implements an operation of the SingleVectorShadowSet feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSingleVectorShadowSet function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SingleVectorShadowSetEnable( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_SingleVectorShadowSetDisable (INT_MODULE_ID index)

PLIB_INT_SingleVectorShadowSetEnable Function

Enables the Shadow Register Set in Single Vector mode.

File

plib_int.h

C

```
void PLIB_INT_SingleVectorShadowSetEnable(INT_MODULE_ID index);
```

Returns

None.

Description

This function enables usage of the "shadow" set of processor registers when operating in Single Vector mode.

Remarks

This function implements an operation of the SingleVectorShadowSet feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSingleVectorShadowSet function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SingleVectorShadowSetEnable( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_SingleVectorShadowSetEnable (INT_MODULE_ID index)

PLIB_INT_SetState Function

Restores the status of CPU interrupts based on the value passed into the function.

File

plib_int.h

C

```
void PLIB_INT_SetState(INT_MODULE_ID index, INT_STATE_GLOBAL interrupt_state);
```

Returns

None.

Description

This function will use the value passed in, to set the state of global interrupts.

Remarks

This function should be paired with the use of PLIB_INT_GetStateAndDisable. The value returned from PLIB_INT_GetStateAndDisable should be passed into this function.

Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsEnableControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_STATE_GLOBAL interrupt_value;

interrupt_value = PLIB_INT_GetStateAndDisable( INT_ID_0 );

PLIB_INT_SetState(INT_ID_0, interrupt_value);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

interrupt_state	value returned from previous call to PLIB_INT_GetStateAndDisable.

Function

void PLIB_INT_SetState (INT_MODULE_ID index, INT_STATE_GLOBAL interrupt_state)

PLIB_INT_ShadowRegisterAssign Function

Assigns a shadow register set for an interrupt priority level.

File

plib_int.h

C

```
void PLIB_INT_ShadowRegisterAssign(INT_MODULE_ID index, INT_PRIORITY_LEVEL priority, INT_SHADOW_REGISTER shadowRegister);
```

Returns

None.

Description

The function assigns a shadow register set for an interrupt priority level.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsShadowRegisterAssign function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_ShadowRegisterAssign( INT_ID_0, INT_PRIORITY_LEVEL5, INT_SHADOW_REGISTER_5 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all devices that have only one Interrupt module).
priority	Interrupt priority level for the shadow register set that is to be assigned.
shadowRegister	Shadow register set number.

Function

void PLIB_INT_ShadowRegisterAssign (INT_MODULE_ID index,
INT_PRIORITY_LEVEL priority,
INT_SHADOW_REGISTER shadowRegister)

PLIB_INT_VariableVectorOffsetSet Function

Sets the offset specific to an interrupt source number.

File

plib_int.h

C

```
void PLIB_INT_VariableVectorOffsetSet(INT_MODULE_ID index, INT_VECTOR vector, uint32_t offset);
```

Returns

None.

Description

The function sets the offset specific to an interrupt source number.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVariableOffset function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Set 200 bytes offset
PLIB_INT_VariableVectorOffsetSet ( INT_ID_0, INT_VECTOR_USB1, 200 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
vector	Interrupt source, one of the possible value from INT_VECTOR enum.
offset	Offset in number of bytes.

Function

void PLIB_INT_VariableVectorOffsetSet (INT_MODULE_ID index, INT_VECTOR source, uint32_t offset)

PLIB_INT_SoftwareNMITrigger Function

Triggers software NMI.

File

plib_int.h

C

```
void PLIB_INT_SoftwareNMITrigger(INT_MODULE_ID index);
```

Returns

None.

Description

This function triggers the software NMI.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_INT_ExistsSoftwareNMI in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SoftwareNMITrigger( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

void PLIB_INT_SoftwareNMITrigger(INT_MODULE_ID index)

b) Interrupt Source Control Functions

PLIB_INT_SourceDisable Function

Disables the interrupt source

File

plib_int.h

C

```
void PLIB_INT_SourceDisable(INT_MODULE_ID index, INT_SOURCE source);
```

Returns

None.

Description

This function disables the given interrupt source.

Remarks

This function implements an operation of the SourceControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSourceControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SourceDisable(INT_ID_0,INT_SOURCE_TIMER_1);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One of the possible values from INT_SOURCE.

Function

void PLIB_INT_SourceDisable (INT_MODULE_ID index, INT_SOURCE source)

PLIB_INT_SourceEnable Function

Enables the interrupt source.

File

plib_int.h

C

```
void PLIB_INT_SourceEnable(INT_MODULE_ID index, INT_SOURCE source);
```

Returns

None.

Description

This function enables the interrupt source. The interrupt flag is set when the interrupt request is sampled. The pending interrupt request will not cause further processing if the interrupt is not enabled using this function or if interrupts are not enabled (on processors that support the PLIB_INT_Enable feature).

Remarks

This function implements an operation of the SourceControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSourceControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SourceEnable(INT_ID_0,INT_SOURCE_TIMER_1);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One of the possible values from INT_SOURCE.

Function

void PLIB_INT_SourceEnable (INT_MODULE_ID index, INT_SOURCE source)

PLIB_INT_SourceFlagClear Function

Clears the status of the interrupt flag for the selected source.

File

plib_int.h

C

```
void PLIB_INT_SourceFlagClear(INT_MODULE_ID index, INT_SOURCE source);
```

Returns

None.

Description

This function clears the status of the interrupt flag for the selected source. The flag is set when the interrupt request is identified. The pending interrupt request will not cause further processing if the interrupt is not enabled using the function PLIB_INT_InterruptSourceEnable or if interrupts are not enabled (on processors that support the PLIB_INT_Enable feature).

Remarks

This function implements an operation of the SourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSourceFlag function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_INT_SourceFlagGet(INT_ID_0,INT_SOURCE_TIMER_1))
{
    PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_1);
}
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One of the possible values from INT_SOURCE.

Function

void PLIB_INT_SourceFlagClear (INT_MODULE_ID index, INT_SOURCE source)

PLIB_INT_SourceFlagGet Function

Returns the status of the interrupt flag for the selected source.

File

plib_int.h

C

```
bool PLIB_INT_SourceFlagGet(INT_MODULE_ID index, INT_SOURCE source);
```

Returns

- true - If the interrupt request is recognized for the source
- false - If the interrupt request is not recognized for the source

Description

This function returns the status of the interrupt flag for the selected source. The flag is set when the interrupt request is recognized. The pending interrupt request will not cause further processing if the interrupt is not enabled using the function PLIB_INT_InterruptSourceEnable or if interrupts are not enabled (on processors that support the PLIB_INT_Enable feature).

Remarks

This function implements an operation of the SourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSourceFlag function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_TIMER_1))
{
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
}
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One of the possible values from INT_SOURCE.

Function

bool PLIB_INT_SourceFlagGet (INT_MODULE_ID index, INT_SOURCE source)

PLIB_INT_SourceFlagSet Function

Sets the status of the interrupt flag for the selected source.

File

plib_int.h

C

```
void PLIB_INT_SourceFlagSet(INT_MODULE_ID index, INT_SOURCE source);
```

Returns

None.

Description

This function sets the status of the interrupt flag for the selected source. This function will not be used during normal operation of the system. It is used to generate test interrupts for debug and testing purposes.

Remarks

This function implements an operation of the SourceFlag feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSourceFlag function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_SourceFlagSet(INT_ID_0,INT_SOURCE_TIMER_1);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One of the possible values from INT_SOURCE.

Function

void PLIB_INT_SourceFlagSet (INT_MODULE_ID index, INT_SOURCE source)

PLIB_INT_SourceIsEnabled Function

Gets the enable state of the interrupt source.

File

plib_int.h

C

```
bool PLIB_INT_SourceIsEnabled(INT_MODULE_ID index, INT_SOURCE source);
```

Returns

- true - If the interrupt source is enabled
- false - If the interrupt source is disabled

Description

This function gets the enable state of the interrupt source.

Remarks

This function implements an operation of the SourceControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsSourceControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_INT_SourceIsEnabled(INT_ID_0,INT_SOURCE_TIMER_1) != true)
{
    PLIB_INT_SourceEnable(INT_ID_0,INT_SOURCE_TIMER_1);
}
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
source	One of the possible values from INT_SOURCE.

Function

bool PLIB_INT_SourceIsEnabled (INT_MODULE_ID index, INT_SOURCE source)

PLIB_INT_VectorPriorityGet Function

Gets the priority of the interrupt vector.

File

plib_int.h

C

```
INT_PRIORITY_LEVEL PLIB_INT_VectorPriorityGet(INT_MODULE_ID index, INT_VECTOR vector);
```

Returns

- priority - One of the possible values from INT_PRIORITY_LEVEL

Description

Gets the priority of the interrupt vector. The priority is one of the possible values from INT_PRIORITY_LEVEL

Remarks

This function implements an operation of the VectorPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVectorPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_PRIORITY_LEVEL level;
level = PLIB_INT_VectorPriorityGet(INT_ID_0, INT_VECTOR_T1);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
vector	One of the possible values from INT_VECTOR

Function

PLIB_INT_PRIORITY_LEVEL INT_VectorPriorityGet (INT_MODULE_ID index, INT_VECTOR vector)

PLIB_INT_VectorPrioritySet Function

Sets the priority of the interrupt vector.

File

plib_int.h

C

```
void PLIB_INT_VectorPrioritySet(INT_MODULE_ID index, INT_VECTOR vector, INT_PRIORITY_LEVEL priority);
```

Returns

None.

Description

Sets the priority of the interrupt vector. The priority is one of the possible values from INT_PRIORITY_LEVEL.

Remarks

This function implements an operation of the VectorPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVectorPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T1, INT_PRIORITY_LEVEL4);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
vector	One of the possible values from INT_VECTOR
priority	One of the possible values from INT_PRIORITY_LEVEL

Function

void PLIB_INT_VectorPrioritySet(INT_MODULE_ID index,
INT_VECTOR vector,

INT_PRIORITY_LEVEL priority)

PLIB_INT_VectorSubPriorityGet Function

Gets the sub-priority of the interrupt vector.

File

plib_int.h

C

```
INT_SUBPRIORITY_LEVEL PLIB_INT_VectorSubPriorityGet(INT_MODULE_ID index, INT_VECTOR vector);
```

Returns

- priority - One of the possible values from INT_SUBPRIORITY_LEVEL

Description

This function gets the sub-priority of the interrupt vector. The priority is one of the possible values from INT_SUBPRIORITY_LEVEL.

Remarks

This function implements an operation of the VectorPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVectorPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_SUBPRIORITY_LEVEL level;
level = PLIB_INT_VectorSubPriorityGet(INT_ID_0, INT_VECTOR_T1);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
vector	One of the possible values from INT_VECTOR

Function

INT_SUBPRIORITY_LEVEL PLIB_INT_VectorSubPriorityGet (INT_MODULE_ID index,
INT_VECTOR vector)

PLIB_INT_VectorSubPrioritySet Function

Sets the sub-priority of the interrupt vector.

File

plib_int.h

C

```
void PLIB_INT_VectorSubPrioritySet(INT_MODULE_ID index, INT_VECTOR vector, INT_SUBPRIORITY_LEVEL
subPriority);
```

Returns

None.

Description

This function sets the sub priority of the interrupt vector. The sub-priority is one of the possible values from INT_SUBPRIORITY_LEVEL.

Remarks

This function implements an operation of the VectorPriority feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVectorPriority function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T1, INT_SUBPRIORITY_LEVEL1);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
vector	One of the possible values from INT_VECTOR
subPriority	One of the possible values from INT_SUBPRIORITY_LEVEL

Function

void PLIB_INT_VectorSubPrioritySet (INT_MODULE_ID index,
INT_VECTOR vector,
INT_SUBPRIORITY_LEVEL subPriority)

c) Other Status and Control Functions

PLIB_INT_CPUCurrentPriorityLevelGet Function

Gets the interrupt priority level at which the CPU is currently operating.

File

plib_int.h

C

```
INT_PRIORITY_LEVEL PLIB_INT_CPUCurrentPriorityLevelGet(INT_MODULE_ID index);
```

Returns

- priority - The current interrupt priority of the CPU. Range (0 - 15)

Description

This function gets the interrupt priority level at which the CPU is currently operating.

Remarks

User interrupts are disabled when the CPU priority is more than 7.

This function implements an operation of the CPUCurrentPriorityLevel feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsCPUCurrentPriorityLevel function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_PRIORITY_LEVEL currentCPUPriority;
currentCPUPriority = PLIB_INT_CPUCurrentPriorityLevelGet( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

INT_PRIORITY_LEVEL PLIB_INT_CPUCurrentPriorityLevelGet (INT_MODULE_ID index)

PLIB_INT_ProximityTimerGet Function

Returns the value used by the temporal proximity timer as a reload value when the temporal proximity timer is triggered by an interrupt event.

File

plib_int.h

C

```
uint32_t PLIB_INT_ProximityTimerGet(INT_MODULE_ID index);
```

Returns

Timer reload value.

Description

This function returns the value used by the temporal proximity timer as a reload value when the temporal proximity timer is triggered by an interrupt event.

Remarks

This function implements an operation of the ProximityTimerControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsProximityTimerControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t timer;

timer = PLIB_INT_ProximityTimerGet( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

uint32_t PLIB_INT_ProximityTimerGet (INT_MODULE_ID index)

PLIB_INT_ProximityTimerSet Function

Sets the temporal proximity timer reload value. This value is used to reload the proximity timer after it has been triggered by an interrupt event.

File

plib_int.h

C

```
void PLIB_INT_ProximityTimerSet(INT_MODULE_ID index, uint32_t timerreloadvalue);
```

Returns

None.

Description

This function sets the temporal proximity timer reload value. This value is used to reload the proximity timer after it has been triggered by an interrupt event.

Remarks

This function implements an operation of the ProximityTimerControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsProximityTimerControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_INT_ProximityTimerSet(INT_ID_0, 0x12345678);
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
timerReloadValue	Timer reload value.

Function

void PLIB_INT_ProximityTimerSet (INT_MODULE_ID index, uint32_t timerReloadValue)

PLIB_INT_VectorGet Function

Returns the interrupt vector that is presented to the CPU.

File

plib_int.h

C

```
INT_VECTOR PLIB_INT_VectorGet(INT_MODULE_ID index);
```

Returns

One of the possible values from INT_VECTOR.

Description

This function returns the interrupt vector that is presented to the CPU.

Remarks

This value should only be used when the interrupt controller is configured for single vector mode using the function PLIB_INT_MultiVectorDisable.

This function implements an operation of the INTCPUVector feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsINTCPUVector function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_VECTOR level = PLIB_INT_VectorGet( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

INT_VECTOR PLIB_INT_VectorGet (INT_MODULE_ID index)

PLIB_INT_GetStateAndDisable Function

Disables the generation of interrupts to the CPU.

File

plib_int.h

C

```
INT_STATE_GLOBAL PLIB_INT_GetStateAndDisable(INT_MODULE_ID index);
```

Returns

Value of CP0 Status register before interrupts were disabled globally.

Description

This function disables the generation of interrupts to the CPU.

Remarks

This function should be paired with the use of PLIB_INT_SetState. The value returned from this function should be passed into PLIB_INT_SetState.

Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsEnableControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_STATE_GLOBAL interrupt_value;

interrupt_value = PLIB_INT_GetStateAndDisable( INT_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).

Function

INT_STATE_GLOBAL PLIB_INT_GetStateAndDisable (INT_MODULE_ID index)

PLIB_INT_ShadowRegisterGet Function

Gets the shadow register set assigned for an interrupt priority level.

File

plib_int.h

C

```
INT_SHADOW_REGISTER PLIB_INT_ShadowRegisterGet(INT_MODULE_ID index, INT_PRIORITY_LEVEL priority);
```

Returns

None.

Description

The function gets the shadow register set assigned for an interrupt priority level.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsShadowRegisterAssign function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
INT_SHADOW_REGISTER shadowReg;

shadowReg = PLIB_INT_ShadowRegisterGet( INT_ID_0, INT_PRIORITY_LEVEL5 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
priority	Interrupt priority level for the shadow register set that is to be assigned.

Function

INT_SHADOW_REGISTER PLIB_INT_ShadowRegisterGet (INT_MODULE_ID index,

INT_PRIORITY_LEVEL priority)

PLIB_INT_VariableVectorOffsetGet Function

Gets the offset specific to an interrupt source number.

File

plib_int.h

C

```
uint32_t PLIB_INT_VariableVectorOffsetGet(INT_MODULE_ID index, INT_VECTOR vector);
```

Returns

Offset value.

Description

The function gets the offset specific to an interrupt source number.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use the PLIB_INT_ExistsVariableOffset function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t offset;

offset = PLIB_INT_VariableVectorOffsetGet ( INT_ID_0, INT_VECTOR_USB1 );
```

Parameters

Parameters	Description
index	Identifier for the module instance to be configured (it should be INT_ID_0 for all of the devices that have only one Interrupt module).
vector	Interrupt source, one of the possible value from INT_VECTOR enum.

Function

uint32_t PLIB_INT_VariableVectorOffsetGet (INT_MODULE_ID index, INT_VECTOR vector)

d) Feature Existence Functions

PLIB_INT_ExistsCPUCurrentPriorityLevel Function

Identifies whether the CPUCurrentPriorityLevel feature exists on the Interrupts module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsCPUCurrentPriorityLevel(INT_MODULE_ID index);
```

Returns

- true - The CPUCurrentPriorityLevel feature is supported on the device
- false - The CPUCurrentPriorityLevel feature is not supported on the device

Description

This function identifies whether the CPUCurrentPriorityLevel feature is available on the Interrupt module. When this function returns true, this function is supported on the device:

- PLIB_INT_CPUCurrentPriorityLevelGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsCPUCurrentPriorityLevel(INT_MODULE_ID index)

PLIB_INT_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsEnableControl(INT_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_Enable
- PLIB_INT_Disable
- PLIB_INT_IsEnabled
- PLIB_INT_SetState
- PLIB_INT_GetStateAndDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsEnableControl(INT_MODULE_ID index)

PLIB_INT_ExistsExternalINTEdgeSelect Function

Identifies whether the ExternalINTEdgeSelect feature exists on the Interrupts module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsExternalINTEdgeSelect(INT_MODULE_ID index);
```

Returns

- true - The ExternalINTEdgeSelect feature is supported on the device
- false - The ExternalINTEdgeSelect feature is not supported on the device

Description

This function identifies whether the ExternalINTEdgeSelect feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_ExternalRisingEdgeSelect
- PLIB_INT_ExternalFallingEdgeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsExternalINTEdgeSelect(INT_MODULE_ID index)

PLIB_INT_ExistsINTCPUPriority Function

Identifies whether the INTCPUPriority feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsINTCPUPriority(INT_MODULE_ID index);
```

Returns

- true - The INTCPUPriority feature is supported on the device
- false - The INTCPUPriority feature is not supported on the device

Description

This function identifies whether the INTCPUPriority feature is available on the Interrupt module. When this function returns true, this function is supported on the device:

- PLIB_INT_PriorityGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsINTCPUPriority(INT_MODULE_ID index)

PLIB_INT_ExistsINTCPUVector Function

Identifies whether the INTCPUVector feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsINTCPUVector(INT_MODULE_ID index);
```

Returns

- true - The INTCPUVector feature is supported on the device
- false - The INTCPUVector feature is not supported on the device

Description

This function identifies whether the INTCPUVector feature is available on the Interrupt module. When this function returns true, this function is supported on the device:

- PLIB_INT_VectorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsINTCPUVector(INT_MODULE_ID index)

PLIB_INT_ExistsProximityTimerControl Function

Identifies whether the ProximityTimerControl feature exists on the Interrupts module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsProximityTimerControl(INT_MODULE_ID index);
```

Returns

- true - The ProximityTimerControl feature is supported on the device
- false - The ProximityTimerControl feature is not supported on the device

Description

This function identifies whether the ProximityTimerControl feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_ProximityTimerSet
- PLIB_INT_ProximityTimerGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsProximityTimerControl(INT_MODULE_ID index)

PLIB_INT_ExistsProximityTimerEnable Function

Identifies whether the ProximityTimerEnable feature exists on the Interrupts module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsProximityTimerEnable(INT_MODULE_ID index);
```

Returns

- true - The ProximityTimerEnable feature is supported on the device
- false - The ProximityTimerEnable feature is not supported on the device

Description

This function identifies whether the ProximityTimerEnable feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_ProximityTimerEnable
- PLIB_INT_ProximityTimerDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsProximityTimerEnable(INT_MODULE_ID index)

PLIB_INT_ExistsSingleVectorShadowSet Function

Identifies whether the SingleVectorShadowSet feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsSingleVectorShadowSet(INT_MODULE_ID index);
```

Returns

- true - The SingleVectorShadowSet feature is supported on the device
- false - The SingleVectorShadowSet feature is not supported on the device

Description

This function identifies whether the SingleVectorShadowSet feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_SingleVectorShadowSetDisable
- PLIB_INT_SingleVectorShadowSetEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsSingleVectorShadowSet(INT_MODULE_ID index)

PLIB_INT_ExistsSourceControl Function

Identifies whether the SourceControl feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsSourceControl(INT_MODULE_ID index);
```

Returns

- true - The SourceControl feature is supported on the device
- false - The SourceControl feature is not supported on the device

Description

This function identifies whether the SourceControl feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_SourceEnable
- PLIB_INT_SourceDisable
- PLIB_INT_SourceIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsSourceControl(INT_MODULE_ID index)

PLIB_INT_ExistsSourceFlag Function

Identifies whether the SourceFlag feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsSourceFlag(INT_MODULE_ID index);
```

Returns

- true - The SourceFlag feature is supported on the device
- false - The SourceFlag feature is not supported on the device

Description

This function identifies whether the SourceFlag feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_SourceFlagGet
- PLIB_INT_SourceFlagSet

- PLIB_INT_SourceFlagClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsSourceFlag(INT_MODULE_ID index)

PLIB_INT_ExistsVectorPriority Function

Identifies whether the VectorPriority feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsVectorPriority(INT_MODULE_ID index);
```

Returns

- true - The VectorPriority feature is supported on the device
- false - The VectorPriority feature is not supported on the device

Description

This function identifies whether the VectorPriority feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_VectorPrioritySet
- PLIB_INT_VectorPriorityGet
- PLIB_INT_VectorSubPrioritySet
- PLIB_INT_VectorSubPriorityGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsVectorPriority(INT_MODULE_ID index)

PLIB_INT_ExistsVectorSelect Function

Identifies whether the VectorSelect feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsVectorSelect(INT_MODULE_ID index);
```

Returns

- true - The VectorSelect feature is supported on the device
- false - The VectorSelect feature is not supported on the device

Description

This function identifies whether the VectorSelect feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_MultiVectorSelect
- PLIB_INT_SingleVectorSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsVectorSelect(INT_MODULE_ID index)

PLIB_INT_ExistsShadowRegisterAssign Function

Identifies whether the ShadowRegisterAssign feature exists on the Interrupts module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsShadowRegisterAssign(INT_MODULE_ID index);
```

Returns

- true - The ShadowRegisterAssign feature is supported on the device
- false - The ShadowRegisterAssign feature is not supported on the device

Description

This function identifies whether the ShadowRegisterAssign feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_ShadowRegisterAssign
- PLIB_INT_ShadowRegisterGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsShadowRegisterAssign(INT_MODULE_ID index)

PLIB_INT_ExistsVariableOffset Function

Identifies whether the VariableOffset feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsVariableOffset(INT_MODULE_ID index);
```

Returns

- true - The VariableOffset feature is supported on the device
- false - The VariableOffset feature is not supported on the device

Description

This function identifies whether the VariableOffset feature is available on the Interrupt module. When this function returns true, these functions are supported on the device:

- PLIB_INT_VariableOffsetSet
- PLIB_INT_VariableOffsetGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsVariableOffset(INT_MODULE_ID index)

PLIB_INT_ExistsSoftwareNMI Function

Identifies whether the SoftwareNMI feature exists on the Interrupt module.

File

plib_int.h

C

```
bool PLIB_INT_ExistsSoftwareNMI(INT_MODULE_ID index);
```

Returns

- true - The SoftwareNMI feature is supported on the device
- false - The SoftwareNMI feature is not supported on the device

Description

This function identifies whether the SoftwareNMI feature is available on the Interrupt module. When this function returns true, this function is supported on the device:

- PLIB_INT_SoftwareNMITrigger

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_INT_ExistsSoftwareNMI(INT_MODULE_ID index)

e) Data Types and Constants

INT_EXTERNAL_SOURCES Enumeration

Lists the possible external sources that can cause an interrupt to occur.

File

plib_int_help.h

C

```
typedef enum {
  INT_EXTERNAL_INT_SOURCE0,
  INT_EXTERNAL_INT_SOURCE1,
  INT_EXTERNAL_INT_SOURCE2,
  INT_EXTERNAL_INT_SOURCE3,
  INT_EXTERNAL_INT_SOURCE4
} INT_EXTERNAL_SOURCES;
```

Members

Members	Description
INT_EXTERNAL_INT_SOURCE0	External Interrupt Source 0
INT_EXTERNAL_INT_SOURCE1	External Interrupt Source 1
INT_EXTERNAL_INT_SOURCE2	External Interrupt Source 2
INT_EXTERNAL_INT_SOURCE3	External Interrupt Source 3
INT_EXTERNAL_INT_SOURCE4	External Interrupt Source 4

Description

External Interrupt Sources

This enumeration lists the possible external sources that can cause an interrupt to occur.

Remarks

This enumeration is processor-specific and is defined in the processor-specific header files (see processor.h).

INT_PRIORITY_LEVEL Enumeration

Lists the possible interrupt priority levels.

File

plib_int_help.h

C

```
typedef enum {
  INT_PRIORITY_LEVEL0,
  INT_PRIORITY_LEVEL1,
  INT_PRIORITY_LEVEL2,
  INT_PRIORITY_LEVEL3,
  INT_PRIORITY_LEVEL4,
  INT_PRIORITY_LEVEL5,
  INT_PRIORITY_LEVEL6,
  INT_PRIORITY_LEVEL7
} INT_PRIORITY_LEVEL;
```

Members

Members	Description
INT_PRIORITY_LEVEL0	Disabled
INT_PRIORITY_LEVEL1	Priority 1
INT_PRIORITY_LEVEL2	Priority 2
INT_PRIORITY_LEVEL3	Priority 3
INT_PRIORITY_LEVEL4	Priority 4
INT_PRIORITY_LEVEL5	Priority 5

INT_PRIORITY_LEVEL6	Priority 6
INT_PRIORITY_LEVEL7	Priority 7

Description

Priority Level

This enumeration lists the possible interrupt priority levels that can be selected for each source.

Remarks

This enumeration is processor-specific and is defined in the processor-specific header files (see processor.h).

INT_SOURCE Enumeration

Lists the possible sources that can cause an interrupt to occur.

File

plib_int_help.h

C

```
typedef enum {
  INT_SOURCE_SOFTWARE_0,
  INT_SOURCE_SOFTWARE_1,
  INT_SOURCE_EXTERNAL_0,
  INT_SOURCE_EXTERNAL_1,
  INT_SOURCE_EXTERNAL_2,
  INT_SOURCE_EXTERNAL_3,
  INT_SOURCE_EXTERNAL_4,
  INT_SOURCE_TIMER_CORE,
  INT_SOURCE_TIMER_1,
  INT_SOURCE_TIMER_2,
  INT_SOURCE_TIMER_3,
  INT_SOURCE_TIMER_4,
  INT_SOURCE_TIMER_5,
  INT_SOURCE_INPUT_CAPTURE_1,
  INT_SOURCE_INPUT_CAPTURE_1_ERROR,
  INT_SOURCE_INPUT_CAPTURE_2,
  INT_SOURCE_INPUT_CAPTURE_2_ERROR,
  INT_SOURCE_INPUT_CAPTURE_3,
  INT_SOURCE_INPUT_CAPTURE_3_ERROR,
  INT_SOURCE_INPUT_CAPTURE_4,
  INT_SOURCE_INPUT_CAPTURE_4_ERROR,
  INT_SOURCE_INPUT_CAPTURE_5,
  INT_SOURCE_INPUT_CAPTURE_5_ERROR,
  INT_SOURCE_OUTPUT_COMPARE_1,
  INT_SOURCE_OUTPUT_COMPARE_2,
  INT_SOURCE_OUTPUT_COMPARE_3,
  INT_SOURCE_OUTPUT_COMPARE_4,
  INT_SOURCE_OUTPUT_COMPARE_5,
  INT_SOURCE_SPI_1_ERROR,
  INT_SOURCE_SPI_1_RECEIVE,
  INT_SOURCE_SPI_1_TRANSMIT,
  INT_SOURCE_SPI_2_ERROR,
  INT_SOURCE_SPI_2_RECEIVE,
  INT_SOURCE_SPI_2_TRANSMIT,
  INT_SOURCE_SPI_3_ERROR,
  INT_SOURCE_SPI_3_RECEIVE,
  INT_SOURCE_SPI_3_TRANSMIT,
  INT_SOURCE_SPI_4_ERROR,
  INT_SOURCE_SPI_4_RECEIVE,
  INT_SOURCE_SPI_4_TRANSMIT,
  INT_SOURCE_USART_1_ERROR,
  INT_SOURCE_USART_1_RECEIVE,
  INT_SOURCE_USART_1_TRANSMIT,
  INT_SOURCE_USART_2_ERROR,
  INT_SOURCE_USART_2_RECEIVE,
  INT_SOURCE_USART_2_TRANSMIT,
  INT_SOURCE_USART_3_ERROR,
  INT_SOURCE_USART_3_RECEIVE,
  INT_SOURCE_USART_3_TRANSMIT,
  INT_SOURCE_USART_4_ERROR,
  INT_SOURCE_USART_4_RECEIVE,
```

```
    INT_SOURCE_USART_4_TRANSMIT,
    INT_SOURCE_USART_5_ERROR,
    INT_SOURCE_USART_5_RECEIVE,
    INT_SOURCE_USART_5_TRANSMIT,
    INT_SOURCE_USART_6_ERROR,
    INT_SOURCE_USART_6_RECEIVE,
    INT_SOURCE_USART_6_TRANSMIT,
    INT_SOURCE_I2C_1_ERROR,
    INT_SOURCE_I2C_1_SLAVE,
    INT_SOURCE_I2C_1_MASTER,
    INT_SOURCE_I2C_2_ERROR,
    INT_SOURCE_I2C_2_SLAVE,
    INT_SOURCE_I2C_2_MASTER,
    INT_SOURCE_I2C_3_ERROR,
    INT_SOURCE_I2C_3_SLAVE,
    INT_SOURCE_I2C_3_MASTER,
    INT_SOURCE_I2C_4_ERROR,
    INT_SOURCE_I2C_4_SLAVE,
    INT_SOURCE_I2C_4_MASTER,
    INT_SOURCE_I2C_5_ERROR,
    INT_SOURCE_I2C_5_SLAVE,
    INT_SOURCE_I2C_5_MASTER,
    INT_SOURCE_CHANGE_NOTICE,
    INT_SOURCE_CHANGE_NOTICE_A,
    INT_SOURCE_CHANGE_NOTICE_B,
    INT_SOURCE_CHANGE_NOTICE_C,
    INT_SOURCE_CHANGE_NOTICE_D,
    INT_SOURCE_CHANGE_NOTICE_E,
    INT_SOURCE_CHANGE_NOTICE_F,
    INT_SOURCE_CHANGE_NOTICE_G,
    INT_SOURCE_CHANGE_NOTICE_H,
    INT_SOURCE_CHANGE_NOTICE_J,
    INT_SOURCE_CHANGE_NOTICE_K,
    INT_SOURCE_ADC_1,
    INT_SOURCE_ADC_1_DC1,
    INT_SOURCE_ADC_1_DC2,
    INT_SOURCE_ADC_1_DC3,
    INT_SOURCE_ADC_1_DC4,
    INT_SOURCE_ADC_1_DC5,
    INT_SOURCE_ADC_1_DC6,
    INT_SOURCE_ADC_1_DF1,
    INT_SOURCE_ADC_1_DF2,
    INT_SOURCE_ADC_1_DF3,
    INT_SOURCE_ADC_1_DF4,
    INT_SOURCE_ADC_1_DF5,
    INT_SOURCE_ADC_1_DF6,
    INT_SOURCE_ADC_1_DATA0,
    INT_SOURCE_ADC_1_DATA1,
    INT_SOURCE_ADC_1_DATA2,
    INT_SOURCE_ADC_1_DATA3,
    INT_SOURCE_ADC_1_DATA4,
    INT_SOURCE_ADC_1_DATA5,
    INT_SOURCE_ADC_1_DATA6,
    INT_SOURCE_ADC_1_DATA7,
    INT_SOURCE_ADC_1_DATA8,
    INT_SOURCE_ADC_1_DATA9,
    INT_SOURCE_ADC_1_DATA10,
    INT_SOURCE_ADC_1_DATA11,
    INT_SOURCE_ADC_1_DATA12,
    INT_SOURCE_ADC_1_DATA13,
    INT_SOURCE_ADC_1_DATA14,
    INT_SOURCE_ADC_1_DATA15,
    INT_SOURCE_ADC_1_DATA16,
    INT_SOURCE_ADC_1_DATA17,
    INT_SOURCE_ADC_1_DATA18,
    INT_SOURCE_ADC_1_DATA19,
    INT_SOURCE_ADC_1_DATA20,
    INT_SOURCE_ADC_1_DATA21,
    INT_SOURCE_ADC_1_DATA22,
    INT_SOURCE_ADC_1_DATA23,
    INT_SOURCE_ADC_1_DATA24,
    INT_SOURCE_ADC_1_DATA25,
    INT_SOURCE_ADC_1_DATA26,
    INT_SOURCE_ADC_1_DATA27,
    INT_SOURCE_ADC_1_DATA28,
```

```
    INT_SOURCE_ADC_1_DATA29,
    INT_SOURCE_ADC_1_DATA30,
    INT_SOURCE_ADC_1_DATA31,
    INT_SOURCE_ADC_1_DATA32,
    INT_SOURCE_ADC_1_DATA33,
    INT_SOURCE_ADC_1_DATA34,
    INT_SOURCE_ADC_1_DATA35,
    INT_SOURCE_ADC_1_DATA36,
    INT_SOURCE_ADC_1_DATA37,
    INT_SOURCE_ADC_1_DATA38,
    INT_SOURCE_ADC_1_DATA39,
    INT_SOURCE_ADC_1_DATA40,
    INT_SOURCE_ADC_1_DATA41,
    INT_SOURCE_ADC_1_DATA42,
    INT_SOURCE_ADC_1_DATA43,
    INT_SOURCE_ADC_1_DATA44,
    INT_SOURCE_ADC_END_OF_SCAN,
    INT_SOURCE_ADC_ANALOG_CIRCUIT_READY,
    INT_SOURCE_ADC_UPDATE_READY,
    INT_SOURCE_ADC_GROUP,
    INT_SOURCE_ADC_0_EARLY,
    INT_SOURCE_ADC_1_EARLY,
    INT_SOURCE_ADC_2_EARLY,
    INT_SOURCE_ADC_3_EARLY,
    INT_SOURCE_ADC_4_EARLY,
    INT_SOURCE_ADC_7_EARLY,
    INT_SOURCE_ADC_0_WARM,
    INT_SOURCE_ADC_1_WARM,
    INT_SOURCE_ADC_2_WARM,
    INT_SOURCE_ADC_3_WARM,
    INT_SOURCE_ADC_4_WARM,
    INT_SOURCE_ADC_7_WARM,
    INT_SOURCE_PARALLEL_PORT,
    INT_SOURCE_PARALLEL_PORT_ERROR,
    INT_SOURCE_COMPARATOR_1,
    INT_SOURCE_COMPARATOR_2,
    INT_SOURCE_CLOCK_MONITOR,
    INT_SOURCE_RTCC,
    INT_SOURCE_DMA_0,
    INT_SOURCE_DMA_1,
    INT_SOURCE_DMA_2,
    INT_SOURCE_DMA_3,
    INT_SOURCE_DMA_4,
    INT_SOURCE_DMA_5,
    INT_SOURCE_DMA_6,
    INT_SOURCE_DMA_7,
    INT_SOURCE_FLASH_CONTROL,
    INT_SOURCE_USB_1,
    INT_SOURCE_CAN_1,
    INT_SOURCE_CAN_2,
    INT_SOURCE_ETH_1,
    INT_SOURCE_ADC_FAULT,
    INT_SOURCE_CRYPTO,
    INT_SOURCE_SPI_5_RECEIVE,
    INT_SOURCE_SPI_5_TRANSMIT,
    INT_SOURCE_SPI_6_ERROR,
    INT_SOURCE_SPI_6_RECEIVE,
    INT_SOURCE_SPI_6_TRANSMIT
} INT_SOURCE;
```

Members

Members	Description
INT_SOURCE_SOFTWARE_0	Software interrupt 0
INT_SOURCE_SOFTWARE_1	Software interrupt 1
INT_SOURCE_EXTERNAL_0	External interrupt 0
INT_SOURCE_EXTERNAL_1	External interrupt 1
INT_SOURCE_EXTERNAL_2	External interrupt 2
INT_SOURCE_EXTERNAL_3	External interrupt 3
INT_SOURCE_EXTERNAL_4	External interrupt 4
INT_SOURCE_TIMER_CORE	Core timer interrupt
INT_SOURCE_TIMER_1	Timer 1 interrupt

INT_SOURCE_TIMER_2	Timer 2 interrupt
INT_SOURCE_TIMER_3	Timer 3 interrupt
INT_SOURCE_TIMER_4	Timer 4 interrupt
INT_SOURCE_TIMER_5	Timer 5 interrupt
INT_SOURCE_INPUT_CAPTURE_1	Input Capture 1 interrupt
INT_SOURCE_INPUT_CAPTURE_1_ERROR	Input Capture 1 Error interrupt
INT_SOURCE_INPUT_CAPTURE_2	Input Capture 2 interrupt
INT_SOURCE_INPUT_CAPTURE_2_ERROR	Input Capture 2 Error interrupt
INT_SOURCE_INPUT_CAPTURE_3	Input Capture 3 interrupt
INT_SOURCE_INPUT_CAPTURE_3_ERROR	Input Capture 3 Error interrupt
INT_SOURCE_INPUT_CAPTURE_4	Input Capture 4 interrupt
INT_SOURCE_INPUT_CAPTURE_4_ERROR	Input Capture 4 Error interrupt
INT_SOURCE_INPUT_CAPTURE_5	Input Capture 5 interrupt
INT_SOURCE_INPUT_CAPTURE_5_ERROR	Input Capture 5 Error interrupt
INT_SOURCE_OUTPUT_COMPARE_1	Output Compare 1 interrupt
INT_SOURCE_OUTPUT_COMPARE_2	Output Compare 2 interrupt
INT_SOURCE_OUTPUT_COMPARE_3	Output Compare 3 interrupt
INT_SOURCE_OUTPUT_COMPARE_4	Output Compare 4 interrupt
INT_SOURCE_OUTPUT_COMPARE_5	Output Compare 5 interrupt
INT_SOURCE_SPI_1_ERROR	SPI1 Error interrupt
INT_SOURCE_SPI_1_RECEIVE	SPI1 Receive Done interrupt
INT_SOURCE_SPI_1_TRANSMIT	SPI1 Transmit Done interrupt
INT_SOURCE_SPI_2_ERROR	SPI2 Error interrupt
INT_SOURCE_SPI_2_RECEIVE	SPI2 Receive Done interrupt
INT_SOURCE_SPI_2_TRANSMIT	SPI2 Transmit Done interrupt
INT_SOURCE_SPI_3_ERROR	SPI3 Error interrupt
INT_SOURCE_SPI_3_RECEIVE	SPI3 Receive Done interrupt
INT_SOURCE_SPI_3_TRANSMIT	SPI3 Transmit Done interrupt
INT_SOURCE_SPI_4_ERROR	SPI4 Error interrupt
INT_SOURCE_SPI_4_RECEIVE	SPI4 Receive Done interrupt
INT_SOURCE_SPI_4_TRANSMIT	SPI4 Transmit Done interrupt
INT_SOURCE_USART_1_ERROR	UART1 Error interrupt
INT_SOURCE_USART_1_RECEIVE	UART1 Receive interrupt
INT_SOURCE_USART_1_TRANSMIT	UART1 Transmit interrupt
INT_SOURCE_USART_2_ERROR	UART2 Error interrupt
INT_SOURCE_USART_2_RECEIVE	UART2 Receive interrupt
INT_SOURCE_USART_2_TRANSMIT	UART2 Transmit interrupt
INT_SOURCE_USART_3_ERROR	UART3 Error interrupt
INT_SOURCE_USART_3_RECEIVE	UART3 Receive interrupt
INT_SOURCE_USART_3_TRANSMIT	UART3 Transmit interrupt
INT_SOURCE_USART_4_ERROR	UART4 Error interrupt
INT_SOURCE_USART_4_RECEIVE	UART4 Receive interrupt
INT_SOURCE_USART_4_TRANSMIT	UART4 Transmit interrupt
INT_SOURCE_USART_5_ERROR	UART5 Error interrupt
INT_SOURCE_USART_5_RECEIVE	UART5 Receive interrupt
INT_SOURCE_USART_5_TRANSMIT	UART5 Transmit interrupt
INT_SOURCE_USART_6_ERROR	UART6 Error interrupt
INT_SOURCE_USART_6_RECEIVE	UART6 Receive interrupt
INT_SOURCE_USART_6_TRANSMIT	UART6 Transmit interrupt
INT_SOURCE_I2C_1_ERROR	I2C1 Bus Error Event interrupt
INT_SOURCE_I2C_1_SLAVE	I2C1 Slave Event interrupt
INT_SOURCE_I2C_1_MASTER	I2C1 Master Event interrupt
INT_SOURCE_I2C_2_ERROR	I2C2 Bus Error Event interrupt
INT_SOURCE_I2C_2_SLAVE	I2C2 Slave Event interrupt
INT_SOURCE_I2C_2_MASTER	I2C2 Master Event interrupt

INT_SOURCE_I2C_3_ERROR	I2C3 Bus Error Event interrupt
INT_SOURCE_I2C_3_SLAVE	I2C3 Slave Event interrupt
INT_SOURCE_I2C_3_MASTER	I2C3 Master Event interrupt
INT_SOURCE_I2C_4_ERROR	I2C4 Bus Error Event interrupt
INT_SOURCE_I2C_4_SLAVE	I2C4 Slave Event interrupt
INT_SOURCE_I2C_4_MASTER	I2C4 Master Event interrupt
INT_SOURCE_I2C_5_ERROR	I2C5 Bus Error Event interrupt
INT_SOURCE_I2C_5_SLAVE	I2C5 Slave Event interrupt
INT_SOURCE_I2C_5_MASTER	I2C5 Master Event interrupt
INT_SOURCE_CHANGE_NOTICE	Input Change Notice interrupt
INT_SOURCE_CHANGE_NOTICE_A	Input Change Notice Channel A interrupt
INT_SOURCE_CHANGE_NOTICE_B	Input Change Notice Channel B interrupt
INT_SOURCE_CHANGE_NOTICE_C	Input Change Notice Channel C interrupt
INT_SOURCE_CHANGE_NOTICE_D	Input Change Notice Channel D interrupt
INT_SOURCE_CHANGE_NOTICE_E	Input Change Notice Channel E interrupt
INT_SOURCE_CHANGE_NOTICE_F	Input Change Notice Channel F interrupt
INT_SOURCE_CHANGE_NOTICE_G	Input Change Notice Channel G interrupt
INT_SOURCE_CHANGE_NOTICE_H	Input Change Notice Channel H interrupt
INT_SOURCE_CHANGE_NOTICE_J	Input Change Notice Channel J interrupt
INT_SOURCE_CHANGE_NOTICE_K	Input Change Notice Channel K interrupt
INT_SOURCE_ADC_1	Analog-to-Digital Converter 1 interrupt
INT_SOURCE_ADC_1_DC1	ADC Digital Comparator 1
INT_SOURCE_ADC_1_DC2	ADC Digital Comparator 2
INT_SOURCE_ADC_1_DC3	ADC Digital Comparator 3
INT_SOURCE_ADC_1_DC4	ADC Digital Comparator 4
INT_SOURCE_ADC_1_DC5	ADC Digital Comparator 5
INT_SOURCE_ADC_1_DC6	ADC Digital Comparator 6
INT_SOURCE_ADC_1_DF1	ADC Digital Filter 1
INT_SOURCE_ADC_1_DF2	ADC Digital Filter 2
INT_SOURCE_ADC_1_DF3	ADC Digital Filter 3
INT_SOURCE_ADC_1_DF4	ADC Digital Filter 4
INT_SOURCE_ADC_1_DF5	ADC Digital Filter 5
INT_SOURCE_ADC_1_DF6	ADC Digital Filter 6
INT_SOURCE_ADC_1_DATA0	ADC Data 0
INT_SOURCE_ADC_1_DATA1	ADC Data 1
INT_SOURCE_ADC_1_DATA2	ADC Data 2
INT_SOURCE_ADC_1_DATA3	ADC Data 3
INT_SOURCE_ADC_1_DATA4	ADC Data 4
INT_SOURCE_ADC_1_DATA5	ADC Data 5
INT_SOURCE_ADC_1_DATA6	ADC Data 6
INT_SOURCE_ADC_1_DATA7	ADC Data 7
INT_SOURCE_ADC_1_DATA8	ADC Data 8
INT_SOURCE_ADC_1_DATA9	ADC Data 9
INT_SOURCE_ADC_1_DATA10	ADC Data 10
INT_SOURCE_ADC_1_DATA11	ADC Data 11
INT_SOURCE_ADC_1_DATA12	ADC Data 12
INT_SOURCE_ADC_1_DATA13	ADC Data 13
INT_SOURCE_ADC_1_DATA14	ADC Data 14
INT_SOURCE_ADC_1_DATA15	ADC Data 15
INT_SOURCE_ADC_1_DATA16	ADC Data 16
INT_SOURCE_ADC_1_DATA17	ADC Data 17
INT_SOURCE_ADC_1_DATA18	ADC Data 18
INT_SOURCE_ADC_1_DATA19	ADC Data 19
INT_SOURCE_ADC_1_DATA20	ADC Data 20
INT_SOURCE_ADC_1_DATA21	ADC Data 21

INT_SOURCE_ADC_1_DATA22	ADC Data 22
INT_SOURCE_ADC_1_DATA23	ADC Data 23
INT_SOURCE_ADC_1_DATA24	ADC Data 24
INT_SOURCE_ADC_1_DATA25	ADC Data 25
INT_SOURCE_ADC_1_DATA26	ADC Data 26
INT_SOURCE_ADC_1_DATA27	ADC Data 27
INT_SOURCE_ADC_1_DATA28	ADC Data 28
INT_SOURCE_ADC_1_DATA29	ADC Data 29
INT_SOURCE_ADC_1_DATA30	ADC Data 30
INT_SOURCE_ADC_1_DATA31	ADC Data 31
INT_SOURCE_ADC_1_DATA32	ADC Data 32
INT_SOURCE_ADC_1_DATA33	ADC Data 33
INT_SOURCE_ADC_1_DATA34	ADC Data 34
INT_SOURCE_ADC_1_DATA35	ADC Data 35
INT_SOURCE_ADC_1_DATA36	ADC Data 36
INT_SOURCE_ADC_1_DATA37	ADC Data 37
INT_SOURCE_ADC_1_DATA38	ADC Data 38
INT_SOURCE_ADC_1_DATA39	ADC Data 39
INT_SOURCE_ADC_1_DATA40	ADC Data 40
INT_SOURCE_ADC_1_DATA41	ADC Data 41
INT_SOURCE_ADC_1_DATA42	ADC Data 42
INT_SOURCE_ADC_1_DATA43	ADC Data 43
INT_SOURCE_ADC_1_DATA44	ADC Data 44
INT_SOURCE_ADC_END_OF_SCAN	ADC End of Scan
INT_SOURCE_ADC_ANALOG_CIRCUIT_READY	ADC Analog Circuit Ready
INT_SOURCE_ADC_UPDATE_READY	ADC Update Ready
INT_SOURCE_ADC_GROUP	ADC Group
INT_SOURCE_ADC_0_EARLY	ADC0 Early Interrupt
INT_SOURCE_ADC_1_EARLY	ADC1 Early Interrupt
INT_SOURCE_ADC_2_EARLY	ADC2 Early Interrupt
INT_SOURCE_ADC_3_EARLY	ADC3 Early Interrupt
INT_SOURCE_ADC_4_EARLY	ADC4 Early Interrupt
INT_SOURCE_ADC_7_EARLY	ADC7 Early Interrupt
INT_SOURCE_ADC_0_WARM	ADC0 Warm Interrupt
INT_SOURCE_ADC_1_WARM	ADC1 Warm Interrupt
INT_SOURCE_ADC_2_WARM	ADC2 Warm Interrupt
INT_SOURCE_ADC_3_WARM	ADC3 Warm Interrupt
INT_SOURCE_ADC_4_WARM	ADC4 Warm Interrupt
INT_SOURCE_ADC_7_WARM	ADC7 Warm Interrupt
INT_SOURCE_PARALLEL_PORT	Parallel Master Port interrupt
INT_SOURCE_PARALLEL_PORT_ERROR	Parallel Master Port Error interrupt
INT_SOURCE_COMPARATOR_1	Comparator 1 interrupt
INT_SOURCE_COMPARATOR_2	Comparator 2 interrupt
INT_SOURCE_CLOCK_MONITOR	Fail-Safe Clock Monitor interrupt
INT_SOURCE_RTCC	Real-Time Clock and Calender interrupt
INT_SOURCE_DMA_0	Direct Memory Access Channel 0 interrupt
INT_SOURCE_DMA_1	Direct Memory Access Channel 1 interrupt
INT_SOURCE_DMA_2	Direct Memory Access Channel 2 interrupt
INT_SOURCE_DMA_3	Direct Memory Access Channel 3 interrupt
INT_SOURCE_DMA_4	Direct Memory Access Channel 4 interrupt
INT_SOURCE_DMA_5	Direct Memory Access Channel 5 interrupt
INT_SOURCE_DMA_6	Direct Memory Access Channel 6 interrupt
INT_SOURCE_DMA_7	Direct Memory Access Channel 7 interrupt
INT_SOURCE_FLASH_CONTROL	Flash Control Event interrupt
INT_SOURCE_USB_1	Universal Serial Bus 1 interrupt

INT_SOURCE_CAN_1	Controller Area Network 1 interrupt
INT_SOURCE_CAN_2	Controller Area Network 2 interrupt
INT_SOURCE_ETH_1	Ethernet interrupt
INT_SOURCE_ADC_FAULT	ADC Fault interrupt
INT_SOURCE_CRYPTO	Crypto interrupt
INT_SOURCE_SPI_5_RECEIVE	SPI 5 Receive Done interrupt
INT_SOURCE_SPI_5_TRANSMIT	SPI 5 Transmit Done interrupt
INT_SOURCE_SPI_6_ERROR	SPI 6 Error interrupt
INT_SOURCE_SPI_6_RECEIVE	SPI 6 Receive Done interrupt
INT_SOURCE_SPI_6_TRANSMIT	SPI 6 Transmit Done interrupt

Description

Interrupt Source

This enumeration lists the possible sources that can cause an interrupt to occur.

Remarks

This enumeration is processor-specific and is defined in the processor-specific header files (see processor.h).

INT_SUBPRIORITY_LEVEL Enumeration

Lists the possible interrupt sub priority levels.

File

plib_int_help.h

C

```
typedef enum {
  INT_SUBPRIORITY_LEVEL0,
  INT_SUBPRIORITY_LEVEL1,
  INT_SUBPRIORITY_LEVEL2,
  INT_SUBPRIORITY_LEVEL3
} INT_SUBPRIORITY_LEVEL;
```

Members

Members	Description
INT_SUBPRIORITY_LEVEL0	Sub Priority 0
INT_SUBPRIORITY_LEVEL1	Sub Priority 1
INT_SUBPRIORITY_LEVEL2	Sub Priority 2
INT_SUBPRIORITY_LEVEL3	Sub Priority 3

Description

Sub Priority Level

This enumeration lists the possible interrupt sub priority levels that can be selected for each source.

Remarks

This enumeration (INT_SUBPRIORITY_LEVEL) is processor-specific and is defined in the processor-specific header files (see processor.h).

INT_VECTOR Enumeration

Lists the possible interrupt vectors.

File

plib_int_help.h

C

```
typedef enum {
  INT_VECTOR_CT,
  INT_VECTOR_CS0,
  INT_VECTOR_CS1,
  INT_VECTOR_INT0,
  INT_VECTOR_T1,
```

```
    INT_VECTOR_IC1,
    INT_VECTOR_OC1,
    INT_VECTOR_INT1,
    INT_VECTOR_T2,
    INT_VECTOR_IC2,
    INT_VECTOR_OC2,
    INT_VECTOR_INT2,
    INT_VECTOR_T3,
    INT_VECTOR_IC3,
    INT_VECTOR_OC3,
    INT_VECTOR_INT3,
    INT_VECTOR_T4,
    INT_VECTOR_IC4,
    INT_VECTOR_OC4,
    INT_VECTOR_INT4,
    INT_VECTOR_T5,
    INT_VECTOR_IC5,
    INT_VECTOR_OC5,
    INT_VECTOR_SPI1,
    INT_VECTOR_UART1,
    INT_VECTOR_SPI3,
    INT_VECTOR_I2C3,
    INT_VECTOR_I2C1,
    INT_VECTOR_CN,
    INT_VECTOR_AD1,
    INT_VECTOR_PMP,
    INT_VECTOR_CMP1,
    INT_VECTOR_CMP2,
    INT_VECTOR_UART3,
    INT_VECTOR_SPI2,
    INT_VECTOR_I2C4,
    INT_VECTOR_UART2,
    INT_VECTOR_SPI4,
    INT_VECTOR_I2C5,
    INT_VECTOR_I2C2,
    INT_VECTOR_FSCM,
    INT_VECTOR_RTCC,
    INT_VECTOR_DMA0,
    INT_VECTOR_DMA1,
    INT_VECTOR_DMA2,
    INT_VECTOR_DMA3,
    INT_VECTOR_DMA4,
    INT_VECTOR_DMA5,
    INT_VECTOR_DMA6,
    INT_VECTOR_DMA7,
    INT_VECTOR_FCE,
    INT_VECTOR_USB,
    INT_VECTOR_CAN1,
    INT_VECTOR_CAN2,
    INT_VECTOR_ETH,
    INT_VECTOR_UART4,
    INT_VECTOR_UART6,
    INT_VECTOR_UART5,
    INT_VECTOR_ADC1,
    INT_VECTOR_ADC1_DC1,
    INT_VECTOR_ADC1_DC2,
    INT_VECTOR_ADC1_DC3,
    INT_VECTOR_ADC1_DC4,
    INT_VECTOR_ADC1_DC5,
    INT_VECTOR_ADC1_DC6,
    INT_VECTOR_ADC1_DF1,
    INT_VECTOR_ADC1_DF2,
    INT_VECTOR_ADC1_DF3,
    INT_VECTOR_ADC1_DF4,
    INT_VECTOR_ADC1_DF5,
    INT_VECTOR_ADC1_DF6,
    INT_VECTOR_ADC1_DATA0,
    INT_VECTOR_ADC1_DATA1,
    INT_VECTOR_ADC1_DATA2,
    INT_VECTOR_ADC1_DATA3,
    INT_VECTOR_ADC1_DATA4,
    INT_VECTOR_ADC1_DATA5,
    INT_VECTOR_ADC1_DATA6,
    INT_VECTOR_ADC1_DATA7,
    INT_VECTOR_ADC1_DATA8,
```

```
    INT_VECTOR_ADC1_DATA9,
    INT_VECTOR_ADC1_DATA10,
    INT_VECTOR_ADC1_DATA11,
    INT_VECTOR_ADC1_DATA12,
    INT_VECTOR_ADC1_DATA13,
    INT_VECTOR_ADC1_DATA14,
    INT_VECTOR_ADC1_DATA15,
    INT_VECTOR_ADC1_DATA16,
    INT_VECTOR_ADC1_DATA17,
    INT_VECTOR_ADC1_DATA18,
    INT_VECTOR_ADC1_DATA19,
    INT_VECTOR_ADC1_DATA20,
    INT_VECTOR_ADC1_DATA21,
    INT_VECTOR_ADC1_DATA22,
    INT_VECTOR_ADC1_DATA23,
    INT_VECTOR_ADC1_DATA24,
    INT_VECTOR_ADC1_DATA25,
    INT_VECTOR_ADC1_DATA26,
    INT_VECTOR_ADC1_DATA27,
    INT_VECTOR_ADC1_DATA28,
    INT_VECTOR_ADC1_DATA29,
    INT_VECTOR_ADC1_DATA30,
    INT_VECTOR_ADC1_DATA31,
    INT_VECTOR_ADC1_DATA32,
    INT_VECTOR_ADC1_DATA33,
    INT_VECTOR_ADC1_DATA34,
    INT_VECTOR_ADC1_DATA35,
    INT_VECTOR_ADC1_DATA36,
    INT_VECTOR_ADC1_DATA37,
    INT_VECTOR_ADC1_DATA38,
    INT_VECTOR_ADC1_DATA39,
    INT_VECTOR_ADC1_DATA40,
    INT_VECTOR_ADC1_DATA41,
    INT_VECTOR_ADC1_DATA42,
    INT_VECTOR_ADC1_DATA43,
    INT_VECTOR_ADC1_DATA44,
    INT_VECTOR_ADC_END_OF_SCAN,
    INT_VECTOR_ADC_ANALOG_CIRCUIT_READY,
    INT_VECTOR_ADC_UPDATE_READY,
    INT_VECTOR_ADC_GROUP,
    INT_VECTOR_ADC_0_EARLY,
    INT_VECTOR_ADC_1_EARLY,
    INT_VECTOR_ADC_2_EARLY,
    INT_VECTOR_ADC_3_EARLY,
    INT_VECTOR_ADC_4_EARLY,
    INT_VECTOR_ADC_7_EARLY,
    INT_VECTOR_ADC_0_WARM,
    INT_VECTOR_ADC_1_WARM,
    INT_VECTOR_ADC_2_WARM,
    INT_VECTOR_ADC_3_WARM,
    INT_VECTOR_ADC_4_WARM,
    INT_VECTOR_ADC_7_WARM,
    INT_VECTOR_ADC_FAULT,
    INT_VECTOR_CRYPTO,
    INT_VECTOR_I2C2_BUS,
    INT_VECTOR_I2C2_SLAVE,
    INT_VECTOR_I2C2_MASTER,
    INT_VECTOR_SPI5_FAULT,
    INT_VECTOR_SPI5_RX,
    INT_VECTOR_SPI5_TX,
    INT_VECTOR_SPI6_FAULT,
    INT_VECTOR_SPI6_RX,
    INT_VECTOR_SPI6_TX,
    INT_VECTOR_CHANGE_NOTICE_A,
    INT_VECTOR_CHANGE_NOTICE_B,
    INT_VECTOR_CHANGE_NOTICE_C,
    INT_VECTOR_CHANGE_NOTICE_D,
    INT_VECTOR_CHANGE_NOTICE_E,
    INT_VECTOR_CHANGE_NOTICE_F,
    INT_VECTOR_CHANGE_NOTICE_G,
    INT_VECTOR_CHANGE_NOTICE_H,
    INT_VECTOR_CHANGE_NOTICE_J,
    INT_VECTOR_CHANGE_NOTICE_K
} INT_VECTOR;
```

Members

Members	Description
INT_VECTOR_CT	Core Timer Interrupt
INT_VECTOR_CS0	Core Software Interrupt 0
INT_VECTOR_CS1	Core Software Interrupt 1
INT_VECTOR_INT0	External Interrupt 0
INT_VECTOR_T1	Timer 1 Interrupt
INT_VECTOR_IC1	Input Capture 1 Interrupt
INT_VECTOR_OC1	Output Compare 1 Interrupt
INT_VECTOR_INT1	External Interrupt 1
INT_VECTOR_T2	Timer 2 Interrupt
INT_VECTOR_IC2	Input Capture 2 Interrupt
INT_VECTOR_OC2	Output Compare 2 Interrupt
INT_VECTOR_INT2	External Interrupt 2
INT_VECTOR_T3	Timer 3 Interrupt
INT_VECTOR_IC3	Input Capture 3 Interrupt
INT_VECTOR_OC3	Output Compare 3 Interrupt
INT_VECTOR_INT3	External Interrupt 3
INT_VECTOR_T4	Timer 4 Interrupt
INT_VECTOR_IC4	Input Capture 4 Interrupt
INT_VECTOR_OC4	Output Compare 4 Interrupt
INT_VECTOR_INT4	External Interrupt 4
INT_VECTOR_T5	Timer 5 Interrupt
INT_VECTOR_IC5	Input Capture 5 Interrupt
INT_VECTOR_OC5	Output Compare 5 Interrupt
INT_VECTOR_SPI1	SPI 1 Interrupt
INT_VECTOR_UART1	UART 1 Interrupt
INT_VECTOR_SPI3	SPI 3 Interrupt
INT_VECTOR_I2C3	I2C 3 Interrupt
INT_VECTOR_I2C1	I2C 1 Interrupt
INT_VECTOR_CN	Change Notification Interrupt
INT_VECTOR_AD1	ADC Interrupt
INT_VECTOR_PMP	PMP Interrupt
INT_VECTOR_CMP1	Comparator 1 Interrupt
INT_VECTOR_CMP2	Comparator 2 Interrupt
INT_VECTOR_UART3	UART 3 Interrupt
INT_VECTOR_SPI2	SPI 2 Interrupt
INT_VECTOR_I2C4	I2C 4 Interrupt
INT_VECTOR_UART2	UART 2 Interrupt
INT_VECTOR_SPI4	SPI 4 Interrupt
INT_VECTOR_I2C5	I2C 5 Interrupt
INT_VECTOR_I2C2	I2C 2 Interrupt
INT_VECTOR_FSCM	Fail Safe Clock Monitor Interrupt
INT_VECTOR_RTCC	RTCC Interrupt
INT_VECTOR_DMA0	DMA 0 Interrupt
INT_VECTOR_DMA1	DMA 1 Interrupt
INT_VECTOR_DMA2	DMA 2 Interrupt
INT_VECTOR_DMA3	DMA 3 Interrupt
INT_VECTOR_DMA4	DMA 4 Interrupt
INT_VECTOR_DMA5	DMA 5 Interrupt
INT_VECTOR_DMA6	DMA 6 Interrupt
INT_VECTOR_DMA7	DMA 7 Interrupt
INT_VECTOR_FCE	Flash Control Event Interrupt
INT_VECTOR_USB	USB Interrupt

INT_VECTOR_CAN1	CAN 1 Interrupt
INT_VECTOR_CAN2	CAN 2 Interrupt
INT_VECTOR_ETH	Ethernet Interrupt
INT_VECTOR_UART4	UART 4 Interrupt
INT_VECTOR_UART6	UART 6 Interrupt
INT_VECTOR_UART5	UART 5 Interrupt
INT_VECTOR_ADC1	Analog-to-Digital Converter 1 interrupt
INT_VECTOR_ADC1_DC1	ADC Digital Comparator 1
INT_VECTOR_ADC1_DC2	ADC Digital Comparator 2
INT_VECTOR_ADC1_DC3	ADC Digital Comparator 3
INT_VECTOR_ADC1_DC4	ADC Digital Comparator 4
INT_VECTOR_ADC1_DC5	ADC Digital Comparator 5
INT_VECTOR_ADC1_DC6	ADC Digital Comparator 6
INT_VECTOR_ADC1_DF1	ADC Digital Filter 1
INT_VECTOR_ADC1_DF2	ADC Digital Filter 2
INT_VECTOR_ADC1_DF3	ADC Digital Filter 3
INT_VECTOR_ADC1_DF4	ADC Digital Filter 4
INT_VECTOR_ADC1_DF5	ADC Digital Filter 5
INT_VECTOR_ADC1_DF6	ADC Digital Filter 6
INT_VECTOR_ADC1_DATA0	ADC Data 0
INT_VECTOR_ADC1_DATA1	ADC Data 1
INT_VECTOR_ADC1_DATA2	ADC Data 2
INT_VECTOR_ADC1_DATA3	ADC Data 3
INT_VECTOR_ADC1_DATA4	ADC Data 4
INT_VECTOR_ADC1_DATA5	ADC Data 5
INT_VECTOR_ADC1_DATA6	ADC Data 6
INT_VECTOR_ADC1_DATA7	ADC Data 7
INT_VECTOR_ADC1_DATA8	ADC Data 8
INT_VECTOR_ADC1_DATA9	ADC Data 9
INT_VECTOR_ADC1_DATA10	ADC Data 10
INT_VECTOR_ADC1_DATA11	ADC Data 11
INT_VECTOR_ADC1_DATA12	ADC Data 12
INT_VECTOR_ADC1_DATA13	ADC Data 13
INT_VECTOR_ADC1_DATA14	ADC Data 14
INT_VECTOR_ADC1_DATA15	ADC Data 15
INT_VECTOR_ADC1_DATA16	ADC Data 16
INT_VECTOR_ADC1_DATA17	ADC Data 17
INT_VECTOR_ADC1_DATA18	ADC Data 18
INT_VECTOR_ADC1_DATA19	ADC Data 19
INT_VECTOR_ADC1_DATA20	ADC Data 20
INT_VECTOR_ADC1_DATA21	ADC Data 21
INT_VECTOR_ADC1_DATA22	ADC Data 22
INT_VECTOR_ADC1_DATA23	ADC Data 23
INT_VECTOR_ADC1_DATA24	ADC Data 24
INT_VECTOR_ADC1_DATA25	ADC Data 25
INT_VECTOR_ADC1_DATA26	ADC Data 26
INT_VECTOR_ADC1_DATA27	ADC Data 27
INT_VECTOR_ADC1_DATA28	ADC Data 28
INT_VECTOR_ADC1_DATA29	ADC Data 29
INT_VECTOR_ADC1_DATA30	ADC Data 30
INT_VECTOR_ADC1_DATA31	ADC Data 31
INT_VECTOR_ADC1_DATA32	ADC Data 32
INT_VECTOR_ADC1_DATA33	ADC Data 33
INT_VECTOR_ADC1_DATA34	ADC Data 34
INT_VECTOR_ADC1_DATA35	ADC Data 35

INT_VECTOR_ADC1_DATA36	ADC Data 36
INT_VECTOR_ADC1_DATA37	ADC Data 37
INT_VECTOR_ADC1_DATA38	ADC Data 38
INT_VECTOR_ADC1_DATA39	ADC Data 39
INT_VECTOR_ADC1_DATA40	ADC Data 40
INT_VECTOR_ADC1_DATA41	ADC Data 41
INT_VECTOR_ADC1_DATA42	ADC Data 42
INT_VECTOR_ADC1_DATA43	ADC Data 43
INT_VECTOR_ADC1_DATA44	ADC Data 44
INT_VECTOR_ADC_END_OF_SCAN	ADC End of Scan
INT_VECTOR_ADC_ANALOG_CIRCUIT_READY	ADC Analog Circuit Ready
INT_VECTOR_ADC_UPDATE_READY	ADC Update Ready
INT_VECTOR_ADC_GROUP	ADC Group
INT_VECTOR_ADC_0_EARLY	ADC0 Early Interrupt
INT_VECTOR_ADC_1_EARLY	ADC1 Early Interrupt
INT_VECTOR_ADC_2_EARLY	ADC2 Early Interrupt
INT_VECTOR_ADC_3_EARLY	ADC3 Early Interrupt
INT_VECTOR_ADC_4_EARLY	ADC4 Early Interrupt
INT_VECTOR_ADC_7_EARLY	ADC7 Early Interrupt
INT_VECTOR_ADC_0_WARM	ADC0 Warm Interrupt
INT_VECTOR_ADC_1_WARM	ADC1 Warm Interrupt
INT_VECTOR_ADC_2_WARM	ADC2 Warm Interrupt
INT_VECTOR_ADC_3_WARM	ADC3 Warm Interrupt
INT_VECTOR_ADC_4_WARM	ADC4 Warm Interrupt
INT_VECTOR_ADC_7_WARM	ADC7 Warm Interrupt
INT_VECTOR_ADC_FAULT	ADC Fault interrupt
INT_VECTOR_CRYPTO	Crypto interrupt
INT_VECTOR_I2C2_BUS	I2C 2 Bus Interrupt
INT_VECTOR_I2C2_SLAVE	I2C 2 Bus Interrupt
INT_VECTOR_I2C2_MASTER	I2C 2 Bus Interrupt
INT_VECTOR_SPI5_FAULT	SPI 5 Error interrupt
INT_VECTOR_SPI5_RX	SPI 5 Receive Done interrupt
INT_VECTOR_SPI5_TX	SPI 5 Transmit Done interrupt
INT_VECTOR_SPI6_FAULT	SPI 6 Error interrupt
INT_VECTOR_SPI6_RX	SPI 6 Receive Done interrupt
INT_VECTOR_SPI6_TX	SPI 6 Transmit Done interrupt
INT_VECTOR_CHANGE_NOTICE_A	Input Change Notice Channel A interrupt
INT_VECTOR_CHANGE_NOTICE_B	Input Change Notice Channel B interrupt
INT_VECTOR_CHANGE_NOTICE_C	Input Change Notice Channel C interrupt
INT_VECTOR_CHANGE_NOTICE_D	Input Change Notice Channel D interrupt
INT_VECTOR_CHANGE_NOTICE_E	Input Change Notice Channel E interrupt
INT_VECTOR_CHANGE_NOTICE_F	Input Change Notice Channel F interrupt
INT_VECTOR_CHANGE_NOTICE_G	Input Change Notice Channel G interrupt
INT_VECTOR_CHANGE_NOTICE_H	Input Change Notice Channel H interrupt
INT_VECTOR_CHANGE_NOTICE_J	Input Change Notice Channel J interrupt
INT_VECTOR_CHANGE_NOTICE_K	Input Change Notice Channel K interrupt

Description

Interrupt Vectors

This enumeration lists the possible interrupt vectors.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

INT_STATE_GLOBAL Type

Data type defining the global interrupt state.

File

plib_int.h

C

```
typedef uint32_t INT_STATE_GLOBAL;
```

Description

INT global state type definition

This data type is used for interrupt enable and disable functions.

Remarks

None.

INT_SHADOW_REGISTER Enumeration

Lists the possible shadow register sets.

File

plib_int_help.h

C

```
typedef enum {
  INT_SHADOW_REGISTER_0,
  INT_SHADOW_REGISTER_1,
  INT_SHADOW_REGISTER_2,
  INT_SHADOW_REGISTER_3,
  INT_SHADOW_REGISTER_4,
  INT_SHADOW_REGISTER_5,
  INT_SHADOW_REGISTER_6,
  INT_SHADOW_REGISTER_7
} INT_SHADOW_REGISTER;
```

Members

Members	Description
INT_SHADOW_REGISTER_0	Shadow register set 0
INT_SHADOW_REGISTER_1	Shadow register set 1
INT_SHADOW_REGISTER_2	Shadow register set 2
INT_SHADOW_REGISTER_3	Shadow register set 3
INT_SHADOW_REGISTER_4	Shadow register set 4
INT_SHADOW_REGISTER_5	Shadow register set 5
INT_SHADOW_REGISTER_6	Shadow register set 6
INT_SHADOW_REGISTER_7	Shadow register set 7

Description

Shadow Register Sets

This enumeration lists the possible shadow register sets.

Remarks

This feature may not be available on all the devices. Refer to the specific device header files for availability.

INT_VECTOR_SPACING Enumeration

Lists the possible interrupt vector spacing.

File

plib_int_help.h

C

```
typedef enum {
  INT_VECTOR_SPACING_0,
  INT_VECTOR_SPACING_8,
  INT_VECTOR_SPACING_16,
  INT_VECTOR_SPACING_32,
  INT_VECTOR_SPACING_64,
  INT_VECTOR_SPACING_128,
  INT_VECTOR_SPACING_256,
  INT_VECTOR_SPACING_512
} INT_VECTOR_SPACING;
```

Members

Members	Description
INT_VECTOR_SPACING_0	0 Byte Vector Spacing
INT_VECTOR_SPACING_8	8 Byte Vector Spacing
INT_VECTOR_SPACING_16	16 Byte Vector Spacing
INT_VECTOR_SPACING_32	32 Byte Vector Spacing
INT_VECTOR_SPACING_64	64 Byte Vector Spacing
INT_VECTOR_SPACING_128	128 Byte Vector Spacing
INT_VECTOR_SPACING_256	256 Byte Vector Spacing
INT_VECTOR_SPACING_512	512 Byte Vector Spacing

Description

Interrupt Vector Spacing

This enumeration lists the possible interrupt vector spacing, which can be provided between two interrupt vectors.

Remarks

This feature may not be available in all the devices. Refer to the specific device header files for availability.

Files

Files

Name	Description
plib_int.h	Defines the Interrupt Peripheral Library interface
plib_int_help.h	

Description

This section lists the source and header files used by the library.

plib_int.h

Defines the Interrupt Peripheral Library interface

Functions

	Name	Description
	PLIB_INT_CPUCurrentPriorityLevelGet	Gets the interrupt priority level at which the CPU is currently operating.
	PLIB_INT_Disable	Disables the generation of interrupts to the CPU.
	PLIB_INT_Enable	Enables the generation of interrupts to the CPU.
	PLIB_INT_ExistsCPUCurrentPriorityLevel	Identifies whether the CPUCurrentPriorityLevel feature exists on the Interrupts module.
	PLIB_INT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Interrupt module.
	PLIB_INT_ExistsExternalINTEdgeSelect	Identifies whether the ExternalINTEdgeSelect feature exists on the Interrupts module.
	PLIB_INT_ExistsINTCPUPriority	Identifies whether the INTCPUPriority feature exists on the Interrupt module.
	PLIB_INT_ExistsINTCPUVector	Identifies whether the INTCPUVector feature exists on the Interrupt module.
	PLIB_INT_ExistsProximityTimerControl	Identifies whether the ProximityTimerControl feature exists on the Interrupts module.
	PLIB_INT_ExistsProximityTimerEnable	Identifies whether the ProximityTimerEnable feature exists on the Interrupts module.
	PLIB_INT_ExistsShadowRegisterAssign	Identifies whether the ShadowRegisterAssign feature exists on the Interrupts module.
	PLIB_INT_ExistsSingleVectorShadowSet	Identifies whether the SingleVectorShadowSet feature exists on the Interrupt module.

	PLIB_INT_ExistsSoftwareNMI	Identifies whether the SoftwareNMI feature exists on the Interrupt module.
	PLIB_INT_ExistsSourceControl	Identifies whether the SourceControl feature exists on the Interrupt module.
	PLIB_INT_ExistsSourceFlag	Identifies whether the SourceFlag feature exists on the Interrupt module.
	PLIB_INT_ExistsVariableOffset	Identifies whether the VariableOffset feature exists on the Interrupt module.
	PLIB_INT_ExistsVectorPriority	Identifies whether the VectorPriority feature exists on the Interrupt module.
	PLIB_INT_ExistsVectorSelect	Identifies whether the VectorSelect feature exists on the Interrupt module.
	PLIB_INT_ExternalFallingEdgeSelect	Selects the falling edge as the edge polarity of the external interrupt.
	PLIB_INT_ExternalRisingEdgeSelect	Selects the rising edge as the edge polarity of the external interrupt.
	PLIB_INT_GetStateAndDisable	Disables the generation of interrupts to the CPU.
	PLIB_INT_IsEnabled	Identifies if interrupts are currently enabled or disabled at the top level.
	PLIB_INT_MultiVectorSelect	Configures the Interrupt Controller for Multiple Vector mode.
	PLIB_INT_PriorityGet	Returns the priority level of the latest interrupt presented to the CPU.
	PLIB_INT_ProximityTimerDisable	Disables the interrupt temporal-proximity timer.
	PLIB_INT_ProximityTimerEnable	Enables the interrupt temporal-proximity timer and selects the priority levels that start the timer.
	PLIB_INT_ProximityTimerGet	Returns the value used by the temporal proximity timer as a reload value when the temporal proximity timer is triggered by an interrupt event.
	PLIB_INT_ProximityTimerSet	Sets the temporal proximity timer reload value. This value is used to reload the proximity timer after it has been triggered by an interrupt event.
	PLIB_INT_SetState	Restores the status of CPU interrupts based on the value passed into the function.
	PLIB_INT_ShadowRegisterAssign	Assigns a shadow register set for an interrupt priority level.
	PLIB_INT_ShadowRegisterGet	Gets the shadow register set assigned for an interrupt priority level.
	PLIB_INT_SingleVectorSelect	Configures the Interrupt Controller for Single Vector mode.
	PLIB_INT_SingleVectorShadowSetDisable	Disables the Shadow Register Set in Single Vector mode.
	PLIB_INT_SingleVectorShadowSetEnable	Enables the Shadow Register Set in Single Vector mode.
	PLIB_INT_SoftwareNMITrigger	Triggers software NMI.
	PLIB_INT_SourceDisable	Disables the interrupt source
	PLIB_INT_SourceEnable	Enables the interrupt source.
	PLIB_INT_SourceFlagClear	Clears the status of the interrupt flag for the selected source.
	PLIB_INT_SourceFlagGet	Returns the status of the interrupt flag for the selected source.
	PLIB_INT_SourceFlagSet	Sets the status of the interrupt flag for the selected source.
	PLIB_INT_SourceIsEnabled	Gets the enable state of the interrupt source.
	PLIB_INT_VariableVectorOffsetGet	Gets the offset specific to an interrupt source number.
	PLIB_INT_VariableVectorOffsetSet	Sets the offset specific to an interrupt source number.
	PLIB_INT_VectorGet	Returns the interrupt vector that is presented to the CPU.
	PLIB_INT_VectorPriorityGet	Gets the priority of the interrupt vector.
	PLIB_INT_VectorPrioritySet	Sets the priority of the interrupt vector.
	PLIB_INT_VectorSubPriorityGet	Gets the sub-priority of the interrupt vector.
	PLIB_INT_VectorSubPrioritySet	Sets the sub-priority of the interrupt vector.

Types

	Name	Description
	INT_STATE_GLOBAL	Data type defining the global interrupt state.

Description

Interrupt Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Interrupt Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Interrupt Controller module.

File Name

plib_int.h

Company

Microchip Technology Inc.

plib_int_help.h

Enumerations

	Name	Description
	INT_EXTERNAL_SOURCES	Lists the possible external sources that can cause an interrupt to occur.
	INT_PRIORITY_LEVEL	Lists the possible interrupt priority levels.
	INT_SHADOW_REGISTER	Lists the possible shadow register sets.
	INT_SOURCE	Lists the possible sources that can cause an interrupt to occur.
	INT_SUBPRIORITY_LEVEL	Lists the possible interrupt sub priority levels.
	INT_VECTOR	Lists the possible interrupt vectors.
	INT_VECTOR_SPACING	Lists the possible interrupt vector spacing.

Section

Interrupt Controller Definitions - Processor-Specific Enumerations

****

****

This section defines Interrupt Controller enumerations that are

processor-specific.

NVM Peripheral Library

This section describes the Non-volatile Memory (NVM) Peripheral Library.

Introduction

This library provides a low-level abstraction of the Non-Volatile Memory (NVM)component (Flash and EEPROM) on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

Flash Program Memory

The Flash program memory is readable, writeable and erasable during normal operation over the entire operating voltage range.

A read from program memory is executed at one byte/word at a time depending on the width of the data bus.

A write to the program memory is executed in either blocks of specific sizes or a single word depending on the type of processor used.

An erase is performed in blocks. A bulk erase may be performed from user code depending on the type of processor supporting the operation.

Writing or erasing program memory will cease instruction fetches until the operation is complete thus restricting memory access and therefore preventing code execution. This is controlled by an internal programming timer.

There are three processor dependant methods for Flash program memory modification.

- Run-Time Self Programming (RTSP)
- In-Circuit Serial Programming (ICSP)
- EJTAG programming

Note: This topic covers the RTSP techniques.

EEPROM Memory

The EEPROM memory is the data storage memory.

On device power-up, power is disconnected from the EEPROM and the EEPROM will be in its lowest power mode. When it is enabled, power is applied to the EEPROM, but it will be ready for the access only after a power-up delay. Once ready for access, all read and write operations are executed using the Data EEPROM registers.

Write/Read/Erase operations are possible in words.

A bulk erase may be performed to erase the full EEPROM.

Using the Library

This topic describes the basic architecture of the NVM Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_nvm.h`

The interface to the NVM library is defined in the `plib_nvm.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the NVM library must include `peripheral.h`.

Library File:

The NVM peripheral library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

Hardware Abstraction Model

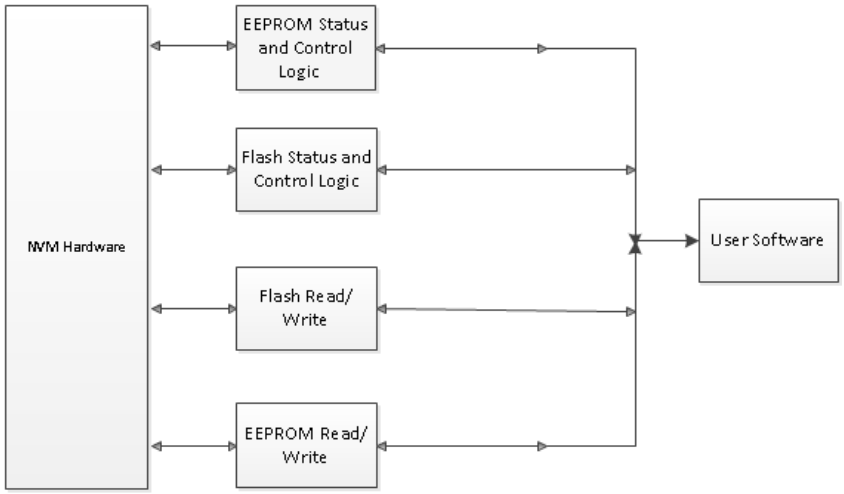
This library provides a low-level abstraction of the NVM module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Abstraction Model

The NVM Peripheral Library provides interface routines to interact with the blocks shown in the diagram.

NVM Software Abstraction Block Diagram



The **Flash Status and Control** logic ensures that the Flash memory is configured appropriately for modification. It also provides the status of the different operations in progress as well as errors, if any. It also decides if the operation is for Flash program memory or the special device Configuration registers.

The **Flash Read/Write** block ensures that the user data is written to/read from the program memory holding latches. It provides a layer of abstraction over the sequence of processor specific instructions which are required to access this data. Depending on the processor type, either block or Word programming options are available.

The **EEPROM Status and Control** logic ensures that the EEPROM is configured appropriately for access and also provides the status of the different operations in progress, as well as errors, if any.

The **EEPROM Read/Write** block ensures that the user data is written to/read from the data EEPROM. All of the EEPROM operations are word programmable with the exception of the bulk erase, which erases the entire EEPROM.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the NVM module

Library Interface Section	Description
Flash Memory Configuration and Status Functions	Provides Flash memory setup, configuration, and status control interface routines.
Flash Memory Functions	Provides Flash memory interface routines.
Other Flash Memory Functions	Provides additional functions for Boot page and Flash memory.

EEPROM Configuration and Status Functions	Provides EEPROM setup, configuration, and status control interface routines.
EEPROM Operation Functions	Provides interface routines for EEPROM operations.
Feature Existence Functions	Determine whether or not certain features are available.

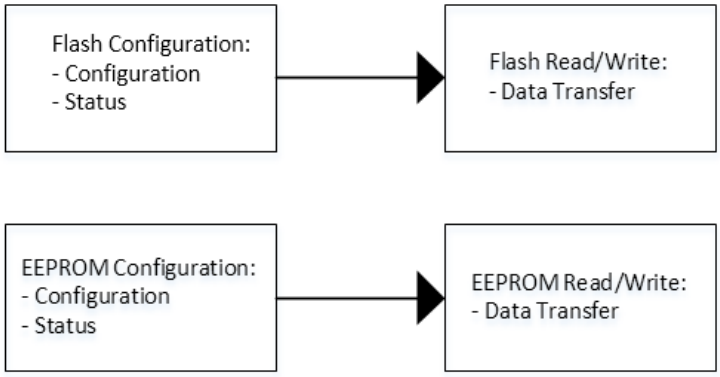
How the Library Works

The usage model for this library is explained in the following sections.

Description

Each of the blocks in the diagram correspond to the library interface section.

Usage Model



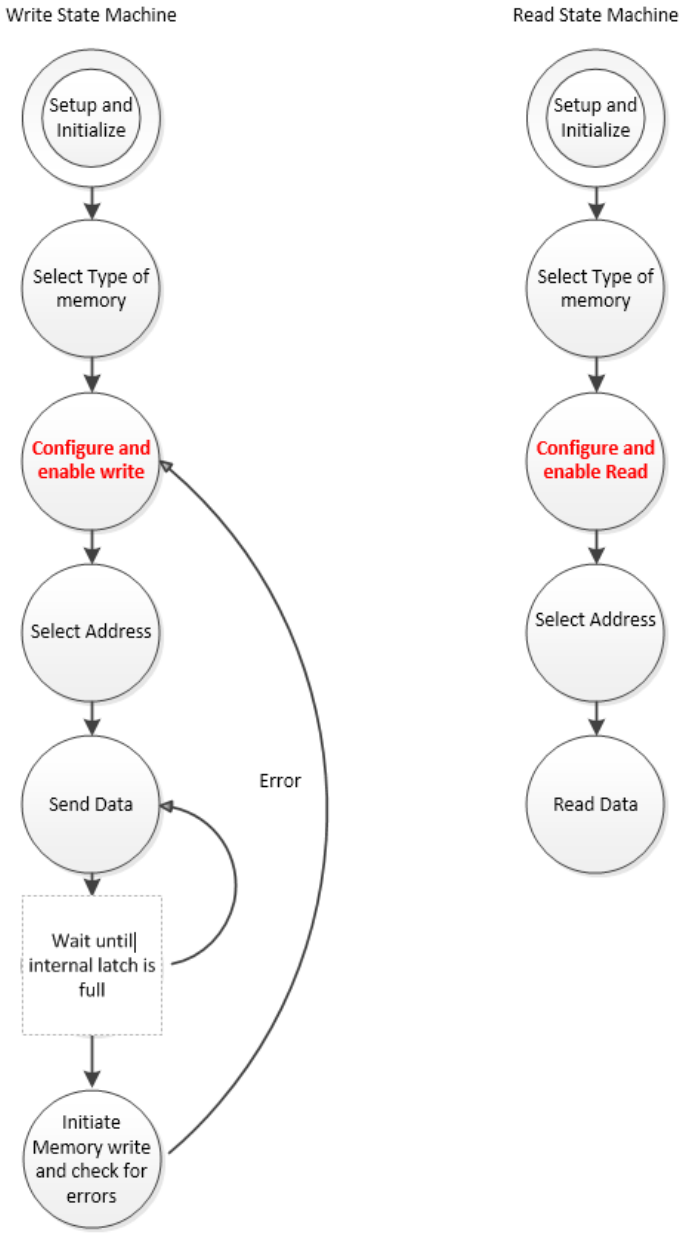
State Machine

This topic describes the NVM Write and Read state machine.

Description

The following state machine is for the NVM Read and Write during the normal operation. This state machine is provided to give a general idea of the usage model of the peripheral library. Refer to the usage model for more detailed steps for the scenario that is being used.

NVM Write and Read State Machine



State	Associated Function
Setup and Initialization	Refer to mode of NVM for detailed instructions of setup.
Select Type of memory	Once the NVM has been appropriately set up and initialized, the state machine waits for the application to select the type of memory to be accessed, either Memory region (Flash/EEPROM) or the corresponding configuration registers.
Configure and enable write/read	Configures the registers and enables the write/read operation.
Select Address	The application provides the write/read address from which to access memory.
Send Data/Read Data	When the application is polling it can fill data/read data into the internal holding registers and initiate write via PLIB_NVM_FlashWriteStart(NVM_ID_0) or PLIB_NVM_EEPROMWriteStart(NVM_ID_0).
Read Error/Write Status	The error/write status is available to the application through the PLIB_NVM_FlashWriteCycleHasCompleted(NVM_ID_0) or PLIB_NVM_EEPROMOperationHasCompleted(NVM_ID_0) functions and other status/error APIs.

Note: Refer to the mode used for the NVM for the setup and initialization steps.

Flash Operations

This topic describes Flash operations.

Description

Note: Flash program memory operations vary across microcontrollers. Please refer to the **"Flash Controller"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by the device in use.

Flash Read Operation Setup

Do the following set up Flash read operation:

1. Provide the Flash program memory from which the data memory has to be read via the API PLIB_NVM_FlashRead function. This address is then updated in the respective address registers or the Table Pointers depending on the device.
2. Ensure the alignment of the address as specified by the specific device data sheet.
3. The size of the data read depends upon the microcontroller in use. For details, please refer to the specific device data sheet.

Example: Flash Program Memory Read

```
// Determine the address from which to read
#define MY_FLASH_BASE_ADDRESS      0xA0007862 // Or any address in Flash to read

uint32_t    address = MY_FLASH_BASE_ADDRESS;
// The Data Available at the memory location
size_t    DataToBeRead;

// where, MY_NVM_INSTANCE - is a specific instance of the hardware peripheral.
// where, address         - The address in the memory from which to read.
DataToBeRead = PLIB_NVM_FlashRead(NVM_ID_0, address);
```

EEPROM Operations

This topic describes EEPROM operations.

Description

Note: EEPROM operations vary across microcontrollers. Please refer to the **"Data EEPROM"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by the device in use.

Before accessing the EEPROM at full speed, it is necessary to program configuration values into the Data EEPROM controller after enabling it. Refer to **"EEPROM Configuration Write Operation Setup"** in Flash Operations for more information.

EEPROM Read Operation Setup

Prior to reading the Data EEPROM, the Data EEPROM must be enabled and ready. There can be no ongoing command when a new command is issued to the Data EEPROM.

Do the following to execute a Data EEPROM read operation:

1. Load the Data EEPROM address to be read. The address must be on a 32-bit boundary and the last two bits must be '00'.
2. Select the read operation command.
3. Enable for read access.
4. Start the read operation.
5. Wait until the read cycle is complete.
6. Read the data from the intermediate register.

Example: EEPROM Read

```
if (PLIB_NVM_EEPROMIsReady( NVM_ID_0 ))
{
    // There should be no ongoing operation.
    if (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ))
    {
        // Set address on 32-bit boundary, last two bits must be '00'
        uint32_t ADDR = 0x0100;
        // Load Data EEPROM read command
        PLIB_NVM_EEPROMOperationSelect( NVM_ID_0 , EEPROM_WORD_READ_OPERATION);
        // Load address and check if it is valid
        if(true ==PLIB_NVM_EEPROMAddress( NVM_ID_0 , ADDR ))
        {
```

```
                // Enable access for read
                PLIB_NVM_EEPROMReadEnable( NVM_ID_0 );
                // Start the read operation
                PLIB_NVM_EEPROMReadStart( NVM_MODULE_ID index );
                // Wait until read is complete
                while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
                // Read the data
                *ee_data = PLIB_NVM_EEPROMRead( NVM_ID_0 );
            }
        }
    }
```

EEPROM Write Operation Setup

Prior to writing to the Data EEPROM, the Data EEPROM must be enabled and ready. There can be no ongoing command when a new command is issued to the Data EEPROM.

Do the following to execute a Data EEPROM write operation:

1. Load the Data EEPROM address where data has to be stored. The address must be on a 32-bit boundary so that the last two bits must be '00'.
2. Select the write operation command.
3. Enable for write access.
4. Write data into the intermediate register.
5. Execute the Data EEPROM unlock sequence.
6. Start the write operation.
7. Wait until the write cycle is complete.

Example: EEPROM Write

```
    if (PLIB_NVM_EEPROMIsReady( NVM_ID_0 ))
    {
        // There should be no ongoing operation.
        if (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ))
        {
            // Set address on 32-bit boundary, last two bits must be '00'
            uint32_t ADDR = 0x0100;
            //Keys to unlock EEPROM
            uint32_t key1 = 0xEDB7;
            uint32_t key2 = 0x1248;
            // Load Data EEPROM write command
            PLIB_NVM_EEPROMOperationSelect( NVM_ID_0 , EEPROM_WORD_WRITE_OPERATION);
            // Load address and check if it is valid
            if(true ==PLIB_NVM_EEPROMAddress( NVM_ID_0 , ADDR ))
            {
                // Enable access to write
                PLIB_NVM_EEPROMWriteEnable( NVM_ID_0 );
                // Load data to intermediate register
                PLIB_NVM_EEPROMDataToWrite ( NVM_ID_0 , ee_data ) ;
                // Unlock the EEPROM
                PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key1);
                PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key2);
                // Start the write operation
                PLIB_NVM_EEPROMWriteStart( NVM_MODULE_ID index );
                // Wait until write is complete
                while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
            }
        }
    }
```

EEPROM Forced Word Erase Operation Setup

Under normal conditions, there is no need to attempt a Forced Word Erase. The Data EEPROM has internal logic which automatically manages all read, erase, and write command sequences.

If a verification error occurs during a write to the Data EEPROM, the user can attempt a Forced Word Erase operation to recover the Data EEPROM word storage location.

The Word Erase operation is similar to the write operation.

Example: EEPROM Word Erase

```
    if (PLIB_NVM_EEPROMIsReady( NVM_ID_0 ))
    {
       // There should be no ongoing operation.
        if (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ))
```

```
    {
        // Set address on 32-bit boundary, last two bits must be '00'
        uint32_t ADDR = 0x0100;
        //Keys to unlock EEPROM
        uint32_t key1 = 0xEDB7;
        uint32_t key2 = 0x1248;
        // Load Data EEPROM forced word erase command
        PLIB_NVM_EEPROMOperationSelect( NVM_ID_0 , EEPROM_FORCED_WORD_ERASE_OPERATION);
        // Load address and check if it is valid
        if(true ==PLIB_NVM_EEPROMAddress( NVM_ID_0 , ADDR ))
        {
            // Enable access to write
            PLIB_NVM_EEPROMWriteEnable( NVM_ID_0 );
            // Unlock the EEPROM
            PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key1);
            PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key2);
            // Start the write operation
            PLIB_NVM_EEPROMEraseStart( NVM_MODULE_ID index );
            // Wait until write is complete
            while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
    }
}
```

EEPROM Bulk Erase Operation Setup

Prior to erasing the entire Data EEPROM, the Data EEPROM must be enabled and ready. There can be no ongoing command when a new command is issued to the Data EEPROM.

The Bulk Erase operation is similar to the write operation except the EEPROM operation is selected for Bulk erase and the data or address does not need to be loaded.

Example: EEPROM Bulk Erase

```
    if (PLIB_NVM_EEPROMIsReady( NVM_ID_0 ))
    {
        // There should be no ongoing operation.
        if (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ))
        {
            //Keys to unlock EEPROM
            uint32_t key1 = 0xEDB7;
            uint32_t key2 = 0x1248;
            // Load Data EEPROM forced word erase command
            PLIB_NVM_EEPROMOperationSelect( NVM_ID_0 , EEPROM_ERASE_ALL_OPERATION);
            // Enable access to write
            PLIB_NVM_EEPROMWriteEnable( NVM_ID_0 );
            // Unlock the EEPROM
            PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key1);
            PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key2);
            // Start the write operation
            PLIB_NVM_EEPROMEraseStart( NVM_MODULE_ID index );
            // Wait until write is complete
            while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
        }
    }
```

EEPROM Configuration Write Operation Setup

Before accessing the EEPROM at full speed, it is necessary to program configuration values into the Data EEPROM controller after enabling it. This is done through the Configuration Write operation. The configuration values to be written to the Data EEPROM Controller are stored in the DEVEE1 through DEVEE8 registers. Eight consecutive word write cycles should be performed with the EEPROM Configuration Write mode operation selected.

Miscellaneous Functions

This topic describes miscellaneous features.

Description

Note: Features vary across microcontrollers. Please refer to the **"Flash Controller"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by the device in use.

This section will detail out a few niche and processor-specific APIs provided by this library.

Stop in Idle Mode

The PLIB_NVM_EEPROMStopInIdleEnable and PLIB_NVM_EEPROMStopInIdleDisable functions cater to this functionality. These functions ensure that EEPROM operations are discontinued or continued in Idle mode.

Low Voltage Detection

The PLIB_NVM_LowVoltageIsDetected and PLIB_NVM_LowVoltageEventIsActive functions support low-voltage error detection and triggering of a Flash memory low-voltage event.

Long Write/Erase Cycle Detection

The PLIB_NVM_EEPROMNextWriteCycleIsLong function allows the application to know whether the next EEPROM Write/Erase cycle is long.

Configuring the Library

The library is configured for the supported NVM module when the processor is chosen in the MPLAB X IDE.

Building the Library

This section list the files that are available in the `src` folder of the NVM driver. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

Library Interface

a) Flash Memory Configuration and Status Functions

	Name	Description
	PLIB_NVM_LowVoltageEventIsActive	Provides low voltage detection status.
	PLIB_NVM_LowVoltageIsDetected	Provides low voltage error detection status.
	PLIB_NVM_MemoryModifyEnable	Allows write cycles to Flash memory.
	PLIB_NVM_MemoryModifyInhibit	Inhibits write cycles to Flash memory.
	PLIB_NVM_MemoryOperationSelect	Selects the operation to be performed on Flash memory.

b) Flash Memory Operation Functions

	Name	Description
	PLIB_NVM_DataBlockSourceAddress	Takes the address parameter in the argument and loads the base address from which data has to be copied into Flash memory.
	PLIB_NVM_FlashAddressToModify	Modifies the Flash memory address.
	PLIB_NVM_FlashEraseStart	Performs erase operation on the selected Flash memory area.
	PLIB_NVM_FlashProvideData	Provides the data to be written into Flash memory.
	PLIB_NVM_FlashProvideQuadData	Provides the quad data to be written into Flash memory.
	PLIB_NVM_FlashRead	Read the specified address of Flash memory.
	PLIB_NVM_FlashWriteKeySequence	Copies the mandatory KEY sequence into the respective registers.
	PLIB_NVM_FlashWriteStart	Performs a write operation on the Flash memory row selected.
	PLIB_NVM_FlashWriteCycleHasCompleted	Provides the status of the Flash write cycle.
	PLIB_NVM_WriteOperationHasTerminated	Provides the status of the Flash write operation or sequence.

c) Other Flash Memory Functions

	Name	Description
	PLIB_NVM_FlashSwapLockSelect	Selects the kind of Flash swap lock required.
	PLIB_NVM_FlashSwapLockStatusGet	Get the status of Swap lock bits.
	PLIB_NVM_FlashWriteProtectMemoryAreaRange	Sets the address below which physical memory will be write protected.
	PLIB_NVM_BootPageWriteProtectionDisable	Write protection for selected boot page is disabled.
	PLIB_NVM_BootPageWriteProtectionEnable	Locks the selected boot page.
	PLIB_NVM_IsBootMemoryLocked	Provides lock status of boot page write-protect bits.
	PLIB_NVM_IsBootPageWriteProtected	Provides write protection status for boot memory page.
	PLIB_NVM_IsProgramFlashMemoryLocked	Provides lock status of Program Flash Write-Protect register.
	PLIB_NVM_LockBootMemory	Locks the boot write-protect bits.
	PLIB_NVM_LockProgramFlashMemory	Lock the Program Flash write-protect register.
	PLIB_NVM_ProgramFlashBank1LowerRegion	Maps Flash Bank 1 to the lower mapped region.
	PLIB_NVM_ProgramFlashBank2LowerRegion	Maps the bank 2 to lower mapped region.

	PLIB_NVM_ProgramFlashBank2IsLowerRegion	Gives the status of Program Flash Bank mapping.
	PLIB_NVM_BootFlashBank1LowerRegion	Maps Boot Flash Bank 1 to lower mapped region.
	PLIB_NVM_BootFlashBank2IsLowerRegion	Gives the status of Boot Flash Bank mapping.
	PLIB_NVM_BootFlashBank2LowerRegion	Maps Boot Flash Bank 2 to the lower mapped region.

d) EEPROM Configuration and Status Functions

	Name	Description
	PLIB_NVM_EEPROMEnable	Enables the EEPROM memory.
	PLIB_NVM_EEPROMDisable	Disables the EEPROM memory.
	PLIB_NVM_EEPROMIsReady	Provides the availability status of the EEPROM.
	PLIB_NVM_EEPROMStopInIdleEnable	Discontinues EEPROM operation when device enters Idle mode.
	PLIB_NVM_EEPROMStopInIdleDisable	Continues EEPROM operation when device enters Idle mode.
	PLIB_NVM_EEPROMStopInIdleIsEnabled	Returns Stop in Idle mode status of the EEPROM operation.
	PLIB_NVM_EEPROMOperationSelect	Selects the operation to be performed on the EEPROM.
	PLIB_NVM_EEPROMWriteEnable	Allows write or erase operation to the EEPROM.
	PLIB_NVM_EEPROMReadEnable	Allows read operation to the EEPROM.
	PLIB_NVM_EEPROMWriteIsEnabled	Returns EEPROM Write permission status.
	PLIB_NVM_EEPROMReadIsEnabled	Returns EEPROM read permission status.
	PLIB_NVM_EEPROMNextWriteCycleIsLong	Informs whether the next write or erase cycle of the EEPROM is long.
	PLIB_NVM_EEPROMErrorGet	Returns the EEPROM operation error.
	PLIB_NVM_EEPROMErrorClear	Clears the EEPROM operation error.

e) EEPROM Operation Functions

	Name	Description
	PLIB_NVM_EEPROMAddress	EEPROM address where operation has to be performed.
	PLIB_NVM_EEPROMDataToWrite	Accepts the data to be written into the EEPROM.
	PLIB_NVM_EEPROMKeySequenceWrite	Write mandatory KEY sequence to unlock the EEPROM write or erase protection.
	PLIB_NVM_EEPROMEraseStart	Initiates EEPROM erase cycle.
	PLIB_NVM_EEPROMWriteStart	Initiates a EEPROM write cycle.
	PLIB_NVM_EEPROMReadStart	Initiates a EEPROM read cycle.
	PLIB_NVM_EEPROMOperationHasCompleted	Provides the status of the EEPROM write or erase or read cycle.
	PLIB_NVM_EEPROMRead	Read the EEPROM data.
	PLIB_NVM_EEPROMOperationAbort	Aborts the current EEPROM operation.

f) Feature Existence Functions

	Name	Description
	PLIB_NVM_ExistsAddressModifyControl	Identifies whether the AddressModifyControl feature exists on the NVM module.
	PLIB_NVM_ExistsBootPageWriteProtect	Identifies whether the BootPageWriteProtect feature exists on the NVM module.
	PLIB_NVM_ExistsFlashBankRegionSelect	Identifies whether the FlashBankRegionSelect feature exists on the NVM module.
	PLIB_NVM_ExistsFlashWPMemoryRangeProvide	Identifies whether the FlashWPMemoryRangeProvide feature exists on the NVM module.
	PLIB_NVM_ExistsKeySequence	Identifies whether the KeySequence feature exists on the NVM module.
	PLIB_NVM_ExistsLockBootSelect	Identifies whether the LockBootSelect feature exists on the NVM module.
	PLIB_NVM_ExistsLockPFMSelect	Identifies whether the LockPFMSelect feature exists on the NVM module.
	PLIB_NVM_ExistsLowVoltageError	Identifies whether the LowVoltageError feature exists on the NVM module.
	PLIB_NVM_ExistsLowVoltageStatus	Identifies whether the LowVoltageStatus feature exists on the NVM module.
	PLIB_NVM_ExistsMemoryModificationControl	Identifies whether the MemoryModificationControl feature exists on the NVM module.
	PLIB_NVM_ExistsOperationMode	Identifies whether the OperationMode feature exists on the NVM module.
	PLIB_NVM_ExistsProvideData	Identifies whether the ProvideData feature exists on the NVM module.
	PLIB_NVM_ExistsProvideQuadData	Identifies whether the ProvideQuadData feature exists on the NVM module.
	PLIB_NVM_ExistsSourceAddress	Identifies whether the SourceAddress feature exists on the NVM module.
	PLIB_NVM_ExistsWriteErrorStatus	Identifies whether the WriteErrorStatus feature exists on the NVM module.
	PLIB_NVM_ExistsWriteOperation	Identifies whether the WriteOperation feature exists on the NVM module.

	PLIB_NVM_ExistsBootFlashBankRegion	Identifies whether the BootFlashBankRegion feature exists on the NVM module.
	PLIB_NVM_ExistsSwapLockControl	Identifies whether the SwapLockControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMAddressControl	Identifies whether the EEPROMAddressControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMDataControl	Identifies whether the EEPROMDataControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMEnableControl	Identifies whether the EEPROMEnableControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMEnableOperationControl	Identifies whether the EEPROMEnableOperationControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMErrorStatus	Identifies whether the EEPROMErrorStatus feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMKeySequence	Identifies whether the EEPROMKeySequence feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMLongWriteStatus	Identifies whether the EEPROMLongWriteStatus feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMOperationAbortControl	Identifies whether the EEPROMOperationAbortControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMOperationModeControl	Identifies whether the EEPROMOperationModeControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMStartOperationControl	Identifies whether the EEPROMStartOperationControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMStopInIdleControl	Identifies whether the EEPROMStopInIdleControl feature exists on the NVM module.

g) Data Types and Constants

	Name	Description
	NVM_BOOT_MEMORY_AREA	Lists the different possible boot memory region.
	NVM_BOOT_MEMORY_PAGE	Lists the different NVM boot memory pages.
	NVM_OPERATION_MODE	Lists the different Flash operation modes.
	NVM_MODULE_ID	Possible instances of the NVM module.
	NVM_FLASH_SWAP_LOCK_TYPE	Lists the possible type of Flash swap lock.
	EEPROM_ERROR	Lists the different EEPROM operation errors.
	EEPROM_OPERATION_MODE	Lists the different EEPROM operation modes.

Description

This section describes the Application Programming Interface (API) functions of the NVM Peripheral Library.

Refer to each section for a detailed description.

a) Flash Memory Configuration and Status Functions

PLIB_NVM_LowVoltageEventIsActive Function

Provides low voltage detection status.

File

plib_nvm.h

C

```
bool PLIB_NVM_LowVoltageEventIsActive(NVM_MODULE_ID index);
```

Returns

- 1 - Low Voltage Event is active
- 0 - Low Voltage Event is not active

Description

This function provides detection of low voltage event, if any.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsLowVoltageStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_NVM_LowVoltageEventIsActive(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_LowVoltageEventIsActive(NVM_MODULE_ID index)

PLIB_NVM_LowVoltageIsDetected Function

Provides low voltage error detection status.

File

plib_nvm.h

C

```
bool PLIB_NVM_LowVoltageIsDetected(NVM_MODULE_ID index);
```

Returns

- 1 - Low Voltage detection and possible data corruption
- 0 - Voltage Level Acceptable for programming

Description

This function provides detection of low voltage error status.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsLowVoltageStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_NVM_LowVoltageIsDetected(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_LowVoltageIsDetected(NVM_MODULE_ID index)

PLIB_NVM_MemoryModifyEnable Function

Allows write cycles to Flash memory.

File

plib_nvm.h

C

```
void PLIB_NVM_MemoryModifyEnable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function allows changing the write control (WR) bit and disables writing into the SWAP and NVMOP bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsMemoryModificationControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_MemoryModifyEnable(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_MemoryModifyEnable(NVM_MODULE_ID index)

PLIB_NVM_MemoryModifyInhibit Function

Inhibits write cycles to Flash memory.

File

plib_nvm.h

C

```
void PLIB_NVM_MemoryModifyInhibit(NVM_MODULE_ID index);
```

Returns

None.

Description

This function disables the writing in the write control (WR) bit and enables writing into the SWAP and NVMOP bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsMemoryModificationControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_MemoryModifyInhibit(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_MemoryModifyInhibit(NVM_MODULE_ID index)

PLIB_NVM_MemoryOperationSelect Function

Selects the operation to be performed on Flash memory.

File

plib_nvm.h

C

```
void PLIB_NVM_MemoryOperationSelect(NVM_MODULE_ID index, NVM_OPERATION_MODE operationmode);
```

Returns

None.

Description

This function selects the operation to be performed on Flash memory.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsOperationMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_MemoryOperationSelect(MY_NVM_INSTANCE, NVM_MEMORY_ROW_PROGRAM_OPERATION);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_MemoryOperationSelect(NVM_MODULE_ID index,
NVM_OPERATION_TYPE_SELECT operationmode)

b) Flash Memory Operation Functions

PLIB_NVM_DataBlockSourceAddress Function

Takes the address parameter in the argument and loads the base address from which data has to be copied into Flash memory.

File

plib_nvm.h

C

```
void PLIB_NVM_DataBlockSourceAddress(NVM_MODULE_ID index, uint32_t address);
```

Returns

None.

Description

This function takes the address parameter in the argument and loads the base address from which data has to be copied into Flash memory. This is to copy a row of data directly into Flash in one iteration without handling any intermediate holding registers.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsSourceAddress in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t    address = MY_RAM_BASE_ADDRESS;
PLIB_NVM_DataBlockSourceAddress(MY_NVM_INSTANCE, address);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	The starting address in the user data memory from which data will be written

Function

void PLIB_NVM_DataBlockSourceAddress(NVM_MODULE_ID index, uint32_t address)

PLIB_NVM_FlashAddressToModify Function

Modifies the Flash memory address.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashAddressToModify(NVM_MODULE_ID index, uint32_t address);
```

Returns

None.

Description

This function takes the address parameter in the argument and loads the address that will be modified by the actual write operation. The write or erase operation following this will write or erase the user data into or from the Flash program memory.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsAddressModifyControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t    address = MY_FLASH_BASE_ADDRESS;
PLIB_NVM_FlashAddressToModify(MY_NVM_INSTANCE, address);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	The starting address in the memory from which data will be written. This address should be a physical address.

Function

void PLIB_NVM_FlashAddressToModify(NVM_MODULE_ID index, uint32_t address)

PLIB_NVM_FlashEraseStart Function

Performs erase operation on the selected Flash memory area.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashEraseStart(NVM_MODULE_ID index);
```

Returns

None.

Description

This function Performs erase operation on the selected Flash memory region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsWriteOperation in your application to determine whether this feature is available.

Preconditions

- The Address of the page to be erased must be provided using PLIB_NVM_FlashAddressToModify
- Erase Operation should be selected using PLIB_NVM_MemoryOperationSelect
- The module should be configured to access Flash memory using PLIB_NVM_MemoryModifyEnable
- The unlock key sequence should be provided using PLIB_NVM_FlashWriteKeySequence

Example

```
PLIB_NVM_FlashEraseStart(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_FlashEraseStart(NVM_MODULE_ID index)

PLIB_NVM_FlashProvideData Function

Provides the data to be written into Flash memory.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashProvideData(NVM_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function provides the user data to intermediate registers before being written into Flash memory.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsProvideData in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t DataToWrite;
PLIB_NVM_FlashProvideData(MY_NVM_INSTANCE, DataToWrite);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to be written

Function

void PLIB_NVM_FlashProvideData(NVM_MODULE_ID index, uint32_t data)

PLIB_NVM_FlashProvideQuadData Function

Provides the quad data to be written into Flash memory.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashProvideQuadData(NVM_MODULE_ID index, uint32_t * data);
```

Returns

None.

Description

This function provides the user quad data to intermediate registers before being written into Flash memory.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsProvideQuadData in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t  DataToWrite[4];
PLIB_NVM_FlashProvideQuadData(MY_NVM_INSTANCE, &DataToWrite);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Address pointing to the data to be written.

Function

void PLIB_NVM_FlashProvideQuadData(NVM_MODULE_ID index, uint32_t *data);

PLIB_NVM_FlashRead Function

Read the specified address of Flash memory.

File

plib_nvm.h

C

```
uint32_t PLIB_NVM_FlashRead(NVM_MODULE_ID index, uint32_t address);
```

Returns

Data value read at the memory address.

Description

This function takes the address parameter in the argument and loads the read address to the appropriate register. The read operation provides data from the given address.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsWriteOperation in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t    DataToBeRead;
```

```
DataToBeRead = PLIB_NVM_FlashRead(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	The address in the memory from which to read

Function

uint32_t PLIB_NVM_FlashRead(NVM_MODULE_ID index, uint32_t address)

PLIB_NVM_FlashWriteKeySequence Function

Copies the mandatory KEY sequence into the respective registers.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashWriteKeySequence(NVM_MODULE_ID index, uint32_t keysequence);
```

Returns

None.

Description

This function copies the mandatory KEY sequence into the respective registers.

Remarks

Without the KEY sequence write or erase operation will not be executed.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsKeySequence in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_FlashWriteKeySequence(MY_NVM_INSTANCE, keysequence);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
keysequence	Mandatory KEY sequence depending on the controller type

Function

void PLIB_NVM_FlashWriteKeySequence(NVM_MODULE_ID index, uint32_t keysequence)

PLIB_NVM_FlashWriteStart Function

Performs a write operation on the Flash memory row selected.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashWriteStart(NVM_MODULE_ID index);
```

Returns

None.

Description

This function performs a write operation on the Flash memory row selected.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsWriteOperation in your application to determine whether this feature is available.

Preconditions

- The Address of the page to be written must be provided using PLIB_NVM_FlashAddressToModify
- Erase Operation should be selected using PLIB_NVM_MemoryOperationSelect
- The module should be configured to access Flash memory using PLIB_NVM_MemoryModifyEnable
- The unlock key sequence should be provided using PLIB_NVM_FlashWriteKeySequence

Example

```
PLIB_NVM_FlashWriteStart(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_FlashWriteStart(NVM_MODULE_ID index)

PLIB_NVM_FlashWriteCycleHasCompleted Function

Provides the status of the Flash write cycle.

File

plib_nvm.h

C

```
bool PLIB_NVM_FlashWriteCycleHasCompleted(NVM_MODULE_ID index);
```

Returns

- 0 - Write or erase cycle is incomplete
- 1 - Write or erase cycle has completed

Description

This function provides the status of the Flash write cycle that was initiated by a write or erase operation.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsWriteOperation in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_NVM_FlashWriteCycleHasCompleted(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_FlashWriteCycleHasCompleted(NVM_MODULE_ID index)

PLIB_NVM_WriteOperationHasTerminated Function

Provides the status of the Flash write operation or sequence.

File

plib_nvm.h

C

```
bool PLIB_NVM_WriteOperationHasTerminated(NVM_MODULE_ID index);
```

Returns

- 1 - Write operation prematurely terminated
- 0 - Write operation completed

Description

This function provides the status of the Flash write operation or sequence that was initiated by a write or erase operation.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsWriteErrorStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_NVM_WriteOperationHasTerminated(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_WriteOperationHasTerminated(NVM_MODULE_ID index)

c) Other Flash Memory Functions

PLIB_NVM_FlashSwapLockSelect Function

Selects the kind of Flash swap lock required.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashSwapLockSelect(NVM_MODULE_ID index, NVM_FLASH_SWAP_LOCK_TYPE lockType);
```

Returns

None.

Description

This function allows user to select which swap bits will be writable.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsSwapLockControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_FlashSwapLockSelect(MY_NVM_INSTANCE, NVM_FLASH_SWAP_UNLOCKED);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
lockType	One of the value from NVM_FLASH_SWAP_LOCK_TYPE enum to identify which swap bit will be locked

Function

void PLIB_NVM_FlashSwapLockSelect
(
NVM_MODULE_ID index,
NVM_FLASH_SWAP_LOCK_TYPE lockType
)

PLIB_NVM_FlashSwapLockStatusGet Function

Get the status of Swap lock bits.

File

plib_nvm.h

C

```
NVM_FLASH_SWAP_LOCK_TYPE PLIB_NVM_FlashSwapLockStatusGet(NVM_MODULE_ID index);
```

Returns

- NVM_FLASH_SWAP_LOCK_TYPE - The lock status of Flash swap bits.

Description

This function allows user to get the status of swap lock bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsSwapLockControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
NVM_FLASH_SWAP_LOCK_TYPE lockStatus;
lockStatus = PLIB_NVM_FlashSwapLockStatusGet(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured.

Function

NVM_FLASH_SWAP_LOCK_TYPE PLIB_NVM_FlashSwapLockStatusGet
(
NVM_MODULE_ID index
)

PLIB_NVM_FlashWriteProtectMemoryAreaRange Function

Sets the address below which physical memory will be write protected.

File

plib_nvm.h

C

```
void PLIB_NVM_FlashWriteProtectMemoryAreaRange(NVM_MODULE_ID index, uint32_t address);
```

Returns

None.

Description

This function sets the address below which physical memory will be protected. Physical memory below address 0x1Dxxxxxx is write protected, where 'xxxxxx' is specified by "address" parameter. When "address" has a value of '0', write protection is disabled for the entire program Flash. If the specified address falls within the page, the entire page and all pages below the current page will be protected.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsFlashWPMemoryRangeProvide in your application to determine whether this feature is available.

Preconditions

The unlock key sequence should be provided using PLIB_NVM_FlashWriteKeySequence.

Example

```
PLIB_NVM_FlashWriteProtectMemoryAreaRange(MY_NVM_INSTANCE, WRITE_PROTECT_PAGE_ADDRESS);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	Flash program write-protect (Page) address

Function

void PLIB_NVM_FlashWriteProtectMemoryAreaRange(NVM_MODULE_ID index, uint32_t address);

PLIB_NVM_BootPageWriteProtectionDisable Function

Write protection for selected boot page is disabled.

File

plib_nvm.h

C

```
void PLIB_NVM_BootPageWriteProtectionDisable(NVM_MODULE_ID index, NVM_BOOT_MEMORY_PAGE bootPage);
```

Returns

None.

Description

This function disables Write protection for selected boot page.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsBootPageWriteProtect in your application to determine whether this feature is available.

Preconditions

Unlock key sequence should be provided using API PLIB_NVM_FlashWriteKeySequence.

Example

```
PLIB_NVM_BootPageWriteProtectionDisable(MY_NVM_INSTANCE, LOWER_BOOT_ALIAS_PAGE4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bootPage	Selects the boot page for which write protection has to be disabled

Function

void PLIB_NVM_BootPageWriteProtectionDisable(NVM_MODULE_ID index, NVM_BOOT_MEMORY_PAGE bootPage);

PLIB_NVM_BootPageWriteProtectionEnable Function

Locks the selected boot page.

File

plib_nvm.h

C

```
void PLIB_NVM_BootPageWriteProtectionEnable(NVM_MODULE_ID index, NVM_BOOT_MEMORY_PAGE bootPage);
```

Returns

None.

Description

This function locks the selected boot page.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsBootPageWriteProtect in your application to determine whether this feature is available.

Preconditions

Unlock key sequence should be provided using API PLIB_NVM_FlashWriteKeySequence.

Example

```
PLIB_NVM_BootPageWriteProtectionEnable(MY_NVM_INSTANCE, LOWER_BOOT_ALIAS_PAGE4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bootPage	Selects the boot page which has to be locked

Function

void PLIB_NVM_BootPageWriteProtectionEnable(NVM_MODULE_ID index, NVM_BOOT_MEMORY_PAGE bootPage);

PLIB_NVM_IsBootMemoryLocked Function

Provides lock status of boot page write-protect bits.

File

plib_nvm.h

C

```
bool PLIB_NVM_IsBootMemoryLocked(NVM_MODULE_ID index, NVM_BOOT_MEMORY_AREA memoryArea);
```

Returns

- 1 - Selected boot alias write-protect bits are locked
- 0 - Selected boot alias write-protect bits are not locked

Description

This function provides lock status of boot page write-protect bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsLockBootSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_NVM_IsBootMemoryLocked(MY_NVM_INSTANCE, LOWER_BOOT_ALIAS_AREA);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
memoryArea	Selects between the Lower or Upper Boot Alias area

Function

bool PLIB_NVM_IsBootMemoryLocked(NVM_MODULE_ID index, NVM_BOOT_MEMORY_AREA memoryArea);

PLIB_NVM_IsBootPageWriteProtected Function

Provides write protection status for boot memory page.

File

plib_nvm.h

C

```
bool PLIB_NVM_IsBootPageWriteProtected(NVM_MODULE_ID index, NVM_BOOT_MEMORY_PAGE bootPage);
```

Returns

- 1 - Selected boot region is write protected
- 0 - Selected boot region is not write protected

Description

This function provides write protection status for selected boot memory page.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsBootPageWriteProtect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_NVM_IsBootPageWriteProtected(MY_NVM_INSTANCE, LOWER_BOOT_ALIAS_PAGE4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bootPage	Selects the boot page region

Function

bool PLIB_NVM_IsBootPageWriteProtected(NVM_MODULE_ID index, NVM_BOOT_MEMORY_PAGE bootPage);

PLIB_NVM_IsProgramFlashMemoryLocked Function

Provides lock status of Program Flash Write-Protect register.

File

plib_nvm.h

C

```
bool PLIB_NVM_IsProgramFlashMemoryLocked(NVM_MODULE_ID index);
```

Returns

- 1 - Program Flash write-protect register is locked
- 0 - Program Flash write-protect register is not locked

Description

This function provides lock status of Program Flash Write-Protect (NVMPWP) register.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsLockPFMSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
    status = PLIB_NVM_IsProgramFlashMemoryLocked(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_IsProgramFlashMemoryLocked(NVM_MODULE_ID index);

PLIB_NVM_LockBootMemory Function

Locks the boot write-protect bits.

File

plib_nvm.h

C

```
void PLIB_NVM_LockBootMemory(NVM_MODULE_ID index, NVM_BOOT_MEMORY_AREA memoryArea);
```

Returns

None.

Description

This function locks the given (lower or upper alias) boot write-protect bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsLockBootSelect in your application to determine whether this feature is available.

Preconditions

The unlock key sequence should be provided using PLIB_NVM_FlashWriteKeySequence.

Example

```
PLIB_NVM_LockBootMemory(MY_NVM_INSTANCE, LOWER_BOOT_ALIAS_AREA);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
memoryArea	Selects between Lower or Upper Boot Alias area

Function

void PLIB_NVM_LockBootMemory(NVM_MODULE_ID index, NVM_BOOT_MEMORY_AREA memoryArea);

PLIB_NVM_LockProgramFlashMemory Function

Lock the Program Flash write-protect register.

File

plib_nvm.h

C

```
void PLIB_NVM_LockProgramFlashMemory(NVM_MODULE_ID index);
```

Returns

None.

Description

This function locks the Program Flash Write-Protect register (NVMPWP).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsLockPFMSelect in your application to determine whether this feature is available.

Preconditions

Unlock key sequence should be provided using API PLIB_NVM_FlashWriteKeySequence.

Example

```
PLIB_NVM_LockProgramFlashMemory(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_LockProgramFlashMemory(NVM_MODULE_ID index);

PLIB_NVM_ProgramFlashBank1LowerRegion Function

Maps Flash Bank 1 to the lower mapped region.

File

plib_nvm.h

C

```
void PLIB_NVM_ProgramFlashBank1LowerRegion(NVM_MODULE_ID index);
```

Returns

None.

Description

This function maps Program Flash Bank 1 to the lower mapped region and programs Flash Bank 2 to the upper mapped region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsFlashBankRegionSelect in your application to determine whether this feature is available.

Preconditions

The WREN bit in the NVMCON register should be set to '0' using PLIB_NVM_MemoryModifyInhibit before swapping the memory regions.

Example

```
PLIB_NVM_ProgramFlashBank1LowerRegion(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_ProgramFlashBank1LowerRegion(NVM_MODULE_ID index);

PLIB_NVM_ProgramFlashBank2LowerRegion Function

Maps the bank 2 to lower mapped region.

File

plib_nvm.h

C

```
void PLIB_NVM_ProgramFlashBank2LowerRegion(NVM_MODULE_ID index);
```

Returns

None.

Description

This function maps Program Flash Bank 2 to the lower mapped region and Program Flash Bank 1 to the upper mapped region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsFlashBankRegionSelect in your application to determine whether this feature is available.

Preconditions

The WREN bit in the NVMCON register should be set to '0' using PLIB_NVM_MemoryModifyInhibit before swapping the memory regions.

Example

```
PLIB_NVM_ProgramFlashBank2LowerRegion(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_ProgramFlashBank2LowerRegion(NVM_MODULE_ID index);

PLIB_NVM_ProgramFlashBank2IsLowerRegion Function

Gives the status of Program Flash Bank mapping.

File

plib_nvm.h

C

```
bool PLIB_NVM_ProgramFlashBank2IsLowerRegion(NVM_MODULE_ID index);
```

Returns

- 1 - Program Flash Bank 2 is mapped to the lower mapped region and Program Flash Bank 1 is mapped to the upper mapped region
- 0 - Program Flash Bank 1 is mapped to the lower mapped region and Program Flash Bank 2 is mapped to the upper mapped region

Description

This function tells which Program Flash Bank is mapped to which region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsFlashBankRegionSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(!PLIB_NVM_ProgramFlashBank2IsLowerRegion(MY_NVM_INSTANCE))
PLIB_NVM_ProgramFlashBank2LowerRegion(NVM_MODULE_ID index);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_ProgramFlashBank2IsLowerRegion(NVM_MODULE_ID index);

PLIB_NVM_BootFlashBank1LowerRegion Function

Maps Boot Flash Bank 1 to lower mapped region.

File

plib_nvm.h

C

```
void PLIB_NVM_BootFlashBank1LowerRegion(NVM_MODULE_ID index);
```

Returns

None.

Description

This function maps Boot Flash Bank 1 to the lower mapped region and Boot Flash Bank 2 to the upper mapped region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsBootFlashBankRegion in your application to determine whether this feature is available.

Preconditions

The WREN bit in the NVMCON register should be set to '0' using PLIB_NVM_MemoryModifyInhibit before swapping the memory regions.

Example

```
PLIB_NVM_BootFlashBank1LowerRegion(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_BootFlashBank1LowerRegion(NVM_MODULE_ID index);

PLIB_NVM_BootFlashBank2IsLowerRegion Function

Gives the status of Boot Flash Bank mapping.

File

plib_nvm.h

C

```
bool PLIB_NVM_BootFlashBank2IsLowerRegion(NVM_MODULE_ID index);
```

Returns

- 1 - Boot Flash Bank 2 is mapped to the lower mapped region and Boot Flash Bank 1 is mapped to the upper mapped region
- 0 - Boot Flash Bank 1 is mapped to the lower mapped region and Boot Flash Bank 2 is mapped to the upper mapped region

Description

This function tells which Boot Flash Bank is mapped to which region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsBootFlashBankRegion in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(!PLIB_NVM_BootFlashBank2IsLowerRegion(MY_NVM_INSTANCE))
PLIB_NVM_BootFlashBank2LowerRegion(NVM_MODULE_ID index);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_BootFlashBank2IsLowerRegion(NVM_MODULE_ID index);

PLIB_NVM_BootFlashBank2LowerRegion Function

Maps Boot Flash Bank 2 to the lower mapped region.

File

plib_nvm.h

C

```
void PLIB_NVM_BootFlashBank2LowerRegion(NVM_MODULE_ID index);
```

Returns

None.

Description

This function maps Boot Flash Bank 2 to the lower mapped region and Boot Flash Bank 1 to the upper mapped region.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsBootFlashBankRegion in your application to determine whether this feature is available.

Preconditions

The WREN bit in the NVMCON register should be set to '0' using PLIB_NVM_MemoryModifyInhibit before swapping the memory regions.

Example

```
PLIB_NVM_BootFlashBank2LowerRegion(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_BootFlashBank2LowerRegion(NVM_MODULE_ID index);

d) EEPROM Configuration and Status Functions

PLIB_NVM_EEPROMEnable Function

Enables the EEPROM memory.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMEnable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function enables the EEPROM memory for access.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_EEPROMEnable(MY_NVM_INSTANCE);
if(PLIB_NVM_EEPROMIsReady(MY_NVM_INSTANCE))
    {
        //Perform operation
    }
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMEnable(NVM_MODULE_ID index)

PLIB_NVM_EEPROMDisable Function

Disables the EEPROM memory.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMDisable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function disables EEPROM memory access.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
if(PLIB_NVM_EEPROMOperationHasCompleted)
    {
        PLIB_NVM_EEPROMDisable(MY_NVM_INSTANCE);
    }
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMDisable(NVM_MODULE_ID index)

PLIB_NVM_EEPROMIsReady Function

Provides the availability status of the EEPROM.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMIsReady(NVM_MODULE_ID index);
```

Returns

- true - EEPROM is ready to use
- false - EEPROM is not yet ready to use

Description

This function provides the ready status of the EEPROM which was initiated by PLIB_NVM_EEPROMEnable.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_EEPROMEnable(MY_NVM_INSTANCE);
if(PLIB_NVM_EEPROMIsReady(MY_NVM_INSTANCE))
    {
        //Perform operation
    }
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_EEPROMIsReady(NVM_MODULE_ID index)

PLIB_NVM_EEPROMStopInIdleEnable Function

Discontinues EEPROM operation when device enters Idle mode.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMStopInIdleEnable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function discontinues EEPROM operation when device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_EEPROMStopInIdleEnable(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMStopInIdleEnable(NVM_MODULE_ID index)

PLIB_NVM_EEPROMStopInIdleDisable Function

Continues EEPROM operation when device enters Idle mode.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMStopInIdleDisable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function continues EEPROM operation when device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_EEPROMStopInIdleDisable(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMStopInIdleDisable(NVM_MODULE_ID index)

PLIB_NVM_EEPROMStopInIdleIsEnabled Function

Returns Stop in Idle mode status of the EEPROM operation.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMStopInIdleIsEnabled(NVM_MODULE_ID index);
```

Returns

- true - EEPROM discontinues operation when the device enters Idle mode
- false - EEPROM continues operation when the device enters Idle mode

Description

This function returns whether the EEPROM continues or discontinues operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool sidl_status;
sidl_status = PLIB_NVM_EEPROMStopInIdleIsEnabled(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_EEPROMStopInIdleIsEnabled(NVM_MODULE_ID index)

PLIB_NVM_EEPROMOperationSelect Function

Selects the operation to be performed on the EEPROM.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMOperationSelect(NVM_MODULE_ID index, EEPROM_OPERATION_MODE mode);
```

Returns

None.

Description

This function selects the operation to be performed on the EEPROM from the set of operations available from the enum EEPROM_OPERATION_MODE.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMOperationModeControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
PLIB_NVM_EEPROMOperationSelect(MY_NVM_INSTANCE, EEPROM_WORD_READ_OPERATION);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMOperationSelect(NVM_MODULE_ID index , EEPROM_OPERATION_MODE mode)

PLIB_NVM_EEPROMWriteEnable Function

Allows write or erase operation to the EEPROM.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMWriteEnable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function enables write or erase operations on the EEPROM and inhibits read.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableOperationControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
PLIB_NVM_EEPROMWriteEnable(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMWriteEnable(NVM_MODULE_ID index)

PLIB_NVM_EEPROMReadEnable Function

Allows read operation to the EEPROM.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMReadEnable(NVM_MODULE_ID index);
```

Returns

None.

Description

This function enables read operations on the EEPROM and inhibits write or erase.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableOperationControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
PLIB_NVM_EEPROMReadEnable(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMReadEnable(NVM_MODULE_ID index)

PLIB_NVM_EEPROMWriteIsEnabled Function

Returns EEPROM Write permission status.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMWriteIsEnabled(NVM_MODULE_ID index);
```

Returns

- true - EEPROM write or erase is enabled
- false - EEPROM write or erase is not enabled

Description

This function returns whether or not the EEPROM write or erase.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableOperationControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
if(PLIB_NVM_EEPROMWriteIsEnabled(MY_NVM_INSTANCE))
{
    //perform write or erase
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_EEPROMWriteIsEnabled(NVM_MODULE_ID index)

PLIB_NVM_EEPROMReadIsEnabled Function

Returns EEPROM read permission status.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMReadIsEnabled(NVM_MODULE_ID index);
```

Returns

- true - EEPROM read is enabled.
- false - EEPROM read is not enabled.

Description

This function returns whether or not the EEPROM read is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMEnableOperationControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
if(PLIB_NVM_EEPROMReadIsEnabled(MY_NVM_INSTANCE))
{
    //perform read
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_EEPROMReadIsEnabled(NVM_MODULE_ID index)

PLIB_NVM_EEPROMNextWriteCycleIsLong Function

Informs whether the next write or erase cycle of the EEPROM is long.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMNextWriteCycleIsLong(NVM_MODULE_ID index);
```

Returns

false - Next write/Erase cycle will not take more time to complete. true - Next write/Erase cycle will take more time to complete.

Description

This function informs whether the next write or erase cycle of the EEPROM is long (i.e., the next write or erase to the EEPROM address (passed by PLIB_NVM_EEPROMAddress) will require more time (~20 ms) than usual).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMLongWriteStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_NVM_EEPROMNextWriteCycleIsLong( MY_NVM_INSTANCE ))
{
    //stays here for ~20ms, better skip to other task for this time.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_EEPROMNextWriteCycleIsLong(NVM_MODULE_ID index)

PLIB_NVM_EEPROMErrorGet Function

Returns the EEPROM operation error.

File

plib_nvm.h

C

```
EEPROM_ERROR PLIB_NVM_EEPROMErrorGet(NVM_MODULE_ID index);
```

Returns

EEPROM_ERROR.

Description

This function returns the EEPROM operation error.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMErrorStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
EEPROM_ERROR error;
error = PLIB_NVM_EEPROMErrorGet(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

EEPROM_ERROR PLIB_NVM_EEPROMErrorGet(NVM_MODULE_ID index)

PLIB_NVM_EEPROMErrorClear Function

Clears the EEPROM operation error.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMErrorClear(NVM_MODULE_ID index);
```

Returns

None.

Description

This function clears the EEPROM operation error.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMErrorStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_EEPROMErrorClear(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMErrorClear(NVM_MODULE_ID index)

e) EEPROM Operation Functions

PLIB_NVM_EEPROMAddress Function

EEPROM address where operation has to be performed.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMAddress(NVM_MODULE_ID index, uint32_t address);
```

Returns

- true - EEPROM address is valid
- false - EEPROM address is invalid, please check if it is in the EEPROM address boundary and last two bits are '00'

Description

This function accepts the EEPROM address upon which to operate. Lower two bits of the address must always be '00'; otherwise, an error will occur when operation is started on EEPROM.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMAddressControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
uint32_t  address = MY_EEPROM_ADDRESS;
// Load address and check if it is valid
if(true ==PLIB_NVM_EEPROMAddress( NVM_ID_0 , address ))
{
    //Perform operation
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	The address of the EEPROM from or to which data will be read or written. This address should be a physical address.

Function

bool PLIB_NVM_EEPROMAddress(NVM_MODULE_ID index, uint32_t address)

PLIB_NVM_EEPROMDataToWrite Function

Accepts the data to be written into the EEPROM.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMDataToWrite(NVM_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function accepts data to be written into the EEPROM.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMDataControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation.

Example

```
PLIB_NVM_EEPROMDataToWrite(MY_NVM_INSTANCE, DATA);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to be written

Function

void PLIB_NVM_EEPROMDataToWrite (NVM_MODULE_ID index , uint32_t data)

PLIB_NVM_EEPROMKeySequenceWrite Function

Write mandatory KEY sequence to unlock the EEPROM write or erase protection.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMKeySequenceWrite(NVM_MODULE_ID index, uint32_t keysequence);
```

Returns

None.

Description

Without the KEY sequence write or erase operation will not be executed. Writing the KEY 0xEDB7 followed by writing the another KEY 0x1248 will unlock the EEPROM control register for write or erase operations. Writing any other value will lock the EEPROM control register. The unlock sequence is not necessary for a read operation.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMKeySequence in your application to determine whether this feature is available.

Preconditions

Interrupts should be disabled when writing the unlock sequence to the EEKEY register to prevent a disruption of the unlock sequence.

Example

```
//EEPROM is locked.
uint32_t key1 = 0xEDB7;
uint32_t key2 = 0x1248;
PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key1);
PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key2);
//EEPROM is now unlocked.
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
keysequence	Mandatory KEY sequence depending on the controller type

Function

void PLIB_NVM_EEPROMKeySequenceWrite (NVM_MODULE_ID index , uint32_t keysequence)

PLIB_NVM_EEPROMEraseStart Function

Initiates EEPROM erase cycle.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMEraseStart(NVM_MODULE_ID index);
```

Returns

None.

Description

This function initiates the EEPROM erase cycle.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStartOperationControl in your application to determine whether this feature is available.

Preconditions

PLIB_NVM_EEPROMWriteEnable should be called.

Example

```
// Enable write or erase operation
PLIB_NVM_EEPROMWriteEnable( NVM_ID_0 );
// unlock
PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key1);
PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key2);
// Start the erase operation
PLIB_NVM_EEPROMEraseStart( NVM_MODULE_ID index );
// Wait until erase is complete
while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMEraseStart(NVM_MODULE_ID index)

PLIB_NVM_EEPROMWriteStart Function

Initiates a EEPROM write cycle.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMWriteStart(NVM_MODULE_ID index);
```

Returns

None.

Description

This function initiates the EEPROM write cycle.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStartOperationControl in your application to determine whether this feature is available.

Preconditions

EEPROM write should be enabled through PLIB_NVM_EEPROMWriteEnable.

Example

```
// Enable write operation
PLIB_NVM_EEPROMWriteEnable( NVM_ID_0 );
// Provide data
PLIB_NVM_EEPROMDataToWrite ( NVM_ID_0 , ee_data ) ;
// unlock
PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key1);
PLIB_NVM_EEPROMKeySequenceWrite(MY_NVM_INSTANCE, key2);
// Start the write operation
PLIB_NVM_EEPROMWriteStart( NVM_MODULE_ID index );
// Wait until write is complete
while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMWriteStart(NVM_MODULE_ID index)

PLIB_NVM_EEPROMReadStart Function

Initiates a EEPROM read cycle.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMReadStart(NVM_MODULE_ID index);
```

Returns

None.

Description

This function initiates the EEPROM read cycle.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStartOperationControl in your application to determine whether this feature is available.

Preconditions

EEPROM read should be enabled through PLIB_NVM_EEPROMReadEnable.

Example

```
// Enable read operation
PLIB_NVM_EEPROMReadEnable( NVM_ID_0 );
// Start the read cycle
PLIB_NVM_EEPROMReadStart( NVM_MODULE_ID index );
// Wait until read is complete
while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
// Read the data
ee_data = PLIB_NVM_EEPROMRead( NVM_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMReadStart(NVM_MODULE_ID index)

PLIB_NVM_EEPROMOperationHasCompleted Function

Provides the status of the EEPROM write or erase or read cycle.

File

plib_nvm.h

C

```
bool PLIB_NVM_EEPROMOperationHasCompleted(NVM_MODULE_ID index);
```

Returns

- true - Write or erase cycle has completed
- false - Write or erase cycle has not completed

Description

This function provides the status of the EEPROM write or erase or read operation status.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMStartOperationControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Start the erase operation
PLIB_NVM_EEPROMEraseStart( NVM_MODULE_ID index );
// Wait until erase is complete
while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_NVM_EEPROMOperationHasCompleted(NVM_MODULE_ID index)

PLIB_NVM_EEPROMRead Function

Read the EEPROM data.

File

plib_nvm.h

C

```
uint32_t PLIB_NVM_EEPROMRead(NVM_MODULE_ID index);
```

Returns

32-bit EEPROM data.

Description

This function returns the EERPOM data that is been fetched by performing the EEPROM read operation sequence.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMDataControl in your application to determine whether this feature is available.

Preconditions

This API will have no affect if there is any ongoing EEPROM operation, or may return junk data.

Example

```
uint32_t EEPROM_Data;
// Start the read operation
PLIB_NVM_EEPROMReadStart( NVM_MODULE_ID index );
// Wait until read is complete
while (PLIB_NVM_EEPROMOperationHasCompleted( NVM_ID_0 ));
// Now read the data
EEPROM_Data = PLIB_NVM_EEPROMRead( NVM_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_NVM_EEPROMRead(NVM_MODULE_ID index)

PLIB_NVM_EEPROMOperationAbort Function

Aborts the current EEPROM operation.

File

plib_nvm.h

C

```
void PLIB_NVM_EEPROMOperationAbort(NVM_MODULE_ID index);
```

Returns

None.

Description

This function aborts the on-going write or erase operation as soon as possible.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_NVM_ExistsEEPROMOperationAbortControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_NVM_EEPROMOperationAbort(MY_NVM_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_NVM_EEPROMOperationAbort(NVM_MODULE_ID index)

f) Feature Existence Functions

PLIB_NVM_ExistsAddressModifyControl Function

Identifies whether the AddressModifyControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsAddressModifyControl(NVM_MODULE_ID index);
```

Returns

- true - The AddressModifyControl feature is supported on the device
- false - The AddressModifyControl feature is not supported on the device

Description

This function identifies whether the AddressModifyControl feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_FlashAddressToModify

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsAddressModifyControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsBootPageWriteProtect Function

Identifies whether the BootPageWriteProtect feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsBootPageWriteProtect(NVM_MODULE_ID index);
```

Returns

- true - The BootPageWriteProtect feature is supported on the device
- false - The BootPageWriteProtect feature is not supported on the device

Description

This function identifies whether the BootPageWriteProtect feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_BootPageWriteProtectionEnable
- PLIB_NVM_BootPageWriteProtectionDisable
- PLIB_NVM_IsBootPageWriteProtected

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsBootPageWriteProtect(NVM_MODULE_ID index)

PLIB_NVM_ExistsFlashBankRegionSelect Function

Identifies whether the FlashBankRegionSelect feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsFlashBankRegionSelect(NVM_MODULE_ID index);
```

Returns

- true - The FlashBankRegionSelect feature is supported on the device
- false - The FlashBankRegionSelect feature is not supported on the device

Description

This function identifies whether the FlashBankRegionSelect feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_ProgramFlashBank1LowerRegion
- PLIB_NVM_ProgramFlashBank2LowerRegion

- PLIB_NVM_ProgramFlashBank2IsLowerRegion

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsFlashBankRegionSelect(NVM_MODULE_ID index)

PLIB_NVM_ExistsFlashWPMemoryRangeProvide Function

Identifies whether the FlashWPMemoryRangeProvide feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsFlashWPMemoryRangeProvide(NVM_MODULE_ID index);
```

Returns

- true - The FlashWPMemoryRangeProvide feature is supported on the device
- false - The FlashWPMemoryRangeProvide feature is not supported on the device

Description

This function identifies whether the FlashWPMemoryRangeProvide feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_FlashWriteProtectMemoryAreaRange

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsFlashWPMemoryRangeProvide(NVM_MODULE_ID index)

PLIB_NVM_ExistsKeySequence Function

Identifies whether the KeySequence feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsKeySequence(NVM_MODULE_ID index);
```

Returns

- true - The KeySequence feature is supported on the device
- false - The KeySequence feature is not supported on the device

Description

This function identifies whether the KeySequence feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_FlashWriteKeySequence

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsKeySequence(NVM_MODULE_ID index)

PLIB_NVM_ExistsLockBootSelect Function

Identifies whether the LockBootSelect feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsLockBootSelect(NVM_MODULE_ID index);
```

Returns

- true - The LockBootSelect feature is supported on the device
- false - The LockBootSelect feature is not supported on the device

Description

This function identifies whether the LockBootSelect feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_LockBootMemory
- PLIB_NVM_IsBootMemoryLocked

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsLockBootSelect(NVM_MODULE_ID index)

PLIB_NVM_ExistsLockPFMSelect Function

Identifies whether the LockPFMSelect feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsLockPFMSelect(NVM_MODULE_ID index);
```

Returns

- true - The LockPFMSelect feature is supported on the device
- false - The LockPFMSelect feature is not supported on the device

Description

This function identifies whether the LockPFMSelect feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_LockProgramFlashMemory
- PLIB_NVM_IsProgramFlashMemoryLocked

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsLockPFMSelect(NVM_MODULE_ID index)

PLIB_NVM_ExistsLowVoltageError Function

Identifies whether the LowVoltageError feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsLowVoltageError(NVM_MODULE_ID index);
```

Returns

- true - The LowVoltageStatus feature is supported on the device
- false - The LowVoltageStatus feature is not supported on the device

Description

This function identifies whether the LowVoltageStatus feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_LowVoltageIsDetected

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsLowVoltageError(NVM_MODULE_ID index)

PLIB_NVM_ExistsLowVoltageStatus Function

Identifies whether the LowVoltageStatus feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsLowVoltageStatus(NVM_MODULE_ID index);
```

Returns

- true - The LowVoltageStatus feature is supported on the device
- false - The LowVoltageStatus feature is not supported on the device

Description

This function identifies whether the LowVoltageStatus feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_LowVoltageEventIsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsLowVoltageStatus(NVM_MODULE_ID index)

PLIB_NVM_ExistsMemoryModificationControl Function

Identifies whether the MemoryModificationControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsMemoryModificationControl(NVM_MODULE_ID index);
```

Returns

- true - The MemoryModificationControl feature is supported on the device
- false - The MemoryModificationControl feature is not supported on the device

Description

This function identifies whether the MemoryModificationControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_MemoryModifyEnable
- PLIB_NVM_MemoryModifyInhibit

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsMemoryModificationControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsOperationMode Function

Identifies whether the OperationMode feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsOperationMode(NVM_MODULE_ID index);
```

Returns

- true - The OperationMode feature is supported on the device
- false - The OperationMode feature is not supported on the device

Description

This function identifies whether the OperationMode feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_MemoryOperationSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsOperationMode(NVM_MODULE_ID index)

PLIB_NVM_ExistsProvideData Function

Identifies whether the ProvideData feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsProvideData(NVM_MODULE_ID index);
```

Returns

- true - The ProvideData feature is supported on the device
- false - The ProvideData feature is not supported on the device

Description

This function identifies whether the ProvideData feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_FlashProvideData

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsProvideData(NVM_MODULE_ID index)

PLIB_NVM_ExistsProvideQuadData Function

Identifies whether the ProvideQuadData feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsProvideQuadData(NVM_MODULE_ID index);
```

Returns

- true - The ProvideQuadData feature is supported on the device
- false - The ProvideQuadData feature is not supported on the device

Description

This function identifies whether the ProvideQuadData feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_FlashProvideQuadData

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsProvideQuadData(NVM_MODULE_ID index)

PLIB_NVM_ExistsSourceAddress Function

Identifies whether the SourceAddress feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsSourceAddress(NVM_MODULE_ID index);
```

Returns

- true - The SourceAddress feature is supported on the device
- false - The SourceAddress feature is not supported on the device

Description

This function identifies whether the SourceAddress feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_DataBlockSourceAddress

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsSourceAddress(NVM_MODULE_ID index)

PLIB_NVM_ExistsWriteErrorStatus Function

Identifies whether the WriteErrorStatus feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsWriteErrorStatus(NVM_MODULE_ID index);
```

Returns

- true - The WriteErrorStatus feature is supported on the device
- false - The WriteErrorStatus feature is not supported on the device

Description

This function identifies whether the WriteErrorStatus feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_WriteOperationHasTerminated

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsWriteErrorStatus(NVM_MODULE_ID index)

PLIB_NVM_ExistsWriteOperation Function

Identifies whether the WriteOperation feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsWriteOperation(NVM_MODULE_ID index);
```

Returns

- true - The WriteOperation feature is supported on the device
- false - The WriteOperation feature is not supported on the device

Description

This function identifies whether the WriteOperation feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_FlashRead
- PLIB_NVM_FlashWriteStart
- PLIB_NVM_FlashEraseStart
- PLIB_NVM_FlashWriteCycleHasCompleted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsWriteOperation(NVM_MODULE_ID index)

PLIB_NVM_ExistsBootFlashBankRegion Function

Identifies whether the BootFlashBankRegion feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsBootFlashBankRegion(NVM_MODULE_ID index);
```

Returns

- true - The BootFlashBankRegion feature is supported on the device
- false - The BootFlashBankRegion feature is not supported on the device

Description

This function identifies whether the BootFlashBankRegion feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_BootFlashBank1LowerRegion
- PLIB_NVM_BootFlashBank2LowerRegion
- PLIB_NVM_BootFlashBank2IsLowerRegion

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsBootFlashBankRegion(NVM_MODULE_ID index)

PLIB_NVM_ExistsSwapLockControl Function

Identifies whether the SwapLockControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsSwapLockControl(NVM_MODULE_ID index);
```

Returns

- true - The SwapLockControl feature is supported on the device
- false - The SwapLockControl feature is not supported on the device

Description

This function identifies whether the SwapLockControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_FlashSwapLockSelect
- PLIB_NVM_FlashSwapLockStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsSwapLockControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMAddressControl Function

Identifies whether the EEPROMAddressControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMAddressControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMAddressControl feature is supported on the device
- false - The EEPROMAddressControl feature is not supported on the device

Description

This function identifies whether the EEPROMAddressControl feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_EEPROMAddress

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMAddressControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMDataControl Function

Identifies whether the EEPROMDataControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMDataControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMDataControl feature is supported on the device
- false - The EEPROMDataControl feature is not supported on the device

Description

This function identifies whether the EEPROMDataControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_EEPROMDataToWrite
- PLIB_NVM_EEPROMRead

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMDataControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMEnableControl Function

Identifies whether the EEPROMEnableControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMEnableControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMEnableControl feature is supported on the device
- false - The EEPROMEnableControl feature is not supported on the device

Description

This function identifies whether the EEPROMEnableControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_EEPROMEnable
- PLIB_NVM_EEPROMDisable
- PLIB_NVM_EEPROMIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMEnableControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMEnableOperationControl Function

Identifies whether the EEPROMEnableOperationControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMEnableOperationControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMEnableOperationControl feature is supported on the device
- false - The EEPROMEnableOperationControl feature is not supported on the device

Description

This function identifies whether the EEPROMEnableOperationControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_EEPROMWriteEnable
- PLIB_NVM_EEPROMWriteIsEnabled
- PLIB_NVM_EEPROMReadEnable
- PLIB_NVM_EEPROMReadIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMEnableOperationControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMErrorStatus Function

Identifies whether the EEPROMErrorStatus feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMErrorStatus(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMErrorStatus feature is supported on the device
- false - The EEPROMErrorStatus feature is not supported on the device

Description

This function identifies whether the EEPROMErrorStatus feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_EEPROMErrorGet
- PLIB_NVM_EEPROMErrorClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMErrorStatus(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMKeySequence Function

Identifies whether the EEPROMKeySequence feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMKeySequence(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMKeySequence feature is supported on the device
- false - The EEPROMKeySequence feature is not supported on the device

Description

This function identifies whether the EEPROMKeySequence feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_EEPROMKeySequenceWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMKeySequence(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMLongWriteStatus Function

Identifies whether the EEPROMLongWriteStatus feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMLongWriteStatus(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMLongWriteStatus feature is supported on the device
- false - The EEPROMLongWriteStatus feature is not supported on the device

Description

This function identifies whether the EEPROMLongWriteStatus feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_EEPROMNextWriteCycleIsLong

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMLongWriteStatus(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMOperationAbortControl Function

Identifies whether the EEPROMOperationAbortControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMOperationAbortControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMOperationAbortControl feature is supported on the device
- false - The EEPROMOperationAbortControl feature is not supported on the device

Description

This function identifies whether the EEPROMOperationAbortControl feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_EEPROMOperationAbort

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMOperationAbortControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMOperationModeControl Function

Identifies whether the EEPROMOperationModeControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMOperationModeControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMOperationModeControl feature is supported on the device
- false - The EEPROMOperationModeControl feature is not supported on the device

Description

This function identifies whether the EEPROMOperationModeControl feature is available on the NVM module. When this function returns true, this function is supported on the device:

- PLIB_NVM_EEPROMOperationSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMOperationModeControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMStartOperationControl Function

Identifies whether the EEPROMStartOperationControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMStartOperationControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMStartOperationControl feature is supported on the device
- false - The EEPROMStartOperationControl feature is not supported on the device

Description

This function identifies whether the EEPROMStartOperationControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_EEPROMReadStart
- PLIB_NVM_EEPROMWriteStart
- PLIB_NVM_EEPROMEraseStart
- PLIB_NVM_EEPROMOperationHasCompleted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMStartOperationControl(NVM_MODULE_ID index)

PLIB_NVM_ExistsEEPROMStopInIdleControl Function

Identifies whether the EEPROMStopInIdleControl feature exists on the NVM module.

File

plib_nvm.h

C

```
bool PLIB_NVM_ExistsEEPROMStopInIdleControl(NVM_MODULE_ID index);
```

Returns

- true - The EEPROMStopInIdleControl feature is supported on the device
- false - The EEPROMStopInIdleControl feature is not supported on the device

Description

This function identifies whether the EEPROMStopInIdleControl feature is available on the NVM module. When this function returns true, these functions are supported on the device:

- PLIB_NVM_EEPROMStopInIdleEnable
- PLIB_NVM_EEPROMStopInIdleDisable
- PLIB_NVM_EEPROMStopInIdleIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_NVM_ExistsEEPROMStopInIdleControl(NVM_MODULE_ID index)

g) Data Types and Constants

NVM_BOOT_MEMORY_AREA Enumeration

Lists the different possible boot memory region.

File

plib_nvm_help.h

C

```
typedef enum {
  LOWER_BOOT_ALIAS_AREA,
  UPPER_BOOT_ALIAS_AREA,
  LOWER_BOOT_MEMORY_AREA_NOT_SUPPORTED
} NVM_BOOT_MEMORY_AREA;
```

Members

Members	Description
LOWER_BOOT_ALIAS_AREA	Lower boot alias region
UPPER_BOOT_ALIAS_AREA	Upper boot alias region
LOWER_BOOT_MEMORY_AREA_NOT_SUPPORTED	Lower boot alias region is not supported

Description

NVM Boot Memory Area

This enumeration lists the different possible boot memory region.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

NVM_BOOT_MEMORY_PAGE Enumeration

Lists the different NVM boot memory pages.

File

plib_nvm_help.h

C

```
typedef enum {
  LOWER_BOOT_ALIAS_PAGE4,
```

```
  LOWER_BOOT_ALIAS_PAGE3,
  LOWER_BOOT_ALIAS_PAGE2,
  LOWER_BOOT_ALIAS_PAGE1,
  LOWER_BOOT_ALIAS_PAGE0,
  UPPER_BOOT_ALIAS_PAGE4,
  UPPER_BOOT_ALIAS_PAGE3,
  UPPER_BOOT_ALIAS_PAGE2,
  UPPER_BOOT_ALIAS_PAGE1,
  UPPER_BOOT_ALIAS_PAGE0,
  LOWER_BOOT_MEMORY_PAGE_NOT_SUPPORTED
} NVM_BOOT_MEMORY_PAGE;
```

Members

Members	Description
LOWER_BOOT_ALIAS_PAGE4	Lower boot alias page 4
LOWER_BOOT_ALIAS_PAGE3	Lower boot alias page 3
LOWER_BOOT_ALIAS_PAGE2	Lower boot alias page 2
LOWER_BOOT_ALIAS_PAGE1	Lower boot alias page 1
LOWER_BOOT_ALIAS_PAGE0	Lower boot alias page 0
UPPER_BOOT_ALIAS_PAGE4	Upper boot alias page 4
UPPER_BOOT_ALIAS_PAGE3	Upper boot alias page 3
UPPER_BOOT_ALIAS_PAGE2	Upper boot alias page 2
UPPER_BOOT_ALIAS_PAGE1	Upper boot alias page 1
UPPER_BOOT_ALIAS_PAGE0	Upper boot alias page 0
LOWER_BOOT_MEMORY_PAGE_NOT_SUPPORTED	Lower boot memory page not supported

Description

NVM Boot Memory Page

This enumeration lists the different NVM boot memory page details.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

NVM_OPERATION_MODE Enumeration

Lists the different Flash operation modes.

File

plib_nvm_help.h

C

```
typedef enum {
  WORD_PROGRAM_OPERATION,
  ROW_PROGRAM_OPERATION,
  PAGE_ERASE_OPERATION,
  FLASH_ERASE_OPERATION,
  UPPER_FLASH_REGION_ERASE_OPERATION,
  LOWER_FLASH_REGION_ERASE_OPERATION,
  QUAD_WORD_PROGRAM_OPERATION,
  NO_OPERATION
} NVM_OPERATION_MODE;
```

Members

Members	Description
WORD_PROGRAM_OPERATION	Word Program Operation
ROW_PROGRAM_OPERATION	Row Program Operation
PAGE_ERASE_OPERATION	Page Erase Operation
FLASH_ERASE_OPERATION	Flash Erase Operation
UPPER_FLASH_REGION_ERASE_OPERATION	Upper Flash Region Erase Operation
LOWER_FLASH_REGION_ERASE_OPERATION	Lower Flash Region Erase Operation
QUAD_WORD_PROGRAM_OPERATION	Quad Word Program Operation
NO_OPERATION	No Operation

Description

NVM Operation Mode

This enumeration lists the different Flash operation modes.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

NVM_MODULE_ID Enumeration

Possible instances of the NVM module.

File

plib_nvm_help.h

C

```
typedef enum {
  NVM_ID_0,
  NVM_NUMBER_OF_MODULES
} NVM_MODULE_ID;
```

Members

Members	Description
NVM_ID_0	NVM Module 0 ID
NVM_NUMBER_OF_MODULES	Number of available NVM modules.

Description

NVM Module ID

This data type defines the possible instances of the NVM module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

NVM_FLASH_SWAP_LOCK_TYPE Enumeration

Lists the possible type of Flash swap lock.

File

plib_nvm_help.h

C

```
typedef enum {
  NVM_FLASH_SWAP_UNLOCKED = 0x0,
  NVM_FLASH_SWAP_LOCKED = 0x1,
  NVM_FLASH_SWAP_LOCKED_UNTIL_RESET = 0x3
} NVM_FLASH_SWAP_LOCK_TYPE;
```

Members

Members	Description
NVM_FLASH_SWAP_UNLOCKED = 0x0	Program Flash Bank and Boot Flash Bank region swapping is Not locked
NVM_FLASH_SWAP_LOCKED = 0x1	Program Flash Bank and Boot Flash Bank region swapping is locked
NVM_FLASH_SWAP_LOCKED_UNTIL_RESET = 0x3	Program Flash Bank and Boot Flash Bank region swapping is locked until device is reset

Description

NVM Flash Swap Lock

This enumeration lists the possible type of Flash swap lock.

Remarks

This feature may not be available on all the devices. Refer to the specific device header files for availability.

EEPROM_ERROR Enumeration

Lists the different EEPROM operation errors.

File

plib_nvm_help.h

C

```
typedef enum {
  NO_ERROR,
  VERIFY_ERROR,
  INVALID_OPERATION,
  BOR_ERROR
} EEPROM_ERROR;
```

Members

Members	Description
NO_ERROR	No error has occurred
VERIFY_ERROR	Error occurred during verification
INVALID_OPERATION	Operation selected and executed are not the same
BOR_ERROR	BOR has suspended EEPROM operation

Description

EEPROM operation error

This enumeration lists the different EEPROM operation errors.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

EEPROM_OPERATION_MODE Enumeration

Lists the different EEPROM operation modes.

File

plib_nvm_help.h

C

```
typedef enum {
  EEPROM_WORD_READ_OPERATION,
  EEPROM_WORD_WRITE_OPERATION,
  EEPROM_FORCED_WORD_ERASE_OPERATION,
  EEPROM_ERASE_ALL_OPERATION,
  EEPROM_CONFIG_WRITE_OPERATION
} EEPROM_OPERATION_MODE;
```

Members

Members	Description
EEPROM_WORD_READ_OPERATION	Word read operation
EEPROM_WORD_WRITE_OPERATION	Word write operation
EEPROM_FORCED_WORD_ERASE_OPERATION	Page erase operation. Under normal conditions, there is no need to attempt a Forced Word Erase. The Data EEPROM has internal logic which automatically manages all read/erase/write command sequences. Software execution of the Forced Word Erase operation should be used in response to write verification error, VERIFY_ERROR
EEPROM_ERASE_ALL_OPERATION	Bulk erase operation. Unlike the read command, the bulk erase command cannot be aborted by software. During an erase cycle, any software attempt to write to the EECON register will be ignored
EEPROM_CONFIG_WRITE_OPERATION	Write to configuration registers operation. Before accessing the EEPROM at full speed, it is necessary to program configuration values into the Data EEPROM controller after enabling it. This is done through the Configuration Write operation

Description

EEPROM Operation Mode

This enumeration lists the different EEPROM operation modes.

Remarks

This enumeration is processor specific and is defined in the processor- specific header files.

Files

Files

Name	Description
plib_nvm.h	NVM PLIB Interface Header for NVM common definitions.
plib_nvm_help.h	Defines the NVM Peripheral Library data types.

Description

This section lists the source and header files used by the library.

plib_nvm.h

NVM PLIB Interface Header for NVM common definitions.

Functions

	Name	Description
	PLIB_NVM_BootFlashBank1LowerRegion	Maps Boot Flash Bank 1 to lower mapped region.
	PLIB_NVM_BootFlashBank2IsLowerRegion	Gives the status of Boot Flash Bank mapping.
	PLIB_NVM_BootFlashBank2LowerRegion	Maps Boot Flash Bank 2 to the lower mapped region.
	PLIB_NVM_BootPageWriteProtectionDisable	Write protection for selected boot page is disabled.
	PLIB_NVM_BootPageWriteProtectionEnable	Locks the selected boot page.
	PLIB_NVM_DataBlockSourceAddress	Takes the address parameter in the argument and loads the base address from which data has to be copied into Flash memory.
	PLIB_NVM_EEPROMAddress	EEPROM address where operation has to be performed.
	PLIB_NVM_EEPROMDataToWrite	Accepts the data to be written into the EEPROM.
	PLIB_NVM_EEPROMDisable	Disables the EEPROM memory.
	PLIB_NVM_EEPROMEnable	Enables the EEPROM memory.
	PLIB_NVM_EEPROMEraseStart	Initiates EEPROM erase cycle.
	PLIB_NVM_EEPROMErrorClear	Clears the EEPROM operation error.
	PLIB_NVM_EEPROMErrorGet	Returns the EEPROM operation error.
	PLIB_NVM_EEPROMIsReady	Provides the availability status of the EEPROM.
	PLIB_NVM_EEPROMKeySequenceWrite	Write mandatory KEY sequence to unlock the EEPROM write or erase protection.
	PLIB_NVM_EEPROMNextWriteCycleIsLong	Informs whether the next write or erase cycle of the EEPROM is long.
	PLIB_NVM_EEPROMOperationAbort	Aborts the current EEPROM operation.
	PLIB_NVM_EEPROMOperationHasCompleted	Provides the status of the EEPROM write or erase or read cycle.
	PLIB_NVM_EEPROMOperationSelect	Selects the operation to be performed on the EEPROM.
	PLIB_NVM_EEPROMRead	Read the EEPROM data.
	PLIB_NVM_EEPROMReadEnable	Allows read operation to the EEPROM.
	PLIB_NVM_EEPROMReadIsEnabled	Returns EEPROM read permission status.
	PLIB_NVM_EEPROMReadStart	Initiates a EEPROM read cycle.
	PLIB_NVM_EEPROMStopInIdleDisable	Continues EEPROM operation when device enters Idle mode.
	PLIB_NVM_EEPROMStopInIdleEnable	Discontinues EEPROM operation when device enters Idle mode.
	PLIB_NVM_EEPROMStopInIdleIsEnabled	Returns Stop in Idle mode status of the EEPROM operation.
	PLIB_NVM_EEPROMWriteEnable	Allows write or erase operation to the EEPROM.
	PLIB_NVM_EEPROMWriteIsEnabled	Returns EEPROM Write permission status.
	PLIB_NVM_EEPROMWriteStart	Initiates a EEPROM write cycle.
	PLIB_NVM_ExistsAddressModifyControl	Identifies whether the AddressModifyControl feature exists on the NVM module.
	PLIB_NVM_ExistsBootFlashBankRegion	Identifies whether the BootFlashBankRegion feature exists on the NVM module.

	PLIB_NVM_ExistsBootPageWriteProtect	Identifies whether the BootPageWriteProtect feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMAddressControl	Identifies whether the EEPROMAddressControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMDataControl	Identifies whether the EEPROMDataControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMEnableControl	Identifies whether the EEPROMEnableControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMEnableOperationControl	Identifies whether the EEPROMEnableOperationControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMErrorStatus	Identifies whether the EEPROMErrorStatus feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMKeySequence	Identifies whether the EEPROMKeySequence feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMLongWriteStatus	Identifies whether the EEPROMLongWriteStatus feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMOperationAbortControl	Identifies whether the EEPROMOperationAbortControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMOperationModeControl	Identifies whether the EEPROMOperationModeControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMStartOperationControl	Identifies whether the EEPROMStartOperationControl feature exists on the NVM module.
	PLIB_NVM_ExistsEEPROMStopInIdleControl	Identifies whether the EEPROMStopInIdleControl feature exists on the NVM module.
	PLIB_NVM_ExistsFlashBankRegionSelect	Identifies whether the FlashBankRegionSelect feature exists on the NVM module.
	PLIB_NVM_ExistsFlashWPMemoryRangeProvide	Identifies whether the FlashWPMemoryRangeProvide feature exists on the NVM module.
	PLIB_NVM_ExistsKeySequence	Identifies whether the KeySequence feature exists on the NVM module.
	PLIB_NVM_ExistsLockBootSelect	Identifies whether the LockBootSelect feature exists on the NVM module.
	PLIB_NVM_ExistsLockPFMSelect	Identifies whether the LockPFMSelect feature exists on the NVM module.
	PLIB_NVM_ExistsLowVoltageError	Identifies whether the LowVoltageError feature exists on the NVM module.
	PLIB_NVM_ExistsLowVoltageStatus	Identifies whether the LowVoltageStatus feature exists on the NVM module.
	PLIB_NVM_ExistsMemoryModificationControl	Identifies whether the MemoryModificationControl feature exists on the NVM module.
	PLIB_NVM_ExistsOperationMode	Identifies whether the OperationMode feature exists on the NVM module.
	PLIB_NVM_ExistsProvideData	Identifies whether the ProvideData feature exists on the NVM module.
	PLIB_NVM_ExistsProvideQuadData	Identifies whether the ProvideQuadData feature exists on the NVM module.
	PLIB_NVM_ExistsSourceAddress	Identifies whether the SourceAddress feature exists on the NVM module.
	PLIB_NVM_ExistsSwapLockControl	Identifies whether the SwapLockControl feature exists on the NVM module.
	PLIB_NVM_ExistsWriteErrorStatus	Identifies whether the WriteErrorStatus feature exists on the NVM module.
	PLIB_NVM_ExistsWriteOperation	Identifies whether the WriteOperation feature exists on the NVM module.
	PLIB_NVM_FlashAddressToModify	Modifies the Flash memory address.
	PLIB_NVM_FlashEraseStart	Performs erase operation on the selected Flash memory area.
	PLIB_NVM_FlashProvideData	Provides the data to be written into Flash memory.
	PLIB_NVM_FlashProvideQuadData	Provides the quad data to be written into Flash memory.
	PLIB_NVM_FlashRead	Read the specified address of Flash memory.
	PLIB_NVM_FlashSwapLockSelect	Selects the kind of Flash swap lock required.
	PLIB_NVM_FlashSwapLockStatusGet	Get the status of Swap lock bits.
	PLIB_NVM_FlashWriteCycleHasCompleted	Provides the status of the Flash write cycle.
	PLIB_NVM_FlashWriteKeySequence	Copies the mandatory KEY sequence into the respective registers.
	PLIB_NVM_FlashWriteProtectMemoryAreaRange	Sets the address below which physical memory will be write protected.
	PLIB_NVM_FlashWriteStart	Performs a write operation on the Flash memory row selected.
	PLIB_NVM_IsBootMemoryLocked	Provides lock status of boot page write-protect bits.
	PLIB_NVM_IsBootPageWriteProtected	Provides write protection status for boot memory page.
	PLIB_NVM_IsProgramFlashMemoryLocked	Provides lock status of Program Flash Write-Protect register.
	PLIB_NVM_LockBootMemory	Locks the boot write-protect bits.
	PLIB_NVM_LockProgramFlashMemory	Lock the Program Flash write-protect register.
	PLIB_NVM_LowVoltageEventIsActive	Provides low voltage detection status.

	PLIB_NVM_LowVoltageIsDetected	Provides low voltage error detection status.
	PLIB_NVM_MemoryModifyEnable	Allows write cycles to Flash memory.
	PLIB_NVM_MemoryModifyInhibit	Inhibits write cycles to Flash memory.
	PLIB_NVM_MemoryOperationSelect	Selects the operation to be performed on Flash memory.
	PLIB_NVM_ProgramFlashBank1LowerRegion	Maps Flash Bank 1 to the lower mapped region.
	PLIB_NVM_ProgramFlashBank2IsLowerRegion	Gives the status of Program Flash Bank mapping.
	PLIB_NVM_ProgramFlashBank2LowerRegion	Maps the bank 2 to lower mapped region.
	PLIB_NVM_WriteOperationHasTerminated	Provides the status of the Flash write operation or sequence.

Description

Non-Volatile Memory (NVM) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the NVM PLIB for all families of Microchip microcontrollers..The definitions in this file are common to NVM peripheral.

File Name

plib_nvm.h

Company

Microchip Technology Inc.

plib_nvm_help.h

Defines the NVM Peripheral Library data types.

Enumerations

	Name	Description
	EEPROM_ERROR	Lists the different EEPROM operation errors.
	EEPROM_OPERATION_MODE	Lists the different EEPROM operation modes.
	NVM_BOOT_MEMORY_AREA	Lists the different possible boot memory region.
	NVM_BOOT_MEMORY_PAGE	Lists the different NVM boot memory pages.
	NVM_FLASH_SWAP_LOCK_TYPE	Lists the possible type of Flash swap lock.
	NVM_MODULE_ID	Possible instances of the NVM module.
	NVM_OPERATION_MODE	Lists the different Flash operation modes.

Description

NVM Peripheral Library Constants Header

This header file contains the definitions of the data types and constants that make up the interface to the NVM Peripheral Library for Microchip microcontrollers. The definitions in this file are for NVM module.

File Name

plib_nvm_help.h

Company

Microchip Technology Inc.

Output Compare Peripheral Library

This section describes the Output Compare Peripheral Library.

Introduction

This library provides a low-level abstraction of the Output Compare Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Output Compare module is primarily used to generate a single pulse or a series of pulses in response to selected time base events.

Output Compare module operation essentially requires a timer and one or two compare values. Users have an option to select either a 32-bit timer or a 16-bit timer. The timer can be programmed to count at a desired frequency and count up to a desired period value. Details regarding timer

operation and setup can be found in the Timer Peripheral Library. The Output Compare module compares the value of the timer with the compare values depending on the selected operating mode. When a compare match occurs between a timer value and compare values, the Output Compare module outputs a change of state on its output pins in accordance with the chosen Output Compare compare mode logic. Either a single pulse or a sequence of pulses may be generated.

Some of the key operating modes of the OC module are:

- Single Compare Set High/Low modes: In these compare modes, a compare match between the timer and the buffer (primary compare value) sets the output High/Low. The output remains at the same state after compare match event unless the module is disabled or the mode is changed.
- Single Compare Toggle mode: Output toggles state at every compare match event between the timer and the buffer (primary compare value)
- Dual compare Single/Continuous Pulse modes: These modes require two compare values. Leading edge of output pulse is generated during compare match of the incrementing timer and the buffer (primary compare value). Trailing edge of output pulse results when a compare match occurs between the incrementing timer and the pulse width value (secondary compare value). The output may be a single pulse or a sequence of pulses.
- Pulse-Width Modulation (PWM) modes: In this mode, output goes high when a compare match occurs between timer and the pulse width value (duty cycle). The output is reset back to its initial state when the timer resets after attaining its maximum count.

The Output Compare module also provides programmable interrupt generation on a compare match event. In PWM mode, hardware-based Fault detection and automatic output disable features are provided.

Figure 1 shows a block diagram of the Output Compare module. A compare match between the timer value and the compare values generates a pulse at the output.

Figure 1: Output Compare Module Block Diagram

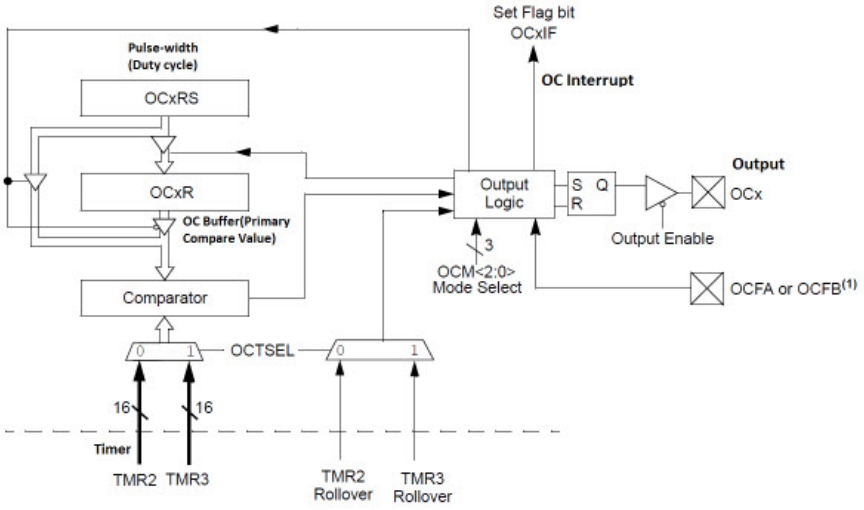
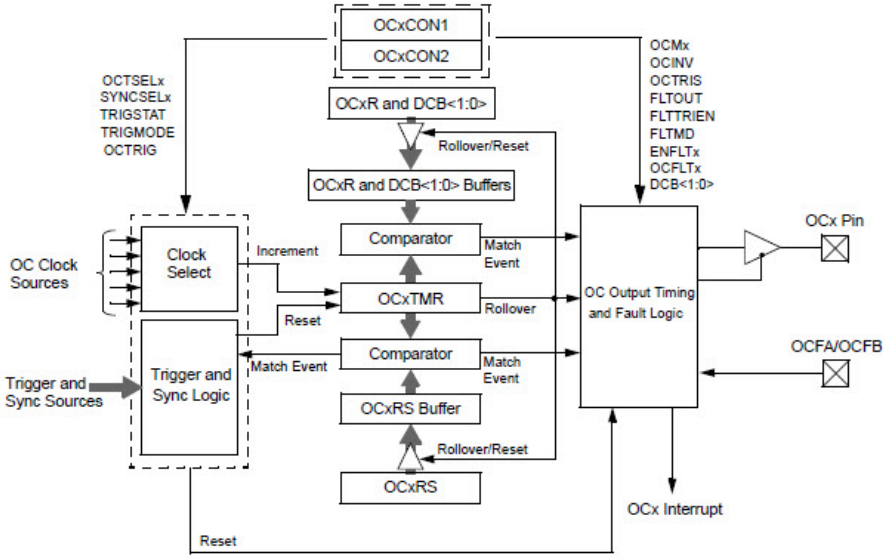


Figure 2 shows a block diagram of the Output Compare module with dedicated timers, present on some devices. It facilitates the use of multiple Output Compare modules operating synchronously or the use of an asynchronous trigger to generate a pulse.

Figure 2: Output Compare Module Block Diagram with Dedicated Timers



Using the Library

This topic describes the basic architecture of the Output Compare Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_oc.h`

The interface to the Output Compare Peripheral library is defined in the `plib_oc.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Output Compare Peripheral Library must include `peripheral.h`.

Library File:

The Output Compare Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the Peripheral interacts with the framework.

Hardware Abstraction Model

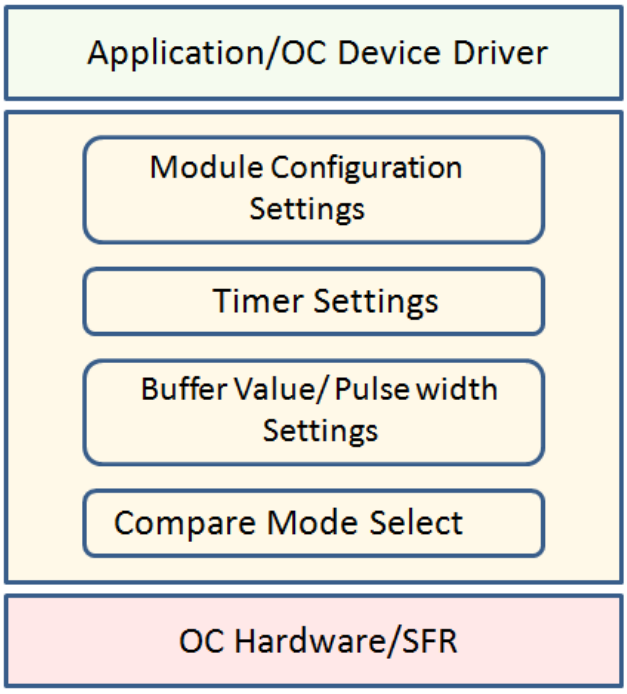
This library provides the low-level abstraction of the Output Compare module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all the functionality of the available Output Compare modules on the device. Refer to the **"Output Compare"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each Output Compare module on your device.

Description

The Output Compare Peripheral Library is used to simplify low-level access to the Output Compare module without having the need to directly interact the module's registers. The function names are generic and lead to easier access to the Output Compare module on different devices.

Output Compare module Software Abstraction Block Diagram



The Output Compare module can be described from a software standpoint as having functions to configure the module itself, to select the timer, set the buffer and pulse width values, and select the mode of operation. The desired nature of the output pulse is thus obtained.

A 16-bit or a 32-bit timer can be selected for the output compare time base. In some devices, the Output Compare module can be synchronized to an external input source. The Output Compare module can also operate asynchronously, based on a trigger by an input source. The Output Compare module entails the use of some functions dealing with Timer set up. These details can be found in Timer Peripheral Library.

Functions have been provided to set buffer values and pulse-width values. Pulse-width values are used only in Dual compare and PWM modes. In PWM mode, the buffer value provides the initial duty cycle for the first time period, while all later time cycles employ the 'pulse-width' value as the duty cycle value.

Compare modes:

- Single Compare Level Mode: Sets the output of Output Compare module at either 'High' or 'Low' at a compare match event
- Single Compare Toggle Mode: Toggles the output of OC module at each compare match event. This mode will produce continuous pulses.

- Dual Compare Mode: Output Compare module output is driven 'High' on compare match with buffer value and driven 'Low' on a compare match with the pulse-width value. The output can be either a single pulse or a continuous stream of pulses.
- PWM Mode: In Edge-aligned PWM mode, the Output Compare module output is driven 'High' whenever the timer resets, and is driven 'Low' on a compare match with the pulse-width value. Faults may or may not be enabled. In Center-aligned PWM mode, the Output Compare module output is driven 'High' on a compare match with the buffer value and driven 'Low' on a compare match with the pulse-width value.

Note: In PWM mode, it is vital that the user selects the mode of operation before selecting a Fault input.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Output Compare module.

Library Interface Section	Description
General Setup Functions	Provides setup, configuration and status control interface routines.
Compare Mode Functions	Provides Compare mode interface routines.
Timer Functions	Provides Timer interface routines.
Power-Saving Modes Functions	Provides Power-Saving modes interface routines.
Faults Functions	Provides Faults interface routines.
Feature Existence Functions	Provides interface routines that determine whether or not certain features are available.

How the Library Works

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

All of the interface functions in this library are classified according to:

- General Setup: These routines deal with general setup of Output Compare module
- Compare/PWM Mode: These routines facilitate the selection of a Compare/PWM mode and setting up the compare values
- Timer: These functions deal with configuring the timer, selection of the clock source, and setting up a synchronous or asynchronous mode of operation for the OC module
- Power-Saving Modes:
- Sleep Mode: The Output Compare module output is driven to the same state as it was in prior to the device entering Sleep mode. In PWM Fault mode, Fault detection is active. A Fault forces output of the OC module to tri-state or to a fixed predetermined state.
- Idle Mode: The Output Compare module can be configured to suspend its operations or continue its operations. If the Output Compare module suspends its operations, it has the same behavior as it does in Sleep mode.
- Faults: These functions are used to select Fault inputs and identify occurrences of Faults when the Output Compare module functions in PWM mode
- Exists: These functions notify whether or not a particular feature is present on a device

The following sections provide examples that depict the use of interface functions to perform general tasks such as initialization and set up of the Output Compare module and setting up the Output Compare module in different compare modes.

Single Compare Set High Mode

The Single Compare Set High mode sets the output of the Output Compare module 'High' at a compare match event. This section illustrates the steps required to configure the Output Compare module in Single Compare Set High mode.

Example:

```
/* This example illustrates setting up Output Compare Module in Single Compare Set High
mode */

/* Configure Timer2 in 16-bit mode. Refer to Timer Peripheral Library for the
API */

/* Disable OC module */
PLIB_OC_Disable(MY_OC_ID);


/* Select Timer2 as time base */
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);
```

```
/* Set period of time base. Refer to Timer Peripheral Library for the API */

/* Select compare mode */
PLIB_OC_ModeSelect(MY_OC_ID, OC_SET_HIGH_SINGLE_PULSE_MODE);

/* Set buffer size to 16-bits */
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_16BIT);

/* Set buffer value */
PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);

/* Configure interrupts associated with Output Compare module. Refer to
Timer Peripheral Library for the API */

/* Enable Timer2. Refer to Timer Peripheral Library for the API */

/* Enable the Output Compare module */
PLIB_OC_Enable(MY_OC_ID);
```

Single Compare Toggle Mode

The Single Compare Toggle mode toggles the output of the Output Compare module at each compare match event. This section illustrates the steps required to configure the Output Compare module in Single Compare Toggle mode.

Example:

```
/* This example illustrates setting up Output Compare Module in Single Compare Toggle mode */

/* Configure Timer2 in 16-bit mode. Refer to Timer Peripheral Library for
the API */

/* Disable OC module */
PLIB_OC_Disable(MY_OC_ID);

/* Select Timer2 as time base */
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);

/* Set period of time base. Refer to Timer Peripheral Library for the API */

/* Select compare mode */
PLIB_OC_ModeSelect(MY_OC_ID, OC_TOGGLE_CONTINOUS_PULSE_MODE);

/* Set buffer size to 16-bits */
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_16BIT);

/* Set Buffer Value */
PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);

/* Configure interrupts associated with Output Compare module. Refer to
Interrupts Peripheral Library for the API */

/* Enable Timer2. Refer to Timer Peripheral Library for the API */

/* Enable Output Compare module */
PLIB_OC_Enable(MY_OC_ID);
```

Dual Compare Continuous Mode

In this mode, the Output Compare module output is driven 'High' on a compare match with the buffer value and driven 'Low' on a compare match with the pulse-width value. A continuous stream of pulses is generated. This section illustrates the steps required to configure the Output Compare module in Dual Compare Continuous Pulse mode.

Example:

```
/* This example illustrates setting up Output Compare Module in Dual Compare Continuous Pulse mode */

/* Configure Timer2 in 16-bit mode. Refer to Timer Peripheral Library for
the API */

/* Disable OC module */
PLIB_OC_Disable(MY_OC_ID);
```

```
/* Select Timer2 as time base */
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);

/* Set period of time base. Refer to Timer Peripheral Library for the API */

/* Select compare mode */
PLIB_OC_ModeSelect(MY_OC_ID, OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE);

/* Set buffer size to 16-bits */
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_16BIT);

/* Set buffer(primary compare) value */
PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);

/*Set pulse width(secondary compare) value */
PLIB_OC_PulseWidth16BitSet(MY_OC_ID, 0x0FFF);

/* Configure interrupts associated with Output Compare module. Refer to
Interrupts Peripheral Library for the API */

/* Enable Timer2. Refer to Timer Peripheral Library for the API */

/* Enable the Output Compare module */
PLIB_OC_Enable(MY_OC_ID);
```

PWM Mode with Enabled Faults

This section illustrates the steps required to configure the Output Compare module in PWM mode.

Note: In PWM mode, it is vital that the user selects the mode of operation before selecting a Fault input.

Example:

This section illustrates the steps required to configure the Output Compare module in PWM mode with Fault protection. This mode can be selected using PLIB_OC_ModeSelect or PLIB_OC_FaultInputSelect.

```
/* Configure Timer2 in 16-bit mode. Refer to Timer Peripheral Library for the API */

/* Disable OC module */
PLIB_OC_Disable(MY_OC_ID);

/* Select Timer2 as time base */
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);

/* Set period of time base. Refer to Timer Peripheral Library for the API */

/* Select compare mode. PWM with fault protection mode is selected ,
   fault selection is preset in the hardware*/

PLIB_OC_ModeSelect(MY_OC_ID, OC_COMPARE_PWM_MODE_WITH_FAULT_PROTECTION );
/*or use PLIB_OC_FaultInputSelect(MY_OC_ID, OC_FAULT_PRESET) to achieve the same*/

/* Set buffer size to 16-bits. Refer to Timer Peripheral Library for the API */
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_16BIT);

/* Set buffer (initial duty cycle) Value */
PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);

/*Set pulse width (Duty cycle) value */
PLIB_OC_PulseWidth16BitSet(MY_OC_ID, 0x0FFF);

/* Configure interrupts associated with Output Compare module. Refer to
Interrupts Peripheral Library for the API */

/* Enable Timer2. Refer to Timer Peripheral Library for the API */

/* Enable OC module */
PLIB_OC_Enable(MY_OC_ID);
```

```
/* Check for PWM Fault */
while(!PLIB_OC_FaultHasOccurred(MY_OC_ID))
{
    /* If no PWM fault, continue normal operation*/
}
```

Configuring the Library

The library is configured for the supported Output Compare module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Setup Functions

	Name	Description
	PLIB_OC_Disable	Disable the Output Compare module.
	PLIB_OC_Enable	Enables the Output Compare module.
	PLIB_OC_IsEnabled	Checks whether the Output Compare module is enabled or not.

b) Compare Mode Functions

	Name	Description
	PLIB_OC_ModeSelect	Selects the compare mode for the Output Compare module.
	PLIB_OC_Buffer16BitSet	Sets a 16-bit primary compare value for compare operations.
	PLIB_OC_Buffer32BitSet	Sets a 32-bit primary compare value for compare operations.
	PLIB_OC_BufferSizeSelect	Sets the buffer size and pulse width to 16-bits or 32-bits.
	PLIB_OC_PulseWidth16BitSet	Sets a 16-bit pulse width for Output Compare module output.
	PLIB_OC_PulseWidth32BitSet	Sets a 32-bit pulse width for Output Compare module output.

c) Timer Functions

	Name	Description
	PLIB_OC_TimerSelect	Selects a clock source for the Output Compare module.
	PLIB_OC_AlternateClockDisable	Selects Timer 2/3, instead of the alternate clock source.
	PLIB_OC_AlternateClockEnable	Selects the alternate clock source.
	PLIB_OC_AlternateTimerSelect	Selects an alternate timer as a clock source for the Output Compare module.

d) Power-Saving Modes Functions

	Name	Description
	PLIB_OC_StopInIdleDisable	Output Compare module continues operating when the device enters Idle mode.
	PLIB_OC_StopInIdleEnable	Discontinues Output Compare module operation when the device enters Idle mode.

e) Faults Functions

	Name	Description
	PLIB_OC_FaultHasOccurred	Checks if a PWM fault has occurred.
	PLIB_OC_FaultInputSelect	Enables/Disables the Fault input for the Output Compare PWM mode.

f) Feature Existence Functions

	Name	Description
	PLIB_OC_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the Output Compare module.
	PLIB_OC_ExistsBufferSize	Identifies whether the BufferSize feature exists on the Output Compare module.
	PLIB_OC_ExistsBufferValue	Identifies whether the BufferValue feature exists on the Output Compare module.
	PLIB_OC_ExistsCompareModeSelect	Identifies whether the CompareModeSelect feature exists on the Output Compare module.
	PLIB_OC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Output Compare module.
	PLIB_OC_ExistsFaultInput	Identifies whether the FaultInput feature exists on the Output Compare module.
	PLIB_OC_ExistsFaultStatus	Identifies whether the FaultStatus feature exists on the Output Compare module.
	PLIB_OC_ExistsPulseWidth	Identifies whether the PulseWidth feature exists on the Output Compare module.
	PLIB_OC_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the Output Compare module.
	PLIB_OC_ExistsTimerSelect	Identifies whether the TimerSelect feature exists on the Output Compare module.
	PLIB_OC_ExistsAlternateTimerSelect	Identifies whether the AlternateTimerSelect feature exists on the Output Compare module.

g) Data Types and Constants

	Name	Description
	OC_16BIT_TIMERS	Lists different 16 bit time bases for Output Compare module.
	OC_BUFFER_SIZE	Lists different buffer sizes for compare value.
	OC_COMPARE_MODES	Lists the different compare modes for the Output Compare module.
	OC_FAULTS	Lists different fault sources for Output Compare module
	OC_MODULE_ID	OC_MODULE_ID This enumeration defines number of modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers. Refer to the specific device data sheet to determine the correct number of modules defined for the desired microcontroller.
	OC_ALT_TIMERS	Lists the different 16-bit alternate timers for the Output Compare module.

Description

This section describes the Application Programming Interface (API) functions of the Output Compare Peripheral Library.

Refer to each section for a detailed description.

a) General Setup Functions

PLIB_OC_Disable Function

Disable the Output Compare module.

File

plib_oc.h

C

```
void PLIB_OC_Disable(OC_MODULE_ID index);
```

Returns

None.

Description

This function disables the Output Compare module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_Disable(MY_OC_ID);
```

Parameters

Parameters	Description
index	Identifies the Output Compare module

Function

void PLIB_OC_Disable(OC_MODULE_ID index)

PLIB_OC_Enable Function

Enables the Output Compare module.

File

plib_oc.h

C

```
void PLIB_OC_Enable(OC_MODULE_ID index);
```

Returns

None.

Description

This function enables the Output Compare module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

The module should be appropriately configured before being enabled.

Example

```
#define MY_OC_ID OC_ID_1

//Do all the other configurations before enabling.

PLIB_OC_Enable(MY_OC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

void PLIB_OC_Enable(OC_MODULE_ID index)

PLIB_OC_IsEnabled Function

Checks whether the Output Compare module is enabled or not.

File

plib_oc.h

C

```
bool PLIB_OC_IsEnabled(OC_MODULE_ID index);
```

Returns

Boolean

- true - The Output Compare module is enabled
- false - The Output Compare module is not enabled

Description

The function returns the enable status of the Output Compare module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_OC_ID OC_ID_1

if(PLIB_OC_IsEnabled(MY_OC_ID))
```

```
{
//Take respective actions
}
else
{
//Take respective actions
}
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

bool PLIB_OC_IsEnabled(OC_MODULE_ID index)

b) Compare Mode Functions

PLIB_OC_ModeSelect Function

Selects the compare mode for the Output Compare module.

File

plib_oc.h

C

```
void PLIB_OC_ModeSelect(OC_MODULE_ID index, OC_COMPARE_MODES cmpMode);
```

Returns

None.

Description

This function selects the compare mode for the Output Compare module.

Remarks

If PLIB_OC_FaultInputSelect is called after PLIB_OC_ModeSelect, the mode selected by the PLIB_OC_ModeSelect function will be overwritten as the PLIB_OC_FaultInputSelect function selects the PWM mode with or without Fault protection.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsCompareModeSelect in your application to determine whether this feature is available.

Preconditions

The Output Compare module must be turned off before a new mode is selected. The Output Compare module is turned off through the PLIB_OC_ModeSelect(MY_OC_ID,OC_COMPARE_TURN_OFF_MODE) function. Refer to the enumeration description for information on different modes and preconditions.

Example

```
#define MY_OC_ID OC_ID_1
// Dual compare continuous pulse mode is selected
PLIB_OC_ModeSelect(MY_OC_ID, OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE );
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
cmpMode	Identifies the compare mode for Output Compare module

Function

void PLIB_OC_ModeSelect(OC_MODULE_ID index, OC_COMPARE_MODES cmpMode)

PLIB_OC_Buffer16BitSet Function

Sets a 16-bit primary compare value for compare operations.

File

plib_oc.h

C

```
void PLIB_OC_Buffer16BitSet(OC_MODULE_ID index, uint16_t val16Bit);
```

Returns

None.

Description

This function sets a 16-bit primary compare value for compare operations in all modes except PWM modes.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsBufferValue in your application to determine whether this feature is available.

Preconditions

The PWM mode of operation should not be selected. The buffer size should be set to 16-bits by the PLIB_OC_BufferSizeSelect function.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_Buffer16BitSet(MY_OC_ID, 0x00FF);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
val16Bit	Sets a 16-bit primary compare value

Function

void PLIB_OC_Buffer16BitSet(OC_MODULE_ID index, uint16_t val16Bit)

PLIB_OC_Buffer32BitSet Function

Sets a 32-bit primary compare value for compare operations.

File

plib_oc.h

C

```
void PLIB_OC_Buffer32BitSet(OC_MODULE_ID index, uint32_t val32Bit);
```

Returns

None.

Description

This function sets a 32-bit primary compare value for compare operations in all modes except PWM modes.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsBufferValue in your application to determine whether this feature is available.

Preconditions

The PWM mode of operation should not be selected. The buffer size should be set to 32-bits by the PLIB_OC_BufferSizeSelect function.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_Buffer32BitSet(MY_OC_ID, 0x000000FF);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
val32Bit	Sets a 32-bit primary compare value

Function

void PLIB_OC_Buffer32BitSet(OC_MODULE_ID index, uint32_t val32Bit)

PLIB_OC_BufferSizeSelect Function

Sets the buffer size and pulse width to 16-bits or 32-bits.

File

plib_oc.h

C

```
void PLIB_OC_BufferSizeSelect(OC_MODULE_ID index, OC_BUFFER_SIZE size);
```

Returns

None.

Description

This function sets the size of the buffer and pulse width to 16-bits or 32-bits. The choice is made based on whether a 16-bit timer or a 32-bit timer is selected.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsBufferSize in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_OC_ID OC_ID_1
// Buffer size and pulse width size are set to 32-bits
PLIB_OC_BufferSizeSelect(MY_OC_ID, OC_BUFFER_SIZE_32BIT);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
size	Identifies the size of compare value

Function

void PLIB_OC_BufferSizeSelect(OC_MODULE_ID index, OC_BUFFER_SIZE size)

PLIB_OC_PulseWidth16BitSet Function

Sets a 16-bit pulse width for Output Compare module output.

File

plib_oc.h

C

```
void PLIB_OC_PulseWidth16BitSet(OC_MODULE_ID index, uint16_t pulseWidth);
```

Returns

None.

Description

This function sets a 16-bit pulse width for the Output Compare module in dual compare modes. A dual compare mode can be selected using the PLIB_OC_ModeSelect function. Secondary compare match event (pulse width) decides the trailing (falling) edge of the Output Compare module

output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsPulseWidth in your application to determine whether this feature is available.

Preconditions

Dual compare operation should be selected. The buffer size should be set to 16-bits by the PLIB_OC_BufferSizeSelect function.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_PulseWidth16BitSet(MY_OC_ID, 0x0FFF);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
pulseWidth	Pulse width value

Function

void PLIB_OC_PulseWidth16BitSet(OC_MODULE_ID index,uint16_t pulseWidth)

PLIB_OC_PulseWidth32BitSet Function

Sets a 32-bit pulse width for Output Compare module output.

File

plib_oc.h

C

```
void PLIB_OC_PulseWidth32BitSet(OC_MODULE_ID index, uint32_t pulseWidth);
```

Returns

None.

Description

This function sets a 32-bit pulse width for Output Compare module in dual compare modes. A dual compare mode can be selected using PLIB_OC_ModeSelect function. Secondary compare match event (pulse width) decides the trailing (falling) edge of the Output Compare module output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsPulseWidth in your application to determine whether this feature is available.

Preconditions

Dual compare operation should be selected. The buffer size should be set to 32-bits by the PLIB_OC_BufferSizeSelect function.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_PulseWidth32BitSet(MY_OC_ID, 0x00000FFF);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
pulseWidth	Pulse width value

Function

void PLIB_OC_PulseWidth32BitSet(OC_MODULE_ID index,uint32_t pulseWidth)

c) Timer Functions

PLIB_OC_TimerSelect Function

Selects a clock source for the Output Compare module.

File

plib_oc.h

C

```
void PLIB_OC_TimerSelect(OC_MODULE_ID index, OC_16BIT_TIMERS tmr);
```

Returns

None.

Description

This function selects a clock source for the Output Compare module if the 16-bit time base is set.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsTimerSelect in your application to determine whether this feature is available.

Preconditions

The 16-bit time base needs to be set.

Example

```
#define MY_OC_ID OC_ID_1
// 16-bit Timer2 is selected
PLIB_OC_TimerSelect(MY_OC_ID, OC_TIMER_16BIT_TMR2);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
tmr	Identifies the timer

Function

void PLIB_OC_TimerSelect(OC_MODULE_ID index, OC_16BIT_TIMERS tmr)

PLIB_OC_AlternateClockDisable Function

Selects Timer 2/3, instead of the alternate clock source.

File

plib_oc.h

C

```
void PLIB_OC_AlternateClockDisable(OC_MODULE_ID index);
```

Returns

None.

Description

This function selects Timer2/Timer3, instead of the alternate clock source.

Remarks

The feature is not supported on all devices. Please refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed. This function applies to all input capture modules, regardless of the OC_MODULE_ID passed in the call.

Preconditions

A system unlock PLIB_DEVCON_SystemUnlock must be performed before this function can be executed.

Example

```
// Call system service to unlock oscillator
#define MY_OC_ID OC_ID_1
PLIB_OC_AlternateClockDisable( MY_OC_ID );
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

void PLIB_OC_AlternateClockDisable(OC_MODULE_ID index)

PLIB_OC_AlternateClockEnable Function

Selects the alternate clock source.

File

plib_oc.h

C

```
void PLIB_OC_AlternateClockEnable(OC_MODULE_ID index);
```

Returns

None.

Description

This function selects the alternate clock source instead of Timer2/Timer3.

Remarks

The feature is not supported on all devices. Please refer to the specific device data sheet to determine availability. A system unlock must be performed before this function can be executed. This function applies to all input capture modules, regardless of the OC_MODULE_ID passed in the call.

Preconditions

A system unlock PLIB_DEVCON_SystemUnlock must be performed before this function can be executed.

Example

```
// Call system service to unlock oscillator
#define MY_OC_ID OC_ID_1
PLIB_OC_AlternateClockEnable( MY_OC_ID );
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

void PLIB_OC_AlternateClockEnable(OC_MODULE_ID index)

PLIB_OC_AlternateTimerSelect Function

Selects an alternate timer as a clock source for the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_AlternateTimerSelect(OC_MODULE_ID index, OC_ALT_TIMERS tmr);
```

Returns

- true - Alternate timer selected successfully
- false - Alternate timer selection failure, select an appropriate alternate timer for the Output Compare module index

Description

This function selects an alternate timer as a clock source for the Output Compare module.

- OC_ID_1,OC_ID_2,OC_ID_3: Can use Timer4 or Timer5 as alternate timers
- OC_ID_4,OC_ID_5,OC_ID_6: Can use Timer2 or Timer3 as alternate timers
- OC_ID_7,OC_ID_8,OC_ID_9: Can use Timer6 or Timer7 as alternate timers

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsAlternateTimerSelect in your application to determine whether this feature is available.

Preconditions

The 16-bit time base needs to be set. The PLIB_OC_AlternateClockEnable function should be called for the Output Compare module to enable the alternate clock selection.

Example

```
#define MY_OC_ID   OC_ID_1
bool result;

//Enabling alternate timer selection
PLIB_OC_AlternateClockEnable( MY_OC_ID );

// 16-bit Timer4 is selected as the clock source for Output Compare module 1
result = PLIB_OC_AlternateTimerSelect(MY_OC_ID, OC_ALT_TIMER_TMR4);

if(false == result)
{
    // Selected alternate timer does not available for the desired Output
    // Compare module.
    // Select the appropriate alternate timer.
}
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
tmr	Identifies the alternate timer

Function

bool PLIB_OC_AlternateTimerSelect(OC_MODULE_ID index, OC_ALT_TIMERS tmr)

d) Power-Saving Modes Functions

PLIB_OC_StopInIdleDisable Function

Output Compare module continues operating when the device enters Idle mode.

File

plib_oc.h

C

```
void PLIB_OC_StopInIdleDisable(OC_MODULE_ID index);
```

Returns

None.

Description

The function continues Output Compare module operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_StopInIdleDisable(MY_OC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

void PLIB_OC_StopInIdleDisable(OC_MODULE_ID index)

PLIB_OC_StopInIdleEnable Function

Discontinues Output Compare module operation when the device enters Idle mode.

File

plib_oc.h

C

```
void PLIB_OC_StopInIdleEnable(OC_MODULE_ID index);
```

Returns

None.

Description

This function discontinues Output Compare module operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_OC_ID OC_ID_1

PLIB_OC_StopInIdleEnable(MY_OC_ID);
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

void PLIB_OC_StopInIdleEnable(OC_MODULE_ID index)

e) Faults Functions

PLIB_OC_FaultHasOccurred Function

Checks if a PWM fault has occurred.

File

plib_oc.h

C

```
bool PLIB_OC_FaultHasOccurred(OC_MODULE_ID index);
```

Returns

- true - PWM Fault has occurred
- false - No PWM fault has occurred

Description

This function returns 'true' if a PWM Fault has occurred and 'false' if no Fault condition exists.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsFaultStatus in your application to determine whether this feature is available.

Preconditions

This function should be used only in Edge or Center-Aligned PWM mode set by the PLIB_OC_ModeSelect() function.

Example

```
#define MY_OC_ID OC_ID_1
if(!PLIB_OC_FaultHasOccurred(MY_OC_ID))
{
    // Do some operation
};
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module

Function

bool PLIB_OC_FaultHasOccurred(OC_MODULE_ID index)

PLIB_OC_FaultInputSelect Function

Enables/Disables the Fault input for the Output Compare PWM mode.

File

plib_oc.h

C

```
void PLIB_OC_FaultInputSelect(OC_MODULE_ID index, OC_FAULTS flt);
```

Returns

None.

Description

This function enables/disables the Fault input for the Output Compare PWM mode.

If some other mode was selected using PLIB_OC_ModeSelect, the mode selected by PLIB_OC_ModeSelect will be overwritten as PLIB_OC_FaultInputSelect selects PWM mode with/without Fault protection.

Fault input is valid if the fault pin is enabled in the hardware. If a logic '0' is detected on the OCFA/OCFB pin, the selected PWM output pin(s) are placed in the tri-state. The user may elect to provide a pull-down or pull-up resistor on the PWM pin to provide for a desired state if a Fault condition occurs. The shutdown of the PWM output is immediate and is not tied to the device clock source. Fault occurrence can be detected by calling the function PLIB_OC_FaultHasOccurred. The Output Compare module will be disabled until the following conditions are met: • The external Fault condition has been removed • The PWM mode is re-enabled

Remarks

This function selects the PWM mode of the Output Compare module with Fault protection or without Fault protection.

These modes can be selected using PLIB_OC_ModeSelect also.

If any other Output Compare mode is selected prior to this function, that mode will be overwritten as this feature is available for PWM mode only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OC_ExistsFaultInput in your application to determine whether this feature is available.

Preconditions

Fault pin: OCFA for OC_ID_1 to OC_ID_4 , OCFB for OC_ID_5 in MX devices. OCFA for OC_ID_1 to OC_ID_3 and OC_ID_7 to OC_ID_9 , OCFB for OC_ID_4 to OC_ID_6 in MZ devices. should be enabled in the hardware if enabling the fault input, that is if selecting the OC_FAULT_PRESET.

Example

```
#define MY_OC_ID OC_ID_1
// Fault pin is enabled in the hardware
// This function selects PWM with fault protection mode for MY_OC_ID instance.
PLIB_OC_FaultInputSelect(MY_OC_ID, OC_FAULT_PRESET);
// Output Compare fault input is now enabled for Output Compare Module
```

Parameters

Parameters	Description
index	Identifies the desired Output Compare module
flt	Identifies the Output Compare module Fault input

Function

void PLIB_OC_FaultInputSelect(OC_MODULE_ID index, OC_FAULTS flt)

f) Feature Existence Functions

PLIB_OC_ExistsAlternateClock Function

Identifies whether the AlternateClock feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsAlternateClock(OC_MODULE_ID index);
```

Returns

- true - The AlternateClock feature is supported on the device
- false - The AlternateClock feature is not supported on the device

Description

This function identifies whether the AlternateClock feature is available on the Output Compare module. When this function returns true, these functions are supported on the device:

- PLIB_OC_AlternateClockEnable
- PLIB_OC_AlternateClockDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OC_ExistsAlternateClock(OC_MODULE_ID index)

PLIB_OC_ExistsBufferSize Function

Identifies whether the BufferSize feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsBufferSize(OC_MODULE_ID index);
```

Returns

- true - If the BufferSize feature is supported on the device
- false - If the BufferSize feature is not supported on the device

Description

This function identifies whether the BufferSize feature is available on the Output Compare module. When this function returns true, this function is supported on the device:

- PLIB_OC_BufferSizeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsBufferSize(OC_MODULE_ID index)

PLIB_OC_ExistsBufferValue Function

Identifies whether the BufferValue feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsBufferValue(OC_MODULE_ID index);
```

Returns

- true - If the BufferValue feature is supported on the device
- false - If the BufferValue feature is not supported on the device

Description

This function identifies whether the BufferValue feature is available on the Output Compare module. When this function returns true, these functions are supported on the device:

- PLIB_OC_Buffer32BitSet
- PLIB_OC_Buffer16BitSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsBufferValue(OC_MODULE_ID index)

PLIB_OC_ExistsCompareModeSelect Function

Identifies whether the CompareModeSelect feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsCompareModeSelect(OC_MODULE_ID index);
```

Returns

- true - If the CompareModeSelect feature is supported on the device
- false - If the CompareModeSelect feature is not supported on the device

Description

This function identifies whether the CompareModeSelect feature is available on the Output Compare module. When this function returns true, this function is supported on the device:

- PLIB_OC_ModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsCompareModeSelect(OC_MODULE_ID index)

PLIB_OC_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsEnableControl(OC_MODULE_ID index);
```

Returns

- true - If the EnableControl feature is supported on the device
- false - If the EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the Output Compare module. When this function returns true, these functions are supported on the device:

- PLIB_OC_Enable
- PLIB_OC_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsEnableControl(OC_MODULE_ID index)

PLIB_OC_ExistsFaultInput Function

Identifies whether the FaultInput feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsFaultInput(OC_MODULE_ID index);
```

Returns

- true - If the FaultInput feature is supported on the device
- false - If the FaultInput feature is not supported on the device

Description

This function identifies whether the FaultInput feature is available on the Output Compare module. When this function returns true, this function is supported on the device:

- PLIB_OC_FaultInputSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsFaultInput(OC_MODULE_ID index)

PLIB_OC_ExistsFaultStatus Function

Identifies whether the FaultStatus feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsFaultStatus(OC_MODULE_ID index);
```

Returns

- true - If the FaultStatus feature is supported on the device
- false - If the FaultStatus feature is not supported on the device

Description

This function identifies whether the FaultStatus feature is available on the Output Compare module. When this function returns true, this function is supported on the device:

- PLIB_OC_FaultHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsFaultStatus(OC_MODULE_ID index)

PLIB_OC_ExistsPulseWidth Function

Identifies whether the PulseWidth feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsPulseWidth(OC_MODULE_ID index);
```

Returns

- true - If the PulseWidth feature is supported on the device
- false - If the PulseWidth feature is not supported on the device

Description

This function identifies whether the PulseWidth feature is available on the Output Compare module. When this function returns true, these functions are supported on the device:

- PLIB_OC_PulseWidth32BitSet
- PLIB_OC_PulseWidth16BitSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsPulseWidth(OC_MODULE_ID index)

PLIB_OC_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsStopInIdle(OC_MODULE_ID index);
```

Returns

- true - If the StopInIdle feature is supported on the device
- false - If the StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the Output Compare module. When this function returns true, these functions are supported on the device:

- PLIB_OC_StopInIdleEnable
- PLIB_OC_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsStopInIdle(OC_MODULE_ID index)

PLIB_OC_ExistsTimerSelect Function

Identifies whether the TimerSelect feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsTimerSelect(OC_MODULE_ID index);
```

Returns

- true - If the TimerSelect feature is supported on the device
- false - If the TimerSelect feature is not supported on the device

Description

This function identifies whether the TimerSelect feature is available on the Output Compare module. When this function returns true, this function is supported on the device:

- PLIB_OC_TimerSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsTimerSelect(OC_MODULE_ID index)

PLIB_OC_ExistsAlternateTimerSelect Function

Identifies whether the AlternateTimerSelect feature exists on the Output Compare module.

File

plib_oc.h

C

```
bool PLIB_OC_ExistsAlternateTimerSelect(OC_MODULE_ID index);
```

Returns

- true - If the AlternateTimerSelect feature is supported on the device
- false - If the AlternateTimerSelect feature is not supported on the device

Description

This function identifies whether the AlternateTimerSelect feature is available on the Output Compare module. When this function returns true, this function is supported on the device:

- PLIB_OC_AlternateTimerSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_OC_ExistsAlternateTimerSelect(OC_MODULE_ID index)

g) Data Types and Constants

OC_16BIT_TIMERS Enumeration

Lists different 16 bit time bases for Output Compare module.

File

plib_oc_help.h

C

```
typedef enum {
  OC_TIMER_16BIT_TMR2,
  OC_TIMER_16BIT_TMR3
} OC_16BIT_TIMERS;
```

Members

Members	Description
OC_TIMER_16BIT_TMR2	Clock source of Timer2 is the clock source of Output Compare module
OC_TIMER_16BIT_TMR3	Clock source of Timer3 is the clock source of Output Compare module

Description

Output Compare 16-bit Timer Select

This enumeration lists different 16 bit time bases for Output Compare module. The time base is set by the PLIB_OC_TimerSelect function.

OC_BUFFER_SIZE Enumeration

Lists different buffer sizes for compare value.

File

plib_oc_help.h

C

```
typedef enum {
  OC_BUFFER_SIZE_16BIT,
  OC_BUFFER_SIZE_32BIT
} OC_BUFFER_SIZE;
```

Members

Members	Description
OC_BUFFER_SIZE_16BIT	Buffer size is 16-bits. Duty cycle and compare values will have 16-bit values.
OC_BUFFER_SIZE_32BIT	Buffer size is 32-bits. Duty cycle and compare values will have 32-bit values.

Description

Output Compare Buffer Size

This enumeration lists different buffer sizes for compare value and duty cycle value.

OC_COMPARE_MODES Enumeration

Lists the different compare modes for the Output Compare module.

File

plib_oc_help.h

C

```
typedef enum {
  OC_COMPARE_PWM_MODE_WITH_FAULT_PROTECTION,
  OC_COMPARE_PWM_MODE_WITHOUT_FAULT_PROTECTION,
  OC_COMPARE_PWM_EDGE_ALIGNED_MODE,
  OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE,
  OC_DUAL_COMPARE_SINGLE_PULSE_MODE,
  OC_TOGGLE_CONTINUOUS_PULSE_MODE,
  OC_SET_LOW_SINGLE_PULSE_MODE,
  OC_SET_HIGH_SINGLE_PULSE_MODE,
  OC_COMPARE_TURN_OFF_MODE
} OC_COMPARE_MODES;
```

Members

Members	Description
OC_COMPARE_PWM_MODE_WITH_FAULT_PROTECTION	Output Compare module output is PWM signal and is fault protected if fault protection pin is enabled. Fault protection is valid if the fault pin is enabled in the hardware. Fault pin: OCFA for OC_ID_1 to OC_ID_4 , OCFB for OC_ID_5 in MX devices. OCFA for OC_ID_1 to OC_ID_3 and OC_ID_7 to OC_ID_9 , OCFB for OC_ID_4 to OC_ID_6 in PIC32MZ devices. If a logic â€˜0â€™ is detected on the OCFA/OCFB pin, the selected PWM output pin(s) are placed in the tri-state. The user may elect to provide a pull-down or pull-up resistor on the PWM pin to provide for a desired state if a Fault condition occurs. The shutdown of the PWM output is immediate and is not tied to the device clock source. Fault occurrence can be detected by calling the function PLIB_OC_FaultHasOccurred. The Output Compare will be disabled until the following conditions are met: 1. The external Fault condition has been removed 2. The PWM mode is re-enabled
OC_COMPARE_PWM_MODE_WITHOUT_FAULT_PROTECTION	Output Compare module output is PWM signal and is not fault protected
OC_COMPARE_PWM_EDGE_ALIGNED_MODE	This element is obsolete and it will be removed from next release
OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE	Dual Compare, Continuous Pulse mode: Output Compare module output is driven high on compare match with primary compare value and driven low on compare match with secondary compare value. A continuous stream of pulses is generated unless the compare mode is changed or the module is disabled. If the secondary compare value is greater than time base period value, secondary compare match does not occur. As a consequence, Output Compare module output stays high.
OC_DUAL_COMPARE_SINGLE_PULSE_MODE	Dual Compare, Single Pulse mode: Output Compare module output is driven high on compare match with primary compare value and driven low on compare match with secondary compare value. If the secondary compare value is greater than time base period value, secondary compare match does not occur. As a consequence, Output Compare module output stays high until a mode change is made or module is disabled
OC_TOGGLE_CONTINUOUS_PULSE_MODE	Single Compare Toggle mode: Output Compare module output is initialized to Low. Output Compare module output toggles at every compare match event with primary compare value with a single peripheral bus clock cycle delay. This scheme generates a square wave with 50% duty cycle. An interrupt is generated each time the output toggles.
OC_SET_LOW_SINGLE_PULSE_MODE	Single Compare Set Low mode: A compare match event with primary compare value will set the output of Output Compare module 'Low' with a single peripheral bus clock cycle delay. Output stays Low unless Output Compare module is disabled or a new compare mode is chosen. An interrupt is generated at compare match event. Output Compare module output is initially forced High.

OC_SET_HIGH_SINGLE_PULSE_MODE	Single Compare Set High mode: A compare match event with primary compare value will set the output of Output Compare module 'High' with a single peripheral bus clock cycle delay. Output stays High unless Output Compare module is disabled or a new compare mode is chosen. An interrupt is generated at compare match event. Output Compare module output is initially forced Low.
OC_COMPARE_TURN_OFF_MODE	Turn OFF mode: Output Compare module is disabled but still draws current. This mode is used to temporarily turn OFF the Output Compare module before a new compare mode is selected

Description

Output Compare Compare Modes

This enumeration lists different compare modes for Output Compare module. The compare mode is set by the PLIB_OC_ModeSelect function.

OC_FAULTS Enumeration

Lists different fault sources for Output Compare module

File

plib_oc_help.h

C

```
typedef enum {
  OC_FAULT_PRESET,
  OC_FAULT_DISABLE
} OC_FAULTS;
```

Members

Members	Description
OC_FAULT_PRESET	Enable Fault protection. PWM operation with fault protection.
OC_FAULT_DISABLE	Disable Fault protection. PWM operation without faults.

Description

Output Compare Fault Select

This enumeration lists different fault sources for Output Compare module. The fault source is selected by the PLIB_OC_FaultInputSelect function.

OC_MODULE_ID Enumeration

File

plib_oc_help.h

C

```
typedef enum {
  OC_ID_1,
  OC_ID_2,
  OC_ID_3,
  OC_ID_4,
  OC_ID_5,
  OC_ID_6,
  OC_ID_7,
  OC_ID_8,
  OC_ID_9,
  OC_NUMBER_OF_MODULES
} OC_MODULE_ID;
```

Members

Members	Description
OC_NUMBER_OF_MODULES	The total number of modules available.

Description

OC_MODULE_ID

This enumeration defines number of modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers.

Refer to the specific device data sheet to determine the correct number of modules defined for the desired microcontroller.

OC_ALT_TIMERS Enumeration

Lists the different 16-bit alternate timers for the Output Compare module.

File

plib_oc_help.h

C

```
typedef enum {
  OC_ALT_TIMER_TMR2,
  OC_ALT_TIMER_TMR3,
  OC_ALT_TIMER_TMR4,
  OC_ALT_TIMER_TMR5,
  OC_ALT_TIMER_TMR6,
  OC_ALT_TIMER_TMR7
} OC_ALT_TIMERS;
```

Members

Members	Description
OC_ALT_TIMER_TMR2	Select Timer2 as the alternate clock source for Output Compare module
OC_ALT_TIMER_TMR3	Select Timer3 as the alternate clock source for Output Compare module
OC_ALT_TIMER_TMR4	Select Timer4 as the alternate clock source for Output Compare module
OC_ALT_TIMER_TMR5	Select Timer5 as the alternate clock source for Output Compare module
OC_ALT_TIMER_TMR6	Select Timer6 as the alternate clock source for Output Compare module
OC_ALT_TIMER_TMR7	Select Timer7 as the alternate clock source for Output Compare module

Description

Output Compare 16-bit alternate Timer Select

This enumeration lists the different 16-bit timers for the Output Compare time base when the Output Compare module is configured with a 16-bit alternate timer resource.

Files

Files

Name	Description
plib_oc.h	This file contains the interface definition for the Output Compare Peripheral Library.
plib_oc_help.h	This is file plib_oc_help.h.

Description

This section lists the source and header files used by the library.

plib_oc.h

This file contains the interface definition for the Output Compare Peripheral Library.

Functions

	Name	Description
	PLIB_OC_AlternateClockDisable	Selects Timer 2/3, instead of the alternate clock source.
	PLIB_OC_AlternateClockEnable	Selects the alternate clock source.
	PLIB_OC_AlternateTimerSelect	Selects an alternate timer as a clock source for the Output Compare module.
	PLIB_OC_Buffer16BitSet	Sets a 16-bit primary compare value for compare operations.
	PLIB_OC_Buffer32BitSet	Sets a 32-bit primary compare value for compare operations.
	PLIB_OC_BufferSizeSelect	Sets the buffer size and pulse width to 16-bits or 32-bits.
	PLIB_OC_Disable	Disable the Output Compare module.
	PLIB_OC_Enable	Enables the Output Compare module.
	PLIB_OC_ExistsAlternateClock	Identifies whether the AlternateClock feature exists on the Output Compare module.
	PLIB_OC_ExistsAlternateTimerSelect	Identifies whether the AlternateTimerSelect feature exists on the Output Compare module.

	PLIB_OC_ExistsBufferSize	Identifies whether the BufferSize feature exists on the Output Compare module.
	PLIB_OC_ExistsBufferValue	Identifies whether the BufferValue feature exists on the Output Compare module.
	PLIB_OC_ExistsCompareModeSelect	Identifies whether the CompareModeSelect feature exists on the Output Compare module.
	PLIB_OC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Output Compare module.
	PLIB_OC_ExistsFaultInput	Identifies whether the FaultInput feature exists on the Output Compare module.
	PLIB_OC_ExistsFaultStatus	Identifies whether the FaultStatus feature exists on the Output Compare module.
	PLIB_OC_ExistsPulseWidth	Identifies whether the PulseWidth feature exists on the Output Compare module.
	PLIB_OC_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the Output Compare module.
	PLIB_OC_ExistsTimerSelect	Identifies whether the TimerSelect feature exists on the Output Compare module.
	PLIB_OC_FaultHasOccurred	Checks if a PWM fault has occurred.
	PLIB_OC_FaultInputSelect	Enables/Disables the Fault input for the Output Compare PWM mode.
	PLIB_OC_IsEnabled	Checks whether the Output Compare module is enabled or not.
	PLIB_OC_ModeSelect	Selects the compare mode for the Output Compare module.
	PLIB_OC_PulseWidth16BitSet	Sets a 16-bit pulse width for Output Compare module output.
	PLIB_OC_PulseWidth32BitSet	Sets a 32-bit pulse width for Output Compare module output.
	PLIB_OC_StopInIdleDisable	Output Compare module continues operating when the device enters Idle mode.
	PLIB_OC_StopInIdleEnable	Discontinues Output Compare module operation when the device enters Idle mode.
	PLIB_OC_TimerSelect	Selects a clock source for the Output Compare module.

Description

Output Compare Module Peripheral Library Interface Header

This library provides a low-level abstraction of the Output Compare module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences between one microcontroller variant and another.

File Name

plib_oc.h

Company

Microchip Technology Inc.

plib_oc_help.h

Enumerations

	Name	Description
	OC_16BIT_TIMERS	Lists different 16 bit time bases for Output Compare module.
	OC_ALT_TIMERS	Lists the different 16-bit alternate timers for the Output Compare module.
	OC_BUFFER_SIZE	Lists different buffer sizes for compare value.
	OC_COMPARE_MODES	Lists the different compare modes for the Output Compare module.
	OC_FAULTS	Lists different fault sources for Output Compare module
	OC_MODULE_ID	OC_MODULE_ID This enumeration defines number of modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on Microchip microcontrollers. Refer to the specific device data sheet to determine the correct number of modules defined for the desired microcontroller.

Description

This is file plib_oc_help.h.

Oscillator Peripheral Library

This section describes the Oscillator Peripheral Library.

Introduction

This library provides a low-level abstraction of the Oscillator module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding

differences from one microcontroller variant to another.

Description

The Oscillator is the heart of the microcontroller and provides the clock on which the core and the peripherals run.

For all of the oscillators present on a Microchip microcontroller, two kinds of configurations exist:

- Through Configuration bits
- At execution time

The first is through 'Configuration bits', which is a one-time configuration done during the programming of the device. These one-time configurations are programmed in the code memory.

The other is 'Execution time' configuration, which deals with features that are allowed to be changed during run-time. This peripheral library provides functions which deal only with the 'execution time' configurable features of the Oscillator module.

Certain Oscillator module features can only be selected through 'Configuration bits'. However, most of the features can be altered during run-time by using the functions provided in this library.

Using the Library

This topic describes the basic architecture of the Oscillator Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_osc.h`

The interface to the Oscillator Peripheral Library is defined in the `plib_osc.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Oscillator Peripheral Library must include `peripheral.h`.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

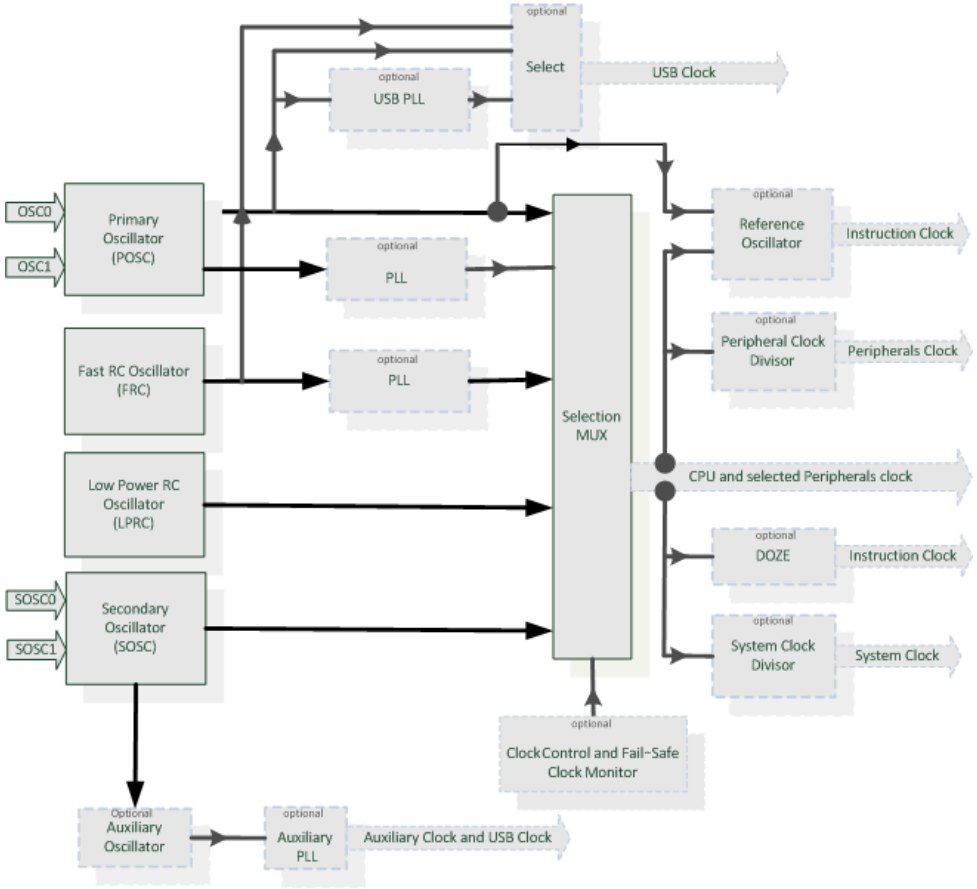
Hardware Abstraction Model

This library provides a low-level abstraction of the Oscillator module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model

The Oscillator hardware model is shown in the following figure. The blocks represented in dashed lines may not be present in all microcontrollers. Refer to the **"Oscillator"** chapter in the specific device data sheet to determine availability.



To understand the Oscillator module and how each feature is mapped in this library, it is important to understand the following terminology.

Clock Source

A clock source is hardware that generates oscillations, which may be internal or external.

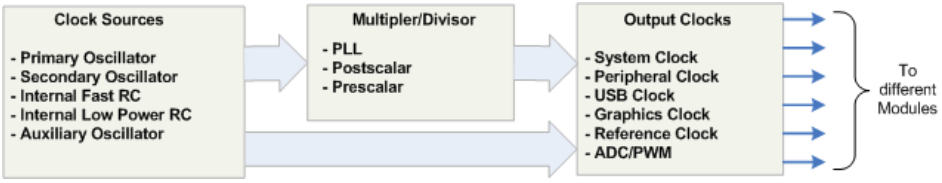
Divisor and Multiplier/PLL

These are hardware modules that can scale the clock. The rate at which the scaling is done may be fixed or configurable.

Clock Outputs

Clock outputs means output lines from the Oscillator module, which may route to different modules of the device or to the CPU (i.e., the system clock).

The following diagram provides a simplified explanation and the relationship between the previously mentioned terms. In most cases, there are multiple clock source options available for each of the clock outputs. However, not all the clock sources are available for all the output clocks. Scaling is an optional feature in most of the cases.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Output Compare module.

Library Interface Section	Description
General Setup Functions	Provides General Configurations • Operation on a `WAIT` instruction • System clock source selection
Primary Oscillator Setup Functions	Provides Configuration Routines for the Primary Oscillator: • Primary Oscillator Sleep Enable

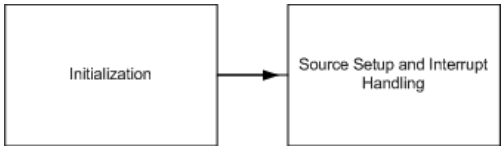
Secondary Oscillator Setup Functions	Provides Configuration Routines for the Secondary Oscillator: • Secondary Oscillator Enable/Disable • Secondary Oscillator Ready Indicator
Auxiliary Oscillator Setup Functions	Provides Configuration Routines for the Auxiliary Oscillator Configuration: • Reference Clock Source for Auxiliary Clock • Auxiliary Oscillator Mode Selection
Reference Oscillator Setup Functions	Provides Configuration Routines for the Reference Oscillator Configuration: • Reference Oscillator Output Enable/Disable • Select Reference Oscillator Output Divider • Reference Oscillator Clock Source Selection • Configure Reference Oscillator in Sleep Mode
Fast RC (FRC) Oscillator Setup Functions	Provides Configuration Routines for FRC Oscillator: • FRC Oscillator clock divider selection • FRC Oscillator tuning
Oscillator Switch Setup Functions	Provides Configuration Routines for Oscillator Switch and New Oscillator Selection: • Initiate an oscillator switch • Select the new oscillator • Get the current oscillator selection bits
Doze Setup Functions	Provides Configuration Routines for Doze mode Configuration • Enable/Disable Doze mode • Select DOZE (CPU Peripheral Clock Ratio) bits
USB and Display Clock Setup Functions	Provides Configuration Routines for USB and Display Clock Configuration: • Select FRC as USB Clock • Select Graphics Controller Clock • Display Module Clock Divider Selection
PLL Setup Functions	Provides Configuration Routines for PLL (Including USB and Auxiliary PLL) • PLL Multiplier • PLL Output Divider • 96 MHz PLL Enable for USB/Graphics Clock • Auxiliary PLL Input Divider • Auxiliary PLL Output Divider • Auxiliary PLL Multiplier • Auxiliary PLL Enable/Disable • PLL and USB PLL Lock Status
Peripheral Clock Setup Functions	Provides Configuration Routines for the Peripheral Clock • Set Peripheral Block Clock Divisor
Clock Fail Monitoring Functions	Provides Routines for Clock Fail Monitoring • Clock Fail Detect Status
Feature Existence Functions	Provides interface routines that determine whether certain features are available.

How the Library Works

The library can be used to control the Oscillator module. The usage model is explained in this section.

Description

The following diagram shows the major components of the usage model:



Oscillator Selection and Switching

Note: The device Configuration options change with the microcontroller family. Refer to the **"Oscillator"** chapter in the specific device data sheet for possible device configurations.

On devices that allow it, the oscillator selection can be overridden at run-time using the Oscillator Peripheral Library, as shown in the following code example:

```
//Do the configuration bit settings
OSC_SYS_TYPE currOsc;
OSC_SYS_TYPE newOsc=OSC_PRIMARY;

//get the current oscillator
currOsc =  PLIB_OSC_CurrentSysClockGet(OSC_ID_0);

//check if the current oscillator is same as new oscillator
if(currOsc != newOsc)
{
    // Unlock the Oscillator Registers
    PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);

    //set the new oscillator request to Fast RC oscillator
    PLIB_OSC_SysClockSelect ( OSC_ID_0, newOsc );
}
```

Example: Oscillator Selection Change During Run-time

In the event the clock switch did not complete, the clock switch logic can be reset by calling the following API:

```
bool oscSwitch_st;

oscSwitch_st = PLIB_OSC_ClockSwitchingIsComplete(OSC_ID_0);

if(!oscSwitch_st)
{
    // Unlock the Oscillator Registers
    PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);

    PLIB_OSC_ClockSwitchingAbort(OSC_ID_0);
}
```

Example: Oscillator Switch Abort

This will abandon the clock switch process, stop and reset the oscillator start-up timer (if applicable), and stop the PLL (if applicable). The clock switch process can be aborted at any time. A clock switch that is already in progress can also be aborted by performing a second clock switch.

Clock Sources

This section explains the available clock sources and their setup.

The following clock sources are available in Microchip microcontrollers:

- Primary Oscillator (Posc)
- Secondary Oscillator (Sosc)
- Internal Fast RC (FRC) Oscillator
- Internal Low-Power RC (LPRC) Oscillator

Primary Oscillator (Posc)

The Primary Oscillator has several operating modes (refer to the **"Oscillator"** chapter in the specific device data sheet for exact operating modes). The selection of the operating mode is done using the device Configuration registers during device programming. During run-time this can be changed using the oscillator switch option. However, changing the operating mode from High Gain (HC) to External Clock (EC) or External Resonator(XT) is not possible.

If a PLL is available, the PLL input divider can only be programmed during device programming. However, the PLL output divider and the PLL multiplier can be changed during run-time using the PLL functions described in the PLL section.

Secondary Oscillator (Sosc)

The optional Secondary Oscillator is designed specifically for low-power operation with an external crystal. The Secondary Oscillator is enabled in hardware by the device Configuration bits. Once it is enabled in hardware, it can be switched on during software run-time.

```
bool oscSecondary_st;
```

```
oscSecondary_st = PLIB_OSC_SecondaryIsEnabled(OSC_ID_0);
if(oscSecondary_st)
{
   // Unlock the Oscillator Registers
   PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);

   PLIB_OSC_SecondaryDisable(OSC_ID_0);
}
```

Internal Fast RC (FRC) Oscillator

The FRC Oscillator is a fast user-trimmable internal RC oscillator with a user-selectable input divider, PLL multiplier, and output divider. See the **"Oscillator"** chapter in the specific device data sheet for more information on the FRC oscillator.

Once the FRC oscillator is selected using the device Configuration registers or an oscillator switch is initiated selecting the FRC as a new oscillator, the FRC Oscillator PLL divider can be changed during run-time.

```
OSC_FRC_DIV FRCDivisor;

//Get the current divider value for FRC
FRCDivisor = PLIB_OSC_FRCDivisorGet(OSC_ID_0);

if(FRCDivisor != OSC_FRC_DIV_4)
{
    // Unlock the Oscillator Registers
    PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);

    PLIB_OSC_FRCDivisorSelect ( OSC_ID_0, OSC_FRC_DIV_4 );
}
```

Internal Low-Power RC (LPRC) Oscillator

The LPRC Oscillator is separate from the FRC Oscillator. It oscillates at a nominal frequency of 31.25 kHz (this value is device-dependent). The LPRC Oscillator can act as the clock source for the Power-up Timer (PWRT), Watchdog Timer (WDT), Fail-Safe Clock Monitor (FSCM), and PLL reference circuits. It can also be used to provide a low-frequency clock source option for the device in those applications where power consumption is critical and timing accuracy is not required.

The LPRC remains enabled after power on in the following conditions:

- Fail-safe clock monitoring is enabled
- Watchdog Timer is enabled
- LPRC is selected as the system clock

PLL Configuration

```
OSC_SYSPLL_OUT_DIV PLLOutDiv;
OSC_SYSPLL_MULTIPLIER pll_multiply;

// Unlock the Oscillator Registers
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);

pll_multiply = PLIB_OSC_SysPLLMultiplierGet(OSC_ID_0);
if(pll_multiply != 15)
{
    PLIB_OSC_SysPLLMultiplierSelect(OSC_ID_0, 15);
}
PLLOutDiv = PLIB_OSC_SysPLLOutputDivisorGet(OSC_ID_0);

if(PLLOutDiv != OSC_SYSPLL_OUT_DIV_8)
{
    PLIB_OSC_SysPLLOutputDivisorSelect(OSC_ID_0, OSC_SYSPLL_OUT_DIV_8);
}
```

Fail-Safe Clock Monitor

Fail-Safe Clock Monitor (FSCM)

The Fail-Safe Clock Monitor (FSCM) is designed to allow continued device operation if the current oscillator fails. It is intended for use with the Primary Oscillator (Posc) and automatically switches to the FRC oscillator if a Posc failure is detected. The switch to the FRC oscillator allows continued device operation and the ability to retry the Posc or to execute the appropriate for a clock failure.

The FSCM can be enabled in the device configuration during the programming of the device. During run-time, the clock failure status can be obtained as follows:

```
bool clockFail;
```

```
//Returns true if clock failure is detected.
clockFail = PLIB_OSC_ClockHasFailed(OSC_ID_0);
```

Internal FRC Oscillator Tuning

Tuning the Oscillator

Oscillator tuning will help compensate for temperature effects on the FRC frequency over a wide range of temperatures. The tuning step size is an approximation, the application is supposed to try different values to achieve the best result. In some of the devices there are different tuning modes available.

Direct Number Method

FRC tuning is based on the number specified by the PLIB_OSC_FRCTuningSelect function.

```
PLIB_OSC_FRCTuningModeSet(OSC_ID_0, OSC_TUNING_USING_NUMBER);

PLIB_OSC_FRCTuningSelect(OSC_ID_0, 0x11);
```

Sequential Dithering

To get the sequential dithering working, the client needs to set eight sequencer values. Value 0 is set using the PLIB_OSC_FRCTuningSelect function. The other seven values (values 1 through 7) are set using the PLIB_OSC_FRCTuningSequenceValueSet function.

```
// Unlock the Oscillator Registers
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);

PLIB_OSC_FRCTuningModeSet(OSC_ID_0, OSC_TUNING_SEQ_DITHER);

PLIB_OSC_FRCTuningSelect(OSC_ID_0, 0x11);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_1,
        OSC_FRC_TUNE_MINUS_2_25_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_2,
        OSC_FRC_TUNE_MINUS_1_5_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_3,
        OSC_FRC_TUNE_MINUS_0_375_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_4,
        OSC_FRC_TUNE_PLUS_0_43_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_5,
        OSC_FRC_TUNE_PLUS_1_29_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_6,
        OSC_FRC_TUNE_PLUS_2_54_PERCENT);

PLIB_OSC_FRCTuningSequenceValueSet(OSC_ID_0, OSC_TUNING_SEQUENCER_7,
        OSC_FRC_TUNE_MINUS_3_PERCENT);

//Configure PWM (period, pulse width and turn on the module)
```

Pseudo-Random Number

Select the tuning mode. Then, configure the PWM module and set the period and pulse width. The Oscillator system generates a 4-bit number based on a pseudo-random number generation algorithm. It then uses this value to tune the FRC oscillator. The module will generate different frequencies corresponding to the generated pseudo-random numbers every eighth PWM cycle.

```
PLIB_OSC_FRCTuningModeSet(OSC_ID_0, OSC_TUNING_PSEUDO_RANDOM);

//15 bit Linear Feedback shift register value
PLIB_OSC_LinearFeedbackShiftRegSet(OSC_ID_0, 0x7FFF);

//Configure PWM (period, pulse width and turn on the module)
```

Configuring the Library

The library is configured for the supported Oscillator module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Setup Functions

	Name	Description
	PLIB_OSC_OnWaitActionGet	Gets the configured operation to be performed when a WAIT instruction is executed.
	PLIB_OSC_OnWaitActionSet	Selects the operation to be performed when a WAIT instruction is executed.
	PLIB_OSC_SlewDisable	Disables the selected type of slewing.
	PLIB_OSC_SlewDivisorStepGet	Get the slew divisor maximum step.
	PLIB_OSC_SlewDivisorStepSelect	Selects division steps used while slewing.
	PLIB_OSC_SlewEnable	Enables the selected type of slewing.
	PLIB_OSC_SlewIsEnabled	Returns 'true' if the reference oscillator is disabled in Idle mode.
	PLIB_OSC_SleepToStartupClockGet	Returns the clock used for the duration when the device wakes from sleep and the clock ready.
	PLIB_OSC_SleepToStartupClockSelect	Selects the clock duration for when the device wakes from sleep and the clock is ready.
	PLIB_OSC_DreamModeDisable	Disables the dream mode.
	PLIB_OSC_DreamModeEnable	Enables the dream mode.
	PLIB_OSC_DreamModeStatus	gets the status of the dream mode.

b) Primary Oscillator Setup Functions

	Name	Description
	PLIB_OSC_ClockIsReady	Get the ready status of clock.
	PLIB_OSC_ClockSlewingIsActive	Returns the status of clock slewing.
	PLIB_OSC_SystemClockDivisorGet	Get the system clock divisor value.
	PLIB_OSC_SystemClockDivisorSelect	Selects system clock divisor.

c) Secondary Oscillator Setup Functions

	Name	Description
	PLIB_OSC_SecondaryDisable	Disables the Secondary Oscillator.
	PLIB_OSC_SecondaryEnable	Enables the Secondary Oscillator.
	PLIB_OSC_SecondaryIsEnabled	Returns 'true' if the Secondary Oscillator is enabled.
	PLIB_OSC_SecondaryIsReady	Returns 'true' if the Secondary Oscillator is ready.

d) Reference Oscillator Setup Functions

	Name	Description
	PLIB_OSC_ReferenceOscBaseClockSelect	Sets the base clock for the reference oscillator.
	PLIB_OSC_ReferenceOscDisable	Disables the reference oscillator output.
	PLIB_OSC_ReferenceOscDivisorValueSet	Selects the reference oscillator divisor value.
	PLIB_OSC_ReferenceOscEnable	Enables the reference oscillator.
	PLIB_OSC_ReferenceOscIsEnabled	Gets the enable status of the reference oscillator output.
	PLIB_OSC_ReferenceOscSourceChangeIsActive	Returns 'true' if a reference oscillator source change request is active.
	PLIB_OSC_ReferenceOscStopInIdleDisable	Enables the reference oscillator in Idle mode.
	PLIB_OSC_ReferenceOscStopInIdleEnable	Configures the reference oscillator to stop operating in Idle mode.
	PLIB_OSC_ReferenceOscStopInIdleIsEnabled	Returns 'true' if the reference oscillator is disabled in Idle mode.
	PLIB_OSC_ReferenceOscStopInSleepDisable	Enables the reference oscillator in Sleep mode.
	PLIB_OSC_ReferenceOscStopInSleepEnable	Configures the reference oscillator to stop operating in Sleep mode.
	PLIB_OSC_ReferenceOscStopInSleepIsEnabled	Returns 'true' if the reference oscillator is disabled in Sleep mode.
	PLIB_OSC_ReferenceOscSwitchIsComplete	Returns 'true' if the reference oscillator base clock switching is complete.
	PLIB_OSC_ReferenceOscTrimSet	Sets the reference oscillator divisor trim value.
	PLIB_OSC_ReferenceOutputDisable	Disables the reference oscillator output.
	PLIB_OSC_ReferenceOutputEnable	Enables the reference oscillator output.
	PLIB_OSC_ReferenceOutputIsEnabled	Returns 'true' if the reference oscillator output is enabled.

e) Fast RC Oscillator Setup Functions

	Name	Description
	PLIB_OSC_FRCDivisorGet	Gets the FRC clock divisor.
	PLIB_OSC_FRCDivisorSelect	Sets the FRC clock divisor to the specified value.
	PLIB_OSC_FRCTuningSelect	Sets the FRC tuning value.

f) Oscillator Switch Setup Functions

	Name	Description
	PLIB_OSC_ClockSwitchingAbort	Aborts an oscillator switch.
	PLIB_OSC_ClockSwitchingIsComplete	Gets the oscillator switch progress status.
	PLIB_OSC_CurrentSysClockGet	Gets the current oscillator selected.
	PLIB_OSC_SysClockSelect	Selects the new oscillator.

g) USB and Display Clock Setup Functions

	Name	Description
	PLIB_OSC_UsbClockSourceGet	Gets the USB module clock source.
	PLIB_OSC_UsbClockSourceSelect	Sets the USB module clock source.

h) PLL Setup Functions

	Name	Description
	PLIB_OSC_PLLClockIsLocked	Gets the lock status for the clock and PLL selections.
	PLIB_OSC_PLLClockLock	Locks the clock and PLL selections.
	PLIB_OSC_PLLClockUnlock	Unlocks the clock and PLL selections.
	PLIB_OSC_PLLIsLocked	Returns 'true' if the selected PLL module is locked.
	PLIB_OSC_SysPLLFrequencyRangeGet	Gets the frequency range for the PLL module.
	PLIB_OSC_SysPLLFrequencyRangeSet	Sets the frequency range for the PLL module.
	PLIB_OSC_SysPLLInputClockSourceGet	Gets the input clock source for the PLL module.
	PLIB_OSC_SysPLLInputClockSourceSet	Sets the input clock source for the PLL module.
	PLIB_OSC_SysPLLInputDivisorGet	Gets the input divisor for the PLL.
	PLIB_OSC_SysPLLMultiplierGet	Gets the PLL multiplier.
	PLIB_OSC_SysPLLMultiplierSelect	Sets the PLL multiplier to the specified value.
	PLIB_OSC_SysPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_SysPLLOutputDivisorSet	Sets the output divider for the PLL to the specified value.
	PLIB_OSC_SysPLLInputDivisorSet	Sets the input divider for the PLL to the specified value.
	PLIB_OSC_BTPLLClockOutDisable	Disables the Bluetooth PLL Clock Output.
	PLIB_OSC_BTPLLClockOutEnable	Enables the Bluetooth PLL clock Ouput.
	PLIB_OSC_BTPLLClockOutStatus	gets the status of the Bluetooth PLL clock Output.
	PLIB_OSC_BTPLLFrequencyRangeGet	Gets the frequency range for the Bluetooth PLL module.
	PLIB_OSC_BTPLLFrequencyRangeSet	Sets the frequency range for the Bluetooth PLL module.
	PLIB_OSC_BTPLLInputClockSourceGet	Gets the input clock source for the Bluetooth PLL module.
	PLIB_OSC_BTPLLInputClockSourceSet	Sets the input clock source for the Bluetooth PLL module.
	PLIB_OSC_BTPLLInputDivisorGet	Gets the input divisor for the Bluetooth PLL.
	PLIB_OSC_BTPLLInputDivisorSet	Sets the input divider for the Bluetooth PLL to the specified value.
	PLIB_OSC_BTPLLMultiplierGet	Gets the Bluetooth PLL multiplier.
	PLIB_OSC_BTPLLMultiplierSelect	Sets the Bluetooth PLL multiplier to the specified value.
	PLIB_OSC_BTPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_BTPLLOutputDivisorSet	Sets the output divider for the Bluetooth PLL to the specified value.
	PLIB_OSC_ForceSPLLLockDisable	Disables the Force PLL Lock feature for specified PLL.
	PLIB_OSC_ForceSPLLLockEnable	Enables the Force PLL Lock feature for specified PLL.
	PLIB_OSC_ForceSPLLLockStatus	gets the status of the force PLL Lock bit of the specified PLL.
	PLIB_OSC_PLLBypassDisable	Disables the PLL Bypass.
	PLIB_OSC_PLLBypassEnable	Enables the PLL Bypass.
	PLIB_OSC_PLLBypassStatus	gets the status of the PLL Bypass.
	PLIB_OSC_UPLLFrequencyRangeGet	Gets the frequency range for the USB PLL module.
	PLIB_OSC_UPLLFrequencyRangeSet	Sets the frequency range for the USB PLL module.

	PLIB_OSC_UPLLInputDivisorGet	Gets the input divisor for the USB PLL.
	PLIB_OSC_UPLLInputDivisorSet	Sets the input divider for the USB PLL to the specified value.
	PLIB_OSC_UPLLMultiplierGet	Gets the USB PLL multiplier.
	PLIB_OSC_UPLLMultiplierSelect	Sets the USB PLL multiplier to the specified value.
	PLIB_OSC_UPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_UPLLOutputDivisorSet	Sets the output divider for the USB PLL to the specified value.
	PLIB_OSC_ResetPLLAssert	Asserts the PLL reset for selected PLL.
	PLIB_OSC_ResetPLLDeassert	Deasserts the PLL reset for selected PLL.
	PLIB_OSC_ResetPLLStatus	gets the status of the PLL reset bit for the specified PLL.

i) Peripheral Clock Setup Functions

	Name	Description
	PLIB_OSC_PBClockDivisorGet	Gets the peripheral bus clock divisor.
	PLIB_OSC_PBClockDivisorIsReady	Checks whether the peripheral bus clock divisor is ready to be written.
	PLIB_OSC_PBClockDivisorSet	Sets the peripheral bus clock divisor to the specified value.
	PLIB_OSC_PBOutputClockDisable	Disables the peripheral bus output clock.
	PLIB_OSC_PBOutputClockEnable	Enables the peripheral bus output clock
	PLIB_OSC_PBOutputClockIsEnabled	Checks whether or not the peripheral bus clock output is enabled.

j) Clock Functions

	Name	Description
	PLIB_OSC_ClockHasFailed	Returns 'true' if the clock fails.
	PLIB_OSC_ClockStart	Starts the specified clock source.
	PLIB_OSC_ClockStop	Stops the specified clock source.
	PLIB_OSC_ClockStopStatus	returns the status of clock stop bit for the specified clock source.

k) Feature Existence Functions

	Name	Description
	PLIB_OSC_ExistsClockFail	Identifies whether the ClockFail feature exists on the Oscillator module.
	PLIB_OSC_ExistsFRCDivisor	Identifies whether the FRCDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsFRCTuning	Identifies whether the FRCTuning feature exists on the Oscillator module.
	PLIB_OSC_ExistsOnWaitAction	Identifies whether the OnWaitAction feature exists on the Oscillator module.
	PLIB_OSC_ExistsOscCurrentGet	Identifies whether the OscCurrentGet feature exists on the Oscillator module.
	PLIB_OSC_ExistsOscSelect	Identifies whether the OscSelect feature exists on the Oscillator module.
	PLIB_OSC_ExistsOscSwitchInit	Identifies whether the OscSwitchInit feature exists on the Oscillator module.
	PLIB_OSC_ExistsPBClockDivisor	Identifies whether the PBClockDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsPBClockOutputEnable	Identifies whether the PBClockOutputEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsPBClockReady	Identifies whether the PBClockReady feature exists on the Oscillator module.
	PLIB_OSC_ExistsPLLClockLock	Identifies whether the PLLClockLock feature exists on the Oscillator module.
	PLIB_OSC_ExistsPLLLockStatus	Identifies whether the PLLLockStatus feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscBaseClock	Identifies whether the ReferenceOscBaseClock feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscChange	Identifies whether the ReferenceOscChange feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscChangeActive	Identifies whether the ReferenceOscChangeActive feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscDivisor	Identifies whether the ReferenceOscDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscEnable	Identifies whether the ReferenceOscEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscStopInIdleEnable	Identifies whether the ReferenceOscStopInIdleEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscStopInSleep	Identifies whether the ReferenceOscStopInSleep feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscTrim	Identifies whether the ReferenceOscTrim feature exists on the Oscillator module.

	PLIB_OSC_ExistsReferenceOutputEnable	Identifies whether the ReferenceOutputEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsSecondaryEnable	Identifies whether the SecondaryEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsSecondaryReady	Identifies whether the SecondaryReady feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLFrequencyRange	Identifies whether the PLLFrequencyRange feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLInputClockSource	Identifies whether the PLLInputClockSource feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLInputDivisor	Identifies whether the PLLInputDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLMultiplier	Identifies whether the PLLMultiplier feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLOutputDivisor	Identifies whether the PLLOutputDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsUsbClockSource	Identifies whether the UsbClockSource feature exists on the Oscillator module.
	PLIB_OSC_ExistsClockReadyStatus	Identifies whether the ClockReadyStatus feature exists on the Oscillator module.
	PLIB_OSC_ExistsClockSlewingStatus	Identifies whether the ClockSlewingStatus feature exists on the Oscillator module.
	PLIB_OSC_ExistsSleepToStartupClock	Identifies whether the SleepToStartupClock feature exists on the Oscillator module.
	PLIB_OSC_ExistsSlewDivisorStepControl	Identifies whether the SlewDivisorStepControl feature exists on the Oscillator module.
	PLIB_OSC_ExistsSlewEnableControl	Identifies whether the SlewEnableControl feature exists on the Oscillator module.
	PLIB_OSC_ExistsSystemClockDivisorControl	Identifies whether the SystemClockDivisorControl feature exists on the Oscillator module.
	PLIB_OSC_ExistsBTPLLClockOut	Identifies whether the BTPLLClockOut feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLFrequencyRange	Identifies whether the BTPLLFrequencyRange feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLInputClockSource	Identifies whether the BTPLLInputClockSource feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLInputDivisor	Identifies whether the BTPLLInputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLMultiplier	Identifies whether the BTPLLMultiplier feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLOutputDivisor	Identifies whether the BTPLLOutputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsClockDiagStatus	Identifies whether the ClockDiagStatus feature exists on the OSC module
	PLIB_OSC_ExistsDreamModeControl	Identifies whether the DreamModeControl feature exists on the OSC module
	PLIB_OSC_ExistsForceLock	Identifies whether the ForceLock feature exists on the OSC module
	PLIB_OSC_ExistsPLLBypass	Identifies whether the SPLLBypass feature exists on the OSC module
	PLIB_OSC_ExistsResetPLL	Identifies whether the ResetPLL feature exists on the OSC module
	PLIB_OSC_ExistsUPLLFrequencyRange	Identifies whether the UPLLFrequencyRange feature exists on the OSC module
	PLIB_OSC_ExistsUPLLInputDivisor	Identifies whether the UPLLInputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsUPLLMultiplier	Identifies whether the UPLLMultiplier feature exists on the OSC module
	PLIB_OSC_ExistsUPLLOutputDivisor	Identifies whether the UPLLOutputDivisor feature exists on the OSC module

l) Data Types and Constants

	Name	Description
	OSC_PB_CLOCK_DIV_TYPE	Type of the oscillator PB Clock divisor value.
	OSC_REF_DIVISOR_TYPE	Reference oscillator divisor type.
	OSC_REFERENCE_MAX_DIV	Defines the reference clock output divisor maximum value.
	OSC_SYSPLL_MULTIPLIER_TYPE	Type of the oscillator system PLL multiplier value.
	OSC_FRC_DIV	Lists the possible Fast RC (FRC) Oscillator divider values.
	OSC_MODULE_ID	Possible instances of the OSC module.
	OSC_OPERATION_ON_WAIT	Lists the possible base clock values for the reference oscillator.
	OSC_PERIPHERAL_BUS	Lists the possible Peripheral buses.
	OSC_PLL_SELECT	Lists the PLLs in the Oscillator module.
	OSC_REFERENCE	Lists the possible reference oscillator.
	OSC_SYS_TYPE	Lists the possible oscillator type values.
	OSC_SYSPLL_FREQ_RANGE	Lists the possible PLL frequency range.
	OSC_SYSPLL_IN_CLK_SOURCE	Lists the possible input clock source for PLL module.
	OSC_SYSPLL_OUT_DIV	Lists the possible PLL output divider values.
	OSC_USBCLOCK_SOURCE	Lists the possible USB clock sources.

	OSC_CLOCK_ID	Lists the clock sources for which ready status can be checked.
	OSC_CLOCK_SLEW_TYPE	Lists the possible type of clock slewing.
	OSC_SLEEP_TO_STARTUP_CLK_TYPE	Lists the possible clock sources for sleep to start-up period.

Description

This section describes the Application Programming Interface (API) functions of the Oscillator Peripheral Library.

Refer to each section for a detailed description.

a) General Setup Functions

PLIB_OSC_OnWaitActionGet Function

Gets the configured operation to be performed when a WAIT instruction is executed.

File

plib_osc.h

C

```
OSC_OPERATION_ON_WAIT PLIB_OSC_OnWaitActionGet(OSC_MODULE_ID index);
```

Returns

On a WAIT action, one of the possible values of OSC_OPERATION_ON_WAIT.

Description

This function gets the configured operation that is to be performed when a WAIT instruction is executed.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsOnWaitAction in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_OSC_OnWaitActionGet(OSC_ID_0) == OSC_ON_WAIT_SLEEP)
{
    //Do some action
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_OPERATION_ON_WAIT PLIB_OSC_OnWaitActionGet (OSC_MODULE_ID index)

PLIB_OSC_OnWaitActionSet Function

Selects the operation to be performed when a WAIT instruction is executed.

File

plib_osc.h

C

```
void PLIB_OSC_OnWaitActionSet(OSC_MODULE_ID index, OSC_OPERATION_ON_WAIT onWait);
```

Returns

None.

Description

This function selects the operation to be performed when a WAIT instruction is executed.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsOnWaitAction in your application to determine whether this feature is available.

If this function is not called, the device will enter Idle mode on execution of a WAIT instruction.

Preconditions

None.

Example

```
PLIB_OSC_OnWaitActionSet(OSC_ID_0, OSC_ON_WAIT_SLEEP);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
onWait	Operation to be performed when a WAIT instruction is executed. One of the possible values of OSC_OPERATION_ON_WAIT.

Function

void PLIB_OSC_OnWaitActionSet (OSC_MODULE_ID index,
OSC_OPERATION_ON_WAIT onWait)

PLIB_OSC_SlewDisable Function

Disables the selected type of slewing.

File

plib_osc.h

C

```
void PLIB_OSC_SlewDisable(OSC_MODULE_ID index, OSC_CLOCK_SLEW_TYPE slewType);
```

Returns

None.

Description

This function disables slewing to an upward or downward frequency based on the selection.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSlewEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SlewDisable(OSC_ID_0, OSC_CLOCK_SLEW_DOWNWARD);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
slewType	One of the possible value from OSC_CLOCK_SLEW_TYPE

Function

void PLIB_OSC_SlewDisable (OSC_MODULE_ID index, OSC_CLOCK_SLEW_TYPE slewType)

PLIB_OSC_SlewDivisorStepGet Function

Get the slew divisor maximum step.

File

plib_osc.h

C

```
uint32_t PLIB_OSC_SlewDivisorStepGet(OSC_MODULE_ID index);
```

Returns

- slewSteps - Number of steps in which slewing is occurring

Description

This function returns the number of division steps used when slewing during a frequency change.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSlewDivisorStepControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t slewSteps;
slewSteps = PLIB_OSC_SlewDivisorStepGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint32_t PLIB_OSC_SlewDivisorStepGet

(

OSC_MODULE_ID index

)

PLIB_OSC_SlewDivisorStepSelect Function

Selects division steps used while slewing.

File

plib_osc.h

C

```
void PLIB_OSC_SlewDivisorStepSelect(OSC_MODULE_ID index, uint32_t slewSteps);
```

Returns

None.

Description

This API selects number of division steps to be used when slewing during a frequency change. If the number of steps chosen is 0, no divisor will be used while slewing. If the number of steps chosen is 1, slewing will start with divide by 2, and then no divisor. Similarly, if the number of steps chosen is 2, slewing will start with divisor 4. Then,after a few seconds the divisor will be 2, and then after a few more seconds no divisor will be used and so on. Therefore, the largest step would be equal to 2^(number of steps), and then the step size will reduce by half every few seconds.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSlewDivisorStepControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SlewDivisorStepSelect(OSC_ID_0, OSC_SLEW_DIVISOR_STEP_MAX_32);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
slewSteps	Number of steps in which slewing is desired

Function

void PLIB_OSC_SlewDivisorStepSelect
(
OSC_MODULE_ID index,
uint32_t slewSteps
)

PLIB_OSC_SlewEnable Function

Enables the selected type of slewing.

File

plib_osc.h

C

```
void PLIB_OSC_SlewEnable(OSC_MODULE_ID index, OSC_CLOCK_SLEW_TYPE slewType);
```

Returns

None.

Description

This function enables slewing to an upward or downward frequency based on the selection.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSlewEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SlewEnable(OSC_ID_0, OSC_CLOCK_SLEW_DOWNWARD);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
slewType	One of the possible value from OSC_CLOCK_SLEW_TYPE

Function

void PLIB_OSC_SlewEnable (OSC_MODULE_ID index, OSC_CLOCK_SLEW_TYPE slewType)

PLIB_OSC_SlewIsEnabled Function

Returns 'true' if the reference oscillator is disabled in Idle mode.

File

plib_osc.h

C

```
bool PLIB_OSC_SlewIsEnabled(OSC_MODULE_ID index, OSC_CLOCK_SLEW_TYPE slewType);
```

Returns

- true - The selected type of Slewing is enabled
- false - The selected type of Slewing is disabled

Description

This function returns 'true' if the reference oscillator is disabled in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSlewEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool slewStatus;
slewStatus = PLIB_OSC_SlewIsEnabled(OSC_ID_0, OSC_CLOCK_SLEW_DOWNWARD);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
slewType	One of the possible value from OSC_CLOCK_SLEW_TYPE.

Function

bool PLIB_OSC_SlewIsEnabled (OSC_MODULE_ID index, OSC_CLOCK_SLEW_TYPE slewType)

PLIB_OSC_SleepToStartupClockGet Function

Returns the clock used for the duration when the device wakes from sleep and the clock ready.

File

plib_osc.h

C

```
OSC_SLEEP_TO_STARTUP_CLK_TYPE PLIB_OSC_SleepToStartupClockGet(OSC_MODULE_ID index);
```

Returns

- startupOsc - One of the possible values from OSC_SLEEP_TO_STARTUP_CLK_TYPE

Description

When a device enters Sleep mode, the source of the system clock usually stops and takes some time after waking up to start running at full speed. This function is to return the clock set for that duration in which the actual clock source is starting up.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSleepToStartupClock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_SLEEP_TO_STARTUP_CLK_TYPE startupClk;
startupClk = PLIB_OSC_SleepToStartupClockGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SLEEP_TO_STARTUP_CLK_TYPE PLIB_OSC_SleepToStartupClockGet
(
OSC_MODULE_ID index
)

PLIB_OSC_SleepToStartupClockSelect Function

Selects the clock duration for when the device wakes from sleep and the clock is ready.

File

plib_osc.h

C

```
void PLIB_OSC_SleepToStartupClockSelect(OSC_MODULE_ID index, OSC_SLEEP_TO_STARTUP_CLK_TYPE startupOsc);
```

Returns

None.

Description

When a device enters Sleep mode, the source of the system clock usually stops and takes some time after waking up to start running at full speed. This function allows the user to select a clock for that duration in which the actual clock source is starting up.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSleepToStartupClock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SleepToStartupClockSelect(OSC_ID_0, OSC_SLEEP_TO_STARTUP_CLK_FRC);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
startupOsc	One of the possible values from OSC_SLEEP_TO_STARTUP_CLK_TYPE

Function

void PLIB_OSC_SleepToStartupClockSelect
(
OSC_MODULE_ID index,
OSC_SLEEP_TO_STARTUP_CLK_TYPE startupOsc
)

PLIB_OSC_DreamModeDisable Function

Disables the dream mode.

File

plib_osc.h

C

```
void PLIB_OSC_DreamModeDisable(OSC_MODULE_ID index);
```

Returns

None.

Description

This function is used to disable the dream mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsDreamModeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_DreamModeDisable(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_DreamModeDisable (OSC_MODULE_ID index)

PLIB_OSC_DreamModeEnable Function

Enables the dream mode.

File

plib_osc.h

C

```
void PLIB_OSC_DreamModeEnable(OSC_MODULE_ID index);
```

Returns

None.

Description

This function is used to enable the dream mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsDreamModeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_DreamModeEnable(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_DreamModeEnable (OSC_MODULE_ID index)

PLIB_OSC_DreamModeStatus Function

gets the status of the dream mode.

File

plib_osc.h

C

```
bool PLIB_OSC_DreamModeStatus(OSC_MODULE_ID index);
```

Returns

Status of dream mode bit.

Description

This function is used to get the status of the dream mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsDreamModeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool drmMode = PLIB_OSC_DreamModeStatus(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_DreamModeStatus (OSC_MODULE_ID index)

b) Primary Oscillator Setup Functions

PLIB_OSC_ClockIsReady Function

Get the ready status of clock.

File

plib_osc.h

C

```
bool PLIB_OSC_ClockIsReady(OSC_MODULE_ID index, OSC_CLOCK_ID clk);
```

Returns

- true - Indicates that the selected clock is running and is stable
- false - Indicates that the selected clock is either turned off or is still warming up

Description

This function returns the ready status of selected clock.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsClockReadyStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool clkReady;
clkReady = PLIB_OSC_ClockIsReady(OSC_ID_0, OSC_CLOCK_FAST_RC);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module.
clk	One of the values of OSC_CLOCK_ID.

Function

bool PLIB_OSC_ClockIsReady (OSC_MODULE_ID index, OSC_CLOCK_ID clk)

PLIB_OSC_ClockSlewingIsActive Function

Returns the status of clock slewing.

File

plib_osc.h

C

```
bool PLIB_OSC_ClockSlewingIsActive(OSC_MODULE_ID index);
```

Returns

- true - The clock frequency is being actively slewed to the new frequency.
- false - The clock switch has reached its final value

Description

This function returns the current status of clock switching slewing.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsClockSlewingStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_OSC_ClockSlewingIsActive(OSC_ID_0))
{
    //wait for clock switch to complete
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module.

Function

bool PLIB_OSC_ClockSlewingIsActive (OSC_MODULE_ID index)

PLIB_OSC_SystemClockDivisorGet Function

Get the system clock divisor value.

File

plib_osc.h

C

```
uint32_t PLIB_OSC_SystemClockDivisorGet(OSC_MODULE_ID index);
```

Returns

- systemClkDivisor - The value by which system clock is divided.

Description

This function returns the divisor set for system clock.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSystemClockDivisorControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t systemClkDivisor;
systemClkDivisor = PLIB_OSC_SystemClockDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint32_t PLIB_OSC_SystemClockDivisorGet
(
OSC_MODULE_ID index
)

PLIB_OSC_SystemClockDivisorSelect Function

Selects system clock divisor.

File

plib_osc.h

C

```
void PLIB_OSC_SystemClockDivisorSelect(OSC_MODULE_ID index, uint32_t systemClkDivisor);
```

Returns

None.

Description

This function selects the system clock divisor.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSystemClockDivisorControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SystemClockDivisorSelect(OSC_ID_0, OSC_SYSTEM_CLOCK_DIVIDED_BY_10);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
systemClkDivisor	The value by which the system clock is to be divided

Function

void PLIB_OSC_SystemClockDivisorSelect
(
OSC_MODULE_ID index,
uint32_t systemClkDivisor
)

c) Secondary Oscillator Setup Functions

PLIB_OSC_SecondaryDisable Function

Disables the Secondary Oscillator.

File

plib_osc.h

C

```
void PLIB_OSC_SecondaryDisable(OSC_MODULE_ID index);
```

Returns

None.

Description

This function disables the Secondary Oscillator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSecondaryEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SecondaryDisable(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_SecondaryDisable (OSC_MODULE_ID index);

PLIB_OSC_SecondaryEnable Function

Enables the Secondary Oscillator.

File

plib_osc.h

C

```
void PLIB_OSC_SecondaryEnable(OSC_MODULE_ID index);
```

Returns

None.

Description

This function enables the Secondary Oscillator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSecondaryEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SecondaryEnable(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_SecondaryEnable (OSC_MODULE_ID index)

PLIB_OSC_SecondaryIsEnabled Function

Returns 'true' if the Secondary Oscillator is enabled.

File

plib_osc.h

C

```
bool PLIB_OSC_SecondaryIsEnabled(OSC_MODULE_ID index);
```

Returns

- true - The Secondary Oscillator is enabled
- false - The Secondary Oscillator is disabled

Description

This function returns 'true' if the Secondary Oscillator is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSecondaryEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool secOscEnable;

secOscEnable = PLIB_OSC_SecondaryIsEnabled(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_SecondaryIsEnabled (OSC_MODULE_ID index);

PLIB_OSC_SecondaryIsReady Function

Returns 'true' if the Secondary Oscillator is ready.

File

plib_osc.h

C

```
bool PLIB_OSC_SecondaryIsReady(OSC_MODULE_ID index);
```

Returns

- true - Indicates that the Secondary Oscillator is running and is stable
- false - Indicates that the Secondary Oscillator is either turned off or is still warming up

Description

This function returns the ready status of the Secondary Oscillator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSecondaryReady in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool secOscReady;

secOscReady = PLIB_OSC_SecondaryIsReady(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_SecondaryIsReady (OSC_MODULE_ID index)

d) Reference Oscillator Setup Functions

PLIB_OSC_ReferenceOscBaseClockSelect Function

Sets the base clock for the reference oscillator.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscBaseClockSelect(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc,
OSC_REF_BASECLOCK refOscBaseClock);
```

Returns

None.

Description

This function sets the base clock for the reference oscillator. There are multiple clock sources by which the user can configure the module to output to the pin. Users can check the accuracy of the clock by probing the pin. Use the PLIB_OSC_ReferenceOscDivisorValueSet function to divide the clock if it is a very high value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscBaseClock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOscBaseClockSelect(OSC_ID_0, OSC_REFERENCE_1, OSC_REF_BASECLOCK_PBCLK);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator
refOscBaseClk	One of the possible values of OSC_REF_BASECLOCK

Function

void PLIB_OSC_ReferenceOscBaseClockSelect (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc, OSC_REF_BASECLOCK refOscBaseClock)

PLIB_OSC_ReferenceOscDisable Function

Disables the reference oscillator output.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscDisable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function disables output from the reference oscillator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOscDisable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOscDisable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscDivisorValueSet Function

Selects the reference oscillator divisor value.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscDivisorValueSet(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc,
OSC_REF_DIVISOR_TYPE refOscDivValue);
```

Returns

None.

Description

This function selects the reference oscillator divisor value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscDivisor in your application to determine whether this feature is available.

The value entered may not be the actual divisor. Please refer to the specific device data sheet to determine the actual divisor corresponding to the value entered.

Preconditions

None.

Example

```
// Select the clock source.
PLIB_OSC_ReferenceOscBaseClockSelect(OSC_ID_0, OSC_REFERENCE_1, OSC_REF_BASECLOCK_PBCLK);

PLIB_OSC_ReferenceOscDivisorValueSet(OSC_ID_0, OSC_REFERENCE_1, 128);

PLIB_OSC_ReferenceOscEnable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator
refOscDivValue	Value for Reference Oscillator Divisor (RODIV) field. If it is '0', the divider is not used.

Function

void PLIB_OSC_ReferenceOscDivisorValueSet (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc,
OSC_REF_DIVISOR_TYPE refOscDivValue)

PLIB_OSC_ReferenceOscEnable Function

Enables the reference oscillator.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscEnable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function enables the reference oscillator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscEnable in your application to determine whether this feature is available.

If the device has a reference clock output enable control, calling this function may not give the reference clock output. Use the PLIB_OSC_ReferenceOutputEnable function to enable the output.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOscEnable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOscEnable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscIsEnabled Function

Gets the enable status of the reference oscillator output.

File

plib_osc.h

C

```
bool PLIB_OSC_ReferenceOscIsEnabled(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

- true - The reference oscillator is enabled
- false - The reference oscillator is disabled

Description

This function gets the enable status of the reference oscillator output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_OSC_ReferenceOscIsEnabled(OSC_ID_0, OSC_REFERENCE_1))
{
//Do some action
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

bool PLIB_OSC_ReferenceOscIsEnabled (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscSourceChangeIsActive Function

Returns 'true' if a reference oscillator source change request is active.

File

plib_osc.h

C

```
bool PLIB_OSC_ReferenceOscSourceChangeIsActive(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

- true - The reference oscillator change request is active
- false - The reference oscillator change request is not active

Description

This function returns 'true' if the reference oscillator source change is in progress. The software is not allowed to give a new source change request.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscChangeActive in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (!PLIB_OSC_ReferenceOscSourceChangeIsActive(OSC_ID_0, OSC_REFERENCE_1))
{
    //Allowed to change the reference clock source
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

bool PLIB_OSC_ReferenceOscSourceChangeIsActive (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscStopInIdleDisable Function

Enables the reference oscillator in Idle mode.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscStopInIdleDisable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function enables the reference oscillator in Idle mode. The reference oscillator continues to run in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscStopInIdleEnable in your application to determine whether this feature is available.

The default state of the device is the reference oscillator is enabled in Idle mode.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOsctopInSleepDisable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOscStopInIdleDisable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscStopInIdleEnable Function

Configures the reference oscillator to stop operating in Idle mode.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscStopInIdleEnable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function configures the reference oscillator to stop operating in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscStopInIdleEnable in your application to determine whether this feature is available.

The default state of the device is reference oscillator enabled in Idle mode. Therefore, calling this function is necessary only if the PLIB_OSC_ReferenceOscStopInIdleDisable function was previously called, and the software wants to enable oscillator operation in Sleep mode.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOscStopInIdleEnable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOscStopInIdleEnable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscStopInIdleIsEnabled Function

Returns 'true' if the reference oscillator is disabled in Idle mode.

File

plib_osc.h

C

```
bool PLIB_OSC_ReferenceOscStopInIdleIsEnabled(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

- true - The reference oscillator is disabled in Idle mode
- false - The reference oscillator is enabled in Idle mode

Description

This function returns 'true' if the reference oscillator is disabled in Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscStopInIdleEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool refOscIdle;

refOscIdle = PLIB_OSC_ReferenceOscStopInIdleIsEnabled(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

bool PLIB_OSC_ReferenceOscStopInIdleIsEnabled (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscStopInSleepDisable Function

Enables the reference oscillator in Sleep mode.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscStopInSleepDisable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function enables the reference oscillator in Sleep mode. The reference oscillator continues to run in Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscStopInSleep in your application to determine whether this feature is available.

The reference clock output will be stopped in Sleep mode if the base clock selected is 'System Clock' or 'Peripheral Clock' regardless of this function.

The default state of the device is for the reference oscillator to be enabled in Sleep mode. Therefore, calling this function is necessary only if the PLIB_OSC_ReferenceOscStopInSleepEnable function was previously called, and the software wants to enable oscillator operation in Sleep mode.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOsctopInSleepDisable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOscStopInSleepDisable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscStopInSleepEnable Function

Configures the reference oscillator to stop operating in Sleep mode.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscStopInSleepEnable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function configures the reference oscillator to stop operating in Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscStopInSleep in your application to determine whether this feature is available.

The default state of the device is for the reference oscillator to be enabled in Sleep mode.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOscStopInSleepEnable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOscStopInSleepEnable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscStopInSleepIsEnabled Function

Returns 'true' if the reference oscillator is disabled in Sleep mode.

File

plib_osc.h

C

```
bool PLIB_OSC_ReferenceOscStopInSleepIsEnabled(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

- true - The reference oscillator is disabled in Sleep mode
- false - The reference oscillator is enabled in Sleep mode

Description

This function returns 'true' if the reference oscillator is disabled in Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscStopInSleep in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool refOscSleep;

refOscSleep = PLIB_OSC_ReferenceOscStopInSleepIsEnabled(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

bool PLIB_OSC_ReferenceOscStopInSleepIsEnabled (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscSwitchIsComplete Function

Returns 'true' if the reference oscillator base clock switching is complete.

File

plib_osc.h

C

```
bool PLIB_OSC_ReferenceOscSwitchIsComplete(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

- true - The reference clock base clock switching is complete
- false - The reference clock base clock switching is not complete; switching is not started

Description

This function returns 'true' if the reference oscillator base clock switching is complete.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscChange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool refOscIdle;

refOscIdle = PLIB_OSC_ReferenceOscSwitchIsComplete(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

bool PLIB_OSC_ReferenceOscSwitchIsComplete (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOscTrimSet Function

Sets the reference oscillator divisor trim value.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOscTrimSet(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc, OSC_REF_TRIM_TYPE
trimValue);
```

Returns

None.

Description

This function selects the reference oscillator divisor trim value. The value selected divided by OSC_REF_TRIM_MAX_VALUE will be added to the oscillator divisor value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOscTrim in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOscTrimSet(OSC_ID_0, OSC_REFERENCE_1, 50);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator
trimValue	Reference oscillator trim value

Function

void PLIB_OSC_ReferenceOscTrimSet (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc,
OSC_REF_TRIM_TYPE trimValue)

PLIB_OSC_ReferenceOutputDisable Function

Disables the reference oscillator output.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOutputDisable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function disables the reference oscillator output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOutputEnable in your application to determine whether this feature is available.

The default state of the device is reference oscillator output disabled.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOutputDisable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOutputDisable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOutputEnable Function

Enables the reference oscillator output.

File

plib_osc.h

C

```
void PLIB_OSC_ReferenceOutputEnable(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

None.

Description

This function enables the reference oscillator output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOutputEnable in your application to determine whether this feature is available.

The default state of the device is reference oscillator output disabled.

Preconditions

None.

Example

```
PLIB_OSC_ReferenceOutputEnable(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

void PLIB_OSC_ReferenceOutputEnable (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

PLIB_OSC_ReferenceOutputIsEnabled Function

Returns 'true' if the reference oscillator output is enabled.

File

plib_osc.h

C

```
bool PLIB_OSC_ReferenceOutputIsEnabled(OSC_MODULE_ID index, OSC_REFERENCE referenceOsc);
```

Returns

- true - The reference oscillator output is enabled
- false - The reference oscillator output is disabled

Description

This function returns 'true' if the reference oscillator output is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsReferenceOutputEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool refOscIdle;

refOscIdle = PLIB_OSC_ReferenceOutputIsEnabled(OSC_ID_0, OSC_REFERENCE_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
referenceOsc	Identifies the desired reference oscillator

Function

bool PLIB_OSC_ReferenceOutputIsEnabled (OSC_MODULE_ID index, OSC_REFERENCE referenceOsc)

e) Fast RC Oscillator Setup Functions

PLIB_OSC_FRCDivisorGet Function

Gets the FRC clock divisor.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_FRCDivisorGet(OSC_MODULE_ID index);
```

Returns

The FRC divisor value.

Description

This function gets the FRC clock divisor. The value returned will be direct number and not enum equivalent.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsFRCDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t divisorFRC;

divisorFRC = PLIB_OSC_FRCDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_FRCDivisorGet(OSC_MODULE_ID index)

PLIB_OSC_FRCDivisorSelect Function

Sets the FRC clock divisor to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_FRCDivisorSelect(OSC_MODULE_ID index, OSC_FRC_DIV divisorFRC);
```

Returns

None.

Description

This function sets the FRC clock divisor to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsFRCDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_FRCDivisorSelect ( OSC_ID_0, OSC_FRC_DIV_4 );
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
divisorFRC	One of the possible values from OSC_FRC_DIV

Function

void PLIB_OSC_FRCDivisorSelect (OSC_MODULE_ID index,
OSC_FRC_DIV divisorFRC)

PLIB_OSC_FRCTuningSelect Function

Sets the FRC tuning value.

File

plib_osc.h

C

```
void PLIB_OSC_FRCTuningSelect(OSC_MODULE_ID index, OSC_FRC_TUNE_TYPE tuningValue);
```

Returns

None.

Description

This function tunes the FRC oscillator to the value specified. The application is supposed to try different values and find the best one. If the device has different tuning modes, this function will be used differently. See the example provided for the PLIB_OSC_FRCTuningSequenceValueSet function for details.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsFRCTuning in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_FRCTuningSelect(OSC_ID_0, 0x05);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
tuningValue	Tuning value. One of the possible values from OSC_FRC_TUNE_TYPE.

Function

void PLIB_OSC_FRCTuningSelect (OSC_MODULE_ID index,

OSC_FRC_TUNE_TYPE tuningValue)

f) Oscillator Switch Setup Functions

PLIB_OSC_ClockSwitchingAbort Function

Aborts an oscillator switch.

File

plib_osc.h

C

```
void PLIB_OSC_ClockSwitchingAbort(OSC_MODULE_ID index);
```

Returns

None.

Description

This function aborts the oscillator switch to the selection specified by the new oscillator selection bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsOscSwitchInit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ClockSwitchingAbort(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_ClockSwitchingAbort (OSC_MODULE_ID index)

PLIB_OSC_ClockSwitchingIsComplete Function

Gets the oscillator switch progress status.

File

plib_osc.h

C

```
bool PLIB_OSC_ClockSwitchingIsComplete(OSC_MODULE_ID index);
```

Returns

- true - The oscillator switch is complete
- false - The oscillator switch is in progress

Description

This function gets the status of the oscillator switch progress.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsOscSwitchInit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SysClockSelect(OSC_ID_0, OSC_PRIMARY);

while(!PLIB_OSC_ClockSwitchingIsComplete(OSC_ID_0));
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_ClockSwitchingIsComplete(OSC_MODULE_ID index);

PLIB_OSC_CurrentSysClockGet Function

Gets the current oscillator selected.

File

plib_osc.h

C

```
OSC_SYS_TYPE PLIB_OSC_CurrentSysClockGet(OSC_MODULE_ID index);
```

Returns

One of the possible values from OSC_SYS_TYPE.

Description

This function gets the current oscillator. If the application has not changed the oscillator selection, this will be same as the oscillator selected through the Configuration bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsOscCurrentGet in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_SYS_TYPE oscCurrent;

oscCurrent = PLIB_OSC_CurrentSysClockGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SYS_TYPE PLIB_OSC_CurrentSysClockGet (OSC_MODULE_ID index)

PLIB_OSC_SysClockSelect Function

Selects the new oscillator.

File

plib_osc.h

C

```
void PLIB_OSC_SysClockSelect(OSC_MODULE_ID index, OSC_SYS_TYPE newOsc);
```

Returns

None.

Description

This function selects the new oscillator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsOscSelect in your application to determine whether this feature is available.

This function adds the necessary delay (NOP instructions) after switching the oscillator. Therefore, the user need not add any delay as specified in the device data sheet.

Preconditions

None.

Example

```
PLIB_OSC_SysClockSelect(OSC_ID_0, OSC_PRIMARY);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
newOsc	One of the possible values from OSC_SYS_TYPE

Function

void PLIB_OSC_SysClockSelect (OSC_MODULE_ID index,
OSC_SYS_TYPE newOsc)

g) USB and Display Clock Setup Functions

PLIB_OSC_UsbClockSourceGet Function

Gets the USB module clock source.

File

plib_osc.h

C

```
OSC_USBCLOCK_SOURCE PLIB_OSC_UsbClockSourceGet(OSC_MODULE_ID index);
```

Returns

USB module clock source. One of the possible values from OSC_USBCLOCK_SOURCE.

Description

This function gets the USB module clock source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUsbClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (SYS_OSC_USBCLK_FRC == PLIB_OSC_UsbClockSourceGet(OSC_ID_0))
{
//Do some action
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_USBCLOCK_SOURCE PLIB_OSC_UsbClockSourceGet (OSC_MODULE_ID index)

PLIB_OSC_UsbClockSourceSelect Function

Sets the USB module clock source.

File

plib_osc.h

C

```
void PLIB_OSC_UsbClockSourceSelect(OSC_MODULE_ID index, OSC_USBCLOCK_SOURCE usbClock);
```

Returns

None.

Description

This function sets the USB module clock source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUsbClockSource in your application to determine whether this feature is available.

Before placing the USB module in Suspend mode, use this function to enable the FRC clock.

Preconditions

None.

Example

```
PLIB_OSC_UsbClockSourceSelect(OSC_ID_0, SYS_OSC_USBCLK_FRC);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
usbClock	Select the USB module clock source. One of the possible values from OSC_USBCLOCK_SOURCE.

Function

void PLIB_OSC_UsbClockSourceSelect (OSC_MODULE_ID index,
OSC_USBCLOCK_SOURCE usbClock)

h) PLL Setup Functions

PLIB_OSC_PLLClockIsLocked Function

Gets the lock status for the clock and PLL selections.

File

plib_osc.h

C

```
bool PLIB_OSC_PLLClockIsLocked(OSC_MODULE_ID index);
```

Returns

- true - The clock and PLL selections are locked
- false - The clock and PLL selections are not locked

Description

This function gets the lock status for the clock and PLL selections.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPLLClockLock in your application to determine whether this feature is available.

If the PLL does not stabilize properly during start-up, this function may not reflect the actual status of PLL lock, nor does it detect when the PLL loses lock during normal operation.

Preconditions

None.

Example

```
bool clockPLL_st;

clockPLL_st = PLIB_OSC_PLLClockIsLocked(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_PLLClockIsLocked (OSC_MODULE_ID index)

PLIB_OSC_PLLClockLock Function

Locks the clock and PLL selections.

File

plib_osc.h

C

```
void PLIB_OSC_PLLClockLock(OSC_MODULE_ID index);
```

Returns

None.

Description

This function locks the clock and PLL selections.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPLLClockLock in your application to determine whether this feature is available.

The data given by this function is only valid if clock switching and monitoring are enabled. Otherwise Clock and PLL selections are never locked and may be modified.

Preconditions

The Fail-Safe Clock Monitor (FSCM) should be enabled.

Example

```
PLIB_OSC_PLLClockLock(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_PLLClockLock (OSC_MODULE_ID index)

PLIB_OSC_PLLClockUnlock Function

Unlocks the clock and PLL selections.

File

plib_osc.h

C

```
void PLIB_OSC_PLLClockUnlock(OSC_MODULE_ID index);
```

Returns

None.

Description

This function unlocks the clock and PLL selection so that the clock and PLL may be modified.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPLLClockLock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_PLLClockUnlock(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_PLLClockUnlock (OSC_MODULE_ID index)

PLIB_OSC_PLLIsLocked Function

Returns 'true' if the selected PLL module is locked.

File

plib_osc.h

C

```
bool PLIB_OSC_PLLIsLocked(OSC_MODULE_ID index, OSC_PLL_SELECT pllselect);
```

Returns

- true - The PLL module is in lock or the PLL module start-up timer is satisfied
- false - The PLL module is out of lock, the PLL start-up timer is running, or the PLL module is disabled

Description

This function returns the lock status of the selected PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPLLLockStatus in your application to determine whether this feature is available.

If the PLL does not stabilize properly during start-up, this function may not reflect the actual status of PLL lock, nor does it detect when the PLL loses lock during normal operation.

Preconditions

None.

Example

```
bool lockPLL_status;

lockPLL_status = PLIB_OSC_PLLIsLocked(OSC_ID_0, OSC_PLL_SYSTEM);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllselect	Selects the PLL module

Function

bool PLIB_OSC_PLLIsLocked (OSC_MODULE_ID index, OSC_PLL_SELECT pllselect)

PLIB_OSC_SysPLLFrequencyRangeGet Function

Gets the frequency range for the PLL module.

File

plib_osc.h

C

```
OSC_SYSPLL_FREQ_RANGE PLIB_OSC_SysPLLFrequencyRangeGet(OSC_MODULE_ID index);
```

Returns

- PLLFrequencyRange - One of the possible values from OSC_SYSPLL_FREQ_RANGE

Description

This function returns the frequency range set for the PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLFrequencyRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_SYSPLL_FREQ_RANGE PLLFrequencyRange;
PLLFrequencyRange = PLIB_OSC_SysPLLFrequencyRangeGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SYSPLL_FREQ_RANGE PLIB_OSC_SysPLLFrequencyRangeGet (OSC_MODULE_ID index)

PLIB_OSC_SysPLLFrequencyRangeSet Function

Sets the frequency range for the PLL module.

File

plib_osc.h

C

```
void PLIB_OSC_SysPLLFrequencyRangeSet(OSC_MODULE_ID index, OSC_SYSPLL_FREQ_RANGE PLLFrequencyRange);
```

Returns

None.

Description

This function sets the frequency range for the PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLFrequencyRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SysPLLFrequencyRangeSet(OSC_ID_0, OSC_SYSPLL_FREQ_RANGE_5M_TO_10M);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLFrequencyRange	One of the possible values from OSC_SYSPLL_FREQ_RANGE

Function

void PLIB_OSC_SysPLLFrequencyRangeSet (OSC_MODULE_ID index,
OSC_SYSPLL_FREQ_RANGE PLLFrequencyRange)

PLIB_OSC_SysPLLInputClockSourceGet Function

Gets the input clock source for the PLL module.

File

plib_osc.h

C

```
OSC_SYSPLL_IN_CLK_SOURCE PLIB_OSC_SysPLLInputClockSourceGet(OSC_MODULE_ID index);
```

Returns

- PLLInClockSource - One of the possible values from OSC_SYSPLL_IN_CLK_SOURCE

Description

This function returns the input clock source for the PLL module

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLInputClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_SYSPLL_IN_CLK_SOURCE PLLInClockSource;
PLLInClockSource = PLIB_OSC_SysPLLInputClockSourceGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SYSPLL_IN_CLK_SOURCE PLIB_OSC_SysPLLInputClockSourceGet (OSC_MODULE_ID index)

PLIB_OSC_SysPLLInputClockSourceSet Function

Sets the input clock source for the PLL module.

File

plib_osc.h

C

```
void PLIB_OSC_SysPLLInputClockSourceSet(OSC_MODULE_ID index, OSC_SYSPLL_IN_CLK_SOURCE PLLInClockSource);
```

Returns

None.

Description

This function sets the input clock source for the PLL module

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLInputClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SysPLLInputClockSourceSet(OSC_ID_0, OSC_SYSPLL_IN_CLK_SOURCE_FRC);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLInClockSource	One of the possible values from OSC_SYSPLL_IN_CLK_SOURCE

Function

void PLIB_OSC_SysPLLInputClockSourceSet (OSC_MODULE_ID index,
OSC_SYSPLL_IN_CLK_SOURCE PLLInClockSource)

PLIB_OSC_SysPLLInputDivisorGet Function

Gets the input divisor for the PLL.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_SysPLLInputDivisorGet(OSC_MODULE_ID index);
```

Returns

The System PLL input divisor as a number.

Description

This function returns the input divisor for the PLL.

Remarks

The direct register value itself is returned instead of the register value equivalent.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLInputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t pllInDiv;

PLLInDiv = PLIB_OSC_SysPLLInputDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_SysPLLInputDivisorGet (OSC_MODULE_ID index)

PLIB_OSC_SysPLLMultiplierGet Function

Gets the PLL multiplier.

File

plib_osc.h

C

```
OSC_SYSPLL_MULTIPLIER_TYPE PLIB_OSC_SysPLLMultiplierGet(OSC_MODULE_ID index);
```

Returns

One of the possible values of PB clock divisor of type OSC_SYSPLL_MULTIPLIER_TYPE.

Description

This function returns the PLL multiplier.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLMultiplier in your application to determine whether this feature is available.

The actual multiplier value will be returned, and NOT the 'PLLMULT' field value. Refer to the specific device data sheet for information.

Preconditions

None.

Example

```
OSC_SYSPLL_MULTIPLIER_TYPE pll_multiply;

pll_multiply = PLIB_OSC_SysPLLMultiplierGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SYSPLL_MULTIPLIER_TYPE PLIB_OSC_SysPLLMultiplierGet (OSC_MODULE_ID index)

PLIB_OSC_SysPLLMultiplierSelect Function

Sets the PLL multiplier to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_SysPLLMultiplierSelect(OSC_MODULE_ID index, OSC_SYSPLL_MULTIPLIER_TYPE pll_multiplier);
```

Returns

None.

Description

This function sets the PLL multiplier to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLMultiplier in your application to determine whether this feature is available.

Use the PLL Multiplier value directly for the parameter 'pll_multiplier', and NOT the value of the 'PLLMULT' field. Use of the PLL Multiplier value is not supported by the selected device, and therefore, library behavior is undefined. Refer to the specific device data sheet for information.

Preconditions

None.

Example

```
PLIB_OSC_SysPLLMultiplierSelect (OSC_ID_0, 0x08);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pll_multiplier	One of the possible values of PB clock divisor of type OSC_PLL_MUL_TYPE

Function

void PLIB_OSC_SysPLLMultiplierSelect (OSC_MODULE_ID index,
OSC_SYSPLL_MULTIPLIER_TYPE pll_multiplier)

PLIB_OSC_SysPLLOutputDivisorGet Function

Gets the output divisor for the PLL.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_SysPLLOutputDivisorGet(OSC_MODULE_ID index);
```

Returns

System PLL output divisor value.

Description

This function returns the output divisor for the System PLL. The value is the actual divisor and not the enum value corresponding to it.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLOutputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t pllOutDiv;

pllOutDiv = PLIB_OSC_SysPLLOutputDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_SysPLLOutputDivisorGet (OSC_MODULE_ID index)

PLIB_OSC_SysPLLOutputDivisorSet Function

Sets the output divider for the PLL to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_SysPLLOutputDivisorSet(OSC_MODULE_ID index, OSC_SYSPLL_OUT_DIV PLLOutDiv);
```

Returns

None.

Description

This function sets the output divider for the PLL to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLOutputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SysPLLOutputDivisorSelect(OSC_ID_0, OSC_SYSPLL_OUT_DIV_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLOutDiv	One of the possible values from OSC_SYSPLL_OUT_DIV

Function

void PLIB_OSC_SysPLLOutputDivisorSet (OSC_MODULE_ID index,
OSC_SYSPLL_OUT_DIV PLLOutDiv)

PLIB_OSC_SysPLLInputDivisorSet Function

Sets the input divider for the PLL to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_SysPLLInputDivisorSet(OSC_MODULE_ID index, uint16_t PLLInDiv);
```

Returns

None.

Description

This function sets the input divider for the PLL to the specified value.

Remarks

Pass the direct divisor instead of the register value equivalent.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSysPLLInputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_SysPLLInputDivisorSet( OSC_ID_0, 3 );
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLInDiv	System PLL input divisor value

Function

void PLIB_OSC_SysPLLInputDivisorSet (OSC_MODULE_ID index,

uint16_t PLLInDiv)

PLIB_OSC_BTPLLClockOutDisable Function

Disables the Bluetooth PLL Clock Output.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLClockOutDisable(OSC_MODULE_ID index);
```

Returns

None.

Description

This function is used to disable the Bluetooth PLL clock output to SoC pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLClockOut in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLClockOutDisable(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_BTPLLClockOutDisable (OSC_MODULE_ID index)

PLIB_OSC_BTPLLClockOutEnable Function

Enables the Bluetooth PLL clock Ouput.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLClockOutEnable(OSC_MODULE_ID index);
```

Returns

None.

Description

This function is used to enable the Bluetooth PLL clock Output to the SoC pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLClockOut in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLClockOutEnable(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

void PLIB_OSC_BTPLLClockOutEnable (OSC_MODULE_ID index)

PLIB_OSC_BTPLLClockOutStatus Function

gets the status of the Bluetooth PLL clock Output.

File

plib_osc.h

C

```
bool PLIB_OSC_BTPLLClockOutStatus(OSC_MODULE_ID index);
```

Returns

Status of Bluetooth PLL clock output.

Description

This function is used to get the status of the Bluetooth PLL clock Output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLClockOut in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool btpllClkOut = PLIB_OSC_BTPLLClockOutStatus(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_BTPLLClockOutStatus (OSC_MODULE_ID index)

PLIB_OSC_BTPLLFrequencyRangeGet Function

Gets the frequency range for the Bluetooth PLL module.

File

plib_osc.h

C

```
OSC_BTPLL_FREQ_RANGE PLIB_OSC_BTPLLFrequencyRangeGet(OSC_MODULE_ID index);
```

Returns

- PLLFrequencyRange - One of the possible values from OSC_BTPLL_FREQ_RANGE

Description

This function returns the frequency range set for the Bluetooth PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLFrequencyRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_BTPLL_FREQ_RANGE PLLFrequencyRange;
PLLFrequencyRange = PLIB_OSC_BTPLLFrequencyRangeGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_BTPLL_FREQ_RANGE PLIB_OSC_BTPLLFrequencyRangeGet (OSC_MODULE_ID index)

PLIB_OSC_BTPLLFrequencyRangeSet Function

Sets the frequency range for the Bluetooth PLL module.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLFrequencyRangeSet(OSC_MODULE_ID index, OSC_BTPLL_FREQ_RANGE PLLFrequencyRange);
```

Returns

None.

Description

This function sets the frequency range for the Bluetooth PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLFrequencyRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLFrequencyRangeSet(OSC_ID_0, OSC_BTPLL_FREQ_RANGE_5M_TO_10M);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLFrequencyRange	One of the possible values from OSC_BTPLL_FREQ_RANGE

Function

void PLIB_OSC_BTPLLFrequencyRangeSet (OSC_MODULE_ID index,
OSC_BTPLL_FREQ_RANGE PLLFrequencyRange)

PLIB_OSC_BTPLLInputClockSourceGet Function

Gets the input clock source for the Bluetooth PLL module.

File

plib_osc.h

C

```
OSC_SYSPLL_IN_CLK_SOURCE PLIB_OSC_BTPLLInputClockSourceGet(OSC_MODULE_ID index);
```

Returns

- PLLInClockSource - One of the possible values from OSC_BTPLL_IN_CLK_SOURCE

Description

This function returns the input clock source for the Bluetooth PLL module

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLInputClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_BTPLL_IN_CLK_SOURCE PLLInClockSource;
PLLInClockSource = PLIB_OSC_BTPLLInputClockSourceGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_BTPLL_IN_CLK_SOURCE PLIB_OSC_BTPLLInputClockSourceGet (OSC_MODULE_ID index)

PLIB_OSC_BTPLLInputClockSourceSet Function

Sets the input clock source for the Bluetooth PLL module.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLInputClockSourceSet(OSC_MODULE_ID index, OSC_BTPLL_IN_CLK_SOURCE PLLInClockSource);
```

Returns

None.

Description

This function sets the input clock source for the Bluetooth PLL module

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLInputClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLInputClockSourceSet(OSC_ID_0, OSC_BTPLL_IN_CLK_SOURCE_FRC);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLInClockSource	One of the possible values from OSC_BTPLL_IN_CLK_SOURCE

Function

void PLIB_OSC_BTPLLInputClockSourceSet (OSC_MODULE_ID index,
OSC_BTPLL_IN_CLK_SOURCE PLLInClockSource)

PLIB_OSC_BTPLLInputDivisorGet Function

Gets the input divisor for the Bluetooth PLL.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_BTPLLInputDivisorGet(OSC_MODULE_ID index);
```

Returns

The Bluetooth PLL input divisor as a number.

Description

This function returns the input divisor for the Bluetooth PLL.

Remarks

The direct register value itself is returned instead of the register value equivalent.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLInputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t pllInDiv;

PLLInDiv = PLIB_OSC_BTPLLInputDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_BTPLLInputDivisorGet (OSC_MODULE_ID index)

PLIB_OSC_BTPLLInputDivisorSet Function

Sets the input divider for the Bluetooth PLL to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLInputDivisorSet(OSC_MODULE_ID index, uint16_t PLLInDiv);
```

Returns

None.

Description

This function sets the input divider for the Bluetooth PLL to the specified value.

Remarks

Pass the direct divisor instead of the register value equivalent.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLInputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLInputDivisorSet( OSC_ID_0, 3 );
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLInDiv	USB PLL input divisor value

Function

void PLIB_OSC_BTPLLInputDivisorSet (OSC_MODULE_ID index, uint16_t PLLInDiv)

PLIB_OSC_BTPLLMultiplierGet Function

Gets the Bluetooth PLL multiplier.

File

plib_osc.h

C

```
OSC_SYSPLL_MULTIPLIER_TYPE PLIB_OSC_BTPLLMultiplierGet(OSC_MODULE_ID index);
```

Returns

One of the possible values of Bluetooth Multiplier of type OSC_SYSPLL_MULTIPLIER_TYPE.

Description

This function returns the Bluetooth PLL multiplier.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLMultiplier in your application to determine whether this feature is available.

The actual multiplier value will be returned, and NOT the 'BTPLLMULT' field value. Refer to the specific device data sheet for information.

Preconditions

None.

Example

```
OSC_SYSPLL_MULTIPLIER_TYPE pll_multiply;
```

```
pll_multiply = PLIB_OSC_BTPLLMultiplierGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SYSPLL_MULTIPLIER_TYPE PLIB_OSC_BTPLLMultiplierGet (OSC_MODULE_ID index)

PLIB_OSC_BTPLLMultiplierSelect Function

Sets the Bluetooth PLL multiplier to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLMultiplierSelect(OSC_MODULE_ID index, OSC_SYSPLL_MULTIPLIER_TYPE pll_multiplier);
```

Returns

None.

Description

This function sets the Bluetooth PLL multiplier to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLMultiplier in your application to determine whether this feature is available.

Use the USB PLL Multiplier value directly for the parameter 'pll_multiplier', and NOT the value of the 'BTPLLMULT' field. Use of the PLL Multiplier value is not supported by the selected device, and therefore, library behavior is undefined. Refer to the specific device data sheet for information.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLMultiplierSelect (OSC_ID_0, 0x08);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pll_multiplier	One of the possible values between 0 and 255

Function

void PLIB_OSC_BTPLLMultiplierSelect (OSC_MODULE_ID index,
OSC_SYSPLL_MULTIPLIER_TYPE pll_multiplier)

PLIB_OSC_BTPLLOutputDivisorGet Function

Gets the output divisor for the PLL.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_BTPLLOutputDivisorGet(OSC_MODULE_ID index);
```

Returns

Bluetooth PLL output divisor value.

Description

This function returns the output divisor for the Bluetooth PLL. The value is the actual divisor and not the enum value corresponding to it.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLOutputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t pllOutDiv;

pllOutDiv = PLIB_OSC_BTPLLOutputDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_BTPLLOutputDivisorGet (OSC_MODULE_ID index)

PLIB_OSC_BTPLLOutputDivisorSet Function

Sets the output divider for the Bluetooth PLL to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_BTPLLOutputDivisorSet(OSC_MODULE_ID index, OSC_BTPLL_OUT_DIV PLLOutDiv);
```

Returns

None.

Description

This function sets the output divider for the Bluetooth PLL to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsBTPLLOutputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_BTPLLOutputDivisorSelect(OSC_ID_0, OSC_BTPLL_OUT_DIV_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLOutDiv	One of the possible values from OSC_BTPLL_OUT_DIV

Function

void PLIB_OSC_BTPLLOutputDivisorSet (OSC_MODULE_ID index,
OSC_BTPLL_OUT_DIV PLLOutDiv)

PLIB_OSC_ForceSPLLLockDisable Function

Disables the Force PLL Lock feature for specified PLL.

File

plib_osc.h

C

```
void PLIB_OSC_ForceSPLLLockDisable(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

None.

Description

This function is used to disable the Force PLL Lock feature for the specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsForceLock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ForceSPLLLockDisable(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL for which force locked to be disabled

Function

void PLIB_OSC_ForceSPLLLockDisable (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_ForceSPLLLockEnable Function

Enables the Force PLL Lock feature for specified PLL.

File

plib_osc.h

C

```
void PLIB_OSC_ForceSPLLLockEnable(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

None.

Description

This function is used to enable the Force PLL Lock feature for the specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsForceLock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ForceSPLLLockEnable(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL to be force locked

Function

void PLIB_OSC_ForceSPLLLockEnable (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_ForceSPLLLockStatus Function

gets the status of the force PLL Lock bit of the specified PLL.

File

plib_osc.h

C

```
bool PLIB_OSC_ForceSPLLLockStatus(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

Status of Force PLL Lock Status for the specified PLL.

Description

This function is used to get the status of the force PLL Lock bit of the specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsForceLock in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool spllFLock = PLIB_OSC_ForceSPLLLockStatus(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL for which force lock status to be read.

Function

bool PLIB_OSC_ForceSPLLLockStatus (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_PLLBypassDisable Function

Disables the PLL Bypass.

File

plib_osc.h

C

```
void PLIB_OSC_PLLBypassDisable(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

None.

Description

This function is used to disable the PLL Bypass for a specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSPLLBypass in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_PLLBypassDisable(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL for which bypass has to be disabled

Function

void PLIB_OSC_SPLLBypassDisable (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_PLLBypassEnable Function

Enables the PLL Bypass.

File

plib_osc.h

C

```
void PLIB_OSC_PLLBypassEnable(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

None.

Description

This function is used to enable the PLL Bypass for a specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPLLBypass in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_PLLBypassEnable(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL for which bypass has to be enabled

Function

void PLIB_OSC_PLLBypassEnable (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_PLLBypassStatus Function

gets the status of the PLL Bypass.

File

plib_osc.h

C

```
bool PLIB_OSC_PLLBypassStatus(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

Status of SPLL Bypass bit.

Description

This function is used to get the status of the PLL Bypass bit for a specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsSPLLBypass in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool spllBypass = PLIB_OSC_SPLLBypassStatus(OSC_ID_0. OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL whose Bypass status has to be read

Function

bool PLIB_OSC_SPLLBypassStatus (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_UPLLFrequencyRangeGet Function

Gets the frequency range for the USB PLL module.

File

plib_osc.h

C

```
OSC_UPLL_FREQ_RANGE PLIB_OSC_UPLLFrequencyRangeGet(OSC_MODULE_ID index);
```

Returns

- PLLFrequencyRange - One of the possible values from OSC_UPLL_FREQ_RANGE

Description

This function returns the frequency range set for the USB PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLFrequencyRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
OSC_UPLL_FREQ_RANGE PLLFrequencyRange;
PLLFrequencyRange = PLIB_OSC_UPLLFrequencyRangeGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_UPLL_FREQ_RANGE PLIB_OSC_UPLLFrequencyRangeGet (OSC_MODULE_ID index)

PLIB_OSC_UPLLFrequencyRangeSet Function

Sets the frequency range for the USB PLL module.

File

plib_osc.h

C

```
void PLIB_OSC_UPLLFrequencyRangeSet(OSC_MODULE_ID index, OSC_UPLL_FREQ_RANGE PLLFrequencyRange);
```

Returns

None.

Description

This function sets the frequency range for the USB PLL module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLFrequencyRange in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_UPLLFrequencyRangeSet(OSC_ID_0, OSC_UPLL_FREQ_RANGE_5M_TO_10M);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLFrequencyRange	One of the possible values from OSC_UPLL_FREQ_RANGE

Function

void PLIB_OSC_UPLLFrequencyRangeSet (OSC_MODULE_ID index,

OSC_UPLL_FREQ_RANGE PLLFrequencyRange)

PLIB_OSC_UPLLInputDivisorGet Function

Gets the input divisor for the USB PLL.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_UPLLInputDivisorGet(OSC_MODULE_ID index);
```

Returns

The USB PLL input divisor as a number.

Description

This function returns the input divisor for the USB PLL.

Remarks

The direct register value itself is returned instead of the register value equivalent.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLInputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t pllInDiv;

PLLInDiv = PLIB_OSC_UPLLInputDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_UPLLInputDivisorGet (OSC_MODULE_ID index)

PLIB_OSC_UPLLInputDivisorSet Function

Sets the input divider for the USB PLL to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_UPLLInputDivisorSet(OSC_MODULE_ID index, uint16_t PLLInDiv);
```

Returns

None.

Description

This function sets the input divider for the USB PLL to the specified value.

Remarks

Pass the direct divisor instead of the register value equivalent.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLInputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_UPLLInputDivisorSet( OSC_ID_0, 3 );
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLInDiv	USB PLL input divisor value

Function

void PLIB_OSC_UPLLInputDivisorSet (OSC_MODULE_ID index,

uint16_t PLLInDiv)

PLIB_OSC_UPLLMultiplierGet Function

Gets the USB PLL multiplier.

File

plib_osc.h

C

```
OSC_SYSPLL_MULTIPLIER_TYPE PLIB_OSC_UPLLMultiplierGet(OSC_MODULE_ID index);
```

Returns

One of the possible values of USB Multiplier of type OSC_SYSPLL_MULTIPLIER_TYPE.

Description

This function returns the USB PLL multiplier.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLMultiplier in your application to determine whether this feature is available.

The actual multiplier value will be returned, and NOT the 'UPLLMULT' field value. Refer to the specific device data sheet for information.

Preconditions

None.

Example

```
OSC_SYSPLL_MULTIPLIER_TYPE pll_multiply;

pll_multiply = PLIB_OSC_UPLLMultiplierGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

OSC_SYSPLL_MULTIPLIER_TYPE PLIB_OSC_UPLLMultiplierGet (OSC_MODULE_ID index)

PLIB_OSC_UPLLMultiplierSelect Function

Sets the USB PLL multiplier to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_UPLLMultiplierSelect(OSC_MODULE_ID index, OSC_SYSPLL_MULTIPLIER_TYPE pll_multiplier);
```

Returns

None.

Description

This function sets the USB PLL multiplier to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLMultiplier in your application to determine whether this feature is available.

Use the USB PLL Multiplier value directly for the parameter 'pll_multiplier', and NOT the value of the 'UPLLMULT' field. Use of the PLL Multiplier value is not supported by the selected device, and therefore, library behavior is undefined. Refer to the specific device data sheet for information.

Preconditions

None.

Example

```
PLIB_OSC_UPLLMultiplierSelect (OSC_ID_0, 0x08);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pll_multiplier	One of the possible values between 0 and 255

Function

void PLIB_OSC_UPLLMultiplierSelect (OSC_MODULE_ID index,
OSC_SYSPLL_MULTIPLIER_TYPE pll_multiplier)

PLIB_OSC_UPLLOutputDivisorGet Function

Gets the output divisor for the PLL.

File

plib_osc.h

C

```
uint16_t PLIB_OSC_UPLLOutputDivisorGet(OSC_MODULE_ID index);
```

Returns

USB PLL output divisor value.

Description

This function returns the output divisor for the USB PLL. The value is the actual divisor and not the enum value corresponding to it.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLOutputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t pllOutDiv;

pllOutDiv = PLIB_OSC_UPLLOutputDivisorGet(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

uint16_t PLIB_OSC_UPLLOutputDivisorGet (OSC_MODULE_ID index)

PLIB_OSC_UPLLOutputDivisorSet Function

Sets the output divider for the USB PLL to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_UPLLOutputDivisorSet(OSC_MODULE_ID index, OSC_UPLL_OUT_DIV PLLOutDiv);
```

Returns

None.

Description

This function sets the output divider for the USB PLL to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsUPLLOutputDivisor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_UPLLOutputDivisorSelect(OSC_ID_0, OSC_UPLL_OUT_DIV_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
PLLOutDiv	One of the possible values from OSC_UPLL_OUT_DIV

Function

void PLIB_OSC_UPLLOutputDivisorSet (OSC_MODULE_ID index,
OSC_UPLL_OUT_DIV PLLOutDiv)

PLIB_OSC_ResetPLLAssert Function

Asserts the PLL reset for selected PLL.

File

plib_osc.h

C

```
void PLIB_OSC_ResetPLLAssert(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

None.

Description

This function is used to assert the PLL reset for the selected PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsResetPLL in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ResetPLLAssert(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL for which reset to be asserted

Function

void PLIB_OSC_ResetPLLAssert (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_ResetPLLDeassert Function

Deasserts the PLL reset for selected PLL.

File

plib_osc.h

C

```
void PLIB_OSC_ResetPLLDeassert(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

None.

Description

This function is used to deassert the PLL reset for the selected PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsResetPLL in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ResetPLLDeassert(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL for which reset to be deasserted

Function

void PLIB_OSC_ResetPLLDeassert (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

PLIB_OSC_ResetPLLStatus Function

gets the status of the PLL reset bit for the specified PLL.

File

plib_osc.h

C

```
bool PLIB_OSC_ResetPLLStatus(OSC_MODULE_ID index, OSC_PLL_SELECT pllSel);
```

Returns

Status of PLL reset bit for the specified PLL.

Description

This function is used to get the status of the PLL reset bit for the specified PLL.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsResetPLL in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool upllReset = PLIB_OSC_ResetPLLStatus(OSC_ID_0, OSC_PLL_USB);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
pllSel	PLL whose reset status to be checked

Function

bool PLIB_OSC_ResetPLLStatus (OSC_MODULE_ID index, OSC_PLL_SELECT pllSel)

i) Peripheral Clock Setup Functions

PLIB_OSC_PBClockDivisorGet Function

Gets the peripheral bus clock divisor.

File

plib_osc.h

C

```
OSC_PB_CLOCK_DIV_TYPE PLIB_OSC_PBClockDivisorGet(OSC_MODULE_ID index, OSC_PERIPHERAL_BUS
peripheralBusNumber);
```

Returns

One of the possible values of PB clock divisor of type OSC_PB_CLOCK_DIV_TYPE.

Description

This function returns the peripheral bus clock divisor.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPBClockDivisor in your application to determine whether this feature is available.

Actual Divisor value will be returned, NOT the 'PBDIV' field value. Refer the data sheet of the device for the detail.

Preconditions

None.

Example

```
OSC_PB_CLOCK_DIV_TYPE peripheralBusClkDiv;

peripheralBusClkDiv = PLIB_OSC_PBClockDivisorGet(OSC_ID_0,
                                                 OSC_PERIPHERAL_BUS_1);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Function

OSC_PB_CLOCK_DIV_TYPE PLIB_OSC_PBClockDivisorGet (OSC_MODULE_ID index,
OSC_PERIPHERAL_BUS peripheralBusNumber)

PLIB_OSC_PBClockDivisorIsReady Function

Checks whether the peripheral bus clock divisor is ready to be written.

File

plib_osc.h

C

```
bool PLIB_OSC_PBClockDivisorIsReady(OSC_MODULE_ID index, OSC_PERIPHERAL_BUS peripheralBusNumber);
```

Returns

- true - The peripheral bus clock divisor can be written
- false - The peripheral bus clock divisor cannot be written

Description

This function checks whether the peripheral bus clock divisor is ready to be written.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPBClockReady in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_OSC_PBClockDivisorIsReady(OSC_ID_0, OSC_PERIPHERAL_BUS_1))
{
    PLIB_OSC_PBClockDivisorSet (OSC_ID_0, OSC_PERIPHERAL_BUS_1,
                                0x01);
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Function

bool PLIB_OSC_PBClockDivisorIsReady(OSC_MODULE_ID index,
OSC_PERIPHERAL_BUS peripheralBusNumber)

PLIB_OSC_PBClockDivisorSet Function

Sets the peripheral bus clock divisor to the specified value.

File

plib_osc.h

C

```
void PLIB_OSC_PBClockDivisorSet(OSC_MODULE_ID index, OSC_PERIPHERAL_BUS peripheralBusNumber,
OSC_PB_CLOCK_DIV_TYPE peripheralBusClkDiv);
```

Returns

None.

Description

This function sets the peripheral bus clock divisor to the specified value.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPBClockDivisor in your application to determine whether this feature is available.

Use PB Divider value directly for the parameter 'peripheralBusClkDiv', and NOT the value of the 'PBDIV' field. Use of the PB Divider value is not supported by the selected device, and therefore, library behavior is undefined. Refer the the specific device data sheet for information.

Preconditions

Peripheral bus clock divisor should be ready to be written.

Example

```
PLIB_OSC_PBClockDivisorSet (OSC_ID_0, OSC_PERIPHERAL_BUS_1, 0x01);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus
peripheralBusClkDiv	One of the possible values of PB clock divisor of type OSC_PB_CLOCK_DIV_TYPE

Function

void PLIB_OSC_PBClockDivisorSet(OSC_MODULE_ID index,
OSC_PERIPHERAL_BUS peripheralBusNumber,
OSC_PB_CLOCK_DIV_TYPE peripheralBusClkDiv)

PLIB_OSC_PBOutputClockDisable Function

Disables the peripheral bus output clock.

File

plib_osc.h

C

```
void PLIB_OSC_PBOutputClockDisable(OSC_MODULE_ID index, OSC_PERIPHERAL_BUS peripheralBusNumber);
```

Returns

None

Description

This function disables the peripheral bus output clock.

Remarks

The clock for peripheral bus 1 cannot be turned off.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPBClockOutputEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_PBOutputClockDisable (OSC_ID_0, OSC_PERIPHERAL_BUS_2);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Function

void PLIB_OSC_PBOutputClockDisable(OSC_MODULE_ID index,
OSC_PERIPHERAL_BUS peripheralBusNumber)

PLIB_OSC_PBOutputClockEnable Function

Enables the peripheral bus output clock

File

plib_osc.h

C

```
void PLIB_OSC_PBOutputClockEnable(OSC_MODULE_ID index, OSC_PERIPHERAL_BUS peripheralBusNumber);
```

Returns

None

Description

This function enables the peripheral bus output clock

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPBClockOutputEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_PBOutputClockEnable (OSC_ID_0, OSC_PERIPHERAL_BUS_2);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Function

void PLIB_OSC_PBOutputClockEnable(OSC_MODULE_ID index,
OSC_PERIPHERAL_BUS peripheralBusNumber)

PLIB_OSC_PBOutputClockIsEnabled Function

Checks whether or not the peripheral bus clock output is enabled.

File

plib_osc.h

C

```
bool PLIB_OSC_PBOutputClockIsEnabled(OSC_MODULE_ID index, OSC_PERIPHERAL_BUS peripheralBusNumber);
```

Returns

- true - The peripheral bus clock output is enabled
- false - The peripheral bus clock output is disabled

Description

This function checks whether or not the peripheral bus clock output is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsPBClockOutputEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_OSC_PBOutputClockIsEnabled(OSC_ID_0, OSC_PERIPHERAL_BUS_2))
{
    PLIB_OSC_PBOutputClockDisable(OSC_ID_0, OSC_PERIPHERAL_BUS_2);
}
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
peripheralBusNumber	Identifies the desired peripheral bus

Function

bool PLIB_OSC_PBOutputClockIsEnabled(OSC_MODULE_ID index,
OSC_PERIPHERAL_BUS peripheralBusNumber)

j) Clock Functions

PLIB_OSC_ClockHasFailed Function

Returns 'true' if the clock fails.

File

plib_osc.h

C

```
bool PLIB_OSC_ClockHasFailed(OSC_MODULE_ID index);
```

Returns

- true - The FSCM detected a clock failure
- false - The no clock failure has been detected

Description

This function returns 'true' if the clock fails. Monitors the Fail-Safe Clock Monitor (FSCM).

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsClockFail in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool clockStatus;
clockStatus = PLIB_OSC_ClockHasFailed(OSC_ID_0);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module

Function

bool PLIB_OSC_ClockHasFailed (OSC_MODULE_ID index);

PLIB_OSC_ClockStart Function

Starts the specified clock source.

File

plib_osc.h

C

```
void PLIB_OSC_ClockStart(OSC_MODULE_ID index, OSC_CLOCK_DIAG clk);
```

Returns

None.

Description

This function is used to start a specified clock source if it has already been stopped.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsClockDiagStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ClockStart(OSC_ID_0, OSC_CLOCK_POSC_STOP);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
clk	Clock to be started

Function

void PLIB_OSC_ClockStart (OSC_MODULE_ID index, OSC_CLOCK_DIAG clk)

PLIB_OSC_ClockStop Function

Stops the specified clock source.

File

plib_osc.h

C

```
void PLIB_OSC_ClockStop(OSC_MODULE_ID index, OSC_CLOCK_DIAG clk);
```

Returns

None.

Description

This function is used to stop a specified clock source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsClockDiagStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_OSC_ClockStop(OSC_ID_0, OSC_CLOCK_POSC_STOP);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
clk	Clock to be stopped

Function

void PLIB_OSC_ClockStop (OSC_MODULE_ID index, OSC_CLOCK_DIAG clk)

PLIB_OSC_ClockStopStatus Function

returns the status of clock stop bit for the specified clock source.

File

plib_osc.h

C

```
bool PLIB_OSC_ClockStopStatus(OSC_MODULE_ID index, OSC_CLOCK_DIAG clk);
```

Returns

Status of the clock stop bit for a specified clock source.

Description

This function is used to get the status of clock stop bit for the specified clock source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_OSC_ExistsClockDiagStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool PoscStat = PLIB_OSC_ClockStart(OSC_ID_0, OSC_CLOCK_POSC_STOP);
```

Parameters

Parameters	Description
index	Identifies the desired oscillator module
clk	Clock source whose status to be checked

Function

bool PLIB_OSC_ClockStopStatus (OSC_MODULE_ID index, OSC_CLOCK_DIAG clk)

k) Feature Existence Functions

PLIB_OSC_ExistsClockFail Function

Identifies whether the ClockFail feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsClockFail(OSC_MODULE_ID index);
```

Returns

- true - If the ClockFail feature is supported on the device
- false - If the ClockFail feature is not supported on the device

Description

This function identifies whether the ClockFail feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ClockHasFailed

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsClockFail(OSC_MODULE_ID index)

PLIB_OSC_ExistsFRCDivisor Function

Identifies whether the FRCDivisor feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsFRCDivisor(OSC_MODULE_ID index);
```

Returns

- true - If the FRCDivisor feature is supported on the device
- false - If the FRCDivisor feature is not supported on the device

Description

This function identifies whether the FRCDivisor feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_FRCDivisorSelect
- PLIB_OSC_FRCDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsFRCDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsFRCTuning Function

Identifies whether the FRCTuning feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsFRCTuning(OSC_MODULE_ID index);
```

Returns

- true - If the FRCTuning feature is supported on the device
- false - If the FRCTuning feature is not supported on the device

Description

This function identifies whether the FRCTuning feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_FRCTuningSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsFRCTuning(OSC_MODULE_ID index)

PLIB_OSC_ExistsOnWaitAction Function

Identifies whether the OnWaitAction feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsOnWaitAction(OSC_MODULE_ID index);
```

Returns

- true - If the OnWaitAction feature is supported on the device
- false - If the OnWaitAction feature is not supported on the device

Description

This function identifies whether the OnWaitAction feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_OnWaitActionSet
- PLIB_OSC_OnWaitActionGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsOnWaitAction(OSC_MODULE_ID index)

PLIB_OSC_ExistsOscCurrentGet Function

Identifies whether the OscCurrentGet feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsOscCurrentGet(OSC_MODULE_ID index);
```

Returns

- true - If the OscCurrentGet feature is supported on the device
- false - If the OscCurrentGet feature is not supported on the device

Description

This function identifies whether the OscCurrentGet feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_CurrentSysClockGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsOscCurrentGet(OSC_MODULE_ID index)

PLIB_OSC_ExistsOscSelect Function

Identifies whether the OscSelect feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsOscSelect(OSC_MODULE_ID index);
```

Returns

- true - If the OscSelect feature is supported on the device
- false - If the OscSelect feature is not supported on the device

Description

This function identifies whether the OscSelect feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SysClockSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsOscSelect(OSC_MODULE_ID index)

PLIB_OSC_ExistsOscSwitchInit Function

Identifies whether the OscSwitchInit feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsOscSwitchInit(OSC_MODULE_ID index);
```

Returns

- true - If the OscSwitchInit feature is supported on the device
- false - If the OscSwitchInit feature is not supported on the device

Description

This function identifies whether the OscSwitchInit feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ClockSwitchingAbort
- PLIB_OSC_ClockSwitchingIsComplete

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsOscSwitchInit(OSC_MODULE_ID index)

PLIB_OSC_ExistsPBClockDivisor Function

Identifies whether the PBClockDivisor feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsPBClockDivisor(OSC_MODULE_ID index);
```

Returns

- true - If the PBClockDivisor feature is supported on the device
- false - If the PBClockDivisor feature is not supported on the device

Description

This function identifies whether the PBClockDivisor feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_PBClockDivisorGet
- PLIB_OSC_PBClockDivisorSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsPBClockDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsPBClockOutputEnable Function

Identifies whether the PBClockOutputEnable feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsPBClockOutputEnable(OSC_MODULE_ID index);
```

Returns

- true - If the PBClockOutputEnable feature is supported on the device
- false - If the PBClockOutputEnable feature is not supported on the device

Description

This function identifies whether the PBClockOutputEnable feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_PBOutputClockEnable
- PLIB_OSC_PBOutputClockDisable
- PLIB_OSC_PBOutputClockIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsPBClockOutputEnable(OSC_MODULE_ID index)

PLIB_OSC_ExistsPBClockReady Function

Identifies whether the PBClockReady feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsPBClockReady(OSC_MODULE_ID index);
```

Returns

- true - If the PBClockReady feature is supported on the device
- false - If the PBClockReady feature is not supported on the device

Description

This function identifies whether the PBClockReady feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_PBClockDivisorIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsPBClockReady(OSC_MODULE_ID index)

PLIB_OSC_ExistsPLLClockLock Function

Identifies whether the PLLClockLock feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsPLLClockLock(OSC_MODULE_ID index);
```

Returns

- true - If the PLLClockLock feature is supported on the device
- false - If the PLLClockLock feature is not supported on the device

Description

This function identifies whether the PLLClockLock feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_PLLClockLock
- PLIB_OSC_PLLClockUnlock
- PLIB_OSC_PLLClockIsLocked

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsPLLClockLock(OSC_MODULE_ID index)

PLIB_OSC_ExistsPLLLockStatus Function

Identifies whether the PLLLockStatus feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsPLLLockStatus(OSC_MODULE_ID index);
```

Returns

- true - If the PLLLockStatus feature is supported on the device
- false - If the PLLLockStatus feature is not supported on the device

Description

This function identifies whether the PLLLockStatus feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_PLLIsLocked

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsPLLLockStatus(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscBaseClock Function

Identifies whether the ReferenceOscBaseClock feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscBaseClock(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscBaseClock feature is supported on the device
- false - If the ReferenceOscBaseClock feature is not supported on the device

Description

This function identifies whether the ReferenceOscBaseClock feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscBaseClockSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscBaseClock(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscChange Function

Identifies whether the ReferenceOscChange feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscChange(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscChange feature is supported on the device
- false - If the ReferenceOscChange feature is not supported on the device

Description

This function identifies whether the ReferenceOscChange feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscSwitchIsComplete

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscChange(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscChangeActive Function

Identifies whether the ReferenceOscChangeActive feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscChangeActive(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscChangeActive feature is supported on the device
- false - If the ReferenceOscChangeActive feature is not supported on the device

Description

This function identifies whether the ReferenceOscChangeActive feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscSourceChangeIsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscChangeActive(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscDivisor Function

Identifies whether the ReferenceOscDivisor feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscDivisor(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscDivisor feature is supported on the device
- false - If the ReferenceOscDivisor feature is not supported on the device

Description

This function identifies whether the ReferenceOscDivisor feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscDivisorValueSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscEnable Function

Identifies whether the ReferenceOscEnable feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscEnable(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscEnable feature is supported on the device
- false - If the ReferenceOscEnable feature is not supported on the device

Description

This function identifies whether the ReferenceOscEnable feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscEnable
- PLIB_OSC_ReferenceOscDisable
- PLIB_OSC_ReferenceOscIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscEnable(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscStopInIdleEnable Function

Identifies whether the ReferenceOscStopInIdleEnable feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscStopInIdleEnable(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscStopInIdleEnable feature is supported on the device
- false - If the ReferenceOscStopInIdleEnable feature is not supported on the device

Description

This function identifies whether the ReferenceOscStopInIdleEnable feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscStopInIdleEnable
- PLIB_OSC_ReferenceOscStopInIdleDisable
- PLIB_OSC_ReferenceOscStopInIdleIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscStopInIdleEnable(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscStopInSleep Function

Identifies whether the ReferenceOscStopInSleep feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscStopInSleep(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscStopInSleep feature is supported on the device
- false - If the ReferenceOscStopInSleep feature is not supported on the device

Description

This function identifies whether the ReferenceOscStopInSleep feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscStopInSleepEnable
- PLIB_OSC_ReferenceOscStopInSleepDisable

- PLIB_OSC_ReferenceOscStopInSleepIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscStopInSleep(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOscTrim Function

Identifies whether the ReferenceOscTrim feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOscTrim(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOscTrim feature is supported on the device
- false - If the ReferenceOscTrim feature is not supported on the device

Description

This function identifies whether the ReferenceOscTrim feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOscTrimSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOscTrim(OSC_MODULE_ID index)

PLIB_OSC_ExistsReferenceOutputEnable Function

Identifies whether the ReferenceOutputEnable feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsReferenceOutputEnable(OSC_MODULE_ID index);
```

Returns

- true - If the ReferenceOutputEnable feature is supported on the device
- false - If the ReferenceOutputEnable feature is not supported on the device

Description

This function identifies whether the ReferenceOutputEnable feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ReferenceOutputEnable
- PLIB_OSC_ReferenceOutputDisable
- PLIB_OSC_ReferenceOutputIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsReferenceOutputEnable(OSC_MODULE_ID index)

PLIB_OSC_ExistsSecondaryEnable Function

Identifies whether the SecondaryEnable feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSecondaryEnable(OSC_MODULE_ID index);
```

Returns

- true - If the SecondaryEnable feature is supported on the device
- false - If the SecondaryEnable feature is not supported on the device

Description

This function identifies whether the SecondaryEnable feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SecondaryEnable
- PLIB_OSC_SecondaryDisable
- PLIB_OSC_SecondaryIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSecondaryEnable(OSC_MODULE_ID index)

PLIB_OSC_ExistsSecondaryReady Function

Identifies whether the SecondaryReady feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSecondaryReady(OSC_MODULE_ID index);
```

Returns

- true - If the SecondaryReady feature is supported on the device
- false - If the SecondaryReady feature is not supported on the device

Description

This function identifies whether the SecondaryReady feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SecondaryIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSecondaryReady(OSC_MODULE_ID index)

PLIB_OSC_ExistsSysPLLFrequencyRange Function

Identifies whether the PLLFrequencyRange feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSysPLLFrequencyRange(OSC_MODULE_ID index);
```

Returns

- true - If the PLLFrequencyRange feature is supported on the device
- false - If the PLLFrequencyRange feature is not supported on the device

Description

This function identifies whether the PLLFrequencyRange feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SysPLLFrequencyRangeSet
- PLIB_OSC_SysPLLFrequencyRangeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSysPLLFrequencyRange(OSC_MODULE_ID index)

PLIB_OSC_ExistsSysPLLInputClockSource Function

Identifies whether the PLLInputClockSource feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSysPLLInputClockSource(OSC_MODULE_ID index);
```

Returns

- true - If the PLLInputClockSource feature is supported on the device
- false - If the PLLInputClockSource feature is not supported on the device

Description

This function identifies whether the PLLInputClockSource feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SysPLLInputClockSourceSet
- PLIB_OSC_SysPLLInputClockSourceGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSysPLLInputClockSource(OSC_MODULE_ID index)

PLIB_OSC_ExistsSysPLLInputDivisor Function

Identifies whether the PLLInputDivisor feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSysPLLInputDivisor(OSC_MODULE_ID index);
```

Returns

- true - If the PLLInputDivisor feature is supported on the device
- false - If the PLLInputDivisor feature is not supported on the device

Description

This function identifies whether the PLLInputDivisor feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SysPLLInputDivisorSet
- PLIB_OSC_SysPLLInputDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSysPLLInputDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsSysPLLMultiplier Function

Identifies whether the PLLMultiplier feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSysPLLMultiplier(OSC_MODULE_ID index);
```

Returns

- true - If the PLLMultiplier feature is supported on the device
- false - If the PLLMultiplier feature is not supported on the device

Description

This function identifies whether the PLLMultiplier feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SysPLLMultiplierSelect
- PLIB_OSC_SysPLLMultiplierGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSysPLLMultiplier(OSC_MODULE_ID index)

PLIB_OSC_ExistsSysPLLOutputDivisor Function

Identifies whether the PLLOutputDivisor feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSysPLLOutputDivisor(OSC_MODULE_ID index);
```

Returns

- true - If the PLLOutputDivisor feature is supported on the device
- false - If the PLLOutputDivisor feature is not supported on the device

Description

This function identifies whether the PLLOutputDivisor feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SysPLLOutputDivisorSet
- PLIB_OSC_SysPLLOutputDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSysPLLOutputDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsUsbClockSource Function

Identifies whether the UsbClockSource feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsUsbClockSource(OSC_MODULE_ID index);
```

Returns

- true - If the UsbClockSource feature is supported on the device
- false - If the UsbClockSource feature is not supported on the device

Description

This function identifies whether the UsbClockSource feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_UsbClockSourceSelect
- PLIB_OSC_UsbClockSourceGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsUsbClockSource(OSC_MODULE_ID index)

PLIB_OSC_ExistsClockReadyStatus Function

Identifies whether the ClockReadyStatus feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsClockReadyStatus(OSC_MODULE_ID index);
```

Returns

- true - The ClockReadyStatus feature is supported on the device
- false - The ClockReadyStatus feature is not supported on the device

Description

This function identifies whether the ClockReadyStatus feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ClockIsReady

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsClockReadyStatus(OSC_MODULE_ID index)

PLIB_OSC_ExistsClockSlewingStatus Function

Identifies whether the ClockSlewingStatus feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsClockSlewingStatus(OSC_MODULE_ID index);
```

Returns

- true - The ClockSlewingStatus feature is supported on the device
- false - The ClockSlewingStatus feature is not supported on the device

Description

This function identifies whether the ClockSlewingStatus feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ClockSlewingIsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsClockSlewingStatus(OSC_MODULE_ID index)

PLIB_OSC_ExistsSleepToStartupClock Function

Identifies whether the SleepToStartupClock feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSleepToStartupClock(OSC_MODULE_ID index);
```

Returns

- true - The SleepToStartupClock feature is supported on the device
- false - The SleepToStartupClock feature is not supported on the device

Description

This function identifies whether the SleepToStartupClock feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SleepToStartupClockSelect
- PLIB_OSC_SleepToStartupClockGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSleepToStartupClock(OSC_MODULE_ID index)

PLIB_OSC_ExistsSlewDivisorStepControl Function

Identifies whether the SlewDivisorStepControl feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSlewDivisorStepControl(OSC_MODULE_ID index);
```

Returns

- true - The SlewDivisorStepControl feature is supported on the device
- false - The SlewDivisorStepControl feature is not supported on the device

Description

This function identifies whether the SlewDivisorStepControl feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SlewDivisorStepSelect
- PLIB_OSC_SlewDivisorStepGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSlewDivisorStepControl(OSC_MODULE_ID index)

PLIB_OSC_ExistsSlewEnableControl Function

Identifies whether the SlewEnableControl feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSlewEnableControl(OSC_MODULE_ID index);
```

Returns

- true - The SlewEnableControl feature is supported on the device
- false - The SlewEnableControl feature is not supported on the device

Description

This function identifies whether the SlewEnableControl feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SlewEnable
- PLIB_OSC_SlewDisable
- PLIB_OSC_SlewIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSlewEnableControl(OSC_MODULE_ID index)

PLIB_OSC_ExistsSystemClockDivisorControl Function

Identifies whether the SystemClockDivisorControl feature exists on the Oscillator module.

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsSystemClockDivisorControl(OSC_MODULE_ID index);
```

Returns

- true - The SystemClockDivisorControl feature is supported on the device
- false - The SystemClockDivisorControl feature is not supported on the device

Description

This function identifies whether the SystemClockDivisorControl feature is available on the Oscillator module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_SystemClockDivisorSelect
- PLIB_OSC_SystemClockDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsSystemClockDivisorControl(OSC_MODULE_ID index)

PLIB_OSC_ExistsBTPLLClockOut Function

Identifies whether the BTPLLClockOut feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsBTPLLClockOut(OSC_MODULE_ID index);
```

Returns

- true - The BTPLLClockOut feature is supported on the device
- false - The BTPLLClockOut feature is not supported on the device

Description

This function identifies whether the BTPLLClockOut feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_BTPLLClockOutEnable
- PLIB_OSC_BTPLLClockOutDisable
- PLIB_OSC_BTPLLClockOutStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsBTPLLClockOut(OSC_MODULE_ID index)

PLIB_OSC_ExistsBTPLLFrequencyRange Function

Identifies whether the BTPLLFrequencyRange feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsBTPLLFrequencyRange(OSC_MODULE_ID index);
```

Returns

- true - The BTPLLFrequencyRange feature is supported on the device
- false - The BTPLLFrequencyRange feature is not supported on the device

Description

This function identifies whether the BTPLLFrequencyRange feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_BTPLLFrequencyRangeSet
- PLIB_OSC_BTPLLFrequencyRangeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsBTPLLFrequencyRange(OSC_MODULE_ID index)

PLIB_OSC_ExistsBTPLLInputClockSource Function

Identifies whether the BTPLLInputClockSource feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsBTPLLInputClockSource(OSC_MODULE_ID index);
```

Returns

- true - The BTPLLInputClockSource feature is supported on the device
- false - The BTPLLInputClockSource feature is not supported on the device

Description

This function identifies whether the BTPLLInputClockSource feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_BTPLLInputClockSourceSet
- PLIB_OSC_BTPLLInputClockSourceGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsBTPLLInputClockSource(OSC_MODULE_ID index)

PLIB_OSC_ExistsBTPLLInputDivisor Function

Identifies whether the BTPLLInputDivisor feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsBTPLLInputDivisor(OSC_MODULE_ID index);
```

Returns

- true - The BTPLLInputDivisor feature is supported on the device
- false - The BTPLLInputDivisor feature is not supported on the device

Description

This function identifies whether the BTPLLInputDivisor feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_BTPLLInputDivisorSet
- PLIB_OSC_BTPLLInputDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsBTPLLInputDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsBTPLLMultiplier Function

Identifies whether the BTPLLMultiplier feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsBTPLLMultiplier(OSC_MODULE_ID index);
```

Returns

- true - The BTPLLMultiplier feature is supported on the device
- false - The BTPLLMultiplier feature is not supported on the device

Description

This function identifies whether the BTPLLMultiplier feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_BTPLLMultiplierSelect
- PLIB_OSC_BTPLLMultiplierGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsBTPLLMultiplier(OSC_MODULE_ID index)

PLIB_OSC_ExistsBTPLLOutputDivisor Function

Identifies whether the BTPLLOutputDivisor feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsBTPLLOutputDivisor(OSC_MODULE_ID index);
```

Returns

- true - The BTPLLOutputDivisor feature is supported on the device

- false - The BTPLLOutputDivisor feature is not supported on the device

Description

This function identifies whether the BTPLLOutputDivisor feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_BTPLLOutputDivisorSet
- PLIB_OSC_BTPLLOutputDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsBTPLLOutputDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsClockDiagStatus Function

Identifies whether the ClockDiagStatus feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsClockDiagStatus(OSC_MODULE_ID index);
```

Returns

- true - The ClockDiagStatus feature is supported on the device
- false - The ClockDiagStatus feature is not supported on the device

Description

This function identifies whether the ClockDiagStatus feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ClockStop
- PLIB_OSC_ClockStart
- PLIB_OSC_ClockStopStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsClockDiagStatus(OSC_MODULE_ID index)

PLIB_OSC_ExistsDreamModeControl Function

Identifies whether the DreamModeControl feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsDreamModeControl(OSC_MODULE_ID index);
```

Returns

- true - The DreamModeControl feature is supported on the device
- false - The DreamModeControl feature is not supported on the device

Description

This function identifies whether the DreamModeControl feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_DreamModeEnable
- PLIB_OSC_DreamModeDisable
- PLIB_OSC_DreamModeStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsDreamModeControl(OSC_MODULE_ID index)

PLIB_OSC_ExistsForceLock Function

Identifies whether the ForceLock feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsForceLock(OSC_MODULE_ID index);
```

Returns

- true - The ForceLock feature is supported on the device
- false - The ForceLock feature is not supported on the device

Description

This function identifies whether the ForceLock feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ForceSPLLLockEnable
- PLIB_OSC_ForceSPLLLockDisable
- PLIB_OSC_ForceSPLLLockStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsForceLock(OSC_MODULE_ID index)

PLIB_OSC_ExistsPLLBypass Function

Identifies whether the SPLLBypass feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsPLLBypass(OSC_MODULE_ID index);
```

Returns

- true - The PLLBypass feature is supported on the device
- false - The PLLBypass feature is not supported on the device

Description

This function identifies whether the PLLBypass feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_PLLBypassEnable
- PLIB_OSC_PLLBypassDisable
- PLIB_OSC_PLLBypassStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsPLLBypass(OSC_MODULE_ID index)

PLIB_OSC_ExistsResetPLL Function

Identifies whether the ResetPLL feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsResetPLL(OSC_MODULE_ID index);
```

Returns

- true - The ResetPLL feature is supported on the device
- false - The ResetPLL feature is not supported on the device

Description

This function identifies whether the ResetPLL feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_ResetPLLAssert
- PLIB_OSC_ResetPLLDeassert
- PLIB_OSC_ResetPLLStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsResetPLL(OSC_MODULE_ID index)

PLIB_OSC_ExistsUPLLFrequencyRange Function

Identifies whether the UPLLFrequencyRange feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsUPLLFrequencyRange(OSC_MODULE_ID index);
```

Returns

- true - The UPLLFrequencyRange feature is supported on the device
- false - The UPLLFrequencyRange feature is not supported on the device

Description

This function identifies whether the UPLLFrequencyRange feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_UPLLFrequencyRangeSet
- PLIB_OSC_UPLLFrequencyRangeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsUPLLFrequencyRange(OSC_MODULE_ID index)

PLIB_OSC_ExistsUPLLInputDivisor Function

Identifies whether the UPLLInputDivisor feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsUPLLInputDivisor(OSC_MODULE_ID index);
```

Returns

- true - The UPLLInputDivisor feature is supported on the device
- false - The UPLLInputDivisor feature is not supported on the device

Description

This function identifies whether the UPLLInputDivisor feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_UPLLInputDivisorSet
- PLIB_OSC_UPLLInputDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsUPLLInputDivisor(OSC_MODULE_ID index)

PLIB_OSC_ExistsUPLLMultiplier Function

Identifies whether the UPLLMultiplier feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsUPLLMultiplier(OSC_MODULE_ID index);
```

Returns

- true - The UPLLMultiplier feature is supported on the device
- false - The UPLLMultiplier feature is not supported on the device

Description

This function identifies whether the UPLLMultiplier feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_UPLLMultiplierSelect
- PLIB_OSC_UPLLMultiplierGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsUPLLMultiplier(OSC_MODULE_ID index)

PLIB_OSC_ExistsUPLLOutputDivisor Function

Identifies whether the UPLLOutputDivisor feature exists on the OSC module

File

plib_osc.h

C

```
bool PLIB_OSC_ExistsUPLLOutputDivisor(OSC_MODULE_ID index);
```

Returns

- true - The UPLLOutputDivisor feature is supported on the device
- false - The UPLLOutputDivisor feature is not supported on the device

Description

This function identifies whether the UPLLOutputDivisor feature is available on the OSC module. When this function returns true, these functions are supported on the device:

- PLIB_OSC_UPLLOutputDivisorSet
- PLIB_OSC_UPLLOutputDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_OSC_ExistsUPLLOutputDivisor(OSC_MODULE_ID index)

l) Data Types and Constants

OSC_PB_CLOCK_DIV_TYPE Macro

Type of the oscillator PB Clock divisor value.

File

plib_osc_help.h

C

```
#define OSC_PB_CLOCK_DIV_TYPE SFR_TYPE
```

Description

Oscillator Peripheral Bus Clock Divisor Value Type

This macro defines the type of the oscillator PB Clock divisor value.

Remarks

None.

OSC_REF_DIVISOR_TYPE Macro

Reference oscillator divisor type.

File

plib_osc_help.h

C

```
#define OSC_REF_DIVISOR_TYPE SFR_TYPE
```

Description

Reference oscillator divisor type

This macro defines the reference oscillator divisor type. Please refer to the specific device data sheet to determine the possible values of this type.

Remarks

None.

OSC_REFERENCE_MAX_DIV Macro

Defines the reference clock output divisor maximum value.

File

plib_osc_help.h

C

```
#define OSC_REFERENCE_MAX_DIV 65534
```

Description

Reference clock output divisor maximum

This macro defines the reference clock output divisor maximum value.

Remarks

None.

OSC_SYSPLL_MULTIPLIER_TYPE Macro

Type of the oscillator system PLL multiplier value.

File

plib_osc_help.h

C

```
#define OSC_SYSPLL_MULTIPLIER_TYPE SFR_TYPE
```

Description

Oscillator System PLL Multiplier Value Type

This macro defines the type of the oscillator system PLL multiplier value.

Remarks

None.

OSC_FRC_DIV Enumeration

Lists the possible Fast RC (FRC) Oscillator divider values.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_FRC_DIV_1,
  OSC_FRC_DIV_2,
  OSC_FRC_DIV_4,
  OSC_FRC_DIV_8,
  OSC_FRC_DIV_16,
  OSC_FRC_DIV_32,
  OSC_FRC_DIV_64,
  OSC_FRC_DIV_256
} OSC_FRC_DIV;
```

Members

Members	Description
OSC_FRC_DIV_1	Fast RC Oscillator output Divide by 1
OSC_FRC_DIV_2	Fast RC Oscillator output Divide by 2
OSC_FRC_DIV_4	Fast RC Oscillator output Divide by 4
OSC_FRC_DIV_8	Fast RC Oscillator output Divide by 8
OSC_FRC_DIV_16	Fast RC Oscillator output Divide by 16
OSC_FRC_DIV_32	Fast RC Oscillator output Divide by 32
OSC_FRC_DIV_64	Fast RC Oscillator output Divide by 64
OSC_FRC_DIV_256	Fast RC Oscillator output Divide by 256

Description

FRC divider.

This enumeration lists the possible FRC Oscillator divider values.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_MODULE_ID Enumeration

Possible instances of the OSC module.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_ID_0
} OSC_MODULE_ID;
```

Members

Members	Description
OSC_ID_0	first instance of the OSC

Description

OSC module ID

This data type defines the possible instances of the Oscillator module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_OPERATION_ON_WAIT Enumeration

Lists the possible base clock values for the reference oscillator.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_REF_BASECLOCK_SYSCLK,
  OSC_REF_BASECLOCK_PBCLK,
  OSC_REF_BASECLOCK_PRIMARY,
  OSC_REF_BASECLOCK_FRC,
  OSC_REF_BASECLOCK_LPRC,
  OSC_REF_BASECLOCK_SOSC,
  OSC_REF_BASECLOCK_USBCLK,
  OSC_REF_BASECLOCK_SYSPLLOUT,
  OSC_ON_WAIT_IDLE,
  OSC_ON_WAIT_SLEEP
} OSC_OPERATION_ON_WAIT;
```

Members

Members	Description
OSC_REF_BASECLOCK_SYSCLK	The base clock for reference clock is System Clock
OSC_REF_BASECLOCK_PBCLK	The base clock for reference clock is Peripheral Clock
OSC_REF_BASECLOCK_PRIMARY	The base clock for reference clock is Primary Oscillator
OSC_REF_BASECLOCK_FRC	The base clock for reference clock is FRC Oscillator
OSC_REF_BASECLOCK_LPRC	The base clock for reference clock is internal Low-Power RC
OSC_REF_BASECLOCK_SOSC	The base clock for reference clock is Secondary Oscillator
OSC_REF_BASECLOCK_USBCLK	The base clock for reference clock is USB Clock
OSC_REF_BASECLOCK_SYSPLLOUT	The base clock for reference clock is System PLL output

OSC_ON_WAIT_IDLE	Idle on Wait Instruction
OSC_ON_WAIT_SLEEP	Idle on Sleep Instruction

Description

Base clock for reference oscillator.

This enumeration lists the possible base clock values for the reference oscillator.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_PERIPHERAL_BUS Enumeration

Lists the possible Peripheral buses.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_PERIPHERAL_BUS_1,
  OSC_PERIPHERAL_BUS_2,
  OSC_PERIPHERAL_BUS_3,
  OSC_PERIPHERAL_BUS_4,
  OSC_PERIPHERAL_BUS_5,
  OSC_PERIPHERAL_BUS_6,
  OSC_PERIPHERAL_BUS_7,
  OSC_PERIPHERAL_BUS_8
} OSC_PERIPHERAL_BUS;
```

Members

Members	Description
OSC_PERIPHERAL_BUS_1	Peripheral bus 1
OSC_PERIPHERAL_BUS_2	Peripheral bus 2
OSC_PERIPHERAL_BUS_3	Peripheral bus 3
OSC_PERIPHERAL_BUS_4	Peripheral bus 4
OSC_PERIPHERAL_BUS_5	Peripheral bus 5
OSC_PERIPHERAL_BUS_6	Peripheral bus 6
OSC_PERIPHERAL_BUS_7	Peripheral bus 7
OSC_PERIPHERAL_BUS_8	Peripheral bus 8

Description

Peripheral Buses

This enumeration lists the possible peripheral buses available in the device

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_PLL_SELECT Enumeration

Lists the PLLs in the Oscillator module.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_PLL_SYSTEM,
  OSC_PLL_USB
} OSC_PLL_SELECT;
```

Members

Members	Description
OSC_PLL_SYSTEM	Select system PLL
OSC_PLL_USB	Select USB PLL

Description

Oscillator PLL selection.

This enumeration lists the available PLLs in the Oscillator module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_REFERENCE Enumeration

Lists the possible reference oscillator.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_REFERENCE_1,
  OSC_REFERENCE_2,
  OSC_REFERENCE_3,
  OSC_REFERENCE_4
} OSC_REFERENCE;
```

Members

Members	Description
OSC_REFERENCE_1	Reference Oscillator 1
OSC_REFERENCE_2	Reference Oscillator 2
OSC_REFERENCE_3	Reference Oscillator 3
OSC_REFERENCE_4	Reference Oscillator 4

Description

Reference Oscillators

This enumeration lists the possible reference oscillator instances available in the device.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_SYS_TYPE Enumeration

Lists the possible oscillator type values.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_FRC,
  OSC_FRC_WITH_PLL,
  OSC_PRIMARY,
  OSC_PRIMARY_WITH_PLL,
  OSC_SECONDARY,
  OSC_LPRC,
  OSC_FRC_DIV_BY_16,
  OSC_FRC_BY_FRCDIV
} OSC_SYS_TYPE;
```

Members

Members	Description
OSC_FRC	Fast RC Oscillator. This includes the 8 MHz oscillator
OSC_FRC_WITH_PLL	Fast RC Oscillator with PLL
OSC_PRIMARY	Primary Oscillator
OSC_PRIMARY_WITH_PLL	Fast RC Oscillator with PLL and PostScaler. Includes the 8MHz oscillator
OSC_SECONDARY	Secondary Oscillator
OSC_LPRC	Low Power RC Oscillator
OSC_FRC_DIV_BY_16	Low-Power Fast RC Oscillator with PostScaler
OSC_FRC_BY_FRCDIV	Fast RC Oscillator divided by FRCDIV bits

Description

Oscillator type.

This enumeration lists the possible oscillator type values.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_SYSPLL_FREQ_RANGE Enumeration

Lists the possible PLL frequency range.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_SYSPLL_FREQ_RANGE_BYPASS,
  OSC_SYSPLL_FREQ_RANGE_5M_TO_10M,
  OSC_SYSPLL_FREQ_RANGE_8M_TO_16M,
  OSC_SYSPLL_FREQ_RANGE_13M_TO_26M,
  OSC_SYSPLL_FREQ_RANGE_21M_TO_42M,
  OSC_SYSPLL_FREQ_RANGE_34M_TO_68M
} OSC_SYSPLL_FREQ_RANGE;
```

Members

Members	Description
OSC_SYSPLL_FREQ_RANGE_BYPASS	PLL freq range bypass
OSC_SYSPLL_FREQ_RANGE_5M_TO_10M	PLL frequency range is 5 MHz to 10 MHz
OSC_SYSPLL_FREQ_RANGE_8M_TO_16M	PLL frequency range is 8 MHz to 16 MHz
OSC_SYSPLL_FREQ_RANGE_13M_TO_26M	PLL frequency range is 13 MHz to 26 MHz
OSC_SYSPLL_FREQ_RANGE_21M_TO_42M	PLL frequency range is 21 MHz to 42 MHz
OSC_SYSPLL_FREQ_RANGE_34M_TO_68M	PLL frequency range is 34 MHz to 68 MHz

Description

System PLL Frequency Range

This enumeration lists the possible frequency range for PLL module available in the device.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_SYSPLL_IN_CLK_SOURCE Enumeration

Lists the possible input clock source for PLL module.

File

plib_osc_help.h

C

```
typedef enum {
```

```
  OSC_SYSPLL_IN_CLK_SOURCE_FRC,
  OSC_SYSPLL_IN_CLK_SOURCE_PRIMARY
} OSC_SYSPLL_IN_CLK_SOURCE;
```

Members

Members	Description
OSC_SYSPLL_IN_CLK_SOURCE_FRC	FRC is input clock source for PLL module
OSC_SYSPLL_IN_CLK_SOURCE_PRIMARY	Primary clock is the input clock source for PLL module

Description

System PLL Input Clock Source

This enumeration lists the possible input clock source for PLL module available in the device.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_SYSPLL_OUT_DIV Enumeration

Lists the possible PLL output divider values.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_SYSPLL_OUT_DIV_1,
  OSC_SYSPLL_OUT_DIV_2,
  OSC_SYSPLL_OUT_DIV_4,
  OSC_SYSPLL_OUT_DIV_8,
  OSC_SYSPLL_OUT_DIV_16,
  OSC_SYSPLL_OUT_DIV_32,
  OSC_SYSPLL_OUT_DIV_64,
  OSC_SYSPLL_OUT_DIV_256
} OSC_SYSPLL_OUT_DIV;
```

Members

Members	Description
OSC_SYSPLL_OUT_DIV_1	Divide by 1
OSC_SYSPLL_OUT_DIV_2	Divide by 2
OSC_SYSPLL_OUT_DIV_4	Divide by 4
OSC_SYSPLL_OUT_DIV_8	Divide by 8
OSC_SYSPLL_OUT_DIV_16	Divide by 16
OSC_SYSPLL_OUT_DIV_32	Divide by 32
OSC_SYSPLL_OUT_DIV_64	Divide by 64
OSC_SYSPLL_OUT_DIV_256	Divide by 256

Description

PLL Output divider.

This enumeration lists the possible PLL output divider values.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_USBCLOCK_SOURCE Enumeration

Lists the possible USB clock sources.

File

plib_osc_help.h

C

```
typedef enum {
```

```
  SYS_OSC_USBCLK_PRIMARY,
  SYS_OSC_USBCLK_FRC
} OSC_USBCLOCK_SOURCE;
```

Members

Members	Description
SYS_OSC_USBCLK_PRIMARY	USB clock source is primary oscillator
SYS_OSC_USBCLK_FRC	USB clock source isFRC oscillator

Description

USB clock sources.

This enumeration lists the the possible USB clock sources.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files.

OSC_CLOCK_ID Enumeration

Lists the clock sources for which ready status can be checked.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_CLOCK_FAST_RC,
  OSC_CLOCK_PRIMARY_OSC,
  OSC_CLOCK_SECONDARY_OSC,
  OSC_CLOCK_LOW_POWER_RC,
  OSC_CLOCK_SYSTEM_PLL
} OSC_CLOCK_ID;
```

Members

Members	Description
OSC_CLOCK_FAST_RC	Fast RC Oscillator
OSC_CLOCK_PRIMARY_OSC	Primary Oscillator
OSC_CLOCK_SECONDARY_OSC	Secondary Oscillator
OSC_CLOCK_LOW_POWER_RC	Low Power RC Oscillator
OSC_CLOCK_SYSTEM_PLL	System PLL

Description

OSC Clock ID

This enumeration lists all the possible clock sources for which the ready status can be checked.

Remarks

This feature may not be available in all the devices. Refer to the specific device header files for availability.

OSC_CLOCK_SLEW_TYPE Enumeration

Lists the possible type of clock slewing.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_CLOCK_SLEW_DOWNWARD,
  OSC_CLOCK_SLEW_UPWARD
} OSC_CLOCK_SLEW_TYPE;
```

Members

Members	Description
OSC_CLOCK_SLEW_DOWNWARD	Slewing to a lower clock frequency
OSC_CLOCK_SLEW_UPWARD	Slewing to a higher clock frequency

Description

OSC Clock Slew Rate

This enumeration lists the possible type of clock slewing.

Remarks

This feature may not be available in all the devices. Refer to the specific device header files for availability.

OSC_SLEEP_TO_STARTUP_CLK_TYPE Enumeration

Lists the possible clock sources for sleep to start-up period.

File

plib_osc_help.h

C

```
typedef enum {
  OSC_SLEEP_TO_STARTUP_CLK_FRC,
  OSC_SLEEP_TO_STARTUP_NO_ADDITIONAL_CLK
} OSC_SLEEP_TO_STARTUP_CLK_TYPE;
```

Members

Members	Description
OSC_SLEEP_TO_STARTUP_CLK_FRC	FRC will be used after wakeup from sleep until selected clock is ready
OSC_SLEEP_TO_STARTUP_NO_ADDITIONAL_CLK	No additional clock will be used after wakeup from sleep, selected clock will be used directly once it is ready

Description

Sleep to Speed Startup Clock

This enumeration lists the possible clock sources that can be used from the time when the device wakes from sleep until the actual clock source is ready to be used.

Remarks

This feature may not be available in all the devices. Check device specific header files for availability.

Files

Files

Name	Description
plib_osc.h	Defines the Oscillator (OSC) Peripheral Library interface.
plib_osc_help.h	Defines the Oscillator Peripheral Library data types.

Description

This section lists the source and header files used by the library.

plib_osc.h

Defines the Oscillator (OSC) Peripheral Library interface.

Functions

	Name	Description
	PLIB_OSC_BTPLLClockOutDisable	Disables the Bluetooth PLL Clock Output.
	PLIB_OSC_BTPLLClockOutEnable	Enables the Bluetooth PLL clock Ouput.
	PLIB_OSC_BTPLLClockOutStatus	gets the status of the Bluetooth PLL clock Output.

	PLIB_OSC_BTPLLFrequencyRangeGet	Gets the frequency range for the Bluetooth PLL module.
	PLIB_OSC_BTPLLFrequencyRangeSet	Sets the frequency range for the Bluetooth PLL module.
	PLIB_OSC_BTPLLInputClockSourceGet	Gets the input clock source for the Bluetooth PLL module.
	PLIB_OSC_BTPLLInputClockSourceSet	Sets the input clock source for the Bluetooth PLL module.
	PLIB_OSC_BTPLLInputDivisorGet	Gets the input divisor for the Bluetooth PLL.
	PLIB_OSC_BTPLLInputDivisorSet	Sets the input divider for the Bluetooth PLL to the specified value.
	PLIB_OSC_BTPLLMultiplierGet	Gets the Bluetooth PLL multiplier.
	PLIB_OSC_BTPLLMultiplierSelect	Sets the Bluetooth PLL multiplier to the specified value.
	PLIB_OSC_BTPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_BTPLLOutputDivisorSet	Sets the output divider for the Bluetooth PLL to the specified value.
	PLIB_OSC_ClockHasFailed	Returns 'true' if the clock fails.
	PLIB_OSC_ClockIsReady	Get the ready status of clock.
	PLIB_OSC_ClockSlewingIsActive	Returns the status of clock slewing.
	PLIB_OSC_ClockStart	Starts the specified clock source.
	PLIB_OSC_ClockStop	Stops the specified clock source.
	PLIB_OSC_ClockStopStatus	returns the status of clock stop bit for the specified clock source.
	PLIB_OSC_ClockSwitchingAbort	Aborts an oscillator switch.
	PLIB_OSC_ClockSwitchingIsComplete	Gets the oscillator switch progress status.
	PLIB_OSC_CurrentSysClockGet	Gets the current oscillator selected.
	PLIB_OSC_DreamModeDisable	Disables the dream mode.
	PLIB_OSC_DreamModeEnable	Enables the dream mode.
	PLIB_OSC_DreamModeStatus	gets the status of the dream mode.
	PLIB_OSC_ExistsBTPLLClockOut	Identifies whether the BTPLLClockOut feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLFrequencyRange	Identifies whether the BTPLLFrequencyRange feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLInputClockSource	Identifies whether the BTPLLInputClockSource feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLInputDivisor	Identifies whether the BTPLLInputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLMultiplier	Identifies whether the BTPLLMultiplier feature exists on the OSC module
	PLIB_OSC_ExistsBTPLLOutputDivisor	Identifies whether the BTPLLOutputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsClockDiagStatus	Identifies whether the ClockDiagStatus feature exists on the OSC module
	PLIB_OSC_ExistsClockFail	Identifies whether the ClockFail feature exists on the Oscillator module.
	PLIB_OSC_ExistsClockReadyStatus	Identifies whether the ClockReadyStatus feature exists on the Oscillator module.
	PLIB_OSC_ExistsClockSlewingStatus	Identifies whether the ClockSlewingStatus feature exists on the Oscillator module.
	PLIB_OSC_ExistsDreamModeControl	Identifies whether the DreamModeControl feature exists on the OSC module
	PLIB_OSC_ExistsForceLock	Identifies whether the ForceLock feature exists on the OSC module
	PLIB_OSC_ExistsFRCDivisor	Identifies whether the FRCDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsFRCTuning	Identifies whether the FRCTuning feature exists on the Oscillator module.
	PLIB_OSC_ExistsOnWaitAction	Identifies whether the OnWaitAction feature exists on the Oscillator module.
	PLIB_OSC_ExistsOscCurrentGet	Identifies whether the OscCurrentGet feature exists on the Oscillator module.
	PLIB_OSC_ExistsOscSelect	Identifies whether the OscSelect feature exists on the Oscillator module.
	PLIB_OSC_ExistsOscSwitchInit	Identifies whether the OscSwitchInit feature exists on the Oscillator module.
	PLIB_OSC_ExistsPBClockDivisor	Identifies whether the PBClockDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsPBClockOutputEnable	Identifies whether the PBClockOutputEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsPBClockReady	Identifies whether the PBClockReady feature exists on the Oscillator module.
	PLIB_OSC_ExistsPLLBypass	Identifies whether the SPLLBypass feature exists on the OSC module
	PLIB_OSC_ExistsPLLClockLock	Identifies whether the PLLClockLock feature exists on the Oscillator module.
	PLIB_OSC_ExistsPLLLockStatus	Identifies whether the PLLLockStatus feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscBaseClock	Identifies whether the ReferenceOscBaseClock feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscChange	Identifies whether the ReferenceOscChange feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscChangeActive	Identifies whether the ReferenceOscChangeActive feature exists on the Oscillator module.

	PLIB_OSC_ExistsReferenceOscDivisor	Identifies whether the ReferenceOscDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscEnable	Identifies whether the ReferenceOscEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscStopInIdleEnable	Identifies whether the ReferenceOscStopInIdleEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscStopInSleep	Identifies whether the ReferenceOscStopInSleep feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOscTrim	Identifies whether the ReferenceOscTrim feature exists on the Oscillator module.
	PLIB_OSC_ExistsReferenceOutputEnable	Identifies whether the ReferenceOutputEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsResetPLL	Identifies whether the ResetPLL feature exists on the OSC module
	PLIB_OSC_ExistsSecondaryEnable	Identifies whether the SecondaryEnable feature exists on the Oscillator module.
	PLIB_OSC_ExistsSecondaryReady	Identifies whether the SecondaryReady feature exists on the Oscillator module.
	PLIB_OSC_ExistsSleepToStartupClock	Identifies whether the SleepToStartupClock feature exists on the Oscillator module.
	PLIB_OSC_ExistsSlewDivisorStepControl	Identifies whether the SlewDivisorStepControl feature exists on the Oscillator module.
	PLIB_OSC_ExistsSlewEnableControl	Identifies whether the SlewEnableControl feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLFrequencyRange	Identifies whether the PLLFrequencyRange feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLInputClockSource	Identifies whether the PLLInputClockSource feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLInputDivisor	Identifies whether the PLLInputDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLMultiplier	Identifies whether the PLLMultiplier feature exists on the Oscillator module.
	PLIB_OSC_ExistsSysPLLOutputDivisor	Identifies whether the PLLOutputDivisor feature exists on the Oscillator module.
	PLIB_OSC_ExistsSystemClockDivisorControl	Identifies whether the SystemClockDivisorControl feature exists on the Oscillator module.
	PLIB_OSC_ExistsUPLLFrequencyRange	Identifies whether the UPLLFrequencyRange feature exists on the OSC module
	PLIB_OSC_ExistsUPLLInputDivisor	Identifies whether the UPLLInputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsUPLLMultiplier	Identifies whether the UPLLMultiplier feature exists on the OSC module
	PLIB_OSC_ExistsUPLLOutputDivisor	Identifies whether the UPLLOutputDivisor feature exists on the OSC module
	PLIB_OSC_ExistsUsbClockSource	Identifies whether the UsbClockSource feature exists on the Oscillator module.
	PLIB_OSC_ForceSPLLLockDisable	Disables the Force PLL Lock feature for specified PLL.
	PLIB_OSC_ForceSPLLLockEnable	Enables the Force PLL Lock feature for specified PLL.
	PLIB_OSC_ForceSPLLLockStatus	gets the status of the force PLL Lock bit of the specified PLL.
	PLIB_OSC_FRCDivisorGet	Gets the FRC clock divisor.
	PLIB_OSC_FRCDivisorSelect	Sets the FRC clock divisor to the specified value.
	PLIB_OSC_FRCTuningSelect	Sets the FRC tuning value.
	PLIB_OSC_OnWaitActionGet	Gets the configured operation to be performed when a WAIT instruction is executed.
	PLIB_OSC_OnWaitActionSet	Selects the operation to be performed when a WAIT instruction is executed.
	PLIB_OSC_PBClockDivisorGet	Gets the peripheral bus clock divisor.
	PLIB_OSC_PBClockDivisorIsReady	Checks whether the peripheral bus clock divisor is ready to be written.
	PLIB_OSC_PBClockDivisorSet	Sets the peripheral bus clock divisor to the specified value.
	PLIB_OSC_PBOutputClockDisable	Disables the peripheral bus output clock.
	PLIB_OSC_PBOutputClockEnable	Enables the peripheral bus output clock
	PLIB_OSC_PBOutputClockIsEnabled	Checks whether or not the peripheral bus clock output is enabled.
	PLIB_OSC_PLLBypassDisable	Disables the PLL Bypass.
	PLIB_OSC_PLLBypassEnable	Enables the PLL Bypass.
	PLIB_OSC_PLLBypassStatus	gets the status of the PLL Bypass.
	PLIB_OSC_PLLClockIsLocked	Gets the lock status for the clock and PLL selections.
	PLIB_OSC_PLLClockLock	Locks the clock and PLL selections.
	PLIB_OSC_PLLClockUnlock	Unlocks the clock and PLL selections.
	PLIB_OSC_PLLIsLocked	Returns 'true' if the selected PLL module is locked.
	PLIB_OSC_ReferenceOscBaseClockSelect	Sets the base clock for the reference oscillator.

	PLIB_OSC_ReferenceOscDisable	Disables the reference oscillator output.
	PLIB_OSC_ReferenceOscDivisorValueSet	Selects the reference oscillator divisor value.
	PLIB_OSC_ReferenceOscEnable	Enables the reference oscillator.
	PLIB_OSC_ReferenceOscIsEnabled	Gets the enable status of the reference oscillator output.
	PLIB_OSC_ReferenceOscSourceChangeIsActive	Returns 'true' if a reference oscillator source change request is active.
	PLIB_OSC_ReferenceOscStopInIdleDisable	Enables the reference oscillator in Idle mode.
	PLIB_OSC_ReferenceOscStopInIdleEnable	Configures the reference oscillator to stop operating in Idle mode.
	PLIB_OSC_ReferenceOscStopInIdleIsEnabled	Returns 'true' if the reference oscillator is disabled in Idle mode.
	PLIB_OSC_ReferenceOscStopInSleepDisable	Enables the reference oscillator in Sleep mode.
	PLIB_OSC_ReferenceOscStopInSleepEnable	Configures the reference oscillator to stop operating in Sleep mode.
	PLIB_OSC_ReferenceOscStopInSleepIsEnabled	Returns 'true' if the reference oscillator is disabled in Sleep mode.
	PLIB_OSC_ReferenceOscSwitchIsComplete	Returns 'true' if the reference oscillator base clock switching is complete.
	PLIB_OSC_ReferenceOscTrimSet	Sets the reference oscillator divisor trim value.
	PLIB_OSC_ReferenceOutputDisable	Disables the reference oscillator output.
	PLIB_OSC_ReferenceOutputEnable	Enables the reference oscillator output.
	PLIB_OSC_ReferenceOutputIsEnabled	Returns 'true' if the reference oscillator output is enabled.
	PLIB_OSC_ResetPLLAssert	Asserts the PLL reset for selected PLL.
	PLIB_OSC_ResetPLLDeassert	Deasserts the PLL reset for selected PLL.
	PLIB_OSC_ResetPLLStatus	gets the status of the PLL reset bit for the specified PLL.
	PLIB_OSC_SecondaryDisable	Disables the Secondary Oscillator.
	PLIB_OSC_SecondaryEnable	Enables the Secondary Oscillator.
	PLIB_OSC_SecondaryIsEnabled	Returns 'true' if the Secondary Oscillator is enabled.
	PLIB_OSC_SecondaryIsReady	Returns 'true' if the Secondary Oscillator is ready.
	PLIB_OSC_SleepToStartupClockGet	Returns the clock used for the duration when the device wakes from sleep and the clock ready.
	PLIB_OSC_SleepToStartupClockSelect	Selects the clock duration for when the device wakes from sleep and the clock is ready.
	PLIB_OSC_SlewDisable	Disables the selected type of slewing.
	PLIB_OSC_SlewDivisorStepGet	Get the slew divisor maximum step.
	PLIB_OSC_SlewDivisorStepSelect	Selects division steps used while slewing.
	PLIB_OSC_SlewEnable	Enables the selected type of slewing.
	PLIB_OSC_SlewIsEnabled	Returns 'true' if the reference oscillator is disabled in Idle mode.
	PLIB_OSC_SysClockSelect	Selects the new oscillator.
	PLIB_OSC_SysPLLFrequencyRangeGet	Gets the frequency range for the PLL module.
	PLIB_OSC_SysPLLFrequencyRangeSet	Sets the frequency range for the PLL module.
	PLIB_OSC_SysPLLInputClockSourceGet	Gets the input clock source for the PLL module.
	PLIB_OSC_SysPLLInputClockSourceSet	Sets the input clock source for the PLL module.
	PLIB_OSC_SysPLLInputDivisorGet	Gets the input divisor for the PLL.
	PLIB_OSC_SysPLLInputDivisorSet	Sets the input divider for the PLL to the specified value.
	PLIB_OSC_SysPLLMultiplierGet	Gets the PLL multiplier.
	PLIB_OSC_SysPLLMultiplierSelect	Sets the PLL multiplier to the specified value.
	PLIB_OSC_SysPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_SysPLLOutputDivisorSet	Sets the output divider for the PLL to the specified value.
	PLIB_OSC_SystemClockDivisorGet	Get the system clock divisor value.
	PLIB_OSC_SystemClockDivisorSelect	Selects system clock divisor.
	PLIB_OSC_UPLLFrequencyRangeGet	Gets the frequency range for the USB PLL module.
	PLIB_OSC_UPLLFrequencyRangeSet	Sets the frequency range for the USB PLL module.
	PLIB_OSC_UPLLInputDivisorGet	Gets the input divisor for the USB PLL.
	PLIB_OSC_UPLLInputDivisorSet	Sets the input divider for the USB PLL to the specified value.
	PLIB_OSC_UPLLMultiplierGet	Gets the USB PLL multiplier.
	PLIB_OSC_UPLLMultiplierSelect	Sets the USB PLL multiplier to the specified value.
	PLIB_OSC_UPLLOutputDivisorGet	Gets the output divisor for the PLL.
	PLIB_OSC_UPLLOutputDivisorSet	Sets the output divider for the USB PLL to the specified value.
	PLIB_OSC_UsbClockSourceGet	Gets the USB module clock source.
	PLIB_OSC_UsbClockSourceSelect	Sets the USB module clock source.

Description

Oscillator Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Oscillator (OSC) Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Oscillator module.

File Name

plib_osc.h

Company

Microchip Technology Inc.

plib_osc_help.h

Defines the Oscillator Peripheral Library data types.

Enumerations

	Name	Description
	OSC_CLOCK_ID	Lists the clock sources for which ready status can be checked.
	OSC_CLOCK_SLEW_TYPE	Lists the possible type of clock slewing.
	OSC_FRC_DIV	Lists the possible Fast RC (FRC) Oscillator divider values.
	OSC_MODULE_ID	Possible instances of the OSC module.
	OSC_OPERATION_ON_WAIT	Lists the possible base clock values for the reference oscillator.
	OSC_PERIPHERAL_BUS	Lists the possible Peripheral buses.
	OSC_PLL_SELECT	Lists the PLLs in the Oscillator module.
	OSC_REFERENCE	Lists the possible reference oscillator.
	OSC_SLEEP_TO_STARTUP_CLK_TYPE	Lists the possible clock sources for sleep to start-up period.
	OSC_SYS_TYPE	Lists the possible oscillator type values.
	OSC_SYSPLL_FREQ_RANGE	Lists the possible PLL frequency range.
	OSC_SYSPLL_IN_CLK_SOURCE	Lists the possible input clock source for PLL module.
	OSC_SYSPLL_OUT_DIV	Lists the possible PLL output divider values.
	OSC_USBCLOCK_SOURCE	Lists the possible USB clock sources.

Macros

	Name	Description
	OSC_PB_CLOCK_DIV_TYPE	Type of the oscillator PB Clock divisor value.
	OSC_REF_DIVISOR_TYPE	Reference oscillator divisor type.
	OSC_REFERENCE_MAX_DIV	Defines the reference clock output divisor maximum value.
	OSC_SYSPLL_MULTIPLIER_TYPE	Type of the oscillator system PLL multiplier value.

Description

Oscillator Peripheral Library Interface Header

This header file contains the definitions of the data types and constants that make up the interface to the Oscillator Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Oscillator module.

File Name

plib_osc_help.h

Company

Microchip Technology Inc.

PMP Peripheral Library

This section describes the Parallel Master Port (PMP) Peripheral Library.

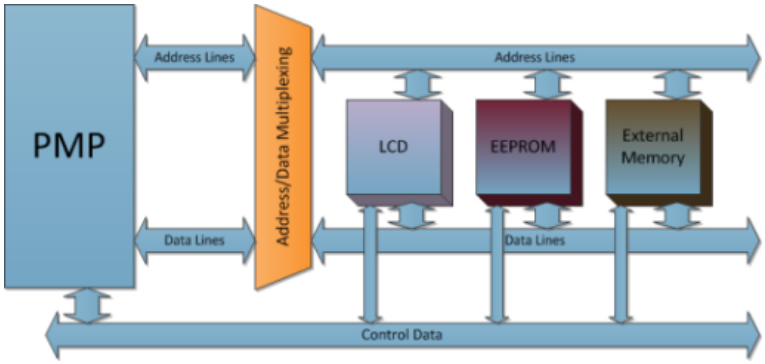
Introduction

This library provides a low-level abstraction of the Parallel Master Port (PMP) module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby

hiding differences from one microcontroller variant to another.

Description

The following diagram shows a block diagram of the PMP module and how it interacts with other peripherals.



Using the Library

This topic describes the basic architecture of the PMP Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_pmp.h`

The interface to the PMP Peripheral library is defined in the `plib_pmp.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the PMP Peripheral Library must include `peripheral.h`.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

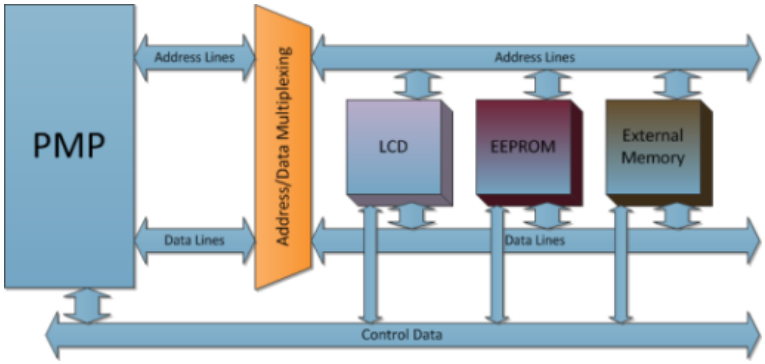
Hardware Abstraction Model

This library provides a low-level abstraction of the Parallel Master Port (PMP) module on Microchip's microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The PMP module provides interface routines to interact with external peripherals such as LCD, EEPROM, Flash memory, etc., as shown in the following diagram. The diagram shows the PMP module acting as a master. The PMP module can be easily configured to act as a slave. The address and data lines can be multiplexed to suit the application. The address and data buffers are up to 2-byte (16-bit) buffers for data transmitted or received by the parallel interface to the PMP bus over the data and address lines synchronized with control logic including the read and write strobe.

PMP Hardware Abstraction Model Diagram



The desired timing wait states to suit different peripheral timings can also be programmed using the PMP module.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the PMP module

Library Interface Section	Description
General Initialization Functions	Provides configuration for: • Interrupt mode • Address Incrementing mode • Stop-in-Idle operation • Operating mode • Multiplexing mode • Chip Select mode
Wait States Initialization Functions	Provides configuration routines to tune the wait states of the PMP module.
Data Read and Write Functions	Provides functions for receiving the data from another module when operating in Master or Slave mode. Also provides routines for transmitting data from the module when operating in Master or Slave mode.
Address Handling	Provides functions to set the PMP module to point to the desired addresses.
Port Configuration	Provides functions for configuring the PMP pins either for PMP purpose or general purpose.
Polarity Configuration	Provides routines for configuring the PMP pins polarity either as active-high or active-low.
Error Status/Control Functions	Provides functions and status for error handling (either as a master or a slave).
General Status Functions	Provides status of the PMP module.

Note: The enhanced feature APIs are not available on all devices. Please refer to the specific device data sheet to determine the availability of these enhanced features.

How the Library Works

Provides information on how the library works.

Description

Before enabling the PMP module, the caller must initialize basic parameters such as wait states timings, and interrupt mode features (see the Initialization section).

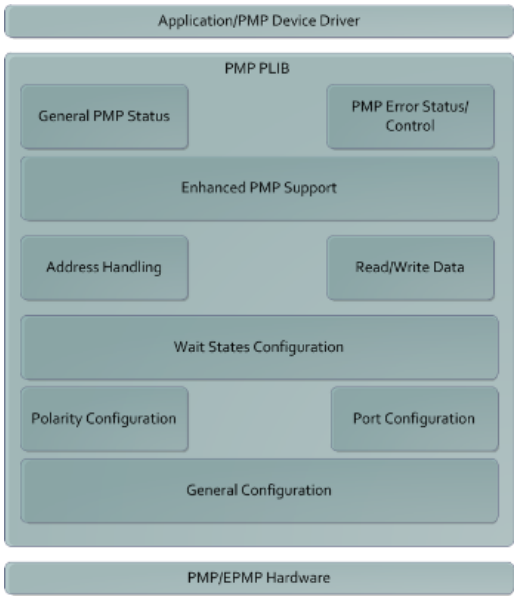
After the module has been enabled, it can begin monitoring the bus as a slave device waiting to be addressed by an external master (see the Operating as a Slave section). A slave device only transfers data on the bus when it has been addressed by a master. If the module is used to start a transfer, it is operating as a master. A master addresses a slave and controls the transfer of data by providing the clock (see the Operating as a Master section).

Some operations in the PMP peripheral library initiate actions on the PMP bus that require time to complete. Also, some events occur asynchronously on the bus. In each of these cases, the module causes either a "master", "slave", or "error" interrupt. The State Machine section describes the different states that can cause an interrupt and show what causes the transition from one state to another. The Handling Errors section describes the various errors that can occur and how to deal with them.

Usage Model

The following diagram shows the high level organization of the PMP Peripheral Library usage model. The items in the diagram correspond to the groupings in the Library Interface section.

PMP Library Usage Model



Initialization

This section describes the general initialization of the PMP module.

Description

The important configurations to look for are as follows.

Operating Mode

Depending on the device, the PMP module can function in Master modes 1 or 2, buffered/enhanced slave mode or the legacy slave mode. To set the PMP module in the appropriate mode, use PLIB_PMP_OperationModeSelect.

Multiplexing Mode

Depending on the application requirement or hardware arrangement, the data lines can be used to multiplex the addressing information. Various multiplexing modes supported are listed by PMP_MUX_MODE. Use PLIB_PMP_MultiplexModeSelect to select the appropriate multiplexing mode.

Chip Select Mode

As needed, the Chip Select lines can be made to function as Chip Select or as address lines. Use PLIB_PMP_ChipSelectFunctionSelect to select the required functionality of Chip Select lines.

Interrupt Mode

PMP generates interrupts based on the option selected for the interrupt mode. Interrupts can be enabled, disabled, generated at the end of a read or write cycle, or when other events occur. Use the PLIB_PMP_InterruptModeSelect to select these options.

Address Increment Mode

After every read/write cycle, the PMP module can be configured to automatically increment or decrement the address it is accessing. Use PLIB_PMP_AddressIncrementModeSelect to select this option.

To initialize the PMP module, perform the following sequence:

1. Select the desired initialization options using the General Initialization Functions (see the Library Interface section) to select the desired operation of the following features:

- Operating mode
- Multiplexing mode
- Chip Select mode
- Interrupt mode
- Address Incrementing mode
- Stop-in-Idle operation

2. Program the wait states of the PMP module. Refer to Wait States Initialization Functions in the Library Interface section for an example.
3. If operating in Master mode, set the desired address (8-bit in this case, the size may vary depending on the application) using PLIB_PMP_AddressSet.
4. If running in an interrupt-driven environment, clear any pending interrupts and enable the appropriate (master, slave, and error) PMP interrupts.

Note: Refer to the **"Interrupt Controller"** chapter in the specific device data sheet for details.

5. Enable the module for operation using PLIB_PMP_Enable.

Example

```
// Configure General PMP Options
// Select the PMP operation mode
PLIB_PMP_OperationModeSelect( PMP_ID_0, PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT  );
// Set the multiplexing to none. Address and data lines are not multiplexed.
PLIB_PMP_MultiplexModeSelect( PMP_ID_0, PMP_MUX_NONE );
// Select the function of chip-select lines
PLIB_PMP_ChipSelectFunctionSelect( PMP_ID_0, PMCS1_PMCS2_AS_ADDRESS_LINES );
// Select the interrupt mode
PLIB_PMP_InterruptModeSelect( PMP_ID_0, PMP_INTERRUPT_NONE );
// disable the auto increment/decrement of address after each read/write
PLIB_PMP_AddressIncrementModeSelect( PMP_ID_0, PMP_ADDRESS_INCREMENT_DECREMENT_DISABLE );
// Enable stop in Idle mode.
PLIB_PMP_StopInIdleEnable( PMP_ID_0 );
// Set Desired Wait State Values
// Set the data wait states
PLIB_PMP_WaitStatesDataSetUpSelect( PMP_ID_0, PMP_DATA_WAIT_FOUR );
// Set the strobe wait states
PLIB_PMP_WaitStatesStrobeSelect( PMP_ID_0, PMP_STROBE_WAIT_1 );
// Set the data hold wait states.
PLIB_PMP_WaitStatesDataHoldSelect( PMP_ID_0, PMP_DATA_HOLD_1 );
// Set the appropriate address (MASTER only)
PLIB_PMP_AddressSet ( PMP_ID_0, 0x12 );

// Optional:  Clear and enable interrupts before enabling the PMP module.
// Enable the module
PLIB_PMP_Enable( PMP_ID_0 );
```

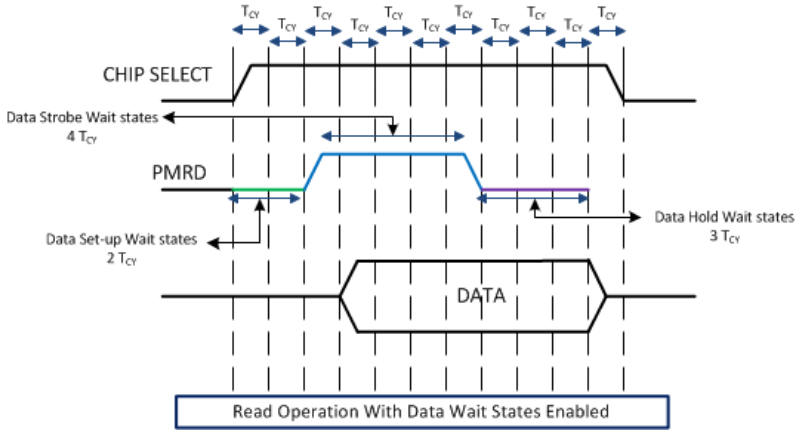
Note: Refer to the **"Interrupt Controller"** chapter in the specific device data sheet for details on how to clear and enable the PMP module interrupts.

Wait States Initialization

This section describes set up and initialization of the wait states for the PMP module.

Description

In Master mode, the user can control the duration of the read, write and address cycles by configuring the module Wait states. One Wait state period is equivalent to one peripheral bus clock cycles. The following figure shows an example of a Master Read operation using Wait states.



Wait States

Wait states can be added to the beginning, middle and end of any read or write cycle using the corresponding bits in the PMP Configuration. The wait states differ depending on which PMP mode you are in, but setting the wait states is the same per mode and device, as shown in the following example.

Initializing Wait States

The following sequence can be used to set up the wait states for the PMP Master mode.

Preconditions:

None.

Process:

1. Set the data hold time using PLIB_PMP_WaitStatesDataHoldSelect.
2. Set the strobe wait time using PLIB_PMP_WaitStatesStrobeSelect.
3. Set the data wait time using PLIB_PMP_WaitStatesDataSetUpSelect.

Example

```
// Set the data wait states
PLIB_PMP_WaitStatesDataSetUpSelect( PMP_ID_0, PMP_DATA_WAIT_ONE );
// Set the strobe wait states
PLIB_PMP_WaitStatesStrobeSelect( PMP_ID_0, PMP_STROBE_WAIT_1 );
// Set the data hold wait states.
PLIB_PMP_WaitStatesDataHoldSelect( PMP_ID_0, PMP_DATA_HOLD_1 );
```

Operating as a Master

This section describes how to set up the PMP module to operate as a master.

Description

The PMP module, while acting as a master, waits for its input or output buffers to be read or written. Once the buffers have been manipulated, the appropriate action takes place.

A PMP bus transfer always begins with reading or writing to the appropriate PMP input/output buffers. A read from a PMP input buffer performs a PMP read. A write to a PMP output buffer performs a PMP write. The address pins associated with the transfer will have the value that is inside the address register of the PMP module. These steps are summarized as follows.

Summary of Steps

- Set up the proper PMP address
- Send or read data bytes

Each of these steps making up a transfer will take some time to complete. By monitoring the busy bit of the PMP module, the user application can determine whether the transfer is complete. The completion of each step can cause a PMP interrupt.

Sending a Data Byte

The following sequence can be used to send a data byte and repeated to send an arbitrary number of data bytes.

Preconditions:

- The address of the destination must be configured in the PMP module
- The proper control signals are configured for PMP operation
- The PMP module is configured for the desired Master mode
- The PMP module is enabled

Process:

1. Ensure that the PMP module is not busy by using PLIB_PMP_PortIsBusy.
2. Write the output data buffer using PLIB_PMP_MasterSend.

Example:

```
uint8_t data = 'a';
// Set the destination address to be written
PLIB_PMP_AddressSet(PMP_ID_0, 0x1234 );
// Check to see if the PMP is busy, and then send the byte
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
  {
    // Send the data
    PLIB_PMP_MasterSend( PMP_ID_0, data );
  }
```

Receiving a Data byte

The following sequence can be used to send a data byte and repeated to send an arbitrary number of data bytes.

Preconditions:

- The address of the destination must be configured in the PMP module
- The proper control signal are configured for PMP operation

Process:

1. Ensure that the PMP module is not busy by using PLIB_PMP_PortIsBusy.
2. Write the output data buffer using PLIB_PMP_MasterReceive.

Example:

```
uint8_t data;
// Set the source address
PLIB_PMP_AddressSet(PMP_ID_0, 0x1234 );
// Check to see the PMP is not busy, and then read the data
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
  {
    // receive the data
    data = PLIB_PMP_MasterReceive( PMP_ID_0 );
  }
```

Operating as a Slave

This section describes how to set up the PMP module to operate as a slave.

Description

The PMP module while acting as a slave, waits for read or write strobes to occur. Once the buffers have been manipulated, the appropriate action takes place.

Summary of Steps

- Check to see if data is available to be read
- Prepare output buffer for slave read

Each of these steps making up a transfer will take some time to complete. By monitoring the busy bit of the PMP module, the user application can determine whether the transfer is complete. The completion of each step can cause a PMP interrupt.

Reading Available Data

The following sequence can be used to receive a data byte and repeated to receive an arbitrary number of data bytes.

Preconditions:

- The address of the destination must be set in the PMP module
- The proper control signals are initialized for PMP operation

Process:

1. Check to see if data is available using PLIB_PMP_InputBufferXIsFull.
2. Read the available data using PLIB_PMP_InputBufferXByteReceive.

Example:

```
uint8_t data;
// Check to see if data is available, and then receive the byte
if(PLIB_PMP_InputBufferXIsFull( PMP_ID_0, 1))
  {
    // Receive the data from buffer one.
    data = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 1 );
  }
```

Preparing Output Buffer for Slave Write

The following sequence can be used to send a data byte and repeated to send an arbitrary number of data bytes.

Preconditions:

The proper control signals are initialized for PMP operation.

Process:

1. Ensure that the output buffer to be written is empty by using PLIB_PMP_OutputBufferXIsEmpty.
2. Write the output data buffer using PLIB_PMP_OutputBufferXByteSend.

Example:

```
 uint8_t data;
// Check to see if output buffer is available, and then send the byte
if(PLIB_PMP_OutputBufferXIsEmpty( PMP_ID_0, 1))
  {
    // Fill the output buffer-1 with the data to be sent
    PLIB_PMP_OutputBufferXByteSend( PMP_ID_0, 1, value);
  }
```

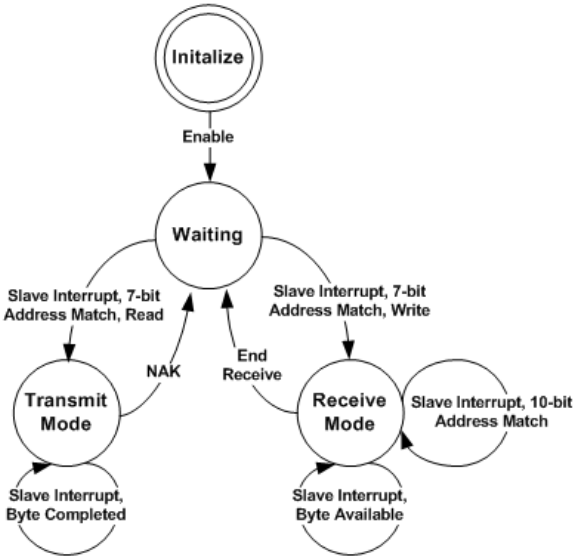
State Machine

The PMP state machine can be used in either a polled or an interrupt-driven manner. However, in both cases, software must check the state of the master, slave, and error interrupt flags to identify when a state transition occurs.

Description

The PMP module has one basic interrupt that is triggered when data has been written or read. In Buffered Slave mode, this interrupt can be set to wait until an 'X' amount of buffers are written or read. This interrupt must be cleared in software, but can also be used in DMA transactions to trigger the next DMA transfer.

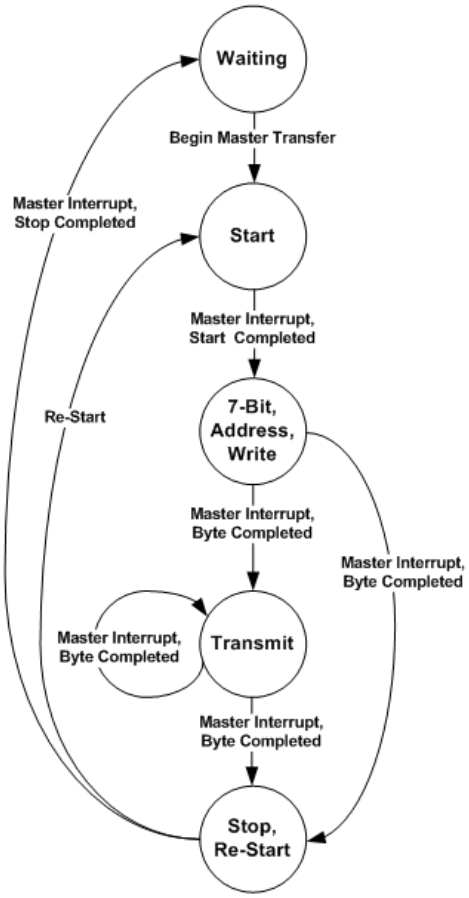
The PMP module will not start generating interrupts (setting the PMP interrupt flags, even if interrupts are disabled) until it is properly configured and enabled. After that, interrupts are generated as described in subsequent sections. Software has to respond appropriately once the interrupt has occurred (the flag has been set) to allow the state machine to advance to the next state. These actions are described in the following sections for the state machine diagram to which they refer.



Slave Mode State Machine

Slave-Mode State Transitions

In Slave mode, state transitions occur under hardware control automatically or in response to an interrupt. Software must respond appropriately to ensure that the PMP module continues to operate correctly. Basically, if a read/write strobe occurs, the appropriate read/write occurs, which in turn sets the interrupt.



Master Mode State Machine

Master-Mode State Transitions

In Master mode, state transitions occur under hardware control automatically or in response to an interrupt. Software must respond appropriately to ensure that the PMP module continues to operate correctly. Basically, if an input or output register is manipulated, the appropriate read/write happens, which in turn sets the interrupt. The PMP module can also be configured to have read/write strobes indicating whether a read/write has completed.

Handling Errors

There are two basic types of errors that can occur during various slave PMP operations:

- Input Buffer Overrun
- Output Buffer Underflow

This section provides information on handling these types of errors.

Description

Handling Input Buffer Overrun Errors

An input buffer overflow error occurs when the software is not reading or clearing the input buffers fast enough for the master PMP device attached to it. When this occurs, the master write is not allowed and the "input buffer overflow" status bit is set. This can be identified by calling PLIB_PMP_InputOverflowHasOccurred. Additional attempts to write to the input buffer will not be allowed until the overflow error is cleared by calling PLIB_PMP_InputOverflowClear. This type of error should be either avoided by having proper communication techniques with the PMP Master device or checked using PLIB_PMP_InputBuffersAreFull, which informs the controller that no buffers have data in them (overflow) .

Interrupts: The input buffer overflow error does not trigger an interrupt.

Handling Output Buffer Underflow Errors

An output buffer underflow error occurs when the software is not writing to the output buffers fast enough for the master PMP device attached to it. When this occurs, the master read is not allowed and the "output buffer underflow" status bit is set. This can be identified by calling PLIB_PMP_OutputUnderflowHasOccurred. Additional attempts to read from the output buffer will not be allowed until the overflow error is cleared by calling PLIB_PMP_OutputUnderflowClear. This sort of error should be either avoided by having proper communication techniques with the PMP Master device, or checked using PLIB_PMP_OutputBuffersAreEmpty, which informs the controller that buffers have data in them (underflow) .

Interrupts: The input buffer overflow error does not trigger an interrupt.

Other Features

This section provides information on additional features that may be available. These features are not available on all devices. Please refer to the **"Parallel Master Port (PMP)"** chapter in the specific device data sheet to determine which features are supported by your device.

Description

Operation During Sleep Mode

When the device enters Sleep mode, the system clock is disabled. The consequences of Sleep mode depend on which mode the PMP module is configured in at the time that Sleep mode is invoked.

Master

If the device enters Sleep mode while the PMP module is operating in Master mode, PMP operation will be suspended in its current state until clock execution resumes. As this may cause unexpected control pin timings, users should avoid invoking Sleep mode when continuous use of the module is needed.

Slave

While the PMP module is inactive, but enabled for any Slave mode operation, any read or write operations occurring at that time will be able to complete without the use of the microcontroller clock. Once the operation is completed, the module will issue an interrupt.

This interrupt may be used to wake the device from Sleep or Idle modes, depending on the configuration and capabilities of the device. Refer to the specific device data sheet or the related peripheral section in the reference manual to determine the capabilities of your device.

Operation During Idle Mode

When the device enters Idle mode, the system clock sources remain functional and the CPU stops code execution. PMP operation during Idle mode can be controlled using the PLIB_PMP_StopInIdleEnable or the PLIB_PMP_StopInIdleDisable. By default, the PMP module continues to operate in Idle mode and provide full functionality.

Configuring the Library

The library is configured for the supported PMP module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Initialization Functions

	Name	Description
	PLIB_PMP_AddressIncrementModeGet	Gets the increment mode being used with the address of the PMP module.
	PLIB_PMP_AddressIncrementModeSelect	Configures the increment mode being used with the address of the PMP module.
	PLIB_PMP_AddressLatchStrobeDisable	Disables the specific address latch strobe.
	PLIB_PMP_AddressLatchStrobeEnable	Enables the specific address latch strobe.
	PLIB_PMP_ChipSelectFunctionSelect	Defines the functionality of the pins intended to be used as Chip Select.
	PLIB_PMP_ChipSelectXDisable	Configures the Chip Select.
	PLIB_PMP_ChipSelectXEnable	Configures the Chip Select.
	PLIB_PMP_ChipSelectXIsActive	Gets the current status of the specified Chip Select.
	PLIB_PMP_DataSizeSelect	Enables data transfer size for the PMP module.
	PLIB_PMP_Disable	Disables the specific PMP module.
	PLIB_PMP_Enable	Enables the specific PMP module.
	PLIB_PMP_InputBufferTypeSelect	Selects the input buffer based on the input passed.
	PLIB_PMP_InterruptModeGet	Gets the current configured interrupt mode being used with the PMP module.
	PLIB_PMP_InterruptModeSelect	Configures the interrupt request mode being used with the PMP module.
	PLIB_PMP_MultiplexModeGet	Gets the current multiplexing mode configured between the address and data of the PMP module.
	PLIB_PMP_MultiplexModeSelect	Configures the multiplexing between the address and data of the PMP module.
	PLIB_PMP_OperationModeGet	Gets the current operation mode of the PMP module.
	PLIB_PMP_OperationModeSelect	Configures the operation mode of the PMP module.
	PLIB_PMP_StopInIdleDisable	Enables the PMP module to continue operation in Idle mode.
	PLIB_PMP_StopInIdleEnable	Discontinues PMP module operation when the device enters Idle mode.
	PLIB_PMP_DualModeReadAddressGet	Gets the current read address of the PMP module.

	PLIB_PMP_DualModeReadAddressSet	Sets the address to be written in Dual mode.
	PLIB_PMP_DualModeWriteAddressGet	Gets the current write address of the PMP module.
	PLIB_PMP_DualModeWriteAddressSet	Sets the address to be written in Dual mode.
	PLIB_PMP_DualModeMasterReceive	Receives the data in the Master Dual mode.
	PLIB_PMP_DualBufferEnable	Enables PMP dual Read/Write buffer.
	PLIB_PMP_DualBufferDisable	Disables the specific PMP module.
	PLIB_PMP_DualModeMasterSend	Sends the specified data in Dual Master mode.

b) Enhanced General Initialization Functions

	Name	Description
	PLIB_PMP_ReadChipSelectXDisable	Configures the Read Chip Select.
	PLIB_PMP_ReadChipSelectXEnable	Configures the Read Chip Select.
	PLIB_PMP_WriteChipSelectXDisable	Configures the Write Chip Select.
	PLIB_PMP_WriteChipSelectXEnable	Configures the Write Chip Select.

c) General Status Functions

	Name	Description
	PLIB_PMP_IsEnabled	Checks whether or not the PMP module is enabled.
	PLIB_PMP_DualBufferIsEnabled	Checks whether or not the PMP module is enabled.
	PLIB_PMP_PortIsBusy	Identifies if the (Master mode) PMP port is busy.
	PLIB_PMP_InputOverflowHasOccurred	Identifies if there was a receiver overflow error.
	PLIB_PMP_OutputUnderflowHasOccurred	Identifies if there was an output buffer sent out with no data.

d) Error Status/Control Functions

	Name	Description
	PLIB_PMP_InputOverflowClear	Clears a PMP Overflow error.
	PLIB_PMP_OutputUnderflowClear	Clears a PMP output underflow error.

e) Data Read and Write Functions

	Name	Description
	PLIB_PMP_InputBuffersAreFull	Gets the state of the input buffers.
	PLIB_PMP_InputBufferXByteReceive	Data to be received in Byte mode.
	PLIB_PMP_InputBufferXIsFull	Gets the state of the identified input buffer.
	PLIB_PMP_IsDataReceived	Checks and returns if the data has been received in the specified buffer in Slave mode.
	PLIB_PMP_IsDataTransmitted	Checks and returns if the data has been transmitted from the specified buffer in Slave mode.
	PLIB_PMP_MasterReceive	Receives the data in Master mode.
	PLIB_PMP_MasterSend	Sends the specified data in Master mode.
	PLIB_PMP_OutputBuffersAreEmpty	Gets the state of the output buffers.
	PLIB_PMP_OutputBufferXByteSend	Data to be transmitted in Byte mode.
	PLIB_PMP_OutputBufferXIsEmpty	Gets the state of the input buffer.
	PLIB_PMP_SlaveReceive	Receives the data in Slave mode.
	PLIB_PMP_SlaveSend	Sends the specified data in Slave mode.
	PLIB_PMP_ReadCycleIsStarted	Checks whether or not the read cycle on PMP bus is enabled.
	PLIB_PMP_ReadCycleStart	Starts a read cycle on the PMP bus.

f) Wait States Initialization Functions

	Name	Description
	PLIB_PMP_WaitStatesDataHoldSelect	Configures the data hold states (after data transfer) needed to be used with the PMP module.
	PLIB_PMP_WaitStatesDataSetUpSelect	Configures the data wait states (before the data transfer) needed to be used with the PMP module.
	PLIB_PMP_WaitStatesStrobeSelect	Configures the strobe wait states (during the data transfer) needed to be used with the PMP module.

g) Address Handling Functions

	Name	Description
	PLIB_PMP_AddressGet	Gets the current address of the PMP module.
	PLIB_PMP_AddressLinesA0A1Get	Gets the value of the address lines PMA0 and PMA1.
	PLIB_PMP_AddressLinesA0A1Set	Sets the address lines PMA0 and PMA1 with the value specified.
	PLIB_PMP_AddressSet	Sets the current address of the PMP module to the specified address.

h) Port Configuration Functions

	Name	Description
	PLIB_PMP_AddressPortDisable	Disables the port lines specified as PMP address lines.
	PLIB_PMP_AddressPortEnable	Enables the port lines specified as PMP address lines.
	PLIB_PMP_ReadWriteStrobePortDisable	Disables the PMP module read strobe port.
	PLIB_PMP_ReadWriteStrobePortEnable	Enables the PMP module read strobe port.
	PLIB_PMP_WriteEnableStrobePortDisable	Disables the PMP module write strobe port.
	PLIB_PMP_WriteEnableStrobePortEnable	Enables the PMP module write strobe port.

i) Polarity Configuration Functions

	Name	Description
	PLIB_PMP_AddressLatchPolaritySelect	Sets the address latch strobe polarity.
	PLIB_PMP_ChipSelectXPolaritySelect	Sets the specified Chip Select polarity.
	PLIB_PMP_ReadWriteStrobePolaritySelect	Sets the polarity of the read strobe.
	PLIB_PMP_WriteEnableStrobePolaritySelect	Sets the polarity of the write enable strobe.

j) Feature Existence Functions

	Name	Description
	PLIB_PMP_ExistsAddressControl	Identifies whether the AddressControl feature exists on the PMP module.
	PLIB_PMP_ExistsAddressLatchPolarity	Identifies whether the AddressLatchPolarity feature exists on the PMP module.
	PLIB_PMP_ExistsAddressLatchStrobePortControl	Identifies whether the AddressLatchStrobePortControl feature exists on the PMP module.
	PLIB_PMP_ExistsAddressPortPinControl	Identifies whether the AddressPortPinControl feature exists on the PMP module.
	PLIB_PMP_ExistsBufferOverFlow	Identifies whether the BufferOverFlow feature exists on the PMP module.
	PLIB_PMP_ExistsBufferRead	Identifies whether the BufferRead feature exists on the PMP module.
	PLIB_PMP_ExistsBufferType	Identifies whether the BufferType feature exists on the PMP module.
	PLIB_PMP_ExistsBufferUnderFlow	Identifies whether the BufferUnderFlow feature exists on the PMP module.
	PLIB_PMP_ExistsBufferWrite	Identifies whether the BufferWrite feature exists on the PMP module.
	PLIB_PMP_ExistsBusyStatus	Identifies whether the BusyStatus feature exists on the PMP module.
	PLIB_PMP_ExistsChipSelectEnable	Identifies whether the ChipSelectEnable feature exists on the PMP module.
	PLIB_PMP_ExistsChipSelectoperation	Identifies whether the ChipSelectoperation feature exists on the PMP module.
	PLIB_PMP_ExistsChipXPolarity	Identifies whether the ChipXPolarity feature exists on the PMP module.
	PLIB_PMP_ExistsCSXActiveStatus	Identifies whether the CSXActiveStatus feature exists on the PMP module.
	PLIB_PMP_ExistsDataHoldWaitStates	Identifies whether the DataHoldWaitStates feature exists on the PMP module.
	PLIB_PMP_ExistsDataSetUpWaitStates	Identifies whether the DataSetUpWaitStates feature exists on the PMP module.
	PLIB_PMP_ExistsDataStrobeWaitStates	Identifies whether the DataStrobeWaitStates feature exists on the PMP module.
	PLIB_PMP_ExistsDataTransferSize	Identifies whether the DataTransferSize feature exists on the PMP module.
	PLIB_PMP_ExistsEnableControl	Identifies whether the EnableControl feature exists on the PMP module.
	PLIB_PMP_ExistsIncrementMode	Identifies whether the IncrementMode feature exists on the PMP module.
	PLIB_PMP_ExistsInputBufferFull	Identifies whether the InputBufferFull feature exists on the PMP module.
	PLIB_PMP_ExistsInputBufferXStatus	Identifies whether the InputBufferXStatus feature exists on the PMP module.
	PLIB_PMP_ExistsInterruptMode	Identifies whether the InterruptMode feature exists on the PMP module.
	PLIB_PMP_ExistsMasterRXTX	Identifies whether the MasterRXTX feature exists on the PMP module.
	PLIB_PMP_ExistsMUXModeSelect	Identifies whether the MUXModeSelect feature exists on the PMP module.
	PLIB_PMP_ExistsOperationMode	Identifies whether the OperationMode feature exists on the PMP module.
	PLIB_PMP_ExistsOutPutBufferEmpty	Identifies whether the OutPutBufferEmpty feature exists on the PMP module.
	PLIB_PMP_ExistsOutputBufferXStatus	Identifies whether the OutputBufferXStatus feature exists on the PMP module.

	PLIB_PMP_ExistsReadWritePolarity	Identifies whether the ReadWritePolarity feature exists on the PMP module.
	PLIB_PMP_ExistsReadWriteStrobePortControl	Identifies whether the ReadWriteStrobePortControl feature exists on the PMP module.
	PLIB_PMP_ExistsSlaveRX	Identifies whether the SlaveRX feature exists on the PMP module.
	PLIB_PMP_ExistsSlaveTX	Identifies whether the SlaveTX feature exists on the PMP module.
	PLIB_PMP_ExistsStopInIdleControl	Identifies whether the StopInIdleControl feature exists on the PMP module.
	PLIB_PMP_ExistsWriteEnablePolarity	Identifies whether the WriteEnablePolarity feature exists on the PMP module.
	PLIB_PMP_ExistsWriteEnablePortControl	Identifies whether the WriteEnablePortControl feature exists on the PMP module.
	PLIB_PMP_ExistsDualBufferControl	Identifies whether the DualBufferControl feature exists on the PMP module.
	PLIB_PMP_ExistsDualModeMasterRXTX	Identifies whether the DualModeMasterRXTX feature exists on the PMP module.
	PLIB_PMP_ExistsDualModeReadAddressControl	Identifies whether the DualModeReadAddressControl feature exists on the PMP module.
	PLIB_PMP_ExistsDualModeWriteAddressControl	Identifies whether the DualModeWriteAddressControl feature exists on the PMP module.
	PLIB_PMP_ExistsReadChipSelectEnable	Identifies whether the ReadChipSelectEnable feature exists on the PMP module.
	PLIB_PMP_ExistsWriteChipSelectEnable	Identifies whether the WriteChipSelectEnable feature exists on the PMP module.
	PLIB_PMP_ExistsStartReadControl	Identifies whether the StartReadControl feature exists on the PMP module.

k) Data Types and Constants

	Name	Description
	PMP_ACK_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_ADDRESS_HOLD_LATCH_WAIT_STATES	PMP hold after address latch strobe wait states configuration.
	PMP_ADDRESS_LATCH	Address Latch Strobe configuration.
	PMP_ADDRESS_LATCH_WAIT_STATES	PMP address latch strobe wait states configuration.
	PMP_ADDRESS_PORT	Defines the different address lines that are available for configuration.
	PMP_ALTERNATE_MASTER_WAIT_STATES	PMP alternate master wait states.
	PMP_CHIP_SELECT	PMP Chip Select data type.
	PMP_CHIPSELECT_FUNCTION	Defines different functions available for the Chip Select lines multiplexed with address lines.
	PMP_DATA_HOLD_STATES	PMP Data Hold after strobe wait state.
	PMP_DATA_LENGTH	Possible data lengths handled by the PMP module.
	PMP_DATA_SIZE	PMP data size.
	PMP_DATA_WAIT_STATES	PMP data setup time configuration.
	PMP_INCREMENT_MODE	PMP address incrementing configuration.
	PMP_INPUT_BUFFER_TYPE	PMP Input Buffers type.
	PMP_INTERRUPT_MODE	PMP interrupt request mode data type.
	PMP_MASTER_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_MODULE_ID	Possible instances of the PMP module.
	PMP_MUX_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_OPERATION_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_PMBE_PORT	Defines the available Byte Enable ports.
	PMP_POLARITY_LEVEL	Possible polarity levels for the PMP pins.
	PMP_STROBE_WAIT_STATES	PMP strobe signal wait time configuration.

Description

This section describes the Application Programming Interface (API) functions of the PMP Peripheral Library.

Refer to each section for a detailed description.

a) General Initialization Functions

PLIB_PMP_AddressIncrementModeGet Function

Gets the increment mode being used with the address of the PMP module.

File

plib_pmp.h

C

```
PMP_INCREMENT_MODE PLIB_PMP_AddressIncrementModeGet(PMP_MODULE_ID index);
```

Returns

PMP_INCREMENT_MODE - One of the possible values from PMP_INCREMENT_MODE

Description

This function gets the pins used by the PMP module. Refer to the enumeration PMP_INCREMENT_MODE for the possible settings.

Remarks

This function implements an operation of the IncrementMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsIncrementMode in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_INCREMENT_MODE curAddressIncMode;
curAddressIncMode = PLIB_PMP_AddressIncrementModeGet( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

PMP_INCREMENT_MODE PLIB_PMP_AddressIncrementModeGet (PMP_MODULE_ID index)

PLIB_PMP_AddressIncrementModeSelect Function

Configures the increment mode being used with the address of the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressIncrementModeSelect(PMP_MODULE_ID index, PMP_INCREMENT_MODE incrementMode);
```

Returns

None.

Description

This function configures the pins used by the PMP module. Refer to the enumeration PMP_INCREMENT_MODE for the possible settings.

Remarks

This function implements an operation of the IncrementMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsIncrementMode in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_AddressIncrementModeSelect( PMP_ID_0, PMP_ADDRESS_INCREMENT_DECREMENT_DISABLE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
incrementMode	One of the possible values from PMP_INCREMENT_MODE

Function

void PLIB_PMP_AddressIncrementModeSelect(PMP_MODULE_ID index,
PMP_INCREMENT_MODE incrementMode)

PLIB_PMP_AddressLatchStrobeDisable Function

Disables the specific address latch strobe.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressLatchStrobeDisable(PMP_MODULE_ID index, PMP_ADDRESS_LATCH latch);
```

Returns

None.

Description

This function disables the PMP module with a specific address latch strobe depending on which strobes are not needed.

Remarks

This function implements an operation of the AddressLatchStrobePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressLatchStrobePortControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_AddressLatchStrobeDisable( PMP_ID_0, PMP_ADDRESS_LATCH_UPPER );
PLIB_PMP_AddressLatchStrobeDisable( PMP_ID_0, PMP_ADDRESS_LATCH_HIGH );
PLIB_PMP_AddressLatchStrobeDisable( PMP_ID_0, PMP_ADDRESS_LATCH_LOW );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
latch	Identifier for the latch to be disabled

Function

void PLIB_PMP_AddressLatchStrobeDisable (PMP_MODULE_ID index,
PMP_ADDRESS_LATCH latch)

PLIB_PMP_AddressLatchStrobeEnable Function

Enables the specific address latch strobe.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressLatchStrobeEnable(PMP_MODULE_ID index, PMP_ADDRESS_LATCH latch);
```

Returns

None.

Description

This function enables the PMP module with a specific address latch strobe depending on which strobes are needed.

Remarks

This function implements an operation of the AddressLatchStrobePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressLatchStrobePortControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_AddressLatchStrobeEnable( PMP_ID_0, PMP_ADDRESS_LATCH_UPPER );
PLIB_PMP_AddressLatchStrobeEnable( PMP_ID_0, PMP_ADDRESS_LATCH_HIGH );
PLIB_PMP_AddressLatchStrobeEnable( PMP_ID_0, PMP_ADDRESS_LATCH_LOW );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
latch	Identifier for the latch to be enabled

Function

void PLIB_PMP_AddressLatchStrobeEnable (PMP_MODULE_ID index,
PMP_ADDRESS_LATCH latch)

PLIB_PMP_ChipSelectFunctionSelect Function

Defines the functionality of the pins intended to be used as Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_ChipSelectFunctionSelect(PMP_MODULE_ID index, PMP_CHIPSELECT_FUNCTION chipselfunct);
```

Returns

None.

Description

This function selects the PMCS1/PMCS2 as either Chip Select or as address lines depending on the way these bits are programmed.

Remarks

This function implements an operation of the ChipSelectoperation feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsChipSelectoperation in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_ChipSelectFunctionSelect( PMP_ID_0, PMCS1_PMCS2_AS_ADDRESS_LINES );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipselfunct	One of the possible values from PMP_CHIPSELECT_FUNCTION

Function

void PLIB_PMP_ChipSelectFunctionSelect(PMP_MODULE_ID index,
PMP_CHIPSELECT_FUNCTION chipselfunct)

PLIB_PMP_ChipSelectXDisable Function

Configures the Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_ChipSelectXDisable(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

None.

Description

This function configures the Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as the address pin.

Remarks

This function implements an operation of the ChipSelectEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsChipSelectEnable in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_ChipSelectXDisable( PMP_ID_0, chipSelect );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Function

void PLIB_PMP_ChipSelectXDisable(PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect)

PLIB_PMP_ChipSelectXEnable Function

Configures the Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_ChipSelectXEnable(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

None.

Description

This function configures the Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as chipSelect.

Remarks

This function implements an operation of the ChipSelectEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsChipSelectEnable in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_ChipSelectXEnable( PMP_ID_0, chipSelect );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Function

void PLIB_PMP_ChipSelectXEnable(PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect)

PLIB_PMP_ChipSelectXIsActive Function

Gets the current status of the specified Chip Select.

File

plib_pmp.h

C

```
bool PLIB_PMP_ChipSelectXIsActive(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

- true - Chip Select is active
- false - Chip Select is not active

Description

This function returns the current status of the specified Chip Select.

Remarks

This function implements an operation of the CSXActiveStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsCSXActiveStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
if(PLIB_PMP_ChipSelectXIsActive( PMP_ID_0, chipSelect ))
{
  //Do something useful
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for the Chip Select to be checked

Function

bool PLIB_PMP_ChipSelectXIsActive (PMP_MODULE_ID index,

PMP_CHIP_SELECT chipSelect)

PLIB_PMP_DataSizeSelect Function

Enables data transfer size for the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_DataSizeSelect(PMP_MODULE_ID index, PMP_DATA_SIZE size);
```

Returns

None.

Description

This function enables 4-bit, 8-bit, or 16-bit data transfer for the PMP module.

Remarks

This function implements an operation of the DataTransferSize feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDataTransferSize in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_DataSizeSelect( PMP_ID_0, PMP_DATA_SIZE_8_BITS );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
size	Identifier for the data size to be used

Function

void PLIB_PMP_DataSizeSelect (PMP_MODULE_ID index,
PMP_DATA_SIZE size)

PLIB_PMP_Disable Function

Disables the specific PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_Disable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific PMP module.

Remarks

The default state after any reset for a PMP module is the disable state. If the PMP is disabled, all the related pins are in control of the general purpose I/O logic.

Disabling the PMP module resets the buffers to empty states. Any data characters in the buffers are lost. All error and status bits are also reset.

Disabling the PMP while the PMP is active, will abort all pending transmissions and receptions. Re-enabling the PMP will restart the module in the same configuration.

When disabled, the PMP power consumption is minimal.

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsEnableControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_Disable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_Disable (PMP_MODULE_ID index)

PLIB_PMP_Enable Function

Enables the specific PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_Enable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific PMP module.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsEnableControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_Enable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_Enable (PMP_MODULE_ID index)

PLIB_PMP_InputBufferTypeSelect Function

Selects the input buffer based on the input passed.

File

plib_pmp.h

C

```
void PLIB_PMP_InputBufferTypeSelect(PMP_MODULE_ID index, PMP_INPUT_BUFFER_TYPE inputBuffer);
```

Returns

None.

Description

This function selects the input buffer based on the input passed. Either TTL or Schmitt Trigger input buffers are selected.

Remarks

This function implements an operation of the BufferType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_InputBufferTypeSelect( PMP_ID_0, PMP_INPUT_BUFFER_TTL );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
inputBuffer	One of the possible input buffer types

Function

void PLIB_PMP_InputBufferTypeSelect (PMP_MODULE_ID index,
PMP_INPUT_BUFFER_TYPE inputBuffer)

PLIB_PMP_InterruptModeGet Function

Gets the current configured interrupt mode being used with the PMP module.

File

plib_pmp.h

C

```
PMP_INTERRUPT_MODE PLIB_PMP_InterruptModeGet(PMP_MODULE_ID index);
```

Returns

One of the possible values from PMP_INTERRUPT_MODE.

Description

This function gets the current configured interrupt mode being used with the PMP module.

Remarks

This function implements an operation of the InterruptMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsInterruptMode in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_INTERRUPT_MODE currentIntMode;
currentIntMode = PLIB_PMP_InterruptModeGet ( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

PMP_INTERRUPT_MODE PLIB_PMP_InterruptModeGet (PMP_MODULE_ID index)

PLIB_PMP_InterruptModeSelect Function

Configures the interrupt request mode being used with the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_InterruptModeSelect(PMP_MODULE_ID index, PMP_INTERRUPT_MODE interruptMode);
```

Returns

None.

Description

This function configures the pins used by the PMP module. Refer to the enumeration PMP_INTERRUPT_MODE for the possible settings.

Remarks

This function implements an operation of the InterruptMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsInterruptMode in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_InterruptModeSelect( PMP_ID_0, PMP_INTERRUPT_NONE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
interruptMode	One of the possible values from PMP_INTERRUPT_MODE

Function

void PLIB_PMP_InterruptModeSelect(PMP_MODULE_ID index,
PMP_INTERRUPT_MODE interruptMode)

PLIB_PMP_MultiplexModeGet Function

Gets the current multiplexing mode configured between the address and data of the PMP module.

File

plib_pmp.h

C

```
PMP_MUX_MODE PLIB_PMP_MultiplexModeGet(PMP_MODULE_ID index);
```

Returns

index - Identifier for the device instance to be addressed PMP_MUX_MODE - One of the possible values from PMP_MUX_MODE

Description

This function gets the current multiplexing mode configured between the address and data of the PMP module.

Remarks

This function implements an operation of the MUXModeSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsMUXModeSelect in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_MUX_MODE currentMuxMode;
currentMuxMode = PLIB_PMP_MultiplexModeGet( PMP_ID_0 );
```

Function

PMP_MUX_MODE PLIB_PMP_MultiplexModeGet(PMP_MODULE_ID index)

PLIB_PMP_MultiplexModeSelect Function

Configures the multiplexing between the address and data of the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_MultiplexModeSelect(PMP_MODULE_ID index, PMP_MUX_MODE multiplexMode);
```

Returns

None.

Description

This function configures the pins used by the PMP module. Refer to the enumeration PMP_MUX_MODE for the possible settings.

Remarks

This function implements an operation of the MUXModeSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsMUXModeSelect in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_MultiplexModeSelect( PMP_ID_0, PMP_MUX_NONE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
multiplexMode	One of the possible values from the PMP_MUX_MODE

Function

void PLIB_PMP_MultiplexModeSelect(PMP_MODULE_ID index,
PMP_MUX_MODE multiplexMode)

PLIB_PMP_OperationModeGet Function

Gets the current operation mode of the PMP module.

File

plib_pmp.h

C

```
PMP_OPERATION_MODE PLIB_PMP_OperationModeGet(PMP_MODULE_ID index);
```

Returns

PMP_OPERATION_MODE - One of the possible values from PMP_OPERATION_MODE

Description

This function gets the current operation mode of the PMP module.

Remarks

This function implements an operation of the OperationMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsOperationMode in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_OPERATION_MODE curOpMode;
curOpMode = PLIB_PMP_OperationModeGet( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

PMP_OPERATION_MODE PLIB_PMP_OperationModeGet (PMP_MODULE_ID index)

PLIB_PMP_OperationModeSelect Function

Configures the operation mode of the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_OperationModeSelect(PMP_MODULE_ID index, PMP_OPERATION_MODE operationMode);
```

Returns

None.

Description

This function configures operation mode of the PMP module. Refer to the enumeration PMP_OPERATION_MODE for the possible settings.

Remarks

This function implements an operation of the OperationMode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsOperationMode in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_OperationModeSelect( PMP_ID_0,
                              PMP_MASTER_READ_WRITE_STROBES_MULTIPLEXED );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
operationMode	One of the possible values from PMP_OPERATION_MODE

Function

void PLIB_PMP_OperationModeSelect(PMP_MODULE_ID index,
PMP_OPERATION_MODE operationMode)

PLIB_PMP_StopInIdleDisable Function

Enables the PMP module to continue operation in Idle mode.

File

plib_pmp.h

C

```
void PLIB_PMP_StopInIdleDisable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the stop in idle flag. The PMP module continues operation in Idle mode.

Remarks

By default, the PMP module will continue operation in Idle mode.

This function implements an operation of the StopInIdleControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsStopInIdleControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_StopInIdleDisable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_StopInIdleDisable (PMP_MODULE_ID index)

PLIB_PMP_StopInIdleEnable Function

Discontinues PMP module operation when the device enters Idle mode.

File

plib_pmp.h

C

```
void PLIB_PMP_StopInIdleEnable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the PMP module to discontinue operation when the device enters Idle mode.

Remarks

This function implements an operation of the StopInIdleControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsStopInIdleControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_StopInIdleEnable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_StopInIdleEnable (PMP_MODULE_ID index)

PLIB_PMP_DualModeReadAddressGet Function

Gets the current read address of the PMP module.

File

plib_pmp.h

C

```
uint32_t PLIB_PMP_DualModeReadAddressGet(PMP_MODULE_ID index);
```

Returns

- readAddress - Device address from where PMP read will occur

Description

This function gets the current read address of the PMP module.

Remarks

This function implements an operation of the DualModeReadAddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualModeReadAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
uint32_t readAddress;
readAddress = PLIB_PMP_DualModeReadAddressGet( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

uint32_t PLIB_PMP_DualModeReadAddressGet (PMP_MODULE_ID index)

PLIB_PMP_DualModeReadAddressSet Function

Sets the address to be written in Dual mode.

File

plib_pmp.h

C

```
void PLIB_PMP_DualModeReadAddressSet(PMP_MODULE_ID index, uint32_t readAddress);
```

Returns

None.

Description

This function sets the address to be written to the specified value in Dual mode.

Remarks

This function implements an operation of the DualModeReadAddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualModeReadAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
uint8_t no_of_addressLines = 8;
PLIB_PMP_DualModeReadAddressSet( PMP_ID_0, 0xff );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
readAddress	Device address from where PMP read will happen

Function

void PLIB_PMP_DualModeReadAddressSet (PMP_MODULE_ID index,

uint32_t readAddress)

PLIB_PMP_DualModeWriteAddressGet Function

Gets the current write address of the PMP module.

File

plib_pmp.h

C

```
uint32_t PLIB_PMP_DualModeWriteAddressGet(PMP_MODULE_ID index);
```

Returns

- writeAddress - Device Write address to be set

Description

This function gets the current write address of the PMP module.

Remarks

This function implements an operation of the DualModeWriteAddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualModeWriteAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
uint32_t address;
address = PLIB_PMP_DualModeWriteAddressGet( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

uint32_t PLIB_PMP_DualModeWriteAddressGet (PMP_MODULE_ID index)

PLIB_PMP_DualModeWriteAddressSet Function

Sets the address to be written in Dual mode.

File

plib_pmp.h

C

```
void PLIB_PMP_DualModeWriteAddressSet(PMP_MODULE_ID index, uint32_t writeAddress);
```

Returns

None.

Description

This function sets the address to be written to the specified value in Dual mode.

Remarks

This function implements an operation of the DualModeWriteAddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualModeWriteAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
uint8_t no_of_addressLines = 8;
PLIB_PMP_DualModeWriteAddressSet( PMP_ID_0, 0xff );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
writeAddress	Device write address to be set

Function

void PLIB_PMP_DualModeWriteAddressSet (PMP_MODULE_ID index,

uint32_t writeAddress)

PLIB_PMP_DualModeMasterReceive Function

Receives the data in the Master Dual mode.

File

plib_pmp.h

C

```
uint16_t PLIB_PMP_DualModeMasterReceive(PMP_MODULE_ID index);
```

Returns

- uint16_t - Data received

Description

This function receives the data in Dual mode. The flow of data is from the slave to the master.

Remarks

This function to be used only when the PMP is acting as master. This function implements an operation of the DualModeMasterRXTX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualModeMasterRXTX in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
uint16_t data;
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    data = PLIB_PMP_DualModeMasterReceive( PMP_ID_0 );
```

```
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed.

Function

uint16_t PLIB_PMP_DualModeMasterReceive (PMP_MODULE_ID index)

PLIB_PMP_DualBufferEnable Function

Enables PMP dual Read/Write buffer.

File

plib_pmp.h

C

```
void PLIB_PMP_DualBufferEnable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function enables dual Read/Write buffers for PMP module. Once enabled, PMP uses separate registers for read and write.

- Registers used for Reads: PMRADDR and PMRDIN
- Registers used for Writes: PMRWADDR and PMDOUT

Remarks

This feature is only valid in Master mode.

This function implements an operation of the DualBufferControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualBufferControl in your application to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master.

Example

```
PLIB_PMP_DualBufferEnable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_DualBufferEnable (PMP_MODULE_ID index)

PLIB_PMP_DualBufferDisable Function

Disables the specific PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_DualBufferDisable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific PMP module.

Remarks

This feature is only valid in Master mode.

This function implements an operation of the DualBufferControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualBufferControl in your application to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master.

Example

```
PLIB_PMP_DualBufferDisable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_DualBufferDisable (PMP_MODULE_ID index)

PLIB_PMP_DualModeMasterSend Function

Sends the specified data in Dual Master mode.

File

plib_pmp.h

C

```
void PLIB_PMP_DualModeMasterSend(PMP_MODULE_ID index, uint16_t data);
```

Returns

None.

Description

This function sends the specified data in dual mode. The data flow is from master to slave.

Remarks

This function to be used only when the PMP is acting as master. This function implements an operation of the DualModeMasterRXTX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualModeMasterRXTX in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
uint16_t data = 'a';
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    PLIB_PMP_DualModeMasterSend( PMP_ID_0, data );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
data	Data to be transmitted

Function

void PLIB_PMP_DualModeMasterSend (PMP_MODULE_ID index,

uint16_t data)

b) Enhanced General Initialization Functions

PLIB_PMP_ReadChipSelectXDisable Function

Configures the Read Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_ReadChipSelectXDisable(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

None.

Description

This function configures the Read Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as the address pin.

Remarks

This function implements an operation of the ReadChipSelectEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsReadChipSelectEnable in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_ReadChipSelectXDisable( PMP_ID_0, chipSelect );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Function

void PLIB_PMP_ReadChipSelectXDisable(PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect)

PLIB_PMP_ReadChipSelectXEnable Function

Configures the Read Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_ReadChipSelectXEnable(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

None.

Description

This function configures the Read Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as chipSelect.

Remarks

This function implements an operation of the ReadChipSelectEnable feature. This feature may not be available on all devices. Please refer to the

specific device data sheet to determine availability or use PLIB_PMP_ExistsReadChipSelectEnable in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_ReadChipSelectXEnable( PMP_ID_0, chipSelect );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Function

void PLIB_PMP_ReadChipSelectXEnable(PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect)

PLIB_PMP_WriteChipSelectXDisable Function

Configures the Write Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_WriteChipSelectXDisable(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

None.

Description

This function configures the Write Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as the address pin.

Remarks

This function implements an operation of the WriteChipSelectEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsWriteChipSelectEnable in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_WriteChipSelectXDisable( PMP_ID_0, chipSelect );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Function

void PLIB_PMP_WriteChipSelectXDisable(PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect)

PLIB_PMP_WriteChipSelectXEnable Function

Configures the Write Chip Select.

File

plib_pmp.h

C

```
void PLIB_PMP_WriteChipSelectXEnable(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect);
```

Returns

None.

Description

This function configures the Write Chip Select(s) being used by the PMP module. The specified Chip Select pin functions as chipSelect.

Remarks

This function implements an operation of the WriteChipSelectEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsWriteChipSelectEnable in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as master. PMP Dual mode should be enabled using the API PLIB_PMP_DualBufferEnable.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_WriteChipSelectXEnable( PMP_ID_0, chipSelect );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for which Chip Select to configure

Function

void PLIB_PMP_WriteChipSelectXEnable(PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect)

c) General Status Functions

PLIB_PMP_IsEnabled Function

Checks whether or not the PMP module is enabled.

File

plib_pmp.h

C

```
bool PLIB_PMP_IsEnabled(PMP_MODULE_ID index);
```

Returns

- true - If the PMP module is enabled
- false - if the PMP module is disabled

Description

This function checks whether or not the PMP module is enabled.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsEnableControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool pmpStatus;
pmpStatus = PLIB_PMP_IsEnabled( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_IsEnabled (PMP_MODULE_ID index)

PLIB_PMP_DualBufferIsEnabled Function

Checks whether or not the PMP module is enabled.

File

plib_pmp.h

C

```
bool PLIB_PMP_DualBufferIsEnabled(PMP_MODULE_ID index);
```

Returns

- true - If the PMP module is enabled
- false - if the PMP module is disabled

Description

This function checks whether or not the PMP module is enabled.

Remarks

This feature is only valid in Master mode.

This function implements an operation of the DualBufferControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDualBufferControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool pmpStatus;
pmpStatus = PLIB_PMP_DualBufferIsEnabled( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_DualBufferIsEnabled (PMP_MODULE_ID index)

PLIB_PMP_PortIsBusy Function

Identifies if the (Master mode) PMP port is busy.

File

plib_pmp.h

C

```
bool PLIB_PMP_PortIsBusy(PMP_MODULE_ID index);
```

Returns

- true - If the port is busy
- false - If the port is not busy

Description

This function identifies if the PMP port is busy (in Master mode).

Remarks

Works only in Master mode. This function implements an operation of the BusyStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBusyStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_PMP_PortIsBusy( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_PortIsBusy (PMP_MODULE_ID index)

PLIB_PMP_InputOverflowHasOccurred Function

Identifies if there was a receiver overflow error.

File

plib_pmp.h

C

```
bool PLIB_PMP_InputOverflowHasOccurred(PMP_MODULE_ID index);
```

Returns

- true - If the input buffer has overflowed
- false - If the input buffer has not overflowed

Description

This function identifies if there was a receiver overflow error.

Remarks

This function implements an operation of the BufferOverFlow feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferOverFlow in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_PMP_InputOverflowHasOccurred( PMP_ID_0 ))
{
    PLIB_PMP_InputOverflowClear( PMP_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_InputOverflowHasOccurred (PMP_MODULE_ID index)

PLIB_PMP_OutputUnderflowHasOccurred Function

Identifies if there was an output buffer sent out with no data.

File

plib_pmp.h

C

```
bool PLIB_PMP_OutputUnderflowHasOccurred(PMP_MODULE_ID index);
```

Returns

- true - If the input buffer was empty when data was sent
- false - If the output buffer was not empty when data was sent

Description

This function identifies if there was a output buffer was sent out with no data.

Remarks

This function implements an operation of the BufferUnderFlow feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferUnderFlow in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_PMP_OutputUnderflowHasOccurred( PMP_ID_0 ))
{
    PLIB_PMP_OutputUnderflowClear( PMP_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_OutputUnderflowHasOccurred (PMP_MODULE_ID index)

d) Error Status/Control Functions

PLIB_PMP_InputOverflowClear Function

Clears a PMP Overflow error.

File

plib_pmp.h

C

```
void PLIB_PMP_InputOverflowClear(PMP_MODULE_ID index);
```

Returns

None.

Description

This function clears an overflow error. Clearing the error resets the receive buffer.

Remarks

This function implements an operation of the BufferOverFlow feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferOverFlow in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_PMP_InputOverflowHasOccurred( PMP_ID_0 ))
{
    PLIB_PMP_InputOverflowClear( PMP_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_InputOverflowClear (PMP_MODULE_ID index)

PLIB_PMP_OutputUnderflowClear Function

Clears a PMP output underflow error.

File

plib_pmp.h

C

```
void PLIB_PMP_OutputUnderflowClear(PMP_MODULE_ID index);
```

Returns

None.

Description

This function clears a PMP output underflow error.

Remarks

This function implements an operation of the BufferUnderFlow feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferUnderFlow in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_PMP_OutputUnderflowHasOccurred( PMP_ID_0 ))
{
    PLIB_PMP_OutputUnderflowClear( PMP_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_OutputUnderflowClear (PMP_MODULE_ID index)

e) Data Read and Write Functions

PLIB_PMP_InputBuffersAreFull Function

Gets the state of the input buffers.

File

plib_pmp.h

C

```
bool PLIB_PMP_InputBuffersAreFull(PMP_MODULE_ID index);
```

Returns

- true - If all input buffers are full
- false - If all input buffers are not full

Description

This function gets the state of the input buffers.

Remarks

This function implements an operation of the InputBufferFull feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsInputBufferFull in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t value;
if(PLIB_PMP_InputBuffersAreFull( PMP_ID_0 ))
{
 value = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 1 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_InputBuffersAreFull (PMP_MODULE_ID index)

PLIB_PMP_InputBufferXByteReceive Function

Data to be received in Byte mode.

File

plib_pmp.h

C

```
uint8_t PLIB_PMP_InputBufferXByteReceive(PMP_MODULE_ID index, uint8_t buffer);
```

Returns

- data - Data to be received

Description

This function specifies the data to be received in Byte mode from the desired PMP module.

Remarks

This function implements an operation of the BufferRead feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferRead in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t mydata;
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    // get data from buffer-1
    mydata = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 1 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	One of the possible buffers (valid values are 0 to 3)

Function

uint8_t PLIB_PMP_InputBufferXByteReceive (PMP_MODULE_ID index,
uint8_t buffer)

PLIB_PMP_InputBufferXIsFull Function

Gets the state of the identified input buffer.

File

plib_pmp.h

C

```
bool PLIB_PMP_InputBufferXIsFull(PMP_MODULE_ID index, uint8_t buffer);
```

Returns

- true - If all input buffers are full
- false - If all input buffers are not full

Description

This function gets the state of the identified input buffer.

Remarks

This function implements an operation of the InputBufferXStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsInputBufferXStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t value;
// Check the status of buffer-2
if(PLIB_PMP_InputBufferXIsFull( PMP_ID_0, 2 ))
{
   // get data from buffer 2
   value = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 2 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	The input buffer number (valid values are 0 to 3)

Function

bool PLIB_PMP_InputBufferXIsFull (PMP_MODULE_ID index,
uint8_t buffer)

PLIB_PMP_IsDataReceived Function

Checks and returns if the data has been received in the specified buffer in Slave mode.

File

plib_pmp.h

C

```
bool PLIB_PMP_IsDataReceived(PMP_MODULE_ID index, uint8_t buffer);
```

Returns

- true - Data has been received in the specified buffer
- false - Data has not been received in the specified buffer

Description

This function checks and returns if the data has been received in the specified buffer in Slave mode.

Remarks

This function implements an operation of the InputBufferXStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsInputBufferXStatus in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module should be configured for Slave mode operation.

Example

```
int8_t data;
// Check if data is received on buffer-2
if(PLIB_PMP_IsDataReceived( PMP_ID_0, 2 ))
{
    // get data from buffer-2
    data = PLIB_PMP_InputBufferXByteReceive( PMP_ID_0, 2 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	One of the possible input buffers (valid values are 0 to 3)

Function

bool PLIB_PMP_IsDataReceived (PMP_MODULE_ID index,
uint8_t buffer)

PLIB_PMP_IsDataTransmitted Function

Checks and returns if the data has been transmitted from the specified buffer in Slave mode.

File

plib_pmp.h

C

```
bool PLIB_PMP_IsDataTransmitted(PMP_MODULE_ID index, uint8_t buffer);
```

Returns

- true - If data has been transmitted from the specified buffer

- false - If data has not been transmitted from the specified buffer

Description

This function checks and returns if data has been transmitted from the specified buffer.

Remarks

This function implements an operation of the OutputBufferXStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsOutputBufferXStatus in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module should be configured for Slave mode operation.

Example

```
uint8_t data;
if(PLIB_PMP_IsDataTransmitted( PMP_ID_0, 2 ))
{
    PLIB_PMP_OutputBufferXByteSend( PMP_ID_0, 2, data );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	One of the possible output buffers (valid range is 0 to 3)

Function

bool PLIB_PMP_IsDataTransmitted (PMP_MODULE_ID index,
uint8_t buffer)

PLIB_PMP_MasterReceive Function

Receives the data in Master mode.

File

plib_pmp.h

C

```
uint16_t PLIB_PMP_MasterReceive(PMP_MODULE_ID index);
```

Returns

uint16_t - Data received

Description

This function receives the data. The flow of data is from the slave to the master.

Remarks

This function to be used only when the PMP is acting as master. This function implements an operation of the MasterRXTX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsMasterRXTX in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a master.

Example

```
uint16_t data;
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    data = PLIB_PMP_MasterReceive( PMP_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed.

Function

uint16_t PLIB_PMP_MasterReceive (PMP_MODULE_ID index)

PLIB_PMP_MasterSend Function

Sends the specified data in Master mode.

File

plib_pmp.h

C

```
void PLIB_PMP_MasterSend(PMP_MODULE_ID index, uint16_t data);
```

Returns

None.

Description

This function sends the specified data. The data flow is from master to slave.

Remarks

This function to be used only when the PMP is acting as master. This function implements an operation of the MasterRXTX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsMasterRXTX in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured for Master mode.

Example

```
uint16_t data = 'a';
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    PLIB_PMP_MasterSend( PMP_ID_0, data );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
data	Data to be transmitted

Function

void PLIB_PMP_MasterSend (PMP_MODULE_ID index,

uint16_t data)

PLIB_PMP_OutputBuffersAreEmpty Function

Gets the state of the output buffers.

File

plib_pmp.h

C

```
bool PLIB_PMP_OutputBuffersAreEmpty(PMP_MODULE_ID index);
```

Returns

- true - If all output buffers are empty
- false - If all output buffers are not empty

Description

This function returns the state of the output buffers.

Remarks

This function implements an operation of the OutPutBufferEmpty feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsOutPutBufferEmpty in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t value=0xEF;

if(PLIB_PMP_OutputBuffersAreEmpty( PMP_ID_0 ))
{
 PLIB_PMP_OutputBufferXByteSend( PMP_ID_0, 1, value);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_OutputBuffersAreEmpty (PMP_MODULE_ID index)

PLIB_PMP_OutputBufferXByteSend Function

Data to be transmitted in Byte mode.

File

plib_pmp.h

C

```
void PLIB_PMP_OutputBufferXByteSend(PMP_MODULE_ID index, uint8_t buffer, uint8_t data);
```

Returns

None.

Description

This function specifies the data to be transmitted in Byte mode for the desired PMP module.

Remarks

This function implements an operation of the BufferWrite feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsBufferWrite in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t data = 'a';
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    PLIB_PMP_OutputBufferXByteSend( PMP_ID_0, 1, data );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

buffer	One of the possible output buffers (valid range is 0 to 3)
data	Data to be transmitted

Function

void PLIB_PMP_OutputBufferXByteSend (PMP_MODULE_ID index,
uint8_t buffer,
uint8_t data)

PLIB_PMP_OutputBufferXIsEmpty Function

Gets the state of the input buffer.

File

plib_pmp.h

C

```
bool PLIB_PMP_OutputBufferXIsEmpty(PMP_MODULE_ID index, uint8_t buffer);
```

Returns

- true - If the identified output buffer is empty
- false - If the identified output buffer is not empty

Description

This function returns the state of the input buffer.

Remarks

This function implements an operation of the OutputBufferXStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsOutputBufferXStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t value = 0xEF;

if(PLIB_PMP_OutputBufferXIsEmpty( PMP_ID_0, 1 ) )
{
 PLIB_PMP_OutputBufferXByteSend( PMP_ID_0,1, value);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
buffer	Output buffer number (valid range is 0 to 3)

Function

bool PLIB_PMP_OutputBufferXIsEmpty (PMP_MODULE_ID index,
uint8_t buffer)

PLIB_PMP_SlaveReceive Function

Receives the data in Slave mode.

File

plib_pmp.h

C

```
uint16_t PLIB_PMP_SlaveReceive(PMP_MODULE_ID index);
```

Returns

- uint16_t - Data received

Description

This function receives the data. The flow of data is from the master to the slave.

Remarks

This function to be used only when the PMP is acting as slave. This function implements an operation of the SlaveRX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsSlaveRX in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured as a slave.

Example

```
uint16_t data;
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    data = PLIB_PMP_SlaveReceive( PMP_ID_0 );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed.

Function

uint16_t PLIB_PMP_SlaveReceive (PMP_MODULE_ID index)

PLIB_PMP_SlaveSend Function

Sends the specified data in Slave mode.

File

plib_pmp.h

C

```
void PLIB_PMP_SlaveSend(PMP_MODULE_ID index, uint16_t data);
```

Returns

None.

Description

This function sends the specified data. The flow of data is from the slave to the master.

Remarks

This function to be used only when the PMP is acting as slave. This function implements an operation of the SlaveTX feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsSlaveTX in your application to to automatically determine whether this feature is available.

Preconditions

The PMP module is configured for Slave mode.

Example

```
uint16_t data = 'a';
if(!PLIB_PMP_PortIsBusy( PMP_ID_0 ))
{
    PLIB_PMP_SlaveSend( PMP_ID_0, data );
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

data	Data to be transmitted

Function

void PLIB_PMP_SlaveSend (PMP_MODULE_ID index,
uint16_t data)

PLIB_PMP_ReadCycleIsStarted Function

Checks whether or not the read cycle on PMP bus is enabled.

File

plib_pmp.h

C

```
bool PLIB_PMP_ReadCycleIsStarted(PMP_MODULE_ID index);
```

Returns

- true - If the PMP Read cycle is enabled
- false - if the PMP Read cycle is not enabled

Description

This function checks whether or not the read cycle on PMP bus is enabled.

Remarks

This function implements an operation of the StartReadControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsStartReadControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool pmpReadStartStatus;
pmpReadStartStatus = PLIB_PMP_ReadCycleIsStarted( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

bool PLIB_PMP_ReadCycleIsStarted (PMP_MODULE_ID index)

PLIB_PMP_ReadCycleStart Function

Starts a read cycle on the PMP bus.

File

plib_pmp.h

C

```
void PLIB_PMP_ReadCycleStart(PMP_MODULE_ID index);
```

Returns

None.

Description

This function stars a read cycle on the bus for the selected PMP module. This bit is cleared by hardware at the end of the read cycle

Remarks

This function implements an operation of the StartReadControl feature. This feature may not be available on all devices. Please refer to the

specific device data sheet to determine availability or use PLIB_PMP_ExistsStartReadControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_ReadCycleStart( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_ReadCycleStart(PMP_MODULE_ID index)

f) Wait States Initialization Functions

PLIB_PMP_WaitStatesDataHoldSelect Function

Configures the data hold states (after data transfer) needed to be used with the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_WaitStatesDataHoldSelect(PMP_MODULE_ID index, PMP_DATA_HOLD_STATES dataHoldState);
```

Returns

None.

Description

This function configures the number of peripheral bus clock cycles needed for wait states. Refer to the enumeration PMP_DATA_HOLD_STATES for the possible settings.

Remarks

This function implements an operation of the DataHoldWaitStates feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDataHoldWaitStates in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_WaitStatesDataHoldSelect( PMP_ID_0, PMP_DATA_HOLD_2 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
dataHoldState	One of the possible values from PMP_DATA_HOLD_STATES

Function

void PLIB_PMP_WaitStatesDataHoldSelect(PMP_MODULE_ID index,
PMP_DATA_HOLD_STATES dataHoldState)

PLIB_PMP_WaitStatesDataSetUpSelect Function

Configures the data wait states (before the data transfer) needed to be used with the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_WaitStatesDataSetUpSelect(PMP_MODULE_ID index, PMP_DATA_WAIT_STATES dataWaitState);
```

Returns

None.

Description

This function configures the number of peripheral bus clock cycles needed for wait states. Refer to the enumeration PMP_DATA_WAIT_STATES for the possible settings.

Remarks

This function implements an operation of the DataSetUpWaitStates feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDataSetUpWaitStates in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_WaitStatesDataSetUpSelect( PMP_ID_0, PMP_DATA_WAIT_TWO );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
dataWaitState	One of the possible values from PMP_DATA_WAIT_STATES

Function

void PLIB_PMP_WaitStatesDataSetUpSelect(PMP_MODULE_ID index,
PMP_DATA_WAIT_STATES dataWaitState)

PLIB_PMP_WaitStatesStrobeSelect Function

Configures the strobe wait states (during the data transfer) needed to be used with the PMP module.

File

plib_pmp.h

C

```
void PLIB_PMP_WaitStatesStrobeSelect(PMP_MODULE_ID index, PMP_STROBE_WAIT_STATES strobeWaitState);
```

Returns

None.

Description

This function configures the number of peripheral bus clock cycles needed for wait states. Refer to the enumeration PMP_STROBE_WAIT_STATES for the possible settings.

Remarks

This function implements an operation of the DataStrobeWaitStates feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsDataStrobeWaitStates in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_WaitStatesStrobeSelect( PMP_ID_0, PMP_STROBE_WAIT_2 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
strobeWaitState	One of the possible values from PMP_STROBE_WAIT_STATES

Function

void PLIB_PMP_WaitStatesStrobeSelect(PMP_MODULE_ID index,
PMP_STROBE_WAIT_STATES strobeWaitState)

g) Address Handling Functions

PLIB_PMP_AddressGet Function

Gets the current address of the PMP module.

File

plib_pmp.h

C

```
uint32_t PLIB_PMP_AddressGet(PMP_MODULE_ID index);
```

Returns

- address - Device address to be set

Description

This function gets the current address of the PMP module.

Remarks

This function implements an operation of the AddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t address;
address = PLIB_PMP_AddressGet( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

uint32_t PLIB_PMP_AddressGet (PMP_MODULE_ID index)

PLIB_PMP_AddressLinesA0A1Get Function

Gets the value of the address lines PMA0 and PMA1.

File

plib_pmp.h

C

```
uint8_t PLIB_PMP_AddressLinesA0A1Get(PMP_MODULE_ID index);
```

Returns

uint8_t - The two-bit address

Description

This function gets the value of the address lines PMA0 and PMA1. This function is used in the addressable parallel slave port mode.

Remarks

This function implements an operation of the AddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t bufferAddress;
bufferAddress = PLIB_PMP_AddressLinesA0A1Get( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

uint8_t PLIB_PMP_AddressLinesA0A1Get (PMP_MODULE_ID index)

PLIB_PMP_AddressLinesA0A1Set Function

Sets the address lines PMA0 and PMA1 with the value specified.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressLinesA0A1Set(PMP_MODULE_ID index, uint8_t address);
```

Returns

None.

Description

This function sets the address lines PMA0 and PMA1 with the value specified. This function is used in the addressable parallel slave port mode.

Remarks

This function implements an operation of the AddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t bufferAddress = 0x2;
PLIB_PMP_AddressLinesA0A1Set( PMP_ID_0, bufferAddress );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
address	The two-bit address

Function

void PLIB_PMP_AddressLinesA0A1Set (PMP_MODULE_ID index,

uint8_t address)

PLIB_PMP_AddressSet Function

Sets the current address of the PMP module to the specified address.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressSet(PMP_MODULE_ID index, uint32_t address);
```

Returns

None.

Description

This function sets the current address of the PMP module to the specified value.

Remarks

This function implements an operation of the AddressControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t no_of_addressLines = 8;
PLIB_PMP_AddressSet( PMP_ID_0, 0xff );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
address	Device address to be set

Function

void PLIB_PMP_AddressSet (PMP_MODULE_ID index,

uint32_t address)

h) Port Configuration Functions

PLIB_PMP_AddressPortDisable Function

Disables the port lines specified as PMP address lines.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressPortDisable(PMP_MODULE_ID index, PMP_ADDRESS_PORT portfunctions);
```

Returns

None.

Description

This function disables the port lines specified as PMP address lines. They function as normal I/O lines.

Remarks

This function implements an operation of the AddressPortPinControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressPortPinControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
//Example-1
PMP_ADDRESS_PORT portfunctions = PMP_PMA2_TO_PMA13_PORTS;
PLIB_PMP_AddressPortDisable( PMP_ID_0, PMP_PMA2_TO_PMA13_PORTS );

//Example-2
PLIB_PMP_AddressPortDisable( PMP_ID_0,
                             ( PMP_PMA14_PORT | PMP_PMA15_PORT ) );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
portfunctions	One of the possible values from PMP_ADDRESS_PORT

Function

void PLIB_PMP_AddressPortDisable (PMP_MODULE_ID index,
PMP_ADDRESS_PORT portfunctions)

PLIB_PMP_AddressPortEnable Function

Enables the port lines specified as PMP address lines.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressPortEnable(PMP_MODULE_ID index, PMP_ADDRESS_PORT portfunctions);
```

Returns

None.

Description

This function enables the port lines specified as PMP address lines.

Remarks

This function implements an operation of the AddressPortPinControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressPortPinControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
//Example-1
PMP_ADDRESS_PORT portfunctions = PMP_PMA2_TO_PMA13_PORTS;
PLIB_PMP_AddressPortEnable( PMP_ID_0, PMP_PMA2_TO_PMA13_PORTS );

//Example-2
PLIB_PMP_AddressPortEnable( PMP_ID_0,
                            ( PMP_PMA14_PORT | PMP_PMA15_PORT ) );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
portfunctions	One of the possible values from PMP_ADDRESS_PORT

Function

void PLIB_PMP_AddressPortEnable (PMP_MODULE_ID index,

PMP_ADDRESS_PORT portfunctions)

PLIB_PMP_ReadWriteStrobePortDisable Function

Disables the PMP module read strobe port.

File

plib_pmp.h

C

```
void PLIB_PMP_ReadWriteStrobePortDisable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the read strobe port of the PMP module.

Remarks

This function implements an operation of the ReadWriteStrobePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsReadWriteStrobePortControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_ReadWriteStrobePortDisable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_ReadWriteStrobePortDisable (PMP_MODULE_ID index)

PLIB_PMP_ReadWriteStrobePortEnable Function

Enables the PMP module read strobe port.

File

plib_pmp.h

C

```
void PLIB_PMP_ReadWriteStrobePortEnable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the read strobe port of the PMP module.

Remarks

This function implements an operation of the ReadWriteStrobePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsReadWriteStrobePortControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_ReadWriteStrobePortEnable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_ReadWriteStrobePortEnable (PMP_MODULE_ID index)

PLIB_PMP_WriteEnableStrobePortDisable Function

Disables the PMP module write strobe port.

File

plib_pmp.h

C

```
void PLIB_PMP_WriteEnableStrobePortDisable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function disables the write strobe port of the PMP module.

Remarks

This function implements an operation of the WriteEnablePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsWriteEnablePortControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_WriteEnableStrobePortDisable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_WriteEnableStrobePortDisable (PMP_MODULE_ID index)

PLIB_PMP_WriteEnableStrobePortEnable Function

Enables the PMP module write strobe port.

File

plib_pmp.h

C

```
void PLIB_PMP_WriteEnableStrobePortEnable(PMP_MODULE_ID index);
```

Returns

None.

Description

This function enables the write strobe port of the PMP module.

Remarks

This function implements an operation of the WriteEnablePortControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsWriteEnablePortControl in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_WriteEnableStrobePortEnable( PMP_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed

Function

void PLIB_PMP_WriteEnableStrobePortEnable (PMP_MODULE_ID index)

i) Polarity Configuration Functions

PLIB_PMP_AddressLatchPolaritySelect Function

Sets the address latch strobe polarity.

File

plib_pmp.h

C

```
void PLIB_PMP_AddressLatchPolaritySelect(PMP_MODULE_ID index, PMP_POLARITY_LEVEL polarity);
```

Returns

None.

Description

This function sets the address latch polarity to the level specified.

Remarks

This function implements an operation of the AddressLatchPolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsAddressLatchPolarity in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_AddressLatchPolaritySelect( PMP_ID_0, PMP_POLARITY_ACTIVE_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
polarity	Possible polarity levels

Function

void PLIB_PMP_AddressLatchPolaritySelect(PMP_MODULE_ID index,
PMP_POLARITY_LEVEL polarity)

PLIB_PMP_ChipSelectXPolaritySelect Function

Sets the specified Chip Select polarity.

File

plib_pmp.h

C

```
void PLIB_PMP_ChipSelectXPolaritySelect(PMP_MODULE_ID index, PMP_CHIP_SELECT chipSelect, PMP_POLARITY_LEVEL
polarity);
```

Returns

None.

Description

This function sets the specified Chip Select polarity to the level specified.

Remarks

This function implements an operation of the ChipXPolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsChipXPolarity in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PMP_CHIP_SELECT chipSelect = PMP_CHIP_SELECT_ONE;
PLIB_PMP_ChipSelectXPolaritySelect( PMP_ID_0,
                                    chipSelect,
                                    PMP_POLARITY_ACTIVE_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
chipSelect	Identifier for Chip Select
polarity	Possible polarity levels

Function

void PLIB_PMP_ChipSelectXPolaritySelect (PMP_MODULE_ID index,
PMP_CHIP_SELECT chipSelect,
PMP_POLARITY_LEVEL polarity)

PLIB_PMP_ReadWriteStrobePolaritySelect Function

Sets the polarity of the read strobe.

File

plib_pmp.h

C

```
void PLIB_PMP_ReadWriteStrobePolaritySelect(PMP_MODULE_ID index, PMP_POLARITY_LEVEL polarity);
```

Returns

None.

Description

This function sets polarity of the read strobe to the level specified.

Remarks

This function implements an operation of the ReadWritePolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsReadWritePolarity in your application to to automatically determine

whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_ReadWriteStrobePolaritySelect( PMP_ID_0, PMP_POLARITY_ACTIVE_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
polarity	Possible polarity levels

Function

void PLIB_PMP_ReadWriteStrobePolaritySelect (PMP_MODULE_ID index,
PMP_POLARITY_LEVEL polarity)

PLIB_PMP_WriteEnableStrobePolaritySelect Function

Sets the polarity of the write enable strobe.

File

plib_pmp.h

C

```
void PLIB_PMP_WriteEnableStrobePolaritySelect(PMP_MODULE_ID index, PMP_POLARITY_LEVEL polarity);
```

Returns

None.

Description

This function sets the polarity of the write enable strobe to the level specified.

Remarks

This function implements an operation of the WriteEnablePolarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PMP_ExistsWriteEnablePolarity in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PMP_WriteEnableStrobePolaritySelect( PMP_ID_0, PMP_POLARITY_ACTIVE_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be addressed
polarity	Possible polarity levels

Function

void PLIB_PMP_WriteEnableStrobePolaritySelect (PMP_MODULE_ID index,
PMP_POLARITY_LEVEL polarity)

j) Feature Existence Functions

PLIB_PMP_ExistsAddressControl Function

Identifies whether the AddressControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsAddressControl(PMP_MODULE_ID index);
```

Returns

- true - The AddressControl feature is supported on the device
- false - The AddressControl feature is not supported on the device

Description

This function identifies whether the AddressControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_AddressSet
- PLIB_PMP_AddressGet
- PLIB_PMP_AddressLinesA0A1Set
- PLIB_PMP_AddressLinesA0A1Get

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsAddressControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsAddressLatchPolarity Function

Identifies whether the AddressLatchPolarity feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsAddressLatchPolarity(PMP_MODULE_ID index);
```

Returns

- true - The AddressLatchPolarity feature is supported on the device
- false - The AddressLatchPolarity feature is not supported on the device

Description

This function identifies whether the AddressLatchPolarity feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_AddressLatchPolaritySelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsAddressLatchPolarity(PMP_MODULE_ID index)

PLIB_PMP_ExistsAddressLatchStrobePortControl Function

Identifies whether the AddressLatchStrobePortControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsAddressLatchStrobePortControl(PMP_MODULE_ID index);
```

Returns

- true - The AddressLatchStrobePortControl feature is supported on the device
- false - The AddressLatchStrobePortControl feature is not supported on the device

Description

This function identifies whether the AddressLatchStrobePortControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_AddressLatchStrobeEnable
- PLIB_PMP_AddressLatchStrobeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsAddressLatchStrobePortControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsAddressPortPinControl Function

Identifies whether the AddressPortPinControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsAddressPortPinControl(PMP_MODULE_ID index);
```

Returns

- true - The AddressPortPinControl feature is supported on the device
- false - The AddressPortPinControl feature is not supported on the device

Description

This function identifies whether the AddressPortPinControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_AddressPortEnable
- PLIB_PMP_AddressPortDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsAddressPortPinControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsBufferOverFlow Function

Identifies whether the BufferOverFlow feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsBufferOverFlow(PMP_MODULE_ID index);
```

Returns

- true - The BufferOverFlow feature is supported on the device
- false - The BufferOverFlow feature is not supported on the device

Description

This function identifies whether the BufferOverFlow feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_InputOverflowHasOccurred
- PLIB_PMP_InputOverflowClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsBufferOverFlow(PMP_MODULE_ID index)

PLIB_PMP_ExistsBufferRead Function

Identifies whether the BufferRead feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsBufferRead(PMP_MODULE_ID index);
```

Returns

- true - The BufferRead feature is supported on the device
- false - The BufferRead feature is not supported on the device

Description

This function identifies whether the BufferRead feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_InputBufferXByteReceive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsBufferRead(PMP_MODULE_ID index)

PLIB_PMP_ExistsBufferType Function

Identifies whether the BufferType feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsBufferType(PMP_MODULE_ID index);
```

Returns

- true - The BufferType feature is supported on the device
- false - The BufferType feature is not supported on the device

Description

This function identifies whether the BufferType feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_InputBufferTypeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsBufferType(PMP_MODULE_ID index)

PLIB_PMP_ExistsBufferUnderFlow Function

Identifies whether the BufferUnderFlow feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsBufferUnderFlow(PMP_MODULE_ID index);
```

Returns

- true - The BufferUnderFlow feature is supported on the device
- false - The BufferUnderFlow feature is not supported on the device

Description

This function identifies whether the BufferUnderFlow feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_OutputUnderflowHasOccurred
- PLIB_PMP_OutputUnderflowClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsBufferUnderFlow(PMP_MODULE_ID index)

PLIB_PMP_ExistsBufferWrite Function

Identifies whether the BufferWrite feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsBufferWrite(PMP_MODULE_ID index);
```

Returns

- true - The BufferWrite feature is supported on the device
- false - The BufferWrite feature is not supported on the device

Description

This function identifies whether the BufferWrite feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_OutputBufferXByteSend

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsBufferWrite(PMP_MODULE_ID index)

PLIB_PMP_ExistsBusyStatus Function

Identifies whether the BusyStatus feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsBusyStatus(PMP_MODULE_ID index);
```

Returns

- true - The BusyStatus feature is supported on the device
- false - The BusyStatus feature is not supported on the device

Description

This function identifies whether the BusyStatus feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_PortIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsBusyStatus(PMP_MODULE_ID index)

PLIB_PMP_ExistsChipSelectEnable Function

Identifies whether the ChipSelectEnable feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsChipSelectEnable(PMP_MODULE_ID index);
```

Returns

- true - The ChipSelectEnable feature is supported on the device
- false - The ChipSelectEnable feature is not supported on the device

Description

This function identifies whether the ChipSelectEnable feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_ChipSelectXEnable
- PLIB_PMP_ChipSelectXDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsChipSelectEnable(PMP_MODULE_ID index)

PLIB_PMP_ExistsChipSelectoperation Function

Identifies whether the ChipSelectoperation feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsChipSelectoperation(PMP_MODULE_ID index);
```

Returns

- true - The ChipSelectoperation feature is supported on the device
- false - The ChipSelectoperation feature is not supported on the device

Description

This function identifies whether the ChipSelectoperation feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_ChipSelectFunctionSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsChipSelectoperation(PMP_MODULE_ID index)

PLIB_PMP_ExistsChipXPolarity Function

Identifies whether the ChipXPolarity feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsChipXPolarity(PMP_MODULE_ID index);
```

Returns

- true - The ChipXPolarity feature is supported on the device
- false - The ChipXPolarity feature is not supported on the device

Description

This function identifies whether the ChipXPolarity feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_ChipSelectXPolaritySelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsChipXPolarity(PMP_MODULE_ID index)

PLIB_PMP_ExistsCSXActiveStatus Function

Identifies whether the CSXActiveStatus feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsCSXActiveStatus(PMP_MODULE_ID index);
```

Returns

- true - The CSXActiveStatus feature is supported on the device
- false - The CSXActiveStatus feature is not supported on the device

Description

This function identifies whether the CSXActiveStatus feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_ChipSelectXIsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsCSXActiveStatus(PMP_MODULE_ID index)

PLIB_PMP_ExistsDataHoldWaitStates Function

Identifies whether the DataHoldWaitStates feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDataHoldWaitStates(PMP_MODULE_ID index);
```

Returns

- true - The DataHoldWaitStates feature is supported on the device
- false - The DataHoldWaitStates feature is not supported on the device

Description

This function identifies whether the DataHoldWaitStates feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_WaitStatesDataHoldSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDataHoldWaitStates(PMP_MODULE_ID index)

PLIB_PMP_ExistsDataSetUpWaitStates Function

Identifies whether the DataSetUpWaitStates feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDataSetUpWaitStates(PMP_MODULE_ID index);
```

Returns

- true - The DataSetUpWaitStates feature is supported on the device
- false - The DataSetUpWaitStates feature is not supported on the device

Description

This function identifies whether the DataSetUpWaitStates feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_WaitStatesDataSetUpSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDataSetUpWaitStates(PMP_MODULE_ID index)

PLIB_PMP_ExistsDataStrobeWaitStates Function

Identifies whether the DataStrobeWaitStates feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDataStrobeWaitStates(PMP_MODULE_ID index);
```

Returns

- true - The DataStrobeWaitStates feature is supported on the device
- false - The DataStrobeWaitStates feature is not supported on the device

Description

This function identifies whether the DataStrobeWaitStates feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_WaitStatesStrobeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDataStrobeWaitStates(PMP_MODULE_ID index)

PLIB_PMP_ExistsDataTransferSize Function

Identifies whether the DataTransferSize feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDataTransferSize(PMP_MODULE_ID index);
```

Returns

- true - The DataTransferSize feature is supported on the device
- false - The DataTransferSize feature is not supported on the device

Description

This function identifies whether the DataTransferSize feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_DataSizeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDataTransferSize(PMP_MODULE_ID index)

PLIB_PMP_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsEnableControl(PMP_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_Disable
- PLIB_PMP_Enable
- PLIB_PMP_IsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsEnableControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsIncrementMode Function

Identifies whether the IncrementMode feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsIncrementMode(PMP_MODULE_ID index);
```

Returns

- true - The IncrementMode feature is supported on the device
- false - The IncrementMode feature is not supported on the device

Description

This function identifies whether the IncrementMode feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_AddressIncrementModeSelect
- PLIB_PMP_AddressIncrementModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsIncrementMode(PMP_MODULE_ID index)

PLIB_PMP_ExistsInputBufferFull Function

Identifies whether the InputBufferFull feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsInputBufferFull(PMP_MODULE_ID index);
```

Returns

- true - The InputBufferFull feature is supported on the device
- false - The InputBufferFull feature is not supported on the device

Description

This function identifies whether the InputBufferFull feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_InputBuffersAreFull

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsInputBufferFull(PMP_MODULE_ID index)

PLIB_PMP_ExistsInputBufferXStatus Function

Identifies whether the InputBufferXStatus feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsInputBufferXStatus(PMP_MODULE_ID index);
```

Returns

- true - The InputBufferXStatus feature is supported on the device
- false - The InputBufferXStatus feature is not supported on the device

Description

This function identifies whether the InputBufferXStatus feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_InputBufferXIsFull
- PLIB_PMP_IsDataReceived

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsInputBufferXStatus(PMP_MODULE_ID index)

PLIB_PMP_ExistsInterruptMode Function

Identifies whether the InterruptMode feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsInterruptMode(PMP_MODULE_ID index);
```

Returns

- true - The InterruptMode feature is supported on the device
- false - The InterruptMode feature is not supported on the device

Description

This function identifies whether the InterruptMode feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_InterruptModeSelect
- PLIB_PMP_InterruptModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsInterruptMode(PMP_MODULE_ID index)

PLIB_PMP_ExistsMasterRXTX Function

Identifies whether the MasterRXTX feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsMasterRXTX(PMP_MODULE_ID index);
```

Returns

- true - The MasterRXTX feature is supported on the device
- false - The MasterRXTX feature is not supported on the device

Description

This function identifies whether the MasterRXTX feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_MasterSend
- PLIB_PMP_MasterReceive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsMasterRXTX(PMP_MODULE_ID index)

PLIB_PMP_ExistsMUXModeSelect Function

Identifies whether the MUXModeSelect feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsMUXModeSelect(PMP_MODULE_ID index);
```

Returns

- true - The MUXModeSelect feature is supported on the device
- false - The MUXModeSelect feature is not supported on the device

Description

This function identifies whether the MUXModeSelect feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_MultiplexModeSelect
- PLIB_PMP_MultiplexModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsMUXModeSelect(PMP_MODULE_ID index)

PLIB_PMP_ExistsOperationMode Function

Identifies whether the OperationMode feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsOperationMode(PMP_MODULE_ID index);
```

Returns

- true - The OperationMode feature is supported on the device
- false - The OperationMode feature is not supported on the device

Description

This function identifies whether the OperationMode feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_OperationModeSelect
- PLIB_PMP_OperationModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsOperationMode(PMP_MODULE_ID index)

PLIB_PMP_ExistsOutPutBufferEmpty Function

Identifies whether the OutPutBufferEmpty feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsOutPutBufferEmpty(PMP_MODULE_ID index);
```

Returns

- true - The OutPutBufferEmpty feature is supported on the device
- false - The OutPutBufferEmpty feature is not supported on the device

Description

This function identifies whether the OutPutBufferEmpty feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_OutputBuffersAreEmpty

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsOutPutBufferEmpty(PMP_MODULE_ID index)

PLIB_PMP_ExistsOutputBufferXStatus Function

Identifies whether the OutputBufferXStatus feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsOutputBufferXStatus(PMP_MODULE_ID index);
```

Returns

- true - The OutputBufferXStatus feature is supported on the device
- false - The OutputBufferXStatus feature is not supported on the device

Description

This function identifies whether the OutputBufferXStatus feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_OutputBufferXIsEmpty
- PLIB_PMP_IsDataTransmitted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsOutputBufferXStatus(PMP_MODULE_ID index)

PLIB_PMP_ExistsReadWritePolarity Function

Identifies whether the ReadWritePolarity feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsReadWritePolarity(PMP_MODULE_ID index);
```

Returns

- true - The ReadWritePolarity feature is supported on the device
- false - The ReadWritePolarity feature is not supported on the device

Description

This function identifies whether the ReadWritePolarity feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_ReadWriteStrobePolaritySelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsReadWritePolarity(PMP_MODULE_ID index)

PLIB_PMP_ExistsReadWriteStrobePortControl Function

Identifies whether the ReadWriteStrobePortControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsReadWriteStrobePortControl(PMP_MODULE_ID index);
```

Returns

- true - The ReadWriteStrobePortControl feature is supported on the device
- false - The ReadWriteStrobePortControl feature is not supported on the device

Description

This function identifies whether the ReadWriteStrobePortControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_ReadWriteStrobePortEnable
- PLIB_PMP_ReadWriteStrobePortDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsReadWriteStrobePortControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsSlaveRX Function

Identifies whether the SlaveRX feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsSlaveRX(PMP_MODULE_ID index);
```

Returns

- true - The SlaveRX feature is supported on the device
- false - The SlaveRX feature is not supported on the device

Description

This function identifies whether the SlaveRX feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_SlaveReceive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsSlaveRX(PMP_MODULE_ID index)

PLIB_PMP_ExistsSlaveTX Function

Identifies whether the SlaveTX feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsSlaveTX(PMP_MODULE_ID index);
```

Returns

- true - The SlaveTX feature is supported on the device
- false - The SlaveTX feature is not supported on the device

Description

This function identifies whether the SlaveTX feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_SlaveSend

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsSlaveTX(PMP_MODULE_ID index)

PLIB_PMP_ExistsStopInIdleControl Function

Identifies whether the StopInIdleControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsStopInIdleControl(PMP_MODULE_ID index);
```

Returns

- true - The StopInIdleControl feature is supported on the device
- false - The StopInIdleControl feature is not supported on the device

Description

This function identifies whether the StopInIdleControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_StopInIdleEnable
- PLIB_PMP_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsStopInIdleControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsWriteEnablePolarity Function

Identifies whether the WriteEnablePolarity feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsWriteEnablePolarity(PMP_MODULE_ID index);
```

Returns

- true - The WriteEnablePolarity feature is supported on the device
- false - The WriteEnablePolarity feature is not supported on the device

Description

This function identifies whether the WriteEnablePolarity feature is available on the PMP module. When this function returns true, this function is supported on the device:

- PLIB_PMP_WriteEnableStrobePolaritySelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsWriteEnablePolarity(PMP_MODULE_ID index)

PLIB_PMP_ExistsWriteEnablePortControl Function

Identifies whether the WriteEnablePortControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsWriteEnablePortControl(PMP_MODULE_ID index);
```

Returns

- true - The WriteEnablePortControl feature is supported on the device
- false - The WriteEnablePortControl feature is not supported on the device

Description

This function identifies whether the WriteEnablePortControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_WriteEnableStrobePortEnable
- PLIB_PMP_WriteEnableStrobePortDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsWriteEnablePortControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsDualBufferControl Function

Identifies whether the DualBufferControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDualBufferControl(PMP_MODULE_ID index);
```

Returns

- true - The DualBufferControl feature is supported on the device
- false - The DualBufferControl feature is not supported on the device

Description

This function identifies whether the DualBufferControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_DualBufferDisable
- PLIB_PMP_DualBufferEnable
- PLIB_PMP_DualBufferIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDualBufferControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsDualModeMasterRXTX Function

Identifies whether the DualModeMasterRXTX feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDualModeMasterRXTX(PMP_MODULE_ID index);
```

Returns

- true - The DualModeMasterRXTX feature is supported on the device
- false - The DualModeMasterRXTX feature is not supported on the device

Description

This function identifies whether the DualModeMasterRXTX feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_DualModeMasterSend
- PLIB_PMP_DualModeMasterReceive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDualModeMasterRXTX(PMP_MODULE_ID index)

PLIB_PMP_ExistsDualModeReadAddressControl Function

Identifies whether the DualModeReadAddressControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDualModeReadAddressControl(PMP_MODULE_ID index);
```

Returns

- true - The DualModeReadAddressControl feature is supported on the device
- false - The DualModeReadAddressControl feature is not supported on the device

Description

This function identifies whether the DualModeReadAddressControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_DualModeReadAddressSet
- PLIB_PMP_DualModeReadAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDualModeReadAddressControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsDualModeWriteAddressControl Function

Identifies whether the DualModeWriteAddressControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsDualModeWriteAddressControl(PMP_MODULE_ID index);
```

Returns

- true - The DualModeWriteAddressControl feature is supported on the device
- false - The DualModeWriteAddressControl feature is not supported on the device

Description

This function identifies whether the DualModeWriteAddressControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_DualModeWriteAddressSet
- PLIB_PMP_DualModeWriteAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsDualModeWriteAddressControl(PMP_MODULE_ID index)

PLIB_PMP_ExistsReadChipSelectEnable Function

Identifies whether the ReadChipSelectEnable feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsReadChipSelectEnable(PMP_MODULE_ID index);
```

Returns

- true - The ReadChipSelectEnable feature is supported on the device
- false - The ReadChipSelectEnable feature is not supported on the device

Description

This function identifies whether the ReadChipSelectEnable feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_ReadChipSelectXEnable
- PLIB_PMP_ReadChipSelectXDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsReadChipSelectEnable(PMP_MODULE_ID index)

PLIB_PMP_ExistsWriteChipSelectEnable Function

Identifies whether the WriteChipSelectEnable feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsWriteChipSelectEnable(PMP_MODULE_ID index);
```

Returns

- true - The WriteChipSelectEnable feature is supported on the device
- false - The WriteChipSelectEnable feature is not supported on the device

Description

This function identifies whether the WriteChipSelectEnable feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_WriteChipSelectXEnable
- PLIB_PMP_WriteChipSelectXDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsWriteChipSelectEnable(PMP_MODULE_ID index)

PLIB_PMP_ExistsStartReadControl Function

Identifies whether the StartReadControl feature exists on the PMP module.

File

plib_pmp.h

C

```
bool PLIB_PMP_ExistsStartReadControl(PMP_MODULE_ID index);
```

Returns

- true - The StartReadControl feature is supported on the device
- false - The StartReadControl feature is not supported on the device

Description

This function identifies whether the StartReadControl feature is available on the PMP module. When this function returns true, these functions are supported on the device:

- PLIB_PMP_ReadCycleStart
- PLIB_PMP_ReadCycleIsStarted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PMP_ExistsStartReadControl(PMP_MODULE_ID index)

k) Data Types and Constants

PMP_ACK_MODE Enumeration

Defines the different mode configurations in which the PMP can be enabled.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_USE_ACK_WITH_TIMEOUT,
  PMP_USE_ACK,
```

```
  PMP_ACK_DISABLED
} PMP_ACK_MODE;
```

Members

Members	Description
PMP_USE_ACK_WITH_TIMEOUT	Acknowledge used with a timeout
PMP_USE_ACK	Acknowledge is used
PMP_ACK_DISABLED	Acknowledge is not used

Description

PMP Acknowledge Modes

This data type defines the different configurations by which the PMP can be enabled.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_ADDRESS_HOLD_LATCH_WAIT_STATES Enumeration

PMP hold after address latch strobe wait states configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_ADDRESS_HOLD_ONE_AND_ONE_QUARTER,
  PMP_ADDRESS_HOLD_ONE_QUARTER
} PMP_ADDRESS_HOLD_LATCH_WAIT_STATES;
```

Members

Members	Description
PMP_ADDRESS_HOLD_ONE_AND_ONE_QUARTER	Data Wait of 1 1/4 Peripheral Bus Clock Cycles
PMP_ADDRESS_HOLD_ONE_QUARTER	Data Wait of 1/4 Peripheral Bus Clock Cycles

Description

PMP Address Latch Strobe Wait States

This data type defines the different configurations by which the PMP holds after address latch strobe wait states can be configured. Refer to the specific device data sheet for the exact clock cycle timing.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_ADDRESS_LATCH Enumeration

Address Latch Strobe configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_ADDRESS_LATCH_UPPER,
  PMP_ADDRESS_LATCH_HIGH,
  PMP_ADDRESS_LATCH_LOW
} PMP_ADDRESS_LATCH;
```

Members

Members	Description
PMP_ADDRESS_LATCH_UPPER	Address latch upper
PMP_ADDRESS_LATCH_HIGH	Address latch high
PMP_ADDRESS_LATCH_LOW	Address latch low

Description

PMP Address Latch

This data type defines the different configurations by which the PMP address latch strobes can be configured.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_ADDRESS_LATCH_WAIT_STATES Enumeration

PMP address latch strobe wait states configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_ADDRESS_WAIT_THREE_AND_HALF,
  PMP_ADDRESS_WAIT_TWO_AND_HALF,
  PMP_ADDRESS_WAIT_ONE_AND_HALF,
  PMP_ADDRESS_WAIT_HALF
} PMP_ADDRESS_LATCH_WAIT_STATES;
```

Members

Members	Description
PMP_ADDRESS_WAIT_THREE_AND_HALF	Data Wait of 3 1/2 Peripheral Bus Clock Cycles
PMP_ADDRESS_WAIT_TWO_AND_HALF	Data Wait of 2 1/2 Peripheral Bus Clock Cycles
PMP_ADDRESS_WAIT_ONE_AND_HALF	Data Wait of 1 1/2 Peripheral Bus Clock Cycles
PMP_ADDRESS_WAIT_HALF	Data Wait of 1/2 Peripheral Bus Clock Cycles

Description

PMP Address Latch Strobe Wait States

This data type defines the different configurations by which the PMP address latch strobe wait states can be configured. Refer to the specific device data sheet for the exact clock cycle timing.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_ADDRESS_PORT Enumeration

Defines the different address lines that are available for configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_PMA0_PORT,
  PMP_PMA1_PORT,
  PMP_PMA2_TO_PMA13_PORTS,
  PMP_PMA2_PORT,
  PMP_PMA3_PORT,
  PMP_PMA4_PORT,
  PMP_PMA5_PORT,
  PMP_PMA6_PORT,
  PMP_PMA7_PORT,
  PMP_PMA8_PORT,
  PMP_PMA9_PORT,
  PMP_PMA10_PORT,
  PMP_PMA11_PORT,
  PMP_PMA12_PORT,
  PMP_PMA13_PORT,
  PMP_PMA14_PORT,
  PMP_PMA15_PORT,
  PMP_PMA16_TO_PMA22_PORTS,
```

```
  PMP_PMA2_TO_PMA10_PORTS
} PMP_ADDRESS_PORT;
```

Members

Members	Description
PMP_PMA0_PORT	Address line 0 port (make as general purpose I/O or PMP address line)
PMP_PMA1_PORT	Address line 1 port (make as general purpose I/O or PMP address line)
PMP_PMA2_TO_PMA13_PORTS	Address line 2 to 13 ports (make as general purpose I/O or PMP address lines)
PMP_PMA2_PORT	Address line 2 port (make as general purpose I/O or PMP address line)
PMP_PMA3_PORT	Address line 3 port (make as general purpose I/O or PMP address line)
PMP_PMA4_PORT	Address line 4 port (make as general purpose I/O or PMP address line)
PMP_PMA5_PORT	Address line 5 port (make as general purpose I/O or PMP address line)
PMP_PMA6_PORT	Address line 6 port (make as general purpose I/O or PMP address line)
PMP_PMA7_PORT	Address line 7 port (make as general purpose I/O or PMP address line)
PMP_PMA8_PORT	Address line 8 port (make as general purpose I/O or PMP address line)
PMP_PMA9_PORT	Address line 9 port (make as general purpose I/O or PMP address line)
PMP_PMA10_PORT	Address line 10 port (make as general purpose I/O or PMP address line)
PMP_PMA11_PORT	Address line 11 port (make as general purpose I/O or PMP address line)
PMP_PMA12_PORT	Address line 12 port (make as general purpose I/O or PMP address line)
PMP_PMA13_PORT	Address line 13 port (make as general purpose I/O or PMP address line)
PMP_PMA14_PORT	Address line 14 port (make as general purpose I/O or PMP address line)
PMP_PMA15_PORT	Address line 15 port (make as general purpose I/O or PMP address line)
PMP_PMA16_TO_PMA22_PORTS	Address line 16 to 22 ports (make as general purpose I/O or PMP address lines)
PMP_PMA2_TO_PMA10_PORTS	Address line 2 to 10 ports (make as general purpose I/O or PMP address lines)

Description

PMP Address Port Pins

This data type defines the different address that can be configured by the PMP module. The user application can make each pin as a general purpose I/O or part of PMP functionality.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h). These enumerator values can be passed in a bitwise ORed fashion to select multiple ports at a time.

PMP_ALTERNATE_MASTER_WAIT_STATES Enumeration

PMP alternate master wait states.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_ALTERNATE_MASTER_WAIT_10,
  PMP_ALTERNATE_MASTER_WAIT_9,
  PMP_ALTERNATE_MASTER_WAIT_8,
  PMP_ALTERNATE_MASTER_WAIT_7,
  PMP_ALTERNATE_MASTER_WAIT_6,
  PMP_ALTERNATE_MASTER_WAIT_5,
  PMP_ALTERNATE_MASTER_WAIT_4,
  PMP_ALTERNATE_MASTER_WAIT_3
} PMP_ALTERNATE_MASTER_WAIT_STATES;
```

Members

Members	Description
PMP_ALTERNATE_MASTER_WAIT_10	Wait of 10 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_9	Wait of 9 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_8	Wait of 8 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_7	Wait of 7 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_6	Wait of 6 Alternate Master Cycles

PMP_ALTERNATE_MASTER_WAIT_5	Wait of 5 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_4	Wait of 4 Alternate Master Cycles
PMP_ALTERNATE_MASTER_WAIT_3	Wait of 3 Alternate Master Cycles

Description

PMP Alternate Master Wait States

This data type defines the different configurations by which the PMP alternate master wait states can be configured.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_CHIP_SELECT Enumeration

PMP Chip Select data type.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_CHIP_SELECT_ONE,
  PMP_CHIP_SELECT_TWO
} PMP_CHIP_SELECT;
```

Members

Members	Description
PMP_CHIP_SELECT_ONE	Chip Select One
PMP_CHIP_SELECT_TWO	Chip Select Two

Description

PMP Chip Select

This data type defines the different Chip Select lines of the PMP module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_CHIPSELECT_FUNCTION Enumeration

Defines different functions available for the Chip Select lines multiplexed with address lines.

File

plib_pmp_help.h

C

```
typedef enum {
  PMCS1_AS_CHIP_SELECT,
  PMCS1_AS_ADDRESS_LINE,
  PMCS1_PMCS2_AS_ADDRESS_LINES,
  PMCS1_AS_ADDRESS_LINE_PMCS2_AS_CHIP_SELECT,
  PMCS1_AND_PMCS2_AS_CHIP_SELECT
} PMP_CHIPSELECT_FUNCTION;
```

Members

Members	Description
PMCS1_AS_CHIP_SELECT	CS1 used as Chip Select
PMCS1_AS_ADDRESS_LINE	CS1 used as address line
PMCS1_PMCS2_AS_ADDRESS_LINES	CS1 and CS2 used as address lines
PMCS1_AS_ADDRESS_LINE_PMCS2_AS_CHIP_SELECT	CS1 used as address line and CS2 as Chip Select
PMCS1_AND_PMCS2_AS_CHIP_SELECT	Both CS1 and CS2 used as Chip Selects

Description

PMP Chip Select pin functions

This data type defines different functions of Chip Select pins.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_DATA_HOLD_STATES Enumeration

PMP Data Hold after strobe wait state.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_DATA_HOLD_FOUR,
  PMP_DATA_HOLD_THREE,
  PMP_DATA_HOLD_TWO,
  PMP_DATA_HOLD_ONE
} PMP_DATA_HOLD_STATES;
```

Members

Members	Description
PMP_DATA_HOLD_FOUR	Data Hold of 4 Peripheral Bus Clock Cycles
PMP_DATA_HOLD_THREE	Data Hold of 3 Peripheral Bus Clock Cycles
PMP_DATA_HOLD_TWO	Data Hold of 2 Peripheral Bus Clock Cycles
PMP_DATA_HOLD_ONE	Data Hold of 1 Peripheral Bus Clock Cycles

Description

PMP Data Hold after strobe wait state

This data type defines the different data Hold after strobe wait states.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_DATA_LENGTH Enumeration

Possible data lengths handled by the PMP module.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_DATA_8_BITS,
  PMP_DATA_16_BITS,
  PMP_DATA_32_BITS
} PMP_DATA_LENGTH;
```

Members

Members	Description
PMP_DATA_8_BITS	8-bit data length
PMP_DATA_16_BITS	16-bit data length
PMP_DATA_32_BITS	32-bit data length

Description

PMP data length

This data type defines the possible data lengths handled by the PMP module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_DATA_SIZE Enumeration

PMP data size.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_DATA_SIZE_8_BITS = 0x0,
  PMP_DATA_SIZE_16_BITS = 0x1
} PMP_DATA_SIZE;
```

Members

Members	Description
PMP_DATA_SIZE_8_BITS = 0x0	Data length of 8-bits
PMP_DATA_SIZE_16_BITS = 0x1	Data length of 16-bits

Description

PMP DATA Size

This data type defines the different configurations for the data lengths handled by the PMP module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_DATA_WAIT_STATES Enumeration

PMP data setup time configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_DATA_WAIT_FOUR,
  PMP_DATA_WAIT_THREE,
  PMP_DATA_WAIT_TWO,
  PMP_DATA_WAIT_ONE
} PMP_DATA_WAIT_STATES;
```

Members

Members	Description
PMP_DATA_WAIT_FOUR	Data Wait of 4 Peripheral Bus Clock Cycles
PMP_DATA_WAIT_THREE	Data Wait of 3 Peripheral Bus Clock Cycles
PMP_DATA_WAIT_TWO	Data Wait of 2 Peripheral Bus Clock Cycles
PMP_DATA_WAIT_ONE	Data Wait of 1 Peripheral Bus Clock Cycles

Description

PMP Data setup to read/write/address phase wait state configuration

This data type defines the different wait state configurations for setup to read/write/address phase.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_INCREMENT_MODE Enumeration

PMP address incrementing configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_BUFFERS_AUTO_INCREMENT,
  PMP_ADDRESS_AUTO_DECREMENT,
  PMP_ADDRESS_AUTO_INCREMENT,
  PMP_ADDRESS_INCREMENT_DECREMENT_DISABLE
} PMP_INCREMENT_MODE;
```

Members

Members	Description
PMP_BUFFERS_AUTO_INCREMENT	Read and write buffers auto-increment
PMP_ADDRESS_AUTO_DECREMENT	Decrement PMP Address by one every read/write cycle
PMP_ADDRESS_AUTO_INCREMENT	Increment PMP Address by one every read/write cycle
PMP_ADDRESS_INCREMENT_DECREMENT_DISABLE	PMP Address does not change

Description

PMP Increment Modes

This data type defines the different configurations by which the PMP address incrementing can be configured.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_INPUT_BUFFER_TYPE Enumeration

PMP Input Buffers type.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_INPUT_BUFFER_TTL,
  PMP_INPUT_BUFFER_SCHMITT
} PMP_INPUT_BUFFER_TYPE;
```

Members

Members	Description
PMP_INPUT_BUFFER_TTL	TTL input buffer
PMP_INPUT_BUFFER_SCHMITT	Schmitt trigger input buffer

Description

PMP Input Buffers type

This data type defines the input buffer types.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_INTERRUPT_MODE Enumeration

PMP interrupt request mode data type.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_INTERRUPT_BUFFER_3_IS_ACCESSED,
  PMP_INTERRUPT_EVERY_RW_CYCLE,
```

```
  PMP_INTERRUPT_NONE
} PMP_INTERRUPT_MODE;
```

Members

Members	Description
PMP_INTERRUPT_BUFFER_3_IS_ACCESSED	Interrupt generated when Read/Write Buffer 3 is read/written
PMP_INTERRUPT_EVERY_RW_CYCLE	Interrupt Occurs Every Read/Write Cycle
PMP_INTERRUPT_NONE	No interrupt generated

Description

PMP Interrupt Mode

This data type defines the different configurations by which the PMP can be configured for interrupt requests.

Remarks

None.

PMP_MASTER_MODE Enumeration

Defines the different mode configurations in which the PMP can be enabled.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_ALTERNATE_MASTER_DIRECT,
  PMP_ALTERNATE_MASTER,
  PMP_CPU_MASTER
} PMP_MASTER_MODE;
```

Members

Members	Description
PMP_ALTERNATE_MASTER_DIRECT	Alternate Master has PMP control with Direct I/O access
PMP_ALTERNATE_MASTER	Alternate Master has PMP control
PMP_CPU_MASTER	CPU has PMP control

Description

PMP Master Modes

This data type defines the different configurations by which the PMP can be enabled.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_MODULE_ID Enumeration

Possible instances of the PMP module.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_ID_0
} PMP_MODULE_ID;
```

Members

Members	Description
PMP_ID_0	first instance of the PMP

Description

PMP module ID

This data type defines the possible instances of the PMP module.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_MUX_MODE Enumeration

Defines the different mode configurations in which the PMP can be enabled.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_MUX_ADDRESS_PHASES_3,
  PMP_MUX_ADDRESS_PHASES_2,
  PMP_MUX_ADDRESS_PHASES_1,
  PMP_MUX_LINES_16_ADDRESS_16_DATA,
  PMP_MUX_LINES_16_ADDRESS_8_DATA,
  PMP_MUX_LINES_8_ADDRESS_8_DATA,
  PMP_MUX_NONE
} PMP_MUX_MODE;
```

Members

Members	Description
PMP_MUX_ADDRESS_PHASES_3	Lower address bits are multiplexed with data bits using 3 address phase
PMP_MUX_ADDRESS_PHASES_2	Lower address bits are multiplexed with data bits using 2 address phases
PMP_MUX_ADDRESS_PHASES_1	Lower address bits are multiplexed with data bits using 1 address phase
PMP_MUX_LINES_16_ADDRESS_16_DATA	16 address lines are multiplexed on 16 data lines
PMP_MUX_LINES_16_ADDRESS_8_DATA	16 address lines multiplexed on 8 data lines
PMP_MUX_LINES_8_ADDRESS_8_DATA	8 bits of address is multiplexed on 8 data lines
PMP_MUX_NONE	No multiplexing of address and data lines

Description

PMP Multiplex Modes

This data type defines the different configurations by which the PMP can be enabled.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_OPERATION_MODE Enumeration

Defines the different mode configurations in which the PMP can be enabled.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_MASTER_READ_WRITE_STROBES_MULTIPLEXED,
  PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT,
  PMP_BUFFERED_SLAVE,
  PMP_ENHANCED_SLAVE,
  PMP_LEGACY_SLAVE
} PMP_OPERATION_MODE;
```

Members

Members	Description
PMP_MASTER_READ_WRITE_STROBES_MULTIPLEXED	Master mode 1, the read and the write strobes share a single line. The enable strobe is used to decode the info sent on read/write strobe line
PMP_MASTER_READ_WRITE_STROBES_INDEPENDENT	Master mode 2, the read and write strobes are on independent lines
PMP_BUFFERED_SLAVE	Buffered Slave mode
PMP_ENHANCED_SLAVE	Enhanced Slave mode

PMP_LEGACY_SLAVE	Legacy Parallel Slave mode

Description

PMP Operation Modes

This data type defines the different configurations by which the PMP can be enabled.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_PMBE_PORT Enumeration

Defines the available Byte Enable ports.

File

plib_pmp_help.h

C

```
typedef enum {
  PMBE_0,
  PMBE_1
} PMP_PMBE_PORT;
```

Members

Members	Description
PMBE_0	Byte Enable Port-0
PMBE_1	Byte Enable Port-1

Description

PMP Byte Enable port.

This data type defines the available Byte Enable ports.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PMP_POLARITY_LEVEL Enumeration

Possible polarity levels for the PMP pins.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_POLARITY_ACTIVE_HIGH,
  PMP_POLARITY_ACTIVE_LOW
} PMP_POLARITY_LEVEL;
```

Members

Members	Description
PMP_POLARITY_ACTIVE_HIGH	Active high polarity
PMP_POLARITY_ACTIVE_LOW	Active low polarity

Description

PMP Pins Polarity

This data type defines the possible polarity levels for the PMP pins.

Remarks

None.

PMP_STROBE_WAIT_STATES Enumeration

PMP strobe signal wait time configuration.

File

plib_pmp_help.h

C

```
typedef enum {
  PMP_STROBE_WAIT_FIFTEEN,
  PMP_STROBE_WAIT_FOURTEEN,
  PMP_STROBE_WAIT_THIRTEEN,
  PMP_STROBE_WAIT_TWELVE,
  PMP_STROBE_WAIT_ELEVEN,
  PMP_STROBE_WAIT_TEN,
  PMP_STROBE_WAIT_NINE,
  PMP_STROBE_WAIT_EIGHT,
  PMP_STROBE_WAIT_SEVEN,
  PMP_STROBE_WAIT_SIX,
  PMP_STROBE_WAIT_FIVE,
  PMP_STROBE_WAIT_FOUR,
  PMP_STROBE_WAIT_THREE,
  PMP_STROBE_WAIT_TWO,
  PMP_STROBE_WAIT_ONE,
  PMP_STROBE_WAIT_NONE
} PMP_STROBE_WAIT_STATES;
```

Members

Members	Description
PMP_STROBE_WAIT_FIFTEEN	Strobe Wait of 15 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_FOURTEEN	Strobe Wait of 14 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_THIRTEEN	Strobe Wait of 13 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_TWELVE	Strobe Wait of 12 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_ELEVEN	Strobe Wait of 11 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_TEN	Strobe Wait of 10 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_NINE	Strobe Wait of 9 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_EIGHT	Strobe Wait of 8 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_SEVEN	Strobe Wait of 7 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_SIX	Strobe Wait of 6 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_FIVE	Strobe Wait of 5 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_FOUR	Strobe Wait of 4 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_THREE	Strobe Wait of 3 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_TWO	Strobe Wait of 2 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_ONE	Strobe Wait of 1 Peripheral Bus Clock Cycles
PMP_STROBE_WAIT_NONE	No wait states

Description

PMP Read to Byte enable strobe configuration

This data type defines the different configurations by which the PMP strobe wait signal time can be configured.

Remarks

This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

Files

Files

Name	Description
plib_pmp.h	Parallel Master Port (PMP) Peripheral Library Interface Header.
plib_pmp_help.h	Parallel Master Port (PMP) Peripheral Library Help Header.

Description

This section lists the source and header files used by the library.

plib_pmp.h

Parallel Master Port (PMP) Peripheral Library Interface Header.

Functions

	Name	Description
	PLIB_PMP_AddressGet	Gets the current address of the PMP module.
	PLIB_PMP_AddressIncrementModeGet	Gets the increment mode being used with the address of the PMP module.
	PLIB_PMP_AddressIncrementModeSelect	Configures the increment mode being used with the address of the PMP module.
	PLIB_PMP_AddressLatchPolaritySelect	Sets the address latch strobe polarity.
	PLIB_PMP_AddressLatchStrobeDisable	Disables the specific address latch strobe.
	PLIB_PMP_AddressLatchStrobeEnable	Enables the specific address latch strobe.
	PLIB_PMP_AddressLinesA0A1Get	Gets the value of the address lines PMA0 and PMA1.
	PLIB_PMP_AddressLinesA0A1Set	Sets the address lines PMA0 and PMA1 with the value specified.
	PLIB_PMP_AddressPortDisable	Disables the port lines specified as PMP address lines.
	PLIB_PMP_AddressPortEnable	Enables the port lines specified as PMP address lines.
	PLIB_PMP_AddressSet	Sets the current address of the PMP module to the specified address.
	PLIB_PMP_ChipSelectFunctionSelect	Defines the functionality of the pins intended to be used as Chip Select.
	PLIB_PMP_ChipSelectXDisable	Configures the Chip Select.
	PLIB_PMP_ChipSelectXEnable	Configures the Chip Select.
	PLIB_PMP_ChipSelectXIsActive	Gets the current status of the specified Chip Select.
	PLIB_PMP_ChipSelectXPolaritySelect	Sets the specified Chip Select polarity.
	PLIB_PMP_DataSizeSelect	Enables data transfer size for the PMP module.
	PLIB_PMP_Disable	Disables the specific PMP module.
	PLIB_PMP_DualBufferDisable	Disables the specific PMP module.
	PLIB_PMP_DualBufferEnable	Enables PMP dual Read/Write buffer.
	PLIB_PMP_DualBufferIsEnabled	Checks whether or not the PMP module is enabled.
	PLIB_PMP_DualModeMasterReceive	Receives the data in the Master Dual mode.
	PLIB_PMP_DualModeMasterSend	Sends the specified data in Dual Master mode.
	PLIB_PMP_DualModeReadAddressGet	Gets the current read address of the PMP module.
	PLIB_PMP_DualModeReadAddressSet	Sets the address to be written in Dual mode.
	PLIB_PMP_DualModeWriteAddressGet	Gets the current write address of the PMP module.
	PLIB_PMP_DualModeWriteAddressSet	Sets the address to be written in Dual mode.
	PLIB_PMP_Enable	Enables the specific PMP module.
	PLIB_PMP_ExistsAddressControl	Identifies whether the AddressControl feature exists on the PMP module.
	PLIB_PMP_ExistsAddressLatchPolarity	Identifies whether the AddressLatchPolarity feature exists on the PMP module.
	PLIB_PMP_ExistsAddressLatchStrobePortControl	Identifies whether the AddressLatchStrobePortControl feature exists on the PMP module.
	PLIB_PMP_ExistsAddressPortPinControl	Identifies whether the AddressPortPinControl feature exists on the PMP module.
	PLIB_PMP_ExistsBufferOverFlow	Identifies whether the BufferOverFlow feature exists on the PMP module.
	PLIB_PMP_ExistsBufferRead	Identifies whether the BufferRead feature exists on the PMP module.
	PLIB_PMP_ExistsBufferType	Identifies whether the BufferType feature exists on the PMP module.
	PLIB_PMP_ExistsBufferUnderFlow	Identifies whether the BufferUnderFlow feature exists on the PMP module.
	PLIB_PMP_ExistsBufferWrite	Identifies whether the BufferWrite feature exists on the PMP module.
	PLIB_PMP_ExistsBusyStatus	Identifies whether the BusyStatus feature exists on the PMP module.
	PLIB_PMP_ExistsChipSelectEnable	Identifies whether the ChipSelectEnable feature exists on the PMP module.
	PLIB_PMP_ExistsChipSelectoperation	Identifies whether the ChipSelectoperation feature exists on the PMP module.
	PLIB_PMP_ExistsChipXPolarity	Identifies whether the ChipXPolarity feature exists on the PMP module.
	PLIB_PMP_ExistsCSXActiveStatus	Identifies whether the CSXActiveStatus feature exists on the PMP module.
	PLIB_PMP_ExistsDataHoldWaitStates	Identifies whether the DataHoldWaitStates feature exists on the PMP module.
	PLIB_PMP_ExistsDataSetUpWaitStates	Identifies whether the DataSetUpWaitStates feature exists on the PMP module.

	PLIB_PMP_ExistsDataStrobeWaitStates	Identifies whether the DataStrobeWaitStates feature exists on the PMP module.
	PLIB_PMP_ExistsDataTransferSize	Identifies whether the DataTransferSize feature exists on the PMP module.
	PLIB_PMP_ExistsDualBufferControl	Identifies whether the DualBufferControl feature exists on the PMP module.
	PLIB_PMP_ExistsDualModeMasterRXTX	Identifies whether the DualModeMasterRXTX feature exists on the PMP module.
	PLIB_PMP_ExistsDualModeReadAddressControl	Identifies whether the DualModeReadAddressControl feature exists on the PMP module.
	PLIB_PMP_ExistsDualModeWriteAddressControl	Identifies whether the DualModeWriteAddressControl feature exists on the PMP module.
	PLIB_PMP_ExistsEnableControl	Identifies whether the EnableControl feature exists on the PMP module.
	PLIB_PMP_ExistsIncrementMode	Identifies whether the IncrementMode feature exists on the PMP module.
	PLIB_PMP_ExistsInputBufferFull	Identifies whether the InputBufferFull feature exists on the PMP module.
	PLIB_PMP_ExistsInputBufferXStatus	Identifies whether the InputBufferXStatus feature exists on the PMP module.
	PLIB_PMP_ExistsInterruptMode	Identifies whether the InterruptMode feature exists on the PMP module.
	PLIB_PMP_ExistsMasterRXTX	Identifies whether the MasterRXTX feature exists on the PMP module.
	PLIB_PMP_ExistsMUXModeSelect	Identifies whether the MUXModeSelect feature exists on the PMP module.
	PLIB_PMP_ExistsOperationMode	Identifies whether the OperationMode feature exists on the PMP module.
	PLIB_PMP_ExistsOutPutBufferEmpty	Identifies whether the OutPutBufferEmpty feature exists on the PMP module.
	PLIB_PMP_ExistsOutputBufferXStatus	Identifies whether the OutputBufferXStatus feature exists on the PMP module.
	PLIB_PMP_ExistsReadChipSelectEnable	Identifies whether the ReadChipSelectEnable feature exists on the PMP module.
	PLIB_PMP_ExistsReadWritePolarity	Identifies whether the ReadWritePolarity feature exists on the PMP module.
	PLIB_PMP_ExistsReadWriteStrobePortControl	Identifies whether the ReadWriteStrobePortControl feature exists on the PMP module.
	PLIB_PMP_ExistsSlaveRX	Identifies whether the SlaveRX feature exists on the PMP module.
	PLIB_PMP_ExistsSlaveTX	Identifies whether the SlaveTX feature exists on the PMP module.
	PLIB_PMP_ExistsStartReadControl	Identifies whether the StartReadControl feature exists on the PMP module.
	PLIB_PMP_ExistsStopInIdleControl	Identifies whether the StopInIdleControl feature exists on the PMP module.
	PLIB_PMP_ExistsWriteChipSelectEnable	Identifies whether the WriteChipSelectEnable feature exists on the PMP module.
	PLIB_PMP_ExistsWriteEnablePolarity	Identifies whether the WriteEnablePolarity feature exists on the PMP module.
	PLIB_PMP_ExistsWriteEnablePortControl	Identifies whether the WriteEnablePortControl feature exists on the PMP module.
	PLIB_PMP_InputBuffersAreFull	Gets the state of the input buffers.
	PLIB_PMP_InputBufferTypeSelect	Selects the input buffer based on the input passed.
	PLIB_PMP_InputBufferXByteReceive	Data to be received in Byte mode.
	PLIB_PMP_InputBufferXIsFull	Gets the state of the identified input buffer.
	PLIB_PMP_InputOverflowClear	Clears a PMP Overflow error.
	PLIB_PMP_InputOverflowHasOccurred	Identifies if there was a receiver overflow error.
	PLIB_PMP_InterruptModeGet	Gets the current configured interrupt mode being used with the PMP module.
	PLIB_PMP_InterruptModeSelect	Configures the interrupt request mode being used with the PMP module.
	PLIB_PMP_IsDataReceived	Checks and returns if the data has been received in the specified buffer in Slave mode.
	PLIB_PMP_IsDataTransmitted	Checks and returns if the data has been transmitted from the specified buffer in Slave mode.
	PLIB_PMP_IsEnabled	Checks whether or not the PMP module is enabled.
	PLIB_PMP_MasterReceive	Receives the data in Master mode.
	PLIB_PMP_MasterSend	Sends the specified data in Master mode.
	PLIB_PMP_MultiplexModeGet	Gets the current multiplexing mode configured between the address and data of the PMP module.
	PLIB_PMP_MultiplexModeSelect	Configures the multiplexing between the address and data of the PMP module.
	PLIB_PMP_OperationModeGet	Gets the current operation mode of the PMP module.
	PLIB_PMP_OperationModeSelect	Configures the operation mode of the PMP module.
	PLIB_PMP_OutputBuffersAreEmpty	Gets the state of the output buffers.
	PLIB_PMP_OutputBufferXByteSend	Data to be transmitted in Byte mode.
	PLIB_PMP_OutputBufferXIsEmpty	Gets the state of the input buffer.
	PLIB_PMP_OutputUnderflowClear	Clears a PMP output underflow error.

	PLIB_PMP_OutputUnderflowHasOccurred	Identifies if there was an output buffer sent out with no data.
	PLIB_PMP_PortIsBusy	Identifies if the (Master mode) PMP port is busy.
	PLIB_PMP_ReadChipSelectXDisable	Configures the Read Chip Select.
	PLIB_PMP_ReadChipSelectXEnable	Configures the Read Chip Select.
	PLIB_PMP_ReadCycleIsStarted	Checks whether or not the read cycle on PMP bus is enabled.
	PLIB_PMP_ReadCycleStart	Starts a read cycle on the PMP bus.
	PLIB_PMP_ReadWriteStrobePolaritySelect	Sets the polarity of the read strobe.
	PLIB_PMP_ReadWriteStrobePortDisable	Disables the PMP module read strobe port.
	PLIB_PMP_ReadWriteStrobePortEnable	Enables the PMP module read strobe port.
	PLIB_PMP_SlaveReceive	Receives the data in Slave mode.
	PLIB_PMP_SlaveSend	Sends the specified data in Slave mode.
	PLIB_PMP_StopInIdleDisable	Enables the PMP module to continue operation in Idle mode.
	PLIB_PMP_StopInIdleEnable	Discontinues PMP module operation when the device enters Idle mode.
	PLIB_PMP_WaitStatesDataHoldSelect	Configures the data hold states (after data transfer) needed to be used with the PMP module.
	PLIB_PMP_WaitStatesDataSetUpSelect	Configures the data wait states (before the data transfer) needed to be used with the PMP module.
	PLIB_PMP_WaitStatesStrobeSelect	Configures the strobe wait states (during the data transfer) needed to be used with the PMP module.
	PLIB_PMP_WriteChipSelectXDisable	Configures the Write Chip Select.
	PLIB_PMP_WriteChipSelectXEnable	Configures the Write Chip Select.
	PLIB_PMP_WriteEnableStrobePolaritySelect	Sets the polarity of the write enable strobe.
	PLIB_PMP_WriteEnableStrobePortDisable	Disables the PMP module write strobe port.
	PLIB_PMP_WriteEnableStrobePortEnable	Enables the PMP module write strobe port.

Description

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the PMP Peripheral Library.

File Name

plib_pmp.h

Company

Microchip Technology Inc.

plib_pmp_help.h

Parallel Master Port (PMP) Peripheral Library Help Header.

Enumerations

	Name	Description
	PMP_ACK_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_ADDRESS_HOLD_LATCH_WAIT_STATES	PMP hold after address latch strobe wait states configuration.
	PMP_ADDRESS_LATCH	Address Latch Strobe configuration.
	PMP_ADDRESS_LATCH_WAIT_STATES	PMP address latch strobe wait states configuration.
	PMP_ADDRESS_PORT	Defines the different address lines that are available for configuration.
	PMP_ALTERNATE_MASTER_WAIT_STATES	PMP alternate master wait states.
	PMP_CHIP_SELECT	PMP Chip Select data type.
	PMP_CHIPSELECT_FUNCTION	Defines different functions available for the Chip Select lines multiplexed with address lines.
	PMP_DATA_HOLD_STATES	PMP Data Hold after strobe wait state.
	PMP_DATA_LENGTH	Possible data lengths handled by the PMP module.
	PMP_DATA_SIZE	PMP data size.
	PMP_DATA_WAIT_STATES	PMP data setup time configuration.
	PMP_INCREMENT_MODE	PMP address incrementing configuration.
	PMP_INPUT_BUFFER_TYPE	PMP Input Buffers type.
	PMP_INTERRUPT_MODE	PMP interrupt request mode data type.

	PMP_MASTER_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_MODULE_ID	Possible instances of the PMP module.
	PMP_MUX_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_OPERATION_MODE	Defines the different mode configurations in which the PMP can be enabled.
	PMP_PMBE_PORT	Defines the available Byte Enable ports.
	PMP_POLARITY_LEVEL	Possible polarity levels for the PMP pins.
	PMP_STROBE_WAIT_STATES	PMP strobe signal wait time configuration.

Description

PMP/EPMP Peripheral Library Help Header

This header file contains the data types and constants that make up the interface to the PMP Peripheral Library.

File Name

plib_pmp_help.h

Company

Microchip Technology Inc.

Prefetch Cache Peripheral Library

This section describes the Prefetch Cache Peripheral Library.

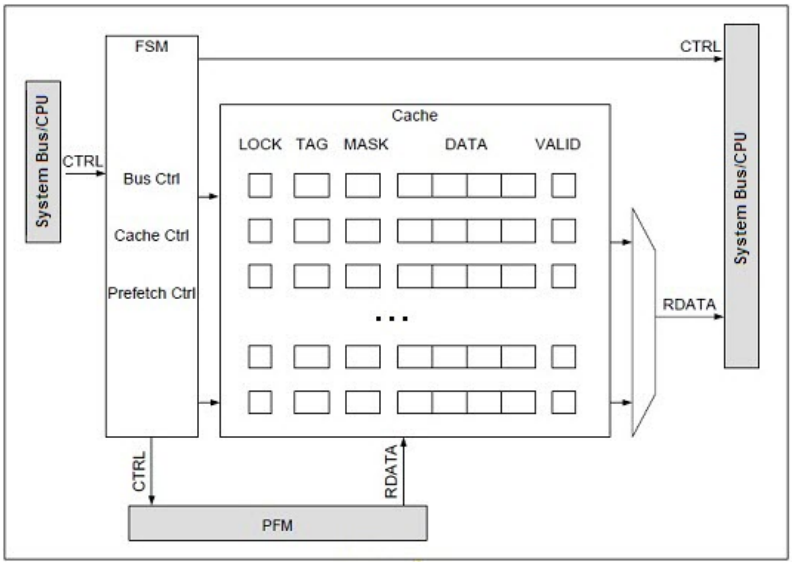
Introduction

This library provides a low-level abstraction of the Prefetch Cache module on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Prefetch Cache module increases system performance for most applications by:

- Predictive prefetching of instructions ahead of the current program counter, hiding the access time of the Flash memory
- Instruction and data caching



The program Flash memory (PFM) contains a Prefetch module and a number of cache lines, each containing four words. Some of the cache lines may be allocated to data or peripherals. Each cache line has an associated tag. The tag contains the address of the data in the cache line, and a number of status bits. The number of cache lines and number and type of status bits may vary between devices. Refer to your specific device data sheet for details.

Using the Library

This topic describes the basic architecture of the Prefetch Cache Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_pcache.h`

The interface to the Prefetch Cache Peripheral Library is defined in the `plib_pcache.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Prefetch Cache Peripheral Library must include `peripheral.h`.

Library File:

The Prefetch Cache Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

Hardware Abstraction Model

This library provides a low-level abstraction of the Prefetch Cache module on Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Model

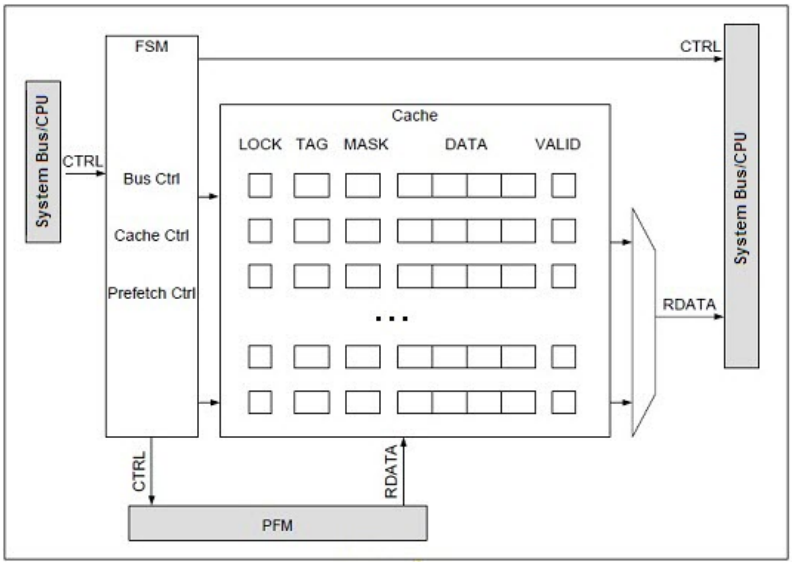
The Prefetch Cache module is a performance enhancing module included in some microcontrollers. When running at high clock rates, wait states must be inserted into program Flash memory (PFM) read transactions to meet the access time of the PFM. Wait states can be hidden to the core by prefetching and storing instructions in a temporary holding area that the CPU can access quickly.

There are two main functions that the Prefetch Cache module performs: Caching instructions when they are accessed, and prefetching instructions from the PFM before they are needed.

The cache holds a subset of the cacheable memory in temporary holding spaces known as cache lines. Each cache line has a tag describing what it is currently holding, and the address where it is mapped. Normally, the cache lines just hold a copy of what is currently in memory to make data available to the CPU without wait states.

CPU-requested data may or may not be in the cache. A cache-miss occurs if the CPU requests cacheable data that is not in the cache. In this case, a read is performed to the PFM at the correct address, the data is supplied to the cache and to the CPU. A cache-hit occurs if the cache contains the data that the CPU requests. In the case of a cache-hit, data is supplied to the CPU without wait states.

The second main function of the Prefetch Cache module is to prefetch CPU instructions. The module calculates the address of the next cache line and performs a read of the PFM to get the next cache line. This line is placed into a prefetch cache buffer in anticipation of executing straight-line code.



Cache Control

This library provides a set of functions to control the behavior of the overall Cache Module. Those functions include:

- Setting or reading the number of clock wait cycles
- Setting or reading the number of cache lines allocated to data
- Controlling which cache lines are invalidated on a PFM write cycle

Cache Line Operations

This library provides a set of functions to read, write and control the behavior of individual Cache Lines. Operations on individual Cache Lines are performed using a two-step process:

1. Select the individual cache line with PLIB_PCACHE_CacheLineSelect.
2. Perform the required cache line operation using any of the routines in the Cache Line Operations section. The operations available are:

- Set the cache line to data or instruction type

- Mark the cache line as valid or invalid
- Lock or unlock the cache line
- Read or write the Flash type (boot or program) associated with the cache line
- Read or write the address associated with the cache line
- Read or write the mask field to force a match between the cache line and multiple physical addresses
- Read or write any individual word in the cache line data array

Once the operation is performed, the cache line remains selected until PLIB_PCACHE_CacheLineDeselect is performed. Only one cache line can be selected at a time.

Cache Status

The cache maintains a count of hits and misses for profiling purposes. The cache hit counter increments each time the processor issues an instruction fetch or load that hits the prefetch cache from a cacheable region. Similarly, the cache miss counter increments each time the processor issues an instruction fetch or load that misses the prefetch cache from a cacheable region. Accesses to non-cacheable regions do not affect the counters. These values can be read or written using the Cache Status routines.

The cache uses a Least Recently Used (LRU) algorithm to replace cache lines on a miss. The specific algorithm may vary between devices. In some devices, the state of the LRU algorithm encoding bits can be read with PLIB_PCACHE_LeastRecentlyUsedStateRead.

Prefetch Control

Predictive prefetch is controlled by the Prefetch Control routines. Devices with no L1 CPU cache can prefetch from cacheable or non-cacheable regions or both. Devices with L1 CPU cache can prefetch from any address, CPU instructions or CPU instructions and data. Predictive prefetch can be disabled for either type of device.

Prefetch Status

The Prefetch Cache module maintains a count of prefetch aborts for profiling purposes. The prefetch abort counter is incremented each time a prefetch is aborted due to a non-sequential instruction fetch, load or store. This value can be read or written using the Prefetch Status routines.

Flash ECC

On some devices, the Flash ECC module has the ability to detect single and double bit errors on PFM reads. When a single bit error is detected, the error is corrected and the Single Error Correction (SEC) counter is decremented. When the SEC counter reaches zero, an interrupt is generated to the CPU. The user has the ability to enable or disable the SEC interrupt. It is disabled by default. The user may set the SEC counter value at any time. The maximum counter value may be device-specific and is defined for each processor by the PLIB_PCACHE_MAX_SEC_COUNT constant.

When a double bit error is detected, the device will set the DED status to true and generate a bus exception to the initiator.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Prefetch Cache module.

Library Interface Section	Description
Cache Control Functions	Provides Cache Control interface routines.
Cache Line Functions	Provides Cache Line interface routines.
Cache Status Functions	Provides Cache Status interface routines.
Prefetch Control Functions	Provides Prefetch Control interface routines.
Prefetch Status Functions	Provides Prefetch Status interface routines.
Flash ECC Functions	Provides Flash ECC interface routines.
Feature Existence Functions	Provides interface routines that determine whether or not certain features are available.

How the Library Works

Provides information on how the library works.

Description

This library can be used to optimize the performance of the processor while executing out of cacheable program Flash memory. This is done by implementing the following:

- Instruction caching
- Data caching
- Cache line control
- Predictive prefetching
- ECC detection

Arrays

The cache consists of two arrays: tag and data. The data array contains a number of lines of program instructions or data words. The word size is

the same as the processor data width.

The tag array contains a number of corresponding fields, each containing:

- Mask – address mask value
- Tag – tag address to match against
- Valid – indicates whether the data in the cache line is valid or not
- Lock – indicates the line is locked to hold its contents
- Type – instruction, data or peripheral

Note: Not all Tag fields are available on all devices. The number of cache lines may vary between devices. Refer to the specific device data sheet for details.

Example Cache Line

Tag Array					Data Array			
Mask	Tag	Valid	Lock	Type	Word 3	Word 2	Word 1	Word 0

Operations

This library allows the programmer to perform various operations on the cache tag and data arrays:

- Enable/Disable
- Set the Wait state
- Read, write, lock, invalidate

Note: Not all operations are available on all devices. Refer to the specific device data sheet for details.

Cache Control Operations

Cache Wait States

When operating at high clock rates, the Prefetch Cache module must insert wait states into Program Flash Memory (PFM) read operations to meet PFM access time requirements. One Wait state is equivalent to one clock cycle delay. At reset, the Prefetch Cache Module is set to the maximum number of wait states for the device. The PFM read time can be optimized for the device by selecting the minimum number of wait states for the clock rate used. The number of wait states for a given clock frequency may vary between devices.

Note: Refer to the specific device data sheet for details on how to calculate the optimum number of wait states for your device.

Example: Setting the Number of PFM wait states based on the System Clock

This example assumes one wait state for every 20 MHz of system clock.

```
if (sysclk < 20000000)
{
 PLIB_PCACHE_WaitStateSet(0);
}
else if (sysclk < 40000000)
{
 PLIB_PCACHE_WaitStateSet(1);
}
...
```

Data Caching

Some Prefetch Cache modules allow the user to allocate cache lines to data. This is to provide improved CPU access to constant data stored in PFM. By default, data caching is disabled. The Prefetch Cache Peripheral Library allows the user to set the number of cache lines allocated to data. The number of lines available may vary among devices. The available settings are provided by the device-dependent PLIB_PCACHE_DATA_ENABLE enumeration.

Example: Allocating two Cache Lines to Data

```
PLIB_PCACHE_DataCacheEnableSet(PLIB_PCACHE_DATA_2LINE);
```

Cache Line Invalidating on PFM Write

It is not possible to execute out of cache while programming the PFM. The PFM controller stalls the cache during the programming sequence. Therefore, user code that initiates a programming sequence should not be located in a cacheable address region.

During program operation, the Prefetch Cache is flushed by invalidating some or all of the cache lines. To invalidate all of the cache lines on a PFM write operation, the function PLIB_PCACHE_InvalidateOnPFMProgramAll should be called prior to the PFM write. This will invalidate and unlock every cache line during the write operation. The cache tags and masks are also cleared for all lines.

If PLIB_PCACHE_InvalidateOnPFMProgramUnlocked is called before the PFM write, only lines that are not locked will be forced invalid.

By default, only unlocked cache lines are invalidated during a PFM write.

Cache Line Operations

Performing a cache line operation is a two-step process. Before calling a Cache Line operation function, the cache line must be selected and write-enabled. Once selected and write-enabled, any number of cache line operations may be performed on the selected line until the line is cleared or a different line is selected and write-enabled.

Cache lines are selected and write-enabled using PLIB_PCACHE_CacheLineSelect. Cache lines are write-disabled using PLIB_PCACHE_CacheLineDeselect.

The following example shows operations on two different cache lines. The code comments describe each operation.

Example: Cache Line Operations

```
uint32_t cacheLine[4] = {0x01234567, 0x12345678, 0x23456789, 0x34567890};

//Select Cache Line 4
PLIB_PCACHE_CacheLineSelect(4);

//Invalidate Cache Line 4
PLIB_PCACHE_CacheLineInvalid();

//Select Cache Line 6
PLIB_PCACHE_CacheLineSelect(6);

//Set Cache Line 6 to Data Type
PLIB_PCACHE_CacheLineData();

//Write Data to Cache Line 6
for (i=0; i<PLIB_PCACHE_NUM_WORDS_PER_LINE; i++)
{
    PLIB_PCACHE_WordWrite(i, cacheLine[i]);
}

//Lock Cache Line 6
PLIB_PCACHE_CacheLineLock();

//Disable Writes to Cache Line 6
PLIB_PCACHE_CacheLineDeselect();
```

Cache Status Operations

Cache Statistics Counters

It is often useful to gather statistics related to cache utilization for performance or profiling reasons. Some devices with a Prefetch Cache module maintain cache hit and miss counters for this purpose. This library provides routines to read and write the cache hit and miss counters.

The cache hit counter increments each time the processor issues an instruction fetch or load that hits the prefetch cache from a cacheable region. Non-cacheable accesses do not modify this value. This value can be read with PLIB_PCACHE_CacheHitRead.

Similarly, the cache miss counter increments each time the processor issues an instruction fetch or load that misses the prefetch cache from a cacheable region. This value can be read with PLIB_PCACHE_CacheMissRead.

Cache Replacement State

The Prefetch Cache module uses a pseudo-Least Recently Used (LRU) algorithm to replace cache lines. The specific algorithm may vary among devices.

Note: Refer to the specific device data sheet for details on a specific device.

Some devices maintain the state of the LRU algorithm for use by software. The state of the LRU algorithm can be read with PLIB_PCACHE_LeastRecentlyUsedStateRead.

Prefetch Control Operations

The Prefetch Cache module is configured differently for different CPU configurations. For devices with a CPU L1 cache, the Prefetch Cache module holds one line of CPU instructions, one line of CPU data, and two lines of peripheral data. For devices without a CPU L1 cache, the Prefetch Cache module holds up to 16 lines of instructions and up to four lines of data. Control of either configuration is done using the PLIB_PCACHE_PrefetchEnableSet function.

Note: Refer to the specific device data sheet for the Prefetch Cache module configuration for a specific device.

For the CPU L1 configuration, the Prefetch Cache module can be configured to enable any address, CPU instructions and data, or CPU instructions only. For devices without a CPU L1 cache, predictive prefetch can be enabled for cacheable regions, uncacheable regions, or both. In either configuration, predictive prefetch may be disabled entirely.

The PLIB_PCACHE_PrefetchEnableSet and PLIB_PCACHE_PrefetchEnableGet functions take or return the enumerated value PLIB_PCACHE_PREFETCH_ENABLE. This enumeration will be defined differently for different CPU configurations.

Example: Enable Prefetch for Non-Cacheable Regions Only (no L1 configuration)

```
PLIB_PCACHE_PrefetchEnableSet(PLIB_PCACHE_PREFETCH_ENABLE_NONCACHED_REGIONS);
```

Example: Enable Prefetch for CPU Instructions and CPU Data (L1 configuration)

```
PLIB_PCACHE_PrefetchEnableSet(PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST_DATA);
```

Prefetch Status Operations

Prefetch Abort Statistics Counter

Much like the cache hit/miss counters, the Prefetch Abort module maintains a prefetch abort counter. This counter is incremented each time an automatic prefetch is aborted due to a non-sequential instruction fetch, load or store. This value can be read using the PLIB_PCACHE_PrefetchAbortRead function.

Flash ECC Operations

The Prefetch Cache module may handle and report information about two error types: ECC Double-bit Error Detected (DED) and ECC Single-bit Error Corrected (SEC).

A read from the Flash memory that results in a program Flash memory ECC DED causes the Prefetch Cache module to return a bus exception to the initiator. An ECC DED will also set the DED status bit. The state of the DED status bit may be read using the PLIB_PCACHE_FlashDEDStatusGet function.

A single-bit error read from the Flash memory is not a critical error, and as such is reported as an interrupt. The user has the option to enable or disable the SEC interrupt, and to set the number of SEC events that must occur before an SEC interrupt is generated. An SEC interrupt is generated when all of the following conditions are met:

- The Flash SEC counter is zero
- The Flash SEC interrupt is enabled
- An SEC event occurs

The Flash SEC counter is decremented once each time an SEC event occurs, and holds at zero. When a Flash SEC interrupt is triggered, the Flash SEC status bit is set. Once an SEC interrupt occurs, the Flash SEC status must be cleared by software and the Flash SEC counter must be reloaded by software.

Example: Set Flash SEC to Interrupt after three SEC events

```
PLIB_PCACHE_FlashSECStatusClear();
PLIB_PCACHE_FlashSECCountSet(3);
PLIB_PCACHE_FlashSECIntEnable();
```

It is possible to generate an SEC interrupt for testing purposes by enabling the Flash SEC interrupt and setting the Flash SEC status while the Flash SEC counter is zero.

Example: Generate an SEC Interrupt

```
PLIB_PCACHE_FlashSECStatusClear();
PLIB_PCACHE_FlashSECCountSet(0);
PLIB_PCACHE_FlashSECIntEnable();
PLIB_PCACHE_FlashSECStatusSet();
```

Configuring the Library

The library is configured for the supported Prefetch Cache module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Cache Control Functions

	Name	Description
	PLIB_PCACHE_DataCacheEnableGet	Gets the data cache enable bits.
	PLIB_PCACHE_DataCacheEnableSet	Sets the data cache enable bits.
	PLIB_PCACHE_InvalidateOnPFMProgramAll	Sets Prefetch Cache to invalidate all data and instruction lines on a PFM program cycle.
	PLIB_PCACHE_InvalidateOnPFMProgramUnlocked	Sets Prefetch Cache to invalidate all unlocked data and instruction lines on a PFM program cycle.
	PLIB_PCACHE_WaitStateGet	Gets the Prefetch Cache access time.

	PLIB_PCACHE_WaitStateSet	Sets the Prefetch Cache access time.

b) Cache Line Functions

	Name	Description
	PLIB_PCACHE_CacheLineAddrGet	Gets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineAddrSet	Sets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineData	Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to data type.
	PLIB_PCACHE_CacheLineDeselect	Disables write access to the cache line specified by cache_line.
	PLIB_PCACHE_CacheLineFlashTypeBoot	Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Boot Flash.
	PLIB_PCACHE_CacheLineFlashTypeInst	Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Instruction Flash.
	PLIB_PCACHE_CacheLineFlashTypeIsInst	Returns true if the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect shows the line is residing in Instruction Flash.
	PLIB_PCACHE_CacheLineInst	Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to instruction type.
	PLIB_PCACHE_CacheLineInvalid	Invalidates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineIsInst	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is set to instruction type.
	PLIB_PCACHE_CacheLineIsLocked	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is locked.
	PLIB_PCACHE_CacheLineIsValid	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is valid.
	PLIB_PCACHE_CacheLineLock	Locks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineMaskGet	Returns the current state of the cache tag mask for the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineMaskSet	Sets the cache tag mask to force a match on set bits on the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineSelect	Enables write access to the cache line specified by cache_line.
	PLIB_PCACHE_CacheLineUnlock	Unlocks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineValid	Validates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_WordRead	Reads the word specified by word from cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_WordWrite	Writes the word (specified by word parameter) with data (specified by data parameter) to cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

c) Cache Status Functions

	Name	Description
	PLIB_PCACHE_CacheHitRead	Reads the number of cache hits.
	PLIB_PCACHE_CacheHitWrite	Sets the number of cache hits.
	PLIB_PCACHE_CacheMissRead	Reads the number of cache misses.
	PLIB_PCACHE_CacheMissWrite	Sets the number of cache misses.
	PLIB_PCACHE_LeastRecentlyUsedStateRead	Reads the state of the cache Least Recently Used (LRU) encoding bits.

d) Prefetch Control Functions

	Name	Description
	PLIB_PCACHE_PrefetchEnableGet	Gets the predictive Prefetch state for the Prefetch Cache module.
	PLIB_PCACHE_PrefetchEnableSet	Sets the predictive Prefetch state for the Prefetch Cache module.

e) Prefetch Status Functions

	Name	Description
	PLIB_PCACHE_PrefetchAbortRead	Reads the number of prefetch aborts.
	PLIB_PCACHE_PrefetchAbortWrite	Sets the number of prefetch aborts.

f) Flash ECC Functions

	Name	Description
	PLIB_PCACHE_FlashDEDStatusClear	Clears a bus exception caused by a double-bit error.
	PLIB_PCACHE_FlashDEDStatusGet	Determines if a bus exception was caused by a double-bit error.
	PLIB_PCACHE_FlashSECCountGet	Returns the current value of the Flash SEC counter.
	PLIB_PCACHE_FlashSECCountSet	Sets the number of single-bit error corrections that must occur before the SEC status is set to true.
	PLIB_PCACHE_FlashSECIntDisable	Disables an interrupt on SEC.
	PLIB_PCACHE_FlashSECIntEnable	Enables an interrupt on SEC.
	PLIB_PCACHE_FlashSECStatusClear	Sets the single-bit error corrected status to false.
	PLIB_PCACHE_FlashSECStatusGet	Determines if a SEC event triggered a Prefetch Cache interrupt.
	PLIB_PCACHE_FlashSECStatusSet	Sets the single-bit error corrected status to true.

g) Feature Existence Functions

	Name	Description
	PLIB_PCACHE_ExistsCacheHit	Identifies that the CacheHit feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLine	Identifies that the CacheLineSelect feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineAddr	Identifies that the CacheLineAddr feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineFlashType	Identifies that the CacheLineFlashType feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineLock	Identifies that the CacheLineLock feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineMask	Identifies that the CacheLineMask feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineType	Identifies that the CacheLineType feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineValid	Identifies that the CacheLineValid feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheMiss	Identifies that the CacheMiss feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsDataCacheEnable	Identifies that the DataCacheEnable feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashDEDStatus	Identifies that the FlashDEDStatus feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECCount	Identifies that the FlashSECCount feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECInt	Identifies that the FlashSECInt feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECStatus	Identifies that the FlashSECStatus feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsInvalidateOnPFMProgram	Identifies that the InvalidateOnPFMProgram feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsLeastRecentlyUsedState	Identifies that the LeastRecentlyUsedState feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsPrefetchAbort	Identifies that the PrefetchAbort feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsPrefetchEnable	Identifies that the PrefetchEnable feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsWaitState	Identifies that the WaitState feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsWord	Identifies that the Word feature exists on the Prefetch Cache module.

h) Data Types and Constants

	Name	Description
	PLIB_PCACHE_MAX_SEC_COUNT	Defines the maximum value for single bit error counter.
	PLIB_PCACHE_NUM_LINES	Defines the number of available Prefetch Cache Lines.
	PLIB_PCACHE_NUM_WORDS_PER_LINE	Defines the number of Words per Prefetch Cache Line.
	PCACHE_MODULE_ID	Lists the possible Module IDs for the Prefetch Cache.
	PLIB_PCACHE_DATA_ENABLE	Lists the possible predictive prefetch states for the Prefetch Cache.
	PLIB_PCACHE_PREFETCH_ENABLE	Lists the possible predictive prefetch states for the Prefetch Cache.

Description

This section describes the Application Programming Interface (API) functions of the Prefetch Cache Peripheral Library.

Refer to each section for a detailed description.

a) Cache Control Functions

PLIB_PCACHE_DataCacheEnableGet Function

Gets the data cache enable bits.

File

plib_pcache.h

C

```
PLIB_PCACHE_DATA_ENABLE PLIB_PCACHE_DataCacheEnableGet(PCACHE_MODULE_ID index);
```

Returns

PLIB_PCACHE_DATA_ENABLE

Description

This function gets the data cache enable bits.

Remarks

At reset, data caching is disabled, and this function will return PLIB_PCACHE_DATA_DISABLE.

This function implements an operation of the DataCacheEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsDataCacheEnable in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_DATA_ENABLE dcache_en;
dcache_en = PLIB_PCACHE_DataCacheEnableGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

PLIB_PCACHE_DATA_ENABLE PLIB_PCACHE_DataCacheEnableGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_DataCacheEnableSet Function

Sets the data cache enable bits.

File

plib_pcache.h

C

```
void PLIB_PCACHE_DataCacheEnableSet(PCACHE_MODULE_ID index, PLIB_PCACHE_DATA_ENABLE dcache_en);
```

Returns

None.

Description

This function sets the data cache enable bits.

Remarks

Data caching is disabled at reset.

This function implements an operation of the DataCacheEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsDataCacheEnable in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_DataCacheEnableSet(PCACHE_ID_0, PLIB_PCACHE_DATA_4LINE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dcache_en	PLIB_PCACHE_DATA_ENABLE

Function

void PLIB_PCACHE_DataCacheEnableSet(PCACHE_MODULE_ID index,
PLIB_PCACHE_DATA_ENABLE dcache_en)

PLIB_PCACHE_InvalidateOnPFMProgramAll Function

Sets Prefetch Cache to invalidate all data and instruction lines on a PFM program cycle.

File

plib_pcache.h

C

```
void PLIB_PCACHE_InvalidateOnPFMProgramAll(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the Prefetch Cache to invalidate all data and instruction lines on a PFM program cycle.

Remarks

This function implements an operation of the InvalidateOnPFMProgram feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsInvalidateOnPFMProgram in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_InvalidateOnPFMProgramAll(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_InvalidateOnPFMProgramAll(PCACHE_MODULE_ID index)

PLIB_PCACHE_InvalidateOnPFMProgramUnlocked Function

Sets Prefetch Cache to invalidate all unlocked data and instruction lines on a PFM program cycle.

File

plib_pcache.h

C

```
void PLIB_PCACHE_InvalidateOnPFMProgramUnlocked(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the Prefetch Cache to invalidate all unlocked data and instruction lines on a PFM program cycle.

Remarks

This is the default state at reset.

This function implements an operation of the InvalidateOnPFMProgram feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsInvalidateOnPFMProgram in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_InvalidateOnPFMProgramUnlocked(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_InvalidateOnPFMProgramUnlocked(PCACHE_MODULE_ID index)

PLIB_PCACHE_WaitStateGet Function

Gets the Prefetch Cache access time.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_WaitStateGet(PCACHE_MODULE_ID index);
```

Returns

Number of clock cycles.

Description

This function gets the Prefetch Cache access time in terms of SYSCLK wait. states.

Remarks

At reset, this value is set to seven wait states.

This function implements an operation of the WaitState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWaitState in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t wait;
wait = PLIB_PCACHE_WaitStateGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

uint32_t PLIB_PCACHE_WaitStateGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_WaitStateSet Function

Sets the Prefetch Cache access time.

File

plib_pcache.h

C

```
void PLIB_PCACHE_WaitStateSet(PCACHE_MODULE_ID index, uint32_t clocks);
```

Returns

None.

Description

This function sets the Prefetch Cache access time in terms of SYSCLK wait. states.

Remarks

At reset, this value is set to seven wait states.

This function implements an operation of the WaitState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWaitState in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_WaitStateSet(PCACHE_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
clocks	Identifies the wait state in terms of number of SYSCLK cycles

Function

void PLIB_PCACHE_WaitStateSet(PCACHE_MODULE_ID index, uint32_t clocks)

b) Cache Line Functions

PLIB_PCACHE_CacheLineAddrGet Function

Gets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_CacheLineAddrGet(PCACHE_MODULE_ID index);
```

Returns

The address in the cache line TAG register.

Description

This function gets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineAddr feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineAddr in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t addr;
addr = PLIB_PCACHE_CacheLineAddrGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

uint32_t PLIB_PCACHE_CacheLineAddrGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineAddrSet Function

Sets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineAddrSet(PCACHE_MODULE_ID index, uint32_t addr);
```

Returns

None.

Description

This function Sets the TAG address in the cache line previously write- enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineAddr feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineAddr in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t addr = 0x1FC01234;
PLIB_PCACHE_CacheLineAddrSet(PCACHE_ID_0, addr);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
addr	Address to program into the cache line TAG register

Function

void PLIB_PCACHE_CacheLineAddrSet(PCACHE_MODULE_ID index, uint32_t addr)

PLIB_PCACHE_CacheLineData Function

Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to data type.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineData(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the cache line previously enabled by PLIB_PCACHE_CacheLineSelect to data type.

Remarks

This function implements an operation of the CacheLineType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineData(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineData(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineDeselect Function

Disables write access to the cache line specified by cache_line.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineDeselect(PCACHE_MODULE_ID index, uint32_t cache_line);
```

Returns

None.

Description

This function disables write access to the cache line specified by cache_line.

Remarks

This function implements an operation of the CacheLineSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineSelect in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineDeselect(PCACHE_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cache_line	Specifies the cache line to disable write access

Function

void PLIB_PCACHE_CacheLineDeselect(PCACHE_MODULE_ID index,

uint32_t cache_line)

PLIB_PCACHE_CacheLineFlashTypeBoot Function

Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Boot Flash.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineFlashTypeBoot(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Boot Flash.

Remarks

This function implements an operation of the CacheLineFlashType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineFlashType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineFlashTypeBoot(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineFlashTypeBoot(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineFlashTypeInst Function

Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Instruction Flash.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineFlashTypeInst(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Instruction Flash.

Remarks

This function implements an operation of the CacheLineFlashType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineFlashType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineFlashTypeInst(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineFlashTypeInst(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineFlashTypeIsInst Function

Returns true if the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect shows the line is residing in Instruction Flash.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_CacheLineFlashTypeIsInst(PCACHE_MODULE_ID index);
```

Returns

- true - The cache line resides in Instruction Flash
- false - The cache line resides in Boot Flash

Description

This function returns true if the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect shows the line is residing in Instruction Flash.

Remarks

This function implements an operation of the CacheLineFlashType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineFlashType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool LineFlashType;
LineFlashType = PLIB_PCACHE_CacheLineFlashTypeIsInst(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_PCACHE_CacheLineFlashTypeIsInst(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineInst Function

Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to instruction type.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineInst(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to instruction type.

Remarks

This function implements an operation of the CacheLineType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineInst(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineInst(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineInvalid Function

Invalidates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineInvalid(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function invalidates the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineValid feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineValid in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineInvalid(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineInvalid(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineIsInst Function

Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is set to instruction type.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_CacheLineIsInst(PCACHE_MODULE_ID index);
```

Returns

- true - The cache line is instruction
- false - The cache line is data

Description

This function returns true if the cache line previously enabled by PLIB_PCACHE_CacheLineSelect is set to instruction type.

Remarks

This function implements an operation of the CacheLineType feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineType in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool CacheLineType;
CacheLineType = PLIB_PCACHE_CacheLineIsInst(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

bool PLIB_PCACHE_CacheLineIsInst(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineIsLocked Function

Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is locked.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_CacheLineIsLocked(PCACHE_MODULE_ID index);
```

Returns

- true - The cache line is locked
- false - The cache line is unlocked

Description

This function returns true if the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is locked.

Remarks

This function implements an operation of the CacheLineLock feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineLock in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool CacheLineLocked;
CacheLineLocked = PLIB_PCACHE_CacheLineIsLocked(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_PCACHE_CacheLineIsLocked(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineIsValid Function

Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is valid.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_CacheLineIsValid(PCACHE_MODULE_ID index);
```

Returns

- true - The cache line is valid
- false - The cache line is invalid

Description

This function returns true if the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is valid.

Remarks

This function implements an operation of the CacheLineValid feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineValid in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool CacheLineValid;
CacheLineValid = PLIB_PCACHE_CacheLineIsValid(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_PCACHE_CacheLineIsValid(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineLock Function

Locks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineLock(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function locks the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineLock feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineLock in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineLock(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineLock(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineMaskGet Function

Returns the current state of the cache tag mask for the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_CacheLineMaskGet(PCACHE_MODULE_ID index);
```

Returns

- Address mask - Bits set to '1' will force a match (valid bits: <15:5>).

Description

This function returns the current state of the cache tag mask for the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineMask feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineMask in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t mask;
mask = PLIB_PCACHE_CacheLineMaskGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

uint32_t PLIB_PCACHE_CacheLineMaskGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineMaskSet Function

Sets the cache tag mask to force a match on set bits on the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineMaskSet(PCACHE_MODULE_ID index, uint32_t mask);
```

Returns

None.

Description

This function sets the cache tag mask to force a match on set bits on the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineMask feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineMask in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t mask = 0x0C0;
PLIB_PCACHE_CacheLineMaskSet(PCACHE_ID_0, mask);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mask	Address mask (bits set to '1' will force a match; valid bits: <15:5>)

Function

void PLIB_PCACHE_CacheLineMaskSet(PCACHE_MODULE_ID index, uint32_t mask)

PLIB_PCACHE_CacheLineSelect Function

Enables write access to the cache line specified by cache_line.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineSelect(PCACHE_MODULE_ID index, uint32_t cache_line);
```

Returns

None.

Description

This function enables write access to the cache line specified by cache_line.

Remarks

At reset, all cache lines are read-only.

This function implements an operation of the CacheLineSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineSelect in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineSelect(PCACHE_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cache_line	Specifies the cache line to enable write access

Function

void PLIB_PCACHE_CacheLineSelect(PCACHE_MODULE_ID index,

uint32_t cache_line)

PLIB_PCACHE_CacheLineUnlock Function

Unlocks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineUnlock(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function unlocks the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineLock feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineLock in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineUnlock(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineUnlock(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheLineValid Function

Validates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheLineValid(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function validates the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the CacheLineValid feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheLineValid in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheLineValid(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_CacheLineValid(PCACHE_MODULE_ID index)

PLIB_PCACHE_WordRead Function

Reads the word specified by word from cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_WordRead(PCACHE_MODULE_ID index, uint32_t word);
```

Returns

A 32-bit unsigned word from the cache data array.

Description

This function reads the word specified by word from the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the Word feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWord in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t word;
word = PLIB_PCACHE_WordRead(PCACHE_ID_0, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read
word	Location of the word in the data array to be read

Function

uint32_t PLIB_PCACHE_WordRead(PCACHE_MODULE_ID index, uint32_t word)

PLIB_PCACHE_WordWrite Function

Writes the word (specified by word parameter) with data (specified by data parameter) to cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

File

plib_pcache.h

C

```
void PLIB_PCACHE_WordWrite(PCACHE_MODULE_ID index, uint32_t word, uint32_t data);
```

Returns

None.

Description

This function writes the word (specified by word parameter) with data (specified by data parameter) to the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Remarks

This function implements an operation of the Word feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWord in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t word = 0;
uint32_t data = 0xDEADBEEF;
PLIB_PCACHE_WordWrite(PCACHE_ID_0, word, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be written.
word	Location of the word in the data array to be written
data	32-bit unsigned word to write to cache data array

Function

void PLIB_PCACHE_WordWrite(PCACHE_MODULE_ID index, uint32_t word,

uint32_t data)

c) Cache Status Functions

PLIB_PCACHE_CacheHitRead Function

Reads the number of cache hits.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_CacheHitRead(PCACHE_MODULE_ID index);
```

Returns

The number of cache hits.

Description

This function reads and returns the number of cache hits.

Remarks

This function implements an operation of the CacheHit feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheHit in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t cache_hits;
cache_hits = PLIB_PCACHE_CacheHitRead(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

uint32_t PLIB_PCACHE_CacheHitRead(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheHitWrite Function

Sets the number of cache hits.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheHitWrite(PCACHE_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function sets the number of cache hits.

Remarks

This function implements an operation of the CacheHit feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheHit in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheHitWrite(PCACHE_ID_0, 0xF00);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be written
data	Number of cache hits to write

Function

void PLIB_PCACHE_CacheHitWrite(PCACHE_MODULE_ID index, uint32_t data)

PLIB_PCACHE_CacheMissRead Function

Reads the number of cache misses.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_CacheMissRead(PCACHE_MODULE_ID index);
```

Returns

The number of cache misses.

Description

This function reads and returns the number of cache misses.

Remarks

This function implements an operation of the CacheMiss feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheMiss in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t cache_misses;
cache_misses = PLIB_PCACHE_CacheMissRead(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read.

Function

uint32_t PLIB_PCACHE_CacheMissRead(PCACHE_MODULE_ID index)

PLIB_PCACHE_CacheMissWrite Function

Sets the number of cache misses.

File

plib_pcache.h

C

```
void PLIB_PCACHE_CacheMissWrite(PCACHE_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function sets the number of cache misses.

Remarks

This function implements an operation of the CacheMiss feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCacheMiss in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_CacheMissWrite(PCACHE_ID_0, 0xF00);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be written
data	Number of cache misses to write

Function

void PLIB_PCACHE_CacheMissWrite(PCACHE_MODULE_ID index, uint32_t data)

PLIB_PCACHE_LeastRecentlyUsedStateRead Function

Reads the state of the cache Least Recently Used (LRU) encoding bits.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_LeastRecentlyUsedStateRead(PCACHE_MODULE_ID index);
```

Returns

The state of the LRU encoding bits represented as a 25-bit unsigned integer.

Description

This function reads the state of the cache LRU encoding bits.

Remarks

This function implements an operation of the LeastRecentlyUsedState feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsLeastRecentlyUsedState in your application to to automatically

determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t lru_state;
lru_state = PLIB_PCACHE_LeastRecentlyUsedStateRead(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read.

Function

uint32_t PLIB_PCACHE_LeastRecentlyUsedStateRead(PCACHE_MODULE_ID index)

d) Prefetch Control Functions

PLIB_PCACHE_PrefetchEnableGet Function

Gets the predictive Prefetch state for the Prefetch Cache module.

File

plib_pcache.h

C

```
PLIB_PCACHE_PREFETCH_ENABLE PLIB_PCACHE_PrefetchEnableGet(PCACHE_MODULE_ID index);
```

Returns

PLIB_PCACHE_PREFETCH_ENABLE

Description

This function gets the predictive Prefetch state for the Prefetch Cache module.

Remarks

At reset, the Prefetch Cache module is disabled and this function will return PLIB_PCACHE_DISABLE.

This function implements an operation of the PrefetchEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsPrefetchEnable in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_PREFETCH_ENABLE region;
region = PLIB_PCACHE_PrefetchEnableGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

PLIB_PCACHE_PREFETCH_ENABLE PLIB_PCACHE_PrefetchEnableGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_PrefetchEnableSet Function

Sets the predictive Prefetch state for the Prefetch Cache module.

File

plib_pcache.h

C

```
void PLIB_PCACHE_PrefetchEnableSet(PCACHE_MODULE_ID index, PLIB_PCACHE_PREFETCH_ENABLE region);
```

Returns

None.

Description

This function sets the predictive Prefetch state for the Prefetch Cache module.

Remarks

At reset, the Prefetch Cache module is disabled.

This function implements an operation of the PrefetchEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsPrefetchEnable in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_PrefetchEnableSet(PCACHE_ID_0,
                              PLIB_PCACHE_PREFETCH_ENABLE_ALL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
region	PLIB_PCACHE_PREFETCH_ENABLE

Function

void PLIB_PCACHE_PrefetchEnableSet(PCACHE_MODULE_ID index,
PLIB_PCACHE_PREFETCH_ENABLE region)

e) Prefetch Status Functions

PLIB_PCACHE_PrefetchAbortRead Function

Reads the number of prefetch aborts.

File

plib_pcache.h

C

```
uint32_t PLIB_PCACHE_PrefetchAbortRead(PCACHE_MODULE_ID index);
```

Returns

The number of prefetch aborts.

Description

This function reads and returns the number of prefetch aborts.

Remarks

This function implements an operation of the PrefetchAbort feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsPrefetchAbort in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t pfabt;
pfabt = PLIB_PCACHE_PrefetchAbortRead(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

uint32_t PLIB_PCACHE_PrefetchAbortRead(PCACHE_MODULE_ID index)

PLIB_PCACHE_PrefetchAbortWrite Function

Sets the number of prefetch aborts.

File

plib_pcache.h

C

```
void PLIB_PCACHE_PrefetchAbortWrite(PCACHE_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function Sets the number of prefetch aborts.

Remarks

This function implements an operation of the PrefetchAbort feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsPrefetchAbort in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_PrefetchAbortWrite(PCACHE_ID_0, 0xF00);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be written
data	Number of prefetch aborts

Function

void PLIB_PCACHE_PrefetchAbortWrite(PCACHE_MODULE_ID index, uint32_t data)

f) Flash ECC Functions

PLIB_PCACHE_FlashDEDStatusClear Function

Clears a bus exception caused by a double-bit error.

File

plib_pcache.h

C

```
void PLIB_PCACHE_FlashDEDStatusClear(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function clears a bus exception caused by a double-bit error.

Remarks

The DED status is set by hardware.

This function implements an operation of the FlashDEDStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashDEDStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_FlashDEDStatusClear(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be cleared

Function

void PLIB_PCACHE_FlashDEDStatusClear(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashDEDStatusGet Function

Determines if a bus exception was caused by a double-bit error.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_FlashDEDStatusGet(PCACHE_MODULE_ID index);
```

Returns

- true - A double-bit error was detected
- false - A double-bit error was not detected

Description

This function determines if a bus exception was caused by a double-bit error.

Remarks

The Double-bit Error Detected (DED) status is set by hardware.

This function implements an operation of the FlashDEDStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashDEDStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool dedStatus;
dedStatus = PLIB_PCACHE_FlashDEDStatusGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

bool PLIB_PCACHE_FlashDEDStatusGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashSECCountGet Function

Returns the current value of the Flash SEC counter.

File

plib_pcache.h

C

```
uint8_t PLIB_PCACHE_FlashSECCountGet(PCACHE_MODULE_ID index);
```

Returns

- count - Number of SEC events to occur before the SEC status is set to true

Description

This function returns the current value of the Flash SEC counter.

Remarks

This function implements an operation of the FlashSECCount feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECCount in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t count;
count = PLIB_PCACHE_FlashSECCountGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

uint8_t PLIB_PCACHE_FlashSECCountGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashSECCountSet Function

Sets the number of single-bit error corrections that must occur before the SEC status is set to true.

File

plib_pcache.h

C

```
void PLIB_PCACHE_FlashSECCountSet(PCACHE_MODULE_ID index, uint8_t count);
```

Returns

None.

Description

This function sets the number of single-bit error corrections that must occur before the SEC status is set to true.

Remarks

This function implements an operation of the FlashSECCount feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECCount in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_FlashSECCountSet(PCACHE_ID_0, 255);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

count	Number of SEC events to occur before SEC status is set to true

Function

void PLIB_PCACHE_FlashSECCountSet(PCACHE_MODULE_ID index, uint8_t count)

PLIB_PCACHE_FlashSECIntDisable Function

Disables an interrupt on SEC.

File

plib_pcache.h

C

```
void PLIB_PCACHE_FlashSECIntDisable(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function disables an interrupt on SEC.

Remarks

At reset the SEC interrupt is disabled.

This function implements an operation of the FlashSECInt feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECInt in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_FlashSECIntDisable(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be disabled

Function

void PLIB_PCACHE_FlashSECIntDisable(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashSECIntEnable Function

Enables an interrupt on SEC.

File

plib_pcache.h

C

```
void PLIB_PCACHE_FlashSECIntEnable(PCACHE_MODULE_ID index);
```

Returns

None

Description

This function enables an interrupt on SEC.

Remarks

At reset the SEC interrupt is disabled.

This function implements an operation of the FlashSECInt feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECInt in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_FlashSECIntEnable(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be enabled

Function

void PLIB_PCACHE_FlashSECIntEnable(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashSECStatusClear Function

Sets the single-bit error corrected status to false.

File

plib_pcache.h

C

```
void PLIB_PCACHE_FlashSECStatusClear(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the single-bit error corrected status to false.

Remarks

The SEC status must be cleared in response to a Prefetch Cache SEC interrupt.

This function implements an operation of the FlashSECStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_FlashSECStatusClear(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be cleared

Function

void PLIB_PCACHE_FlashSECStatusClear(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashSECStatusGet Function

Determines if a SEC event triggered a Prefetch Cache interrupt.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_FlashSECStatusGet(PCACHE_MODULE_ID index);
```

Returns

- true - A single-bit error was corrected

- false - A double-bit error was not corrected

Description

This function determines if a SEC event triggered a Prefetch Cache interrupt.

Remarks

The SEC bit is set when a single-bit error is corrected and the SEC counter is zero. If the Flash SEC interrupt is enabled, an error event is reported to the CPU by a Prefetch Cache interrupt event.

This function implements an operation of the FlashSECStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool secStatus;
secStatus = PLIB_PCACHE_FlashSECStatusGet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be read

Function

bool PLIB_PCACHE_FlashSECStatusGet(PCACHE_MODULE_ID index)

PLIB_PCACHE_FlashSECStatusSet Function

Sets the single-bit error corrected status to true.

File

plib_pcache.h

C

```
void PLIB_PCACHE_FlashSECStatusSet(PCACHE_MODULE_ID index);
```

Returns

None.

Description

This function sets the single-bit error corrected status to true.

Remarks

This function is provided for code testing and debug purposes. Setting the SEC status while the SEC count is zero and the SEC interrupt is enabled will trigger a Prefetch Cache interrupt to the CPU.

This function implements an operation of the FlashSECStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsFlashSECStatus in your application to to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PCACHE_FlashSECStatusSet(PCACHE_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PCACHE_FlashSECStatusSet(PCACHE_MODULE_ID index)

g) Feature Existence Functions

PLIB_PCACHE_ExistsCacheHit Function

Identifies that the CacheHit feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheHit(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheHit feature is supported on the device
- false - The CacheHit feature is not supported on the device

Description

This interface identifies that the CacheHit feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheHitRead
- PLIB_PCACHE_CacheHitWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheHit(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLine Function

Identifies that the CacheLineSelect feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLine(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineSelect feature is supported on the device
- false - The CacheLineSelect feature is not supported on the device

Description

This interface identifies that the CacheLineSelect feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineSelect
- PLIB_PCACHE_CacheLineDeselect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLine(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLineAddr Function

Identifies that the CacheLineAddr feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLineAddr(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineAddr feature is supported on the device
- false - The CacheLineAddr feature is not supported on the device

Description

This interface identifies that the CacheLineAddr feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineAddrSet
- PLIB_PCACHE_CacheLineAddrGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLineAddr(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLineFlashType Function

Identifies that the CacheLineFlashType feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLineFlashType(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineFlashType feature is supported on the device
- false - The CacheLineFlashType feature is not supported on the device

Description

This interface identifies that the CacheLineFlashType feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineFlashTypeBoot
- PLIB_PCACHE_CacheLineFlashTypeInst
- PLIB_PCACHE_CacheLineFlashTypeIsInst

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLineFlashType(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLineLock Function

Identifies that the CacheLineLock feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLineLock(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineLock feature is supported on the device
- false - The CacheLineLock feature is not supported on the device

Description

This interface identifies that the CacheLineLock feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineLock
- PLIB_PCACHE_CacheLineUnlock
- PLIB_PCACHE_CacheLineIsLocked

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLineLock(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLineMask Function

Identifies that the CacheLineMask feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLineMask(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineMask feature is supported on the device
- false - The CacheLineMask feature is not supported on the device

Description

This interface identifies that the CacheLineMask feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineMaskSet
- PLIB_PCACHE_CacheLineMaskGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLineMask(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLineType Function

Identifies that the CacheLineType feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLineType(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineType feature is supported on the device
- false - The CacheLineType feature is not supported on the device

Description

This interface identifies that the CacheLineType feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineData
- PLIB_PCACHE_CacheLineInst
- PLIB_PCACHE_CacheLineIsInst

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLineType(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheLineValid Function

Identifies that the CacheLineValid feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheLineValid(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheLineValid feature is supported on the device
- false - The CacheLineValid feature is not supported on the device

Description

This interface identifies that the CacheLineValid feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheLineValid
- PLIB_PCACHE_CacheLineInvalid
- PLIB_PCACHE_CacheLineIsValid

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheLineValid(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsCacheMiss Function

Identifies that the CacheMiss feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsCacheMiss(PCACHE_MODULE_ID index);
```

Returns

- true - The CacheMiss feature is supported on the device
- false - The CacheMiss feature is not supported on the device

Description

This interface identifies that the CacheMiss feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_CacheMissRead
- PLIB_PCACHE_CacheMissWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsCacheMiss(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsDataCacheEnable Function

Identifies that the DataCacheEnable feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsDataCacheEnable(PCACHE_MODULE_ID index);
```

Returns

- true - The DataCacheEnable feature is supported on the device
- false - The DataCacheEnable feature is not supported on the device

Description

This interface identifies that the DataCacheEnable feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_DataCacheEnableSet
- PLIB_PCACHE_DataCacheEnableGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsDataCacheEnable(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsFlashDEDStatus Function

Identifies that the FlashDEDStatus feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsFlashDEDStatus(PCACHE_MODULE_ID index);
```

Returns

- true - The FlashDEDStatus feature is supported on the device
- false - The FlashDEDStatus feature is not supported on the device

Description

This interface identifies that the FlashDEDStatus feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_FlashDEDStatusGet
- PLIB_PCACHE_FlashDEDStatusClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsFlashDEDStatus(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsFlashSECCount Function

Identifies that the FlashSECCount feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsFlashSECCount(PCACHE_MODULE_ID index);
```

Returns

- true - The FlashSECCount feature is supported on the device
- false - The FlashSECCount feature is not supported on the device

Description

This interface identifies that the FlashSECCount feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_FlashSECCountSet
- PLIB_PCACHE_FlashSECCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsFlashSECCount(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsFlashSECInt Function

Identifies that the FlashSECInt feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsFlashSECInt(PCACHE_MODULE_ID index);
```

Returns

- true - The FlashSECInt feature is supported on the device
- false - The FlashSECInt feature is not supported on the device

Description

This interface identifies that the FlashSECInt feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_FlashSECIntEnable
- PLIB_PCACHE_FlashSECIntDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsFlashSECInt(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsFlashSECStatus Function

Identifies that the FlashSECStatus feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsFlashSECStatus(PCACHE_MODULE_ID index);
```

Returns

- true - The FlashSECStatus feature is supported on the device
- false - The FlashSECStatus feature is not supported on the device

Description

This interface identifies that the FlashSECStatus feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_FlashSECStatusGet
- PLIB_PCACHE_FlashSECStatusSet
- PLIB_PCACHE_FlashSECStatusClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsFlashSECStatus(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsInvalidateOnPFMProgram Function

Identifies that the InvalidateOnPFMProgram feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsInvalidateOnPFMProgram(PCACHE_MODULE_ID index);
```

Returns

- true - The InvalidateOnPFMProgram feature is supported on the device
- false - The InvalidateOnPFMProgram feature is not supported on the device

Description

This interface identifies that the InvalidateOnPFMProgram feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_InvalidateOnPFMProgramAll
- PLIB_PCACHE_InvalidateOnPFMProgramUnlocked

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsInvalidateOnPFMProgram(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsLeastRecentlyUsedState Function

Identifies that the LeastRecentlyUsedState feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsLeastRecentlyUsedState(PCACHE_MODULE_ID index);
```

Returns

- true - The LeastRecentlyUsedState feature is supported on the device
- false - The LeastRecentlyUsedState feature is not supported on the device

Description

This interface identifies that the LeastRecentlyUsedState feature is available on the Prefetch Cache module. When this interface returns true, this function is supported on the device:

- PLIB_PCACHE_LeastRecentlyUsedStateRead

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsLeastRecentlyUsedState(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsPrefetchAbort Function

Identifies that the PrefetchAbort feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsPrefetchAbort(PCACHE_MODULE_ID index);
```

Returns

- true - The PrefetchAbort feature is supported on the device
- false - The PrefetchAbort feature is not supported on the device

Description

This interface identifies that the PrefetchAbort feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_PrefetchAbortRead
- PLIB_PCACHE_PrefetchAbortWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsPrefetchAbort(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsPrefetchEnable Function

Identifies that the PrefetchEnable feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsPrefetchEnable(PCACHE_MODULE_ID index);
```

Returns

- true - The PrefetchEnable feature is supported on the device
- false - The PrefetchEnable feature is not supported on the device

Description

This interface identifies that the PrefetchEnable feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_PrefetchEnableSet
- PLIB_PCACHE_PrefetchEnableGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsPrefetchEnable(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsWaitState Function

Identifies that the WaitState feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsWaitState(PCACHE_MODULE_ID index);
```

Returns

- true - The WaitState feature is supported on the device
- false - The WaitState feature is not supported on the device

Description

This interface identifies that the WaitState feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_WaitStateSet
- PLIB_PCACHE_WaitStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsWaitState(PCACHE_MODULE_ID index)

PLIB_PCACHE_ExistsWord Function

Identifies that the Word feature exists on the Prefetch Cache module.

File

plib_pcache.h

C

```
bool PLIB_PCACHE_ExistsWord(PCACHE_MODULE_ID index);
```

Returns

- true - The Word feature is supported on the device
- false - The Word feature is not supported on the device

Description

This interface identifies that the Word feature is available on the Prefetch Cache module. When this interface returns true, these functions are supported on the device:

- PLIB_PCACHE_WordRead
- PLIB_PCACHE_WordWrite

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PCACHE_ExistsWord(PCACHE_MODULE_ID index)

h) Data Types and Constants

PLIB_PCACHE_MAX_SEC_COUNT Macro

Defines the maximum value for single bit error counter.

File

help_plib_pcache.h

C

```
#define PLIB_PCACHE_MAX_SEC_COUNT 255
```

Description

Maximum Value for Single Bit Error Counter

This definition specifies the maximum value for single bit error counter.

Remarks

The actual maximum value of the SEC counter is processor specific and is defined in the processor-specific header files (see processor.h).

PLIB_PCACHE_NUM_LINES Macro

Defines the number of available Prefetch Cache Lines.

File

help_plib_pcache.h

C

```
#define PLIB_PCACHE_NUM_LINES
```

Description

Number of Cache Lines

This definition specifies the number of Prefetch Cache Lines available.

Remarks

The actual number of cache lines is processor specific and is defined in the processor-specific header files (see processor.h).

PLIB_PCACHE_NUM_WORDS_PER_LINE Macro

Defines the number of Words per Prefetch Cache Line.

File

help_plib_pcache.h

C

```
#define PLIB_PCACHE_NUM_WORDS_PER_LINE
```

Description

Number of Words per Cache Line

This definition specifies the number of Words per Prefetch Cache Line.

Remarks

The actual number of words per cache line is processor specific and is defined in the processor-specific header files (see processor.h).

PCACHE_MODULE_ID Enumeration

Lists the possible Module IDs for the Prefetch Cache.

File

help_plib_pcache.h

C

```
typedef enum {
  PCACHE_ID_1,
  PCACHE_NUMBER_OF_MODULES
} PCACHE_MODULE_ID;
```

Description

Module ID

This enumeration lists the possible Module IDs for the Prefetch Cache.

Remarks

Refer to the data sheet to get the correct number of modules defined for the desired device.

PLIB_PCACHE_DATA_ENABLE Enumeration

Lists the possible predictive prefetch states for the Prefetch Cache.

File

help_plib_pcache.h

C

```
typedef enum {
  PLIB_PCACHE_DATA_DISABLE,
  PLIB_PCACHE_DATA_1LINE,
  PLIB_PCACHE_DATA_2LINE,
  PLIB_PCACHE_DATA_4LINE
} PLIB_PCACHE_DATA_ENABLE;
```

Members

Members	Description
PLIB_PCACHE_DATA_DISABLE	Disable data caching
PLIB_PCACHE_DATA_1LINE	Enable data caching with a size of 1 Line
PLIB_PCACHE_DATA_2LINE	Enable data caching with a size of 2 Lines
PLIB_PCACHE_DATA_4LINE	Enable data caching with a size of 4 Lines

Description

Data Cache Enable

This enumeration lists the possible predictive prefetch states for the Prefetch Cache.

Remarks

Not all settings are available for all devices. See the device-specific data sheet for details. This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

PLIB_PCACHE_PREFETCH_ENABLE Enumeration

Lists the possible predictive prefetch states for the Prefetch Cache.

File

help_plib_pcache.h

C

```
typedef enum {
  PLIB_PCACHE_PREFETCH_DISABLE,
  PLIB_PCACHE_PREFETCH_ENABLE_CACHED_REGIONS,
  PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST,
  PLIB_PCACHE_PREFETCH_ENABLE_NONCACHED_REGIONS,
  PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST_DATA,
  PLIB_PCACHE_PREFETCH_ENABLE_ALL
} PLIB_PCACHE_PREFETCH_ENABLE;
```

Members

Members	Description
PLIB_PCACHE_PREFETCH_DISABLE	Disable predictive prefetch enable
PLIB_PCACHE_PREFETCH_ENABLE_CACHED_REGIONS	Enable predictive prefetch cache for cacheable regions only
PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST	Enable predictive prefetch cache for cacheable regions only
PLIB_PCACHE_PREFETCH_ENABLE_NONCACHED_REGIONS	Enable predictive prefetch cache for non-cacheable regions only
PLIB_PCACHE_PREFETCH_ENABLE_CPU_INST_DATA	Enable predictive prefetch cache for non-cacheable regions only
PLIB_PCACHE_PREFETCH_ENABLE_ALL	Enable predictive prefetch cache for cacheable and non-cacheable regions

Description

Predictive Prefetch Cache Enable Bits

This enumeration lists the possible predictive prefetch states for the Prefetch Cache.

Remarks

Not all settings are available for all devices. See the device-specific data sheet for details. This enumeration is processor specific and is defined in the processor-specific header files (see processor.h).

Files

Files

Name	Description
plib_pcache.h	Defines the Prefetch Cache Peripheral Library Interface
help_plib_pcache.h	This file is used for documentation purposes

Description

This section lists the source and header files used by the library.

plib_pcache.h

Defines the Prefetch Cache Peripheral Library Interface

Functions

	Name	Description
	PLIB_PCACHE_CacheHitRead	Reads the number of cache hits.
	PLIB_PCACHE_CacheHitWrite	Sets the number of cache hits.
	PLIB_PCACHE_CacheLineAddrGet	Gets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineAddrSet	Sets the TAG address in the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineData	Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to data type.
	PLIB_PCACHE_CacheLineDeselect	Disables write access to the cache line specified by cache_line.
	PLIB_PCACHE_CacheLineFlashTypeBoot	Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Boot Flash.
	PLIB_PCACHE_CacheLineFlashTypeInst	Set the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect as residing in Instruction Flash.
	PLIB_PCACHE_CacheLineFlashTypeIsInst	Returns true if the cache line tag for the line previously write-enabled by PLIB_PCACHE_CacheLineSelect shows the line is residing in Instruction Flash.
	PLIB_PCACHE_CacheLineInst	Sets cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect to instruction type.
	PLIB_PCACHE_CacheLineInvalid	Invalidates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineIsInst	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is set to instruction type.
	PLIB_PCACHE_CacheLineIsLocked	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is locked.
	PLIB_PCACHE_CacheLineIsValid	Returns true if cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect is valid.

	PLIB_PCACHE_CacheLineLock	Locks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineMaskGet	Returns the current state of the cache tag mask for the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineMaskSet	Sets the cache tag mask to force a match on set bits on the cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineSelect	Enables write access to the cache line specified by cache_line.
	PLIB_PCACHE_CacheLineUnlock	Unlocks cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheLineValid	Validates cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_CacheMissRead	Reads the number of cache misses.
	PLIB_PCACHE_CacheMissWrite	Sets the number of cache misses.
	PLIB_PCACHE_DataCacheEnableGet	Gets the data cache enable bits.
	PLIB_PCACHE_DataCacheEnableSet	Sets the data cache enable bits.
	PLIB_PCACHE_ExistsCacheHit	Identifies that the CacheHit feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLine	Identifies that the CacheLineSelect feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineAddr	Identifies that the CacheLineAddr feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineFlashType	Identifies that the CacheLineFlashType feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineLock	Identifies that the CacheLineLock feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineMask	Identifies that the CacheLineMask feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineType	Identifies that the CacheLineType feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheLineValid	Identifies that the CacheLineValid feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsCacheMiss	Identifies that the CacheMiss feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsDataCacheEnable	Identifies that the DataCacheEnable feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashDEDStatus	Identifies that the FlashDEDStatus feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECCount	Identifies that the FlashSECCount feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECInt	Identifies that the FlashSECInt feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsFlashSECStatus	Identifies that the FlashSECStatus feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsInvalidateOnPFMProgram	Identifies that the InvalidateOnPFMProgram feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsLeastRecentlyUsedState	Identifies that the LeastRecentlyUsedState feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsPrefetchAbort	Identifies that the PrefetchAbort feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsPrefetchEnable	Identifies that the PrefetchEnable feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsWaitState	Identifies that the WaitState feature exists on the Prefetch Cache module.
	PLIB_PCACHE_ExistsWord	Identifies that the Word feature exists on the Prefetch Cache module.
	PLIB_PCACHE_FlashDEDStatusClear	Clears a bus exception caused by a double-bit error.
	PLIB_PCACHE_FlashDEDStatusGet	Determines if a bus exception was caused by a double-bit error.
	PLIB_PCACHE_FlashSECCountGet	Returns the current value of the Flash SEC counter.
	PLIB_PCACHE_FlashSECCountSet	Sets the number of single-bit error corrections that must occur before the SEC status is set to true.
	PLIB_PCACHE_FlashSECIntDisable	Disables an interrupt on SEC.
	PLIB_PCACHE_FlashSECIntEnable	Enables an interrupt on SEC.
	PLIB_PCACHE_FlashSECStatusClear	Sets the single-bit error corrected status to false.
	PLIB_PCACHE_FlashSECStatusGet	Determines if a SEC event triggered a Prefetch Cache interrupt.
	PLIB_PCACHE_FlashSECStatusSet	Sets the single-bit error corrected status to true.

	PLIB_PCACHE_InvalidateOnPFMProgramAll	Sets Prefetch Cache to invalidate all data and instruction lines on a PFM program cycle.
	PLIB_PCACHE_InvalidateOnPFMProgramUnlocked	Sets Prefetch Cache to invalidate all unlocked data and instruction lines on a PFM program cycle.
	PLIB_PCACHE_LeastRecentlyUsedStateRead	Reads the state of the cache Least Recently Used (LRU) encoding bits.
	PLIB_PCACHE_PrefetchAbortRead	Reads the number of prefetch aborts.
	PLIB_PCACHE_PrefetchAbortWrite	Sets the number of prefetch aborts.
	PLIB_PCACHE_PrefetchEnableGet	Gets the predictive Prefetch state for the Prefetch Cache module.
	PLIB_PCACHE_PrefetchEnableSet	Sets the predictive Prefetch state for the Prefetch Cache module.
	PLIB_PCACHE_WaitStateGet	Gets the Prefetch Cache access time.
	PLIB_PCACHE_WaitStateSet	Sets the Prefetch Cache access time.
	PLIB_PCACHE_WordRead	Reads the word specified by word from cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.
	PLIB_PCACHE_WordWrite	Writes the word (specified by word parameter) with data (specified by data parameter) to cache line previously write-enabled by PLIB_PCACHE_CacheLineSelect.

Description

Prefetch Cache Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Prefetch Cache Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Prefetch Cache module.

File Name

plib_pcache.h

Company

Microchip Technology Inc.

help_plib_pcache.h

Enumerations

	Name	Description
	PCACHE_MODULE_ID	Lists the possible Module IDs for the Prefetch Cache.
	PLIB_PCACHE_DATA_ENABLE	Lists the possible predictive prefetch states for the Prefetch Cache.
	PLIB_PCACHE_PREFETCH_ENABLE	Lists the possible predictive prefetch states for the Prefetch Cache.

Macros

	Name	Description
	PLIB_PCACHE_MAX_SEC_COUNT	Defines the maximum value for single bit error counter.
	PLIB_PCACHE_NUM_LINES	Defines the number of available Prefetch Cache Lines.
	PLIB_PCACHE_NUM_WORDS_PER_LINE	Defines the number of Words per Prefetch Cache Line.

Description

This file is used for documentation purposes

Ports Peripheral Library

This section describes the Ports Peripheral Library.

Introduction

This library provides a low-level abstraction of the I/O Ports module on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The general purpose I/O pins can be considered the simplest of peripherals. These I/O pins allow the microcontroller to monitor and control other devices. To add flexibility and functionality to a device, some pins are multiplexed with alternate function(s). These functions depend on which peripheral features are on the device. In general, when a peripheral is functioning, that pin may not be used as a general purpose I/O pin.

A major challenge in general purpose devices is providing the largest possible set of peripheral features while minimizing the conflict of features on I/O pins. The Peripheral Pin Select (PPS) feature simplifies this challenge by enabling the user's peripheral set selection and their placement on a wide range of I/O pins. By increasing the pin out options available on a particular device, users can better tailor the microcontroller to their entire application. The PPS feature operates over a fixed subset of digital I/O pins. Users may independently map the input and/or output of any one of many digital peripherals to any one of these I/O pins. PPS is performed in software and generally does not require the device to be reprogrammed. Hardware safeguards are included that prevent accidental or spurious changes to the peripheral mapping once it has been established.

Note: Peripheral Pin Select is not available on all devices. Please refer to the specific device data sheet to determine availability of this feature.

Following are some of the key features of the I/O Ports module:

- Individual output pin/port open-drain control
- Individual input pin/port pull-up/down control
- Monitor select inputs and generate interrupt on mismatch condition (Change Notification)
- Operation during Sleep and Idle modes
- Port line analog/digital selection
- Port slew rate control
- PPS peripheral function remapping

Using the Library

This topic describes the basic architecture of the Ports Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_ports.h`

The interface to the Ports Peripheral Library is defined in the `plib_ports.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Ports Peripheral Library must include `peripheral.h`.

Library File:

The Ports Peripheral Library is part of the processor-specific peripheral library installed with the compiler. This library is automatically available (in the default search path) for any project built using the Microchip compilers.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

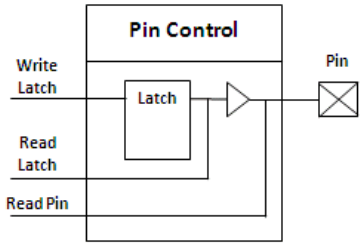
Hardware Abstraction Model

This topic provides detailed information for general purpose I/O and pin mapping.

Description

General Purpose I/O

All port pins have three controls directly associated with their operation as digital I/O. The data direction control determines whether the pin is an input or an output. All port pins are defined as inputs after a Reset. Writes to a port are latched and, if programmed as an output, driven by a buffer to the pin. The value of pin can be read directly from an input buffer, (after it has been synchronized to the clock), regardless of whether the pin direction has been selected as input or output. Additionally, the value being driven by the latch before the output buffer, can also be read.



Internal pull-up resistors are available on selected port pins to eliminate the need to use external pull-up resistors. These pull-ups can be controlled by this library. The open-drain feature allows the generation of outputs higher than VDD on any desired digital-only pins by using external pull-up resistors.

Note: For detailed specifications on the maximum allowed open-drain voltage, refer to the **"Electrical Characteristics"** chapter in the specific device data sheet.

The input change notification feature of the I/O ports allows the microcontrollers to generate interrupt requests to the processor in response to a change of state on selected input pins. This feature is capable of detecting input change of states even in Sleep mode, when the clocks are disabled.

The alternate pin function selections are used to steer specific peripheral input and output functions between different pins.

The output slew rate of some port pins are programmable to select either the standard transition rate or a reduced transition rate of 'x' times the standard to minimize EMI.

Peripheral Pin Select

In addition to general purpose IO control, this library also provides a low-level abstraction of the PPS module on Microchip microcontrollers with a convenient C language interface.

Available Pins

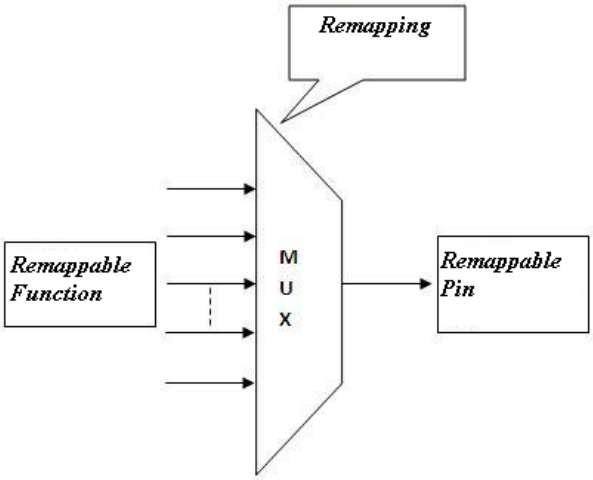
The number of available pins is dependent on the particular device and its pin count. Pins that support the PPS feature include the designation 'RPn' or 'RPIn' in their full pin designation, where 'RP' designates a remappable peripheral and 'n' is the remappable pin number. RP is used to designate pins that support both remappable input and output functions, while RPI indicates pins that only support remappable input functions.

Available Peripherals

The peripherals managed by PPS are all digital-only peripherals. These include general serial communications (UART and SPI), general purpose timer clock inputs, timer-related peripherals (Input Capture and Output Compare), comparator digital output, interrupt-on-change inputs, etc.

In comparison, some digital-only peripheral modules are never included in the peripheral pin select feature. This is because the peripheral's function requires special I/O circuitry on a specific port and cannot be easily connected to multiple pins. These modules include I2C, among others. A similar requirement excludes all modules with analog inputs, such as the Analog-to-Digital Converter (ADC).

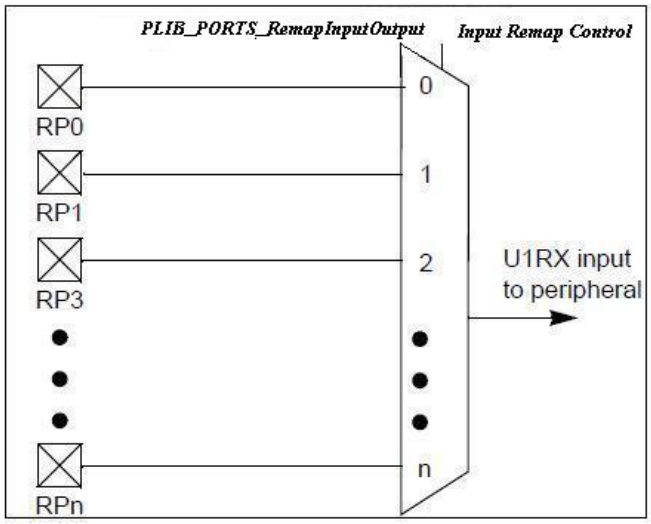
If multiple peripherals are enabled on the same pin(s), there is an internal priority that decides which function is mapped to the pin. When a remappable peripheral is active on a given I/O pin, it takes priority over all other digital I/O and digital communication peripherals associated with the pin. Priority is given regardless of the type of peripheral that is mapped. Remappable peripherals never take priority over any analog functions associated with the pin. In other words, If an analog function is enabled on the pin, the PPS input will be disabled.



Input Mapping

The inputs of the peripheral pin select options are mapped on the basis of the peripheral.

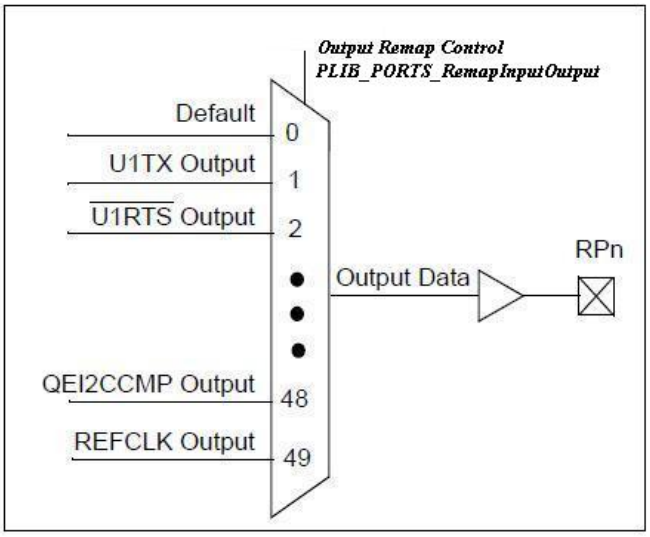
Remappable Input Example for USART 1 Receive



Output Mapping

In contrast to inputs, the outputs of the peripheral pin select options are mapped on the basis of the pin. In this case, a control associated with a particular pin dictates the peripheral output to be mapped.

Multiplexing of Remappable Output for RPn



Mapping Limitations

The control schema of the peripheral select pins is not limited to a small range of fixed peripheral configurations. There are no mutual-exclusion or hardware-enforced lockouts between any of the peripheral mapping SFRs. Literally any combination of peripheral mappings across any or all of the RPn pins is possible. This includes both many-to-one and one-to-many mappings of peripheral inputs and outputs to pins. While such mappings may be technically possible from a configuration point of view, they may not be supportable from an electrical point of view and may cause damage to the part.

Notes:

1. For detailed specifications on the mapping limitations, refer to the **"I/O Ports"** chapter in the specific device data sheet.
2. Regrading pin priorities, the priority of the pins are assigned from left to right, left being the highest priority and the right most being the least priority.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Ports module.

Library Interface Section	Description
Port Functions	Port access read/write/toggle/clear interfaces with 8-/16-/32-bit widths based on the selected device.
Port Pin Functions	Port bit/pin read/write/toggle/clear interfaces.
Change Notification Functions	Port change notification.
Peripheral Pin Select Functions	Interface routines for mapping the digital input/output to a specific the PPS remappable input/output pin.
Feature Existence Functions	Determine whether certain features exist.

How the Library Works

How the Library Works

The following topics discuss the processes involved while using a Ports module:

- Ports Control
- Ports Function Remap
- Ports Change Notification
- Special Considerations

Ports Control

Port Functions Usage:

- Port Read - Ports can be read at byte/word level using PLIB_PORTS_Read, and at the bit/pin level using PLIB_PORTS_PinGet
- Port Write - Ports can be written to at byte/word level using PLIB_PORTS_Write and at the bit/pin level using PLIB_PORTS_PinWrite. The function PLIB_PORTS_PinSet can be used to set a specific bit/pin of a port. A mask-based PORTx write can be accomplished using the function PLIB_PORTS_Set with the appropriate mask as a parameter along with the value.
- Port Clear - Ports can be cleared at byte/word level using PLIB_PORTS_Clear with appropriate mask as a parameter and at the bit/pin level using PLIB_PORTS_PinClear
- Port Direction - The ports read direction can be set at byte/word level using PLIB_PORTS_DirectionInputSet and at the bit/pin level using PLIB_PORTS_PinDirectionInputSet. Similarly, the ports write direction can be set at the byte/word level using PLIB_PORTS_DirectionOutputSet and at the bit/pin level using PLIB_PORTS_PinDirectionOutputSet. The direction information can be read using PLIB_PORTS_DirectionGet.
- Port Toggle - Ports can be toggled at the byte/word level using PLIB_PORTS_Toggle with the appropriate mask as a parameter and at the bit/pin level using PLIB_PORTS_PinToggle
- Port Open Drain - Ports can be enabled for open-drain functionality at the byte/word level using the interface PLIB_PORTS_OpenDrainEnable and at the bit/pin level using PLIB_PORTS_PinOpenDrainEnable. Similarly, the ports can be disabled for open-drain functionality at the byte/word level using PLIB_PORTS_OpenDrainDisable and at the bit/pin level using PLIB_PORTS_PinOpenDrainDisable.

Example:

```
// PORT Direction setting for output
   PLIB_PORTS_DirectionOutputSet(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0xFFFF);

// Writing a value into a PORT
   PLIB_PORTS_Write(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_TYPE)0x1234);

// PORT Direction setting for input
   PLIB_PORTS_DirectionInputSet(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0xFFFF);

// Reading back the previously written value
   PORTS_DATA_TYPE readData = PLIB_PORTS_Read(MY_PORTS_INSTANCE, MY_CHANNEL);

// Clearing the PORT
   PLIB_PORTS_Clear(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);

// Writing the value based on the mask, only 0x34 gets written to the PORT
   PLIB_PORTS_Set(MY_PORTS_INSTANCE, MY_CHANNEL, 0x1234, (PORTS_DATA_MASK)0x00FF);

// Toggling a PORT
   PLIB_PORTS_Toggle(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);
```

Ports Function Remap

Using the PPS Feature:

1. Ensure that any fixed digital peripherals on the pins to be used are disabled.
2. Ensure that pins with analog functionality are set to Digital mode.
3. Enable writing to the Input/Output control register using PLIB_DEVCON_SystemUnlock and PLIB_DEVCON_DeviceRegistersUnlock.
4. Remap the input and/or output pins using the internal PPS module. This can be achieved with the functions PLIB_PORTS_RemapInput and PLIB_PORTS_RemapOutput with the respective parameters defined in the processor specific headers.
5. Lock the Input/Output control register using PLIB_DEVCON_SystemUnlock and PLIB_DEVCON_DeviceRegistersUnlock.
6. Configure and enable newly mapped PPS peripherals.

Example:

```
// System Unlock
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);
// Unlock PPS registers
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID_0, DEVCON_PPS_REGISTERS);

// Remapping input function 'Input Capture 1' to the Remappable pin 'RPD2'
PLIB_PORTS_RemapInput(PORTS_ID_0, INPUT_FUNC_IC1, INPUT_PIN_RPD2 );

// Remapping output function 'UART3 Transmit' to the Remappable pin 'RPA14'
PLIB_PORTS_RemapOutput(PORTS_ID_0, OUTPUT_FUNC_U3TX, OUTPUT_PIN_RPA14);

// Lock PPS registers
PLIB_DEVCON_DeviceRegistersLock(DEVCON_ID_0, DEVCON_PPS_REGISTERS);
```

Note: For more information, refer to the **"I/O Ports"** chapter in the specific device data sheet or the family reference manual section specified in that chapter.

Ports Change Notification

Change Notification Feature Usage

1. Ensure that the change notify pin is configured as a digital input by setting the associated bit in the direction register using the PLIB_PORTS_PinDirectionInputSet or PLIB_PORTS_DirectionInputSet functions.

2. Select the desired change notification pin using PLIB_PORTS_PinChangeNoticeEnable.

3. Turn on the weak pull-up devices (if desired) for the selected change notification pins using PLIB_PORTS_ChangeNoticePullUpEnable (weak pull-ups can be disabled using PLIB_PORTS_ChangeNoticePullUpDisable).

4. Clear the selected change notification interrupt flag.

5. Select the desired interrupt priority for change notification pin.

6. Enable change notification interrupts.

7. Change notification can be disabled after the successful usage using PLIB_PORTS_PinChangeNoticeDisable. Certain microcontrollers support the global control over the change notification feature using the following functions: PLIB_PORTS_ChangeNoticeEnable and PLIB_PORTS_ChangeNoticeDisable. If there are any requirements to control the pull-downs, the following two functions could be used: PLIB_PORTS_ChangeNoticePullDownPerPortEnable and PLIB_PORTS_ChangeNoticePullDownPerPortDisable.

Change Notification Operation in Sleep and Idle Modes

The change notification feature continues to operate during Sleep or Idle mode. Its operation can be enabled or disabled using PLIB_PORTS_ChangeNoticeInIdleEnable or PLIB_PORTS_ChangeNoticeInIdleDisable. If one of the enabled change notification pins changes states, the respective status bit will be set, which can be monitored using PLIB_PORTS_ChangeNoticePerPortHasOccurred.

Example:

```
// Enabling the global change notification
PLIB_PORTS_ChangeNoticeEnable(MY_PORTS_INSTANCE);
// Enabling weak pull-ups for the change notification PIN 10
PLIB_PORTS_ChangeNoticePullUpEnable(MY_PORTS_INSTANCE, PORTS_CHANGE_NOTICE_PIN_10);
// Enabling change notification on PIN 10
PLIB_PORTS_PinChangeNoticeEnable(MY_PORTS_INSTANCE, PORTS_CHANGE_NOTICE_PIN_10);
```

Note: For more information, refer to the **"I/O Ports"** chapter in the specific device data sheet or the related family reference manual section specified in that chapter.

Special Considerations

Considerations on Ports Usage

When reading a port register, all pins configured as analog input channels will read as cleared.

Pins configured as digital inputs will not operate correctly with analog peripherals. Analog levels on any pin that is defined as a digital input may cause the input buffer of analog peripherals to consume current that exceeds the device specifications.

Pull-ups and pull-downs on change notification pins should always be disabled when the port pin is configured as a digital output.

One instruction cycle is required between a port direction change or port write operation and a read operation of the same port. Typically, this instruction would be a NOP.

Considerations for PPS

The ability to control PPS options introduces several considerations into application design that could be overlooked. This is particularly true for several common peripherals that are available only as remappable peripherals.

The main consideration is that the peripheral pin selects are not available on default pins in the device's default (i.e., Reset) state, all PPS inputs are tied to VSS and all PPS outputs are disconnected. This condition requires the user to initialize the device with the proper peripheral configuration before any other application code is executed. For application safety, however, it is best to lock the configuration after writing to the PPS control.

The assignment of a peripheral signal to a particular pin does not automatically perform any other configuration of the pin's I/O circuitry. In theory, this means adding a pin-selectable output to a pin may mean inadvertently driving an existing peripheral input when the output is driven. Users must be familiar with the behavior of other fixed peripherals that share a remappable pin and know when to enable or disable them. To be safe, fixed digital peripherals that share the same pin should be disabled when not in use. In addition, configuring a remappable pin for a specific peripheral does not automatically turn on that feature. The peripheral must be specifically configured for operation and enabled, as if it were tied to a fixed pin. Where this occurs in the application code (immediately following a device Reset and peripheral configuration or inside the main application routine) depends on the peripheral and its use in the application.

A final consideration is that PPS functions neither override analog inputs, nor reconfigure pins with analog functions for digital I/O. If a pin is configured as an analog input on a device Reset, it must be explicitly reconfigured as a digital I/O when used with PPS.

Note: For more information, refer to the **"I/O Ports"** chapter in the specific device data sheet or the family reference manual section specified in that chapter.

Configuring the Library

The library is configured for the supported Ports module when the supported processor is chosen in the MPLAB X IDE.

Library Interface

a) Port Pin Functions

	Name	Description
	PLIB_PORTS_PinClear	Clears the selected digital pin/latch.
	PLIB_PORTS_PinDirectionInputSet	Makes the selected pin direction input
	PLIB_PORTS_PinDirectionOutputSet	Makes the selected pin direction output
	PLIB_PORTS_PinGet	Reads/Gets data from the selected digital pin.
	PLIB_PORTS_PinModeSelect	Enables the selected pin as analog or digital.
	PLIB_PORTS_PinSet	Sets the selected digital pin/latch.
	PLIB_PORTS_PinToggle	Toggles the selected digital pin/latch.
	PLIB_PORTS_PinWrite	Writes the selected digital pin/latch.
	PLIB_PORTS_PinModePerPortSelect	Enables the selected port pin as analog or digital.
	PLIB_PORTS_PinGetLatched	Reads/Gets data from the selected latch.
	PLIB_PORTS_PinChangeNoticeEdgeHasOccurred	Check Change Notice edge status.
	PLIB_PORTS_PinChangeNoticeEdgeIsEnabled	Check if Change Notice edge is enabled or not.
	PLIB_PORTS_PinSlewRateGet	Gets the slew rate for selected port pin.
	PLIB_PORTS_PinOpenDrainDisable	Disables the open drain functionality for the selected pin.
	PLIB_PORTS_PinOpenDrainEnable	Enables the open drain functionality for the selected pin.

b) Port Functions

	Name	Description
	PLIB_PORTS_Clear	Clears the selected digital port/latch bits.

	PLIB_PORTS_DirectionGet	Reads the direction of the selected digital port.
	PLIB_PORTS_DirectionInputSet	Makes the selected pins direction input.
	PLIB_PORTS_DirectionOutputSet	Makes the selected pins direction output.
	PLIB_PORTS_Read	Reads the selected digital port.
	PLIB_PORTS_Set	Sets the selected bits of the port.
	PLIB_PORTS_Toggle	Toggles the selected digital port/latch.
	PLIB_PORTS_Write	Writes the selected digital port/latch.
	PLIB_PORTS_OpenDrainDisable	Disables the open drain functionality for the selected port.
	PLIB_PORTS_OpenDrainEnable	Enables the open drain functionality for the selected port pins.
	PLIB_PORTS_ReadLatched	Reads and returns data from the selected Latch.
	PLIB_PORTS_ChannelModeSelect	Enables the selected channel pins as analog or digital.
	PLIB_PORTS_ChannelSlewRateSelect	Selects the slew rate for selected channel pins.

c) Peripheral Pin Select Functions

	Name	Description
	PLIB_PORTS_RemapInput	Input function remapping.
	PLIB_PORTS_RemapOutput	Output function remapping.

d) Change Notification Functions

	Name	Description
	PLIB_PORTS_ChangeNoticeDisable	Global Change Notice disable.
	PLIB_PORTS_ChangeNoticeEnable	Global Change Notice enable.
	PLIB_PORTS_ChangeNoticeInIdleDisable	CPU Idle halts the Change Notice operation.
	PLIB_PORTS_ChangeNoticeInIdleEnable	CPU Idle mode does not affect Change Notice operation.
	PLIB_PORTS_ChangeNoticeInIdlePerPortDisable	Change Notification halts in Idle mode for selected channel.
	PLIB_PORTS_ChangeNoticeInIdlePerPortEnable	Allows CN to be working in Idle mode for selected channel.
	PLIB_PORTS_ChangeNoticePerPortHasOccured	This is function PLIB_PORTS_ChangeNoticePerPortHasOccured.
	PLIB_PORTS_ChangeNoticePerPortTurnOff	Disables the change notification for selected port.
	PLIB_PORTS_ChangeNoticePerPortTurnOn	Enables the change notification for selected port.
	PLIB_PORTS_ChangeNoticePullDownPerPortDisable	Disables the pull-down for selected Change Notice pins.
	PLIB_PORTS_ChangeNoticePullDownPerPortEnable	Enables the pull-down for selected Change Notice pins.
	PLIB_PORTS_ChangeNoticePullUpDisable	Disable pull-up on input change.
	PLIB_PORTS_ChangeNoticePullUpEnable	Enable pull-up on input change.
	PLIB_PORTS_ChangeNoticePullUpPerPortDisable	Disables weak pull-up for the selected pin.
	PLIB_PORTS_ChangeNoticePullUpPerPortEnable	Enables the pull-up for selected Change Notice pins.
	PLIB_PORTS_PinChangeNoticeDisable	Port pin Change Notice disable.
	PLIB_PORTS_PinChangeNoticeEnable	Port pin Change Notice interrupt enable.
	PLIB_PORTS_PinChangeNoticePerPortDisable	Disables CN interrupt for the selected pin.
	PLIB_PORTS_PinChangeNoticePerPortEnable	Enables CN interrupt for the selected pin.
	PLIB_PORTS_ChangeNoticePerPortHasOccurred	checks the status of change on the pin
	PLIB_PORTS_ChannelChangeNoticeDisable	Disables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_ChannelChangeNoticeEnable	Enables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_ChannelChangeNoticePullDownDisable	Disables Change Notice pull-down for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticePullDownEnable	Enables Change Notice pull-down for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticePullUpDisable	Disables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticePullUpEnable	Enables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticeEdgeDisable	Disables selected type of edge for selected CN pins.
	PLIB_PORTS_ChannelChangeNoticeEdgeEnable	Enables selected type of edge for selected CN pins.
	PLIB_PORTS_ChannelChangeNoticeMethodGet	Gets the Change Notice style for the selected port channel.
	PLIB_PORTS_ChannelChangeNoticeMethodSelect	Selects the Change Notice style for selected port channel.
	PLIB_PORTS_AnPinsModeSelect	Enables the selected AN pins as analog or digital.
	PLIB_PORTS_CnPinsDisable	Disables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_CnPinsEnable	Enables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_CnPinsPullUpDisable	Disables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_CnPinsPullUpEnable	Enables Change Notice pull-up for the selected channel pins.

e) Feature Existence Functions

	Name	Description
	PLIB_PORTS_ExistsChangeNotice	Identifies whether the ChangeNotice feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticeInIdle	Identifies whether the ChangeNoticeInIdle feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePerPortInIdle	Identifies whether the ChangeNoticeInIdlePerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePerPortStatus	Identifies whether the ChangeNoticePerPortStatus feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePerPortTurnOn	Identifies whether the ChangeNoticePerPortTurnOn feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePullDownPerPort	Identifies whether the ChangeNoticePullDownPerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePullUp	Identifies whether the ChangeNoticePullup feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePullUpPerPort	Identifies whether the ChangeNoticePullUpPerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsPinChangeNotice	Identifies whether the PinChangeNotice feature exists on the Ports module.
	PLIB_PORTS_ExistsPinChangeNoticePerPort	Identifies whether the PinChangeNoticePerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsPinMode	Identifies whether the PinMode feature exists on the Ports module.
	PLIB_PORTS_ExistsPinModePerPort	Identifies whether the PinModePerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsDirection	Identifies whether the PortsDirection feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsOpenDrain	Identifies whether the PortsOpenDrain feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsRead	Identifies whether the PortsRead feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsWrite	Identifies whether the PortsWrite feature exists on the Ports module.
	PLIB_PORTS_ExistsRemapInput	Identifies whether the RemapInput feature exists on the Ports module.
	PLIB_PORTS_ExistsRemapOutput	Identifies whether the RemapOutput feature exists on the Ports module.
	PLIB_PORTS_ExistsLatchRead	Identifies whether the LatchRead feature exists on the Ports module.
	PLIB_PORTS_ExistsAnPinsMode	Identifies whether the AnPinsMode feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticeEdgeControl	Identifies whether the ChangeNoticeEdgeControl feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticeEdgeStatus	Identifies whether the ChangeNoticeEdgeStatus feature exists on the Ports module.
	PLIB_PORTS_ExistsChannelChangeNoticeMethod	Identifies whether the ChannelChangeNoticeMethod feature exists on the Ports module.
	PLIB_PORTS_ExistsSlewRateControl	Identifies whether the SlewRateControl feature exists on the Ports module.

f) Data Types and Constants

	Name	Description
	PORTS_ANALOG_PIN	Data type defining the different analog input pins.
	PORTS_BIT_POS	Lists the constants that hold different PORTS bit positions.
	PORTS_CHANGE_NOTICE_PIN	Data type defining the different Change Notification (CN) pins enumerations.
	PORTS_CHANNEL	Identifies the available Ports channels.
	PORTS_DATA_MASK	Data type defining the Ports data mask
	PORTS_DATA_TYPE	Data type defining the Ports data type.
	PORTS_MODULE_ID	Identifies the available Ports modules.
	PORTS_PIN_MODE	Identifies the available pin modes.
	PORTS_REMAP_FUNCTION	Data type defining the different remap function enumerations.
	PORTS_REMAP_INPUT_FUNCTION	Data type defining the different remap input function enumerations.
	PORTS_REMAP_INPUT_PIN	Data type defining the different Ports I/O input pins enumerations.
	PORTS_REMAP_OUTPUT_FUNCTION	Data type defining the different remap output function enumerations.
	PORTS_REMAP_OUTPUT_PIN	Data type defining the different Ports I/O output pins enumerations.
	PORTS_REMAP_PIN	Data type defining the different remappable input/output enumerations.
	PORTS_AN_PIN	Data type defining the different analog input pins.
	PORTS_CN_PIN	Data type defining the different Change Notification (CN) pins enumerations.
	PORTS_CHANGE_NOTICE_EDGE	Data type defining the different edge type for change notification.
	PORTS_CHANGE_NOTICE_METHOD	Data type defining the different method of ports pin change notification.

	PORTS_PIN_SLEW_RATE	Data type defining the different slew rates for port pins.

Description

This section describes the Application Programming Interface (API) functions of the Ports Peripheral Library.

Refer to each section for a detailed description.

a) Port Pin Functions

PLIB_PORTS_PinClear Function

Clears the selected digital pin/latch.

File

plib_ports.h

C

```
void PLIB_PORTS_PinClear(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function clears the selected digital pin/latch.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Clears port pin RC4
PLIB_PORTS_PinClear(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void PLIB_PORTS_PinClear(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinDirectionInputSet Function

Makes the selected pin direction input

File

plib_ports.h

C

```
void PLIB_PORTS_PinDirectionInputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function makes the selected pin direction as input

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsDirection in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// make pin RC4 as input
PLIB_PORTS_PinDirectionInputSet(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS direction that has to be made input

Function

void PLIB_PORTS_PinDirectionInputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinDirectionOutputSet Function

Makes the selected pin direction output

File

plib_ports.h

C

```
void PLIB_PORTS_PinDirectionOutputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function makes the selected pin direction as output

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsDirection in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// make pin RC4 as output
PLIB_PORTS_PinDirectionOutputSet(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS direction that has to be made output

Function

void PLIB_PORTS_PinDirectionOutputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinGet Function

Reads/Gets data from the selected digital pin.

File

plib_ports.h

C

```
bool PLIB_PORTS_PinGet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

Port pin read data.

Description

This function reads/gets data from the selected digital PORT i/o pin. This function should be used to read the live data at the pin.

Remarks

For reading the Latched data, PLIB_PORTS_PinGetLatched function should be used.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsRead in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// read port pin RC4
bool bitStatus = PLIB_PORTS_PinGet(PORTS_ID_0, PORT_CHANNEL_C,
                                                    PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

bool PLIB_PORTS_PinGet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinModeSelect Function

Enables the selected pin as analog or digital.

File

plib_ports.h

C

```
void PLIB_PORTS_PinModeSelect(PORTS_MODULE_ID index, PORTS_ANALOG_PIN pin, PORTS_PIN_MODE mode);
```

Returns

None.

Description

This function enables the selected pin as analog or digital.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPinMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make AN0 pin as Analog
PLIB_PORTS_PinModeSelect(PORTS_ID_0, PORTS_ANALOG_PIN_0, PORTS_PIN_MODE_ANALOG);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pin	Possible values of PORTS_ANALOG_PIN
mode	Possible values of PORTS_PIN_MODE

Function

void PLIB_PORTS_PinModeSelect(PORTS_MODULE_ID index,
PORTS_ANALOG_PIN pin,
PORTS_PIN_MODE mode);

PLIB_PORTS_PinSet Function

Sets the selected digital pin/latch.

File

plib_ports.h

C

```
void PLIB_PORTS_PinSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function sets the selected digital pin/latch.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Sets port pin RC4
PLIB_PORTS_PinSet(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void PLIB_PORTS_PinSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinToggle Function

Toggles the selected digital pin/latch.

File

plib_ports.h

C

```
void PLIB_PORTS_PinToggle(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function toggles the selected digital pin/latch.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Toggles port pin RC4
PLIB_PORTS_PinToggle(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void PLIB_PORTS_PinToggle(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinWrite Function

Writes the selected digital pin/latch.

File

plib_ports.h

C

```
void PLIB_PORTS_PinWrite(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos, bool value);
```

Returns

None.

Description

This function writes to the selected digital pin/latch.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// write 'one' in port RC4
PLIB_PORTS_PinWrite(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4, 1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS
value	Value to be written to the specific pin/latch
true	sets the bit, false - clears the bit

Function

void PLIB_PORTS_PinWrite(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos,
bool value)

PLIB_PORTS_PinModePerPortSelect Function

Enables the selected port pin as analog or digital.

File

plib_ports.h

C

```
void PLIB_PORTS_PinModePerPortSelect(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos,
PORTS_PIN_MODE mode);
```

Returns

None.

Description

This function enables the selected port pin as analog or digital.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_PinModeSelect function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPinModePerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make RC5 pin Analog
PLIB_PORTS_PinModePerPortSelect(PORTS_ID_0, PORT_CHANNEL_C,
                                PORTS_BIT_POS_5, PORTS_PIN_MODE_ANALOG);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins
mode	Possible values of PORTS_PIN_MODE

Function

void PLIB_PORTS_PinModePerPortSelect(PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos,
PORTS_PIN_MODE mode);

PLIB_PORTS_PinGetLatched Function

Reads/Gets data from the selected latch.

File

plib_ports.h

C

```
bool PLIB_PORTS_PinGetLatched(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

Latch read data.

Description

This function reads/gets data from the selected PORTx Data Latch, not from the port I/O pins.

Remarks

For reading the Live data from the i/o pin, PLIB_PORTS_PinGet function should be used.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsLatchRead in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// read latch RC4
bool bitStatus = PLIB_PORTS_PinGetLatched(PORTS_ID_0, PORT_CHANNEL_C,
                                                      PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

bool PLIB_PORTS_PinGetLatched(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinChangeNoticeEdgeHasOccurred Function

Check Change Notice edge status.

File

plib_ports.h

C

```
bool PLIB_PORTS_PinChangeNoticeEdgeHasOccurred(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);
```

Returns

- true - Change Notice edge transition has occurred
- false - Change Notice edge transition has not occurred

Description

This function checks whether or no a Change Notice edge transition has occurred on the selected port pin.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeEdgeStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if Change Notice edge transition has occurred for pin RC1.
if (PLIB_PORTS_PinChangeNoticeEdgeHasOccurred(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_1))
{
```

```
    // do something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	One of the possible values of PORTS_BIT_POS

Function

bool PLIB_PORTS_PinChangeNoticeEdgeHasOccurred

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_BIT_POS bitPos

);

PLIB_PORTS_PinChangeNoticeEdgeIsEnabled Function

Check if Change Notice edge is enabled or not.

File

plib_ports.h

C

```
bool PLIB_PORTS_PinChangeNoticeEdgeIsEnabled(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos, PORTS_CHANGE_NOTICE_EDGE cnEdgeType);
```

Returns

- true - Selected type of Change Notice Edge is enabled.
- false - Selected type of Change Notice Edge is not enabled.

Description

This function checks if selected type of Change Notice edge is enabled on a particular port pin or not.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeEdgeControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Check if Change Notice at rising edge is enabled or not for pin RC1.
if (PLIB_PORTS_PinChangeNoticeEdgeIsEnabled(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_1,
PORTS_CHANGE_NOTICE_EDGE_RISING))
{
    // do something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	One of the possible values of PORTS_BIT_POS.
cnEdgeType	Type of the edge which has to be checked.

Function

bool PLIB_PORTS_PinChangeNoticeEdgeIsEnabled

(

PORTS_MODULE_ID index,

```
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos,
    PORTS_CHANGE_NOTICE_EDGE  cnEdgeType
);
```

PLIB_PORTS_PinSlewRateGet Function

Gets the slew rate for selected port pin.

File

plib_ports.h

C

```
PORTS_PIN_SLEW_RATE PLIB_PORTS_PinSlewRateGet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);
```

Returns

One of the possible values of PORTS_PIN_SLEW_RATE.

Description

This function gets the slew rate of selected port pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsSlewRateControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PORTS_PIN_SLEW_RATE slewRate;

// Get the slew rate of pin RC1
slewRate = PLIB_PORTS_PinSlewRateGet(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	One of the possible values of PORTS_BIT_POS.

Function

```
PORTS_PIN_SLEW_RATE PLIB_PORTS_PinSlewRateGet
(
    PORTS_MODULE_ID  index,
    PORTS_CHANNEL channel,
    PORTS_BIT_POS bitPos
);
```

PLIB_PORTS_PinOpenDrainDisable Function

Disables the open drain functionality for the selected pin.

File

plib_ports.h

C

```
void PLIB_PORTS_PinOpenDrainDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function disables the open drain functionality for the selected pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsOpenDrain in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable open drain for pin RC4
PLIB_PORTS_PinOpenDrainDisable(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	One of the possible values of PORTS_BIT_POS.

Function

void PLIB_PORTS_PinOpenDrainDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

PLIB_PORTS_PinOpenDrainEnable Function

Enables the open drain functionality for the selected pin.

File

plib_ports.h

C

```
void PLIB_PORTS_PinOpenDrainEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function enables the open drain functionality for the selected pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsOpenDrain in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable open drain for pin RC4
PLIB_PORTS_PinOpenDrainEnable(PORTS_ID_0, PORT_CHANNEL_C, PORTS_BIT_POS_4);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void PLIB_PORTS_PinOpenDrainEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

b) Port Functions

PLIB_PORTS_Clear Function

Clears the selected digital port/latch bits.

File

plib_ports.h

C

```
void PLIB_PORTS_Clear(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK clearMask);
```

Returns

None.

Description

This function clears the selected digital port/latch bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Clears the three least significant Port C bits
PLIB_PORTS_Clear(PORTS_ID_0, PORT_CHANNEL_C, 0x0007);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
clearMask	Identifies the bits to be cleared

Function

void PLIB_PORTS_Clear(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK clearMask)

PLIB_PORTS_DirectionGet Function

Reads the direction of the selected digital port.

File

plib_ports.h

C

```
PORTS_DATA_MASK PLIB_PORTS_DirectionGet(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

Direction of the selected port of type PORTS_DATA_MASK

Description

This function reads the direction of the selected digital port.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsDirection in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Reads the direction of Port C pins
PORTS_DATA_MASK readDir = PLIB_PORTS_DirectionGet(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.

Function

PORTS_DATA_MASK PLIB_PORTS_DirectionGet(PORTS_MODULE_ID index, PORTS_CHANNEL channel)

PLIB_PORTS_DirectionInputSet Function

Makes the selected pins direction input.

File

plib_ports.h

C

```
void PLIB_PORTS_DirectionInputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function makes the selected pins direction input.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsDirection in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make RC0, RC1 and RC2 pins as Input
PLIB_PORTS_DirectionInputSet(PORTS_ID_0, PORT_CHANNEL_C, 0x0007);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
mask	Identifies the pins direction that has to be made input

Function

void PLIB_PORTS_DirectionInputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask)

PLIB_PORTS_DirectionOutputSet Function

Makes the selected pins direction output.

File

plib_ports.h

C

```
void PLIB_PORTS_DirectionOutputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function makes the selected pins direction output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsDirection in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make RC0, RC1 and RC2 pins as Output
PLIB_PORTS_DirectionOutputSet(PORTS_ID_0, PORT_CHANNEL_C, 0x0007);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
mask	Identifies the pins direction that has to be made output

Function

void PLIB_PORTS_DirectionOutputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask)

PLIB_PORTS_Read Function

Reads the selected digital port.

File

plib_ports.h

C

```
PORTS_DATA_TYPE PLIB_PORTS_Read(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

data on a port with width PORTS_DATA_TYPE

Description

This function reads from the selected digital port.

Remarks

For reading the Latched data, PLIB_PORTS_ReadLatched function should be used.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsRead in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Read PORT C
PORTS_DATA_TYPE readData = PLIB_PORTS_Read(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.

Function

PORTS_DATA_TYPE PLIB_PORTS_Read(PORTS_MODULE_ID index, PORTS_CHANNEL channel)

PLIB_PORTS_Set Function

Sets the selected bits of the port.

File

plib_ports.h

C

```
void PLIB_PORTS_Set(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_TYPE value, PORTS_DATA_MASK
mask);
```

Returns

None.

Description

This function performs an 'AND' operation on the value and mask parameters, and then sets the bits in the port channel that were set by the result of the 'AND' operation.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// MY_VALUE - 0x1234
PORTS_DATA_MASK myMask = (PORTS_DATA_MASK)0x00FF;

// Set the PORT C bit positions 2,4 and 5 (0x0034 = b0000 0000 0011 0100)
PLIB_PORTS_Set(MY_PORTS_INSTANCE, PORT_CHANNEL_C, MY_VALUE, myMask);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
value	Consists of information about which port bit has to be set and which not
mask	Identifies the bits which could be intended for setting

Function

void PLIB_PORTS_Set(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_TYPE value,
PORTS_DATA_MASK mask)

PLIB_PORTS_Toggle Function

Toggles the selected digital port/latch.

File

plib_ports.h

C

```
void PLIB_PORTS_Toggle(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK toggleMask);
```

Returns

None.

Description

This function toggles the selected digital port/latch.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Toggles the three least significant Port C bits
PLIB_PORTS_Toggle(PORTS_ID_0, PORT_CHANNEL_C, 0x0007);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
toggleMask	Identifies the bits to be toggled

Function

void PLIB_PORTS_Toggle(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK toggleMask)

PLIB_PORTS_Write Function

Writes the selected digital port/latch.

File

plib_ports.h

C

```
void PLIB_PORTS_Write(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_TYPE value);
```

Returns

None.

Description

This function writes to the selected digital port/latch.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsWrite in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Write 0x12 into PORT C
PLIB_PORTS_Write(PORTS_ID_0, PORT_CHANNEL_C, 0x12);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

channel	Identifier for the Ports channel A, B, C, etc.
value	Value to be written into a port of width PORTS_DATA_TYPE

Function

void PLIB_PORTS_Write(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_TYPE value)

PLIB_PORTS_OpenDrainDisable Function

Disables the open drain functionality for the selected port.

File

plib_ports.h

C

```
void PLIB_PORTS_OpenDrainDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function disables the open drain functionality for the selected port.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsOpenDrain in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable Open Drain for RC0, RC1 and RC2 pins
PLIB_PORTS_OpenDrainDisable(PORTS_ID_0, PORT_CHANNEL_C, 0x0007);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
mask	Identifies the pins for the open drain to be disabled

Function

void PLIB_PORTS_OpenDrainDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask)

PLIB_PORTS_OpenDrainEnable Function

Enables the open drain functionality for the selected port pins.

File

plib_ports.h

C

```
void PLIB_PORTS_OpenDrainEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function enables the open drain functionality for the selected port pins.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPortsOpenDrain in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable Open Drain for RC0, RC1 and RC2 pins
PLIB_PORTS_OpenDrainEnable(PORTS_ID_0, PORT_CHANNEL_C, 0x0007);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.
mask	Identifies the pins for the open drain to be enabled

Function

void PLIB_PORTS_OpenDrainEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask)

PLIB_PORTS_ReadLatched Function

Reads and returns data from the selected Latch.

File

plib_ports.h

C

```
PORTS_DATA_TYPE PLIB_PORTS_ReadLatched(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

Latch read data.

Description

This function reads and returns the data from the selected Latch.

Remarks

For reading the Live data, PLIB_PORTS_Read function should be used.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsLatchRead in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Read latch C
PORTS_DATA_TYPE bitStatus = PLIB_PORTS_ReadLatched(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the Ports channel A, B, C, etc.

Function

```
  PORTS_DATA_TYPE PLIB_PORTS_ReadLatched
(
    PORTS_MODULE_ID index,
    PORTS_CHANNEL channel
)
```

PLIB_PORTS_ChannelModeSelect Function

Enables the selected channel pins as analog or digital.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelModeSelect(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK modeMask,
PORTS_PIN_MODE mode);
```

Returns

None.

Description

This function enables the selected channel pins as analog or digital.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_AnPinsModeSelect function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPinModePerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make pins RC5, RC8 and RC13 Analog
PLIB_PORTS_ChannelModeSelect(PORTS_ID_0, PORT_CHANNEL_C, 0x2120, PORTS_PIN_MODE_ANALOG);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
modeMask	Identifies the pins whose mode has to be modified. Modes of the pins whose corresponding bit is '1' get modified, mode of the other pins remains the same.
mode	Possible values of PORTS_PIN_MODE (Analog or Digital)

Function

void PLIB_PORTS_ChannelModeSelect

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_DATA_MASK modeMask,

PORTS_PIN_MODE mode

);

PLIB_PORTS_ChannelSlewRateSelect Function

Selects the slew rate for selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelSlewRateSelect(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK
channelMask, PORTS_PIN_SLEW_RATE slewRate);
```

Returns

None.

Description

This function selects the slew rate for selected channel pins.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsSlewRateControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make slew rate of pins RC5, RC8 and RC13 slowest
PLIB_PORTS_ChannelSlewRateSelect(PORTS_ID_0,
                                 PORT_CHANNEL_C,
                                 0x2120,
                                 PORTS_PIN_SLEW_RATE_SLOWEST);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
channelMask	Identifies the pins for which slew rate has to be modified. Slew rate of the pins which corresponding bit is "1" get modified, slew rate of the other pins remains the same.
slewRate	One of the possible values of PORTS_PIN_SLEW_RATE.

Function

void PLIB_PORTS_ChannelSlewRateSelect

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_DATA_MASK channelMask,

PORTS_PIN_SLEW_RATE slewRate

);

c) Peripheral Pin Select Functions

PLIB_PORTS_RemapInput Function

Input function remapping.

File

plib_ports.h

C

```
void PLIB_PORTS_RemapInput(PORTS_MODULE_ID index, PORTS_REMAP_INPUT_FUNCTION inputFunction,
PORTS_REMAP_INPUT_PIN remapInputPin);
```

Returns

None.

Description

This function controls the Input function remapping. It allows user to map any of the input functionality on any of the remappable input pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsRemapInputOutput in your application to determine whether this feature is available.

Preconditions

IOLOCK bit of configuration register should be clear to allow any remapping. PLIB_DEVCON_DeviceRegistersUnlock function can be used for that purpose. Refer DEVCON PLIB (or System Service) and the specific device data sheet to find more information.

Example

```
// System Unlock
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);
// Unlock PPS registers
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID_0, DEVCON_PPS_REGISTERS);
// Remapping input function 'Input Capture 1' to the Remappable pin 'RPD2'
PLIB_PORTS_RemapInput(PORTS_ID_0, INPUT_FUNC_IC1, INPUT_PIN_RPD2 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
inputFunction	One of the possible values of PORTS_REMAP_INPUT_FUNCTION
remapInputPin	One of the possible values of PORTS_REMAP_INPUT_PIN

Function

```
void PLIB_PORTS_RemapInput(  PORTS_MODULE_ID      index,
                             PORTS_REMAP_INPUT_FUNCTION inputFunction,
                             PORTS_REMAP_INPUT_PIN      remapInputPin );
```

PLIB_PORTS_RemapOutput Function

Output function remapping.

File

plib_ports.h

C

```
void PLIB_PORTS_RemapOutput(PORTS_MODULE_ID index, PORTS_REMAP_OUTPUT_FUNCTION outputFunction,
PORTS_REMAP_OUTPUT_PIN remapOutputPin);
```

Returns

None.

Description

This function controls the Output function remapping. it allows user to map any of the output functionality on any of the remappable output pin.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsRemapInputOutput in your application to determine whether this feature is available.

Preconditions

IOLOCK bit of configuration register should be clear to allow any remapping. PLIB_DEVCON_DeviceRegistersUnlock function can be used for that purpose. Refer DEVCON PLIB (or System Service) and the specific device data sheet to find more information.

Example

```
// System Unlock
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);
// Unlock PPS registers
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID_0, DEVCON_PPS_REGISTERS);
// Remapping output function 'UART3 Transmit' to the Remappable pin 'RPA14'
PLIB_PORTS_RemapOutput(PORTS_ID_0, OUTPUT_FUNC_U3TX, OUTPUT_PIN_RPA14);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
outputFunction	One of the possible values of PORTS_REMAP_OUTPUT_FUNCTION
remapOutputPin	One of the possible values of PORTS_REMAP_OUTPUT_PIN

Function

void PLIB_PORTS_RemapOutput(PORTS_MODULE_ID index,
PORTS_REMAP_OUTPUT_FUNCTION outputFunction,
PORTS_REMAP_OUTPUT_PIN remapOutputPin);

d) Change Notification Functions

PLIB_PORTS_ChangeNoticeDisable Function

Global Change Notice disable.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticeDisable(PORTS_MODULE_ID index);
```

Returns

None.

Description

This function disables the global Change Notice feature.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_ChangeNoticePerPortTurnOff function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNotice in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable Change Notification
PLIB_PORTS_ChangeNoticeDisable(PORTS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PORTS_ChangeNoticeDisable(PORTS_MODULE_ID index)

PLIB_PORTS_ChangeNoticeEnable Function

Global Change Notice enable.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticeEnable(PORTS_MODULE_ID index);
```

Returns

None.

Description

This function enables the global Change Notice feature.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_ChangeNoticePerPortTurnOn function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNotice in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable Change Notification
PLIB_PORTS_ChangeNoticeEnable(PORTS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PORTS_ChangeNoticeEnable(PORTS_MODULE_ID index)

PLIB_PORTS_ChangeNoticeInIdleDisable Function

CPU Idle halts the Change Notice operation.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticeInIdleDisable(PORTS_MODULE_ID index);
```

Returns

None.

Description

This function halts the Change Notice operation when the CPU enters Idle mode.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_ChangeNoticeInIdlePerPortDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Halts the Change notification operation when CPU enters Idle mode
PLIB_PORTS_ChangeNoticeInIdleDisable(PORTS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PORTS_ChangeNoticeInIdleDisable(PORTS_MODULE_ID index)

PLIB_PORTS_ChangeNoticeInIdleEnable Function

CPU Idle mode does not affect Change Notice operation.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticeInIdleEnable(PORTS_MODULE_ID index);
```

Returns

None.

Description

This function makes sure that Change Notice feature continues working in Idle mode.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_ChangeNoticeInIdlePerPortEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Change notification feature will be working even when CPU goes to
// Idle mode
PLIB_PORTS_ChangeNoticeInIdleEnable(PORTS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_PORTS_ChangeNoticeInIdleEnable(PORTS_MODULE_ID index)

PLIB_PORTS_ChangeNoticeInIdlePerPortDisable Function

Change Notification halts in Idle mode for selected channel.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticeInIdlePerPortDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

None.

Description

This function makes sure that change notification feature halts in Idle mode for the selected channel.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_ChangeNoticeInIdleDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePerPortInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Change notification halts in Idle mode for Port C
PLIB_PORTS_ChangeNoticeInIdlePerPortDisable(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel

Function

void PLIB_PORTS_ChangeNoticeInIdlePerPortDisable(PORTS_MODULE_ID index,
PORTS_CHANNEL channel);

PLIB_PORTS_ChangeNoticeInIdlePerPortEnable Function

Allows CN to be working in Idle mode for selected channel.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticeInIdlePerPortEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

None.

Description

This function makes sure that change notification feature keeps working in Idle mode for the selected channel.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_ChangeNoticeInIdleEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePerPortInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Change notification continues working in Idle mode for Port C
PLIB_PORTS_ChangeNoticeInIdlePerPortEnable(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel

Function

void PLIB_PORTS_ChangeNoticeInIdlePerPortEnable (PORTS_MODULE_ID index,
PORTS_CHANNEL channel);

PLIB_PORTS_ChangeNoticePerPortHasOccured Function

File

plib_ports.h

C

```
bool PLIB_PORTS_ChangeNoticePerPortHasOccured(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Description

This is function PLIB_PORTS_ChangeNoticePerPortHasOccured.

PLIB_PORTS_ChangeNoticePerPortTurnOff Function

Disables the change notification for selected port.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePerPortTurnOff(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

None.

Description

This function disables the change notification for selected port.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_ChangeNoticeDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePerPortTurnOn in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable Change notification for Port C
PLIB_PORTS_ChangeNoticePerPortTurnOff(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel

Function

void PLIB_PORTS_ChangeNoticePerPortTurnOff(PORTS_MODULE_ID index,
PORTS_CHANNEL channel);

PLIB_PORTS_ChangeNoticePerPortTurnOn Function

Enables the change notification for selected port.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePerPortTurnOn(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

None.

Description

This function enables the change notification for selected port.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_ChangeNoticeEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePerPortTurnOn in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable Change notification for Port C
PLIB_PORTS_ChangeNoticePerPortTurnOn(PORTS_ID_0, PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel

Function

void PLIB_PORTS_ChangeNoticePerPortTurnOn (PORTS_MODULE_ID index,
PORTS_CHANNEL channel);

PLIB_PORTS_ChangeNoticePullDownPerPortDisable Function

Disables the pull-down for selected Change Notice pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePullDownPerPortDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function disables the pull-down for selected Change Notice pins.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullDownPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable pull-down for RC5 pin
PLIB_PORTS_ChangeNoticePullDownPerPortDisable(PORTS_ID_0, PORT_CHANNEL_C,
                                                PORTS_BIT_POS_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

void PLIB_PORTS_ChangeNoticePullDownPerPortDisable(PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos)

PLIB_PORTS_ChangeNoticePullDownPerPortEnable Function

Enables the pull-down for selected Change Notice pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePullDownPerPortEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function enables the pull-down for selected Change Notice pins.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullDownPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable pull-down for RC5 pin
PLIB_PORTS_ChangeNoticePullDownPerPortEnable(PORTS_ID_0, PORT_CHANNEL_C,
                                                        PORTS_BIT_POS_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

void PLIB_PORTS_ChangeNoticePullDownPerPortEnable(PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos)

PLIB_PORTS_ChangeNoticePullUpDisable Function

Disable pull-up on input change.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePullUpDisable(PORTS_MODULE_ID index, PORTS_CHANGE_NOTICE_PIN pinNum);
```

Returns

None.

Description

This function disables pull-up on selected input change notification pin.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_ChangeNoticePullUpPerPortDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUp in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable pull-up on pin CN13
PLIB_PORTS_ChangeNoticePullUpDisable(PORTS_ID_0, CN13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Function

void PLIB_PORTS_ChangeNoticePullUpDisable(PORTS_MODULE_ID index,
PORTS_CHANGE_NOTICE_PIN pinNum)

PLIB_PORTS_ChangeNoticePullUpEnable Function

Enable pull-up on input change.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePullUpEnable(PORTS_MODULE_ID index, PORTS_CHANGE_NOTICE_PIN pinNum);
```

Returns

None.

Description

This function enables pull-up on selected input change notification pin

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_ChangeNoticePullUpPerPortEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUp in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable pull-up on pin CN13
PLIB_PORTS_ChangeNoticePullUpEnable(PORTS_ID_0, CN13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Function

void PLIB_PORTS_ChangeNoticePullUpEnable(PORTS_MODULE_ID index,
PORTS_CHANGE_NOTICE_PIN pinNum)

PLIB_PORTS_ChangeNoticePullUpPerPortDisable Function

Disables weak pull-up for the selected pin.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePullUpPerPortDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function disables weak pull-up for the selected port pin.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_ChangeNoticePullUpDisable function.

Pull-ups on change notification pins should always be disabled when the port pin is configured as a digital output.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUpPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable pull-up for RC5 pin
PLIB_PORTS_ChangeNoticePullUpPerPortDisable(PORTS_ID_0, PORT_CHANNEL_C,
                                                        PORTS_BIT_POS_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

void PLIB_PORTS_ChangeNoticePullUpPerPortDisable(PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);

PLIB_PORTS_ChangeNoticePullUpPerPortEnable Function

Enables the pull-up for selected Change Notice pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChangeNoticePullUpPerPortEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);
```

Returns

None.

Description

This function enables the pull-up for selected Change Notice pins.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_ChangeNoticePullUpEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUpPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable pull-up for RC5 pin
PLIB_PORTS_ChangeNoticePullUpPerPortEnable(PORTS_ID_0, PORT_CHANNEL_C,
                                                        PORTS_BIT_POS_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

void PLIB_PORTS_ChangeNoticePullUpPerPortEnable (PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos)

PLIB_PORTS_PinChangeNoticeDisable Function

Port pin Change Notice disable.

File

plib_ports.h

C

```
void PLIB_PORTS_PinChangeNoticeDisable(PORTS_MODULE_ID index, PORTS_CHANGE_NOTICE_PIN pinNum);
```

Returns

None.

Description

This function disables the port pin Change Notice feature.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_PinChangeNoticePerPortDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPinChangeNotice in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable Change Notice interrupt for pin CN13
PLIB_PORTS_PinChangeNoticeDisable(PORTS_ID_0, CN13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Function

void PLIB_PORTS_PinChangeNoticeDisable(PORTS_MODULE_ID index,
PORTS_CHANGE_NOTICE_PIN pinNum)

PLIB_PORTS_PinChangeNoticeEnable Function

Port pin Change Notice interrupt enable.

File

plib_ports.h

C

```
void PLIB_PORTS_PinChangeNoticeEnable(PORTS_MODULE_ID index, PORTS_CHANGE_NOTICE_PIN pinNum);
```

Returns

None.

Description

This function enables the port pin Change Notice feature.

Remarks

This function is only available in devices without PPS. For PPS devices, use the PLIB_PORTS_PinChangeNoticePerPortEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_PORTS_ExistsPinChangeNotice in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable Change Notice interrupt for pin CN13
PLIB_PORTS_PinChangeNoticeEnable(PORTS_ID_0, CN13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Function

void PLIB_PORTS_PinChangeNoticeEnable(PORTS_MODULE_ID index,
PORTS_CHANGE_NOTICE_PIN pinNum)

PLIB_PORTS_PinChangeNoticePerPortDisable Function

Disables CN interrupt for the selected pin.

File

plib_ports.h

C

```
void PLIB_PORTS_PinChangeNoticePerPortDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);
```

Returns

None.

Description

This function disables Change Notice interrupt for the selected port pin.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_PinChangeNoticeDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeIntPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable CN interrupt for RC5 pin
PLIB_PORTS_PinChangeNoticePerPortDisable(PORTS_ID_0, PORT_CHANNEL_C,
                                                    PORTS_BIT_POS_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

void PLIB_PORTS_PinChangeNoticePerPortDisable(PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos)

PLIB_PORTS_PinChangeNoticePerPortEnable Function

Enables CN interrupt for the selected pin.

File

plib_ports.h

C

```
void PLIB_PORTS_PinChangeNoticePerPortEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);
```

Returns

None.

Description

This function enables Change Notice interrupt for the selected port pin.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_PinChangeNoticeEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeIntPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable CN interrupt for RC5 pin
PLIB_PORTS_PinChangeNoticePerPortEnable(PORTS_ID_0, PORT_CHANNEL_C,
                                                    PORTS_BIT_POS_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

void PLIB_PORTS_PinChangeNoticePerPortEnable (PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos)

PLIB_PORTS_ChangeNoticePerPortHasOccurred Function

checks the status of change on the pin

File

plib_ports.h

C

```
bool PLIB_PORTS_ChangeNoticePerPortHasOccurred(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);
```

Returns

None.

Description

This function checks if the change has occurred on the given pin or not.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePerPortStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_PORTS_ChangeNoticePerPortHasOccurred( PORTS_ID_0,
                                PORT_CHANNEL_C, PORTS_BIT_POS_4 ) == True)
{
 //do something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
bitPos	Position in the PORT pins

Function

bool PLIB_PORTS_ChangeNoticePerPortHasOccurred (PORTS_MODULE_ID index,
PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);

PLIB_PORTS_ChannelChangeNoticeDisable Function

Disables CN interrupt for the selected pins of a channel.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticeDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK
mask);
```

Returns

None.

Description

This function Disables Change Notice interrupt for the selected port pins of a channel.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_CnPinsDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeIntPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable CN interrupt for RC5, RC8 and RC13 pins
PLIB_PORTS_ChannelChangeNoticeDisable(PORTS_ID_0, PORT_CHANNEL_C, 0x2120);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
mask	Identifies the pins for which change notification is to be disabled

Function

void PLIB_PORTS_ChannelChangeNoticeDisable
(
PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_DATA_MASK mask
);

PLIB_PORTS_ChannelChangeNoticeEnable Function

Enables CN interrupt for the selected pins of a channel.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticeEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK
mask);
```

Returns

None.

Description

This function enables Change Notice interrupt for the selected port pins of a channel.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_CnPinsEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeIntPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable CN interrupt for RC5, RC8 and RC13 pins
PLIB_PORTS_ChannelChangeNoticeEnable(PORTS_ID_0, PORT_CHANNEL_C, 0x2120);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
mask	Identifies the pins for which change notification is to be enabled

Function

void PLIB_PORTS_ChannelChangeNoticeEnable

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_DATA_MASK mask

);

PLIB_PORTS_ChannelChangeNoticePullDownDisable Function

Disables Change Notice pull-down for the selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticePullDownDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function Disables the Change Notice pull-down for the selected channel pins.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullDownPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable pull-down for RC5, RC8 and RC13 pins
PLIB_PORTS_ChannelChangeNoticePullDownDisable(PORTS_ID_0, PORT_CHANNEL_C,
                                                0x2120);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
mask	Identifies the pins for the pull-down to be disabled

Function

void PLIB_PORTS_ChannelChangeNoticePullDownDisable

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_DATA_MASK mask

);

PLIB_PORTS_ChannelChangeNoticePullDownEnable Function

Enables Change Notice pull-down for the selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticePullDownEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function enables the Change Notice pull-down for the selected channel pins.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullDownPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable pull-down for RC5, RC8 and RC13 pins
PLIB_PORTS_ChannelChangeNoticePullDownEnable(PORTS_ID_0, PORT_CHANNEL_C,
                                                0x2120);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
mask	Identifies the pins for the pull-down to be enabled

Function

void PLIB_PORTS_ChannelChangeNoticePullDownEnable
(
PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_DATA_MASK mask
);

PLIB_PORTS_ChannelChangeNoticePullUpDisable Function

Disables Change Notice pull-up for the selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticePullUpDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function Disables the Change Notice pull-up for the selected channel pins.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_CnPinsPullUpDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUpPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable pull-up for RC5, RC8 and RC13 pins
PLIB_PORTS_ChannelChangeNoticePullUpDisable(PORTS_ID_0, PORT_CHANNEL_C,
                                                        0x2120);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
mask	Identifies the pins of the pull-up to be disabled

Function

void PLIB_PORTS_ChannelChangeNoticePullUpDisable
(
PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_DATA_MASK mask
);

PLIB_PORTS_ChannelChangeNoticePullUpEnable Function

Enables Change Notice pull-up for the selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticePullUpEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function enables the Change Notice pull-up for the selected channel pins.

Remarks

This function is only available in devices with PPS. For Non-PPS devices, use the PLIB_PORTS_CnPinsPullUpEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUpPerPort in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable pull-up for RC5, RC8 and RC13 pins
PLIB_PORTS_ChannelChangeNoticePullUpEnable(PORTS_ID_0, PORT_CHANNEL_C,
                                                       0x2120);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
mask	Identifies the pins of the pull-up to be enabled

Function

void PLIB_PORTS_ChannelChangeNoticePullUpEnable

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_DATA_MASK mask

);

PLIB_PORTS_ChannelChangeNoticeEdgeDisable Function

Disables selected type of edge for selected CN pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticeEdgeDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK edgeRisingMask, PORTS_DATA_MASK edgeFallingMask);
```

Returns

None.

Description

This function Disables selected type of edge (falling or rising) for selected CN pins of a port channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeEdgeControl in your application to determine whether this feature is available.

Preconditions

Change Notice method should be selected as "PORTS_CHANGE_NOTICE_METHOD_EDGE_DETECT" using PLIB_PORTS_ChannelChangeNoticeMethodSelect before using this function.

Example

```
// Disable Change Notice at rising edge for RC1 pin and at falling edge for
// RC1 & RC5 pins.
PLIB_PORTS_ChannelChangeNoticeEdgeDisable(PORTS_ID_0, PORT_CHANNEL_C, 0x0002, 0x0022);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
edgeRisingMask	Identifies the pins for which Change Notice has to be enabled for rising edge. Change notice interrupt at rising edge is enabled for the pins which corresponding bit is '1', for the other pins it remains the same.
edgeFallingMask	Identifies the pins for which Change Notice has to be enabled for falling edge. Change notice interrupt at falling edge is enabled for the pins which corresponding bit is '1', for the other pins it remains the same.

Function

void PLIB_PORTS_ChannelChangeNoticeEdgeDisable

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_DATA_MASK edgeRisingMask,

PORTS_DATA_MASK edgeFallingMask

);

PLIB_PORTS_ChannelChangeNoticeEdgeEnable Function

Enables selected type of edge for selected CN pins.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticeEdgeEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK
edgeRisingMask, PORTS_DATA_MASK edgeFallingMask);
```

Returns

None.

Description

This function Enables selected type of edge (falling or rising) for selected CN pins of a port channel.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeEdgeControl in your application to determine whether this feature is available.

Preconditions

Change Notice method should be selected as "PORTS_CHANGE_NOTICE_METHOD_EDGE_DETECT" using PLIB_PORTS_ChannelChangeNoticeMethodSelect before using this function.

Example

```
// Enable Change Notice at rising edge for RC1 pin and at falling edge for
// RC1 & RC5 pins.
PLIB_PORTS_ChannelChangeNoticeEdgeEnable(PORTS_ID_0, PORT_CHANNEL_C, 0x0002, 0x0022);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel

edgeRisingMask	Identifies the pins for which Change Notice has to be enabled for rising edge. Change Notice interrupt at rising edge is enabled for the pins which corresponding bit is '1', for the other pins it remains the same.
edgeFallingMask	Identifies the pins for which Change Notice has to be enabled for falling edge. Change Notice interrupt at falling edge is enabled for the pins which corresponding bit is '1', for the other pins it remains the same.

Function

void PLIB_PORTS_ChannelChangeNoticeEdgeEnable
(
PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_DATA_MASK edgeRisingMask,
PORTS_DATA_MASK edgeFallingMask
);

PLIB_PORTS_ChannelChangeNoticeMethodGet Function

Gets the Change Notice style for the selected port channel.

File

plib_ports.h

C

```
PORTS_CHANGE_NOTICE_METHOD PLIB_PORTS_ChannelChangeNoticeMethodGet(PORTS_MODULE_ID index, PORTS_CHANNEL
channel);
```

Returns

One of the possible values of PORTS_CHANGE_NOTICE_METHOD.

Description

This function gets the Change Notice style (or method) for the selected port channel.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticeMethod in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PORTS_CHANGE_NOTICE_METHOD changeNoticeMethod;
changeNoticeMethod = PLIB_PORTS_ChannelChangeNoticeMethodGet(PORTS_ID_0,
PORT_CHANNEL_C);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel

Function

PORTS_CHANGE_NOTICE_METHOD PLIB_PORTS_ChannelChangeNoticeMethodGet
(
PORTS_MODULE_ID index,
PORTS_CHANNEL channel
);

PLIB_PORTS_ChannelChangeNoticeMethodSelect Function

Selects the Change Notice style for selected port channel.

File

plib_ports.h

C

```
void PLIB_PORTS_ChannelChangeNoticeMethodSelect(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_CHANGE_NOTICE_METHOD changeNoticeMethod);
```

Returns

None.

Description

This function selects the Change Notice style (or method) for selected port channel. It allows user to select whether the Change Notice detection will happen based on edge transition or level transition on all the CN pins of a particular channel.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChannelChangeNoticeMethod in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_PORTS_ChannelChangeNoticeMethodSelect(PORTS_ID_0, PORT_CHANNEL_C,
                                           PORTS_CHANGE_NOTICE_METHOD_EDGE_DETECT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Port pin channel
changeNoticeMethod	One of the possible values of PORTS_CHANGE_NOTICE_METHOD.

Function

void PLIB_PORTS_ChannelChangeNoticeMethodSelect

(

PORTS_MODULE_ID index,

PORTS_CHANNEL channel,

PORTS_CHANGE_NOTICE_METHOD changeNoticeMethod

);

PLIB_PORTS_AnPinsModeSelect Function

Enables the selected AN pins as analog or digital.

File

plib_ports.h

C

```
void PLIB_PORTS_AnPinsModeSelect(PORTS_MODULE_ID index, PORTS_AN_PIN anPins, PORTS_PIN_MODE mode);
```

Returns

None.

Description

This function enables the selected AN pins as analog or digital.

Remarks

This function is only available in devices without PPS feature. For PPS devices, use the PLIB_PORTS_ChannelModeSelect function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsAnPinsMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Make pins AN5, AN8 and AN13 Analog
PLIB_PORTS_AnPinsModeSelect(PORTS_ID_0, PORTS_AN_PIN_5 |
                            PORTS_AN_PIN_8 |
                            PORTS_AN_PIN_13,
                            PORTS_PIN_MODE_ANALOG);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
anPins	AN pins whose mode is to be changed. Multiple AN pins can be ORed.
mode	Possible values of PORTS_PIN_MODE (Analog or Digital)

Function

void PLIB_PORTS_AnPinsModeSelect

(

PORTS_MODULE_ID index,

PORTS_AN_PIN anPins,

PORTS_PIN_MODE mode

);

PLIB_PORTS_CnPinsDisable Function

Disables CN interrupt for the selected pins of a channel.

File

plib_ports.h

C

```
void PLIB_PORTS_CnPinsDisable(PORTS_MODULE_ID index, PORTS_CN_PIN cnPins);
```

Returns

None.

Description

This function Disables Change Notice interrupt for the selected port pins of a channel.

Remarks

This function is only available in devices without PPS feature. For PPS devices, use the PLIB_PORTS_ChannelChangeNoticeDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPinChangeNotice in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable CN interrupt for CN5, CN8 and CN13 pins
PLIB_PORTS_CnPinsDisable(PORTS_ID_0,
                         CHANGE_NOTICE_PIN_5 |
                         CHANGE_NOTICE_PIN_8 |
                         CHANGE_NOTICE_PIN_13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cnPins	CN pins to be disabled. Multiple CN pins can be ORed.

Function

void PLIB_PORTS_CnPinsDisable
(
PORTS_MODULE_ID index,
PORTS_CN_PIN cnPins
);

PLIB_PORTS_CnPinsEnable Function

Enables CN interrupt for the selected pins of a channel.

File

plib_ports.h

C

```
void PLIB_PORTS_CnPinsEnable(PORTS_MODULE_ID index, PORTS_CN_PIN cnPins);
```

Returns

None.

Description

This function enables Change Notice interrupt for the selected port pins of a channel.

Remarks

This function is only available in devices without PPS feature. For PPS devices, use the PLIB_PORTS_ChannelChangeNoticeEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsPinChangeNotice in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable CN interrupt for CN5, CN8 and CN13 pins
PLIB_PORTS_CnPinsEnable(PORTS_ID_0,
                        CHANGE_NOTICE_PIN_5 |
                        CHANGE_NOTICE_PIN_8 |
                        CHANGE_NOTICE_PIN_13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cnPins	CN pins to be enabled. Multiple CN pins can be ORed.

Function

void PLIB_PORTS_CnPinsEnable
(
PORTS_MODULE_ID index,
PORTS_CN_PIN cnPins
);

PLIB_PORTS_CnPinsPullUpDisable Function

Disables Change Notice pull-up for the selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_CnPinsPullUpDisable(PORTS_MODULE_ID index, PORTS_CN_PIN cnPins);
```

Returns

None.

Description

This function Disables the Change Notice pull-up for the selected channel pins.

Remarks

This function is only available in devices without PPS feature. For PPS devices, use the PLIB_PORTS_ChannelChangeNoticePullUpDisable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUp in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Disable pull-up for CN5, CN8 and CN13 pins
PLIB_PORTS_CnPinsPullUpDisable(PORTS_ID_0,
                               CHANGE_NOTICE_PIN_5 |
                               CHANGE_NOTICE_PIN_8 |
                               CHANGE_NOTICE_PIN_13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cnPins	CN pins whose pull-up is to be disabled. Multiple CN pins can be ORed.

Function

void PLIB_PORTS_CnPinsPullUpDisable

(

PORTS_MODULE_ID index,

PORTS_CN_PIN cnPins

);

PLIB_PORTS_CnPinsPullUpEnable Function

Enables Change Notice pull-up for the selected channel pins.

File

plib_ports.h

C

```
void PLIB_PORTS_CnPinsPullUpEnable(PORTS_MODULE_ID index, PORTS_CN_PIN cnPins);
```

Returns

None.

Description

This function enables the Change Notice pull-up for the selected channel pins.

Remarks

This function is only available in devices without PPS feature. For PPS devices, use the PLIB_PORTS_ChannelChangeNoticePullUpEnable function.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_PORTS_ExistsChangeNoticePullUp in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Enable pull-up for CN5, CN8 and CN13 pins
```

```
PLIB_PORTS_CnPinsPullUpEnable(PORTS_ID_0, CHANGE_NOTICE_PIN_5 |
                                          CHANGE_NOTICE_PIN_8 |
                                          CHANGE_NOTICE_PIN_13);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
cnPins	CN pins whose pull-up is to be enabled. Multiple CN pins can be ORed.

Function

void PLIB_PORTS_CnPinsPullUpEnable
(
PORTS_MODULE_ID index,
PORTS_CN_PIN cnPins
);

e) Feature Existence Functions

PLIB_PORTS_ExistsChangeNotice Function

Identifies whether the ChangeNotice feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNotice(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNotice feature is supported on the device
- false - The ChangeNotice feature is not supported on the device

Description

This function identifies whether the ChangeNotice feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticeEnable
- PLIB_PORTS_ChangeNoticeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNotice(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticeInIdle Function

Identifies whether the ChangeNoticeInIdle feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticeInIdle(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticeInIdle feature is supported on the device
- false - The ChangeNoticeInIdle feature is not supported on the device

Description

This function identifies whether the ChangeNoticeInIdle feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticeInIdleEnable
- PLIB_PORTS_ChangeNoticeInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticeInIdle(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticePerPortInIdle Function

Identifies whether the ChangeNoticeInIdlePerPort feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticePerPortInIdle(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticeInIdlePerPort feature is supported on the device
- false - The ChangeNoticeInIdlePerPort feature is not supported on the device

Description

This function identifies whether the ChangeNoticeInIdlePerPort feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticeInIdlePerPortEnable
- PLIB_PORTS_ChangeNoticeInIdlePerPortDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticePerPortInIdle(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticePerPortStatus Function

Identifies whether the ChangeNoticePerPortStatus feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticePerPortStatus(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticePerPortStatus feature is supported on the device
- false - The ChangeNoticePerPortStatus feature is not supported on the device

Description

This function identifies whether the ChangeNoticePerPortStatus feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticePerPortHasOccured
- PLIB_PORTS_ChangeNoticePerPortHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticePerPortStatus(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticePerPortTurnOn Function

Identifies whether the ChangeNoticePerPortTurnOn feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticePerPortTurnOn(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticePerPortTurnOn feature is supported on the device
- false - The ChangeNoticePerPortTurnOn feature is not supported on the device

Description

This function identifies whether the ChangeNoticePerPortTurnOn feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticePerPortTurnOn
- PLIB_PORTS_ChangeNoticePerPortTurnOff

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticePerPortTurnOn(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticePullDownPerPort Function

Identifies whether the ChangeNoticePullDownPerPort feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticePullDownPerPort(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticePullDownPerPort feature is supported on the device
- false - The ChangeNoticePullDownPerPort feature is not supported on the device

Description

This function identifies whether the ChangeNoticePullDownPerPort feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticePullDownPerPortEnable
- PLIB_PORTS_ChangeNoticePullDownPerPortDisable
- PLIB_PORTS_ChannelChangeNoticePullDownEnable
- PLIB_PORTS_ChannelChangeNoticePullDownDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticePullDownPerPort(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticePullUp Function

Identifies whether the ChangeNoticePullup feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticePullUp(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticePullup feature is supported on the device
- false - The ChangeNoticePullup feature is not supported on the device

Description

This function identifies whether the ChangeNoticePullup feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticePullUpEnable

- PLIB_PORTS_ChangeNoticePullUpDisable
- PLIB_PORTS_CnPinsPullUpEnable
- PLIB_PORTS_CnPinsPullUpDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticePullUp(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticePullUpPerPort Function

Identifies whether the ChangeNoticePullUpPerPort feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticePullUpPerPort(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticePullUpPerPort feature is supported on the device
- false - The ChangeNoticePullUpPerPort feature is not supported on the device

Description

This function identifies whether the ChangeNoticePullUpPerPort feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChangeNoticePullUpPerPortEnable
- PLIB_PORTS_ChangeNoticePullUpPerPortDisable
- PLIB_PORTS_ChannelChangeNoticePullUpEnable
- PLIB_PORTS_ChannelChangeNoticePullUpDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticePullUpPerPort(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPinChangeNotice Function

Identifies whether the PinChangeNotice feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPinChangeNotice(PORTS_MODULE_ID index);
```

Returns

- true - The PinChangeNotice feature is supported on the device
- false - The PinChangeNotice feature is not supported on the device

Description

This function identifies whether the PinChangeNotice feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinChangeNoticeEnable
- PLIB_PORTS_PinChangeNoticeDisable
- PLIB_PORTS_CnPinsEnable
- PLIB_PORTS_CnPinsDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPinChangeNotice(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPinChangeNoticePerPort Function

Identifies whether the PinChangeNoticePerPort feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPinChangeNoticePerPort(PORTS_MODULE_ID index);
```

Returns

- true - The PinChangeNoticePerPort feature is supported on the device
- false - The PinChangeNoticePerPort feature is not supported on the device

Description

This function identifies whether the PinChangeNoticePerPort feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinChangeNoticePerPortEnable
- PLIB_PORTS_PinChangeNoticePerPortDisable
- PLIB_PORTS_ChannelChangeNoticeEnable
- PLIB_PORTS_ChannelChangeNoticeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPinChangeNoticePerPort(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPinMode Function

Identifies whether the PinMode feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPinMode(PORTS_MODULE_ID index);
```

Returns

- true - The PinMode feature is supported on the device
- false - The PinMode feature is not supported on the device

Description

This function identifies whether the PinMode (Analog Pin or Digital Pin) feature is available on the Ports module. When this function returns true, this function is supported on the device:

- PLIB_PORTS_PinModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPinMode(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPinModePerPort Function

Identifies whether the PinModePerPort feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPinModePerPort(PORTS_MODULE_ID index);
```

Returns

- true - The PinModePerPort feature is supported on the device
- false - The PinModePerPort feature is not supported on the device

Description

This function identifies whether the PinModePerPort (Analog Pin or Digital Pin) feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinModePerPortSelect
- PLIB_PORTS_ChannelModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPinModePerPort(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPortsDirection Function

Identifies whether the PortsDirection feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPortsDirection(PORTS_MODULE_ID index);
```

Returns

- true - The PortsDirection feature is supported on the device
- false - The PortsDirection feature is not supported on the device

Description

This function identifies whether the PortsDirection feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinDirectionInputSet
- PLIB_PORTS_PinDirectionOutputSet
- PLIB_PORTS_DirectionInputSet
- PLIB_PORTS_DirectionOutputSet
- PLIB_PORTS_DirectionGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPortsDirection(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPortsOpenDrain Function

Identifies whether the PortsOpenDrain feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPortsOpenDrain(PORTS_MODULE_ID index);
```

Returns

- true - The PortsOpenDrain feature is supported on the device
- false - The PortsOpenDrain feature is not supported on the device

Description

This function identifies whether the PortsOpenDrain feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinOpenDrainEnable
- PLIB_PORTS_PinOpenDrainDisable
- PLIB_PORTS_OpenDrainEnable
- PLIB_PORTS_OpenDrainDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPortsOpenDrain(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPortsRead Function

Identifies whether the PortsRead feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPortsRead(PORTS_MODULE_ID index);
```

Returns

- true - The PortsRead feature is supported on the device
- false - The PortsRead feature is not supported on the device

Description

This function identifies whether the PortsRead feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinGet
- PLIB_PORTS_Read

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPortsRead(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsPortsWrite Function

Identifies whether the PortsWrite feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsPortsWrite(PORTS_MODULE_ID index);
```

Returns

- true - The PortsWrite feature is supported on the device
- false - The PortsWrite feature is not supported on the device

Description

This function identifies whether the PortsWrite feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinWrite
- PLIB_PORTS_PinSet
- PLIB_PORTS_PinClear
- PLIB_PORTS_PinToggle
- PLIB_PORTS_Write
- PLIB_PORTS_Set
- PLIB_PORTS_Toggle
- PLIB_PORTS_Clear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsPortsWrite(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsRemapInput Function

Identifies whether the RemapInput feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsRemapInput(PORTS_MODULE_ID index);
```

Returns

- true - The RemapInput feature is supported on the device
- false - The RemapInput feature is not supported on the device

Description

This function identifies whether the RemapInput feature is available on the Ports module. When this function returns true, this function is supported on the device:

- PLIB_PORTS_RemapInput

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsRemapInput(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsRemapOutput Function

Identifies whether the RemapOutput feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsRemapOutput(PORTS_MODULE_ID index);
```

Returns

- true - The RemapOutput feature is supported on the device
- false - The RemapOutput feature is not supported on the device

Description

This function identifies whether the RemapOutput feature is available on the Ports module. When this function returns true, this function is supported on the device:

- PLIB_PORTS_RemapOutput

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsRemapOutput(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsLatchRead Function

Identifies whether the LatchRead feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsLatchRead(PORTS_MODULE_ID index);
```

Returns

- true - The LatchRead feature is supported on the device
- false - The LatchRead feature is not supported on the device

Description

This function identifies whether the LatchRead feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_PinGetLatched
- PLIB_PORTS_ReadLatched

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsLatchRead(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsAnPinsMode Function

Identifies whether the AnPinsMode feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsAnPinsMode(PORTS_MODULE_ID index);
```

Returns

- true - The AnPinsMode feature is supported on the device
- false - The AnPinsMode feature is not supported on the device

Description

This function identifies whether the AnPinsMode feature is available on the Ports module. When this function returns true, this function is supported on the device:

- PLIB_PORTS_AnPinsModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsAnPinsMode(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticeEdgeControl Function

Identifies whether the ChangeNoticeEdgeControl feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticeEdgeControl(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticeEdgeControl feature is supported on the device
- false - The ChangeNoticeEdgeControl feature is not supported on the device

Description

This function identifies whether the ChangeNoticeEdgeControl feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChannelChangeNoticeEdgeEnable
- PLIB_PORTS_ChannelChangeNoticeEdgeDisable
- PLIB_PORTS_PinChangeNoticeEdgeIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticeEdgeControl(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChangeNoticeEdgeStatus Function

Identifies whether the ChangeNoticeEdgeStatus feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChangeNoticeEdgeStatus(PORTS_MODULE_ID index);
```

Returns

- true - The ChangeNoticeEdgeStatus feature is supported on the device
- false - The ChangeNoticeEdgeStatus feature is not supported on the device

Description

This function identifies whether the ChangeNoticeEdgeStatus feature is available on the Ports module. When this function returns true, this function is supported on the device:

- PLIB_PORTS_PinChangeNoticeEdgeHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChangeNoticeEdgeStatus(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsChannelChangeNoticeMethod Function

Identifies whether the ChannelChangeNoticeMethod feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsChannelChangeNoticeMethod(PORTS_MODULE_ID index);
```

Returns

- true - The ChannelChangeNoticeMethod feature is supported on the device
- false - The ChannelChangeNoticeMethod feature is not supported on the device

Description

This function identifies whether the ChannelChangeNoticeMethod feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChannelChangeNoticeMethodSelect
- PLIB_PORTS_ChannelChangeNoticeMethodGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsChannelChangeNoticeMethod(PORTS_MODULE_ID index)

PLIB_PORTS_ExistsSlewRateControl Function

Identifies whether the SlewRateControl feature exists on the Ports module.

File

plib_ports.h

C

```
bool PLIB_PORTS_ExistsSlewRateControl(PORTS_MODULE_ID index);
```

Returns

- true - The SlewRateControl feature is supported on the device
- false - The SlewRateControl feature is not supported on the device

Description

This function identifies whether the SlewRateControl feature is available on the Ports module. When this function returns true, these functions are supported on the device:

- PLIB_PORTS_ChannelSlewRateSelect
- PLIB_PORTS_PinSlewRateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_PORTS_ExistsSlewRateControl(PORTS_MODULE_ID index)

f) Data Types and Constants

PORTS_ANALOG_PIN Enumeration

Data type defining the different analog input pins.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_ANALOG_PIN_0,
  PORTS_ANALOG_PIN_1,
  PORTS_ANALOG_PIN_2,
  PORTS_ANALOG_PIN_3,
  PORTS_ANALOG_PIN_4,
  PORTS_ANALOG_PIN_5,
  PORTS_ANALOG_PIN_6,
  PORTS_ANALOG_PIN_7,
  PORTS_ANALOG_PIN_8,
  PORTS_ANALOG_PIN_9,
  PORTS_ANALOG_PIN_10,
  PORTS_ANALOG_PIN_11,
  PORTS_ANALOG_PIN_12,
  PORTS_ANALOG_PIN_13,
  PORTS_ANALOG_PIN_14,
  PORTS_ANALOG_PIN_15,
  PORTS_ANALOG_PIN_16,
  PORTS_ANALOG_PIN_17,
  PORTS_ANALOG_PIN_18,
  PORTS_ANALOG_PIN_19,
  PORTS_ANALOG_PIN_20,
  PORTS_ANALOG_PIN_21,
  PORTS_ANALOG_PIN_22,
  PORTS_ANALOG_PIN_23,
  PORTS_ANALOG_PIN_24,
  PORTS_ANALOG_PIN_25,
  PORTS_ANALOG_PIN_26,
  PORTS_ANALOG_PIN_27,
  PORTS_ANALOG_PIN_28,
  PORTS_ANALOG_PIN_29,
  PORTS_ANALOG_PIN_30,
  PORTS_ANALOG_PIN_31,
  PORTS_ANALOG_PIN_32,
  PORTS_ANALOG_PIN_33,
  PORTS_ANALOG_PIN_34,
  PORTS_ANALOG_PIN_35,
  PORTS_ANALOG_PIN_36,
  PORTS_ANALOG_PIN_37,
  PORTS_ANALOG_PIN_38,
  PORTS_ANALOG_PIN_39,
  PORTS_ANALOG_PIN_40,
  PORTS_ANALOG_PIN_41,
  PORTS_ANALOG_PIN_42,
  PORTS_ANALOG_PIN_43,
  PORTS_ANALOG_PIN_44,
  PORTS_ANALOG_PIN_45,
  PORTS_ANALOG_PIN_46,
  PORTS_ANALOG_PIN_47,
  PORTS_ANALOG_PIN_48,
  PORTS_ANALOG_PIN_49
} PORTS_ANALOG_PIN;
```

Members

Members	Description
PORTS_ANALOG_PIN_0	Analog Input Pin 0
PORTS_ANALOG_PIN_1	Analog Input Pin 1
PORTS_ANALOG_PIN_2	Analog Input Pin 2
PORTS_ANALOG_PIN_3	Analog Input Pin 3
PORTS_ANALOG_PIN_4	Analog Input Pin 4
PORTS_ANALOG_PIN_5	Analog Input Pin 5
PORTS_ANALOG_PIN_6	Analog Input Pin 6
PORTS_ANALOG_PIN_7	Analog Input Pin 7
PORTS_ANALOG_PIN_8	Analog Input Pin 8

PORTS_ANALOG_PIN_9	Analog Input Pin 9
PORTS_ANALOG_PIN_10	Analog Input Pin 10
PORTS_ANALOG_PIN_11	Analog Input Pin 11
PORTS_ANALOG_PIN_12	Analog Input Pin 12
PORTS_ANALOG_PIN_13	Analog Input Pin 13
PORTS_ANALOG_PIN_14	Analog Input Pin 14
PORTS_ANALOG_PIN_15	Analog Input Pin 15
PORTS_ANALOG_PIN_16	Analog Input Pin 16
PORTS_ANALOG_PIN_17	Analog Input Pin 17
PORTS_ANALOG_PIN_18	Analog Input Pin 18
PORTS_ANALOG_PIN_19	Analog Input Pin 19
PORTS_ANALOG_PIN_20	Analog Input Pin 20
PORTS_ANALOG_PIN_21	Analog Input Pin 21
PORTS_ANALOG_PIN_22	Analog Input Pin 22
PORTS_ANALOG_PIN_23	Analog Input Pin 23
PORTS_ANALOG_PIN_24	Analog Input Pin 24
PORTS_ANALOG_PIN_25	Analog Input Pin 25
PORTS_ANALOG_PIN_26	Analog Input Pin 26
PORTS_ANALOG_PIN_27	Analog Input Pin 27
PORTS_ANALOG_PIN_28	Analog Input Pin 28
PORTS_ANALOG_PIN_29	Analog Input Pin 29
PORTS_ANALOG_PIN_30	Analog Input Pin 30
PORTS_ANALOG_PIN_31	Analog Input Pin 31
PORTS_ANALOG_PIN_32	Analog Input Pin 32
PORTS_ANALOG_PIN_33	Analog Input Pin 33
PORTS_ANALOG_PIN_34	Analog Input Pin 34
PORTS_ANALOG_PIN_35	Analog Input Pin 35
PORTS_ANALOG_PIN_36	Analog Input Pin 36
PORTS_ANALOG_PIN_37	Analog Input Pin 37
PORTS_ANALOG_PIN_38	Analog Input Pin 38
PORTS_ANALOG_PIN_39	Analog Input Pin 39
PORTS_ANALOG_PIN_40	Analog Input Pin 40
PORTS_ANALOG_PIN_41	Analog Input Pin 41
PORTS_ANALOG_PIN_42	Analog Input Pin 42
PORTS_ANALOG_PIN_43	Analog Input Pin 43
PORTS_ANALOG_PIN_44	Analog Input Pin 44
PORTS_ANALOG_PIN_45	Analog Input Pin 45
PORTS_ANALOG_PIN_46	Analog Input Pin 46
PORTS_ANALOG_PIN_47	Analog Input Pin 47
PORTS_ANALOG_PIN_48	Analog Input Pin 48
PORTS_ANALOG_PIN_49	Analog Input Pin 49

Description

PORTS Analog input pins

This data type defines the different analog input pins.

Remarks

Values of these AN Pin enums are different from the other similar enumerators by name PORTS_AN_PIN.

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_BIT_POS Enumeration

Lists the constants that hold different PORTS bit positions.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_BIT_POS_0,
  PORTS_BIT_POS_1,
  PORTS_BIT_POS_2,
  PORTS_BIT_POS_3,
  PORTS_BIT_POS_4,
  PORTS_BIT_POS_5,
  PORTS_BIT_POS_6,
  PORTS_BIT_POS_7,
  PORTS_BIT_POS_8,
  PORTS_BIT_POS_9,
  PORTS_BIT_POS_10,
  PORTS_BIT_POS_11,
  PORTS_BIT_POS_12,
  PORTS_BIT_POS_13,
  PORTS_BIT_POS_14,
  PORTS_BIT_POS_15
} PORTS_BIT_POS;
```

Members

Members	Description
PORTS_BIT_POS_0	PORT bit position 0
PORTS_BIT_POS_1	PORT bit position 1
PORTS_BIT_POS_2	PORT bit position 2
PORTS_BIT_POS_3	PORT bit position 3
PORTS_BIT_POS_4	PORT bit position 4
PORTS_BIT_POS_5	PORT bit position 5
PORTS_BIT_POS_6	PORT bit position 6
PORTS_BIT_POS_7	PORT bit position 7
PORTS_BIT_POS_8	PORT bit position 8
PORTS_BIT_POS_9	PORT bit position 9
PORTS_BIT_POS_10	PORT bit position 10
PORTS_BIT_POS_11	PORT bit position 11
PORTS_BIT_POS_12	PORT bit position 12
PORTS_BIT_POS_13	PORT bit position 13
PORTS_BIT_POS_14	PORT bit position 14
PORTS_BIT_POS_15	PORT bit position 15

Description

PORTS bit positions

Lists the constants that hold different PORTS bit positions.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_CHANGE_NOTICE_PIN Enumeration

Data type defining the different Change Notification (CN) pins enumerations.

File

help_plib_ports.h

C

```
typedef enum {
  CN0,
  CN1,
  CN2,
  CN3,
  CN4,
  CN5,
  CN6,
  CN7,
```

```
  CN8,
  CN9,
  CN10,
  CN11,
  CN12,
  CN13,
  CN14,
  CN15,
  CN16,
  CN17,
  CN18,
  CN19,
  CN20,
  CN21
} PORTS_CHANGE_NOTICE_PIN;
```

Members

Members	Description
CN0	Change Notification Pin 0
CN1	Change Notification Pin 1
CN2	Change Notification Pin 2
CN3	Change Notification Pin 3
CN4	Change Notification Pin 4
CN5	Change Notification Pin 5
CN6	Change Notification Pin 6
CN7	Change Notification Pin 7
CN8	Change Notification Pin 8
CN9	Change Notification Pin 9
CN10	Change Notification Pin 10
CN11	Change Notification Pin 11
CN12	Change Notification Pin 12
CN13	Change Notification Pin 13
CN14	Change Notification Pin 14
CN15	Change Notification Pin 15
CN16	Change Notification Pin 16
CN17	Change Notification Pin 17
CN18	Change Notification Pin 18
CN19	Change Notification Pin 19
CN20	Change Notification Pin 20
CN21	Change Notification Pin 21

Description

Change Notification Pins enumeration

This data type defines the different CN pins enumerations.

Remarks

Values of these CN Pin enums are different from the other similar enumerators by name PORTS_CN_PIN. These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_CHANNEL Enumeration

Identifies the available Ports channels.

File

help_plib_ports.h

C

```
typedef enum {
  PORT_CHANNEL_A,
  PORT_CHANNEL_B,
  PORT_CHANNEL_C,
  PORT_CHANNEL_D,
```

```
    PORT_CHANNEL_E,
    PORT_CHANNEL_F,
    PORT_CHANNEL_G,
    PORT_CHANNEL_H,
    PORT_CHANNEL_J,
    PORT_CHANNEL_K
} PORTS_CHANNEL;
```

Members

Members	Description
PORT_CHANNEL_A	Port A
PORT_CHANNEL_B	Port B
PORT_CHANNEL_C	Port C
PORT_CHANNEL_D	Port D
PORT_CHANNEL_E	Port E
PORT_CHANNEL_F	Port F
PORT_CHANNEL_G	Port G
PORT_CHANNEL_H	Port H
PORT_CHANNEL_J	Port J
PORT_CHANNEL_K	Port K

Description

PORT Channel ID

This enumeration identifies the available Ports channels.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules are available on all devices. Refer to the specific device data sheet to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

PORTS_DATA_MASK Type

Data type defining the Ports data mask

File

plib_ports.h

C

```
typedef uint16_t PORTS_DATA_MASK;
```

Description

Ports data mask definition

This data type defines the Ports data mask

Remarks

None.

PORTS_DATA_TYPE Type

Data type defining the Ports data type.

File

plib_ports.h

C

```
typedef uint32_t PORTS_DATA_TYPE;
```

Description

Ports data type definition

This data type defines the Ports data type.

Remarks

None.

PORTS_MODULE_ID Enumeration

Identifies the available Ports modules.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_ID_0,
  PORT_NUMBER_OF_MODULES
} PORTS_MODULE_ID;
```

Members

Members	Description
PORT_NUMBER_OF_MODULES	Max number of Instances

Description

PORT Module ID

This enumeration identifies the available Ports modules.

Remarks

Not all modules are available on all devices. Refer to the specific device data sheet to determine which modules are supported. The numbers used in the enumeration values will match the numbers given in the data sheet.

PORTS_PIN_MODE Enumeration

Identifies the available pin modes.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_PIN_MODE_ANALOG,
  PORTS_PIN_MODE_DIGITAL
} PORTS_PIN_MODE;
```

Members

Members	Description
PORTS_PIN_MODE_ANALOG	Port Pin is in Analog Mode
PORTS_PIN_MODE_DIGITAL	Port pin is in Digital Mode

Description

PORTs Pin Mode

This enumeration identifies the available pin modes.

Remarks

Not all modules are available on all devices. Refer to the specific device data sheet to determine which modules are supported.

PORTS_REMAP_FUNCTION Enumeration

Data type defining the different remap function enumerations.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_REMAP_FUNCTION_EXT_INT1,
  PORTS_REMAP_FUNCTION_EXT_INT2,
  PORTS_REMAP_FUNCTION_EXT_INT3,
  PORTS_REMAP_FUNCTION_EXT_INT4,
  PORTS_REMAP_FUNCTION_IC1,
  PORTS_REMAP_FUNCTION_IC2,
  PORTS_REMAP_FUNCTION_IC3,
  PORTS_REMAP_FUNCTION_IC4,
  PORTS_REMAP_FUNCTION_IC5,
  PORTS_REMAP_FUNCTION_IC6,
  PORTS_REMAP_FUNCTION_IC7,
  PORTS_REMAP_FUNCTION_IC8,
  PORTS_REMAP_FUNCTION_IC9,
  PORTS_REMAP_FUNCTION_OC_FAULTA,
  PORTS_REMAP_FUNCTION_OC_FAULTB,
  PORTS_REMAP_FUNCTION_SPI1_CLOCK,
  PORTS_REMAP_FUNCTION_SPI1_DATA,
  PORTS_REMAP_FUNCTION_SPI1_SLAVE_SEL,
  PORTS_REMAP_FUNCTION_SPI2_CLOCK,
  PORTS_REMAP_FUNCTION_SPI2_DATA,
  PORTS_REMAP_FUNCTION_SPI2_SLAVE_SEL,
  PORTS_REMAP_FUNCTION_SPI3_CLOCK,
  PORTS_REMAP_FUNCTION_SPI3_DATA,
  PORTS_REMAP_FUNCTION_SPI3_SLAVE_SEL,
  PORTS_REMAP_FUNCTION_TMR2_EXT_CLOCK,
  PORTS_REMAP_FUNCTION_TMR3_EXT_CLOCK,
  PORTS_REMAP_FUNCTION_TMR4_EXT_CLOCK,
  PORTS_REMAP_FUNCTION_TMR5_EXT_CLOCK,
  PORTS_REMAP_FUNCTION_USART1_CTS,
  PORTS_REMAP_FUNCTION_USART1_RX,
  PORTS_REMAP_FUNCTION_USART2_CTS,
  PORTS_REMAP_FUNCTION_USART2_RX,
  PORTS_REMAP_FUNCTION_USART3_CTS,
  PORTS_REMAP_FUNCTION_USART3_RX,
  PORTS_REMAP_FUNCTION_USART4_CTS,
  PORTS_REMAP_FUNCTION_USART4_RX,
  PORTS_REMAP_FUNCTION_REFCLKI,
  PORTS_REMAP_FUNCTION_NULL,
  PORTS_REMAP_FUNCTION_C1OUT,
  PORTS_REMAP_FUNCTION_C2OUT,
  PORTS_REMAP_FUNCTION_U1TX,
  PORTS_REMAP_FUNCTION_U1RTS,
  PORTS_REMAP_FUNCTION_U2TX,
  PORTS_REMAP_FUNCTION_U2RTS,
  PORTS_REMAP_FUNCTION_SDO1,
  PORTS_REMAP_FUNCTION_SCK1OUT,
  PORTS_REMAP_FUNCTION_SS1OUT,
  PORTS_REMAP_FUNCTION_SDO2,
  PORTS_REMAP_FUNCTION_SCK2OUT,
  PORTS_REMAP_FUNCTION_SS2OUT,
  PORTS_REMAP_FUNCTION_OC1,
  PORTS_REMAP_FUNCTION_OC2,
  PORTS_REMAP_FUNCTION_OC3,
  PORTS_REMAP_FUNCTION_OC4,
  PORTS_REMAP_FUNCTION_OC5,
  PORTS_REMAP_FUNCTION_OC6,
  PORTS_REMAP_FUNCTION_OC7,
  PORTS_REMAP_FUNCTION_OC8,
  PORTS_REMAP_FUNCTION_U3TX,
  PORTS_REMAP_FUNCTION_U3RTS,
  PORTS_REMAP_FUNCTION_U4TX,
  PORTS_REMAP_FUNCTION_U4RTS,
  PORTS_REMAP_FUNCTION_SDO3,
  PORTS_REMAP_FUNCTION_SCK3OUT,
  PORTS_REMAP_FUNCTION_SS3OUT,
  PORTS_REMAP_FUNCTION_OC9,
  PORTS_REMAP_FUNCTION_C3OUT,
  PORTS_REMAP_FUNCTION_REFCLKO
} PORTS_REMAP_FUNCTION;
```

Members

Members	Description
PORTS_REMAP_FUNCTION_EXT_INT1	Input Function Name - External Interrupt 1
PORTS_REMAP_FUNCTION_EXT_INT2	Input Function Name - External Interrupt 2
PORTS_REMAP_FUNCTION_EXT_INT3	Input Function Name - External Interrupt 3
PORTS_REMAP_FUNCTION_EXT_INT4	Input Function Name - External Interrupt 4
PORTS_REMAP_FUNCTION_IC1	Input Function Name - Input Capture 1
PORTS_REMAP_FUNCTION_IC2	Input Function Name - Input Capture 2
PORTS_REMAP_FUNCTION_IC3	Input Function Name - Input Capture 3
PORTS_REMAP_FUNCTION_IC4	Input Function Name - Input Capture 4
PORTS_REMAP_FUNCTION_IC5	Input Function Name - Input Capture 5
PORTS_REMAP_FUNCTION_IC6	Input Function Name - Input Capture 6
PORTS_REMAP_FUNCTION_IC7	Input Function Name - Input Capture 7
PORTS_REMAP_FUNCTION_IC8	Input Function Name - Input Capture 8
PORTS_REMAP_FUNCTION_IC9	Input Function Name - Input Capture 9
PORTS_REMAP_FUNCTION_OC_FAULTA	Input Function Name - Output Compare Fault A
PORTS_REMAP_FUNCTION_OC_FAULTB	Input Function Name - Output Compare Fault B
PORTS_REMAP_FUNCTION_SPI1_CLOCK	Input Function Name - SPI1 Clock
PORTS_REMAP_FUNCTION_SPI1_DATA	Input Function Name - SPI1 Data
PORTS_REMAP_FUNCTION_SPI1_SLAVE_SEL	Input Function Name - SPI1 Slave Select
PORTS_REMAP_FUNCTION_SPI2_CLOCK	Input Function Name - SPI2 Clock
PORTS_REMAP_FUNCTION_SPI2_DATA	Input Function Name - SPI2 Data
PORTS_REMAP_FUNCTION_SPI2_SLAVE_SEL	Input Function Name - SPI2 Slave Select
PORTS_REMAP_FUNCTION_SPI3_CLOCK	Input Function Name - SPI3 Clock
PORTS_REMAP_FUNCTION_SPI3_DATA	Input Function Name - SPI3 Data
PORTS_REMAP_FUNCTION_SPI3_SLAVE_SEL	Input Function Name - SPI3 Slave Select
PORTS_REMAP_FUNCTION_TMR2_EXT_CLOCK	Input Function Name - Timer2 External CLock
PORTS_REMAP_FUNCTION_TMR3_EXT_CLOCK	Input Function Name - Timer3 External Clock
PORTS_REMAP_FUNCTION_TMR4_EXT_CLOCK	Input Function Name - Timer4 External Clock
PORTS_REMAP_FUNCTION_TMR5_EXT_CLOCK	Input Function Name - Timer5 External Clock
PORTS_REMAP_FUNCTION_USART1_CTS	Input Function Name - USART1 Clear To Send
PORTS_REMAP_FUNCTION_USART1_RX	Input Function Name - USART1 Receive
PORTS_REMAP_FUNCTION_USART2_CTS	Input Function Name - USART2 Clear To Send
PORTS_REMAP_FUNCTION_USART2_RX	Input Function Name - USART2 Receive
PORTS_REMAP_FUNCTION_USART3_CTS	Input Function Name - USART3 Clear To Send
PORTS_REMAP_FUNCTION_USART3_RX	Input Function Name - USART3 Receive
PORTS_REMAP_FUNCTION_USART4_CTS	Input Function Name - USART4 Clear To Send
PORTS_REMAP_FUNCTION_USART4_RX	Input Function Name - USART4 Receive
PORTS_REMAP_FUNCTION_REFCLKI	Input Function Name - Reference Clock Input
PORTS_REMAP_FUNCTION_NULL	Output Function Name - Null
PORTS_REMAP_FUNCTION_C1OUT	Output Function Name - Comparator 1 Output
PORTS_REMAP_FUNCTION_C2OUT	Output Function Name - Comparator 2 Output
PORTS_REMAP_FUNCTION_U1TX	Output Function Name - UART1 Transmit
PORTS_REMAP_FUNCTION_U1RTS	Output Function Name - UART1 Request To Send
PORTS_REMAP_FUNCTION_U2TX	Output Function Name - UART2 Transmit
PORTS_REMAP_FUNCTION_U2RTS	Output Function Name - UART2 Request To Send
PORTS_REMAP_FUNCTION_SDO1	Output Function Name - SPI1 Data Output
PORTS_REMAP_FUNCTION_SCK1OUT	Output Function Name - SPI1 Clock Output
PORTS_REMAP_FUNCTION_SS1OUT	Output Function Name - SPI1 Slave Select Output
PORTS_REMAP_FUNCTION_SDO2	Output Function Name - SPI2 Data Output
PORTS_REMAP_FUNCTION_SCK2OUT	Output Function Name - SPI2 Clock Output
PORTS_REMAP_FUNCTION_SS2OUT	Output Function Name - SPI2 Slave Select Output
PORTS_REMAP_FUNCTION_OC1	Output Function Name - Output Compare 1
PORTS_REMAP_FUNCTION_OC2	Output Function Name - Output Compare 2

PORTS_REMAP_FUNCTION_OC3	Output Function Name - Output Compare 3
PORTS_REMAP_FUNCTION_OC4	Output Function Name - Output Compare 4
PORTS_REMAP_FUNCTION_OC5	Output Function Name - Output Compare 5
PORTS_REMAP_FUNCTION_OC6	Output Function Name - Output Compare 6
PORTS_REMAP_FUNCTION_OC7	Output Function Name - Output Compare 7
PORTS_REMAP_FUNCTION_OC8	Output Function Name - Output Compare 8
PORTS_REMAP_FUNCTION_U3TX	Output Function Name - UART3 Transmit
PORTS_REMAP_FUNCTION_U3RTS	Output Function Name - UART3 Request To Send
PORTS_REMAP_FUNCTION_U4TX	Output Function Name - UART4 Transmit
PORTS_REMAP_FUNCTION_U4RTS	Output Function Name - UART4 Request To Send
PORTS_REMAP_FUNCTION_SDO3	Output Function Name - SPI3 Data Output
PORTS_REMAP_FUNCTION_SCK3OUT	Output Function Name - SPI3 Clock Output
PORTS_REMAP_FUNCTION_SS3OUT	Output Function Name - SPI3 Slave Select Output
PORTS_REMAP_FUNCTION_OC9	Output Function Name - Output Compare 9
PORTS_REMAP_FUNCTION_C3OUT	Output Function Name - Comparator 3 Output
PORTS_REMAP_FUNCTION_REFCLKO	Output Function Name - Reference Clock Output

Description

Remap Function Enumeration

This data type defines the different remap function enumerations.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_REMAP_INPUT_FUNCTION Enumeration

Data type defining the different remap input function enumerations.

File

help_plib_ports.h

C

```
typedef enum {
  INPUT_FUNC_INT1,
  INPUT_FUNC_INT2,
  INPUT_FUNC_INT3,
  INPUT_FUNC_INT4,
  INPUT_FUNC_T2CK,
  INPUT_FUNC_T3CK,
  INPUT_FUNC_T4CK,
  INPUT_FUNC_T5CK,
  INPUT_FUNC_T6CK,
  INPUT_FUNC_T7CK,
  INPUT_FUNC_T8CK,
  INPUT_FUNC_T9CK,
  INPUT_FUNC_IC1,
  INPUT_FUNC_IC2,
  INPUT_FUNC_IC3,
  INPUT_FUNC_IC4,
  INPUT_FUNC_IC5,
  INPUT_FUNC_IC6,
  INPUT_FUNC_IC7,
  INPUT_FUNC_IC8,
  INPUT_FUNC_IC9,
  INPUT_FUNC_OCFA,
  INPUT_FUNC_OCFB,
  INPUT_FUNC_U1RX,
  INPUT_FUNC_U1CTS,
  INPUT_FUNC_U2RX,
  INPUT_FUNC_U2CTS,
  INPUT_FUNC_U3RX,
  INPUT_FUNC_U3CTS,
  INPUT_FUNC_U4RX,
  INPUT_FUNC_U4CTS,
  INPUT_FUNC_U5RX,
```

```
  INPUT_FUNC_U5CTS,
  INPUT_FUNC_U6RX,
  INPUT_FUNC_U6CTS,
  INPUT_FUNC_SDI1,
  INPUT_FUNC_SS1,
  INPUT_FUNC_SDI2,
  INPUT_FUNC_SS2,
  INPUT_FUNC_SDI3,
  INPUT_FUNC_SS3,
  INPUT_FUNC_SDI4,
  INPUT_FUNC_SS4,
  INPUT_FUNC_SDI5,
  INPUT_FUNC_SS5,
  INPUT_FUNC_SDI6,
  INPUT_FUNC_SS6,
  INPUT_FUNC_C1RX,
  INPUT_FUNC_C2RX,
  INPUT_FUNC_REFCLKI1,
  INPUT_FUNC_REFCLKI3,
  INPUT_FUNC_REFCLKI4,
  REMAP_NOT_SUPPORTED
} PORTS_REMAP_INPUT_FUNCTION;
```

Description

Remap Input Function Enumeration

This data type defines the different remap input function enumerations.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_REMAP_INPUT_PIN Enumeration

Data type defining the different Ports I/O input pins enumerations.

File

help_plib_ports.h

C

```
typedef enum {
  INPUT_PIN_RPD2,
  INPUT_PIN_RPG8,
  INPUT_PIN_RPF4,
  INPUT_PIN_RPD10,
  INPUT_PIN_RPF1,
  INPUT_PIN_RPB9,
  INPUT_PIN_RPB10,
  INPUT_PIN_RPC14,
  INPUT_PIN_RPB5,
  INPUT_PIN_RPC1,
  INPUT_PIN_RPD14,
  INPUT_PIN_RPG1,
  INPUT_PIN_RPA14,
  INPUT_PIN_RPD6,
  INPUT_PIN_RPD3,
  INPUT_PIN_RPG7,
  INPUT_PIN_RPF5,
  INPUT_PIN_RPD11,
  INPUT_PIN_RPF0,
  INPUT_PIN_RPB1,
  INPUT_PIN_RPE5,
  INPUT_PIN_RPC13,
  INPUT_PIN_RPB3,
  INPUT_PIN_RPC4,
  INPUT_PIN_RPD15,
  INPUT_PIN_RPG0,
  INPUT_PIN_RPA15,
  INPUT_PIN_RPD7,
  INPUT_PIN_RPD9,
  INPUT_PIN_RPG6,
  INPUT_PIN_RPB8,
  INPUT_PIN_RPB15,
```

```
    INPUT_PIN_RPD4,
    INPUT_PIN_RPB0,
    INPUT_PIN_RPE3,
    INPUT_PIN_RPB7,
    INPUT_PIN_RPF12,
    INPUT_PIN_RPD12,
    INPUT_PIN_RPF8,
    INPUT_PIN_RPC3,
    INPUT_PIN_RPE9,
    INPUT_PIN_RPD1,
    INPUT_PIN_RPG9,
    INPUT_PIN_RPB14,
    INPUT_PIN_RPD0,
    INPUT_PIN_RPB6,
    INPUT_PIN_RPD5,
    INPUT_PIN_RPB2,
    INPUT_PIN_RPF3,
    INPUT_PIN_RPF13,
    INPUT_PIN_NC,
    INPUT_PIN_RPF2,
    INPUT_PIN_RPC2,
    INPUT_PIN_RPE8,
    INPUT_PIN_RPF7,
    INPUT_PIN_RPD8,
    INPUT_PIN_RPA0,
    INPUT_PIN_RPB4,
    INPUT_PIN_RPC7,
    INPUT_PIN_RPC0,
    INPUT_PIN_RPC5,
    INPUT_PIN_RPA1,
    INPUT_PIN_RPB11,
    INPUT_PIN_RPA8,
    INPUT_PIN_RPC8,
    INPUT_PIN_RPA9,
    INPUT_PIN_RPA2,
    INPUT_PIN_RPA4,
    INPUT_PIN_RPB13,
    INPUT_PIN_RPC6,
    INPUT_PIN_RPA3,
    INPUT_PIN_RPC9,
    PIN_REMAP_NOT_SUPPORTED
} PORTS_REMAP_INPUT_PIN;
```

Description

PORTS IO Input Pins enumeration

This data type defines the different Ports I/O input pins enumerations.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_REMAP_OUTPUT_FUNCTION Enumeration

Data type defining the different remap output function enumerations.

File

help_plib_ports.h

C

```
typedef enum {
    OUTPUT_FUNC_U3TX,
    OUTPUT_FUNC_U4RTS,
    OUTPUT_FUNC_SDO1,
    OUTPUT_FUNC_SDO2,
    OUTPUT_FUNC_SDO3,
    OUTPUT_FUNC_SDO5,
    OUTPUT_FUNC_SS6,
    OUTPUT_FUNC_OC3,
    OUTPUT_FUNC_OC6,
    OUTPUT_FUNC_REFCLKO4,
    OUTPUT_FUNC_C2OUT,
    OUTPUT_FUNC_C1TX,
```

```
  OUTPUT_FUNC_U1TX,
  OUTPUT_FUNC_U2RTS,
  OUTPUT_FUNC_U5TX,
  OUTPUT_FUNC_U6RTS,
  OUTPUT_FUNC_SDO4,
  OUTPUT_FUNC_OC4,
  OUTPUT_FUNC_OC7,
  OUTPUT_FUNC_REFCLKO1,
  OUTPUT_FUNC_U3RTS,
  OUTPUT_FUNC_U4TX,
  OUTPUT_FUNC_U6TX,
  OUTPUT_FUNC_SS1,
  OUTPUT_FUNC_SS3,
  OUTPUT_FUNC_SS4,
  OUTPUT_FUNC_SS5,
  OUTPUT_FUNC_SDO6,
  OUTPUT_FUNC_OC5,
  OUTPUT_FUNC_OC8,
  OUTPUT_FUNC_C1OUT,
  OUTPUT_FUNC_REFCLKO3,
  OUTPUT_FUNC_U1RTS,
  OUTPUT_FUNC_U2TX,
  OUTPUT_FUNC_U5RTS,
  OUTPUT_FUNC_SS2,
  OUTPUT_FUNC_OC2,
  OUTPUT_FUNC_OC1,
  OUTPUT_FUNC_OC9,
  OUTPUT_FUNC_C2TX,
  OUTPUT_FUNC_C3OUT,
  OUTPUT_FUNC_REFCLKO,
  OUTPUT_FUNC_NO_CONNECT
} PORTS_REMAP_OUTPUT_FUNCTION;
```

Description

Remap Output Function Enumeration

This data type defines the different remap output function enumerations.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_REMAP_OUTPUT_PIN Enumeration

Data type defining the different Ports I/O output pins enumerations.

File

help_plib_ports.h

C

```
typedef enum {
  OUTPUT_PIN_RPA0,
  OUTPUT_PIN_RPA1,
  OUTPUT_PIN_RPA2,
  OUTPUT_PIN_RPA3,
  OUTPUT_PIN_RPA4,
  OUTPUT_PIN_RPA8,
  OUTPUT_PIN_RPA9,
  OUTPUT_PIN_RPA14,
  OUTPUT_PIN_RPA15,
  OUTPUT_PIN_RPB0,
  OUTPUT_PIN_RPB1,
  OUTPUT_PIN_RPB2,
  OUTPUT_PIN_RPB3,
  OUTPUT_PIN_RPB4,
  OUTPUT_PIN_RPB5,
  OUTPUT_PIN_RPB6,
  OUTPUT_PIN_RPB7,
  OUTPUT_PIN_RPB8,
  OUTPUT_PIN_RPB9,
  OUTPUT_PIN_RPB10,
  OUTPUT_PIN_RPB11,
  OUTPUT_PIN_RPB13,
```

```
  OUTPUT_PIN_RPB14,
  OUTPUT_PIN_RPB15,
  OUTPUT_PIN_RPC0,
  OUTPUT_PIN_RPC1,
  OUTPUT_PIN_RPC2,
  OUTPUT_PIN_RPC3,
  OUTPUT_PIN_RPC4,
  OUTPUT_PIN_RPC5,
  OUTPUT_PIN_RPC6,
  OUTPUT_PIN_RPC7,
  OUTPUT_PIN_RPC8,
  OUTPUT_PIN_RPC9,
  OUTPUT_PIN_RPC13,
  OUTPUT_PIN_RPC14,
  OUTPUT_PIN_RPD0,
  OUTPUT_PIN_RPD1,
  OUTPUT_PIN_RPD2,
  OUTPUT_PIN_RPD3,
  OUTPUT_PIN_RPD4,
  OUTPUT_PIN_RPD5,
  OUTPUT_PIN_RPD6,
  OUTPUT_PIN_RPD7,
  OUTPUT_PIN_RPD8,
  OUTPUT_PIN_RPD9,
  OUTPUT_PIN_RPD10,
  OUTPUT_PIN_RPD11,
  OUTPUT_PIN_RPD12,
  OUTPUT_PIN_RPD14,
  OUTPUT_PIN_RPD15,
  OUTPUT_PIN_RPE3,
  OUTPUT_PIN_RPE5,
  OUTPUT_PIN_RPE8,
  OUTPUT_PIN_RPE9,
  OUTPUT_PIN_RPF0,
  OUTPUT_PIN_RPF1,
  OUTPUT_PIN_RPF2,
  OUTPUT_PIN_RPF3,
  OUTPUT_PIN_RPF4,
  OUTPUT_PIN_RPF5,
  OUTPUT_PIN_RPF6,
  OUTPUT_PIN_RPF7,
  OUTPUT_PIN_RPF8,
  OUTPUT_PIN_RPF12,
  OUTPUT_PIN_RPF13,
  OUTPUT_PIN_RPG0,
  OUTPUT_PIN_RPG1,
  OUTPUT_PIN_RPG6,
  OUTPUT_PIN_RPG7,
  OUTPUT_PIN_RPG8,
  OUTPUT_PIN_RPG9,
  REMAP_NOT_SUPPORTED
} PORTS_REMAP_OUTPUT_PIN;
```

Description

PORTS IO Output Pins enumeration

This data type defines the different Ports I/O output pins enumerations.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_REMAP_PIN Enumeration

Data type defining the different remappable input/output enumerations.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_REMAP_PIN_RPI0,
  PORTS_REMAP_PIN_RPI1,
```

```
    PORTS_REMAP_PIN_RPI2,
    PORTS_REMAP_PIN_RPI3,
    PORTS_REMAP_PIN_RPI4,
    PORTS_REMAP_PIN_RPI5,
    PORTS_REMAP_PIN_RPI6,
    PORTS_REMAP_PIN_RPI7,
    PORTS_REMAP_PIN_RPI8,
    PORTS_REMAP_PIN_RPI9,
    PORTS_REMAP_PIN_RPI10,
    PORTS_REMAP_PIN_RPI11,
    PORTS_REMAP_PIN_RPI12,
    PORTS_REMAP_PIN_RPI13,
    PORTS_REMAP_PIN_RPI14,
    PORTS_REMAP_PIN_RPI15,
    PORTS_REMAP_PIN_RPI16,
    PORTS_REMAP_PIN_RPI17,
    PORTS_REMAP_PIN_RPI18,
    PORTS_REMAP_PIN_RPI19,
    PORTS_REMAP_PIN_RPI20,
    PORTS_REMAP_PIN_RPI21,
    PORTS_REMAP_PIN_RPI22,
    PORTS_REMAP_PIN_RPI23,
    PORTS_REMAP_PIN_RPI24,
    PORTS_REMAP_PIN_RPI25,
    PORTS_REMAP_PIN_RPI26,
    PORTS_REMAP_PIN_RPI27,
    PORTS_REMAP_PIN_RPI28,
    PORTS_REMAP_PIN_RPI29,
    PORTS_REMAP_PIN_RPI30,
    PORTS_REMAP_PIN_RPI31,
    PORTS_REMAP_PIN_RPI32,
    PORTS_REMAP_PIN_RPI33,
    PORTS_REMAP_PIN_RPI34,
    PORTS_REMAP_PIN_RPI35,
    PORTS_REMAP_PIN_RPI36,
    PORTS_REMAP_PIN_RPI37,
    PORTS_REMAP_PIN_RPI38,
    PORTS_REMAP_PIN_RPI39,
    PORTS_REMAP_PIN_RPI40,
    PORTS_REMAP_PIN_RPI41,
    PORTS_REMAP_PIN_RPI42,
    PORTS_REMAP_PIN_RPI43,
    PORTS_REMAP_PIN_RPI44,
    PORTS_REMAP_PIN_RPI45,
    PORTS_REMAP_PIN_RP0,
    PORTS_REMAP_PIN_RP1,
    PORTS_REMAP_PIN_RP2,
    PORTS_REMAP_PIN_RP3,
    PORTS_REMAP_PIN_RP4,
    PORTS_REMAP_PIN_RP5,
    PORTS_REMAP_PIN_RP6,
    PORTS_REMAP_PIN_RP7,
    PORTS_REMAP_PIN_RP8,
    PORTS_REMAP_PIN_RP9,
    PORTS_REMAP_PIN_RP10,
    PORTS_REMAP_PIN_RP11,
    PORTS_REMAP_PIN_RP12,
    PORTS_REMAP_PIN_RP13,
    PORTS_REMAP_PIN_RP14,
    PORTS_REMAP_PIN_RP15,
    PORTS_REMAP_PIN_RP16,
    PORTS_REMAP_PIN_RP17,
    PORTS_REMAP_PIN_RP18,
    PORTS_REMAP_PIN_RP19,
    PORTS_REMAP_PIN_RP20,
    PORTS_REMAP_PIN_RP21,
    PORTS_REMAP_PIN_RP22,
    PORTS_REMAP_PIN_RP23,
    PORTS_REMAP_PIN_RP24,
    PORTS_REMAP_PIN_RP25,
    PORTS_REMAP_PIN_RP26,
    PORTS_REMAP_PIN_RP27,
    PORTS_REMAP_PIN_RP28,
    PORTS_REMAP_PIN_RP29,
    PORTS_REMAP_PIN_RP30,
```

```
    PORTS_REMAP_PIN_RP31,
    PORTS_REMAP_PIN_RPA0,
    PORTS_REMAP_PIN_RPB3,
    PORTS_REMAP_PIN_RPB4,
    PORTS_REMAP_PIN_RPB15,
    PORTS_REMAP_PIN_RPB7,
    PORTS_REMAP_PIN_RPC7,
    PORTS_REMAP_PIN_RPC0,
    PORTS_REMAP_PIN_RPC5,
    PORTS_REMAP_PIN_RPA1,
    PORTS_REMAP_PIN_RPB5,
    PORTS_REMAP_PIN_RPB1,
    PORTS_REMAP_PIN_RPB11,
    PORTS_REMAP_PIN_RPB8,
    PORTS_REMAP_PIN_RPA8,
    PORTS_REMAP_PIN_RPC8,
    PORTS_REMAP_PIN_RPA9,
    PORTS_REMAP_PIN_RPA2,
    PORTS_REMAP_PIN_RPB6,
    PORTS_REMAP_PIN_RPA4,
    PORTS_REMAP_PIN_RPB13,
    PORTS_REMAP_PIN_RPB2,
    PORTS_REMAP_PIN_RPC6,
    PORTS_REMAP_PIN_RPC1,
    PORTS_REMAP_PIN_RPC3,
    PORTS_REMAP_PIN_RPA3,
    PORTS_REMAP_PIN_RPB14,
    PORTS_REMAP_PIN_RPB0,
    PORTS_REMAP_PIN_RPB10,
    PORTS_REMAP_PIN_RPB9,
    PORTS_REMAP_PIN_RPC9,
    PORTS_REMAP_PIN_RPC2,
    PORTS_REMAP_PIN_RPC4
} PORTS_REMAP_PIN;
```

Members

Members	Description
PORTS_REMAP_PIN_RPI0	Remappable input RPI0
PORTS_REMAP_PIN_RPI1	Remappable input RPI1
PORTS_REMAP_PIN_RPI2	Remappable input RPI2
PORTS_REMAP_PIN_RPI3	Remappable input RPI3
PORTS_REMAP_PIN_RPI4	Remappable input RPI4
PORTS_REMAP_PIN_RPI5	Remappable input RPI5
PORTS_REMAP_PIN_RPI6	Remappable input RPI6
PORTS_REMAP_PIN_RPI7	Remappable input RPI7
PORTS_REMAP_PIN_RPI8	Remappable input RPI8
PORTS_REMAP_PIN_RPI9	Remappable input RPI9
PORTS_REMAP_PIN_RPI10	Remappable input RPI10
PORTS_REMAP_PIN_RPI11	Remappable input RPI11
PORTS_REMAP_PIN_RPI12	Remappable input RPI12
PORTS_REMAP_PIN_RPI13	Remappable input RPI13
PORTS_REMAP_PIN_RPI14	Remappable input RPI14
PORTS_REMAP_PIN_RPI15	Remappable input RPI15
PORTS_REMAP_PIN_RPI16	Remappable input RPI16
PORTS_REMAP_PIN_RPI17	Remappable input RPI17
PORTS_REMAP_PIN_RPI18	Remappable input RPI18
PORTS_REMAP_PIN_RPI19	Remappable input RPI19
PORTS_REMAP_PIN_RPI20	Remappable input RPI20
PORTS_REMAP_PIN_RPI21	Remappable input RPI21
PORTS_REMAP_PIN_RPI22	Remappable input RPI22
PORTS_REMAP_PIN_RPI23	Remappable input RPI23
PORTS_REMAP_PIN_RPI24	Remappable input RPI24
PORTS_REMAP_PIN_RPI25	Remappable input RPI25
PORTS_REMAP_PIN_RPI26	Remappable input RPI26

PORTS_REMAP_PIN_RPI27	Remappable input RPI27
PORTS_REMAP_PIN_RPI28	Remappable input RPI28
PORTS_REMAP_PIN_RPI29	Remappable input RPI29
PORTS_REMAP_PIN_RPI30	Remappable input RPI30
PORTS_REMAP_PIN_RPI31	Remappable input RPI31
PORTS_REMAP_PIN_RPI32	Remappable input RPI32
PORTS_REMAP_PIN_RPI33	Remappable input RPI33
PORTS_REMAP_PIN_RPI34	Remappable input RPI34
PORTS_REMAP_PIN_RPI35	Remappable input RPI35
PORTS_REMAP_PIN_RPI36	Remappable input RPI36
PORTS_REMAP_PIN_RPI37	Remappable input RPI37
PORTS_REMAP_PIN_RPI38	Remappable input RPI38
PORTS_REMAP_PIN_RPI39	Remappable input RPI39
PORTS_REMAP_PIN_RPI40	Remappable input RPI40
PORTS_REMAP_PIN_RPI41	Remappable input RPI41
PORTS_REMAP_PIN_RPI42	Remappable input RPI42
PORTS_REMAP_PIN_RPI43	Remappable input RPI43
PORTS_REMAP_PIN_RPI44	Remappable input RPI44
PORTS_REMAP_PIN_RPI45	Remappable input RPI45
PORTS_REMAP_PIN_RP0	Remappable output RP0
PORTS_REMAP_PIN_RP1	Remappable output RP1
PORTS_REMAP_PIN_RP2	Remappable output RP2
PORTS_REMAP_PIN_RP3	Remappable output RP3
PORTS_REMAP_PIN_RP4	Remappable output RP4
PORTS_REMAP_PIN_RP5	Remappable output RP5
PORTS_REMAP_PIN_RP6	Remappable output RP6
PORTS_REMAP_PIN_RP7	Remappable output RP7
PORTS_REMAP_PIN_RP8	Remappable output RP8
PORTS_REMAP_PIN_RP9	Remappable output RP9
PORTS_REMAP_PIN_RP10	Remappable output RP10
PORTS_REMAP_PIN_RP11	Remappable output RP11
PORTS_REMAP_PIN_RP12	Remappable output RP12
PORTS_REMAP_PIN_RP13	Remappable output RP13
PORTS_REMAP_PIN_RP14	Remappable output RP14
PORTS_REMAP_PIN_RP15	Remappable output RP15
PORTS_REMAP_PIN_RP16	Remappable output RP16
PORTS_REMAP_PIN_RP17	Remappable output RP17
PORTS_REMAP_PIN_RP18	Remappable output RP18
PORTS_REMAP_PIN_RP19	Remappable output RP19
PORTS_REMAP_PIN_RP20	Remappable output RP20
PORTS_REMAP_PIN_RP21	Remappable output RP21
PORTS_REMAP_PIN_RP22	Remappable output RP22
PORTS_REMAP_PIN_RP23	Remappable output RP23
PORTS_REMAP_PIN_RP24	Remappable output RP24
PORTS_REMAP_PIN_RP25	Remappable output RP25
PORTS_REMAP_PIN_RP26	Remappable output RP26
PORTS_REMAP_PIN_RP27	Remappable output RP27
PORTS_REMAP_PIN_RP28	Remappable output RP28
PORTS_REMAP_PIN_RP29	Remappable output RP29
PORTS_REMAP_PIN_RP30	Remappable output RP30
PORTS_REMAP_PIN_RP31	Remappable output RP31
PORTS_REMAP_PIN_RPA0	Remappable output RPA0
PORTS_REMAP_PIN_RPB3	Remappable output RPB3
PORTS_REMAP_PIN_RPB4	Remappable output RPB4
PORTS_REMAP_PIN_RPB15	Remappable output RPB15

PORTS_REMAP_PIN_RPB7	Remappable output RPB7
PORTS_REMAP_PIN_RPC7	Remappable output RPC7
PORTS_REMAP_PIN_RPC0	Remappable output RPC0
PORTS_REMAP_PIN_RPC5	Remappable output RPC5
PORTS_REMAP_PIN_RPA1	Remappable output RPA1
PORTS_REMAP_PIN_RPB5	Remappable output RPB5
PORTS_REMAP_PIN_RPB1	Remappable output RPB1
PORTS_REMAP_PIN_RPB11	Remappable output RPB11
PORTS_REMAP_PIN_RPB8	Remappable output RPB8
PORTS_REMAP_PIN_RPA8	Remappable output RPA8
PORTS_REMAP_PIN_RPC8	Remappable output RPC8
PORTS_REMAP_PIN_RPA9	Remappable output RPA9
PORTS_REMAP_PIN_RPA2	Remappable output RPA2
PORTS_REMAP_PIN_RPB6	Remappable output RPB6
PORTS_REMAP_PIN_RPA4	Remappable output RPA4
PORTS_REMAP_PIN_RPB13	Remappable output RPB13
PORTS_REMAP_PIN_RPB2	Remappable output RPB2
PORTS_REMAP_PIN_RPC6	Remappable output RPC6
PORTS_REMAP_PIN_RPC1	Remappable output RPC1
PORTS_REMAP_PIN_RPC3	Remappable output RPC3
PORTS_REMAP_PIN_RPA3	Remappable output RPA3
PORTS_REMAP_PIN_RPB14	Remappable output RPB14
PORTS_REMAP_PIN_RPB0	Remappable output RPB0
PORTS_REMAP_PIN_RPB10	Remappable output RPB10
PORTS_REMAP_PIN_RPB9	Remappable output RPB9
PORTS_REMAP_PIN_RPC9	Remappable output RPC9
PORTS_REMAP_PIN_RPC2	Remappable output RPC2
PORTS_REMAP_PIN_RPC4	Remappable output RPC4

Description

Remappable Input/Output Enumeration

This data type defines the different remappable input/output enumerations.

Remarks

These are the super set enumerations provided for documentation, not all constants are available on all devices.

PORTS_AN_PIN Enumeration

Data type defining the different analog input pins.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_AN_PIN_0,
  PORTS_AN_PIN_1,
  PORTS_AN_PIN_2,
  PORTS_AN_PIN_3,
  PORTS_AN_PIN_4,
  PORTS_AN_PIN_5,
  PORTS_AN_PIN_6,
  PORTS_AN_PIN_7,
  PORTS_AN_PIN_8,
  PORTS_AN_PIN_9,
  PORTS_AN_PIN_10,
  PORTS_AN_PIN_11,
  PORTS_AN_PIN_12,
  PORTS_AN_PIN_13,
  PORTS_AN_PIN_14,
  PORTS_AN_PIN_15,
```

```
  PORTS_AN_PIN_ALL
} PORTS_AN_PIN;
```

Members

Members	Description
PORTS_AN_PIN_0	AN Pin 0
PORTS_AN_PIN_1	AN Pin 1
PORTS_AN_PIN_2	AN Pin 2
PORTS_AN_PIN_3	AN Pin 3
PORTS_AN_PIN_4	AN Pin 4
PORTS_AN_PIN_5	AN Pin 5
PORTS_AN_PIN_6	AN Pin 6
PORTS_AN_PIN_7	AN Pin 7
PORTS_AN_PIN_8	AN Pin 8
PORTS_AN_PIN_9	AN Pin 9
PORTS_AN_PIN_10	AN Pin 10
PORTS_AN_PIN_11	AN Pin 11
PORTS_AN_PIN_12	AN Pin 12
PORTS_AN_PIN_13	AN Pin 13
PORTS_AN_PIN_14	AN Pin 14
PORTS_AN_PIN_15	AN Pin 15
PORTS_AN_PIN_ALL	All the AN Pins

Description

Analog input pins

This data type defines the different analog input pins.

Remarks

Values of these AN Pin enums are different from the other similar enumerators by name PORTS_ANALOG_PIN.

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_CN_PIN Enumeration

Data type defining the different Change Notification (CN) pins enumerations.

File

help_plib_ports.h

C

```
typedef enum {
  CHANGE_NOTICE_PIN_0,
  CHANGE_NOTICE_PIN_1,
  CHANGE_NOTICE_PIN_2,
  CHANGE_NOTICE_PIN_3,
  CHANGE_NOTICE_PIN_4,
  CHANGE_NOTICE_PIN_5,
  CHANGE_NOTICE_PIN_6,
  CHANGE_NOTICE_PIN_7,
  CHANGE_NOTICE_PIN_8,
  CHANGE_NOTICE_PIN_9,
  CHANGE_NOTICE_PIN_10,
  CHANGE_NOTICE_PIN_11,
  CHANGE_NOTICE_PIN_12,
  CHANGE_NOTICE_PIN_13,
  CHANGE_NOTICE_PIN_14,
  CHANGE_NOTICE_PIN_15,
  CHANGE_NOTICE_PIN_16,
  CHANGE_NOTICE_PIN_17,
  CHANGE_NOTICE_PIN_18,
  CHANGE_NOTICE_PIN_19,
  CHANGE_NOTICE_PIN_20,
  CHANGE_NOTICE_PIN_21,
  CHANGE_NOTICE_PIN_ALL
} PORTS_CN_PIN;
```

Members

Members	Description
CHANGE_NOTICE_PIN_0	Change Notice Pin 0
CHANGE_NOTICE_PIN_1	Change Notice Pin 1
CHANGE_NOTICE_PIN_2	Change Notice Pin 2
CHANGE_NOTICE_PIN_3	Change Notice Pin 3
CHANGE_NOTICE_PIN_4	Change Notice Pin 4
CHANGE_NOTICE_PIN_5	Change Notice Pin 5
CHANGE_NOTICE_PIN_6	Change Notice Pin 6
CHANGE_NOTICE_PIN_7	Change Notice Pin 7
CHANGE_NOTICE_PIN_8	Change Notice Pin 8
CHANGE_NOTICE_PIN_9	Change Notice Pin 9
CHANGE_NOTICE_PIN_10	Change Notice Pin 10
CHANGE_NOTICE_PIN_11	Change Notice Pin 11
CHANGE_NOTICE_PIN_12	Change Notice Pin 12
CHANGE_NOTICE_PIN_13	Change Notice Pin 13
CHANGE_NOTICE_PIN_14	Change Notice Pin 14
CHANGE_NOTICE_PIN_15	Change Notice Pin 15
CHANGE_NOTICE_PIN_16	Change Notice Pin 16
CHANGE_NOTICE_PIN_17	Change Notice Pin 17
CHANGE_NOTICE_PIN_18	Change Notice Pin 18
CHANGE_NOTICE_PIN_19	Change Notice Pin 19
CHANGE_NOTICE_PIN_20	Change Notice Pin 20
CHANGE_NOTICE_PIN_21	Change Notice Pin 21
CHANGE_NOTICE_PIN_ALL	All the Change Notice Pins

Description

Change Notice Pins enumeration

This data type defines the different CN pins enumerations.

Remarks

Values of these CN Pin enums are different from the other similar enumerators by name PORTS_CHANGE_NOTICE_PIN. These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_CHANGE_NOTICE_EDGE Enumeration

Data type defining the different edge type for change notification.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_CHANGE_NOTICE_EDGE_RISING = 0,
  PORTS_CHANGE_NOTICE_EDGE_FALLING = 1
} PORTS_CHANGE_NOTICE_EDGE;
```

Members

Members	Description
PORTS_CHANGE_NOTICE_EDGE_RISING = 0	Change Notification on rising edge
PORTS_CHANGE_NOTICE_EDGE_FALLING = 1	Change Notification on falling edge

Description

PORTS Change Notice Edge Type

This data type defines the different edge type for change notification.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_CHANGE_NOTICE_METHOD Enumeration

Data type defining the different method of ports pin change notification.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_CHANGE_NOTICE_METHOD_SAMPLED = 0,
  PORTS_CHANGE_NOTICE_METHOD_EDGE_DETECT = 1
} PORTS_CHANGE_NOTICE_METHOD;
```

Members

Members	Description
PORTS_CHANGE_NOTICE_METHOD_SAMPLED = 0	Change notification happens when there is level transition
PORTS_CHANGE_NOTICE_METHOD_EDGE_DETECT = 1	Change notification happens when there is edge transition

Description

PORTS Pin Change Notice Method

This data type defines the different method of ports pin change notification.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

PORTS_PIN_SLEW_RATE Enumeration

Data type defining the different slew rates for port pins.

File

help_plib_ports.h

C

```
typedef enum {
  PORTS_PIN_SLEW_RATE_FASTEST = 0x00,
  PORTS_PIN_SLEW_RATE_FAST = 0x01,
  PORTS_PIN_SLEW_RATE_SLOW = 0x02,
  PORTS_PIN_SLEW_RATE_SLOWEST = 0x03
} PORTS_PIN_SLEW_RATE;
```

Members

Members	Description
PORTS_PIN_SLEW_RATE_FASTEST = 0x00	Slew rate fastest
PORTS_PIN_SLEW_RATE_FAST = 0x01	Slew rate fast
PORTS_PIN_SLEW_RATE_SLOW = 0x02	Slew rate slow
PORTS_PIN_SLEW_RATE_SLOWEST = 0x03	Slew rate slowest

Description

PORTS Pin Slew Rate

This data type defines the different slew rates for port pins.

Remarks

These are the superset enumerations provided for documentation, not all constants are available on all devices.

Files

Files

Name	Description
plib_ports.h	Ports Peripheral Library Interface header for Ports function definitions.
help_plib_ports.h	This is file help_plib_ports.h.

Description

This section lists the source and header files used by the library.

plib_ports.h

Ports Peripheral Library Interface header for Ports function definitions.

Functions

	Name	Description
	PLIB_PORTS_AnPinsModeSelect	Enables the selected AN pins as analog or digital.
	PLIB_PORTS_ChangeNoticeDisable	Global Change Notice disable.
	PLIB_PORTS_ChangeNoticeEnable	Global Change Notice enable.
	PLIB_PORTS_ChangeNoticeInIdleDisable	CPU Idle halts the Change Notice operation.
	PLIB_PORTS_ChangeNoticeInIdleEnable	CPU Idle mode does not affect Change Notice operation.
	PLIB_PORTS_ChangeNoticeInIdlePerPortDisable	Change Notification halts in Idle mode for selected channel.
	PLIB_PORTS_ChangeNoticeInIdlePerPortEnable	Allows CN to be working in Idle mode for selected channel.
	PLIB_PORTS_ChangeNoticePerPortHasOccured	This is function PLIB_PORTS_ChangeNoticePerPortHasOccured.
	PLIB_PORTS_ChangeNoticePerPortHasOccurred	checks the status of change on the pin
	PLIB_PORTS_ChangeNoticePerPortTurnOff	Disables the change notification for selected port.
	PLIB_PORTS_ChangeNoticePerPortTurnOn	Enables the change notification for selected port.
	PLIB_PORTS_ChangeNoticePullDownPerPortDisable	Disables the pull-down for selected Change Notice pins.
	PLIB_PORTS_ChangeNoticePullDownPerPortEnable	Enables the pull-down for selected Change Notice pins.
	PLIB_PORTS_ChangeNoticePullUpDisable	Disable pull-up on input change.
	PLIB_PORTS_ChangeNoticePullUpEnable	Enable pull-up on input change.
	PLIB_PORTS_ChangeNoticePullUpPerPortDisable	Disables weak pull-up for the selected pin.
	PLIB_PORTS_ChangeNoticePullUpPerPortEnable	Enables the pull-up for selected Change Notice pins.
	PLIB_PORTS_ChannelChangeNoticeDisable	Disables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_ChannelChangeNoticeEdgeDisable	Disables selected type of edge for selected CN pins.
	PLIB_PORTS_ChannelChangeNoticeEdgeEnable	Enables selected type of edge for selected CN pins.
	PLIB_PORTS_ChannelChangeNoticeEnable	Enables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_ChannelChangeNoticeMethodGet	Gets the Change Notice style for the selected port channel.
	PLIB_PORTS_ChannelChangeNoticeMethodSelect	Selects the Change Notice style for selected port channel.
	PLIB_PORTS_ChannelChangeNoticePullDownDisable	Disables Change Notice pull-down for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticePullDownEnable	Enables Change Notice pull-down for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticePullUpDisable	Disables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_ChannelChangeNoticePullUpEnable	Enables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_ChannelModeSelect	Enables the selected channel pins as analog or digital.
	PLIB_PORTS_ChannelSlewRateSelect	Selects the slew rate for selected channel pins.
	PLIB_PORTS_Clear	Clears the selected digital port/latch bits.
	PLIB_PORTS_CnPinsDisable	Disables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_CnPinsEnable	Enables CN interrupt for the selected pins of a channel.
	PLIB_PORTS_CnPinsPullUpDisable	Disables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_CnPinsPullUpEnable	Enables Change Notice pull-up for the selected channel pins.
	PLIB_PORTS_DirectionGet	Reads the direction of the selected digital port.
	PLIB_PORTS_DirectionInputSet	Makes the selected pins direction input.
	PLIB_PORTS_DirectionOutputSet	Makes the selected pins direction output.
	PLIB_PORTS_ExistsAnPinsMode	Identifies whether the AnPinsMode feature exists on the Ports module.

	PLIB_PORTS_ExistsChangeNotice	Identifies whether the ChangeNotice feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticeEdgeControl	Identifies whether the ChangeNoticeEdgeControl feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticeEdgeStatus	Identifies whether the ChangeNoticeEdgeStatus feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticeInIdle	Identifies whether the ChangeNoticeInIdle feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePerPortInIdle	Identifies whether the ChangeNoticeInIdlePerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePerPortStatus	Identifies whether the ChangeNoticePerPortStatus feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePerPortTurnOn	Identifies whether the ChangeNoticePerPortTurnOn feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePullDownPerPort	Identifies whether the ChangeNoticePullDownPerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePullUp	Identifies whether the ChangeNoticePullup feature exists on the Ports module.
	PLIB_PORTS_ExistsChangeNoticePullUpPerPort	Identifies whether the ChangeNoticePullUpPerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsChannelChangeNoticeMethod	Identifies whether the ChannelChangeNoticeMethod feature exists on the Ports module.
	PLIB_PORTS_ExistsLatchRead	Identifies whether the LatchRead feature exists on the Ports module.
	PLIB_PORTS_ExistsPinChangeNotice	Identifies whether the PinChangeNotice feature exists on the Ports module.
	PLIB_PORTS_ExistsPinChangeNoticePerPort	Identifies whether the PinChangeNoticePerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsPinMode	Identifies whether the PinMode feature exists on the Ports module.
	PLIB_PORTS_ExistsPinModePerPort	Identifies whether the PinModePerPort feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsDirection	Identifies whether the PortsDirection feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsOpenDrain	Identifies whether the PortsOpenDrain feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsRead	Identifies whether the PortsRead feature exists on the Ports module.
	PLIB_PORTS_ExistsPortsWrite	Identifies whether the PortsWrite feature exists on the Ports module.
	PLIB_PORTS_ExistsRemapInput	Identifies whether the RemapInput feature exists on the Ports module.
	PLIB_PORTS_ExistsRemapOutput	Identifies whether the RemapOutput feature exists on the Ports module.
	PLIB_PORTS_ExistsSlewRateControl	Identifies whether the SlewRateControl feature exists on the Ports module.
	PLIB_PORTS_OpenDrainDisable	Disables the open drain functionality for the selected port.
	PLIB_PORTS_OpenDrainEnable	Enables the open drain functionality for the selected port pins.
	PLIB_PORTS_PinChangeNoticeDisable	Port pin Change Notice disable.
	PLIB_PORTS_PinChangeNoticeEdgeHasOccurred	Check Change Notice edge status.
	PLIB_PORTS_PinChangeNoticeEdgeIsEnabled	Check if Change Notice edge is enabled or not.
	PLIB_PORTS_PinChangeNoticeEnable	Port pin Change Notice interrupt enable.
	PLIB_PORTS_PinChangeNoticePerPortDisable	Disables CN interrupt for the selected pin.
	PLIB_PORTS_PinChangeNoticePerPortEnable	Enables CN interrupt for the selected pin.
	PLIB_PORTS_PinClear	Clears the selected digital pin/latch.
	PLIB_PORTS_PinDirectionInputSet	Makes the selected pin direction input
	PLIB_PORTS_PinDirectionOutputSet	Makes the selected pin direction output
	PLIB_PORTS_PinGet	Reads/Gets data from the selected digital pin.
	PLIB_PORTS_PinGetLatched	Reads/Gets data from the selected latch.
	PLIB_PORTS_PinModePerPortSelect	Enables the selected port pin as analog or digital.
	PLIB_PORTS_PinModeSelect	Enables the selected pin as analog or digital.
	PLIB_PORTS_PinOpenDrainDisable	Disables the open drain functionality for the selected pin.
	PLIB_PORTS_PinOpenDrainEnable	Enables the open drain functionality for the selected pin.
	PLIB_PORTS_PinSet	Sets the selected digital pin/latch.
	PLIB_PORTS_PinSlewRateGet	Gets the slew rate for selected port pin.
	PLIB_PORTS_PinToggle	Toggles the selected digital pin/latch.
	PLIB_PORTS_PinWrite	Writes the selected digital pin/latch.
	PLIB_PORTS_Read	Reads the selected digital port.
	PLIB_PORTS_ReadLatched	Reads and returns data from the selected Latch.
	PLIB_PORTS_RemapInput	Input function remapping.

	PLIB_PORTS_RemapOutput	Output function remapping.
	PLIB_PORTS_Set	Sets the selected bits of the port.
	PLIB_PORTS_Toggle	Toggles the selected digital port/latch.
	PLIB_PORTS_Write	Writes the selected digital port/latch.

Types

	Name	Description
	PORTS_DATA_MASK	Data type defining the Ports data mask
	PORTS_DATA_TYPE	Data type defining the Ports data type.

Description

Ports Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Ports Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the Ports peripheral.

File Name

plib_ports.h

Company

Microchip Technology Inc.

help_plib_ports.h

Enumerations

	Name	Description
	PORTS_AN_PIN	Data type defining the different analog input pins.
	PORTS_ANALOG_PIN	Data type defining the different analog input pins.
	PORTS_BIT_POS	Lists the constants that hold different PORTS bit positions.
	PORTS_CHANGE_NOTICE_EDGE	Data type defining the different edge type for change notification.
	PORTS_CHANGE_NOTICE_METHOD	Data type defining the different method of ports pin change notification.
	PORTS_CHANGE_NOTICE_PIN	Data type defining the different Change Notification (CN) pins enumerations.
	PORTS_CHANNEL	Identifies the available Ports channels.
	PORTS_CN_PIN	Data type defining the different Change Notification (CN) pins enumerations.
	PORTS_MODULE_ID	Identifies the available Ports modules.
	PORTS_PIN_MODE	Identifies the available pin modes.
	PORTS_PIN_SLEW_RATE	Data type defining the different slew rates for port pins.
	PORTS_REMAP_FUNCTION	Data type defining the different remap function enumerations.
	PORTS_REMAP_INPUT_FUNCTION	Data type defining the different remap input function enumerations.
	PORTS_REMAP_INPUT_PIN	Data type defining the different Ports I/O input pins enumerations.
	PORTS_REMAP_OUTPUT_FUNCTION	Data type defining the different remap output function enumerations.
	PORTS_REMAP_OUTPUT_PIN	Data type defining the different Ports I/O output pins enumerations.
	PORTS_REMAP_PIN	Data type defining the different remappable input/output enumerations.

Description

This is file help_plib_ports.h.

Power Peripheral Library

This section describes the Power Peripheral Library.

Introduction

This library provides a low-level abstraction of the Power Controller modules on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Power Controller is a key component of a microcontroller. Power control features allow the user to obtain the balance of power consumption and performance that is best suited to their application.

The features provided in this library discuss specific software commands that are used dynamically to place the device in low-power (i.e., power-saving) modes, which disable power and clocking of selected features. When implementing a system to achieve the lowest possible power consumption while maintain the needed performance, the oscillator configuration and clocking of all peripherals must also be considered. Oscillator configuration and peripheral clocking are addressed in the Oscillator Peripheral Library. In addition, each peripheral may have its own low-power mode settings that are controlled by the library of that peripheral. The user is advised to refer to the libraries of the peripherals used for their particular application.

Key features present on a power controller include (a microcontroller can support one or more of these power features):

- **Power Source Configuration**: Provides the ability to enable and disable a source that may regulate the power consumption in the device
- **Power Status Flags:** Indicate the status of the particular power feature in the device

Note: Not all devices support all of the features discussed in this section. Please refer to specific device data sheet to determine the supported features for your device.

Using the Library

This topic describes the basic architecture of the Power Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_power.h`

The interface to the Power Peripheral Library is defined in the `plib_power.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Power Peripheral Library must include `peripheral.h`.

Peripheral Module IDs

Peripheral libraries are indexed to allow a single library to control any number of instances of a peripheral in a single microcontroller. To support this, the first parameter to each operation in a peripheral library is the module instance ID. The module instance ID is defined by an enumeration that is defined in the processor-specific header files (included by the library's interface header). Not all microcontrollers will have all instances of the module listed in this enumeration. Please refer to the specific data sheet to determine availability.

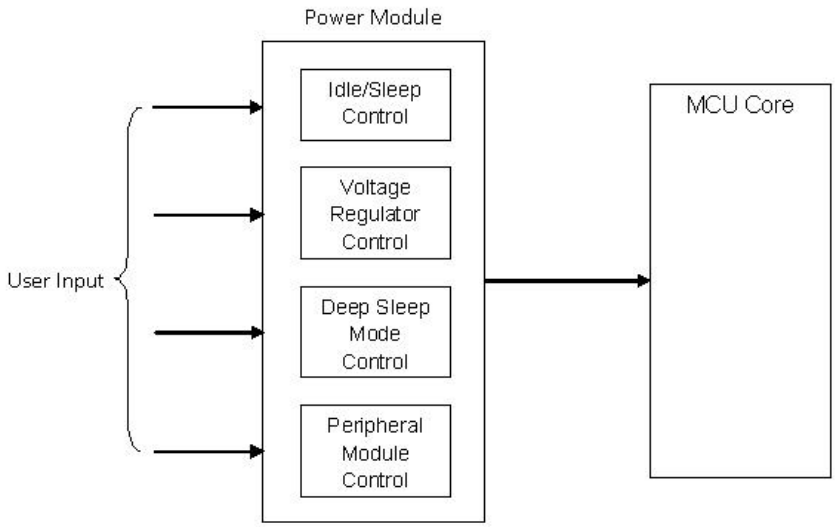
Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

Hardware Abstraction Model

This library provides a low-level abstraction of the Power Controller module on Microchip microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

Hardware Abstraction Block Diagram



This section discusses the various power-saving features implemented in hardware. Combinations of these methods can be used to selectively tailor an application’s power consumption, while still maintaining critical or timing-sensitive application features.

Note: Not all devices support all of the features discussed in this section. Please refer to specific device data sheet to determine the supported features for your device.

Instruction-based Power-Saving Modes

In addition to full-power operation, otherwise known as Run mode, PIC microcontrollers also provide other instruction-based power-saving modes. By powering down to different modes, different functional areas of the microcontroller allow progressive reductions of operating and idle power consumption. In addition, these modes can be tailored for more power reduction, but at a trade-off of some operating features.

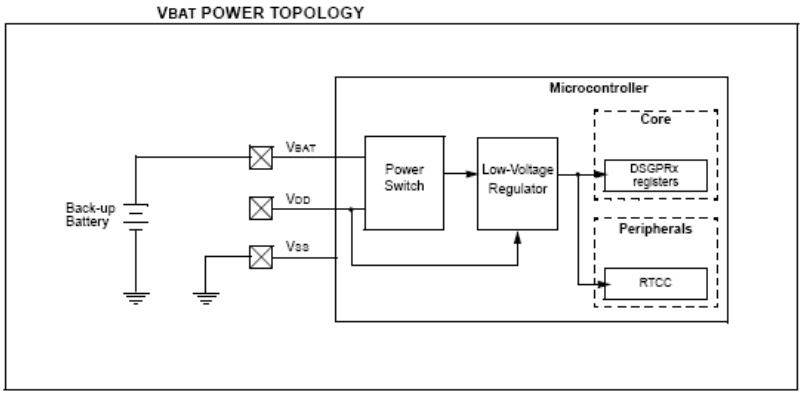
- **Idle Mode:** In Idle mode, the CPU is halted, but the System Clock (SYSCLK) source is still enabled. This allows peripherals to continue to operate when the CPU is halted. Peripherals can be individually configured to halt when entering Idle mode by setting their respective SIDL bit. Latency when exiting Idle mode is very low due to the CPU oscillator source remaining active.
- **Sleep Mode:** The CPU, SYSCLK source and any peripherals that operate on the system clock source are disabled
- **Deep Sleep Mode:** The CPU SYSCLK source, and all the peripherals, with the exception of the Real-Time Clock and Calendar (RTCC) and the Deep Sleep Watchdog Timer (DSWDT), are disabled. This is the lowest power mode for the device. The power to RAM and Flash is also disabled. Deep Sleep mode represents the lowest power mode available without removing power from the application.

The instruction-based power-saving modes are exited as a result of several different hardware triggers. When the device exits one of these three operating modes, it is said to ‘wake-up'.

Hardware-based Power-Saving Mode

VBAT mode is a hardware-based power-saving mode that maintains only the most critical operations when a power loss occurs on VDD. The mode does this by powering the systems from a back-up power source connected to the VBAT pin. In this mode, the RTCC can run even when there is no power on VDD.

VBAT mode is entered whenever power is removed from VDD. An on-chip power switch detects the power loss from VDD and connects the VBAT pin to the low-voltage regulator. Entering VBAT mode requires that a power source, distinct from the main VDD power source, be available on the VBAT pin and that VDD be completely removed from the VDD pin(s). Removing VDD can be either unintentional, as in a power failure, or as part of a deliberate power reduction strategy.



Note: Not all devices support all of the features discussed in this section. Please refer to specific device data sheet to determine the supported features for your device.

Selective Peripheral Power Control

Sleep and Idle modes allow users to substantially reduce power consumption by slowing or stopping the CPU clock. Even so, peripheral modules may still remain clocked, and thus, consume some amount of power. There may be cases where the application needs what these modes do not provide: the ability to allocate limited power resources to the CPU while eliminating power consumption from the peripherals. PIC microcontrollers address this requirement by allowing peripheral modules to be selectively enabled or disabled, reducing or eliminating their power consumption.

In addition to the selective peripheral power control implemented as part of this library, devices will also include power saving features that are implemented in their specific libraries. These features include the ability to stop (shutdown) automatically when entering idle mode. On some peripherals the ability to continue operation in Sleep mode is available. Always review the libraries for the peripherals used by your application for specific and unique features that will help minimize power consumption.

Disabling Peripheral Modules

Most of the peripheral modules in the PIC microcontroller family architecture can be selectively disabled, reducing or essentially eliminating their power consumption during all operating modes.

All peripheral modules (except for I/O ports) also have a Control bit that can disable their functionality. These bits, known as the Peripheral Module Disable (PMD) bits, are generically named: 'xxMD'. These bits are located in the PMDx Special Function Registers (SFRs). In contrast to the module enable bits, the xxMD bit must be set to a '1' to disable the module.

While the PMD and module enable bits both disable a peripheral’s functionality, the PMD bit completely shuts down the peripheral, effectively powering down all circuits and removing all clock sources. This has the additional effect of making any of the module’s control and buffer registers, mapped in the SFR space, unavailable for operations. In other words, when the PMD bit is used to disable a module, the peripheral ceases to exist until the PMD bit is cleared.

The PMD bit is most useful in highly power-sensitive applications, where even tiny savings in power consumption can determine the ability of an application to function. In these cases, the bits can be set before the main body of the application to remove those peripherals that will not be needed.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Power Controller module.

Library Interface Section	Description
Initialization Functions	This section provides a set of functions for configuring the Power-Saving features and enabling or disabling the particular power feature.
Status Functions	This section provides a function to read the status of a particular power feature.
HLVD Functions	This section provides High/Low-Voltage Detect functions.
Deep Sleep Functions	This section provides Deep Sleep mode functions.
Feature Existence Functions	This section provides functions that can be used to determine available features.

Note: The interface provided is a superset of all power functionality available on the device. Refer to the **"Power-Saving Features"** chapter in the specific device data sheet for availability.

How the Library Works

Provides information on how the library works.

Description

Usage of this library is partitioned into two major sections.

Initialization

The steps that are required to initialize the Power Controller module vary for different microcontrollers. Refer to the specific device data sheet to determine the correct initialization sequence. The following information provides a general overview.

Power Source Setup

- A module can be disabled for the microcontroller to be able to save power. Use the function PLIB_POWER_ModuleDisable to disable the particular module. POWER_MODULE_DISABLE provides a list of possible modules sources.
- On most microcontrollers, the voltage regulator can be controlled during Sleep mode. It can be enabled (turned on) using PLIB_POWER_VoltageRegulatorEnable and disabled (turned off) using PLIB_POWER_VoltageRegulatorDisable.
- On some microcontrollers, Deep Sleep mode is enabled using PLIB_POWER_DeepSleepModeEnable. The Deep Sleep mode is disabled using PLIB_POWER_DeepSleepModeDisable.

Power Status

Status of the particular power feature indicates the previous status of that feature. This feature can be used to determine whether the device was in Sleep or Idle mode before waking up. It also allows software to clear the previous status of the device.

Configuring the Library

The library is configured for the supported Power Controller modules when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Initialization Functions

	Name	Description
	PLIB_POWER_DeepSleepModeDisable	Disables Deep Sleep mode.
	PLIB_POWER_DeepSleepModeEnable	Enables Deep Sleep mode.
	PLIB_POWER_PeripheralModuleDisable	Disable the power supply for the module selected in the source.
	PLIB_POWER_PeripheralModuleEnable	Enable the power supply for the module selected in the source.
	PLIB_POWER_PeripheralModuleIsEnabled	Checks to see whether or not the selected peripheral is enabled.
	PLIB_POWER_VoltageRegulatorDisable	Disables the voltage regulator during Sleep mode.

	PLIB_POWER_VoltageRegulatorEnable	Enable the voltage regulator during Sleep mode.
	PLIB_POWER_DeepSleepGPRsRetentionDisable	Disables the General Purpose Registers retention.
	PLIB_POWER_DeepSleepGPRsRetentionEnable	Enables the General Purpose Registers retention.
	PLIB_POWER_DeepSleepModeIsEnabled	Returns the enable/disable status of Deep Sleep mode.
	PLIB_POWER_DeepSleepModuleDisable	Disables the module in Deep Sleep mode.
	PLIB_POWER_DeepSleepModuleEnable	Enables the module in Deep Sleep mode.
	PLIB_POWER_DeepSleepPortPinsStateRelease	Releases I/O pins upon wake-up from Deep Sleep mode.
	PLIB_POWER_DeepSleepPortPinsStateRetain	Enables the I/O pins to retain their previous states upon wake-up from Deep Sleep mode.
	PLIB_POWER_DeepSleepWakeupEventDisable	Disables wake-up from Deep Sleep mode by the selected event.
	PLIB_POWER_DeepSleepWakeupEventEnable	Enables wake-up from the Deep Sleep mode by the selected event.

b) Status Functions

	Name	Description
	PLIB_POWER_ClearIdleStatus	Clears the Idle mode status of the device.
	PLIB_POWER_ClearSleepStatus	Clear the Sleep mode status bit of the device.
	PLIB_POWER_DeviceWasInIdleMode	Returns the Idle mode status of the device.
	PLIB_POWER_DeviceWasInSleepMode	Returns the Sleep mode status of the device.
	PLIB_POWER_HighVoltageOnVDD1V8HasOccurred	Returns the DDR2 High Voltage Detection status of the device.
	PLIB_POWER_VoltageRegulatorIsEnabled	Provides the status of the voltage regulator during Sleep mode.
	PLIB_POWER_DeepSleepStatusClear	Clears the Deep Sleep mode status bit of the device.
	PLIB_POWER_DeepSleepEventStatusClear	Clear any events that occurred during Deep Sleep mode.
	PLIB_POWER_DeepSleepEventStatusGet	Returns the events that occurred during Deep Sleep mode.
	PLIB_POWER_DeepSleepModeHasOccurred	Returns the Deep Sleep mode status of the device.

c) HLVD Functions

	Name	Description
	PLIB_POWER_HLVDBandGapVoltageIsStable	Returns the status of Band Gap voltage.
	PLIB_POWER_HLVDDisable	Disables High/Low-Voltage Detection on VDD.
	PLIB_POWER_HLVDEnable	Enables High/Low-Voltage Detection (HLVD) on VDD.
	PLIB_POWER_HLVDIsEnabled	Returns the enable/disable status of High/Low-Voltage Detection on VDD.
	PLIB_POWER_HLVDLimitSelect	Selects the HLVD limit.
	PLIB_POWER_HLVDModeSelect	Selects the Voltage Detection mode.
	PLIB_POWER_HLVDStatusGet	Returns the HLVD event status.
	PLIB_POWER_HLVDStopInIdleDisable	Continues HLVD operation when the device enters Idle mode.
	PLIB_POWER_HLVDStopInIdleEnable	Discontinues HLVD operation when the device enters Idle mode.
	PLIB_POWER_HLVDStopInIdleIsEnabled	Returns the Stop in Idle mode status of the HLVD operation.

d) Deep Sleep Functions

	Name	Description
	PLIB_POWER_DeepSleepGPRRead	Reads 32-bit data from the Deep Sleep General Purpose Register.
	PLIB_POWER_DeepSleepGPRWrite	Writes 32-bit data into the Deep Sleep General Purpose Register.

e) Feature Existence Functions

	Name	Description
	PLIB_POWER_ExistsDeepSleepMode	Identifies whether the DeepSleepModeControl feature exists on the Power module.
	PLIB_POWER_ExistsIdleStatus	Identifies whether the IdleStatus feature exists on the Power module.
	PLIB_POWER_ExistsPeripheralModuleControl	Identifies whether the PeripheralModuleControl feature exists on the Power module.
	PLIB_POWER_ExistsSleepStatus	Identifies whether the SleepStatus feature exists on the Power module.
	PLIB_POWER_ExistsVoltageRegulatorControl	Identifies whether the VoltageRegulatorControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepEventStatus	Identifies whether the DeepSleepEventStatus feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepGPROperation	Identifies whether the DeepSleepGPROperation feature exists on the Power module.

	PLIB_POWER_ExistsDeepSleepGPRsRetentionControl	Identifies whether the DeepSleepGPRsRetentionControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepModuleControl	Identifies whether the DeepSleepModuleControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepPortPinsStateControl	Identifies whether the DeepSleepPortPinsStateControl feature exists on the Power module.
	PLIB_POWER_ExistsHighVoltageOnVDD1V8	Identifies whether the HighVoltageOnVDD1V8 feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepModeOccurrence	Identifies whether the DeepSleepModeOccurrence feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepWakeupEventControl	Identifies whether the DeepSleepWakeupEventControl feature exists on the Power module.
	PLIB_POWER_ExistsHLVDBandGapVoltageStability	Identifies whether the HLVDBandGapVoltageStability feature exists on the Power module.
	PLIB_POWER_ExistsHLVDEnableControl	Identifies whether the HLVDEnableControl feature exists on the Power module.
	PLIB_POWER_ExistsHLVDLimitSelection	Identifies whether the HLVDLimitSelection feature exists on the Power module.
	PLIB_POWER_ExistsHLVDModeControl	Identifies whether the HLVDModeControl feature exists on the Power module.
	PLIB_POWER_ExistsHLVDStatus	Identifies whether the HLVDStatus feature exists on the Power module.
	PLIB_POWER_ExistsHLVDStopInIdleControl	Identifies whether the HLVDStopInIdleControl feature exists on the Power module.

f) Data Types and Constants

	Name	Description
	DEEP_SLEEP_EVENT	Lists the events that occurred during Deep Sleep mode.
	DEEP_SLEEP_GPR	Lists the Deep Sleep General Purpose Registers (GPRs).
	DEEP_SLEEP_MODULE	Lists the modules that can be enabled/disabled during Deep Sleep mode.
	DEEP_SLEEP_WAKE_UP_EVENT	Lists the events that can be used to wake the device from Deep Sleep mode.
	POWER_MODULE	List of the modules whose power can be controlled.
	POWER_MODULE_ID	Identifies the supported Power modules.
	HLVD_LIMIT	Lists the voltage limits for the HLVD module.
	HLVD_MODE	Lists the modes for the High/Low Voltage Detection (HLVD) module.

Description

This section describes the Application Programming Interface (API) functions of the Power Peripheral Library.

Refer to each section for a detailed description.

a) Initialization Functions

PLIB_POWER_DeepSleepModeDisable Function

Disables Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepModeDisable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function disables Deep Sleep mode. Deep Sleep mode is intended to provide the lowest levels of power consumption available without requiring the use of external switches to completely remove all power from the device.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_POWER_ExistsDeepSleepMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepModeDisable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepModeDisable (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepModeEnable Function

Enables Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepModeEnable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function enables Deep Sleep mode. Deep Sleep mode is intended to provide the lowest levels of power consumption available without requiring the use of external switches to completely remove all power from the device.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepModeEnable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepModeEnable (POWER_MODULE_ID index)

PLIB_POWER_PeripheralModuleDisable Function

Disable the power supply for the module selected in the source.

File

plib_power.h

C

```
void PLIB_POWER_PeripheralModuleDisable(POWER_MODULE_ID index, POWER_MODULE source);
```

Returns

None.

Description

This function completely shuts down the selected peripheral, effectively powering down all circuits and removing all clock sources. This has the additional affect of making any of the module's control and buffer registers, which are mapped in the SFR space, unavailable for operations.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsPeripheralModuleControl in your application to determine whether this feature is available.

Disabling a peripheral module while it's ON bit is set, may result in undefined behavior. The ON bit for the associated peripheral module must be cleared prior to disable a module via this API.

Preconditions

The PMDLOCK bit of the Configuration register should be cleared using the function PLIB_DEVCON_DeviceRegistersUnlock (this function allows changes in the PMD registers). Refer to the Device Control (DEVCON) Peripheral Library Help (or System Service) and the specific device data sheet for more information.

Example

```
// System Unlock
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);
// Unlock PMD registers
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID_0, DEVCON_PMD_REGISTERS);
// Disable ADC module
PLIB_POWER_PeripheralModuleDisable (POWER_ID_0, POWER_MODULE_ADC);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Select the module to be disabled from POWER_MODULE enum

Function

void PLIB_POWER_PeripheralModuleDisable (POWER_MODULE_ID index, POWER_MODULE source)

PLIB_POWER_PeripheralModuleEnable Function

Enable the power supply for the module selected in the source.

File

plib_power.h

C

```
void PLIB_POWER_PeripheralModuleEnable(POWER_MODULE_ID index, POWER_MODULE source);
```

Returns

None.

Description

This function enables the power supply for the selected module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsPeripheralModuleControl in your application to determine whether this feature is available.

Preconditions

The PMDLOCK bit of the Configuration register should be cleared using the function PLIB_DEVCON_DeviceRegistersUnlock (this function allows changes in the PMD registers). Refer to the Device Control (DEVCON) Peripheral Library Help (or System Service) and the specific device data sheet for more information.

Example

```
// System Unlock
PLIB_DEVCON_SystemUnlock(DEVCON_ID_0);
// Unlock PMD registers
PLIB_DEVCON_DeviceRegistersUnlock(DEVCON_ID_0, DEVCON_PMD_REGISTERS);
// Enable ADC module
```

```
PLIB_POWER_PeripheralModuleEnable (POWER_ID_0, POWER_MODULE_ADC);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Select the module to be enabled from POWER_MODULE enum

Function

void PLIB_POWER_PeripheralModuleEnable (POWER_MODULE_ID index, POWER_MODULE source)

PLIB_POWER_PeripheralModuleIsEnabled Function

Checks to see whether or not the selected peripheral is enabled.

File

plib_power.h

C

```
bool PLIB_POWER_PeripheralModuleIsEnabled(POWER_MODULE_ID index, POWER_MODULE source);
```

Returns

- true - Power of the selected peripheral is enabled
- false - Power of the selected peripheral is disabled

Description

This function checks to see whether or not the selected peripheral is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsPeripheralModuleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_POWER_PeripheralModuleIsEnabled ( POWER_MODULE_0, POWER_MODULE_ADC ))
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Select the module to be enabled from the POWER_MODULE enum

Function

bool PLIB_POWER_PeripheralModuleIsEnabled (POWER_MODULE_ID index, POWER_MODULE source)

PLIB_POWER_VoltageRegulatorDisable Function

Disables the voltage regulator during Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_VoltageRegulatorDisable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function disables the voltage regulator during Sleep mode for the selected device.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsVoltageRegulatorControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_VoltageRegulatorDisable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_VoltageRegulatorDisable (POWER_MODULE_ID index)

PLIB_POWER_VoltageRegulatorEnable Function

Enable the voltage regulator during Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_VoltageRegulatorEnable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function enables the voltage regulator during Sleep mode for the selected device.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsVoltageRegulatorControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_VoltageRegulatorEnable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_VoltageRegulatorEnable (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepGPRsRetentionDisable Function

Disables the General Purpose Registers retention.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepGPRsRetentionDisable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function disables the Deep Sleep General Purpose Registers to retain their values when the device enters Deep Sleep/VBAT mode, which means the power to DEEP_SLEEP_GPR_1 to DEEP_SLEEP_GPR_32 will be dismissed. Power to DEEP_SLEEP_GPR_0 will be retained by default.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepGPRsRetentionControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepGPRsRetentionDisable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepGPRsRetentionDisable (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepGPRsRetentionEnable Function

Enables the General Purpose Registers retention.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepGPRsRetentionEnable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function enables the Deep Sleep General Purpose Registers to retain their values when the device enters Deep Sleep/VBAT mode, which means the power to DEEP_SLEEP_GPR_1 to DEEP_SLEEP_GPR_32 will be maintained. Power to DEEP_SLEEP_GPR_0 will be retained by default.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepGPRsRetentionControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepGPRsRetentionEnable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepGPRsRetentionEnable (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepModeIsEnabled Function

Returns the enable/disable status of Deep Sleep mode.

File

plib_power.h

C

```
bool PLIB_POWER_DeepSleepModeIsEnabled(POWER_MODULE_ID index);
```

Returns

- true - Deep Sleep mode is enabled
- false - Deep Sleep mode is not enabled

Description

This function returns whether or not Deep Sleep mode is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeepSleepModeIsEnabled(POWER_ID_0))
{
    //Enter Deep Sleep by executing Sleep mode.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_DeepSleepModeIsEnabled (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepModuleDisable Function

Disables the module in Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepModuleDisable(POWER_MODULE_ID index, DEEP_SLEEP_MODULE module);
```

Returns

None.

Description

This function disables the selected module in Deep Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepModuleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepModuleDisable(POWER_ID_0 , DEEP_SLEEP_MODULE_RTCC);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
module	Possible module from DEEP_SLEEP_MODULE

Function

void PLIB_POWER_DeepSleepModuleDisable(POWER_MODULE_ID index, DEEP_SLEEP_MODULE module)

PLIB_POWER_DeepSleepModuleEnable Function

Enables the module in Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepModuleEnable(POWER_MODULE_ID index, DEEP_SLEEP_MODULE module);
```

Returns

None.

Description

This function enables the selected module in Deep Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepModuleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepModuleEnable(POWER_ID_0 , DEEP_SLEEP_MODULE_RTCC);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
module	Possible module from DEEP_SLEEP_MODULE

Function

void PLIB_POWER_DeepSleepModuleEnable(POWER_MODULE_ID index, DEEP_SLEEP_MODULE module)

PLIB_POWER_DeepSleepPortPinsStateRelease Function

Releases I/O pins upon wake-up from Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepPortPinsStateRelease(POWER_MODULE_ID index);
```

Returns

None.

Description

This function releases the I/O pins upon wake-up from Deep Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepPortPinsStateControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepPortPinsStateRelease(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepPortPinsStateRelease (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepPortPinsStateRetain Function

Enables the I/O pins to retain their previous states upon wake-up from Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepPortPinsStateRetain(POWER_MODULE_ID index);
```

Returns

None.

Description

This function enables the I/O pins to retain their previous states upon wake-up from Deep Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepPortPinsStateControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepPortPinsStateRetain(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepPortPinsStateRetain (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepWakeupEventDisable Function

Disables wake-up from Deep Sleep mode by the selected event.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepWakeupEventDisable(POWER_MODULE_ID index, DEEP_SLEEP_WAKE_UP_EVENT event);
```

Returns

None.

Description

This function disables wake-up from Deep Sleep mode by the selected event.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepWakeupEventControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepWakeupEventDisable(POWER_ID_0 , DEEP_SLEEP_WAKE_UP_EVENT_RTCC);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
event	Possible event from DEEP_SLEEP_WAKE_UP_EVENT

Function

void PLIB_POWER_DeepSleepWakeupEventDisable(POWER_MODULE_ID index, DEEP_SLEEP_WAKE_UP_EVENT event)

PLIB_POWER_DeepSleepWakeupEventEnable Function

Enables wake-up from the Deep Sleep mode by the selected event.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepWakeupEventEnable(POWER_MODULE_ID index, DEEP_SLEEP_WAKE_UP_EVENT event);
```

Returns

None.

Description

This function enables wake-up from the Deep Sleep mode by the selected event.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepWakeupEventControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_DeepSleepWakeupEventEnable(POWER_ID_0 , DEEP_SLEEP_WAKE_UP_EVENT_RTCC);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
event	Possible event from DEEP_SLEEP_WAKE_UP_EVENT

Function

void PLIB_POWER_DeepSleepWakeupEventEnable(POWER_MODULE_ID index, DEEP_SLEEP_WAKE_UP_EVENT event)

b) Status Functions

PLIB_POWER_ClearIdleStatus Function

Clears the Idle mode status of the device.

File

plib_power.h

C

```
void PLIB_POWER_ClearIdleStatus(POWER_MODULE_ID index);
```

Returns

None.

Description

This function clears the Idle status bit of the device if it was previously in Idle mode. However, it does not mean that it wakes the device from Idle.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsIdleStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeviceWasInIdleMode(POWER_ID_0))
{
    PLIB_POWER_ClearIdleStatus (POWER_ID_0);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_ClearIdleStatus (POWER_MODULE_ID index)

PLIB_POWER_ClearSleepStatus Function

Clear the Sleep mode status bit of the device.

File

plib_power.h

C

```
void PLIB_POWER_ClearSleepStatus(POWER_MODULE_ID index);
```

Returns

None.

Description

This function clears the Sleep status bit of the device if it was previously in Sleep mode. However, it does not mean that it wakes the device from sleep.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsSleepStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeviceWasInSleepMode(POWER_ID_0))
{
    PLIB_POWER_ClearSleepStatus (POWER_ID_0);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_ClearSleepStatus (POWER_MODULE_ID index)

PLIB_POWER_DeviceWasInIdleMode Function

Returns the Idle mode status of the device.

File

plib_power.h

C

```
bool PLIB_POWER_DeviceWasInIdleMode(POWER_MODULE_ID index);
```

Returns

- true - The device was in Idle mode before waking up
- false - The device was not in Idle mode

Description

This function returns the Idle mode status of the device. It indicates whether or not the device was in Idle mode before waking up.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsIdleStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeviceWasInIdleMode(POWER_ID_0))
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_DeviceWasInIdleMode (POWER_MODULE_ID index)

PLIB_POWER_DeviceWasInSleepMode Function

Returns the Sleep mode status of the device.

File

plib_power.h

C

```
bool PLIB_POWER_DeviceWasInSleepMode(POWER_MODULE_ID index);
```

Returns

- true - The device was in Sleep mode before waking up
- false - The device was not in Sleep mode

Description

This function returns the Sleep mode status of the device. It indicates whether or not the device was in Sleep mode before waking up.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsSleepStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeviceWasInSleepMode(POWER_ID_0))
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_DeviceWasInSleepMode (POWER_MODULE_ID index)

PLIB_POWER_HighVoltageOnVDD1V8HasOccurred Function

Returns the DDR2 High Voltage Detection status of the device.

File

plib_power.h

C

```
bool PLIB_POWER_HighVoltageOnVDD1V8HasOccurred(POWER_MODULE_ID index);
```

Returns

- true - A high-voltage condition on the VDD1V8 voltage has occurred
- false - A high-voltage condition on the VDD1V8 voltage has not occurred

Description

This function returns the DDR2 High Voltage Detection status of the device.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHighVoltageOnVDD1V8 in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_HighVoltageOnVDD1V8HasOccurred(POWER_ID_0))
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_HighVoltageOnVDD1V8HasOccurred (POWER_MODULE_ID index)

PLIB_POWER_VoltageRegulatorIsEnabled Function

Provides the status of the voltage regulator during Sleep mode.

File

plib_power.h

C

```
bool PLIB_POWER_VoltageRegulatorIsEnabled(POWER_MODULE_ID index);
```

Returns

- true - The voltage regulator will be enabled in Sleep mode
- false - The voltage regulator will be disabled in Sleep mode

Description

This function provides the status of the voltage regulator during Sleep mode for the selected device.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsVoltageRegulatorControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_POWER_VoltageRegulatorIsEnabled(POWER_ID_0))
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_VoltageRegulatorIsEnabled (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepStatusClear Function

Clears the Deep Sleep mode status bit of the device.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepStatusClear(POWER_MODULE_ID index);
```

Returns

None.

Description

This function clears the Deep Sleep status bit of the device if it was previously in Deep Sleep mode. However, it does not mean that it wakes the device from Deep Sleep.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepModeOccurrence in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeepSleepModeHasOccurred(POWER_ID_0))
{
    PLIB_POWER_DeepSleepStatusClear (POWER_ID_0);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_DeepSleepStatusClear (POWER_MODULE_ID index)

PLIB_POWER_DeepSleepEventStatusClear Function

Clear any events that occurred during Deep Sleep mode.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepEventStatusClear(POWER_MODULE_ID index, DEEP_SLEEP_EVENT event);
```

Returns

None.

Description

This function accept the events to clear that occurred during Deep Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepEventStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if((PLIB_POWER_DeepSleepEventStatusGet(POWER_ID_0)& DEEP_SLEEP_EVENT_BOR) != 0)
{
    PLIB_POWER_DeepSleepEventStatusClear(POWER_ID_0, DEEP_SLEEP_EVENT_BOR);
    //BOR event occurred in Deep Sleep Mode
    //Therefore, take appropriate action

}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
event	The event that occurred during the Deep Sleep

Function

void PLIB_POWER_DeepSleepEventStatusClear(POWER_MODULE_ID index , DEEP_SLEEP_EVENT event)

PLIB_POWER_DeepSleepEventStatusGet Function

Returns the events that occurred during Deep Sleep mode.

File

plib_power.h

C

```
DEEP_SLEEP_EVENT PLIB_POWER_DeepSleepEventStatusGet(POWER_MODULE_ID index);
```

Returns

DEEP_SLEEP_EVENT - The Event that has occurred during Deep Sleep

Description

This function returns the events that occurred during Deep Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepEventStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if((PLIB_POWER_DeepSleepEventStatusGet(POWER_ID_0)&
(DEEP_SLEEP_EVENT_RTCC_ALARM|DEEP_SLEEP_EVENT_BOR) != 0)
{
    //BOR or RTCC alarm event occurred in Deep Sleep Mode
    //Therefore, take appropriate action
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

DEEP_SLEEP_EVENT PLIB_POWER_DeepSleepEventStatusGet(POWER_MODULE_ID index)

PLIB_POWER_DeepSleepModeHasOccurred Function

Returns the Deep Sleep mode status of the device.

File

plib_power.h

C

```
bool PLIB_POWER_DeepSleepModeHasOccurred(POWER_MODULE_ID index);
```

Returns

- true - The device was in Deep Sleep mode before waking up
- false - The device was not in Deep Sleep mode

Description

This function returns whether or not the device was in Deep Sleep mode before waking up.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepModeOccurrence in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_DeepSleepModeHasOccurred(POWER_ID_0))
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_DeepSleepModeHasOccurred (POWER_MODULE_ID index)

c) HLVD Functions

PLIB_POWER_HLVDBandGapVoltageIsStable Function

Returns the status of Band Gap voltage.

File

plib_power.h

C

```
bool PLIB_POWER_HLVDBandGapVoltageIsStable(POWER_MODULE_ID index);
```

Returns

- true - Band Gap Voltage is stable
- false - Band Gap Voltage is unstable

Description

This function returns whether the Band Gap Voltage is stable or not.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDBandGapVoltageStability in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool BGStatus;
BGStatus = PLIB_POWER_HLVDBandGapVoltageIsStable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_HLVDBandGapVoltageIsStable (POWER_MODULE_ID index)

PLIB_POWER_HLVDDisable Function

Disables High/Low-Voltage Detection on VDD.

File

plib_power.h

C

```
void PLIB_POWER_HLVDDisable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function disables the HLVD module. The HLVD module is used to detect high/low-voltage conditions on VDD.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_HLVDDisable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_HLVDDisable (POWER_MODULE_ID index)

PLIB_POWER_HLVDEnable Function

Enables High/Low-Voltage Detection (HLVD) on VDD.

File

plib_power.h

C

```
void PLIB_POWER_HLVDEnable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function enables the High/Low-Voltage Detection (HLVD) module. The HLVD module is used to detect high/low-voltage conditions on VDD.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDEnableControl in your application to determine whether this feature is available.

Preconditions

Select Low-Voltage or High-Voltage Detection mode by calling PLIB_POWER_HLVDModeSelect.

Example

```
PLIB_POWER_HLVDEnable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_HLVDEnable (POWER_MODULE_ID index)

PLIB_POWER_HLVDIsEnabled Function

Returns the enable/disable status of High/Low-Voltage Detection on VDD.

File

plib_power.h

C

```
bool PLIB_POWER_HLVDIsEnabled(POWER_MODULE_ID index);
```

Returns

- true - HLVD on VDD is enabled
- false - HLVD on VDD is not enabled

Description

This function returns whether or not High/Low-Voltage Detection on VDD is enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_HLVDIsEnabled(POWER_ID_0))
{
    // HLVD is enabled.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_HLVDIsEnabled (POWER_MODULE_ID index)

PLIB_POWER_HLVDLimitSelect Function

Selects the HLVD limit.

File

plib_power.h

C

```
void PLIB_POWER_HLVDLimitSelect(POWER_MODULE_ID index, HLVD_LIMIT limit);
```

Returns

None.

Description

This function selects voltage limit for HLVD on VDD.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDLimitSelection in your application to determine whether this feature is available.

Preconditions

The HLVD module should be disabled by calling PLIB_POWER_HLVDDisable.

Example

```
PLIB_POWER_HLVDLimitSelect( POWER_ID_0, HLVD_LIMIT_ANALOG_INPUT_ON_HLVDIN );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
limit	Voltage limit for HLVD on VDD

Function

void PLIB_POWER_HLVDLimitSelect (POWER_MODULE_ID index, HLVD_LIMIT limit)

PLIB_POWER_HLVDModeSelect Function

Selects the Voltage Detection mode.

File

plib_power.h

C

```
void PLIB_POWER_HLVDModeSelect(POWER_MODULE_ID index, HLVD_MODE mode);
```

Returns

None.

Description

This function selects either the High-Voltage or Low-Voltage Detection mode.

In High-Voltage Detection (HVD) mode, an event occurs when the voltage equals or exceeds the voltage limit.

In Low-Voltage Detection (LVD) mode, an event occurs when the voltage equals or falls below the voltage limit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDModeControl in your application to determine whether this feature is available.

Preconditions

The HLVD module should be disabled by calling PLIB_POWER_HLVDDisable.

Example

```
PLIB_POWER_HLVDModeSelect( POWER_ID_0, HLVD_MODE_HIGH_VOLTAGE_DETECTION );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Voltage Detection mode, either HVD or LVD

Function

void PLIB_POWER_HLVDModeSelect (POWER_MODULE_ID index, HLVD_MODE mode)

PLIB_POWER_HLVDStatusGet Function

Returns the HLVD event status.

File

plib_power.h

C

```
bool PLIB_POWER_HLVDStatusGet(POWER_MODULE_ID index);
```

Returns

- true - HLVD event interrupt is active
- false - HLVD event interrupt is not active

Description

This function returns the HLVD event interrupt status.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if(PLIB_POWER_HLVDStatusGet(POWER_ID_0))
{
    //HLVD event has occurred.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_HLVDStatusGet(POWER_MODULE_ID index)

PLIB_POWER_HLVDStopInIdleDisable Function

Continues HLVD operation when the device enters Idle mode.

File

plib_power.h

C

```
void PLIB_POWER_HLVDStopInIdleDisable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function continues HLVD operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_HLVDStopInIdleDisable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_HLVDStopInIdleDisable(POWER_MODULE_ID index)

PLIB_POWER_HLVDStopInIdleEnable Function

Discontinues HLVD operation when the device enters Idle mode.

File

plib_power.h

C

```
void PLIB_POWER_HLVDStopInIdleEnable(POWER_MODULE_ID index);
```

Returns

None.

Description

This function discontinues HLVD operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_POWER_HLVDStopInIdleEnable(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_POWER_HLVDStopInIdleEnable(POWER_MODULE_ID index)

PLIB_POWER_HLVDStopInIdleIsEnabled Function

Returns the Stop in Idle mode status of the HLVD operation.

File

plib_power.h

C

```
bool PLIB_POWER_HLVDStopInIdleIsEnabled(POWER_MODULE_ID index);
```

Returns

- true - HLVD discontinues operation when the device enters Idle mode
- false - HLVD continues operation when the device enters Idle mode

Description

This function returns whether HLVD continues or discontinues operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsHLVDStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
bool sidl_status;
sidl_status = PLIB_POWER_HLVDStopInIdleIsEnabled(POWER_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_POWER_HLVDStopInIdleIsEnabled(POWER_MODULE_ID index)

d) Deep Sleep Functions

PLIB_POWER_DeepSleepGPRRead Function

Reads 32-bit data from the Deep Sleep General Purpose Register.

File

plib_power.h

C

```
uint32_t PLIB_POWER_DeepSleepGPRRead(POWER_MODULE_ID index, DEEP_SLEEP_GPR gpr);
```

Returns

32-bit data from the selected Deep Sleep General Purpose Register.

Description

This function reads 32-bit value from the Deep Sleep General Purpose Register. User can store content in these registers (any application critical content) before entering the Deep Sleep/VBAT modes and read them upon exit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepGPROperation in your application to determine whether this feature is available.

Preconditions

The content of the DEEP_SLEEP_GPR_0 register is retained even in the Deep Sleep and VBAT modes, but the DEEP_SLEEP_GPR_1 through DEEP_SLEEP_GPR_32 registers are disabled by default in Deep Sleep and VBAT modes. They can be enabled with PLIB_POWER_DeepSleepGPRsRetentionEnable.

Example

```
uint32_t data;
//Enable power to DEEP_SLEEP_GPR_1 through DEEP_SLEEP_GPR_32 in Deep Sleep
PLIB_POWER_DeepSleepGPRsRetentionEnable(POWER_ID_0);

//Write 32-bit data into the DEEP_SLEEP_GPR_1
PLIB_POWER_DeepSleepGPRWrite(POWER_ID_0, DEEP_SLEEP_GPR_1, 0x1234);

//Enter the Deep Sleep mode and Exit
//Now read the data from DEEP_SLEEP_GPR_1
data = PLIB_POWER_DeepSleepGPRRead(POWER_ID_0, DEEP_SLEEP_GPR_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
gpr	The available Deep Sleep General Purpose Register

Function

uint32_t PLIB_POWER_DeepSleepGPRRead(POWER_MODULE_ID index, DEEP_SLEEP_GPR gpr)

PLIB_POWER_DeepSleepGPRWrite Function

Writes 32-bit data into the Deep Sleep General Purpose Register.

File

plib_power.h

C

```
void PLIB_POWER_DeepSleepGPRWrite(POWER_MODULE_ID index, DEEP_SLEEP_GPR gpr, uint32_t data);
```

Returns

None.

Description

This function accepts a 32-bit value to write into the Deep Sleep General Purpose Register. User can store content in these registers (any application critical content) before entering the Deep Sleep/VBAT modes and read them upon exit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_POWER_ExistsDeepSleepGPROperation in your application to determine whether this feature is available.

Preconditions

The content of the DEEP_SLEEP_GPR_0 register is retained even in the Deep Sleep and VBAT modes, but the DEEP_SLEEP_GPR_1 through DEEP_SLEEP_GPR_32 registers are disabled by default in Deep Sleep and VBAT modes. They can be enabled with PLIB_POWER_DeepSleepGPRsRetentionEnable.

Example

```
uint32_t data;
//Enable power to DEEP_SLEEP_GPR_1 through DEEP_SLEEP_GPR_32 in Deep Sleep
PLIB_POWER_DeepSleepGPRsRetentionEnable(POWER_ID_0);

//Write 32-bit data into the DEEP_SLEEP_GPR_1
PLIB_POWER_DeepSleepGPRWrite(POWER_ID_0, DEEP_SLEEP_GPR_1, 0x1234);

//Enter the Deep Sleep mode and Exit
//Now read the data from DEEP_SLEEP_GPR_1
data = PLIB_POWER_DeepSleepGPRRead(POWER_ID_0, DEEP_SLEEP_GPR_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
gpr	The available Deep Sleep General Purpose Register
data	32-bit data to write into the Deep Sleep General Purpose Register

Function

void PLIB_POWER_DeepSleepGPRWrite(POWER_MODULE_ID index, DEEP_SLEEP_GPR gpr, uint32_t data)

e) Feature Existence Functions

PLIB_POWER_ExistsDeepSleepMode Function

Identifies whether the DeepSleepModeControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepMode(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepModeControl feature is supported on the device
- false - The DeepSleepModeControl feature is not supported on the device

Description

This function identifies whether the DeepSleepModeControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeepSleepModeEnable
- PLIB_POWER_DeepSleepModeDisable
- PLIB_POWER_DeepSleepModeIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepMode(POWER_MODULE_ID index)

PLIB_POWER_ExistsIdleStatus Function

Identifies whether the IdleStatus feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsIdleStatus(POWER_MODULE_ID index);
```

Returns

- true - The IdleStatus feature is supported on the device
- false - The IdleStatus feature is not supported on the device

Description

This function identifies whether the IdleStatus feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeviceWasInIdleMode
- PLIB_POWER_ClearIdleStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsIdleStatus(POWER_MODULE_ID index)

PLIB_POWER_ExistsPeripheralModuleControl Function

Identifies whether the PeripheralModuleControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsPeripheralModuleControl(POWER_MODULE_ID index);
```

Returns

- true - The PeripheralModuleControl feature is supported on the device
- false - The PeripheralModuleControl feature is not supported on the device

Description

This function identifies whether the PeripheralModuleControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_PeripheralModuleDisable
- PLIB_POWER_PeripheralModuleEnable
- PLIB_POWER_PeripheralModuleIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsPeripheralModuleControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsSleepStatus Function

Identifies whether the SleepStatus feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsSleepStatus(POWER_MODULE_ID index);
```

Returns

- true - The SleepStatus feature is supported on the device
- false - The SleepStatus feature is not supported on the device

Description

This function identifies whether the SleepStatus feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeviceWasInSleepMode
- PLIB_POWER_ClearSleepStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsSleepStatus(POWER_MODULE_ID index)

PLIB_POWER_ExistsVoltageRegulatorControl Function

Identifies whether the VoltageRegulatorControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsVoltageRegulatorControl(POWER_MODULE_ID index);
```

Returns

- true - The VoltageRegulatorControl feature is supported on the device
- false - The VoltageRegulatorControl feature is not supported on the device

Description

This function identifies whether the VoltageRegulatorControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_VoltageRegulatorEnable
- PLIB_POWER_VoltageRegulatorDisable
- PLIB_POWER_VoltageRegulatorIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsVoltageRegulatorControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepEventStatus Function

Identifies whether the DeepSleepEventStatus feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepEventStatus(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepEventStatus feature is supported on the device
- false - The DeepSleepEventStatus feature is not supported on the device

Description

This function identifies whether the DeepSleepEventStatus feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeepSleepEventStatusGet
- PLIB_POWER_DeepSleepEventStatusClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepEventStatus(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepGPROperation Function

Identifies whether the DeepSleepGPROperation feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepGPROperation(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepGPROperation feature is supported on the device
- false - The DeepSleepGPROperation feature is not supported on the device

Description

This function identifies whether the DeepSleepGPROperation feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeepSleepGPRWrite
- PLIB_POWER_DeepSleepGPRRead

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepGPROperation(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepGPRsRetentionControl Function

Identifies whether the DeepSleepGPRsRetentionControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepGPRsRetentionControl(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepGPRsRetentionControl feature is supported on the device
- false - The DeepSleepGPRsRetentionControl feature is not supported on the device

Description

This function identifies whether the DeepSleepGPRsRetentionControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeepSleepGPRsRetentionEnable
- PLIB_POWER_DeepSleepGPRsRetentionDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepGPRsRetentionControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepModuleControl Function

Identifies whether the DeepSleepModuleControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepModuleControl(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepModuleControl feature is supported on the device
- false - The DeepSleepModuleControl feature is not supported on the device

Description

This function identifies whether the DeepSleepModuleControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeepSleepModuleEnable
- PLIB_POWER_DeepSleepModuleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepModuleControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepPortPinsStateControl Function

Identifies whether the DeepSleepPortPinsStateControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepPortPinsStateControl(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepPortPinsStateControl feature is supported on the device
- false - The DeepSleepPortPinsStateControl feature is not supported on the device

Description

This function identifies whether the DeepSleepPortPinsStateControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeepSleepPortPinsStateRetain
- PLIB_POWER_DeepSleepPortPinsStateRelease

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepPortPinsStateControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsHighVoltageOnVDD1V8 Function

Identifies whether the HighVoltageOnVDD1V8 feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHighVoltageOnVDD1V8(POWER_MODULE_ID index);
```

Returns

- true - The HighVoltageOnVDD1V8 feature is supported on the device
- false - The HighVoltageOnVDD1V8 feature is not supported on the device

Description

This function identifies whether the HighVoltageOnVDD1V8 feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HighVoltageOnVDD1V8HasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHighVoltageOnVDD1V8(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepModeOccurrence Function

Identifies whether the DeepSleepModeOccurrence feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepModeOccurrence(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepModeOccurrence feature is supported on the device
- false - The DeepSleepModeOccurrence feature is not supported on the device

Description

This function identifies whether the DeepSleepModeOccurrence feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_DeepSleepModeHasOccurred
- PLIB_POWER_DeepSleepStatusClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepModeOccurrence(POWER_MODULE_ID index)

PLIB_POWER_ExistsDeepSleepWakeupEventControl Function

Identifies whether the DeepSleepWakeupEventControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsDeepSleepWakeupEventControl(POWER_MODULE_ID index);
```

Returns

- true - The DeepSleepWakeupEventControl feature is supported on the device
- false - The DeepSleepWakeupEventControl feature is not supported on the device

Description

This function identifies whether the DeepSleepWakeupEventControl feature is available on the Power module. When this function returns true, these functions are supported on the device:

- PLIB_POWER_DeepSleepWakeupEventEnable
- PLIB_POWER_DeepSleepWakeupEventDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsDeepSleepWakeupEventControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsHLVDBandGapVoltageStability Function

Identifies whether the HLVDBandGapVoltageStability feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHLVDBandGapVoltageStability(POWER_MODULE_ID index);
```

Returns

- true - The HLVDBandGapVoltageStability feature is supported on the device
- false - The HLVDBandGapVoltageStability feature is not supported on the device

Description

This function identifies whether the HLVDBandGapVoltageStability feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HLVDBandGapVoltageIsStable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHLVDBandGapVoltageStability(POWER_MODULE_ID index)

PLIB_POWER_ExistsHLVDEnableControl Function

Identifies whether the HLVDEnableControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHLVDEnableControl(POWER_MODULE_ID index);
```

Returns

- true - The HLVDEnableControl feature is supported on the device
- false - The HLVDEnableControl feature is not supported on the device

Description

This function identifies whether the HLVDEnableControl feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HLVDEnable
- PLIB_POWER_HLVDDisable
- PLIB_POWER_HLVDIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHLVDEnableControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsHLVDLimitSelection Function

Identifies whether the HLVDLimitSelection feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHLVDLimitSelection(POWER_MODULE_ID index);
```

Returns

- true - The HLVDLimitSelection feature is supported on the device
- false - The HLVDLimitSelection feature is not supported on the device

Description

This function identifies whether the HLVDLimitSelection feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HLVDLimitSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHLVDLimitSelection(POWER_MODULE_ID index)

PLIB_POWER_ExistsHLVDModeControl Function

Identifies whether the HLVDModeControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHLVDModeControl(POWER_MODULE_ID index);
```

Returns

- true - The HLVDModeControl feature is supported on the device
- false - The HLVDModeControl feature is not supported on the device

Description

This function identifies whether the HLVDModeControl feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HLVDModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHLVDModeControl(POWER_MODULE_ID index)

PLIB_POWER_ExistsHLVDStatus Function

Identifies whether the HLVDStatus feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHLVDStatus(POWER_MODULE_ID index);
```

Returns

- true - The HLVDStatus feature is supported on the device
- false - The HLVDStatus feature is not supported on the device

Description

This function identifies whether the HLVDStatus feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HLVDStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHLVDStatus(POWER_MODULE_ID index)

PLIB_POWER_ExistsHLVDStopInIdleControl Function

Identifies whether the HLVDStopInIdleControl feature exists on the Power module.

File

plib_power.h

C

```
bool PLIB_POWER_ExistsHLVDStopInIdleControl(POWER_MODULE_ID index);
```

Returns

- true - The HLVDStopInIdleControl feature is supported on the device
- false - The HLVDStopInIdleControl feature is not supported on the device

Description

This function identifies whether the HLVDStopInIdleControl feature is available on the Power module. When this function returns true, this function is supported on the device:

- PLIB_POWER_HLVDStopInIdleEnable
- PLIB_POWER_HLVDStopInIdleDisable
- PLIB_POWER_HLVDStopInIdleIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_POWER_ExistsHLVDStopInIdleControl(POWER_MODULE_ID index)

f) Data Types and Constants

DEEP_SLEEP_EVENT Enumeration

Lists the events that occurred during Deep Sleep mode.

File

help_plib_power.h

C

```
typedef enum {
  DEEP_SLEEP_EVENT_BOR,
  DEEP_SLEEP_EVENT_RTCC_ALARM,
  DEEP_SLEEP_EVENT_EXTERNAL_INTERRUPT,
  DEEP_SLEEP_EVENT_FAULT_DETECTION,
  DEEP_SLEEP_EVENT_WDT_TIMEOUT,
  DEEP_SLEEP_EVENT_MCLR
} DEEP_SLEEP_EVENT;
```

Members

Members	Description
DEEP_SLEEP_EVENT_BOR	BOR event has occurred during Deep Sleep
DEEP_SLEEP_EVENT_RTCC_ALARM	RTCC alarm has occurred during Deep Sleep
DEEP_SLEEP_EVENT_EXTERNAL_INTERRUPT	Interrupt-On-Change on pin INT0 has occurred during Deep Sleep
DEEP_SLEEP_EVENT_FAULT_DETECTION	Fault has occurred during Deep Sleep and some Deep Sleep configuration settings may have been corrupted
DEEP_SLEEP_EVENT_WDT_TIMEOUT	DSWDT time-out has occurred during Deep Sleep
DEEP_SLEEP_EVENT_MCLR	Master Clear (MCLR) has occurred during Deep Sleep

Description

Deep Sleep events

This enumeration lists the events that occurred during Deep Sleep mode.

Remarks

Not all events exist on all devices. Please refer to the specific device data sheet for availability.

DEEP_SLEEP_GPR Enumeration

Lists the Deep Sleep General Purpose Registers (GPRs).

File

help_plib_power.h

C

```
typedef enum {
  DEEP_SLEEP_GPR_0,
  DEEP_SLEEP_GPR_1,
  DEEP_SLEEP_GPR_2,
  DEEP_SLEEP_GPR_3,
  DEEP_SLEEP_GPR_4,
  DEEP_SLEEP_GPR_5,
  DEEP_SLEEP_GPR_6,
  DEEP_SLEEP_GPR_7,
  DEEP_SLEEP_GPR_8,
  DEEP_SLEEP_GPR_9,
  DEEP_SLEEP_GPR_10,
  DEEP_SLEEP_GPR_11,
  DEEP_SLEEP_GPR_12,
  DEEP_SLEEP_GPR_13,
  DEEP_SLEEP_GPR_14,
  DEEP_SLEEP_GPR_15,
  DEEP_SLEEP_GPR_16,
  DEEP_SLEEP_GPR_17,
```

```
    DEEP_SLEEP_GPR_18,
    DEEP_SLEEP_GPR_19,
    DEEP_SLEEP_GPR_20,
    DEEP_SLEEP_GPR_21,
    DEEP_SLEEP_GPR_22,
    DEEP_SLEEP_GPR_23,
    DEEP_SLEEP_GPR_24,
    DEEP_SLEEP_GPR_25,
    DEEP_SLEEP_GPR_26,
    DEEP_SLEEP_GPR_27,
    DEEP_SLEEP_GPR_28,
    DEEP_SLEEP_GPR_29,
    DEEP_SLEEP_GPR_30,
    DEEP_SLEEP_GPR_31,
    DEEP_SLEEP_GPR_32
} DEEP_SLEEP_GPR;
```

Members

Members	Description
DEEP_SLEEP_GPR_0	Deep Sleep General Purpose Register 0. This will be active in Deep Sleep by default.
DEEP_SLEEP_GPR_1	Deep Sleep General Purpose Register 1.
DEEP_SLEEP_GPR_2	Deep Sleep General Purpose Register 2.
DEEP_SLEEP_GPR_3	Deep Sleep General Purpose Register 3.
DEEP_SLEEP_GPR_4	Deep Sleep General Purpose Register 4.
DEEP_SLEEP_GPR_5	Deep Sleep General Purpose Register 5.
DEEP_SLEEP_GPR_6	Deep Sleep General Purpose Register 6.
DEEP_SLEEP_GPR_7	Deep Sleep General Purpose Register 7.
DEEP_SLEEP_GPR_8	Deep Sleep General Purpose Register 8.
DEEP_SLEEP_GPR_9	Deep Sleep General Purpose Register 9.
DEEP_SLEEP_GPR_10	Deep Sleep General Purpose Register 10.
DEEP_SLEEP_GPR_11	Deep Sleep General Purpose Register 11.
DEEP_SLEEP_GPR_12	Deep Sleep General Purpose Register 12.
DEEP_SLEEP_GPR_13	Deep Sleep General Purpose Register 13.
DEEP_SLEEP_GPR_14	Deep Sleep General Purpose Register 14.
DEEP_SLEEP_GPR_15	Deep Sleep General Purpose Register 15.
DEEP_SLEEP_GPR_16	Deep Sleep General Purpose Register 16.
DEEP_SLEEP_GPR_17	Deep Sleep General Purpose Register 17.
DEEP_SLEEP_GPR_18	Deep Sleep General Purpose Register 18.
DEEP_SLEEP_GPR_19	Deep Sleep General Purpose Register 19.
DEEP_SLEEP_GPR_20	Deep Sleep General Purpose Register 20.
DEEP_SLEEP_GPR_21	Deep Sleep General Purpose Register 21.
DEEP_SLEEP_GPR_22	Deep Sleep General Purpose Register 22.
DEEP_SLEEP_GPR_23	Deep Sleep General Purpose Register 23.
DEEP_SLEEP_GPR_24	Deep Sleep General Purpose Register 24.
DEEP_SLEEP_GPR_25	Deep Sleep General Purpose Register 25.
DEEP_SLEEP_GPR_26	Deep Sleep General Purpose Register 26.
DEEP_SLEEP_GPR_27	Deep Sleep General Purpose Register 27.
DEEP_SLEEP_GPR_28	Deep Sleep General Purpose Register 28.
DEEP_SLEEP_GPR_29	Deep Sleep General Purpose Register 29.
DEEP_SLEEP_GPR_30	Deep Sleep General Purpose Register 30.
DEEP_SLEEP_GPR_31	Deep Sleep General Purpose Register 31.
DEEP_SLEEP_GPR_32	Deep Sleep General Purpose Register 32.

Description

Deep Sleep General Purpose Register set.

This enumeration lists the Deep Sleep General Purpose Registers. These registers can be used to save some application critical content while entering into Deep Sleep mode and can be read upon exit.

Remarks

Not all GPRs exist on all devices. Please refer to the specific device data sheet for availability.

DEEP_SLEEP_MODULE Enumeration

Lists the modules that can be enabled/disabled during Deep Sleep mode.

File

help_plib_power.h

C

```
typedef enum {
  DEEP_SLEEP_MODULE_RTCC
} DEEP_SLEEP_MODULE;
```

Members

Members	Description
DEEP_SLEEP_MODULE_RTCC	RTCC module

Description

Deep Sleep Module

This enumeration lists the modules that can be enabled/disabled during Deep Sleep mode.

Remarks

Not all modules exist on all devices. Please refer to the specific device data sheet for availability.

DEEP_SLEEP_WAKE_UP_EVENT Enumeration

Lists the events that can be used to wake the device from Deep Sleep mode.

File

help_plib_power.h

C

```
typedef enum {
  DEEP_SLEEP_WAKE_UP_EVENT_RTCC,
  DEEP_SLEEP_WAKE_UP_EVENT_EXTERNAL_INTERRUPT
} DEEP_SLEEP_WAKE_UP_EVENT;
```

Members

Members	Description
DEEP_SLEEP_WAKE_UP_EVENT_RTCC	Deep Sleep wake-up by RTCC alarm event
DEEP_SLEEP_WAKE_UP_EVENT_EXTERNAL_INTERRUPT	Deep Sleep wake-up by Interrupt-On-Change (IOC) on INT0 pin

Description

Deep Sleep Wake-up events

This enumeration lists the events that can be used to wake the device from Deep Sleep mode.

Remarks

Not all events exist on all devices. Please refer to the specific device data sheet for availability.

POWER_MODULE Enumeration

List of the modules whose power can be controlled.

File

help_plib_power.h

C

```
typedef enum {
  POWER_MODULE_ADC1,
  POWER_MODULE_CTMU,
  POWER_MODULE_CVR,
  POWER_MODULE_HLVD,
```

```
    POWER_MODULE_HVD1V8,
    POWER_MODULE_CMP1,
    POWER_MODULE_CMP2,
    POWER_MODULE_CMP3,
    POWER_MODULE_IC1,
    POWER_MODULE_IC2,
    POWER_MODULE_IC3,
    POWER_MODULE_IC4,
    POWER_MODULE_IC5,
    POWER_MODULE_IC6,
    POWER_MODULE_IC7,
    POWER_MODULE_IC8,
    POWER_MODULE_IC9,
    POWER_MODULE_OC1,
    POWER_MODULE_OC2,
    POWER_MODULE_OC3,
    POWER_MODULE_OC4,
    POWER_MODULE_OC5,
    POWER_MODULE_OC6,
    POWER_MODULE_OC7,
    POWER_MODULE_OC8,
    POWER_MODULE_OC9,
    POWER_MODULE_TMR1,
    POWER_MODULE_TMR2,
    POWER_MODULE_TMR3,
    POWER_MODULE_TMR4,
    POWER_MODULE_TMR5,
    POWER_MODULE_TMR6,
    POWER_MODULE_TMR7,
    POWER_MODULE_TMR8,
    POWER_MODULE_TMR9,
    POWER_MODULE_UART1,
    POWER_MODULE_UART2,
    POWER_MODULE_UART3,
    POWER_MODULE_UART4,
    POWER_MODULE_UART5,
    POWER_MODULE_UART6,
    POWER_MODULE_SPI1,
    POWER_MODULE_SPI2,
    POWER_MODULE_SPI3,
    POWER_MODULE_SPI4,
    POWER_MODULE_SPI5,
    POWER_MODULE_SPI6,
    POWER_MODULE_I2C1,
    POWER_MODULE_I2C2,
    POWER_MODULE_I2C3,
    POWER_MODULE_I2C4,
    POWER_MODULE_I2C5,
    POWER_MODULE_USB,
    POWER_MODULE_CAN1,
    POWER_MODULE_CAN2,
    POWER_MODULE_RTCC,
    POWER_MODULE_REF_CLK_OUTPUT,
    POWER_MODULE_REF_CLK_OUTPUT1,
    POWER_MODULE_REF_CLK_OUTPUT2,
    POWER_MODULE_REF_CLK_OUTPUT3,
    POWER_MODULE_REF_CLK_OUTPUT4,
    POWER_MODULE_REF_CLK_OUTPUT5,
    POWER_MODULE_PMP,
    POWER_MODULE_EBI,
    POWER_MODULE_GPU,
    POWER_MODULE_GLCD,
    POWER_MODULE_SDHC,
    POWER_MODULE_SQI,
    POWER_MODULE_ETHERNET,
    POWER_MODULE_DMA,
    POWER_MODULE_RANDOM_NUM_GENERATOR,
    POWER_MODULE_DDR2,
    POWER_MODULE_CRYPTO
} POWER_MODULE;
```

Members

Members	Description
POWER_MODULE_ADC1	ADC module

POWER_MODULE_CTMU	Charge Time Measurement Unit module
POWER_MODULE_CVR	Comparator Voltage Reference module
POWER_MODULE_HLVD	Low-Voltage Detection module
POWER_MODULE_HVD1V8	High-Voltage Detection module
POWER_MODULE_CMP1	Comparator module 1
POWER_MODULE_CMP2	Comparator module 2
POWER_MODULE_CMP3	Comparator module 3
POWER_MODULE_IC1	Input Capture 1 module
POWER_MODULE_IC2	Input Capture 2 module
POWER_MODULE_IC3	Input Capture 3 module
POWER_MODULE_IC4	Input Capture 4 module
POWER_MODULE_IC5	Input Capture 5 module
POWER_MODULE_IC6	Input Capture 6 module
POWER_MODULE_IC7	Input Capture 7 module
POWER_MODULE_IC8	Input Capture 8 module
POWER_MODULE_IC9	Input Capture 9 module
POWER_MODULE_OC1	Output Compare 1 module
POWER_MODULE_OC2	Output Compare 2 module
POWER_MODULE_OC3	Output Compare 3 module
POWER_MODULE_OC4	Output Compare 4 module
POWER_MODULE_OC5	Output Compare 5 module
POWER_MODULE_OC6	Output Compare 6 module
POWER_MODULE_OC7	Output Compare 7 module
POWER_MODULE_OC8	Output Compare 8 module
POWER_MODULE_OC9	Output Compare 9 module
POWER_MODULE_TMR1	Timer1 module
POWER_MODULE_TMR2	Timer2 module
POWER_MODULE_TMR3	Timer3 module
POWER_MODULE_TMR4	Timer4 module
POWER_MODULE_TMR5	Timer5 module
POWER_MODULE_TMR6	Timer6 module
POWER_MODULE_TMR7	Timer7 module
POWER_MODULE_TMR8	Timer8 module
POWER_MODULE_TMR9	Timer9 module
POWER_MODULE_UART1	UART1 module
POWER_MODULE_UART2	UART2 module
POWER_MODULE_UART3	UART3 module
POWER_MODULE_UART4	UART4 module
POWER_MODULE_UART5	UART5 module
POWER_MODULE_UART6	UART6 module
POWER_MODULE_SPI1	SPI1 module
POWER_MODULE_SPI2	SPI2 module
POWER_MODULE_SPI3	SPI3 module
POWER_MODULE_SPI4	SPI4 module
POWER_MODULE_SPI5	SPI5 module
POWER_MODULE_SPI6	SPI6 module
POWER_MODULE_I2C1	I2C1 module
POWER_MODULE_I2C2	I2C2 module
POWER_MODULE_I2C3	I2C3 module
POWER_MODULE_I2C4	I2C4 module
POWER_MODULE_I2C5	I2C5 module
POWER_MODULE_USB	USB module
POWER_MODULE_CAN1	CAN1 module
POWER_MODULE_CAN2	CAN2 module
POWER_MODULE_RTCC	Real-Time Clock and Calender module

POWER_MODULE_REF_CLK_OUTPUT	Reference Clock Output module
POWER_MODULE_REF_CLK_OUTPUT1	Reference Clock Output module 1
POWER_MODULE_REF_CLK_OUTPUT2	Reference Clock Output module 2
POWER_MODULE_REF_CLK_OUTPUT3	Reference Clock Output module 3
POWER_MODULE_REF_CLK_OUTPUT4	Reference Clock Output module 4
POWER_MODULE_REF_CLK_OUTPUT5	Reference Clock Output module 5
POWER_MODULE_PMP	Parallel Master Port module
POWER_MODULE_EBI	External Bus Interface module
POWER_MODULE_GPU	Graphics Processing Unit module
POWER_MODULE_GLCD	Graphics LCD module
POWER_MODULE_SDHC	Secure Digital Host Controller module
POWER_MODULE_SQI	Serial Quad Interface module
POWER_MODULE_ETHERNET	Ethernet module
POWER_MODULE_DMA	Data Memory Access module
POWER_MODULE_RANDOM_NUM_GENERATOR	Random Number Generator module
POWER_MODULE_DDR2	DDR2 module
POWER_MODULE_CRYPTO	Cryptographic module

Description

POWER module enumeration

This enumeration lists the modules whose power can be controlled by the Peripheral Module Disable (PMD) Registers.

Remarks

Not all modules exist on all devices. Please refer to the specific device data sheet for availability.

POWER_MODULE_ID Enumeration

Identifies the supported Power modules.

File

help_plib_power.h

C

```
typedef enum {
  POWER_ID_0,
  POWER_NUMBER_OF_MODULES
} POWER_MODULE_ID;
```

Members

Members	Description
POWER_ID_0	POWER Module ID
POWER_NUMBER_OF_MODULES	Number of available POWER modules.

Description

Power Module ID

This enumeration identifies the Power modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on a Microchip microcontroller.

Remarks

The caller should not rely on the specific numbers assigned to any of these values, as they may change from one processor to the next.

Not all modules are available on all microcontrollers. Refer to the "Power-Saving Features" chapter in the specific device data sheet to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

HLVD_LIMIT Enumeration

Lists the voltage limits for the HLVD module.

File

help_plib_power.h

C

```
typedef enum {
  HLVD_LIMIT_TRIP_POINT_4,
  HLVD_LIMIT_TRIP_POINT_5,
  HLVD_LIMIT_TRIP_POINT_6,
  HLVD_LIMIT_TRIP_POINT_7,
  HLVD_LIMIT_TRIP_POINT_8,
  HLVD_LIMIT_TRIP_POINT_9,
  HLVD_LIMIT_TRIP_POINT_10,
  HLVD_LIMIT_TRIP_POINT_11,
  HLVD_LIMIT_TRIP_POINT_12,
  HLVD_LIMIT_TRIP_POINT_13,
  HLVD_LIMIT_TRIP_POINT_14,
  HLVD_LIMIT_ANALOG_INPUT_ON_HLVDIN
} HLVD_LIMIT;
```

Members

Members	Description
HLVD_LIMIT_TRIP_POINT_4	Voltage limit is trip point 4
HLVD_LIMIT_TRIP_POINT_5	Voltage limit is trip point 5
HLVD_LIMIT_TRIP_POINT_6	Voltage limit is trip point 6
HLVD_LIMIT_TRIP_POINT_7	Voltage limit is trip point 7
HLVD_LIMIT_TRIP_POINT_8	Voltage limit is trip point 8
HLVD_LIMIT_TRIP_POINT_9	Voltage limit is trip point 9
HLVD_LIMIT_TRIP_POINT_10	Voltage limit is trip point 10
HLVD_LIMIT_TRIP_POINT_11	Voltage limit is trip point 11
HLVD_LIMIT_TRIP_POINT_12	Voltage limit is trip point 12
HLVD_LIMIT_TRIP_POINT_13	Voltage limit is trip point 13
HLVD_LIMIT_TRIP_POINT_14	Voltage limit is trip point 14
HLVD_LIMIT_ANALOG_INPUT_ON_HLVDIN	Voltage limit is external analog voltage on the pin HLVDIN

Description

High/Low-Voltage Detection limits

This enumeration lists the voltage limits that can be used as reference to High/Low-Voltage Detection on VDD.

Refer to the â€œElectrical Characteristicsâ€• chapter of the specific device data sheet for the actual trip points (i.e., voltages).

Remarks

Not all voltage limits exist on all devices. Please refer to the specific device data sheet for availability.

HLVD_MODE Enumeration

Lists the modes for the High/Low Voltage Detection (HLVD) module.

File

help_plib_power.h

C

```
typedef enum {
  HLVD_MODE_LOW_VOLTAGE_DETECTION,
  HLVD_MODE_HIGH_VOLTAGE_DETECTION
} HLVD_MODE;
```

Members

Members	Description
HLVD_MODE_LOW_VOLTAGE_DETECTION	In Low-Voltage Detection (LVD) mode, an event occurs when the voltage equals or falls below the voltage limit.
HLVD_MODE_HIGH_VOLTAGE_DETECTION	In High-Voltage Detection (HVD) mode, an event occurs when the voltage equals or exceeds the voltage limit.

Description

High/Low-Voltage Detection modes

This enumeration lists the modes those can be used for High/Low Voltage Detection on VDD.

Remarks

None.

Files

Files

Name	Description
plib_power.h	Defines the Power Peripheral Library interface.
help_plib_power.h	This is file help_plib_power.h.

Description

This section lists the source and header files used by the library.

plib_power.h

Defines the Power Peripheral Library interface.

Functions

	Name	Description
	PLIB_POWER_ClearIdleStatus	Clears the Idle mode status of the device.
	PLIB_POWER_ClearSleepStatus	Clear the Sleep mode status bit of the device.
	PLIB_POWER_DeepSleepEventStatusClear	Clear any events that occurred during Deep Sleep mode.
	PLIB_POWER_DeepSleepEventStatusGet	Returns the events that occurred during Deep Sleep mode.
	PLIB_POWER_DeepSleepGPRRead	Reads 32-bit data from the Deep Sleep General Purpose Register.
	PLIB_POWER_DeepSleepGPRsRetentionDisable	Disables the General Purpose Registers retention.
	PLIB_POWER_DeepSleepGPRsRetentionEnable	Enables the General Purpose Registers retention.
	PLIB_POWER_DeepSleepGPRWrite	Writes 32-bit data into the Deep Sleep General Purpose Register.
	PLIB_POWER_DeepSleepModeDisable	Disables Deep Sleep mode.
	PLIB_POWER_DeepSleepModeEnable	Enables Deep Sleep mode.
	PLIB_POWER_DeepSleepModeHasOccurred	Returns the Deep Sleep mode status of the device.
	PLIB_POWER_DeepSleepModeIsEnabled	Returns the enable/disable status of Deep Sleep mode.
	PLIB_POWER_DeepSleepModuleDisable	Disables the module in Deep Sleep mode.
	PLIB_POWER_DeepSleepModuleEnable	Enables the module in Deep Sleep mode.
	PLIB_POWER_DeepSleepPortPinsStateRelease	Releases I/O pins upon wake-up from Deep Sleep mode.
	PLIB_POWER_DeepSleepPortPinsStateRetain	Enables the I/O pins to retain their previous states upon wake-up from Deep Sleep mode.
	PLIB_POWER_DeepSleepStatusClear	Clears the Deep Sleep mode status bit of the device.
	PLIB_POWER_DeepSleepWakeupEventDisable	Disables wake-up from Deep Sleep mode by the selected event.
	PLIB_POWER_DeepSleepWakeupEventEnable	Enables wake-up from the Deep Sleep mode by the selected event.
	PLIB_POWER_DeviceWasInIdleMode	Returns the Idle mode status of the device.
	PLIB_POWER_DeviceWasInSleepMode	Returns the Sleep mode status of the device.
	PLIB_POWER_ExistsDeepSleepEventStatus	Identifies whether the DeepSleepEventStatus feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepGPROperation	Identifies whether the DeepSleepGPROperation feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepGPRsRetentionControl	Identifies whether the DeepSleepGPRsRetentionControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepMode	Identifies whether the DeepSleepModeControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepModeOccurrence	Identifies whether the DeepSleepModeOccurrence feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepModuleControl	Identifies whether the DeepSleepModuleControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepPortPinsStateControl	Identifies whether the DeepSleepPortPinsStateControl feature exists on the Power module.
	PLIB_POWER_ExistsDeepSleepWakeupEventControl	Identifies whether the DeepSleepWakeupEventControl feature exists on the Power module.

	PLIB_POWER_ExistsHighVoltageOnVDD1V8	Identifies whether the HighVoltageOnVDD1V8 feature exists on the Power module.
	PLIB_POWER_ExistsHLVDBandGapVoltageStability	Identifies whether the HLVDBandGapVoltageStability feature exists on the Power module.
	PLIB_POWER_ExistsHLVDEnableControl	Identifies whether the HLVDEnableControl feature exists on the Power module.
	PLIB_POWER_ExistsHLVDLimitSelection	Identifies whether the HLVDLimitSelection feature exists on the Power module.
	PLIB_POWER_ExistsHLVDModeControl	Identifies whether the HLVDModeControl feature exists on the Power module.
	PLIB_POWER_ExistsHLVDStatus	Identifies whether the HLVDStatus feature exists on the Power module.
	PLIB_POWER_ExistsHLVDStopInIdleControl	Identifies whether the HLVDStopInIdleControl feature exists on the Power module.
	PLIB_POWER_ExistsIdleStatus	Identifies whether the IdleStatus feature exists on the Power module.
	PLIB_POWER_ExistsPeripheralModuleControl	Identifies whether the PeripheralModuleControl feature exists on the Power module.
	PLIB_POWER_ExistsSleepStatus	Identifies whether the SleepStatus feature exists on the Power module.
	PLIB_POWER_ExistsVoltageRegulatorControl	Identifies whether the VoltageRegulatorControl feature exists on the Power module.
	PLIB_POWER_HighVoltageOnVDD1V8HasOccurred	Returns the DDR2 High Voltage Detection status of the device.
	PLIB_POWER_HLVDBandGapVoltageIsStable	Returns the status of Band Gap voltage.
	PLIB_POWER_HLVDDisable	Disables High/Low-Voltage Detection on VDD.
	PLIB_POWER_HLVDEnable	Enables High/Low-Voltage Detection (HLVD) on VDD.
	PLIB_POWER_HLVDIsEnabled	Returns the enable/disable status of High/Low-Voltage Detection on VDD.
	PLIB_POWER_HLVDLimitSelect	Selects the HLVD limit.
	PLIB_POWER_HLVDModeSelect	Selects the Voltage Detection mode.
	PLIB_POWER_HLVDStatusGet	Returns the HLVD event status.
	PLIB_POWER_HLVDStopInIdleDisable	Continues HLVD operation when the device enters Idle mode.
	PLIB_POWER_HLVDStopInIdleEnable	Discontinues HLVD operation when the device enters Idle mode.
	PLIB_POWER_HLVDStopInIdleIsEnabled	Returns the Stop in Idle mode status of the HLVD operation.
	PLIB_POWER_PeripheralModuleDisable	Disable the power supply for the module selected in the source.
	PLIB_POWER_PeripheralModuleEnable	Enable the power supply for the module selected in the source.
	PLIB_POWER_PeripheralModuleIsEnabled	Checks to see whether or not the selected peripheral is enabled.
	PLIB_POWER_VoltageRegulatorDisable	Disables the voltage regulator during Sleep mode.
	PLIB_POWER_VoltageRegulatorEnable	Enable the voltage regulator during Sleep mode.
	PLIB_POWER_VoltageRegulatorIsEnabled	Provides the status of the voltage regulator during Sleep mode.

Description

Power Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Power Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Power Controller module.

File Name

plib_power.h

Company

Microchip Technology Inc.

help_plib_power.h

Enumerations

	Name	Description
	DEEP_SLEEP_EVENT	Lists the events that occurred during Deep Sleep mode.
	DEEP_SLEEP_GPR	Lists the Deep Sleep General Purpose Registers (GPRs).
	DEEP_SLEEP_MODULE	Lists the modules that can be enabled/disabled during Deep Sleep mode.
	DEEP_SLEEP_WAKE_UP_EVENT	Lists the events that can be used to wake the device from Deep Sleep mode.
	HLVD_LIMIT	Lists the voltage limits for the HLVD module.
	HLVD_MODE	Lists the modes for the High/Low Voltage Detection (HLVD) module.

	POWER_MODULE	List of the modules whose power can be controlled.
	POWER_MODULE_ID	Identifies the supported Power modules.

Description

This is file help_plib_power.h.

Reset Peripheral Library

This section describes the Reset Peripheral Library.

Introduction

This library provides a low-level abstraction of the Reset Controller modules on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The Reset Controller is a key component of a microcontroller. The Reset Controller gives the reset status of the device at any point of time.

The following are the key features present on a Reset Controller:

- **Reset Source Configuration**: Provides the ability to enable and disable a source that may reset the device
- **Reset Status:** Identifies the source of a reset and clears it

A microcontroller can support one or more of the previously described reset modes.

Note: Please refer to the **"Resets"** chapter in the specific device data sheet to determine the exact set of supported features.

Using the Library

This topic describes the basic architecture of the Reset Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_reset.h`

The interface to the Reset Peripheral Library is defined in the `plib_reset.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Reset Peripheral Library must include `peripheral.h`.

Peripheral Module IDs

Peripheral libraries are indexed to allow a single library to control any number of instances of a peripheral in a single microcontroller. To support this, the first parameter to each operation in a peripheral library is the module instance ID. The module instance ID is defined by an enumeration that is defined in the processor-specific header files (included by the library's interface header). Not all microcontrollers will have all instances of the module listed in this enumeration. Please refer to the **"Resets"** chapter in the specific device data sheet for availability.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

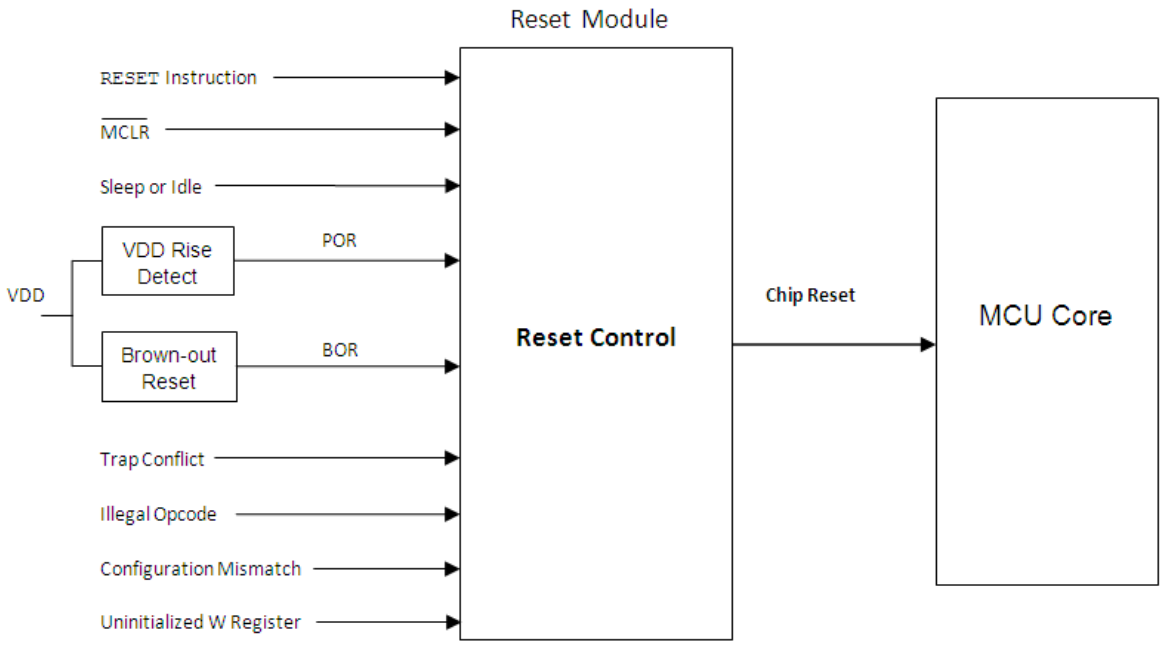
Hardware Abstraction Model

This library provides a low-level abstraction of the Reset Controller module on Microchip PIC microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The following figure illustrates the hardware abstraction model for the Reset Peripheral Library.

Hardware Abstraction Model



The **Reset Controller** receives requests from multiple **Reset Sources**. The **Reset Controller** controls the overall operation of the reset controller.

Source Enable Configuration enables or disables a particular source of reset. Some of the reset sources such as Traps are not controlled by the users, whereas others can be user controlled.

Note: Not all features are available on all devices. Refer to the **"Resets"** chapter in the specific device data sheet for availability.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Reset Controller module.

Library Interface Section	Description
Initialization Functions	This section provides a set of functions for configuring and enabling a reset.
Status Functions	This section provides a function to read the status of a particular reset.
NMI Events Functions	This section provides functions for NMI events.
Feature Existence Functions	This section provides functions that determine whether certain features exist.

Note: Please refer to the **"Resets"** chapter in the specific device data sheet to determine the exact set of supported features.

How the Library Works

Initialization

The steps that are required to initialize the Reset Controller module vary for different microcontrollers. Refer to the **"Resets"** chapter in the specific device data sheet for the correct initialization sequence. The following information provides a general overview.

Description

Reset Source Setup

Some reset sources need to be enabled for it to be able to Reset the device. Use the function PLIB_RESET_SoftwareResetEnable to enable the reset source.

Reset Status

The status of the reset indicates which source generated the reset. The status of the reset source can be obtained using the function PLIB_RESET_ReasonGet. The reason for the reset can be cleared using the function PLIB_RESET_ReasonClear.

Configuring the Library

The library is configured for the supported Reset Controller modules when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Initialization Functions

	Name	Description
	PLIB_RESET_SoftwareResetEnable	Enables the software reset.

b) Status Functions

	Name	Description
	PLIB_RESET_ReasonClear	Clears the RCON register.
	PLIB_RESET_ReasonGet	Returns the reason for a reset.
	PLIB_RESET_ConfigRegReadErrorGet	Gets the Configuration register read error.
	PLIB_RESET_WdtTimeOutHasOccurredInSleep	Returns the state of the device when WDT time-out occurred

c) NMI Events Functions

	Name	Description
	PLIB_RESET_NmiCounterValueGet	Gets the NMI counter value.
	PLIB_RESET_NmiCounterValueSet	Sets the NMI counter.
	PLIB_RESET_NmiEventClear	Clears the NMI event
	PLIB_RESET_NmiEventTrigger	Triggers the NMI event.
	PLIB_RESET_NmiReasonGet	Returns the reason for the Non-Maskable Interrupt (NMI).

d) Feature Existence Functions

	Name	Description
	PLIB_RESET_ExistsResetReasonStatus	Identifies whether the ResetReasonStatus feature exists on the Reset module.
	PLIB_RESET_ExistsSoftwareResetTrigger	Identifies whether the SoftwareResetTrigger feature exists on the Reset module.
	PLIB_RESET_ExistsConfigRegReadError	Identifies whether the ConfigRegReadError feature exists on the Reset module.
	PLIB_RESET_ExistsNmiControl	Identifies whether the NmiControl feature exists on the Reset module.
	PLIB_RESET_ExistsNmiCounter	Identifies whether the NmiCounter feature exists on the Reset module.
	PLIB_RESET_ExistsWdtoInSleep	Identifies whether the WdtoInSleep feature exists on the Reset module.

e) Data Types and Constants

	Name	Description
	RESET_MODULE_ID	Identifies the supported Reset modules.
	RESET_REASON	Lists the reset sources.
	RESET_CONFIG_REG_READ_ERROR	Lists the Configuration register read errors.
	RESET_NMI_COUNT_TYPE	Defines NMI counter data type
	RESET_NMI_REASON	Lists the NMI reasons.

Description

This section describes the Application Programming Interface (API) functions of the Reset Peripheral Library.
Refer to each section for a detailed description.

a) Initialization Functions

PLIB_RESET_SoftwareResetEnable Function

Enables the software reset.

File

plib_reset.h

C

```
void PLIB_RESET_SoftwareResetEnable(RESET_MODULE_ID index);
```

Returns

None.

Description

This function triggers a software reset.

Remarks

This function implements an operation of the SoftwareResetTrigger feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsSoftwareResetTrigger function in your application to determine whether this feature is available.

Preconditions

The system unlock sequence must be performed before calling PLIB_RESET_SoftwareResetEnable.

Example

```
//Call system service to unlock the system
PLIB_RESET_SoftwareResetEnable( RESET_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RESET_SoftwareResetEnable (RESET_MODULE_ID index)

b) Status Functions

PLIB_RESET_ReasonClear Function

Clears the RCON register.

File

plib_reset.h

C

```
void PLIB_RESET_ReasonClear(RESET_MODULE_ID index, RESET_REASON reason);
```

Returns

None

Description

This function clears the particular reset bit in the RCON register. Multiple reasons for a reset can be ORed together and given as an input parameter to clear them simultaneously.

Remarks

This function implements an operation of the ResetReasonStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsResetReasonStatus function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RESET_ReasonClear( RESET_ID_0, RESET_REASON_MCLR | RESET_REASON_POWERON );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
reason	Select the reset type from enum RESET_REASON

Function

void PLIB_RESET_ReasonClear(RESET_MODULE_ID index, RESET_REASON reason)

PLIB_RESET_ReasonGet Function

Returns the reason for a reset.

File

plib_reset.h

C

```
RESET_REASON PLIB_RESET_ReasonGet(RESET_MODULE_ID index);
```

Returns

RESET_REASON - Type of reset (when there is more than one reason for a reset, this function will logically OR (bitwise) the reasons and return the value.

Description

This function returns the reason a reset has occurred for the selected device.

Remarks

This function implements an operation of the ResetReasonStatus feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsResetReasonStatus function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
RESET_REASON type;

type = PLIB_RESET_ReasonGet ( RESET_ID_0 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

RESET_REASON PLIB_RESET_ReasonGet (RESET_MODULE_ID index)

PLIB_RESET_ConfigRegReadErrorGet Function

Gets the Configuration register read error.

File

plib_reset.h

C

```
RESET_CONFIG_REG_READ_ERROR PLIB_RESET_ConfigRegReadErrorGet(RESET_MODULE_ID index);
```

Returns

RESET_CONFIG_REG_READ_ERROR - Type of Configuration Register Read Error

Description

This function returns the Configuration register read error, if any, after the reset.

Remarks

This function implements an operation of the ConfigRegReadError feature. This feature may not be available on all devices. Refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsConfigRegReadError function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_RESET_ConfigRegReadErrorGet( RESET_ID_0 )== PRIMARY_CONFIG_REG_READ_ERROR )
{
    //Do Something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

RESET_CONFIG_REG_READ_ERROR PLIB_RESET_ConfigRegReadErrorGet(RESET_MODULE_ID index);

PLIB_RESET_WdtTimeOutHasOccurredInSleep Function

Returns the state of the device when WDT time-out occurred

File

plib_reset.h

C

```
bool PLIB_RESET_WdtTimeOutHasOccurredInSleep(RESET_MODULE_ID index);
```

Returns

- true - The device was in Sleep mode when a WDT time-out occurred
- false - The device was not in Sleep mode when a WDT time-out occurred

Description

This function returns whether or not the the device was in Sleep mode when a WDT time-out event occurred.

Remarks

This function implements an operation of the WdtoInSleep feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsWdtoInSleep function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_RESET_WdtTimeOutHasOccurredInSleep( RESET_ID_0 ))
{
    //Do Something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_RESET_WdtTimeOutHasOccurredInSleep (RESET_MODULE_ID index)

c) NMI Events Functions

PLIB_RESET_NmiCounterValueGet Function

Gets the NMI counter value.

File

plib_reset.h

C

```
RESET_NMI_COUNT_TYPE PLIB_RESET_NmiCounterValueGet(RESET_MODULE_ID index);
```

Returns

nmi_count - NMI counter value

Description

This function returns the NMI Reset counter value.

Remarks

This function implements an operation of the NmiCounter feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsNmiCounter function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
RESET_NMI_COUNT_TYPE nmi_count;
nmi_count = PLIB_RESET_NmiCounterValueGet( RESET_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

RESET_NMI_COUNT_TYPE PLIB_RESET_NmiCounterValueGet (RESET_MODULE_ID index)

PLIB_RESET_NmiCounterValueSet Function

Sets the NMI counter.

File

plib_reset.h

C

```
void PLIB_RESET_NmiCounterValueSet(RESET_MODULE_ID index, RESET_NMI_COUNT_TYPE nmi_count);
```

Returns

None.

Description

This function sets the NMI counter to be expired for a WDT or DMT reset.

Remarks

NMI counter value range may vary between devices. Please refer to the specific device data sheet. This function implements an operation of the NmiCounter feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsNmiCounter function in your application to determine whether this feature is available.

Preconditions

The system unlock sequence must be performed before calling PLIB_RESET_NmiCounterValueSet

Example

```
//Call system service to unlock the system
PLIB_RESET_NmiCounterValueSet( RESET_ID_0,  0x54);
//Call system service to lock the system
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
nmi_count	NMI counter value

Function

void PLIB_RESET_NmiCounterValueSet (RESET_MODULE_ID index, RESET_NMI_COUNT_TYPE nmi_count)

PLIB_RESET_NmiEventClear Function

Clears the NMI event

File

plib_reset.h

C

```
void PLIB_RESET_NmiEventClear(RESET_MODULE_ID index, RESET_NMI_REASON nmi_reason);
```

Returns

None

Description

This function clears the NMI event. If a WDTO or DMTO NMI event is cleared before the NMI counter reaches zero, no device Reset is asserted.

Remarks

This function implements an operation of the NmiControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsNmiControl function in your application to determine whether this feature is available.

Preconditions

The system unlock sequence must be performed before calling PLIB_RESET_NmiEventTrigger.

Example

```
//Call system service to unlock the system
PLIB_RESET_NmiEventClear( RESET_ID_0, SOFTWARE_NMI );
//Call system service to lock the system
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
nmi_reason	Sets of reasons that can cause a NMI

Function

void PLIB_RESET_NmiEventClear (RESET_MODULE_ID index, RESET_NMI_REASON nmi_reason)

PLIB_RESET_NmiEventTrigger Function

Triggers the NMI event.

File

plib_reset.h

C

```
void PLIB_RESET_NmiEventTrigger(RESET_MODULE_ID index, RESET_NMI_REASON nmi_reason);
```

Returns

None

Description

This function triggers the NMI. In case of a Deadman Timer (DMT) or Watchdog Timer (WDT) NMI event, it will also trigger the NMI counter to start the countdown.

Remarks

This function implements an operation of the NmiControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsNmiControl function in your application to determine whether this feature is available.

Preconditions

The system unlock sequence must be performed before calling PLIB_RESET_NmiEventTrigger.

Example

```
//Call system service to unlock the system
PLIB_RESET_NmiEventTrigger( RESET_ID_0, SOFTWARE_NMI );
//Call system service to lock the system
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
nmi_reason	Sets of reasons which can cause NMI

Function

void PLIB_RESET_NmiEventTrigger (RESET_MODULE_ID index, RESET_NMI_REASON nmi_reason)

PLIB_RESET_NmiReasonGet Function

Returns the reason for the Non-Maskable Interrupt (NMI).

File

plib_reset.h

C

```
RESET_NMI_REASON PLIB_RESET_NmiReasonGet(RESET_MODULE_ID index);
```

Returns

RESET_NMI_REASON - Sets of reasons that can cause a NMI.

Description

This function returns the reason that caused the NMI.

Remarks

This function implements an operation of the NmiControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or include the PLIB_RESET_ExistsNmiControl function in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_RESET_NmiReasonGet( RESET_ID_0 )== WDTO_NMI )
{
    //Do Something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

RESET_NMI_REASON PLIB_RESET_NmiReasonGet (RESET_MODULE_ID index)

d) Feature Existence Functions

PLIB_RESET_ExistsResetReasonStatus Function

Identifies whether the ResetReasonStatus feature exists on the Reset module.

File

plib_reset.h

C

```
bool PLIB_RESET_ExistsResetReasonStatus(RESET_MODULE_ID index);
```

Returns

- true - The ResetReasonStatus feature is supported on the device
- false - The ResetReasonStatus feature is not supported on the device

Description

This function identifies whether the ResetReasonStatus feature is available on the Reset module. When this function returns true, these functions are supported on the device:

- PLIB_RESET_ReasonGet
- PLIB_RESET_ReasonClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RESET_ExistsResetReasonStatus(RESET_MODULE_ID index)

PLIB_RESET_ExistsSoftwareResetTrigger Function

Identifies whether the SoftwareResetTrigger feature exists on the Reset module.

File

plib_reset.h

C

```
bool PLIB_RESET_ExistsSoftwareResetTrigger(RESET_MODULE_ID index);
```

Returns

- true - The SoftwareResetTrigger feature is supported on the device
- false - The SoftwareResetTrigger feature is not supported on the device

Description

This function identifies whether the SoftwareResetTrigger feature is available on the Reset module. When this function returns true, this function is supported on the device:

- PLIB_RESET_SoftwareResetEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RESET_ExistsSoftwareResetTrigger(RESET_MODULE_ID index)

PLIB_RESET_ExistsConfigRegReadError Function

Identifies whether the ConfigRegReadError feature exists on the Reset module.

File

plib_reset.h

C

```
bool PLIB_RESET_ExistsConfigRegReadError(RESET_MODULE_ID index);
```

Returns

- true - The ConfigRegReadError feature is supported on the device
- false - The ConfigRegReadError feature is not supported on the device

Description

This function identifies whether the ConfigRegReadError feature is available on the Reset module. When this function returns true, this function is supported on the device:

- PLIB_RESET_ConfigRegReadErrorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RESET_ExistsConfigRegReadError(RESET_MODULE_ID index)

PLIB_RESET_ExistsNmiControl Function

Identifies whether the NmiControl feature exists on the Reset module.

File

plib_reset.h

C

```
bool PLIB_RESET_ExistsNmiControl(RESET_MODULE_ID index);
```

Returns

- true - The NmiControl feature is supported on the device
- false - The NmiControl feature is not supported on the device

Description

This function identifies whether the NmiControl feature is available on the Reset module. When this function returns true, these functions are supported on the device:

- PLIB_RESET_NmiReasonGet

- PLIB_RESET_NmiEventTrigger
- PLIB_RESET_NmiEventClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RESET_ExistsNmiControl(RESET_MODULE_ID index)

PLIB_RESET_ExistsNmiCounter Function

Identifies whether the NmiCounter feature exists on the Reset module.

File

plib_reset.h

C

```
bool PLIB_RESET_ExistsNmiCounter(RESET_MODULE_ID index);
```

Returns

- true - The NmiCounter feature is supported on the device
- false - The NmiCounter feature is not supported on the device

Description

This function identifies whether the NmiCounter feature is available on the Reset module. When this function returns true, these functions are supported on the device:

- PLIB_RESET_NmiCounterValueSet
- PLIB_RESET_NmiCounterValueGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RESET_ExistsNmiCounter(RESET_MODULE_ID index)

PLIB_RESET_ExistsWdtoInSleep Function

Identifies whether the WdtoInSleep feature exists on the Reset module.

File

plib_reset.h

C

```
bool PLIB_RESET_ExistsWdtoInSleep(RESET_MODULE_ID index);
```

Returns

- true - The WdtoInSleep feature is supported on the device

- false - The WdtoInSleep feature is not supported on the device

Description

This function identifies whether the WdtoInSleep feature is available on the Reset module. When this function returns true, this function is supported on the device:

- PLIB_RESET_WdtTimeOutHasOccurredInSleep

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RESET_ExistsWdtoInSleep(RESET_MODULE_ID index)

e) Data Types and Constants

RESET_MODULE_ID Enumeration

Identifies the supported Reset modules.

File

help_plib_reset.h

C

```
typedef enum {
  RESET_ID_0
} RESET_MODULE_ID;
```

Members

Members	Description
RESET_ID_0	RESET Module ID 0

Description

Reset Module ID

This enumeration identifies the Reset modules that are available on the microcontroller. This is the superset of all the possible instances that might be available on a Microchip microcontroller.

Remarks

Caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules are available on all devices. Refer to the "Resets" chapter in the specific device data sheet to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

RESET_REASON Enumeration

Lists the reset sources.

File

help_plib_reset.h

C

```
typedef enum {
  RESET_REASON_NONE = 0x00000000,
  RESET_REASON_POWERON = 0x00000003,
  RESET_REASON_BROWNOUT = 0x00000002,
  RESET_REASON_WDT_TIMEOUT = 0x00000010,
```

```
  RESET_REASON_SOFTWARE = 0x00000040,
  RESET_REASON_MCLR = 0x00000080,
  RESET_REASON_CONFIG_MISMATCH = 0x00000200,
  RESET_REASON_VBAT = 0x00010000,
  RESET_REASON_VBPOR = 0x00020000,
  RESET_REASON_HIGH_VOLTAGE_DETECT = 0x20000000,
  RESET_REASON_DMT_TIMEOUT = 0x00000020,
  RESET_REASON_ALL = 0x200002D3
} RESET_REASON;
```

Members

Members	Description
RESET_REASON_NONE = 0x00000000	The last reset was of unknown type
RESET_REASON_POWERON = 0x00000003	The last reset was a power on reset
RESET_REASON_BROWNOUT = 0x00000002	The last reset was a brown out reset
RESET_REASON_WDT_TIMEOUT = 0x00000010	The last reset was a watch dog timer time-out reset
RESET_REASON_SOFTWARE = 0x00000040	The last reset was a software reset
RESET_REASON_MCLR = 0x00000080	The last reset was a master clear(MCLR) reset
RESET_REASON_CONFIG_MISMATCH = 0x00000200	The last reset was a configuration mismatch reset
RESET_REASON_VBAT = 0x00010000	The last reset was a VBAT reset
RESET_REASON_VBPOR = 0x00020000	The last reset was a VBPOR reset, because of no supply or Brown-out on VBAT pin
RESET_REASON_HIGH_VOLTAGE_DETECT = 0x20000000	The last reset was high voltage detect reset
RESET_REASON_DMT_TIMEOUT = 0x00000020	The last reset was a Deadman Timer time-out reset
RESET_REASON_ALL = 0x200002D3	All reset flags are high

Description

RESET source select enumeration

This enumeration lists the possible reset sources.

Remarks

Not all reset sources exist on all devices. Please refer to the specific device data sheet for availability.

RESET_CONFIG_REG_READ_ERROR Enumeration

Lists the Configuration register read errors.

File

help_plib_reset.h

C

```
typedef enum {
  PRIMARY_CONFIG_REG_READ_ERROR,
  PRIMARY_AND_ALTERNATIVE_CONFIG_REG_READ_ERROR,
  NO_CONFIG_REG_READ_ERROR
} RESET_CONFIG_REG_READ_ERROR;
```

Members

Members	Description
PRIMARY_CONFIG_REG_READ_ERROR	An error occurred during the read of primary configuration registers
PRIMARY_AND_ALTERNATIVE_CONFIG_REG_READ_ERROR	An error occurred during the read of primary and alternate configuration register
NO_CONFIG_REG_READ_ERROR	No error occurred during the read of configuration registers

Description

RESET Config Register Read Error Enumeration

This enumeration lists the possible errors while reading Configuration registers.

Remarks

Not all errors exist on all devices. Please refer to the specific device data sheet for availability.

RESET_NMI_COUNT_TYPE Type

Defines NMI counter data type

File

plib_reset.h

C

```
typedef unsigned int RESET_NMI_COUNT_TYPE;
```

Description

RESET_NMI_COUNT_TYPE data type

NMI counter data type definition.

RESET_NMI_REASON Enumeration

Lists the NMI reasons.

File

help_plib_reset.h

C

```
typedef enum {
  WDTO_NMI,
  DMTO_NMI,
  SOFTWARE_NMI,
  GLOBAL_NMI,
  LVD_NMI,
  CF_NMI,
  WDTS_NMI
} RESET_NMI_REASON;
```

Members

Members	Description
WDTO_NMI	Watch Dog Timer time-out has caused NMI
DMTO_NMI	Deadman Timer time-out has caused NMI
SOFTWARE_NMI	Software triggered NMI will be generated
GLOBAL_NMI	General NMI or user-initiated NMI has occurred
LVD_NMI	Low Voltage Detection Condition has caused NMI
CF_NMI	Clock Failure has caused NMI
WDTS_NMI	Watch Dog Timer in Sleep has caused NMI

Description

NMI Reason enumeration

This enumeration lists the possible NMI sources.

Remarks

Not all NMI sources exist on all devices. Please refer to the specific device data sheet for availability.

Files

Files

Name	Description
plib_reset.h	Defines the Reset Peripheral Library interface.
help_plib_reset.h	This is file help_plib_reset.h.

Description

This section lists the source and header files used by the library.

plib_reset.h

Defines the Reset Peripheral Library interface.

Functions

	Name	Description
	PLIB_RESET_ConfigRegReadErrorGet	Gets the Configuration register read error.
	PLIB_RESET_ExistsConfigRegReadError	Identifies whether the ConfigRegReadError feature exists on the Reset module.
	PLIB_RESET_ExistsNmiControl	Identifies whether the NmiControl feature exists on the Reset module.
	PLIB_RESET_ExistsNmiCounter	Identifies whether the NmiCounter feature exists on the Reset module.
	PLIB_RESET_ExistsResetReasonStatus	Identifies whether the ResetReasonStatus feature exists on the Reset module.
	PLIB_RESET_ExistsSoftwareResetTrigger	Identifies whether the SoftwareResetTrigger feature exists on the Reset module.
	PLIB_RESET_ExistsWdtoInSleep	Identifies whether the WdtoInSleep feature exists on the Reset module.
	PLIB_RESET_NmiCounterValueGet	Gets the NMI counter value.
	PLIB_RESET_NmiCounterValueSet	Sets the NMI counter.
	PLIB_RESET_NmiEventClear	Clears the NMI event
	PLIB_RESET_NmiEventTrigger	Triggers the NMI event.
	PLIB_RESET_NmiReasonGet	Returns the reason for the Non-Maskable Interrupt (NMI).
	PLIB_RESET_ReasonClear	Clears the RCON register.
	PLIB_RESET_ReasonGet	Returns the reason for a reset.
	PLIB_RESET_SoftwareResetEnable	Enables the software reset.
	PLIB_RESET_WdtTimeOutHasOccurredInSleep	Returns the state of the device when WDT time-out occurred

Types

	Name	Description
	RESET_NMI_COUNT_TYPE	Defines NMI counter data type

Description

Reset Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Reset Peripheral Library for Microchip microcontrollers. The definitions in this file are for the Reset Controller module.

File Name

plib_reset.h

Company

Microchip Technology Inc.

help_plib_reset.h

Enumerations

	Name	Description
	RESET_CONFIG_REG_READ_ERROR	Lists the Configuration register read errors.
	RESET_MODULE_ID	Identifies the supported Reset modules.
	RESET_NMI_REASON	Lists the NMI reasons.
	RESET_REASON	Lists the reset sources.

Description

This is file help_plib_reset.h.

RTCC Peripheral Library

This section describes the Real-Time Clock and Calendar (RTCC) Peripheral Library.

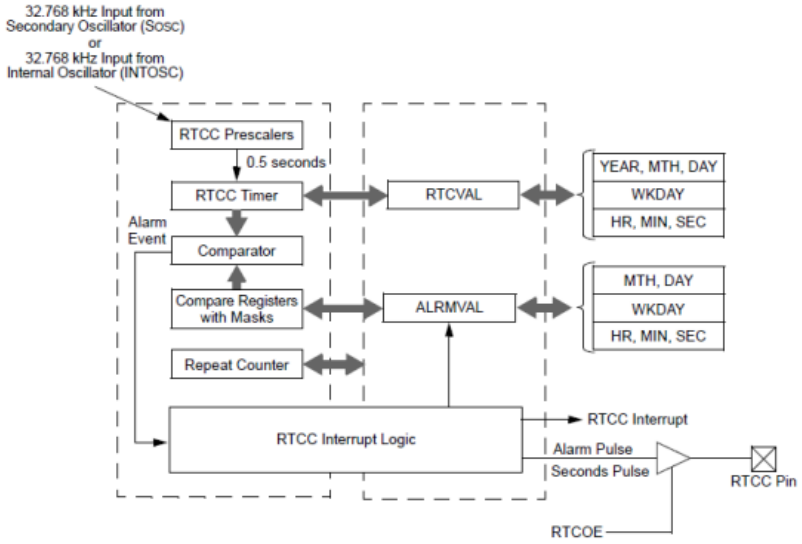
Introduction

This module provides real-time clock and calendar functions. The RTCC is intended for applications where accurate time must be maintained for extended periods with minimum to no intervention from the CPU. The module is optimized for low-power usage to provide extended battery life while keeping track of time.

Description

The RTCC module is a 100-year clock and calendar with automatic leap year detection. The range of the clock is from 00:00:00 (midnight) on January 1, 2000 to 23:59:59 on December 31, 2099. The hours are available in 24-hour (military time) format. The clock provides a granularity of one second with half-second visibility to the user.

RTCC Block Diagram



Operating Modes

Note: Please refer to the **"RTCC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by your device.

RTCC Mode

The RTCC module is intended to be clocked by an external real-time clock crystal oscillating at 32.768 kHz. The prescaler divides the crystal oscillator frequency to produce a 1 Hz frequency for the clock and calendar. The current Date and Time are stored in a BCD counter.

Alarm Mode

The RTCC module provides an alarm function that is configurable from a half-second to one year. After the alarm is enabled, the RTCC module can be configured to repeat the alarm at preconfigured intervals. The indefinite repetition of the alarm is provided through the Chime feature.

Using the Library

This topic describes the basic architecture of the RTCC Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_rtcc.h`

The interface to the RTCC Peripheral Library is defined in the `plib_rtcc.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the RTCC Peripheral Library should include `peripheral.h`.

Library File: The RTCC Peripheral Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for information on how the peripheral interacts with the framework.

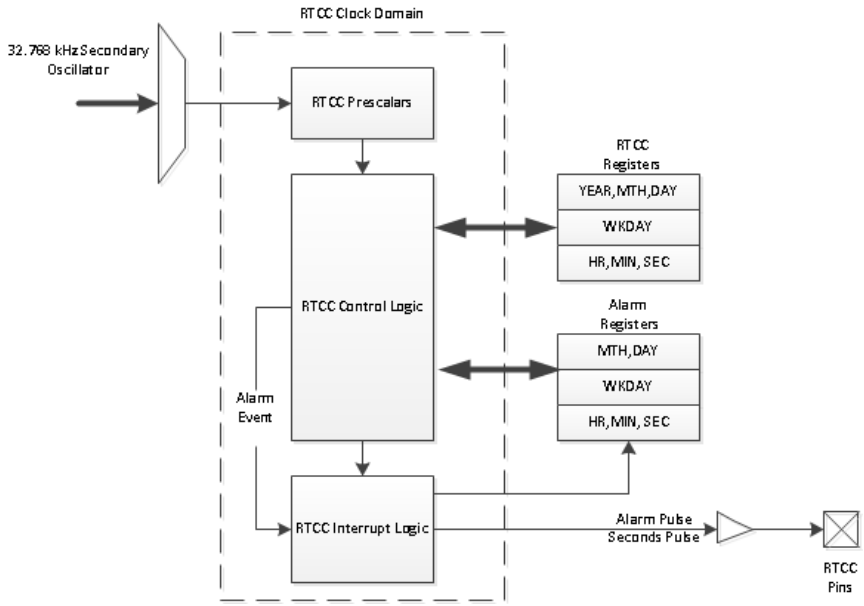
Hardware Abstraction Model

This library provides a low-level abstraction of the RTCC module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The RTCC peripheral library provides interface routines to interact with the blocks shown in the following diagram.

Hardware Abstraction Model



The **RTCC Control Logic** provides the capability to operate the RTCC module in the RTCC and Alarm modes. The control logic also handles the comparison and the counter increments, which in turn control the alarm generation. This module is also responsible for the generation of a square wave at the RTCC output pin. In addition, the RTCC module provides the RTCC drift calibration.

The **RTCC Interrupt Logic** is primarily used in alarm generation. The various configurations of the alarm and their repetition period may be defined.

The **RTCC Registers** store the actual date and time. Separate registers are present for RTCC and alarm functionality.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the RTCC module.

Library Interface Section	Description
General Functions	Provides the setup and configuration of general functionalities in the RTCC module. This section also covers the drift calibration.
RTCC Mode Functions	Provides configuration of the RTCC-related registers. Updating the Date and Time registers along with reading them accurately is performed.
Alarm Mode Functions	Provides configuration of the Alarm-related registers. Updating the Date and Time registers along with reading them accurately is performed. In addition, the rate at which an alarm occurs and the frequency is also configured in this mode.
Other Functions	Provides additional RTCC functions.
Feature Existence Functions	Provides functions that determine whether certain features exist on a device.

How the Library Works

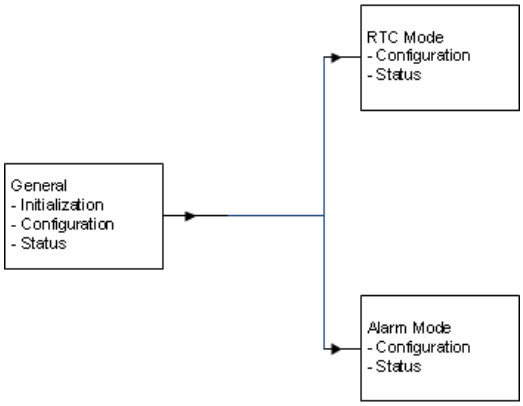
Provides information on how the library works.

Description

The usage model for this library is explained in the following sections.

The following diagram describes the major components of the usage model.

Usage Model



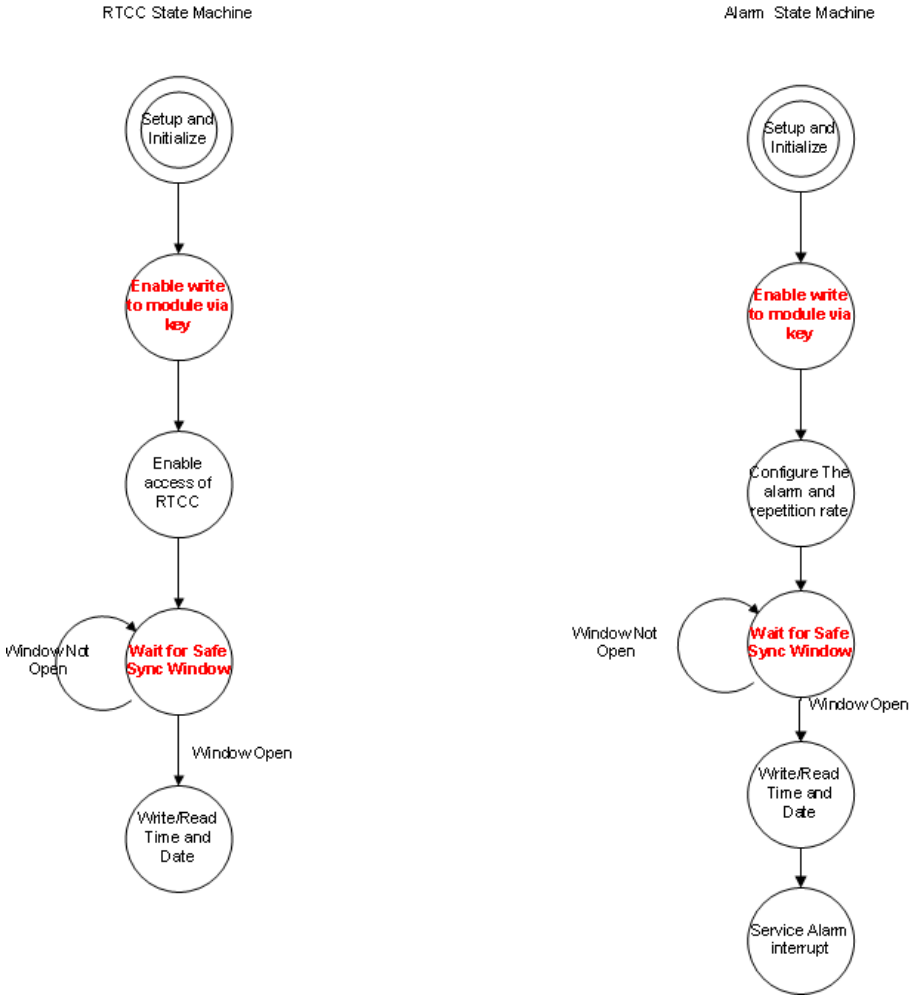
State Machine

This RTCC state machine is provided to give a general idea of the usage model of the peripheral library. Refer to the usage model for more detailed steps for the scenario that is being used.

Description

The following state machine diagram is for RTCC mode and Alarm Mode during normal operation.

RTCC State Machine



State	Associated Function
Setup and Initialization	Refer to mode of RTCC for detailed instructions of setup.

Enable Write via Key Sequence	For RTCC to perform any function, the module has to be enabled via a series of commands in a special sequence. This is abstracted and is achieved using PLIB_RTCC_WriteEnable.
Enable Access of RTC	These include the functions needed to set up the RTCC/Alarm, which will be explained in detail in the following sections.
Wait for Safe Sync Window	To update the RTCC on-the-fly, a safe window period is available, corresponding to 32 clock cycles called RTSYNC. Further sections will explain the functionality in greater detail.
Write/Read Date and Time	The application may read or write the date and time using PLIB_RTCC_RTCDateGet and PLIB_RTCC_RTCTimeGet or PLIB_RTCC_RTCDateSet and PLIB_RTCC_RTCTimeSet, among others. These functions are device-specific and may not be available for all devices.

RTCC Mode Operations

This section describes the RTCC mode of operation of the RTCC module.

Note: Please refer to the **"RTCC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by your device.

Description

Enabling the RTCC Module for Write Operations

To perform a write to any of the RTCC timer registers the RTCWREN bit must be set. It is recommended that this bit be reset unless a write operation is specifically required.

A sequence of commands, each varying across processor families must be executed to set the RTCWREN.

Example: Enabling a Write

```
// Assume Interrupts are disabled.
// Assume the DMA controller is suspended

PLIB_RTCC_WriteEnable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

// This API performs the writing of the sequence
// 0xaa996655 and 0x556699aa to the SYSKET in MCU32
// and 0x55 and 0xaa in case of MCU 16 and MCU 8
// This API then sets the RTCWREN.
```

Updating RTCC Time and Date

1. Turn off the RTCC using the API PLIB_RTCC_Disable.
2. Wait for the sync clock to turn off by reading the status of PLIB_RTCC_RTCSyncStatusGet (this operation may not be required on all devices).
3. Update the Date and Time registers using PLIB_RTCC_RTCDateSet and PLIB_RTCC_RTCTimeSet.
4. Turn on the RTCC module using PLIB_RTCC_Enable.

Example: Updating RTCC Time and Date

```
// Assume Write Enable has been performed correctly
unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006

PLIB_RTCC_Disable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

while(PLIB_RTCC_RTCSyncStatusGet(MY_RTCC_INSTANCE)); // Wait for clock to turn off

PLIB_RTCC_RTCTimeSet(MY_RTCC_INSTANCE, time);  // Update the Time

PLIB_RTCC_RTCDateSet(MY_RTCC_INSTANCE, date);  // Update the Date

PLIB_RTCC_Enable(MY_RTCC_INSTANCE);
```

Updating RTCC Time and Date using RTSYNC Window

1. Wait for the sync clock to turn off by reading the status of PLIB_RTCC_RTCSyncStatusGet.
2. Update the RTC Date and Time registers using PLIB_RTCC_RTCDateSet and PLIB_RTCC_RTCTimeSet.

Example: Updating RTCC Time and Date Using RTSYNC Window

```
// Assume Write Enable has been performed correctly
unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

//Disable Critical Interrupts here ....

while(PLIB_RTCC_RTCSyncStatusGet(MY_RTCC_INSTANCE)); // Wait for clock to turn off

PLIB_RTCC_RTCTimeSet(MY_RTCC_INSTANCE, time);  // Update the Time

PLIB_RTCC_RTCDateSet(MY_RTCC_INSTANCE, date);  // Update the Date

//Enable Critical Interrupts here ....
```

Updating RTCC Time and Date Using Register Pointers

1. Turn off the RTCC using PLIB_RTCC_Disable.
2. Identify whether the RTCTimeValue and RTCDateValue features exist on the RTCC module using PLIB_RTCC_ExistsRTCTime and PLIB_RTCC_ExistsRTCDate.
3. Update the RTC Date and Time registers using PLIB_RTCC_RTCDateSet
4. and PLIB_RTCC_RTCtimeSet.
5. Turn on the RTCC module using PLIB_RTCC_Enable.

Example: Updating RTCC Time and Date Using Register Pointers

```
// Assume Write Enable has been performed correctly

PLIB_RTCC_Disable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

PLIB_RTCC_RTCYearSet(MY_RTCC_INSTANCE, 0x0007);   // Update the Year

PLIB_RTCC_RTCMonthSet(MY_RTCC_INSTANCE,0x10);    // Set Month to October

PLIB_RTCC_RTCDaySet(MY_RTCC_INSTANCE,0x28);      // Set Day to the 27th

PLIB_RTCC_RTCWeekDaySet(MY_RTCC_INSTANCE,0x01); // Set the Day of the Week to Sunday

PLIB_RTCC_RTCHourSet(MY_RTCC_INSTANCE,0x10);     // Set the Hour value to 10

PLIB_RTCC_RTCMinuteSet(MY_RTCC_INSTANCE,0x00);   // Set Minute value to 0

PLIB_RTCC_RTCSecondSet(MY_RTCC_INSTANCE,0x00);   // Set Seconds value to 0

PLIB_RTCC_Enable(MY_RTCC_INSTANCE);
```

Alarm Mode Operations

This section describes the Alarm mode of operation of the RTCC module.

Note: Please refer to the **"RTCC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by your device.

Description

Configuring the One-Time-Per-Day Alarm

1. Turn off the Alarm and also the chime, alarm repeats, and masks using PLIB_RTCC_AlarmDisable.
2. Wait for the sync clock to turn off by reading the status of PLIB_RTCC_AlarmSyncStatusGet (This operation may not be required on all devices).
3. Update the Alarm Date and Time registers using PLIB_RTCC_AlarmTimeSet and PLIB_RTCC_AlarmDateSet.
4. Set the alarm mask to every half-second and the repeat counter to zero using PLIB_RTCC_AlarmMaskModeSelect and PLIB_RTCC_AlarmRepeatCountSet, respectively.
5. Turn on the RTCC Alarm module using PLIB_RTCC_AlarmEnable.

Example: Configuring the One-Time-Per-Day Alarm

```
// Assume Write Enable has been performed correctly
unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006

PLIB_RTCC_AlarmDisable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

while(PLIB_RTCC_AlarmSyncStatusGet(MY_RTCC_INSTANCE)); // Wait for clock to turn off

PLIB_RTCC_AlarmTimeSet(MY_RTCC_INSTANCE, time);  // Update the Time

PLIB_RTCC_AlarmDateSet(MY_RTCC_INSTANCE, date);  // Update the Date

PLIB_RTCC_AlarmMaskModeSelect(MY_RTCC_INSTANCE, RTCC_ALARM_EVERY_HALF_SECOND);

PLIB_RTCC_AlarmRepeatCountSet(MY_RTCC_INSTANCE, 0 /*Number of repetitions*/ );

PLIB_RTCC_AlarmEnable(MY_RTCC_INSTANCE);
```

Configuring the Repeat Alarm

1. Turn off the Alarm and also the Chime, alarm repeats, and masks using PLIB_RTCC_AlarmDisable.
2. Wait for the sync clock to turn off by reading the status of PLIB_RTCC_AlarmSyncStatusGet (this operation may not be required on all devices).
3. Update the Alarm Date and Time registers using PLIB_RTCC_AlarmTimeSet and PLIB_RTCC_AlarmDateSet.
4. Set the alarm mask to every half-second and the repeat counter to 10 using PLIB_RTCC_AlarmMaskModeSelect and PLIB_RTCC_AlarmRepeatCountSet, respectively.
5. Turn on the RTCC Alarm module using PLIB_RTCC_AlarmEnable.

Example: Configuring the Repeat Alarm

```
// Assume Write Enable has been performed correctly
unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006

PLIB_RTCC_AlarmDisable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

while(PLIB_RTCC_AlarmSyncStatusGet(MY_RTCC_INSTANCE)); // Wait for clock to turn off

PLIB_RTCC_AlarmTimeSet(MY_RTCC_INSTANCE, time);  // Update the Time

PLIB_RTCC_AlarmDateSet(MY_RTCC_INSTANCE, date);  // Update the Date

PLIB_RTCC_AlarmMaskModeSelect(MY_RTCC_INSTANCE, RTCC_ALARM_EVERY_HALF_SECOND);

PLIB_RTCC_AlarmRepeatCountSet(MY_RTCC_INSTANCE, 10 /*No of repetitions*/ );

PLIB_RTCC_AlarmEnable(MY_RTCC_INSTANCE);
```

Configuring the Indefinite One-Per-Day Alarm

1. Turn off the Alarm and also the Chime, alarm repeats and masks using PLIB_RTCC_AlarmDisable.
2. Wait for the sync clock to turn off by reading the status of PLIB_RTCC_AlarmSyncStatusGet (This operation may not be required on all devices).
3. Update the Alarm Date and Time registers using PLIB_RTCC_AlarmTimeSet and PLIB_RTCC_AlarmDateSet.
4. Set the alarm mask to every half-second and the repeat counter to '0' using PLIB_RTCC_AlarmMaskModeSelect and PLIB_RTCC_AlarmRepeatCountSet, respectively.
5. Turn on the RTCC Alarm module using PLIB_RTCC_AlarmEnable and enable chime using PLIB_RTCC_AlarmChimeEnable.

Example: Configuring the Indefinite One-Per-Day Alarm

```
// Assume Write Enable has been performed correctly
unsigned long time = 0x04153300; // Set time 04 hrs 15 mins and 33 sec
unsigned long date = 0x06102705; // Set date to Friday 27 Oct 2006

PLIB_RTCC_AlarmDisable(MY_RTCC_INSTANCE);

// where, MY_RTCC_INSTANCE - is a specific instance of the hardware peripheral.

while(PLIB_RTCC_AlarmSyncStatusGet(MY_RTCC_INSTANCE)); // Wait for clock to turn off
```

```
PLIB_RTCC_AlarmTimeSet(MY_RTCC_INSTANCE, time);  // Update the Time

PLIB_RTCC_AlarmDateSet(MY_RTCC_INSTANCE, date);  // Update the Date

PLIB_RTCC_AlarmMaskModeSelect(MY_RTCC_INSTANCE, RTCC_ALARM_EVERY_HALF_SECOND);

PLIB_RTCC_AlarmRepeatCountSet(MY_RTCC_INSTANCE, 0 /*No of repetitions*/ );

PLIB_RTCC_AlarmEnable(MY_RTCC_INSTANCE);

PLIB_RTCC_AlarmChimeEnable(MY_RTCC_INSTANCE);
```

Other Functionality

This section describes the Drift Calibration mode of operation of the RTCC module.

Note: Please refer to the **"RTCC"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine which features are supported by your device.

Description

The real-time crystal input can be calibrated using the periodic auto-adjust feature. When properly calibrated, the RTCC module can provide an error of less than 0.66 seconds per month. Calibration has the ability to eliminate an error of up to 260 ppm.

The calibration is accomplished by finding the number of error clock pulses and writing this value into the calibration bit fields using PLIB_RTCC_DriftCalibrateSet.

Drift Calibration

1. Using another timer resource, the user must find the error of the 32.768 kHz crystal.
2. Once the error is known, it must be converted to the number of error clock pulses per minute, as shown by ***(Ideal Frequency(32,758) - Measured Frequency) * 60 = Error Clocks Per Minute***.
3. If the error is negative, the input to PLIB_RTCC_DriftCalibrateSet is negative. Likewise, if the error is positive, the input is positive. The negative and positive values are the actual number of clock pulses to be added or subtracted from the timer counter per minute.

Example: Configuring the One-Time-Per-Day Alarm

```
// Assume Write Enable has been performed correctly
// Assume steps 1 and 2 are performed and the error is determined.

int driftValue = 0x3FD   // 10 bits Adjustment, -3 in value
int T0,T1;

do
{
    T0 = PLIB_RTCC_RTCTimeGet(MY_RTCC_INSTANCE);
    T1 = PLIB_RTCC_RTCTimeGet(MY_RTCC_INSTANCE);
    // where, MY_RTCC_INSTANCE is a specific instance of the hardware peripheral
}while(T0!=T1)     // Read Valid Time Value

if((T0 & 0XFF) == 0) // we are at seconds exactly 00, wait for auto adjust
{
    while PLIB_RTCC_HalfSecondStatusGet(MY_RTCC_INSTANCE); // Wait for the second Half
}

PLIB_RTCC_DriftCalibrateSet(MY_RTCC_INSTANCE, 0x00); //Clear the calibration

PLIB_RTCC_DriftCalibrateSet(MY_RTCC_INSTANCE, 0xdriftValue);
```

Configuring the Library

The library is configured for the supported RTCC modules when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Functions

	Name	Description
	PLIB_RTCC_ClockOutputDisable	Disables the specific RTCC module's output pin.
	PLIB_RTCC_ClockOutputEnable	Enables the specific RTCC module's output pin.
	PLIB_RTCC_ClockRunningStatus	Provides the status of the RTCC clock.

b) RTCC Mode Functions

	Name	Description
	PLIB_RTCC_Disable	Disables the specific RTCC module on the device.
	PLIB_RTCC_Enable	Enables the specific RTCC module on the device.
	PLIB_RTCC_WriteDisable	Disables writing to the specific RTCC module's value registers.
	PLIB_RTCC_WriteEnable	Enables writing to the specific RTCC module's value registers.
	PLIB_RTCC_RTCDateGet	Returns the contents of the specific RTCC module's Date register.
	PLIB_RTCC_RTCDateSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCDayGet	Returns the contents of the Days bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCDaySet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCHourGet	Returns the contents of the Hours bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCHourSet	Writes the contents of the Hours bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMinuteGet	Returns the contents of the Minutes bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMinuteSet	Writes the contents of Minutes bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMonthGet	Returns the contents of the Months bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCMonthSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCSecondGet	The function returns the contents of the Seconds bits in the specific RTCC device's Time register.
	PLIB_RTCC_RTCSecondSet	Writes the contents of Seconds bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCTimeGet	Returns the contents of the specific RTCC module's Time register.
	PLIB_RTCC_RTCTimeSet	Writes to the specific RTCC module's Time register.
	PLIB_RTCC_RTCWeekDayGet	Returns the contents of the WeekDay bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCWeekDaySet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCYearGet	Returns the contents of the Year bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCYearSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCSyncStatusGet	The function returns the synchronization status bit.

c) Alarm Mode Functions

	Name	Description
	PLIB_RTCC_AlarmChimeDisable	Disables the specific RTCC module's chime.
	PLIB_RTCC_AlarmChimeEnable	Enables the specific RTCC module's chime.
	PLIB_RTCC_AlarmDateGet	Returns the contents of the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmDateSet	Writes to the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmDayGet	Returns the contents of the Day bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmDaySet	Writes to the specific RTCC module's Alarm Date value register.
	PLIB_RTCC_AlarmDisable	Disables the specific RTCC module's alarm.
	PLIB_RTCC_AlarmEnable	Enables the specific RTCC module's alarm.
	PLIB_RTCC_AlarmHourGet	Returns the contents of Hours bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmHourSet	The function returns the contents of Hours bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMaskModeSelect	Sets the specific RTCC module's alarm mask Configuration bits.
	PLIB_RTCC_AlarmMinuteGet	Returns the contents of Minutes bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMinuteSet	Returns the contents of the Minutes bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMonthGet	Returns the contents of the Month bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmMonthSet	Writes to the specific RTCC module's Alarm Date register.

	PLIB_RTCC_AlarmPulseInitialGet	Returns the state of the initial alarm pulse.
	PLIB_RTCC_AlarmPulseInitialSet	Enables the determination of the initial alarm pulse.
	PLIB_RTCC_AlarmRepeatCountGet	Reads the specific RTCC module's alarm repeat counter.
	PLIB_RTCC_AlarmRepeatCountSet	Sets the specific RTCC module's alarm repeat counter.
	PLIB_RTCC_AlarmSecondGet	Returns the contents of the Seconds bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmSecondSet	Returns the contents of Seconds bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmTimeGet	Returns the contents of the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmTimeSet	Writes to the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmValueRegisterPointer	Sets the specific RTCC module's Alarm register pointer.
	PLIB_RTCC_AlarmWeekDayGet	Returns the contents of the WeekDay bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmWeekDaySet	Writes to the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmSyncStatusGet	The function returns the synchronization status bit.

d) Other Functions

	Name	Description
	PLIB_RTCC_ClockSourceSelect	Selects the clock source for the RTCC module.
	PLIB_RTCC_DriftCalibrateGet	Reads the specific RTCC module's drift calibration bits.
	PLIB_RTCC_DriftCalibrateSet	Sets the specific RTCC module's drift calibration bits.
	PLIB_RTCC_HalfSecondStatusGet	The function returns the half second status bit.
	PLIB_RTCC_OutputSelect	Selects which signal will be presented on the RTCC pin
	PLIB_RTCC_StopInIdleDisable	Continues normal RTCC operation when the device enters Idle mode.
	PLIB_RTCC_StopInIdleEnable	Disables access to the RTCC module by the Peripheral Bus Clock (PBCLK) when the CPU enters Idle mode.

e) Feature Existence Functions

	Name	Description
	PLIB_RTCC_ExistsAlarmChimeControl	Identifies whether the AlarmChimeControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmControl	Identifies whether the AlarmControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmDate	Identifies whether the AlarmDate feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmMaskControl	Identifies whether the AlarmMaskControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmPulseInitial	Identifies whether the AlarmPulseInitial feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmRepeatControl	Identifies whether the AlarmRepeatControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmTime	Identifies whether the AlarmTime feature exists on the RTCC module.
	PLIB_RTCC_ExistsCalibration	Identifies whether the Calibration feature exists on the RTCC module.
	PLIB_RTCC_ExistsClockRunning	Identifies whether the ClockRunning feature exists on the RTCC module.
	PLIB_RTCC_ExistsClockSelect	Identifies whether the ClockSelect feature exists on the RTCC module.
	PLIB_RTCC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsHalfSecond	Identifies whether the HalfSecond feature exists on the RTCC module.
	PLIB_RTCC_ExistsOutputControl	Identifies whether the OutputControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsOutputSelect	Identifies whether the OutputSelect feature exists on the RTCC module.
	PLIB_RTCC_ExistsRTCDate	Identifies whether the RTCDateValue feature exists on the RTCC module.
	PLIB_RTCC_ExistsRTCTime	Identifies whether the RTCTimeValue feature exists on the RTCC module.
	PLIB_RTCC_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the RTCC module.
	PLIB_RTCC_ExistsSynchronization	Identifies whether the Synchronization feature exists on the RTCC module.
	PLIB_RTCC_ExistsWriteEnable	Identifies whether the WriteEnable feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmSynchronization	Identifies whether the AlarmSynchronization feature exists on the RTCC module.

f) Data Types and Constants

	Name	Description
	RTCC_ALARM_MASK_CONFIGURATION	Data type defining the different configurations for the alarm mask bits.

	RTCC_MODULE_ID	Enumeration: RTCC_MODULE_ID This enumeration defines number of modules which are available on the microcontroller. This is the superset of all of the possible instances that may be available on the Microchip microcontrollers. Refer to the data sheet to get the correct number of modules defined for desired microcontroller.
	RTCC_VALUE_REGISTER_POINTER	Data type defining the different configurations by which the RTCC Date and Time Registers can be accessed.

Description

This section describes the Application Programming Interface (API) functions of the RTCC Peripheral Library.

Refer to each section for a detailed description.

a) General Functions

PLIB_RTCC_ClockOutputDisable Function

Disables the specific RTCC module's output pin.

File

plib_rtcc.h

C

```
void PLIB_RTCC_ClockOutputDisable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific RTCC module's output pin.

Remarks

This function implements an operation of the OutputControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsOutputControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_ClockOutputDisable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_ClockOutputDisable (RTCC_MODULE_ID index)

PLIB_RTCC_ClockOutputEnable Function

Enables the specific RTCC module's output pin.

File

plib_rtcc.h

C

```
void PLIB_RTCC_ClockOutputEnable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific RTCC module's output and generates a square wave using either the alarm or the 1 Hz clock output on the RTCC pin.

Remarks

This function implements an operation of the OutputControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsOutputControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_ClockOutputEnable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_ClockOutputEnable (RTCC_MODULE_ID index)

PLIB_RTCC_ClockRunningStatus Function

Provides the status of the RTCC clock.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ClockRunningStatus(RTCC_MODULE_ID index);
```

Returns

The status of the RTCC clock.

Description

This function provides the status of the RTCC clock.

Remarks

This function implements an operation of the ClockRunning feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsClockRunning in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
bool status;
status = PLIB_RTCC_ClockRunningStatus(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_RTCC_ClockRunningStatus (RTCC_MODULE_ID index)

b) RTCC Mode Functions

PLIB_RTCC_Disable Function

Disables the specific RTCC module on the device.

File

plib_rtcc.h

C

```
void PLIB_RTCC_Disable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific RTCC module on the device.

Remarks

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsEnableControl in your application to automatically determine whether this feature is available.

Preconditions

The RTCC module should be unlocked for writing using the function PLIB_RTCC_WriteEnable before this function is called.

Example

```
PLIB_RTCC_Disable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_Disable (RTCC_MODULE_ID index)

PLIB_RTCC_Enable Function

Enables the specific RTCC module on the device.

File

plib_rtcc.h

C

```
void PLIB_RTCC_Enable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific RTCC module on the device.

Remarks

By calling this function, the RTCC pins are controlled by the RTCC module. The RTCC module will continue to function when the device is held in reset.

This function implements an operation of the EnableControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsEnableControl in your application to automatically determine whether this feature is available.

Preconditions

The RTCC module should be unlocked for writing using the function PLIB_RTCC_WriteEnable before this function is called.

Example

```
PLIB_RTCC_Enable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_Enable (RTCC_MODULE_ID index)

PLIB_RTCC_WriteDisable Function

Disables writing to the specific RTCC module's value registers.

File

plib_rtcc.h

C

```
void PLIB_RTCC_WriteDisable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function disables writing to the specific RTCC module's value registers.

Remarks

This function implements an operation of the WriteEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWriteEnable in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_WriteDisable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_WriteDisable (RTCC_MODULE_ID index)

PLIB_RTCC_WriteEnable Function

Enables writing to the specific RTCC module's value registers.

File

plib_rtcc.h

C

```
void PLIB_RTCC_WriteEnable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function enables writing to the specific RTCC module's value registers.

Remarks

This function implements an operation of the WriteEnable feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsWriteEnable in your application to automatically determine whether this feature is available.

Preconditions

The SYSLOCK unlock sequence must be executed prior to calling this function by calling the PLIB_CORE_SysUnlock function.

Example

```
PLIB_RTCC_WriteEnable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_WriteEnable (RTCC_MODULE_ID index)

PLIB_RTCC_RTCDateGet Function

Returns the contents of the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCDateGet(RTCC_MODULE_ID index);
```

Returns

Date register contents.

Description

The function returns the contents of the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Date;
Date = PLIB_RTCC_RTCDateGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCDateGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCDateSet Function

Writes to the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCDateSet(RTCC_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t data = 0x06102705;     //Date = 27 Oct 2006 Friday
PLIB_RTCC_RTCDateSet(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_RTCDateSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_RTCDayGet Function

Returns the contents of the Days bits in the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCDayGet(RTCC_MODULE_ID index);
```

Returns

Days bits in the Date register.

Description

The function returns the contents of the Days bits in the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Day;
Day = PLIB_RTCC_RTCDayGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCDayGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCDaySet Function

Writes to the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCDaySet(RTCC_MODULE_ID index, uint32_t day);
```

Returns

None.

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t Day = 0x27;        //Day = 27th of the month
PLIB_RTCC_RTCDaySet(RTCC_ID_0, Day);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
day	The BCD value of the day to set in the Date register

Function

void PLIB_RTCC_RTCDaySet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_RTCHourGet Function

Returns the contents of the Hours bits in the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCHourGet(RTCC_MODULE_ID index);
```

Returns

BCD value of the Hours bits in the Time register.

Description

The function returns the contents of the Hours bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Example

```
uint32_t Hour;
Hour = PLIB_RTCC_RTCHourGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCHourGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCHourSet Function

Writes the contents of the Hours bits in the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCHourSet(RTCC_MODULE_ID index, uint32_t hour);
```

Returns

None.

Description

The function writes the contents of the Hours bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Hour = 0x04;
PLIB_RTCC_RTCHourSet(RTCC_ID_0, Hour);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
hour	BCD value to be written to the Hours bits in the Time register

Function

uint32_t PLIB_RTCC_RTCHourSet (RTCC_MODULE_ID index, uint32_t hour)

PLIB_RTCC_RTCMinuteGet Function

Returns the contents of the Minutes bits in the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCMinuteGet(RTCC_MODULE_ID index);
```

Returns

BCD value of the Minutes bits in the Time register.

Description

The function returns the contents of the Minutes bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Minute;
Minute = PLIB_RTCC_RTCMinuteGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCMinuteGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCMinuteSet Function

Writes the contents of Minutes bits in the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCMinuteSet(RTCC_MODULE_ID index, uint32_t minute);
```

Returns

None.

Description

The function writes the contents of these bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Minute = 0x15;
PLIB_RTCC_RTCMinuteSet(RTCC_ID_0, Minute);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
minute	BCD value to be written to the Minutes bits in the Time register

Function

uint32_t PLIB_RTCC_RTCMinuteSet (RTCC_MODULE_ID index, uint32_t minute)

PLIB_RTCC_RTCMonthGet Function

Returns the contents of the Months bits in the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCMonthGet(RTCC_MODULE_ID index);
```

Returns

Months bits in the Date register.

Description

The function returns the contents of the Months bits in the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Month;
Month = PLIB_RTCC_RTCMonthGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCMonthGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCMonthSet Function

Writes to the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCMonthSet(RTCC_MODULE_ID index, uint32_t month);
```

Returns

None.

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t Month = 0x10;      //Month = October
PLIB_RTCC_RTCMonthSet(RTCC_ID_0, Month);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
month	The BCD value of the month to set in the Date register

Function

void PLIB_RTCC_RTCMonthSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_RTCSecondGet Function

The function returns the contents of the Seconds bits in the specific RTCC device's Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCSecondGet(RTCC_MODULE_ID index);
```

Returns

BCD value of the Seconds bits in the Time register.

Description

The function returns the contents of the Seconds bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Second;
Second = PLIB_RTCC_RTCSecondGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCSecondGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCSecondSet Function

Writes the contents of Seconds bits in the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCSecondSet(RTCC_MODULE_ID index, uint32_t second);
```

Returns

None.

Description

The function writes the contents of the Seconds bits in the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Second = 0x33;
PLIB_RTCC_RTCSecondSet(RTCC_ID_0, Second);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
second	BCD value to be written to the Seconds bits in the Time register

Function

uint32_t PLIB_RTCC_RTCSecondSet (RTCC_MODULE_ID index, uint32_t second)

PLIB_RTCC_RTCTimeGet Function

Returns the contents of the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCTimeGet(RTCC_MODULE_ID index);
```

Returns

Time register contents.

Description

The function returns the contents of the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Example

```
uint32_t time;
time = PLIB_RTCC_RTCTimeGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCTimeGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCTimeSet Function

Writes to the specific RTCC module's Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCTimeSet(RTCC_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

The function writes to the specific RTCC module's Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the RTCTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCTime in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Time register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t data = 0x04153300;     // Time = 4 hours, 15 minutes, and 33 seconds
PLIB_RTCC_RTCTimeSet(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_RTCTimeSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_RTCWeekDayGet Function

Returns the contents of the WeekDay bits in the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCWeekDayGet(RTCC_MODULE_ID index);
```

Returns

WeekDay field in the Date register.

Description

The function returns the contents of the WeekDay bits in the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t WeekDay;
WeekDay = PLIB_RTCC_RTCWeekDayGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCWeekDayGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCWeekDaySet Function

Writes to the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCWeekDaySet(RTCC_MODULE_ID index, uint32_t weekday);
```

Returns

None.

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t WeekDay = 0x05;        //WeekDay = Friday
PLIB_RTCC_RTCWeekDaySet(RTCC_ID_0, WeekDay);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
weekday	The BCD value of the weekday to set in the Date register

Function

void PLIB_RTCC_RTCWeekDaySet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_RTCYearGet Function

Returns the contents of the Year bits in the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_RTCYearGet(RTCC_MODULE_ID index);
```

Returns

Year bits in the Date register.

Description

The function returns the contents of the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Year;
Year = PLIB_RTCC_RTCYearGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_RTCYearGet (RTCC_MODULE_ID index)

PLIB_RTCC_RTCYearSet Function

Writes to the specific RTCC module's Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_RTCYearSet(RTCC_MODULE_ID index, uint32_t year);
```

Returns

None.

Description

The function writes to the specific RTCC module's Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the RTCDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsRTCDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t Year = 0x06;       //Year = 2006
PLIB_RTCC_RTCYearSet(RTCC_ID_0, Year);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
year	The BCD value of the year to set in the Date register

Function

void PLIB_RTCC_RTCYearSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_RTCSyncStatusGet Function

The function returns the synchronization status bit.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_RTCSyncStatusGet(RTCC_MODULE_ID index);
```

Returns

- true - Date and time will change within 32 RTCC clocks
- false - Date and time are safe to read, and will not change soon

Description

The function returns the synchronization status bit, which is used to determine whether it is safe to read the date/time values, or if the values will change within 32 RTCC clocks.

Remarks

This bit is read-only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsSynchronization in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_RTCC_RTCSyncStatusGet(RTCC_ID_0))
{
...
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_RTCC_RTCSyncStatusGet (RTCC_MODULE_ID index)

c) Alarm Mode Functions

PLIB_RTCC_AlarmChimeDisable Function

Disables the specific RTCC module's chime.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmChimeDisable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific RTCC module's chime. The alarm repeat count value bits stop once they reach zero.

Remarks

This function implements an operation of the AlarmChimeControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmChimeControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_AlarmChimeDisable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_AlarmChimeDisable (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmChimeEnable Function

Enables the specific RTCC module's chime.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmChimeEnable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific RTCC module's chime. The alarm repeat count bits are allowed to rollover.

Remarks

This function implements an operation of the AlarmChimeControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmChimeControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_AlarmChimeEnable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_AlarmChimeEnable (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmDateGet Function

Returns the contents of the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmDateGet(RTCC_MODULE_ID index);
```

Returns

Value register.

Description

The function returns the contents of the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Date;
Date = PLIB_RTCC_AlarmDateGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmDateGet (AlarmC_MODULE_ID index)

PLIB_RTCC_AlarmDateSet Function

Writes to the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmDateSet(RTCC_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

The function writes to the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Alarm Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t data = 0x06102705;     //Date = 27 Oct 2006 Friday
PLIB_RTCC_AlarmDateSet(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	The value to set the Alarm Date register to, in BCD format

Function

void PLIB_RTCC_AlarmDateSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_AlarmDayGet Function

Returns the contents of the Day bits in the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmDayGet(RTCC_MODULE_ID index);
```

Returns

Days bits in the Alarm Date register.

Description

The function returns the contents of Day bits in the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Day;
Day = PLIB_RTCC_AlarmDayGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmDayGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmDaySet Function

Writes to the specific RTCC module's Alarm Date value register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmDaySet(RTCC_MODULE_ID index, uint32_t day);
```

Returns

None.

Description

The function writes to the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Alarm Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t Day = 0x27;        //Day = 27th of the month
PLIB_RTCC_AlarmDaySet(RTCC_ID_0, Day);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
day	The BCD value of the day to set in the Alarm Date register

Function

void PLIB_RTCC_AlarmDaySet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_AlarmDisable Function

Disables the specific RTCC module's alarm.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmDisable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific RTCC module's alarm. RTCC Alarm bit shouldn't be modified when RTCC on bit is enabled and PLIB_RTCC_AlarmSyncStatusGet (ALRMSYNC bit) returns true. Meaning the check RTCC ON (RTCCON<15>) bit and the ALRMSYNC bit should be equal to '1'.

Remarks

This function implements an operation of the AlarmControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// check RTCC enable bit and PLIB_RTCC_AlarmSyncStatusGet return value and
// then modify the Alarm bit.
PLIB_RTCC_AlarmDisable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_AlarmDisable (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmEnable Function

Enables the specific RTCC module's alarm.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmEnable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific RTCC module's alarm. RTCC Alarm bit shouldn't be modified when RTCC on bit is enabled and

PLIB_RTCC_AlarmSyncStatusGet (ALRMSYNC bit) returns true. Meaning the check RTCC ON (RTCCON<15>) bit and the ALRMSYNC bit should be equal to '1'.

Remarks

The alarm enable bit is cleared automatically after an alarm event whenever the alarm is not set up to repeat, and chime is disabled.

This function implements an operation of the AlarmControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// check RTCC enable bit and PLIB_RTCC_AlarmSyncStatusGet return value and
// then modify the Alarm bit.
PLIB_RTCC_AlarmEnable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_AlarmEnable (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmHourGet Function

Returns the contents of Hours bits in the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmHourGet(RTCC_MODULE_ID index);
```

Returns

BCD value of the Hours bits in the Alarm Time register.

Description

The function returns the contents of the Hours bits in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Hour;
Hour = PLIB_RTCC_AlarmHourGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmHourGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmHourSet Function

The function returns the contents of Hours bits in the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmHourSet(RTCC_MODULE_ID index, uint32_t hour);
```

Returns

None

Description

Returns the contents of the Hours bits in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Hour = 0x04;
PLIB_RTCC_AlarmHourSet(RTCC_ID_0, Hour);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
hour	BCD value to be written to the Hours bits in the Alarm Time register

Function

uint32_t PLIB_RTCC_AlarmHourSet (RTCC_MODULE_ID index, uint32_t hour)

PLIB_RTCC_AlarmMaskModeSelect Function

Sets the specific RTCC module's alarm mask Configuration bits.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmMaskModeSelect(RTCC_MODULE_ID index, RTCC_ALARM_MASK_CONFIGURATION data);
```

Returns

None.

Description

This function sets the specific RTCC module's alarm mask Configuration bits.

Remarks

The actual definition of this enumeration is device-specific.

This function implements an operation of the AlarmMaskControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmMaskControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t data = 0;
PLIB_RTCC_AlarmMaskModeSelect(RTCC_ID_0, RTCC_ALARM_EVERY_HALF_SECOND);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Alarm mask Configuration bits

Function

void PLIB_RTCC_AlarmMaskModeSelect (RTCC_MODULE_ID index, RTCC_ALARM_MASK_CONFIGURATION data)

PLIB_RTCC_AlarmMinuteGet Function

Returns the contents of Minutes bits in the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmMinuteGet(RTCC_MODULE_ID index);
```

Returns

BCD value of the Minutes bits in the Alarm Time register.

Description

The function returns the contents of the field in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Minute;
Minute = PLIB_RTCC_AlarmMinuteGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmMinuteGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmMinuteSet Function

Returns the contents of the Minutes bits in the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmMinuteSet(RTCC_MODULE_ID index, uint32_t minute);
```

Returns

None

Description

The function returns the contents of the Minutes bits in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Minute = 0x15;
PLIB_RTCC_AlarmMinuteSet(RTCC_ID_0, Minute);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
minute	BCD value to be written to the Minutes bits in the Alarm Time register

Function

uint32_t PLIB_RTCC_AlarmMinuteSet (RTCC_MODULE_ID index, uint32_t minute)

PLIB_RTCC_AlarmMonthGet Function

Returns the contents of the Month bits in the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmMonthGet(RTCC_MODULE_ID index);
```

Returns

Months bits in the Date register.

Description

The function returns the contents of the Months bits in the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Month;
Month = PLIB_RTCC_AlarmMonthGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmMonthGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmMonthSet Function

Writes to the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmMonthSet(RTCC_MODULE_ID index, uint32_t month);
```

Returns

None.

Description

The function writes to the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Alarm Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t Month = 0x10;      //Month = October
PLIB_RTCC_AlarmMonthSet(RTCC_ID_0, Month);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
month	The BCD value of the month to set in the Alarm Date register

Function

void PLIB_RTCC_AlarmMonthSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_AlarmPulseInitialGet Function

Returns the state of the initial alarm pulse.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_AlarmPulseInitialGet(RTCC_MODULE_ID index);
```

Returns

- 1 - Logical High
- 0 - Logical Low

Description

This function returns the state of the initial alarm pulse.

Remarks

This function implements an operation of the AlarmPulseInitial feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmPulseInitial in your application to automatically determine whether this feature is available.

Preconditions

The ALRMEN bit should be '1' indicating the PLIB_RTCC_AlarmEnable function was called.

Example

```
bool PulseValue;
PulseValue = PLIB_RTCC_AlarmPulseInitialGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_RTCC_AlarmPulseInitialGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmPulseInitialSet Function

Enables the determination of the initial alarm pulse.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmPulseInitialSet(RTCC_MODULE_ID index, bool data);
```

Returns

None.

Description

This function enables the determination of initial alarm pulse.

Remarks

This function implements an operation of the AlarmPulseInitial feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmPulseInitial in your application to automatically determine whether this feature is available.

Preconditions

The ALRMEN bit should be '0' indicating the PLIB_RTCC_AlarmDisable was called. This function must not be called when the RTCC is ON and the Alarm Sync is 1.

Example

```
bool data = 0;
PLIB_RTCC_AlarmPulseInitialSet(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_AlarmPulseInitialSet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmRepeatCountGet Function

Reads the specific RTCC module's alarm repeat counter.

File

plib_rtcc.h

C

```
uint8_t PLIB_RTCC_AlarmRepeatCountGet(RTCC_MODULE_ID index);
```

Returns

uint8_t - The current value of the alarm repeat counter

Description

This function reads the specific RTCC module's alarm repeat counter.

Remarks

The counter decrements on any alarm event. The counter is prevented from rolling over unless chime is enabled.

This function implements an operation of the AlarmRepeatControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmRepeatControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t currentCount;
currentCount = PLIB_RTCC_AlarmRepeatCountGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_RTCC_AlarmRepeatCountGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmRepeatCountSet Function

Sets the specific RTCC module's alarm repeat counter.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmRepeatCountSet(RTCC_MODULE_ID index, uint8_t data);
```

Returns

None.

Description

This function sets the specific RTCC module's alarm repeat counter.

Remarks

The counter decrements on any alarm event. The counter is prevented from rolling over unless chime is enabled.

This function implements an operation of the AlarmRepeatControl feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmRepeatControl in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t data = 0xFF;
PLIB_RTCC_AlarmRepeatCountSet(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Alarm repeat counter bits

Function

void PLIB_RTCC_AlarmRepeatCountSet (RTCC_MODULE_ID index, uint8_t data)

PLIB_RTCC_AlarmSecondGet Function

Returns the contents of the Seconds bits in the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmSecondGet(RTCC_MODULE_ID index);
```

Returns

BCD value of the Seconds bits in the Alarm Time register.

Description

The function returns the contents of the Seconds bits in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Second;
Second = PLIB_RTCC_AlarmSecondGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmSecondGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmSecondSet Function

Returns the contents of Seconds bits in the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmSecondSet(RTCC_MODULE_ID index, uint32_t second);
```

Returns

None

Description

The function returns the contents of the field in the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t Second = 0x33;
PLIB_RTCC_AlarmSecondSet(RTCC_ID_0, Second);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
second	BCD value to be written to the Seconds bits in the Alarm Time register

Function

uint32_t PLIB_RTCC_AlarmSecondSet (RTCC_MODULE_ID index, uint32_t second)

PLIB_RTCC_AlarmTimeGet Function

Returns the contents of the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmTimeGet(RTCC_MODULE_ID index);
```

Returns

Value register.

Description

The function returns the contents of the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t time;
time = PLIB_RTCC_AlarmTimeGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmTimeGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmTimeSet Function

Writes to the specific RTCC module's Alarm Time register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmTimeSet(RTCC_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

The function writes to the specific RTCC module's Alarm Time register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the AlarmTime feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmTime in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Alarm Time register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t data = 0x04153300;     // Time = 4 hours, 15 minutes, and 33 seconds
PLIB_RTCC_AlarmTimeSet(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_AlarmTimeSet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_AlarmValueRegisterPointer Function

Sets the specific RTCC module's Alarm register pointer.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmValueRegisterPointer(RTCC_MODULE_ID index, uint8_t data);
```

Returns

None.

Description

This function sets the specific RTCC module's Alarm register pointer.

Remarks

None.

Preconditions

None.

Example

```
uint8_t data = 2;
PLIB_RTCC_AlarmValueRegisterPointer(RTCC_ID_0, data);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Alarm register pointer

Function

void PLIB_RTCC_AlarmValueRegisterPointer (RTCC_MODULE_ID index, uint8_t data)

PLIB_RTCC_AlarmWeekDayGet Function

Returns the contents of the WeekDay bits in the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
uint32_t PLIB_RTCC_AlarmWeekDayGet(RTCC_MODULE_ID index);
```

Returns

WeekDay bits in the Alarm Date register.

Description

The function returns the contents of Weekday bits in the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint32_t WeekDay;
WeekDay = PLIB_RTCC_AlarmWeekDayGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_RTCC_AlarmWeekDayGet (RTCC_MODULE_ID index)

PLIB_RTCC_AlarmWeekDaySet Function

Writes to the specific RTCC module's Alarm Date register.

File

plib_rtcc.h

C

```
void PLIB_RTCC_AlarmWeekDaySet(RTCC_MODULE_ID index, uint32_t weekday);
```

Returns

None.

Description

The function writes to the specific RTCC module's Alarm Date register. Please refer to the specific device data sheet for the exact sequence of digits.

Remarks

A write to this register is only allowed when access is allowed by using the PLIB_RTCC_WriteEnable function.

This function implements an operation of the AlarmDate feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmDate in your application to automatically determine whether this feature is available.

Preconditions

Prior to writing to the Alarm Date register, an RTCC write must be enabled using the exact sequences required by the device.

Example

```
uint32_t WeekDay = 0x05;        //WeekDay = Friday
PLIB_RTCC_AlarmWeekDaySet(RTCC_ID_0, WeekDay);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
weekday	The BCD value of the weekday to set in the field of the Alarm Date register

Function

void PLIB_RTCC_AlarmWeekDaySet (RTCC_MODULE_ID index, uint32_t data)

PLIB_RTCC_AlarmSyncStatusGet Function

The function returns the synchronization status bit.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_AlarmSyncStatusGet(RTCC_MODULE_ID index);
```

Returns

- true - Alarm repeat count may change as a result of a half-second rollover during a read.
- false - Alarm repeat count is safe to read, and will not change in less than 32 RTC clocks.

Description

The function returns the synchronization status bit, which is used to determine whether it is safe to read the date/time values, or if the values will change within 32 RTCC clocks.

Remarks

This bit is read-only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsAlarmSynchronization in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_RTCC_AlarmSyncStatusGet(RTCC_ID_0))
{
...
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_RTCC_AlarmSyncStatusGet (RTCC_MODULE_ID index)

d) Other Functions

PLIB_RTCC_ClockSourceSelect Function

Selects the clock source for the RTCC module.

File

plib_rtcc.h

C

```
void PLIB_RTCC_ClockSourceSelect(RTCC_MODULE_ID index, RTCC_CLOCK_SOURCE_SELECT source);
```

Returns

None.

Description

This function determines which clock source the RTCC module will use depending on the features of the device.

Remarks

This function implements an operation of the ClockSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsClockSelect in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_ClockSourceSelect(RTCC_ID_0, RTCC_CLOCK_SOURCE_SELECT_NONE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
source	Which clock source will be used

Function

void PLIB_RTCC_ClockSourceSelect (RTCC_MODULE_ID index, RTCC_CLOCK_SOURCE_SELECT source)

PLIB_RTCC_DriftCalibrateGet Function

Reads the specific RTCC module's drift calibration bits.

File

plib_rtcc.h

C

```
uint16_t PLIB_RTCC_DriftCalibrateGet(RTCC_MODULE_ID index);
```

Returns

uint16_t - The current drift calibration value

Description

This function reads the specific RTCC module's drift calibration bits.

Remarks

This function implements an operation of the Calibration feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCalibration in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t calibrationbits;
calibrationbits = PLIB_RTCC_DriftCalibrateGet(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint16_t PLIB_RTCC_DriftCalibrateGet (RTCC_MODULE_ID index)

PLIB_RTCC_DriftCalibrateSet Function

Sets the specific RTCC module's drift calibration bits.

File

plib_rtcc.h

C

```
void PLIB_RTCC_DriftCalibrateSet(RTCC_MODULE_ID index, uint16_t calibrationbits);
```

Returns

None.

Description

This function sets the specific RTCC module's drift calibration bits. The error between the system clock and the external clock has to be computed and calibration input must be provided to this function.

Remarks

This function implements an operation of the Calibration feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsCalibration in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
uint16_t calibrationbits = 3; //Positive 3 adjustment derived from the formula
                              // Error = (Ideal Freq(32758) - Measured)*60;
PLIB_RTCC_DriftCalibrateSet(RTCC_ID_0, calibrationbits);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
calibrationbits	Drift calibration bits

Function

void PLIB_RTCC_DriftCalibrateSet (RTCC_MODULE_ID index, uint16_t calibrationbits)

PLIB_RTCC_HalfSecondStatusGet Function

The function returns the half second status bit.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_HalfSecondStatusGet(RTCC_MODULE_ID index);
```

Returns

- true - Second half period of a second
- false - First half period of a second

Description

The function returns the half second status bit, which is used in the calibration procedure. When the seconds byte is zero, the calibration value must be updated when the half second bit becomes '1'.

Remarks

This bit is read-only. It is cleared to '0' on a write to the seconds value. This function implements an operation of the HalfSecond feature. This

feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsHalfSecond in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Wait for the half second status bit to be '1'.
while(PLIB_RTCC_HalfSecondStatusGet(RTCC_ID_0));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_RTCC_HalfSecondStatusGet (RTCC_MODULE_ID index)

PLIB_RTCC_OutputSelect Function

Selects which signal will be presented on the RTCC pin

File

plib_rtcc.h

C

```
void PLIB_RTCC_OutputSelect(RTCC_MODULE_ID index, RTCC_OUTPUT_SELECT data);
```

Returns

None.

Description

This function selects which signal will be presented on the RTCC pin.

Remarks

The RTCC module's output pin should be enabled using the function PLIB_RTCC_OutputEnable.

This function implements an operation of the OutputSelect feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsOutputSelect in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_OutputSelect(RTCC_ID_0, RTCC_OUTPUT_SECONDS_CLOCK);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Enumerated value of which signal to present

Function

void PLIB_RTCC_OutputSelect (RTCC_MODULE_ID index, RTCC_OUTPUT_SELECT data)

PLIB_RTCC_StopInIdleDisable Function

Continues normal RTCC operation when the device enters Idle mode.

File

plib_rtcc.h

C

```
void PLIB_RTCC_StopInIdleDisable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function continues normal RTCC operation when the device enters Idle mode.

Remarks

None.

This function implements an operation of the StopInIdle feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsStopInIdle in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_StopInIdleDisable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_StopInIdleDisable(RTCC_MODULE_ID index)

PLIB_RTCC_StopInIdleEnable Function

Disables access to the RTCC module by the Peripheral Bus Clock (PBCLK) when the CPU enters Idle mode.

File

plib_rtcc.h

C

```
void PLIB_RTCC_StopInIdleEnable(RTCC_MODULE_ID index);
```

Returns

None.

Description

This function disables access to the RTCC module by the PBCLK when the CPU is in Idle mode.

Remarks

This function implements an operation of the StopInIdle feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_RTCC_ExistsStopInIdle in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_RTCC_StopInIdleEnable(RTCC_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_RTCC_StopInIdleEnable(RTCC_MODULE_ID index)

e) Feature Existence Functions

PLIB_RTCC_ExistsAlarmChimeControl Function

Identifies whether the AlarmChimeControl feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmChimeControl(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmChimeControl feature is supported on the device
- false - The AlarmChimeControl feature is not supported on the device

Description

This function identifies whether the AlarmChimeControl feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_AlarmChimeEnable
- PLIB_RTCC_AlarmChimeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmChimeControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmControl Function

Identifies whether the AlarmControl feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmControl(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmControl feature is supported on the device
- false - The AlarmControl feature is not supported on the device

Description

This function identifies whether the AlarmControl feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_AlarmEnable
- PLIB_RTCC_AlarmDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmDate Function

Identifies whether the AlarmDate feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmDate(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmDate feature is supported on the device
- false - The AlarmDate feature is not supported on the device

Description

This function identifies whether the AlarmDate feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_AlarmDateGet
- PLIB_RTCC_AlarmDateSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmDate(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmMaskControl Function

Identifies whether the AlarmMaskControl feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmMaskControl(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmMaskControl feature is supported on the device
- false - The AlarmMaskControl feature is not supported on the device

Description

This function identifies whether the AlarmMaskControl feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_AlarmMaskModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmMaskControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmPulseInitial Function

Identifies whether the AlarmPulseInitial feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmPulseInitial(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmPulseInitial feature is supported on the device
- false - The AlarmPulseInitial feature is not supported on the device

Description

This function identifies whether the AlarmPulseInitialValue feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_AlarmPulseInitialSet
- PLIB_RTCC_AlarmPulseInitialGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmPulseInitial(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmRepeatControl Function

Identifies whether the AlarmRepeatControl feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmRepeatControl(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmRepeatControl feature is supported on the device
- false - The AlarmRepeatControl feature is not supported on the device

Description

This function identifies whether the AlarmRepeatControl feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_AlarmRepeatCountSet
- PLIB_RTCC_AlarmRepeatCountRead

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmRepeatControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmTime Function

Identifies whether the AlarmTime feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmTime(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmTime feature is supported on the device
- false - The AlarmTime feature is not supported on the device

Description

This function identifies whether the AlarmTime feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_AlarmTimeGet
- PLIB_RTCC_AlarmTimeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmTime(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsCalibration Function

Identifies whether the Calibration feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsCalibration(RTCC_MODULE_ID index);
```

Returns

- true - The Calibration feature is supported on the device
- false - The Calibration feature is not supported on the device

Description

This function identifies whether the Calibration feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_DriftCalibrate

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsCalibration(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsClockRunning Function

Identifies whether the ClockRunning feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsClockRunning(RTCC_MODULE_ID index);
```

Returns

- true - The ClockRunning feature is supported on the device
- false - The ClockRunning feature is not supported on the device

Description

This function identifies whether the ClockRunning feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_ClockRunningStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsClockRunning(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsClockSelect Function

Identifies whether the ClockSelect feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsClockSelect(RTCC_MODULE_ID index);
```

Returns

- true - The ClockSelect feature is supported on the device
- false - The ClockSelect feature is not supported on the device

Description

This function identifies whether the ClockSelect feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_SourceClockSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsClockSelect(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsEnableControl(RTCC_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_Enable
- PLIB_RTCC_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsEnableControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsHalfSecond Function

Identifies whether the HalfSecond feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsHalfSecond(RTCC_MODULE_ID index);
```

Returns

- true - The HalfSecond feature is supported on the device
- false - The HalfSecond feature is not supported on the device

Description

This function identifies whether the HalfSecond feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_HalfSecondStatusBit

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsHalfSecond(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsOutputControl Function

Identifies whether the OutputControl feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsOutputControl(RTCC_MODULE_ID index);
```

Returns

- true - The OutputControl feature is supported on the device
- false - The OutputControl feature is not supported on the device

Description

This function identifies whether the OutputControl feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_ClockOutputEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsOutputControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsOutputSelect Function

Identifies whether the OutputSelect feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsOutputSelect(RTCC_MODULE_ID index);
```

Returns

- true - The OutputSelect feature is supported on the device
- false - The OutputSelect feature is not supported on the device

Description

This function identifies whether the OutputSelect feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_SecondsClockOutputSelect
- PLIB_RTCC_AlarmPulseOutputSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsOutputSelect(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsRTCDate Function

Identifies whether the RTCDateValue feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsRTCDate(RTCC_MODULE_ID index);
```

Returns

- true - The RTCDate feature is supported on the device
- false - The RTCDate feature is not supported on the device

Description

This function identifies whether the RTCDateValue feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_RTCDateGet
- PLIB_RTCC_RTCDateSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsRTCDate(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsRTCTime Function

Identifies whether the RTCTimeValue feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsRTCTime(RTCC_MODULE_ID index);
```

Returns

- true - The RTCTime feature is supported on the device
- false - The RTCTime feature is not supported on the device

Description

This function identifies whether the RTCTimeValue feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_RTCTimeGet
- PLIB_RTCC_RTCTimeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsRTCTime(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsStopInIdleControl Function

Identifies whether the StopInIdle feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsStopInIdleControl(RTCC_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_StopInIdleEnable
- PLIB_RTCC_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsStopInIdleControl(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsSynchronization Function

Identifies whether the Synchronization feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsSynchronization(RTCC_MODULE_ID index);
```

Returns

- true - The Synchronization feature is supported on the device
- false - The Synchronization feature is not supported on the device

Description

This function identifies whether the Synchronization feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_RTCSyncStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsSynchronization(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsWriteEnable Function

Identifies whether the WriteEnable feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsWriteEnable(RTCC_MODULE_ID index);
```

Returns

- true - The WriteEnable feature is supported on the device
- false - The WriteEnable feature is not supported on the device

Description

This function identifies whether the WriteEnable feature is available on the RTCC module. When this interface returns true, these functions are supported on the device:

- PLIB_RTCC_WriteEnable
- PLIB_RTCC_WriteDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsWriteEnable(RTCC_MODULE_ID index)

PLIB_RTCC_ExistsAlarmSynchronization Function

Identifies whether the AlarmSynchronization feature exists on the RTCC module.

File

plib_rtcc.h

C

```
bool PLIB_RTCC_ExistsAlarmSynchronization(RTCC_MODULE_ID index);
```

Returns

- true - The AlarmSynchronization feature is supported on the device
- false - The AlarmSynchronization feature is not supported on the device

Description

This function identifies whether the AlarmSynchronization feature is available on the RTCC module. When this interface returns true, this function is supported on the device:

- PLIB_RTCC_AlarmSyncGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_RTCC_ExistsAlarmSynchronization(RTCC_MODULE_ID index)

f) Data Types and Constants

RTCC_ALARM_MASK_CONFIGURATION Enumeration

Data type defining the different configurations for the alarm mask bits.

File

plib_rtcc_help.h

C

```
typedef enum {
  RTCC_ALARM_EVERY_HALF_SECOND,
  RTCC_ALARM_EVERY_SECOND,
  RTCC_ALARM_EVERY_10_SECONDS,
  RTCC_ALARM_EVERY_MINUTE,
  RTCC_ALARM_EVERY_10_MINUTES,
  RTCC_ALARM_EVERY_HOUR,
  RTCC_ALARM_ONCE_A_DAY,
  RTCC_ALARM_ONCE_A_WEEK,
  RTCC_ALARM_ONCE_A_MONTH,
  RTCC_ALARM_ONCE_A_YEAR
} RTCC_ALARM_MASK_CONFIGURATION;
```

Members

Members	Description
RTCC_ALARM_EVERY_HALF_SECOND	Alarm every half second
RTCC_ALARM_EVERY_SECOND	Alarm every second
RTCC_ALARM_EVERY_10_SECONDS	Alarm every 10 seconds
RTCC_ALARM_EVERY_MINUTE	Alarm every minute
RTCC_ALARM_EVERY_10_MINUTES	Alarm every 10 minutes
RTCC_ALARM_EVERY_HOUR	Alarm every hour
RTCC_ALARM_ONCE_A_DAY	Alarm every day
RTCC_ALARM_ONCE_A_WEEK	Alarm every week
RTCC_ALARM_ONCE_A_MONTH	Alarm every month
RTCC_ALARM_ONCE_A_YEAR	Alarm every year

Description

Alarm Mask Configuration

This data type defines the different configurations for accessing the alarm mask Configuration bits.

Remarks

The actual definition of this enumeration is device-specific.

RTCC_MODULE_ID Enumeration

File

plib_rtcc_help.h

C

```
typedef enum {
  RTCC_ID_1,
  RTCC_NUMBER_OF_MODULES
} RTCC_MODULE_ID;
```

Members

Members	Description
RTCC_NUMBER_OF_MODULES	The total number of modules available.

Description

Enumeration: RTCC_MODULE_ID

This enumeration defines number of modules which are available on the microcontroller. This is the superset of all of the possible instances that may be available on the Microchip microcontrollers.

Refer to the data sheet to get the correct number of modules defined for desired microcontroller.

RTCC_VALUE_REGISTER_POINTER Enumeration

Data type defining the different configurations by which the RTCC Date and Time Registers can be accessed.

File

plib_rtcc_help.h

C

```
typedef enum {
  RTCC_REG_YEAR,
  RTCC_REG_DAY_MONTH,
  RTCC_REG_HOURS_WEEKDAY,
  RTCC_REG_MINUTES_SECONDS
} RTCC_VALUE_REGISTER_POINTER;
```

Members

Members	Description
RTCC_REG_YEAR	The year register is being pointed
RTCC_REG_DAY_MONTH	The day and month register is being pointed
RTCC_REG_HOURS_WEEKDAY	The hours and weekday register is being pointed
RTCC_REG_MINUTES_SECONDS	The minutes and seconds register is being pointed

Description

Value Register Pointer

This data type defines the different configurations by which the RTCC Date and Time Registers can be accessed.

Remarks

The actual definition of this enumeration is device-specific.

Files

Files

Name	Description
plib_rtcc.h	RTCC Peripheral Library interface header for RTCC common definitions.
plib_rtcc_help.h	This is file plib_rtcc_help.h.

Description

This section lists the source and header files used by the library.

plib_rtcc.h

RTCC Peripheral Library interface header for RTCC common definitions.

Functions

	Name	Description
	PLIB_RTCC_AlarmChimeDisable	Disables the specific RTCC module's chime.
	PLIB_RTCC_AlarmChimeEnable	Enables the specific RTCC module's chime.
	PLIB_RTCC_AlarmDateGet	Returns the contents of the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmDateSet	Writes to the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmDayGet	Returns the contents of the Day bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmDaySet	Writes to the specific RTCC module's Alarm Date value register.
	PLIB_RTCC_AlarmDisable	Disables the specific RTCC module's alarm.
	PLIB_RTCC_AlarmEnable	Enables the specific RTCC module's alarm.
	PLIB_RTCC_AlarmHourGet	Returns the contents of Hours bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmHourSet	The function returns the contents of Hours bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMaskModeSelect	Sets the specific RTCC module's alarm mask Configuration bits.
	PLIB_RTCC_AlarmMinuteGet	Returns the contents of Minutes bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMinuteSet	Returns the contents of the Minutes bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmMonthGet	Returns the contents of the Month bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmMonthSet	Writes to the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmPulseInitialGet	Returns the state of the initial alarm pulse.
	PLIB_RTCC_AlarmPulseInitialSet	Enables the determination of the initial alarm pulse.

	PLIB_RTCC_AlarmRepeatCountGet	Reads the specific RTCC module's alarm repeat counter.
	PLIB_RTCC_AlarmRepeatCountSet	Sets the specific RTCC module's alarm repeat counter.
	PLIB_RTCC_AlarmSecondGet	Returns the contents of the Seconds bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmSecondSet	Returns the contents of Seconds bits in the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmSyncStatusGet	The function returns the synchronization status bit.
	PLIB_RTCC_AlarmTimeGet	Returns the contents of the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmTimeSet	Writes to the specific RTCC module's Alarm Time register.
	PLIB_RTCC_AlarmValueRegisterPointer	Sets the specific RTCC module's Alarm register pointer.
	PLIB_RTCC_AlarmWeekDayGet	Returns the contents of the WeekDay bits in the specific RTCC module's Alarm Date register.
	PLIB_RTCC_AlarmWeekDaySet	Writes to the specific RTCC module's Alarm Date register.
	PLIB_RTCC_ClockOutputDisable	Disables the specific RTCC module's output pin.
	PLIB_RTCC_ClockOutputEnable	Enables the specific RTCC module's output pin.
	PLIB_RTCC_ClockRunningStatus	Provides the status of the RTCC clock.
	PLIB_RTCC_ClockSourceSelect	Selects the clock source for the RTCC module.
	PLIB_RTCC_Disable	Disables the specific RTCC module on the device.
	PLIB_RTCC_DriftCalibrateGet	Reads the specific RTCC module's drift calibration bits.
	PLIB_RTCC_DriftCalibrateSet	Sets the specific RTCC module's drift calibration bits.
	PLIB_RTCC_Enable	Enables the specific RTCC module on the device.
	PLIB_RTCC_ExistsAlarmChimeControl	Identifies whether the AlarmChimeControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmControl	Identifies whether the AlarmControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmDate	Identifies whether the AlarmDate feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmMaskControl	Identifies whether the AlarmMaskControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmPulseInitial	Identifies whether the AlarmPulseInitial feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmRepeatControl	Identifies whether the AlarmRepeatControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmSynchronization	Identifies whether the AlarmSynchronization feature exists on the RTCC module.
	PLIB_RTCC_ExistsAlarmTime	Identifies whether the AlarmTime feature exists on the RTCC module.
	PLIB_RTCC_ExistsCalibration	Identifies whether the Calibration feature exists on the RTCC module.
	PLIB_RTCC_ExistsClockRunning	Identifies whether the ClockRunning feature exists on the RTCC module.
	PLIB_RTCC_ExistsClockSelect	Identifies whether the ClockSelect feature exists on the RTCC module.
	PLIB_RTCC_ExistsEnableControl	Identifies whether the EnableControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsHalfSecond	Identifies whether the HalfSecond feature exists on the RTCC module.
	PLIB_RTCC_ExistsOutputControl	Identifies whether the OutputControl feature exists on the RTCC module.
	PLIB_RTCC_ExistsOutputSelect	Identifies whether the OutputSelect feature exists on the RTCC module.
	PLIB_RTCC_ExistsRTCDate	Identifies whether the RTCDateValue feature exists on the RTCC module.
	PLIB_RTCC_ExistsRTCTime	Identifies whether the RTCTimeValue feature exists on the RTCC module.
	PLIB_RTCC_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the RTCC module.
	PLIB_RTCC_ExistsSynchronization	Identifies whether the Synchronization feature exists on the RTCC module.
	PLIB_RTCC_ExistsWriteEnable	Identifies whether the WriteEnable feature exists on the RTCC module.
	PLIB_RTCC_HalfSecondStatusGet	The function returns the half second status bit.
	PLIB_RTCC_OutputSelect	Selects which signal will be presented on the RTCC pin
	PLIB_RTCC_RTCDateGet	Returns the contents of the specific RTCC module's Date register.
	PLIB_RTCC_RTCDateSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCDayGet	Returns the contents of the Days bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCDaySet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCHourGet	Returns the contents of the Hours bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCHourSet	Writes the contents of the Hours bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMinuteGet	Returns the contents of the Minutes bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMinuteSet	Writes the contents of Minutes bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCMonthGet	Returns the contents of the Months bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCMonthSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCSecondGet	The function returns the contents of the Seconds bits in the specific RTCC device's Time register.
	PLIB_RTCC_RTCSecondSet	Writes the contents of Seconds bits in the specific RTCC module's Time register.
	PLIB_RTCC_RTCSyncStatusGet	The function returns the synchronization status bit.

	PLIB_RTCC_RTCTimeGet	Returns the contents of the specific RTCC module's Time register.
	PLIB_RTCC_RTCTimeSet	Writes to the specific RTCC module's Time register.
	PLIB_RTCC_RTCWeekDayGet	Returns the contents of the WeekDay bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCWeekDaySet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_RTCYearGet	Returns the contents of the Year bits in the specific RTCC module's Date register.
	PLIB_RTCC_RTCYearSet	Writes to the specific RTCC module's Date register.
	PLIB_RTCC_StopInIdleDisable	Continues normal RTCC operation when the device enters Idle mode.
	PLIB_RTCC_StopInIdleEnable	Disables access to the RTCC module by the Peripheral Bus Clock (PBCLK) when the CPU enters Idle mode.
	PLIB_RTCC_WriteDisable	Disables writing to the specific RTCC module's value registers.
	PLIB_RTCC_WriteEnable	Enables writing to the specific RTCC module's value registers.

Description

Real-Time Clock and Calendar (RTCC) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the RTCC Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the RTCC peripheral.

File Name

plib_rtcc.h

Company

Microchip Technology Inc.

plib_rtcc_help.h

Enumerations

	Name	Description
	RTCC_ALARM_MASK_CONFIGURATION	Data type defining the different configurations for the alarm mask bits.
	RTCC_MODULE_ID	Enumeration: RTCC_MODULE_ID This enumeration defines number of modules which are available on the microcontroller. This is the superset of all of the possible instances that may be available on the Microchip microcontrollers. Refer to the data sheet to get the correct number of modules defined for desired microcontroller.
	RTCC_VALUE_REGISTER_POINTER	Data type defining the different configurations by which the RTCC Date and Time Registers can be accessed.

Description

This is file plib_rtcc_help.h.

System Bus Peripheral Library

This section describes the System Bus Peripheral Library.

Introduction

This library provides a low-level abstraction of the System Bus on Microchip microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The System Bus is a multi-layer crossbar switching matrix that connects initiators to targets. An initiator is a device or peripheral that initiates a data transfer to or from a target. Examples of initiators include the CPU, DMA and Ethernet controller. A target is a memory, bus or peripheral that sends or receives data to/from an initiator.

The System Bus allows the system programmer to control which initiators can access each target by defining permission groups for each initiator. Each target has one or more regions (physical address spaces) that can be individually programmed to allow access to different permission groups. By setting initiator permission groups and target region read/write permissions, the system programmer can allow or deny task or program access to system resources.

Using the Library

This topic describes the basic architecture of the System Bus Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_sb.h`

The interface to the System Bus Peripheral library is defined in the `plib_sb.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the System Bus Peripheral library must include `peripheral.h`.

Library File:

The System Bus Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

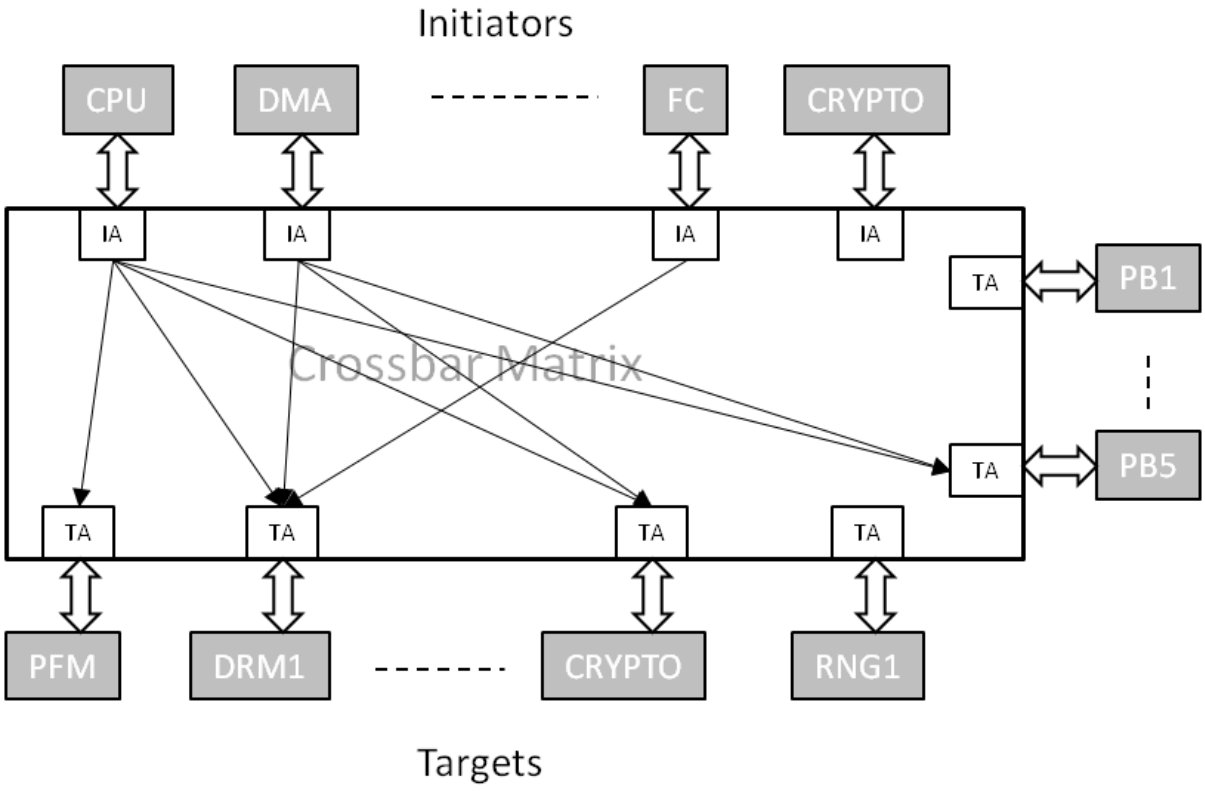
Please refer to the What is MPLAB Harmony? section for how the peripheral interacts with the framework.

Hardware Abstraction Model

This library provides the low-level abstraction of the System Bus on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

System Bus Software Abstraction Block Diagram



Hardware Abstraction Model

The System Bus Peripheral Library provides interface routines to initialize the bus matrix such that:

- Initiator permission groups are assigned
- Target regions are defined and given read/write access permissions
- Errors are identified, logged and cleared when a permission group violation occurs
- CPU and DMA priorities are set

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the System Bus module.

Library Interface Section	Description
Target Initialization Functions	This section provides target initialization functions.
PGV Error Functions	This section provides PGV error functions.
Other Functions	This section provides additional System Bus functions.
Feature Existence Functions	This section provides functions that determine whether certain features exist.

How the Library Works

The System Bus is a crossbar interconnect matrix that provides for simultaneous data transfer between the CPU, memory, and various peripherals on the PIC32 device. In addition, it provides a protection mechanism that allows trusted code to define and restrict access to memory regions and peripherals. A typical application would be a secure bootloader that prevents a third party or untrusted application from accessing regions of Flash memory that contain privileged code or information. The protection mechanism works by dividing the target physical address space into regions, each of which may be accessed by one or more initiators. Individual targets may have one or more regions. Each region can be individually programmed to allow read or write access to one or more permission groups. Each initiator is assigned to a permission group.

This library provides interface routines to initialize the System Bus and initiator permission groups. On reset or NMI, the CPU is assigned to permission group 0 (full access).

Note: Refer to the **"CPU"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for information on target region mapping and initiator permission groups for a specific device.

Initiator Initialization

Initializing the initiators for the System Bus is a two-step process:

- Set CPU and DMA priorities (if needed)
- Assign permission groups to each initiator

Priority assignments for CPU and DMA

By default, the System Bus uses a least-recently-serviced (LRS) priority arbitration scheme for most initiators, including the CPU and DMA. For applications that require low latency, the user may set the CPU and/or DMA arbitration to fixed high priority. For the CPU, fixed high priority means that the CPU gets arbitration preference while servicing an interrupt over all initiators using LRS arbitration.

Priority is set using the PLIB_SB_CPUPrioritySet and PLIB_SB_DMAPrioritySet library functions.

Initiator permission groups

At reset, all initiators default to permission group 0. Each target region can be individually programmed to allow read or write access to any or all permission groups. By assigning permission groups to initiators and restricting target region read/write access based on permission group, it is possible to restrict initiator access to target regions during normal operations.

Permission groups are assigned using the library function PLIB_SB_InitPermGrpSet.

Target Initialization

Each System Bus target resides in a physical address space that contains one or more regions. Regions are used to provide separate access permissions for different areas within a target address space. Region 0 for any target is defined as the entire address space for that target. At reset, region 0 for all targets is accessible by any initiator with group 0 permissions.

Target regions are defined by base (physical) address and size. Read and write permissions are programmed individually for each region. The number of regions for each target varies, depending on the target. Some regions have fixed addresses, sizes and/or permissions. For example, region 0 for all targets encompasses the entire address space for the target, so the address and size are preprogrammed. Similarly, write permission for PFM is disabled for all initiators except the Flash controller.

Target region initialization is a four step process:

1. Set the base address of a region using PLIB_SB_PGRegionAddrSet. This function takes an enumerated value for the target region. Only valid target regions should be enumerated for any specific device.
2. Set the size of the region using PLIB_SB_PGRegionSizeSet.
3. Set the read permissions using PLIB_SB_PGRegionReadPermSet.
4. Set the write permissions using PLIB_SB_PGRegionWritePermSet.

Note: Please refer to the **"Memory Organization"** chapter in the specific device data sheet or the family reference manual section specified that chapter for information on target region mapping and initiator permission groups.

PGV Error Handling

A permission group violation (PGV) occurs when an initiator attempts to access a target without the proper permissions. Any permission group violation will trigger an interrupt to the CPU. If enabled, the System Bus will report a PGV to the error log registers in the System Bus. The PGV interrupt handler can then use this library to retrieve information about the violation and act accordingly.

By default, error reporting is disabled in the System Bus and must be enabled during the start-up or boot code. Errors are divided into two categories: primary and secondary. Primary errors are errors arising from secure accesses, such as trying to access a target region without the proper permissions. Secondary errors are errors arising from non-secure accesses, such as trying to access a memory location outside of a target region.

PIC32 devices support logging of primary errors only. Primary error logging is enabled using the library function PLIB_SB_PGVErrorReportPrimaryEnable. Primary error reporting can be disabled using the library function PLIB_SB_PGVErrorReportPrimaryDisable. Error logging must be enabled or disabled separately for each target.

Once a permission group violation error is logged, the user can use this library to retrieve information from the System Bus error log registers. Each target has its own set of error log registers. The error information available and the library API used to retrieve it are shown in the following table.

Error Log Data	System Bus Peripheral Library Function
Has the specified target reported an error?	PLIB_SB_PGVErrorStatus
Has the specified target reported more than one error since last cleared?	PLIB_SB_PGVErrorMulti
Type of error.	PLIB_SB_PGVErrorCode
Initiator ID.	PLIB_SB_PGVErrorInitiatorID
OCP command code of the transaction.	PLIB_SB_PGVErrorCommandCode
The region number of the specified target accessed during the violation.	PLIB_SB_PGVErrorRegion
Permission group of initiator that caused the violation.	PLIB_SB_PGVErrorPermissionGroup

Once an error is logged, it needs to be cleared by software before any data for a subsequent violation can be logged (for the same target). This is typically done in the exception handler that reads the data. Which function is used depends on whether a single or multiple error violation occurred. Single violations are cleared with PLIB_SB_PGVErrorClearSingle, and multiple violations are cleared with PLIB_SB_PGVErrorClearMulti.

Configuring the Library

The library is configured for the supported System Bus module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Target Initialization Functions

	Name	Description
	PLIB_SB_PGRegionAddrGet	Returns the base address for a permission group region within a target's physical address space.
	PLIB_SB_PGRegionAddrSet	Sets the base address for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionReadPermClear	Clears the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionReadPermSet	Sets the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionSizeGet	Returns the size for a permission group region within a target's physical address space.
	PLIB_SB_PGRegionSizeSet	Sets the size for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionWritePermClear	Clears the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionWritePermSet	Sets the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.

b) PGV Error Functions

	Name	Description
	PLIB_SB_PGVErrorClearMulti	Clears multiple permission group errors for the specified target.

	PLIB_SB_PGVErrorClearSingle	Clears a single permission group error for the specified target.
	PLIB_SB_PGVErrorCode	Returns a value corresponding to the type of error logged for the specified target.
	PLIB_SB_PGVErrorCommandCode	Returns the OCP command code of the transaction that caused the protection violation for the specified target.
	PLIB_SB_PGVErrorInitiatorID	Returns the ID of the Initiator that caused the protection violation
	PLIB_SB_PGVErrorLogClearMulti	Clears a multiple protection group violations error from the specified target error log register.
	PLIB_SB_PGVErrorLogClearSingle	Clears a single protection group violation error from the specified target error log register.
	PLIB_SB_PGVErrorMulti	Indicates if more than one permission group violation has occurred since last cleared.
	PLIB_SB_PGVErrorPermissionGroup	Returns the permission group of the protection region in a target address space that caused the protection violation for the specified target.
	PLIB_SB_PGVErrorRegion	Returns the number of the protection region in the specified target address space that caused the protection violation.
	PLIB_SB_PGVErrorReportPrimaryDisable	Disables primary permission group error reporting for the specified target to the SB flag register.
	PLIB_SB_PGVErrorReportPrimaryEnable	Enables primary permission group error reporting for the specified target to the SB flag register.
	PLIB_SB_PGVErrorStatus	Identifies whether a permission group violation has been reported for the specified target.
	PLIB_SB_PGVErrGroup0Status	Identifies whether a permission group violation has been reported for the Target Group 0.
	PLIB_SB_PGVErrGroup1Status	Identifies whether a permission group violation has been reported for the Target Group 1.
	PLIB_SB_PGVErrGroup2Status	Identifies whether a permission group violation has been reported for the Target Group 2.
	PLIB_SB_PGVErrGroup3Status	Identifies whether a permission group violation has been reported for the Target Group 3.
	PLIB_SB_PGVErrGroupStatus	Identifies whether a permission group violation has been reported for the specified target group.

c) Other Functions

	Name	Description
	PLIB_SB_CPUPrioritySet	Sets the CPU arbitration policy to SRAM when servicing an interrupt
	PLIB_SB_DMAPrioritySet	Sets the DMA arbitration policy
	PLIB_SB_InitPermGrpSet	Sets the read/write permission group(s) for an initiator. The region must also allow read/write access for the permission group for a read/write to occur.

d) Feature Existence Functions

	Name	Description
	PLIB_SB_ExistsCPUPriority	Identifies whether the CPUPriority feature exists on the System Bus module.
	PLIB_SB_ExistsDMAPriority	Identifies whether the DMAPriority feature exists on the System Bus module.
	PLIB_SB_ExistsInitPermGrp	Identifies whether the InitPermGrp feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegAddr	Identifies whether the PGRegAddr feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegRdPerm	Identifies whether the PGRegRdPerm feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegSize	Identifies whether the PGRegSize feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegWrPerm	Identifies whether the PGRegWrPerm feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrClear	Identifies whether the PGVErrClear feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrClrMulti	Identifies whether the PGVErrClrMulti feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrClrSingle	Identifies whether the PGVErrClrSingle feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrCmdCode	Identifies whether the PGVErrCmdCode feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrInitID	Identifies whether the PGVErrInitID feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrPG	Identifies whether the PGVErrPG feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrRegion	Identifies whether the PGVErrRegion feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrRptPri	Identifies whether the PGVErrRptPri feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrStatus	Identifies whether the PGVErrStatus feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrGroup0Status	Identifies whether the PGVErrGroup0Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroup1Status	Identifies whether the PGVErrGroup1Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroup2Status	Identifies whether the PGVErrGroup2Status feature exists on the SB module

	PLIB_SB_ExistsPGVErrGroup3Status	Identifies whether the PGVErrGroup3Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroupStatus	Identifies whether the PGVErrGroupStatus feature exists on the SB module

e) Data Types and Constants

	Name	Description
	PLIB_SB_ARB_POLICY	This enumeration lists the possible arbitration policies that can be assigned to the CPU and DMA for SRAM access.
	PLIB_SB_ERROR	Lists the possible System Bus Transaction Error Codes.
	PLIB_SB_INIT_ID	Lists the possible System Bus Initiator IDs.
	PLIB_SB_INIT_PG	Lists the possible Initiator permission groups
	PLIB_SB_OCP_CMD_CODE	Lists the possible OCP Command codes.
	PLIB_SB_PG_INITIATOR	Lists the possible permission group Initiators.
	PLIB_SB_REGION_PG	This enumeration lists the possible permission groups assigned to a region for read and/or write access.
	PLIB_SB_TGT_ID	Lists the possible System Bus Target IDs.
	PLIB_SB_TGT_REGION	Lists the programmable target regions.
	SB_MODULE_ID	Lists the possible Module IDs for the System Bus.

Description

This section describes the Application Programming Interface (API) functions of the System Bus Peripheral Library.

Refer to each section for a detailed description.

a) Target Initialization Functions

PLIB_SB_PGRegionAddrGet Function

Returns the base address for a permission group region within a target's physical address space.

File

plib_sb.h

C

```
uint32_t PLIB_SB_PGRegionAddrGet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region);
```

Returns

uint32_t - The base address of the region.

Description

This function returns the base address for a permission group region within a target's physical address space.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
uint32_t T1R6BaseAddress;
T1R6BaseAddress = PLIB_SB_PGRegionAddrGet(SB_ID_1, PLIB_T1_REGION_6);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region for which the address is returned.

Function

uint32_t PLIB_SB_PGRegionAddrGet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region)

PLIB_SB_PGRegionAddrSet Function

Sets the base address for a permission group region within a target's physical address space. Not all regions are programmable.

File

plib_sb.h

C

```
void PLIB_SB_PGRegionAddrSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, uint32_t phys_addr);
```

Returns

None.

Description

This function sets the base address for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Region 0 of all targets have a defined start address which may not be changed. Some regions (such as those containing peripheral SFRs) are fixed and may not be changed. Please refer to the specific device data sheet for details.

Preconditions

None.

Example

```
// Set up two regions in PFM with full read and write permission
#define REGION_5_BASE_ADDR  0x1D000000
#define REGION_6_BASE_ADDR  (REGION5_BASE_ADDR + (32*1024))
#define REGION_5_SIZE       0x06 // 32KB
#define REGION_6_SIZE       0x05 // 16KB
#define FULL_PERM           (REGION_PG_0 | REGION_PG_1 | REGION_PG_2 | REGION_PG_3)

PLIB_SB_PGRegionAddrSet(SB_ID_1, PLIB_SB_T1_REGION_5, REGION_5_BASE_ADDR);
PLIB_SB_PGRegionAddrSet(SB_ID_1, PLIB_SB_T1_REGION_6, REGION_6_BASE_ADDR);

PLIB_SB_PGRegionSizeSet(SB_ID_1, PLIB_SB_T1_REGION_5, REGION_5_SIZE);
PLIB_SB_PGRegionSizeSet(SB_ID_1, PLIB_SB_T1_REGION_6, REGION_6_SIZE);

PLIB_SB_PGRegionReadPermSet(SB_ID_1, PLIB_T1_REGION_5, FULL_PERM);
PLIB_SB_PGRegionReadPermSet(SB_ID_1, PLIB_T1_REGION_6, FULL_PERM);

PLIB_SB_PGRegionWritePermSet(SB_ID_1, PLIB_T1_REGION_5, FULL_PERM);
PLIB_SB_PGRegionWritePermSet(SB_ID_1, PLIB_T1_REGION_6, FULL_PERM);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region for which the base address is set.
phys_addr	The base address of the region. Must be aligned to the intended size of the region.

Function

void PLIB_SB_PGRegionAddrSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region,

uint32_t phys_addr)

PLIB_SB_PGRegionReadPermClear Function

Clears the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.

File

plib_sb.h

C

```
void PLIB_SB_PGRegionReadPermClear(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, PLIB_SB_REGION_PG
readPerm);
```

Returns

None.

Description

This function clears the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Default reset value for all programmable regions is read permission for all groups.

Preconditions

None.

Example

See the code example for PLIB_SB_PGRegionAddrSet.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region number within the target's address space to be given permissions. Not all regions are programmable.
readPerm	Bitwise OR of the groups given read permission for the region.

Function

void PLIB_SB_PGRegionReadPermClear(SB_MODULE_ID index, PLIB_SB_TGT_REGION region,
PLIB_SB_REGION_PG readPerm)

PLIB_SB_PGRegionReadPermSet Function

Sets the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.

File

plib_sb.h

C

```
void PLIB_SB_PGRegionReadPermSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, PLIB_SB_REGION_PG readPerm);
```

Returns

None.

Description

This function sets the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Default reset value for all programmable regions is read permission for all groups.

Preconditions

None.

Example

See the code example for PLIB_SB_PGRegionAddrSet.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region number within the target's address space to be given permissions. Not all regions are programmable.
readPerm	Bitwise OR of the groups given read permission for the region.

Function

void PLIB_SB_PGRegionReadPermSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region,
PLIB_SB_REGION_PG readPerm)

PLIB_SB_PGRegionSizeGet Function

Returns the size for a permission group region within a target's physical address space.

File

plib_sb.h

C

```
uint32_t PLIB_SB_PGRegionSizeGet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region);
```

Returns

The size of the region.

Description

This function returns the size for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Region 0 of all targets encompasses the entire address range of the target, and may not be changed. Some regions (such as those containing peripheral SFRs) may not be changed. Please refer to the specific device data sheet for details.

Preconditions

None.

Example

```
uint32_t T1R6Size;
T1R6Size = PLIB_SB_PGRegionSizeGet(SB_ID_1, PLIB_T1_REGION_6);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region for which the size is to be set. Not all regions are programmable.

Function

uint32_t PLIB_SB_PGRegionSizeGet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region)

PLIB_SB_PGRegionSizeSet Function

Sets the size for a permission group region within a target's physical address space. Not all regions are programmable.

File

plib_sb.h

C

```
void PLIB_SB_PGRegionSizeSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, uint32_t size);
```

Returns

None.

Description

This function sets the size for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Region 0 of all targets encompasses the entire address range of the target, and may not be changed. Some regions (such as those containing peripheral SFRs) may not be changed. Please refer to the specific device data sheet for details.

Preconditions

None.

Example

See the code example for PLIB_SB_PGRegionAddrSet.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region for which the size is to be set. Not all regions are programmable.
size	The actual size of the region being programmed is calculated as
follows	2**(size-1)*1024 bytes

Function

void PLIB_SB_PGRegionSizeSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, uint32_t size)

PLIB_SB_PGRegionWritePermClear Function

Clears the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.

File

plib_sb.h

C

```
void PLIB_SB_PGRegionWritePermClear(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, PLIB_SB_REGION_PG writePerm);
```

Returns

None.

Description

This function clears the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Default reset value for all programmable regions is write permission for all groups. Some regions may not be programmable by all initiators (flash, for example). Please refer to the specific device data sheet for details.

Preconditions

None.

Example

See the code example for PLIB_SB_PGRegionAddrSet.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region number within the target's address space to be given permissions. Not all regions are programmable.
writePerm	Bitwise OR of the groups given read permission for the region.

Function

void PLIB_SB_PGRegionWritePermClear(SB_MODULE_ID index, PLIB_SB_TGT_REGION region,
PLIB_SB_REGION_PG writePerm)

PLIB_SB_PGRegionWritePermSet Function

Sets the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.

File

plib_sb.h

C

```
void PLIB_SB_PGRegionWritePermSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region, PLIB_SB_REGION_PG
writePerm);
```

Returns

None.

Description

This function sets the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Default reset value for all programmable regions is write permission for all groups. Some regions may not be programmable by all initiators (flash, for example). Please refer to the specific device data sheet for details.

Preconditions

None.

Example

See the code example for PLIB_SB_PGRegionAddrSet.

Parameters

Parameters	Description
index	Identifier for the device instance.
region	The region number within the target's address space to be given permissions. Not all regions are programmable.
writePerm	Bitwise OR of the groups given read permission for the region.

Function

void PLIB_SB_PGRegionWritePermSet(SB_MODULE_ID index, PLIB_SB_TGT_REGION region,
PLIB_SB_REGION_PG writePerm)

b) PGV Error Functions

PLIB_SB_PGVErrorClearMulti Function

Clears multiple permission group errors for the specified target.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrorClearMulti(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

mult - Returns the value of the CLEAR bit in the SBTxECLRM register for the specified target. The act of reading this bit clears the error.

Description

This function clears multiple permission group errors for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool mult;
mult = PLIB_SB_PGVErrorClearMulti(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

bool PLIB_SB_PGVErrorClearMulti(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorClearSingle Function

Clears a single permission group error for the specified target.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrorClearSingle(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

sing - Returns the value of the CLEAR bit in the SBTxECLRM register for the specified target. The act of reading this bit clears the error.

Description

This function clears a single permission group error for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sing;
sing = PLIB_SB_PGVErrorClearSingle(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

bool PLIB_SB_PGVErrorClearSingle(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorCode Function

Returns a value corresponding to the type of error logged for the specified target.

File

plib_sb.h

C

```
PLIB_SB_ERROR PLIB_SB_PGVErrorCode(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

PLIB_SB_ERROR enumeration representing the type of SB error logged states.

Description

This function returns a value corresponding to the type of error logged for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
PLIB_SB_ERROR error;
error = PLIB_SB_ERROR PLIB_SB_PGVErrorCode(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target to be read.

Function

PLIB_SB_ERROR PLIB_SB_PGVErrorCode(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorCommandCode Function

Returns the OCP command code of the transaction that caused the protection violation for the specified target.

File

plib_sb.h

C

```
PLIB_SB_OCP_CMD_CODE PLIB_SB_PGVErrorCommandCode(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

OCP command code

Description

This function returns the OCP command code of the transaction that caused the protection violation for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
PLIB_SB_OCP_CMD_CODE commandCode;
commandCode = PLIB_SB_PGVErrorCommandCode(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

PLIB_SB_OCP_CMD_CODE PLIB_SB_PGVErrorCommandCode(SB_MODULE_ID index , PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorInitiatorID Function

Returns the ID of the Initiator that caused the protection violation

File

plib_sb.h

C

```
PLIB_SB_INIT_ID PLIB_SB_PGVErrorInitiatorID(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

An enumerated value representing the ID of the initiator that caused the protection violation.

Description

This function returns the ID of the Initiator that caused the protection violation.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
PLIB_SB_INIT_ID id;
id = PLIB_SB_PGVErrorInitiatorID(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

PLIB_SB_INIT_ID PLIB_SB_PGVErrorInitiatorID(SB_MODULE_ID index , PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorLogClearMulti Function

Clears a multiple protection group violations error from the specified target error log register.

File

plib_sb.h

C

```
void PLIB_SB_PGVErrorLogClearMulti(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

None.

Description

This function clears a multiple protection group violations error from the specified target error log register. Multiple errors are cleared by writing a '1' to both the MULTI and CODE fields of the SBTxELOG1 register for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
PLIB_SB_PGVErrorLogClearMulti(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target to be cleared.

Function

void PLIB_SB_PGVErrorLogClearMulti(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorLogClearSingle Function

Clears a single protection group violation error from the specified target error log register.

File

plib_sb.h

C

```
void PLIB_SB_PGVErrorLogClearSingle(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

None.

Description

This function clears a single protection group violation error from the specified target error log register. Single errors are cleared by writing a '1' to the CODE field of the SBTxELOG1 register for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
PLIB_SB_PGVErrorLogClearSingle(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target to be cleared.

Function

void PLIB_SB_PGVErrorLogClearSingle(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorMulti Function

Indicates if more than one permission group violation has occurred since last cleared.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrorMulti(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

- true - Multiple permission group violations.
- false - Single or no permission group violations.

Description

This function indicates if more than one permission group violation has occurred since last cleared.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool multiPGV;
multiPGV = PLIB_SB_PGVErrorMulti(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target to be read.

Function

bool PLIB_SB_PGVErrorMulti(SB_MODULE_ID index , PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorPermissionGroup Function

Returns the permission group of the protection region in a target address space that caused the protection violation for the specified target.

File

plib_sb.h

C

```
int PLIB_SB_PGVErrorPermissionGroup(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

None.

Description

This function returns the permission group of the protection region in a target address space that caused the protection violation for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
int pg;
pg = PLIB_SB_PGVErrorPermissionGroup(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

int PLIB_SB_PGVErrorPermissionGroup(SB_MODULE_ID index , PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorRegion Function

Returns the number of the protection region in the specified target address space that caused the protection violation.

File

plib_sb.h

C

```
int PLIB_SB_PGVErrorRegion(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

Region number in target address space or -1 on unrecognized target.

Description

This function returns the number of the protection region in the specified target address space that caused the protection violation. Note that if there are no other region matches, region 0 (the default region that spans the entire target address space) will always match, and this function will return 0.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
int region;
region = PLIB_SB_PGVErrorRegion(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

int PLIB_SB_PGVErrorRegion(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorReportPrimaryDisable Function

Disables primary permission group error reporting for the specified target to the SB flag register.

File

plib_sb.h

C

```
void PLIB_SB_PGVErrorReportPrimaryDisable(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

None.

Description

This function disables primary permission group error reporting for the specified target to the SB flag register.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Reporting of primary errors is disabled at reset.

Preconditions

None.

Example

```
PLIB_SB_PGVErrorReportPrimaryDisable(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.

target	The target.

Function

void PLIB_SB_PGVErrorReportPrimaryDisable(SB_MODULE_ID index, PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorReportPrimaryEnable Function

Enables primary permission group error reporting for the specified target to the SB flag register.

File

plib_sb.h

C

```
void PLIB_SB_PGVErrorReportPrimaryEnable(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

None.

Description

This function enables primary permission group error reporting for the specified target to the SB flag register.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Reporting of primary errors is disabled at reset.

Preconditions

None.

Example

```
PLIB_SB_PGVErrorReportPrimaryEnable(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	The target.

Function

void PLIB_SB_PGVErrorReportPrimaryEnable(SB_MODULE_ID index , PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrorStatus Function

Identifies whether a permission group violation has been reported for the specified target.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrorStatus(SB_MODULE_ID index, PLIB_SB_TGT_ID target);
```

Returns

- true - Target is reporting a permission group violation.
- false - Target is not reporting a permission group violation.

Description

This function identifies whether a permission group violation has been reported for the specified target.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sbPgv;
sbPgv = PLIB_SB_PGVErrorStatus(SB_ID_1, PLIB_SB_TGT_ID_T0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
target	Target to be checked.

Function

bool PLIB_SB_PGVErrorStatus (SB_MODULE_ID index , PLIB_SB_TGT_ID target)

PLIB_SB_PGVErrGroup0Status Function

Identifies whether a permission group violation has been reported for the Target Group 0.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrGroup0Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP0_TGT targetId);
```

Returns

- true - Target is reporting a permission group violation.
- false - Target is not reporting a permission group violation.

Description

This function identifies whether a permission group violation has been reported for Target Group 0.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sbPgv;
sbPgv = PLIB_SB_PGVErrGroup0Status(SB_ID_0, PLIB_SB_PGV_GROUP0_T1_PGV);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
targetId	Target to be checked.

Function

bool PLIB_SB_PGVErrGroup0Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP0_TGT targetId);

PLIB_SB_PGVErrGroup1Status Function

Identifies whether a permission group violation has been reported for the Target Group 1.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrGroup1Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP1_TGT targetId);
```

Returns

- true - Target is reporting a permission group violation.
- false - Target is not reporting a permission group violation.

Description

This function identifies whether a permission group violation has been reported for Target Group 1.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sbPgv;
sbPgv = PLIB_SB_PGVErrGroup1Status(SB_ID_0, PLIB_SB_PGV_GROUP1_T11_PGV);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
targetId	Target to be checked.

Function

bool PLIB_SB_PGVErrGroup1Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP1_TGT targetId);

PLIB_SB_PGVErrGroup2Status Function

Identifies whether a permission group violation has been reported for the Target Group 2.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrGroup2Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP2_TGT targetId);
```

Returns

- true - Target is reporting a permission group violation.
- false - Target is not reporting a permission group violation.

Description

This function identifies whether a permission group violation has been reported for Target Group 2.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sbPgv;
sbPgv = PLIB_SB_PGVErrGroup2Status(SB_ID_0, PLIB_SB_PGV_GROUP2_T14_PGV);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
targetId	Target to be checked.

Function

bool PLIB_SB_PGVErrGroup2Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP2_TGT targetId);

PLIB_SB_PGVErrGroup3Status Function

Identifies whether a permission group violation has been reported for the Target Group 3.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrGroup3Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP3_TGT targetId);
```

Returns

- true - Target is reporting a permission group violation.
- false - Target is not reporting a permission group violation.

Description

This function identifies whether a permission group violation has been reported for Target Group 3.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sbPgv;
sbPgv = PLIB_SB_PGVErrGroup3Status(SB_ID_0, PLIB_SB_PGV_GROUP3_T16_PGV);
```

Parameters

Parameters	Description
index	Identifier for the device instance.
targetId	Target to be checked.

Function

bool PLIB_SB_PGVErrGroup3Status(SB_MODULE_ID index, PLIB_SB_PGV_GROUP3_TGT targetId);

PLIB_SB_PGVErrGroupStatus Function

Identifies whether a permission group violation has been reported for the specified target group.

File

plib_sb.h

C

```
bool PLIB_SB_PGVErrGroupStatus(SB_MODULE_ID index, PLIB_SB_PGV_GROUP_ID groupId);
```

Returns

- true - Target group is reporting a permission group violation.
- false - Target group is not reporting a permission group violation.

Description

This function identifies whether a permission group violation has been reported for the specified target group.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
bool sbPgv;
sbPgv = PLIB_SB_PGVErrGroupStatus(SB_ID_0, PLIB_SB_PGV_GROUP0);
```

Parameters

Parameters	Description
index	Identifier for the device instance.

groupId	Target group to be checked.

Function

bool PLIB_SB_PGVErrGroupStatus(SB_MODULE_ID index, PLIB_SB_PGV_GROUP_ID groupId);

c) Other Functions

PLIB_SB_CPUPrioritySet Function

Sets the CPU arbitration policy to SRAM when servicing an interrupt

File

plib_sb.h

C

```
void PLIB_SB_CPUPrioritySet(SB_MODULE_ID index, PLIB_SB_ARB_POLICY priority);
```

Returns

None.

Description

This function sets the CPU arbitration policy to SRAM when servicing an interrupt.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

This function writes to the soft configuration register CFGCON, which is not part of the SB. This should be done by the boot code prior to programming the SB. Note that the system must be unlocked before the priority can be set.

Default at reset is PRIORITY_LRS.

Preconditions

System must be unlocked before the priority can be set.

Example

```
PLIB_SB_PriorityUnlock();
PLIB_SB_CPUPrioritySet(SB_ID_1, PRIORITY_HI);
```

Parameters

Parameters	Description
priority	Use either high priority or least-recently-serviced algorithm.

Function

void PLIB_SB_CPUPrioritySet(SB_MODULE_ID index, PLIB_SB_ARB_POLICY priority)

PLIB_SB_DMAPrioritySet Function

Sets the DMA arbitration policy

File

plib_sb.h

C

```
void PLIB_SB_DMAPrioritySet(SB_MODULE_ID index, PLIB_SB_ARB_POLICY priority);
```

Returns

None.

Description

This function sets the DMA arbitration.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

This function writes to the soft configuration register CFGCON, which is not part of the SB. This should be done by the boot code prior to programming the SB. Note that the system must be unlocked before the priority can be set.

Default at reset is PRIORITY_LRS.

Preconditions

System must be unlocked before the priority can be set.

Example

```
PLIB_SB_PriorityUnlock();
void PLIB_SB_DMAPrioritySet(SB_ID_1, PRIORITY_HI);
```

Parameters

Parameters	Description
priority	Use either high priority or least-recently-serviced algorithm.

Function

void PLIB_SB_DMAPrioritySet(SB_MODULE_ID index, PLIB_SB_ARB_POLICY priority)

PLIB_SB_InitPermGrpSet Function

Sets the read/write permission group(s) for an initiator. The region must also allow read/write access for the permission group for a read/write to occur.

File

plib_sb.h

C

```
void PLIB_SB_InitPermGrpSet(SB_MODULE_ID index, PLIB_SB_PG_INITIATOR initiator, PLIB_SB_INIT_PG pg);
```

Returns

None.

Description

This function sets the read/write permission group(s) for an initiator. The region must also allow read/write access for the permission group for a read/write to occur.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet for availability.

This function writes to the soft configuration register CFGPG, which is not part of the SB. Permission groups should be assigned by the boot code prior to programming the SB.

Default permission group at reset for all initiators is 0.

After an NMI, the CPU permission group reverts to 0. All other initiator permission groups remain unchanged.

The effective CPU permission group value in debug mode is controlled by boot configuration value DBGPER[2:0]. If DBGPER denies access to the group CPU1PG selects, the effective value selects group 3.

Preconditions

None.

Example

```
PLIB_SB_InitPermGrpSet(SB_ID_1, PLIB_SB_PG_INITIATOR_CPU, PLIB_SB_INIT_PG_1);
```

Parameters

Parameters	Description
initiator	The initiator for which permission groups are assigned.
pg	The permission group(s) to which the initiator is assigned.

Function

void PLIB_SB_InitPermGrpSet(SB_MODULE_ID index, PLIB_SB_PG_INITIATOR initiator, PLIB_SB_INIT_PG pg)

d) Feature Existence Functions

PLIB_SB_ExistsCPUPriority Function

Identifies whether the CPUPriority feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsCPUPriority(SB_MODULE_ID index);
```

Returns

- true - The CPUPriority feature is supported on the device
- false - The CPUPriority feature is not supported on the device

Description

This function identifies whether the CPUPriority feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_CPUPrioritySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsCPUPriority(SB_MODULE_ID index)

PLIB_SB_ExistsDMAPriority Function

Identifies whether the DMAPriority feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsDMAPriority(SB_MODULE_ID index);
```

Returns

- true - The DMAPriority feature is supported on the device
- false - The DMAPriority feature is not supported on the device

Description

This function identifies whether the DMAPriority feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_DMAPrioritySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsDMAPriority(SB_MODULE_ID index)

PLIB_SB_ExistsInitPermGrp Function

Identifies whether the InitPermGrp feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsInitPermGrp(SB_MODULE_ID index);
```

Returns

- true - The InitPermGrp feature is supported on the device
- false - The InitPermGrp feature is not supported on the device

Description

This function identifies whether the InitPermGrp feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_InitPermGrpSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsInitPermGrp(SB_MODULE_ID index)

PLIB_SB_ExistsPGRegAddr Function

Identifies whether the PGRegAddr feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGRegAddr(SB_MODULE_ID index);
```

Returns

- true - The PGRegAddr feature is supported on the device
- false - The PGRegAddr feature is not supported on the device

Description

This function identifies whether the PGRegAddr feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGRegionAddrSet
- PLIB_SB_PGRegionAddrGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGRegAddr(SB_MODULE_ID index)

PLIB_SB_ExistsPGRegRdPerm Function

Identifies whether the PGRegRdPerm feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGRegRdPerm(SB_MODULE_ID index);
```

Returns

- true - The PGRegRdPerm feature is supported on the device
- false - The PGRegRdPerm feature is not supported on the device

Description

This function identifies whether the PGRegRdPerm feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGRegionReadPermSet
- PLIB_SB_PGRegionReadPermClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGRegRdPerm(SB_MODULE_ID index)

PLIB_SB_ExistsPGRegSize Function

Identifies whether the PGRegSize feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGRegSize(SB_MODULE_ID index);
```

Returns

- true - The PGRegSize feature is supported on the device
- false - The PGRegSize feature is not supported on the device

Description

This function identifies whether the PGRegSize feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGRegionSizeSet
- PLIB_SB_PGRegionSizeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGRegSize(SB_MODULE_ID index)

PLIB_SB_ExistsPGRegWrPerm Function

Identifies whether the PGRegWrPerm feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGRegWrPerm(SB_MODULE_ID index);
```

Returns

- true - The PGRegWrPerm feature is supported on the device
- false - The PGRegWrPerm feature is not supported on the device

Description

This function identifies whether the PGRegWrPerm feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGRegionWritePermSet
- PLIB_SB_PGRegionWritePermClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGRegWrPerm(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrClear Function

Identifies whether the PGVErrClear feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrClear(SB_MODULE_ID index);
```

Returns

- true - The PGVErrClear feature is supported on the device
- false - The PGVErrClear feature is not supported on the device

Description

This function identifies whether the PGVErrClear feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrorMulti
- PLIB_SB_PGVErrorCode
- PLIB_SB_PGVErrorLogClearSingle
- PLIB_SB_PGVErrorLogClearMulti

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrClear(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrClrMulti Function

Identifies whether the PGVErrClrMulti feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrClrMulti(SB_MODULE_ID index);
```

Returns

- true - The PGVErrClrMulti feature is supported on the device
- false - The PGVErrClrMulti feature is not supported on the device

Description

This function identifies whether the PGVErrClrMulti feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_PGVErrorClearMulti

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrClrMulti(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrClrSingle Function

Identifies whether the PGVErrClrSingle feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrClrSingle(SB_MODULE_ID index);
```

Returns

- true - The PGVErrClrSingle feature is supported on the device
- false - The PGVErrClrSingle feature is not supported on the device

Description

This function identifies whether the PGVErrClrSingle feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_PGVErrorClearSingle

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrClrSingle(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrCmdCode Function

Identifies whether the PGVErrCmdCode feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrCmdCode(SB_MODULE_ID index);
```

Returns

- true - The PGVErrCmdCode feature is supported on the device
- false - The PGVErrCmdCode feature is not supported on the device

Description

This function identifies whether the PGVErrCmdCode feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_PGVErrorCommandCode

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrCmdCode(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrInitID Function

Identifies whether the PGVErrInitID feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrInitID(SB_MODULE_ID index);
```

Returns

- true - The PGVErrInitID feature is supported on the device
- false - The PGVErrInitID feature is not supported on the device

Description

This function identifies whether the PGVErrInitID feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_PGVErrorInitiatorID

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrInitID(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrPG Function

Identifies whether the PGVErrPG feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrPG(SB_MODULE_ID index);
```

Returns

- true - The PGVErrPG feature is supported on the device
- false - The PGVErrPG feature is not supported on the device

Description

This function identifies whether the PGVErrPG feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_PGVErrorPermissionGroup

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrPG(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrRegion Function

Identifies whether the PGVErrRegion feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrRegion(SB_MODULE_ID index);
```

Returns

- true - The PGVErrRegion feature is supported on the device
- false - The PGVErrRegion feature is not supported on the device

Description

This function identifies whether the PGVErrRegion feature is available on the System Bus module. When this function returns true, this function is supported on the device:

- PLIB_SB_PGVErrorRegion

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrRegion(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrRptPri Function

Identifies whether the PGVErrRptPri feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrRptPri(SB_MODULE_ID index);
```

Returns

- true - The PGVErrRptPri feature is supported on the device
- false - The PGVErrRptPri feature is not supported on the device

Description

This function identifies whether the PGVErrRptPri feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrorReportPrimaryEnable
- PLIB_SB_PGVErrorReportPrimaryDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrRptPri(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrStatus Function

Identifies whether the PGVErrStatus feature exists on the System Bus module.

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrStatus(SB_MODULE_ID index);
```

Returns

- true - The PGVErrStatus feature is supported on the device
- false - The PGVErrStatus feature is not supported on the device

Description

This function identifies whether the PGVErrStatus feature is available on the System Bus module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrorStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrStatus(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrGroup0Status Function

Identifies whether the PGVErrGroup0Status feature exists on the SB module

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrGroup0Status(SB_MODULE_ID index);
```

Returns

- true - The PGVErrGroup0Status feature is supported on the device
- false - The PGVErrGroup0Status feature is not supported on the device

Description

This function identifies whether the PGVErrGroup0Status feature is available on the SB module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrGroup0Status

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrGroup0Status(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrGroup1Status Function

Identifies whether the PGVErrGroup1Status feature exists on the SB module

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrGroup1Status(SB_MODULE_ID index);
```

Returns

- true - The PGVErrGroup1Status feature is supported on the device
- false - The PGVErrGroup1Status feature is not supported on the device

Description

This function identifies whether the PGVErrGroup1Status feature is available on the SB module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrGroup1Status

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrGroup1Status(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrGroup2Status Function

Identifies whether the PGVErrGroup2Status feature exists on the SB module

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrGroup2Status(SB_MODULE_ID index);
```

Returns

- true - The PGVErrGroup2Status feature is supported on the device
- false - The PGVErrGroup2Status feature is not supported on the device

Description

This function identifies whether the PGVErrGroup2Status feature is available on the SB module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrGroup2Status

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrGroup2Status(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrGroup3Status Function

Identifies whether the PGVErrGroup3Status feature exists on the SB module

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrGroup3Status(SB_MODULE_ID index);
```

Returns

- true - The PGVErrGroup3Status feature is supported on the device
- false - The PGVErrGroup3Status feature is not supported on the device

Description

This function identifies whether the PGVErrGroup3Status feature is available on the SB module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrGroup3Status

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrGroup3Status(SB_MODULE_ID index)

PLIB_SB_ExistsPGVErrGroupStatus Function

Identifies whether the PGVErrGroupStatus feature exists on the SB module

File

plib_sb.h

C

```
bool PLIB_SB_ExistsPGVErrGroupStatus(SB_MODULE_ID index);
```

Returns

- true - The PGVErrGroupStatus feature is supported on the device
- false - The PGVErrGroupStatus feature is not supported on the device

Description

This function identifies whether the PGVErrGroupStatus feature is available on the SB module. When this function returns true, these functions are supported on the device:

- PLIB_SB_PGVErrGroupStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SB_ExistsPGVErrGroupStatus(SB_MODULE_ID index)

e) Data Types and Constants

PLIB_SB_ARB_POLICY Enumeration

This enumeration lists the possible arbitration policies that can be assigned to the CPU and DMA for SRAM access.

File

help_plib_sb.h

C

```
typedef enum {
  PRIORITY_LRS,
  PRIORITY_HI
} PLIB_SB_ARB_POLICY;
```

Members

Members	Description
PRIORITY_LRS	Least Recently Serviced Arbitration
PRIORITY_HI	High Priority

Description

System Bus Arbitration Policy

This enumeration lists the possible arbitration policies that can be assigned to the CPU and DMA for SRAM access.

PLIB_SB_ERROR Enumeration

Lists the possible System Bus Transaction Error Codes.

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_ERROR_NONE,
  PLIB_SB_ERROR_PGV
```

```
} PLIB_SB_ERROR;
```

Members

Members	Description
PLIB_SB_ERROR_NONE	No Error
PLIB_SB_ERROR_PGV	Permission Group Violation

Description

PG Error Code

This enumeration lists the possible transaction error codes for the System Bus.

PLIB_SB_INIT_ID Enumeration

Lists the possible System Bus Initiator IDs.

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_INIT_ID_NONE,
  PLIB_SB_INIT_ID_CPU_LRS,
  PLIB_SB_INIT_ID_CPU_HI,
  PLIB_SB_INIT_ID_DMA1_RD_LRS,
  PLIB_SB_INIT_ID_DMA1_RD_HI,
  PLIB_SB_INIT_ID_DMA1_WR_LRS,
  PLIB_SB_INIT_ID_DMA1_WR_HI,
  PLIB_SB_INIT_ID_USB1,
  PLIB_SB_INIT_ID_ETH1_RD,
  PLIB_SB_INIT_ID_ETH1_WR,
  PLIB_SB_INIT_ID_CAN1,
  PLIB_SB_INIT_ID_CAN2,
  PLIB_SB_INIT_ID_SQI1,
  PLIB_SB_INIT_ID_FLASH_CTL,
  PLIB_SB_INIT_ID_CRYPTO
} PLIB_SB_INIT_ID;
```

Description

System Bus Initiator ID

This enumeration lists the possible System Bus Initiator IDs. This ID is used for self-reporting and error logging. LRS and HI are System Bus arbitration schemes set by the soft configuration register CFGCON.

PLIB_SB_INIT_PG Enumeration

Lists the possible Initiator permission groups

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_INIT_PG_0,
  PLIB_SB_INIT_PG_1,
  PLIB_SB_INIT_PG_2,
  PLIB_SB_INIT_PG_3
} PLIB_SB_INIT_PG;
```

Members

Members	Description
PLIB_SB_INIT_PG_0	Privilege Group 0
PLIB_SB_INIT_PG_1	Privilege Group 1
PLIB_SB_INIT_PG_2	Privilege Group 2
PLIB_SB_INIT_PG_3	Privilege Group 3

Description

System Bus Initiator Permission Groups

This enumeration lists the possible initiator permission groups for the System Bus.

Remarks

These values are used to program the CFGPG soft configuration register, which is not part of the System Bus. This should be done by the boot code to set up the desired initiator permissions prior to programming the System Bus.

PLIB_SB_OCP_CMD_CODE Enumeration

Lists the possible OCP Command codes.

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_OCP_CMD_IDLE,
  PLIB_SB_OCP_CMD_WRITE,
  PLIB_SB_OCP_CMD_READ,
  PLIB_SB_OCP_CMD_READEX,
  PLIB_SB_OCP_CMD_WRITE_NON_POST,
  PLIB_SB_OCP_CMD_BROADCAST
} PLIB_SB_OCP_CMD_CODE;
```

Description

OCP Command Codes

This enumeration lists the possible OCP command codes. An OCP command code is logged when a transaction violation occurs. The command code of the offending command can then be read.

PLIB_SB_PG_INITIATOR Enumeration

Lists the possible permission group Initiators.

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_PG_INITIATOR_CPU,
  PLIB_SB_PG_INITIATOR_DMA1,
  PLIB_SB_PG_INITIATOR_USB1,
  PLIB_SB_PG_INITIATOR_CAN1,
  PLIB_SB_PG_INITIATOR_CAN2,
  PLIB_SB_PG_INITIATOR_ETH1,
  PLIB_SB_PG_INITIATOR_SQI1,
  PLIB_SB_PG_INITIATOR_FLASH_CTL,
  PLIB_SB_PG_INITIATOR_CRYPTO
} PLIB_SB_PG_INITIATOR;
```

Description

System Bus Initiators

This enumeration lists the possible permission group Initiators.

PLIB_SB_REGION_PG Enumeration

This enumeration lists the possible permission groups assigned to a region for read and/or write access.

File

help_plib_sb.h

C

```
typedef enum {
  REGION_PG_0,
```

```
  REGION_PG_1,
  REGION_PG_2,
  REGION_PG_3
} PLIB_SB_REGION_PG;
```

Members

Members	Description
REGION_PG_0	Privilege Group 0 has read/write permission
REGION_PG_1	Privilege Group 1 has read/write permission
REGION_PG_2	Privilege Group 2 has read/write permission
REGION_PG_3	Privilege Group 3 has read/write permission

Description

System Bus Region Permission Groups

Lists the possible permission groups assigned to a region for read and/or write access.

PLIB_SB_TGT_ID Enumeration

Lists the possible System Bus Target IDs.

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_TGT_ID_T0,
  PLIB_SB_TGT_ID_T1,
  PLIB_SB_TGT_ID_T2,
  PLIB_SB_TGT_ID_T3,
  PLIB_SB_TGT_ID_T4,
  PLIB_SB_TGT_ID_T5,
  PLIB_SB_TGT_ID_T6,
  PLIB_SB_TGT_ID_T7,
  PLIB_SB_TGT_ID_T8,
  PLIB_SB_TGT_ID_T9,
  PLIB_SB_TGT_ID_T10,
  PLIB_SB_TGT_ID_T11,
  PLIB_SB_TGT_ID_T12,
  PLIB_SB_TGT_ID_T13
} PLIB_SB_TGT_ID;
```

Members

Members	Description
PLIB_SB_TGT_ID_T0	System Bus
PLIB_SB_TGT_ID_T1	Prefetch Module
PLIB_SB_TGT_ID_T2	Data RAM 1
PLIB_SB_TGT_ID_T3	Data RAM 2
PLIB_SB_TGT_ID_T4	External Bus Interface
PLIB_SB_TGT_ID_T5	Peripheral Bus 1
PLIB_SB_TGT_ID_T6	Peripheral Bus 2
PLIB_SB_TGT_ID_T7	Peripheral Bus 3
PLIB_SB_TGT_ID_T8	Peripheral Bus 4
PLIB_SB_TGT_ID_T9	Peripheral Bus 5
PLIB_SB_TGT_ID_T10	USB
PLIB_SB_TGT_ID_T11	Serial Quad Interface
PLIB_SB_TGT_ID_T12	Crypto
PLIB_SB_TGT_ID_T13	Random Number Generator

Description

System Bus Target ID

This enumeration lists the possible System Bus Target IDs.

PLIB_SB_TGT_REGION Enumeration

Lists the programmable target regions.

File

help_plib_sb.h

C

```
typedef enum {
  PLIB_SB_T0_REGION_0,
  PLIB_SB_T0_REGION_1,
  PLIB_SB_T1_REGION_0,
  PLIB_SB_T1_REGION_1,
  PLIB_SB_T1_REGION_2,
  PLIB_SB_T1_REGION_3,
  PLIB_SB_T1_REGION_4,
  PLIB_SB_T1_REGION_5,
  PLIB_SB_T1_REGION_6,
  PLIB_SB_T1_REGION_7,
  PLIB_SB_T1_REGION_8,
  PLIB_SB_T2_REGION_0,
  PLIB_SB_T2_REGION_1,
  PLIB_SB_T2_REGION_2,
  PLIB_SB_T3_REGION_0,
  PLIB_SB_T3_REGION_1,
  PLIB_SB_T3_REGION_2,
  PLIB_SB_T4_REGION_0,
  PLIB_SB_T4_REGION_1,
  PLIB_SB_T4_REGION_2,
  PLIB_SB_T5_REGION_0,
  PLIB_SB_T5_REGION_1,
  PLIB_SB_T5_REGION_2,
  PLIB_SB_T6_REGION_0,
  PLIB_SB_T6_REGION_1,
  PLIB_SB_T7_REGION_0,
  PLIB_SB_T7_REGION_1,
  PLIB_SB_T8_REGION_0,
  PLIB_SB_T8_REGION_1,
  PLIB_SB_T9_REGION_0,
  PLIB_SB_T9_REGION_1,
  PLIB_SB_T10_REGION_0,
  PLIB_SB_T11_REGION_0,
  PLIB_SB_T11_REGION_1,
  PLIB_SB_T12_REGION_0,
  PLIB_SB_T13_REGION_0
} PLIB_SB_TGT_REGION;
```

Members

Members	Description
PLIB_SB_T0_REGION_0	System Bus Region 0
PLIB_SB_T0_REGION_1	System Bus Region 1
PLIB_SB_T1_REGION_0	Prefetch Module Region 0
PLIB_SB_T1_REGION_1	Prefetch Module Region 1
PLIB_SB_T1_REGION_2	Prefetch Module Region 2
PLIB_SB_T1_REGION_3	Prefetch Module Region 3
PLIB_SB_T1_REGION_4	Prefetch Module Region 4
PLIB_SB_T1_REGION_5	Prefetch Module Region 5
PLIB_SB_T1_REGION_6	Prefetch Module Region 6
PLIB_SB_T1_REGION_7	Prefetch Module Region 7
PLIB_SB_T1_REGION_8	Prefetch Module Region 8
PLIB_SB_T2_REGION_0	Data RAM 1 Region 0
PLIB_SB_T2_REGION_1	Data RAM 1 Region 1
PLIB_SB_T2_REGION_2	Data RAM 1 Region 2
PLIB_SB_T3_REGION_0	Data RAM 2 Region 0
PLIB_SB_T3_REGION_1	Data RAM 2 Region 1

PLIB_SB_T3_REGION_2	Data RAM 2 Region 2
PLIB_SB_T4_REGION_0	External Bus Interface Region 0
PLIB_SB_T4_REGION_1	External Bus Interface Region 1
PLIB_SB_T4_REGION_2	External Bus Interface Region 2
PLIB_SB_T5_REGION_0	Peripheral Bus 1 Region 0
PLIB_SB_T5_REGION_1	Peripheral Bus 1 Region 1
PLIB_SB_T5_REGION_2	Peripheral Bus 1 Region 2
PLIB_SB_T6_REGION_0	Peripheral Bus 2 Region 0
PLIB_SB_T6_REGION_1	Peripheral Bus 2 Region 1
PLIB_SB_T7_REGION_0	Peripheral Bus 3 Region 0
PLIB_SB_T7_REGION_1	Peripheral Bus 3 Region 1
PLIB_SB_T8_REGION_0	Peripheral Bus 4 Region 0
PLIB_SB_T8_REGION_1	Peripheral Bus 4 Region 1
PLIB_SB_T9_REGION_0	Peripheral Bus 5 Region 0
PLIB_SB_T9_REGION_1	Peripheral Bus 5 Region 1
PLIB_SB_T10_REGION_0	USB Region 0
PLIB_SB_T11_REGION_0	Serial Quad Interface Region 0
PLIB_SB_T11_REGION_1	Serial Quad Interface Region 1
PLIB_SB_T12_REGION_0	Crypto Region 9
PLIB_SB_T13_REGION_0	Random Number Generator 0

Description

Regions

This enumeration lists the programmable System Bus target regions.

SB_MODULE_ID Enumeration

Lists the possible Module IDs for the System Bus.

File

help_plib_sb.h

C

```
typedef enum {
  SB_ID_1,
  SB_NUMBER_OF_MODULES
} SB_MODULE_ID;
```

Description

Module ID

This enumeration lists the possible Module IDs for the System Bus.

Remarks

Refer to the data sheet to get the correct number of modules defined for the desired device.

Files

Files

Name	Description
plib_sb.h	Defines the System Bus Peripheral Library interface
help_plib_sb.h	This file is used for documentation purposes

Description

This section lists the source and header files used by the library.

plib_sb.h

Defines the System Bus Peripheral Library interface

Functions

	Name	Description
	PLIB_SB_CPUPrioritySet	Sets the CPU arbitration policy to SRAM when servicing an interrupt
	PLIB_SB_DMAPrioritySet	Sets the DMA arbitration policy
	PLIB_SB_ExistsCPUPriority	Identifies whether the CPUPriority feature exists on the System Bus module.
	PLIB_SB_ExistsDMAPriority	Identifies whether the DMAPriority feature exists on the System Bus module.
	PLIB_SB_ExistsInitPermGrp	Identifies whether the InitPermGrp feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegAddr	Identifies whether the PGRegAddr feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegRdPerm	Identifies whether the PGRegRdPerm feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegSize	Identifies whether the PGRegSize feature exists on the System Bus module.
	PLIB_SB_ExistsPGRegWrPerm	Identifies whether the PGRegWrPerm feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrClear	Identifies whether the PGVErrClear feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrClrMulti	Identifies whether the PGVErrClrMulti feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrClrSingle	Identifies whether the PGVErrClrSingle feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrCmdCode	Identifies whether the PGVErrCmdCode feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrGroup0Status	Identifies whether the PGVErrGroup0Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroup1Status	Identifies whether the PGVErrGroup1Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroup2Status	Identifies whether the PGVErrGroup2Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroup3Status	Identifies whether the PGVErrGroup3Status feature exists on the SB module
	PLIB_SB_ExistsPGVErrGroupStatus	Identifies whether the PGVErrGroupStatus feature exists on the SB module
	PLIB_SB_ExistsPGVErrInitID	Identifies whether the PGVErrInitID feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrPG	Identifies whether the PGVErrPG feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrRegion	Identifies whether the PGVErrRegion feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrRptPri	Identifies whether the PGVErrRptPri feature exists on the System Bus module.
	PLIB_SB_ExistsPGVErrStatus	Identifies whether the PGVErrStatus feature exists on the System Bus module.
	PLIB_SB_InitPermGrpSet	Sets the read/write permission group(s) for an initiator. The region must also allow read/write access for the permission group for a read/write to occur.
	PLIB_SB_PGRegionAddrGet	Returns the base address for a permission group region within a target's physical address space.
	PLIB_SB_PGRegionAddrSet	Sets the base address for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionReadPermClear	Clears the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionReadPermSet	Sets the permission bit(s) corresponding to the requested read permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionSizeGet	Returns the size for a permission group region within a target's physical address space.
	PLIB_SB_PGRegionSizeSet	Sets the size for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionWritePermClear	Clears the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGRegionWritePermSet	Sets the permission bit(s) corresponding to the requested write permissions for a permission group region within a target's physical address space. Not all regions are programmable.
	PLIB_SB_PGVErrGroup0Status	Identifies whether a permission group violation has been reported for the Target Group 0.
	PLIB_SB_PGVErrGroup1Status	Identifies whether a permission group violation has been reported for the Target Group 1.
	PLIB_SB_PGVErrGroup2Status	Identifies whether a permission group violation has been reported for the Target Group 2.
	PLIB_SB_PGVErrGroup3Status	Identifies whether a permission group violation has been reported for the Target Group 3.
	PLIB_SB_PGVErrGroupStatus	Identifies whether a permission group violation has been reported for the specified target group.
	PLIB_SB_PGVErrorClearMulti	Clears multiple permission group errors for the specified target.
	PLIB_SB_PGVErrorClearSingle	Clears a single permission group error for the specified target.
	PLIB_SB_PGVErrorCode	Returns a value corresponding to the type of error logged for the specified target.

	PLIB_SB_PGVErrorCommandCode	Returns the OCP command code of the transaction that caused the protection violation for the specified target.
	PLIB_SB_PGVErrorInitiatorID	Returns the ID of the Initiator that caused the protection violation
	PLIB_SB_PGVErrorLogClearMulti	Clears a multiple protection group violations error from the specified target error log register.
	PLIB_SB_PGVErrorLogClearSingle	Clears a single protection group violation error from the specified target error log register.
	PLIB_SB_PGVErrorMulti	Indicates if more than one permission group violation has occurred since last cleared.
	PLIB_SB_PGVErrorPermissionGroup	Returns the permission group of the protection region in a target address space that caused the protection violation for the specified target.
	PLIB_SB_PGVErrorRegion	Returns the number of the protection region in the specified target address space that caused the protection violation.
	PLIB_SB_PGVErrorReportPrimaryDisable	Disables primary permission group error reporting for the specified target to the SB flag register.
	PLIB_SB_PGVErrorReportPrimaryEnable	Enables primary permission group error reporting for the specified target to the SB flag register.
	PLIB_SB_PGVErrorStatus	Identifies whether a permission group violation has been reported for the specified target.

Description

System Bus Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the System Bus Peripheral Library for Microchip microcontrollers. The definitions in this file are for the System Bus controller module.

File Name

plib_sb.h

Company

Microchip Technology Inc.

help_plib_sb.h

Enumerations

	Name	Description
	PLIB_SB_ARB_POLICY	This enumeration lists the possible arbitration policies that can be assigned to the CPU and DMA for SRAM access.
	PLIB_SB_ERROR	Lists the possible System Bus Transaction Error Codes.
	PLIB_SB_INIT_ID	Lists the possible System Bus Initiator IDs.
	PLIB_SB_INIT_PG	Lists the possible Initiator permission groups
	PLIB_SB_OCP_CMD_CODE	Lists the possible OCP Command codes.
	PLIB_SB_PG_INITIATOR	Lists the possible permission group Initiators.
	PLIB_SB_REGION_PG	This enumeration lists the possible permission groups assigned to a region for read and/or write access.
	PLIB_SB_TGT_ID	Lists the possible System Bus Target IDs.
	PLIB_SB_TGT_REGION	Lists the programmable target regions.
	SB_MODULE_ID	Lists the possible Module IDs for the System Bus.

Description

This file is used for documentation purposes

SPI Peripheral Library

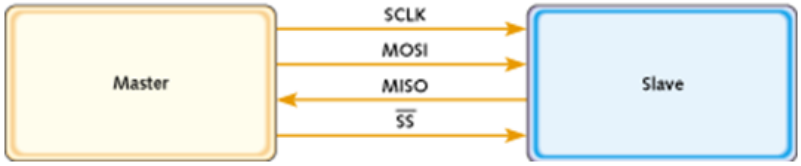
This section describes the Serial Peripheral Interface (SPI) Peripheral Library.

Introduction

This library provides a low-level abstraction of the Serial Peripheral Interface (SPI) module that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The Serial Peripheral Interface (SPI) module is a synchronous serial interface useful for communicating with other peripherals or microcontroller device. These peripheral devices may be serial EEPROMs, shift registers, display drivers, analog-to-digital converters, etc.



SPI is a synchronous serial data link operating at full duplex Master/slave relationship.

Two data lines:

- MOSI – Master Data Output, Slave Data Input
- MISO – Master Data Input, Slave Data Output

Two control lines:

- SCLK – Clock
- /SS – Slave Select (no addressing)

Devices communicate in Master/Slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (Chip Select) lines. The SPI is sometimes referred to as a "four-wire" serial bus, contrasting with three-, two-, and one-wire serial buses.

Using the Library

This topic describes the basic architecture of the SPI Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_spi.h`

The interface to the SPI Peripheral Library is defined in the `plib_spi.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the SPI Peripheral library must include `peripheral.h`.

Library File:

The SPI Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

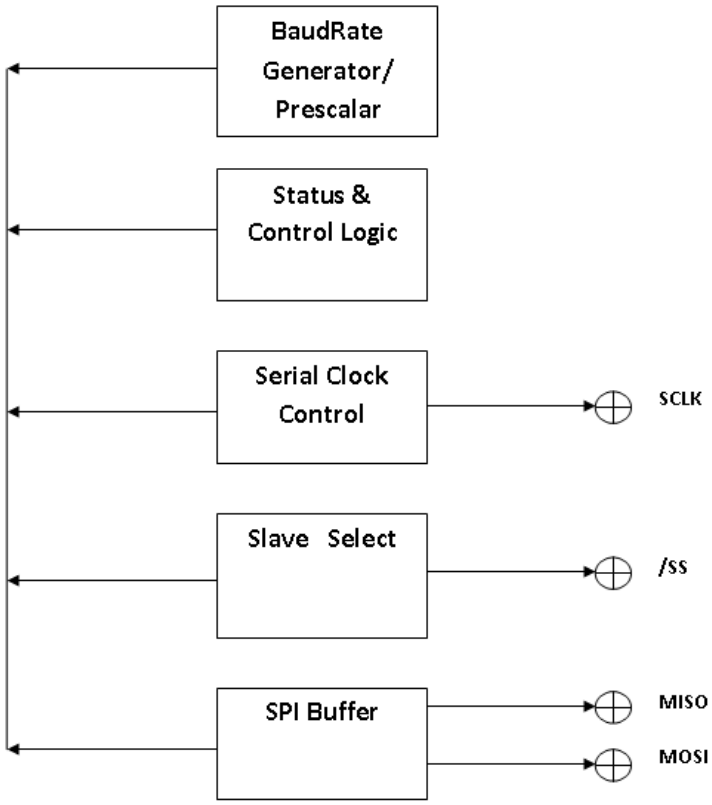
Hardware Abstraction Model

This library provides a low-level abstraction of the SPI module on the Microchip family microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The SPI Peripheral Library provides interface routines to interact with the blocks shown in the following diagram.

Hardware Abstraction Model



The Serial Peripheral Interface (API) module is a synchronous serial interface useful for communicating with external peripherals and other microcontroller devices.

The SPI bus specifies four logic signals:

- SCLK: serial clock (output from master)
- MOSI: master output, slave input (output from master)
- MISO: master input, slave output (output from slave)
- /SS: slave select (active low, output from master); on certain devices, this pin is implemented using general purpose I/O (GPIO)

The SPI bus can operate with a single master device and with one or more slave devices. To begin a communication, the master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports. In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data.

The master then transmits the appropriate chip select bit for the desired chip to a logic '0'. A logic '0' is transmitted because the Chip Select line (/SS) is active low, meaning its off state is a logic '1'; the on state is asserted with a logic 0. The master issues the clock cycles.

During each SPI clock cycle, a full duplex data transmission occurs:

- The master sends a bit on the MOSI line; the slave reads it from that same line
- The slave sends a bit on the MISO line; the master reads it from that same line

Transmissions may involve any number of clock cycles. When there are no more data to be transmitted, the master stops toggling its clock. Normally, it then deselects the slave.

The **Baud Rate Generator/Prescaler** controls the timing, the desired baud rate can be programmed in the baud rate controller.

In the Master mode, the **clock** becomes the serial clock and is provided to external devices via the SCK pin (the clock can be prescaled by the primary prescaler and the secondary prescaler if device supports).

The **Buffers** are for data transmitted or received by the SPI module over the MISO and MOSI line synchronized with the SCK line by the clock control logic.

The **Status and Control logic**, provide the capability to control different ways of enabling or disabling the master and slave. It also can provides status about the transmitter and receiver.

Active-Low **Slave Select** or Frame Synchronization I/O Pulse allows for a Synchronous Slave mode.

The SPI module supports the following four SPI modes.

- **Standard Mode:** In this mode of operation, data can be thought of as taking a direct path between the Most Significant bit (MSb) of one module's shift register and the Least Significant bit (LSb) of the other, and then into the appropriate Transmit or Receive Buffer. The master provides the serial clock and synchronization signals required to the slave device.
- **Enhanced Buffer Mode:** The operation of this mode is very similar to that of Standard mode. The difference is that data can be thought of as moving from the Shift register to a receive FIFO buffer and moving from the transmit FIFO buffer to the Shift register.
- **Framed Mode:** In this mode of operation, the Frame Master controls the generation of the frame synchronization pulse and provides this pulse to other devices at the Slave Select (/SS) pin. The SPI clock is generated by the SPI Master and is continuously running. The SPI slave module uses a frame synchronization pulse received at the SS pin.
- **Audio Protocol Mode:** The SPI module provides support to the audio protocol modes; with this mode, the SPI module can be interfaced to

most codec devices available today to provide PIC microcontroller-based audio solution

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SPI module.

Library Interface Section	Description
General Configuration and Status Functions	This section provides a set of functions and data types for configuring the SPI and to read the status of the SPI.
Data Transfer Functions	This section provides a set of functions and data types for Reading and Writing the SPI buffer values.
Transmitter Functions	This section provides a set of functions for transmitter.
Receiver Functions	This section provides a set of functions for receiver.
Framed Mode Functions	Provides control, status, and data transfer routines for Framed SPI mode.
Audio Mode Functions	Provides control, status routines to support audio protocol functionality.
Feature Existence Functions	Provides functions that determine whether certain features exist on a device.

Note: Not all modes are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

How the Library Works

Provides information on how the library works.

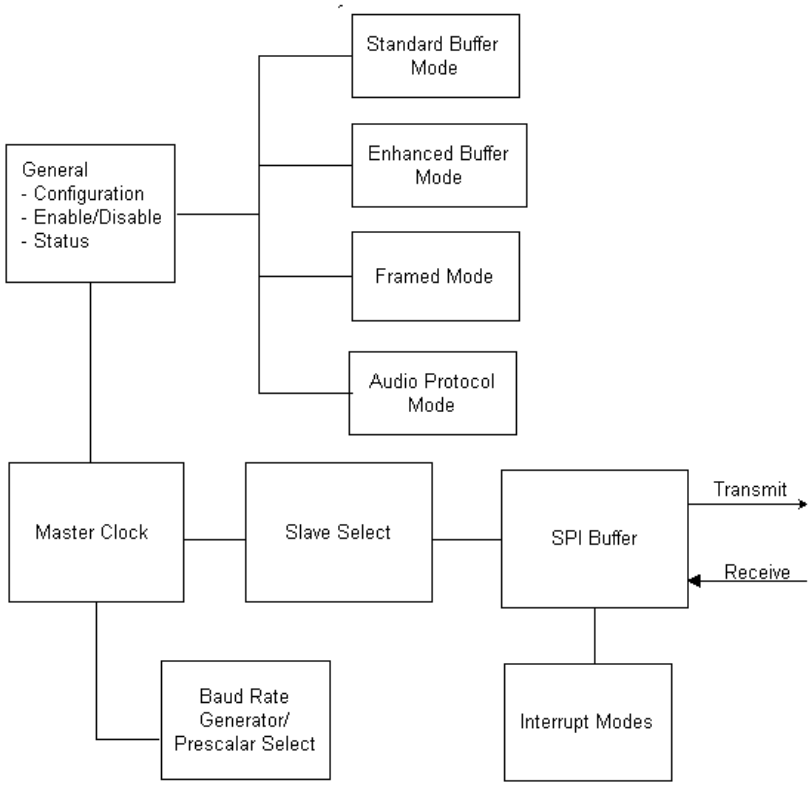
Description

The SPI module has the following operating modes:

- Standard SPI
- Enhanced Buffer SPI
- Framed SPI
- Audio Protocol Interface

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

The following diagram shows the general architecture of the SPI Peripheral Library.



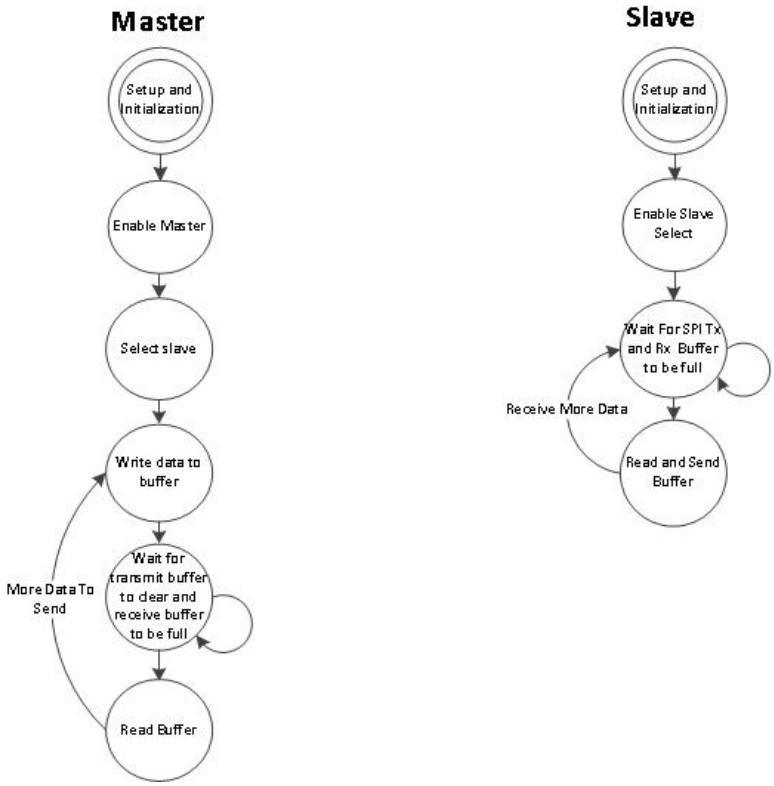
State Machine

This state machine is provided to give a general idea of the usage model of the peripheral library. Refer to the usage model for more detailed steps for the scenario that is being used.

Description

SPI State Machine

The following state machine diagrams shows the transmission and reception during normal operation.



SPI Routines

State	Associated Function
Setup and Initialization	Initialize the SPI by setting prescaler/baud rate generator, interrupt modes.
Enable Master	Once the SPI has been appropriately set up and initialized, the state machine enables Master mode (PLIB_SPI_MasterEnable).
Select Slave	Select slave by pulling the /SSx pin low to transmit the data.
Write Data To Transfer	Write data to the buffer to transmit using PLIB_SPI_BufferWrite.
Wait for Transmit Buffer to Clear	Buffer Data will be transmitted to transmit buffer and transmit buffer flag will be full. The state machine waits for transmit buffer to clear. Check for status by calling PLIB_SPI_TransmitBufferIsFull.
Wait for Receive Buffer Full to Set	Slave sends received data back to master. Upon data received receive buffer flag will set. The state machine waits for Receive buffer to clear. Check for status by calling PLIB_SPI_ReceiverBufferIsFull.
Read Buffer	Read the buffer using PLIB_SPI_BufferRead.
Enable Slave Select	Enable slave select by calling PLIB_SPI_PinEnable.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Standard SPI Mode

In Standard (legacy) Master and Slave modes, data can be thought of as taking a direct path between the Most Significant bit (MSb) of one module's shift register and the Least Significant bit (LSb) of the other, and then into the appropriate Transmit or Receive buffer. The module configured as the master module provides the serial clock and synchronization signals (as required) to the slave device.

Standard Master Mode

In Standard Master mode, the input clock is used as the serial clock. The serial clock is output via the SCK pin to slave devices. Clock pulses are only generated when there is data to be transmitted.

Description

Setup

1. Select on which edge of the clock data transmission occurs, using PLIB_SPI_OutputDataPhaseSelect and PLIB_SPI_ClockPolaritySelect.
2. Set SPI baud rate using PLIB_SPI_BaudRateSet.
3. If the module needs to operate as a Master, use PLIB_SPI_MasterEnable.

Example: Standard Master Mode Communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional: Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_SLAVE_SELECT|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_MasterEnable(MY_SPI_ID);
PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);

// Optional: Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

1. Data to be transmitted is written to the SPI buffer (PLIB_SPI_BufferWrite).
2. When contents of buffer are moved to the shift register, the SPI transmit buffer full flag is cleared (this can be verified using PLIB_SPI_TransmitBufferIsFull).
3. A series of 8/16/32 clock pulses shifts out 8/16/32 bits of transmit data from the shift register to the data out (SDO) pin and simultaneously shifts in the data at the data in (SDI) pin into the shift register.
4. When the transfer is complete, the following events occur:

- The SPI interrupt flag is set. Interrupts will occur if SPI interrupts are enabled. The SPI interrupt flag is not cleared automatically by the hardware.
- Also, when the ongoing transmit and receive operation is completed, the contents of the shift register are moved to the SPI receive register.
- The SPI receive buffer full flag (PLIB_SPI_ReceiverBufferIsFull) is set by the module, indicating that the receive buffer is full. Once the SPI buffer is read by the user application using PLIB_SPI_BufferRead, the hardware clears the SPI receive buffer full flag.

5. If the SPI receive buffer full flag is set when the SPI module needs to transfer data from SPI shift register to SPI receive buffer, the module will set the SPI receive overflow flag (PLIB_SPI_ReceiverHasOverflowed), indicating an overflow condition.
6. Data to be transmitted can be written to SPI buffer (PLIB_SPI_BufferWrite) by the user software at any time as long as the SPI Transmit buffer full flag is clear (PLIB_SPI_TransmitBufferIsFull). The write can occur while SPI shift register is shifting out the previously written data, allowing continuous transmission.

Example: Standard Master Mode Communication Transfer

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID,data);

// Either Poll for Receiver buffer to be full using
// PLIB_SPI_ReceiverBufferIsFull or wait for the interrupt
```

```
// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Standard Slave Mode

Describes Standard Slave mode.

Description

Slave Select Synchronization

The Slave Select pin (/SS) allows for Synchronous Slave mode. If the slave select is enabled (PLIB_SPI_PinEnable), transmission and reception are enabled in Slave mode only if the /SS pin is driven to a low state. The port output or other peripheral outputs must not be driven to allow the /SS pin to function as an input. If the slave select is enabled and the /SS pin is driven high, the data output (SDO) pin is no longer driven and will tri-state even if the module is in the middle of a transmission. An aborted transmission will be retried the next time the /SS pin is driven low, using the data held in the SPI Transmit buffer. If slave select is not enabled, the /SS pin does not affect the module operation in Slave mode.

SPI Transmit Buffer Full Status Operation

The function of the SPI Transmit buffer full (PLIB_SPI_TransmitBufferIsFull) is different in the Slave mode of operation. If slave select is disabled (PLIB_SPI_PinDisable), the PLIB_SPI_TransmitBufferIsFull returns a '1' when the SPI buffer is loaded by the user application. It is cleared when the module transfers data from SPI transmit buffer to SPI shift register. This is similar to the SPI Transmit buffer full function in Master mode.

If slave select is enabled (PLIB_SPI_PinEnable), PLIB_SPI_TransmitBufferIsFull returns a '1' when the SPI buffer is loaded by the user application. However,PLIB_SPI_TransmitBufferIsFull returns zero only when the SPI module completes data transmission. A transmission will be aborted when the Slave Select pin goes high and may be retried at a later time. Each data word is held in SPI transmit buffer until all bits are transmitted to the receiver.

Setup

In Standard Slave mode, data is transmitted and received as the external clock pulses appear on the SCK pin. The clock polarity and clock edge (PLIB_SPI_ClockPolaritySelect) determine upon which edge of the clock data transmission occurs.

Example: Standard Slave Mode Communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID,SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);//clock polarity and edge is subject
to change ...
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);// ... based on
your communication mode.

PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_SlaveEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
```

```
// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer (PLIB_SPI_BufferWrite/ PLIB_SPI_BufferRead). The remainder of the operation of the module is identical to that of Standard Master mode.

Example: Standard Slave Mode Communication Receive

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflowed(MY_SPI_ID))
{
   // error
   PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID);// clear overflow
   data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
     // error
}
else
{
   data = PLIB_SPI_BufferRead (MY_SPI_ID); // read data
   while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
   PLIB_SPI_BufferWrite(MY_SPI_ID, data);                // send back
}
```

Notes:

1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Enhanced Buffer SPI Mode

The operation of Enhanced Buffer Master and Slave modes is very similar to Standard Master and Slave modes. The difference is that data can be thought of as moving from the Shift register to a receive FIFO buffer and moving from the transmit FIFO buffer to the Shift register.

Enhanced Buffer Master Mode

In Enhanced Buffer Master mode (referred to as a FIFO), the input clock used as the serial clock. The serial clock is output via the SCK pin to slave devices. Clock pulses are only generated when there is data to be transmitted.

Description

Setup

1. Select on which edge of the clock data transmission occurs, using PLIB_SPI_OutputDataPhaseSelect and PLIB_SPI_ClockPolaritySelect.
2. Set SPI baud rate using PLIB_SPI_BaudRateSet.
3. If the module needs to operate as a Master, use PLIB_SPI_MasterEnable.

Example: Enhanced Master Mode Communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);
```

```
// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_SLAVE_SELECT|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_MasterEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
PLIB_SPI_FIFOEnable(MY_SPI_ID);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

The CPU loads data to be transmitted into the transmit buffer by writing the SPI buffer (PLIB_SPI_BufferWrite). An SPI transmission begins after the first buffer write. Up to all pending transmissions can be loaded. The number of pending transfers is indicated by the Buffer Element Count bits through PLIB_SPI_FIFOCountGet.

In Master mode, this count reflects the number of transfers pending in the transmit buffer. In Slave mode, it reflects the number of unread receptions in the receive buffer. If the Shift register is empty, the first write will immediately load the Shift register, leaving all transmit buffer locations available.

After an SPI transfer completes, the receive buffer location is updated with the received data. The CPU accesses the received data by reading the SPI buffer. After each CPU read, the SPI buffer points to the next buffer location. SPI transfers continue until all pending data transfers have completed.

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

1. Data to be transmitted is written to the SPI buffer (PLIB_SPI_BufferWrite) and is loaded into the next available transmit buffer location.
2. The SPI transmit buffer full flag (PLIB_SPI_TransmitBufferIsFull) and SPI interrupt flag are set after pending transfers are loaded.
3. The current buffer location's contents are moved to the Shift register. The SPI transmit buffer is cleared by the module if a buffer location is available for a CPU write.
4. A series of 8/16/32 clock pulses shifts out 8/16/32 bits of transmit data from the shift register to the data out (SDO) pin and simultaneously shifts in the data at the data in (SDI) pin into the shift register.
5. When the transfer is complete, the following events occur:

- The contents of the SPI shift register are moved into the next available location in the receive buffer.
- If the last unread location is written by the SPI module, the SPI receive buffer full flag (PLIB_SPI_ReceiveBufferIsFull) is set, indicating that all buffer locations are full. Enable the SPI interrupts. The SPI interrupt flag is not cleared automatically by the hardware.
- Once the SPI buffer is read (PLIB_SPI_BufferRead) by the user application, the hardware clears the SPI receive buffer full flag (PLIB_SPI_ReceiverBufferIsFull) and the SPI Buffer increments to the next unread receive buffer location. SPI buffer reads beyond the last unread location will not increment the buffer location.

6. When PLIB_SPI_ReceiverBufferIsFull is set, if the SPI module needs to transfer one more data from SPI shift register to the buffer, the module will enable the receive overflow flag (PLIB_SPI_ReceiverHasOverflowed) indicating an overflow condition.
7. Data to be transmitted can be written to the SPI buffer (PLIB_SPI_BufferWrite) by the user application at any time as long as the SPI transmit buffer full flag is clear (PLIB_SPI_TransmitBufferIsFull). Up to all pending transfers can be loaded into the buffer allowing continuous transmission.

Example: Enhanced Master Mode communication Transfer

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID,data);

// Either Poll for Receiver buffer to be full using
// PLIB_SPI_ReceiverBufferIsFull or wait for the interrupt

// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Enhanced Buffer Slave Mode

Describes Enhanced Buffer Slave mode.

Description

Slave Select Synchronization

The Slave Select pin allows a Synchronous Slave mode. If the slave select enabled (PLIB_SPI_PinEnable), transmission and reception is enabled in Slave mode only if the /SS pin is driven to a low state. The port output or other peripheral outputs must not be driven to allow the /SS pin to function as an input. If the slave select is enabled and the /SS pin is driven high, the SDO pin is no longer driven and will tri-state even if the module is in the middle of a transmission. An aborted transmission will be retried the next time the /SS pin is driven low using the data held in the SPI transmit buffer. If the Slave select is disabled (PLIB_SPI_PinDisable), the /SS pin does not affect the module operation in Slave mode.

SPI Transmit Buffer Full Operation

The function of the SPI transmit buffer full flag (PLIB_SPI_TransmitBufferIsFull) is different in the Slave mode of operation. If Slave select is disabled (PLIB_SPI_PinDisable), the SPI Transmit Buffer Full flag (PLIB_SPI_TransmitBufferIsFull) is set when the last available buffer location is loaded by the user application. It is cleared when the module transfers data from the buffer to SPI status register and a buffer location is available for a CPU write. This is similar to the SPI transmit buffer in Master mode.

If Slave select is enabled (PLIB_SPI_PinEnable), the SPI transmit buffer is set when the last available buffer location is loaded by the user application. However, it is cleared only when the SPI module completes data transmission, leaving a buffer location available for a CPU write. A transmission will be aborted when the /SS pin goes high and may be retried at a later time. Each data word is held in the buffer until all bits are transmitted to the receiver.

Setup

In Slave mode, data is transmitted and received as the external clock pulses appear on the SCK pin. PLIB_SPI_OutputDataPhaseSelect and PLIB_SPI_ClockPolaritySelect determine upon which edge of the clock data transmission occurs. The rest of the operation of the module is identical to that of Enhanced Buffer Master mode.

Example: Enhanced Slave Mode communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_PinEnable(MY_SPI_ID,SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_INPUT_SAMPLING_PHASE_AT_END);//clock polarity and edge is
subject to change ...
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);// ... based on
your communication mode.

PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_SlaveEnable(MY_SPI_ID);
PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
PLIB_SPI_FIFOEnable(MY_SPI_ID);

// Optional:  Enable interrupts.
```

```
// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive:

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer (PLIB_SPI_BufferWrite/PLIB_SPI_BufferRead). The remainder of the operation of the module is identical to that of Enhanced Buffer Master mode.

Example: Enhanced Slave Mode communication Receive

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflowed(MY_SPI_ID))
{
   // error
   PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID);// clear overflow
   data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
   // error
}
else
{
   data = PLIB_SPI_BufferRead (MY_SPI_ID);       // read data
   while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
   PLIB_SPI_BufferWrite(MY_SPI_ID, data);               // send back
}
```

Notes:

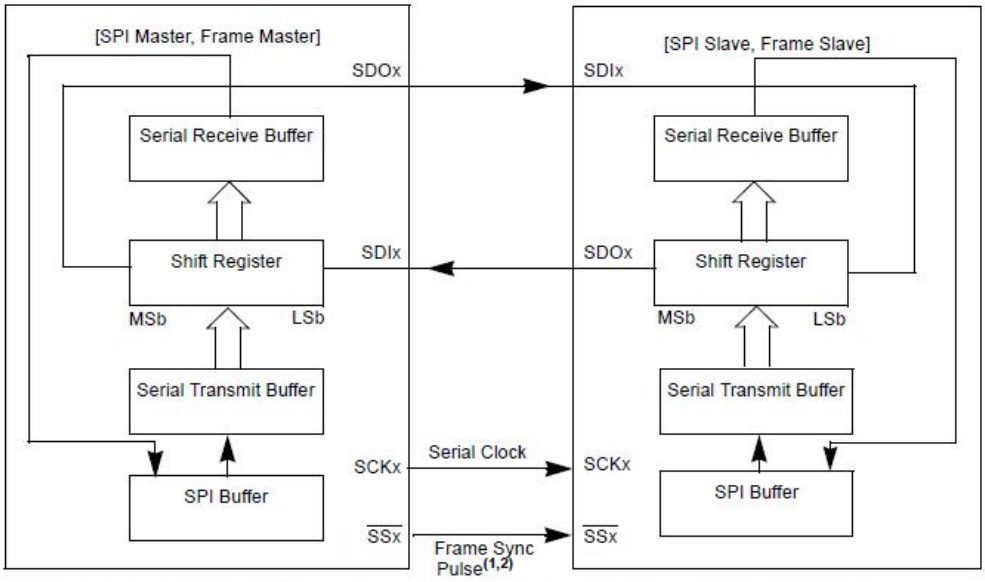
1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Framed SPI Modes

The SPI module supports a basic framed SPI protocol while operating in either Master or Slave modes.

Description

The SPI module supports two framed modes of operation. In Frame Master mode, the SPI module generates the frame synchronization pulse and provides this pulse to other devices at the /SS pin. In Frame Slave mode, the SPI module uses a frame synchronization pulse received at the /SS pin.



Note 1: In Framed SPI modes, the $\overline{SSx}$ pin is used to transmit/receive the frame synchronization pulse.
2: Framed SPI modes require the use of all four pins (i.e., using the $\overline{SSx}$ pin is not optional).

The framed SPI modes are supported in conjunction with the unframed Master and Slave modes. This makes four framed SPI configurations:

- SPI Master mode and Frame Master mode
- SPI Master mode and Frame Slave mode
- SPI Slave mode and Frame Master mode
- SPI Slave mode and Frame Slave mode

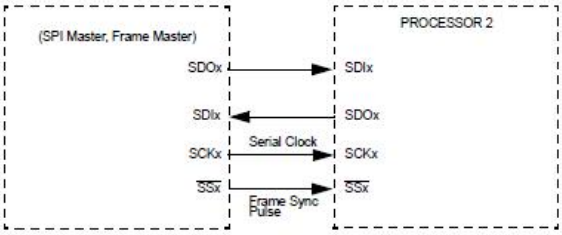
These modes determine whether or not the SPI module generates the serial clock and the frame synchronization pulse.

SPI Master Mode and Frame Master Mode

Describes Master/Frame Master mode.

Description

In Master/Frame Master mode, the SPI module generates both the clock and frame synchronization signals, as shown in the following figure.



In this mode, the serial clock is output continuously at the SCK pin, regardless of whether the module is transmitting. When the SPI buffer is written (PLIB_SPI_BufferWrite), the /SS pin will be driven to its active state on the appropriate transmit edge of the SCK clock, and remain active for one data frame. If the PLIB_SPI_FrameSyncPulseEdgeSelect function decides whether sync pulse precedes the data transmission or coincides with the beginning of the data transmission. The module starts transmitting data on the next transmit edge of the SCK.

Setup

The mode is enabled by calling PLIB_SPI_MasterEnable, PLIB_SPI_FramedCommunicationEnable and PLIB_SPI_FrameSyncPulseDirectionSelect (as output).

Example: SPI Master Mode and Frame Master Mode Communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
```

```
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_SLAVE_SELECT|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_MasterEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationEnable(MY_SPI_ID);
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_ID, SPI_FRAME_PULSE_DIRECTION_INPUT);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: SPI Master Mode and Frame Master Mode Communication Transfer

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID,data);

// Either Poll for Receiver buffer to be full using
// PLIB_SPI_ReceiverBufferIsFull or wait for the interrupt

// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

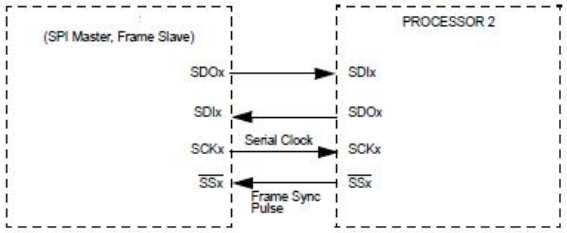
1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

SPI Master Mode and Frame Slave Mode

Describes Master/Frame Slave mode.

Description

In Master/Frame Slave mode, the module generates the clock signal but uses the slave module's frame synchronization signal for data transmission, as shown in the following figure.



In this mode, the /SS pin is an input and it is sampled on the sample edge of the SPI clock. When it is sampled in its active state, data will be transmitted on the subsequent transmit edge of the SPI clock. The SPI interrupt flag is set when the transmission is complete. The user application must make sure that the correct data is loaded into the SPI buffer for transmission before the signal is received at the /SS pin.

Setup

The mode is enabled by calling PLIB_SPI_MasterEnable, PLIB_SPI_FramedCommunicationEnable, and PLIB_SPI_FrameSyncPulseDirectionSelect.

Example: SPI Master Mode and Frame Slave Mode Communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_SLAVE_SELECT|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_FrameSyncPulseCounterSelect(MY_SPI_ID,SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER);
PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_MasterEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationEnable(MY_SPI_ID);
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_ID,SPI_FRAME_PULSE_DIRECTION_INPUT);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: SPI Master Mode and Frame Slave Mode Communication Transfer

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID,data);

// Either Poll for Receiver buffer to be full using
// PLIB_SPI_ReceiverBufferIsFull or wait for the interrupt

// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

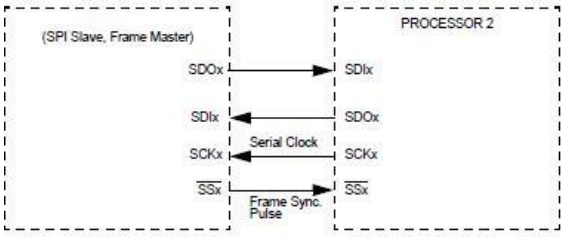
1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

SPI Slave Mode and Frame Master Mode

Describes Slave/Frame Master mode.

Description

In Slave/Frame Master mode, the module acts as the SPI slave and takes its clock from the other SPI module; however, it produces frame synchronization signals to control data transmission, as shown in the following figure.



The input SPI clock will be continuous in Slave mode. The /SS pin will be an output when the Frame sync pulse is output. Therefore, when the SPI buffer is written, the module drives the /SS pin to the active state on the appropriate transmit edge of the SPI clock for one SPI clock cycle. Data will start transmitting on the appropriate SPI clock transmit edge.

Setup

The mode is enabled by calling PLIB_SPI_SlaveEnable, PLIB_SPI_FramedCommunicationEnable, and PLIB_SPI_FrameSyncPulseDirectionSelect.

Example: SPI Slave Mode and Frame Master Mode Communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID,SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);//clock polarity and edge is subject
to change ...
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);// ... based on
your communication mode.
PLIB_SPI_FrameSyncPulseCounterSelect(MY_SPI_ID,SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER);

PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_SlaveEnable(MY_SPI_ID);
PLIB_SPI_FramedCommunicationEnable(MY_SPI_ID);
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_ID,SPI_FRAME_PULSE_DIRECTION_INPUT);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: SPI Slave Mode and Frame Master Mode Communication Receive

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflowed(MY_SPI_ID))
{
   // error
```

```
    PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID);// clear overflow
    data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
    // error
}
else
{
    data = PLIB_SPI_BufferRead (MY_SPI_ID);       // read data
    while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
    PLIB_SPI_BufferWrite(MY_SPI_ID, data);                // send back
}
```

Notes:

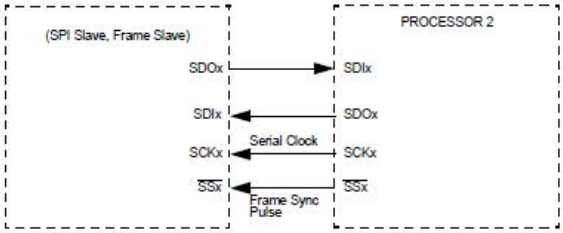
1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

SPI Slave Mode and Frame Master Mode

Describes Slave/Frame Slave mode.

Description

In Slave/Frame Slave mode, the module obtains both its clock and frame synchronization signal from the master module, as shown in the following figure.



In this mode, both the clock and Slave Select pins will be inputs. The /SS pin is sampled on the sample edge of the SPI clock. When /SS is sampled at its active state, data will be transmitted on the appropriate transmit edge of SCK.

Setup

The mode is enabled by calling PLIB_SPI_SlaveEnable,PLIB_SPI_FramedCommunicationEnable and PLIB_SPI_FrameSyncPulseDirectionSelect (as input).

Example: SPI Slave Mode and Frame Slave Mode Communication Setup

```
#define MY_SPI_ID             SPI_ID_1
#define PERIPHERAL_BUS_CLK    80000000
#define SPI_BAUD_RATE         10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID,SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);//clock polarity and edge is subject
to change ...
```

```
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);// ... based on
your communication mode.
PLIB_SPI_FrameSyncPulseCounterSelect(MY_SPI_ID,SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER);

PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_SlaveEnable(MY_SPI_ID);

PLIB_SPI_FramedCommunicationEnable(MY_SPI_ID);
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_ID,SPI_FRAME_PULSE_DIRECTION_INPUT);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: SPI Slave Mode and Frame Slave Mode Communication Receive

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflowed(MY_SPI_ID))
{
   // error
   PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID);// clear overflow
   data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
   // error
}
else
{
   data = PLIB_SPI_BufferRead (MY_SPI_ID);       // read data
   while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
   PLIB_SPI_BufferWrite(MY_SPI_ID, data);               // send back
}
```

Notes:

1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Audio Protocol Interface Mode

The SPI module can be interfaced to most codec devices available today to provide PIC microcontroller-based audio solutions.

Description

The SPI module provides support to the audio protocol functionality via four standard I/O pins. The four pins that make up the audio protocol interface modes are:

- SDIx: Serial Data Input for receiving sample digital audio data
- SDOx: Serial Data Output for transmitting digital audio data
- SCKx: Serial Clock (also known as bit clock)
- /SSx: Left/Right Channel Clock

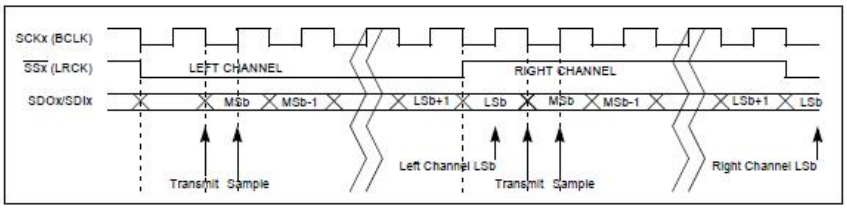
The SPI module supports four audio protocol modes and can be operated in any one of these modes:

Note: Each of the modes can additionally support some or all of the following features. Please refer to the"**Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for more information.

I2S Mode

The Inter-IC Sound (I2S) protocol enables transmission of two channels of digital audio data over a single serial interface. The I2S specification defines a half-duplex interface that supports transmit or receive, but not both at the same time. With both SDO and SDI available, full-duplex

operation is supported by this peripheral, as shown in the following figure.



Data Transmit and Clocking:

- The transmitter shifts the audio sample data's Most Significant bit (MSb) on the first falling edge of SCK after an LRCK transition
- The receiver samples the MSB on the second rising edge of SCK
- The left channel data shifts out while LRCK is low and right channel data is shifted out while LRCK is high
- The data in the left and right channel consists of a single frame

To set the module to I2S mode, the following bits must be set:

- Set SPI to I2S Mode by calling PLIB_SPI_AudioProtocolModeSelect
- Select PLIB_SPI_FrameSyncPulsePolaritySelect (as active low)
- Select PLIB_SPI_ClockPolaritySelect (as active low)

Setting these bits enables the SDO and LRCK (/SSx) transitions to occur on the falling edge of SCK (BCLK) and sampling of SDI to occur on the rising edge of SCK.

Left-Justified Mode

The Left-Justified mode is similar to I2S mode; however, in this mode, the SPI shifts the audio data's MSb on the first SCK edge that is coincident with an LRCK transition. On the receiver side, the SPI module samples the MSb on the next SCK edge.

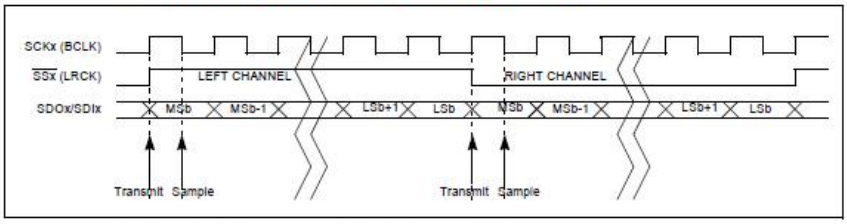
In general, a codec using justified protocols defaults to transmitting data on the rising edge of SCK and receiving data on the falling edge of SCK.

To set the module to Left-Justified mode, the following bits must to be set

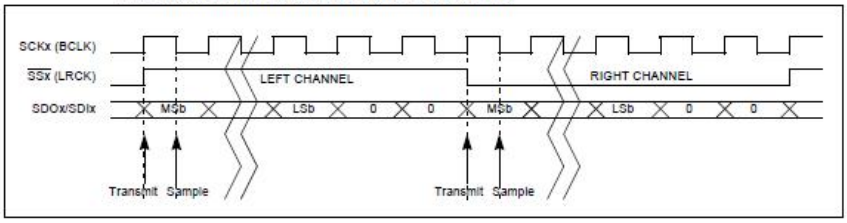
- Set SPI to Left Justified Mode by calling PLIB_SPI_AudioProtocolModeSelect
- Select PLIB_SPI_FrameSyncPulsePolaritySelect (as active high)
- Select PLIB_SPI_ClockPolaritySelect (as active high)

This enables the SDO and LRCK transitions to occur on the rising edge of SCK. Refer to the following sample waveform diagrams for 16-, 24-, and 32-bit audio data transfers.

Left-Justified with 16-bit Data/Channel or 32-bit Data/Channel



Left-Justified with 16/24-bit Data and 32-bit Channel



Right-Justified Mode

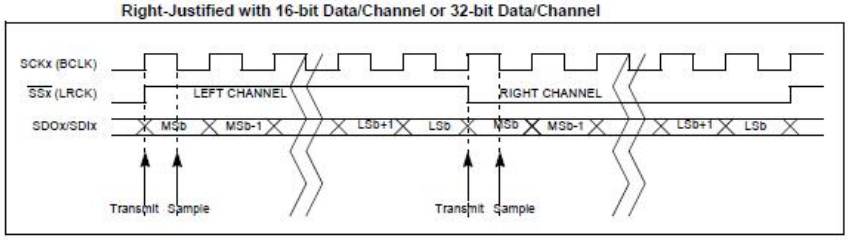
In Right-Justified mode, the SPI module shifts the audio sample data's MSb after aligning the data to the last clock cycle. The bits preceding the audio sample data can be driven to logic level 0 .

To set the module to Right-Justified mode, the following bits must to be set:

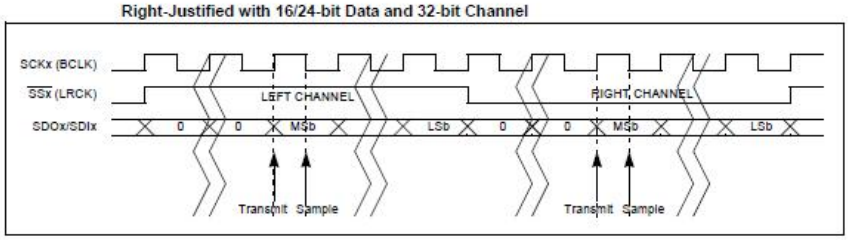
- Set SPI to Right Justified Mode by calling PLIB_SPI_AudioProtocolModeSelect
- Select PLIB_SPI_FrameSyncPulsePolaritySelect (as active high)
- Select PLIB_SPI_ClockPolaritySelect (as active high)

This enables the SDO and LRCK transitions to occur on the rising edge of SCK after the Least Significant bit (LSb) being aligned to the last clock cycle. Refer to the following sample waveform diagrams for 16-, 24-, and 32-bit audio data transfers.

Right-Justified with 16-bit Data/Channel or 32-bit Data/Channel



Right-Justified with 16/24-bit Data and 32-bit Channel



PCM/DSP Mode

This mode modifies the behavior of LRCK and audio data spacing. In PCM/DSP mode, the LRCK can be a single bit wide (i.e., 1 SCK) or as wide as the audio data (16-, 24-, and 32-bits). The audio data is packed in the frame with the left channel data immediately followed by the right channel data. The frame length is still either 32 or 64 clocks when this device is the master.

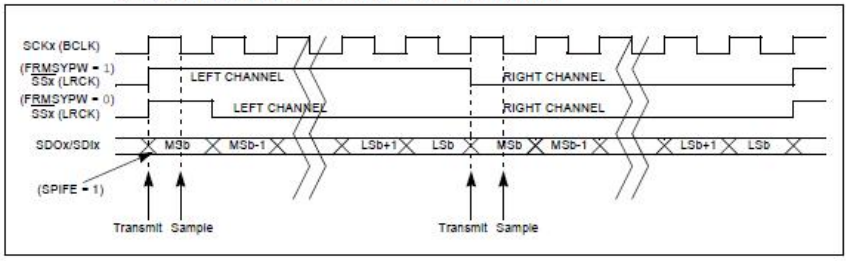
In PCM/DSP mode, the transmitter drives the audio data's (left channel) MSb on the first or second transmit edge of SCK (after an LRCK transition). Immediately after the (left channel) LSb, the transmitter drives the (right channel) MSb.

To set the module to Left-Justified mode, the following bit must to be set:

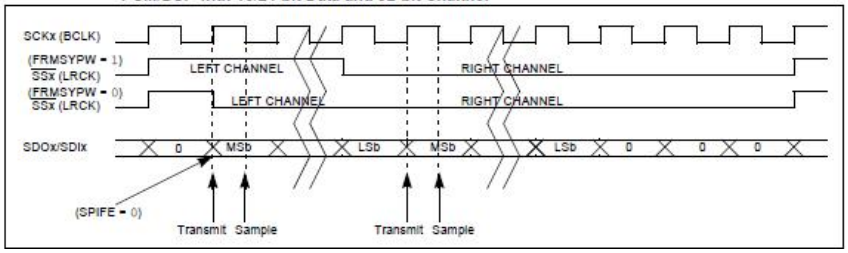
- Set the SPI module to Right-Justified Mode by calling PLIB_SPI_AudioProtocolModeSelect

Refer to the following sample waveform for 16-, 24-, and 32-bit audio data transfers.

PCM/DSP with 16-bit Data/Channel or 32-bit Data/Channel



PCM/DSP with 16/24-bit Data and 32-bit Channel



Mono Mode Versus Stereo Mode

The SPI module enables the audio data transmission in Mono or Stereo mode by setting PLIB_SPI_AudioTransmitModeSelect. In Stereo mode, the shift register uses each FIFO location once, which gives each channel a unique stream of data for stereo data. In Mono mode, the shift register uses each FIFO location twice, to give each channel the same mono stream of audio data.

Master Mode

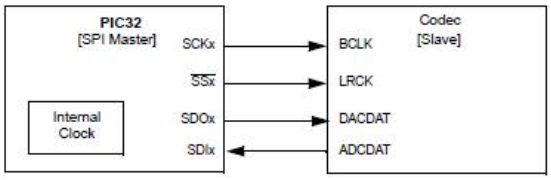
Describes Master mode for the Audio Protocol Interface mode.

Description

A few characteristics of Master mode are:

- This mode enables the device to generate SCK and LRCK pulses as long as the master mode enabled
- The SPI module generates LRCK and SCK continuously in all the cases, regardless of the transmit data while in Master mode
- The SPI module drives the leading edge of LRCK and SCK within 1 SCK period and the serial data shifts in and out continuously even when the Transmit FIFO is empty

The following figure shows a typical interface between the master and slave while in Master mode.



Setup

To configure the device in Audio Protocol Master, enable master mode through PLIB_SPI_MasterEnable and enable audio mode through PLIB_SPI_AudioProtocolEnable.

Example: Audio Mode Communication Setup

```
#define MY_SPI_ID             SPI_ID_1
#define PERIPHERAL_BUS_CLK    80000000
#define SPI_BAUD_RATE         10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear (MY_SPI_ID);

// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID, SPI_PIN_DATA_IN|SPI_PIN_DATA_OUT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_MasterEnable(MY_SPI_ID);
PLIB_SPI_AudioProtocolModeSelect(MY_SPI_ID,SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED );

PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
PLIB_SPI_AudioProtocolEnable(MY_SPI_ID);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer.

Example: Audio Mode Communication Transfer

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;
data = 0x00ac;

// write to buffer for TX
PLIB_SPI_BufferWrite (MY_SPI_ID,data);

// wait for transfer to complete
while(!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID));

// Read the received value
data = PLIB_SPI_BufferRead (MY_SPI_ID);
```

Notes:

1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Slave Mode

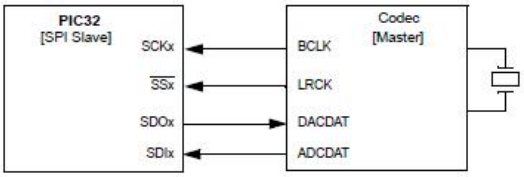
Describes Slave mode for the Audio Protocol Interface mode.

Description

A few characteristics of Slave mode are:

- This mode enables the device to receive SCK and LRCK pulses as long as slave mode is enabled
- The SPI module drives zeros out of SDO, but does not shift data out or in (SDI) until the module receives the LRCK (i.e., the edge that precedes the left channel)
- Once the module receives the leading edge of LRCK, it starts receiving data if PLIB_SPI_PinEnable (enables data in) is selected and the serial data shifts out continuously even when the TX FIFO is empty

The following figure shows the interface between a SPI module in Audio Slave Interface mode to a codec master device.



Setup

The SPI module can be configured in Audio Protocol Slave mode by setting PLIB_SPI_SlaveEnable and enabling the audio protocol through PLIB_SPI_AudioProtocolEnable.

Example: Slave Mode communication Setup

```
#define MY_SPI_ID            SPI_ID_1
#define PERIPHERAL_BUS_CLK   80000000
#define SPI_BAUD_RATE        10000000

//Disable SPI
PLIB_SPI_Disable(MY_SPI_ID);

// Optional:  Clear SPI interrupts and status flag.

//clear SPI buffer
PLIB_SPI_BufferClear(MY_SPI_ID);

// Configure General SPI Options
// Configure General SPI Options
PLIB_SPI_StopInIdleDisable(MY_SPI_ID);
PLIB_SPI_PinEnable(MY_SPI_ID,SPI_PIN_DATA_OUT|SPI_PIN_SLAVE_SELECT);
PLIB_SPI_CommunicationWidthSelect(MY_SPI_ID, SPI_COMMUNICATION_WIDTH_16BITS);

// SMP must be cleared when SPI is used in Slave mode
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_ID,SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
PLIB_SPI_ClockPolaritySelect(MY_SPI_ID, SPI_CLOCK_POLARITY_IDLE_HIGH);//clock polarity and edge is subject
to change ...
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_ID, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);// ... based on
your communication mode.

PLIB_SPI_BaudRateSet(MY_SPI_ID,PERIPHERAL_BUS_CLK,SPI_BAUD_RATE);
PLIB_SPI_SlaveEnable(MY_SPI_ID);
PLIB_SPI_AudioProtocolModeSelect(MY_SPI_ID,SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED );

PLIB_SPI_FramedCommunicationDisable(MY_SPI_ID);
PLIB_SPI_AudioProtocolEnable(MY_SPI_ID);

// Optional:  Enable interrupts.

// Enable the module
PLIB_SPI_Enable(MY_SPI_ID);
```

Transmission and Receive

Both data to be transmitted and data that is received are respectively written into, or read from, the SPI buffer (PLIB_SPI_BufferWrite/ PLIB_SPI_BufferRead).

Example: Slave Mode communication Receive

```
#define MY_SPI_ID   SPI_ID_1

uint16_t data;

if(PLIB_SPI_ReceiverHasOverflowed(MY_SPI_ID))
{
  // error
  PLIB_SPI_ReceiverOverflowClear(MY_SPI_ID);// clear overflow
  data = PLIB_SPI_BufferRead (MY_SPI_ID);
}
else if (!PLIB_SPI_ReceiverBufferIsFull(MY_SPI_ID))
{
   // error
}
else
{
   data = PLIB_SPI_BufferRead (MY_SPI_ID);       // read data
   while(PLIB_SPI_TransmitBufferIsFull(MY_SPI_ID));
   PLIB_SPI_BufferWrite(MY_SPI_ID, data);               // send back
}
```

Notes:

1. Refer to the **"Interrupts"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details on how to clear and enable the SPI module interrupts and flags.
2. MY_SPI_ID is the SPI instance selected for use by the application developer.
3. Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Other Features

This topic describes the additional features available in the SPI Peripheral Library.

Communication Mode

The SPI module allows three types of data widths when transmitting and receiving data over a SPI bus. The selection of data width determines the minimum length of the SPI data. The user application should select the appropriate data width to maximize its data throughput. To change the mode of operation on the fly, the SPI module must be idle. If the SPI module is switched off, the new mode will be available when the module is switched on again.

The macro, PLIB_SPI_CommunicationWidthSet, allows the module to communicate in either 8-/16-/32-bit modes. The functionality will be the same for each mode, except for the number of bits that are received and transmitted.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Power-Saving Modes

Note: Each of the following modes can additionally support some or all of the following features. Please refer to the **"Power-Saving Modes"** section of the specific device data sheet or the family reference manual section specified in that chapter for more information.

Sleep Mode

When the device enters Sleep mode, the device clock source and the entire device is shut down. The consequences of entering Sleep depend upon which mode (Master or Slave) the module is configured in at the time that Sleep mode is invoked.

Idle Mode

When the device enters Idle mode, the device clock is operational, but the CPU and selected peripherals are shut down.

The SPI module can continue to operate in Idle mode by calling PLIB_SPI_StopInIdleDisable, or can stop operation in Idle mode by calling PLIB_SPI_StopInIdleEnable.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet to determine availability.

SPI Receive-only Operation

Calling PLIB_SPI_PinDisable (data in pin) disables the transmission at the SDO pin. This allows the SPI module to be configured for a receive-only mode of operation. This function is applicable to all SPI operating modes.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

SPI Error Handling

If a new data word has been shifted and the previous buffer contents have not been read, the status can be read by calling PLIB_SPI_ReceiverHasOverflowed. Any received data is not transferred, and further data reception is disabled until the buffer is cleared by calling PLIB_SPI_ReceiverOverflowClear.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

SPI Master Mode Clock Frequency

In Master mode, the PBCLK is divided and then used as the serial clock. The division is based on the settings in the SPIxBRG register. The serial clock is output via the SCKx pin to slave devices.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Interrupts

The SPI module has the ability to generate interrupts reflecting the events that occur during the data communication. Interrupts can be enabled or disabled using PLIB_SPI_ErrorInterruptEnable/PLIB_SPI_ErrorInterruptDisable.

The following types of interrupts can be generated:

- Receive data available interrupts are signalled by SPI receive and transmit flag. This event occurs when there is new data assembled in the SPI receive buffer.
- Transmit buffer empty interrupts are signalled by SPI receive and transmit flag. This event occurs when there is space available in the SPI transmit buffer and new data can be written.
- Error interrupts are signalled by SPI receive and transmit flag. This event occurs when there is an overflow condition for the receive buffer when there is an underrun of the transmit buffer, or when a frame error event occurs.

Note: Not all features are available on all devices. Refer to the **"Serial Peripheral Interface (SPI)"** chapter in the specific device data sheet for availability.

Configuring the Library

The library is configured for the supported SPI modules when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration and Status Functions

	Name	Description
	PLIB_SPI_BaudRateClockSelect	Selects the type of clock is used by the Baud Rate Generator.
	PLIB_SPI_BaudRateSet	Sets the baud rate to the desired value.
	PLIB_SPI_ClockPolaritySelect	Enables clock polarity.
	PLIB_SPI_CommunicationWidthSelect	Selects the data width for the SPI communication.
	PLIB_SPI_Disable	Disables the SPI module.
	PLIB_SPI_Enable	Enables the SPI module.
	PLIB_SPI_ErrorInterruptDisable	Enables SPI error interrupts.
	PLIB_SPI_ErrorInterruptEnable	Enables SPI error interrupts

	PLIB_SPI_FIFOCountGet	Reads the SPI Buffer Element Count bits for either receive or transmit.
	PLIB_SPI_FIFODisable	Disables the SPI enhanced buffer.
	PLIB_SPI_FIFOEnable	Enables the SPI enhanced buffer.
	PLIB_SPI_FIFOInterruptModeSelect	Selects the SPI buffer interrupt mode.
	PLIB_SPI_FIFOShiftRegisterIsEmpty	Returns the current status of the SPI shift register.
	PLIB_SPI_InputSamplePhaseSelect	Selects the SPI data input sample phase.
	PLIB_SPI_IsBusy	Returns the current SPI module activity status.
	PLIB_SPI_MasterEnable	Enables the SPI in Master mode.
	PLIB_SPI_OutputDataPhaseSelect	Selects serial output data change.
	PLIB_SPI_PinDisable	Enables the selected SPI pins.
	PLIB_SPI_PinEnable	Enables the selected SPI pins.
	PLIB_SPI_ReadDataIsSignExtended	Returns the current status of the receive (RX) FIFO sign-extended data.
	PLIB_SPI_SlaveEnable	Enables the SPI in Slave mode.
	PLIB_SPI_SlaveSelectDisable	Disables Master mode slave select.
	PLIB_SPI_SlaveSelectEnable	Enables Master mode slave select.
	PLIB_SPI_StopInIdleDisable	Continues module operation when the device enters Idle mode.
	PLIB_SPI_StopInIdleEnable	Discontinues module operation when the device enters Idle mode.

b) Data Transfer Functions

	Name	Description
	PLIB_SPI_BufferClear	Clears the SPI buffer.
	PLIB_SPI_BufferRead	Returns the SPI buffer value.
	PLIB_SPI_BufferAddressGet	Returns the address of the SPIxBUF (Transmit(SPIxTXB) and Receive (SPIxRXB)) register.
	PLIB_SPI_BufferRead16bit	Returns 16-bit SPI buffer value.
	PLIB_SPI_BufferRead32bit	Returns 32-bit SPI buffer value.
	PLIB_SPI_BufferWrite	Write the data to the SPI buffer.
	PLIB_SPI_BufferWrite16bit	Writes 16-bit data to the SPI buffer.
	PLIB_SPI_BufferWrite32bit	Write 32-bit data to the SPI buffer.

c) Framed Mode Functions

	Name	Description
	PLIB_SPI_FramedCommunicationDisable	Disables framed SPI support.
	PLIB_SPI_FramedCommunicationEnable	Enables framed SPI support.
	PLIB_SPI_FrameErrorStatusGet	Returns the current status of the SPI frame error.
	PLIB_SPI_FrameErrorStatusClear	Clears the SPI frame error flag.
	PLIB_SPI_FrameSyncPulseCounterSelect	Selects at which character the SPI frame sync pulse is generated.
	PLIB_SPI_FrameSyncPulseDirectionSelect	Selects the frame sync pulse direction.
	PLIB_SPI_FrameSyncPulseEdgeSelect	Selects the frame sync pulse edge.
	PLIB_SPI_FrameSyncPulsePolaritySelect	Selects the frame sync pulse polarity.
	PLIB_SPI_FrameSyncPulseWidthSelect	Sets the frame sync pulse width.

d) Audio Mode Functions

	Name	Description
	PLIB_SPI_AudioCommunicationWidthSelect	Selects the data width for the SPI audio communication.
	PLIB_SPI_AudioErrorDisable	Disables the SPI error.
	PLIB_SPI_AudioErrorEnable	Enables the SPI error.
	PLIB_SPI_AudioProtocolDisable	Audio protocol is disabled.
	PLIB_SPI_AudioProtocolEnable	Audio protocol is enabled.
	PLIB_SPI_AudioProtocolModeSelect	Selects the Audio Protocol mode.
	PLIB_SPI_AudioTransmitModeSelect	Selects the transmit audio data format.

e) Transmitter Functions

	Name	Description
	PLIB_SPI_TransmitBufferIsEmpty	Returns the current status of the transmit buffer.
	PLIB_SPI_TransmitBufferIsFull	Returns the current transmit buffer status of the SPI module.

	PLIB_SPI_TransmitUnderRunStatusGet	Returns the current status of the transmit underrun.
	PLIB_SPI_TransmitUnderRunStatusClear	Clears the SPI transmit underrun flag.

f) Receiver Functions

	Name	Description
	PLIB_SPI_ReceiverBufferIsFull	Returns the current status of the SPI receive buffer.
	PLIB_SPI_ReceiverFIFOIsEmpty	Returns the current status of the SPI receive FIFO.
	PLIB_SPI_ReceiverHasOverflowed	Returns the current status of the SPI receiver overflow.
	PLIB_SPI_ReceiverOverflowClear	Clears the SPI receive overflow flag.

g) Feature Existence Functions

	Name	Description
	PLIB_SPI_ExistsAudioCommunicationWidth	Identifies whether the AudioCommunicationWidth feature exists on the SPI module.
	PLIB_SPI_ExistsAudioErrorControl	Identifies whether the AudioErrorControl feature exists on the SPI module.
	PLIB_SPI_ExistsAudioProtocolControl	Identifies whether the AudioProtocolControl feature exists on the SPI module.
	PLIB_SPI_ExistsAudioProtocolMode	Identifies whether the AudioProtocolMode feature exists on the SPI module.
	PLIB_SPI_ExistsAudioTransmitMode	Identifies whether the AudioTransmitMode feature exists on the SPI module.
	PLIB_SPI_ExistsBaudRate	Identifies whether the BaudRate feature exists on the SPI module.
	PLIB_SPI_ExistsBaudRateClock	Identifies whether the BaudRateClock feature exists on the SPI module.
	PLIB_SPI_ExistsBuffer	Identifies whether the Buffer feature exists on the SPI module.
	PLIB_SPI_ExistsBusStatus	Identifies whether the BusStatus feature exists on the SPI module.
	PLIB_SPI_ExistsClockPolarity	Identifies whether the ClockPolarity feature exists on the SPI module.
	PLIB_SPI_ExistsCommunicationWidth	Identifies whether the CommunicationWidth feature exists on the SPI module.
	PLIB_SPI_ExistsEnableControl	Identifies whether the EnableControl feature exists on the SPI module.
	PLIB_SPI_ExistsErrorInterruptControl	Identifies whether the ErrorInterruptControl feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOControl	Identifies whether the FIFOControl feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOCount	Identifies whether the FIFOCount feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOInterruptMode	Identifies whether the FIFOInterruptMode feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus	Identifies whether the FIFOShiftRegisterEmptyStatus feature exists on the SPI module.
	PLIB_SPI_ExistsFramedCommunication	Identifies whether the FramedCommunication feature exists on the SPI module.
	PLIB_SPI_ExistsFrameErrorStatus	Identifies whether the FrameErrorStatus feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseCounter	Identifies whether the FrameSyncPulseCounter feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseDirection	Identifies whether the FrameSyncPulseDirection feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseEdge	Identifies whether the FrameSyncPulseEdge feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulsePolarity	Identifies whether the FrameSyncPulsePolarity feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseWidth	Identifies whether the FrameSyncPulseWidth feature exists on the SPI module.
	PLIB_SPI_ExistsInputSamplePhase	Identifies whether the InputSamplePhase feature exists on the SPI module.
	PLIB_SPI_ExistsMasterControl	Identifies whether the MasterControl feature exists on the SPI module.
	PLIB_SPI_ExistsOutputDataPhase	Identifies whether the OutputDataPhase feature exists on the SPI module.
	PLIB_SPI_ExistsPinControl	Identifies whether the PinControl feature exists on the SPI module.
	PLIB_SPI_ExistsReadDataSignStatus	Identifies whether the ReadDataSignStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiveBufferStatus	Identifies whether the ReceiveBufferStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiveFIFOStatus	Identifies whether the ReceiveFIFOStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiverOverflow	Identifies whether the ReceiverOverflow feature exists on the SPI module.
	PLIB_SPI_ExistsSlaveSelectControl	Identifies whether the SlaveSelectControl feature exists on the SPI module.
	PLIB_SPI_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitBufferEmptyStatus	Identifies whether the TransmitBufferEmptyStatus feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitBufferFullStatus	Identifies whether the TransmitBufferFullStatus feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitUnderRunStatus	Identifies whether the TransmitUnderRunStatus feature exists on the SPI module.
	PLIB_SPI_Exists16bitBuffer	Identifies whether the Buffer16bit feature exists on the SPI module.
	PLIB_SPI_Exists32bitBuffer	Identifies whether the Buffer32bit feature exists on the SPI module.

h) Data Types and Constants

	Name	Description
	SPI_AUDIO_COMMUNICATION_WIDTH	Defines the list of SPI audio communication width.
	SPI_AUDIO_ERROR	Defines the list of SPI audio error.
	SPI_AUDIO_PROTOCOL	Data type defining the audio protocol mode.
	SPI_AUDIO_TRANSMIT_MODE	Defines the list of SPI transmit audio mode format.
	SPI_BAUD_RATE_CLOCK	Defines the list of SPI Baud Rate Generator (BRG).
	SPI_CLOCK_POLARITY	Defines the list of SPI clock polarity.
	SPI_COMMUNICATION_WIDTH	Defines the list of SPI communication width.
	SPI_DATA_TYPE	Data type defining the SPI data size.
	SPI_ERROR_INTERRUPT	Defines the list of SPI error interrupts.
	SPI_FIFO_INTERRUPT	Defines the list of SPI Buffer Interrupt mode.
	SPI_FIFO_TYPE	Defines the list of SPI buffer mode.
	SPI_FRAME_PULSE_DIRECTION	Defines the list of SPI frame sync pulse direction.
	SPI_FRAME_PULSE_EDGE	Defines the list of SPI frame sync pulse edge.
	SPI_FRAME_PULSE_POLARITY	Defines the list of SPI frame sync pulse polarity.
	SPI_FRAME_PULSE_WIDTH	Defines the list of SPI frame sync pulse width.
	SPI_FRAME_SYNC_PULSE	Data type defining the frame sync pulse counter values.
	SPI_INPUT_SAMPLING_PHASE	Defines the list of SPI data input sample phase.
	SPI_MODULE_ID	Identifies the supported SPI modules.
	SPI_OUTPUT_DATA_PHASE	Defines the list of SPI serial output data changes.
	SPI_PIN	Data type defining the SPI pin.

Description

This section describes the Application Programming Interface (API) functions of the SPI Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration and Status Functions

PLIB_SPI_BaudRateClockSelect Function

Selects the type of clock is used by the Baud Rate Generator.

File

plib_spi.h

C

```
void PLIB_SPI_BaudRateClockSelect(SPI_MODULE_ID index, SPI_BAUD_RATE_CLOCK type);
```

Returns

None.

Description

This function selects the type of clock is used by the Baud Rate Generator.

Remarks

This function implements an operation of the baud rate clock control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsBaudRateClock" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_BaudRateClockSelect (MY_SPI_INSTANCE, SPI_BAUD_RATE_MCLK_CLOCK);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
type	One of the SPI_BAUD_RATE_CLOCK enumeration values as the SPI baud clock

Function

void PLIB_SPI_BaudRateClockSelect (SPI_MODULE_ID index,
SPI_BAUD_RATE_CLOCK type)

PLIB_SPI_BaudRateSet Function

Sets the baud rate to the desired value.

File

plib_spi.h

C

```
void PLIB_SPI_BaudRateSet(SPI_MODULE_ID index, uint32_t clockFrequency, uint32_t baudRate);
```

Returns

None.

Description

This function sets the baud rate to the desired value.

Remarks

Setting a new baud rate value causes the baud rate timer to reset. This ensures that the baud rate timer does not have to overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receive error or data loss may result. To avoid this issue, verify that no receptions are in progress before changing the system clock.

This function implements an operation of the baud rate set feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsBaudRate" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
 // Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_BaudRateSet(MY_SPI_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
clockFrequency	Clock frequency
baudrate	Baud rate value

Function

void PLIB_SPI_BaudRateSet(SPI_MODULE_ID index, uint32_t clockFrequency,
uint32_t baudRate)

PLIB_SPI_ClockPolaritySelect Function

Enables clock polarity.

File

plib_spi.h

C

```
void PLIB_SPI_ClockPolaritySelect(SPI_MODULE_ID index, SPI_CLOCK_POLARITY polarity);
```

Returns

None.

Description

This function enables clock polarity.

Remarks

This function implements an operation of the clock polarity feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsClockPolarity" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_ClockPolaritySelect(MY_SPI_INSTANCE, SPI_CLOCK_POLARITY_IDLE_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
polarity	One of the SPI_CLOCK_POLARITY enumeration values as the SPI clock polarity

Function

void PLIB_SPI_ClockPolaritySelect(SPI_MODULE_ID index,
SPI_CLOCK_POLARITY polarity)

PLIB_SPI_CommunicationWidthSelect Function

Selects the data width for the SPI communication.

File

plib_spi.h

C

```
void PLIB_SPI_CommunicationWidthSelect(SPI_MODULE_ID index, SPI_COMMUNICATION_WIDTH width);
```

Returns

None.

Description

This function selects the data width for the SPI communication.

Remarks

This function implements an operation of the communication width feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsCommunicationWidth" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_CommunicationWidthSelect(MY_SPI_INSTANCE, SPI_COMMUNICATION_WIDTH_8BITS);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
width	One of the SPI_COMMUNICATION_WIDTH enumeration values as the SPI buffer width

Function

void PLIB_SPI_CommunicationWidthSelect (SPI_MODULE_ID index,
SPI_COMMUNICATION_WIDTH width)

PLIB_SPI_Disable Function

Disables the SPI module.

File

plib_spi.h

C

```
void PLIB_SPI_Disable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function disables the SPI module.

Remarks

This function implements an operation of the enable control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsEnableControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_Disable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_Disable (SPI_MODULE_ID index)

PLIB_SPI_Enable Function

Enables the SPI module.

File

plib_spi.h

C

```
void PLIB_SPI_Enable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SPI module.

Remarks

The SCKx, SDOx, SDIx and SSx pins must be assigned to available RPn pins before use.

This function implements an operation of the enable control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsEnableControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_Enable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_Enable(SPI_MODULE_ID index)

PLIB_SPI_ErrorInterruptDisable Function

Enables SPI error interrupts.

File

plib_spi.h

C

```
void PLIB_SPI_ErrorInterruptDisable(SPI_MODULE_ID index, SPI_ERROR_INTERRUPT error);
```

Returns

None.

Description

This function enables SPI error interrupts.

Remarks

This function implements an operation of the error interrupt control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsErrorInterruptControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_ErrorInterruptDisable (MY_SPI_INSTANCE, SPI_ERROR_INTERRUPT_FRAME_ERROR_OVERFLOW);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
error	One of the SPI_ERROR_INTERRUPT enumeration values as the SPI interrupt error

Function

void PLIB_SPI_ErrorInterruptDisable (SPI_MODULE_ID index,
SPI_ERROR_INTERRUPT error)

PLIB_SPI_ErrorInterruptEnable Function

Enables SPI error interrupts

File

plib_spi.h

C

```
void PLIB_SPI_ErrorInterruptEnable(SPI_MODULE_ID index, SPI_ERROR_INTERRUPT error);
```

Returns

None.

Description

This function enables SPI error interrupts.

Remarks

This function implements an operation of the error interrupt control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsErrorInterruptControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_ErrorInterruptEnable (MY_SPI_INSTANCE, SPI_ERROR_INTERRUPT_FRAME_ERROR_OVERFLOW);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
type	One of the SPI_ERROR_INTERRUPT enumeration values as the SPI interrupt error

Function

void PLIB_SPI_ErrorInterruptEnable (SPI_MODULE_ID index,
SPI_ERROR_INTERRUPT error)

PLIB_SPI_FIFOCountGet Function

Reads the SPI Buffer Element Count bits for either receive or transmit.

File

plib_spi.h

C

```
uint8_t PLIB_SPI_FIFOCountGet(SPI_MODULE_ID index, SPI_FIFO_TYPE type);
```

Returns

CountValue - Buffer element count bits

Description

This function reads the number of SPI transfers pending for Master mode and the number of unread SPI transfers for Slave mode.

Remarks

Valid in Enhanced Buffer mode.

This function implements an operation of the FIFO control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFIFOCount" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
uint8_t count = PLIB_SPI_FIFOCountGet(MY_SPI_INSTANCE,SPI_FIFO_TYPE_TRANSMIT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
type	One of the SPI_FIFO_TYPE enumeration values

Function

uint8_t PLIB_SPI_FIFOCountGet (SPI_MODULE_ID index, SPI_FIFO_TYPE type)

PLIB_SPI_FIFODisable Function

Disables the SPI enhanced buffer.

File

plib_spi.h

C

```
void PLIB_SPI_FIFODisable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function disables the SPI enhanced buffer.

Remarks

Enables the legacy standard single buffer mode.

This function implements an operation of the FIFO control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFIFOControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FIFODisable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_FIFODisable (SPI_MODULE_ID index)

PLIB_SPI_FIFOEnable Function

Enables the SPI enhanced buffer.

File

plib_spi.h

C

```
void PLIB_SPI_FIFOEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SPI enhanced buffer.

Remarks

This enables the enhanced buffer mode.

This function implements an operation of the FIFO control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFIFOControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FIFOEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_FIFOEnable (SPI_MODULE_ID index)

PLIB_SPI_FIFOInterruptModeSelect Function

Selects the SPI buffer interrupt mode.

File

plib_spi.h

C

```
void PLIB_SPI_FIFOInterruptModeSelect(SPI_MODULE_ID index, SPI_FIFO_INTERRUPT mode);
```

Returns

None.

Description

This function selects the SPI buffer interrupt mode from SPI_FIFO_INTERRUPT.

Remarks

Valid in Enhanced Buffer mode.

This function implements an operation of the FIFO interrupt feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFIFOInterruptMode" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FIFOInterruptModeSelect(MY_SPI_INSTANCE,
                SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_NOT_FULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the SPI_FIFO_INTERRUPT enumeration values as the SPI buffer interrupt mode

Function

void PLIB_SPI_FIFOInterruptModeSelect (SPI_MODULE_ID index,
SPI_FIFO_INTERRUPT mode)

PLIB_SPI_FIFOShiftRegisterIsEmpty Function

Returns the current status of the SPI shift register.

File

plib_spi.h

C

```
bool PLIB_SPI_FIFOShiftRegisterIsEmpty(SPI_MODULE_ID index);
```

Returns

- true - SPI shift register is empty and ready to send or receive
- false - SPI shift register is not empty

Description

This function returns the current status of the SPI shift register.

Remarks

Valid in Enhanced Buffer mode.

This function implements an operation of the FIFO status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool Status = PLIB_SPI_FIFOShiftRegisterIsEmpty(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_FIFOShiftRegisterIsEmpty (SPI_MODULE_ID index)

PLIB_SPI_InputSamplePhaseSelect Function

Selects the SPI data input sample phase.

File

plib_spi.h

C

```
void PLIB_SPI_InputSamplePhaseSelect(SPI_MODULE_ID index, SPI_INPUT_SAMPLING_PHASE phase);
```

Returns

None.

Description

This function selects the input sampling phase in Master mode.

Remarks

This function implements an operation of the input sample phase feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsInputSamplePhase" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_InputSamplePhaseSelect(MY_SPI_INSTANCE, SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
phase	One of the SPI_INPUT_SAMPLING_PHASE as the SPI input sampling phase

Function

void PLIB_SPI_InputSamplePhaseSelect(SPI_MODULE_ID index,
SPI_INPUT_SAMPLING_PHASE phase)

PLIB_SPI_IsBusy Function

Returns the current SPI module activity status.

File

plib_spi.h

C

```
bool PLIB_SPI_IsBusy(SPI_MODULE_ID index);
```

Returns

- true - SPI module is currently busy with some transactions
- false - SPI module is currently idle

Description

This function returns the current SPI module activity status.

Remarks

This function implements an operation of the bus status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsBusStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool status = PLIB_SPI_IsBusy(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_IsBusy (SPI_MODULE_ID index)

PLIB_SPI_MasterEnable Function

Enables the SPI in Master mode.

File

plib_spi.h

C

```
void PLIB_SPI_MasterEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SPI in Master mode.

Remarks

This function implements an operation of the master enable control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsMasterControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_MasterEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_MasterEnable(SPI_MODULE_ID index)

PLIB_SPI_OutputDataPhaseSelect Function

Selects serial output data change.

File

plib_spi.h

C

```
void PLIB_SPI_OutputDataPhaseSelect(SPI_MODULE_ID index, SPI_OUTPUT_DATA_PHASE phase);
```

Returns

None.

Description

This function selects serial output data change.

Remarks

This function implements an operation of the output data phase feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsOutputDataPhase" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_OutputDataPhaseSelect(MY_SPI_INSTANCE, SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	One of the SPI_OUTPUT_DATA_PHASE enumeration values as the SPI serial output data change

Function

void PLIB_SPI_OutputDataPhaseSelect (SPI_MODULE_ID index,
SPI_OUTPUT_DATA_PHASE data)

PLIB_SPI_PinDisable Function

Enables the selected SPI pins.

File

plib_spi.h

C

```
void PLIB_SPI_PinDisable(SPI_MODULE_ID index, SPI_PIN pin);
```

Returns

None.

Description

This function enables the selected SPI pins.

Remarks

This function implements an operation of the pin control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsPinControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_PinDisable(MY_SPI_INSTANCE, SPI_PIN_SLAVE_SELECT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pin	One of the SPI_PIN enumeration values as the SPI pin

Function

void PLIB_SPI_PinDisable (SPI_MODULE_ID index, SPI_PIN pin)

PLIB_SPI_PinEnable Function

Enables the selected SPI pins.

File

plib_spi.h

C

```
void PLIB_SPI_PinEnable(SPI_MODULE_ID index, SPI_PIN pin);
```

Returns

None.

Description

This function enables the selected SPI pins.

Remarks

This function implements an operation of the pin control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsPinControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_PinEnable(MY_SPI_INSTANCE, SPI_PIN_SLAVE_SELECT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pin	One of the SPI_PIN enumeration values as the SPI pin

Function

void PLIB_SPI_PinEnable (SPI_MODULE_ID index,SPI_PIN pin)

PLIB_SPI_ReadDataIsSignExtended Function

Returns the current status of the receive (RX) FIFO sign-extended data.

File

plib_spi.h

C

```
bool PLIB_SPI_ReadDataIsSignExtended(SPI_MODULE_ID index);
```

Returns

- true - Data from RX FIFO is sign-extended
- false - Data from RX FIFO is not sign-extended

Description

This function returns the current status of the receive (RX) FIFO sign-extended data.

Remarks

This function implements an operation of the data sign feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsReadDataSignStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool Status = PLIB_SPI_ReadDataIsSignExtended(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_ReadDataIsSignExtended(SPI_MODULE_ID index)

PLIB_SPI_SlaveEnable Function

Enables the SPI in Slave mode.

File

plib_spi.h

C

```
void PLIB_SPI_SlaveEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SPI in Slave mode.

Remarks

This function implements an operation of the master enable control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsMasterControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_SlaveEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_SlaveEnable(SPI_MODULE_ID index)

PLIB_SPI_SlaveSelectDisable Function

Disables Master mode slave select.

File

plib_spi.h

C

```
void PLIB_SPI_SlaveSelectDisable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function disables Master mode slave select.

Remarks

This feature does not support Framed SPI mode.

This function implements an operation of the slave select feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsSlaveSelectControl" in your application to automatically determine whether this feature is available.

To disable Slave mode slave select pin (SSEN), PLIB_SPI_PinDisable API can be used.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_SlaveSelectDisable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_SlaveSelectDisable(SPI_MODULE_ID index)

PLIB_SPI_SlaveSelectEnable Function

Enables Master mode slave select.

File

plib_spi.h

C

```
void PLIB_SPI_SlaveSelectEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables Master mode slave select.

Remarks

This feature does not support Framed SPI mode.

This function implements an operation of the Master mode slave select feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsSlaveSelectControl" in your application to automatically determine whether this feature is available.

To enable Slave mode slave select pin (SSEN), PLIB_SPI_PinEnable API can be used.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_SlaveSelectEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_SlaveSelectEnable(SPI_MODULE_ID index)

PLIB_SPI_StopInIdleDisable Function

Continues module operation when the device enters Idle mode.

File

plib_spi.h

C

```
void PLIB_SPI_StopInIdleDisable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function sets up the SPI module such that module operation is continued when the device enters Idle mode.

Remarks

This function implements an operation of the stop in idle control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsStopInIdleControl" in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_StopInIdleDisable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_StopInIdleDisable (SPI_MODULE_ID index)

PLIB_SPI_StopInIdleEnable Function

Discontinues module operation when the device enters Idle mode.

File

plib_spi.h

C

```
void PLIB_SPI_StopInIdleEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function sets up the SPI module such that module operation is disabled when the device enters Idle mode.

Remarks

This function implements an operation of the stop in idle control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsStopInIdleControl" in your application to automatically determine if this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_StopInIdleEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_StopInIdleEnable (SPI_MODULE_ID index)

b) Data Transfer Functions

PLIB_SPI_BufferClear Function

Clears the SPI buffer.

File

plib_spi.h

C

```
void PLIB_SPI_BufferClear(SPI_MODULE_ID index);
```

Returns

None.

Description

This function clears the SPI buffer.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsBuffer" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
 // Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_BufferClear(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_BufferClear (SPI_MODULE_ID index)

PLIB_SPI_BufferRead Function

Returns the SPI buffer value.

File

plib_spi.h

C

```
uint8_t PLIB_SPI_BufferRead(SPI_MODULE_ID index);
```

Returns

Reads the SPI buffer.

Description

This function returns the SPI buffer value.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsBuffer" in your application to automatically determine whether this feature is

available.

Preconditions

None.

Example

```
 // Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
uint8_t bufferValue = PLIB_SPI_BufferRead(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SPI_BufferRead (SPI_MODULE_ID index)

PLIB_SPI_BufferAddressGet Function

Returns the address of the SPIxBUF (Transmit(SPIxTXB) and Receive (SPIxRXB)) register.

File

plib_spi.h

C

```
void* PLIB_SPI_BufferAddressGet(SPI_MODULE_ID index);
```

Returns

The address of the SPIxBUF register

Description

This function returns the address of the SPIxBUF (Transmit(SPIxTXB) and Receive (SPIxRXB)) register.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_BufferAddressGet(SPI_MODULE_ID index)

PLIB_SPI_BufferRead16bit Function

Returns 16-bit SPI buffer value.

File

plib_spi.h

C

```
uint16_t PLIB_SPI_BufferRead16bit(SPI_MODULE_ID index);
```

Returns

Returns the SPI 16-bit buffer value.

Description

This function returns 16-bit SPI buffer value.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_Exists16bitBuffer" in your application to automatically determine whether this feature is available.

Preconditions

SPI 16-bit wide communication must be selected using PLIB_SPI_CommunicationWidthSelect.

Example

```
#define MY_SPI_INSTANCE SPI_ID_1

uint16_t bufferValue = PLIB_SPI_BufferRead16bit( MY_SPI_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint16_t PLIB_SPI_BufferRead16bit (SPI_MODULE_ID index)

PLIB_SPI_BufferRead32bit Function

Returns 32-bit SPI buffer value.

File

plib_spi.h

C

```
uint32_t PLIB_SPI_BufferRead32bit(SPI_MODULE_ID index);
```

Returns

Returns the SPI 32-bit buffer value.

Description

This function returns 32-bit SPI buffer value.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_Exists32bitBuffer" in your application to automatically determine whether this feature is available.

Preconditions

SPI 32-bit wide communication must be selected using PLIB_SPI_CommunicationWidthSelect.

Example

```
#define MY_SPI_INSTANCE SPI_ID_1

uint32_t bufferValue = PLIB_SPI_BufferRead32bit( MY_SPI_INSTANCE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SPI_BufferRead32bit (SPI_MODULE_ID index)

PLIB_SPI_BufferWrite Function

Write the data to the SPI buffer.

File

plib_spi.h

C

```
void PLIB_SPI_BufferWrite(SPI_MODULE_ID index, uint8_t data);
```

Returns

None.

Description

This function writes data to the SPI buffer.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsBuffer" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_BufferWrite ( MY_SPI_INSTANCE, 0xFF );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to written to the SPI buffer

Function

void PLIB_SPI_BufferWrite (SPI_MODULE_ID index , uint8_t data)

PLIB_SPI_BufferWrite16bit Function

Writes 16-bit data to the SPI buffer.

File

plib_spi.h

C

```
void PLIB_SPI_BufferWrite16bit(SPI_MODULE_ID index, uint16_t data);
```

Returns

None.

Description

This function writes 16-bit data to the SPI buffer.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_Exists16bitBuffer" in your application to automatically determine whether this feature is available.

Preconditions

SPI 16-bit wide communication must be selected using PLIB_SPI_CommunicationWidthSelect.

Example

```
#define MY_SPI_INSTANCE SPI_ID_1

PLIB_SPI_BufferWrite16bit ( MY_SPI_INSTANCE, 0x55AA );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	16-bit data to be written to the SPI buffer

Function

void PLIB_SPI_BufferWrite16bit (SPI_MODULE_ID index , uint16_t data)

PLIB_SPI_BufferWrite32bit Function

Write 32-bit data to the SPI buffer.

File

plib_spi.h

C

```
void PLIB_SPI_BufferWrite32bit(SPI_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function writes 32-bit data to the SPI buffer.

Remarks

This function implements an operation of the buffer control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_Exists32bitBuffer" in your application to automatically determine whether this feature is available.

Preconditions

SPI 32-bit wide communication must be selected using PLIB_SPI_CommunicationWidthSelect.

Example

```
#define MY_SPI_INSTANCE SPI_ID_1

PLIB_SPI_BufferWrite ( MY_SPI_INSTANCE, 0x55AA55AA );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	32-bit data to be written to the SPI buffer

Function

void PLIB_SPI_BufferWrite32bit (SPI_MODULE_ID index , uint32_t data)

c) Framed Mode Functions

PLIB_SPI_FramedCommunicationDisable Function

Disables framed SPI support.

File

plib_spi.h

C

```
void PLIB_SPI_FramedCommunicationDisable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function disables framed SPI support.

Remarks

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFramedCommunication" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FramedCommunicationDisable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_FramedCommunicationDisable (SPI_MODULE_ID index)

PLIB_SPI_FramedCommunicationEnable Function

Enables framed SPI support.

File

plib_spi.h

C

```
void PLIB_SPI_FramedCommunicationEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables framed SPI support.

Remarks

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFramedCommunication" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FramedCommunicationEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_FramedCommunicationEnable (SPI_MODULE_ID index)

PLIB_SPI_FrameErrorStatusGet Function

Returns the current status of the SPI frame error.

File

plib_spi.h

C

```
bool PLIB_SPI_FrameErrorStatusGet(SPI_MODULE_ID index);
```

Returns

- true - Frame error detected
- false - No frame error detected

Description

This function returns the current status of the SPI frame error.

Remarks

Valid only if Frame mode is enabled.

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFrameErrorStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool status = PLIB_SPI_FrameErrorStatusGet(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_FrameErrorStatusGet (SPI_MODULE_ID index)

PLIB_SPI_FrameErrorStatusClear Function

Clears the SPI frame error flag.

File

plib_spi.h

C

```
void PLIB_SPI_FrameErrorStatusClear(SPI_MODULE_ID index);
```

Returns

None.

Description

This function clears the SPI frame error flag.

Remarks

This function implements an operation of the frame error status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "ExistsFrameErrorStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FrameErrorStatusClear(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_FrameErrorStatusClear(SPI_MODULE_ID index)

PLIB_SPI_FrameSyncPulseCounterSelect Function

Selects at which character the SPI frame sync pulse is generated.

File

plib_spi.h

C

```
void PLIB_SPI_FrameSyncPulseCounterSelect(SPI_MODULE_ID index, SPI_FRAME_SYNC_PULSE pulse);
```

Returns

None.

Description

This function selects at which character the SPI frame sync pulse is generated.

Remarks

This is valid only when PLIB_SPI_FramedCommunicationEnable is enabled.

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFrameSyncPulseCounter" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FrameSyncPulseCounterSelect(MY_SPI_INSTANCE,
                                     SPI_FRAME_SYNC_PULSE_ON_EVERY_32_DATA_CHARACTER );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pulse	One of the SPI_FRAME_SYNC_PULSE enumeration values as the SPI frame sync pulse count

Function

void PLIB_SPI_FrameSyncPulseCounterSelect (SPI_MODULE_ID index,
SPI_FRAME_SYNC_PULSE pulse)

PLIB_SPI_FrameSyncPulseDirectionSelect Function

Selects the frame sync pulse direction.

File

plib_spi.h

C

```
void PLIB_SPI_FrameSyncPulseDirectionSelect(SPI_MODULE_ID index, SPI_FRAME_PULSE_DIRECTION direction);
```

Returns

None.

Description

This function selects the frame sync pulse direction.

Remarks

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFrameSyncPulseDirection" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FrameSyncPulseDirectionSelect(MY_SPI_INSTANCE, SPI_FRAME_PULSE_DIRECTION_INPUT );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
direction	One of the SPI_FRAME_PULSE_DIRECTION enumeration values as the SPI frame sync pulse polarity

Function

void PLIB_SPI_FrameSyncPulseDirectionSelect (SPI_MODULE_ID index,
SPI_FRAME_PULSE_DIRECTION direction)

PLIB_SPI_FrameSyncPulseEdgeSelect Function

Selects the frame sync pulse edge.

File

plib_spi.h

C

```
void PLIB_SPI_FrameSyncPulseEdgeSelect(SPI_MODULE_ID index, SPI_FRAME_PULSE_EDGE edge);
```

Returns

None.

Description

This function selects the frame sync pulse edge.

Remarks

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFrameSyncPulseEdge" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FrameSyncPulseEdgeSelect(MY_SPI_INSTANCE,
                SPI_FRAME_PULSE_EDGE_COINCIDES_FIRST_BIT_CLOCK);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
edge	One of the SPI_FRAME_PULSE_EDGE enumeration values as the SPI frame sync pulse edge

Function

void PLIB_SPI_FrameSyncPulseEdgeSelect (SPI_MODULE_ID index,
SPI_FRAME_PULSE_EDGE edge)

PLIB_SPI_FrameSyncPulsePolaritySelect Function

Selects the frame sync pulse polarity.

File

plib_spi.h

C

```
void PLIB_SPI_FrameSyncPulsePolaritySelect(SPI_MODULE_ID index, SPI_FRAME_PULSE_POLARITY polarity);
```

Returns

None.

Description

This function selects the frame sync pulse polarity.

Remarks

Available only for Frame mode.

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFrameSyncPulsePolarity" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FrameSyncPulsePolaritySelect(MY_SPI_INSTANCE, SPI_FRAME_PULSE_POLARITY_ACTIVE_HIGH );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
polarity	One of the SPI_FRAME_PULSE_POLARITY enumeration values as the SPI frame sync pulse polarity

Function

void PLIB_SPI_FrameSyncPulsePolaritySelect (SPI_MODULE_ID index,
SPI_FRAME_PULSE_POLARITY polarity)

PLIB_SPI_FrameSyncPulseWidthSelect Function

Sets the frame sync pulse width.

File

plib_spi.h

C

```
void PLIB_SPI_FrameSyncPulseWidthSelect(SPI_MODULE_ID index, SPI_FRAME_PULSE_WIDTH width);
```

Returns

None.

Description

This function sets the frame sync pulse width.

Remarks

Length of the word is dependent on the communication mode.

This function implements an operation of the framed communication feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsFrameSyncPulseWidth" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_FrameSyncPulseWidthSelect (MY_SPI_INSTANCE,SPI_FRAME_PULSE_WIDTH_ONE_WORD_LENGTH);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
edge	One of the SPI_FRAME_PULSE_WIDTH enumeration values as the SPI frame sync pulse width.

Function

void PLIB_SPI_FrameSyncPulseWidthSelect (SPI_MODULE_ID index,
SPI_FRAME_PULSE_WIDTH width)

d) Audio Mode Functions

PLIB_SPI_AudioCommunicationWidthSelect Function

Selects the data width for the SPI audio communication.

File

plib_spi.h

C

```
void PLIB_SPI_AudioCommunicationWidthSelect(SPI_MODULE_ID index, SPI_AUDIO_COMMUNICATION_WIDTH mode);
```

Returns

None.

Description

This function selects the data width for the SPI audio communication.

Remarks

This mode is available only when PLIB_SPI_AudioProtocolEnable is enabled.

This function implements an operation of the audio communication width feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioCommunicationWidth" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
```

```
PLIB_SPI_AudioCommunicationWidthSelect(MY_SPI_INSTANCE,
                        SPI_AUDIO_COMMUNICATION_32DATA_32FIFO_32CHANNEL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
width	One of the SPI_AUDIO_COMMUNICATION_WIDTH enumeration values as the SPI buffer width

Function

void PLIB_SPI_AudioCommunicationWidthSelect (SPI_MODULE_ID index,
SPI_AUDIO_COMMUNICATION_WIDTH width)

PLIB_SPI_AudioErrorDisable Function

Disables the SPI error.

File

plib_spi.h

C

```
void PLIB_SPI_AudioErrorDisable(SPI_MODULE_ID index, SPI_AUDIO_ERROR error);
```

Returns

None.

Description

This function disables the SPI error.

Remarks

This function implements an operation of the audio error control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioErrorControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_AudioErrorDisable (MY_SPI_INSTANCE, SPI_AUDIO_ERROR_RECEIVE_OVERFLOW);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
error	One of the SPI_AUDIO_ERROR enumeration values as the SPI error

Function

void PLIB_SPI_AudioErrorDisable (SPI_MODULE_ID index, SPI_AUDIO_ERROR error)

PLIB_SPI_AudioErrorEnable Function

Enables the SPI error.

File

plib_spi.h

C

```
void PLIB_SPI_AudioErrorEnable(SPI_MODULE_ID index, SPI_AUDIO_ERROR error);
```

Returns

None.

Description

This function enables the SPI error.

Remarks

This function implements an operation of the audio error control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioErrorControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_AudioErrorEnable (MY_SPI_INSTANCE, SPI_AUDIO_ERROR_RECEIVE_OVERFLOW);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
error	One of the SPI_AUDIO_ERROR enumeration values as the SPI error

Function

void PLIB_SPI_AudioErrorEnable (SPI_MODULE_ID index, SPI_Audio_ERROR error)

PLIB_SPI_AudioProtocolDisable Function

Audio protocol is disabled.

File

plib_spi.h

C

```
void PLIB_SPI_AudioProtocolDisable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function disables the audio protocol.

Remarks

This function implements an operation of the audio protocol control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioProtocolControl" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_AudioProtocolDisable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_AudioProtocolDisable (SPI_MODULE_ID index)

PLIB_SPI_AudioProtocolEnable Function

Audio protocol is enabled.

File

plib_spi.h

C

```
void PLIB_SPI_AudioProtocolEnable(SPI_MODULE_ID index);
```

Returns

None.

Description

This function enables the audio protocol.

Remarks

This function implements an operation of the audio protocol control feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioProtocolControl" in your application to automatically determine whether this feature is available.

Preconditions

Disable the SPI module by calling PLIB_SPI_Disable.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_AudioProtocolEnable(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_AudioProtocolEnable (SPI_MODULE_ID index)

PLIB_SPI_AudioProtocolModeSelect Function

Selects the Audio Protocol mode.

File

plib_spi.h

C

```
void PLIB_SPI_AudioProtocolModeSelect(SPI_MODULE_ID index, SPI_AUDIO_PROTOCOL mode);
```

Returns

None.

Description

This function selects the Audio Protocol mode.

Remarks

This function implements an operation of the audio protocol mode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioProtocolMode" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_AudioProtocolModeSelect(MY_SPI_INSTANCE,SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the SPI_AUDIO_PROTOCOL enumeration values as the audio protocol

Function

void PLIB_SPI_AudioProtocolModeSelect(SPI_MODULE_ID index,
SPI_AUDIO_PROTOCOL mode)

PLIB_SPI_AudioTransmitModeSelect Function

Selects the transmit audio data format.

File

plib_spi.h

C

```
void PLIB_SPI_AudioTransmitModeSelect(SPI_MODULE_ID index, SPI_AUDIO_TRANSMIT_MODE mode);
```

Returns

None.

Description

This function selects the transmit audio data format.

Remarks

This function implements an operation of the audio transmit mode feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsAudioTransmitMode" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_AudioTransmitModeSelect (MY_SPI_INSTANCE, SPI_AUDIO_TRANSMIT_MONO);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	One of the SPI_TRANSMIT_AUDIO_MODE enumeration values as the transmit audio format

Function

void PLIB_SPI_AudioTransmitModeSelect (SPI_MODULE_ID index,
SPI_AUDIO_TRANSMIT_MODE mode)

e) Transmitter Functions

PLIB_SPI_TransmitBufferIsEmpty Function

Returns the current status of the transmit buffer.

File

plib_spi.h

C

```
bool PLIB_SPI_TransmitBufferIsEmpty(SPI_MODULE_ID index);
```

Returns

- true - Transmit buffer is empty
- false - Transmit buffer is not empty

Description

This function returns the current status of the transmit buffer.

Remarks

This function implements an operation of the transmit buffer empty status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsTransmitBufferEmptyStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool Status = PLIB_SPI_TransmitBufferIsEmpty(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_TransmitBufferIsEmpty (SPI_MODULE_ID index)

PLIB_SPI_TransmitBufferIsFull Function

Returns the current transmit buffer status of the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_TransmitBufferIsFull(SPI_MODULE_ID index);
```

Returns

- true - Transmit not yet started, transmit buffer is full
- false - Transmit started, transmit buffer is empty/not full

Description

This function returns the current transmit buffer status of the SPI module.

Remarks

In Standard Buffer mode - automatically set in hardware when SPI buffer writes occur, loading the transmit buffer. Automatically cleared in hardware when the SPI module transfers data from the transmit buffer to the shift register.

In Enhanced Buffer mode - automatically set in hardware when SPI buffer writes occur, loading the last available buffer. Automatically cleared in hardware when the buffer is available for writing.

This function implements an operation of the transmit buffer status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsTransmitBufferFullStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool buffullstate = PLIB_SPI_TransmitBufferIsFull (MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_TransmitBufferIsFull (SPI_MODULE_ID index)

PLIB_SPI_TransmitUnderRunStatusGet Function

Returns the current status of the transmit underrun.

File

plib_spi.h

C

```
bool PLIB_SPI_TransmitUnderRunStatusGet(SPI_MODULE_ID index);
```

Returns

- true - Transmit buffer has encountered an underrun condition
- false - Transmit buffer run has not encountered an underrun condition

Description

This function returns the current status of the transmit underrun.

Remarks

Valid in Framed Sync mode.

This function implements an operation of the transmit underrun status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsTransmitUnderRunStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool Status = PLIB_SPI_TransmitUnderRunStatusGet(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_TransmitUnderRunStatusGet(SPI_MODULE_ID index)

PLIB_SPI_TransmitUnderRunStatusClear Function

Clears the SPI transmit underrun flag.

File

plib_spi.h

C

```
void PLIB_SPI_TransmitUnderRunStatusClear(SPI_MODULE_ID index);
```

Returns

None.

Description

This function clears the SPI transmit underrun flag.

Remarks

This function implements an operation of the transmit underrun status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "ExistsTransmitUnderRunStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_TransmitUnderRunStatusClear(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_TransmitUnderRunStatusClear(SPI_MODULE_ID index)

f) Receiver Functions

PLIB_SPI_ReceiverBufferIsFull Function

Returns the current status of the SPI receive buffer.

File

plib_spi.h

C

```
bool PLIB_SPI_ReceiverBufferIsFull(SPI_MODULE_ID index);
```

Returns

Receiver Buffer Full Status:

- true - Receive complete, receive buffer is full
- false - Receive is not complete, receive buffer is empty

Description

This function returns the current status of the SPI receive buffer.

Remarks

In Standard Buffer mode - automatically set in hardware when the SPI module transfers data from the shift register to the receive buffer. Automatically cleared in hardware when the core reads the SPI buffer, read in the receive buffer.

In Enhanced Buffer mode - automatically set in hardware when the SPI module transfers data from the shift register to the receive buffer, filling the last unread buffer. Automatically cleared in hardware when a buffer is available for a transfer from the shift register.

This function implements an operation of the receiver buffer status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsReceiveBufferStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
```

```
// application developer.
bool receivefullstate = PLIB_SPI_ReceiverBufferIsFull (MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_ReceiverBufferIsFull (SPI_MODULE_ID index)

PLIB_SPI_ReceiverFIFOIsEmpty Function

Returns the current status of the SPI receive FIFO.

File

plib_spi.h

C

```
bool PLIB_SPI_ReceiverFIFOIsEmpty(SPI_MODULE_ID index);
```

Returns

- true - Receive FIFO is empty
- false - Receive FIFO is not empty

Description

This function returns the current status of the SPI receive FIFO.

Remarks

Valid in Enhanced Buffer mode.

This function implements an operation of the FIFO status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsReceiveFIFOStatus" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool fifostate = PLIB_SPI_ReceiverFIFOIsEmpty (MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_ReceiverFIFOIsEmpty (SPI_MODULE_ID index)

PLIB_SPI_ReceiverHasOverflowed Function

Returns the current status of the SPI receiver overflow.

File

plib_spi.h

C

```
bool PLIB_SPI_ReceiverHasOverflowed(SPI_MODULE_ID index);
```

Returns

SPI receiver overflow status:

- true - A new byte/word is completely received and discarded. The user software has not read the previous data in the SPI buffer register.
- false - No Overflow has occurred

Description

This function returns the current status of the SPI receiver overflow.

Remarks

This function implements an operation of the receiver overflow status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsReceiverOverflow" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
bool overflowstate = PLIB_SPI_ReceiverHasOverflowed(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SPI_ReceiverHasOverflowed (SPI_MODULE_ID index)

PLIB_SPI_ReceiverOverflowClear Function

Clears the SPI receive overflow flag.

File

plib_spi.h

C

```
void PLIB_SPI_ReceiverOverflowClear(SPI_MODULE_ID index);
```

Returns

None.

Description

This function clears the SPI receive overflow flag.

Remarks

This function implements an operation of the receiver overflow status feature. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use "PLIB_SPI_ExistsReceiverOverflow" in your application to automatically determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_SPI_INSTANCE, is the SPI instance selected for use by the
// application developer.
PLIB_SPI_ReceiverOverflowClear(MY_SPI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SPI_ReceiverOverflowClear(SPI_MODULE_ID index)

g) Feature Existence Functions

PLIB_SPI_ExistsAudioCommunicationWidth Function

Identifies whether the AudioCommunicationWidth feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsAudioCommunicationWidth(SPI_MODULE_ID index);
```

Returns

- true - The AudioCommunicationWidth feature is supported on the device
- false - The AudioCommunicationWidth feature is not supported on the device

Description

This function identifies whether the AudioCommunicationWidth feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_AudioCommunicationWidthSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsAudioCommunicationWidth(SPI_MODULE_ID index)

PLIB_SPI_ExistsAudioErrorControl Function

Identifies whether the AudioErrorControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsAudioErrorControl(SPI_MODULE_ID index);
```

Returns

- true - The AudioErrorControl feature is supported on the device
- false - The AudioErrorControl feature is not supported on the device

Description

This function identifies whether the AudioErrorControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_AudioErrorEnable
- PLIB_SPI_AudioErrorDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsAudioErrorControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsAudioProtocolControl Function

Identifies whether the AudioProtocolControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsAudioProtocolControl(SPI_MODULE_ID index);
```

Returns

- true - The AudioProtocolControl feature is supported on the device
- false - The AudioProtocolControl feature is not supported on the device

Description

This function identifies whether the AudioProtocolControl feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_AudioProtocolEnable
- PLIB_SPI_AudioProtocolDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsAudioProtocolControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsAudioProtocolMode Function

Identifies whether the AudioProtocolMode feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsAudioProtocolMode(SPI_MODULE_ID index);
```

Returns

- true - The AudioProtocolMode feature is supported on the device
- false - The AudioProtocolMode feature is not supported on the device

Description

This function identifies whether the AudioProtocolMode feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_AudioProtocolModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsAudioProtocolMode(SPI_MODULE_ID index)

PLIB_SPI_ExistsAudioTransmitMode Function

Identifies whether the AudioTransmitMode feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsAudioTransmitMode(SPI_MODULE_ID index);
```

Returns

- true - The AudioTransmitMode feature is supported on the device
- false - The AudioTransmitMode feature is not supported on the device

Description

This function identifies whether the AudioTransmitMode feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_AudioTransmitModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsAudioTransmitMode(SPI_MODULE_ID index)

PLIB_SPI_ExistsBaudRate Function

Identifies whether the BaudRate feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsBaudRate(SPI_MODULE_ID index);
```

Returns

- true - The BaudRate feature is supported on the device
- false - The BaudRate feature is not supported on the device

Description

This function identifies whether the BaudRate feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_BaudRateSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsBaudRate(SPI_MODULE_ID index)

PLIB_SPI_ExistsBaudRateClock Function

Identifies whether the BaudRateClock feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsBaudRateClock(SPI_MODULE_ID index);
```

Returns

- true - The BaudRateClock feature is supported on the device
- false - The BaudRateClock feature is not supported on the device

Description

This function identifies whether the BaudRateClock feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_BaudRateClockSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsBaudRateClock(SPI_MODULE_ID index)

PLIB_SPI_ExistsBuffer Function

Identifies whether the Buffer feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsBuffer(SPI_MODULE_ID index);
```

Returns

- true - The Buffer feature is supported on the device
- false - The Buffer feature is not supported on the device

Description

This function identifies whether the Buffer feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_BufferClear
- PLIB_SPI_BufferRead
- PLIB_SPI_BufferWrite
- PLIB_SPI_BufferAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsBuffer(SPI_MODULE_ID index)

PLIB_SPI_ExistsBusStatus Function

Identifies whether the BusStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsBusStatus(SPI_MODULE_ID index);
```

Returns

- true - The BusStatus feature is supported on the device
- false - The BusStatus feature is not supported on the device

Description

This function identifies whether the BusStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_IsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsBusStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsClockPolarity Function

Identifies whether the ClockPolarity feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsClockPolarity(SPI_MODULE_ID index);
```

Returns

- true - The ClockPolarity feature is supported on the device
- false - The ClockPolarity feature is not supported on the device

Description

This function identifies whether the ClockPolarity feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_ClockPolaritySelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsClockPolarity(SPI_MODULE_ID index)

PLIB_SPI_ExistsCommunicationWidth Function

Identifies whether the CommunicationWidth feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsCommunicationWidth(SPI_MODULE_ID index);
```

Returns

- true - The CommunicationWidth feature is supported on the device
- false - The CommunicationWidth feature is not supported on the device

Description

This function identifies whether the CommunicationWidth feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_CommunicationWidthSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsCommunicationWidth(SPI_MODULE_ID index)

PLIB_SPI_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsEnableControl(SPI_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_Enable
- PLIB_SPI_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsEnableControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsErrorInterruptControl Function

Identifies whether the ErrorInterruptControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsErrorInterruptControl(SPI_MODULE_ID index);
```

Returns

- true - The ErrorInterruptControl feature is supported on the device
- false - The ErrorInterruptControl feature is not supported on the device

Description

This function identifies whether the ErrorInterruptControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_ErrorInterruptEnable
- PLIB_SPI_ErrorInterruptDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsErrorInterruptControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsFIFOControl Function

Identifies whether the FIFOControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFIFOControl(SPI_MODULE_ID index);
```

Returns

- true - The FIFOControl feature is supported on the device
- false - The FIFOControl feature is not supported on the device

Description

This function identifies whether the FIFOControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_FIFOEnable
- PLIB_SPI_FIFODisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFIFOControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsFIFOCount Function

Identifies whether the FIFOCount feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFIFOCount(SPI_MODULE_ID index);
```

Returns

- true - The FIFOCount feature is supported on the device
- false - The FIFOCount feature is not supported on the device

Description

This function identifies whether the FIFOCount feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FIFOCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFIFOCount(SPI_MODULE_ID index)

PLIB_SPI_ExistsFIFOInterruptMode Function

Identifies whether the FIFOInterruptMode feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFIFOInterruptMode(SPI_MODULE_ID index);
```

Returns

- true - The FIFOInterruptMode feature is supported on the device
- false - The FIFOInterruptMode feature is not supported on the device

Description

This function identifies whether the FIFOInterruptMode feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FIFOInterruptModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFIFOInterruptMode(SPI_MODULE_ID index)

PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus Function

Identifies whether the FIFOShiftRegisterEmptyStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus(SPI_MODULE_ID index);
```

Returns

- true - The FIFOShiftRegisterEmptyStatus feature is supported on the device
- false - The FIFOShiftRegisterEmptyStatus feature is not supported on the device

Description

This function identifies whether the FIFOShiftRegisterEmptyStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FIFOShiftRegisterIsEmpty

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsFramedCommunication Function

Identifies whether the FramedCommunication feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFramedCommunication(SPI_MODULE_ID index);
```

Returns

- true - The FramedCommunication feature is supported on the device
- false - The FramedCommunication feature is not supported on the device

Description

This function identifies whether the FramedCommunication feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_FramedCommunicationEnable
- PLIB_SPI_FramedCommunicationDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFramedCommunication(SPI_MODULE_ID index)

PLIB_SPI_ExistsFrameErrorStatus Function

Identifies whether the FrameErrorStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFrameErrorStatus(SPI_MODULE_ID index);
```

Returns

- true - The FrameErrorStatus feature is supported on the device
- false - The FrameErrorStatus feature is not supported on the device

Description

This function identifies whether the FrameErrorStatus feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_FrameErrorStatusGet
- PLIB_SPI_FrameErrorStatusClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFrameErrorStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsFrameSyncPulseCounter Function

Identifies whether the FrameSyncPulseCounter feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFrameSyncPulseCounter(SPI_MODULE_ID index);
```

Returns

- true - The FrameSyncPulseCounter feature is supported on the device
- false - The FrameSyncPulseCounter feature is not supported on the device

Description

This function identifies whether the FrameSyncPulseCounter feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FrameSyncPulseCounterSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFrameSyncPulseCounter(SPI_MODULE_ID index)

PLIB_SPI_ExistsFrameSyncPulseDirection Function

Identifies whether the FrameSyncPulseDirection feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFrameSyncPulseDirection(SPI_MODULE_ID index);
```

Returns

- true - The FrameSyncPulseDirection feature is supported on the device
- false - The FrameSyncPulseDirection feature is not supported on the device

Description

This function identifies whether the FrameSyncPulseDirection feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FrameSyncPulseDirectionSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFrameSyncPulseDirection(SPI_MODULE_ID index)

PLIB_SPI_ExistsFrameSyncPulseEdge Function

Identifies whether the FrameSyncPulseEdge feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFrameSyncPulseEdge(SPI_MODULE_ID index);
```

Returns

- true - The FrameSyncPulseEdge feature is supported on the device
- false - The FrameSyncPulseEdge feature is not supported on the device

Description

This function identifies whether the FrameSyncPulseEdge feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FrameSyncPulseEdgeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFrameSyncPulseEdge(SPI_MODULE_ID index)

PLIB_SPI_ExistsFrameSyncPulsePolarity Function

Identifies whether the FrameSyncPulsePolarity feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFrameSyncPulsePolarity(SPI_MODULE_ID index);
```

Returns

- true - The FrameSyncPulsePolarity feature is supported on the device
- false - The FrameSyncPulsePolarity feature is not supported on the device

Description

This function identifies whether the FrameSyncPulsePolarity feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FrameSyncPulsePolaritySelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFrameSyncPulsePolarity(SPI_MODULE_ID index)

PLIB_SPI_ExistsFrameSyncPulseWidth Function

Identifies whether the FrameSyncPulseWidth feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsFrameSyncPulseWidth(SPI_MODULE_ID index);
```

Returns

- true - The FrameSyncPulseWidth feature is supported on the device
- false - The FrameSyncPulseWidth feature is not supported on the device

Description

This function identifies whether the FrameSyncPulseWidth feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_FrameSyncPulseWidthSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsFrameSyncPulseWidth(SPI_MODULE_ID index)

PLIB_SPI_ExistsInputSamplePhase Function

Identifies whether the InputSamplePhase feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsInputSamplePhase(SPI_MODULE_ID index);
```

Returns

- true - The InputSamplePhase feature is supported on the device
- false - The InputSamplePhase feature is not supported on the device

Description

This function identifies whether the InputSamplePhase feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_InputSamplePhaseSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsInputSamplePhase(SPI_MODULE_ID index)

PLIB_SPI_ExistsMasterControl Function

Identifies whether the MasterControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsMasterControl(SPI_MODULE_ID index);
```

Returns

- true - The MasterControl feature is supported on the device
- false - The MasterControl feature is not supported on the device

Description

This function identifies whether the MasterControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_MasterEnable
- PLIB_SPI_SlaveEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsMasterControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsOutputDataPhase Function

Identifies whether the OutputDataPhase feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsOutputDataPhase(SPI_MODULE_ID index);
```

Returns

- true - The OutputDataPhase feature is supported on the device
- false - The OutputDataPhase feature is not supported on the device

Description

This function identifies whether the OutputDataPhase feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_OutputDataPhaseSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsOutputDataPhase(SPI_MODULE_ID index)

PLIB_SPI_ExistsPinControl Function

Identifies whether the PinControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsPinControl(SPI_MODULE_ID index);
```

Returns

- true - The PinControl feature is supported on the device
- false - The PinControl feature is not supported on the device

Description

This function identifies whether the PinControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_PinEnable
- PLIB_SPI_PinDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsPinControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsReadDataSignStatus Function

Identifies whether the ReadDataSignStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsReadDataSignStatus(SPI_MODULE_ID index);
```

Returns

- true - The ReadDataSignStatus feature is supported on the device
- false - The ReadDataSignStatus feature is not supported on the device

Description

This function identifies whether the ReadDataSignStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_ReadDataIsSignExtended

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsReadDataSignStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsReceiveBufferStatus Function

Identifies whether the ReceiveBufferStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsReceiveBufferStatus(SPI_MODULE_ID index);
```

Returns

- true - The ReceiveBufferStatus feature is supported on the device
- false - The ReceiveBufferStatus feature is not supported on the device

Description

This function identifies whether the ReceiveBufferStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_ReceiverBufferIsFull

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsReceiveBufferStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsReceiveFIFOStatus Function

Identifies whether the ReceiveFIFOStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsReceiveFIFOStatus(SPI_MODULE_ID index);
```

Returns

- true - The ReceiveFIFOStatus feature is supported on the device
- false - The ReceiveFIFOStatus feature is not supported on the device

Description

This function identifies whether the ReceiveFIFOStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_ReceiverFIFOIsEmpty

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsReceiveFIFOStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsReceiverOverflow Function

Identifies whether the ReceiverOverflow feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsReceiverOverflow(SPI_MODULE_ID index);
```

Returns

- true - The ReceiverOverflow feature is supported on the device
- false - The ReceiverOverflow feature is not supported on the device

Description

This function identifies whether the ReceiverOverflow feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_ReceiverHasOverflowed
- PLIB_SPI_ReceiverOverflowClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsReceiverOverflow(SPI_MODULE_ID index)

PLIB_SPI_ExistsSlaveSelectControl Function

Identifies whether the SlaveSelectControl feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsSlaveSelectControl(SPI_MODULE_ID index);
```

Returns

- true - The SlaveSelectControl feature is supported on the device
- false - The SlaveSelectControl feature is not supported on the device

Description

This function identifies whether the SlaveSelectControl feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_SlaveSelectEnable
- PLIB_SPI_SlaveSelectDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsSlaveSelectControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsStopInIdleControl Function

Identifies whether the StopInIdle feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsStopInIdleControl(SPI_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_StopInIdleEnable
- PLIB_SPI_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsStopInIdleControl(SPI_MODULE_ID index)

PLIB_SPI_ExistsTransmitBufferEmptyStatus Function

Identifies whether the TransmitBufferEmptyStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsTransmitBufferEmptyStatus(SPI_MODULE_ID index);
```

Returns

- true - The TransmitBufferEmptyStatus feature is supported on the device
- false - The TransmitBufferEmptyStatus feature is not supported on the device

Description

This function identifies whether the TransmitBufferEmptyStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_TransmitBufferIsEmpty

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsTransmitBufferEmptyStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsTransmitBufferFullStatus Function

Identifies whether the TransmitBufferFullStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsTransmitBufferFullStatus(SPI_MODULE_ID index);
```

Returns

- true - The TransmitBufferFullStatus feature is supported on the device
- false - The TransmitBufferFullStatus feature is not supported on the device

Description

This function identifies whether the TransmitBufferFullStatus feature is available on the SPI module. When this function returns true, this function is supported on the device:

- PLIB_SPI_TransmitBufferIsFull

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsTransmitBufferFullStatus(SPI_MODULE_ID index)

PLIB_SPI_ExistsTransmitUnderRunStatus Function

Identifies whether the TransmitUnderRunStatus feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_ExistsTransmitUnderRunStatus(SPI_MODULE_ID index);
```

Returns

- true - The TransmitUnderRunStatus feature is supported on the device
- false - The TransmitUnderRunStatus feature is not supported on the device

Description

This function identifies whether the TransmitUnderRunStatus feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_TransmitUnderRunStatusGet
- PLIB_SPI_TransmitUnderRunStatusClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_ExistsTransmitUnderRunStatus(SPI_MODULE_ID index)

PLIB_SPI_Exists16bitBuffer Function

Identifies whether the Buffer16bit feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_Exists16bitBuffer(SPI_MODULE_ID index);
```

Returns

- true - The Buffer16bit feature is supported on the device
- false - The Buffer16bit feature is not supported on the device

Description

This function identifies whether the Buffer16bit feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_BufferRead16bit
- PLIB_SPI_BufferWrite16bit

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_Exists16bitBuffer(SPI_MODULE_ID index)

PLIB_SPI_Exists32bitBuffer Function

Identifies whether the Buffer32bit feature exists on the SPI module.

File

plib_spi.h

C

```
bool PLIB_SPI_Exists32bitBuffer(SPI_MODULE_ID index);
```

Returns

- true - The Buffer32bit feature is supported on the device
- false - The Buffer32bit feature is not supported on the device

Description

This function identifies whether the Buffer32bit feature is available on the SPI module. When this function returns true, these functions are supported on the device:

- PLIB_SPI_BufferRead32bit
- PLIB_SPI_BufferWrite32bit

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SPI_Exists32bitBuffer(SPI_MODULE_ID index)

h) Data Types and Constants

SPI_AUDIO_COMMUNICATION_WIDTH Enumeration

Defines the list of SPI audio communication width.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_AUDIO_COMMUNICATION_24DATA_32FIFO_32CHANNEL,
  SPI_AUDIO_COMMUNICATION_32DATA_32FIFO_32CHANNEL,
  SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_32CHANNEL,
  SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_16CHANNEL
} SPI_AUDIO_COMMUNICATION_WIDTH;
```

Members

Members	Description
SPI_AUDIO_COMMUNICATION_24DATA_32FIFO_32CHANNEL	Communication is 24-bit Data,32-bit FIFO,32-bit Channel/64-bit Frame
SPI_AUDIO_COMMUNICATION_32DATA_32FIFO_32CHANNEL	Communication is 32-bit Data,32-bit FIFO,32-bit Channel/64-bit Frame
SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_32CHANNEL	Communication is 16-bit Data,16-bit FIFO,32-bit Channel/64-bit Frame
SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_16CHANNEL	Communication is 16-bit Data,16-bit FIFO,16-bit Channel/64-bit Frame

Description

SPI Audio Communication Width

This macro defines the list of SPI audio communication width.

SPI_AUDIO_ERROR Enumeration

Defines the list of SPI audio error.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_AUDIO_ERROR_RECEIVE_OVERFLOW,
  SPI_AUDIO_ERROR_TRANSMIT_UNDERRUN
} SPI_AUDIO_ERROR;
```

Members

Members	Description
SPI_AUDIO_ERROR_RECEIVE_OVERFLOW	SPI error for receive overflow
SPI_AUDIO_ERROR_TRANSMIT_UNDERRUN	SPI error for transmit underrun

Description

SPI Audio Error

This macro defines the list of SPI audio error.

SPI_AUDIO_PROTOCOL Enumeration

Data type defining the audio protocol mode.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_AUDIO_PROTOCOL_PCM_DSP,
  SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED,
  SPI_AUDIO_PROTOCOL_LEFT_JUSTIFIED,
  SPI_AUDIO_PROTOCOL_I2S
} SPI_AUDIO_PROTOCOL;
```

Members

Members	Description
SPI_AUDIO_PROTOCOL_PCM_DSP	Audio protocol set to PCM/DSP mode
SPI_AUDIO_PROTOCOL_RIGHT_JUSTIFIED	Audio protocol set to Right-Justified mode
SPI_AUDIO_PROTOCOL_LEFT_JUSTIFIED	Audio protocol set to Left-Justified mode
SPI_AUDIO_PROTOCOL_I2S	Audio protocol set to I2C mode

Description

Audio Protocol Mode enumeration

This data type defining the audio protocol mode.

Remarks

This enumeration is processor specific.

SPI_AUDIO_TRANSMIT_MODE Enumeration

Defines the list of SPI transmit audio mode format.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_AUDIO_TRANSMIT_MONO,
  SPI_AUDIO_TRANSMIT_STEREO
} SPI_AUDIO_TRANSMIT_MODE;
```

Members

Members	Description
SPI_AUDIO_TRANSMIT_MONO	SPI Transmit Audio Data Format is Mono
SPI_AUDIO_TRANSMIT_STEREO	SPI Transmit Audio Data Format is Stereo

Description

Audio Transmit mode format

This macro defines the list of SPI transmit audio mode format.

SPI_BAUD_RATE_CLOCK Enumeration

Defines the list of SPI Baud Rate Generator (BRG).

File

help_plib_spi.h

C

```
typedef enum {
  SPI_BAUD_RATE_MCLK_CLOCK,
  SPI_BAUD_RATE_PBCLK_CLOCK
} SPI_BAUD_RATE_CLOCK;
```

Members

Members	Description
SPI_BAUD_RATE_MCLK_CLOCK	MCLK is used by the Baud Rate Generator
SPI_BAUD_RATE_PBCLK_CLOCK	Peripheral bus clock is used by the Baud Rate Generator

Description

SPI Baud Rate Generator

This macro defines the list of the SPI Baud Rate Generator.

SPI_CLOCK_POLARITY Enumeration

Defines the list of SPI clock polarity.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_CLOCK_POLARITY_IDLE_HIGH,
  SPI_CLOCK_POLARITY_IDLE_LOW
} SPI_CLOCK_POLARITY;
```

Members

Members	Description
SPI_CLOCK_POLARITY_IDLE_HIGH	Idle state for clock is a high level;active state is a low level
SPI_CLOCK_POLARITY_IDLE_LOW	Idle state for clock is a low level;active state is a high level

Description

SPI Clock polarity

This macro defines the list of SPI clock polarity.

SPI_COMMUNICATION_WIDTH Enumeration

Defines the list of SPI communication width.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_COMMUNICATION_WIDTH_32BITS,
  SPI_COMMUNICATION_WIDTH_16BITS,
  SPI_COMMUNICATION_WIDTH_8BITS
} SPI_COMMUNICATION_WIDTH;
```

Members

Members	Description
SPI_COMMUNICATION_WIDTH_32BITS	Communication is 32-bit-wide
SPI_COMMUNICATION_WIDTH_16BITS	Communication is word-wide
SPI_COMMUNICATION_WIDTH_8BITS	Communication is byte-wide

Description

SPI Communication Width

This macro defines the list of SPI communication width.

SPI_DATA_TYPE Type

Data type defining the SPI data size.

File

help_plib_spi.h

C

```
typedef uint16_t SPI_DATA_TYPE;
```

Description

SPI Data Type definition

This data type defines the SPI data size.

SPI_ERROR_INTERRUPT Enumeration

Defines the list of SPI error interrupts.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_ERROR_INTERRUPT_FRAME_ERROR_OVERFLOW,
  SPI_ERROR_INTERRUPT_RECEIVE_OVERFLOW,
  SPI_ERROR_INTERRUPT_TRANSMIT_UNDERRUN
} SPI_ERROR_INTERRUPT;
```

Members

Members	Description
SPI_ERROR_INTERRUPT_FRAME_ERROR_OVERFLOW	SPI interrupt for frame error
SPI_ERROR_INTERRUPT_RECEIVE_OVERFLOW	SPI interrupt for receive overflow error
SPI_ERROR_INTERRUPT_TRANSMIT_UNDERRUN	SPI interrupt for transmit underrun error

Description

SPI Error Interrupt

This macro defines the list of SPI error interrupts.

SPI_FIFO_INTERRUPT Enumeration

Defines the list of SPI Buffer Interrupt mode.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_NOT_FULL,
  SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_1HALF_EMPTY_OR_MORE,
  SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_COMPLETELY_EMPTY,
  SPI_FIFO_INTERRUPT_WHEN_TRANSMISSION_IS_COMPLETE,
  SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_FULL,
  SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_1HALF_FULL_OR_MORE,
  SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_NOT_EMPTY,
  SPI_FIFO_INTERRUPT_WHEN_BUFFER_IS_EMPTY
} SPI_FIFO_INTERRUPT;
```

Members

Members	Description
SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_NOT_FULL	Interrupt when the transmit buffer is not full
SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_1HALF_EMPTY_OR_MORE	Interrupt when the transmit buffer is half empty
SPI_FIFO_INTERRUPT_WHEN_TRANSMIT_BUFFER_IS_COMPLETELY_EMPTY	Interrupt when the transmit buffer is empty
SPI_FIFO_INTERRUPT_WHEN_TRANSMISSION_IS_COMPLETE	Interrupt when transmission is complete
SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_FULL	Interrupt when receive buffer is full
SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_1HALF_FULL_OR_MORE	Interrupt when the receive buffer half full or more
SPI_FIFO_INTERRUPT_WHEN_RECEIVE_BUFFER_IS_NOT_EMPTY	Interrupt when the receive buffer is not empty
SPI_FIFO_INTERRUPT_WHEN_BUFFER_IS_EMPTY	Interrupt when the receive buffer is empty

Description

SPI Buffer Interrupt Mode

This macro defines the list of SPI Buffer Interrupt mode.

SPI_FIFO_TYPE Enumeration

Defines the list of SPI buffer mode.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FIFO_TYPE_TRNSMIT,
  SPI_FIFO_TYPE_RECEIVE
} SPI_FIFO_TYPE;
```

Members

Members	Description
SPI_FIFO_TYPE_TRNSMIT	Buffer type is transmit
SPI_FIFO_TYPE_RECEIVE	Buffer type is receive

Description

SPI Buffer Mode

This macro defines the list of SPI buffer mode.

SPI_FRAME_PULSE_DIRECTION Enumeration

Defines the list of SPI frame sync pulse direction.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FRAME_PULSE_DIRECTION_OUTPUT,
  SPI_FRAME_PULSE_DIRECTION_INPUT
} SPI_FRAME_PULSE_DIRECTION;
```

Members

Members	Description
SPI_FRAME_PULSE_DIRECTION_OUTPUT	Frame sync pulse direction is output
SPI_FRAME_PULSE_DIRECTION_INPUT	Frame sync pulse direction is input

Description

SPI Frame sync pulse direction

This macro defines the list of SPI frame sync pulse direction.

SPI_FRAME_PULSE_EDGE Enumeration

Defines the list of SPI frame sync pulse edge.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FRAME_PULSE_EDGE_COINCIDES_FIRST_BIT_CLOCK,
  SPI_FRAME_PULSE_EDGE_PRECEDES_FIRST_BIT_CLOCK
} SPI_FRAME_PULSE_EDGE;
```

Members

Members	Description
SPI_FRAME_PULSE_EDGE_COINCIDES_FIRST_BIT_CLOCK	Frame sync pulse coincides with first bit clock
SPI_FRAME_PULSE_EDGE_PRECEDES_FIRST_BIT_CLOCK	Frame sync pulse precedes first bit clock

Description

SPI Frame sync pulse edge

This macro defines the list of SPI frame sync pulse edge.

SPI_FRAME_PULSE_POLARITY Enumeration

Defines the list of SPI frame sync pulse polarity.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FRAME_PULSE_POLARITY_ACTIVE_HIGH,
  SPI_FRAME_PULSE_POLARITY_ACTIVE_LOW
} SPI_FRAME_PULSE_POLARITY;
```

Members

Members	Description
SPI_FRAME_PULSE_POLARITY_ACTIVE_HIGH	Frame sync pulse is active high
SPI_FRAME_PULSE_POLARITY_ACTIVE_LOW	Frame sync pulse is active low

Description

SPI Frame sync pulse polarity

This macro defines the list of SPI frame sync pulse polarity.

SPI_FRAME_PULSE_WIDTH Enumeration

Defines the list of SPI frame sync pulse width.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FRAME_PULSE_WIDTH_ONE_WORD_LENGTH,
  SPI_FRAME_PULSE_WIDTH_ONE_CLOCK_WIDE
} SPI_FRAME_PULSE_WIDTH;
```

Members

Members	Description
SPI_FRAME_PULSE_WIDTH_ONE_WORD_LENGTH	Frame sync Pulse width as one word length wide
SPI_FRAME_PULSE_WIDTH_ONE_CLOCK_WIDE	Frame sync Pulse width as one clock wide

Description

SPI Frame sync pulse width

This macro defines the list of SPI frame sync pulse width.

SPI_FRAME_SYNC_PULSE Enumeration

Data type defining the frame sync pulse counter values.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_FRAME_SYNC_PULSE_ON_EVERY_32_DATA_CHARACTER,
  SPI_FRAME_SYNC_PULSE_ON_EVERY_16_DATA_CHARACTER,
  SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER,
  SPI_FRAME_SYNC_PULSE_ON_EVERY_4_DATA_CHARACTER,
  SPI_FRAME_SYNC_PULSE_ON_EVERY_2_DATA_CHARACTER,
  SPI_FRAME_SYNC_PULSE_ON_EVERY_DATA_CHARACTER
} SPI_FRAME_SYNC_PULSE;
```

Members

Members	Description
SPI_FRAME_SYNC_PULSE_ON_EVERY_32_DATA_CHARACTER	Generate a frame sync pulse on every 32 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_16_DATA_CHARACTER	Generate a frame sync pulse on every 16 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_8_DATA_CHARACTER	Generate a frame sync pulse on every 8 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_4_DATA_CHARACTER	Generate a frame sync pulse on every 4 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_2_DATA_CHARACTER	Generate a frame sync pulse on every 2 data character
SPI_FRAME_SYNC_PULSE_ON_EVERY_DATA_CHARACTER	Generate a frame sync pulse on every data character

Description

Frame Sync Pulse Counter enumeration

This data type defining the frame sync pulse counter values.

SPI_INPUT_SAMPLING_PHASE Enumeration

Defines the list of SPI data input sample phase.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE,
  SPI_INPUT_SAMPLING_PHASE_AT_END
} SPI_INPUT_SAMPLING_PHASE;
```

Members

Members	Description
SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE	SPI Data Input Sample phase in middle
SPI_INPUT_SAMPLING_PHASE_AT_END	SPI Data Input Sample phase at end

Description

SPI Data Input Sample Phase

This macro defines the list of SPI data input sample phase.

SPI_MODULE_ID Enumeration

Identifies the supported SPI modules.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_ID_1,
  SPI_ID_2,
  SPI_ID_3,
  SPI_ID_4,
  SPI_NUMBER_OF_MODULES
} SPI_MODULE_ID;
```

Members

Members	Description
SPI_ID_1	SPI Module 1 ID
SPI_ID_2	SPI Module 2 ID
SPI_ID_3	SPI Module 3 ID
SPI_ID_4	SPI Module 4 ID
SPI_NUMBER_OF_MODULES	Number of available SPI modules.

Description

SPI Module ID

This enumeration identifies the SPI modules that are available on a microcontroller. This is the superset of all the possible instances that might be available on Microchip microcontrollers.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

SPI_OUTPUT_DATA_PHASE Enumeration

Defines the list of SPI serial output data changes.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK,
  SPI_OUTPUT_DATA_PHASE_ON_IDLE_TO_ACTIVE_CLOCK
} SPI_OUTPUT_DATA_PHASE;
```

Members

Members	Description
SPI_OUTPUT_DATA_PHASE_ON_ACTIVE_TO_IDLE_CLOCK	Serial output data changes on transition from active clock state to idle clock state
SPI_OUTPUT_DATA_PHASE_ON_IDLE_TO_ACTIVE_CLOCK	Serial output data changes on transition from idle clock state to active clock state.

Description

SPI Output Data Phase

This macro defines the list of SPI serial output data changes.

SPI_PIN Enumeration

Data type defining the SPI pin.

File

help_plib_spi.h

C

```
typedef enum {
  SPI_PIN_DATA_OUT,
  SPI_PIN_DATA_IN,
  SPI_PIN_SLAVE_SELECT
} SPI_PIN;
```

Members

Members	Description
SPI_PIN_DATA_OUT	SPI data output pin
SPI_PIN_DATA_IN	SPI data input pin
SPI_PIN_SLAVE_SELECT	SPI slave select pin

Description

SPI pin

This data type defining the SPI pin.

Remarks

This enumeration is processor specific.

Files

Files

Name	Description
plib_spi.h	SPI Peripheral Library Interface Header for common definitions.
help_plib_spi.h	Identifies the various enumerations in SPI modules supported

Description

This section lists the source and header files used by the library.

plib_spi.h

SPI Peripheral Library Interface Header for common definitions.

Functions

	Name	Description
	PLIB_SPI_AudioCommunicationWidthSelect	Selects the data width for the SPI audio communication.
	PLIB_SPI_AudioErrorDisable	Disables the SPI error.
	PLIB_SPI_AudioErrorEnable	Enables the SPI error.
	PLIB_SPI_AudioProtocolDisable	Audio protocol is disabled.
	PLIB_SPI_AudioProtocolEnable	Audio protocol is enabled.
	PLIB_SPI_AudioProtocolModeSelect	Selects the Audio Protocol mode.
	PLIB_SPI_AudioTransmitModeSelect	Selects the transmit audio data format.
	PLIB_SPI_BaudRateClockSelect	Selects the type of clock is used by the Baud Rate Generator.
	PLIB_SPI_BaudRateSet	Sets the baud rate to the desired value.
	PLIB_SPI_BufferAddressGet	Returns the address of the SPIxBUF (Transmit(SPIxTXB) and Receive (SPIxRXB)) register.
	PLIB_SPI_BufferClear	Clears the SPI buffer.
	PLIB_SPI_BufferRead	Returns the SPI buffer value.
	PLIB_SPI_BufferRead16bit	Returns 16-bit SPI buffer value.
	PLIB_SPI_BufferRead32bit	Returns 32-bit SPI buffer value.
	PLIB_SPI_BufferWrite	Write the data to the SPI buffer.
	PLIB_SPI_BufferWrite16bit	Writes 16-bit data to the SPI buffer.
	PLIB_SPI_BufferWrite32bit	Write 32-bit data to the SPI buffer.
	PLIB_SPI_ClockPolaritySelect	Enables clock polarity.
	PLIB_SPI_CommunicationWidthSelect	Selects the data width for the SPI communication.
	PLIB_SPI_Disable	Disables the SPI module.

	PLIB_SPI_Enable	Enables the SPI module.
	PLIB_SPI_ErrorInterruptDisable	Enables SPI error interrupts.
	PLIB_SPI_ErrorInterruptEnable	Enables SPI error interrupts
	PLIB_SPI_Exists16bitBuffer	Identifies whether the Buffer16bit feature exists on the SPI module.
	PLIB_SPI_Exists32bitBuffer	Identifies whether the Buffer32bit feature exists on the SPI module.
	PLIB_SPI_ExistsAudioCommunicationWidth	Identifies whether the AudioCommunicationWidth feature exists on the SPI module.
	PLIB_SPI_ExistsAudioErrorControl	Identifies whether the AudioErrorControl feature exists on the SPI module.
	PLIB_SPI_ExistsAudioProtocolControl	Identifies whether the AudioProtocolControl feature exists on the SPI module.
	PLIB_SPI_ExistsAudioProtocolMode	Identifies whether the AudioProtocolMode feature exists on the SPI module.
	PLIB_SPI_ExistsAudioTransmitMode	Identifies whether the AudioTransmitMode feature exists on the SPI module.
	PLIB_SPI_ExistsBaudRate	Identifies whether the BaudRate feature exists on the SPI module.
	PLIB_SPI_ExistsBaudRateClock	Identifies whether the BaudRateClock feature exists on the SPI module.
	PLIB_SPI_ExistsBuffer	Identifies whether the Buffer feature exists on the SPI module.
	PLIB_SPI_ExistsBusStatus	Identifies whether the BusStatus feature exists on the SPI module.
	PLIB_SPI_ExistsClockPolarity	Identifies whether the ClockPolarity feature exists on the SPI module.
	PLIB_SPI_ExistsCommunicationWidth	Identifies whether the CommunicationWidth feature exists on the SPI module.
	PLIB_SPI_ExistsEnableControl	Identifies whether the EnableControl feature exists on the SPI module.
	PLIB_SPI_ExistsErrorInterruptControl	Identifies whether the ErrorInterruptControl feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOControl	Identifies whether the FIFOControl feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOCount	Identifies whether the FIFOCount feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOInterruptMode	Identifies whether the FIFOInterruptMode feature exists on the SPI module.
	PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus	Identifies whether the FIFOShiftRegisterEmptyStatus feature exists on the SPI module.
	PLIB_SPI_ExistsFramedCommunication	Identifies whether the FramedCommunication feature exists on the SPI module.
	PLIB_SPI_ExistsFrameErrorStatus	Identifies whether the FrameErrorStatus feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseCounter	Identifies whether the FrameSyncPulseCounter feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseDirection	Identifies whether the FrameSyncPulseDirection feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseEdge	Identifies whether the FrameSyncPulseEdge feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulsePolarity	Identifies whether the FrameSyncPulsePolarity feature exists on the SPI module.
	PLIB_SPI_ExistsFrameSyncPulseWidth	Identifies whether the FrameSyncPulseWidth feature exists on the SPI module.
	PLIB_SPI_ExistsInputSamplePhase	Identifies whether the InputSamplePhase feature exists on the SPI module.
	PLIB_SPI_ExistsMasterControl	Identifies whether the MasterControl feature exists on the SPI module.
	PLIB_SPI_ExistsOutputDataPhase	Identifies whether the OutputDataPhase feature exists on the SPI module.
	PLIB_SPI_ExistsPinControl	Identifies whether the PinControl feature exists on the SPI module.
	PLIB_SPI_ExistsReadDataSignStatus	Identifies whether the ReadDataSignStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiveBufferStatus	Identifies whether the ReceiveBufferStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiveFIFOStatus	Identifies whether the ReceiveFIFOStatus feature exists on the SPI module.
	PLIB_SPI_ExistsReceiverOverflow	Identifies whether the ReceiverOverflow feature exists on the SPI module.
	PLIB_SPI_ExistsSlaveSelectControl	Identifies whether the SlaveSelectControl feature exists on the SPI module.
	PLIB_SPI_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitBufferEmptyStatus	Identifies whether the TransmitBufferEmptyStatus feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitBufferFullStatus	Identifies whether the TransmitBufferFullStatus feature exists on the SPI module.
	PLIB_SPI_ExistsTransmitUnderRunStatus	Identifies whether the TransmitUnderRunStatus feature exists on the SPI module.
	PLIB_SPI_FIFOCountGet	Reads the SPI Buffer Element Count bits for either receive or transmit.
	PLIB_SPI_FIFODisable	Disables the SPI enhanced buffer.
	PLIB_SPI_FIFOEnable	Enables the SPI enhanced buffer.
	PLIB_SPI_FIFOInterruptModeSelect	Selects the SPI buffer interrupt mode.
	PLIB_SPI_FIFOShiftRegisterIsEmpty	Returns the current status of the SPI shift register.
	PLIB_SPI_FramedCommunicationDisable	Disables framed SPI support.
	PLIB_SPI_FramedCommunicationEnable	Enables framed SPI support.
	PLIB_SPI_FrameErrorStatusClear	Clears the SPI frame error flag.
	PLIB_SPI_FrameErrorStatusGet	Returns the current status of the SPI frame error.
	PLIB_SPI_FrameSyncPulseCounterSelect	Selects at which character the SPI frame sync pulse is generated.
	PLIB_SPI_FrameSyncPulseDirectionSelect	Selects the frame sync pulse direction.

	PLIB_SPI_FrameSyncPulseEdgeSelect	Selects the frame sync pulse edge.
	PLIB_SPI_FrameSyncPulsePolaritySelect	Selects the frame sync pulse polarity.
	PLIB_SPI_FrameSyncPulseWidthSelect	Sets the frame sync pulse width.
	PLIB_SPI_InputSamplePhaseSelect	Selects the SPI data input sample phase.
	PLIB_SPI_IsBusy	Returns the current SPI module activity status.
	PLIB_SPI_MasterEnable	Enables the SPI in Master mode.
	PLIB_SPI_OutputDataPhaseSelect	Selects serial output data change.
	PLIB_SPI_PinDisable	Enables the selected SPI pins.
	PLIB_SPI_PinEnable	Enables the selected SPI pins.
	PLIB_SPI_ReadDataIsSignExtended	Returns the current status of the receive (RX) FIFO sign-extended data.
	PLIB_SPI_ReceiverBufferIsFull	Returns the current status of the SPI receive buffer.
	PLIB_SPI_ReceiverFIFOIsEmpty	Returns the current status of the SPI receive FIFO.
	PLIB_SPI_ReceiverHasOverflowed	Returns the current status of the SPI receiver overflow.
	PLIB_SPI_ReceiverOverflowClear	Clears the SPI receive overflow flag.
	PLIB_SPI_SlaveEnable	Enables the SPI in Slave mode.
	PLIB_SPI_SlaveSelectDisable	Disables Master mode slave select.
	PLIB_SPI_SlaveSelectEnable	Enables Master mode slave select.
	PLIB_SPI_StopInIdleDisable	Continues module operation when the device enters Idle mode.
	PLIB_SPI_StopInIdleEnable	Discontinues module operation when the device enters Idle mode.
	PLIB_SPI_TransmitBufferIsEmpty	Returns the current status of the transmit buffer.
	PLIB_SPI_TransmitBufferIsFull	Returns the current transmit buffer status of the SPI module.
	PLIB_SPI_TransmitUnderRunStatusClear	Clears the SPI transmit underrun flag.
	PLIB_SPI_TransmitUnderRunStatusGet	Returns the current status of the transmit underrun.

Description

SPI Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the SPI PLIB.

File Name

plib_spi.h

Company

Microchip Technology Inc.

help_plib_spi.h

Identifies the various enumerations in SPI modules supported

Enumerations

	Name	Description
	SPI_AUDIO_COMMUNICATION_WIDTH	Defines the list of SPI audio communication width.
	SPI_AUDIO_ERROR	Defines the list of SPI audio error.
	SPI_AUDIO_PROTOCOL	Data type defining the audio protocol mode.
	SPI_AUDIO_TRANSMIT_MODE	Defines the list of SPI transmit audio mode format.
	SPI_BAUD_RATE_CLOCK	Defines the list of SPI Baud Rate Generator (BRG).
	SPI_CLOCK_POLARITY	Defines the list of SPI clock polarity.
	SPI_COMMUNICATION_WIDTH	Defines the list of SPI communication width.
	SPI_ERROR_INTERRUPT	Defines the list of SPI error interrupts.
	SPI_FIFO_INTERRUPT	Defines the list of SPI Buffer Interrupt mode.
	SPI_FIFO_TYPE	Defines the list of SPI buffer mode.
	SPI_FRAME_PULSE_DIRECTION	Defines the list of SPI frame sync pulse direction.
	SPI_FRAME_PULSE_EDGE	Defines the list of SPI frame sync pulse edge.
	SPI_FRAME_PULSE_POLARITY	Defines the list of SPI frame sync pulse polarity.
	SPI_FRAME_PULSE_WIDTH	Defines the list of SPI frame sync pulse width.
	SPI_FRAME_SYNC_PULSE	Data type defining the frame sync pulse counter values.
	SPI_INPUT_SAMPLING_PHASE	Defines the list of SPI data input sample phase.

	SPI_MODULE_ID	Identifies the supported SPI modules.
	SPI_OUTPUT_DATA_PHASE	Defines the list of SPI serial output data changes.
	SPI_PIN	Data type defining the SPI pin.

Types

	Name	Description
	SPI_DATA_TYPE	Data type defining the SPI data size.

Description

This enumeration identifies the SPI modules that are available on a microcontroller. This is the superset of all the possible instances that might be available on Microchip microcontrollers.

File Name

help_plib_spi.h

Company

Microchip Technology Inc.

SQI Peripheral Library

This section describes the Serial Quad Interface (SQI) Peripheral Library.

Introduction

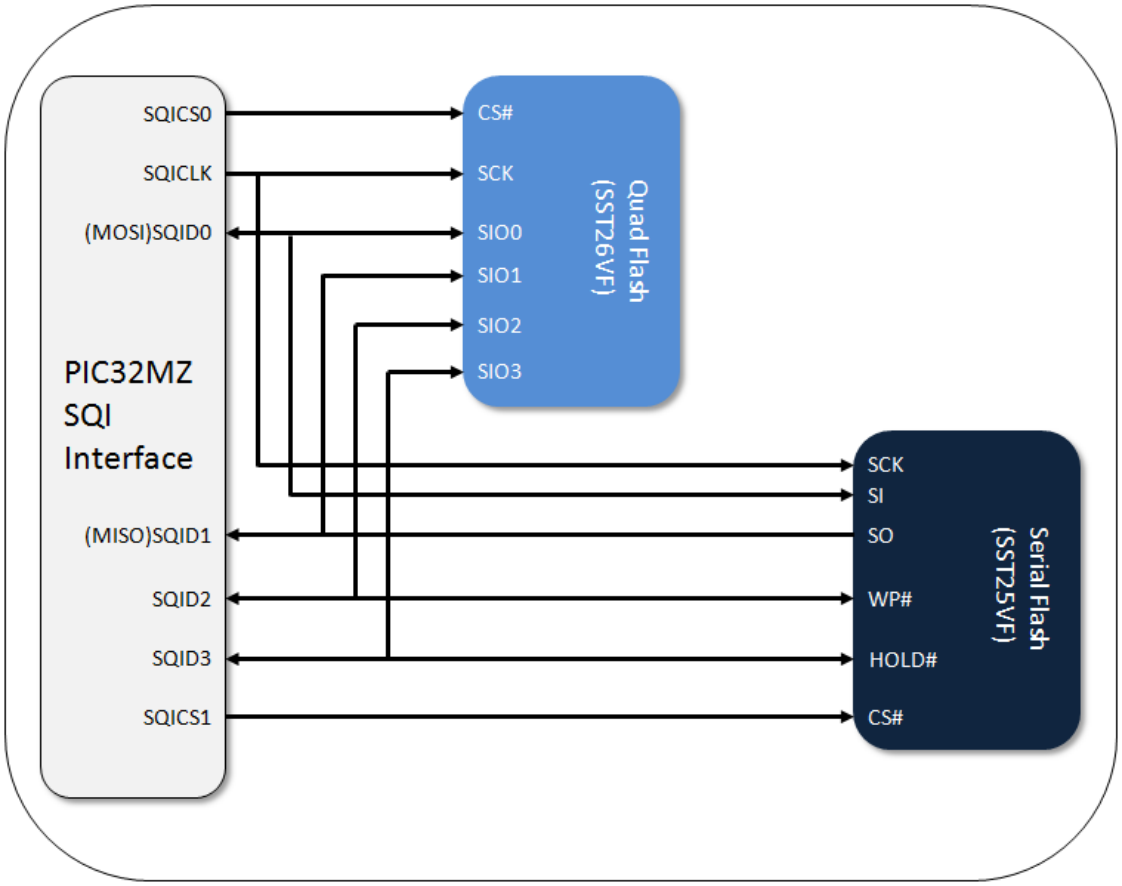
This library provides a low-level abstraction of the Serial Quad Interface (SQI) Peripheral that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the SQI module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The SQI module is a synchronous serial interface that provides access to serial Flash memories and other serial devices. The SQI bus interface consists of four data lines (SQID3-SQID0), a clock line (SQICLK), two select lines (SQICS0 and SQICS1) and the module supports single Lane (identical to SPI), dual Lane, and quad Lane interface modes. SQI operates in only master mode. The SQI module has configurable transmit and receive buffers, programmable baud rates through the internal clock divider, clock phase, and clock polarity control for efficient data operations. Transmit and receive buffers can be accessed through SQITXDATA and SQIRXDATA registers. Similarly, the control buffer can be accessed through the SQI1CON register and is mainly used to pipeline the operations.

The SQI module operates in three transfer modes: DMA, PIO, and XIP using three data flow modes:SPI Mode 0 and Mode 3, and a high-speed Serial Flash mode . All three modes use the control buffer to pipeline the command/data sequences on the SQI bus. Following diagrams describe typical SQI hardware interface and software flow. Figure 1 shows the typical hardware interface of SQI module.

Figure 1: Hardware Interface Diagram



Refer to **Section 46. "Serial Quad Interface (SQI)"** (DS60001244), which is available on the Microchip website for additional details on control register information and operation.

Using the Library

This topic describes the basic architecture of the SQI Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_sqi.h`

The interface to the SQI peripheral library is defined in the `plib_sqi.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the SQI peripheral library must include `peripheral.h`.

Library File:

The SQI peripheral library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Hardware Abstraction Model

This library provides the low-level abstraction of the SQI module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Before jumping into the software abstraction, it is important to understand the SQI hardware a little more in detail.

Figure 2 provides a glimpse of SQI hardware block diagram. As shown the diagram the features of SQI are divided into multiple logic blocks and the SQI peripheral library provides APIs to configure the various logic block thus the features.

The control and status SFRs block contains all the registers that are used to configure the SQI in a specific mode of operation. The PIO mode of operation of SQI requires a set of registers to be configured by the Host processor and writing to/ reading from the TXDATA and RXDATA registers. The XIP logic facilitates the SQI modules eXecute-In-Place mode of operation, where Host processor programs the set of registers that configure the XIP mode and the engine automatically reads the external Flash and the read data gets mapped into a direct memory region. Where as DMA mode requires the Host processor to program a set of buffer descriptors and instructs SQI to point to the address of those buffer descriptors for the reads and writes. As mentioned earlier there are 3 buffers; transmit, receive and control that are accessed through SQIxTXDATA, SQIxRXDATA, and SQIxCON registers, respectively.

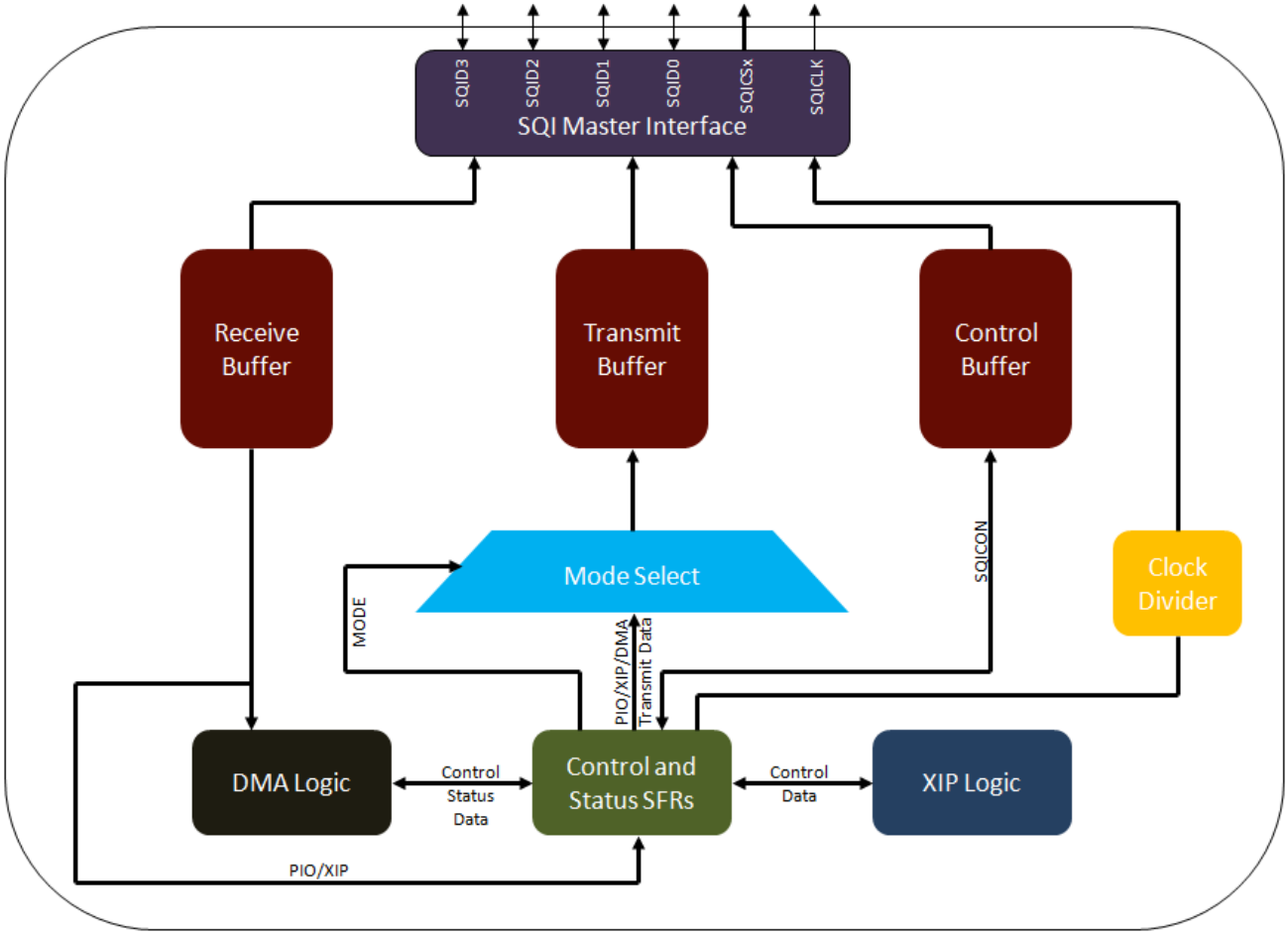


Figure 2: SQI Hardware Block Diagram

The SQI Peripheral Library takes requests from application software or the SQI device driver and controls the SQI hardware as per the inputs passed to the Library Interface. Figure 3 provides the SQI peripheral library software abstraction layer diagram.

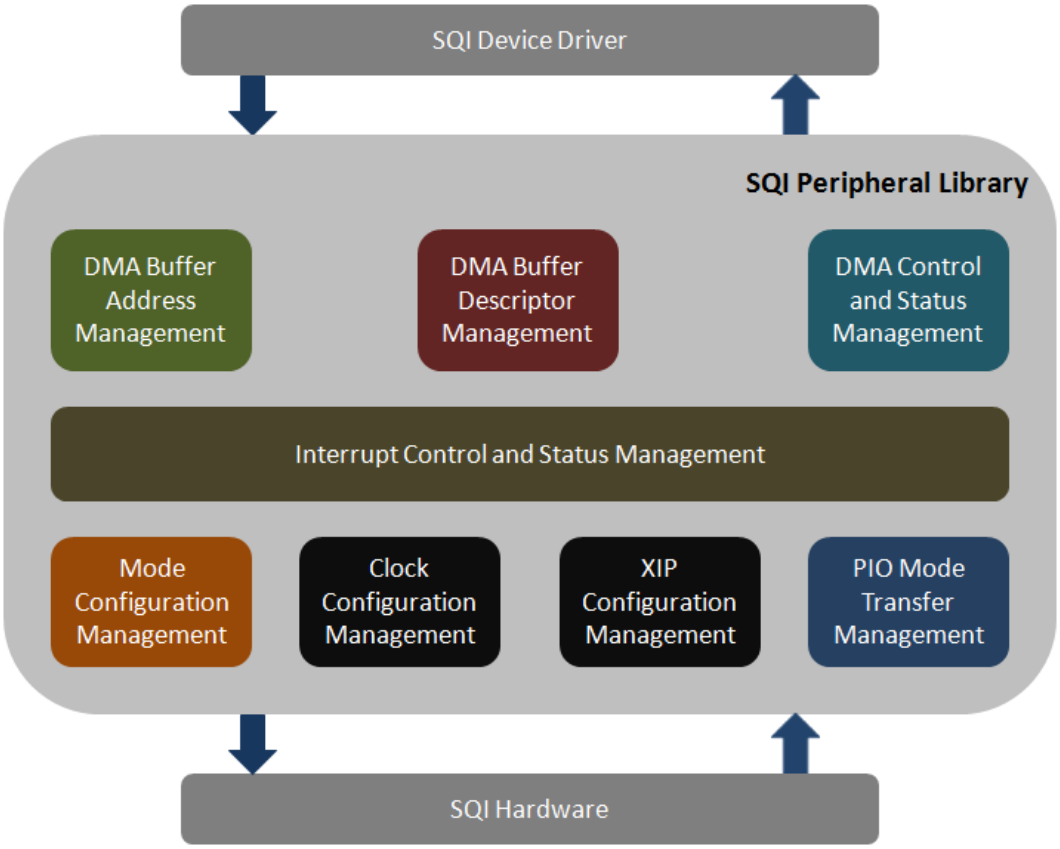


Figure 3: SQI Software Abstraction Model

The major components are SQI Peripheral Library are:

- Configuration Management
- Interrupt Control and Status Management
- DMA Mode Management

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SQI module.

Library Interface Section	Description
Mode Configuration Management Functions	These functions are used to set up transfer mode (DMA/XIP/PIO), data mode (SPI Mode 0/Mode 3/Serial Flash), lane mode (single/dual/quad) and XIP control registers.
Clock Configuration Management Functions	These functions are used to enable the clock, set up a clock divider and check the status of the clock to see if it is stable.
XIP Configuration Management Functions	These functions are used to set up parameters for XIP mode of operation.
PIO Mode Transfer Management Functions	These functions are used to set up TXDATA, RXDATA registers and status checks.
Interrupt Control and Status Management Functions	These functions are used to control the interrupts. SQI supports only status type of interrupts (no error interrupts).
DMA Buffer Address Management Functions	These functions are used to set up the address of the base descriptor, check the address of the current descriptor
DMA Buffer Descriptor Management Functions	These functions are used to set up and control 4 words (BD_CTRL, BD_STATUS (reserved), BD_BUF_ADDR and BD_NEXT_PTR)
DMA Control and Status Management Functions	These functions are used to control the buffer descriptor fetch process and check the status during the buffer descriptor fetches.

How the Library Works

Note: Not all features are available on all devices. Refer to the **"Serial Quad Interface (SQI)"** chapter in the specific device data sheet for availability.

Core Functionality

SQI modules core functionality is to transfer data between the serial device (mostly quad Flash) attached and the PIC microcontroller. To achieve this, SQI module uses 3 transfer modes: DMA, PIO and XIP. Each transfer mode can be configured to use any of the 3 available data flow modes (Mode 0, Mode 3 and Serial Flash Mode) to achieve the transfers. DMA and PIO modes are typically used to transfer data and XIP mode is used for code execution.

Each mode of operation requires setting-up the clock, selecting data flow mode and selecting lane mode and other parameters as required by the device driver or application. SQI peripheral library includes complete set of API necessary to perform these operations.

The following sections describe the SQI core functionality in each mode of operation.

XIP Mode

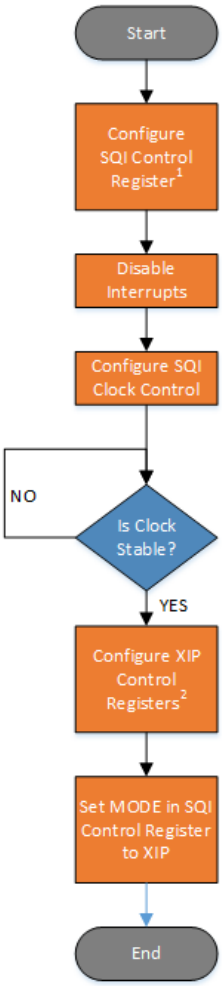
Describes XIP mode.

Description

XIP (eXecute-In-Place) mode is mostly used to execute the code out of the attached serial Flash device. However, this mode can also be used to transfer data as deemed necessary. In order for SQI to operate in XIP mode, host processor would have to set up the two SFRs dedicated to the XIP mode in addition to SQI configuration and clock control SFRs. Please refer to the specific data sheet and family reference manual for the configuration details.

Please note that all the XIP configuration values depend on the parameters of the interfaced serial Flash memory device. In XIP mode, reading values of any other SFRs return the contents of the serial Flash device, the way it is mapped.

The following figure shows the XIP mode software flow.



Notes:
1. This step configures the SQI configuration fields, SQIEN, CSEN, and DATAEN, with the exception of MODE<1:0>.
2. There are two XIP control registers: SQI1XCON1 and SQI1XCON2.

Example:

```
// Example is written to access SST26VF032 Flash device
```

```
// Example assumes the Flash Device is connected to Chip Select 0

//Global defines
#define SST26VF_HS_READ    0x0B
#define SST26VF_MODE_CODE  0x0
#define SST26VF_MODE_BYTES 0x0

// Set up SQI Configuration Bits
PLIB_SQI_Enable(SQI_ID_1);
PLIB_SQI_CSOutputEnableSelect(SQI_ID_1, SQI_CS_OEN_0);
PLIB_SQI_DataOutputEnableSelect(SQI_ID_1, SQI_DATA_OEN_QUAD);

// Disable All Interrupts
PLIB_SQI_InterruptDisable(SQI_ID_1, SQI_ALL_INT);
PLIB_SQI_InterruptSignalDisable(SQI_ID_1, SQI_ALL_INT);

// Set up SQI Transfer Clock
PLIB_SQI_ClockDividerSet(SQI_ID_1,CLK_DIV_1);
PLIB_SQI_ClockEnable(SQI_ID_1);
while (!PLIB_SQI_ClockIsStable(SQI_ID_1));

// Configure XIP Control 1 SFR
PLIB_SQI_XIPControlWord1Set (SQI_ID_1,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SST26VF_HS_READ,
                             ADDR_BYTES_3,
                             DUMMY_BYTES_2
                            );

// Configure XIP Control 2 SFR
PLIB_SQI_XIPControlWord2Set (SQI_ID_1,
                             SST26VF_MODE_CODE,
                             MODE_BYTES_0,
                             SQI_CS_0
                            );

// Set Transfer Mode to XIP
PLIB_SQI_TransferModeSet(SQI_ID_1, SQI_XFER_MODE_XIP);

// The SQI is now in XIP mode and should start fetching the code
```

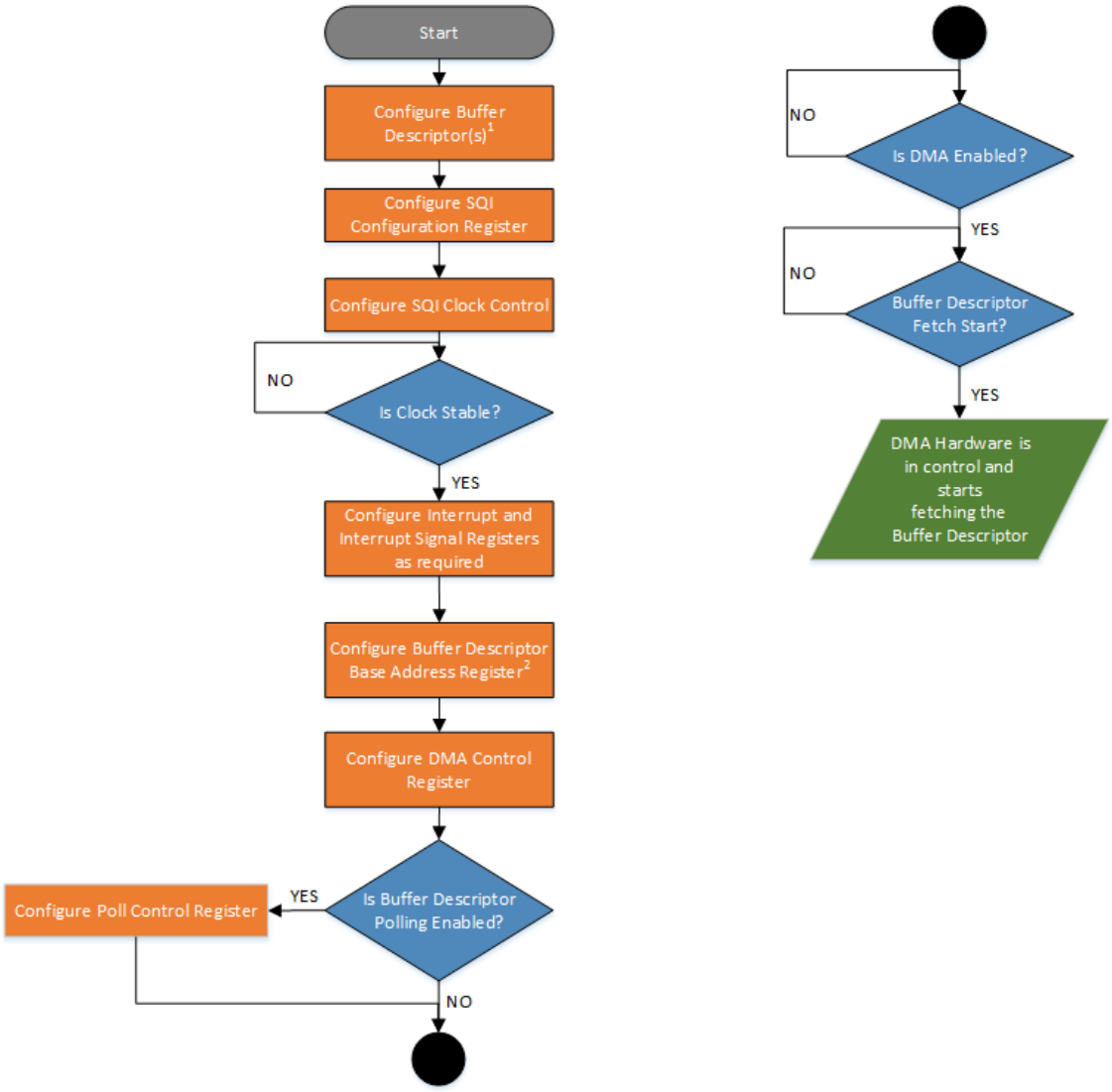
DMA Mode

Describes DMA mode.

Description

DMA mode is used for higher throughput data transfers. In DMA mode, the SQI DMA engine controls the data transfers off-loading the host processor. However, the host processor would have to configure the buffer descriptor and some SQI control SFRs to initiate the DMA process. Once the transactions are in progress, the DMA will be in control and the host processor can get the control through the interrupts or by disabling the DMA.

The following figure shows the DMA mode software flow.



Notes:

1. Buffer descriptor is a data structure in memory containing 4 words (BD_CTRL, BD_STAT, BD_BUFADDR and BD_NXTPTR).
2. Base address register points to first buffer descriptor in the chain.

Example:

```
// Example is written to access SST26VF032 Flash device
// Example assumes the Flash Device is connected to Chip Select 0

//Global defines
#define SST26VF_EQIO        0x38
#define SST26VF_HS_READ     0x0B
#define SST26VF_MODE_CODE   0x0
#define SST26VF_MODE_BYTES  0x0
#define NUMBER_OF_BDs       5
#define POLL_CON_VALUE      10

{

// Set up the Buffer Descriptors
for ( i=0; i <(NUMBER_OF_BDs-1); i++)
{
// Set up Buffer Descriptors. Handle structures as application
// needs.

// Configure SQI Control Fields
PLIB_SQI_Enable(SQI_ID_1);
PLIB_SQI_CSOutputEnableSelect(SQI_ID_1,SQI_CS_OEN_0);
PLIB_SQI_DataOutputEnableSelect(SQI_ID_1,SQI_DATA_OEN_2);
```

```
PLIB_SQI_BurstEnable(SQI_ID_1);
PLIB_SQI_SerialModeSet(SQI_ID_1);
PLIB_SQI_TransferModeSet(SQI_ID_1,SQI_XFER_MODE_PIO);

// Configure the SQI Transfer Clock
PLIB_SQI_ClockDividerSet(SQI_ID_1,CLK_DIV_4);
PLIB_SQI_ClockEnable(SQI_ID_1);
while (!PLIB_SQI_ClockIsStable(SQI_ID_1));

//Enable Buffer Descriptor Done and Packet Done Interrupts
PLIB_SQI_InterruptEnable(SQI_ID_1,BDDONE);
PLIB_SQI_InterruptSignalEnable(SQI_ID_1,BDDONE);
PLIB_SQI_InterruptEnable(SQI_ID_1,PKTCOMP);
PLIB_SQI_InterruptSignalEnable(SQI_ID_1,PKTCOMP);

// Send EQIO(0x38) Command to SST26VF032 to Enable Quad Lane Mode
// Note: These steps are not in the flow as these are specific to
// one Flash Device (SST26VF032)
PLIB_SQI_ChipSelectSet(SQI_ID_1,SQI_CS_0);
PLIB_SQI_LaneModeSet(SQI_ID_1,SQI_SINGLE_LANE);
PLIB_SQI_TransferCommandSet(SQI_ID_1,SQI_CMD_TRANSMIT);
// Transfers 1 Command Byte, No Address/Dummy Bytes
PLIB_SQI_TransferCountSet(SQI_ID_1, 0x1);
PLIB_SQI_TransmitterDataSend(SQI_ID_1, SST26VF_EQIO);

// Set the SQI in DMA Mode
PLIB_SQI_TransferModeSet(SQI_ID_1,SQI_XFER_MODE_DMA);

// Set up the Base Buffer Descriptor Address
if (!PLIB_SQI_DMAIsActive(SQI_ID_1))
    PLIB_SQI_DMABDBaseAddressSet(SQI_ID_1, (void *) (&MY_BASE_BD_STRUCT));

//Set up DMA Control Register
PLIB_SQI_DMABDEnable(SQI_ID_1);
PLIB_SQI_DMABDPollEnable(SQI_ID_1);

// Set up DMA Polling
If (PLIB_SQI_DMABDPollIsEnabled(SQI_ID_1))
   // Poll DMA every 10 cycles
   PLIB_SQI_PollControlSet(SQI_ID_1, MY_POLL_CONTROL_VALUE);

// Start the DMA Process - DMA Engine is now in Control
If (PLIB_SQI_DMAIsEnabled(SQI_ID_1))
{
   if (!PLIB_SQI_DMAIsActive(MY_SQI_INSTANCE))
      PLIB_SQI_DMABDFetchStart(SQI_ID_1);

}

}
```

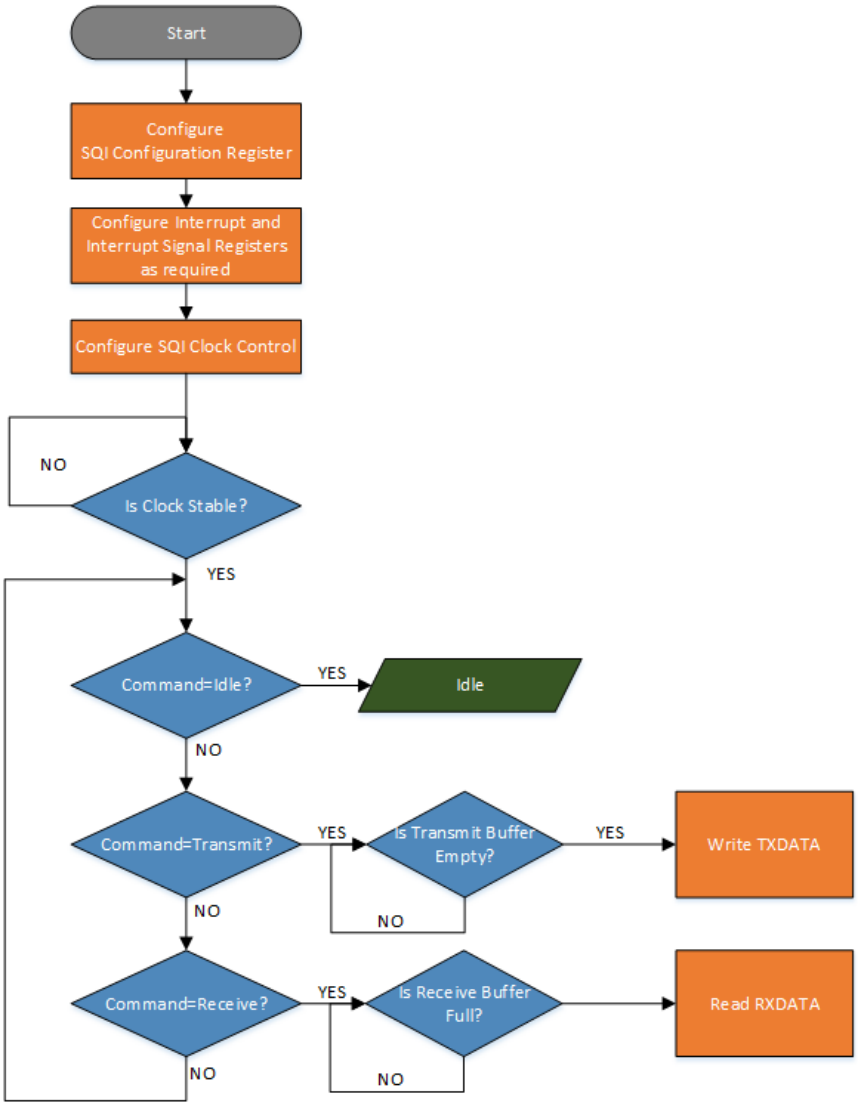
PIO Mode

Describes PIO mode.

Description

PIO mode is mostly used for data transfers. In PIO mode, the SQI module is entirely under the control of host processor. For the SQI module to operate in PIO mode, the host processor would have to set up the configuration, control, and clock control SFRs. Once the transactions are in progress, the host processor can use interrupts and status flags to control the operation. Please refer to the specific device data sheet and family reference manual for the configuration details.

The following figure shows the PIO mode software flow.



Example:

```
// Example is written to access SST26VF032 Flash device
// Example assumes the Flash Device is connected to Chip Select 0
// Example reads the device ID of the SST26VF032

//Global defines
#define FALSE                0
#define TRUE                 1
#define SST26VF_EQIO         0x38
#define SST26VF_HS_READ      0x0B
#define SST26VF_QJEDID_READ  0x9F
#define SST26VF_JEDEC_ID     0xBF2602

{
int32_t jedecID;

// Set up SQI Configuration (SQI1CFG) Register
PLIB_SQI_ConfigWordSet(MY_SQI_INSTANCE,
                       1,
                       SQI_CS_OEN_0,
                       SQI_DATA_OEN_QUAD,
                       0,
                       1,
                       0,
                       0,
                       0,
                       SQI_DATA_FORMAT_MSBF,
                       SQI_DATA_MODE_3,
```

```
                        SQI_XFER_MODE_PIO
                        );

  PLIB_SQI_ControlBufferThresholdSet(SQI_ID_1,4);

  // Enable Specific Interrupts
  PLIB_SQI_InterruptEnable(SQI_ID_1,TXEMPTY|RXFULL/TXTHRIE|RXTHRIE);

  // Set up SQI Transfer Clock
  PLIB_SQI_ClockDividerSet(SQI_ID_1,CLK_DIV_1);
  PLIB_SQI_ClockEnable(SQI_ID_1);
  while (!PLIB_SQI_ClockIsStable(SQI_ID_1));

  // Set up SQI Transmit Command Threshold
  PLIB_SQI_TxCommandThresholdSet(SQI_ID_1,3);

  // Set up SQI Receive Buffer Threshold
  PLIB_SQI_TxBufferThresholdIntSet(SQI_ID_1,3);

  // Configure SQI Control Fields and Send EQIO(0x38) Command to
  // SST26VF032 to Enable Quad Lane Mode
  PLIB_SQI_ControlWordSet(MY_SQI_INSTANCE,
                          1,
                          SQI_CS_1,
                          SQI_LANE_SINGLE,
                          SQI_CMD_TRANSMIT,
                          1
                          );
  PLIB_SQI_ControlWordSet(MY_SQI_INSTANCE,
                          0,
                          SQI_CS_1,
                          SQI_LANE_QUAD,
                          SQI_CMD_TRANSMIT,
                          1
                          );
  PLIB_SQI_ControlWordSet(MY_SQI_INSTANCE,
                          0,
                          SQI_CS_1,
                          SQI_LANE_QUAD,
                          SQI_CMD_RECEIVE,
                          3
                          );
  if ( PLIB_SQI_InterruptFlagGet(SQI_ID_1,TXEMPTY) )
     PLIB_SQI_TransmitData(SQI_ID_1, SST26VF_EQIO);

  //Configure SQI to read JEDEC-ID in Quad Mode
  if ( PLIB_SQI_InterruptFlagGet(SQI_ID_1,TXEMPTY) )
     PLIB_SQI_TransmitData(SQI_ID_1, SST26VF_QJEDID_READ);

  if (PLIB_SQI_InterruptFlagGet(SQI_ID_1,RXFULL))
     jedecID = PLIB_SQI_ReceiveData(SQI_ID_1);

  if (jedecID == SST26VF_JEDEC_ID)
  return TRUE;
  else
  return FALSE;
  }
```

Configuring the Library

The library is configured for the supported SQI module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) Mode Configuration Management Functions

	Name	Description
	PLIB_SQI_BurstEnable	Sets the burst enable (BURSTEN) function for higher throughput. This function is an artifact of the System Bus architecture.
	PLIB_SQI_ClockDisable	Disables the SQI transfer clock.
	PLIB_SQI_ControlBufferThresholdGet	Returns the transmit buffer space in bytes.
	PLIB_SQI_ControlBufferThresholdSet	Sets the control buffer threshold value.
	PLIB_SQI_ControlWordGet	Get the SQI Control Word.
	PLIB_SQI_ControlWordSet	Sets the SQI Control Word.
	PLIB_SQI_CSOutputEnableSelect	Selects the output enables on SQI Chip Select pins.
	PLIB_SQI_DataFormatGet	Returns the data format.
	PLIB_SQI_DataFormatSet	Sets the data format to Least Significant Bit First (LSBF)..
	PLIB_SQI_DataModeSet	Sets the SQI data mode of operation.
	PLIB_SQI_DataOutputEnableSelect	Selects the output enables on SQI data outputs.
	PLIB_SQI_Disable	Disables the SQI module.
	PLIB_SQI_Enable	Enables the SQI module.
	PLIB_SQI_HoldClear	Clears the hold function to be disabled on SQID3 in single and dual lane modes.
	PLIB_SQI_HoldGet	Gets the status of HOLD function on SQID3 pin.
	PLIB_SQI_HoldSet	Sets the hold function to be enabled on SQID3 in single or dual lane modes.
	PLIB_SQI_LaneModeGet	Returns the lane mode (single/dual/quad).
	PLIB_SQI_LaneModeSet	Sets the data lane mode (single/dual/quad).
	PLIB_SQI_NumberOfReceiveBufferReads	Returns the number of receive buffer reads.
	PLIB_SQI_ReceiveData	Reads the data from the receive buffer.
	PLIB_SQI_ReceiveLatchDisable	Disables the receive latch so receive data is discarded when in transmit mode.
	PLIB_SQI_ReceiveLatchEnable	Enables the receive latch so receive data is latched during transmit mode.
	PLIB_SQI_ReceiveLatchGet	Returns the receive latch status in transmit mode.
	PLIB_SQI_SoftReset	Issues a soft reset to the SQI module.
	PLIB_SQI_TransferModeGet	Returns the SQI transfer mode of operation.
	PLIB_SQI_TransferModeSet	Sets the SQI transfer mode of operation.
	PLIB_SQI_TransmitData	Writes data into the SQI transmit buffer.
	PLIB_SQI_WriteProtectClear	Clears the Write-Protect function to be disabled on SQID2 in single or dual lane modes.
	PLIB_SQI_WriteProtectGet	Gets the state of the write-protect feature on SQID2.
	PLIB_SQI_WriteProtectSet	Sets the write-protect feature to be enabled on SQID2 in single or dual lane modes only.
	PLIB_SQI_ConBufferSoftReset	Issues a soft reset to the SQI control buffer.
	PLIB_SQI_RxBufferSoftReset	Issues a soft reset to the SQI receive buffer.
	PLIB_SQI_TxBufferSoftReset	Issues a soft reset to the SQI transmit buffer.
	PLIB_SQI_TapDelaySet	Sets the tap delays.
	PLIB_SQI_DDRModeGet	Return the SQI data rate mode (single/double) value.
	PLIB_SQI_DDRModeSet	Sets SQI communication to DDR mode.

b) Clock Configuration Management Functions

	Name	Description
	PLIB_SQI_ClockDividerSet	Sets the SQI clock (that drives the SQI protocol) divider value. Divides the base clock to generate the SQI clock.
	PLIB_SQI_ClockEnable	Enables the SQI transfer clock.
	PLIB_SQI_ClockIsStable	Returns SQI transfer clock state.

c) XIP Configuration Management Functions

	Name	Description
	PLIB_SQI_XIPAddressBytesGet	Returns the number of address bytes.
	PLIB_SQI_XIPAddressBytesSet	Sets the number of address bytes.
	PLIB_SQI_XIPChipSelectGet	Returns the current active Chip Select.

	PLIB_SQI_XIPChipSelectSet	Activates the Chip Select in XIP mode.
	PLIB_SQI_XIPControlWord1Get	Get the XIP mode Control Word 1.
	PLIB_SQI_XIPControlWord1Set	Sets the XIP mode Control Word 1.
	PLIB_SQI_XIPControlWord2Get	Gets the XIP mode Control Word 2.
	PLIB_SQI_XIPControlWord2Set	Sets the XIP mode Control Word 2.
	PLIB_SQI_XIPDummyBytesGet	Sets the number of dummy bytes.
	PLIB_SQI_XIPDummyBytesSet	Sets the number of dummy bytes.
	PLIB_SQI_XIPLaneModeSet	Selects the lane (Single/Dual/Quad) mode for different transaction in XIP mode.
	PLIB_SQI_XIPModeBytesGet	Returns the number of bytes used for mode code command.
	PLIB_SQI_XIPModeBytesSet	Allocates the bytes for mode code command.
	PLIB_SQI_XIPModeCodeGet	Returns the mode code op-code.
	PLIB_SQI_XIPModeCodeSet	Sets the mode code command.
	PLIB_SQI_XIPReadOpcodeGet	Returns the read op code in XIP mode.
	PLIB_SQI_XIPReadOpcodeSet	Sets the read op code in XIP mode.
	PLIB_SQI_XIPControlWord3Get	Gets the XIP mode Control Word 3.
	PLIB_SQI_XIPControlWord3Set	Sets the XIP mode Control Word 3.
	PLIB_SQI_XIPControlWord4Get	Gets the XIP mode Control Word 4.
	PLIB_SQI_XIPDDRModeSet	Selects the rate mode (SDR/DDR) for different transactions in XIP mode.
	PLIB_SQI_XIPSDRCommandDDRDataGet	Returns the SQI command in SDR mode and Data in DDR mode bit value.
	PLIB_SQI_XIPSDRCommandDDRDataSet	Sets the SQI command in SDR mode.
	PLIB_SQI_XIPControlWord4Set	Sets the XIP mode Control Word 4.

d) PIO Mode Transfer Management Functions

	Name	Description
	PLIB_SQI_ByteCountGet	Returns the current transmit/receive count.
	PLIB_SQI_ByteCountSet	Sets the transmit/receive count.
	PLIB_SQI_ChipSelectDeassertDisable	Clears the Chip Select deassert.
	PLIB_SQI_ChipSelectDeassertEnable	Sets the Chip Select deassert.
	PLIB_SQI_ChipSelectGet	Returns the Chip Select that is currently active.
	PLIB_SQI_ChipSelectSet	Activates the Chip Select.
	PLIB_SQI_ConfigWordGet	Gets the SQI Configuration Word.
	PLIB_SQI_ConfigWordSet	Sets SQI Configuration Word.
	PLIB_SQI_ReceiveBufferIsUnderrun	Returns the status of receive buffer.
	PLIB_SQI_RxBufferThresholdGet	Returns the receive command threshold.
	PLIB_SQI_RxBufferThresholdIntGet	Sets the receive buffer threshold interrupt.
	PLIB_SQI_RxBufferThresholdIntSet	Sets the receive buffer threshold for interrupt.
	PLIB_SQI_RxBufferThresholdSet	Sets the receive command threshold.
	PLIB_SQI_TransferDirectionGet	Returns the transfer command.
	PLIB_SQI_TransferDirectionSet	Sets the transfer command.
	PLIB_SQI_TransmitBufferFreeSpaceGet	Returns the number of transmit buffer words available.
	PLIB_SQI_TransmitBufferHasOverflowed	Returns the current status of the transmit buffer.
	PLIB_SQI_TxBufferThresholdGet	Returns the transmit command threshold value.
	PLIB_SQI_TxBufferThresholdIntGet	Returns the value to trigger the transmit buffer threshold interrupt.
	PLIB_SQI_TxBufferThresholdIntSet	Sets the value to trigger the transmit buffer threshold interrupt.
	PLIB_SQI_TxBufferThresholdSet	Sets the transmit command threshold.

e) Interrupt Control and Status Management Functions

	Name	Description
	PLIB_SQI_InterruptDisable	Disables the interrupt source.
	PLIB_SQI_InterruptEnable	Enables the passed interrupt source.
	PLIB_SQI_InterruptFlagGet	Return SQI Interrupt flag status.
	PLIB_SQI_InterruptIsEnabled	Returns the interrupt state.
	PLIB_SQI_InterruptSignalDisable	Disables the interrupt signal source.
	PLIB_SQI_InterruptSignalEnable	Enables the passed interrupt signal source.
	PLIB_SQI_InterruptSignalIsEnabled	Returns the interrupt signal state.

	PLIB_SQI_DataLineStatus	Returns the logical status of the SQI data lines.
	PLIB_SQI_CommandStatusGet	Returns the SQI command (transmit/receive/idle).
	PLIB_SQI_ConBufWordsAvailable	Returns the number of control buffer words available.

f) DMA Buffer Address Management Functions

	Name	Description
	PLIB_SQI_DMABDBaseAddressGet	Returns the address of the base buffer descriptor.
	PLIB_SQI_DMABDBaseAddressSet	Sets the address of the base buffer descriptor.
	PLIB_SQI_DMABDCurrentAddressGet	Returns the address of the current buffer descriptor in process.

g) DMA Buffer Descriptor Management Functions

	Name	Description
	PLIB_SQI_DMABDControlWordGet	Returns Current Buffer Descriptor Control Word Information.
	PLIB_SQI_DMABDReceiveBufferCountGet	Returns the receive buffer space in bytes.
	PLIB_SQI_DMABDReceiveBufferLengthGet	Returns the receive length in bytes.
	PLIB_SQI_DMABDReceiveStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMABDTransmitBufferCountGet	Returns the transmit buffer space in bytes.
	PLIB_SQI_DMABDTransmitBufferLengthGet	Returns the transmit length in bytes.
	PLIB_SQI_DMABDTransmitStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMABDFetchStop	Stops the DMA buffer descriptor fetch process.

h) DMA Control and Status Management Functions

	Name	Description
	PLIB_SQI_DMABDFetchStart	Starts the DMA buffer descriptor fetch process.
	PLIB_SQI_DMABDIsBusy	Returns whether or not the DMA Buffer Descriptor is busy.
	PLIB_SQI_DMABDPollCounterSet	Sets the poll counter value.
	PLIB_SQI_DMABDPollDisable	Disables the buffer descriptor polling.
	PLIB_SQI_DMABDPollEnable	Enables the buffer descriptor polling.
	PLIB_SQI_DMABDPollIsEnabled	Returns whether or not the DMA buffer descriptor poll is enabled.
	PLIB_SQI_DMABDStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMADisable	Disables the built-in DMA logic.
	PLIB_SQI_DMAEnable	Enables the built-in DMA logic.
	PLIB_SQI_DMAHasStarted	Returns whether or no the DMA process has started.
	PLIB_SQI_DMAIsEnabled	Returns whether or not DMA is enabled.

i) Other Functions

	Name	Description
	PLIB_SQI_StatusCheckSet	Sets the Flash status check related control bits.

j) Feature Existence Functions

	Name	Description
	PLIB_SQI_ExistsBDBaseAddress	Identifies whether the BDBaseAddress feature exists on the SQI module.
	PLIB_SQI_ExistsBDControlWord	Identifies whether the BDControlWord feature exists on the SQI module.
	PLIB_SQI_ExistsBDCurrentAddress	Identifies whether the BDCurrentAddress feature exists on the SQI module.
	PLIB_SQI_ExistsBDPollCount	Identifies whether the BDPollCount feature exists on the SQI module.
	PLIB_SQI_ExistsBDPollingEnable	Identifies whether the BDPollingEnable feature exists on the SQI module.
	PLIB_SQI_ExistsBDProcessState	Identifies whether the BDProcessState feature exists on the SQI module.
	PLIB_SQI_ExistsBDRxBufCount	Identifies whether the BDRxBufCount feature exists on the SQI module.
	PLIB_SQI_ExistsBDRxLength	Identifies whether the BDRxLength feature exists on the SQI module.
	PLIB_SQI_ExistsBDRxState	Identifies whether the BDRxState feature exists on the SQI module.
	PLIB_SQI_ExistsBDTxBufCount	Identifies whether the BDTxBufCount feature exists on the SQI module.
	PLIB_SQI_ExistsBDTxLength	Identifies whether the BDTxLength feature exists on the SQI module.
	PLIB_SQI_ExistsBDTxState	Identifies whether the BDTxState feature exists on the SQI module.
	PLIB_SQI_ExistsBurstControl	Identifies whether the BurstControl feature exists on the SQI module.
	PLIB_SQI_ExistsChipSelect	Identifies whether the ChipSelect feature exists on the SQI module.

	PLIB_SQI_ExistsClockControl	Identifies whether the ClockControl feature exists on the SQI module.
	PLIB_SQI_ExistsClockDivider	Identifies whether the ClockDivider feature exists on the SQI module.
	PLIB_SQI_ExistsClockReady	Identifies whether the ClockReady feature exists on the SQI module.
	PLIB_SQI_ExistsConBufThreshold	Identifies whether the ConBufThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsConfigWord	Identifies whether the ConfigWord feature exists on the SQI module.
	PLIB_SQI_ExistsControlWord	Identifies whether the ControlWord feature exists on the SQI module.
	PLIB_SQI_ExistsCSDeassert	Identifies whether the CSDeassert feature exists on the SQI module.
	PLIB_SQI_ExistsCSOutputEnable	Identifies whether the CSOutputEnable feature exists on the SQI module.
	PLIB_SQI_ExistsDataFormat	Identifies whether the DataFormat feature exists on the SQI module.
	PLIB_SQI_ExistsDataModeControl	Identifies whether the DataModeControl feature exists on the SQI module.
	PLIB_SQI_ExistsDataOutputEnable	Identifies whether the DataOutputEnable feature exists on the SQI module.
	PLIB_SQI_ExistsDataPinStatus	Identifies whether the DataPinStatus feature exists on the SQI module.
	PLIB_SQI_ExistsDmaEnable	Identifies whether the DMAEnable feature exists on the SQI module.
	PLIB_SQI_ExistsDMAEngineBusy	Identifies whether the DMAEngineBusy feature exists on the SQI module.
	PLIB_SQI_ExistsDMAProcessInProgress	Identifies whether the DMAProcessInProgress feature exists on the SQI module.
	PLIB_SQI_ExistsEnableControl	Identifies whether the EnableControl feature exists on the SQI module.
	PLIB_SQI_ExistsHoldPinControl	Identifies whether the HoldPinControl feature exists on the SQI module.
	PLIB_SQI_ExistsInterruptControl	Identifies whether the InterruptControl feature exists on the SQI module.
	PLIB_SQI_ExistsInterruptSignalControl	Identifies whether the InterruptSignalControl feature exists on the SQI module.
	PLIB_SQI_ExistsInterruptStatus	Identifies whether the InterruptStatus feature exists on the SQI module.
	PLIB_SQI_ExistsLaneMode	Identifies whether the LaneMode feature exists on the SQI module.
	PLIB_SQI_ExistsReceiveLatch	Identifies whether the ReceiveLatch feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufferCount	Identifies whether the RxBufferCount feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufIntThreshold	Identifies whether the RxBufIntThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufThreshold	Identifies whether the RxBufThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsRxData	Identifies whether the RxData feature exists on the SQI module.
	PLIB_SQI_ExistsRxUnderRun	Identifies whether the RxUnderRun feature exists on the SQI module.
	PLIB_SQI_ExistsSoftReset	Identifies whether the SoftReset feature exists on the SQI module.
	PLIB_SQI_ExistsTransferCommand	Identifies whether the TransferCommand feature exists on the SQI module.
	PLIB_SQI_ExistsTransferCount	Identifies whether the TransferCount feature exists on the SQI module.
	PLIB_SQI_ExistsTransferModeControl	Identifies whether the TransferModeControl feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufferFree	Identifies whether the TxBufferFree feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufIntThreshold	Identifies whether the TxBufIntThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufThreshold	Identifies whether the TxBufThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsTxData	Identifies whether the TxData feature exists on the SQI module.
	PLIB_SQI_ExistsTxOverFlow	Identifies whether the TxOverFlow feature exists on the SQI module.
	PLIB_SQI_ExistsWPPinControl	Identifies whether the WPPinControl feature exists on the SQI module.
	PLIB_SQI_ExistsXIPChipSelect	Identifies whether the XIPChipSelect feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord1	Identifies whether the XIPControlWord1 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord2	Identifies whether the XIPControlWord2 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPLaneMode	Identifies whether the XIPLaneMode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPModeBytes	Identifies whether the XIPModeBytes feature exists on the SQI module.
	PLIB_SQI_ExistsXIPModeCode	Identifies whether the XIPModeCode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPNumberOfAddressBytes	Identifies whether the XIPNumberOfAddressBytes feature exists on the SQI module.
	PLIB_SQI_ExistsXIPNumberOfDummyBytes	Identifies whether the XIPNumberOfDummyBytes feature exists on the SQI module.
	PLIB_SQI_ExistsXIPReadOpCode	Identifies whether the XIPReadOpCode feature exists on the SQI module.
	PLIB_SQI_ExistsCommandStatus	Identifies whether the CommandStatus feature exists on the SQI module.
	PLIB_SQI_ExistsConBufAvailable	Identifies whether the ConBufWordsAvailable feature exists on the SQI module.
	PLIB_SQI_ExistsConBufferSoftReset	Identifies whether the control buffer soft reset feature exists on the SQI module.
	PLIB_SQI_ExistsDDRMode	Identifies whether the DDRMode feature exists on the SQI module.
	PLIB_SQI_ExistsDMABDFetch	Identifies whether the DMABDFetch feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufferSoftReset	Identifies whether the receive buffer soft reset feature exists on the SQI module.
	PLIB_SQI_ExistsStatusCheck	Identifies whether the StatusCheckSet feature exists on the SQI module.
	PLIB_SQI_ExistsTapDelay	Identifies whether the TapDelay feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufferSoftReset	Identifies whether the transmit buffer soft reset feature exists on the SQI module.

	PLIB_SQI_ExistsXIPControlWord3	Identifies whether the XIPControlWord3 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord4	Identifies whether the XIPControlWord4 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPDDRMode	Identifies whether the XIPDDRMode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPSDRCommandDDRData	Identifies whether the XIPSDRCommandDDRData feature exists on the SQI module.

k) Data Types and Constants

	Name	Description
	SQI_ADDR_BYTES	Defines the list of SQI address bytes.
	SQI_BD_CTRL_WORD	Defines the list of SQI Buffer Descriptor control word.
	SQI_BD_STATE	Defines the list of SQI Buffer Descriptor states.
	SQI_CLK_DIV	Defines the list of SQI Clock Divider values.
	SQI_CS_NUM	Defines the list of SQI Chip Selects.
	SQI_CS_OEN	Defines the list of SQI Chip Selects on which output is enable.
	SQI_DATA_FORMAT	Defines the Data Format Options available (LSBF/MSBF).
	SQI_DATA_MODE	Defines the list of SQI Data modes.
	SQI_DATA_OEN	Defines the list of SQI Data pins on which output is enabled.
	SQI_DATA_TYPE	Data type defining the SQI Data size.
	SQI_DUMMY_BYTES	Defines the list of SQI dummy bytes.
	SQI_INTERRUPTS	Defines the list of SQI interrupts.
	SQI_LANE_MODE	Defines the list of SQI Lane modes (single/dual/quad).
	SQI_MODE_BYTES	Defines the list of SQI mode bytes.
	SQI_MODULE_ID	Identifies the supported SQI modules.
	SQI_XFER_CMD	Defines the list of SQI transfer commands.
	SQI_XFER_MODE	Defines the list of SQI Transfer modes.

Description

This section describes the Application Programming Interface (API) functions of the SQI Peripheral Library.

Refer to each section for a detailed description.

a) Mode Configuration Management Functions

PLIB_SQI_BurstEnable Function

Sets the burst enable (BURSTEN) function for higher throughput. This function is an artifact of the System Bus architecture.

File

plib_sqi.h

C

```
void PLIB_SQI_BurstEnable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables burst mode for higher throughput. Burst mode should always be enabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_BurstEnable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_BurstEnable (SQI_MODULE_ID index)

PLIB_SQI_ClockDisable Function

Disables the SQI transfer clock.

File

plib_sqi.h

C

```
void PLIB_SQI_ClockDisable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the SQI transfer clock (divided clock).

Remarks

SQICLK is disabled.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ClockDisable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ClockDisable (SQI_MODULE_ID index)

PLIB_SQI_ControlBufferThresholdGet Function

Returns the transmit buffer space in bytes.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_ControlBufferThresholdGet(SQI_MODULE_ID index);
```

Returns

Control buffer threshold space.

Description

This function returns the threshold value for the control buffer in bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t conBufThres = PLIB_SQI_ControlBufferThresholdGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_ControlBufferThresholdGet(SQI_MODULE_ID index)

PLIB_SQI_ControlBufferThresholdSet Function

Sets the control buffer threshold value.

File

plib_sqi.h

C

```
void PLIB_SQI_ControlBufferThresholdSet(SQI_MODULE_ID index, uint8_t threshold);
```

Returns

None.

Description

This function sets the control buffer threshold value in bytes, that is used to signal control buffer threshold interrupts.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_CONBUF_THRESHOLD is the threshold value.
PLIB_SQI_ControlBufferThresholdSet(MY_SQI_INSTANCE, MY_CONBUF_THRESHOLD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Control buffer threshold

Function

void PLIB_SQI_ControlBufferThresholdSet(SQI_MODULE_ID index, uint8_t threshold)

PLIB_SQI_ControlWordGet Function

Get the SQI Control Word.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_ControlWordGet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the SQI Control Word.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint32_t sqiCon = PLIB_SQI_ControlWordGet (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_ControlWordGet (SQI_MODULE_ID index)

PLIB_SQI_ControlWordSet Function

Sets the SQI Control Word.

File

plib_sqi.h

C

```
void PLIB_SQI_ControlWordSet(SQI_MODULE_ID index, bool csDeassert, SQI_CS_NUM csNum, SQI_LANE_MODE
laneMode, SQI_XFER_CMD command, uint16_t xferCount);
```

Returns

None.

Description

This function sets the SQI Control Word. This function is a combination of several functions in case the driver plans to write the complete Control Word. In PIO mode, the Control word is before each transfer

Remarks

Chip select is not actually asserted, only enabled to be asserted.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ControlWordSet(MY_SQI_INSTANCE,
                        1,
                        SQI_CS_1,
                        SQI_LANE_QUAD,
                        SQI_CMD_TRANSMIT,
                        5
                        );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

csDeassert	Chip select deassert after transfer
csNum	Active Chip Select number (0/1)
laneMode	SQI lane mode (QUAL/DUAL/SINGLE)
command	Transfer command (TRANSMIT/RECIEVE/IDLE)
xferCount	Number of bytes to be transmitted/received

Function

void PLIB_SQI_ControlWordSet(SQI_MODULE_ID index,
bool csDeassert,
SQI_CS_NUM csNum,
SQI_LANE_MODE laneMode,
SQI_XFER_CMD command,
uint16_t xferCount
)

PLIB_SQI_CSOutputEnableSelect Function

Selects the output enables on SQI Chip Select pins.

File

plib_sqi.h

C

```
void PLIB_SQI_CSOutputEnableSelect(SQI_MODULE_ID index, SQI_CS_OEN csPins);
```

Returns

None.

Description

This function enables the selected SQI chip selects as outputs.

Remarks

Chip select is not actually asserted, only enabled to be asserted.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_CSOutputEnableSelect(MY_SQI_INSTANCE, SQI_CS_OEN_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
csPins	Chip select pins for which outputs are enabled

Function

void PLIB_SQI_CSOutputEnableSelect(SQI_MODULE_ID index, SQI_CS_OEN csPins)

PLIB_SQI_DataFormatGet Function

Returns the data format.

File

plib_sqi.h

C

```
SQI_DATA_FORMAT PLIB_SQI_DataFormatGet(SQI_MODULE_ID index);
```

Returns

- true - Data Format is LSBF (i.e., LITTLE-ENDIAN)
- false - Data Format is MSBF

Description

This function returns the SQI data format (LSBF or MSBF).

Remarks

Typical data format in most of systems is LITTLE-ENDIAN.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_DATA_FORMAT dataFormat = PLIB_SQI_DataFormatGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_DATA_FORMAT PLIB_SQI_DataFormatGet(SQI_MODULE_ID index)

PLIB_SQI_DataFormatSet Function

Sets the data format to Least Significant Bit First (LSBF)..

File

plib_sqi.h

C

```
void PLIB_SQI_DataFormatSet(SQI_MODULE_ID index, SQI_DATA_FORMAT dataformat);
```

Returns

None.

Description

This function sets the SQI data format to LSBF (i.e., LITTLE-ENDIAN).

Remarks

Typical data format in most of the Systems is LITTLE ENDIAN.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DataFormatSet(MY_SQI_INSTANCE, SQI_DATA_FORMAT_LSBF);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DataFormatSet(SQI_MODULE_ID index, SQI_DATA_FORMAT dataformat)

PLIB_SQI_DataModeSet Function

Sets the SQI data mode of operation.

File

plib_sqi.h

C

```
void PLIB_SQI_DataModeSet(SQI_MODULE_ID index, SQI_DATA_MODE mode);
```

Returns

None.

Description

This function sets the data mode to be SPI Mode 0, SPI Mode 3, or Serial Flash.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and SQI_DATA_MODE_3 is an enum element.
PLIB_SQI_DataModeSet(MY_SQI_INSTANCE, SQI_DATA_MODE_SF);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Data mode (Mode 0/Mode 3)

Function

void PLIB_SQI_DataModeSet(SQI_MODULE_ID index, SQI_DATA_MODE mode)

PLIB_SQI_DataOutputEnableSelect Function

Selects the output enables on SQI data outputs.

File

plib_sqi.h

C

```
void PLIB_SQI_DataOutputEnableSelect(SQI_MODULE_ID index, SQI_DATA_OEN dataPins);
```

Returns

None.

Description

This function enables the selected SQI data lines as outputs.

Remarks

Chip select is not actually asserted, only enabled to be asserted.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DataOutputEnableSelect(MY_SQI_INSTANCE, SQI_DATA_OEN_DUAL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dataPins	Data pins for which outputs are enabled

Function

void PLIB_SQI_DataOutputEnableSelect (SQI_MODULE_ID index, SQI_DATA_OEN dataPins)

PLIB_SQI_Disable Function

Disables the SQI module.

File

plib_sqi.h

C

```
void PLIB_SQI_Disable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the SQI module.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_Disable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_Disable (SQI_MODULE_ID index)

PLIB_SQI_Enable Function

Enables the SQI module.

File

plib_sqi.h

C

```
void PLIB_SQI_Enable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SQI module.

Remarks

The SQICLK, SQICSx, SQID0, SQID1, SQID2, and SQID3 pins must be assigned to available RPn pins before use.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_Enable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_Enable(SQI_MODULE_ID index)

PLIB_SQI_HoldClear Function

Clears the hold function to be disabled on SQID3 in single and dual lane modes.

File

plib_sqi.h

C

```
void PLIB_SQI_HoldClear(SQI_MODULE_ID index);
```

Returns

None.

Description

This function sets SQID3 to be controlled by SQI for normal data operation.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_HoldClear(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_HoldClear (SQI_MODULE_ID index)

PLIB_SQI_HoldGet Function

Gets the status of HOLD function on SQID3 pin.

File

plib_sqi.h

C

```
bool PLIB_SQI_HoldGet(SQI_MODULE_ID index);
```

Returns

SQID3 as HOLD pin value.

Description

This function gets the status of HOLD function on SQID3 pin.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool holdStatus = PLIB_SQI_HoldGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_HoldGet (SQI_MODULE_ID index)

PLIB_SQI_HoldSet Function

Sets the hold function to be enabled on SQID3 in single or dual lane modes.

File

plib_sqi.h

C

```
void PLIB_SQI_HoldSet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function sets the SQID3 pin to HIGH/LOW to be be used for hold function in single and dual lane modes for supported Flash memories.

Remarks

This function should be used only when SQI is in single/dual lane modes.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_HoldSet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_HoldSet (SQI_MODULE_ID index)

PLIB_SQI_LaneModeGet Function

Returns the lane mode (single/dual/quad).

File

plib_sqi.h

C

```
SQI_LANE_MODE PLIB_SQI_LaneModeGet(SQI_MODULE_ID index);
```

Returns

Lane mode (single/dual/quad).

Description

This function returns the number of lanes (single/dual/quad) used for transfers.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_LANE_MODE laneMode = PLIB_SQI_LaneModeGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_LANE_MODE PLIB_SQI_LaneModeGet (SQI_MODULE_ID index)

PLIB_SQI_LaneModeSet Function

Sets the data lane mode (single/dual/quad).

File

plib_sqi.h

C

```
void PLIB_SQI_LaneModeSet(SQI_MODULE_ID index, SQI_LANE_MODE mode);
```

Returns

None.

Description

This function sets the number of lanes (single/dual/quad) used for transfers.

Remarks

None.

Preconditions

Make sure the output enable is selected on the data lines using PLIB_SQI_DataOutputEnableSelect. The device needs to be programmed to the same mode that the SQI controller is set to (may require special commands).

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_LaneModeSet(MY_SQI_INSTANCE, SQI_LANE_QUAD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Lane mode (single/dual/quad)

Function

void PLIB_SQI_LaneModeSet (SQI_MODULE_ID index, SQI_LANE_MODE mode)

PLIB_SQI_NumberOfReceiveBufferReads Function

Returns the number of receive buffer reads.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_NumberOfReceiveBufferReads(SQI_MODULE_ID index);
```

Returns

Number of Receive Buffer Reads.

Description

This function returns the number of receive buffer reads for debug purpose.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t rxBufReads = PLIB_SQI_ReceiveBufferReadsGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_ReceiveBufferReadsGet(SQI_MODULE_ID index)

PLIB_SQI_ReceiveData Function

Reads the data from the receive buffer.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_ReceiveData(SQI_MODULE_ID index);
```

Returns

None.

Description

This function reads the data from the SQI receive buffer.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
```

```
// application developer.
uint32_t receivedData= PLIB_SQI_ReceiveData(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_ReceiveData (SQI_MODULE_ID index)

PLIB_SQI_ReceiveLatchDisable Function

Disables the receive latch so receive data is discarded when in transmit mode.

File

plib_sqi.h

C

```
void PLIB_SQI_ReceiveLatchDisable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the receive latch, which disables the receive data to be latched in transmit mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ReceiveLatchDisable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ReceiveLatchDisable (SQI_MODULE_ID index)

PLIB_SQI_ReceiveLatchEnable Function

Enables the receive latch so receive data is latched during transmit mode.

File

plib_sqi.h

C

```
void PLIB_SQI_ReceiveLatchEnable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables the receive latch, which latches receive data in transmit mode. Otherwise, receive data in transmit mode is discarded.

Remarks

As most of the SQI communication is half-duplex, enable this function only when it is absolutely required.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ReceiveLatchEnable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ReceiveLatchEnable (SQI_MODULE_ID index)

PLIB_SQI_ReceiveLatchGet Function

Returns the receive latch status in transmit mode.

File

plib_sqi.h

C

```
bool PLIB_SQI_ReceiveLatchGet(SQI_MODULE_ID index);
```

Returns

- true - Receive latch is set
- false - Receive latch is not set

Description

This function returns the receive latch status in transmit mode. Returns true, if latch is set (enabling latching of receive buffer data), false if latch is not set (disabling the latching of the receive buffer data).

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool rxLatch = PLIB_SQI_ReceiveLatchGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_ReceiveLatchGet (SQI_MODULE_ID index)

PLIB_SQI_SoftReset Function

Issues a soft reset to the SQI module.

File

plib_sqi.h

C

```
void PLIB_SQI_SoftReset(SQI_MODULE_ID index);
```

Returns

None.

Description

This function issues a software reset to the SQI module clearing all the read/write register, internal state machines and data buffers.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_SoftReset(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_SoftReset (SQI_MODULE_ID index)

PLIB_SQI_TransferModeGet Function

Returns the SQI transfer mode of operation.

File

plib_sqi.h

C

```
SQI_XFER_MODE PLIB_SQI_TransferModeGet(SQI_MODULE_ID index);
```

Returns

Transfer mode (PIO/DMA/XIP).

Description

This function returns the SQI transfer mode of operation: PIO, DMA, or XIP.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_XFER_MODE xferMode = PLIB_SQI_TransferModeGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured.

Function

SQI_XFER_MODE PLIB_SQI_TransferModeGet (SQI_MODULE_ID index)

PLIB_SQI_TransferModeSet Function

Sets the SQI transfer mode of operation.

File

plib_sqi.h

C

```
void PLIB_SQI_TransferModeSet(SQI_MODULE_ID index, SQI_XFER_MODE mode);
```

Returns

None.

Description

This function sets the SQI transfer mode of operation, (PIO, DMA, or XIP.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and SQI_XFER_MODE_DMA is an enum element.
PLIB_SQI_TransferModeSet(MY_SQI_INSTANCE, SQI_XFER_MODE_DMA);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	Transfer mode (PIO/DMA/XIP)

Function

void PLIB_SQI_TransferModeSet (SQI_MODULE_ID index, SQI_XFER_MODE mode)

PLIB_SQI_TransmitData Function

Writes data into the SQI transmit buffer.

File

plib_sqi.h

C

```
void PLIB_SQI_TransmitData(SQI_MODULE_ID index, uint32_t data);
```

Returns

None.

Description

This function writes the data into the SQI transmit buffer, which will be eventually sent out on SQI bus.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_TRANSMIT_DATA is the data to be sent.
PLIB_SQI_TransmitData(MY_SQI_INSTANCE, MY_TRANSMIT_DATA);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

data	Data to be transmitted

Function

void PLIB_SQI_TransmitData (SQI_MODULE_ID index, uint32_t data)

PLIB_SQI_WriteProtectClear Function

Clears the Write-Protect function to be disabled on SQID2 in single or dual lane modes.

File

plib_sqi.h

C

```
void PLIB_SQI_WriteProtectClear(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the SQID2 pin to be used for write-protect.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_WriteProtectClear(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_WriteProtectClear (SQI_MODULE_ID index)

PLIB_SQI_WriteProtectGet Function

Gets the state of the write-protect feature on SQID2.

File

plib_sqi.h

C

```
bool PLIB_SQI_WriteProtectGet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the write-protect feature status on the SQID2 pin.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool wpStatus = PLIB_SQI_WriteProtectGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_WriteProtectGet (SQI_MODULE_ID index)

PLIB_SQI_WriteProtectSet Function

Sets the write-protect feature to be enabled on SQID2 in single or dual lane modes only.

File

plib_sqi.h

C

```
void PLIB_SQI_WriteProtectSet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SQID2 pin to be used for write-protect in single and dual lane modes for supported Flash memories.

Remarks

This function should be used only when SQI is in single/dual lane modes.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_WriteProtectSet (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_WriteProtectSet (SQI_MODULE_ID index)

PLIB_SQI_ConBufferSoftReset Function

Issues a soft reset to the SQI control buffer.

File

plib_sqi.h

C

```
void PLIB_SQI_ConBufferSoftReset(SQI_MODULE_ID index);
```

Returns

None.

Description

This function issues a soft reset to the SQI control buffer clearing all the data in the buffer and addresses.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ConBufferSoftReset(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ConBufferSoftReset (SQI_MODULE_ID index)

PLIB_SQI_RxBufferSoftReset Function

Issues a soft reset to the SQI receive buffer.

File

plib_sqi.h

C

```
void PLIB_SQI_RxBufferSoftReset(SQI_MODULE_ID index);
```

Returns

None.

Description

This function issues a soft reset to the SQI receive buffer clearing all the data in the buffer and addresses.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_RxBufferSoftReset(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_RxBufferSoftReset (SQI_MODULE_ID index)

PLIB_SQI_TxBufferSoftReset Function

Issues a soft reset to the SQI transmit buffer.

File

plib_sqi.h

C

```
void PLIB_SQI_TxBufferSoftReset(SQI_MODULE_ID index);
```

Returns

None.

Description

This function issues a soft reset to the SQI transmit buffer clearing all the data in the buffer and addresses.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_TxBufferSoftReset(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_TxBufferSoftReset (SQI_MODULE_ID index)

PLIB_SQI_TapDelaySet Function

Sets the tap delays.

File

plib_sqi.h

C

```
void PLIB_SQI_TapDelaySet(SQI_MODULE_ID index, uint8_t sdrClkInDly, uint8_t dataOutDly, uint8_t clkOutDly);
```

Returns

SQI command (transmit/receive/idle) that is currently being executed.

Description

Sets the tap delays that might be required at the higher interface speeds to handle data set-up and hold delays.

Remarks

This function should be used only when the SQI returns failures at maximum frequency.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
void PLIB_SQI_TapDelaySet(SQI_MODULE_ID index,
                          8,
                          0,
                          2
                          );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_TapDelaySet(SQI_MODULE_ID index,

```
uint8_t sdrClkInDly,
uint8_t dataOutDly,
uint8_t clkOutDly
);
```

PLIB_SQI_DDRModeGet Function

Return the SQI data rate mode (single/double) value.

File

plib_sqi.h

C

```
bool PLIB_SQI_DDRModeGet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the SQI communication mode value.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool pioDDRMode = PLIB_SQI_DDRModeGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_DDRModeGet(SQI_MODULE_ID index)

PLIB_SQI_DDRModeSet Function

Sets SQI communication to DDR mode.

File

plib_sqi.h

C

```
void PLIB_SQI_DDRModeSet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function sets SQI communication to DDR mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DDRModeSet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DDRModeSet(SQI_MODULE_ID index)

b) Clock Configuration Management Functions

PLIB_SQI_ClockDividerSet Function

Sets the SQI clock (that drives the SQI protocol) divider value. Divides the base clock to generate the SQI clock.

File

plib_sqi.h

C

```
void PLIB_SQI_ClockDividerSet(SQI_MODULE_ID index, SQI_CLK_DIV clkDivider);
```

Returns

None.

Description

This function sets the SQI clock divider value.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ClockDividerSet(MY_SQI_INSTANCE, CLK_DIV_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
clkDivider	Clock divider value

Function

void PLIB_SQI_ClockDividerSet (SQI_MODULE_ID index, SQI_CLK_DIV clkDivider)

PLIB_SQI_ClockEnable Function

Enables the SQI transfer clock.

File

plib_sqi.h

C

```
void PLIB_SQI_ClockEnable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables the SQI transfer clock (divided clock).

Remarks

SQICLK is enabled.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ClockEnable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ClockEnable (SQI_MODULE_ID index)

PLIB_SQI_ClockIsStable Function

Returns SQI transfer clock state.

File

plib_sqi.h

C

```
bool PLIB_SQI_ClockIsStable(SQI_MODULE_ID index);
```

Returns

True if clock is stable and false if it is not.

Description

This function returns the SQI transfer clock state.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool clockState = PLIB_SQI_ClockIsStable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_ClockIsStable (SQI_MODULE_ID index)

c) XIP Configuration Management Functions

PLIB_SQI_XIPAddressBytesGet Function

Returns the number of address bytes.

File

plib_sqi.h

C

```
SQI_ADDR_BYTES PLIB_SQI_XIPAddressBytesGet(SQI_MODULE_ID index);
```

Returns

Number of Address Bytes.

Description

This function returns the number of address bytes to be sent to the flash.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t addressBytes = PLIB_SQI_XIPAddressBytesGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_ADDR_BYTES PLIB_SQI_XIPAddressBytesGet (SQI_MODULE_ID index)

PLIB_SQI_XIPAddressBytesSet Function

Sets the number of address bytes.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPAddressBytesSet(SQI_MODULE_ID index, SQI_ADDR_BYTES bytes);
```

Returns

None.

Description

This function sets the number of address bytes to be sent to the flash. Typical flash address bytes are 3 (24-bit address).

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPAddressBytesSet(MY_SQI_INSTANCE, ADDR_BYTES_3);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bytes	Number of address bytes

Function

void PLIB_SQI_XIPAddressBytesSet (SQI_MODULE_ID index, SQI_ADDR_BYTES bytes)

PLIB_SQI_XIPChipSelectGet Function

Returns the current active Chip Select.

File

plib_sqi.h

C

```
SQI_CS_NUM PLIB_SQI_XIPChipSelectGet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the active Chip Select in XIP mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_CS_NUM xipCSActive = (PLIB_SQI_XIPChipSelectGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_CS_NUM PLIB_SQI_XIPChipSelectGet (SQI_MODULE_ID index)

PLIB_SQI_XIPChipSelectSet Function

Activates the Chip Select in XIP mode.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPChipSelectSet(SQI_MODULE_ID index, SQI_CS_NUM csNum);
```

Returns

None.

Description

This function sets the Chip Select that is active in XIP mode.

Remarks

None.

Preconditions

Make sure the Chip Select output enable is selected on the CS lines (PLIB_SQI_CSOutputEnableSelect).

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPChipSelectSet(MY_SQI_INSTANCE, SQI_CS_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
csNum	Chip select number

Function

void PLIB_SQI_XIPChipSelectSet (SQI_MODULE_ID index, SQI_CS_NUM csNum)

PLIB_SQI_XIPControlWord1Get Function

Get the XIP mode Control Word 1.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_XIPControlWord1Get(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the XIP mode Control Word 1.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint32_t xipCon1 = PLIB_SQI_XIPControlWord1Set (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_XIPControlWord1Get (SQI_MODULE_ID index)

PLIB_SQI_XIPControlWord1Set Function

Sets the XIP mode Control Word 1.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPControlWord1Set(SQI_MODULE_ID index, SQI_DUMMY_BYTES dummyBytes, SQI_ADDR_BYTES
addressBytes, uint8_t readOpcode, SQI_LANE_MODE dataLaneMode, SQI_LANE_MODE dummyLaneMode, SQI_LANE_MODE
modeCodeLaneMode, SQI_LANE_MODE addressLaneMode, SQI_LANE_MODE cmdLaneMode);
```

Returns

None.

Description

This function sets XIP mode Control Word 1. This function combines work of multiple PLIB APIs and can be used by the driver where complete XIP Control Word 1 is being modified.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPControlWord1Set (MY_SQI_INSTANCE,
                             DUMMY_BYTES_2,
                             ADDR_BYTES_3,
                             0x0B,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                             SQI_LANE_QUAD,
                            );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dummyBytes	Number of dummy bytes (0-7)
addressBytes	Number of address bytes (0-4)
readOpcode	Quad flash read opcode (ex: 0x0B)
dataLaneMode	Number of SQI data lanes used for sending data bytes
dummyLaneMode	Number of SQI data lanes used for sending dummy bytes
modeCodeLaneMode	Number of SQI data lanes used for sending mode code
addressLaneMode	Number of SQI data lanes used for sending address
cmdLaneMode	Number of SQI data lanes used for sending command

Function

```
void PLIB_SQI_XIPControlWord1Set (  SQI_MODULE_ID   index,
                                    SQI_DUMMY_BYTES dummyBytes,
                                    SQI_ADDR_BYTES  addressBytes,
uint8_t         readOpcode,
                                    SQI_LANE_MODE   dataLaneMode,
                                    SQI_LANE_MODE   dummyLaneMode,
                                    SQI_LANE_MODE   modeCodeLaneMode,
                                    SQI_LANE_MODE   addressLaneMode,
                                    SQI_LANE_MODE   cmdLaneMode
)
```

PLIB_SQI_XIPControlWord2Get Function

Gets the XIP mode Control Word 2.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_XIPControlWord2Get(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the XIP mode Control Word 2.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint32_t xipCon2 = PLIB_SQI_XIPControlWord2Get (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_XIPControlWord2Get (SQI_MODULE_ID index)

PLIB_SQI_XIPControlWord2Set Function

Sets the XIP mode Control Word 2.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPControlWord2Set(SQI_MODULE_ID index, SQI_CS_NUM devSelect, SQI_MODE_BYTES modeBytes,
uint8_t modeCode);
```

Returns

None.

Description

This function sets XIP mode Control Word 2. This function combines work of multiple PLIB APIs and can be used by the driver where complete XIP Control Word 2 is being modified.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPControlWord2Set (MY_SQI_INSTANCE,
                             SQI_CS_0,
                             MODE_BYTES_0,
                             0x0
                            );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
modeCode	Mode code used for supported flash devices

modeBytes	Number of mode code bytes
devSelect	Chip select for XIP mode

Function

```
void PLIB_SQI_XIPControlWord2Set ( SQI_MODULE_ID index,
                                   SQI_CS_NUM devSelect,
                                   SQI_MODE_BYTES modeBytes,
uint8_t modeCode
)
```

PLIB_SQI_XIPDummyBytesGet Function

Sets the number of dummy bytes.

File

plib_sqi.h

C

```
SQI_DUMMY_BYTES PLIB_SQI_XIPDummyBytesGet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the number of dummy bytes to be sent to the flash after the address bytes, i.e., before doing a fast read.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t dummyBytes = PLIB_SQI_XIPDummyBytesGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_DUMMY_BYTES PLIB_SQI_XIPDummyBytesGet (SQI_MODULE_ID index)

PLIB_SQI_XIPDummyBytesSet Function

Sets the number of dummy bytes.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPDummyBytesSet(SQI_MODULE_ID index, SQI_DUMMY_BYTES bytes);
```

Returns

None.

Description

This function sets the number of dummy bytes to be sent to the flash after the address bytes, i.e., before doing a fast read.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPDummyBytesSet(MY_SQI_INSTANCE, DUMMY_BYTE_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bytes	Number of dummy bytes

Function

void PLIB_SQI_XIPDummyBytesSet (SQI_MODULE_ID index, SQI_DUMMY_BYTES bytes)

PLIB_SQI_XIPLaneModeSet Function

Selects the lane (Single/Dual/Quad) mode for different transaction in XIP mode.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPLaneModeSet(SQI_MODULE_ID index, SQI_LANE_MODE dataLanes, SQI_LANE_MODE dummyLanes,
SQI_LANE_MODE modeLanes, SQI_LANE_MODE addrLanes, SQI_LANE_MODE cmdLanes);
```

Returns

None.

Description

This function selects the lane (Single/Dual/Quad) mode for different transaction in XIP mode.

Remarks

This function can't be called when in XIP mode.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPLaneModeSet(MY_SQI_INSTANCE,
                        SQI_QUAD_SINGLE,
                        SQI_QUAD_SINGLE,
                        SQI_QUAD_SINGLE,
                        SQI_QUAD_SINGLE,
                        SQI_QUAD_SINGLE
                       );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dataLanes	Data lane mode (single/dual/quad)
dummyLanes	Dummy lane mode (single/dual/quad)
modeLanes	Mode lane mode (single/dual/quad)
addrLanes	Address lane mode (single/dual/quad)
cmdLanes	Command lane mode (single/dual/quad)

Function

void PLIB_SQI_XIPLaneModeSet (SQI_MODULE_ID index,
SQI_LANE_MODE dataLanes,
SQI_LANE_MODE dummyLanes,
SQI_LANE_MODE modeLanes,
SQI_LANE_MODE addrLanes,
SQI_LANE_MODE cmdLanes
)

PLIB_SQI_XIPModeBytesGet Function

Returns the number of bytes used for mode code command.

File

plib_sqi.h

C

```
SQI_MODE_BYTES PLIB_SQI_XIPModeBytesGet(SQI_MODULE_ID index);
```

Returns

Number of Bytes used for Mode Code Command.

Description

This function returns the number of bytes for the mode code command in XIP mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_MODE_BYTES modeCodeBytes = PLIB_SQI_XIPModeBytesGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_MODE_BYTES PLIB_SQI_XIPModeBytesGet (SQI_MODULE_ID index)

PLIB_SQI_XIPModeBytesSet Function

Allocates the bytes for mode code command.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPModeBytesSet(SQI_MODULE_ID index, SQI_MODE_BYTES bytes);
```

Returns

None.

Description

This function sets the number of bytes for the mode code command in XIP mode.

Remarks

Refer to serial device data sheet for the details on this command.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPModeBytesSet(MY_SQI_INSTANCE, MODE_BYTES_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
bytes	Number of bytes of Mode code

Function

void PLIB_SQI_XIPModeBytesSet (SQI_MODULE_ID index, SQI_MODE_BYTES bytes)

PLIB_SQI_XIPModeCodeGet Function

Returns the mode code op-code.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_XIPModeCodeGet(SQI_MODULE_ID index);
```

Returns

Mode Code Opcode.

Description

This function returns the mode code command (opcode) in XIP mode for the devices that support it.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t modeCode = PLIB_SQI_XIPModeCodeGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_XIPModeCodeGet (SQI_MODULE_ID index)

PLIB_SQI_XIPModeCodeSet Function

Sets the mode code command.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPModeCodeSet(SQI_MODULE_ID index, uint8_t code);
```

Returns

None.

Description

This function sets the mode code command in XIP mode for the supported flash devices.

Remarks

Some of the devices seems to support this command, refer to specific serial device data sheet for op-code and sequence details.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPModeCodeSet(MY_SQI_INSTANCE, MY_MODE_CODE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
code	Mode code (byte)

Function

void PLIB_SQI_XIPModeCodeSet (SQI_MODULE_ID index, uint8_t code)

PLIB_SQI_XIPReadOpcodeGet Function

Returns the read op code in XIP mode.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_XIPReadOpcodeGet(SQI_MODULE_ID index);
```

Returns

Read Opcode.

Description

This function returns the read op code used in XIP mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t readOpcode = PLIB_SQI_XIPReadOpcodeGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_XIPReadOpcodeGet (SQI_MODULE_ID index)

PLIB_SQI_XIPReadOpcodeSet Function

Sets the read op code in XIP mode.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPReadOpcodeSet(SQI_MODULE_ID index, uint8_t opCode);
```

Returns

None.

Description

This function sets the flash read opcode in XIP mode. Value of read op code depends on the Flash device attached.

Remarks

Refer to the Flash device data sheet for supported read op codes.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_READ_OPCODE is the op code dependent on
// attached serial device.
PLIB_SQI_XIPReadOpcodeSet(MY_SQI_INSTANCE, MY_READ_OPCODE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
opCode	Flash read op code

Function

void PLIB_SQI_XIPReadOpcodeSet (SQI_MODULE_ID index, uint8_t opCode)

PLIB_SQI_XIPControlWord3Get Function

Gets the XIP mode Control Word 3.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_XIPControlWord3Get(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the XIP mode Control Word 3.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
```

```
uint32_t xipCon3 = PLIB_SQI_XIPControlWord3Get (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_XIPControlWord3Get (SQI_MODULE_ID index)

PLIB_SQI_XIPControlWord3Set Function

Sets the XIP mode Control Word 3.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPControlWord3Set(SQI_MODULE_ID index, bool initStatCheck, uint8_t initCmdCount,
SQI_LANE_MODE initCmdType, uint8_t initCmd3, uint8_t initCmd2, uint8_t initCmd1);
```

Returns

None.

Description

This function sets XIP mode Control Word 3. This function is used to set multiple commands in XIP mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPControlWord2Set (MY_SQI_INSTANCE,
                             0,
                             3,
                             SQI_LANE_QUAD,
                             0x06,              //WEN
                             0x99,              //RST
                             0x66               //RSTEN
                            );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
initStatCheck	Flash status check at the end of initialization commands
initCmdCount	Number of mode code bytes
initCmdType	Chip select for XIP mode
initCmd3	Command 3
initCmd2	Command 2
initCmd1	Command 1

Function

void PLIB_SQI_XIPControlWord3Set (SQI_MODULE_ID index,
bool initStatCheck,
uint8_t initCmdCount,
SQI_LANE_MODE initCmdType,
uint8_t initCmd3,
uint8_t initCmd2,

uint8_t initCmd1
)

PLIB_SQI_XIPControlWord4Get Function

Gets the XIP mode Control Word 4.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_XIPControlWord4Get(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the XIP mode Control Word 4.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint32_t xipCon4 = PLIB_SQI_XIPControlWord4Get (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_XIPControlWord4Get (SQI_MODULE_ID index)

PLIB_SQI_XIPDDRModeSet Function

Selects the rate mode (SDR/DDR) for different transactions in XIP mode.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPDDRModeSet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function selects the double data rate mode for different transactions (command, address, dummy and data etc.,) in XIP mode of operation

Remarks

This function can't be called when in XIP mode. The device is set in SDR mode if this function is not invoked.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
```

```
PLIB_SQI_XIPDDRModeSet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_XIPDDRModeSet (SQI_MODULE_ID index)

PLIB_SQI_XIPSDRCommandDDRDataGet Function

Returns the SQI command in SDR mode and Data in DDR mode bit value.

File

plib_sqi.h

C

```
bool PLIB_SQI_XIPSDRCommandDDRDataGet(SQI_MODULE_ID index);
```

Returns

Returns true if the bit is set and false if it's cleared.

Description

This function returns the command to be transferred in SDR mode and data in DDR mode bit value. Some serial flash devices require the sequence to send command in SDR mode and the rest of the data (including address bytes) in DDR mode

Remarks

This function can't be called when in XIP mode.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool cmdsdrValue = PLIB_SQI_XIPSDRCommandDDRDataGet (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_XIPSDRCommandDDRDataGet (SQI_MODULE_ID index)

PLIB_SQI_XIPSDRCommandDDRDataSet Function

Sets the SQI command in SDR mode.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPSDRCommandDDRDataSet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function sets the command to be transferred in SDR mode and data in DDR mode. Some serial flash devices require the sequence to send command in SDR mode and the rest of the data (including address bytes) in DDR mode

Remarks

This function can't be called when in XIP mode. This function has no affect if the set-up is working in single lane mode.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPSDRCommandDDRDataSet (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_XIPSDRCommandDDRDataSet (SQI_MODULE_ID index)

PLIB_SQI_XIPControlWord4Set Function

Sets the XIP mode Control Word 4.

File

plib_sqi.h

C

```
void PLIB_SQI_XIPControlWord4Set(SQI_MODULE_ID index, bool initStatCheck, uint8_t initCmdCount,
SQI_LANE_MODE initCmdType, uint8_t initCmd3, uint8_t initCmd2, uint8_t initCmd1);
```

Returns

None.

Description

This function sets XIP mode Control Word 4. This function is used to set multiple commands in XIP mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_XIPControlWord4Set (MY_SQI_INSTANCE,
                             0,
                             1,
                             SQI_LANE_QUAD,
                             0x00,
                             0x00,
                             0xC7               //Chip Erase
                            );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
initStatCheck	Flash status check at the end of initialization commands
initCmdCount	Number of mode code bytes
initCmdType	Chip select for XIP mode
initCmd3	Command 3
initCmd2	Command 2

initCmd1	Command 1

Function

void PLIB_SQI_XIPControlWord4Set (SQI_MODULE_ID index,
bool initStatCheck,
uint8_t initCmdCount,
SQI_LANE_MODE initCmdType,
uint8_t initCmd3,
uint8_t initCmd2,
uint8_t initCmd1
)

d) PIO Mode Transfer Management Functions

PLIB_SQI_ByteCountGet Function

Returns the current transmit/receive count.

File

plib_sqi.h

C

```
uint16_t PLIB_SQI_ByteCountGet(SQI_MODULE_ID index);
```

Returns

Transfer Count.

Description

This function returns the transmit/receive count, which is set by software and is actively controlled and maintained by hardware.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint16_t xferCount = PLIB_SQI_ByteCountGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
count	Transmit/Receive count

Function

uint16_t PLIB_SQI_ByteCountGet (SQI_MODULE_ID index)

PLIB_SQI_ByteCountSet Function

Sets the transmit/receive count.

File

plib_sqi.h

C

```
void PLIB_SQI_ByteCountSet(SQI_MODULE_ID index, uint16_t xferCount);
```

Returns

None.

Description

This function sets the number of bytes to transmit or receive, which is set by software and is actively controlled and maintained by hardware.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_XFER_COUNT is the transfer count.
PLIB_SQI_ByteCountSet(MY_SQI_INSTANCE, MY_XFER_COUNT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
count	Transmit/Receive count

Function

void PLIB_SQI_ByteCountSet (SQI_MODULE_ID index, uint16_t xferCount)

PLIB_SQI_ChipSelectDeassertDisable Function

Clears the Chip Select deassert.

File

plib_sqi.h

C

```
void PLIB_SQI_ChipSelectDeassertDisable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the Chip Select deassert. Chip Select stays asserted after transmission or reception of specified number of bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ChipSelectDeassertDisable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ChipSelectDeassertDisable(SQI_MODULE_ID index)

PLIB_SQI_ChipSelectDeassertEnable Function

Sets the Chip Select deassert.

File

plib_sqi.h

C

```
void PLIB_SQI_ChipSelectDeassertEnable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables Chip Select deassert. Chip Select is deasserted after transmission or reception of the specified number of bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ChipSelectDeassertEnable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_ChipSelectDeassertEnable (SQI_MODULE_ID index)

PLIB_SQI_ChipSelectGet Function

Returns the Chip Select that is currently active.

File

plib_sqi.h

C

```
SQI_CS_NUM PLIB_SQI_ChipSelectGet(SQI_MODULE_ID index);
```

Returns

Chip Select (2-bit) - Current Chip Select active (0/1).

Description

This function returns the Chip Select that is currently active.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_CS_NUM csNum = PLIB_SQI_ChipSelectGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_CS_NUM PLIB_SQI_ChipSelectGet(SQI_MODULE_ID index)

PLIB_SQI_ChipSelectSet Function

Activates the Chip Select.

File

plib_sqi.h

C

```
void PLIB_SQI_ChipSelectSet(SQI_MODULE_ID index, SQI_CS_NUM csNum);
```

Returns

None.

Description

This function sets the Chip Select to be activated on the next transaction.

Remarks

None.

Preconditions

Make sure the Chip Select output enable is selected on the CS lines (PLIB_SQI_CSOutputEnableSelect).

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and SQI_CS_NUM_0 is the enum element.
PLIB_SQI_ChipSelectSet(MY_SQI_INSTANCE, SQI_CS_NUM_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
csNum	Chip select number

Function

void PLIB_SQI_ChipSelectSet (SQI_MODULE_ID index, SQI_CS_NUM csNum)

PLIB_SQI_ConfigWordGet Function

Gets the SQI Configuration Word.

File

plib_sqi.h

C

```
uint32_t PLIB_SQI_ConfigWordGet(SQI_MODULE_ID index);
```

Returns

None.

Description

This function returns the SQI Configuration Word.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint32_t sqiCfg = PLIB_SQI_ConfigWordGet (MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_SQI_ConfigWordGet (SQI_MODULE_ID index)

PLIB_SQI_ConfigWordSet Function

Sets SQI Configuration Word.

File

plib_sqi.h

C

```
void PLIB_SQI_ConfigWordSet(SQI_MODULE_ID index, bool sqiEnable, SQI_CS_OEN csPins, SQI_DATA_OEN dataPins,
bool reset, bool burstEn, bool hold, bool writeProtect, bool rxLatch, SQI_DATA_FORMAT lsbf, SQI_DATA_MODE
dataMode, SQI_XFER_MODE xferMode);
```

Returns

None.

Description

This function sets the SQI Configuration Word. This function is a combination of several function in case the driver plans to write the complete Configuration Word.

Remarks

Chip select is not actually asserted, only enabled to be asserted.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_ConfigWordSet(MY_SQI_INSTANCE,
                       1,
                       SQI_CS_OEN_0,
                       SQI_DATA_OEN_QUAD,
                       0,
                       1,
                       0,
                       0,
                       0,
                       SQI_DATA_FORMAT_LSBF,
                       SQI_DATA_MODE_3,
                       SQI_XFER_MODE_PIO
                       );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
sqiEnable	Enables/Disables the SQI module
csPins	Chip Select Output Enable

dataPins	Data Output Enable
reset	Resets control, transmit, receive buffers and state machines
burstEn	Burst Enable (always set to '1')
hold	SQID2 to act as HOLD# signal in single and dual lane modes
writeProtect	SQID3 to act as WP# signal in single and dual lane modes
rxLatch	Activates receive latch in transmit mode
lsbf	Sets data endian mode to least significant bit first (LSBF)
dataMode	Sets data mode to mode 0/mode 1/Serial Flash mode
xferMode	Sets transfer mode to XIP/DMA/PIO mode

Function

void PLIB_SQI_ConfigWordSet(SQI_MODULE_ID index,
bool sqiEnable,
SQI_CS_OEN csPins,
SQI_DATA_OEN dataPins,
bool reset,
bool burstEn,
bool hold,
bool writeProtect,
bool rxLatch,
SQI_DATA_FORMAT lsbf,
SQI_DATA_MODE dataMode,
SQI_XFER_MODE xferMode
)

PLIB_SQI_ReceiveBufferIsUnderrun Function

Returns the status of receive buffer.

File

plib_sqi.h

C

```
bool PLIB_SQI_ReceiveBufferIsUnderrun(SQI_MODULE_ID index);
```

Returns

- true - Receive Buffer is underrun
- false - Receive Buffer is not underrun

Description

This function returns the status of the receive buffer.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool rxun = PLIB_SQI_ReceiveBufferIsUnderrun(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_ReceiveBufferIsUnderrun (SQI_MODULE_ID index)

PLIB_SQI_RxBufferThresholdGet Function

Returns the receive command threshold.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_RxBufferThresholdGet(SQI_MODULE_ID index);
```

Returns

Receive Buffer Threshold value.

Description

This function returns the receive command threshold that is used to monitor receives based on the receive buffer space availability.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t rxBufThres = PLIB_SQI_RxCommandThresholdGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_RxBufferThresholdGet (SQI_MODULE_ID index)

PLIB_SQI_RxBufferThresholdIntGet Function

Sets the receive buffer threshold interrupt.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_RxBufferThresholdIntGet(SQI_MODULE_ID index);
```

Returns

Receive Buffer Threshold value (used to trigger an interrupt).

Description

This function returns the receive buffer threshold used to set an interrupt.

Remarks

This is a 5-bit field and bits 7, 6, and 5 are ignored in the char.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t rxBufIntThres = PLIB_SQI_RxBufferThresholdIntGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Receive interrupt (buffer) threshold

Function

uint8_t PLIB_SQI_RxBufferThresholdIntGet (SQI_MODULE_ID index)

PLIB_SQI_RxBufferThresholdIntSet Function

Sets the receive buffer threshold for interrupt.

File

plib_sqi.h

C

```
void PLIB_SQI_RxBufferThresholdIntSet(SQI_MODULE_ID index, uint8_t threshold);
```

Returns

None.

Description

This function sets the receive buffer threshold used to trigger an interrupt. Sets an interrupt condition when receive buffer count is larger than or equal to the receive interrupt threshold bytes.

Remarks

This is a 5-bit field and bits 7,6,5 are ignored in the char.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_RX_INT_THRESHOLD is the threshold value.
PLIB_SQI_RxBufferThresholdIntSet(MY_SQI_INSTANCE, MY_RX_INT_THRESHOLD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Receive buffer threshold for interrupt

Function

void PLIB_SQI_RxBufferThresholdIntSet (SQI_MODULE_ID index, uint8_t threshold)

PLIB_SQI_RxBufferThresholdSet Function

Sets the receive command threshold.

File

plib_sqi.h

C

```
void PLIB_SQI_RxBufferThresholdSet(SQI_MODULE_ID index, uint8_t threshold);
```

Returns

None.

Description

This function sets the receive command threshold that is used to monitor receives based on the receive buffer space availability.

Remarks

Valid threshold values are 0-31.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_RECEIVE_THRESHOLD is the receive threshold value.
PLIB_SQI_RxCommandThresholdSet(MY_SQI_INSTANCE, MY_RECEIVE_THRESHOLD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Receive command (buffer) threshold

Function

void PLIB_SQI_RxBufferThresholdSet (SQI_MODULE_ID index, uint8_t threshold)

PLIB_SQI_TransferDirectionGet Function

Returns the transfer command.

File

plib_sqi.h

C

```
SQI_XFER_CMD PLIB_SQI_TransferDirectionGet(SQI_MODULE_ID index);
```

Returns

Transfer Command (Idle/Receive/Transmit).

Description

This function returns the transfer command that is active currently.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
SQI_XFER_CMD xferDirection = PLIB_SQI_TransferDirectionGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_XFER_CMD PLIB_SQI_TransferDirectionGet (SQI_MODULE_ID index)

PLIB_SQI_TransferDirectionSet Function

Sets the transfer command.

File

plib_sqi.h

C

```
void PLIB_SQI_TransferDirectionSet(SQI_MODULE_ID index, SQI_XFER_CMD command);
```

Returns

None.

Description

This function sets the transfer command to Idle/Transmit/Receive.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and SQI_CMD_TRANSMIT is an enum element.
PLIB_SQI_TransferDirectionSet(MY_SQI_INSTANCE, SQI_CMD_TRANSMIT);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
command	Transfer command

Function

void PLIB_SQI_TransferDirectionSet (SQI_MODULE_ID index, SQI_XFER_CMD command)

PLIB_SQI_TransmitBufferFreeSpaceGet Function

Returns the number of transmit buffer words available.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_TransmitBufferFreeSpaceGet(SQI_MODULE_ID index);
```

Returns

Amount of transmit buffer space free in bytes.

Description

This function returns the number of transmit buffer bytes available.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t txBufFreeSpace = PLIB_SQI_TransmitBufferFreeSpaceGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_TransmitBufferFreeSpaceGet(SQI_MODULE_ID index)

PLIB_SQI_TransmitBufferHasOverflowed Function

Returns the current status of the transmit buffer.

File

plib_sqi.h

C

```
bool PLIB_SQI_TransmitBufferHasOverflowed(SQI_MODULE_ID index);
```

Returns

- true - Transmit Buffer has overflowed
- false - Transmit Buffer has not overflowed

Description

This function returns the current state of the transmit buffer.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool txOv = PLIB_SQI_TransmitBufferHasOverflowed(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_TransmitBufferHasOverflowed (SQI_MODULE_ID index)

PLIB_SQI_TxBufferThresholdGet Function

Returns the transmit command threshold value.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_TxBufferThresholdGet(SQI_MODULE_ID index);
```

Returns

Transmit buffer threshold value.

Description

This function returns the transmit command threshold value that is used to monitor transmits based on the transmit buffer space availability.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t txThreshold = PLIB_SQI_TxBufferThresholdGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_TxBufferThresholdGet(SQI_MODULE_ID index)

PLIB_SQI_TxBufferThresholdIntGet Function

Returns the value to trigger the transmit buffer threshold interrupt.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_TxBufferThresholdIntGet(SQI_MODULE_ID index);
```

Returns

Transmit buffer threshold for interrupt.

Description

This function returns the transmit buffer threshold used to set an interrupt. When enabled, interrupt is triggered when transmit buffer has more space than the transmit interrupt threshold bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t txBufIntThres = PLIB_SQI_TxBufferThresholdIntGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_TxBufferThresholdIntGet (SQI_MODULE_ID index)

PLIB_SQI_TxBufferThresholdIntSet Function

Sets the value to trigger the transmit buffer threshold interrupt.

File

plib_sqi.h

C

```
void PLIB_SQI_TxBufferThresholdIntSet(SQI_MODULE_ID index, uint8_t threshold);
```

Returns

None.

Description

This function sets the transmit buffer threshold used for an interrupt. When enabled, interrupt is triggered when transmit buffer has more space than the transmit interrupt threshold bytes.

Remarks

This is a 5-bit field and bits 7,6,5 are ignored in the char.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_TX_INT_THRESHOLD is the threshold value.
PLIB_SQI_TxBufferThresholdIntSet(MY_SQI_INSTANCE, MY_TX_INT_THRESHOLD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Transmit interrupt (buffer) threshold

Function

void PLIB_SQI_TxBufferThresholdIntSet (SQI_MODULE_ID index, uint8_t threshold)

PLIB_SQI_TxBufferThresholdSet Function

Sets the transmit command threshold.

File

plib_sqi.h

C

```
void PLIB_SQI_TxBufferThresholdSet(SQI_MODULE_ID index, uint8_t threshold);
```

Returns

None.

Description

This function sets the transmit command threshold, which is used to control transmits based on the transmit buffer space availability.

Remarks

Valid threshold values are 0-31.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and MY_TRANSMIT_THRESHOLD is the threshold value.
PLIB_SQI_TxBufferThresholdSet(MY_SQI_INSTANCE, MY_TRANSMIT_THRESHOLD);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
threshold	Transmit command (buffer) threshold

Function

void PLIB_SQI_TxBufferThresholdSet(SQI_MODULE_ID index, uint8_t threshold)

e) Interrupt Control and Status Management Functions

PLIB_SQI_InterruptDisable Function

Disables the interrupt source.

File

plib_sqi.h

C

```
void PLIB_SQI_InterruptDisable(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function disables the interrupt source passed into the function.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer and TXFULL is an enum element.
PLIB_SQI_InterruptDisable(MY_SQI_INSTANCE, SQI_TXFULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt to be disabled

Function

void PLIB_SQI_InterruptDisable (SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_InterruptEnable Function

Enables the passed interrupt source.

File

plib_sqi.h

C

```
void PLIB_SQI_InterruptEnable(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function enables the interrupt source passed into the function.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_InterruptEnable(MY_SQI_INSTANCE, SQI_TXFULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt to be enabled

Function

void PLIB_SQI_InterruptEnable (SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_InterruptFlagGet Function

Return SQI Interrupt flag status.

File

plib_sqi.h

C

```
bool PLIB_SQI_InterruptFlagGet(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

Interrupt status.

Description

This function returns the SQI interrupt source flag status (set/cleared).

Remarks

None.

Preconditions

None.

Example

```
if ( PLIB_SQI_InterruptFlagGet(MY_SQI_INSTANCE,SQI_INT_ANY) )
    if ( PLIB_SQI_InterruptFlagGet(MY_SQI_INSTANCE,SQI_TXFULL) )
    {
       ...
    }
    if ( PLIB_SQI_InterruptFlagGet(MY_SQI_INSTANCE,SQI_RXFULL) )
    {
        ...
    }
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt flag of interest

Function

bool PLIB_SQI_InterruptFlagGet(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_InterruptIsEnabled Function

Returns the interrupt state.

File

plib_sqi.h

C

```
bool PLIB_SQI_InterruptIsEnabled(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

- true - Interrupt is enabled
- false - Interrupt is disabled

Description

This function returns whether or not the interrupt state is enabled or disabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
if (PLIB_SQI_InterruptIsEnabled(MY_SQI_INSTANCE, SQI_TXFULL))
{
    ..
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt under check

Function

bool PLIB_SQI_InterruptIsEnabled (SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_InterruptSignalDisable Function

Disables the interrupt signal source.

File

plib_sqi.h

C

```
void PLIB_SQI_InterruptSignalDisable(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function disables the interrupt signals source passed into the function, thus prohibiting it from reaching to the external interrupt controller.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_InterruptSignalDisable(MY_SQI_INSTANCE, SQI_TXFULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt to be disabled

Function

void PLIB_SQI_InterruptSignalDisable (SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_InterruptSignalEnable Function

Enables the passed interrupt signal source.

File

plib_sqi.h

C

```
void PLIB_SQI_InterruptSignalEnable(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function enables the interrupt signal source to be passed into the function, which allows it to go out to the external Interrupt Controller.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_InterruptSignalEnable(MY_SQI_INSTANCE, SQI_TXFULL);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt to be enabled

Function

void PLIB_SQI_InterruptSignalEnable (SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_InterruptSignalIsEnabled Function

Returns the interrupt signal state.

File

plib_sqi.h

C

```
bool PLIB_SQI_InterruptSignalIsEnabled(SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag);
```

Returns

- true - Interrupt signal is enabled
- false - Interrupt signal is disabled

Description

This function returns whether the interrupt signal state is enabled or disabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
```

```
if (PLIB_SQI_InterruptSignalIsEnabled(MY_SQI_INSTANCE, SQI_TXFULL))
{
    ..
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptFlag	Interrupt signal under check

Function

bool PLIB_SQI_InterruptSignalIsEnabled (SQI_MODULE_ID index, SQI_INTERRUPTS interruptFlag)

PLIB_SQI_DataLineStatus Function

Returns the logical status of the SQI data lines.

File

plib_sqi.h

C

```
bool PLIB_SQI_DataLineStatus(SQI_MODULE_ID index, uint8_t dataPin);
```

Returns

SQIDx Status (High/Low).

Description

This function returns the logical status of the data lines (0/1).

Remarks

Parsing values other than 0/1/2/3 returns SQID0 pin status.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool sqiDataLineStatus = PLIB_SQI_DataLineStatus(MY_SQI_INSTANCE, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
dataPin	Data pin for which status will be returned (0/1/2/3)

Function

bool PLIB_SQI_DataLineStatus(SQI_MODULE_ID index, uint8_t dataPin)

PLIB_SQI_CommandStatusGet Function

Returns the SQI command (transmit/receive/idle).

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_CommandStatusGet(SQI_MODULE_ID index);
```

Returns

SQI command (transmit/receive/idle) that is currently being executed.

Description

This function returns the SQI command (transmit/receive/idle) that is currently being executed.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t cmdStat = PLIB_SQI_CommandStatusGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_CommandStatusGet(SQI_MODULE_ID index)

PLIB_SQI_ConBufWordsAvailable Function

Returns the number of control buffer words available.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_ConBufWordsAvailable(SQI_MODULE_ID index);
```

Returns

Number of words available.

Description

This function returns the number of control buffer words available.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t numConBuf = PLIB_SQI_ConBufWordsAvailable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_ConBufWordsAvailable(SQI_MODULE_ID index)

f) DMA Buffer Address Management Functions

PLIB_SQI_DMABDBaseAddressGet Function

Returns the address of the base buffer descriptor.

File

plib_sqi.h

C

```
void* PLIB_SQI_DMABDBaseAddressGet(SQI_MODULE_ID index);
```

Returns

Base Buffer Descriptor address.

Description

This function returns the address of the base DMA buffer descriptor.

Remarks

Check to make sure if DMA Buffer Descriptor fetch is in progress.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
void *baseBDAddress;

If (!PLIB_SQI_DMAIsActive(MY_SQI_INSTANCE))
{
   baseBDAddress = PLIB_SQI_DMABDBaseAddressGet(MY_SQI_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void * PLIB_SQI_DMABDBaseAddressGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDBaseAddressSet Function

Sets the address of the base buffer descriptor.

File

plib_sqi.h

C

```
void PLIB_SQI_DMABDBaseAddressSet(SQI_MODULE_ID index, void * baseBDAddress);
```

Returns

None.

Description

This function writes the address of the base (first/only) buffer descriptor into the buffer descriptor base address register.

Remarks

Check to make sure if DMA Buffer Descriptor fetch is in progress.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
if (!PLIB_SQI_DMAIsActive(MY_SQI_INSTANCE))
    PLIB_SQI_DMABDBaseAddressSet(MY_SQI_INSTANCE, (void *) (&MY_BD_STRUCT));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baseBDAddress	Base buffer descriptor address

Function

void PLIB_SQI_DMABDBaseAddressSet (SQI_MODULE_ID index, void *baseBDAddress)

PLIB_SQI_DMABDCurrentAddressGet Function

Returns the address of the current buffer descriptor in process.

File

plib_sqi.h

C

```
void* PLIB_SQI_DMABDCurrentAddressGet(SQI_MODULE_ID index);
```

Returns

Current Buffer Descriptor Address.

Description

This function returns the address of the DMA buffer descriptor that is currently in progress.

Remarks

Check to make sure if DMA Buffer Descriptor fetch is in progress.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint32_t currentBDAddress;

If (PLIB_SQI_DMAIsActive(MY_SQI_INSTANCE))
{
   void* currentBDAddress = PLIB_SQI_DMABDCurrentAddressGet(MY_SQI_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void* PLIB_SQI_DMABDCurrentAddressGet (SQI_MODULE_ID index)

g) DMA Buffer Descriptor Management Functions

PLIB_SQI_DMABDControlWordGet Function

Returns Current Buffer Descriptor Control Word Information.

File

plib_sqi.h

C

```
uint16_t PLIB_SQI_DMABDControlWordGet(SQI_MODULE_ID index);
```

Returns

Control Word - DMA Buffer Descriptor Control Word

Description

This function returns current buffer descriptor Control Word information excluding buffer length. This information is returned in transmit and receive status functions.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint16_t dmabdconword;

dmabdconword =  PLIB_SQI_DMABDControlWordGet(MY_SQI_INSTANCE);

switch (dmabdconword)
{
    case BD_ENABLED: ...;
    case BD_DISABLED: ...;
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint16_t PLIB_SQI_DMABDControlWordGet (SQI_MODULE_ID index)

PLIB_SQI_DMABDReceiveBufferCountGet Function

Returns the receive buffer space in bytes.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_DMABDReceiveBufferCountGet(SQI_MODULE_ID index);
```

Returns

Receive buffer space in bytes.

Description

This function returns the current receive buffer space in bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdRxBufCount = PLIB_SQI_DMABDReceiveBufferCountGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_DMABDReceiveBufferCountGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDReceiveBufferLengthGet Function

Returns the receive length in bytes.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_DMABDReceiveBufferLengthGet(SQI_MODULE_ID index);
```

Returns

Receive buffer space in bytes.

Description

This function returns the current receive length in bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdRxBufLength = PLIB_SQI_DMABDReceiveBufferLengthGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_DMABDReceiveBufferLengthGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDReceiveStateGet Function

Returns the current state of the buffer descriptor in progress.

File

plib_sqi.h

C

```
SQI_BD_STATE PLIB_SQI_DMABDReceiveStateGet(SQI_MODULE_ID index);
```

Returns

Status - DMA Buffer Descriptor State

Description

This function returns the current state of the buffer descriptor in progress.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdrxState = PLIB_SQI_DMABDReceiveStateGet(MY_SQI_INSTANCE);

switch (bdRxState)
{
   case BD_IDLE: ...;
   case BD_STATE_FETCH_REQ_PENDING: ...;
   .
   .
   .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_BD_STATE uint8_t PLIB_SQI_DMABDReceiveStateGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDTransmitBufferCountGet Function

Returns the transmit buffer space in bytes.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_DMABDTransmitBufferCountGet(SQI_MODULE_ID index);
```

Returns

Transmit buffer space in bytes.

Description

This function returns the current transmit buffer space in bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdTxBufCount = PLIB_SQI_DMABDTransmitBufferCountGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_DMABDTransmitBufferCountGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDTransmitBufferLengthGet Function

Returns the transmit length in bytes.

File

plib_sqi.h

C

```
uint8_t PLIB_SQI_DMABDTransmitBufferLengthGet(SQI_MODULE_ID index);
```

Returns

Transmit buffer space in bytes.

Description

This function returns the current transmit length in bytes.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdTxBufLength = PLIB_SQI_DMABDTransmitBufferLengthGet(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_SQI_DMABDTransmitBufferLengthGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDTransmitStateGet Function

Returns the current state of the buffer descriptor in progress.

File

plib_sqi.h

C

```
SQI_BD_STATE PLIB_SQI_DMABDTransmitStateGet(SQI_MODULE_ID index);
```

Returns

Status - DMA Buffer Descriptor State

Description

This function returns the current state of the buffer descriptor in progress.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
```

```
// application developer.
uint8_t bdTxState = PLIB_SQI_DMABDTransmitStateGet(MY_SQI_INSTANCE);

switch (bdTxState)
{
   case BD_IDLE: ...;
   case BD_FETCH_REQ_PENDING: ...;
   .
   .
   .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_BD_STATE PLIB_SQI_DMABDTransmitStateGet(SQI_MODULE_ID index)

PLIB_SQI_DMABDFetchStop Function

Stops the DMA buffer descriptor fetch process.

File

plib_sqi.h

C

```
void PLIB_SQI_DMABDFetchStop(SQI_MODULE_ID index);
```

Returns

None.

Description

This function stops the DMA buffer descriptor fetch process.

Remarks

None.

Preconditions

PLIB_SQI_DMABDFetchStart is called.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMABDFetchStop(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DMABDFetchStop (SQI_MODULE_ID index)

h) DMA Control and Status Management Functions

PLIB_SQI_DMABDFetchStart Function

Starts the DMA buffer descriptor fetch process.

File

plib_sqi.h

C

```
void PLIB_SQI_DMABDFetchStart(SQI_MODULE_ID index);
```

Returns

None.

Description

This function starts the DMA buffer descriptor fetch process.

Remarks

None.

Preconditions

Make sure the buffer descriptors are set up and the buffer descriptor base address register is pointing to the first/only buffer descriptor. Also ensure any previous buffer descriptor processing is fixed.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMABDFetchStart(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DMABDFetchStart (SQI_MODULE_ID index)

PLIB_SQI_DMABDIsBusy Function

Returns whether or not the DMA Buffer Descriptor is busy.

File

plib_sqi.h

C

```
bool PLIB_SQI_DMABDIsBusy(SQI_MODULE_ID index);
```

Returns

- true - DMA Buffer Descriptor is busy
- false - DMA Buffer Descriptor is not busy

Description

This function returns whether or not the DMA buffer descriptor process is busy.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool bdBusy =  PLIB_SQI_DMABDIsBusy(MY_SQI_INSTANCE)
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_DMABDIsBusy (SQI_MODULE_ID index)

PLIB_SQI_DMABDPollCounterSet Function

Sets the poll counter value.

File

plib_sqi.h

C

```
void PLIB_SQI_DMABDPollCounterSet(SQI_MODULE_ID index, uint16_t pollCount);
```

Returns

None.

Description

This function sets the poll counter value that indicates the number of cycles the DMA would wait before fetching the next descriptor word, if the current descriptor fetched was disabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
if (PLIB_SQI_DMABDPollIsEnabled(MY_SQI_INSTANCE)
{
    PLIB_SQI_DMABDPollCounterSet(MY_SQI_INSTANCE, MY_POLL_VALUE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pollControl	Polling value

Function

void PLIB_SQI_DMABDPollCounterSet(SQI_MODULE_ID index, uint16_t pollCount)

PLIB_SQI_DMABDPollDisable Function

Disables the buffer descriptor polling.

File

plib_sqi.h

C

```
void PLIB_SQI_DMABDPollDisable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the buffer descriptor polling.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMABDPollDisable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DMABDPollDisable (SQI_MODULE_ID index)

PLIB_SQI_DMABDPollEnable Function

Enables the buffer descriptor polling.

File

plib_sqi.h

C

```
void PLIB_SQI_DMABDPollEnable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables the buffer descriptor polling and works in tandem with poll control register.

Remarks

Enable this control bit only when you are planning to have dead descriptors in the linked list.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMABDPollEnable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DMABDPollEnable (SQI_MODULE_ID index)

PLIB_SQI_DMABDPollIsEnabled Function

Returns whether or not the DMA buffer descriptor poll is enabled.

File

plib_sqi.h

C

```
bool PLIB_SQI_DMABDPollIsEnabled(SQI_MODULE_ID index);
```

Returns

- true - The DMA Poll Control is enabled

- false - The DMA Poll Control is disabled

Description

This function returns whether or not the DMA buffer descriptor poll is enabled or disabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
If (PLIB_SQI_DMABDPollIsEnabled(MY_SQI_INSTANCE))
{
   PLIB_SQI_PollControlSet(MY_SQI_INSTANCE, MY_POLL_CONTROL_VALUE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_DMABDPollIsEnabled (SQI_MODULE_ID index)

PLIB_SQI_DMABDStateGet Function

Returns the current state of the buffer descriptor in progress.

File

plib_sqi.h

C

```
SQI_BD_STATE PLIB_SQI_DMABDStateGet(SQI_MODULE_ID index);
```

Returns

Status - DMA Buffer Descriptor State

Description

This function returns the current state of the buffer descriptor in progress.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
uint8_t bdState = PLIB_SQI_DMABDStateGet(MY_SQI_INSTANCE);

switch (bdState)
{
   case BD_IDLE: ...;
   case BD_FETCH_REQ_PENDING: ...;
   .
   .
   .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

SQI_BD_STATE PLIB_SQI_DMABDStateGet(SQI_MODULE_ID index)

PLIB_SQI_DMADisable Function

Disables the built-in DMA logic.

File

plib_sqi.h

C

```
void PLIB_SQI_DMADisable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function disables the built-in DMA logic for data transfer.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMADisable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DMADisable (SQI_MODULE_ID index)

PLIB_SQI_DMAEnable Function

Enables the built-in DMA logic.

File

plib_sqi.h

C

```
void PLIB_SQI_DMAEnable(SQI_MODULE_ID index);
```

Returns

None.

Description

This function enables the built-in DMA logic for data transfer.

Remarks

None.

Preconditions

DMA buffer descriptors need to be setup before enabling the DMA.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DMAEnable(MY_SQI_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_SQI_DMAEnable (SQI_MODULE_ID index)

PLIB_SQI_DMAHasStarted Function

Returns whether or no the DMA process has started.

File

plib_sqi.h

C

```
bool PLIB_SQI_DMAHasStarted(SQI_MODULE_ID index);
```

Returns

- true - The DMA process has started
- false - The DMA process has not started

Description

This function returns whether or not the DMA process has started.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
bool dmaStarted =  PLIB_SQI_DMAHasStarted (MY_SQI_INSTANCE)
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_DMAHasStarted (SQI_MODULE_ID index)

PLIB_SQI_DMAIsEnabled Function

Returns whether or not DMA is enabled.

File

plib_sqi.h

C

```
bool PLIB_SQI_DMAIsEnabled(SQI_MODULE_ID index);
```

Returns

- true - DMA is enabled
- false - DMA is disabled

Description

This function returns whether or not the DMA is enabled or disabled.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
If (PLIB_SQI_DMAIsEnabled(MY_SQI_INSTANCE))
{
   ...
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_SQI_DMAIsEnabled (SQI_MODULE_ID index)

i) Other Functions

PLIB_SQI_StatusCheckSet Function

Sets the Flash status check related control bits.

File

plib_sqi.h

C

```
void PLIB_SQI_StatusCheckSet(SQI_MODULE_ID index, uint16_t statCheckCmd, uint8_t numStatBytes,
SQI_LANE_MODE statCmdType, bool statBitPos);
```

Returns

None.

Description

This function sets the Flash status check related control bits and enables the status check for PIO mode.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_SQI_INSTANCE, is the SQI instance selected for use by the
// application developer.
PLIB_SQI_DDRModeSet(MY_SQI_INSTANCE,
                    0x05,
                    1,
                    SQI_LANE_QUAD,
                    0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
statCheckCmd	Status check command to be sent to the Flash
numStatBytes	Number of status command bytes
statCmdType	Lane mode (Single/Dual/Quad) for status command

Function

void PLIB_SQI_StatusCheckSet(SQI_MODULE_ID index,
uint16_t statCheckCmd,
uint8_t numStatBytes,
SQI_LANE_MODE statCmdType,
bool statBitPos
)

j) Feature Existence Functions

PLIB_SQI_ExistsBDBaseAddress Function

Identifies whether the BDBaseAddress feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDBaseAddress(SQI_MODULE_ID index);
```

Returns

- true - The BDBaseAddress feature is supported on the device
- false - The BDBaseAddress feature is not supported on the device

Description

This function identifies whether the BDBaseAddress feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_DMABDBaseAddressSet
- PLIB_SQI_DMABDBaseAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDBaseAddress(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDControlWord Function

Identifies whether the BDControlWord feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDControlWord(SQI_MODULE_ID index);
```

Returns

- true - The BDControlWord feature is supported on the device
- false - The BDControlWord feature is not supported on the device

Description

This function identifies whether the BDControlWord feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDControlWordGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDControlWord(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDCurrentAddress Function

Identifies whether the BDCurrentAddress feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDCurrentAddress(SQI_MODULE_ID index);
```

Returns

- true - The BDCurrentAddress feature is supported on the device
- false - The BDCurrentAddress feature is not supported on the device

Description

This function identifies whether the BDCurrentAddress feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDCurrentAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDCurrentAddress(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDPollCount Function

Identifies whether the BDPollCount feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDPollCount(SQI_MODULE_ID index);
```

Returns

- true - The BDPollCount feature is supported on the device
- false - The BDPollCount feature is not supported on the device

Description

This function identifies whether the BDPollCount feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDPollCounterSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDPollCount(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDPollingEnable Function

Identifies whether the BDPollingEnable feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDPollingEnable(SQI_MODULE_ID index);
```

Returns

- true - The BDPollingEnable feature is supported on the device
- false - The BDPollingEnable feature is not supported on the device

Description

This function identifies whether the BDPollingEnable feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_DMABDPollEnable
- PLIB_SQI_DMABDPollDisable
- PLIB_SQI_DMABDPollIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDPollingEnable(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDProcessState Function

Identifies whether the BDProcessState feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDProcessState(SQI_MODULE_ID index);
```

Returns

- true - The BDProcessState feature is supported on the device
- false - The BDProcessState feature is not supported on the device

Description

This function identifies whether the BDProcessState feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDProcessState(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDRxBufCount Function

Identifies whether the BDRxBufCount feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDRxBufCount(SQI_MODULE_ID index);
```

Returns

- true - The BDRxBufCount feature is supported on the device
- false - The BDRxBufCount feature is not supported on the device

Description

This function identifies whether the BDRxBufCount feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDReceiveBufferCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDRxBufCount(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDRxLength Function

Identifies whether the BDRxLength feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDRxLength(SQI_MODULE_ID index);
```

Returns

- true - The BDRxLength feature is supported on the device
- false - The BDRxLength feature is not supported on the device

Description

This function identifies whether the BDRxLength feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDReceiveBufferLengthGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDRxLength(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDRxState Function

Identifies whether the BDRxState feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDRxState(SQI_MODULE_ID index);
```

Returns

- true - The BDRxState feature is supported on the device
- false - The BDRxState feature is not supported on the device

Description

This function identifies whether the BDRxState feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDReceiveStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDRxState(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDTxBufCount Function

Identifies whether the BDTxBufCount feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDTxBufCount(SQI_MODULE_ID index);
```

Returns

- true - The BDTxBufCount feature is supported on the device
- false - The BDTxBufCount feature is not supported on the device

Description

This function identifies whether the BDTxBufCount feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDTransmitBufferCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDTxBufCount(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDTxLength Function

Identifies whether the BDTxLength feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDTxLength(SQI_MODULE_ID index);
```

Returns

- true - The BDTxLength feature is supported on the device
- false - The BDTxLength feature is not supported on the device

Description

This function identifies whether the BDTxLength feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDTransmitBufferLengthGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDTxLength(SQI_MODULE_ID index)

PLIB_SQI_ExistsBDTxState Function

Identifies whether the BDTxState feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBDTxState(SQI_MODULE_ID index);
```

Returns

- true - The BDTxState feature is supported on the device
- false - The BDTxState feature is not supported on the device

Description

This function identifies whether the BDTxState feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDTransmitStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBDTxState(SQI_MODULE_ID index)

PLIB_SQI_ExistsBurstControl Function

Identifies whether the BurstControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsBurstControl(SQI_MODULE_ID index);
```

Returns

- true - The BurstControl feature is supported on the device
- false - The BurstControl feature is not supported on the device

Description

This function identifies whether the BurstControl feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_BurstEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsBurstControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsChipSelect Function

Identifies whether the ChipSelect feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsChipSelect(SQI_MODULE_ID index);
```

Returns

- true - The ChipSelect feature is supported on the device
- false - The ChipSelect feature is not supported on the device

Description

This function identifies whether the ChipSelect feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ChipSelectSet
- PLIB_SQI_ChipSelectGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsChipSelect(SQI_MODULE_ID index)

PLIB_SQI_ExistsClockControl Function

Identifies whether the ClockControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsClockControl(SQI_MODULE_ID index);
```

Returns

- true - The ClockControl feature is supported on the device
- false - The ClockControl feature is not supported on the device

Description

This function identifies whether the ClockControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ClockEnable
- PLIB_SQI_ClockDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsClockControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsClockDivider Function

Identifies whether the ClockDivider feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsClockDivider(SQI_MODULE_ID index);
```

Returns

- true - The ClockDivider feature is supported on the device
- false - The ClockDivider feature is not supported on the device

Description

This function identifies whether the ClockDivider feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_ClockDividerSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsClockDivider(SQI_MODULE_ID index)

PLIB_SQI_ExistsClockReady Function

Identifies whether the ClockReady feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsClockReady(SQI_MODULE_ID index);
```

Returns

- true - The ClockReady feature is supported on the device
- false - The ClockReady feature is not supported on the device

Description

This function identifies whether the ClockReady feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_ClockIsStable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsClockReady(SQI_MODULE_ID index)

PLIB_SQI_ExistsConBufThreshold Function

Identifies whether the ConBufThreshold feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsConBufThreshold(SQI_MODULE_ID index);
```

Returns

- true - The ConBufThreshold feature is supported on the device
- false - The ConBufThreshold feature is not supported on the device

Description

This function identifies whether the ConBufThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ControlBufferThresholdSet
- PLIB_SQI_ControlBufferThresholdGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsConBufThreshold(SQI_MODULE_ID index)

PLIB_SQI_ExistsConfigWord Function

Identifies whether the ConfigWord feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsConfigWord(SQI_MODULE_ID index);
```

Returns

- true - The ConfigWord feature is supported on the device
- false - The ConfigWord feature is not supported on the device

Description

This function identifies whether the ConfigWord feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ConfigWordSet
- PLIB_SQI_ConfigWordGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsConfigWord(SQI_MODULE_ID index)

PLIB_SQI_ExistsControlWord Function

Identifies whether the ControlWord feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsControlWord(SQI_MODULE_ID index);
```

Returns

- true - The ControlWord feature is supported on the device
- false - The ControlWord feature is not supported on the device

Description

This function identifies whether the ControlWord feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ControlWordSet
- PLIB_SQI_ControlWordGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsControlWord(SQI_MODULE_ID index)

PLIB_SQI_ExistsCSDeassert Function

Identifies whether the CSDeassert feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsCSDeassert(SQI_MODULE_ID index);
```

Returns

- true - The CSDeassert feature is supported on the device
- false - The CSDeassert feature is not supported on the device

Description

This function identifies whether the CSDeassert feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ChipSelectDeassertEnable
- PLIB_SQI_ChipSelectDeassertDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsCSDeassert(SQI_MODULE_ID index)

PLIB_SQI_ExistsCSOutputEnable Function

Identifies whether the CSOutputEnable feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsCSOutputEnable(SQI_MODULE_ID index);
```

Returns

- true - The CSOutputEnable feature is supported on the device
- false - The CSOutputEnable feature is not supported on the device

Description

This function identifies whether the CSOutputEnable feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_CSOutputEnableSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsCSOutputEnable(SQI_MODULE_ID index)

PLIB_SQI_ExistsDataFormat Function

Identifies whether the DataFormat feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDataFormat(SQI_MODULE_ID index);
```

Returns

- true - The DataFormat feature is supported on the device
- false - The DataFormat feature is not supported on the device

Description

This function identifies whether the DataFormat feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_DataFormatSet
- PLIB_SQI_DataFormatGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDataFormat(SQI_MODULE_ID index)

PLIB_SQI_ExistsDataModeControl Function

Identifies whether the DataModeControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDataModeControl(SQI_MODULE_ID index);
```

Returns

- true - The DataModeControl feature is supported on the device
- false - The DataModeControl feature is not supported on the device

Description

This function identifies whether the DataModeControl feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DataModeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDataModeControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsDataOutputEnable Function

Identifies whether the DataOutputEnable feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDataOutputEnable(SQI_MODULE_ID index);
```

Returns

- true - The DataOutputEnable feature is supported on the device
- false - The DataOutputEnable feature is not supported on the device

Description

This function identifies whether the DataOutputEnable feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DataOutputEnableSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDataOutputEnable(SQI_MODULE_ID index)

PLIB_SQI_ExistsDataPinStatus Function

Identifies whether the DataPinStatus feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDataPinStatus(SQI_MODULE_ID index);
```

Returns

- true - The DataPinStatus feature is supported on the device
- false - The DataPinStatus feature is not supported on the device

Description

This function identifies whether the DataPinStatus feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DataLineStatus

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDataPinStatus(SQI_MODULE_ID index)

PLIB_SQI_ExistsDmaEnable Function

Identifies whether the DMAEnable feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDmaEnable(SQI_MODULE_ID index);
```

Returns

- true - The DMAEnable feature is supported on the device
- false - The DMAEnable feature is not supported on the device

Description

This function identifies whether the DMAEnable feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_DMAEnable
- PLIB_SQI_DMADisable
- PLIB_SQI_DMAIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDmaEnable(SQI_MODULE_ID index)

PLIB_SQI_ExistsDMAEngineBusy Function

Identifies whether the DMAEngineBusy feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDMAEngineBusy(SQI_MODULE_ID index);
```

Returns

- true - The DMAEngineBusy feature is supported on the device
- false - The DMAEngineBusy feature is not supported on the device

Description

This function identifies whether the DMAEngineBusy feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMABDIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDMAEngineBusy(SQI_MODULE_ID index)

PLIB_SQI_ExistsDMAProcessInProgress Function

Identifies whether the DMAProcessInProgress feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDMAProcessInProgress(SQI_MODULE_ID index);
```

Returns

- true - The DMAProcessInProgress feature is supported on the device
- false - The DMAProcessInProgress feature is not supported on the device

Description

This function identifies whether the DMAProcessInProgress feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_DMAHasStarted

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDMAProcessInProgress(SQI_MODULE_ID index)

PLIB_SQI_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsEnableControl(SQI_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_Enable
- PLIB_SQI_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsEnableControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsHoldPinControl Function

Identifies whether the HoldPinControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsHoldPinControl(SQI_MODULE_ID index);
```

Returns

- true - The HoldPinControl feature is supported on the device
- false - The HoldPinControl feature is not supported on the device

Description

This function identifies whether the HoldPinControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_HoldSet
- PLIB_SQI_HoldClear
- PLIB_SQI_HoldGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsHoldPinControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsInterruptControl Function

Identifies whether the InterruptControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsInterruptControl(SQI_MODULE_ID index);
```

Returns

- true - The InterruptControl feature is supported on the device
- false - The InterruptControl feature is not supported on the device

Description

This function identifies whether the InterruptControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_InterruptEnable
- PLIB_SQI_InterruptDisable
- PLIB_SQI_InterruptIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsInterruptControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsInterruptSignalControl Function

Identifies whether the InterruptSignalControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsInterruptSignalControl(SQI_MODULE_ID index);
```

Returns

- true - The InterruptSignalControl feature is supported on the device
- false - The InterruptSignalControl feature is not supported on the device

Description

This function identifies whether the InterruptSignalControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_InterruptSignalEnable
- PLIB_SQI_InterruptSignalDisable
- PLIB_SQI_InterruptSignalIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsInterruptSignalControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsInterruptStatus Function

Identifies whether the InterruptStatus feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsInterruptStatus(SQI_MODULE_ID index);
```

Returns

- true - The InterruptStatus feature is supported on the device
- false - The InterruptStatus feature is not supported on the device

Description

This function identifies whether the InterruptStatus feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_InterruptFlagGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsInterruptStatus(SQI_MODULE_ID index)

PLIB_SQI_ExistsLaneMode Function

Identifies whether the LaneMode feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsLaneMode(SQI_MODULE_ID index);
```

Returns

- true - The LaneMode feature is supported on the device
- false - The LaneMode feature is not supported on the device

Description

This function identifies whether the LaneMode feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_LaneModeSet

- PLIB_SQI_LaneModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsLaneMode(SQI_MODULE_ID index)

PLIB_SQI_ExistsReceiveLatch Function

Identifies whether the ReceiveLatch feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsReceiveLatch(SQI_MODULE_ID index);
```

Returns

- true - The ReceiveLatch feature is supported on the device
- false - The ReceiveLatch feature is not supported on the device

Description

This function identifies whether the ReceiveLatch feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ReceiveLatchEnable
- PLIB_SQI_ReceiveLatchDisable
- PLIB_SQI_ReceiveLatchGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsReceiveLatch(SQI_MODULE_ID index)

PLIB_SQI_ExistsRxBufferCount Function

Identifies whether the RxBufferCount feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsRxBufferCount(SQI_MODULE_ID index);
```

Returns

- true - The RxBufferCount feature is supported on the device

- false - The RxBufferCount feature is not supported on the device

Description

This function identifies whether the RxBufferCount feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_NumberOfReceiveBufferReads

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsRxBufferCount(SQI_MODULE_ID index)

PLIB_SQI_ExistsRxBufIntThreshold Function

Identifies whether the RxBufIntThreshold feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsRxBufIntThreshold(SQI_MODULE_ID index);
```

Returns

- true - The RxBufIntThreshold feature is supported on the device
- false - The RxBufIntThreshold feature is not supported on the device

Description

This function identifies whether the RxBufIntThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_RxBufferThresholdIntSet
- PLIB_SQI_RxBufferThresholdIntGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsRxBufIntThreshold(SQI_MODULE_ID index)

PLIB_SQI_ExistsRxBufThreshold Function

Identifies whether the RxBufThreshold feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsRxBufThreshold(SQI_MODULE_ID index);
```

Returns

- true - The RxBufThreshold feature is supported on the device
- false - The RxBufThreshold feature is not supported on the device

Description

This function identifies whether the RxBufThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_RxBufferThresholdSet
- PLIB_SQI_RxBufferThresholdGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsRxBufThreshold(SQI_MODULE_ID index)

PLIB_SQI_ExistsRxData Function

Identifies whether the RxData feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsRxData(SQI_MODULE_ID index);
```

Returns

- true - The RxData feature is supported on the device
- false - The RxData feature is not supported on the device

Description

This function identifies whether the RxData feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_ReceiveData

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsRxData(SQI_MODULE_ID index)

PLIB_SQI_ExistsRxUnderRun Function

Identifies whether the RxUnderRun feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsRxUnderRun(SQI_MODULE_ID index);
```

Returns

- true - The RxUnderRun feature is supported on the device
- false - The RxUnderRun feature is not supported on the device

Description

This function identifies whether the RxUnderRun feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_ReceiveBufferIsUnderrun

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsRxUnderRun(SQI_MODULE_ID index)

PLIB_SQI_ExistsSoftReset Function

Identifies whether the SoftReset feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsSoftReset(SQI_MODULE_ID index);
```

Returns

- true - The SoftReset feature is supported on the device
- false - The SoftReset feature is not supported on the device

Description

This function identifies whether the SoftReset feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_SoftReset

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsSoftReset(SQI_MODULE_ID index)

PLIB_SQI_ExistsTransferCommand Function

Identifies whether the TransferCommand feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTransferCommand(SQI_MODULE_ID index);
```

Returns

- true - The TransferCommand feature is supported on the device
- false - The TransferCommand feature is not supported on the device

Description

This function identifies whether the TransferCommand feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_TransferDirectionSet
- PLIB_SQI_TransferDirectionGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTransferCommand(SQI_MODULE_ID index)

PLIB_SQI_ExistsTransferCount Function

Identifies whether the TransferCount feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTransferCount(SQI_MODULE_ID index);
```

Returns

- true - The TransferCount feature is supported on the device
- false - The TransferCount feature is not supported on the device

Description

This function identifies whether the TransferCount feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_ByteCountSet
- PLIB_SQI_ByteCountGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTransferCount(SQI_MODULE_ID index)

PLIB_SQI_ExistsTransferModeControl Function

Identifies whether the TransferModeControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTransferModeControl(SQI_MODULE_ID index);
```

Returns

- true - The TransferModeControl feature is supported on the device
- false - The TransferModeControl feature is not supported on the device

Description

This function identifies whether the TransferModeControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_TransferModeSet
- PLIB_SQI_TransferModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTransferModeControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsTxBufferFree Function

Identifies whether the TxBufferFree feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTxBufferFree(SQI_MODULE_ID index);
```

Returns

- true - The TxBufferFree feature is supported on the device
- false - The TxBufferFree feature is not supported on the device

Description

This function identifies whether the TxBufferFree feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_TransmitBufferFreeSpaceGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTxBufferFree(SQI_MODULE_ID index)

PLIB_SQI_ExistsTxBufIntThreshold Function

Identifies whether the TxBufIntThreshold feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTxBufIntThreshold(SQI_MODULE_ID index);
```

Returns

- true - The TxBufIntThreshold feature is supported on the device
- false - The TxBufIntThreshold feature is not supported on the device

Description

This function identifies whether the TxBufIntThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_TxBufferThresholdIntSet
- PLIB_SQI_TxBufferThresholdIntGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTxBufIntThreshold(SQI_MODULE_ID index)

PLIB_SQI_ExistsTxBufThreshold Function

Identifies whether the TxBufThreshold feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTxBufThreshold(SQI_MODULE_ID index);
```

Returns

- true - The TxBufThreshold feature is supported on the device
- false - The TxBufThreshold feature is not supported on the device

Description

This function identifies whether the TxBufThreshold feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_TxBufferThresholdSet
- PLIB_SQI_TxBufferThresholdGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTxBufThreshold(SQI_MODULE_ID index)

PLIB_SQI_ExistsTxData Function

Identifies whether the TxData feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTxData(SQI_MODULE_ID index);
```

Returns

- true - The TxData feature is supported on the device
- false - The TxData feature is not supported on the device

Description

This function identifies whether the TxData feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_TransmitData

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTxData(SQI_MODULE_ID index)

PLIB_SQI_ExistsTxOverFlow Function

Identifies whether the TxOverFlow feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTxOverFlow(SQI_MODULE_ID index);
```

Returns

- true - The TxOverFlow feature is supported on the device
- false - The TxOverFlow feature is not supported on the device

Description

This function identifies whether the TxOverFlow feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_TransmitBufferHasOverflowed

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTxOverFlow(SQI_MODULE_ID index)

PLIB_SQI_ExistsWPPinControl Function

Identifies whether the WPPinControl feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsWPPinControl(SQI_MODULE_ID index);
```

Returns

- true - The WPPinControl feature is supported on the device
- false - The WPPinControl feature is not supported on the device

Description

This function identifies whether the WPPinControl feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_WriteProtectSet
- PLIB_SQI_WriteProtectClear
- PLIB_SQI_WriteProtectGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsWPPinControl(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPChipSelect Function

Identifies whether the XIPChipSelect feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPChipSelect(SQI_MODULE_ID index);
```

Returns

- true - The XIPChipSelect feature is supported on the device
- false - The XIPChipSelect feature is not supported on the device

Description

This function identifies whether the XIPChipSelect feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPChipSelectSet
- PLIB_SQI_XIPChipSelectGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPChipSelect(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPControlWord1 Function

Identifies whether the XIPControlWord1 feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPControlWord1(SQI_MODULE_ID index);
```

Returns

- true - The XIPControlWord1 feature is supported on the device
- false - The XIPControlWord1 feature is not supported on the device

Description

This function identifies whether the XIPControlWord1 feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPControlWord1Set
- PLIB_SQI_XIPControlWord1Get

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPControlWord1(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPControlWord2 Function

Identifies whether the XIPControlWord2 feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPControlWord2(SQI_MODULE_ID index);
```

Returns

- true - The XIPControlWord2 feature is supported on the device
- false - The XIPControlWord2 feature is not supported on the device

Description

This function identifies whether the XIPControlWord2 feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPControlWord2Set
- PLIB_SQI_XIPControlWord2Get

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPControlWord2(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPLaneMode Function

Identifies whether the XIPLaneMode feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPLaneMode(SQI_MODULE_ID index);
```

Returns

- true - The XIPLaneMode feature is supported on the device
- false - The XIPLaneMode feature is not supported on the device

Description

This function identifies whether the XIPLaneMode feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_XIPLaneModeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPLaneMode(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPModeBytes Function

Identifies whether the XIPModeBytes feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPModeBytes(SQI_MODULE_ID index);
```

Returns

- true - The XIPModeBytes feature is supported on the device
- false - The XIPModeBytes feature is not supported on the device

Description

This function identifies whether the XIPModeBytes feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPModeBytesSet
- PLIB_SQI_XIPModeBytesGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPModeBytes(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPModeCode Function

Identifies whether the XIPModeCode feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPModeCode(SQI_MODULE_ID index);
```

Returns

- true - The XIPModeCode feature is supported on the device
- false - The XIPModeCode feature is not supported on the device

Description

This function identifies whether the XIPModeCode feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPModeCodeSet
- PLIB_SQI_XIPModeCodeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPModeCode(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPNumberOfAddressBytes Function

Identifies whether the XIPNumberOfAddressBytes feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPNumberOfAddressBytes(SQI_MODULE_ID index);
```

Returns

- true - The XIPNumberOfAddressBytes feature is supported on the device
- false - The XIPNumberOfAddressBytes feature is not supported on the device

Description

This function identifies whether the XIPNumberOfAddressBytes feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPAddressBytesSet
- PLIB_SQI_XIPAddressBytesGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPNumberOfAddressBytes(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPNumberOfDummyBytes Function

Identifies whether the XIPNumberOfDummyBytes feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPNumberOfDummyBytes(SQI_MODULE_ID index);
```

Returns

- true - The XIPNumberOfDummyBytes feature is supported on the device
- false - The XIPNumberOfDummyBytes feature is not supported on the device

Description

This function identifies whether the XIPNumberOfDummyBytes feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPDummyBytesSet
- PLIB_SQI_XIPDummyBytesGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPNumberOfDummyBytes(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPReadOpCode Function

Identifies whether the XIPReadOpCode feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPReadOpCode(SQI_MODULE_ID index);
```

Returns

- true - The XIPReadOpCode feature is supported on the device
- false - The XIPReadOpCode feature is not supported on the device

Description

This function identifies whether the XIPReadOpCode feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPReadOpcodeSet
- PLIB_SQI_XIPReadOpcodeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPReadOpCode(SQI_MODULE_ID index)

PLIB_SQI_ExistsCommandStatus Function

Identifies whether the CommandStatus feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsCommandStatus(SQI_MODULE_ID index);
```

Returns

- true - The CommandStatus feature is supported on the device
- false - The CommandStatus feature is not supported on the device

Description

This function identifies whether the CommandStatus feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_CommandStatusGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsCommandStatus(SQI_MODULE_ID index)

PLIB_SQI_ExistsConBufAvailable Function

Identifies whether the ConBufWordsAvailable feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsConBufAvailable(SQI_MODULE_ID index);
```

Returns

- true - The ConBufWordsAvailable feature is supported on the device
- false - The ConBufWordsAvailable feature is not supported on the device

Description

This function identifies whether the ConBufWordsAvailable feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_ConBufWordsAvailable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsConBufAvailable(SQI_MODULE_ID index)

PLIB_SQI_ExistsConBufferSoftReset Function

Identifies whether the control buffer soft reset feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsConBufferSoftReset(SQI_MODULE_ID index);
```

Returns

- true - The control buffer soft reset feature is supported on the device
- false - The control buffer soft reset feature is not supported on the device

Description

This function identifies whether the control buffer soft reset feature is available on the SQI module. When this function returns true, the following function is supported on the device:

- PLIB_SQI_ConBufferSoftReset

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsConBufferSoftReset(SQI_MODULE_ID index)

PLIB_SQI_ExistsDDRMode Function

Identifies whether the DDRMode feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDDRMode(SQI_MODULE_ID index);
```

Returns

- true - The DDRModeSet and DDRModeGet feature is supported on the device
- false - The DDRModeSet and DDRModeGet feature is not supported on the device

Description

This function identifies whether the DDRModeSet feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_DDRModeSet
- PLIB_SQI_DDRModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDDRMode(SQI_MODULE_ID index)

PLIB_SQI_ExistsDMABDFetch Function

Identifies whether the DMABDFetch feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsDMABDFetch(SQI_MODULE_ID index);
```

Returns

- true - The DMABDFetch feature is supported on the device
- false - The DMABDFetch feature is not supported on the device

Description

This function identifies whether the DMABDFetch feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_DMABDFetchStart
- PLIB_SQI_DMABDFetchStop

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsDMABDFetch(SQI_MODULE_ID index)

PLIB_SQI_ExistsRxBufferSoftReset Function

Identifies whether the receive buffer soft reset feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsRxBufferSoftReset(SQI_MODULE_ID index);
```

Returns

- true - The receive buffer soft reset feature is supported on the device
- false - The receive buffer soft reset feature is not supported on the device

Description

This function identifies whether the receive buffer soft reset feature is available on the SQI module. When this function returns true, the following function is supported on the device:

- PLIB_SQI_RxBufferSoftReset

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsRxBufferSoftReset(SQI_MODULE_ID index)

PLIB_SQI_ExistsStatusCheck Function

Identifies whether the StatusCheckSet feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsStatusCheck(SQI_MODULE_ID index);
```

Returns

- true - The StatusCheckSet feature is supported on the device
- false - The StatusCheckSet feature is not supported on the device

Description

This function identifies whether the StatusCheck feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_StatusCheckSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsStatusCheck(SQI_MODULE_ID index)

PLIB_SQI_ExistsTapDelay Function

Identifies whether the TapDelay feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTapDelay(SQI_MODULE_ID index);
```

Returns

- true - The TapDelaySet feature is supported on the device
- false - The TapDelaySet feature is not supported on the device

Description

This function identifies whether the TapDelay feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_TapDelaySet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTapDelay(SQI_MODULE_ID index)

PLIB_SQI_ExistsTxBufferSoftReset Function

Identifies whether the transmit buffer soft reset feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsTxBufferSoftReset(SQI_MODULE_ID index);
```

Returns

- true - The transmit buffer soft reset feature is supported on the device
- false - The transmit buffer soft reset feature is not supported on the device

Description

This function identifies whether the transmit buffer soft reset feature is available on the SQI module. When this function returns true, the following function is supported on the device:

- PLIB_SQI_TxBufferSoftReset

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsTxBufferSoftReset(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPControlWord3 Function

Identifies whether the XIPControlWord3 feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPControlWord3(SQI_MODULE_ID index);
```

Returns

- true - The XIPControlWord3Set and XIPControlWord3Get feature is supported on the device
- false - The XIPControlWord3Set and XIPControlWord3Get feature is not supported on the device

Description

This function identifies whether the XIPControlWord3 feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPControlWord3Set

- PLIB_SQI_XIPControlWord3Get

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPControlWord3(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPControlWord4 Function

Identifies whether the XIPControlWord4 feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPControlWord4(SQI_MODULE_ID index);
```

Returns

- true - The XIPControlWord4Set and XIPControlWord4Get feature is supported on the device
- false - The XIPControlWord4Set and XIPControlWord4Get feature is not supported on the device

Description

This function identifies whether the XIPControlWord4 feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPControlWord4Set
- PLIB_SQI_XIPControlWord4Get

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPControlWord4(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPDDRMode Function

Identifies whether the XIPDDRMode feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPDDRMode(SQI_MODULE_ID index);
```

Returns

- true - The XIPDDRModeSet feature is supported on the device
- false - The XIPDDRModeSet feature is not supported on the device

Description

This function identifies whether the XIPDDRMode feature is available on the SQI module. When this function returns true, this function is supported on the device:

- PLIB_SQI_XIPDDRModeSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPDDRMode(SQI_MODULE_ID index)

PLIB_SQI_ExistsXIPSDRCommandDDRData Function

Identifies whether the XIPSDRCommandDDRData feature exists on the SQI module.

File

plib_sqi.h

C

```
bool PLIB_SQI_ExistsXIPSDRCommandDDRData(SQI_MODULE_ID index);
```

Returns

- true - The XIPSDRCommandDDRDataSet and XIPSDRCommandDDRDataSetfeatureGet is supported on the device
- false - The XIPSDRCommandDDRDataSet and XIPSDRCommandDDRDataGet not supported on the device

Description

This function identifies whether the XIPSDRCommandDDRData feature is available on the SQI module. When this function returns true, these functions are supported on the device:

- PLIB_SQI_XIPSDRCommandDDRDataSet
- PLIB_SQI_XIPSDRCommandDDRDataGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_SQI_ExistsXIPSDRCommandDDRData(SQI_MODULE_ID index)

k) Data Types and Constants

SQI_ADDR_BYTES Enumeration

Defines the list of SQI address bytes.

File

help_plib_sqi.h

C

```
typedef enum {
  ADDR_BYTES_4,
  ADDR_BYTES_3,
  ADDR_BYTES_2,
  ADDR_BYTES_1,
  ADDR_BYTES_0
} SQI_ADDR_BYTES;
```

Members

Members	Description
ADDR_BYTES_4	SQI is set to Transmit 4 Address Bytes
ADDR_BYTES_3	SQI is set to Transmit 3 Address Bytes
ADDR_BYTES_2	SQI is set to Transmit 2 Address Bytes
ADDR_BYTES_1	SQI is set to Transmit 1 Address Bytes
ADDR_BYTES_0	SQI is set to Transmit No Address Bytes

Description

SQI Number of Address Bytes

This macro defines the list of SQI address bytes.

SQI_BD_CTRL_WORD Enumeration

Defines the list of SQI Buffer Descriptor control word.

File

help_plib_sqi.h

C

```
typedef enum {
  BD_ENABLE,
  BD_DISABLE,
  CS_1_ACTIVE,
  CS_0_ACTIVE,
  FLASH_SC_ENABLE,
  FLASH_SC_DISABLE,
  SET_LSBF,
  SET_MSBF,
  DMA_LANE_QUAD,
  DMA_LANE_DUAL,
  DMA_LANE_SINGLE,
  DMA_IN_TRANSMIT,
  DMA_IN_RECEIVE,
  LAST_BD,
  NOT_LAST_BD,
  LAST_PKT,
  NOT_LAST_PKT,
  PKT_INT_ENABLE,
  PKT_INT_DISABLE,
  BD_DONE_INT_ENABLE,
  BD_DONE_INT_DISABLE,
  BD_BUF_LENGTH
} SQI_BD_CTRL_WORD;
```

Members

Members	Description
BD_ENABLE	Current Buffer Descriptor is Enabled
BD_DISABLE	Current Buffer Descriptor is Disabled
CS_1_ACTIVE	Chip Select 1 is Active
CS_0_ACTIVE	Chip Select 0 is Active
FLASH_SC_ENABLE	Automatic Status Check Enabled
FLASH_SC_DISABLE	Automatic Status Check Disabled
SET_LSBF	Least Significant Bit First
SET_MSBF	Most Significant Bit First

DMA_LANE_QUAD	DMA writes/reads in Quad Lane Mode
DMA_LANE_DUAL	DMA writes/reads in Dual Lane Mode
DMA_LANE_SINGLE	DMA writes/reads in Single Lane Mode
DMA_IN_TRANSMIT	DMA is Transmitting
DMA_IN_RECEIVE	DMA is Receiving
LAST_BD	Indicates Last Buffer Descriptor in the List
NOT_LAST_BD	Indicates Non Last Buffer Descriptor in the List
LAST_PKT	Indicates Last Packet of the Data Chunk
NOT_LAST_PKT	Indicates Non Last Packet of the Data Chunk
PKT_INT_ENABLE	Indicates Packet Interrupt is Enabled
PKT_INT_DISABLE	Indicates Packet Interrupt is Disabled
BD_DONE_INT_ENABLE	Indicates Buffer Descriptor Done Interrupt is Enabled
BD_DONE_INT_DISABLE	Indicates Buffer Descriptor Done Interrupt is Disabled
BD_BUF_LENGTH	Indicates Buffer Length

Description

SQI Buffer Descriptor Control Words

This macro defines the list of SQI Buffer Descriptor control word.

SQI_BD_STATE Enumeration

Defines the list of SQI Buffer Descriptor states.

File

help_plib_sqi.h

C

```
typedef enum {
  BD_STATE_DISABLED,
  BD_STATE_DONE,
  BD_STATE_PROCESSING_DATA,
  BD_STATE_BEING_FETCHED,
  BD_STATE_FETCH_REQ_PENDING,
  BD_STATE_IDLE
} SQI_BD_STATE;
```

Members

Members	Description
BD_STATE_DISABLED	Fetched Buffer Descriptor is Disabled
BD_STATE_DONE	Fetched Buffer Descriptor Processed
BD_STATE_PROCESSING_DATA	Fetched Buffer Descriptor is in Data transfer phase
BD_STATE_BEING_FETCHED	In the process of Fetching the Buffer Descriptor
BD_STATE_FETCH_REQ_PENDING	Buffer Descriptor Fetch Request Pending
BD_STATE_IDLE	Buffer Descriptor process is idle

Description

SQI Buffer Descriptor (BD) State

This macro defines the list of SQI Buffer Descriptor states.

SQI_CLK_DIV Enumeration

Defines the list of SQI Clock Divider values.

File

help_plib_sqi.h

C

```
typedef enum {
  CLK_DIV_256,
  CLK_DIV_128,
```

```
  CLK_DIV_64,
  CLK_DIV_32,
  CLK_DIV_16,
  CLK_DIV_8,
  CLK_DIV_4,
  CLK_DIV_2,
  CLK_DIV_1
} SQI_CLK_DIV;
```

Members

Members	Description
CLK_DIV_256	SQI Clock is divided by 256
CLK_DIV_128	SQI Clock is divided by 128
CLK_DIV_64	SQI Clock is divided by 64
CLK_DIV_32	SQI Clock is divided by 32
CLK_DIV_16	SQI Clock is divided by 16
CLK_DIV_8	SQI Clock is divided by 8
CLK_DIV_4	SQI Clock is divided by 4
CLK_DIV_2	SQI Clock is divided by 2
CLK_DIV_1	SQI Clock is not divided

Description

SQI Clock Divider

This macro defines the list of SQI Clock Divider values.

SQI_CS_NUM Enumeration

Defines the list of SQI Chip Selects.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_CS_1,
  SQI_CS_0
} SQI_CS_NUM;
```

Members

Members	Description
SQI_CS_1	SQI Chip Select 1
SQI_CS_0	SQI Chip Select 0

Description

SQI Device Select or Chip Select

This macro defines the list of SQI Chip Selects.

SQI_CS_OEN Enumeration

Defines the list of SQI Chip Selects on which output is enable.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_CS_OEN_BOTH,
  SQI_CS_OEN_1,
  SQI_CS_OEN_0,
  SQI_CS_OEN_NONE
} SQI_CS_OEN;
```

Members

Members	Description
SQI_CS_OEN_BOTH	SQI chip select 0 and 1 are enabled
SQI_CS_OEN_1	SQI chip select 1 is enabled but chip select 0 is disabled
SQI_CS_OEN_0	SQI chip select 0 is enabled but chip select 1 is disabled
SQI_CS_OEN_NONE	SQI chip select 0 and 1 are disabled

Description

SQI Chip Select Output Enable

This macro defines the list of SQI Chip Selects on which output is enabled.

SQI_DATA_FORMAT Enumeration

Defines the Data Format Options available (LSBF/MSBF).

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_DATA_FORMAT_LSBF,
  SQI_DATA_FORMAT_MSBF
} SQI_DATA_FORMAT;
```

Members

Members	Description
SQI_DATA_FORMAT_LSBF	SQI Data is Least Significant Bit First Formatted
SQI_DATA_FORMAT_MSBF	SQI Data is Most Significant Bit First Formatted

Description

SQI Data Mode

This macro defines the Data Formats (LSBF/MSBF).

SQI_DATA_MODE Enumeration

Defines the list of SQI Data modes.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_DATA_MODE_3,
  SQI_DATA_MODE_0
} SQI_DATA_MODE;
```

Members

Members	Description
SQI_DATA_MODE_3	SQI is in SPI mode 3
SQI_DATA_MODE_0	SQI is in SPI mode 0

Description

SQI Data Mode

This macro defines the list of SQI Data modes.

SQI_DATA_OEN Enumeration

Defines the list of SQI Data pins on which output is enabled.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_DATA_OEN_QUAD,
  SQI_DATA_OEN_DUAL,
  SQI_DATA_OEN_SINGLE
} SQI_DATA_OEN;
```

Members

Members	Description
SQI_DATA_OEN_QUAD	SQI data outputs 3 to 0 are enabled (quad lane mode)
SQI_DATA_OEN_DUAL	SQI data outputs 1 and 0 are enabled (dual lane mode)
SQI_DATA_OEN_SINGLE	SQI data output 0 is enabled (single lane mode)

Description

SQI Data Output Enable

This macro defines the list of SQI Data Output pins on which output is enabled.

SQI_DATA_TYPE Type

Data type defining the SQI Data size.

File

help_plib_sqi.h

C

```
typedef uint32_t SQI_DATA_TYPE;
```

Description

SQI Data Type definition

This data type defines the SQI Data size

SQI_DUMMY_BYTES Enumeration

Defines the list of SQI dummy bytes.

File

help_plib_sqi.h

C

```
typedef enum {
  DUMMY_BYTES_7,
  DUMMY_BYTES_6,
  DUMMY_BYTES_5,
  DUMMY_BYTES_4,
  DUMMY_BYTES_3,
  DUMMY_BYTES_2,
  DUMMY_BYTE_1,
  DUMMY_BYTES_0
} SQI_DUMMY_BYTES;
```

Members

Members	Description
DUMMY_BYTES_7	SQI is set to Transmit 7 Dummy Bytes
DUMMY_BYTES_6	SQI is set to Transmit 6 Dummy Bytes
DUMMY_BYTES_5	SQI is set to Transmit 5 Dummy Bytes
DUMMY_BYTES_4	SQI is set to Transmit 4 Dummy Bytes
DUMMY_BYTES_3	SQI is set to Transmit 3 Dummy Bytes
DUMMY_BYTES_2	SQI is set to Transmit 2 Dummy Bytes

DUMMY_BYTE_1	SQI is set to Transmit 1 Dummy Bytes
DUMMY_BYTES_0	SQI is set to Transmit No Dummy Bytes

Description

SQI Number of Dummy Bytes

This macro defines the list of SQI dummy bytes after address bytes.

SQI_INTERRUPTS Enumeration

Defines the list of SQI interrupts.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_PKTCOMP,
  SQI_BDDONE,
  SQI_CONTHR,
  SQI_CONEMPTY,
  SQI_CONFULL,
  SQI_RXTHR,
  SQI_RXFULL,
  SQI_RXEMPTY,
  SQI_TXTHR,
  SQI_TXFULL,
  SQI_TXEMPTY
} SQI_INTERRUPTS;
```

Members

Members	Description
SQI_PKTCOMP	SQI Packet Complete Interrupt (used in DMA mode)
SQI_BDDONE	SQI Buffer Descriptor Done Interrupt (used in DMA mode)
SQI_CONTHR	SQI Control Buffer Threshold Interrupt
SQI_CONEMPTY	SQI Control Buffer Empty Interrupt
SQI_CONFULL	SQI Control Buffer Full Interrupt
SQI_RXTHR	SQI Receive Buffer Threshold Interrupt
SQI_RXFULL	SQI Receive Buffer Empty Interrupt
SQI_RXEMPTY	SQI Receive Buffer Full Interrupt
SQI_TXTHR	SQI Transmit Buffer Threshold Interrupt
SQI_TXFULL	SQI Transmit Buffer Empty Interrupt
SQI_TXEMPTY	SQI Transmit Buffer Full Interrupt

Description

SQI Interrupt List

This macro defines the list of SQI interrupts.

SQI_LANE_MODE Enumeration

Defines the list of SQI Lane modes (single/dual/quad).

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_LANE_QUAD,
  SQI_LANE_DUAL,
  SQI_LANE_SINGLE
} SQI_LANE_MODE;
```

Members

Members	Description
SQI_LANE_QUAD	SQI is set into Quad Lane Mode
SQI_LANE_DUAL	SQI is set into Dual Lane Mode
SQI_LANE_SINGLE	SQI is set into Single Lane Mode

Description

SQI Lane Mode

This macro defines the list of SQI Lane modes.

SQI_MODE_BYTES Enumeration

Defines the list of SQI mode bytes.

File

help_plib_sqi.h

C

```
typedef enum {
  MODE_BYTES_3,
  MODE_BYTES_2,
  MODE_BYTES_1,
  MODE_BYTES_0
} SQI_MODE_BYTES;
```

Members

Members	Description
MODE_BYTES_3	SQI is set to Transmit 3 Mode Bytes
MODE_BYTES_2	SQI is set to Transmit 2 Mode Bytes
MODE_BYTES_1	SQI is set to Transmit 1 Mode Bytes
MODE_BYTES_0	SQI is set to Transmit No Mode Bytes

Description

SQI Number of Mode Bytes

This macro defines the list of SQI mode bytes allocated for mode code.

SQI_MODULE_ID Enumeration

Identifies the supported SQI modules.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_ID_1,
  SQI_NUMBER_OF_MODULES
} SQI_MODULE_ID;
```

Members

Members	Description
SQI_ID_1	SQI Module 1 ID
SQI_NUMBER_OF_MODULES	Number of available SQI modules

Description

SQI Module ID

This enumeration identifies the SQI modules that are available on the microcontroller. This is the super set of all the possible instances that might be available on the Microchip microcontrollers.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

SQI_XFER_CMD Enumeration

Defines the list of SQI transfer commands.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_CMD_RECEIVE,
  SQI_CMD_TRANSMIT,
  SQI_CMD_IDLE
} SQI_XFER_CMD;
```

Members

Members	Description
SQI_CMD_RECEIVE	SQI issues a Receive Command
SQI_CMD_TRANSMIT	SQI issues a Transmit Command
SQI_CMD_IDLE	SQI is Idle

Description

SQI Transfer Command

This macro defines the list of SQI transfer commands.

SQI_XFER_MODE Enumeration

Defines the list of SQI Transfer modes.

File

help_plib_sqi.h

C

```
typedef enum {
  SQI_XFER_MODE_XIP,
  SQI_XFER_MODE_DMA,
  SQI_XFER_MODE_PIO
} SQI_XFER_MODE;
```

Members

Members	Description
SQI_XFER_MODE_XIP	SQI is in XIP mode
SQI_XFER_MODE_DMA	SQI is in DMA mode
SQI_XFER_MODE_PIO	SQI is in PIO mode

Description

SQI Transfer Mode

This macro defines the list of SQI Transfer modes.

Files

Files

Name	Description
plib_sqi.h	SQI Peripheral Library Interface Header for common definitions.
help_plib_sqi.h	Identifies the various enumerations in SQI modules supported.

Description

This section lists the source and header files used by the library.

plib_sqi.h

SQI Peripheral Library Interface Header for common definitions.

Functions

	Name	Description
	PLIB_SQI_BurstEnable	Sets the burst enable (BURSTEN) function for higher throughput. This function is an artifact of the System Bus architecture.
	PLIB_SQI_ByteCountGet	Returns the current transmit/receive count.
	PLIB_SQI_ByteCountSet	Sets the transmit/receive count.
	PLIB_SQI_ChipSelectDeassertDisable	Clears the Chip Select deassert.
	PLIB_SQI_ChipSelectDeassertEnable	Sets the Chip Select deassert.
	PLIB_SQI_ChipSelectGet	Returns the Chip Select that is currently active.
	PLIB_SQI_ChipSelectSet	Activates the Chip Select.
	PLIB_SQI_ClockDisable	Disables the SQI transfer clock.
	PLIB_SQI_ClockDividerSet	Sets the SQI clock (that drives the SQI protocol) divider value. Divides the base clock to generate the SQI clock.
	PLIB_SQI_ClockEnable	Enables the SQI transfer clock.
	PLIB_SQI_ClockIsStable	Returns SQI transfer clock state.
	PLIB_SQI_CommandStatusGet	Returns the SQI command (transmit/receive/idle).
	PLIB_SQI_ConBufferSoftReset	Issues a soft reset to the SQI control buffer.
	PLIB_SQI_ConBufWordsAvailable	Returns the number of control buffer words available.
	PLIB_SQI_ConfigWordGet	Gets the SQI Configuration Word.
	PLIB_SQI_ConfigWordSet	Sets SQI Configuration Word.
	PLIB_SQI_ControlBufferThresholdGet	Returns the transmit buffer space in bytes.
	PLIB_SQI_ControlBufferThresholdSet	Sets the control buffer threshold value.
	PLIB_SQI_ControlWordGet	Get the SQI Control Word.
	PLIB_SQI_ControlWordSet	Sets the SQI Control Word.
	PLIB_SQI_CSOutputEnableSelect	Selects the output enables on SQI Chip Select pins.
	PLIB_SQI_DataFormatGet	Returns the data format.
	PLIB_SQI_DataFormatSet	Sets the data format to Least Significant Bit First (LSBF)..
	PLIB_SQI_DataLineStatus	Returns the logical status of the SQI data lines.
	PLIB_SQI_DataModeSet	Sets the SQI data mode of operation.
	PLIB_SQI_DataOutputEnableSelect	Selects the output enables on SQI data outputs.
	PLIB_SQI_DDRModeGet	Return the SQI data rate mode (single/double) value.
	PLIB_SQI_DDRModeSet	Sets SQI communication to DDR mode.
	PLIB_SQI_Disable	Disables the SQI module.
	PLIB_SQI_DMABDBaseAddressGet	Returns the address of the base buffer descriptor.
	PLIB_SQI_DMABDBaseAddressSet	Sets the address of the base buffer descriptor.
	PLIB_SQI_DMABDControlWordGet	Returns Current Buffer Descriptor Control Word Information.
	PLIB_SQI_DMABDCurrentAddressGet	Returns the address of the current buffer descriptor in process.
	PLIB_SQI_DMABDFetchStart	Starts the DMA buffer descriptor fetch process.
	PLIB_SQI_DMABDFetchStop	Stops the DMA buffer descriptor fetch process.
	PLIB_SQI_DMABDIsBusy	Returns whether or not the DMA Buffer Descriptor is busy.
	PLIB_SQI_DMABDPollCounterSet	Sets the poll counter value.
	PLIB_SQI_DMABDPollDisable	Disables the buffer descriptor polling.
	PLIB_SQI_DMABDPollEnable	Enables the buffer descriptor polling.
	PLIB_SQI_DMABDPollIsEnabled	Returns whether or not the DMA buffer descriptor poll is enabled.
	PLIB_SQI_DMABDReceiveBufferCountGet	Returns the receive buffer space in bytes.
	PLIB_SQI_DMABDReceiveBufferLengthGet	Returns the receive length in bytes.
	PLIB_SQI_DMABDReceiveStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMABDStateGet	Returns the current state of the buffer descriptor in progress.

	PLIB_SQI_DMABDTransmitBufferCountGet	Returns the transmit buffer space in bytes.
	PLIB_SQI_DMABDTransmitBufferLengthGet	Returns the transmit length in bytes.
	PLIB_SQI_DMABDTransmitStateGet	Returns the current state of the buffer descriptor in progress.
	PLIB_SQI_DMADisable	Disables the built-in DMA logic.
	PLIB_SQI_DMAEnable	Enables the built-in DMA logic.
	PLIB_SQI_DMAHasStarted	Returns whether or no the DMA process has started.
	PLIB_SQI_DMAIsEnabled	Returns whether or not DMA is enabled.
	PLIB_SQI_Enable	Enables the SQI module.
	PLIB_SQI_ExistsBDBaseAddress	Identifies whether the BDBaseAddress feature exists on the SQI module.
	PLIB_SQI_ExistsBDControlWord	Identifies whether the BDControlWord feature exists on the SQI module.
	PLIB_SQI_ExistsBDCurrentAddress	Identifies whether the BDCurrentAddress feature exists on the SQI module.
	PLIB_SQI_ExistsBDPollCount	Identifies whether the BDPollCount feature exists on the SQI module.
	PLIB_SQI_ExistsBDPollingEnable	Identifies whether the BDPollingEnable feature exists on the SQI module.
	PLIB_SQI_ExistsBDProcessState	Identifies whether the BDProcessState feature exists on the SQI module.
	PLIB_SQI_ExistsBDRxBufCount	Identifies whether the BDRxBufCount feature exists on the SQI module.
	PLIB_SQI_ExistsBDRxLength	Identifies whether the BDRxLength feature exists on the SQI module.
	PLIB_SQI_ExistsBDRxState	Identifies whether the BDRxState feature exists on the SQI module.
	PLIB_SQI_ExistsBDTxBufCount	Identifies whether the BDTxBufCount feature exists on the SQI module.
	PLIB_SQI_ExistsBDTxLength	Identifies whether the BDTxLength feature exists on the SQI module.
	PLIB_SQI_ExistsBDTxState	Identifies whether the BDTxState feature exists on the SQI module.
	PLIB_SQI_ExistsBurstControl	Identifies whether the BurstControl feature exists on the SQI module.
	PLIB_SQI_ExistsChipSelect	Identifies whether the ChipSelect feature exists on the SQI module.
	PLIB_SQI_ExistsClockControl	Identifies whether the ClockControl feature exists on the SQI module.
	PLIB_SQI_ExistsClockDivider	Identifies whether the ClockDivider feature exists on the SQI module.
	PLIB_SQI_ExistsClockReady	Identifies whether the ClockReady feature exists on the SQI module.
	PLIB_SQI_ExistsCommandStatus	Identifies whether the CommandStatus feature exists on the SQI module.
	PLIB_SQI_ExistsConBufAvailable	Identifies whether the ConBufWordsAvailable feature exists on the SQI module.
	PLIB_SQI_ExistsConBufferSoftReset	Identifies whether the control buffer soft reset feature exists on the SQI module.
	PLIB_SQI_ExistsConBufThreshold	Identifies whether the ConBufThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsConfigWord	Identifies whether the ConfigWord feature exists on the SQI module.
	PLIB_SQI_ExistsControlWord	Identifies whether the ControlWord feature exists on the SQI module.
	PLIB_SQI_ExistsCSDeassert	Identifies whether the CSDeassert feature exists on the SQI module.
	PLIB_SQI_ExistsCSOutputEnable	Identifies whether the CSOutputEnable feature exists on the SQI module.
	PLIB_SQI_ExistsDataFormat	Identifies whether the DataFormat feature exists on the SQI module.
	PLIB_SQI_ExistsDataModeControl	Identifies whether the DataModeControl feature exists on the SQI module.
	PLIB_SQI_ExistsDataOutputEnable	Identifies whether the DataOutputEnable feature exists on the SQI module.
	PLIB_SQI_ExistsDataPinStatus	Identifies whether the DataPinStatus feature exists on the SQI module.
	PLIB_SQI_ExistsDDRMode	Identifies whether the DDRMode feature exists on the SQI module.
	PLIB_SQI_ExistsDMABDFetch	Identifies whether the DMABDFetch feature exists on the SQI module.
	PLIB_SQI_ExistsDmaEnable	Identifies whether the DMAEnable feature exists on the SQI module.
	PLIB_SQI_ExistsDMAEngineBusy	Identifies whether the DMAEngineBusy feature exists on the SQI module.
	PLIB_SQI_ExistsDMAProcessInProgress	Identifies whether the DMAProcessInProgress feature exists on the SQI module.
	PLIB_SQI_ExistsEnableControl	Identifies whether the EnableControl feature exists on the SQI module.
	PLIB_SQI_ExistsHoldPinControl	Identifies whether the HoldPinControl feature exists on the SQI module.
	PLIB_SQI_ExistsInterruptControl	Identifies whether the InterruptControl feature exists on the SQI module.
	PLIB_SQI_ExistsInterruptSignalControl	Identifies whether the InterruptSignalControl feature exists on the SQI module.
	PLIB_SQI_ExistsInterruptStatus	Identifies whether the InterruptStatus feature exists on the SQI module.
	PLIB_SQI_ExistsLaneMode	Identifies whether the LaneMode feature exists on the SQI module.
	PLIB_SQI_ExistsReceiveLatch	Identifies whether the ReceiveLatch feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufferCount	Identifies whether the RxBufferCount feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufferSoftReset	Identifies whether the receive buffer soft reset feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufIntThreshold	Identifies whether the RxBufIntThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsRxBufThreshold	Identifies whether the RxBufThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsRxData	Identifies whether the RxData feature exists on the SQI module.
	PLIB_SQI_ExistsRxUnderRun	Identifies whether the RxUnderRun feature exists on the SQI module.

	PLIB_SQI_ExistsSoftReset	Identifies whether the SoftReset feature exists on the SQI module.
	PLIB_SQI_ExistsStatusCheck	Identifies whether the StatusCheckSet feature exists on the SQI module.
	PLIB_SQI_ExistsTapDelay	Identifies whether the TapDelay feature exists on the SQI module.
	PLIB_SQI_ExistsTransferCommand	Identifies whether the TransferCommand feature exists on the SQI module.
	PLIB_SQI_ExistsTransferCount	Identifies whether the TransferCount feature exists on the SQI module.
	PLIB_SQI_ExistsTransferModeControl	Identifies whether the TransferModeControl feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufferFree	Identifies whether the TxBufferFree feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufferSoftReset	Identifies whether the transmit buffer soft reset feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufIntThreshold	Identifies whether the TxBufIntThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsTxBufThreshold	Identifies whether the TxBufThreshold feature exists on the SQI module.
	PLIB_SQI_ExistsTxData	Identifies whether the TxData feature exists on the SQI module.
	PLIB_SQI_ExistsTxOverFlow	Identifies whether the TxOverFlow feature exists on the SQI module.
	PLIB_SQI_ExistsWPPinControl	Identifies whether the WPPinControl feature exists on the SQI module.
	PLIB_SQI_ExistsXIPChipSelect	Identifies whether the XIPChipSelect feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord1	Identifies whether the XIPControlWord1 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord2	Identifies whether the XIPControlWord2 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord3	Identifies whether the XIPControlWord3 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPControlWord4	Identifies whether the XIPControlWord4 feature exists on the SQI module.
	PLIB_SQI_ExistsXIPDDRMode	Identifies whether the XIPDDRMode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPLaneMode	Identifies whether the XIPLaneMode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPModeBytes	Identifies whether the XIPModeBytes feature exists on the SQI module.
	PLIB_SQI_ExistsXIPModeCode	Identifies whether the XIPModeCode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPNumberOfAddressBytes	Identifies whether the XIPNumberOfAddressBytes feature exists on the SQI module.
	PLIB_SQI_ExistsXIPNumberOfDummyBytes	Identifies whether the XIPNumberOfDummyBytes feature exists on the SQI module.
	PLIB_SQI_ExistsXIPReadOpCode	Identifies whether the XIPReadOpCode feature exists on the SQI module.
	PLIB_SQI_ExistsXIPSDRCommandDDRData	Identifies whether the XIPSDRCommandDDRData feature exists on the SQI module.
	PLIB_SQI_HoldClear	Clears the hold function to be disabled on SQID3 in single and dual lane modes.
	PLIB_SQI_HoldGet	Gets the status of HOLD function on SQID3 pin.
	PLIB_SQI_HoldSet	Sets the hold function to be enabled on SQID3 in single or dual lane modes.
	PLIB_SQI_InterruptDisable	Disables the interrupt source.
	PLIB_SQI_InterruptEnable	Enables the passed interrupt source.
	PLIB_SQI_InterruptFlagGet	Return SQI Interrupt flag status.
	PLIB_SQI_InterruptIsEnabled	Returns the interrupt state.
	PLIB_SQI_InterruptSignalDisable	Disables the interrupt signal source.
	PLIB_SQI_InterruptSignalEnable	Enables the passed interrupt signal source.
	PLIB_SQI_InterruptSignalIsEnabled	Returns the interrupt signal state.
	PLIB_SQI_LaneModeGet	Returns the lane mode (single/dual/quad).
	PLIB_SQI_LaneModeSet	Sets the data lane mode (single/dual/quad).
	PLIB_SQI_NumberOfReceiveBufferReads	Returns the number of receive buffer reads.
	PLIB_SQI_ReceiveBufferIsUnderrun	Returns the status of receive buffer.
	PLIB_SQI_ReceiveData	Reads the data from the receive buffer.
	PLIB_SQI_ReceiveLatchDisable	Disables the receive latch so receive data is discarded when in transmit mode.
	PLIB_SQI_ReceiveLatchEnable	Enables the receive latch so receive data is latched during transmit mode.
	PLIB_SQI_ReceiveLatchGet	Returns the receive latch status in transmit mode.
	PLIB_SQI_RxBufferSoftReset	Issues a soft reset to the SQI receive buffer.
	PLIB_SQI_RxBufferThresholdGet	Returns the receive command threshold.
	PLIB_SQI_RxBufferThresholdIntGet	Sets the receive buffer threshold interrupt.
	PLIB_SQI_RxBufferThresholdIntSet	Sets the receive buffer threshold for interrupt.
	PLIB_SQI_RxBufferThresholdSet	Sets the receive command threshold.
	PLIB_SQI_SoftReset	Issues a soft reset to the SQI module.
	PLIB_SQI_StatusCheckSet	Sets the Flash status check related control bits.
	PLIB_SQI_TapDelaySet	Sets the tap delays.
	PLIB_SQI_TransferDirectionGet	Returns the transfer command.
	PLIB_SQI_TransferDirectionSet	Sets the transfer command.

	PLIB_SQI_TransferModeGet	Returns the SQI transfer mode of operation.
	PLIB_SQI_TransferModeSet	Sets the SQI transfer mode of operation.
	PLIB_SQI_TransmitBufferFreeSpaceGet	Returns the number of transmit buffer words available.
	PLIB_SQI_TransmitBufferHasOverflowed	Returns the current status of the transmit buffer.
	PLIB_SQI_TransmitData	Writes data into the SQI transmit buffer.
	PLIB_SQI_TxBufferSoftReset	Issues a soft reset to the SQI transmit buffer.
	PLIB_SQI_TxBufferThresholdGet	Returns the transmit command threshold value.
	PLIB_SQI_TxBufferThresholdIntGet	Returns the value to trigger the transmit buffer threshold interrupt.
	PLIB_SQI_TxBufferThresholdIntSet	Sets the value to trigger the transmit buffer threshold interrupt.
	PLIB_SQI_TxBufferThresholdSet	Sets the transmit command threshold.
	PLIB_SQI_WriteProtectClear	Clears the Write-Protect function to be disabled on SQID2 in single or dual lane modes.
	PLIB_SQI_WriteProtectGet	Gets the state of the write-protect feature on SQID2.
	PLIB_SQI_WriteProtectSet	Sets the write-protect feature to be enabled on SQID2 in single or dual lane modes only.
	PLIB_SQI_XIPAddressBytesGet	Returns the number of address bytes.
	PLIB_SQI_XIPAddressBytesSet	Sets the number of address bytes.
	PLIB_SQI_XIPChipSelectGet	Returns the current active Chip Select.
	PLIB_SQI_XIPChipSelectSet	Activates the Chip Select in XIP mode.
	PLIB_SQI_XIPControlWord1Get	Get the XIP mode Control Word 1.
	PLIB_SQI_XIPControlWord1Set	Sets the XIP mode Control Word 1.
	PLIB_SQI_XIPControlWord2Get	Gets the XIP mode Control Word 2.
	PLIB_SQI_XIPControlWord2Set	Sets the XIP mode Control Word 2.
	PLIB_SQI_XIPControlWord3Get	Gets the XIP mode Control Word 3.
	PLIB_SQI_XIPControlWord3Set	Sets the XIP mode Control Word 3.
	PLIB_SQI_XIPControlWord4Get	Gets the XIP mode Control Word 4.
	PLIB_SQI_XIPControlWord4Set	Sets the XIP mode Control Word 4.
	PLIB_SQI_XIPDDRModeSet	Selects the rate mode (SDR/DDR) for different transactions in XIP mode.
	PLIB_SQI_XIPDummyBytesGet	Sets the number of dummy bytes.
	PLIB_SQI_XIPDummyBytesSet	Sets the number of dummy bytes.
	PLIB_SQI_XIPLaneModeSet	Selects the lane (Single/Dual/Quad) mode for different transaction in XIP mode.
	PLIB_SQI_XIPModeBytesGet	Returns the number of bytes used for mode code command.
	PLIB_SQI_XIPModeBytesSet	Allocates the bytes for mode code command.
	PLIB_SQI_XIPModeCodeGet	Returns the mode code op-code.
	PLIB_SQI_XIPModeCodeSet	Sets the mode code command.
	PLIB_SQI_XIPReadOpcodeGet	Returns the read op code in XIP mode.
	PLIB_SQI_XIPReadOpcodeSet	Sets the read op code in XIP mode.
	PLIB_SQI_XIPSDRCommandDDRDataGet	Returns the SQI command in SDR mode and Data in DDR mode bit value.
	PLIB_SQI_XIPSDRCommandDDRDataSet	Sets the SQI command in SDR mode.

Description

Serial Quad Interface (SQI) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the SQI Peripheral Library.

File Name

plib_sqi.h

Company

Microchip Technology Inc.

help_plib_sqi.h

Identifies the various enumerations in SQI modules supported.

Enumerations

	Name	Description
	SQI_ADDR_BYTES	Defines the list of SQI address bytes.
	SQI_BD_CTRL_WORD	Defines the list of SQI Buffer Descriptor control word.
	SQI_BD_STATE	Defines the list of SQI Buffer Descriptor states.
	SQI_CLK_DIV	Defines the list of SQI Clock Divider values.
	SQI_CS_NUM	Defines the list of SQI Chip Selects.
	SQI_CS_OEN	Defines the list of SQI Chip Selects on which output is enable.
	SQI_DATA_FORMAT	Defines the Data Format Options available (LSBF/MSBF).
	SQI_DATA_MODE	Defines the list of SQI Data modes.
	SQI_DATA_OEN	Defines the list of SQI Data pins on which output is enabled.
	SQI_DUMMY_BYTES	Defines the list of SQI dummy bytes.
	SQI_INTERRUPTS	Defines the list of SQI interrupts.
	SQI_LANE_MODE	Defines the list of SQI Lane modes (single/dual/quad).
	SQI_MODE_BYTES	Defines the list of SQI mode bytes.
	SQI_MODULE_ID	Identifies the supported SQI modules.
	SQI_XFER_CMD	Defines the list of SQI transfer commands.
	SQI_XFER_MODE	Defines the list of SQI Transfer modes.

Types

	Name	Description
	SQI_DATA_TYPE	Data type defining the SQI Data size.

Description

This enumeration identifies the SQI modules which are available on the microcontroller. This is the super set of all the possible instances that might be available on the Microchip microcontrollers.

File Name

help_plib_sqi.h

Company

Microchip Technology Inc.

Timer Peripheral Library

This section describes the Timer Peripheral Library.

Introduction

This library provides a low-level abstraction of the Timer module on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

The timer function is one of the basic features of a microcontroller. Timers are useful for generating accurate time-based periodic interrupts for software application or real-time operating systems. Other uses include counting external pulses or accurate timing measurement of external events using the timer's gate functions or producing precise time delays. Some timers can also be used to control timing of other peripherals.

Using the Library

This topic describes the basic architecture of the Timer Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_tmr.h`

The interface to the Timer Peripheral Library is defined in the `plib_tmr.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the Timer Peripheral Library must include `peripheral.h`.

Library File:

The Timer Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive

are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Hardware Abstraction Model

This library provides the low-level abstraction of the Timer module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all the functionality of the available timers on the device. Refer to the **"Timer"** chapters in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each Timer module on your device.

Description

Microchip timers can be classified as:

- Overflow-based
- Period match-based

These timers are essentially counters of a specific size (8-, 16-, or 32-bits wide) that increment based on the clock cycle and the timer prescaler. An application can monitor these counters to determine how much time has elapsed.

The prescaler determines the timer granularity (resolution). It is a mechanism for generating the clock for the timer by the CPU clock. Every CPU has a clock source and the frequency of this source decides the rate at which instructions are executed by the processor. Most of the timers have the option of providing input clock prescalers, while some have the optional postscalers. The prescaler is used to divide the clock frequency of the Timer module and produce a clock for the timer.

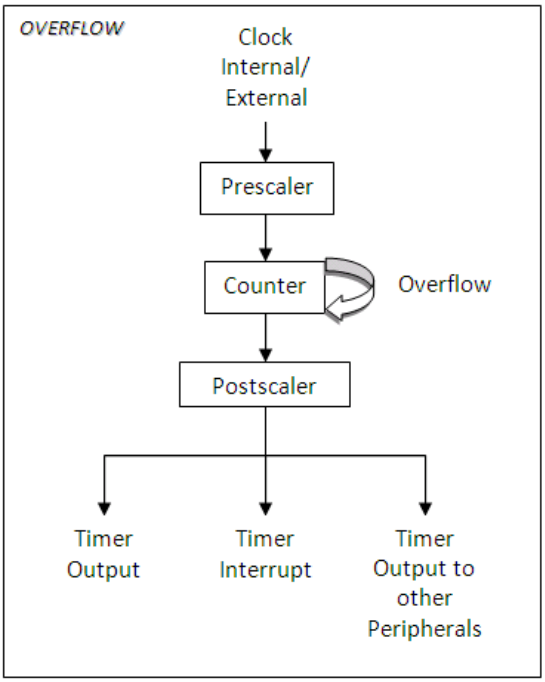
Each timer can be configured with a different source (internal or external) meaning, the source of the timer's input clock is selectable. Both internal and external sources are available, depending on the part in use, with some timers providing a clock synchronization mechanisms with reference to external source.

On some timers the clock edge, either the rising or the falling edge, on which the increment occurs, is selectable.

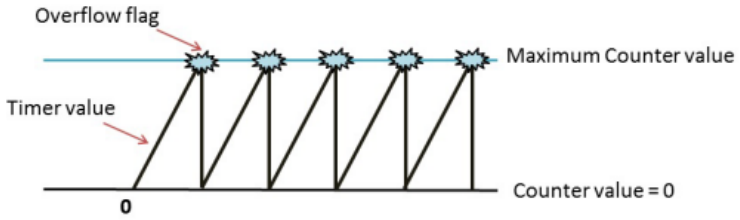
Overflow-based Timers

Overflow-based timers provide timer outputs or generate interrupts after a rollover from the maximum possible timer count.

Overflow Timer Abstraction Diagram



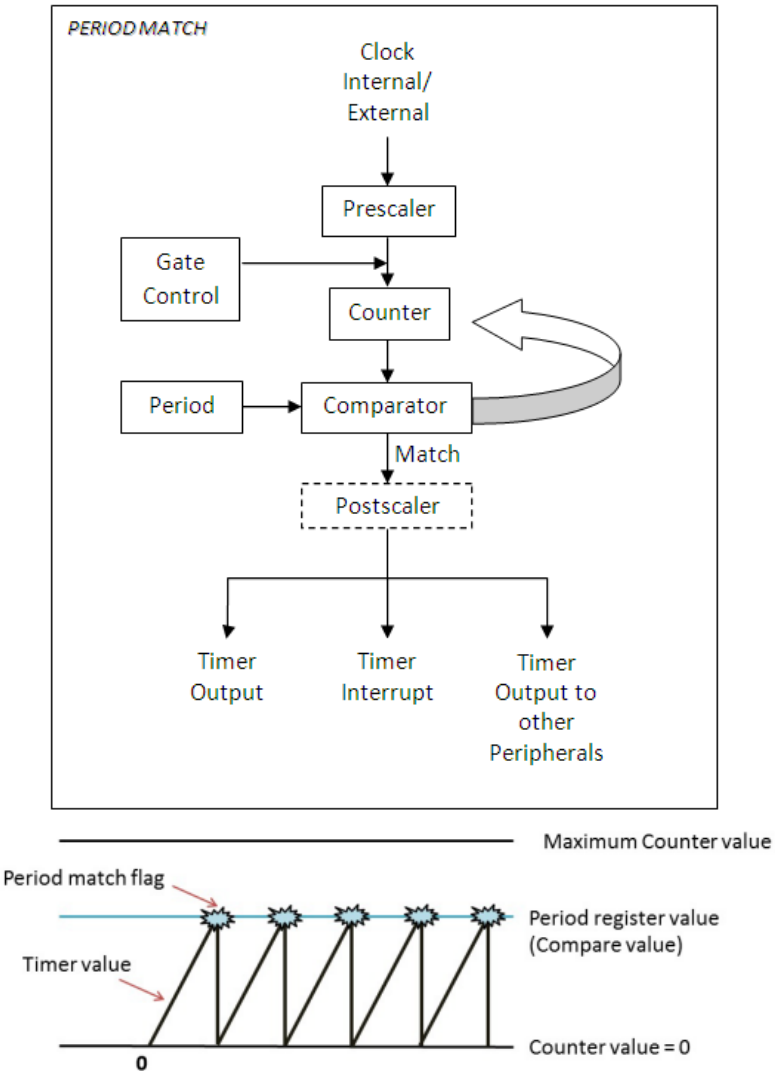
An overflow interrupt is triggered whenever the timer register overflows (i.e., reaches its maximum value based on its size as depicted by the following diagram).



Period Match-based Timers

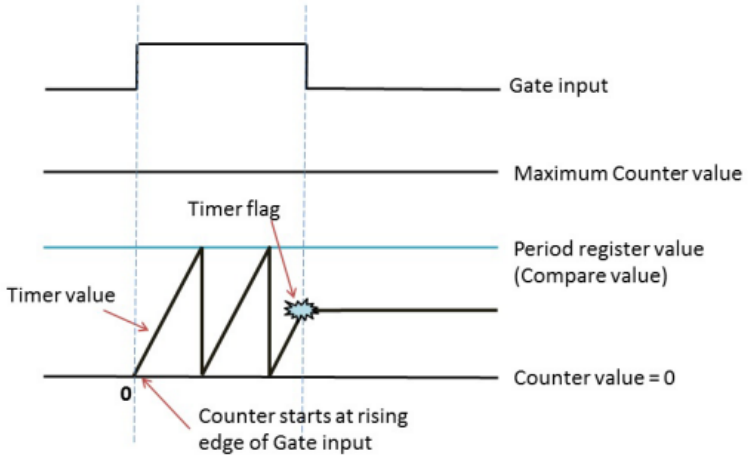
Period match-based timers provide the timer outputs or generate interrupts upon a match between the counter's count value against the preprogrammed period.

Period Match Timer Abstraction Diagram



Gated Time Accumulation/Gated Control Mode

The Gated Time Accumulation/Gate control mode allows the internal timer to increment based upon the duration of the high time applied to the timer input pin. In the Gated Time Accumulation mode, the timer clock source is derived from the internal system clock. When the input clock pin state is high, the timer will count up until a period match has occurred, or the input pin state is changed to a low state. A pin state transition from high-to-low will trigger an interrupt.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Timer module.

Library Interface Section	Description
General Setup Functions	This section provides a set of functions that are used for: • Starting and stopping the timer • Setting up Single-shot or Continuous mode of operation • Timer oscillator control • Selecting if the external event can reset the counter • Timer assignment to various modules as the clock source • Setting up the counter resolution
Power Control Functions	This section provides functions that are required for controlling the timer operation in power-saving modes such as Idle and Sleep.
Clock Source Control Functions	This section provides a set of functions that are used for: • Selecting the input clock source • Setting up external clock synchronization • Selecting the source edge to increment the counter • Getting the source edge information
Gate Control Functions	This section provides a set of functions for controlling the gating logic of the timers.
Preprocessing Functions	This section provides a set of functions that are used for prescaler setup.
Period Control Functions	This section provides a set of functions for reading and writing the period.
Counter Control Functions	This section provides a set of functions for: • Reading and writing the counter values • Asynchronous write control
Post-processing Functions	This section provides a set of functions for controlling how the Timer module operates with other peripherals. This section only covers the peripherals that can be controlled from inside of the Timer module. For peripherals that have a dedicated timer, or the control lies within the peripheral, refer to that peripheral for the interface into the timers. This section can also provide a postscaling option, if available, on the microcontroller.
Miscellaneous Functions	This section provides a set of functions that are required to read status information.

How the Library Works

Timer modules can operate in the following modes:

- Synchronous Clock Counter
- Synchronous External Clock Counter
- Asynchronous External Clock Counter
- Gated Timer

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine availability.

Synchronous Internal Clock Counter

This section provides information and examples for setting up Synchronous Internal Clock Counter operation.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine availability.

Description

Synchronous Clock Counter operation provides the following capabilities:

- Elapsed time measurements
- Time delays
- Periodic timer interrupts

Setup

Use the following steps for synchronous internal clock counter setup:

1. Some timer modules support multiple resolutions, which can be selected using PLIB_TMR_Mode16BitEnable or PLIB_TMR_Mode32BitEnable.
2. Set the start value of the timer using PLIB_TMR_Counter16BitSet or PLIB_TMR_Counter32BitSet.
3. The period register value can be set using PLIB_TMR_Period8BitSet, PLIB_TMR_Period16BitSet or PLIB_TMR_Period32BitSet. Use the appropriate function based on the resolution of the timer.
4. Clear the timer register using PLIB_TMR_Counter16BitClear or PLIB_TMR_Counter32BitClear.
5. In this mode, the input clock source for the timer is the internal clock, and is selected using PLIB_TMR_ClockSourceSelect, depending on which clock is used as the internal clock in the timer module of the device.
6. The prescaler for the clock can be selected using PLIB_TMR_PrescaleSelect. Pass in the desired prescaler into the function from the enumeration TMR_PRESCALE.
7. Start the timer once the setup is complete using PLIB_TMR_Start. The timer operation can be stopped using PLIB_TMR_Stop.

Example: Synchronous Counter (Internal Clock) Setup

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.

// Stop the timer
PLIB_TMR_Stop(MY_TIMER_INSTANCE);

// Set the prescaler, and set the clock source as internal
PLIB_TMR_PrescaleSelect(MY_TIMER_INSTANCE, TMR_PRESCALE_TMRX_VALUE_1);
PLIB_TMR_ClockSourceSelect(MY_TIMER_INSTANCE, TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);

// Clear the timer
PLIB_TMR_Counter16BitClear(MY_TIMER_INSTANCE);

// Load the period register
PLIB_TMR_Period16BitSet(MY_TIMER_INSTANCE, 0xFFFF);

// Start the timer
PLIB_TMR_Start(MY_TIMER_INSTANCE);
```

Timer interrupt is generated if enabled, when the timer count matches the period value if available, or when the counter overflows.

Example: Changing Counter Value During Timer Operation

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.
uint16_t timer_value;

// Clear the timer
timer_value = PLIB_TMR_Counter16BitGet(MY_TIMER_INSTANCE);

if(timer_value > 0x0ff0)
{
   PLIB_TMR_Counter16BitSet(MY_TIMER_INSTANCE, 0x000a);
}
```

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine availability.

Synchronous External Clock Counter

This section provides information and examples for Synchronous External Clock Counter operation.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine availability.

Description

Synchronous external clock counter operation provides the following capabilities:

- Counting periodic or non-periodic pulses
- Use external clock as time base for timers

Setup

Use the following steps for synchronous external clock counter setup:

1. Some timer modules support multiple resolutions, which can be selected using PLIB_TMR_Mode16BitEnable or PLIB_TMR_Mode32BitEnable.
2. The period register value can be set using PLIB_TMR_Period16BitSet or PLIB_TMR_Period32BitSet. Use the appropriate function based on the resolution of the timer.
3. Set the start value of the timer using PLIB_TMR_Counter16BitSet or PLIB_TMR_Counter32BitSet.
4. Some timer modules, can select an external clock as its input source. Use PLIB_TMR_ClockSourceSelect to select the external clock source.
5. The clock source is synchronized with the internal clock for synchronous operation. Use PLIB_TMR_ClockSourceExternalSyncEnable to synchronize the external clock source.
6. The prescaler for the clock can be selected using PLIB_TMR_PrescaleSelect. Pass in the desired prescaler into the function from the enumeration TMR_PRESCALE.
7. Clear the timer register using PLIB_TMR_Counter16BitClear or PLIB_TMR_Counter32BitClear.
8. Start the timer once the setup is complete using PLIB_TMR_Start. The timer operation can be stopped using PLIB_TMR_Stop.

Example: Synchronous Counter (External Clock) Setup

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.

// Stop the timer
PLIB_TMR_Stop(MY_TIMER_INSTANCE);

// Set the prescaler, and set the clock source as external and enable synchronization
PLIB_TMR_PrescaleSelect(MY_TIMER_INSTANCE, TMR_PRESCALE_VALUE_256);
PLIB_TMR_ClockSourceSelect(MY_TIMER_INSTANCE, TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN);
PLIB_TMR_ClockSourceExternalSyncEnable(MY_TIMER_INSTANCE);

// Clear the timer
PLIB_TMR_Counter16BitClear(MY_TIMER_INSTANCE);

// Load the period register
PLIB_TMR_Period16BitSet(MY_TIMER_INSTANCE, 0xFFFF);

// Start the timer
PLIB_TMR_Start(MY_TIMER_INSTANCE);
```

A timer interrupt is generated if enabled, when the timer count matches the period value if available, or when the counter overflows.

Example: Changing Counter Value During Timer Operation

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.
uint16_t timer_value;

// read the current timer value
timer_value = PLIB_TMR_Counter16BitGet(MY_TIMER_INSTANCE);

if(timer_value > 0x0ff0)
{
   PLIB_TMR_Counter16BitSet(MY_TIMER_INSTANCE, 0x000a);
}
```

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine availability.

Asynchronous Counter

This section provides information and examples for setting up Asynchronous Counter operation.

Note: Not all features are available on all devices. Refer to the **"Timer"** chapters in the specific device data sheet for availability.

Description

Asynchronous Counter operation provides the following capabilities:

- The timer can operate in Sleep mode and can generate an interrupt on a period match
- The processor can wake up from Sleep on the timer interrupt
- The timer can be clocked from the Secondary Oscillator for real-time clock applications

Setup

Use the following steps for Asynchronous Counter setup:

1. Some timer modules support multiple resolutions, which can be selected using PLIB_TMR_Mode16BitEnable or PLIB_TMR_Mode32BitEnable.
2. The period register value can be set using PLIB_TMR_Period16BitSet or PLIB_TMR_Period32BitSet. Use the appropriate function based on the resolution of the timer.
3. Set the start value of the timer using PLIB_TMR_Counter16BitSet and PLIB_TMR_Counter32BitSet.
4. Some timer modules can select an external clock as its input source. Use PLIB_TMR_ClockSourceSelect to select the external clock source.
5. For Asynchronous mode operation, the clock source is not synchronized with the internal clock. Use PLIB_TMR_ClockSourceExternalSyncDisable to disable the synchronization of the external clock source.
6. The prescaler for the clock can be selected using PLIB_TMR_PrescaleSelect. Pass in the desired prescaler into the function from the enumeration TMR_PRESCALE.
7. Clear the timer register using PLIB_TMR_Counter16BitClear or PLIB_TMR_Counter32BitClear.
8. Start the timer once the setup is complete using PLIB_TMR_Start. The timer operation can be stopped using PLIB_TMR_Stop.

Example: Asynchronous Counter Setup

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.

// Stop the timer
PLIB_TMR_Stop(MY_TIMER_INSTANCE);

// Set the prescaler, and set the clock source as external and disable synchronization
PLIB_TMR_PrescaleSelect(MY_TIMER_INSTANCE, TMR_PRESCALE_VALUE_64);
PLIB_TMR_ClockSourceSelect(MY_TIMER_INSTANCE, TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN);
PLIB_TMR_ClockSourceExternalSyncDisable(MY_TIMER_INSTANCE);

// Clear the timer
PLIB_TMR_Counter16BitClear(MY_TIMER_INSTANCE);

// Load the period register
PLIB_TMR_Period16BitSet(MY_TIMER_INSTANCE, 0x7F);

// Start the timer
PLIB_TMR_Start(MY_TIMER_INSTANCE);
```

A timer interrupt is generated if enabled, when the timer count matches the period value if available, or when the counter overflows.

Note: Not all features are available on all devices. Refer to the **"Timer"** chapters in the specific device data sheet for availability.

Gated Timer

This section provides information and examples for setting up Gated Timer operation.

Note: Not all features are available on all devices. Refer to the **"Timer"** chapters in the specific device data sheet for availability.

Description

Gated Timer operation can be used to allow measurement of an external signal.

Setup

Use the following steps for Gated Timer setup:

1. Some timer modules support multiple resolutions, which can be selected using PLIB_TMR_Mode16BitEnable or PLIB_TMR_Mode32BitEnable.
2. The period register value can be set using PLIB_TMR_Period16BitSet or PLIB_TMR_Period32BitSet. Use the appropriate function based on the resolution of the timer.
3. Set the start value of the timer using PLIB_TMR_Counter16BitSet and PLIB_TMR_Counter32BitSet.
4. In this mode, the input clock source for the timer is the internal clock, and is selected using PLIB_TMR_ClockSourceSelect, depending on which clock is used as the internal clock in the timer module of the device.
5. The prescaler for the clock can be selected using PLIB_TMR_PrescaleSelect. Pass in the desired pre scaler into the function from the enumeration TMR_PRESCALE.
6. Clear the timer register using PLIB_TMR_Counter16BitClear or PLIB_TMR_Counter32BitClear.
7. Gated operation can be enabled using the function PLIB_TMR_GateEnable.
8. Start the timer once the setup is complete using PLIB_TMR_Start. The timer operation can be stopped using PLIB_TMR_Stop.

Example: Gated Timer Setup

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the application
// developer.

// Stop the timer
PLIB_TMR_Stop(MY_TIMER_INSTANCE);

// Set the prescaler, and set the clock source as external and disable synchronization
PLIB_TMR_PrescaleSelect(MY_TIMER_INSTANCE, TMR_PRESCALE_VALUE_64);
PLIB_TMR_ClockSourceSelect(MY_TIMER_INSTANCE, TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN);
PLIB_TMR_ClockSourceExternalSyncDisable(MY_TIMER_INSTANCE);

// Clear the timer
PLIB_TMR_Counter16BitClear(MY_TIMER_INSTANCE);

// Load the period register
PLIB_TMR_Period16BitSet(MY_TIMER_INSTANCE, 0x7F00);

// Enable the gate timer
PLIB_TMR_GateEnable(MY_TIMER_INSTANCE);

// Start the timer
PLIB_TMR_Start(MY_TIMER_INSTANCE);
```

A timer interrupt is generated if enabled, on the falling edge of the gate signal.

Note: Not all features are available on all devices. Refer to the **"Timer"** chapters in the specific device data sheet for availability.

Other Features

This section provides information on additional features.

Description

Each of the following modes can additionally support some or all of the features listed. Please refer to the specific device data sheet or family reference manual for details.

Operation During Sleep Mode

When the Timer module is configured to operate in Asynchronous mode (with an external clock source), it will continue to increment each timer clock (or prescale multiple of clocks) during Sleep mode.

Operation During Idle Mode

A Timer module can continue to operate in Idle mode by calling PLIB_TMR_StopInIdleDisable, or can stop operation in Idle mode by calling PLIB_TMR_StopInIdleEnable.

Timer Mode Control

- PLIB_TMR_Mode16BitEnable: This interface enables 16-bit mode and disables 32-bit mode
- PLIB_TMR_Mode32BitEnable: This interface enables 32-bit mode and disables 16-bit mode

Timer Usage Mode

Some timers operate in either Period Match mode or Overflow mode. The mode state can be retrieved from the Timer Peripheral Library using PLI_TMR_IsPeriodMatchBased. This function returns 'true' Period Match mode and 'false' for Overflow mode.

Timer Prescaler Information

Prescaler divisor information associated with the respective prescaler value selected through TMR_PRESCALE can be retrieved through PLIB_TMR_PrescaleDivisorGet.

The true prescaler value can be obtained by using PLIB_TMR_PrescaleGet.

Configuring the Library

The library is configured for the supported Timer modules when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Setup Functions

	Name	Description
	PLIB_TMR_Mode16BitEnable	Enables the Timer module for 16-bit operation and disables all other modes.
	PLIB_TMR_Mode32BitEnable	Enables 32-bit operation on the Timer module combination.
	PLIB_TMR_Start	Starts/enables the indexed timer.
	PLIB_TMR_Stop	Stops/disables the indexed timer.

b) Power Control Functions

	Name	Description
	PLIB_TMR_StopInIdleDisable	Continue module operation when the device enters Idle mode.
	PLIB_TMR_StopInIdleEnable	Discontinues module operation when the device enters Idle mode.

c) Clock Source Control Functions

	Name	Description
	PLIB_TMR_ClockSourceExternalSyncDisable	Disables the clock synchronization of the external input.
	PLIB_TMR_ClockSourceExternalSyncEnable	Enables the clock synchronization of the external input.
	PLIB_TMR_ClockSourceSelect	Selects the clock source.

d) Gate Control Functions

	Name	Description
	PLIB_TMR_GateDisable	Enables counting regardless of the corresponding timer gate function.
	PLIB_TMR_GateEnable	Enables counting controlled by the corresponding gate function.

e) Preprocessing Functions

	Name	Description
	PLIB_TMR_PrescaleDivisorGet	Gets the prescaler divisor information.
	PLIB_TMR_PrescaleGet	Gets the prescaler divisor value.
	PLIB_TMR_PrescaleSelect	Selects the clock prescaler.

f) Period Control Functions

	Name	Description
	PLIB_TMR_Period16BitGet	Gets the 16-bit period value.
	PLIB_TMR_Period16BitSet	Sets the 16-bit period value.
	PLIB_TMR_Period32BitGet	Gets the 32-bit period value.
	PLIB_TMR_Period32BitSet	Sets the 32-bit period value.

g) Counter Control Functions

	Name	Description
	PLIB_TMR_Counter16BitClear	Clears the 16-bit timer value.
	PLIB_TMR_Counter16BitGet	Gets the 16-bit timer value.
	PLIB_TMR_Counter16BitSet	Sets the 16-bit timer value.
	PLIB_TMR_Counter32BitClear	Clears the 32-bit timer value.
	PLIB_TMR_Counter32BitGet	Gets the 32-bit timer value.
	PLIB_TMR_Counter32BitSet	Sets the 32-bit timer value.

	PLIB_TMR_CounterAsyncWriteDisable	Disables the writes to the counter register until the pending write operation completes.
	PLIB_TMR_CounterAsyncWriteEnable	Enables back-to-back writes with legacy asynchronous timer functionality.
	PLIB_TMR_CounterAsyncWriteInProgress	Returns the status of the counter write in Asynchronous mode.

i) Miscellaneous Functions

	Name	Description
	PLIB_TMR_IsPeriodMatchBased	Gets the operating mode state of the Timer module based on Period Match or Overflow mode.

j) Feature Existence Functions

	Name	Description
	PLIB_TMR_ExistsClockSource	Identifies whether the ClockSource feature exists on the Timer module.
	PLIB_TMR_ExistsClockSourceSync	Identifies whether the ClockSourceSync feature exists on the Timer module.
	PLIB_TMR_ExistsCounter16Bit	Identifies whether the Counter16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounter32Bit	Identifies whether the Counter32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounterAsyncWriteControl	Identifies whether the CounterAsyncWriteControl feature exists on the Timer module.
	PLIB_TMR_ExistsCounterAsyncWriteInProgress	Identifies whether the CounterAsyncWriteInProgress feature exists on the Timer module.
	PLIB_TMR_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Timer module.
	PLIB_TMR_ExistsGatedTimeAccumulation	Identifies whether the GatedTimeAccumulation feature exists on the Timer module.
	PLIB_TMR_ExistsMode16Bit	Identifies whether the Mode16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsMode32Bit	Identifies whether the Mode32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPeriod16Bit	Identifies whether the Period16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPeriod32Bit	Identifies whether the Period32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPrescale	Identifies whether the Prescale feature exists on the Timer module.
	PLIB_TMR_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the Timer module.
	PLIB_TMR_ExistsTimerOperationMode	Identifies whether the TimerOperationMode feature exists on the Timer module.

k) Data Types and Constants

	Name	Description
	TMR_CLOCK_SOURCE	Data type defining the different clock sources.
	TMR_MODULE_ID	Identifies the supported Timer modules.
	TMR_PRESCALE	Defines the list of possible prescaler values.

Description

This section describes the Application Programming Interface (API) functions of the Timer Peripheral Library.

Refer to each section for a detailed description.

a) General Setup Functions

PLIB_TMR_Mode16BitEnable Function

Enables the Timer module for 16-bit operation and disables all other modes.

File

plib_tmr.h

C

```
void PLIB_TMR_Mode16BitEnable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function enables the Timer module for 16-bit operation and disables all other modes.

Remarks

Calling this function disables the operation of the Timer module 8-bit mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsMode16Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Mode16BitEnable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_Mode16BitEnable(TMR_MODULE_ID index)

PLIB_TMR_Mode32BitEnable Function

Enables 32-bit operation on the Timer module combination.

File

plib_tmr.h

C

```
void PLIB_TMR_Mode32BitEnable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function enables the Timer module and the index +1 Timer module to act as one 32-bit timer module and disables all other modes.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsMode32Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Mode32BitEnable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_Mode32BitEnable(TMR_MODULE_ID index)

PLIB_TMR_Start Function

Starts/enables the indexed timer.

File

plib_tmr.h

C

```
void PLIB_TMR_Start(TMR_MODULE_ID index);
```

Returns

None.

Description

This function starts/enables the indexed timer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Start(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_Start(TMR_MODULE_ID index)

PLIB_TMR_Stop Function

Stops/disables the indexed timer.

File

plib_tmr.h

C

```
void PLIB_TMR_Stop(TMR_MODULE_ID index);
```

Returns

None.

Description

This function stops/disables the indexed timer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Stop(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_Stop(TMR_MODULE_ID index)

b) Power Control Functions

PLIB_TMR_StopInIdleDisable Function

Continue module operation when the device enters Idle mode.

File

plib_tmr.h

C

```
void PLIB_TMR_StopInIdleDisable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function continues module operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_StopInIdleDisable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_StopInIdleDisable(TMR_MODULE_ID index)

PLIB_TMR_StopInIdleEnable Function

Discontinues module operation when the device enters Idle mode.

File

plib_tmr.h

C

```
void PLIB_TMR_StopInIdleEnable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function discontinues module operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsStopInIdleControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_StopInIdleEnable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_StopInIdleEnable(TMR_MODULE_ID index)

c) Clock Source Control Functions

PLIB_TMR_ClockSourceExternalSyncDisable Function

Disables the clock synchronization of the external input.

File

plib_tmr.h

C

```
void PLIB_TMR_ClockSourceExternalSyncDisable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function disables the clock synchronization of the external input.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsClockSourceSync in your application to determine whether this feature is available.

Preconditions

The timer module must be configured to use the external clock using the function PLIB_TMR_ClockSourceSelect with the clock source as TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN. If the function PLIB_TMR_ClockSourceSelect configures the clock with some other enumeration, this function will have no effect.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_ClockSourceExternalSyncDisable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_ClockSourceExternalSyncDisable(TMR_MODULE_ID index)

PLIB_TMR_ClockSourceExternalSyncEnable Function

Enables the clock synchronization of the external input.

File

plib_tmr.h

C

```
void PLIB_TMR_ClockSourceExternalSyncEnable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function enables the clock synchronization of the external input.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsClockSourceSync in your application to determine whether this feature is available.

Preconditions

The timer module must be configured to use the external clock using the function PLIB_TMR_ClockSourceSelect with the clock source as TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN. If the function PLIB_TMR_ClockSourceSelect configures the clock with some other enumeration, this function will have no effect.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_ClockSourceExternalSyncEnable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_ClockSourceExternalSyncEnable(TMR_MODULE_ID index)

PLIB_TMR_ClockSourceSelect Function

Selects the clock source.

File

plib_tmr.h

C

```
void PLIB_TMR_ClockSourceSelect(TMR_MODULE_ID index, TMR_CLOCK_SOURCE source);
```

Returns

None.

Description

This function selects the clock source.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsClockSource in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_ClockSourceSelect(MY_TMR_INSTANCE, TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

source	One of the possible values of TMR_CLOCK_SOURCE

Function

void PLIB_TMR_ClockSourceSelect(TMR_MODULE_ID index, TMR_CLOCK_SOURCE source)

d) Gate Control Functions

PLIB_TMR_GateDisable Function

Enables counting regardless of the corresponding timer gate function.

File

plib_tmr.h

C

```
void PLIB_TMR_GateDisable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function enables counting regardless of the gate function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsGatedTimeAccumulation in your application to determine whether this feature is available.

Preconditions

The timer must be enabled for this function to take effect.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_GateDisable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_GateDisable(TMR_MODULE_ID index)

PLIB_TMR_GateEnable Function

Enables counting controlled by the corresponding gate function.

File

plib_tmr.h

C

```
void PLIB_TMR_GateEnable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function enables counting controlled by the gate function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_TMR_ExistsGatedTimeAccumulation in your application to determine whether this feature is available.

Preconditions

The timer must be enabled, for this function to take effect.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_GateEnable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_GateEnable(TMR_MODULE_ID index)

e) Preprocessing Functions

PLIB_TMR_PrescaleDivisorGet Function

Gets the prescaler divisor information.

File

plib_tmr.h

C

```
uint16_t PLIB_TMR_PrescaleDivisorGet(TMR_MODULE_ID index, TMR_PRESCALE prescale);
```

Returns

prescale divisor value

Description

This function returns the prescaler divisor information.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPrescale in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint16_t div = PLIB_TMR_PrescaleDivisorGet (MY_TMR_INSTANCE, TMR_PRESCALE_VALUE_1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
prescale	One of the possible values of TMR_PRESCALE

Function

uint16_t PLIB_TMR_PrescaleDivisorGet(TMR_MODULE_ID index, TMR_PRESCALE prescale)

PLIB_TMR_PrescaleGet Function

Gets the prescaler divisor value.

File

plib_tmr.h

C

```
uint16_t PLIB_TMR_PrescaleGet(TMR_MODULE_ID index);
```

Returns

Prescaler divisor value.

Description

This function returns the prescaler divisor value. The value returned will be direct number and not the enum equivalent.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPrescale in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint16_t prescale;
prescale = PLIB_TMR_PrescaleGet (MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint16_t PLIB_TMR_PrescaleGet (TMR_MODULE_ID index)

PLIB_TMR_PrescaleSelect Function

Selects the clock prescaler.

File

plib_tmr.h

C

```
void PLIB_TMR_PrescaleSelect(TMR_MODULE_ID index, TMR_PRESCALE prescale);
```

Returns

None.

Description

This function selects the clock prescaler.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPrescale in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_PrescaleSelect(MY_TMR_INSTANCE, TMR_PRESCALE_VALUE_8);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
prescale	One of the possible values of TMR_PRESCALE

Function

void PLIB_TMR_PrescaleSelect(TMR_MODULE_ID index, TMR_PRESCALE prescale)

f) Period Control Functions

PLIB_TMR_Period16BitGet Function

Gets the 16-bit period value.

File

plib_tmr.h

C

```
uint16_t PLIB_TMR_Period16BitGet(TMR_MODULE_ID index);
```

Returns

period - 16-bit period value

Description

This function gets the 16-bit period value.

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPeriod16Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint16_t period = PLIB_TMR_Period16BitGet(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint16_t PLIB_TMR_Period16BitGet(TMR_MODULE_ID index)

PLIB_TMR_Period16BitSet Function

Sets the 16-bit period value.

File

plib_tmr.h

C

```
void PLIB_TMR_Period16BitSet(TMR_MODULE_ID index, uint16_t period);
```

Returns

None.

Description

This function sets the 16-bit period value.

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPeriod16Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
// where, MY_PERIOD_VALUE is the 16-bit value which needs to be stored in the
// period register.
PLIB_TMR_Period16BitSet(MY_TMR_INSTANCE, MY_PERIOD_VALUE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
period	16-bit period register value

Function

void PLIB_TMR_Period16BitSet(TMR_MODULE_ID index, uint16_t period)

PLIB_TMR_Period32BitGet Function

Gets the 32-bit period value.

File

plib_tmr.h

C

```
uint32_t PLIB_TMR_Period32BitGet(TMR_MODULE_ID index);
```

Returns

period - 32-bit period value

Description

This function gets the 32-bit period value.

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPeriod32Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint32_t period = PLIB_TMR_Period32BitGet(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_TMR_Period32BitGet(TMR_MODULE_ID index)

PLIB_TMR_Period32BitSet Function

Sets the 32-bit period value.

File

plib_tmr.h

C

```
void PLIB_TMR_Period32BitSet(TMR_MODULE_ID index, uint32_t period);
```

Returns

None.

Description

This function sets the 32-bit period value.

Remarks

The timer period register may be written at any time before the timer is started or after the timer is stopped. The timer period register can also be written when servicing the interrupt for the timer. When the timer is in operation, it is not recommended to write to the period register.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsPeriod32Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
// where, MY_PERIOD_VALUE is the 32-bit value which needs to be stored in the
// period register.
PLIB_TMR_Period32BitSet(MY_TMR_INSTANCE, MY_PERIOD_VALUE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
period	32-bit period register value

Function

void PLIB_TMR_Period32BitSet(TMR_MODULE_ID index, uint32_t period)

g) Counter Control Functions

PLIB_TMR_Counter16BitClear Function

Clears the 16-bit timer value.

File

plib_tmr.h

C

```
void PLIB_TMR_Counter16BitClear(TMR_MODULE_ID index);
```

Returns

None.

Description

This function clears the 16-bit timer value.

Remarks

When the timer register is written, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounter16Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Counter16BitClear(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_TMR_Counter16BitClear(TMR_MODULE_ID index)

PLIB_TMR_Counter16BitGet Function

Gets the 16-bit timer value.

File

plib_tmr.h

C

```
uint16_t PLIB_TMR_Counter16BitGet(TMR_MODULE_ID index);
```

Returns

16-bit timer value.

Description

This function gets the 16-bit timer value.

Remarks

PLIB_TMR_Counter16BitGet does not prevent the timer from incrementing during the same instruction cycle.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounter16Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint16_t timerValue = PLIB_TMR_Counter16BitGet(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint16_t PLIB_TMR_Counter16BitGet(TMR_MODULE_ID index)

PLIB_TMR_Counter16BitSet Function

Sets the 16-bit timer value.

File

plib_tmr.h

C

```
void PLIB_TMR_Counter16BitSet(TMR_MODULE_ID index, uint16_t value);
```

Returns

None.

Description

This function sets the 16-bit timer value.

Remarks

When the timer register is written to, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounter16Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Counter16BitSet(MY_TMR_INSTANCE, 0x100);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	16-bit value to be set

Function

void PLIB_TMR_Counter16BitSet(TMR_MODULE_ID index, uint16_t value)

PLIB_TMR_Counter32BitClear Function

Clears the 32-bit timer value.

File

plib_tmr.h

C

```
void PLIB_TMR_Counter32BitClear(TMR_MODULE_ID index);
```

Returns

None.

Description

This function clears the 32-bit timer value.

Remarks

When the timer register is written, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounter32Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
```

```
// application developer.
PLIB_TMR_Counter32BitClear(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_TMR_Counter32BitClear(TMR_MODULE_ID index)

PLIB_TMR_Counter32BitGet Function

Gets the 32-bit timer value.

File

plib_tmr.h

C

```
uint32_t PLIB_TMR_Counter32BitGet(TMR_MODULE_ID index);
```

Returns

32 bit timer value.

Description

This function gets the 32-bit timer value.

Remarks

PLIB_TMR_Counter32BitGet does not prevent the timer from incrementing during the same instruction cycle.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounter32Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
uint32_t timerValue = PLIB_TMR_Counter32BitGet(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint32_t PLIB_TMR_Counter32BitGet(TMR_MODULE_ID index)

PLIB_TMR_Counter32BitSet Function

Sets the 32-bit timer value.

File

plib_tmr.h

C

```
void PLIB_TMR_Counter32BitSet(TMR_MODULE_ID index, uint32_t value);
```

Returns

None.

Description

This function sets the 32-bit timer value.

Remarks

When the timer register is written, the timer increment does not occur for that cycle. Writes to the timer register in the asynchronous counter mode should be avoided.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounter32Bit in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_Counter32BitSet(MY_TMR_INSTANCE, 0x1000000);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	32-bit value to be set

Function

void PLIB_TMR_Counter32BitSet(TMR_MODULE_ID index, uint32_t value)

PLIB_TMR_CounterAsyncWriteDisable Function

Disables the writes to the counter register until the pending write operation completes.

File

plib_tmr.h

C

```
void PLIB_TMR_CounterAsyncWriteDisable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function disables the writes to the counter register until the pending write operation completes.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounterAsyncWriteControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_CounterAsyncWriteDisable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_CounterAsyncWriteDisable(TMR_MODULE_ID index)

PLIB_TMR_CounterAsyncWriteEnable Function

Enables back-to-back writes with legacy asynchronous timer functionality.

File

plib_tmr.h

C

```
void PLIB_TMR_CounterAsyncWriteEnable(TMR_MODULE_ID index);
```

Returns

None.

Description

This function enables back-to-back writes with legacy asynchronous timer functionality.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounterAsyncWriteControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
PLIB_TMR_CounterAsyncWriteEnable(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_TMR_CounterAsyncWriteEnable(TMR_MODULE_ID index)

PLIB_TMR_CounterAsyncWriteInProgress Function

Returns the status of the counter write in Asynchronous mode.

File

plib_tmr.h

C

```
bool PLIB_TMR_CounterAsyncWriteInProgress(TMR_MODULE_ID index);
```

Returns

- true - Write is in progress
- false - Write is complete

Description

This function returns the status of the counter write in Asynchronous mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsCounterAsyncWriteInProgress in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
bool inProgress = PLIB_TMR_CounterAsyncWriteInProgress(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_TMR_CounterAsyncWriteInProgress(TMR_MODULE_ID index)

h) Post-processing Functions

i) Miscellaneous Functions

PLIB_TMR_IsPeriodMatchBased Function

Gets the operating mode state of the Timer module based on Period Match or Overflow mode.

File

plib_tmr.h

C

```
bool PLIB_TMR_IsPeriodMatchBased(TMR_MODULE_ID index);
```

Returns

- true - Operation in Period Match mode
- false - Operation in Overflow mode

Description

This function gets the operating mode state of the Timer module based on Period Match or Overflow mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_TMR_ExistsTimerOperationMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_TMR_INSTANCE, is the timer instance selected for use by the
// application developer.
bool status = PLIB_TMR_IsPeriodMatchBased(MY_TMR_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_TMR_IsPeriodMatchBased (TMR_MODULE_ID index)

j) Feature Existence Functions

PLIB_TMR_ExistsClockSource Function

Identifies whether the ClockSource feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsClockSource(TMR_MODULE_ID index);
```

Returns

- true - The ClockSource feature is supported on the device
- false - The ClockSource feature is not supported on the device

Description

This function identifies whether the ClockSource feature is available on the Timer module. When this function returns true, this function is supported on the device:

- PLIB_TMR_ClockSourceSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsClockSource(TMR_MODULE_ID index)

PLIB_TMR_ExistsClockSourceSync Function

Identifies whether the ClockSourceSync feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsClockSourceSync(TMR_MODULE_ID index);
```

Returns

- true - The ClockSourceSync feature is supported on the device
- false - The ClockSourceSync feature is not supported on the device

Description

This function identifies whether the ClockSourceSync feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_ClockSourceExternalSyncEnable
- PLIB_TMR_ClockSourceExternalSyncDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsClockSourceSync(TMR_MODULE_ID index)

PLIB_TMR_ExistsCounter16Bit Function

Identifies whether the Counter16Bit feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsCounter16Bit(TMR_MODULE_ID index);
```

Returns

- true - The Counter16Bit feature is supported on the device
- false - The Counter16Bit feature is not supported on the device

Description

This function identifies whether the Counter16Bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_Counter16BitSet
- PLIB_TMR_Counter16BitGet
- PLIB_TMR_Counter16BitClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsCounter16Bit(TMR_MODULE_ID index)

PLIB_TMR_ExistsCounter32Bit Function

Identifies whether the Counter32Bit feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsCounter32Bit(TMR_MODULE_ID index);
```

Returns

- true - The Counter32Bit feature is supported on the device
- false - The Counter32Bit feature is not supported on the device

Description

This function identifies whether the Counter32Bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_Counter32BitSet
- PLIB_TMR_Counter32BitGet
- PLIB_TMR_Counter32BitClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsCounter32Bit(TMR_MODULE_ID index)

PLIB_TMR_ExistsCounterAsyncWriteControl Function

Identifies whether the CounterAsyncWriteControl feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsCounterAsyncWriteControl(TMR_MODULE_ID index);
```

Returns

- true - The CounterAsyncWriteControl feature is supported on the device
- false - The CounterAsyncWriteControl feature is not supported on the device

Description

This function identifies whether the CounterAsyncWriteControl feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_CounterAsyncWriteEnable
- PLIB_TMR_CounterAsyncWriteDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsCounterAsyncWriteControl(TMR_MODULE_ID index)

PLIB_TMR_ExistsCounterAsyncWriteInProgress Function

Identifies whether the CounterAsyncWriteInProgress feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsCounterAsyncWriteInProgress(TMR_MODULE_ID index);
```

Returns

- true - The CounterAsyncWriteInProgress feature is supported on the device
- false - The CounterAsyncWriteInProgress feature is not supported on the device

Description

This function identifies whether the CounterAsyncWriteInProgress feature is available on the Timer module. When this function returns true, this function is supported on the device:

- PLIB_TMR_CounterAsyncWriteInProgress

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsCounterAsyncWriteInProgress(TMR_MODULE_ID index)

PLIB_TMR_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsEnableControl(TMR_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_Start
- PLIB_TMR_Stop

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsEnableControl(TMR_MODULE_ID index)

PLIB_TMR_ExistsGatedTimeAccumulation Function

Identifies whether the GatedTimeAccumulation feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsGatedTimeAccumulation(TMR_MODULE_ID index);
```

Returns

- true - The GatedTimeAccumulation feature is supported on the device
- false - The GatedTimeAccumulation feature is not supported on the device

Description

This function identifies whether the GatedTimeAccumulation feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_GateEnable
- PLIB_TMR_GateDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsGatedTimeAccumulation(TMR_MODULE_ID index)

PLIB_TMR_ExistsMode16Bit Function

Identifies whether the Mode16Bit feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsMode16Bit(TMR_MODULE_ID index);
```

Returns

- true - The Mode16Bit feature is supported on the device
- false - The Mode16Bit feature is not supported on the device

Description

This function identifies whether the Mode16Bit feature is available on the Timer module. When this function returns true, this function is supported on the device:

- PLIB_TMR_Mode16BitEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsMode16Bit(TMR_MODULE_ID index)

PLIB_TMR_ExistsMode32Bit Function

Identifies whether the Mode32Bit feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsMode32Bit(TMR_MODULE_ID index);
```

Returns

- true - The Mode32Bit feature is supported on the device
- false - The Mode32Bit feature is not supported on the device

Description

This function identifies whether the Mode32Bit feature is available on the Timer module. When this function returns true, this function is supported on the device:

- PLIB_TMR_Mode32BitEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsMode32Bit(TMR_MODULE_ID index)

PLIB_TMR_ExistsPeriod16Bit Function

Identifies whether the Period16Bit feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsPeriod16Bit(TMR_MODULE_ID index);
```

Returns

- true - The Period16Bit feature is supported on the device
- false - The Period16Bit feature is not supported on the device

Description

This function identifies whether the Period16Bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_Period16BitSet
- PLIB_TMR_Period16BitGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsPeriod16Bit(TMR_MODULE_ID index)

PLIB_TMR_ExistsPeriod32Bit Function

Identifies whether the Period32Bit feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsPeriod32Bit(TMR_MODULE_ID index);
```

Returns

- true - The Period32Bit feature is supported on the device
- false - The Period32Bit feature is not supported on the device

Description

This function identifies whether the Period32Bit feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_Period32BitSet
- PLIB_TMR_Period32BitGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsPeriod32Bit(TMR_MODULE_ID index)

PLIB_TMR_ExistsPrescale Function

Identifies whether the Prescale feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsPrescale(TMR_MODULE_ID index);
```

Returns

- true - The Prescale feature is supported on the device
- false - The Prescale feature is not supported on the device

Description

This function identifies whether the Prescale feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_PrescaleSelect
- PLIB_TMR_PrescaleGet
- PLIB_TMR_PrescaleDivisorGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsPrescale(TMR_MODULE_ID index)

PLIB_TMR_ExistsStopInIdleControl Function

Identifies whether the StopInIdle feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsStopInIdleControl(TMR_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the Timer module. When this function returns true, these functions are supported on the device:

- PLIB_TMR_StopInIdleEnable
- PLIB_TMR_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsStopInIdleControl(TMR_MODULE_ID index)

PLIB_TMR_ExistsTimerOperationMode Function

Identifies whether the TimerOperationMode feature exists on the Timer module.

File

plib_tmr.h

C

```
bool PLIB_TMR_ExistsTimerOperationMode(TMR_MODULE_ID index);
```

Returns

- true - The TimerOperationMode feature is supported on the device
- false - The TimerOperationMode feature is not supported on the device

Description

This function identifies whether the TimerOperationMode feature is available on the Timer module. When this function returns true, this function is supported on the device:

- PLIB_TMR_IsPeriodMatchBased

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_TMR_ExistsTimerOperationMode(TMR_MODULE_ID index)

k) Data Types and Constants

TMR_CLOCK_SOURCE Enumeration

Data type defining the different clock sources.

File

help_plib_tmr.h

C

```
typedef enum {
  TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK,
  TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN
} TMR_CLOCK_SOURCE;
```

Members

Members	Description
TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK	Clock source is the internal clock = FOSC/n (where n, is processor specific)
TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN	Clock source is the external input pin

Description

Clock sources selection enumeration

This data type defines the different clock sources.

TMR_MODULE_ID Enumeration

Identifies the supported Timer modules.

File

help_plib_tmr.h

C

```
typedef enum {
  TMR_ID_0,
  TMR_ID_1,
  TMR_ID_2,
  TMR_ID_3,
  TMR_ID_4,
  TMR_ID_5,
  TMR_ID_6,
  TMR_ID_7,
  TMR_ID_8,
  TMR_ID_9,
  TMR_NUMBER_OF_MODULES
} TMR_MODULE_ID;
```

Members

Members	Description
TMR_ID_0	Timer 0
TMR_ID_1	Timer 1
TMR_ID_2	Timer 2
TMR_ID_3	Timer 3
TMR_ID_4	Timer 4

TMR_ID_5	Timer 5
TMR_ID_6	Timer 6
TMR_ID_7	Timer 7
TMR_ID_8	Timer 8
TMR_ID_9	Timer 9
TMR_NUMBER_OF_MODULES	Max number of timers

Description

TMR Module ID

This enumeration identifies the available Timer modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Not all modules will be available on all microcontrollers. Refer to the data sheet for the specific controller in use to determine which modules are supported. The numbers used in the enumeration values will match the numbers provided in the data sheet.

TMR_PRESCALE Enumeration

Defines the list of possible prescaler values.

File

help_plib_tmr.h

C

```
typedef enum {
  TMR_PRESCALE_VALUE_1,
  TMR_PRESCALE_VALUE_2,
  TMR_PRESCALE_VALUE_4,
  TMR_PRESCALE_VALUE_8,
  TMR_PRESCALE_VALUE_16,
  TMR_PRESCALE_VALUE_32,
  TMR_PRESCALE_VALUE_64,
  TMR_PRESCALE_VALUE_256
} TMR_PRESCALE;
```

Members

Members	Description
TMR_PRESCALE_VALUE_1	Timer Prescaler Value 1:1
TMR_PRESCALE_VALUE_2	Timer Prescaler Value 1:2
TMR_PRESCALE_VALUE_4	Timer Prescaler Value 1:4
TMR_PRESCALE_VALUE_8	Timer Prescaler Value 1:8
TMR_PRESCALE_VALUE_16	Timer Prescaler Value 1:16
TMR_PRESCALE_VALUE_32	Timer Prescaler Value 1:32
TMR_PRESCALE_VALUE_64	Timer Prescaler Value 1:64
TMR_PRESCALE_VALUE_256	Timer Prescaler Value 1:256

Description

Prescaler Values

This macro defines the list of possible prescaler values.

Files

Files

Name	Description
plib_tmr.h	Timer Peripheral Library interface header.
help_plib_tmr.h	This is file help_plib_tmr.h.

Description

This section lists the source and header files used by the library.

plib_tmr.h

Timer Peripheral Library interface header.

Functions

	Name	Description
	PLIB_TMR_ClockSourceExternalSyncDisable	Disables the clock synchronization of the external input.
	PLIB_TMR_ClockSourceExternalSyncEnable	Enables the clock synchronization of the external input.
	PLIB_TMR_ClockSourceSelect	Selects the clock source.
	PLIB_TMR_Counter16BitClear	Clears the 16-bit timer value.
	PLIB_TMR_Counter16BitGet	Gets the 16-bit timer value.
	PLIB_TMR_Counter16BitSet	Sets the 16-bit timer value.
	PLIB_TMR_Counter32BitClear	Clears the 32-bit timer value.
	PLIB_TMR_Counter32BitGet	Gets the 32-bit timer value.
	PLIB_TMR_Counter32BitSet	Sets the 32-bit timer value.
	PLIB_TMR_CounterAsyncWriteDisable	Disables the writes to the counter register until the pending write operation completes.
	PLIB_TMR_CounterAsyncWriteEnable	Enables back-to-back writes with legacy asynchronous timer functionality.
	PLIB_TMR_CounterAsyncWriteInProgress	Returns the status of the counter write in Asynchronous mode.
	PLIB_TMR_ExistsClockSource	Identifies whether the ClockSource feature exists on the Timer module.
	PLIB_TMR_ExistsClockSourceSync	Identifies whether the ClockSourceSync feature exists on the Timer module.
	PLIB_TMR_ExistsCounter16Bit	Identifies whether the Counter16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounter32Bit	Identifies whether the Counter32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsCounterAsyncWriteControl	Identifies whether the CounterAsyncWriteControl feature exists on the Timer module.
	PLIB_TMR_ExistsCounterAsyncWriteInProgress	Identifies whether the CounterAsyncWriteInProgress feature exists on the Timer module.
	PLIB_TMR_ExistsEnableControl	Identifies whether the EnableControl feature exists on the Timer module.
	PLIB_TMR_ExistsGatedTimeAccumulation	Identifies whether the GatedTimeAccumulation feature exists on the Timer module.
	PLIB_TMR_ExistsMode16Bit	Identifies whether the Mode16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsMode32Bit	Identifies whether the Mode32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPeriod16Bit	Identifies whether the Period16Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPeriod32Bit	Identifies whether the Period32Bit feature exists on the Timer module.
	PLIB_TMR_ExistsPrescale	Identifies whether the Prescale feature exists on the Timer module.
	PLIB_TMR_ExistsStopInIdleControl	Identifies whether the StopInIdle feature exists on the Timer module.
	PLIB_TMR_ExistsTimerOperationMode	Identifies whether the TimerOperationMode feature exists on the Timer module.
	PLIB_TMR_GateDisable	Enables counting regardless of the corresponding timer gate function.
	PLIB_TMR_GateEnable	Enables counting controlled by the corresponding gate function.
	PLIB_TMR_IsPeriodMatchBased	Gets the operating mode state of the Timer module based on Period Match or Overflow mode.
	PLIB_TMR_Mode16BitEnable	Enables the Timer module for 16-bit operation and disables all other modes.
	PLIB_TMR_Mode32BitEnable	Enables 32-bit operation on the Timer module combination.
	PLIB_TMR_Period16BitGet	Gets the 16-bit period value.
	PLIB_TMR_Period16BitSet	Sets the 16-bit period value.
	PLIB_TMR_Period32BitGet	Gets the 32-bit period value.
	PLIB_TMR_Period32BitSet	Sets the 32-bit period value.
	PLIB_TMR_PrescaleDivisorGet	Gets the prescaler divisor information.
	PLIB_TMR_PrescaleGet	Gets the prescaler divisor value.
	PLIB_TMR_PrescaleSelect	Selects the clock prescaler.
	PLIB_TMR_Start	Starts/enables the indexed timer.
	PLIB_TMR_Stop	Stops/disables the indexed timer.
	PLIB_TMR_StopInIdleDisable	Continue module operation when the device enters Idle mode.
	PLIB_TMR_StopInIdleEnable	Discontinues module operation when the device enters Idle mode.

Description

Timer Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Timer Peripheral Library.

File Name

plib_tmr.h

Company

Microchip Technology Inc.

help_plib_tmr.h

Enumerations

	Name	Description
	TMR_CLOCK_SOURCE	Data type defining the different clock sources.
	TMR_MODULE_ID	Identifies the supported Timer modules.
	TMR_PRESCALE	Defines the list of possible prescaler values.

Description

This is file help_plib_tmr.h.

USART Peripheral Library

This section describes the USART Peripheral Library.

Introduction

This library provides a low-level abstraction of the USART Peripheral Library on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) module is the key component of the serial communications sub-system of a computer. The USART takes the data and transmits the individual bits in a sequential fashion. While receiving, the USART reassembles the bits into complete bytes. The data can be **8-bit** or **9-bit**.

There are two primary forms of serial transmission: Asynchronous and Synchronous.

Asynchronous

Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, the sender and receiver must agree on timing parameters (i.e., baud rate) in advance and special bits are added to each word, which are used to synchronize the sending and receiving units.

When a word is given to the USART for asynchronous transmissions, a bit called the "Start" bit is added to the beginning of each word that is to be transmitted. The Start bit is used to alert the receiver that a word of data is about to be sent. After the Start bit, the individual bits of the word of data are sent, with the Least Significant bit (LSb) being sent first. When the entire data word has been sent, the transmitter may add a Parity bit that the transmitter generates. The Parity bit may be used by the receiver to perform simple error checking. Then, at least one Stop bit is sent by the transmitter.

The Start bit is always a space and the Stop bits are always marks. Each transmitted bit persists for a period of 1/(Baud Rate). An on-chip dedicated Baud Rate Generator (BRG) is used to derive standard baud rate frequencies.

When the receiver has received all of the bits in the data word, it may check for the Parity bits (both sender and receiver must agree on whether a Parity bit is to be used), and then the receiver looks for a Stop bit. If the Stop bit does not appear when it is supposed to, the USART module considers the entire word to be unintelligible and will report a framing error to the host processor when the data word is read.

Note: Please refer to the **"USART"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details.

Synchronous

Synchronous serial transmission requires that the sender and receiver share a clock with one another, or that the sender provide a strobe or other timing signal so that the receiver knows when to "read" the next bit of the data. This mode is not available on all microcontrollers.

It is typically used in systems with a single master and one or more slaves. The master device contains the necessary circuitry for baud rate generation and supplies the clock for all devices in the system.

There are two signal lines a bidirectional data line and a clock line. Since the data line is bidirectional, synchronous operation is only half-duplex.

Start and Stop bits are not used in synchronous transmissions.

Using the Library

This topic describes the basic architecture of the USART Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_usart.h`

The interface to the USART Peripheral library is defined in the `plib_usart.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the USART Peripheral library must include `peripheral.h`.

Library File:

The USART Peripheral library archive (`.a`) file is installed with MPLAB Harmony.

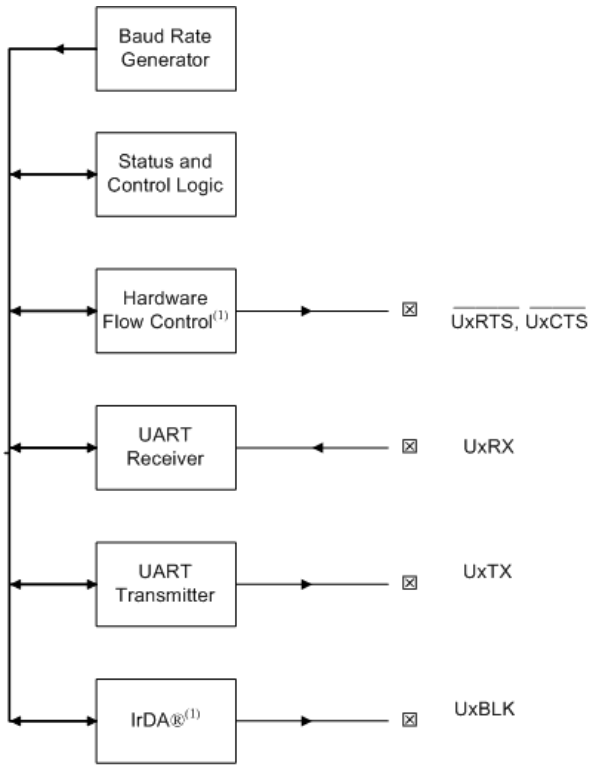
Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Hardware Abstraction Model

This section describes the hardware abstraction model.

Description

Hardware Abstraction Model



Note 1: This feature is not available on all devices. Refer to the **"USART"** chapter in the specific device data sheet to determine availability.

Hardware Abstraction Model

The USART Peripheral Library provides interface routines to interact with the blocks shown in the diagram.

The **Baud Rate Controller** controls the timing in Asynchronous mode, the desired baud rate can be programmed in the baud rate controller or, the baud rate controller can be configured to determine the baud rate of the transmitting device automatically.

The **Transmitter** and **Receiver** are capable of handling 8-bit or 9-bit data, which can be programmed in the control logic. The 9th bit is used to transfer the address or the parity. Enable 9-bit mode if the hardware is to be used for address or parity detection.

The **Status and Control logic**, provide the ability to control different ways the transmitter, receiver, and the baud rate controller can function. It also can provides status about the transmitter, receiver, or the baud rate controller.

The **Hardware Flow Control** uses CTS and the RTS lines to perform handshaking. Flow control is the ability to stop, and then restart the flow without any loss of bytes. Flow control is needed for modems and other hardware to allow a jump in instantaneous flow rates.

The USART module provides two types of infrared USART support: one is the IrDA clock output to support the external IrDA encoder and decoder, and the other is the full implementation of the IrDA encoder and decoder.

Note: These features are not available on all devices. Please refer to the specific device data sheet to determine availability.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the USART module.

Library Interface Section	Description
General Functions	Provides setup and configuration interface routines for • IrDA • Hardware Flow Control • Operation in power saving modes. • and other general setup
Baud Rate Generator Functions	Provides setup and configuration interface routines together with status routines for Baud Rate Generator (BRG).
Transmitter Functions	Provides setup, data transfer, error and the status interface routines for the transmitter.
Receiver Functions	Provides setup, data transfer, error and the status interface routines for the receiver.
Feature Existence Functions	Provides interface routines to determine existence of features.

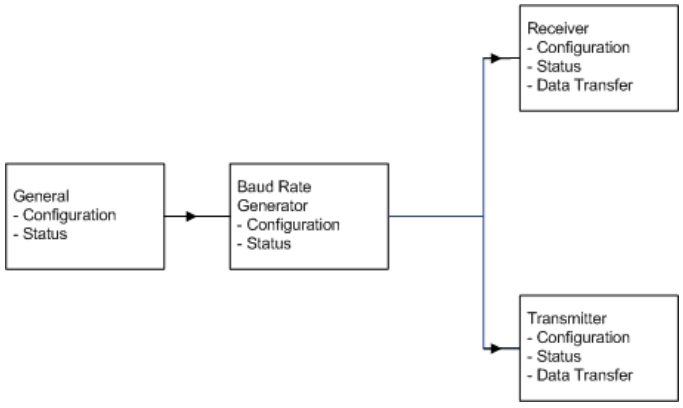
How the Library Works

Provides information on how the library works.

Description

This section provides information on using the peripheral library with the USART module. The usage model for different scenarios is also described.

Usage Model



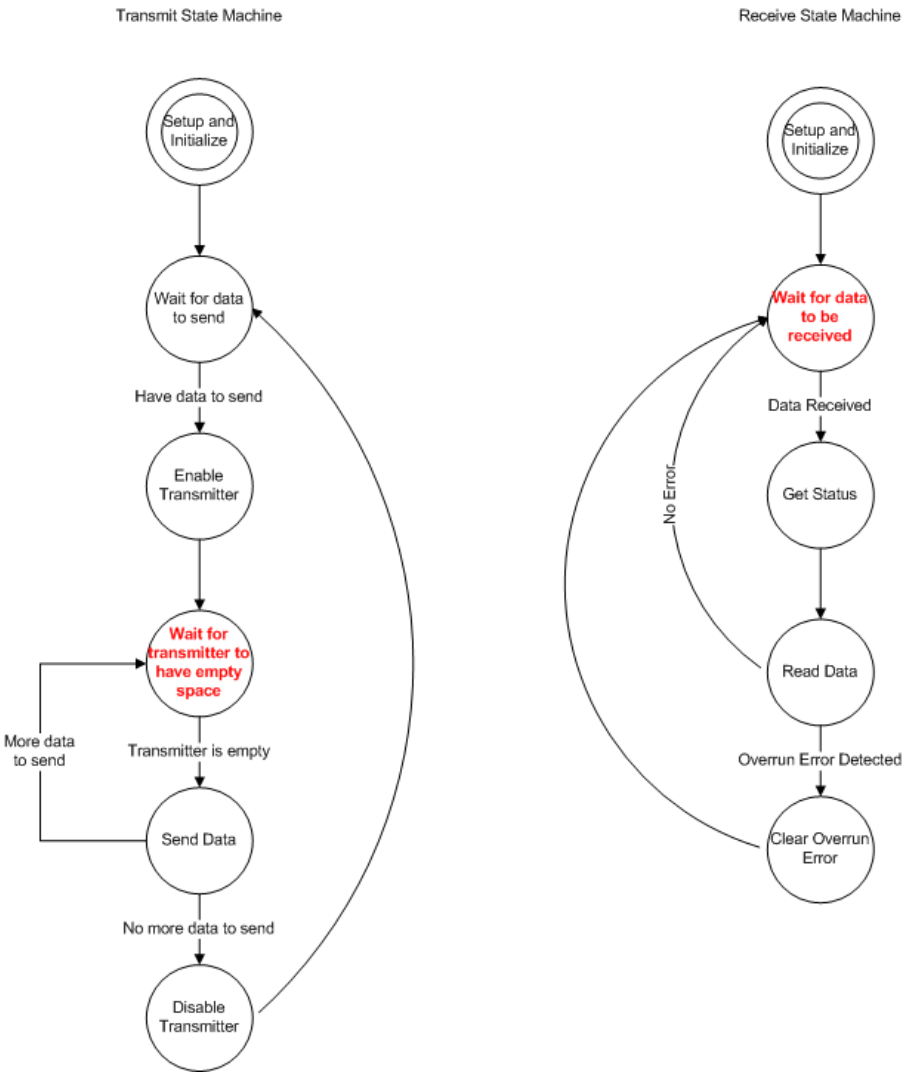
State Machine

This section describes the transmission and reception state machine.

Description

The following state machine is for the transmission and reception during normal operation. This state machine provides a general idea of the usage model of the USART Peripheral Library. Refer to the usage model for more detailed steps for the scenario that is being used.

Transmit and Receive State machine



Notes:

1. The wait states shown in red in the diagram can either be achieved using polled functions or using interrupts.
2. Refer to the specific USART mode in use for the setup and initialization steps.
3. Refer to the following table for the functions that can be used at each state.

State	Associated Function
Setup and initialization	Refer to the specific USART mode for detailed setup instructions.
Wait for data to send	Once the USART module has been appropriately set up and initialized, the state machine waits for the application to issue one of the send data functions.
Wait for data to be received	When the application is using Polled mode, the application can use the function PLIB_USART_ReceiverDataIsAvailable. When the application is using Interrupt mode, the application should wait for a receive interrupt.
Enable transmitter	Use the function PLIB_USART_TransmitterEnable to enable the transmitter
Wait for transmitter to have empty space	When the application is using polled mode, the application can use the function PLIB_USART_TransmitterBufferIsFull. When the application is using Interrupt mode, the application should wait for a transmit interrupt.
Send data	Use the function PLIB_USART_TransmitterByteSend to send data. If an address needs to be transmitted or if a byte needs to be transmitted with a Parity bit use the function PLIB_USART_Transmitter9BitsSend.
Disable transmitter	Use the function PLIB_USART_TransmitterDisable to disable the transmitter.
Get status	To get the status of the hardware module, use the set of functions, PLIB_USART_ReceiverFramingErrorHasOccurred, PLIB_USART_ReceiverParityErrorHasOccurred, and PLIB_USART_ReceiverOverrunHasOccurred.

Read data	Use the function PLIB_USART_ReceiverByteReceive to read data.
Clear overrun error	Clear the overrun error using the function PLIB_USART_ReceiverOverrunErrorClear.

Asynchronous USART Mode

This section provides processes for setting up asynchronous transmission and reception.

Description

Asynchronous Transmission

Setup

1. Set the desired baud rate using either PLIB_USART_BaudRateHighSet or PLIB_USART_BaudRateSet.
2. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
3. *If available on the device,* enable the asynchronous serial port using PLIB_USART_SyncModeSelect.
4. Set the number of data bits (8 or 9) and other line control modes using PLIB_USART_LineControlModeSelect.
5. *Optional (if available on the device):* If desired, enable the transmit interrupt.

- Ensure that the interrupts are enabled for the system and the peripherals
- Select the interrupt priority
- Set up the transmit FIFO interrupt mode using PLIB_USART_TransmitterInterruptModeSelect. Refer to USART_TRANSMIT_INTR_MODE for a list of possible values.

6. Enable the USART module using the function PLIB_USART_Enable

Example: Asynchronous Transmission Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently
//being used by the system.
PLIB_USART_BaudRateSet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY, baudRate);

// Enable the asynchronous serial port.
// enable asynchronous mode if the part supports it
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);

/* Select 8 data bits, No parity and one stop bit */
PLIB_USART_LineControlModeSelect(MY_USART_INSTANCE, USART_8N1);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.
```

Enable Transmission

1. Enable the transmission using PLIB_USART_TransmitterEnable.
2. Ensure that the transmit buffer is not full, before attempting to write into the transmit buffer. Either poll using PLIB_USART_TransmitterBufferIsFull or wait for the transmit interrupt.
3. Start the transmission using PLIB_USART_TransmitterByteSend (if transmitting 8-bit data) or PLIB_USART_Transmitter9BitsSend (if transmitting an address or if transmitting a byte with parity).
4. After the transmission is complete, disable the transmitter.

Example: Asynchronous Transmission

```
// Enable the transmission
PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);

// TODO - Either wait for Transmit buffer to be not full or wait for
//an transmit interrupt.

// Transmit the byte when Transmit buffer is empty.
// where, my_byte is the 8 bit data to be transmitted
PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, my_byte);
```

Asynchronous Transmission of Break Characters

A break characters consists of a **Start Bit** followed by twelve bits of "0" and a **Stop Bit.**

Setup and Transmission

1. Configure the desired mode, using the previous steps described.
2. Enable the transmitter using PLIB_USART_TransmitterEnable.
3. Transmit the break character using PLIB_USART_TransmitterBreakSend function.
4. Check if the break transmission is complete using PLIB_USART_TransmitterBreakSendIsComplete.
5. Transmission of the next bytes can start.

Example: USART Break and Sync Transmission

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently
//being used by the system.
PLIB_USART_BaudRateSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the asynchronous serial port.
// enable asynchronous mode if the part supports it
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);

/* Select 8 data bits, No parity and one stop bit */
PLIB_USART_LineControlModeSelect(MY_USART_INSTANCE, USART_8N1);

// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Enable the transmission
PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);

// Start the transmission
// Transmit the break character
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);

// Transmit the byte
while(PLIB_USART_TransmitterBreakSendIsComplete(MY_USART_INSTANCE));

PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, 0x55);
```

Refer to the following steps for setting up asynchronous receptions and how to receive data.

Asynchronous Reception

Setup

1. Set the desired baud rate using PLIB_USART_BaudRateHighSet or PLIB_USART_BaudRateSet.
2. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
3. *If available on the device*, enable the asynchronous serial port using PLIB_USART_SyncModeSelect.
4. Set the number of data bits (8 or 9) and other line control modes using PLIB_USART_LineControlModeSelect.
5. *Optional (if available on the device):* If desired, enable the receive interrupt.

- Ensure that the interrupts are enabled for the system and the peripherals
- Select the interrupt priority
- Set up the receive FIFO interrupt mode using PLIB_USART_ReceiverInterruptModeSelect. Refer to USART_RECEIVE_INTR_MODE for a list of possible values.

6. *Optional:* If inverted receive polarity is desired, use PLIB_USART_ReceiverIdleStateLowEnable.
7. *If available on the device*, enable the reception using PLIB_USART_ReceiverEnable.
8. Enable the USART module using PLIB_USART_Enable.

Example: Asynchronous Reception Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
```

```
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently being
// used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous slave serial port.
// enable asynchronous mode if the part supports it
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Start the reception
// Enable the continuous reception
PLIB_USART_ReceiverEnable(MY_USART_INSTANCE);
```

Reception

1. Either wait for an interrupt, which will be generated when the reception is completed, or poll using PLIB_USART_ReceiverDataIsAvailable.
2. Read the status of the device using PLIB_USART_ReceiverOverrunHasOccurred, PLIB_USART_ReceiverParityErrorHasOccurred, and PLIB_USART_ReceiverFramingErrorHasOccurred.
3. Read the data using PLIB_USART_ReceiverByteReceive.
4. If an overrun error occurred, clear the error using PLIB_USART_ReceiverOverrunErrorClear.

Example: Asynchronous Reception

```
// TODO - Either wait for Receive buffer to have data or wait for
// a receive interrupt.
// read the data
my_byte = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
// where, my_byte is the 8 bit data received

// If overrun error clear the error.
if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

Asynchronous Reception with Address Detection Mode

A typical multi-processor communication protocol will differentiate between data bytes and address/control bytes. A common method is to use a 9th data bit to identify whether a data byte is address or data information. If the 9th bit is set, the data is processed as address or control information. If the 9th bit is cleared, the received data word is processed as data associated with the previous address/control byte.

The protocol operates as follows:

- The master device transmits a data word with 9th bit set. The data word contains the address of a slave device.
- All slave devices in the communication chain receive the address word and check the slave address.
- The slave device that was addressed will receive and process subsequent data bytes sent by the master device. All other slave devices will discard subsequent data bytes until a new address word(9th bit set) is received.

This mode will typically be used in RS-485 systems.

Setup

1. Set the desired baud rate using PLIB_USART_BaudRateHighSet or PLIB_USART_BaudRateSet.
2. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
3. *If available on the device*, enable the asynchronous serial port using PLIB_USART_SyncModeSelect.
4. Set the number of data bits (8 or 9) and other line control modes using PLIB_USART_LineControlModeSelect.
5. *Optional (if available on the device):* If desired, enable the receive interrupt.

- Ensure that the interrupts are enabled for the system and the peripherals. Also select the interrupt priority.
- Set up the receive FIFO interrupt mode by using the function PLIB_USART_ReceiverInterruptModeSelect. Refer to USART_RECEIVE_INTR_MODE for a list of possible values.
- For Address Detect mode, configure PLIB_USART_ReceiverInterruptModeSelect with the receive FIFO interrupt mode as USART_RECEIVE_FIFO_ONE_CHAR.

6. *Optional:* If inverted receive polarity is desired, use PLIB_USART_ReceiverIdleStateLowEnable.
7. *If available on the device*, enable the reception using PLIB_USART_ReceiverEnable.
8. Enable address detection using PLIB_USART_ReceiverAddressDetectEnable.
9. Enable the USART module using PLIB_USART_Enable.

Note: This feature is not available on all devices. On devices where automatic address detection is available, configure automatic address detection using PLIB_USART_ReceiverAddressAutoDetectEnable. Skip steps 2-6 in the following Reception section during the reception phase.

Example: Asynchronous Reception Setup (with address detect enable)

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently
// being used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous slave serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Set the reception to 9 bits to enable address byte.
PLIB_USART_LineControlModeSelect(MY_USART_INSTANCE, USART_9N1);

// Enable address detection.
PLIB_USART_ReceiverAddressDetectEnable(MY_USART_INSTANCE);

// Start the reception
// Enable the continuous reception
PLIB_USART_ReceiverEnable(MY_USART_INSTANCE);
```

Reception

1. Either wait for an interrupt, which will be generated when the reception is completed or poll using PLIB_USART_ReceiverDataIsAvailable.
2. Read the address using PLIB_USART_ReceiverByteReceive.
3. The application determines if this is the device address. If the device is addressed, disable address detect using PLIB_USART_ReceiverAddressDetectDisable to allow subsequent data bytes to be received; otherwise, discard the received word.
4. Read the status of the device using PLIB_USART_ReceiverOverrunHasOccurred, PLIB_USART_ReceiverParityErrorHasOccurred, and PLIB_USART_ReceiverFramingErrorHasOccurred.
5. If an overrun error occurred, clear the error using PLIB_USART_ReceiverOverrunErrorClear.
6. If the device has been addressed, disable the address detection using PLIB_USART_ReceiverAddressDetectDisable, which will allow all received data into the receive buffer and generate interrupts if desired.
7. *Optional (if available on the device):* If a long packet is expected, the receive interrupt mode could be changed to buffer more than one data word between interrupts. Use PLIB_USART_ReceiverInterruptModeSelect with USART_RECEIVE_FIFO_HALF_FULL or USART_RECEIVE_FIFO_3B4FULL.
8. When the last data byte has been received, enable the address detect using PLIB_USART_ReceiverAddressDetectEnable, so that only address bytes will be received.
9. *Optional (if available on the device):* Ensure that the receive interrupt is configured to be triggered after each received word using PLIB_USART_ReceiverInterruptModeSelect with USART_RECEIVE_FIFO_ONE_CHAR.

Example: Asynchronous Reception (with Address Detect)

```
// Either wait for Receive buffer to have data or wait for a receive interrupt.
int8_t my_address;

// Receive the address
if(PLIB_USART_ReceiverDataIsAvailable(MY_USART_INSTANCE))
{
    my_address = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
    // where, my_address is the address received
}

// If overrun error clear the error.
if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}

if(my_address == MY_DEVICE_ADDRESS)
{
    // The device has been addressed.
    PLIB_USART_ReceiverAddressDetectDisable(MY_USART_INSTANCE);
}
```

```
// TODO - Receive the data bytes for the packet. Either wait for RX buffer
//to have data or wait for a receive interrupt.

// When all the bytes for the packet are received.

// Enable address detection mode
PLIB_USART_ReceiverAddressDetectEnable(MY_USART_INSTANCE);
```

Synchronous Master Mode

This section provides processes for setting up synchronous master transmission and reception.

Note: The features described in this section are not available on all devices. Please refer to the **"USART"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details.

Description

Synchronous Master Transmission

Setup

1. Set the desired baud rate using PLIB_USART_BaudRateSet or PLIB_USART_BaudRateHighSet.
2. Enable the synchronous master serial port using PLIB_USART_SyncModeSelect, PLIB_USART_SyncClockSourceSelect, and PLIB_USART_Enable.
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. Disable receive using PLIB_USART_ReceiverDisable.
5. *Optional:* If desired set the number of data bits to 9 (to send address byte or to enable parity check) using PLIB_USART_LineControlModeSelect.
6. *Optional:* If desired, enable the transmit interrupt, ensuring that the interrupts are enabled for the system and the peripherals.

Example: Synchronous Master Transmission Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently being used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous master serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_ASYNC_MODE);
// enable master clock
PLIB_USART_SyncClockSourceSelect(MY_USART_INSTANCE, USART_SYNC_CLOCK_SOURCE_MASTER);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO Call the appropriate function.
```

Transmission

1. Enable transmission using PLIB_USART_TransmitterEnable.
2. Ensure that the transmit buffer is not full before attempting to write onto the transmit buffer. Either poll using PLIB_USART_TransmitterBufferIsFull or wait for the transmit interrupt.

Note: If using interrupts, make sure that interrupts are also enabled for the system and the peripherals.

3. Start the transmission using PLIB_USART_TransmitterByteSend (if transmitting 8 - bit data) or PLIB_USART_Transmitter9BitsSend (if transmitting address or if transmitting a byte with parity).
4. After the transmission is complete, disable the transmitter.

Example: Synchronous Master Transmission

```
// Enable the transmission
PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);

// TODO Either wait for Transmit buffer to be not full or wait for an transmit interrupt.

// Transmit the byte when Transmit buffer is empty.
// where, my_byte is the 8 bit data to be transmitted
PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, my_byte);
```

Synchronous Master Reception

Setup

1. Set the desired baud rate using PLIB_USART_BaudRateSet or PLIB_USART_BaudRateHighSet.
2. Enable the synchronous master serial port using PLIB_USART_SyncModeSelect, PLIB_USART_SyncClockSourceSelect, and PLIB_USART_Enable.
3. Set the appropriate IO direction and corresponding to RX and TX I/O pins.
4. *Optional:* If desired, enable the receive interrupt, ensuring that the interrupts are enabled for the system and the peripherals.
5. Start the reception using PLIB_USART_ReceiverDisable.

Example: Synchronous Master Reception Setup

```
uint32_t baudRate = 9600;
// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently being used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

PLIB_USART_HandshakeModeSelect(MY_USART_INSTANCE,
                                    USART_HANDSHAKE_MODE_SIMPLEX);
// Enable the synchronous master serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_SYNC_MODE);
// enable master clock
PLIB_USART_SyncClockSourceSelect(MY_USART_INSTANCE, USART_SYNC_CLOCK_SOURCE_MASTER);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Enable receive mode
PLIB_USART_ReceiverEnable(MY_USART_INSTANCE);
```

Reception

1. Either wait for an interrupt, which will be generated when the reception is completed, or poll using PLIB_USART_ReceiverDataIsAvailable.

Note: If using interrupts, make sure that interrupts are also enabled for the system and the peripherals.

2. Read the status of the device using PLIB_USART_ReceiverOverrunHasOccurred, PLIB_USART_ReceiverParityErrorHasOccurred and PLIB_USART_ReceiverFramingErrorHasOccurred.
3. Read the data using PLIB_USART_ReceiverByteReceive.
4. If an overrun error occurred, clear the error using the function PLIB_USART_ReceiverOverrunErrorClear.

Example: Synchronous Master Reception

```
// Read the byte
// where, my_byte is the 8 bit data to be transmitted
my_byte = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);

if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

Synchronous Slave Mode

This section provides processes for setting up synchronous slave transmission and reception.

Note: This feature is not available on all devices. Please refer to the **"USART"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details.

Description

Synchronous Slave Transmission

Setup

1. Set the desired baud rate using either PLIB_USART_BaudRateHighSet or PLIB_USART_BaudRateSet.
2. Enable the synchronous slave serial port using PLIB_USART_SyncModeSelect, PLIB_USART_SyncClockSourceSelect, and PLIB_USART_Enable.

3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. Disable receive mode using PLIB_USART_ReceiverDisable.
5. *Optional:* If desired, enable the transmit interrupt,ensuring that the interrupts are enabled for the system and the peripherals.

Example: Synchronous Slave Transmission Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware
//peripheral MY_CLOCK_FREQUENCY - is hardware clock frequency which is
//currently being used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous slave serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_SYNC_MODE);
// enable slave clock
PLIB_USART_SyncClockSourceSelect(MY_USART_INSTANCE, USART_SYNC_CLOCK_SOURCE_SLAVE);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Disable receive mode
PLIB_USART_ReceiverDisable(MY_USART_INSTANCE);

// TODO - If using interrupts Enable the transmit interrupt
// TODO - Call the appropriate function for the transmit enable interrupt,
// TODO - Also, make sure that the interrupts are enabled for system and the peripherals.
```

Transmission

1. Enable transmission using PLIB_USART_TransmitterEnable.
2. Ensure that the transmit buffer is not full before attempting to write into the transmit buffer. Either poll using PLIB_USART_TransmitterBufferIsFull or wait for the transmit interrupt.

Note: If using interrupts, ensure that interrupts are also enabled for the system and the peripherals.

3. Start the transmission using PLIB_USART_TransmitterByteSend (if transmitting 8-bit data) or PLIB_USART_Transmitter9BitsSend (if transmitting address if transmitting a byte with parity)
4. After the transmission is complete, disable the transmitter.

Example: Synchronous Slave Transmission

```
// Enable the transmission
PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);

// TODO - Either wait for Transmit buffer to be not full or wait
//for an transmit interrupt.

// Transmit the byte when Transmit buffer is empty.
PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, my_word);
// where, my_word is the 9 bit data to be transmitted
```

Synchronous Slave Reception

Setup

1. Set the desired baud rate using PLIB_USART_BaudRateHighSet or PLIB_USART_BaudRateSet.
2. Enable the synchronous master serial port using PLIB_USART_SyncModeSelect, PLIB_USART_SyncClockSourceSelect, and PLIB_USART_Enable.
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. *Optional:* If desired, enable the receive interrupt, ensuring that the interrupts are enabled for the system and the peripherals.
5. Start the reception using PLIB_USART_ReceiverEnable.

Example: Synchronous Slave Reception Setup

```
// Set the desired baud rate
uint32_t baudRate = 9600;

// where, MY_USART_INSTANCE - is a specific instance of the hardware peripheral.
// where, MY_CLOCK_FREQUENCY - is hardware clock frequency which is currently
```

```
//being used by the system.
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, baudRate, MY_CLOCK_FREQUENCY);

// Enable the synchronous slave serial port.
// enable sync mode
PLIB_USART_SyncModeSelect(MY_USART_INSTANCE, USART_SYNC_MODE);
// enable master clock
PLIB_USART_SyncClockSourceSelect(MY_USART_INSTANCE, USART_SYNC_CLOCK_SOURCE_SLAVE);
// enable USART
PLIB_USART_Enable(MY_USART_INSTANCE);

// Set the appropriate IO direction for USART pins
// TODO - Call the appropriate function.

// Start the reception
// Enable the continuous reception
PLIB_USART_ReceiverDisable(MY_USART_INSTANCE);
```

Reception

1. Either wait for an interrupt, which will be generated when the reception is completed, or poll using PLIB_USART_ReceiverDataIsAvailable.

Note: If using interrupts, ensure that interrupts are also enabled for the system and the peripherals.

2. Read the status of the device using PLIB_USART_ReceiverOverrunHasOccurred, PLIB_USART_ReceiverParityErrorHasOccurred, and PLIB_USART_ReceiverFramingErrorHasOccurred.
3. Read the data using PLIB_USART_ReceiverByteReceive.
4. If an overrun error occurred, clear the error using PLIB_USART_ReceiverOverrunErrorClear.

Example: Synchronous Slave Reception

```
// Transmit the data
my_byte = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
// where, my_byte is the 8 bit data received

// If overrun error clear the error.
if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
 PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

Other Features

This section describes additional features of the USART Peripheral Library.

Note: The features described in this section are not available on all devices. Please refer to the **"USART"** chapter in the specific device data sheet or the family reference manual section specified in that chapter for details.

Description

IrDA Support

Support for IrDA is available in some microcontrollers.

1. Enable IrDA support using PLIB_USART_IrDAEnable. IrDA support can be disabled using PLIB_USART_IrDADisable.
2. *Optional:* If using the external IrDA encoder and decoder, use the function PLIB_USART_OperationModeSelect with USART_ENABLE_TX_RX_BCLK_USED to output the 16x baud clock.
3. *Optional:* IrDA Transmit Polarity can be inverted using PLIB_USART_TransmitterIdleIsLowEnable.
4. *Optional:* IrDA Receive Polarity can be inverted using PLIB_USART_ReceiverIdleStateLowEnable.
5. Enable the USART using PLIB_USART_Enable.

Loopback Support

This is the mode in which the transmission is internally connected to reception.

1. Set the desired baud rate using PLIB_USART_BaudRateSet or PLIB_USART_BaudRateHighSet.
2. Set up the line control parameters (data bits, stop bits, parity, flow control) using PLIB_USART_LineControlModeSelect. Refer to USART_LINECONTROL_MODE for a list of possible values.
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. *Optional:* If desired, enable the transmit interrupt, also make sure that the interrupts are enabled for the system and the peripherals.
5. *Optional:* Also select the interrupt priority.

6. *Optional:* Also, set up the transmit FIFO interrupt mode using PLIB_USART_TransmitterInterruptModeSelect. Refer to USART_TRANSMIT_INTR_MODE for a list of possible values.
7. Enable the USART module using PLIB_USART_Enable and enable the transmission using PLIB_USART_TransmitterEnable.
8. Enable loopback using PLIB_USART_LoopbackEnable. Loopback can be disabled at any time using PLIB_USART_LoopbackDisable
9. Continue with transmission and/or reception using the same steps as for asynchronous transmission/reception.

Auto Baud Support

1. Enable the auto baud detection using the function PLIB_USART_BaudRateAutoDetectEnable, which requires a reception of the Sync character (0x55).
2. *Optional:* Poll PLIB_USART_BaudRateAutoDetectIsComplete to check if the baud rate is detected.
3. *Optional:* Call PLIB_USART_BaudRateGet to get the baud rate that was detected.
4. Clear the receive interrupt.
5. On some microcontrollers, reading the data using PLIB_USART_ReceiverByteReceive may be required.

Flow Control Support

The USART can use Simplex or Flow Control modes:

- Use PLIB_USART_HandshakeModeSelect to enable flow control (USART_HANDSHAKE_MODE_FLOW_CONTROL) or use simplex mode (USART_HANDSHAKE_MODE_SIMPLEX) selected from USART_HANDSHAKE_MODE.
- To configure the USART line control features, use PLIB_USART_LineControlModeSelect selected from USART_LINECONTROL_MODE.
- Additionally, the USART pins may need to be configured using PLIB_USART_OperationModeSelect and the appropriate values from USART_OPERATION_MODE.

Operation During Sleep Mode

When the device enters Sleep mode, all clock sources supplied to the USART modules are shut down. If the device enters Sleep mode in the middle of a USART transmission or reception operation, the operation is aborted and the USART pins are driven to default state.

A Start bit, when detected, can wake up the device.

- The application needs to enable the wake-up on start bit using PLIB_USART_WakeOnStartEnable just before entering Sleep mode, to wake-up the device
- For the device to wake-up from sleep, reception of the Sync character or Wake-up Signal character is required
- *Optional:* If using interrupts, a receive interrupt is generated

Operation During Idle Mode

When the device enters Idle mode, the system clock sources remain functional and the CPU stops code execution.

- USART operation during Idle mode can be stopped using PLIB_USART_StopInIdleEnable. When the device enters Idle mode, the USART module operation is similar to that as in Sleep mode(Refer to Operation in Sleep Mode).
- By default, the USART module continues to operate in Idle mode and provide full functionality

USART Operation With DMA

The DMA module can be used to transfer data between the USART and the CPU, without CPU assistance.

1. Set the desired baud rate using the function PLIB_USART_BaudRateSet or PLIB_USART_BaudRateHighSet.
2. Set up the line control parameters (data bits, stop bits, parity, flow control) using the function PLIB_USART_LineControlModeSelect. Refer to USART_LINECONTROL_MODE for a list of possible values.
3. Set the appropriate I/O direction and corresponding RX and TX I/O pins.
4. Enable the USART module using PLIB_USART_Enable and enable the transmission using PLIB_USART_TransmitterEnable.
5. Enable the Transmit, Receive, and Error interrupts, ensuring that the interrupts are enabled for the system and the peripherals.
6. Select the interrupt priority.
7. Set up the Transmit FIFO Interrupt mode and Receive FIFO Interrupt mode using the PLIB_USART_TransmitterInterruptModeSelect and PLIB_USART_ReceiverInterruptModeSelect functions. To generate interrupts for every character received, use USART_RECEIVE_FIFO_ONE_CHAR from USART_RECEIVE_INTR_MODE.
8. Configure the DMA to be used with the USART receiver and transmitter.

Configuring the Library

The library is configured for the supported USART modules when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Functions

	Name	Description
	PLIB_USART_Disable	Disables the specific USART module
	PLIB_USART_Enable	Enables the specific USART module.
	PLIB_USART_HandshakeModeSelect	Sets the data flow configuration.
	PLIB_USART_IrDADisable	Disables the IrDA encoder and decoder.
	PLIB_USART_IrDAEnable	Enables the IrDA encoder and decoder.
	PLIB_USART_LineControlModeSelect	Sets the data flow configuration.
	PLIB_USART_LoopbackDisable	Disables Loopback mode.
	PLIB_USART_LoopbackEnable	Enables Loopback mode.
	PLIB_USART_OperationModeSelect	Configures the operation mode of the USART module.
	PLIB_USART_StopInIdleDisable	Disables the Stop in Idle mode (the USART module continues operation when the device is in Idle mode).
	PLIB_USART_StopInIdleEnable	Discontinues operation when the device enters Idle mode.
	PLIB_USART_WakeOnStartDisable	Disables the wake-up on start bit detection feature during Sleep mode.
	PLIB_USART_WakeOnStartEnable	Enables the wake-up on start bit detection feature during Sleep mode.
	PLIB_USART_WakeOnStartIsEnabled	Gets the state of the sync break event completion.
	PLIB_USART_ErrorsGet	Return the status of all errors in the specified USART module.
	PLIB_USART_InitializeModeGeneral	Enables or disables general features of the USART module.
	PLIB_USART_InitializeOperation	Configures the Receive and Transmit FIFO interrupt levels and the hardware lines to be used by the module.
	PLIB_USART_AddressGet	Gets the address for the Address Detect mode.
	PLIB_USART_AddressMaskGet	Gets the address mask for the Address Detect mode.
	PLIB_USART_AddressMaskSet	Sets the address mask for the Address Detect mode.
	PLIB_USART_AddressSet	Sets the address for the Address Detect mode.
	PLIB_USART_ModuleIsBusy	Returns the USART module's running status.
	PLIB_USART_RunInOverflowDisable	Disables the Run in overflow condition mode.
	PLIB_USART_RunInOverflowEnable	Enables the USART module to continue to operate when an overflow error condition has occurred.
	PLIB_USART_RunInOverflowIsEnabled	Gets the status of the Run in Overflow condition.
	PLIB_USART_RunInSleepModeDisable	Turns off the USART module's BRG clock during Sleep mode.
	PLIB_USART_RunInSleepModeEnable	Allows the USART module's BRG clock to run when the device enters Sleep mode.
	PLIB_USART_RunInSleepModeIsEnabled	Gets the status of Run in Sleep mode.

b) Baud Rate Generator Functions

	Name	Description
	PLIB_USART_BaudRateAutoDetectEnable	Enables baud rate measurement on the next character, which requires reception of the Sync character.
	PLIB_USART_BaudRateAutoDetectIsComplete	Gets the state of the automatic baud detection.
	PLIB_USART_BaudRateGet	Gets the baud rate current in use.
	PLIB_USART_BaudRateHighDisable	Disables the high baud rate selection.
	PLIB_USART_BaudRateHighEnable	Enables high baud rate selection.
	PLIB_USART_BaudRateHighSet	Sets the baud rate to the desired value.
	PLIB_USART_BaudRateSet	Sets the baud rate to the desired value.
	PLIB_USART_BaudSetAndEnable	Sets the baud rate to the desired value and enables the USART receiver, transmitter and the USART module.
	PLIB_USART_BRGClockSourceGet	Gets the BRG clock source of the USART module.
	PLIB_USART_BRGClockSourceSelect	Configures the BRG clock source of the USART module.

c) Transmitter Functions

	Name	Description
	PLIB_USART_Transmitter9BitsSend	Data to be transmitted in the byte mode with the 9th bit.
	PLIB_USART_TransmitterBreakSend	Transmits the break character.

	PLIB_USART_TransmitterBreakSendIsComplete	Returns the status of the break transmission
	PLIB_USART_TransmitterBufferIsFull	Gets the transmit buffer full status.
	PLIB_USART_TransmitterByteSend	Data to be transmitted in the Byte mode.
	PLIB_USART_TransmitterDisable	Disables the specific USART module transmitter.
	PLIB_USART_TransmitterEnable	Enables the specific USART module transmitter.
	PLIB_USART_TransmitterIdleIsLowDisable	Disables the Transmit Idle Low state.
	PLIB_USART_TransmitterIdleIsLowEnable	Enables the Transmit Idle Low state.
	PLIB_USART_TransmitterInterruptModeSelect	Sets the USART transmitter interrupt mode.
	PLIB_USART_TransmitterIsEmpty	Gets the transmit shift register empty status.
	PLIB_USART_TransmitterAddressGet	Returns the address of the USART TX register

d) Receiver Functions

	Name	Description
	PLIB_USART_ReceiverAddressAutoDetectDisable	Disables the automatic Address Detect mode.
	PLIB_USART_ReceiverAddressAutoDetectEnable	Setup the automatic Address Detect mode.
	PLIB_USART_ReceiverAddressDetectDisable	Enables the Address Detect mode.
	PLIB_USART_ReceiverAddressDetectEnable	Enables the Address Detect mode.
	PLIB_USART_ReceiverAddressIsReceived	Checks and return if the data received is an address.
	PLIB_USART_ReceiverByteReceive	Data to be received in the Byte mode.
	PLIB_USART_ReceiverDataIsAvailable	Identifies if the receive data is available for the specified USART module.
	PLIB_USART_ReceiverDisable	Disables the USART receiver.
	PLIB_USART_ReceiverEnable	Enables the USART receiver.
	PLIB_USART_ReceiverFramingErrorHasOccurred	Gets the framing error status.
	PLIB_USART_ReceiverIdleStateLowDisable	Disables receive polarity inversion.
	PLIB_USART_ReceiverIdleStateLowEnable	Enables receive polarity inversion.
	PLIB_USART_ReceiverInterruptModeSelect	Sets the USART receiver FIFO level.
	PLIB_USART_ReceiverIsIdle	Identifies if the receiver is idle.
	PLIB_USART_ReceiverOverrunErrorClear	Clears a USART vverrun error.
	PLIB_USART_ReceiverOverrunHasOccurred	Identifies if there was a receiver overrun error.
	PLIB_USART_ReceiverParityErrorHasOccurred	Gets the parity error status.
	PLIB_USART_ReceiverAddressGet	Returns the address of the USART RX register
	PLIB_USART_Receiver9BitsReceive	Data to be received in the byte mode with the 9th bit.

e) Feature Existence Functions

	Name	Description
	PLIB_USART_ExistsBaudRate	Identifies whether the BaudRate feature exists on the USART module.
	PLIB_USART_ExistsBaudRateAutoDetect	Identifies whether the BaudRateAutoDetect feature exists on the USART module.
	PLIB_USART_ExistsBaudRateHigh	Identifies whether the BaudRateHigh feature exists on the USART module.
	PLIB_USART_ExistsEnable	Identifies whether the EnableControl feature exists on the USART module.
	PLIB_USART_ExistsHandshakeMode	Identifies whether the HandShakeMode feature exists on the USART module.
	PLIB_USART_ExistsIrDA	Identifies whether the IrDAControl feature exists on the USART module.
	PLIB_USART_ExistsLineControlMode	Identifies whether the LineControlMode feature exists on the USART module.
	PLIB_USART_ExistsLoopback	Identifies whether the Loopback feature exists on the USART module.
	PLIB_USART_ExistsOperationMode	Identifies whether the OperationMode feature exists on the USART module.
	PLIB_USART_ExistsReceiver	Identifies whether the Receiver feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddressAutoDetect	Identifies whether the ReceiverAddressAutoDetect feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddressDetect	Identifies whether the ReceiverAddressDetect feature exists on the USART module.
	PLIB_USART_ExistsReceiverDataAvailableStatus	Identifies whether the ReceiverDataAvailable feature exists on the USART module
	PLIB_USART_ExistsReceiverEnable	Identifies whether the ReceiverEnableControl feature exists on the USART module.
	PLIB_USART_ExistsReceiverFramingErrorStatus	Identifies whether the ReceiverFramingError feature exists on the USART module.

	PLIB_USART_ExistsReceiverIdleStateLowEnable	Identifies whether the ReceiverPolarityInvert feature exists on the USART module.
	PLIB_USART_ExistsReceiverIdleStatus	Identifies whether the ReceiverIdle feature exists on the USART module.
	PLIB_USART_ExistsReceiverInterruptMode	Identifies whether the ReceiverInterruptMode feature exists on the USART module.
	PLIB_USART_ExistsReceiverOverrunStatus	Identifies whether the ReceiverOverrunError feature exists on the USART module.
	PLIB_USART_ExistsReceiverParityErrorStatus	Identifies whether the ReceiverParityError feature exists on the USART module.
	PLIB_USART_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the USART module.
	PLIB_USART_ExistsTransmitter	Identifies whether the Transmitter feature exists on the USART module.
	PLIB_USART_ExistsTransmitter9BitsSend	Identifies whether the Transmitter9Bits feature exists on the USART module.
	PLIB_USART_ExistsTransmitterBreak	Identifies whether the TransmitterBreak feature exists on the USART module.
	PLIB_USART_ExistsTransmitterBufferFullStatus	Identifies whether the TransmitterBufferFull feature exists on the USART module.
	PLIB_USART_ExistsTransmitterEmptyStatus	Identifies whether the TransmitterEmpty feature exists on the USART module.
	PLIB_USART_ExistsTransmitterEnable	Identifies whether the TransmitterEnableControl feature exists on the USART module
	PLIB_USART_ExistsTransmitterIdleIsLow	Identifies whether the TransmitterIdleIsLow feature exists on the USART module.
	PLIB_USART_ExistsTransmitterInterruptMode	Identifies whether the TransmitterInterruptMode feature exists on the USART module.
	PLIB_USART_ExistsWakeOnStart	Identifies whether the WakeOnStart feature exists on the USART module.
	PLIB_USART_ExistsReceiver9Bits	Identifies whether the 9 Bits Receiver feature exists on the USART module.
	PLIB_USART_ExistsBRGClockSourceSelect	Identifies whether the BRG Clock source select feature exists on the USART module.
	PLIB_USART_ExistsModuleBusyStatus	Identifies whether the module running status feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddress	Identifies whether the Receiver Address feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddressMask	Identifies whether the Receiver Address Mask feature exists on the USART module.
	PLIB_USART_ExistsRunInOverflow	Identifies whether the Run in overflow condition feature exists on the USART module.
	PLIB_USART_ExistsRunInSleepMode	Identifies whether the Run in Sleep mode feature exists on the USART module.

f) Data Types and Constants

	Name	Description
	USART_HANDSHAKE_MODE	Lists the USART handshake modes.
	USART_LINECONTROL_MODE	Data type defining the different configurations by which the USART data flow can be configured.
	USART_MODULE_ID	Enumeration: USART_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on Microchip microcontrollers. Refer to the data sheet to get the correct number of modules defined for desired microcontroller.
	_USART_MODULE_ID	
	USART_OPERATION_MODE	Data type defining the different configurations by which the USART can be enabled.
	USART_RECEIVE_INTR_MODE	Data type defining the different Receive FIFO levels by which the USART receive interrupt modes can be configured.
	USART_TRANSMIT_INTR_MODE	Data type defining the different transmit FIFO levels by which the USART transmit interrupt modes can be configured.

Description

This section describes the Application Programming Interface (API) functions of the USART Peripheral Library.

Refer to each section for a detailed description.

a) General Functions

PLIB_USART_Disable Function

Disables the specific USART module

File

plib_usart.h

C

```
void PLIB_USART_Disable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific USART module.

Remarks

The default state after any reset for a USART module is the disable state. If the USART is disabled, all of the related pins are in control of the general purpose I/O logic.

Disabling the USART module resets the buffers to empty states. Any data characters in the buffers are lost and the baud rate is reset. All error and status bits are also reset.

Disabling the USART while the USART is active, will abort all pending transmissions and receptions. Re-enabling the USART will restart the module in the same configuration.

When disabled, the USART power consumption is minimal. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_Disable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_Disable (USART_MODULE_ID index)

PLIB_USART_Enable Function

Enables the specific USART module.

File

plib_usart.h

C

```
void PLIB_USART_Enable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific USART module.

Remarks

By calling this function, the USART pins are controlled by the USART module. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_Enable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_Enable (USART_MODULE_ID index)

PLIB_USART_HandshakeModeSelect Function

Sets the data flow configuration.

File

plib_usart.h

C

```
void PLIB_USART_HandshakeModeSelect(USART_MODULE_ID index, USART_HANDSHAKE_MODE handshakeConfig);
```

Returns

None.

Description

This function sets the USART data flow configuration based on the mask provided and the specified baud rate.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsHandshakeMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_HandshakeModeSelect(MY_USART_INSTANCE,
                                    USART_HANDSHAKE_MODE_SIMPLEX);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	For possible data flow configurations, refer to USART_HANDSHAKE_MODE

Function

void PLIB_USART_HandshakeModeSelect(USART_MODULE_ID index,
USART_HANDSHAKE_MODE handshakeConfig)

PLIB_USART_IrDADisable Function

Disables the IrDA encoder and decoder.

File

plib_usart.h

C

```
void PLIB_USART_IrDADisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the IrDA encoder and decoder.

Remarks

By default, the IrDA Encoder and Decoder are disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsIrDA in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_IrDADisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_IrDADisable (USART_MODULE_ID index)

PLIB_USART_IrDAEnable Function

Enables the IrDA encoder and decoder.

File

plib_usart.h

C

```
void PLIB_USART_IrDAEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the IrDA encoder and decoder.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsIrDA in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_IrDAEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_IrDAEnable (USART_MODULE_ID index)

PLIB_USART_LineControlModeSelect Function

Sets the data flow configuration.

File

plib_usart.h

C

```
void PLIB_USART_LineControlModeSelect(USART_MODULE_ID index, USART_LINECONTROL_MODE dataFlowConfig);
```

Returns

None.

Description

This function sets the USART data flow configuration based on the mask provided and the specified baud rate.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsLineControlMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_LineControlModeSelect(MY_USART_INSTANCE,
                                 USART_8N1);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mode	For possible data flow configurations, refer to USART_LINECONTROL_MODE

Function

void PLIB_USART_LineControlModeSelect(USART_MODULE_ID index,
USART_LINECONTROL_MODE dataFlowConfig)

PLIB_USART_LoopbackDisable Function

Disables Loopback mode.

File

plib_usart.h

C

```
void PLIB_USART_LoopbackDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables Loopback mode.

Remarks

By default, Loopback mode is disabled. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsLoopback in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_LoopbackDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_LoopbackDisable (USART_MODULE_ID index)

PLIB_USART_LoopbackEnable Function

Enables Loopback mode.

File

plib_usart.h

C

```
void PLIB_USART_LoopbackEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables Loopback mode.

Remarks

By default, Loopback mode is disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsLoopback in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_LoopbackEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_LoopbackEnable (USART_MODULE_ID index)

PLIB_USART_OperationModeSelect Function

Configures the operation mode of the USART module.

File

plib_usart.h

C

```
void PLIB_USART_OperationModeSelect(USART_MODULE_ID index, USART_OPERATION_MODE operationmode);
```

Returns

None.

Description

This function configures the operation mode of the USART (i.e., controls the pins used by the USART module). Refer to USART_OPERATION_MODE for the possible settings.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsOperationMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_OperationModeSelect(MY_USART_INSTANCE, USART_ENABLE_TX_RX_BCLK_USED);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
operationmode	One of the possible values from USART_OPERATION_MODE

Function

void PLIB_USART_OperationModeSelect(USART_MODULE_ID index,
USART_OPERATION_MODE operationmode)

PLIB_USART_StopInIdleDisable Function

Disables the Stop in Idle mode (the USART module continues operation when the device is in Idle mode).

File

plib_usart.h

C

```
void PLIB_USART_StopInIdleDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the Stop in Idle mode. The USART module continues operation when the device is in Idle mode.

Remarks

By default, the USART module will continue operation in Idle mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_StopInIdleDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_StopInIdleDisable (USART_MODULE_ID index)

PLIB_USART_StopInIdleEnable Function

Discontinues operation when the device enters Idle mode.

File

plib_usart.h

C

```
void PLIB_USART_StopInIdleEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the USART module to discontinue operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_StopInIdleEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_StopInIdleEnable (USART_MODULE_ID index)

PLIB_USART_WakeOnStartDisable Function

Disables the wake-up on start bit detection feature during Sleep mode.

File

plib_usart.h

C

```
void PLIB_USART_WakeOnStartDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the wake-up on start bit detection feature during Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsWakeOnStart in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_WakeOnStartDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_WakeOnStartDisable (USART_MODULE_ID index)

PLIB_USART_WakeOnStartEnable Function

Enables the wake-up on start bit detection feature during Sleep mode.

File

plib_usart.h

C

```
void PLIB_USART_WakeOnStartEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the wake-up on start feature during Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsWakeOnStart in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_WakeOnStartEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_WakeOnStartEnable (USART_MODULE_ID index)

PLIB_USART_WakeOnStartIsEnabled Function

Gets the state of the sync break event completion.

File

plib_usart.h

C

```
bool PLIB_USART_WakeOnStartIsEnabled(USART_MODULE_ID index);
```

Returns

None.

Description

This function returns the status of the sync break event, when called after enabling using PLIB_USART_WakeOnStartEnable.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsWakeOnStart in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

//Call the interface just prior to entering the sleep mode.
PLIB_USART_WakeOnStartEnable(MY_USART_INSTANCE);

// Check the status if the Sync break event is over.
if(PLIB_USART_WakeOnStartIsEnabled(MY_USART_INSTANCE))
{
   // Do Something
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_WakeOnStartIsEnabled (USART_MODULE_ID index)

PLIB_USART_ErrorsGet Function

Return the status of all errors in the specified USART module.

File

plib_usart.h

C

```
USART_ERROR PLIB_USART_ErrorsGet(USART_MODULE_ID index);
```

Returns

Returns a bitmap of USART error status.

Description

This function returns status of all errors in the specified USART module. The return value can be bitwise AND'ed with a USART_ERROR type to know the status of a specific error.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability. Availability of this function can also be determined if all of the followings functions return true:

- PLIB_USART_ExistsReceiverFramingErrorStatus
- PLIB_USART_ExistsReceiverParityErrorStatus
- PLIB_USART_ExistsReceiverOverrunStatus

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

USART_ERROR error;

// Get the status of all errors.
```

```
error = PLIB_USART_ErrorsGet(MY_USART_INSTANCE);

// Check if parity error is active
if(error & USART_ERROR_PARITY)
{
    // Parity error is active.
}
else if(error & USART_ERROR_FRAMING)
{
    // Framing error is active.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

USART_ERROR PLIB_USART_ErrorsGet (USART_MODULE_ID index)

PLIB_USART_InitializeModeGeneral Function

Enables or disables general features of the USART module.

File

plib_usart.h

C

```
void PLIB_USART_InitializeModeGeneral(USART_MODULE_ID index, bool autobaud, bool loopBackMode, bool
wakeFromSleep, bool irdaMode, bool stopInIdle);
```

Returns

None.

Description

This function enables or disables general features of the USART module.

Remarks

Enabling the wake from sleep feature will cause the first character that is received by the USART module to be discarded. This feature should only be enabled if the CPU is to placed in power saving mode.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability. Availability of this function can also be determined if all of the following functions return true:

- PLIB_USART_ExistsLoopback
- PLIB_USART_ExistsBaudRateAutoDetect
- PLIB_USART_ExistsWakeOnStart
- PLIB_USART_ExistsIrDA
- PLIB_USART_ExistsStopInIdle

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Enable loopback, disable IrDA, disable auto baud detection and disable
// wake from sleep. Enable stop in idle
PLIB_USART_InitializeModeGeneral(MY_USART_INSTANCE, false, true,
                                                    false, false, true);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
autobaud	If true, auto baud rate detection is enabled. If false the feature is disabled.
loopBackMode	If true, loop back is enabled. If false the feature is disabled.

wakeFromSleep	If true, the USART module will wake up the CPU from sleep mode on USART activity. If false the feature is disabled.
irdaMode	If true, the IrDA mode is enabled. If false the feature is disabled.
stopInIdle	If true, module will stop functioning when CPU enters Idle mode. If false, the feature is disabled.

Function

void PLIB_USART_InitializeModeGeneral(USART_MODULE_ID index, bool autobaud,
bool loopBackMode, bool wakeFromSleep, bool irdaMode, bool stopInIdle);

PLIB_USART_InitializeOperation Function

Configures the Receive and Transmit FIFO interrupt levels and the hardware lines to be used by the module.

File

plib_usart.h

C

```
void PLIB_USART_InitializeOperation(USART_MODULE_ID index, USART_RECEIVE_INTR_MODE receiveInterruptMode,
USART_TRANSMIT_INTR_MODE transmitInterruptMode, USART_OPERATION_MODE operationMode);
```

Returns

None.

Description

This function configures the Receive and Transmit FIFO interrupt levels and the hardware lines to be used by the module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability. Availability of this function can also be determined if all of the following functions return true:

- PLIB_USART_ExistsReceiverInterruptMode
- PLIB_USART_ExistsTransmitterInterruptMode
- PLIB_USART_ExistsOperationMode

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Set receive FIFO to interrupt when FIFO is 3/4 level full
// Set Transmit FIFO to interrupt when FIFO is empty
// USART module will only use RX and TX hardware lines

PLIB_USART_InitializeOperation(MY_USART_INSTANCE, USART_RECEIVE_FIFO_3B4FULL,
    USART_TRANSMIT_FIFO_EMPTY  , USART_ENABLE_TX_RX_USED);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
receiveInterruptMode	Receiver FIFO interrupt level
transmitInterruptMode	Transmit FIFO interrupt level
operationMode	Hardware lines to be used by the USART.

Function

void PLIB_USART_InitializeOperation(USART_MODULE_ID index ,
USART_RECEIVE_INTR_MODE receiveInterruptMode,
USART_TRANSMIT_INTR_MODE transmitInterruptMode,
USART_OPERATION_MODE operationMode);

PLIB_USART_AddressGet Function

Gets the address for the Address Detect mode.

File

plib_usart.h

C

```
uint8_t PLIB_USART_AddressGet(USART_MODULE_ID index);
```

Returns

- address - The address being used

Description

This function returns the address value being used for the Address Detect mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddress in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1
uint8_t address = 0;
address = PLIB_USART_AddressGet (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

uint8_t PLIB_USART_AddressGet

(

USART_MODULE_ID index

)

PLIB_USART_AddressMaskGet Function

Gets the address mask for the Address Detect mode.

File

plib_usart.h

C

```
uint8_t PLIB_USART_AddressMaskGet(USART_MODULE_ID index);
```

Returns

- mask - Address mask being used

Description

This function returns the address mask value being used for the Address Detect mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressMask in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1
uint8_t mask = 0;
mask = PLIB_USART_AddressMaskGet (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_USART_AddressMaskGet

(

USART_MODULE_ID index

)

PLIB_USART_AddressMaskSet Function

Sets the address mask for the Address Detect mode.

File

plib_usart.h

C

```
void PLIB_USART_AddressMaskSet(USART_MODULE_ID index, uint8_t mask);
```

Returns

None.

Description

This function sets the address mask for the Address Detect mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressMask in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1
uint8_t mask = 0x0F;
PLIB_USART_AddressMaskSet (MY_USART_INSTANCE, mask);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
mask	Address match mask bits

Function

void PLIB_USART_AddressMaskSet

(

USART_MODULE_ID index,

uint8_t mask

)

PLIB_USART_AddressSet Function

Sets the address for the Address Detect mode.

File

plib_usart.h

C

```
void PLIB_USART_AddressSet(USART_MODULE_ID index, uint8_t address);
```

Returns

None.

Description

This function sets the address for the Address Detect mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddress in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1
uint8_t address = 0x02;
PLIB_USART_AddressSet (MY_USART_INSTANCE, address);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	Address

Function

void PLIB_USART_AddressSet
(
    USART_MODULE_ID index,
uint8_t address
)

PLIB_USART_ModuleIsBusy Function

Returns the USART module's running status.

File

plib_usart.h

C

```
bool PLIB_USART_ModuleIsBusy(USART_MODULE_ID index);
```

Returns

- true - USART module is busy
- false - USART module is idle

Description

This function checks if the USART module status is busy. The following functions should not be used when the module status is busy:

- PLIB_USART_LineControlModeSelect
- PLIB_USART_BaudRateHighSet
- PLIB_USART_BaudRateHighEnable
- PLIB_USART_BaudRateHighDisable
- PLIB_USART_ReceiverIdleStateLowEnable
- PLIB_USART_ReceiverIdleStateLowDisable
- PLIB_USART_BaudRateAutoDetectEnable
- PLIB_USART_LoopbackEnable

- PLIB_USART_LoopbackDisable
- PLIB_USART_WakeOnStartEnable
- PLIB_USART_WakeOnStartDisable
- PLIB_USART_OperationModeSelect
- PLIB_USART_HandshakeModeSelect
- PLIB_USART_IrDAEnable
- PLIB_USART_IrDADisable
- PLIB_USART_StopInIdleEnable
- PLIB_USART_StopInIdleDisable
- PLIB_USART_RunInOverflowEnable
- PLIB_USART_RunInOverflowDisable
- PLIB_USART_BRGClockSourceSelect
- PLIB_USART_RunInSleepModeEnable
- PLIB_USART_RunInSleepModeDisable

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsModuleBusyStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1
bool moduleStatus;

moduleStatus = PLIB_USART_ModuleIsBusy (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_USART_ModuleIsBusy (USART_MODULE_ID index)

PLIB_USART_RunInOverflowDisable Function

Disables the Run in overflow condition mode.

File

plib_usart.h

C

```
void PLIB_USART_RunInOverflowDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the USART module from accepting new data when an overflow error condition is detected.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsRunInOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_RunInOverflowDisable (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_RunInOverflowDisable (USART_MODULE_ID index)

PLIB_USART_RunInOverflowEnable Function

Enables the USART module to continue to operate when an overflow error condition has occurred.

File

plib_usart.h

C

```
void PLIB_USART_RunInOverflowEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the USART module to continue to operate when an overflow error condition has occurred.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsRunInOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_RunInOverflowEnable (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_RunInOverflowEnable (USART_MODULE_ID index)

PLIB_USART_RunInOverflowIsEnabled Function

Gets the status of the Run in Overflow condition.

File

plib_usart.h

C

```
bool PLIB_USART_RunInOverflowIsEnabled(USART_MODULE_ID index);
```

Returns

- true - Run in overflow condition is enabled
- false - Run in overflow condition is disabled

Description

This function indicates if the USART module has been enabled to run in an overflow condition.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use

PLIB_USART_ExistsRunInOverflow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

bool status;
status = PLIB_USART_RunInOverflowIsEnabled (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_USART_RunInOverflowIsEnabled (USART_MODULE_ID index)

PLIB_USART_RunInSleepModeDisable Function

Turns off the USART module's BRG clock during Sleep mode.

File

plib_usart.h

C

```
void PLIB_USART_RunInSleepModeDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function turns off the USART module's BRG clock during Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsRunInSleepMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_RunInSleepModeDisable (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_RunInSleepModeDisable (USART_MODULE_ID index)

PLIB_USART_RunInSleepModeEnable Function

Allows the USART module's BRG clock to run when the device enters Sleep mode.

File

plib_usart.h

C

```
void PLIB_USART_RunInSleepModeEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the USART module's BRG clock to continue operation when the device enters the Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsRunInSleepMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_RunInSleepModeEnable (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_RunInSleepModeEnable (USART_MODULE_ID index)

PLIB_USART_RunInSleepModeIsEnabled Function

Gets the status of Run in Sleep mode.

File

plib_usart.h

C

```
bool PLIB_USART_RunInSleepModeIsEnabled(USART_MODULE_ID index);
```

Returns

- true - Run in Sleep mode is enabled
- false - Run in Sleep mode is disabled

Description

This function indicates if the USART module has been enabled to run in Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsRunInSleepMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

bool status;
status = PLIB_USART_RunInSleepModeIsEnabled (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

bool PLIB_USART_RunInSleepModeIsEnabled (USART_MODULE_ID index)

b) Baud Rate Generator Functions

PLIB_USART_BaudRateAutoDetectEnable Function

Enables baud rate measurement on the next character, which requires reception of the Sync character.

File

plib_usart.h

C

```
void PLIB_USART_BaudRateAutoDetectEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the baud rate measurement on the next character, which requires reception of the Sync character.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRateAutoDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_BaudRateAutoDetectEnable(MY_USART_INSTANCE);

// Wait until the baud rate is detected.
while(!PLIB_USART_BaudRateAutoDetectIsComplete(MY_USART_INSTANCE));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_BaudRateAutoDetectEnable (USART_MODULE_ID index)

PLIB_USART_BaudRateAutoDetectIsComplete Function

Gets the state of the automatic baud detection.

File

plib_usart.h

C

```
bool PLIB_USART_BaudRateAutoDetectIsComplete(USART_MODULE_ID index);
```

Returns

- true - Baud rate detection is not complete
- false - Baud rate detection is complete

Description

This function gets the state of the automatic baud detection and returns a '0' if the baud rate auto detection is complete.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRateAutoDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_BaudRateAutoDetectEnable(MY_USART_INSTANCE);

// Wait until the baud rate is detected.
while(!PLIB_USART_BaudRateAutoDetectIsComplete(MY_USART_INSTANCE));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_BaudRateAutoDetectIsComplete (USART_MODULE_ID index)

PLIB_USART_BaudRateGet Function

Gets the baud rate current in use.

File

plib_usart.h

C

```
uint32_t PLIB_USART_BaudRateGet(USART_MODULE_ID index, int32_t clockFrequency);
```

Returns

- BaudRate - Baud rate value

Description

This function gets the baud rate that is currently in use. The clock frequency needs to be passed to the function.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRate in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint32_t baudRate ;
PLIB_USART_BaudRateSet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
baudRate = PLIB_USART_BaudRateGet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
clockFrequency	Clock Frequency

Function

uint32_t PLIB_USART_BaudRateGet (USART_MODULE_ID index,
int32_t clockFrequency);

PLIB_USART_BaudRateHighDisable Function

Disables the high baud rate selection.

File

plib_usart.h

C

```
void PLIB_USART_BaudRateHighDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the high baud rate selection.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRateHigh in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_BaudRateHighDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_BaudRateHighDisable (USART_MODULE_ID index)

PLIB_USART_BaudRateHighEnable Function

Enables high baud rate selection.

File

plib_usart.h

C

```
void PLIB_USART_BaudRateHighEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables high baud rate selection.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRateHigh in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_BaudRateHighEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_BaudRateHighEnable (USART_MODULE_ID index)

PLIB_USART_BaudRateHighSet Function

Sets the baud rate to the desired value.

File

plib_usart.h

C

```
void PLIB_USART_BaudRateHighSet(USART_MODULE_ID index, uint32_t clockFrequency, uint32_t baudRate);
```

Returns

None.

Description

This function sets the baud rate to the desired value.

Remarks

Setting a new baud rate value causes the baud rate timer to reset. This ensures that the baud rate timer does not have to overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receiver error or data loss may result. To avoid this issue verify that no receptions are in progress before changing the system clock.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRateHigh in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint32_t baudRateValue ;
PLIB_USART_BaudRateHighSet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
baudRateValue = PLIB_USART_BaudRateGet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baudRate	Baud Rate Value, it is the baud rate value
clockFrequency	Clock Frequency

Function

void PLIB_USART_BaudRateHighSet (USART_MODULE_ID index,

uint32_t clockFrequency, uint32_t baudRate);

PLIB_USART_BaudRateSet Function

Sets the baud rate to the desired value.

File

plib_usart.h

C

```
void PLIB_USART_BaudRateSet(USART_MODULE_ID index, uint32_t clockFrequency, uint32_t baudRate);
```

Returns

None.

Description

This function sets the baud rate to the desired value.

Remarks

Setting a new baud rate value causes the baud rate timer to reset. This ensures that the baud rate timer does not have to overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receiver error or data loss may result. To avoid this issue verify that no receptions are in progress before changing the system clock.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBaudRate in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint32_t baudRateValue ;
PLIB_USART_BaudRateSet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
baudRateValue = PLIB_USART_BaudRateGet(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baudRate	Baud Rate Value
clockFrequency	Clock Frequency

Function

void PLIB_USART_BaudRateSet (USART_MODULE_ID index, uint32_t clockFrequency,

uint32_t baudRate);

PLIB_USART_BaudSetAndEnable Function

Sets the baud rate to the desired value and enables the USART receiver, transmitter and the USART module.

File

plib_usart.h

C

```
void PLIB_USART_BaudSetAndEnable(USART_MODULE_ID index, uint32_t systemClock, uint32_t baud);
```

Returns

None.

Description

This function sets the baud rate to the desired value and enables the USART receiver, USART transmitter and USART module.

Remarks

Setting a new baud rate value causes the baud rate timer to reset. This ensures that the baud rate timer does not have to overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receiver error or data loss may result. To avoid this issue verify that no receptions are in progress before changing the system clock.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability. Availability of this function can also be determined if all of the following functions return true:

- PLIB_USART_ExistsBaudRate
- PLIB_USART_ExistsTransmitterEnable
- PLIB_USART_ExistsReceiverEnable

- PLIB_USART_ExistsEnable

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint32_t baudRateValue ;
PLIB_USART_BaudSetAndEnable(MY_USART_INSTANCE, MY_CLOCK_FREQUENCY, 9600);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
baudRate	Baud Rate Value
clockFrequency	Clock Frequency

Function

void PLIB_USART_BaudSetAndEnable (USART_MODULE_ID index, uint32_t clockFrequency, uint32_t baudRate);

PLIB_USART_BRGClockSourceGet Function

Gets the BRG clock source of the USART module.

File

plib_usart.h

C

```
USART_BRG_CLOCK_SOURCE PLIB_USART_BRGClockSourceGet(USART_MODULE_ID index);
```

Returns

One of the possible values of USART_BRG_CLOCK_SOURCE

Description

This function returns the BRG Clock source of the USART. Refer to USART_BRG_CLOCK_SOURCE for the possible clock sources.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBRGClockSourceSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

USART_BRG_CLOCK_SOURCE brgClockSource;
brgClockSource = PLIB_USART_BRGClockSourceGet (MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance

Function

USART_BRG_CLOCK_SOURCE PLIB_USART_BRGClockSourceGet (USART_MODULE_ID index)

PLIB_USART_BRGClockSourceSelect Function

Configures the BRG clock source of the USART module.

File

plib_usart.h

C

```
void PLIB_USART_BRGClockSourceSelect(USART_MODULE_ID index, USART_BRG_CLOCK_SOURCE brgClockSource);
```

Returns

None.

Description

This function configures the BRG Clock source of the USART. Refer to USART_BRG_CLOCK_SOURCE for the possible clock sources.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsBRGClockSourceSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_BRGClockSourceSelect (MY_USART_INSTANCE, USART_BRG_CLOCK_SOURCE_FRC_IN_SLEEP);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
brgClockSource	One of the possible values from USART_BRG_CLOCK_SOURCE

Function

void PLIB_USART_BRGClockSourceSelect
(
USART_MODULE_ID index,
USART_BRG_CLOCK_SOURCE brgClockSource
)

c) Transmitter Functions

PLIB_USART_Transmitter9BitsSend Function

Data to be transmitted in the byte mode with the 9th bit.

File

plib_usart.h

C

```
void PLIB_USART_Transmitter9BitsSend(USART_MODULE_ID index, int8_t data, bool Bit9th);
```

Returns

None.

Description

The data is transmitted in the byte mode for the specified USART module, with 9th bit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitter9BitsSend in your application to determine whether this feature is available.

Preconditions

The USART module should be configured to use the 9 data bits.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint8_t data = 'a';

if(!PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE))
{
    PLIB_USART_Transmitter9BitsSend(MY_USART_INSTANCE, data, false);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to be transmitted.
9thBit	9th bit of the data to be transmitted.

Function

void PLIB_USART_Transmitter9BitsSend (USART_MODULE_ID index,

int8_t data, bool 9thBit)

PLIB_USART_TransmitterBreakSend Function

Transmits the break character.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterBreakSend(USART_MODULE_ID index);
```

Returns

None.

Description

This function transmits the break character.

Remarks

After the break has been transmitted, the application can start transmitting next bytes.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterBreak in your application to determine whether this feature is available.

Preconditions

The application should wait for the transmitter to be idle, before calling this function.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Wait for the Transmit buffer to be empty.
while(PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE));

// Transmit the break character.
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);

// wait for the break transmission to complete
while(!PLIB_USART_TransmitterBreakSendIsComplete(MY_USART_INSTANCE));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_TransmitterBreakSend (USART_MODULE_ID index)

PLIB_USART_TransmitterBreakSendIsComplete Function

Returns the status of the break transmission

File

plib_usart.h

C

```
bool PLIB_USART_TransmitterBreakSendIsComplete(USART_MODULE_ID index);
```

Returns

- true - Transmit break on the next transmission
- false - Break transmission completed or not started

Description

The function returns the status of the break transmission.

Remarks

After the break has been transmitted, the application can start transmitting next bytes.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterBreak in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Wait for the Transmit buffer to be empty.
while(PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE));

// Transmit the break character.
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);

// wait for the break transmission to complete
while(!PLIB_USART_TransmitterBreakSendIsComplete(MY_USART_INSTANCE));
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_TransmitterBreakSendIsComplete (USART_MODULE_ID index)

PLIB_USART_TransmitterBufferIsFull Function

Gets the transmit buffer full status.

File

plib_usart.h

C

```
bool PLIB_USART_TransmitterBufferIsFull(USART_MODULE_ID index);
```

Returns

- true - The transmit buffer is full
- false - The transmit buffer is not full, at least one more character can be written

Description

This function gets the transmit status of the specified USART module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterBufferFullStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Wait for the Transmit buffer to be empty.
while(PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE));

// Transmit the break character.
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_TransmitterBufferIsFull (USART_MODULE_ID index)

PLIB_USART_TransmitterByteSend Function

Data to be transmitted in the Byte mode.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterByteSend(USART_MODULE_ID index, int8_t data);
```

Returns

None.

Description

The data is transmitted in the Byte mode for the specified USART module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitter in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint8_t data = 'a';
if(!PLIB_USART_TransmitterBufferIsFull(MY_USART_INSTANCE))
{
    PLIB_USART_TransmitterByteSend(MY_USART_INSTANCE, data);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
data	Data to be transmitted.

Function

void PLIB_USART_TransmitterByteSend (USART_MODULE_ID index, int8_t data)

PLIB_USART_TransmitterDisable Function

Disables the specific USART module transmitter.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the specific USART module transmitter.

Remarks

Disabling the transmitter during a transmission will cause the transmission to be aborted.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_TransmitterDisable (USART_MODULE_ID index)

PLIB_USART_TransmitterEnable Function

Enables the specific USART module transmitter.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the specific USART module transmitter.

Remarks

The transmitter should not be enabled until the USART is enabled. The transmissions will not be enabled otherwise.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterEnable in your application to determine whether this feature is available.

Preconditions

The USART module should be enabled using the function PLIB_USART_Enable before this function is called.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_TransmitterEnable (USART_MODULE_ID index)

PLIB_USART_TransmitterIdleIsLowDisable Function

Disables the Transmit Idle Low state.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterIdleIsLowDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the Transmit Idle Low state. In USART Synchronous mode, this function configures that the TX polarity the idle state is high.

When IrDA is enabled, this function sets the IrDA encoded Transmit Idle state to a '0'.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterIdleIsLow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterIdleIsLowDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_TransmitterIdleIsLowDisable (USART_MODULE_ID index)

PLIB_USART_TransmitterIdleIsLowEnable Function

Enables the Transmit Idle Low state.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterIdleIsLowEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the Transmit Idle Low state. In the USART Synchronous mode, this function configures that the TX polarity, the idle state is low.

When IrDA is enabled, this function sets that IrDA encoded Transmit Idle state to a '1'.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterIdleIsLow in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterIdleIsLowEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_TransmitterIdleIsLowEnable (USART_MODULE_ID index)

PLIB_USART_TransmitterInterruptModeSelect Function

Sets the USART transmitter interrupt mode.

File

plib_usart.h

C

```
void PLIB_USART_TransmitterInterruptModeSelect(USART_MODULE_ID index, USART_TRANSMIT_INTR_MODE fifolevel);
```

Returns

None.

Description

This function sets the condition in which the USART module should generate an interrupt.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterInterruptMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_TransmitterInterruptModeSelect(MY_USART_INSTANCE,
    USART_TRANSMIT_FIFO_EMPTY );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
interruptMode	Interrupt mode; for possible configurations, refer to USART_TRANSMIT_INTR_MODE

Function

void PLIB_USART_TransmitterInterruptModeSelect(USART_MODULE_ID index,
USART_TRANSMIT_INTR_MODE interruptMode)

PLIB_USART_TransmitterIsEmpty Function

Gets the transmit shift register empty status.

File

plib_usart.h

C

```
bool PLIB_USART_TransmitterIsEmpty(USART_MODULE_ID index);
```

Returns

- true - The transmit shift register is empty
- false - The transmit shift register is not empty

Description

This function gets the transmit shift register empty status.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsTransmitterEmptyStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

// Wait for the Transmit buffer to be empty.
while(!PLIB_USART_TransmitterIsEmpty(MY_USART_INSTANCE));

// Transmit the break character.
PLIB_USART_TransmitterBreakSend(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_TransmitterIsEmpty (USART_MODULE_ID index)

PLIB_USART_TransmitterAddressGet Function

Returns the address of the USART TX register

File

plib_usart.h

C

```
void* PLIB_USART_TransmitterAddressGet(USART_MODULE_ID index);
```

Returns

Address of the USART TX register

Description

This function returns the address of the USART TX register.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_TransmitterAddressGet(USART_MODULE_ID index)

d) Receiver Functions

PLIB_USART_ReceiverAddressAutoDetectDisable Function

Disables the automatic Address Detect mode.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverAddressAutoDetectDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the automatic Address Detect mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressAutoDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverAddressAutoDetectDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverAddressAutoDetectDisable (
USART_MODULE_ID index)

PLIB_USART_ReceiverAddressAutoDetectEnable Function

Setup the automatic Address Detect mode.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverAddressAutoDetectEnable(USART_MODULE_ID index, int8_t Mask);
```

Returns

None.

Description

This function configures the automatic Address Detect mode. Uses the mask as the address character for automatic address detection.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressAutoDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverAddressAutoDetectEnable(MY_USART_INSTANCE,
                                           MY_DEVICE_ADDRESS);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
Mask	Address character to be used, when enabled

Function

void PLIB_USART_ReceiverAddressAutoDetectEnable(USART_MODULE_ID index,
int8_t Mask)

PLIB_USART_ReceiverAddressDetectDisable Function

Enables the Address Detect mode.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverAddressDetectDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the Address Detect mode. If it is enabled, and the device is addressed, disable the Address Detect mode to continue receiving bytes.

Remarks

All bytes are received, and the 9th bit can be used as the parity bit. By default, the address detect is disabled.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverAddressDetectDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverAddressDetectDisable (USART_MODULE_ID index)

PLIB_USART_ReceiverAddressDetectEnable Function

Enables the Address Detect mode.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverAddressDetectEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the Address Detect mode. If it is enabled, and the device is addressed, disable the Address Detect mode to continue receiving bytes.

Remarks

If 9 data bits are not selected, this bit has no effect.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressDetect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverAddressDetectEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverAddressDetectEnable (USART_MODULE_ID index)

PLIB_USART_ReceiverAddressIsReceived Function

Checks and return if the data received is an address.

File

plib_usart.h

C

```
bool PLIB_USART_ReceiverAddressIsReceived(USART_MODULE_ID index);
```

Returns

- true - if the data received has the 9th bit set
- false - if the address received has the 9th bit cleared

Description

Checks and return if the data received is an address. The address has the 9th bit set. If data received has the 9th bit set, the function returns true; otherwise, the function returns false.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverAddressDetect in your application to determine whether this feature is available.

Preconditions

The USART module should be configured to use the 9 data bits.

Example

```
#define MY_USART_INSTANCE USART_ID_1

int8_t address;

if(PLIB_USART_ReceiverAddressIsReceived(MY_USART_INSTANCE))
{
    address = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_ReceiverAddressIsReceived (USART_MODULE_ID index)

PLIB_USART_ReceiverByteReceive Function

Data to be received in the Byte mode.

File

plib_usart.h

C

```
int8_t PLIB_USART_ReceiverByteReceive(USART_MODULE_ID index);
```

Returns

- data - Data to be received

Description

The data to be received in Byte mode from the specified USART module. Call the functions PLIB_USART_ReceiverFramingErrorHasOccurred, PLIB_USART_ReceiverParityErrorHasOccurred and PLIB_USART_ReceiverOverrunHasOccurred to get any error that occurred.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiver in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

bool isError;
uint8_t mydata;

if(PLIB_USART_ReceiverDataIsAvailable(MY_USART_INSTANCE))
{
    mydata = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
}

isError = PLIB_USART_ReceiverFramingErrorHasOccurred(MY_USART_INSTANCE)     |
              PLIB_USART_ReceiverParityErrorHasOccurred(MY_USART_INSTANCE) |
              PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE);
```

```
if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

int8_t PLIB_USART_ReceiverByteReceive (USART_MODULE_ID index)

PLIB_USART_ReceiverDataIsAvailable Function

Identifies if the receive data is available for the specified USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ReceiverDataIsAvailable(USART_MODULE_ID index);
```

Returns

- true - The data is available
- false - The data is not available

Description

This function identifies if the receive data is available for the specified USART module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverDataAvailableStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

int8_t mydata;

if(PLIB_USART_ReceiverDataIsAvailable(MY_USART_INSTANCE))
{
    mydata = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_ReceiverDataIsAvailable (USART_MODULE_ID index)

PLIB_USART_ReceiverDisable Function

Disables the USART receiver.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables the USART receiver.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverDisable (USART_MODULE_ID index)

PLIB_USART_ReceiverEnable Function

Enables the USART receiver.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables the USART receiver.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverEnable (USART_MODULE_ID index)

PLIB_USART_ReceiverFramingErrorHasOccurred Function

Gets the framing error status.

File

plib_usart.h

C

```
bool PLIB_USART_ReceiverFramingErrorHasOccurred(USART_MODULE_ID index);
```

Returns

- true - The framing error was detected on the current character
- false - The framing error was not detected on the current character

Description

This function gets the framing error status.

Remarks

Reading the error clears the error.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverFramingErrorStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

status = PLIB_USART_ReceiverFramingErrorHasOccurred(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_ReceiverFramingErrorHasOccurred (USART_MODULE_ID index)

PLIB_USART_ReceiverIdleStateLowDisable Function

Disables receive polarity inversion.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverIdleStateLowDisable(USART_MODULE_ID index);
```

Returns

None.

Description

This function disables receive polarity inversion. In the USART Synchronous mode, this function configures that the data is not inverted.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverIdleStateLowEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverIdleStateLowDisable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverIdleStateLowDisable (USART_MODULE_ID index);

PLIB_USART_ReceiverIdleStateLowEnable Function

Enables receive polarity inversion.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverIdleStateLowEnable(USART_MODULE_ID index);
```

Returns

None.

Description

This function enables receive polarity inversion. In the USART Synchronous mode, this function configures that the data is inverted.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverIdleStateLowEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverIdleStateLowEnable(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverIdleStateLowEnable (USART_MODULE_ID index)

PLIB_USART_ReceiverInterruptModeSelect Function

Sets the USART receiver FIFO level.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverInterruptModeSelect(USART_MODULE_ID index, USART_RECEIVE_INTR_MODE interruptMode);
```

Returns

None.

Description

This function sets the USART receiver FIFO level.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverInterruptMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

PLIB_USART_ReceiverInterruptModeSelect(MY_USART_INSTANCE,
    USART_RECEIVE_FIFO_ONE_CHAR );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
fifolevel	For possible configurations, refer to USART_RECEIVE_INTR_MODE

Function

void PLIB_USART_ReceiverInterruptModeSelect(USART_MODULE_ID index,
USART_RECEIVE_INTR_MODE interruptMode)

PLIB_USART_ReceiverIsIdle Function

Identifies if the receiver is idle.

File

plib_usart.h

C

```
bool PLIB_USART_ReceiverIsIdle(USART_MODULE_ID index);
```

Returns

- true - The receive buffer is idle
- false - The receive buffer is not idle

Description

This function identifies if the receiver is idle.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverIdleStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

int8_t mydata;

if(PLIB_USART_ReceiverIsIdle(MY_USART_INSTANCE))
{
    mydata = PLIB_USART_ReceiverByteReceive(MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_ReceiverIsIdle (USART_MODULE_ID index)

PLIB_USART_ReceiverOverrunErrorClear Function

Clears a USART vverrun error.

File

plib_usart.h

C

```
void PLIB_USART_ReceiverOverrunErrorClear(USART_MODULE_ID index);
```

Returns

None.

Description

This function clears an overrun error. Clearing the error, resets the receive buffer.

Remarks

WARNING: Calling this API will clear all of the previously received data.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverOverrunStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USART_ReceiverOverrunErrorClear (USART_MODULE_ID index)

PLIB_USART_ReceiverOverrunHasOccurred Function

Identifies if there was a receiver overrun error.

File

plib_usart.h

C

```
bool PLIB_USART_ReceiverOverrunHasOccurred(USART_MODULE_ID index);
```

Returns

- true - The receive buffer has overflowed
- false - The receive buffer has not overflowed

Description

This function identifies if there was a receiver overrun error.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverOverrunStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

if(PLIB_USART_ReceiverOverrunHasOccurred(MY_USART_INSTANCE))
{
    PLIB_USART_ReceiverOverrunErrorClear(MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_ReceiverOverrunHasOccurred (USART_MODULE_ID index)

PLIB_USART_ReceiverParityErrorHasOccurred Function

Gets the parity error status.

File

plib_usart.h

C

```
bool PLIB_USART_ReceiverParityErrorHasOccurred(USART_MODULE_ID index);
```

Returns

- true - The parity error was detected on the current character
- false - The parity error was not detected on the current character

Description

This function gets the parity error status.

Remarks

Reading the error clears the error. A Parity error is irrelevant in case of 9-bit mode. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiverParityErrorStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

status = PLIB_USART_ReceiverParityErrorHasOccurred(MY_USART_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

bool PLIB_USART_ReceiverParityErrorHasOccurred (USART_MODULE_ID index)

PLIB_USART_ReceiverAddressGet Function

Returns the address of the USART RX register

File

plib_usart.h

C

```
void* PLIB_USART_ReceiverAddressGet(USART_MODULE_ID index);
```

Returns

Address of the USART RX register

Description

This function returns the address of the USART RX register.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ReceiverAddressGet(USART_MODULE_ID index)

PLIB_USART_Receiver9BitsReceive Function

Data to be received in the byte mode with the 9th bit.

File

plib_usart.h

C

```
int16_t PLIB_USART_Receiver9BitsReceive(USART_MODULE_ID index);
```

Returns

- data - Data to be received

Description

The data to be received in Byte mode from the specified USART module. with the 9th bit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USART_ExistsReceiver9Bits in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#define MY_USART_INSTANCE USART_ID_1

uint16_t mydata;

if(PLIB_USART_ReceiverDataIsAvailable(MY_USART_INSTANCE))
{
    mydata = PLIB_USART_Receiver9BitsReceive (MY_USART_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

int16_t PLIB_USART_Receiver9BitsReceive (USART_MODULE_ID index)

e) Feature Existence Functions

PLIB_USART_ExistsBaudRate Function

Identifies whether the BaudRate feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsBaudRate(USART_MODULE_ID index);
```

Returns

- true - The BaudRate feature is supported on the device
- false - The BaudRate feature is not supported on the device

Description

This function identifies whether the BaudRate feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_BaudRateSet
- PLIB_USART_BaudRateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsBaudRate(USART_MODULE_ID index)

PLIB_USART_ExistsBaudRateAutoDetect Function

Identifies whether the BaudRateAutoDetect feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsBaudRateAutoDetect(USART_MODULE_ID index);
```

Returns

- true - The BaudRateAutoDetect feature is supported on the device
- false - The BaudRateAutoDetect feature is not supported on the device

Description

This function identifies whether the BaudRateAutoDetect feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_BaudRateAutoDetectEnable
- PLIB_USART_BaudRateAutoDetectIsComplete

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsBaudRateAutoDetect(USART_MODULE_ID index)

PLIB_USART_ExistsBaudRateHigh Function

Identifies whether the BaudRateHigh feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsBaudRateHigh(USART_MODULE_ID index);
```

Returns

- true - The BaudRateHigh feature is supported on the device
- false - The BaudRateHigh feature is not supported on the device

Description

This function identifies whether the BaudRateHigh feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_BaudRateHighSet
- PLIB_USART_BaudRateHighDisable
- PLIB_USART_BaudRateHighEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsBaudRateHigh(USART_MODULE_ID index)

PLIB_USART_ExistsEnable Function

Identifies whether the EnableControl feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsEnable(USART_MODULE_ID index);
```

Returns

- true - The EnableControl feature is supported on the device
- false - The EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_Disable
- PLIB_USART_Enable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsEnable(USART_MODULE_ID index)

PLIB_USART_ExistsHandshakeMode Function

Identifies whether the HandShakeMode feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsHandshakeMode(USART_MODULE_ID index);
```

Returns

- true - The HandShakeMode feature is supported on the device
- false - The HandShakeMode feature is not supported on the device

Description

This function identifies whether the HandShakeMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_HandshakeModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsHandshakeMode(USART_MODULE_ID index)

PLIB_USART_ExistsIrDA Function

Identifies whether the IrDAControl feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsIrDA(USART_MODULE_ID index);
```

Returns

- true - The IrDAControl feature is supported on the device
- false - The IrDAControl feature is not supported on the device

Description

This function identifies whether the IrDAControl feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_IrDADisable
- PLIB_USART_IrDAEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsIrDA(USART_MODULE_ID index)

PLIB_USART_ExistsLineControlMode Function

Identifies whether the LineControlMode feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsLineControlMode(USART_MODULE_ID index);
```

Returns

- true - The LineControlMode feature is supported on the device
- false - The LineControlMode feature is not supported on the device

Description

This function identifies whether the LineControlMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_LineControlModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsLineControlMode(USART_MODULE_ID index)

PLIB_USART_ExistsLoopback Function

Identifies whether the Loopback feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsLoopback(USART_MODULE_ID index);
```

Returns

- true - The Loopback feature is supported on the device
- false - The Loopback feature is not supported on the device

Description

This function identifies whether the Loopback feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_LoopbackEnable
- PLIB_USART_LoopbackDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsLoopback(USART_MODULE_ID index)

PLIB_USART_ExistsOperationMode Function

Identifies whether the OperationMode feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsOperationMode(USART_MODULE_ID index);
```

Returns

- true - The OperationMode feature is supported on the device
- false - The OperationMode feature is not supported on the device

Description

This function identifies whether the OperationMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_OperationModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsOperationMode(USART_MODULE_ID index)

PLIB_USART_ExistsReceiver Function

Identifies whether the Receiver feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiver(USART_MODULE_ID index);
```

Returns

- true - The Receiver feature is supported on the device
- false - The Receiver feature is not supported on the device

Description

This function identifies whether the Receiver feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_ReceiverByteReceive
- PLIB_USART_ReceiverAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiver(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverAddressAutoDetect Function

Identifies whether the ReceiverAddressAutoDetect feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverAddressAutoDetect(USART_MODULE_ID index);
```

Returns

- true - The ReceiverAddressAutoDetect feature is supported on the device
- false - The ReceiverAddressAutoDetect feature is not supported on the device

Description

This function identifies whether the ReceiverAddressAutoDetect feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_ReceiverAddressAutoDetectEnable
- PLIB_USART_ReceiverAddressAutoDetectDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverAddressAutoDetect(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverAddressDetect Function

Identifies whether the ReceiverAddressDetect feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverAddressDetect(USART_MODULE_ID index);
```

Returns

- true - The ReceiverAddressDetect feature is supported on the device
- false - The ReceiverAddressDetect feature is not supported on the device

Description

This function identifies whether the ReceiverAddressDetect feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_ReceiverAddressDetectEnable
- PLIB_USART_ReceiverAddressDetectDisable
- PLIB_USART_ReceiverAddressIsReceived

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverAddressDetect(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverDataAvailableStatus Function

Identifies whether the ReceiverDataAvailable feature exists on the USART module

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverDataAvailableStatus(USART_MODULE_ID index);
```

Returns

- true - The ReceiverDataAvailable feature is supported on the device
- false - The ReceiverDataAvailable feature is not supported on the device

Description

This function identifies whether the ReceiverDataAvailable feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_ReceiverDataIsAvailable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverDataAvailableStatus(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverEnable Function

Identifies whether the ReceiverEnableControl feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverEnable(USART_MODULE_ID index);
```

Returns

- true - The ReceiverEnableControl feature is supported on the device
- false - The ReceiverEnableControl feature is not supported on the device

Description

This function identifies whether the ReceiverEnableControl feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_ReceiverEnable
- PLIB_USART_ReceiverDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverEnable(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverFramingErrorStatus Function

Identifies whether the ReceiverFramingError feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverFramingErrorStatus(USART_MODULE_ID index);
```

Returns

- true - The ReceiverFramingError feature is supported on the device
- false - The ReceiverFramingError feature is not supported on the device

Description

This function identifies whether the ReceiverFramingError feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_ReceiverFramingErrorHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverFramingErrorStatus(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverIdleStateLowEnable Function

Identifies whether the ReceiverPolarityInvert feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverIdleStateLowEnable(USART_MODULE_ID index);
```

Returns

- true - The ReceiverPolarityInvert feature is supported on the device
- false - The ReceiverPolarityInvert feature is not supported on the device

Description

This function identifies whether the ReceiverPolarityInvert feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_ReceiverIdleStateLowEnable
- PLIB_USART_ReceiverIdleStateLowDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverIdleStateLowEnable(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverIdleStatus Function

Identifies whether the ReceiverIdle feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverIdleStatus(USART_MODULE_ID index);
```

Returns

- true - The ReceiverIdle feature is supported on the device
- false - The ReceiverIdle feature is not supported on the device

Description

This function identifies whether the ReceiverIdle feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_ReceiverIsIdle

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverIdleStatus(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverInterruptMode Function

Identifies whether the ReceiverInterruptMode feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverInterruptMode(USART_MODULE_ID index);
```

Returns

- true - The ReceiverInterruptMode feature is supported on the device
- false - The ReceiverInterruptMode feature is not supported on the device

Description

This function identifies whether the ReceiverInterruptMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_ReceiverInterruptModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverInterruptMode(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverOverrunStatus Function

Identifies whether the ReceiverOverrunError feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverOverrunStatus(USART_MODULE_ID index);
```

Returns

- true - The ReceiverOverrunError feature is supported on the device
- false - The ReceiverOverrunError feature is not supported on the device

Description

This function identifies whether the ReceiverOverrunError feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_ReceiverOverrunErrorClear
- PLIB_USART_ReceiverOverrunHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverOverrunStatus(USART_MODULE_ID index)

PLIB_USART_ExistsReceiverParityErrorStatus Function

Identifies whether the ReceiverParityError feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverParityErrorStatus(USART_MODULE_ID index);
```

Returns

- true - The ReceiverParityError feature is supported on the device
- false - The ReceiverParityError feature is not supported on the device

Description

This function identifies whether the ReceiverParityError feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_ReceiverParityErrorHasOccurred

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverParityErrorStatus(USART_MODULE_ID index)

PLIB_USART_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsStopInIdle(USART_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_StopInIdleEnable
- PLIB_USART_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsStopInIdle(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitter Function

Identifies whether the Transmitter feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitter(USART_MODULE_ID index);
```

Returns

- true - The Transmitter feature is supported on the device
- false - The Transmitter feature is not supported on the device

Description

This function identifies whether the Transmitter feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_TransmitterByteSend
- PLIB_USART_TransmitterAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitter(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitter9BitsSend Function

Identifies whether the Transmitter9Bits feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitter9BitsSend(USART_MODULE_ID index);
```

Returns

- true - The Transmitter9Bits feature is supported on the device
- false - The Transmitter9Bits feature is not supported on the device

Description

This function identifies whether the Transmitter9Bits feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_Transmitter9BitsSend

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitter9BitsSend(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitterBreak Function

Identifies whether the TransmitterBreak feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitterBreak(USART_MODULE_ID index);
```

Returns

- true - The TransmitterBreak feature is supported on the device
- false - The TransmitterBreak feature is not supported on the device

Description

This function identifies whether the TransmitterBreak feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_TransmitterBreakSend
- PLIB_USART_TransmitterBreakSendIsComplete

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitterBreak(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitterBufferFullStatus Function

Identifies whether the TransmitterBufferFull feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitterBufferFullStatus(USART_MODULE_ID index);
```

Returns

- true - The TransmitterBufferFull feature is supported on the device
- false - The TransmitterBufferFull feature is not supported on the device

Description

This function identifies whether the TransmitterBufferFull feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_TransmitterBufferIsFull

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitterBufferFullStatus(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitterEmptyStatus Function

Identifies whether the TransmitterEmpty feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitterEmptyStatus(USART_MODULE_ID index);
```

Returns

- true - The TransmitterEmpty feature is supported on the device
- false - The TransmitterEmpty feature is not supported on the device

Description

This function identifies whether the TransmitterEmpty feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_TransmitterIsEmpty

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitterEmptyStatus(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitterEnable Function

Identifies whether the TransmitterEnableControl feature exists on the USART module

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitterEnable(USART_MODULE_ID index);
```

Returns

- true - The TransmitterEnableControl feature is supported on the device
- false - The TransmitterEnableControl feature is not supported on the device

Description

This function identifies whether the TransmitterEnableControl feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_TransmitterEnable
- PLIB_USART_TransmitterDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitterEnable(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitterIdleIsLow Function

Identifies whether the TransmitterIdleIsLow feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitterIdleIsLow(USART_MODULE_ID index);
```

Returns

- true - The TransmitterIdleIsLow feature is supported on the device
- false - The TransmitterIdleIsLow feature is not supported on the device

Description

This function identifies whether the TransmitterIdleIsLow feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_TransmitterIdleIsLowDisable
- PLIB_USART_TransmitterIdleIsLowEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitterIdleIsLow(USART_MODULE_ID index)

PLIB_USART_ExistsTransmitterInterruptMode Function

Identifies whether the TransmitterInterruptMode feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsTransmitterInterruptMode(USART_MODULE_ID index);
```

Returns

- true - The TransmitterInterruptMode feature is supported on the device
- false - The TransmitterInterruptMode feature is not supported on the device

Description

This function identifies whether the TransmitterInterruptMode feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_TransmitterInterruptModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsTransmitterInterruptMode(USART_MODULE_ID index)

PLIB_USART_ExistsWakeOnStart Function

Identifies whether the WakeOnStart feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsWakeOnStart(USART_MODULE_ID index);
```

Returns

- true - The WakeOnStart feature is supported on the device
- false - The WakeOnStart feature is not supported on the device

Description

This function identifies whether the WakeOnStart feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_WakeOnStartEnable
- PLIB_USART_WakeOnStartDisable
- PLIB_USART_WakeOnStartIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsWakeOnStart(USART_MODULE_ID index)

PLIB_USART_ExistsReceiver9Bits Function

Identifies whether the 9 Bits Receiver feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiver9Bits(USART_MODULE_ID index);
```

Returns

- true - The feature is supported on the device
- false - The feature is not supported on the device

Description

This function identifies whether the 9 Bits Receiver feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_Receiver9BitsReceive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiver9Bits (USART_MODULE_ID index)

PLIB_USART_ExistsBRGClockSourceSelect Function

Identifies whether the BRG Clock source select feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsBRGClockSourceSelect(USART_MODULE_ID index);
```

Returns

- true - The BRG clock source select feature is supported on the device
- false - The BRG clock source select feature is not supported on the device

Description

This function identifies whether the BRG Clock source select feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_BRGClockSourceSelect
- PLIB_USART_BRGClockSourceGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsBRGClockSourceSelect (USART_MODULE_ID index)

PLIB_USART_ExistsModuleBusyStatus Function

Identifies whether the module running status feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsModuleBusyStatus(USART_MODULE_ID index);
```

Returns

- true - The Module running status feature is supported on the device
- false - The Module running status feature is not supported on the device

Description

This function identifies whether the module running status feature is available on the USART module. When this function returns true, this function is supported on the device:

- PLIB_USART_ModuleIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsModuleBusyStatus (USART_MODULE_ID index)

PLIB_USART_ExistsReceiverAddress Function

Identifies whether the Receiver Address feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverAddress(USART_MODULE_ID index);
```

Returns

- true - The Receiver address feature is supported on the device
- false - The Receiver address feature is not supported on the device

Description

This function identifies whether the Receiver Address feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_AddressSet
- PLIB_USART_AddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverAddress (USART_MODULE_ID index)

PLIB_USART_ExistsReceiverAddressMask Function

Identifies whether the Receiver Address Mask feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsReceiverAddressMask(USART_MODULE_ID index);
```

Returns

- true - The Receiver address mask feature is supported on the device
- false - The Receiver address mask feature is not supported on the device

Description

This function identifies whether the Receiver Address Mask feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_AddressMaskSet
- PLIB_USART_AddressMaskGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsReceiverAddressMask (USART_MODULE_ID index)

PLIB_USART_ExistsRunInOverflow Function

Identifies whether the Run in overflow condition feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsRunInOverflow(USART_MODULE_ID index);
```

Returns

- true - The Run in Overflow condition feature is supported on the device
- false - The Run in Overflow condition feature is not supported on the device

Description

This function identifies whether the Run in Overflow condition feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_RunInOverflowEnable
- PLIB_USART_RunInOverflowDisable
- PLIB_USART_RunInOverflowIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsRunInOverflow (USART_MODULE_ID index)

PLIB_USART_ExistsRunInSleepMode Function

Identifies whether the Run in Sleep mode feature exists on the USART module.

File

plib_usart.h

C

```
bool PLIB_USART_ExistsRunInSleepMode(USART_MODULE_ID index);
```

Returns

- true - The Run in Sleep mode feature is supported on the device
- false - The Run in Sleep mode feature is not supported on the device

Description

This function identifies whether the Run in Sleep mode feature is available on the USART module. When this function returns true, these functions are supported on the device:

- PLIB_USART_RunInSleepModeEnable
- PLIB_USART_RunInSleepModeDisable
- PLIB_USART_RunInSleepModeIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USART_ExistsRunInSleepMode (USART_MODULE_ID index)

f) Data Types and Constants

USART_HANDSHAKE_MODE Enumeration

Lists the USART handshake modes.

File

plib_usart_help.h

C

```
typedef enum {
  USART_HANDSHAKE_MODE_FLOW_CONTROL,
  USART_HANDSHAKE_MODE_SIMPLEX
} USART_HANDSHAKE_MODE;
```

Members

Members	Description
USART_HANDSHAKE_MODE_FLOW_CONTROL	Enables flow control
USART_HANDSHAKE_MODE_SIMPLEX	Enables simplex mode communication, no flow control

Description

USART Handshake modes

This enumeration lists the USART handshake modes.

Remarks

None.

USART_LINECONTROL_MODE Enumeration

Data type defining the different configurations by which the USART data flow can be configured.

File

plib_usart_help.h

C

```
typedef enum {
  USART_8N1,
  USART_8E1,
  USART_8O1,
  USART_8N2,
  USART_8E2,
  USART_8O2,
  USART_9N1,
  USART_9N2
} USART_LINECONTROL_MODE;
```

Members

Members	Description
USART_8N1	8 Data Bits, No Parity,one Stop Bit
USART_8E1	8 Data Bits, Even Parity, 1 stop bit
USART_8O1	8 Data Bits, odd Parity, 1 stop bit
USART_8N2	8 Data Bits, No Parity,two Stop Bits
USART_8E2	8 Data Bits, Even Parity, 2 stop bits
USART_8O2	8 Data Bits, odd Parity, 2 stop bits
USART_9N1	9 Data Bits, No Parity, 1 stop bit
USART_9N2	9 Data Bits, No Parity, 2 stop bits

Description

Data Flow configuration

This data type defines the different configurations by which the USART can be configured for the data flow.

Remarks

None.

USART_MODULE_ID Enumeration

File

plib_usart_help.h

C

```
typedef enum {
  USART_ID_1,
  USART_ID_2,
  USART_ID_3,
  USART_ID_4,
  USART_ID_5,
  USART_ID_6,
  USART_NUMBER_OF_MODULES
} USART_MODULE_ID;
```

Members

Members	Description
USART_ID_1	USART 1
USART_ID_2	USART 2
USART_ID_3	USART 3
USART_ID_4	USART 4
USART_ID_5	USART 5
USART_ID_6	USART 6
USART_NUMBER_OF_MODULES	Total number of USART modules available

Description

Enumeration: USART_MODULE_ID

This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on Microchip microcontrollers.

Refer to the data sheet to get the correct number of modules defined for desired microcontroller.

_USART_MODULE_ID Macro

File

plib_usart_help.h

C

```
#define _USART_MODULE_ID
```

USART_OPERATION_MODE Enumeration

Data type defining the different configurations by which the USART can be enabled.

File

plib_usart_help.h

C

```
typedef enum {
  USART_ENABLE_TX_RX_BCLK_USED,
  USART_ENABLE_TX_RX_CTS_RTS_USED,
  USART_ENABLE_TX_RX_RTS_USED,
  USART_ENABLE_TX_RX_USED
} USART_OPERATION_MODE;
```

Members

Members	Description
USART_ENABLE_TX_RX_BCLK_USED	TX, RX and BCLK pins are used by USART module
USART_ENABLE_TX_RX_CTS_RTS_USED	TX, RX, CTS and RTS pins are used by USART module
USART_ENABLE_TX_RX_RTS_USED	TX, RX and RTS pins are used by USART module
USART_ENABLE_TX_RX_USED	TX and RX pins are used by USART module

Description

Enable configuration

This data type defines the different configurations by which the USART can be enabled.

Remarks

The actual definition of this enumeration is device-specific.

USART_RECEIVE_INTR_MODE Enumeration

Data type defining the different Receive FIFO levels by which the USART receive interrupt modes can be configured.

File

plib_usart_help.h

C

```
typedef enum {
  USART_RECEIVE_FIFO_HALF_FULL,
  USART_RECEIVE_FIFO_3B4FULL,
  USART_RECEIVE_FIFO_ONE_CHAR
} USART_RECEIVE_INTR_MODE;
```

Members

Members	Description
USART_RECEIVE_FIFO_HALF_FULL	Interrupt when receive buffer is half full
USART_RECEIVE_FIFO_3B4FULL	Interrupt when receive buffer is 3/4 full
USART_RECEIVE_FIFO_ONE_CHAR	Interrupt when a character is received

Description

Receive Interrupt mode configuration

This data type defining the different Receive FIFO levels by which the USART receive interrupt modes can be configured.

Remarks

None.

USART_TRANSMIT_INTR_MODE Enumeration

Data type defining the different transmit FIFO levels by which the USART transmit interrupt modes can be configured.

File

plib_usart_help.h

C

```
typedef enum {
  USART_TRANSMIT_FIFO_EMPTY,
  USART_TRANSMIT_FIFO_IDLE,
  USART_TRANSMIT_FIFO_NOT_FULL
} USART_TRANSMIT_INTR_MODE;
```

Members

Members	Description
USART_TRANSMIT_FIFO_EMPTY	Interrupt when the transmit buffer becomes empty
USART_TRANSMIT_FIFO_IDLE	Interrupt when all characters are transmitted
USART_TRANSMIT_FIFO_NOT_FULL	Interrupt when at least one location is empty in the transmit buffer

Description

Transmit Interrupt mode configuration

This data type defining the different transmit FIFO levels by which the USART transmit interrupt modes can be configured.

Remarks

None.

Files

Files

Name	Description
plib_usart.h	USART Peripheral Library interface header.
plib_usart_help.h	

Description

This section lists the source and header files used by the library.

plib_usart.h

USART Peripheral Library interface header.

Functions

	Name	Description
	PLIB_USART_AddressGet	Gets the address for the Address Detect mode.
	PLIB_USART_AddressMaskGet	Gets the address mask for the Address Detect mode.
	PLIB_USART_AddressMaskSet	Sets the address mask for the Address Detect mode.
	PLIB_USART_AddressSet	Sets the address for the Address Detect mode.
	PLIB_USART_BaudRateAutoDetectEnable	Enables baud rate measurement on the next character, which requires reception of the Sync character.
	PLIB_USART_BaudRateAutoDetectIsComplete	Gets the state of the automatic baud detection.
	PLIB_USART_BaudRateGet	Gets the baud rate current in use.

	PLIB_USART_BaudRateHighDisable	Disables the high baud rate selection.
	PLIB_USART_BaudRateHighEnable	Enables high baud rate selection.
	PLIB_USART_BaudRateHighSet	Sets the baud rate to the desired value.
	PLIB_USART_BaudRateSet	Sets the baud rate to the desired value.
	PLIB_USART_BaudSetAndEnable	Sets the baud rate to the desired value and enables the USART receiver, transmitter and the USART module.
	PLIB_USART_BRGClockSourceGet	Gets the BRG clock source of the USART module.
	PLIB_USART_BRGClockSourceSelect	Configures the BRG clock source of the USART module.
	PLIB_USART_Disable	Disables the specific USART module
	PLIB_USART_Enable	Enables the specific USART module.
	PLIB_USART_ErrorsGet	Return the status of all errors in the specified USART module.
	PLIB_USART_ExistsBaudRate	Identifies whether the BaudRate feature exists on the USART module.
	PLIB_USART_ExistsBaudRateAutoDetect	Identifies whether the BaudRateAutoDetect feature exists on the USART module.
	PLIB_USART_ExistsBaudRateHigh	Identifies whether the BaudRateHigh feature exists on the USART module.
	PLIB_USART_ExistsBRGClockSourceSelect	Identifies whether the BRG Clock source select feature exists on the USART module.
	PLIB_USART_ExistsEnable	Identifies whether the EnableControl feature exists on the USART module.
	PLIB_USART_ExistsHandshakeMode	Identifies whether the HandShakeMode feature exists on the USART module.
	PLIB_USART_ExistsIrDA	Identifies whether the IrDAControl feature exists on the USART module.
	PLIB_USART_ExistsLineControlMode	Identifies whether the LineControlMode feature exists on the USART module.
	PLIB_USART_ExistsLoopback	Identifies whether the Loopback feature exists on the USART module.
	PLIB_USART_ExistsModuleBusyStatus	Identifies whether the module running status feature exists on the USART module.
	PLIB_USART_ExistsOperationMode	Identifies whether the OperationMode feature exists on the USART module.
	PLIB_USART_ExistsReceiver	Identifies whether the Receiver feature exists on the USART module.
	PLIB_USART_ExistsReceiver9Bits	Identifies whether the 9 Bits Receiver feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddress	Identifies whether the Receiver Address feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddressAutoDetect	Identifies whether the ReceiverAddressAutoDetect feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddressDetect	Identifies whether the ReceiverAddressDetect feature exists on the USART module.
	PLIB_USART_ExistsReceiverAddressMask	Identifies whether the Receiver Address Mask feature exists on the USART module.
	PLIB_USART_ExistsReceiverDataAvailableStatus	Identifies whether the ReceiverDataAvailable feature exists on the USART module
	PLIB_USART_ExistsReceiverEnable	Identifies whether the ReceiverEnableControl feature exists on the USART module.
	PLIB_USART_ExistsReceiverFramingErrorStatus	Identifies whether the ReceiverFramingError feature exists on the USART module.
	PLIB_USART_ExistsReceiverIdleStateLowEnable	Identifies whether the ReceiverPolarityInvert feature exists on the USART module.
	PLIB_USART_ExistsReceiverIdleStatus	Identifies whether the ReceiverIdle feature exists on the USART module.
	PLIB_USART_ExistsReceiverInterruptMode	Identifies whether the ReceiverInterruptMode feature exists on the USART module.
	PLIB_USART_ExistsReceiverOverrunStatus	Identifies whether the ReceiverOverrunError feature exists on the USART module.
	PLIB_USART_ExistsReceiverParityErrorStatus	Identifies whether the ReceiverParityError feature exists on the USART module.
	PLIB_USART_ExistsRunInOverflow	Identifies whether the Run in overflow condition feature exists on the USART module.
	PLIB_USART_ExistsRunInSleepMode	Identifies whether the Run in Sleep mode feature exists on the USART module.
	PLIB_USART_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the USART module.
	PLIB_USART_ExistsTransmitter	Identifies whether the Transmitter feature exists on the USART module.
	PLIB_USART_ExistsTransmitter9BitsSend	Identifies whether the Transmitter9Bits feature exists on the USART module.
	PLIB_USART_ExistsTransmitterBreak	Identifies whether the TransmitterBreak feature exists on the USART module.
	PLIB_USART_ExistsTransmitterBufferFullStatus	Identifies whether the TransmitterBufferFull feature exists on the USART module.
	PLIB_USART_ExistsTransmitterEmptyStatus	Identifies whether the TransmitterEmpty feature exists on the USART module.

	PLIB_USART_ExistsTransmitterEnable	Identifies whether the TransmitterEnableControl feature exists on the USART module
	PLIB_USART_ExistsTransmitterIdleIsLow	Identifies whether the TransmitterIdleIsLow feature exists on the USART module.
	PLIB_USART_ExistsTransmitterInterruptMode	Identifies whether the TransmitterInterruptMode feature exists on the USART module.
	PLIB_USART_ExistsWakeOnStart	Identifies whether the WakeOnStart feature exists on the USART module.
	PLIB_USART_HandshakeModeSelect	Sets the data flow configuration.
	PLIB_USART_InitializeModeGeneral	Enables or disables general features of the USART module.
	PLIB_USART_InitializeOperation	Configures the Receive and Transmit FIFO interrupt levels and the hardware lines to be used by the module.
	PLIB_USART_IrDADisable	Disables the IrDA encoder and decoder.
	PLIB_USART_IrDAEnable	Enables the IrDA encoder and decoder.
	PLIB_USART_LineControlModeSelect	Sets the data flow configuration.
	PLIB_USART_LoopbackDisable	Disables Loopback mode.
	PLIB_USART_LoopbackEnable	Enables Loopback mode.
	PLIB_USART_ModuleIsBusy	Returns the USART module's running status.
	PLIB_USART_OperationModeSelect	Configures the operation mode of the USART module.
	PLIB_USART_Receiver9BitsReceive	Data to be received in the byte mode with the 9th bit.
	PLIB_USART_ReceiverAddressAutoDetectDisable	Disables the automatic Address Detect mode.
	PLIB_USART_ReceiverAddressAutoDetectEnable	Setup the automatic Address Detect mode.
	PLIB_USART_ReceiverAddressDetectDisable	Enables the Address Detect mode.
	PLIB_USART_ReceiverAddressDetectEnable	Enables the Address Detect mode.
	PLIB_USART_ReceiverAddressGet	Returns the address of the USART RX register
	PLIB_USART_ReceiverAddressIsReceived	Checks and return if the data received is an address.
	PLIB_USART_ReceiverByteReceive	Data to be received in the Byte mode.
	PLIB_USART_ReceiverDataIsAvailable	Identifies if the receive data is available for the specified USART module.
	PLIB_USART_ReceiverDisable	Disables the USART receiver.
	PLIB_USART_ReceiverEnable	Enables the USART receiver.
	PLIB_USART_ReceiverFramingErrorHasOccurred	Gets the framing error status.
	PLIB_USART_ReceiverIdleStateLowDisable	Disables receive polarity inversion.
	PLIB_USART_ReceiverIdleStateLowEnable	Enables receive polarity inversion.
	PLIB_USART_ReceiverInterruptModeSelect	Sets the USART receiver FIFO level.
	PLIB_USART_ReceiverIsIdle	Identifies if the receiver is idle.
	PLIB_USART_ReceiverOverrunErrorClear	Clears a USART vverrun error.
	PLIB_USART_ReceiverOverrunHasOccurred	Identifies if there was a receiver overrun error.
	PLIB_USART_ReceiverParityErrorHasOccurred	Gets the parity error status.
	PLIB_USART_RunInOverflowDisable	Disables the Run in overflow condition mode.
	PLIB_USART_RunInOverflowEnable	Enables the USART module to continue to operate when an overflow error condition has occurred.
	PLIB_USART_RunInOverflowIsEnabled	Gets the status of the Run in Overflow condition.
	PLIB_USART_RunInSleepModeDisable	Turns off the USART module's BRG clock during Sleep mode.
	PLIB_USART_RunInSleepModeEnable	Allows the USART module's BRG clock to run when the device enters Sleep mode.
	PLIB_USART_RunInSleepModeIsEnabled	Gets the status of Run in Sleep mode.
	PLIB_USART_StopInIdleDisable	Disables the Stop in Idle mode (the USART module continues operation when the device is in Idle mode).
	PLIB_USART_StopInIdleEnable	Discontinues operation when the device enters Idle mode.
	PLIB_USART_Transmitter9BitsSend	Data to be transmitted in the byte mode with the 9th bit.
	PLIB_USART_TransmitterAddressGet	Returns the address of the USART TX register
	PLIB_USART_TransmitterBreakSend	Transmits the break character.
	PLIB_USART_TransmitterBreakSendIsComplete	Returns the status of the break transmission
	PLIB_USART_TransmitterBufferIsFull	Gets the transmit buffer full status.
	PLIB_USART_TransmitterByteSend	Data to be transmitted in the Byte mode.
	PLIB_USART_TransmitterDisable	Disables the specific USART module transmitter.
	PLIB_USART_TransmitterEnable	Enables the specific USART module transmitter.
	PLIB_USART_TransmitterIdleIsLowDisable	Disables the Transmit Idle Low state.

	PLIB_USART_TransmitterIdleIsLowEnable	Enables the Transmit Idle Low state.
	PLIB_USART_TransmitterInterruptModeSelect	Sets the USART transmitter interrupt mode.
	PLIB_USART_TransmitterIsEmpty	Gets the transmit shift register empty status.
	PLIB_USART_WakeOnStartDisable	Disables the wake-up on start bit detection feature during Sleep mode.
	PLIB_USART_WakeOnStartEnable	Enables the wake-up on start bit detection feature during Sleep mode.
	PLIB_USART_WakeOnStartIsEnabled	Gets the state of the sync break event completion.

Description

USART Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USART Peripheral Library for all families of Microchip microcontrollers. The functions in this file are common to the USART module.

File Name

plib_usart.h

Company

Microchip Technology Inc.

plib_usart_help.h

Enumerations

	Name	Description
	USART_HANDSHAKE_MODE	Lists the USART handshake modes.
	USART_LINECONTROL_MODE	Data type defining the different configurations by which the USART data flow can be configured.
	USART_MODULE_ID	Enumeration: USART_MODULE_ID This enumeration defines the number of modules that are available on the microcontroller. This is the superset of all of the possible instances that might be available on Microchip microcontrollers. Refer to the data sheet to get the correct number of modules defined for desired microcontroller.
	USART_OPERATION_MODE	Data type defining the different configurations by which the USART can be enabled.
	USART_RECEIVE_INTR_MODE	Data type defining the different Receive FIFO levels by which the USART receive interrupt modes can be configured.
	USART_TRANSMIT_INTR_MODE	Data type defining the different transmit FIFO levels by which the USART transmit interrupt modes can be configured.

Macros

	Name	Description
	_USART_MODULE_ID	

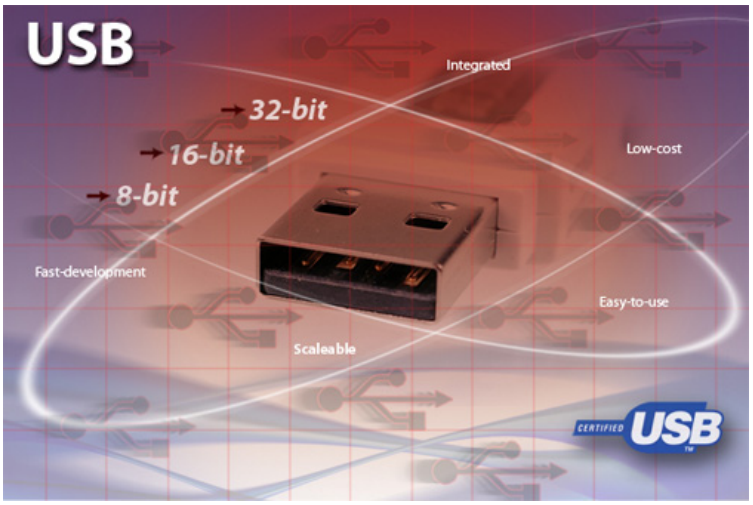
USB Peripheral Library

This section describes the USB Peripheral Library.

Introduction

This library provides a low-level abstraction of the USB module on Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description



USB Overview

USB is an asynchronous serial interface with a tiered star configuration. USB is implemented as a master/slave configuration. On a given bus, there can be multiple (up to 127) slaves (devices), but there is only one master (host). There are three possible module implementations: host, device and OTG dual role. The user should have an understanding of the USB documents available on the USB implementers web site (www.usb.org).

Device

The USB device accepts data from the host and responds to requests for data. It performs some peripheral functions, such as a mouse or data storage device.

- Functionality may be class or vendor-specific
- Draws 100 mA or less before configuration
- Can draw up to 500 mA after successful negotiation with the host
- Can support low-speed, full-speed or high-speed protocol. Hi-speed support requires implementation of full-speed. (*Low, Full, and Hi-Speed supported by this module.*)
- Supports control transfers. Supports data transfers required for implementation
- Optionally supports Session Request Protocol (SRP)
- Can be bus-powered or self-powered

Host

The host is the master in a USB system and is responsible for identifying all devices connected to it (enumeration), initiating all transfers, allocating bus bandwidth and supplying power to any bus-powered USB devices connected directly to it. There are two types of hosts.

USB Standard Host:

- A large variety of devices are supported
- This host supports all USB transfer types
- USB hubs are supported to allow connection of multiple devices simultaneously
- Device drivers can be updated to support new devices
- A type ‘A’ receptacle is used for each port
- Each port must be able to deliver a minimum of 100 mA for a configured or unconfigured
- device, and optionally, up to 500 mA for a configured device
- Full-Speed and Low-Speed must be supported. Hi-Speed can be supported.
- This is a typical personal computer implementation

Embedded Host

- Only supports a specific list of devices, referred to as a Targeted Peripheral List (TPL)
- This type of host is only required to support transfer types required by devices in the TPL
- USB hub support is optional (*Provided in this module*)
- Device drivers are not required to be pocketable
- A type ‘A’ receptacle is used for each port
- Only speeds required by devices in the TLP must be supported. (*Low, Full, and Hi-Speed supported by this module.*)
- Each port must be able to deliver a minimum of 100 mA for a configured or unconfigured device, and optionally, up to 500 mA for a configured device

- This is a typical implementation for a microcontroller

USB On-the-Go (OTG)

The OTG dual role device supports both USB host and device functionality. OTG dual role devices use a micro-AB receptacle. This allows a micro-A or a micro-B plug to be attached. Both the micro-A and micro-B plugs have an additional pin, the ID pin, to signify the connection type. The plug type, micro-A or micro-B, determines the default role of the OTG device, host or USB device. An OTG device will perform the role of a host when a micro-A plug is detected. When a micro-B plug is detected, the role of a USB device is performed.

When an OTG device is directly connected to another OTG device using an OTG cable, micro-A to micro-B, Host Negotiation Protocol (HNP) can be used to swap the roles of the host and USB device between the two without disconnecting and reconnecting cabling. To differentiate between the two OTG devices, the term, "A-device", is used to refer to the device connected to the micro-A plug and "B-device" is used to refer to the OTG device connected to the micro-B plug.

OTG dual role operating as a host (A-device):

- Only supports a specific list of devices, referred to as a Targeted Peripheral List (TPL). Generic class support is not allowed.
- Only required to support transaction types required by devices in the TPL
- USB hub support is optional. (*Multi-point support provided by this module.*)
- Device drivers are not required to be pocketable
- Only a single micro-AB receptacle is used
- Only Full-Speed must be supported. Hi-Speed and/or Low-Speed can be supported. (*Low, Full, and Hi-Speed supported by this module.*)
- The USB port must be able to deliver a minimum of 8 mA for a configured or unconfigured device, and optionally, up to 500 mA for a configured device
- Supports Host Negotiation Protocol (HNP) and Session Request Protocol (SRP). The host can switch roles to become a device. The initial role as a host or device is determined by the plug type, micro-A or micro-B, inserted into the micro-AB receptacle
- The A-device supplies VBUS power, when the bus is powered, even if the roles are swapped using HNP

OTG dual role operating as a USB device (B-device):

- Class or vendor-specific functionality
- Draws 8 mA or less before configuration
- Is typically self-powered due to low-current requirements, but can draw up to 500 mA after successful negotiation with the host
- Only a single micro-AB receptacle is used
- Must support Full-Speed. Support of Low-Speed and/or Hi-Speed is optional
- Supports control transactions. Supports data transactions required for implementation.
- Supports Session Request Protocol (SRP) and/or Host Negotiation Protocol (HNP). (*This module supports both SRP and HNP.*)
- The A-device supplies VBUS power, when the bus is powered, even if the roles are swapped using HNP

Additional Features of the Hi-Speed USB Module

- Operates either as a function controller of a Hi-Speed/Full-Speed USB device or as the host/device in a point-to-point or multi-point communications with other USB function
- Supports OTG communications with on or more Hi-Speed, Full-Speed, or Low-Speed devices
- Provides soft connect/disconnect.
- In addition to Endpoint Zero, supports seven transmit and seven receive endpoints
- Dynamic FIFO sizing for Endpoints 1-7. (Endpoint Zero FIFO fixed at 64 bytes.) FIFOs use module-internal SRAM.
- Module-internal eight channel DMA with access to all FIFOs
- All host transaction scheduling supported in hardware
- Supports Link Power Management

Using the Library

This topic describes the basic architecture of the USB Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_usb.h`

The interface to the USB Peripheral Library is defined in the `plib_usb.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the USB Peripheral Library must include `peripheral.h`.

Library File:

The USB Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Hardware Abstraction Models

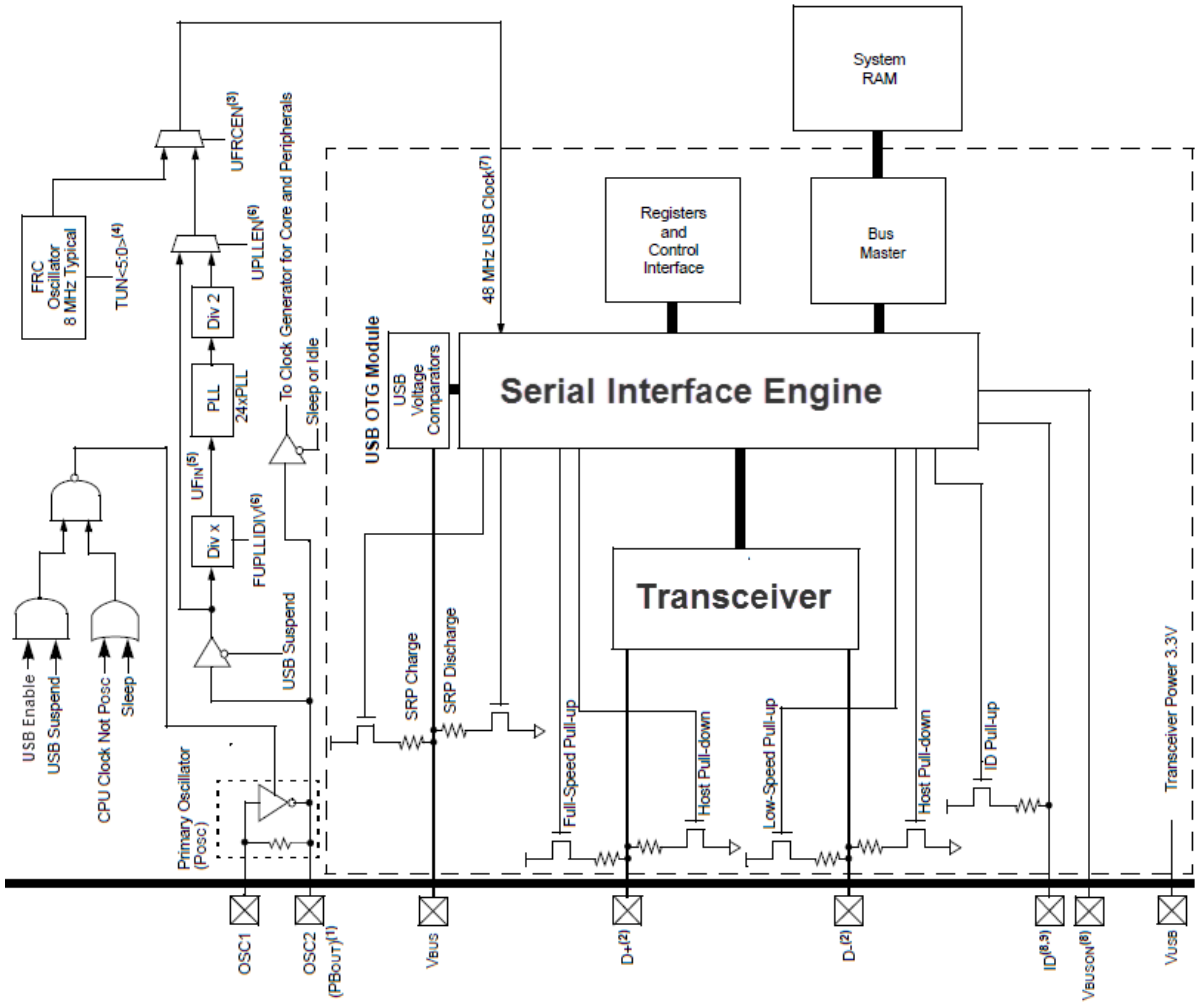
This section describes the hardware abstraction model for the USB Peripheral Library.

Description

Hardware Abstraction Model

The following figure shows the USB Module for the PIC32 family of devices.

PIC32 USB Module



Note 1: PB clock is only available on this pin for select EC modes.
2: Pins can be used as digital inputs when USB is not enabled.
3: This bit field is contained in the OSCCON register.
4: This bit field is contained in the OSCTRM register.
5: USB PLL UF_{IN} requirements: 4 MHz $\leq UF_{IN} \leq$ 5 MHz.
6: This bit field is contained in the DEVCFG2 register.
7: A 48 MHz clock is required for proper USB operation.
8: Pins can be used as GPIO when the USB OTG module is disabled.
9: Pin is pulled high internally when USB OTG module is enabled.

The USB OTG module contains analog and digital components to provide a USB 2.0 full-speed and low-speed embedded host, full-speed device, or On-The-Go (OTG) implementation with a minimum of external components. This module in Host mode is intended for use as an embedded host and therefore does not implement a Universal Host Controller Interface (UHCI) or a Open Host Controller Interface (OHCI).

The USB OTG module consists of the clock generator, the USB voltage comparators, the transceiver, the Serial Interface Engine (SIE), a dedicated USB Bus Master, pull-up and pull-down resistors and the register interface.

The clock generator provides the 48 MHz clock, which is required for USB full-speed and low-speed communication. The voltage comparators monitor the voltage on the VBUS pin to determine the state of the bus. The transceiver provides the analog translation between the USB bus and the digital logic. The SIE is a state machine that transfers data to and from the endpoint buffers, and generates the hardware protocol for data transfers. The USB Bus Master transfers data between the data buffers in RAM and the SIE. The integrated pull-up and pull-down resistors eliminate the need for external signaling components. The register interface allows the CPU to configure and communicate with the module.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the USB module.

Library Interface Section	Description
USB Setup Functions	This section provides functions to perform general USB peripheral setup such as functions to set USB Speed, control on-chip pull ups, etc.
Buffer Descriptor Table Functions	This section provides functions that allow the application to setup, configure and access/modify the Buffer Descriptors in the Buffer Descriptor Table.
USB Activity Functions	This section provides function that allow the application to monitor bus conditions on the USB.
USB Bus Signaling Functions	This section provides functions that allow the application to generate Reset and Resume Signaling.
Last Transaction Status Functions	This section provides functions that allow the application to query conditions of the USB peripheral.
Endpoints Functions	This section provides functions that allow the application to manage endpoints.
Interrupts Functions	This section provides functions that allow the application to enable, disable and query the status interrupts in the USB peripheral.
Host Functions	This section provides functions that are required to operate the USB module while in Host mode.
On-The-Go (OTG) Functions	This section provides functions that are required to operate the USB module while in OTG mode.
External Transceiver Support Functions	This section provides function that are required to operate to enable/disable the external transceiver interface.

VBUS Support Functions	This section provides functions that allow VBUS level monitoring and VBUS boost PWM module control.
Test Support Functions	This section provides functions that enable/disable test signals needed for eye pattern measurement.

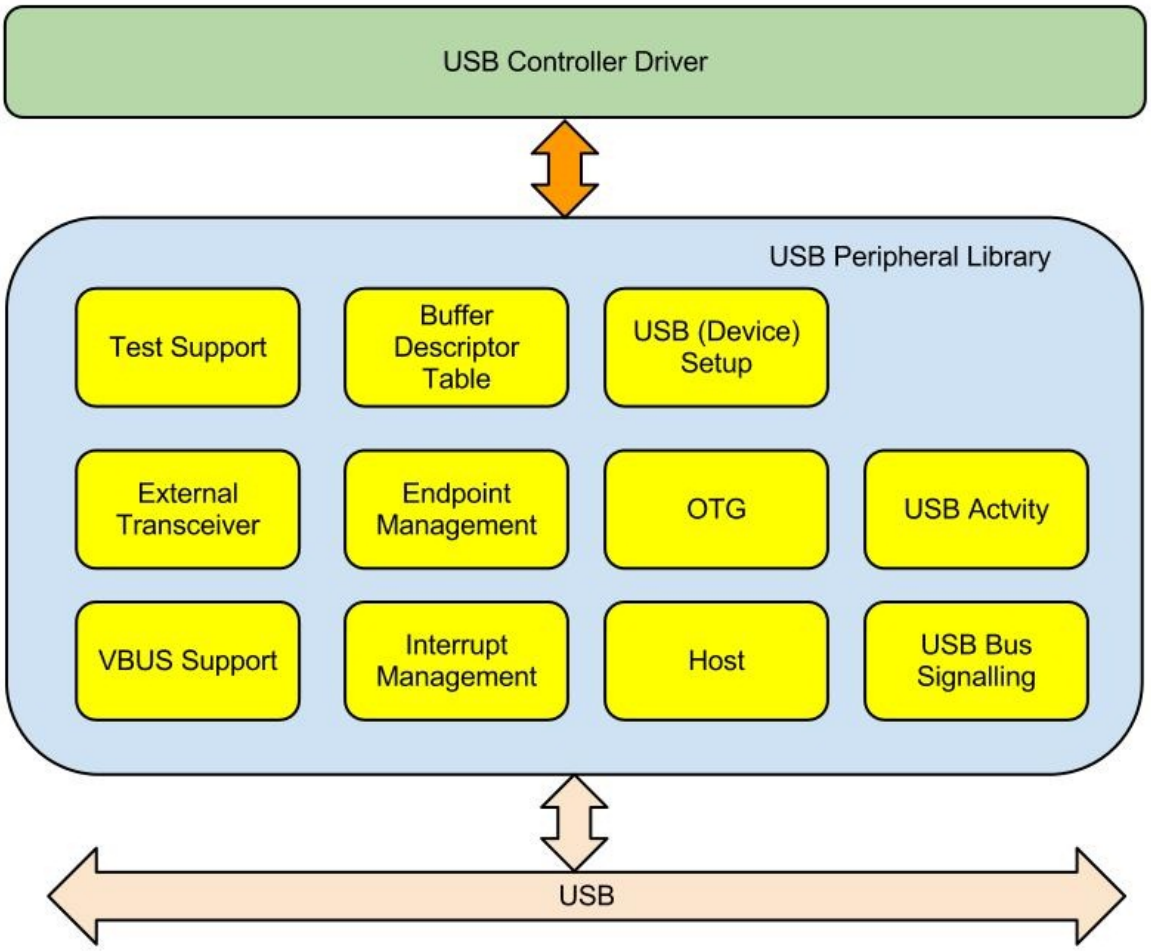
How the Library Works

Provides information on how the library works.

Description

The following figure describes the abstraction model employed by the USB Peripheral Library.

USB Peripheral Library Abstraction Model



USB Buffers and the Buffer Descriptor Table (BDT)

This section describes USB buffers and the Buffer Descriptor Table.

Description

All USB endpoints are implemented using buffers and control bits in RAM. Both software and the USB module have access to these buffers and control bits. To arbitrate this access a semaphore flag system is used.

Each endpoint can be configured for transmit only, receive only or transmit *and* receive. Transmit and receive functions have separate buffers. For each buffer there is a Buffer Descriptor (BD) in the Buffer Descriptor Table (BDT). On most devices the BDT has room for two transmit and two receive buffers for each endpoint, supporting ping-pong buffering. Buffer descriptors must be aligned on 512 byte boundaries to enable the USB module to correctly read each entry.

The starting address of the Buffer Descriptor Table is set by the PLIB_USB_BDTBaseAddressSet.

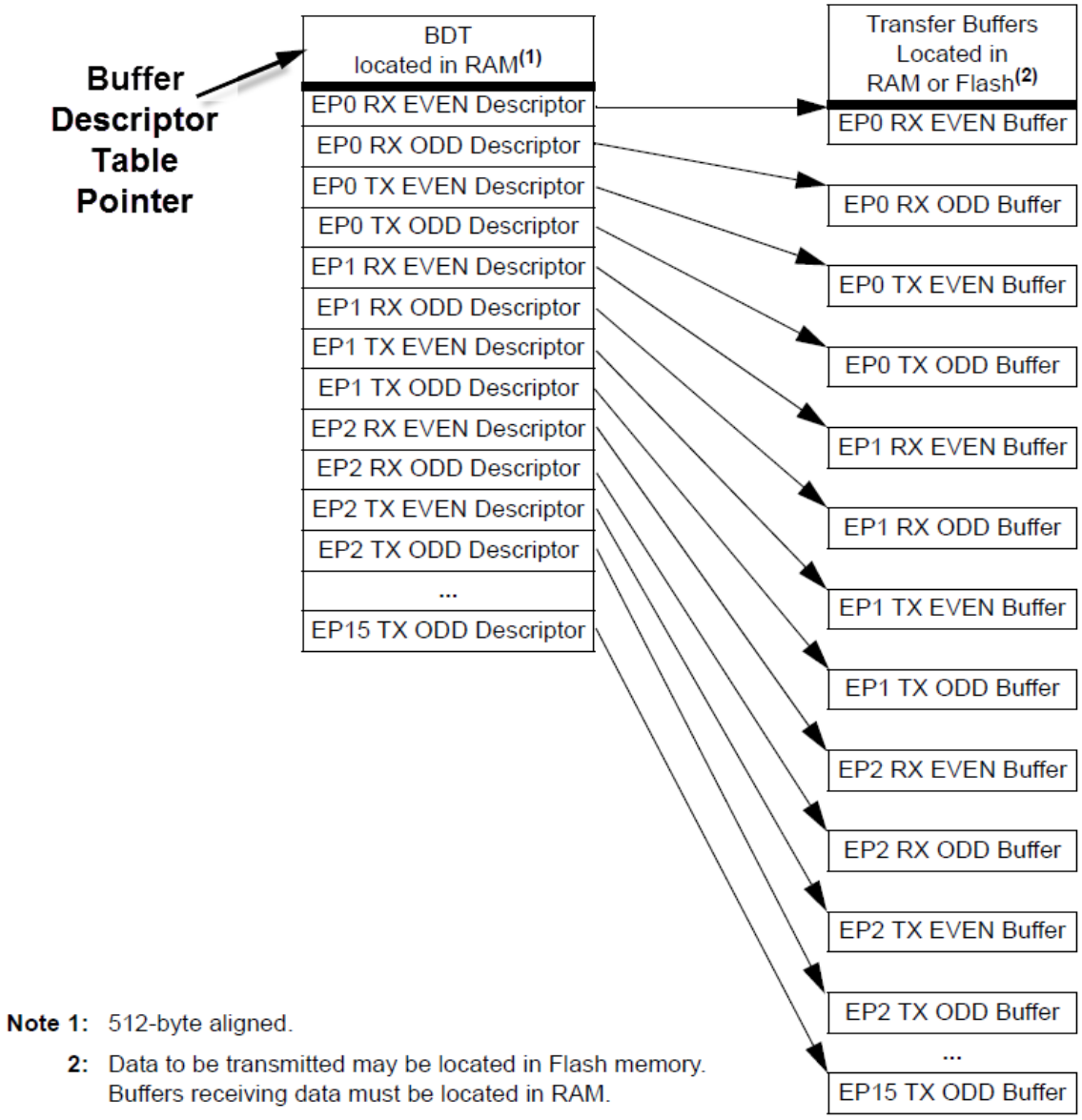
```
#define USB_MAX_EP_NUMBER 15
```

```
#define BDT_NUM_ENTRIES      (((USB_MAX_EP_NUMBER + 1) * 4)-2)

volatile union USB_BDT_ENTRY_TAG gBDT[BDT_NUM_ENTRIES] __attribute__ (( aligned (512) ));
```

Each Buffer Descriptor entry in the BDT has a **bufferAddress** pointer to the Buffer Descriptor's buffer in memory (RAM or FLASH).

The following figure shows the structure for PIC32 devices. These devices support ping-pong buffering for all endpoints.

Buffer Descriptor Table and Endpoint Buffers



Note 1: 512-byte aligned.

2: Data to be transmitted may be located in Flash memory. Buffers receiving data must be located in RAM.

With 16 endpoints (Endpoints 0 - 15) there are 64 possible entries in the Buffer Descriptor Table:

```
//                                          (Even,Odd) or (Ping,Pong)
//                                             (RX,TX) |
//                           (EP0      to       EPn)  |  |
volatile USB_BDT_ENTRY myBDT[(USB_MAX_EP_NUMBER + 1)][2][2] __attribute__ (( aligned (512) ));
```

Peripheral library functions are provided to support reading each BD while hiding the details of BDT addressing that can change when ping-pong buffering is disabled for one or more endpoints. The format of Buffer Descriptors vary from family to family and it is not important to know the differences, since peripheral library functions hide these details. To understand the information stored in each Buffer Descriptor Table entry it is informative to look at the C language description for one family of devices. (All device families support the same information but with different ways of packing the data into memory.)

For PIC32 devices each BD is 8 bytes long and is described by this C code:

```
typedef union _USB_BDT_ENTRY __attribute__ ((packed))
{
    uint64_t          dlValue;        //Double Long Integer Value
    uint32_t           lValue[2];     //Long  Integer Values
    uint16_t           sValue[4];     //Short Integer Values
```

```
        struct __attribute__ ((packed))
        {
            USB_BD_STATUS  bufferStatus;  //(RW) Buffer Status
            uint16_t       byteCount:10;  //(RW) Byte Count
            uint8_t                : 6;  //(  ) Reserved
            uint32_t       bufferAddress; //(RW) Buffer Address in Data Ram or Flash
        };
    } USB_BDT_ENTRY;
```

For transmit buffers, byteCount, is set to the length of the buffer using PLIB_USB_BufferByteCountSet. For receive buffers PLIB_USB_BufferByteCountSet is used to set the maximum allowable receive data length. PLIB_USB_BufferByteCountGet is used to determine how many bytes were actually transmitted or received. The memory address of each buffer is found in bufferAddress. The functions PLIB_USB_BufferAddressSet and PLIB_USB_BufferAddressGet support this field in the Buffer Descriptor.

Buffer status is stored in the 16 least significant bits of each BDT entry:

```
    typedef union _USB_BD_STATUS __attribute__ ((packed))
    {
        uint16_t       sValue;                  //Short Integer Value
        struct __attribute__ ((packed))
        {
            uint8_t                          :2; //(  )Reserved
            uint8_t    stallEnable           :1; //( W) Buffer Stall Enable
            uint8_t    dataToggleSyncEnable :1; //( W) Data Toggle Synch Enable
            uint8_t                          :1; //(  ) Reserved
            uint8_t                          :1; //(  ) Reserved
            uint8_t    dataToggle            :1; //(RW) Data Toggle Synch Value
            uint8_t    uSBOwnsBuffer         :1; //(RW) USB Ownership of buffer and BDT entry
        };
        struct __attribute__ ((packed))
        {
            uint8_t                          :2; //(  ) Reserved
            uint8_t    packetID              :4; //(R ) Packet Identifier (PID)
        };
    } USB_BD_STATUS;
```

The fields stallEnable and dataToggleSyncEnable in the Buffer Descriptor status structure are writeable but not readable. The function PLIB_USB_BufferStallEnable sets the stallEnable bit. (Hardware will clear the bit after receiving a SETUP token from the host.) The functions PLIB_USB_BufferDataToggleSyncEnable and PLIB_USB_BufferDataToggleSyncDisable manipulate dataToggleSyncEnable .

On a read from the status structure stallEnable and dataToggleSyncEnable and the next two (reserved) bits are replaced by the Packet Identifier (PID), called packetID in the structure. The function PLIB_USB_BufferPIDGet reads packetID.

The rest of the status structure is writeable and readable: dataToggle and uSBOwnsBuffer.

The dataToggle (DATA0 or DATA1) of a receive buffer is determined by using PLIB_USB_BufferDataToggleGet. For transmit buffers the dataToggle value is set using PLIB_USB_BufferDataToggleSelect .

The field uSBOwnsBuffer is used by both software and the USB module as a semaphore. The function PLIB_USB_BufferReleaseToUSB releases the buffer to the USB module and the function PLIB_USB_BufferReleasedToSW returns a Boolean true when the buffer is released back to software by the USB module. As long as PLIB_USB_BufferReleasedToSW returns false, software should not change the Buffer Descriptor Table entry or its associated buffer.

USB Setup Example

This section provides example code for setting up the USB module.

Description

Following is an example of how to set up the USB module:

```
#define USB_MAX_EP_NUMBER 15

#define BDT_NUM_ENTRIES      (((USB_MAX_EP_NUMBER + 1) * 4)-2)

volatile union USB_BDT_ENTRY_TAG gBDT[BDT_NUM_ENTRIES] __attribute__ (( aligned (512) ));

unsigned int bufferTransactionCount[BDT_NUM_ENTRIES];

    // Turn off USB module
    PLIB_USB_Disable( usbID );

    // Set up the Hardware
    if ( usbModuleSetup.StopInIdle )
    {
        PLIB_USB_StopInIdleEnable( usbID );
```

```
    }
    else
    {
        PLIB_USB_StopInIdleDisable( usbID );
    }

    if ( usbModuleSetup.SuspendInSleep )
    {
        PLIB_USB_AutoSuspendEnable( usbID );
    }
    else
    {
        PLIB_USB_AutoSuspendDisable( usbID );
    }

    PLIB_USB_OperatingModeSelect( usbID, usbModuleSetup.OpMode );

    PLIB_USB_PingPongModeSelect( usbID, usbModuleSetup.ppMode );

    // Reset all ping pong buffers to "Even"
    PLIB_USB_PingPongFreeze( usbID );
    PLIB_USB_PingPongUnfreeze( usbID );

    // Interrupt flag cleared on the safer side
    IFS1bits.USBIF = 0;

    // Enable USB module interrupts
    PLIB_USB_InterruptEnable( usbID, DRV_USB_GEN_INT_ENABLES );

    // Disable all OTG interrupts
    PLIB_USB_OTG_InterruptDisable( usbID, DRV_USB_OTB_INT_ENABLES );

    // Enable all Error interrupts
    PLIB_USB_ErrorInterruptEnable( usbID, DRV_USB_ERR_INT_ENABLES );

    // Enable the interrupt source in case of interrupt mode
    IEC1bits.USBIE = 1;

    // Setting the Interrupt Priority in case of interrupt mode.
    IPC11bits.USBIP = 4;

    // Initialize BDT
    for ( iEntry = 0; iEntry < BDT_NUM_ENTRIES; iEntry++ )
    {
        gBDT[iEntry].lValue[0] = 0ul;
        gBDT[iEntry].lValue[1] = 0ul;

        bufferTransactionCount[iEntry] = 0;
    }//end for ( iEntry = 0; iEntry < BDT_NUM_ENTRIES; iEntry++ )

    // Inform USB module of BDT's address in memory
    PLIB_USB_BDTBaseAddressSet( usbID , (void *)((uint32_t)KVA_TO_PA(gBDT)) );

    /***********************/
    /* SET UP ENDPOINT ZERO */
    /***********************/
    PLIB_USB_BufferAddressSet( usbID, (void*)gBDT, PLIB_USB_PingPongModeGet( usbID ),
                               0, USB_BUFFER_RX, USB_BUFFER_EVEN, pSetupRcvBuffer );

    // Configure Endpoint Zero control register
    PLIB_USB_EP0LSDirectConnectDisable( usbID );   // For Hosts, included for completeness
    PLIB_USB_EP0NakRetryEnable( usbID );           // For Hosts, included for completeness
    PLIB_USB_EPnControlTransferEnable( usbID, 0 ); // Enable control transfers
    PLIB_USB_EPnRxSelect( usbID, 0, USB_EP_RX );   // Enable receive
    PLIB_USB_EPnHandshakeEnable( usbID, 0 );       // Enable handshaking

    // Set up Endpoint Zero's receive buffer BDT entries
    ppMode = PLIB_USB_PingPongModeGet( usbID ); // Ping Pong Mode needed for BDT entry indexing
```

```
    // Clear PID bits
    PLIB_USB_BufferPIDBitsClear(usbID,(void*)gBDT,ppMode,0,USB_BUFFER_RX,USB_BUFFER_EVEN); //Rx0

    // Disable buffer stall
    PLIB_USB_BufferStallDisable(usbID,(void*)gBDT,ppMode,0,USB_BUFFER_RX,USB_BUFFER_EVEN); //Rx0

    // Enable Data Toggle Synchronization
    PLIB_USB_BufferDataToggleSyncEnable(usbID,(void*)gBDT,ppMode,0,USB_BUFFER_RX,USB_BUFFER_EVEN); //Rx0

    // Set Data0/1 to Data0
    PLIB_USB_BufferDataToggleSelect(usbID,(void*)gBDT,ppMode,0,USB_BUFFER_RX,USB_BUFFER_EVEN,USB_BUFFER_DATA
0);
//Rx0

    // Setup packets have 8 bytes of data payload
    PLIB_USB_BufferByteCountSet(usbID,(void*)gBDT,ppMode,0,USB_BUFFER_RX,USB_BUFFER_EVEN,8);

    // Release buffers to USB module
    PLIB_USB_BufferReleaseToUSB(usbID,(void*)gBDT,ppMode,0,USB_BUFFER_RX,USB_BUFFER_EVEN);

    PLIB_USB_Enable( gDrvUSBObj[hDriver].usbID ); // Turn on USB module
```

Configuring the Library

The library is configured for the supported USB module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) USB Setup Functions

	Name	Description
	PLIB_USB_AllInterruptEnable	Configures the USB peripheral general interrupts, error interrupts and OTG interrupts.
	PLIB_USB_AutoSuspendDisable	Disables USB OTG Auto-suspend mode.
	PLIB_USB_AutoSuspendEnable	Enables USB Auto-suspend mode.
	PLIB_USB_DeviceAddressGet	Returns the address of the USB module in Device mode.
	PLIB_USB_DeviceAddressSet	Sets the USB Device's address.
	PLIB_USB_Disable	Disables (powers down) the USB module.
	PLIB_USB_Enable	Enables (powers up) the USB module.
	PLIB_USB_FullSpeedDisable	Forces the USB module to operate at low speed.
	PLIB_USB_FullSpeedEnable	Enables the USB to operate at full speed.
	PLIB_USB_OnChipPullUpDisable	Disables on-chip pull-ups.
	PLIB_USB_OnChipPullUpEnable	Enables on-chip pull-ups.
	PLIB_USB_OperatingModeSelect	Selects the operating mode of the USB module.
	PLIB_USB_PingPongModeGet	Returns the Ping-Pong Configuration setting.
	PLIB_USB_PingPongModeSelect	Selects the Ping-Pong Configuration setting.
	PLIB_USB_SleepGuardDisable	This function disables Sleep Guard. Entry into Sleep mode is immediate.
	PLIB_USB_SleepGuardEnable	Entry into Sleep mode is blocked if bus activity is detected or if an interrupt is pending.
	PLIB_USB_StopInIdleDisable	Allows the USB module to continue operation when the device enters Idle mode.
	PLIB_USB_StopInIdleEnable	Enables USB module operation to stop when the device enters Idle mode.
	PLIB_USB_SuspendDisable	Disables USB OTG Suspend mode.
	PLIB_USB_SuspendEnable	Enables USB Suspend mode.
	PLIB_USB_UOEMonitorDisable	Disables the OE signal output.
	PLIB_USB_UOEMonitorEnable	Enables the OE signal output.

b) Buffer Descriptor Table Functions

	Name	Description
	PLIB_USB_BDTBaseAddressGet	Returns the base address of the Buffer Descriptor Table.
	PLIB_USB_BDTBaseAddressSet	Sets the base address for the Buffer Descriptor Table for PIC32 devices.
	PLIB_USB_BufferAddressGet	Gets the memory address of an endpoint buffer.
	PLIB_USB_BufferAddressSet	Sets the endpoint buffer address.

	PLIB_USB_BufferAllCancelReleaseToUSB	Cancels all endpoint buffer releases to the USB module and hands over the buffer to the CPU.
	PLIB_USB_BufferByteCountGet	Returns the endpoint buffer byte count.
	PLIB_USB_BufferByteCountSet	Sets the buffer byte count.
	PLIB_USB_BufferCancelReleaseToUSB	Cancels release of the endpoint buffer by software, allowing software to again access the buffer.
	PLIB_USB_BufferClearAll	Clears (zeros out) entries in the Buffer Descriptor Table.
	PLIB_USB_BufferClearAllDTSEnable	Clears the endpoint descriptor entry and enables data toggle synchronization.
	PLIB_USB_BufferDataToggleGet	Returns data synchronization (DATA0 or DATA1) for the endpoint buffer.
	PLIB_USB_BufferDataToggleSelect	Sets the endpoint buffer to DATA0 or DATA1.
	PLIB_USB_BufferDataToggleSyncDisable	Disables DATA0/DATA1 synchronization between the device and host.
	PLIB_USB_BufferDataToggleSyncEnable	Enables DATA0/DATA1 synchronization between the device and host.
	PLIB_USB_BufferEP0RxStatusInitialize	Initializes the Endpoint 0 RX endpoint buffer descriptors.
	PLIB_USB_BufferIndexGet	Gets the Buffer Descriptor Table index for a buffer.
	PLIB_USB_BufferPIDBitsClear	Clears the Buffer Status bits in the Buffer Descriptor Table.
	PLIB_USB_BufferPIDGet	Returns the token packet ID (PID) from the endpoint buffer status.
	PLIB_USB_BufferReleasedToSW	Returns the boolean flag value of 'true' when the buffer has been released by the USB module.
	PLIB_USB_BufferReleaseToUSB	Releases the endpoint buffer by software, allowing the USB module access to the buffer.
	PLIB_USB_BufferSchedule	Hands over a buffer to the USB module along with the buffer address and byte count.
	PLIB_USB_BufferStallDisable	Disables STALL handshaking for the associated endpoint buffer.
	PLIB_USB_BufferStallEnable	Enables STALL handshaking for the associated endpoint buffer.
	PLIB_USB_BufferStallGet	Returns the buffer stall status for an endpoint/direction/ping-pong.

c) Endpoints Functions

	Name	Description
	PLIB_USB_EP0HostSetup	Sends token to the specified address.
	PLIB_USB_EP0LSDirectConnectDisable	Disables direct connection to a low-speed device for Endpoint 0.
	PLIB_USB_EP0LSDirectConnectEnable	Enables direct connection to a low-speed device for Endpoint 0.
	PLIB_USB_EP0NakRetryDisable	Disables retrying of NAKed transactions.
	PLIB_USB_EP0NakRetryEnable	Enables retrying NAK'd transactions for Endpoint 0.
	PLIB_USB_EPnAttributesClear	Clears the set attributes of the specified endpoint.
	PLIB_USB_EPnAttributesSet	Configures attributes of the endpoint such as direction, handshake capability and direction.
	PLIB_USB_EPnControlTransferDisable	Disables endpoint control transfers.
	PLIB_USB_EPnControlTransferEnable	Enables endpoint control transfers.
	PLIB_USB_EPnDirectionDisable	Disables the specified endpoint direction.
	PLIB_USB_EPnHandshakeDisable	Disables endpoint handshaking.
	PLIB_USB_EPnHandshakeEnable	Enables endpoint handshaking.
	PLIB_USB_EPnIsStalled	Tests whether the endpoint epValue is stalled.
	PLIB_USB_EPnRxDisable	Disables an endpoint's ability to process IN tokens.
	PLIB_USB_EPnRxEnable	Enables an endpoint to process IN tokens.
	PLIB_USB_EPnRxSelect	Selects receive capabilities of an endpoint.
	PLIB_USB_EPnStallClear	Clears an endpoint's stalled flag.
	PLIB_USB_EPnTxDisable	Disables an endpoint's ability to process OUT tokens.
	PLIB_USB_EPnTxEnable	Enables an endpoint to process OUT tokens.
	PLIB_USB_EPnTxRxSelect	Selects transmit and/or receive capabilities of an endpoint.
	PLIB_USB_EPnTxSelect	Selects transmit capabilities of an endpoint.

d) Interrupts Functions

	Name	Description
	PLIB_USB_InterruptDisable	Disables a general interrupt for the USB module.
	PLIB_USB_InterruptEnable	Enables a general interrupt for the USB module.
	PLIB_USB_InterruptEnableGet	Returns the enable/disable status of general USB module interrupts
	PLIB_USB_InterruptFlagAllGet	Returns a logically ORed bit map of active general USB interrupt flags.

	PLIB_USB_InterruptFlagClear	Clears a general interrupt flag for the USB module.
	PLIB_USB_InterruptFlagGet	Tests a general interrupt flag for the USB module.
	PLIB_USB_InterruptFlagSet	Sets a general interrupt flag for the USB module.
	PLIB_USB_InterruptIsEnabled	Returns true if interrupts are enabled.

e) Error Interrupts Functions

	Name	Description
	PLIB_USB_ErrorInterruptDisable	Disables an error interrupt for the USB module.
	PLIB_USB_ErrorInterruptEnable	Enables an error interrupt for the USB module.
	PLIB_USB_ErrorInterruptFlagAllGet	Returns a logically ORed bit map of active error interrupt flags.
	PLIB_USB_ErrorInterruptFlagClear	Clears an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptFlagGet	Tests an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptFlagSet	Sets an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptIsEnabled	Returns true if interrupts are enabled.

f) Last Transaction Status Functions

	Name	Description
	PLIB_USB_LastTransactionDetailsGet	Returns the details of the last completed transaction.
	PLIB_USB_LastTransactionDirectionGet	Indicates the direction of the last transaction.
	PLIB_USB_LastTransactionEndPtGet	Returns the endpoint number of the last USB transfer.
	PLIB_USB_LastTransactionPingPongStateGet	Indicates whether the last transaction was to an EVEN buffer or an ODD buffer.

g) Host Functions

	Name	Description
	PLIB_USB_IsBusyWithToken	Indicates whether there is a token being executed by the USB module as Host.
	PLIB_USB_SOFDisable	Disables the automatic generation of the SOF token.
	PLIB_USB_SOFEnable	Enables the automatic generation of the SOF token every 1 ms.
	PLIB_USB_SOFThresholdGet	Returns the Start-of-Frame (SOF) Count bits.
	PLIB_USB_SOFThresholdSet	Sets the Start-of-Frame (SOF) threshold value.
	PLIB_USB_TokenEPGet	Returns the specified Endpoint address.
	PLIB_USB_TokenEPSet	Sets the Endpoint address for a host transaction.
	PLIB_USB_TokenPIDGet	Returns the token transaction type.
	PLIB_USB_TokenPIDSet	Sets the token transaction type to pidValue.
	PLIB_USB_TokenSpeedSelect	Selects low speed or full speed for subsequent token executions.

h) USB Bus Signaling Functions

	Name	Description
	PLIB_USB_ResetSignalDisable	Disables reset signaling on the USB bus.
	PLIB_USB_ResetSignalEnable	Enables reset signaling on the USB bus.
	PLIB_USB_ResumeSignalingDisable	Disables resume signaling.
	PLIB_USB_ResumeSignalingEnable	Enables resume signaling.

i) On-The-Go (OTG) Functions

	Name	Description
	PLIB_USB_OTG_BSessionHasEnded	Returns the status of the B-Session End Indicator bit.
	PLIB_USB_OTG_IDPinStateIsTypeA	Returns the ID Pin state.
	PLIB_USB_OTG_LineStateIsStable	Returns the status of the Line Stable Indicator bit.
	PLIB_USB_OTG_PullUpPullDownSetup	Enables or disables pull-up and pull-down resistors.
	PLIB_USB_OTG_SessionValid	Returns the status of the Session Valid Indicator bit.
	PLIB_USB_OTG_VBusChargeDisable	Disables VBUS line charge.
	PLIB_USB_OTG_VBusChargeEnable	Enables the VBUS line to be charged through a pull-up resistor.
	PLIB_USB_OTG_VBusChargeTo3V	Sets the VBUS line to charge to 3.3V.
	PLIB_USB_OTG_VBusChargeTo5V	Sets the VBUS line to charge to 5V.
	PLIB_USB_OTG_VBusDischargeDisable	Disables VBUS line discharge.
	PLIB_USB_OTG_VBusDischargeEnable	Enables VBUS line to be discharged through a resistor.

	PLIB_USB_OTG_VBusPowerOff	Turns off power on the VBUS Line.
	PLIB_USB_OTG_VBusPowerOn	Turns on power for the VBUS line.
	PLIB_USB_OTG_VBusValid	Returns the status of the A-VBUS valid indicator.

j) OTG Interrupts Functions

	Name	Description
	PLIB_USB_OTG_InterruptDisable	Disables a USB On-The-Go (OTG) Interrupt for the USB module.
	PLIB_USB_OTG_InterruptEnable	Enables a USB On-The-Go (OTG) Interrupt for the USB module.
	PLIB_USB_OTG_InterruptFlagClear	Clears a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptFlagGet	Tests a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptFlagSet	Sets a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptIsEnabled	Returns whether or not interrupts are enabled.

k) USB Activity Functions

	Name	Description
	PLIB_USB_ActivityPending	Returns whether or not USB activity is pending.
	PLIB_USB_FrameNumberGet	Returns the USB frame number.
	PLIB_USB_JStateIsActive	Live differential receiver J State flag.
	PLIB_USB_PacketTransferDisable	Disables the Serial Interface Engine (SIE).
	PLIB_USB_PacketTransferEnable	Re-enables the Serial Interface Engine (SIE), allowing token and packet processing.
	PLIB_USB_PacketTransferIsDisabled	Indicates that a setup token has been received from the Host and that token/packet processing is disabled.
	PLIB_USB_SE0InProgress	Returns whether a single-ended zero event is in progress.

l) External Transceiver Support Functions

	Name	Description
	PLIB_USB_I2CInterfaceForExtModuleDisable	Specifies external module(s) are controlled via dedicated pins.
	PLIB_USB_I2CInterfaceForExtModuleEnable	Specifies external module(s) are controlled via the I2C interface.
	PLIB_USB_TransceiverDisable	Disables the on-chip transceiver
	PLIB_USB_TransceiverEnable	Enables the on-chip transceiver.

m) VBUS Support Functions

	Name	Description
	PLIB_USB_ExternalComparatorMode2Pin	Sets the 2-pin input configuration for VBUS comparators.
	PLIB_USB_ExternalComparatorMode3Pin	Sets the 3-pin input configuration for VBUS Comparators.
	PLIB_USB_PWMCounterDisable	Disables the PWM counter used to generate the VBUS for the USB module.
	PLIB_USB_PWMCounterEnable	Enables the PWM counter used to generate the VBUS for the USB module.
	PLIB_USB_PWMDisable	Disables the PWM Generator.
	PLIB_USB_PWMEnable	Enables the PWM Generator.
	PLIB_USB_PWMPolaritiyActiveLow	Sets the PWM output to active-high and resets low.
	PLIB_USB_PWMPolarityActiveHigh	Sets the PWM output to active-low and resets high.
	PLIB_USB_VBoostDisable	Disables the On-Chip 5V Boost Regulator Circuit Disabled bit.
	PLIB_USB_VBoostEnable	Enables the On-Chip 5V Boost Regulator Circuit Enabled bit.
	PLIB_USB_VBUSComparatorDisable	Disables the on-chip VBUS Comparator.
	PLIB_USB_VBUSComparatorEnable	Enables the on-chip VBUS Comparator.
	PLIB_USB_VBUSPullUpDisable	Disables the pull-up on the VBUS pin.
	PLIB_USB_VBUSPullUpEnable	Enables the pull-up on the VBUS pin.

n) Test Support Functions

	Name	Description
	PLIB_USB_EyePatternDisable	Disables the USB eye pattern test.
	PLIB_USB_EyePatternEnable	Enables USB eye pattern test.

o) Other Functions

	Name	Description
	PLIB_USB_ModuleIsBusy	Indicates if the USB module is not ready to be enabled.

	PLIB_USB_PingPongFreeze	Resets all Ping-Pong buffer pointers to even buffers.
	PLIB_USB_PingPongReset	Resets the USB peripheral internal Ping-Pong indicator to point to even buffers.
	PLIB_USB_PingPongUnfreeze	Enables Ping-Pong buffering.
	PLIB_USB_TokenSend	Sends token to the specified address.

p) Feature Existence Functions

	Name	Description
	PLIB_USB_ExistsActivityPending	Identifies whether the ActivityPending feature exists on the USB module.
	PLIB_USB_ExistsALL_Interrupt	Identifies whether the ALL_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsAutomaticSuspend	Identifies whether the AutomaticSuspend feature exists on the USB module.
	PLIB_USB_ExistsBDTBaseAddress	Identifies whether the BDTBaseAddress feature exists on the USB module.
	PLIB_USB_ExistsBDTFunctions	Identifies whether the BDTFunctions feature exists on the USB module.
	PLIB_USB_ExistsBufferFreeze	Identifies whether the BufferFreeze feature exists on the USB module.
	PLIB_USB_ExistsDeviceAddress	Identifies whether the DeviceAddress feature exists on the USB module.
	PLIB_USB_ExistsEP0LowSpeedConnect	Identifies whether the EP0LowSpeedConnect feature exists on the USB module.
	PLIB_USB_ExistsEP0NAKRetry	Identifies whether the EP0NAKRetry feature exists on the USB module.
	PLIB_USB_ExistsEPnRxEnable	Identifies whether the EPnRxEnableEnhanced feature exists on the USB module.
	PLIB_USB_ExistsEPnTxRx	Identifies whether the EPnTxRx feature exists on the USB module.
	PLIB_USB_ExistsERR_Interrupt	Identifies whether the ERR_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsERR_InterruptStatus	Identifies whether the ERR_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsEyePattern	Identifies whether the EyePattern feature exists on the USB module.
	PLIB_USB_ExistsFrameNumber	Identifies whether the FrameNumber feature exists on the USB module.
	PLIB_USB_ExistsGEN_Interrupt	Identifies whether the GEN_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsGEN_InterruptStatus	Identifies whether the GEN_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsHostBusyWithToken	Identifies whether the HostBusyWithToken feature exists on the USB module.
	PLIB_USB_ExistsHostGeneratesReset	Identifies whether the HostGeneratesReset feature exists on the USB module.
	PLIB_USB_ExistsLastDirection	Identifies whether the LastDirection feature exists on the USB module.
	PLIB_USB_ExistsLastEndpoint	Identifies whether the LastEndpoint feature exists on the USB module.
	PLIB_USB_ExistsLastPingPong	Identifies whether the LastPingPong feature exists on the USB module.
	PLIB_USB_ExistsLastTransactionDetails	Identifies whether the LastTransactionDetails feature exists on the USB module.
	PLIB_USB_ExistsLiveJState	Identifies whether the LiveJState feature exists on the USB module.
	PLIB_USB_ExistsLiveSingleEndedZero	Identifies whether the LiveSingleEndedZero feature exists on the USB module.
	PLIB_USB_ExistsModuleBusy	Identifies whether the ModuleBusy feature exists on the USB module.
	PLIB_USB_ExistsModulePower	Identifies whether the ModulePower feature exists on the USB module.
	PLIB_USB_ExistsNextTokenSpeed	Identifies whether the NextTokenSpeed feature exists on the USB module.
	PLIB_USB_ExistsOnChipPullup	Identifies whether the OnChipPullup feature exists on the USB module.
	PLIB_USB_ExistsOnChipTransceiver	Identifies whether the OnChipTransceiver feature exists on the USB module.
	PLIB_USB_ExistsOpModeSelect	Identifies whether the OpModeSelect feature exists on the USB module.
	PLIB_USB_ExistsOTG_ASessionValid	Identifies whether the OTG_ASessionValid feature exists on the USB module.
	PLIB_USB_ExistsOTG_BSessionEnd	Identifies whether the OTG_BSessionEnd feature exists on the USB module.
	PLIB_USB_ExistsOTG_IDPinState	Identifies whether the OTG_IDPinState feature exists on the USB module.
	PLIB_USB_ExistsOTG_Interrupt	Identifies whether the OTG_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsOTG_InterruptStatus	Identifies whether the OTG_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsOTG_LineState	Identifies whether the OTG_LineState feature exists on the USB module.
	PLIB_USB_ExistsOTG_PullUpPullDown	Identifies whether the OTG_PullUpPullDown feature exists on the USB module.
	PLIB_USB_ExistsOTG_SessionValid	Identifies whether the OTG_SessionValid feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusCharge	Identifies whether the OTG_VbusCharge feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusDischarge	Identifies whether the OTG_VbusDischarge feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusPowerOnOff	Identifies whether the OTG_VbusPowerOnOff feature exists on the USB module.
	PLIB_USB_ExistsPacketTransfer	Identifies whether the PacketTransfer feature exists on the USB module.
	PLIB_USB_ExistsPingPongMode	Identifies whether the PingPongMode feature exists on the USB module.
	PLIB_USB_ExistsResumeSignaling	Identifies whether the ResumeSignaling feature exists on the USB module.
	PLIB_USB_ExistsSleepEntryGuard	Identifies whether the SleepEntryGuard feature exists on the USB module.
	PLIB_USB_ExistsSOFThreshold	Identifies whether the SOFThreshold feature exists on the USB module.
	PLIB_USB_ExistsSpeedControl	Identifies whether the SpeedControl feature exists on the USB module.

	PLIB_USB_ExistsStartOfFrames	Identifies whether the StartOfFrames feature exists on the USB module.
	PLIB_USB_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the USB module.
	PLIB_USB_ExistsSuspend	Identifies whether the Suspend feature exists on the USB module.
	PLIB_USB_ExistsTokenEP	Identifies whether the TokenEP feature exists on the USB module.
	PLIB_USB_ExistsTokenPID	Identifies whether the TokenPID feature exists on the USB module.
	PLIB_USB_ExistsUOEMonitor	Identifies whether the UOEMonitor feature exists on the USB module.

q) Data Types and Constants

	Name	Description
	USB_BUFFER_DATA01	Provides enumeration data toggle for a buffer.
	USB_BUFFER_DIRECTION	Provides enumeration transmit/receive direction for a buffer.
	USB_BUFFER_PING_PONG	Enumerates the ping-pong buffer (Even vs. Odd).
	USB_BUFFER_SCHEDULE_DATA01	Provides enumeration data toggle for a buffer.
	USB_EP_TXRX	Provides enumeration transmit/receive setup for an endpoint.
	USB_OPMODES	Provides enumeration of operating modes supported by USB.
	USB_OTG_INTERRUPTS	Provides enumeration of interrupts related to the USB On-The-Go (OTG) module.
	USB_OTG_PULL_UP_PULL_DOWN	USB OTG pull-Up and pull-Down resistors for D+ and D- .
	USB_PID	Legal PID values.
	USB_PING_PONG_MODE	Supports the four modes of ping-pong buffering.
	USB_PING_PONG_STATE	Decodes which buffer (Even vs. Odd) was used for the last transaction.
	USB_TOKEN_SPEED	Provides enumeration of available token speeds.
	USB_MAX_EP_NUMBER	Maximum number of endpoints supported (not including EP0).

Description

This section describes the Application Programming Interface (API) functions of the USB Peripheral Library.

Refer to each section for a detailed description.

a) USB Setup Functions

PLIB_USB_AllInterruptEnable Function

Configures the USB peripheral general interrupts, error interrupts and OTG interrupts.

File

plib_usb.h

C

```
void PLIB_USB_AllInterruptEnable(USB_MODULE_ID index, USB_INTERRUPTS usbInterruptsFlag,
USB_ERROR_INTERRUPTS usbErrorInterruptsFlag, USB_OTG_INTERRUPTS otgInterruptFlag);
```

Returns

None.

Description

This function configures the USB peripheral general interrupts, error interrupts and OTG interrupts.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsALL_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// This code snippet disables all OTG interrupts, disables
// the SOF interrupt and enables all error interrupts.
USB_OTG_INTERRUPTS otgInterruptEnables = ~USB_OTG_INT_ALL ;
USB_INTERRUPTS generalInterruptEnables = USB_INT_ALL & ~USB_INT_SOF ;
```

```
USB_ERROR_INTERRUPTS errorInterruptEnables = USB_ERR_INT_ALL ;

PLIB_USB_AllInterruptEnable(USB_MODULE_ID index, USB_INTERRUPTS usbInterruptsFlag,
    USB_ERROR_INTERRUPTS usbErrorInterruptsFlag,
    USB_OTG_INTERRUPTS otgInterruptFlag);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
usbInterruptsFlag	General interrupts to be configured
usbErrorInterruptsFlag	USB Error interrupts to be configured
otgInterruptFlag	OTG interrupts to be configured

Function

void PLIB_USB_AllInterruptEnable(USB_MODULE_ID index, USB_INTERRUPTS usbInterruptsFlag,
USB_ERROR_INTERRUPTS usbErrorInterruptsFlag, USB_OTG_INTERRUPTS otgInterruptFlag);

PLIB_USB_AutoSuspendDisable Function

Disables USB OTG Auto-suspend mode.

File

plib_usb.h

C

```
void PLIB_USB_AutoSuspendDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables USB OTG Auto-suspend mode. The USB OTG module will operate normally and does not automatically suspend upon entry to Sleep mode. Software must use PLIB_USB_SuspendEnable to suspend the module, including the USB 48 MHz clock

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsAutomaticSuspend in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_AutoSuspendDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_AutoSuspendDisable (USB_MODULE_ID index)

PLIB_USB_AutoSuspendEnable Function

Enables USB Auto-suspend mode.

File

plib_usb.h

C

```
void PLIB_USB_AutoSuspendEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables USB Auto-suspend mode. The USB module automatically suspends upon entry to Sleep mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsAutomaticSuspend in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_AutoSuspendEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_AutoSuspendEnable (USB_MODULE_ID index)

PLIB_USB_DeviceAddressGet Function

Returns the address of the USB module in Device mode.

File

plib_usb.h

C

```
uint8_t PLIB_USB_DeviceAddressGet(USB_MODULE_ID index);
```

Returns

Device Address

Description

This function returns the address of the USB module in Device mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsDeviceAddressin your application to determine whether this feature is available.

Preconditions

None.

Example

```
myUSBAddress = PLIB_USB_DeviceAddressGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

uint8_t PLIB_USB_DeviceAddressGet (USB_MODULE_ID index)

PLIB_USB_DeviceAddressSet Function

Sets the USB Device's address.

File

plib_usb.h

C

```
void PLIB_USB_DeviceAddressSet(USB_MODULE_ID index, uint8_t address);
```

Returns

None.

Description

This function sets the USB Device's address as part of enumeration.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsDeviceAddressin your application to determine whether this feature is available.

Preconditions

USB module must be in Host mode.

Example

```
uint8_t myUSBAddress = ....;
PLIB_USB_DeviceAddressSet(MY_USB_INSTANCE, myUSBAddress);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
address	USB address

Function

void PLIB_USB_DeviceAddressSet (USB_MODULE_ID index, uint8_t address)

PLIB_USB_Disable Function

Disables (powers down) the USB module.

File

plib_usb.h

C

```
void PLIB_USB_Disable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables (powers down) the USB module.

Remarks

For PIC32 devices, the USB module must be in Device mode before the USB module is powered down.

For PIC32 devices, all reads or writes to module registers after powering down the module will be invalid until PLIB_USB_ModuleIsBusy (MY_USB_INSTANCE) == false.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsModulePower in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#if defined(__PIC32MX__)
  // Disable Host, Device, or OTG before powering down
  PLIB_USB_OperatingModeSelect( MY_USB_INSTANCE, USB_OPMODE_NONE );
```

```
    // Turn off USB
    PLIB_USB_Disable(MY_USB_INSTANCE);
    // For PIC32, wait until module is no longer busy before trying to
    // access any USB module registers.
    while ( PLIB_USB_ModuleIsBusy (MY_USB_INSTANCE) )
    {
        //wait
    }
#endif
    // Can now read or modify USB module status
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_Disable (USB_MODULE_ID index)

PLIB_USB_Enable Function

Enables (powers up) the USB module.

File

plib_usb.h

C

```
void PLIB_USB_Enable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables (powers up) the USB module.

Remarks

See also PLIB_USB_ModuleIsBusy.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsModulePoweryour application to determine whether this feature is available.

Preconditions

None.

Example

```
    // Complete Needed setup for the module
    PLIB_USB_Enable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_Enable (USB_MODULE_ID index)

PLIB_USB_FullSpeedDisable Function

Forces the USB module to operate at low speed.

File

plib_usb.h

C

```
void PLIB_USB_FullSpeedDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function forces the USB module to operate at low speed.

Remarks

For PIC32 devices: Host mode only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSpeedControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_FullSpeedDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_FullSpeedDisable (USB_MODULE_ID index)

PLIB_USB_FullSpeedEnable Function

Enables the USB to operate at full speed.

File

plib_usb.h

C

```
void PLIB_USB_FullSpeedEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the USB to operate at full speed.

Remarks

For PIC32 devices: Host mode only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSpeedControl in your application to determine whether this feature is available.

Preconditions

Use only before the USB module is enabled by calling PLIB_USB_Enable.

Example

```
PLIB_USB_FullSpeedEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_FullSpeedEnable (USB_MODULE_ID index)

PLIB_USB_OnChipPullUpDisable Function

Disables on-chip pull-ups.

File

plib_usb.h

C

```
void PLIB_USB_OnChipPullUpDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables on-chip pull-ups.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOnChipPullup in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OnChipPullUpDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OnChipPullUpDisable (USB_MODULE_ID index)

PLIB_USB_OnChipPullUpEnable Function

Enables on-chip pull-ups.

File

plib_usb.h

C

```
void PLIB_USB_OnChipPullUpEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables on-chip pull-ups. Pull-up on D+ in Full-Speed mode. Pull-up on D- in Low-Speed mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOnChipPullup in your application to determine whether this feature is available.

Preconditions

Use only before the USB module is enabled by calling PLIB_USB_Enable.

Example

```
PLIB_USB_OnChipPullUpEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OnChipPullUpEnable (USB_MODULE_ID index)

PLIB_USB_OperatingModeSelect Function

Selects the operating mode of the USB module.

File

plib_usb.h

C

```
void PLIB_USB_OperatingModeSelect(USB_MODULE_ID index, USB_OPMODES opMode);
```

Returns

None.

Description

This function selects the operating mode of the USB module, either Host, Device, or OTG.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOpModeSelect in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OperatingModeSelect( MY_USB_INSTANCE, USB_OPMODE_DEVICE );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
opMode	Selected operating mode: USB_OPMODE_DEVICE, USB_OPMODE_HOST, or USB_OPMODE_OTG

Function

void PLIB_USB_OperatingModeSelect(USB_MODULE_ID index, USB_OPMODES opMode)

PLIB_USB_PingPongModeGet Function

Returns the Ping-Pong Configuration setting.

File

plib_usb.h

C

```
USB_PING_PONG_MODE PLIB_USB_PingPongModeGet(USB_MODULE_ID index);
```

Returns

Ping-Pong Mode - One of USB_PING_PONG__ALL_BUT_EP0, USB_PING_PONG__FULL_PING_PONG, USB_PING_PONG__EP0_OUT_ONLY, USB_PING_PONG__NO_PING_PONG

Description

This function returns the Ping-Pong Configuration setting.

Remarks

None.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsPingPongMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
ppConfig = PLIB_USB_PingPongModeGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_PING_PONG_MODE PLIB_USB_PingPongModeGet (USB_MODULE_ID index)

PLIB_USB_PingPongModeSelect Function

Selects the Ping-Pong Configuration setting.

File

plib_usb.h

C

```
void PLIB_USB_PingPongModeSelect(USB_MODULE_ID index, USB_PING_PONG_MODE ppConfig);
```

Returns

None.

Description

This function selects the Ping-Pong Configuration setting.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsPingPongMode in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_PingPongModeSelect(MY_USB_INSTANCE,USB_PING_PONG__ALL_BUT_EP0);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
ppConfig	Ping-Pong configuration selection. One of USB_PING_PONG__ALL_BUT_EP0, USB_PING_PONG__FULL_PING_PONG, USB_PING_PONG__EP0_OUT_ONLY, USB_PING_PONG__NO_PING_PONG

Function

void PLIB_USB_PingPongModeSelect (USB_MODULE_ID index, USB_PING_PONG_MODE ppConfig)

PLIB_USB_SleepGuardDisable Function

File

plib_usb.h

C

```
void PLIB_USB_SleepGuardDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables Sleep Guard. Entry into Sleep mode is immediate.

Remarks

Not available on all PIC32 devices. Refer to the specific device data sheet for details.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSleepEntryGuard in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_SleepGuardDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_SleepGuardDisable (USB_MODULE_ID index)

Summary.

Disables Sleep Guard. Entry into Sleep mode is immediate.

PLIB_USB_SleepGuardEnable Function

Entry into Sleep mode is blocked if bus activity is detected or if an interrupt is pending.

File

plib_usb.h

C

```
void PLIB_USB_SleepGuardEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function block entry into Sleep mode if bus activity is detected or if an interrupt is pending.

Remarks

Not available on all PIC32 devices. Refer to the specific device data sheet for details. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSleepEntryGuard in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_SleepGuardEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_SleepGuardEnable (USB_MODULE_ID index)

PLIB_USB_StopInIdleDisable Function

Allows the USB module to continue operation when the device enters Idle mode.

File

plib_usb.h

C

```
void PLIB_USB_StopInIdleDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function allows the USB module to continue operation when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_StopInIdleDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_StopInIdleDisable (USB_MODULE_ID index)

PLIB_USB_StopInIdleEnable Function

Enables USB module operation to stop when the device enters Idle mode.

File

plib_usb.h

C

```
void PLIB_USB_StopInIdleEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables USB module operation to stop when the device enters Idle mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsStopInIdle in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_StopInIdleEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_StopInIdleEnable (USB_MODULE_ID index)

PLIB_USB_SuspendDisable Function

Disables USB OTG Suspend mode.

File

plib_usb.h

C

```
void PLIB_USB_SuspendDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables USB OTG Suspend mode. The USB OTG module will operate normally.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSuspend in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_SuspendDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_SuspendDisable (USB_MODULE_ID index)

PLIB_USB_SuspendEnable Function

Enables USB Suspend mode.

File

plib_usb.h

C

```
void PLIB_USB_SuspendEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables USB Suspend mode. The 48 MHz USB clock will be gated off. The transceiver is placed in a low-power state.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSuspend in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_SuspendEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_SuspendEnable (USB_MODULE_ID index)

PLIB_USB_UOEMonitorDisable Function

Disables the OE signal output.

File

plib_usb.h

C

```
void PLIB_USB_UOEMonitorDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the OE signal output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsUOEMonitor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Disable the OE output.
PLIB_USB_UOEMonitorDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_UOEMonitorDisable(USB_MODULE_ID index);

PLIB_USB_UOEMonitorEnable Function

Enables the OE signal output.

File

plib_usb.h

C

```
void PLIB_USB_UOEMonitorEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the OE signal output.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsUOEMonitor in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Enable the OE output.
PLIB_USB_UOEMonitorEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_UOEMonitorEnable(USB_MODULE_ID index);

b) Buffer Descriptor Table Functions

PLIB_USB_BDTBaseAddressGet Function

Returns the base address of the Buffer Descriptor Table.

File

plib_usb.h

C

```
void* PLIB_USB_BDTBaseAddressGet(USB_MODULE_ID index);
```

Returns

None.

Description

This function returns the base address of the Buffer Descriptor Table.

Remarks

Must be set for PIC32 devices using PLIB_USB_BDTBaseAddressSet.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTBaseAddress in your application to determine whether this feature is available.

Preconditions

None.

Example

```
void * pMyBDT;
pMyBDT = PLIB_USB_BDTBaseAddressGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void * PLIB_USB_BDTBaseAddressGet (USB_MODULE_ID index)

PLIB_USB_BDTBaseAddressSet Function

Sets the base address for the Buffer Descriptor Table for PIC32 devices.

File

plib_usb.h

C

```
void PLIB_USB_BDTBaseAddressSet(USB_MODULE_ID index, void* address);
```

Returns

None.

Description

This function sets the base address for the Buffer Descriptor Table. This function is only available on PIC32 devices.

Remarks

The address of the Buffer Descriptor Table must be 512 byte-aligned. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTBaseAddress in your application to determine whether this feature is available.

Preconditions

None.

Example

```
#if defined(__PIC32MX__)
  // For PIC32
  PLIB_USB_BDTBaseAddressSet(MY_USB_INSTANCE, (void *)((uint32_t)KVA_TO_PA(&myBDT)) );
#else
  // Everybody else
  PLIB_USB_BDTBaseAddressSet(MY_USB_INSTANCE, (void *)(&myBDT) );
#endif
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
address	Physical memory address in RAM of Buffer Descriptor Table

Function

void PLIB_USB_BDTBaseAddressSet (USB_MODULE_ID index, void * address)

PLIB_USB_BufferAddressGet Function

Gets the memory address of an endpoint buffer.

File

plib_usb.h

C

```
void* PLIB_USB_BufferAddressGet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

Buffer address in memory.

Description

This function gets the memory address of an endpoint buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint Value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void * PLIB_USB_BufferAddressGet (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferAddressSet Function

Sets the endpoint buffer address.

File

plib_usb.h

C

```
void PLIB_USB_BufferAddressSet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong, void* bufferAddress);
```

Returns

None.

Description

This function sets the endpoint buffer address.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD
bufferAddress	address in memory of endpoint transmit or receive buffer

Function

void PLIB_USB_BufferAddressSet (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong,
void * bufferAddress)

PLIB_USB_BufferAllCancelReleaseToUSB Function

Cancels all endpoint buffer releases to the USB module and hands over the buffer to the CPU.

File

plib_usb.h

C

```
void PLIB_USB_BufferAllCancelReleaseToUSB(USB_MODULE_ID index, void * pBDT, USB_PING_PONG_MODE ppMode, int nEndpoints);
```

Returns

None.

Description

This function cancels all endpoint buffer releases to the USB module and hands over the buffer to the CPU.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Cancel all buffer releases to USB.
//BDT has 3 Endpoints.

PLIB_USB_BufferAllCancelReleaseToUSB(MY_USB_INSTANCE, pBDT,
                                USB_PING_PONG_NO_PING_PONG, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pBDT	Pointer to the Buffer Descriptor Table
ppMode	Buffer Descriptor Table Ping-Pong mode
nEndpoints	Number of endpoints in the Buffer-Descriptor table

Function

void PLIB_USB_BufferAllCancelReleaseToUSB(USB_MODULE_ID index,
void * pBDT, USB_PING_PONG_MODE ppMode, int nEndpoints);

PLIB_USB_BufferByteCountGet Function

Returns the endpoint buffer byte count.

File

plib_usb.h

C

```
uint16_t PLIB_USB_BufferByteCountGet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

Endpoint buffer byte count.

Description

This function returns the endpoint buffer byte count, the actual number of bytes transmitted or received.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

uint16_t PLIB_USB_BufferByteCountGet (USB_MODULE_ID index,

void * pBDT,

USB_PING_PONG_MODE ppMode,

uint8_t epValue,

USB_BUFFER_DIRECTION bufferDirection,

USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferByteCountSet Function

Sets the buffer byte count.

File

plib_usb.h

C

```
void PLIB_USB_BufferByteCountSet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong, uint16_t
bufferByteCount);
```

Returns

None.

Description

This function sets the number of bytes to be transmitted or the maximum number of bytes to be received.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD
bufferByteCount	number of bytes to be transmitted or the maximum number of bytes to be received

Function

PLIB_USB_BufferByteCountSet (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong,
uint16_t bufferByteCount)

PLIB_USB_BufferCancelReleaseToUSB Function

Cancels release of the endpoint buffer by software, allowing software to again access the buffer.

File

plib_usb.h

C

```
void PLIB_USB_BufferCancelReleaseToUSB(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function cancels the release of the endpoint buffer by software, allowing software to again access the buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferCancelReleaseToUSB (USB_MODULE_ID index,
void * pBDT,

USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferClearAll Function

Clears (zeros out) entries in the Buffer Descriptor Table.

File

plib_usb.h

C

```
void PLIB_USB_BufferClearAll(USB_MODULE_ID index, void * pBDT, USB_PING_PONG_MODE ppMode, uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function clears (zeros out) the entries in the Buffer Descriptor Table.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferClearAll(USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferClearAllDTSEnable Function

Clears the endpoint descriptor entry and enables data toggle synchronization.

File

plib_usb.h

C

```
void PLIB_USB_BufferClearAllDTSEnable(USB_MODULE_ID index, void * pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection);
```

Returns

None.

Description

This function clears the endpoint descriptor entry and enables data toggle synchronization.

Remarks

None.

Preconditions

None.

Example

```
//Clear endpoint 6 buffer descriptor transmit entry and
//enable data toggle synchronization.

PLIB_USB_BufferClearAllDTSEnable ( MY_USB_INSTANCE, pBDT,
    USB_PING_PONG_NO_PING_PONG, 6, USB_BUFFER_TX);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pBDT	pointer to Buffer Descriptor Table
pingpong	Ping-Pong mode.
epvalue	Endpoint to be be affected
bufferDirection	Endpoint direction

Function

void PLIB_USB_BufferClearAllDTSEnable(USB_MODULE_ID index,

void * pBDT,

USB_PING_PONG_MODE ppMode,

uint8_t epValue,

USB_BUFFER_DIRECTION bufferDirection);

PLIB_USB_BufferDataToggleGet Function

Returns data synchronization (DATA0 or DATA1) for the endpoint buffer.

File

plib_usb.h

C

```
USB_BUFFER_DATA01 PLIB_USB_BufferDataToggleGet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode,
uint8_t epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

Data Toggle value, USB_BUFFER_DATA0 or USB_BUFFER_DATA1, for the buffer

Description

This function returns data synchronization (DATA0 or DATA1) for the endpoint buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

USB_BUFFER_DATA01

PLIB_USB_BufferDataToggleGet (USB_MODULE_ID index,

void * pBDT,

USB_PING_PONG_MODE ppMode,

uint8_t epValue,

USB_BUFFER_DIRECTION bufferDirection,

USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferDataToggleSelect Function

Sets the endpoint buffer to DATA0 or DATA1.

File

plib_usb.h

C

```
void PLIB_USB_BufferDataToggleSelect(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong, USB_BUFFER_DATA01
bufferData01);
```

Returns

None.

Description

This function sets the endpoint buffer to DATA0 or DATA1.

Remarks

See PLIB_USB_BufferDataToggleGet to determine the received data toggle setting. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD
bufferData01	USB_BUFFER_DATA0 or USB_BUFFER_DATA1

Function

void PLIB_USB_BufferDataToggleSelect (USB_MODULE_ID index,

void * pBDT,

USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong,
USB_BUFFER_DATA01 bufferData01)

PLIB_USB_BufferDataToggleSyncDisable Function

Disables DATA0/DATA1 synchronization between the device and host.

File

plib_usb.h

C

```
void PLIB_USB_BufferDataToggleSyncDisable(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode,
uint8_t epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function disables DATA0/DATA1 synchronization between the device and host.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferDataToggleSyncDisable (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferDataToggleSyncEnable Function

Enables DATA0/DATA1 synchronization between the device and host.

File

plib_usb.h

C

```
void PLIB_USB_BufferDataToggleSyncEnable(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode,
uint8_t epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function enables DATA0/DATA1 synchronization between the device and host.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferDataToggleSyncEnable (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferEP0RxStatusInitialize Function

Initializes the Endpoint 0 RX endpoint buffer descriptors.

File

plib_usb.h

C

```
void PLIB_USB_BufferEP0RxStatusInitialize(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode,
USB_BUFFER_PING_PONG pingpong, uint16_t bufferByteCount);
```

Returns

None.

Description

This function initializes the Endpoint 0 RX endpoint buffer descriptors. This function will clear the Endpoint 0 RX Buffer Descriptor status field, load the endpoint size and release the buffer to the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Initialize EP0 RX even buffer descriptor and release back to
//USB. Buffer size is 64
```

```
PLIB_USB_BufferEP0RxStatusInitialize ( MY_USB_INSTANCE, pBDT,
    USB_PING_PONG_NO_PING_PONG, USB_BUFFER_EVEN, 64);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pBDT	Pointer to Buffer Descriptor Table
pingpong	Ping-Pong mode
bufferByteCount	size of the EP0 RX buffer in bytes

Function

void PLIB_USB_BufferEP0RxStatusInitialize (USB_MODULE_ID index,
void* pBDT,
USB_PING_PONG_MODE ppMode,
USB_BUFFER_PING_PONG pingpong,
uint16_t bufferByteCount);

PLIB_USB_BufferIndexGet Function

Gets the Buffer Descriptor Table index for a buffer.

File

plib_usb.h

C

```
uint8_t PLIB_USB_BufferIndexGet(USB_MODULE_ID index, USB_PING_PONG_MODE ppMode, uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

Buffer index into the Buffer Descriptor Table.

Description

This function gets the Buffer Descriptor Table index for a buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

uint8_t PLIB_USB_BufferIndexGet (USB_MODULE_ID index,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferPIDBitsClear Function

Clears the Buffer Status bits in the Buffer Descriptor Table.

File

plib_usb.h

C

```
void PLIB_USB_BufferPIDBitsClear(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function clears the Buffer Status bits in the Buffer Descriptor Table.

Remarks

Call PLIB_USB_BufferPIDBitsClear before setting buffer control bits. This is equivalent to: PLIB_USB_BufferCancelReleaseToUSB(...) PLIB_USB_BufferDataToggleSelect(...,USB_BUFFER_DATA0) PLIB_USB_BufferDataToggleSyncDisable(...) PLIB_USB_BufferStallDisable(...) This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

The associated buffer must have been released by the USB module (i.e., PLIB_USB_BufferReleasedToSW returns 'true'.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferPIDBitsClear (USB_MODULE_ID index,

void * pBDT,

USB_PING_PONG_MODE ppMode,

uint8_t epValue,

USB_BUFFER_DIRECTION bufferDirection,

USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferPIDGet Function

Returns the token packet ID (PID) from the endpoint buffer status.

File

plib_usb.h

C

```
uint8_t PLIB_USB_BufferPIDGet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

Endpoint buffer packet ID (PID).

Description

This function returns the token packet ID (PID) from the endpoint buffer status.

Remarks

There is no equivalent "Set" routine, since this field is read-only in the buffer status register within the Buffer Descriptor Table. It is set when the buffer has been transmitted or received by the USB module and the usbOwnsBuffer field has been cleared by the USB module, releasing the buffer for software access. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

uint8_t PLIB_USB_BufferPIDGet (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferReleasedToSW Function

Returns the boolean flag value of 'true' when the buffer has been released by the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_BufferReleasedToSW(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

- true - The buffer has been released by hardware
- false - The buffer is still controlled by hardware

Description

This function returns the boolean flag value of 'true' when the buffer has been released by the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

bool PLIB_USB_BufferReleasedToSW (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferReleaseToUSB Function

Releases the endpoint buffer by software, allowing the USB module access to the buffer.

File

plib_usb.h

C

```
void PLIB_USB_BufferReleaseToUSB(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function releases the endpoint buffer by software, allowing the USB module access to buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferReleaseToUSB (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,

USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferSchedule Function

Hands over a buffer to the USB module along with the buffer address and byte count.

File

plib_usb.h

C

```
void PLIB_USB_BufferSchedule(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong, void * bufferAddress, int16_t
bufferByteCount, USB_BUFFER_SCHEDULE_DATA01 bufferData01);
```

Returns

None.

Description

This function sets the endpoint descriptor buffer address, sets the send/receive byte count, and then hands over the buffer to the USB module.

Remarks

This function does the work of three other functions: PLIB_USB_BufferAddressSet, PLIB_USB_BufferByteCountSet, PLIB_USB_BufferReleaseToUSB

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

None.

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD
bufferAddress	Address of the application buffer
bufferByteCount	Send or expected receive byte count
bufferData01	USB_BUFFER_SET_DATA0, USB_BUFFER_SET_DATA1, or USB_BUFFER_DONTCHANGE (The last choice leaves the existing DATA0/1 value of the buffer alone.)

Function

void PLIB_USB_BufferSchedule(USB_MODULE_ID index ,
void* pBDT ,
USB_PING_PONG_MODE ppMode ,
uint8_t epValue ,
USB_BUFFER_DIRECTION bufferDirection ,
USB_BUFFER_PING_PONG bufferPingPong ,
void * bufferAddress ,
int16_t bufferByteCount ,
USB_BUFFER_SCHEDULE_DATA01 bufferData01)

PLIB_USB_BufferStallDisable Function

Disables STALL handshaking for the associated endpoint buffer.

File

plib_usb.h

C

```
void PLIB_USB_BufferStallDisable(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function disables STALL handshaking for the associated endpoint buffer.

Remarks

Release of a STALL handshake for the buffer is done by hardware when the host sends a SETUP token to the associated endpoint or resets the USB module This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available. .

Preconditions

The associated buffer must have been released by the USB module (i.e., PLIB_USB_BufferReleasedToSW returns 'true').

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferStallDisable (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferStallEnable Function

Enables STALL handshaking for the associated endpoint buffer.

File

plib_usb.h

C

```
void PLIB_USB_BufferStallEnable(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t
epValue, USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

None.

Description

This function enables STALL handshaking for the associated endpoint buffer.

Remarks

Release of a STALL handshake for the buffer is done by hardware when the host sends a SETUP token to the associated endpoint or resets the USB module. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

The associated buffer must have been released by the USB module (i.e. PLIB_USB_BufferReleasedToSW returns 'true').

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

void PLIB_USB_BufferStallEnable (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,
uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

PLIB_USB_BufferStallGet Function

Returns the buffer stall status for an endpoint/direction/ping-pong.

File

plib_usb.h

C

```
bool PLIB_USB_BufferStallGet(USB_MODULE_ID index, void* pBDT, USB_PING_PONG_MODE ppMode, uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection, USB_BUFFER_PING_PONG bufferPingPong);
```

Returns

- true - Buffer stall is enabled
- false - Buffer stall is not enabled

Description

This function returns the buffer stall status for an endpoint/direction/ping-pong.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBDTFunctions in your application to determine whether this feature is available.

Preconditions

the associated buffer must have been released by the USB module (i.e., PLIB_USB_BufferReleasedToSW returns 'true').

Example

Parameters

Parameters	Description
index	Dummy argument, identifier for the device instance of interest
pBDT	Pointer to start of Buffer Descriptor Table
ppMode	Ping-Pong buffering mode
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
bufferDirection	USB_BUFFER_RX or USB_BUFFER_TX
bufferPingPong	USB_BUFFER_EVEN or USB_BUFFER_ODD

Function

bool PLIB_USB_BufferStallGet (USB_MODULE_ID index,
void * pBDT,
USB_PING_PONG_MODE ppMode,

uint8_t epValue,
USB_BUFFER_DIRECTION bufferDirection,
USB_BUFFER_PING_PONG bufferPingPong)

c) Endpoints Functions

PLIB_USB_EP0HostSetup Function

Sends token to the specified address.

File

plib_usb.h

C

```
void PLIB_USB_EP0HostSetup(USB_MODULE_ID index);
```

Returns

None.

Description

This function configures endpoint 0 for typical host operation. Control transfers are enable. Transmit and Receive is enabled. Handshaking is enabled. Low Speed connection is disabled. NAK retry is disabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Configure endpoint 0 for host operation

PLBIB_USB_EP0HostSetup(USB_MODULE_ID index);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EP0HostSetup(USB_MODULE_ID index);

PLIB_USB_EP0LSDirectConnectDisable Function

Disables direct connection to a low-speed device for Endpoint 0.

File

plib_usb.h

C

```
void PLIB_USB_EP0LSDirectConnectDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables direct connection to a low-speed device for Endpoint 0.

Remarks

Host mode and U1EP0 only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEP0LowSpeedConnect in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
PLIB_USB_EP0LSDirectConnectDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EP0LSDirectConnectDisable (USB_MODULE_ID index)

PLIB_USB_EP0LSDirectConnectEnable Function

Enables direct connection to a low-speed device for Endpoint 0.

File

plib_usb.h

C

```
void PLIB_USB_EP0LSDirectConnectEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables direct connection to a low-speed device for Endpoint 0.

Remarks

Host Mode and U1EP0 only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEP0LowSpeedConnect in your application to determine whether this feature is available.

Preconditions

USB module must be in Host mode.

Example

```
PLIB_USB_EP0LSDirectConnectEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EP0LSDirectConnectEnable (USB_MODULE_ID index)

PLIB_USB_EP0NakRetryDisable Function

Disables retrying of NAKed transactions.

File

plib_usb.h

C

```
void PLIB_USB_EP0NakRetryDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables retrying of NAKed transactions.

Remarks

Host/OTG only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEP0NAKRetry in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host or OTG modes.

Example

```
PLIB_USB_EP0NakRetryDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EP0NakRetryDisable (USB_MODULE_ID index)

PLIB_USB_EP0NakRetryEnable Function

Enables retrying NAK'd transactions for Endpoint 0.

File

plib_usb.h

C

```
void PLIB_USB_EP0NakRetryEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables retrying NAK'd transactions for Endpoint 0.

Remarks

Host/OTG only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEP0NAKRetry in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host or OTG modes.

Example

```
PLIB_USB_EP0NakRetryEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EP0NakRetryEnable (USB_MODULE_ID index)

PLIB_USB_EPnAttributesClear Function

Clears the set attributes of the specified endpoint.

File

plib_usb.h

C

```
void PLIB_USB_EPnAttributesClear(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

Clears the set attributes of the specified endpoint. The endpoint transmit receive, handshake and setup packet handling capability is disabled.

Remarks

None.

Preconditions

None.

Example

```
// This clears up the endpoint 0 attributes and thus disables
// the endpoints
PLIB_USB_EPnAttributesClear(MY_USB_INSTANCE, 0);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_EPnAttributesClear (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnAttributesSet Function

Configures attributes of the endpoint such as direction, handshake capability and direction.

File

plib_usb.h

C

```
void PLIB_USB_EPnAttributesSet(USB_MODULE_ID index, uint8_t epValue, int direction, bool isControl, bool
handshake);
```

Returns

None.

Description

Configures attributes of the endpoint such as direction, handshake capability and direction. If the isControl flag is true, then the direction and handshake parameters are ignored and the endpoint is configured for control transfers.

Remarks

None.

Preconditions

None.

Example

```
// This enables endpoint 0 for control transfers.
PLIB_USB_EPnAttributesSet(MY_USB_INSTANCE, 0, 0, true, true);
// This enables endpoint 2 for non control transfer, direction
// is RX and handshake enable.
PLIB_USB_EPnAttributesSet(MY_USB_INSTANCE, 2, 1, false, true);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
direction	Endpoint direction, if 1 then RX and if 0 then TX.

isControl	If true endpoint is configured for control transfers.
handshake	If true, then handshake is enabled on the endpoint else it is disabled.

Function

void PLIB_USB_EPnAttributesSet (USB_MODULE_ID index,
uint8_t epValue, int direction, bool isControl, bool handshake)

PLIB_USB_EPnControlTransferDisable Function

Disables endpoint control transfers.

File

plib_usb.h

C

```
void PLIB_USB_EPnControlTransferDisable(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

This function disables endpoint control transfers when endpoint transmit and endpoint receive are both enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnControlTransferDisable(MY_USB_INSTANCE, someEP);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_EPnControlTransferDisable (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnControlTransferEnable Function

Enables endpoint control transfers.

File

plib_usb.h

C

```
void PLIB_USB_EPnControlTransferEnable(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

This function enables endpoint control transfers when endpoint transmit and endpoint receive are both enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnControlTransferEnable(MY_USB_INSTANCE, someEP);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_EPnControlTransferEnable (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnDirectionDisable Function

Disables the specified endpoint direction.

File

plib_usb.h

C

```
void PLIB_USB_EPnDirectionDisable(USB_MODULE_ID index, uint8_t epValue, int direction);
```

Returns

None.

Description

Disables the specified endpoint direction.

Remarks

None.

Preconditions

None.

Example

```
// This function disables the TX direction of endpoint 1
PLIB_USB_EPnDirectionDisable(MY_USB_INSTANCE, 1, 1);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
direction	If 1, then TX direction is disabled. If 0 RX direction is disabled.

Function

void PLIB_USB_EPnDirectionDisable (USB_MODULE_ID index,

uint8_t epValue, int direction)

PLIB_USB_EPnHandshakeDisable Function

Disables endpoint handshaking.

File

plib_usb.h

C

```
void PLIB_USB_EPnHandshakeDisable(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

This function disables endpoint handshaking. Typically used for Isochronous endpoints.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnHandshakeDisable(MY_USB_INSTANCE, someEP);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_EPnHandshakeDisable (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnHandshakeEnable Function

Enables endpoint handshaking.

File

plib_usb.h

C

```
void PLIB_USB_EPnHandshakeEnable(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

This function enables endpoint handshaking.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnHandshakeEnable(MY_USB_INSTANCE, someEP);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_EPnHandshakeEnable (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnIsStalled Function

Tests whether the endpoint epValue is stalled.

File

plib_usb.h

C

```
bool PLIB_USB_EPnIsStalled(USB_MODULE_ID index, uint8_t epValue);
```

Returns

- true - The endpoint is stalled
- false - The endpoint is not stalled

Description

This function tests whether the endpoint epValue is stalled.

Remarks

Not valid before an endpoint is enabled. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if( PLIB_USB_EPnIsStalled(MY_USB_INSTANCE, someEP) )
{
    // Handle the stall
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

bool PLIB_USB_EPnIsStalled (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnRxDisable Function

Disables an endpoint's ability to process IN tokens.

File

plib_usb.h

C

```
void PLIB_USB_EPnRxDisable(USB_MODULE_ID index, uint8_t endpoint);
```

Returns

None.

Description

This function disables an endpoint's ability to process IN tokens.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//De-provision endpoint 3 to process IN Token
PLIB_USB_EPnRxDisable(MY_USB_INSTANCE, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
endpoint	Endpoint to be affected

Function

void PLIB_USB_EPnRxDisable(USB_MODULE_ID index, uint8_t endpoint);

PLIB_USB_EPnRxEnable Function

Enables an endpoint to process IN tokens.

File

plib_usb.h

C

```
void PLIB_USB_EPnRxEnable(USB_MODULE_ID index, uint8_t endpoint);
```

Returns

None.

Description

This function enables an endpoint to process IN tokens.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//Provision endpoint 3 to process IN Token
PLIB_USB_EPnRxEnable(MY_USB_INSTANCE, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
endpoint	Endpoint to be affected

Function

void PLIB_USB_EPnRxEnable(USB_MODULE_ID index, uint8_t endpoint);

PLIB_USB_EPnRxSelect Function

Selects receive capabilities of an endpoint.

File

plib_usb.h

C

```
void PLIB_USB_EPnRxSelect(USB_MODULE_ID index, uint8_t epValue, USB_EP_TXRX epTxRx);
```

Returns

None.

Description

This function selects receive capabilities of an endpoint.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnRxSelect(MY_USB_INSTANCE, someEP, USB_EP_RX);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
epTxRx	Transmit/Receive setting for endpoint: USB_EP_RX, USB_EP_NOTXRX

Function

void PLIB_USB_EPnRxSelect (USB_MODULE_ID index, uint8_t epValue, USB_EP_TXRX epTxRx)

PLIB_USB_EPnStallClear Function

Clears an endpoint's stalled flag.

File

plib_usb.h

C

```
void PLIB_USB_EPnStallClear(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

This function clears an endpoint's stalled flag.

Remarks

Not valid before an endpoint is enabled. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if( PLIB_USB_EPnIsStalled(MY_USB_INSTANCE, someEP) )
{
    // Handle the stall
    PLIB_USB_EPnStallClear(MY_USB_INSTANCE, someEP);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_EPnStallClear (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_EPnTxDisable Function

Disables an endpoint's ability to process OUT tokens.

File

plib_usb.h

C

```
void PLIB_USB_EPnTxDisable(USB_MODULE_ID index, uint8_t endpoint);
```

Returns

None.

Description

This function disables an endpoint's ability to process OUT tokens.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
//De-provision endpoint 3 to process OUT Token
PLIB_USB_EPnTxDisable(MY_USB_INSTANCE, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
endpoint	Endpoint to be affected

Function

void PLIB_USB_EPnTxDisable(USB_MODULE_ID index, uint8_t endpoint);

PLIB_USB_EPnTxEnable Function

Enables an endpoint to process OUT tokens.

File

plib_usb.h

C

```
void PLIB_USB_EPnTxEnable(USB_MODULE_ID index, uint8_t endpoint);
```

Returns

None.

Description

This function enables an endpoint to process OUT tokens.

Remarks

None.

Preconditions

None.

Example

```
//Provision endpoint 3 to process OUT Token
PLIB_USB_EPnTxEnable(MY_USB_INSTANCE, 3);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
endpoint	Endpoint to be affected

Function

void PLIB_USB_EPnTxEnable(USB_MODULE_ID index, uint8_t endpoint);

PLIB_USB_EPnTxRxSelect Function

Selects transmit and/or receive capabilities of an endpoint.

File

plib_usb.h

C

```
void PLIB_USB_EPnTxRxSelect(USB_MODULE_ID index, uint8_t epValue, USB_EP_TXRX epTxRx);
```

Returns

None.

Description

This function selects transmit and/or receive capabilities of an endpoint.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnTxRxSelect(MY_USB_INSTANCE, someEP, USB_EP_TXRX);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
epTxRx	Transmit/Receive setting for endpoint: USB_EP_TX, USB_EP_RX, USB_EP_TX_RX, USB_EP_NOTXRX

Function

void PLIB_USB_EPnTxRxSelect (USB_MODULE_ID index, uint8_t epValue, USB_EP_TXRX epTxRx)

PLIB_USB_EPnTxSelect Function

Selects transmit capabilities of an endpoint.

File

plib_usb.h

C

```
void PLIB_USB_EPnTxSelect(USB_MODULE_ID index, uint8_t epValue, USB_EP_TXRX epTxRx);
```

Returns

None.

Description

This function selects transmit capabilities of an endpoint.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEPnRxEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EPnTxSelect(MY_USB_INSTANCE, someEP, USB_EP_TX);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint
epTxRx	Transmit/Receive setting for endpoint: USB_EP_TX, USB_EP_NOTXRX

Function

void PLIB_USB_EPnTxSelect (USB_MODULE_ID index, uint8_t epValue, USB_EP_TXRX epTxRx)

d) Interrupts Functions

PLIB_USB_InterruptDisable Function

Disables a general interrupt for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_InterruptDisable(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function disables a general interrupt source for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_InterruptDisable( MY_USB_INSTANCE, USB_INT_ERROR );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_InterruptDisable(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

PLIB_USB_InterruptEnable Function

Enables a general interrupt for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_InterruptEnable(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function enables general interrupt sources of the USB module to trigger a USB interrupt.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_InterruptEnable( MY_USB_INSTANCE, USB_INT_ERROR );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_InterruptEnable(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

PLIB_USB_InterruptEnableGet Function

Returns the enable/disable status of general USB module interrupts

File

plib_usb.h

C

```
USB_INTERRUPTS PLIB_USB_InterruptEnableGet(USB_MODULE_ID index);
```

Returns

A bit map containing status of enabled interrupts.

Description

Returns the enable/disable status of general USB module interrupts

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
USB_INTERRUPTS enabledInterrupts;
enabledInterrupts = PLIB_USB_InterruptEnableGet( MY_USB_INSTANCE);
if(enabledInterrupts|USB_INT_ATTACH)
{
    // This means Attach interrupt is enabled.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_INTERRUPTS PLIB_USB_InterruptEnableGet(USB_MODULE_ID index)

PLIB_USB_InterruptFlagAllGet Function

Returns a logically ORed bit map of active general USB interrupt flags.

File

plib_usb.h

C

```
USB_INTERRUPTS PLIB_USB_InterruptFlagAllGet(USB_MODULE_ID index);
```

Returns

Returns a logically ORed bit map of active general USB interrupt flags.

Description

This function returns a logically ORed bit map of active general USB interrupt flags.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
USB_INTERRUPTS interruptEnables;

interruptEnables = PLIB_USB_InterruptFlagAllGet(MY_USB_INSTANCE);
if(interruptEnables | USB_INT_DEVICE_RESET)
{
    // Device received reset signaling
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_INTERRUPTS PLIB_USB_InterruptFlagAllGet(USB_MODULE_ID index);

PLIB_USB_InterruptFlagClear Function

Clears a general interrupt flag for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_InterruptFlagClear(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function clears a general interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_InterruptFlagClear( MY_USB_INSTANCE, USB_INT_ERROR );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_InterruptFlagClear(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

PLIB_USB_InterruptFlagGet Function

Tests a general interrupt flag for the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_InterruptFlagGet(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function tests a general interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if ( PLIB_USB_InterruptFlagGet(MY_USB_INSTANCE,USB_INT_ANY) )
    if ( PLIB_USB_InterruptFlagGet(MY_USB_INSTANCE,USB_INT_ERROR) )
    {
        PLIB_USB_InterruptFlagClear(MY_USB_INSTANCE,USB_INT_ERROR);
        // Error clean up
    }
    if ( PLIB_USB_InterruptFlagGet(MY_USB_INSTANCE,USB_INT_HOST_DETACH) )
    {
        PLIB_USB_InterruptFlagClear(MY_USB_INSTANCE,USB_INT_HOST_DETACH);
        // Device detached clean up
    }
```

```
        .
        .
        .
    }
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

bool PLIB_USB_InterruptFlagGet(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

PLIB_USB_InterruptFlagSet Function

Sets a general interrupt flag for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_InterruptFlagSet(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function sets a general interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_InterruptFlagSet( MY_USB_INSTANCE, USB_INT_ERROR );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_InterruptFlagSet(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

PLIB_USB_InterruptIsEnabled Function

Returns true if interrupts are enabled.

File

plib_usb.h

C

```
bool PLIB_USB_InterruptIsEnabled(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

- true - Interrupts are enabled
- false - Interrupts are not enabled

Description

This function returns true if interrupts are enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if ( PLIB_USB_InterruptIsEnabled( MY_USB_INSTANCE, USB_INT_ERROR ) )
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

bool PLIB_USB_InterruptIsEnabled(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

e) Error Interrupts Functions

PLIB_USB_ErrorInterruptDisable Function

Disables an error interrupt for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_ErrorInterruptDisable(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function disables an error interrupt source for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsERR_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_ErrorInterruptDisable( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_ErrorInterruptDisable(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag)

PLIB_USB_ErrorInterruptEnable Function

Enables an error interrupt for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_ErrorInterruptEnable(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function enables error interrupt sources of the USB module to trigger a USB interrupt.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsERR_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_ErrorInterruptEnable( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_ErrorInterruptEnable(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag)

PLIB_USB_ErrorInterruptFlagAllGet Function

Returns a logically ORed bit map of active error interrupt flags.

File

plib_usb.h

C

```
USB_ERROR_INTERRUPTS PLIB_USB_ErrorInterruptFlagAllGet(USB_MODULE_ID index);
```

Returns

Returns a logically ORed bit map of active error interrupt flags.

Description

This function returns a logically ORed bit map of active error interrupt flags.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsERR_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
USB_ERROR_INTERRUPTS errorInterruptEnables;

errorInterruptEnables = PLIB_USB_ErrorInterruptFlagAllGet(MY_USB_INSTANCE);
```

```
if(errorInterruptEnables | USB_ERR_INT_DEVICE_EOF_ERROR)
{
    // End of frame error occurred.
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_ERROR_INTERRUPTS PLIB_USB_ErrorInterruptFlagAllGet(USB_MODULE_ID index);

PLIB_USB_ErrorInterruptFlagClear Function

Clears an error interrupt flag for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_ErrorInterruptFlagClear(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function clears an error interrupt source flag for the USB module.

Remarks

None.

Preconditions

None.

Example

```
PLIB_USB_ErrorInterruptFlagClear( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_ErrorInterruptFlagClear(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag)

PLIB_USB_ErrorInterruptFlagGet Function

Tests an error interrupt flag for the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ErrorInterruptFlagGet(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function tests an error interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsERR_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if ( PLIB_USB_ErrorInterruptFlagGet(MY_USB_INSTANCE,USB_ERR_INT_ANY) )
    if ( PLIB_USB_ErrorInterruptFlagGet(MY_USB_INSTANCE,USB_ERR_INT_PID_CHECK_FAILURE) )
    {
        PLIB_USB_ErrorInterruptFlagClear(MY_USB_INSTANCE,USB_ERR_INT_PID_CHECK_FAILURE);
        // PID Error Check failure cleanup
    }
    if ( PLIB_USB_ErrorInterruptFlagGet(MY_USB_INSTANCE,USB_ERR_INT_DEVICE_EOF_ERROR) )
    {
        PLIB_USB_ErrorInterruptFlagClear(MY_USB_INSTANCE,USB_ERR_INT_DEVICE_EOF_ERROR);
        // EOF error cleanup
    }
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

bool PLIB_USB_ErrorInterruptFlagGet(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag)

PLIB_USB_ErrorInterruptFlagSet Function

Sets an error interrupt flag for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_ErrorInterruptFlagSet(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function sets an error interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsERR_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_ErrorInterruptFlagSet( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_ErrorInterruptFlagSet(USB_MODULE_ID index, USB_ERROR_INTERRUPTS interruptFlag)

PLIB_USB_ErrorInterruptIsEnabled Function

Returns true if interrupts are enabled.

File

plib_usb.h

C

```
bool PLIB_USB_ErrorInterruptIsEnabled(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

- true - Interrupts are enabled
- false - Interrupts are not enabled

Description

This function determines whether interrupts are enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsERR_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if ( PLIB_USB_ErrorInterruptIsEnabled( MY_USB_INSTANCE, USB_ERR_INT_BAD_CRC16 ) )
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

bool PLIB_USB_ErrorInterruptIsEnabled(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

f) Last Transaction Status Functions

PLIB_USB_LastTransactionDetailsGet Function

Returns the details of the last completed transaction.

File

plib_usb.h

C

```
void PLIB_USB_LastTransactionDetailsGet(USB_MODULE_ID index, USB_BUFFER_DIRECTION * direction,
USB_PING_PONG_STATE * pingpong, uint8_t * endpoint);
```

Returns

None.

Description

This function returns the details of the last completed transaction.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsLastTransactionDetails in your application to determine whether this feature is available.

Preconditions

None.

Example

```
USB_BUFFER_DIRECTION direction;
USB_PING_PONG_STATE pingpong;
uint8_t endpoint;

interruptEnables = PLIB_USB_InterruptFlagAllGet(MY_USB_INSTANCE);
if(interruptEnables | USB_INT_TOKEN_DONE)
{
    // Find out details of the token
    PLIB_USB_LastTransactionDetailsGet(MY_USB_INSTANCE, &direction,
                                       &pingpong, &endpoint);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
direction	Return value contains direction of the last transfer
pingpong	Return value contains Ping-Pong indication of the the last transfer
endpoint	Return value contains the endpoint which processed the last transfer

Function

void PLIB_USB_LastTransactionDetailsGet(USB_MODULE_ID index,

USB_BUFFER_DIRECTION * direction,

USB_PING_PONG_STATE * pingpong,

uint8_t * endpoint);

PLIB_USB_LastTransactionDirectionGet Function

Indicates the direction of the last transaction.

File

plib_usb.h

C

```
USB_BUFFER_DIRECTION PLIB_USB_LastTransactionDirectionGet(USB_MODULE_ID index);
```

Returns

USB_LastDirection: USB_RECEIVE_TRANSFER or USB_TRANSMIT_TRANSFER

Description

This function indicates the direction of the last transaction, either transmit or receive.

Remarks

None.

Preconditions

None.

Example

See PLIB_USB_LastTransactionEndPtGet.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_BUFFER_DIRECTION PLIB_USB_LastTransactionDirectionGet (USB_MODULE_ID index)

PLIB_USB_LastTransactionEndPtGet Function

Returns the endpoint number of the last USB transfer.

File

plib_usb.h

C

```
uint8_t PLIB_USB_LastTransactionEndPtGet(USB_MODULE_ID index);
```

Returns

endPoint - Endpoint of last completed USB transfer

Description

This function returns the endpoint number of the last USB transfer, which is actually the index into the Buffer Descriptor Table of the last USB transfer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsLastDirection in your application to determine whether this feature is available.

Preconditions

None.

Example

```
while ( !PLIB_USB_INT_FlagGet(MY_USB_INSTANCE,USB_INT_GEN,TOKEN_DONE) )
{
    // Do nothing, wait until completion of next transaction
}

// Retrieve information relating to the last completed transaction
endpoint  = PLIB_USB_LastTransactionEndPtGet(MY_USB_INSTANCE);
direction = PLIB_USB_LastTransactionDirectionGet(MY_USB_INSTANCE);
pingPongState = PLIB_USB_LastTransactionPingPongStateGet(MY_USB_INSTANCE);

// Clearing the Token Processing Done flag advances the status FIFO to
// oldest transaction in the FIFO.  Wait for completion of next transaction
// before using PLIB_USB_Last*Get functions again to read status.
PLIB_USB_INT_FlagClear(MY_USB_INSTANCE,USB_INT_GEN,TOKEN_DONE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

uint8_t PLIB_USB_LastTransactionEndPtGet (USB_MODULE_ID index)

PLIB_USB_LastTransactionPingPongStateGet Function

Indicates whether the last transaction was to an EVEN buffer or an ODD buffer.

File

plib_usb.h

C

```
USB_PING_PONG_STATE PLIB_USB_LastTransactionPingPongStateGet(USB_MODULE_ID index);
```

Returns

USB_PING_PONG_STATE.

Description

This function indicates whether the last transaction was to an Even buffer or an Odd buffer.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsLastPingPong in your application to determine whether this feature is available.

Preconditions

None.

Example

See PLIB_USB_LastTransactionEndPtGet.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_PING_PONG_STATE PLIB_USB_LastTransactionPingPongStateGet (USB_MODULE_ID index)

g) Host Functions

PLIB_USB_IsBusyWithToken Function

Indicates whether there is a token being executed by the USB module as Host.

File

plib_usb.h

C

```
bool PLIB_USB_IsBusyWithToken(USB_MODULE_ID index);
```

Returns

None.

Description

This function indicates whether there is a token being executed by the USB module as Host. Software should check that the previous token is finished before issuing a new token.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsHostBusyWithToken in your application to determine whether this feature is available.

Preconditions

USB module must be in Host mode.

Example

```
while( PLIB_USB_IsBusyWithToken(MY_USB_INSTANCE) )
{
    // do nothing
}
// Issue new token
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_IsBusyWithToken (USB_MODULE_ID index)

PLIB_USB_SOFDisable Function

Disables the automatic generation of the SOF token.

File

plib_usb.h

C

```
void PLIB_USB_SOFDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the automatic generation of the SOF token.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsStartOfFrames in your application to determine whether this feature is available.

Preconditions

USB module must be in Host mode.

Example

```
PLIB_USB_SOFDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_SOFDisable (USB_MODULE_ID index)

PLIB_USB_SOFEnable Function

Enables the automatic generation of the SOF token every 1 ms.

File

plib_usb.h

C

```
void PLIB_USB_SOFEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the automatic generation of the SOF token every 1 ms.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsStartOfFrames in your application to determine whether this feature is available.

Preconditions

USB module must be in Host mode.

Example

```
PLIB_USB_SOFEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_SOFEnable (USB_MODULE_ID index)

PLIB_USB_SOFThresholdGet Function

Returns the Start-of-Frame (SOF) Count bits.

File

plib_usb.h

C

```
uint8_t PLIB_USB_SOFThresholdGet(USB_MODULE_ID index);
```

Returns

SOF threshold value.

Description

This function returns the Start-of-Frame (SOF) Count bits. (Value represents 10 + (packet size of n bytes);).

Remarks

Host mode only.

```
    SOF Threshold Value = packet byte count + 10
                        = 0b0100_1010 = 0x4A = 74 for 64-byte packet
                        = 0b0010_1010 = 0x2A = 42 for 32-byte packet
                        = 0b0001_1010 = 0x1A = 26 for 16-byte packet
                        = 0x0001_0010 = 0x12 = 18 for  8-byte packet
```

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSOFThreshold in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
  thresholdSOF = PLIB_USB_SOFThresholdGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

uint8_t PLIB_USB_SOFThresholdGet (USB_MODULE_ID index)

PLIB_USB_SOFThresholdSet Function

Sets the Start-of-Frame (SOF) threshold value.

File

plib_usb.h

C

```
void PLIB_USB_SOFThresholdSet(USB_MODULE_ID index, uint8_t threshold);
```

Returns

None.

Description

This function sets the Start-of-Frame (SOF) threshold value.

Remarks

Host mode only.

```
    SOF Threshold Value = packet byte count + 10
                        = 0b0100_1010 = 0x4A = 74 for 64-byte packet
                        = 0b0010_1010 = 0x2A = 42 for 32-byte packet
                        = 0b0001_1010 = 0x1A = 26 for 16-byte packet
                        = 0x0001_0010 = 0x12 = 18 for  8-byte packet
```

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsSOFThreshold in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
// Set SOF threshold for 64-byte packets
PLIB_USB_SOFThresholdSet(MY_USB_INSTANCE,64+10);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
threshold	SOF threshold

Function

void PLIB_USB_SOFThresholdSet (USB_MODULE_ID index, uint8_t threshold)

PLIB_USB_TokenEPGet Function

Returns the specified Endpoint address.

File

plib_usb.h

C

```
uint8_t PLIB_USB_TokenEPGet(USB_MODULE_ID index);
```

Returns

Endpoint value - 0 <= epValue <= Module Maximum Endpoint.

Description

This function returns the address of the specified Endpoint.

Remarks

Host mode only.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsTokenEP in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
someEP = PLIB_USB_TokenEPGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

uint8_t PLIB_USB_TokenEPGet (USB_MODULE_ID index)

PLIB_USB_TokenEPSet Function

Sets the Endpoint address for a host transaction.

File

plib_usb.h

C

```
void PLIB_USB_TokenEPSet(USB_MODULE_ID index, uint8_t epValue);
```

Returns

None.

Description

This function sets the Endpoint address for a host transaction.

Remarks

Host mode only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsTokenEP in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
uint8_t someEP = 0x03;
PLIB_USB_TokenEPSet(MY_USB_INSTANCE, someEP);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
epValue	Endpoint value, 0 <= epValue <= Module Maximum Endpoint

Function

void PLIB_USB_TokenEPSet (USB_MODULE_ID index, uint8_t epValue)

PLIB_USB_TokenPIDGet Function

Returns the token transaction type.

File

plib_usb.h

C

```
USB_PID PLIB_USB_TokenPIDGet(USB_MODULE_ID index);
```

Returns

Packet ID of token, USB_PID_SETUP, USB_PID_IN, or USB_PID_OUT

Description

This function returns the token transaction type.

Remarks

Host mode only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsTokenPID in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
somePID = PLIB_USB_TokenPIDGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_PID PLIB_USB_TokenPIDGet (USB_MODULE_ID index)

PLIB_USB_TokenPIDSet Function

Sets the token transaction type to pidValue.

File

plib_usb.h

C

```
void PLIB_USB_TokenPIDSet(USB_MODULE_ID index, USB_PID pidValue);
```

Returns

None.

Description

This function sets the token transaction type to pidValue.

Remarks

Host mode only. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsTokenPID in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
somePID = USB_PID_SETUP;
PLIB_USB_TokenPIDSet (MY_USB_INSTANCE, somePID );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pidValue	USB_PID_SETUP, USB_PID_IN, or USB_PID_OUT

Function

void PLIB_USB_TokenPIDSet (USB_MODULE_ID index, USB_PID pidValue)

PLIB_USB_TokenSpeedSelect Function

Selects low speed or full speed for subsequent token executions.

File

plib_usb.h

C

```
void PLIB_USB_TokenSpeedSelect(USB_MODULE_ID index, USB_TOKEN_SPEED tokenSpeed);
```

Returns

None.

Description

This function selects low speed or full speed for subsequent token executions.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsNextTokenSpeed in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
PLIB_USB_TokenSpeedSet(MY_USB_INSTANCE,USB_LOWSPEED_TOKENS);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
tokenSpeed	Speed for next token execution: USB_LOWSPEED_TOKENS or USB_FULLSPEED_TOKENS

Function

void PLIB_USB_TokenSpeedSelect (USB_MODULE_ID index, USB_TOKEN_SPEED tokenSpeed)

h) USB Bus Signaling Functions

PLIB_USB_ResetSignalDisable Function

Disables reset signaling on the USB bus.

File

plib_usb.h

C

```
void PLIB_USB_ResetSignalDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables reset signaling on the USB bus.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsHostGeneratesReset in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

See PLIB_USB_ResetSignalEnable.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_ResetSignalDisable (USB_MODULE_ID index)

PLIB_USB_ResetSignalEnable Function

Enables reset signaling on the USB bus.

File

plib_usb.h

C

```
void PLIB_USB_ResetSignalEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables reset signaling on the USB bus.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsHostGeneratesReset in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
// Snippet to perform a software reset:
PLIB_USB_ResetSignalEnable(MY_USB_INSTANCE);
// ... delay 50ms
PLIB_USB_ResetSignalDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_ResetSignalEnable (USB_MODULE_ID index)

PLIB_USB_ResumeSignalingDisable Function

Disables resume signaling.

File

plib_usb.h

C

```
void PLIB_USB_ResumeSignalingDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables resume signaling.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsResumeSignaling in your application to determine whether this feature is available.

Preconditions

None.

Example

See PLIB_USB_ResumeSignalingEnable.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_ResumeSignalingDisable (USB_MODULE_ID index)

PLIB_USB_ResumeSignalingEnable Function

Enables resume signaling.

File

plib_usb.h

C

```
void PLIB_USB_ResumeSignalingEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables resume signaling. Resume allows the peripheral to perform a remote wake-up by executing resume signaling.

Remarks

Software must enable resume signaling for 10 ms if the device is in Device mode, or for 25 ms if the device is in Host mode, and then disable resume signaling to enable remote wake-up. In Host mode, the USB module will append a low-speed EOP to the end resume signaling when it is disabled. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsResumeSignaling in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Perform resume signaling:
PLIB_USB_ResumeSignalingEnable(MY_USB_INSTANCE);
// Delay 10ms
PLIB_USB_ResumeSignalingDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_ResumeSignalingEnable (USB_MODULE_ID index)

i) On-The-Go (OTG) Functions

PLIB_USB_OTG_BSessionHasEnded Function

Returns the status of the B-Session End Indicator bit.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_BSessionHasEnded(USB_MODULE_ID index);
```

Returns

- true - The VBUS Voltage is below VB_SESS_END on the B-device
- false - The VBUS voltage is above VB_SESS_END on the B-device

Description

This function returns the status of the B-Session End Indicator bit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_BSessionEnd in your application to determine whether this feature is available.

Preconditions

The USB module must be in OTG mode.

Example

```
if ( !PLIB_USB_OTG_BSessionHasEnded(MY_USB_INSTANCE) )
{
    // B session valid
}
else
{
    // B session not valid
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_OTG_BSessionHasEnded (USB_MODULE_ID index)

PLIB_USB_OTG_IDPinStateIsTypeA Function

Returns the ID Pin state.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_IDPinStateIsTypeA(USB_MODULE_ID index);
```

Returns

- true - Type A Cable attached,
- false - No cable is attached or a Type B cable is attached

Description

This function returns ID Pin state.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_IDPinState in your application to determine whether this feature is available.

Preconditions

The USB module must be in OTG mode.

Example

```
if ( PLIB_USB_OTG_IDPinStateIsTypeA(MY_USB_INSTANCE) )
{
    // Type A cable attached
}
else
{
    // No cable or Type B cable attached
};
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_OTG_IDPinStateIsTypeA (USB_MODULE_ID index)

PLIB_USB_OTG_LineStateIsStable Function

Returns the status of the Line Stable Indicator bit.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_LineStateIsStable(USB_MODULE_ID index);
```

Returns

- true - The USB line state has been stable for the previous 1 ms
- false - The USB line state has not been stable for the previous 1 ms

Description

This function returns the status of the Line Stable Indicator bit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_LineState in your application to determine whether this feature is available.

Preconditions

The USB module must be in OTG mode.

Example

```
if( PLIB_USB_OTG_LineStateIsStable(MY_USB_INSTANCE) ) {
    // Line has been stable
    // ... rest of code ...
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_OTG_LineStateIsStable (USB_MODULE_ID index)

PLIB_USB_OTG_PullUpPullDownSetup Function

Enables or disables pull-up and pull-down resistors.

File

plib_usb.h

C

```
void PLIB_USB_OTG_PullUpPullDownSetup(USB_MODULE_ID index, USB_OTG_PULL_UP_PULL_DOWN resistor, bool
enableResistor);
```

Returns

None.

Description

This function enables or disables pull-up and pull-down resistors.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_PullUpPullDown in your application to determine whether this feature is available.

Preconditions

USB On-The-Go (OTG) must be enabled.

Example

```
// Enable pull-up resistor for D+
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE,USB_OTG_DPLUS_PULLUP,true);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
resistor	USB_OTG_DPLUS_PULLUP, USB_OTG_DMINUS_PULLUP, USB_OTG_DPLUS_PULLDN, or USB_OTG_DMINUS_PULLDN
enableResistor	true to enable resistor, false to disable it

Function

void PLIB_USB_OTG_PullUpPullDownSetup(USB_MODULE_ID index,
USB_OTG_PULL_UP_PULL_DOWN resistor,
bool enableResistor)

PLIB_USB_OTG_SessionValid Function

Returns the status of the Session Valid Indicator bit.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_SessionValid(USB_MODULE_ID index);
```

Returns

- true - The VBUS voltage is above Session Valid on the A or B device
- false - The VBUS voltage is below Session Valid on the A or B device

Description

This function returns the status of the Session Valid Indicator bit.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_SessionValid in your application to determine whether this feature is available.

Preconditions

The USB module must be in OTG mode.

Example

```
if ( PLIB_USB_OTG_SessionValid(MY_USB_INSTANCE) )
{
    // Session valid
}
else
{
    // Session not valid
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_OTG_SessionValid (USB_MODULE_ID index)

PLIB_USB_OTG_VBusChargeDisable Function

Disables VBUS line charge.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusChargeDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables VBUS line charge.

Remarks

Not available on PIC32 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_VbusCharge in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusChargeDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusChargeDisable (USB_MODULE_ID index)

PLIB_USB_OTG_VBusChargeEnable Function

Enables the VBUS line to be charged through a pull-up resistor.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusChargeEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the VBUS line to be charged through a pull-up resistor.

Remarks

Not available on PIC32 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_VbusDischarge in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusChargeEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusChargeEnable (USB_MODULE_ID index)

PLIB_USB_OTG_VBusChargeTo3V Function

Sets the VBUS line to charge to 3.3V.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusChargeTo3V(USB_MODULE_ID index);
```

Returns

None.

Description

This function sets the VBUS line to charge to 3.3V.

Remarks

Not available on PIC32 devices.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusChargeTo3V(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusChargeTo3V(USB_MODULE_ID index)

PLIB_USB_OTG_VBusChargeTo5V Function

Sets the VBUS line to charge to 5V.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusChargeTo5V(USB_MODULE_ID index);
```

Returns

None.

Description

This function sets the VBUS line to charge to 5V.

Remarks

Not available on PIC32 devices.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusChargeTo5V (MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusChargeTo5V (USB_MODULE_ID index)

PLIB_USB_OTG_VBusDischargeDisable Function

Disables VBUS line discharge.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusDischargeDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables VBUS line discharge.

Remarks

Not available on PIC32 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_VbusDischarge in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusDischargeDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusDischargeDisable (USB_MODULE_ID index)

PLIB_USB_OTG_VBusDischargeEnable Function

Enables VBUS line to be discharged through a resistor.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusDischargeEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the VBUS line to be discharged through a resistor.

Remarks

Not available on PIC32 devices. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_VbusCharge in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusDischargeEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusDischargeEnable (USB_MODULE_ID index)

PLIB_USB_OTG_VBusPowerOff Function

Turns off power on the VBUS Line.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusPowerOff(USB_MODULE_ID index);
```

Returns

None.

Description

This function turns off power on the VBUS Line.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_VbusPowerOnOff in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusPowerOff(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusPowerOff (USB_MODULE_ID index)

PLIB_USB_OTG_VBusPowerOn Function

Turns on power for the VBUS line.

File

plib_usb.h

C

```
void PLIB_USB_OTG_VBusPowerOn(USB_MODULE_ID index);
```

Returns

None.

Description

This function turns on power for the VBUS line.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_VbusPowerOnOff in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_VBusPowerOn(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_OTG_VBusPowerOn (USB_MODULE_ID index)

PLIB_USB_OTG_VBusValid Function

Returns the status of the A-VBUS valid indicator.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_VBusValid(USB_MODULE_ID index);
```

Returns

- true - The VBUS voltage is above VA_VBUS_VLD on the A-device,
- false - The VBUS voltage is below VA_VBUS_VLD on the A-device

Description

This function returns the status of the A-VBUS valid indicator.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_ASessionValid in your application to determine whether this feature is available.

Preconditions

The USB module must be in OTG mode.

Example

```
if ( PLIB_USB_OTG_VBusValid(MY_USB_INSTANCE) )
{
    // VBUS voltage above session valid for A device
}
else
{
    // VBUS voltage below session valid for A device
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_OTG_VBusValid (USB_MODULE_ID index)

j) OTG Interrupts Functions

PLIB_USB_OTG_InterruptDisable Function

Disables a USB On-The-Go (OTG) Interrupt for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_OTG_InterruptDisable(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function disables a USB On-The-Go (OTG) interrupt source for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_InterruptDisable( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_OTG_InterruptDisable(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag)

PLIB_USB_OTG_InterruptEnable Function

Enables a USB On-The-Go (OTG) Interrupt for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_OTG_InterruptEnable(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function enables USB On-The-Go (OTG) interrupt sources of the USB module to trigger a USB interrupt.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_InterruptEnable( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_OTG_InterruptEnable(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag)

PLIB_USB_OTG_InterruptFlagClear Function

Clears a USB On-The-Go (OTG) Interrupt flag for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_OTG_InterruptFlagClear(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function clears a USB On-The-Go (OTG) interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_InterruptFlagClear( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_OTG_InterruptFlagClear(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag)

PLIB_USB_OTG_InterruptFlagGet Function

Tests a USB On-The-Go (OTG) Interrupt flag for the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_InterruptFlagGet(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function tests a USB On-The-Go (OTG) interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if ( PLIB_USB_OTG_InterruptFlagGet(MY_USB_INSTANCE,USB_OTG_INT_ANY) )
    if ( PLIB_USB_OTG_InterruptFlagGet(MY_USB_INSTANCE,USB_OTG_INT_BDEVICE_SESSION_END) )
    {
        PLIB_USB_OTG_InterruptFlagClear(MY_USB_INSTANCE,USB_OTG_INT_ADEVICE_VBUS_VALID );
        // Device A VBUS Valid Change
    }
    if ( PLIB_USB_OTG_InterruptFlagGet(MY_USB_INSTANCE,USB_OTG_INT_BDEVICE_SESSION_END) )
    {
        PLIB_USB_OTG_InterruptFlagClear(MY_USB_INSTANCE,USB_OTG_INT_BDEVICE_SESSION_END);
        // Device B VBUS Valid Change
    }
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

bool PLIB_USB_OTG_InterruptFlagGet(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag)

PLIB_USB_OTG_InterruptFlagSet Function

Sets a USB On-The-Go (OTG) Interrupt flag for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_OTG_InterruptFlagSet(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag);
```

Returns

None.

Description

This function sets a USB On-The-Go (OTG) interrupt source flag for the USB module.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_InterruptStatus in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_OTG_InterruptFlagSet( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USB_OTG_InterruptFlagSet(USB_MODULE_ID index, USB_OTG_INTERRUPTS interruptFlag)

PLIB_USB_OTG_InterruptIsEnabled Function

Returns whether or not interrupts are enabled.

File

plib_usb.h

C

```
bool PLIB_USB_OTG_InterruptIsEnabled(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag);
```

Returns

- true - Interrupts are enabled
- false - Interrupts are not enabled

Description

This function returns whether or not interrupts are enabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOTG_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if ( PLIB_USB_OTG_InterruptIsEnabled( MY_USB_INSTANCE, USB_OTG_INT_ID_STATE_CHANGE  ) )
{

}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

bool PLIB_USB_OTG_InterruptIsEnabled(USB_MODULE_ID index, USB_INTERRUPTS interruptFlag)

k) USB Activity Functions

PLIB_USB_ActivityPending Function

Returns whether or not USB activity is pending.

File

plib_usb.h

C

```
bool PLIB_USB_ActivityPending(USB_MODULE_ID index);
```

Returns

- true - The USB module should not be suspended
- false - No interrupts are pending or module may be suspended or powered down

Description

This function returns whether or not USB bus activity has been detected, an interrupt is pending, or an interrupt is yet to be generated.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsActivityPending in your application to determine whether this feature is available.

Preconditions

None.

Example

```
while ( PLIB_USB_ActivityPending(MY_USB_INSTANCE) )
{
    // Wait
}
// Suspend USB module.
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_ActivityPending (USB_MODULE_ID index)

PLIB_USB_FrameNumberGet Function

Returns the USB frame number.

File

plib_usb.h

C

```
uint16_t PLIB_USB_FrameNumberGet(USB_MODULE_ID index);
```

Returns

Current frame number in lower 11 bits.

Description

This function returns the USB frame number in the lower 11 bits.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsFrameNumber in your application to determine whether this feature is available.

Preconditions

None.

Example

```
frameNumber = PLIB_USB_FrameNumberGet(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

uint16_t PLIB_USB_FrameNumberGet (USB_MODULE_ID index)

PLIB_USB_JStateIsActive Function

Live differential receiver J State flag.

File

plib_usb.h

C

```
bool PLIB_USB_JStateIsActive(USB_MODULE_ID index);
```

Returns

None.

Description

This function indicates the live JState (differential '0' in low speed, differential '1' in full speed) on the bus.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsLiveJState in your application to determine whether this feature is available.

Preconditions

The USB module must be in Host mode.

Example

```
// Enable Host Mode
PLIB_USB_OperatingModeSelect(MY_USB_INSTANCE,USB_OPMODE_HOST);

// Enable D+ and D- pull-down resistors
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE,USB_OTG_DPLUS_PULLDN, true);
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE,USB_OTG_DMINUS_PULLDN,true);

// Disable D+ and D- pull-up resistors
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE,USB_OTG_DPLUS_PULLUP, false);
PLIB_USB_OTG_PullUpPullDownSetup(MY_USB_INSTANCE,USB_OTG_DMINUS_PULLUP,false);

// Enable SOF Packet generation
PLIB_USB_SOFEnable(MY_USB_INSTANCE);

// Enable the device attach interrupt
PLIB_USB_INT_Enable(MY_USB_INSTANCE,USB_INT_OTG,ACTIVITY_DETECT);

// Wait for the Attach interrupt.
while(!PLIB_USB_INT_FlagGet(MY_USB_INSTANCE,USB_INT_OTG,ACTIVITY_DETECT) )
{
    //Do nothing
}

// Check JState
if( PLIB_USB_JStateIsActive(MY_USB_INSTANCE) )
{
    // Full Speed
    PLIB_USB_FullSpeedEnable(MY_USB_INSTANCE);
}
else
{
    // Low Speed
    PLIB_USB_FullSpeedDisable(MY_USB_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_JStateIsActive (USB_MODULE_ID index)

PLIB_USB_PacketTransferDisable Function

Disables the Serial Interface Engine (SIE).

File

plib_usb.h

C

```
void PLIB_USB_PacketTransferDisable(USB_MODULE_ID index);
```

Returns

None.

```
  if( PLIB_USB_PacketTransferIsDisabled(MY_USB_INSTANCE) )
  {
      // SETUP token received, do the needful operations
      .
      .
      .
      // SETUP handling completed, enable Setup token and packet processing:
      PLIB_USB_PacketTransferDisable(MY_USB_INSTANCE);
  }
```

Description

This function disables the Serial Interface Engine (SIE).

Remarks

Not valid when the USB module is in Host mode. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsPacketTransfer in your application to determine whether this feature is available.

Preconditions

USB module must be in device mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PacketTransferDisable (USB_MODULE_ID index)

PLIB_USB_PacketTransferEnable Function

Re-enables the Serial Interface Engine (SIE), allowing token and packet processing.

File

plib_usb.h

C

```
void PLIB_USB_PacketTransferEnable(USB_MODULE_ID index);
```

Returns

None.

```
  if( PLIB_USB_PacketTransferIsDisabled(MY_USB_INSTANCE) )
  {
      // SETUP token received, do the needful operations
      .
      .
      .
      // SETUP handling completed, enable Setup token and packet processing:
      PLIB_USB_PacketTransferEnable(MY_USB_INSTANCE);
  }
```

Description

This function re-enables the Serial Interface Engine (SIE), allowing token and packet processing.

Remarks

Not valid when the USB module is in Host mode. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsPacketTransfer in your application to determine whether this feature is available.

Preconditions

USB module must be in device mode.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PacketTransferEnable (USB_MODULE_ID index)

PLIB_USB_PacketTransferIsDisabled Function

Indicates that a setup token has been received from the Host and that token/packet processing is disabled.

File

plib_usb.h

C

```
bool PLIB_USB_PacketTransferIsDisabled(USB_MODULE_ID index);
```

Returns

None.

Description

This function indicates that a setup token has been received from the Host and that the Serial Interface Engine (SIE) has been turned off, disabling token and packet processing.

Remarks

Not valid when USB is Host. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsPacketTransfer in your application to determine whether this feature is available.

Preconditions

USB module must be in device mode.

Example

```
if( PLIB_USB_PacketTransferIsDisabled(MY_USB_INSTANCE) )
{
    // SETUP token received, do the needful operations
    .
    .
    .
    // SETUP handling completed, enable Setup token and packet processing:
    PLIB_USB_PacketTransferEnable(MY_USB_INSTANCE);
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_PacketTransferIsDisabled (USB_MODULE_ID index)

PLIB_USB_SE0InProgress Function

Returns whether a single-ended zero event is in progress.

File

plib_usb.h

C

```
bool PLIB_USB_SE0InProgress(USB_MODULE_ID index);
```

Returns

- true - A single ended zero event (SE0) is occurring

- false - A single-ended zero event (SE0) is not occurring

Description

This function returns whether a single-ended zero event is in progress.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsLiveSingleEndedZero in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if( PLIB_USB_SE0InProgress(MY_USB_INSTANCE) )
{
    // handle the SE0 event
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_SE0InProgress(USB_MODULE_ID index)

l) External Transceiver Support Functions

PLIB_USB_I2CInterfaceForExtModuleDisable Function

Specifies external module(s) are controlled via dedicated pins.

File

plib_usb.h

C

```
void PLIB_USB_I2CInterfaceForExtModuleDisable(USB_MODULE_ID index);
```

Returns

None.

Description

Specifies that external module(s) are controlled via dedicated pins.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_I2CInterfaceForExtModuleDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_I2CInterfaceForExtModuleDisable (USB_MODULE_ID index)

PLIB_USB_I2CInterfaceForExtModuleEnable Function

Specifies external module(s) are controlled via the I2C interface.

File

plib_usb.h

C

```
void PLIB_USB_I2CInterfaceForExtModuleEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function specifies that external module(s) are controlled via the I2C interface.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_I2CInterfaceForExtModuleEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_I2CInterfaceForExtModuleEnable (USB_MODULE_ID index)

PLIB_USB_TransceiverDisable Function

Disables the on-chip transceiver

File

plib_usb.h

C

```
void PLIB_USB_TransceiverDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the on-chip transceiver and enables the interface to the off-chip transceiver.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOnChipTransceiver in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_TransceiverDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_TransceiverDisable (USB_MODULE_ID index)

PLIB_USB_TransceiverEnable Function

Enables the on-chip transceiver.

File

plib_usb.h

C

```
void PLIB_USB_TransceiverEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the on-chip transceiver. The interface to the off-chip transceiver is disabled.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsOnChipTransceiver in your application to determine whether this feature is available.

Preconditions

Use only before the USB module is enabled by calling PLIB_USB_Enable.

Example

```
PLIB_USB_TransceiverEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_TransceiverEnable (USB_MODULE_ID index)

m) VBUS Support Functions

PLIB_USB_ExternalComparatorMode2Pin Function

Sets the 2-pin input configuration for VBUS comparators.

File

plib_usb.h

C

```
void PLIB_USB_ExternalComparatorMode2Pin(USB_MODULE_ID index);
```

Returns

None.

Description

This function sets the 2-pin input configuration for VBUS Comparators.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_ExternalComparatorMode2Pin(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_ExternalComparatorMode2Pin (USB_MODULE_ID index)

PLIB_USB_ExternalComparatorMode3Pin Function

Sets the 3-pin input configuration for VBUS Comparators.

File

plib_usb.h

C

```
void PLIB_USB_ExternalComparatorMode3Pin(USB_MODULE_ID index);
```

Returns

None.

Description

This function sets the 3-pin input configuration for VBUS comparators.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_ExternalComparatorMode3Pin(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_ExternalComparatorMode3Pin (USB_MODULE_ID index)

PLIB_USB_PWMCounterDisable Function

Disables the PWM counter used to generate the VBUS for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_PWMCounterDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the PWM counter used to generate the VBUS for the USB module.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_PWMDisable(MY_USB_INSTANCE);
PLIB_USB_PWMCounterDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PWMCounterDisable(USB_MODULE_ID index);

PLIB_USB_PWMCounterEnable Function

Enables the PWM counter used to generate the VBUS for the USB module.

File

plib_usb.h

C

```
void PLIB_USB_PWMCounterEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the PWM counter used to generate the VBUS for the USB module.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_PWMEnable(MY_USB_INSTANCE);
PLIB_USB_PWMCounterEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PWMCounterEnable(USB_MODULE_ID index)

PLIB_USB_PWMDisable Function

Disables the PWM Generator.

File

plib_usb.h

C

```
void PLIB_USB_PWMDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the PWM Generator. PWM output held in a reset state defined by PLIB_USB_PWMPolarityActiveHigh or PLIB_USB_PWMPolarityActiveLow.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_PWMDisable (MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PWMDisable (USB_MODULE_ID index)

PLIB_USB_PWMEnable Function

Enables the PWM Generator.

File

plib_usb.h

C

```
void PLIB_USB_PWMEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the PWM Generator.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_PWMEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PWMEnable (USB_MODULE_ID index)

PLIB_USB_PWMPolaritiyActiveLow Function

Sets the PWM output to active-high and resets low.

File

plib_usb.h

C

```
void PLIB_USB_PWMPolaritiyActiveLow(USB_MODULE_ID index);
```

Returns

None.

Description

This function sets the PWM output to active-high and resets low.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PWMPolaritiyActiveLow (USB_MODULE_ID index)

PLIB_USB_PWMPolarityActiveHigh Function

Sets the PWM output to active-low and resets high.

File

plib_usb.h

C

```
void PLIB_USB_PWMPolarityActiveHigh(USB_MODULE_ID index);
```

Returns

None.

Description

This function sets the PWM output to active-low and resets high.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_PWMPolaritiy (MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PWMPolarityActiveHigh (USB_MODULE_ID index)

PLIB_USB_VBoostDisable Function

Disables the On-Chip 5V Boost Regulator Circuit Disabled bit.

File

plib_usb.h

C

```
void PLIB_USB_VBoostDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the On-Chip 5V Boost Regulator Circuit Disabled bit.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_VBoostDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_VBoostDisable (USB_MODULE_ID index)

PLIB_USB_VBoostEnable Function

Enables the On-Chip 5V Boost Regulator Circuit Enabled bit.

File

plib_usb.h

C

```
void PLIB_USB_VBoostEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the On-Chip 5V Boost Regulator Circuit Enabled bit.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_VBoostEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_VBoostEnable (USB_MODULE_ID index)

PLIB_USB_VBUSComparatorDisable Function

Disables the on-chip VBUS Comparator.

File

plib_usb.h

C

```
void PLIB_USB_VBUSComparatorDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the on-chip VBUS Comparator.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_VBUSComparatorDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_VBUSComparatorDisable (USB_MODULE_ID index)

PLIB_USB_VBUSComparatorEnable Function

Enables the on-chip VBUS Comparator.

File

plib_usb.h

C

```
void PLIB_USB_VBUSComparatorEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the on-chip VBUS Comparator.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_VBUSComparatorEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_VBUSComparatorEnable (USB_MODULE_ID index)

PLIB_USB_VBUSPullUpDisable Function

Disables the pull-up on the VBUS pin.

File

plib_usb.h

C

```
void PLIB_USB_VBUSPullUpDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the pull-up on the VBUS pin.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_VBUSPullUpDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_VBUSPullUpDisable (USB_MODULE_ID index)

PLIB_USB_VBUSPullUpEnable Function

Enables the pull-up on the VBUS pin.

File

plib_usb.h

C

```
void PLIB_USB_VBUSPullUpEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the pull-up on the VBUS pin.

Remarks

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is available on your device.

Preconditions

None.

Example

```
PLIB_USB_VBUSPullUpEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_VBUSPullUpEnable (USB_MODULE_ID index)

n) Test Support Functions

PLIB_USB_EyePatternDisable Function

Disables the USB eye pattern test.

File

plib_usb.h

C

```
void PLIB_USB_EyePatternDisable(USB_MODULE_ID index);
```

Returns

None.

Description

This function disables the USB eye pattern test.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEyePattern in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EyePatternDisable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EyePatternDisable (USB_MODULE_ID index)

PLIB_USB_EyePatternEnable Function

Enables USB eye pattern test.

File

plib_usb.h

C

```
void PLIB_USB_EyePatternEnable(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables the USB eye pattern test.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsEyePattern in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_EyePatternEnable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_EyePatternEnable (USB_MODULE_ID index)

o) Other Functions

PLIB_USB_ModuleIsBusy Function

Indicates if the USB module is not ready to be enabled.

File

plib_usb.h

C

```
bool PLIB_USB_ModuleIsBusy(USB_MODULE_ID index);
```

Returns

- true - USB module is active or disabled, but not ready to be enabled
- false - USB module is not active and is ready to be enabled

Description

This function indicates if the USB module is not ready to be enabled.

Remarks

If PLIB_USB_ModuleIsBusy(MY_USB_INSTANCE) == true and the USB module is disabled, and all status returned for the module, all enables/disables for the module will produce undefined results.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsModuleBusyin your application to determine whether this feature is available.

Preconditions

None.

Example

```
#ifdef (__PIC32MX__)
  while ( PLIB_USB_ModuleIsBusy(MY_USB_INSTANCE) )
  {
      // wait
  }
#endif
  PLIB_USB_Disable(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

bool PLIB_USB_ModuleIsBusy (USB_MODULE_ID index)

PLIB_USB_PingPongFreeze Function

Resets all Ping-Pong buffer pointers to even buffers.

File

plib_usb.h

C

```
void PLIB_USB_PingPongFreeze(USB_MODULE_ID index);
```

Returns

None.

Description

This function resets all Ping-Pong buffer pointers to even buffers.

Remarks

Buffers remain "frozen" at "Even" until they are unfrozen using PLIB_USB_PingPongUnfreeze.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBufferFreeze in your application to determine whether this feature is available.

Preconditions

None.

Example

```
  // Reset all ping-pong buffers to "Even"
  PLIB_USB_PingPongFreeze(MY_USB_INSTANCE);
  PLIB_USB_PingPongUnfreeze(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PingPongFreeze (USB_MODULE_ID index)

PLIB_USB_PingPongReset Function

Resets the USB peripheral internal Ping-Pong indicator to point to even buffers.

File

plib_usb.h

C

```
void PLIB_USB_PingPongReset(USB_MODULE_ID index);
```

Returns

None.

Description

This function resets the USB peripheral internal Ping-Pong indicator to point to Even buffers.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBufferFreeze in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_USB_PingPongReset(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PingPongReset (USB_MODULE_ID index)

PLIB_USB_PingPongUnfreeze Function

Enables Ping-Pong buffering.

File

plib_usb.h

C

```
void PLIB_USB_PingPongUnfreeze(USB_MODULE_ID index);
```

Returns

None.

Description

This function enables Ping-Pong buffering.

Remarks

See PLIB_USB_PingPongFreeze. This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsBufferFreeze in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Reset all Ping-Pong buffers to "Even"
PLIB_USB_PingPongFreeze(MY_USB_INSTANCE);
PLIB_USB_PingPongUnfreeze(MY_USB_INSTANCE);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

void PLIB_USB_PingPongUnfreeze (USB_MODULE_ID index)

PLIB_USB_TokenSend Function

Sends token to the specified address.

File

plib_usb.h

C

```
void PLIB_USB_TokenSend(USB_MODULE_ID index, USB_PID pidValue, uint8_t endpoint, uint8_t deviceAddress,
bool isLowSpeed);
```

Returns

None.

Description

This function sends the specified token to the specified endpoint and address. The token is placed on the bus at the next available time. The token can be executed at low speed.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_USB_ExistsGEN_Interrupt in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Send an OUT token to endpoint 1 device address 2 at full speed
PLIB_USB_SendToken(MY_USB_INSTANCE, USB_PID_OUT, 1, 2, false);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
pidValue	PID of the token to be placed on the bus.
endpoint	Device endpoint to which the token should be sent.
isLowSpeed	Is true if the token should be executed at low speed.

Function

void PLIB_USB_TokenSend(USB_MODULE_ID index, USB_PID pidValue,

uint8_t endpoint, uint8_t deviceAddress, bool isLowSpeed);

p) Feature Existence Functions

PLIB_USB_ExistsActivityPending Function

Identifies whether the ActivityPending feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsActivityPending(USB_MODULE_ID index);
```

Returns

- true - The ActivityPending feature is supported on the device
- false - The ActivityPending feature is not supported on the device

Description

This function identifies whether the ActivityPending feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_ActivityPending

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsActivityPending(USB_MODULE_ID index)

PLIB_USB_ExistsALL_Interrupt Function

Identifies whether the ALL_Interrupt feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsALL_Interrupt(USB_MODULE_ID index);
```

Returns

- true - The ALL_Interrupt feature is supported on the device
- false - The ALL_Interrupt feature is not supported on the device

Description

This function identifies whether the ALL_Interrupt feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_AllInterruptEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsALL_Interrupt(USB_MODULE_ID index)

PLIB_USB_ExistsAutomaticSuspend Function

Identifies whether the AutomaticSuspend feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsAutomaticSuspend(USB_MODULE_ID index);
```

Returns

- true - The AutomaticSuspend feature is supported on the device
- false - The AutomaticSuspend feature is not supported on the device

Description

This function identifies whether the AutomaticSuspend feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_AutoSuspendDisable

- PLIB_USB_AutoSuspendEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsAutomaticSuspend(USB_MODULE_ID index)

PLIB_USB_ExistsBDTBaseAddress Function

Identifies whether the BDTBaseAddress feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsBDTBaseAddress(USB_MODULE_ID index);
```

Returns

- true - The BDTBaseAddress feature is supported on the device
- false - The BDTBaseAddress feature is not supported on the device

Description

This function identifies whether the BDTBaseAddress feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_BDTBaseAddressGet
- PLIB_USB_BDTBaseAddressSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsBDTBaseAddress(USB_MODULE_ID index)

PLIB_USB_ExistsBDTFunctions Function

Identifies whether the BDTFunctions feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsBDTFunctions(USB_MODULE_ID index);
```

Returns

- true - The BDTFunctions feature is supported on the device
- false - The BDTFunctions feature is not supported on the device

Description

This function identifies whether the BDTFunctions feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_BufferAddressGet
- PLIB_USB_BufferAddressSet
- PLIB_USB_BufferByteCountGet
- PLIB_USB_BufferByteCountSet
- PLIB_USB_BufferCancelReleaseToUSB
- PLIB_USB_BufferAllCancelReleaseToUSB
- PLIB_USB_BufferClearAll
- PLIB_USB_BufferDataToggleGet
- PLIB_USB_BufferDataToggleSelect
- PLIB_USB_BufferDataToggleSyncEnable
- PLIB_USB_BufferDataToggleSyncDisable
- PLIB_USB_BufferIndexGet
- PLIB_USB_BufferPIDBitsClear
- PLIB_USB_BufferPIDGet
- PLIB_USB_BufferReleasedToSW
- PLIB_USB_BufferReleaseToUSB
- PLIB_USB_BufferSchedule
- PLIB_USB_BufferStallDisable
- PLIB_USB_BufferStallEnable
- PLIB_USB_BufferStallGet
- PLIB_USB_BufferEP0RxStatusInitialize
- PLIB_USB_BufferClearAllDTSEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsBDTFunctions(USB_MODULE_ID index)

PLIB_USB_ExistsBufferFreeze Function

Identifies whether the BufferFreeze feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsBufferFreeze(USB_MODULE_ID index);
```

Returns

- true - The BufferFreeze feature is supported on the device
- false - The BufferFreeze feature is not supported on the device

Description

This function identifies whether the BufferFreeze feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_PingPongFreeze
- PLIB_USB_PingPongUnfreeze
- PLIB_USB_PingPongReset

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsBufferFreeze(USB_MODULE_ID index)

PLIB_USB_ExistsDeviceAddress Function

Identifies whether the DeviceAddress feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsDeviceAddress(USB_MODULE_ID index);
```

Returns

- true - The DeviceAddress feature is supported on the device
- false - The DeviceAddress feature is not supported on the device

Description

This function identifies whether the DeviceAddress feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_DeviceAddressSet
- PLIB_USB_DeviceAddressGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsDeviceAddress(USB_MODULE_ID index)

PLIB_USB_ExistsEP0LowSpeedConnect Function

Identifies whether the EP0LowSpeedConnect feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsEP0LowSpeedConnect(USB_MODULE_ID index);
```

Returns

- true - The EP0LowSpeedConnect feature is supported on the device
- false - The EP0LowSpeedConnect feature is not supported on the device

Description

This function identifies whether the EP0LowSpeedConnect feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_EP0LSDirectConnectEnable
- PLIB_USB_EP0LSDirectConnectDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsEP0LowSpeedConnect(USB_MODULE_ID index)

PLIB_USB_ExistsEP0NAKRetry Function

Identifies whether the EP0NAKRetry feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsEP0NAKRetry(USB_MODULE_ID index);
```

Returns

- true - The EP0NAKRetry feature is supported on the device
- false - The EP0NAKRetry feature is not supported on the device

Description

This function identifies whether the EP0NAKRetry feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_EP0NakRetryEnable
- PLIB_USB_EP0NakRetryDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsEP0NAKRetry(USB_MODULE_ID index)

PLIB_USB_ExistsEPnRxEnable Function

Identifies whether the EPnRxEnableEnhanced feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsEPnRxEnable(USB_MODULE_ID index);
```

Returns

- true - The EPnRxEnableEnhanced feature is supported on the device
- false - The EPnRxEnableEnhanced feature is not supported on the device

Description

This function identifies whether the EPnRxEnableEnhanced feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_EPnRxEnable
- PLIB_USB_EPnRxDisable
- PLIB_USB_EPnTxEnable
- PLIB_USB_EPnTxDisable
- PLIB_USB_EPnHandshakeEnable
- PLIB_USB_EPnHandshakeDisable
- PLIB_USB_EPnControlTransferEnable
- PLIB_USB_EPnControlTransferDisable
- PLIB_USB_EPnIsStalled
- PLIB_USB_EPnStallClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsEPnRxEnable(USB_MODULE_ID index)

PLIB_USB_ExistsEPnTxRx Function

Identifies whether the EPnTxRx feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsEPnTxRx(USB_MODULE_ID index);
```

Returns

- true - The EPnTxRx feature is supported on the device
- false - The EPnTxRx feature is not supported on the device

Description

This function identifies whether the EPnTxRx feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_EPnTxSelect
- PLIB_USB_EPnRxSelect
- PLIB_USB_EPnTxRxSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsEPnTxRx(USB_MODULE_ID index)

PLIB_USB_ExistsERR_Interrupt Function

Identifies whether the ERR_Interrupt feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsERR_Interrupt(USB_MODULE_ID index);
```

Returns

- true - The ERR_Interrupt feature is supported on the device
- false - The ERR_Interrupt feature is not supported on the device

Description

This function identifies whether the ERR_Interrupt feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_ErrorInterruptEnable
- PLIB_USB_ErrorInterruptDisable
- PLIB_USB_ErrorInterruptIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsERR_Interrupt(USB_MODULE_ID index)

PLIB_USB_ExistsERR_InterruptStatus Function

Identifies whether the ERR_InterruptStatus feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsERR_InterruptStatus(USB_MODULE_ID index);
```

Returns

- true - The ERR_InterruptStatus feature is supported on the device
- false - The ERR_InterruptStatus feature is not supported on the device

Description

This function identifies whether the ERR_InterruptStatus feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_ErrorInterruptFlagSet
- PLIB_USB_ErrorInterruptFlagClear

- PLIB_USB_ErrorInterruptFlagGet
- PLIB_USB_ErrorInterruptFlagAllGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsERR_InterruptStatus(USB_MODULE_ID index)

PLIB_USB_ExistsEyePattern Function

Identifies whether the EyePattern feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsEyePattern(USB_MODULE_ID index);
```

Returns

- true - The EyePattern feature is supported on the device
- false - The EyePattern feature is not supported on the device

Description

This function identifies whether the EyePattern feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_EyePatternDisable
- PLIB_USB_EyePatternEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsEyePattern(USB_MODULE_ID index)

PLIB_USB_ExistsFrameNumber Function

Identifies whether the FrameNumber feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsFrameNumber(USB_MODULE_ID index);
```

Returns

- true - The FrameNumber feature is supported on the device

- false - The FrameNumber feature is not supported on the device

Description

This function identifies whether the FrameNumber feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_FrameNumberGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsFrameNumber(USB_MODULE_ID index)

PLIB_USB_ExistsGEN_Interrupt Function

Identifies whether the GEN_Interrupt feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsGEN_Interrupt(USB_MODULE_ID index);
```

Returns

- true - The GEN_Interrupt feature is supported on the device
- false - The GEN_Interrupt feature is not supported on the device

Description

This function identifies whether the GEN_Interrupt feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_InterruptEnable
- PLIB_USB_InterruptDisable
- PLIB_USB_InterruptIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsGEN_Interrupt(USB_MODULE_ID index)

PLIB_USB_ExistsGEN_InterruptStatus Function

Identifies whether the GEN_InterruptStatus feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsGEN_InterruptStatus(USB_MODULE_ID index);
```

Returns

- true - The GEN_InterruptStatus feature is supported on the device
- false - The GEN_InterruptStatus feature is not supported on the device

Description

This function identifies whether the GEN_InterruptStatus feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_InterruptFlagSet
- PLIB_USB_InterruptFlagClear
- PLIB_USB_InterruptFlagGet
- PLIB_USB_InterruptFlagAllGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsGEN_InterruptStatus(USB_MODULE_ID index)

PLIB_USB_ExistsHostBusyWithToken Function

Identifies whether the HostBusyWithToken feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsHostBusyWithToken(USB_MODULE_ID index);
```

Returns

- true - The HostBusyWithToken feature is supported on the device
- false - The HostBusyWithToken feature is not supported on the device

Description

This function identifies whether the HostBusyWithToken feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_IsBusyWithToken

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsHostBusyWithToken(USB_MODULE_ID index)

PLIB_USB_ExistsHostGeneratesReset Function

Identifies whether the HostGeneratesReset feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsHostGeneratesReset(USB_MODULE_ID index);
```

Returns

- true - The HostGeneratesReset feature is supported on the device
- false - The HostGeneratesReset feature is not supported on the device

Description

This function identifies whether the HostGeneratesReset feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_ResetSignalEnable
- PLIB_USB_ResetSignalDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsHostGeneratesReset(USB_MODULE_ID index)

PLIB_USB_ExistsLastDirection Function

Identifies whether the LastDirection feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsLastDirection(USB_MODULE_ID index);
```

Returns

- true - The LastDirection feature is supported on the device
- false - The LastDirection feature is not supported on the device

Description

This function identifies whether the LastDirection feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_LastTransactionDirectionGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsLastDirection(USB_MODULE_ID index)

PLIB_USB_ExistsLastEndpoint Function

Identifies whether the LastEndpoint feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsLastEndpoint(USB_MODULE_ID index);
```

Returns

- true - The LastEndpoint feature is supported on the device
- false - The LastEndpoint feature is not supported on the device

Description

This function identifies whether the LastEndpoint feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_LastTransactionEndPtGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsLastEndpoint(USB_MODULE_ID index)

PLIB_USB_ExistsLastPingPong Function

Identifies whether the LastPingPong feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsLastPingPong(USB_MODULE_ID index);
```

Returns

- true - The LastPingPong feature is supported on the device
- false - The LastPingPong feature is not supported on the device

Description

This function identifies whether the LastPingPong feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_LastTransactionPingPongStateGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsLastPingPong(USB_MODULE_ID index)

PLIB_USB_ExistsLastTransactionDetails Function

Identifies whether the LastTransactionDetails feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsLastTransactionDetails(USB_MODULE_ID index);
```

Returns

- true - The LastTransactionDetails feature is supported on the device
- false - The LastTransactionDetails feature is not supported on the device

Description

This function identifies whether the LastTransactionDetails feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_LastTransactionDetailsGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsLastTransactionDetails(USB_MODULE_ID index)

PLIB_USB_ExistsLiveJState Function

Identifies whether the LiveJState feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsLiveJState(USB_MODULE_ID index);
```

Returns

- true - The LiveJState feature is supported on the device
- false - The LiveJState feature is not supported on the device

Description

This function identifies whether the LiveJState feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_JStateIsActive

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsLiveJState(USB_MODULE_ID index)

PLIB_USB_ExistsLiveSingleEndedZero Function

Identifies whether the LiveSingleEndedZero feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsLiveSingleEndedZero(USB_MODULE_ID index);
```

Returns

- true - The LiveSingleEndedZero feature is supported on the device
- false - The LiveSingleEndedZero feature is not supported on the device

Description

This function identifies whether the LiveSingleEndedZero feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_SE0InProgress

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsLiveSingleEndedZero(USB_MODULE_ID index)

PLIB_USB_ExistsModuleBusy Function

Identifies whether the ModuleBusy feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsModuleBusy(USB_MODULE_ID index);
```

Returns

- true - The ModuleBusy feature is supported on the device
- false - The ModuleBusy feature is not supported on the device

Description

This function identifies whether the ModuleBusy feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_ModuleIsBusy

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsModuleBusy(USB_MODULE_ID index)

PLIB_USB_ExistsModulePower Function

Identifies whether the ModulePower feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsModulePower(USB_MODULE_ID index);
```

Returns

- true - The ModulePower feature is supported on the device
- false - The ModulePower feature is not supported on the device

Description

This function identifies whether the ModulePower feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_Enable
- PLIB_USB_Disable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsModulePower(USB_MODULE_ID index)

PLIB_USB_ExistsNextTokenSpeed Function

Identifies whether the NextTokenSpeed feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsNextTokenSpeed(USB_MODULE_ID index);
```

Returns

- true - The NextTokenSpeed feature is supported on the device
- false - The NextTokenSpeed feature is not supported on the device

Description

This function identifies whether the NextTokenSpeed feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_TokenSpeedSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsNextTokenSpeed(USB_MODULE_ID index)

PLIB_USB_ExistsOnChipPullup Function

Identifies whether the OnChipPullup feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOnChipPullup(USB_MODULE_ID index);
```

Returns

- true - The OnChipPullup feature is supported on the device
- false - The OnChipPullup feature is not supported on the device

Description

This function identifies whether the OnChipPullup feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_OnChipPullUpDisable
- PLIB_USB_OnChipPullUpEnable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOnChipPullup(USB_MODULE_ID index)

PLIB_USB_ExistsOnChipTransceiver Function

Identifies whether the OnChipTransceiver feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOnChipTransceiver(USB_MODULE_ID index);
```

Returns

- true - The OnChipTransceiver feature is supported on the device
- false - The OnChipTransceiver feature is not supported on the device

Description

This function identifies whether the OnChipTransceiver feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_TransceiverEnable
- PLIB_USB_TransceiverDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOnChipTransceiver(USB_MODULE_ID index)

PLIB_USB_ExistsOpModeSelect Function

Identifies whether the OpModeSelect feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOpModeSelect(USB_MODULE_ID index);
```

Returns

- true - The OpModeSelect feature is supported on the device
- false - The OpModeSelect feature is not supported on the device

Description

This function identifies whether the OpModeSelect feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OperatingModeSelect

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOpModeSelect(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_ASessionValid Function

Identifies whether the OTG_ASessionValid feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_ASessionValid(USB_MODULE_ID index);
```

Returns

- true - The OTG_ASessionValid feature is supported on the device
- false - The OTG_ASessionValid feature is not supported on the device

Description

This function identifies whether the OTG_ASessionValid feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OTG_VBusValid

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_ASessionValid(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_BSessionEnd Function

Identifies whether the OTG_BSessionEnd feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_BSessionEnd(USB_MODULE_ID index);
```

Returns

- true - The OTG_BSessionEnd feature is supported on the device
- false - The OTG_BSessionEnd feature is not supported on the device

Description

This function identifies whether the OTG_BSessionEnd feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OTG_BSessionHasEnded

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_BSessionEnd(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_IDPinState Function

Identifies whether the OTG_IDPinState feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_IDPinState(USB_MODULE_ID index);
```

Returns

- true - The OTG_IDPinState feature is supported on the device
- false - The OTG_IDPinState feature is not supported on the device

Description

This function identifies whether the OTG_IDPinState feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OTG_IDPinStateIsTypeA

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_IDPinState(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_Interrupt Function

Identifies whether the OTG_Interrupt feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_Interrupt(USB_MODULE_ID index);
```

Returns

- true - The OTG_Interrupt feature is supported on the device
- false - The OTG_Interrupt feature is not supported on the device

Description

This function identifies whether the OTG_Interrupt feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_OTG_InterruptEnable
- PLIB_USB_OTG_InterruptDisable
- PLIB_USB_OTG_InterruptIsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_Interrupt(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_InterruptStatus Function

Identifies whether the OTG_InterruptStatus feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_InterruptStatus(USB_MODULE_ID index);
```

Returns

- true - The OTG_InterruptStatus feature is supported on the device
- false - The OTG_InterruptStatus feature is not supported on the device

Description

This function identifies whether the OTG_InterruptStatus feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_OTG_InterruptFlagSet
- PLIB_USB_OTG_InterruptFlagClear
- PLIB_USB_OTG_InterruptFlagGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_InterruptStatus(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_LineState Function

Identifies whether the OTG_LineState feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_LineState(USB_MODULE_ID index);
```

Returns

- true - The OTG_LineState feature is supported on the device
- false - The OTG_LineState feature is not supported on the device

Description

This function identifies whether the OTG_LineState feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OTG_LineStateIsStable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_LineState(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_PullUpPullDown Function

Identifies whether the OTG_PullUpPullDown feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_PullUpPullDown(USB_MODULE_ID index);
```

Returns

- true - The OTG_PullUpPullDown feature is supported on the device
- false - The OTG_PullUpPullDown feature is not supported on the device

Description

This function identifies whether the OTG_PullUpPullDown feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OTG_PullUpPullDownSetup

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_PullUpPullDown(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_SessionValid Function

Identifies whether the OTG_SessionValid feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_SessionValid(USB_MODULE_ID index);
```

Returns

- true - The OTG_SessionValid feature is supported on the device
- false - The OTG_SessionValid feature is not supported on the device

Description

This function identifies whether the OTG_SessionValid feature is available on the USB module. When this function returns true, this function is supported on the device:

- PLIB_USB_OTG_SessionValid

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_SessionValid(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_VbusCharge Function

Identifies whether the OTG_VbusCharge feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_VbusCharge(USB_MODULE_ID index);
```

Returns

- true - The OTG_VbusCharge feature is supported on the device
- false - The OTG_VbusCharge feature is not supported on the device

Description

This function identifies whether the OTG_VbusCharge feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_OTG_VBusChargeEnable
- PLIB_USB_OTG_VBusChargeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_VbusCharge(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_VbusDischarge Function

Identifies whether the OTG_VbusDischarge feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_VbusDischarge(USB_MODULE_ID index);
```

Returns

- true - The OTG_VbusDischarge feature is supported on the device
- false - The OTG_VbusDischarge feature is not supported on the device

Description

This function identifies whether the OTG_VbusDischarge feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_OTG_VBusDischargeEnable
- PLIB_USB_OTG_VBusDischargeDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_VbusDischarge(USB_MODULE_ID index)

PLIB_USB_ExistsOTG_VbusPowerOnOff Function

Identifies whether the OTG_VbusPowerOnOff feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsOTG_VbusPowerOnOff(USB_MODULE_ID index);
```

Returns

- true - The OTG_VbusPowerOnOff feature is supported on the device
- false - The OTG_VbusPowerOnOff feature is not supported on the device

Description

This function identifies whether the OTG_VbusPowerOnOff feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_OTG_VBusPowerOff
- PLIB_USB_OTG_VBusPowerOn

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsOTG_VbusPowerOnOff(USB_MODULE_ID index)

PLIB_USB_ExistsPacketTransfer Function

Identifies whether the PacketTransfer feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsPacketTransfer(USB_MODULE_ID index);
```

Returns

- true - The PacketTransfer feature is supported on the device
- false - The PacketTransfer feature is not supported on the device

Description

This function identifies whether the PacketTransfer feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_PacketTransferIsDisabled
- PLIB_USB_PacketTransferEnable
- PLIB_USB_PacketTransferDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsPacketTransfer(USB_MODULE_ID index)

PLIB_USB_ExistsPingPongMode Function

Identifies whether the PingPongMode feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsPingPongMode(USB_MODULE_ID index);
```

Returns

- true - The PingPongMode feature is supported on the device
- false - The PingPongMode feature is not supported on the device

Description

This function identifies whether the PingPongMode feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_PingPongModeSelect
- PLIB_USB_PingPongModeGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsPingPongMode(USB_MODULE_ID index)

PLIB_USB_ExistsResumeSignaling Function

Identifies whether the ResumeSignaling feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsResumeSignaling(USB_MODULE_ID index);
```

Returns

- true - The ResumeSignaling feature is supported on the device
- false - The ResumeSignaling feature is not supported on the device

Description

This function identifies whether the ResumeSignaling feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_ResumeSignalingEnable
- PLIB_USB_ResumeSignalingDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsResumeSignaling(USB_MODULE_ID index)

PLIB_USB_ExistsSleepEntryGuard Function

Identifies whether the SleepEntryGuard feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsSleepEntryGuard(USB_MODULE_ID index);
```

Returns

- true - The SleepEntryGuard feature is supported on the device
- false - The SleepEntryGuard feature is not supported on the device

Description

This function identifies whether the SleepEntryGuard feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_SleepGuardEnable
- PLIB_USB_SleepGuardDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsSleepEntryGuard(USB_MODULE_ID index)

PLIB_USB_ExistsSOFThreshold Function

Identifies whether the SOFThreshold feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsSOFThreshold(USB_MODULE_ID index);
```

Returns

- true - The SOFThreshold feature is supported on the device
- false - The SOFThreshold feature is not supported on the device

Description

This function identifies whether the SOFThreshold feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_SOFThresholdGet
- PLIB_USB_SOFThresholdSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsSOFThreshold(USB_MODULE_ID index)

PLIB_USB_ExistsSpeedControl Function

Identifies whether the SpeedControl feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsSpeedControl(USB_MODULE_ID index);
```

Returns

- true - The SpeedControl feature is supported on the device
- false - The SpeedControl feature is not supported on the device

Description

This function identifies whether the SpeedControl feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_FullSpeedEnable
- PLIB_USB_FullSpeedDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsSpeedControl(USB_MODULE_ID index)

PLIB_USB_ExistsStartOfFrames Function

Identifies whether the StartOfFrames feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsStartOfFrames(USB_MODULE_ID index);
```

Returns

- true - The StartOfFrames feature is supported on the device
- false - The StartOfFrames feature is not supported on the device

Description

This function identifies whether the StartOfFrames feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_SOFEnable
- PLIB_USB_SOFDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsStartOfFrames(USB_MODULE_ID index)

PLIB_USB_ExistsStopInIdle Function

Identifies whether the StopInIdle feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsStopInIdle(USB_MODULE_ID index);
```

Returns

- true - The StopInIdle feature is supported on the device
- false - The StopInIdle feature is not supported on the device

Description

This function identifies whether the StopInIdle feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_StopInIdleEnable
- PLIB_USB_StopInIdleDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsStopInIdle(USB_MODULE_ID index)

PLIB_USB_ExistsSuspend Function

Identifies whether the Suspend feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsSuspend(USB_MODULE_ID index);
```

Returns

- true - The Suspend feature is supported on the device
- false - The Suspend feature is not supported on the device

Description

This function identifies whether the Suspend feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_SuspendEnable
- PLIB_USB_SuspendDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsSuspend(USB_MODULE_ID index)

PLIB_USB_ExistsTokenEP Function

Identifies whether the TokenEP feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsTokenEP(USB_MODULE_ID index);
```

Returns

- true - The TokenEP feature is supported on the device
- false - The TokenEP feature is not supported on the device

Description

This function identifies whether the TokenEP feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_TokenEPGet
- PLIB_USB_TokenEPSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsTokenEP(USB_MODULE_ID index)

PLIB_USB_ExistsTokenPID Function

Identifies whether the TokenPID feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsTokenPID(USB_MODULE_ID index);
```

Returns

- true - The TokenPID feature is supported on the device
- false - The TokenPID feature is not supported on the device

Description

This function identifies whether the TokenPID feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_TokenPIDGet
- PLIB_USB_TokenPIDSet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsTokenPID(USB_MODULE_ID index)

PLIB_USB_ExistsUOEMonitor Function

Identifies whether the UOEMonitor feature exists on the USB module.

File

plib_usb.h

C

```
bool PLIB_USB_ExistsUOEMonitor(USB_MODULE_ID index);
```

Returns

- true - The UOEMonitor feature is supported on the device
- false - The UOEMonitor feature is not supported on the device

Description

This function identifies whether the UOEMonitor feature is available on the USB module. When this function returns true, these functions are supported on the device:

- PLIB_USB_UOEMonitorEnable
- PLIB_USB_UOEMonitorDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USB_ExistsUOEMonitor(USB_MODULE_ID index)

q) Data Types and Constants

USB_BUFFER_DATA01 Enumeration

Provides enumeration data toggle for a buffer.

File

plib_usb.h

C

```
typedef enum {
  USB_BUFFER_DATA0,
  USB_BUFFER_DATA1
} USB_BUFFER_DATA01;
```

Members

Members	Description
USB_BUFFER_DATA0	DATA0/1 = 0
USB_BUFFER_DATA1	DATA0/1 = 1

Description

USB Endpoint Buffer Data Toggle Enumeration

This data type provides enumeration data toggle for a buffer.

Remarks

None.

USB_BUFFER_DIRECTION Enumeration

Provides enumeration transmit/receive direction for a buffer.

File

plib_usb.h

C

```
typedef enum {
  USB_BUFFER_RX,
  USB_BUFFER_TX
} USB_BUFFER_DIRECTION;
```

Members

Members	Description
USB_BUFFER_RX	Receive
USB_BUFFER_TX	Transmit

Description

USB Endpoint Buffer Direction Enumeration

This data type provides enumeration transmit/receive direction for a buffer.

Remarks

None.

USB_BUFFER_PING_PONG Enumeration

Enumerates the ping-pong buffer (Even vs. Odd).

File

plib_usb.h

C

```
typedef enum {
  USB_BUFFER_EVEN,
  USB_BUFFER_ODD
} USB_BUFFER_PING_PONG;
```

Members

Members	Description
USB_BUFFER_EVEN	Even Buffer
USB_BUFFER_ODD	Odd Buffer

Description

Enumeration of USB Buffer Ping-Pong

This data type enumerates the ping-pong buffer (Even vs. Odd).

Remarks

None.

USB_BUFFER_SCHEDULE_DATA01 Enumeration

Provides enumeration data toggle for a buffer.

File

plib_usb.h

C

```
typedef enum {
  USB_BUFFER_DONTCHANGE,
  USB_BUFFER_SET_DATA0,
```

```
  USB_BUFFER_SET_DATA1
} USB_BUFFER_SCHEDULE_DATA01;
```

Members

Members	Description
USB_BUFFER_DONTCHANGE	Don't Change DATA0/1
USB_BUFFER_SET_DATA0	DATA0/1 = 0
USB_BUFFER_SET_DATA1	DATA0/1 = 1

Description

USB Endpoint Buffer Data Toggle Enumeration for Buffer Schedulint

This data type provides enumeration data toggle for a buffer.

Remarks

None.

USB_EP_TXRX Enumeration

Provides enumeration transmit/receive setup for an endpoint.

File

plib_usb.h

C

```
typedef enum {
  USB_EP_NOTXRX,
  USB_EP_RX,
  USB_EP_TX,
  USB_EP_TX_RX
} USB_EP_TXRX;
```

Members

Members	Description
USB_EP_NOTXRX	Nothing enabled for endpoint
USB_EP_RX	Receive enabled for endpoint
USB_EP_TX	Transmit enabled for endpoint
USB_EP_TX_RX	Transmit and Receive enabled for endpoint

Description

Enumeration of USB Endpoint Transmit/Receive Setup

This data type provides enumeration transmit/receive setup for an endpoint.

Remarks

None.

USB_OPMODES Enumeration

Provides enumeration of operating modes supported by USB.

File

plib_usb.h

C

```
typedef enum {
  USB_OPMODE_NONE,
  USB_OPMODE_DEVICE,
  USB_OPMODE_HOST,
  USB_OPMODE_OTG
} USB_OPMODES;
```

Members

Members	Description
USB_OPMODE_NONE	None
USB_OPMODE_DEVICE	Device
USB_OPMODE_HOST	Host
USB_OPMODE_OTG	OTG

Description

USB Operating Modes Enumeration

This data type provides enumeration of the operating modes supported by the USB module.

Remarks

None.

USB_OTG_INTERRUPTS Enumeration

Provides enumeration of interrupts related to the USB On-The-Go (OTG) module.

File

plib_usb.h

C

```
typedef enum {
  USB_OTG_INT_ADEVICE_VBUS_VALID,
  USB_OTG_INT_OTG_RESERVED,
  USB_OTG_INT_BDEVICE_SESSION_END,
  USB_OTG_INT_SESSION_VALID,
  USB_OTG_INT_ACTIVITY_DETECT,
  USB_OTG_INT_STABLE_LINE_STATE,
  USB_OTG_INT_ONE_MS_TIMEOUT,
  USB_OTG_INT_ID_STATE_CHANGE,
  USB_OTG_INT_ANY,
  USB_OTG_INT_ALL
} USB_OTG_INTERRUPTS;
```

Members

Members	Description
USB_OTG_INT_ADEVICE_VBUS_VALID	State of (VBUS > Va_vbus_vld) on the A device has changed
USB_OTG_INT_OTG_RESERVED	Reserved. Don't use.
USB_OTG_INT_BDEVICE_SESSION_END	State of (VBUS < Vb_sess_end) on the B device has changed
USB_OTG_INT_SESSION_VALID	State of (VBUS > Va_sess_vld) on the A or B devices has changed
USB_OTG_INT_ACTIVITY_DETECT	Activity detected on the D+, D-, ID, or VBUS lines
USB_OTG_INT_STABLE_LINE_STATE	USB line state has been stable for 1 ms, but different from last time
USB_OTG_INT_ONE_MS_TIMEOUT	One millisecond timer has expired
USB_OTG_INT_ID_STATE_CHANGE	Change in state of ID pin detected.
USB_OTG_INT_ANY	All or Any of the above
USB_OTG_INT_ALL	All or Any of the above

Description

USB OTG Interrupts Enumeration

This data type provides enumeration of interrupts related to the USB OTG module.

Remarks

Not applicable if the USB OTG module is not enabled.

USB_OTG_PULL_UP_PULL_DOWN Enumeration

USB OTG pull-Up and pull-Down resistors for D+ and D- .

File

plib_usb.h

C

```
typedef enum {
  USB_OTG_DPLUS_PULLUP,
  USB_OTG_DMINUS_PULLUP,
  USB_OTG_DPLUS_PULLDN,
  USB_OTG_DMINUS_PULLDN
} USB_OTG_PULL_UP_PULL_DOWN;
```

Members

Members	Description
USB_OTG_DPLUS_PULLUP	D+ Pull-Up
USB_OTG_DMINUS_PULLUP	D- Pull-Up
USB_OTG_DPLUS_PULLDN	D+ Pull-Down
USB_OTG_DMINUS_PULLDN	D- Pull-Down

Description

Enumeration of Pull-Up and Pull-Down Resistors for OTG

This data type enumerates the OTG Pull-Up and Pull-Down resistors for D+ and D- .

Remarks

None.

USB_PID Enumeration

Legal PID values.

File

plib_usb.h

C

```
typedef enum {
  USB_PID_SETUP,
  USB_PID_IN,
  USB_PID_OUT
} USB_PID;
```

Members

Members	Description
USB_PID_SETUP	Setup token
USB_PID_IN	IN token
USB_PID_OUT	OUT token

Description

Enumeration of Legal Packet IDs (PIDs)

This data type enumerates the valid (i.e., legal) PID values. While the PID field is four bits long, only these values are legal and should be used. The use of any other values may cause unpredictable results.

USB_PING_PONG_MODE Enumeration

Supports the four modes of ping-pong buffering.

File

plib_usb.h

C

```
typedef enum {
  USB_PING_PONG_ALL_BUT_EP0,
  USB_PING_PONG_FULL_PING_PONG,
  USB_PING_PONG_EP0_OUT_ONLY,
```

```
  USB_PING_PONG_NO_PING_PONG
} USB_PING_PONG_MODE;
```

Members

Members	Description
USB_PING_PONG_ALL_BUT_EP0	Ping-Pong buffering on all endpoints except Endpoint Zero
USB_PING_PONG_FULL_PING_PONG	Ping-Pong buffering on all endpoints
USB_PING_PONG_EP0_OUT_ONLY	Ping-Pong buffering on just Endpoint Zero transmit
USB_PING_PONG_NO_PING_PONG	No ping-pong buffering

Description

Enumeration of USB Ping-Pong Modes

This data type supports the four modes of ping-pong buffering.

Remarks

None.

USB_PING_PONG_STATE Enumeration

Decodes which buffer (Even vs. Odd) was used for the last transaction.

File

plib_usb.h

C

```
typedef enum {
  USB_PING_PONG_EVEN,
  USB_PING_PONG_ODD
} USB_PING_PONG_STATE;
```

Members

Members	Description
USB_PING_PONG_EVEN	Last transaction on Even Buffer
USB_PING_PONG_ODD	Last transaction on Odd Buffer

Description

Enumeration of USB Ping-Pong Indicator

This data type decodes which buffer (Even vs. Odd) was used for the last transaction.

Remarks

None.

USB_TOKEN_SPEED Enumeration

Provides enumeration of available token speeds.

File

plib_usb.h

C

```
typedef enum {
  USB_LOWSPEED_TOKENS,
  USB_FULLSPEED_TOKENS
} USB_TOKEN_SPEED;
```

Members

Members	Description
USB_LOWSPEED_TOKENS	Low Speed Tokens
USB_FULLSPEED_TOKENS	Full Speed Tokens

Description

USB Token Speeds Enumeration

This data type provides enumeration of available token speeds.

Remarks

For Host mode only.

USB_MAX_EP_NUMBER Macro

Maximum number of endpoints supported (not including EP0).

File

plib_usb.h

C

```
#define USB_MAX_EP_NUMBER 15
```

Description

Maximum number of endpoints

This constant defines the maximum number of endpoints supported (not including EP0). It is used in dimensioning the Buffer Descriptor Table (BDT) array.

Files

Files

Name	Description
plib_usb.h	USB Peripheral Library Interface Header for common definitions

Description

This section lists the source and header files used by the library.

plib_usb.h

USB Peripheral Library Interface Header for common definitions

Enumerations

	Name	Description
	USB_BUFFER_DATA01	Provides enumeration data toggle for a buffer.
	USB_BUFFER_DIRECTION	Provides enumeration transmit/receive direction for a buffer.
	USB_BUFFER_PING_PONG	Enumerates the ping-pong buffer (Even vs. Odd).
	USB_BUFFER_SCHEDULE_DATA01	Provides enumeration data toggle for a buffer.
	USB_EP_TXRX	Provides enumeration transmit/receive setup for an endpoint.
	USB_OPMODES	Provides enumeration of operating modes supported by USB.
	USB_OTG_INTERRUPTS	Provides enumeration of interrupts related to the USB On-The-Go (OTG) module.
	USB_OTG_PULL_UP_PULL_DOWN	USB OTG pull-Up and pull-Down resistors for D+ and D- .
	USB_PID	Legal PID values.
	USB_PING_PONG_MODE	Supports the four modes of ping-pong buffering.
	USB_PING_PONG_STATE	Decodes which buffer (Even vs. Odd) was used for the last transaction.
	USB_TOKEN_SPEED	Provides enumeration of available token speeds.

Functions

	Name	Description
	PLIB_USB_ActivityPending	Returns whether or not USB activity is pending.
	PLIB_USB_AllInterruptEnable	Configures the USB peripheral general interrupts, error interrupts and OTG interrupts.
	PLIB_USB_AutoSuspendDisable	Disables USB OTG Auto-suspend mode.
	PLIB_USB_AutoSuspendEnable	Enables USB Auto-suspend mode.

	PLIB_USB_BDTBaseAddressGet	Returns the base address of the Buffer Descriptor Table.
	PLIB_USB_BDTBaseAddressSet	Sets the base address for the Buffer Descriptor Table for PIC32 devices.
	PLIB_USB_BufferAddressGet	Gets the memory address of an endpoint buffer.
	PLIB_USB_BufferAddressSet	Sets the endpoint buffer address.
	PLIB_USB_BufferAllCancelReleaseToUSB	Cancels all endpoint buffer releases to the USB module and hands over the buffer to the CPU.
	PLIB_USB_BufferByteCountGet	Returns the endpoint buffer byte count.
	PLIB_USB_BufferByteCountSet	Sets the buffer byte count.
	PLIB_USB_BufferCancelReleaseToUSB	Cancels release of the endpoint buffer by software, allowing software to again access the buffer.
	PLIB_USB_BufferClearAll	Clears (zeros out) entries in the Buffer Descriptor Table.
	PLIB_USB_BufferClearAllDTSEnable	Clears the endpoint descriptor entry and enables data toggle synchronization.
	PLIB_USB_BufferDataToggleGet	Returns data synchronization (DATA0 or DATA1) for the endpoint buffer.
	PLIB_USB_BufferDataToggleSelect	Sets the endpoint buffer to DATA0 or DATA1.
	PLIB_USB_BufferDataToggleSyncDisable	Disables DATA0/DATA1 synchronization between the device and host.
	PLIB_USB_BufferDataToggleSyncEnable	Enables DATA0/DATA1 synchronization between the device and host.
	PLIB_USB_BufferEP0RxStatusInitialize	Initializes the Endpoint 0 RX endpoint buffer descriptors.
	PLIB_USB_BufferIndexGet	Gets the Buffer Descriptor Table index for a buffer.
	PLIB_USB_BufferPIDBitsClear	Clears the Buffer Status bits in the Buffer Descriptor Table.
	PLIB_USB_BufferPIDGet	Returns the token packet ID (PID) from the endpoint buffer status.
	PLIB_USB_BufferReleasedToSW	Returns the boolean flag value of 'true' when the buffer has been released by the USB module.
	PLIB_USB_BufferReleaseToUSB	Releases the endpoint buffer by software, allowing the USB module access to the buffer.
	PLIB_USB_BufferSchedule	Hands over a buffer to the USB module along with the buffer address and byte count.
	PLIB_USB_BufferStallDisable	Disables STALL handshaking for the associated endpoint buffer.
	PLIB_USB_BufferStallEnable	Enables STALL handshaking for the associated endpoint buffer.
	PLIB_USB_BufferStallGet	Returns the buffer stall status for an endpoint/direction/ping-pong.
	PLIB_USB_DeviceAddressGet	Returns the address of the USB module in Device mode.
	PLIB_USB_DeviceAddressSet	Sets the USB Device's address.
	PLIB_USB_Disable	Disables (powers down) the USB module.
	PLIB_USB_Enable	Enables (powers up) the USB module.
	PLIB_USB_EP0HostSetup	Sends token to the specified address.
	PLIB_USB_EP0LSDirectConnectDisable	Disables direct connection to a low-speed device for Endpoint 0.
	PLIB_USB_EP0LSDirectConnectEnable	Enables direct connection to a low-speed device for Endpoint 0.
	PLIB_USB_EP0NakRetryDisable	Disables retrying of NAKed transactions.
	PLIB_USB_EP0NakRetryEnable	Enables retrying NAK'd transactions for Endpoint 0.
	PLIB_USB_EPnAttributesClear	Clears the set attributes of the specified endpoint.
	PLIB_USB_EPnAttributesSet	Configures attributes of the endpoint such as direction, handshake capability and direction.
	PLIB_USB_EPnControlTransferDisable	Disables endpoint control transfers.
	PLIB_USB_EPnControlTransferEnable	Enables endpoint control transfers.
	PLIB_USB_EPnDirectionDisable	Disables the specified endpoint direction.
	PLIB_USB_EPnHandshakeDisable	Disables endpoint handshaking.
	PLIB_USB_EPnHandshakeEnable	Enables endpoint handshaking.
	PLIB_USB_EPnIsStalled	Tests whether the endpoint epValue is stalled.
	PLIB_USB_EPnRxDisable	Disables an endpoint's ability to process IN tokens.
	PLIB_USB_EPnRxEnable	Enables an endpoint to process IN tokens.
	PLIB_USB_EPnRxSelect	Selects receive capabilities of an endpoint.
	PLIB_USB_EPnStallClear	Clears an endpoint's stalled flag.
	PLIB_USB_EPnTxDisable	Disables an endpoint's ability to process OUT tokens.
	PLIB_USB_EPnTxEnable	Enables an endpoint to process OUT tokens.
	PLIB_USB_EPnTxRxSelect	Selects transmit and/or receive capabilities of an endpoint.
	PLIB_USB_EPnTxSelect	Selects transmit capabilities of an endpoint.
	PLIB_USB_ErrorInterruptDisable	Disables an error interrupt for the USB module.

	PLIB_USB_ErrorInterruptEnable	Enables an error interrupt for the USB module.
	PLIB_USB_ErrorInterruptFlagAllGet	Returns a logically ORed bit map of active error interrupt flags.
	PLIB_USB_ErrorInterruptFlagClear	Clears an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptFlagGet	Tests an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptFlagSet	Sets an error interrupt flag for the USB module.
	PLIB_USB_ErrorInterruptIsEnabled	Returns true if interrupts are enabled.
	PLIB_USB_ExistsActivityPending	Identifies whether the ActivityPending feature exists on the USB module.
	PLIB_USB_ExistsALL_Interrupt	Identifies whether the ALL_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsAutomaticSuspend	Identifies whether the AutomaticSuspend feature exists on the USB module.
	PLIB_USB_ExistsBDTBaseAddress	Identifies whether the BDTBaseAddress feature exists on the USB module.
	PLIB_USB_ExistsBDTFunctions	Identifies whether the BDTFunctions feature exists on the USB module.
	PLIB_USB_ExistsBufferFreeze	Identifies whether the BufferFreeze feature exists on the USB module.
	PLIB_USB_ExistsDeviceAddress	Identifies whether the DeviceAddress feature exists on the USB module.
	PLIB_USB_ExistsEP0LowSpeedConnect	Identifies whether the EP0LowSpeedConnect feature exists on the USB module.
	PLIB_USB_ExistsEP0NAKRetry	Identifies whether the EP0NAKRetry feature exists on the USB module.
	PLIB_USB_ExistsEPnRxEnable	Identifies whether the EPnRxEnableEnhanced feature exists on the USB module.
	PLIB_USB_ExistsEPnTxRx	Identifies whether the EPnTxRx feature exists on the USB module.
	PLIB_USB_ExistsERR_Interrupt	Identifies whether the ERR_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsERR_InterruptStatus	Identifies whether the ERR_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsEyePattern	Identifies whether the EyePattern feature exists on the USB module.
	PLIB_USB_ExistsFrameNumber	Identifies whether the FrameNumber feature exists on the USB module.
	PLIB_USB_ExistsGEN_Interrupt	Identifies whether the GEN_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsGEN_InterruptStatus	Identifies whether the GEN_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsHostBusyWithToken	Identifies whether the HostBusyWithToken feature exists on the USB module.
	PLIB_USB_ExistsHostGeneratesReset	Identifies whether the HostGeneratesReset feature exists on the USB module.
	PLIB_USB_ExistsLastDirection	Identifies whether the LastDirection feature exists on the USB module.
	PLIB_USB_ExistsLastEndpoint	Identifies whether the LastEndpoint feature exists on the USB module.
	PLIB_USB_ExistsLastPingPong	Identifies whether the LastPingPong feature exists on the USB module.
	PLIB_USB_ExistsLastTransactionDetails	Identifies whether the LastTransactionDetails feature exists on the USB module.
	PLIB_USB_ExistsLiveJState	Identifies whether the LiveJState feature exists on the USB module.
	PLIB_USB_ExistsLiveSingleEndedZero	Identifies whether the LiveSingleEndedZero feature exists on the USB module.
	PLIB_USB_ExistsModuleBusy	Identifies whether the ModuleBusy feature exists on the USB module.
	PLIB_USB_ExistsModulePower	Identifies whether the ModulePower feature exists on the USB module.
	PLIB_USB_ExistsNextTokenSpeed	Identifies whether the NextTokenSpeed feature exists on the USB module.
	PLIB_USB_ExistsOnChipPullup	Identifies whether the OnChipPullup feature exists on the USB module.
	PLIB_USB_ExistsOnChipTransceiver	Identifies whether the OnChipTransceiver feature exists on the USB module.
	PLIB_USB_ExistsOpModeSelect	Identifies whether the OpModeSelect feature exists on the USB module.
	PLIB_USB_ExistsOTG_ASessionValid	Identifies whether the OTG_ASessionValid feature exists on the USB module.
	PLIB_USB_ExistsOTG_BSessionEnd	Identifies whether the OTG_BSessionEnd feature exists on the USB module.
	PLIB_USB_ExistsOTG_IDPinState	Identifies whether the OTG_IDPinState feature exists on the USB module.
	PLIB_USB_ExistsOTG_Interrupt	Identifies whether the OTG_Interrupt feature exists on the USB module.
	PLIB_USB_ExistsOTG_InterruptStatus	Identifies whether the OTG_InterruptStatus feature exists on the USB module.
	PLIB_USB_ExistsOTG_LineState	Identifies whether the OTG_LineState feature exists on the USB module.
	PLIB_USB_ExistsOTG_PullUpPullDown	Identifies whether the OTG_PullUpPullDown feature exists on the USB module.
	PLIB_USB_ExistsOTG_SessionValid	Identifies whether the OTG_SessionValid feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusCharge	Identifies whether the OTG_VbusCharge feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusDischarge	Identifies whether the OTG_VbusDischarge feature exists on the USB module.
	PLIB_USB_ExistsOTG_VbusPowerOnOff	Identifies whether the OTG_VbusPowerOnOff feature exists on the USB module.
	PLIB_USB_ExistsPacketTransfer	Identifies whether the PacketTransfer feature exists on the USB module.
	PLIB_USB_ExistsPingPongMode	Identifies whether the PingPongMode feature exists on the USB module.
	PLIB_USB_ExistsResumeSignaling	Identifies whether the ResumeSignaling feature exists on the USB module.
	PLIB_USB_ExistsSleepEntryGuard	Identifies whether the SleepEntryGuard feature exists on the USB module.
	PLIB_USB_ExistsSOFThreshold	Identifies whether the SOFThreshold feature exists on the USB module.
	PLIB_USB_ExistsSpeedControl	Identifies whether the SpeedControl feature exists on the USB module.
	PLIB_USB_ExistsStartOfFrames	Identifies whether the StartOfFrames feature exists on the USB module.

	PLIB_USB_ExistsStopInIdle	Identifies whether the StopInIdle feature exists on the USB module.
	PLIB_USB_ExistsSuspend	Identifies whether the Suspend feature exists on the USB module.
	PLIB_USB_ExistsTokenEP	Identifies whether the TokenEP feature exists on the USB module.
	PLIB_USB_ExistsTokenPID	Identifies whether the TokenPID feature exists on the USB module.
	PLIB_USB_ExistsUOEMonitor	Identifies whether the UOEMonitor feature exists on the USB module.
	PLIB_USB_ExternalComparatorMode2Pin	Sets the 2-pin input configuration for VBUS comparators.
	PLIB_USB_ExternalComparatorMode3Pin	Sets the 3-pin input configuration for VBUS Comparators.
	PLIB_USB_EyePatternDisable	Disables the USB eye pattern test.
	PLIB_USB_EyePatternEnable	Enables USB eye pattern test.
	PLIB_USB_FrameNumberGet	Returns the USB frame number.
	PLIB_USB_FullSpeedDisable	Forces the USB module to operate at low speed.
	PLIB_USB_FullSpeedEnable	Enables the USB to operate at full speed.
	PLIB_USB_I2CInterfaceForExtModuleDisable	Specifies external module(s) are controlled via dedicated pins.
	PLIB_USB_I2CInterfaceForExtModuleEnable	Specifies external module(s) are controlled via the I2C interface.
	PLIB_USB_InterruptDisable	Disables a general interrupt for the USB module.
	PLIB_USB_InterruptEnable	Enables a general interrupt for the USB module.
	PLIB_USB_InterruptEnableGet	Returns the enable/disable status of general USB module interrupts
	PLIB_USB_InterruptFlagAllGet	Returns a logically ORed bit map of active general USB interrupt flags.
	PLIB_USB_InterruptFlagClear	Clears a general interrupt flag for the USB module.
	PLIB_USB_InterruptFlagGet	Tests a general interrupt flag for the USB module.
	PLIB_USB_InterruptFlagSet	Sets a general interrupt flag for the USB module.
	PLIB_USB_InterruptIsEnabled	Returns true if interrupts are enabled.
	PLIB_USB_IsBusyWithToken	Indicates whether there is a token being executed by the USB module as Host.
	PLIB_USB_JStateIsActive	Live differential receiver J State flag.
	PLIB_USB_LastTransactionDetailsGet	Returns the details of the last completed transaction.
	PLIB_USB_LastTransactionDirectionGet	Indicates the direction of the last transaction.
	PLIB_USB_LastTransactionEndPtGet	Returns the endpoint number of the last USB transfer.
	PLIB_USB_LastTransactionPingPongStateGet	Indicates whether the last transaction was to an EVEN buffer or an ODD buffer.
	PLIB_USB_ModuleIsBusy	Indicates if the USB module is not ready to be enabled.
	PLIB_USB_OnChipPullUpDisable	Disables on-chip pull-ups.
	PLIB_USB_OnChipPullUpEnable	Enables on-chip pull-ups.
	PLIB_USB_OperatingModeSelect	Selects the operating mode of the USB module.
	PLIB_USB_OTG_BSessionHasEnded	Returns the status of the B-Session End Indicator bit.
	PLIB_USB_OTG_IDPinStateIsTypeA	Returns the ID Pin state.
	PLIB_USB_OTG_InterruptDisable	Disables a USB On-The-Go (OTG) Interrupt for the USB module.
	PLIB_USB_OTG_InterruptEnable	Enables a USB On-The-Go (OTG) Interrupt for the USB module.
	PLIB_USB_OTG_InterruptFlagClear	Clears a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptFlagGet	Tests a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptFlagSet	Sets a USB On-The-Go (OTG) Interrupt flag for the USB module.
	PLIB_USB_OTG_InterruptIsEnabled	Returns whether or not interrupts are enabled.
	PLIB_USB_OTG_LineStateIsStable	Returns the status of the Line Stable Indicator bit.
	PLIB_USB_OTG_PullUpPullDownSetup	Enables or disables pull-up and pull-down resistors.
	PLIB_USB_OTG_SessionValid	Returns the status of the Session Valid Indicator bit.
	PLIB_USB_OTG_VBusChargeDisable	Disables VBUS line charge.
	PLIB_USB_OTG_VBusChargeEnable	Enables the VBUS line to be charged through a pull-up resistor.
	PLIB_USB_OTG_VBusChargeTo3V	Sets the VBUS line to charge to 3.3V.
	PLIB_USB_OTG_VBusChargeTo5V	Sets the VBUS line to charge to 5V.
	PLIB_USB_OTG_VBusDischargeDisable	Disables VBUS line discharge.
	PLIB_USB_OTG_VBusDischargeEnable	Enables VBUS line to be discharged through a resistor.
	PLIB_USB_OTG_VBusPowerOff	Turns off power on the VBUS Line.
	PLIB_USB_OTG_VBusPowerOn	Turns on power for the VBUS line.
	PLIB_USB_OTG_VBusValid	Returns the status of the A-VBUS valid indicator.
	PLIB_USB_PacketTransferDisable	Disables the Serial Interface Engine (SIE).
	PLIB_USB_PacketTransferEnable	Re-enables the Serial Interface Engine (SIE), allowing token and packet processing.

	PLIB_USB_PacketTransferIsDisabled	Indicates that a setup token has been received from the Host and that token/packet processing is disabled.
	PLIB_USB_PingPongFreeze	Resets all Ping-Pong buffer pointers to even buffers.
	PLIB_USB_PingPongModeGet	Returns the Ping-Pong Configuration setting.
	PLIB_USB_PingPongModeSelect	Selects the Ping-Pong Configuration setting.
	PLIB_USB_PingPongReset	Resets the USB peripheral internal Ping-Pong indicator to point to even buffers.
	PLIB_USB_PingPongUnfreeze	Enables Ping-Pong buffering.
	PLIB_USB_PWMCounterDisable	Disables the PWM counter used to generate the VBUS for the USB module.
	PLIB_USB_PWMCounterEnable	Enables the PWM counter used to generate the VBUS for the USB module.
	PLIB_USB_PWMDisable	Disables the PWM Generator.
	PLIB_USB_PWMEnable	Enables the PWM Generator.
	PLIB_USB_PWMPolaritiyActiveLow	Sets the PWM output to active-high and resets low.
	PLIB_USB_PWMPolarityActiveHigh	Sets the PWM output to active-low and resets high.
	PLIB_USB_ResetSignalDisable	Disables reset signaling on the USB bus.
	PLIB_USB_ResetSignalEnable	Enables reset signaling on the USB bus.
	PLIB_USB_ResumeSignalingDisable	Disables resume signaling.
	PLIB_USB_ResumeSignalingEnable	Enables resume signaling.
	PLIB_USB_SE0InProgress	Returns whether a single-ended zero event is in progress.
	PLIB_USB_SleepGuardDisable	This function disables Sleep Guard. Entry into Sleep mode is immediate.
	PLIB_USB_SleepGuardEnable	Entry into Sleep mode is blocked if bus activity is detected or if an interrupt is pending.
	PLIB_USB_SOFDisable	Disables the automatic generation of the SOF token.
	PLIB_USB_SOFEnable	Enables the automatic generation of the SOF token every 1 ms.
	PLIB_USB_SOFThresholdGet	Returns the Start-of-Frame (SOF) Count bits.
	PLIB_USB_SOFThresholdSet	Sets the Start-of-Frame (SOF) threshold value.
	PLIB_USB_StopInIdleDisable	Allows the USB module to continue operation when the device enters Idle mode.
	PLIB_USB_StopInIdleEnable	Enables USB module operation to stop when the device enters Idle mode.
	PLIB_USB_SuspendDisable	Disables USB OTG Suspend mode.
	PLIB_USB_SuspendEnable	Enables USB Suspend mode.
	PLIB_USB_TokenEPGet	Returns the specified Endpoint address.
	PLIB_USB_TokenEPSet	Sets the Endpoint address for a host transaction.
	PLIB_USB_TokenPIDGet	Returns the token transaction type.
	PLIB_USB_TokenPIDSet	Sets the token transaction type to pidValue.
	PLIB_USB_TokenSend	Sends token to the specified address.
	PLIB_USB_TokenSpeedSelect	Selects low speed or full speed for subsequent token executions.
	PLIB_USB_TransceiverDisable	Disables the on-chip transceiver
	PLIB_USB_TransceiverEnable	Enables the on-chip transceiver.
	PLIB_USB_UOEMonitorDisable	Disables the OE signal output.
	PLIB_USB_UOEMonitorEnable	Enables the OE signal output.
	PLIB_USB_VBoostDisable	Disables the On-Chip 5V Boost Regulator Circuit Disabled bit.
	PLIB_USB_VBoostEnable	Enables the On-Chip 5V Boost Regulator Circuit Enabled bit.
	PLIB_USB_VBUSComparatorDisable	Disables the on-chip VBUS Comparator.
	PLIB_USB_VBUSComparatorEnable	Enables the on-chip VBUS Comparator.
	PLIB_USB_VBUSPullUpDisable	Disables the pull-up on the VBUS pin.
	PLIB_USB_VBUSPullUpEnable	Enables the pull-up on the VBUS pin.

Macros

	Name	Description
	USB_MAX_EP_NUMBER	Maximum number of endpoints supported (not including EP0).

Description

USB Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB Peripheral Library.

File Name

plib_usb.h

Company

Microchip Technology Inc.

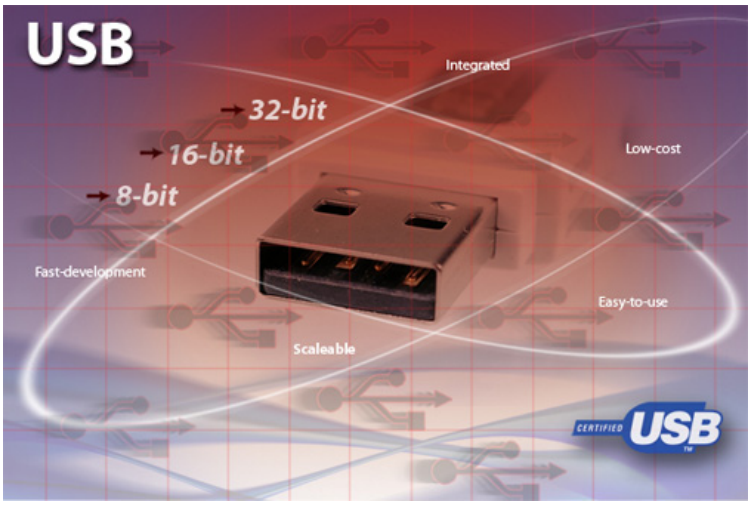
USBHS Peripheral Library

This section describes the Hi-Speed USB (USBHS) Peripheral Library.

Introduction

This library provides a low-level abstraction of the USBHS module on Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thus hiding differences from one microcontroller variant to another.

Description



USB Overview

USB is an asynchronous serial interface with a tiered star configuration. USB is implemented as a master/slave configuration. On a given bus, there can be multiple (up to 127) slaves (devices), but there is only one master (host). There are three possible module implementations: host, device and OTG dual role. The user should have an understanding of the USB documents available on the USB implementers web site (www.usb.org).

Features of the Hi-Speed USB (USBHS) Module

- Operates either as a function controller of a Hi-Speed/Full-Speed USB device or as the host/device in a point-to-point or multi-point communications with other USB function
- Supports OTG communications with on or more Hi-Speed, Full-Speed, or Low-Speed devices
- Provides soft connect/disconnect.
- In addition to Endpoint Zero, supports seven transmit and seven receive endpoints
- Dynamic FIFO sizing for Endpoints 1-7. (Endpoint Zero FIFO fixed at 64 bytes.) FIFOs use module-internal SRAM.
- Module-internal eight channel DMA with access to all FIFOs
- All host transaction scheduling supported in hardware
- Supports Link Power Management

Using the Library

This topic describes the basic architecture of the USBHS Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_usbhs.h`

The interface to the USBHS Peripheral Library is defined in the `plib_usbhs.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the USBHS Peripheral Library must include `peripheral.h`.

Library File:

The USBHS Peripheral Library is part of the processor-specific peripheral library archive (`.a`) file installed with the compiler. Libraries in this archive are automatically available to the linker (in the default search path) for any project built using the Microchip compiler.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the USBHS module.

Library Interface Section	Description
USBHS Setup Functions	This section provides functions to perform general USBHS peripheral setup such as functions to set USB Speed, control on-chip pull ups, etc.
Endpoints Functions	This section provides functions that allow the application to manage endpoints.
Interrupts Functions	This section provides functions that allow the application to enable, disable and query the status interrupts in the USBHS peripheral.
Device Functions	This section provides functions that are required to operate the USBHS module while in Device mode.
Host Functions	This section provides functions that are required to operate the USBHS module while in Host mode.
Status Functions	This section provides functions that read the status of the USBHS module.
VBUS Support Functions	This section provides functions that allow VBUS level monitoring and VBUS boost PWM module control.
Feature Existence Functions	This section provides functions that identify whether a particular feature exists on the USBHS module.

How the Library Works

Provides information on how the library works.

Configuring the Library

The library is configured for the supported High-Speed USB module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) USBHS Setup Functions

	Name	Description
	PLIB_USBHS_HighSpeedDisable	This is function PLIB_USBHS_HighSpeedDisable.
	PLIB_USBHS_HighSpeedEnable	Sets the operation speed of the USB Module.
	PLIB_USBHS_ResetDisable	This is function PLIB_USBHS_ResetDisable.
	PLIB_USBHS_ResetEnable	This is function PLIB_USBHS_ResetEnable.
	PLIB_USBHS_ResumeDisable	This is function PLIB_USBHS_ResumeDisable.
	PLIB_USBHS_ResumeEnable	This is function PLIB_USBHS_ResumeEnable.
	PLIB_USBHS_SessionDisable	This is function PLIB_USBHS_SessionDisable.
	PLIB_USBHS_SessionEnable	This is function PLIB_USBHS_SessionEnable.
	PLIB_USBHS_SoftResetDisable	Disables soft reset.
	PLIB_USBHS_SoftResetEnable	Enables soft reset.
	PLIB_USBHS_SuspendDisable	This is function PLIB_USBHS_SuspendDisable.
	PLIB_USBHS_SuspendEnable	This is function PLIB_USBHS_SuspendEnable.
	PLIB_USBHS_DMAOperationEnable	This is function PLIB_USBHS_DMAOperationEnable.
	PLIB_USBHS_TestModeEnter	This is function PLIB_USBHS_TestModeEnter.
	PLIB_USBHS_TestModeExit	This is function PLIB_USBHS_TestModeExit.
	PLIB_USBHS_PhyIDMonitoringDisable	This is function PLIB_USBHS_PhyIDMonitoringDisable.
	PLIB_USBHS_PhyIDMonitoringEnable	This is function PLIB_USBHS_PhyIDMonitoringEnable.
	PLIB_USBHS_USBIDOverrideDisable	This is function PLIB_USBHS_USBIDOverrideDisable.

	PLIB_USBHS_USBIDOverrideEnable	This is function PLIB_USBHS_USBIDOverrideEnable.
	PLIB_USBHS_USBIDOverrideValueSet	This is function PLIB_USBHS_USBIDOverrideValueSet.
	PLIB_USBHS_IsBDevice	This is function PLIB_USBHS_IsBDevice.

b) Endpoints Functions

	Name	Description
	PLIB_USBHS_Endpoint0FIFOFlush	This is function PLIB_USBHS_Endpoint0FIFOFlush.
	PLIB_USBHS_Endpoint0SetupPacketLoad	Loads the endpoint 0 FIFO with provided setup packet and then enables the endpoint transmit.
	PLIB_USBHS_Endpoint0SetupPacketUnload	This is function PLIB_USBHS_Endpoint0SetupPacketUnload.
	PLIB_USBHS_EndpointFIFOLoad	Loads the endpoint FIFO with provided data and then enables the endpoint transmit.
	PLIB_USBHS_EndpointFIFOUnload	Unloads the endpoint FIFO.
	PLIB_USBHS_EndpointRxFIFOFlush	This is function PLIB_USBHS_EndpointRxFIFOFlush.
	PLIB_USBHS_EndpointRxRequestClear	This is function PLIB_USBHS_EndpointRxRequestClear.
	PLIB_USBHS_EndpointRxRequestEnable	This is function PLIB_USBHS_EndpointRxRequestEnable.
	PLIB_USBHS_EndpointTxFIFOFlush	This is function PLIB_USBHS_EndpointTxFIFOFlush.
	PLIB_USBHS_EP0DataEndSet	This is function PLIB_USBHS_EP0DataEndSet.
	PLIB_USBHS_EP0INHandshakeClear	This is function PLIB_USBHS_EP0INHandshakeClear.
	PLIB_USBHS_EP0INHandshakeSend	This is function PLIB_USBHS_EP0INHandshakeSend.
	PLIB_USBHS_EP0INTokenSend	This is function PLIB_USBHS_EP0INTokenSend.
	PLIB_USBHS_EP0OUTHandshakeSend	This is function PLIB_USBHS_EP0OUTHandshakeSend.
	PLIB_USBHS_EP0RxPktRdyServiced	This is function PLIB_USBHS_EP0RxPktRdyServiced.
	PLIB_USBHS_EP0RxPktRdyServicedDataEnd	This is function PLIB_USBHS_EP0RxPktRdyServicedDataEnd.
	PLIB_USBHS_EP0SentStallClear	This is function PLIB_USBHS_EP0SentStallClear.
	PLIB_USBHS_EP0SetupEndServiced	This is function PLIB_USBHS_EP0SetupEndServiced.
	PLIB_USBHS_EP0StallDisable	This is function PLIB_USBHS_EP0StallDisable.
	PLIB_USBHS_EP0StallEnable	This is function PLIB_USBHS_EP0StallEnable.
	PLIB_USBHS_EP0StatusClear	This is function PLIB_USBHS_EP0StatusClear.
	PLIB_USBHS_EP0StatusGet	This is function PLIB_USBHS_EP0StatusGet.
	PLIB_USBHS_EP0TxPktRdy	This is function PLIB_USBHS_EP0TxPktRdy.
	PLIB_USBHS_EP0TxPktRdyDataEnd	This is function PLIB_USBHS_EP0TxPktRdyDataEnd.
	PLIB_USBHS_HostRxEndpointConfigure	This is function PLIB_USBHS_HostRxEndpointConfigure.
	PLIB_USBHS_HostTxEndpointConfigure	This is function PLIB_USBHS_HostTxEndpointConfigure.
	PLIB_USBHS_RxEPINTokenSend	This is function PLIB_USBHS_RxEPINTokenSend.
	PLIB_USBHS_RxEPStatusClear	This is function PLIB_USBHS_RxEPStatusClear.
	PLIB_USBHS_RxEPStatusGet	This is function PLIB_USBHS_RxEPStatusGet.
	PLIB_USBHS_TxEPStatusClear	This is function PLIB_USBHS_TxEPStatusClear.
	PLIB_USBHS_TxEPStatusGet	This is function PLIB_USBHS_TxEPStatusGet.
	PLIB_USBHS_GetEP0CSRAddress	This is function PLIB_USBHS_GetEP0CSRAddress.
	PLIB_USBHS_GetEP0FIFOAddress	This is function PLIB_USBHS_GetEP0FIFOAddress.
	PLIB_USBHS_LoadEPInIndex	This is function PLIB_USBHS_LoadEPInIndex.

c) Interrupts Functions

	Name	Description
	PLIB_USBHS_TxInterruptDisable	Disables a TX endpoint interrupt source for the USB module.
	PLIB_USBHS_TxInterruptEnable	Enables a TX endpoint Interrupt for the USB module.
	PLIB_USBHS_GenInterruptDisable	Disables a general interrupt for the USB module.
	PLIB_USBHS_GenInterruptEnable	Enables a general interrupt for the USB module.
	PLIB_USBHS_GenInterruptFlagsGet	Gets general interrupt flags.
	PLIB_USBHS_InterruptEnableSet	Enables USB module event interrupts.
	PLIB_USBHS_DMAInterruptDisable	Disables DMA channel interrupts.
	PLIB_USBHS_DMAInterruptEnable	Enables DMA channel interrupts.
	PLIB_USBHS_DMAInterruptFlagsGet	Gets the DMA channel interrupt flags.
	PLIB_USBHS_TxInterruptFlagsGet	Gets the TX endpoint interrupt flags.
	PLIB_USBHS_RxInterruptDisable	Disables a RX endpoint interrupt for the USB module.

	PLIB_USBHS_RxInterruptEnable	Enables a RX endpoint interrupt for the USB module.
	PLIB_USBHS_RxInterruptFlagsGet	Gets RX endpoint interrupt flags.
	PLIB_USBHS_DMAInterruptGet	This is function PLIB_USBHS_DMAInterruptGet.
	PLIB_USBHS_GlobalInterruptDisable	This is function PLIB_USBHS_GlobalInterruptDisable.
	PLIB_USBHS_GlobalInterruptEnable	This is function PLIB_USBHS_GlobalInterruptEnable.

d) Device Functions

	Name	Description
	PLIB_USBHS_DeviceAddressGet	Returns the current USB device address.
	PLIB_USBHS_DeviceAddressSet	Sets the device address.
	PLIB_USBHS_DeviceAttach	This is function PLIB_USBHS_DeviceAttach.
	PLIB_USBHS_DeviceConnect	Tri-states the USB D+ and D- lines.
	PLIB_USBHS_DeviceDetach	This is function PLIB_USBHS_DeviceDetach.
	PLIB_USBHS_DeviceDisconnect	Tri-states the USB D+ and D- lines.
	PLIB_USBHS_DeviceEPFIFOLoad	This is function PLIB_USBHS_DeviceEPFIFOLoad.
	PLIB_USBHS_DeviceEPFIFOUnload	This is function PLIB_USBHS_DeviceEPFIFOUnload.
	PLIB_USBHS_DeviceRxEndpointConfigure	This is function PLIB_USBHS_DeviceRxEndpointConfigure.
	PLIB_USBHS_DeviceRxEndpointStallDisable	This is function PLIB_USBHS_DeviceRxEndpointStallDisable.
	PLIB_USBHS_DeviceRxEndpointStallEnable	This is function PLIB_USBHS_DeviceRxEndpointStallEnable.
	PLIB_USBHS_DeviceTxEndpointConfigure	This is function PLIB_USBHS_DeviceTxEndpointConfigure.
	PLIB_USBHS_DeviceTxEndpointPacketReady	This is function PLIB_USBHS_DeviceTxEndpointPacketReady.
	PLIB_USBHS_DeviceTxEndpointStallDisable	This is function PLIB_USBHS_DeviceTxEndpointStallDisable.
	PLIB_USBHS_DeviceTxEndpointStallEnable	This is function PLIB_USBHS_DeviceTxEndpointStallEnable.

e) Host Functions

	Name	Description
	PLIB_USBHS_HostRxEndpointDataToggleClear	This is function PLIB_USBHS_HostRxEndpointDataToggleClear.
	PLIB_USBHS_HostTxEndpointDataToggleClear	This is function PLIB_USBHS_HostTxEndpointDataToggleClear.

f) Status Functions

	Name	Description
	PLIB_USBHS_FullOrHighSpeedIsConnected	This is function PLIB_USBHS_FullOrHighSpeedIsConnected.
	PLIB_USBHS_HighSpeedIsConnected	This is function PLIB_USBHS_HighSpeedIsConnected.
	PLIB_USBHS_HostModeIsEnabled	This is function PLIB_USBHS_HostModeIsEnabled.
	PLIB_USBHS_ModuleSpeedGet	Returns the speed at which the module is operating.
	PLIB_USBHS_DMAErrorGet	This is function PLIB_USBHS_DMAErrorGet.
	PLIB_USBHS_GetReceiveDataCount	This is function PLIB_USBHS_GetReceiveDataCount.

g) VBUS Support Functions

	Name	Description
	PLIB_USBHS_VbusLevelGet	Returns the current VBUS level encode using the USBHS_VBUS_DETECT_LEVEL enumeration.
	PLIB_USBHS_VBUSLevelGet	This is function PLIB_USBHS_VBUSLevelGet.

h) Feature Existence Functions

	Name	Description
	PLIB_USBHS_ExistsEndpointFIFO	Identifies that the Endpoint FIFO feature exists on the Hi-Speed USB module.
	PLIB_USBHS_ExistsEndpointOperations	This is function PLIB_USBHS_ExistsEndpointOperations.
	PLIB_USBHS_ExistsEP0Status	This is function PLIB_USBHS_ExistsEP0Status.
	PLIB_USBHS_ExistsHighSpeedSupport	This is function PLIB_USBHS_ExistsHighSpeedSupport.
	PLIB_USBHS_ExistsInterrupts	This is function PLIB_USBHS_ExistsInterrupts.
	PLIB_USBHS_ExistsModuleControl	This is function PLIB_USBHS_ExistsModuleControl.
	PLIB_USBHS_ExistsRxEPStatus	This is function PLIB_USBHS_ExistsRxEPStatus.
	PLIB_USBHS_ExistsSoftReset	This is function PLIB_USBHS_ExistsSoftReset.
	PLIB_USBHS_ExistsTxEPStatus	This is function PLIB_USBHS_ExistsTxEPStatus.
	PLIB_USBHS_ExistsClockResetControl	This is function PLIB_USBHS_ExistsClockResetControl.

	PLIB_USBHS_ExistsUSBIDControl	This is function PLIB_USBHS_ExistsUSBIDControl.

i) Data Types and Constants

	Name	Description
	USBHS_CONFIGURATION	Provides the enumeration Configuration bits.
	USBHS_DATA01	Provides an enumeration data toggle for a packet.
	USBHS_DMA_ASSERT_TIMING	Provides enumeration DMA assertion timing (early versus late).
	USBHS_DMA_BURST_MODE	Provides enumeration of all DMA burst modes.
	USBHS_DMA_INTERRUPT	Provides enumeration of interrupts for DMA channels 0-7.
	USBHS_DMA_REQUEST_MODE	Used as an argument to set DMA request mode.
	USBHS_DMA_TRANSFER_MODE	Provides enumeration of all DMA transfer modes.
	USBHS_DYN_FIFO_PACKET_BUFFERING	Provides enumeration of dynamic FIFO double-packet versus single-packet buffering.
	USBHS_DYN_FIFO_SIZE	Provides enumeration of dynamic FIFO sizes.
	USBHS_ENDPOINT_DIRECTION	Used as an argument to identify an endpoint direction.
	USBHS_LPM_INTERRUPT	Provides an enumeration of LPM interrupt sources.
	USBHS_LPM_LINK_STATE	Provides enumeration requested device state after accepting an LPM transaction.
	USBHS_LPM_MODE	Provides enumeration of Link Power Management (LPM) modes.
	USBHS_OPMODES	Provides enumeration of operating modes supported by USB.
	USBHS_PKTS_PER_MICROFRAME	Provides an enumeration of the allowable isochronous packets per microframe when operating in High-Speed mode.
	USBHS_SPEED	Provides enumeration Host endpoint speeds.
	USBHS_TEST_SPEED	Used as an argument for in setting module speeds in Test mode.
	USBHS_TRANSACTION_TYPE	Provides an enumeration of transaction types.
	USBHS_TXRX_FIFO_STATE	Provides enumeration of receive and transmit FIFO states, as reported by status bits.
	USBHS_MAX_DMA_CHANNEL_NUMBER	Number of available DMA Channels.
	USBHS_MAX_EP_NUMBER	Maximum number of endpoints supported (not including EP0).

Description

This section describes the Application Programming Interface (API) functions of the USBHS Peripheral Library.

Refer to each section for a detailed description.

a) USBHS Setup Functions

PLIB_USBHS_HighSpeedDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_HighSpeedDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_HighSpeedDisable.

PLIB_USBHS_HighSpeedEnable Function

Sets the operation speed of the USB Module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_HighSpeedEnable(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function sets the operation speed of the USB module.

Remarks

None.

Preconditions

None.

Example

```
// Enable the USB module for high speed support.

PLIB_USBHS_HighSpeedEnable(USBHS_ID_0, USB_SPEED_HIGH);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
speed	The operation speed of the USB module

Function

void PLIB_USBHS_ModuleSpeedSet(USBHS_MODULE_ID index, uint32_t speed)

PLIB_USBHS_ResetDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_ResetDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ResetDisable.

PLIB_USBHS_ResetEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_ResetEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ResetEnable.

PLIB_USBHS_ResumeDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_ResumeDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ResumeDisable.

PLIB_USBHS_ResumeEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_ResumeEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ResumeEnable.

PLIB_USBHS_SessionDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_SessionDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_SessionDisable.

PLIB_USBHS_SessionEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_SessionEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_SessionEnable.

PLIB_USBHS_SoftResetDisable Function

Disables soft reset.

File

plib_usbhs.h

C

```
void PLIB_USBHS_SoftResetDisable(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function disables soft reset.

Remarks

Valid in both Peripheral and Host mode.

Preconditions

None.

Example

```
PLIB_USBHS_SoftResetDisable(USBHS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USBHS_SoftResetDisable(USBHS_MODULE_ID index);

PLIB_USBHS_SoftResetEnable Function

Enables soft reset.

File

plib_usbhs.h

C

```
void PLIB_USBHS_SoftResetEnable(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function enables soft reset.

Remarks

Valid in both Peripheral and Host mode.

Preconditions

None.

Example

```
PLIB_USBHS_SoftResetEnable(USBHS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USBHS_SoftResetEnable(USBHS_MODULE_ID index);

PLIB_USBHS_SuspendDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_SuspendDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_SuspendDisable.

PLIB_USBHS_SuspendEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_SuspendEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_SuspendEnable.

PLIB_USBHS_DMAOperationEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DMAOperationEnable(USBHS_MODULE_ID index, uint8_t endpoint, uint8_t dmaChannel, void *
address, uint32_t count, bool direction);
```

Description

This is function PLIB_USBHS_DMAOperationEnable.

PLIB_USBHS_TestModeEnter Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_TestModeEnter(USBHS_MODULE_ID index, uint8_t testMode);
```

Description

This is function PLIB_USBHS_TestModeEnter.

PLIB_USBHS_TestModeExit Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_TestModeExit(USBHS_MODULE_ID index, uint8_t testMode);
```

Description

This is function PLIB_USBHS_TestModeExit.

PLIB_USBHS_PhyIDMonitoringDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_PhyIDMonitoringDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_PhyIDMonitoringDisable.

PLIB_USBHS_PhyIDMonitoringEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_PhyIDMonitoringEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_PhyIDMonitoringEnable.

PLIB_USBHS_USBIDOverrideDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_USBIDOverrideDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_USBIDOverrideDisable.

PLIB_USBHS_USBIDOverrideEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_USBIDOverrideEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_USBIDOverrideEnable.

PLIB_USBHS_USBIDOverrideValueSet Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_USBIDOverrideValueSet(USBHS_MODULE_ID index, USBHS_USBID_OVERRIDE_VALUE id);
```

Description

This is function PLIB_USBHS_USBIDOverrideValueSet.

PLIB_USBHS_IsBDevice Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_IsBDevice(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_IsBDevice.

b) Endpoints Functions

PLIB_USBHS_Endpoint0FIFOFlush Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_Endpoint0FIFOFlush(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_Endpoint0FIFOFlush.

PLIB_USBHS_Endpoint0SetupPacketLoad Function

Loads the endpoint 0 FIFO with provided setup packet and then enables the endpoint transmit.

File

plib_usbhs.h

C

```
void PLIB_USBHS_Endpoint0SetupPacketLoad(USBHS_MODULE_ID index, void * setupPacket, uint8_t deviceAddress,
uint8_t hubAddress, uint8_t hubPortAddress, uint32_t speed);
```

Returns

None.

Description

This function loads the endpoint 0 FIFO with provided setup packet. This operation would typically be performed at the start of a endpoint 0 control transfer. Once the FIFO is loaded the function will load the target device address, hub and hub port address and then transmit the packet.

Remarks

Valid in Host mode only. Setup packet size should be 8 bytes. Packet will always be targeted to endpoint 0 of the target device.

Preconditions

None.

Example

```
// Send a setup packet to device 7 control endpoint
// connected directly to module.

PLIB_USBHS_Endpoint0SetupPacketLoad(USBHS_ID_0, setupPacket, 7, 0,
                                                    0, USB_SPEED_HIGH);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
setupPacket	setup packet to be sent to the device
deviceAddress	USB address of the device
hubAddress	USB address of the hub if the device is connected to a hub. Should be zero otherwise.
hubPortAddress	Port number of the hub port to which the device is connected if it is connected to a hub. Should be zero otherwise.
speed	Speed of the device to which the packet should be sent.

Function

void PLIB_USBHS_Endpoint0SetupPacketLoad(USBHS_MODULE_ID index,

void * setupPacket, uint8_t deviceAddress,

uint8_t hubAddress, uint8_t hubPortAddress,

uint32_t speed)

PLIB_USBHS_Endpoint0SetupPacketUnload Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_Endpoint0SetupPacketUnload(USBHS_MODULE_ID index, void * dest);
```

Description

This is function PLIB_USBHS_Endpoint0SetupPacketUnload.

PLIB_USBHS_EndpointFIFOLoad Function

Loads the endpoint FIFO with provided data and then enables the endpoint transmit.

File

plib_usbhs.h

C

```
void PLIB_USBHS_EndpointFIFOLoad(USBHS_MODULE_ID index, uint8_t endpoint, void * source, size_t nBytes);
```

Returns

None.

Description

This function loads the endpoint FIFO with provided data. This operation would typically be performed for a endpoint transmit operation. Once the FIFO is loaded the function will enable the enable the endpoint for transmit.

Remarks

Valid in both Peripheral and Host mode.

Preconditions

None.

Example

```
// Load endpoint 1 FIFO

PLIB_USBHS_EndpointFIFOLoad(USBHS_ID_0, 1, data, 10);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
endpoint	endpoint of which FIFO needs to be loaded
source	data that should be loaded
nBytes	number of bytes to load.

Function

void PLIB_USBHS_EndpointFIFOLoad(USBHS_MODULE_ID index, uint8_t endpoint,

void * source, size_t nBytes)

PLIB_USBHS_EndpointFIFOUnload Function

Unloads the endpoint FIFO.

File

plib_usbhs.h

C

```
int PLIB_USBHS_EndpointFIFOUnload(USBHS_MODULE_ID index, uint8_t endpoint, void * dest);
```

Returns

Returns the number of functions that were read.

Description

This function unloads the endpoint FIFO. This operation would typically be performed for a endpoint receive operation. The unloaded data is stored in provided buffer.

Remarks

Valid in both Peripheral and Host mode.

Preconditions

The USB module must be configured for Device mode operation.

Example

```
// Load endpoint 1 FIFO
int count;

count = PLIB_USBHS_EndpointFIFOUnload(USBHS_ID_0, 1, dest, 10);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
endpoint	endpoint of which FIFO to be unloaded.
dest	target address where unloaded data should be stored.

Function

int PLIB_USBHS_EndpointFIFOUnload(USBHS_MODULE_ID index, uint8_t endpoint,
void * destination)

PLIB_USBHS_EndpointRxFIFOFlush Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EndpointRxFIFOFlush(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_EndpointRxFIFOFlush.

PLIB_USBHS_EndpointRxRequestClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EndpointRxRequestClear(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_EndpointRxRequestClear.

PLIB_USBHS_EndpointRxRequestEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EndpointRxRequestEnable(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_EndpointRxRequestEnable.

PLIB_USBHS_EndpointTxFIFOFlush Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EndpointTxFIFOFlush(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_EndpointTxFIFOFlush.

PLIB_USBHS_EP0DataEndSet Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0DataEndSet(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0DataEndSet.

PLIB_USBHS_EP0INHandshakeClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0INHandshakeClear(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0INHandshakeClear.

PLIB_USBHS_EP0INHandshakeSend Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0INHandshakeSend(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0INHandshakeSend.

PLIB_USBHS_EP0INTokenSend Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0INTokenSend(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0INTokenSend.

PLIB_USBHS_EP0OUTHandshakeSend Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0OUTHandshakeSend(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0OUTHandshakeSend.

PLIB_USBHS_EP0RxPktRdyServiced Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0RxPktRdyServiced(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0RxPktRdyServiced.

PLIB_USBHS_EP0RxPktRdyServicedDataEnd Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0RxPktRdyServicedDataEnd(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0RxPktRdyServicedDataEnd.

PLIB_USBHS_EP0SentStallClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0SentStallClear(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0SentStallClear.

PLIB_USBHS_EP0SetupEndServiced Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0SetupEndServiced(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0SetupEndServiced.

PLIB_USBHS_EP0StallDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0StallDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0StallDisable.

PLIB_USBHS_EP0StallEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0StallEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0StallEnable.

PLIB_USBHS_EP0StatusClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0StatusClear(USBHS_MODULE_ID index, USBHS_EP0_ERROR error);
```

Description

This is function PLIB_USBHS_EP0StatusClear.

PLIB_USBHS_EP0StatusGet Function

File

plib_usbhs.h

C

```
uint8_t PLIB_USBHS_EP0StatusGet(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0StatusGet.

PLIB_USBHS_EP0TxPktRdy Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0TxPktRdy(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0TxPktRdy.

PLIB_USBHS_EP0TxPktRdyDataEnd Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_EP0TxPktRdyDataEnd(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_EP0TxPktRdyDataEnd.

PLIB_USBHS_HostRxEndpointConfigure Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_HostRxEndpointConfigure(USBHS_MODULE_ID index, uint8_t hostEndpoint, uint32_t speed,
uint32_t pipeType, uint16_t endpointSize, uint16_t receiveFIFOAddress, uint16_t fifoSize, uint8_t
targetEndpoint, uint8_t targetDevice, uint8_t targetHub, uint8_t targetHubPort, uint8_t nakInterval);
```

Description

This is function PLIB_USBHS_HostRxEndpointConfigure.

PLIB_USBHS_HostTxEndpointConfigure Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_HostTxEndpointConfigure(USBHS_MODULE_ID index, uint8_t hostEndpoint, uint32_t speed,
uint32_t pipeType, uint16_t endpointSize, uint16_t receiveFIFOAddress, uint16_t fifoSize, uint8_t
targetEndpoint, uint8_t targetDevice, uint8_t targetHub, uint8_t targetHubPort, uint8_t nakInterval);
```

Description

This is function PLIB_USBHS_HostTxEndpointConfigure.

PLIB_USBHS_RxEPINTokenSend Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_RxEPINTokenSend(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_RxEPINTokenSend.

PLIB_USBHS_RxEPStatusClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_RxEPStatusClear(USBHS_MODULE_ID index, uint8_t endpoint, USBHS_RXEP_ERROR error);
```

Description

This is function PLIB_USBHS_RxEPStatusClear.

PLIB_USBHS_RxEPStatusGet Function

File

plib_usbhs.h

C

```
uint8_t PLIB_USBHS_RxEPStatusGet(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_RxEPStatusGet.

PLIB_USBHS_TxEPStatusClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_TxEPStatusClear(USBHS_MODULE_ID index, uint8_t endpoint, USBHS_TXEP_ERROR error);
```

Description

This is function PLIB_USBHS_TxEPStatusClear.

PLIB_USBHS_TxEPStatusGet Function

File

plib_usbhs.h

C

```
uint8_t PLIB_USBHS_TxEPStatusGet(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_TxEPStatusGet.

PLIB_USBHS_GetEP0CSRAddress Function

File

plib_usbhs.h

C

```
uint8_t * PLIB_USBHS_GetEP0CSRAddress(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_GetEP0CSRAddress.

PLIB_USBHS_GetEP0FIFOAddress Function

File

plib_usbhs.h

C

```
uint8_t * PLIB_USBHS_GetEP0FIFOAddress(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_GetEP0FIFOAddress.

PLIB_USBHS_LoadEPInIndex Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_LoadEPInIndex(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_LoadEPInIndex.

c) Interrupts Functions

PLIB_USBHS_TxInterruptDisable Function

Disables a TX endpoint interrupt source for the USB module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_TxInterruptDisable(USBHS_MODULE_ID index, USBHS_EPTXRX_INTERRUPT interruptFlag);
```

Returns

None.

Description

This function disables a TX endpoint interrupt source for the USB module.

Remarks

See also PLIB_USBHS_TxInterruptEnable.

Preconditions

None.

Example

```
PLIB_USBHS_TxInterruptDisable( MY_USB_INSTANCE, USBHS_TXRXINT_EP1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USBHS_TxInterruptDisable(USBHS_MODULE_ID index,
USBHS_EPTXRX_INTERRUPT interruptFlag)

PLIB_USBHS_TxInterruptEnable Function

Enables a TX endpoint Interrupt for the USB module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_TxInterruptEnable(USBHS_MODULE_ID index, USBHS_EPTXRX_INTERRUPT interruptFlag);
```

Returns

None.

Description

This function enables the TX endpoint interrupt sources of the USB module that are used to trigger a USB interrupt.

Remarks

See also PLIB_USBHS_TxInterruptDisable.

Preconditions

None.

Example

```
PLIB_USBHS_TxInterruptEnable( MY_USB_INSTANCE, USBHS_TXRXINT_EP1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USBHS_TxInterruptEnable(USBHS_MODULE_ID index,
USBHS_EPTXRX_INTERRUPT interruptFlag)

PLIB_USBHS_GenInterruptDisable Function

Disables a general interrupt for the USB module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_GenInterruptDisable(USBHS_MODULE_ID index, USBHS_GEN_INTERRUPT interruptFlag);
```

Returns

None.

Description

This function disables a general interrupt source for the USB module.

Remarks

See also PLIB_USBHS_GenInterruptEnable.

Preconditions

None.

Example

```
PLIB_USBHS_GenInterruptDisable( MY_USB_INSTANCE, USBHS_GENINT_VBUSERR );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USBHS_GenInterruptDisable(USBHS_MODULE_ID index,
USBHS_GEN_INTERRUPT interruptFlag)

PLIB_USBHS_GenInterruptEnable Function

Enables a general interrupt for the USB module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_GenInterruptEnable(USBHS_MODULE_ID index, USBHS_GEN_INTERRUPT interruptFlag);
```

Returns

None.

Description

This function enables the general interrupt sources of the USB module that are used to trigger a USB interrupt.

Remarks

See also PLIB_USBHS_GenInterruptDisable.

Preconditions

None.

Example

```
PLIB_USBHS_GenInterruptEnable( MY_USB_INSTANCE, USBHS_GENINT_VBUSERR );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USBHS_GenInterruptEnable(USBHS_MODULE_ID index,
USBHS_GEN_INTERRUPT interruptFlag)

PLIB_USBHS_GenInterruptFlagsGet Function

Gets general interrupt flags.

File

plib_usbhs.h

C

```
USBHS_GEN_INTERRUPT PLIB_USBHS_GenInterruptFlagsGet(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function gets the general interrupt flags.

Remarks

None.

Preconditions

None.

Example

```
USBHS_GEN_INTERRUPT interruptFlags;
interruptFlags = PLIB_USBHS_GenInterruptFlagsGet( MY_USB_INSTANCE );
// All interrupt flags cleared on read.

if ( interruptFlags > 0 )
    if ( interruptFlags & USBHS_GENINT_SUSPEND )
    {
       // Device has detected suspend signaling on the bus.
    }
    if ( interruptFlags & USBHS_GENINT_RESUME )
    {
        //
    }
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USBHS_GEN_INTERRUPT PLIB_USBHS_GenInterruptFlagsGet(USBHS_MODULE_ID index)

PLIB_USBHS_InterruptEnableSet Function

Enables USB module event interrupts.

File

plib_usbhs.h

C

```
void PLIB_USBHS_InterruptEnableSet(USBHS_MODULE_ID index, USBHS_GEN_INTERRUPT generalInterrupts,
USBHS_EPTXRX_INTERRUPT transmitInterrupts, USBHS_EPTXRX_INTERRUPT receiveInterrupts);
```

Returns

None.

Description

This function enables USB module event interrupts. Combines the functionality of the PLIB_USBHS_RxInterruptEnable, PLIB_USBHS_TxInterruptEnable, and PLIB_USBHS_GenInterruptEnable functions.

Remarks

None.

Preconditions

None.

Example

```
// Enable the reset interrupt, endpoint 1 transmit interrupt
// and endpoint 2 receive interrupts

PLIB_USBHS_InterruptEnableSet (USBHS_ID_0, USBHS_GENINT_RESET,
    USBHS_TXRXINT_EP1, USBHS_TXRXINT_EP2 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
generalInterrupts	General Module interrupts.
transmitInterrupts	Transmit Endpoint Interrupts to be enabled.
receiveInterrupts	Receive Endpoint Interrupts to be enabled.

Function

void PLIB_USBHS_InterruptEnableSet(USBHS_MODULE_ID index,

USBHS_GEN_INTERRUPT generalInterrupts,

USBHS_EPTXRX_INTERRUPT transmitInterrupts,

USBHS_EPTXRX_INTERRUPT receiveInterrupts);

PLIB_USBHS_DMAInterruptDisable Function

Disables DMA channel interrupts.

File

plib_usbhs.h

C

```
void PLIB_USBHS_DMAInterruptDisable(USBHS_MODULE_ID index, USBHS_DMA_INTERRUPT nChannelNumber);
```

Returns

None.

Description

This function disables DMA Channel interrupts.

Remarks

None.

Preconditions

None.

Example

```
PLIB_USBHS_DMAInterruptDisable (USBHS_ID_0, USBHS_DMAINT_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
dmaChannelInterrupt	DMA channel Interrupt to disable

Function

void PLIB_USBHS_DMAInterruptDisable(USBHS_MODULE_ID index,
USBHS_DMA_INTERRUPT dmaChannelInterrupt)

PLIB_USBHS_DMAInterruptEnable Function

Enables DMA channel interrupts.

File

plib_usbhs.h

C

```
void PLIB_USBHS_DMAInterruptEnable(USBHS_MODULE_ID index, USBHS_DMA_INTERRUPT nChannelNumber);
```

Returns

None.

Description

This function enables DMA Channel interrupts.

Remarks

None.

Preconditions

None.

Example

```
PLIB_USBHS_DMAInterruptEnable (USBHS_ID_0, USBHS_DMAINT_5);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
dmaChannelInterrupt	DMA channel Interrupt to enable

Function

void PLIB_USBHS_DMAInterruptEnable(USBHS_MODULE_ID index,
USBHS_DMA_INTERRUPT dmaChannelInterrupt)

PLIB_USBHS_DMAInterruptFlagsGet Function

Gets the DMA channel interrupt flags.

File

plib_usbhs.h

C

```
USBHS_DMA_INTERRUPT PLIB_USBHS_DMAInterruptFlagsGet(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function gets the DMA channel interrupt flags.

Remarks

None.

Preconditions

None.

Example

```
USBHS_DMA_INTERRUPT interruptFlags;
interruptFlags = PLIB_USBHS_DMAInterruptFlagsGet( MY_USB_INSTANCE );
// All interrupt flags cleared on read.

if ( interruptFlags > 0 )
    if ( interruptFlags & USBHS_DMAINT_1 )
    {
        // DMA Channel 1
    }
    if (interruptFlags & USBHS_DMAINT_2)
    {
        // DMA Channel 2
    }
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USBHS_DMA_INTERRUPT PLIB_USBHS_DMAInterruptFlagsGet(USBHS_MODULE_ID index)

PLIB_USBHS_TxInterruptFlagsGet Function

Gets the TX endpoint interrupt flags.

File

plib_usbhs.h

C

```
USBHS_EPTXRX_INTERRUPT PLIB_USBHS_TxInterruptFlagsGet(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function gets the TX endpoint interrupt flags.

Remarks

TX interrupts must first be enabled by calling PLIB_USBHS_TxInterruptEnable.

Preconditions

None.

Example

```
USBHS_EPTXRX_INTERRUPT interruptFlags;
interruptFlags = PLIB_USBHS_TxInterruptFlagsGet( MY_USB_INSTANCE );
// All interrupt flags cleared on read.

if ( interruptFlags > 0 )
    if ( interruptFlags & USBHS_TXRXINT_EP0 )
    {
       // Endpoint Zero
    }
    if ( interruptFlags & USBHS_TXRXINT_EP1 )
    {
        // Endpoint One
    }
    .
    .
    .
}
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USBHS_EPTXRX_INTERRUPT PLIB_USBHS_TxInterruptFlagsGet(USBHS_MODULE_ID index)

PLIB_USBHS_RxInterruptDisable Function

Disables a RX endpoint interrupt for the USB module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_RxInterruptDisable(USBHS_MODULE_ID index, USBHS_EPTXRX_INTERRUPT interruptFlag);
```

Returns

None.

Description

This function disables a RX endpoint interrupt source for the USB module.

Remarks

See also PLIB_USBHS_RxInterruptEnable. USBHS_TXRXINT_EP0 is not a valid argument. For endpoint zero, use PLIB_USBHS_TxInterruptDisable.

Preconditions

None.

Example

```
PLIB_USBHS_RxInterruptDisable( MY_USB_INSTANCE, USBHS_TXRXINT_EP1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USBHS_RxInterruptDisable(USBHS_MODULE_ID index,

USBHS_EPTXRX_INTERRUPT interruptFlag)

PLIB_USBHS_RxInterruptEnable Function

Enables a RX endpoint interrupt for the USB module.

File

plib_usbhs.h

C

```
void PLIB_USBHS_RxInterruptEnable(USBHS_MODULE_ID index, USBHS_EPTXRX_INTERRUPT interruptFlag);
```

Returns

None.

Description

This function enables RX endpoint interrupt sources of the USB module to trigger a USB interrupt.

Remarks

See also PLIB_USBHS_RxInterruptDisable. USBHS_TXRXINT_EP0 is not a valid argument. For endpoint zero use PLIB_USBHS_TxInterruptEnable.

Preconditions

None.

Example

```
PLIB_USBHS_RxInterruptEnable( MY_USB_INSTANCE, USBHS_TXRXINT_EP1 );
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest
interruptFlag	Interrupt

Function

void PLIB_USBHS_RxInterruptEnable(USBHS_MODULE_ID index,

USBHS_EPTXRX_INTERRUPT interruptFlag)

PLIB_USBHS_RxInterruptFlagsGet Function

Gets RX endpoint interrupt flags.

File

plib_usbhs.h

C

```
USBHS_EPTXRX_INTERRUPT PLIB_USBHS_RxInterruptFlagsGet(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function gets the RX endpoint interrupt flags.

Remarks

RX interrupts must first be enabled by calling PLIB_USBHS_RxInterruptEnable. USBHS_TXRXINT_EP0 is not a valid argument. For endpoint zero use PLIB_USBHS_TxInterruptFlagsGet.

Preconditions

None.

Example

```
USBHS_EPTXRX_INTERRUPT interruptFlags;
```

```
    interruptFlags = PLIB_USBHS_RxInterruptFlagsGet( MY_USB_INSTANCE );
    // All interrupt flags cleared on read.

    if ( interruptFlags > 0 )
        if ( interruptFlags & USBHS_TXRXINT_EP1 )
        {
           // Endpoint Zero
        }
        if ( interruptFlags & USBHS_TXRXINT_EP2 )
        {
            // Endpoint One
        }
        .
        .
        .
    }
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USBHS_EPTXRX_INTERRUPT PLIB_USBHS_RxInterruptFlagsGet(USBHS_MODULE_ID index)

PLIB_USBHS_DMAInterruptGet Function

File

plib_usbhs.h

C

```
uint8_t PLIB_USBHS_DMAInterruptGet(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_DMAInterruptGet.

PLIB_USBHS_GlobalInterruptDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_GlobalInterruptDisable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_GlobalInterruptDisable.

PLIB_USBHS_GlobalInterruptEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_GlobalInterruptEnable(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_GlobalInterruptEnable.

d) Device Functions

PLIB_USBHS_DeviceAddressGet Function

Returns the current USB device address.

File

plib_usbhs.h

C

```
uint8_t PLIB_USBHS_DeviceAddressGet(USBHS_MODULE_ID index);
```

Returns

7-bit unsigned integer value indicating the current USB device address.

Description

This function returns the current USB device address.

Remarks

None.

Preconditions

The USB module should have been configured for Device mode operation.

Example

```
// This code example reads the current assigned USB device address.

uint8_t address;
address = PLIB_USBHS_DeviceAddressGet(USBHS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

uint8_t PLIB_USBHS_DeviceAddressGet (USBHS_MODULE_ID index)

PLIB_USBHS_DeviceAddressSet Function

Sets the device address.

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceAddressSet(USBHS_MODULE_ID index, uint8_t address);
```

Returns

None.

Description

This function sets the device address. This function should be called with the address received from the host in the SET_ADDRESS request.

Remarks

None.

Preconditions

The USB module must be configured for Device mode operation.

Example

```
// This code example assigns a USB device address.

uint8_t address;
```

```
PLIB_USBHS_DeviceAddressSet(USBHS_ID_0, address);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
address	7-bit USB Device address that should be assigned to this device

Function

void PLIB_USBHS_DeviceAddressSet (USBHS_MODULE_ID index, uint8_t address)

PLIB_USBHS_DeviceAttach Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceAttach(USBHS_MODULE_ID index, uint32_t speed);
```

Description

This is function PLIB_USBHS_DeviceAttach.

PLIB_USBHS_DeviceConnect Function

Tri-states the USB D+ and D- lines.

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceConnect(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function enables the USB D+ and D- lines, which connects the device from the host.

Remarks

Only valid for Device mode.

Preconditions

The USB module must be configured for Device mode operation.

Example

```
// Disconnect the device from the host

PLIB_USBHS_DeviceConnect(USBHS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USBHS_DeviceConnect(USBHS_MODULE_ID index)

PLIB_USBHS_DeviceDetach Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceDetach(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_DeviceDetach.

PLIB_USBHS_DeviceDisconnect Function

Tri-states the USB D+ and D- lines.

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceDisconnect(USBHS_MODULE_ID index);
```

Returns

None.

Description

This function tri-states the USB D+ and D- lines, which disconnects the device from the host.

Remarks

Only valid for Device mode.

Preconditions

None.

Example

```
// Disconnect the device from the host

PLIB_USBHS_DeviceDisconnect(USBHS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void PLIB_USBHS_DeviceDisconnect(USBHS_MODULE_ID index)

PLIB_USBHS_DeviceEPFIFOLoad Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceEPFIFOLoad(USBHS_MODULE_ID index, uint8_t endpoint, void * source, size_t nBytes);
```

Description

This is function PLIB_USBHS_DeviceEPFIFOLoad.

PLIB_USBHS_DeviceEPFIFOUnload Function

File

plib_usbhs.h

C

```
int PLIB_USBHS_DeviceEPFIFOUnload(USBHS_MODULE_ID index, uint8_t endpoint, void * dest);
```

Description

This is function PLIB_USBHS_DeviceEPFIFOUnload.

PLIB_USBHS_DeviceRxEndpointConfigure Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceRxEndpointConfigure(USBHS_MODULE_ID index, uint8_t endpoint, uint16_t endpointSize,
uint16_t fifoAddress, uint8_t fifoSize, uint32_t transferType);
```

Description

This is function PLIB_USBHS_DeviceRxEndpointConfigure.

PLIB_USBHS_DeviceRxEndpointStallDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceRxEndpointStallDisable(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_DeviceRxEndpointStallDisable.

PLIB_USBHS_DeviceRxEndpointStallEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceRxEndpointStallEnable(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_DeviceRxEndpointStallEnable.

PLIB_USBHS_DeviceTxEndpointConfigure Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceTxEndpointConfigure(USBHS_MODULE_ID index, uint8_t endpoint, uint16_t endpointSize,
uint16_t fifoAddress, uint8_t fifoSize, uint32_t transferType);
```

Description

This is function PLIB_USBHS_DeviceTxEndpointConfigure.

PLIB_USBHS_DeviceTxEndpointPacketReady Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceTxEndpointPacketReady(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_DeviceTxEndpointPacketReady.

PLIB_USBHS_DeviceTxEndpointStallDisable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceTxEndpointStallDisable(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_DeviceTxEndpointStallDisable.

PLIB_USBHS_DeviceTxEndpointStallEnable Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_DeviceTxEndpointStallEnable(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_DeviceTxEndpointStallEnable.

e) Host Functions

PLIB_USBHS_HostRxEndpointDataToggleClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_HostRxEndpointDataToggleClear(USBHS_MODULE_ID index, uint8_t hostEndpoint);
```

Description

This is function PLIB_USBHS_HostRxEndpointDataToggleClear.

PLIB_USBHS_HostTxEndpointDataToggleClear Function

File

plib_usbhs.h

C

```
void PLIB_USBHS_HostTxEndpointDataToggleClear(USBHS_MODULE_ID index, uint8_t hostEndpoint);
```

Description

This is function PLIB_USBHS_HostTxEndpointDataToggleClear.

f) Status Functions

PLIB_USBHS_FullOrHighSpeedIsConnected Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_FullOrHighSpeedIsConnected(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_FullOrHighSpeedIsConnected.

PLIB_USBHS_HighSpeedIsConnected Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_HighSpeedIsConnected(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_HighSpeedIsConnected.

PLIB_USBHS_HostModeIsEnabled Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_HostModeIsEnabled(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_HostModeIsEnabled.

PLIB_USBHS_ModuleSpeedGet Function

Returns the speed at which the module is operating.

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ModuleSpeedGet(USBHS_MODULE_ID index);
```

Returns

Current module operation speed on the USB.

Description

This function returns the speed at which the module is operating. In case of device mode operation, this function returns the speed at which the device attached to the host.

Remarks

None.

Preconditions

None.

Example

```
USB_SPEED speed;
speed = PLIB_USBHS_ModuleSpeedGet(USBHS_ID_0);
```

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USB_SPEED PLIB_USBHS_ModuleSpeedGet(USBHS_MODULE_ID index)

PLIB_USBHS_DMAErrorGet Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_DMAErrorGet(USBHS_MODULE_ID index, uint8_t dmaChannel);
```

Description

This is function PLIB_USBHS_DMAErrorGet.

PLIB_USBHS_GetReceiveDataCount Function

File

plib_usbhs.h

C

```
uint32_t PLIB_USBHS_GetReceiveDataCount(USBHS_MODULE_ID index, uint8_t endpoint);
```

Description

This is function PLIB_USBHS_GetReceiveDataCount.

g) VBUS Support Functions

PLIB_USBHS_VbusLevelGet Function

Returns the current VBUS level encode using the USBHS_VBUS_DETECT_LEVEL enumeration.

File

plib_usbhs.h

C

```
USBHS_VBUS_LEVEL PLIB_USBHS_VbusLevelGet(USBHS_MODULE_ID index);
```

Returns

Detected VBUS level, see USBHS_VBUS_DETECT_LEVEL enumeration.

Description

Returns the current VBUS level encode using the USBHS_VBUS_DETECT_LEVEL enumeration.

Remarks

None.

Preconditions

None.

Example

Parameters

Parameters	Description
index	Identifier for the device instance of interest

Function

USBHS_VBUS_DETECT_LEVEL PLIB_USBHS_VbusLevelGet(USBHS_MODULE_ID index)

PLIB_USBHS_VBUSLevelGet Function

File

plib_usbhs.h

C

```
USBHS_VBUS_LEVEL PLIB_USBHS_VBUSLevelGet(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_VBUSLevelGet.

h) Feature Existence Functions

PLIB_USBHS_ExistsEndpointFIFO Function

Identifies that the Endpoint FIFO feature exists on the Hi-Speed USB module.

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsEndpointFIFO(USBHS_MODULE_ID index);
```

Returns

- true - The feature is supported
- false - The feature is not supported

Description

This interface identifies that the Endpoint FIFO feature is available on the Hi-Speed USB module. When this interface returns true, these functions are supported on the device:

- PLIB_USBHS_EndpointFIFOLoad
- PLIB_USBHS_EndpointFIFOUnload

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_USBHS_ExistsEndpointFIFO(USBHS_MODULE_ID index)

PLIB_USBHS_ExistsEndpointOperations Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsEndpointOperations(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsEndpointOperations.

PLIB_USBHS_ExistsEP0Status Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsEP0Status(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsEP0Status.

PLIB_USBHS_ExistsHighSpeedSupport Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsHighSpeedSupport(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsHighSpeedSupport.

PLIB_USBHS_ExistsInterrupts Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsInterrupts(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsInterrupts.

PLIB_USBHS_ExistsModuleControl Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsModuleControl(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsModuleControl.

PLIB_USBHS_ExistsRxEPStatus Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsRxEPStatus(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsRxEPStatus.

PLIB_USBHS_ExistsSoftReset Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsSoftReset(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsSoftReset.

PLIB_USBHS_ExistsTxEPStatus Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsTxEPStatus(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsTxEPStatus.

PLIB_USBHS_ExistsClockResetControl Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsClockResetControl(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsClockResetControl.

PLIB_USBHS_ExistsUSBIDControl Function

File

plib_usbhs.h

C

```
bool PLIB_USBHS_ExistsUSBIDControl(USBHS_MODULE_ID index);
```

Description

This is function PLIB_USBHS_ExistsUSBIDControl.

i) Data Types and Constants

USBHS_CONFIGURATION Enumeration

Provides the enumeration Configuration bits.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_CONFIG_SOFTCONN,
  USBHS_CONFIG_DYNFIFOSIZE,
```

```
  USBHS_CONFIG_HBWTXISO,
  USBHS_CONFIG_HBWRXISO,
  USBHS_CONFIG_BIGENDIAN,
  USBHS_CONFIG_AUTOSPLIT,
  USBHS_CONFIG_AUTOJOIN
} USBHS_CONFIGURATION;
```

Members

Members	Description
USBHS_CONFIG_SOFTCONN	Soft connect/disconnect supported
USBHS_CONFIG_DYNFIFOSIZE	Dynamic sizing of FIFO buffers supported
USBHS_CONFIG_HBWTXISO	High bandwidth TX isochronous transfers supported
USBHS_CONFIG_HBWRXISO	High bandwidth RX isochronous transfers supported
USBHS_CONFIG_BIGENDIAN	Big Endian byte ordering supported instead of Little Endian
USBHS_CONFIG_AUTOSPLIT	Automatic splitting of bulk packets supported
USBHS_CONFIG_AUTOJOIN	Automatic combining of bulk packets supported

Description

USBHS Hardware Configuration Bits Enumeration

This data type provides the enumeration configuration bits returned by PLIB_USBHS_ConfigurationBitsGet.

Remarks

See also PLIB_USBHS_ConfigurationBitsGet.

USBHS_DATA01 Enumeration

Provides an enumeration data toggle for a packet.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_DATA0,
  USBHS_DATA1
} USBHS_DATA01;
```

Members

Members	Description
USBHS_DATA0	DATA0/1 = 0
USBHS_DATA1	DATA0/1 = 1

Description

USBHS Endpoint Data Toggle Enumeration

This data type provides an enumeration data toggle for a packet.

Remarks

None.

USBHS_DMA_ASSERT_TIMING Enumeration

Provides enumeration DMA assertion timing (early versus late).

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_DMA_ASSERT_EARLY,
  USBHS_DMA_ASSERT_LATE
} USBHS_DMA_ASSERT_TIMING;
```

Members

Members	Description
USBHS_DMA_ASSERT_EARLY	Assert DMA 8 bytes before filling FIFO
USBHS_DMA_ASSERT_LATE	Assert DMA when FIFO is full

Description

USBHS DMA Assertion Timing Enumeration

This data type provides enumeration DMA assertion timing (early versus late).

Remarks

None.

USBHS_DMA_BURST_MODE Enumeration

Provides enumeration of all DMA burst modes.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_DMA_BURST_MODE0,
  USBHS_DMA_BURST_MODE1,
  USBHS_DMA_BURST_MODE2,
  USBHS_DMA_BURST_MODE3
} USBHS_DMA_BURST_MODE;
```

Members

Members	Description
USBHS_DMA_BURST_MODE0	Burst Mode 0: Bursts of unspecified length
USBHS_DMA_BURST_MODE1	Burst Mode 1: INCR4 or unspecified length
USBHS_DMA_BURST_MODE2	Burst Mode 2: INCR8, INCR4 or unspecified length
USBHS_DMA_BURST_MODE3	Burst Mode 3: INCR16, INCR8, INCR4 or unspecified length

Description

USBHS DMA Burst Modes Enumeration

This data type provides enumeration of all DMA burst modes.

Remarks

None.

USBHS_DMA_INTERRUPT Enumeration

Provides enumeration of interrupts for DMA channels 0-7.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_DMAINT_1,
  USBHS_DMAINT_2,
  USBHS_DMAINT_3,
  USBHS_DMAINT_4,
  USBHS_DMAINT_5,
  USBHS_DMAINT_6,
  USBHS_DMAINT_7,
  USBHS_DMAINT_8,
  USBHS_DMAINT_ANY,
  USBHS_DMAINT_ALL
} USBHS_DMA_INTERRUPT;
```

Description

USBHS DMA Interrupts Enumeration

This data type provides the enumeration of interrupts for DMA channels 0-7.

Remarks

None.

USBHS_DMA_REQUEST_MODE Enumeration

Used as an argument to set DMA request mode.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_DMA_MODE0,
  USBHS_DMA_MODE1
} USBHS_DMA_REQUEST_MODE;
```

Description

USBHS DMA Request Mode Enumeration

THis data type is used as an argument to set DMA request mode. Valid only for Endpoints 1-7.

Remarks

Used by PLIB_USBHS_EPnDMARequestModeSet.

USBHS_DMA_TRANSFER_MODE Enumeration

Provides enumeration of all DMA transfer modes.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_DMA_TRANSFER_MODE0,
  USBHS_DMA_TRANSFER_MODE1
} USBHS_DMA_TRANSFER_MODE;
```

Description

USBHS DMA Transfer Modes Enumeration

This data type provides enumeration of all DMA Transfer modes.

Remarks

None.

USBHS_DYN_FIFO_PACKET_BUFFERING Enumeration

Provides enumeration of dynamic FIFO double-packet versus single-packet buffering.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_FIFO_SINGLEPKT,
  USBHS_FIFO_DOUBLEPKT
} USBHS_DYN_FIFO_PACKET_BUFFERING;
```

Description

USBHS Dynamic FIFO Double Packet versus Single Packet Buffering

This data type provides enumeration of dynamic FIFO double-packet versus single-packet buffering.

Remarks

None.

USBHS_DYN_FIFO_SIZE Enumeration

Provides enumeration of dynamic FIFO sizes.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_FIFO_SIZE_8BYTES,
  USBHS_FIFO_SIZE_16BYTES,
  USBHS_FIFO_SIZE_32BYTES,
  USBHS_FIFO_SIZE_64BYTES,
  USBHS_FIFO_SIZE_128BYTES,
  USBHS_FIFO_SIZE_256BYTES,
  USBHS_FIFO_SIZE_512BYTES,
  USBHS_FIFO_SIZE_1024BYTES,
  USBHS_FIFO_SIZE_2048BYTES,
  USBHS_FIFO_SIZE_4096BYTES
} USBHS_DYN_FIFO_SIZE;
```

Description

USBHS Dynamic FIFO Size Enumeration

This data type provides enumeration of dynamic FIFO sizes.

Remarks

FIFO size = 2^(FIFO_Size<3:0> + 3)

USBHS_ENDPOINT_DIRECTION Enumeration

Used as an argument to identify an endpoint direction.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_ENDPOINT_RX,
  USBHS_ENDPOINT_TX
} USBHS_ENDPOINT_DIRECTION;
```

Members

Members	Description
USBHS_ENDPOINT_RX	RX endpoint
USBHS_ENDPOINT_TX	TX endpoint

Description

USBHS Endpoint Direction Enumeration

This data type is used as an argument to identify an endpoint direction. Valid only for Endpoints 1-7.

Remarks

None.

USBHS_LPM_INTERRUPT Enumeration

Provides an enumeration of LPM interrupt sources.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_LPMINT_STALL,
  USBHS_LPMINT_NYET,
  USBHS_LPMINT_ACK,
  USBHS_LPMINT_NOTCOMPLETE,
  USBHS_LPMINT_RESUMED,
  USBHS_LPMINT_ERROR
} USBHS_LPM_INTERRUPT;
```

Members

Members	Description
USBHS_LPMINT_STALL	Host: Device responded with STALL. Peripheral/Device: Sent a STALL.
USBHS_LPMINT_NYET	Host: Device responded with NYET. Peripheral/Device: Sent a NYET.
USBHS_LPMINT_ACK	Host: Device responded with ACK. Peripheral/Device: Sent an ACK.
USBHS_LPMINT_NOTCOMPLETE	Host: LPM transaction failed to complete. Peripheral/Device: NYET sent because data is pending in RX FIFOs.
USBHS_LPMINT_RESUMED	USB Module has resumed operation.
USBHS_LPMINT_ERROR	Host: Received Bit Stuff or PID error. Peripheral/Device: Unsupported LinkState field received.

Description

USBHS Link Power Management (LPM) Interrupts Enumeration

This data type provides an enumeration of LPM interrupt sources.

Remarks

None.

USBHS_LPM_LINK_STATE Enumeration

Provides enumeration requested device state after accepting an LPM transaction.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_LPM_RESERVED_0,
  USBHS_LPM_L1_STATE,
  USBHS_LPM_RESERVED_2,
  USBHS_LPM_RESERVED_3,
  USBHS_LPM_RESERVED_4,
  USBHS_LPM_RESERVED_5,
  USBHS_LPM_RESERVED_6,
  USBHS_LPM_RESERVED_7,
  USBHS_LPM_RESERVED_8,
  USBHS_LPM_RESERVED_9,
  USBHS_LPM_RESERVED_A,
  USBHS_LPM_RESERVED_B,
  USBHS_LPM_RESERVED_C,
  USBHS_LPM_RESERVED_D,
  USBHS_LPM_RESERVED_E,
  USBHS_LPM_RESERVED_F
} USBHS_LPM_LINK_STATE;
```

Description

USBHS Link Power Management Requested State Enumeration

This data type provides enumeration requested device state after accepting an LPM transaction.

Remarks

Used by PLIB_USBHS_LPMRequestedLinkStateGet and PLIB_USBHS_LPMRequestedLinkStateSet.

USBHS_LPM_MODE Enumeration

Provides enumeration of Link Power Management (LPM) modes.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_LPM_DISABLED,
  USBHS_LPM_EXTENDEDNOLPM,
  USBHS_LPM_DISABLED2,
  USBHS_LPM_LPMEXTENDED
} USBHS_LPM_MODE;
```

Members

Members	Description
USBHS_LPM_DISABLED	LPM and Extended transactions not supported and will time-out
USBHS_LPM_EXTENDEDNOLPM	Extended transactions supported but LPM transactions not supported and produce STALLs
USBHS_LPM_DISABLED2	LPM and Extended transactions not supported and will timeout
USBHS_LPM_LPMEXTENDED	LPM and Extended transactions are supported

Description

USBHS Link Power Management (LPM) Mode Enumeration

This data type provides enumeration of Link Power Management (LPM) modes.

Remarks

Used by PLIB_USBHS_LPMModeSet.

USBHS_OPMODES Enumeration

Provides enumeration of operating modes supported by USB.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_OPMODE_NONE,
  USBHS_OPMODE_DEVICE,
  USBHS_OPMODE_HOST,
  USBHS_OPMODE_OTG
} USBHS_OPMODES;
```

Members

Members	Description
USBHS_OPMODE_NONE	None
USBHS_OPMODE_DEVICE	Device
USBHS_OPMODE_HOST	Host
USBHS_OPMODE_OTG	OTG

Description

USB Operating Modes Enumeration

This data type provides enumeration of the operating modes supported by the USB module.

Remarks

None.

USBHS_PKTS_PER_MICROFRAME Enumeration

Provides an enumeration of the allowable isochronous packets per microframe when operating in High-Speed mode.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_ONE_PKTS,
  USBHS_TWO_PKTS,
  USBHS_THREE_PKTS,
  USBHS_RSVD_PKTS
} USBHS_PKTS_PER_MICROFRAME;
```

Description

USBHS Iso Packets Per Microframe Enumeration

This data type provides an enumeration of the allowable isochronous packets per microframe when operating in High-Speed mode.

Remarks

Used by PLIB_USBHS_EPnPacketsInMicroFrameSet.

For Transmit Endpoints: PIC32MZ EC device SFR fields: USBIENCSR0.MULT<1:0> = USBIENCSR0<12:11>, with USBCSR3.ENDPOINT = 1,2,...7

For Receive Endpoints: MG register fields: RXMAXP.m-1 = RXMAXP<12:11>, with INDEX.SelectedEndpoint = 1,2,...7 PIC32MZ EC device SFR fields: USBIENCSR1.MULT<1:0> = USBIENCSR1<12:11>, with USBCSR3.ENDPOINT = 1,2,...7

USBHS_SPEED Enumeration

Provides enumeration Host endpoint speeds.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_LOW_SPEED,
  USBHS_FULL_SPEED,
  USBHS_HIGH_SPEED
} USBHS_SPEED;
```

Description

USBHS Endpoint Speed Enumeration

This data type provides enumeration Host endpoint speeds.

Remarks

Used by PLIB_USBHS_EPnSpeedSet.

USBHS_TEST_SPEED Enumeration

Used as an argument for in setting module speeds in Test mode.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_TEST_LOW_SPEED,
  USBHS_TEST_FULL_SPEED,
  USBHS_TEST_HIGH_SPEED
} USBHS_TEST_SPEED;
```

Description

USBHS Test Speeds Enumeration

This data type is used as an argument for setting module speeds in Test mode.

Remarks

For use with PLIB_USBHS_TestSpeedForce.

USBHS_TRANSACTION_TYPE Enumeration

Provides an enumeration of transaction types.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_TRANSACTION_CONTROL,
  USBHS_TRANSACTION_ISO,
  USBHS_TRANSACTION_BULK,
  USBHS_TRANSACTION_INTERRUPT
} USBHS_TRANSACTION_TYPE;
```

Description

USBHS Endpoint Transaction Type Enumeration

This data type provides an enumeration of transaction types.

Remarks

Used by PLIB_USBHS_EPnTransactionTypeSet

USBHS_TXRX_FIFO_STATE Enumeration

Provides enumeration of receive and transmit FIFO states, as reported by status bits.

File

plib_usbhs.h

C

```
typedef enum {
  USBHS_TX_FIFO_EMPTY,
  USBHS_TX_FIFO_NOT_EMPTY,
  USBHS_RX_FIFO_FULL,
  USBHS_RX_FIFO_NOTFULL
} USBHS_TXRX_FIFO_STATE;
```

Members

Members	Description
USBHS_TX_FIFO_EMPTY	TX FIFO Empty
USBHS_TX_FIFO_NOT_EMPTY	TX FIFO NOT Empty
USBHS_RX_FIFO_FULL	RX FIFO Full
USBHS_RX_FIFO_NOTFULL	RX FIFO NOT Full

Description

USBHS FIFO State Enumeration

This data type provides enumeration of receive and transmit FIFO states, as reported by status bits.

Remarks

None.

For Transmit Endpoints: MG register fields: TXCSRL.FIFONotEmpty = TXCSRL<1>, with INDEX.SelectedEndpoint = 1,2,...7 PIC32MZ EC device SFR fields: USBIENCSR0.FIFONE = USBIENCSR0<17>, with USBCSR3.ENDPOINT = 1,2,...7

For Receive Endpoints: MG register fields: RXCSRL.FIFOFull = RXCSRL<1>, with INDEX.SelectedEndpoint = 1,2,...7 PIC32MZ EC device SFR fields: USBIENCSR1.FIFOFULL = USBIENCSR1<17>, with USBCSR3.ENDPOINT = 1,2,...7

USBHS_MAX_DMA_CHANNEL_NUMBER Macro

Number of available DMA Channels.

File

plib_usbhs.h

C

```
#define USBHS_MAX_DMA_CHANNEL_NUMBER 8
```

Description

Number of DMA Channels

This constant defines the number of available DMA Channels.

USBHS_MAX_EP_NUMBER Macro

Maximum number of endpoints supported (not including EP0).

File

plib_usbhs.h

C

```
#define USBHS_MAX_EP_NUMBER 7
```

Description

Maximum number of endpoints

This constant defines the maximum number of endpoints supported (not including EP0).

Files

Files

Name	Description
plib_usbhs.h	HS USB PLIB Interface Header for definitions common to the Hi-Speed USB module.

Description

This section lists the source and header files used by the library.

plib_usbhs.h

HS USB PLIB Interface Header for definitions common to the Hi-Speed USB module.

Enumerations

	Name	Description
	USBHS_CONFIGURATION	Provides the enumeration Configuration bits.
	USBHS_DATA01	Provides an enumeration data toggle for a packet.
	USBHS_DMA_ASSERT_TIMING	Provides enumeration DMA assertion timing (early versus late).
	USBHS_DMA_BURST_MODE	Provides enumeration of all DMA burst modes.
	USBHS_DMA_INTERRUPT	Provides enumeration of interrupts for DMA channels 0-7.
	USBHS_DMA_REQUEST_MODE	Used as an argument to set DMA request mode.
	USBHS_DMA_TRANSFER_MODE	Provides enumeration of all DMA transfer modes.
	USBHS_DYN_FIFO_PACKET_BUFFERING	Provides enumeration of dynamic FIFO double-packet versus single-packet buffering.
	USBHS_DYN_FIFO_SIZE	Provides enumeration of dynamic FIFO sizes.
	USBHS_ENDPOINT_DIRECTION	Used as an argument to identify an endpoint direction.
	USBHS_LPM_INTERRUPT	Provides an enumeration of LPM interrupt sources.
	USBHS_LPM_LINK_STATE	Provides enumeration requested device state after accepting an LPM transaction.
	USBHS_LPM_MODE	Provides enumeration of Link Power Management (LPM) modes.
	USBHS_OPMODES	Provides enumeration of operating modes supported by USB.
	USBHS_PKTS_PER_MICROFRAME	Provides an enumeration of the allowable isochronous packets per microframe when operating in High-Speed mode.
	USBHS_SPEED	Provides enumeration Host endpoint speeds.
	USBHS_TEST_SPEED	Used as an argument for in setting module speeds in Test mode.
	USBHS_TRANSACTION_TYPE	Provides an enumeration of transaction types.
	USBHS_TXRX_FIFO_STATE	Provides enumeration of receive and transmit FIFO states, as reported by status bits.

Functions

	Name	Description
	PLIB_USBHS_DeviceAddressGet	Returns the current USB device address.
	PLIB_USBHS_DeviceAddressSet	Sets the device address.
	PLIB_USBHS_DeviceAttach	This is function PLIB_USBHS_DeviceAttach.
	PLIB_USBHS_DeviceConnect	Tri-states the USB D+ and D- lines.
	PLIB_USBHS_DeviceDetach	This is function PLIB_USBHS_DeviceDetach.
	PLIB_USBHS_DeviceDisconnect	Tri-states the USB D+ and D- lines.
	PLIB_USBHS_DeviceEPFIFOLoad	This is function PLIB_USBHS_DeviceEPFIFOLoad.
	PLIB_USBHS_DeviceEPFIFOUnload	This is function PLIB_USBHS_DeviceEPFIFOUnload.
	PLIB_USBHS_DeviceRxEndpointConfigure	This is function PLIB_USBHS_DeviceRxEndpointConfigure.
	PLIB_USBHS_DeviceRxEndpointStallDisable	This is function PLIB_USBHS_DeviceRxEndpointStallDisable.
	PLIB_USBHS_DeviceRxEndpointStallEnable	This is function PLIB_USBHS_DeviceRxEndpointStallEnable.
	PLIB_USBHS_DeviceTxEndpointConfigure	This is function PLIB_USBHS_DeviceTxEndpointConfigure.
	PLIB_USBHS_DeviceTxEndpointPacketReady	This is function PLIB_USBHS_DeviceTxEndpointPacketReady.
	PLIB_USBHS_DeviceTxEndpointStallDisable	This is function PLIB_USBHS_DeviceTxEndpointStallDisable.
	PLIB_USBHS_DeviceTxEndpointStallEnable	This is function PLIB_USBHS_DeviceTxEndpointStallEnable.
	PLIB_USBHS_DMAErrorGet	This is function PLIB_USBHS_DMAErrorGet.
	PLIB_USBHS_DMAInterruptDisable	Disables DMA channel interrupts.
	PLIB_USBHS_DMAInterruptEnable	Enables DMA channel interrupts.
	PLIB_USBHS_DMAInterruptFlagsGet	Gets the DMA channel interrupt flags.
	PLIB_USBHS_DMAInterruptGet	This is function PLIB_USBHS_DMAInterruptGet.
	PLIB_USBHS_DMAOperationEnable	This is function PLIB_USBHS_DMAOperationEnable.
	PLIB_USBHS_Endpoint0FIFOFlush	This is function PLIB_USBHS_Endpoint0FIFOFlush.
	PLIB_USBHS_Endpoint0SetupPacketLoad	Loads the endpoint 0 FIFO with provided setup packet and then enables the endpoint transmit.
	PLIB_USBHS_Endpoint0SetupPacketUnload	This is function PLIB_USBHS_Endpoint0SetupPacketUnload.
	PLIB_USBHS_EndpointFIFOLoad	Loads the endpoint FIFO with provided data and then enables the endpoint transmit.
	PLIB_USBHS_EndpointFIFOUnload	Unloads the endpoint FIFO.
	PLIB_USBHS_EndpointRxFIFOFlush	This is function PLIB_USBHS_EndpointRxFIFOFlush.
	PLIB_USBHS_EndpointRxRequestClear	This is function PLIB_USBHS_EndpointRxRequestClear.
	PLIB_USBHS_EndpointRxRequestEnable	This is function PLIB_USBHS_EndpointRxRequestEnable.
	PLIB_USBHS_EndpointTxFIFOFlush	This is function PLIB_USBHS_EndpointTxFIFOFlush.
	PLIB_USBHS_EP0DataEndSet	This is function PLIB_USBHS_EP0DataEndSet.
	PLIB_USBHS_EP0INHandshakeClear	This is function PLIB_USBHS_EP0INHandshakeClear.
	PLIB_USBHS_EP0INHandshakeSend	This is function PLIB_USBHS_EP0INHandshakeSend.
	PLIB_USBHS_EP0INTokenSend	This is function PLIB_USBHS_EP0INTokenSend.
	PLIB_USBHS_EP0OUTHandshakeSend	This is function PLIB_USBHS_EP0OUTHandshakeSend.
	PLIB_USBHS_EP0RxPktRdyServiced	This is function PLIB_USBHS_EP0RxPktRdyServiced.
	PLIB_USBHS_EP0RxPktRdyServicedDataEnd	This is function PLIB_USBHS_EP0RxPktRdyServicedDataEnd.
	PLIB_USBHS_EP0SentStallClear	This is function PLIB_USBHS_EP0SentStallClear.
	PLIB_USBHS_EP0SetupEndServiced	This is function PLIB_USBHS_EP0SetupEndServiced.
	PLIB_USBHS_EP0StallDisable	This is function PLIB_USBHS_EP0StallDisable.
	PLIB_USBHS_EP0StallEnable	This is function PLIB_USBHS_EP0StallEnable.
	PLIB_USBHS_EP0StatusClear	This is function PLIB_USBHS_EP0StatusClear.
	PLIB_USBHS_EP0StatusGet	This is function PLIB_USBHS_EP0StatusGet.
	PLIB_USBHS_EP0TxPktRdy	This is function PLIB_USBHS_EP0TxPktRdy.
	PLIB_USBHS_EP0TxPktRdyDataEnd	This is function PLIB_USBHS_EP0TxPktRdyDataEnd.
	PLIB_USBHS_ExistsClockResetControl	This is function PLIB_USBHS_ExistsClockResetControl.
	PLIB_USBHS_ExistsEndpointFIFO	Identifies that the Endpoint FIFO feature exists on the Hi-Speed USB module.
	PLIB_USBHS_ExistsEndpointOperations	This is function PLIB_USBHS_ExistsEndpointOperations.
	PLIB_USBHS_ExistsEP0Status	This is function PLIB_USBHS_ExistsEP0Status.
	PLIB_USBHS_ExistsHighSpeedSupport	This is function PLIB_USBHS_ExistsHighSpeedSupport.
	PLIB_USBHS_ExistsInterrupts	This is function PLIB_USBHS_ExistsInterrupts.

	PLIB_USBHS_ExistsModuleControl	This is function PLIB_USBHS_ExistsModuleControl.
	PLIB_USBHS_ExistsRxEPStatus	This is function PLIB_USBHS_ExistsRxEPStatus.
	PLIB_USBHS_ExistsSoftReset	This is function PLIB_USBHS_ExistsSoftReset.
	PLIB_USBHS_ExistsTxEPStatus	This is function PLIB_USBHS_ExistsTxEPStatus.
	PLIB_USBHS_ExistsUSBIDControl	This is function PLIB_USBHS_ExistsUSBIDControl.
	PLIB_USBHS_FullOrHighSpeedIsConnected	This is function PLIB_USBHS_FullOrHighSpeedIsConnected.
	PLIB_USBHS_GenInterruptDisable	Disables a general interrupt for the USB module.
	PLIB_USBHS_GenInterruptEnable	Enables a general interrupt for the USB module.
	PLIB_USBHS_GenInterruptFlagsGet	Gets general interrupt flags.
	PLIB_USBHS_GetEP0CSRAddress	This is function PLIB_USBHS_GetEP0CSRAddress.
	PLIB_USBHS_GetEP0FIFOAddress	This is function PLIB_USBHS_GetEP0FIFOAddress.
	PLIB_USBHS_GetReceiveDataCount	This is function PLIB_USBHS_GetReceiveDataCount.
	PLIB_USBHS_GlobalInterruptDisable	This is function PLIB_USBHS_GlobalInterruptDisable.
	PLIB_USBHS_GlobalInterruptEnable	This is function PLIB_USBHS_GlobalInterruptEnable.
	PLIB_USBHS_HighSpeedDisable	This is function PLIB_USBHS_HighSpeedDisable.
	PLIB_USBHS_HighSpeedEnable	Sets the operation speed of the USB Module.
	PLIB_USBHS_HighSpeedIsConnected	This is function PLIB_USBHS_HighSpeedIsConnected.
	PLIB_USBHS_HostModeIsEnabled	This is function PLIB_USBHS_HostModeIsEnabled.
	PLIB_USBHS_HostRxEndpointConfigure	This is function PLIB_USBHS_HostRxEndpointConfigure.
	PLIB_USBHS_HostRxEndpointDataToggleClear	This is function PLIB_USBHS_HostRxEndpointDataToggleClear.
	PLIB_USBHS_HostTxEndpointConfigure	This is function PLIB_USBHS_HostTxEndpointConfigure.
	PLIB_USBHS_HostTxEndpointDataToggleClear	This is function PLIB_USBHS_HostTxEndpointDataToggleClear.
	PLIB_USBHS_InterruptEnableSet	Enables USB module event interrupts.
	PLIB_USBHS_IsBDevice	This is function PLIB_USBHS_IsBDevice.
	PLIB_USBHS_LoadEPInIndex	This is function PLIB_USBHS_LoadEPInIndex.
	PLIB_USBHS_ModuleSpeedGet	Returns the speed at which the module is operating.
	PLIB_USBHS_PhyIDMonitoringDisable	This is function PLIB_USBHS_PhyIDMonitoringDisable.
	PLIB_USBHS_PhyIDMonitoringEnable	This is function PLIB_USBHS_PhyIDMonitoringEnable.
	PLIB_USBHS_ResetDisable	This is function PLIB_USBHS_ResetDisable.
	PLIB_USBHS_ResetEnable	This is function PLIB_USBHS_ResetEnable.
	PLIB_USBHS_ResumeDisable	This is function PLIB_USBHS_ResumeDisable.
	PLIB_USBHS_ResumeEnable	This is function PLIB_USBHS_ResumeEnable.
	PLIB_USBHS_RxEPINTokenSend	This is function PLIB_USBHS_RxEPINTokenSend.
	PLIB_USBHS_RxEPStatusClear	This is function PLIB_USBHS_RxEPStatusClear.
	PLIB_USBHS_RxEPStatusGet	This is function PLIB_USBHS_RxEPStatusGet.
	PLIB_USBHS_RxInterruptDisable	Disables a RX endpoint interrupt for the USB module.
	PLIB_USBHS_RxInterruptEnable	Enables a RX endpoint interrupt for the USB module.
	PLIB_USBHS_RxInterruptFlagsGet	Gets RX endpoint interrupt flags.
	PLIB_USBHS_SessionDisable	This is function PLIB_USBHS_SessionDisable.
	PLIB_USBHS_SessionEnable	This is function PLIB_USBHS_SessionEnable.
	PLIB_USBHS_SoftResetDisable	Disables soft reset.
	PLIB_USBHS_SoftResetEnable	Enables soft reset.
	PLIB_USBHS_SuspendDisable	This is function PLIB_USBHS_SuspendDisable.
	PLIB_USBHS_SuspendEnable	This is function PLIB_USBHS_SuspendEnable.
	PLIB_USBHS_TestModeEnter	This is function PLIB_USBHS_TestModeEnter.
	PLIB_USBHS_TestModeExit	This is function PLIB_USBHS_TestModeExit.
	PLIB_USBHS_TxEPStatusClear	This is function PLIB_USBHS_TxEPStatusClear.
	PLIB_USBHS_TxEPStatusGet	This is function PLIB_USBHS_TxEPStatusGet.
	PLIB_USBHS_TxInterruptDisable	Disables a TX endpoint interrupt source for the USB module.
	PLIB_USBHS_TxInterruptEnable	Enables a TX endpoint Interrupt for the USB module.
	PLIB_USBHS_TxInterruptFlagsGet	Gets the TX endpoint interrupt flags.
	PLIB_USBHS_USBIDOverrideDisable	This is function PLIB_USBHS_USBIDOverrideDisable.
	PLIB_USBHS_USBIDOverrideEnable	This is function PLIB_USBHS_USBIDOverrideEnable.
	PLIB_USBHS_USBIDOverrideValueSet	This is function PLIB_USBHS_USBIDOverrideValueSet.

	PLIB_USBHS_VbusLevelGet	Returns the current VBUS level encode using the USBHS_VBUS_DETECT_LEVEL enumeration.
	PLIB_USBHS_VBUSLevelGet	This is function PLIB_USBHS_VBUSLevelGet.

Macros

	Name	Description
	USBHS_MAX_DMA_CHANNEL_NUMBER	Number of available DMA Channels.
	USBHS_MAX_EP_NUMBER	Maximum number of endpoints supported (not including EP0).

Description

Hi-Speed USB (HS USB) Module Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB PLIB for families of Microchip microcontrollers that feature the Hi-Speed USB module.

Remarks

Required to access the High Speed USB Module registers.

File Name

plib_usbhs.h

Company

Microchip Technology Inc.

Watchdog Timer Peripheral Library

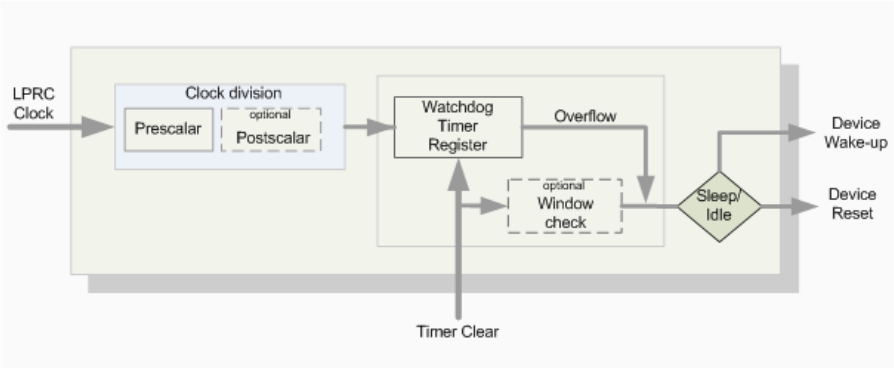
This section describes the Watchdog Timer (WDT) Peripheral Library.

Introduction

This library provides a low-level abstraction of the WDT Peripheral Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by abstracting differences from one microcontroller variant to another.

Description

The primary function of the Watchdog Timer (WDT) is to reset the microcontroller, in the event of a software malfunction, by resetting the device if it has not been cleared by software. To ensure that application does not hang, the application is supposed to reset the timer periodically. It can also be used to wake the device from Sleep or Idle mode. The WDT is a free-running timer, which uses the Low-Power RC (LPRC) Oscillator and requires no external components. Therefore, the WDT will continue to operate even if the system's primary clock source is stopped.



Using the Library

This topic describes the basic architecture of the WDT Peripheral Library and provides information and examples on its use.

Description

Interface Header File: `plib_wdt.h`

The interface to the WDT Peripheral Library is defined in the `plib_wdt.h` header file, which is included by the peripheral library header file, `peripheral.h`. Any C language source (`.c`) file that uses the WDT Peripheral Library must include `peripheral.h`.

Library File:

The WDT Peripheral Library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for information on how the library interacts with the framework.

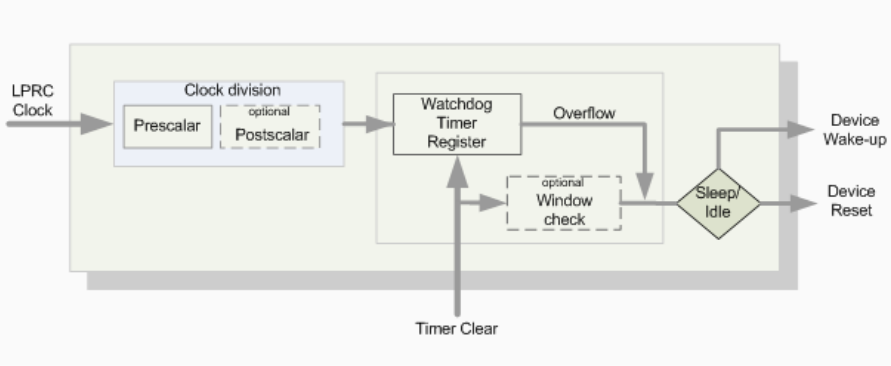
Hardware Abstraction Model

This library provides the low-level abstraction of the *Watchdog timer* module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Note: The interface provided is a superset of all of the functionality of the available Watchdog Timer modules on the device. Please refer to the **"Watchdog Timer"** chapter in the specific device data sheet or the family reference manual section specified in that chapter to determine the set of functions that are supported for each Watchdog Timer module on your device.

Description

Watchdog Timer Software Abstraction Block Diagram



The WDT uses the internal Low-Power RC (LPRC) Oscillator as the clock source. The clock is divided by the configured prescaler value. There may be one more postscaler divisors. The divided clock is then used to increment a counter. The WDT module uses the watchdog register as a timer. If there is no reset signal from the software and if the counter overflows, this results in a reset in normal mode. In the case of Sleep or Idle mode, the overflow will result in a device wake-up.

For Windowed mode, resetting the counter when the count is not in the specified window will also lead to a reset. If the device is in Sleep or Idle mode, the reset signal will be used to wake-up the device.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the WDT module

Library Interface Section	Description
General Configuration Functions	This includes module enable, module disable, watchdog timer window enable, watchdog timer window disable and the timer reset routines.
General Status Functions	Status routines for the WDT.

How the Library Works

This section describes how the Watchdog Timer Peripheral Library works.

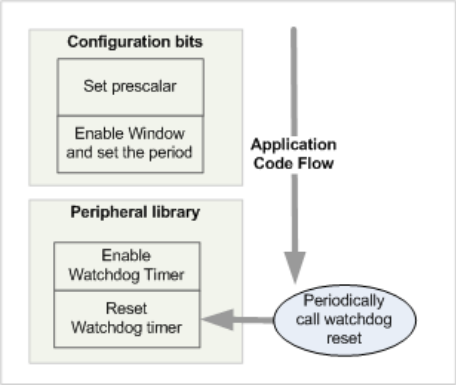
Description

Initializing the WDT module involves these processes.

- Selecting the postscaler
- Enabling the WDT using PLIB_WDT_Enable
- Selecting the window size if the application wants the WDT in Windowed mode using PLIB_WDT_WindowEnable

Note: Selecting the postscaler and window size should be done through the Configuration bits.

The application should periodically clear the timer once it is enabled; otherwise, a time-out will lead to a device Reset. Use PLIB_WDT_TimerClear to clear the WDT. A timer clear before a time-out may also lead to a reset in Windowed mode. The user application can clear the window only in the allowed window. Refer to the specific device data sheet to determine the allowed window period for your device.



Example

```
/*************************************************************
* This code fragment assumes the WDT is not enabled through   *
* device configuration bits. The Postscaler value must be set *
* through the device configuration and window size should be  *
* set if applicable                                           *
*************************************************/

/* Initializing the watchdog involves  */
PLIB_WDT_Enable(WDT_ID_0);

//Application loop
while(1)
{
    PLIB_WDT_TimerClear(WDT_ID_0);

    //user code
}
```

Note: Refer to the specific device data sheet for information on the allowed window periods for your device and other Configuration bit settings.

Configuring the Library

The library is configured for the supported Watchdog Timer module when the processor is chosen in the MPLAB X IDE.

Library Interface

a) General Configuration Functions

	Name	Description
	PLIB_WDT_Disable	Disables the WDT module.
	PLIB_WDT_Enable	Enables the WDT module.
	PLIB_WDT_WindowDisable	Disables the WDT Windowed mode.
	PLIB_WDT_WindowEnable	Enables the WDT Window mode.
	PLIB_WDT_TimerClear	Resets the WDT module.

b) General Status Functions

	Name	Description
	PLIB_WDT_IsEnabled	Returns the watchdog timer on/off(enable/disable) status.
	PLIB_WDT_PostscalerValueGet	Returns the WDT postscaler value.
	PLIB_WDT_SleepModePostscalerValueGet	Returns the WDT Sleep Mode postscaler value.

c) Feature Existence Functions

	Name	Description
	PLIB_WDT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the WDT module.
	PLIB_WDT_ExistsPostscalerValue	Identifies whether the PostscalerValue feature exists on the WDT module.
	PLIB_WDT_ExistsTimerClear	Identifies whether the TimerClear feature exists on the WDT module.

	PLIB_WDT_ExistsWindowEnable	Identifies whether the WindowEnable feature exists on the WDT module.
	PLIB_WDT_ExistsSleepModePostscalerValue	Identifies whether the SleepModePostscalerValue feature exists on the WDT module.

d) Data Types and Constants

	Name	Description
	WDT_MODULE_ID	Identifies the supported Watchdog Timer modules.

Description

This section describes the Application Programming Interface (API) functions of the Watchdog Timer Peripheral Library.

Refer to each section for a detailed description.

a) General Configuration Functions

PLIB_WDT_Disable Function

Disables the WDT module.

File

plib_wdt.h

C

```
void PLIB_WDT_Disable(WDT_MODULE_ID index);
```

Returns

None.

Description

This function disables the WDT module if it is enabled in software.

Remarks

This function will not disable the WDT module if it is enabled through its Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

The WDT module must be enabled through software.

Example

```
PLIB_WDT_Disable ( WDT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

void PLIB_WDT_Disable (WDT_MODULE_ID index)

PLIB_WDT_Enable Function

Enables the WDT module.

File

plib_wdt.h

C

```
void PLIB_WDT_Enable(WDT_MODULE_ID index);
```

Returns

None.

Description

This function enables the WDT module. If it is already enabled through the Configuration bits, it will keep it enabled.

Remarks

Calling this function is not necessary if it is enabled through its Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_WDT_Enable ( WDT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

void PLIB_WDT_Enable (WDT_MODULE_ID index)

PLIB_WDT_WindowDisable Function

Disables the WDT Windowed mode.

File

plib_wdt.h

C

```
void PLIB_WDT_WindowDisable(WDT_MODULE_ID index);
```

Returns

None.

Description

This function disables the WDT Windowed mode.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsWindowEnable in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_WDT_WindowDisable ( WDT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

void PLIB_WDT_WindowDisable (WDT_MODULE_ID index)

PLIB_WDT_WindowEnable Function

Enables the WDT Window mode.

File

plib_wdt.h

C

```
void PLIB_WDT_WindowEnable(WDT_MODULE_ID index);
```

Returns

None.

Description

This function enables the WDT Windowed mode.

Remarks

The window size must be set through the Configuration bits.

The example code doesn't include the settings that should be done through the Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsWindowEnable in your application to determine whether this feature is available.

Preconditions

The window size must be set through the Configuration bits.

Example

```
PLIB_WDT_WindowEnable ( WDT_ID_0 );
PLIB_WDT_Enable ( WDT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

void PLIB_WDT_WindowEnable (WDT_MODULE_ID index)

PLIB_WDT_TimerClear Function

Resets the WDT module.

File

plib_wdt.h

C

```
void PLIB_WDT_TimerClear(WDT_MODULE_ID index);
```

Returns

None.

Description

This function resets the WDT module. The WDT module should be cleared periodically before the count crosses and forces the device to reset.

Remarks

Resetting the device before the count reaches the window will cause a reset in Windowed mode.

The example code doesn't include the settings that should be done through the Configuration bits.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsTimerClear in your application to determine whether this feature is available.

Preconditions

None.

Example

```
PLIB_WDT_Enable ( WDT_ID_0 );

//Application loop
```

```
while(1)
{
    PLIB_WDT_TimerClear ( WDT_ID_0 );
    //user code
}
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

void PLIB_WDT_TimerClear (WDT_MODULE_ID index)

b) General Status Functions

PLIB_WDT_IsEnabled Function

Returns the watchdog timer on/off(enable/disable) status.

File

plib_wdt.h

C

```
bool PLIB_WDT_IsEnabled(WDT_MODULE_ID index);
```

Returns

- true - If the watchdog timer is on
- false - If the watchdog timer is off

Description

Returns the 'true', if the watchdog timer is already ON. Otherwise returns 'false'.

Remarks

This function returns 'true' if the device is enabled either though the Configuration bits or in the software.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsEnableControl in your application to determine whether this feature is available.

Preconditions

None.

Example

```
if (PLIB_WDT_IsEnabled ( WDT_ID_0 ) )
{
    //Do some action
}
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

bool PLIB_WDT_IsEnabled (WDT_MODULE_ID index)

PLIB_WDT_PostscalerValueGet Function

Returns the WDT postscaler value.

File

plib_wdt.h

C

```
char PLIB_WDT_PostscalerValueGet(WDT_MODULE_ID index);
```

Returns

The postscaler value.

Description

This function returns the WDT postscaler value. The value will correspond to a division factor.

Remarks

The value returned will be right-aligned.

Refer to the specific device data sheet to get the division factor corresponding to the value.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsPostscalerValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t value;

PLIB_WDT_Enable ( WDT_ID_0 );
value = PLIB_WDT_PostscalerValueGet ( WDT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

char PLIB_WDT_PostscalerValueGet (WDT_MODULE_ID index)

PLIB_WDT_SleepModePostscalerValueGet Function

Returns the WDT Sleep Mode postscaler value.

File

plib_wdt.h

C

```
char PLIB_WDT_SleepModePostscalerValueGet(WDT_MODULE_ID index);
```

Returns

The Sleep Mode postscaler value.

Description

This function returns the WDT postscaler value in Sleep Mode. The value will correspond to a division factor.

Remarks

The value returned will be right-aligned.

Refer to the specific device data sheet to get the division factor corresponding to the value.

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use PLIB_WDT_ExistsSleepModePostscalerValue in your application to determine whether this feature is available.

Preconditions

None.

Example

```
uint8_t value;
value = PLIB_WDT_SleepModePostscalerValueGet ( WDT_ID_0 );
```

Parameters

Parameters	Description
index	Identifies the desired WDT module

Function

char PLIB_WDT_SleepModePostscalerValueGet (WDT_MODULE_ID index)

c) Feature Existence Functions

PLIB_WDT_ExistsEnableControl Function

Identifies whether the EnableControl feature exists on the WDT module.

File

plib_wdt.h

C

```
bool PLIB_WDT_ExistsEnableControl(WDT_MODULE_ID index);
```

Returns

Existence of the EnableControl feature:

- true - When EnableControl feature is supported on the device
- false - When EnableControl feature is not supported on the device

Description

This function identifies whether the EnableControl feature is available on the WDT module. When this function returns true, these functions are supported on the device:

- PLIB_WDT_Enable
- PLIB_WDT_Disable
- PLIB_WDT_IsEnabled

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_WDT_ExistsEnableControl(WDT_MODULE_ID index)

PLIB_WDT_ExistsPostscalerValue Function

Identifies whether the PostscalerValue feature exists on the WDT module.

File

plib_wdt.h

C

```
bool PLIB_WDT_ExistsPostscalerValue(WDT_MODULE_ID index);
```

Returns

- true - The PostscalerValue feature is supported on the device
- false - The PostscalerValue feature is not supported on the device

Description

This function identifies whether the PostscalerValue feature is available on the WDT module. When this function returns true, this function is supported on the device:

- PLIB_WDT_PostscalerValueGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_WDT_ExistsPostscalerValue(WDT_MODULE_ID index)

PLIB_WDT_ExistsTimerClear Function

Identifies whether the TimerClear feature exists on the WDT module.

File

plib_wdt.h

C

```
bool PLIB_WDT_ExistsTimerClear(WDT_MODULE_ID index);
```

Returns

- true - The TimerClear feature is supported on the device
- false - The TimerClear feature is not supported on the device

Description

This function identifies whether the TimerClear feature is available on the WDT module. When this function returns true, this function is supported on the device:

- PLIB_WDT_TimerClear

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_WDT_ExistsTimerClear(WDT_MODULE_ID index)

PLIB_WDT_ExistsWindowEnable Function

Identifies whether the WindowEnable feature exists on the WDT module.

File

plib_wdt.h

C

```
bool PLIB_WDT_ExistsWindowEnable(WDT_MODULE_ID index);
```

Returns

- true - The WindowEnable feature is supported on the device
- false - The WindowEnable feature is not supported on the device

Description

This function identifies whether the WindowEnable feature is available on the WDT module. When this function returns true, these functions are supported on the device:

- PLIB_WDT_WindowEnable
- PLIB_WDT_WindowDisable

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_WDT_ExistsWindowEnable(WDT_MODULE_ID index)

PLIB_WDT_ExistsSleepModePostscalerValue Function

Identifies whether the SleepModePostscalerValue feature exists on the WDT module.

File

plib_wdt.h

C

```
bool PLIB_WDT_ExistsSleepModePostscalerValue(WDT_MODULE_ID index);
```

Returns

- true - The SleepModePostscalerValue feature is supported on the device
- false - The SleepModePostscalerValue feature is not supported on the device

Description

This function identifies whether the SleepModePostscalerValue feature is available on the WDT module. When this function returns true, this function is supported on the device:

- PLIB_WDT_SleepModePostscalerValueGet

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the device instance

Function

PLIB_WDT_ExistsSleepModePostscalerValue(WDT_MODULE_ID index)

d) Data Types and Constants

WDT_MODULE_ID Enumeration

Identifies the supported Watchdog Timer modules.

File

plib_wdt_help.h

C

```
typedef enum {
  WDT_ID_0,
  WDT_NUMBER_OF_MODULES
} WDT_MODULE_ID;
```

Members

Members	Description
WDT_ID_0	WDT Module 1 ID
WDT_NUMBER_OF_MODULES	Number of available WDT modules.

Description

Watchdog Timer (WDT) Module ID

This enumeration identifies the available Watchdog Timer modules.

Remarks

The caller should not rely on the specific numbers assigned to any of these values as they may change from one processor to the next.

Files

Files

Name	Description
plib_wdt.h	Watchdog Timer (WDT) Peripheral Library interface header for Watchdog Timer common definitions.
plib_wdt_help.h	

Description

This section lists the source and header files used by the library.

plib_wdt.h

Watchdog Timer (WDT) Peripheral Library interface header for Watchdog Timer common definitions.

Functions

	Name	Description
	PLIB_WDT_Disable	Disables the WDT module.
	PLIB_WDT_Enable	Enables the WDT module.
	PLIB_WDT_ExistsEnableControl	Identifies whether the EnableControl feature exists on the WDT module.
	PLIB_WDT_ExistsPostscalerValue	Identifies whether the PostscalerValue feature exists on the WDT module.
	PLIB_WDT_ExistsSleepModePostscalerValue	Identifies whether the SleepModePostscalerValue feature exists on the WDT module.
	PLIB_WDT_ExistsTimerClear	Identifies whether the TimerClear feature exists on the WDT module.
	PLIB_WDT_ExistsWindowEnable	Identifies whether the WindowEnable feature exists on the WDT module.
	PLIB_WDT_IsEnabled	Returns the watchdog timer on/off(enable/disable) status.
	PLIB_WDT_PostscalerValueGet	Returns the WDT postscaler value.
	PLIB_WDT_SleepModePostscalerValueGet	Returns the WDT Sleep Mode postscaler value.
	PLIB_WDT_TimerClear	Resets the WDT module.
	PLIB_WDT_WindowDisable	Disables the WDT Windowed mode.
	PLIB_WDT_WindowEnable	Enables the WDT Window mode.

Description

Watchdog Timer (WDT) Peripheral Library Interface Header

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the Watchdog Timer Peripheral Library for all families of Microchip microcontrollers. The definitions in this file are common to the Watchdog Timer peripheral.

File Name

plib_wdt.h

Company

Microchip Technology Inc.

plib_wdt_help.h

Enumerations

	Name	Description
	WDT_MODULE_ID	Identifies the supported Watchdog Timer modules.

Section

Data Types & Constants

Sample Library Help

This section describes the Sample Library that is available in MPLAB Harmony.

Introduction

This library provides a simple example of a MPLAB Harmony module.

Description

The Sample Library provides a simple example of a MPLAB Harmony module; however, its actual functionality is very limited and is not suitable for a practical application. The Sample Library provides an interface function to accept a single integer data value, another to see if the module currently holds any data, and a third to obtain any data currently held. The library can only hold up to two integer values at any given time.

Using the Library

This topic describes the basic architecture of the Sample Library and provides information and examples on its use.

Description

The interface to the Sample Library is defined in the `sample_module.h` header file.

Any C language source (`.c`) file that uses the Sample Library must include `sample_module.h`.

Library Overview

This section provides an overview of the Sample Library.

Description

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Sample Library.

Library Interface Section	Description
System Interface Functions	Functions used by the MPLAB Harmony system configuration to initialize and maintain the sample library module.
Client Interface Functions	Functions used by client modules and applications to interact with the Sample module.
Data Types and Constants	This section describes the various data types utilized in the Sample module interface.

How the Library Works

This section provides information on how the Sample Library works.

Core Functionality

Provides information on the core functionality of the Sample Library.

Description

The primary purpose of the Sample Library is to serve as an example of a dynamic MPLAB Harmony module (similar to a middleware library module). The library implements the system interface functions normally supported by a MPLAB Harmony module and provides a simple client-level interface to store and retrieve an integer data value. The library can hold up to two integer values per instance of the library. And, while it has no associated hardware or interrupts, the library can be used in an interrupt-driven configuration utilizing a timer (or other appropriate peripheral) providing it is appropriately configured and initialized.

Configuring the Library

Macros

	Name	Description
	SAMPLE_MODULE_INTERRUPT_MODE	Determines whether or not the sample module will support interrupt-driven execution mode.
	SAMPLE_MODULE_INSTANCES_NUMBER	Defines how many instances of the sample module are supported.
	SAMPLE_MODULE_TIMEOUT	Defines OSAL mutex timeout for Sample module (in milliseconds).

Description

The configuration of the Sample Library is based on the file `system_config.h`.

This header file contains the configuration selections for the Sample Library. Based on the selections made, the Sample Library may support the selected features.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

SAMPLE_MODULE_INTERRUPT_MODE Macro

Determines whether or not the sample module will support interrupt-driven execution mode.

File

sample_module_config_template.h

C

```
#define SAMPLE_MODULE_INTERRUPT_MODE false
```

Description

Sample Module, Interrupt Mode

This macro determines whether or not the sample module will support interrupt interrupt driven execution. If defined as true, then the sample module will enable, disable, and clear the interrupt source identified in the SAMPLE_MODULE_INIT_DATA structure. Otherwise, it shoud be defined as false and the module will only support polled operation.

SAMPLE_MODULE_INSTANCES_NUMBER Macro

Defines how many instances of the sample module are supported.

File

sample_module_config_template.h

C

```
#define SAMPLE_MODULE_INSTANCES_NUMBER 2
```

Description

Sample Module, Number of Instances

This macro defines how many instances of the sample module are supported.

Remarks

Defaults to 1 and displays a build warning if not specified..

SAMPLE_MODULE_TIMEOUT Macro

Defines OSAL mutex timeout for Sample module (in milliseconds).

File

sample_module_config_template.h

C

```
#define SAMPLE_MODULE_TIMEOUT 100
```

Description

Sample Module, Timeout

This macro defines how many milliseconds the sample module will block, waiting to obtain a mutex, when accessing its internal shared resources. This is only utilized in interrupt-driven modes, but needs to be defined to build correclty.

Remarks

Used by OSAL_MUTEX_Lock.

Building the Library

This section lists the files that are available in the Sample Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/sample`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/sample_module.h`	This file provides the interface definitions of the Sample Library.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/sample_module.c`	Sample Library implementation.

Optional File(s)

There are no optional files for the Sample Library.

Module Dependencies

The Sample Library is dependent upon the following libraries and header files.

- Interrupt System Service Library
- Operating System Abstraction Layer (OSAL)
- sys_common.h
- sys_module.h

Library Interface

a) System Interface Functions

	Name	Description
	SAMPLE_Initialize	Initializes the sample module.
	SAMPLE_Reinitialize	Reinitializes the sample module and refreshes any associated internal settings.
	SAMPLE_Deinitialize	Deinitializes the specified instance of the sample module.
	SAMPLE_Tasks	Maintains the sample module instance's state machine.
	SAMPLE_Status	Provides the current status of the sample module instance.

b) Client Interface Functions

	Name	Description
	SAMPLE_DataGive	Gives data to the sample module.
	SAMPLE_DataGet	Gets data from the sample module.
	SAMPLE_DataStatus	Identifies the current status of the data processed by the module.

c) Data Types and Constants

	Name	Description
	SAMPLE_INDEX_0	Labels for module index numbers.
	SAMPLE_INDEX_1	This is macro SAMPLE_INDEX_1.
	SAMPLE_MODULE_DATA_STATUS	Possible data processing states.
	SAMPLE_MODULE_INIT_DATA	Data necessary to initialize the sample module.

	_sample_module_data_status	Possible data processing states.
	_sample_module_init_data	Data necessary to initialize the sample module.

Description

This section describes the APIs of the Sample Library. Refer to each section for a description.

a) System Interface Functions

SAMPLE_Initialize Function

Initializes the sample module.

File

sample_module.h

C

```
SYS_MODULE_OBJ SAMPLE_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a system module object. Otherwise, it returns SYS_MODULE_OBJ_INVALID. The returned object handle must be passed as an argument to SAMPLE_Reinitialize, SAMPLE_Deinitialize, SAMPLE_Tasks and SAMPLE_Status routines.

Description

This routine initializes the sample module, making it ready for clients to open and use it.

Remarks

This routine must be called before any other sample module routine is called.

This routine should only be called once during system initialization unless SAMPLE_Deinitialize is called to deinitialize the module instance.

This routine will NEVER block for internal state changes. If the operation requires time to complete, it will be reported by the SAMPLE_Status operation. The system can use SAMPLE_Status to find out when the associated instance of the sample module is in a ready state.

Preconditions

None.

Example

```
SAMPLE_MODULE_INIT_DATA init;
SYS_MODULE_OBJ          obj;

// Populate the sample initialization structure
init.sys.powerState     = SYS_MODULE_POWER_RUN_FULL;
init.dataSome           = 42;

// Initialize the first instance of the sample module.
obj = SAMPLE_Initialize(SAMPLE_INDEX_0, (SYS_MODULE_INIT *)&init);
if (SYS_MODULE_OBJ_INVALID == obj)
{
    // Handle failure
}
```

Parameters

Parameters	Description
index	Index for the module instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the module.

Function

SYS_MODULE_OBJ SAMPLE_Initialize(const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init)

SAMPLE_Reinitialize Function

Reinitializes the sample module and refreshes any associated internal settings.

File

sample_module.h

C

```
void SAMPLE_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None

Description

This routine reinitializes the sample module and refreshes any associated settings using the initialization data given. However, it will not interrupt ongoing client operations.

Remarks

This function can be called multiple times to reinitialize the module.

This operation can be used to refresh any supported internal settings as specified by the initialization data or to change the power state of the module.

This routine will NEVER block for internal state changes. If the operation requires time to compete, it will be reported by the SAMPLE_Status operation. The system can use SAMPLE_Status to find out when the specified sample module instance is in a ready state.

Preconditions

Function SAMPLE_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ          obj;    // Returned from SAMPLE_Initialize
SAMPLE_MODULE_INIT_DATA init;
SYS_STATUS              status;

// Populate the sample initialization structure
init.sys.powerState     = SYS_MODULE_POWER_SLEEP;
init.dataSome           = 0;

SAMPLE_Reinitialize(obj, (SYS_MODULE_INIT*)&init);

status = SAMPLE_Status(obj);
if (SYS_STATUS_BUSY == status)
{
    // Check again later to ensure the driver is ready
}
else if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Sample module object handle, returned from the SAMPLE_Initialize routine
init	Pointer to the initialization data structure

Function

void SAMPLE_Reinitialize(SYS_MODULE_OBJ object,
const SYS_MODULE_INIT * const init)

SAMPLE_Deinitialize Function

Deinitializes the specified instance of the sample module.

File

sample_module.h

C

```
void SAMPLE_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the sample module, disabling its operation. Invalidates all the internal data and client operations.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This routine will NEVER block waiting for internal state changes. If the operation requires time to complete, it will be reported by the SAMPLE_Status operation. The system can to use SAMPLE_Status to find out when the module is in the SYS_STATUS_UNINITIALIZED state.

Preconditions

Function SAMPLE_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ handle must have been returned.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from SAMPLE_Initialize
SYS_STATUS          status;

SAMPLE_Deinitialize(object);

status = SAMPLE_Status(object);
if (SYS_MODULE_DEINITIALIZED == status)
{
    // Check again later if you need to know when the sample module
    // instance has been deinitialized.
}
```

Parameters

Parameters	Description
object	Sample module instance's object handle, returned from the SAMPLE_Initialize routine.

Function

void SAMPLE_Deinitialize (SYS_MODULE_OBJ object)

SAMPLE_Tasks Function

Maintains the sample module instance's state machine.

File

sample_module.h

C

```
void SAMPLE_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the sample module instance's internal state machine.

Remarks

This routine should not be called directly a client of the sample module. It is called by the system's Tasks routine (SYS_Tasks) or by an appropriate ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The SAMPLE_Initialize routine must have been called for the specified SAMPLE module instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from SAMPLE_Initialize

while (true)
{
    SAMPLE_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified sample module instance (returned from SAMPLE_Initialize).

Function

void SAMPLE_Tasks(SYS_MODULE_OBJ object)

SAMPLE_Status Function

Provides the current status of the sample module instance.

File

sample_module.h

C

```
SYS_STATUS SAMPLE_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_ERROR - Indicates that the module instance is in an error state. Note: Any value less than SYS_STATUS_ERROR is also an error state.
- SYS_STATUS_UNINITIALIZED - Indicates that the driver has been deinitialized.

Description

This routine provides the current status of the sample module instance.

Remarks

This value is greater than SYS_STATUS_ERROR

- SYS_STATUS_BUSY - Indicates that the module instance is busy with a previous system level operation and cannot start another
- SYS_STATUS_READY - Indicates that the module instance is running and ready to begin another system level operation.

Any value greater than SYS_STATUS_READY is also a normal running state in which the module is ready to accept new operations

The this operation can be used to determine when any of the system operations have completed.

If the status operation returns SYS_STATUS_BUSY, the a previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This routine will NEVER block waiting for an internal state change.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

Function SAMPLE_Initialize must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from SAMPLE_Initialize
SYS_STATUS          status;

status = SAMPLE_Status(object);
else if (SYS_STATUS_ERROR >= status)
{
    // Handle error
```

```
}
```

Parameters

Parameters	Description
object	Sample module instance's object handle, returned from the SAMPLE_Initialize routine.

Function

SYS_STATUS SAMPLE_Status (SYS_MODULE_OBJ object)

b) Client Interface Functions

SAMPLE_DataGive Function

Gives data to the sample module.

File

sample_module.h

C

```
bool SAMPLE_DataGive(const SYS_MODULE_INDEX index, int data);
```

Returns

- true - If successful.
- false - If an error occurred.

Description

This function gives data to the sample module.

Remarks

Any data value given to the sample module will be available via the SAMPLE_DataGet function later, after the sample module state machine is ready to give it back.

Preconditions

The sample module should be initialized before calling this API function.

Example

```
if (SAMPLE_DataGive(SAMPLE_INDEX_0, 42))
{
    // Data given
}
```

Parameters

Parameters	Description
index	Index of the desired module instance.
data	Data value to give to the sample module.

Function

bool SAMPLE_DataGive (const SYS_MODULE_INDEX index, int data)

SAMPLE_DataGet Function

Gets data from the sample module.

File

sample_module.h

C

```
bool SAMPLE_DataGet(const SYS_MODULE_INDEX index, int * data);
```

Returns

- true - If successful.
- false - If an error occurred. If an error occurs, the variable pointed to by pData is not updated.

Description

This function gets data from the sample module. If no data has been given to the sample module yet, it will return a failure

Remarks

Any data value given to the sample module by the SAMPLE_DataGive function will be available via this function later, after the sample module state machine is ready to give it back.

Preconditions

The sample module should be initialized before calling this API function.

Example

```
int data;

if (SAMPLE_DataGet(SAMPLE_INDEX_0, &data))
{
    // Data valid
}
```

Parameters

Parameters	Description
index	Index of the desired module instance.
pData	Pointer to the variable to receive the data value.

Function

bool SAMPLE_DataGet (const SYS_MODULE_INDEX index, int *pData)

SAMPLE_DataStatus Function

Identifies the current status of the data processed by the module.

File

sample_module.h

C

```
SAMPLE_MODULE_DATA_STATUS SAMPLE_DataStatus(const SYS_MODULE_INDEX index);
```

Returns

- SAMPLE_MODULE_DATA_NONE - If the sample module has no data to give.
- SAMPLE_MODULE_DATA_BUSY - If the sample module is busy processing and will have data available soon.
- SAMPLE_MODULE_DATA_AVAILABLE - If the sample module has data available.

Description

This function identifies the current status of the data processed by the module.

Remarks

In a polled "super-loop" system, you cannot poll this function waiting for the sample module to change status from SAMPLE_MODULE_DATA_BUSY to SAMPLE_MODULE_DATA_AVAILABLE. The loop must be allowed to continue so that the sample module can finish processing the data before it will become available.

Preconditions

The sample module must be initialized before calling this API function.

Example

```
if (SAMPLE_MODULE_DATA_BUSY == SAMPLE_DataStatus(SAMPLE_INDEX_0))
{
    // Check again later.
}
```

Parameters

Parameters	Description
index	Index of the desired module instance.

Function

SAMPLE_MODULE_DATA_STATUS SAMPLE_DataStatus (const SYS_MODULE_INDEX index)

c) Data Types and Constants

SAMPLE_INDEX_0 Macro

Labels for module index numbers.

File

sample_module.h

C

```
#define SAMPLE_INDEX_0 0
```

Description

Module Instance Indexes

The following symbolic labels can be used in place of numeric literals to identify module indexes.

SAMPLE_INDEX_1 Macro

File

sample_module.h

C

```
#define SAMPLE_INDEX_1 1
```

Description

This is macro SAMPLE_INDEX_1.

SAMPLE_MODULE_DATA_STATUS Enumeration

Possible data processing states.

File

sample_module.h

C

```
typedef enum _sample_module_data_status {
  SAMPLE_MODULE_DATA_NONE = 0,
  SAMPLE_MODULE_DATA_BUSY,
  SAMPLE_MODULE_DATA_AVAILABLE
} SAMPLE_MODULE_DATA_STATUS;
```

Members

Members	Description
SAMPLE_MODULE_DATA_NONE = 0	Indicates that the sample module has no data available.
SAMPLE_MODULE_DATA_BUSY	Indicates that the sample module is busy processing and will have data available soon.
SAMPLE_MODULE_DATA_AVAILABLE	Indicates that the sample module has available data.

Description

Sample Module Data Status

This enumeration defines the possible data processing states of the sample module.

Remarks

This status is provided by the SAMPLE_DataStatus function..

SAMPLE_MODULE_INIT_DATA Structure

Data necessary to initialize the sample module.

File

sample_module.h

C

```
typedef struct _sample_module_init_data {
  SYS_MODULE_INIT moduleInit;
  int dataSome;
  TMR_MODULE_ID tmr;
  TMR_PRESCALE prescale;
  uint16_t period;
  INT_SOURCE interrupt;
  INT_VECTOR vector;
  INT_PRIORITY_LEVEL priority;
  INT_SUBPRIORITY_LEVEL subpriority;
} SAMPLE_MODULE_INIT_DATA;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	Standard system module init data.
int dataSome;	Some initialization data.
TMR_MODULE_ID tmr;	Timer used in interrupt-driven configurations.
TMR_PRESCALE prescale;	Timer prescaler setting.
uint16_t period;	Timer period setting.
INT_SOURCE interrupt;	Timer interrupt source.
INT_VECTOR vector;	Timer interrupt vector.
INT_PRIORITY_LEVEL priority;	Timer interrupt priority.
INT_SUBPRIORITY_LEVEL subpriority;	Timer interrupt subpriority.

Description

Sample Module Initialization Data

This data structure contains data necessary to initialize an instance of the sample module.

Remarks

A pointer to a persistent copy of this structure must be passed into the SAMPLE_Initialize function.

Files

Files

Name	Description
sample_module.h	Sample of a MPLAB Harmony library module.
sample_module_config_template.h	Sample module configuration templates.

Description

This section lists the source and header files used by the SAMPLE Library.

sample_module.h

Sample of a MPLAB Harmony library module.

Enumerations

	Name	Description
	_sample_module_data_status	Possible data processing states.

	SAMPLE_MODULE_DATA_STATUS	Possible data processing states.

Functions

	Name	Description
	SAMPLE_DataGet	Gets data from the sample module.
	SAMPLE_DataGive	Gives data to the sample module.
	SAMPLE_DataStatus	Identifies the current status of the data processed by the module.
	SAMPLE_Deinitialize	Deinitializes the specified instance of the sample module.
	SAMPLE_Initialize	Initializes the sample module.
	SAMPLE_Reinitialize	Reinitializes the sample module and refreshes any associated internal settings.
	SAMPLE_Status	Provides the current status of the sample module instance.
	SAMPLE_Tasks	Maintains the sample module instance's state machine.

Macros

	Name	Description
	SAMPLE_INDEX_0	Labels for module index numbers.
	SAMPLE_INDEX_1	This is macro SAMPLE_INDEX_1.

Structures

	Name	Description
	_sample_module_init_data	Data necessary to initialize the sample module.
	SAMPLE_MODULE_INIT_DATA	Data necessary to initialize the sample module.

Description

MPLAB Harmony Sample Library Module Interface Header

This is a sample of a MPLAB Harmony library interface module. This file defines the system and application interface.

File Name

sample_module.h

Company

Microchip Technology Inc.

sample_module_config_template.h

Sample module configuration templates.

Macros

	Name	Description
	SAMPLE_MODULE_INSTANCES_NUMBER	Defines how many instances of the sample module are supported.
	SAMPLE_MODULE_INTERRUPT_MODE	Determines whether or not the sample module will support interrupt-driven execution mode.
	SAMPLE_MODULE_TIMEOUT	Defines OSAL mutex timeout for Sample module (in milliseconds).

Description

Sample Module Configuration Templates

This file contains examples of constants to configure the sample module.

File Name

sample_module_config_template.h

Company

Microchip Technology Inc.

System Service Libraries Help

This section provides descriptions of the System Service libraries that are available in MPLAB Harmony.

System Service Overview

This section provides an overview of the System Service libraries that are available in MPLAB Harmony.

Introduction

Describes system services provided by MPLAB Harmony.

Description

MPLAB Harmony provides system service libraries to support common functionality and manage resources that are shared by multiple drivers, libraries, and other modules.

A system service encapsulates code that manages a shared resource or implements a common capability in a single location so that it does not need to be replicated by individual drivers and libraries. This feature eliminates duplicated code and creates consistency across all modules, and also helps to eliminate potential conflicts and complex configuration issues and runtime interactions, resulting in a smaller and simpler overall solution.

System services may directly manage one or more peripherals or core processor capabilities by utilizing peripheral libraries, special function registers, special CPU instructions, or coprocessor registers. Some system services may utilize drivers, other system services, or even entire middleware stacks to share or emulate a common resource.

System services may be implemented statically (possibly generated by the MPLAB Harmony Configurator (MHC)) or dynamically to support multiple channels and instances like a driver. However, system services will not normally provide common *Open* and *Close* functions like a device driver.

In general, the distinguishing feature of a system service is that it implements a common capability that would otherwise "cut horizontally" across multiple modules in a system, causing interoperability and compatibility conflicts if the capability were implemented by other libraries.

System service functions use the following naming convention:

`SYS_<module-abbreviation>_[<feature-short-name>]<operation>`

Where,

- `SYS_` indicates that this is a system service function
- `<module-abbreviation>` is the abbreviated name of the system service module to which this function belongs
- `[<feature-short-name>]` is an optional short (or shortened) name that identifies the feature of the associated system service module to which this function refers. The feature short name will appear in the name of all functions that interact with or provide access to that particular feature.
- `<operation>` is a verb that identifies the action taken by the function

For example, SYS_TMR_AlarmSet, where:

- `<module-abbreviation>` = `TMR`, which indicates that this is a Timer System Service function
- `<feature-short-name>` = `Alarm`, which indicates that this function controls the alarm feature of the Timer System Service
- `<operation>` = `Set`, which indicates that this function sets the value of the alarm feature of the Timer System Service, as indicated by the function's parameters (not shown above).

System State Machine

Describes the MPLAB Harmony main function and system-wide state machine.

Description

In its most basic configuration, a MPLAB Harmony system operates in a single polled *super* loop that is implemented in the project's *main* function (in the `<project-name>/firmware/src/main.c` file) that is generated by the MHC, as shown in the following example. The *main* function calls two system-wide state machine functions, SYS_Initialize and SYS_Tasks that are also generated by the MHC to initialize and run the system.

MPLAB Harmony "main" Function

```
MAIN_RETURN main ( void )
{
    SYS_Initialize(NULL);

    while(true)
    {
        SYS_Tasks();
    }
```

```
    return(EXIT_FAILURE);
}
```

The SYS_Initialize function calls the initialization functions for all library and application modules that are used in the system to place them in their initial states. Each module's initialization function must prepare the module so it is safe to call its *Tasks* function(s). The SYS_Initialize function's implementation (and necessary support code) is generated by the MHC in the `<project-name>/firmware/src/system_config/<configuration-name>/system_init.c` file.

After initializing all modules, the *main* function contains the system-wide super loop that executes continuously until the system is powered off or reset. Inside the super loop, the *main* function calls the SYS_Tasks function. This function in turn calls the *Tasks* functions for any library or application modules whose state machines operate in a polled manner in the system. Of course, those state machines must first have been initialized by a prior call to their initialization functions by the SYS_Initialize function. The SYS_Task function's implementation is also generated by the MHC, but it is contained in the `<project-name>/firmware/src/system_config/<configuration-name>/system_tasks.c` file.

The polled super loop configuration described above is the most basic configuration of a MPLAB Harmony project. However, a single MPLAB Harmony project can have multiple configurations and different configurations may change modules so that they operate either polled or interrupt driven or in an RTOS thread. However, regardless of the configuration selected, the *main* function does not normally change.

MPLAB Harmony Documentation Volumes

For further information on MPLAB Harmony configurations and execution models, refer to the following MPLAB Harmony documentation.

- Volume I: Getting Started with MPLAB Harmony
- Volume II: MPLAB Harmony Configurator (MHC)
- Volume III: MPLAB Harmony Development

Note: Although the MPLAB Harmony *main* function and system state machine functions (and other system configuration code) are generated as part of a MPLAB Harmony project for your convenience, these files are unique to your project and can be modified or implemented by other means if so desired, or even removed from the project to create a binary library (`.a` file) containing all configured MPLAB Harmony libraries. Refer to the *"MPLAB® X IDE User's Guide"* (DS50002027) and the *"MPLAB® XC32 C/C++ Compiler User's Guide"* (DS50001686) for information on creating library projects.

MPLAB Harmony Module System Interface

Describes the MPLAB Harmony module system interface and provides usage examples.

Description

To support the system-wide state machine (see System State Machine), an MPLAB Harmony module must provide an initialization function and a *Tasks* function. In addition, a MPLAB Harmony module may (optionally) provide deinitialization, reinitialization, and status functions. This set of functions is considered the module's "system interface". The system state machine, system scheduler, or any other system-management code uses a module's system interface to initialize, run, and otherwise control the execution of a module in the MPLAB Harmony system.

To define the calling signature of these system interface functions, the `sys_module.h` header defines function pointer data types for each. These data types could be used to develop a dynamic system with capabilities beyond the basic system state machine, such as dynamic task registration, power management, advanced schedulers, or even your own operating system.

The following examples show how the system interface could be used to create a simple dynamic polled system tasks scheduler.

Example Dynamic System Data Structures

```
typedef struct _system_module_interface
{
    SYS_MODULE_INITIALIZE_ROUTINE   initialize;
    SYS_MODULE_TASKS_ROUTINE        tasks;
    SYS_MODULE_REINITIALIZE_ROUTINE reinitialize;
    SYS_MODULE_DEINITIALIZE_ROUTINE deinitialize;
    SYS_MODULE_STATUS_ROUTINE       status;

} SYSTEM_MODULE_INTERFACE;


typedef struct _system_module_data
{
    SYSTEM_MODULE_INTERFACE function;
    SYS_MODULE_INDEX        index;
    SYS_MODULE_INIT        *initData;
    SYS_MODULE_OBJ          obj;
    SYS_STATUS              status;
    uint8_t                 powerState;

} SYSTEM_MODULE_DATA;
```

```
SYSTEM_MODULE_DATA gModules[CONFIG_NUMBER_OF_MODULES];
```

In the previous example code, the SYSTEM_MODULE_INTERFACE structure contains pointers to all of a module's system interface functions. This structure could be filled in with pointers to a module's system interface functions and a pointer to the structure passed into a dynamic module registration function, along with the module index number and a pointer to any initialization data required by the module. The following example shows how this dynamic registration function might appear.

Example Dynamic Module Registration Function

```
bool SYS_ModuleRegister ( SYSTEM_MODULE_DATA *module,
                          SYS_MODULE_INDEX    index,
                          SYS_MODULE_INIT    *iniData )
{
    SYSTEM_MODULE_DATA  module;
    int                 i;
    bool                success = false;

    for (i=0; i < CONFIG_NUMBER_OF_MODULES; i++)
    {
        if (gModules[i].function.initialize != NULL)
        {
            module = &gModules[i];
        }
    }

    if (i < CONFIG_NUMBER_OF_MODULES)
    {
        module->function.initialize     = module->initialize;
        module->function.tasks          = module->tasks;
        module->function.reinitialize   = module->reinitialize;
        module->function.deinitialize   = module->deinitialize;
        module->function.status         = module->status;

        module->index       = index;
        module->initData    = initData;
        module->obj         = SYS_MODULE_OBJ_INVALID;
        module->status      = SYS_STATUS_UNINITIALIZED;
        module->powerState  = SYS_MODULE_POWER_RUN_FULL;

        success = true;
    }

    return success;
}
```

The SYS_ModuleRegister function could then scan a system-global array (`gModules`, from the previous system data structures example) to find an empty SYSTEM_MODULE_DATA structure (using the initialization function pointer as a "flag" to indicate if the structure is in use or not), copy the newly registered module's interface and other data into the structure, and initialize the other members of the structure.

Once all modules to be used have been similarly registered, the entire system could be initialized when desired by calling a SYS_InitializeDynamic function implementation, similar to the following example.

Example Dynamic System Initialization Function

```
void SYS_InitializeDynamic ( void *data )
{
    SYSTEM_MODULE_DATA  module = (SYSTEM_MODULE_DATA *)data;
    int                 i;

    for (i=0; i < CONFIG_NUMBER_OF_MODULES; i++)
    {
        if (module->function.initialize != NULL)
        {
            module->obj = module->function.initialize(module->index,
                                                      module->initData);

            module->status     = SYS_STATUS_BUSY;
            module->powerState = SYS_MODULE_POWER_RUN_FULL;

            if (module->obj == SYS_MODULE_OBJ_INVALID)
            {
                module->function.initialize = NULL;
            }
```

```
        }
    }
}
```

The previous SYS_InitializeDynamic example function iterates through the global array of module data structures, calling the initialization functions for any modules that have been registered and skipping over any structures that have no module registered (again using the required initialization function pointer as a sort of flag). If a module's initialization function successfully initializes the module it returns a valid module object handle, which the dynamic system initialization function captures in the module's data structure so it can call the module's other system interface routines. If the module is not successfully initialized, the object handle reported will be invalid (SYS_MODULE_OBJECT_INVALID) and the module is deregistered by nulling the pointer to the initialization function.

Once the system has been initialized, a dynamic tasks function like the following example would also iterate through the module data array and call the tasks functions for each module registered.

Example Dynamic System Tasks Function

```
void SYS_TasksDynamic ( void *data )
{
    SYSTEM_MODULE_DATA  module = (SYSTEM_MODULE_DATA *)data;
    int                 i;

    for (i=0; i < CONFIG_NUMBER_OF_MODULES; i++)
    {
        if (module->function.initialize != NULL &&
            module->function.tasks      != NULL )
        {
            if (module->status(module->obj) >= SYS_STATUS_UNINITIALIZED)
            {
                module->function.tasks(module->obj);
            }
            else
            {
                module->function.deinitialize(module->obj);
                module->function.initialize = NULL;
            }
        }
    }
}
```

After calling a module's *Tasks* function, the previous example checks the module's status by calling its *Status* function. If the module reports an error status (any status less than SYS_STATUS_UNINITIALIZED), the module is deinitialized by calling its *Deinitialize* function and deregistered by nulling out the *Initialize* function pointer.

Please note that the example code provided is for informational purposes only, and is used to describe the purpose and usage of the MPLAB Harmony module system interface. A real dynamic tasks scheduler would need to deal with additional complexities. For example, MPLAB Harmony modules may have zero or more *Tasks* functions (only the initialization function is absolutely required). Therefore, a single tasks-function pointer would not be sufficient. The previous example demonstrates usage of the system status and power state data, but it does not actually update or manage these items. Also, this example does not demonstrate the usage of a module's reinitialization function. This function allows a module to provide a way to change its initial parameters while the module is active (after it has been initialized) without disrupting active clients of the module. *However, at the time of this writing, most MPLAB Harmony modules do not implement this function, so this capability is not usually available.*

Using the SYS_ASSERT Macro

Describes the purpose and usage of the system assertion macro.

Description

The SYS_ASSERT macro is a testing and debugging tool used throughout MPLAB Harmony libraries to verify (or assert) critical assumptions before any action is taken on them. However, it is not usually desirable to have these tests in production code because system assertion failures are normally fatal events that stop current execution, either hanging or resetting the system because it is not safe to continue. Also, even if the tests pass, they would add significant overhead, affecting code size and execution time. To avoid these issues, the default implementation of this macro is empty to eliminate the assertion tests and failure messages from the build, as shown in the following example.

Default SYS_ASSERT Macro Definition

```
#define SYS_ASSERT(test,message)
```

Conversely, when developing, debugging, and testing a project, having SYS_ASSERT statements in the code (particularly in library code) can be very helpful. This macro is used to check key parameter values and intermediate results and provide messages that explain the consequences that occur when they don't match what was expected when the library was implemented. A SYS_ASSERT failure during testing may save significant time spent debugging a MPLAB Harmony library only to find that a configuration setting or value passed into a library was incorrect or unexpected.

The SYS_ASSERT macro provides a convenient way to obtain the desired behavior when an assertion fails. In production code, simply accept the default definition that disposes of the assertion test code and the failure message. If things go wrong during testing, either enable a predefined

implementation of this macro or define it in any way that is convenient for the current application.

One particularly useful definition of the SYS_ASSERT macro is to execute a hard-coded breakpoint instruction if the assertion fails, as shown in the following example.

Example SYS_ASSERT Macro Breakpoint Definition

```
#define SYS_ASSERT(test, message) \
            do{ if(!(test)) SYS_DEBUG_Breakpoint(); }while(true)
```

When using the previous definition, if the assertion test fails, the processor will reach a hard-coded breakpoint and execution will stop in the debugger at the SYS_ASSERT call. This behavior identifies the point in the source code where the assertion call failed. Then, the assertion call conveniently provides an explanation of what the failure means in the associated message string. A system assertion failure is a catastrophic failure, so the `do... while` loop hangs the system to prevent execution of code under an invalid assumption that may cause incorrect behavior or even damage to the system. It is also a way to guarantee that the `if` statement is correctly interpreted by the compiler and not accidentally associated with a following `else` statement.

Another useful implementation of the SYS_ASSERT macro might use the Debug System Service to display the message, as shown in the following example.

Example SYS_ASSERT Macro Definition Using Debug Message Service

```
#define SYS_ASSERT(test, message) \
            do{ if(!(test)) \
                    SYS_DEBUG_MESSAGE(SYS_ERROR_FATAL,(message)); \
            }while(false)
```

The previous definition will display the assertion failure message using whatever method the SYS_DEBUG service is configured to use. However, it has the drawback that it does not hang the system (because the message service may need to continue running), so incorrect or unsafe behavior may result.

It is also possible to combine these two example definitions, which is particularly useful if the debug message service is configured to use the debugger's output window, or to create any another definition that is more useful for a given situation.

Obtaining System Version Information

Describes the purpose and usage of the system version functions.

Description

It is possible to programmatically obtain the version information for the release of MPLAB Harmony using the SYS_Version group of functions. For example, once the Debug System Service has been initialized and is running, the following code would retrieve the current version number and display it on the configured debug console if it were lower than a given value.

Example Version Function Usage

```
#define REQUIRED_VERSION 10700

if (SYS_VersionGet() < REQUIRED_VERSION)
{
    SYS_DEBUG_MESSAGE( SYS_ERROR_WARNING, "Version %s is too low",
                       SYS_VersionStrGet() );
}
```

In the previous example, the SYS_VersionGet function is used to get the numeric representation of the version number so that it can be easily compared to a known value and the SYS_VersionStrGet is used to get a string representation of it for displaying on the debug console.

Library Interface

Data Types and Constants

	Name	Description
	MAIN_RETURN_CODES	Defines return codes for "main".
	SYS_MODULE_DEINITIALIZE_ROUTINE	Pointer to a routine that deinitializes a system module (driver, library, or system-maintained application).
	SYS_MODULE_INDEX	Identifies which instance of a system module should be initialized or opened.
	SYS_MODULE_INIT	Initializes a module (including device drivers) in a current power status as requested by the system or power manager.
	SYS_MODULE_INITIALIZE_ROUTINE	Pointer to a routine that initializes a system module (driver, library, or system-maintained application).
	SYS_MODULE_OBJ	Handle to an instance of a system module.
	SYS_MODULE_REINITIALIZE_ROUTINE	Pointer to a routine that reinitializes a system module (driver, library, or system-maintained application)
	SYS_MODULE_STATUS_ROUTINE	Pointer to a routine that gets the current status of a system module (driver, library, or system-maintained application).

	SYS_MODULE_TASKS_ROUTINE	Pointer to a routine that performs the tasks necessary to maintain a state machine in a module system module (driver, library, or system-maintained application).
	SYS_STATUS	Identifies the current status/state of a system module (including device drivers).
	MAIN_RETURN_CODE	Casts the given value to the correct type for the return code from "main".
	SYS_MODULE_OBJ_INVALID	Object handle value returned if unable to initialize the requested instance of a system module.
	SYS_MODULE_OBJ_STATIC	Object handle value returned by static modules.
	SYS_MODULE_POWER_IDLE_RUN	Module power-state idle-run state code.
	SYS_MODULE_POWER_IDLE_STOP	Module power-state idle-stop state code.
	SYS_MODULE_POWER_OFF	Module power-state power off state code.
	SYS_MODULE_POWER_RUN_FULL	Module power-state run-full state code.
	SYS_MODULE_POWER_SLEEP	Module power-state sleep state code.

Main Function Support Macro

	Name	Description
	MAIN_RETURN	Defines the correct return type for the "main" routine.

System Assert Macro

	Name	Description
	SYS_ASSERT	Implements default system assert routine, asserts that "test" is true.

System State Machine Functions

	Name	Description
	SYS_Tasks	Function that performs all polled system tasks.
	SYS_Initialize	Initializes the board, services, drivers, and other modules

Version Functions

	Name	Description
	SYS_VersionGet	Gets SYS_COMMON version in numerical format.
	SYS_VersionStrGet	Gets SYS_COMMON version in string format.

Description

This section describes the APIs of the System Service Library.

Refer to each section for a detailed description.

Main Function Support Macro

MAIN_RETURN Macro

Defines the correct return type for the "main" routine.

File

sys_common.h

C

```
#define MAIN_RETURN void
```

Description

Main Function Return Type

This macro defines the correct return type for the "main" routine for the selected Microchip microcontroller family.

Remarks

The main function return type may change, depending upon which family of Microchip microcontrollers is chosen. Refer to the user documentation for the C-language compiler in use for more information.

Example

```
MAIN_RETURN main ( void )
{
```

```
    // Initialize the system
    SYS_Initialize(...);

    // Main Loop
    while(true)
    {
        SYS_Tasks();
    }

    return MAIN_RETURN_CODE(MAIN_RETURN_SUCCESS);
}
```

System Assert Macro

SYS_ASSERT Macro

Implements default system assert routine, asserts that "test" is true.

File

sys_common.h

C

```
#define SYS_ASSERT(test,message)
```

Returns

None. Normally hangs in a loop, depending on the implementation to which it is mapped.

Description

This macro implements the default system assert routine that asserts that the provided boolean test is true.

Remarks

Can be overridden as desired by defining your own SYS_ASSERT macro before including system.h.

The default definition removes this macro from all code because it adds significant size to all projects. The most efficient use is to enable it in individual libraries and build them separately for debugging.

Preconditions

None, depending on the implementation to which this macro is mapped.

Example

```
void MyFunc ( int *pointer )
{
    SYS_ASSERT(NULL != pointer, "NULL Pointer passed to MyFunc");

    Do something with pointer.
}
```

Parameters

Parameters	Description
test	This is an expression that resolves to a boolean value (zero=false, non-zero=true)
message	This is a NULL-terminated ASCII character string that can be displayed on a debug output terminal if "test" is false (if supported).

Function

void SYS_ASSERT (bool test, char *message)

System State Machine Functions

SYS_Tasks Function

Function that performs all polled system tasks.

File

sys_module.h

C

```
void SYS_Tasks();
```

Returns

None.

Description

System Tasks Function

This function performs all polled system tasks by calling the state machine "tasks" functions for all polled modules in the system, including drivers, services, middleware and applications.

Remarks

If the module is interrupt driven, the system will call this routine from an interrupt context.

Preconditions

The SYS_Initialize function must have been called and completed.

Example

```
SYS_Initialize ( NULL );

while ( true )
{
    SYS_Tasks ( );
}
```

Function

void SYS_Tasks (void);

SYS_Initialize Function

Initializes the board, services, drivers, and other modules

File

sys_init.h

C

```
void SYS_Initialize(void * data);
```

Returns

None.

Description

This function initializes the board, services, drivers, and other modules as configured at build time. It should be called almost immediately after entering the "main" function.

Remarks

Basic System Initialization Sequence:

1. Initialize core processor services.
2. Initialize board support package.
3. Initialize RTOS (if utilized).
4. Initialize drivers.
5. Initialize system services.
6. Initialize middleware.
7. Initialize application(s).

This function may be overridden and implemented as desired as long as it appropriately initializes all necessary board, services, and modules.

Most MPLAB Harmony libraries are designed so that the order in which they are initialized is not important. However, core processor services and board support packgage initialization should be completed before any other initialization takes place and RTOS initialization (if utilized) shoudl take palce before drivers, system services and middleware are initialized. Applications should be initialized last.

Preconditions

The C-language run-time environment and stack must have been initialized.

Example

```
int main ( void )
{
    SYS_Initialize(NULL);

    while (true)
    {
        SYS_Tasks();
    }
}
```

Parameters

Parameters	Description
data	Pointer to any system initialization data required. Normally passed as NULL for static system implementations.

Function

void SYS_Initialize (void *data)

Version Functions

SYS_VersionGet Macro

Gets SYS_COMMON version in numerical format.

File

sys_common.h

C

```
#define SYS_VersionGet( void ) SYS_VERSION
```

Returns

Current driver version in numerical format.

Description

This routine gets the SYS_COMMON version. The version is encoded as major * 10000 + minor * 100 + patch. The string version can be obtained using SYS_VersionStrGet()

Function

SYS_VersionGet(void)

SYS_VersionStrGet Macro

Gets SYS_COMMON version in string format.

File

sys_common.h

C

```
#define SYS_VersionStrGet( void ) SYS_VERSION_STR
```

Returns

Current SYS_COMMON version in the string format.

Description

Macro: char * SYS_VersionStrGet (void)

This routine gets the SYS_COMMON version in string format. The version is returned as major.minor.path[type], where type is optional. The

numerical version can be obtained using SYS_VersionGet()

Remarks

None.

Data Types and Constants

MAIN_RETURN_CODES Enumeration

Defines return codes for "main".

File

sys_common.h

C

```
typedef enum {
  MAIN_RETURN_FAILURE = -1,
  MAIN_RETURN_SUCCESS = 0
} MAIN_RETURN_CODES;
```

Description

Main Routine Codes Enumeration

This enumeration provides a predefined list of return codes for the main function. These codes can be passed into the MAIN_RETURN_CODE macro to convert them to the appropriate type (or discard them if not needed) for the Microchip C-language compiler in use.

Remarks

The main function return type may change, depending upon which family of Microchip microcontrollers is chosen. Refer to the user documentation for the C-language compiler in use for more information.

Example

```
MAIN_RETURN main ( void )
{
    SYS_Initialize(...);

    while(true)
    {
        SYS_Tasks();
    }

    return MAIN_RETURN_CODE(MAIN_RETURN_SUCCESS);
}
```

SYS_MODULE_DEINITIALIZE_ROUTINE Type

Pointer to a routine that deinitializes a system module (driver, library, or system-maintained application).

File

sys_module.h

C

```
typedef void (* SYS_MODULE_DEINITIALIZE_ROUTINE)(SYS_MODULE_OBJ object);
```

Returns

None.

Description

System Module Deinitialization Routine Pointer. This data type is a pointer to a routine that deinitializes a system module (driver, library, or system-maintained application).

Remarks

If the module instance has to be used again, the module's "initialize" function must first be called.

Preconditions

The low-level board initialization must have (and will be) completed and the module's initialization routine will have been called before the system will call the deinitialization routine for any modules.

Example

None.

Parameters

Parameters	Description
object	Handle to the module instance

Function

void (* SYS_MODULE_DEINITIALIZE_ROUTINE) (SYS_MODULE_OBJ object)

SYS_MODULE_INDEX Type

Identifies which instance of a system module should be initialized or opened.

File

sys_module.h

C

```
typedef unsigned short int SYS_MODULE_INDEX;
```

Description

System Module Index

This data type identifies to which instance of a system module a call to that module's "Initialize" and "Open" routines refers.

Remarks

Each individual module will usually define macro names for the index values it supports (e.g., DRV_TMR_INDEX_1, DRV_TMR_INDEX_2, ...).

SYS_MODULE_INIT Union

Initializes a module (including device drivers) in a current power status as requested by the system or power manager.

File

sys_module.h

C

```
typedef union {
  uint8_t value;
  struct {
    uint8_t powerState : 4;
    uint8_t reserved : 4;
  } sys;
} SYS_MODULE_INIT;
```

Members

Members	Description
uint8_t powerState : 4;	Requested power state
uint8_t reserved : 4;	Module-definable field, module-specific usage

Description

System Module Init

This structure provides the necessary data to initialize or reinitialize a module (including device drivers) into a requested power state. The structure can be extended in a module specific way as to carry module specific initialization data.

Remarks

This structure is used in the device driver routines DRV__Initialize and DRV__Reinitialize that are defined by each device driver.

The "powerState" member has several predefined values (shown below). All other available values (within the 4-bit field) are available for

module-specific meaning.

Predefined powerState Values:

- 0. SYS_MODULE_POWER_OFF - Module power-state power off state code
- 1. SYS_MODULE_POWER_SLEEP - Module power-state sleep state code
- 2. SYS_MODULE_POWER_IDLE_STOP - Module power-state idle-stop state code
- 3. SYS_MODULE_POWER_IDLE_RUN - Module power-state idle-run state code
- 4. through 14. - Module-specific meaning
- 15. SYS_MODULE_POWER_RUN_FULL - Module power-state run-full state code

SYS_MODULE_INITIALIZE_ROUTINE Type

Pointer to a routine that initializes a system module (driver, library, or system-maintained application).

File

sys_module.h

C

```
typedef SYS_MODULE_OBJ (* SYS_MODULE_INITIALIZE_ROUTINE)(const SYS_MODULE_INDEX index, const
SYS_MODULE_INIT * const init);
```

Returns

A handle to the instance of the system module that was initialized. This handle is a necessary parameter to all of the other system-module level routines for that module.

Description

System Module Initialization Routine Pointer

This data type is a pointer to a routine that initializes a system module (driver, library, or system-maintained application).

Remarks

This function will only be called once during system initialization.

Preconditions

The low-level board initialization must have (and will be) completed before the system will call the initialization routine for any modules.

Parameters

Parameters	Description
index	Identifier for the module instance to be initialized
init	Pointer to the data structure containing any data necessary to initialize the module. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ (* SYS_MODULE_INITIALIZE_ROUTINE) (
const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init)

SYS_MODULE_OBJ Type

Handle to an instance of a system module.

File

sys_module.h

C

```
typedef uintptr_t SYS_MODULE_OBJ;
```

Description

System Module Object

This data type is a handle to a specific instance of a system module (such as a device driver).

Remarks

Code outside of a specific module should consider this as an opaque type (much like a void *). Do not make any assumptions about base type as

it may change in the future or about the value stored in a variable of this type.

SYS_MODULE_REINITIALIZE_ROUTINE Type

Pointer to a routine that reinitializes a system module (driver, library, or system-maintained application)

File

sys_module.h

C

```
typedef void (* SYS_MODULE_REINITIALIZE_ROUTINE)(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

System Module Reinitialization Routine Pointer

This data type is a pointer to a routine that reinitializes a system module (driver, library, or system-maintained application).

Remarks

This operation uses the same initialization data structure as the Initialize operation.

This operation can be used to change the power state of a module.

This operation can also be used to refresh the hardware state as defined by the initialization data, thus it must guarantee that all hardware state has been refreshed.

This function can be called multiple times to reinitialize the module.

Preconditions

The low-level board initialization must have (and will be) completed and the module's initialization routine will have been called before the system will call the reinitialization routine for any modules.

Example

None.

Parameters

Parameters	Description
object	Handle to the module instance
init	Pointer to the data structure containing any data necessary to initialize the module. This pointer may be null if no data is required and default initialization is to be used.

Function

void (* SYS_MODULE_REINITIALIZE_ROUTINE) (SYS_MODULE_OBJ object,
const SYS_MODULE_INIT * const init)

SYS_MODULE_STATUS_ROUTINE Type

Pointer to a routine that gets the current status of a system module (driver, library, or system-maintained application).

File

sys_module.h

C

```
typedef SYS_STATUS (* SYS_MODULE_STATUS_ROUTINE)(SYS_MODULE_OBJ object);
```

Returns

One of the possible status codes from SYS_STATUS

Description

System Module Status Routine Pointer

This data type is a pointer to a routine that gets the current status of a system module (driver, library, or system-maintained application).

Remarks

A module's status operation can be used to determine when any of the other module level operations has completed as well as to obtain general status of the module. The value returned by the status routine will be checked after calling any of the module operations to find out when they have completed.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). A module may define module-specific error values of less or equal SYS_STATUS_ERROR_EXTENDED (-10).

The status function must NEVER block.

If the status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The low-level board initialization must have (and will be) completed and the module's initialization routine will have been called before the system will call the status routine for any modules.

Example

None.

Parameters

Parameters	Description
object	Handle to the module instance

Function

SYS_STATUS (* SYS_MODULE_STATUS_ROUTINE) (SYS_MODULE_OBJ object)

SYS_MODULE_TASKS_ROUTINE Type

Pointer to a routine that performs the tasks necessary to maintain a state machine in a module system module (driver, library, or system-maintained application).

File

sys_module.h

C

```
typedef void (* SYS_MODULE_TASKS_ROUTINE)(SYS_MODULE_OBJ object);
```

Returns

None.

Description

System Module Tasks Routine Pointer

This data type is a pointer to a routine that performs the tasks necessary to maintain a state machine in a module system module (driver, library, or system-maintained application).

Remarks

If the module is interrupt driven, the system will call this routine from an interrupt context.

Preconditions

The low-level board initialization must have (and will be) completed and the module's initialization routine will have been called before the system will call the deinitialization routine for any modules.

Example

None.

Parameters

Parameters	Description
object	Handle to the module instance

Function

void (* SYS_MODULE_TASKS_ROUTINE) (SYS_MODULE_OBJ object)

SYS_STATUS Enumeration

Identifies the current status/state of a system module (including device drivers).

File

sys_module.h

C

```
typedef enum {
  SYS_STATUS_ERROR_EXTENDED = -10,
  SYS_STATUS_ERROR = -1,
  SYS_STATUS_UNINITIALIZED = 0,
  SYS_STATUS_BUSY = 1,
  SYS_STATUS_READY = 2,
  SYS_STATUS_READY_EXTENDED = 10
} SYS_STATUS;
```

Members

Members	Description
SYS_STATUS_ERROR_EXTENDED = -10	Indicates that a non-system defined error has occurred. The caller must call the extended status routine for the module in question to identify the error.
SYS_STATUS_ERROR = -1	An unspecified error has occurred.
SYS_STATUS_UNINITIALIZED = 0	The module has not yet been initialized
SYS_STATUS_BUSY = 1	An operation is currently in progress
SYS_STATUS_READY = 2	Any previous operations have succeeded and the module is ready for additional operations
SYS_STATUS_READY_EXTENDED = 10	Indicates that the module is in a non-system defined ready/run state. The caller must call the extended status routine for the module in question to identify the state.

Description

System Module Status

This enumeration identifies the current status/state of a system module (including device drivers).

Remarks

This enumeration is the return type for the system-level status routine defined by each device driver or system module (for example, DRV_I2C_Status).

MAIN_RETURN_CODE Macro

Casts the given value to the correct type for the return code from "main".

File

sys_common.h

C

```
#define MAIN_RETURN_CODE(c)
```

Description

Main Routine Code Macro

This macro cases the given value to the correct type for the return code from the main function.

Remarks

The main function return type may change, depending upon which family of Microchip microcontrollers is chosen. Refer to the user documentation for the C-language compiler in use for more information.

Example

```
MAIN_RETURN main ( void )
{
    // Initialize the system
    SYS_Initialize(...);

    // Main Loop
    while(true)
```

```
    {
        SYS_Tasks();
    }

    return MAIN_RETURN_CODE(MAIN_RETURN_SUCCESS);
}
```

SYS_MODULE_OBJ_INVALID Macro

Object handle value returned if unable to initialize the requested instance of a system module.

File

sys_module.h

C

```
#define SYS_MODULE_OBJ_INVALID ((SYS_MODULE_OBJ) -1 )
```

Description

System Module Object Invalid

This is the object handle value returned if unable to initialize the requested instance of a system module.

Remarks

Do not rely on the actual value of this constant. It may change in future implementations.

SYS_MODULE_OBJ_STATIC Macro

Object handle value returned by static modules.

File

sys_module.h

C

```
#define SYS_MODULE_OBJ_STATIC ((SYS_MODULE_OBJ) 0 )
```

Description

System Module Object Static

This is the object handle value returned by static system modules.

Remarks

Do not rely on the actual value of this constant. It may change in future implementations.

SYS_MODULE_POWER_IDLE_RUN Macro

Module power-state idle-run state code.

File

sys_module.h

C

```
#define SYS_MODULE_POWER_IDLE_RUN 3
```

Description

System Module Power Idle-Run State

This value identifies the current power status/state of a system module (including device drivers). It is used to indicate that the module should prepare to enter an idle-run state.

Remarks

An idle-run state indicates that the core CPU clock may be stopped, but the module's peripheral clock may continue running and peripheral operations may continue as long as no code needs to be executed. If code needs to execute, the module must cause an interrupt.

This value is passed in the powerState field of the SYS_MODULE_INIT structure that takes part in all modules initialization and reinitialization.

The power state codes between SYS_MODULE_POWER_IDLE_RUN (with a value of 3) and SYS_MODULE_POWER_RUN_FULL (with a value of 15) are available for module-specific definition and usage.

SYS_MODULE_POWER_IDLE_STOP Macro

Module power-state idle-stop state code.

File

sys_module.h

C

```
#define SYS_MODULE_POWER_IDLE_STOP 2
```

Description

System Module Power Idle-Stop State

This value identifies the current power status/state of a system module (including device drivers). It is used to indicate that the module should prepare to enter an idle-stop state.

Remarks

An idle-stop state indicates that the core CPU clock may be stopped, but the module's peripheral clock may continue running. However, the peripheral should prepare to stop operations when the idle state is entered.

This value is passed in the powerState field of the SYS_MODULE_INIT structure that takes part in all modules initialization and reinitialization.

The power state codes between SYS_MODULE_POWER_IDLE_RUN (with a value of 3) and SYS_MODULE_POWER_RUN_FULL (with a value of 15) are available for module-specific definition and usage.

SYS_MODULE_POWER_OFF Macro

Module power-state power off state code.

File

sys_module.h

C

```
#define SYS_MODULE_POWER_OFF 0
```

Description

System Module Power Off State

This value identifies the current power status/state of a system module (including device drivers). It is used to indicate that the module should prepare to enter a full power-off state.

Remarks

A power off state indicates that power may be completely removed (0 Volts).

This value is passed in the powerState field of the SYS_MODULE_INIT structure that takes part in all modules initialization and reinitialization.

The power state codes between SYS_MODULE_POWER_IDLE_RUN (with a value of 3) and SYS_MODULE_POWER_RUN_FULL (with a value of 15) are available for module-specific definition and usage.

SYS_MODULE_POWER_RUN_FULL Macro

Module power-state run-full state code.

File

sys_module.h

C

```
#define SYS_MODULE_POWER_RUN_FULL 15
```

Description

System Module Power Run-Full State. This value identifies the current power status/state of a system module (including device drivers). It is used to indicate that the module should prepare to enter an run-full state.

Remarks

An run-full state indicates that the core CPU and peripheral clocks are operational at their normal configured speed and the module should be ready for normal operation.

This value is passed in the powerState field of the SYS_MODULE_INIT structure that takes part in all modules initialization and reinitialization.

The power state codes between SYS_MODULE_POWER_IDLE_RUN (with a value of 3) and SYS_MODULE_POWER_RUN_FULL (with a value of 15) are available for module-specific definition and usage.

SYS_MODULE_POWER_SLEEP Macro

Module power-state sleep state code.

File

sys_module.h

C

```
#define SYS_MODULE_POWER_SLEEP 1
```

Description

System Module Power Sleep State

This value identifies the current power status/state of a system module (including device drivers). It is used to indicate that the module should prepare to enter a sleep state.

Remarks

A Sleep state indicates that the core CPU and peripheral clocks may be stopped and no code will execute and any module hardware will be stopped.

This value is passed in the powerState field of the SYS_MODULE_INIT structure that takes part in all modules initialization and reinitialization.

The power state codes between SYS_MODULE_POWER_IDLE_RUN (with a value of 3) and SYS_MODULE_POWER_RUN_FULL (with a value of 15) are available for module-specific definition and usage.

Files

Files

Name	Description
sys_common.h	Common System Services definitions and declarations.
sys_module.h	Defines definitions and declarations related to system modules.
system.h	Top level common system services library interface header.
sys_init.h	Initialization System Service Library definitions and declarations.

Description

This section lists the source and header files used by the library.

sys_common.h

Common System Services definitions and declarations.

Enumerations

	Name	Description
	MAIN_RETURN_CODES	Defines return codes for "main".

Macros

	Name	Description
	MAIN_RETURN	Defines the correct return type for the "main" routine.
	MAIN_RETURN_CODE	Casts the given value to the correct type for the return code from "main".
	SYS_ASSERT	Implements default system assert routine, asserts that "test" is true.
	SYS_VersionGet	Gets SYS_COMMON version in numerical format.
	SYS_VersionStrGet	Gets SYS_COMMON version in string format.

Description

System Services Common Library Header

This file provides commonsystem services definitions and declarations.

Remarks

This file is included by "system.h" and does not need to be included directly.

File Name

sys_common.h

Company

Microchip Technology Inc.

sys_module.h

Defines definitions and declarations related to system modules.

Enumerations

	Name	Description
	SYS_STATUS	Identifies the current status/state of a system module (including device drivers).

Functions

	Name	Description
	SYS_Tasks	Function that performs all polled system tasks.

Macros

	Name	Description
	SYS_MODULE_OBJ_INVALID	Object handle value returned if unable to initialize the requested instance of a system module.
	SYS_MODULE_OBJ_STATIC	Object handle value returned by static modules.
	SYS_MODULE_POWER_IDLE_RUN	Module power-state idle-run state code.
	SYS_MODULE_POWER_IDLE_STOP	Module power-state idle-stop state code.
	SYS_MODULE_POWER_OFF	Module power-state power off state code.
	SYS_MODULE_POWER_RUN_FULL	Module power-state run-full state code.
	SYS_MODULE_POWER_SLEEP	Module power-state sleep state code.

Types

	Name	Description
	SYS_MODULE_DEINITIALIZE_ROUTINE	Pointer to a routine that deinitializes a system module (driver, library, or system-maintained application).
	SYS_MODULE_INDEX	Identifies which instance of a system module should be initialized or opened.
	SYS_MODULE_INITIALIZE_ROUTINE	Pointer to a routine that initializes a system module (driver, library, or system-maintained application).
	SYS_MODULE_OBJ	Handle to an instance of a system module.
	SYS_MODULE_REINITIALIZE_ROUTINE	Pointer to a routine that reinitializes a system module (driver, library, or system-maintained application)
	SYS_MODULE_STATUS_ROUTINE	Pointer to a routine that gets the current status of a system module (driver, library, or system-maintained application).
	SYS_MODULE_TASKS_ROUTINE	Pointer to a routine that performs the tasks necessary to maintain a state machine in a module system module (driver, library, or system-maintained application).

Unions

	Name	Description
	SYS_MODULE_INIT	Initializes a module (including device drivers) in a current power status as requested by the system or power manager.

Description

System Module Header

This file defines definitions and interfaces related to system modules.

Remarks

This file is included via "system.h" and does not normally need to be included directly.

File Name

sys_module.h

Company

Microchip Technology Inc.

system.h

Top level common system services library interface header.

Description

System Services Library Interface Header

This file is the top level common system services library interface header. It defines (or includes files that define) the common system service types, prototypes, and other definitions that are commonly used by MPLAB Harmony libraries and system services.

System services provide common functionality that would otherwise need to be duplicated by multiple other modules or that would force them to interact in complex and hard to manage ways. System services eliminate conflicts by controlling access shared resources.

Remarks

The parent directory to the "system" directory should be added to the compiler's search path for header files such that the following include statment will successfully include this file.

#include "system/system.h"

File Name

system.h

Company

Microchip Technology Inc.

sys_init.h

Initialization System Service Library definitions and declarations.

Functions

	Name	Description
	SYS_Initialize	Initializes the board, services, drivers, and other modules

Description

Initialization System Service Library Definitions

This file contains Initialization System Service Library definitions and declarations.

Remarks

This file is included by "system.h" and need not be included directly.

File Name

sys_init.h

Company

Microchip Technology Inc.

Clock System Service Library

This section describes the Clock System Service Library.

Introduction

This library provides an interface to manage the Oscillator module on the Microchip family of microcontrollers during Execution time using the Clock System Service Library.

Description

The Clock System Service can be configured to run in either Static mode or Dynamic mode via the MPLAB Harmony Configurator. In Static mode,

the service will configure the Oscillator module to the user-desired system clock frequency, peripheral bus clock frequency, reference bus clock frequency, and USB PLL clock frequency. These settings are set at initialization and are designed to remain the same for the duration of run-time. In Dynamic mode, the service will also set all of these frequencies at initialization; however it offers the function support to allow users to change the system clock frequency, peripheral bus frequency, and if the device supports it, the reference clock frequency.

Design Recommendations

If performance and small code size are critical, and there is no requirement to change the clock speed, the recommendation is to use Static mode. If there is a need to change the system, peripheral, or reference clock frequencies during run-time, the recommendation is to use Dynamic mode.

Configuration

The Oscillator is the heart of the microcontroller, which provides the clock on which the core and the peripherals run. For all of the oscillators present on a Microchip microcontroller, two types of configurations exist:

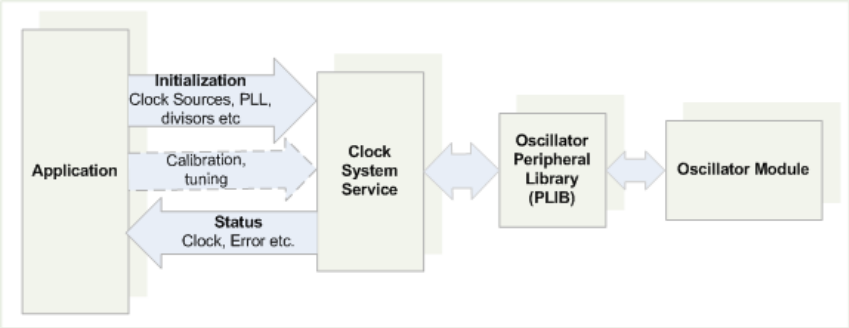
- Through Configuration bits
- At Execution time

Configuring the oscillator through the Configuration bits is a one-time process that it is done during the programming of the device. This one-time configuration is programmed in the code memory. The Execution time configuration provides features that are allowed to be changed during code execution. The Clock System Service Library provides functions the Execution time configurable features of the Oscillator module.

The Oscillator module supports the CPU and other peripherals by providing the clock. Therefore, there is more initialization and little or no run-time operations. The run-time operations set the system clock or any other output clocks. If the application needs to make any initialization changes, they can be done at run-time.

The Oscillator module as a whole is a group of oscillators. These are provided for the application to choose the correct oscillator for the application. It is important to wisely choose the oscillator and the frequency at which the CPU and the other peripherals should run in power-critical environments.

The following figure illustrates the interaction of the Clock System Service Library.



Using the Library

This topic describes the basic architecture of the Clock System Service and provides information and examples on its use.

Description

Interface Header File: `sys_osc.h`

The interface to the Clock System Service is defined in the `sys_osc.h` header file, which is included by the `sys.h` header file.

Any C language source (`.c`) file that uses the Clock System Service must include `sys.h`.

Abstraction Model

This library provides a low-level abstraction of the Clock System Service Library. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

To understand the Oscillator module and how each of its feature is mapped in this library, it is important to understand the following terminologies.

Oscillators/Clock Sources

A clock source is hardware that generates oscillations. This may be internal or external.

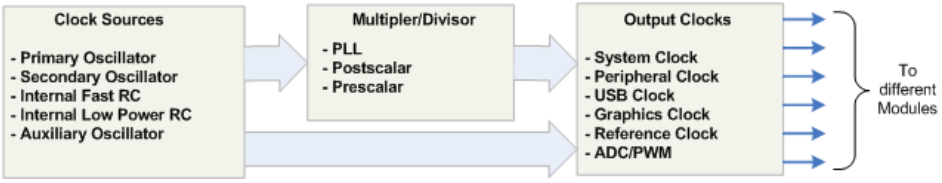
Divisor and Multiplier/PLL

These are hardware modules that can scale the clock. The rate at which the scaling is done may be fixed or configurable.

Clocks

Clock outputs are output lines from the Oscillator module, which may route to different modules of the device or to the CPU (the system clock).

The following diagram provides a simplified explanation and the relationship between the previously mentioned terms. In most cases, there are multiple clock source options available for each of the clock outputs. However, not all clock sources are available for all output clocks. Scaling is an optional feature in most cases.



Library Overview

Please refer to the System Service Introduction for a detailed description of MPLAB Harmony system services.

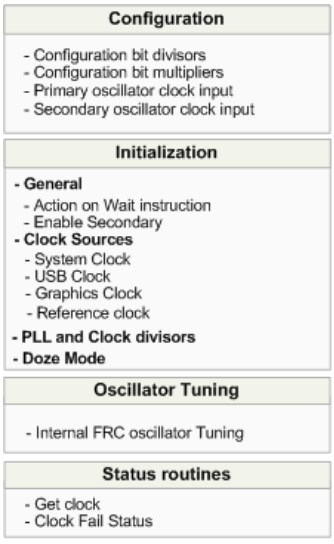
The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Clock System Service module.

Library Interface Section	Description
System Clock Functions	Provides functions to set and get the system clock frequency.
Peripheral Clock Functions	Provides functions to set and get the peripheral clock frequency.
Reference Clock Functions	Provides functions to set up the reference clock, and set and get the reference clock frequency.
Clock Failure Functions	Provides functions to allow registration of a callback function that will be triggered on a clock failure and to inform the user on a clock failure by invoking the registered call back function.
Initialization Functions	Provides an initialization routine to configure the module at the start-up of the system and reinitialization if the application wants to change any of the initialization settings.

How the Library Works

The following diagram shows the flow of an application call to the Clock System Service Library.

Refer to the System Services Introduction section for information on how the system services operate in a system.



Initialization

Provides information and code examples for initializing the Oscillator.

Description

The Oscillator is a module with little or no run-time operations. Therefore, the application must do most of the configurations at the time of system initialization even though it is allowed to change the clock frequency at run-time.

The Oscillator needs some time to settle. Therefore, it is recommended to perform the oscillator initialization before initializing other modules. Check for the oscillator status once all of the initializations are complete and before the application task routine takes control.

Note: Not all devices require all of the initialization parameters of the structure. Refer to the specific device data sheet to determine which features are available for your device.

Clock is Set Through Configuration bits

In most cases, setting the clock is done through Configuration bits. Therefore, what the application expects from the Clock System Service Library is to provide all of the clock frequency values on a function call. To use the Clock System Service in this way, call the SYS_CLK_Initialize function by passing a NULL to it, as shown in the following code example. The function reads the registers and will initialize its internal data structure.

```
SYS_CLK_Initialize ( NULL );
//As per my setting, the system clock should be 80 MHz
If ( SYS_CLK_SystemFrequencyGet ( )  != 80000000 )
{
     //Something is wrong, check the Configuration bit settings
}
```

Alter the Configuration bit Setting

To alter the settings made in the Configuration bit, pass the structure to the SYS_CLK_Initialize function.

```
SYS_CLK_INIT generalInit;

// Populate the oscillator initialization structure
generalInit.systemClockSource = SYS_CLK_SOURCE_FRC_SYSPLL;
generalInit.systemClockFrequencyHz = 30000000;
generalInit.onWaitInstruction = OSC_ON_WAIT_SLEEP;
generalInit.secondaryOscKeepEnabled = true;
SYS_CLK_Initialize (&generalInit);
```

Changing the Clock

Provides information on changing the clock.

Description

The module initializes only the system clock as part of the SYS_CLK_Initialize function. All other clocks including the peripheral clock must be set by calling specific functions.

Changing the System Clock

Changing the system clock is done using the SYS_CLK_SystemFrequencySet function, which expects the clock source and the frequency as arguments. If the clock source is a PLL-based source, a flag is provided so that a user can chose whether to wait until the operation is complete or just trigger the change and exit. The function will return zero if the requested clock is not achievable.

Changing the Peripheral Clock

Changing the Peripheral clock is done using the SYS_CLK_PeripheralFrequencySet function, which expects the clock source and the frequency as arguments. The function will return zero if the requested clock is not achievable. When this occurs, call the SYS_CLK_ PeripheralClosestFrequencyGet function by passing the bus number to it, to determine the closest possible clock.

```
switch ( clockState )
{
case 1:
SYS_CLK_PeripheralFrequencySet ( CLK_BUS_PERIPHERAL_1, CLK_SOURCE_PERIPHERAL_SYSTEMCLK, 8000000, true );

clockState++;

break;

case 2:
if ( SYS_CLK_PeripheralOperationStatus ( CLK_BUS_PERIPHERAL_1 ) == SYS_CLK_OPERATION_COMPLETE )
{
//Peripheral clock setting is completed
clockState++;
}
break;
}
```

Changing/Setting the Reference Clock

Changing or setting the reference clock is done using two functions. Set the module's response to Sleep and Idle mode using the function SYS_CLK_ReferenceClockSetup. Use the SYS_CLK_ReferenceFrequencySet function to set the clock out, which expects the clock source and the frequency as arguments. The function will return zero if the requested clock is not achievable. When this occurs, call the SYS_CLK_ ReferenceClosestFrequencyGet function by passing the bus number to it, to determine the closest possible clock.

```
uint32_t achievedFrequencyHz;
```

```
//I want 200 kHz as output
achievedFrequencyHz = SYS_CLK_ReferenceFrequencySet ( CLK_BUS_REFERENCE_3, CLK_SOURCE_REF_FRC, 200000, true
);

if ( achievedFrequencyHz != 0 )
{
        //Frequency successfully set
}
```

Using Status Functions

Provides a code example to check the clock output.

Description

Checking the Clock Output

The following code provides an example of checking the clock output.

```
uint32_t usbClockOutputHz;
unsigned long sysClockOutputHz;
unsigned long peripheralClockHz;
uint32_t sysClockHz;

    sysClockHz = SYS_CLK_SystemFrequencyGet ( );

    peripheralClockHz = SYS_CLK_PeripheralFrequencyGet ( CLK_BUS_PERIPHERAL_1);
    usbClockOutputHz = SYS_CLK_USBClockFrequencyGet ( CLK_BUS_USB_1 );

    sysClockOutputHz = SYS_CLK_ReferenceFrequencyGet ( CLK_BUS_REFERENCE_1 );
```

Oscillator Tuning

Provides information on tuning the Oscillator.

Description

Oscillator tuning will help compensate for temperature effects on the FRC frequency over a wide range of temperatures. The tuning step size is an approximation; the application is supposed to try different values to achieve the best result. In some devices, there are different tuning modes available.

Direct Number Method

```
//Software must unlock before the start of the tuning.
SYS_OSC_FRCTUNING_DATA tuningInput;
OSC_FRC_TUNING_VALUE tuningValue = OSC_TUNE_TO_CENTRAL_FREQ_PLUS_0_86;

tuningInput.tuningMode = OSC_TUNING_USING_NUMBER;
tuningInput.tuningData = &tuningValue;
SYS_OSC_FRCTuningSet(&tuningInput);
```

Sequential Dithering

To get the Sequential Dithering working, the application is supposed to set the value in seven sequencers and also in the tuning register. Next, the PWM module is configured and the period and pulse width are set. The Oscillator module generates frequencies corresponding to the value specified in these registers in every eighth PWM cycle.

```
//Software must unlock before the start of the tuning.
SYS_OSC_FRCTUNING_DATA tuningInput;
OSC_FRC_TUNING_VALUE tuningValue[8];

//Initialize with tuning values
tuningValue[0] = OSC_TUNE_TO_CENTRAL_MINUS_2_25_PERC;
tuningValue[1] = OSC_TUNE_TO_CENTRAL_MINUS_1_5_PERC;
tuningValue[2] = OSC_TUNE_TO_CENTRAL_MINUS_0_375_PERC;
tuningValue[3] = OSC_TUNE_TO_CENTRAL_PLUS_0_43_PERC;
tuningValue[4] = OSC_TUNE_TO_CENTRAL_PLUS_1_29_PERC;
tuningValue[5] = OSC_TUNE_TO_CENTRAL_PLUS_2_54_PERC;
tuningValue[6] = OSC_TUNE_TO_CENTRAL_MINUS_3_PERC;
tuningValue[7] = OSC_TUNE_TO_CENTRAL_MINUS_3_PERC;

tuningInput.tuningMode = OSC_TUNING_SEQ_DITHER;
```

```
tuningInput.tuningData = tuningValue;
SYS_OSC_FRCTuningSet(&tuningInput);
```

Pseudo-Random Number

In this method, select the tuning mode. Next, configure the PWM module and set the period and pulse width. The Oscillator system generates a 4-bit number based on a pseudo-random number generation algorithm. The Oscillator system then uses this value to tune the FRC oscillator. The module will generate different frequencies corresponding to the generated pseudo-random numbers every eighth PWM cycle.

```
//Software must unlock before the start of the tuning.
SYS_OSC_FRCTUNING_DATA tuningInput;
OSC_FRC_TUNING_VALUE tuningValue = 0x7FFF;

tuningInput.tuningMode = OSC_TUNING_PSEUDO_RANDOM;
tuningInput.tuningData = &tuningValue;
SYS_OSC_FRCTuningSet(&tuningInput);
```

Configuring the Library

Macros

	Name	Description
	SYS_CLK_CONFIG_FREQ_ERROR_LIMIT	Sets the allowed frequency configuration error Hertz.
	SYS_CLK_CONFIG_PRIMARY_XTAL	Gives the Primary oscillator input frequency. This is from the hardware and not the output required frequency.
	SYS_CLK_CONFIG_SECONDARY_XTAL	Gives the Secondary oscillator input frequency. This is from the hardware and not the output required frequency.
	SYS_CLK_CONFIG_SYSPLL_INP_DIVISOR	Gives the System PLL input divisor configured through configuration bits. For some of the devices, the system PLL input divisor is not accessible through software. So use this macro to inform the System service about the input divisor.
	SYS_CLK_CONFIGBIT_USBPLL_DIVISOR	Gives the Oscillator USB clock divisor.
	SYS_CLK_CONFIGBIT_USBPLL_ENABLE	Defines the status of the USB PLL.
	SYS_CLK_CONFIG_EXTERNAL_CLOCK	Gives the clock that is provided on the Reference clock output pin.
	SYS_CLK_UPLL_BEFORE_DIV2_FREQ	Defines the value of the USB clock for the devices which does not have a software controllable USB clock.

Description

The configuration of the Clock System Service Library is based on the file `system_config.h`.

This header file contains the configuration selection for the Clock System Service Library. Based on the selections made, the Clock System Service Library may support the selected features. These configuration settings will apply to all instances of the Clock System Service Library.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

SYS_CLK_CONFIG_FREQ_ERROR_LIMIT Macro

Sets the allowed frequency configuration error Hertz.

File

sys_clk_config_template.h

C

```
#define SYS_CLK_CONFIG_FREQ_ERROR_LIMIT 10
```

Description

Frequency Configuration error

This macro sets the allowed frequency error in Hertz.

Remarks

None.

SYS_CLK_CONFIG_PRIMARY_XTAL Macro

Gives the Primary oscillator input frequency. This is from the hardware and not the output required frequency.

File

sys_clk_config_template.h

C

```
#define SYS_CLK_CONFIG_PRIMARY_XTAL 8000000L
```

Description

Primary Oscillator clock input frequency

This macro gives the Primary Oscillator clock input frequency.

Remarks

Find this out by checking the hardware. examine the crystal connected to the Primary Oscillator input pin.

SYS_CLK_CONFIG_SECONDARY_XTAL Macro

Gives the Secondary oscillator input frequency. This is from the hardware and not the output required frequency.

File

sys_clk_config_template.h

C

```
#define SYS_CLK_CONFIG_SECONDARY_XTAL 8000000
```

Description

Secondary Oscillator clock input frequency

This macro gives the Secondary Oscillator clock input frequency.

Remarks

Find this out by checking the hardware. examine the crystal connected to the Secondary Oscillator input pin.

SYS_CLK_CONFIG_SYSPLL_INP_DIVISOR Macro

Gives the System PLL input divisor configured through configuration bits. For some of the devices, the system PLL input divisor is not accessible through software. So use this macro to inform the System service about the input divisor.

File

sys_clk_config_template.h

C

```
#define SYS_CLK_CONFIG_SYSPLL_INP_DIVISOR 2
```

Description

System PLL input divisor configured through configuration bits

Gives the System PLL input divisor configured through configuration bits. For some of the devices, the system PLL input divisor is not accessible through software. So use this macro to inform the System service about the input divisor. In case of some devices PLL expects its input frequency to be in a specific range. So it is important to set the input divisor get the right input frequency.

Remarks

This must be changed every time you change the System PLL input divisor through the configuration bits.

SYS_CLK_CONFIGBIT_USBPLL_DIVISOR Macro

Gives the Oscillator USB clock divisor.

File

sys_clk_config_template.h

C

```
#define SYS_CLK_CONFIGBIT_USBPLL_DIVISOR 2
```

Description

Oscillator USB clock divisor configured through configuration bits

This macro sets up the Oscillator USB clock divisor. Use OSC_CONFIGBITS_PLLIDIV to select the configuration.

Remarks

This must be changed every time you change the USB PLL divisor through the configuration bits.

SYS_CLK_CONFIGBIT_USBPLL_ENABLE Macro

Defines the status of the USB PLL.

File

sys_clk_config_template.h

C

```
#define SYS_CLK_CONFIGBIT_USBPLL_ENABLE true
```

Description

System clock PLL enable configuration

Setting this macro to 'true' states that USB PLL is enabled through the Configuration bits.

Remarks

This must be changed every time you enable/disable the USB PLL through the configuration bits.

SYS_CLK_CONFIG_EXTERNAL_CLOCK Macro

Gives the clock that is provided on the Reference clock output pin.

File

sys_clk_config_template.h

C

```
#define SYS_CLK_CONFIG_EXTERNAL_CLOCK 4000000
```

Description

External clock frequency

This macro gives the clock that is provided on the Reference clock output pin.

Remarks

None.

SYS_CLK_UPLL_BEFORE_DIV2_FREQ Macro

Defines the value of the USB clock for the devices which does not have a software controllable USB clock.

File

sys_clk_config_template.h

C

```
#define SYS_CLK_UPLL_BEFORE_DIV2_FREQ 48000000
```

Description

USB clock frequency

Defines the value of the USB clock for the devices which does not have a software controllable USB clock(Fully controlled through configuration bits).

Remarks

None.

Building the Library

This section lists the files that are available in the Clock System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/clk`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
sys_clk.h	Clock System Service Library API header file.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/sys_clk.c	Clock System Service Library implementation.
/src/sys_clk_pic32mx.c /src/sys_clk_pic32mx.h	System clock implementation specific to PIC32MX devices.
/src/sys_clk_pic32mz.c /src/sys_clk_pic32mz.h	System clock implementation specific to PIC32MZ devices.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Clock System Service is not dependent upon other modules.

Library Interface

a) System Clock Functions

	Name	Description
	SYS_CLK_SystemFrequencyGet	Gets the system clock frequency in Hertz. **Implementation:** Static/Dynamic
	SYS_CLK_SystemFrequencySet	Configures the System clock of the device to the value specified. **Implementation:** Dynamic

b) Peripheral Clock Functions

	Name	Description
	SYS_CLK_PeripheralFrequencyGet	Gets the selected clock peripheral bus frequency in Hertz. **Implementation:** Static/Dynamic
	SYS_CLK_PeripheralFrequencySet	Configures the peripheral clock of the device to the value specified. **Implementation:** Dynamic

c) Reference Clock Functions

	Name	Description
	SYS_CLK_ReferenceClockSetup	Configures the reference clock of the device to the value specified. **Implementation:** Dynamic

	SYS_CLK_ReferenceFrequencySet	Configures the reference clock of the device to the value specified. **Implementation:** Dynamic
	SYS_CLK_ReferenceFrequencyGet	Gets the selected Reference clock bus frequency in Hertz. **Implementation:** Static/Dynamic

d) Clock Failure Functions

	Name	Description
	SYS_CLK_ClockFailureCallbackRegister	Allows registration of a call back function that will be triggered on a clock failure. **Implementation:** Dynamic
	SYS_CLK_TaskError	Informs the user on a clock failure by invoking the registered call back function. **Implementation:** Dynamic

e) Initialization Functions

	Name	Description
	SYS_CLK_Initialize	Initializes hardware and internal data structure of the System Clock. **Implementation:** Static/Dynamic

f) Secondary Oscillator Functions

	Name	Description
	SYS_CLK_SecondaryOscillatorDisable	Disables the secondary oscillator. **Implementation:** Static/Dynamic
	SYS_CLK_SecondaryOscillatorEnable	Enables the secondary oscillator. **Implementation:** Static/Dynamic
	SYS_CLK_SecondaryOscillatorIsEnabled	Identifies whether secondary oscillator is enabled or disabled. **Implementation:** Static/Dynamic

g) Data Types and Constants

	Name	Description
	SYS_CLK_INIT	Defines the data required to initialize the Oscillator for the Clock System Service.
	CLK_BUSES_PERIPHERAL	Lists the available peripheral clock buses.
	CLK_BUSES_REFERENCE	Lists the available reference clock buses.
	CLK_SOURCES_PERIPHERAL	Lists the available clock sources for the peripheral clock.
	CLK_SOURCES_REFERENCE	Lists the available clock sources for the Reference clock.
	CLK_SOURCES_SYSTEM	Lists the available clock sources for the system clock.
	SYS_CLK_ERROR_HANDLER	Pointer to a CLK System service SYS_CLK_ClockFailureCallbackRegister function.
	SYS_CLK_REFERENCE_SETUP	Defines the data required to initialize the Oscillator for the Clock System Service.
	SYS_CLK_STATUS	Lists all the possible status of a system clock operation.
	SYS_CLK_FRCTune	This function is used for direct value based FRC oscillator tuning. **Implementation:** Dynamic
	SYS_CLK_FRC_TUNING_TYPE	Defines the data type for tuning the FRC Oscillator.

Description

This section describes the APIs of the Clock System Service Library.

Refer to each section for a detailed description.

a) System Clock Functions

SYS_CLK_SystemFrequencyGet Function

Gets the system clock frequency in Hertz.

Implementation: Static/Dynamic

File

sys_clk.h

C

```
uint32_t SYS_CLK_SystemFrequencyGet();
```

Returns

System clock frequency in Hertz.

Description

This function gets the System clock frequency in Hertz.

Preconditions

The SYS_CLK_Initialize function should be called before calling this API.

Example

```
uint32_t sysClockHz;

sysClockHz = SYS_CLK_SystemFrequencyGet ( );
```

Function

uint32_t SYS_CLK_SystemFrequencyGet (void)

SYS_CLK_SystemFrequencySet Function

Configures the System clock of the device to the value specified.

Implementation: Dynamic

File

sys_clk.h

C

```
uint32_t SYS_CLK_SystemFrequencySet(CLK_SOURCES_SYSTEM systemSource, uint32_t systemClockHz, bool
waitUntilComplete);
```

Returns

'Zero' on an unsuccessful operation. Configured system clock frequency on successful operation.

Description

This function configures the clock multipliers and divisors to achieve requested System clock frequency. Initially it checks the difference between the requested value and possible value. If it is not within 'SYS_CLK_CONFIG_FREQ_ERROR_LIMIT', the registers values will not be changed and a value '0' will be returned to let user know that the operation was unsuccessful. If the value requested is acceptable, then it writes to the oscillator registers and return with the newly set frequency.

Remarks

Getting the new frequency doesn't mean that the operation is complete if the API is not called to 'wait until' the operation is complete. The API will return with the possible value immediately. The actual hardware switching may take longer. Use SYS_CLK_SystemClockStatus function to get the status of completion.

Calling this API is only necessary to change the system clock. Use SYS_CLK_Initialize function to set up the system clock initially.

Preconditions

The SYS_CLK_Initialize function should be called before calling this API. To change the clock source, clock switching must be enabled through 'configuration bits'.

Example

```
uint32_t achievedFrequencyHz;

// ******** Blocking call******
achievedFrequencyHz = SYS_CLK_SystemFrequencySet (
    SYS_CLK_SOURCE_PRIMARY, 8000000, true );

if ( achievedFrequencyHz != 0 )
{
    //Clock setting complete
}
```

Parameters

Parameters	Description
systemSource	Clock source means the path of the clock signal. One of the possible value from CLK_SOURCES_SYSTEM enum. See the block diagram in the data sheet to get an idea.
systemClockHz	Required System Clock frequency in Hertz.
waitUntilComplete	Passing this parameter as 'true' will make until the actual hardware operation is complete.

Function

uint32_t SYS_CLK_SystemFrequencySet (CLK_SOURCES_SYSTEM systemSource,
uint32_t systemClockHz, bool waitUntilComplete)

b) Peripheral Clock Functions

SYS_CLK_PeripheralFrequencyGet Function

Gets the selected clock peripheral bus frequency in Hertz.

Implementation: Static/Dynamic

File

sys_clk.h

C

```
uint32_t SYS_CLK_PeripheralFrequencyGet(CLK_BUSES_PERIPHERAL peripheralBus);
```

Returns

Clock frequency in Hertz.

Description

This function gets the selected peripheral bus clock frequency in Hertz.

Remarks

Most devices do not have multiple peripheral clock buses. In those instances, pass CLK_USB_PERIPHERAL_1 as the bus number.

Preconditions

The SYS_CLK_Initialize function should be called before calling this API.

Example

```
unsigned long peripheralClockHz;

peripheralClockHz = SYS_CLK_PeripheralFrequencyGet ( CLK_BUS_PERIPHERAL_5 );
```

Parameters

Parameters	Description
peripheralBus	Reference clock bus selection. One of the possible value from CLK_BUSES_PERIPHERAL enum. For devices that do not have multiple clock channels for Reference clock, CLK_BUS_PERIPHERAL_1 should be the selection.

Function

uint32_t SYS_CLK_PeripheralFrequencyGet (CLK_BUSES_PERIPHERAL peripheralBus)

SYS_CLK_PeripheralFrequencySet Function

Configures the peripheral clock of the device to the value specified.

Implementation: Dynamic

File

sys_clk.h

C

```
uint32_t SYS_CLK_PeripheralFrequencySet(CLK_BUSES_PERIPHERAL peripheralBus, CLK_SOURCES_PERIPHERAL
peripheralSource, uint32_t peripheralClockHz, bool waitUntilComplete);
```

Returns

None.

Description

This function configures the clock multipliers and divisors to achieve requested Peripheral clock frequency. Initially it checks the difference between the requested value and possible value. If it is not within 'SYS_CLK_CONFIG_FREQ_ERROR_LIMIT', the registers values will not be changed and a value '0' will be returned to let user know that the operation was unsuccessful. If the value requested is acceptable, then it writes to the oscillator registers and return with the newly set frequency. If the operation is a failure, SYS_CLK_PeripheralClosestFrequencyGet function will give the closest possible frequency. If the closest possible value is acceptable, user can reconfigure with that value.

Remarks

Most devices do not have multiple Peripheral clock buses. In those instances, pass CLK_BUS_PERIPHERAL_1 as the bus number.

Preconditions

The SYS_CLK_Initialize function should be called before calling this API.

Example

```
uint32_t achievedFrequencyHz;

//Example for MX, single peripheral bus
achievedFrequencyHz = SYS_CLK_PeripheralFrequencySet ( CLK_BUS_PERIPHERAL_1,
    CLK_SOURCE_PERIPHERAL_SYSTEMCLK, 8000000, true );

if ( achievedFrequencyHz !=  0 )
{
    //Frequency successfully set
}

//Example for MZ, multiple peripheral buses
achievedFrequencyHz = SYS_CLK_PeripheralFrequencySet ( CLK_BUS_PERIPHERAL_4,
    CLK_SOURCE_PERIPHERAL_SYSTEMCLK, 8000000, true );

if ( achievedFrequencyHz !=  0 )
{
    //Frequency successfully set
}
```

Parameters

Parameters	Description
peripheralBus	Peripheral bus selection. One of the possible value from CLK_BUSES_PERIPHERAL enum. For devices that do not have multiple clock channels for Peripheral clock, CLK_BUS_PERIPHERAL_1 should be the selection.
peripheralSource	Clock source means the path of the clock signal. One of the possible value from CLK_SOURCES_PERIPHERAL enum. See the block diagram in the data sheet to get an idea.
peripheralClockHz	Clock frequency in Hertz.
waitUntilComplete	Passing this parameter as 'true' will make until the actual hardware operation is complete.

Function

uint32_t SYS_CLK_PeripheralFrequencySet (CLK_BUSES_PERIPHERAL peripheralBus,
CLK_SOURCES_PERIPHERAL peripheralSource, uint32_t peripheralClockHz,
bool waitUntilComplete)

c) Reference Clock Functions

SYS_CLK_ReferenceClockSetup Function

Configures the reference clock of the device to the value specified.

Implementation: Dynamic

File

sys_clk.h

C

```
void SYS_CLK_ReferenceClockSetup(CLK_BUSES_REFERENCE referenceBus, SYS_CLK_REFERENCE_SETUP * refSetup);
```

Returns

None.

Description

This function configures the clock multipliers and divisors to achieve requested Reference clock frequency. Initially it checks the difference between the requested value and possible value. If it is not within 'SYS_CLK_CONFIG_FREQ_ERROR_LIMIT', the registers values will not be changed and a value '0' will be returned to let user know that the operation was unsuccessful. If the value requested is acceptable, then it writes to the oscillator registers and return with the newly set frequency. If the operation is a failure, SYS_CLK_ReferenceClosestFrequencyGet function will give the closest possible frequency. If the closest possible value is acceptable, user can reconfigure with that value.

Remarks

Most devices do not have multiple reference clock buses. In those instances, pass CLK_BUS_REFERENCE_1 as the bus number.

Preconditions

The SYS_CLK_Initialize function should be called before calling this API.

Example

```
SYS_CLK_REFERENCE_SETUP refSetup;

refSetup.suspendInSleep = true;
refSetup.stopInIdle = true;

SYS_CLK_ReferenceClockSetup ( ( CLK_BUS_REFERENCE_3, &refSetup );
```

Parameters

Parameters	Description
referenceBus	Reference clock bus selection. One of the possible value from CLK_BUSES_REFERENCE enum. For devices that do not have multiple clock channels for Reference clock, CLK_BUS_REFERENCE_1 should be the selection.
refSetup	A structure which holds the reference oscillator configuration.

Function

void SYS_CLK_ReferenceClockSetup (CLK_BUSES_REFERENCE referenceBus,
SYS_CLK_REFERENCE_SETUP refSetup)

SYS_CLK_ReferenceFrequencySet Function

Configures the reference clock of the device to the value specified.

Implementation: Dynamic

File

sys_clk.h

C

```
uint32_t SYS_CLK_ReferenceFrequencySet(CLK_BUSES_REFERENCE referenceBus, CLK_SOURCES_REFERENCE
referenceSource, uint32_t referenceClockHz, bool waitUntilComplete);
```

Returns

None.

Description

This function configures the clock multipliers and divisors to achieve requested Reference clock frequency. Initially it checks the difference between the requested value and possible value. If it is not within 'SYS_CLK_CONFIG_FREQ_ERROR_LIMIT', the registers values will not be changed and a value '0' will be returned to let user know that the operation was unsuccessful. If the value requested is acceptable, then it writes to the oscillator registers and return with the newly set frequency. If the operation is a failure, SYS_CLK_ReferenceClosestFrequencyGet function will

give the closest possible frequency. If the closest possible value is acceptable, user can reconfigure with that value.

Remarks

Most devices do not have multiple reference clock buses. In those instances, pass CLK_BUS_REFERENCE_1 as the bus number.

Preconditions

The SYS_CLK_Initialize function should be called before calling this API. To change the clock source clock switching must be enabled through 'configuration bits'.

Example

```
uint32_t achievedFrequencyHz;

//I want 200 KHz as output
achievedFrequencyHz = SYS_CLK_ReferenceFrequencySet ( CLK_BUS_REFERENCE_3,
    CLK_SOURCE_REF_FRC, 200000, true );

if ( achievedFrequencyHz != 0 )
{
    //Frequency successfully set
}
```

Parameters

Parameters	Description
referenceBus	Reference clock bus selection. One of the possible value from CLK_BUSES_REFERENCE enum. For devices that do not have multiple clock channels for Reference clock, CLK_BUS_REFERENCE_1 should be the selection.
referenceSource	Clock source means the path of the clock signal. One of the possible value from CLK_SOURCES_REFERENCE enum. See the block diagram in the data sheet to get an idea.
referenceClockHz	Clock frequency in Hertz.
waitUntilComplete	Passing this parameter as 'true' will make until the actual hardware operation is complete.

Function

uint32_t SYS_CLK_ReferenceFrequencySet (CLK_BUSES_REFERENCE referenceBus,
CLK_SOURCES_REFERENCE referenceSource, uint32_t referenceClockHz,
bool waitUntilComplete)

SYS_CLK_ReferenceFrequencyGet Function

Gets the selected Reference clock bus frequency in Hertz.

Implementation: Static/Dynamic

File

sys_clk.h

C

```
uint32_t SYS_CLK_ReferenceFrequencyGet(CLK_BUSES_REFERENCE referenceBus);
```

Returns

Clock frequency in Hertz.

Description

This function gets frequency of the selected Reference clock bus in Hertz.

Remarks

None.

Preconditions

The SYS_CLK_Initialize function should be called before calling this API.

Example

```
unsigned long sysClockOutputHz;

sysClockOutputHz = SYS_CLK_ReferenceFrequencyGet ( CLK_BUS_REFERENCE_3 );
```

Parameters

Parameters	Description
peripheralBus	Reference clock bus selection. One of the possible value from CLK_BUSES_REFERENCE enum. For devices that do not have multiple clock channels for Reference clock, CLK_BUS_REFERENCE_1 should be the selection.

Function

uint32_t SYS_CLK_ReferenceFrequencyGet (CLK_BUSES_REFERENCE referenceBus)

d) Clock Failure Functions

SYS_CLK_ClockFailureCallbackRegister Function

Allows registration of a call back function that will be triggered on a clock failure.

Implementation: Dynamic

File

sys_clk.h

C

```
void SYS_CLK_ClockFailureCallbackRegister(SYS_CLK_ERROR_HANDLER callback);
```

Returns

None.

Description

This function allows registration of a callback function that will be triggered on a clock failure.

Remarks

This function is not available on all devices. Please refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
SYS_CLK_ClockFailureCallbackRegister ( &ErrorHandle );

void ErrorHandle (void)
{
    //Handle the error.
}
```

Parameters

Parameters	Description
callback	Address of a callback function of type 'SYS_CLK_ERROR_HANDLER'.

Function

void SYS_CLK_ClockFailureCallbackRegister (SYS_CLK_ERROR_HANDLER callback)

SYS_CLK_TaskError Function

Informs the user on a clock failure by invoking the registered call back function.

Implementation: Dynamic

File

sys_clk.h

C

```
void SYS_CLK_TaskError();
```

Returns

None.

Description

This function informs the user on a clock failure by invoking the registered call back function. This must be called from the Fail Safe Clock Monitor (FSCM) interrupt service routine.

Remarks

None.

Preconditions

None.

Example

```
void clockMonitorISR ( void )
{
    SYS_CLK_TaskError ( void );
}
```

Parameters

Parameters	Description
index	Identifies the desired System Clock

Function

void SYS_CLK_TaskError (void)

e) Initialization Functions

SYS_CLK_Initialize Function

Initializes hardware and internal data structure of the System Clock.

Implementation: Static/Dynamic

File

sys_clk.h

C

```
void SYS_CLK_Initialize(const SYS_CLK_INIT * clkInit);
```

Returns

None.

Description

This function initializes the hardware and internal data structure of System Clock Service.

Remarks

This is API must be be called at the time of system initialization to initialize the oscillator. Call this again to change any of the initialization settings.

If the frequency settings are already done through the 'configuration bits', call this API by passing 'NULL' to it. In any case calling this API is necessary get the internal data structures updated.

To change the clock source clock switching must be enabled through 'configuration bits'.

Preconditions

None.

Example

```
//Example 1: Do not alter the configuration bit settings
SYS_CLK_Initialize ( NULL );

//Example 2: Want to change the configuration bit settings
SYS_CLK_INIT generalInit;
```

```
// Populate the oscillator initialization structure
generalInit.systemClockSource = SYS_CLK_SOURCE_FRC_SYSPLL;
generalInit.systemClockFrequencyHz = 30000000;
generalInit.onWaitInstruction = OSC_ON_WAIT_SLEEP;
generalInit.secondaryOscKeepEnabled = true;
SYS_CLK_Initialize (&generalInit);

//Wait until complete
while ( SYS_CLK_SystemOperationStatus ( ) !=  SYS_CLK_OPERATION_COMPLETE );

//Configuration success
```

Parameters

Parameters	Description
clkInit	Pointer to a data structure containing any data necessary to initialize the System Clock. This pointer can be NULL if no data is required as static overrides have been provided.

Function

void SYS_CLK_Initialize (const SYS_CLK_INIT * clkInit)

f) Secondary Oscillator Functions

SYS_CLK_SecondaryOscillatorDisable Function

Disables the secondary oscillator.

Implementation: Static/Dynamic

File

sys_clk.h

C

```
void SYS_CLK_SecondaryOscillatorDisable();
```

Returns

None.

Description

This function disables the secondary oscillator.

Remarks

None.

Preconditions

The SYS_CLK_Initialize function should be called before calling this API.

Example

```
SYS_CLK_SecondaryOscillatorDisable ( );
```

Function

void SYS_CLK_SecondaryOscillatorDisable (void)

SYS_CLK_SecondaryOscillatorEnable Function

Enables the secondary oscillator.

Implementation: Static/Dynamic

File

sys_clk.h

C

```
void SYS_CLK_SecondaryOscillatorEnable();
```

Returns

None.

Description

This function enables secondary oscillator which can be used as a clock source for peripherals like RTCC, Timer etc.. The SOSC requires a warm-up period of 1024 before it can be used as a clock source.

Remarks

None.

Preconditions

The SYS_CLK_Initialize function should be called before calling this API.

Example

```
SYS_CLK_SecondaryOscillatorEnable ( );
```

Function

void SYS_CLK_SecondaryOscillatorEnable (void)

SYS_CLK_SecondaryOscillatorIsEnabled Function

Identifies whether secondary oscillator is enabled or disabled.

Implementation: Static/Dynamic

File

sys_clk.h

C

```
bool SYS_CLK_SecondaryOscillatorIsEnabled();
```

Returns

true - If Secondary oscillator is enabled. false - If Secondary oscillator is disabled.

Description

This function identifies whether the secondary oscillator is enabled or disabled.

Remarks

None.

Preconditions

The SYS_CLK_Initialize function should be called before calling this API.

Example

```
bool status;

status = SYS_CLK_SecondaryOscillatorIsEnabled ( );
```

Function

bool SYS_CLK_SecondaryOscillatorIsEnabled (void)

g) Data Types and Constants

SYS_CLK_INIT Structure

Defines the data required to initialize the Oscillator for the Clock System Service.

File

sys_clk.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  CLK_SOURCES_SYSTEM systemClockSource;
  uint32_t systemClockFrequencyHz;
  bool waitTillComplete;
  bool secondaryOscKeepEnabled;
  OSC_OPERATION_ON_WAIT onWaitInstruction;
} SYS_CLK_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
CLK_SOURCES_SYSTEM systemClockSource;	Initializations for System clock
uint32_t systemClockFrequencyHz;	Set the System clock frequency
bool waitTillComplete;	Wait until the clock switch operation is completed
bool secondaryOscKeepEnabled;	Even though the secondary oscillator is not used, keeping the oscillator running, allows a fast switch to the lower system clock for low-power operation
OSC_OPERATION_ON_WAIT onWaitInstruction;	System action on a 'Wait' instruction

Description

Clock System Service Reference Oscillator initialization data

This structure defines the data required to initialize the Oscillator for the Clock System Service.

Remarks

This structure only includes the necessary settings for the clock module. Other features like USB clock and reference clock are considered to be optional and not every system will use it. There are dedicated APIs for configuring those.

CLK_BUSES_PERIPHERAL Enumeration

Lists the available peripheral clock buses.

File

sys_clk.h

C

```
typedef enum {
  CLK_BUS_PERIPHERAL_1 = 0x00,
  CLK_BUS_PERIPHERAL_2 = 0x01,
  CLK_BUS_PERIPHERAL_3 = 0x02,
  CLK_BUS_PERIPHERAL_4 = 0x03,
  CLK_BUS_PERIPHERAL_5 = 0x04,
  CLK_BUS_PERIPHERAL_6 = 0x05,
  CLK_BUS_PERIPHERAL_7 = 0x06,
  CLK_BUS_PERIPHERAL_8 = 0x07
} CLK_BUSES_PERIPHERAL;
```

Members

Members	Description
CLK_BUS_PERIPHERAL_1 = 0x00	Select peripheral bus 1. This should be the selection for the devices which doesn't have multiple peripheral buses
CLK_BUS_PERIPHERAL_2 = 0x01	Select peripheral bus 2
CLK_BUS_PERIPHERAL_3 = 0x02	Select peripheral bus 3
CLK_BUS_PERIPHERAL_4 = 0x03	Select peripheral bus 4
CLK_BUS_PERIPHERAL_5 = 0x04	Select peripheral bus 5
CLK_BUS_PERIPHERAL_6 = 0x05	Select peripheral bus 6
CLK_BUS_PERIPHERAL_7 = 0x06	Select peripheral bus 7
CLK_BUS_PERIPHERAL_8 = 0x07	Select peripheral bus 8

Description

Peripherals Clock Buses enumeration

This enumeration lists all of the available peripheral clock buses. This is used by the SYS_CLK_PeripheralFrequencyGet and SYS_CLK_PeripheralFrequencySet functions.

Remarks

The system service covers a wide range of devices. Not all the elements listed in here will be relevant for the your device.

CLK_BUSES_REFERENCE Enumeration

Lists the available reference clock buses.

File

sys_clk.h

C

```
typedef enum {
  CLK_BUS_REFERENCE_1 = 0x00,
  CLK_BUS_REFERENCE_2 = 0x01,
  CLK_BUS_REFERENCE_3 = 0x02,
  CLK_BUS_REFERENCE_4 = 0x03,
  CLK_BUS_REFERENCE_5 = 0x04
} CLK_BUSES_REFERENCE;
```

Members

Members	Description
CLK_BUS_REFERENCE_1 = 0x00	Select Reference bus 1. This should be the selection for the devices which doesn't have multiple Reference buses
CLK_BUS_REFERENCE_2 = 0x01	Select Reference bus 2
CLK_BUS_REFERENCE_3 = 0x02	Select Reference bus 3
CLK_BUS_REFERENCE_4 = 0x03	Select Reference bus 4
CLK_BUS_REFERENCE_5 = 0x04	Select Reference bus 5

Description

Reference Clock Buses enumeration

This enumeration lists all of the available Reference clock buses. This is used by the SYS_CLK_ReferenceFrequencyGet and SYS_CLK_ReferenceFrequencySet functions.

Remarks

The system service covers a wide range of devices. Not all the elements listed in here will be relevant for the your device.

CLK_SOURCES_PERIPHERAL Enumeration

Lists the available clock sources for the peripheral clock.

File

sys_clk.h

C

```
typedef enum {
  CLK_SOURCE_PERIPHERAL_SYSTEMCLK
} CLK_SOURCES_PERIPHERAL;
```

Members

Members	Description
CLK_SOURCE_PERIPHERAL_SYSTEMCLK	Source of clock is system clock

Description

Peripheral clock sources enumeration

This enumeration lists all the available peripheral clock sources. This is used by the SYS_CLK_PeripheralFrequencySet function.

Remarks

None.

CLK_SOURCES_REFERENCE Enumeration

Lists the available clock sources for the Reference clock.

File

sys_clk.h

C

```
typedef enum {
  CLK_SOURCE_REF_SYSTEMCLK,
  CLK_SOURCE_REF_PBCLK_BUS1,
  CLK_SOURCE_REF_PRIMARY,
  CLK_SOURCE_REF_FRC,
  CLK_SOURCE_REF_LPRC,
  CLK_SOURCE_REF_SECONDARY,
  CLK_SOURCE_REF_USBPLL_OUT,
  CLK_SOURCE_REF_SYSPLL_OUT,
  CLK_SOURCE_REF_EXTERNAL,
  CLK_SOURCE_REF_BFRC
} CLK_SOURCES_REFERENCE;
```

Members

Members	Description
CLK_SOURCE_REF_SYSTEMCLK	Source of Reference clock is System clock
CLK_SOURCE_REF_PBCLK_BUS1	Source of Reference clock is Peripheral clock
CLK_SOURCE_REF_PRIMARY	Source of Reference clock is Primary oscillator
CLK_SOURCE_REF_FRC	Source of Reference clock is Fast RC oscillator
CLK_SOURCE_REF_LPRC	Source of Reference clock is Low power RC oscillator
CLK_SOURCE_REF_SECONDARY	Source of Reference clock is Secondary oscillator
CLK_SOURCE_REF_USBPLL_OUT	Source of clock is output of USB PLL This is used only for the Reference clock.
CLK_SOURCE_REF_SYSPLL_OUT	Source of clock is the output of System PLL. Input to the system PLL may be Primary or FRC
CLK_SOURCE_REF_EXTERNAL	Source of clock is external(from the pin)
CLK_SOURCE_REF_BFRC	Source of Reference clock is backup Fast RC oscillator

Description

Reference clock sources enumeration

This enumeration lists all the available peripheral clock sources. This is used by the SYS_CLK_ReferenceFrequencySet function.

Remarks

None.

CLK_SOURCES_SYSTEM Enumeration

Lists the available clock sources for the system clock.

File

sys_clk.h

C

```
typedef enum {
  SYS_CLK_SOURCE_FRC,
  SYS_CLK_SOURCE_FRC_SYSPLL,
  SYS_CLK_SOURCE_PRIMARY,
  SYS_CLK_SOURCE_PRIMARY_SYSPLL,
  SYS_CLK_SOURCE_SECONDARY,
  SYS_CLK_SOURCE_LPRC,
  SYS_CLK_SOURCE_FRC_BY_16,
  SYS_CLK_SOURCE_FRC_BY_DIV
} CLK_SOURCES_SYSTEM;
```

Members

Members	Description
SYS_CLK_SOURCE_FRC	Source of clock is internal fast RC
SYS_CLK_SOURCE_FRC_SYSPLL	Source of clock is internal fast RC multiplied by system PLL
SYS_CLK_SOURCE_PRIMARY	Source of clock is primary oscillator
SYS_CLK_SOURCE_PRIMARY_SYSPLL	Source of clock is primary oscillator multiplied by the System PLL value and divided by the divisor configured by software
SYS_CLK_SOURCE_SECONDARY	Source of clock is secondary oscillator
SYS_CLK_SOURCE_LPRC	Source of clock is internal low power RC
SYS_CLK_SOURCE_FRC_BY_16	Source of clock is internal fast RC divided by the divisor configured in software
SYS_CLK_SOURCE_FRC_BY_DIV	Source of clock is internal fast RC divided by the divisor configured in software

Description

System Clock sources enumeration

This enumeration lists all the available clock sources for the system clock. This is used by the SYS_CLK_INIT structure.

Remarks

None.

SYS_CLK_ERROR_HANDLER Type

Pointer to a CLK System service SYS_CLK_ClockFailureCallbackRegister function.

File

sys_clk.h

C

```
typedef void (* SYS_CLK_ERROR_HANDLER)(CLK_SOURCES_SYSTEM systemSource, uint32_t systemFrequencyHz);
```

Returns

None.

Description

CLK System Service Error Handler Function Pointer

This data type defines the type of the CLK System Service Error Handler callback function. The parameters are described here and a partial example implementation is provided. On a clock failure, the device will switch its clock source to the FRC clock. By registering the callback function user will get the current System clock source and the clock frequency. User can use this information to try switching back.

Remarks

None.

Example

```
void SYS_CLK_ClockFailureCallbackRegister(  errorHandler );

errorHandler ( CLK_SOURCES_SYSTEM systemSource, uint32_t systemFrequencyHz )
{
    //Log a warning
    //Try switching back to the old source
}
```

Parameters

Parameters	Description
systemSource	System clock source.
systemFrequencyHz	System clock frequency in hertz.

SYS_CLK_REFERENCE_SETUP Structure

Defines the data required to initialize the Oscillator for the Clock System Service.

File

sys_clk.h

C

```
typedef struct {
  bool suspendInSleep;
  bool stopInIdle;
} SYS_CLK_REFERENCE_SETUP;
```

Members

Members	Description
bool suspendInSleep;	Reference clock module should be stop its operation in 'Sleep' mode
bool stopInIdle;	Reference clock module should be stop its operation in 'Idle' mode

Description

Clock System Service Reference Oscillator initialization data

This structure defines the data required to initialize the Oscillator for the Clock System Service.

Remarks

This structure only includes the necessary settings for the clock module. Other features like USB clock and reference clock are considered to be optional and not every system will use it. There are dedicated APIs for configuring those.

SYS_CLK_STATUS Enumeration

Lists all the possible status of a system clock operation.

File

sys_clk.h

C

```
typedef enum {
  SYS_CLK_OPERATION_COMPLETE,
  SYS_CLK_OPERATION_IN_PROGRESS,
  SYS_CLK_OPERATION_FAIL_SWITCHING_DISABLED,
  SYS_CLK_OPERATION_FAIL_NOT_ACHIEVABLE,
  SYS_CLK_OPERATION_FAIL_REGISTER_NOT_READY,
  SYS_CLK_OPERATION_FAIL_USBPLL_NOT_ENABLED,
  SYS_CLK_OPERATION_FAIL_PLL_LOCK_TIMEOUT,
  SYS_CLK_PLL_INVALID_INP_FREQUENCY
} SYS_CLK_STATUS;
```

Members

Members	Description
SYS_CLK_OPERATION_COMPLETE	The last operation was successful
SYS_CLK_OPERATION_IN_PROGRESS	Operation started, may be PLL is not yet locked
SYS_CLK_OPERATION_FAIL_SWITCHING_DISABLED	The last operation was not successful, Clock switching is not enabled through configuration bits
SYS_CLK_OPERATION_FAIL_NOT_ACHIEVABLE	The last operation was not successful, requested frequency is not achievable
SYS_CLK_OPERATION_FAIL_REGISTER_NOT_READY	The last operation was not successful, register is not ready to take the new value
SYS_CLK_OPERATION_FAIL_USBPLL_NOT_ENABLED	The last operation was not successful, USB PLL is not enabled in through the 'configuration bits'. The same must be update in system_config.h
SYS_CLK_OPERATION_FAIL_PLL_LOCK_TIMEOUT	PLL Lock time out. A restart is the only option
SYS_CLK_PLL_INVALID_INP_FREQUENCY	PLL cannot work with the current input clock frequency, adjust the clock input or the input divisor. Check the device data sheet to know the PLL input requirement

Description

System clock operation status

This enumeration lists all the possible status of a system clock operation.

Remarks

None.

SYS_CLK_FRCTune Function

This function is used for direct value based FRC oscillator tuning.

Implementation: Dynamic

File

sys_clk.h

C

```
void SYS_CLK_FRCTune(SYS_CLK_FRC_TUNING_TYPE tuningData);
```

Returns

None.

Description

This function tunes the FRC as per the given value. FRC tuning functionality has been provided to help customers compensate for temperature effects on the FRC frequency over a wide range of temperatures.

Remarks

The tuning step size is an approximation, and is neither characterized, nor tested.

This API can only be used with devices that support direct value based FRC tuning. Refer to the specific device data sheet to determine whether this feature exists for your device.

Preconditions

The device selected must support the oscillator tuning feature.

Example

```
SYS_CLK_FRCTune ( ox30 );
```

Parameters

Parameters	Description
tuningData	One of the possible value of Tuning. Refer to the specific device data sheet for possible values.

Function

void SYS_CLK_FRCTune (SYS_CLK_FRC_TUNING_TYPE tuningData)

SYS_CLK_FRC_TUNING_TYPE Type

Defines the data type for tuning the FRC Oscillator.

File

sys_clk.h

C

```
typedef unsigned int SYS_CLK_FRC_TUNING_TYPE;
```

Description

Clock module FRC tuning type

This structure defines the data type to tune the FRC Oscillator.

Remarks

None.

Files

Files

Name	Description
sys_clk.h	Clock system service interface header.

sys_clk_config_template.h	Clock System Service configuration templates.

Description

This section lists the source and header files used by the library.

sys_clk.h

Clock system service interface header.

Enumerations

	Name	Description
	CLK_BUSES_PERIPHERAL	Lists the available peripheral clock buses.
	CLK_BUSES_REFERENCE	Lists the available reference clock buses.
	CLK_SOURCES_PERIPHERAL	Lists the available clock sources for the peripheral clock.
	CLK_SOURCES_REFERENCE	Lists the available clock sources for the Reference clock.
	CLK_SOURCES_SYSTEM	Lists the available clock sources for the system clock.
	SYS_CLK_STATUS	Lists all the possible status of a system clock operation.

Functions

	Name	Description
	SYS_CLK_ClockFailureCallbackRegister	Allows registration of a call back function that will be triggered on a clock failure. **Implementation:** Dynamic
	SYS_CLK_FRCTune	This function is used for direct value based FRC oscillator tuning. **Implementation:** Dynamic
	SYS_CLK_Initialize	Initializes hardware and internal data structure of the System Clock. **Implementation:** Static/Dynamic
	SYS_CLK_PeripheralFrequencyGet	Gets the selected clock peripheral bus frequency in Hertz. **Implementation:** Static/Dynamic
	SYS_CLK_PeripheralFrequencySet	Configures the peripheral clock of the device to the value specified. **Implementation:** Dynamic
	SYS_CLK_ReferenceClockSetup	Configures the reference clock of the device to the value specified. **Implementation:** Dynamic
	SYS_CLK_ReferenceFrequencyGet	Gets the selected Reference clock bus frequency in Hertz. **Implementation:** Static/Dynamic
	SYS_CLK_ReferenceFrequencySet	Configures the reference clock of the device to the value specified. **Implementation:** Dynamic
	SYS_CLK_SecondaryOscillatorDisable	Disables the secondary oscillator. **Implementation:** Static/Dynamic
	SYS_CLK_SecondaryOscillatorEnable	Enables the secondary oscillator. **Implementation:** Static/Dynamic
	SYS_CLK_SecondaryOscillatorIsEnabled	Identifies whether secondary oscillator is enabled or disabled. **Implementation:** Static/Dynamic
	SYS_CLK_SystemFrequencyGet	Gets the system clock frequency in Hertz. **Implementation:** Static/Dynamic
	SYS_CLK_SystemFrequencySet	Configures the System clock of the device to the value specified. **Implementation:** Dynamic
	SYS_CLK_TaskError	Informs the user on a clock failure by invoking the registered call back function. **Implementation:** Dynamic

Structures

	Name	Description
	SYS_CLK_INIT	Defines the data required to initialize the Oscillator for the Clock System Service.
	SYS_CLK_REFERENCE_SETUP	Defines the data required to initialize the Oscillator for the Clock System Service.

Types

	Name	Description
	SYS_CLK_ERROR_HANDLER	Pointer to a CLK System service SYS_CLK_ClockFailureCallbackRegister function.
	SYS_CLK_FRC_TUNING_TYPE	Defines the data type for tuning the FRC Oscillator.

Description

Clock System Service Interface Definition

This file contains the interface definition for the Clock System Service. It provides a way to interact with the Clock subsystem to manage the timing requests supported by the system.

File Name

sys_clk.h

Company

Microchip Technology Inc.

sys_clk_config_template.h

Clock System Service configuration templates.

Macros

	Name	Description
	SYS_CLK_CONFIG_EXTERNAL_CLOCK	Gives the clock that is provided on the Reference clock output pin.
	SYS_CLK_CONFIG_FREQ_ERROR_LIMIT	Sets the allowed frequency configuration error Hertz.
	SYS_CLK_CONFIG_PRIMARY_XTAL	Gives the Primary oscillator input frequency. This is from the hardware and not the output required frequency.
	SYS_CLK_CONFIG_SECONDARY_XTAL	Gives the Secondary oscillator input frequency. This is from the hardware and not the output required frequency.
	SYS_CLK_CONFIG_SYSPLL_INP_DIVISOR	Gives the System PLL input divisor configured through configuration bits. For some of the devices, the system PLL input divisor is not accessible through software. So use this macro to inform the System service about the input divisor.
	SYS_CLK_CONFIGBIT_USBPLL_DIVISOR	Gives the Oscillator USB clock divisor.
	SYS_CLK_CONFIGBIT_USBPLL_ENABLE	Defines the status of the USB PLL.
	SYS_CLK_UPLL_BEFORE_DIV2_FREQ	Defines the value of the USB clock for the devices which does not have a software controllable USB clock.

Description

Clock System Service Configuration Templates

This file contains constants to configure the Clock System Service.

File Name

sys_clk_config_template.h

Company

Microchip Technology Inc.

Command Processor System Service Library

This section describes the Command Processor System Service Library.

Introduction

This library provides an abstraction of the Command Processor System Service Library that is available on the Microchip family of PIC32 microcontrollers with a convenient C language interface. It provides the framework for a command console that can support commands from multiple client sources.

Description

The Command Processor System Service provides the developer with simple APIs to implement a command console. The console may support commands from one or more client software modules.

Using the Library

This topic describes the basic architecture of the Command Processor System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_command.h`

The interface to the Command Processor System Service library is defined in the `sys_command.h` header file.

Please refer to the What is MPLAB Harmony? section for how the Command Processor System Service interacts with the framework.

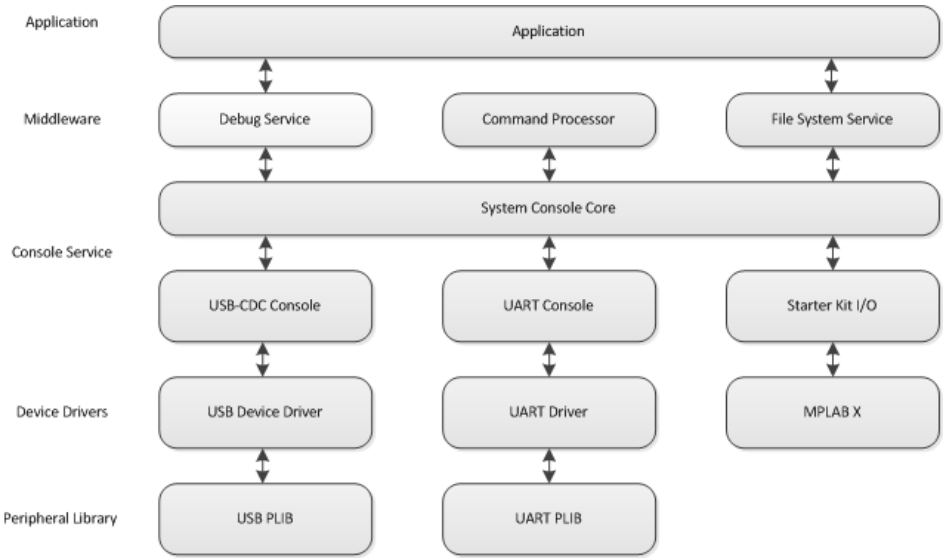
Abstraction Model

This topic provides a description of the software abstraction for the Command Processor System Service.

Description

The Command Processor System Service Library is a collection of operations specific to supporting user input commands. The commands can be uniquely native to each client service and can be dynamically added. The library can support multiple client services at once.

The Command Processor System Service is a module that works closely with the Console System Service to present a user interface command prompt.



Library Overview

Refer to the section System Services Introduction for how the system services operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Device Control module.

Library Interface Section	Description
System Interaction Functions	Provides system module APIs. Service initialization, deinitialization, reinitialization and status functions.
Core Functionality Functions	State Machine and control functions.
Data Types and Constants	Enumerations, structures, and constants.

How the Library Works

System Interaction

Initialization and Reinitialization

Initialization of the Command Processor System Service initializes the status of the module and sets the state of the internal state machine.

The Command Processor Status and Tasks routines are required for the normal operation of the service.

Core Functionality

Adding Commands

The Command Processor System Service will accept commands dynamically during run-time through the follow interface:

```
SYS_COMMAND_ADDGRP(const _SYS_CMD_DCPT* pCmdTbl, int nCmds, const char*
                   groupName, const char* menuStr)
```

Command Prompt

In conjunction with the Console System Service, the Command Processor System Service provides the user with an ASCII command prompt. It will interpret the commands entered at the prompt and process accordingly. It also supports command history, as well as command help.

Configuring the Library

The configuration of the Command Processor System Service is based on the file `system_config.h`.

This header file contains the configuration selection for the Device Control system service. Based on the selections made, the Device Control System Service Library may support the selected features. These configuration settings will apply to all instances of the Device Control System Service.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Command Processor System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/command`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
sys_command.h	Command Processor System Service Library API header file.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/sys_command.c	Command Processor System Service Library implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Command Processor System Service is dependent on the following library:

- Console System Service Library

Library Interface

a) System Interaction Functions

	Name	Description
	SYS_CMD_Initialize	Initializes data for the instance of the Command Processor module.
	SYS_CMD_DELETE	Removes CMDIO parameters from the Command Processor System Service console.
	SYS_CMD_ADDGRP	Allows clients to add command process to the Command Processor System Service.
	SYS_CMD_MESSAGE	Outputs a message to the Command Processor System Service console.
	SYS_CMD_PRINT	Outputs a printout to the Command Processor System Service console.
	SYS_CMD_READY_TO_READ	Allows upper layer application to confirm that the command module is ready to accept command input
	SYS_CMD_READY_TO_WRITE	Allows upper layer application to confirm that the command module is ready to write output to the Console System Service.
	SYS_CMD_Tasks	Maintains the Command Processor System Service's internal state machine.
	SYS_CMD_RegisterCallback	Registers a callback function with the command service that will be executed when the lower layer read or write queue is empty.

b) Command I/O Functions

	Name	Description
	SYS_CMDIO_GET_HANDLE	Gets the CMDIO Handle base via index.
	SYS_CMDIO_ADD	Adds CMDIO parameters to the Command Processor System Service console.

c) Data Types and Constants

	Name	Description
	_promptStr	prompt string
	COMMAND_HISTORY_DEPTH	Command Processor System Service Maximum Depth of Command History.
	ESC_SEQ_SIZE	standard VT100 escape sequences
	LINE_TERM	line terminator
	MAX_CMD_ARGS	Command Processor System Service Maximum Number of Argument definitions.
	MAX_CMD_GROUP	Command Processor System Service Maximum Number of Command Group definitions.
	SYS_CMD_MAX_LENGTH	Command Processor System Service Command Buffer Maximum Length definition.
	SYS_CMD_READ_BUFFER_SIZE	Command Processor System Service Read Buffer Size definition.
	SYS_CMD_DEVICE_NODE	Defines the data structure to store each command instance.
	SYS_CMD_INIT	Identifies the system command initialize structure.
	SYS_CMD_API	Identifies the Command API structure.
	SYS_CMD_DATA_RDY_FNC	Identifies a data available function API.
	SYS_CMD_DESCRIPTOR	a simple command descriptor
	SYS_CMD_DESCRIPTOR_TABLE	table containing the supported commands
	SYS_CMD_DEVICE_LIST	Defines the list structure to store a list of command instances.
	SYS_CMD_FNC	Identifies the command process function API.
	SYS_CMD_GETC_FNC	Identifies a get single character function API.
	SYS_CMD_HANDLE	Identifies a particular Command I/O instance.
	SYS_CMD_INIT_DATA	Defines the data structure to store each command.
	SYS_CMD_MSG_FNC	Identifies a message function API.
	SYS_CMD_PRINT_FNC	Identifies a print function API.
	SYS_CMD_PUTC_FNC	Identifies a single character print function API.
	SYS_CMD_READC_FNC	Identifies a read single character function API.
	SYS_CMD_STATE	Defines the various states that can be achieved by a Command instance.
	SYS_CMD_CONSOLE_IO_PARAM	Defines whether the command module is single character or full command read
	SYS_CMD_BUFFER_DMA_READY	Define this for MX #define SYS_CMD_BUFFER_DMA_READY __attribute__((coherent)) __attribute__((aligned(4))) //Define this for MZ
	SYS_CONSOLE_MESSAGE	ifdef SYS_CONSOLE_REMOVE_APIS
	SYS_CONSOLE_PRINT	This is macro SYS_CONSOLE_PRINT.
	SYS_CMD_DEVICE_MAX_INSTANCES	This is macro SYS_CMD_DEVICE_MAX_INSTANCES.
	SYS_CMD_MESSAGE	This is macro SYS_CMD_MESSAGE.

	SYS_CMD_PRINT	This is macro SYS_CMD_PRINT.
	SYS_CMD_CallbackFunction	Command Callback Function Handle.
	SYS_CMD_EVENT	Identifies the Command Event Types

Description

This section describes the APIs of the Command Processor System Service Library.

Refer to each section for a detailed description.

a) System Interaction Functions

SYS_CMD_Initialize Function

Initializes data for the instance of the Command Processor module.

File

sys_command.h

C

```
bool SYS_CMD_Initialize(const SYS_MODULE_INIT * const init);
```

Returns

- true - Indicates success
- false - Indicates an error occurred

Description

This function initializes the Command Processor module. It also initializes any internal data structures.

Remarks

This routine should only be called once during system initialization.

Preconditions

None.

Parameters

Parameters	Description
init	Pointer to a data structure containing any data necessary to initialize the sys command. This pointer may be null if no data is required because static overrides have been provided.

Function

bool SYS_CMD_Initialize(const SYS_MODULE_INIT * const init)

SYS_CMD_DELETE Function

Removes CMDIO parameters from the Command Processor System Service console.

File

sys_command.h

C

```
bool SYS_CMD_DELETE(SYS_CMD_DEVICE_NODE* pDevNode);
```

Returns

None.

Description

This function removes CMDIO parameters from the Command Processor System Service console.

Remarks

None.

Preconditions

SYS_CMD_Initialize was successfully run once.

Function

bool SYS_CMD_DELETE(SYS_CMD_DEVICE_NODE* pDevNode);

SYS_CMD_ADDGRP Function

Allows clients to add command process to the Command Processor System Service.

File

sys_command.h

C

```
bool SYS_CMD_ADDGRP(const SYS_CMD_DESCRIPTOR* pCmdTbl, int nCmds, const char* groupName, const char* menuStr);
```

Returns

- true - Indicates success
- false - Indicates an error occurred

Description

Client modules may call this function to add command process to the Command Processor System Service.

Remarks

None.

Preconditions

SYS_CMD_Initialize was successfully run once.

Example

static const SYS_CMD_DESCRIPTOR cmdTbl[]= { {"command_as_typed_at_the_prompt", _Function_Name_That_Supports_The_Command, ": Helpful description of the command for the user"}, };

bool APP_AddCommandFunction() { if (!SYS_CMD_ADDGRP(cmdTbl, sizeof(cmdTbl)/sizeof(*cmdTbl), "Command Group Name", ": Command Group Description")) { return false; } return true; }

Function

bool SYS_CMD_ADDGRP(const SYS_CMD_DESCRIPTOR* pCmdTbl, int nCmds,

const char* groupName,

const char* menuStr)

SYS_CMD_MESSAGE Function

Outputs a message to the Command Processor System Service console.

File

sys_command.h

C

```
void SYS_CMD_MESSAGE(const char* message);
```

Returns

None.

Description

This function outputs a message to the Command Processor System Service console.

Remarks

None.

Preconditions

SYS_CMD_Initialize was successfully run once.

Function

void SYS_CMD_MESSAGE (const char* message)

SYS_CMD_PRINT Function

Outputs a printout to the Command Processor System Service console.

File

sys_command.h

C

```
void SYS_CMD_PRINT(const char * format, ...);
```

Returns

None.

Description

This function outputs a printout to the Command Processor System Service console.

Remarks

None.

Preconditions

SYS_CMD_Initialize was successfully run once.

Function

void SYS_CMD_PRINT(const char *format, ...)

SYS_CMD_READY_TO_READ Function

Allows upper layer application to confirm that the command module is ready to accept command input

File

sys_command.h

C

```
bool SYS_CMD_READY_TO_READ();
```

Returns

- true - Indicates command module is ready
- false - Indicates command module is not ready

Description

This function allows upper layer application to confirm that the command module is ready to accept command input

Remarks

None.

Preconditions

SYS_CMD_Initialize was successfully run once.

Function

bool SYS_CMD_READY_TO_READ(void)

SYS_CMD_READY_TO_WRITE Function

Allows upper layer application to confirm that the command module is ready to write output to the Console System Service.

File

sys_command.h

C

```
bool SYS_CMD_READY_TO_WRITE();
```

Returns

- true - Indicates command module is ready
- false - Indicates command module is not ready

Description

This function allows upper layer application to confirm that the command module is ready to write output to the Console System Service.

Remarks

None.

Preconditions

SYS_CMD_Initialize was successfully run once.

Function

bool SYS_CMD_READY_TO_WRITE(void)

SYS_CMD_Tasks Function

Maintains the Command Processor System Service's internal state machine.

File

sys_command.h

C

```
bool SYS_CMD_Tasks();
```

Returns

- true - Indicates success
- false - Indicates an error occurred

Description

This function is used to maintain the Command Processor System Service internal state machine.

Remarks

None.

Preconditions

SYS_CMD_Initialize was successfully run once.

Function

bool SYS_CMD_Tasks(void)

SYS_CMD_RegisterCallback Function

Registers a callback function with the command service that will be executed when the lower layer read or write queue is empty.

File

sys_command.h

C

```
void SYS_CMD_RegisterCallback(SYS_CMD_CallbackFunction cbFunc, SYS_CMD_EVENT event);
```

Returns

None.

Description

This function is used by an application to register a callback function with the command service. The callback function is called in response to an event. Separate callback functions are required for each event.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
cbFunc	The name of the callback function
event	Enumerated list of events that can trigger a callback

Function

void SYS_CMD_RegisterCallback(SYS_CMD_CallbackFunction cbFunc, SYS_CMD_EVENT event)

b) Command I/O Functions

SYS_CMDIO_GET_HANDLE Function

Gets the CMDIO Handle base via index.

File

sys_command.h

C

```
SYS_CMD_DEVICE_NODE* SYS_CMDIO_GET_HANDLE(short num);
```

Returns

- SYS_CMD_DEVICE_NODE Handle for the CMDIO - Indicates success
- NULL - Indicates not successful

Description

This function returns the handle for the CMDIO when supplied with an index.

Remarks

None.

Preconditions

SYS_CMD_Initialize was successfully run once.

Function

SYS_CMD_DEVICE_NODE* SYS_CMDIO_GET_HANDLE(short num)

SYS_CMDIO_ADD Function

Adds CMDIO parameters to the Command Processor System Service console.

File

sys_command.h

C

```
SYS_CMD_DEVICE_NODE* SYS_CMDIO_ADD(const SYS_CMD_API* opApi, const void* cmdIoParam, const
SYS_CMD_CONSOLE_IO_PARAM cmdIoType);
```

Returns

None.

Description

This function adds CMDIO Parameters to the Command Processor System Service console.

Remarks

None.

Preconditions

SYS_CMD_Initialize was successfully run once.

Function

SYS_CMD_DEVICE_NODE* SYS_CMDIO_ADD(const SYS_CMD_API* opApi, const void* cmdIoParam, const SYS_CMD_CONSOLE_IO_PARAM cmdIoType)

c) Data Types and Constants

_promptStr Macro

File

sys_command.h

C

```
#define _promptStr ">"              // prompt string
```

Description

prompt string

COMMAND_HISTORY_DEPTH Macro

Command Processor System Service Maximum Depth of Command History.

File

sys_command.h

C

```
#define COMMAND_HISTORY_DEPTH 3
```

Description

SYS CMD Processor Command History Depth

This macro defines the maximum depth of the command history.

Remarks

None.

ESC_SEQ_SIZE Macro

File

sys_command.h

C

```
#define ESC_SEQ_SIZE 2              // standard VT100 escape sequences
```

Description

standard VT100 escape sequences

LINE_TERM Macro

File

sys_command.h

C

```
#define LINE_TERM "\r\n"          // line terminator
```

Description

line terminator

MAX_CMD_ARGS Macro

Command Processor System Service Maximum Number of Argument definitions.

File

sys_command.h

C

```
#define MAX_CMD_ARGS 15
```

Description

SYS CMD Processor Maximum Number of Command Arguments

This macro defines the maximum number of arguments per command.

Remarks

None.

MAX_CMD_GROUP Macro

Command Processor System Service Maximum Number of Command Group definitions.

File

sys_command.h

C

```
#define MAX_CMD_GROUP 8
```

Description

SYS CMD Processor Maximum Number of Command Group

This macro defines the maximum number of command groups.

Remarks

None.

SYS_CMD_MAX_LENGTH Macro

Command Processor System Service Command Buffer Maximum Length definition.

File

sys_command.h

C

```
#define SYS_CMD_MAX_LENGTH 128
```

Description

SYS CMD Processor Buffer Maximum Length

This macro defines the maximum length of the command buffer.

Remarks

None.

SYS_CMD_READ_BUFFER_SIZE Macro

Command Processor System Service Read Buffer Size definition.

File

sys_command.h

C

```
#define SYS_CMD_READ_BUFFER_SIZE 128
```

Description

SYS CMD Processor Read Buffer Size

This macro defines the maximum size of the command buffer.

Remarks

None.

SYS_CMD_DEVICE_NODE Structure

Defines the data structure to store each command instance.

File

sys_command.h

C

```
struct SYS_CMD_DEVICE_NODE {
  char* cmdPnt;
  char* cmdEnd;
  char cmdBuff SYS_CMD_BUFFER_DMA_READY[SYS_CMD_MAX_LENGTH+1];
  const SYS_CMD_API* pCmdApi;
  const void* cmdIoParam;
  SYS_CMD_CONSOLE_IO_PARAM cmdIoType;
  struct SYS_CMD_DEVICE_NODE* next;
  SYS_CMD_STATE cmdState;
};
```

Members

Members	Description
const SYS_CMD_API* pCmdApi;	Cmd IO APIs
const void* cmdIoParam;	channel specific parameter

Description

SYS CMD Command Instance Node Structure

This data structure stores all the data relevant to a uniquely entered command instance. It is a node for a linked list structure to support the Command Processor System Service's command history feature

Remarks

None.

SYS_CMD_INIT Structure

Identifies the system command initialize structure.

File

sys_command.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
```

```
  uint8_t consoleCmdIOParam;
  SYS_CMD_CONSOLE_IO_PARAM cmdIoType;
} SYS_CMD_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization

Description

SYS CMD INIT structure

This structure identifies the system command initialize structure.

Remarks

None.

SYS_CMD_API Structure

Identifies the Command API structure.

File

sys_command.h

C

```
typedef struct {
  SYS_CMD_MSG_FNC msg;
  SYS_CMD_PRINT_FNC print;
  SYS_CMD_PUTC_FNC putc;
  SYS_CMD_DATA_RDY_FNC isRdy;
  SYS_CMD_GETC_FNC getc;
  SYS_CMD_READC_FNC readc;
} SYS_CMD_API;
```

Members

Members	Description
SYS_CMD_MSG_FNC msg;	Message function API
SYS_CMD_PRINT_FNC print;	Print function API
SYS_CMD_PUTC_FNC putc;	Put single char function API
SYS_CMD_DATA_RDY_FNC isRdy;	Data available API
SYS_CMD_GETC_FNC getc;	Get single data API
SYS_CMD_READC_FNC readc;	Read single data API

Description

SYS CMD API structure

This structure identifies the Command API structure.

Remarks

None.

SYS_CMD_DATA_RDY_FNC Type

Identifies a data available function API.

File

sys_command.h

C

```
typedef bool (* SYS_CMD_DATA_RDY_FNC)(const void* cmdIoParam);
```

Description

Ready Status Check function API. This handle identifies the interface structure of the data available function API within the Command IO encapsulation.

Remarks

None.

SYS_CMD_DESCRIPTOR Structure

File

sys_command.h

C

```
typedef struct {
  const char* cmdStr;
  SYS_CMD_FNC cmdFnc;
  const char* cmdDescr;
} SYS_CMD_DESCRIPTOR;
```

Members

Members	Description
const char* cmdStr;	string identifying the command
SYS_CMD_FNC cmdFnc;	function to execute for this command
const char* cmdDescr;	simple command description

Description

a simple command descriptor

SYS_CMD_DESCRIPTOR_TABLE Structure

File

sys_command.h

C

```
typedef struct {
  int nCmds;
  const SYS_CMD_DESCRIPTOR* pCmd;
  const char* cmdGroupName;
  const char* cmdMenuStr;
} SYS_CMD_DESCRIPTOR_TABLE;
```

Members

Members	Description
int nCmds;	number of commands available in the table
const SYS_CMD_DESCRIPTOR* pCmd;	pointer to an array of command descriptors
const char* cmdGroupName;	name identifying the commands
const char* cmdMenuStr;	help string

Description

table containing the supported commands

SYS_CMD_DEVICE_LIST Structure

Defines the list structure to store a list of command instances.

File

sys_command.h

C

```
typedef struct {
  int num;
  SYS_CMD_DEVICE_NODE* head;
  SYS_CMD_DEVICE_NODE* tail;
} SYS_CMD_DEVICE_LIST;
```

Description

SYS CMD Command List Structure

This data structure defines he linked list structure to support the Command Processor System Service's command history feature

Remarks

None.

SYS_CMD_FNC Type

Identifies the command process function API.

File

sys_command.h

C

```
typedef int (* SYS_CMD_FNC)(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char **argv);
```

Description

SYS CMD Command Function

Command Process Function API. This handle identifies the interface structure of the command process function API.

Remarks

None.

SYS_CMD_GETC_FNC Type

Identifies a get single character function API.

File

sys_command.h

C

```
typedef char (* SYS_CMD_GETC_FNC)(const void* cmdIoParam);
```

Description

Get Single Character function API. This handle identifies the interface structure of the get single character function API within the Command I/O encapsulation.

Remarks

None.

SYS_CMD_HANDLE Type

Identifies a particular Command I/O instance.

File

sys_command.h

C

```
typedef const void* SYS_CMD_HANDLE;
```

Description

Command I/O Handle. This event handle identifies a registered instance of a Command IO object. Every time the application that tries to access the parameters with respect to a particular event, this event handle is used to refer to that event.

Remarks

None.

SYS_CMD_INIT_DATA Structure

Defines the data structure to store each command.

File

sys_command.h

C

```
typedef struct {
  size_t bytesRead;
  int seqBytesRead;
  char seqBuff[ESC_SEQ_SIZE + 1];
  SYS_MODULE_INDEX moduleIndices[SYS_CMD_DEVICE_MAX_INSTANCES];
  int moduleInFd;
  int moduleOutFd;
} SYS_CMD_INIT_DATA;
```

Description

SYS CMD Command App Init Structure

This data structure stores all the data relevant to a uniquely entered command. It is a node for a linked list structure to support the command history functionality

Remarks

None.

SYS_CMD_MSG_FNC Type

Identifies a message function API.

File

sys_command.h

C

```
typedef void (* SYS_CMD_MSG_FNC)(const void* cmdIoParam, const char* str);
```

Description

Message function API. This handle identifies the interface structure of the message function API within the Command I/O encapsulation.

Remarks

None.

SYS_CMD_PRINT_FNC Type

Identifies a print function API.

File

sys_command.h

C

```
typedef void (* SYS_CMD_PRINT_FNC)(const void* cmdIoParam, const char* format, ...);
```

Description

Print function API. This handle identifies the interface structure of the print function API within the Command I/O encapsulation.

Remarks

None.

SYS_CMD_PUTC_FNC Type

Identifies a single character print function API.

File

sys_command.h

C

```
typedef void (* SYS_CMD_PUTC_FNC)(const void* cmdIoParam, char c);
```

Description

Single Character Print function API. This handle identifies the interface structure of single character print function API within the Command I/O encapsulation.

Remarks

None.

SYS_CMD_READC_FNC Type

Identifies a read single character function API.

File

sys_command.h

C

```
typedef size_t (* SYS_CMD_READC_FNC)(const void* cmdIoParam);
```

Description

Read Single Character function API

This handle identifies the interface structure of read single character function API within the Command I/O encapsulation.

Remarks

None.

SYS_CMD_STATE Enumeration

Defines the various states that can be achieved by a Command instance.

File

sys_command.h

C

```
typedef enum {
  SYS_CMD_STATE_DISABLE,
  SYS_CMD_STATE_SETUP_READ,
  SYS_CMD_STATE_WAIT_FOR_READ_DONE,
  SYS_CMD_STATE_PROCESS_FULL_READ
} SYS_CMD_STATE;
```

Description

SYS CMD State Machine States

This enumeration defines the various states that can be achieved by the command operation.

Remarks

None.

SYS_CMD_CONSOLE_IO_PARAM Enumeration

Defines whether the command module is single character or full command read

File

sys_command.h

C

```
typedef enum {
  SYS_CMD_SINGLE_CHARACTER_READ_CONSOLE_IO_PARAM = 0,
  SYS_CMD_FULL_COMMAND_READ_CONSOLE_IO_PARAM = 1,
  SYS_CMD_TELNET_COMMAND_READ_CONSOLE_IO_PARAM = 2
} SYS_CMD_CONSOLE_IO_PARAM;
```

Description

SYS CMD Console I/O Parameter

This enumeration defines whether the command module is single character or full command read.

Remarks

None.

SYS_CMD_BUFFER_DMA_READY Macro

File

sys_command.h

C

```
#define SYS_CMD_BUFFER_DMA_READY
```

Description

Define this for MX #define SYS_CMD_BUFFER_DMA_READY __attribute__((coherent)) __attribute__((aligned(4))) //Define this for MZ

SYS_CONSOLE_MESSAGE Macro

File

sys_console.h

C

```
#define SYS_CONSOLE_MESSAGE(message)
```

Description

ifdef SYS_CONSOLE_REMOVE_APIS

SYS_CONSOLE_PRINT Macro

File

sys_console.h

C

```
#define SYS_CONSOLE_PRINT(fmt, ...)
```

Description

This is macro SYS_CONSOLE_PRINT.

SYS_CMD_DEVICE_MAX_INSTANCES Macro

File

sys_command.h

C

```
#define SYS_CMD_DEVICE_MAX_INSTANCES 1
```

Description

This is macro SYS_CMD_DEVICE_MAX_INSTANCES.

SYS_CMD_MESSAGE Macro

File

sys_command.h

C

```
#define SYS_CMD_MESSAGE(message)
```

Description

This is macro SYS_CMD_MESSAGE.

SYS_CMD_PRINT Macro

File

sys_command.h

C

```
#define SYS_CMD_PRINT(fmt, ...)
```

Description

This is macro SYS_CMD_PRINT.

SYS_CMD_CallbackFunction Type

Command Callback Function Handle.

File

sys_command.h

C

```
typedef void (* SYS_CMD_CallbackFunction)(void *handle);
```

Description

Command Callback Function Handle.

Remarks

None.

SYS_CMD_EVENT Enumeration

Identifies the Command Event Types

File

sys_command.h

C

```
typedef enum {
  SYS_CMD_EVENT_WRITE_COMPLETE,
  SYS_CMD_EVENT_READ_COMPLETE
} SYS_CMD_EVENT;
```

Description

SYS CMD EVENT structure

This structure identifies the Command Event Types.

Remarks

None.

Files

Files

Name	Description
sys_command.h	Command Processor System Service interface definition.

Description

This section lists the source and header files used by the library.

sys_command.h

Command Processor System Service interface definition.

Enumerations

	Name	Description
	SYS_CMD_CONSOLE_IO_PARAM	Defines whether the command module is single character or full command read
	SYS_CMD_EVENT	Identifies the Command Event Types
	SYS_CMD_STATE	Defines the various states that can be achieved by a Command instance.

Functions

	Name	Description
	SYS_CMD_ADDGRP	Allows clients to add command process to the Command Processor System Service.
	SYS_CMD_DELETE	Removes CMDIO parameters from the Command Processor System Service console.
	SYS_CMD_Initialize	Initializes data for the instance of the Command Processor module.
	SYS_CMD_MESSAGE	Outputs a message to the Command Processor System Service console.
	SYS_CMD_PRINT	Outputs a printout to the Command Processor System Service console.
	SYS_CMD_READY_TO_READ	Allows upper layer application to confirm that the command module is ready to accept command input
	SYS_CMD_READY_TO_WRITE	Allows upper layer application to confirm that the command module is ready to write output to the Console System Service.
	SYS_CMD_RegisterCallback	Registers a callback function with the command service that will be executed when the lower layer read or write queue is empty.
	SYS_CMD_Tasks	Maintains the Command Processor System Service's internal state machine.
	SYS_CMDIO_ADD	Adds CMDIO parameters to the Command Processor System Service console.
	SYS_CMDIO_GET_HANDLE	Gets the CMDIO Handle base via index.

Macros

	Name	Description
	_promptStr	prompt string
	COMMAND_HISTORY_DEPTH	Command Processor System Service Maximum Depth of Command History.
	ESC_SEQ_SIZE	standard VT100 escape sequences
	LINE_TERM	line terminator
	MAX_CMD_ARGS	Command Processor System Service Maximum Number of Argument definitions.
	MAX_CMD_GROUP	Command Processor System Service Maximum Number of Command Group definitions.
	SYS_CMD_BUFFER_DMA_READY	Define this for MX #define SYS_CMD_BUFFER_DMA_READY __attribute__((coherent)) __attribute__((aligned(4))) //Define this for MZ
	SYS_CMD_DEVICE_MAX_INSTANCES	This is macro SYS_CMD_DEVICE_MAX_INSTANCES.
	SYS_CMD_MAX_LENGTH	Command Processor System Service Command Buffer Maximum Length definition.
	SYS_CMD_MESSAGE	This is macro SYS_CMD_MESSAGE.
	SYS_CMD_PRINT	This is macro SYS_CMD_PRINT.
	SYS_CMD_READ_BUFFER_SIZE	Command Processor System Service Read Buffer Size definition.
	SYS_DEBUG	This is macro SYS_DEBUG.
	SYS_ERROR	This is macro SYS_ERROR.
	SYS_ERROR_PRINT	This is macro SYS_ERROR_PRINT.

Structures

	Name	Description
	SYS_CMD_DEVICE_NODE	Defines the data structure to store each command instance.
	SYS_CMD_API	Identifies the Command API structure.
	SYS_CMD_DESCRIPTOR	a simple command descriptor
	SYS_CMD_DESCRIPTOR_TABLE	table containing the supported commands
	SYS_CMD_DEVICE_LIST	Defines the list structure to store a list of command instances.
	SYS_CMD_INIT	Identifies the system command initialize structure.
	SYS_CMD_INIT_DATA	Defines the data structure to store each command.

Types

	Name	Description
	SYS_CMD_CallbackFunction	Command Callback Function Handle.
	SYS_CMD_DATA_RDY_FNC	Identifies a data available function API.

	SYS_CMD_FNC	Identifies the command process function API.
	SYS_CMD_GETC_FNC	Identifies a get single character function API.
	SYS_CMD_HANDLE	Identifies a particular Command I/O instance.
	SYS_CMD_MSG_FNC	Identifies a message function API.
	SYS_CMD_PRINT_FNC	Identifies a print function API.
	SYS_CMD_PUTC_FNC	Identifies a single character print function API.
	SYS_CMD_READC_FNC	Identifies a read single character function API.

Description

Command Processor System Service Interface Definition

This file contains the interface definition for the Command Processor System Service. It provides a way to interact with the Command Processor subsystem to manage the ASCII command requests from the user supported by the system.

File Name

sys_command.h

Company

Microchip Technology Inc.

Console System Service Library

This section describes the Console System Service Library.

Introduction

The Console System Service routes data or message traffic between a console device and a middleware layer or application. The most common use of the Console Service is to route debug or error messages from a PIC32-based device to a terminal program on a host development system. When fully implemented, the Console Service will be capable of routing data from any supported console device to a variety of middleware layers.

Description

The Console System Service consists of a core layer and a console device layer. The core layer handles module initialization and system functions. It provides a common API for use by applications and middleware libraries to send and receive data to and from one or more console devices. The Console System core layer maintains the status and device descriptor information for each console instance that has been initialized. The maximum number of simultaneous console instances supported by the service is defined by a configuration option. The default is four.

The console device layer contains the functional implementation of the core layer APIs. Each console device may have a different implementation, but the behavior of the API should be uniform across different devices. The console device layer interacts with the peripheral device drivers to send and receive data to and from the hardware peripheral. The details of the data transaction are abstracted by the console device layer and the peripheral device driver.

Both blocking and non-blocking implementations are provided. For a blocking implementation, the API to transfer data to or from the console device will return after the data has been successfully transferred, or an error has occurred. Non-blocking implementations return immediately and the data transfer is controlled by a state machine implemented in the console device layer. The calling program is notified of the data transfer completion by a callback mechanism, or can optionally poll the status of the console device layer to see if it is busy or ready for more data.

The console device layer provides the ability to queue data buffers written to the console. The size of the queue is defined by a configuration option. Read data is queued internally in the console device layer depending on the console device. The size of this queue is also defined by a configuration option. Each console device has configuration options specific to that device. They are explained in their respective sections in this document.

Using the Library

This topic describes the basic architecture of the Console System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_console.h`

The interface to the Console System Service library is defined in the `sys_console.h` header file. This file is included by the `sys.h` file. Any C language source (`.c`) file that uses the Console System Service library should include `sys.h`.

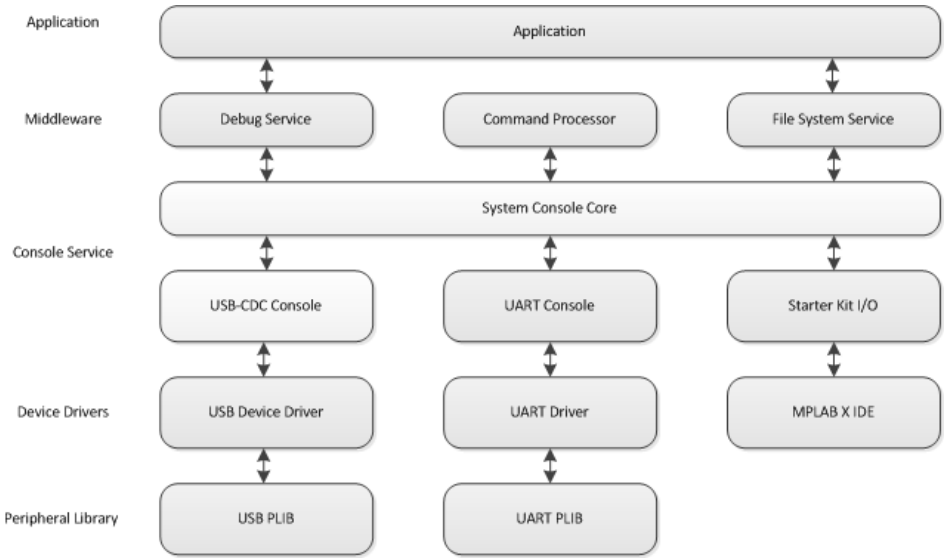
Please refer to the What is MPLAB Harmony? section for how the System Service interacts with the framework.

Abstraction Model

This library provides a set of functions that send and receive data to and from a console I/O device.

Description

Console Service Software Abstraction Block Diagram



The Console System Service is a middleware library that is part of a layered software architecture. The purpose of this software layering is to allow each module to provide a consistent interface to its client, thus making it easy for the developer to add and remove modules as needed. The console core layer provides a POSIX-like read/write API to applications and middleware libraries to send and receive data to and from the console device. Data and message buffering, along with the details of the data transfer are contained within the console device and device driver layers.

Additional APIs are provided to handle system functions, register a callback function for non-blocking implementations, and flush (reset) the internal queues in the event of an error condition.

Library Overview

Refer to the section System Services Introduction for how the system services operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the System Console module.

Library Interface Section	Description
System Functions	Provides system module interfaces, device initialization, tasks and status functions.
Core Functions	Reads or writes data to or from the console I/O device.

How the Library Works

The Console System Service is used to transfer data between an application or middleware layer and a console device.

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

System Functions

This topic describes Console System Service Library system functions.

Description

The Console System Service must be initialized before use. Initialization opens a specific console module instance and initializes the internal data structures for that instance. The initialization routine is provided a pointer to a SYS_CONSOLE_INIT data structure, which contains a pointer to a device descriptor for a particular console device.

USB CDC Example

```
SYS_CONSOLE_INIT consInit =
{
    .moduleInit = {0},
    .consDevDesc = &consUsbCdcDevDesc,
};
```

UART Example

```
SYS_CONSOLE_INIT consInit =
{
    .moduleInit = {0},
    .consDevDesc = &consUsartDevDesc,
};
```

The console device descriptor contains information specific to the device associated with that console instance, including pointers to device-specific implementations of the Console System service API. The device descriptor for each console device is declared in the console device implementation. A pointer to the descriptor must be provided to the initialization routine.

Status

The SYS_CONSOLE_Status function returns the status of the console device module. The internal status of the console device module is updated as the module cycles through its internal state machine. For example, the console device module may need to wait for its corresponding device driver to initialize before moving to an initialized state. Module status is converted from an internal data type to the generic Harmony SYS_STATUS data type.

Tasks

The SYS_CONSOLE_Tasks function advances the state machine of the console device module. A pointer to the console device module's Tasks() routine is provided as part of the console device descriptor during initialization.

Core Functions

This topic describes core Console System Service functions.

Description

Read

The SYS_CONSOLE_Read function reads data from a console device into a read buffer. In a blocking implementation, the API will return the number of successfully read bytes when the read operation completes. In a non-blocking implementation, the API will return immediately with number of successfully queued bytes. Completion of the read triggers a callback mechanism in which a user-defined function is executed. A pointer to the read buffer for that transaction is provided as an argument to the callback function to allow console users to distinguish between read events.

For single read events, the user may also poll the status of the console device module.

Write

The SYS_CONSOLE_Write function writes data from a write buffer or string to a console device. In a blocking implementation, the API will return the number of successfully written bytes when the write operation completes. In a non-blocking implementation, the API will return immediately with the number of successfully queued bytes. Completion of the write triggers a callback mechanism in which a user-defined function is executed. A pointer to the write buffer for that transaction is provided as an argument to the callback function to allow console users to distinguish between write events.

For single write events, the user may also poll the status of the console device module.

Register Callback

Users wishing to take advantage of the callback mechanism must register a callback function with the console service. A separate function must be registered for each event capable of triggering a callback. If no function is registered, the operation will complete normally without triggering any callback function.

Flush

Error conditions may be triggered by a queue overflow or data transfer errors. An error condition may present itself as an unexpected return value from a read or write call, or an error status returned by the SYS_CONSOLE_Status function. The SYS_CONSOLE_Flush function is provided to allow the user to terminate any pending transfers and reset the read/write queues internal to the console device.

Console Device Modules

Each supported console device has its own implementation of the Console System service APIs. Function pointers to the console device implementations are provided in the console device descriptor registered to each console instance. This allows the user to utilize the same API for all console devices, making application code more portable across different devices.

Description

Each supported console device has its own implementation of the Console System service APIs. Function pointers to the console device implementations are provided in the console device descriptor registered to each console instance. This allows the user to utilize the same API for all console devices, making application code more portable across different devices.

Application I/O (AppIO)

The AppIO provides a convenient user I/O interface to applications running on a PIC32 using MPLAB X IDE.

Description

Read

The read function accepts user input through the Input Format Field in the PIC AppIO window of MPLAX X IDE. It is the responsibility of the application programmer to properly allocate the read buffer.

Write

The write function sends output to the Output Format Window in MPLAB X IDE, a line at a time. It is the responsibility of the application programmer to properly allocate the write buffer.

USB-CDC Console Device

The USB-CDC Console Device provides a convenient user I/O interface to applications running on a PIC32 with USB support.

Description

In this implementation, the PIC32 acts as a USB device, communicating with a host PC running a terminal emulation program such as Tera Term. This provides a convenient interface for error or debug messages, or a means of communicating with a user interactively. The application does not use the USB-CDC APIs directly, instead all interaction is through the Console System service APIs.

The USB-CDC Console Device supports only non-blocking operation.

Read

While the user may request multiple bytes of data per read, a typical terminal emulator program sends only one byte per USB transaction. When multiple bytes are requested, the USB-CDC Console Device layer will queue individual byte reads until all bytes are read. The size of the read queue is a configuration option. The bytes are sent to the user-supplied read buffer as they are read, and the read callback is triggered when all bytes have been read. Only one read operation can be active at a time.

If the user requests more data bytes than the read queue will hold, the read function will queue and return the size of the read queue. In this scenario, the user can either do multiple reads until all the data is read, or configure the service to use a larger read queue.

The read function accepts a pointer to a read buffer as one of the arguments. It is the responsibility of the application programmer to properly allocate the read buffer.

Write

The USB-CDC console device is capable of writing multiple data bytes per USB transaction, and queuing multiple write transactions. The size of the write queue is a configuration option. If the user tries to queue more write transactions than the write queue can accommodate, the write function will return -1 and set the module status to SYS_STATUS_ERROR.

The write function accepts a pointer to a write buffer as one of the arguments. It is the responsibility of the application programmer to properly allocate the write buffer.

State Machine

The USB-CDC Console Device layer implements a state machine that is advanced by each call to the tasks routine. The Console System tasks routine is called periodically by the SYS_Tasks routine, which is called in a loop by main(). The state machine manages initialization of the Console Device, schedules reads and writes, and dispatches callback functions.

Register Callback

The callback registration mechanism associates a user function with a read or write completion event. The callback function includes an argument containing a pointer to the read or write buffer associated with the transfer that initiated the callback. Separate callback functions must be registered for read and write events. If there is no callback function registered, the event will complete as normal, with no callback executed.

Flush

The flush routine in the USB-CDC implementation clears any error conditions, terminates any read or write transactions, and resets the read and write queues. A flush operation is typically triggered in response to an error condition.

UART Console Device

The UART Console Device provides a convenient user I/O interface to applications running on a PIC32 with UART support.

Description

Read

While the user may request multiple bytes of data per read, a typical terminal emulator program sends only one byte per transaction. When multiple bytes are requested, the UART Console Device layer will queue individual byte reads until all bytes are read. The size of the read queue is a configuration option. The bytes are sent to the user-supplied read buffer as they are read, and the read callback is triggered when all bytes have been read. Only one read operation can be active at a time.

If the user requests more data bytes than the read queue will hold, the read function will queue and return the size of the read queue. In this scenario, the user can either do multiple reads until all the data is read, or configure the service to use a larger read queue.

The read function accepts a pointer to a read buffer as one of the arguments. It is the responsibility of the application programmer to properly allocate the read buffer.

Write

The UART console device is capable of writing multiple data bytes per transaction, and queuing multiple write transactions. The size of the write queue is a configuration option. If the user tries to queue more write transactions than the write queue can accommodate, the write function will return -1 and set the module status to SYS_STATUS_ERROR.

The write function accepts a pointer to a write buffer as one of the arguments. It is the responsibility of the application programmer to properly allocate the write buffer.

State Machine

The UART Console Device layer implements a state machine that is advanced by each call to the tasks routine. The Console System tasks routine is called periodically by the SYS_Tasks routine, which is called in a loop by main(). The state machine manages initialization of the Console Device, schedules reads and writes, and dispatches callback functions.

Register Callback

The callback registration mechanism associates a user function with a read or write completion event. The callback function includes an argument containing a pointer to the read or write buffer associated with the transfer that initiated the callback. Separate callback functions must be registered for read and write events. If there is no callback function registered, the event will complete as normal, with no callback executed.

Flush

The flush routine in the UART implementation clears any error conditions, terminates any read or write transactions, and resets the read and write queues. A flush operation is typically triggered in response to an error condition.

Configuring the Library

The configuration of the Console System Service is based on the file `system_config.h`.

This header file contains the configuration selection for the Console System Service. Based on the selections made, the Console System Service may support the selected features. These configuration settings will apply to all instances of the Console System Service.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

The Console System Service supports these configuration options:

```
/* Maximum number of simultaneous console instances */
#define CONSOLE_DEVICE_MAX_INSTANCES            1
```

Additional configuration options may be required for the selected Console Device.

Command AppIO Console Device Configuration Options

This topic provided configuration option examples for the Command AppIO Console Device.

Description

```
/* maximum number of console instances */
#define SYS_CONSOLE_DEVICE_MAX_INSTANCES        1

/* number of messages that may be queued up for read */
#define SYS_CONSOLE_APPIO_RD_QUEUE_DEPTH        8

/* number of messages that may be queued up for write */
#define SYS_CONSOLE_APPIO_WR_QUEUE_DEPTH        128
```

USB-CDC Console Device Configuration Options

This topic provided configuration option examples for the USB-CDC Console Device.

Description

USB-CDC Queues

```
/* Number of characters that may be queued for a single read */
#define CONSOLE_USB_CDC_RD_QUEUE_DEPTH          64

/* Number of messages that may be queued for write */
#define CONSOLE_USB_CDC_WR_QUEUE_DEPTH          64
```

USB Controller Driver

```
/* Enables Device Support */
#define DRV_USB_DEVICE_SUPPORT      true
```

```
/* Disables host support */
#define DRV_USB_HOST_SUPPORT        false

/* Provides 3 endpoints*/
#define DRV_USB_ENDPOINTS_NUMBER    3

/* Only one instance of the USB Peripheral*/
#define DRV_USB_INSTANCES_NUMBER    1

/* Enables interrupt mode */
#define DRV_USB_INTERRUPT_MODE      true
```

USB Device Layer

```
/* Maximum device layer instances */
#define USB_DEVICE_MAX_INSTANCES         1

/* Maximum clients for Device Layer */
#define USB_DEVICE_MAX_CLIENTS             1

/* Maximum function drivers allowed per instance of the USB device layer */
#define USB_DEVICE_FUNCTION_DRIVERS_NUMBER      1

/* EP0 size in bytes */
#define USB_DEVICE_EP0_BUFFER_SIZE          64
```

USB-CDC Function Driver

```
/* Maximum instances of CDC function driver */
#define USB_DEVICE_CDC_INSTANCES_NUMBER   1

/* CDC Transfer Queue Size for both read and
 * write. Applicable to all instances of the
 * function driver */
#define USB_DEVICE_CDC_WRITE_QUEUE_DEPTH         4
#define USB_DEVICE_CDC_READ_QUEUE_DEPTH          1
#define USB_DEVICE_CDC_SSN_QUEUE_DEPTH           1
#define USB_DEVICE_CDC_QUEUE_DEPTH_COMBINED
             (USB_DEVICE_CDC_WRITE_QUEUE_DEPTH +
              USB_DEVICE_CDC_READ_QUEUE_DEPTH +
              USB_DEVICE_CDC_SSN_QUEUE_DEPTH)
```

UART Console Device Configuration Options

This topic provided configuration option examples for the UART Console Device.

Description

```
/* Maximum number of simultaneous console instances */
#define CONSOLE_DEVICE_MAX_INSTANCES         1

/* Number of characters that may be queued for a single read */
#define CONSOLE_UART_RD_QUEUE_DEPTH          1

/* Number of messages that may be queued for write */
#define CONSOLE_UART_WR_QUEUE_DEPTH          64

// *****************************************************************************
// *****************************************************************************
// Section: UART Driver Configuration
// *****************************************************************************
// *****************************************************************************

#define DRV_USART_INTERRUPT_MODE                true

#define DRV_USART_INSTANCES_NUMBER              1

#define DRV_USART_CLIENTS_NUMBER                1
```

```
#define DRV_USART_BUFFER_QUEUE_SUPPORT          true

#define DRV_USART_BYTE_MODEL_SUPPORT            false

#define DRV_USART_READ_WRITE_MODEL_SUPPORT      false

#define DRV_USART_WRITE_QUEUE_DEPTH             CONSOLE_UART_RD_QUEUE_DEPTH

#define DRV_USART_READ_QUEUE_DEPTH              CONSOLE_UART_WR_QUEUE_DEPTH

#define DRV_USART_SSN_QUEUE_DEPTH               1

#define DRV_USART_QUEUE_DEPTH_COMBINED          (DRV_USART_WRITE_QUEUE_DEPTH +
                                                 DRV_USART_READ_QUEUE_DEPTH +
                                                 DRV_USART_SSN_QUEUE_DEPTH)
```

Building the Library

This section lists the files that are available in the Console System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/console`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
sys_console.h	Console System Service Library API header file.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/sys_console.c	Console System Service Library implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
/src/sys_console_usb_cdc.c	Required for USB-CDC console device.
/src/sys_console_uart.c	Required for UART console device.
/src/sys_console_local.h	Data structures and definitions used by the Console System Service, but not exposed to clients.
/src/sys_console_usb_cdc_local.h	Data structures and definitions used by the USB-CDC console device, but not exposed to clients.

Module Dependencies

The Console System Service is dependent upon the following modules:

- Device Control System Service Library
- USART Driver Library (if UART)
- USB Driver Library (if USB-CDC)

Library Interface

a) System Functions

	Name	Description
	SYS_CONSOLE_Initialize	Initializes data for the instance of the Console module and opens the specific module instance. **Implementation:** Static/Dynamic
	SYS_CONSOLE_Reinitialize	Reinitializes and refreshes the data structure for the instance of the Console module. **Implementation:** Static/Dynamic
	SYS_CONSOLE_Deinitialize	Deinitializes the specific module instance of the Console module. **Implementation:** Static/Dynamic
	SYS_CONSOLE_Tasks	Maintains the console's state machine. **Implementation:** Dynamic
	SYS_CONSOLE_Status	Returns status of the specific instance of the Console module. **Implementation:** Dynamic

b) Core Functions

	Name	Description
	SYS_CONSOLE_Read	Reads data from the console device. **Implementation:** Static/Dynamic
	SYS_CONSOLE_Write	Writes data to the console device. **Implementation:** Static/Dynamic
	SYS_CONSOLE_RegisterCallback	Registers a callback function with the console service that will be executed when the read or write queue is emptied. **Implementation:** Dynamic
	SYS_CONSOLE_Flush	Flushes the read and write queues and resets an overflow error for the console. **Implementation:** Static/Dynamic

c) Data Types and Constants

	Name	Description
	SYS_CONSOLE_OBJECT_INSTANCE	System Console object instance structure.
	SYS_CONSOLE_DEVICE	Lists the available console devices.
	SYS_CONSOLE_DEV_DESC	This is type SYS_CONSOLE_DEV_DESC.
	SYS_CONSOLE_INIT	Identifies the system console initialize structure.
	STDERR_FILENO	This is macro STDERR_FILENO.
	STDIN_FILENO	These are in unistd.h
	STDOUT_FILENO	This is macro STDOUT_FILENO.
	SYS_CONSOLE_INDEX_0	Console System Service index definitions.
	SYS_CONSOLE_INDEX_1	This is macro SYS_CONSOLE_INDEX_1.
	SYS_CONSOLE_INDEX_2	This is macro SYS_CONSOLE_INDEX_2.
	SYS_CONSOLE_INDEX_3	This is macro SYS_CONSOLE_INDEX_3.
	CONSOLE_DEVICE_HANDLE_INVALID	This is macro CONSOLE_DEVICE_HANDLE_INVALID.
	SYS_CONSOLE_BUFFER_DMA_READY	This is added here to prevent built errors, just in-case MHC was never run

Description

This section describes the APIs of the Console System Service Library.

Refer to each section for a detailed description.

a) System Functions

SYS_CONSOLE_Initialize Function

Initializes data for the instance of the Console module and opens the specific module instance.

Implementation: Static/Dynamic

File

sys_console.h

C

```
SYS_MODULE_OBJ SYS_CONSOLE_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to an object. Otherwise, it returns SYS_MODULE_OBJ_INVALID. The returned object must be passed as argument to SYS_CONSOLE_Reinitialize, SYS_CONSOLE_Deinitialize, SYS_CONSOLE_Tasks and SYS_CONSOLE_Status routines.

Description

This function initializes the Console module, and selects the I/O device to be used. It also initializes any internal data structures.

Remarks

This routine should only be called once during system initialization unless SYS_Console_Deinitialize is first called to deinitialize the device instance before reinitializing it. If the system was already initialized it safely returns without causing any disturbance.

Preconditions

None.

Example

```
SYS_MODULE_OBJ  objectHandle;

// Populate the console initialization structure
SYS_CONSOLE_INIT consInit =
{
    .moduleInit = {0},
    .consDevDesc = &consUsbCdcDevDesc,
};

objectHandle = SYS_Console_Initialize (SYS_CONSOLE_INDEX_0, (SYS_MODULE_INIT*)&consInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Index for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the Console System service. This pointer may be null if no data is required because static overrides have been provided.

Function

SYS_MODULE_OBJ SYS_CONSOLE_Initialize(const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init)

SYS_CONSOLE_Reinitialize Function

Reinitializes and refreshes the data structure for the instance of the Console module.

Implementation: Static/Dynamic

File

sys_console.h

C

```
bool SYS_CONSOLE_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function reinitializes and refreshes the data structure for the instance of the Console module using the supplied data.

Remarks

This operation uses the same initialization data structure as the SYS_CONSOLE_Initialize operation. This function can be called multiple times to reinitialize the module.

Preconditions

The SYS_CONSOLE_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ  objectHandle;

// Populate the console initialization structure
SYS_CONSOLE_INIT consInit =
{
    .moduleInit = {0},
    .consDevDesc = &consUsbCdcDevDesc,
};

SYS_CONSOLE_Reinitialize (objectHandle, (SYS_MODULE_INIT*)&consInit);
```

Parameters

Parameters	Description
object	Identifies the SYS CONSOLE Object returned by the Initialize interface
init	Pointer to the data structure containing any data necessary to initialize the hardware

Function

void SYS_CONSOLE_Reinitialize(SYS_MODULE_OBJ object,
const SYS_MODULE_INIT * const init)

SYS_CONSOLE_Deinitialize Function

Deinitializes the specific module instance of the Console module.

Implementation: Static/Dynamic

File

sys_console.h

C

```
bool SYS_CONSOLE_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function deinitializes the specific module instance disabling its operation (and any hardware for driver modules). Resets all of the internal data structures and fields for the specified instance to the default settings.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Preconditions

The SYS_CONSOLE_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from SYS_CONSOLE_Initialize

SYS_CONSOLE_Deinitialize (object);
```

Parameters

Parameters	Description
object	SYS CONSOLE object handle, returned from SYS_CONSOLE_Initialize

Function

void SYS_CONSOLE_Deinitialize(SYS_MODULE_OBJ object)

SYS_CONSOLE_Tasks Function

Maintains the console's state machine.

Implementation: Dynamic

File

sys_console.h

C

```
void SYS_CONSOLE_Tasks(SYS_MODULE_OBJ object);
```

Returns

None

Description

This function is used to maintain the Console System Service internal state machine and implement its ISR for interrupt-driven implementations.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

Preconditions

The SYS_CONSOLE_Initialize function must have been called for the specified CONSOLE driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from SYS_CONSOLE_Initialize

while (true)
{
    SYS_CONSOLE_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	SYS CONSOLE object handle, returned from SYS_CONSOLE_Initialize

Function

void SYS_CONSOLE_Tasks(SYS_MODULE_OBJ object)

SYS_CONSOLE_Status Function

Returns status of the specific instance of the Console module.

Implementation: Dynamic

File

sys_console.h

C

```
SYS_STATUS SYS_CONSOLE_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that the driver is busy with a previous system level operation and cannot start another. Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

- SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another.
- SYS_STATUS_ERROR - Indicates that the driver is in an error state. Any value less than SYS_STATUS_ERROR is also an error state.
- SYS_MODULE_DEINITIALIZED - Indicates that the driver has been deinitialized. This value is less than SYS_STATUS_ERROR.

Description

This function returns the status of the specific module instance disabling its operation (and any hardware for driver modules).

Remarks

None.

Preconditions

The SYS_CONSOLE_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from SYS_CONSOLE_Initialize
SYS_STATUS          consStatus;

consStatus = SYS_CONSOLE_Status (object);
if (SYS_STATUS_ERROR >= consStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	SYS CONSOLE object handle, returned from SYS_CONSOLE_Initialize

Function

SYS_STATUS SYS_CONSOLE_Status(SYS_MODULE_OBJ object)

b) Core Functions

SYS_CONSOLE_Read Function

Reads data from the console device.

Implementation: Static/Dynamic

File

sys_console.h

C

```
ssize_t SYS_CONSOLE_Read(const SYS_MODULE_INDEX index, int fd, void * buf, size_t count);
```

Returns

Number of bytes actually read.

Description

This function reads the data from the console device.

Remarks

None.

Preconditions

None.

Example

```
ssize_t nr;
char myBuffer[MY_BUFFER_SIZE];
nr = SYS_CONSOLE_Read( SYS_CONSOLE_INDEX_0, STDIN_FILENO, myBuffer, MY_BUFFER_SIZE );
if (nr != MY_BUFFER_SIZE)
{
    // handle error
```

```
}
```

Parameters

Parameters	Description
index	Console instance index
fd	I/O stream handle. Maintained for backward compatibility. NULL value can be passed as a parameter.
buf	Buffer to hold the read data.
count	Number of bytes to read.

Function

int SYS_CONSOLE_Read(int handle, char *buffer, int len)

SYS_CONSOLE_Write Function

Writes data to the console device.

Implementation: Static/Dynamic

File

sys_console.h

C

```
ssize_t SYS_CONSOLE_Write(const SYS_MODULE_INDEX index, int fd, const char * buf, size_t count);
```

Returns

Number of bytes written or -1 if an error occurred.

Description

This function writes data to the console device.

Remarks

None.

Preconditions

None.

Example

```
ssize_t nr;
char myBuffer[] = "message";
nr = SYS_CONSOLE_Write( SYS_CONSOLE_INDEX_0, STDOUT_FILENO, myBuffer, strlen(myBuffer) );
if (nr != strlen(myBuffer))
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Console instance index
fd	I/O stream handle. Maintained for backward compatibility. NULL value can be passed as a parameter.
buf	Buffer holding the data to be written.
count	Number of bytes to write.

Function

ssize_t SYS_CONSOLE_Write(const SYS_MODULE_INDEX index, int fd, const char *buf, size_t count)

SYS_CONSOLE_RegisterCallback Function

Registers a callback function with the console service that will be executed when the read or write queue is emptied.

Implementation: Dynamic

File

sys_console.h

C

```
void SYS_CONSOLE_RegisterCallback(const SYS_MODULE_INDEX index, consoleCallbackFunction cbFunc,
SYS_CONSOLE_EVENT event);
```

Returns

None.

Description

This function is used by an application to register a callback function with the console service. The callback function is called in response to an event. Separate callback functions are required for each event.

Remarks

None.

Preconditions

None.

Example

```
SYS_CONSOLE_RegisterCallback(SYS_CONSOLE_INDEX_0, APP_ReadComplete,
                             SYS_CONSOLE_EVENT_READ_COMPLETE);
SYS_CONSOLE_RegisterCallback(SYS_CONSOLE_INDEX_0, APP_WriteComplete,
                             SYS_CONSOLE_EVENT_WRITE_COMPLETE);
```

Parameters

Parameters	Description
index	Console instance index
consCallbackFunc	The name of the callback function
event	Enumerated list of events that can trigger a callback

Function

void SYS_CONSOLE_RegisterCallback(const SYS_MODULE_INDEX index,

consoleCallbackFunction cbFunc, SYS_CONSOLE_EVENT event)

SYS_CONSOLE_Flush Function

Flushes the read and write queues and resets an overflow error for the console.

Implementation: Static/Dynamic

File

sys_console.h

C

```
void SYS_CONSOLE_Flush(const SYS_MODULE_INDEX index);
```

Returns

None.

Description

This function flushes the read and write queues and resets an overflow error for the console.

Remarks

None.

Preconditions

None.

Example

```
SYS_CONSOLE_Flush(SYS_CONSOLE_INDEX_0);
```

Parameters

Parameters	Description
index	Console instance index

Function

void SYS_CONSOLE_Flush(const SYS_MODULE_INDEX index)

c) Data Types and Constants

SYS_CONSOLE_OBJECT_INSTANCE Structure

System Console object instance structure.

File

sys_console.h

C

```
typedef struct {
  SYS_STATUS consoleDeviceInstanceStatus;
  SYS_CONSOLE_DEV_DESC * consoleInstanceDevDesc;
  CONSOLE_DEVICE_HANDLE consoleDevHandle;
} SYS_CONSOLE_OBJECT_INSTANCE;
```

Members

Members	Description
SYS_STATUS consoleDeviceInstanceStatus;	State of this instance

Description

SYS CONSOLE OBJECT INSTANCE structure

This data type defines the System Console object instance.

Remarks

None.

SYS_CONSOLE_DEVICE Enumeration

Lists the available console devices.

File

sys_console.h

C

```
typedef enum {
  SYS_CONSOLE_DEV_USART,
  SYS_CONSOLE_DEV_USB_CDC,
  SYS_CONSOLE_DEV_APPIO,
  SYS_CONSOLE_NR_DEVS
} SYS_CONSOLE_DEVICE;
```

Description

Console device enumeration

This enumeration lists all of the available console devices. A console device is a physical peripheral used by the console service to send and receive data.

Remarks

None.

SYS_CONSOLE_DEV_DESC Structure

File

sys_console.h

C

```
typedef struct {
  SYS_CONSOLE_DEVICE consoleDevice;
  DRV_IO_INTENT intent;
  char (* sysConsoleReadC)(int fd);
  ssize_t (* sysConsoleRead)(int fd, void *buf, size_t count);
  ssize_t (* sysConsoleWrite)(int fd, const void *buf, size_t count);
  void (* sysConsoleRegisterCallback)(consoleCallbackFunction cbFunc, SYS_CONSOLE_EVENT event);
  void (* sysConsoleTasks)(SYS_MODULE_OBJ object);
  SYS_CONSOLE_STATUS (* sysConsoleStatus)(void);
  void (* sysConsoleFlush)(void);
} SYS_CONSOLE_DEV_DESC;
```

Description

This is type SYS_CONSOLE_DEV_DESC.

SYS_CONSOLE_INIT Structure

Identifies the system console initialize structure.

File

sys_console.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_CONSOLE_DEV_DESC * consDevDesc;
} SYS_CONSOLE_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization

Description

SYS Console Initialize structure

This structure identifies the system console initialize structure.

Remarks

None.

STDERR_FILENO Macro

File

sys_console.h

C

```
#define STDERR_FILENO 2
```

Description

This is macro STDERR_FILENO.

STDIN_FILENO Macro

File

sys_console.h

C

```
#define STDIN_FILENO 0
```

Description

These are in unistd.h

STDOUT_FILENO Macro

File

sys_console.h

C

```
#define STDOUT_FILENO 1
```

Description

This is macro STDOUT_FILENO.

SYS_CONSOLE_INDEX_0 Macro

Console System Service index definitions.

File

sys_console.h

C

```
#define SYS_CONSOLE_INDEX_0 0
```

Description

SYS Console Module Index Numbers

These constants provide Console System Service index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

SYS_CONSOLE_INDEX_1 Macro

File

sys_console.h

C

```
#define SYS_CONSOLE_INDEX_1 1
```

Description

This is macro SYS_CONSOLE_INDEX_1.

SYS_CONSOLE_INDEX_2 Macro

File

sys_console.h

C

```
#define SYS_CONSOLE_INDEX_2 2
```

Description

This is macro SYS_CONSOLE_INDEX_2.

SYS_CONSOLE_INDEX_3 Macro

File

sys_console.h

C

```
#define SYS_CONSOLE_INDEX_3 3
```

Description

This is macro SYS_CONSOLE_INDEX_3.

CONSOLE_DEVICE_HANDLE_INVALID Macro

File

sys_console.h

C

```
#define CONSOLE_DEVICE_HANDLE_INVALID ((CONSOLE_DEVICE_HANDLE)(-1))
```

Description

This is macro CONSOLE_DEVICE_HANDLE_INVALID.

SYS_CONSOLE_BUFFER_DMA_READY Macro

File

sys_console.h

C

```
#define SYS_CONSOLE_BUFFER_DMA_READY
```

Description

This is added here to prevent built errors, just in-case MHC was never run

Files

Files

Name	Description
sys_console.h	Console System Service interface definitions.

Description

This section lists the source and header files used by the library.

sys_console.h

Console System Service interface definitions.

Enumerations

	Name	Description
	SYS_CONSOLE_DEVICE	Lists the available console devices.

Functions

	Name	Description
	SYS_CONSOLE_Deinitialize	Deinitializes the specific module instance of the Console module. **Implementation:** Static/Dynamic
	SYS_CONSOLE_Flush	Flushes the read and write queues and resets an overflow error for the console. **Implementation:** Static/Dynamic

	SYS_CONSOLE_Initialize	Initializes data for the instance of the Console module and opens the specific module instance. **Implementation:** Static/Dynamic
	SYS_CONSOLE_Read	Reads data from the console device. **Implementation:** Static/Dynamic
	SYS_CONSOLE_RegisterCallback	Registers a callback function with the console service that will be executed when the read or write queue is emptied. **Implementation:** Dynamic
	SYS_CONSOLE_Reinitialize	Reinitializes and refreshes the data structure for the instance of the Console module. **Implementation:** Static/Dynamic
	SYS_CONSOLE_Status	Returns status of the specific instance of the Console module. **Implementation:** Dynamic
	SYS_CONSOLE_Tasks	Maintains the console's state machine. **Implementation:** Dynamic
	SYS_CONSOLE_Write	Writes data to the console device. **Implementation:** Static/Dynamic

Macros

	Name	Description
	CONSOLE_DEVICE_HANDLE_INVALID	This is macro CONSOLE_DEVICE_HANDLE_INVALID.
	STDERR_FILENO	This is macro STDERR_FILENO.
	STDIN_FILENO	These are in unistd.h
	STDOUT_FILENO	This is macro STDOUT_FILENO.
	SYS_CONSOLE_BUFFER_DMA_READY	This is added here to prevent built errors, just in-case MHC was never run
	SYS_CONSOLE_INDEX_0	Console System Service index definitions.
	SYS_CONSOLE_INDEX_1	This is macro SYS_CONSOLE_INDEX_1.
	SYS_CONSOLE_INDEX_2	This is macro SYS_CONSOLE_INDEX_2.
	SYS_CONSOLE_INDEX_3	This is macro SYS_CONSOLE_INDEX_3.
	SYS_CONSOLE_MESSAGE	ifdef SYS_CONSOLE_REMOVE_APIS
	SYS_CONSOLE_PRINT	This is macro SYS_CONSOLE_PRINT.

Structures

	Name	Description
	SYS_CONSOLE_DEV_DESC	This is type SYS_CONSOLE_DEV_DESC.
	SYS_CONSOLE_INIT	Identifies the system console initialize structure.
	SYS_CONSOLE_OBJECT_INSTANCE	System Console object instance structure.

Description

Console System Service Interface Definition

This file contains the interface definition for the Console system service. It provides a way to interact with the Console subsystem to manage the timing requests supported by the system.

File Name

sys_console.h

Company

Microchip Technology Inc.

Debug System Service Library

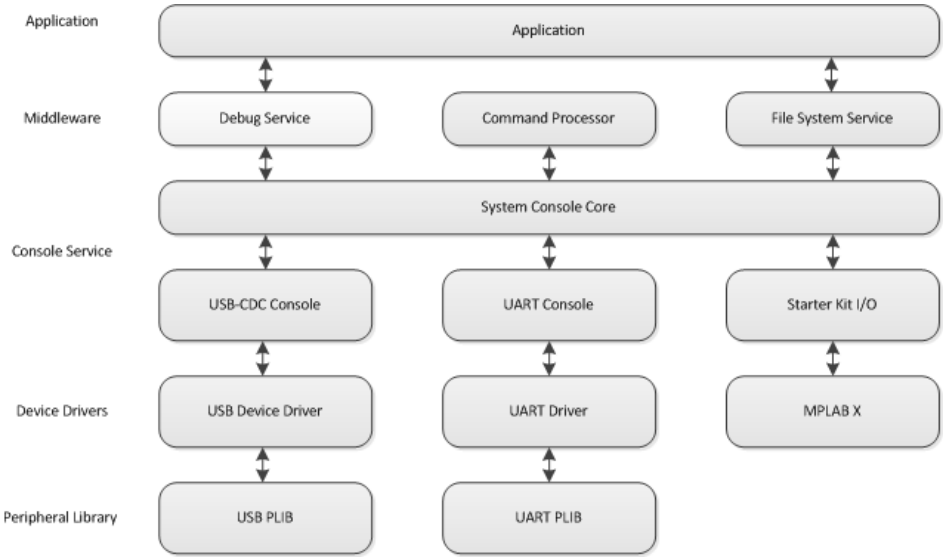
This section describes the Debug System Service Library.

Introduction

This library implements the Debug System Service. The Debug System Service provides a convenient mechanism for the application developer to send formatted or unformatted messages to a system console. The console may direct the messages to a terminal program, file, or any other supported console device. The Debug System Service maintains a global error level, which may be set during initialization and changed dynamically at run-time. Both formatted and unformatted messages can be output or not, depending on the current global error level.

Description

The Debug System Service provides a set of functions that allow the developer to output debug and/or error messages based on a global error level. The messages are sent to the System Console Service, where they are routed to a console device.



Using the Library

This topic describes the basic architecture of the Debug System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_debug.h`

The interface to the Debug System Service library is defined in the `sys_debug.h` header file. This file is included by the `sys.h` file. Any C language source (`.c`) file that uses the Debug System Service Library should include `sys.h`.

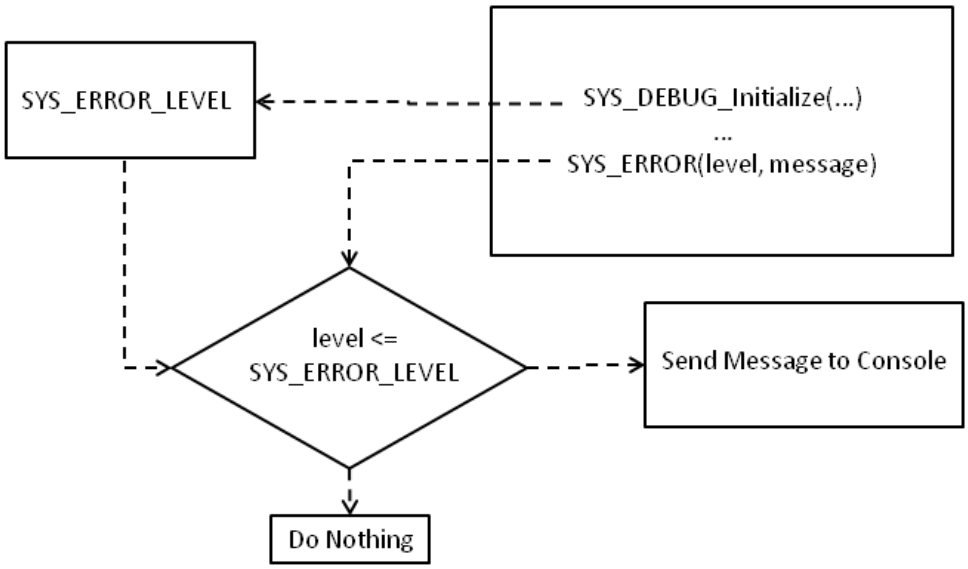
Please refer to the What is MPLAB Harmony? section for how the Debug System Service interacts with the framework.

Abstraction Model

This library provides a set of functions that allow the developer to quickly and easily provide debug and error messaging during and after program development.

Description

Debug Service Software Abstraction Block Diagram



When the Debug System Service is initialized, it sets the global system error level to the specified level. This level determines the threshold at which debug and error messages are sent to the console. This allows different debug and error reporting verbosity depending on the needs of the developer. The Debug System Service also provides APIs to dynamically set the error level during program execution. This allows the developer to increase or decrease debug verbosity to specific areas of interest within the program.

Library Overview

Refer to the section System Services Introduction for how the system services operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Debug System Service module.

Library Interface Section	Description
System Functions	Initializes the Debug System Service and sets the global system error level.
Changing System Error Level Functions	Reads or sets the global system debug level.

How the Library Works

The Debug System Service library can be used by a device driver, middleware layer, or application to report error conditions and output debug messages to a console device during program operation.

Debug System Initialization

The Debug System Service library must be initialized by calling the SYS_DEBUG_Initialize function. Initialization opens a specific debug module instance and initializes the internal data structures for that instance.

Example: Debug System Service Initialization

```
SYS_DEBUG_INIT debugInit =
{
    .moduleInit = {0},
    .errorLevel = SYS_ERROR_DEBUG
};
```

Debug Messages and Error Reporting

The following macros are available to output debug and error messages:

- SYS_MESSAGE(message) prints a simple message string to the output device
- SYS_DEBUG(level, message) prints a debug message to the console device if the global error level is equal to or lower than that specified by the "level" argument. The default implementation of SYS_DEBUG(level, message) resolves to nothing by the preprocessor. This is to allow the developer to leave debug messaging in released code without impacting code size or performance. Typically, the developer would define SYS_DEBUG to map to _SYS_DEBUG in a configuration file for debug builds.

  ```
  #define SYS_DEBUG(level,message) _SYS_DEBUG(level,message)
  ```
- SYS_PRINT(fmt, ...) prints formatted messages to the console. The message formatting is the same as printf.
- SYS_ERROR(level, fmt, ...) prints formatted messages to the console if the global error level is equal to or lower than that specified by the "level" argument. The message formatting is the same as printf.

Changing the System Error Level

Two functions are provided to manipulate the global system error level at runtime. This may be useful when you want to increase the debug verbosity for a particular section of code, but not for the entire program.

- SYS_ERROR_LevelGet returns the current global system error level
- SYS_ERROR_LevelSet(level) sets the current global system error level

Example: Changing the system error level

```
SYS_ERROR_LEVEL lvl;
/* Get current level */
lvl = SYS_ERROR_LevelGet();
/* Set new level */
SYS_ERROR_LevelSet(SYS_ERROR_DEBUG);
// Do something
SYS_DEBUG("Okay until here\n");
/* Restore previous level */
SYS_ERROR_LevelSet(lvl);
```

Configuring the Library

The configuration of the Debug System Service is based on the file `system_config.h`.

This header file contains the configuration selection for the Debug System Service. Based on the selections made, the Debug System Service may support the selected features. These configuration settings will apply to all instances of the Debug System Service.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

The Debug System Service supports these configuration options:

```
/* Enable debug messaging */
#define SYS_DEBUG(level,message) _SYS_DEBUG(level,message)

/* Circular buffer for formatted output */
#define DEBUG_PRINT_BUFFER_SIZE                 512
```

Building the Library

This section lists the files that are available in the Debug System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/debug`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
sys_debug.h	Debug System Service Library API header file.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/sys_debug.c	Debug System Service Library implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
/src/sys_debug_local.h	Data structures used by the Debug System Service Library, but not exposed to clients.

Module Dependencies

The Debug System Service is dependent upon the following modules:

- Device Control System Service Library
- Console System Service Library

Library Interface

a) System Functions

	Name	Description
	SYS_DEBUG_Deinitialize	Deinitializes the specific module instance of the Debug module.
	SYS_DEBUG_Initialize	Initializes the global error level and specific module instance.
	SYS_DEBUG_Reinitialize	Reinitializes and refreshes the data structure for the instance of the Debug module.
	SYS_DEBUG_Status	Returns status of the specific instance of the debug service module.

	SYS_DEBUG_Tasks	Maintains the debug module's state machine.
	SYS_DEBUG_Message	Prints a message to the console regardless of the system error level.
	SYS_DEBUG_Print	Formats and prints a message with a variable number of arguments to the console regardless of the system error level.

b) Changing System Error Level Functions

	Name	Description
	SYS_DEBUG_ErrorLevelGet	Returns the global system Error reporting level.
	SYS_DEBUG_ErrorLevelSet	Sets the global system error reporting level.

c) Data Types and Constants

	Name	Description
	SYS_DEBUG_INIT	Defines the data required to initialize the debug system service.
	SYS_ERROR_LEVEL	System error message priority levels.
	SYS_DEBUG	This is macro SYS_DEBUG.
	SYS_DEBUG_INDEX_0	Debug System Service index.
	SYS_MESSAGE	Prints a message to the console regardless of the system error level.
	SYS_ERROR	This is macro SYS_ERROR.
	SYS_PRINT	Formats and prints an error message with a variable number of arguments regardless of the system error level.
	SYS_DEBUG_MESSAGE	Prints a debug message if the system error level is defined defined at or lower than the level specified.
	SYS_DEBUG_PRINT	Formats and prints an error message if the system error level is defined at or lower than the level specified.
	SYS_DEBUG_BreakPoint	Inserts a software breakpoint instruction when building in Debug mode.
	SYS_ERROR_PRINT	This is macro SYS_ERROR_PRINT.
	_SYS_DEBUG_MESSAGE	Prints a debug message if the specified level is at or below the global system error level.
	_SYS_DEBUG_PRINT	Formats and prints a debug message if the specified level is at or below the global system error level.

Description

This section describes the APIs of the Debug System Service Library.

Refer to each section for a detailed description.

a) System Functions

SYS_DEBUG_Deinitialize Function

Deinitializes the specific module instance of the Debug module.

File

sys_debug.h

C

```
void SYS_DEBUG_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the specific module instance disabling its operation (and any hardware for driver modules). Resets all of the internal data structures and fields for the specified instance to the default settings.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Preconditions

The SYS_DEBUG_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from SYS_DEBUG_Initialize
SYS_DEBUG_Deinitialize (object);
```

Parameters

Parameters	Description
object	SYS DEBUG object handle, returned from SYS_DEBUG_Initialize

Function

void SYS_DEBUG_Deinitialize(SYS_MODULE_OBJ object)

SYS_DEBUG_Initialize Function

Initializes the global error level and specific module instance.

File

sys_debug.h

C

```
SYS_MODULE_OBJ SYS_DEBUG_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to an object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the global error level. It also initializes any internal system debug module data structures.

Remarks

This routine should only be called once during system initialization. If the system was already initialized it safely returns without causing any disturbance.

Preconditions

None.

Example

```
SYS_MODULE_OBJ objectHandle;
SYS_DEBUG_INIT debugInit =
{
    .moduleInit = {0},
    .errorLevel = SYS_ERROR_DEBUG,
};

objectHandle = SYS_Debug_Initialize (SYS_DEBUG_CONSOLE, (SYS_MODULE_INIT*)&debugInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Index for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the debug service. This pointer may be null if no data is required because static overrides have been provided.

Function

SYS_MODULE_OBJ SYS_DEBUG_Initialize(const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init)

SYS_DEBUG_Reinitialize Function

Reinitializes and refreshes the data structure for the instance of the Debug module.

File

sys_debug.h

C

```
void SYS_DEBUG_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None

Description

This function reinitializes and refreshes the data structure for the instance of the Debug module using the supplied data.

Remarks

This operation uses the same initialization data structure as the SYS_DEBUG_Initialize operation. This function can be called multiple times to reinitialize the module.

Preconditions

The SYS_DEBUG_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ  objectHandle;

// Populate the console initialization structure
SYS_DEBUG_INIT dbgInit =
{
    .moduleInit = {0},
    .errorLevel = SYS_ERROR_DEBUG,
};

SYS_DEBUG_Reinitialize (objectHandle, (SYS_MODULE_INIT*)&dbgInit);
```

Parameters

Parameters	Description
object	Identifies the SYS DEBUG Object returned by the Initialize interface
init	Pointer to the data structure containing any data necessary to initialize the hardware

Function

void SYS_DEBUG_Reinitialize(SYS_MODULE_OBJ object,
const SYS_MODULE_INIT * const init)

SYS_DEBUG_Status Function

Returns status of the specific instance of the debug service module.

File

sys_debug.h

C

```
SYS_STATUS SYS_DEBUG_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that the module is running and ready to service requests. Any value greater than SYS_STATUS_READY is also a normal running state in which the module is ready to accept new operations.
- SYS_STATUS_BUSY - Indicates that the module is busy with a previous system level operation and cannot start another.
- SYS_STATUS_ERROR - Indicates that the module is in an error state. Any value less than SYS_STATUS_ERROR is also an error state.
- SYS_STATUS_UNINITIALIZED - Indicates that the module has not been initialized or has been deinitialized.

Description

This function returns the status of the specific debug service module instance.

Remarks

None.

Preconditions

The SYS_CONSOLE_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from SYS_CONSOLE_Initialize
SYS_STATUS          debugStatus;

debugStatus = SYS_DEBUG_Status (object);
if (SYS_STATUS_ERROR >= debugStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Debug module object handle, returned from SYS_DEBUG_Initialize

Function

SYS_STATUS SYS_DEBUG_Status(SYS_MODULE_OBJ object)

SYS_DEBUG_Tasks Function

Maintains the debug module's state machine.

File

sys_debug.h

C

```
void SYS_DEBUG_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the debug module's internal state machine and implement its ISR for interrupt-driven implementations.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

Preconditions

The SYS_DEBUG_Initialize function must have been called for the specified CONSOLE driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from SYS_DEBUG_Initialize

while (true)
{
    SYS_DEBUG_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	SYS DEBUG object handle, returned from SYS_DEBUG_Initialize

Function

void SYS_DEBUG_Tasks(SYS_MODULE_OBJ object)

SYS_DEBUG_Message Function

Prints a message to the console regardless of the system error level.

File

sys_debug.h

C

```
void SYS_DEBUG_Message(const char * message);
```

Returns

None.

Description

This function prints a message to the console regardless of the system error level. It can be used as an implementation of the SYS_MESSAGE and SYS_DEBUG_MESSAGE macros.

Remarks

Do not call this function directly. Call the SYS_MESSAGE or SYS_DEBUG_MESSAGE macros instead.

The default SYS_MESSAGE and SYS_DEBUG_MESSAGE macro definitions remove the messages and message function calls from the source code. To access and utilize the messages, define the SYS_DEBUG_USE_CONSOLE macro or override the definitions of the individual macros.

Preconditions

SYS_DEBUG_Initialize must have returned a valid object handle and the SYS_DEBUG_Tasks function must be called by the system to complete the message request.

Example

```
// In system_config.h:
#define SYS_MESSAGE(message)  SYS_DEBUG_Message(message)

// In source (.c) files:
SYS_MESSAGE("My Messagenr");
```

Parameters

Parameters	Description
message	Pointer to a message string to be displayed.

Function

SYS_DEBUG_Message(const char* message);

SYS_DEBUG_Print Function

Formats and prints a message with a variable number of arguments to the console regardless of the system error level.

File

sys_debug.h

C

```
void SYS_DEBUG_Print(const char * format, ...);
```

Returns

None.

Description

This function formats and prints a message with a variable number of arguments to the console regardless of the system error level. It can be used to implement the SYS_PRINT and SYS_DEBUG_PRINT macros.

Remarks

The format string and arguments follow the printf convention.

Do not call this function directly. Call the SYS_PRINT or SYS_DEBUG_PRINT macros instead.

The default SYS_PRINT and SYS_DEBUG_PRINT macro definitions remove the messages and message function calls. To access and utilize the messages, define the SYS_DEBUG_USE_CONSOLE macro or override the definitions of the individual macros.

Preconditions

SYS_DEBUG_Initialize must have returned a valid object handle and the SYS_DEBUG_Tasks function must be called by the system to complete the message request.

Example

```
// In system_config.h
#define SYS_PRINT(format, ...)  SYS_DEBUG_Print(format, ##__VA_ARG__)

// In source code
int result;

result = SomeOperation();
if (result > MAX_VALUE)
{
    SYS_PRINT("Result of %d exceeds max valuern", result);
}
```

Parameters

Parameters	Description
format	Pointer to a buffer containing the format string for the message to be displayed.
...	Zero or more optional parameters to be formated as defined by the format string.

Function

SYS_DEBUG_Print(const char* format, ...)

b) Changing System Error Level Functions

SYS_DEBUG_ErrorLevelGet Function

Returns the global system Error reporting level.

File

sys_debug.h

C

```
SYS_ERROR_LEVEL SYS_DEBUG_ErrorLevelGet();
```

Returns

The global System Error Level.

Description

This function returns the global System Error reporting level.

Remarks

None.

Preconditions

SYS_DEBUG_Initialize must have returned a valid object handle.

Example

```
SYS_ERROR_LEVEL level;

level = SYS_DEBUG_ErrorLevelGet();
```

Function

SYS_ERROR_LEVEL SYS_DEBUG_ErrorLevelGet(void);

SYS_DEBUG_ErrorLevelSet Function

Sets the global system error reporting level.

File

sys_debug.h

C

```
void SYS_DEBUG_ErrorLevelSet(SYS_ERROR_LEVEL level);
```

Returns

None.

Description

This function sets the global system error reporting level.

Remarks

None.

Preconditions

SYS_DEBUG_Initialize must have returned a valid object handle.

Example

```
SYS_DEBUG_ErrorLevelSet(SYS_ERROR_WARNING);
```

Parameters

Parameters	Description
level	The desired system error level.

Function

void SYS_DEBUG_ErrorLevelSet(SYS_ERROR_LEVEL level);

c) Data Types and Constants

SYS_DEBUG_INIT Structure

Defines the data required to initialize the debug system service.

File

sys_debug.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_ERROR_LEVEL errorLevel;
  SYS_MODULE_INDEX consoleIndex;
} SYS_DEBUG_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_ERROR_LEVEL errorLevel;	Initial system error level setting.
SYS_MODULE_INDEX consoleIndex;	Console index to receive debug messages

Description

SYS Debug Initialize structure

This structure defines the data required to initialize the debug system service.

Remarks

None.

SYS_ERROR_LEVEL Enumeration

System error message priority levels.

File

sys_debug.h

C

```
typedef enum {
  SYS_ERROR_FATAL = 0,
  SYS_ERROR_ERROR = 1,
  SYS_ERROR_WARNING = 2,
  SYS_ERROR_INFO = 3,
  SYS_ERROR_DEBUG = 4
} SYS_ERROR_LEVEL;
```

Members

Members	Description
SYS_ERROR_FATAL = 0	Errors that have the potential to cause a system crash.
SYS_ERROR_ERROR = 1	Errors that have the potential to cause incorrect behavior.
SYS_ERROR_WARNING = 2	Warnings about potentially unexpected behavior or side effects.
SYS_ERROR_INFO = 3	Information helpful to understanding potential errors and warnings.
SYS_ERROR_DEBUG = 4	Verbose information helpful during debugging and testing.

Description

SYS_ERROR_LEVEL enumeration

This enumeration defines the supported system error message priority values.

Remarks

Used by debug message macros to compare individual message priority against a global system-wide error message priority level to determine if an individual message should be displayed.

SYS_DEBUG Macro

File

sys_command.h

C

```
#define SYS_DEBUG(level,message) SYS_DEBUG_MESSAGE(level,message)
```

Description

This is macro SYS_DEBUG.

SYS_DEBUG_INDEX_0 Macro

Debug System Service index.

File

sys_debug.h

C

```
#define SYS_DEBUG_INDEX_0 0
```

Description

SYS Debug Module Index Number

This constant defines a symbolic name for the debug system service index.

Remarks

There can only be a single debug system service instance in the system.

SYS_MESSAGE Macro

Prints a message to the console regardless of the system error level.

File

sys_debug.h

C

```
#define SYS_MESSAGE(message)
```

Returns

None.

Description

Macro: SYS_MESSAGE(const char* message)

This macro is used to print a message to the console regardless of the system error level. It can be mapped to any desired implementation.

Remarks

By default, this macro is defined as nothing, effectively removing all code generated by calls to it. To process SYS_MESSAGE calls, this macro must be defined in a way that maps calls to it to the desired implementation (see example, above).

This macro can be mapped to the system console service (along with other system debug macros) by defining SYS_DEBUG_USE_CONSOLE in the system configuration (system_config.h) instead of defining it individually.

Preconditions

If mapped to the _SYS_MESSAGE function, then the system debug service must be initialized and running.

Example

```
// In system_config.h:
#define SYS_MESSAGE(message)  _SYS_MESSAGE(message)

// In source (.c) files:
SYS_MESSAGE("My Messagenr");
```

Parameters

Parameters	Description
message	Pointer to a buffer containing the message string to be displayed.

SYS_ERROR Macro

File

sys_command.h

C

```
#define SYS_ERROR(level,fmt, ...) SYS_ERROR_PRINT(level, fmt, ##__VA_ARGS__)
```

Description

This is macro SYS_ERROR.

SYS_PRINT Macro

Formats and prints an error message with a variable number of arguments regardless of the system error level.

File

sys_debug.h

C

```
#define SYS_PRINT(fmt, ...)
```

Returns

None.

Description

This function formats and prints an error message with a variable number of if the system error level is defined at or lower than the level specified.

Remarks

The format string and arguments follow the printf convention. This function is called by the macros SYS_PRINT and SYS_ERROR_PRINT

By default, this macro is defined as nothing, effectively removing all code generated by calls to it. To process SYS_PRINT calls, this macro must be defined in a way that maps calls to it to the desired implementation (see example, above).

This macro can be mapped to the system console service (along with other system debug macros) by defining SYS_DEBUG_USE_CONSOLE in the system configuration (system_config.h) instead of defining it individually.

Preconditions

SYSTEM_CURRENT_ERROR_LEVEL must be defined.

Example

```
// In system_config.h
#define SYS_PRINT(level, format, ...)  _SYS_DEBUG_PRINT(level, format, ##__VA_ARG__)

// In source code
int result;

result = SomeOperation();
if (result > MAX_VALUE)
{
    SYS_PRINT("Result of %d exceeds max valuern", result);
    // Take appropriate action
}
```

Parameters

Parameters	Description
format	Pointer to a buffer containing the format string for the message to be displayed.
...	Zero or more optional parameters to be formated as defined by the format string.

Function

SYS_PRINT(const char* format, ...);

SYS_DEBUG_MESSAGE Macro

Prints a debug message if the system error level is defined defined at or lower than the level specified.

File

sys_debug.h

C

```
#define SYS_DEBUG_MESSAGE(level,message)
```

Returns

None.

Description

This function prints a debug message if the system error level is defined at or lower than the level specified.

Remarks

By default, this macro is defined as nothing, effectively removing all code generated by calls to it. To process SYS_DEBUG_MESSAGE calls, this macro must be defined in a way that maps calls to it to the desired implementation (see example, above).

This macro can be mapped to the system console service (along with other system debug macros) by defining SYS_DEBUG_USE_CONSOLE in the system configuration (system_config.h) instead of defining it individually.

Preconditions

SYSTEM_CURRENT_ERROR_LEVEL must be defined as SYS_ERROR_DEBUG.

Example

```
#define SYS_DEBUG_MESSAGE   _SYS_DEBUG_MESSAGE
SYS_ERROR_LevelSet(SYS_ERROR_DEBUG);
SYS_DEBUG_MESSAGE("System Debug Message nr");
```

Parameters

Parameters	Description
level	The current error level threshold for displaying the message.
message	Pointer to a buffer containing the message to be displayed.

Function

SYS_DEBUG_MESSAGE(const char* message)

SYS_DEBUG_PRINT Macro

Formats and prints an error message if the system error level is defined at or lower than the level specified.

File

sys_debug.h

C

```
#define SYS_DEBUG_PRINT(level, fmt, ...)
```

Returns

None.

Description

Macro: SYS_DEBUG_PRINT(SYS_ERROR_LEVEL level, const char* format, ...)

This macro formats and prints an error message if the system error level is defined at or lower than the level specified.o

Remarks

The format string and arguments follow the printf convention. This function is called by the macros SYS_PRINT and SYS_DEBUG_PRINT.

By default, this macro is defined as nothing, effectively removing all code generated by calls to it. To process SYS_MESSAGE calls, this macro must be defined in a way that maps calls to it to the desired implementation (see example, above).

This macro can be mapped to the system console service (along with other system debug macros) by defining SYS_DEBUG_USE_CONSOLE in the system configuration (system_config.h) instead of defining it individually.

Preconditions

SYS_DEBUG_Initialize must have returned a valid object handle.

Example

```
// In system_config.h
#define SYS_DEBUG_PRINT(level, format, ...)  _SYS_DEBUG_PRINT(level, format, ##__VA_ARG__)

// In source code
int result;

result = SomeOperation();
if (result > MAX_VALUE)
{
    SYS_DEBUG_PRINT(SYS_ERROR_WARNING, "Result of %d exceeds max valuern", result);
    // Take appropriate action
}
```

Parameters

Parameters	Description
level	The current error level threshold for displaying the message.
format	Pointer to a buffer containing the format string for the message to be displayed.
...	Zero or more optional parameters to be formated as defined by the format string.

SYS_DEBUG_BreakPoint Macro

Inserts a software breakpoint instruction when building in Debug mode.

File

sys_debug.h

C

```
#define SYS_DEBUG_BreakPoint
```

Returns

None.

Description

Macro: SYS_DEBUG_BreakPoint(void)

This macro inserts a software breakpoint instruction when building in Debug mode.

Remarks

Compiles out if not built for debugging.

Preconditions

None.

Example

```
if (myDebugTestFailed)
{
    SYS_DEBUG_BreakPoint();
}
```

SYS_ERROR_PRINT Macro

File

sys_command.h

C

```
#define SYS_ERROR_PRINT(level, fmt, ...) do { if((level) <= gblErrLvl) SYS_CMD_PRINT(fmt, ##__VA_ARGS__); }
while (0)
```

Description

This is macro SYS_ERROR_PRINT.

_SYS_DEBUG_MESSAGE Macro

Prints a debug message if the specified level is at or below the global system error level.

File

sys_debug.h

C

```
#define _SYS_DEBUG_MESSAGE(level, message) do { if((level) <= SYS_DEBUG_ErrorLevelGet())
SYS_DEBUG_Message(message); }while(0)
```

Returns

None.

Description

Macro: _SYS_DEBUG_MESSAGE(SYS_ERROR_LEVEL level, const char* message)

This macro prints a debug message if the specified level is at or below the global error level. It can be used to implement the SYS_DEBUG_MESSAGE macro.

Remarks

Do not call this macro directly. Call the SYS_DEBUG_MESSAGE macro instead.

The default SYS_DEBUG_MESSAGE macro definition removes the message and function call from the source code. To access and utilize the message, define the SYS_DEBUG_USE_CONSOLE macro or override the definition of the SYS_DEBUG_MESSAGE macro.

Preconditions

SYS_DEBUG_Initialize must have returned a valid object handle.

Example

```
// In system_config.h
#define SYS_DEBUG_MESSAGE(level,message)  _SYS_DEBUG_MESSAGE(level,message)

// In library source (.c) code.
SYS_DEBUG_MESSAGE(SYS_ERROR_WARNING, "My debug warning messagenr");
```

Parameters

Parameters	Description
level	The current error level threshold for displaying the message.
message	Pointer to a buffer containing the message to be displayed.

_SYS_DEBUG_PRINT Macro

Formats and prints a debug message if the specified level is at or below the global system error level.

File

sys_debug.h

C

```
#define _SYS_DEBUG_PRINT(level, format, ...) do { if((level) <= SYS_DEBUG_ErrorLevelGet())
SYS_DEBUG_Print(format, ##__VA_ARGS__); } while (0)
```

Returns

None.

Description

Macro: _SYS_DEBUG_PRINT(SYS_ERROR_LEVEL level, const char* format, ...)

This function formats and prints a debug message if the specified level is at or below the global system error level. It can be used to implement the SYS_DEBUG_PRINT macro.

Remarks

Do not call this macro directly. Call the SYS_DEBUG_PRINT macro instead.

The default SYS_DEBUG_PRINT macro definition removes the message and function call from the source code. To access and utilize the message, define the SYS_DEBUG_USE_CONSOLE macro or override the definition of the SYS_DEBUG_PRINT macro.

Preconditions

SYS_DEBUG_Initialize must have returned a valid object handle.

Example

```
// In system_config.h
#define SYS_DEBUG_PRINT(level, format, ...)  _SYS_DEBUG_PRINT(level, format, ##__VA_ARG__)

// In source code
int result;

result = SomeOperation();
if (result > MAX_VALUE)
{
    SYS_DEBUG_PRINT(SYS_ERROR_WARNING, "Result of %d exceeds max valuern", result);
    // Take appropriate action
}
```

Parameters

Parameters	Description
level	The current error level threshold for displaying the message.
format	Pointer to a buffer containing the format string for the message to be displayed.
...	Zero or more optional parameters to be formated as defined by the format string.

Files

Files

Name	Description
sys_debug.h	Defines the common debug definitions and interfaces used by MPLAB Harmony libraries to report errors and debug information to the user.

Description

This section lists the source and header files used by the library.

sys_debug.h

Defines the common debug definitions and interfaces used by MPLAB Harmony libraries to report errors and debug information to the user.

Enumerations

	Name	Description
	SYS_ERROR_LEVEL	System error message priority levels.

Functions

	Name	Description
	SYS_DEBUG_Deinitialize	Deinitializes the specific module instance of the Debug module.
	SYS_DEBUG_ErrorLevelGet	Returns the global system Error reporting level.
	SYS_DEBUG_ErrorLevelSet	Sets the global system error reporting level.
	SYS_DEBUG_Initialize	Initializes the global error level and specific module instance.
	SYS_DEBUG_Message	Prints a message to the console regardless of the system error level.
	SYS_DEBUG_Print	Formats and prints a message with a variable number of arguments to the console regardless of the system error level.
	SYS_DEBUG_Reinitialize	Reinitializes and refreshes the data structure for the instance of the Debug module.
	SYS_DEBUG_Status	Returns status of the specific instance of the debug service module.
	SYS_DEBUG_Tasks	Maintains the debug module's state machine.

Macros

	Name	Description
	_SYS_DEBUG_MESSAGE	Prints a debug message if the specified level is at or below the global system error level.
	_SYS_DEBUG_PRINT	Formats and prints a debug message if the specified level is at or below the global system error level.
	SYS_DEBUG_BreakPoint	Inserts a software breakpoint instruction when building in Debug mode.
	SYS_DEBUG_BUFFER_DMA_READY	This should be defined in system_config.h. It is added here as a build safe-guard.
	SYS_DEBUG_INDEX_0	Debug System Service index.
	SYS_DEBUG_MESSAGE	Prints a debug message if the system error level is defined defined at or lower than the level specified.
	SYS_DEBUG_PRINT	Formats and prints an error message if the system error level is defined at or lower than the level specified.
	SYS_MESSAGE	Prints a message to the console regardless of the system error level.
	SYS_PRINT	Formats and prints an error message with a variable number of arguments regardless of the system error level.

Structures

	Name	Description
	SYS_DEBUG_INIT	Defines the data required to initialize the debug system service.

Description

Debug System Services Library Header

This header file defines the common debug definitions and interface macros (summary below) and prototypes used by MPLAB Harmony libraries to report errors and debug information to the user.

File Name

sys_debug.h

Company

Microchip Technology Inc.

Device Control System Service Library

This section describes the Device Control System Service Library.

Introduction

This library provides a low-level abstraction of the Device Control System Service Library that is available on the Microchip family of PIC32 microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The Device Control System Service provides the developer with simple APIs to configure and control device-specific functions. These functions may involve one or more peripherals, or no peripherals at all.

Using the Library

This topic describes the basic architecture of the Device Control System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_devcon.h`

The interface to the Device Control System Service library is defined in the `sys_devcon.h` header file.

Please refer to the What is MPLAB Harmony? section for how the Device Control System Service interacts with the framework.

Abstraction Model

This topic provides a description of the software abstraction for the Device Control System Service.

Description

The Device Control System Service is a collection of device-specific operations that may not be tied to any specific peripheral, multiple peripherals, or may not involve any peripherals at all. This service provides functions to optimize system performance, perform cache management, and do other system-level tasks that may be needed by multiple modules.

Library Overview

Refer to the section System Services Introduction for how the system services operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Device Control module.

Library Interface Section	Description
System Interaction Functions	Provides system module APIs. Device initialization, deinitialization, reinitialization and status functions.
Core Functionality Functions	Device-specific configuration and control functions.
Cache Functions	Configuration and control functions specific to L1 Cache.
Data Types and Constants	Device-specific enumerations, structures, and constants.

How the Library Works

System Interaction

Initialization and Reinitialization

Initialization of the Device Control System Service initializes the status of the module and sets the state of the internal state machine.

The Device Control Status and Tasks routines are provided as stubs for future enhancement. They are not currently required for this system service.

Core Functionality

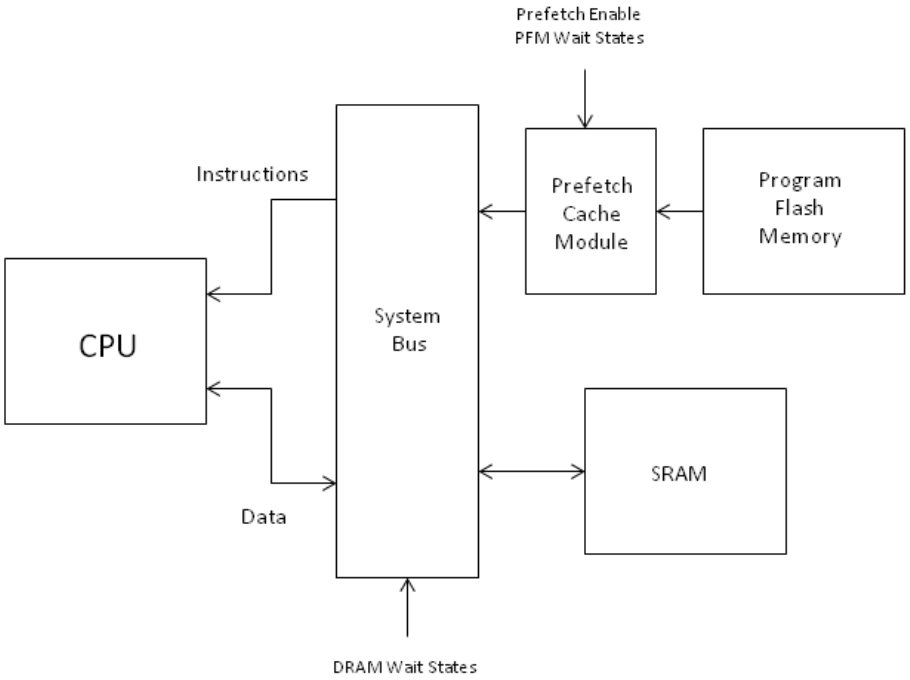
System Performance Tuning

When any PIC32 device is powered up, the Program Flash Memory and Data RAM memory wait states are not set to optimum values. By setting these wait states to the minimum required for the programmed clock speed, enabling the prefetch cache (if available), performance can be optimized. This service provides a single API that performs this optimization across all PIC32 devices.

The SYS_DEVCON_PerformanceConfig function takes the current system clock frequency as an argument and uses this value to set the program Flash wait states to the optimum value. It enables the Prefetch Cache module on both PIC32MX and PIC32MZ devices, and sets the SRAM Wait state for PIC32MX devices.

```
SYS_DEVCON_PerformanceConfig( unsigned int sysclk )
```

Performance Tuning Block Diagram



System Locking and Unlocking

Many operations within the PIC32 device require the system to be unlocked before the operation is performed. This is a safety feature which prevents errant code from altering critical settings by mistake. The unlock sequence consists of writing a series of "magic numbers" to a specific register in sequence. The critical operation is then performed, and the system is relocked by writing to the same register. This service provides convenient APIs that work across all PIC32 devices to perform this lock/unlock sequence.

```
SYS_DEVCON_SystemUnlock();
// critical operation
SYS_DEVCON_SystemLock();
```

Note: For the operations that require a lock/unlock sequence, refer to the specific device data sheet to determine the requirement for your device.

L1 Cache

This topic provides a description of the management functions for the Level 1 (L1) Cache that is available in PIC32MZ family devices.

Description

The Device Control System Service contains an interface for controlling the L1 Cache in the PIC32MZ family of devices.

The L1 cache is divided into two parts, a Data Cache (D-cache) and an Instruction Cache (I-cache). These blocks of high-speed memory both serve to compensate for the lengthy access time of main memory, by fetching instructions and data for the CPU ahead of time. The CPU can then access the information directly through the cache in a single clock cycle, rather than having to wait multiple clock cycles for accesses to main memory. The L1 cache provides a drastic increase in performance, but the user must be aware of hazards that exist when using the cache.

Cache Coherency

Cache coherency is the discipline of ensuring that the data stored in main memory matches the corresponding data in the cache. The majority of the cache-related APIs deal with cache coherency. These functions allow the user to flush, clean and invalidate entire cache(s), or just a range of addresses within the cache.

Caches most often lose coherency when a bus master other than the CPU modifies memory. This happens frequently with DMA. Two examples are provided in the following section.

Examples

Example 1:

Imagine a situation where you would like to transfer data from a source buffer to a destination buffer using DMA. You would write data to the source buffer, start the DMA transfer, and then expect that the same data now appears in the destination buffer. With the cache in write-back mode (the default mode for the PIC32MZ family), this will not be the case. When transferring data out of memory using DMA, it is possible that the desired data is held in the D-cache, but has never been written back to main memory. Therefore, in this case, you write data to the source buffer and it gets stored in cache. When the DMA transfer executes, it will pull the data from the source buffer out of RAM and then transfer it to the destination buffer in RAM. The problem is that the fresh data was stored in the cache but never written back to RAM, so what has happened is that stale data was copied over rather than the intended data. What is needed is a way to force the cache to write its data back to main memory before the DMA transfer. This is known as a *write-back operation* and would be performed with the use of the function:
SYS_DEVCON_DataCacheClean(uint32_t addr, size_t len)

The example application, devcon_cache_clean, in the Device Control System Service Examples demonstrates this situation and shows how to resolve the issue.

Example 2:

The second situation involves writing data into memory using DMA. Imagine that the cache is holding a chunk of data known as destination_buffer. You then execute a DMA transfer to copy some new data from a source buffer into destination_buffer. The issue here is that main memory now contains the correct data, but the cache holds a copy of stale data for destination_buffer. The CPU cannot see this problem and it will keep pulling the data out of the cache, not even realizing that it's stale. What is needed is a way to tell the cache to pull the fresh data out of main memory, to replace the stale data that the cache contains. This is known as an *invalidate operation*. It is performed with the use of the function:
SYS_DEVCON_DataCacheInvalidate(uint32_t addr, size_t len)

The example application, devcon_cache_invalidate, in the Device Control System Service Examples demonstrates this situation and shows how to resolve the issue.

Configuring the Library

No configuration is required for the Device Control System Service.

Building the Library

This section lists the files that are available in the Device Control System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/devcon`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`sys_devcon.h`	Device Control System Service Library API header file.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/sys_devcon.c	Device Control System Service Library implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
src/sys_devcon_pic32mx.c	PIC32MX-specific implementation of SYS_DEVCON_PerformanceConfig.
/src/sys_devcon_pic32mz.c	PIC32MZ-specific implementation of SYS_DEVCON_PerformanceConfig.
/src/sys_devcon_local.h	Data structures and definitions used by the Device Control System Service, but not exposed to clients.
/src/sys_devcon_cache_pic3mz.S	PIC32MZ-specific implementation to support L1 Cache.

Module Dependencies

The Device Control System Service is not dependent upon any other modules.

Library Interface

a) System Interaction Functions

	Name	Description
	SYS_DEVCON_Deinitialize	Deinitializes the specific module instance of the DEVCON module
	SYS_DEVCON_Initialize	Initializes data for the instance of the Device Control module and opens the specific module instance.
	SYS_DEVCON_Reinitialize	Reinitializes and refreshes the hardware for the instance of the Device Control module.
	SYS_DEVCON_Status	Returns status of the specific instance of the Device Control module.
	SYS_DEVCON_Tasks	Maintains the system Device Control state machine.

b) Core Functionality Functions

	Name	Description
	SYS_DEVCON_SystemLock	Performs a system lock sequence by writing to the SYSKEY register.
	SYS_DEVCON_SystemUnlock	Performs a system unlock sequence by writing to the SYSKEY register.
	SYS_DEVCON_JTAGDisable	Disables the JTAG port on the device.
	SYS_DEVCON_JTAGEnable	Enables the JTAG port on the device.
	SYS_DEVCON_TraceDisable	Disables the trace output port on the device.
	SYS_DEVCON_TraceEnable	Enables the trace output port on the device.
	SYS_DEVCON_PerformanceConfig	Configures the PFM wait states and Prefetch Cache module for maximum performance.
	SYS_DEVCON_PowerModeEnter	Puts the device in a low-power state.

c) Cache Functions

	Name	Description
	SYS_DEVCON_CacheClean	Writes back and invalidates an address range in either cache.
	SYS_DEVCON_CacheCoherencyGet	Returns the current cache coherency attribute for kseg0.
	SYS_DEVCON_CacheCoherencySet	Sets the cache coherency attribute for kseg0.
	SYS_DEVCON_CacheFlush	Flushes the L1 cache.
	SYS_DEVCON_CacheInit	Initializes the L1 cache.
	SYS_DEVCON_CacheSync	Synchronizes the instruction and data caches.
	SYS_DEVCON_DataCacheClean	Writes back and invalidates an address range in the data cache.
	SYS_DEVCON_DataCacheFlush	Flushes the L1 data cache.

	SYS_DEVCON_DataCacheInvalidate	Invalidates an address range in the data cache.
	SYS_DEVCON_DataCacheLock	Fetches and locks a block of data in the data cache.
	SYS_DEVCON_DataCacheAssociativityGet	Returns the number of ways in the data cache.
	SYS_DEVCON_DataCacheLineSizeGet	Returns the data cache line size.
	SYS_DEVCON_DataCacheLinesPerWayGet	Returns the number of lines per way in the data cache.
	SYS_DEVCON_DataCacheSizeGet	Returns the total number of bytes in the data cache.
	SYS_DEVCON_InstructionCacheFlush	Flushes (invalidates) the L1 instruction cache.
	SYS_DEVCON_InstructionCacheInvalidate	Invalidates an address range in the instruction cache.
	SYS_DEVCON_InstructionCacheLock	Fetches and locks a block of instructions in the instruction cache.
	SYS_DEVCON_InstructionCacheAssociativityGet	Returns the number of ways in the instruction cache.
	SYS_DEVCON_InstructionCacheLineSizeGet	Returns the instruction cache line size.
	SYS_DEVCON_InstructionCacheLinesPerWayGet	Returns the number of lines per way in the instruction cache.
	SYS_DEVCON_InstructionCacheSizeGet	Returns the total number of bytes in the instruction cache.

d) Data Types and Constants

	Name	Description
	SYS_DEVCON_HANDLE	Identifies a particular registered event instance.
	SYS_DEVCON_INIT	Identifies the system device control initialize structure.
	SYS_DEVCON_INDEX_0	Device Control System Service index definitions.
	SYS_CACHE_COHERENCY	System L1 cache coherency settings.
	SYS_POWER_MODE	System power mode settings.

Description

This section describes the APIs of the Device Control System Service Library.

Refer to each section for a detailed description.

a) System Interaction Functions

SYS_DEVCON_Deinitialize Function

Deinitializes the specific module instance of the DEVCON module

File

sys_devcon.h

C

```
void SYS_DEVCON_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the specific module instance disabling its operation (and any hardware for driver modules). Resets all of the internal data structures and fields for the specified instance to the default settings.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Preconditions

The SYS_DEVCON_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from SYS_DEVCON_Initialize
SYS_STATUS          status;

SYS_DEVCON_Deinitialize (object);

status = SYS_DEVCON_Status (object);
if (SYS_MODULE_DEINITIALIZED == status)
```

```
{
    // Check again later if you need to know
    // when the SYS DEVCON is deinitialized.
}
```

Parameters

Parameters	Description
object	SYS DEVCON object handle, returned from SYS_DEVCON_Initialize

Function

void SYS_DEVCON_Deinitialize(SYS_MODULE_OBJ object)

SYS_DEVCON_Initialize Function

Initializes data for the instance of the Device Control module and opens the specific module instance.

File

sys_devcon.h

C

```
SYS_MODULE_OBJ SYS_DEVCON_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to an object. Otherwise, it returns SYS_MODULE_OBJ_INVALID. The returned object must be passed as argument to SYS_DEVCON_Reinitialize, SYS_DEVCON_Deinitialize, SYS_DEVCON_Tasks and SYS_DEVCON_Status routines.

Description

This function initializes the instance of the Device Control module, using the specified initialization data. It also initializes any internal data structures.

Remarks

This routine should only be called once during system initialization unless SYS_DEVCON_Deinitialize is first called to deinitialize the device instance before reinitializing it. If the system was already initialized it safely returns without causing any disturbance.

Preconditions

None.

Example

```
SYS_MODULE_OBJ  objectHandle;
SYS_DEVCON_INIT    initConfig;

// Populate the DEVCON initialization structure
initConfig.moduleInit.value     = SYS_MODULE_POWER_RUN_FULL;

objectHandle = SYS_DEVCON_Initialize (SYS_DEVCON_INDEX_0, (SYS_MODULE_INIT*)&initConfig);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Index for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the Device Control module. This pointer may be null if no data is required because static overrides have been provided.

Function

SYS_MODULE_OBJ SYS_DEVCON_Initialize(const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init)

SYS_DEVCON_Reinitialize Function

Reinitializes and refreshes the hardware for the instance of the Device Control module.

File

sys_devcon.h

C

```
void SYS_DEVCON_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None

Description

This function reinitializes the instance of the Device Control module using the supplied data. It modifies the internal data structure.

Remarks

This operation uses the same initialization data structure as the SYS_DEVCON_Initialize operation. This operation can be used to change the power state of a DEVCON module. This function can be called multiple times to reinitialize the module. This operation uses the same initialization data structure as the Initialize operation. This operation can also be used to refresh the hardware registers as defined by the initialization data.

Preconditions

The SYS_DEVCON_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ  objectHandle;
SYS_DEVCON_INIT    initConfig;
SYS_STATUS      devconStatus;

// Populate the device control initialization structure
initConfig.moduleInit.value     = SYS_MODULE_POWER_RUN_FULL;

SYS_DEVCON_Reinitialize (objectHandle, (SYS_MODULE_INIT*)&initConfig);

devconStatus = SYS_DEVCON_Status (object);
if (SYS_STATUS_ERROR >= devconStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Identifies the SYS DEVCON Object returned by the Initialize interface
init	Pointer to the data structure containing any data necessary to initialize the hardware

Function

void SYS_DEVCON_Reinitialize(SYS_MODULE_OBJ object,
const SYS_MODULE_INIT * const init)

SYS_DEVCON_Status Function

Returns status of the specific instance of the Device Control module.

File

sys_devcon.h

C

```
SYS_STATUS SYS_DEVCON_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that any previous operations have succeeded and the module is ready for additional operations. Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

- SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another.
- SYS_STATUS_ERROR - Indicates that the driver is in an error state. Any value less than SYS_STATUS_ERROR is also an error state.
- SYS_MODULE_DEINITIALIZED - Indicates that the driver has been deinitialized. This value is less than SYS_STATUS_ERROR. Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

Description

This function returns the status of the specific module instance disabling its operation (and any hardware for driver modules).

Remarks

None.

Preconditions

The SYS_DEVCON_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from SYS_DEVCON_Initialize
SYS_STATUS          tmrStatus;

devconStatus = SYS_DEVCON_Status (object);
else if (SYS_STATUS_ERROR >= devconStatus)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	SYS DEVCON object handle, returned from SYS_DEVCON_Initialize

Function

SYS_STATUS SYS_DEVCON_Status(SYS_MODULE_OBJ object)

SYS_DEVCON_Tasks Function

Maintains the system Device Control state machine.

File

sys_devcon.h

C

```
void SYS_DEVCON_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the system Device Control internal state machine.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

Preconditions

The SYS_DEVCON_Initialize function must have been called for the specified DEVCON driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from SYS_DEVCON_Initialize

while (true)
{
    SYS_DEVCON_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	SYS DEVCON object handle, returned from SYS_DEVCON_Initialize

Function

void SYS_DEVCON_Tasks(SYS_MODULE_OBJ object)

b) Core Functionality Functions

SYS_DEVCON_SystemLock Function

Performs a system lock sequence by writing to the SYSKEY register.

File

sys_devcon.h

C

```
void SYS_DEVCON_SystemLock();
```

Returns

None.

Description

Performs a system lock sequence by writing to the SYSKEY register. A system lock sequence is required after performing the action that required a system lock sequence.

Remarks

None.

Preconditions

None.

Example

```
SYS_DEVCON_SystemUnlock();
// Perform some action
SYS_DEVCON_SystemLock();
```

Function

void SYS_DEVCON_SystemLock(void)

SYS_DEVCON_SystemUnlock Function

Performs a system unlock sequence by writing to the SYSKEY register.

File

sys_devcon.h

C

```
void SYS_DEVCON_SystemUnlock();
```

Returns

None.

Description

Performs a system unlock sequence by writing to the SYSKEY register. A system unlock sequence is required before performing certain actions such as changing a clock frequency or I/O unlocking.

Remarks

None.

Preconditions

None.

Example

```
SYS_DEVCON_SystemUnlock();
// Perform some action
SYS_DEVCON_SystemLock();
```

Function

void SYS_DEVCON_SystemUnlock(void)

SYS_DEVCON_JTAGDisable Function

Disables the JTAG port on the device.

File

sys_devcon.h

C

```
void SYS_DEVCON_JTAGDisable();
```

Returns

None.

Description

This function disables the JTAG port on the device.

Remarks

None.

Preconditions

None.

Example

```
SYS_DEVCON_JTAGDisable();
```

Function

void SYS_DEVCON_JTAGDisable(void)

SYS_DEVCON_JTAGEnable Function

Enables the JTAG port on the device.

File

sys_devcon.h

C

```
void SYS_DEVCON_JTAGEnable();
```

Returns

None.

Description

This function enables the JTAG port on the device.

Remarks

None.

Preconditions

None.

Example

```
SYS_DEVCON_JTAGEnable();
```

Function

void SYS_DEVCON_JTAGEnable(void)

SYS_DEVCON_TraceDisable Function

Disables the trace output port on the device.

File

sys_devcon.h

C

```
void SYS_DEVCON_TraceDisable();
```

Returns

None.

Description

This function disables the trace output port on the device.

Remarks

None.

Preconditions

None.

Example

```
SYS_DEVCON_TraceDisable();
```

Function

void SYS_DEVCON_TraceDisable(void)

SYS_DEVCON_TraceEnable Function

Enables the trace output port on the device.

File

sys_devcon.h

C

```
void SYS_DEVCON_TraceEnable();
```

Returns

None.

Description

This function enables the trace output port on the device.

Remarks

None.

Preconditions

None.

Example

```
SYS_DEVCON_TraceEnable();
```

Function

void SYS_DEVCON_TraceEnable(void)

SYS_DEVCON_PerformanceConfig Function

Configures the PFM wait states and Prefetch Cache module for maximum performance.

File

sys_devcon.h

C

```
void SYS_DEVCON_PerformanceConfig(unsigned int sysclk);
```

Returns

None.

Description

This function configures the PFM wait states and Prefetch Cache module for maximum performance.

Remarks

None.

Preconditions

The SYS_DEVCON_Initialize function should have been called before calling this function.

Example

```
SYS_DEVCON_PerformanceConfig(SYS_FREQUENCY);
```

Function

void SYS_DEVCON_PerformanceConfig(unsigned int sysclk)

SYS_DEVCON_PowerModeEnter Function

Puts the device in a low-power state.

File

sys_devcon.h

C

```
void SYS_DEVCON_PowerModeEnter(SYS_POWER_MODE pwrMode);
```

Returns

None.

Description

This function places the device in a low-power state.

Remarks

None.

Preconditions

None.

Example

```
SYS_DEVCON_PowerModeEnter(SYS_POWER_MODE_SLEEP);
```

Parameters

Parameters	Description
pwrMode	The low power state to enter.

Function

void SYS_DEVCON_PowerModeEnter(SYS_POWER_MODE pwrMode)

c) Cache Functions

SYS_DEVCON_CacheClean Function

Writes back and invalidates an address range in either cache.

File

sys_devcon.h

C

```
void SYS_DEVCON_CacheClean(uint32_t addr, size_t len);
```

Returns

None.

Description

This function writes back (data) and invalidates (data and address) an address range in either cache.

Remarks

None.

Preconditions

None.

Example

```
SYS_DEVCON_CacheClean(&myData, sizeof(myData));
```

Parameters

Parameters	Description
addr	Starting address of data block.
len	Size of data block.

Function

void SYS_DEVCON_CacheClean(uint32_t addr, size_t len)

SYS_DEVCON_CacheCoherencyGet Function

Returns the current cache coherency attribute for kseg0.

File

sys_devcon.h

C

```
SYS_CACHE_COHERENCY SYS_DEVCON_CacheCoherencyGet();
```

Returns

The current cache coherency attribute for kseg0.

Description

This function returns the current cache coherency attribute for kseg0.

Remarks

None.

Preconditions

None.

Example

```
SYS_CACHE_COHERENCY cacheCoherency;
cacheCoherency = SYS_DEVCON_CacheCoherencyGet();
```

Function

SYS_CACHE_COHERENCY SYS_DEVCON_CacheCoherencyGet(void)

SYS_DEVCON_CacheCoherencySet Function

Sets the cache coherency attribute for kseg0.

File

sys_devcon.h

C

```
void SYS_DEVCON_CacheCoherencySet(SYS_CACHE_COHERENCY cacheCoherency);
```

Returns

None.

Description

This function sets the cache coherency attribute for kseg0.

Remarks

Use with caution. May cause unpredictable behavior in a running system.

Preconditions

None.

Example

```
SYS_DEVCON_CacheCoherencySet(SYS_CACHE_WRITEBACK_WRITEALLOCATE);
```

Parameters

Parameters	Description
cacheCoherency	The desired coherency attribute.

Function

void SYS_DEVCON_CacheCoherencySet(SYS_CACHE_COHERENCY cacheCoherency)

SYS_DEVCON_CacheFlush Function

Flushes the L1 cache.

File

sys_devcon.h

C

```
void SYS_DEVCON_CacheFlush();
```

Returns

None.

Description

This function flushes both instruction and data caches. Invalidate entire instruction cache; writes back and invalidates the entire data cache.

Remarks

Simplest way to synchronize caches with memory, but not necessarily the most efficient.

Preconditions

None.

Example

```
SYS_DEVCON_CacheFlush();
```

Function

void SYS_DEVCON_CacheFlush(void)

SYS_DEVCON_CacheInit Function

Initializes the L1 cache.

File

sys_devcon.h

C

```
void SYS_DEVCON_CacheInit(SYS_CACHE_COHERENCY cacheCoherency);
```

Returns

None.

Description

This function initializes both instruction and data caches. Invalidates all entries and zeros all tags. Sets coherency attribute for kseg0.

Remarks

Use with caution. Invalidates all cache lines without writing data back to memory. Any dirty data in the cache will be lost.

Preconditions

None.

Example

```
SYS_DEVCON_CacheInit(SYS_CACHE_WRITEBACK_WRITEALLOCATE);
```

Parameters

Parameters	Description
cacheCoherency	The desired kseg0 coherency attribute.

Function

void SYS_DEVCON_CacheInit(SYS_CACHE_COHERENCY cacheCoherency)

SYS_DEVCON_CacheSync Function

Synchronizes the instruction and data caches.

File

sys_devcon.h

C

```
void SYS_DEVCON_CacheSync(uint32_t addr, size_t len);
```

Returns

None.

Description

This function synchronizes the instruction and data caches. Used when modifying the instruction stream (breakpoints, self-modifying code, relocating executable code to RAM). Flushes an address range from the data cache and invalidates that same range from the instruction cache.

Remarks

None.

Preconditions

None.

Example

```
SYS_DEVCON_CacheSync(&myCode, myCodeSize);
```

Parameters

Parameters	Description
addr	Starting address of code block.
len	Size of code block.

Function

void SYS_DEVCON_CacheSync(uint32_t addr, size_t len)

SYS_DEVCON_DataCacheClean Function

Writes back and invalidates an address range in the data cache.

File

sys_devcon.h

C

```
void SYS_DEVCON_DataCacheClean(uint32_t addr, size_t len);
```

Returns

None.

Description

This function writes back and invalidates an address range in the data cache.

Remarks

None.

Preconditions

None.

Example

```
SYS_DEVCON_DataCacheClean(&myData, sizeof(myData));
```

Parameters

Parameters	Description
addr	Starting address of data block.
len	Size of data block.

Function

void SYS_DEVCON_DataCacheClean(uint32_t addr, size_t len)

SYS_DEVCON_DataCacheFlush Function

Flushes the L1 data cache.

File

sys_devcon.h

C

```
void SYS_DEVCON_DataCacheFlush();
```

Returns

None.

Description

This function writes back and invalidates the entire data cache.

Remarks

None.

Preconditions

None.

Example

```
SYS_DEVCON_DataCacheFlush();
```

Function

void SYS_DEVCON_DataCacheFlush(void)

SYS_DEVCON_DataCacheInvalidate Function

Invalidates an address range in the data cache.

File

sys_devcon.h

C

```
void SYS_DEVCON_DataCacheInvalidate(uint32_t addr, size_t len);
```

Returns

None.

Description

This function invalidates an address range in the data cache.

Remarks

Use caution. Any dirty data in the cache will be lost.

Preconditions

None.

Example

```
SYS_DEVCON_DataCacheInvalidate(&myData, sizeof(myData));
```

Parameters

Parameters	Description
addr	Starting address of data block.
len	Size of data block.

Function

void SYS_DEVCON_DataCacheInvalidate(uint32_t addr, size_t len)

SYS_DEVCON_DataCacheLock Function

Fetches and locks a block of data in the data cache.

File

sys_devcon.h

C

```
void SYS_DEVCON_DataCacheLock(uint32_t addr, size_t len);
```

Returns

None.

Description

This function fetches and locks a block of data in the data cache.

Remarks

Global flush functions will invalidate and unlock any cache lines locked with this function.

Preconditions

None.

Example

```
SYS_DEVCON_DataCacheLock(&myData, sizeof(myData));
```

Parameters

Parameters	Description
addr	Starting address of data block.
len	Size of data block.

Function

void SYS_DEVCON_DataCacheLock(uint32_t addr, size_t len)

SYS_DEVCON_DataCacheAssociativityGet Function

Returns the number of ways in the data cache.

File

sys_devcon.h

C

```
size_t SYS_DEVCON_DataCacheAssociativityGet();
```

Returns

The number of ways in the data cache.

Description

This function returns the number of ways in the data cache.

Remarks

None.

Preconditions

None.

Example

```
unsigned int size;
size = SYS_DEVCON_DataCacheAssociativityGet();
```

Function

size_t SYS_DEVCON_DataCacheAssociativityGet(void)

SYS_DEVCON_DataCacheLineSizeGet Function

Returns the data cache line size.

File

sys_devcon.h

C

```
size_t SYS_DEVCON_DataCacheLineSizeGet();
```

Returns

The number of bytes per line in the data cache.

Description

This function returns the data cache line size.

Remarks

None.

Preconditions

None.

Example

```
unsigned int size;
size = SYS_DEVCON_DataCacheLineSizeGet();
```

Function

size_t SYS_DEVCON_DataCacheLineSizeGet(void)

SYS_DEVCON_DataCacheLinesPerWayGet Function

Returns the number of lines per way in the data cache.

File

sys_devcon.h

C

```
size_t SYS_DEVCON_DataCacheLinesPerWayGet();
```

Returns

The number of lines per way in the data cache.

Description

This function returns the number of lines per way in the data cache.

Remarks

None.

Preconditions

None.

Example

```
unsigned int size;
size = SYS_DEVCON_DataCacheLinesPerWayGet();
```

Function

size_t SYS_DEVCON_DataCacheLinesPerWayGet(void)

SYS_DEVCON_DataCacheSizeGet Function

Returns the total number of bytes in the data cache.

File

sys_devcon.h

C

```
size_t SYS_DEVCON_DataCacheSizeGet();
```

Returns

The total number of bytes in the data cache.

Description

This function returns the total number of bytes in the data cache.

Remarks

None.

Preconditions

None.

Example

```
unsigned int size;
size = SYS_DEVCON_DataCacheSizeGet();
```

Function

size_t SYS_DEVCON_DataCacheSizeGet(void)

SYS_DEVCON_InstructionCacheFlush Function

Flushes (invalidates) the L1 instruction cache.

File

sys_devcon.h

C

```
void SYS_DEVCON_InstructionCacheFlush();
```

Returns

None.

Description

This function invalidates the entire instruction cache.

Remarks

None.

Preconditions

None.

Example

```
SYS_DEVCON_InstructionCacheFlush();
```

Function

void SYS_DEVCON_InstructionCacheFlush(void)

SYS_DEVCON_InstructionCacheInvalidate Function

Invalidates an address range in the instruction cache.

File

sys_devcon.h

C

```
void SYS_DEVCON_InstructionCacheInvalidate(uint32_t addr, size_t len);
```

Returns

None.

Description

This function invalidates an address range in the instruction cache.

Remarks

None.

Preconditions

None.

Example

```
SYS_DEVCON_InstructionCacheInvalidate(&myCode, myCodeSize);
```

Parameters

Parameters	Description
addr	Starting address of code block.
len	Size of code block.

Function

void SYS_DEVCON_InstructionCacheInvalidate(uint32_t addr, size_t len)

SYS_DEVCON_InstructionCacheLock Function

Fetches and locks a block of instructions in the instruction cache.

File

sys_devcon.h

C

```
void SYS_DEVCON_InstructionCacheLock(uint32_t addr, size_t len);
```

Returns

None.

Description

This function fetches and locks a block of instructions in the instruction cache.

Remarks

Global flush functions will invalidate and unlock any cache lines locked with this function.

Preconditions

None.

Example

```
SYS_DEVCON_InstructionCacheLock(&myCode, myCodeSize);
```

Parameters

Parameters	Description
addr	Starting address of code block.
len	Size of code block.

Function

void SYS_DEVCON_InstructionCacheLock(uint32_t addr, size_t len)

SYS_DEVCON_InstructionCacheAssociativityGet Function

Returns the number of ways in the instruction cache.

File

sys_devcon.h

C

```
size_t SYS_DEVCON_InstructionCacheAssociativityGet();
```

Returns

The number of ways in the instruction cache.

Description

This function returns the number of ways in the instruction cache.

Remarks

None.

Preconditions

None.

Example

```
unsigned int size;
size = SYS_DEVCON_InstructionCacheAssociativityGet();
```

Function

size_t SYS_DEVCON_InstructionCacheAssociativityGet(void)

SYS_DEVCON_InstructionCacheLineSizeGet Function

Returns the instruction cache line size.

File

sys_devcon.h

C

```
size_t SYS_DEVCON_InstructionCacheLineSizeGet();
```

Returns

The number of bytes per line in the instruction cache.

Description

This function returns the instruction cache line size.

Remarks

None.

Preconditions

None.

Example

```
unsigned int size;
size = SYS_DEVCON_InstructionCacheLineSizeGet();
```

Function

size_t SYS_DEVCON_InstructionCacheLineSizeGet(void)

SYS_DEVCON_InstructionCacheLinesPerWayGet Function

Returns the number of lines per way in the instruction cache.

File

sys_devcon.h

C

```
size_t SYS_DEVCON_InstructionCacheLinesPerWayGet();
```

Returns

The number of lines per way in the instruction cache.

Description

This function returns the number of lines per way in the instruction cache.

Remarks

None.

Preconditions

None.

Example

```
unsigned int size;
size = SYS_DEVCON_InstructionCacheLinesPerWayGet();
```

Function

size_t SYS_DEVCON_InstructionCacheLinesPerWayGet(void)

SYS_DEVCON_InstructionCacheSizeGet Function

Returns the total number of bytes in the instruction cache.

File

sys_devcon.h

C

```
size_t SYS_DEVCON_InstructionCacheSizeGet();
```

Returns

The total number of bytes in the instruction cache.

Description

This function returns the total number of bytes in the instruction cache.

Remarks

None.

Preconditions

None.

Example

```
unsigned int size;
size = SYS_DEVCON_InstructionCacheSizeGet();
```

Function

size_t SYS_DEVCON_InstructionCacheSizeGet(void)

d) Data Types and Constants

SYS_DEVCON_HANDLE Type

Identifies a particular registered event instance.

File

sys_devcon.h

C

```
typedef int8_t SYS_DEVCON_HANDLE;
```

Description

SYS DEVCON Handle. This event handle identifies a registered instance of an event. Every time the application that tries to access the parameters with respect to a particular event, shall used this event handle to refer to that event.

Remarks

None.

SYS_DEVCON_INIT Structure

Identifies the system device control initialize structure.

File

sys_devcon.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
} SYS_DEVCON_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization

Description

SYS DEVCON Initialize structure

This structure identifies the system device control initialize structure.

Remarks

None.

SYS_DEVCON_INDEX_0 Macro

Device Control System Service index definitions.

File

sys_devcon.h

C

```
#define SYS_DEVCON_INDEX_0 0
```

Description

SYS Device Control Module Index Numbers

These constants provide Device Control System Service index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

SYS_CACHE_COHERENCY Enumeration

System L1 cache coherency settings.

File

sys_devcon.h

C

```
typedef enum {
  SYS_CACHE_WRITETHROUGH_NO_WRITEALLOCATE = 0,
  SYS_CACHE_WRITETHROUGH_WRITEALLOCATE = 1,
  SYS_CACHE_DISABLE = 2,
  SYS_CACHE_WRITEBACK_WRITEALLOCATE = 3
} SYS_CACHE_COHERENCY;
```

Description

SYS_CACHE_COHERENCY enumeration

This enumeration defines the supported system L1 cache coherency settings.

Remarks

Used to read or write cache coherency policy for kseg0.

SYS_POWER_MODE Enumeration

System power mode settings.

File

sys_devcon.h

C

```
typedef enum {
  SYS_POWER_MODE_IDLE,
  SYS_POWER_MODE_SLEEP
} SYS_POWER_MODE;
```

Description

SYS_POWER_MODE enumeration

This enumeration defines the supported system power mode settings.

Remarks

None.

Files

Files

Name	Description
sys_devcon.h	Device Control (DEVCON) System Service interface definition.

Description

This section lists the source and header files used by the library.

sys_devcon.h

Device Control (DEVCON) System Service interface definition.

Enumerations

	Name	Description
	SYS_CACHE_COHERENCY	System L1 cache coherency settings.
	SYS_POWER_MODE	System power mode settings.

Functions

	Name	Description
	SYS_DEVCON_CacheClean	Writes back and invalidates an address range in either cache.
	SYS_DEVCON_CacheCoherencyGet	Returns the current cache coherency attribute for kseg0.
	SYS_DEVCON_CacheCoherencySet	Sets the cache coherency attribute for kseg0.
	SYS_DEVCON_CacheFlush	Flushes the L1 cache.
	SYS_DEVCON_CacheInit	Initializes the L1 cache.
	SYS_DEVCON_CacheSync	Synchronizes the instruction and data caches.
	SYS_DEVCON_DataCacheAssociativityGet	Returns the number of ways in the data cache.
	SYS_DEVCON_DataCacheClean	Writes back and invalidates an address range in the data cache.
	SYS_DEVCON_DataCacheFlush	Flushes the L1 data cache.
	SYS_DEVCON_DataCacheInvalidate	Invalidates an address range in the data cache.
	SYS_DEVCON_DataCacheLineSizeGet	Returns the data cache line size.
	SYS_DEVCON_DataCacheLinesPerWayGet	Returns the number of lines per way in the data cache.
	SYS_DEVCON_DataCacheLock	Fetches and locks a block of data in the data cache.
	SYS_DEVCON_DataCacheSizeGet	Returns the total number of bytes in the data cache.
	SYS_DEVCON_Deinitialize	Deinitializes the specific module instance of the DEVCON module
	SYS_DEVCON_Initialize	Initializes data for the instance of the Device Control module and opens the specific module instance.
	SYS_DEVCON_InstructionCacheAssociativityGet	Returns the number of ways in the instruction cache.
	SYS_DEVCON_InstructionCacheFlush	Flushes (invalidates) the L1 instruction cache.
	SYS_DEVCON_InstructionCacheInvalidate	Invalidates an address range in the instruction cache.
	SYS_DEVCON_InstructionCacheLineSizeGet	Returns the instruction cache line size.

	SYS_DEVCON_InstructionCacheLinesPerWayGet	Returns the number of lines per way in the instruction cache.
	SYS_DEVCON_InstructionCacheLock	Fetches and locks a block of instructions in the instruction cache.
	SYS_DEVCON_InstructionCacheSizeGet	Returns the total number of bytes in the instruction cache.
	SYS_DEVCON_JTAGDisable	Disables the JTAG port on the device.
	SYS_DEVCON_JTAGEnable	Enables the JTAG port on the device.
	SYS_DEVCON_PerformanceConfig	Configures the PFM wait states and Prefetch Cache module for maximum performance.
	SYS_DEVCON_PowerModeEnter	Puts the device in a low-power state.
	SYS_DEVCON_Reinitialize	Reinitializes and refreshes the hardware for the instance of the Device Control module.
	SYS_DEVCON_Status	Returns status of the specific instance of the Device Control module.
	SYS_DEVCON_SystemLock	Performs a system lock sequence by writing to the SYSKEY register.
	SYS_DEVCON_SystemUnlock	Performs a system unlock sequence by writing to the SYSKEY register.
	SYS_DEVCON_Tasks	Maintains the system Device Control state machine.
	SYS_DEVCON_TraceDisable	Disables the trace output port on the device.
	SYS_DEVCON_TraceEnable	Enables the trace output port on the device.

Macros

	Name	Description
	SYS_DEVCON_INDEX_0	Device Control System Service index definitions.

Structures

	Name	Description
	SYS_DEVCON_INIT	Identifies the system device control initialize structure.

Types

	Name	Description
	SYS_DEVCON_HANDLE	Identifies a particular registered event instance.

Description

Device Control System Service Interface Definition

This file contains the interface definition for the Device Control System Service. It provides a way to interact with the Device Control subsystem to manage the device control requests supported by the system.

File Name

sys_devcon.h

Company

Microchip Technology Inc.

Direct Memory Access (DMA) System Service Library

This section describes the Direct Memory Access (DMA) System Service Library.

Introduction

This library provides an interface to interact with the DMA subsystem to control and manage the data transfer between different peripherals and/or memory without intervention from the CPU.

Description

The Direct Memory Access (DMA) controller is a bus master module that is useful for data transfers between different peripherals without intervention from the CPU. The source and destination of a DMA transfer can be any of the memory-mapped modules. For example, memory, or one of the Peripheral Bus (PBUS) devices such as the SPI, UART, and so on.

Using the Library

This topic describes the basic architecture of the DMA System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_dma.h`

The interface to the DMA System Service library is defined in the `sys_dma.h` header file. This file is included by the `system.h` file. Any C language source (`.c`) file that uses the DMA System Service Library should include `system.h`.

Please refer to the What is MPLAB Harmony? section for how the DMA System Service interacts with the framework.

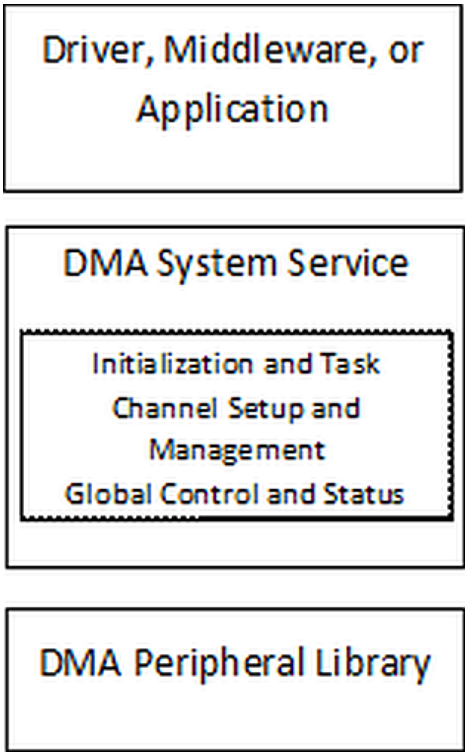
Abstraction Model

This model explains how the system interfaces with the DMA System Service and the application.

Description

This library provides an abstraction of the DMA subsystem that is used by device drivers, middleware libraries and applications to transmit and receive data.

DMA Software Abstraction Block Diagram



DMA System Service

The DMA system services provide support for initializing the DMA controller, managing the transfer state machine, setup/management of channels, and global module control and status management.

Initialization and Tasks

Each software module (device driver, middleware, or application) that needs to use the DMA for data transfer must enable the DMA controller. This is normally done by calling the initialization routine of the DMA subsystem in the module's initialization routine, which is called by the SYS_Initialize service. The initialization routine returns a DMA module object, which should be used as a parameter in the call to Task routines.

The Task routines implement the data transfer state machine for synchronous and asynchronous data transfer operations. If Asynchronous (interrupt) mode of operation is desired, the Task routine(SYS_DMA_TasksISR) should be called from the respective channel ISR. If Synchronous mode of operation if desired, the Task routine(SYS_DMA_Tasks) should be called from the SYS_Tasks function.

Channel Setup and Management

Any module that needs to use the DMA system service must request for channel allocation. An allocated channel is used to setup the channel parameters like the mode of operation(Basic, CRC, chaining etc). Setup the transfer trigger types(Synchronous/Asynchronous). Add a transfer by Setting up the source, destination address and transfer sizes. The DMA transfer starts either forcefully or based on events according to the setting. The channel status events are used to manage the data transfer.

Global Control and Status Management

Provides for control and status of the DMA module. The user can suspend a DMA operation or alternatively resume an already suspended operation. The status of last DMA operation can also be retrieved.

Library Overview

Please refer to the System Service Introduction for a detailed description of MPLAB Harmony system services.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DMA module.

Library Interface Section	Description
Initialization and Task Functions	These functions initialize and enable the DMA subsystem and manage the task State machine.
Channel Setup and Management Functions	These DMA Channel and Setup Management functions enable: • Channel Allocation and Release • Setting up of a channel for basic and advanced modes of operations • Channel Enable/Disable • Adding a data transfer • Synchronous data transfer start/abort • Asynchronous data transfer start/abort • Channel Busy status • Channel Computed CRC • Setting the transfer event trigger callback
Global Control and Status Functions	These DMA functions enable global control and status.

How the Library Works

Initialization and Tasks

Describes the functions that can be used for DMA initialization, and provides examples of their usage.

Description

Initialization Function Usage

The DMA subsystem is initialized by calling the initialization routine. The routine also enables the DMA module.

Function Name: SYS_DMA_Initialize

Example

```
//To Enable the Stop in Idle mode feature

SYS_MODULE_OBJ objectHandle;
SYS_DMA_INIT initParam;

initParam.sidl = SYS_DMA_SIDL_ENABLE;

objectHandle = SYS_DMA_Initialize((SYS_MODULE_INIT*)&initParam;);
if(SYS_MODULE_OBJ_INVALID == objectHandle)
{
// Handle error
}
```

Task Function Usage

The task routine implements the data transfer state machine and returns a callback on the completion, abortion or error in a data transfer.

Function Name: SYS_DMA_Tasks

Example

```
// 'object' Returned from SYS_DMA_Initialize
    while (true)
    {
        SYS_DMA_Tasks (object);
        // Do other tasks
    }
```

Note: The SYS_DMA_Tasks function should be used in Synchronous (polling) mode only. This function should not be called from an ISR.

Function Name: SYS_DMA_TasksError

Example

```
// 'object' Returned from SYS_DMA_Initialize
    while (true)
    {
        SYS_DMA_TasksError (object);
        // Do other tasks
    }
```

Note: The SYS_DMA_TasksError function should be used in Synchronous (polling) mode only. This function should not be called from an ISR.

Function Name: SYS_DMA_TasksISR

Example

```
// 'object' Returned from SYS_DMA_Initialize
// Channel 3 is setup for receiving data by USART peripheral in interrupt mode

// Following is the DMA Channel 3 ISR

void __ISR(_DMA3_VECTOR,ipl5) _InterruptHandler_BT_USART_RX_DMA_CHANNEL(void)
{
 SYS_DMA_TasksISR(object, DMA_CHANNEL_3);
}
```

Note: The SYS_DMA_TasksISR function should be used in Asynchronous (interrupt) mode only. This function should not be called from the SYS_Tasks function.

Function Name: SYS_DMA_TasksErrorISR

Example

```
// 'object' Returned from SYS_DMA_Initialize
// Channel 3 is setup for receiving data by USART peripheral in interrupt mode

// Following is the DMA Channel 3 ISR

void __ISR(_DMA3_VECTOR,ipl5) _InterruptHandler_BT_USART_RX_DMA_CHANNEL(void)
{
 SYS_DMA_TasksErrorISR(object, DMA_CHANNEL_3);
}
```

Note: The SYS_DMA_TasksErrorISR function should be used in Asynchronous (interrupt) mode only. This function should not be called from the SYS_Tasks function.

Channel Setup and Management

Describes the functions that can be used for DMA channel setup and management, and provides examples of their usage.

Description

Channel Allocation and Release Functions

Channel Allocate and Release functions allocate/release a particular channel from the available channels on the particular device.

Channel Allocation Function

Channel Allocate function takes a parameter specifying the requested channel number. If the requested channel is available the function allocates the channel and returns a channel handle. If the user is not particular about any specific channel, the user can specify DMA_CHANNEL_ANY enumerator values. When DMA_CHANNEL_ANY values are specified an available channel is allocated and a channel handle is returned. The function returns an invalid channel handle SYS_DMA_CHANNEL_HANDLE_INVALID when the requested channel is not available for allocation.

The valid channel handle returned by this function must be used in all subsequent DMA channel function calls.

Function Name: SYS_DMA_ChannelAllocate

Example 1

```
/* The following examples requests for allocation of a channel handle with Channel number 2 */
SYS_DMA_CHANNEL_HANDLE  handle
```

```
DMA_CHANNEL             channel;
channel = DMA_CHANNEL_2;
handle  = SYS_DMA_ChannelAllocate(channel);
```

Example 2

```
/* The following example requests for allocation of a channel handle with no choice of channel number */
SYS_DMA_CHANNEL_HANDLE  handle
DMA_CHANNEL             channel;
channel = DMA_CHANNEL_ANY;
handle  = SYS_DMA_ChannelAllocate(channel);
```

Precondition: DMA should be initialized by calling SYS_DMA_Initialize.

Channel Release Function

Channel Release function Deallocates and frees the DMA channel specified by the channel handle.

Function Name: SYS_DMA_ChannelRelease

Example

```
/* The following example requests for release of a channel handle with Channel number 2 */
DMA_CHANNEL             channel;
SYS_DMA_CHANNEL_HANDLE  handle;
channel = DMA_CHANNEL_2;
handle  = SYS_DMA_ChannelAllocate(channel);
SYS_DMA_ChannelRelease(handle);
```

Precondition: DMA should be initialized by calling SYS_DMA_Initialize.

Channel Setup Functions

The DMA subsystem supports the following modes of operations. The setting up of these modes of operation can be done by the following functions.

- Basic Transfer Mode
- Pattern Match Abort Mode
- Channel Chaining Mode
- Channel Auto Enable Mode
- CRC Mode

Note: The operation modes are not mutually exclusive.

General Channel Setup Function

This function sets up the channel for the supported operating modes. The function does the following:

- Sets up the channel priority
- Enables the specified mode
- Sets up the DMA asynchronous transfer mode

If the DMA channel transfer is intended to be synchronous, the parameter 'eventSrc' (asynchronous trigger source) can be specified as ‘DMA_TRIGGER_SOURCE_NONE’. When the channel trigger source is specified as ‘DMA_TRIGGER_SOURCE_NONE’, The DMA channel transfer needs to be forcefully started by calling the respective function.

Notes:

1. Enabling of the available operation mode is not mutually exclusive. More than one operation mode can be enabled by bitwise ORing the operating mode enable parameter.
2. To setup the specific features of the supported operation modes the corresponding function needs to be called after calling this function.

Function Name: SYS_DMA_ChannelSetup

Example

```
/* Configure channel number, priority and enables basic and CRC mode */
SYS_DMA_CHANNEL_HANDLE  handle;
DMA_CHANNEL             channel;
SYS_DMA_CHANNEL_OP_MODE modeEnable;
DMA_TRIGGER_SOURCE      eventSrc;
channel     = DMA_CHANNEL_2;
modeEnable  = (SYS_DMA_CHANNEL_OP_MODE_BASIC | SYS_DMA_CHANNEL_OP_MODE_CRC);
eventSrc    = DMA_TRIGGER_USART_1_TRANSMIT;
handle      = SYS_DMA_ChannelAllocate(channel);
SYS_DMA_ChannelSetup(handle, modeEnable,eventSrc);
```

Precondition: DMA should be initialized by calling SYS_DMA_Initialize.

Pattern Match Abort Mode Function

This function sets up the termination of DMA operation when the specified pattern is matched. Additionally on supported devices, the function also sets up the ignoring of part of a pattern (8-bit) from match abort pattern (16-bit).

Before calling this function the pattern match termination mode must have been enabled by calling the general channel setup function

Function Name: SYS_DMA_ChannelSetupMatchAbortMode

Example 1

```
/* The following code is for a device with 8-bit pattern value and no support for pattern match ignore
feature */
SYS_DMA_CHANNEL_HANDLE       handle;
DMA_CHANNEL                  channel;
uint16_t                     pattern;
DMA_PATTERN_LENGTH           length
SYS_DMA_CHANNEL_IGNORE_MATCH ignore;
uint8_t                      ignorePattern;
channel            = DMA_CHANNEL_2;
priority           = DMA_CHANNEL_PRIORITY_1;
pattern            = 0x00; //Stop transfer on detection of a NULL character
length             = DMA_PATTERN_LENGTH_NONE;
ignore             = SYS_DMA_CHANNEL_IGNORE_MATCH_DISABLE;
ignorePattern      = 0;
handle             = SYS_DMA_ChannelAllocate(channel);
SYS_DMA_ChannelSetupMatchAbortMode(handle, pattern, length, ignoreEnable, ignorePattern);
```

Example 2

```
/* The following code is for a device with 16-bit pattern value and support for pattern match ignore
feature */
SYS_DMA_CHANNEL_HANDLE       handle;
DMA_CHANNEL                  channel;
uint16_t                     pattern;
DMA_PATTERN_LENGTH           length
SYS_DMA_CHANNEL_IGNORE_MATCH ignore;
uint8_t                      ignorePattern;
priority           = DMA_CHANNEL_PRIORITY_1;
channel            = DMA_CHANNEL_2;
pattern            = 0x0D0A; //Stop transfer on detection of '\r\n'
length             = DMA_PATTERN_MATCH_LENGTH_2BYTES;
ignore             = SYS_DMA_CHANNEL_IGNORE_MATCH_ENABLE;
ignorePattern      = 0x00; \\ Any null character between the termination pattern '\r' and '\n' is ignored.
handle             = SYS_DMA_ChannelAllocate(channel);
SYS_DMA_ChannelSetupMatchAbortMode(handle, pattern, length, ignore, ignorePattern);
```

Preconditions:

1. DMA should be initialized by calling SYS_DMA_Initialize.

2. Channel should be setup and pattern match mode enabled by calling SYS_DMA_ChannelSetup.

CRC Operation Mode Setup Function

This function sets up the CRC computation features of the channel. Before calling this function the CRC mode must have been enabled by calling the general channel setup function

Function Name: SYS_DMA_ChannelSetupCRCMode

Example

```
/* DMA calculation using the CRC background mode */
SYS_DMA_CHANNEL_HANDLE             handle;
DMA_CHANNEL                        channel;
SYS_DMA_CHANNEL_OPERATION_MODE_CRC crc;
channel        = DMA_CHANNEL_2;
crc.type       = DMA_CRC_LFSR;
crc.mode       = SYS_DMA_CHANNEL_CRC_MODE_BACKGROUND;
crc.polyLength = 16;
crc.bitOrder   = DMA_CRC_BIT_ORDER_LSB;
crc.byteOrder  = DMA_CRC_BYTEORDER_NO_SWAPPING;
crc.writeOrder = SYS_DMA_CRC_WRITE_ORDER_MAINTAIN;
crc.data       = 0xFFFF;
crc.xorBitMask = 0x1021;
handle         = SYS_DMA_ChannelAllocate(channel);
SYS_DMA_ChannelCRCSet(handle, crc);
```

Preconditions:

1. DMA should be initialized by calling SYS_DMA_Initialize.
2. Channel should be setup and CRC mode enabled by calling SYS_DMA_ChannelSetup.

Channel Enable/Disable Functions

The Enable/Disable functions allow to enable/disable a channel on the run.

Note: When a data transfer is added by calling SYS_DMA_ChannelTransferAdd, the channel is automatically enabled.

Function Name: SYS_DMA_ChannelEnable

Example

```
SYS_DMA_CHANNEL_HANDLE handle;
DMA_CHANNEL            channel;
channel  = DMA_CHANNEL_2;
handle   = SYS_DMA_ChannelAllocate(channel);
SYS_DMA_ChannelEnable(handle);
```

Precondition: DMA should be initialized by calling SYS_DMA_Initialize.

Function Name: SYS_DMA_ChannelDisable

Example

```
SYS_DMA_CHANNEL_HANDLE  handle;
DMA_CHANNEL             channel;
channel = DMA_CHANNEL_2;
handle  = SYS_DMA_ChannelAllocate(channel);
SYS_DMA_ChannelDisable(handle);
```

Precondition: DMA should be initialized by calling SYS_DMA_Initialize.

Channel Data Transfer Function

This function adds a data transfer to a DMA channel and Enables the channel to start data transfer. The source and the destination addresses, source and destination lengths, The number of bytes transferred per cell event are set and the channel is enabled to start the data transfer.

Function Name: SYS_DMA_ChannelTransferAdd

Example

```
/* Add 10 bytes of data transfer to UART */
SYS_DMA_CHANNEL_HANDLE  handle;
uint8_t                 buf[10];
void                    *srcAddr;
void                    *destAddr;
size_t                  srcSize;
size_t                  destSize;
size_t                  cellSize;
DMA_CHANNEL             channel;
channel  = DMA_CHANNEL_2;
srcAddr  = (uint8_t *) buf;
srcSize  = 10;
destAddr = (uin8_t*) &U2TXREG; //Uart 2 TX register is the DMA destination
destSize = 1;
cellSize = 1;
handle   = SYS_DMA_ChannelAllocate(channel);
SYS_DMA_ChannelTransferAdd(handle,srcAddr,srcSize,destAddr,destSize,cellSize);
```

Preconditions:

1. DMA should be initialized by calling SYS_DMA_Initialize.
2. Channel should be setup and enabled by calling SYS_DMA_ChannelSetup.

Synchronous Data Transfer Start/Abort Functions

These functions allows to force start/abort data transfer on the selected channel.

Note: The DMA should have been set up, transfer added before calling these functions.

Function Name: SYS_DMA_ChannelForceStart

Example

```
SYS_DMA_CHANNEL_HANDLE  handle;
DMA_CHANNEL             channel;
channel = DMA_CHANNEL_2;
handle  = SYS_DMA_ChannelAllocate(channel);
SYS_DMA_ChannelForceStart(handle);
```

Function Name: SYS_DMA_ChannelForceAbort

Example

```
SYS_DMA_CHANNEL_HANDLE  handle;
DMA_CHANNEL             channel;
channel   = DMA_CHANNEL_2;
handle    = SYS_DMA_ChannelAllocate(channel);
SYS_DMA_ChannelForceAbort(handle);
```

Asynchronous Data Transfer Abort Function

This function sets an event source and enables cell transfer abort event for the same for the selected channel.

Note: Asynchronous data transfer is started in general channel setup function. A channel is setup for asynchronous data transfer by default when the appropriate trigger source is specified in general channel setup function call.

Function Name: SYS_DMA_ChannelAbortEventSet

Example

```
SYS_DMA_CHANNEL_HANDLE  handle;
DMA_CHANNEL             channel;
DMA_TRIGGER_SOURCE      eventSrc;
channel  = DMA_CHANNEL_2;
eventSrc = DMA_TRIGGER_CTMU;
handle   = SYS_DMA_ChannelAllocate(channel);
SYS_DMA_ChannelAbortEventSet(handle, eventSrc);
```

Preconditions:

1. DMA should be initialized by calling SYS_DMA_Initialize.

2. Channel should be setup and enabled by calling SYS_DMA_ChannelSetup.

Channel Busy Status Function

This function gets the busy status of the selected channel.

Function Name: SYS_DMA_ChannelIsBusy

Example

```
SYS_DMA_CHANNEL_HANDLE  handle;
DMA_CHANNEL             channel;
bool                    busyStat;
channel  = DMA_CHANNEL_2;
handle   = SYS_DMA_ChannelAllocate(channel);
busyStat = SYS_DMA_ChannelIsBusy(handle);
```

Precondition: DMA should be initialized by calling SYS_DMA_Initialize.

Channel Computed CRC Function

This function gets the computed CRC.

Note: The CRC generator must have been previously set up by calling the CRC mode setup function.

To get the computed CRC value this function must be called after the block transfer completion event.

Function Name: SYS_DMA_ChannelGetCRC

Example

```
SYS_DMA_CHANNEL_HANDLE handle;
DMA_CHANNEL            channel;
uint32_t               computedCRC;
channel     = DMA_CHANNEL_2;
handle      = SYS_DMA_ChannelAllocate(channel);
computedCRC = SYS_DMA_ChannelCRCSet();
```

Precondition: DMA should be initialized by calling SYS_DMA_Initialize.

Channel Transfer Event Handler Set Function

This function allows to set an event handler for the transfer complete, abort or error events

Function Name: SYS_DMA_ChannelTransferEventHandlerSet

Example

```
SYS_DMA_CHANNEL_HANDLE handle;
DMA_CHANNEL channel;
MY_APP_OBJ myAppObj;   //Application specific object
channel = DMA_CHANNEL_2;
```

```
handle = SYS_DMA_ChannelAllocate(channel);
SYS_DMA_ChannelTransferEventHandlerSet(handle, APP_DMASYSTransferEventHandler, (uintptr_t)&myAppObj);

// Event Processing Technique. Event is received when the transfer is processed.

    void APP_DMASYSTransferEventHandler(SYS_DMA_TRANSFER_EVENT event, SYS_DMA_CHANNEL_HANDLE handle,
uintptr_t contextHandle)
    {
        switch(event)
        {
            case SYS_DMA_TRANSFER_EVENT_COMPLETE:
                // This means the data was transferred.
                break;
            case SYS_DMA_TRANSFER_EVENT_ERROR:
                // Error handling here.
                break;
            default:
                break;
        }
    }
```

Precondition: DMA should be initialized by calling SYS_DMA_Initialize.

Channel Error Get Function

This function returns the error associated with the channel access.

Function Name: SYS_DMA_ChannelErrorGet

Example

```
SYS_DMA_CHANNEL_HANDLE handle;
DMA_CHANNEL channel;
channel = DMA_CHANNEL_2;
handle = SYS_DMA_ChannelAllocate(channel);
// Do Channel setup and Transfer Add.
// In the Even Handler Check if there was an Error
if(SYS_DMA_ERROR_ADDRESS_ERROR == SYS_DMA_ChannelErrorGet(handle))
 {
        // There was an address error.
    // Do error handling here.
}
```

Precondition: DMA should be initialized by calling SYS_DMA_Initialize.

Global Control and Status

Global Control and Status Function Usage:

Suspend/Resume Functions

These functions allows an enabled DMA module to suspend DMA operations. Operation of the suspended module can also be resumed using these functions.

Function Name: SYS_DMA_Suspend

Example 1

```
SYS_DMA_Suspend();
```

Precondition: DMA should be initialized by calling SYS_DMA_Initialize.

Function Name: SYS_DMA_Resume

Example 2

```
SYS_DMA_Resume();
```

Precondition: DMA should be initialized by calling SYS_DMA_Initialize.

Busy Status Function

This function gets the busy status of DMA module.

Function Name: SYS_DMA_IsBusy

Example

```
bool busyStat;
busyStat = SYS_DMA_IsBusy();
```

Precondition: DMA should be initialized by calling SYS_DMA_Initialize.

Memory to Memory Transfer

Provides a code example for a memory to memory data transfer.

Description

The following code is a typical usage example of using the DMA System Service for a memory to memory data transfer.

```
/* Destination Memory Address pointer */
uint8_t                     *pDmaSrc;
/* Source Memory Address pointer */
uint8_t                     *pDmaDst;
/* Transfer Size */
size_t                      txferSize;

/* Maximum Transfer Size */
#define     MAX_TRANSFER_SIZE           100
/* The data pattern to be transferred*/
#define     SOURCE_TRANSFER_PATTERN     'W'

 /*****************************************************************
  The below routine sets up a memory to memory data transfer. It also
  registers an event handler which is called when the data transfer
  completes */
void SYS_DMA_TEST_Memory2Memory(void)
{
    uint8_t                     *pSrcTemp;
    uint32_t                    index;
    SYS_DMA_INIT                dmaInit;
    SYS_DMA_CHANNEL_HANDLE      channelHandle;

    /* Initializing source and destination variables */
    pDmaSrc             =   (uint8_t *) NULL;
    pDmaDst             =   (uint8_t *) NULL;
    pSrcTemp            =   (uint8_t *) NULL;
    txferSize           =   MAX_TRANSFER_SIZE;
    pDmaSrc             =   (uint8_t*) malloc(txferSize);
    pDmaDst             =   (uint8_t*) malloc(txferSize);
    pSrcTemp            =   pDmaSrc;


    if(((uint8_t*) NULL != pDmaSrc) && ((uint8_t*) NULL != pDmaDst))
    {
        /* Initialize the source memory block with the pattern */
        for(index=0; index < txferSize; index++)
        {
            *pSrcTemp++= SOURCE_TRANSFER_PATTERN;
        }

        /* Initialize the DMA system service */
        dmaInit.sidl = SYS_DMA_SIDL_DISABLE;
        sysObj = SYS_DMA_Initialize((SYS_MODULE_INIT*)&dmaInit);

        /* Allocate a DMA channel */
        channelHandle = SYS_DMA_ChannelAllocate(DMA_CHANNEL_1);
        if(SYS_DMA_CHANNEL_HANDLE_INVALID != channelHandle)
        {
            /* Register an event handler for the channel */
            SYS_DMA_ChannelTransferEventHandlerSet(channelHandle,
                    Sys_DMA_Mem2Mem_Event_Handler, (uintptr_t)&sysContext);

            /* Setup the channel */
            SYS_DMA_ChannelSetup(channelHandle,
                                 SYS_DMA_CHANNEL_OP_MODE_BASIC,
                                 DMA_TRIGGER_SOURCE_NONE);

            /* Add the memory block transfer request */
            SYS_DMA_ChannelTransferAdd(channelHandle,pDmaSrc,txferSize,
                                       pDmaDst,txferSize,txferSize);
```

```
            /* Start the DMA transfer */
            SYS_DMA_ChannelForceStart(channelHandle);
        }
        else
        {
            /* Channel Handle not available */
            ;
        }

    }
}


/* Below is an Event handler for the registered memory to memory DMA channel */
static void Sys_DMA_Mem2Mem_Event_Handler(SYS_DMA_TRANSFER_EVENT event,
        SYS_DMA_CHANNEL_HANDLE handle, uintptr_t contextHandle)
{
    int32_t fail=0;
    uint32_t index;

    /* Success event */
    if(SYS_DMA_TRANSFER_EVENT_COMPLETE == event)
    {
        /* Verify the contents of destination block matches the
         * memory contents of the source block */
        for(index=0; index < txferSize; index++)
        {
            if(*pDmaSrc++!=*pDmaDst++) // compare the buffers
            {
                fail = -1;
            }
        }
        if(0==fail)
        {
            /* Transfer Success */
        }
        else
        {
            /* Transfer Failed */
        }
    }
    /* Failure Event */
    else if(SYS_DMA_TRANSFER_EVENT_ABORT == event)
    {
    }
}
```

Configuring the Library

The configuration of the DMA System Service is based on the file `system_config.h`

This header file contains the configuration selection for the DMA system service. Based on the selections made, the DMA System Service may support the selected features. These configuration settings will apply to all instances of the DMA System Service.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Direct Memory Access (DMA) System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/dma`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
sys_dma.h	DMA System Service Library API header file.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/sys_dma.c	DMA System Service Library implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for the DMA System Service Library.

Module Dependencies

The DMA System Service Library does not depend on any other modules.

Library Interface

a) Initialization and Task Functions

	Name	Description
	SYS_DMA_Tasks	Maintains the system service's state machine.
	SYS_DMA_TasksError	Maintains the system service's error state machine.
	SYS_DMA_Initialize	Initializes and Enables the DMA Controller.
	SYS_DMA_TasksErrorISR	Maintains the system service's state machine and implements its ISR.
	SYS_DMA_TasksISR	Maintains the system service's state machine and implements its ISR.

b) Channel Setup and Management Functions

	Name	Description
	SYS_DMA_ChannelAbortEventSet	Sets an event source and enables cell transfer abort event for the same for the selected channel.
	SYS_DMA_ChannelAllocate	Allocates the specified DMA channel and returns a handle to it.
	SYS_DMA_ChannelDisable	Disables a channel.
	SYS_DMA_ChannelEnable	Enables a channel.
	SYS_DMA_ChannelForceAbort	Force abort of transfer on the selected channel.
	SYS_DMA_ChannelForceStart	Force start of transfer on the selected channel.
	SYS_DMA_ChannelIsBusy	Returns the busy status of the specified DMA channel.
	SYS_DMA_ChannelRelease	Deallocates and frees the channel specified by the handle.
	SYS_DMA_ChannelSetupMatchAbortMode	Setup the pattern match abort mode.
	SYS_DMA_ChannelTransferAdd	Adds a data transfer to a DMA channel and Enables the channel to start data transfer.
	SYS_DMA_ChannelErrorGet	This function returns the error(if any) associated with the last client request.
	SYS_DMA_ChannelTransferEventHandlerSet	This function allows a DMA system service client to set an event handler.
	SYS_DMA_ChannelCRCGet	Returns the computed CRC.
	SYS_DMA_ChannelCRCSet	Sets up the CRC operation mode.
	SYS_DMA_ChannelSetup	Setup the DMA channel parameters.
	SYS_DMA_ChannelDestinationTransferredSizeGet	Returns the number of bytes transferred to destination.
	SYS_DMA_ChannelSourceTransferredSizeGet	Returns the number of bytes transferred from source.
	SYS_DMA_ChannelTransferSet	Sets up a data transfer to a DMA channel.

c) Global Control and Status Functions

	Name	Description
	SYS_DMA_IsBusy	Returns the busy status of the DMA module.
	SYS_DMA_Resume	Resume DMA operations.
	SYS_DMA_Suspend	Suspend DMA transfers.

d) Data Types and Constants

	Name	Description
	SYS_DMA_CHANNEL_CHAIN_PRIO	Identifies the priority of chaining channel.
	SYS_DMA_CHANNEL_CRC_MODE	Identifies the operation mode of the CRC module.
	SYS_DMA_CHANNEL_OP_MODE	Identifies the available DMA operating modes.
	SYS_DMA_CHANNEL_OPERATION_MODE_CRC	Defines the attributes for CRC operation mode.
	SYS_DMA_CRC_WRITE_ORDER	Identifies altering/maintaining write order post CRC computation.
	SYS_DMA_TRANSFER_EVENT	Enumeration of possible DMA System service events.
	SYS_DMA_CHANNEL_HANDLE	Assigns the handle for requested channel.
	SYS_DMA_CHANNEL_IGNORE_MATCH	Enables/Disables the feature to ignore a byte between a 2-byte match abort pattern.
	SYS_DMA_INIT	Indicates the error information for the DMA System service.
	SYS_DMA_SIDL	Enable/Disable DMA operations in Idle mode.
	SYS_DMA_CHANNEL_COUNT	Identifies the maximum number of DMA channel handles to be defined.
	SYS_DMA_CHANNEL_HANDLE_INVALID	Defines an invalid channel handle.

Description

This section describes the APIs of the DMA System Service Library.

Refer to each section for a detailed description.

a) Initialization and Task Functions

SYS_DMA_Tasks Function

Maintains the system service's state machine.

File

sys_dma.h

C

```
void SYS_DMA_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the DMA system service's internal state machine. This function is specifically designed for non DMA interrupt trigger implementations(polling mode), and should be used only in polling mode. this function should be called from the SYS_Tasks() function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

This function is deprecated. Any calls made to this functions will have no effect.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize.

Example

```
// 'object' Returned from SYS_DMA_Initialize

while (true)
{
    SYS_DMA_Tasks ((object) );
```

```
    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the DMA module (returned from SYS_DMA_Initialize)

Function

void SYS_DMA_Tasks(SYS_MODULE_OBJ object)

SYS_DMA_TasksError Function

Maintains the system service's error state machine.

File

sys_dma.h

C

```
void SYS_DMA_TasksError(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the DMA system service's error state machine. This function is specifically designed for non DMA interrupt trigger implementations(polling mode), and should be used only in polling mode. this function should be called from the SYS_Tasks() function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize.

Example

```
// 'object' Returned from SYS_DMA_Initialize

while (true)
{
    SYS_DMA_TasksError (object);
    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the DMA module (returned from SYS_DMA_Initialize)

Function

void SYS_DMA_TasksError(SYS_MODULE_OBJ object)

SYS_DMA_Initialize Function

Initializes and Enables the DMA Controller.

File

sys_dma.h

C

```
SYS_MODULE_OBJ SYS_DMA_Initialize(const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to the DMA module object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function Enables the DMA module. Enable/Disable stop in idle mode feature based on the passed parameter value.

This function initializes the DMA module making it ready for clients to open and use it. The initialization data is specified by the init parameter.

Remarks

This function must be called before any other DMA systems service routines are called.

Not all features are available on all micro-controllers.

Preconditions

None.

Example

```
SYS_MODULE_OBJ objectHandle;
SYS_DMA_INIT initParam;

initParam.sidl = SYS_DMA_SIDL_ENABLE;
objectHandle = SYS_DMA_Initialize(DRV_I2S_INDEX_1,
                                  (SYS_MODULE_INIT*)initParam);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ SYS_DMA_Initialize(const SYS_MODULE_INIT * const init)

SYS_DMA_TasksErrorISR Function

Maintains the system service's state machine and implements its ISR.

File

sys_dma.h

C

```
void SYS_DMA_TasksErrorISR(SYS_MODULE_OBJ object, DMA_CHANNEL activeChannel);
```

Returns

None.

Description

This function is used to maintain the DMA system service's internal error state machine and implement its ISR for DMA interrupt trigger implementations(interrupt mode). This function is specifically designed for DMA interrupt trigger implementations (interrupt mode). In interrupt mode, this function should be called in the interrupt service routine of the DMA channel that is associated with this transfer.

Remarks

It is called by appropriate raw ISR.

This function may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize.

Example

```
// 'object' Returned from SYS_DMA_Initialize

void __ISR(_DMA3_VECTOR,ipl5) _InterruptHandler_BT_USART_RX_DMA_CHANNEL(void)
{
    // ....
```

```
    SYS_DMA_TasksErrorISR(object, DMA_CHANNEL_3);
}
```

Parameters

Parameters	Description
object	Object handle for the DMA module (returned from SYS_DMA_Initialize)
activeChannel	DMA channel number of the ISR being serviced.

Function

void SYS_DMA_TasksErrorISR(SYS_MODULE_OBJ object, DMA_CHANNEL activeChannel)

SYS_DMA_TasksISR Function

Maintains the system service's state machine and implements its ISR.

File

sys_dma.h

C

```
void SYS_DMA_TasksISR(SYS_MODULE_OBJ object, DMA_CHANNEL activeChannel);
```

Returns

None.

Description

This function is used to maintain the DMA system service's internal state machine and implement its ISR for DMA interrupt trigger implementations(interrupt mode). This function is specifically designed for DMA interrupt trigger implementations (interrupt mode). In interrupt mode, this function should be called in the interrupt service routine of the DMA channel that is associated with this transfer.

Remarks

It is called by appropriate raw ISR.

This function may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize.

Example

```
// 'object' Returned from SYS_DMA_Initialize

void __ISR(_DMA3_VECTOR,ipl5) _InterruptHandler_BT_USART_RX_DMA_CHANNEL(void)
{
    SYS_DMA_TasksISR(object, DMA_CHANNEL_3);
}
```

Parameters

Parameters	Description
object	Object handle for the DMA module (returned from SYS_DMA_Initialize)
activeChannel	DMA channel number of the ISR being serviced.

Function

void SYS_DMA_TasksISR(SYS_MODULE_OBJ object, DMA_CHANNEL activeChannel)

b) Channel Setup and Management Functions

SYS_DMA_ChannelAbortEventSet Function

Sets an event source and enables cell transfer abort event for the same for the selected channel.

File

sys_dma.h

C

```
void SYS_DMA_ChannelAbortEventSet(SYS_DMA_CHANNEL_HANDLE handle, DMA_TRIGGER_SOURCE eventSrc);
```

Returns

None.

Description

This functions enables a cell transfer abort event for the selected source event.

Remarks

If the parameter 'eventSrc' is specified as DMA_TRIGGER_SOURCE_NONE the current DMA transfer will be aborted. The behavior is a same as calling SYS_DMA_ChannelForceAbort.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate. The function SYS_DMA_ChannelSetup must have been called to setup and enable the required features. The function 'SYS_DMA_ChannelTransferAdd' to add a transfer.

Example

```
// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.

// pDmaSrc, pDmaDst is the  source,destination address
// txferSrcSize, txferDesSize is the source,destination transfer size
// cellSize is the cell size
MY_APP_OBJ          myAppObj;
// Client registers an event handler with service. This is done once

SYS_DMA_ChannelTransferEventHandlerSet( handle, APP_DMATransferEventHandle,
                                (uintptr_t)&myAppObj );

SYS_DMA_ChannelSetup(handle, SYS_DMA_CHANNEL_OP_MODE_BASIC,
                             DMA_TRIGGER_SOURCE_NONE);

SYS_DMA_ChannelTransferAdd(handle,pDmaSrc,txferSrcSize,
                           pDmaDst,txferDesSize,cellSize);

SYS_DMA_ChannelAbortEventSet(handle, DMA_TRIGGER_CTMU);

SYS_DMA_ChannelForceStart(handle);

if(SYS_DMA_CHANNEL_HANDLE_INVALID == handle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the transfer is processed.

void APP_DMATransferEventHandle( SYS_DMA_TRANSFER_EVENT event,
        SYS_DMA_CHANNEL_HANDLE handle, uintptr_t contextHandle )
{

    switch(event)
    {
        case SYS_DMA_TRANSFER_EVENT_ABORT:

            // This means the data was transferred.
            break;

        case SYS_DMA_TRANSFER_EVENT_ERROR:

            // Error handling here.
```

```
            if(SYS_DMA_ERROR_ADDRESS_ERROR == DRV_I2S_ErrorGet(myI2SHandle))
            {
                // There was an address error.
                // Do error handling here.
            }

            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the DMA channel as returned by the SYS_DMA_ChannelAllocate function.
eventSrc	The event causing the cell transfer abort

Function

void SYS_DMA_ChannelAbortEventSet
(
SYS_DMA_CHANNEL_HANDLE handle,
DMA_TRIGGER_SOURCE eventSrc
)

SYS_DMA_ChannelAllocate Function

Allocates the specified DMA channel and returns a handle to it.

File

sys_dma.h

C

```
SYS_DMA_CHANNEL_HANDLE SYS_DMA_ChannelAllocate(DMA_CHANNEL channel);
```

Returns

The channel handle for the requested channel number.

If an error occurs, the return value is SYS_DMA_CHANNEL_HANDLE_INVALID. Error can occur.

- if the requested channel number is invalid.
- if the requested channel number is not free.

Description

This function opens the specified DMA channel and provides a handle that must be provided to all other client-level operations to identify the caller and the DMA channel.

Remarks

The handle returned is valid until the SYS_DMA_ChannelRelease routine is called. This function must be called before any other DMA channel Setup and management routines are called

Preconditions

Function SYS_DMA_Initialize must have been called before calling this function.

Example

```
DMA_CHANNEL channel;
SYS_DMA_CHANNEL_HANDLE handle

channel = DMA_CHANNEL_2;
handle = SYS_DMA_ChannelAllocate(channel);
if (SYS_DMA_CHANNEL_HANDLE_INVALID == handle)
{
    // Failed to allocate the channel
}
```

```
else
{
    // Proceed with setting up the channel and adding the transfer
}
```

Parameters

Parameters	Description
channel	Channel number requested for allocation. When channel number specified is DMA_CHANNEL_ANY a random channel is allocated for DMA transfers.

Function

SYS_DMA_CHANNEL_HANDLE SYS_DMA_ChannelAllocate (DMA_CHANNEL channel)

SYS_DMA_ChannelDisable Function

Disables a channel.

File

sys_dma.h

C

```
void SYS_DMA_ChannelDisable(SYS_DMA_CHANNEL_HANDLE handle);
```

Returns

None.

Description

This function disables a channel.

Remarks

None.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate. The function SYS_DMA_ChannelSetup must have been called to setup and enable the required features. A DMA channel should have been enabled either by calling 'SYS_DMA_ChannelTransferAdd' or 'SYS_DMA_ChannelEnable'

Example

```
// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.

SYS_DMA_ChannelDisable(handle);
```

Parameters

Parameters	Description
handle	Handle of the DMA channel as returned by the SYS_DMA_ChannelAllocate function.

Function

void SYS_DMA_ChannelDisable(SYS_DMA_CHANNEL_HANDLE handle)

SYS_DMA_ChannelEnable Function

Enables a channel.

File

sys_dma.h

C

```
void SYS_DMA_ChannelEnable(SYS_DMA_CHANNEL_HANDLE handle);
```

Returns

None.

Description

This function enables a channel.

Remarks

This function may not required to be called when starting DMA setup (by SYS_DMA_ChannelSetup) and transfer Add (by SYS_DMA_ChannelTransferAdd). But may be needed to be called in the registered callback to enable the channel and continue the data transfer with the existing transfer parameters previously set with 'SYS_DMA_ChannelTransferAdd'. The DMA channel is by default disabled on the completion of block transfer(callback generated)

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate. The function SYS_DMA_ChannelSetup must have been called to setup and enable the required features.

Example

```
// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.

SYS_DMA_ChannelEnable(handle);
```

Parameters

Parameters	Description
handle	Handle of the DMA channel as returned by the SYS_DMA_ChannelAllocate function.

Function

void SYS_DMA_ChannelEnable(SYS_DMA_CHANNEL_HANDLE handle)

SYS_DMA_ChannelForceAbort Function

Force abort of transfer on the selected channel.

File

sys_dma.h

C

```
void SYS_DMA_ChannelForceAbort(SYS_DMA_CHANNEL_HANDLE handle);
```

Returns

None.

Description

The function aborts a DMA transfer to occur for the selected channel.

Remarks

This function must be used to abort the DMA transfer when the channel has been setup(by calling SYS_DMA_ChannelSetup) with the eventSrc as DMA_TRIGGER_SOURCE_NONE. and SYS_DMA_ChannelAbortEventSet has not been called.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate. The function SYS_DMA_ChannelSetup must have been called to setup and enable the required features. The function 'SYS_DMA_ChannelTransferAdd' to add a transfer.

Example

```
// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.

// pDmaSrc, pDmaDst is the  source,destination address
// txferSrcSize, txferDesSize is the source,destination transfer size
// cellSize is the cell size

MY_APP_OBJ          myAppObj;
// Client registers an event handler with service. This is done once

SYS_DMA_ChannelTransferEventHandlerSet( handle, APP_DMATransferEventHandle,
```

```
                                    (uintptr_t)&myAppObj );

    SYS_DMA_ChannelSetup(handle, SYS_DMA_CHANNEL_OP_MODE_BASIC,
                                 DMA_TRIGGER_SOURCE_NONE);

    SYS_DMA_ChannelTransferAdd(handle,pDmaSrc,txferSrcSize,
                               pDmaDst,txferDesSize,cellSize);

    SYS_DMA_ChannelForceStart(handle);

    if(SYS_DMA_CHANNEL_HANDLE_INVALID == handle)
    {
        // Error handling here
    }

    ....
    ....
    // Client may need to abort a transfer
    SYS_DMA_ChannelForceAbort(handle);

    // Event Processing Technique. Event is received when
    // the transfer is processed.

    void APP_DMATransferEventHandle( SYS_DMA_TRANSFER_EVENT event,
            SYS_DMA_CHANNEL_HANDLE handle, uintptr_t contextHandle )
    {

        switch(event)
        {
            case SYS_DMA_TRANSFER_EVENT_ABORT:

                // This means the data was transferred.
                break;

            case SYS_DMA_TRANSFER_EVENT_ERROR:

                // Error handling here.

                if(SYS_DMA_ERROR_ADDRESS_ERROR == DRV_I2S_ErrorGet(myI2SHandle))
                {
                    // There was an address error.
                    // Do error handling here.
                }

                break;

            default:
                break;
        }
    }
```

Parameters

Parameters	Description
handle	Handle of the DMA channel as returned by the SYS_DMA_ChannelAllocate function.

Function

void SYS_DMA_ChannelForceAbort(SYS_DMA_CHANNEL_HANDLE handle)

SYS_DMA_ChannelForceStart Function

Force start of transfer on the selected channel.

File

sys_dma.h

C

```
void SYS_DMA_ChannelForceStart(SYS_DMA_CHANNEL_HANDLE handle);
```

Returns

None.

Description

The function force start a DMA transfer to occur for the selected channel.

Remarks

This function must be used to start the DMA transfer when the channel has been setup(by calling SYS_DMA_ChannelSetup) with the eventSrc as DMA_TRIGGER_SOURCE_NONE.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate. The function SYS_DMA_ChannelSetup must have been called to setup and enable the required features. The function 'SYS_DMA_ChannelTransferAdd' to add a transfer.

Example

```
// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.

// pDmaSrc, pDmaDst is the  source,destination address
// txferSrcSize, txferDesSize is the source,destination transfer size
// cellSize is the cell size
MY_APP_OBJ          myAppObj;
// Client registers an event handler with service. This is done once

SYS_DMA_ChannelTransferEventHandlerSet( handle, APP_DMATransferEventHandle,
                                (uintptr_t)&myAppObj );

SYS_DMA_ChannelSetup(handle, SYS_DMA_CHANNEL_OP_MODE_BASIC,
                             DMA_TRIGGER_SOURCE_NONE);

SYS_DMA_ChannelTransferAdd(handle,pDmaSrc,txferSrcSize,
                           pDmaDst,txferDesSize,cellSize);

SYS_DMA_ChannelForceStart(handle);

if(SYS_DMA_CHANNEL_HANDLE_INVALID == handle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the transfer is processed.

void APP_DMATransferEventHandle( SYS_DMA_TRANSFER_EVENT event,
        SYS_DMA_CHANNEL_HANDLE handle, uintptr_t contextHandle )
{

    switch(event)
    {
        case SYS_DMA_TRANSFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case SYS_DMA_TRANSFER_EVENT_ERROR:

            // Error handling here.

            if(SYS_DMA_ERROR_ADDRESS_ERROR == DRV_I2S_ErrorGet(myI2SHandle))
            {
                // There was an address error.
                // Do error handling here.
            }

            break;
```

```
            default:
                break;
        }
    }
```

Parameters

Parameters	Description
handle	Handle of the DMA channel as returned by the SYS_DMA_ChannelAllocate function.

Function

void SYS_DMA_ChannelForceStart(SYS_DMA_CHANNEL_HANDLE handle)

SYS_DMA_ChannelIsBusy Function

Returns the busy status of the specified DMA channel.

File

sys_dma.h

C

```
bool SYS_DMA_ChannelIsBusy(SYS_DMA_CHANNEL_HANDLE handle);
```

Returns

bool - true, if the selected DMA channel is active or enabled

- false, if the selected DMA channel is inactive or disabled

Description

This function returns the busy status of the selected DMA channel

Remarks

This feature may not be available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate.

Example

```
bool                        busyStat;

busyStat    = SYS_DMA_ChannelGetBusy(handle);
```

Parameters

Parameters	Description
handle	Handle of the DMA channel as returned by the SYS_DMA_ChannelAllocate function.

Function

bool SYS_DMA_ChannelIsBusy(SYS_DMA_CHANNEL_HANDLE handle)

SYS_DMA_ChannelRelease Function

Deallocates and frees the channel specified by the handle.

File

sys_dma.h

C

```
void SYS_DMA_ChannelRelease(SYS_DMA_CHANNEL_HANDLE handle);
```

Returns

None.

Description

This function deallocates an allocated-channel of the DMA module, invalidating the handle.

Remarks

After calling this routine, the handle passed in "handle" must not be used with any of the remaining service's routines. A new handle must be obtained by calling SYS_DMA_ChannelAllocate before the caller may use the service again

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate

Example

```
DMA_CHANNEL channel;
SYS_DMA_CHANNEL_HANDLE handle;

channel = DMA_CHANNEL_2;
handle = SYS_DMA_ChannelAllocate(channel);
SYS_DMA_ChannelRelease(handle);
```

Parameters

Parameters	Description
handle	A valid allocated-channel handle, returned from the service's Allocate routine

Function

void SYS_DMA_ChannelRelease(SYS_DMA_CHANNEL_HANDLE handle)

SYS_DMA_ChannelSetupMatchAbortMode Function

Setup the pattern match abort mode.

File

sys_dma.h

C

```
void SYS_DMA_ChannelSetupMatchAbortMode(SYS_DMA_CHANNEL_HANDLE handle, uint16_t pattern, DMA_PATTERN_LENGTH
length, SYS_DMA_CHANNEL_IGNORE_MATCH ignore, uint8_t ignorePattern);
```

Returns

None.

Description

This function sets up the termination of DMA operation when the specified pattern is matched. Additionally on supported parts the function also sets up the ignoring of part of a pattern(8-bit) from match abort pattern(16-bit).

Remarks

The parameter 'pattern' (8-bit or 16-bit) is device-specific. Not all features are available on all devices. Refer to the specific device data sheet for details.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate. The function SYS_DMA_ChannelSetup must be called to enable SYS_DMA_CHANNEL_OP_MODE_MATCH_ABORT before setting up pattern match mode features.

Example

```
// Example 1
// The following code is for a device with an 8-bit pattern value and no
// support for pattern match ignore feature

// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.
uint16_t                        pattern;
DMA_PATTERN_LENGTH              length;
SYS_DMA_CHANNEL_IGNORE_MATCH    ignore;
uint8_t                         ignorePattern;
```

```
pattern         = 0x00; //Stop transfer on detection of a NULL character
length          = DMA_PATTERN_LENGTH_NONE;
ignore          = SYS_DMA_CHANNEL_IGNORE_MATCH_DISABLE;
ignorePattern   = 0;
SYS_DMA_ChannelSetupMatchAbortMode(handle, pattern, length,
                                   ignoreEnable, ignorePattern);

// Example 2
// The following code is for a device with a 16-bit pattern value and
// support for pattern match ignore feature

// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.
uint16_t                        pattern;
DMA_PATTERN_LENGTH              length;
SYS_DMA_CHANNEL_IGNORE_MATCH    ignore;
uint8_t                         ignorePattern;

pattern         = 0x0D0A; //Stop transfer on detection of 'rn'
length          = DMA_PATTERN_MATCH_LENGTH_2BYTES;
ignore          = SYS_DMA_CHANNEL_IGNORE_MATCH_ENABLE;
ignorePattern   = 0x00; \ Any null character between the termination pattern
                        \ 'r' and 'n' is ignored.
SYS_DMA_ChannelSetupMatchAbortMode(handle, pattern, length,
                                       ignore, ignorePattern);
```

Parameters

Parameters	Description
handle	Handle of the DMA channel as returned by the SYS_DMA_ChannelAllocate function.
pattern	The pattern that needs to be matched to abort a DMA transfer.
length	Match pattern length can be 1-byte or 2-byte.
ignore	Enable/Disable a byte between a 2-byte pattern match.
ignorePattern	The part of the pattern(8-bit) that needs to be ignored from the match abort pattern(16-bit)

Function

void SYS_DMA_ChannelSetupMatchAbortMode

(

SYS_DMA_CHANNEL_HANDLE handle,

uint16_t pattern,

DMA_PATTERN_LENGTH length,

SYS_DMA_CHANNEL_IGNORE_MATCH ignore,

uint8_t ignorePattern

)

SYS_DMA_ChannelTransferAdd Function

Adds a data transfer to a DMA channel and Enables the channel to start data transfer.

File

sys_dma.h

C

```
void SYS_DMA_ChannelTransferAdd(SYS_DMA_CHANNEL_HANDLE handle, const void * srcAddr, size_t srcSize, const
void * destAddr, size_t destSize, size_t cellSize);
```

Returns

None.

Description

This function adds a data transfer characteristics for a DMA channel. The The source and the destination addresses, source and destination lengths, The number of bytes transferred per cell event are set. It also enables the channel to start data transfer.

If the requesting client registered an event callback with the service, the service will issue a SYS_DMA_TRANSFER_EVENT_COMPLETE or

SYS_DMA_TRANSFER_EVENT_ABORT event if the transfered was processed successfully of SYS_DMA_TRANSFER_EVENT_ERROR event if the transfer was not processed successfully.

Remarks

For PIC32MZ series of devices, if the source/destination addresses are RAM memory addresses, the the source/destination buffers should be made coherent to avoid the cache coherency issues. For example:

```
uint8_t buffer[1024];
// The buffer can be made coherent by adding the 'coherent' attribute, as follows:
uint8_t __attribute__((coherent)) __attribute__((aligned(16))) buffer[1024];
```

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate. The function SYS_DMA_ChannelSetup must have been called to setup and enable the required features.

Example

```
// Add 10 bytes of data transfer to UART TX

// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.
MY_APP_OBJ          myAppObj;
uint8_t             buf[10];
void                *srcAddr;
void                *destAddr;
size_t              srcSize;
size_t              destSize;
size_t              cellSize;

srcAddr     = (uint8_t *) buf;
srcSize     = 10;
destAddr    = (uin8_t*) &U2TXREG;
destSize    = 1;
cellSize    = 1;

// User registers an event handler with system service. This is done once

SYS_DMA_ChannelTransferEventHandlerSet(handle, APP_DMASYSTransferEventHandler,
                                                    (uintptr_t)&myAppObj);

SYS_DMA_ChannelTransferAdd(handle,srcAddr,srcSize,destAddr,destSize,cellSize);

if(SYS_DMA_CHANNEL_HANDLE_INVALID == handle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the transfer is processed.

void APP_DMASYSTransferEventHandler(SYS_DMA_TRANSFER_EVENT event,
        SYS_DMA_CHANNEL_HANDLE handle, uintptr_t contextHandle)
{
    switch(event)
    {
        case SYS_DMA_TRANSFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case SYS_DMA_TRANSFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the DMA channel as returned by the SYS_DMA_ChannelAllocate function.
srcAddr	Source of the DMA transfer
srcSize	Size of the source
destAddr	Destination of the DMA transfer
destSize	Size of the destination
cellSize	Size of the cell

Function

void SYS_DMA_ChannelTransferAdd
(
SYS_DMA_CHANNEL_HANDLE handle,
const void *srcAddr, size_t srcSize
const void *destAddr, size_t destSize,
size_t cellSize
)

SYS_DMA_ChannelErrorGet Function

This function returns the error(if any) associated with the last client request.

File

sys_dma.h

C

```
SYS_DMA_ERROR SYS_DMA_ChannelErrorGet(SYS_DMA_CHANNEL_HANDLE handle);
```

Returns

A SYS_DMA_ERROR type indicating last known error status.

Description

This function returns the error(if any) associated with the last client request. If the service send a SYS_DMA_TRANSFER_EVENT_ERROR to the client, the client can call this function to know the error cause. The error status will be updated on every operation and should be read frequently (ideally immediately after the service operation has completed) to know the relevant error status.

Remarks

It is the client's responsibility to make sure that the error status is obtained frequently. The service will update the error status regardless of whether this has been examined by the client.

Preconditions

The SYS_DMA_Initialize routine must have been called for the DMA sub system. SYS_DMA_ChannelAllocate must have been called to obtain a valid opened channel handle.

Example

```
// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.

// pDmaSrc, pDmaDst is the  source,destination address
// txferSrcSize, txferDesSize is the source,destination transfer size
// cellSize is the cell size
MY_APP_OBJ          myAppObj;
// Client registers an event handler with service. This is done once

SYS_DMA_ChannelTransferEventHandlerSet( handle, APP_DMATransferEventHandle,
                                (uintptr_t)&myAppObj );

SYS_DMA_ChannelSetup(handle, SYS_DMA_CHANNEL_OP_MODE_BASIC,
                             DMA_TRIGGER_SOURCE_NONE);

SYS_DMA_ChannelTransferAdd(handle,pDmaSrc,txferSrcSize,
```

```
                                pDmaDst,txferDesSize,cellSize);

    SYS_DMA_ChannelForceStart(handle);

    if(SYS_DMA_CHANNEL_HANDLE_INVALID == handle)
    {
        // Error handling here
    }

    // Event Processing Technique. Event is received when
    // the transfer is processed.

    void APP_DMATransferEventHandle( SYS_DMA_TRANSFER_EVENT event,
            SYS_DMA_CHANNEL_HANDLE handle, uintptr_t contextHandle )
    {

        switch(event)
        {
            case SYS_DMA_TRANSFER_EVENT_COMPLETE:

                // This means the data was transferred.
                break;

            case SYS_DMA_TRANSFER_EVENT_ERROR:

                // Error handling here.

                if(SYS_DMA_ERROR_ADDRESS_ERROR == SYS_DMA_ChannelErrorGet(handle))
                {
                    // There was an address error.
                    // Do error handling here.
                }

                break;

            default:
                break;
        }
    }
```

Parameters

Parameters	Description
handle	Handle of the DMA channel as returned by the SYS_DMA_ChannelAllocate function.

Function

SYS_DMA_ERROR SYS_DMA_ChannelErrorGet(SYS_DMA_CHANNEL_HANDLE handle)

SYS_DMA_ChannelTransferEventHandlerSet Function

This function allows a DMA system service client to set an event handler.

File

sys_dma.h

C

```
void SYS_DMA_ChannelTransferEventHandlerSet(SYS_DMA_CHANNEL_HANDLE handle, const
SYS_DMA_CHANNEL_TRANSFER_EVENT_HANDLER eventHandler, const uintptr_t contextHandle);
```

Returns

None.

Description

This function allows a client to set an event handler. The client may want to receive transfer related events in cases when it uses non-blocking read and write functions. The event handler should be set before the client intends to perform operations that could generate events.

This function accepts a contextHandle parameter. This parameter could be set by the client to contain (or point to) any client specific data object that should be associated with this DMA channel.

Remarks

None.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate.

Example

```
// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.
MY_APP_OBJ          myAppObj;
uint8_t             buf[10];
void                *srcAddr;
void                *destAddr;
size_t              srcSize;
size_t              destSize;
size_t              cellSize;

srcAddr     = (uint8_t *) buf;
srcSize     = 10;
destAddr    = (uin8_t*) &U2TXREG;
destSize    = 1;
cellSize    = 1;
channelHandle = SYS_DMA_ChannelAllocate(channel);
// User registers an event handler with system service. This is done once

SYS_DMA_ChannelTransferEventHandlerSet(handle, APP_DMASYSTransferEventHandler,
                                                        (uintptr_t)&myAppObj);

SYS_DMA_ChannelTransferAdd(handle,srcAddr,srcSize,destAddr,destSize,cellSize);

if(SYS_DMA_CHANNEL_HANDLE_INVALID == handle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the transfer is processed.

void APP_DMASYSTransferEventHandler(SYS_DMA_TRANSFER_EVENT event,
        SYS_DMA_CHANNEL_HANDLE handle, uintptr_t contextHandle)
{
    switch(event)
    {
        case SYS_DMA_TRANSFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case SYS_DMA_TRANSFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid channel handle, returned from the system service's Allocate routine
eventHandler	Pointer to the event handler function.
contextHandle	Value identifying the context of the application/driver/middleware that registered the event handling function.

Function

void SYS_DMA_ChannelTransferEventHandlerSet
(
SYS_DMA_CHANNEL_HANDLE handle,
const SYS_DMA_CHANNEL_TRANSFER_EVENT_HANDLER eventHandler,
const uintptr_t contextHandle
)

SYS_DMA_ChannelCRCGet Function

Returns the computed CRC.

File

sys_dma.h

C

```
uint32_t SYS_DMA_ChannelCRCGet();
```

Returns

uint32_t - The Computed CRC.

Description

This function returns the computed CRC

Remarks

To get the computed CRC value this function must be called after the block transfer completion event (i.e., after getting and processing the callback registered with SYS_DMA_ChannelTransferEventHandlerSet).

This feature may not be available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate. The function SYS_DMA_ChannelSetup must be called to enable SYS_DMA_CHANNEL_OP_MODE_CRC before setting up the CRC mode. The CRC generator must have been previously setup using SYS_DMA_ChannelCRCSet.

Example

```
uint32_t                    computedCRC;

computedCRC     = SYS_DMA_ChannelCRCGet();
```

Function

uint32_t SYS_DMA_ChannelCRCGet(void)

SYS_DMA_ChannelCRCSet Function

Sets up the CRC operation mode.

File

sys_dma.h

C

```
void SYS_DMA_ChannelCRCSet(SYS_DMA_CHANNEL_HANDLE handle, SYS_DMA_CHANNEL_OPERATION_MODE_CRC crc);
```

Returns

None.

Description

This function sets up the CRC computation features.

Remarks

This feature may not be available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate. The function SYS_DMA_ChannelSetup must be called to enable SYS_DMA_CHANNEL_OP_MODE_CRC before setting up the CRC mode.

Example

```
//Example 1
// DMA calculation using the CRC background mode

// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.
SYS_DMA_CHANNEL_OPERATION_MODE_CRC  crc;

crc.type            = DMA_CRC_LFSR;
crc.mode            = SYS_DMA_CHANNEL_CRC_MODE_BACKGROUND;
crc.polyLength      = 16;
crc.bitOrder        = DMA_CRC_BIT_ORDER_LSB;
crc.byteOrder       = DMA_CRC_BYTEORDER_NO_SWAPPING;
crc.writeOrder      = SYS_DMA_CRC_WRITE_ORDER_MAINTAIN;
crc.data            = 0xFFFF;
crc.xorBitMask      = 0x1021;
SYS_DMA_ChannelCRCSet(handle, crc);
```

Parameters

Parameters	Description
handle	Handle of the DMA channel as returned by the SYS_DMA_ChannelAllocate function.
crc.type	CRC will calculate an IP header checksum or an LFSR CRC.
crc.mode	Compute the CRC in Background/Append mode.
crc.polyLength	Denotes the length of the polynomial.
crc.bitOrder	CRC is calculated LSb/MSb first.
crc.byteOrder	Byte selection order input pre-CRC Generator
crc.writeOrder	Write byte order selection post-CRC computation
crc.data	Computed/Seed CRC
crc.xorBitMask	Enable/Disable XOR bit mask on the corresponding bits when mode is LFSR

Function

void SYS_DMA_ChannelCRCSet

(

SYS_DMA_CHANNEL_HANDLE handle,

SYS_DMA_CHANNEL_OPERATION_MODE_CRC crc

)

SYS_DMA_ChannelSetup Function

Setup the DMA channel parameters.

File

sys_dma.h

C

```
void SYS_DMA_ChannelSetup(SYS_DMA_CHANNEL_HANDLE handle, SYS_DMA_CHANNEL_OP_MODE modeEnable,
DMA_TRIGGER_SOURCE eventSrc);
```

Returns

None.

Description

This function sets up the DMA channel parameters. It sets the channel priority and enables the mode of operations for the current system design.

Remarks

If SYS_DMA_CHANNEL_OP_MODE_MATCH_ABORT, SYS_DMA_CHANNEL_OP_MODE_CHAIN or SYS_DMA_CHANNEL_OP_MODE_CRC

mode of operation is enabled, then corresponding mode setup API's needs to be called to set the related parameters.

If the parameter 'eventSrc' is specified as DMA_TRIGGER_SOURCE_NONE then SYS_DMA_ChannelForceStart must be called to start the DMA channel transfer.

Not all features are available on all microcontrollers.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate.

Example

```
    // 'handle' is a valid handle returned
    // by the SYS_DMA_ChannelAllocate function.

SYS_DMA_CHANNEL_OP_MODE     modeEnable;
DMA_TRIGGER_SOURCE          eventSrc;

channel         =   DMA_CHANNEL_2;
modeEnable      =   (SYS_DMA_CHANNEL_OP_MODE_BASIC | SYS_DMA_CHANNEL_OP_MODE_CRC);
eventSrc        =   DMA_TRIGGER_USART_1_TRANSMIT;
 // Setup channel number, and enables basic and CRC mode
SYS_DMA_ChannelSetup(handle, modeEnable,eventSrc);
```

Parameters

Parameters	Description
handle	Handle of the DMA channel as returned by the SYS_DMA_ChannelAllocate function.
priority	The priority to be associated to the channel.
modeEnable	The supported operating modes to be enabled. This parameter can be logically ORed to specify multiple options.
eventSrc	The event causing the cell transfer start.

Function

void SYS_DMA_ChannelSetup

(

SYS_DMA_CHANNEL_HANDLE handle,

SYS_DMA_CHANNEL_OP_MODE modeEnable

DMA_TRIGGER_SOURCE eventSrc

)

SYS_DMA_ChannelDestinationTransferredSizeGet Function

Returns the number of bytes transferred to destination.

File

sys_dma.h

C

```
size_t SYS_DMA_ChannelDestinationTransferredSizeGet(SYS_DMA_CHANNEL_HANDLE handle);
```

Returns

size_t - Returns the number of bytes received from the submitted size.

Description

When a data transfer request is submitted. At any time while the transmission is in progress the size of the amount of data transferred to destination can be known by calling this function.

Remarks

None.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate. Data transfer should have been initiated by calling SYS_DMA_ChannelTransferAdd or SYS_DMA_ChannelTransferSet.

Example

```
// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.
MY_APP_OBJ          myAppObj;
uint8_t             buf[10];
void                *srcAddr;
void                *destAddr;
size_t              srcSize;
size_t              destSize;
size_t              cellSize;
size_t              transferredSize;

srcAddr     = (uin8_t*) &U2RXREG;
srcSize     = 1;
destAddr    = (uint8_t *) buf ;
destSize    = 10;
cellSize    = 1;
channelHandle = SYS_DMA_ChannelAllocate(channel);
// User registers an event handler with system service. This is done once

SYS_DMA_ChannelTransferEventHandlerSet(handle, APP_DMASYSTransferEventHandler,
                                                    (uintptr_t)&myAppObj);

SYS_DMA_ChannelTransferAdd(handle,srcAddr,srcSize,destAddr,destSize,cellSize);

if(SYS_DMA_CHANNEL_HANDLE_INVALID == handle)
{
    // Error handling here
}

// The data is being transferred after adding the transfer to the DMA channel.
// The user can get to know dynamically the amount of data
// transmitted to destination by calling SYS_DMA_ChannelDestinationTransferredSizeGet
transferredSize = SYS_DMA_ChannelDestinationTransferredSizeGet(channelHandle);
```

Parameters

Parameters	Description
handle	A valid channel handle, returned from the system service's Allocate routine

Function

size_t SYS_DMA_ChannelDestinationTransferredSizeGet(SYS_DMA_CHANNEL_HANDLE handle)

SYS_DMA_ChannelSourceTransferredSizeGet Function

Returns the number of bytes transferred from source.

File

sys_dma.h

C

```
size_t SYS_DMA_ChannelSourceTransferredSizeGet(SYS_DMA_CHANNEL_HANDLE handle);
```

Returns

size_t - Returns the number of bytes transferred from the submitted size.

Description

When a data transfer request is submitted. At any time while the transmission is in progress the size of the amount of data transferred from source can be known by calling this function.

Remarks

None.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling

SYS_DMA_ChannelAllocate. Data transfer should have been initiated by calling SYS_DMA_ChannelTransferAdd or SYS_DMA_ChannelTransferSet.

Example

```
// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.
MY_APP_OBJ          myAppObj;
uint8_t             buf[10];
void                *srcAddr;
void                *destAddr;
size_t              srcSize;
size_t              destSize;
size_t              cellSize;
size_t              transferredSize;

srcAddr     = (uint8_t *) buf;
srcSize     = 10;
destAddr    = (uin8_t*) &U2TXREG;
destSize    = 1;
cellSize    = 1;
channelHandle = SYS_DMA_ChannelAllocate(channel);
// User registers an event handler with system service. This is done once

SYS_DMA_ChannelTransferEventHandlerSet(handle, APP_DMASYSTransferEventHandler,
                                                    (uintptr_t)&myAppObj);

SYS_DMA_ChannelTransferAdd(handle,srcAddr,srcSize,destAddr,destSize,cellSize);

if(SYS_DMA_CHANNEL_HANDLE_INVALID == handle)
{
    // Error handling here
}

// The data is being transferred after adding the transfer to the DMA channel.
// The user can get to know dynamically the amount of data
// transmitted from source by calling SYS_DMA_ChannelSourceTransferredSizeGet
transferredSize = SYS_DMA_ChannelSourceTransferredSizeGet(channelHandle);
```

Parameters

Parameters	Description
handle	A valid channel handle, returned from the system service's Allocate routine

Function

size_t SYS_DMA_ChannelSourceTransferredSizeGet(SYS_DMA_CHANNEL_HANDLE handle)

SYS_DMA_ChannelTransferSet Function

Sets up a data transfer to a DMA channel.

File

sys_dma.h

C

```
void SYS_DMA_ChannelTransferSet(SYS_DMA_CHANNEL_HANDLE handle, const void * srcAddr, size_t srcSize, const
void * destAddr, size_t destSize, size_t cellSize);
```

Returns

None.

Description

This function sets up data transfer characteristics for a DMA channel. The The source and the destination addresses, source and destination lengths, The number of bytes transferred per cell event are set. This function does not enables the DMA channel. The channel has to be explicitly enabled to start the data transfer. The above functions could be used in situations where in the user intends to setup transfer parameters but do not intend to enable the channel immediately. For example to chain to DMA channels in a cyclic order where the channels remains disabled. The channel is enabled automatically by the DMA when the transfer trigger condition occurs.

Remarks

For PIC32MZ series of devices, if the source/destination addresses are RAM memory addresses, the the source/destination buffers should be made coherent to avoid the cache coherency issues. For example:

```
uint8_t buffer[1024];
// The buffer can be made coherent by adding the 'coherent' attribute, as follows:
uint8_t __attribute__((coherent)) __attribute__((aligned(16))) buffer[1024];
```

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize. DMA channel should have been allocated by calling SYS_DMA_ChannelAllocate. The function SYS_DMA_ChannelSetup must have been called to setup and enable the required features.

Example

```
// Set up 10 bytes of data transfer to UART TX

// 'handle' is a valid handle returned
// by the SYS_DMA_ChannelAllocate function.
MY_APP_OBJ          myAppObj;
uint8_t             buf[10];
void                *srcAddr;
void                *destAddr;
size_t              srcSize;
size_t              destSize;
size_t              cellSize;

srcAddr     = (uint8_t *) buf;
srcSize     = 10;
destAddr    = (uin8_t*) &U2TXREG;
destSize    = 1;
cellSize    = 1;

// User registers an event handler with system service. This is done once

SYS_DMA_ChannelTransferEventHandlerSet(handle, APP_DMASYSTransferEventHandler,
                                                        (uintptr_t)&myAppObj);

SYS_DMA_ChannelTransferSet(handle,srcAddr,srcSize,destAddr,destSize,cellSize);
SYS_DMA_ChannelEnable(handle);

if(SYS_DMA_CHANNEL_HANDLE_INVALID == handle)
{
    // Error handling here
}

// Event Processing Technique. Event is received when
// the transfer is processed.

void APP_DMASYSTransferEventHandler(SYS_DMA_TRANSFER_EVENT event,
        SYS_DMA_CHANNEL_HANDLE handle, uintptr_t contextHandle)
{
    switch(event)
    {
        case SYS_DMA_TRANSFER_EVENT_COMPLETE:

            // This means the data was transferred.
            break;

        case SYS_DMA_TRANSFER_EVENT_ERROR:

            // Error handling here.
            break;

        default:
            break;
    }
}
```

Parameters

Parameters	Description
handle	Handle of the DMA channel as returned by the SYS_DMA_ChannelAllocate function.
srcAddr	Source of the DMA transfer
srcSize	Size of the source
destAddr	Destination of the DMA transfer
destSize	Size of the destination
cellSize	Size of the cell

Function

void SYS_DMA_ChannelTransferSet
(
SYS_DMA_CHANNEL_HANDLE handle,
const void *srcAddr, size_t srcSize
const void *destAddr, size_t destSize,
size_t cellSize
)

c) Global Control and Status Functions

SYS_DMA_IsBusy Function

Returns the busy status of the DMA module.

File

sys_dma.h

C

```
bool SYS_DMA_IsBusy();
```

Returns

Boolean

- true - The DMA module is active
- false - The DMA module is inactive and disabled

```
bool busyStat;
busyStat = SYS_DMA_IsBusy();
```

Description

This function returns the busy status of the DMA module

Remarks

This feature may not be available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize.

Function

bool SYS_DMA_IsBusy(void)

SYS_DMA_Resume Function

Resume DMA operations.

File

sys_dma.h

C

```
void SYS_DMA_Resume();
```

Returns

None.

Description

This function disables DMA suspend. It resumes the DMA operation suspended by calling SYS_DMA_Suspend. The DMA operates normally.

Remarks

None

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize.

Example

```
SYS_DMA_Resume();
```

Function

void SYS_DMA_Resume(void)

SYS_DMA_Suspend Function

Suspend DMA transfers.

File

sys_dma.h

C

```
void SYS_DMA_Suspend();
```

Returns

None.

Description

This function suspends DMA transfers to allow CPU uninterrupted access to data bus

Remarks

None

Preconditions

DMA should have been initialized by calling SYS_DMA_Initialize.

Example

```
SYS_DMA_Suspend();
```

Function

void SYS_DMA_Suspend(void)

d) Data Types and Constants

SYS_DMA_CHANNEL_CHAIN_PRIO Enumeration

Identifies the priority of chaining channel.

File

sys_dma.h

C

```
typedef enum {
  SYS_DMA_CHANNEL_CHAIN_PRIO_HIGH,
  SYS_DMA_CHANNEL_CHAIN_PRIO_LOW
} SYS_DMA_CHANNEL_CHAIN_PRIO;
```

Members

Members	Description
SYS_DMA_CHANNEL_CHAIN_PRIO_HIGH	Chain to channel higher in natural priority
SYS_DMA_CHANNEL_CHAIN_PRIO_LOW	Chain to channel lower in natural priority

Description

DMA Channel chaining priority

This data type identifies the priority of chaining channel.

Channel chaining priority identifies the channel that will be enabled on completion of block data transfer on the master channel.

Remarks

None.

SYS_DMA_CHANNEL_CRC_MODE Enumeration

Identifies the operation mode of the CRC module.

File

sys_dma.h

C

```
typedef enum {
  SYS_DMA_CHANNEL_CRC_MODE_BACKGROUND,
  SYS_DMA_CHANNEL_CRC_MODE_APPEND
} SYS_DMA_CHANNEL_CRC_MODE;
```

Members

Members	Description
SYS_DMA_CHANNEL_CRC_MODE_BACKGROUND	DMA reads the data from the source, passes it through the CRC module and writes it to the destination. the calculated CRC is left in the DCRCDATA register at the end of the block transfer.
SYS_DMA_CHANNEL_CRC_MODE_APPEND	DMA only feeds source data to the CRC module. it does not write source data to the destination address. When a block transfer complete or pattern abort event occur, The DMA writes the CRC value to the destination address

Description

DMA channel CRC mode

This data type identifies CRC operating mode. The CRC computation can be in background mode or append mode.

Remarks

None.

SYS_DMA_CHANNEL_OP_MODE Enumeration

Identifies the available DMA operating modes.

File

sys_dma.h

C

```
typedef enum {
  SYS_DMA_CHANNEL_OP_MODE_BASIC,
  SYS_DMA_CHANNEL_OP_MODE_MATCH_ABORT,
  SYS_DMA_CHANNEL_OP_MODE_CHAIN_HIGH,
  SYS_DMA_CHANNEL_OP_MODE_CHAIN_LOW,
  SYS_DMA_CHANNEL_OP_MODE_AUTO,
  SYS_DMA_CHANNEL_OP_MODE_CRC
} SYS_DMA_CHANNEL_OP_MODE;
```

Members

Members	Description
SYS_DMA_CHANNEL_OP_MODE_BASIC	The normal DMA operation mode. The DMA channel will transfer data from a source to a destination without CPU intervention
SYS_DMA_CHANNEL_OP_MODE_MATCH_ABORT	Pattern Match abort mode allows the user to end a transfer if data written during a transaction matches a specific pattern, as defined by the DCHxDAT register
SYS_DMA_CHANNEL_OP_MODE_CHAIN_HIGH	Channel chaining operating mode enhances the normal DMA channel operations. A channel(slave) can be chained to an adjacent channel(master). The slave channel will be enabled when a block transfer of the master channel completes. only one of the below two chaining priorities must be provided. Chaining priority High.
SYS_DMA_CHANNEL_OP_MODE_CHAIN_LOW	Chaining priority Low
SYS_DMA_CHANNEL_OP_MODE_AUTO	Auto enable operating mode allows a channel to be kept active, even if a block transfer completes or pattern match occurs. This prevents the user from having to re-enable the channel each time a block transfer completes.
SYS_DMA_CHANNEL_OP_MODE_CRC	CRC operating mode allows computation of 16, 32-bit CRC.

Description

DMA channel operation modes

This data type Identifies the available DMA operating modes.

The supported DMA modes are not mutually exclusive, but can be simultaneously operational.

Remarks

These supported DMA modes can be logically OR'd together. They are passed into the SYS_DMA_ChannelSetup function through the modeEnable parameter.

SYS_DMA_CHANNEL_OPERATION_MODE_CRC Structure

Defines the attributes for CRC operation mode.

File

sys_dma.h

C

```
typedef struct {
  DMA_CRC_TYPE type;
  SYS_DMA_CHANNEL_CRC_MODE mode;
  uint8_t polyLength;
  DMA_CRC_BIT_ORDER bitOrder;
  DMA_CRC_BYTE_ORDER byteOrder;
  SYS_DMA_CRC_WRITE_ORDER writeOrder;
  uint32_t data;
  uint32_t xorBitMask;
} SYS_DMA_CHANNEL_OPERATION_MODE_CRC;
```

Members

Members	Description
DMA_CRC_TYPE type;	CRC type (PLIB-level). Type Selection identifies whether the CRC module will calculate an IP header checksum or an LFSR CRC
SYS_DMA_CHANNEL_CRC_MODE mode;	CRC mode, Background or Append
uint8_t polyLength;	Polynomial length, This value is redundant when the selected CRC type is IP header checksum. When the CRC type is LFSR, Identifies the length of the polynomial.
DMA_CRC_BIT_ORDER bitOrder;	Bit order (PLIB-level). Identifies whether the CRC is computed LSb or MSb first
DMA_CRC_BYTE_ORDER byteOrder;	Byte order (PLIB-level). Identifies the byte selection order input pre-CRC Generator.
SYS_DMA_CRC_WRITE_ORDER writeOrder;	Post CRC Write order
uint32_t data;	CRC data feed: While enabling the CRC mode, this field gives the seed for the CRC computation. On block transfer complete or pattern match abort the field have the computed CRC.
uint32_t xorBitMask;	XOR bit mask, This value is redundant when the selected CRC type is IP header checksum. When the CRC type is LFSR, Identifies the XOR bit mask input to the shift register during CRC computation.

Description

DMA channel CRC mode

This data type defines the attributes for CRC operation mode.

Remarks

This feature may not be available on all devices. Refer to the specific device data sheet to determine availability.

SYS_DMA_CRC_WRITE_ORDER Enumeration

Identifies altering/maintaining write order post CRC computation.

File

sys_dma.h

C

```
typedef enum {
  SYS_DMA_CRC_WRITE_ORDER_MAINTAIN,
  SYS_DMA_CRC_WRITE_ORDER_CHANGE
} SYS_DMA_CRC_WRITE_ORDER;
```

Members

Members	Description
SYS_DMA_CRC_WRITE_ORDER_MAINTAIN	Source data is written to the destination unchanged regardless of the selected pre-CRC byte order.
SYS_DMA_CRC_WRITE_ORDER_CHANGE	Source data is written to the destination reordered according to the selected pre-CRC byte order.

Description

DMA channel CRC write order

This data type identifies write byte order selection post CRC computation.

Remarks

None.

SYS_DMA_TRANSFER_EVENT Enumeration

Enumeration of possible DMA System service events.

File

sys_dma.h

C

```
typedef enum {
  SYS_DMA_TRANSFER_EVENT_COMPLETE,
  SYS_DMA_TRANSFER_EVENT_ERROR,
  SYS_DMA_TRANSFER_EVENT_ABORT
} SYS_DMA_TRANSFER_EVENT;
```

Members

Members	Description
SYS_DMA_TRANSFER_EVENT_COMPLETE	Data was transferred successfully.
SYS_DMA_TRANSFER_EVENT_ERROR	Error while processing the request
SYS_DMA_TRANSFER_EVENT_ABORT	Data transfer was aborted.

Description

DMA System service Events

This data type provides an enumeration of all possible DMA System service events.

Remarks

None.

SYS_DMA_CHANNEL_HANDLE Type

Assigns the handle for requested channel.

File

sys_dma.h

C

```
typedef uintptr_t SYS_DMA_CHANNEL_HANDLE;
```

Description

DMA System Service Channel Handle

A channel handle value is returned by a call to the SYS_DMA_ChannelAllocate function. This handle is associated with the channel number that is used for data transfer and it allows the application to track the data transfer.

The channel handle once assigned to a client expires when the client calls SYS_DMA_ChannelRelease. exits.

Remarks

None.

SYS_DMA_CHANNEL_IGNORE_MATCH Enumeration

Enables/Disables the feature to ignore a byte between a 2-byte match abort pattern.

File

sys_dma.h

C

```
typedef enum {
  SYS_DMA_CHANNEL_IGNORE_MATCH_DISABLE,
  SYS_DMA_CHANNEL_IGNORE_MATCH_ENABLE
} SYS_DMA_CHANNEL_IGNORE_MATCH;
```

Members

Members	Description
SYS_DMA_CHANNEL_IGNORE_MATCH_DISABLE	Disabling the feature to ignore a byte between a 2-byte match abort pattern.
SYS_DMA_CHANNEL_IGNORE_MATCH_ENABLE	Enable the feature to ignore a byte between a 2-byte match abort pattern.

Description

DMA Channel ignore pattern match

This data type allows enabling/disabling the feature to ignore a byte between a 2-byte match abort pattern.

Remarks

This feature may not be available on all devices. Refer to the specific device data sheet to determine availability.

SYS_DMA_INIT Enumeration

Indicates the error information for the DMA System service.

File

sys_dma.h

C

```
typedef enum {
} SYS_DMA_INIT;
```

Description

DMA System service Error

This enumeration indicates the error information for the DMA System service.

Remarks

None.

SYS_DMA_SIDL Enumeration

Enable/Disable DMA operations in Idle mode.

File

sys_dma.h

C

```
typedef enum {
  SYS_DMA_SIDL_DISABLE,
  SYS_DMA_SIDL_ENABLE
} SYS_DMA_SIDL;
```

Members

Members	Description
SYS_DMA_SIDL_DISABLE	When the CPU enters idle mode, the DMA module continue operations
SYS_DMA_SIDL_ENABLE	When the CPU enters idle mode, the DMA module will discontinue operations

Description

DMA stop in idle mode

This data type allows enabling/disabling of DMA operations in idle mode.

Remarks

The field 'sidl' (in init data structure 'SYS_DMA_INIT') is enable/disable while initializing the DMA module by calling SYS_DMA_Initialize.

This feature may not be available on all devices. Refer to the specific device data sheet to determine availability.

SYS_DMA_CHANNEL_COUNT Macro

Identifies the maximum number of DMA channel handles to be defined.

File

sys_dma.h

C

```
#define SYS_DMA_CHANNEL_COUNT DMA_NUMBER_OF_CHANNELS
```

Description

DMA System Service Channel Count

This constant identifies the maximum number of DMA channel handles that should be defined.

Remarks

This value is derived from device-specific header files defined as part of the peripheral libraries.

SYS_DMA_CHANNEL_HANDLE_INVALID Macro

Defines an invalid channel handle.

File

sys_dma.h

C

```
#define SYS_DMA_CHANNEL_HANDLE_INVALID ((SYS_DMA_CHANNEL_HANDLE)(-1))
```

Description

DMA System Service Invalid Channel Handle

This is the definition of an invalid channel handle. An invalid buffer handle is returned by SYS_DMA_ChannelAllocate function if the channel number request was not successful.

Remarks

None.

Files

Files

Name	Description
sys_dma.h	DMA System Service.

Description

This section lists the source and header files used by the library.

sys_dma.h

DMA System Service.

Enumerations

	Name	Description
	SYS_DMA_CHANNEL_CHAIN_PRIO	Identifies the priority of chaining channel.
	SYS_DMA_CHANNEL_CRC_MODE	Identifies the operation mode of the CRC module.
	SYS_DMA_CHANNEL_IGNORE_MATCH	Enables/Disables the feature to ignore a byte between a 2-byte match abort pattern.
	SYS_DMA_CHANNEL_OP_MODE	Identifies the available DMA operating modes.
	SYS_DMA_CRC_WRITE_ORDER	Identifies altering/maintaining write order post CRC computation.
	SYS_DMA_INIT	Indicates the error information for the DMA System service.
	SYS_DMA_SIDL	Enable/Disable DMA operations in Idle mode.
	SYS_DMA_TRANSFER_EVENT	Enumeration of possible DMA System service events.

Functions

	Name	Description
	SYS_DMA_ChannelAbortEventSet	Sets an event source and enables cell transfer abort event for the same for the selected channel.
	SYS_DMA_ChannelAllocate	Allocates the specified DMA channel and returns a handle to it.
	SYS_DMA_ChannelCRCGet	Returns the computed CRC.
	SYS_DMA_ChannelCRCSet	Sets up the CRC operation mode.
	SYS_DMA_ChannelDestinationTransferredSizeGet	Returns the number of bytes transferred to destination.
	SYS_DMA_ChannelDisable	Disables a channel.
	SYS_DMA_ChannelEnable	Enables a channel.
	SYS_DMA_ChannelErrorGet	This function returns the error(if any) associated with the last client request.
	SYS_DMA_ChannelForceAbort	Force abort of transfer on the selected channel.
	SYS_DMA_ChannelForceStart	Force start of transfer on the selected channel.
	SYS_DMA_ChannelIsBusy	Returns the busy status of the specified DMA channel.
	SYS_DMA_ChannelRelease	Deallocates and frees the channel specified by the handle.
	SYS_DMA_ChannelSetup	Setup the DMA channel parameters.
	SYS_DMA_ChannelSetupMatchAbortMode	Setup the pattern match abort mode.
	SYS_DMA_ChannelSourceTransferredSizeGet	Returns the number of bytes transferred from source.
	SYS_DMA_ChannelTransferAdd	Adds a data transfer to a DMA channel and Enables the channel to start data transfer.
	SYS_DMA_ChannelTransferEventHandlerSet	This function allows a DMA system service client to set an event handler.
	SYS_DMA_ChannelTransferSet	Sets up a data transfer to a DMA channel.
	SYS_DMA_Initialize	Initializes and Enables the DMA Controller.
	SYS_DMA_IsBusy	Returns the busy status of the DMA module.
	SYS_DMA_Resume	Resume DMA operations.
	SYS_DMA_Suspend	Suspend DMA transfers.
	SYS_DMA_Tasks	Maintains the system service's state machine.
	SYS_DMA_TasksError	Maintains the system service's error state machine.
	SYS_DMA_TasksErrorISR	Maintains the system service's state machine and implements its ISR.
	SYS_DMA_TasksISR	Maintains the system service's state machine and implements its ISR.

Macros

	Name	Description
	SYS_DMA_CHANNEL_COUNT	Identifies the maximum number of DMA channel handles to be defined.
	SYS_DMA_CHANNEL_HANDLE_INVALID	Defines an invalid channel handle.

Structures

	Name	Description
	SYS_DMA_CHANNEL_OPERATION_MODE_CRC	Defines the attributes for CRC operation mode.

Types

	Name	Description
	SYS_DMA_CHANNEL_HANDLE	Assigns the handle for requested channel.

Description

DMA System Service Library Interface Definition

This file contains the interface definition for the DMA System Service. It provides a way to interact with the DMA subsystem to manage the data transfer between different peripherals and/or memory without intervention from the CPU.

File Name

sys_dma.h

Company

Microchip Technology Inc.

File System Service Library

This section describes the File System Service Library.

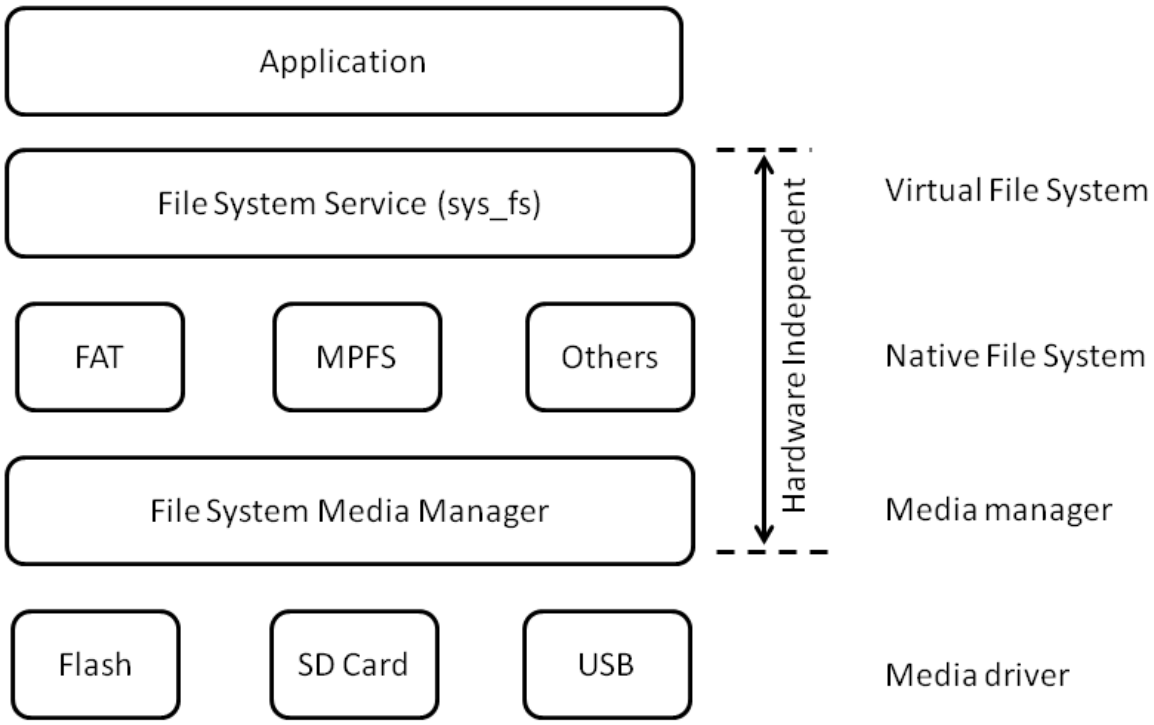
Introduction

Introduction to the MPLAB Harmony File System (FS).

Description

The MPLAB Harmony File System (FS) provides file system services to MPLAB Harmony based applications. The architecture of the File System Service is shown in the following figure.

File System Architecture



The File System Service provides an application programming interface (API) through which a utility or user program requests services of a file system. Some file system APIs may also include interfaces for maintenance operations, such as creating or initializing a file system and verifying the file system for integrity.

The File System service is really a framework designed to support multiple file systems (native file system) and multiple media in the same application. Examples of native file systems are FAT12,FAT16, FAT32, and the Microchip File System (MPFS) among others. MPFS is a read-only file system, which is used to store the files that do not change for example Web pages, images, etc. Each of these native file systems have a common set of APIs that can be used to access the files of that particular native file system.

The File System Service abstracts the native file system calls and provides a common interface to the user/application layer. For example, while the application layer requests for a file read or write from a disk, due to the presence of the this abstraction, the application need not be bothered about the native file system implemented on that disk. Instead, the application can call the read/write API of the File System, which in turn translates to the read/write command of the native file system used on the required disk.

This simplifies the implementation of the higher application layer and also provides a mechanism to add more native file system to the File System framework in the future.

Note: "File System Service" and "sys_fs" are synonymous.

File System Service Library Porting Guide

This section provides information for porting an existing File System (FS) application that uses legacy MLA code to MPLAB Harmony.

Description

Differences between the legacy MLA code and MPLAB Harmony are described and examples are provided in the following topics:

- Initialization
- System Configuration
- Mounting a Volume
- Opening a File
- Reading a File
- Writing a File
- Closing a File
- File EOF
- File Tell
- File Seek
- SYS_FS_Tasks

Comparison of API Names

This topic lists the API names for MLA versus the API name in MPLAB Harmony.

Description

API Name in MLA	API Name in MPLAB Harmony
N/A	SYS_FS_Mount
N/A	SYS_FS_Unmount
N/A	SYS_FS_FileError
FSInit	SYS_FS_Initialize
N/A	SYS_FS_Tasks
Fsfclose	SYS_FS_FileClose
Fsfeof	SYS_FS_FileEOF
FSfread	SYS_FS_FileRead
FSftell	SYS_FS_FileTell
FSfwrite	SYS_FS_FileWrite
N/A	SYS_FS_FileSize
wFSfopen	SYS_FS_FileOpen
N/A	SYS_FS_FileStat
FSfseek	SYS_FS_FileSeek

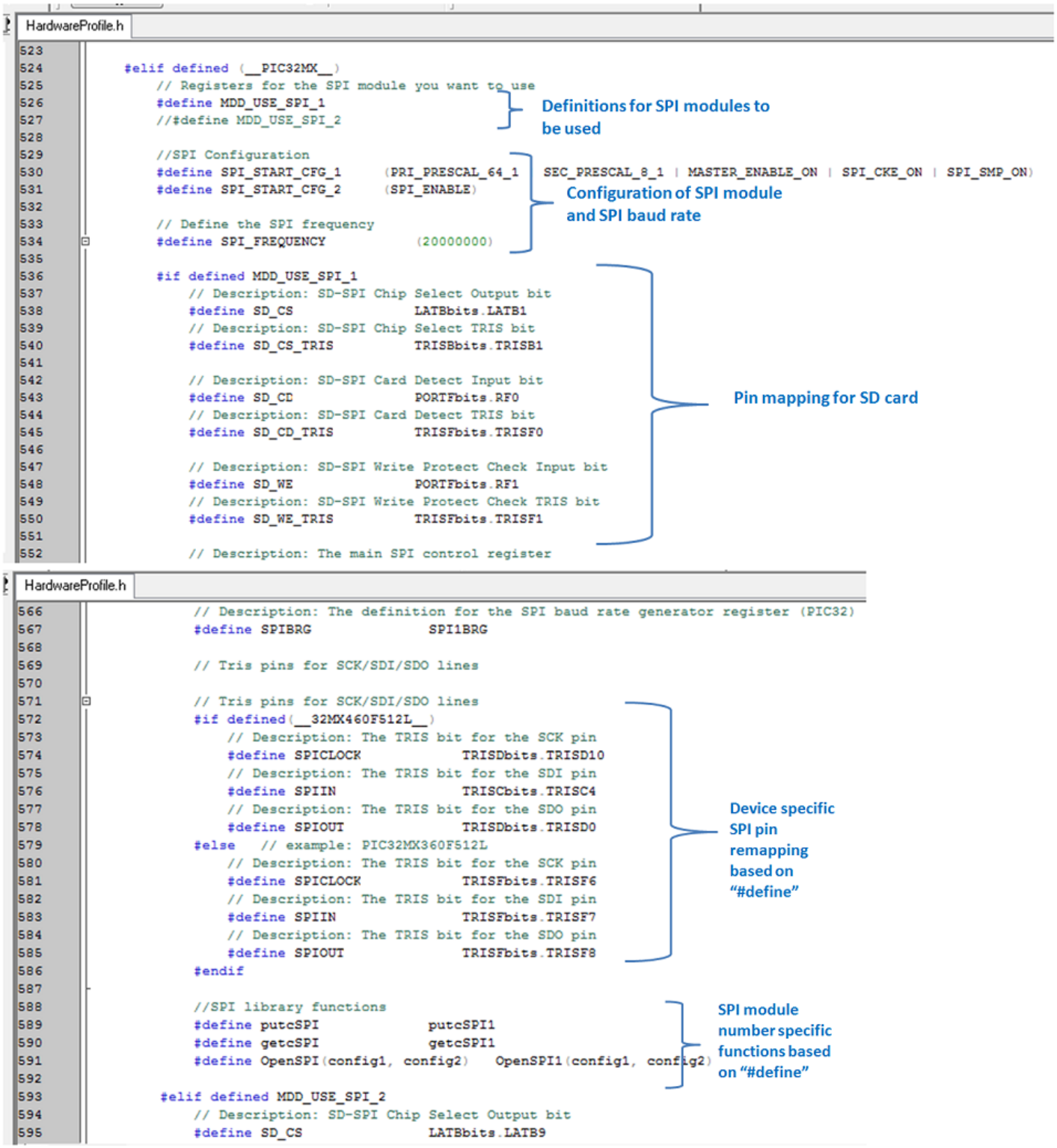
Initialization

This topic describes the differences in initialization.

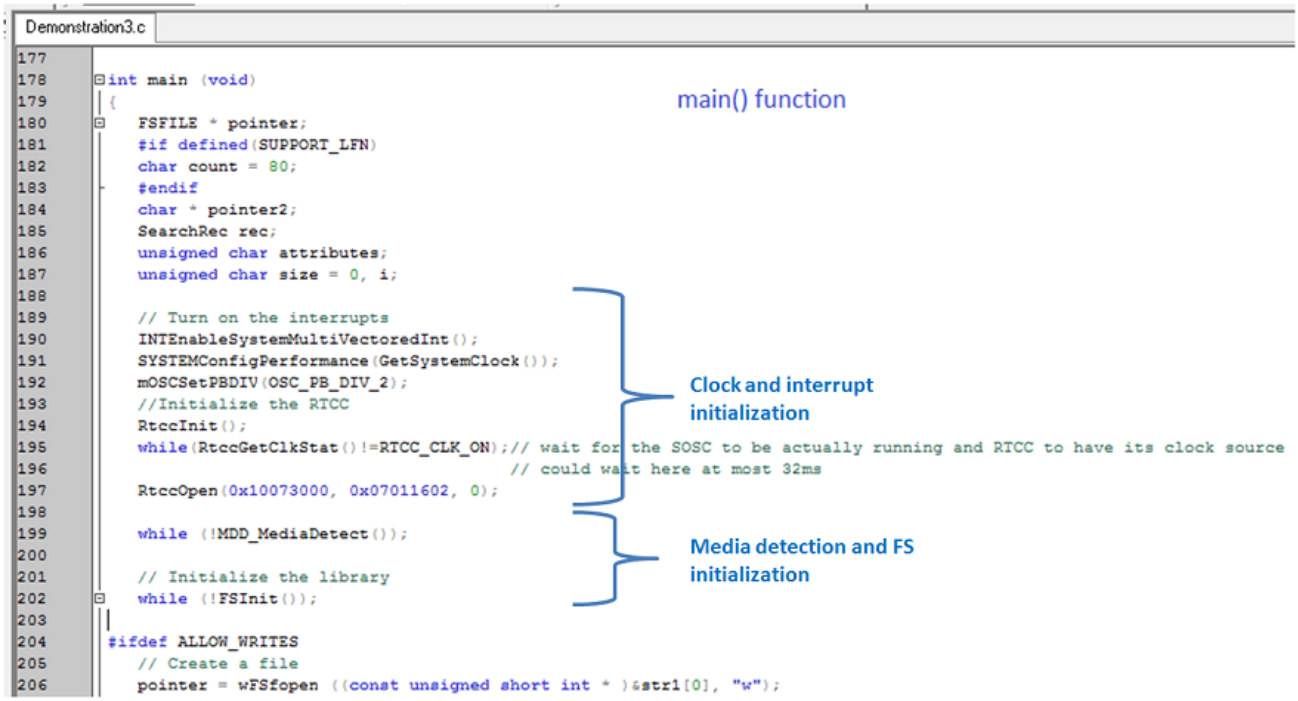
Description

MLA Initialization (Legacy code):

In legacy MLA code, the initialization involves modification of the header file `HardwareProfile.h`. The header file has definitions for the SPI modules used (SPI1 or SPI2), configuration for SPI module, clock frequency, pin mapping for SD card, and pin remapping for SPI (SDI, SDO and SCLK pins). The following two images illustrate these modifications.



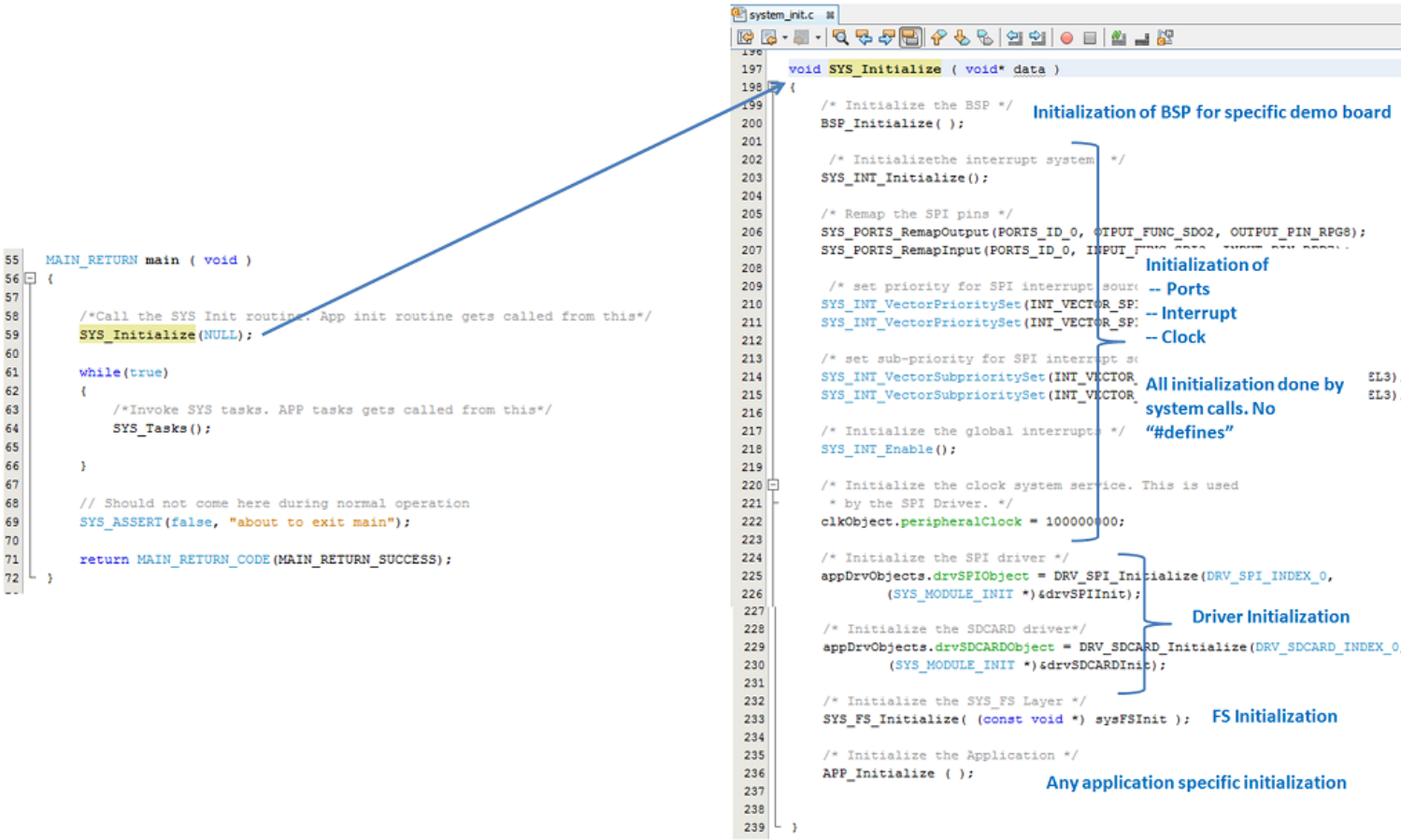
Inside the main function, the initialization of interrupts and the clock occurs in a series of function calls. Then, the media detect function is called for the media used. Later, the FS init function is called, which internally performs the disk mount, and if the mount process and FS initialization is successful, the functions returns as '1'.



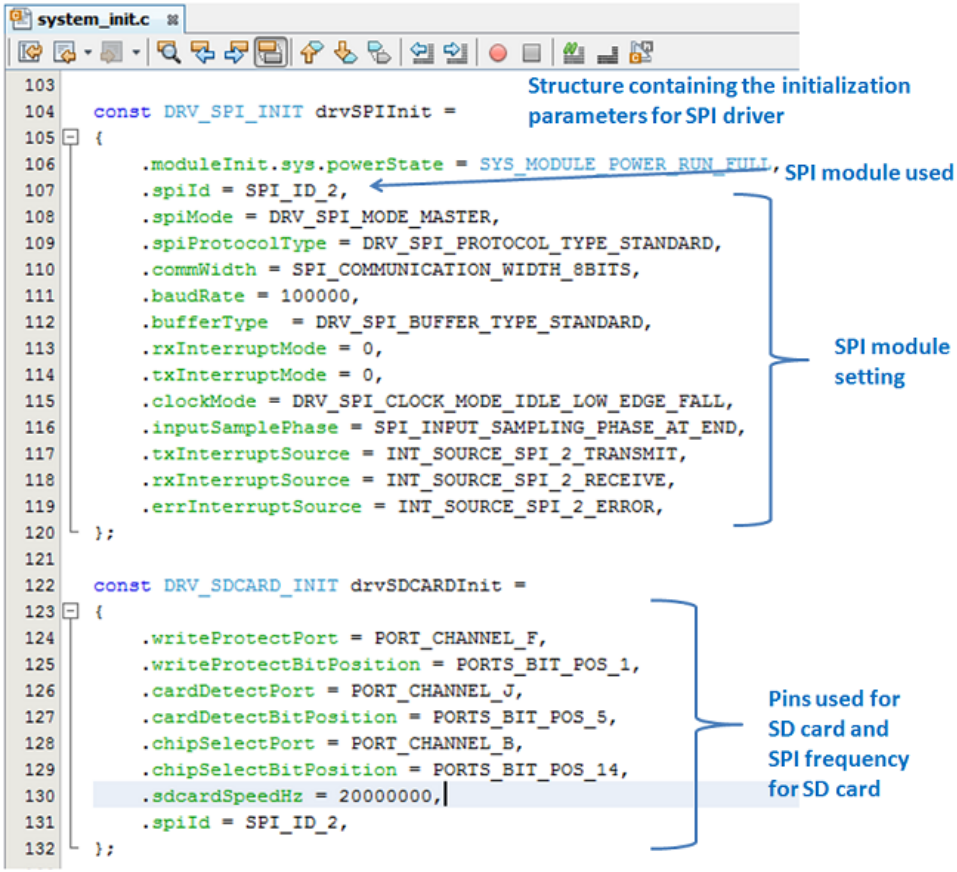
MPLAB Harmony Initialization:

In the case of MPLAB Harmony, the initialization is done by the SYS_Initialize function, which is called from the main function.

Inside the SYS_Initialize function, the BSP, interrupt, clock, port remapping, driver initialization, FS initialization and application-specific initialization is done. All initialization is done by calling specific system function calls, as shown in the following image. For more information, refer to the demonstration code provided as a part of the MPLAB Harmony release package.

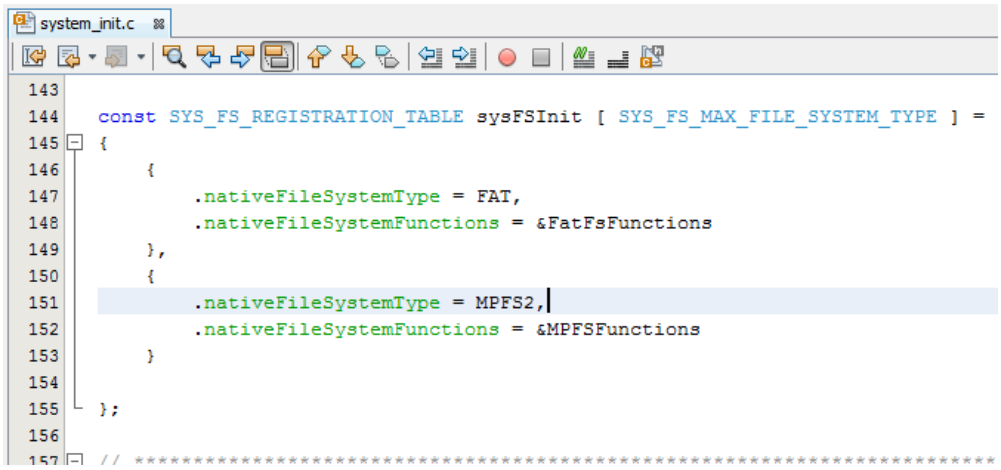


The SPI module selection and configuration of the SPI module is done through the SPI driver initialization structure. Similarly, the selection of pins related to SD card functionality and the selection of the SPI clock frequency is done through the SD card driver initialization structure. The initialization structures are passed as input parameters during driver initialization function calls. There are no `#define` used, as was done in the case of legacy MLA code.




```
const DRV_SPI_INIT drvSPIInit =
{
    .moduleInit.sys.powerState = SYS_MODULE_POWER_RUN_FULL,
    .spiId = SPI_ID_2,
    .spiMode = DRV_SPI_MODE_MASTER,
    .spiProtocolType = DRV_SPI_PROTOCOL_TYPE_STANDARD,
    .commWidth = SPI_COMMUNICATION_WIDTH_8BITS,
    .baudRate = 100000,
    .bufferType  = DRV_SPI_BUFFER_TYPE_STANDARD,
    .rxInterruptMode = 0,
    .txInterruptMode = 0,
    .clockMode = DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL,
    .inputSamplePhase = SPI_INPUT_SAMPLING_PHASE_AT_END,
    .txInterruptSource = INT_SOURCE_SPI_2_TRANSMIT,
    .rxInterruptSource = INT_SOURCE_SPI_2_RECEIVE,
    .errInterruptSource = INT_SOURCE_SPI_2_ERROR,
};

const DRV_SDCARD_INIT drvSDCARDInit =
{
    .writeProtectPort = PORT_CHANNEL_F,
    .writeProtectBitPosition = PORTS_BIT_POS_1,
    .cardDetectPort = PORT_CHANNEL_J,
    .cardDetectBitPosition = PORTS_BIT_POS_5,
    .chipSelectPort = PORT_CHANNEL_B,
    .chipSelectBitPosition = PORTS_BIT_POS_14,
    .sdcardSpeedHz = 20000000,
    .spiId = SPI_ID_2,
};
```

Since MPLAB Harmony supports multiple file systems, there is a structure which needs to be defined and then passed as an input parameter to the SYS_FS_Init function.




```
const SYS_FS_REGISTRATION_TABLE sysFSInit [ SYS_FS_MAX_FILE_SYSTEM_TYPE ] =
{
    {
        .nativeFileSystemType = FAT,
        .nativeFileSystemFunctions = &FatFsFunctions
    },
    {
        .nativeFileSystemType = MPFS2,
        .nativeFileSystemFunctions = &MPFSFunctions
    }

};

// ******************************************************************************
```

System Configuration

This topic describes differences in system configuration.

Description

system_config.h file in MPLAB Harmony:

The file `system_config.h` contains the various system configurations required to run an application. Examples of system configurations related to, but not limited to drivers are:

- SPI driver configuration: such as DRV_SPI_INSTANCES_NUMBER, DRV_SPI_CLIENTS_NUMBER, DRV_SPI_INTERRUPT_MODE, DRV_SPI_BUFFER_SIZE, etc.
- SD Card driver configuration: such as DRV_SDCARD_INSTANCES_NUMBER, DRV_SDCARD_CLIENTS_NUMBER, DRV_SDCARD_QUEUE_POOL_SIZE, etc.

These configurations are required to configure the drivers used for the FS. Since the driver concept was not present in the legacy MLA code, this is a new addition to MPLAB Harmony. To get detailed information about each of the configuration parameter, please refer to the specific driver

documentation.

Media manager configuration:

- SYS_FS_MEDIA_NUMBER - Number of media that will be used in the application. For example, if the application uses SD card and Mass storage device the SYS_FS_MEDIA_NUMBER should be defined as 2.
- SYS_FS_VOLUME_NUMBER - Number of volumes that will be used in the application. MPLAB Harmony supports multi-partitioned media. If the application uses a SD card that has three partitions and a Mass storage device, which has one partition, the SYS_FS_VOLUME_NUMBER should be defined as 4.
- Clock related configurations are present in the file, which are used to set the clock for the device

File system related configuration:

- SYS_FS_MAX_FILE_SYSTEM_TYPE - Number of native file system that will be used in the application. MPLAB Harmony supports multiple native file systems. If the application uses two file systems, FAT FS and MPFS2, SYS_FS_MAX_FILE_SYSTEM_TYPE should be defined as 2.
- SYS_FS_MAX_FILES - Maximum number of files that will be opened by the application at a time
- Other application-specific configuration settings

SYS_Tasks in MPLAB Harmony:

In the case of MPLAB Harmony, certain tasks need to be executed periodically. These tasks are executed from a common function called SYS_Tasks, which is in turn called from the main loop (while(1)) loop. While the concept of running certain functions from the main loop is not new, the legacy MLA demonstration code did not have any functions that had to run from the main loop. However, in case of MPLAB Harmony, it is mandatory that certain tasks should be running periodically from the main loop.

In the case of the FS application, tasks that have to run from SYS_Tasks function are:

- SYS_FS_Tasks - This task maintains the working of the SYS_FS layer and other file system related layers. It is extremely essential that this function runs periodically from the SYS_Tasks function.
- Driver task - Consider a case where the FS application uses a SD card; there is no hardware module that controls the SD card. In such a case, the task routine for the SD card must run from SYS_Tasks. This is required so that the internal driver mechanism for the SD card continues working.
- Any other application related task

system_tasks.c

```
88      Remarks:
89        When not using the dynamic system "Tasks" service, this routine must be
90        implemented by the application's system configuration (in the application's
91        configuration-specific "sys_tasks.c" file).
92    */
93
94  void SYS_Tasks ( void )
95  {
96      /* Task routine for sys_fs */
97      SYS_FS_Tasks();
98
99      /* Call the SDCARD Task */
100     DRV_SDCARD_Tasks(appDrvObjects.drvSDCARDObject);
101
102     /* Call the application's tasks routine */
103     APP_Tasks ( );
104
105 }
```

Mounting a Volume

This topic describes differences when mounting a volume.

Description

Mounting a Volume in MLA versus MPLAB Harmony:

In MLA, the complete mounting of a volume was achieved by calling the two functions, MDD_MediaDetect and FSInit, until the function returns success. The features of mounting a volume in MLA were:

- There are two functions that completely achieve the mounting process
- Both functions were of input type as "void" (no input argument)
- Both functions were called in blocking mode

```
193     //Initialize the RTCC
194     RtccInit();
195     while(RtccGetClkStat()!=RTCC_CLK_ON);// wait for the SOSC to be actually runni
196                                          // could wait here at most 32ms
197     RtccOpen(0x10073000, 0x07011602, 0);
198
199     while (!MDD_MediaDetect());
200
201     // Initialize the library
202     while (!FSInit());
203
204 #ifdef ALLOW_WRITES
205     // Create a file
206     pointer = wFSfopen ((const unsigned short int * )&str1[0], "w");
207     if (pointer == NULL)
208         while(1);
```

In MPLAB Harmony, the mounting of a volume is achieved by calling the SYS_FS_Mount function until the function returns success. The features of mounting a volume in MPLAB Harmony are:

- SYS_FS_Mount function includes both media detection and mounting the volume. Please note that, MPLAB Harmony FS still needs the function SYS_FS_Initialize to be called during system initialization. Though the name SYS_FS_Initialize seems similar to the MLA function FSInit, the task achieved by calling these functions are very different.
- SYS_FS_Initialize (MPLAB Harmony function) - Just does the initialization of SYS_FS layer. It does not do any mounting of volumes.
- FSInit (MLA function) - Did the initialization of FS and also did mount the disk.
- SYS_FS_Mount function accepts input parameter such as devName, mntName, fileSystemType, etc. To know each of the parameter in detail, pleas refer to the documentation of the SYS_FS_Mount function.
- SYS_FS_Mount has to be running in a non-blocking mode. The following image shows an application where SYS_FS_Mount is called from the state machine implementation. This is required as the implementation of the state machine allows the SYS_Tasks function to be run periodically. This is quite different from the earlier implementations of MLA.

```
140
141     switch(appData.state)
142     {
143         case APP_MOUNT_DISK:
144             if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive", FAT, 0, NULL) != 0)
145             {
146                 /* The disk could not be mounted. Try
147                  * mounting again untill success. */
148
149                 appData.state = APP_MOUNT_DISK;
150             }
151             else
152             {
153                 /* Mount was successful. Unmount the disk, for testing. */
154
155                 appData.state = APP_UNMOUNT_DISK;
156             }
157             break;
```

Opening a File

This topic describes differences when opening a file.

Description

Opening a file in MLA versus MPLAB Harmony:

Opening a file remains similar for both MLA and MPLAB Harmony. However, a critical difference in MPLAB Harmony is that, while opening the file, the complete path of the file must be specified. In addition, the path must be preceded by the string `/mnt/`. The complete path is required because the MPLAB Harmony FS implements a multi-partition media support. Therefore, the file path should include the name of volume (assigned media partition), and from where the file is to be opened. For more details, please refer to the documentation of the SYS_FS_FileOpen function. The following two figures illustrate opening a file in MLA as opposed to MPLAB Harmony.

In MLA, the error in opening a file was indicated by the file open function returning a NULL, while, in MPLAB harmony, the file open returns SYS_FS_HANDLE_INVALID.

```
244 #endif
245
246     // Open file 1 in read mode
247     pointer = wFSfopen ((const unsigned short int * )&str1[0], "r");
248     if (pointer == NULL)
249         while(1);
250
251     if (wFSrename ((const unsigned short int * )&str3[0], pointer))
252         while(1);
253
254     // Read one four-byte object
255     if (FSfread (receiveBuffer, 4, 1, pointer) != 1)
256         while(1);
```

Figure 1: Opening a File in MLA

```
173
174         case APP_OPEN_FILE:
175
176             appData.fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE_TOO_LONG_NAME_EXAMPLE_123.JPG",
177                     (SYS_FS_FILE_OPEN_READ));
178             if(appData.fileHandle == SYS_FS_HANDLE_INVALID)
179             {
180                 /* Could not open the file. Error out*/
181                 appData.state = APP_ERROR;
182             }
183             else
184             {
```

Figure 2: Opening a File in MPLAB Harmony

Reading a File

This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS, while reading a file.

Description

Reading a file in MLA versus MPLAB Harmony:

Reading a file remains similar for both MLA and MPLAB Harmony, as depicted in the following figures. The smaller differences are that, the name of the function to read the file is different, the order of parameters passed are different, and MPLAB Harmony only takes three parameters. The MPLAB Harmony function only enables byte based access.

Although both the MLA and MPLAB Harmony file read functions return the number of bytes read, the MPLAB Harmony functions also return '-1' if there were any errors while reading the file.

Demonstration3.c

```
253
254     // Read one four-byte object
255     if (FSfread (receiveBuffer, 4, 1, pointer) != 1)
256        while(1);
257
258     // Check if this is the end of the file- it shouldn't be
259     if (FSfeof (pointer))
```

Figure 1: Reading a File in MLA

Start Page | app.c | main.c

```
201
202             if(SYS_FS_FileRead((void *)appData.data, 512, appData.fileHandle) == -1)
203             {
204                 /* There was an error while reading the file.
205                  * Close the file and error out. */
206
207                 SYS_FS_FileClose(appData.fileHandle);
208                 appData.state = APP_ERROR;
209             }
```

Figure 2: Reading a File in MPLAB Harmony

Writing a File

This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS, while writing to a file.

Description

Writing to a file in MLA versus MPLAB Harmony:

Writing to a file remains similar for both MLA and MPLAB Harmony, as depicted in the following figures. The smaller differences are that the name of the function to write to the file is different, the order of parameters passed are different, and MPLAB Harmony only takes three parameters. The MPLAB Harmony function only enables byte based access.

Although both the MLA and MPLAB Harmony file read functions return the number of bytes written, the MPLAB Harmony functions also return '-1' if there were any errors while writing to the file.

```
Demonstration3.c  FSIO.h
208         while(1);
209
210     // Write 21 1-byte objects from sendBuffer into the file
211     if (FSfwrite (sendBuffer, 1, 21, pointer) != 21)
212         while(1);
213
```

Figure 1: Writing to a File in MLA

```
Start Page  app.c  main.c
212             /* If read was success, try writing to the new file */
213             if(SYS_FS_FileWrite((const void *)appData.data, 512, appData.fileHandle1) == -1)
214             {
215                 /* Write was not successful. Close the file
```

Figure 2: Writing to a File in MPLAB Harmony

Closing a File

This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS, while closing a file.

Description

Closing a file in MLA versus MPLAB Harmony:

Closing a file remains similar for both MLA and MPLAB Harmony, as depicted in the following figures. The smaller difference is that the name of the function is different.

```
Demonstration3.c  FSIO.h
227
228     // Close the file
229     if (FSfclose (pointer))
230         while(1);
231
```

Figure 1: Closing a File in MLA

```
Start Page  app.c  main.c
229             /* Close both files */
230             SYS_FS_FileClose(appData.fileHandle);
231             SYS_FS_FileClose(appData.fileHandle1);
232              /* The test was successful. Lets idle. */
```

Figure 2: Closing a File in MPLAB Harmony

File EOF

This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS, while checking for end of file.

Description

Checking for EOF in MLA versus MPLAB Harmony:

The EOF function remains similar for both MLA and MPLAB Harmony, as depicted in the following figures. The smaller difference is that the name of the function is different.

```
Demonstration3.c  FSIO.h
257
258     // Check if this is the end of the file- it shouldn't be
259     if (FSfeof (pointer))
260         while(1);
```

Figure 1: EOF in MLA

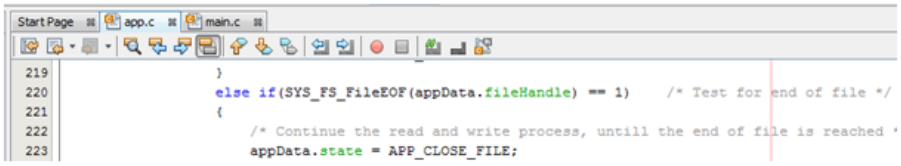


Figure 2: EOF in MPLAB Harmony

File Tell

This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS, while checking for file pointer (tell).

Description

Checking for file pointer (tell) in MLA versus MPLAB Harmony:

The function to check for file pointer (tell) remains similar for both MLA and MPLAB Harmony, as depicted in the following figures. The smaller difference is that the name of the function is different.

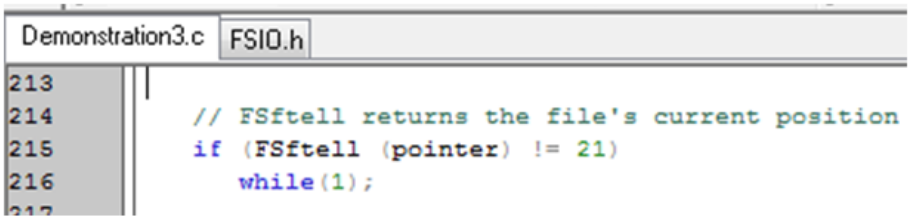


Figure 1: Tell Function in MLA

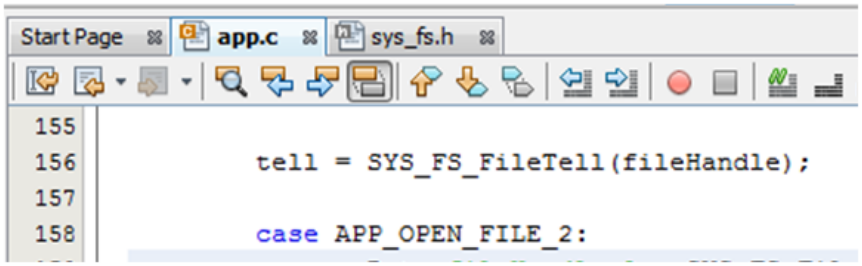


Figure 2: Tell Function in MLA Harmony

File Seek

This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS, while performing a file seek.

Description

File seek in MLA versus MPLAB Harmony:

The function to perform file seek remains similar for both MLA and MPLAB Harmony, as depicted in the following figures. The smaller difference is that the name of the function is different.

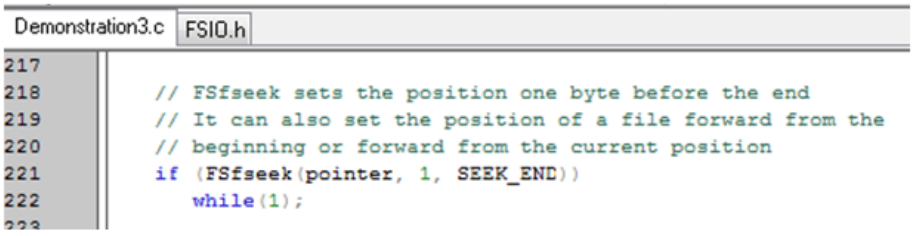


Figure 1: File Seek in MLA

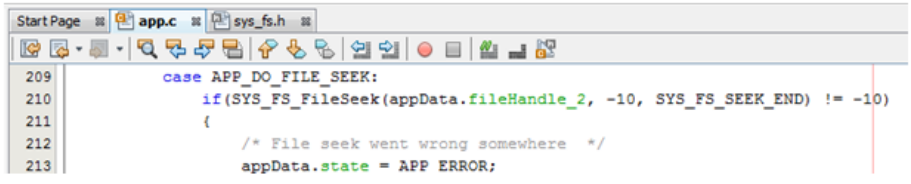


Figure 2: File Seek in MPLAB Harmony

SYS_FS_Tasks

This topic describes the differences between legacy MLA code for FS and MPLAB Harmony FS with respect to the SYS_FS_Tasks function.

Description

SYS_FS_Tasks in MLA Versus MPLAB Harmony:

MPLAB Harmony needs the SYS_FS_Tasks function to be running periodically from the SYS_Tasks function. The MLA code did not have any such function that had to be running periodically.

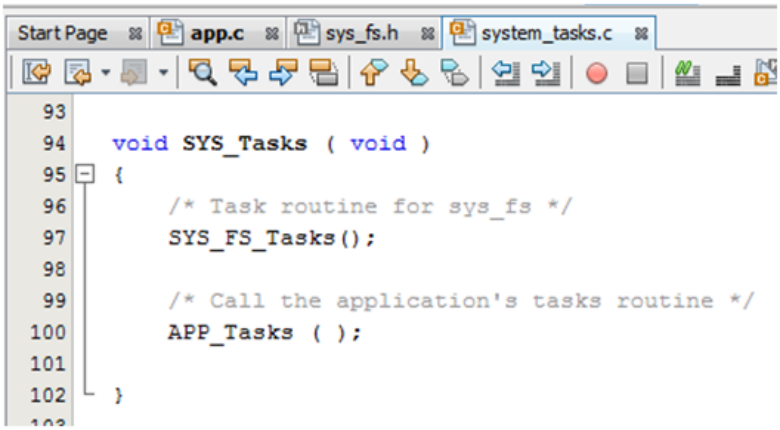


Figure 1: Running SYS_FS_Tasks Function from SYS_Tasks

Using the Library

This topic describes the basic architecture of the File System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_fs.h`

The interface to the File System Service library is defined in the `sys_fs.h` header file. This file is included by the system.h file. Any C language source (`.c`) file that uses the File System Service Library should include `system.h`.

Please refer to the What is MPLAB Harmony? section for how the File System Service interacts with the framework.

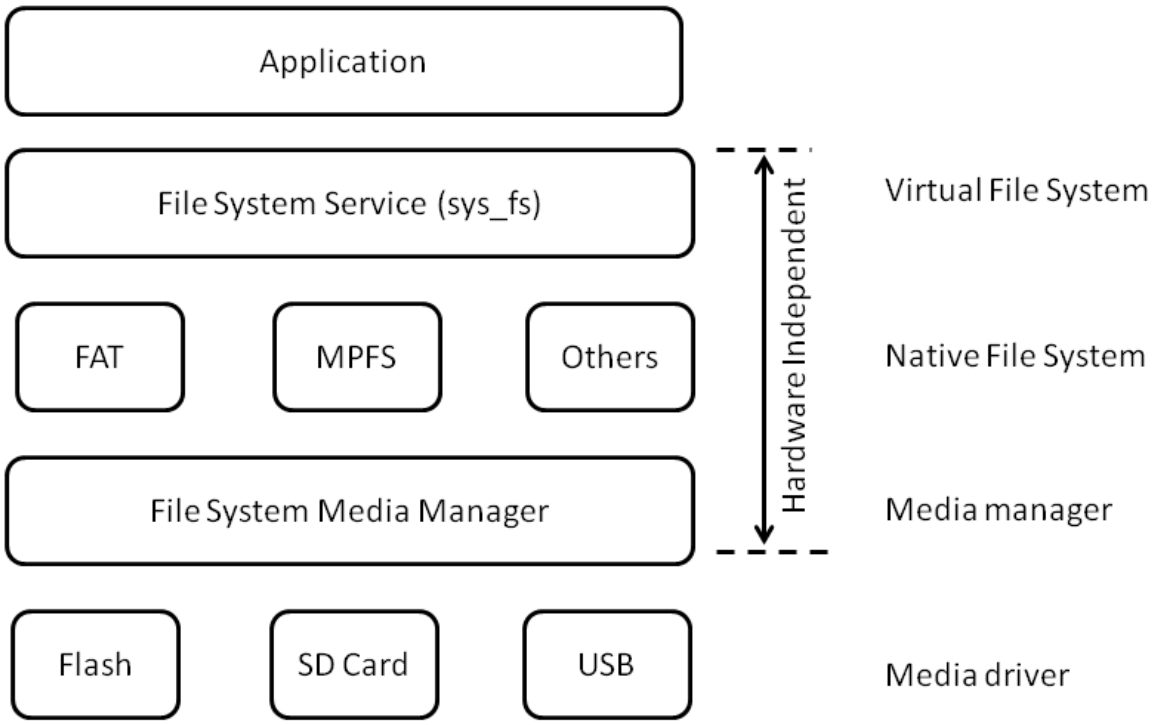
Abstraction Model

This topic describes the abstraction model of the MPLAB Harmony File System.

Description

The FS framework features a modular and layered architecture, as shown in the following figure.

FS Framework Architecture



As seen in the figure, the FS Framework consists of the following major blocks:

- The **Driver** for the physical media has to be included as a part of the FS Framework. This layer provides a low-level interface to access the physical media. This layer also enables multiple instances of media. Examples of drivers are:
- NVM driver – To access files using NVM (Flash memory)
- SPI driver – To access files from SD card, which interfaces using the SPI peripheral
- The **Media driver** provides a mechanism to access the media as "sectors". Sectors are the smallest storage element accessed by a file system and are contiguous memory locations. Typically, each sector has 512 bytes. Depending on the requirement, in some cases, the driver and media driver could be combined as one layer.
- The **Media manager** implements a disk and sector based media access mechanism. It also performs disk allocated/deallocated on media attach/detach. Due to the implementation of this layer, the FS Framework can support multiple disks. The media manager detects and analyzes a media based on its Master Boot Record (MBR). Therefore, it is mandatory for the media to have a MBR for it to work with the FS.
- The **Native file system** implements support for the media file system format. Examples of native file systems are: FAT12, FAT32, and MPFS, among other. At present, only the FAT and MPFS files systems are supported by the FS framework; however, more native file systems can be included.
- The **Virtual file system (or SYS_FS)** layer provides a file system independent file system operation interface. This layer translates virtual file systems calls to native file system calls. Due to this layer, applications can now support multiple file systems. Interfaces provided by this layer, but not limited to, include:
- SYS_FS_mount
- SYS_FS_open
- SYS_FS_read
- SYS_FS_write
- SYS_FS_close

Library Overview

Please refer to the System Service Introduction for a detailed description of MPLAB Harmony system services.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the File System module.

Library Interface Section	Description
File and Directory Operation Functions	Provides file and directory operation interface routines.
General Operation Functions	Provides general operation interface routines.

How the Library Works

This topic provides information on how the MPLAB Harmony File System works.

Description

The MPLAB Harmony File System (FS) provides embedded application developers with a file system framework for retrieving and storing data from various media.

The MPLAB Harmony file system is designed to support multiple file systems (native file systems) and multiple media at the same time. Examples of native file systems are FAT12, FAT32, MPFS, and JFS, among others. Each of these native file systems has a common set of APIs that can be used to access the files of that particular native file system. The FS is a part of the MPLAB Harmony installation and is accompanied by demonstration applications that highlight usage. These demonstrations can also be modified or updated to build custom applications.

FS features include the following:

- Support for multiple file system (FAT, MPFS)
- Supports multiple physical media (NVM, SD card)
- More physical media can be interfaced with the FS, once the driver is available for the media
- Modular and Layered architecture

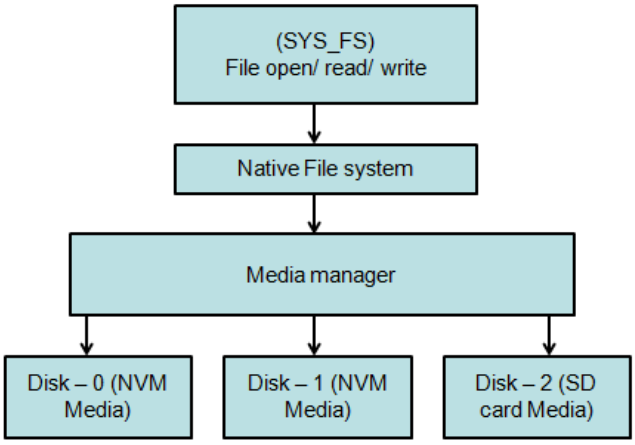
Application Interaction

This topic describes how an application must interact with the File System.

Description

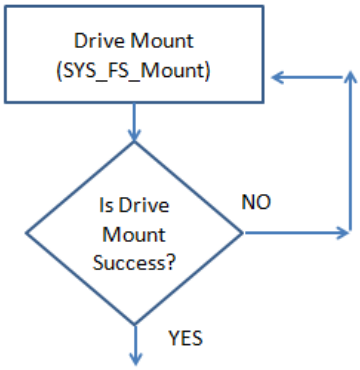
The interaction of various layers is shown in the following figure.

Application Interaction with FS Framework



In the process of using the FS Framework, the application must first **mount** the media drive for the FS Framework to access the media. Unless the mounting process returns successfully, the application should continue trying to mount the drive. If the drive is not attached, the mounting process will fail. In such a situation, the application should not proceed further unless the mounting is success.

Application Mounts a Drive

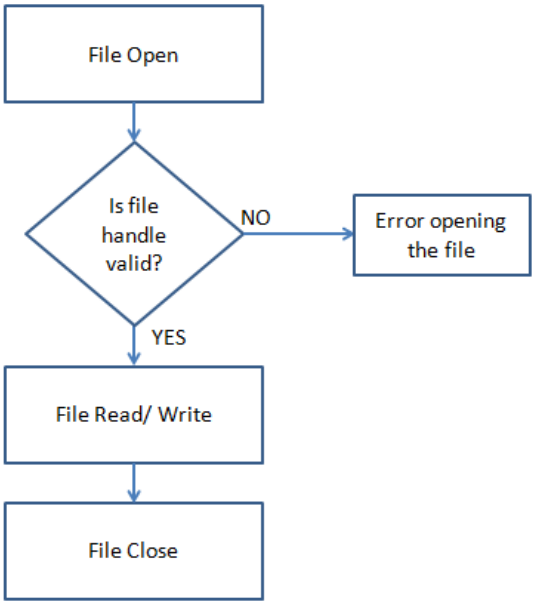


Once the drive is mounted, the application code can then **open the file** from the drive with different attributes (such as read-only or write). If the file open returns a valid handle, the application can proceed further. Otherwise, the application will enter an error state. The reason for an invalid

handle could be that the application was trying to read a file from the drive that does not exist. Another reason for an invalid handle is when the application tries to write to a drive that is write-protected.

Once the file is opened, the valid file handle is further used to **read/write** data to the file. Once the required operation is performed on the file, the file can then be **closed** by the application by passing the file handle. The following figure illustrates the process.

Further File System Operations



Using the File System

This topic describes how to use the File System.

Description

Use the Available Library Demonstration Applications

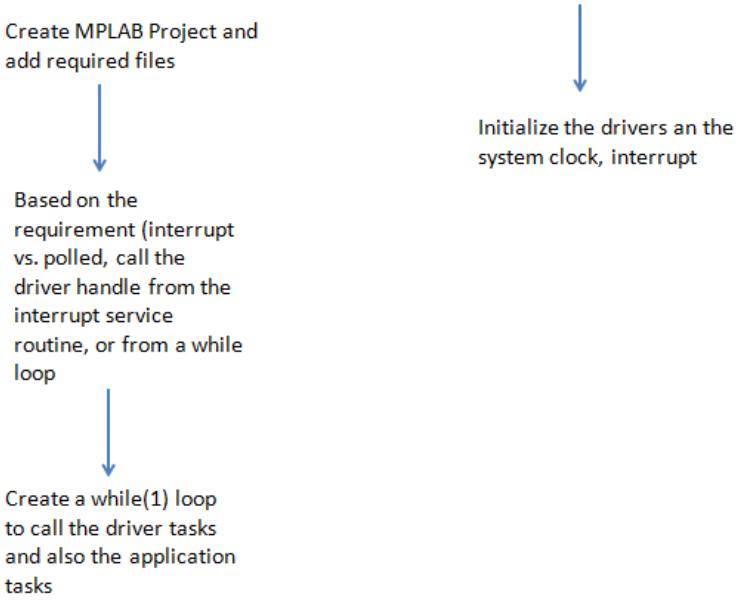
The FS framework release package contains a set of demonstration applications that are representative of common scenario (single/multi-media and single/multi-native file systems). These demonstrations can be easily modified to include application-specific initialization and application logic. The application logic must be non-blocking and could be implemented as a state machine.

- The application specific initialization can be called in the SYS_Initialize function (in the `system_init.c` file). The SYS_Initialize function is called when the device comes out of Power-on Reset (POR).
- The application logic can be called in the SYS_Tasks function (in the `system_tasks.c` file). The application logic can interact with the FS layer by using relevant API calls, as provided in the APP_Tasks (in the `app.c` file)

Building a FS Application from Scratch

In a case where the available demonstration applications do not meet the end application requirements, an application to use the FS framework can be created from scratch. The following figure shows a flowchart for the steps that need to be performed.

Steps to Create a FS Application



Step 1:

Create a MPLAB Project and add the required FS framework files to the project. The following files are needed to build the a FS project

- `system_config.h` - This file should contain the compile time configuration macros for the Driver, Media Layer, and FS layer. The file also contains the clock speed setting, which is set for the microcontroller.
- `system_init.c` – This file should contain the initial settings for each driver. It should also call the functions required to initialize different drivers to be used by the FS.
- `ff.c`, `diskio.c`, `mpfs.c`, `sys_fs.c`, `sys_fs_media_manager.c` – These files are part of the FS, which must be included in the project
- `sys_int_pic32.c` and `plib_int_pic32.c` - These files implement the system interrupt service that is required by the FS
- `sys_ports.c` – If the FS is using a SD card as media, this file needs to be included (for Chip Select)
- Driver – The driver for media to be used by FS should also be included
- Application specific files - These file will implement the application logic

Step 2:

Since the MPLAB Harmony drivers included with the File System operate in interrupt mode, a driver Handler should be defined as follows:

```
/* Use this for PIC32MX */
void __ISR ( _SPI1_VECTOR,ipl4 ) _InterruptHandler_SPI_stub ( void )
{
    DRV_SPI_Tasks((SYS_MODULE_OBJ)appDrvObjects.drvSPIObject);
}

/* Use this for PIC32MZ */
void __ISR ( _SPI2_RX_VECTOR,ipl4 ) _InterruptHandler_SPI_RX_stub ( void )
{
    DRV_SPI_Tasks((SYS_MODULE_OBJ)appDrvObjects.drvSPIObject);
}

void __ISR ( _SPI2_TX_VECTOR,ipl4 ) _InterruptHandler_SPI_TX_stub ( void )
{
    DRV_SPI_Tasks((SYS_MODULE_OBJ)appDrvObjects.drvSPIObject);
}
```

Step 3:

The application should create a `while(1)` loop that continuously updates the driver layer State Machine and the application state machine. This requires the application state machine to be non-blocking.

```
/* This while(1) loop will continuously update the driver layer state machine and the application state
machine */

while(1)
{
    /* Task routine for sys_fs */
    SYS_FS_Tasks();

    /* Call the SDCARD Task */
    DRV_SDCARD_Tasks(appDrvObjects.drvSDCARDObject);
```

```
    /* Call the application's tasks routine */
    APP_Tasks ( );

}
```

Step 4:

If interrupt-based operation is needed, the interrupts need to be enabled first. The application should then initialize the driver layer. Refer to the driver specific documentation regarding usage.

```
/* Initialize the interrupt system  */

SYS_INT_Initialize();

/* Initialize the global interrupts */
SYS_INT_Enable();

/* set priority for SPI interrupt source */
SYS_INT_VectorPrioritySet(INT_VECTOR_SPI1, INT_PRIORITY_LEVEL3);

/* set sub-priority for SPI interrupt source */
SYS_INT_VectorSubprioritySet(INT_VECTOR_SPI1, INT_SUBPRIORITY_LEVEL3);

/* Initialize the global interrupts */
SYS_INT_Enable();

/* Initialize the SPI driver */
appDrvObjects.drvSPIObject = DRV_SPI_Initialize(DRV_SPI_INDEX_0,
        (SYS_MODULE_INIT *)&drvSPIInit);

/* Initialize the SDCARD driver*/
appDrvObjects.drvSDCARDObject = DRV_SDCARD_Initialize(DRV_SDCARD_INDEX_0,
        (SYS_MODULE_INIT *)&drvSDCARDInit);
```

Step 5:

The application code can be implemented as a non-blocking state machine inside the APP_TASKS function, as shown in the following example for an application to read a file and write the content into another newly created file.

The input file `TEST.JPG` is not provided with the release package. It could be any arbitrary JPEG file chosen by the user, and then suitably renamed to `TEST.JPG`. The reason for choosing a JPEG file for test purposes is that the duplicate file `TEST1.JPG` created by the FS demonstration could be easily verified for correctness by inserting the SD card in the computer and opening the `TEST1.JPG` file.

```
void APP_Tasks ( void )
{
    /* The application task state machine */

    switch(appData.state)
    {
        case APP_MOUNT_DISK:
            if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive", FAT, 0, NULL) != 0)
            {
                /* The disk could not be mounted. Try
                 * mounting again until success. */

                appData.state = APP_MOUNT_DISK;
            }
            else
            {
                /* Mount was successful. Open a file.
                 * Let the switch case fall through. */
                appData.state = APP_OPEN_FILE;
            }
            break;

        case APP_OPEN_FILE:

            appData.fileHandle = SYS_FS_FileOpen("/mnt/myDrive/TEST.JPG",
                    (FA_READ));
            if(appData.fileHandle == SYS_FS_HANDLE_INVALID)
            {
                /* Could not open the file. Error out*/
                appData.state = APP_ERROR;
            }
```

```
            else
            {
                appData.fileHandle1 = SYS_FS_FileOpen("/mnt/myDrive/TEST1.JPG",
                        (FA_WRITE|FA_CREATE_ALWAYS));
                if(appData.fileHandle == SYS_FS_HANDLE_INVALID)
                {
                    /* Could not open the file. Error out*/
                    appData.state = APP_ERROR;
                }
                else
                {
                    /* Check the file to be read */
                    appData.state = APP_CHECK_FILE;
                }
            }
            break;

        case APP_CHECK_FILE:
            /* check the size of file */
            fileSize = SYS_FS_FileSize(appData.fileHandle);
            /* since, we will read 512 bytes at a time, find the number of times, the read has to be
             performed */
            sectorCounter = integralSector = (fileSize/512);
            /* find the remaining bytes */
            balanceSector = (fileSize%512);

            appData.state = APP_READ_WRITE_TO_FILE;

            break;

        case APP_READ_WRITE_TO_FILE:

            if(SYS_FS_FileRead((void *)appData.data, 512, appData.fileHandle) == -1)
            {
                /* There was an error while reading the file.
                 * Close the file and error out. */

                SYS_FS_FileClose(appData.fileHandle);
                appData.state = APP_ERROR;
            }
            else if(SYS_FS_FileWrite((const void *)appData.data, 512, appData.fileHandle1) == -1)
            {
                /* Write was not successful. Close the file
                 * and error out.*/
                SYS_FS_FileClose(appData.fileHandle1);
                appData.state = APP_ERROR;
            }
            else
            {
                sectorCounter--;
                /* if entire integral sectors are written, write the balance sector*/
                if(sectorCounter == 0)
                {
                    if(SYS_FS_FileRead((void *)appData.data, balanceSector, appData.fileHandle) == -1)
                    {
                        /* There was an error while reading the file.
                         * Close the file and error out. */

                        SYS_FS_FileClose(appData.fileHandle);
                        appData.state = APP_ERROR;
                    }
                    else if(SYS_FS_FileWrite((const void *)appData.data, balanceSector,
appData.fileHandle1) == -1)
                    {
                        /* Write was not successful. Close the file
                         * and error out.*/
                        SYS_FS_FileClose(appData.fileHandle1);
                        appData.state = APP_ERROR;
                    }
```

```
                    else
                    {
                        appData.state = APP_CLOSE_FILE;
                    }
                }
            }
            break;

        case APP_CLOSE_FILE:
            /* Close both files */
            SYS_FS_FileClose(appData.fileHandle);
            SYS_FS_FileClose(appData.fileHandle1);
             /* The test was successful. Lets idle. */
            appData.state = APP_IDLE;
            break;

        case APP_IDLE:
            /* The application comes here when the demonstration
               has completed successfully. Switch on green LED. */
            BSP_SwitchONLED(LED_2);
            break;
        case APP_ERROR:
            /* The application comes here when the demonstration
               has failed. Switch on the red LED.*/
            BSP_SwitchONLED(LED_1);
            break;
        default:
            break;

    }

} //End of APP_Tasks
```

Configuring the Library

Lists and describes the library configuration macros.

Macros

	Name	Description
	SYS_FS_AUTOMOUNT_ENABLE	Enable/Disable Auto Mount Feature of File system
	SYS_FS_MAX_FILE_SYSTEM_TYPE	Number of file systems used in the application
	SYS_FS_MAX_FILES	Number of simultaneous files access
	SYS_FS_MEDIA_MAX_BLOCK_SIZE	Media Sector Size information
	SYS_FS_MEDIA_NUMBER	Number of media used in the application
	SYS_FS_VOLUME_NUMBER	This macro defines number of volumes used in the application

Description

The configuration of the File System Service is based on the file `system_config.h`.

This header file contains the configuration selection for the File System service. Based on the selections made, the File System Service may support the selected features. These configuration settings will apply to all instances of the File System Service.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

SYS_FS_AUTOMOUNT_ENABLE Macro

Enable/Disable Auto Mount Feature of File system

File

sys_fs_config_template.h

C

```
#define SYS_FS_AUTOMOUNT_ENABLE 1
```

Description

Enable/Disable Auto Mount Feature of File system

Enable/Disable Auto Mount Feature of File system

Remarks

None.

SYS_FS_MAX_FILE_SYSTEM_TYPE Macro

Number of file systems used in the application

File

sys_fs_config_template.h

C

```
#define SYS_FS_MAX_FILE_SYSTEM_TYPE 1
```

Description

Number of File system types

Number of fil systems used in the application

Remarks

None.

SYS_FS_MAX_FILES Macro

Number of simultaneous files access

File

sys_fs_config_template.h

C

```
#define SYS_FS_MAX_FILES 1
```

Description

Number of simultaneous files access

Number of simultaneous files access

Remarks

None.

SYS_FS_MEDIA_MAX_BLOCK_SIZE Macro

Media Sector Size information

File

sys_fs_config_template.h

C

```
#define SYS_FS_MEDIA_MAX_BLOCK_SIZE 512
```

Description

Media Sector Size information

Media Sector Size information

Remarks

None.

SYS_FS_MEDIA_NUMBER Macro

Number of media used in the application

File

sys_fs_config_template.h

C

```
#define SYS_FS_MEDIA_NUMBER 1
```

Description

Number of media used in the application

Number of media used in the application

Remarks

None.

SYS_FS_VOLUME_NUMBER Macro

This macro defines number of volumes used in the application

File

sys_fs_config_template.h

C

```
#define SYS_FS_VOLUME_NUMBER 1
```

Description

Number of Volumes

This macro defines the number of volumes used in the application

Remarks

None.

Building the Library

This section lists the files that are available in the File System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/fs`.

Interface File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/sys_fs.h	This file contains function and type declarations required to Interact with the MPLAB Harmony File System Framework.
/fat_fs/src/file_system/ff.h	FAT File System module include file. This file should be included when using the FAT File System.
/mpfs/mpfs.h	This file contains the interface definition for handling the Microchip File System (MPFS). This file should be included when using MPFS.

Required File(s)

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/sys_fs.c	This file contains implementation of File System interfaces.
/src/dynamic/sys_fs_media_manager.c	This file contains implementation File System Media Manager functions.
/fat_fs/src/file_system/ff.c	This file implements the FAT File system functions. This file should be included when using FAT File System.
/fat_fs/src/hardware_access/diskio.c	Low-level disk I/O module for FAT File System. This file should be included when using FAT File System.
/mpfs/src/mpfs.c	This file implements the MPFS functions. This file should be included when using MPFS.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The File System Service Library depends on the following modules:

- NVM Driver Library (if media is NVM)
- Secure Digital (SD) Card Driver Library (if media is a SD Card)
- SPI Driver Library (if media is a SD Card)
- Device Control System Service Library
- Interrupt System Service Library
- Ports System Service Library
- Clock System Service Library

Library Interface

a) File and Directory Operation Functions

	Name	Description
	SYS_FS_FileClose	Closes a file.
	SYS_FS_FileEOF	Checks for end of file.
	SYS_FS_FileOpen	Opens a file.
	SYS_FS_FileSeek	Moves the file pointer by the requested offset.
	SYS_FS_FileSize	Returns the size of the file in bytes.
	SYS_FS_FileTell	Obtains the file pointer position.
	SYS_FS_FileNameGet	Reads the file name.
	SYS_FS_FilePrintf	Writes a formatted string into a file.
	SYS_FS_FileTestError	Checks for errors in the file.
	SYS_FS_FileDirectoryModeSet	Sets the mode for the file or directory.
	SYS_FS_FileDirectoryRemove	Removes a file or directory.
	SYS_FS_FileDirectoryRenameMove	Renames or moves a file or directory.
	SYS_FS_FileDirectoryTimeSet	Sets or changes the time for a file or directory.
	SYS_FS_FileTruncate	Truncates a file
	SYS_FS_FileCharacterPut	Writes a character to a file.
	SYS_FS_FileStringGet	Reads a string from the file into a buffer.
	SYS_FS_FileStringPut	Writes a string to a file.
	SYS_FS_Error	Returns the last error.
	SYS_FS_FileError	Returns the file specific error.
	SYS_FS_FileRead	Read data from the file.
	SYS_FS_FileWrite	Writes data to the file.
	SYS_FS_CurrentWorkingDirectoryGet	Gets the current working directory
	SYS_FS_DirectoryChange	Changes to a the directory specified.
	SYS_FS_DirectoryMake	Makes a directory.
	SYS_FS_DirClose	Closes an opened directory.

	SYS_FS_DirOpen	Open a directory
	SYS_FS_DirRead	Reads the specified directory.
	SYS_FS_DirSearch	Searches for a file or directory.
	SYS_FS_FileSync	Flushes the cached information when writing to a file.
	SYS_FS_DirRewind	Rewinds to the beginning of the directory.
	SYS_FS_FileStat	Gets file status.

b) General Operation Functions

	Name	Description
	SYS_FS_Initialize	Initializes the file system abstraction layer (sys_fs layer).
	SYS_FS_Mount	Mounts the file system.
	SYS_FS_Tasks	Maintains the File System tasks and functionalities.
	SYS_FS_Unmount	Unmounts the file system.
	SYS_FS_CurrentDriveGet	Gets the current drive
	SYS_FS_CurrentDriveSet	Sets the drive.
	SYS_FS_DriveLabelGet	Gets the drive label.
	SYS_FS_DriveLabelSet	Sets the drive label
	SYS_FS_DriveFormat	Formats a drive.
	SYS_FS_DrivePartition	Partitions a physical drive (media).
	SYS_FS_DriveSectorGet	Obtains total number of sectors and number of free sectors for the specified drive.
	SYS_FS_EventHandlerSet	Allows a client to identify an event handling function for the file system to call back when mount/unmount operation has completed.

c) Data Types and Constants

	Name	Description
	SYS_FS_ERROR	Lists the various error cases.
	SYS_FS_FILE_OPEN_ATTRIBUTES	Lists the various attributes (modes) in which a file can be opened.
	SYS_FS_FILE_SEEK_CONTROL	Lists the various modes of file seek.
	SYS_FS_FILE_SYSTEM_TYPE	Enumerated data type identifying native file systems supported.
	SYS_FS_FSTAT	File System status
	SYS_FS_FUNCTIONS	SYS FS Function signature structure for native file systems.
	SYS_FS_HANDLE	This type defines the file handle.
	SYS_FS_REGISTRATION_TABLE	The sys_fs layer has to be initialized by passing this structure with suitably initialized members.
	SYS_FS_RESULT	Lists the various results of a file operation.
	FAT_FS_MAX_LFN	Maximum length of the Long File Name.
	FAT_FS_MAX_SS	Lists the definitions for FAT file system sector size.
	FAT_FS_USE_LFN	Lists the definitions for FAT file system LFN selection.
	SYS_FS_HANDLE_INVALID	Invalid file handle
	SYS_FS_FILE_DIR_ATTR	Enumerated data type identifying the various attributes for file/directory.
	SYS_FS_TIME	The structure to specify the time for a file or directory.
	SYS_FS_FORMAT	Specifes the partitioning rule.
	SYS_FS_EVENT	Identifies the possible file system events.
	SYS_FS_EVENT_HANDLER	Pointer to the File system Handler function.

Description

This section describes the APIs of the File System Service Library.

Refer to each section for a detailed description.

a) File and Directory Operation Functions

SYS_FS_FileClose Function

Closes a file.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_FileClose(SYS_FS_HANDLE handle);
```

Returns

SYS_FS_RES_SUCCESS - File close operation was successful. SYS_FS_RES_FAILURE - File close operation failed. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function closes an opened file.

Remarks

None.

Preconditions

A valid file handle must be obtained before closing a file.

Example

```
SYS_FS_HANDLE fileHandle;
fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
        (SYS_FS_FILE_OPEN_READ));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}

SYS_FS_FileClose(fileHandle);
```

Parameters

Parameters	Description
handle	A valid handle, which was obtained while opening the file.

Function

```
SYS_FS_RESULT SYS_FS_FileClose
(
    SYS_FS_HANDLE handle
);
```

SYS_FS_FileEOF Function

Checks for end of file.

File

sys_fs.h

C

```
bool SYS_FS_FileEOF(SYS_FS_HANDLE handle);
```

Returns

On success returns true indicating that the file pointer has reached the end of the file. On failure returns false. This could be due to file pointer having not reached the end of the file. Or due to an invalid file handle. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

Checks whether or not the file position indicator is at the end of the file.

Remarks

None.

Preconditions

A valid file handle must be obtained before knowing a EOF.

Example

```
SYS_FS_HANDLE fileHandle;
bool eof;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
        (SYS_FS_FILE_OPEN_READ));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}
...
...

eof = SYS_FS_FileEOF(fileHandle);

if(eof == false)
{
    // Check the error state using SYS_FS_FileError
}
```

Parameters

Parameters	Description
handle	file handle obtained during file Open.

Function

bool SYS_FS_FileEOF

(

SYS_FS_HANDLE handle

);

SYS_FS_FileOpen Function

Opens a file.

File

sys_fs.h

C

```
SYS_FS_HANDLE SYS_FS_FileOpen(const char* fname, SYS_FS_FILE_OPEN_ATTRIBUTES attributes);
```

Returns

On success - A valid file handle will be returned On failure - SYS_FS_HANDLE_INVALID. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function opens a file with the requested attributes.

Remarks

None.

Preconditions

Prior to opening a file, the name of the volume on which the file resides should be known and the volume should be mounted.

Example

```
SYS_FS_HANDLE fileHandle;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
        (SYS_FS_FILE_OPEN_READ));
```

```
if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open succeeded.
}
else
{
    // File open failed.
}

// Using SYS_FS_CurrentDriveSet () function.

SYS_FS_HANDLE fileHandle;

SYS_FS_CurrentDriveSet("/mnt/myDrive");

fileHandle = SYS_FS_FileOpen("FILE.JPG", (SYS_FS_FILE_OPEN_READ));
if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open succeeded.
}
else
{
    // File open failed.
}
```

Parameters

Parameters	Description
fname	The name of the file to be opened along with the path. The fname
format is as follows	"/mnt/volumeName/dirName/fileName". volumeName is the name of the volume/drive. dirName is the name of the directory under which the file is located. fileName is the name of the file to be opened. The "/mnt/volumeName" portion from the fName can be omitted if the SYS_FS_CurrentDriveSet () has been invoked to set the current drive/volume.
attributes	Access mode of the file, of type SYS_FS_FILE_OPEN_ATTRIBUTES

Function

SYS_FS_HANDLE SYS_FS_FileOpen
(
const char* fname,
SYS_FS_FILE_OPEN_ATTRIBUTES attributes
);

SYS_FS_FileSeek Function

Moves the file pointer by the requested offset.

File

sys_fs.h

C

```
int32_t SYS_FS_FileSeek(SYS_FS_HANDLE fildes, int32_t offset, SYS_FS_FILE_SEEK_CONTROL whence);
```

Returns

On success - The number of bytes by which file pointer is moved (0 or positive number) On Failure - (-1) If the chosen offset value was (-1), the success or failure can be determined with SYS_FS_Error.

Description

This function sets the file pointer for a open file associated with the file handle, as follows: whence = SYS_FS_SEEK_SET - File offset is set to offset bytes from the beginning. whence = SYS_FS_SEEK_CUR - File offset is set to its current location plus offset. whence = SYS_FS_SEEK_END - File offset is set to the size of the file plus offset. The offset specified for this option should be negative for the file pointer to be valid.

Trying to move the file pointer using SYS_FS_FileSeek, beyond the range of file will only cause the pointer to be moved to the last location of the file.

Remarks

None.

Preconditions

A valid file handle must be obtained before seeking a file.

Example

```
SYS_FS_HANDLE fileHandle;
int status;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
        (SYS_FS_FILE_OPEN_READ));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}
...
...

status = SYS_FS_FileSeek(fileHandle, 5, SYS_FS_SEEK_CUR);

if((status != -1) && (status == 5))
{
    // Success
}
```

Parameters

Parameters	Description
handle	A valid file handle obtained during file open.
offset	The number of bytes which act as file offset. This value could be a positive or negative value.
whence	Type of File Seek operation as specified in SYS_FS_FILE_SEEK_CONTROL.

Function

int32_t SYS_FS_FileSeek
(
    SYS_FS_HANDLE handle,
int32_t offset,
    SYS_FS_FILE_SEEK_CONTROL whence
);

SYS_FS_FileSize Function

Returns the size of the file in bytes.

File

sys_fs.h

C

```
int32_t SYS_FS_FileSize(SYS_FS_HANDLE handle);
```

Returns

On success returns the size of the file in bytes. On failure returns -1. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function returns the size of the file as pointed by the handle.

Remarks

None.

Preconditions

A valid file handle must be obtained before knowing a file size.

Example

```
SYS_FS_HANDLE fileHandle;
long fileSize;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
        (SYS_FS_FILE_OPEN_READ));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}
...
...

fileSize = SYS_FS_FileSize(fileHandle);

if(fileSize != -1)
{
    // Success
}
```

Parameters

Parameters	Description
handle	File handle obtained during file Open.

Function

int32_t SYS_FS_FileSize
(
    SYS_FS_HANDLE handle
);

SYS_FS_FileTell Function

Obtains the file pointer position.

File

sys_fs.h

C

```
int32_t SYS_FS_FileTell(SYS_FS_HANDLE handle);
```

Returns

On success returns the current file position. On failure returns -1. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

Obtains the current value of the file position indicator for the file pointed to by handle.

Remarks

None.

Preconditions

A valid file handle must be obtained before performing a file tell.

Example

```
SYS_FS_HANDLE fileHandle;
int32_t tell;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
        (SYS_FS_FILE_OPEN_READ));
```

```
if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}
...
...

tell = SYS_FS_FileTell(fileHandle);

if(tell != -1)
{
    // Success
}
```

Parameters

Parameters	Description
handle	File handle obtained during file Open.

Function

int32_t SYS_FS_FileTell
(
    SYS_FS_HANDLE handle
);

SYS_FS_FileNameGet Function

Reads the file name.

File

sys_fs.h

C

```
bool SYS_FS_FileNameGet(SYS_FS_HANDLE handle, uint8_t* cName, uint16_t wLen);
```

Returns

Returns true if the file name was read successfully. Returns false if the file name was not read successfully. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function reads the file name of a file that is already open.

Remarks

None.

Preconditions

The file handle referenced by handle is already open.

Example

```
SYS_FS_HANDLE fileHandle;
bool stat;
uint8_t fileName[255];

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
        (SYS_FS_FILE_OPEN_READ));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}
...
...

stat = SYS_FS_FileNameGet(fileHandle, fileName, 8 );
```

```
if(stat == false)
{
    // file not located based on handle passed
    // Check the error state using SYS_FS_FileError
}
```

Parameters

Parameters	Description
handle	File handle obtained during file Open.
cName	Where to store the name of the file.
wLen	The maximum length of data to store in cName.

Function

```
bool SYS_FS_FileNameGet
(
    SYS_FS_HANDLE handle,
uint8_t* cName,
uint16_t wLen
);
```

SYS_FS_FilePrintf Function

Writes a formatted string into a file.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_FilePrintf(SYS_FS_HANDLE handle, const char * string, ...);
```

Returns

SYS_FS_RES_SUCCESS - Formatted string write operation was successful. SYS_FS_RES_FAILURE - Formatted string write operation was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function writes a formatted string into a file.

Remarks

None.

Preconditions

The file into which a string has to be written, must exist and should be open.

Example

```
SYS_FS_RESULT res;
SYS_FS_HANDLE fileHandle;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.txt", (SYS_FS_FILE_OPEN_WRITE_PLUS));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}

// Write a string
res = SYS_FS_FilePrintf(fileHandle, "%d", 1234);
if( res != SYS_FS_RES_SUCCESS)
{
    // write operation failed.
}
```

Parameters

Parameters	Description
handle	File handle to which formatted string is to be written.
string	Pointer to formatted string which has to be written into file.

Function

```
SYS_FS_RESULT SYS_FS_FilePrintf
(
    SYS_FS_HANDLE handle,
const char *string,
...
);
```

SYS_FS_FileTestError Function

Checks for errors in the file.

File

sys_fs.h

C

```
bool SYS_FS_FileTestError(SYS_FS_HANDLE handle);
```

Returns

On success returns false indicating that the file has no errors. On failure returns true. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function checks whether or not file has any errors.

Remarks

None.

Preconditions

A valid file handle must be obtained before passing to the function

Example

```
SYS_FS_HANDLE fileHandle;
bool err;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG", (SYS_FS_FILE_OPEN_READ));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}
...
...

err = SYS_FS_FileTestError(fileHandle);
if(err == true)
{
    // either file has error, or there
    // was an error in working with the "SYS_FS_FileTestError" function
}
```

Parameters

Parameters	Description
handle	file handle obtained during file Open.

Function

bool SYS_FS_FileTestError
(
SYS_FS_HANDLE handle
);

SYS_FS_FileDirectoryModeSet Function

Sets the mode for the file or directory.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_FileDirectoryModeSet(const char* path, SYS_FS_FILE_DIR_ATTR attr, SYS_FS_FILE_DIR_ATTR
mask);
```

Returns

SYS_FS_RES_SUCCESS - Mode set operation was successful. SYS_FS_RES_FAILURE - Mode set operation was unsucessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function sets the mode for a file or directory from the specified list of attributes.

Remarks

None.

Preconditions

The file or directory for which the mode is to be set must exist.

Example

```
// Set read-only flag, clear archive flag and others are retained.
SYS_FS_FileDirectoryModeSet("file.txt", SYS_FS_ATTR_RDO, SYS_FS_ATTR_RDO | SYS_FS_ATTR_ARC);
```

Parameters

Parameters	Description
path	Path for the file/directory, for which the mode is to be set.
attr	Attribute flags to be set in one or more combination of the type SYS_FS_FILE_DIR_ATTR. The specified flags are set and others are cleared.
mask	Attribute mask of type SYS_FS_FILE_DIR_ATTR that specifies which attribute is changed. The specified attributes are set or cleared.

Function

SYS_FS_RESULT SYS_FS_FileDirectoryModeSet
(
const char* path,
SYS_FS_FILE_DIR_ATTR attr,
SYS_FS_FILE_DIR_ATTR mask
);

SYS_FS_FileDirectoryRemove Function

Removes a file or directory.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_FileDirectoryRemove(const char* path);
```

Returns

SYS_FS_RES_SUCCESS - Indicates that the file or directory remove operation was successful. SYS_FS_RES_FAILURE - Indicates that the file or directory remove operation was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function removes a file or directory as specified by the path.

Remarks

None.

Preconditions

- The disk has to be mounted before a directory could be removed.
- The file or directory to be removed has to present.
- The file/sub-directory must not have read-only attribute (AM_RDO), or the function will be rejected with FR_DENIED.
- The sub-directory must be empty and must not be current directory, or the function will be rejected with FR_DENIED.
- The file/sub-directory must not be opened.

Example

```
SYS_FS_RESULT res;

res = SYS_FS_FileDirectoryRemove("Dir1");

if(res == SYS_FS_RES_FAILURE)
{
    // Directory remove operation failed
}
//...
//...
```

Parameters

Parameters	Description
path	Path of the File or directory to be removed.

Function

SYS_FS_RESULT SYS_FS_FileDirectoryRemove
(
const char* path
);

SYS_FS_FileDirectoryRenameMove Function

Renames or moves a file or directory.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_FileDirectoryRenameMove(const char * oldPath, const char * newPath);
```

Returns

SYS_FS_RES_SUCCESS - Rename/move operation was successful. SYS_FS_RES_FAILURE - Rename/move operation was unsucessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function renames or moves a file or directory.

Remarks

This function cannot move files/ directory from one drive to another. Do not rename/ move files which are open.

Preconditions

The file or directory to be renamed or moved must exist. This function cannot move files or directories from one drive to another. Do not rename or

move files that are open.

Example

```
SYS_FS_RESULT res;

// rename "file.txt" to "renamed_file.txt"
res = SYS_FS_FileDirectoryRenameMove("file.txt", "renamed_file.txt");
if( res != SYS_FS_RES_SUCCESS)
{
    // Rename operation failed.
}

// Now, move "renamed_file.txt" inside directory "Dir1"
res = SYS_FS_FileDirectoryRenameMove("renamed_file.txt", "Dir1/renamed_file.txt");
if( res != SYS_FS_RES_SUCCESS)
{
    // File move operation failed.
}
```

Parameters

Parameters	Description
oldPath	Path for the file/directory, which has to be renamed/moved.
newPath	New Path for the file/directory.

Function

SYS_FS_RESULT SYS_FS_FileDirectoryRenameMove
(
const char *oldPath,
const char *newPath
);

SYS_FS_FileDirectoryTimeSet Function

Sets or changes the time for a file or directory.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_FileDirectoryTimeSet(const char* path, SYS_FS_TIME * time);
```

Returns

SYS_FS_RES_SUCCESS - Set time operation was successful. SYS_FS_RES_FAILURE - Set time operation was unsucessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function sets or change the time for a file or directory.

Remarks

None.

Preconditions

The file/directory for which time is to be set must exist.

Example

```
void setTime(void)
{
    SYS_FS_RESULT res;
    SYS_FS_TIME time;

    time.packedTime = 0;

    // All FAT FS times are calculated based on 0 = 1980
```

```
        time.discreteTime.year = (2013 - 1980);  // Year is 2013
        time.discreteTime.month = 8;             // Month (August)
        time.discreteTime.day = 9;               // Day (9)
        time.discreteTime.hour = 15;             // 3 PM
        time.discreteTime.minute = 06;           // 06 minutes
        time.discreteTime.second = 00;           // 00 seconds

        res = SYS_FS_FileDirectoryTimeSet("file.txt", &time);
        if( res != SYS_FS_RES_SUCCESS)
        {
            // time change has gone wrong
        }
    }
```

Parameters

Parameters	Description
path	A path for the file/directory, for which the time is to be set.
ptr	Pointer to the structure of type SYS_FS_TIME, which contains the time data to be set.

Function

```
  SYS_FS_RESULT SYS_FS_FileDirectoryTimeSet
(
const char* path,
    SYS_FS_TIME *time
);
```

SYS_FS_FileTruncate Function

Truncates a file

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_FileTruncate(SYS_FS_HANDLE handle);
```

Returns

SYS_FS_RES_SUCCESS - File truncate operation was successful. SYS_FS_RES_FAILURE - File truncate operation was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function truncates the file size to the current file read/write pointer. This function has no effect if the file read/write pointer is already pointing to end of the file.

Remarks

None.

Preconditions

A valid file handle has to be passed as input to the function. The file has to be opened in a mode where writes to file is possible (such as read plus or write mode).

Example

```
SYS_FS_HANDLE fileHandle;
size_t nbytes;
size_t bytes_read;
SYS_FS_RESULT res;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
        (SYS_FS_FILE_OPEN_READ));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}
```

```
// Read the file content
nbytes = sizeof(buf);
bytes_read = SYS_FS_FileRead(buf, nbytes, fileHandle);
// Truncate the file
res = SYS_FS_FileTruncate(fileHandle);
if(res != SYS_FS_RES_SUCCESS)
{
    // Truncation failed.
}

SYS_FS_FileClose(fileHandle);
```

Parameters

Parameters	Description
handle	A valid handle which was obtained while opening the file.

Function

```
SYS_FS_RESULT SYS_FS_FileTruncate
(
    SYS_FS_HANDLE handle
);
```

SYS_FS_FileCharacterPut Function

Writes a character to a file.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_FileCharacterPut(SYS_FS_HANDLE handle, char data);
```

Returns

SYS_FS_RES_SUCCESS - Write operation was successful. SYS_FS_RES_FAILURE - Write operation was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function writes a character to a file.

Remarks

None.

Preconditions

The file into which a character has to be written, has to be present and should have been opened.

Example

```
SYS_FS_RESULT res;
SYS_FS_HANDLE fileHandle;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG", (SYS_FS_FILE_OPEN_WRITE_PLUS));
if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}

// Write a character to the file.
res = SYS_FS_FileCharacterPut(fileHandle, 'c');
if(res != SYS_FS_RES_SUCCESS)
{
    // Character write operation failed.
}
```

Parameters

Parameters	Description
handle	file handle to which the character is to be written.
data	character to be written to the file.

Function

```
    SYS_FS_RESULT SYS_FS_FileCharacterPut
(
      SYS_FS_HANDLE handle,
char data
);
```

SYS_FS_FileStringGet Function

Reads a string from the file into a buffer.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_FileStringGet(SYS_FS_HANDLE handle, char* buff, uint32_t len);
```

Returns

SYS_FS_RES_SUCCESS - String read operation was successful. SYS_FS_RES_FAILURE - String read operation was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function reads a string of specified length from the file into a buffer. The read operation continues until

1. 'n' is stored
2. reached end of the file or
3. the buffer is filled with len - 1 characters.

The read string is terminated with a '0'.

Remarks

None.

Preconditions

The file from which a string has to be read, has to be present and should have been opened.

Example

```
SYS_FS_RESULT res;
SYS_FS_HANDLE fileHandle;
char buffer[100];

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG", (SYS_FS_FILE_OPEN_WRITE_PLUS));
if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}

// Read a string from the file.
res = SYS_FS_FileStringGet(fileHandle, buffer, 50);
if( res != SYS_FS_RES_SUCCESS)
{
    //String read operation failed.
}
```

Parameters

Parameters	Description
handle	Handle of the file from which string is to be read.
buff	Buffer in which the string is to be stored.

len	length of string to be read.

Function

SYS_FS_RESULT SYS_FS_FileStringGet
(
SYS_FS_HANDLE handle,
char* buff,
uint32_t len
);

SYS_FS_FileStringPut Function

Writes a string to a file.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_FileStringPut(SYS_FS_HANDLE handle, const char * string);
```

Returns

SYS_FS_RES_SUCCESS - String write operation was successful. SYS_FS_RES_FAILURE - String write operation was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function writes a string into a file. The string to be written should be NULL terminated. The terminator character will not be written.

Remarks

None.

Preconditions

The file into which a string has to be written, has to be present and should have been opened.

Example

```
SYS_FS_RESULT res;
SYS_FS_HANDLE fileHandle;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG", SYS_FS_FILE_OPEN_WRITE_PLUS));
if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}

// Write a string
res = SYS_FS_FileStringPut(fileHandle, "Hello World");
if(res != SYS_FS_RES_SUCCESS)
{
    // String write operation failed.
}
```

Parameters

Parameters	Description
handle	File handle to which string is to be written.
string	Pointer to the null terminated string which has to be written into file.

Function

SYS_FS_RESULT SYS_FS_FileStringPut
(
SYS_FS_HANDLE handle,
const char *string
);

SYS_FS_Error Function

Returns the last error.

File

sys_fs.h

C

```
SYS_FS_ERROR SYS_FS_Error();
```

Returns

Error code of type SYS_FS_ERROR.

Description

When a file system operation fails, the application can know the reason of failure by calling the SYS_FS_Error. This function only reports the errors which are not file (or file handle) specific. For example, for functions such as SYS_FS_Mount and SYS_FS_FileOpen, which do not take handle, any errors happening inside such function calls could be reported using SYS_FS_Error function. Even for functions, which take handle as its input parameters, the SYS_FS_Error function can be used to report the type of error for cases where the passed handle itself is invalid.

Remarks

None.

Preconditions

This function has to be called immediately after a failure is observed while doing a file operation. Any subsequent failure will overwrite the cause of previous failure.

Example

```
SYS_FS_HANDLE fileHandle;
SYS_FS_ERROR err;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG",
        (SYS_FS_FILE_OPEN_READ));

if(fileHandle == SYS_FS_HANDLE_INVALID)
{
    // If failure, now know the specific reason for failure
    err = SYS_FS_Error();
}
```

Function

SYS_FS_ERROR SYS_FS_Error
(
void
)

SYS_FS_FileError Function

Returns the file specific error.

File

sys_fs.h

C

```
SYS_FS_ERROR SYS_FS_FileError(SYS_FS_HANDLE handle);
```

Returns

Error code of type SYS_FS_ERROR.

Description

For file system functions which accepts valid handle, any error happening in those functions could be retrieved with SYS_FS_FileError. This function returns errors which are file specific.

Please note that if an invalid handle is passed to a file system function, in such a case, SYS_FS_FileError will not return the correct type of error,

as the handle was invalid. Therefore, it would be prudent to check the errors using the SYS_FS_Error function.

Remarks

None.

Preconditions

This function has to be called immediately after a failure is observed while doing a file operation. Any subsequent failure will overwrite the cause of previous failure.

Example

```
 ...
const char *buf = "Hello World";
size_t nbytes;
size_t bytes_written;
SYS_FS_HANDLE fd;
SYS_FS_ERROR err;
...

bytes_written = SYS_FS_FileWrite((const void *)buf, nbytes, fd);

if(bytes_written == -1)
{
    // error while writing file
    // find the type (reason) of error
    err = SYS_FS_FileError(fd);
}
```

Parameters

Parameters	Description
handle	A valid file handle

Function

SYS_FS_ERROR SYS_FS_FileError
(
SYS_FS_HANDLE handle
);

SYS_FS_FileRead Function

Read data from the file.

File

sys_fs.h

C

```
size_t SYS_FS_FileRead(SYS_FS_HANDLE handle, void * buf, size_t nbyte);
```

Returns

On success returns the number of bytes read successfully(0 or positive number). On failure returns -1. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function attempts to read nbyte bytes of data from the file associated with the file handle into the buffer pointed to by buf.

Remarks

None.

Preconditions

A valid file handle must be obtained before reading a file.

Example

```
...
char buf[20];
```

```
size_t nbytes;
size_t bytes_read;
SYS_FS_HANDLE fd;
...
nbytes = sizeof(buf);
bytes_read = SYS_FS_FileRead(fd, buf, nbytes);
...
```

Parameters

Parameters	Description
handle	File handle obtained during file open.
buf	Pointer to buffer into which data is read.
nbyte	Number of bytes to be read

Function

size_t SYS_FS_FileRead
(
SYS_FS_HANDLE handle,
void *buf,
size_t nbyte
);

SYS_FS_FileWrite Function

Writes data to the file.

File

sys_fs.h

C

```
size_t SYS_FS_FileWrite(SYS_FS_HANDLE handle, const void * buf, size_t nbyte);
```

Returns

On success returns the number of bytes written successfully(0 or positive number). On failure returns -1. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function attempts to write nbyte bytes from the buffer pointed to by buf to the file associated with the file handle.

Remarks

None.

Preconditions

A valid file handle must be obtained before writing a file.

Example

```
...
const char *buf = "Hello World";
size_t nbytes;
size_t bytes_written;
SYS_FS_HANDLE fd;
...

bytes_written = SYS_FS_FileWrite(fd, (const void *)buf, nbytes);
...
```

Parameters

Parameters	Description
handle	File handle obtained during file open.
buf	Pointer to buffer from which data is to be written
nbyte	Number of bytes to be written

Function

size_t SYS_FS_FileWrite
(
SYS_FS_HANDLE handle,
const void *buf,
size_t nbyte
);

SYS_FS_CurrentWorkingDirectoryGet Function

Gets the current working directory

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_CurrentWorkingDirectoryGet(char * buff, uint32_t len);
```

Returns

SYS_FS_RES_SUCCESS - Get current working directory operation was successful. SYS_FS_RES_FAILURE - Get current working directory operation was unsucessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function gets the current working directory path along with the working drive.

Remarks

None.

Preconditions

At least one disk must be mounted.

Example

```
SYS_FS_RESULT res;
char buffer[16];

switch(appState)
{
    case TRY_MOUNT:
        if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive", FAT, 0, NULL) != SYS_FS_RES_SUCCESS)
        {
            // Failure, try mounting again
        }
        else
        {
            // Mount was successful. Create a directory.
            appState = CREATE_DIR;
        }
        break;

    case CREATE_DIR:
        res = SYS_FS_DirectoryMake("Dir1");
        if(res == SYS_FS_RES_FAILURE)
        {
            // Directory creation failed
            appState = ERROR;
        }
        else
        {
            // Directory creation was successful. Change to the new
            // directory.
            appState = CHANGE_DIR;
        }
        break;
```

```
        case CHANGE_DIR:
            res = SYS_FS_DirectoryChange("Dir1");
            if(res == SYS_FS_RES_FAILURE)
            {
                // Directory change failed
                appState = ERROR;
            }
            else
            {
                // Directory change was successful. Get current working
                // directory
                appState = GET_CWD;
            }
            break;

        case GET_CWD:
            res = SYS_FS_CurrentWorkingDirectoryGet(buffer, 15);
            if(res == SYS_FS_RES_FAILURE)
            {
                // Get current directory operation failed
                appState = ERROR;
            }
            break;
    }
```

Parameters

Parameters	Description
buff	Pointer to a buffer which will contain the name of the current working directory and drive, once the function completes.
len	Size of the buffer.

Function

SYS_FS_RESULT SYS_FS_CurrentWorkingDirectoryGet

(

char *buff,

uint32_t len

);

SYS_FS_DirectoryChange Function

Changes to a the directory specified.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_DirectoryChange(const char* path);
```

Returns

SYS_FS_RES_SUCCESS - Indicates that the directory change operation was successful. SYS_FS_RES_FAILURE - Indicates that the directory change operation was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function changes the present directory to a new directory.

Remarks

None.

Preconditions

The disk has to be mounted and the directory to be changed must exist.

Example

```
SYS_FS_RESULT res;
```

```
res = SYS_FS_DirectoryChange("Dir1");

if(res == SYS_FS_RES_FAILURE)
{
    // Directory change failed
}
```

Parameters

Parameters	Description
path	Path of the directory to be changed to.

Function

SYS_FS_RESULT SYS_FS_DirectoryChange
(
const char* path
);

SYS_FS_DirectoryMake Function

Makes a directory.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_DirectoryMake(const char* path);
```

Returns

SYS_FS_RES_SUCCESS - Indicates that the creation of the directory was successful. SYS_FS_RES_FAILURE - Indicates that the creation of the directory was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function makes a new directory as per the specified path.

Remarks

None.

Preconditions

The disk has to be mounted before a directory could be made.

Example

```
SYS_FS_RESULT res;

res = SYS_FS_DirectoryMake("Dir1");

if(res == SYS_FS_RES_FAILURE)
{
    // Directory make failed
}
```

Parameters

Parameters	Description
path	Path of the new directory

Function

SYS_FS_RESULT SYS_FS_DirectoryMake
(
const char* path
);

SYS_FS_DirClose Function

Closes an opened directory.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_DirClose(SYS_FS_HANDLE handle);
```

Returns

SYS_FS_RES_SUCCESS - Directory close operation was successful. SYS_FS_RES_FAILURE - Directory close operation was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function closes a directory that was opened earlier opened with the SYS_FS_DirOpen function.

Remarks

None.

Preconditions

A valid directory handle must be obtained before closing the directory.

Example

```
SYS_FS_HANDLE dirHandle;

dirHandle = SYS_FS_DirOpen("/mnt/myDrive/Dir1");

if(dirHandle != SYS_FS_HANDLE_INVALID)
{
    // Directory open is successful
}

// Perform required operation on the directory

// Close the directory
if(SYS_FS_DirClose(dirHandle) == SYS_FS_RES_FAILURE)
{
    // Close operation failed.
}
```

Parameters

Parameters	Description
handle	directory handle obtained during directory open.

Function

```
SYS_FS_RESULT SYS_FS_DirClose
(
    SYS_FS_HANDLE handle
);
```

SYS_FS_DirOpen Function

Open a directory

File

sys_fs.h

C

```
SYS_FS_HANDLE SYS_FS_DirOpen(const char* path);
```

Returns

On success a valid handle to the directory will be returned. On failure SYS_FS_HANDLE_INVALID will be returned. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function opens the requested directory.

Remarks

None

Preconditions

The volume on which the directory is present should be mounted.

Example

```
SYS_FS_HANDLE dirHandle;

dirHandle = SYS_FS_DirOpen("/mnt/myDrive/Dir1");
// For root directory, end with a "/"
// dirHandle = SYS_FS_DirOpen("/mnt/myDrive/");

if(dirHandle != SYS_FS_HANDLE_INVALID)
{
    // Directory open is successful
}
```

Parameters

Parameters	Description
path	Path to the directory along with the volume name. The string of volume and directory name has to be preceded by "/mnt/". Also, the volume name and directory name has to be separated by a slash "/". If the directory specified is only the root directory, the path has to be ended with "/".

Function

SYS_FS_HANDLE SYS_FS_DirOpen

(

const char* path

);

SYS_FS_DirRead Function

Reads the specified directory.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_DirRead(SYS_FS_HANDLE handle, SYS_FS_FSTAT * stat);
```

Returns

SYS_FS_RES_SUCCESS - Indicates that the directory read operation was successful. SYS_FS_RES_FAILURE - Indicates that the directory read operation was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function reads the files and directories specified in the open directory.

Remarks

None.

Preconditions

A valid directory handle must be obtained before reading a directory.

Example

```
SYS_FS_HANDLE dirHandle;
SYS_FS_FSTAT stat;
char longFileName[300];
uintptr_t  longFileSize;

dirHandle = SYS_FS_DirOpen("/mnt/myDrive/Dir1");

if(dirHandle != SYS_FS_HANDLE_INVALID)
{
    // Directory open is successful
}

// If long file name is used, the following elements of the "stat"
// structure needs to be initialized with address of proper buffer.
stat.lfname = longFileName;
stat.lfsize = 300;

if(SYS_FS_DirRead(dirHandle, &stat) == SYS_FS_RES_FAILURE)
{
    // Directory read failed.
}
```

Parameters

Parameters	Description
handle	Directory handle obtained during directory open.
stat	Empty structure of type SYS_FS_FSTAT, where the properties of the open directory will be populated after the SYS_FS_DirRead function returns successfully. If LFN is used, then the "lfname" member of the SYS_FS_FSTAT structure should be initialized with the address of suitable buffer. Also, the "lfsize" should be initialized with the size of the buffer. Once the function returns, the buffer whose address is held in "lfname" will have the file name(long file name)

Function

```
  SYS_FS_RESULT SYS_FS_DirRead
(
    SYS_FS_HANDLE handle,
    SYS_FS_FSTAT *stat
);
```

SYS_FS_DirSearch Function

Searches for a file or directory.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_DirSearch(SYS_FS_HANDLE handle, const char * name, SYS_FS_FILE_DIR_ATTR attr,
SYS_FS_FSTAT * stat);
```

Returns

SYS_FS_RES_SUCCESS - Indicates that the file or directory was found. The stat parameter will contain information about the file or directory.
SYS_FS_RES_FAILURE - Indicates that the file or directory was not found. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function searches for the requested file or directory. The file or directory is specified in the attr parameter, which is of type SYS_FS_FILE_DIR_ATTR.

Remarks

None.

Preconditions

A valid directory handle must be obtained before searching the directory.

Example

```
SYS_FS_HANDLE dirHandle;
SYS_FS_FSTAT stat;
char longFileName[300];
uintptr_t  longFileSize;

dirHandle = SYS_FS_DirOpen("/mnt/myDrive/Dir1");

if(dirHandle != SYS_FS_HANDLE_INVALID)
{
   // Directory open is successful
}

// If long file name is used, the following elements of the "stat"
// structure needs to be initialized with address of proper buffer.
stat.lfname = longFileName;
stat.lfsize = 300;

if(SYS_FS_DirSearch(dirHandle, "FIL*.*", SYS_FS_ATTR_ARC, &stat) == SYS_FS_RES_FAILURE)
{
   // Specified file not found
}
else
{
   // File found. Read the complete file name from "stat.lfname" and
   // other file parameters from the "stat" structure
}
```

Parameters

Parameters	Description
handle	directory handle obtained during directory open.
name	name of file or directory needed to be searched. The file
name can have wild card entries as follows	• - Indicates the rest of the filename or extension can vary (e.g. FILE.*)
?	Indicates that one character in a filename can vary (e.g. F?LE.T?T)
attr	Attribute of the name of type SYS_FS_FILE_DIR_ATTR. This attribute specifies whether to search a file or a directory. Other attribute types could also be specified.
stat	Empty structure of type SYS_FS_FSTAT, where the properties of the file/directory will be populated. If LFN is used, then the "lfname" member of the SYS_FS_FSTAT structure should be initialized with address of suitable buffer. Also, the "lfsize" should be initialized with the size of buffer. Once the function returns, the buffer whose address is held in "lfname" will have the file name (long file name).

Function

```
  SYS_FS_RESULT SYS_FS_DirSearch
(
    SYS_FS_HANDLE handle,
const char * name,
    SYS_FS_FILE_DIR_ATTR attr,
    SYS_FS_FSTAT *stat
);
```

SYS_FS_FileSync Function

Flushes the cached information when writing to a file.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_FileSync(SYS_FS_HANDLE handle);
```

Returns

SYS_FS_RES_SUCCESS - File sync operation was successful. SYS_FS_RES_FAILURE - File sync operation was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function flushes the cached information when writing to a file. The SYS_FS_FileSync function performs the same process as SYS_FS_FileClose function; however, the file is left open and can continue read/write/seek operations to the file.

Remarks

None.

Preconditions

A valid file handle has to be passed as input to the function. The file which has to be flushed, has to be present and should have been opened in write mode.

Example

```
SYS_FS_RESULT res;
SYS_FS_HANDLE fileHandle;
const char *buf = "Hello World";
size_t nbytes;
size_t bytes_written;

fileHandle = SYS_FS_FileOpen("/mnt/myDrive/FILE.JPG", (SYS_FS_FILE_OPEN_WRITE_PLUS));

if(fileHandle != SYS_FS_HANDLE_INVALID)
{
    // File open is successful
}

// Write data to the file
bytes_written = SYS_FS_FileWrite((const void *)buf, nbytes, fileHandle);

// Flush the file
res = SYS_FS_FileSync(fileHandle);
if( res != SYS_FS_RES_SUCCESS)
{
    // renaming has gone wrong
}
```

Parameters

Parameters	Description
handle	valid file handle

Function

```
SYS_FS_RESULT SYS_FS_FileSync
(
    SYS_FS_HANDLE handle
);
```

SYS_FS_DirRewind Function

Rewinds to the beginning of the directory.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_DirRewind(SYS_FS_HANDLE handle);
```

Returns

SYS_FS_RES_SUCCESS - Directory rewind operation was successful. SYS_FS_RES_FAILURE - Directory rewind operation was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error or SYS_FS_FileError.

Description

This function rewinds the directory to the start. Once a search of directory or directory read is completed, the rewind function is used to begin searching the directory from the start.

Remarks

None.

Preconditions

A valid directory handle must be obtained before reading a directory.

Example

```
SYS_FS_HANDLE dirHandle;
SYS_FS_FSTAT stat;
char longFileName[300];
uintptr_t  longFileSize;

dirHandle = SYS_FS_DirOpen("/mnt/myDrive/Dir1");

if(dirHandle != SYS_FS_HANDLE_INVALID)
{
    // Directory open is successful
}

// If long file name is used, the following elements of the "stat"
// structure needs to be initialized with address of proper buffer.
stat.lfname = longFileName;
stat.lfsize = 300;

if(SYS_FS_DirRead(dirHandle, &stat) == SYS_FS_RES_FAILURE)
{
   // Directory read operation failed.
}

// Do more search
// Do some more search

// Now, rewind the directory to begin search from start

if(SYS_FS_DirRewind(dirHandle) == SYS_FS_RES_FAILURE)
{
   // Directory rewind failed.
}
```

Parameters

Parameters	Description
handle	directory handle obtained during directory open.

Function

```
SYS_FS_RESULT SYS_FS_DirRewind
(
    SYS_FS_HANDLE handle
);
```

SYS_FS_FileStat Function

Gets file status.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_FileStat(const char * fname, SYS_FS_FSTAT * buf);
```

Returns

SYS_FS_RES_SUCCESS - File stat operation was successful. SYS_FS_RES_FAILURE - File stat operation was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function obtains information about a file associated with the file name, and populates the information in the structure pointed to by buf. This function can read the status of file regardless of whether a file is opened or not.

Remarks

None.

Preconditions

Prior to opening a file, the name of the volume on which the file resides should be known and the volume should be mounted.

Example

```
SYS_FS_fStat fileStat;

if(SYS_FS_FileStat("/mnt/myDrive/FILE.TXT", &fileStat) == SYS_FS_RES_SUCCESS)
{
    // Successfully read the status of file "FILE.TXT"
}
```

Parameters

Parameters	Description
fname	Name of the file with the path and the volume name. The string of volume and file name has to be preceded by "/mnt/". Also, the volume name and file name has to be separated by a slash "/".
buf	pointer to SYS_FS_FSTAT structure.

Function

```
SYS_FS_RESULT SYS_FS_FileStat
(
const char   *fname,
    SYS_FS_FSTAT *buf
)
```

b) General Operation Functions

SYS_FS_Initialize Function

Initializes the file system abstraction layer (sys_fs layer).

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_Initialize(const void* initData);
```

Returns

SYS_FS_RES_SUCCESS - SYS FS Layer was initialized successfully. SYS_FS_RES_FAILURE - SYS FS Layer initialization failed. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function initializes the abstraction layer (sys_fs layer) and sets up the necessary parameters.

Preconditions

This is the first function to be called during usage of sys_fs. Calling other functions of sys_fs without initializing the sys_fs will cause unpredictable

behavior.

Example

```
// This code shows an example of how the SYS FS is initialized
// Only one file system is used

#define SYS_FS_MAX_FILE_SYSTEM_TYPE            1

// Function pointer table for FAT FS
const SYS_FS_FUNCTIONS FatFsFunctions =
{
    .mount   = f_mount,
    .unmount = f_unmount,
    .open    = f_open,
    .read    = f_read,
    .write   = f_write,
    .close   = f_close,
    .seek    = f_lseek,
    .tell    = f_tell,
    .eof     = f_eof,
    .size    = f_size,
    .fstat   = f_stat,
};

const SYS_FS_REGISTRATION_TABLE sysFSInit [ SYS_FS_MAX_FILE_SYSTEM_TYPE ] =
{
    {
    .nativeFileSystemType = FAT,
    .nativeFileSystemFunctions = &FatFsFunctions
    }
};

SYS_FS_Initialize((const void *)sysFSInit);
```

Parameters

Parameters	Description
initData	Pointer to an array of type SYS_FS_REGISTRATION_TABLE. The number of elements of array is decided by the definition SYS_FS_MAX_FILE_SYSTEM_TYPE. If the application uses one file system (say only FAT FS), SYS_FS_MAX_FILE_SYSTEM_TYPE is defined to be 1. Otherwise, if the application uses 2 file systems (say FAT FS and MPFS2), SYS_FS_MAX_FILE_SYSTEM_TYPE is defined to be 2.

Function

SYS_FS_RESULT SYS_FS_Initialize
(
const void* initData
);

SYS_FS_Mount Function

Mounts the file system.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_Mount(const char * devName, const char * mountName, SYS_FS_FILE_SYSTEM_TYPE
filesystemtype, unsigned long mountflags, const void * data);
```

Returns

SYS_FS_RES_SUCCESS - Mount was successful. SYS_FS_RES_FAILURE - Mount was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

The mount command attaches the file system specified to a volume. The call to the mount should be non blocking in nature. The application code

has to allow the SYS_FS_Tasks to run periodically while calling the SYS_FS_Mount function. If the SYS_FS_Mount is called in a blocking mode, then the SYS_Tasks() never gets a chance to run and therefore, the media will not be analyzed and finally, the SYS_FS_Mount will never succeed. This will result in a deadlock.

There is no mechanism available for the application to know if the specified volume (devName) is really attached or not. The only available possibility is to keep trying to mount the volume (with the devname), until success is achieved.

It is prudent that the application code implements a time-out mechanism while trying to mount a volume (by calling SYS_FS_Mount). The trial for mount should continue at least 10 times before before assuming that the mount will never succeed. This has to be done for every new volume to be mounted.

The standard names for volumes (devName) used in the MPLAB Harmony file system is as follows: NVM - "nvm" "media number" "volume number" SD card - "mmcblk" "media number" "volume number" MSD - "sd" "media number" "volume number"

Where, "media number" a, b, c... depends on the number of the type of connected media, and where, "volume number" 1, 2, 3... depends on the number of partitions in that media.

The convention for assigning names to volumes is further described below with examples:

If a SD card (with four partitions) is attached to the system, and assuming all four partitions are recognized, there will be four devNames:

1. mmcblka1
2. mmcblka2
3. mmcblka3 and
4. mmcblka4

Subsequently, if NVM media is attached that has only one partition, the devname will be: nvma1.

Later, if another SD card is attached to the system that has one partition, the devname will be mmcblkb1.

Finally, there will be six volume names (or devNames), which are available for the application to be mounted and used for the file system.

Remarks

None

Preconditions

The "devName" name for the volume has to be known. The file system type with which each of the volumes are formatted has to be known. Trying to mount a volume with a file system which is different from what the volume is actually formatted, will cause mount failure.

Example

```
switch(appState)
{
    case TRY_MOUNT:
        if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive", FAT, 0, NULL) != SYS_FS_RES_SUCCESS)
        {
            // Failure, try mounting again
        }
        else
        {
            // Mount was successful. Do further file operations
            appState = DO_FURTHER_STUFFS;
        }
    break;
}
```

Parameters

Parameters	Description
devName	The device name (name of volume) which needs to be mounted. The devName has to be preceded by the string "/dev/".
mountName	Mount name for the device to be mounted. This is a name provided by the user. In future, while accessing the mounted volume (say, during SYS_FS_FileOpen operation), the mountName is used to refer the path for file. The mount name has to be preceded by the string "/mnt/"
filesystemtype	Native file system of SYS_FS_FILE_SYSTEM_TYPE type.
mountflags	Mounting control flags. This parameter is reserved for future enhancements. Therefore, always pass zero.
data	The data argument is interpreted by the different file systems. This parameter is reserved for future enhancements. Therefore, always pass NULL.

Function

SYS_FS_RESULT SYS_FS_Mount

(

const char *devName,

const char *mountName,
SYS_FS_FILE_SYSTEM_TYPE filesystemtype,
unsigned long mountflags,
const void *data
);

SYS_FS_Tasks Function

Maintains the File System tasks and functionalities.

File

sys_fs.h

C

```
void SYS_FS_Tasks();
```

Returns

None.

Description

This function is used to run the various tasks and functionalities of sys_fs layer.

Remarks

This function is not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

Preconditions

The SYS_FS_Initialize routine must have been called before running the tasks.

Example

```
void SYS_Tasks ( void )
{
    SYS_FS_Tasks ();
    // Do other tasks
}
```

Function

void SYS_FS_Tasks
(
void
);

SYS_FS_Unmount Function

Unmounts the file system.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_Unmount(const char * mountName);
```

Returns

SYS_FS_RES_SUCCESS - Unmount was successful. SYS_FS_RES_FAILURE - Unmount was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function removes (unmounts) the attachment of the volume from the file system.

Preconditions

The volume name has to be know in order to pass as input to Unmount. The specified volume name to be unmounted should have been already mounted.

Example

```
if(SYS_FS_Unmount("/mnt/myDrive") != SYS_FS_RES_SUCCESS)
{
    // Failure, try unmounting again
}
else
{
    // Unmount was successful.
}
```

Parameters

Parameters	Description
mountName	Mount name for the volume to be unmounted. The mount name has to be preceded by the string "/mnt/".

Function

SYS_FS_RESULT SYS_FS_Unmount

(

const char *mountName

);

SYS_FS_CurrentDriveGet Function

Gets the current drive

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_CurrentDriveGet(char* buffer);
```

Returns

SYS_FS_RES_SUCCESS - Current drive get operation was successful. SYS_FS_RES_FAILURE - Current drive get operation was unsucessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function gets the present drive being used. The drive information is populated in the buffer.

Remarks

None.

Preconditions

The disk has to be mounted.

Example

```
SYS_FS_RESULT res;
char buffer[255];

res = SYS_FS_CurrentDriveGet(buffer);
if(res == SYS_FS_RES_FAILURE)
{
    // Operation failed.
}
```

Parameters

Parameters	Description
buffer	Pointer to buffer which will hold the name of present drive being used.

Function

SYS_FS_RESULT SYS_FS_CurrentDriveGet

(

char* buffer
);

SYS_FS_CurrentDriveSet Function

Sets the drive.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_CurrentDriveSet(const char* path);
```

Returns

SYS_FS_RES_SUCCESS - Current drive set operation was successful. SYS_FS_RES_FAILURE - Current drive set operation was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function sets the present drive to the one as specified by the path. By default, the drive mounted last becomes the current drive for the system. This is useful for applications where only one drive (volume) is used. In such an application, there is no need to call the SYS_FS_CurrentDriveSet function. However, in the case of an application where there are multiple volumes, the user can select the current drive for the application by calling this function.

Remarks

None.

Preconditions

The disk has to be mounted.

Example

```
SYS_FS_RESULT res;

res = SYS_FS_CurrentDriveSet("/mnt/myDrive");
if(res == SYS_FS_RES_FAILURE)
{
    // Drive change failed
}
```

Parameters

Parameters	Description
path	Path for the drive to be set.

Function

SYS_FS_RESULT SYS_FS_CurrentDriveSet
(
const char* path
);

SYS_FS_DriveLabelGet Function

Gets the drive label.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_DriveLabelGet(const char* drive, char * buff, uint32_t * sn);
```

Returns

SYS_FS_RES_SUCCESS - Drive label information retrieval was successful. SYS_FS_RES_FAILURE - Drive label information retrieval was unsucessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function gets the label for the drive specified. If no drive is specified, the label for the current drive is obtained.

Remarks

None.

Preconditions

At least one disk must be mounted.

Example

```
SYS_FS_RESULT res;
char buffer[255];
uint32_t serialNo;

switch(appState)
{
    case TRY_MOUNT:
        if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive", FAT, 0, NULL) != SYS_FS_RES_SUCCESS)
        {
            // Failure, try mounting again
        }
        else
        {
            // Mount was successful. Get label now
            appState = GET_LABEL;
        }
        break;

    case GET_LABEL:
        res = SYS_FS_DriveLabelGet("/mnt/myDrive", buffer, &serialNo);

        if(res == SYS_FS_RES_FAILURE)
        {
            // Fetching drive label information failed
        }
        //...
        //...
        break;
}
```

Parameters

Parameters	Description
drive	Pointer to buffer which will hold the name of drive being for which the label is requested. If this string is NULL, then then label of the current drive is obtained by using this function.
buff	Buffer which will hold the string of label.
sn	Serial number of the drive. If this information is not needed, it can be set as NULL.

Function

SYS_FS_RESULT SYS_FS_DriveLabelGet

(

const char* drive,

char *buff,

uint32_t *sn

);

SYS_FS_DriveLabelSet Function

Sets the drive label

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_DriveLabelSet(const char * drive, const char * label);
```

Returns

SYS_FS_RES_SUCCESS - Drive label set operation was successful. SYS_FS_RES_FAILURE - Drive label set operation was unsucessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function sets the label for the drive specified. If no drive is specified, the label for the current drive is set.

Remarks

None.

Preconditions

At least one disk must be mounted.

Example

```
SYS_FS_RESULT res;

switch(appState)
{
    case TRY_MOUNT:
        if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive", FAT, 0, NULL) != SYS_FS_RES_SUCCESS)
        {
            // Failure, try mounting again
        }
        else
        {
            // Mount was successful. Get label now
            appState = GET_LABEL;
        }
        break;

    case GET_LABEL:
        res = SYS_FS_DriveLabelSet("/mnt/myDrive", "MY_LABEL");
        if(res == SYS_FS_RES_FAILURE)
        {
            // Drive label get failed
        }
        //...
        //...
        break;
}
```

Parameters

Parameters	Description
drive	Pointer to string that holds the name of drive being for which the label is to be set. If this string is NULL, the label of the current drive is set by using this function.
label	Pointer to string which contains the label to be set.

Function

SYS_FS_RESULT SYS_FS_DriveLabelSet

(

const char* drive,

const char *label

);

SYS_FS_DriveFormat Function

Formats a drive.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_DriveFormat(const char* drive, SYS_FS_FORMAT fmt, uint32_t clusterSize);
```

Returns

SYS_FS_RES_SUCCESS - Drive format was successful. SYS_FS_RES_FAILURE - Drive format was unsucessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function formats a logic drive (create a FAT file system on the logical drive), as per the format specified.

If the logical drive that has to be formatted has been bound to any partition (1-4) by multiple partition feature, the FAT volume is created into the specified partition. In this case, the second argument fmt is ignored. The physical drive must have been partitioned prior to using this function.

Remarks

None.

Preconditions

At least one disk must be mounted. The physical drive must have already been partitioned.

Example

```
SYS_FS_RESULT res;

switch(appState)
{
    case TRY_MOUNT:
        if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive", FAT, 0, NULL) != SYS_FS_RES_SUCCESS)
        {
            // Failure, try mounting again
        }
        else
        {
            // Mount was successful. Format now.
            appState = FORMAT_DRIVE;
        }
        break;

    case FORMAT_DRIVE:
        res = SYS_FS_DriveFormat("/mnt/myDrive", SYS_FS_FORMAT_SFD, 0);
        if(res == SYS_FS_RES_FAILURE)
        {
            // Format of the drive failed.
        }
        //...
        break;
}
```

Parameters

Parameters	Description
drive	Pointer to buffer which will hold the name of drive being for which the format is to be done. If this string is NULL, then then current drive will be formatted. It is important to end the drive name with a "/".
fmt	Format type.
clusterSize	Cluster size. The value must be sector (size * n), where n is 1 to 128 and power of 2. When a zero is given, the cluster size depends on the volume size.

Function

```
SYS_FS_RESULT SYS_FS_DriveFormat
(
const char* drive,
    SYS_FS_FORMAT fmt,
uint32_t clusterSize
);
```

SYS_FS_DrivePartition Function

Partitions a physical drive (media).

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_DrivePartition(const char * path, const uint32_t partition[], void * work);
```

Returns

SYS_FS_RES_SUCCESS - Partition was successful. SYS_FS_RES_FAILURE - Partition was unsuccessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

This function partitions a physical drive (media) into requested partition sizes. This function will alter the MBR of the physical drive and make it into multi partitions. Windows operating systems do not support multi partitioned removable media. Maximum 4 partitions can be created on a media.

Remarks

None

Preconditions

Prior to partitioning the media, the media should have a valid MBR and it should be mounted as a volume with the file system.

Example

```
//==============================================================================
// Initially, consider the case of a SD card that has only one partition.
//==============================================================================
SYS_FS_RESULT res;

// Following 4 element array specifies the size of 2 partitions as
// 256MB (=524288 sectors). The 3rd and 4th partition are not created
// since, the sizes of those are zero.
uint32_t plist[] = {524288, 524288, 0, 0};

// Work area for function SYS_FS_DrivePartition
char work[FAT_FS_MAX_SS];

switch(appState)
{
    case TRY_MOUNT:
        if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive", FAT, 0, NULL) != SYS_FS_RES_SUCCESS)
        {
            // Failure, try mounting again
        }
        else
        {
            // Mount was successful. Partition now.
            appState = PARTITION_DRIVE;
        }
        break;

    case PARTITION_DRIVE:
        res = SYS_FS_DrivePartition("/mnt/myDrive", plist, work);
        if(res == SYS_FS_RES_FAILURE)
        {
            // Drive partition went wrong
        }
        else
        {
            // Partition was successful. Power cycle the board so that
            // all partitions are recognized. Then try mounting both
            // partitions.
        }
        break;
```

```
        default:
            break;
    }

    //=======================================================================
    //The following code is after the SD card is partitioned and then
    //powered ON.
    //=======================================================================
    SYS_FS_RESULT res;

    switch(appState)
    {
        case TRY_MOUNT_1ST_PARTITION:
            if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive1", FAT, 0, NULL) != SYS_FS_RES_SUCCESS)
            {
                // Failure, try mounting again
                appState = TRY_MOUNT_1ST_PARTITION;
            }
            else
            {
                // Mount was successful. Mount second partition.
                appState = TRY_MOUNT_2ND_PARTITION;
            }
            break;

        case TRY_MOUNT_2ND_PARTITION:
            if(SYS_FS_Mount("/dev/mmcblka2", "/mnt/myDrive2", FAT, 0, NULL) != SYS_FS_RES_SUCCESS)
            {
                // Failure, try mounting again
                appState = TRY_MOUNT_2ND_PARTITION;
            }
            else
            {
                // Mount was successful. Try formating first partition.
                appState = TRY_FORMATING_1ST_PARTITION;
            }
            break;

        case TRY_FORMATING_1ST_PARTITION:
            if(SYS_FS_DriveFormat("/mnt/myDrive1/", SYS_FS_FORMAT_FDISK, 0) == SYS_FS_RES_FAILURE)
            {
                // Failure
            }
            else
            {
                // Try formating second partitions.
                appState = TRY_FORMATING_2ND_PARTITION;
            }

        case TRY_FORMATING_2ND_PARTITION:
            if(SYS_FS_DriveFormat("/mnt/myDrive2/", SYS_FS_FORMAT_FDISK, 0) == SYS_FS_RES_FAILURE)
            {
                // Failure
            }
            else
            {
                // Use both partitions as 2 separate volumes.
            }

        default:
            break;
    }
```

Parameters

Parameters	Description
path	Path to the volume with the volume name. The string of volume name has to be preceded by "/mnt/". Also, the volume name and directory name has to be separated by a slash "/".

partition	Array with 4 items, where each items mentions the sizes of each partition in terms of number of sector. 0th element of array specifies the number of sectors for first partition and 3rd element of array specifies the number of sectors for fourth partition.
work	Pointer to the buffer for function work area. The size must be at least FAT_FS_MAX_SS bytes.

Function

SYS_FS_RESULT SYS_FS_DrivePartition
(
const char *path,
const uint32_t partition[],
void * work
);

SYS_FS_DriveSectorGet Function

Obtains total number of sectors and number of free sectors for the specified drive.

File

sys_fs.h

C

```
SYS_FS_RESULT SYS_FS_DriveSectorGet(const char * path, uint32_t * totalSectors, uint32_t * freeSectors);
```

Returns

SYS_FS_RES_SUCCESS - Sector information get operation was successful. SYS_FS_RES_FAILURE - Sector information get operation was unsucessful. The reason for the failure can be retrieved with SYS_FS_Error.

Description

Function to obtain the total number of sectors and number of free sectors in a drive (media).

Remarks

None.

Preconditions

The drive for which the information is to be retrieved should be mounted.

Example

```
uint32_t totalSectors, freeSectors;
SYS_FS_RESULT res;

if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive", FAT, 0, NULL) != SYS_FS_RES_SUCCESS)
{
    // Failure, try mounting again
}
else
{
    // Mount was successful.
    // Do other FS stuffs.
}
// Perform usual FS tasks.
//....
//....

// Now, determine the total sectors and free sectors
res = SYS_FS_DriveSectorGet("/mnt/myDrive", &totalSectors, &freeSectors);
if(res == SYS_FS_RES_FAILURE)
{
    //Sector information get operation failed.
}
```

Parameters

Parameters	Description
path	Path to the volume with the volume name. The string of volume name must be preceded by "/mnt/". Also, the volume name and directory name must be separated by a slash "/".
totalSectors	Pointer to a variable passed to the function, which will contain the total number of sectors available in the drive (media).
freeSectors	Pointer to a variable passed to the function, which will contain the free number of sectors available in the drive (media).

Function

SYS_FS_RESULT SYS_FS_DriveSectorGet
(
const char* path,
uint32_t *totalSectors,
uint32_t *freeSectors
);

SYS_FS_EventHandlerSet Function

Allows a client to identify an event handling function for the file system to call back when mount/unmount operation has completed.

File

sys_fs.h

C

```
void SYS_FS_EventHandlerSet(const void * eventHandler, const uintptr_t context);
```

Returns

None.

Description

This function allows a client to identify an event handling function for the File System to call back when mount/unmount operation has completed. The file system will pass mount name back to the client by calling "eventHandler".

Remarks

If the client does not want to be notified when the mount/unmount operation has completed, it does not need to register a callback.

Preconditions

The SYS_FS_Initialize() routine must have been called.

Example

```
// Client registers an event handler with file system. This is done once.
SYS_FS_EventHandlerSet(APP_SysFSEventHandler, (uintptr_t)NULL);

// Event Processing Technique. Event is received when operation is done.
void APP_SysFSEventHandler
(
    SYS_FS_EVENT event,
    void* eventData,
    uintptr_t context
)
{
    switch(event)
    {
        case SYS_FS_EVENT_MOUNT:
            if(0 == strcmp((const char *)mountName,"/mnt/myDrive1"))
            {
                gSDCardMountFlag = true;
            }
            else if(0 == strcmp((const char *)mountName,"/mnt/myDrive2"))
            {
                gNVMMountFlag = true;
```

```
                }
                break;

            case SYS_FS_EVENT_UNMOUNT:
                if(0 == strcmp((const char *)mountName,"/mnt/myDrive1"))
                {
                    gSDCardMountFlag = false;
                }
                else if(0 == strcmp((const char *)mountName,"/mnt/myDrive2"))
                {
                    gNVMMountFlag = false;
                }
                appData.state = APP_ERROR;
                break;

            case SYS_FS_EVENT_ERROR:
                break;
        }
    }
```

Parameters

Parameters	Description
eventHandler	Pointer to the event handler function implemented by the user
context	The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

Function

void SYS_FS_EventHandlerSet

(

const void * eventHandler,

const uintptr_t context

);

c) Data Types and Constants

SYS_FS_ERROR Enumeration

Lists the various error cases.

File

sys_fs.h

C

```
typedef enum {
  SYS_FS_ERROR_OK = 0,
  SYS_FS_ERROR_DISK_ERR,
  SYS_FS_ERROR_INT_ERR,
  SYS_FS_ERROR_NOT_READY,
  SYS_FS_ERROR_NO_FILE,
  SYS_FS_ERROR_NO_PATH,
  SYS_FS_ERROR_INVALID_NAME,
  SYS_FS_ERROR_DENIED,
  SYS_FS_ERROR_EXIST,
  SYS_FS_ERROR_INVALID_OBJECT,
  SYS_FS_ERROR_WRITE_PROTECTED,
  SYS_FS_ERROR_INVALID_DRIVE,
  SYS_FS_ERROR_NOT_ENABLED,
  SYS_FS_ERROR_NO_FILESYSTEM,
  SYS_FS_ERROR_FORMAT_ABORTED,
  SYS_FS_ERROR_TIMEOUT,
  SYS_FS_ERROR_LOCKED,
  SYS_FS_ERROR_NOT_ENOUGH_CORE,
  SYS_FS_ERROR_TOO_MANY_OPEN_FILES,
```

```
  SYS_FS_ERROR_INVALID_PARAMETER,
  SYS_FS_ERROR_NOT_ENOUGH_FREE_VOLUME,
  SYS_FS_ERROR_FS_NOT_SUPPORTED,
  SYS_FS_ERROR_FS_NOT_MATCH_WITH_VOLUME,
  SYS_FS_ERROR_NOT_SUPPORTED_IN_NATIVE_FS
} SYS_FS_ERROR;
```

Members

Members	Description
SYS_FS_ERROR_OK = 0	Success
SYS_FS_ERROR_DISK_ERR	(1) A hard error occurred in the low level disk I/O layer
SYS_FS_ERROR_INT_ERR	(2) Assertion failed
SYS_FS_ERROR_NOT_READY	(3) The physical drive cannot work
SYS_FS_ERROR_NO_FILE	(4) Could not find the file
SYS_FS_ERROR_NO_PATH	(5) Could not find the path
SYS_FS_ERROR_INVALID_NAME	(6) The path name format is invalid
SYS_FS_ERROR_DENIED	(7) Access denied due to prohibited access or directory full
SYS_FS_ERROR_EXIST	(8) Access denied due to prohibited access
SYS_FS_ERROR_INVALID_OBJECT	(9) The file/directory object is invalid
SYS_FS_ERROR_WRITE_PROTECTED	(10) The physical drive is write protected
SYS_FS_ERROR_INVALID_DRIVE	(11) The logical drive number is invalid
SYS_FS_ERROR_NOT_ENABLED	(12) The volume has no work area
SYS_FS_ERROR_NO_FILESYSTEM	(13) There is no valid volume
SYS_FS_ERROR_FORMAT_ABORTED	(14) The Format() aborted due to any parameter error
SYS_FS_ERROR_TIMEOUT	(15) Could not get a grant to access the volume within defined period
SYS_FS_ERROR_LOCKED	(16) The operation is rejected according to the file sharing policy
SYS_FS_ERROR_NOT_ENOUGH_CORE	(17) LFN working buffer could not be allocated
SYS_FS_ERROR_TOO_MANY_OPEN_FILES	(18) Number of open files
SYS_FS_ERROR_INVALID_PARAMETER	(19) Given parameter is invalid
SYS_FS_ERROR_NOT_ENOUGH_FREE_VOLUME	(20) Too many mounts requested. Not enough free volume available
SYS_FS_ERROR_FS_NOT_SUPPORTED	(21) Requested native file system is not supported
SYS_FS_ERROR_FS_NOT_MATCH_WITH_VOLUME	(22) Requested native file system does not match the format of volume
SYS_FS_ERROR_NOT_SUPPORTED_IN_NATIVE_FS	(23) Function not supported in native file system layer

Description

File Error enumeration

This enumeration lists the various error cases. When the application calls for a file system function which has a return type of SYS_FS_RESULT and if the return value is SYS_FS_RES_FAILURE, the application can know the specific reason for failure by calling the SYS_FS_FileError function. The return value of SYS_FS_FileError function will be one of the enumeration of type SYS_FS_ERROR.

Remarks

None.

SYS_FS_FILE_OPEN_ATTRIBUTES Enumeration

Lists the various attributes (modes) in which a file can be opened.

File

sys_fs.h

C

```
typedef enum {
  SYS_FS_FILE_OPEN_READ = 0,
  SYS_FS_FILE_OPEN_WRITE,
  SYS_FS_FILE_OPEN_APPEND,
  SYS_FS_FILE_OPEN_READ_PLUS,
  SYS_FS_FILE_OPEN_WRITE_PLUS,
  SYS_FS_FILE_OPEN_APPEND_PLUS
} SYS_FS_FILE_OPEN_ATTRIBUTES;
```

Members

Members	Description
SYS_FS_FILE_OPEN_READ = 0	reading the file = possible, if file exists. reading the file = file open returns error, if file does not exist. writing to the file = not possible. Write operation returns error
SYS_FS_FILE_OPEN_WRITE	reading the file = not possible. Read operation returns error. writing to the file = possible. If file exists, write happens from the beginning of the file, overwriting the existing content of the file. writing to the file = If file does not exist, a new file will be created and data will be written into the newly created file.
SYS_FS_FILE_OPEN_APPEND	reading the file = not possible. Read operation returns error writing to the file = possible. If file exists, write happens from the end of the file, preserving the existing content of the file. writing to the file = If file does not exist, a new file will be created and data will be written into the newly created file.
SYS_FS_FILE_OPEN_READ_PLUS	reading the file = possible, if file exists. reading the file = file open returns error, if file does not exist. writing to the file = possible, if file exists, staring from the beginning of the file (overwriting). writing to the file = file open returns error, if file does not exist.
SYS_FS_FILE_OPEN_WRITE_PLUS	reading the file = possible, if file exists. reading the file = If file does not exist, a new file will be created. writing to the file = possible. If file exists, write happens from the beginning of the file, overwriting the existing content of the file. writing to the file = If file does not exist, a new file will be created and data will be written into the newly created file.
SYS_FS_FILE_OPEN_APPEND_PLUS	reading the file = possible, if file exists. File read pointer will be moved to end of the file in this mode. reading the file = If file does not exist, a new file will be created. writing to the file = possible. If file exists, write happens from the end of the file, preserving the existing content of the file. writing to the file = If file does not exist, a new file will be created and data will be written into the newly created file.

Description

File open attributes

This enumeration lists the various attributes (modes) in which a file can be opened.

Remarks

None.

SYS_FS_FILE_SEEK_CONTROL Enumeration

Lists the various modes of file seek.

File

sys_fs.h

C

```
typedef enum {
  SYS_FS_SEEK_SET,
  SYS_FS_SEEK_CUR,
  SYS_FS_SEEK_END
} SYS_FS_FILE_SEEK_CONTROL;
```

Members

Members	Description
SYS_FS_SEEK_SET	Set file offset to input number of bytes from the start of file
SYS_FS_SEEK_CUR	Set file offset to its current location plus input number of bytes
SYS_FS_SEEK_END	Set file offset to size of the file plus input number of bytes

Description

File Seek control

This enumeration lists the various modes of file seek. When the application calls the SYS_FS_FileSeek function, it specifies the kind of seek that needs to be performed.

Remarks

None.

SYS_FS_FILE_SYSTEM_TYPE Enumeration

Enumerated data type identifying native file systems supported.

File

sys_fs.h

C

```
typedef enum {
  UNSUPPORTED_FS = 0,
  FAT,
  MPFS2
} SYS_FS_FILE_SYSTEM_TYPE;
```

Members

Members	Description
UNSUPPORTED_FS = 0	Unsupported File System
FAT	FAT FS native File system
MPFS2	MPFS2 native File system

Description

File System type

These enumerated values identify the native file system supported by the SYS FS.

Remarks

None.

SYS_FS_FSTAT Structure

File System status

File

sys_fs.h

C

```
typedef struct {
  uint32_t fsize;
  uint16_t fdate;
  uint16_t ftime;
  uint8_t fattrib;
  char fname[13];
  char * lfname;
  uint32_t lfsize;
} SYS_FS_FSTAT;
```

Members

Members	Description
uint32_t fsize;	File size
uint16_t fdate;	Last modified date
uint16_t ftime;	Last modified time
uint8_t fattrib;	Attribute
char fname[13];	Short file name (8.3 format)
char * lfname;	Pointer to the LFN buffer
uint32_t lfsize;	Size of LFN buffer in TCHAR

Description

SYS FS File status structure

This structure holds the various status of a file. The SYS_FS_FileStat () populates the contents of this structure.

Remarks

None.

SYS_FS_FUNCTIONS Structure

SYS FS Function signature structure for native file systems.

File

sys_fs.h

C

```
typedef struct {
  int (* mount)(uint8_t vol);
  int (* unmount)(uint8_t vol);
  int (* open)(uintptr_t handle, const char* path, uint8_t mode);
  int (* read)(uintptr_t fp, void* buff, uint32_t btr, uint32_t *br);
  int (* write)(uintptr_t fp, const void* buff, uint32_t btw, uint32_t* bw);
  int (* close)(uintptr_t fp);
  int (* seek)(uintptr_t handle, uint32_t offset);
  uint32_t (* tell)(uintptr_t handle);
  bool (* eof)(uintptr_t handle);
  uint32_t (* size)(uintptr_t handle);
  int (* fstat)(const char* path, uintptr_t fno);
  int (* mkdir)(const char *path);
  int (* chdir)(const char *path);
  int (* remove)(const char *path);
  int (* getlabel)(const char *path, char *buff, uint32_t *sn);
  int (* setlabel)(const char *label);
  int (* truncate)(uintptr_t handle);
  int (* currWD)(char* buff, uint32_t len);
  int (* chdrive)(uint8_t drive);
  int (* chmode)(const char* path, uint8_t attr, uint8_t mask);
  int (* chtime)(const char* path, uintptr_t ptr);
  int (* rename)(const char *oldPath, const char *newPath);
  int (* sync)(uintptr_t fp);
  char * (* getstrn)(char* buff, int len, uintptr_t handle);
  int (* putchr)(char c, uintptr_t handle);
  int (* putstrn)(const char* str, uintptr_t handle);
  int (* formattedprint)(uintptr_t handle, const char *str, ...);
  bool (* testerror)(uintptr_t handle);
  int (* formatDisk)(uint8_t vol, uint8_t sfd, uint32_t au);
  int (* openDir)(uintptr_t handle, const char *path);
  int (* readDir)(uintptr_t handle, uintptr_t stat);
  int (* closeDir)(uintptr_t handle);
  int (* partitionDisk)(uint8_t pdrv, const uint32_t szt[], void* work);
  int (* getCluster)(const char *path, uint32_t *tot_sec, uint32_t *free_sec);
} SYS_FS_FUNCTIONS;
```

Members

Members	Description
int (* mount)(uint8_t vol);	Function pointer of native file system for mounting a volume
int (* unmount)(uint8_t vol);	Function pointer of native file system for unmounting a volume
int (* open)(uintptr_t handle, const char* path, uint8_t mode);	Function pointer of native file system for opening a file
int (* read)(uintptr_t fp, void* buff, uint32_t btr, uint32_t *br);	Function pointer of native file system for reading a file
int (* write)(uintptr_t fp, const void* buff, uint32_t btw, uint32_t* bw);	Function pointer of native file system for writing to a file
int (* close)(uintptr_t fp);	Function pointer of native file system for closing a file
int (* seek)(uintptr_t handle, uint32_t offset);	Function pointer of native file system for moving the file pointer by a • desired offset
uint32_t (* tell)(uintptr_t handle);	Function pointer of native file system for finding the position of the • file pointer
bool (* eof)(uintptr_t handle);	Function pointer of native file system to check if the end of file is • reached
uint32_t (* size)(uintptr_t handle);	Function pointer of native file system to know the size of file
int (* fstat)(const char* path, uintptr_t fno);	Function pointer of native file system to know the status of file
int (* mkdir)(const char *path);	Function pointer of native file system to create a directory
int (* chdir)(const char *path);	Function pointer of native file system to change a directory
int (* remove)(const char *path);	Function pointer of native file system to remove a file or directory
int (* getlabel)(const char *path, char *buff, uint32_t *sn);	Function pointer of native file system to get the volume label

int (* setlabel)(const char *label);	Function pointer of native file system to set the volume label
int (* truncate)(uintptr_t handle);	Function pointer of native file system to truncate the file
int (* currWD)(char* buff, uint32_t len);	Function pointer of native file system to obtain the current working • directory
int (* chdrive)(uint8_t drive);	Function pointer of native file system to set the current drive
int (* chmode)(const char* path, uint8_t attr, uint8_t mask);	Function pointer of native file system to change the attribute for file • or directory
int (* chtime)(const char* path, uintptr_t ptr);	Function pointer of native file system to change the time for a file or • directory
int (* rename)(const char *oldPath, const char *newPath);	Function pointer of native file system to rename a file or directory
int (* sync)(uintptr_t fp);	Function pointer of native file system to flush file
char * (* getstrn)(char* buff, int len, uintptr_t handle);	Function pointer of native file system to read a string from a file
int (* putchr)(char c, uintptr_t handle);	Function pointer of native file system to write a character into a file
int (* putstrn)(const char* str, uintptr_t handle);	Function pointer of native file system to write a string into a file
int (* formattedprint)(uintptr_t handle, const char *str, ...);	Function pointer of native file system to print a formatted string to • file
bool (* testerror)(uintptr_t handle);	Function pointer of native file system to test an error in a file
int (* formatDisk)(uint8_t vol, uint8_t sfd, uint32_t au);	Function pointer of native file system to format a disk
int (* openDir)(uintptr_t handle, const char *path);	Function pointer of native file system to open a directory
int (* readDir)(uintptr_t handle, uintptr_t stat);	Function pointer of native file system to read a directory
int (* closeDir)(uintptr_t handle);	Function pointer of native file system to close an opened directory
int (* partitionDisk)(uint8_t pdrv, const uint32_t szt[], void* work);	Function pointer of native file system to partition a physical drive
int (* getCluster)(const char *path, uint32_t *tot_sec, uint32_t *free_sec);	Function pointer of native file system to get total sectors and free • sectors

Description

SYS FS Function signature structure for native file systems

The SYS FS layer supports functions from each native file system layer. This structure specifies the signature for each function from native file system (parameter that needs to be passed to each function and return type for each function). If a new native file system is to be integrated with the SYS FS layer, the functions should follow the signature.

The structure of function pointer for the two native file systems: FAT FS and MPFS2 is already provided in the respective source files for the native file system. Hence the following structure is not immediately useful for the user. But the explanation for the structure is still provided for advanced users who would wish to integrate a new native file system to the MPLAB Harmony File System framework.

Remarks

None.

SYS_FS_HANDLE Type

This type defines the file handle.

File

sys_fs.h

C

```
typedef uintptr_t SYS_FS_HANDLE;
```

Description

SYS FS File Handle

This type defines the file handle. File handle is returned by the File Open function on successful operation.

Remarks

None.

SYS_FS_REGISTRATION_TABLE Structure

The sys_fs layer has to be initialized by passing this structure with suitably initialized members.

File

sys_fs.h

C

```
typedef struct {
  SYS_FS_FILE_SYSTEM_TYPE nativeFileSystemType;
  const SYS_FS_FUNCTIONS * nativeFileSystemFunctions;
} SYS_FS_REGISTRATION_TABLE;
```

Members

Members	Description
SYS_FS_FILE_SYSTEM_TYPE nativeFileSystemType;	Native file system of type SYS_FS_FILE_SYSTEM_TYPE
const SYS_FS_FUNCTIONS * nativeFileSystemFunctions;	Pointer to the structure of type SYS_FS_FUNCTIONS which has the list of

Description

SYS_FS_REGISTRATION_TABLE structure

When the SYS FS layer is initialized, it has to know the type of native file system it has to support and the list of functions for native file system. The members of this structure can be initialized with suitable values and then passed on to SYS_FS_Initialize initialization function. Please refer to the example code provided for SYS_FS_Initialize.

Remarks

None.

SYS_FS_RESULT Enumeration

Lists the various results of a file operation.

File

sys_fs.h

C

```
typedef enum {
  SYS_FS_RES_SUCCESS = 0,
  SYS_FS_RES_FAILURE = -1
} SYS_FS_RESULT;
```

Members

Members	Description
SYS_FS_RES_SUCCESS = 0	Operation succeeded
SYS_FS_RES_FAILURE = -1	Operation failed

Description

File operation result enum

This enumeration lists the various results of a file operation. When a file operation function is called from the application, and if the return type of the function is SYS_FS_RESULT, then the enumeration below specifies the possible values returned by the function.

Remarks

None.

FAT_FS_MAX_LFN Macro

Maximum length of the Long File Name.

File

sys_fs.h

C

```
#define FAT_FS_MAX_LFN 255
```

Description

FAT File System LFN (Long File Name) max length

Defines the maximum length of file name during LFN selection. Set the value to 255.

Remarks

None.

FAT_FS_MAX_SS Macro

Lists the definitions for FAT file system sector size.

File

sys_fs.h

C

```
#define FAT_FS_MAX_SS 512
```

Description

FAT File System Sector size

Maximum sector size to be handled. Always set the value of sector size to 512

Remarks

None.

FAT_FS_USE_LFN Macro

Lists the definitions for FAT file system LFN selection.

File

sys_fs.h

C

```
#define FAT_FS_USE_LFN 1
```

Description

FAT File System LFN (long file name) selection

The FAT_FS_USE_LFN option switches the LFN support. Set the value to 1.

Remarks

None.

SYS_FS_HANDLE_INVALID Macro

Invalid file handle

File

sys_fs.h

C

```
#define SYS_FS_HANDLE_INVALID ((SYS_FS_HANDLE)(-1))
```

Description

SYS FS File Invalid Handle

This value defines the invalid file handle. Invalid file handle is returned on an unsucessful File Open operation.

Remarks

None.

SYS_FS_FILE_DIR_ATTR Enumeration

Enumerated data type identifying the various attributes for file/directory.

File

sys_fs.h

C

```
typedef enum {
  SYS_FS_ATTR_RDO = 0x01,
  SYS_FS_ATTR_HID = 0x02,
  SYS_FS_ATTR_SYS = 0x04,
  SYS_FS_ATTR_VOL = 0x08,
  SYS_FS_ATTR_LFN = 0x0F,
  SYS_FS_ATTR_DIR = 0x10,
  SYS_FS_ATTR_ARC = 0x20,
  SYS_FS_ATTR_MASK = 0x3F
} SYS_FS_FILE_DIR_ATTR;
```

Members

Members	Description
SYS_FS_ATTR_RDO = 0x01	Read only
SYS_FS_ATTR_HID = 0x02	Hidden
SYS_FS_ATTR_SYS = 0x04	System
SYS_FS_ATTR_VOL = 0x08	Volume label
SYS_FS_ATTR_LFN = 0x0F	LFN entry
SYS_FS_ATTR_DIR = 0x10	Directory
SYS_FS_ATTR_ARC = 0x20	Archive
SYS_FS_ATTR_MASK = 0x3F	Mask of defined bits

Description

File or directory attribute

These enumerated values are the possible attributes for a file or directory.

Remarks

None.

SYS_FS_TIME Union

The structure to specify the time for a file or directory.

File

sys_fs.h

C

```
typedef union {
  struct discreteTime {
    unsigned second : 5;
    unsigned minute : 6;
    unsigned hour : 5;
    unsigned day : 5;
    unsigned month : 4;
    unsigned year : 7;
  }
  struct timeDate {
    uint16_t time;
    uint16_t date;
  }
  uint32_t packedTime;
} SYS_FS_TIME;
```

Members

Members	Description
unsigned second : 5;	Second / 2 (0..29)
unsigned minute : 6;	Minute (0..59)
unsigned hour : 5;	Hour (0..23)
unsigned day : 5;	Day in month(1..31)
unsigned month : 4;	Month (1..12)
unsigned year : 7;	Year from 1980 (0..127)
uint16_t time;	Time (hour, min, seconds)
uint16_t date;	Date (year, month, day)
uint32_t packedTime;	Combined time information in a 32-bit value

Description

SYS FS File time structure

This structure holds the date and time to be used to set for a file or directory.

bits 31-25: Year from 1980 (0..127) bits 24-21: Month (1..12) bits 20-16: Day in month(1..31) bits 15-11: Hour (0..23) bits 10-5 : Minute (0..59) bits 4-0 : Seconds / 2 (0..29)

Remarks

None.

SYS_FS_FORMAT Enumeration

Specifes the partitioning rule.

File

sys_fs.h

C

```
typedef enum {
  SYS_FS_FORMAT_FDISK = 0,
  SYS_FS_FORMAT_SFD = 1
} SYS_FS_FORMAT;
```

Members

Members	Description
SYS_FS_FORMAT_FDISK = 0	Format disk with multiple partition
SYS_FS_FORMAT_SFD = 1	Format disk with single partition

Description

File formating partition rule

This type specifies the partitioning rule. When SYS_FS_FORMAT_FDISK format is specified, a primary partition occupying the entire disk space is created and then an FAT volume is created on the partition. When SYS_FS_FORMAT_SFD format is specified, the FAT volume starts from the first sector of the physical drive.

The SYS_FS_FORMAT_FDISK partitioning is usually used for hard disk, MMC, SDC, CFC and U Disk. It can divide a physical drive into one or more partitions with a partition table on the MBR. However Windows does not support multiple partition on the removable media. The SYS_FS_FORMAT_SFD is non-partitioned method. The FAT volume starts from the first sector on the physical drive without partition table. It is usually used for floppy disk, micro drive, optical disk, and super-floppy media.

SYS_FS_EVENT Enumeration

Identifies the possible file system events.

File

sys_fs.h

C

```
typedef enum {
  SYS_FS_EVENT_MOUNT,
  SYS_FS_EVENT_UNMOUNT,
```

```
    SYS_FS_EVENT_ERROR
} SYS_FS_EVENT;
```

Members

Members	Description
SYS_FS_EVENT_MOUNT	Media has been mounted successfully.
SYS_FS_EVENT_UNMOUNT	Media has been unmounted successfully.
SYS_FS_EVENT_ERROR	There was an error during the operation

Description

SYS FS Media Events

This enumeration identifies the possible events that can result from a file system.

Remarks

One of these values is passed in the "event" parameter of the event handling callback function that client registered with the file system by setting the event handler when media mount or unmount is completed.

SYS_FS_EVENT_HANDLER Type

Pointer to the File system Handler function.

File

sys_fs.h

C

```
typedef void (* SYS_FS_EVENT_HANDLER)(SYS_FS_EVENT event, void* eventData, uintptr_t context);
```

Returns

None.

Description

File System Event Handler function pointer

This data type defines the required function signature for the file system event handling callback function. A client must register a pointer to an event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event call backs from the file system.

Remarks

None.

Parameters

Parameters	Description
event	Identifies the type of event
eventData	Handle returned from the media operation requests
context	Value identifying the context of the application that registered the event handling function

Files

Files

Name	Description
sys_fs.h	Functions and type declarations required to interact with the MPLAB Harmony File System Service.
sys_fs_config_template.h	File System Service configuration templates.

Description

This section lists the source and header files used by the library.

sys_fs.h

Functions and type declarations required to interact with the MPLAB Harmony File System Service.

Enumerations

	Name	Description
	SYS_FS_ERROR	Lists the various error cases.
	SYS_FS_EVENT	Identifies the possible file system events.
	SYS_FS_FILE_DIR_ATTR	Enumerated data type identifying the various attributes for file/directory.
	SYS_FS_FILE_OPEN_ATTRIBUTES	Lists the various attributes (modes) in which a file can be opened.
	SYS_FS_FILE_SEEK_CONTROL	Lists the various modes of file seek.
	SYS_FS_FILE_SYSTEM_TYPE	Enumerated data type identifying native file systems supported.
	SYS_FS_FORMAT	Specifes the partitioning rule.
	SYS_FS_RESULT	Lists the various results of a file operation.

Functions

	Name	Description
	SYS_FS_CurrentDriveGet	Gets the current drive
	SYS_FS_CurrentDriveSet	Sets the drive.
	SYS_FS_CurrentWorkingDirectoryGet	Gets the current working directory
	SYS_FS_DirClose	Closes an opened directory.
	SYS_FS_DirectoryChange	Changes to a the directory specified.
	SYS_FS_DirectoryMake	Makes a directory.
	SYS_FS_DirOpen	Open a directory
	SYS_FS_DirRead	Reads the specified directory.
	SYS_FS_DirRewind	Rewinds to the beginning of the directory.
	SYS_FS_DirSearch	Searches for a file or directory.
	SYS_FS_DriveFormat	Formats a drive.
	SYS_FS_DriveLabelGet	Gets the drive label.
	SYS_FS_DriveLabelSet	Sets the drive label
	SYS_FS_DrivePartition	Partitions a physical drive (media).
	SYS_FS_DriveSectorGet	Obtains total number of sectors and number of free sectors for the specified drive.
	SYS_FS_Error	Returns the last error.
	SYS_FS_EventHandlerSet	Allows a client to identify an event handling function for the file system to call back when mount/unmount operation has completed.
	SYS_FS_FileCharacterPut	Writes a character to a file.
	SYS_FS_FileClose	Closes a file.
	SYS_FS_FileDirectoryModeSet	Sets the mode for the file or directory.
	SYS_FS_FileDirectoryRemove	Removes a file or directory.
	SYS_FS_FileDirectoryRenameMove	Renames or moves a file or directory.
	SYS_FS_FileDirectoryTimeSet	Sets or changes the time for a file or directory.
	SYS_FS_FileEOF	Checks for end of file.
	SYS_FS_FileError	Returns the file specific error.
	SYS_FS_FileNameGet	Reads the file name.
	SYS_FS_FileOpen	Opens a file.
	SYS_FS_FilePrintf	Writes a formatted string into a file.
	SYS_FS_FileRead	Read data from the file.
	SYS_FS_FileSeek	Moves the file pointer by the requested offset.
	SYS_FS_FileSize	Returns the size of the file in bytes.
	SYS_FS_FileStat	Gets file status.
	SYS_FS_FileStringGet	Reads a string from the file into a buffer.
	SYS_FS_FileStringPut	Writes a string to a file.
	SYS_FS_FileSync	Flushes the cached information when writing to a file.
	SYS_FS_FileTell	Obtains the file pointer position.
	SYS_FS_FileTestError	Checks for errors in the file.
	SYS_FS_FileTruncate	Truncates a file
	SYS_FS_FileWrite	Writes data to the file.
	SYS_FS_Initialize	Initializes the file system abstraction layer (sys_fs layer).
	SYS_FS_Mount	Mounts the file system.

	SYS_FS_Tasks	Maintains the File System tasks and functionalities.
	SYS_FS_Unmount	Unmounts the file system.

Macros

	Name	Description
	FAT_FS_MAX_LFN	Maximum length of the Long File Name.
	FAT_FS_MAX_SS	Lists the definitions for FAT file system sector size.
	FAT_FS_USE_LFN	Lists the definitions for FAT file system LFN selection.
	SYS_FS_HANDLE_INVALID	Invalid file handle

Structures

	Name	Description
	SYS_FS_FSTAT	File System status
	SYS_FS_FUNCTIONS	SYS FS Function signature structure for native file systems.
	SYS_FS_REGISTRATION_TABLE	The sys_fs layer has to be initialized by passing this structure with suitably initialized members.

Types

	Name	Description
	SYS_FS_EVENT_HANDLER	Pointer to the File system Handler function.
	SYS_FS_HANDLE	This type defines the file handle.

Unions

	Name	Description
	SYS_FS_TIME	The structure to specify the time for a file or directory.

Description

File System Service Library Interface Declarations and Types

This file contains function and type declarations required to interact with the MPLAB Harmony File System Service.

File Name

sys_fs.h

Company

Microchip Technology Inc.

sys_fs_config_template.h

File System Service configuration templates.

Macros

	Name	Description
	SYS_FS_AUTOMOUNT_ENABLE	Enable/Disable Auto Mount Feature of File system
	SYS_FS_MAX_FILE_SYSTEM_TYPE	Number of file systems used in the application
	SYS_FS_MAX_FILES	Number of simultaneous files access
	SYS_FS_MEDIA_MAX_BLOCK_SIZE	Media Sector Size information
	SYS_FS_MEDIA_NUMBER	Number of media used in the application
	SYS_FS_VOLUME_NUMBER	This macro defines number of volumes used in the application

Description

File System Service Configuration Templates

This file contains constants to configure the File System Service.

File Name

sys_fs_config_template.h

Company

Microchip Technology Inc.

Interrupt System Service Library

This section describes the Interrupt System Service Library.

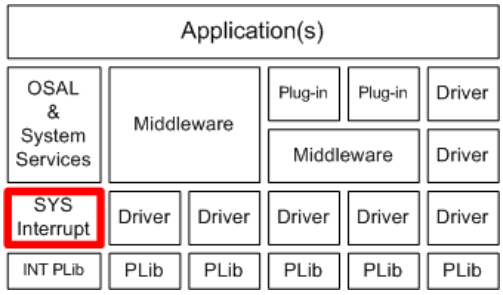
Introduction

Interrupt System Service for Microchip Microcontrollers

This library provides a low-level abstraction of the Interrupt System Service Library that is available on the Microchip family of PIC32 microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, thereby hiding differences from one microcontroller variant to another.

Description

This library implements the Interrupt System Service. It is part of the system services that provides support for processing interrupts. The Interrupt System provides support for initializing the processor's interrupt controller, registering Interrupt Service Routines (ISRs) and managing interrupts. These features enable making efficient and dynamic applications, drivers, and middleware that respond to external events as they occur in real time.



Using the Library

This topic describes the basic architecture of the Interrupt System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_int.h`

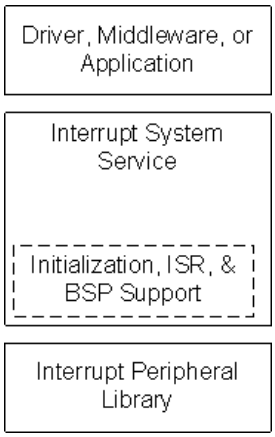
The interface to the Interrupt System Service library is defined in the `sys_int.h` header file, which is included by the `sys.h` system service header file. Any C language source (`.c`) file that uses the Interrupt System Service library should include `sys.h`.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Abstraction Model

This library provides an abstraction of the interrupt subsystem that is used by device drivers, middleware libraries and applications to receive and control interrupts in real time.

Description



Interrupt System Service

The interrupt system services provide support for initializing the processor's interrupt controller, managing Interrupt Service Routines (ISRs) and

managing interrupts.

Initialization

Each software module (device driver, library, or application) that needs to receive an interrupt must enable that interrupt itself. This is normally done in the module's initialization routine which is called by the SYS_Initialize service.

A module that intends to use an interrupt must first register the Tasks function that is to be called when the desired source causes an interrupt. Then, it must enable that source, once it is ready to start receiving interrupts.

If the interrupt system service is configured for static usage, the routine that dynamically registers the Tasks function will be nulled out by a macro (generating no run-time code) and, instead, the Tasks routine must be called statically from the function that implements the raw ISR vector. How this is done is different for each processor family, as explained in the following section.

Interrupt Service Routine (ISR)

Each software module (device driver, library, or application) that needs to receive an interrupt must implement a Tasks routine to handle that interrupt. In order for the module to operate in an interrupt-driven mode, the Tasks routine must be called from within the appropriate "raw" Interrupt Service Routine (ISR).

How the raw ISR is implemented is highly dependent upon the specific processor being used. Libraries are available that implement raw ISRs for each processor family in a way that allows dynamic registration and deregistration of Tasks routines. These libraries maintain tables that associate the Tasks routine registered by the SYS INIT service with each interrupt source in the system.

Alternately, in a statically-linked system implementation, the ISR may be implemented by the system designer or integrator (in the configuration-specific `system_interrupt.c` file). Such "static" ISR implementations must identify the source of the interrupt then directly call the appropriate module's Tasks routine. This requires knowledge of the modules that have been included in the system and cannot be implemented in advance as a library.

Note: It is also possible, in a highly optimized system (or to support highly resource-restricted parts), to implement the logic of the module's Tasks routine directly in the raw ISR. However, this method is not recommended unless absolutely necessary to meet system timing requirements.

Board Support Packages (BSPs)

If the processor is affixed directly to the board, the BSP may also implement any required "raw" ISRs, eliminating the need for the system designer or integrator to implement the ISR(s) himself. Refer to the documentation for the BSP in use for details on what initialization and ISR support it provides. This support is not implemented by the Interrupt System Services library.

Library Overview

Refer to the System Service Introduction section for how the system services operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Interrupt System Service.

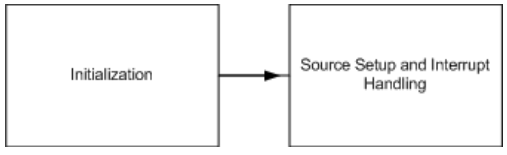
Library Interface Section	Description
Interrupt System Setup Functions	Provides processor specific initialization of the interrupt system
Global Interrupt Management Functions	Provide interface routines to enable/disable all interrupts on the system
Interrupt Source Management Functions	Provides setup and status routines for: • Setting up the source enable configuration • Querying the source interrupt status

How the Library Works

The Interrupt System Service Library can be used by a device driver, middleware layer, or application to provide access to, and control over, interrupts to the processor.

Description

The following diagram describes the major components of the usage model.



Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the modes supported for your device.

Interrupt System Setup

The Interrupt System Service library must be initialized by calling the SYS_INT_Initialize function. If the MPLAB Harmony dynamic initialization service is used, the SYS_INT_Initialize function will be called automatically when the SYS_Initialize function is called. In a statically initialized system, the system designer or integrator must implement the SYS_Initialize function and that function must call SYS_INT_Initialize before initializing any modules that might require use if the interrupt system service. Once the library has been initialized, call the function SYS_INT_Enable to enable interrupts to the processor. However, before enabling the generation of interrupts to the processor, each individual module (driver, library, or application) must have a "Tasks" routine to in place (either registered with SYS_INT_DynamicRegister or statically linked to the raw ISR) to handle the interrupt before it enables its own interrupt.

Example: Initializing the System Interrupt Library

```
SYS_INT_Initialize();
// Initialize all interrupt-aware software modules
SYS_INT_Enable();
```

Critical Sections

Critical Sections

Critical sections of code are small sections of code that must execute atomically, with no possibility of being interrupted. To support this, the following technique can be used.

Global Interrupt Management provides routines to create a global critical section of code.

Global Critical Section

If no interrupts of any kind can be allowed within a critical section of code, the following routines can be used to ensure this.

- SYS_INT_Disable: To start a critical section, all interrupts are disabled with the call of this function
- SYS_INT_Enable : To end a critical section, interrupts are enabled from the interrupt controller to the core
- SYS_INT_IsEnabled: Status to indicate if whether or interrupts are currently enabled

Example: Global Critical Section

```
bool flag;
flag = SYS_INT_Disable();

// Do something critical

if (flag)
{
    SYS_INT_Enable();
}
```

Source Interrupt Management provides interface routines to create local critical sections.

Local Critical Sections

Normally, it is not necessary to globally disable all possible interrupts. For example, in a driver for a specific device, it is not normally important if an unrelated interrupt occurs in the middle of a critical section of code. However, if the interrupt for the source that the driver manages must not occur within a critical section of code, it can be protected using the following technique.

Example: Local Critical Section

```
bool flag;
// interrupt source enable status before disable is called
flag = SYS_INT_SourceDisable(MY_DRIVER_INTERRUPT_SOURCE);

// Do something critical

if (flag)
{
    SYS_INT_SourceEnable(MY_DRIVER_INTERRUPT_SOURCE);
}
```

Note: These methods of protecting critical sections is usually implemented as part of an Operating System Abstraction Layer (OSAL), so it is not normally necessary to use these examples explicitly. Normally, the OSAL will provide single functions or macros that implement this functionality. So, if available, an OSAL method is preferred over implementing the critical section code as shown in the previous examples.

Source Interrupt Management

The driver, middleware, or application's interrupt-handling Tasks routine must do two things at a minimum, in the following order.

1. Remove the cause of the interrupt.
2. Clear the interrupt source by calling the function SYS_INT_SourceStatusClear.

Exactly what actions are necessary to remove the cause of an interrupt is completely dependent on the source of the interrupt. This is normally the main purpose of the driver itself and is beyond the scope of this section. Refer to the documentation for the peripheral being managed.

Warning The cause of the interrupt must be removed before clearing the interrupt source or the interrupt may reoccur immediately after the source is cleared potentially causing an infinite loop. An infinite loop may also occur if the source is not cleared before the interrupt-handler returns.

Example: Handling Interrupts

```
void DRV_MYDEV_Tasks( SYS_MODULE_OBJ object )
{
    // Remove the cause of the interrupt
    //...

    // Clear Interrupt source
    SYS_INT_SourceStatusClear(myIntSourceID);
}
```

Note: The value of `myIntSourceID` is usually either a static or dynamic configuration option. Refer to the documentation for the specific device driver to identify how to define the interrupt source ID.

Testing Interrupt

Sometimes it is necessary to cause an interrupt in software, possibly for testing purposes. To support this, the function SYS_INT_SourceStatusSet is provided.

Example: Causing an Interrupt in Software

```
SYS_INT_SourceStatusSet(MY_DRIVER_INTERRUPT_SOURCE);
```

Note: This feature is not available for all interrupt sources on all Microchip microcontrollers. Refer to the specific device data sheet to determine whether it is possible for software to set a specific interrupt source.

Configuring the Library

This section provides information on configuring the Interrupt System Service Library.

Description

To use the Interrupt System Service Library, the following must be correctly configured:

- Select the Appropriate Processor
- Initialize the Interrupt System Service
- Configure the Raw ISR Support

Select the Appropriate Processor

The following data types are dependent on the processor selection and are actually defined in the Interrupt Peripheral Library for the specific microcontroller being used.

- INT_SOURCE
- INT_PRIORITY
- INT_SUBPRIORITY

These data types are configured by selecting the appropriate processor in MPLAB X IDE, which adds the "mprocessor" option to the compiler command line to identify the correct processor and processor-specific implementation of the peripheral library to use. Since the Interrupt System Service Library is part of the Microchip Firmware Framework, it will be built with the correct definition of these data types.

Initialize the Interrupt System Service

There are two ways to initialize the interrupt system service, depending on whether you are using a static configuration or a dynamic configuration.

For a Dynamic configuration the constant SYS_INT_DYNAMIC needs to be defined. This makes the SYS_INT_DynamicRegister and SYS_INT_DynamicDeregister functions available. The required driver tasks routines need to be registered using SYS_INT_DynamicRegister function.

For a Static configuration, the system designer or integrator must implement the SYS_INT_Initialize function. This function's purpose is to perform any actions necessary to initialize the interrupt subsystem and interrupt controller on the specific processor and system, usually interacting directly

with the Interrupt Peripheral Library to accomplish these tasks.

Configure the Raw ISR Support

In some systems, there may only be a single actual (raw) ISR to handle all interrupts. In this sort of system, most of the Interrupt System Service Library may be implemented in software, with only the highest level interrupt being supported by hardware. In other systems, all interrupts may be supported by separate ISRs and vector selection and prioritization will be supported by hardware.

ISRs may be dynamically linked to specific interrupt sources or they may be statically linked at build time. If a dynamic interrupt library is used (by defining the constant `SYS_INT_DYNAMIC`), the calls to the SYS_INT_DynamicRegister function will register a pointer to the given Tasks routine for each registered interrupt source in an internal table. The dynamic library will then determine the source of the interrupt and call the given Tasks routine.

If a static configuration is desired, the "raw" ISR support must be implemented so that it directly calls (using static, build-time linkage) the appropriate module's Tasks routine. This requires the system implementer or integrator to implement the raw ISR, but it reduces the amount of overhead necessary to handle interrupts, reducing both interrupt latency and code size.

Static Configuration

When statically configuring raw ISR support, the system implementer or integrator must directly implement the raw ISRs in an appropriate manner for the selected processor. The raw ISR, must then call the appropriate Tasks routine to properly handle and clear the interrupt source.

Description

A static configuration of the raw ISR support for MPLAB Harmony requires processor-family-specific knowledge. Or, more accurately, it requires compiler-specific knowledge. The following example shows how to implement a raw ISR for the PIC32 family of devices. Refer to the compiler manual for details of how to implement an ISR.

Raw ISR Responsibilities:

- Identify the interrupt source
- Call the appropriate module's Tasks routine

The first thing a raw ISR must do is identify the source of the interrupt. Each interrupt source has its own interrupt "vector". This means that the only time a specific ISR is called is when a specific source has caused an interrupt. Therefore, the raw ISR can assume that every time it is called, its source has caused an interrupt. Once the raw ISR has identified the interrupt source, it must call the appropriate module's Tasks routine to service and clear the interrupt.

Example: PIC32 Timer1 Raw ISR

```
void __ISR ( _TIMER_1_VECTOR ) _InterruptHandler_TMR_1_stub( void )
{
    /* Call the timer driver's "Tasks" routine */
    DRV_TMR_Tasks ( gTMRObject );
}
```

In the example, gTMRObject holds the return value from the DRV_TMR_Initialize function.

The SYS_INT_DynamicRegister and SYS_INT_DynamicDeregister functions are macro switched to compile away to nothing if a static configuration is chosen.

Dynamic Configuration

When dynamically configuring raw ISR support, the system implementer or integrator must register each interrupt-driven driver or module's Tasks routine with the dynamic system interrupt service for the appropriate interrupt source. The dynamic SYS INT service will then ensure that the appropriate Tasks routine is called when an interrupt occurs.

Description

When using the dynamic system interrupt (SYS INT) service, it is not necessary to implement raw ISRs for interrupt-driven modules. The processor-family-specific, dynamic SYS INT implementation provided with MPLAB Harmony implements the raw ISRs so the system developer or integrator does not have to. Instead, the system developer must register the module's Tasks" routine using the SYS_ModuleRegister function after registering the module in the module registration routine (described in the SYS INIT documentation). The following example shows how a module must register its ISR Tasks routine.

Example: Dynamic Registration of an Interrupt-Driven Module

```
// Register the TMR driver's "Tasks" routine with the SYS INT service
SYS_INT_DynamicRegister(object, DRV_TMR_Tasks, PLIB_INT_SOURCE_TIMER_1);
```

The module init routine must register the module's Tasks routine with the SYS INT service instead of the SYS TASKS service. To do this, it calls the SYS_INT_DynamicRegister function, passing in the object (the same object handle returned by the module's initialization routine), along with a pointer to the module's Tasks routine and the interrupt source with which it will be associated.

Note: Dynamic interrupt registration functionality is currently not supported in the Interrupt System Service.

Building the Library

This section lists the files that are available in the Interrupt System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/int`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
sys_int.h	Interrupt System Service Library API header file.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/sys_int_pic32.c	Interrupt System Service Library implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Interrupt System Service does not depend on any other modules.

Library Interface

a) Interrupt System Setup Functions

	Name	Description
	SYS_INT_Initialize	Configures and initializes the interrupt subsystem.
	SYS_INT_DynamicDeregister	Deregisters the current ISR from the given interrupt source.
	SYS_INT_DynamicRegister	Registers an Interrupt "Tasks" Routine for the specified interrupt source or trap).
	SYS_INT_ShadowRegisterAssign	Assigns a shadow register set for an interrupt priority level.
	SYS_INT_ShadowRegisterGet	Gets the shadow register set assigned for an interrupt priority level.
	SYS_INT_StatusGetAndDisable	Disables interrupts to the processor and return the previous status.
	SYS_INT_StatusRestore	Restores the processor status.

b) Global Interrupt Management Functions

	Name	Description
	SYS_INT_Disable	Disables interrupts to the processor.
	SYS_INT_Enable	Enables global interrupts to the processor.
	SYS_INT_IsEnabled	Identifies if interrupts are currently enabled or disabled at the top level.
	SYS_INT_ExternalInterruptTriggerSet	Sets the external interrupt trigger type.

c) Interrupt Source Management Functions

	Name	Description
	SYS_INT_SourceDisable	Disables the specified source from generating interrupts to the processor.
	SYS_INT_SourceEnable	Enables the specified source to generate interrupts to the processor.

	SYS_INT_SourceIsEnabled	Identifies if the specified source is enabled or disabled.
	SYS_INT_SourceStatusClear	Clears the interrupt request for the specified interrupt source.
	SYS_INT_SourceStatusGet	Determines the status of the specified interrupt source.
	SYS_INT_SourceStatusSet	Sets the specified interrupt source.
	SYS_INT_VectorPrioritySet	Sets the given interrupt vector to the specified priority.
	SYS_INT_VectorSubprioritySet	Sets the specified interrupt vector to the given sub priority.

d) Data Types and Constants

	Name	Description
	SYS_INT_TASKS_POINTER	Pointer to an interrupt-handling "Tasks" routine.
	INT_EXTERNAL_EDGE_TRIGGER	Lists the available external interrupt trigger options.
	SYS_INT_PROCESSOR_STATUS	CPU Processor status

Description

This section describes the APIs of the Interrupt System Service Library.

Refer to each section for a detailed description.

a) Interrupt System Setup Functions

SYS_INT_Initialize Function

Configures and initializes the interrupt subsystem.

File

sys_int.h

C

```
void SYS_INT_Initialize();
```

Returns

None.

Description

This function configures and initializes the interrupt subsystem appropriately for the current system design.

Remarks

None.

The System Interrupt library must be initialized by calling the SYS_INT_Initialize routine. This is normally done in the "SYS_Initialize" routine before any interrupt support is used. If the dynamic interrupt system service is not used, the "SYS_Initialize" routine must be implemented by the system designer or integrator as required by the system design and is not implemented by the System Interrupt library. The global interrupts are enabled as a part the call to "SYS_INT_Initialize". However, before enabling the generation of interrupts to the processor, each individual module (driver, library, or application) must have a "Tasks" routine to in place (statically linked to the raw ISR) to handle the interrupt before it enables it's own interrupt.

Example: Initializing the System Interrupt Library

```
SYS_INT_Initialize();
```

Preconditions

None.

Example

```
// Initialize the interrupt system. This needs to done in the initialization
// code.
SYS_INT_Initialize();
```

Function

void SYS_INT_Initialize (void)

SYS_INT_DynamicDeregister Function

Deregisters the current ISR from the given interrupt source.

File

sys_int.h

C

```
void SYS_INT_DynamicDeregister(INT_SOURCE source);
```

Returns

None.

Description

This function deregisters the current Interrupt Service Routine (ISR), if any, from the specified interrupt source.

Remarks

It is safe to call this routine, even of no ISR has been registered for the given interrupt source.

Calling this routine is optional. If the system is designed such that the given ISR is expected to always be available once the system has been initialized, this routine does not need to be called.

Preconditions

SYS_INT_Initialize must have been called.

Example

```
SYS_INT_DynamicDeregister(SYS_INT_TIMER_1);
```

Parameters

Parameters	Description
source	Identifier for the desired interrupt source

Function

void SYS_INT_DynamicDeregister (INT_SOURCE source)

SYS_INT_DynamicRegister Function

Registers an Interrupt "Tasks" Routine for the specified interrupt source or trap).

File

sys_int.h

C

```
void SYS_INT_DynamicRegister(INT_SOURCE source, SYS_INT_TASKS_POINTER tasks, SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function registers an Interrupt "Tasks" Routine for the specified interrupt source or trap).

Remarks

This routine only generates executable code when a driver is configured to register dynamically its "Tasks" routine with the system interrupt service. However, it should be called even if the ISR-to-source association is defined statically at compile time to maintain source code compatibility. A device driver normally registers its own ISR from its initialization routine.

In the example code, the macros MY_DRIVER_INTERRUPT_SOURCE would be defined in the appropriate configuration header, which would be included by the driver source file where the "DRV_MYDEV_Tasks" routine and the MyParam data would be defined.

It is safe to call this routine without first calling SYS_INT_DynamicDeregister, even if a previous ISR has been registered. The effect will be that the new ISR supplants the old one.

The System Interrupt library must be initialized by calling the SYS_INT_Initialize routine. This is normally done in the "SYS_Initialize" routine before any interrupt support is used.

If the dynamic interrupt system service is not used, the "SYS_Initialize" routine must be implemented by the system designer or integrator as

required by the system design and is not implemented by the System Interrupt library.

Once the library has been initialized, call the function SYS_INT_Enable to enable interrupts to the processor. However, before enabling the generation of interrupts to the processor, each individual module (driver, library, or application) must have a "Tasks" routine to in place (either registered with SYS_INT_DynamicRegister or statically linked to the raw ISR) to handle the interrupt before it enables it's own interrupt.

Example: Initializing the System Interrupt Library

```
// Initialize the interrupt system.
SYS_INT_Initialize();
// Initialize all interrupt-aware software modules
SYS_INT_Enable();
```

Preconditions

SYS_INT_Initialize must have been called.

Example

```
SYS_INT_Initialize();
SYS_INT_DynamicRegister(MY_DRIVER_INTERRUPT_SOURCE, DRV_MYDEV_Tasks, MyObject);
```

Parameters

Parameters	Description
source	Identifier for the desired interrupt source
tasks	Pointer to the tasks routine
object	Handle to the module instance

Function

void SYS_INT_DynamicRegister(INT_SOURCE source,
SYS_INT_TASKS_POINTER tasks,
SYS_MODULE_OBJ object)

SYS_INT_ShadowRegisterAssign Function

Assigns a shadow register set for an interrupt priority level.

File

sys_int.h

C

```
void SYS_INT_ShadowRegisterAssign(INT_PRIORITY_LEVEL priority, INT_SHADOW_REGISTER shadowRegister);
```

Returns

None.

Description

The function assigns a shadow register set for an interrupt priority level.

Remarks

This feature may not be available on all devices.

Preconditions

None.

Example

```
SYS_INT_ShadowRegisterAssign( INT_PRIORITY_LEVEL5, INT_SHADOW_REGISTER_5 );
```

Parameters

Parameters	Description
priority	Interrupt priority level for which the shadow register set has to be assigned.
shadowRegister	Shadow register set number.

Function

void SYS_INT_ShadowRegisterAssign (INT_PRIORITY_LEVEL priority,
INT_SHADOW_REGISTER shadowRegister)

SYS_INT_ShadowRegisterGet Function

Gets the shadow register set assigned for an interrupt priority level.

File

sys_int.h

C

```
INT_SHADOW_REGISTER SYS_INT_ShadowRegisterGet(INT_PRIORITY_LEVEL priority);
```

Returns

None.

Description

The function gets the shadow register set assigned for an interrupt priority level.

Remarks

This feature may not be available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
INT_SHADOW_REGISTER shadowReg;

shadowReg = SYS_INT_ShadowRegisterGet( INT_PRIORITY_LEVEL5 );
```

Parameters

Parameters	Description
priority	Interrupt priority level for which the shadow register set has to be assigned.

Function

INT_SHADOW_REGISTER SYS_INT_ShadowRegisterGet (INT_MODULE_ID index,
INT_PRIORITY_LEVEL priority)

SYS_INT_StatusGetAndDisable Function

Disables interrupts to the processor and return the previous status.

File

sys_int.h

C

```
SYS_INT_PROCESSOR_STATUS SYS_INT_StatusGetAndDisable();
```

Returns

- SYS_INT_PROCESSOR_STATUS - 32 bit value holding the processor status before disabling the interrupt

Description

This function disables interrupts to the processor at the top level. This function can be called to prevent any source from being able to generate an interrupt. It returns the processor status (which includes global interrupt status and Interrupt Priority status) before disabling the interrupts.

Remarks

Previous Master interrupt status can be found by checking 0th bit of the returned value.

This function should be paired with the use of SYS_INT_StatusRestore(). The value returned from this function should be passed into SYS_INT_StatusRestore() function.

Preconditions

SYS_INT_Initialize must have been called.

Example

```
SYS_INT_PROCESSOR_STATUS  processorStatus;

// Save the processor status (which includes global interrupt status)
// and Disable the global interrupts
processorStatus = SYS_INT_StatusGetAndDisable();

{
    // Do something critical
}

// set the processor status back to the one which was there before
// disabling the global interrupt
SYS_INT_StatusRestore(processorStatus);
```

Function

SYS_INT_PROCESSOR_STATUS SYS_INT_StatusGetAndDisable (void)

SYS_INT_StatusRestore Function

Restores the processor status.

File

sys_int.h

C

```
void SYS_INT_StatusRestore(SYS_INT_PROCESSOR_STATUS processorStatus);
```

Returns

None.

Description

This function sets the processor status based on the 32 bit value passed as a parameter. 0th bit of the status is for Master Interrupt status.

Remarks

This function should be paired with the use of SYS_INT_GetStateAndDisable(). The value returned from SYS_INT_GetStateAndDisable() should be passed into this function.

Preconditions

None.

Example

```
SYS_INT_PROCESSOR_STATUS processorStatus;

processorStatus = SYS_INT_GetStateAndDisable();
{
    // do some critical work
}
SYS_INT_StatusRestore(processorStatus);
```

Parameters

Parameters	Description
processorStatus	value returned from previous call to SYS_INT_StatusGetAndDisable().

Function

void SYS_INT_StatusRestore (SYS_INT_PROCESSOR_STATUS processorStatus)

b) Global Interrupt Management Functions

SYS_INT_Disable Function

Disables interrupts to the processor.

File

sys_int.h

C

```
bool SYS_INT_Disable();
```

Returns

true - Global Interrupts are enabled (before the call to disable) false - Global Interrupts are disabled (before the call to disable)

Description

This function disables interrupts to the processor at the top level. This function can be called to prevent any source from being able to generate an interrupt. It returns the global interrupt status before disabling the interrupts.

Remarks

This API will be deprecated. Use "SYS_INT_StatusGetAndDisable" instead.

Preconditions

SYS_INT_Initialize must have been called.

Example

```
// Interrupt enable status
bool flag;
// Disable the global interrupts
flag = SYS_INT_Disable();
// Do something critical
// Check if interrupts were disabled/enabled
if (flag)
{
    // enable the global interrupts if they were enabled before the
    // call to SYS_INT_Disable()
    SYS_INT_Enable();
}
```

Function

bool SYS_INT_Disable (void)

SYS_INT_Enable Function

Enables global interrupts to the processor.

File

sys_int.h

C

```
void SYS_INT_Enable();
```

Returns

None.

Description

This function enables interrupts to the processor at the top level, allowing any currently enabled source to generate an interrupt. This function must be called before any source will be able to generate an interrupt.

Remarks

SYS_INT_Enable is called from the SYS_INT_Initialize() function.

Preconditions

None.

Example

```
// Check if global interrupts are enabled
if(!SYS_INT_IsEnabled())
{
    // Enable the global interrupts.
    SYS_INT_Enable();
}
```

Function

void SYS_INT_Enable (void)

SYS_INT_IsEnabled Function

Identifies if interrupts are currently enabled or disabled at the top level.

File

sys_int.h

C

```
bool SYS_INT_IsEnabled();
```

Returns

- true - If the interrupts are currently enabled
- false - If the interrupts are currently disabled

Description

This function identifies if interrupts are enabled or disabled at the top level.

Remarks

None.

Preconditions

SYS_INT_Initialize must have been called.

Example

```
// Check if global interrupts are enabled
if ( SYS_INT_IsEnabled() )
{
    // Interrupt enable status
    bool flag;
    // Disable the global interrupts.
    flag = SYS_INT_Disable();
}
```

Function

bool SYS_INT_IsEnabled (void)

SYS_INT_ExternalInterruptTriggerSet Function

Sets the external interrupt trigger type.

File

sys_int.h

C

```
void SYS_INT_ExternalInterruptTriggerSet(INT_EXTERNAL_SOURCES source, INT_EXTERNAL_EDGE_TRIGGER
edgeTrigger);
```

Returns

None.

Description

This function sets the External interrupt trigger type. User can set for multiple sources in a single call.

Remarks

Not supported for all interrupt sources. Check the specific data sheet to know the supported interrupt sources.

Preconditions

None.

Example

```
SYS_INT_ExternalInterruptTriggerSet (   INT_EXTERNAL_INT_SOURCE0|INT_EXTERNAL_INT_SOURCE0,
                                        INT_EDGE_TRIGGER_RISING );
```

Parameters

Parameters	Description
source	Identifier for the desired interrupt source

Function

void SYS_INT_ExternalInterruptTriggerSet (INT_EXTERNAL_SOURCES source,
INT_EXTERNAL_EDGE_TRIGGER edgeTrigger)

c) Interrupt Source Management Functions

SYS_INT_SourceDisable Function

Disables the specified source from generating interrupts to the processor.

File

sys_int.h

C

```
bool SYS_INT_SourceDisable(INT_SOURCE source);
```

Returns

- true - The Interrupt source is enabled (before the call to SYS_INT_SourceDisable)
- false - The Interrupt source is disabled (before the call to SYS_INT_SourceDisable)

Description

This function disables the given source from generating interrupts the processor when events occur.It returns the interrupt source enable/disable status before disabling the interrupt source.

Remarks

None.

Preconditions

SYS_INT_Initialize must have been called.

Example

```
// interrupt source enable/disable status.
bool flag
// Initialize the interrupt system.This needs to done in the initialization
// code.
SYS_INT_Initialize();
// Disable the interrupt source
flag = SYS_INT_SourceDisable(INT_SOURCE_PARALLEL_PORT);
// before enabling the source check the enable/disable status
if(flag)
{
    SYS_INT_SourceEnable(INT_SOURCE_PARALLEL_PORT);
}
```

Parameters

Parameters	Description
source	Identifier for the desired interrupt source.

Function

bool SYS_INT_SourceDisable (INT_SOURCE source)

SYS_INT_SourceEnable Function

Enables the specified source to generate interrupts to the processor.

File

sys_int.h

C

```
void SYS_INT_SourceEnable(INT_SOURCE source);
```

Returns

None.

Description

This function enables the specified source to generate interrupts to the processor when events occur.

Remarks

An Interrupt Service Routine (ISR) for the given interrupt source must be ready to receive the call before the source is enabled.

Preconditions

SYS_INT_Initialize must have been called and an ISR must have been registered for the source.

Example

```
// Initialize the interrupt system. This needs to done in the initialization
// code.
SYS_INT_Initialize();
// Enable the interrupt source
SYS_INT_SourceEnable(INT_SOURCE_PARALLEL_PORT);
```

Parameters

Parameters	Description
source	Identifier for the desired interrupt source

Function

void SYS_INT_SourceEnable (INT_SOURCE source)

SYS_INT_SourceIsEnabled Function

Identifies if the specified source is enabled or disabled.

File

sys_int.h

C

```
bool SYS_INT_SourceIsEnabled(INT_SOURCE source);
```

Returns

- true - If the given source is currently enabled.
- false - If the given source is currently disabled.

Description

This function identifies if the specified source is currently enabled or is currently disabled.

Remarks

None.

Preconditions

SYS_INT_Initialize must have been called.

Example

```
// Initialize the interrupt system. This needs to done in the initialization
// code.
SYS_INT_Initialize();
// Check if the required interrupt source is enabled
if ( SYS_INT_SourceIsEnabled(INT_SOURCE_PARALLEL_PORT))
{
    // App code
}
```

Parameters

Parameters	Description
source	Identifier for the desired interrupt source.

Function

bool SYS_INT_SourceIsEnabled (INT_SOURCE source)

SYS_INT_SourceStatusClear Function

Clears the interrupt request for the specified interrupt source.

File

sys_int.h

C

```
void SYS_INT_SourceStatusClear(INT_SOURCE source);
```

Returns

None.

Description

This function clears the interrupt request for the specified interrupt source.

Remarks

None.

Preconditions

SYS_INT_Initialize must have been called.

Example

```
// Initialize the interrupt system. This needs to done in the initialization
// code.
SYS_INT_Initialize();
// Check if the interrupt source flag is set
if ( SYS_INT_SourceStatusGet(INT_SOURCE_PARALLEL_PORT) )
{
    // Clear the interrupt flag
    SYS_INT_SourceStatusClear(INT_SOURCE_PARALLEL_PORT);
}
```

Parameters

Parameters	Description
source	Identifier for the desired interrupt source

Function

void SYS_INT_SourceStatusClear (INT_SOURCE source)

SYS_INT_SourceStatusGet Function

Determines the status of the specified interrupt source.

File

sys_int.h

C

```
bool SYS_INT_SourceStatusGet(INT_SOURCE source);
```

Returns

- true - If the given interrupt source is currently set
- false - If the given interrupt source is not currently set

Description

This function determines the current status of the interrupt source.

Remarks

Works even if the interrupt source or interrupts in general have not been enabled, so it can be used for polling implementations.

Preconditions

SYS_INT_Initialize must have been called.

Example

```
// Initialize the interrupt system.This needs to done in the initialization
// code.
SYS_INT_Initialize();
// Check if the required interrupt source is set
if ( SYS_INT_SourceStatusGet(INT_SOURCE_PARALLEL_PORT) )
{
    // Handle interrupt
}
```

Parameters

Parameters	Description
source	Identifier for the desired interrupt source

Function

bool SYS_INT_SourceStatusGet (INT_SOURCE source)

SYS_INT_SourceStatusSet Function

Sets the specified interrupt source.

File

sys_int.h

C

```
void SYS_INT_SourceStatusSet(INT_SOURCE source);
```

Returns

None.

Description

This function sets the specified interrupt source, causing the processor to be interrupted if interrupts are enabled, the source has been enabled, and the priority is higher than the current priority.

Remarks

Not supported for all interrupt sources. Check the specific data sheet for software clear only interrupt sources.

The driver, middleware, or application's interrupt-handling "Tasks" routine must do two things at a minimum, in the following order.

1. Remove the cause of the interrupt

2. Clear the interrupt source by calling the function SYS_INT_SourceStatusClear

Exactly what actions are necessary to remove the cause of an interrupt is completely dependent on the source of the interrupt. This is normally the main purpose of the driver itself and is beyond the scope of this section. Refer to the documentation for the peripheral being managed.

WARNING! The cause of the interrupt must be removed before clearing the interrupt source or the interrupt may re-occur immediately after the source is cleared potentially causing an infinite loop. An infinite loop may also occur if the source is not cleared before the interrupt-handler returns.

Example: Handling Interrupts

```
void DRV_MYDEV_Tasks( SYS_MODULE_OBJ object )
{
    // Remove the cause of the interrupt
    //...

    // Clear Interrupt source
    SYS_INT_SourceStatusClear(myIntSourceID);
}
```

Note: the value of "myIntSourceID" is usually either a static or dynamic configuration option. Refer to the documentation for the specific device driver to identify how to define the interrupt source ID.

Testing Interrupt

Some times it is necessary to cause an interrupt in software, possibly for testing purposes. To support this, the function SYS_INT_SourceStatusSet is provided.

Example: Causing an Interrupt in Software

```
SYS_INT_SourceStatusSet(MY_DRIVER_INTERRUPT_SOURCE);
```

Note: This feature is not available for all interrupt sources on all Microchip microcontrollers. Refer to the data sheet for the microcontroller being used to determine if it is possible for software to set a specific interrupt source.

Preconditions

SYS_INT_Initialize must have been called and an ISR must have been registered for the source (if interrupts and the source are enabled).

Example

```
// Initialize the interrupt system. This needs to done in the initialization
// code.
SYS_INT_Initialize();
// Check if interrupt source flag is set
if ( !SYS_INT_SourceStatusGet(INT_SOURCE_PARALLEL_PORT) )
{
    // Set the interrupt source flag
    SYS_INT_SourceStatusSet(INT_SOURCE_PARALLEL_PORT);
}
```

Parameters

Parameters	Description
source	Identifier for the desired interrupt source

Function

void SYS_INT_SourceStatusSet (INT_SOURCE source)

SYS_INT_VectorPrioritySet Function

Sets the given interrupt vector to the specified priority.

File

sys_int.h

C

```
void SYS_INT_VectorPrioritySet(INT_VECTOR vector, INT_PRIORITY_LEVEL priority);
```

Returns

None.

Description

This routine sets the given interrupt vector to the specified priority.

Remarks

This feature is not supported on all devices. Refer to the specific device data sheet or family reference manual to determine whether this feature is supported.

In the example code, the macros MY_DRIVER_INTERRUPT_VECTOR, MY_DRIVER_ISR_PRIORITY would be defined appropriately during configuration.

Preconditions

SYS_INT_Initialize must have been called.

Example

```
#define MY_DRIVER_INTERRUPT_VECTOR  INT_VECTOR_T1
#define MY_DRIVER_ISR_PRIORITY      INT_PRIORITY_LEVEL2
// Initialize the interrupt system.This needs to done in the initialization
// code.
SYS_INT_Initialize();
// Assign priority to the interrupt vector
SYS_INT_VectorPrioritySet(MY_DRIVER_INTERRUPT_VECTOR, MY_DRIVER_ISR_PRIORITY);
```

Parameters

Parameters	Description
vector	Identifier for the desired interrupt vector
priority	Priority (if supported)

Function

void SYS_INT_VectorPrioritySet (INT_VECTOR vector, INT_PRIORITY_LEVEL priority)

SYS_INT_VectorSubprioritySet Function

Sets the specified interrupt vector to the given sub priority.

File

sys_int.h

C

```
void SYS_INT_VectorSubprioritySet(INT_VECTOR vector, INT_SUBPRIORITY_LEVEL subpriority);
```

Returns

None.

Description

This function sets the specified interrupt vector to the specified sub-priority.

Remarks

This feature is not supported on all devices. Refer to the specific device data sheet or family reference manual to determine whether this feature is supported.

In the example code, the macros MY_DRIVER_INTERRUPT_VECTOR, MY_DRIVER_ISR_SUB_PRIORITY would be defined appropriately during configuration.

Preconditions

SYS_INT_Initialize must have been called.

Example

```
#define MY_DRIVER_INTERRUPT_VECTOR  INT_VECTOR_T1
#define MY_DRIVER_ISR_PRIORITY      INT_PRIORITY_LEVEL2
#define MY_DRIVER_ISR_SUB_PRIORITY  INT_SUBPRIORITY_LEVEL1
// Initialize the interrupt system.This needs to done in the initialization
// code.
SYS_INT_Initialize();
// Assign priority to the interrupt vector
SYS_INT_VectorPrioritySet(MY_DRIVER_INTERRUPT_VECTOR, MY_DRIVER_ISR_PRIORITY);
// Assign sub-priority to the interrupt vector
SYS_INT_VectorSubprioritySet(MY_DRIVER_INTERRUPT_VECTOR, MY_DRIVER_ISR_SUB_PRIORITY);
```

Parameters

Parameters	Description
vector	Identifier for the desired interrupt vector
subpriority	Subpriority (if supported)

Function

void SYS_INT_VectorSubprioritySet(INT_VECTOR vector,
INT_SUBPRIORITY_LEVEL subpriority)

d) Data Types and Constants

SYS_INT_TASKS_POINTER Type

Pointer to an interrupt-handling "Tasks" routine.

File

sys_int.h

C

```
typedef void (* SYS_INT_TASKS_POINTER)(SYS_MODULE_OBJ object);
```

Description

Interrupt Tasks Routine Pointer

This data type defines a pointer to an interrupt-handling "Tasks" routine. The form of a tasks routine is as follows:

void My_Tasks (SYS_MODULE_OBJ object);

Where "MyTasks" is the name of the tasks routine and object is a Handle to the module instance.

Remarks

"Tasks" is normally defined by a device driver, middleware, or system layer.

The term Interrupt Service Routine (ISR) is used for the "raw" ISR code that is either located directly at the interrupt vector address or whose address is loaded from the interrupt vector. The term "Tasks" routine is used to identify the driver-specific routine that is called by the actual ISR to perform the tasks necessary to handle and clear the interrupt.

INT_EXTERNAL_EDGE_TRIGGER Enumeration

Lists the available external interrupt trigger options.

File

sys_int.h

C

```
typedef enum {
  INT_EDGE_TRIGGER_FALLING,
  INT_EDGE_TRIGGER_RISING
} INT_EXTERNAL_EDGE_TRIGGER;
```

Members

Members	Description
INT_EDGE_TRIGGER_FALLING	External interrupt trigger on falling edge
INT_EDGE_TRIGGER_RISING	External interrupt trigger on rising edge

Description

Interrupt external edge selection.

This enumeration lists all of the available external interrupt trigger options.

Remarks

None.

SYS_INT_PROCESSOR_STATUS Type

CPU Processor status

File

sys_int.h

C

```
typedef uint32_t SYS_INT_PROCESSOR_STATUS;
```

Description

Interrupt Processor Status

This data type holds the status of CPU register 2

Remarks

None.

Files

Files

Name	Description
sys_int.h	Interrupt System Service.

Description

This section lists the source and header files used by the library.

sys_int.h

Interrupt System Service.

Enumerations

	Name	Description
	INT_EXTERNAL_EDGE_TRIGGER	Lists the available external interrupt trigger options.

Functions

	Name	Description
	SYS_INT_Disable	Disables interrupts to the processor.
	SYS_INT_DynamicDeregister	Deregisters the current ISR from the given interrupt source.
	SYS_INT_DynamicRegister	Registers an Interrupt "Tasks" Routine for the specified interrupt source or trap).
	SYS_INT_Enable	Enables global interrupts to the processor.
	SYS_INT_ExternalInterruptTriggerSet	Sets the external interrupt trigger type.
	SYS_INT_Initialize	Configures and initializes the interrupt subsystem.
	SYS_INT_IsEnabled	Identifies if interrupts are currently enabled or disabled at the top level.
	SYS_INT_ShadowRegisterAssign	Assigns a shadow register set for an interrupt priority level.
	SYS_INT_ShadowRegisterGet	Gets the shadow register set assigned for an interrupt priority level.
	SYS_INT_SourceDisable	Disables the specified source from generating interrupts to the processor.
	SYS_INT_SourceEnable	Enables the specified source to generate interrupts to the processor.
	SYS_INT_SourceIsEnabled	Identifies if the specified source is enabled or disabled.
	SYS_INT_SourceStatusClear	Clears the interrupt request for the specified interrupt source.
	SYS_INT_SourceStatusGet	Determines the status of the specified interrupt source.
	SYS_INT_SourceStatusSet	Sets the specified interrupt source.
	SYS_INT_StatusGetAndDisable	Disables interrupts to the processor and return the previous status.
	SYS_INT_StatusRestore	Restores the processor status.
	SYS_INT_VectorPrioritySet	Sets the given interrupt vector to the specified priority.
	SYS_INT_VectorSubprioritySet	Sets the specified interrupt vector to the given sub priority.

Types

	Name	Description
	SYS_INT_PROCESSOR_STATUS	CPU Processor status
	SYS_INT_TASKS_POINTER	Pointer to an interrupt-handling "Tasks" routine.

Description

Interrupt System Service Library Interface Definition

This file contains the interface definition for the Interrupt System Service. It provides a way to interact with the interrupt subsystem to manage the occurrence of interrupts for sources supported by the system.

File Name

sys_int.h

Company

Microchip Technology Inc.

Memory System Service Library

This section describes the Memory System Service Library.

Introduction

Introduction to the Memory System Service.

Description

The Memory System Service provides support for the initialization of external memory and external memory controllers. It is configured to run in static mode by the MPLAB Harmony Configurator (MHC). When an external memory interface is enabled by MHC, the initialization code for the controller is generated and added to the system initialization sequence in the application. The specific timing parameters for the external memory to be used are selected by the user via MHC.

Using the Library

This topic describes the basic architecture of the Memory System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_memory.h`

The interface to the Memory System Service library is defined in the `sys_memory.h` header file, which is included by the `system.h` system service header file. Any C language source (`.c`) file that uses the Memory System Service library should include `system.h`.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Abstraction Model

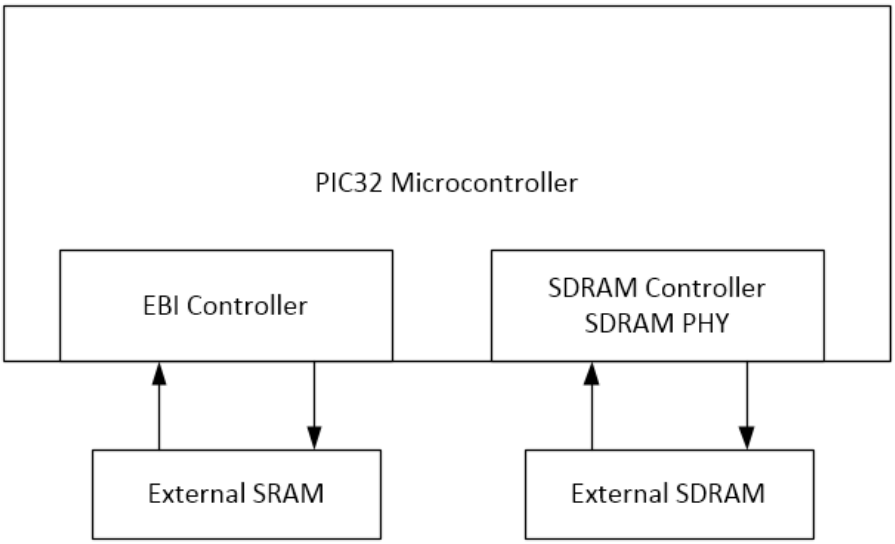
Provides information on the abstraction model for the Memory System Service.

Description

The Memory System Service manages the initialization of interfaces to external memory devices on some PIC32 microcontrollers. The initialization is performed during the system initialization sequence, so the memory is available to the application when the application tasks are initiated. The memory devices accessed via the interfaces initialized by this service are mapped into the virtual address space, and once initialized, are accessed directly just like internal SRAM. The type and number of external memory devices, as well as the virtual address mapping may vary between devices. Please refer to the specific device data sheet for details of a particular device.

The following figure provides the abstraction model for the Memory System Service.

Memory System Service Abstraction Model



Library Overview

Please refer to the System Service Introduction for a detailed description of MPLAB Harmony system services.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Memory System Service.

How the Library Works

The Memory System Service Library can be used by a device driver, middleware layer, or application to provide access to, and control over, interrupts to the processor.

Description

Initialization

Once the service is initialized, there is no further run-time configuration or operation required. The external memory is simply accessed via virtual address, just like internal SRAM. Therefore, there is a single API that manages the initialization of the selected external memory devices: SYS_MEMORY_Initialize.

The SYS_MEMORY_Initialize API will invoke the initialization function(s) of external memory devices enabled via MHC. There are presently two external memory controllers supported on some PIC32 microcontrollers, the External Bus Interface (EBI) and the SDRAM controller. The EBI can interface with external asynchronous SRAM or NOR Flash, while the SDRAM controller can interface with external DDR2 SDRAM.

```
SYS_MEMORY_EBI_Initialize();
SYS_MEMORY_DDR_Initialize();
```

Configuring the Library

The configuration of the Memory System Service is based on the file `system_config.h`.

This header file contains the configuration selection for the Memory System service. Based on the selections made, the Memory System Service may support the selected features. These configuration settings will apply to all instances of the Memory System Service.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Memory System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/memory`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
sys_memory.h	Memory System Service Library API header file.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
N/A	No source files are available for this library.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Memory System Service is not dependent upon other modules.

Library Interface

a) Functions

	Name	Description
	SYS_MEMORY_DDR_Initialize	Initializes and Enables the DDR External Memory Controller.
	SYS_MEMORY_EBI_Initialize	Initializes and Enables the EBI External Memory Controller.
	SYS_MEMORY_Initialize	Initializes and Enables the External Memory Controller(s).

Description

This section describes the APIs of the Memory System Service Library.

a) Functions

SYS_MEMORY_DDR_Initialize Function

Initializes and Enables the DDR External Memory Controller.

File

sys_memory.h

C

```
void SYS_MEMORY_DDR_Initialize();
```

Returns

None.

Description

This function Enables the external DDR memory controller module.

Remarks

This routine must be called before any attempt to access external DDR memory.

Not all features are available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
SYS_MEMORY_DDR_Initialize(NULL);
```

Parameters

Parameters	Description
data	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

void SYS_MEMORY_DDR_Initialize (void * data)

SYS_MEMORY_EBI_Initialize Function

Initializes and Enables the EBI External Memory Controller.

File

sys_memory.h

C

```
void SYS_MEMORY_EBI_Initialize();
```

Returns

None.

Description

This function Enables the external EBI memory controller module.

Remarks

This routine must be called before any attempt to access external EBI memory.

Not all features are available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
SYS_MEMORY_EBI_Initialize(NULL);
```

Parameters

Parameters	Description
data	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

void SYS_MEMORY_EBI_Initialize (void * data)

SYS_MEMORY_Initialize Function

Initializes and Enables the External Memory Controller(s).

File

sys_memory.h

C

```
void SYS_MEMORY_Initialize();
```

Returns

None.

Description

This function Enables the external memory controller module(s).

Remarks

This routine must be called before any attempt to access external memory.

Not all features are available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
SYS_MEMORY_Initialize(NULL);
```

Parameters

Parameters	Description
data	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

void SYS_MEMORY_Initialize (void * data)

Files

Files

Name	Description
sys_memory.h	Memory System Service Implementation.

Description

This section lists the source and header files used by the library.

sys_memory.h

Memory System Service Implementation.

Functions

	Name	Description
	SYS_MEMORY_DDR_Initialize	Initializes and Enables the DDR External Memory Controller.
	SYS_MEMORY_EBI_Initialize	Initializes and Enables the EBI External Memory Controller.
	SYS_MEMORY_Initialize	Initializes and Enables the External Memory Controller(s).

Description

Memory System Service Implementation

The Memory System Service provides a simple interface to manage the memory controllers. This file implements the core interface routines for the Memory System Service.

File Name

sys_memory.h

Company

Microchip Technology Inc.

Messaging System Service Library

This section describes the Messaging System Service Library.

Introduction

This library provides intra-process and inter-process communication by the sending and receiving of simple messages. The format of these messages is under developer control, providing flexibility to tune message format for each application. The number and size of message queues is under developer control, with a message priority scheme implemented by multiple queues. Each priority queue has an configurable size. The number of message types and the number of receiving mailboxes is also configurable.

Description

This library provides intra-process and inter-process communication by the sending and receiving of simple messages. The format of these messages is under developer control, providing flexibility to tune message format for each application. The number and size of message queues is under developer control, with a message priority scheme implemented by multiple queues. Each priority queue has an configurable size. The number of message types and the number of receiving mailboxes is also configurable.

Consider an application that combines graphics on a display screen, with a touch overlay on top of the screen, and several buttons. System messages can be used by touch software to alert the graphics part of the application to update the screen as well as alert other parts of the application to perhaps change the audio volume whenever a slider value has changed. This is easily supported by the Messaging System Services Library.

Using the Library

This topic describes the basic architecture of the Messaging System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_msg.h`

The interface to the Messaging System Service library is defined in the `sys_msg.h` header file, which is included by the `system.h` system service header file. Any C language source (`.c`) file that uses the Messaging System Service library should include `system.h`.

Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Abstraction Model

This library provides an abstraction of the messaging subsystem that is used by device drivers, middleware libraries and applications to receive and control interrupts in real time.

This library uses `calloc` to allocate memory for:

- Message queues for each priority as part of SYS_MSG_Initialize
- Mailbox definition objects
- Message type objects

If calls to `calloc` fails to allocate the needed memory then object handles are returned with a value of `SYS_OBJ_HANDLE_INVALID`.

Library Overview

Please refer to the System Service Introduction for a detailed description of MPLAB Harmony system services.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Messaging System Service module

Library Interface Section	Description
Initialization, Tasks, and Versioning Functions	Provides configuration and control functions.
Mailbox Functions	Provides mailbox APIs.
Message Type Functions	Provides messaging type APIs.
Message Send/Receive Functions	Provides messaging send and receive APIs.
Utility Functions	Provides messaging utility APIs.

How the Library Works

The Messaging System Service Library can be used by a device driver, middleware layer, or application to provide access to, and control over, interrupts to the processor.

Description

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the modes supported for your device.

The following example can be run on any PIC32 starter kit:

```
#include <sys/appio.h>
#include <GenericTypeDefs.h>

#include "system/common/sys_module.h"
#include "system/msg/sys_msg.h"
```

```
#include "system/msg/src/sys_msg_local.h"

// Strawman callback functions for three mailboxes
void myCallBack0( SYS_MSG_OBJECT *pMessage )
{
    DBPRINTF("CallBack0:: Message Type: %d, nSource: %d\r\n"
             "            param 0: %d, param1: %d, param2: %d \r\n",
             pMessage->nMessageTypeID,
             pMessage->nSource, pMessage->param0,
             pMessage->param1,  pMessage->param2
            );
}

void myCallBack1( SYS_MSG_OBJECT *pMessage )
{
    DBPRINTF("CallBack1:: Message Type: %d, nSource: %d\r\n"
             "            param 0: %d, param1: %d, param2: %d \r\n",
             pMessage->nMessageTypeID,
             pMessage->nSource, pMessage->param0,
             pMessage->param1,  pMessage->param2
            );
}

void myCallBack2( SYS_MSG_OBJECT *pMessage )
{
    DBPRINTF("CallBack2:: Message Type: %d, nSource: %d\r\n"
             "            param 0: %d, param1: %d, param2: %d \r\n",
             pMessage->nMessageTypeID,
             pMessage->nSource, pMessage->param0,
             pMessage->param1,  pMessage->param2
            );
}


int main(void)
{
    SYS_MSG_MESSAGING_OBJECT oSysMsg;
    SYS_OBJ_HANDLE hSysMsg, hMsgType[5], hMailbox[3];
    SYS_MSG_INSTANCE iSysMsg = SYS_MSG_0;
    SYS_MSG_OBJECT myMessage[5];


    {// SYS_MSG_Initialize
        uint8_t nQSizes[] = SYS_MSG_BUFFER_SIZES;

        // Initialize the messaging system. This needs to done in the initialization code.
        hSysMsg = SYS_MSG_Initialize(iSysMsg,SYS_MSG_MAX_PRIORITY+1,nQSizes);

        SYS_ASSERT( SYS_OBJ_HANDLE_INVALID != hSysMsg,"Bad hSysMsg!" );
    }

    // Create the message types to be used
    //                                    ID:    :Priority
    hMsgType[0] = SYS_MSG_TypeCreate(iSysMsg,1<<0,0);
    hMsgType[1] = SYS_MSG_TypeCreate(iSysMsg,1<<1,1);
    hMsgType[2] = SYS_MSG_TypeCreate(iSysMsg,1<<2,2);
    hMsgType[3] = SYS_MSG_TypeCreate(iSysMsg,1<<3,3);
    hMsgType[4] = SYS_MSG_TypeCreate(iSysMsg,1<<4,4);

    // Create the mailboxes to be used
    hMailbox[0] = SYS_MSB_MailboxOpen( iSysMsg, &myCallBack0 );
    hMailbox[1] = SYS_MSB_MailboxOpen( iSysMsg, &myCallBack1 );
    hMailbox[2] = SYS_MSB_MailboxOpen( iSysMsg, &myCallBack2 );

 // Identify which messages are of interest for each mailbox.
    SYS_MSG_MailboxMsgAdd(hMailbox[0],hMsgType[0]);
    SYS_MSG_MailboxMsgAdd(hMailbox[0],SYS_MSG_ID2hMsgType(iSysMsg,1<<1));
    SYS_MSG_MailboxMsgAdd(hMailbox[0],SYS_MSG_ID2hMsgType(iSysMsg,1<<2));
    SYS_MSG_MailboxMsgAdd(hMailbox[0],SYS_MSG_ID2hMsgType(iSysMsg,1<<3));
```

```
SYS_MSG_MailboxMsgAdd(hMailbox[0],SYS_MSG_ID2hMsgType(iSysMsg,1<<4));
SYS_MSG_MailboxMsgAdd(hMailbox[1],hMsgType[0]);
SYS_MSG_MailboxMsgAdd(hMailbox[2],hMsgType[0]);

{//Send and receive messages
    SYS_MSGQ_ELEMENT *pQElement;
    SYS_MSG_OBJECT  *pMessage;
    SYS_MSG_RESULTS myResult;
    SYS_MSG_QUEUE_STATUS qStatus[5];
    uint16_t iPriority;

    for (iPriority=0;iPriority<=SYS_MSG_MAX_PRIORITY;iPriority++)
    {
        myMessage[iPriority].nSource = SYS_MSG_MAX_PRIORITY-iPriority;
        myMessage[iPriority].nMessageTypeID = 1<<iPriority;
        myMessage[iPriority].param0 = iPriority;
        myMessage[iPriority].param1 = 0;
        myMessage[iPriority].param2 = 0;
    }

    myMessage[4].param1 = 0;
    myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[4] );
    SYS_ASSERT(myResult == SYS_MSG_SENT,"Message not sent!");

    myMessage[4].param1++;
    myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[4] );
    SYS_ASSERT(myResult == SYS_MSG_SENT,"Message not sent!");

    myMessage[4].param1++;
    myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[4] );
    SYS_ASSERT(myResult == SYS_MSG_SENT,"Message not sent!");

    myMessage[4].param1++;
    myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[4] );
    SYS_ASSERT(myResult == SYS_MSG_SENT,"Message not sent!");

    myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[4] ); // Should fail, queue full!
    SYS_ASSERT(myResult == SYS_MSG_NOT_SENT,"Result should NOT Be Sent!");

    myMessage[3].param1 = myMessage[4].param1;
    myMessage[3].param1++;
    myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[3] );
    SYS_ASSERT(myResult == SYS_MSG_SENT,"Message not sent!");

    myMessage[3].param1++;
    myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[3] );
    SYS_ASSERT(myResult == SYS_MSG_SENT,"Message not sent!");

    myMessage[2].param1 = myMessage[3].param1;
    myMessage[2].param1++;
    myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[2] );
    SYS_ASSERT(myResult == SYS_MSG_SENT,"Message not sent!");

    myMessage[2].param1++;
    myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[2] );
    SYS_ASSERT(myResult == SYS_MSG_SENT,"Message not sent!");

    myMessage[1].param1 = myMessage[2].param1;
    myMessage[1].param1++;
    myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[1] );
    SYS_ASSERT(myResult == SYS_MSG_SENT,"Message not sent!");

    myMessage[1].param1++;
    myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[1] );
    SYS_ASSERT(myResult == SYS_MSG_SENT,"Message not sent!");

    myMessage[0].param1 = myMessage[1].param1;
    myMessage[0].param1++;
    myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[0] );
```

```
        SYS_ASSERT(myResult == SYS_MSG_SENT,"Message not sent!");

        myMessage[0].param1++;
        myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[0] );
        SYS_ASSERT(myResult == SYS_MSG_SENT,"Message not sent!");

        myMessage[0].param1++;
        myResult = SYS_MSG_MessageSend( iSysMsg, &myMessage[0] );
        SYS_ASSERT(myResult == SYS_MSG_SENT,"Message not sent!");

        // Preemptively read all Mailbox Zero messages
        while ( NULL != (pMessage = SYS_MSG_MailboxMessagesGet(hMailbox[0])) )
        {
            // "Process" message.
            DBPRINTF("Mailbox Zero: Message Type: %d, nSource: %d\r\n"
                     "              param 0: %d, param1: %d, param2: %d \r\n",
                     pMessage->nMessageTypeID,
                     pMessage->nSource, pMessage->param0,
                     pMessage->param1,  pMessage->param2
                    );
        }

        // Deliver messages for all mailboxes.
        DBPRINTF("\r\nGot Messages: %d\r\n",SYS_MSG_GotMessages(iSysMsg));
        for (iPriority=0;iPriority<=SYS_MSG_MAX_PRIORITY;iPriority++)
        {
            qStatus[iPriority] =
               SYS_MSG_QueueStatus((SYS_OBJ_HANDLE)iSysMsg,iPriority);
        }
        DBPRINTF("Queue Status (4:-1:0): %d, %d, %d, %d, %d\r\n\r\n",
                  qStatus[4],qStatus[3],qStatus[2],qStatus[1],qStatus[0]);
        while ( (pQElement = SYS_MSG_MessageReceive(iSysMsg)) != NULL )
        {
            SYS_MSG_MessageDeliver(iSysMsg,pQElement);
            for (iPriority=0;iPriority<=SYS_MSG_MAX_PRIORITY;iPriority++)
            {
                qStatus[iPriority] =
                   SYS_MSG_QueueStatus((SYS_OBJ_HANDLE)iSysMsg,iPriority);
            }
            DBPRINTF("Queue Status (4:-1:0): %d, %d, %d, %d, %d\r\n\r\n",
                      qStatus[4],qStatus[3],qStatus[2],qStatus[1],qStatus[0]);
        }
        DBPRINTF("Got Messages: %d\r\n",SYS_MSG_GotMessages(iSysMsg));
    }

    return 0;
}
```

Configuring the Library

The file `sys_msg_config.h` file provides configuration parameters that the implementer can use to adapt the Messaging System Service Library to a particular application.

Macros

	Name	Description
	SYS_MSG_BUFFER_SIZES	define SYS_MSG_BUFFER_SIZES { 4, 4, 4, 4, 1
	SYS_MSG_MAX_MAILBOXES	Specifies the maximum number of mailboxes possible.
	SYS_MSG_MAX_MSGS_DELIVERED	Specifies the maximum number of messages delivered per each call to SYS_MSG_Tasks.
	SYS_MSG_MAX_PRIORITY	Specifies the maximum message priority.
	SYS_MSG_MAX_TYPES	Specifies the maximum number of message types possible.
	_SYS_MSG_CONFIG_TEMPLATE__H	This is macro _SYS_MSG_CONFIG_TEMPLATE__H.

Description

By default, the system message format provides 64 bits of information in a message:

```
    typedef union
    {
```

```
        struct
        {
            uint16_t         param0; // Message parameter zero
            uint16_t         param1; // Message parameter one
            uint16_t         param2; // Message parameter two
            uint16_t nMessageTypeID; // Message type identifier
        };
        struct
        {
            uint16_t        nSource; // Message source identifier
            uintptr_t      *  pData; // Pointer to additional message data
        };
    } SYS_MSG_OBJECT;
```

The only required field in the message format definition is `nMessageTypeID`. However, it does not need to be 16 bits long.

The maximum number of mailboxes is defined by:

```
#define SYS_MSG_MAX_MAILBOXES  (32)
```

The minimum number of mailboxes is one.

The maximum number of message types is defined by:

```
#define SYS_MSG_MAX_TYPES  (32)
```

The minimum number of message types is one.

Message priorities run from zero to SYS_MSG_MAX_PRIORITY, which is defined by:

```
#define SYS_MSG_MAX_PRIORITY   (4)
```

For each message priority from 0,1, to SYS_MSG_MAX_PRIORITY a queue is created. The size of each queue is defined by:

```
//           Message Priority:  0   1   2   3  4
#define SYS_MSG_BUFFER_SIZES { 64, 32, 16,  8, 4 }
```

In this example there are five priorities, 0, 1, ..4, and the sizes of each message queue is provided in the definition of SYS_MSB_BUFFER_SIZES.

Then, this information is used when initializing the Messaging System Service in the application start-up code:

```
SYS_OBJ_HANDLE hSysMsg;
{//SYS_MSG_Initialize
    uint16_t nQSizes[] = SYS_MSG_BUFFER_SIZES;
    hSysMsg = SYS_MSG_Initialize(SYS_MSG_MAX_PRIORITY+1,nQSizes);
    SYS_ASSERT( SYS_OBJ_HANDLE_INVALID != hSysMsg,"Bad hSysMsg!" );
}
```

SYS_MSG_BUFFER_SIZES Macro

File

sys_msg_config.h

C

```
#define SYS_MSG_BUFFER_SIZES { 16 }
```

Description

define SYS_MSG_BUFFER_SIZES { 4, 4, 4, 4, 1

SYS_MSG_MAX_MAILBOXES Macro

Specifies the maximum number of mailboxes possible.

File

sys_msg_config.h

C

```
#define SYS_MSG_MAX_MAILBOXES (2)
```

Description

System Messaging Max Number of Mailboxes

Specifies the maximum number of mailboxes possible.

Remarks

Minimum number is 1 mailbox.

SYS_MSG_MAX_MSGS_DELIVERED Macro

Specifies the maximum number of messages delivered per each call to SYS_MSG_Tasks.

File

sys_msg_config.h

C

```
#define SYS_MSG_MAX_MSGS_DELIVERED (1)
```

Description

System Messaging Maximum Number of Messages Delivered per call to SYS_MSG_Tasks

Specifies the maximum number of messages delivered per each call to SYS_MSG_Tasks. If zero then all message queues are emptied before the tasks routine finishes execution.

Remarks

0 implies all queues are empty after SYS_MSG_Tasks is done.

SYS_MSG_MAX_PRIORITY Macro

Specifies the maximum message priority.

File

sys_msg_config.h

C

```
#define SYS_MSG_MAX_PRIORITY (0)
```

Description

System Messaging Maximum Priority

Specifies the maximum message priority.

Remarks

Message priorities run from 0 to SYS_MSG_MAX_PRIORITIES. The number of message queues is SYS_MSG_MAX_PRIORITIES+1.

SYS_MSG_MAX_TYPES Macro

Specifies the maximum number of message types possible.

File

sys_msg_config.h

C

```
#define SYS_MSG_MAX_TYPES (2)
```

Description

System Messaging Max Number of Message Types

Specifies the maximum number of message types possible.

Remarks

Minimum number is 1.

_SYS_MSG_CONFIG_TEMPLATE__H Macro

File

sys_msg_config.h

C

```
#define _SYS_MSG_CONFIG_TEMPLATE__H
```

Description

This is macro _SYS_MSG_CONFIG_TEMPLATE__H.

Building the Library

This section lists the files that are available in the Messaging System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/msg`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`sys_msg.h`	Messaging System Service Library API header file.
`/src/sys_msg_local.h`	System messaging local declarations and definitions.
`/config/sys_msg_config.h`	System messaging configuration.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/sys_msg.c`	Messaging System Service Library implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Messaging System Service is not dependent upon other modules.

Library Interface

a) Initialization, Tasks, and Versioning Functions

	Name	Description
	SYS_MSG_Deinitialize	Deinitializes System Messaging Instance.
	SYS_MSG_Initialize	Configures and initializes the messaging subsystem.
	SYS_MSG_Tasks	System Messaging Service Tasks routine.

b) Mailbox Functions

	Name	Description
	SYS_MSG_MailboxMessagesGet	Gets queued messages for a mailbox.
	SYS_MSG_MailboxMsgAdd	Adds a message type to the list of messages received by a mailbox.
	SYS_MSG_MailboxMsgRemove	Removes a message type from the list of messages received by a mailbox.
	SYS_MSG_MailboxClose	Closes (destroys) a mailbox previously opened with SYS_MSG_MailboxOpen.
	SYS_MSG_MailboxOpen	Opens a system messaging mailbox.
	SYS_MSG_MailboxReinit	Reinitializes a previously opened mailbox.

c) Message Type Functions

	Name	Description
	SYS_MSG_TypeCreate	Creates a new message type.
	SYS_MSG_TypeRemove	Removes an existing message type.

d) Message Send/Receive Functions

	Name	Description
	SYS_MSG_GotMessages	Returns true if system messaging has undelivered messages, false otherwise.
	SYS_MSG_MessageDeliver	Delivers messages to mailboxes.
	SYS_MSG_MessageReceive	Receives the next message in the message queues.
	SYS_MSG_MessageSend	Sends a message, as defined by a message structure.

e) Utility Functions

	Name	Description
	SYS_MSG_ID2hMsgType	Translates message type identifier into handle of corresponding message type object.
	SYS_MSG_QueueStatus	Returns the message queue status for a given message priority.

f) Data Types and Constants

	Name	Description
	SYS_MSG_INIT	Contains all the data necessary to initialize an instance of the System Messaging Service.
	SYS_MSG_INSTANCE	System Messaging instances numbering is from 0,1, to SYS_MSG_MAX_INSTANCE.
	SYS_MSG_OBJECT	This is type SYS_MSG_OBJECT.
	SYS_MSG_QUEUE_STATUS	Messaging queue status enumeration.
	SYS_MSG_RECEIVE_CALLBACK	Pointer to the System message received callback function.
	SYS_MSG_RESULTS	Enumeration of message send results.
	SYS_MSGQ_ELEMENT	Defines queue element for message queue belonging to each priority.
	SYS_MSG_MAILBOXES_ADDONE	don't need to round up the number of bitmaps
	SYS_MSG_NUM_MAILBOX_BITMAPS	This is macro SYS_MSG_NUM_MAILBOX_BITMAPS.
	SYS_OBJ_HANDLE_INVALID	This is macro SYS_OBJ_HANDLE_INVALID.
	SYS_OBJ_HANDLE_STATIC	This is macro SYS_OBJ_HANDLE_STATIC.
	SYS_OBJ_HANDLE	SYS_MODULE_OBJ was poorly named. It should be SYS_MODULE_OBJ_HANDLE or something shorter. For brevity, it was renamed to SYS_OBJ_HANDLE.

Description

This section describes the APIs of the Messaging System Service Library.

Refer to each section for a detailed description.

a) Initialization, Tasks, and Versioning Functions

SYS_MSG_Deinitialize Function

Deinitializes System Messaging Instance.

File

sys_msg.h

C

```
void SYS_MSG_Deinitialize(SYS_OBJ_HANDLE handleSysMsg);
```

Returns

None.

Description

This function deinitializes the System Messaging Instance and frees up allocated memory for it.

Remarks

None.

Preconditions

None.

Example

```
SYS_OBJ_HANDLE hSysMsg;
uint16_t nQSizes[] = SYS_MSG_BUFFER_SIZES;

// Initialize the messaging system. This needs to done in the initialization code.
// Choose System Messaging instance that supports 8 byte messages
hSysMsg = SYS_MSG_Initialize(SYS_MSG_8Bytes,SYS_MSG_MAX_PRIORITY+1,nQSizes);

if (SYS_OBJ_HANDLE_INVALID == hSysMsg)
{
    // Handle error
}
.
.
.
// Remove this instance.
SYS_MSG_Deinitialize( hSysMsg );
```

Parameters

Parameters	Description
hSysMsg	handle to System Messaging Object for instance to be removed.

Function

void SYS_MSG_Deinitialize (SYS_OBJ_HANDLE hSysMsg)

SYS_MSG_Initialize Function

Configures and initializes the messaging subsystem.

File

sys_msg.h

C

```
SYS_OBJ_HANDLE SYS_MSG_Initialize(const SYS_MSG_INSTANCE iSysMsg, SYS_OBJ_HANDLE pInitializeSysMsg);
```

Returns

Handle to the System Messaging object created. Returns SYS_OBJ_HANDLE_INVALID if allocation of data structure fails. Returns SYS_OBJ_HANDLE_INVALID if pointer to initialization data structure is NULL.

Description

This function configures and initializes the messaging subsystem appropriately for the current system design.

Remarks

None.

Preconditions

None.

Example

```
  SYS_OBJ_HANDLE hSysMsg;
  SYS_MSG_INIT sInitSysMsg = { 0, (SYS_MSG_MAX_PRIORITY+1), { SYS_MSG_BUFFER_SIZES } };
//uint16_t nQSizes[] = SYS_MSG_BUFFER_SIZES;

  // Initialize the messaging system. This needs to done in the initialization code.
  // Choose System Messaging instance that supports 8 byte messages
  hSysMsg = SYS_MSG_Initialize(SYS_MSG_8Bytes,&sInitSysMsg);
```

```
if (SYS_OBJ_HANDLE_INVALID == hSysMsg)
{
    // Handle error
}
```

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service to be initialized.
pInitSysMsg	Pointer to System Messaging initialization data structure. If NULL default config values are used.

Function

SYS_OBJ_HANDLE SYS_MSG_Initialize (const SYS_MSG_INSTANCE iSysMsg,
SYS_OBJ_HANDLE pInitSysMsg)

SYS_MSG_Tasks Function

System Messaging Service Tasks routine.

File

sys_msg.h

C

```
void SYS_MSG_Tasks(SYS_OBJ_HANDLE handleSysMsg);
```

Returns

None.

Description

This function is the System Messaging Service Tasks routine.

Remarks

None.

Preconditions

hSysMsg must have been returned from a call to SYS_MSG_Initialize.

Example

```
while ( SYS_MSG_GotMessages(iSysMsg) )
{
    SYS_MSG_Tasks(hSysMsg);
}
```

Parameters

Parameters	Description
hSysMsg	handle to System Messaging Object.

Function

void SYS_MSG_Tasks (SYS_OBJ_HANDLE hSysMsg)

b) Mailbox Functions

SYS_MSG_MailboxMessagesGet Function

Gets queued messages for a mailbox.

File

sys_msg.h

C

```
SYS_MSG_OBJECT * SYS_MSG_MailboxMessagesGet(SYS_OBJ_HANDLE hMailbox);
```

Returns

Pointer to next message in the queue(s) that is of interest to the mailbox. Function returns NULL if no messages are found in the queue.

Description

This function gets the queued messages for a mailbox. Messages returned by this function will not be received via the mailbox's callback function.

Remarks

None.

Preconditions

hMailbox provided by call to SYS_MSG_MailboxOpen.

Example

```
SYS_MSG_OBJECT *pNextMessage;
while ( NULL != (pNextMessage = SYS_MSG_MailboxMessagesGet(hMyMailbox)) )
{
    // Process message at *pNextMessage.
}
```

Parameters

Parameters	Description
hMailbox	Object handle to mailbox

Function

SYS_MSG_OBJECT *SYS_MSG_MailboxMessagesGet(SYS_OBJ_HANDLE hMailbox)

SYS_MSG_MailboxMsgAdd Function

Adds a message type to the list of messages received by a mailbox.

File

sys_msg.h

C

```
void SYS_MSG_MailboxMsgAdd(SYS_OBJ_HANDLE hMailbox, SYS_OBJ_HANDLE hMsgType);
```

Returns

None.

Description

This function adds a message type to the list of messages received by a mailbox.

Remarks

When the message type handle is unknown but the message ID is known use SYS_MSG_ID2hMsgType to provide the message type handle. See code example.

Preconditions

hMailbox provided by call to SYS_MSG_MailboxOpen.

Example

```
const SYS_MSG_INSTANCE iSysMsg;
SYS_OBJ_HANDLE hMsgType[5], hMailbox[3];

// Create three mailboxes
hMailbox[0] = SYS_MSG_MailboxOpen( iSysMsg, &myCallBack0 );
hMailbox[1] = SYS_MSG_MailboxOpen( iSysMsg, &myCallBack1 );
hMailbox[2] = SYS_MSG_MailboxOpen( iSysMsg, &myCallBack2 );

// Create five message types
//                              Message ID:  :Priority
hMsgType[0] = SYS_MSG_TypeCreate(iSysMsg,1<<0,0);
hMsgType[1] = SYS_MSG_TypeCreate(iSysMsg,1<<1,1);
hMsgType[2] = SYS_MSG_TypeCreate(iSysMsg,1<<2,2);
```

```
hMsgType[3] = SYS_MSG_TypeCreate(iSysMsg,1<<3,3);
hMsgType[4] = SYS_MSG_TypeCreate(iSysMsg,1<<4,4);

// Add messages to each mailbox
SYS_MSG_MailboxMsgAdd(hMailbox[0],hMsgType[0]);
SYS_MSG_MailboxMsgAdd(hMailbox[1],hMsgType[1]);
SYS_MSG_MailboxMsgAdd(hMailbox[2],hMsgType[2]);

SYS_MSG_MailboxMsgAdd(hMailbox[0],SYS_MSG_ID2hMsgType(iSysMsg,1<<1));
SYS_MSG_MailboxMsgAdd(hMailbox[0],SYS_MSG_ID2hMsgType(iSysMsg,1<<2));
SYS_MSG_MailboxMsgAdd(hMailbox[0],SYS_MSG_ID2hMsgType(iSysMsg,1<<3));
SYS_MSG_MailboxMsgAdd(hMailbox[0],SYS_MSG_ID2hMsgType(iSysMsg,1<<4));
```

Parameters

Parameters	Description
hMailbox	Object handle to mailbox
hMsgType	Handle to message type of interest for this mailbox.

Function

void SYS_MSG_MailboxMsgAdd(SYS_OBJ_HANDLE hMailbox, SYS_OBJ_HANDLE hMsgType)

SYS_MSG_MailboxMsgRemove Function

Removes a message type from the list of messages received by a mailbox.

File

sys_msg.h

C

```
void SYS_MSG_MailboxMsgRemove(SYS_OBJ_HANDLE hMailbox, SYS_OBJ_HANDLE hMsgType);
```

Returns

None.

Description

This function removes a message type from the list of messages received by a mailbox.

Remarks

When the message type handle is unknown but the message ID is known use SYS_MSG_ID2hMsgType to provide the message type handle. See code example.

Preconditions

hMailbox provided by call to SYS_MSG_MailboxOpen.

Example

```
const SYS_MSG_INSTANCE iSysMsg;
SYS_OBJ_HANDLE hMsgType[5], hMailbox[3];

// Create three mailboxes
hMailbox[0] = SYS_MSG_MailboxOpen( iSysMsg, &myCallBack0 );
hMailbox[1] = SYS_MSG_MailboxOpen( iSysMsg, &myCallBack1 );
hMailbox[2] = SYS_MSG_MailboxOpen( iSysMsg, &myCallBack2 );

// Create five message types
//                              Message ID:  :Priority
hMsgType[0] = SYS_MSG_TypeCreate(iSysMsg,1<<0,0);
hMsgType[1] = SYS_MSG_TypeCreate(iSysMsg,1<<1,1);
hMsgType[2] = SYS_MSG_TypeCreate(iSysMsg,1<<2,2);
hMsgType[3] = SYS_MSG_TypeCreate(iSysMsg,1<<3,3);
hMsgType[4] = SYS_MSG_TypeCreate(iSysMsg,1<<4,4);

// Add messages to each mailbox
SYS_MSG_MailboxMsgAdd(hMailbox[0],hMsgType[0]);
SYS_MSG_MailboxMsgAdd(hMailbox[1],hMsgType[1]);
SYS_MSG_MailboxMsgAdd(hMailbox[2],hMsgType[2]);
```

```
SYS_MSG_MailboxMsgAdd(hMailbox[0],SYS_MSG_ID2hMsgType(iSysMsg,1<<1));
SYS_MSG_MailboxMsgAdd(hMailbox[0],SYS_MSG_ID2hMsgType(iSysMsg,1<<2));
SYS_MSG_MailboxMsgAdd(hMailbox[0],SYS_MSG_ID2hMsgType(iSysMsg,1<<3));
SYS_MSG_MailboxMsgAdd(hMailbox[0],SYS_MSG_ID2hMsgType(iSysMsg,1<<4));
```

Parameters

Parameters	Description
hMailbox	Object handle to mailbox
hMsgType	Handle to message type to be ignored by this mailbox.

Function

void SYS_MSG_MailboxMsgRemove(SYS_OBJ_HANDLE hMailbox, SYS_OBJ_HANDLE hMsgType)

SYS_MSG_MailboxClose Function

Closes (destroys) a mailbox previously opened with SYS_MSG_MailboxOpen.

File

sys_msg.h

C

```
void SYS_MSG_MailboxClose(SYS_OBJ_HANDLE hMailbox);
```

Returns

None.

None.

Description

This function closes (destroys) a mailbox previously opened with SYS_MSG_MailboxOpen.

Preconditions

hMailbox provided by call to SYS_MSG_MailboxOpen.

Example

```
SYS_OBJ_HANDLE hMyMailbox;

hMyMailbox = SYS_MSG_MailboxOpen( iSysMsg, &myCallBackFunction );
SYS_MSG_MailboxClose(hMyMailbox);
```

Parameters

Parameters	Description
hMailbox	Handle to mailbox that is to be closed (destroyed).

Function

void SYS_MSG_MailboxClose(SYS_OBJ_HANDLE hMailbox)

SYS_MSG_MailboxOpen Function

Opens a system messaging mailbox.

File

sys_msg.h

C

```
SYS_OBJ_HANDLE SYS_MSG_MailboxOpen(const SYS_MSG_INSTANCE iSysMsg, SYS_MSG_RECEIVE_CALLBACK
msgCallBackFunction);
```

Returns

Handle to new system messaging mailbox.

Description

This function opens a system messaging mailbox, providing a message callback function that is called whenever a message is received.

Remarks

A null callback function disables messaging callbacks.

Preconditions

iSysMsg must have been used in a call to SYS_MSG_Initialize. Message callback function will not be called until SYS_MSG_MailboxSignUp has been used to sign up the mailbox for messages of interest.

Example

```
SYS_OBJ_HANDLE hMyMailbox;

hMyMailbox = SYS_MSG_MailboxOpen( iSysMsg, &myCallBackFunction );
if (SYS_OBJ_HANDLE_INVALID == SYS_MSG_TypeRemove)
{
    // Handle error
}
```

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.
msgCallBackFunction	pointer to message callback function

Function

SYS_OBJ_HANDLE SYS_MSG_MailboxOpen(const SYS_MSG_INSTANCE iSysMsg,
SYS_MSG_RECEIVE_CALLBACK msgCallBackFunction);

SYS_MSG_MailboxReinit Function

Reinitializes a previously opened mailbox.

File

sys_msg.h

C

```
void SYS_MSG_MailboxReinit(SYS_OBJ_HANDLE hMailbox, SYS_MSG_RECEIVE_CALLBACK msgCallBackFunction);
```

Returns

None.

Description

This function reinitializes a previously opened mailbox by providing a new callback function and clearing all message type assignments.

Remarks

A null callback function disables messaging callbacks.

Preconditions

hMailbox provided by call to SYS_MSG_MailboxOpen.

Example

```
SYS_OBJ_HANDLE hMyMailbox;

hMyMailbox = SYS_MSG_MailboxOpen( iSysMsg, &myCallBackFunction );
SYS_MSG_MailboxReinit(hMyMailbox,&anotherCallBackFunction );
```

Parameters

Parameters	Description
hMailbox	Object handle to mailbox
msgCallBackFunction	pointer to new message callback function

Function

void SYS_MSG_MailboxReinit(SYS_OBJ_HANDLE hMailbox,
SYS_MSG_RECEIVE_CALLBACK msgCallBackFunction);

c) Message Type Functions

SYS_MSG_TypeCreate Function

Creates a new message type.

File

sys_msg.h

C

```
SYS_OBJ_HANDLE SYS_MSG_TypeCreate(const SYS_MSG_INSTANCE iSysMsg, uint8_t nMessageTypeID, uint8_t
nMessagePriority);
```

Returns

Handle to new message type definition.

Description

This function creates a new message type, defining an integer message type and priority.

Remarks

None.

Preconditions

iSysMsg must have been used in a call to SYS_MSG_Initialize.

Example

```
SYS_OBJ_HANDLE hMsgType;
hMsgType = SYS_MSG_TypeCreate( iSysMsg, 1, 3 );
if (SYS_OBJ_HANDLE_INVALID == hMsgType)
{
    // Handle error
}
```

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.
nMessageTypeID	Integer message type identifier
nMessagePriority	Message priority, between 0 and SYS_MSG_MAX_PRIORITIES

Function

SYS_OBJ_HANDLE SYS_MSG_TypeCreate(const SYS_MSG_INSTANCE iSysMsg,
uint8_t nMessageTypeID,
uint8_t nMessagePriority)

SYS_MSG_TypeRemove Function

Removes an existing message type.

File

sys_msg.h

C

```
void SYS_MSG_TypeRemove(SYS_OBJ_HANDLE hMsgType);
```

Returns

None.

Description

This function removes an existing message type.

Remarks

When the message type handle is unknown but the message ID is known use SYS_MSG_ID2hMsgType to provide the message type handle. See code example.

Preconditions

None.

Example

```
SYS_OBJ_HANDLE hMsgType;
hMsgType = SYS_MSG_TypeCreate( iSysMsg, 1, 3 );
SYS_MSG_TypeRemove( hMsgType );
```

Alternately:

```
SYS_OBJ_HANDLE hMsgType;
hMsgType = SYS_MSG_TypeCreate( iSysMsg, 1, 3 );
SYS_MSG_TypeRemove( SYS_MSG_ID2hMsgType(iSysMsg,1) );
```

Parameters

Parameters	Description
hMsgType	Handle to message type that is to be removed

Function

void SYS_MSG_TypeRemove(SYS_OBJ_HANDLE hMsgType)

d) Message Send/Receive Functions

SYS_MSG_GotMessages Function

Returns true if system messaging has undelivered messages, false otherwise.

File

sys_msg.h

C

```
bool SYS_MSG_GotMessages(const SYS_MSG_INSTANCE iSysMsg);
```

Returns

- true - Undelivered system messages exist
- false - No undelivered system messages exist

Description

This function returns the status when system messaging has undelivered messages.

Remarks

None.

Preconditions

iSysMsg must have been used in a call to SYS_MSG_Initialize.

Example

```
DBPRINTF("rnGot Messages: %drn",SYS_MSG_GotMessages(iSysMsg));
while ( (pNextMessage = SYS_MSG_MessageReceive(iSysMsg)) != NULL )
{
    SYS_MSG_MessageDeliver(iSysMsg,pNextMessage);
    DBPRINTF("rn");
}
```

```
DBPRINTF("Got Messages: %drn",SYS_MSG_GotMessages(ihSysMsg));
```

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.

Function

bool SYS_MSG_GotMessages(const SYS_MSG_INSTANCE iSysMsg)

SYS_MSG_MessageDeliver Function

Delivers messages to mailboxes.

File

sys_msg.h

C

```
void SYS_MSG_MessageDeliver(const SYS_MSG_INSTANCE iSysMsg, SYS_MSGQ_ELEMENT * pQElement);
```

Returns

None.

Description

This function delivers messages to mailboxes, and removes the message from the queue when done.

Remarks

None.

Preconditions

iSysMsg must have been used in a call to SYS_MSG_Initialize.

Example

```
SYS_MSGQ_ELEMENT *pQElement;
SYS_MSG_OBJECT nextMessage;
while ( (pQElement = SYS_MSG_MessageReceive(iSysMsg)) != NULL )
{
    // In case you desire to examine message before delivering it.
    nextMessage = pQElement->sMessage;

    // Deliver message to all interested mailboxes
    SYS_MSG_MessageDeliver(iSysMsg,pQElement);
}
```

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.
pQElement	pointer to queue element to be delivered.

Function

void SYS_MSG_MessageDeliver(const SYS_MSG_INSTANCE iSysMsg, SYS_MSGQ_ELEMENT *pQElement);

SYS_MSG_MessageReceive Function

Receives the next message in the message queues.

File

sys_msg.h

C

```
SYS_MSGQ_ELEMENT * SYS_MSG_MessageReceive(const SYS_MSG_INSTANCE iSysMsg);
```

Returns

Pointer to next message, as found in a message queue element, NULL if there are no messages.

Description

This function receives the next message in the message queues, returning NULL if the queues are empty.

Remarks

None.

Preconditions

iSysMsg must have been used in a call to SYS_MSG_Initialize.

Example

```
SYS_MSGQ_ELEMENT *pQElement;
SYS_MSG_OBJECT nextMessage;
while ( (pQElement = SYS_MSG_MessageReceive(iSysMsg)) != NULL )
{
    // In case you desire to examine message before delivering it.
    nextMessage = pQElement->sMessage;

    // Deliver message to all interested mailboxes
    SYS_MSG_MessageDeliver(iSysMsg,pQElement);
}
```

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.

Function

SYS_MSGQ_ELEMENT *SYS_MSG_MessageReceive(const SYS_MSG_INSTANCE iSysMsg)

SYS_MSG_MessageSend Function

Sends a message, as defined by a message structure.

File

sys_msg.h

C

```
SYS_MSG_RESULTS SYS_MSG_MessageSend(const SYS_MSG_INSTANCE iSysMsg, SYS_MSG_OBJECT * pMessage);
```

Returns

Message result from SYS_MSG_RESULTS enumeration.

Description

This function sends a message, as defined by a message structure.

Remarks

None.

Preconditions

iSysMsg must have been used in a call to SYS_MSG_Initialize.

Example

```
SYS_MSG_OBJECT myMessage;
SYS_MSG_RESULTS myMessageStatus;
SYS_OBJ_HANDLE hMyMsgType;

hMyMsgType = SYS_MSG_TypeCreate( iSysMsg, 1, 3 );
if (SYS_OBJ_HANDLE_INVALID == hMyMsgType)
{
    // Handle error
}
```

```
myMessage.hMsgType = hMyMsgType;
myMessage.nSource  = myMsgSource;
myMessage.param1   = parameterOneValue;
myMessage.param1   = parameterTwoValue;

myMessageStatus = SYS_MSG_MessageSend(iSysMsg,&myMessage);
SYS_ASSERT( myMessageStatus > 0, "Bad message status!" );
```

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.
pMessage	Pointer to message definition.

Function

SYS_MSG_RESULTS SYS_MSG_MessageSend(const SYS_MSG_INSTANCE iSysMsg,
SYS_MSG_OBJECT *pMessage)

e) Utility Functions

SYS_MSG_ID2hMsgType Function

Translates message type identifier into handle of corresponding message type object.

File

sys_msg.h

C

```
SYS_OBJ_HANDLE SYS_MSG_ID2hMsgType(const SYS_MSG_INSTANCE iSysMsg, uint8_t nMessageTypeID);
```

Returns

Handle to message type definition object corresponding to the message type identifier. Returns NULL if the message type is not defined.

Description

This function translates the message type identifier into the handle of the corresponding message type object.

Remarks

This function is useful in situations where the message type identifier is known but the message type handle is not. This allows applications to statically define message type IDs and use them in code instead of having to wait until message handles are known and then dynamically sharing message handles via global variables.

Preconditions

iSysMsg must have been used in a call to SYS_MSG_Initialize.

Example

```
#define MY_MESSAGE_TYPE_ID 1;

  // Create message type
  SYS_OBJ_HANDLE hMsgType;
  uint8_t nPriority = 3;
  hMsgType = SYS_MSG_TypeCreate( iSysMsg, MY_MESSAGE_TYPE_ID, nPriority );
  if (SYS_OBJ_HANDLE_INVALID == hMsgType)
  {
      // Handle error
  }
  .
  .
  .
  // Remove message type without knowing message type handle
  SYS_MSG_TypeRemove(iSysMsg,SYS_MSG_ID2hMsgType(iSysMsg,MY_MESSAGE_TYPE_ID));
```

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.
nMessageTypeID	Integer message type identifier.

Function

SYS_OBJ_HANDLE SYS_MSG_ID2hMsgType(const SYS_MSG_INSTANCE iSysMsg,
uint8_t nMessageTypeID)

SYS_MSG_QueueStatus Function

Returns the message queue status for a given message priority.

File

sys_msg.h

C

```
SYS_MSG_QUEUE_STATUS SYS_MSG_QueueStatus(const SYS_MSG_INSTANCE iSysMsg, uint8_t nMessagePriority);
```

Returns

Number of messages in queue if not full or SYS_MSG_QUEUE_FULL if full. If message priority is not legal, returns SYS_MSG_QUEUE_BAD.

Description

This function returns the message queue status for a given message priority.

Remarks

None.

Preconditions

iSysMsg must have been used in a call to SYS_MSG_Initialize.

Example

Parameters

Parameters	Description
iSysMsg	Index of System Messaging Service instance.
nMessagePriority	message priority of interest, from zero to SYS_MSG_MAX_PRIORITIES.

Function

SYS_MSG_QUEUE_STATUS SYS_MSG_QueueStatus(const SYS_MSG_INSTANCE iSysMsg, uint8_t nMessagePriority)

f) Data Types and Constants

SYS_MSG_INIT Structure

Contains all the data necessary to initialize an instance of the System Messaging Service.

File

sys_msg.h

C

```
typedef struct {
  uint8_t nMaxMsgsDelivered;
  uint8_t nMessagePriorities;
  uint16_t * nQSizes;
} SYS_MSG_INIT;
```

Members

Members	Description
uint8_t nMaxMsgsDelivered;	Maximum number of messages delivered per call to SYS_MSG_Tasks
uint8_t nMessagePriorities;	Number of message priorities desired
uint16_t * nQSizes;	Array of queue sizes for priorities 0,1,...SYS_MSG_MAX_PRIORITY

Description

System Messaging Service Initialization Data

This data type contains all of the data necessary to initialize an instance of the System Messaging Service.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the SYS_MSG_Initialize routine. If nMaxMsgsDelivered == 0. ALL messages in priority queues are delivered each time SYS_MSG_Tasks is called.

SYS_MSG_INSTANCE Enumeration

System Messaging instances numbering is from 0,1, to SYS_MSG_MAX_INSTANCE.

File

sys_msg.h

C

```
typedef enum {
  SYS_MSG_0,
  SYS_MSG_1,
  SYS_MSG_2,
  SYS_MSG_3,
  SYS_MSG_4,
  SYS_MSG_NUM_INSTANCES
} SYS_MSG_INSTANCE;
```

Description

Enumeration of the Allowable of System Messaging Instances

System Messaging instances numbering is from 0,1, to SYS_MSG_MAX_INSTANCE.

Remarks

None.

SYS_MSG_OBJECT Structure

File

sys_msg.h

C

```
typedef struct {
  union {
    struct {
      uint8_t nMessageTypeID;
      uint8_t nSource;
      uint16_t param0;
      uint16_t param1;
      uint16_t param2;
    }
    struct {
      uint16_t dummy;
      uint16_t nSizeData;
      uintptr_t * pData;
    }
  }
} SYS_MSG_OBJECT;
```

Members

Members	Description
uint8_t nMessageTypeID;	Message type identifier
uint8_t nSource;	Message source identifier
uint16_t param0;	Message parameter zero
uint16_t param1;	Message parameter one
uint16_t param2;	Message parameter two
uint16_t nSizeData;	Size of data that pData identifies
uintptr_t * pData;	Pointer to additional message data

Description

This is type SYS_MSG_OBJECT.

SYS_MSG_QUEUE_STATUS Enumeration

Messaging queue status enumeration.

File

sys_msg.h

C

```
typedef enum {
  SYS_MSG_QUEUE_BAD,
  SYS_MSG_QUEUE_FULL,
  SYS_MSG_QUEUE_EMPTY
} SYS_MSG_QUEUE_STATUS;
```

Members

Members	Description
SYS_MSG_QUEUE_BAD	QUEUE Status: full
SYS_MSG_QUEUE_FULL	QUEUE Status: full
SYS_MSG_QUEUE_EMPTY	QUEUE Status: empty

Description

System Messaging Queue Status Enumeration

This enumeration provides the messaging queue status. Positive values indicate the number of messages in the queue.

Remarks

None.

SYS_MSG_RECEIVE_CALLBACK Type

Pointer to the System message received callback function.

File

sys_msg.h

C

```
typedef void (* SYS_MSG_RECEIVE_CALLBACK)(SYS_MSG_OBJECT *pMessage);
```

Description

Pointer to the System Message Received Callback Function

This data type is a pointer to the function provided for each system messaging mailbox that is called when a system message is received for each mailbox.

SYS_MSG_RESULTS Enumeration

Enumeration of message send results.

File

sys_msg.h

C

```
typedef enum {
  SYS_MSG_NOT_SENT_QFULL,
  SYS_MSG_BAD_PRIORITY,
  SYS_MSG_BAD_MSGTYPE,
  SYS_MSG_NOT_SENT,
  SYS_MSG_SENT
} SYS_MSG_RESULTS;
```

Members

Members	Description
SYS_MSG_NOT_SENT_QFULL	Message could not be sent, no room available in priority queues
SYS_MSG_BAD_PRIORITY	Message could not be sent, Message Message priority bad
SYS_MSG_BAD_MSGTYPE	Message could not be sent, Message type bad
SYS_MSG_NOT_SENT	Message could not be sent, no other information available
SYS_MSG_SENT	Message sent

Description

System Messaging Results Enumeration

This enumeration provides message send results.

Remarks

SYS_MSG_SENT aligns with SYS_MSGQ_Success. SYS_MSG_NOT_SENT aligns with SYS_MSGQ_Failure

SYS_MSGQ_ELEMENT Structure

Defines queue element for message queue belonging to each priority.

File

sys_msg.h

C

```
typedef struct {
  SYS_MSG_OBJECT sMessage;
  uint16_t mailboxInterestBitMap[SYS_MSG_NUM_MAILBOX_BITMAPS];
} SYS_MSGQ_ELEMENT;
```

Members

Members	Description
SYS_MSG_OBJECT sMessage;	System Message Bit map for mailboxes interested in this message type, modified as each mailbox is notified.

Description

System Messaging Queues Element

This data type defines the queue element for the message queue belonging to each priority.

Remarks

None.

SYS_MSG_MAILBOXES_ADDONE Macro

File

sys_msg.h

C

```
#define SYS_MSG_MAILBOXES_ADDONE 0
```

Description

don't need to round up the number of bitmaps

SYS_MSG_NUM_MAILBOX_BITMAPS Macro

File

sys_msg.h

C

```
#define SYS_MSG_NUM_MAILBOX_BITMAPS (SYS_MSG_MAX_MAILBOXES/16 + SYS_MSG_MAILBOXES_ADDONE)
```

Description

This is macro SYS_MSG_NUM_MAILBOX_BITMAPS.

SYS_OBJ_HANDLE_INVALID Macro

File

sys_msg.h

C

```
#define SYS_OBJ_HANDLE_INVALID ( (SYS_OBJ_HANDLE) -1 )
```

Description

This is macro SYS_OBJ_HANDLE_INVALID.

SYS_OBJ_HANDLE_STATIC Macro

File

sys_msg.h

C

```
#define SYS_OBJ_HANDLE_STATIC ( (SYS_OBJ_HANDLE)  0 )
```

Description

This is macro SYS_OBJ_HANDLE_STATIC.

SYS_OBJ_HANDLE Type

SYS_MODULE_OBJ was poorly named. It should be SYS_MODULE_OBJ_HANDLE or something shorter. For brevity, it was renamed to SYS_OBJ_HANDLE.

File

sys_msg.h

C

```
typedef uintptr_t SYS_OBJ_HANDLE;
```

Description

SYS_MODULE_OBJ Rename

SYS_MODULE_OBJ was poorly named. It should be SYS_MODULE_OBJ_HANDLE or something shorter. For brevity, it was renamed to SYS_OBJ_HANDLE.

Remarks

None.

Files

Files

Name	Description
sys_msg.h	System Service for the messaging module.
sys_msg_config.h	System Messaging Configuration definitions file

Description

This section lists the source and header files used by the library.

sys_msg.h

System Service for the messaging module.

Enumerations

	Name	Description
	SYS_MSG_INSTANCE	System Messaging instances numbering is from 0,1, to SYS_MSG_MAX_INSTANCE.
	SYS_MSG_QUEUE_STATUS	Messaging queue status enumeration.
	SYS_MSG_RESULTS	Enumeration of message send results.

Functions

	Name	Description
	SYS_MSG_Deinitialize	Deinitializes System Messaging Instance.
	SYS_MSG_GotMessages	Returns true if system messaging has undelivered messages, false otherwise.
	SYS_MSG_ID2hMsgType	Translates message type identifier into handle of corresponding message type object.
	SYS_MSG_Initialize	Configures and initializes the messaging subsystem.
	SYS_MSG_MailboxClose	Closes (destroys) a mailbox previously opened with SYS_MSG_MailboxOpen.
	SYS_MSG_MailboxMessagesGet	Gets queued messages for a mailbox.
	SYS_MSG_MailboxMsgAdd	Adds a message type to the list of messages received by a mailbox.
	SYS_MSG_MailboxMsgRemove	Removes a message type from the list of messages received by a mailbox.
	SYS_MSG_MailboxOpen	Opens a system messaging mailbox.
	SYS_MSG_MailboxReinit	Reinitializes a previously opened mailbox.
	SYS_MSG_MessageDeliver	Delivers messages to mailboxes.
	SYS_MSG_MessageReceive	Receives the next message in the message queues.
	SYS_MSG_MessageSend	Sends a message, as defined by a message structure.
	SYS_MSG_QueueStatus	Returns the message queue status for a given message priority.
	SYS_MSG_Tasks	System Messaging Service Tasks routine.
	SYS_MSG_TypeCreate	Creates a new message type.
	SYS_MSG_TypeRemove	Removes an existing message type.

Macros

	Name	Description
	SYS_MSG_MAILBOXES_ADDONE	don't need to round up the number of bitmaps
	SYS_MSG_NUM_MAILBOX_BITMAPS	This is macro SYS_MSG_NUM_MAILBOX_BITMAPS.
	SYS_OBJ_HANDLE_INVALID	This is macro SYS_OBJ_HANDLE_INVALID.
	SYS_OBJ_HANDLE_STATIC	This is macro SYS_OBJ_HANDLE_STATIC.

Structures

	Name	Description
	SYS_MSG_INIT	Contains all the data necessary to initialize an instance of the System Messaging Service.
	SYS_MSG_OBJECT	This is type SYS_MSG_OBJECT.
	SYS_MSGQ_ELEMENT	Defines queue element for message queue belonging to each priority.

Types

	Name	Description
	SYS_MSG_RECEIVE_CALLBACK	Pointer to the System message received callback function.
	SYS_OBJ_HANDLE	SYS_MODULE_OBJ was poorly named. It should be SYS_MODULE_OBJ_HANDLE or something shorter. For brevity, it was renamed to SYS_OBJ_HANDLE.

Description

Messaging System Service Library Interface Definitions

This file contains the interface definition for the messaging system service. It provides a way to interact with the messaging subsystem.

File Name

sys_msg.h

Company

Microchip Technology Inc.

sys_msg_config.h

System Messaging Configuration definitions file

Macros

	Name	Description
	_SYS_MSG_CONFIG_TEMPLATE__H	This is macro _SYS_MSG_CONFIG_TEMPLATE__H.
	SYS_MSG_BUFFER_SIZES	define SYS_MSG_BUFFER_SIZES { 4, 4, 4, 4, 1
	SYS_MSG_MAX_MAILBOXES	Specifies the maximum number of mailboxes possible.
	SYS_MSG_MAX_MSGS_DELIVERED	Specifies the maximum number of messages delivered per each call to SYS_MSG_Tasks.
	SYS_MSG_MAX_PRIORITY	Specifies the maximum message priority.
	SYS_MSG_MAX_TYPES	Specifies the maximum number of message types possible.

Description

System Messaging Configuration Definitions

These definitions statically define the operation of the System Messaging service.

File Name

sys_msg_config.h

Company

Microchip Technology Inc.

Ports System Service Library

This section describes the Ports System Service Library.

Introduction

This library provides an interface to manage and control general purpose input or output ports controlled by the Ports modules on the Microchip families of microcontrollers.

Description

One challenge designers of general purpose microcontroller devices face is to provide a large set of available peripheral features on parts with a limited number of I/O pins. To help meet this challenge and to provide flexibility for board designers, many devices provide an ability to route I/O signals for selected peripherals to different I/O pins. However, in some cases, general purpose I/O (GPIO) pins must be used to implement the desired functionality under direct software control. The purpose of the ports system service is to provide direct control if general purpose I/O pins and to support the selection of the desired peripheral functionality for supported I/O pins and ports.

General purpose I/O pins can be controlled individually, but they are also grouped together in sets and can be controlled together as a unit. These groups of GPIO pins are called "ports" on Microchip microcontrollers and this library provides the ability to read and write data patterns to or from them in sets or as individual pins. In addition peripheral IO routing and pin/port control, this library provides the ability to select and configure several other features of the I/O pins and ports available on Microchip microcontrollers, as follows.

Other Features

- Individual output pin/port open-drain enable/disable
- Individual input pin/port pull-up enable/disable
- Monitor select inputs and generate interrupt on mismatch condition [Change Notification]
- Operate during Sleep and Idle modes
- Port line Analog/Digital Selection
- Port slew rate control

Note: Trademarks and Intellectual Property are property of their respective owners. Customers are responsible for obtaining appropriate licensing or rights before using this software. Refer to the MPLAB Harmony *Software License Agreement* for complete licensing information. A copy of this agreement is available in the `<install-dir>/doc` folder of your MPLAB Harmony installation.

Using the Library

This topic describes the basic architecture of the Ports System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_ports.h`

The interface to the Ports System Service Library is defined in the `sys_ports.h` header file, which is included by the `sys.h` header file.

Any C language source (`.c`) file that uses the Ports System Service must include `sys.h`.

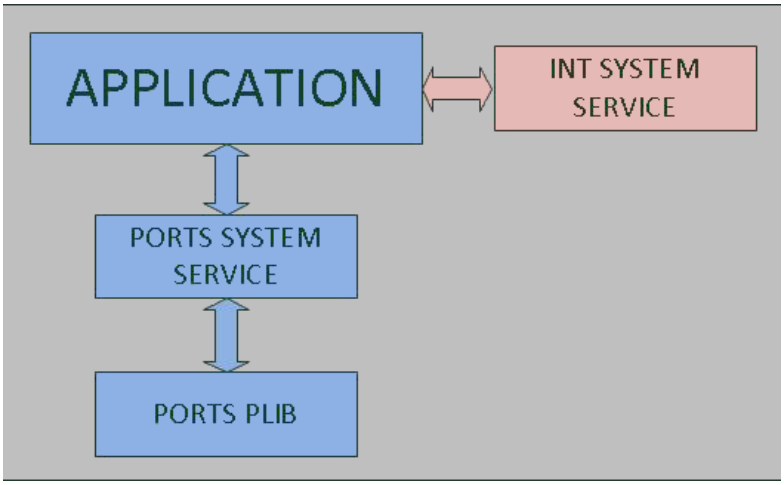
Please refer to the What is MPLAB Harmony? section for how the library interacts with the framework.

Abstraction Model

This library provides a low-level abstraction of the Ports System Service Library. This topic describes how that abstraction is modeled in software and introduces the library's interface.

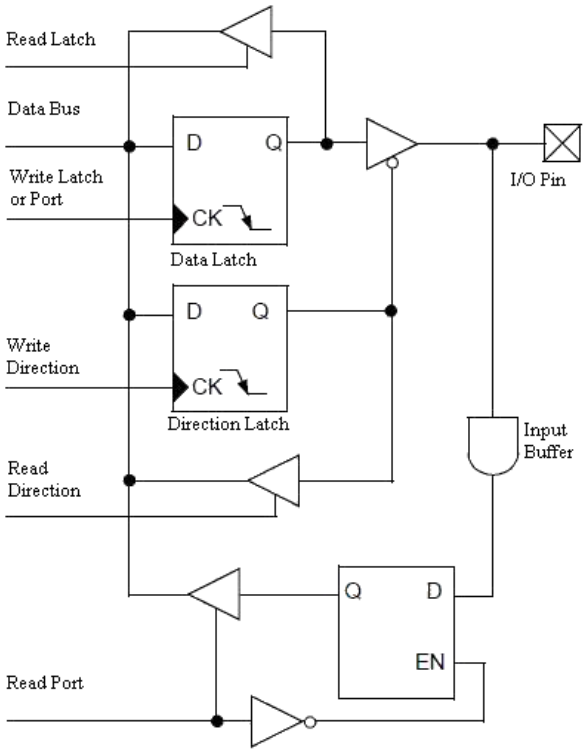
Description

This model explains how the system interfaces with the Ports System Service and the application as illustrated in the following diagram.



General Purpose I/O

All port pins have three registers directly associated with their operation as digital I/O. The Data Direction register determines whether the pin is an input or an output. If the data direction bit is a '1', the pin is an input. All port pins are defined as inputs after a Reset. Reads from the Output Latch register, read the latch, while writes to the latch, write the latch. Reads from the port, read the port pins, while writes to the port pins, write the latch.



The *pull-ups* act as a current source or sink source connected to the pin, and eliminates the need for external resistors when push-button or keypad devices are connected. These *pull-ups* prevent floating state of the pins by providing voltage to it. These features are available on some pins and some parts. Please refer to the specific device data sheet for further information.

The *open-drain* feature allows the generation of outputs higher than VDD (e.g., 5V) on any desired digital only pins by using external pull-up resistors. The maximum open-drain voltage allowed is the same as the maximum VIH specification.

The *input change notification* function of the I/O ports allows the microcontrollers to generate interrupt requests to the processor in response to a change of state on selected input pins. This feature is capable of detecting input change of states even in Sleep mode, when the clocks are disabled.

The *Alternate Pin Function selections* are used to steer specific peripheral input and output functions between different pins.

The *output slew rate* of each port is programmable to select either the standard transition rate or a reduced transition rate of x times the standard to minimize EMI. The reduced transition time is the default slew rate for all ports.

Peripheral Pin Select

Available Pins:

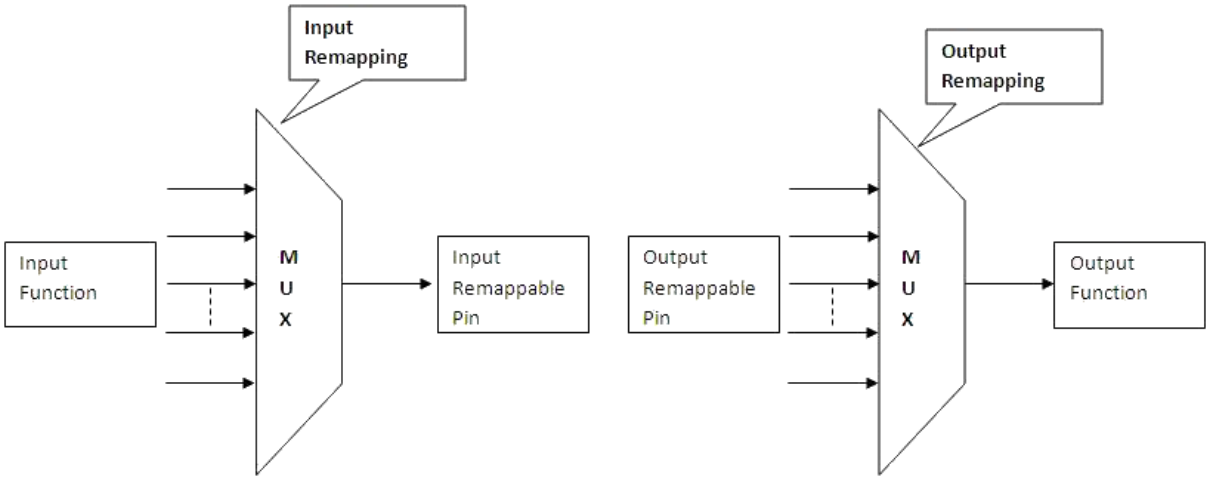
The number of available pins is dependent on the particular device and its pin count. Pins that support the peripheral pin select feature include the designation "RPn" in their full pin designation, where "RP" designates a remappable peripheral and "n" is the remappable port number.

Available Peripherals:

The peripherals managed by the peripheral pin select are all digital-only peripherals. These include general serial communications (UART and SPI), general purpose timer clock inputs, timer-related peripherals (Input Capture and Output Compare) and interrupt-on-change inputs.

In comparison, some digital-only peripheral modules are never included in the peripheral pin select feature. This is because the peripheral's function requires special I/O circuitry on a specific port and cannot be easily connected to multiple pins. These modules include I2C among others. A similar requirement excludes all modules with analog inputs, such as the A/D converter. A key difference between remappable and non-remappable peripherals is that remappable peripherals are not associated with a default I/O pin. The peripheral must always be assigned to a specific I/O pin before it can be used. In contrast, non-remappable peripherals are always available on a default pin, assuming that the peripheral is active and not conflicting with another peripheral.

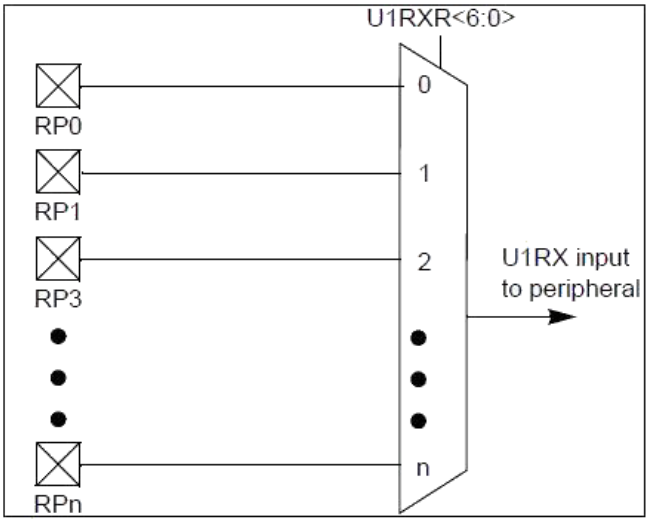
When a remappable peripheral is active on a given I/O pin, it takes priority over all other digital I/O and digital communication peripherals associated with the pin. Priority is given regardless of the type of peripheral that is mapped. Remappable peripherals never take priority over any analog functions associated with the pin.



Input Mapping:

The inputs of the peripheral pin select options are mapped on the basis of the peripheral. That is, a control register associated with a peripheral dictates to which pin it will be mapped.

Remappable Input Example for USART 1 Receive

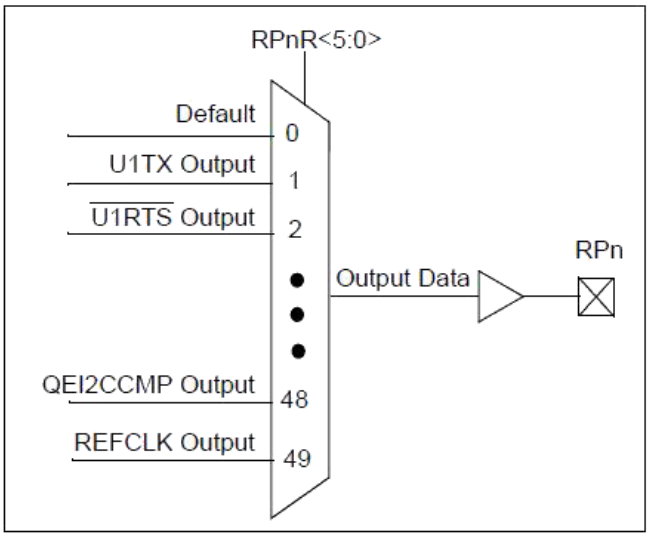


Note: For input only, peripheral pin select functionality does not have priority over I/O port settings. Therefore, when configuring Remappable Pin for input,the corresponding bit in the I/O port register must also be configured for input (set to '1').

Output Mapping:

In contrast to inputs, the outputs of the peripheral pin select options are mapped on the basis of the pin. In this case, a control register associated with a particular pin dictates the peripheral output to be mapped.

Multiplexing of Remappable Output for RPn



Mapping Limitations:

The control schema of the peripheral select pins is not limited to a small range of fixed peripheral configurations. There are no mutual or hardware-enforced lockouts between any of the peripheral mapping SFRs. Literally any combination of peripheral mappings across any or all of the RPn pins is possible. This includes both many-to-one and one-to-many mappings of peripheral inputs and outputs to pins. While such mappings may be technically possible from a configuration point of view, they may not be supportable from an electrical point of view.

Library Overview

Please refer to the System Service Introduction for a detailed description of MPLAB Harmony system services.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Ports System Service Library.

Library Interface Section	Description
Pin Control Functions	Port bit/pin read/write/toggle/clear/set interfaces.
Ports Control Functions	Port access read/write/toggle/clear/set interfaces.
Change Notification Functions	Interface routines for Port line change notification.
Peripheral Pin Select Functions	Interface routines for mapping the digital input/output to a specific PPS Remappable input/output pin.

How the Library Works

Pin Control

Pins Functions Usage

- **Pin Read:** Port pin can be read at bit/pin level using SYS_PORTS_PinRead with appropriate parameters
- **Pin Write:** Port pin can be written at bit/pin level using SYS_PORTS_PinWrite with appropriate parameters
- **Pin Clear:** Port pin can be cleared at bit/pin level using SYS_PORTS_PinClear with appropriate parameters
- **Pin Set:** Port pin can be set at bit/pin level using SYS_PORTS_PinSet with appropriate parameters
- **Pin Direction Control:** Port pin direction can be set at bit/pin level using SYS_PORTS_PinDirectionSelect with appropriate parameters. The Direction information can be obtained through the interface SYS_PORTS_DirectionGet.
- **Pin Toggle:** Port pin can be toggled at bit/pin level using SYS_PORTS_PinToggle with appropriate parameters
- **Pin Open Drain:** Port pin can be enabled open drain functionality at bit/pin level using SYS_PORTS_PinOpenDrainEnable with appropriate parameters. Similarly, the Port pin can be disabled open drain functionality at bit/pin level using SYS_PORTS_PinOpenDrainDisable with appropriate parameters.

Example:

```
// PORT Direction setting for output
SYS_PORTS_PinDirectionSelect(MY_PORTS_INSTANCE, SYS_PORTS_DIRECTION_OUTPUT, MY_CHANNEL, MY_PINNUM);
```

```
// PORT Direction setting for input
SYS_PORTS_PinDirectionSelect(MY_PORTS_INSTANCE, SYS_PORTS_DIRECTION_INPUT, MY_CHANNEL, MY_PINNUM);
// Writing a value into a PORT
SYS_PORTS_PinWrite(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM, MY_VALUE);
// Reading back the previously written value
bool readData = SYS_PORTS_PinRead(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
// Clearing the PORT
SYS_PORTS_PinClear(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
// Setting the port
SYS_PORTS_PinSet(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
// Toggling a PORT
SYS_PORTS_PinToggle(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

Note: Not all features are available on all devices. Please refer to the specific device data sheet to determine availability.

Ports Control

Port Functions Usage

- **Port Read:** Ports can be read at byte/word level using the interface SYS_PORTS_Read with appropriate parameters
- **Port Write:** Ports can be written to at byte/word level using the interface SYS_PORTS_Write appropriate parameters
- **Port Clear:** Ports can be cleared at byte/word level using the interface SYS_PORTS_Clear with appropriate parameters
- **Port Set:** Ports can be set at byte/word level using the interface SYS_PORTS_Set with appropriate parameters
- **Port Direction Control:** Ports direction can be set at byte/word level using the interface SYS_PORTS_DirectionSelect with appropriate parameters. The Direction information can be obtained through the interface SYS_PORTS_DirectionGet.
- **Port Toggle:** Ports can be toggled at byte/word level using the interface SYS_PORTS_Toggle with appropriate parameters
- **Port Open Drain:** Ports can be enabled open drain functionality at byte/word level using the interface SYS_PORTS_OpenDrainEnable with appropriate parameters. Similarly, the Ports can be disabled open drain functionality at byte/word level using the interface SYS_PORTS_OpenDrainDisable with appropriate parameters.

Example:

```
// PORT Direction setting for output
SYS_PORTS_DirectionSelect(MY_PORTS_INSTANCE, SYS_PORTS_DIRECTION_OUTPUT, MY_CHANNEL,
(PORTS_DATA_MASK)0xFFFF);
// PORT Direction setting for input
SYS_PORTS_DirectionSelect(MY_PORTS_INSTANCE, SYS_PORTS_DIRECTION_INPUT, MY_CHANNEL,
(PORTS_DATA_MASK)0xFFFF);
// Writing a value into a PORT
SYS_PORTS_Write(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_TYPE)0x1234)
// Reading back the previously written value
PORTS_DATA_TYPE readData = SYS_PORTS_Read(MY_PORTS_INSTANCE, MY_CHANNEL);
// Clearing the PORT
SYS_PORTS_Clear(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);
// Setting the port
SYS_PORTS_Set(MY_PORTS_INSTANCE, MY_CHANNEL, 0x1234, (PORTS_DATA_MASK)0x00FF);
// Toggling a PORT
SYS_PORTS_Toggle(MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF);
```

Note: Not all features are available on all devices. Please refer to the specific device data sheet to determine availability.

Change Notification

Change Notification Feature Usage

The change notification feature can be enabled using "SYS_PORTS_ChangeNotificationEnable". This routine performs the following operations:

- Change notification can be disabled after the successful usage using the interface SYS_PORTS_ChangeNotificationDisable
- Certain microcontrollers support the global control over the change notification feature using the following interfaces SYS_PORTS_ChangeNotificationGlobalEnable and SYS_PORTS_ChangeNotificationGlobalDisable
- If there are any requirements to control the pull-ups SYS_PORTS_ChangeNotificationPullUpEnable and SYS_PORTS_ChangeNotificationPullUpDisable could be used

Change Notification Operation in Sleep and Idle Modes

The change notification module continues to operate during Sleep or Idle mode. Its operation can be enabled and disabled using the interfaces SYS_PORTS_ChangeNotificationInIdleModeEnable and SYS_PORTS_ChangeNotificationInIdleModeDisable, respectively.

Example:

```
// Enabling the global change notification
SYS_PORTS_ChangeNotificationGlobalEnable(MY_PORTS_INSTANCE);
// Enabling weak pull-ups for the change notification PIN 10
SYS_PORTS_ChangeNotificationPullUpEnable(MY_PORTS_INSTANCE, PORTS_CHANGE_NOTICE_PIN_10);
// Enabling change notification on PIN 10
SYS_PORTS_ChangeNotificationEnable(MY_PORTS_INSTANCE, PORTS_CHANGE_NOTICE_PIN_10, SYS_PORTS_PULLUP_ENABLE);
// Enabling the change notification in idle mode.
SYS_PORTS_ChangeNotificationInIdleModeEnable(MY_PORTS_INSTANCE);
```

Note: Not all features are available on all devices. Please refer to the specific device data sheet to determine availability.

Peripheral Pin Select

Ports Remapping or Peripheral Pin Select Usage

Input/Output Function Remapping:

The SYS_PORTS_RemapInput and SYS_PORTS_RemapOutput functions with appropriate parameters can be used to remap a particular port pin as input/output for a peripheral.

Example:

```
// Remapping input function 'Input Capture 1' to the Remappable input pin 'RPD2'
SYS_PORTS_RemapInput (PORTS_ID_0, INPUT_FUNC_IC1, INPUT_PIN_RPD2 );
// Remapping output function 'UART1 Transmit' to the Remappable output pin 'RPA14'
SYS_PORTS_RemapInputOutput(PORTS_ID_0, OUTPUT_FUNC_U1TX, OUTPUT_PIN_RPA14);
```

Note: Not all features are available on all devices. Please refer to the specific device data sheet to determine availability.

Miscellaneous

Other Usage

Slew Rate:

Slew rate of a particular port can be controlled though the interfaces SYS_PORTS_SlewRateSetReduced and SYS_PORTS_SlewRateSetStandard.

Open Drain:

Peripheral based open drain can be controlled through the interfaces SYS_PORTS_PeripheralOpenDrainEnable and SYS_PORTS_PeripheralOpenDrainDisable.

Note: Not all features are available on all devices. Please refer to the specific device data sheet to determine availability.

Special Considerations

Note on Ports Usage:

1. Setting a port pin as an analog input also requires that the corresponding direction be set. If the direction is set to output, the digital output level (VOH or VOL) will be outputted.
2. When reading the port register, all pins configured as analog input channels will read as cleared.
3. Pins configured as digital inputs will not output an analog input. Analog levels on any pin that is defined as a digital input may cause the input buffer to consume current that exceeds the device specifications.
4. Pull-ups and pull-downs on change notification pins should always be disabled when the port pin is configured as a digital output.

Considerations for the Peripheral Pin Select:

The ability to control Peripheral Pin Select options introduces several considerations into application design that could be overlooked. This is particularly true for several common peripherals that are available only as remappable peripherals.

The main consideration is that the Peripheral Pin Selects are not available on default pins in the device's default (Reset) state, all Peripheral Pin Select inputs are tied to Vss and all Peripheral Pin Select outputs are disconnected. This situation requires the user to initialize the device with the proper peripheral configuration before any other application code is executed. For application safety, however, it is best to lock the configuration after writing to the control registers.

A final consideration is that Peripheral Pin Select functions neither override analog inputs, nor reconfigure pins with analog functions for digital I/O. If a pin is configured as an analog input on device Reset, it must be explicitly reconfigured as digital I/O when used with a Peripheral Pin Select.

Note: Not all features are available on all devices. Please refer to the specific device data sheet to determine availability.

Configuring the Library

The configuration of the Ports System Service is based on the file `system_config.h`.

This header file contains the configuration selection for the Ports System Service. Based on the selections made, the Ports System Service may support the selected features. These configuration settings will apply to all instances of the Ports System Service.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Ports System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/ports`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
sys_ports.h	Ports System Service Library API header file.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/sys_ports.c	Ports System Service Library implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The Reset System Service Library depends on the following modules:

- Ports Peripheral Library
- Device Control Peripheral Library

Library Interface

a) Pin Control Functions

	Name	Description
	SYS_PORTS_PinModeSelect	Enables the selected pin as analog or digital. **Implementation:** Dynamic
	SYS_PORTS_PinOpenDrainDisable	Disables the open-drain functionality for the selected pin. **Implementation:** Dynamic
	SYS_PORTS_PinOpenDrainEnable	Enables the open-drain functionality for the selected pin. **Implementation:** Dynamic
	SYS_PORTS_PinRead	Reads the selected digital pin. **Implementation:** Dynamic
	SYS_PORTS_PinSet	Sets the selected digital pin/latch. **Implementation:** Dynamic

	SYS_PORTS_PinToggle	Toggles the selected digital pin. **Implementation:** Dynamic
	SYS_PORTS_PinWrite	Writes the selected digital pin. **Implementation:** Dynamic
	SYS_PORTS_PinClear	Clears the selected digital pin. **Implementation:** Dynamic
	SYS_PORTS_PinDirectionSelect	Enables the direction for the selected pin. **Implementation:** Dynamic

b) Ports Control Functions

	Name	Description
	SYS_PORTS_Clear	Clears the selected digital port. **Implementation:** Dynamic
	SYS_PORTS_DirectionGet	Reads the direction for the selected port. **Implementation:** Dynamic
	SYS_PORTS_DirectionSelect	Enables the direction for the selected port. **Implementation:** Dynamic
	SYS_PORTS_OpenDrainDisable	Disables the open-drain functionality for the selected port. **Implementation:** Dynamic
	SYS_PORTS_OpenDrainEnable	Enables the open-drain functionality for the selected port. **Implementation:** Dynamic
	SYS_PORTS_Read	Reads the data from the I/O port. **Implementation:** Dynamic
	SYS_PORTS_Set	Sets the selected digital port/latch based on the mask. **Implementation:** Dynamic
	SYS_PORTS_Toggle	Toggles the selected digital port pins. **Implementation:** Dynamic
	SYS_PORTS_Write	Writes the data to the I/O port. **Implementation:** Dynamic
	SYS_PORTS_Initialize	Initializes PORT Pins/Channels. **Implementation:** Static/Dynamic

c) Change Notification Functions

	Name	Description
	SYS_PORTS_ChangeNotificationDisable	Disables the change notification for the selected port. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationEnable	Enables the change notification for the selected port. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationGlobalDisable	Globally disables the change notification for the selected port. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationGlobalEnable	Globally enables the change notification for the selected port. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationInIdleModeDisable	Disables the change notification for the selected port in Sleep or Idle mode. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationInIdleModeEnable	Enables the change notification for the selected port in Sleep or Idle mode. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationPullUpDisable	Disables a weak pull-up on the change notification pin. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationPullUpEnable	Enables a weak pull-up on the change notification pin. **Implementation:** Dynamic

d) Peripheral Pin Select Functions

	Name	Description
	SYS_PORTS_RemapInput	Input/Output (I/O) function remapping. **Implementation:** Dynamic
	SYS_PORTS_RemapOutput	Input/Output (I/O) function remapping. **Implementation:** Dynamic

e) Data Types and Constants

	Name	Description
	SYS_PORTS_PIN_DIRECTION	Defines the direction of the port pins.
	SYS_PORTS_PULLUP_PULLDOWN_STATUS	Provides the pull-up and pull-down status.

Description

This section describes the APIs of the Ports System Service Library.

Refer to each section for a detailed description.

a) Pin Control Functions

SYS_PORTS_PinModeSelect Function

Enables the selected pin as analog or digital.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_PinModeSelect(PORTS_MODULE_ID index, PORTS_ANALOG_PIN pin, PORTS_PIN_MODE mode);
```

Returns

None.

Description

This function enables the selected pin as analog or digital.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PIN       - PORTS_ANALOG_PIN_AN0
// MY_PIN_MODE  - PORTS_PIN_MODE_ANALOG
SYS_PORTS_PinModSYS_PORTS_PinModeSelecteSelect( MY_PORTS_INSTANCE, MY_PIN, MY_PIN_MODE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pin	Possible values of PORTS_ANALOG_PIN
mode	Possible values of PORTS_PIN_MODE

Function

void SYS_PORTS_PinModeSelect (PORTS_MODULE_ID index,
PORTS_ANALOG_PIN pin,
PORTS_PIN_MODE mode)

SYS_PORTS_PinOpenDrainDisable Function

Disables the open-drain functionality for the selected pin.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_PinOpenDrainDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function disables the open-drain functionality for the selected pin.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_PIN_10
SYS_PORTS_PinOpenDrainDisable(MY_PORTS_INSTANCE, MY_PINNUM);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void SYS_PORTS_PinOpenDrainDisable (PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

SYS_PORTS_PinOpenDrainEnable Function

Enables the open-drain functionality for the selected pin.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_PinOpenDrainEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function enables the open-drain functionality for the selected pin.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
```

```
// MY_PINNUM - PORTS_PIN_10
SYS_PORTS_PinOpenDrainEnable( MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void SYS_PORTS_PinOpenDrainEnable (PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

SYS_PORTS_PinRead Function

Reads the selected digital pin.

Implementation: Dynamic

File

sys_ports.h

C

```
bool SYS_PORTS_PinRead(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

The status of the port pin.

Description

This function reads the selected digital pin, not the Latch.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_PIN_10
bool bitStatus = SYS_PORTS_PinRead(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

bool SYS_PORTS_PinRead (PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

SYS_PORTS_PinSet Function

Sets the selected digital pin/latch.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_PinSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function sets the selected digital pin/latch.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_PIN_10
SYS_PORTS_PinSet(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void SYS_PORTS_PinSet(PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

SYS_PORTS_PinToggle Function

Toggles the selected digital pin.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_PinToggle(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function toggles the selected digital pin.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
```

```
// MY_PINNUM - PORTS_PIN_10
SYS_PORTS_PinToggle( MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void SYS_PORTS_PinToggle (PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

SYS_PORTS_PinWrite Function

Writes the selected digital pin.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_PinWrite(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos, bool value);
```

Returns

None.

Description

This function writes the selected digital pin.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_PIN_10
SYS_PORTS_PinWrite(MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM, value);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS
value	Value to be written to the specific pin/latch: • true - Sets the bit • false - Clears the bit

Function

void SYS_PORTS_PinWrite (PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos
bool value)

SYS_PORTS_PinClear Function

Clears the selected digital pin.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_PinClear(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function clears the selected digital pin.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_IO_PIN_10
SYS_PORTS_PinClear( MY_PORTS_INSTANCE, MY_CHANNEL, MY_PINNUM );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void SYS_PORTS_PinClear (PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

SYS_PORTS_PinDirectionSelect Function

Enables the direction for the selected pin.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_PinDirectionSelect(PORTS_MODULE_ID index, SYS_PORTS_PIN_DIRECTION pinDir, PORTS_CHANNEL
channel, PORTS_BIT_POS bitPos);
```

Returns

None.

Description

This function enables the direction for the selected pin.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_PIN_10
SYS_PORTS_PIN_DIRECTION pinDir;
pinDir = SYS_PORTS_DIRECTION_INPUT;
SYS_PORTS_PinDirectionSelect(MY_PORTS_INSTANCE, pinDir, MY_CHANNEL, MY_PINNUM);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinDir	Pin direction
channel	Identifier for the PORT channel: A, B, C, etc.
bitPos	Possible values of PORTS_BIT_POS

Function

void SYS_PORTS_PinDirectionSelect (PORTS_MODULE_ID index,
SYS_PORTS_PIN_DIRECTION pinDir,
PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos)

b) Ports Control Functions

SYS_PORTS_Clear Function

Clears the selected digital port.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_Clear(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK clearMask);
```

Returns

None.

Description

This function clears the selected digital port.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
PORTS_DATA_MASK clearMask = (PORTS_DATA_MASK)0x00FF;
SYS_PORTS_Clear( MY_PORTS_INSTANCE, MY_CHANNEL, clearMask );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.

clearMask	Identifies the bits to be cleared

Function

void SYS_PORTS_Clear (PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK clearMask)

SYS_PORTS_DirectionGet Function

Reads the direction for the selected port.

Implementation: Dynamic

File

sys_ports.h

C

```
PORTS_DATA_MASK SYS_PORTS_DirectionGet(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

Direction of the port.

Description

This function reads the direction for the selected port.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
PORTS_DATA_MASK value;
value = SYS_PORTS_DirectionGet( MY_PORTS_INSTANCE, MY_CHANNEL );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.

Function

PORTS_DATA_MASK SYS_PORTS_DirectionGet(PORTS_MODULE_ID index, PORTS_CHANNEL channel)

SYS_PORTS_DirectionSelect Function

Enables the direction for the selected port.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_DirectionSelect(PORTS_MODULE_ID index, SYS_PORTS_PIN_DIRECTION pinDir, PORTS_CHANNEL
channel, PORTS_DATA_MASK mask);
```

Returns

None

Description

This function enables the direction for the selected port.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
SYS_PORTS_PIN_DIRECTION pinDir;
pinDir = SYS_PORTS_DIRECTION_INPUT;
PORTS_DATA_MASK myMask = (PORTS_DATA_MASK)0x00FF;
SYS_PORTS_DirectionSelect(MY_PORTS_INSTANCE, pinDir, MY_CHANNEL, myMask );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinDir	Pin direction
channel	Identifier for the PORT channel: A, B, C, etc.
mask	Mask for the direction of width PORTS_DATA_MASK

Function

void SYS_PORTS_DirectionSelect(PORTS_MODULE_ID index,
SYS_PORTS_PIN_DIRECTION pinDir,
PORTS_CHANNEL channel,
PORTS_DATA_MASK mask)

SYS_PORTS_OpenDrainDisable Function

Disables the open-drain functionality for the selected port.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_OpenDrainDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function disables the open-drain functionality for the selected port.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
SYS_PORTS_PeripheralOpenDrainDisable( MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
mask	Mask of type PORTS_DATA_MASK

Function

void SYS_PORTS_OpenDrainDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask)

SYS_PORTS_OpenDrainEnable Function

Enables the open-drain functionality for the selected port.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_OpenDrainEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK mask);
```

Returns

None.

Description

This function enables the open-drain functionality for the selected port.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
SYS_PORTS_OpenDrainEnable( MY_PORTS_INSTANCE, MY_CHANNEL, (PORTS_DATA_MASK)0x00FF );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
mask	Mask of type PORTS_DATA_MASK

Function

void SYS_PORTS_OpenDrainEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask)

SYS_PORTS_Read Function

Reads the data from the I/O port.

Implementation: Dynamic

File

sys_ports.h

C

```
PORTS_DATA_TYPE SYS_PORTS_Read(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Returns

Returns the data read from the port.

Description

This function reads the data from the I/O port.

Remarks

None.

Preconditions

The direction of the port to be set as input.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
PORTS_DATA_TYPE readData;
readData = SYS_PORTS_Read( MY_PORTS_INSTANCE, MY_CHANNEL );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.

Function

PORTS_DATA_TYPE SYS_PORTS_Read(PORTS_MODULE_ID index, PORTS_CHANNEL channel)

SYS_PORTS_Set Function

Sets the selected digital port/latch based on the mask.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_Set(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_TYPE value, PORTS_DATA_MASK
mask);
```

Returns

None.

Description

This function Sets the selected digital port/latch relative to the mask.

This function "AND" value and mask parameters and then set the bits in the port channel that were set in the result of the ANDing operation.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_VALUE - 0x1234
PORTS_DATA_MASK myMask = (PORTS_DATA_MASK)0x00FF;

// Set the MY_CHANNEL bit positions 2,4 and 5 (0x0034 = b0000 0000 0011 0100)
SYS_PORTS_Set(MY_PORTS_INSTANCE, MY_CHANNEL, MY_VALUE, myMask);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
value	Consists of information about which port bit has to be set
mask	Identifies the bits which could be intended for setting

Function

void SYS_PORTS_Set(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_TYPE value,
PORTS_DATA_MASK mask)

SYS_PORTS_Toggle Function

Toggles the selected digital port pins.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_Toggle(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK toggleMask);
```

Returns

None.

Description

This function toggles the selected digital port pins.

Remarks

None.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
PORTS_DATA_MASK toggleMask = (PORTS_DATA_MASK)0x00FF;
SYS_PORTS_Toggle( MY_PORTS_INSTANCE, MY_CHANNEL, toggleMask );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
toggleMask	Identifies the bits to be toggled

Function

void SYS_PORTS_Toggle(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK toggleMask)

SYS_PORTS_Write Function

Writes the data to the I/O port.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_Write(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_TYPE value);
```

Returns

None.

Description

This function writes the data to the I/O port.

Remarks

None.

Preconditions

The direction of the port to be set as output.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
PORTS_DATA_TYPE writeData;
SYS_PORTS_Write( MY_PORTS_INSTANCE, MY_CHANNEL, MY_VALUE );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
channel	Identifier for the PORT channel: A, B, C, etc.
value	Value to be written into a port of width PORTS_DATA_TYPE

Function

void SYS_PORTS_Write(PORTS_MODULE_ID index,
PORTS_CHANNEL channel,
PORTS_DATA_TYPE value)

SYS_PORTS_Initialize Function

Initializes PORT Pins/Channels.

Implementation: Static/Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_Initialize();
```

Returns

None.

Description

This function initializes different port pins/channels to the desired state. It also remaps the pins to the desired specific function.

Remarks

This API must be be called at the time of system initialization to initialize the ports pins.

Preconditions

None.

Example

```
SYS_PORTS_Initialize();
```

Function

void SYS_PORTS_Initialize()

c) Change Notification Functions

SYS_PORTS_ChangeNotificationDisable Function

Disables the change notification for the selected port.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_ChangeNotificationDisable(PORTS_MODULE_ID index, PORTS_CHANGE_NOTICE_PIN pinNum);
```

Returns

None.

Description

This function disables the change notification for the selected port.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
PORTS_CHANGE_NOTICE_PIN pinNum;
SYS_PORTS_ChangeNotificationDisable( index, pinNum );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Function

void SYS_PORTS_ChangeNotificationDisable(PORTS_MODULE_ID index,
PORTS_CHANGE_NOTICE_PIN pinNum)

SYS_PORTS_ChangeNotificationEnable Function

Enables the change notification for the selected port.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_ChangeNotificationEnable(PORTS_MODULE_ID index, PORTS_CHANGE_NOTICE_PIN pinNum,
SYS_PORTS_PULLUP_PULLDOWN_STATUS value);
```

Returns

None.

Description

This function enables the change notification for the selected port.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
SYS_PORTS_PULLUP_PULLDOWN_STATUS value;
```

```
PORTS_CHANGE_NOTICE_PIN pinNum;
SYS_PORTS_ChangeNotificationEnable( index, pinNum, value );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
value	Pull-up enable or disable value
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Function

void SYS_PORTS_ChangeNotificationEnable(PORTS_MODULE_ID index,
PORTS_CHANGE_NOTICE_PIN pinNum,
SYS_PORTS_PULLUP_PULLDOWN_STATUS value)

SYS_PORTS_ChangeNotificationGlobalDisable Function

Globally disables the change notification for the selected port.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_ChangeNotificationGlobalDisable(PORTS_MODULE_ID index);
```

Returns

None.

Description

This function globally disables the change notification for the selected port.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
SYS_PORTS_ChangeNotificationGlobalDisable( MY_PORTS_INSTANCE);
```

Function

void SYS_PORTS_ChangeNotificationGlobalDisable(PORTS_MODULE_ID index);

SYS_PORTS_ChangeNotificationGlobalEnable Function

Globally enables the change notification for the selected port.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_ChangeNotificationGlobalEnable(PORTS_MODULE_ID index);
```

Returns

None.

Description

This function globally enables the change notification for the selected port.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
SYS_PORTS_ChangeNotificationGlobalEnable( MY_PORTS_INSTANCE );
```

Function

void SYS_PORTS_ChangeNotificationGlobalEnable(PORTS_MODULE_ID index)

SYS_PORTS_ChangeNotificationInIdleModeDisable Function

Disables the change notification for the selected port in Sleep or Idle mode.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_ChangeNotificationInIdleModeDisable(PORTS_MODULE_ID index);
```

Returns

None.

Description

This function disables the change notification for the selected port in Sleep or Idle mode.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
SYS_PORTS_ChangeNotificationInIdleModeDisable( MY_PORTS_INSTANCE );
```

Function

void SYS_PORTS_ChangeNotificationInIdleModeDisable(PORTS_MODULE_ID index);

SYS_PORTS_ChangeNotificationInIdleModeEnable Function

Enables the change notification for the selected port in Sleep or Idle mode.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_ChangeNotificationInIdleModeEnable(PORTS_MODULE_ID index);
```

Returns

None.

Description

This function enables the change notification for the selected port in Sleep or Idle mode.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
SYS_PORTS_ChangeNotificationInIdleModeEnable( MY_PORTS_INSTANCE );
```

Function

void SYS_PORTS_ChangeNotificationInIdleModeEnable(PORTS_MODULE_ID index);

SYS_PORTS_ChangeNotificationPullUpDisable Function

Disables a weak pull-up on the change notification pin.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_ChangeNotificationPullUpDisable(PORTS_MODULE_ID index, PORTS_CHANGE_NOTICE_PIN pinNum);
```

Returns

None.

Description

This function Disables a weak pull-up on the change notification pin.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_CHANGE_NOTICE_PIN_10
SYS_PORTS_ChangeNotificationPullUpDisable( MY_PORTS_INSTANCE, MY_PINNUM );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Function

void SYS_PORTS_ChangeNotificationPullUpDisable (PORTS_MODULE_ID index,
PORTS_CHANGE_NOTICE_PIN pinNum)

SYS_PORTS_ChangeNotificationPullUpEnable Function

Enables a weak pull-up on the change notification pin.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_ChangeNotificationPullUpEnable(PORTS_MODULE_ID index, PORTS_CHANGE_NOTICE_PIN pinNum);
```

Returns

None.

Description

This function enables a weak pull-up on the change notification pin.

Remarks

Not all features are available on all devices. Refer to the specific device data sheet for availability.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// MY_PINNUM - PORTS_CHANGE_NOTICE_PIN_10
SYS_PORTS_ChangeNotificationPullUpEnable( MY_PORTS_INSTANCE, MY_PINNUM );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured
pinNum	Possible values of PORTS_CHANGE_NOTICE_PIN

Function

void SYS_PORTS_ChangeNotificationPullUpEnable (PORTS_MODULE_ID index,
PORTS_CHANGE_NOTICE_PIN pinNum)

d) Peripheral Pin Select Functions

SYS_PORTS_RemapInput Function

Input/Output (I/O) function remapping.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_RemapInput(PORTS_MODULE_ID index, PORTS_REMAP_INPUT_FUNCTION function, PORTS_REMAP_INPUT_PIN
remapPin);
```

Returns

None.

Description

This function controls the I/O function remapping.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use SYS_PORTS_ExistsRemapInputOutput in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
```

```
// Remapping input function 'Input Capture 1' to the Remappable pin 'RPD2'
SYS_PORTS_RemapInput(MY_PORTS_INSTANCE, INPUT_FUNC_IC1, INPUT_PIN_RPD2 );
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void SYS_PORTS_RemapInput(PORTS_MODULE_ID index,
PORTS_REMAP_INPUT_FUNCTION function,
PORTS_REMAP_INPUT_PIN remapPin)

One of the possible values of PORTS_REMAP_INPUT_FUNCTION

remapPin - One of the possible values of PORTS_REMAP_INPUT_PIN

SYS_PORTS_RemapOutput Function

Input/Output (I/O) function remapping.

Implementation: Dynamic

File

sys_ports.h

C

```
void SYS_PORTS_RemapOutput(PORTS_MODULE_ID index, PORTS_REMAP_OUTPUT_FUNCTION function,
PORTS_REMAP_OUTPUT_PIN remapPin);
```

Returns

None.

Description

This function controls the I/O function remapping.

Remarks

This feature may not be available on all devices. Please refer to the specific device data sheet to determine availability or use SYS_PORTS_ExistsRemapInputOutput in your application to determine whether this feature is available.

Preconditions

None.

Example

```
// Where MY_PORTS_INSTANCE, is the ports instance selected for use by the
// application developer.
// Remapping output function 'UART3 Transmit' to the Remappable pin 'RPA14'
SYS_PORTS_RemapInputOutput(MY_PORTS_INSTANCE, OUTPUT_FUNC_U3TX, OUTPUT_PIN_RPA14);
```

Parameters

Parameters	Description
index	Identifier for the device instance to be configured

Function

void SYS_PORTS_RemapOutput(PORTS_MODULE_ID index,
PORTS_REMAP_OUTPUT_FUNCTION function,
PORTS_REMAP_OUTPUT_PIN remapPin)

One of the possible values of PORTS_REMAP_OUTPUT_FUNCTION

remapPin - One of the possible values of PORTS_REMAP_OUTPUT_PIN

e) Data Types and Constants

SYS_PORTS_PIN_DIRECTION Enumeration

Defines the direction of the port pins.

File

sys_ports.h

C

```
typedef enum {
  SYS_PORTS_DIRECTION_OUTPUT,
  SYS_PORTS_DIRECTION_INPUT
} SYS_PORTS_PIN_DIRECTION;
```

Members

Members	Description
SYS_PORTS_DIRECTION_OUTPUT	Direction as output
SYS_PORTS_DIRECTION_INPUT	Direction as input

Description

SYS PORTS PIN DIRECTION

These constants provide the port pin direction definitions.

Remarks

None.

SYS_PORTS_PULLUP_PULLDOWN_STATUS Enumeration

Provides the pull-up and pull-down status.

File

sys_ports.h

C

```
typedef enum {
  SYS_PORTS_PULLUP_DISABLE,
  SYS_PORTS_PULLUP_ENABLE
} SYS_PORTS_PULLUP_PULLDOWN_STATUS;
```

Members

Members	Description
SYS_PORTS_PULLUP_DISABLE	PULLUP Disable
SYS_PORTS_PULLUP_ENABLE	PULLUP Enable

Description

SYS PORTS PULLUP status

These constants provide the pull-up or pull-down status definitions.

Remarks

None.

Files

Files

Name	Description
sys_ports.h	Ports System Service interface definitions

Description

This section lists the source and header files used by the library.

sys_ports.h

Ports System Service interface definitions

Enumerations

	Name	Description
	SYS_PORTS_PIN_DIRECTION	Defines the direction of the port pins.
	SYS_PORTS_PULLUP_PULLDOWN_STATUS	Provides the pull-up and pull-down status.

Functions

	Name	Description
	SYS_PORTS_ChangeNotificationDisable	Disables the change notification for the selected port. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationEnable	Enables the change notification for the selected port. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationGlobalDisable	Globally disables the change notification for the selected port. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationGlobalEnable	Globally enables the change notification for the selected port. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationInIdleModeDisable	Disables the change notification for the selected port in Sleep or Idle mode. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationInIdleModeEnable	Enables the change notification for the selected port in Sleep or Idle mode. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationPullUpDisable	Disables a weak pull-up on the change notification pin. **Implementation:** Dynamic
	SYS_PORTS_ChangeNotificationPullUpEnable	Enables a weak pull-up on the change notification pin. **Implementation:** Dynamic
	SYS_PORTS_Clear	Clears the selected digital port. **Implementation:** Dynamic
	SYS_PORTS_DirectionGet	Reads the direction for the selected port. **Implementation:** Dynamic
	SYS_PORTS_DirectionSelect	Enables the direction for the selected port. **Implementation:** Dynamic
	SYS_PORTS_Initialize	Initializes PORT Pins/Channels. **Implementation:** Static/Dynamic
	SYS_PORTS_OpenDrainDisable	Disables the open-drain functionality for the selected port. **Implementation:** Dynamic
	SYS_PORTS_OpenDrainEnable	Enables the open-drain functionality for the selected port. **Implementation:** Dynamic
	SYS_PORTS_PinClear	Clears the selected digital pin. **Implementation:** Dynamic
	SYS_PORTS_PinDirectionSelect	Enables the direction for the selected pin. **Implementation:** Dynamic
	SYS_PORTS_PinModeSelect	Enables the selected pin as analog or digital. **Implementation:** Dynamic
	SYS_PORTS_PinOpenDrainDisable	Disables the open-drain functionality for the selected pin. **Implementation:** Dynamic
	SYS_PORTS_PinOpenDrainEnable	Enables the open-drain functionality for the selected pin. **Implementation:** Dynamic
	SYS_PORTS_PinRead	Reads the selected digital pin. **Implementation:** Dynamic
	SYS_PORTS_PinSet	Sets the selected digital pin/latch. **Implementation:** Dynamic
	SYS_PORTS_PinToggle	Toggles the selected digital pin. **Implementation:** Dynamic
	SYS_PORTS_PinWrite	Writes the selected digital pin. **Implementation:** Dynamic

	SYS_PORTS_Read	Reads the data from the I/O port. **Implementation:** Dynamic
	SYS_PORTS_RemapInput	Input/Output (I/O) function remapping. **Implementation:** Dynamic
	SYS_PORTS_RemapOutput	Input/Output (I/O) function remapping. **Implementation:** Dynamic
	SYS_PORTS_Set	Sets the selected digital port/latch based on the mask. **Implementation:** Dynamic
	SYS_PORTS_Toggle	Toggles the selected digital port pins. **Implementation:** Dynamic
	SYS_PORTS_Write	Writes the data to the I/O port. **Implementation:** Dynamic

Description

Ports System Service Interface Definition

This file contains the interface definition for the Ports system service. It provides a way to interact with the Ports subsystem to manage the timing requests supported by the system

File Name

sys_ports.h

Company

Microchip Technology Inc.

Random Number Generator (RNG) System Service Library

This section describes the Random Number Generator (RNG) System Service Library.

Introduction

This library provides a Random Number Generator (RNG) service with a convenient C language interface.

Description

The Random Number Generator (RNG) System Service provides a simple C-language interface for generating random numbers at both pseudo-random and cryptographic strengths.

Using the Library

This topic describes the basic architecture of the RNG System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_random.h`

The interface to the RNG System Service library is defined in the `sys_random.h` header file.

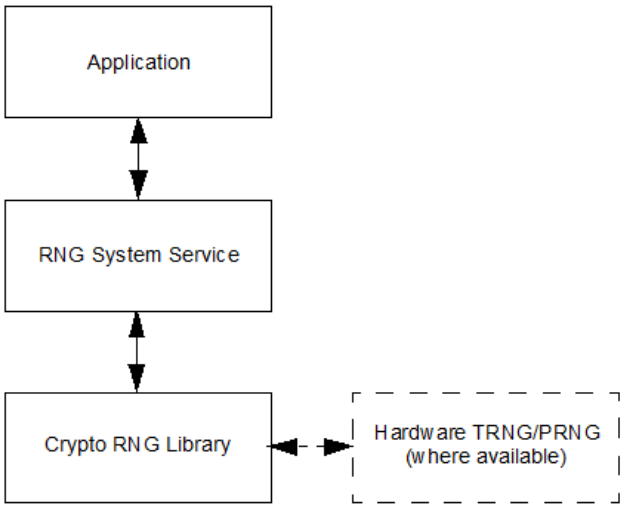
Please refer to the What is MPLAB Harmony? section for how the RNG System Service interacts with the framework.

Abstraction Model

This topic provides a description of the software abstraction for the RNG System Service.

Description

This model explains how the system interfaces with the RNG System Service and the application, as illustrated in the following diagram.



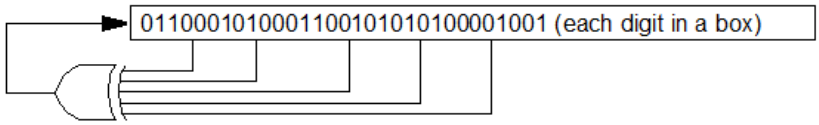
Random Numbers

Random numbers are often desired in systems in order to provide a change in the system that cannot necessarily be predicted. For example, a game might use random numbers for a dice roll or a playing card shuffle. Another example is when a collision occurs in an Ethernet network. A delay is inserted in the next attempt to communicate on the network, and the length of the delay is determined by a random number.

Pseudo Random Numbers (PRN)

Due to the deterministic nature of computer systems, it is actually very difficult to produce true randomness in number generation. For many systems, a pseudo-random number is sufficient. Typically, the pseudo-random number is generated via a Linear Feedback Shift Register (LFSR). The LFSR takes a seed value, and generates a new number typically by having taps on the LFSR in different places and XORing the bits together. This new bit is appended to the beginning of the number after the original number has been shifted one bit to the right.

And example of a LFSR is shown in the following figure.



Cryptographically-secure Pseudo Random Numbers (CSPRN)

Because pseudo-random numbers can be predicted if an attacker knows the setup of the LFSR, it is desired in certain applications to have a way of minimizing the predictability of the random numbers. To accomplish this, a sub-set of PRNGs are run through statistical tests to determine their security in cryptographic applications. A Cryptographically-secure Pseudo Random Number generator is one that has passed all such statistical tests.

True Random Number Generator (TRNG)

To provide seed values for the PRNG, and in applications that require true randomness, there are hardware options for a true random number generator (TRNG). These generators use ring oscillators and the inherent thermal noise within silicon to create the individual bits of a number. Because of the nature of these oscillators, it is not possible to predict the generated numbers.

Library Overview

Please refer to the System Service Introduction for a detailed description of MPLAB Harmony system services.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Random Number Generator module.

Library Interface Section	Description
Random Number Generator Functions	Provides system module APIs. Device initialization, deinitialization, reinitialization and status functions.
Data Types and Constants	Device-specific enumerations, structures, and constants.

How the Library Works

Describes how the RNG Library works.

Description

Service Initialization

To begin using the Random Number System Service, it is necessary to initialize the service using the SYS_RANDOM_Initialize function.

Example:

```
// Initialization data for the RNG
SYS_RANDOM_INIT rngInit = { 0x12345678, NULL, 32 };
// Initialize the System Random Service
SYS_RANDOM_Initialize( 0, rngInit );
```

Pseudo-Random Numbers

The Pseudo-Random Number functions use the C-language functions seed() and rand() to provide the pseudo-random numbers.

Example:

```
// Seed the PRNG
SYS_RANDOM_PseudoSeedSet(0x876543210);
// Get a Pseudo-random Number
uint32_t myRandomNum = SYS_RANDOM_PseudoGet();
```

Cryptographic Pseudo-Random Numbers

The Cryptographic Random Number functions use the Crypto RNG library functions to seed and generate numbers. On PIC32 devices with a PRNG/TRNG module, it will use the hardware to generate the numbers.

Example:

```
// Get a 32-bit Crypto Random Number
uint32_t myRandomNum = SYS_RANDOM_CryptoGet();
// Fill an array of random numbers
uint32_t myRandomNumArray[10];
SYS_RANDOM_CryptoBlockGet(myRandomNumArray, 10);
// Get a 1-byte Random Number
uint8_t myRandomByte = SYS_RANDOM_CryptoByteGet();
```

Configuring the Library

Macros

	Name	Description
	SYS_RANDOM_CRYPTO_SEED_SIZE	

Description

The configuration of the RNG System Service is based on the file `system_config.h`.

This header file contains the configuration selection for the Random Number Generator System Service. Based on the selections made, the Random Number Generator System Service may support the selected features. These configuration settings will apply to all instances of the Random Number Generator System Service.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

SYS_RANDOM_CRYPTO_SEED_SIZE Macro

File

sys_random_config_template.h

C

```
#define SYS_RANDOM_CRYPTO_SEED_SIZE (256/8)
```

Description

Cryptographic Quality Seed Value

Building the Library

This section lists the files that are available in the Random Number Generator System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/random`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`sys_random.h`	Random Number Generator System Service Library API header file.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/sys_random.c`	Random Number Generator System Service Library implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this system service.

Module Dependencies

The Random Number Generator System Service is dependent upon the following modules:

- Cryptographic (Crypto) Library

Library Interface

a) Random Number Generator Functions

	Name	Description
	SYS_RANDOM_CryptoBlockGet	Generates a sequence of random bytes using the cryptographic quality Random Number Generator.
	SYS_RANDOM_CryptoByteGet	Returns a random byte from the cryptographic quality Random Number Generator.
	SYS_RANDOM_CryptoEntropyAdd	Adds randomness to the cryptographic quality Random Number Generator.
	SYS_RANDOM_CryptoGet	Returns a random 32 bit value from the cryptographic quality Random Number Generator.
	SYS_RANDOM_CryptoSeedSet	Reseeds the cryptographic quality Random Number Generator.
	SYS_RANDOM_CryptoSeedSizeGet	Reseeds the cryptographic quality Random Number Generator.
	SYS_RANDOM_Deinitialize	Deinitializes the Random Number Generator system service.
	SYS_RANDOM_Initialize	Initializes the Random Number Generator system service.
	SYS_RANDOM_PseudoGet	Returns a random value from the Pseudo-random Number Generator.
	SYS_RANDOM_PseudoSeedSet	Reseeds the Pseudo-random Number Generator.

b) Data Types and Constants

	Name	Description
	SYS_RANDOM_INIT	Random Number Generator Service Initialization Data

Description

This section describes the APIs of the RNG System Service Library.

Refer to each section for a detailed description.

a) Random Number Generator Functions

SYS_RANDOM_CryptoBlockGet Function

Generates a sequence of random bytes using the cryptographic quality Random Number Generator.

File

sys_random.h

C

```
void SYS_RANDOM_CryptoBlockGet(void * buffer, size_t size);
```

Returns

None.

Description

This function uses the Cryptographic-quality Random Number Generator to fill a block of data with random numbers.

Remarks

When run, it will fill the memory starting at buffer with size bytes of random numbers.

Preconditions

System service must be initialized by calling SYS_RANDOM_Initialize before this function is called.

Example

```
#define RNG_DATA_SIZE   32

uint8_t randomData[RNG_DATA_SIZE];
SYS_MODULE_OBJ rngObject;

rngObject = SYS_RANDOM_Initialize();
...
SYS_RANDOM_CryptoBlockGet((void *)randomData, RNG_DATA_SIZE);
```

Parameters

Parameters	Description
buffer	Pointer to the memory location to fill with random data.
size	The amount of random data, in bytes, to put in memory.

Function

void SYS_RANDOM_CryptoBlockGet(uint8_t buffer, size_t bufferSize);

SYS_RANDOM_CryptoByteGet Function

Returns a random byte from the cryptographic quality Random Number Generator.

File

sys_random.h

C

```
uint8_t SYS_RANDOM_CryptoByteGet();
```

Returns

Returns one byte of cryptographic-quality random data.

Description

This function returns a single byte from the Cryptographic-quality Random Number Generator.

Remarks

None.

Preconditions

System service must be initialized by calling SYS_RANDOM_Initialize before this function is called.

Example

```
uint8_t randomData;
SYS_MODULE_OBJ rngObject;

rngObject = SYS_RANDOM_Initialize();
randomData = SYS_RANDOM_CryptoByteGet();
```

Function

uint8_t SYS_RANDOM_CryptoByteGet(void);

SYS_RANDOM_CryptoEntropyAdd Function

Adds randomness to the cryptographic quality Random Number Generator.

File

sys_random.h

C

```
void SYS_RANDOM_CryptoEntropyAdd(uint8_t data);
```

Returns

None.

Description

This function adds entropy to the Cryptographic-quality Random Number Generator in order to adjust the randomness of the generated numbers.

Remarks

This function currently does not function.

Preconditions

None.

Example

```
SYS_MODULE_OBJ rngObject;

rngObject = SYS_RANDOM_Initialize();
SYS_RANDOM_CryptoEntropyAdd(23);
```

Parameters

Parameters	Description
data	8-bit value to add to the entropy calculation.

Function

void SYS_RANDOM_CryptoEntropyAdd(uint8_t data);

SYS_RANDOM_CryptoGet Function

Returns a random 32 bit value from the cryptographic quality Random Number Generator.

File

sys_random.h

C

```
uint32_t SYS_RANDOM_CryptoGet();
```

Returns

32-bit random number.

Description

This function returns a 32-bit random number from the Cryptographic- quality Random Number Generator.

Remarks

On PIC32MZ devices with a hardware Random Number Generator, this function returns a value from that generator. Other devices will use a software library to generate the random number.

Preconditions

System service must be initialized by calling SYS_RANDOM_Initialize before this function is called.

Example

```
SYS_MODULE_OBJ rngObject;
uint32_t rngData;

rngObject = SYS_RANDOM_Initialize();
...
rngData = SYS_RANDOM_CryptoGet();
```

Function

uint32_t SYS_RANDOM_CryptoGet(void);

SYS_RANDOM_CryptoSeedSet Function

Reseeds the cryptographic quality Random Number Generator.

File

sys_random.h

C

```
void SYS_RANDOM_CryptoSeedSet(void * seed, size_t size);
```

Returns

None.

Description

This function inserts a seed value into the Cryptographic-quality Random Number Generator. The value will be used to generate the next random number.

Remarks

None.

Preconditions

System service must be initialized by calling SYS_RANDOM_Initialize before this function is called.

Example

```
SYS_MODULE_OBJ rngObject;
uint8_t rngSeed[SYS_RANDOM_CRYPTO_SEED_SIZE];

rngObject = SYS_RANDOM_Initialize();
// Fill the rngSeed
SYS_RANDOM_CryptoSeedSet(rngSeed, sizeof(rngSeed));
```

Parameters

Parameters	Description
seed	Pointer to the memory location containing the new seed.
size	Must be less or equal to SYS_RANDOM_CRYPTO_SEED_SIZE

Function

void SYS_RANDOM_CryptoSeedSet(void *seed, size_t size);

SYS_RANDOM_CryptoSeedSizeGet Function

Reseeds the cryptographic quality Random Number Generator.

File

sys_random.h

C

```
size_t SYS_RANDOM_CryptoSeedSizeGet();
```

Returns

Returns the size (in bytes) of the most recently set seed value for the cryptographic strength random number generator.

Description

This function returns the current size of the seed used in the Cryptographic- quality Random Number Generator.

Remarks

None.

Preconditions

System service must be initialized by calling SYS_RANDOM_Initialize before this function is called.

Example

```
SYS_MODULE_OBJ rngObject;
size_t seedSize;

rngObject = SYS_RANDOM_Initialize();
...
seedSize = SYS_RANDOM_CryptoSeedSizeGet();
```

Function

size_t SYS_RANDOM_CryptoSeedSizeGet(void);

SYS_RANDOM_Deinitialize Function

Deinitializes the Random Number Generator system service.

File

sys_random.h

C

```
void SYS_RANDOM_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the Random Number Generator system service.

Remarks

None.

Preconditions

System service must be initialized by calling SYS_RANDOM_Initialize before this function is called.

Example

```
SYS_MODULE_OBJ rngObject;

rngObject = SYS_RANDOM_Initialize();
```

Parameters

Parameters	Description
object	SYS RANDOM object handle, returned from SYS_RANDOM_Initialize

Function

void SYS_RANDOM_Deinitialize(SYS_MODULE_OBJ object);

SYS_RANDOM_Initialize Function

Initializes the Random Number Generator system service.

File

sys_random.h

C

```
SYS_MODULE_OBJ SYS_RANDOM_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to an object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes the Random Number System service, including seeding the Pseudo Random Number Generator, the Crypto RNG library, and setting the seed size.

Remarks

None.

Preconditions

None.

Example

```
SYS_MODULE_OBJ rngObject;

rngObject = SYS_RANDOM_Initialize();
SYS_RANDOM_Deinitialize(rngObject);
```

Parameters

Parameters	Description
index	Module instance index.
init	initialization data for the random system service (cast of a pointer to a SYS_RANDOM_INIT structure to a SYS_MODULE_INDEX structure pointer).

Function

SYS_MODULE_OBJ SYS_RANDOM_Initialize(const SYS_RANDOM_INIT* const randomInit);

SYS_RANDOM_PseudoGet Function

Returns a random value from the Pseudo-random Number Generator.

File

sys_random.h

C

```
uint32_t SYS_RANDOM_PseudoGet();
```

Returns

A 32-bit pseudo-random number.

Description

This function returns one 32-bit Pseudo-random Number.

Remarks

None.

Preconditions

None.

Example

```
uint32_t rngNumber;

rngNumber = SYS_RANDOM_PseudoGet();
```

Function

uint32_t SYS_RANDOM_PseudoGet(void);

SYS_RANDOM_PseudoSeedSet Function

Reseeds the Pseudo-random Number Generator.

File

sys_random.h

C

```
void SYS_RANDOM_PseudoSeedSet(uint32_t seed);
```

Returns

None.

Description

This function inserts a seed value into the Pseudo-Random Number Generator. The value will be used to generate the next random number.

Preconditions

None.

Example

```
uint32_t rngSeed = 0x12345678;

SYS_RANDOM_PseudoSeedSet(rngSeed);
Remarks:   None.
```

Parameters

Parameters	Description
seed	32-bit value to use as the seed for the Pseudo RNG.

Function

void SYS_RANDOM_PseudoSeedSet(uint32_t seed);

b) Data Types and Constants

SYS_RANDOM_INIT Structure

File

sys_random.h

C

```
typedef struct {
  uint32_t seedPseudo;
  void * seedCrypto;
  size_t seedCryptoSize;
} SYS_RANDOM_INIT;
```

Description

Random Number Generator Service Initialization Data

Files

Files

Name	Description
sys_random.h	Random Number Generator System Service interface definition.
sys_random_config_template.h	Configuration options template file for the Random Number Generator system service.

Description

This section lists the source and header files used by the library.

sys_random.h

Random Number Generator System Service interface definition.

Functions

	Name	Description
	SYS_RANDOM_CryptoBlockGet	Generates a sequence of random bytes using the cryptographic quality Random Number Generator.
	SYS_RANDOM_CryptoByteGet	Returns a random byte from the cryptographic quality Random Number Generator.
	SYS_RANDOM_CryptoEntropyAdd	Adds randomness to the cryptographic quality Random Number Generator.
	SYS_RANDOM_CryptoGet	Returns a random 32 bit value from the cryptographic quality Random Number Generator.
	SYS_RANDOM_CryptoSeedSet	Reseeds the cryptographic quality Random Number Generator.
	SYS_RANDOM_CryptoSeedSizeGet	Reseeds the cryptographic quality Random Number Generator.
	SYS_RANDOM_Deinitialize	Deinitializes the Random Number Generator system service.
	SYS_RANDOM_Initialize	Initializes the Random Number Generator system service.
	SYS_RANDOM_PseudoGet	Returns a random value from the Pseudo-random Number Generator.
	SYS_RANDOM_PseudoSeedSet	Reseeds the Pseudo-random Number Generator.

Structures

	Name	Description
	SYS_RANDOM_INIT	Random Number Generator Service Initialization Data

Description

Random Number Generator (RNG) System Service Interface Definition

This file contains the interface definition for the Random Number Generator System Service.

File Name

sys_random.h

Company

Microchip Technology Inc.

sys_random_config_template.h

Configuration options template file for the Random Number Generator system service.

Macros

	Name	Description
	SYS_RANDOM_CRYPTO_SEED_SIZE	

Description

Random Number Generator System Service Configuration Definitions Template

This file contains example definitions of configuration options for the random number generator system service. It is for documentation purposes only. Do not include it directly source files.

File Name

sys_random_config_template.h

Company

Microchip Technology Inc.

Reset System Service Library

This section describes the Reset System Service Library.

Introduction

This library provides an interface to manage the Reset module on the Microchip family of microcontrollers during different modes of operation.

Description

The Reset System Service Library is a MPLAB Harmony system service that can be used to identify the reason for the most recent processor reset, manage the system's reset reason status flags, and even cause a reset under software control if the processor in use supports that functionality.

Using the Library

This topic describes the basic architecture of the Reset System Service and provides information and examples on its use.

Description

Interface Header File: `sys_reset.h`

The interface to the Clock System Service is defined in the `sys_reset.h` header file, which is included by the `sys.h` header file.

Any C language source (`.c`) file that uses the Reset System Service must include `sys.h`.

Abstraction Model

This topic describes the basic abstraction model and functionality of the Reset System Service.

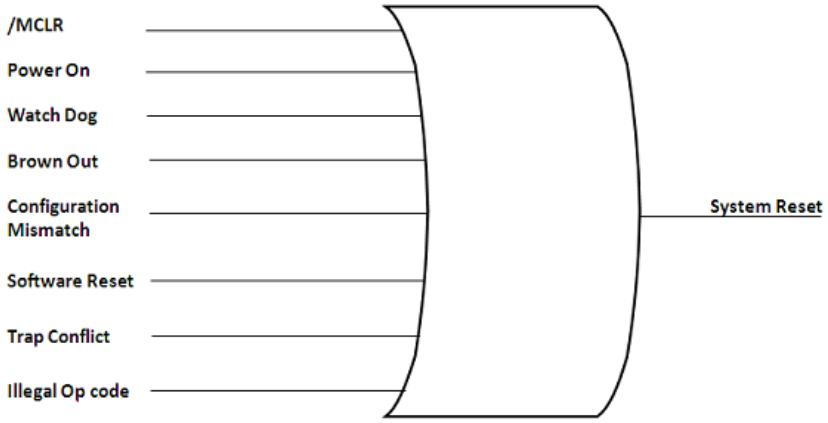
Description

The Reset System Service Library provides the following functionality:

- Microcontroller Reset Reason Management
- Triggering Software Reset

When one of the sources described in the following section asserts a reset request or the software requests a reset using this library, the microcontroller will be immediately reset.

The following figure illustrates the hardware abstraction model for the reset system service,



Reset Sources

- **/MCLR:** Whenever the /MCLR pin is driven low, the device asynchronously asserts reset. The processor continues to use the existing clock source that was in use before the /MCLR Reset occurred.
- **Power On [POR]:** A power-on event generates an internal power-on reset pulse when a VDD rise is detected above POR voltage. In particular, VDD must fall below POR voltage before a new POR is initiated.
- **Watch Dog Timer [WDT]:** Whenever a watchdog timer time-out occurs, the device asynchronously asserts system reset. The clock source

remains unchanged. Note that a WDT time-out during Sleep or Idle mode will wake-up the processor, but will not reset the processor.

- **Brown Out:** If the voltage supplied to the internal regulator is inadequate to maintain a regulated level, the regulator reset circuitry will generate a Brown-Out Reset (BOR). Brown out reset is applicable only when the regulator is enabled.
- **Configuration Mismatch:** To maintain the integrity of the stored configuration values, all device configuration bits are implemented as a complementary set of register bits. For each bit, as the actual value of the register is written as '1', a complementary value, '0', is stored in its corresponding background register and vice versa. The bit pairs are compared every time, including sleep mode. During this comparison, if the configuration bit values are not found opposite to each other, a configuration mismatch event is generated which causes a device reset.
- **Software Reset:** Whenever the reset instruction or a reset sequence is executed, the device asserts system reset. This reset state does not reinitialize the clock. The clock source that is in effect prior to the reset instruction or reset sequence execution remains in effect.
- **Trap Conflict:** A trap conflict reset occurs when a hard and a soft trap occur at the same time.
- **Illegal Opcode:** A device reset is generated if the device attempts to execute an illegal opcode value that was fetched from program memory.

Note: For more information on the electrical specifications, refer to the "**Electrical Characteristics"** chapter of the specific device data sheet.

Library Overview

Please refer to the System Service Introduction for a detailed description of MPLAB Harmony system services.

The library interface routines are divided into the following sub-sections, which address one of the blocks or the overall operation of the Reset System Service Library.

Library Interface Section	Description
Functions	Provides functions to read the reset reason and clear the respective status, to trigger the software reset, and reset the NMI delay count.

How the Library Works

Reset Reason Management

Describes how to use the library to identify and clear the reason a reset occurred.

Description

Reset Reason

Other software modules may identify the reason for the most recent system reset using the SYS_RESET_ReasonGet function. After obtaining the reset reason, its status flag can be cleared using the SYS_RESET_ReasonClear function.

Example: Identifying and Clearing Reset Reason

```
RESET_REASON reasonType;

reasonType = SYS_RESET_ReasonGet();
if( reasonType == RESET_REASON_MCLR )
{
    // Take some reset reason specific action
    // ...

    // Clear the reset reason status flag
    SYS_RESET_ReasonClear( RESET_REASON_MCLR );
}
```

Example: Clearing Multiple Reset Reasons

```
// Clear multiple reason flags
SYS_RESET_ReasonClear( RESET_REASON_MCLR|RESET_REASON_POWERON );

// Clear all current reason flags
SYS_RESET_ReasonClear( RESET_REASON_ALL );
```

Note: Not every reset reason is supported by every device, refer to the device specific information for more information.

Triggering a Software Reset

Describes how to use the Reset System Service Library to cause a system reset.

Description

Software Reset Trigger

Software can trigger the a system reset by using the SYS_RESET_SoftwareReset function.

Example

```
SYS_RESET_SoftwareReset();
```

Note: On devices that support software reset functionality, this routine never returns. Execution continues from the reset vector, following the reset. On devices that do not support software reset, calling this routine will give a build warning and execution will continue immediately following this function.

Configuring the Library

Describes how to configure the system reset library.

Description

The configuration of the Reset System Service is affected by the following:

- Processor Selection
- System Configuration

Processor Selection

Different processors support different reset reasons. Not all reset reasons are defined for all processors. Refer to the specific device data sheet or family reference manual for the processor in use to identify which reset reasons it supports.

System Configuration

Many MPLAB Harmony libraries provide configuration options that can be selected by defining their desired values at build-time using the C-language preprocessor "#define" statement. To obtain these definitions, MPLAB Harmony libraries include a file named "system_config.h" that must be defined as part of the over-all system configuration and placed in the compiler's include file search path.

Refer to the Applications Help section for more details.

System Reset Library Configuration Options

This library does not support any build-time configuration items.

Building the Library

This section lists the files that are available in the Reset System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/reset`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
sys_reset.h	Reset System Service Library API header file.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/sys_reset.c	Reset System Service Library implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The Reset System Service Library depends on the following modules:

- Reset Peripheral Library
- Device Control Peripheral Library

Library Interface

a) Functions

	Name	Description
	SYS_RESET_ReasonClear	Clears the status flag for the specified reset reason.
	SYS_RESET_ReasonGet	Returns the reason for the reset.
	SYS_RESET_SoftwareReset	Triggers a software reset.
	SYS_RESET_NMIDelayCountSet	Sets the Delay for WDT/DMT NMI reset events.

Description

This section describes the APIs of the Reset System Service Library.

Refer to each section for a detailed description.

a) Functions

SYS_RESET_ReasonClear Function

Clears the status flag for the specified reset reason.

File

sys_reset.h

C

```
void SYS_RESET_ReasonClear(RESET_REASON reason);
```

Returns

None.

Description

This function clears the specified status flag(s) that were previously set by the processor to identify the reason for the most recent reset.

Remarks

None.

Preconditions

Hardware should have set the reset reason.

Example

```
// To clear a single reset reason,
SYS_RESET_ReasonClear( RESET_REASON_MCLR );

// To clear more than one reason,
SYS_RESET_ReasonClear( RESET_REASON_MCLR|RESET_REASON_POWERON );
```

```
// To clear all the reasons,
SYS_RESET_ReasonClear( RESET_REASON_ALL );
```

Parameters

Parameters	Description
reason	One of the possible values of the enum RESET_REASON

Function

void SYS_RESET_ReasonClear(RESET_REASON reason)

SYS_RESET_ReasonGet Function

Returns the reason for the reset.

File

sys_reset.h

C

```
RESET_REASON SYS_RESET_ReasonGet();
```

Returns

A value identifying the reason for the most recent reset.

Description

This function identifies the reason that the most recent reset occurred.

Remarks

None.

Preconditions

None.

Example

```
RESET_REASON reasonType;
reasonType = SYS_RESET_ReasonGet();
```

Function

RESET_REASON SYS_RESET_ReasonGet(void)

SYS_RESET_SoftwareReset Function

Triggers a software reset.

File

sys_reset.h

C

```
void SYS_RESET_SoftwareReset();
```

Returns

None.

Description

This function triggers a processor.

Remarks

The software reset feature must be supported by the processor in use.

Preconditions

None.

Example

```
SYS_RESET_SoftwareReset();
```

Function

void SYS_RESET_SoftwareReset(void)

SYS_RESET_NMIDelayCountSet Function

Sets the Delay for WDT/DMT NMI reset events.

File

sys_reset.h

C

```
void SYS_RESET_NMIDelayCountSet(RESET_NMI_COUNT_TYPE nmi_count);
```

Returns

None

Description

This function sets the NMI counter value, that is equivalent to the number of SYSCLK cycles needed for the device Reset to occur when a WDT/DMT NMI event is triggered. This NMI reset counter is only applicable to these two specific NMI events only.

Remarks

The NMI feature must be supported by the processor in use.

Preconditions

None.

Example

```
SYS_RESET_NMIDelayCountSet( 0x54 );
```

Parameters

Parameters	Description
nmi_count	NMI counter value.

Function

void SYS_RESET_NMIDelayCountSet(RESET_NMI_COUNT_TYPE nmi_count)

Files

Files

Name	Description
sys_reset.h	Reset System Service interface definition.

Description

sys_reset.h

Reset System Service interface definition.

Functions

	Name	Description
	SYS_RESET_NMIDelayCountSet	Sets the Delay for WDT/DMT NMI reset events.
	SYS_RESET_ReasonClear	Clears the status flag for the specified reset reason.
	SYS_RESET_ReasonGet	Returns the reason for the reset.
	SYS_RESET_SoftwareReset	Triggers a software reset.

Description

Reset System Service Interface Definition

This file contains the interface definition for the Reset System Service. It provides a way to interact with the Reset subsystem to manage the system resets and their status.

File Name

sys_reset.h

Company

Microchip Technology Inc.

Timer System Service Library

This section describes the Timer System Service Library.

Introduction

This library provides interfaces to manage alarms and/or delays.

Description

The Timer System Service Library is capable of providing periodic or one-shot alarm delays to the user. It works as a client for the Timer driver and opens one of the instances of the driver to perform the activities. The periodicity or the delay is an integer multiple of the Timer driver alarm period.

Using the Library

This topic describes the basic architecture of the Timer System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_tmr.h`

The interface to the Timer System Service library is defined in the `sys_tmr.h` header file, which is included by the `sys.h` system service header file. Any C language source (`.c`) file that uses the Timer System Service library should include `sys.h`.

Please refer to the What is MPLAB Harmony? section for how the System Service interacts with the framework.

Abstraction Model

The Timer System Service module uses the Timer driver abstraction layer to provide the following functionalities,

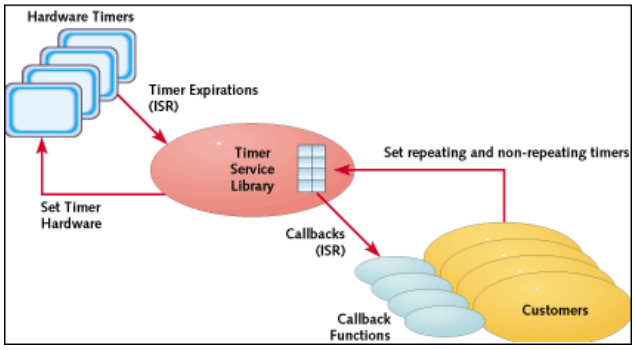
- Periodic Callback
- One Shot/Single Callback
- Delays

Note: The Timer System Service will work in both the polling and interrupt driven environments; however, Interrupt mode is recommended.

Description

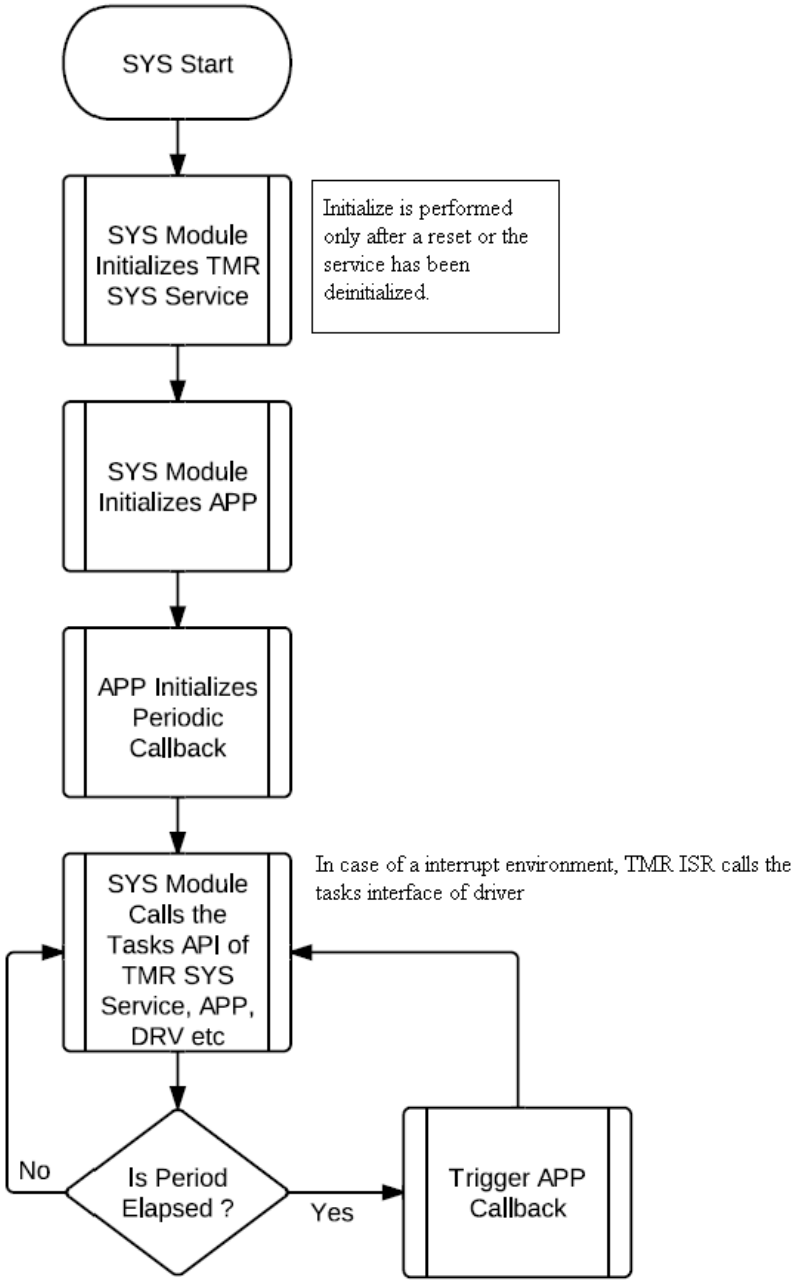
Abstraction Model

The following diagram depicts the Timer System Service abstraction model.



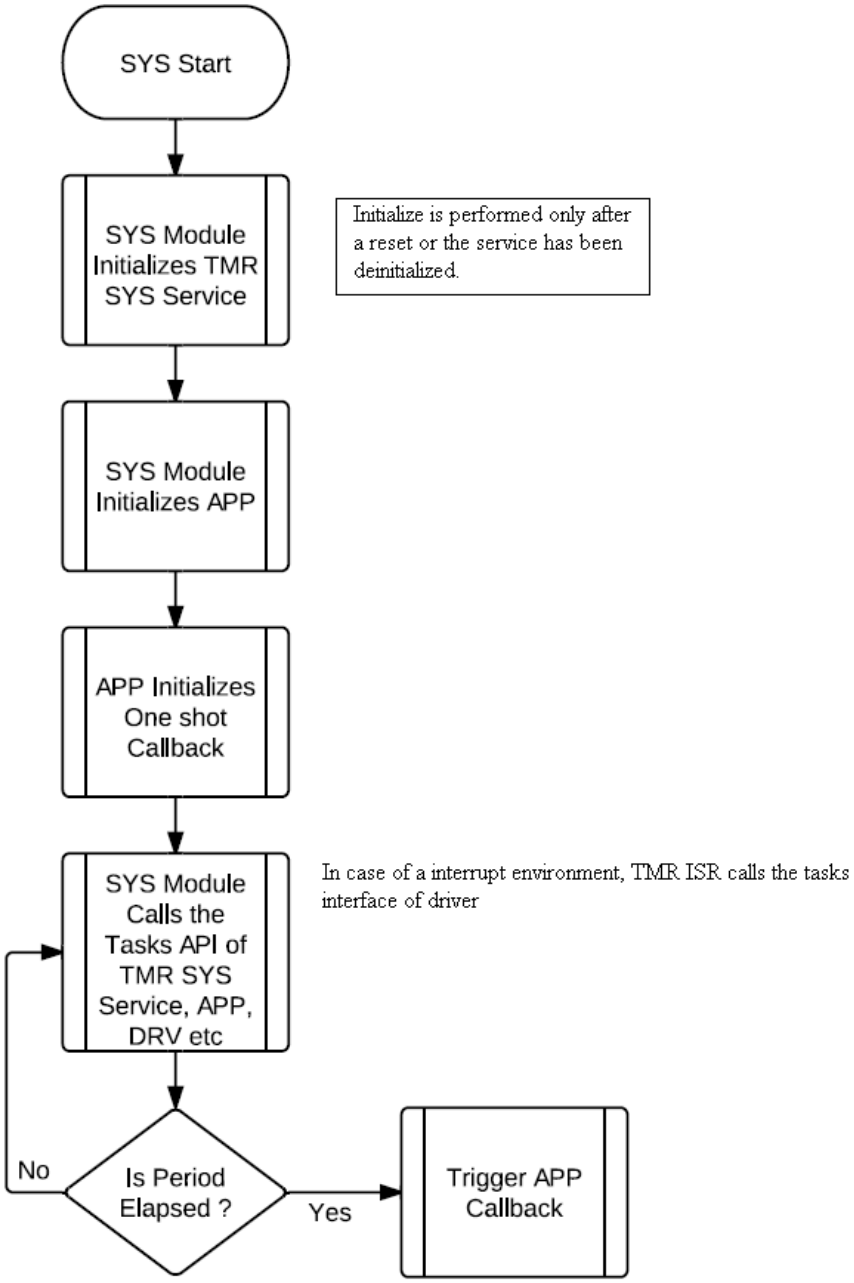
Periodic Callback

The following flow diagram depicts the Periodic Callback usage model.



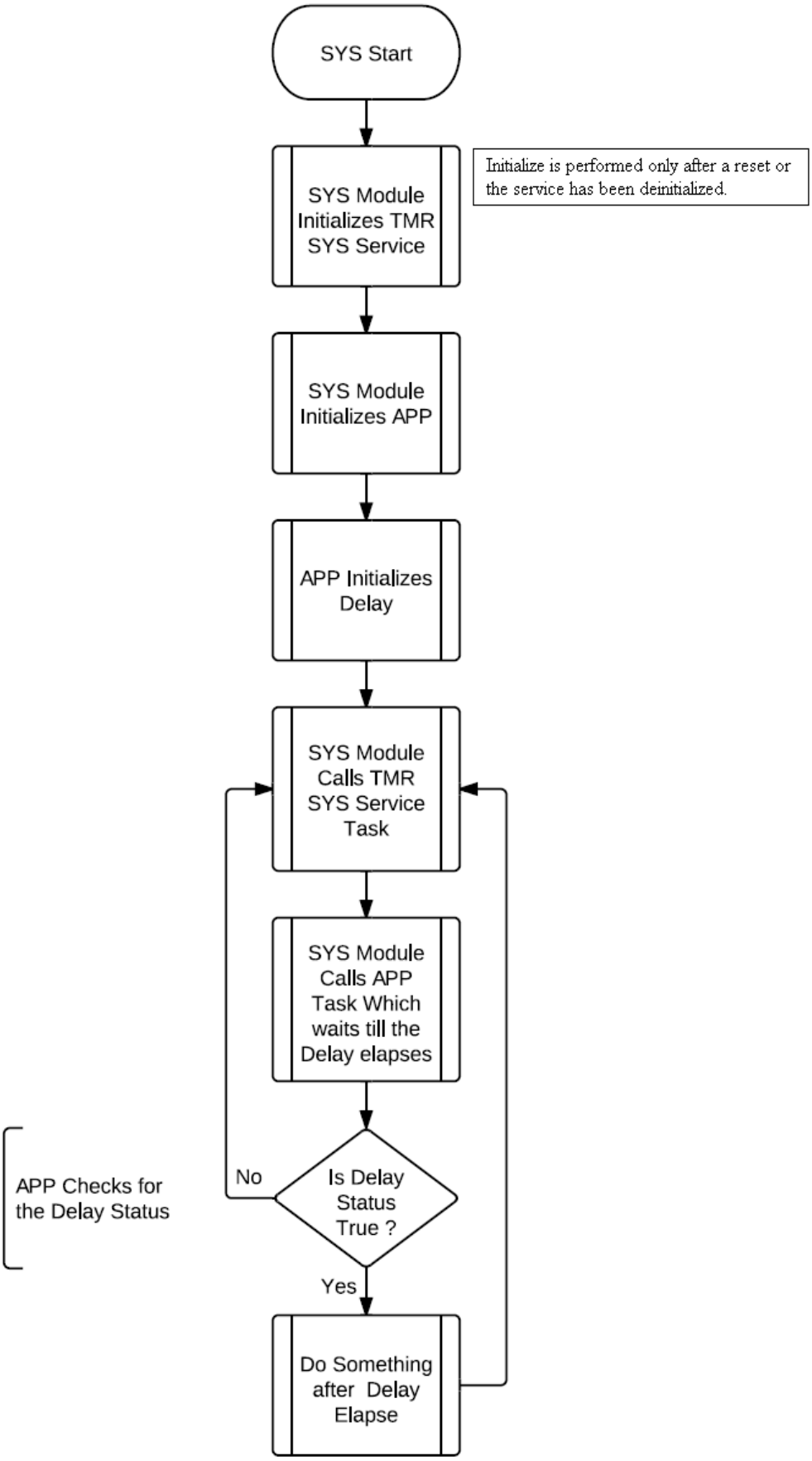
One-Shot Callback

The following flow diagram depicts the One-Shot Callback usage model.



Delay

The following flow diagram depicts the Delay usage model.


SYS Start
SYS Module Initializes TMR SYS Service
Initialize is performed only after a reset or the service has been deinitialized.
SYS Module Initializes APP
APP Initializes Delay
SYS Module Calls TMR SYS Service Task
SYS Module Calls APP Task Which waits till the Delay elapses
APP Checks for the Delay Status
No
Is Delay Status True ?
Yes
Do Something after Delay Elapse

Library Overview

Please refer to the System Service Introduction for a detailed description of MPLAB Harmony system services.

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Timer System Service Library.

Library Interface Section	Description
System Level Interaction Functions	Provides system module APIs. Device initialization, deinitialization, reinitialization and status functions.
Timed Callback Functions	Provides interfaces to handle timed periodic or one shot callbacks
Timed Delay Functions	Provides interfaces to handle timed delays
Miscellaneous Functions	Provides interfaces for timer tick counts, etc.

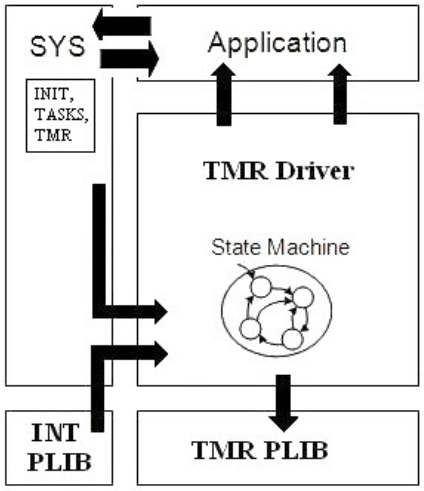
How the Library Works

The library provides interfaces to support:

- System Interaction
- Periodic Callback
- One shot/Single Callback
- Delays
- Tick count Information

Description

The following model gives information on interaction between various modules.



Note: The arrows in the diagram indicate the dependencies or support.

Module interaction occurs in the following order:

1. The initialization, reinitialization, deinitialization are handled by the SYS INIT module.
2. The Timer System Service Initialization function initializes the library's internal data structures.
3. After the initialization, Timer System Service Tasks function would be called by the SYS Tasks module to open the Timer Driver and do other configuration activities. Once the Timer driver is ready to be used the Timer System Service Tasks API makes the system ready to be used by setting the status of the Timer System Service module to SYS_STATUS_READY.
4. The application can now enable either periodic callback or one-shot callback or enable delays.

Notes:

1. It is possible to have multiple clients access the system timer service, each with different periodic or single callbacks or delays.
2. If the underlying Timer Driver uses Interrupt mode, it is the responsibility of the user or the application to ensure that the Interrupt system module has been initialized and that priorities have been set accordingly.
3. Ensure that the selected timer clock source is DRV_TMR_CLKSOURCE_INTERNAL, as the Timer System Service will not function if the timer clock source is external.

System Interaction

Initialization and Reinitialization:

The SYS module performs the initialization and the reinitialization of the Timer System Service. During initialization the following information is populated into the internal data structures from the init structure SYS_TMR_INIT passed as a parameter:

Initialization Member	Description
moduleInit	System module initialization of the power state
drvIndex	Timer driver module/instance index, the same index used to initialize the timer driver
tmrFreq	Required System Timer frequency in Hertz, which is the base frequency for all of the system time-keeping.

The SYS_TMR_Initialize API returns a handle of the type SYS_MODULE_OBJ. After this, the object handle returned by the Initialize interface would be used by the other system interfaces such as SYS_TMR_Reinitialize, SYS_TMR_Deinitialize, SYS_TMR_Status, and SYS_TMR_Tasks.

Note: If an attempt is made to initialize the SYS TMR module if it is already initialized, will safely return without modifying the behavior of the system.

Example for Timer System Service initialization:

```
SYS_MODULE_OBJ  objectHandle;
SYS_TMR_INIT    initConfig;

// Populate the SYS TMR initialization structure
initConfig.moduleInit.value     = SYS_MODULE_POWER_RUN_FULL;
initConfig.drvIndex             = DRV_TMR_INDEX_0;
initConfig.tmrFreq          = 1000;

objectHandle = SYS_TMR_Initialize( SYS_TMR_INDEX_0, ( SYS_MODULE_INIT * )&initConfig );
if( SYS_MODULE_OBJ_INVALID == objectHandle )
{
    // Handle error
}
```

Sample initialization sequence in polling environment:

```
void SYS_Initialize( SYS_INIT_DATA *data )
{
    /* Application Initialization */
    App_Init();

    /* Initialize the Timer driver */
    drvTmrObject = DRV_TMR_Initialize( DRV_TMR_INDEX_0, (SYS_MODULE_INIT *)&tmrInitData );

    /* Initialize the SYS TMR Module */
    sysTmrObject = SYS_TMR_Initialize( SYS_TMR_INDEX_0, (SYS_MODULE_INIT *)&sysTmrInitData );
}
```

Sample initialization sequence in an interrupt driven environment:'

```
void SYS_Initialize( SYS_INIT_DATA *data )
{
    /* Application Initialization */
    App_Init();

    /* Initialize the Timer driver */
    drvTmrObject = DRV_TMR_Initialize( DRV_TMR_INDEX_0, ( SYS_MODULE_INIT * )&tmrInitData );

    /* Initialize the SYS TMR Module */
    sysTmrObject = SYS_TMR_Initialize( SYS_TMR_INDEX_0, ( SYS_MODULE_INIT * )&sysTmrInitData );

/* Set the Timer Interrupt priority, sub-priority if supported by the device */
    SYS_INT_VectorPrioritySet(INT_VECTOR_T1, INT_PRIORITY_LEVEL1 );
    SYS_INT_VectorSubprioritySet(INT_VECTOR_T1, INT_SUBPRIORITY_LEVEL1 );

    /* Initialize the interrupt sub system module */
    SYS_INT_Initialize();
}
```

Deinitialization:

The deinitialize operation (SYS_TMR_Deinitialize) places the module in inactive state. Once the initialize operation has been called, the deinitialize

operation must be called before the initialize is called again. If the operation requires time to allow the hardware to complete, this will be reported by the SYS_TMR_Status operation.

Status:

Status of the system timer module can be checked using SYS_TMR_Status interface. After the initialization, reinitialization or the deinitialization activities the System timer Service status can be retrieved and checked for further actions.

Tasks Routine:

The system will call SYS_TMR_Tasks from System Task Service. This interface gets the system timer module into running mode. This interface needs to be called after the timer driver is successfully initialized.

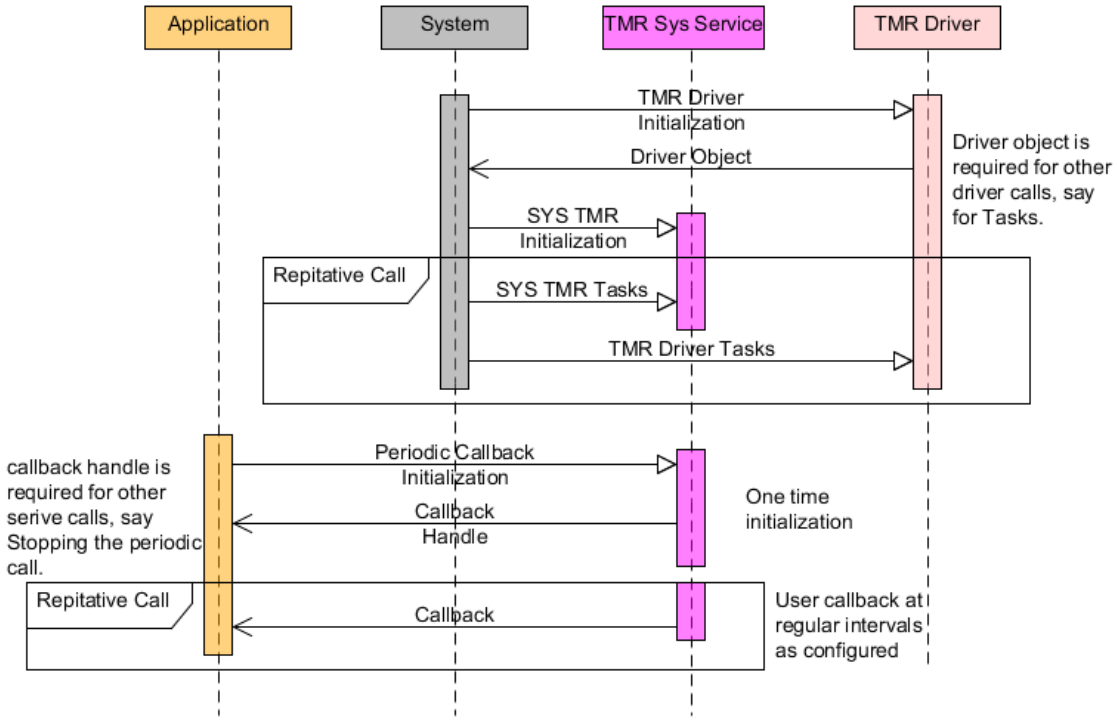
Sample call order:

```
void SYS_Tasks( void )
{
    /* Call Driver TMR Tasks API */
    DRV_TMR_Tasks( drvTmrObject );

    /* Call SYS TMR Tasks API */
    SYS_TMR_Tasks( sysTmrObject );

    /* Call the Application Tasks */
    App_Tasks();
}
```

Periodic Callback

This section describes the periodic callback registration and its usage.

The following diagram depicts the interaction across various modules for the periodic callback feature,



Steps involved in registering and using the feature:

1. The Timer System Service should be in a ready state. Use the SYS_TMR_STATUS function to verify that the service is in this state.
2. Register the periodic callback feature using the SYS_TMR_CallbackPeriodic function with the period, a context pointer, and the pointer to the callback routine. The interface SYS_TMR_CallbackPeriodic returns a valid handle after successful registration.

Notes:

1. The handle can be later used by the user to stop the periodic callback by the user using the interface SYS_TMR_CallbackStop. Based on the periodicity set by the user the respective callback would be triggered repetitively after it elapses.
2. The period parameter will be adjusted internally to match the Timer System running frequency.

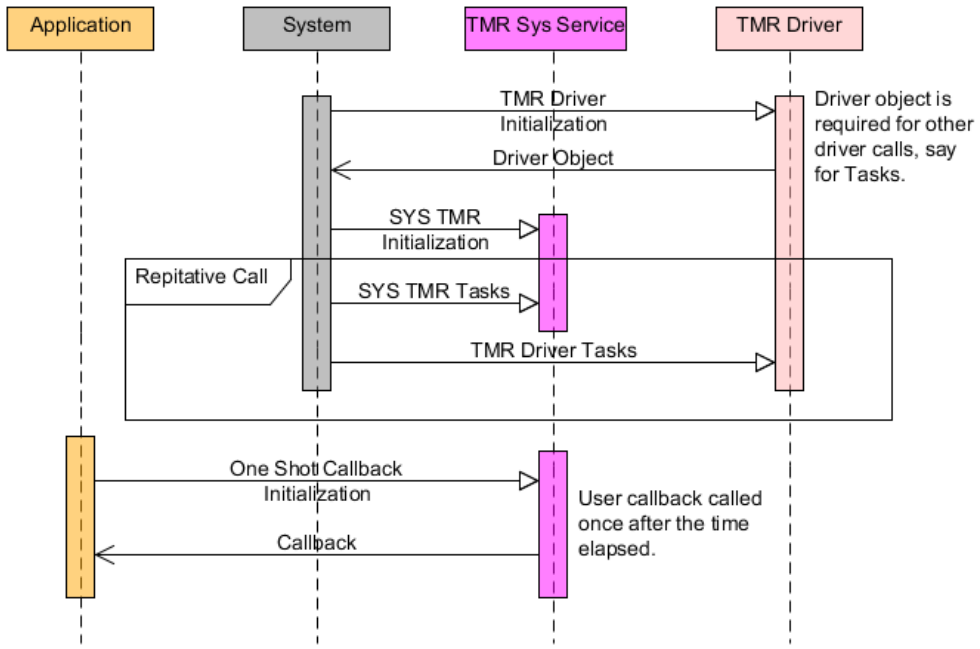
Example:

```
void SYS_Initialize ( SYS_INIT_DATA *data )
{
    /* Initialize the Timer driver */
    drvTmrObject = DRV_TMR_Initialize ( SYS_INDEX, (SYS_MODULE_INIT *)&tmrInitData );

    /* Initialize the SYS TMR Module */
    sysTmrObject = SYS_TMR_Initialize ( SYS_INDEX, (SYS_MODULE_INIT *)&sysTmrInitData );

    /* Application Initialization */
    App_Initialize ();

}

void SYS_Tasks(void)
{
    /* Call Driver TMR Tasks API */
    DRV_TMR_Tasks ( drvTmrObject );

    /* Call SYS TMR Tasks API */
    SYS_TMR_Tasks ( sysTmrObject );

    /* Call the App Tasks */
    App_Tasks ();
}

void App_Tasks ( void )
{
    switch (testState)
    {
        /* Wait in Init state until the SYS TMR Module is in running mode */
        case TEST_STATE_Init:
                if (SYS_STATUS_READY == SYS_TMR_Status(sysTmrObject))
                {
                    /* SYS TMR is in running mode */
                    testState = TEST_STATE_Config_Periodic;
                }
                else
                {
                    testState = TEST_STATE_Init;
                }
            break;

        case TEST_STATE_Config_Periodic:
                /* Activate periodic callback */
                handle = SYS_TMR_CallbackPeriodic (120, 0, &Test_Callback);
                testState = TEST_STATE_Count_Read;
             break;

        case TEST_STATE_Count_Read:
                testCount1 = SYS_TMR_TickCountGet ();
            break;
    }
}

// Callback which will be called repetitively after the configured
// time elapses
void Test_Callback ( uintptr_t context, uint32_t currTick )
{
    PORTA = ~PORTA;
}
```

One Shot Callback

This section describes the one-shot/single callback registration and its usage.

The following diagram depicts the interaction across various modules for the one-shot callback feature,



Steps involved in registering and using the feature:

1. Initial requirement is to have the Timer System Service module in the SYS_STATUS_READY state. The user can call the function SYS_TMR_Status to get the current status information of the module.
2. One-shot/Single callback feature can be registered by calling the function SYS_TMR_CallbackSingle with the parameters periodicity of the callback, a context pointer, and a pointer to the user's callback routine.

Based on the periodicity set by the user, the respective callback would be triggered once after it elapses.

Notes:

1. The handle can be later used by the user to stop the one-shot callback by the user using the interface SYS_TMR_CallbackStop.
2. The period parameter will be adjusted internally to match the Timer System running frequency.

Example:

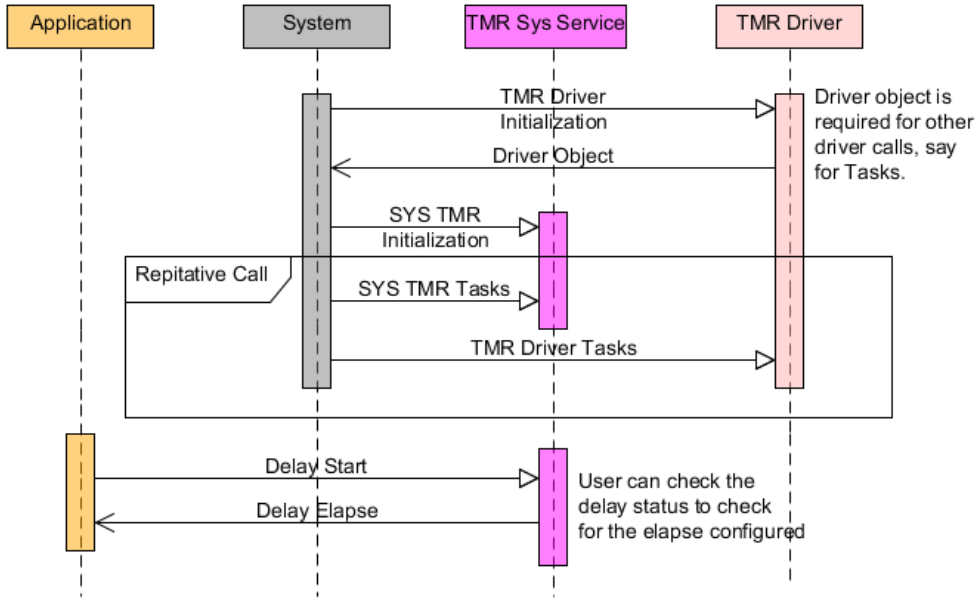
```
void SYS_Initialize ( SYS_INIT_DATA *data )
{
    /* Initialize the Timer driver */
    drvTmrObject = DRV_TMR_Initialize ( SYS_INDEX, (SYS_MODULE_INIT *)&tmrInitData );

    /* Initialize the SYS TMR Module */
    sysTmrObject = SYS_TMR_Initialize ( SYS_INDEX, (SYS_MODULE_INIT *)&sysTmrInitData );

    /* Application Initialization */
    App_Initialize ();

}

void SYS_Tasks(void)
{
    /* Call Driver TMR Tasks API */
    DRV_TMR_Tasks ( drvTmrObject );

    /* Call SYS TMR Tasks API */
    SYS_TMR_Tasks ( sysTmrObject );

    /* Call the App Tasks */
    App_Tasks ();
}

void App_Tasks ( void )
{
    switch (testState)
    {
```

```
        /* Wait in Init state until the SYS TMR Module is in running mode */
        case TEST_STATE_Init:
                if (SYS_STATUS_READY == SYS_TMR_Status(sysTmrObject))
                {
                    /* SYS TMR is in running mode */
                    testState = TEST_STATE_Config_Single;
                }
                else
                {
                    testState = TEST_STATE_Init;
                }
            break;

        case TEST_STATE_Config_Single:
                /* Activate one shot/single callback */
                handle = SYS_TMR_CallbackSingle (120, 0, &Test_Callback);
                testState = TEST_STATE_Count_Read;
             break;

        case TEST_STATE_Count_Read:
                testCount1 = SYS_TMR_TickCountGet ();
            break;
    }
}

// Callback which will be called repetitively after the configured
// time elapses
void Test_Callback ( uintptr_t context, uint32_t currTick )
{
    PORTA = ~PORTA;
}
```

Delays

This section describes the delay registration and its usage. The mechanism depicted supports only non-blocking delays.

The following diagram depicts the interaction across various modules for the delay feature.



Steps involved in registering and using the feature:

1. Initial requirement is to have the Timer System Service module in the SYS_STATUS_READY state. The user can call the function SYS_TMR_Status to get the current status information of the module.
2. Delay feature can be registered by calling the function SYS_TMR_DelayMS with the delay parameter.

Notes:

1. The API SYS_TMR_DelayMS returns a valid handle after successful registration. This handle can be later used by the user to check the status of the delay using the API SYS_TMR_DelayStatusGet.
2. The period parameter will be adjusted internally to match the Timer System running frequency.

Example:

```
case DELAY_START_STATE:
    handle = SYS_TMR_DelayMS (500);
    state = DELAY_CHECKSTATE;
break;

case DELAY_CHECKSTATE:
    if (true == SYS_TMR_DelayStatusGet(handle))
    {
        state = DELAY_START_STATE;
    }
    else
    {
        state = DELAY_CHECKSTATE;
    }
break;
```

Miscellaneous

Tick Counts Information: The API SYS_TMR_TickCountGet provides the current tick information.

Examples

Sample Application Tasks

Periodic callback with only one period with 120 ms period:

```
switch (state)
{
  case STATE_Init:
    if (SYS_STATUS_READY == SYS_TMR_Status(sysTmrObject))
    {
      state = STATE_Config_Periodic;
    }
    else
    {
      state = STATE_Init;
    }
  break;

  case STATE_Config_Periodic:
    handle = SYS_TMR_CallbackPeriodic (120, 0, &Test_Callback);
    state = STATE_Count_Read;
  break;

  case STATE_Count_Read:
    testCount = SYS_TMR_TickCountGet ();
    state = STATE_Count_Read;
  break;
}
```

Periodic callback with multiple periods 120 ms and 250 ms:

```
switch (state)
{
  case STATE_Init:
    if (SYS_STATUS_READY == SYS_TMR_Status(sysTmrObject))
    {
      state = STATE_Config_Periodic;
    }
    else
    {
      state = STATE_Init;
    }
  break;
```

```
  case STATE_Config_Periodic:
    handle1 = SYS_TMR_CallbackPeriodic (120, 0, &Test_Callback1);
    handle2 = SYS_TMR_CallbackPeriodic (250, 0, &Test_Callback2);
    state = STATE_Count_Read;
  break;

  case STATE_Count_Read:
    testCount = SYS_TMR_TickCountGet ();
    state = STATE_Count_Read;
  break;
}

void Test_Callback1 ( uintptr_t context, uint32_t currTick )
{
  /* Assume RA0 is connected to one LED */
  PORTAbits.RA0 = ~PORTAbits.RA0;
}

void Test_Callback2 ( uintptr_t context, uint32_t currTick )
{
  /* Assume RA1 is connected to another LED */
    PORTAbits.RA1 = ~PORTAbits.RA1;
}
```

One-Shot Callback:

```
switch (state)
{
  case STATE_Init:
    if (SYS_STATUS_READY == SYS_TMR_Status(sysTmrObject))
    {
      state = STATE_Config_Single;
    }
    else
    {
      state = STATE_Init;
    }
  break;

  case STATE_Config_Single:
    SYS_TMR_CallbackSingle (500, 0, &Test_Callback);
    state = STATE_Count_Read;
  break;

  case STATE_Count_Read:
    testCount = SYS_TMR_TickCountGet ();
    state = STATE_Count_Read;
  break;
}
```

Delay:

```
switch (state)
{
  case STATE_Init:
    if (SYS_STATUS_READY == SYS_TMR_Status(sysTmrObject))
    {
      state = STATE_DelayStart;
    }
    else
    {
      state = STATE_Init;
    }
  break;

  case STATE_DelayStart:
    dlHandle = SYS_TMR_DelayMS (500);
    state = STATE_DelayCheck;
  break;

  case STATE_DelayCheck:
    if (SYS_TMR_DELAY_EXPIRED == SYS_TMR_DelayStatusGet(dlHandle))
```

```
      {
        state = STATE_DelayStart;
      }
      else
      {
        state = STATE_DelayCheck;
      }
    break;
  }
```

Configuring the Library

Macros

	Name	Description
	_SYS_TMR_CONFIG_TEMPLATE_H	This is macro _SYS_TMR_CONFIG_TEMPLATE_H.

Description

The configuration of the Timer System Service is based on the file `system_config.h`.

This header file contains the configuration selection for the Timer System Service build. Based on the selections made, the Timer System Service may support the selected features. These configuration settings will apply to all instances of the Timer System Service.

This header can be placed anywhere, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

_SYS_TMR_CONFIG_TEMPLATE_H Macro

File

sys_tmr_config_template.h

C

```
#define _SYS_TMR_CONFIG_TEMPLATE_H
```

Description

This is macro _SYS_TMR_CONFIG_TEMPLATE_H.

Building the Library

This section lists the files that are available in the Timer System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/timer`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using #include) by any code that uses this library.

Source File Name	Description
`sys_tmr.h`	Timer System Service Library API header file.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/sys_tmr.c`	Timer System Service Library implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The Timer System Service Library depends on the following modules:

- Timer Driver Library

Library Interface

a) System Level Interaction Functions

	Name	Description
	SYS_TMR_Deinitialize	Deinitializes the specific module instance of the TMR module
	SYS_TMR_Initialize	Initializes hardware and data for the instance of the Timer module and opens the specific module instance.
	SYS_TMR_Status	Returns status of the specific instance of the Timer module.
	SYS_TMR_Tasks	Maintains the system timer's state machine and implements its ISR.

b) Timed Callback Functions

	Name	Description
	SYS_TMR_CallbackStop	Stops a periodic timer object.
	SYS_TMR_CallbackPeriodic	Creates a permanent timer object that calls back to the client periodically.
	SYS_TMR_CallbackSingle	Creates a one-shot/single callback timer object.

c) Timed Delay Functions

	Name	Description
	SYS_TMR_DelayStatusGet	Checks the status of the previously requested delay timer object.
	SYS_TMR_DelayMS	Creates a timer object that times out after the specified delay.

d) Timed Object Functions

	Name	Description
	SYS_TMR_ObjectCountGet	Returns the current millisecond count of a timer object.
	SYS_TMR_ObjectCreate	Creates a new timer object.
	SYS_TMR_ObjectDelete	Deletes an existent timer object.
	SYS_TMR_ObjectReload	Reloads an existing timer object.

e) Miscellaneous Functions

	Name	Description
	SYS_TMR_TickCountGet	Provides the current counter value.
	SYS_TMR_TickCountGetLong	Provides the current counter value with 64-bit resolution.
	SYS_TMR_TickCounterFrequencyGet	Provides the number of ticks per second
	SYS_TMR_SystemCountGet	Provides the current system time count value.
	SYS_TMR_SystemCountFrequencyGet	Provides the current system count frequency.
	SYS_TMR_ModuleStatusGet	Returns status of the specific instance of the Timer module.

f) Data Types and Constants

	Name	Description
	SYS_TMR_CALLBACK	This data type defines a pointer callback function.
	SYS_TMR_HANDLE	Identifies a client timer object.
	SYS_TMR_INIT	Identifies the system timer initialize structure.
	SYS_TMR_HANDLE_INVALID	Identifies the invalid handle of the system timer.
	SYS_TMR_INDEX_0	Timer System Service index definitions.
	SYS_TMR_FLAGS	Defines the types and flags supported by a client timer object.
	SYS_TMR_CLIENT_TOLERANCE	Sets the accepted client tolerance in %.
	SYS_TMR_DRIVER_INDEX	Sets the default timer driver object to be used by the system timer service.
	SYS_TMR_FREQUENCY	Specifies the operating frequency for the system timer, Hz.

	SYS_TMR_FREQUENCY_TOLERANCE	Sets the error tolerance in %.
	SYS_TMR_INTERRUPT_NOTIFICATION	Activates the notification mechanism from within the timer interrupt.
	SYS_TMR_MAX_CLIENT_OBJECTS	Sets the maximum number of timer clients that can be active at a given time.
	SYS_TMR_MODULE_INIT	Sets the default module init value by the system timer service.
	SYS_TMR_UNIT_RESOLUTION	Specifies the internal units for the time keeping.

Description

This section describes the APIs of the Timer System Service Library.

Refer to each section for a detailed description.

a) System Level Interaction Functions

SYS_TMR_Deinitialize Function

Deinitializes the specific module instance of the TMR module

File

sys_tmr.h

C

```
void SYS_TMR_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function deinitializes the specific module instance disabling its operation (and any hardware for driver modules). Resets all of the internal data structures and fields for the specified instance to the default settings.

Remarks

Once the Initialize operation has been called, the De-initialize operation must be called before the Initialize operation can be called again.

Preconditions

The SYS_TMR_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from SYS_TMR_Initialize
SYS_STATUS          status;

SYS_TMR_Deinitialize (object);

status = SYS_TMR_Status (object);

if (SYS_MODULE_UNINITIALIZED == status)
{
    // Check again later if you need to know
    // when the SYS TMR is deinitialized.
}
```

Parameters

Parameters	Description
object	SYS TMR object handle, returned from SYS_TMR_Initialize

Function

void SYS_TMR_Deinitialize (SYS_MODULE_OBJ object)

SYS_TMR_Initialize Function

Initializes hardware and data for the instance of the Timer module and opens the specific module instance.

File

sys_tmr.h

C

```
SYS_MODULE_OBJ SYS_TMR_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to an object. Otherwise, it returns SYS_MODULE_OBJ_INVALID. The returned object must be passed as argument to SYS_TMR_Reinitialize, SYS_TMR_Deinitialize, SYS_TMR_Tasks and SYS_TMR_Status routines.

Description

This function initializes hardware for the instance of the Timer module, using the specified hardware initialization data. It also initializes any internal data structures.

Remarks

This routine should only be called once during system initialization.

Preconditions

None.

Example

```
SYS_MODULE_OBJ  objectHandle;
SYS_TMR_INIT    initConfig;

// Populate the timer initialization structure
initConfig.moduleInit.value     = SYS_MODULE_POWER_RUN_FULL;
initConfig.drvIndex             = DRV_TMR_INDEX_0;
initConfig.tmrFreq              = 1000;

objectHandle = SYS_TMR_Initialize (SYS_TMR_INDEX_0, (SYS_MODULE_INIT*)&initConfig);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
index	Index for the instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the sys timer. This pointer may be null if no data is required because static overrides have been provided.

Function

SYS_MODULE_OBJ SYS_TMR_Initialize (const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init)

SYS_TMR_Status Function

Returns status of the specific instance of the Timer module.

File

sys_tmr.h

C

```
SYS_STATUS SYS_TMR_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that the driver initialization is complete and ready to be used.

Description

This function returns the status of the specific module instance.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations

- SYS_STATUS_ERROR - Indicates that the driver is in an error state.

Any value less than SYS_STATUS_ERROR is also an error state.

- SYS_MODULE_UNINITIALIZED - Indicates that the driver has been deinitialized.

This value is less than SYS_STATUS_ERROR.

None.

Preconditions

The SYS_TMR_Initialize function should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from SYS_TMR_Initialize
SYS_STATUS          tmrStatus;

tmrStatus = SYS_TMR_Status (object);

if ( SYS_STATUS_ERROR <= tmrStatus )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	SYS TMR object handle, returned from SYS_TMR_Initialize

Function

SYS_STATUS SYS_TMR_Status (SYS_MODULE_OBJ object)

SYS_TMR_Tasks Function

Maintains the system timer's state machine and implements its ISR.

File

sys_tmr.h

C

```
void SYS_TMR_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the system timer's internal state machine. It should be always be called from 'Sys_Tasks' function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks) or by the appropriate raw ISR.

Preconditions

The SYS_TMR_Initialize function must have been called for the specified TMR driver instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from SYS_TMR_Initialize

while (true)
{
    SYS_TMR_Tasks ( object );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	SYS TMR object handle, returned from SYS_TMR_Initialize

Function

void SYS_TMR_Tasks (SYS_MODULE_OBJ object)

b) Timed Callback Functions

SYS_TMR_CallbackStop Function

Stops a periodic timer object.

File

sys_tmr.h

C

```
void SYS_TMR_CallbackStop(SYS_TMR_HANDLE handle);
```

Returns

None.

Description

This function deletes a previously created periodic timer object. All the associated resources are released.

Remarks

This function is equivalent to SYS_TMR_ObjectDelete(handle);.

Preconditions

The SYS_TMR_CallbackPeriodic function should have been called to obtain a valid timer handle.

Example

```
int count, state;
void Test_Callback ( uintptr_t context, uint32_t currTick )
{
    count++;
}

void DelayTask ( )
{
    SYS_TMR_HANDLE handle;
    switch (state)
    {
        case 1:
            handle = SYS_TMR_CallbackPeriodic ( 20, 1, Test_Callback );
            state++;
            break;
        case 2:
            if ( count > 100 )
            {
                SYS_TMR_CallbackStop ( handle );
                state++;
            }
            count++;
            break;
    }
}
```

Parameters

Parameters	Description
handle	A valid periodic timer handle, returned by a SYS_TMR_CallbackPeriodic call.

Function

void SYS_TMR_CallbackStop (SYS_TMR_HANDLE handle)

SYS_TMR_CallbackPeriodic Function

Creates a permanent timer object that calls back to the client periodically.

File

sys_tmr.h

C

```
SYS_TMR_HANDLE SYS_TMR_CallbackPeriodic(uint32_t periodMs, uintptr_t context, SYS_TMR_CALLBACK callback);
```

Returns

A valid timer object handle of type SYS_TMR_HANDLE if the call succeeds. SYS_TMR_HANDLE_INVALID otherwise.

Description

This function creates a timer object that will continuously notify the client once the time-out occurs. The timer object will be deleted by a call to SYS_TMR_CallbackStop

Remarks

The created timer object is persistent and needs to be deleted with SYS_TMR_CallbackStop when no longer needed.

This function is equivalent to SYS_TMR_ObjectCreate(periodMs, context, callback, SYS_TMR_FLAG_PERIODIC);

Preconditions

The SYS_TMR_Initialize function should have been called before calling this function.

Example

```
SYS_TMR_HANDLE handle20ms, handle30ms;

void Test_Callback ( uintptr_t context, uint32_t currTick );

handle20ms = SYS_TMR_CallbackPeriodic ( 20, 1, Test_Callback );
handle30ms = SYS_TMR_CallbackPeriodic ( 20, 2, Test_Callback );

void Test_Callback ( uintptr_t context, uint32_t currTick )
{
    if ( context == 1 )
    {
        //20 ms
    }
    else
    {
        //30 ms
    }
}
```

Parameters

Parameters	Description
periodMs	Periodic delay in milliseconds
context	A client parameter that's passed in the callback function. This will help to identify the callback.
callback	Pointer to a callback routine that will be called periodically once the time-out occurs.

Function

SYS_TMR_HANDLE SYS_TMR_CallbackPeriodic (uint32_t periodMs, uintptr_t context,
SYS_TMR_CALLBACK callback)

SYS_TMR_CallbackSingle Function

Creates a one-shot/single callback timer object.

File

sys_tmr.h

C

```
SYS_TMR_HANDLE SYS_TMR_CallbackSingle(uint32_t periodMs, uintptr_t context, SYS_TMR_CALLBACK callback);
```

Returns

A valid timer object handle of type SYS_TMR_HANDLE if the call succeeds. SYS_TMR_HANDLE_INVALID otherwise.

Description

This function creates a self-destroying one-shot/single timer object that will notify the client once the time-out occurs. Once the time-out occurs and the notification is sent the timer object will be invalidated.

Remarks

The timer object will delete itself and free the associated resources. The SYS_TMR_HANDLE will be no longer valid.

However, SYS_TMR_CallbackStop() can be called to kill the timer object before its time-out expires, if not needed anymore.

This function is equivalent to SYS_TMR_ObjectCreate(periodMs, context, callback, SYS_TMR_FLAG_SINGLE | SYS_TMR_FLAG_AUTO_DELETE);

Preconditions

The SYS_TMR_Initialize function should have been called before calling this function.

Example

```
SYS_TMR_HANDLE handle;
void Test_Callback20ms ( uintptr_t context, uint32_t currTick );

handle = SYS_TMR_CallbackSingle ( 20, 1, Test_Callback20ms );
```

Parameters

Parameters	Description
periodMs	Periodic delay in milliseconds
context	A client parameter that's passed in the callback function. This will help to identify the callback.
callback	Pointer to a callback routine which will be called once

Function

SYS_TMR_HANDLE SYS_TMR_CallbackSingle (uint32_t periodMs, uintptr_t context,
SYS_TMR_CALLBACK callback)

c) Timed Delay Functions

SYS_TMR_DelayStatusGet Function

Checks the status of the previously requested delay timer object.

File

sys_tmr.h

C

```
bool SYS_TMR_DelayStatusGet(SYS_TMR_HANDLE handle);
```

Returns

- true - If the delay is elapsed
- false - If the delay is not elapsed

Description

This function checks the status of the previously requested delay timer object.

Remarks

Once the required delay is achieved and SYS_TMR_DelayStatusGet returns true the SYS_TMR_HANDLE will be no longer valid and the created timer object is deleted.

Preconditions

The SYS_TMR_DelayMS function should have been called before calling this function.

Example

```
SYS_TMR_HANDLE tmrHandle;

case APP_ADD_DELAY:
    tmrHandle = SYS_TMR_DelayMS ( 50 );
    state = APP_CHECK_DELAY;
    break;
case APP_CHECK_DELAY:
    if ( SYS_TMR_DelayStatusGet ( tmrHandle ) == true )
    {
        state = APP_DELAY_COMPLETE;
    }
    break;
```

Parameters

Parameters	Description
handle	A valid timer object handle, returned by SYS_TMR_DelayMS

Function

bool SYS_TMR_DelayStatusGet (SYS_TMR_HANDLE handle)

SYS_TMR_DelayMS Function

Creates a timer object that times out after the specified delay.

File

sys_tmr.h

C

```
SYS_TMR_HANDLE SYS_TMR_DelayMS(uint32_t delayMs);
```

Returns

A valid timer handler of type SYS_TMR_HANDLE is returned if the call is successful. SYS_TMR_HANDLE_INVALID otherwise.

Description

This function creates a self-destroying timer object that times out after the required delay. The delay status can be checked using SYS_TMR_DelayStatusGet();

Remarks

Once the required delay is achieved and SYS_TMR_DelayStatusGet() returns true the SYS_TMR_HANDLE will be no longer valid and the created timer object is deleted.

However, SYS_TMR_CallbackStop() can be called to kill the timer object before its time-out expires, if not needed anymore.

This function is equivalent to SYS_TMR_ObjectCreate(delayMs, 0, 0, SYS_TMR_FLAG_SINGLE | SYS_TMR_FLAG_AUTO_DELETE);

Preconditions

The SYS_TMR_Initialize function should have been called before calling this function.

Example

```
SYS_TMR_HANDLE tmrHandle;

case APP_ADD_DELAY:
    tmrHandle = SYS_TMR_DelayMS ( 50 );
    state = APP_CHECK_DELAY;
    break;
case APP_CHECK_DELAY:
    if ( SYS_TMR_DelayStatusGet (tmrHandle ) == true )
    {
        state = APP_DELAY_COMPLETE;
    }
    break;
```

Parameters

Parameters	Description
delayMs	Indicates the delay required in milliseconds

Function

SYS_TMR_HANDLE SYS_TMR_DelayMS (uint32_t delayMs)

d) Timed Object Functions

SYS_TMR_ObjectCountGet Function

Returns the current millisecond count of a timer object.

File

sys_tmr.h

C

```
uint32_t SYS_TMR_ObjectCountGet(SYS_TMR_HANDLE handle, uint32_t* pRateMs);
```

Returns

- > 0 - the current timer object millisecond count
- = 0 - time-out has occurred
- < 0 - if invalid timer object

Description

This function returns the current count and the time-out values of a timer object, in milliseconds.

Remarks

This function will destroy the auto-delete timer objects if the time-out has occurred.

Preconditions

The SYS_TMR_ObjectCreate function should have been called to get a valid timer object handle.

Parameters

Parameters	Description
handle	A valid timer handle, returned by a SYS_TMR_ObjectCreate call.
pRateMs	pointer to address to store the current object reload rate, in milliseconds. Could be NULL, if not needed.

Function

uint32_t SYS_TMR_ObjectCountGet (SYS_TMR_HANDLE handle, uint32_t* pRateMs)

SYS_TMR_ObjectCreate Function

Creates a new timer object.

File

sys_tmr.h

C

```
SYS_TMR_HANDLE SYS_TMR_ObjectCreate(uint32_t periodMs, uintptr_t context, SYS_TMR_CALLBACK callback,
SYS_TMR_FLAGS flags);
```

Returns

A valid timer object handle of type SYS_TMR_HANDLE if the call succeeds. SYS_TMR_HANDLE_INVALID otherwise.

Description

This function creates a new timer object of requested type. If a callback is supplied, it will notify the user once the time-out occurs. The timer object will be deleted by a call to SYS_TMR_ObjectDelete

Remarks

The created timer object needs to be deleted with SYS_TMR_ObjectDelete when no longer needed (unless is auto-delete type).

The auto-delete objects are deleted if the time-out has occurred and the client has been notified, meaning:

- The object has a notification/callback function, which has been called upon time-out.
- SYS_TMR_ObjectCountGet()/SYS_TMR_DelayStatusGet has been called

Preconditions

The SYS_TMR_Initialize function should have been called before calling this function.

Example

```
SYS_TMR_HANDLE handle20ms, handle30ms;

void Test_Callback ( uintptr_t context, uint32_t currTick );

handle20ms = SYS_TMR_ObjectCreate ( 20, 1, Test_Callback, SYS_TMR_FLAG_PERIODIC );
handle30ms = SYS_TMR_ObjectCreate ( 30, 2, Test_Callback, SYS_TMR_FLAG_PERIODIC );

void Test_Callback ( uintptr_t context, uint32_t currTick )
{
    if ( context == 1 )
    {
        //20 ms
    }
    else
    {
        //30 ms
    }
}

// when done with the timer objects
SYS_TMR_ObjectDelete(handle20ms);
SYS_TMR_ObjectDelete(handle30ms);
```

Parameters

Parameters	Description
periodMs	Periodic delay in milliseconds
context	A client parameter that's passed in the callback function. This will help to identify the callback. Can be 0 if not needed.
callback	Pointer to a callback routine that will be called periodically once the time-out occurs. Can be NULL if not needed.
flags	multiple flags specifying what type of object is created: • SYS_TMR_FLAG_PERIODIC - a periodic timer object is requested • SYS_TMR_FLAG_SINGLE - a single timer object is requested • SYS_TMR_FLAG_AUTO_DELETE - auto-delete object (SYS_TMR_FLAG_SINGLE only)

Function

SYS_TMR_HANDLE SYS_TMR_ObjectCreate (uint32_t periodMs, uintptr_t context,
SYS_TMR_CALLBACK callback, SYS_TMR_FLAGS flags)

SYS_TMR_ObjectDelete Function

Deletes an existent timer object.

File

sys_tmr.h

C

```
void SYS_TMR_ObjectDelete(SYS_TMR_HANDLE handle);
```

Returns

None.

Description

This function deletes a timer object and release the associated resources.

Remarks

The created timer objects need to be deleted when it is no longer needed (unless of auto-delete type).

Preconditions

The SYS_TMR_ObjectCreate function should have been called to get a valid timer object handle.

Parameters

Parameters	Description
handle	A valid timer handle, returned by a SYS_TMR_ObjectCreate call.

Function

void SYS_TMR_ObjectDelete (SYS_TMR_HANDLE handle)

SYS_TMR_ObjectReload Function

Reloads an existing timer object.

File

sys_tmr.h

C

```
bool SYS_TMR_ObjectReload(SYS_TMR_HANDLE handle, uint32_t periodMs, uintptr_t context, SYS_TMR_CALLBACK callback);
```

Returns

- true - If the call succeeds
- false - If the call fails

Description

This function changes the parameters for an already existing timer object. The timer object should be valid.

Remarks

The function works for any kind of timer object. It will fail if the timer object was auto-delete type and the time-out has already expired.

Preconditions

The SYS_TMR_ObjectCreate function should have been called to get a valid timer object handle.

Parameters

Parameters	Description
handle	A valid timer handle, returned by a SYS_TMR_ObjectCreate call.
periodMs	Periodic delay in milliseconds
context	A client parameter that's passed in the callback function. This will help to identify the callback. Can be '0' if not needed.
callback	Pointer to a callback routine that will be called periodically once the time-out occurs. Can be NULL if not needed.

Function

bool SYS_TMR_ObjectReload (SYS_TMR_HANDLE handle, uint32_t periodMs,
uintptr_t context, SYS_TMR_CALLBACK callback)

e) Miscellaneous Functions

SYS_TMR_TickCountGet Function

Provides the current counter value.

File

sys_tmr.h

C

```
uint32_t SYS_TMR_TickCountGet();
```

Returns

The current system timer tick count value.

Description

This function provides the current tick counter value.

Remarks

None.

Preconditions

The SYS_TMR_Initialize function should have been called before calling this function.

Example

```
uint32_t count;
count = SYS_TMR_TickCountGet ();
```

Function

uint32_t SYS_TMR_TickCountGet(void)

SYS_TMR_TickCountGetLong Function

Provides the current counter value with 64-bit resolution.

File

sys_tmr.h

C

```
uint64_t SYS_TMR_TickCountGetLong();
```

Returns

The current system timer tick count value.

Description

This function provides the current tick counter value as a 64-bit value.

Remarks

None.

Preconditions

The SYS_TMR_Initialize function should have been called before calling this function.

Example

```
uint64_t lCount;
lCount = SYS_TMR_TickCountGetLong ();
```

Function

uint64_t SYS_TMR_TickCountGetLong(void)

SYS_TMR_TickCounterFrequencyGet Function

Provides the number of ticks per second

File

sys_tmr.h

C

```
uint32_t SYS_TMR_TickCounterFrequencyGet();
```

Returns

The current system timer ticks per second.

Description

This function provides the system tick resolution, the number of ticks per second

Remarks

None.

Preconditions

The SYS_TMR_Initialize function should have been called before calling this function.

Example

```
uint32_t tickFrequencyHz;

tickFrequencyHz = SYS_TMR_TickCounterFrequencyGet ( );
```

Function

uint32_t SYS_TMR_TickCounterFrequencyGet (void)

SYS_TMR_SystemCountGet Function

Provides the current system time count value.

File

sys_tmr.h

C

```
uint64_t SYS_TMR_SystemCountGet();
```

Returns

The current system count value.

Description

This function provides the current system time count value with the greatest possible resolution. It is the number of counts from the system start up as counted with the system count frequency (SYS_TMR_SystemCountFrequencyGet).

Remarks

None.

Preconditions

The SYS_TMR_Initialize function should have been called before calling this function.

Example

```
uint64_t sysCount;
sysCount = SYS_TMR_SystemCountGet ();
// the system has been running for sysCount/SYS_TMR_SystemCountFrequencyGet seconds
```

Function

uint64_t SYS_TMR_SystemCountGet(void)

SYS_TMR_SystemCountFrequencyGet Function

Provides the current system count frequency.

File

sys_tmr.h

C

```
uint32_t SYS_TMR_SystemCountFrequencyGet();
```

Returns

The current system frequency value in Hertz.

Description

This function provides the current system count frequency in Hz. The tick count measurement is based on this value. This is the resolution of the system time base.

Remarks

None.

Preconditions

The SYS_TMR_Initialize function should have been called before calling this function.

Example

```
uint32_t sysCountFreq;
sysCountFreq = SYS_TMR_SystemCountFrequencyGet ();
```

Function

uint32_t SYS_TMR_SystemCountFrequencyGet(void)

SYS_TMR_ModuleStatusGet Function

Returns status of the specific instance of the Timer module.

File

sys_tmr.h

C

```
SYS_STATUS SYS_TMR_ModuleStatusGet(const SYS_MODULE_INDEX index);
```

Returns

- SYS_STATUS_READY - Indicates that the service initialization is complete and the timer is ready to be used.
- SYS_STATUS_ERROR - Indicates that the timer service is in an error state

Description

This function returns the status of the specific module instance. It uses the index of the instance rather than an object handle.

Remarks

Any value less than SYS_STATUS_ERROR is also an error state.

- SYS_MODULE_UNINITIALIZED - Indicates that the timer service has been deinitialized

None.

Preconditions

The SYS_TMR_Initialize function should have been called before calling this function.

Example

```
SYS_STATUS tmrStatus = SYS_TMR_ModuleStatusGet (SYS_TMR_INDEX_0);

if ( SYS_STATUS_ERROR <= tmrStatus )
{
    // Handle error
}
```

Parameters

Parameters	Description
object	SYS TMR object handle, returned from SYS_TMR_Initialize

Function

SYS_STATUS SYS_TMR_ModuleStatusGet (const SYS_MODULE_INDEX index)

f) Data Types and Constants

SYS_TMR_CALLBACK Type

This data type defines a pointer callback function.

File

sys_tmr.h

C

```
typedef void (* SYS_TMR_CALLBACK)(uintptr_t context, uint32_t currTick);
```

Description

SYS TMR Callback Function

Defines a pointer to a client notification function that's called when a timer object times out.

Remarks

The client notification function should be kept as short as possible and non blocking. It is meant for some quick operation (like flag setting) and not for extensive operation. It can be called within interrupt context.

Parameters

Parameters	Description
context	A client parameter that's passed in the callback function. This will help to identify the callback.
currTick	The current system tick when the notification is called.

SYS_TMR_HANDLE Type

Identifies a client timer object.

File

sys_tmr.h

C

```
typedef uintptr_t SYS_TMR_HANDLE;
```

Description

SYS TMR Handle

This handle identifies a registered instance of a client timer object. Access to a timer object parameters and functionality has to use a valid client timer handle.

Remarks

None.

SYS_TMR_INIT Structure

Identifies the system timer initialize structure.

File

sys_tmr.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  SYS_MODULE_INDEX drvIndex;
  uint32_t tmrFreq;
} SYS_TMR_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
SYS_MODULE_INDEX drvIndex;	Driver Module index
uint32_t tmrFreq;	System timer required frequency, Hz This is the base frequency for all time keeping

Description

SYS TMR Initialize structure

This structure identifies the system timer initialize structure.

Remarks

None.

SYS_TMR_HANDLE_INVALID Macro

Identifies the invalid handle of the system timer.

File

sys_tmr.h

C

```
#define SYS_TMR_HANDLE_INVALID ( ( uintptr_t ) -1 )
```

Description

SYS Timer invalid handle macro definition

This enumeration identifies the invalid handle of the system timer.

Remarks

None.

SYS_TMR_INDEX_0 Macro

Timer System Service index definitions.

File

sys_tmr.h

C

```
#define SYS_TMR_INDEX_0 0
```

Description

SYS Timer Module Index Numbers

These constants provide Timer System Service index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

SYS_TMR_FLAGS Enumeration

Defines the types and flags supported by a client timer object.

File

sys_tmr.h

C

```
typedef enum {
  SYS_TMR_FLAG_PERIODIC = 0x0001,
  SYS_TMR_FLAG_SINGLE = 0x0000,
  SYS_TMR_FLAG_AUTO_DELETE = 0x1000
} SYS_TMR_FLAGS;
```

Members

Members	Description
SYS_TMR_FLAG_PERIODIC = 0x0001	the timer is of periodic type as the time-out is achieved the timer is reloaded so it will periodically provide notifications
SYS_TMR_FLAG_SINGLE = 0x0000	the timer is a single shot type it fires just once, when the time-out occurs
SYS_TMR_FLAG_AUTO_DELETE = 0x1000	as time-out occurs, the timer deletes itself only SINGLE and DELAY timer types support this feature

Description

SYS TMR Flags

This enumerated type describes the types and run time flags supported by a client timer object.

Remarks

Multiple flags can be ORed.

SYS_TMR_CLIENT_TOLERANCE Macro

Sets the accepted client tolerance in %.

File

sys_tmr_config_template.h

C

```
#define SYS_TMR_CLIENT_TOLERANCE (10)
```

Description

Client accepted Tolerance configuration

This macro sets the percentage of the accepted tolerance for the requested client delay. If the system timer cannot achieve a delay within the specified tolerance it will reject the client request.

Remarks

This error depends on the actual frequency that the system timer uses as a time base.

SYS_TMR_DRIVER_INDEX Macro

Sets the default timer driver object to be used by the system timer service.

File

sys_tmr_config_template.h

C

```
#define SYS_TMR_DRIVER_INDEX (DRV_TMR_INDEX_0)
```

Description

Default timer driver object configuration

This macro sets the default timer driver object to be used by the system timer service.

Remarks

This value can be overridden by a run time initialization value.

SYS_TMR_FREQUENCY Macro

Specifies the operating frequency for the system timer, Hz.

File

sys_tmr_config_template.h

C

```
#define SYS_TMR_FREQUENCY (1250)
```

Description

System timer running frequency

This macro sets the operating frequency for the system timer, in Hz. All the timing is maintained based on this frequency. This is actually the rate of the interrupt that's requested to the hardware timer that the system timer uses.

The higher the rate, the better time keeping but the operation of the CPU will be interrupted more frequently (if the underlying timer driver is working in interrupts) and the result will be a greater overhead. Therefore is is not recommended to go higher than 1 KHz (below 1 ms) when the timer driver works in interrupts.

Remarks

The system timer cannot obtain timing with a finer resolution than this frequency.

This value can be overridden by a run time initialization value.

The actual running frequency depends on the timer driver running frequency. Select a value that makes sense to minimize the error (see SYS_TMR_FREQUENCY_TOLERANCE notes).

SYS_TMR_FREQUENCY_TOLERANCE Macro

Sets the error tolerance in %.

File

sys_tmr_config_template.h

C

```
#define SYS_TMR_FREQUENCY_TOLERANCE (10)
```

Description

Error Tolerance configuration

This macro sets the accepted tolerance percentage for the requested system timer frequency (SYS_TMR_FREQUENCY).

Remarks

This error depends on the time base that the system timer uses (i.e. the underlying timer driver timing frequency). The actual running frequency for the system timer may be slighlty different from the requested SYS_TMR_FREQUENCY.

SYS_TMR_INTERRUPT_NOTIFICATION Macro

Activates the notification mechanism from within the timer interrupt.

File

sys_tmr_config_template.h

C

```
#define SYS_TMR_INTERRUPT_NOTIFICATION (false)
```

Description

Interrupt Notification Mode configuration

This macro enables/disables the notification towards the timer clients to be delivered directly from the underlying time base interrupt routine(the routine that notifies the system timer itself that another system tick has elapsed).

Enabling interrupt notifications provide a more accurate and responsive callback towards the clients. However the client callbacks need to be non blocking and as short as possible (the client callback are meant for setting a flag to indicate that further processing is needed and not to execute lengthy processing inside the callback itself).

- true - Client notification in interrupt enabled
- false - Client notification in interrupt disabled

Remarks

None.

SYS_TMR_MAX_CLIENT_OBJECTS Macro

Sets the maximum number of timer clients that can be active at a given time.

File

sys_tmr_config_template.h

C

```
#define SYS_TMR_MAX_CLIENT_OBJECTS (5)
```

Description

Maximum client timer objects configuration

This macro sets the maximum number of clients that the Timer system service can support.

Remarks

None.

SYS_TMR_MODULE_INIT Macro

Sets the default module init value by the system timer service.

File

sys_tmr_config_template.h

C

```
#define SYS_TMR_MODULE_INIT (SYS_MODULE_POWER_RUN_FULL)
```

Description

Default module init object configuration

This macro sets the default module init value by the system timer service.

Remarks

This value can be overridden by a run time initialization value.

SYS_TMR_UNIT_RESOLUTION Macro

Specifies the internal units for the time keeping.

File

sys_tmr_config_template.h

C

```
#define SYS_TMR_UNIT_RESOLUTION (10000)
```

Description

System timer internal units resolution

This macro sets the system timer internal time units, in Hz. The higher the value, the better resolution but the range of the delays that can be obtained is shortened.

A value too low will limit the resolution of delays that can be obtained. In order to be able to obtain milliseconds delays the value should be greater than 1000 Hz.

The internal timing calculation is done with integer arithmetic so in order to do a precise enough calculation for ms values, a finer time unit needs to be used.

Remarks

10000 Hz (meaning 0.1 ms units) is the recommended value because the delays than need to be achieved are in the ms range.

The service tries to achieve enough resolution for delays in the range of milliseconds. That means that this value cannot be < 1000 and the service will fail to initilize if such a value is attempted.

The setting affects the range of delays that can be obtained. delay < (2^32 -1)/SYS_TMR_RESOLUTION gives the maximum delay value, in seconds, that can be achieved For example: delay < 119.3 hours (for SYS_TMR_RESOLUTION == 10000).

Files

Files

Name	Description
sys_tmr.h	Timer System Service interface definition.
sys_tmr_config_template.h	Contains configuration definitions that are common to timer system services and aggregates the configuration files for the system services.

Description

This section lists the source and header files used by the library.

sys_tmr.h

Timer System Service interface definition.

Enumerations

	Name	Description
	SYS_TMR_FLAGS	Defines the types and flags supported by a client timer object.

Functions

	Name	Description
	SYS_TMR_CallbackPeriodic	Creates a permanent timer object that calls back to the client periodically.
	SYS_TMR_CallbackSingle	Creates a one-shot/single callback timer object.
	SYS_TMR_CallbackStop	Stops a periodic timer object.
	SYS_TMR_Deinitialize	Deinitializes the specific module instance of the TMR module
	SYS_TMR_DelayMS	Creates a timer object that times out after the specified delay.
	SYS_TMR_DelayStatusGet	Checks the status of the previously requested delay timer object.
	SYS_TMR_Initialize	Initializes hardware and data for the instance of the Timer module and opens the specific module instance.
	SYS_TMR_ModuleStatusGet	Returns status of the specific instance of the Timer module.
	SYS_TMR_ObjectCountGet	Returns the current millisecond count of a timer object.
	SYS_TMR_ObjectCreate	Creates a new timer object.
	SYS_TMR_ObjectDelete	Deletes an existent timer object.
	SYS_TMR_ObjectReload	Reloads an existing timer object.
	SYS_TMR_Status	Returns status of the specific instance of the Timer module.
	SYS_TMR_SystemCountFrequencyGet	Provides the current system count frequency.
	SYS_TMR_SystemCountGet	Provides the current system time count value.
	SYS_TMR_Tasks	Maintains the system timer's state machine and implements its ISR.
	SYS_TMR_TickCounterFrequencyGet	Provides the number of ticks per second
	SYS_TMR_TickCountGet	Provides the current counter value.
	SYS_TMR_TickCountGetLong	Provides the current counter value with 64-bit resolution.

Macros

	Name	Description
	SYS_TMR_HANDLE_INVALID	Identifies the invalid handle of the system timer.
	SYS_TMR_INDEX_0	Timer System Service index definitions.

Structures

	Name	Description
	SYS_TMR_INIT	Identifies the system timer initialize structure.

Types

	Name	Description
	SYS_TMR_CALLBACK	This data type defines a pointer callback function.
	SYS_TMR_HANDLE	Identifies a client timer object.

Description

Timer System Service Interface Definition

This file contains the interface definition for the Timer System Service. It provides a way to interact with the Timer subsystem to manage the timing requests supported by the system.

File Name

sys_tmr.h

Company

Microchip Technology Inc.

sys_tmr_config_template.h

Contains configuration definitions that are common to timer system services and aggregates the configuration files for the system services.

Macros

	Name	Description
	_SYS_TMR_CONFIG_TEMPLATE_H	This is macro _SYS_TMR_CONFIG_TEMPLATE_H.
	SYS_TMR_CLIENT_TOLERANCE	Sets the accepted client tolerance in %.
	SYS_TMR_DRIVER_INDEX	Sets the default timer driver object to be used by the system timer service.
	SYS_TMR_FREQUENCY	Specifies the operating frequency for the system timer, Hz.
	SYS_TMR_FREQUENCY_TOLERANCE	Sets the error tolerance in %.
	SYS_TMR_INTERRUPT_NOTIFICATION	Activates the notification mechanism from within the timer interrupt.
	SYS_TMR_MAX_CLIENT_OBJECTS	Sets the maximum number of timer clients that can be active at a given time.
	SYS_TMR_MODULE_INIT	Sets the default module init value by the system timer service.
	SYS_TMR_UNIT_RESOLUTION	Specifies the internal units for the time keeping.

Description

Timer System Service Configuration Definitions for the Template Version

This file contains configuration definitions that are common to timer system services and aggregates the configuration files for the system services.

File Name

sys_tmr_config_template.h

Company

Microchip Technology Inc.

Touch System Service Library

The Touch System Service is a thin layer specifically for abstracting input from various touch drivers. It is designed to interpret the data from the touch driver and translate it into single-touch behavior that can be used by other modules in the framework, such as the Graphics Library.

The Touch System Service supports both resistive touch driver (ADC10BIT) and capacitive touch drivers (MTCH6301 and MTCH36303).

Currently, the Touch System Service is only available to support non-gestural single finger touch screen input.

Introduction

This library provides a low-level abstraction of the Device Control System Service Library that is available on the Microchip family of PIC32 microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by hiding differences from one microcontroller variant to another.

Description

The Touch System Service is accessed via the Message System Service. For details on how to register to the Message System Service, please refer to its section.

Note: The MPLAB Harmony Graphics Composer is designed to automatically configure the Touch System Service and the Message System Service based on the user's request for touch screen input.

Using the Library

This topic describes the basic architecture of the Touch System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_touch.h`

The interface to the Touch System Service library is defined in the `sys_touch.h` header file.

Please refer to the What is MPLAB Harmony? section for how the Touch System Service interacts with the framework.

Library Overview

A client needs to register a callback to allow the Messaging System Service to send messages of type "TYPE_TOUCHSCREEN" to it. Since it is a dependency, the Messaging System Service is automatically enabled in the MPLAB Harmony Configurator configuration tree when the Touch System Service is enabled.

For detail with how to interface the Message System Service, please refer to the Messaging System Service Library section.

Configuring the Library

No configuration is required for the Touch System Service.

Building the Library

This section lists the files that are available in the Touch System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/touch`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`sys_touch.h`	Touch System Service Library API header file.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/sys_touch.c`	Touch System Service Library implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for the library.

Module Dependencies

The Touch System Service Library is not dependent upon other modules.

Library Interface

a) Functions

	Name	Description
	SYS_TOUCH_CalibrationSet	Sets the calibration values for Touch client specified by moduleIndex.
	SYS_TOUCH_Initialize	Initializes and Enables the Touch Controller.
	SYS_TOUCH_Open	Opens the touch service specified by the moduleIndex and returns a handle to it.
	SYS_TOUCH_TouchMsgGet	Populates the Graphics Object Library (GOL) message structure.
	SYS_TOUCH_Tasks	Maintains the system service's state machine.

b) Data Types and Constants

	Name	Description
	NVM_READ_FUNC	typedef for read function pointer
	NVM_SECTORERASE_FUNC	typedef for sector erase function pointer
	NVM_WRITE_FUNC	typedef for write function pointer
	SYS_TOUCH_HANDLE	Handle for requested Touch interface.
	SYS_TOUCH_INDEX	Identifies the Touch System Service module index definitions.
	SYS_TOUCH_INIT	Identifies the touch attributes supported
	SYS_TOUCH_STATUS	Identifies the current status/state of touch.
	SYS_TOUCH_HANDLE_INVALID	Invalid touch handle.
	TouchShowMessage	macro to draw repeating text
	SYS_INPUT_DEVICE_TYPE	Specifies the different user input devices supported in the library.
	SYS_INPUT_DEVICE_EVENT	Specifies the different user input device events supported in the library.
	_SYS_TOUCH_H	This is macro _SYS_TOUCH_H.

Description

This section describes the APIs of the Touch System Service Library.

a) Functions

SYS_TOUCH_CalibrationSet Function

Sets the calibration values for Touch client specified by moduleIndex.

File

sys_touch.h

C

```
void SYS_TOUCH_CalibrationSet(SYS_MODULE_INDEX moduleIndex, DRV_TOUCH_SAMPLE_POINTS * samplePoints);
```

Returns

none.

Description

This function sets the calibration values provided by samplePoints for the specified client.

Remarks

None.

Preconditions

Function SYS_TOUCH_Open must have been called before calling this function.

Example

```
SYS_MODULE_INDEX moduleIndex;
SYS_TOUCH_SAMPLE_POINTS samplePoints;
SYS_TOUCH_CalibrationSet( moduleIndex,  samplePoints);
```

Parameters

Parameters	Description
moduleIndex	Identifier for the touch to be opened
samplePoints	sample points

Function

void SYS_TOUCH_CalibrationSet(SYS_MODULE_INDEX moduleIndex,
DRV_TOUCH_SAMPLE_POINTS * samplePoints)

SYS_TOUCH_Initialize Function

Initializes and Enables the Touch Controller.

File

sys_touch.h

C

```
SYS_MODULE_OBJ SYS_TOUCH_Initialize(const SYS_MODULE_INDEX moduleIndex, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to the DMA module object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function Enables the Touch module. Enable/Disable stop in idle mode feature based on the passed parameter value.

This routine initializes the Touch module making it ready for clients to open and use it. The initialization data is specified by the init parameter.

Remarks

This routine must be called before any other TOUCH systems service routines are called.

Not all features are available on all devices. Refer to the specific device data sheet to determine availability.

Preconditions

None.

Example

```
SYS_MODULE_OBJ objectHandle;
SYS_TOUCH_INIT touchInit;

objectHandle = SYS_TOUCH_Initialize(SYS_TOUCH_INDEX_1,
                                    (SYS_MODULE_INIT*)touchInit);
if (SYS_MODULE_OBJ_INVALID == objectHandle)
{
    // Handle error
}
```

Parameters

Parameters	Description
init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ SYS_TOUCH_Initialize (const SYS_MODULE_INIT * const init)

SYS_TOUCH_Open Function

Opens the touch service specified by the moduleIndex and returns a handle to it.

File

sys_touch.h

C

```
SYS_TOUCH_HANDLE SYS_TOUCH_Open(SYS_MODULE_INDEX moduleIndex);
```

Returns

If successful, the routine returns a valid open-instance handle (a number identifying both the caller and the module instance).

If an error occurs, the return value is SYS_TOUCH_HANDLE_INVALID.

Description

This function opens the touch service specified by the index and provides a handle that must be provided to all other client-level operations to identify the caller.

Remarks

The handle returned is valid until the SYS_TOUCH_lose routine is called.

Preconditions

The SYS_TOUCH_Initialize function must have been called before calling this function.

Example

```
SYS_TOUCH_HANDLE handle;

handle = SYS_TOUCH_Open(SYS_TOUCH_INDEX_0);
if (SYS_TOUCH_HANDLE_INVALID == handle)
{
    // Unable to open the service
    // May be the service is not initialized or the initialization
    // is not complete.
}
```

Parameters

Parameters	Description
moduleIndex	Identifier for the touch to be opened

Function

SYS_TOUCH_HANDLE SYS_TOUCH_Open(SYS_MODULE_INDEX moduleIndex)

SYS_TOUCH_TouchMsgGet Function

Populates the Graphics Object Library (GOL) message structure.

File

sys_touch.h

C

```
void SYS_TOUCH_TouchMsgGet(SYS_MODULE_INDEX moduleIndex);
```

Returns

None.

Description

This function populates the GOL message structure.

Remarks

None.

Preconditions

None.

Function

void SYS_TOUCH_TouchMsgGet(SYS_MODULE_INDEX moduleIndex)

SYS_TOUCH_Tasks Function

Maintains the system service's state machine.

File

sys_touch.h

C

```
void SYS_TOUCH_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the Touch system service's internal state machine. This function is specifically designed for non interrupt trigger implementations(polling mode), and should be used only in polling mode. this function should be called from the SYS_Tasks() function.

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks).

Preconditions

Touch should have been initialized by calling SYS_Touch_Initialize.

Example

```
// 'object' Returned from SYS_Touch_Initialize

while (true)
{
    SYS_Touch_Tasks ((object) );

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the Touch module (returned from SYS_Touch_Initialize)

Function

void SYS_Touch_Tasks(SYS_MODULE_OBJ object);

b) Data Types and Constants

NVM_READ_FUNC Type

File

sys_touch.h

C

```
typedef uint16_t (* NVM_READ_FUNC)(uint32_t);
```

Description

typedef for read function pointer

NVM_SECTORERASE_FUNC Type

File

sys_touch.h

C

```
typedef void (* NVM_SECTORERASE_FUNC)(uint32_t);
```

Description

typedef for sector erase function pointer

NVM_WRITE_FUNC Type

File

sys_touch.h

C

```
typedef void (* NVM_WRITE_FUNC)(uint16_t, uint32_t);
```

Description

typedef for write function pointer

SYS_TOUCH_HANDLE Type

Handle for requested Touch interface.

File

sys_touch.h

C

```
typedef uintptr_t SYS_TOUCH_HANDLE;
```

Description

Touch System Service handle

A Touch handle is returned by a call to the SYS_TOUCH_Open function. This handle is an identification for the associated touch interface.

The handle once assigned to a client expires when the client calls SYS_TOUCH_Close.

Remarks

None.

SYS_TOUCH_INDEX Enumeration

Identifies the Touch System Service module index definitions.

File

sys_touch.h

C

```
typedef enum {
  SYS_TOUCH_INDEX_0,
  SYS_TOUCH_INDEX_1
} SYS_TOUCH_INDEX;
```

Members

Members	Description
SYS_TOUCH_INDEX_0	First Touch service instance
SYS_TOUCH_INDEX_1	Second Touch service

Description

SYS TOUCH Module Index Numbers

These constants provide the Touch System Service index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals.

These values should be passed into the SYS_TOUCH_Initialize and function to identify the driver instance in use.

SYS_TOUCH_INIT Structure

Identifies the touch attributes supported

File

sys_touch.h

C

```
typedef struct {
  SYS_MODULE_INDEX driverModuleIndex;
} SYS_TOUCH_INIT;
```

Members

Members	Description
SYS_MODULE_INDEX driverModuleIndex;	The Module index of the driver to which touch is attached.

Description

Touch System Service Initialization

This data type defines the touch id and relate with the driver that should be used to initialize touch system.

These functions help to align touch with hardware touch driver specific driver function calls.

Remarks

None.

SYS_TOUCH_STATUS Enumeration

Identifies the current status/state of touch.

File

sys_touch.h

C

```
typedef enum {
  SYS_TOUCH_ERROR = -1,
  SYS_TOUCH_UNINITIALIZED = 0,
  SYS_TOUCH_BUSY = 1,
  SYS_TOUCH_READY = 2
} SYS_TOUCH_STATUS;
```

Members

Members	Description
SYS_TOUCH_ERROR = -1	An unspecified error has occurred.
SYS_TOUCH_UNINITIALIZED = 0	The module has not yet been initialized
SYS_TOUCH_BUSY = 1	An operation is currently in progress
SYS_TOUCH_READY = 2	Any previous operations have succeeded and the service is ready for additional operations

Description

Touch System Service status

Identifies the current status/state of touch.

Remarks

This enumeration is the return type for the status routine.

SYS_TOUCH_HANDLE_INVALID Macro

Invalid touch handle.

File

sys_touch.h

C

```
#define SYS_TOUCH_HANDLE_INVALID (((SYS_TOUCH_HANDLE) -1))
```

Description

Invalid Touch Handle

If the touch system service is unable to allow an additional clients to use it, it must then return the special value SYS_TOUCH_HANDLE_INVALID. Callers should check the handle returned for this value to ensure this value was not returned before attempting to call any other driver routines using the handle.

Remarks

None.

TouchShowMessage Macro

File

sys_touch.h

C

```
#define TouchShowMessage( pStr, color, x, y, width, height) \
                {                                           \
                    GFX_ColorSet(gfxIndex, color);                    \
                    while(GFX_TextStringBoxDraw(gfxIndex, x,y,width,height,pStr, 0, GFX_ALIGN_LEFT) ==
GFX_STATUS_FAILURE);        \
                }
```

Description

macro to draw repeating text

SYS_INPUT_DEVICE_TYPE Enumeration

Specifies the different user input devices supported in the library.

File

sys_touch.h

C

```
typedef enum {
  TYPE_UNKNOWN = 0,
  TYPE_KEYBOARD,
  TYPE_TOUCHSCREEN
} SYS_INPUT_DEVICE_TYPE;
```

Members

Members	Description
TYPE_UNKNOWN = 0	Unknown device.
TYPE_KEYBOARD	Keyboard.
TYPE_TOUCHSCREEN	Touchscreen.

Description

Input Device Type: SYS_INPUT_DEVICE_TYPE

This enumeration specifies the different user input devices supported in the library.

Remarks

Beta. The enum will be relocated to SYS_INPUT when available.

SYS_INPUT_DEVICE_EVENT Enumeration

Specifies the different user input device events supported in the library.

File

sys_touch.h

C

```
typedef enum {
  EVENT_INVALID = 0,
  EVENT_MOVE,
  EVENT_PRESS,
  EVENT_STILLPRESS,
```

```
    EVENT_RELEASE,
    EVENT_KEYSCAN,
    EVENT_CHARCODE,
    EVENT_SET,
    EVENT_SET_STATE,
    EVENT_CLR_STATE
} SYS_INPUT_DEVICE_EVENT;
```

Members

Members	Description
EVENT_INVALID = 0	Invalid event.
EVENT_MOVE	Move event.
EVENT_PRESS	Press event.
EVENT_STILLPRESS	Continuous press event.
EVENT_RELEASE	Release event.
EVENT_KEYSCAN	Key scan event, parameters for the object ID and keyboard scan code will be sent with this event in the GFX_GOL_MESSAGE as parameter.
EVENT_CHARCODE	Character code event. The actual character code will be sent with this event in the GFX_GOL_MESSAGE as parameter.
EVENT_SET	Generic set event.
EVENT_SET_STATE	Generic set state event.
EVENT_CLR_STATE	Generic clear state event.

Description

Input device event: INPUT_DEVICE_EVENT

This enumeration specifies the different user input device events supported in the graphics library.

Remarks

None.

_SYS_TOUCH_H Macro

File

sys_touch.h

C

```
#define _SYS_TOUCH_H
```

Description

This is macro _SYS_TOUCH_H.

Files

Files

Name	Description
sys_touch.h	Touch System Service Implementation.

Description

This section lists the source and header files used by the library.

sys_touch.h

Touch System Service Implementation.

Enumerations

	Name	Description
	SYS_INPUT_DEVICE_EVENT	Specifies the different user input device events supported in the library.
	SYS_INPUT_DEVICE_TYPE	Specifies the different user input devices supported in the library.
	SYS_TOUCH_INDEX	Identifies the Touch System Service module index definitions.

	SYS_TOUCH_STATUS	Identifies the current status/state of touch.

Functions

	Name	Description
	SYS_TOUCH_CalibrationSet	Sets the calibration values for Touch client specified by moduleIndex.
	SYS_TOUCH_Initialize	Initializes and Enables the Touch Controller.
	SYS_TOUCH_Open	Opens the touch service specified by the moduleIndex and returns a handle to it.
	SYS_TOUCH_Tasks	Maintains the system service's state machine.
	SYS_TOUCH_TouchMsgGet	Populates the Graphics Object Library (GOL) message structure.

Macros

	Name	Description
	_SYS_TOUCH_H	This is macro _SYS_TOUCH_H.
	SYS_TOUCH_HANDLE_INVALID	Invalid touch handle.
	TouchShowMessage	macro to draw repeating text

Structures

	Name	Description
	SYS_TOUCH_INIT	Identifies the touch attributes supported

Types

	Name	Description
	NVM_READ_FUNC	typedef for read function pointer
	NVM_SECTORERASE_FUNC	typedef for sector erase function pointer
	NVM_WRITE_FUNC	typedef for write function pointer
	SYS_TOUCH_HANDLE	Handle for requested Touch interface.

Description

Touch System Service Implementation

The Touch System Service provides a simple interface to manage the touch screen drivers. This file implements the core interface routines for the Touch System Service. This is a resistive touch screen driver that is using the Microchip Graphics Library. The calibration values are automatically checked (by reading a specific memory location on the non-volatile memory) when initializing the module if the function pointers to the read and write callback functions are initialized. If the read value is invalid calibration will automatically be executed. Otherwise, the calibration values will be loaded and used. The driver assumes that the application side provides the read and write routines to a non-volatile memory. If the callback functions are not initialized, the calibration routine will always be called at startup to initialize the global calibration values. This driver assumes that the Graphics Library is initialized and will be using the default font of the library.

File Name

sys_touch.c

Company

Microchip Technology Inc.

Watchdog Timer System Service Library

This section describes the Watchdog Timer (WDT) System Service Library.

Introduction

This library provides a low-level abstraction of the Watchdog Timer System Service Library that is available on the Microchip family of microcontrollers with a convenient C language interface. It can be used to simplify low-level access to the module without the necessity of interacting directly with the module's registers, there by abstracting differences from one microcontroller variant to another.

Description

The primary function of the Watchdog Timer (WDT) is to reset the microcontroller, in the event of a software malfunction, by resetting the device if it has not been cleared by software. To ensure that application does not hang, the application is required to reset the timer periodically. It can also be used to wake the device from Sleep or Idle mode. The WDT is a free-running timer which uses the low-power RC oscillator and requires no external components. Therefore, the WDT will continue to operate even if the system’s primary clock source is stopped.

Using the Library

This topic describes the basic architecture of the Watchdog Timer System Service Library and provides information and examples on its use.

Description

Interface Header File: `sys_wdt.h`

The interface to the Watchdog Timer System Service library is defined in the `sys_wdt.h` header file. This file is included by the `sys.h` file. Any C language source (`.c`) file that uses the Watchdog Timer System Service library should include `sys.h`.

Please refer to the What is MPLAB Harmony? section for how the System Service interacts with the framework.

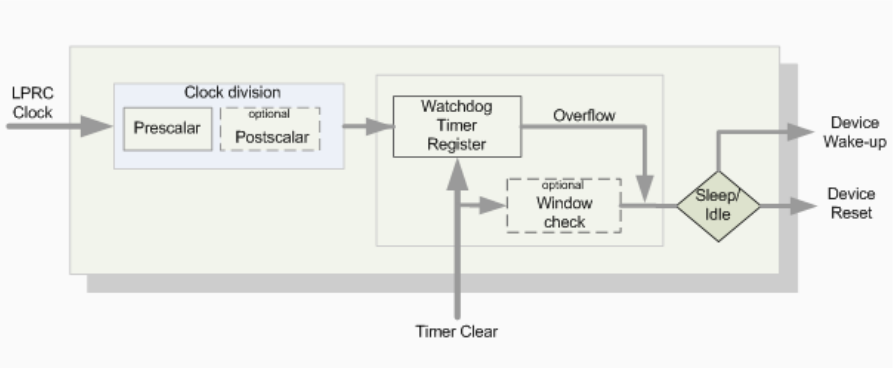
Abstraction Model

This library provides a low-level abstraction of the Watchdog Timer System Service Library. This topic describes how that abstraction is modeled in software and introduces the library's interface.

Description

The Watchdog Timer uses the internal Low-Power RC (LPRC) Oscillator as the source of clock. The clock is divided by the configured prescaler value. There may be one more postscaler divisors and these should be set through the Configuration bits. The divided clock is then used to increment a counter. If the software does not clear the counter in time, the counter overflows and that will result in reset in normal mode. In Sleep or Idle mode, the overflow will result in a device wake-up. In Windowed mode, resetting the counter when the count is not in the specified window will also lead to a reset.

Watchdog Timer Software Abstraction Block Diagram



Library Overview

Refer to the System Service Introduction section for how the system services operates in a system.

The library interface routines are divided into various sub-sections, which address one of the blocks of the overall operation of the Watchdog Timer module.

Library Interface Section	Description
Enable and Disable Functions	APIs to enable and disable the Watchdog Timer
Clearing the Timer Functions	API to clear the Watchdog Timer

How the Library Works

Provides information on how the library works.

Description

Watchdog Timer Initialization:

For devices that have Windowed mode, use the argument to control the mode. The argument will be discarded for devices that do not have Windowed mode.

```
//Enable the Watchdog Timer with Window mode enabled
SYS_WDT_Enable(true);

//Enable the Watchdog Timer with Window mode disabled
SYS_WDT_Enable(false);
```

```
//Some code
SYS_WDT_Disable();
```

Note: The previous code example will not work if the Watchdog Timer is controlled through the Configuration bits.

Service the Watchdog Timer:

To ensure the normal operation of the system, the software must periodically clear the Watchdog Timer.
`SYS_WDT_TimerClear();`

Note: Not all modes are available on all devices. Please refer to the specific device data sheet to determine the supported modes.

Configuring the Library

Macros

	Name	Description
	WDT_PLIB_ID	Configures the module's ID used by the peripheral library.

Description

The configuration of the Watchdog Timer System Service is based on the file `system_config.h`.

This header file contains the configuration selection for the Watchdog Timer System Service. Based on the selections made, the Watchdog Timer System Service may support the selected features. These configuration settings will apply to all instances of the Watchdog Timer System Service.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

WDT_PLIB_ID Macro

Configures the module's ID used by the peripheral library.

File

sys_wdt_config.h

C

```
#define WDT_PLIB_ID WDT_ID_1
```

Description

Watchdog timer module ID used by the Peripheral Library

This macro configures the module's ID used by the peripheral library.

Remarks

None.

Building the Library

This section lists the files that are available in the Watchdog Timer System Service Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/system/wdt`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`sys_wdt.h`	Watchdog Timer System Service Library API header file.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/sys_wdt.c	Watchdog Timer System Service Library implementation.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Watchdog Timer System Service is not dependent upon other modules.

Library Interface

a) Enable and Disable Functions

	Name	Description
	SYS_WDT_Disable	Disables the WDT if it is enabled in software.
	SYS_WDT_Enable	Enables the WDT. The argument 'windowModeEnable' will be used only for those devices that support 'window mode'. Otherwise, it will be discarded.

b) Clearing the Timer Functions

	Name	Description
	SYS_WDT_TimerClear	Reset the WDT.

Description

This section describes the Application Programming Interface (API) functions of the Watchdog Timer System Service.

Refer to each section for a detailed description.

a) Enable and Disable Functions

SYS_WDT_Disable Function

Disables the WDT if it is enabled in software.

File

sys_wdt.h

C

```
void SYS_WDT_Disable();
```

Returns

None.

Description

This function disables the WDT if it is enabled in software. If the WDT is enabled through 'configuration bits' it cannot be disabled using this function.

Remarks

The example code doesn't include the settings that should be done through configuration bits.

Preconditions

The WDT should be disabled through 'configuration bits'.

Example

```
SYS_WDT_Disable();
```

Function

void SYS_WDT_Disable (void)

SYS_WDT_Enable Function

Enables the WDT. The argument 'windowModeEnable' will be used only for those devices that support 'window mode'. Otherwise, it will be discarded.

File

sys_wdt.h

C

```
void SYS_WDT_Enable(bool windowModeEnable);
```

Returns

None.

Description

This function enables the WDT. The argument 'windowModeEnable' will be used only for those devices that support 'window mode'. Otherwise, the argument will be discarded. This function could be called multiple times to enable/disable the 'window mode'.

Remarks

Calling this function is not necessary if the WDT is enabled through Configuration bits.

Preconditions

None.

Example

```
bool windowModeEnable = true;

SYS_WDT_Enable(windowModeEnable);
```

Parameters

Parameters	Description
windowModeEnable	A flag indicates whether to enable/disable the 'window mode'. • true - Enable the 'window mode' • false - Disable the 'window mode'

Function

void SYS_WDT_Enable (bool windowModeEnable)

b) Clearing the Timer Functions

SYS_WDT_TimerClear Function

Reset the WDT.

File

sys_wdt.h

C

```
void SYS_WDT_TimerClear();
```

Returns

None.

Description

This function clears the WDT counter. The WDT should be cleared periodically before the count overflows and forces the device to Reset.

Remarks

Clearing the WDT before the count reaches the window, will cause a reset in Windowed mode.

The example code doesn't include the settings that should be done through Configuration bits.

This feature is not available on all devices. Please refer to the specific device data sheet to determine whether this feature is supported.

Preconditions

None.

Example

```
//Application loop
while(1)
{
    SYS_WDT_TimerClear();
    //user code
}
```

Function

void SYS_WDT_TimerClear (void)

Files

Files

Name	Description
sys_wdt.h	Watchdog Timer (WDT) System Service interface definition.
sys_wdt_config.h	Watchdog Timer (WDT) System Service interface definition.

Description

This section lists the source and header files used by the library.

sys_wdt.h

Watchdog Timer (WDT) System Service interface definition.

Functions

	Name	Description
	SYS_WDT_Disable	Disables the WDT if it is enabled in software.
	SYS_WDT_Enable	Enables the WDT. The argument 'windowModeEnable' will be used only for those devices that support 'window mode'. Otherwise, it will be discarded.
	SYS_WDT_TimerClear	Reset the WDT.

Description

Watchdog Timer System Service Interface Definition

This file contains the interface definition for the WDT System Service. It provides a way to interact with the WDT subsystem to manage the timing requests supported by the system.

File Name

sys_wdt.h

Company

Microchip Technology Inc.

sys_wdt_config.h

Watchdog Timer (WDT) System Service interface definition.

Macros

	Name	Description
	WDT_PLIB_ID	Configures the module's ID used by the peripheral library.

Description

Watchdog Timer System Service Interface Definition

This file contains the interface definition for the WDT System Service. It provides a way to interact with the Watchdog timer subsystem to manage the timing requests supported by the system

File Name

sys_wdt_config.h

Company

Microchip Technology Inc.

TCP/IP Stack Libraries Help

This section provides information on the TCP/IP Stack Library and its modules that are available in MPLAB Harmony.

TCP/IP Stack Library Overview

This section provides an overview of the TCP/IP Stack libraries that are available in MPLAB Harmony.

Introduction

This topic provides an overview of the TCP/IP Stack in MPLAB Harmony.

Description

The MPLAB Harmony TCP/IP Stack provides a foundation for embedded network applications by handling most of the interaction required between the physical network port and your application. It includes modules for several commonly used application layers, including HTTP for serving web pages, SMTP for sending e-mails, SNMP for providing status and control, Telnet, TFTP, Serial-to-Ethernet, and much more. In addition, the stack includes light-weight and high-performance implementations of the TCP and UDP transport layers, as well as other supporting modules such as IP, ICMP, DHCP, ARP, and DNS.

Network Metrics

This topic provides TCP/IP network metrics information for PIC32MX and PIC32MZ devices.

Description

Network Metrics for the PIC32MX

Test Setup:

- PIC32 MZ Embedded Connectivity Starter Kit
- PIC32 MX Ethernet Starter Kit II
- Starter Kit I/O Expansion Board
- Fast 100 Mbps Ethernet PICtail Plus
- Windows 7 PC
- TX buffer: 7350 bytes
- RX buffer: 4400 bytes

Iperf Bandwidth Measurements

All bandwidth numbers are in kilobits per second.

Table 1: Internal MAC Measurements for PIC32MX Devices

Type	-O0	-O1	-O2	-O3	-Os
TCP Client	25,963	47,652	53,650	56,682	50,191
TCP Server	29,046	45,193	50,484	52,603	45,659
UDP Client	33,795	60,997	71,105	72,049	63,298
UDP Server	36,451	78,546	86,994	88,757	80,948

Table 2: Internal MAC Measurements for PIC32MZ Devices

Type	-O0	-O1	-O2	-O3	-Os
TCP Client	31,722	56,929	61,361	64,479	59,742
TCP Server	28,924	41,570	44,168	44,721	42,437
UDP Client	45,534	80,116	87,043	89,377	84,044
UDP Server	53,155	90,766	95,943	95,899	94,411

Table 3: ENCx24J600 MAC Measurements for PIC32MX Devices

Type	-O0	-O1	-O2	-O3	-Os
TCP Client	6,023	7,225	8,211	7,633	7,373
TCP Server	5,760	7,476	8,118	8,166	7,588

UDP Client	7,035	8,826	9,159	9,148	8,949
UDP Server	7,849	9,651	10,358	10,371	9,722

Table 4: Internal MAC Measurements for PIC32MX + MIPS16 Devices

Type	-O0	-O1	-O2	-O3	-Os
TCP Client	19,748	29,707	32,488	33,920	30,206
TCP Server	17,369	23,294	25,945	28,018	23,421
UDP Client	23,901	34,285	38,598	51,959	36,252
UDP Server	27,720	37,914	39,186	60,143	37,401

Table 5: ENCx24J600 MAC Measurements for PIC32MX + MIPS16 Devices

Type	-O0	-O1	-O2	-O3
TCP Client	5,480	6,232	6,985	6,345
TCP Server	4,984	6,262	6,454	6,644
UDP Client	6,370	7,727	8,069	8,586
UDP Server	7,109	8,471	8,702	9,289

Commands

The following commands were used to generate the network performance numbers as measured with Iperf:

UDP/IP Client:

PIC32 Board	Personal Computer
`iperf -c <PC IP Address> -u -b 100M -i 5 -t 20`	`iperf -s -u -i 5`

UDP/IP Server:

PIC32 Board	Personal Computer
`iperf -s -u -i 5`	`iperf -c mchpboard_e -u -b 100M -i 5 -t 20`

TCP/IP Client:

PIC32 Board	Personal Computer
`iperf -c <PC IP Address> -t 10 -i 5 -l 1024 -x 100M -M 1460`	`iperf -s -i 5 -m -w 10K –N`

TCP/IP Server:

PIC32 Board	Personal Computer
`iperf -s -i 5 -x 100M`	`iperf -c mchpboard_e -t 10 -i 5 -l 1024`

Flash and RAM Usage

Provides information on Flash and RAM usage by device family.

Description

Flash and RAM Usage

Table 1 and Table 2 provide Flash and RAM usage for PIC32MX and PIC32MZ devices, respectively.

Table 1: PIC32MX Device Flash and Ram Usage

Optimization	0	0	1	1	2	2	3	3	s	s
Module	**Flash**	**RAM**	**Flash**	**RAM**	**Flash**	**RAM**	**Flash**	**RAM**	**Flash**	**RAM**
ARP	12,836	136	6,992	136	7,332	136	7,548	136	6,640	136
Berkeley API	19,000	76	11,080	76	10,620	76	12,368	76	9,196	76
DDNS	7,332	630	4,688	618	4,092	618	4,636	568	3,644	618

DHCP Client	11,104	76	6,444	76	6,604	76	7,376	80	5,936	76
DHCP Server	17,816	100	9,128	100	9,124	100	10,216	100	8,552	100
DNS Client	16,724	148	9,020	148	8,960	148	11,320	148	7,844	148
DNS Server	13,192	280	7,156	280	6,528	280	7,348	284	5,948	280
FTP Server	20,548	2,009	11,376	1,989	11,360	1,989	11,628	1,993	10,728	1,989
HTTP Server	18,988	2,320	10,052	2,320	10,264	2,276	11,404	2,264	9,024	2,276
ICMP (IPv4)	2,416	28	1,260	28	1,248	28	1,272	28	1,180	28
ICMPv6 (IPv6)	12,464	28	7,320	28	7,292	28	8,012	28	6,724	28
IPv4	5,144	52	2,764	52	2,864	52	2,996	52	2,652	52
IPv6	46,756	544	28,768	544	29,304	544	41,432	544	25,712	528
NetBIOS Name Server	2,244	16	1,164	16	1,136	16	1,956	16	1,128	16
SMTP Client	7,996	262	4,976	266	4,808	266	5,964	268	4,168	266
SNTP Client	3,332	182	2,232	182	2,256	182	2,288	184	2,064	182
TCP	37,736	48	20,096	48	21,096	56	24,088	60	18,524	56
Announce Support	3,488	44	1,972	44	2,004	44	2,168	44	1,916	44
Reboot Server	756	26	412	26	436	26	436	26	420	26
UDP	22,984	48	11,748	48	12,556	48	14,588	56	10,780	48
Zeroconf Local Link	188	5	140	5	152	5	152	5	132	5
Zeroconf MDNS (Bonjour Service)	4,764	24	2,600	24	2,628	24	2,688	24	2,428	24
Stack Core Library	63,368	1,776	31,168	1,768	32,848	1,768	41,112	1,768	28,132	1,768
File System Wrappers	684	16	424	16	420	16	656	16	372	16
Ethernet MAC Driver	23,412	524	12,924	524	12,756	524	14,536	524	11,576	524
Ethernet PHY Driver	17,820	296	9,580	296	10,804	296	10,924	280	9,904	296
wolfSSL Library	237,940	25,596	130,040	25,300	129,024	28,868	178,844	28,828	110,772	28,868

Table 2: PIC32MZ Device Flash and Ram Usage

Optimization	**0**	**0**	**1**	**1**	**2**	**2**	**3**	**3**	**s**	**s**
Module	**Flash**	**RAM**	**Flash**	**RAM**	**Flash**	**RAM**	**Flash**	**RAM**	**Flash**	**RAM**
ARP	12,876	168	6,920	168	7,328	168	7,580	168	6,640	168
Berkeley API	19,016	76	11,028	76	10,600	76	12,372	76	9,196	76
DDNS	7,332	630	4,676	618	4,024	618	4,628	568	3,636	618
DHCP Client	11,164	120	6,448	120	6,528	120	7,512	124	6,020	120
DHCP Server	17,980	248	9,120	248	9,200	212	10,212	212	8,312	212
DNS Client	16,752	148	8,932	148	8,944	148	11,352	148	7,912	148
DNS Server	13,224	280	7,020	280	6,640	280	7,556	284	6,000	280
FTP Server	20,676	2,009	11,384	1,989	11,336	1,989	11,684	1,993	10,720	1,989
HTTP Server	19,164	2,516	10,144	2,516	10,340	2,472	11,472	2,460	9,112	2,472
ICMP (IPv4)	2,416	28	1,252	28	1,248	28	1,272	28	1,188	28
ICMPv6 (IPv6)	12,464	28	7,300	28	7,108	28	7,960	28	6,744	28
IPv4	5,176	100	2,772	100	2,892	100	3,024	100	2,720	100
IPv6	46,836	544	28,544	544	29,300	544	41,740	544	25,680	528
NetBIOS Name Server	2,252	16	1,168	16	1,128	16	1,956	16	1,128	16
SMTP Client	7,996	262	4,908	266	4,784	266	5,944	268	4,136	266
SNTP Client	3,332	182	2,208	182	2,256	182	2,284	184	2,072	182
TCP	37,928	116	19,904	116	21,016	124	24,096	128	18,652	124
Announce Support	3,488	44	1,964	44	2,004	44	2,144	44	1,892	44

Reboot Server	788	58	460	58	476	58	476	58	460	58
UDP	23,116	144	11,704	144	12,524	144	14,600	152	10,920	144
Zeroconf Local Link	4,836	120	2,664	120	2,704	120	152	5	128	5
Zeroconf MDNS (Bonjour Service)	16,812	132	9,368	132	9,628	132	2,760	120	2,492	120
Stack Core Library	65,400	2,716	32,064	2,708	34,304	2,708	42,308	2,704	29,180	2,704
File System Wrappers	684	16	400	16	420	16	668	4	140	4
Ethernet MAC Driver	23,736	604	13,096	604	13,264	604	8,288	600	6,796	600
Ethernet PHY Driver	17,824	300	9,552	300	10,868	300	11,116	280	9,936	300
wolfSSL Library (see **Note**)	237,940	25,596	130,424	25,300	129,276	28,868	176,976	28,880	110,796	28,868

Note: wolfSSL is built with the following features:

- SSL 3.0, TLS 1.0, TLS 1.1, TLS 1.2
- Ciphers: AES, ARC4, DES, DH, DSA, HMAC, rabbit, RSA
- Hashes: md4, md5, sha, sha256

Getting Help

This topic provides information for requesting support assistance with the TCP/IP Stack.

Description

The TCP/IP Stack is supported through Microchip's standard support channels. If you encounter difficulties, you may submit ticket requests at http://support.microchip.com.

The Microchip forums are also an excellent source of information, with a very lively community dedicated specifically to Ethernet and TCP/IP discussions at http://forum.microchip.com.

Microchip also offers embedded network classes through Regional Training Centers. For more information, visit http://techtrain.microchip.com/rtcv2/.

Building the Library

This section lists the files that are available in the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	File that includes all of the TCP/IP modules.

Required File(s)

MHC ***All of the required files listed in the following table are automatically loaded into the MPLAB X IDE project by the MHC.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/common/big_int.c`	Library for integers greater than 32 bits.
`/src/common/hashes.c`	Utility file for MD5 and SHA-1 hash generation.
`/src/common/helpers.c`	Helper functions for string manipulation and binary to string.
`/src/common/lfsr.c`	Simple Linear Feedback Shift Register implementation.
`/src/hash_fnv.c`	FNV Hash routines

/src/ohash.c	Hash Table manipulation routines.
/src/tcpip_heap_alloc.c	Memory heap management for the TCP/IP Stack Library.
/src/tcpip_helpers.c	Network string helper functions (i.e., IP address string to IP).
/src/tcpip_managers.c	Central manager for TCP/IP Stack Library.
/src/tcpip_notify.c	Functions to handle the notification heap.
/src/tcpip_packet.c	Functions for managing the preallocated packet memory.
/src/system/system_debug.c	Temporary implementation of the system debug server.
/src/system/drivers/db_appio.c	Temporary driver implementation for application debug input/output.
/src/system/drivers/usart.c	Temporary driver implementation for the UART.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
/src/system/system_commands.c	Temporary implementation of the debug command processor.

Module Dependencies

The TCP/IP Stack Library depends on the following modules:

- Ethernet MAC Driver Library (if using the Ethernet interface)
- Ethernet PHY Driver Library (if using the Ethernet interface)
- MRF24WG Wi-Fi Driver Library (if using the Wi-Fi interface)
- SPI Driver Library (if using the Wi-Fi interface)
- Operating System Abstraction Layer (OSAL) Library Help
- Clock System Service Library
- System Service Library
- Console System Service Library
- File System Service Library
- Interrupt System Service Library
- Timer System Service Library
- Random Number Generator System Service Library
- Debug System Service Library

TCP/IP Stack Library Porting Guide

This section provides porting information for the TCP/IP Stack libraries.

Introduction

This section provides information for porting from a previous version of the TCP/IP Stack to the TCP/IP Stack within MPLAB Harmony.

Upgrading from the V5 TCP/IP Stack to the MPLAB Harmony TCP/IP Stack

The MPLAB Harmony TCP/IP Stack is a completely new distribution, which is included as part of the MPLAB Harmony installation. No files need to be maintained from an existing V5 TCP/IP Stack installation.

MPLAB Harmony TCP/IP Stack Key Features

This topic provides a description of some of the key features of the MPLAB Harmony TCP/IP Stack that are not available in the V5 version of the stack.

Description

Note: The PIC18 device family is not supported by the MPLAB Harmony TCP/IP Stack.

The following are the key features of the MPLAB Harmony TCP/IP Stack:

- Multiple interfaces - Ethernet and/or Wi-Fi interfaces are supported

- Dual stack with IPv4 and/or IPv6 support:
- UDP and TCP sockets are capable of supporting either/or IPv4/IPv6 connections
- Fully dynamic:
- Stack initialization/deinitialization
- Up/down interface
- Resource management
- Module configuration
- Improved modularity and stack layout:
- The stack uses the drivers and services provided by the MPLAB Harmony framework (e.g., the SPI driver for the Wi-Fi interface). This improvement offers better isolation, modularity, and maintainability of the stack.
- Run-time configuration through the TCP/IP console:
- The stack uses the System Console and Debug system services
- Interrupt driven operation
- RTOS friendly, with easy RTOS integration:
- OSAL calls are added in all stack modules for multi-threading support
- The BSD layer compatibility has been greatly improved, new dynamic socket options have been added as well as IPv6 support.
- Run time benchmarking support using standard industry tools - Iperf
- The MPLAB Harmony file system (SYS FS) is used by the HTTP, FTP, SNMP, modules, among others. This makes the stack independent of the specific file structure of a particular media and allows web page storage on a multitude of storage media.
- The stack uses the high performance MPLAB Harmony Cryptographic Library, which is a very efficient and modern implementation supporting the latest encryption and key exchanges algorithms
- The stack security has been greatly improved by adding support of the modern wolfSSL TLS libraries

All of these features bring flexibility and many new options that were not present in the previous V5 MLA TCP/IP implementation.

Please note that a direct comparison of the code size between demo applications using the V5 and the MPLAB Harmony TCP/IP stack can be misleading because of the multitude of the extra options, features, and services that are included as part of a standard Harmony distribution. Many of the features can be disabled from the build of the stack and the comparison of the results will be more realistic.

MPLAB Harmony TCP/IP Stack Structure

This topic describes the structure of the MPLAB Harmony TCP/IP Stack.

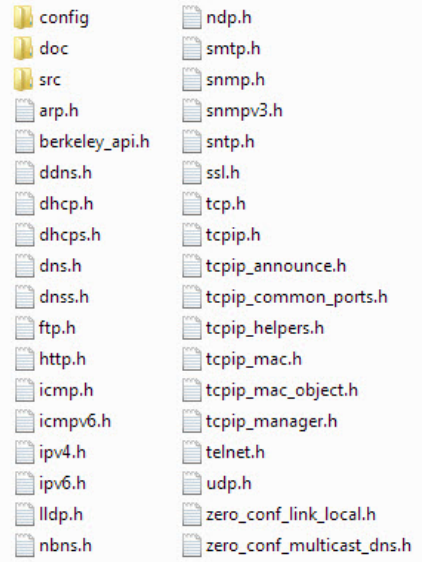
Description

The MPLAB Harmony TCP/IP Stack consists of the following structure:

- Lowercase file names
- Underscores are used (e.g., `tcpip_manager.h`)
- Private headers are no longer exposed and have been moved to the source folder
- System services have been removed from the stack, such as:
- Interrupts
- Drivers
- Timer services
- File system
- Board Support Package (BSP)

By default, the TCP/IP Stack is placed into the following location during installation of MPLAB Harmony on a Windows® system: `C:\Microchip\harmony\<version>\framework\tcpip`.

The following figure shows the list of files in the `tcpip` folder.



MPLAB Harmony TCP/IP Stack Design

This topic discusses the design of the MPLAB Harmony TCP/IP Stack.

Description

The MPLAB Harmony TCP/IP Stack is designed as a part of a system running other applications, middleware, etc. Therefore, it makes use of the system services that are available to other modules in the system or to the application, such as:

- File system
- System interrupts
- System driver services
- System timers
- System device drivers
- System command processor
- System console
- System debug services

Refer to the `./framework/tcpip/src/system` folder for the header files that expose the system wide available API.

MPLAB Harmony TCP/IP Stack API Changes

This topic discusses the reasons behind the API changes to the MPLAB Harmony TCP/IP Stack.

Description

For most modules, the API should be backward compatible to the V5 TCP/IP Stack.

The changes in the stack API have been minimized so that the porting process is straightforward. However, new functionality has been added because of new features, such as IPv6 support, multiple network interfaces, and the dynamic configuration of the stack. Other than these new features, the changes to the API were only made when an existing V5 TCP/IP Stack function did not support the required parameters/flexibility, or they were confusing.

The API changes are at the TCP, UDP, ARP, HTTP, SMTP, and stack access and initialization level.

TCP Changes

This topic discusses the changes made to the `tcp.h` file.

Description

TCP Changes

The TCP/IP Stack include header file, `tcp.h`, has been updated as follows.

- *DNS Resolution* - DNS resolution is no longer performed automatically when opening a TCP socket. The TCP layer takes only an IP address as an input parameter. If you need a DNS resolution, you have to perform it before opening a socket (see `dns.h`).
- *IPv6 Support* - Support for IPv6 has been added

Opening TCP Sockets

To open TCP sockets, the required functions are now:

```
TCP_SOCKET TCPIP_TCP_ServerOpen(IP_ADDRESS_TYPE addType, TCP_PORT localPort, IP_MULTI_ADDRESS* localAddress)
```

and

```
TCP_SOCKET TCPIP_TCP_ClientOpen(IP_ADDRESS_TYPE addType, TCP_PORT remotePort, IP_MULTI_ADDRESS*
remoteAddress)
```

These two new functions replace the V5 TCP/IP Stack TCPOpen function. The application code must replace the calls to TCPOpen with TCP_SOCKET TCPIP_TCP_ServerOpen or TCP_SOCKET TCPIP_TCP_ClientOpen, as appropriate.

The new calls add the possibility to use IPV6 addresses and make clear the choice of parameters for server or client sockets. The parameters are:

typedef union

```
{
IPV4_ADDR v4Add;
IPV6_ADDR v6Add;
}IP_MULTI_ADDRESS;
```

typedef enum

```
{
IP_ADDRESS_TYPE_ANY = 0, // either IPv4 or IPv6; unspecified;
IP_ADDRESS_TYPE_IPV4 = 1, // IPv4 address type
IP_ADDRESS_TYPE_IPV6 // IPv6 address type
}IP_ADDRESS_TYPE;
```

Note: Currently, `IP_ADDRESS_TYPE_ANY` is only supported for server sockets.

Disconnecting a Socket

The functions to disconnect a socket have changed to include added functionality:

```
TCPIP_TCP_Disconnect(TCP_SOCKET hTCP)
```

This function closes the TX side of a connection by sending a FIN (if currently connected) to the remote node of the connection. It no longer sends a RST signal to the remote node so that a sequence of two sequential calls to the function is no longer needed.

If the socket has the Linger option set (default), the queued TX data transmission will be attempted before sending the FIN. If the Linger option is off, the queued TX data will be discarded.

No more data can be sent by this socket, but more data can be received (the socket will be eventually closed when a FIN is received from the remote node or by a time-out dictated by the TCP_FIN_WAIT_2_TIMEOUT value in `tcp_config.h`).

Please note that this call may fail in which case it can be reissued.

Setting the socket options is done using the function TCPIP_TCP_OptionsSet.

```
TCPIP_TCP_Abort(TCP_SOCKET hTCP, bool killSocket)
```

This function aborts a connection to a remote node by sending a RST (if currently connected). Any pending TX/RX data is discarded.

A client socket will always be closed and the associated resources released. The socket cannot be used again after this call.

A server socket will abort the current connection if:

- `killSocket == false` (the socket will remain listening)
- `killSocket == true` (the socket will be closed and all associated resources released. The socket cannot be used again after this call.)

```
TCPIP_TCP_Close(TCP_SOCKET hTCP)
```

This function disconnects an open socket and destroys the socket handle, releasing the associated resources.

If the graceful option is set for the socket (default) a `TCPIP_TCP_Disconnect` will be tried (FIN will be sent):

- If the linger option is set (default) the `TCPIP_TCP_Disconnect` will attempt to send any queued TX data before issuing the FIN
- If the FIN send operation fails or the socket is not connected, an abort is generated
- If the graceful option is not set, or the previous step could not send the FIN:
- A TCPIP_TCP_Abort(is called, sending a RST to the remote node and communication is closed. The socket is no longer valid and the associated resources are freed.

Setting the socket options is done using the call TCPIP_TCP_OptionsSet();.

Adjusting Socket Size

The function to adjust the size of a socket's RX and TX buffers has changed to include added functionality:

```
TCPIP_TCP_FifoSizeAdjust(TCP_SOCKET hTCP, uint16_t wMinRXSize, uint16_t wMinTXSize, TCP_ADJUST_FLAGS vFlags)
```

The TX and RX FIFOs (buffers) associated with a socket are now completely separate and independent.

Two new flags, TCP_ADJUST_TX_ONLY and TCP_ADJUST_RX_ONLY, have been added, which allow changing the size of TX and RX buffers independently. This is the preferred option.

However, when either flag is not set, for the purpose of this function, the TX and RX FIFOs are considered to be contiguous so that the total FIFO space is divided between the TX and RX FIFOs. This provides backward compatibility with previous versions of this function.

Note: The TX or RX associated buffer sizes can be independently changed too using the socket options. See the `TCPIP_TCP_OptionsSet` function for more information.

UDP Changes

This topic discusses the changes made to the udp.h file.

Description

UDP Changes

The TCP/IP Stack include header file, udp.h, has been updated as follows.

- *DNS Resolution* - DNS resolution is no longer performed automatically when opening a UDP socket. The UDP layer takes only an IP address as an input parameter. If you need a DNS resolution, you have to perform it before opening a socket (see dns.h).
- *IPv6 Support* - Support for IPv6 has been added

Opening UDP Sockets

To open UDP sockets, the required functions are now:

```
UDP_SOCKET TCPIP_UDP_ServerOpen(IP_ADDRESS_TYPE addType, UDP_PORT localPort, IP_MULTI_ADDRESS*
localAddress);
```

and

```
UDP_SOCKET TCPIP_UDP_ClientOpen(IP_ADDRESS_TYPE addType, UDP_PORT remotePort, IP_MULTI_ADDRESS*
remoteAddress);
```

These two new functions replace the V5 TCP/IP Stack UDPOpen function.

The application code must replace any calls to UDPOpen with UDP_SOCKET TCPIP_UDP_ServerOpen or UDP_SOCKET TCPIP_UDP_ClientOpen, as appropriate.

Note: Currently, IP_ADDRESS_TYPE_ANY is only supported for server sockets.

ARP Changes

This topic discusses the changes made to the arp.h file.

Description

ARP Changes

The header file, arp.h, has been updated as follows.

The ARP module has been redesigned to support multiple network interfaces and to implement internal storage (caches) per interface for eliminating the need for frequent access to the network.

Some of the most important changes include:

- Manipulation/control of the cache entries (TCPIP_ARP_EntryGet, TCPIP_ARP_EntrySet, TCPIP_ARP_EntryRemove) with permanent entries support
- Dynamic notification mechanism (TCPIP_ARP_HandlerRegister, TCPIP_ARP_HandlerDeRegister) for signaling of the ARP events
- Resolution calls (TCPIP_ARP_Resolve, TCPIP_ARP_Probe, TCPIP_ARP_IsResolved)

Normally, the application does not need access to the ARP module. The address resolution is performed internally by the stack.

The ARP module cache manipulation access is meant for TCP/IP Stack control and configuration depending on the actual network topology.

Berkeley Changes

This topic discusses the changes made to the Berkeley API file.

Description

The Berkeley API include file has been renamed to berkeley_api.h (formerly BerkeleyAPI.h). The following changes are included:

- IPv6 Support – Support for IPv6 has been added
- setsockopt, getsockopt, gethostbyname have been added

API functionality has changed in the following ways to bring the MPLAB Harmony implementation to be more in line with industry standard Berkeley Socket API specification.

- *errno* – APIs now report specific error codes through a global variable: errno. This variable is only valid between calls to the Berkeley API, and is currently not tread safe, there is on errno variable shared across all threads.
- *bind* – bind now supports binding to a specific IP address. If an invalid address is specified the function will now return an error and set errno accordingly
- *accept* – accept no longer returns INVALID_SOCKET on error but SOCKET_ERROR and errno is set to the specific error.
- *connect* - connect no longer returns INVALID_SOCKET on error but SOCKET_ERROR and errno is set to the specific error. In addition, connect does not return SOCKET_CNXN_IN_PROGRESS; instead, SOCKET_ERROR is returned and errno is set to EINPROGRESS. Finally,

connect now supports setting a specific local IP address; if the IP address is invalid, the SOCKET_ERROR is returned and errno is set with the error.

- *sendto* – sendto now supports sending from a specific IP address. If an invalid address is specified, the function returns SOCKET_ERROR and sets errno.

HTTP Changes

This topic discusses the changes made to the `http.h` file.

Note: This file was named `http2.h` in the V6 Beta TCP/IP Stack.

Description

HTTP Changes

The TCP/IP Stack include header file, `http.h`, has been updated as follows.

Function Name Changes

Prior Function Name	New Function Name
HTTPExecutePost	TCPIP_HTTP_PostExecute
HTTPExecuteGet	TCPIP_HTTP_GetExecute
HTTPNeedsAuth	TCPIP_HTTP_FileAuthenticate
HTTPCheckAuth	TCPIP_HTTP_UserAuthenticate
HTTPReadPostPair	TCPIP_HTTP_PostReadPair
HTTPPrint_varname(...)	TCPIP_HTTP_Print_varname(…)

All of the functions that the HTTP exposes as its API now take a first parameter, a HTTP connection handle, named HTTP_CONN_HANDLE. This allows a cleaner and better behavior in both multi-threaded environments and in the situation where we it may be desired to run multiple instances of the HTTP process itself, allowing for serving multiple independent connections.

For example, the HTTP functions now appear as:

- TCPIP_HTTP_FileInclude(HTTP_CONN_HANDLE connHandle, const uint8_t* cFile) ;
- TCPIP_HTTP_GetExecute(HTTP_CONN_HANDLE connHandle);
- TCPIP_HTTP_PostExecute(HTTP_CONN_HANDLE connHandle);
- TCPIP_HTTP_CurrentConnectionFileGet(HTTP_CONN_HANDLE connHandle);
- TCPIP_HTTP_CurrentConnectionCallbackPosGet(HTTP_CONN_HANDLE connHandle);

Support has been added for the HTTP module client to store/retrieve its own connection related data. For example:

- TCPIP_HTTP_CurrentConnectionUserDataSet(HTTP_CONN_HANDLE connHandle, const void* uData);
- TCPIP_HTTP_CurrentConnectionUserDataGet(HTTP_CONN_HANDLE connHandle);

HTTP Print

Also, the functions exposed in `http_print.h` take the connection handle parameter. For example:

- TCPIP_HTTP_Print(HTTP_CONN_HANDLE connHandle, uint32_t callbackID);
- TCPIP_HTTP_Print_hellomsg(HTTP_CONN_HANDLE connHandle);

These changes affect all of the function calls in the file, `custom_http_app.c`, such as:

- TCPIP_HTTP_GetExecute(HTTP_CONN_HANDLE connHandle);
- TCPIP_HTTP_PostExecute(HTTP_CONN_HANDLE connHandle);

To port an existing application, the extra parameter will have to be added. The connection handle is passed to the application as part of the HTTP callback. The modifications should be minimal and possible using an editor without any other impact.

Please note that the MPFS2 generator utility (`mpfs2.jar`) has been updated to support the new HTTP API.

See the list of the complete API changes in `http.h` and `http_print.h`.

MPLAB Harmony TCP/IP Stack Storage Changes

This topic describes the storage changes in the MPLAB Harmony TCP/IP Stack.

Description

MPLAB Harmony TCP/IP Stack Storage Changes

The V5 TCP/IP Stack include header file, `tcpip_storage.h`, *does not* exist in the MPLAB Harmony TCP/IP Stack.

The TCP/IP Stack storage has become obsolete and is no longer maintained. This is mainly due to the way the dynamic initialization of the

stack is done in the V5 TCP/IP Stack:

- Each module has its own initialization data
- There are many parameters that could be relevant for an application and that may require storage besides the IP address, IP Mask, or SSID, etc.
- The data is passed dynamically at the stack initialization. There is no longer a "build time" set of parameters against which to check at stack configuration (although support for a default configuration set at build time is present). The actual data contents are the application's responsibility.
- A system database service will be added that will use the File System. The database service will maintain configurations for all the modules in the system, including the TCP/IP. The system database service API will be available to the applications as well for storing/retrieving proprietary information.
- The parameter, TCPIP_STACK_USE_STORAGE, in the file, `tcpip_config.h`, should not be enabled. Future plans call for this service to be removed from the distribution.

MPLAB Harmony TCP/IP Stack Configuration Changes

This topic describes the configuration changes in the MPLAB Harmony TCP/IP Stack.

Description

The stack configuration has a completely *new* structure. Each project has its own configuration folder that stores multiple configurations based on CPU platforms and hardware boards.

Note: V5 TCP/IP Stack configuration files *cannot* be reused in the MPLAB Harmony TCP/IP Stack.

The following is a list of the most important features:

- `system_config` folder for projects – all profiles are stored in this location
- Multiple profiles per project, which are separated by CPU platform. The TCP/IP profile is usually common.
- `tcpip_config.h` – selects modules that are part of the build
- Each module has its own profile: `arp_config.h`, `tcp_config.h`, `ssl_config.h`, etc.
- BSP profiles per development board containing:
- `hardware_config.h` – the BSP specific hardware settings
- `media_storage_config.h` – storage media and partitioning settings
- `sys_fs_config.h` – file system configuration and settings
- `mpfs_config.h` – MPFS file system settings
- `network_config.h` – configuration for every network interface: NetBIOS names, default IP addresses, etc.
- `system_config.h` – system configuration (i.e, system tick, system console selection, system debug services, etc.)

MPLAB Harmony TCP/IP Stack Configuration

This topic provides information on the configuration of the MPLAB Harmony TCP/IP Stack.

Description

MPLAB Harmony TCP/IP Stack Configuration

The presented structure is simply a model and a different layout can be chosen. The only requirement is to use the proper path in your project so that the necessary configuration files are found at build time.

For example, depending on your exact hardware platform or development board, a path similar to the following can be added to your project path:

```
..\harmony\<version>\apps\tcpip\tcpip_web_server_demo_app\firmware\src\system_config\pic32_eth_sk_int_dyn\tcpip_profile
```

MPLAB Harmony TCP/IP Stack Heap Configuration

This topic provides information on the heap configuration of the MPLAB Harmony TCP/IP Stack.

Description

MPLAB Harmony TCP/IP Stack Heap Configuration

The MPLAB Harmony TCP/IP Stack uses dynamic memory for both initialization and run time buffers. Therefore, there are two requirements for a project containing the stack to work properly.

The first requirement is that the TCP/IP stack should be configured with enough heap space. The amount of heap used by the stack is specified by the TCPIP_STACK_DRAM_SIZE parameter in the `tcpip_config.h` file.

The value of the required TCP/IP heap for a project is application dependent. Some of the most important factors that impact the heap size are:

- Number of TCP sockets
- By default, each TCP socket requires 512 bytes for a RX buffer and 512 bytes for a TX buffer. These parameters can be adjusted by modifying the values specified in `tcp_config.h` or dynamically through the TCPIP_TCP_OptionsSet function.
- Number of UDP sockets:
- For each UDP socket that needs to initiate a transmission, the IP layer will have to allocate the required space (suggested by the functions, UDPv4IsTxPutReady or UDPv4IsTxPutReady
- Once the UDP buffering will be added, each socket will have its own RX and TX buffers. These parameters can be adjusted by modifying the values specified in the file, `udp_config.h`, or dynamically through the TCPIP_UDP_OptionsSet function
- The type of Ethernet MAC that is used:
- The PIC32MX6XX/7XX devices with a built-in 100 Mbps Ethernet controller use system memory for buffering incoming RX traffic. For sustained 100 Mbps operation, adequate RX buffer space must be provided. This parameter can be adjusted by modifying the values specified in the file, `network_config.h`, using the array, TCPIP_HOSTS_CONFIGURATION.
- If SSL is enabled, it will require additional buffering for encryption/decryption at run-time. The SSL module buffering requirement is dependent on the RSA key length specified in `ssl_config.h`.

The second requirement is that the project that includes the MPLAB Harmony TCP/IP Stack must be built with sufficient heap size to accommodate the stack (at a minimum, the project needs the TCPIP_STACK_DRAM_SIZE bytes of heap). This parameter is adjusted on the Linker tab in the project properties.

In general, for a TCP/IP project running on a PIC32MX device with an embedded Ethernet controller, at least 8 KB of heap space is needed by the stack. However, the following is implied:

- 100 Mbps traffic is not sustained
- No SSL
- Relatively few sockets

A typical value for comfortably handling 100 Mbps Ethernet traffic would be 40 KB of TCP/IP heap space.

The amount of the required heap is less for an external Ethernet MAC.

Keep in mind the following when assigning heap space:

- If there is not enough heap space the stack initialization may fail
- If there is not enough heap space some run time buffer allocation may fail and some packets transmission will have to be deferred until a later time, thus impacting the stack performance
- It is always a good idea to have a reserve, at least an extra 2 KB of heap space above the total amount of space that is used
- A very useful tool in understanding the heap allocation and how the space is distributed among the stack modules is the TCP/IP command processor

By enabling the parameter, TCPIP_STACK_DRAM_DEBUG_ENABLE, in the `tcpip_config.h` file, the stack will output debug messages when it runs out of memory. Then, using the `heapinfo` command at the TCP/IP command processor prompt will return a snapshot of the current TCP/IP heap status and can help in early detection of problems.

Optionally, enabling the parameter, TCPIP_STACK_DRAM_TRACE_ENABLE in the `tcpip_config.h` file, will instruct the TCP/IP heap allocation module to store trace information that will be displayed with the `heapinfo` command.

New MPLAB Harmony TCP/IP Stack API Functions

This topic describes some of the important new API functions in the MPLAB Harmony TCP/IP Stack.

Description

There are many new functions available that have been introduced to take advantage of new features, such as IPv6 and support of multiple interfaces. However, there is no concern for the porting process regarding the new API calls as the previous stack implementation did not support this kind of service.

A new API function should be added to the application only when the access to the new feature is needed.

Existent applications should port easily without using the new API.

For reference, the following are a few of the most important new API functions that could prove useful in the porting of your application:

- Initialization of the stack network interfaces:
- TCPIP_STACK_NetUp()
- TCPIP_STACK_NetDown()
- Default interface selection:
- TCPIP_STACK_NetDefaultGet()
- TCPIP_STACK_NetDefaultSet()
- TCP/UDP multiple interface support:
- TCPIP_TCP_Bind()
- TCPIP_TCP_RemoteBind()
- TCPIP_TCP_SocketNetGet()
- TCPIP_TCP_SocketNetSet()
- TCPIP_UDPBind()
- TCPIP_UDP_RemoteBind()

- TCPIP_UDP_SocketNetGet()
- TCPIP_UDP_SocketNetSet()

Refer to the file, `tcpip_manager.h`, for a complete list of all new network interface APIs.

Main Program Changes

This topic describes the main program changes in the MPLAB Harmony TCP/IP Stack.

Description

Main Program Changes

There are few changes that the main program loop has to take care of. A full working example is given with the file, `main_demo.c`, which is provided with the stack distribution.

The main program should call the following functions:

- SYS_Initialize() – This function is what initializes the system and runs the Board Support Package (BSP) specific code
- SYS_Tasks() – This is the function that takes care of the system tasks and must be called periodically
- TCPIP_STACK_Task() – This is the TCP/IP stack tasks function that runs all of the state machines that are part of the TCP/IP stack. This function must be called periodically. Please note the name change.

To gain access to a network interface the TCPIP_STACK_NetHandle("iname") function should be used. Examples are provided in the previous section.

MPLAB Harmony TCP/IP Stack Initialization Changes

This topic describes the initialization updates to the MPLAB Harmony TCP/IP Stack.

Description

The TCP/IP Stack include header file, `tcpip_manager.h`, has been updated as follows.

To initialize the stack the call is now:

```
TCPIP_STACK_Initialize(const TCPIP_NETWORK_CONFIG* pNetConf, int nNets, TCPIP_STACK_MODULE_CONFIG*
pModConfig, int nModules);
```

Where:

- `pNetConf` – is a pointer to an array of configurations for all the initialized network interfaces
- `nNets` – is the number of the network interfaces that the stack is to support
- `pModConfig` – is a pointer to an table storing configuration data for the TCP/IP stack modules
- `nModules` – is the number of entries in this table, (i.e., the number of initialized modules)

The default TCP/IP stack configuration is provided with the stack distribution and consists of:

- `network_config.h`: TCPIP_HOSTS_CONFIGURATION[] – an array of TCPIP_NETWORK_CONFIG entries describing the default configuration of each network interface
- `tcpip_modules_config.h`: TCPIP_STACK_MODULE_CONFIG_TBL[] – an array of TCPIP_STACK_MODULE_CONFIG entries storing the default configuration data for each module of the TCP/IP stack. This is just an aggregate of all the default configurations per module found in the module configuration header files, such as `tcp_config.h`, `udp_config.h`, `arp_config.h`, and so on.

These tables are defined by default in the configuration files. See MPLAB Harmony TCP/IP Stack Configuration Changes for details.

You can change the parameters of these structures as appropriate.

An example of the TCP/IP initialization is part of the distributed file, `main_demo.c`. This file is located in the following Windows® path:

`.\harmony<version>\apps\tcpip\tcpip_web_server_demo_app\firmware\src`

Note: The stack initialization may fail. The application code must check the result of this call to detect if the stack failed to initialize properly.

MPLAB Harmony TCP/IP Stack Access Changes

This topic describes the stack access updates that were made to the MPLAB Harmony TCP/IP Stack.

Description

The TCP/IP Stack include header file, `tcpip_manager.h`, has been updated as follows.

Direct access to the internally maintained stack structures has been removed. To access the information for a specific network interface a handle must be obtained using the interface name. For example:

```
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("PIC32INT");
```

or

```
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("MRF24W");
```

Once a handle to the interface has been obtained, different parameters of that network interface can be queried and set. For example:

- TCPIP_STACK_NetAddress(TCPIP_NET_HANDLE netH) – returns the network interface address
- TCPIP_STACK_NetAddressGateway(TCPIP_NET_HANDLE netH) – returns the interface gateway address

Refer to the `tcpip_manager.h` file for a complete list of functions.

Note: The TCPIP_NET_HANDLE is an opaque data type.

The well known names of the network interfaces are currently in the file, `tcpip.h`.

The exact header file that exposes the network interfaces names may change in the future; however, the names of the supported network interfaces will be retained.

Use the file, `network_config.h`, as an example.

Refer to the file, `main_demo.c`, for an example of how to use the interface handle functions.

MPLAB Harmony TCP/IP Stack Utilities

This topic describes the MPLAB Harmony TPC/IP Stack utilities.

Description

The MPLAB Harmony TCP/IP Stack includes the following utilities:

- MPFS2 - The MPFS2 Java utility that assists the application in generating the MPFS image of the files needed by the Web server has been updated. This utility supports both the V5 and MPLAB Harmony versions of the TCP/IP stack with improved functionality. The V5 TCP/IP Stack utility can still be used; however, some minor adjustments may need to be done manually. Refer to the MPFS2 Utility section for more information.
- TCP/IP Discoverer - The TCP/IP Discoverer Java utility has been updated to support not only the MPLAB Harmony TCP/IP Stack, but also IPV6. This utility is up and running and is backward compatible with the V5 TCP/IP Stack. Refer to the TCP/IP Discoverer Utility section for more information.
- TCP/IP Configuration - The TCP/IP Configuration utility is currently not supported. The need of individually configuring TCP and UDP sockets is no longer needed, as all sockets behave similarly and they are allocated dynamically as needed.

MPLAB Harmony TCP/IP Stack SSL/RSA Usage

This topic provides information regarding the usage of SSL and RSA.

Description

The SSL design has been updated to support multiple interfaces. The same is true for the RSA engine. However, the SSL module does not currently use the Microchip Cryptographic (Crypto) Library. Until the proper Crypto Library support is added to SSL, the stack is distributed with the RSA and RC4 code embedded into the stack.

In the future, the Crypto Library will be completely removed from the TCP/IP stack, and the stack will be just a client to the Crypto Library.

Porting Applications from the V6 Beta TCP/IP Stack to the MPLAB Harmony TCP/IP Stack

Porting an application from the V6 Beta TCP/IP Stack to the TCP/IP Stack within MPLAB Harmony consists of updating existing TCP/IP Stack function names in your application to the new MPLAB Harmony function names.

MPLAB Harmony TCP/IP Stack Function Name Compliance

This topic provides header file function mapping tables that show the V6 Beta TCP/IP Stack function name and the compliant name within the MPLAB Harmony TCP/IP Stack.

Description

The replacement names chosen to make existing TCP/IP Stack functions compliant with MPLAB Harmony are shown in individual tables per header file name.

The files listed in the tables are located in the following two MPLAB Harmony installation folders:

`.\harmony\<version>\framework\tcpip`

`.\harmony\<version>\framework\tcpip\src`

Each table shows the original and new names used. Note that other than name changes, no other modifications were made to the middleware functions.

Header File: arp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
ARP_ENTRY_QUERY	TCPIP_ARP_ENTRY_QUERY
ARP_ENTRY_TYPE	TCPIP_ARP_ENTRY_TYPE
ARP_EVENT_HANDLER	TCPIP_ARP_EVENT_HANDLER
ARP_EVENT_TYPE	TCPIP_ARP_EVENT_TYPE
ARP_HANDLE	TCPIP_ARP_HANDLE
ARP_OPERATION_TYPE	TCPIP_ARP_OPERATION_TYPE
ARP_RESULT	TCPIP_ARP_RESULT
ARPCacheGetEntriesNo	TCPIP_ARP_CacheEntriesNoGet
ARPCacheSetThreshold	TCPIP_ARP_CacheThresholdSet
ARPEntryGet	TCPIP_ARP_EntryGet
ARPEntryQuery	TCPIP_ARP_EntryQuery
ARPEntryRemove	TCPIP_ARP_EntryRemove
ARPEntryRemoveAll	TCPIP_ARP_EntryRemoveAll
ARPEntryRemoveNet	TCPIP_ARP_EntryRemoveNet
ARPEntrySet	TCPIP_ARP_EntrySet
ARPDeRegisterHandler	TCPIP_ARP_HandlerDeRegister
ARPIsResolved	TCPIP_ARP_IsResolved
ARPProbe	TCPIP_ARP_Probe
ARPRegisterHandler	TCPIP_ARP_HandlerRegister
ARPResolve	TCPIP_ARP_Resolve
ARPRegisterCallbacks	TCPIP_ARP_CallbacksRegister

Header File: dhcp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
DHCPDeRegisterHandler	TCPIP_DHCP_HandlerDeRegister
DHCPRegisterHandler	TCPIP_DHCP_HandlerRegister
DHCPSetRequestTimeout	TCPIP_DHCP_RequestTimeoutSet
DHCP_EVENT_HANDLER	TCPIP_DHCP_EVENT_HANDLER
DHCP_EVENT_TYPE	TCPIP_DHCP_EVENT_TYPE
DHCP_HANDLE	TCPIP_DHCP_HANDLE
DHCPDisable	TCPIP_DHCP_Disable
DHCPEnable	TCPIP_DHCP_Enable
DHCPIsBound	TCPIP_DHCP_IsBound
DHCPIsEnabled	TCPIP_DHCP_IsEnabled
DHCPIsServerDetected	TCPIP_DHCP_IsServerDetected

Header File: dhcps.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
DHCPServerIsEnabled	TCPIP_DHCPS_IsEnabled
DHCPServerLeaseEntryGet	TCPIP_DHCPS_LeaseEntryGet
DCHPServerGetPoolEntries	TCPIP_DHCPS_GetPoolEntries
DCHPServerRemovePoolEntries	TCPIP_DHCPS_RemovePoolEntries
DHCPServerDisable	TCPIP_DHCPS_Disable
DHCPServerEnable	TCPIP_DHCPS_Enable

DHCPServerIsEnabled	TCPIP_DHCPS_IsEnabled

Header File: ddns.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
DDNSForceUpdate	TCPIP_DDNS_UpdateForce
DDNSSetService	TCPIP_DDNS_ServiceSet
DDNSGetLastIP	TCPIP_DDNS_LastIPGet
DDNSGetLastStatus	TCPIP_DDNS_LastStatusGet

Header File: dns.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
DNSBeginUsage	TCPIP_DNS_UsageBegin
DNSEndUsage	TCPIP_DNS_UsageEnd
DNSResolve	TCPIP_DNS_Resolve
DNSIsResolved	TCPIP_DNS_IsResolved

Header File: http.h (formerly http2.h)

V6 Beta TCP/IP Stack Name (http2.h)	MPLAB Harmony Compliant Name (http.h)
HTTPURLDecode	TCPIP_HTTP_URLDecode
HTTPGetArg	TCPIP_HTTP_ArgGet
HTTPReadPostName	TCPIP_HTTP_PostNameRead
HTTPReadPostValue	TCPIP_HTTP_PostValueRead
HTTPIncFile	TCPIP_HTTP_FileInclude
HTTPCurConnectionFileGet	TCPIP_HTTP_CurrentConnectionFileGet
HTTPCurConnectionPostSmGet	TCPIP_HTTP_CurrentConnectionPostSmGet
HTTPCurConnectionPostSmSet	TCPIP_HTTP_CurrentConnectionPostSmSet
HTTPCurConnectionDataBufferGet	TCPIP_HTTP_CurrentConnectionDataBufferGet
HTTPCurConnectionCallbackPosGet	TCPIP_HTTP_CurrentConnectionCallbackPosGet
HTTPCurConnectionCallbackPosSet	TCPIP_HTTP_CurrentConnectionCallbackPosSet
HTTPCurConnectionStatusSet	TCPIP_HTTP_CurrentConnectionStatusSet
HTTPCurConnectionHasArgsSet	TCPIP_HTTP_CurrentConnectionHasArgsSet
HTTPCurConnectionByteCountGet	TCPIP_HTTP_CurrentConnectionByteCountGet
HTTPCurConnectionByteCountSet	TCPIP_HTTP_CurrentConnectionByteCountSet
HTTPCurConnectionByteCountDec	TCPIP_HTTP_CurrentConnectionByteCountDec
HTTPCurConnectionSocketGet	TCPIP_HTTP_CurrentConnectionSocketGet

Header File: icmp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
ICMPDeRegisterCallback	TCPIP_ICMP_CallbackDeregister
ICMPRegisterCallback	TCPIP_ICMP_CallbackRegister
ICMPSendEchoRequest	TCPIP_ICMP_EchoRequestSend

Header File: icmpv6.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_ICMPV6_DeRegisterCallback	TCPIP_ICMPV6_CallbackDeregister
TCPIP_ICMPV6_PutHeaderEchoRequest	TCPIP_ICMPV6_HeaderEchoRequestPut

TCPIP_ICMPV6_RegisterCallback	TCPIP_ICMPV6_CallbackRegister
TCPIP_IPV6_GetDestAddress	TCPIP_IPV6_AddressDestGet
TCPIP_IPV6_GetSourceAddress	TCPIP_IPV6_AddressSourceGet
TCPIP_IPV6_SetDestAddress	TCPIP_IPV6_AddressDestSet
TCPIP_IPV6_SetSourceAddress	TCPIP_IPV6_AddressSourceSet
TCPIP_IPV6_DAS_SelectSourceAddress	TCPIP_IPV6_DAS_AddressSourceSelect
TCPIP_IPV6_AddUnicastAddress	TCPIP_IPV6_AddressUnicastAdd
TCPIP_IPV6_RemoveUnicastAddress	TCPIP_IPV6_AddressUnicastRemove
TCPIP_IPV6_AddMulticastListener	TCPIP_IPV6_MulticastListenerAdd
TCPIP_IPV6_RemoveMulticastListener	TCPIP_IPV6_MulticastListenerRemove
TCPIP_IPV6_RegisterHandler	TCPIP_IPV6_HandlerRegister
TCPIP_IPV6_DeRegisterHandler	TCPIP_IPV6_HandlerDeRegister

Header File: ipv6.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_IPV6_InterfaceIsReady	Not Applicable
TCPIP_IPV6_Put	Not Applicable
TCPIP_IPV6_AddressIsSolicitedNodeMulticast	Not Applicable
TCPIP_IPV6_Flush	Not Applicable
TCPIP_IPV6_PutArrayHelper	TCPIP_IPV6_ArrayPutHelper
TCPIP_IPV6_DAS_SelectSourceAddress	TCPIP_IPV6_DASSourceAddressSelect
TCPIP_IPV6_GetDestAddress	TCPIP_IPV6_DestAddressGet
TCPIP_IPV6_SetDestAddress	TCPIP_IPV6_DestAddressSet
TCPIP_IPV6_DeRegisterHandler	TCPIP_IPV6_HandlerDeregister
TCPIP_IPV6_RegisterHandler	TCPIP_IPV6_HandlerRegister
TCPIP_IPV6_AddMulticastListener	TCPIP_IPV6_MulticastListenerAdd
IPv6RemoveMulticastListener	TCPIP_IPV6_MulticastListenerRemove
TCPIP_IPV6_FreePacket	TCPIP_IPV6_PacketFree
TCPIP_IPV6_SetPayload	TCPIP_IPV6_PayloadSet
TCPIP_IPV6_GetSourceAddress	TCPIP_IPV6_SourceAddressGet
TCPIP_IPV6_SetSourceAddress	TCPIP_IPV6_SourceAddressSet
TCPIP_IPV6_IsTxPutReady	TCPIP_IPV6_TxIsPutReady
TCPIP_IPV6_AllocateTxPacket	TCPIP_IPV6_TxPacketAllocate
TCPIP_IPV6_AddUnicastAddress	TCPIP_IPV6_UnicastAddressAdd
TCPIP_IPV6_GetArray	TCPIP_IPV6_ArrayGet
IPv6RemoveUnicastAddress	TCPIP_IPV6_UnicastAddressRemove

Header File: ndp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_NDP_NeighborConfirmReachability	TCPIP_NDP_NborReachConfirm

Header File: smtp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
SMTPPutArray	TCPIP_SMTP_ArrayPut
SMTPFlush	TCPIP_SMTP_Flush
SMTPIsBusy	TCPIP_SMTP_IsBusy

SMTPIsPutReady	TCPIP_SMTP_IsPutReady
SMTPSendMail	TCPIP_SMTP_MailSend
SMTPPut	TCPIP_SMTP_Put
SMTPPutDone	TCPIP_SMTP_PutIsDone
SMTPPutString	TCPIP_SMTP_StringPut
SMTPBeginUsage	TCPIP_SMTP_UsageBegin
SMTPEndUsage	TCPIP_SMTP_UsageEnd

Header File: snmp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
SNMPNotify	TCPIP_SNMP_Notify
SNMPIsNotifyReady	TCPIP_SNMP_NotifyIsReady
SNMPNotifyPrepare	TCPIP_SNMP_NotifyPrepare
SNMPGetPktProcessingDynMemStubPtrs	TCPIP_SNMP_PacketProcStubPtrsGet
SNMPGetProcessBuffData	TCPIP_SNMP_ProcessBufferDataGet
SNMPPutDataToProcessBuff	TCPIP_SNMP_DataCopyToProcessBuffer
SNMPRetrieveReadCommunity	TCPIP_SNMP_ReadCommunityGet
SNMPGetTrapTime	TCPIP_SNMP_TrapTimeGet
SNMPUdpClientGetNet	TCPIP_SNMP_ClientGetNet
SNMPRetrieveWriteCommunity	TCPIP_SNMP_WriteCommunityGet

Header File: sntp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
SNTPGetUTCSeconds	TCPIP_SNTP_UTCSecondsGet

Header File: tcpip_manager.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_STACK_SetNetAddress	TCPIP_STACK_NetAddressSet
TCPIP_STACK_SetNetGatewayAddress	TCPIP_STACK_NetAddressGatewaySet
TCPIP_STACK_SetNetPriDNSAddress	TCPIP_STACK_NetAddressDnsPrimarySet
TCPIP_STACK_SetNetSecondDNSAddress	TCPIP_STACK_NetAddressDnsSecondSet
TCPIP_STACK_SetNetMask	TCPIP_STACK_NetMaskSet
TCPIP_STACK_SetNetMacAddress	TCPIP_STACK_NetAddressMacSet
TCPIP_STACK_NetGatewayAddress	TCPIP_STACK_NetAddressGateway
TCPIP_STACK_NetPriDNSAddress	TCPIP_STACK_NetAddressDnsPrimary
TCPIP_STACK_NetSecondDNSAddress	TCPIP_STACK_NetAddressDnsSecond
TCPIP_STACK_NetMacAddress	TCPIP_STACK_NetAddressMac
TCPIP_STACK_NetBcastAddress	TCPIP_STACK_NetAddressBcast
TCPIP_STACK_SetNetBIOSName	TCPIP_STACK_NetBiosNameSet
TCPIP_STACK_GetDefaultNet	TCPIP_STACK_NetDefaultGet
TCPIP_STACK_SetDefaultNet	TCPIP_STACK_NetDefaultSet
TCPIP_STACK_IsNetLinked	TCPIP_STACK_NetIsLinked
TCPIP_STACK_CreateNetInfo	TCPIP_STACK_NetInfoCreate
TCPIP_STACK_IsNetUp	TCPIP_STACK_NetIsUp

Header File: tcp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIsGetReady	TCPIP_TCP_GetIsReady
TCPGetArray	TCPIP_TCP_ArrayGet
TCPPeek	TCPIP_TCP_Peek
TCPPeekArray	TCPIP_TCP_ArrayPeek
TCPGetRxFIFOFree	TCPIP_TCP_FifoRxFreeGet
TCPGetTxFIFOFull	TCPIP_TCP_FifoTxFullGet
TCPDiscard	TCPIP_TCP_Discard
TCPGet	TCPIP_TCP_Get
TCPFind	TCPIP_TCP_Find
TCPFindArray	TCPIP_TCP_ArrayFind
TCPAdjustFIFOSize	TCPIP_TCP_FifoSizeAdjust
TCPOpenServer	TCPIP_TCP_ServerOpen
TCPOpenClient	TCPIP_TCP_ClientOpen
TCPBind	TCPIP_TCP_Bind
TCPRemoteBind	TCPIP_TCP_RemoteBind
TCPSetOptions	TCPIP_TCP_OptionsSet
TCPGetOptions	TCPIP_TCP_OptionsGet
TCPIsConnected	TCPIP_TCP_IsConnected
TCPWasReset	TCPIP_TCP_WasReset
TCPDisconnect	TCPIP_TCP_Disconnect
TCPClose	TCPIP_TCP_Close
TCPGetSocketInfo	TCPIP_TCP_SocketInfoGet
TCPSocketSetNet	TCPIP_TCP_SocketNetSet
TCPSocketGetNet	TCPIP_TCP_SocketNetGet
TCPIsPutReady	TCPIP_TCP_PutIsReady
TCPPut	TCPIP_TCP_Put
TCPPutArray	TCPIP_TCP_ArrayPut
TCPPutString	TCPIP_TCP_StringPut
TCPFlush	TCPIP_TCP_Flush
TCPGetRxFIFOFull	TCPIP_TCP_FifoRxFullGet
TCPGetTxFIFOFree	TCPIP_TCP_FifoTxFreeGet
TCPAbort	TCPIP_TCP_Abort
TCPAddSSLListener	TCPIP_TCPSSL_ListenerAdd
TCPIsSSL	TCPIP_TCP_SocketIsSecuredBySSL
TCPSetDestinationIPAddress	TCPIP_TCP_DestinationIPAddressSet
TCPSetSourceIPAddress	TCPIP_TCP_SourceIPAddressSet
TCPSSLIsHandshaking	TCPIP_TCPSSL_StillHandshaking
TCpSSLMessageTransmit	TCPIP_TCPSSL_MessageTransmit
TCPSSLPutRecordHeader	TCPIP_TCPSSL_RecordHeaderPut
TCPStartSSLClient	TCPIP_TCPSSL_ClientStart
TCPStartSSLClientEx	TCPIP_TCPSSL_ClientBegin
TCPStartSSLServer	TCPIP_TCPSSL_ServerStart

Header File: tcpip_events.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_STACK_GetPendingEvents	TCPIP_STACK_EventsPendingGet
TCPIP_STACK_RegisterHandler	TCPIP_STACK_HandlerRegister
TCPIP_STACK_DeRegisterHandler	TCPIP_STACK_HandlerDeregister

Header File: ipv4.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
TCPIP_IPV4_GetArray	TCPIP_IPV4_ArrayGet
TCPIP_IPV4_PutArray	TCPIP_IPV4_ArrayPut
TCPIP_IPV4_PutArrayHelper	TCPIP_IPV4_ArrayPutHelper
TCPIP_IPV4_GetDestAddress	TCPIP_IPV4_DestAddressGet
TCPIP_IPV4_SetDestAddress	TCPIP_IPV4_DestAddressSet
TCPIP_IPV4_FreePacket	TCPIP_IPV4_PacketFree
TCPIP_IPV4_SetPayload	TCPIP_IPV4_PayloadSet
TCPIP_IPV4_GetSourceAddress	TCPIP_IPV4_SourceAddressGet
TCPIP_IPV4_SetSourceAddress	TCPIP_IPV4_SourceAddressSet
TCPIP_IPV4_IsTxPutReady	TCPIP_IPV4_TxIsPutReady
TCPIP_IPV4_IsTxReady	TCPIP_IPV4_TxIsReady
TCPIP_IPV4_AllocateTxPacket	TCPIP_IPV4_TxPacketAllocate

Header File: udp.h

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
UDPIsGetReady	TCPIP_UDP_GetIsReady
UDPSetRxOffset	TCPIP_UDP_RxOffsetSet
UDPGetArray	TCPIP_UDP_ArrayGet
UDPDiscard	TCPIP_UDP_Discard
UDPGet	TCPIP_UDP_Get
UDPOpenServer	TCPIP_UDP_ServerOpen
UDPOpenClient	TCPIP_UDP_ClientOpen
UDPBind	TCPIP_UDP_Bind
UDPRemoteBind	TCPIP_UDP_RemoteBind
UDPSetOptions	TCPIP_UDP_OptionsSet
UDPGetOptions	TCPIP_UDP_OptionsGet
UDPIsConnected	TCPIP_UDP_IsConnected
UDPIsOpened	TCPIP_UDP_IsOpened
UDPClose	TCPIP_UDP_Close
UDPGetSocketInfo	TCPIP_UDP_SocketInfoGet
UDPSocketSetNet	TCPIP_UDP_SocketNetSet
UDPSocketGetNet	TCPIP_UDP_SocketNetGet
UDPSetTxOffset	TCPIP_UDP_TxOffsetSet
UDPIsTxPutReady	TCPIP_UDP_TxPutIsReady
UDPPutArray	TCPIP_UDP_ArrayPut
UDPPutString	TCPIP_UDP_StringPut
UDPGetTxCount	TCPIP_UDP_TxCountGet
UDPFlush	TCPIP_UDP_Flush

UDPPut	TCPIP_UDP_Put
UDPSetBcastIPV4Address	TCPIP_UDP_BcastIPV4AddressSet
UDPSetSourceIPAddress	TCPIP_UDP_SourceIPAddressSet
UDPSetDestinationIPAddress	TCPIP_UDP_DestinationIPAddressSet

Development Information (Advance Information)

This *advance information* is specifically provided to assist MPLAB Harmony driver development for porting the existing Ethernet Controller Library and MAC/PHY drivers to the new platform.

Porting the MPLAB Harmony Drivers and Peripheral Library

This topic provides function mapping tables that show the V6 Beta TCP/IP Stack function name and the compliant name within the MPLAB Harmony TCP/IP Stack for the purpose of porting to the MPLAB Harmony Ethernet Drivers and Ethernet Peripheral Library.

Description

Note: This content in this section is considered to be *Advance Information*.

Porting the MPLAB Harmony Ethernet Drivers and the MPLAB Harmony Ethernet Peripheral Library involves replacing the Ethernet Controller Library with calls to MPLAB Harmony equivalents. This information is listed in Table 1. Please note that function arguments were modified in some instances.

To understand how a function of the Ethernet Controller Library is different from its equivalent MPLAB Harmony function, the header of the MPLAB Harmony function should be examined. The MPLAB Harmony function header always lists the Ethernet Controller Library function(s) that the new MPLAB Harmony function replaces, including the arguments of the old function.

Table 1: TCP/IP Stack Function Mapping

Ethernet Controller Peripheral Library Name	MPLAB Harmony Name
EthDescriptorGetBuffer	DRV_ETHMAC_LegacyDescriptorGetBuffer
EthDescriptorsPoolAdd	DRV_ETHMAC_LegacyDescriptorsPoolAdd
EthDescriptorsPoolCleanUp	DRV_ETHMAC_LegacyDescriptorsPoolCleanUp
EthDescriptorsPoolRemove	DRV_ETHMAC_LegacyDescriptorsPoolRemove
EthEventsClr	PLIB_ETH_EventsClr
EthEventsEnableClr	PLIB_ETH_EventsEnableClr
EthEventsEnableGet	PLIB_ETH_EventsEnableGet
EthEventsEnableSet	PLIB_ETH_EventsEnableSet
EthEventsEnableWrite	PLIB_ETH_EventsEnableWrite
EthEventsGet	PLIB_ETH_EventsGet
EthInit	DRV_ETHMAC_LegacyInit
EthMACGetAddress	PLIB_ETH_MACGetAddress
EthMACOpen	DRV_ETHMAC_LegacyMACOpen
EthMACSetAddress	PLIB_ETH_MACSetAddress
EthMACSetMaxFrame	PLIB_ETH_MACSetMaxFrame
EthRxAcknowledgeBuffer	DRV_ETHMAC_LegacyRxAcknowledgeBuffer
EthRxAcknowledgePacket	DRV_ETHMAC_LegacyRxAcknowledgePacket
EthRxBuffersAppend	DRV_ETHMAC_LegacyRxBuffersAppend
EthRxFiltersClr	PLIB_ETH_RxFiltersClr
EthRxFiltersHTSet	PLIB_ETH_RxFiltersHTSet
EthRxFiltersPMClr	PLIB_ETH_RxFiltersPMClr
EthRxFiltersPMSet	PLIB_ETH_RxFiltersPMSet
EthRxFiltersSet	PLIB_ETH_RxFiltersSet
EthRxFiltersWrite	PLIB_ETH_RxFiltersWrite
EthRxGetBuffer	DRV_ETHMAC_LegacyRxGetBuffer
EthRxGetPacket	DRV_ETHMAC_LegacyRxGetPacket
EthRxSetBufferSize	PLIB_ETH_RxSetBufferSize
EthStatRxAlgnErrCnt	PLIB_ETH_StatRxAlgnErrCnt
EthStatRxFcsErrCnt	PLIB_ETH_StatRxFcsErrCnt
EthStatRxOkCnt	PLIB_ETH_StatRxOkCnt
EthStatRxOvflCnt	PLIB_ETH_StatRxOvflCnt
EthStatTxMColCnt	PLIB_ETH_StatTxMColCnt
EthStatTxOkCnt	PLIB_ETH_StatTxOkCnt
EthStatTxSColCnt	PLIB_ETH_StatTxSColCnt
EthTxAcknowledgeBuffer	DRV_ETHMAC_LegacyTxAcknowledgeBuffer
EthTxAcknowledgePacket	DRV_ETHMAC_LegacyTxAcknowledgePacket
EthTxGetBufferStatus	DRV_ETHMAC_LegacyTxGetBufferStatus
EthTxGetPacketStatus	DRV_ETHMAC_LegacyTxGetPacketStatus
EthTxSendBuffer	DRV_ETHMAC_LegacyTxSendBuffer
EthTxSendPacket	DRV_ETHMAC_LegacyTxSendPacket

Table 2 shows the MPLAB Harmony Ethernet Controller Peripheral Library equivalents for functions in the MPLAB Harmony Ethernet MAC Driver and the MPLAB Harmony Ethernet PHY Driver.

The Ethernet Controller Library files are located in the following Windows® installation path:

`C:\Program Files\Microchip\xc32\<version>\pic32-libs\peripheral\eth\source`

Table 2: Ethernet Controller Peripheral Library Function Mapping

Ethernet Controller Peripheral Library Name	MPLAB Harmony Ethernet MAC Driver Name
EthClose	DRV_ETHMAC_LegacyClose
EthDescriptorGetBuffer	DRV_ETHMAC_LegacyDescriptorGetBuffer
EthDescriptorsPoolAdd	DRV_ETHMAC_LegacyDescriptorsPoolAdd
EthDescriptorsPoolCleanUp	DRV_ETHMAC_LegacyDescriptorsPoolCleanUp
EthDescriptorsPoolRemove	DRV_ETHMAC_LegacyDescriptorsPoolRemove
EthInit	DRV_ETHMAC_LegacyInit
EthMACOpen	DRV_ETHMAC_LegacyMACOpen
EthRxAcknowledgeBuffer	DRV_ETHMAC_LegacyRxAcknowledgeBuffer
EthRxAcknowledgePacket	DRV_ETHMAC_LegacyRxAcknowledgePacket
EthRxBuffersAppend	DRV_ETHMAC_LegacyRxBuffersAppend
EthRxGetBuffer	DRV_ETHMAC_LegacyRxGetBuffer
EthRxGetPacket	DRV_ETHMAC_LegacyRxGetPacket
EthTxAcknowledgeBuffer	DRV_ETHMAC_LegacyTxAcknowledgeBuffer
EthTxAcknowledgePacket	DRV_ETHMAC_LegacyTxAcknowledgePacket
EthTxGetBufferStatus	DRV_ETHMAC_LegacyTxGetBufferStatus
EthTxGetPacketStatus	DRV_ETHMAC_LegacyTxGetPacketStatus
EthTxSendBuffer	DRV_ETHMAC_LegacyTxSendBuffer
EthTxSendPacket	DRV_ETHMAC_LegacyTxSendPacket

Ethernet Controller Peripheral Library Name	MPLAB Harmony Ethernet Peripheral Library Name
EthEventsEnableClr	PLIB_ETH_EventsEnableClr
EthEventsEnableGet	PLIB_ETH_EventsEnableGet
EthEventsEnableSet	PLIB_ETH_EventsEnableSet
EthEventsEnableWrite	PLIB_ETH_EventsEnableWrite
EthEventsGet	PLIB_ETH_EventsGet
EthMACGetAddress	PLIB_ETH_MACGetAddress
EthMACSetAddress	PLIB_ETH_MACSetAddress
EthMACSetMaxFrame	PLIB_ETH_MACSetMaxFrame
EthRxFiltersClr	PLIB_ETH_RxFiltersClr
EthRxFiltersHTSet	PLIB_ETH_RxFiltersHTSet
EthRxFiltersPMClr	PLIB_ETH_RxFiltersPMClr
EthRxFiltersPMSet	PLIB_ETH_RxFiltersPMSet
EthRxFiltersSet	PLIB_ETH_RxFiltersSet
EthRxFiltersWrite	PLIB_ETH_RxFiltersWrite
EthRxSetBufferSize	PLIB_ETH_RxSetBufferSize
EthStatRxAlgnErrCnt	PLIB_ETH_StatRxAlgnErrCnt
EthStatRxFcsErrCnt	PLIB_ETH_StatRxFcsErrCnt
EthStatRxOkCnt	PLIB_ETH_StatRxOkCnt
EthStatRxOvflCnt	PLIB_ETH_StatRxOvflCnt
EthStatTxMColCnt	PLIB_ETH_StatTxMColCnt
EthStatTxOkCnt	PLIB_ETH_StatTxOkCnt
EthStatTxSColCnt	PLIB_ETH_StatTxSColCnt

Table 3 shows the existing and new function names for the TCP/IP MAC driver.

Please note that the following v6 Beta TCP/IP Stack files are now obsolete:

- `eth_pic32_ext_phy.c`
- `eth_pic32_ext_phy_*.c`
- `eth_pic32_int_mac.c`
- `mac_events_pic32.c`

Table 3: TCP/IP MAC Driver (tcpip_mac.h) – V6 Beta TCP/IP Stack to MPLAB Harmony TCP/IP Stack Conversion

V6 Beta TCP/IP Stack Name	MPLAB Harmony Compliant Name
MAC_ADDR	TCPIP_MAC_ADDR

MAC_ETHERNET_HEADER	TCPIP_MAC_ETHERNET_HEADER
MAC_NOTIFY_HANDLER	TCPIP_MAC_NOTIFY_HANDLER
MAC_PACKET	TCPIP_MAC_PACKET
MAC_RX_PKT_STAT	TCPIP_MAC_RX_PKT_STAT
MACEventAck	TCPIP_MAC_EventAck
MACEventClearNotifyEvents	TCPIP_MAC_EventNotifyClear
MACEventGetPending	TCPIP_MAC_EventPendingEventsGet
MACEventSetNotifyEvents	TCPIP_MAC_EventNotifyEventsSet
MACEventSetNotifyHandler	TCPIP_MAC_EventNotifyHandlerSet
MACFCEnable	TCPIP_MAC_FlowControlEnable
MACFCSetPauseValue	TCPIP_MAC_FlowControlPauseValueSet
MACFCSetRxWMark	TCPIP_MAC_FlowControlRxWMarkSet
TCPIP_MAC_FC_TYPE	TCPIP_MAC_FLOWCONTROLTYPE
MACCheckLink	TCPIP_MAC_LinkCheck
MACCalcIPBufferChecksum	TCPIP_MAC_ChecksumIPBufferCalc
MACCalcRxChecksum	TCPIP_MAC_ChecksumRxCalc
MACConnect	TCPIP_MAC_Connect
MACPowerDown	TCPIP_MAC_PowerDown
MACPowerUp	TCPIP_MAC_PowerUp
MACGetHeader	TCPIP_MAC_HeaderGet
MACDiscardRx	TCPIP_MAC_RxDiscard
MACGet	TCPIP_MAC_Get
MACGetArray	TCPIP_MAC_ArrayGet
MACGetReadPtrInRx	TCPIP_MAC_RxReadPtrGet
MACGetRxSize	TCPIP_MAC_RxSizeGet
MACSetBaseReadPtr	TCPIP_MAC_ReadPtrBaseSet
MACSetReadPtr	TCPIP_MAC_ReadPtrSet
MACSetReadPtrInRx	TCPIP_MAC_RxReadPtrSet
MACIsRxReady	TCPIP_MAC_RxIsReady
MACRxAcknowledge	TCPIP_MAC_RxAcknowledge
MACRxGetPacket	TCPIP_MAC_RxPacketGet
MACRxProfile	TCPIP_MAC_RxProfile
MACRxRegisterNotifyHandler	TCPIP_MAC_RxNotifyHandlerRegister
MACRxFilterOperation	TCPIP_MAC_RxFilterOperation
MACRxFilterPatternMode	TCPIP_MAC_RxFilterPatternMode
MACRxFilterSetHashTableEntry	TCPIP_MAC_RxFilterHashTableEntrySet
MACTxPacket	TCPIP_MAC_TxPacket
MACTxProfile	TCPIP_MAC_TxProfile
MACTxRegisterNotifyHandler	TCPIP_MAC_TxNotifyHandlerRegister
MACIsTxReady	TCPIP_MAC_TxIsReady
MACFlush	TCPIP_MAC_Flush
MACGetSslBaseAddr	TCPIP_MAC_SslBaseAddrGet
MACGetTxBaseAddr	TCPIP_MAC_TxBaseAddrGet
MACGetTxBuffSize	TCPIP_MAC_TxBuffSizeGet
MACPut	TCPIP_MAC_Put
MACPutArray	TCPIP_MAC_ArrayPut

MACPutHeader	TCPIP_MAC_HeaderPut
MACSetWritePtr	TCPIP_MAC_WritePtrSet

Announce Module

This section describes the TCP/IP Stack Library Announce module.

Introduction

TCP/IP Stack Library Announce Module for Microchip Microcontrollers

This library provides the API of the Announce Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

This module will facilitate device discovery on DHCP enabled networks by broadcasting a UDP message on port 30303 whenever the local IP address changes.

Using the Library

This topic describes the basic architecture of the Announce TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `tcpip_announce_config.h`

The interface to the Announce TCP/IP Stack library is defined in the `tcpip_announce_config.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the Announce TCP/IP Stack library should include `tcpip.h`.

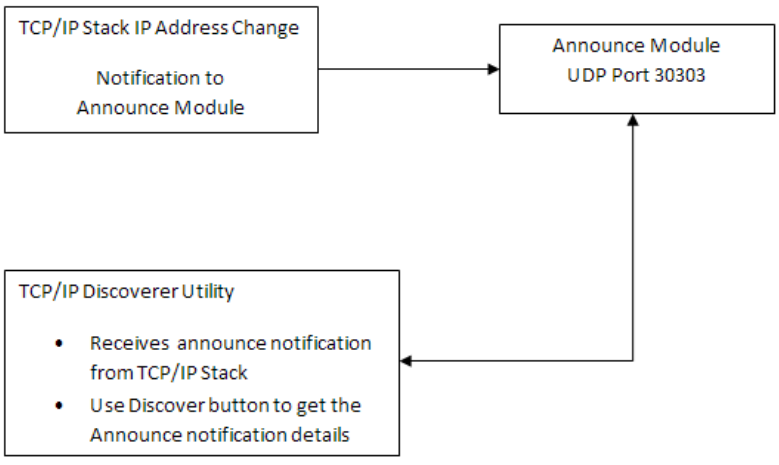
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the Announce TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Announce Software Abstraction Block Diagram



Discovering the Board

Provides information on how the Announce module works in conjunction with the TCP/IP Discoverer personal computer utility to display an interface's MAC and IPv4/IPv6 addresses.

Description

The Announce module provides an easy way to determine useful parameters for your TCP/IP node's interfaces. This module works in conjunction with the TCP/IP Discoverer personal computer utility to display an interface's MAC and IPv4/IPv6 addresses, IPv6 multicast listeners, NBNS host name, and MAC type. During stack operation several events may cause an Announce packet to be broadcast on a particular interface. These include:

- Reception of an Announce packet request on that interface
- A DHCP event on that interface (i.e., the interface has configured a new IPv4 address using DHCP)
- The addition or removal of an IPv6 Unicast Address from an interface (after Duplicate Address Detection)
- The addition or removal of an IPv6 Multicast Listener from an interface
- A user call to the ANNOUNCE_Notify function specifying the interface to which you want to send parameters

Each Announce packet is a UDP packet broadcast from the corresponding interface's IPv4 address. The payload is a series of fields beginning with an 8-bit field ID, followed by field information, followed by a field terminator (0x0D 0x0A). The current field IDs and information formats are:

- 0x01 – This field ID indicates that not all of the interface's address information could be transmitted in one packet. There is no field information for this ID.
- 0x01 0x0A 0x0D
- 0x02 – MAC address (6 bytes).
- 0x02 0x00 0x04 0xA3 0x12 0x0f 0x94 0x0D 0x0A
- 0x03 – MAC type. The field information for this ID is a variable length string describing the MAC type of this interface – "ENCJ60", "ENCJ600", "97J60", "PIC32INT", "MRF24W".
- 0x03 'P' 'I' 'C' '3' '2' 'I' 'N' 'T' 0x0D 0x0A
- 0x04 – Host name. The NBNS host name of the interface, with trailing spaces.
- 0x04 'M' 'C' 'H' 'P' 'B' 'O' 'A' 'R' 'D' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' 0x0D 0x0A
- 0x05 – The interface's IPv4 address (4 bytes, big-endian).
- 0x05 0x0A 0x00 0x01 0x03 0x0D 0x0A
- 0x06 – One of the interface's IPv6 unicast addresses (16 bytes, big-endian). Note that the interface may have more than one of these. The interface also has a corresponding solicited-node multicast address listener for every unicast address that is given. The interface may have static IP address.
- 0x06 0xFE 0x80 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x04 0xA3 0xFF 0xFE 0x12 0x0F 0x94 0x0D 0x0A
- 0x06 0xFD 0xFE 0xDC 0xDA 0x98 0x76 0x00 0x01 0x02 0x04 0xA3 0xFF 0xAA 0xAA 0xAA 0xAA 0x0D 0x0A
- 0x07 – One of the interface's IPv6 multicast address listeners (16 bytes, big-endian). The solicited-node multicast address listeners will not be included.
- 0x07 0xFF 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x0D 0x0A
- 0x08 – One of the interface's IPv6 default router address (16 bytes, big-endian).
- 0x08 0xFD 0xFE 0xDC 0xDA 0x98 0x76 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x0D 0x0A
- 0x09 – One of the interface's IPv6 default gateway address (16 bytes, big-endian).
- 0x08 0xFD 0xFE 0xDC 0xDA 0x98 0x76 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x0D 0x0A

Configuring the Library

Macros

	Name	Description
	TCPIP_ANNOUNCE_TASK_RATE	announce task rate, millseconds The default value is 333 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_ANNOUNCE_MAX_PAYLOAD	Maximum size of a payload sent once

Description

The configuration of the Announce TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the Announce TCP/IP Stack Library. Based on the selections made, the Announce TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the Announce TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_ANNOUNCE_TASK_RATE Macro

File

tcpip_announce_config.h

C

```
#define TCPIP_ANNOUNCE_TASK_RATE 333
```

Description

announce task rate, millseconds The default value is 333 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_ANNOUNCE_MAX_PAYLOAD Macro

File

tcpip_announce_config.h

C

```
#define TCPIP_ANNOUNCE_MAX_PAYLOAD (512)
```

Description

Maximum size of a payload sent once

Building the Library

This section lists the files that are available in the Announce module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/tcpip_announce.c`	Announce implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Announce module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Files

Files

Name	Description
tcpip_announce_config.h	Announce configuration file

Description

This section lists the source and header files used by the library.

tcpip_announce_config.h

Announce configuration file

Macros

	Name	Description
	TCPIP_ANNOUNCE_MAX_PAYLOAD	Maximum size of a payload sent once
	TCPIP_ANNOUNCE_TASK_RATE	announce task rate, millseconds The default value is 333 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

Announce Configuration file

This file contains the Announce module configuration options

File Name

tcpip_announce_config.h

Company

Microchip Technology Inc.

ARP Module

This section describes the TCP/IP Stack Library ARP Module.

Introduction

TCP/IP Stack Library Address Resolution Protocol (ARP) Module for Microchip Microcontrollers

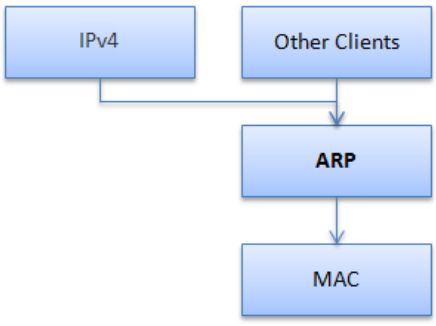
This library provides the API of the ARP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The Address Resolution Protocol, or ARP, is a foundation layer of TCP/IP. It translates IP addresses to physical MAC addresses

TCP and UDP applications will not need to access ARP directly. The IPv4 module will handle the ARP operations transparently.

Responses to incoming ARP requests are processed automatically. Resolution of ARP requests follows a simple state machine, as indicated in the following diagram.



Using the Library

This topic describes the basic architecture of the ARP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `arp.h`

The interface to the arp TCP/IP Stack library is defined in the `arp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the ARP TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

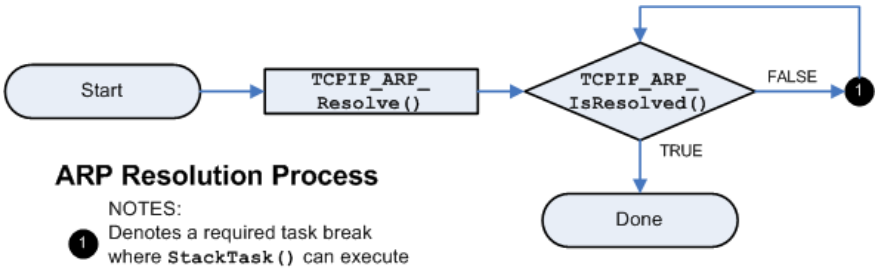
Abstraction Model

This library provides the API of the ARP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

ARP Software Abstraction Block Diagram

This module provides software abstraction of the ARP module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the ARP module.

Library Interface Section	Description
General Functions	Functions exported by the ARP library interface.
Cache Manipulation Functions	This section provides an interface to the internal data storage (cache) that the ARP module implements.
Data Types and Constants	Derived types and enumerations used by the ARP library interface.

How the Library Works

The ARP module provides address resolution capabilities for the TCP/IP stack.

The ARP module is an independent module that maintains its state across calls and updates its state machine. A data storage (cache) is maintained and updated internally for each interface existent in the system. The number of entries in each cache is configurable at initialization time.

The ARP state machine can remove entries from the cache and can provide signals when a specific entry has been resolved or has timed out. The purging of the cache is done internally base on a time-out parameter that is dynamically configurable.

The module provides a notification mechanism which simplifies the design of the ARP clients. The most important client of the ARP module is the IPv4 layer.

Configuring the Library

Macros

	Name	Description
	ARP_CACHE_DELETE_OLD	On initialization, delete the old cache if still in place Else don't re-initialize Default should be 1
	TCPIP_ARP_CACHE_ENTRIES	Number of entries in the cache. Default number of entries per interface.
	TCPIP_ARP_CACHE_ENTRY_RETRIES	Number of ARP requests generated for resolving an entry.
	TCPIP_ARP_CACHE_PENDING_ENTRY_TMO	Timeout for a cache entry pending to be solved, in seconds. The entry will be removed if the tmo elapsed and the entry has not been solved. A solved entry moves to the solved entries timeout.
	TCPIP_ARP_CACHE_PENDING_RETRY_TMO	Timeout for resending an ARP request for a pending entry. In order to prevent the ARP flooding the standard recommends it to be greater than 1 sec. It should be less than TCPIP_ARP_CACHE_PENDING_ENTRY_TMO
	TCPIP_ARP_CACHE_PERMANENT_QUOTA	Max percentage of permanent entries in the cache. Note that since permanent entries cannot be removed they tend to degrade the efficiency of the cache look up.
	TCPIP_ARP_CACHE_PURGE_QUANTA	The number of entries to delete, once the threshold is reached.
	TCPIP_ARP_CACHE_PURGE_THRESHOLD	Default purge threshold, percentage Once the number of resolved entries in the cache gets beyond the threshold some resolved entries will be purged.
	TCPIP_ARP_CACHE_SOLVED_ENTRY_TMO	Timeout for a solved entry in the cache, in seconds. The entry will be removed if the tmo elapsed and the entry has not been referenced again
	TCPIP_ARP_GRATUITOUS_PROBE_COUNT	Number of ARP requests generated when sending a gratuitous ARP probe. Default value should be 1.
	TCPIP_ARP_TASK_PROCESS_RATE	ARP task processing rate, in seconds. The ARP module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other ARP_CACHE_xxx_TMO are multiple of this The default value is 2 seconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

The configuration of the TCP/IP Stack ARP is based on the file `system_config.h` (which may include the file `arp_config.h`).

This header file contains the configuration selection for the TCP/IP Stack ARP. Based on the selections made, the TCP/IP Stack ARP may support the selected features. These configuration settings will apply to the single instance of the TCP/IP Stack ARP.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

ARP_CACHE_DELETE_OLD Macro

File

arp_config.h

C

```
#define ARP_CACHE_DELETE_OLD 1
```

Description

On initialization, delete the old cache if still in place Else don't re-initialize Default should be 1

TCPIP_ARP_CACHE_ENTRIES Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_ENTRIES 5
```

Description

Number of entries in the cache. Default number of entries per interface.

TCPIP_ARP_CACHE_ENTRY_RETRIES Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_ENTRY_RETRIES 3
```

Description

Number of ARP requests generated for resolving an entry.

TCPIP_ARP_CACHE_PENDING_ENTRY_TMO Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_PENDING_ENTRY_TMO (1 * 60)
```

Description

Timeout for a cache entry pending to be solved, in seconds. The entry will be removed if the tmo elapsed and the entry has not been solved. A solved entry moves to the solved entries timeout.

TCPIP_ARP_CACHE_PENDING_RETRY_TMO Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_PENDING_RETRY_TMO (2)
```

Description

Timeout for resending an ARP request for a pending entry. In order to prevent the ARP flooding the standard recommends it to be greater than 1 sec. It should be less than TCPIP_ARP_CACHE_PENDING_ENTRY_TMO

TCPIP_ARP_CACHE_PERMANENT_QUOTA Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_PERMANENT_QUOTA 50
```

Description

Max percentage of permanent entries in the cache. Note that since permanent entries cannot be removed they tend to degrade the efficiency of the cache look up.

TCPIP_ARP_CACHE_PURGE_QUANTA Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_PURGE_QUANTA 3
```

Description

The number of entries to delete, once the threshold is reached.

TCPIP_ARP_CACHE_PURGE_THRESHOLD Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_PURGE_THRESHOLD 75
```

Description

Default purge threshold, percentage Once the number of resolved entries in the cache gets beyond the threshold some resolved entries will be purged.

TCPIP_ARP_CACHE_SOLVED_ENTRY_TMO Macro

File

arp_config.h

C

```
#define TCPIP_ARP_CACHE_SOLVED_ENTRY_TMO (20 * 60)
```

Description

Timeout for a solved entry in the cache, in seconds. The entry will be removed if the tmo elapsed and the entry has not been referenced again

TCPIP_ARP_GRATUITOUS_PROBE_COUNT Macro

File

arp_config.h

C

```
#define TCPIP_ARP_GRATUITOUS_PROBE_COUNT 1
```

Description

Number of ARP requests generated when sending a gratuitous ARP probe. Default value should be 1.

TCPIP_ARP_TASK_PROCESS_RATE Macro

File

arp_config.h

C

```
#define TCPIP_ARP_TASK_PROCESS_RATE (2)
```

Description

ARP task processing rate, in seconds. The ARP module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other ARP_CACHE_xxx_TMO are multiple of this The default value is 2 seconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Building the Library

This section lists the files that are available in the ARP module of the TCP/IP Stack Library

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/tcpip.h	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/arp.c	ARP implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The ARP module depends on the following modules:

- TCP/IP Stack Library

Library Interface

a) General Functions

	Name	Description
	TCPIP_ARP_EntryGet	Gets the current mapping for an IP address.
	TCPIP_ARP_EntryRemove	Removes the mapping of an address, even a permanent one
	TCPIP_ARP_EntryRemoveNet	Removes all the entries belonging to a network interface.
	TCPIP_ARP_HandlerDeRegister	Deregisters the event handler
	TCPIP_ARP_HandlerRegister	Register an ARP resolve handler.
	TCPIP_ARP_IsResolved	Determines if an ARP request has been resolved yet.
	TCPIP_ARP_Probe	Transmits an ARP probe to resolve an IP address.
	TCPIP_ARP_Resolve	Transmits an ARP request to resolve an IP address.
	TCPIP_ARP_Task	Standard TCP/IP stack module task function.
	TCPIP_ARP_EntrySet	Adds an ARP cache entry for the specified interface.

b) Cache Manipulation Functions

	Name	Description
	TCPIP_ARP_CacheEntriesNoGet	Used to retrieve the number of entries for a specific interface.
	TCPIP_ARP_CacheThresholdSet	Sets the cache threshold for the specified interface in percent.
	TCPIP_ARP_EntryQuery	Queries an ARP cache entry using the index of the cache line.
	TCPIP_ARP_EntryRemoveAll	Removes all the mapping belonging to an interface.

c) Data Types and Constants

	Name	Description
	TCPIP_ARP_ENTRY_QUERY	ARP entry query.
	TCPIP_ARP_ENTRY_TYPE	Type of ARP entry.
	TCPIP_ARP_EVENT_HANDLER	Notification handler that can be called when a specific entry is resolved.
	TCPIP_ARP_EVENT_TYPE	Events reported by ARP.
	TCPIP_ARP_HANDLE	ARP handle.
	TCPIP_ARP_OPERATION_TYPE	Type of ARP operation.
	TCPIP_ARP_RESULT	ARP results (success and failure codes).
	TCPIP_ARP_MODULE_CONFIG	This is type TCPIP_ARP_MODULE_CONFIG.

Description

This section describes the Application Programming Interface (API) functions of the ARP module.

Refer to each section for a detailed description.

a) General Functions

TCPIP_ARP_EntryGet Function

Gets the current mapping for an IP address.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryGet(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd, TCPIP_MAC_ADDR* pHwAdd, bool probe);
```

Returns

- ARP_RES_ENTRY_SOLVED - if the required entry is already solved
- ARP_RES_ENTRY_QUEUED - if the required entry was already queued
- ARP_RES_ENTRY_NEW - if the operation succeeded and a new entry was added (and queued for resolving)
- ARP_RES_CACHE_FULL - if new entry could not be inserted, the cache was full
- ARP_RES_BAD_ADDRESS - bad address specified
- ARP_RES_NO_INTERFACE - no such interface

Description

If probe == false The function behaves identical to TCPIP_ARP_IsResolved(): If the corresponding MAC address exists in the cache it is copied to the user supplied pHwAdd

If probe == true The function behaves identical to TCPIP_ARP_Resolve():

- If the corresponding MAC address does not exist in the cache

this function transmits and ARP request. Upon the address resolution it calls the registered handler (if available) with the supplied notification parameter (if != 0)

- If the hardware address exists in the cache, the result is written to pHwAdd

and no network ARP request is sent

Remarks

Similar to TCPIP_ARP_Resolve + TCPIP_ARP_IsResolved, it avoids a double hash search when the mapping exists.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Interface to use
ipAdd	The IP address to get entries from
pHwAdd	pointer to store the hardware address
probe	boolean to specify if ARP probing is initiated or not

Function

TCPIP_ARP_RESULT TCPIP_ARP_EntryGet(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd,
TCPIP_MAC_ADDR* pHwAdd, bool probe)

TCPIP_ARP_EntryRemove Function

Removes the mapping of an address, even a permanent one

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryRemove(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd);
```

Returns

TCPIP_ARP_RESULT

- On Success - ARP_RES_OK
- On Failure - ARP_RES_NO_ENTRY (if no such mapping exists) ARP_RES_NO_INTERFACE (if no such interface exists)

Description

This function removes an existent mapping from the selected interface cache.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Interface to use
ipAdd	IP Address to remove entries for

Function

TCPIP_ARP_RESULT TCPIP_ARP_EntryRemove(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd);

TCPIP_ARP_EntryRemoveNet Function

Removes all the entries belonging to a network interface.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryRemoveNet(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd, IPV4_ADDR* mask,
TCPIP_ARP_ENTRY_TYPE type);
```

Returns

TCPIP_ARP_RESULT

- On Success - ARP_RES_OK
- On Failure - ARP_RES_BAD_TYPE (if no such type exists) ARP_RES_NO_INTERFACE (if no such interface exists)

Description

This function removes all existent mappings belonging to a network interface. The performed operation: if(entry->type == type and entry->ipAdd & mask == ipAdd & mask) then remove entry

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Interface handle to use
ipAdd	IP address
mask	IP address of mask

type	valid types of entries to remove: • ARP_ENTRY_TYPE_PERMANENT • ARP_ENTRY_TYPE_COMPLETE • ARP_ENTRY_TYPE_INCOMPLETE • ARP_ENTRY_TYPE_ANY

Function

TCPIP_ARP_RESULT TCPIP_ARP_EntryRemoveNet(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd, IPV4_ADDR* mask, TCPIP_ARP_ENTRY_TYPE type)

TCPIP_ARP_HandlerDeRegister Function

Deregisters the event handler

File

arp.h

C

```
bool TCPIP_ARP_HandlerDeRegister(TCPIP_ARP_HANDLE hArp);
```

Returns

Boolean

- On Success - true
- On Failure - false (If no such handler registered)

Description

Deregisters a previously registered ARP handler

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hArp	ARP Handle

Function

bool TCPIP_ARP_HandlerDeRegister(TCPIP_ARP_HANDLE hArp)

TCPIP_ARP_HandlerRegister Function

Register an ARP resolve handler.

File

arp.h

C

```
TCPIP_ARP_HANDLE TCPIP_ARP_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_ARP_EVENT_HANDLER handler, const void* hParam);
```

Returns

TCPIP_ARP_HANDLE

- On Success - Returns a valid handle
- On Failure - Null handle

Description

This function will register a notification handler with the ARP module.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Specifies interface to register on. Use hNet == 0 to register on all interfaces available.
handler	Handler to be called for event.
hParam	The hParam is passed by the client and will be used by the ARP when the notification is made. It is used for per-thread content or if more modules, for example, share the same handler and need a way to differentiate the callback.

Function

TCPIP_ARP_HANDLE TCPIP_ARP_HandlerRegister(TCPIP_NET_HANDLE hNet,
TCPIP_ARP_EVENT_HANDLER handler, const void* hParam)

TCPIP_ARP_IsResolved Function

Determines if an ARP request has been resolved yet.

File

arp.h

C

```
bool TCPIP_ARP_IsResolved(TCPIP_NET_HANDLE hNet, IPV4_ADDR* IPAddr, TCPIP_MAC_ADDR* MACAddr);
```

Description

This function checks if an ARP request has been resolved yet, and if so, stores the resolved MAC address in the pointer provided.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	interface to use
IPAddr	The IP address to be resolved. This must match the IP address provided to the TCPIP_ARP_Resolve() function call.
MACAddr	A buffer to store the corresponding MAC address retrieved from the ARP query.

Function

bool TCPIP_ARP_IsResolved(TCPIP_NET_HANDLE hNet, IPV4_ADDR* IPAddr, TCPIP_MAC_ADDR* MACAddr)

TCPIP_ARP_Probe Function

Transmits an ARP probe to resolve an IP address.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_Probe(TCPIP_NET_HANDLE hNet, IPV4_ADDR* IPAddr, IPV4_ADDR* srcAddr,
TCPIP_ARP_OPERATION_TYPE opType);
```

Returns

- ARP_RES_ENTRY_SOLVED - if the required entry is already solved

- ARP_RES_ENTRY_QUEUED - if the required entry was already queued
- ARP_RES_ENTRY_NEW - if the operation succeeded and a new entry was added (and queued for resolving)
- ARP_RES_CACHE_FULL - if new entry could not be inserted, the cache was full
- ARP_RES_BAD_ADDRESS - bad address specified
- ARP_RES_NO_INTERFACE - no such interface

Description

This function transmits and ARP probe to determine the hardware address of a given IP address. The packet will use the type of operation and the source address specified as parameters.

Remarks

This function is a more advanced version of TCPIP_ARP_Resolve. It allows the caller to specify the operation type and the source address of the outgoing ARP packet. It also supports the ARP flags defined in TCPIP_ARP_OPERATION_TYPE.

No check is done for IPAddr to be valid.

To retrieve the ARP query result, call the TCPIP_ARP_IsResolved function.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	interface to use
IPAddr	The IP address to be resolved. The address must be specified in network byte order (big endian).
srcAddr	The source address to be used in the ARP packet
opType	Operation code to be set in the outgoing ARP packet

Function

TCPIP_ARP_RESULT TCPIP_ARP_Probe(TCPIP_NET_HANDLE hNet, IPV4_ADDR* IPAddr,
IPV4_ADDR* srcAddr, TCPIP_ARP_OPERATION_TYPE opType)

TCPIP_ARP_Resolve Function

Transmits an ARP request to resolve an IP address.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_Resolve(TCPIP_NET_HANDLE hNet, IPV4_ADDR* IPAddr);
```

Returns

An element from the TCPIP_ARP_RESULT enumeration. ARP_RES_ENTRY_SOLVED - if the required entry is already solved ARP_RES_ENTRY_QUEUED - if the required entry was already queued ARP_RES_ENTRY_NEW - if the operation succeeded and a new entry was added (and queued for resolving) ARP_RES_CACHE_FULL - if new entry could not be inserted, the cache was full ARP_RES_BAD_ADDRESS - bad address specified ARP_RES_NO_INTERFACE - no such interface

Description

This function transmits and ARP request to determine the hardware address of a given IP address. Upon the address resolution it calls the registered handler (if available) with the supplied notification parameter (if != 0)

Remarks

To retrieve the ARP query result, call the TCPIP_ARP_IsResolved function.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	interface to use

IPAddr	The IP address to be resolved. The address must be specified in network byte order (big endian).

Function

TCPIP_ARP_RESULT TCPIP_ARP_Resolve(TCPIP_NET_HANDLE hNet, IPV4_ADDR* IPAddr)

TCPIP_ARP_Task Function

Standard TCP/IP stack module task function.

File

arp.h

C

```
void TCPIP_ARP_Task();
```

Returns

None.

Description

Performs ARP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Function

void TCPIP_ARP_Task(void)

TCPIP_ARP_EntrySet Function

Adds an ARP cache entry for the specified interface.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntrySet(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd, TCPIP_MAC_ADDR* hwAdd, bool perm);
```

Returns

- On Success - ARP_RES_OK/ARP_RES_ENTRY_EXIST
- On Failure - An Error for example, cache is full with permanent entries that cannot be purged or the permanent quota exceeded)

Description

This function will add an entry to the selected interface cache. The entry can be marked as as permanent (not subject to timeouts or updates from the network). If cache is full, an entry will be deleted to make room.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Interface to use
ipAdd	The IP address
hwAdd	The mapping MAC address for the supplied ipAdd

perm	if true, the entry will be marked as permanent

Function

TCPIP_ARP_RESULT TCPIP_ARP_EntrySet(TCPIP_NET_HANDLE hNet, IPV4_ADDR* ipAdd,
TCPIP_MAC_ADDR* hwAdd, bool perm)

b) Cache Manipulation Functions

TCPIP_ARP_CacheEntriesNoGet Function

Used to retrieve the number of entries for a specific interface.

File

arp.h

C

```
size_t TCPIP_ARP_CacheEntriesNoGet(TCPIP_NET_HANDLE hNet, TCPIP_ARP_ENTRY_TYPE type);
```

Returns

The number of entries of the specified type per interface.

Description

The function will return the number of entries of the specified type that are currently in the cache.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Interface to use
type	Type of ARP entry

Function

size_t TCPIP_ARP_CacheEntriesNoGet(TCPIP_NET_HANDLE hNet, TCPIP_ARP_ENTRY_TYPE type);

TCPIP_ARP_CacheThresholdSet Function

Sets the cache threshold for the specified interface in percent.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_CacheThresholdSet(TCPIP_NET_HANDLE hNet, int purgeThres, int purgeEntries);
```

Returns

- On Success - ARP_RES_OK
- On Failure - ARP_RES_NO_INTERFACE (if no such interface exists)

Description

This function sets the current value of the cache threshold for the selected interface. During the ARP operation, once the number of entries in the cache is greater than the purge threshold a number of purgeEntries (usually one) will be discarded

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	Interface handle to use
purgeThres	Threshold to start cache purging
purgeEntries	Number of entries to purge

Function

TCPIP_ARP_RESULT TCPIP_ARP_CacheThresholdSet(TCPIP_NET_HANDLE hNet,
int purgeThres, int purgeEntries);

TCPIP_ARP_EntryQuery Function

Queries an ARP cache entry using the index of the cache line.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryQuery(TCPIP_NET_HANDLE hNet, size_t index, TCPIP_ARP_ENTRY_QUERY*
pArpQuery);
```

Returns

- On Success - ARP_RES_OK
- On Failure - ARP_RES_BAD_INDEX (if index is out of range) ARP_RES_NO_INTERFACE (if no such interface exists)

Description

This function can be used for displaying the cache contents.

Remarks

None.

Preconditions

The ARP module should have been initialized. The index has to be a valid one. For example, TCPIP_ARP_CacheEntriesNoGet populates the supplied TCPIP_ARP_ENTRY_QUERY query entry.

Parameters

Parameters	Description
hNet	Interface handle to use
index	Index to cache
pArpQuery	entry type, IP address, hardware address

Function

TCPIP_ARP_RESULT TCPIP_ARP_EntryQuery(TCPIP_NET_HANDLE hNet, size_t index,
TCPIP_ARP_ENTRY_QUERY* pArpQuery)

TCPIP_ARP_EntryRemoveAll Function

Removes all the mapping belonging to an interface.

File

arp.h

C

```
TCPIP_ARP_RESULT TCPIP_ARP_EntryRemoveAll(TCPIP_NET_HANDLE hNet);
```

Returns

TCPIP_ARP_RESULT

- On Success - ARP_RES_OK
- On Failure - ARP_RES_NO_INTERFACE (if no such interface exists)

Description

This function removes all existent mappings from the selected interface cache.

Remarks

None.

Preconditions

The ARP module should have been initialized.

Parameters

Parameters	Description
hNet	network interface handle

Function

TCPIP_ARP_RESULT TCPIP_ARP_EntryRemoveAll(TCPIP_NET_HANDLE hNet)

c) Data Types and Constants

TCPIP_ARP_ENTRY_QUERY Structure

ARP entry query.

File

arp.h

C

```
typedef struct {
  TCPIP_ARP_ENTRY_TYPE entryType;
  IPV4_ADDR entryIpAdd;
  TCPIP_MAC_ADDR entryHwAdd;
} TCPIP_ARP_ENTRY_QUERY;
```

Members

Members	Description
TCPIP_ARP_ENTRY_TYPE entryType;	what entry type
IPV4_ADDR entryIpAdd;	the entry IP address
TCPIP_MAC_ADDR entryHwAdd;	the entry hardware address

Description

Structure: TCPIP_ARP_ENTRY_QUERY

Data structure for an ARP query.

TCPIP_ARP_ENTRY_TYPE Enumeration

Type of ARP entry.

File

arp.h

C

```
typedef enum {
  ARP_ENTRY_TYPE_INVALID,
  ARP_ENTRY_TYPE_PERMANENT,
  ARP_ENTRY_TYPE_COMPLETE,
  ARP_ENTRY_TYPE_INCOMPLETE,
  ARP_ENTRY_TYPE_ANY,
  ARP_ENTRY_TYPE_TOTAL
} TCPIP_ARP_ENTRY_TYPE;
```

Members

Members	Description
ARP_ENTRY_TYPE_INVALID	empty entry
ARP_ENTRY_TYPE_PERMANENT	entry valid and permanent
ARP_ENTRY_TYPE_COMPLETE	entry valid
ARP_ENTRY_TYPE_INCOMPLETE	entry not resolved yet
ARP_ENTRY_TYPE_ANY	any busy entry (PERMANENT|COMPLETE|INCOMPLETE)
ARP_ENTRY_TYPE_TOTAL	total entries - the number of entries the cache can store

Description

Enumeration: TCPIP_ARP_ENTRY_TYPE

List of different ARP cache entries.

Remarks

None.

TCPIP_ARP_EVENT_HANDLER Type

Notification handler that can be called when a specific entry is resolved.

File

arp.h

C

```
typedef void (* TCPIP_ARP_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, const IPV4_ADDR* ipAdd, const
TCPIP_MAC_ADDR* MACAddr, TCPIP_ARP_EVENT_TYPE evType, const void* param);
```

Description

Type: TCPIP_ARP_EVENT_HANDLER

The format of a notification handler registered with the ARP module.

Remarks

The parameter member significance is module dependent. It can be an IP address, pointer to some other structure, etc. The handler is called when an event of some sort occurs for a particular IP address entry. If pNetIf == 0, the notification is called for events on any interface

TCPIP_ARP_EVENT_TYPE Enumeration

Events reported by ARP.

File

arp.h

C

```
typedef enum {
  ARP_EVENT_SOLVED = 1,
  ARP_EVENT_UPDATED = 2,
  ARP_EVENT_PERM_UPDATE = 3,
  ARP_EVENT_REMOVED_TMO = -1,
  ARP_EVENT_REMOVED_USER = -2,
  ARP_EVENT_REMOVED_EXPIRED = -3,
  ARP_EVENT_REMOVED_PURGED = -4
} TCPIP_ARP_EVENT_TYPE;
```

Members

Members	Description
ARP_EVENT_SOLVED = 1	a queued cache entry was solved;
ARP_EVENT_UPDATED = 2	an existent cache entry was updated
ARP_EVENT_PERM_UPDATE = 3	an update for an permanent entry was received however the permanent entry was not updated error events - entries removed from ARP cache
ARP_EVENT_REMOVED_TMO = -1	an entry could not be solved and a tmo occurred
ARP_EVENT_REMOVED_USER = -2	a queued cache entry was removed by ARP user

ARP_EVENT_REMOVED_EXPIRED = -3	a queued cache entry expired and was removed
ARP_EVENT_REMOVED_PURGED = -4	a queued cache entry was purged

Description

Enumeration: TCPIP_ARP_EVENT_TYPE

List of events reported by ARP.

Remarks

Possibly multiple events can be set, where it makes sense.

TCPIP_ARP_HANDLE Type

ARP handle.

File

arp.h

C

```
typedef const void* TCPIP_ARP_HANDLE;
```

Description

Type: TCPIP_ARP_HANDLE

A handle that a client needs to use when deregistering a notification handler.

Remarks

This handle can be used by the client after the event handler has been registered.

TCPIP_ARP_OPERATION_TYPE Enumeration

Type of ARP operation.

File

arp.h

C

```
typedef enum {
  ARP_OPERATION_REQ = 1,
  ARP_OPERATION_RESP = 2,
  ARP_OPERATION_MASK = 0x000f,
  ARP_OPERATION_CONFIGURE = 0x1000,
  ARP_OPERATION_GRATUITOUS = 0x2000,
  ARP_OPERATION_PROBE_ONLY = 0x4000
} TCPIP_ARP_OPERATION_TYPE;
```

Members

Members	Description
ARP_OPERATION_REQ = 1	ARP request
ARP_OPERATION_RESP = 2	ARP response
ARP_OPERATION_MASK = 0x000f	extract ARP operation
ARP_OPERATION_CONFIGURE = 0x1000	stack configuration ARP packet
ARP_OPERATION_GRATUITOUS = 0x2000	stack gratuitous ARP packet
ARP_OPERATION_PROBE_ONLY = 0x4000	an ARP probe is sent only once, the target address is not stored

Description

Enumeration: TCPIP_ARP_OPERATION_TYPE

Operation to be performed by an ARP probe.

Remarks

Used for low level functionality, TCPIP_ARP_Probe.

TCPIP_ARP_RESULT Enumeration

ARP results (success and failure codes).

File

arp.h

C

```
typedef enum {
  ARP_RES_OK = 0,
  ARP_RES_ENTRY_NEW,
  ARP_RES_ENTRY_SOLVED,
  ARP_RES_ENTRY_QUEUED,
  ARP_RES_ENTRY_EXIST,
  ARP_RES_PERM_QUOTA_EXCEED,
  ARP_RES_PROBE_OK,
  ARP_RES_NO_ENTRY = -1,
  ARP_RES_CACHE_FULL = -2,
  ARP_RES_TX_FAILED = -3,
  ARP_RES_BAD_INDEX = -4,
  ARP_RES_BAD_ADDRESS = -5,
  ARP_RES_NO_INTERFACE = -6,
  ARP_RES_BAD_TYPE = -7,
  ARP_RES_CONFIGURE_ERR = -8,
  ARP_RES_PROBE_FAILED = -9
} TCPIP_ARP_RESULT;
```

Members

Members	Description
ARP_RES_OK = 0	operation succeeded
ARP_RES_ENTRY_NEW	operation succeeded and a new entry was added
ARP_RES_ENTRY_SOLVED	the required entry is already solved
ARP_RES_ENTRY_QUEUED	the required entry was already queued
ARP_RES_ENTRY_EXIST	the required entry was already cached
ARP_RES_PERM_QUOTA_EXCEED	info: the quota of permanent entries was exceeded
ARP_RES_PROBE_OK	requested probe sent
ARP_RES_NO_ENTRY = -1	no such entry exists
ARP_RES_CACHE_FULL = -2	the cache is full and no entry could be removed to make room
ARP_RES_TX_FAILED = -3	failed to transmit an ARP message
ARP_RES_BAD_INDEX = -4	bad query index
ARP_RES_BAD_ADDRESS = -5	bad IP address specified
ARP_RES_NO_INTERFACE = -6	no such interface exists
ARP_RES_BAD_TYPE = -7	no such type is valid/exists
ARP_RES_CONFIGURE_ERR = -8	interface is configuring now, no ARP probes
ARP_RES_PROBE_FAILED = -9	requested probe failed

Description

Enumeration: TCPIP_ARP_RESULT

Various definitions for success and failure codes.

Remarks

None.

TCPIP_ARP_MODULE_CONFIG Structure

File

arp.h

C

```
typedef struct {
  size_t cacheEntries;
```

```
    bool deleteOld;
    int entrySolvedTmo;
    int entryPendingTmo;
    int entryRetryTmo;
    int permQuota;
    int purgeThres;
    int purgeQuanta;
    int retries;
    int gratProbeCount;
} TCPIP_ARP_MODULE_CONFIG;
```

Members

Members	Description
size_t cacheEntries;	cache entries for this interface
bool deleteOld;	delete old cache if still in place, else don't reinitialize it
int entrySolvedTmo;	solved entry removed after this tmo if not referenced - seconds
int entryPendingTmo;	timeout for a pending to be solved entry in the cache, in seconds
int entryRetryTmo;	timeout for resending an ARP request for a pending entry - seconds 1 sec < tmo < entryPendingTmo
int permQuota;	max percentage of permanent entries allowed in the cache -
int purgeThres;	purge threshold -
int purgeQuanta;	no of entries to delete once the threshold is reached
int retries;	no of retries for resolving an entry
int gratProbeCount;	no of retries done for a gratuitous ARP request

Description

This is type TCPIP_ARP_MODULE_CONFIG.

Files

Files

Name	Description
arp.h	Address Resolution Protocol (ARP) header file.
arp_config.h	ARP configuration file

Description

This section lists the source and header files used by the library.

arp.h

Address Resolution Protocol (ARP) header file.

Enumerations

	Name	Description
	TCPIP_ARP_ENTRY_TYPE	Type of ARP entry.
	TCPIP_ARP_EVENT_TYPE	Events reported by ARP.
	TCPIP_ARP_OPERATION_TYPE	Type of ARP operation.
	TCPIP_ARP_RESULT	ARP results (success and failure codes).

Functions

	Name	Description
	TCPIP_ARP_CacheEntriesNoGet	Used to retrieve the number of entries for a specific interface.
	TCPIP_ARP_CacheThresholdSet	Sets the cache threshold for the specified interface in percent.
	TCPIP_ARP_EntryGet	Gets the current mapping for an IP address.
	TCPIP_ARP_EntryQuery	Queries an ARP cache entry using the index of the cache line.
	TCPIP_ARP_EntryRemove	Removes the mapping of an address, even a permanent one
	TCPIP_ARP_EntryRemoveAll	Removes all the mapping belonging to an interface.
	TCPIP_ARP_EntryRemoveNet	Removes all the entries belonging to a network interface.
	TCPIP_ARP_EntrySet	Adds an ARP cache entry for the specified interface.

	TCPIP_ARP_HandlerDeRegister	Deregisters the event handler
	TCPIP_ARP_HandlerRegister	Register an ARP resolve handler.
	TCPIP_ARP_IsResolved	Determines if an ARP request has been resolved yet.
	TCPIP_ARP_Probe	Transmits an ARP probe to resolve an IP address.
	TCPIP_ARP_Resolve	Transmits an ARP request to resolve an IP address.
	TCPIP_ARP_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_ARP_ENTRY_QUERY	ARP entry query.
	TCPIP_ARP_MODULE_CONFIG	This is type TCPIP_ARP_MODULE_CONFIG.

Types

	Name	Description
	TCPIP_ARP_EVENT_HANDLER	Notification handler that can be called when a specific entry is resolved.
	TCPIP_ARP_HANDLE	ARP handle.

Description

Address Resolution Protocol (ARP) Header file

This source file contains the ARP module API definitions.

File Name

arp.h

Company

Microchip Technology Inc.

arp_config.h

ARP configuration file

Macros

	Name	Description
	ARP_CACHE_DELETE_OLD	On initialization, delete the old cache if still in place Else don't re-initialize Default should be 1
	TCPIP_ARP_CACHE_ENTRIES	Number of entries in the cache. Default number of entries per interface.
	TCPIP_ARP_CACHE_ENTRY_RETRIES	Number of ARP requests generated for resolving an entry.
	TCPIP_ARP_CACHE_PENDING_ENTRY_TMO	Timeout for a cache entry pending to be solved, in seconds. The entry will be removed if the tmo elapsed and the entry has not been solved. A solved entry moves to the solved entries timeout.
	TCPIP_ARP_CACHE_PENDING_RETRY_TMO	Timeout for resending an ARP request for a pending entry. In order to prevent the ARP flooding the standard recommends it to be greater than 1 sec. It should be less than TCPIP_ARP_CACHE_PENDING_ENTRY_TMO
	TCPIP_ARP_CACHE_PERMANENT_QUOTA	Max percentage of permanent entries in the cache. Note that since permanent entries cannot be removed they tend to degrade the efficiency of the cache look up.
	TCPIP_ARP_CACHE_PURGE_QUANTA	The number of entries to delete, once the threshold is reached.
	TCPIP_ARP_CACHE_PURGE_THRESHOLD	Default purge threshold, percentage Once the number of resolved entries in the cache gets beyond the threshold some resolved entries will be purged.
	TCPIP_ARP_CACHE_SOLVED_ENTRY_TMO	Timeout for a solved entry in the cache, in seconds. The entry will be removed if the tmo elapsed and the entry has not been referenced again
	TCPIP_ARP_GRATUITOUS_PROBE_COUNT	Number of ARP requests generated when sending a gratuitous ARP probe. Default value should be 1.
	TCPIP_ARP_TASK_PROCESS_RATE	ARP task processing rate, in seconds. The ARP module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other ARP_CACHE_xxx_TMO are multiple of this The default value is 2 seconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

Address Resolution Protocol (ARP) Configuration file

This file contains the ARP module configuration options

File Name

arp_config.h

Company

Microchip Technology Inc.

Berkeley Module

This section describes the TCP/IP Stack Library Berkeley Module.

Introduction

TCP/IP Stack Library Berkeley Module for Microchip Microcontrollers

This library provides the API of the Berkeley module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The Berkeley Socket Distribution (BSD) APIs provide a BSD wrapper to the native Microchip TCP/IP Stack APIs. Using this interface, programmers familiar with BSD sockets can quickly develop applications using Microchip's TCP/IP Stack.

Using the Library

This topic describes the basic architecture of the Berkeley TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `berkeley_api.h`

The interface to the Berkeley TCP/IP Stack library is defined in the `berkeley_api.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the berkeley TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

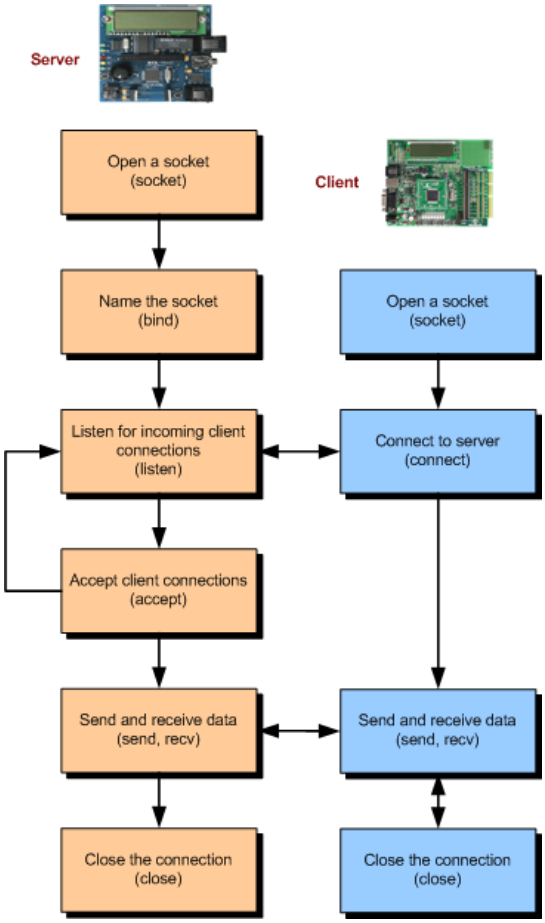
Abstraction Model

This library provides the API of the Berkeley TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Berkeley Software Abstraction Block Diagram

The following illustration shows a typical interaction for a TCP server or client using the Berkeley socket APIs.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Berkeley module.

Library Interface Section	Description
Functions	Routines for Configuring the Berkeley Socket API
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	MAX_BSD_SOCKETS	Berkeley API max number of sockets simultaneous supported

Description

The configuration of the Berkeley TCP/IP Stack is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the Berkeley TCP/IP Stack. Based on the selections made, the Berkeley TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the Berkeley TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

MAX_BSD_SOCKETS Macro

File

berkeley_api_config.h

C

```
#define MAX_BSD_SOCKETS (4)
```

Description

Berkeley API max number of sockets simultaneous supported

Building the Library

This section lists the files that are available in the Berkeley module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/berkeley_api.c`	Berkeley API implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Berkeley module depends on the following modules:

- TCP/IP Stack Library
- UDP Module
- TCP Module

Library Interface

a) Functions

	Name	Description
	accept	This function accepts connection requests queued for a listening socket.
	bind	This function assigns a name to the socket descriptor.
	closesocket	The closesocket function closes an existing socket.
	connect	This function connects to the peer communications end point.
	gethostname	Returns the standard host name for the system.
	listen	The listen function sets the specified socket in a listen mode.
	recv	The recv() function is used to receive incoming data that has been queued for a socket.
	recvfrom	The recvfrom() function is used to receive incoming data that has been queued for a socket.
	send	The send function is used to send outgoing data on an already connected socket.
	sendto	This function used to send the data for both connection oriented and connection-less sockets.

	socket	This function creates a new Berkeley socket.
	getsockopt	Allows setting options to a socket like adjust RX/TX buffer size, etc
	setsockopt	Allows setting options to a socket like adjust RX/TX buffer size, etc
	gethostbyname	The gethostbyname function returns a structure of type hostent for the given host name.
	getsockname	Returns the current address to which the socket is bound.
	TCPIP_BSD_Socket	Returns the native socket number associated with the BSD socket.
	freeaddrinfo	Frees the memory allocated by getaddrinfo
	getaddrinfo	Does an address look up for the provided node name.

b) Data Types and Constants

	Name	Description
	AF_INET	Internet Address Family - IPv4, UDP, TCP, etc.
	INADDR_ANY	IP address for server binding.
	INVALID_TCP_PORT	Invalid TCP port
	IP_ADDR_ANY	IP Address for server binding
	IPPROTO_IP	Indicates IP pseudo-protocol.
	IPPROTO_TCP	Indicates TCP level options
	IPPROTO_UDP	Indicates UDP level options
	SOCK_DGRAM	Connectionless datagram socket. Use UDP for the Internet address family.
	SOCK_STREAM	Connection based byte streams. Use TCP for the Internet address family.
	SOCKET_CNXN_IN_PROGRESS	Socket connection state.
	SOCKET_DISCONNECTED	Socket disconnected
	SOCKET_ERROR	Socket error
	SOCKADDR	generic address structure for all address families
	SOCKADDR_IN	In the Internet address family
	SOCKET	Socket descriptor
	in_addr	in_addr structure
	sockaddr	generic address structure for all address families
	sockaddr_in	In the Internet address family
	addrinfo	This is record addrinfo.
	EAI_AGAIN	Temporary failure in name resolution.
	EAI_BADFLAGS	Invalid value for `ai_flags' field.
	EAI_FAIL	Non-recoverable failure in name res.
	EAI_FAMILY	ai_family' not supported.
	EAI_MEMORY	Memory allocation failure.
	EAI_NONAME	NAME or SERVICE is unknown.
	EAI_OVERFLOW	Argument buffer overflow.
	EAI_SERVICE	SERVICE not supported for `ai_socktype'.
	EAI_SOCKTYPE	ai_socktype' not supported.
	EAI_SYSTEM	System error returned in `errno'.
	ICMP6_FILTER	SEt the IPv6 Filtering options - Not yet supported
	IP_ADD_MEMBERSHIP	Join a multicast group - Not yet supported
	IP_DROP_MEMBERSHIP	Leave a multicast group - Not yet supported
	IP_MULTICAST_IF	Set the Interface that multicast packets should be sent on - Not yet supported
	IP_MULTICAST_LOOP	Specify a copy of multicast packets should be delivered to the sending host - Not yet supported
	IP_MULTICAST_TTL	Set the Time to Live option for multicast packets - Not yet supported
	IP_OPTIONS	IP Header Options - Not yet supported
	IP_TOS	Type of Service - Not yet supported
	IP_TTL	Time to Live - Not yet supported
	IPPROTO_ICMPV6	Indicates IPv6 ICMP Protocol Options
	IPPROTO_IPV6	Indicates IP v6 Header options
	IPV6_CHECKSUM	Sets the IPv6 Checksum options - Not yet supported
	IPV6_JOIN_GROUP	Join an IPv6 multicast group - Not yet supported
	IPV6_LEAVE_GROUP	Leave an IPv6 multicast group - Not yet supported

	IPV6_MULTICAST_HOPS	Set the hop limit for multicast IPv6 packets - Not yet supported
	IPV6_MULTICAST_IF	Set the Interface that multicast IPv6 packets should be sent on - Not yet supported
	IPV6_MULTICAST_LOOP	Specify a copy of multicast IPv6 packets should be delivered to the sending host - Not yet supported
	IPV6_UNICAST_HOPS	Set the hop limit for unicast IPv6 packets - Not yet supported
	IPV6_V6ONLY	Sets the socket to IPv6 only - Not yet supported
	SO_BROADCAST	Enables the Socket for sending broadcast data
	SO_DEBUG	Indicates if low level debut is active - Not yet supported
	SO_DONTROUTE	Bypass normal routing - Not yet supported
	SO_KEEPALIVE	Keep the connection alive by sending periodic transmissions - Not yet supported
	SO_LINGER	Indicates if the system should send any buffered data when a socket is closed
	SO_OOBINLINE	Indicates whether or not Out of Band Data should be received inline with normal data - Not yet supported
	SO_RCVBUF	Receive Buffer Size (TCP only)
	SO_RCVLOWAT	Receive Low Water mark - Not yet supported
	SO_RCVTIMEO	Set the Receive Timeout - Not yet supported
	SO_REUSEADDR	Indicates if the local socket can be reused immediately after close - Not yet supported
	SO_SNDBUF	Send Buffer Size
	SO_SNDLOWAT	Send Low Water mark - Not yet supported
	SO_SNDTIMEO	Set the Send Timeout - Not yet supported
	SOL_SOCKET	Indicates socket level options
	TCP_NODELAY	Indicates if TCP is to buffer packets - Not yet supported
	linger	This is record linger.
	hostent	This is record hostent.
	HOST_NOT_FOUND	Authoritative Answer host not found
	NO_DATA	Valid name, no data record of requested type
	NO_RECOVERY	Non recoverable errors,
	TRY_AGAIN	Non authoritative host not found or server fail
	in6_addr	This is record in6_addr.
	sockaddr_in6	In the Internet address family
	sockaddr_storage	This is record sockaddr_storage.
	SOCKADDR_IN6	This is type SOCKADDR_IN6.
	AF_INET6	Internet Address Family - IPv6
	__ss_aligntype	Structure large enough to hold any socket address (with the historical exception of AF_UNIX). 128 bytes reserved.
	_BERKELEY_API_HEADER_FILE	This is macro _BERKELEY_API_HEADER_FILE.
	_SS_PADSIZE	This is macro _SS_PADSIZE.
	_SS_SIZE	This is macro _SS_SIZE.
	BERKELEY_MODULE_CONFIG	Berkeley API module configuration structure

Description

This section describes the Application Programming Interface (API) functions of the Berkeley module.

Refer to each section for a detailed description.

a) Functions

accept Function

This function accepts connection requests queued for a listening socket.

File

berkeley_api.h

C

```
SOCKET accept(SOCKET s, struct sockaddr* addr, int* addrlen);
```

Returns

If the accept function succeeds, it returns a non-negative integer that is a descriptor for the accepted socket. Otherwise, the value SOCKET_ERROR is returned (and errno is set accordingly).

Description

The accept function is used to accept connection requests queued for a listening socket. If a connection request is pending, accept removes the request from the queue, and a new socket is created for the connection. The original listening socket remains open and continues to queue new connection requests. The socket must be a SOCK_STREAM type socket.

Remarks

None.

Preconditions

The listen function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket. Must be bound to a local name and in listening mode.
addr	Optional pointer to a buffer that receives the address of the connecting entity.
addrlen	Optional pointer to an integer that contains the length of the address addr

Function

SOCKET accept(SOCKET s, struct sockaddr* addr, int* addrlen)

bind Function

This function assigns a name to the socket descriptor.

File

berkeley_api.h

C

```
int bind(SOCKET s, const struct sockaddr* name, int namelen);
```

Returns

If bind is successful, a value of 0 is returned. A return value of SOCKET_ERROR indicates an error (and errno is set accordingly).

Description

The bind function assigns a name to an unnamed socket. The name represents the local address of the communication endpoint. For sockets of type SOCK_STREAM, the name of the remote endpoint is assigned when a connect or accept function is executed.

Remarks

None.

Preconditions

The socket function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
name	pointer to the sockaddr structure containing the local address of the socket
namelen	length of the sockaddr structure

Function

int bind(SOCKET s, const struct sockaddr* name, int namelen)

closesocket Function

The closesocket function closes an existing socket.

File

berkeley_api.h

C

```
int closesocket(SOCKET s);
```

Returns

If closesocket is successful, a value of 0 is returned. A return value of SOCKET_ERROR (-1) indicates an error (and errno is set accordingly).

Description

The closesocket function closes an existing socket. This function releases the socket descriptor s. Any data buffered at the socket is discarded. If the socket s is no longer needed, closesocket() must be called in order to release all resources associated with s.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket

Function

int closesocket(SOCKET s)

connect Function

This function connects to the peer communications end point.

File

berkeley_api.h

C

```
int connect(SOCKET s, struct sockaddr* name, int namelen);
```

Returns

If the connect() function succeeds, it returns 0. Otherwise, the value SOCKET_ERROR is returned to indicate an error condition (and errno is set accordingly). For stream based socket, if the connection is not established yet, connect returns SOCKET_ERROR and errno = EINPROGRESS.

Description

The connect function assigns the address of the peer communications endpoint. For stream sockets, connection is established between the endpoints. For datagram sockets, an address filter is established between the endpoints until changed with another connect() function.

Remarks

None.

Preconditions

The socket function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
name	Pointer to the sockaddr structure containing the peer address and port number
namelen	Length of the sockaddr structure

Function

int connect(SOCKET s, struct sockaddr* name, int namelen)

gethostname Function

Returns the standard host name for the system.

File

berkeley_api.h

C

```
int gethostname(char* name, int namelen);
```

Returns

Success will return a value of 0. If name is too short to hold the host name or any other error occurs, SOCKET_ERROR (-1) will be returned (and errno is set accordingly). On error, *name will be unmodified and no null terminator will be generated.

Description

This function returns the standard host name of the system which is calling this function. The returned name is null-terminated.

Remarks

The function returns the host name as set on the default network interface.

Preconditions

None.

Parameters

Parameters	Description
name	Pointer to a buffer that receives the local host name
namelen	Size of the name array

Function

int gethostname(char* name, int namelen)

listen Function

The listen function sets the specified socket in a listen mode.

File

berkeley_api.h

C

```
int listen(SOCKET s, int backlog);
```

Returns

Returns 0 on success; otherwise returns SOCKET_ERROR (and errno is set accordingly).

Description

This function sets the specified socket in a listen mode. Calling the listen function indicates that the application is ready to accept connection requests arriving at a socket of type SOCK_STREAM. The connection request is queued (if possible) until accepted with an accept function. The backlog parameter defines the maximum number of pending connections that may be queued.

Remarks

None.

Preconditions

bind() must have been called on the s socket first.

Parameters

Parameters	Description
s	Socket identifier returned from a prior socket() call.
backlog	Maximum number of connection requests that can be queued. Note that each backlog requires a TCP socket to be allocated.

Function

int listen(SOCKET s, int backlog)

recv Function

The recv() function is used to receive incoming data that has been queued for a socket.

File

berkeley_api.h

C

```
int recv(SOCKET s, char* buf, int len, int flags);
```

Returns

If the recv function is successful, the socket is valid and it has pending data:

- The supplied buffer is non NULL and has non zero length, the function will return the number of bytes copied to the application buffer.
- The supplied buffer is NULL or has zero length then no data will be copied and the function will return the number of bytes pending in the socket buffer.

A return value of SOCKET_ERROR (-1) indicates an error condition (and errno set accordingly). errno is set to EWOULDBLOCK if there is no data pending in the socket buffer.

A value of zero indicates socket has been shutdown by the peer.

Description

The recv() function is used to receive incoming data that has been queued for a socket. This function can be used with both datagram and stream socket. If the available data is too large to fit in the supplied application buffer buf, excess bytes are discarded in case of SOCK_DGRAM type sockets. For SOCK_STREAM types, the data is buffered internally so the application can retrieve all data by multiple calls of recvfrom.

Remarks

None.

Preconditions

The connect function should be called for TCP and UDP sockets. Server side, accept function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
buf	Application data receive buffer
len	Buffer length in bytes
flags	No significance in this implementation

Function

int recv(SOCKET s, char* buf, int len, int flags)

recvfrom Function

The recvfrom() function is used to receive incoming data that has been queued for a socket.

File

berkeley_api.h

C

```
int recvfrom(SOCKET s, char* buf, int len, int flags, struct sockaddr* from, int* fromlen);
```

Returns

If recvfrom is successful, the number of bytes copied to the application buffer buf is returned. A return value of SOCKET_ERROR (-1) indicates an error condition (and errno is set accordingly). A value of zero indicates socket has been shutdown by the peer.

Description

The recvfrom() function is used to receive incoming data that has been queued for a socket. This function can be used with both datagram and stream type sockets. If the available data is too large to fit in the supplied application buffer buf, excess bytes are discarded in case of SOCK_DGRAM type sockets. For SOCK_STREAM types, the data is buffered internally so the application can retrieve all data by multiple calls of

recvfrom.

Remarks

None.

Preconditions

The socket function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
buf	Application data receive buffer
len	Buffer length in bytes
flags	Message flags (currently this is not supported)
from	Pointer to the sockaddr structure that will be filled in with the destination address
fromlen	Size of buffer pointed by from

Function

int recvfrom(SOCKET s, char* buf, int len, int flags, struct sockaddr* from, int* fromlen)

send Function

The send function is used to send outgoing data on an already connected socket.

File

berkeley_api.h

C

```
int send(SOCKET s, const char* buf, int len, int flags);
```

Returns

On success, send returns number of bytes sent. Zero indicates no data send. In case of error it returns SOCKET_ERROR (and errno is set accordingly).

Description

The send function is used to send outgoing data on an already connected socket. This function is used to send a reliable, ordered stream of data bytes on a socket of type SOCK_STREAM but can also be used to send datagrams on a socket of type SOCK_DGRAM.

Remarks

None.

Preconditions

The connect function should be called for TCP and UDP sockets. Server side, accept function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
buf	Application data buffer containing data to transmit
len	Length of data in bytes
flags	Message flags (currently this field is not supported)

Function

int send(SOCKET s, const char* buf, int len, int flags)

sendto Function

This function used to send the data for both connection oriented and connection-less sockets.

File

berkeley_api.h

C

```
int sendto(SOCKET s, const char* buf, int len, int flags, const struct sockaddr* to, int tolen);
```

Returns

On success, sendto returns number of bytes sent. In case of error returns SOCKET_ERROR (and errno is set accordingly).

Description

The sendto function is used to send outgoing data on a socket. The destination address is given by to and tolen. Both Datagram and stream sockets are supported.

Remarks

None.

Preconditions

The socket function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket.
buf	application data buffer containing data to transmit.
len	length of data in bytes.
flags	message flags. Currently this field is not supported.
to	Optional pointer to the sockaddr structure containing the destination address. If NULL, the currently bound remote port and IP address are used as the destination.
tolen	length of the sockaddr structure.

Function

int sendto(SOCKET s, const char* buf, int len, int flags, const struct sockaddr* to, int tolen)

socket Function

This function creates a new Berkeley socket.

File

berkeley_api.h

C

```
SOCKET socket(int af, int type, int protocol);
```

Returns

New socket descriptor. SOCKET_ERROR in case of error (and errno set accordingly).

Description

This function creates a new BSD socket for the Microchip TCP/IP stack. The return socket descriptor is used for the subsequent BSD operations.

Remarks

None.

Preconditions

BerkeleySocketInit function should be called.

Parameters

Parameters	Description
af	address family - AF_INET for IPv4, AF_INET6 for IPv6
type	socket type SOCK_DGRAM or SOCK_STREAM
protocol	IP protocol IPPROTO_UDP or IPPROTO_TCP

Function

SOCKET socket(int af, int type, int protocol)

getsockopt Function

Allows setting options to a socket like adjust RX/TX buffer size, etc

File

berkeley_api.h

C

```
int getsockopt(SOCKET s, uint32_t level, uint32_t option_name, uint8_t * option_value, uint32_t *
option_length);
```

Returns

- 0 - If successful
- -1 - If unsuccessful

Description

Various options can be set at the socket level. This function provides compatibility with BSD implementations.

Preconditions

None.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
level	On which level the operation is to be performed: • IPPROTO_IP - Applies to the IP protocol layer (not yet supported) • IPPROTO_TCP - Applies to the TCP protocol layer • SOL_SOCKET - Applies to the socket layer • IPPROTO_IPV6 - Applies to the IPv6 protocol layer (not yet supported) • IPPROTO_ICMPV6 - Applies to the ICMPv6 protocol layer (not yet supported)
option_name	The name of the option to be set: • IPPROTO_TCP • TCP_NODELAY - Specifies whether or not the stack should use the Nagle algorithm • SOL_SOCKET • SO_LINGER - Specifies what the stack should do with unsent data on close() • SO_RCVBUF - Specifies the size of the Receive Buffer (TCP only) • SO_SNDBUF - Specifies the size of the Transmit Buffer
option_value	For all values of option_name this is a pointer to the data, which in most cases is an integer. The only exception is SO_LINGER which points to a linger structure.
option_length	The size of the data pointed to by option_value

Function

int getsockopt(SOCKET s, uint32_t level, uint32_t option_name, uint8_t *option_value, uint32_t *option_length)

setsockopt Function

Allows setting options to a socket like adjust RX/TX buffer size, etc

File

berkeley_api.h

C

```
int setsockopt(SOCKET s, uint32_t level, uint32_t option_name, const uint8_t * option_value, uint32_t
option_length);
```

Returns

- 0 - If successful
- -1 - If unsuccessful

Description

Various options can be set at the socket level. This function provides compatibility with BSD implementations.

Preconditions

None.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
level	On which level the operation is to be performed: • IPPROTO_IP - Applies to the IP protocol layer (not yet supported) • IPPROTO_TCP - Applies to the TCP protocol layer • SOL_SOCKET - Applies to the socket layer • IPPROTO_IPV6 - Applies to the IPv6 protocol layer (not yet supported) • IPPROTO_ICMPV6 - Applies to the ICMPv6 protocol layer (not yet supported)
option_name	The name of the option to be set: • IPPROTO_TCP • TCP_NODELAY - Specifies whether or not the stack should use the Nagle algorithm • SOL_SOCKET • SO_LINGER - Specifies what the stack should do with unsent data on close() • SO_RCVBUF - Specifies the size of the Receive Buffer (TCP only) • SO_SNDBUF - Specifies the size of the Transmit Buffer
option_value	For all values of option_name this is a pointer to the data, which in most cases is an integer. The only exception is SO_LINGER which points to a linger structure.
option_length	The size of the data pointed to by option_value

Function

int setsockopt(SOCKET s, uint32_t level, uint32_t option_name, const uint8_t *option_value,
uint32_t option_length)

gethostbyname Function

The gethostbyname function returns a structure of type hostent for the given host name.

File

berkeley_api.h

C

```
struct hostent * gethostbyname(char * name);
```

Returns

The hostent structure on success, or NULL on error.

Remarks

This function supports IPv4 only. h_errno will be set to:

- TRY_AGAIN if the DNS query is current in progress,
- HOST_NOT_FOUND if the DNS query could not find a host name
- NO_RECOVERY if the DNS query had an unrecoverable error

Preconditions

None.

Parameters

Parameters	Description
name	The name of the host to be found

Function

struct hostent * gethostbyname(char *name)

getsockname Function

Returns the current address to which the socket is bound.

File

berkeley_api.h

C

```
int getsockname(SOCKET s, struct sockaddr * addr, int * addrlen);
```

Returns

- On success - 0 is returned and the data is updated accordingly.
- On error - -1 is returned and errno is set appropriately.

Description

The function returns the current address to which the socket is bound, in the buffer pointed to by addr.

Remarks

This function supports IPv4 connections only.

Preconditions

The socket function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket
addr	address to store the current address to which the socket is bound
addrlen	on input it should point to the space (bytes) available in addr on output it points to the actual space required for storing the bound address

Function

int getsockname(SOCKET s, struct sockaddr *addr, int *addrlen);

TCPIP_BSD_Socket Function

Returns the native socket number associated with the BSD socket.

File

berkeley_api.h

C

```
int TCPIP_BSD_Socket(SOCKET s);
```

Returns

- On success - a socket number >= 0 is returned.
- On error - -1 is returned if no such socket exists and errno is set to EBADF.

Description

The function returns the native socket number associated with the BSD socket. Using this call the caller can switch to the native TCP/IP API.

Remarks

This function works for both TCP and UDP sockets.

The native UDP sockets are created after a call to bind (server sockets) or connect (client sockets). The native TCP sockets are created after a call to listen (server sockets) or connect (client sockets). Please note that calling the TCPIP_BSD_Socket before one of these calls will return an INVALID_SOCKET.

Preconditions

The socket function should be called.

Parameters

Parameters	Description
s	Socket descriptor returned from a previous call to socket

Function

int TCPIP_BSD_Socket(SOCKET s);

freeaddrinfo Function

Frees the memory allocated by getaddrinfo

File

berkeley_api.h

C

```
void freeaddrinfo(struct addrinfo * res);
```

Returns

None.

Description

Frees the memory allocated by getaddrinfo

Preconditions

getaddrinfo must have returned successfully.

Parameters

Parameters	Description
res	Memory allocated by getaddrinfo

Function

int freeaddrinfo(struct addrinfo *res);

getaddrinfo Function

Does an address look up for the provided node name.

File

berkeley_api.h

C

```
int getaddrinfo(const char * node, const char * service, const struct addrinfo * hints, struct addrinfo ** res);
```

Returns

On success, a 0 is returned.

- EAI_NONAME - When no name could be found
- EAI_FAIL - When a failure has occurred
- EAI_AGAIN - When the look-up is in progress. Call the function again later to check the results.

Description

This function deprecates gethostbyname. It handles both IPv4 and IPv6.

Preconditions

The MPLAB Harmony DNS client services must be active.

Parameters

Parameters	Description
node	The name of the server to look up.
service	Unused

hints	hints to the function. Currently only ai_family is used. Set to either 0, AF_INET or AF_INET6
res	Memory location of where the results are. Results must be freed with freeaddrinfo

Function

int getaddrinfo(const char *node, const char *service,
const struct addrinfo *hints,
struct addrinfo **res);

b) Data Types and Constants

AF_INET Macro

File

berkeley_api.h

C

```
#define AF_INET 2            // Internet Address Family - IPv4, UDP, TCP, etc.
```

Description

Internet Address Family - IPv4, UDP, TCP, etc.

INADDR_ANY Macro

File

berkeley_api.h

C

```
#define INADDR_ANY 0x01000000u    // IP address for server binding.
```

Description

IP address for server binding.

INVALID_TCP_PORT Macro

File

berkeley_api.h

C

```
#define INVALID_TCP_PORT (0L)  //Invalid TCP port
```

Description

Invalid TCP port

IP_ADDR_ANY Macro

File

berkeley_api.h

C

```
#define IP_ADDR_ANY 0x01000000u           // IP Address for server binding
```

Description

IP Address for server binding

IPPROTO_IP Macro

File

berkeley_api.h

C

```
#define IPPROTO_IP 0   // Indicates IP pseudo-protocol.
```

Description

Indicates IP pseudo-protocol.

IPPROTO_TCP Macro

File

berkeley_api.h

C

```
#define IPPROTO_TCP 6   // Indicates TCP level options
```

Description

Indicates TCP level options

IPPROTO_UDP Macro

File

berkeley_api.h

C

```
#define IPPROTO_UDP 17  // Indicates UDP level options
```

Description

Indicates UDP level options

SOCK_DGRAM Macro

File

berkeley_api.h

C

```
#define SOCK_DGRAM 110  //Connectionless datagram socket. Use UDP for the Internet address family.
```

Description

Connectionless datagram socket. Use UDP for the Internet address family.

SOCK_STREAM Macro

File

berkeley_api.h

C

```
#define SOCK_STREAM 100  //Connection based byte streams. Use TCP for the Internet address family.
```

Description

Connection based byte streams. Use TCP for the Internet address family.

SOCKET_CNXN_IN_PROGRESS Macro

File

berkeley_api.h

C

```
#define SOCKET_CNXN_IN_PROGRESS (-2) //Socket connection state.
```

Description

Socket connection state.

SOCKET_DISCONNECTED Macro

File

berkeley_api.h

C

```
#define SOCKET_DISCONNECTED (-3) //Socket disconnected
```

Description

Socket disconnected

SOCKET_ERROR Macro

File

berkeley_api.h

C

```
#define SOCKET_ERROR (-1) //Socket error
```

Description

Socket error

SOCKADDR Type

File

berkeley_api.h

C

```
typedef struct sockaddr SOCKADDR;
```

Description

generic address structure for all address families

SOCKADDR_IN Type

File

berkeley_api.h

C

```
typedef struct sockaddr_in SOCKADDR_IN;
```

Description

In the Internet address family

SOCKET Type

File

berkeley_api.h

C

```
typedef int16_t SOCKET;
```

Description

Socket descriptor

in_addr Structure

File

berkeley_api.h

C

```
struct in_addr {
  union {
    struct {
      uint8_t s_b1, s_b2, s_b3, s_b4;
    } S_un_b;
    struct {
      uint16_t s_w1, s_w2;
    } S_un_w;
    uint32_t S_addr;
  } S_un;
};
```

Members

Members	Description

union {
struct {
uint8_t s_b1, s_b2, s_b3, s_b4;
} S_un_b;
struct {
uint16_t s_w1, s_w2;
} S_un_w;
uint32_t S_addr;
} S_un;
	union of IP address

struct { uint8_t s_b1, s_b2, s_b3, s_b4; } S_un_b;	IP address in Byte
struct { uint16_t s_w1, s_w2; } S_un_w;	IP address in Word
uint32_t S_addr;	IP address

Description

in_addr structure

sockaddr Structure

File

berkeley_api.h

C

```
struct sockaddr {
  unsigned short sa_family;
  char sa_data[14];
};
```

Members

Members	Description
unsigned short sa_family;	address family
char sa_data[14];	up to 14 bytes of direct address

Description

generic address structure for all address families

sockaddr_in Structure

File

berkeley_api.h

C

```
struct sockaddr_in {
  short sin_family;
  uint16_t sin_port;
  struct in_addr sin_addr;
  char sin_zero[8];
};
```

Members

Members	Description
short sin_family;	Address family; must be AF_INET.
uint16_t sin_port;	Internet Protocol (IP) port.
struct in_addr sin_addr;	IP address in network byte order.
char sin_zero[8];	Padding to make structure the same size as SOCKADDR.

Description

In the Internet address family

addrinfo Structure

File

berkeley_api.h

C

```
struct addrinfo {
  int ai_flags;
  int ai_family;
  int ai_socktype;
  int ai_protocol;
  size_t ai_addrlen;
  struct sockaddr * ai_addr;
  char * ai_canonname;
  struct addrinfo * ai_next;
};
```

Description

This is record addrinfo.

EAI_AGAIN Macro

File

berkeley_api.h

C

```
#define EAI_AGAIN -3    /* Temporary failure in name resolution.  */
```

Description

Temporary failure in name resolution.

EAI_BADFLAGS Macro

File

berkeley_api.h

C

```
#define EAI_BADFLAGS -1    /* Invalid value for `ai_flags' field.  */
```

Description

Invalid value for `ai_flags' field.

EAI_FAIL Macro

File

berkeley_api.h

C

```
#define EAI_FAIL -4    /* Non-recoverable failure in name res.  */
```

Description

Non-recoverable failure in name res.

EAI_FAMILY Macro

File

berkeley_api.h

C

```
#define EAI_FAMILY -6    /* `ai_family' not supported.  */
```

Description

ai_family‘ not supported.

EAI_MEMORY Macro

File

berkeley_api.h

C

```
#define EAI_MEMORY -10   /* Memory allocation failure.  */
```

Description

Memory allocation failure.

EAI_NONAME Macro

File

berkeley_api.h

C

```
#define EAI_NONAME -2    /* NAME or SERVICE is unknown.  */
```

Description

NAME or SERVICE is unknown.

EAI_OVERFLOW Macro

File

berkeley_api.h

C

```
#define EAI_OVERFLOW -12   /* Argument buffer overflow.  */
```

Description

Argument buffer overflow.

EAI_SERVICE Macro

File

berkeley_api.h

C

```
#define EAI_SERVICE -8    /* SERVICE not supported for `ai_socktype'.  */
```

Description

SERVICE not supported for `ai_socktype'.

EAI_SOCKTYPE Macro

File

berkeley_api.h

C

```
#define EAI_SOCKTYPE -7    /* `ai_socktype' not supported.  */
```

Description

ai_socktype' not supported.

EAI_SYSTEM Macro

File

berkeley_api.h

C

```
#define EAI_SYSTEM -11   /* System error returned in `errno'.  */
```

Description

System error returned in `errno'.

ICMP6_FILTER Macro

File

berkeley_api.h

C

```
#define ICMP6_FILTER 1    //SEt the IPv6 Filtering options - Not yet supported
```

Description

SEt the IPv6 Filtering options - Not yet supported

IP_ADD_MEMBERSHIP Macro

File

berkeley_api.h

C

```
#define IP_ADD_MEMBERSHIP 35  //Join a multicast group - Not yet supported
```

Description

Join a multicast group - Not yet supported

IP_DROP_MEMBERSHIP Macro

File

berkeley_api.h

C

```
#define IP_DROP_MEMBERSHIP 36  // Leave a multicast group - Not yet supported
```

Description

Leave a multicast group - Not yet supported

IP_MULTICAST_IF Macro

File

berkeley_api.h

C

```
#define IP_MULTICAST_IF 32  //Set the Interface that multicast packets should be sent on  - Not yet
supported
```

Description

Set the Interface that multicast packets should be sent on - Not yet supported

IP_MULTICAST_LOOP Macro

File

berkeley_api.h

C

```
#define IP_MULTICAST_LOOP 34  //Specify a copy of multicast packets should be delivered to the sending host
- Not yet supported
```

Description

Specify a copy of multicast packets should be delivered to the sending host - Not yet supported

IP_MULTICAST_TTL Macro

File

berkeley_api.h

C

```
#define IP_MULTICAST_TTL 33  //Set the Time to Live option for multicast packets - Not yet supported
```

Description

Set the Time to Live option for multicast packets - Not yet supported

IP_OPTIONS Macro

File

berkeley_api.h

C

```
#define IP_OPTIONS 4   //IP Header Options - Not yet supported
```

Description

IP Header Options - Not yet supported

IP_TOS Macro

File

berkeley_api.h

C

```
#define IP_TOS 1   //Type of Service - Not yet supported
```

Description

Type of Service - Not yet supported

IP_TTL Macro

File

berkeley_api.h

C

```
#define IP_TTL 2   //Time to Live - Not yet supported
```

Description

Time to Live - Not yet supported

IPPROTO_ICMPV6 Macro

File

berkeley_api.h

C

```
#define IPPROTO_ICMPV6 58  // Indicates IPv6 ICMP Protocol Options
```

Description

Indicates IPv6 ICMP Protocol Options

IPPROTO_IPV6 Macro

File

berkeley_api.h

C

```
#define IPPROTO_IPV6 41  // Indicates IP v6 Header options
```

Description

Indicates IP v6 Header options

IPV6_CHECKSUM Macro

File

berkeley_api.h

C

```
#define IPV6_CHECKSUM 7   //Sets the IPv6 Checksum options - Not yet supported
```

Description

Sets the IPv6 Checksum options - Not yet supported

IPV6_JOIN_GROUP Macro

File

berkeley_api.h

C

```
#define IPV6_JOIN_GROUP 20  //Join an IPv6 multicast group - Not yet supported
```

Description

Join an IPv6 multicast group - Not yet supported

IPV6_LEAVE_GROUP Macro

File

berkeley_api.h

C

```
#define IPV6_LEAVE_GROUP 21  //Leave an IPv6 multicast group - Not yet supported
```

Description

Leave an IPv6 multicast group - Not yet supported

IPV6_MULTICAST_HOPS Macro

File

berkeley_api.h

C

```
#define IPV6_MULTICAST_HOPS 18  //Set the hop limit for multicast IPv6 packets - Not yet supported
```

Description

Set the hop limit for multicast IPv6 packets - Not yet supported

IPV6_MULTICAST_IF Macro

File

berkeley_api.h

C

```
#define IPV6_MULTICAST_IF 17  //Set the Interface that multicast IPv6 packets should be sent on  - Not yet
supported
```

Description

Set the Interface that multicast IPv6 packets should be sent on - Not yet supported

IPV6_MULTICAST_LOOP Macro

File

berkeley_api.h

C

```
#define IPV6_MULTICAST_LOOP 19  //Specify a copy of multicast IPv6 packets should be delivered to the
sending host - Not yet supported
```

Description

Specify a copy of multicast IPv6 packets should be delivered to the sending host - Not yet supported

IPV6_UNICAST_HOPS Macro

File

berkeley_api.h

C

```
#define IPV6_UNICAST_HOPS 16  //Set the hop limit for unicast IPv6 packets - Not yet supported
```

Description

Set the hop limit for unicast IPv6 packets - Not yet supported

IPV6_V6ONLY Macro

File

berkeley_api.h

C

```
#define IPV6_V6ONLY 26  //Sets the socket to IPv6 only - Not yet supported
```

Description

Sets the socket to IPv6 only - Not yet supported

SO_BROADCAST Macro

File

berkeley_api.h

C

```
#define SO_BROADCAST 6   //Enables the Socket for sending broadcast data
```

Description

Enables the Socket for sending broadcast data

SO_DEBUG Macro

File

berkeley_api.h

C

```
#define SO_DEBUG 1   //Indicates if low level debut is active - Not yet supported
```

Description

Indicates if low level debut is active - Not yet supported

SO_DONTROUTE Macro

File

berkeley_api.h

C

```
#define SO_DONTROUTE 5   //Bypass normal routing - Not yet supported
```

Description

Bypass normal routing - Not yet supported

SO_KEEPALIVE Macro

File

berkeley_api.h

C

```
#define SO_KEEPALIVE 9   //Keep the connection alive by sending periodic transmissions - Not yet supported
```

Description

Keep the connection alive by sending periodic transmissions - Not yet supported

SO_LINGER Macro

File

berkeley_api.h

C

```
#define SO_LINGER 13  //Indicates if the system should send any buffered data when a socket is closed
```

Description

Indicates if the system should send any buffered data when a socket is closed

SO_OOBINLINE Macro

File

berkeley_api.h

C

```
#define SO_OOBINLINE 10  //Indicates whether or not Out of Band Data should be received inline with normal
data - Not yet supported
```

Description

Indicates whether or not Out of Band Data should be received inline with normal data - Not yet supported

SO_RCVBUF Macro

File

berkeley_api.h

C

```
#define SO_RCVBUF 8   //Receive Buffer Size (TCP only)
```

Description

Receive Buffer Size (TCP only)

SO_RCVLOWAT Macro

File

berkeley_api.h

C

```
#define SO_RCVLOWAT 18  //Receive Low Water mark - Not yet supported
```

Description

Receive Low Water mark - Not yet supported

SO_RCVTIMEO Macro

File

berkeley_api.h

C

```
#define SO_RCVTIMEO 20  //Set the Receive Timeout - Not yet supported
```

Description

Set the Receive Timeout - Not yet supported

SO_REUSEADDR Macro

File

berkeley_api.h

C

```
#define SO_REUSEADDR 2   //Indicates if the local socket can be reused immediately after close - Not yet
supported
```

Description

Indicates if the local socket can be reused immediately after close - Not yet supported

SO_SNDBUF Macro

File

berkeley_api.h

C

```
#define SO_SNDBUF 7   //Send Buffer Size
```

Description

Send Buffer Size

SO_SNDLOWAT Macro

File

berkeley_api.h

C

```
#define SO_SNDLOWAT 19  //Send Low Water mark - Not yet supported
```

Description

Send Low Water mark - Not yet supported

SO_SNDTIMEO Macro

File

berkeley_api.h

C

```
#define SO_SNDTIMEO 21  //Set the Send Timeout - Not yet supported
```

Description

Set the Send Timeout - Not yet supported

SOL_SOCKET Macro

File

berkeley_api.h

C

```
#define SOL_SOCKET 1   // Indicates socket level options
```

Description

Indicates socket level options

TCP_NODELAY Macro

File

berkeley_api.h

C

```
#define TCP_NODELAY 1 //Indicates if TCP is to buffer packets - Not yet supported
```

Description

Indicates if TCP is to buffer packets - Not yet supported

linger Structure

File

berkeley_api.h

C

```
struct linger {
  int l_onoff;
  int l_linger;
};
```

Members

Members	Description
int l_onoff;	Determines if the option is set
int l_linger;	Time to wait before data is discarded

Description

This is record linger.

hostent Structure

File

berkeley_api.h

C

```
struct hostent {
  char * h_name;
  char ** h_alias;
  int h_addrtype;
  int h_length;
  char ** h_addr_list;
};
```

Members

Members	Description
char * h_name;	Points to a string containing the name of the host
char ** h_alias;	points to a null terminated list of pointers that point to the aliases of the host
int h_addrtype;	Contains the address type for the host, currently only AF_INET is supported
int h_length;	Contains the length of the h_addr_list
char ** h_addr_list;	Points to a NULL terminated list of pointers that point to the address of the host

Description

This is record hostent.

HOST_NOT_FOUND Macro

File

berkeley_api.h

C

```
#define HOST_NOT_FOUND 1    // Authoritative Answer host not found
```

Description

Authoritative Answer host not found

NO_DATA Macro

File

berkeley_api.h

C

```
#define NO_DATA 4    // Valid name, no data record of requested type
```

Description

Valid name, no data record of requested type

NO_RECOVERY Macro

File

berkeley_api.h

C

```
#define NO_RECOVERY 3    //Non recoverable errors,
```

Description

Non recoverable errors,

TRY_AGAIN Macro

File

berkeley_api.h

C

```
#define TRY_AGAIN 2    //Non authoritative host not found or server fail
```

Description

Non authoritative host not found or server fail

in6_addr Structure

File

berkeley_api.h

C

```
struct in6_addr {
  union {
    uint8_t u6_addr8[16];
    uint16_t u6_addr16[8];
    uint32_t u6_addr32[4];
  } in6_u;
};
```

Members

Members	Description
uint8_t u6_addr8[16];	IP address in Bytes
uint16_t u6_addr16[8];	IP address in 16 bit Words
uint32_t u6_addr32[4];	IP address in 32 bit Words

Description

This is record in6_addr.

sockaddr_in6 Structure

File

berkeley_api.h

C

```
struct sockaddr_in6 {
  short sin6_family;
  uint16_t sin6_port;
  uint32_t sin6_flowinfo;
  struct in6_addr sin6_addr;
  uint32_t sin6_scope_id;
};
```

Members

Members	Description
short sin6_family;	Address family; must be AF_INET.
uint16_t sin6_port;	Internet Protocol (IP) port.
uint32_t sin6_flowinfo;	IPv6 flow information
struct in6_addr sin6_addr;	IPv6 address in network byte order.
uint32_t sin6_scope_id;	IPv6 Scope Id

Description

In the Internet address family

sockaddr_storage Structure

File

berkeley_api.h

C

```
struct sockaddr_storage {
  short ss_family;
  __ss_aligntype __ss_align;
  char __ss_padding[_SS_PADSIZE];
};
```

Members

Members	Description
short ss_family;	Address family
__ss_aligntype __ss_align;	Force desired alignment.

Description

This is record sockaddr_storage.

SOCKADDR_IN6 Type

File

berkeley_api.h

C

```
typedef struct sockaddr_in6 SOCKADDR_IN6;
```

Description

This is type SOCKADDR_IN6.

AF_INET6 Macro

File

berkeley_api.h

C

```
#define AF_INET6 10          // Internet Address Family - IPv6
```

Description

Internet Address Family - IPv6

__ss_aligntype Macro

File

berkeley_api.h

C

```
#define __ss_aligntype __uint32_t
```

Description

Structure large enough to hold any socket address (with the historical exception of AF_UNIX). 128 bytes reserved.

_BERKELEY_API_HEADER_FILE Macro

File

berkeley_api.h

C

```
#define _BERKELEY_API_HEADER_FILE
```

Description

This is macro _BERKELEY_API_HEADER_FILE.

_SS_PADSIZE Macro

File

berkeley_api.h

C

```
#define _SS_PADSIZE (_SS_SIZE - (2 * sizeof (__ss_aligntype)))
```

Description

This is macro _SS_PADSIZE.

_SS_SIZE Macro

File

berkeley_api.h

C

```
#define _SS_SIZE 128
```

Description

This is macro _SS_SIZE.

BERKELEY_MODULE_CONFIG Structure

File

berkeley_api.h

C

```
typedef struct {
  uint8_t maxSockets;
} BERKELEY_MODULE_CONFIG;
```

Members

Members	Description
uint8_t maxSockets;	Maximum number of sockets supported

Description

Berkeley API module configuration structure

Files

Files

Name	Description
berkeley_api.h	The Berkeley Socket Distribution (BSD) APIs provide a BSD wrapper to the native Microchip TCP/IP Stack APIs.
berkeley_api_config.h	Berkeley Sockets API Client and Server

Description

This section lists the source and header files used by the library.

berkeley_api.h

The Berkeley Socket Distribution (BSD) APIs provide a BSD wrapper to the native Microchip TCP/IP Stack APIs.

Functions

	Name	Description
	accept	This function accepts connection requests queued for a listening socket.

	bind	This function assigns a name to the socket descriptor.
	closesocket	The closesocket function closes an existing socket.
	connect	This function connects to the peer communications end point.
	freeaddrinfo	Frees the memory allocated by getaddrinfo
	getaddrinfo	Does an address look up for the provided node name.
	gethostbyname	The gethostbyname function returns a structure of type hostent for the given host name.
	gethostname	Returns the standard host name for the system.
	getsockname	Returns the current address to which the socket is bound.
	getsockopt	Allows setting options to a socket like adjust RX/TX buffer size, etc
	listen	The listen function sets the specified socket in a listen mode.
	recv	The recv() function is used to receive incoming data that has been queued for a socket.
	recvfrom	The recvfrom() function is used to receive incoming data that has been queued for a socket.
	send	The send function is used to send outgoing data on an already connected socket.
	sendto	This function used to send the data for both connection oriented and connection-less sockets.
	setsockopt	Allows setting options to a socket like adjust RX/TX buffer size, etc
	socket	This function creates a new Berkeley socket.
	TCPIP_BSD_Socket	Returns the native socket number associated with the BSD socket.

Macros

	Name	Description
	__ss_aligntype	Structure large enough to hold any socket address (with the historical exception of AF_UNIX). 128 bytes reserved.
	_BERKELEY_API_HEADER_FILE	This is macro _BERKELEY_API_HEADER_FILE.
	_SS_PADSIZE	This is macro _SS_PADSIZE.
	_SS_SIZE	This is macro _SS_SIZE.
	AF_INET	Internet Address Family - IPv4, UDP, TCP, etc.
	AF_INET6	Internet Address Family - IPv6
	EAI_AGAIN	Temporary failure in name resolution.
	EAI_BADFLAGS	Invalid value for `ai_flags' field.
	EAI_FAIL	Non-recoverable failure in name res.
	EAI_FAMILY	ai_family' not supported.
	EAI_MEMORY	Memory allocation failure.
	EAI_NONAME	NAME or SERVICE is unknown.
	EAI_OVERFLOW	Argument buffer overflow.
	EAI_SERVICE	SERVICE not supported for `ai_socktype'.
	EAI_SOCKTYPE	ai_socktype' not supported.
	EAI_SYSTEM	System error returned in `errno'.
	HOST_NOT_FOUND	Authoritative Answer host not found
	ICMP6_FILTER	SEt the IPv6 Filtering options - Not yet supported
	INADDR_ANY	IP address for server binding.
	INVALID_TCP_PORT	Invalid TCP port
	IP_ADD_MEMBERSHIP	Join a multicast group - Not yet supported
	IP_ADDR_ANY	IP Address for server binding
	IP_DROP_MEMBERSHIP	Leave a multicast group - Not yet supported
	IP_MULTICAST_IF	Set the Interface that multicast packets should be sent on - Not yet supported
	IP_MULTICAST_LOOP	Specify a copy of multicast packets should be delivered to the sending host - Not yet supported
	IP_MULTICAST_TTL	Set the Time to Live option for multicast packets - Not yet supported
	IP_OPTIONS	IP Header Options - Not yet supported
	IP_TOS	Type of Service - Not yet supported
	IP_TTL	Time to Live - Not yet supported
	IPPROTO_ICMPV6	Indicates IPv6 ICMP Protocol Options
	IPPROTO_IP	Indicates IP pseudo-protocol.
	IPPROTO_IPV6	Indicates IP v6 Header options
	IPPROTO_TCP	Indicates TCP level options

	IPPROTO_UDP	Indicates UDP level options
	IPV6_CHECKSUM	Sets the IPv6 Checksum options - Not yet supported
	IPV6_JOIN_GROUP	Join an IPv6 multicast group - Not yet supported
	IPV6_LEAVE_GROUP	Leave an IPv6 multicast group - Not yet supported
	IPV6_MULTICAST_HOPS	Set the hop limit for multicast IPv6 packets - Not yet supported
	IPV6_MULTICAST_IF	Set the Interface that multicast IPv6 packets should be sent on - Not yet supported
	IPV6_MULTICAST_LOOP	Specify a copy of multicast IPv6 packets should be delivered to the sending host - Not yet supported
	IPV6_UNICAST_HOPS	Set the hop limit for unicast IPv6 packets - Not yet supported
	IPV6_V6ONLY	Sets the socket to IPv6 only - Not yet supported
	NO_DATA	Valid name, no data record of requested type
	NO_RECOVERY	Non recoverable errors,
	SO_BROADCAST	Enables the Socket for sending broadcast data
	SO_DEBUG	Indicates if low level debut is active - Not yet supported
	SO_DONTROUTE	Bypass normal routing - Not yet supported
	SO_KEEPALIVE	Keep the connection alive by sending periodic transmissions - Not yet supported
	SO_LINGER	Indicates if the system should send any buffered data when a socket is closed
	SO_OOBINLINE	Indicates whether or not Out of Band Data should be received inline with normal data - Not yet supported
	SO_RCVBUF	Receive Buffer Size (TCP only)
	SO_RCVLOWAT	Receive Low Water mark - Not yet supported
	SO_RCVTIMEO	Set the Receive Timeout - Not yet supported
	SO_REUSEADDR	Indicates if the local socket can be reused immediately after close - Not yet supported
	SO_SNDBUF	Send Buffer Size
	SO_SNDLOWAT	Send Low Water mark - Not yet supported
	SO_SNDTIMEO	Set the Send Timeout - Not yet supported
	SOCK_DGRAM	Connectionless datagram socket. Use UDP for the Internet address family.
	SOCK_STREAM	Connection based byte streams. Use TCP for the Internet address family.
	SOCKET_CNXN_IN_PROGRESS	Socket connection state.
	SOCKET_DISCONNECTED	Socket disconnected
	SOCKET_ERROR	Socket error
	SOL_SOCKET	Indicates socket level options
	TCP_NODELAY	Indicates if TCP is to buffer packets - Not yet supported
	TRY_AGAIN	Non authoritative host not found or server fail

Structures

	Name	Description
	addrinfo	This is record addrinfo.
	hostent	This is record hostent.
	in_addr	in_addr structure
	in6_addr	This is record in6_addr.
	linger	This is record linger.
	sockaddr	generic address structure for all address families
	sockaddr_in	In the Internet address family
	sockaddr_in6	In the Internet address family
	sockaddr_storage	This is record sockaddr_storage.
	BERKELEY_MODULE_CONFIG	Berkeley API module configuration structure

Types

	Name	Description
	SOCKADDR	generic address structure for all address families
	SOCKADDR_IN	In the Internet address family
	SOCKADDR_IN6	This is type SOCKADDR_IN6.
	SOCKET	Socket descriptor

Description

Berkeley Socket Distribution API Header File

The Berkeley Socket Distribution (BSD) APIs provide a BSD wrapper to the native Microchip TCP/IP Stack APIs. Using this interface, programmers familiar with BSD sockets can quickly develop applications using Microchip's TCP/IP Stack.

File Name

berkeley_api.h

Company

Microchip Technology Inc.

berkeley_api_config.h

Berkeley Sockets API Client and Server

Macros

	Name	Description
	MAX_BSD_SOCKETS	Berkeley API max number of sockets simultaneous supported

Description

Berkeley Sockets API Configuration file

This file contains the Berkeley API module configuration options

File Name

berkeley_api_config.h

Company

Microchip Technology Inc.

DHCP Module

This section describes the TCP/IP Stack Library DHCP module.

Introduction

TCP/IP Stack Library Dynamic Host Configuration Protocol (DHCP) Module for Microchip Microcontrollers

This library provides the API of the DHCP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The DHCP client module will allow your application to dynamically obtain an IP address and a subnet mask from a DHCP server on the same network. Additionally, the DHCP client will get other parameters, such as gateway and DNS addresses.

Using the Library

This topic describes the basic architecture of the DHCP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `dhcp.h`

The interface to the DHCP TCP/IP Stack library is defined in the `dhcp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the DHCP TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

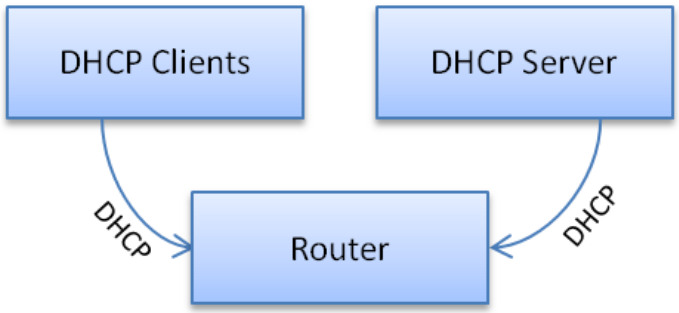
This library provides the API of the DHCP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

This module provides software abstraction of the DHCP module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.

DHCP Software Abstraction Block Diagram

Typical DCHP Usage



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DHCP module.

Library Interface Section	Description
Configuration Functions	Routines for Configuring DHCP
Status Functions	Routines for Obtaining DHCP Status Information
Data Types and Constants	This section provides various definitions describing this API

How the Library Works

The Dynamic Host Configuration protocol (DHCP) is a standard networking protocol used to automatically allocate IP addresses for hosts in a network.

The DHCP server has a pool of IP addresses, which are leased for clients requesting them. The leases have a time-out after which the hosts need to renew the lease or acquire a new one.

The DHCP client module in the TCP/IP stack takes care of the communication with the DHCP server and renewing the lease when the time-out expires.

Configuring the Library

Macros

	Name	Description
	TCPIP_DHCP_BOOT_FILE_NAME_SIZE	size of the storage for the Boot file name should always be <= 128 default value is 128
	TCPIP_DHCP_STORE_BOOT_FILE_NAME	enable the usage of the Boot file name received from the DHCP server

Description

The configuration of the DHCP TCP/IP Stack is based on the file `system_config.h` (which may include a `dhcp_config.h` file).

This header file contains the configuration parameters for the DHCP TCP/IP Stack. Based on the selections made, the DHCP TCP/IP Stack will adjust its behavior.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_DHCP_BOOT_FILE_NAME_SIZE Macro

File

dhcp_config.h

C

```
#define TCPIP_DHCP_BOOT_FILE_NAME_SIZE 128
```

Description

size of the storage for the Boot file name should always be <= 128 default value is 128

TCPIP_DHCP_STORE_BOOT_FILE_NAME Macro

File

dhcp_config.h

C

```
#define TCPIP_DHCP_STORE_BOOT_FILE_NAME
```

Description

enable the usage of the Boot file name received from the DHCP server

Building the Library

This section lists the files that are available in the DHCP module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dhcp.c`	DHCP implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The DHCP module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Library Interface

a) Configuration Functions

	Name	Description
	TCPIP_DHCP_Disable	Disables the DHCP Client for the specified interface.

	TCPIP_DHCP_Enable	Enables the DHCP client for the specified interface.
	TCPIP_DHCP_HandlerDeRegister	Deregisters a previously registered DHCP handler.
	TCPIP_DHCP_HandlerRegister	Registers a DHCP Handler.
	TCPIP_DHCP_HostNameCallbackRegister	Registers a DHCP host name callback with the DHCP client.
	TCPIP_DHCP_Renew	Renews the DHCP lease for the specified interface.
	TCPIP_DHCP_Request	Requests the supplied IPv4 address from a DHCP server.

b) Status Functions

	Name	Description
	TCPIP_DHCP_IsBound	Determines if the DHCP client has an IP address lease on the specified interface.
	TCPIP_DHCP_IsEnabled	Determines if the DHCP client is enabled on the specified interface.
	TCPIP_DHCP_IsServerDetected	Determines if the DHCP client on the specified interface has been able to contact a DHCP server.
	TCPIP_DHCP_IsActive	Determines if the DHCP client is currently active on the specified interface.

c) Data Types and Constants

	Name	Description
	TCPIP_DHCP_EVENT_HANDLER	DHCP event handler prototype.
	TCPIP_DHCP_EVENT_TYPE	DHCP Event Type
	TCPIP_DHCP_HANDLE	DHCP handle.
	TCPIP_DHCP_MODULE_CONFIG	DHCP Module Configuration run-time parameters.
	TCPIP_DHCP_INFO	Reports DHCP module information.
	TCPIP_DHCP_STATUS	Lists the current status of the DHCP module.
	TCPIP_DHCP_HOST_NAME_CALLBACK	DHCP Host name callback function.
	TCPIP_DHCP_HOST_NAME_SIZE	Maximum size of a host name to be advertised to the DHCP server

Description

This section describes the Application Programming Interface (API) functions of the DHCP module.

Refer to each section for a detailed description.

a) Configuration Functions

TCPIP_DHCP_Disable Function

Disables the DHCP Client for the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_Disable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if successful
- false - if unsuccessful

Description

This function disables the DHCP client for the specified interface. If it is already disabled, no action is taken.

Remarks

When the DHCP client is disabled and the interface continues using its old configuration, it is possible that the lease may expire and the DHCP server provide the IP address to another client. The application should not keep the old lease unless it is sure that there is no danger of conflict.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
pNetIf	Interface to disable the DHCP client on.

Function

bool TCPIP_DHCP_Disable(TCPIP_NET_HANDLE hNet)

TCPIP_DHCP_Enable Function

Enables the DHCP client for the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_Enable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if successful
- false - if unsuccessful

Description

This function enables the DHCP client for the specified interface, if it is disabled. If it is already enabled, no action is taken.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface to enable the DHCP client on.

Function

void TCPIP_DHCP_Enable(TCPIP_NET_HANDLE hNet)

TCPIP_DHCP_HandlerDeRegister Function

Deregisters a previously registered DHCP handler.

File

dhcp.h

C

```
bool TCPIP_DHCP_HandlerDeRegister(TCPIP_DHCP_HANDLE hDhcp);
```

Returns

- true - if the call succeeds
- false - if no such handler is registered

Description

This function deregisters the DHCP event handler.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hDhcp	A handle returned by a previous call to TCPIP_DHCP_HandlerRegister.

Function

TCPIP_DHCP_HandlerDeRegister(TCPIP_DHCP_HANDLE hDhcp)

TCPIP_DHCP_HandlerRegister Function

Registers a DHCP Handler.

File

dhcp.h

C

```
TCPIP_DHCP_HANDLE TCPIP_DHCP_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_EVENT_HANDLER handler, const void* hParam);
```

Returns

Returns a valid handle if the call succeeds, or a null handle if the call failed (out of memory, for example).

Description

This function registers a DHCP event handler. The DHCP module will call the registered handler when a DHCP event (TCPIP_DHCP_EVENT_TYPE) occurs.

Remarks

The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

The hParam is passed by the client and will be used by the DHCP when the notification is made. It is used for per-thread content or if more modules, for example, share the same handler and need a way to differentiate the callback.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface handle. Use hNet == 0 to register on all interfaces available.
handler	Handler to be called when a DHCP event occurs.
hParam	Parameter to be used in the handler call. This is user supplied and is not used by the DHCP module.

Function

TCPIP_DHCP_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_EVENT_HANDLER handler,

const void* hParam)

TCPIP_DHCP_HostNameCallbackRegister Function

Registers a DHCP host name callback with the DHCP client.

File

dhcp.h

C

```
bool TCPIP_DHCP_HostNameCallbackRegister(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_HOST_NAME_CALLBACK nameCallback, bool writeBack);
```

Returns

- true - if the call succeeds
- false - if error (no such interface, etc.)

Description

This function registers a DHCP host name callback function. The DHCP module will call the registered callback when a host name needs to be presented to the DHCP server.

Remarks

The nameCallback function has to be valid for the length of DHCP client communication with the server, i.e. when the reported status is bound or some error.

There is no corresponding deregister function. Simply call TCPIP_DHCP_HostNameCallbackRegister with the nameCallback set to 0.

If such a callback is not registered, then the NetBios name will be used for the DHCP host name.

The callback function has to return a character string that's compatible with the rules imposed for the host names:

- Host names may contain only alphanumeric characters, minus signs ("-"), and periods (".")
- They must begin with an alphabetic character and end with an alphanumeric character

To enforce these rules the extra processing is performed internally (on either user supplied host name or NetBios name):

- spaces, if present, will be stripped from the string
- illegal characters will be replaced by TCPIP_DHCP_HOST_REPLACE_CHAR character (default is lower case 'x')

If after processing the resulting string is null, the DHCP host name option is skipped.

The callback function has to return a character string that's at least 2 characters in size;

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface handle. Use hNet == 0 to register on all interfaces available.
nameCallback	Callback to be called to get a DHCP host name
writeBack	if true, the resulting name will be written to the address returned by the nameCallback (it shouldn't be const!)

Function

bool TCPIP_DHCP_HostNameCallbackRegister(TCPIP_NET_HANDLE hNet,
TCPIP_DHCP_HOST_NAME_CALLBACK nameCallback, bool writeBack)

TCPIP_DHCP_Renew Function

Renews the DHCP lease for the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_Renew(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if successful
- false - if unsuccessful

Description

This function attempts to contact the server and renew the DHCP lease for the specified interface. The interface should have the DHCP enabled and in bound state for this call to succeed.

Preconditions

The DHCP module must be initialized and enabled, and have a valid lease.

Parameters

Parameters	Description
hNet	Interface on which to renew the DHCP lease.

Function

void TCPIP_DHCP_Renew(TCPIP_NET_HANDLE hNet)

TCPIP_DHCP_Request Function

Requests the supplied IPv4 address from a DHCP server.

File

dhcp.h

C

```
bool TCPIP_DHCP_Request(TCPIP_NET_HANDLE hNet, IPV4_ADDR reqAddress);
```

Returns

- true - if successful
- false - if the supplied IP address is invalid or the DHCP client is in the middle of a transaction

Description

If the DHCP client is not enabled on that interface, this call will first try to enable it. If this succeeds or the DHCP client was already enabled, the following steps are taken: The DHCP client probes the DHCP server and requests the supplied IPv4 address as a valid lease for the specified interface. If the server acknowledges the request, then this is the new IPv4 address of the interface. If the DHCP server rejects the request, then the whole DHCP process is resumed starting with the DHCP Discovery phase.

Remarks

The requested IPv4 address should be a previous lease that was granted to the host. This call should be used when the host is restarting.

Preconditions

The DHCP module must be initialized and enabled, and have a valid lease.

Parameters

Parameters	Description
hNet	Interface to renew the DHCP lease on.

Function

void TCPIP_DHCP_Request(TCPIP_NET_HANDLE hNet, IPV4_ADDR reqAddress)

b) Status Functions

TCPIP_DHCP_IsBound Function

Determines if the DHCP client has an IP address lease on the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_IsBound(TCPIP_NET_HANDLE hNet);
```

Returns

- true - DHCP client has obtained an IP address lease (and likely other parameters) and these values are currently being used
- false - No IP address is currently leased

Description

This functin returns the status of the current IP address lease on the specified interface.

Preconditions

None.

Parameters

Parameters	Description
hNet	Interface to query

Function

bool TCPIP_DHCP_IsBound(TCPIP_NET_HANDLE hNet)

TCPIP_DHCP_IsEnabled Function

Determines if the DHCP client is enabled on the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_IsEnabled(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if the DHCP client service is enabled on the specified interface
- false - if the DHCP client service is not enabled on the specified interface

Description

This function returns the current state of the DHCP client on the specified interface.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface to query.

Function

bool TCPIP_DHCP_IsEnabled(TCPIP_NET_HANDLE hNet)

TCPIP_DHCP_IsServerDetected Function

Determines if the DHCP client on the specified interface has been able to contact a DHCP server.

File

dhcp.h

C

```
bool TCPIP_DHCP_IsServerDetected(TCPIP_NET_HANDLE hNet);
```

Returns

- true - At least one DHCP server is attached to the specified network interface
- false - No DHCP servers are currently detected on the specified network interface

Description

This function determines if the DHCP client on the specified interface received any reply from a DHCP server.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface to query.

Function

bool TCPIP_DHCP_IsServerDetected(TCPIP_NET_HANDLE hNet)

TCPIP_DHCP_IsActive Function

Determines if the DHCP client is currently active on the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_IsActive(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if the DHCP client service is currently active on the specified interface
- false - if the DHCP client service is not active on the specified interface

Description

This function returns the current state of the DHCP client on the specified interface.

Remarks

The DHCP client service could be enabled bot not active. For example when there was no DHCP server detected on the network.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface to query.

Function

bool TCPIP_DHCP_IsActive(TCPIP_NET_HANDLE hNet)

c) Data Types and Constants

TCPIP_DHCP_EVENT_HANDLER Type

DHCP event handler prototype.

File

dhcp.h

C

```
typedef void (* TCPIP_DHCP_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_EVENT_TYPE evType, const void*
param);
```

Description

Type: TCPIP_DHCP_EVENT_HANDLER

Prototype of a DHCP event handler. Clients can register a handler with the DHCP service. Once an DHCP event occurs the DHCP service will called the registered handler. The handler has to be short and fast. It is meant for setting an event flag, *not* for lengthy processing!

TCPIP_DHCP_EVENT_TYPE Enumeration

DHCP Event Type

File

dhcp.h

C

```
typedef enum {
  DHCP_EVENT_NONE = 0,
  DHCP_EVENT_DISCOVER,
  DHCP_EVENT_REQUEST,
  DHCP_EVENT_ACK,
  DHCP_EVENT_ACK_INVALID,
  DHCP_EVENT_DECLINE,
  DHCP_EVENT_NACK,
  DHCP_EVENT_TIMEOUT,
  DHCP_EVENT_BOUND,
  DHCP_EVENT_REQUEST_RENEW,
  DHCP_EVENT_REQUEST_REBIND,
  DHCP_EVENT_CONN_LOST,
  DHCP_EVENT_CONN_ESTABLISHED,
  DHCP_EVENT_SERVICE_DISABLED
} TCPIP_DHCP_EVENT_TYPE;
```

Members

Members	Description
DHCP_EVENT_NONE = 0	DHCP no event

DHCP_EVENT_DISCOVER	DHCP discovery sent: cycle started
DHCP_EVENT_REQUEST	DHCP request sent
DHCP_EVENT_ACK	DHCP request acknowledge was received
DHCP_EVENT_ACK_INVALID	DHCP acknowledge received but discarded as invalid
DHCP_EVENT_DECLINE	DHCP lease declined
DHCP_EVENT_NACK	DHCP negative acknowledge was received
DHCP_EVENT_TIMEOUT	DHCP server timeout
DHCP_EVENT_BOUND	DHCP lease obtained
DHCP_EVENT_REQUEST_RENEW	lease request renew sent
DHCP_EVENT_REQUEST_REBIND	lease request rebind sent
DHCP_EVENT_CONN_LOST	connection to the DHCP server lost
DHCP_EVENT_CONN_ESTABLISHED	connection re-established
DHCP_EVENT_SERVICE_DISABLED	DHCP service disabled, reverted to the default IP address

Description

Enumeration: TCPIP_DHCP_EVENT_TYPE

None.

TCPIP_DHCP_HANDLE Type

DHCP handle.

File

dhcp.h

C

```
typedef const void* TCPIP_DHCP_HANDLE;
```

Description

Type: TCPIP_DHCP_HANDLE

A handle that a client can use after the event handler has been registered.

TCPIP_DHCP_MODULE_CONFIG Structure

DHCP Module Configuration run-time parameters.

File

dhcp.h

C

```
typedef struct {
  bool dhcpEnable;
  int dhcpTmo;
  int dhcpCliPort;
  int dhcpSrvPort;
} TCPIP_DHCP_MODULE_CONFIG;
```

Members

Members	Description
bool dhcpEnable;	DHCP client enable at module start-up
int dhcpTmo;	timeout to wait for DHCP lease, seconds
int dhcpCliPort;	client port for DHCP client transactions
int dhcpSrvPort;	remote server port for DHCP server messages

Description

DHCP Module Configuration

This structure contains the data that's passed to the DHCP module at the TCP/IP stack initialization.

TCPIP_DHCP_INFO Structure

Reports DHCP module information.

File

dhcp.h

C

```
typedef struct {
  TCPIP_DHCP_STATUS status;
  uint32_t dhcpTime;
  uint32_t leaseStartTime;
  uint32_t leaseDuration;
  uint32_t renewTime;
  uint32_t rebindTime;
  IPV4_ADDR dhcpAddress;
  IPV4_ADDR serverAddress;
  const char* bootFileName;
} TCPIP_DHCP_INFO;
```

Members

Members	Description
TCPIP_DHCP_STATUS status;	current status
uint32_t dhcpTime;	current DHCP time, seconds the following fields are significant only if a lease has been obtained and is currently valid i.e. status >= TCPIP_DHCP_BOUND
uint32_t leaseStartTime;	time when lease was requested
uint32_t leaseDuration;	lease duration as returned by the server, seconds
uint32_t renewTime;	the time for moving to renew state, seconds
uint32_t rebindTime;	the time for moving to rebind state, seconds
IPV4_ADDR dhcpAddress;	IPv4 address obtained by DHCP
IPV4_ADDR serverAddress;	IPv4 address of the server that granted the lease
const char* bootFileName;	pointer to the bootfile name that was returned by the server This will be 0 if TCPIP_DHCP_STORE_BOOT_FILE_NAME option is not enabled!

Description

Structure: TCPIP_DHCP_INFO

This data structure is used for reporting current info and status of the DHCP module. Used in getting info about the DHCP module.

TCPIP_DHCP_STATUS Enumeration

Lists the current status of the DHCP module.

File

dhcp.h

C

```
typedef enum {
  TCPIP_DHCP_IDLE = 0,
  TCPIP_DHCP_WAIT_LINK,
  TCPIP_DHCP_GET_SOCKET,
  TCPIP_DHCP_SEND_DISCOVERY,
  TCPIP_DHCP_GET_OFFER,
  TCPIP_DHCP_SEND_REQUEST,
  TCPIP_DHCP_GET_REQUEST_ACK,
  TCPIP_DHCP_WAIT_LEASE_CHECK,
  TCPIP_DHCP_WAIT_LEASE_RETRY,
  TCPIP_DHCP_BOUND,
  TCPIP_DHCP_BOUND_RENEW,
  TCPIP_DHCP_SEND_RENEW,
  TCPIP_DHCP_GET_RENEW_ACK,
  TCPIP_DHCP_SEND_REBIND,
  TCPIP_DHCP_GET_REBIND_ACK
} TCPIP_DHCP_STATUS;
```

Members

Members	Description
TCPIP_DHCP_IDLE = 0	idle/inactive state
TCPIP_DHCP_WAIT_LINK	waiting for an active connection
TCPIP_DHCP_GET_SOCKET	trying to obtain a socket
TCPIP_DHCP_SEND_DISCOVERY	sending a Discover message
TCPIP_DHCP_GET_OFFER	waiting for a DHCP Offer
TCPIP_DHCP_SEND_REQUEST	sending a REQUEST message (REQUESTING)
TCPIP_DHCP_GET_REQUEST_ACK	waiting for a Request ACK message
TCPIP_DHCP_WAIT_LEASE_CHECK	waiting for received lease verification
TCPIP_DHCP_WAIT_LEASE_RETRY	waiting for another attempt after the lease verification failed
TCPIP_DHCP_BOUND	bound
TCPIP_DHCP_BOUND_RENEW	a lease renew is initiated
TCPIP_DHCP_SEND_RENEW	sending a REQUEST message (RENEW state)
TCPIP_DHCP_GET_RENEW_ACK	waiting for ACK in RENEW state
TCPIP_DHCP_SEND_REBIND	sending REQUEST message (REBIND state)
TCPIP_DHCP_GET_REBIND_ACK	waiting for ACK in REBIND state

Description

Enumeration: TCPIP_DHCP_STATUS

This enumeration lists the current status of the DHCP module. Used in getting information about the DHCP state machine.

TCPIP_DHCP_HOST_NAME_CALLBACK Type

DHCP Host name callback function.

File

dhcp.h

C

```
typedef char* (* TCPIP_DHCP_HOST_NAME_CALLBACK)(TCPIP_NET_HANDLE hNet);
```

Description

Type: TCPIP_DHCP_HOST_NAME_CALLBACK

Prototype of a DHCP callback function that returns the host name to be presented to the server by the DHCP client.

This callback will be called by the DHCP client when communicating to the server and a host name is needed.

Remarks

There are certain restictions that apply to the host name strings. See the TCPIP_DHCP_HostNameCallbackRegister function.

TCPIP_DHCP_HOST_NAME_SIZE Macro

File

dhcp_config.h

C

```
#define TCPIP_DHCP_HOST_NAME_SIZE 20
```

Description

Maximum size of a host name to be advertised to the DHCP server

Files

Files

Name	Description
dhcp.h	Dynamic Host Configuration Protocol (DHCP) client API definitions.

dhcp_config.h	DCHP configuration file

Description

This section lists the source and header files used by the library.

dhcp.h

Dynamic Host Configuration Protocol (DHCP) client API definitions.

Enumerations

	Name	Description
	TCPIP_DHCP_EVENT_TYPE	DHCP Event Type
	TCPIP_DHCP_STATUS	Lists the current status of the DHCP module.

Functions

	Name	Description
	TCPIP_DHCP_Disable	Disables the DHCP Client for the specified interface.
	TCPIP_DHCP_Enable	Enables the DHCP client for the specified interface.
	TCPIP_DHCP_HandlerDeRegister	Deregisters a previously registered DHCP handler.
	TCPIP_DHCP_HandlerRegister	Registers a DHCP Handler.
	TCPIP_DHCP_HostNameCallbackRegister	Registers a DHCP host name callback with the DHCP client.
	TCPIP_DHCP_InfoGet	Returns information about the DHCP client on the specified interface.
	TCPIP_DHCP_IsActive	Determines if the DHCP client is currently active on the specified interface.
	TCPIP_DHCP_IsBound	Determines if the DHCP client has an IP address lease on the specified interface.
	TCPIP_DHCP_IsEnabled	Determines if the DHCP client is enabled on the specified interface.
	TCPIP_DHCP_IsServerDetected	Determines if the DHCP client on the specified interface has been able to contact a DHCP server.
	TCPIP_DHCP_Renew	Renews the DHCP lease for the specified interface.
	TCPIP_DHCP_Request	Requests the supplied IPv4 address from a DHCP server.
	TCPIP_DHCP_RequestTimeoutSet	Sets the DHCP client request and base time-out values.
	TCPIP_DHCP_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_DHCP_INFO	Reports DHCP module information.
	TCPIP_DHCP_MODULE_CONFIG	DHCP Module Configuration run-time parameters.

Types

	Name	Description
	TCPIP_DHCP_EVENT_HANDLER	DHCP event handler prototype.
	TCPIP_DHCP_HANDLE	DHCP handle.
	TCPIP_DHCP_HOST_NAME_CALLBACK	DHCP Host name callback function.

Description

Dynamic Host Configuration Protocol (DHCP) Client API Header File

Provides automatic IP address, subnet mask, gateway address, DNS server address, and other configuration parameters on DHCP enabled networks.

File Name

dhcp.h

Company

Microchip Technology Inc.

dhcp_config.h

DCHP configuration file

Macros

	Name	Description
	TCPIP_DHCP_BOOT_FILE_NAME_SIZE	size of the storage for the Boot file name should always be <= 128 default value is 128
	TCPIP_DHCP_CLIENT_ENABLED	Default value for the enable/disable the DHCP client at stack start-up.
	TCPIP_DHCP_HOST_NAME_SIZE	Maximum size of a host name to be advertised to the DHCP server
	TCPIP_DHCP_STORE_BOOT_FILE_NAME	enable the usage of the Boot file name received from the DHCP server
	TCPIP_DHCP_TASK_TICK_RATE	The DHCP task processing rate: number of milliseconds to generate an DHCP tick. Used by the DHCP state machine The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DHCP_TIMEOUT	Defines how long to wait before a DHCP lease is acquired when the DHCP module is enabled, seconds

Description

Dynamic Host Configuration Protocol (DCHP) Configuration file

This file contains the DCHP module configuration options

File Name

dhcp_config.h

Company

Microchip Technology Inc.

DHCP Server Module

This section describes the TCP/IP Stack Library DHCP Server module.

Introduction

TCP/IP Stack Library DHCP Server Module for Microchip Microcontrollers

This library provides the API of the DHCP Server module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The DHCP Server module is used to assign the IP address to the DHCP client from the configured IP address database. When the server receives a request from a client, the DHCP server determines the network to which the DHCP client is connected, and then allocates an IP address that is appropriate for the client, and sends configuration information appropriate for that client. DHCP servers typically grant IP addresses to clients only for a limited interval. DHCP clients are responsible for renewing their IP address before that interval has expired, and must stop using the address once the interval has expired, if they have not been able to renew it.

Using the Library

This topic describes the basic architecture of the DHCP Server TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `dhcps.h`

The interface to the DHCP Server TCP/IP Stack library is defined in the `dhcps.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the DHCP Server TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the DHCP Server TCP/IP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

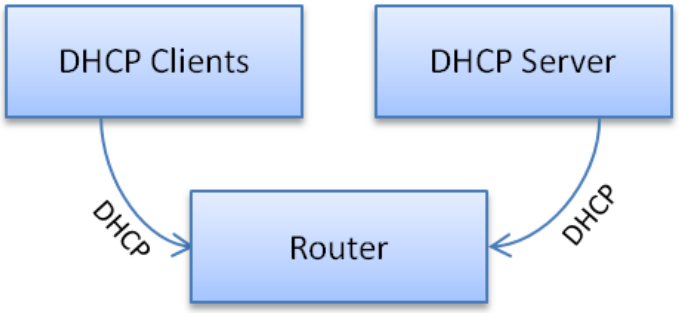
Description

DHCP Server Software Abstraction Block Diagram

This module provides software abstraction of the DHCP Server module existent in any TCP/IP Stack implementation. It removes the overhead of

address resolution from all other modules in the stack.

Typical DCHP Usage



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DHCP Server module.

Library Interface Section	Description
Configuration Functions	Routines for Configuring DHCP server
Status Functions	Routines for Obtaining DHCP Status Information
Data Types and Constants	This section provides various definitions describing this API

How the Library Works

To use DHCP Server, include the files `dhcps.c`, `dhcp.c`, and `dhcp.h` in your project, and add or uncomment the definition "#define TCPIP_STACK_USE_DHCP_SERVER" to `tcpip_config.h`.

Configuring the Library

Macros

	Name	Description
	TCPIP_DHCPS_DEFAULT_IP_ADDRESS_RANGE_START	These below IPv4 DHCP server address details are default address and it is assigned to the network default network interface. for Other interfaces , tcpip_stack_init.c file should be use to configure DHCP_POOL_CONFIG[]. IPv4 Address range is starting from 100, because the from 1 to 100 is reserved. Reserved Address will be used for the gateway address. Start of IP address Range , network_config.h ipaddress and this start of IP address should be in same SUBNET RECOMENDED - network_config.h ipaddress should be 192.168.1.1 if DHCP server ip address range starts from 192.168.1.100.
	TCPIP_DHCPS_DEFAULT_SERVER_IP_ADDRESS	DHCP server Address per interface. DHCP server Address selection should be in the same subnet.
	TCPIP_DHCPS_DEFAULT_SERVER_NETMASK_ADDRESS	DHCP server subnet Address per interface.
	TCPIP_DHCPS_DEFAULT_SERVER_PRIMARY_DNS_ADDRESS	DHCP server DNS primary Address
	TCPIP_DHCPS_DEFAULT_SERVER_SECONDARY_DNS_ADDRESS	DHCP server DNS Secondary Address
	TCPIP_DHCPS_LEASE_DURATION	Timeout for a solved entry in the cache, in seconds the entry will be removed if the TMO elapsed and the entry has not been referenced again
	TCPIP_DHCPS_LEASE_ENTRIES_DEFAULT	The Maximum Number of entries in the lease table Default total number of entries for all the the interface
	TCPIP_DHCPS_LEASE_REMOVED_BEFORE_ACK	Timeout for a unsolved entry , in seconds and should be removed from the entry if there is no REQUEST after OFFER

	TCPIP_DHCPS_LEASE_SOLVED_ENTRY_TMO	Timeout for a solved entry in the cache, in seconds. The entry will be removed if the TMO lapsed and the entry has not been referenced again
	TCPIP_DHCPS_TASK_PROCESS_RATE	DHCPS task processing rate, in milliseconds. The DHCPS module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other TMO are multiple of this The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

The configuration of the DHCP Server TCP/IP Stack is based on the file `dhcps_config.h`.

This header file contains the configuration selection for the DHCP Server TCP/IP Stack. Based on the selections made, the DHCP Server TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the DHCP Server TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_DHCPS_DEFAULT_IP_ADDRESS_RANGE_START Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_DEFAULT_IP_ADDRESS_RANGE_START "192.168.1.100"
```

Description

These below IPv4 DHCP server address details are default address and it is assigned to the network default network interface. for Other interfaces , tcpip_stack_init.c file should be use to configure DHCP_POOL_CONFIG[]. IPv4 Address range is starting from 100, because the from 1 to 100 is reserved. Reserved Address will be used for the gateway address. Start of IP address Range , network_config.h ipaddress and this start of IP address should be in same SUBNET RECOMENDED - network_config.h ipaddress should be 192.168.1.1 if DHCP server ip address range starts from 192.168.1.100.

TCPIP_DHCPS_DEFAULT_SERVER_IP_ADDRESS Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_DEFAULT_SERVER_IP_ADDRESS "192.168.1.1"
```

Description

DHCP server Address per interface. DHCP server Address selection should be in the same subnet.

TCPIP_DHCPS_DEFAULT_SERVER_NETMASK_ADDRESS Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_DEFAULT_SERVER_NETMASK_ADDRESS "255.255.255.0"
```

Description

DHCP server subnet Address per interface.

TCPIP_DHCPS_DEFAULT_SERVER_PRIMARY_DNS_ADDRESS Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_DEFAULT_SERVER_PRIMARY_DNS_ADDRESS "192.168.1.1"
```

Description

DHCP server DNS primary Address

TCPIP_DHCPS_DEFAULT_SERVER_SECONDARY_DNS_ADDRESS Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_DEFAULT_SERVER_SECONDARY_DNS_ADDRESS "192.168.1.1"
```

Description

DHCP server DNS Secondary Address

TCPIP_DHCPS_LEASE_DURATION Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_LEASE_DURATION TCPIP_DHCPS_LEASE_SOLVED_ENTRY_TMO
```

Description

Timeout for a solved entry in the cache, in seconds the entry will be removed if the TMO elapsed and the entry has not been referenced again

TCPIP_DHCPS_LEASE_ENTRIES_DEFAULT Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_LEASE_ENTRIES_DEFAULT 15
```

Description

The Maximum Number of entries in the lease table Default total number of entries for all the the interface

TCPIP_DHCPS_LEASE_REMOVED_BEFORE_ACK Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_LEASE_REMOVED_BEFORE_ACK (5)
```

Description

Timeout for a unsolved entry , in seconds and should be removed from the entry if there is no REQUEST after OFFER

TCPIP_DHCPS_LEASE_SOLVED_ENTRY_TMO Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_LEASE_SOLVED_ENTRY_TMO (20 * 60)
```

Description

Timeout for a solved entry in the cache, in seconds. The entry will be removed if the TMO lapsed and the entry has not been referenced again

TCPIP_DHCPS_TASK_PROCESS_RATE Macro

File

dhcps_config.h

C

```
#define TCPIP_DHCPS_TASK_PROCESS_RATE (200)
```

Description

DHCPS task processing rate, in milliseconds. The DHCPS module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other TMO are multiple of this The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Building the Library

This section lists the files that are available in the DHCP Server module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dhcps.c`	DHCP Server implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The DHCP Server module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Library Interface

a) Configuration Functions

	Name	Description
	TCPIP_DHCPS_Disable	Disables the DHCP Server for the specified interface.

	TCPIP_DHCPS_Enable	Enables the DHCP Server for the specified interface.
	TCPIP_DHCPS_RemovePoolEntries	Removes all the entries or only used entries of a certain type belonging to a network interface.
	TCPIP_DHCPS_LeaseEntryRemove	Remove one entry from the DHCP server leased entry.
	TCPIP_DHCP_RequestTimeoutSet	Sets the DHCP client request and base time-out values.
	TCPIP_DHCP_Task	Standard TCP/IP stack module task function.
	TCPIP_DHCPS_Task	Standard TCP/IP stack module task function.

b) Status Functions

	Name	Description
	TCPIP_DHCPS_IsEnabled	Determines if the DHCP Server is enabled on the specified interface.
	TCPIP_DHCPS_GetPoolEntries	Get all the entries or only used entries of a certain type belonging to a network interface.
	TCPIP_DHCPS_LeaseEntryGet	Get the lease entry details as per TCPIP_DHCPS_LEASE_HANDLE and per interface.
	TCPIP_DHCP_InfoGet	Returns information about the DHCP client on the specified interface.

c) Data Types and Constants

	Name	Description
	TCPIP_DHCPS_LEASE_ENTRY	DHCP Server module runtime and initialization configuration data.
	TCPIP_DHCPS_LEASE_HANDLE	DHCP Server Lease Handle
	TCPIP_DHCPS_POOL_ENTRY_TYPE	DHCP server pool types are used to get and remove the leased IP address details.
	TCPIP_DHCPS_ADDRESS_CONFIG	DHCP server configuration and IP address range.
	TCPIP_DHCP_CLIENT_ENABLED	Default value for the enable/disable the DHCP client at stack start-up.
	TCPIP_DHCP_TASK_TICK_RATE	The DHCP task processing rate: number of milliseconds to generate an DHCP tick. Used by the DHCP state machine The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DHCP_TIMEOUT	Defines how long to wait before a DHCP lease is acquired when the DHCP module is enabled, seconds
	TCPIP_DHCPS_MODULE_CONFIG	DHCP Server module runtime and initialization configuration data.

Description

This section describes the Application Programming Interface (API) functions of the DHCP Server module.

Refer to each section for a detailed description.

a) Configuration Functions

TCPIP_DHCPS_Disable Function

Disables the DHCP Server for the specified interface.

File

dhcps.h

C

```
bool TCPIP_DHCPS_Disable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function disables the DHCP Server for the specified interface. If it is already disabled, no action is taken.

Remarks

When the interface continues using its old configuration, it is possible that the lease may take sometime to expire. And The communication will be there until it is not expired.Lease time is configured in dhcps_config.h.

Preconditions

The DHCP Server module must be initialized.

Parameters

Parameters	Description
hNet	Interface on which to disable the DHCP Server

Function

bool TCPIP_DHCPS_Disable(TCPIP_NET_HANDLE hNet)

TCPIP_DHCPS_Enable Function

Enables the DHCP Server for the specified interface.

File

dhcps.h

C

```
bool TCPIP_DHCPS_Enable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function enables the DHCP Server for the specified interface, if it is disabled. If it is already enabled, nothing is done.

Preconditions

The DHCP Server module must be initialized.

Parameters

Parameters	Description
hNet	Interface on which to enable the DHCP Server

Function

void TCPIP_DHCPS_Enable(TCPIP_NET_HANDLE hNet)

TCPIP_DHCPS_RemovePoolEntries Function

Removes all the entries or only used entries of a certain type belonging to a network interface.

File

dhcps.h

C

```
bool TCPIP_DHCPS_RemovePoolEntries(TCPIP_NET_HANDLE netH, TCPIP_DHCPS_POOL_ENTRY_TYPE type);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function is used to remove the DHCP server entries from the pool as per TCPIP_DHCPS_POOL_ENTRY_TYPE.

Remarks

None.

Preconditions

the DHCP Server module should have been initialized.

Parameters

Parameters	Description
hNet	Interface handle to use

type	type of entries to remove: DHCP_SERVER_POOL_ENTRY_ALL, DHCP_SERVER_POOL_ENTRY_IN_USE

Function

bool TCPIP_DHCPS_RemovePoolEntries(TCPIP_NET_HANDLE netH, TCPIP_DHCPS_POOL_ENTRY_TYPE type);

TCPIP_DHCPS_LeaseEntryRemove Function

Remove one entry from the DHCP server leased entry.

File

dhcps.h

C

```
bool TCPIP_DHCPS_LeaseEntryRemove(TCPIP_NET_HANDLE netH, TCPIP_MAC_ADDR* hwAdd);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function is used to remove one entry from the leased HASH table with respect to the interface and the MAC address.

Remarks

This function is called from the command line to remove one entry and from the Wi-FI Driver module to remove a node that is disconnected from the AP.

Preconditions

The DHCP Server module should have been initialized.

Parameters

Parameters	Description
netH	Interface handle to use
hwAdd	MAC address that need to be removed from the HASH table

Function

bool TCPIP_DHCPS_LeaseEntryRemove(TCPIP_NET_HANDLE netH, TCPIP_MAC_ADDR* hwAdd)

TCPIP_DHCP_RequestTimeoutSet Function

Sets the DHCP client request and base time-out values.

File

dhcp.h

C

```
bool TCPIP_DHCP_RequestTimeoutSet(TCPIP_NET_HANDLE hNet, uint16_t initTmo, uint16_t dhcpBaseTmo);
```

Returns

- true - if successful
- false - if a wrong interface handle or time-out value was provided

Description

This function allows the run time adjustment of the DHCP time-out values. It specifies for how long the client has to wait for a valid DHCP server reply during the initialization process until acquisition of the host address is considered to have failed. It also sets the DHCP base timeout for DHCP transactions with the server. This is automatically incremented by the DHCP client using an exponential back-off algorithm. Recommended value is 2 <= dhcpBaseTmo <= 64 seconds.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface handle
initTmo	Initialization time-out to wait for a valid lease acquisition, in seconds
dhcpBaseTmo	DHCP time-out value for DHCP requests, in seconds

Function

bool TCPIP_DHCP_RequestTimeoutSet(TCPIP_NET_HANDLE hNet, uint16_t initTmo,
uint16_t dhcpBaseTmo)

TCPIP_DHCP_Task Function

Standard TCP/IP stack module task function.

File

dhcp.h

C

```
void TCPIP_DHCP_Task();
```

Returns

None.

Description

This function performs DHCP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

DHCP module should have been initialized

Function

void TCPIP_DHCP_Task(void)

TCPIP_DHCPS_Task Function

Standard TCP/IP stack module task function.

File

dhcps.h

C

```
void TCPIP_DHCPS_Task();
```

Returns

None.

Description

This function performs DHCP Server module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The DHCP Server module should have been initialized.

Function

void TCPIP_DHCPS_Task(void)

b) Status Functions

TCPIP_DHCPS_IsEnabled Function

Determines if the DHCP Server is enabled on the specified interface.

File

dhcps.h

C

```
bool TCPIP_DHCPS_IsEnabled(TCPIP_NET_HANDLE hNet);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function returns the current state of the DHCP Server on the specified interface.

Preconditions

The DHCP Server module must be initialized.

Parameters

Parameters	Description
hNet	Interface to query

Function

bool TCPIP_DHCPS_IsEnabled(TCPIP_NET_HANDLE hNet)

TCPIP_DHCPS_GetPoolEntries Function

Get all the entries or only used entries of a certain type belonging to a network interface.

File

dhcps.h

C

```
int TCPIP_DHCPS_GetPoolEntries(TCPIP_NET_HANDLE netH, TCPIP_DHCPS_POOL_ENTRY_TYPE type);
```

Returns

- true - If successful
- false - If unsuccessful

Description

This function is used to get the DHCP server entries from the pool as per TCPIP_DHCPS_POOL_ENTRY_TYPE.

Remarks

None.

Preconditions

The DHCP server module should have been initialized.

Parameters

Parameters	Description
hNet	Interface handle to use
type	type of entries to remove: • DHCP_SERVER_POOL_ENTRY_ALL, • DHCP_SERVER_POOL_ENTRY_IN_USE

Function

int TCPIP_DHCPS_GetPoolEntries(TCPIP_NET_HANDLE netH, TCPIP_DHCPS_POOL_ENTRY_TYPE type);

TCPIP_DHCPS_LeaseEntryGet Function

Get the lease entry details as per TCPIP_DHCPS_LEASE_HANDLE and per interface.

File

dhcps.h

C

```
TCPIP_DHCPS_LEASE_HANDLE TCPIP_DHCPS_LeaseEntryGet(TCPIP_NET_HANDLE netH, TCPIP_DHCPS_LEASE_ENTRY*
pLeaseEntry, TCPIP_DHCPS_LEASE_HANDLE leaseHandle);
```

Returns

- non-zero TCPIP_DHCPS_LEASE_HANDLE - To be used in the subsequent calls
- 0 - If end of list or wrong interface, or DHCP server is not running on that interface

Description

This function returns a lease entry for the TCPIP_DHCPS_LEASE_HANDLE. if the lease entry is not present for that TCPIP_DHCPS_LEASE_HANDLE, then it will return the next valid lease entry.

Preconditions

The DHCP Server module must be initialized.

Parameters

Parameters	Description
netH	Lease entry for this Interface
pLeaseEntry	Client lease entry details
leaseHandle	Lease index

Function

TCPIP_DHCPS_LEASE_HANDLE TCPIP_DHCPS_LeaseEntryGet(TCPIP_NET_HANDLE netH,
TCPIP_DHCPS_LEASE_ENTRY* pLeaseEntry, TCPIP_DHCPS_LEASE_HANDLE leaseHandle);

TCPIP_DHCP_InfoGet Function

Returns information about the DHCP client on the specified interface.

File

dhcp.h

C

```
bool TCPIP_DHCP_InfoGet(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_INFO* pDhcpInfo);
```

Returns

- true - if the interface is enabled and exists and the DHCP client service information filled in the supplied storage
- false - if the specified interface is not enabled, does not exist, or does not have a DHCP client

Description

This function returns the current state and lease info of the DHCP client on the specified interface.

Preconditions

The DHCP module must be initialized.

Parameters

Parameters	Description
hNet	Interface to query.
pDhcpInfo	storage to return the DHCP info

Function

bool TCPIP_DHCP_InfoGet(TCPIP_NET_HANDLE hNet, TCPIP_DHCP_INFO* pDhcpInfo);

c) Data Types and Constants

TCPIP_DHCPS_LEASE_ENTRY Structure

DHCP Server module runtime and initialization configuration data.

File

dhcps.h

C

```
typedef struct {
  TCPIP_MAC_ADDR hwAdd;
  IPV4_ADDR ipAddress;
  uint32_t leaseTime;
} TCPIP_DHCPS_LEASE_ENTRY;
```

Members

Members	Description
TCPIP_MAC_ADDR hwAdd;	Client MAC address
IPV4_ADDR ipAddress;	Leased IP address
uint32_t leaseTime;	Lease period

Description

Structure: TCPIP_DHCPS_MODULE_CONFIG

DHCP server configuration and initialization data. Configuration is part of tcpip_stack_init.c.

TCPIP_DHCPS_LEASE_HANDLE Type

DHCP Server Lease Handle

File

dhcps.h

C

```
typedef const void* TCPIP_DHCPS_LEASE_HANDLE;
```

Description

Type: TCPIP_DHCPS_LEASE_HANDLE

A handle that server is using to provide the index of a lease entry.

Remarks

This handle is used by command handler to get the Index of Lease entry.

TCPIP_DHCPS_POOL_ENTRY_TYPE Enumeration

DHCP server pool types are used to get and remove the leased IP address details.

File

dhcps.h

C

```
typedef enum {
  DHCP_SERVER_POOL_ENTRY_ALL,
  DHCP_SERVER_POOL_ENTRY_IN_USE
} TCPIP_DHCPS_POOL_ENTRY_TYPE;
```

Members

Members	Description
DHCP_SERVER_POOL_ENTRY_ALL	Get or Remove all the Leased address
DHCP_SERVER_POOL_ENTRY_IN_USE	Get or remove only Leased IP address

Description

Enumeration: TCPIP_DHCPS_POOL_ENTRY_TYPE

DHCP_SERVER_POOL_ENTRY_ALL - Get or Remove all the leased address which includes both solved and unsolved entries.
DHCP_SERVER_POOL_ENTRY_IN_USE - Get or Remove only solved leased IP address.

TCPIP_DHCPS_ADDRESS_CONFIG Structure

DHCP server configuration and IP address range.

File

dhcps.h

C

```
typedef struct {
  int interfaceIndex;
  char* serverIPAddress;
  char* startIPAddRange;
  char* ipMaskAddress;
  char* priDNS;
  char* secondDNS;
  bool poolEnabled;
} TCPIP_DHCPS_ADDRESS_CONFIG;
```

Members

Members	Description
int interfaceIndex;	Interface Index
char* serverIPAddress;	Server IP address
char* startIPAddRange;	Start IP address
char* ipMaskAddress;	Netmask
char* priDNS;	Primary DNS server Address
char* secondDNS;	Secondary DNS server Address
bool poolEnabled;	true if pool is valid , false if pool is invalid

Description

Structure: TCPIP_DHCPS_ADDRESS_CONFIG

DHCP server configuration and network initialization data. Configuration is part of tcpip_stack_init.c.

TCPIP_DHCP_CLIENT_ENABLED Macro

File

dhcp_config.h

C

```
#define TCPIP_DHCP_CLIENT_ENABLED 1
```

Description

Default value for the enable/disable the DHCP client at stack start-up.

Remarks

the interface initialization setting in TCPIP_NETWORK_CONFIG takes precedence!

TCPIP_DHCP_TASK_TICK_RATE Macro

File

dhcp_config.h

C

```
#define TCPIP_DHCP_TASK_TICK_RATE (200)
```

Description

The DHCP task processing rate: number of milliseconds to generate an DHCP tick. Used by the DHCP state machine The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_DHCP_TIMEOUT Macro

File

dhcp_config.h

C

```
#define TCPIP_DHCP_TIMEOUT (10)
```

Description

Defines how long to wait before a DHCP lease is acquired when the DHCP module is enabled, seconds

TCPIP_DHCPS_MODULE_CONFIG Structure

DHCP Server module runtime and initialization configuration data.

File

dhcps.h

C

```
typedef struct {
  bool enabled;
  bool deleteOldLease;
  size_t leaseEntries;
  uint32_t entrySolvedTmo;
  TCPIP_DHCPS_ADDRESS_CONFIG * dhcpServer;
} TCPIP_DHCPS_MODULE_CONFIG;
```

Members

Members	Description
bool enabled;	enable DHCP server
bool deleteOldLease;	delete old cache if still in place, specific DHCP parameters
size_t leaseEntries;	max number of lease entries
uint32_t entrySolvedTmo;	solved entry removed after this tmo in seconds
TCPIP_DHCPS_ADDRESS_CONFIG * dhcpServer;	DHCP server lease address configuration details uint32_t dhcpServerCnt; // Max DHCP server support

Description

Structure: TCPIP_DHCPS_MODULE_CONFIG

DHCP server configuration and initialization data . Configuration is part of tcpip_stack_init.c.

Files

Files

Name	Description
dhcps.h	Dynamic Host Configuration Protocol(DHCP) Server APIs.
dhcps_config.h	DHCPS configuration file

Description

This section lists the source and header files used by the library.

dhcps.h

Dynamic Host Configuration Protocol(DHCP) Server APIs.

Enumerations

	Name	Description
	TCPIP_DHCPS_POOL_ENTRY_TYPE	DHCP server pool types are used to get and remove the leased IP address details.

Functions

	Name	Description
	TCPIP_DHCPS_Disable	Disables the DHCP Server for the specified interface.
	TCPIP_DHCPS_Enable	Enables the DHCP Server for the specified interface.
	TCPIP_DHCPS_GetPoolEntries	Get all the entries or only used entries of a certain type belonging to a network interface.
	TCPIP_DHCPS_IsEnabled	Determines if the DHCP Server is enabled on the specified interface.
	TCPIP_DHCPS_LeaseEntryGet	Get the lease entry details as per TCPIP_DHCPS_LEASE_HANDLE and per interface.
	TCPIP_DHCPS_LeaseEntryRemove	Remove one entry from the DHCP server leased entry.
	TCPIP_DHCPS_RemovePoolEntries	Removes all the entries or only used entries of a certain type belonging to a network interface.
	TCPIP_DHCPS_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_DHCPS_ADDRESS_CONFIG	DHCP server configuration and IP address range.
	TCPIP_DHCPS_LEASE_ENTRY	DHCP Server module runtime and initialization configuration data.
	TCPIP_DHCPS_MODULE_CONFIG	DHCP Server module runtime and initialization configuration data.

Types

	Name	Description
	TCPIP_DHCPS_LEASE_HANDLE	DHCP Server Lease Handle

Description

Dynamic Host Configuration Protocol (DHCP) Server API Header File

The DHCP server assigns a free IP address to a requesting client from the range defined in the dhcps_config.h file. Lease time per IP address is decided as per the TMO configuration which is defined in dhcps_config.h.

File Name

dhcps.h

Company

Microchip Technology Inc.

dhcps_config.h

DHCPS configuration file

Macros

	Name	Description
	TCPIP_DHCPS_DEFAULT_IP_ADDRESS_RANGE_START	These below IPv4 DHCP server address details are default address and it is assigned to the network default network interface. for Other interfaces , tcpip_stack_init.c file should be use to configure DHCP_POOL_CONFIG[]. IPv4 Address range is starting from 100, because the from 1 to 100 is reserved. Reserved Address will be used for the gateway address. Start of IP address Range , network_config.h ipaddress and this start of IP address should be in same SUBNET RECOMENDED - network_config.h ipaddress should be 192.168.1.1 if DHCP server ip address range starts from 192.168.1.100.
	TCPIP_DHCPS_DEFAULT_SERVER_IP_ADDRESS	DHCP server Address per interface. DHCP server Address selection should be in the same subnet.
	TCPIP_DHCPS_DEFAULT_SERVER_NETMASK_ADDRESS	DHCP server subnet Address per interface.
	TCPIP_DHCPS_DEFAULT_SERVER_PRIMARY_DNS_ADDRESS	DHCP server DNS primary Address
	TCPIP_DHCPS_DEFAULT_SERVER_SECONDARY_DNS_ADDRESS	DHCP server DNS Secondary Address
	TCPIP_DHCPS_LEASE_DURATION	Timeout for a solved entry in the cache, in seconds the entry will be removed if the TMO elapsed and the entry has not been referenced again
	TCPIP_DHCPS_LEASE_ENTRIES_DEFAULT	The Maximum Number of entries in the lease table Default total number of entries for all the the interface
	TCPIP_DHCPS_LEASE_REMOVED_BEFORE_ACK	Timeout for a unsolved entry , in seconds and should be removed from the entry if there is no REQUEST after OFFER
	TCPIP_DHCPS_LEASE_SOLVED_ENTRY_TMO	Timeout for a solved entry in the cache, in seconds. The entry will be removed if the TMO lapsed and the entry has not been referenced again
	TCPIP_DHCPS_TASK_PROCESS_RATE	DHCPS task processing rate, in milliseconds. The DHCPS module will process a timer event with this rate for maintaining its own queues, processing timeouts, etc. Choose it so that the other TMO are multiple of this The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

Dynamic Host Configuration Protocol (DHCPS) Configuration file

This file contains the DHCPS module configuration options

File Name

dhcps_config.h

Company

Microchip Technology Inc.

DHCPv6 Module

This section describes the TCP/IP Stack Library DHCPv6 module.

Introduction

TCP/IP Stack Library IPv6 Dynamic Host Configuration Protocol (DHCPv6) Module for Microchip Microcontrollers

This library provides the API of the DHCPv6 module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The DHCPv6 client module will allow your application to dynamically obtain an IPv6 address from a DHCPv6 server. Additionally, the DHCPv6 client will get other parameters, such as DNS servers and the domain search list.

Using the Library

This topic describes the basic architecture of the DHCPv6 TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `dhcpv6.h`

The interface to the DHCPv6 TCP/IP Stack library is defined in the `dhcpv6.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the DHCPv6 TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

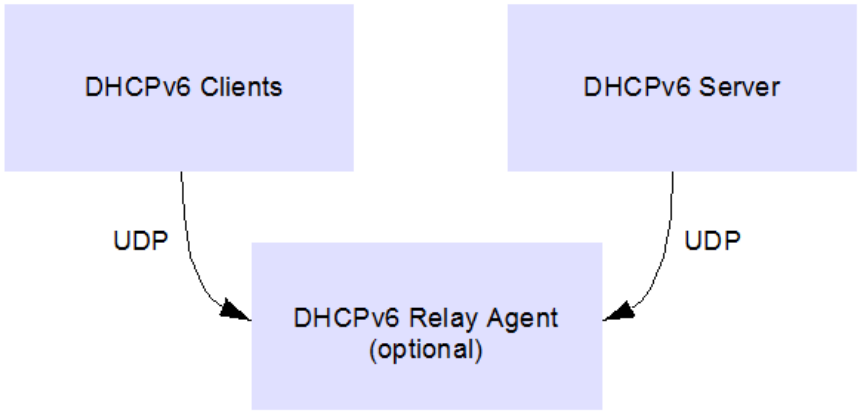
This library provides the API of the DHCPv6 TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

This module provides software abstraction of the DHCPv6 module existent in the TCP/IP Stack implementation. The DHCPv6 module works together with the IPv6 Stateless Address Auto-configuration protocol and provides stateful temporary/non-temporary (IATA, IANA) global addresses for the IPv6 host.

DHCPv6 Software Abstraction Block Diagram

Typical DCHPv6 Usage



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DHCPv6 module.

Library Interface Section	Description
Configuration Functions	Routines for Configuring DHCPv6
Status Functions	Routines for Obtaining DHCPv6 Status Information
Data Types and Constants	This section provides various definitions describing this API

How the Library Works

The IPv6 Dynamic Host Configuration Protocol (DHCPv6) is a standard networking protocol used to automatically allocate IPv6 addresses for hosts in a network.

The DHCPv6 server has a pool of IP addresses, which are leased for clients requesting them. The leases have a limited lifetime after which the hosts need to renew the lease or acquire a new one.

The DHCPv6 client module in the TCP/IP stack takes care of the communication with the DHCPv6 server and renewing the lease when the lifetime expires.

Configuring the Library

Macros

	Name	Description
	TCPIP_DHCPV6_CLIENT_DUID_TYPE	Default DUID type to be used by the client
	TCPIP_DHCPV6_CLIENT_PORT	Clients listen for DHCP messages on UDP port:
	TCPIP_DHCPV6_DNS_SERVERS_NO	number of DNS servers to store from a DHCP server reply
	TCPIP_DHCPV6_DOMAIN_SEARCH_LIST_SIZE	space for the Domain Search List option - multiple of 16
	TCPIP_DHCPV6_FORCED_SERVER_PREFERENCE	preference value that forces the server selection 8 bit value!
	TCPIP_DHCPV6_IA_FREE_DESCRIPTORS_NO	maximum number of free IA descriptors per client
	TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_PREF_LTIME	default lifetimes for the solicited addresses
	TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME	This is macro TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME.
	TCPIP_DHCPV6_IANA_DEFAULT_T1	default values for IANA T1, T2
	TCPIP_DHCPV6_IANA_DEFAULT_T2	This is macro TCPIP_DHCPV6_IANA_DEFAULT_T2.
	TCPIP_DHCPV6_IANA_DESCRIPTORS_NO	maximum number of IANA descriptors per client
	TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO	default addresses for IANA in a solicit message
	TCPIP_DHCPV6_IANA_SOLICIT_DEFAULT_ADDRESS	default values for the IANA Solicit addresses irrelevant if TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO == 0 should be a value for each TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO
	TCPIP_DHCPV6_IANA_SOLICIT_T1	default values for IANA T1, T2 in a solicit message
	TCPIP_DHCPV6_IANA_SOLICIT_T2	This is macro TCPIP_DHCPV6_IANA_SOLICIT_T2.
	TCPIP_DHCPV6_IATA_DEFAULT_T1	default values for IATA T1, T2 If 0, the timeout will be infinite (0xffffffff)
	TCPIP_DHCPV6_IATA_DEFAULT_T2	This is macro TCPIP_DHCPV6_IATA_DEFAULT_T2.
	TCPIP_DHCPV6_IATA_DESCRIPTORS_NO	maximum number of IATA descriptors per client
	TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO	default addresses for IATA in a solicit message
	TCPIP_DHCPV6_IATA_SOLICIT_DEFAULT_ADDRESS	default values for the IANA Solicit addresses irrelevant if TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO == 0 should be a value for each TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO
	TCPIP_DHCPV6_MESSAGE_BUFFER_SIZE	default value of the buffer to assemble messages, etc.
	TCPIP_DHCPV6_MESSAGE_BUFFERS	default number of buffers
	TCPIP_DHCPV6_MIN_UDP_TX_BUFFER_SIZE	minimum size of a UDP buffer
	TCPIP_DHCPV6_SERVER_PORT	Servers and relay agents listen for DHCP messages on UDP port:
	TCPIP_DHCPV6_SKIP_DAD_PROCESS	defining this symbol will skip the DAD processing for DHCPv6 generated addresses
	TCPIP_DHCPV6_STATUS_CODE_MESSAGE_LEN	number of character to reserve for a server status code associated message
	TCPIP_DHCPV6_TASK_TICK_RATE	The DHCPv6 task processing rate: number of milliseconds to generate an DHCPv6 tick. Used by the DHCPv6 state machine The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_STACK_USE_DHCPV6_CLIENT	This symbol enables the DHCPv6 client as being part of the build
	TCPIP_DHCPV6_USER_NOTIFICATION	allow DHCPV6 client user notification if enabled, the TCPIP_DHCPV6_HandlerRegister/TCPIP_DHCPV6_HandlerDeRegister functions exist and can be used

Description

The configuration of the DHCPv6 TCP/IP Stack is based on the file `system_config.h` (which may include a `dhcpv6_config.h` file).

This header file contains the configuration parameters for the DHCPv6 TCP/IP Stack. Based on the selections made, the DHCPv6 TCP/IP Stack will adjust its behavior.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_DHCPV6_CLIENT_DUID_TYPE Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_CLIENT_DUID_TYPE TCPIP_DHCPV6_DUID_TYPE_LL
```

Description

Default DUID type to be used by the client

TCPIP_DHCPV6_CLIENT_PORT Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_CLIENT_PORT 546
```

Description

Clients listen for DHCP messages on UDP port:

TCPIP_DHCPV6_DNS_SERVERS_NO Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_DNS_SERVERS_NO 2
```

Description

number of DNS servers to store from a DHCP server reply

TCPIP_DHCPV6_DOMAIN_SEARCH_LIST_SIZE Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_DOMAIN_SEARCH_LIST_SIZE 64
```

Description

space for the Domain Search List option - multiple of 16

TCPIP_DHCPV6_FORCED_SERVER_PREFERENCE Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_FORCED_SERVER_PREFERENCE 255
```

Description

preference value that forces the server selection 8 bit value!

TCPIP_DHCPV6_IA_FREE_DESCRIPTORS_NO Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IA_FREE_DESCRIPTORS_NO 2
```

Description

maximum number of free IA descriptors per client

TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_PREF_LTIME Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_PREF_LTIME 0
```

Description

default lifetimes for the solicited addresses

TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME 0
```

Description

This is macro TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME.

TCPIP_DHCPV6_IANA_DEFAULT_T1 Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_DEFAULT_T1 0
```

Description

default values for IANA T1, T2

TCPIP_DHCPV6_IANA_DEFAULT_T2 Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_DEFAULT_T2 0
```

Description

This is macro TCPIP_DHCPV6_IANA_DEFAULT_T2.

TCPIP_DHCPV6_IANA_DESCRIPTORS_NO Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_DESCRIPTORS_NO 4
```

Description

maximum number of IANA descriptors per client

TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO 0
```

Description

default addresses for IANA in a solicit message

TCPIP_DHCPV6_IANA_SOLICIT_DEFAULT_ADDRESS Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_SOLICIT_DEFAULT_ADDRESS "::0"
```

Description

default values for the IANA Solicit addresses irrelevant if TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO == 0 should be a value for each TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO

TCPIP_DHCPV6_IANA_SOLICIT_T1 Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_SOLICIT_T1 0
```

Description

default values for IANA T1, T2 in a solicit message

TCPIP_DHCPV6_IANA_SOLICIT_T2 Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IANA_SOLICIT_T2 0
```

Description

This is macro TCPIP_DHCPV6_IANA_SOLICIT_T2.

TCPIP_DHCPV6_IATA_DEFAULT_T1 Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IATA_DEFAULT_T1 0
```

Description

default values for IATA T1, T2 If 0, the timeout will be infinite (0xffffffff)

TCPIP_DHCPV6_IATA_DEFAULT_T2 Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IATA_DEFAULT_T2 0
```

Description

This is macro TCPIP_DHCPV6_IATA_DEFAULT_T2.

TCPIP_DHCPV6_IATA_DESCRIPTORS_NO Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IATA_DESCRIPTORS_NO 2
```

Description

maximum number of IATA descriptors per client

TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO 0
```

Description

default addresses for IATA in a solicit message

TCPIP_DHCPV6_IATA_SOLICIT_DEFAULT_ADDRESS Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_IATA_SOLICIT_DEFAULT_ADDRESS 0
```

Description

default values for the IANA Solicit addresses irrelevant if TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO == 0 should be a value for each TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO

TCPIP_DHCPV6_MESSAGE_BUFFER_SIZE Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_MESSAGE_BUFFER_SIZE 512
```

Description

default value of the buffer to assemble messages, etc.

TCPIP_DHCPV6_MESSAGE_BUFFERS Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_MESSAGE_BUFFERS 8
```

Description

default number of buffers

TCPIP_DHCPV6_MIN_UDP_TX_BUFFER_SIZE Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_MIN_UDP_TX_BUFFER_SIZE 512
```

Description

minimum size of a UDP buffer

TCPIP_DHCPV6_SERVER_PORT Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_SERVER_PORT 547
```

Description

Servers and relay agents listen for DHCP messages on UDP port:

TCPIP_DHCPV6_SKIP_DAD_PROCESS Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_SKIP_DAD_PROCESS
```

Description

defining this symbol will skip the DAD processing for DHCPv6 generated addresses

TCPIP_DHCPV6_STATUS_CODE_MESSAGE_LEN Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_STATUS_CODE_MESSAGE_LEN 0
```

Description

number of character to reserve for a server status code associated message

TCPIP_DHCPV6_TASK_TICK_RATE Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_TASK_TICK_RATE (100)
```

Description

The DHCPv6 task processing rate: number of milliseconds to generate an DHCPv6 tick. Used by the DHCPv6 state machine The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_STACK_USE_DHCPV6_CLIENT Macro

File

dhcpv6_config.h

C

```
#define TCPIP_STACK_USE_DHCPV6_CLIENT
```

Description

This symbol enables the DHCPv6 client as being part of the build

TCPIP_DHCPV6_USER_NOTIFICATION Macro

File

dhcpv6_config.h

C

```
#define TCPIP_DHCPV6_USER_NOTIFICATION false
```

Description

allow DHCPV6 client user notification if enabled, the TCPIP_DHCPV6_HandlerRegister/TCPIP_DHCPV6_HandlerDeRegister functions exist and can be used

Building the Library

This section lists the files that are available in the DHCPv6 module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/tcpip.h	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dhcpv6.c	DHCPv6 implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The DHCPv6 module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Library Interface

a) Configuration Functions

	Name	Description
	TCPIP_DHCPV6_HandlerDeRegister	DHCPV6 event deregistration
	TCPIP_DHCPV6_HandlerRegister	DHCPV6 event registration
	TCPIP_DHCPV6_Task	Standard TCP/IP stack module task function.

b) Status Functions

	Name	Description
	TCPIP_DHCPV6_ClientInfoGet	client status reporting
	TCPIP_DHCPV6_IaInfoGet	IA status reporting

c) Data Types and Constants

	Name	Description
	TCPIP_DHCPV6_CLIENT_INFO	DHCPv6 client info
	TCPIP_DHCPV6_CLIENT_STATE	DHCPv6 Current Status
	TCPIP_DHCPV6_CONFIG_FLAGS	DHCPV6 start up flags
	TCPIP_DHCPV6_DUID_TYPE	types of DUID for DHCPv6
	TCPIP_DHCPV6_EVENT_HANDLER	DHCPv6 event handler prototype.
	TCPIP_DHCPV6_HANDLE	a DHCPV6 handle
	TCPIP_DHCPV6_IA_EVENT	IA event info
	TCPIP_DHCPV6_IA_INFO	DHCPv6 IA info
	TCPIP_DHCPV6_IA_STATE	IA run states
	TCPIP_DHCPV6_IA_SUBSTATE	IA run substates most IA run states that must send a message go through these substates
	TCPIP_DHCPV6_IA_TYPE	supported types of IA
	TCPIP_DHCPV6_MODULE_CONFIG	DHCPv6 module configuration
	TCPIP_DHCPV6_SERVER_STATUS_CODE	DHCPV6 server status code

Description

This section describes the Application Programming Interface (API) functions of the DHCPv6 module.

Refer to each section for a detailed description.

a) Configuration Functions

TCPIP_DHCPV6_HandlerDeRegister Function

File

dhcpv6.h

C

```
bool TCPIP_DHCPV6_HandlerDeRegister(TCPIP_DHCPV6_HANDLE hDhcp);
```

Description

DHCPV6 event deregistration

TCPIP_DHCPV6_HandlerRegister Function

File

dhcpv6.h

C

```
TCPIP_DHCPV6_HANDLE TCPIP_DHCPV6_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_DHCPV6_EVENT_HANDLER handler,
const void* hParam);
```

Description

DHCPV6 event registration

TCPIP_DHCPV6_Task Function

Standard TCP/IP stack module task function.

File

dhcpv6.h

C

```
void TCPIP_DHCPV6_Task();
```

Returns

None.

Description

This function performs DHCPv6 module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

DHCPv6 module should have been initialized

Function

void TCPIP_DHCPV6_Task(void)

b) Status Functions

TCPIP_DHCPV6_ClientInfoGet Function

File

dhcpv6.h

C

```
bool TCPIP_DHCPV6_ClientInfoGet(TCPIP_NET_HANDLE hNet, TCPIP_DHCPV6_CLIENT_INFO* pClientInfo);
```

Description

client status reporting

TCPIP_DHCPV6_IaInfoGet Function

File

dhcpv6.h

C

```
bool TCPIP_DHCPV6_IaInfoGet(TCPIP_NET_HANDLE hNet, int iaIx, TCPIP_DHCPV6_IA_INFO* pIaInfo);
```

Description

IA status reporting

c) Data Types and Constants

TCPIP_DHCPV6_CLIENT_INFO Structure

File

dhcpv6.h

C

```
typedef struct {
  TCPIP_DHCPV6_CLIENT_STATE clientState;
  int nIanas;
  int nIatas;
  int nFreeIas;
  uint32_t dhcpTime;
  TCPIP_DHCPV6_SERVER_STATUS_CODE lastStatusCode;
  void* statusBuff;
  size_t statusBuffSize;
  int nDnsServers;
  IPV6_ADDR* dnsBuff;
  size_t dnsBuffSize;
  int domainSearchListSize;
  void* domainBuff;
  size_t domainBuffSize;
} TCPIP_DHCPV6_CLIENT_INFO;
```

Members

Members	Description
TCPIP_DHCPV6_CLIENT_STATE clientState;	client state at the moment of the call
int nIanas;	number of IANA the client has
int nIatas;	number of IATA the client has
int nFreeIas;	number of free IAs the client has
uint32_t dhcpTime;	current DHCPV6 time; seconds
TCPIP_DHCPV6_SERVER_STATUS_CODE lastStatusCode;	last status code for the client
void* statusBuff;	buffer to copy the latest status message associated with the client
size_t statusBuffSize;	size of this buffer
int nDnsServers;	number of DNS servers
IPV6_ADDR* dnsBuff;	buffer to copy the DNS Servers obtained from the DHCPV6 server
size_t dnsBuffSize;	size of this buffer
int domainSearchListSize;	size of domainSearchList
void* domainBuff;	buffer to store the domain Search list obtained from the DHCPv6 server
size_t domainBuffSize;	size of this buffer

Description

DHCPv6 client info

TCPIP_DHCPV6_CLIENT_STATE Enumeration

DHCPv6 Current Status

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_CLIENT_STATE_INIT = 0,
  TCPIP_DHCPV6_CLIENT_STATE_IDLE,
  TCPIP_DHCPV6_CLIENT_STATE_RUN,
  TCPIP_DHCPV6_CLIENT_STATE_WAIT_LINK,
  TCPIP_DHCPV6_CLIENT_STATE_NUMBER
} TCPIP_DHCPV6_CLIENT_STATE;
```

Members

Members	Description
TCPIP_DHCPV6_CLIENT_STATE_INIT = 0	initialization state/unknown
TCPIP_DHCPV6_CLIENT_STATE_IDLE	idle/inactive state
TCPIP_DHCPV6_CLIENT_STATE_RUN	up and running in one of the run states
TCPIP_DHCPV6_CLIENT_STATE_WAIT_LINK	up and running, waiting for a connection
TCPIP_DHCPV6_CLIENT_STATE_NUMBER	number of states

Description

Enumeration: TCPIP_DHCPV6_CLIENT_STATE

This enumeration lists the current status of the DHCPv6 module. Used in getting info about the DHCPv6 state machine.

TCPIP_DHCPV6_CONFIG_FLAGS Enumeration

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_FLAG_NONE = 0,
  TCPIP_DHCPV6_FLAG_START_ENABLE = 0x01,
  TCPIP_DHCPV6_FLAG_DAD_DISABLE = 0x02,
  TCPIP_DHCPV6_FLAG_IA_IGNORE_RENEW_LTIME = 0x04,
  TCPIP_DHCPV6_FLAG_IA_IGNORE_REBIND_LTIME = 0x08,
  TCPIP_DHCPV6_FLAG_IA_NOTIFY_SUB_STATE = 0x80
} TCPIP_DHCPV6_CONFIG_FLAGS;
```

Members

Members	Description
TCPIP_DHCPV6_FLAG_START_ENABLE = 0x01	enable the DHCPv6 at stack start up
TCPIP_DHCPV6_FLAG_DAD_DISABLE = 0x02	disable the DAD processing for DHCP generated addresses Use only for testing or in special cases Default should be disabled
TCPIP_DHCPV6_FLAG_IA_IGNORE_RENEW_LTIME = 0x04	if enabled, the IA (and its associated address) renew process will be valid as dictated by t1/defaultIataT1 and its address preferred lifetime will be ignored If disabled, the IA and its address will attempt renew when the minimum of address preferred lifetime and t1/defaultIataT1 expired Default should be disabled
TCPIP_DHCPV6_FLAG_IA_IGNORE_REBIND_LTIME = 0x08	if enabled, the IA (and its associated address) rebind process will be valid as dictated by t2/defaultIataT2 and its address valid lifetime will be ignored If disabled, the IA and its address will attempt rebind when the minimum of address valid lifetime and t2/defaultIataT2 expired Default should be disabled

TCPIP_DHCPV6_FLAG_IA_NOTIFY_SUB_STATE = 0x80	if enabled, the IA notifications will be generated for IA substate changes too (finer grain) if disabled, notifications will be generated for IA state changes only Default should be disabled

Description

DHCPV6 start up flags

TCPIP_DHCPV6_DUID_TYPE Enumeration

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_DUID_TYPE_NONE = 0,
  TCPIP_DHCPV6_DUID_TYPE_LLT = 1,
  TCPIP_DHCPV6_DUID_TYPE_EN = 2,
  TCPIP_DHCPV6_DUID_TYPE_LL = 3
} TCPIP_DHCPV6_DUID_TYPE;
```

Members

Members	Description
TCPIP_DHCPV6_DUID_TYPE_NONE = 0	invalid
TCPIP_DHCPV6_DUID_TYPE_LLT = 1	LinkLayer + time
TCPIP_DHCPV6_DUID_TYPE_EN = 2	Enterprise Number
TCPIP_DHCPV6_DUID_TYPE_LL = 3	Link Layer Address

Description

types of DUID for DHCPv6

TCPIP_DHCPV6_EVENT_HANDLER Type

DHCPv6 event handler prototype.

File

dhcpv6.h

C

```
typedef void (* TCPIP_DHCPV6_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, TCPIP_DHCPV6_CLIENT_STATE clientState,
const TCPIP_DHCPV6_IA_EVENT* pDhcpIaEv, const void* param);
```

Description

DHCPv6 reported event structure:

Type: TCPIP_DHCPV6_EVENT_HANDLER

Prototype of a DHCP event handler. Clients can register a handler with the DHCP service. Once an DHCP event occurs the DHCP service will called the registered handler. The handler has to be short and fast. It is meant for setting an event flag, *not* for lengthy processing!

if pDhcpIaEvent == 0, no info is carried for a specific IA if pDhcpIaEvent != 0, the info carried by this pointer is not persistent and is valid only within the context of this event handler!

TCPIP_DHCPV6_HANDLE Type

File

dhcpv6.h

C

```
typedef const void* TCPIP_DHCPV6_HANDLE;
```

Description

a DHCPV6 handle

TCPIP_DHCPV6_IA_EVENT Union

File

dhcpv6.h

C

```
typedef union {
  uint32_t eventVal;
  struct {
    uint8_t iaType;
    uint8_t iaState;
    uint8_t iaSubState;
    uint8_t iaIndex;
  }
} TCPIP_DHCPV6_IA_EVENT;
```

Members

Members	Description
uint8_t iaType;	a TCPIP_DHCPV6_IA_TYPE value
uint8_t iaState;	a TCPIP_DHCPV6_IA_STATE value
uint8_t iaSubState;	a TCPIP_DHCPV6_IA_SUBSTATE value
uint8_t iaIndex;	index/ID of this IA for this client

Description

IA event info

TCPIP_DHCPV6_IA_INFO Structure

File

dhcpv6.h

C

```
typedef struct {
  TCPIP_DHCPV6_IA_TYPE iaType;
  TCPIP_DHCPV6_IA_STATE iaState;
  TCPIP_DHCPV6_IA_SUBSTATE iaSubState;
  int iaIndex;
  uint32_t iaId;
  uint32_t tAcquire;
  uint32_t t1;
  uint32_t t2;
  IPV6_ADDR ipv6Addr;
  uint32_t prefLTime;
  uint32_t validLTime;
  TCPIP_DHCPV6_SERVER_STATUS_CODE lastStatusCode;
  void* statusBuff;
  size_t statusBuffSize;
} TCPIP_DHCPV6_IA_INFO;
```

Members

Members	Description
TCPIP_DHCPV6_IA_TYPE iaType;	IA type
TCPIP_DHCPV6_IA_STATE iaState;	IA state
TCPIP_DHCPV6_IA_SUBSTATE iaSubState;	IA substate
int iaIndex;	index of this IA for this client
uint32_t iaId;	ID of this IA the following fields are meaningful only for iaState >= TCPIP_DHCPV6_IA_STATE_BOUND
uint32_t tAcquire;	time of which the address was acquired
uint32_t t1;	IANA only: extend lifetime contact server time
uint32_t t2;	IANA only: extend lifetime contact any server time
IPV6_ADDR ipv6Addr;	16 bytes IPV6 address associated with this IA
uint32_t prefLTime;	preferred life time for the IPv6 address; seconds

uint32_t validLTime;	valid life time for the IPv6 address; seconds
TCPIP_DHCPV6_SERVER_STATUS_CODE lastStatusCode;	last status code for this IA
void* statusBuff;	buffer to copy the latest status message associated with this IA
size_t statusBuffSize;	size of this buffer

Description

DHCPv6 IA info

TCPIP_DHCPV6_IA_STATE Enumeration

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_IA_STATE_SOLICIT,
  TCPIP_DHCPV6_IA_STATE_REQUEST,
  TCPIP_DHCPV6_IA_STATE_DAD,
  TCPIP_DHCPV6_IA_STATE_DECLINE,
  TCPIP_DHCPV6_IA_STATE_BOUND,
  TCPIP_DHCPV6_IA_STATE_RENEW,
  TCPIP_DHCPV6_IA_STATE_REBIND,
  TCPIP_DHCPV6_IA_STATE_CONFIRM,
  TCPIP_DHCPV6_IA_STATE_RELEASE,
  TCPIP_DHCPV6_IA_STATE_ERROR_TRANSIENT,
  TCPIP_DHCPV6_IA_STATE_ERROR_FATAL,
  TCPIP_DHCPV6_IA_STATE_NUMBER
} TCPIP_DHCPV6_IA_STATE;
```

Members

Members	Description
TCPIP_DHCPV6_IA_STATE_SOLICIT	solicitation
TCPIP_DHCPV6_IA_STATE_REQUEST	perform request
TCPIP_DHCPV6_IA_STATE_DAD	start the DAD state
TCPIP_DHCPV6_IA_STATE_DECLINE	decline the DAD addresses
TCPIP_DHCPV6_IA_STATE_BOUND	bound;
TCPIP_DHCPV6_IA_STATE_RENEW	renew address
TCPIP_DHCPV6_IA_STATE_REBIND	rebind address
TCPIP_DHCPV6_IA_STATE_CONFIRM	confirm address
TCPIP_DHCPV6_IA_STATE_RELEASE	release address
TCPIP_DHCPV6_IA_STATE_ERROR_TRANSIENT	an error occurred for which either user intervention is required
TCPIP_DHCPV6_IA_STATE_ERROR_FATAL	fatal error occurred; not properly configured options/buffers, etc;
TCPIP_DHCPV6_IA_STATE_NUMBER	number of run states

Description

IA run states

TCPIP_DHCPV6_IA_SUBSTATE Enumeration

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_IA_SUBSTATE_START,
  TCPIP_DHCPV6_IA_SUBSTATE_IDELAY,
  TCPIP_DHCPV6_IA_SUBSTATE_TRANSMIT,
  TCPIP_DHCPV6_IA_SUBSTATE_WAIT_REPLY,
  TCPIP_DHCPV6_IA_SUBSTATE_NUMBER
} TCPIP_DHCPV6_IA_SUBSTATE;
```

Members

Members	Description
TCPIP_DHCPV6_IA_SUBSTATE_START	message start/preparation
TCPIP_DHCPV6_IA_SUBSTATE_IDELAY	message wait for iDelay
TCPIP_DHCPV6_IA_SUBSTATE_TRANSMIT	send/transmit message
TCPIP_DHCPV6_IA_SUBSTATE_WAIT_REPLY	wait message reply
TCPIP_DHCPV6_IA_SUBSTATE_NUMBER	number of standard message sub-states

Description

IA run substates most IA run states that must send a message go through these substates

TCPIP_DHCPV6_IA_TYPE Enumeration

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_IA_TYPE_NONE,
  TCPIP_DHCPV6_IA_TYPE_IANA,
  TCPIP_DHCPV6_IA_TYPE_IATA,
  TCPIP_DHCPV6_IA_TYPE_NUMBER
} TCPIP_DHCPV6_IA_TYPE;
```

Members

Members	Description
TCPIP_DHCPV6_IA_TYPE_NONE	unused IA association
TCPIP_DHCPV6_IA_TYPE_IANA	IANA association
TCPIP_DHCPV6_IA_TYPE_IATA	IATA association
TCPIP_DHCPV6_IA_TYPE_NUMBER	number of types

Description

supported types of IA

TCPIP_DHCPV6_MODULE_CONFIG Structure

File

dhcpv6.h

C

```
typedef struct {
  TCPIP_DHCPV6_CONFIG_FLAGS configFlags;
  uint16_t dhcpCliPort;
  uint16_t dhcpSrvPort;
  uint16_t duidType;
  uint16_t nIanaDcpts;
  uint16_t nIataDcpts;
  uint16_t nFreeDcpts;
  uint32_t defaultIanaT1;
  uint32_t defaultIanaT2;
  uint32_t defaultIataT1;
  uint32_t defaultIataT2;
  uint32_t ianaSolicitT1;
  uint32_t ianaSolicitT2;
  uint32_t solicitPrefLTime;
  uint32_t solicitValidLTime;
  int nMsgBuffers;
  int msgBufferSize;
} TCPIP_DHCPV6_MODULE_CONFIG;
```

Members

Members	Description
TCPIP_DHCPV6_CONFIG_FLAGS configFlags;	DHCPv6 client configuration flags
uint16_t dhcpCliPort;	client port for DHCPv6 client transactions
uint16_t dhcpSrvPort;	remote server port for DHCPv6 server messages
uint16_t duidType;	TCPIP_DHCPV6_DUID_TYPE: type to use for the DHCPv6 clients
uint16_t nIanaDcpts;	number of IANAs per client; default should be 1
uint16_t nIataDcpts;	number of IATAs per client; default should be 0
uint16_t nFreeDcpts;	number of free IAs per client - they could be added at run time; default should be 0
uint32_t defaultIanaT1;	The default time at which the client contacts the server to extend the lifetimes of the assigned IA_NA addresses If the IANA t1 value received from the server is 0, then this value will be used to override A value of 0 means the t1 is infinite default value should be 0
uint32_t defaultIanaT2;	The default time at which the client contacts any available server to extend the lifetimes of the assigned IA_NA addresses If the IANA t2 value received from the server is 0, then this value will be used to override if !0 it should be > defaultIanaT1! Has to be > t1 A value of 0 means the t2 is infinite default value should be 0
uint32_t defaultIataT1;	The default time at which the client contacts the server to extend the lifetimes of the assigned IATA addresses If 0, the timeout will be infinite (0xffffffff)
uint32_t defaultIataT2;	The default time at which the client contacts any available server to extend the lifetimes of the assigned IA_TA addresses if !0 it should be > defaultIataT1! If 0, the timeout will be infinite (0xffffffff)
uint32_t ianaSolicitT1;	The default T1 time to solicit from the server default should be 0
uint32_t ianaSolicitT2;	The default T2 time to solicit from the server default should be 0
uint32_t solicitPrefLTime;	default addresses preferred lifetime to solicit from the server default should be 0
uint32_t solicitValidLTime;	default addresses valid lifetime to solicit from the server default should be 0
int nMsgBuffers;	number of message buffers to allocate for this client these buffers are used for the TX/RX operations Enough buffers need to be allocated for gathering server advertisements and being able to transmit messages default should be 4
int msgBufferSize;	size of the message buffers default is 512 bytes

Description

DHCPv6 module configuration

TCPIP_DHCPV6_SERVER_STATUS_CODE Enumeration

File

dhcpv6.h

C

```
typedef enum {
  TCPIP_DHCPV6_SERVER_STAT_SUCCESS = 0,
  TCPIP_DHCPV6_SERVER_STAT_UNSPEC_FAIL = 1,
  TCPIP_DHCPV6_SERVER_STAT_NO_ADDRS_AVAIL = 2,
  TCPIP_DHCPV6_SERVER_STAT_NO_BINDING = 3,
  TCPIP_DHCPV6_SERVER_STAT_NOT_ON_LINK = 4,
  TCPIP_DHCPV6_SERVER_STAT_USE_MULTICAST = 5,
  TCPIP_DHCPV6_SERVER_STAT_MAX_CODE = 5,
  TCPIP_DHCPV6_SERVER_STAT_EXT_ERROR = -1
} TCPIP_DHCPV6_SERVER_STATUS_CODE;
```

Members

Members	Description
TCPIP_DHCPV6_SERVER_STAT_SUCCESS = 0	success
TCPIP_DHCPV6_SERVER_STAT_UNSPEC_FAIL = 1	Failure, reason unspecified; this status code is sent by either a client or a server to indicate a failure not explicitly specified in the RFC.
TCPIP_DHCPV6_SERVER_STAT_NO_ADDRS_AVAIL = 2	Server has no addresses available to assign to the IA(s).
TCPIP_DHCPV6_SERVER_STAT_NO_BINDING = 3	Client record (binding) unavailable.
TCPIP_DHCPV6_SERVER_STAT_NOT_ON_LINK = 4	The prefix for the address is not appropriate for the link to which the client is attached.

TCPIP_DHCPV6_SERVER_STAT_USE_MULTICAST = 5	Sent by a server to a client to force the client to send messages to the server using the All_DHCP_Relay_Agents_and_Servers address.
TCPIP_DHCPV6_SERVER_STAT_MAX_CODE = 5	maximum valid value
TCPIP_DHCPV6_SERVER_STAT_EXT_ERROR = -1	an error occurred, status code not found, etc.

Description

DHCPV6 server status code

Files

Files

Name	Description
dhcpv6.h	Dynamic Host Configuration Protocol (DHCPv6) client API definitions.
dhcpv6_config.h	DCHPv6 configuration file

Description

This section lists the source and header files used by the library.

dhcpv6.h

Dynamic Host Configuration Protocol (DHCPv6) client API definitions.

Enumerations

	Name	Description
	TCPIP_DHCPV6_CLIENT_STATE	DHCPv6 Current Status
	TCPIP_DHCPV6_CONFIG_FLAGS	DHCPV6 start up flags
	TCPIP_DHCPV6_DUID_TYPE	types of DUID for DHCPv6
	TCPIP_DHCPV6_IA_STATE	IA run states
	TCPIP_DHCPV6_IA_SUBSTATE	IA run substates most IA run states that must send a message go through these substates
	TCPIP_DHCPV6_IA_TYPE	supported types of IA
	TCPIP_DHCPV6_SERVER_STATUS_CODE	DHCPV6 server status code

Functions

	Name	Description
	TCPIP_DHCPV6_ClientInfoGet	client status reporting
	TCPIP_DHCPV6_HandlerDeRegister	DHCPV6 event deregistration
	TCPIP_DHCPV6_HandlerRegister	DHCPV6 event registration
	TCPIP_DHCPV6_IaInfoGet	IA status reporting
	TCPIP_DHCPV6_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_DHCPV6_CLIENT_INFO	DHCPv6 client info
	TCPIP_DHCPV6_IA_INFO	DHCPv6 IA info
	TCPIP_DHCPV6_MODULE_CONFIG	DHCPv6 module configuration

Types

	Name	Description
	TCPIP_DHCPV6_EVENT_HANDLER	DHCPv6 event handler prototype.
	TCPIP_DHCPV6_HANDLE	a DHCPV6 handle

Unions

	Name	Description
	TCPIP_DHCPV6_IA_EVENT	IA event info

Description

IPv6 Dynamic Host Configuration Protocol (DHCPv6) Client API Header File

Provides automatic IP address, subnet mask, gateway address, DNS server address, and other configuration parameters on DHCPv6 enabled networks.

File Name

dhcpv6.h

Company

Microchip Technology Inc.

dhcpv6_config.h

DCHPv6 configuration file

Macros

	Name	Description
	TCPIP_DHCPV6_CLIENT_DUID_TYPE	Default DUID type to be used by the client
	TCPIP_DHCPV6_CLIENT_PORT	Clients listen for DHCP messages on UDP port:
	TCPIP_DHCPV6_DNS_SERVERS_NO	number of DNS servers to store from a DHCP server reply
	TCPIP_DHCPV6_DOMAIN_SEARCH_LIST_SIZE	space for the Domain Search List option - multiple of 16
	TCPIP_DHCPV6_FORCED_SERVER_PREFERENCE	preference value that forces the server selection 8 bit value!
	TCPIP_DHCPV6_IA_FREE_DESCRIPTORS_NO	maximum number of free IA descriptors per client
	TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_PREF_LTIME	default lifetimes for the solicited addresses
	TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME	This is macro TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME.
	TCPIP_DHCPV6_IANA_DEFAULT_T1	default values for IANA T1, T2
	TCPIP_DHCPV6_IANA_DEFAULT_T2	This is macro TCPIP_DHCPV6_IANA_DEFAULT_T2.
	TCPIP_DHCPV6_IANA_DESCRIPTORS_NO	maximum number of IANA descriptors per client
	TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO	default addresses for IANA in a solicit message
	TCPIP_DHCPV6_IANA_SOLICIT_DEFAULT_ADDRESS	default values for the IANA Solicit addresses irrelevant if TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO == 0 should be a value for each TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO
	TCPIP_DHCPV6_IANA_SOLICIT_T1	default values for IANA T1, T2 in a solicit message
	TCPIP_DHCPV6_IANA_SOLICIT_T2	This is macro TCPIP_DHCPV6_IANA_SOLICIT_T2.
	TCPIP_DHCPV6_IATA_DEFAULT_T1	default values for IATA T1, T2 If 0, the timeout will be infinite (0xffffffff)
	TCPIP_DHCPV6_IATA_DEFAULT_T2	This is macro TCPIP_DHCPV6_IATA_DEFAULT_T2.
	TCPIP_DHCPV6_IATA_DESCRIPTORS_NO	maximum number of IATA descriptors per client
	TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO	default addresses for IATA in a solicit message
	TCPIP_DHCPV6_IATA_SOLICIT_DEFAULT_ADDRESS	default values for the IANA Solicit addresses irrelevant if TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO == 0 should be a value for each TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO
	TCPIP_DHCPV6_MESSAGE_BUFFER_SIZE	default value of the buffer to assemble messages, etc.
	TCPIP_DHCPV6_MESSAGE_BUFFERS	default number of buffers
	TCPIP_DHCPV6_MIN_UDP_TX_BUFFER_SIZE	minimum size of a UDP buffer
	TCPIP_DHCPV6_SERVER_PORT	Servers and relay agents listen for DHCP messages on UDP port:
	TCPIP_DHCPV6_SKIP_DAD_PROCESS	defining this symbol will skip the DAD processing for DHCPv6 generated addresses
	TCPIP_DHCPV6_STATUS_CODE_MESSAGE_LEN	number of character to reserve for a server status code associated message
	TCPIP_DHCPV6_TASK_TICK_RATE	The DHCPv6 task processing rate: number of milliseconds to generate an DHCPv6 tick. Used by the DHCPv6 state machine The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DHCPV6_USER_NOTIFICATION	allow DHCPV6 client user notification if enabled, the TCPIP_DHCPV6_HandlerRegister/TCPIP_DHCPV6_HandlerDeRegister functions exist and can be used

	TCPIP_STACK_USE_DHCPV6_CLIENT	This symbol enables the DHCPv6 client as being part of the build

Description

IPv6 Dynamic Host Configuration Protocol (DCHPv6) Configuration file

This file contains the DCHPv6 module configuration options

File Name

dhcpv6_config.h

Company

Microchip Technology Inc.

DNS Module

This section describes the TCP/IP Stack Library DNS module.

Introduction

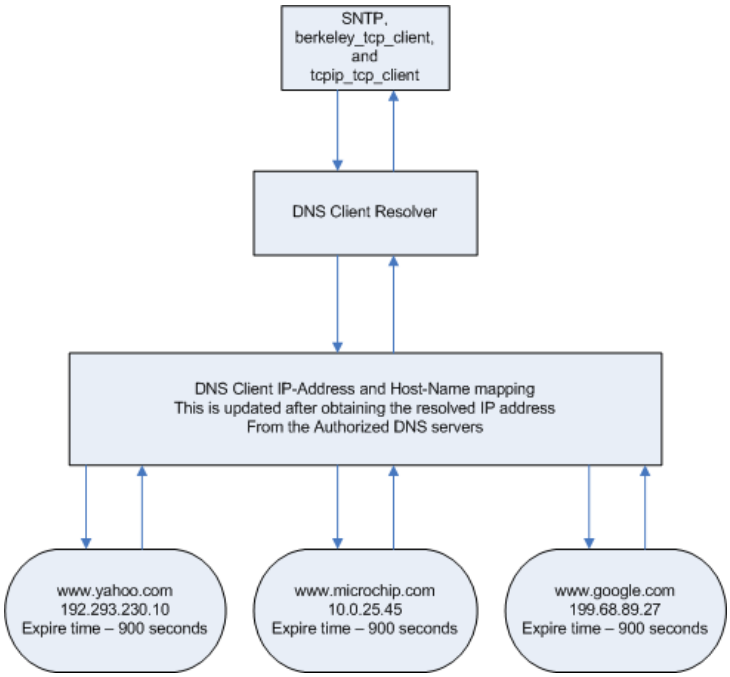
TCP/IP Stack Library Domain Name System (DNS) Module for Microchip Microcontrollers

This library provides the API of the DNS module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The Domain Name Service associates host names (i.e., www.microchip.com) with IP addresses (i.e., 10.0.54.2). The DNS Client module provides DNS resolution capabilities to the stack. As illustrated in the figure, a typical resolution process is as follows:

- The DNS Client, also known as the DNS Resolver, will try to resolve the IP address from its local resources, which includes the Name to Address mappings in the local DNS cache. DNS clients are capable of caching the previous DNS query results and this module supports maximum cache entries as per the configuration parameter of DNS_CLIENT_CACHE_ENTRIES. Each cache entry is capable of storing the maximum number of IPv4 and IPv6 address, which is done per the configured parameters of DNS_CLIENT_CACHE_PER_IPV4_ADDRESS and DNS_CLIENT_CACHE_PER_IPV6_ADDRESS. The DNS resolver will use these available local resources and if it cannot find the IP Address as per DNS Record type, it will then query a DNS server that it knows for the IP Address.
- The DNS server will directly answer the query if it is the authoritative server for the particular domain (ex.: www.microchip.com); otherwise, it will check its local cache of the previous queries. If it cannot find the IP Address, it will then query the DNS servers on the Internet.
- Next, the DNS Server will query one of the root servers requesting a list of Authoritative servers for the .COM domain. The Root server will respond with the list of servers addresses hosting the .COM domain.
- The Authoritative DNS server will then respond with the IP Address or a list of IP Address (if more than one server hosts the website www.microchip.com) for www.microchip.com.
- The address that the DNS server returns back to the DNS client is then passed to the DNS application, which initially requested the IP Address information -- in this case, the TCP/IP client application.
- Finally, the Application then knows where to go to fetch the web site.



Using the Library

This topic describes the basic architecture of the DNS TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `dns.h`

The interface to the DNS TCP/IP Stack library is defined in the `dns.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the DNS TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the DNS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

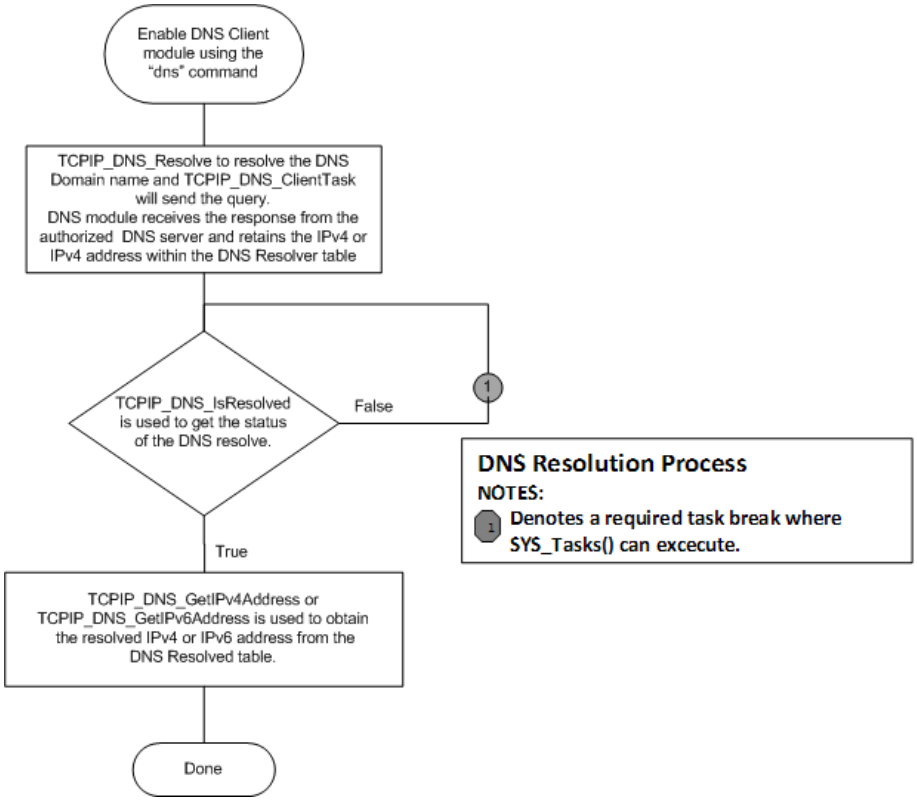
Description

DNS is part of the Application Layer. How the DNS Client works and how it is used is described in the following algorithm and figure.

1. By default, the DNS client is enabled and the DNS module can be enabled from the command prompt using the command "dns PIC32INT on". If the DNS Client module is enabled, a UDP socket will opened for port number 53 for the DNS client applications. The DNS client module can be disabled from the command prompt using "dns PIC32INT off".

   Note: For the Wi-Fi module, the interface name "PIC32INT" will be changed to "MRF24W".

2. Applications or demonstrations such as berkeley_tcp_client, berkeley_udp_client, tcpip_tcp_client, and tcpip_udp_client, use the TCPIP_DNS_Resolve function with Host name and Record type as arguments to resolve the domain host name. The DNS module supports the Record type of "A" and "AAAA". Record Type "A" is used for IPv4 and "AAAA" is used for an IPv6 address.
3. The resolved domain name and the mapped IPv4 or IPv6 address will be stored in a table with the expire time, which is calculated from the query response.
4. The number of the IPv4 and IPv6 address will be cached as per the DNS_CLIENT_CACHE_PER_IPV4_ADDRESS function and per the DNS_CLIENT_CACHE_PER_IPV6_ADDRESS function from the query response.
5. The TCPIP_DNS_IsResolved function will return DNS_RES_OK if the domain name resolve is successful.
6. The TCPIP_DNS_GetNumberOfIPAddresses function is used to obtain the number of the IPv4 or IPv6 address, which is present in the DNS Resolver table. The DNS Resolver table is a mapping table of the IP address, domain name, and the expire time.
7. The functions, TCPIP_DNS_GetIPv4Address and TCPIP_DNS_GetIPv6Address, are used to obtain the IPv4 or Ipv6 address from the resolved table as per the domain host name and index number. The index number should be less than DNS_CLIENT_CACHE_PER_IPV4_ADDRESS or DNS_CLIENT_CACHE_PER_IPV6_ADDRESS. The index number is the location of the resolved table, which will help to fetch the IPv4 or IPv6 address from the exact location. For example, if index == 0, TCPIP_DNS_GetIPv4Address or TCPIP_DNS_GetIPv6Address will return 0th location IPv4 or Ipv6 address if there is more than one IPv4 or IPv6 address.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DNS module.

Library Interface Section	Description
General Functions	This section provides general interface routines to DNS
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_DNS_CLIENT_CACHE_ENTRIES	Number of DNS resolver entries
	TCPIP_DNS_CLIENT_CACHE_ENTRY_TMO	DNS client cache entry time-out. If this symbol is zero then the entry time-out will be the one specified by the DNS server when the name was solved. Otherwise this value will be used as the cache entry time-out. Default should be 0.
	TCPIP_DNS_CLIENT_CACHE_PER_IPV4_ADDRESS	Maximum and default number of IPv4 answers to be considered while processing DNS response from server for a query.
	TCPIP_DNS_CLIENT_CACHE_PER_IPV6_ADDRESS	Maximum and default number of IPv6 answers to be considered while processing DNS response from server for a query.
	TCPIP_DNS_CLIENT_SERVER_TMO	When the DNS Client connected to the DNS Server this is the elapsed time after which an the communication is considered to have timed failed if there was no reply from the server In seconds
	TCPIP_DNS_CLIENT_TASK_PROCESS_RATE	DNS Client task processing rate, in milliseconds. The DNS Client module will process a timer event with this rate for processing its own state machine, etc. The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DNS_CLIENT_CACHE_DEFAULT_TTL_VAL	Default TTL time for a solved entry in the cache This value will be used when the DNS server TTL value for an entry is 0

	TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO	Time-out for the a unsolved name, in seconds. The name resolution will be aborted if the TMO elapsed and the name could not be solved Should be greater than TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO
	TCPIP_DNS_CLIENT_USER_NOTIFICATION	allow DNS client user notification if enabled, the TCPIP_DNS_HandlerRegister/TCPIP_DNS_HandlerDeRegister functions exist and can be used
	TCPIP_DNS_CLIENT_ADDRESS_TYPE	This parameter can be used to choose ithe type of IP connection for the DNS client: IPv4 or IPv6. Currently only IPv4 is supported and this parameter is not used. Reserved for future development
	TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO	Retry lookup for a unsolved entry in the cache, in seconds. If the TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO seconds elapsed and the name has not been solved then the name entry will be marked with server timeout and the resolution will be retried. Should be less than TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO
	TCPIP_DNS_CLIENT_MAX_HOSTNAME_LEN	Max DNS host name size
	TCPIP_DNS_CLIENT_MAX_SELECT_INTERFACES	Max number of interfaces to take part in the DNS selection algorithm Should be always greater than 1: • the default interface should always be considered for DNS resolution Depending on how many active interfaces select those to be considered for DNS resolution

Description

The configuration of the DNS TCP/IP Stack is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the DNS TCP/IP Stack. Based on the selections made, the DNS TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the DNS TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_DNS_CLIENT_CACHE_ENTRIES Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_CACHE_ENTRIES 5
```

Description

Number of DNS resolver entries

TCPIP_DNS_CLIENT_CACHE_ENTRY_TMO Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_CACHE_ENTRY_TMO 0
```

Description

DNS client cache entry time-out. If this symbol is zero then the entry time-out will be the one specified by the DNS server when the name was solved. Otherwise this value will be used as the cache entry time-out. Default should be 0.

TCPIP_DNS_CLIENT_CACHE_PER_IPV4_ADDRESS Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_CACHE_PER_IPV4_ADDRESS 5
```

Description

Maximum and default number of IPv4 answers to be considered while processing DNS response from server for a query.

TCPIP_DNS_CLIENT_CACHE_PER_IPV6_ADDRESS Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_CACHE_PER_IPV6_ADDRESS 1
```

Description

Maximum and default number of IPv6 answers to be considered while processing DNS response from server for a query.

TCPIP_DNS_CLIENT_SERVER_TMO Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_SERVER_TMO (1*60)
```

Description

When the DNS Client connected to the DNS Server this is the elapsed time after which an the communication is considered to have timed failed if there was no reply from the server In seconds

TCPIP_DNS_CLIENT_TASK_PROCESS_RATE Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_TASK_PROCESS_RATE (200)
```

Description

DNS Client task processing rate, in milliseconds. The DNS Client module will process a timer event with this rate for processing its own state machine, etc. The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_DNS_CLIENT_CACHE_DEFAULT_TTL_VAL Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_CACHE_DEFAULT_TTL_VAL (20 * 60)
```

Description

Default TTL time for a solved entry in the cache This value will be used when the DNS server TTL value for an entry is 0

TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO (10)
```

Description

Time-out for the a unsolved name, in seconds. The name resolution will be aborted if the TMO elapsed and the name could not be solved Should be greater than TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO

TCPIP_DNS_CLIENT_USER_NOTIFICATION Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_USER_NOTIFICATION false
```

Description

allow DNS client user notification if enabled, the TCPIP_DNS_HandlerRegister/TCPIP_DNS_HandlerDeRegister functions exist and can be used

TCPIP_DNS_CLIENT_ADDRESS_TYPE Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_ADDRESS_TYPE IP_ADDRESS_TYPE_IPV4
```

Description

This parameter can be used to choose ithe type of IP connection for the DNS client: IPv4 or IPv6. Currently only IPv4 is supported and this parameter is not used. Reserved for future development

TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO (3)
```

Description

Retry lookup for a unsolved entry in the cache, in seconds. If the TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO seconds elapsed and the name has not been solved then the name entry will be marked with server timeout and the resolution will be retried. Should be less than TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO

TCPIP_DNS_CLIENT_MAX_HOSTNAME_LEN Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_MAX_HOSTNAME_LEN 32
```

Description

Max DNS host name size

TCPIP_DNS_CLIENT_MAX_SELECT_INTERFACES Macro

File

dns_config.h

C

```
#define TCPIP_DNS_CLIENT_MAX_SELECT_INTERFACES 4
```

Description

Max number of interfaces to take part in the DNS selection algorithm Should be always greater than 1:

- the default interface should always be considered for DNS resolution

Depending on how many active interfaces select those to be considered for DNS resolution

Building the Library

This section lists the files that are available in the DNS module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dns.c`	DNS implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The DNS module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Library Interface

a) General Functions

	Name	Description
	TCPIP_DNS_Enable	Enables the DNS Client for the specified interface.
	TCPIP_DNS_Disable	Disables the DNS Client for the specified interface.
	TCPIP_DNS_HandlerDeRegister	Deregisters a previously registered DNS client handler.
	TCPIP_DNS_HandlerRegister	Registers a DNS client Handler.
	TCPIP_DNS_IsEnabled	Determines if the DNS client is enabled on that specified interface.
	TCPIP_DNS_IsNameResolved	Determines if the DNS resolution is complete and provides the host IP address.
	TCPIP_DNS_RemoveAll	Removes all the cached entries from DNS resolver.
	TCPIP_DNS_RemoveEntry	Remove a hostname from the DNS Hash entry.
	TCPIP_DNS_ClientTask	Standard TCP/IP stack module task function.
	TCPIP_DNS_ClientInfoGet	Get the current DNS client parameters.
	TCPIP_DNS_EntryQuery	Queries a DNS Resolver specific entry.
	TCPIP_DNS_GetIPAddressesNumber	Get the count of resolved IPv4 and/or IPv6 address for a host name.

	TCPIP_DNS_GetIPv4Addresses	Get IPV4 addresses for a DNS resolved name.
	TCPIP_DNS_GetIPv6Addresses	Get IPV6 addresses for a DNS resolved name.
	TCPIP_DNS_Resolve	Begins resolution of an address.
	TCPIP_DNS_Send_Query	Forces resolution of an address.
	TCPIP_DNS_IsResolved	Determines if the DNS resolution is complete and provides the host IP address.

b) Data Types and Constants

	Name	Description
	TCPIP_DNS_EVENT_HANDLER	Notification handler that can be called when a specific entry is resolved and entry is timed out.
	TCPIP_DNS_EVENT_TYPE	This enumeration is used to notify DNS client applications.
	TCPIP_DNS_HANDLE	DNS client handle.
	TCPIP_DNS_RESULT	DNS client result codes.
	TCPIP_DNS_CLIENT_MODULE_CONFIG	Provides a place holder for DNS client configuration.
	TCPIP_DNS_CLIENT_INFO	General DNS client info.
	TCPIP_DNS_ENTRY_QUERY	DNS module query data for both IPv4 and IPv6 .
	TCPIP_DNS_ENABLE_FLAGS	Flags for enabling the DNS service on an interface.
	TCPIP_DNS_RESOLVE_TYPE	DNS query record type.

Description

This section describes the Application Programming Interface (API) functions of the DNS module.

Refer to each section for a detailed description.

a) General Functions

TCPIP_DNS_Enable Function

Enables the DNS Client for the specified interface.

File

dns.h

C

```
bool TCPIP_DNS_Enable(TCPIP_NET_HANDLE hNet, TCPIP_DNS_ENABLE_FLAGS flags);
```

Returns

- true - if successful
- false - if unsuccessful: the requested interface could not be selected for DNS name resolving.

Description

This function enables the DNS Client name resolution for the specified interface. The additional flags give better control on how the name resolution is performed.

Remarks

The interface selection for the name resolution tries to find a valid interface, i.e. an interface that is up and has a valid DNS server. The selection is done following these rules:

- if a strict interface is set, only that interface is used for name resolution
- else if there is a preferred interface, that one will be tried first
- else the default interface is used
- else any available interface will be used

Additionally, if a retry is attempted using the same selected interface, an alternate DNS server from that interface will be selected, if available.

Only one strict interface can exist at any time. Selecting a new strict interface will replace the old one.

Only one preferred interface can exist at any time. Selecting a new preferred interface will replace the old one.

The selected interface has to be up and running for the call to succeed

Preconditions

The DNS module must be initialized.

Parameters

Parameters	Description
hNet	Interface to enable the DNS Client on
flags	specify further attributes for this interface: act as a strict, preferred or default interface

Function

bool TCPIP_DNS_Enable(TCPIP_NET_HANDLE hNet, TCPIP_DNS_ENABLE_FLAGS flags)

TCPIP_DNS_Disable Function

Disables the DNS Client for the specified interface.

File

dns.h

C

```
bool TCPIP_DNS_Disable(TCPIP_NET_HANDLE hNet, bool clearCache);
```

Returns

- true - if successful
- false - if unsuccessful

Description

This function disables the DNS Client for the specified interface.

Remarks

When the DNS client is disabled on a requested interface the previously solved names will still be part of the cache and will expire when their timeout occurs. If the TTL for a name sent by the DNS server was ignored and another default/arbitrary value was used, then the entry will stay cached until that timeout occurs (i.e. timeout not specified by the DNS server). To avoid this, you can clear the cache by setting the clearCache parameter to true.

If the disabled interface matches the strict interface set by TCPIP_DNS_Enable this function will set the strict interface to 0.

If the disabled interface matches the preferred interface set by TCPIP_DNS_Enable this function will set the preferred interface to 0.

Preconditions

The DNS module must be initialized.

Parameters

Parameters	Description
hNet	Interface for which to disable the DNS Client.
clearCache	If true, all the existent name entries will be cleared from the cache

Function

bool TCPIP_DNS_Disable(TCPIP_NET_HANDLE hNet, bool clearCache)

TCPIP_DNS_HandlerDeRegister Function

Deregisters a previously registered DNS client handler.

File

dns.h

C

```
bool TCPIP_DNS_HandlerDeRegister(TCPIP_DNS_HANDLE hDns);
```

Returns

- true - if the call succeeds
- false - if no such handler is registered

Description

This function deregisters the DNS client event handler.

Preconditions

The DNS module must be initialized.

Parameters

Parameters	Description
hDns	A handle returned by a previous call to TCPIP_DNS_HandlerRegister.

Function

bool TCPIP_DNS_HandlerDeRegister(TCPIP_DNS_HANDLE hDns);

TCPIP_DNS_HandlerRegister Function

Registers a DNS client Handler.

File

dns.h

C

```
TCPIP_DNS_HANDLE TCPIP_DNS_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_DNS_EVENT_HANDLER handler, const void* hParam);
```

Returns

- Returns a valid handle if the call succeeds
- Returns null handle if the call failed (out of memory, for example)

Description

This function registers a DNS client event handler. The DNS client module will call the registered handler when a DNS client event (TCPIP_DNS_EVENT_TYPE) occurs.

Remarks

The handler has to be short and fast. It is meant for setting an event flag, *not* for lengthy processing!

Preconditions

The DNS module must be initialized.

Parameters

Parameters	Description
hNet	Interface handle. Use hNet == 0 to register on all interfaces available.
handler	Handler to be called when a DNS client event occurs.
hParam	Pointer to non-volatile ASCIIZ string to be used in the handler call. It is used as a domain/host name. If not NULL, a DNS notification will be delivered only for a name resolution that matches the hParam. If the hParam == 0, then the notification is triggered for any host name resolution.

Function

TCPIP_DNS_HANDLE

TCPIP_DNS_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_DNS_EVENT_HANDLER handler, const void* hParam);

TCPIP_DNS_IsEnabled Function

Determines if the DNS client is enabled on that specified interface.

File

dns.h

C

```
bool TCPIP_DNS_IsEnabled(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if the DNS client service is enabled on the specified interface
- false - if the DNS client service is not enabled on the specified interface

Description

This function returns the current state of DNS Client on the specified interface.

Preconditions

The DNS module must be initialized.

Parameters

Parameters	Description
hNet	Interface to query.

Function

bool TCPIP_DNS_IsEnabled(CPIP_NET_HANDLE hNet)

TCPIP_DNS_IsNameResolved Function

Determines if the DNS resolution is complete and provides the host IP address.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_IsNameResolved(const char* hostName, IPV4_ADDR* hostIPv4, IPV6_ADDR* hostIPv6);
```

Description

Call this function to determine if the DNS name resolution has been completed. This function allows for retrieval of separate IPv4 and IPv6 addresses for a name.

Remarks

The function will set either an IPv6 or an IPv4 address to the hostIP address, depending on what's available.

Preconditions

TCPIP_DNS_Resolve has been called.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host for which to resolve an IP.
hostIPv4	A pointer to an IPV4_ADDR structure in which to store the resolved IPv4 address if resolution is complete. Could be NULL if not needed.
hostIPv6	A pointer to an IPV6_ADDR structure in which to store the resolved IPv6 address if resolution is complete. Could be NULL if not needed.

Function

TCPIP_DNS_RESULT TCPIP_DNS_IsNameResolved(const char* hostName, IPV4_ADDR* hostIPv4, IPV6_ADDR* hostIPv6);

TCPIP_DNS_RemoveAll Function

Removes all the cached entries from DNS resolver.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_RemoveAll();
```

Returns

- TCPIP_DNS_RES_OK - If successful
- TCPIP_DNS_RES_NO_SERVICE - DNS resolver non existent/uninitialized.

Description

This function is used to remove all the entries from the DNS cache. It removes both the solved and unresolved entries.

Remarks

None.

Preconditions

The DNS module must be initialized.

Function

TCPIP_DNS_RESULT TCPIP_DNS_RemoveAll(void)

TCPIP_DNS_RemoveEntry Function

Remove a hostname from the DNS Hash entry.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_RemoveEntry(const char * hostName);
```

Returns

- TCPIP_DNS_RES_OK - If name was successfully removed
- TCPIP_DNS_RES_INVALID_HOSTNAME - invalid name supplied
- TCPIP_DNS_RES_NO_SERVICE - DNS resolver non existent/uninitialized.
- TCPIP_DNS_RES_NO_NAME_ENTRY - no such name exists

Description

This function is used to remove an entry (host name) from the DNS cache.

Remarks

None.

Preconditions

The DNS module must be initialized.

Parameters

Parameters	Description
hostName	Domain name to be inserted.

Function

TCPIP_DNS_RESULT TCPIP_DNS_RemoveEntry(const char *hostName)

TCPIP_DNS_ClientTask Function

Standard TCP/IP stack module task function.

File

dns.h

C

```
void TCPIP_DNS_ClientTask();
```

Returns

None.

Description

This function performs DNS module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The DNS module should have been initialized.

Function

void TCPIP_DNS_ClientTask(void)

TCPIP_DNS_ClientInfoGet Function

Get the current DNS client parameters.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_ClientInfoGet(TCPIP_DNS_CLIENT_INFO* pClientInfo);
```

Returns

- TCPIP_DNS_RES_OK on success
- TCPIP_DNS_RES_NO_SERVICE - DNS resolver non existent/uninitialized.

Description

This function is used to get the current settings of the DNS client.

Remarks

None

Preconditions

The DNS client module must be initialized.

Parameters

Parameters	Description
pClientInfo	pointer to a TCPIP_DNS_CLIENT_INFO data structure to receive the DNS client information.

Function

TCPIP_DNS_RESULT TCPIP_DNS_ClientInfoGet(TCPIP_DNS_CLIENT_INFO* pClientInfo)

TCPIP_DNS_EntryQuery Function

Queries a DNS Resolver specific entry.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_EntryQuery(TCPIP_DNS_ENTRY_QUERY * pDnsQuery, int queryIndex);
```

Returns

- TCPIP_DNS_RES_OK - valid address for this index, successful

Errors:

- TCPIP_DNS_RES_NO_SERVICE - DNS resolver non existent/uninitialized.
- TCPIP_DNS_RES_INVALID_HOSTNAME - invalid string, len, pDnsQuery provided
- TCPIP_DNS_RES_EMPTY_IX_ENTRY - no name associated with this entry
- TCPIP_DNS_RES_NO_IX_ENTRY - invalid query index

Description

This function is used to query the DNS client for a specified entry. The entry to be queried is selected by its index.

Remarks

None

Preconditions

The DNS client module must be initialized.

Parameters

Parameters	Description
pDnsQuery	address to store the the query result
queryIndex	entry index to be selected; should start with 0

Function

TCPIP_DNS_RESULT TCPIP_DNS_EntryQuery(TCPIP_DNS_ENTRY_QUERY *pDnsQuery, int queryIndex);

TCPIP_DNS_GetIPAddressesNumber Function

Get the count of resolved IPv4 and/or IPv6 address for a host name.

File

dns.h

C

```
int TCPIP_DNS_GetIPAddressesNumber(const char* hostName, IP_ADDRESS_TYPE type);
```

Description

This function returns the total count of IPv4 and/or IPv6 addresses that exist for a resolved host name.

Remarks

None.

Preconditions

TCPIP_DNS_Resolve has been called.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host name
type	IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6/IP_ADDRESS_TYPE_ANY

Function

int TCPIP_DNS_GetIPAddressesNumber(const char* hostName, IP_ADDRESS_TYPE type)

TCPIP_DNS_GetIPv4Addresses Function

Get IPV4 addresses for a DNS resolved name.

File

dns.h

C

```
int TCPIP_DNS_GetIPv4Addresses(const char* hostName, int startIndex, IPV4_ADDR* pIPv4Addr, int
nIPv4Addresses);
```

Returns

> 0 - the number of addresses copied to the pIPv4Addr array 0 - if the host name was not found, invalid index, bad parameter, etc.

Description

This function will return IPv4 addresses for a host name if the DNS resolution has been completed.

Remarks

None.

Preconditions

TCPIP_DNS_Resolve has been called.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host name.
startIndex	starting index of the IP address to be returned when multiple addresses are available
The max number of addresses that can be stored for a host name is given by	TCPIP_DNS_CLIENT_MODULE_CONFIG::nIPv4Entries. The current number of valid entries for an address is given by TCPIP_DNS_GetIPAddressesNumber(). A valid index is [0, TCPIP_DNS_GetIPAddressesNumber(IP_ADDRESS_TYPE_IPV4));
pIPv4Addr	pointer to array of IPv4 addresses to store the host IPv4 addresses
nIPv4Addresses	number of IPv4 addresses in the pIPv4Addr array.

Function

int TCPIP_DNS_GetIPv4Addresses(const char* hostName, int startIndex, IPV4_ADDR* pIPv4Addr, int nIPv4Addresses);

TCPIP_DNS_GetIPv6Addresses Function

Get IPV6 addresses for a DNS resolved name.

File

dns.h

C

```
int TCPIP_DNS_GetIPv6Addresses(const char* hostName, int startIndex, IPV6_ADDR* pIPv6Addr, int
nIPv6Addresses);
```

Returns

> 0 - the number of addresses copied to the pIPv6Addr array 0 - if the host name was not found, invalid index, bad parameter, etc.

Description

This function will return IPv6 addresses for a host name if the DNS resolution has been completed.

Remarks

None.

Preconditions

TCPIP_DNS_Resolve has been called.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host name.
startIndex	starting index of the IP address to be returned when multiple addresses are available
The max number of addresses that can be stored for a host name is given by	TCPIP_DNS_CLIENT_MODULE_CONFIG::nIPv6Entries. The current number of valid entries for an address is given by TCPIP_DNS_GetIPAddressesNumber(). A valid index is [0, TCPIP_DNS_GetIPAddressesNumber(IP_ADDRESS_TYPE_IPV6));
pIPv6Addr	pointer to array of IPv6 addresses to store the host IPv6 addresses
nIPv6Addresses	number of IPv6 addresses in the pIPv6Addr array.

Function

int TCPIP_DNS_GetIPv6Addresses(const char* hostName, int startIndex, IPV6_ADDR* pIPv6Addr, int nIPv6Addresses);

TCPIP_DNS_Resolve Function

Begins resolution of an address.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_Resolve(const char* hostName, TCPIP_DNS_RESOLVE_TYPE type);
```

Returns

TCPIP_DNS_RES_OK - success, name is solved. TCPIP_DNS_RES_PENDING - operation is ongoing
TCPIP_DNS_RES_NAME_IS_IPADDRESS - name request is a IPv4 or IPv6 address

or an error code if an error occurred

Description

This function attempts to resolve a host name to an IP address. When called, it will attempt to send a DNS query to a DNS server for resolving the name. Call TCPIP_DNS_IsResolved to determine if name resolution is complete.

Remarks

To clear the cache use TCPIP_DNS_Disable(hNet, true);

Preconditions

DNS client module initialized.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host for which to resolve an IP.
type	a TCPIP_DNS_RESOLVE_TYPE value specifying the desired resolution

Function

TCPIP_DNS_RESULT TCPIP_DNS_Resolve(const char* hostName, TCPIP_DNS_RESOLVE_TYPE type)

TCPIP_DNS_Send_Query Function

Forces resolution of an address.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_Send_Query(const char* hostName, TCPIP_DNS_RESOLVE_TYPE type);
```

Returns

TCPIP_DNS_RES_OK - success, name is solved. TCPIP_DNS_RES_PENDING - operation is ongoing
TCPIP_DNS_RES_NAME_IS_IPADDRESS - name request is a IPv4 or IPv6 address

or an error code if an error occurred

Description

This function attempts to send a query packet for the supplied host name and DNS type.

Remarks

If the name is already part of the DNS resolution process (has been previously requested with TCPIP_DNS_Resolve or TCPIP_DNS_Send_Query) the function will force a new DNS query . If the name resolution is already solved and completed, this function will mark it as incomplete and a new response from the server will be requested.

If the name was not part of the DNS client resolution, then this function is equivalent to TCPIP_DNS_Resolve().

Preconditions

The DNS client module must be initialized.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host for which to resolve an IP.
type	a TCPIP_DNS_RESOLVE_TYPE value specifying the desired resolution

Function

TCPIP_DNS_RESULT TCPIP_DNS_Send_Query(const char* hostName, TCPIP_DNS_RESOLVE_TYPE type)

TCPIP_DNS_IsResolved Function

Determines if the DNS resolution is complete and provides the host IP address.

File

dns.h

C

```
TCPIP_DNS_RESULT TCPIP_DNS_IsResolved(const char* hostName, IP_MULTI_ADDRESS* hostIP, IP_ADDRESS_TYPE type);
```

Description

Call this function to determine if the DNS resolution of an address has been completed. If so, the resolved address will be provided in hostIP.

Remarks

The function will set either an IPv6 or an IPv4 address to the hostIP address, depending on what's available.

If type IP_ADDRESS_TYPE_ANY is specified the hostIP will be updated with the first available solved address: either IPv6 or IPv4!

Preconditions

TCPIP_DNS_Resolve has been called.

Parameters

Parameters	Description
hostName	A pointer to the null terminated string specifying the host for which to resolve an IP.
hostIP	A pointer to an IP_MULTI_ADDRESS structure in which to store the resolved IPv4/IPv6 address if resolution is complete. Could be NULL if not needed.
type	type of address needed: IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6/IP_ADDRESS_TYPE_ANY

Function

TCPIP_DNS_RESULT TCPIP_DNS_IsResolved(const char* hostName, IP_MULTI_ADDRESS* hostIP, IP_ADDRESS_TYPE type)

b) Data Types and Constants

TCPIP_DNS_EVENT_HANDLER Type

Notification handler that can be called when a specific entry is resolved and entry is timed out.

File

dns.h

C

```
typedef void (* TCPIP_DNS_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, TCPIP_DNS_EVENT_TYPE evType, const char*
name, const void* param);
```

Description

Type: TCPIP_DNS_EVENT_HANDLER

The format of a notification handler registered with the DNS module. Once an DNS event occurs the DNS service will be called for the registered handler.

Remarks

If pNetIf == 0, the notification is called for events on any interface.

Parameters

Parameters	Description
hNet	the interface on which the DNS event occurred
evType	the DNS reported event
name	the host name associated with the event
param	additional user parameter - see TCPIP_DNS_HandlerRegister

TCPIP_DNS_EVENT_TYPE Enumeration

This enumeration is used to notify DNS client applications.

File

dns.h

C

```
typedef enum {
  TCPIP_DNS_EVENT_NONE,
  TCPIP_DNS_EVENT_NAME_QUERY,
  TCPIP_DNS_EVENT_NAME_RESOLVED,
  TCPIP_DNS_EVENT_NAME_EXPIRED,
  TCPIP_DNS_EVENT_NAME_REMOVED,
  TCPIP_DNS_EVENT_NAME_ERROR,
  TCPIP_DNS_EVENT_SOCKET_ERROR,
  TCPIP_DNS_EVENT_NO_INTERFACE
} TCPIP_DNS_EVENT_TYPE;
```

Members

Members	Description
TCPIP_DNS_EVENT_NONE	DNS no event
TCPIP_DNS_EVENT_NAME_QUERY	DNS Query sent
TCPIP_DNS_EVENT_NAME_RESOLVED	DNS Name resolved
TCPIP_DNS_EVENT_NAME_EXPIRED	name entry expired
TCPIP_DNS_EVENT_NAME_REMOVED	name removed to make room for another entry
TCPIP_DNS_EVENT_NAME_ERROR	no such name reported by the DNS server
TCPIP_DNS_EVENT_SOCKET_ERROR	a DNS probe could not be sent, socket was busy, TX failed, etc.
TCPIP_DNS_EVENT_NO_INTERFACE	a DNS probe could not be sent, no DNS interface could be selected

Description

Enumeration: TCPIP_DNS_EVENT_TYPE

These events are used while notifying to the registered applications.

Remarks

None.

TCPIP_DNS_HANDLE Type

DNS client handle.

File

dns.h

C

```
typedef const void* TCPIP_DNS_HANDLE;
```

Description

Type: TCPIP_DNS_HANDLE

A handle that a application needs to use when deregistering a notification handler.

Remarks

This handle can be used by the application after the event handler has been registered.

TCPIP_DNS_RESULT Enumeration

DNS client result codes.

File

dns.h

C

```
typedef enum {
  TCPIP_DNS_RES_OK = 0,
  TCPIP_DNS_RES_PENDING,
  TCPIP_DNS_RES_NAME_IS_IPADDRESS,
  TCPIP_DNS_RES_NO_NAME_ENTRY = -1,
  TCPIP_DNS_RES_NO_IP_ENTRY = -2,
  TCPIP_DNS_RES_NO_IX_ENTRY = -3,
  TCPIP_DNS_RES_EMPTY_IX_ENTRY = -4,
  TCPIP_DNS_RES_SERVER_TMO = -5,
  TCPIP_DNS_RES_NO_SERVICE = -6,
  TCPIP_DNS_RES_NO_INTERFACE = -7,
  TCPIP_DNS_RES_CACHE_FULL = -8,
  TCPIP_DNS_RES_INVALID_HOSTNAME = -9,
  TCPIP_DNS_RES_SOCKET_ERROR = -10
} TCPIP_DNS_RESULT;
```

Members

Members	Description
TCPIP_DNS_RES_OK = 0	operation succeeded
TCPIP_DNS_RES_PENDING	operation is ongoing
TCPIP_DNS_RES_NAME_IS_IPADDRESS	DNS request is a IPv4 or IPv6 address
TCPIP_DNS_RES_NO_NAME_ENTRY = -1	no such name exists
TCPIP_DNS_RES_NO_IP_ENTRY = -2	no such IP type exists
TCPIP_DNS_RES_NO_IX_ENTRY = -3	no such index exists
TCPIP_DNS_RES_EMPTY_IX_ENTRY = -4	no entry associated to this index exists
TCPIP_DNS_RES_SERVER_TMO = -5	DNS server response tmo
TCPIP_DNS_RES_NO_SERVICE = -6	DNS service not implemented or uninitialized
TCPIP_DNS_RES_NO_INTERFACE = -7	no interface for DNS traffic available
TCPIP_DNS_RES_CACHE_FULL = -8	the cache is full and no entry could be added
TCPIP_DNS_RES_INVALID_HOSTNAME = -9	Invalid hostname
TCPIP_DNS_RES_SOCKET_ERROR = -10	DNS UDP socket error: not ready, TX error, etc.

Description

Enumeration: TCPIP_DNS_RESULT

DNS Client operations results.

Remarks

None.

TCPIP_DNS_CLIENT_MODULE_CONFIG Structure

Provides a place holder for DNS client configuration.

File

dns.h

C

```
typedef struct {
  bool deleteOldLease;
  int cacheEntries;
  uint32_t entrySolvedTmo;
  int nIPv4Entries;
  IP_ADDRESS_TYPE ipAddressType;
  int nIPv6Entries;
} TCPIP_DNS_CLIENT_MODULE_CONFIG;
```

Members

Members	Description
bool deleteOldLease;	Delete old cache if still in place, specific DNS parameters
int cacheEntries;	Max number of cache entries
uint32_t entrySolvedTmo;	Solved entry removed after this tmo if not referenced - seconds

int nIPv4Entries;	Number of IPv4 entries per DNS name and Default value is 1.
IP_ADDRESS_TYPE ipAddressType;	IP protocol type to use for connecting to the DNS server: IPv4 or IPv6 Currently only IPv4 is supported and this parameter is not used Reserved for future improvements
int nIPv6Entries;	Number of IPv6 address per DNS Name Default value is 1 and is used only when IPv6 is enabled

Description

Structure: TCPIP_DNS_CLIENT_MODULE_CONFIG

DNS module runtime configuration and initialization parameters.

Remarks

None.

TCPIP_DNS_CLIENT_INFO Structure

General DNS client info.

File

dns.h

C

```
typedef struct {
  TCPIP_NET_HANDLE strictNet;
  TCPIP_NET_HANDLE prefNet;
  uint32_t dnsTime;
  uint16_t pendingEntries;
  uint16_t currentEntries;
  uint16_t totalEntries;
} TCPIP_DNS_CLIENT_INFO;
```

Members

Members	Description
TCPIP_NET_HANDLE strictNet;	current strict DNS interface
TCPIP_NET_HANDLE prefNet;	current preferred DNS interface
uint32_t dnsTime;	current DNS time, seconds
uint16_t pendingEntries;	number of entries that need to be solved
uint16_t currentEntries;	number of solved and unslolved name entries
uint16_t totalEntries;	total number of supported name entries

Description

Structure: TCPIP_DNS_CLIENT_INFO

DNS module data structure used for getting information about the module settings.

Remarks

None.

TCPIP_DNS_ENTRY_QUERY Structure

DNS module query data for both IPv4 and IPv6 .

File

dns.h

C

```
typedef struct {
  char* hostName;
  int nameLen;
  IPV4_ADDR * ipv4Entry;
  int nIPv4Entries;
  IPV6_ADDR * ipv6Entry;
  int nIPv6Entries;
  TCPIP_DNS_RESULT status;
  uint32_t ttlTime;
```

```
  TCPIP_NET_HANDLE hNet;
  int serverIx;
  int nIPv4ValidEntries;
  int nIPv6ValidEntries;
} TCPIP_DNS_ENTRY_QUERY;
```

Members

Members	Description
char* hostName;	Pointer to a name to receive the host name for that particular entry
int nameLen;	hostName buffer size
IPV4_ADDR * ipv4Entry;	array of IPv4 entries/addresses to be populated
int nIPv4Entries;	number of entries in the ipv4Entry[] array;
IPV6_ADDR * ipv6Entry;	array of IPv6 entries/addresses to be populated
int nIPv6Entries;	number of entries in the ipv6Entry[] array;
TCPIP_DNS_RESULT status;	current status for this name: • TCPIP_DNS_RES_OK: name is resolved • TCPIP_DNS_RES_PENDING: name is pending • TCPIP_DNS_RES_SERVER_TMO: server timeout
uint32_t ttlTime;	time to live for a solved DNS entry
TCPIP_NET_HANDLE hNet;	interface the name was obtained or on which the query is currently ongoing
int serverIx;	index of the server used on that interface
int nIPv4ValidEntries;	number of valid entries written to the ipv4Entry
int nIPv6ValidEntries;	number of valid entries written to the ipv6Entry

Description

Structure: TCPIP_DNS_ENTRY_QUERY

DNS module uses this structure to return information about a resolved IPv4 and IPv6 address.

Remarks

None.

TCPIP_DNS_ENABLE_FLAGS Enumeration

Flags for enabling the DNS service on an interface.

File

dns.h

C

```
typedef enum {
  TCPIP_DNS_ENABLE_DEFAULT = 0,
  TCPIP_DNS_ENABLE_STRICT,
  TCPIP_DNS_ENABLE_PREFERRED
} TCPIP_DNS_ENABLE_FLAGS;
```

Members

Members	Description
TCPIP_DNS_ENABLE_DEFAULT = 0	the interface is capable of performing DNS name resolution
TCPIP_DNS_ENABLE_STRICT	only this interface will be used for DNS name resolution
TCPIP_DNS_ENABLE_PREFERRED	prefer this interface when doing DNS name resolution

Description

Enumeration: TCPIP_DNS_ENABLE_FLAGS

This enumeration lists the TCPIP_DNS_ENABLE_FLAGS argument for TCPIP_DNS_Enable.

Remarks

See the TCPIP_DNS_Enable description for details

TCPIP_DNS_RESOLVE_TYPE Enumeration

DNS query record type.

File

dns.h

C

```
typedef enum {
  TCPIP_DNS_TYPE_A = 1,
  TCPIP_DNS_TYPE_MX = 15,
  TCPIP_DNS_TYPE_AAAA = 28u,
  TCPIP_DNS_TYPE_ANY = 0xff
} TCPIP_DNS_RESOLVE_TYPE;
```

Members

Members	Description
TCPIP_DNS_TYPE_A = 1	Indicates an A (standard address) record.
TCPIP_DNS_TYPE_MX = 15	Indicates an MX (mail exchanger) record.
TCPIP_DNS_TYPE_AAAA = 28u	Indicates a quad-A (IPv6 address) address record.

Description

Enumeration: TCPIP_DNS_RESOLVE_TYPE

This enumeration lists the RecordType argument for TCPIP_DNS_Resolve. The stack supports DNS_TYPE_A and DNS_TYPE_AAAA.

Remarks

None.

Files

Files

Name	Description
dns.h	DNS definitions and interface file.
dns_config.h	DNS configuration file

Description

This section lists the source and header files used by the library.

dns.h

DNS definitions and interface file.

Enumerations

	Name	Description
	TCPIP_DNS_ENABLE_FLAGS	Flags for enabling the DNS service on an interface.
	TCPIP_DNS_EVENT_TYPE	This enumeration is used to notify DNS client applications.
	TCPIP_DNS_RESOLVE_TYPE	DNS query record type.
	TCPIP_DNS_RESULT	DNS client result codes.

Functions

	Name	Description
	TCPIP_DNS_ClientInfoGet	Get the current DNS client parameters.
	TCPIP_DNS_ClientTask	Standard TCP/IP stack module task function.
	TCPIP_DNS_Disable	Disables the DNS Client for the specified interface.
	TCPIP_DNS_Enable	Enables the DNS Client for the specified interface.
	TCPIP_DNS_EntryQuery	Queries a DNS Resolver specific entry.
	TCPIP_DNS_GetIPAddressesNumber	Get the count of resolved IPv4 and/or IPv6 address for a host name.
	TCPIP_DNS_GetIPv4Addresses	Get IPV4 addresses for a DNS resolved name.

	TCPIP_DNS_GetIPv6Addresses	Get IPV6 addresses for a DNS resolved name.
	TCPIP_DNS_HandlerDeRegister	Deregisters a previously registered DNS client handler.
	TCPIP_DNS_HandlerRegister	Registers a DNS client Handler.
	TCPIP_DNS_IsEnabled	Determines if the DNS client is enabled on that specified interface.
	TCPIP_DNS_IsNameResolved	Determines if the DNS resolution is complete and provides the host IP address.
	TCPIP_DNS_IsResolved	Determines if the DNS resolution is complete and provides the host IP address.
	TCPIP_DNS_RemoveAll	Removes all the cached entries from DNS resolver.
	TCPIP_DNS_RemoveEntry	Remove a hostname from the DNS Hash entry.
	TCPIP_DNS_Resolve	Begins resolution of an address.
	TCPIP_DNS_Send_Query	Forces resolution of an address.

Structures

	Name	Description
	TCPIP_DNS_CLIENT_INFO	General DNS client info.
	TCPIP_DNS_CLIENT_MODULE_CONFIG	Provides a place holder for DNS client configuration.
	TCPIP_DNS_ENTRY_QUERY	DNS module query data for both IPv4 and IPv6 .

Types

	Name	Description
	TCPIP_DNS_EVENT_HANDLER	Notification handler that can be called when a specific entry is resolved and entry is timed out.
	TCPIP_DNS_HANDLE	DNS client handle.

Description

Domain Name System (DNS) Client API Header file

This source file contains the DNS client module API.

File Name

dns.h

Company

Microchip Technology Inc.

dns_config.h

DNS configuration file

Macros

	Name	Description
	TCPIP_DNS_CLIENT_ADDRESS_TYPE	This parameter can be used to choose ithe type of IP connection for the DNS client: IPv4 or IPv6. Currently only IPv4 is supported and this parameter is not used. Reserved for future development
	TCPIP_DNS_CLIENT_CACHE_DEFAULT_TTL_VAL	Default TTL time for a solved entry in the cache This value will be used when the DNS server TTL value for an entry is 0
	TCPIP_DNS_CLIENT_CACHE_ENTRIES	Number of DNS resolver entries
	TCPIP_DNS_CLIENT_CACHE_ENTRY_TMO	DNS client cache entry time-out. If this symbol is zero then the entry time-out will be the one specified by the DNS server when the name was solved. Otherwise this value will be used as the cache entry time-out. Default should be 0.
	TCPIP_DNS_CLIENT_CACHE_PER_IPV4_ADDRESS	Maximum and default number of IPv4 answers to be considered while processing DNS response from server for a query.
	TCPIP_DNS_CLIENT_CACHE_PER_IPV6_ADDRESS	Maximum and default number of IPv6 answers to be considered while processing DNS response from server for a query.
	TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO	Time-out for the a unsolved name, in seconds. The name resolution will be aborted if the TMO elapsed and the name could not be solved Should be greater than TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO

	TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO	Retry lookup for a unsolved entry in the cache, in seconds. If the TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO seconds elapsed and the name has not been solved then the name entry will be marked with server timeout and the resolution will be retried. Should be less than TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO
	TCPIP_DNS_CLIENT_MAX_HOSTNAME_LEN	Max DNS host name size
	TCPIP_DNS_CLIENT_MAX_SELECT_INTERFACES	Max number of interfaces to take part in the DNS selection algorithm Should be always greater than 1: • the default interface should always be considered for DNS resolution Depending on how many active interfaces select those to be considered for DNS resolution
	TCPIP_DNS_CLIENT_SERVER_TMO	When the DNS Client connected to the DNS Server this is the elapsed time after which an the communication is considered to have timed failed if there was no reply from the server In seconds
	TCPIP_DNS_CLIENT_TASK_PROCESS_RATE	DNS Client task processing rate, in milliseconds. The DNS Client module will process a timer event with this rate for processing its own state machine, etc. The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DNS_CLIENT_USER_NOTIFICATION	allow DNS client user notification if enabled, the TCPIP_DNS_HandlerRegister/TCPIP_DNS_HandlerDeRegister functions exist and can be used

Description

Domain Name Service (CNS) Configuration file

This file contains the DNS module configuration options

File Name

dns_config.h

Company

Microchip Technology Inc.

DNS Server Module

This section describes the TCP/IP Stack Library DNS Server module.

Introduction

TCP/IP Stack Library Domain Name System Server (DNSS) Module for Microchip Microcontrollers

This library provides the API of the DNS Server module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

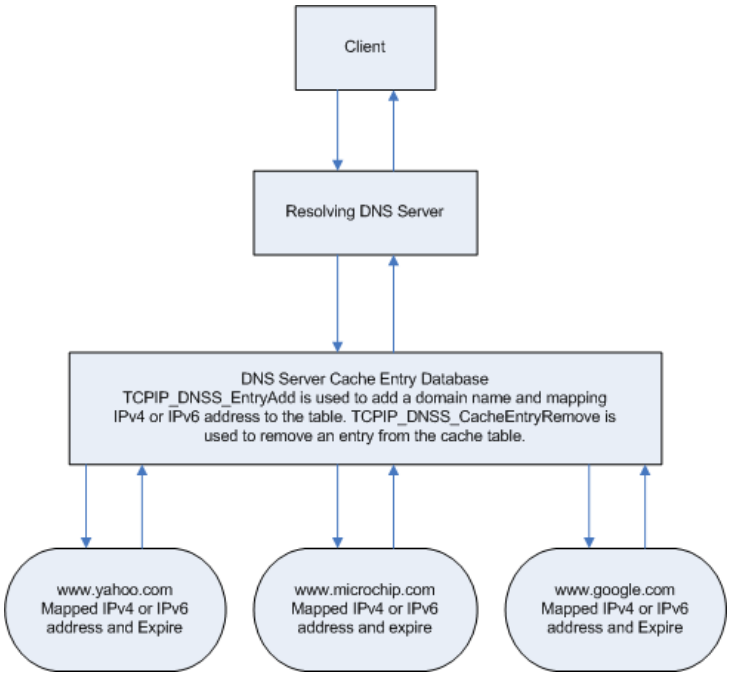
A step-by-step analysis of the DNS Server is as follows:

1. DNS server can be enabled or disabled by using the dnss command (it is not interface based).
2. The client initiates a query to find the Domain Name Server. The client sends the query to the DNS Server of the TCP/IP stack.
3. The DNS Server of the TCP/IP Stack first checks the Boolean value of replyBoardAddr of DNSS_MODULE_CONFIG. If this is true, the DNS server will send an answer with its own board address as the DNS server address to the requested Client. If it is false, it then checks its own cache, whether or not it already knows the answer. But, as the answer is not present, it generates NO response for the client query.

- Cache entry is updated by adding a entry to it using command handler
- Cache entry is the table of Name , IPv4/IPv6 address. and expire time
- The dnsserv command is used to add, delete and display the DNS server details
- Users can add either one IPv4 or IPv6 address at a time
- Maximum Name and IPv4/IPv6 address combination is determined from the IPv4EntriesPerDNSName and IPv6EntriesPerDNSName of DNSS_MODULE_CONFIG

4. Authoritative Name servers and Addition Name servers are not processed.
5. The TCP/IP DNS Server stores the answer in its cache for future use and answers to the client by sending the IP address of the Domain Name Server, which is added by the user using command dnsserv.
6. The client may store the answer to the DNS query in it's own cache for future use. Then, the client communicates directly with the Domain

Name server using the IP address.



Using the Library

This topic describes the basic architecture of the DNSS TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `dnss.h`

The interface to the DNS TCP/IP Stack library is defined in the `dnss.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the DNS TCP/IP Stack library should include `tcpip.h`.

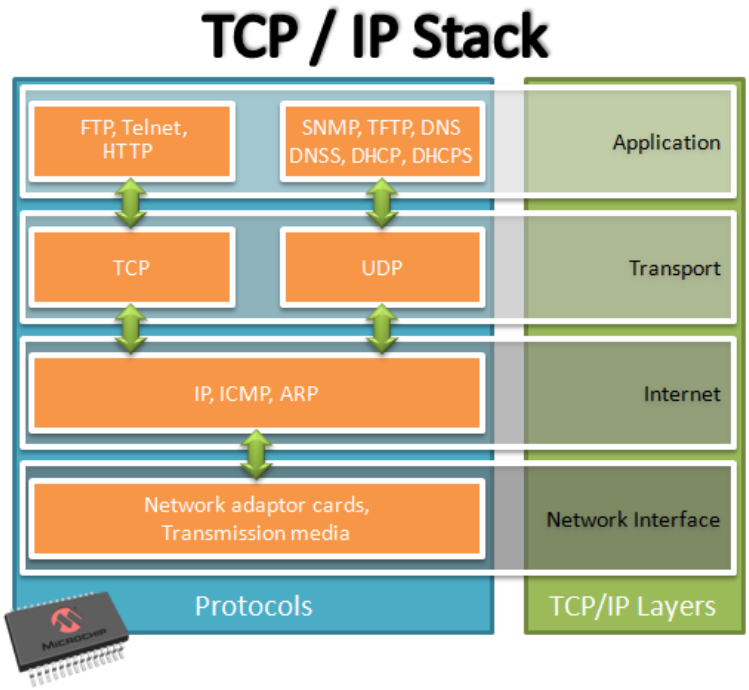
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

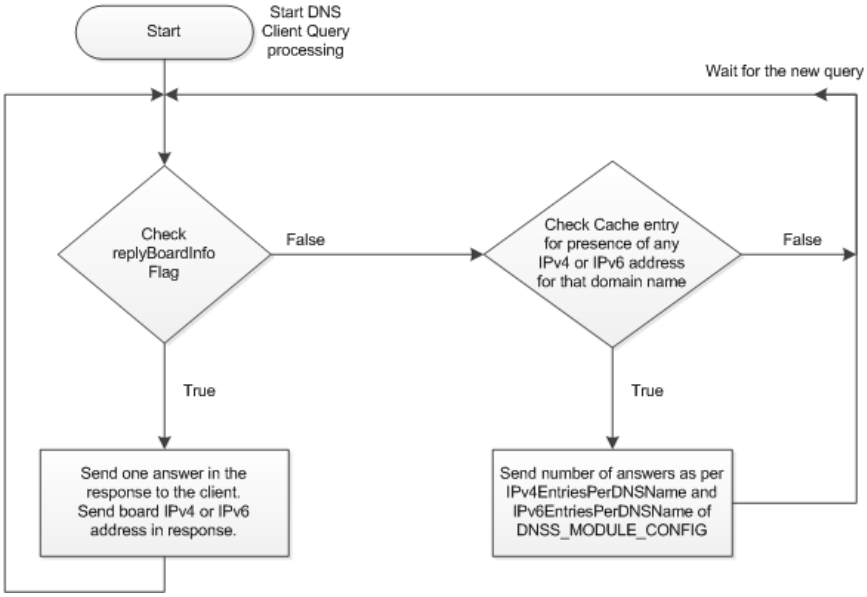
This library provides the API of the DNSS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

DNSS is part of the Application Layer, as illustrated in the following figure.



DNSS resolution operations follow a simple state machine, as indicated in the following diagram.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the DNSS module.

Library Interface Section	Description
General Functions	This section provides general interface routines to DNS
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_DNSS_CACHE_MAX_SERVER_ENTRIES	Maximum DNS server Cache entries. It is the sum of TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS and TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS.
	TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS	Maximum and default number of IPv4 entries will be allowed to be configured from command prompt. and these many entries will be allowed to be sent in response for the DNS query with record type TCPIP_DNS_TYPE_A.
	TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS	Maximum and default number of IPv6 entries will be allowed to be configured from command prompt. and these many entries will be allowed to be sent in response for the DNS query with record type TCPIP_DNS_TYPE_AAAA.
	TCPIP_DNSS_HOST_NAME_LEN	Default DNS host name length
	TCPIP_DNSS_REPLY_BOARD_ADDR	Reply DNS info with Board info only if the requested DNS host name is not present if TCPIP_DNSS_REPLY_BOARD_ADDR != 1 , then return no such name This is used for a boolean variable . the value should be 0 or 1
	TCPIP_DNSS_TASK_PROCESS_RATE	DNS Server time out task processing rate, in milliseconds. The DNS Server module will process a timer event with this rate for processing its own state machine, etc. The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DNSS_TTL_TIME	Default TTL time for a IP address is 10 minutes

Description

The configuration of the DNSS TCP/IP Stack is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the DNSS TCP/IP Stack. Based on the selections made, the DNSS TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the DNSS TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_DNSS_CACHE_MAX_SERVER_ENTRIES Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_CACHE_MAX_SERVER_ENTRIES
(TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS+TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS)
```

Description

Maximum DNS server Cache entries. It is the sum of TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS and TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS.

TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS 2
```

Description

Maximum and default number of IPv4 entries will be allowed to be configured from command prompt. and these many entries will be allowed to be sent in response for the DNS query with record type TCPIP_DNS_TYPE_A.

TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS 1
```

Description

Maximum and default number of IPv6 entries will be allowed to be configured from command prompt. and these many entries will be allowed to be sent in response for the DNS query with record type TCPIP_DNS_TYPE_AAAA.

TCPIP_DNSS_HOST_NAME_LEN Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_HOST_NAME_LEN 64u
```

Description

Default DNS host name length

TCPIP_DNSS_REPLY_BOARD_ADDR Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_REPLY_BOARD_ADDR 1
```

Description

Reply DNS info with Board info only if the requested DNS host name is not present if TCPIP_DNSS_REPLY_BOARD_ADDR != 1 , then return no such name This is used for a boolean variable . the value should be 0 or 1

TCPIP_DNSS_TASK_PROCESS_RATE Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_TASK_PROCESS_RATE (33)
```

Description

DNS Server time out task processing rate, in milliseconds. The DNS Server module will process a timer event with this rate for processing its own state machine, etc. The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_DNSS_TTL_TIME Macro

File

dnss_config.h

C

```
#define TCPIP_DNSS_TTL_TIME (10*60)
```

Description

Default TTL time for a IP address is 10 minutes

Building the Library

This section lists the files that are available in the DNS Server module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dnss.c`	DNS Server implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The DNS Server module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Library Interface

a) General Functions

	Name	Description
	TCPIP_DNSS_AddressCntGet	Get DNS Server IPv4 or IPv4 Address count details for the Input value of Index
	TCPIP_DNSS_CacheEntryRemove	Remove DNS server cache entry by Hostname and IP Type and IP(v4 or v6) address
	TCPIP_DNSS_Disable	Disables the DNS Server for the specified interface.
	TCPIP_DNSS_Enable	Enables the DNS server for the specified interface.
	TCPIP_DNSS_EntryAdd	Add a IPv4 or IPv6 entry to the DNS server table.
	TCPIP_DNSS_IsEnabled	Determines if the DNS Server is enabled on the specified interface.
	TCPIP_DNSS_EntryGet	Get DNS server IP address details from resolver pool entry.
	TCPIP_DNSS_Task	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	TCPIP_DNSS_MODULE_CONFIG	Provides a place holder for DNS server configuration.
	TCPIP_DNSS_RESULT	DNS result codes

Description

This section describes the Application Programming Interface (API) functions of the DNSS module.

Refer to each section for a detailed description.

a) General Functions

TCPIP_DNSS_AddressCntGet Function

Get DNS Server IPv4 or IPv4 Address count details for the Input value of Index

File

dnss.h

C

```
TCPIP_DNSS_RESULT TCPIP_DNSS_AddressCntGet(int index, uint8_t * hostName, uint8_t * ipCount);
```

Returns

TCPIP_DNSS_RESULT

Description

This function is used to get the number of IPv4 and IPv6 address entry details from hash entry table. Here input parameter is index value. Output will be the hostname and IP address count. IP address count is the summation of both IPv4 and IPv6 address.

Remarks

None.

Preconditions

DNSServerInit should be called.

Parameters

Parameters	Description
hostName	Get DNS host name
ipCount	Get the number of IPv4 or IPv6 Server address
index	Server count details per index value

Function

TCPIP_DNSS_RESULT TCPIP_DNSS_AddressCntGet(uint8_t * hostName,uint8_t * ipCount,int index)

TCPIP_DNSS_CacheEntryRemove Function

Remove DNS server cache entry by Hostname and IP Type and IP(v4 or v6) address

File

dnss.h

C

```
TCPIP_DNSS_RESULT TCPIP_DNSS_CacheEntryRemove(const char* name, IP_ADDRESS_TYPE type, IP_MULTI_ADDRESS*
pAdd);
```

Returns

- DNSS_RES_OK - If entry remove is successful
- DNSS_RES_NO_ENTRY - No such entry is present
- DNSS_RES_MEMORY_FAIL - No Memory is present for IPv4 or IPv6 address type

Description

This function is used for command prompt dnsDelSrv and to delete a entry IPv4 or IPv6 from hostName. Every time this is used to delete either one IPv4 or IPv6 address with respect to the DNS hostname, there will be an expire time for a Host entry.

Preconditions

The DNS server must be initialized.

Parameters

Parameters	Description
name	Hostname
type	IP_ADDRESS_TYPE_IPV4 or IP_ADDRESS_TYPE_IPV6
pAdd	v4 or v6 address

Function

TCPIP_DNSS_RESULT TCPIP_DNSS_CacheEntryRemove(const char* name, IP_ADDRESS_TYPE type,
IP_MULTI_ADDRESS* pAdd)

TCPIP_DNSS_Disable Function

Disables the DNS Server for the specified interface.

File

dnss.h

C

```
bool TCPIP_DNSS_Disable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if successful
- false - if unsuccessful

Description

This function disables the DNS Server for the specified interface. If it is already disabled, no action is taken.

Remarks

none

Preconditions

The DNS server must be initialized.

Parameters

Parameters	Description
hNet	Interface on which to disable the DNS Server.

Function

bool TCPIP_DNSS_Disable(TCPIP_NET_HANDLE hNet)

TCPIP_DNSS_Enable Function

Enables the DNS server for the specified interface.

File

dnss.h

C

```
bool TCPIP_DNSS_Enable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if successful
- false - if unsuccessful

Description

This function enables the DNS Server for the specified interface, if it is disabled. If it is already enabled, nothing is done.

Preconditions

The DNS server must be initialized.

Parameters

Parameters	Description
hNet	Interface to enable the DNS Server on.

Function

bool TCPIP_DNSS_Enable(TCPIP_NET_HANDLE hNet)

TCPIP_DNSS_EntryAdd Function

Add a IPv4 or IPv6 entry to the DNS server table.

File

dnss.h

C

```
TCPIP_DNSS_RESULT TCPIP_DNSS_EntryAdd(const char* name, IP_ADDRESS_TYPE type, IP_MULTI_ADDRESS* pAdd,
uint32_t validStartTime);
```

Returns

- DNSS_RES_OK - If entry remove is successful
- DNSS_RES_NO_ENTRY - If Hostname is NULL value and Invalid IP type
- DNSS_RES_MEMORY_FAIL - No Memory is present for IPv4 or IPv6 address type
- DNSS_RES_CACHE_FULL - If there is no space for the new entry

Description

This function is used to add a IPv4 or IPv6 entry. Every time this is used to add either one IPv4 or IPv6 address with respect to the DNS hostname, there will be an expire time for a Host entry.

Remarks

None.

Preconditions

DNS Server Initialized.

Parameters

Parameters	Description
name	Hostname
type	IP_ADDRESS_TYPE_IPV4 or IP_ADDRESS_TYPE_IPV6
pAdd	IPv4 or IPv6 address
validStartTime	Time-out value for the entry

Function

TCPIP_DNSS_RESULT TCPIP_DNSS_EntryAdd(const char* name, IP_ADDRESS_TYPE type,
IP_MULTI_ADDRESS* pAdd,uint32_t validStartTime);

TCPIP_DNSS_IsEnabled Function

Determines if the DNS Server is enabled on the specified interface.

File

dnss.h

C

```
bool TCPIP_DNSS_IsEnabled(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if the DNS Server is enabled on the specified interface
- false - if the DNS Server is not enabled on the specified interface

Description

This function returns the current state of the DNS Server on the specified interface.

Preconditions

The DNS server must be initialized.

Parameters

Parameters	Description
hNet	Interface to query.

Function

bool TCPIP_DNSS_IsEnabled(TCPIP_NET_HANDLE hNet)

TCPIP_DNSS_EntryGet Function

Get DNS server IP address details from resolver pool entry.

File

dnss.h

C

```
TCPIP_DNSS_RESULT TCPIP_DNSS_EntryGet(uint8_t * hostName, IP_ADDRESS_TYPE type, int index,
IP_MULTI_ADDRESS* pGetAdd, uint32_t * ttltime);
```

Returns

TCPIP_DNSS_RESULT

Description

This function is used to get the DNS server IPv4 or IPv6 address from resolver pool entry as per hostname and IP type and from the index value. This IP type can be a IPv4 and IPv6 type. This is used for DNS record type.

Remarks

None.

Preconditions

DNSServerInit should be called.

Parameters

Parameters	Description
hostName	DNS host name
type	DNS IP type (it will be ipv4 or ipv6 type and it is mapped to DNS record type) this can be used for DNS record type
index	get the next entry after this index value
pGetAdd	get the zero th IPv4 or IPv6 address
ttltime	timeout value

Function

TCPIP_DNSS_RESULT TCPIP_DNSS_EntryGet(uint8_t * hostName,IP_ADDRESS_TYPE type,
int index, IP_MULTI_ADDRESS* pGetAdd,uint32_t *ttltime);

TCPIP_DNSS_Task Function

Standard TCP/IP stack module task function.

File

dnss.h

C

```
void TCPIP_DNSS_Task();
```

Returns

None.

Description

This function performs DNSS module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The DNSS module should have been initialized.

Function

void TCPIP_DNSS_Task(void)

b) Data Types and Constants

TCPIP_DNSS_MODULE_CONFIG Structure

Provides a place holder for DNS server configuration.

File

dnss.h

C

```
typedef struct {
  bool deleteOldLease;
  bool replyBoardAddr;
  size_t IPv4EntriesPerDNSName;
  size_t IPv6EntriesPerDNSName;
} TCPIP_DNSS_MODULE_CONFIG;
```

Members

Members	Description
bool deleteOldLease;	Delete old cache if still in place,
bool replyBoardAddr;	Reply with board address specific DNS server parameters
size_t IPv4EntriesPerDNSName;	Number of IPv4 entries per DNS name. Default value is 1.
size_t IPv6EntriesPerDNSName;	Number of IPv6 address per DNS Name. Default value is 1 Used only when IPv6 is enabled

Description

Structure: TCPIP_DNSS_MODULE_CONFIG

DNS Server run-time configuration and initialization parameter.

Remarks

None.

TCPIP_DNSS_RESULT Enumeration

DNS result codes

File

dnss.h

C

```
typedef enum {
  DNSS_RES_OK = 0,
  DNSS_RES_NO_SERVICE = -1,
  DNSS_RES_CACHE_FULL = -2,
  DNSS_RES_NO_ENTRY = -3,
  DNSS_RES_NO_IPADDRESS = -4,
  DNSS_RES_MEMORY_FAIL = -5,
```

```
  DNSS_RES_DUPLICATE_ENTRY = -6
} TCPIP_DNSS_RESULT;
```

Members

Members	Description
DNSS_RES_OK = 0	operation succeeded
DNSS_RES_NO_SERVICE = -1	DNS service not implemented or uninitialized
DNSS_RES_CACHE_FULL = -2	the cache is full and no entry could be added
DNSS_RES_NO_ENTRY = -3	DNSS no such name
DNSS_RES_NO_IPADDRESS = -4	No such IP address
DNSS_RES_MEMORY_FAIL = -5	out of memory failure
DNSS_RES_DUPLICATE_ENTRY = -6	duplicate entry was found

Description

Enumeration: TCPIP_DNSS_RESULT

DNS Server operations results.

Remarks

None.

Files

Files

Name	Description
dnss.h	DNS server definitions and interface file
dnss_config.h	DNSS configuration file

Description

This section lists the source and header files used by the library.

dnss.h

DNS server definitions and interface file

Enumerations

	Name	Description
	TCPIP_DNSS_RESULT	DNS result codes

Functions

	Name	Description
	TCPIP_DNSS_AddressCntGet	Get DNS Server IPv4 or IPv4 Address count details for the Input value of Index
	TCPIP_DNSS_CacheEntryRemove	Remove DNS server cache entry by Hostname and IP Type and IP(v4 or v6) address
	TCPIP_DNSS_Disable	Disables the DNS Server for the specified interface.
	TCPIP_DNSS_Enable	Enables the DNS server for the specified interface.
	TCPIP_DNSS_EntryAdd	Add a IPv4 or IPv6 entry to the DNS server table.
	TCPIP_DNSS_EntryGet	Get DNS server IP address details from resolver pool entry.
	TCPIP_DNSS_IsEnabled	Determines if the DNS Server is enabled on the specified interface.
	TCPIP_DNSS_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_DNSS_MODULE_CONFIG	Provides a place holder for DNS server configuration.

Description

Domain Name System (DNS) server Header file

This source file contains the DNS Server module API

File Name

dnss.h

Company

Microchip Technology Inc.

dnss_config.h

DNSS configuration file

Macros

	Name	Description
	TCPIP_DNSS_CACHE_MAX_SERVER_ENTRIES	Maximum DNS server Cache entries. It is the sum of TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS and TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS.
	TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS	Maximum and default number of IPv4 entries will be allowed to be configured from command prompt. and these many entries will be allowed to be sent in response for the DNS query with record type TCPIP_DNS_TYPE_A.
	TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS	Maximum and default number of IPv6 entries will be allowed to be configured from command prompt. and these many entries will be allowed to be sent in response for the DNS query with record type TCPIP_DNS_TYPE_AAAA.
	TCPIP_DNSS_HOST_NAME_LEN	Default DNS host name length
	TCPIP_DNSS_REPLY_BOARD_ADDR	Reply DNS info with Board info only if the requested DNS host name is not present if TCPIP_DNSS_REPLY_BOARD_ADDR != 1 , then return no such name This is used for a boolean variable . the value should be 0 or 1
	TCPIP_DNSS_TASK_PROCESS_RATE	DNS Server time out task processing rate, in milliseconds. The DNS Server module will process a timer event with this rate for processing its own state machine, etc. The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_DNSS_TTL_TIME	Default TTL time for a IP address is 10 minutes

Description

Domain Name System Server (DNSS) Configuration file

This file contains the DNSS module configuration options

File Name

dnss_config.h

Company

Microchip Technology Inc.

Dynamic DNS Module

This section describes the TCP/IP Stack Library Dynamic DNS (DDNS) module.

Introduction

TCP/IP Stack Library Dynamic Domain Name System (DDNS) Module for Microchip Microcontrollers

This library provides the API of the DDNS module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The Dynamic DNS Client module provides a method for updating a dynamic IP address to a public DDNS service. These services can be used to provide DNS hostname mapping to devices that behind routers, firewalls, and/or on networks that dynamically assign IP addresses.

Note that this only solves one of the two problems for communicating to devices on local subnets from the Internet. While Dynamic DNS can help to locate the device, the router or firewall it sits behind must still properly forward the incoming connection request. This generally requires port forwarding to be configured for the router behind which the device is located.

The Dynamic DNS client supports the popular interface used by No-IP (noip.com) and DNS-O-Matic (dnsomatic.com).

Important! The dynamic DNS services stipulate that updates should be made no more frequently than 10 minutes, and only when the IP address has changed. Updates made more often than that are considered abusive, and may eventually cause your account to be disabled. Production devices that get rebooted frequently may need to store the last known IP in non-volatile memory. You also should not enable this module while testing the rest of your application.

Using the Library

This topic describes the basic architecture of the DDNS TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `ddns.h`

The interface to the DDNS TCP/IP Stack library is defined in the `ddns.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the DDNS TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the DDNS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

DDNS Software Abstraction Block Diagram

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Dynamic DNS module.

Library Interface Section	Description
Functions	Routines for Configuring DDNS
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_DDNS_CHECKIP_SERVER	Default CheckIP server for determining current IP address
	TCPIP_DDNS_TASK_TICK_RATE	dynDNS task rate, ms The default value is hundreds of milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

The configuration of the DDNS TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the DDNS TCP/IP Stack Library. Based on the selections made, the DDNS TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the DDNS TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_DDNS_CHECKIP_SERVER Macro

File

ddns_config.h

C

```
#define TCPIP_DDNS_CHECKIP_SERVER (const uint8_t*)"checkip.dyndns.com"
```

Description

Default CheckIP server for determining current IP address

TCPIP_DDNS_TASK_TICK_RATE Macro

File

ddns_config.h

C

```
#define TCPIP_DDNS_TASK_TICK_RATE 777
```

Description

dynDNS task rate, ms The default value is hundreds of milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Building the Library

This section lists the files that are available in the Dynamic DNS module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/ddns.c`	Dynamic DNS Server implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Dynamic DNS module depends on the following modules:

- TCP/IP Stack Library
- DNS Module
- TCP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_DDNS_LastIPGet	Returns the status of the most recent update.
	TCPIP_DDNS_LastStatusGet	Returns the last known external IP address of the device.
	TCPIP_DDNS_ServiceSet	Selects a preconfigured Dynamic DNS service.
	TCPIP_DDNS_UpdateForce	Forces an immediate DDNS update.
	TCPIP_DDNS_Task	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	DDNS_MODULE_CONFIG	This is type DDNS_MODULE_CONFIG.
	DDNS_POINTERS	Configuration parameters for the Dynamic DNS Client
	DDNS_SERVICES	Dynamic DNS Services. Must support the DynDNS API (Auxlang) and correspond to ddnsServiceHosts and ddnsServicePorts in DynDNS.c.
	DDNS_STATUS	Status message for DynDNS client. GOOD and NOCHG are okay, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

Description

This section describes the Application Programming Interface (API) functions of the Dynamic DNS module.

Refer to each section for a detailed description.

a) Functions

TCPIP_DDNS_LastIPGet Function

Returns the status of the most recent update.

File

ddns.h

C

```
IPV4_ADDR TCPIP_DDNS_LastIPGet();
```

Returns

DDNS_STATUS indicating the status code for the most recent update.

Description

This function returns the status of the most recent update. See the DDNS_STATUS enumeration for possible codes.

Preconditions

None.

Function

DDNS_STATUS TCPIP_DDNS_LastStatusGet(void)

TCPIP_DDNS_LastStatusGet Function

Returns the last known external IP address of the device.

File

ddns.h

C

```
DDNS_STATUS TCPIP_DDNS_LastStatusGet();
```

Returns

The last known external IP address of the device.

Description

This function returns the last known external IP address of the device.

Preconditions

None.

Function

IPV4_ADDR TCPIP_DDNS_LastIPGet(void)

TCPIP_DDNS_ServiceSet Function

Selects a preconfigured Dynamic DNS service.

File

ddns.h

C

```
void TCPIP_DDNS_ServiceSet(DDNS_SERVICES svc);
```

Returns

None.

Description

This function selects a Dynamic DNS service based on parameters configured in ddnsServiceHosts and ddnsServicePorts. These arrays must match the DDNS_SERVICES enumeration.

Preconditions

None.

Parameters

Parameters	Description
svc	one of the DDNS_SERVICES elements to indicate the selected service

Function

void TCPIP_DDNS_ServiceSet(DDNS_SERVICES svc)

TCPIP_DDNS_UpdateForce Function

Forces an immediate DDNS update.

File

ddns.h

C

```
void TCPIP_DDNS_UpdateForce();
```

Returns

None.

Description

This function forces the DDNS Client to execute a full update immediately. Any error message is cleared, and the update will be executed whether the IP address has changed or not. Call this function every time the DDNSClient parameters have been modified.

Preconditions

TCPIP_DDNS_Initialize must have been called.

Function

void TCPIP_DDNS_UpdateForce(void)

TCPIP_DDNS_Task Function

Standard TCP/IP stack module task function.

File

ddns.h

C

```
void TCPIP_DDNS_Task();
```

Returns

None.

Description

Performs DDNS module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

DDNS module should have been initialized

Function

void TCPIP_DDNS_Task(void)

b) Data Types and Constants

DDNS_MODULE_CONFIG Structure

File

ddns.h

C

```
typedef struct {
} DDNS_MODULE_CONFIG;
```

Description

This is type DDNS_MODULE_CONFIG.

DDNS_POINTERS Structure

Configuration parameters for the Dynamic DNS Client

File

ddns.h

C

```
typedef struct {
  union {
    uint8_t * szRAM;
    const uint8_t * szROM;
  } CheckIPServer;
  uint16_t CheckIPPort;
  union {
    uint8_t * szRAM;
    const uint8_t * szROM;
  } UpdateServer;
  uint16_t UpdatePort;
  union {
    uint8_t * szRAM;
    const uint8_t * szROM;
  } Username;
```

```
  union {
    uint8_t * szRAM;
    const uint8_t * szROM;
  } Password;
  union {
    uint8_t * szRAM;
    const uint8_t * szROM;
  } Host;
  struct {
    unsigned char CheckIPServer : 1;
    unsigned char UpdateServer : 1;
    unsigned char Username : 1;
    unsigned char Password : 1;
    unsigned char Host : 1;
  } ROMPointers;
} DDNS_POINTERS;
```

Description

This structure of pointers configures the Dynamic DNS Client. Initially, all pointers will be null and the client will be disabled. Set `DDNSClient.[field name].szRAM` to use a string stored in RAM, or `DDNSClient.[field name].szROM` to use a string stored in const. Where, `[field name]`, is one of the following parameters.

If a const string is specified, `DDNSClient.ROMPointers.[field name]` must also be set to 1 to indicate that this field should be retrieved from const instead of RAM.

Parameters

Parameters	Description
CheckIPServer	The server used to determine the external IP address
CheckIPPort	Port on the above server to connect to
UpdateServer	The server where updates should be posted
UpdatePort	Port on the above server to connect to
Username	The user name for the dynamic DNS server
Password	The password to supply when making updates
Host	The host name you wish to update
ROMPointers	Indicates which parameters to read from const instead of RAM.

DDNS_SERVICES Enumeration

File

ddns.h

C

```
typedef enum {
  DYNDNS_ORG = 0u,
  NO_IP_COM,
  DNSOMATIC_COM
} DDNS_SERVICES;
```

Members

Members	Description
DYNDNS_ORG = 0u	www.dyndns.org
NO_IP_COM	www.no-ip.com
DNSOMATIC_COM	www.dnsomatic.com

Description

Dynamic DNS Services. Must support the DynDNS API (Auxlang) and correspond to ddnsServiceHosts and ddnsServicePorts in DynDNS.c.

DDNS_STATUS Enumeration

File

ddns.h

C

```
typedef enum {
  DDNS_STATUS_GOOD = 0u,
  DDNS_STATUS_NOCHG,
  DDNS_STATUS_ABUSE,
  DDNS_STATUS_BADSYS,
  DDNS_STATUS_BADAGENT,
  DDNS_STATUS_BADAUTH,
  DDNS_STATUS_NOT_DONATOR,
  DDNS_STATUS_NOT_FQDN,
  DDNS_STATUS_NOHOST,
  DDNS_STATUS_NOT_YOURS,
  DDNS_STATUS_NUMHOST,
  DDNS_STATUS_DNSERR,
  DDNS_STATUS_911,
  DDNS_STATUS_UPDATE_ERROR,
  DDNS_STATUS_UNCHANGED,
  DDNS_STATUS_CHECKIP_ERROR,
  DDNS_STATUS_DNS_ERROR,
  DDNS_STATUS_SKT_ERROR,
  DDNS_STATUS_INVALID,
  DDNS_STATUS_UNKNOWN
} DDNS_STATUS;
```

Members

Members	Description
DDNS_STATUS_GOOD = 0u	Update successful, hostname is now updated
DDNS_STATUS_NOCHG	Update changed no setting and is considered abusive. Additional 'nochg' updates will cause hostname to be blocked.
DDNS_STATUS_ABUSE	The hostname specified is blocked for update abuse.
DDNS_STATUS_BADSYS	System parameter not valid. Should be dyndns, statdns or custom.
DDNS_STATUS_BADAGENT	The user agent was blocked or not sent.
DDNS_STATUS_BADAUTH	The username and password pair do not match a real user.
DDNS_STATUS_NOT_DONATOR	An option available only to credited users (such as offline URL) was specified, but the user is not a credited user. If multiple hosts were specified, only a single !donator will be returned.
DDNS_STATUS_NOT_FQDN	The hostname specified is not a fully-qualified domain name (not in the form hostname.dyndns.org or domain.com).
DDNS_STATUS_NOHOST	The hostname specified does not exist in this user account (or is not in the service specified in the system parameter).
DDNS_STATUS_NOT_YOURS	The hostname specified does not belong to this user account.
DDNS_STATUS_NUMHOST	Too many hosts specified in an update.
DDNS_STATUS_DNSERR	Unspecified DNS error encountered by the DDNS service.
DDNS_STATUS_911	There is a problem or scheduled maintenance with the DDNS service.
DDNS_STATUS_UPDATE_ERROR	Error communicating with Update service.
DDNS_STATUS_UNCHANGED	The IP Check indicated that no update was necessary.
DDNS_STATUS_CHECKIP_ERROR	Error communicating with CheckIP service.
DDNS_STATUS_DNS_ERROR	DNS error resolving the CheckIP service.
DDNS_STATUS_SKT_ERROR	TCP socket opening error
DDNS_STATUS_INVALID	DDNS Client data is not valid.
DDNS_STATUS_UNKNOWN	DDNS client has not yet been executed with this configuration.

Description

Status message for DynDNS client. GOOD and NOCHG are okay, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

Files

Files

Name	Description
ddns.h	The Dynamic DNS Client module provides a method for updating a dynamic IP address to a public DDNS service.
ddns_config.h	Dynamic DNS configuration file

Description

This section lists the source and header files used by the library.

ddns.h

The Dynamic DNS Client module provides a method for updating a dynamic IP address to a public DDNS service.

Enumerations

	Name	Description
	DDNS_SERVICES	Dynamic DNS Services. Must support the DynDNS API (Auxlang) and correspond to ddnsServiceHosts and ddnsServicePorts in DynDNS.c.
	DDNS_STATUS	Status message for DynDNS client. GOOD and NOCHG are okay, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

Functions

	Name	Description
	TCPIP_DDNS_LastIPGet	Returns the status of the most recent update.
	TCPIP_DDNS_LastStatusGet	Returns the last known external IP address of the device.
	TCPIP_DDNS_ServiceSet	Selects a preconfigured Dynamic DNS service.
	TCPIP_DDNS_Task	Standard TCP/IP stack module task function.
	TCPIP_DDNS_UpdateForce	Forces an immediate DDNS update.

Structures

	Name	Description
	DDNS_MODULE_CONFIG	This is type DDNS_MODULE_CONFIG.
	DDNS_POINTERS	Configuration parameters for the Dynamic DNS Client

Description

DynDNS Headers for Microchip TCP/IP Stack API Header File

The Dynamic DNS Client module provides a method for updating a dynamic IP address to a public DDNS service. These services can be used to provide DNS hostname mapping to devices that behind routers, firewalls, and/or on networks that dynamically assign IP addresses.

File Name

ddns.h

Company

Microchip Technology Inc.

ddns_config.h

Dynamic DNS configuration file

Macros

	Name	Description
	TCPIP_DDNS_CHECKIP_SERVER	Default CheckIP server for determining current IP address
	TCPIP_DDNS_TASK_TICK_RATE	dynDNS task rate, ms The default value is hundreds of milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

Dynamic Domain Name Service (DynDNS) Configuration file

This file contains the Dynamic DNS module configuration options

File Name

ddns_config.h

Company

Microchip Technology Inc.

FTP Module

This section describes the TCP/IP Stack Library FTP module.

Introduction

TCP/IP Stack Library File Transfer Protocol (FTP) Module for Microchip Microcontrollers

This library provides the API of the FTP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

An embedded File Transfer Protocol (FTP) server is an excellent addition to any network-enabled device. FTP server capability facilitates the uploading of files to, and downloading of files from, an embedded device. Almost all computers have, at the very least, a command line FTP client that will allow a user to connect to an embedded FTP server.

Using the Library

This topic describes the basic architecture of the FTP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `ftp.h`

The interface to the FTP TCP/IP Stack library is defined in the `ftp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the FTP TCP/IP Stack library should include `tcpip.h`.

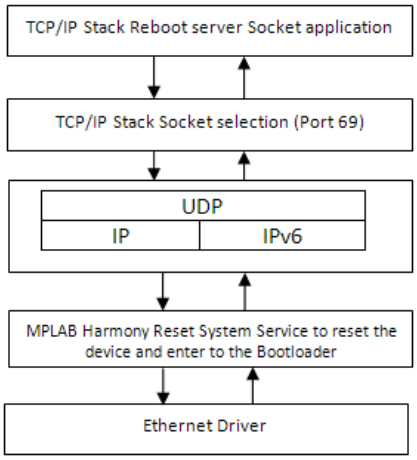
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the FTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

FTP Server Dependencies



Since the FTP server requires the use of the TCP/IP stack and a file system, which can be a MPFS or MDD, it also inherits the program memory and RAM requirements of those as well.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the FTP module.

Library Interface Section	Description
General Functions	This section provides a routine that performs FTP module tasks in the TCP/IP stack.
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the FTP Data socket Use 0 for default TCP RX buffer size. The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the FTP Data socket Use 0 for default TCP TX buffer size. The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_FTP_MAX_CONNECTIONS	Maximum number of FTP connections allowed
	TCPIP_FTP_PASSWD_LEN	Specifies the max length of FTP login password
	TCPIP_FTP_PUT_ENABLED	Comment this line out to disable MPFS
	TCPIP_FTP_USER_NAME_LEN	Specifies the max length for user name
	TCPIP_FTPS_TASK_TICK_RATE	The FTP server task rate, milliseconds The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the FTP server priority and higher transfer rates could be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_FTP_TIMEOUT	FTP timeout, seconds

Description

The configuration of the FTP TCP/IP Stack is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the FTP TCP/IP Stack. Based on the selections made, the FTP TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the FTP TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE 0
```

Description

Define the size of the RX buffer for the FTP Data socket Use 0 for default TCP RX buffer size. The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE 0
```

Description

Define the size of the TX buffer for the FTP Data socket Use 0 for default TCP TX buffer size. The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

TCPIP_FTP_MAX_CONNECTIONS Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_MAX_CONNECTIONS (1)
```

Description

Maximum number of FTP connections allowed

TCPIP_FTP_PASSWD_LEN Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_PASSWD_LEN (10)
```

Description

Specifies the max length of FTP login password

TCPIP_FTP_PUT_ENABLED Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_PUT_ENABLED
```

Description

Comment this line out to disable MPFS

TCPIP_FTP_USER_NAME_LEN Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_USER_NAME_LEN (10)
```

Description

Specifies the max length for user name

TCPIP_FTPS_TASK_TICK_RATE Macro

File

ftp_config.h

C

```
#define TCPIP_FTPS_TASK_TICK_RATE 33
```

Description

The FTP server task rate, milliseconds The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the FTP server priority and higher transfer rates could be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_FTP_TIMEOUT Macro

File

ftp_config.h

C

```
#define TCPIP_FTP_TIMEOUT (1200ul)
```

Description

FTP timeout, seconds

Building the Library

This section lists the files that are available in the FTP module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/ftp.c`	FTP implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The FTP module depends on the following modules:

- TCP/IP Stack Library
- TCP Module
- File System Service Library

Library Interface

a) General Functions

	Name	Description
	TCPIP_FTP_ServerTask	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	TCPIP_FTP_MODULE_CONFIG	FTP Sever module runtime and initialization configuration data.

Description

This section describes the Application Programming Interface (API) functions of the FTP module.

Refer to each section for a detailed description.

a) General Functions

TCPIP_FTP_ServerTask Function

Standard TCP/IP stack module task function.

File

ftp.h

C

```
void TCPIP_FTP_ServerTask();
```

Returns

None.

Description

This function performs FTP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The FTP module should have been initialized.

Function

void TCPIP_FTP_ServerTask(void)

b) Data Types and Constants

TCPIP_FTP_MODULE_CONFIG Structure

FTP Sever module runtime and initialization configuration data.

File

ftp.h

C

```
typedef struct {
  uint16_t nConnections;
  uint16_t dataSktTxBuffSize;
  uint16_t dataSktRxBuffSize;
  char * userName;
  char * password;
} TCPIP_FTP_MODULE_CONFIG;
```

Members

Members	Description
uint16_t nConnections;	number of simultaneous FTP connections allowed
uint16_t dataSktTxBuffSize;	size of Data socket TX buffer for the associated socket; leave 0 for default
uint16_t dataSktRxBuffSize;	size of Data Socket RX buffer for the associated socket; leave 0 for default
char * userName;	FTP login User name. Size should not exceed more than TCPIP_FTP_USER_NAME_LEN
char * password;	FTP login password. Size should not exceed more than TCPIP_FTP_PASSWD_LEN

Description

Structure: TCPIP_FTP_MODULE_CONFIG

FTP server configuration and initialization data . Configuration is part of tcpip_stack_init.c.

Files

Files

Name	Description
ftp.h	An embedded FTP (File Transfer Protocol) server is an excellent addition to any network-enabled device.
ftp_config.h	FTP configuration file

Description

This section lists the source and header files used by the library.

ftp.h

An embedded FTP (File Transfer Protocol) server is an excellent addition to any network-enabled device.

Functions

	Name	Description
	TCPIP_FTP_ServerTask	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_FTP_MODULE_CONFIG	FTP Sever module runtime and initialization configuration data.

Description

FTP Server Definitions for the Microchip TCP/IP Stack

An embedded FTP (File Transfer Protocol) server is an excellent addition to any network-enabled device. FTP server capability facilitates the uploading of files to, and downloading of files from, an embedded device. Almost all computers have, at the very least, a command line FTP client that will allow a user to connect to an embedded FTP server.

File Name

ftp.h

Company

Microchip Technology Inc.

ftp_config.h

FTP configuration file

Macros

	Name	Description
	TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the FTP Data socket Use 0 for default TCP RX buffer size. The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the FTP Data socket Use 0 for default TCP TX buffer size. The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_FTP_MAX_CONNECTIONS	Maximum number of FTP connections allowed
	TCPIP_FTP_PASSWD_LEN	Specifies the max length of FTP login password
	TCPIP_FTP_PUT_ENABLED	Comment this line out to disable MPFS
	TCPIP_FTP_TIMEOUT	FTP timeout, seconds
	TCPIP_FTP_USER_NAME_LEN	Specifies the max length for user name
	TCPIP_FTPS_TASK_TICK_RATE	The FTP server task rate, milliseconds The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the FTP server priority and higher transfer rates could be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

File Transfer Protocol (FTP) Configuration file

This file contains the FTP module configuration options

File Name

ftp_config.h

Company

Microchip Technology Inc.

HTTP Module

This section describes the TCP/IP Stack Library HTTP module.

Introduction

TCP/IP Stack Library Hypertext Transfer Protocol (HTTP) Module for Microchip Microcontrollers

This library provides the API of the HTTP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The HTTP Web Server module allows a development board to act as a Web server. This facilitates an easy method to view status information and control applications using any standard Web browser.

Using the Library

This topic describes the basic architecture of the HTTP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `http.h`

The interface to the HTTP TCP/IP Stack library is defined in the `http.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the HTTP TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

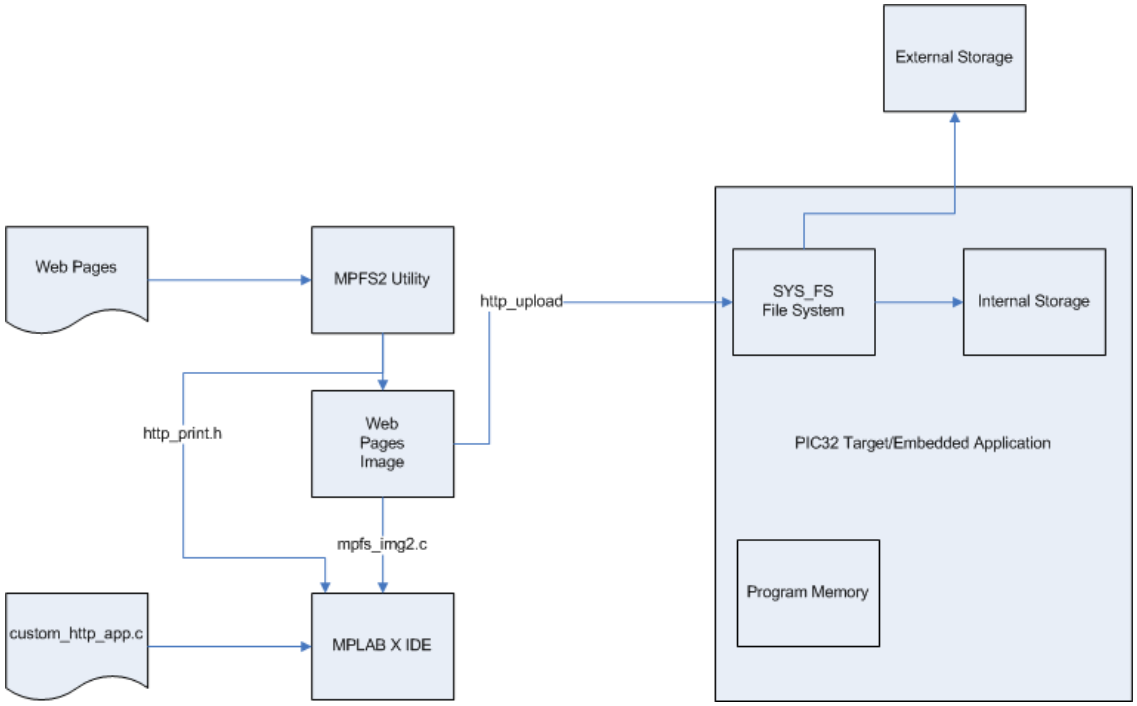
Abstraction Model

This library provides the API of the HTTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

HTTP Software Abstraction Block Diagram

Three main components are necessary to understand how the HTTP web server works: the web pages, the MPFS2 Utility, and the source files `custom_http_app.c` and `http_print.h`. An overview of the entire process is shown in the following diagram.



Web Pages

This includes all the HTML and associated images, CSS style sheets, and JavaScript files necessary to display the website. A sample application including all these components is located in the `web_pages2` folder.

MPFS2 Utility

This program, supplied by Microchip, packages the web pages into a format that can be efficiently stored in either external non-volatile storage, or internal Flash program memory. This program also indexes dynamic variables found in the web pages and updates `http_print.h` with these indices. The MPFS2 Utility generates a SYS_FS compatible file image that can be directly uploaded on the PIC32 development board or stored in the Flash program memory as a source file image to be included in the project.

When dynamic variables are added or removed from your application, the MPFS2 Utility will update `http_print.h`. When this happens, the project must be recompiled in the MPLAB X IDE to ensure that all the new variable indices get added into the application.

custom_http_app.c

This file implements the Web application. It describes the output for dynamic variables (via TCPIP_HTTP_Print_varname callbacks), parses data submitted through forms (in TCPIP_HTTP_GetExecute and TCPIP_HTTP_PostExecutePost) and validates authorization credentials (in TCPIP_HTTP_FileAuthenticate and TCPIP_HTTP_UserAuthenticate).

The functionality of these callbacks is described within the demonstration application's web pages, and is also documented within the `custom_http_app.c` example that is distributed with the stack.

http_print.h

This file is generated automatically by the MPFS2 Utility. It indexes all of the dynamic variables and provides the "glue" between the variables located in the Web pages and their associated TCPIP_HTTP_Print_varname callback functions defined in `custom_http_app.c`. This file does not require modification by the programmer.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the HTTP module.

Library Interface Section	Description
Functions	Routines for Configuring the HTTP module
Data Types and Constants	This section provides various definitions describing this API

HTTP Features

The HTTP Web Server module has many capabilities. The following topics will introduce these features and provide examples.

Dynamic Variables

One of the most basic needs is to provide status information back to the user of your Web application. The HTTP server provides for this using dynamic variable substitution callbacks. These commands in your HTML code will alert the server to execute a callback function at that point, which the developer creates to write data into the web page. Dynamic Variables should be considered the output of your application.

Description

Basic Use

To create a dynamic variable, simply enclose the name of the variable inside a pair of tilde (~) characters within the web pages' HTML source code. (ex: ~myVariable~) When you run the MPFS2 Utility to generate the web pages, it will automatically index these variables in `http_print.h`. This index will instruct your application to invoke the function TCPIP_HTTP_Print_myVariable when this string is encountered.

Here is an example of using a dynamic variable to insert the build date of your application into the web pages:

```
<div class="examplebox code">~builddate~</div>
The associated callback will print the value into the web page:
void TCPIP_HTTP_Print_builddate(HTTP_CONN_HANDLE connHandle)
{
    TCP_SOCKET sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
    TCPIP_TCP_StringPut(sktHTTP, (const void*)__DATE__" "__TIME__);
}
```

Passing Parameters

You can also pass parameters to dynamic variables by placing numeric values inside of parenthesis after the variable name.

For example, ~led(2)~ will print the value of the second LED. The numeric values are passed as 16 bit values to your callback function. You can pass as many parameters as you wish to these functions, and if your C code has constants defined, those will be parsed as well. (ex: ~pair(3,TRUE)~).

The following code inserts the value of the push buttons into the web page, all using the same callback function:

```
<div class="examplebox code">btn(3)~ btn(2)~ btn(1)~ btn(0)~</div>
This associated callback will print the value of the requested button to the web page:
void TCPIP_HTTP_Print_btn(HTTP_CONN_HANDLE connHandle, uint16_t num)
{
    // Determine which button
    switch(num)
    {
        case 0:
            num = BUTTON0_IO;
            break;
        case 1:
            num = BUTTON1_IO;
            break;
        case 2:
            num = BUTTON2_IO;
            break;
        default:
            num = 0;
    }

    // Print the output
    TCPIP_TCP_StringPut(TCPIP_HTTP_CurrentConnectionSocketGet(connHandle), (num ? "up" : "down"));
}
```

Longer Outputs

The HTTP protocol operates in a fixed memory buffer for transmission, so not all data can be sent at once. Care must be taken inside of your callback function to avoid overrunning this buffer.

The callback functions must check how much space is available, write up to that many bytes, and then return. The callback will be invoked again when more space is free.

Note: For increased throughput performance, it is recommended that the callback should always try to write as much data as possible in each call.

To manage the output state, callbacks should make use of the callbackPos variable that is maintained by the Web server for each individual connection. This 32-bit value can be manipulated with the TCPIP_HTTP_CurrentConnectionCallbackPosGet and TCPIP_HTTP_CurrentConnectionCallbackPosSet functions.

The callbackPos is set to zero when a callback is first invoked. If a callback is only writing part of its output, it should set this field to a non-zero

value to indicate that it should be called again when more space is available. This value will be available to the callback during the next call, which allows the function to resume output where it left off. A common use is to store the number of bytes written, or remaining to be written, in this field. Once the callback is finished writing its output, it must set callbackPos back to zero to indicate completion.

The following code outputs the value of the build date of the application into the web pages:

```
<div class="examplebox code">~builddate~</div>

void TCPIP_HTTP_Print_builddate(HTTP_CONN_HANDLE connHandle)
{
    TCP_SOCKET sktHTTP;
    sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);

    TCPIP_HTTP_CurrentConnectionCallbackPosSet(connHandle, 0x01);
    if(TCPIP_TCP_PutIsReady(sktHTTP) < strlen((const char*)__DATE__" "__TIME__))
        return;

    TCPIP_HTTP_CurrentConnectionCallbackPosSet(connHandle, 0x00);
    TCPIP_TCP_StringPut(sktHTTP, (const void*)__DATE__" "__TIME__);
}
```

The initial call to TCPIP_TCP_PutIsReady determines how many bytes can be written to the buffer right now. When there is not enough buffer space for performing the output operation the function sets the current value of the callbackPos and returns. Once the output operation is performed the callbackPos is cleared to '0'.

Including Files

Often it is useful to include the entire contents of another file in your output. Most web pages have at least some portion that does not change, such as the header, menu of links, and footer. These sections can be abstracted out into separate files which makes them easier to manage and conserves storage space.

To include the entire contents of another file, use a dynamic variable that starts with "inc:", such as ~inc:header.inc~.

This sequence will cause the file header.inc to be read from the file system and inserted at this location.

The following example indicates how to include a standard menu bar section into every page:

```
<div id="menu">~inc:menu.inc~</div>
```

At this time, dynamic variables are not recursive, so any variables located inside files included in this manner are not parsed.

Form Processing

Many applications need to accept data from a user. A common solution is to present a form to the user in a Web page, and then have the device process the values submitted via this form. Web forms are usually submitted using one of two methods (GET and POST), and the HTTP Web server supports both.

Description

The GET Method

The GET method appends the data to the end of the URI. This data follows the question mark (?) in the browser's address bar. (ex: http://mchpboard/form.htm?led1=0&led2=1&led3=0) Data sent via GET is automatically decoded and stored in the current connection data buffer. Since it is to be stored in memory, this data is limited to the size of the connection data buffer which by default is 100 bytes (configurable by HTTP_MAX_DATA_LEN build time symbol or by the run time data passes at the HTTP module initialization).

It is generally easier to process data received in this manner.

The callback function TCPIP_HTTP_GetExecute is implemented by the application developer to process this data and perform any necessary actions. The function TCPIP_HTTP_ArgGet provides an easy method to retrieve submitted values for processing.

See the custom _http_app.c for an example of TCPIP_HTTP_GetExecute implementation.

The POST Method

The POST method transmits data after all the request headers have been sent. This data is not visible in the browser's address bar, and can only be seen with a packet capture tool. It does however use the same URL encoding method.

The HTTP server does not perform any preparsing of this data. All POST data is left in the TCP buffer, so the custom application will need to access the TCP buffer directly to retrieve and decode it. The functions TCPIP_HTTP_PostNameRead and TCPIP_HTTP_PostValueRead have been provided to assist with these requirements. However, these functions can only be used when at least entire variables are expected to fit in the TCP buffer at once. Most POST processing functions will be implemented as state machines to use these functions. There is a status variable per each connection that stores the current state accessible with TCPIP_HTTP_CurrentConnectionPostSmGet and TCPIP_HTTP_CurrentConnectionPostSmSet functions. This state machine variable is reset to zero with each new request. Functions should generally implement a state to read a variable name, and another to read an expected value. Additional states may be helpful depending on the application.

The following example form accepts an e-mail address, a subject, and a message body. Since this data will likely total over the size of the internal connection data buffer, it should be submitted via POST.

```
<form method="post" action="/email.htm">
```

```
To: <input type="text" name="to" maxlength="50" /><br />
Subject: <input type="text" name="subject" maxlength="50" /><br />
Message:<br />
<textarea name="msg" rows="6"></textarea><br />
<input type="submit" value="Send Message" /></div>
</form>
```

Suppose a user enters the following data into this form:

To: joe@picsaregood.com

Subject: Sent by a PIC

Message: I sent this message using my development board!

The TCPIP_HTTP_PostExecute function will be called with the following data still in the TCP buffer:

to=joe%40picsaregood.com&subject=Sent+by+a+PIC&msg=I+sent+this+message+using+my+development+board%21

To use the e-mail module, the application needs to read in the address and the subject, store those in RAM, and then send the message. However, since the message is not guaranteed to fit in RAM all at once, it must be read as space is available and passed to the e-mail module. A state machine, coupled with the TCPIP_HTTP_PostNameRead and

TCPIP_HTTP_PostValueRead functions can simplify this greatly.

See the TCPIP_HTTP_PostExecute and HTTPPostEmail functions in the supplied custom_http_app.c for a complete implementation of this example.

Authentication

The HTTP protocol provides a method for servers to request a user name and password from a client before granting access to a page. The HTTP server supports this authentication mechanism, allowing developers to require valid credentials for access to some or all pages.

Description

Authentication functionality is supported by two user-provided callback functions. The first, TCPIP_HTTP_FileAuthenticate, determines if the requested page requires valid credentials to proceed. The second, TCPIP_HTTP_UserAuthenticate, checks the user name and password against an accepted list and determines whether to grant or deny access. The connection stores the current authorization setting which can be manipulated by using the TCPIP_HTTP_CurrentConnectionIsAuthorizedGet and TCPIP_HTTP_CurrentConnectionIsAuthorizedSet functions.

Requiring Authentication

When a request is first made, the function TCPIP_HTTP_FileAuthenticate is called to determine if that page needs password protection. This function returns a value to instruct the HTTP server how to proceed. The most significant bit indicates whether or not access is granted. That is, values 0x80 and higher allow access unconditionally, while values 0x79 and lower will require a user name and password at a later point. The value returned is stored in the connection data so that it can be accessed by future callback functions.

The following example is the simplest case, in which all files require a password for access:

```
uint8_t TCPIP_HTTP_FileAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cFile)
{
return 0;
}
In some cases, only certain files will need to be protected. The second example requires a password for any
file located in the /treasure folder:
uint8_t TCPIP_HTTP_FileAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cFile)
{
    // Compare to "/treasure" folder. Don't use strcmp here, because
    // cFile has additional path info such as "/treasure/gold.htm"
    if(memcmp(cFile, (const void*)"treasure", 8) == 0)
    {   // Authentication will be needed later
        return 0;
    }

    // No authentication required
    return 0x80;
}
```

More complex uses could require an administrative user to access the `/admin` folder, while any authenticated user can access the rest of the site. This requires the TCPIP_HTTP_FileAuthenticate to return different authentication values for a different file.

Validating Credentials

The TCPIP_HTTP_UserAuthenticate function determines if the supplied user name and password are valid to access the specific resource. Again, the most significant bit indicates whether or not access is granted. The value returned is also stored in the connection internal data and it can be accessed by future callback functions.

The following example is the simplest case, in which one user/password pair is accepted for all pages:

```
uint8_t TCPIP_HTTP_UserAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cUser, uint8_t* cPass)
```

```
{
    if(!strcmp((char*)cUser, (const char*)"AliBaba") &&
    !strcmp((char*)cPass, (const char*)"Open Sesame!") )
    {
        return 0x80;
    }

    return 0;
}
```

More complex uses are certainly feasible. Many applications may choose to store the user names and passwords in protected files so that they may be updated by a privileged user. In some cases, you may have multiple users with various levels of access. The application may wish to return various values above 0x80 in TCPIP_HTTP_UserAuthenticate so that later callback functions can determine which user logged in.

Cookies

By design, HTTP is a session-less and state-less protocol; every connection is an independent session with no relation to another.

Description

Cookies were added to the protocol description to solve this problem. This feature allows a web server to store small bits of text in a user's browser. These values will be returned to the server with every request, allowing the server to associate session variables with a request. Cookies are typically used for more advanced authentication systems.

Best practice is generally to store the bulk of the data on the server, and store only a unique identifier with the browser. This cuts down on data overhead and ensures that the user cannot modify the values stored with the session. However, logic must be implemented in the server to expire old sessions and allocate memory for new ones. If sensitive data is being stored, it is also important that the identifier be random enough so as to prevent stealing or spoofing another user's cookies.

Retrieving Cookies

In the HTTP server, cookies are retrieved automatically. They are stored in the connection internal data buffer just as any other GET form argument or URL parameter would be. The proper place to parse these values is therefore in the TCPIP_HTTP_GetExecute callback using the TCPIP_HTTP_ArgGet.

This model consumes some of the limited space available for URL parameters. Ensure that cookies do not consume more space than is available (as defined by HTTP_MAX_DATA_LEN) and that they will fit after any data that may be submitted via a GET form. If enough space is not available, the cookies will be truncated.

Setting Cookies

Cookies can be set in TCPIP_HTTP_GetExecute or TCPIP_HTTP_PostExecute. To set a cookie, store the name/value pairs in the connection internal buffer data as a series of null-terminated strings. Then, call TCPIP_HTTP_CurrentConnectionHasArgsSet with a parameter equal to the number of name/value pairs to be set. For example, the following code sets a cookie indicating a user's preference for a type of cookie:

```
void TCPIP_HTTP_GetExecute(void)
{
...
// Set a cookie
uint8_t* connData = TCPIP_HTTP_CurrentConnectionDataBufferGet(connHandle);
uint16_t httpBuffSize  = TCPIP_HTTP_CurrentConnectionDataBufferSizeGet(connHandle);
if(strlen(“flavor”) + strlen(“oatmeal raisin) + 2 <= httpBuffSize)
{
  strcpy((char*)connData, "flavor");
  strcpy((char*)connData + strlen("flavor") + 1, "oatmeal raisin");
  TCPIP_HTTP_CurrentConnectionHasArgsSet(connHandle, 1);
}

...
}
```

After this, all future requests from this browser will include the parameter "flavor" in the connection data along with the associated value of "oatmeal raisin".

Compression

All modern web browsers can receive files encoded with GZIP compression. For static files (those without dynamic variables), this can decrease the amount of data transmitted by as much as 60%.

Description

The MPFS2 Utility will automatically determine which files can benefit from GZIP compression, and will store the compressed file in the MPFS2 image when possible. This generally includes all JavaScript and CSS files. (Images are typically already compressed, so the MPFS2 Utility will generally decide it is better to store them uncompressed.) This HTTP server will then seamlessly return this compressed file to the browser. Less

non-volatile storage space will be required for storing the pages image, and faster transfers back to the client will result. No special configuration is required for this feature.

To prevent certain extensions from being compressed, use the Advanced Settings dialog in the MPFS2 Utility.

Configuring the Library

Macros

	Name	Description
	TCPIP_STACK_USE_BASE64_DECODE	Authentication requires Base64 decoding Enable basic authentication support
	TCPIP_HTTP_CACHE_LEN	Max lifetime (sec) of static responses as string
	TCPIP_HTTP_CONFIG_FLAGS	Define the HTTP module configuration flags Use 0 for default See HTTP_MODULE_FLAGS definition for possible values
	TCPIP_HTTP_DEFAULT_FILE	Indicate what HTTP file to serve when no specific one is requested
	TCPIP_HTTP_DEFAULT_LEN	For buffer overrun protection. Set to longest length of above two strings.
	TCPIP_HTTP_MAX_CONNECTIONS	Maximum numbers of simultaneous supported HTTP connections.
	TCPIP_HTTP_MAX_DATA_LEN	Define the maximum data length for reading cookie and GET/POST arguments (bytes)
	TCPIP_HTTP_MAX_HEADER_LEN	Set to length of longest string above
	TCPIP_HTTP_MIN_CALLBACK_FREE	Define the minimum number of bytes free in the TX FIFO before executing callbacks
	TCPIP_HTTP_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_HTTP_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_HTTP_TIMEOUT	Max time (sec) to await more data before timing out and disconnecting the socket
	TCPIP_HTTP_USE_AUTHENTICATION	Enable basic authentication support
	TCPIP_HTTP_USE_COOKIES	Enable cookie support
	TCPIP_HTTP_USE_POST	Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support
	TCPIP_HTTPS_DEFAULT_FILE	Indicate what HTTPS file to serve when no specific one is requested
	TCPIP_HTTP_FILE_UPLOAD_ENABLE	Configure MPFS over HTTP updating Comment this line to disable updating via HTTP
	TCPIP_HTTP_FILE_UPLOAD_NAME	This is macro TCPIP_HTTP_FILE_UPLOAD_NAME.
	TCPIP_HTTP_TASK_RATE	The HTTP task rate, ms The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_HTTP_NO_AUTH_WITHOUT_SSL	Uncomment to require secure connection before requesting a password

Description

The configuration of the HTTP TCP/IP Stack is based on the file `tcpip_config.h` (which may include `http_config.h`).

This header file contains the configuration selection for the HTTP TCP/IP Stack. Based on the selections made, the HTTP TCP/IP Stack may support the selected features.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_STACK_USE_BASE64_DECODE Macro

File

http_config.h

C

```
#define TCPIP_STACK_USE_BASE64_DECODE
```

Description

Authentication requires Base64 decoding Enable basic authentication support

TCPIP_HTTP_CACHE_LEN Macro

File

http_config.h

C

```
#define TCPIP_HTTP_CACHE_LEN ("600")
```

Description

Max lifetime (sec) of static responses as string

TCPIP_HTTP_CONFIG_FLAGS Macro

File

http_config.h

C

```
#define TCPIP_HTTP_CONFIG_FLAGS 0
```

Description

Define the HTTP module configuration flags Use 0 for default See HTTP_MODULE_FLAGS definition for possible values

TCPIP_HTTP_DEFAULT_FILE Macro

File

http_config.h

C

```
#define TCPIP_HTTP_DEFAULT_FILE "index.htm"
```

Description

Indicate what HTTP file to serve when no specific one is requested

TCPIP_HTTP_DEFAULT_LEN Macro

File

http_config.h

C

```
#define TCPIP_HTTP_DEFAULT_LEN (10u)
```

Description

For buffer overrun protection. Set to longest length of above two strings.

TCPIP_HTTP_MAX_CONNECTIONS Macro

File

http_config.h

C

```
#define TCPIP_HTTP_MAX_CONNECTIONS (4)
```

Description

Maximum numbers of simultaneous supported HTTP connections.

TCPIP_HTTP_MAX_DATA_LEN Macro

File

http_config.h

C

```
#define TCPIP_HTTP_MAX_DATA_LEN (100u)
```

Description

Define the maximum data length for reading cookie and GET/POST arguments (bytes)

TCPIP_HTTP_MAX_HEADER_LEN Macro

File

http_config.h

C

```
#define TCPIP_HTTP_MAX_HEADER_LEN (15u)
```

Description

Set to length of longest string above

TCPIP_HTTP_MIN_CALLBACK_FREE Macro

File

http_config.h

C

```
#define TCPIP_HTTP_MIN_CALLBACK_FREE (16u)
```

Description

Define the minimum number of bytes free in the TX FIFO before executing callbacks

TCPIP_HTTP_SKT_RX_BUFF_SIZE Macro

File

http_config.h

C

```
#define TCPIP_HTTP_SKT_RX_BUFF_SIZE 0
```

Description

Define the size of the RX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

TCPIP_HTTP_SKT_TX_BUFF_SIZE Macro

File

http_config.h

C

```
#define TCPIP_HTTP_SKT_TX_BUFF_SIZE 0
```

Description

Define the size of the TX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

TCPIP_HTTP_TIMEOUT Macro

File

http_config.h

C

```
#define TCPIP_HTTP_TIMEOUT (45u)
```

Description

Max time (sec) to await more data before timing out and disconnecting the socket

TCPIP_HTTP_USE_AUTHENTICATION Macro

File

http_config.h

C

```
#define TCPIP_HTTP_USE_AUTHENTICATION
```

Description

Enable basic authentication support

TCPIP_HTTP_USE_COOKIES Macro

File

http_config.h

C

```
#define TCPIP_HTTP_USE_COOKIES
```

Description

Enable cookie support

TCPIP_HTTP_USE_POST Macro

File

http_config.h

C

```
#define TCPIP_HTTP_USE_POST
```

Description

Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support

TCPIP_HTTPS_DEFAULT_FILE Macro

File

http_config.h

C

```
#define TCPIP_HTTPS_DEFAULT_FILE "index.htm"
```

Description

Indicate what HTTPS file to serve when no specific one is requested

TCPIP_HTTP_FILE_UPLOAD_ENABLE Macro

File

http_config.h

C

```
#define TCPIP_HTTP_FILE_UPLOAD_ENABLE
```

Description

Configure MPFS over HTTP updating Comment this line to disable updating via HTTP

TCPIP_HTTP_FILE_UPLOAD_NAME Macro

File

http_config.h

C

```
#define TCPIP_HTTP_FILE_UPLOAD_NAME "mpfsupload"
```

Description

This is macro TCPIP_HTTP_FILE_UPLOAD_NAME.

TCPIP_HTTP_TASK_RATE Macro

File

http_config.h

C

```
#define TCPIP_HTTP_TASK_RATE 33
```

Description

The HTTP task rate, ms The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_HTTP_NO_AUTH_WITHOUT_SSL Macro

File

http_config.h

C

```
#define TCPIP_HTTP_NO_AUTH_WITHOUT_SSL
```

Description

Uncomment to require secure connection before requesting a password

Building the Library

This section lists the files that are available in the HTTP module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/http.c	HTTP Server implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The HTTP module depends on the following modules:

- TCP/IP Stack Library
- TCP Module
- File System Service Library

Library Interface

a) Functions

	Name	Description
	TCPIP_HTTP_ArgGet	Locates a form field value in a given data array.
	TCPIP_HTTP_CurrentConnectionByteCountDec	Decrements the connection byte count.
	TCPIP_HTTP_CurrentConnectionByteCountGet	Returns how many bytes have been read so far
	TCPIP_HTTP_CurrentConnectionByteCountSet	Sets how many bytes have been read so far.
	TCPIP_HTTP_CurrentConnectionCallbackPosGet	Returns the callback position indicator.
	TCPIP_HTTP_CurrentConnectionCallbackPosSet	Sets the callback position indicator.
	TCPIP_HTTP_CurrentConnectionHasArgsGet	Checks whether there are get or cookie arguments.
	TCPIP_HTTP_CurrentConnectionHasArgsSet	Sets whether there are get or cookie arguments.
	TCPIP_HTTP_CurrentConnectionPostSmSet	Set the POST state machine state.
	TCPIP_HTTP_CurrentConnectionStatusGet	Gets HTTP status.
	TCPIP_HTTP_FileAuthenticate	Determines if a given file name requires authentication
	TCPIP_HTTP_GetExecute	Processes GET form field variables and cookies.
	TCPIP_HTTP_PostExecute	Processes POST form variables and data.
	TCPIP_HTTP_Print_varname	Inserts dynamic content into a web page
	TCPIP_HTTP_UserAuthenticate	Performs validation on a specific user name and password.
	TCPIP_HTTP_CurrentConnectionDataBufferGet	Returns pointer to connection general purpose data buffer.
	TCPIP_HTTP_CurrentConnectionFileGet	Get handle to current connection's file.
	TCPIP_HTTP_CurrentConnectionIsAuthorizedGet	Gets the authorized state for the current connection.
	TCPIP_HTTP_CurrentConnectionIsAuthorizedSet	Sets the authorized state for the current connection.
	TCPIP_HTTP_CurrentConnectionPostSmGet	Get the POST state machine state.
	TCPIP_HTTP_CurrentConnectionSocketGet	Get the socket for the current connection.
	TCPIP_HTTP_CurrentConnectionStatusSet	Sets HTTP status.
	TCPIP_HTTP_CurrentConnectionUserDataGet	Gets the user data parameter for the current connection.
	TCPIP_HTTP_CurrentConnectionUserDataSet	Sets the user data parameter for the current connection.
	TCPIP_HTTP_FileInclude	Writes a file byte-for-byte to the currently loaded TCP socket.
	TCPIP_HTTP_PostNameRead	Reads a name from a URL encoded string in the TCP buffer.
	TCPIP_HTTP_PostValueRead	Reads a value from a URL encoded string in the TCP buffer.
	TCPIP_HTTP_URLDecode	Parses a string from URL encoding to plain-text.
	TCPIP_HTTP_PostReadPair	Reads a name and value pair from a URL encoded string in the TCP buffer.

	TCPIP_HTTP_Task	Standard TCP/IP stack module task function.
	TCPIP_HTTP_ActiveConnectionCountGet	Gets the number of active connections.
	TCPIP_HTTP_CurrentConnectionHandleGet	Gets the connection handle of a HTTP connection.
	TCPIP_HTTP_CurrentConnectionIndexGet	Gets the index of the HTTP connection.

b) Data Types and Constants

	Name	Description
	HTTP_CONN_HANDLE	HTTP connection identifier, handle of a HTTP connection
	HTTP_FILE_TYPE	File type definitions
	HTTP_IO_RESULT	Result states for execution callbacks
	HTTP_MODULE_FLAGS	HTTP module configuration flags Multiple flags can be OR-ed
	HTTP_READ_STATUS	Result states for TCPIP_HTTP_PostNameRead, TCPIP_HTTP_PostValueRead and TCPIP_HTTP_PostReadPair
	HTTP_STATUS	Supported Commands and Server Response Codes
	TCPIP_HTTP_MODULE_CONFIG	HTTP module dynamic configuration data

Description

This section describes the Application Programming Interface (API) functions of the HTTP module.

Refer to each section for a detailed description.

a) Functions

TCPIP_HTTP_ArgGet Function

Locates a form field value in a given data array.

File

http.h

C

```
const uint8_t* TCPIP_HTTP_ArgGet(const uint8_t* cData, const uint8_t* cArg);
```

Returns

A pointer to the argument value, or NULL if not found.

Description

This function searches through a data array to find the value associated with a given argument. It can be used to find form field values in data received over GET or POST.

The end of data is assumed to be reached when a null name parameter is encountered. This requires the string to have an even number of null-terminated strings, followed by an additional null terminator.

Remarks

None.

Preconditions

The data array has a valid series of null terminated name/value pairs.

Example

```
  void TCPIP_HTTP_Print_cookiename(HTTP_CONN_HANDLE connHandle)
  {
      const uint8_t *ptr;
      TCP_SOCKET sktHTTP;

      ptr = TCPIP_HTTP_ArgGet(TCPIP_HTTP_CurrentConnectionDataBufferGet(connHandle), (const
uint8_t*)"name");
      sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
      if(ptr)
          TCPIP_TCP_StringPut(sktHTTP, ptr);
      else
          TCPIP_TCP_StringPut(sktHTTP, (const uint8_t*)"not set");
```

```
}
```

Parameters

Parameters	Description
cData	the buffer to search
cArg	the name of the argument to find

Function

const uint8_t* TCPIP_HTTP_ArgGet(const uint8_t* cData, const uint8_t* cArg)

TCPIP_HTTP_CurrentConnectionByteCountDec Function

Decrements the connection byte count.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionByteCountDec(HTTP_CONN_HANDLE connHandle, uint32_t byteCount);
```

Returns

None.

Description

This function decrements the current value of the counter showing the number of bytes read from the connection so far.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
byteCount	byte count reduction

Function

void TCPIP_HTTP_CurrentConnectionByteCountDec(HTTP_CONN_HANDLE connHandle, uint32_t byteCount)

TCPIP_HTTP_CurrentConnectionByteCountGet Function

Returns how many bytes have been read so far

File

http.h

C

```
uint32_t TCPIP_HTTP_CurrentConnectionByteCountGet(HTTP_CONN_HANDLE connHandle);
```

Returns

Current connection byte count, how many bytes have been read so far.

Description

This function returns the current value of the counter showing the number of bytes read from the connection so far.

Remarks

None.

Preconditions

None.

Example

```
switch(TCPIP_HTTP_CurrentConnectionPostSmGet(connHandle))
{
    case SM_CFG_SNMP_READ_NAME:
        // If all parameters have been read, end
        if(TCPIP_HTTP_CurrentConnectionByteCountGet(connHandle) == 0u)
        {
            return HTTP_IO_DONE;
        }
     .
     .
     .
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint32_t TCPIP_HTTP_CurrentConnectionByteCountGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionByteCountSet Function

Sets how many bytes have been read so far.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionByteCountSet(HTTP_CONN_HANDLE connHandle, uint32_t byteCount);
```

Returns

None.

Description

This function sets the current value of the counter showing the number of bytes read from the connection so far.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
byteCount	byte count to be set

Function

void TCPIP_HTTP_CurrentConnectionByteCountSet(HTTP_CONN_HANDLE connHandle, uint32_t byteCount)

TCPIP_HTTP_CurrentConnectionCallbackPosGet Function

Returns the callback position indicator.

File

http.h

C

```
uint32_t TCPIP_HTTP_CurrentConnectionCallbackPosGet(HTTP_CONN_HANDLE connHandle);
```

Returns

Callback position indicator for connection defined by connHandle.

Description

This function will return the current value of the callback position indicator for the HTTP connection identified by connHandle. The callback position indicator is used in the processing of the HTTP dynamic variables.

Remarks

None.

Preconditions

None.

Example

```
  uint32_t callbackPos;
  callbackPos = TCPIP_HTTP_CurrentConnectionCallbackPosGet(connHandle);
  if(callbackPos == 0x00u)
      callbackPos = (uint32_t)DDNSClient.Host.szRAM;
  callbackPos = (uint32_t)TCPIP_TCP_StringPut(TCPIP_HTTP_CurrentConnectionSocketGet(connHandle),
(uint8_t*)callbackPos);
  if(*(uint8_t*)callbackPos == '0')
      callbackPos = 0x00;
  TCPIP_HTTP_CurrentConnectionCallbackPosSet(connHandle, callbackPos);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint32_t TCPIP_HTTP_CurrentConnectionCallbackPosGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionCallbackPosSet Function

Sets the callback position indicator.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionCallbackPosSet(HTTP_CONN_HANDLE connHandle, uint32_t callbackPos);
```

Returns

None.

Description

This function will set the current value of the callback position indicator for the HTTP connection identified by connHandle. The callback position indicator is used in the processing of the HTTP dynamic variables. When set to a value != 0, it indicates to the HTTP server that the application has more pending processing that needs to be done.

Remarks

None.

Preconditions

None.

Example

```
  void TCPIP_HTTP_Print_builddate(HTTP_CONN_HANDLE connHandle)
  {
      TCP_SOCKET sktHTTP;
      sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);

      TCPIP_HTTP_CurrentConnectionCallbackPosSet(connHandle, 0x01);
      if(TCPIP_TCP_PutIsReady(sktHTTP) < strlen((const char*)__DATE__" "__TIME__))
```

```
        { // Don't have room to output build date and time
            return;
        }
        TCPIP_HTTP_CurrentConnectionCallbackPosSet(connHandle, 0x00);
        TCPIP_TCP_StringPut(sktHTTP, (const void*)__DATE__" "__TIME__);
    }
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
callbackPos	new connection callback position value

Function

void TCPIP_HTTP_CurrentConnectionCallbackPosSet(HTTP_CONN_HANDLE connHandle, uint32_t callbackPos)

TCPIP_HTTP_CurrentConnectionHasArgsGet Function

Checks whether there are get or cookie arguments.

File

http.h

C

```
uint8_t TCPIP_HTTP_CurrentConnectionHasArgsGet(HTTP_CONN_HANDLE connHandle);
```

Returns

The current value of the connection hasArgs.

Description

The function will get the value of the "cookies or get arguments" that are present.

Remarks

None.

Preconditions

None.

Example

```
uint8_t hasArgs = TCPIP_HTTP_CurrentConnectionHasArgsGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint8_t TCPIP_HTTP_CurrentConnectionHasArgsGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionHasArgsSet Function

Sets whether there are get or cookie arguments.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionHasArgsSet(HTTP_CONN_HANDLE connHandle, uint8_t args);
```

Returns

None.

Description

The function sets the value of the "cookies or get arguments" that are present.

Remarks

None.

Preconditions

None.

Example

```
else if(!memcmp(filename, "cookies.htm", 11))
{
    TCPIP_HTTP_CurrentConnectionHasArgsSet(connHandle, true);
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
args	boolean if there are arguments or not

Function

void TCPIP_HTTP_CurrentConnectionHasArgsSet(HTTP_CONN_HANDLE connHandle, uint8_t args)

TCPIP_HTTP_CurrentConnectionPostSmSet Function

Set the POST state machine state.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionPostSmSet(HTTP_CONN_HANDLE connHandle, uint16_t state);
```

Returns

None.

Description

This function sets the POST state machine state for the connection defined by connHandle. This state is maintained by the HTTP connection and can be used by the user of the HTTP to maintain its own POST state machine. The values of the POST state machine have significance only for the user of the HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
uint8_t* httpDataBuff;
#define SM_POST_LCD_READ_NAME   1
#define SM_POST_LCD_READ_VALUE  2

switch(TCPIP_HTTP_CurrentConnectionPostSmGet(connHandle))
{
    // Find the name
    case SM_POST_LCD_READ_NAME:

        // Read a name
        if(TCPIP_HTTP_PostNameRead(connHandle, httpDataBuff, HTTP_MAX_DATA_LEN) == HTTP_READ_INCOMPLETE)
            return HTTP_IO_NEED_DATA;

        TCPIP_HTTP_CurrentConnectionPostSmSet(connHandle, SM_POST_LCD_READ_VALUE);
        // No break...continue reading value

    // Found the value, so store the LCD and return
    case SM_POST_LCD_READ_VALUE:
```

```
        .
        .
        .
    }
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
state	16 bit integer state for POST state machine

Function

void TCPIP_HTTP_CurrentConnectionPostSmSet(HTTP_CONN_HANDLE connHandle, uint16_t state)

TCPIP_HTTP_CurrentConnectionStatusGet Function

Gets HTTP status.

File

http.h

C

```
HTTP_STATUS TCPIP_HTTP_CurrentConnectionStatusGet(HTTP_CONN_HANDLE connHandle);
```

Returns

A HTTP_STATUS value.

Description

This function returns the current HTTP status of the selected HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
HTTP_STATUS currStat =  TCPIP_HTTP_CurrentConnectionStatusGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

HTTP_STATUS TCPIP_HTTP_CurrentConnectionStatusGet(HTTP_CONN_HANDLE connHandle);

TCPIP_HTTP_FileAuthenticate Function

Determines if a given file name requires authentication

File

http.h

C

```
uint8_t TCPIP_HTTP_FileAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cFile);
```

Returns

- <= 0x79 - valid authentication is required
- >= 0x80 - access is granted for this connection

Description

This function is implemented by the application developer. Its function is to determine if a file being requested requires authentication to view. The user name and password, if supplied, will arrive later with the request headers, and will be processed at that time.

Return values 0x80 - 0xff indicate that authentication is not required, while values from 0x00 to 0x79 indicate that a user name and password are required before proceeding. While most applications will only use a single value to grant access and another to require authorization, the range allows multiple "realms" or sets of pages to be protected, with different credential requirements for each.

The return value of this function is saved for the current connection and can be read using TCPIP_HTTP_CurrentConnectionIsAuthorizedGet(). It will be available to future callbacks, including TCPIP_HTTP_UserAuthenticate and any of the TCPIP_HTTP_GetExecute, TCPIP_HTTP_PostExecute, or TCPIP_HTTP_Print_varname callbacks.

Remarks

This function may NOT write to the TCP buffer.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cFile	the name of the file being requested

Function

uint8_t TCPIP_HTTP_FileAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cFile)

TCPIP_HTTP_GetExecute Function

Processes GET form field variables and cookies.

File

http.h

C

```
HTTP_IO_RESULT TCPIP_HTTP_GetExecute(HTTP_CONN_HANDLE connHandle);
```

Returns

- HTTP_IO_DONE - application is done processing
- HTTP_IO_NEED_DATA - this value may not be returned because more data will not become available
- HTTP_IO_WAITING - the application is waiting for an asynchronous process to complete, and this function should be called again later

Description

This function is implemented by the application developer. Its purpose is to parse the data received from URL parameters (GET method forms) and cookies and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated.

When this function is called, the connection data buffer (see TCPIP_HTTP_CurrentConnectionDataBufferGet()) contains sequential name/value pairs of strings representing the data received. In this format, TCPIP_HTTP_ArgGet can be used to search for specific variables in the input. If data buffer space associated with this connection is required, connection data buffer may be overwritten here once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including TCPIP_HTTP_PostExecute and any TCPIP_HTTP_Print_varname dynamic substitutions.

This function may also issue redirections by setting the connection data buffer to the destination file name or URL, and the connection httpStatus (TCPIP_HTTP_CurrentConnectionStatusSet()) to HTTP_REDIRECT.

Finally, this function may set cookies. Set connection data buffer to a series of name/value string pairs (in the same format in which parameters arrive) and then set the connection hasArgs (TCPIP_HTTP_CurrentConnectionHasArgsSet()) equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in the connection data buffer.

Remarks

This function is only called if variables are received via URL parameters or Cookie arguments. This function may NOT write to the TCP buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use the connection callbackPos (TCPIP_HTTP_CurrentConnectionCallbackPosGet()) or the connection data buffer for storage associated with individual requests.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

HTTP_IO_RESULT TCPIP_HTTP_GetExecute(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_PostExecute Function

Processes POST form variables and data.

File

http.h

C

```
HTTP_IO_RESULT TCPIP_HTTP_PostExecute(HTTP_CONN_HANDLE connHandle);
```

Returns

- HTTP_IO_DONE - application is done processing
- HTTP_IO_NEED_DATA - more data is needed to continue, and this function should be called again later
- HTTP_IO_WAITING - the application is waiting for an asynchronous process to complete, and this function should be called again later

Description

This function is implemented by the application developer. Its purpose is to parse the data received from POST forms and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated before this function is called.

When this function is called, POST data will be waiting in the TCP buffer. The connection byteCount (see TCPIP_HTTP_CurrentConnectionByteCountGet will indicate the number of bytes remaining to be received before the browser request is complete.

Since data is still in the TCP buffer, the application must call TCPIP_TCP_ArrayGet in order to retrieve bytes. When this is done, connection byteCount MUST be updated to reflect how many bytes now remain. The functions TCPIP_TCP_ArrayFind and TCPIP_TCP_Find may be helpful to locate data in the TCP buffer.

In general, data submitted from web forms via POST is URL encoded. The TCPIP_HTTP_URLDecode function can be used to decode this information back to a standard string if required. If data buffer space associated with this connection is required, the connection data buffer (see TCPIP_HTTP_CurrentConnectionDataBufferGet()) may be overwritten here once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including TCPIP_HTTP_PostExecute and any TCPIP_HTTP_Print_varname dynamic substitutions.

Whenever a POST form is processed it is recommended to issue a redirect back to the browser, either to a status page or to the same form page that was posted. This prevents accidental duplicate submissions (by clicking refresh or back/forward) and avoids browser warnings about "resubmitting form data". Redirects may be issued to the browser by setting the connection data buffer to the destination file or URL, and the connection httpStatus (TCPIP_HTTP_CurrentConnectionStatusSet()) to HTTP_REDIRECT.

Finally, this function may set cookies. Set the connection data buffer to a series of name/value string pairs (in the same format in which parameters arrive), and then set the connection hasArgs (TCPIP_HTTP_CurrentConnectionHasArgsSet) equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in the connection data buffer.

Remarks

This function is only called when the request method is POST, and is only used when HTTP_USE_POST is defined. This method may NOT write to the TCP buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use the connection callbackPos (TCPIP_HTTP_CurrentConnectionCallbackPosGet) or connection data buffer for storage associated with individual requests.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

HTTP_IO_RESULT TCPIP_HTTP_PostExecute(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_Print_varname Function

Inserts dynamic content into a web page

File

http.h

C

```
void TCPIP_HTTP_Print_varname(HTTP_CONN_HANDLE connHandle, uint16_t wParam1, uint16_t wParam2, ...);
```

Returns

None.

Description

Functions in this style are implemented by the application developer. These functions generate dynamic content to be inserted into web pages and other files returned by the HTTP server.

Functions of this type are called when a dynamic variable is located in a web page. (i.e., ~varname~) The name between the tilde '~' characters is appended to the base function name. In this example, the callback would be named TCPIP_HTTP_Print_varname.

The function prototype is located in your project's http_print.h, which is automatically generated by the mpfs2.jar utility. The prototype will have uint16_t parameters included for each parameter passed in the dynamic variable. For example, the variable "~myArray(2,6)~" will generate the prototype "void TCPIP_HTTP_Print_varname(uint16_t, uint16_t);".

When called, this function should write its output directly to the TCP socket using any combination of TCPIP_TCP_PutIsReady, TCPIP_TCP_Put, TCPIP_TCP_ArrayPut, TCPIP_TCP_StringPut, TCPIP_TCP_ArrayPut, and TCPIP_TCP_StringPut.

Before calling, the HTTP server guarantees that at least HTTP_MIN_CALLBACK_FREE bytes (defaults to 16 bytes) are free in the output buffer. If the function is writing less than this amount, it should simply write the data to the socket and return.

In situations where a function needs to write more this amount, it must manage its output state using the connection callbackPos (TCPIP_HTTP_CurrentConnectionCallbackPosGet/TCPIP_HTTP_CurrentConnectionCallbackPosSet). This value will be set to zero before the function is called. If the function is managing its output state, it must set this to a non-zero value before returning. Typically this is used to track how many bytes have been written, or how many remain to be written. If the connection callbackPos is non-zero, the function will be called again when more buffer space is available. Once the callback completes, set this value back to zero to resume normal servicing of the request.

Remarks

This function may service multiple HTTP requests simultaneously, especially when managing its output state. Exercise caution when using global or static variables inside this routine. Use the connection callbackPos or the connection data buffer for storage associated with individual requests.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
wParam1	first parameter passed in the dynamic variable (if any)
wParam2	second parameter passed in the dynamic variable (if any)
...	additional parameters as necessary

Function

void TCPIP_HTTP_Print_varname(HTTP_CONN_HANDLE connHandle)

void TCPIP_HTTP_Print_varname(HTTP_CONN_HANDLE connHandle, uint16_t wParam1)

void TCPIP_HTTP_Print_varname(HTTP_CONN_HANDLE connHandle, uint16_t wParam1,

uint16_t wParam2, ...)

TCPIP_HTTP_UserAuthenticate Function

Performs validation on a specific user name and password.

File

http.h

C

```
uint8_t TCPIP_HTTP_UserAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cUser, uint8_t* cPass);
```

Returns

- <= 0x79 - the credentials were rejected
- >= 0x80 - access is granted for this connection

Description

This function is implemented by the application developer. Its function is to determine if the user name and password supplied by the client are acceptable for this resource. This callback function can thus to determine if only specific user names or passwords will be accepted for this resource.

Return values 0x80 - 0xff indicate that the credentials were accepted, while values from 0x00 to 0x79 indicate that authorization failed. While most applications will only use a single value to grant access, flexibility is provided to store multiple values in order to indicate which user (or user's group) logged in.

The value returned by this function is stored in the corresponding connection data and will be available with TCPIP_HTTP_CurrentConnectionIsAuthorizedGet in any of the TCPIP_HTTP_GetExecute, TCPIP_HTTP_PostExecute, or TCPIP_HTTP_Print_varname callbacks.

Remarks

This function is only called when an Authorization header is encountered.

This function may NOT write to the TCP buffer.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cUser	the user name supplied by the client
cPass	the password supplied by the client

Function

uint8_t TCPIP_HTTP_UserAuthenticate(HTTP_CONN_HANDLE connHandle, uint8_t* cUser, uint8_t* cPass)

TCPIP_HTTP_CurrentConnectionDataBufferGet Function

Returns pointer to connection general purpose data buffer.

File

http.h

C

```
uint8_t* TCPIP_HTTP_CurrentConnectionDataBufferGet(HTTP_CONN_HANDLE connHandle);
```

Returns

Pointer to the connection's general purpose data buffer.

Description

This function returns a pointer to the HTTP connection internal data buffer. This gives access to the application to the data that's stored in the HTTP connection buffer.

Remarks

None.

Preconditions

None.

Example

```
    void TCPIP_HTTP_Print_cookiename(HTTP_CONN_HANDLE connHandle)
    {
        const uint8_t *ptr;
        TCP_SOCKET sktHTTP;

        ptr = TCPIP_HTTP_ArgGet(TCPIP_HTTP_CurrentConnectionDataBufferGet(connHandle), (const
uint8_t*)"name");
        sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
        if(ptr)
            TCPIP_TCP_StringPut(sktHTTP, ptr);
        else
```

```
        TCPIP_TCP_StringPut(sktHTTP, (const uint8_t*)"not set");
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint8_t* TCPIP_HTTP_CurrentConnectionDataBufferGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionFileGet Function

Get handle to current connection's file.

File

http.h

C

```
SYS_FS_HANDLE TCPIP_HTTP_CurrentConnectionFileGet(HTTP_CONN_HANDLE connHandle);
```

Returns

Handle to File System file belonging to the connection defined by connHandle

Description

This function returns the handle of the current HTTP connection file.

Remarks

None.

Preconditions

None.

Example

```
uint8_t myBuff[20];

// Get the file handle and read from that file
SYS_FS_FileRead(myBuff, sizeof(myBuff), TCPIP_HTTP_CurrentConnectionFileGet(connHandle));
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

FILE_HANDLE TCPIP_HTTP_CurrentConnectionFileGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionIsAuthorizedGet Function

Gets the authorized state for the current connection.

File

http.h

C

```
uint8_t TCPIP_HTTP_CurrentConnectionIsAuthorizedGet(HTTP_CONN_HANDLE connHandle);
```

Returns

A uint8_t representing the authorization status.

Description

This function returns the authorization status for the current HTTP connection. This is one of the values returned by the TCPIP_HTTP_FileAuthenticate() function.

Remarks

None.

Preconditions

None.

Example

```
uint8_t isAuth;

isAuth = TCPIP_HTTP_CurrentConnectionIsAuthorizedGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint8_t TCPIP_HTTP_CurrentConnectionIsAuthorizedGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionIsAuthorizedSet Function

Sets the authorized state for the current connection.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionIsAuthorizedSet(HTTP_CONN_HANDLE connHandle, uint8_t auth);
```

Returns

None.

Description

This function sets the authorization status for the current HTTP connection. This has to be one of the values in the set returned by the TCPIP_HTTP_FileAuthenticate() function.

Remarks

None.

Preconditions

None.

Example

```
uint8_t auth = 0x80;

TCPIP_HTTP_CurrentConnectionIsAuthorizedSet(connHandle, auth);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
auth	new authorization state

Function

void TCPIP_HTTP_CurrentConnectionIsAuthorizedSet(HTTP_CONN_HANDLE connHandle, uint8_t auth)

TCPIP_HTTP_CurrentConnectionPostSmGet Function

Get the POST state machine state.

File

http.h

C

```
uint16_t TCPIP_HTTP_CurrentConnectionPostSmGet(HTTP_CONN_HANDLE connHandle);
```

Returns

16-bit integer POST state machine state.

Description

This function returns the POST state machine state for the connection defined by connHandle. This state is maintained by the HTTP connection and can be used by the user of the HTTP to maintain its own POST state machine. The values of the POST state machine have significance only for the user of the HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
#define SM_POST_LCD_READ_NAME   1
#define SM_POST_LCD_READ_VALUE  2

switch(TCPIP_HTTP_CurrentConnectionPostSmGet(connHandle))
{
    // Find the name
    case SM_POST_LCD_READ_NAME:
      .
      .
      .
    // Found the value, so store the LCD and return
    case SM_POST_LCD_READ_VALUE:
      .
      .
      .
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint16_t TCPIP_HTTP_CurrentConnectionPostSmGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionSocketGet Function

Get the socket for the current connection.

File

http.h

C

```
TCP_SOCKET TCPIP_HTTP_CurrentConnectionSocketGet(HTTP_CONN_HANDLE connHandle);
```

Returns

TCP_SOCKET for the connection defined by connHandle.

Description

The function returns the TCP socket of the specified HTTP connection. The user gets access to the connection socket which it can use for sending/reading data.

Remarks

None.

Preconditions

None.

Example

```
uint32_t byteCount;
TCP_SOCKET sktHTTP;

byteCount = TCPIP_HTTP_CurrentConnectionByteCountGet(connHandle);
sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
if(byteCount > TCPIP_TCP_GetIsReady(sktHTTP) + TCPIP_TCP_FifoRxFreeGet(sktHTTP))
{   // Configuration Failure
    TCPIP_HTTP_CurrentConnectionStatusSet(connHandle, HTTP_REDIRECT);
    return HTTP_IO_DONE;
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

TCP_SOCKET TCPIP_HTTP_CurrentConnectionSocketGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionStatusSet Function

Sets HTTP status.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionStatusSet(HTTP_CONN_HANDLE connHandle, HTTP_STATUS stat);
```

Returns

None.

Description

Allows write access to the HTTP status of the selected HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
byteCount = TCPIP_HTTP_CurrentConnectionByteCountGet(connHandle);
sktHTTP = TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);
if(byteCount > TCPIP_TCP_GetIsReady(sktHTTP) + TCPIP_TCP_FifoRxFreeGet(sktHTTP))
{   // Configuration Failure
    // 302 Redirect will be returned
    TCPIP_HTTP_CurrentConnectionStatusSet(connHandle, HTTP_REDIRECT);
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
stat	new HTTP_STATUS enumeration value.

Function

void TCPIP_HTTP_CurrentConnectionStatusSet(HTTP_CONN_HANDLE connHandle, HTTP_STATUS stat)

TCPIP_HTTP_CurrentConnectionUserDataGet Function

Gets the user data parameter for the current connection.

File

http.h

C

```
const void* TCPIP_HTTP_CurrentConnectionUserDataGet(HTTP_CONN_HANDLE connHandle);
```

Returns

User data that's stored as part of the connection.

Description

This function returns the user data value for the current HTTP connection. This data belongs to the user and is not used in any way by the HTTP server module. It can be set by the user with TCPIP_HTTP_CurrentConnectionUserDataSet().

Remarks

None.

Preconditions

None.

Example

```
uint32_t myConnData;

myConnData = (uint32_t)TCPIP_HTTP_CurrentConnectionUserDataGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

const void* TCPIP_HTTP_CurrentConnectionUserDataGet(HTTP_CONN_HANDLE connHandle)

TCPIP_HTTP_CurrentConnectionUserDataSet Function

Sets the user data parameter for the current connection.

File

http.h

C

```
void TCPIP_HTTP_CurrentConnectionUserDataSet(HTTP_CONN_HANDLE connHandle, const void* uData);
```

Returns

None.

Description

This function will set the user data value for the current HTTP connection. This data belongs to the user and is not used in any way by the HTTP server module. It is available to the user by calling TCPIP_HTTP_CurrentConnectionUserDataGet.

Remarks

None.

Preconditions

None.

Example

```
uint32_t myConnData;
```

```
TCPIP_HTTP_CurrentConnectionUserDataSet(connHandle, (const void*)myConnData);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
uData	user supplied data

Function

void TCPIP_HTTP_CurrentConnectionUserDataSet(HTTP_CONN_HANDLE connHandle, const void* uData)

TCPIP_HTTP_FileInclude Function

Writes a file byte-for-byte to the currently loaded TCP socket.

File

http.h

C

```
void TCPIP_HTTP_FileInclude(HTTP_CONN_HANDLE connHandle, const uint8_t* cFile);
```

Returns

None.

Description

This function allows an entire file to be included as a dynamic variable, providing a basic templating system for HTML web pages. This reduces unneeded duplication of visual elements such as headers, menus, etc.

When pHttpCon->callbackPos is 0, the file is opened and as many bytes as possible are written. The current position is then saved to pHttpCon->callbackPos and the file is closed. On subsequent calls, reading begins at the saved location and continues. Once the end of the input file is reached, pHttpCon->callbackPos is set back to 0 to indicate completion.

Remarks

Users should not call this function directly, but should instead add dynamic variables in the form of `~inc:filename.ext~` in their HTML code to include (for example) the file "filename.ext" at that specified location. The mpfs2.jar utility will handle the rest.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cFile	the name of the file to be sent

Function

void TCPIP_HTTP_FileInclude(HTTP_CONN_HANDLE connHandle, const uint8_t* cFile)

TCPIP_HTTP_PostNameRead Function

Reads a name from a URL encoded string in the TCP buffer.

File

http.h

C

```
HTTP_READ_STATUS TCPIP_HTTP_PostNameRead(HTTP_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen);
```

Returns

- HTTP_READ_OK - name was successfully read
- HTTP_READ_TRUNCTATED - entire name could not fit in the buffer, so the value was truncated and data has been lost
- HTTP_READ_INCOMPLETE - entire name was not yet in the buffer, so call this function again later to retrieve

Description

This function reads a name from a URL encoded string in the TCP buffer. This function is meant to be called from an TCPIP_HTTP_PostExecute callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least two extra bytes are needed in cData over the maximum length of data expected to be read.

This function will read until the next '=' character, which indicates the end of a name parameter. It assumes that the front of the buffer is the beginning of the name parameter to be read.

This function properly updates pHttpCon->byteCount by decrementing it by the number of bytes read. It also removes the delimiting '=' from the buffer.

Remarks

None.

Preconditions

The front of the TCP buffer is the beginning of a name parameter, and the rest of the TCP buffer contains a URL-encoded string with a name parameter terminated by a '=' character.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the name once it is read
wLen	how many bytes can be written to cData

Function

HTTP_READ_STATUS TCPIP_HTTP_PostNameRead(HTTP_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen)

TCPIP_HTTP_PostValueRead Function

Reads a value from a URL encoded string in the TCP buffer.

File

http.h

C

```
HTTP_READ_STATUS TCPIP_HTTP_PostValueRead(HTTP_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen);
```

Returns

- HTTP_READ_OK - value was successfully read
- HTTP_READ_TRUNCTATED - entire value could not fit in the buffer, so the value was truncated and data has been lost
- HTTP_READ_INCOMPLETE - entire value was not yet in the buffer, so call this function again later to retrieve

Description

This function reads a value from a URL encoded string in the TCP buffer. This function is meant to be called from an TCPIP_HTTP_PostExecute callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least 2 extra bytes are needed in cData above the maximum length of data expected to be read.

This function will read until the next '&' character, which indicates the end of a value parameter. It assumes that the front of the buffer is the beginning of the value parameter to be read. If pHttpCon->byteCount indicates that all expected bytes are in the buffer, it assumes that all remaining data is the value and acts accordingly.

This function properly updates pHttpCon->byteCount by decrementing it by the number of bytes read. The terminating '&' character is also removed from the buffer.

Remarks

None.

Preconditions

The front of the TCP buffer is the beginning of a name parameter, and the rest of the TCP buffer contains a URL-encoded string with a name parameter terminated by a '=' character.

Parameters

Parameters	Description
connHandle	HTTP connection handle

cData	where to store the value once it is read
wLen	how many bytes can be written to cData

Function

HTTP_READ_STATUS TCPIP_HTTP_PostValueRead(HTTP_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen)

TCPIP_HTTP_URLDecode Function

Parses a string from URL encoding to plain-text.

File

http.h

C

```
uint8_t* TCPIP_HTTP_URLDecode(uint8_t* cData);
```

Returns

A pointer to the last null terminator in data, which is also the first free byte for new data.

Description

This function parses a string from URL encoding to plain-text. The following conversions are made: '=' to '0', '&' to '0', '+' to ' ', and "%xx" to a single hex byte.

After completion, the data has been decoded and a null terminator signifies the end of a name or value. A second null terminator (or a null name parameter) indicates the end of all the data.

Remarks

This function is called by the stack to parse GET arguments and cookie data. User applications can use this function to decode POST data, but first need to verify that the string is null-terminated.

Preconditions

The data parameter is null terminated and has at least one extra byte free.

Parameters

Parameters	Description
cData	The string which is to be decoded in place.

Function

uint8_t* TCPIP_HTTP_URLDecode(uint8_t* cData)

TCPIP_HTTP_PostReadPair Macro

Reads a name and value pair from a URL encoded string in the TCP buffer.

File

http.h

C

```
#define TCPIP_HTTP_PostReadPair(connHandle, cData, wLen) TCPIP_HTTP_PostValueRead(connHandle, cData, wLen)
```

Returns

- HTTP_READ_OK - name and value were successfully read
- HTTP_READ_TRUNCTATED - entire name and value could not fit in the buffer, so input was truncated and data has been lost
- HTTP_READ_INCOMPLETE - entire name and value was not yet in the buffer, so call this function again later to retrieve

Description

Reads a name and value pair from a URL encoded string in the TCP buffer. This function is meant to be called from an TCPIP_HTTP_PostExecute callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least 2 extra bytes are needed in cData over the maximum length of data expected to be read.

This function will read until the next '&' character, which indicates the end of a value parameter. It assumes that the front of the buffer is the beginning of the name parameter to be read.

This function properly updates the connection byteCount (see TCPIP_HTTP_CurrentConnectionByteCountGet()) by decrementing it by the number of bytes read. It also removes the delimiting '&' from the buffer.

Once complete, two strings will exist in the cData buffer. The first is the parameter name that was read, while the second is the associated value.

Remarks

This function is aliased to TCPIP_HTTP_PostValueRead, since they effectively perform the same task. The name is provided only for completeness.

Preconditions

The front of the TCP buffer is the beginning of a name parameter, and the rest of the TCP buffer contains a URL-encoded string with a name parameter terminated by a '=' character and a value parameter terminated by a '&'.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the name and value strings once they are read
wLen	how many bytes can be written to cData

Function

HTTP_READ_STATUS TCPIP_HTTP_PostReadPair(HTTP_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen)

TCPIP_HTTP_Task Function

Standard TCP/IP stack module task function.

File

http.h

C

```
void TCPIP_HTTP_Task();
```

Returns

None.

Description

This function performs HTTP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The HTTP module should have been initialized.

Function

void TCPIP_HTTP_Task(void)

TCPIP_HTTP_ActiveConnectionCountGet Function

Gets the number of active connections.

File

http.h

C

```
int TCPIP_HTTP_ActiveConnectionCountGet(int* pOpenCount);
```

Returns

The number of active and total connections.

Description

This function will return the number of active and total HTTP connections at the current time.

Remarks

The value returned by this function is informational only. The number of active connections changes dynamically.

Preconditions

None.

Example

```
int nConns;

nConns = TCPIP_HTTP_ActiveConnectionCountGet(0);
```

Parameters

Parameters	Description
pOpenCount	address to store the number of total opened connections Could be NULL if not needed.

Function

int TCPIP_HTTP_ActiveConnectionCountGet(int* pOpenCount);

TCPIP_HTTP_CurrentConnectionHandleGet Function

Gets the connection handle of a HTTP connection.

File

http.h

C

```
HTTP_CONN_HANDLE TCPIP_HTTP_CurrentConnectionHandleGet(int connIx);
```

Returns

A valid connection handle if the connection index is valid 0 if there is no such connection

Description

This function will return the connection handle of the requested HTTP connection index.

Remarks

None

Preconditions

None.

Example

```
HTTP_CONN_HANDLE connHandle;

connHandle = TCPIP_HTTP_CurrentConnectionHandleGet(0);
```

Parameters

Parameters	Description
connIx	the HTTP connection ix.

Function

HTTP_CONN_HANDLE TCPIP_HTTP_CurrentConnectionHandleGet(int connIx);

TCPIP_HTTP_CurrentConnectionIndexGet Function

Gets the index of the HTTP connection.

File

http.h

C

```
int TCPIP_HTTP_CurrentConnectionIndexGet(HTTP_CONN_HANDLE connHandle);
```

Returns

The connection index.

Description

This function will return the index of the requested HTTP connection.

Remarks

None

Preconditions

None.

Example

```
int connIx;

connIx = TCPIP_HTTP_CurrentConnectionIndexGet(connHandle);
```

Parameters

Parameters	Description
connHandle	the HTTP connection handle.

Function

int TCPIP_HTTP_CurrentConnectionIndexGet(HTTP_CONN_HANDLE connHandle);

b) Data Types and Constants

HTTP_CONN_HANDLE Type

File

http.h

C

```
typedef const void* HTTP_CONN_HANDLE;
```

Description

HTTP connection identifier, handle of a HTTP connection

HTTP_FILE_TYPE Enumeration

File

http.h

C

```
typedef enum {
  HTTP_TXT = 0u,
  HTTP_HTM,
  HTTP_HTML,
  HTTP_CGI,
  HTTP_XML,
  HTTP_CSS,
  HTTP_GIF,
  HTTP_PNG,
  HTTP_JPG,
  HTTP_JS,
  HTTP_JAVA,
  HTTP_WAV,
  HTTP_UNKNOWN
} HTTP_FILE_TYPE;
```

Members

Members	Description
HTTP_TXT = 0u	File is a text document
HTTP_HTM	File is HTML (extension .htm)
HTTP_HTML	File is HTML (extension .html)
HTTP_CGI	File is HTML (extension .cgi)
HTTP_XML	File is XML (extension .xml)
HTTP_CSS	File is stylesheet (extension .css)
HTTP_GIF	File is GIF image (extension .gif)
HTTP_PNG	File is PNG image (extension .png)
HTTP_JPG	File is JPG image (extension .jpg)
HTTP_JS	File is java script (extension .js)
HTTP_JAVA	File is java (extension .class)
HTTP_WAV	File is audio (extension .wav)
HTTP_UNKNOWN	File type is unknown

Description

File type definitions

HTTP_IO_RESULT Enumeration

File

http.h

C

```
typedef enum {
  HTTP_IO_DONE = 0u,
  HTTP_IO_NEED_DATA,
  HTTP_IO_WAITING
} HTTP_IO_RESULT;
```

Members

Members	Description
HTTP_IO_DONE = 0u	Finished with procedure
HTTP_IO_NEED_DATA	More data needed to continue, call again later
HTTP_IO_WAITING	Waiting for asynchronous process to complete, call again later

Description

Result states for execution callbacks

HTTP_MODULE_FLAGS Enumeration

File

http.h

C

```
typedef enum {
  HTTP_MODULE_FLAG_DEFAULT = 0x00,
  HTTP_MODULE_FLAG_ADJUST_SKT_FIFOS = 0x01,
  HTTP_MODULE_FLAG_NO_DELAY = 0x02
} HTTP_MODULE_FLAGS;
```

Members

Members	Description
HTTP_MODULE_FLAG_DEFAULT = 0x00	Default flags value
HTTP_MODULE_FLAG_ADJUST_SKT_FIFOS = 0x01	Adjust corresponding socket FIFO at run time. Improves throughput when the socket buffers are small.
HTTP_MODULE_FLAG_NO_DELAY = 0x02	Create the HTTP sockets with NO-DELAY option. It will flush data as soon as possible.

Description

HTTP module configuration flags Multiple flags can be OR-ed

HTTP_READ_STATUS Enumeration

File

http.h

C

```
typedef enum {
  HTTP_READ_OK = 0u,
  HTTP_READ_TRUNCATED,
  HTTP_READ_INCOMPLETE
} HTTP_READ_STATUS;
```

Members

Members	Description
HTTP_READ_OK = 0u	Read was successful
HTTP_READ_TRUNCATED	Buffer overflow prevented by truncating value
HTTP_READ_INCOMPLETE	Entire object is not yet in the buffer. Try again later.

Description

Result states for TCPIP_HTTP_PostNameRead, TCPIP_HTTP_PostValueRead and TCPIP_HTTP_PostReadPair

HTTP_STATUS Enumeration

File

http.h

C

```
typedef enum {
  HTTP_GET = 0u,
  HTTP_POST,
  HTTP_BAD_REQUEST,
  HTTP_UNAUTHORIZED,
  HTTP_NOT_FOUND,
  HTTP_OVERFLOW,
  HTTP_INTERNAL_SERVER_ERROR,
  HTTP_NOT_IMPLEMENTED,
  HTTP_REDIRECT,
  HTTP_SSL_REQUIRED,
  HTTP_MPFS_FORM,
  HTTP_MPFS_UP,
  HTTP_MPFS_OK,
  HTTP_MPFS_WAIT,
  HTTP_MPFS_ERROR
} HTTP_STATUS;
```

Members

Members	Description
HTTP_GET = 0u	GET command is being processed
HTTP_POST	POST command is being processed
HTTP_BAD_REQUEST	400 Bad Request will be returned
HTTP_UNAUTHORIZED	401 Unauthorized will be returned
HTTP_NOT_FOUND	404 Not Found will be returned
HTTP_OVERFLOW	414 Request-URI Too Long will be returned
HTTP_INTERNAL_SERVER_ERROR	500 Internal Server Error will be returned
HTTP_NOT_IMPLEMENTED	501 Not Implemented (not a GET or POST command)
HTTP_REDIRECT	302 Redirect will be returned
HTTP_SSL_REQUIRED	403 Forbidden is returned, indicating SSL is required
HTTP_MPFS_FORM	Show the MPFS Upload form

HTTP_MPFS_UP	An MPFS Upload is being processed
HTTP_MPFS_OK	An MPFS Upload was successful
HTTP_MPFS_WAIT	An MPFS Upload waiting for the write operation to complete
HTTP_MPFS_ERROR	An MPFS Upload was not a valid image

Description

Supported Commands and Server Response Codes

TCPIP_HTTP_MODULE_CONFIG Structure

File

http.h

C

```
typedef struct {
  uint16_t nConnections;
  uint16_t nTlsConnections;
  uint16_t dataLen;
  uint16_t sktTxBuffSize;
  uint16_t sktRxBuffSize;
  uint16_t tlsSktTxBuffSize;
  uint16_t tlsSktRxBuffSize;
  uint16_t configFlags;
} TCPIP_HTTP_MODULE_CONFIG;
```

Members

Members	Description
uint16_t nConnections;	number of simultaneous HTTP connections allowed
uint16_t nTlsConnections;	Not used in the current implementation; Number of simultaneous HTTPS connections allowed
uint16_t dataLen;	size of the data buffer for reading cookie and GET/POST arguments (bytes)
uint16_t sktTxBuffSize;	size of TX buffer for the associated socket; leave 0 for default
uint16_t sktRxBuffSize;	size of RX buffer for the associated socket; leave 0 for default
uint16_t tlsSktTxBuffSize;	Not used in the current implementation; Size of TLS TX buffer for the associated socket; leave 0 for default (min 512 bytes)
uint16_t tlsSktRxBuffSize;	Not used in the current implementation; Size of TLS RX buffer for the associated socket; leave 0 for default (min 512 bytes)
uint16_t configFlags;	a HTTP_MODULE_FLAGS value.

Description

HTTP module dynamic configuration data

Files

Files

Name	Description
http.h	The HTTP web server module together with a file system (SYS_FS) allow the board to act as a web server.
http_config.h	HTTP configuration file

Description

This section lists the source and header files used by the library.

http.h

The HTTP web server module together with a file system (SYS_FS) allow the board to act as a web server.

Enumerations

	Name	Description
	HTTP_FILE_TYPE	File type definitions

	HTTP_IO_RESULT	Result states for execution callbacks
	HTTP_MODULE_FLAGS	HTTP module configuration flags Multiple flags can be OR-ed
	HTTP_READ_STATUS	Result states for TCPIP_HTTP_PostNameRead, TCPIP_HTTP_PostValueRead and TCPIP_HTTP_PostReadPair
	HTTP_STATUS	Supported Commands and Server Response Codes

Functions

	Name	Description
	TCPIP_HTTP_ActiveConnectionCountGet	Gets the number of active connections.
	TCPIP_HTTP_ArgGet	Locates a form field value in a given data array.
	TCPIP_HTTP_CurrentConnectionByteCountDec	Decrements the connection byte count.
	TCPIP_HTTP_CurrentConnectionByteCountGet	Returns how many bytes have been read so far
	TCPIP_HTTP_CurrentConnectionByteCountSet	Sets how many bytes have been read so far.
	TCPIP_HTTP_CurrentConnectionCallbackPosGet	Returns the callback position indicator.
	TCPIP_HTTP_CurrentConnectionCallbackPosSet	Sets the callback position indicator.
	TCPIP_HTTP_CurrentConnectionDataBufferGet	Returns pointer to connection general purpose data buffer.
	TCPIP_HTTP_CurrentConnectionFileGet	Get handle to current connection's file.
	TCPIP_HTTP_CurrentConnectionHandleGet	Gets the connection handle of a HTTP connection.
	TCPIP_HTTP_CurrentConnectionHasArgsGet	Checks whether there are get or cookie arguments.
	TCPIP_HTTP_CurrentConnectionHasArgsSet	Sets whether there are get or cookie arguments.
	TCPIP_HTTP_CurrentConnectionIndexGet	Gets the index of the HTTP connection.
	TCPIP_HTTP_CurrentConnectionIsAuthorizedGet	Gets the authorized state for the current connection.
	TCPIP_HTTP_CurrentConnectionIsAuthorizedSet	Sets the authorized state for the current connection.
	TCPIP_HTTP_CurrentConnectionPostSmGet	Get the POST state machine state.
	TCPIP_HTTP_CurrentConnectionPostSmSet	Set the POST state machine state.
	TCPIP_HTTP_CurrentConnectionSocketGet	Get the socket for the current connection.
	TCPIP_HTTP_CurrentConnectionStatusGet	Gets HTTP status.
	TCPIP_HTTP_CurrentConnectionStatusSet	Sets HTTP status.
	TCPIP_HTTP_CurrentConnectionUserDataGet	Gets the user data parameter for the current connection.
	TCPIP_HTTP_CurrentConnectionUserDataSet	Sets the user data parameter for the current connection.
	TCPIP_HTTP_FileAuthenticate	Determines if a given file name requires authentication
	TCPIP_HTTP_FileInclude	Writes a file byte-for-byte to the currently loaded TCP socket.
	TCPIP_HTTP_GetExecute	Processes GET form field variables and cookies.
	TCPIP_HTTP_PostExecute	Processes POST form variables and data.
	TCPIP_HTTP_PostNameRead	Reads a name from a URL encoded string in the TCP buffer.
	TCPIP_HTTP_PostValueRead	Reads a value from a URL encoded string in the TCP buffer.
	TCPIP_HTTP_Print_varname	Inserts dynamic content into a web page
	TCPIP_HTTP_Task	Standard TCP/IP stack module task function.
	TCPIP_HTTP_URLDecode	Parses a string from URL encoding to plain-text.
	TCPIP_HTTP_UserAuthenticate	Performs validation on a specific user name and password.

Macros

	Name	Description
	TCPIP_HTTP_PostReadPair	Reads a name and value pair from a URL encoded string in the TCP buffer.

Structures

	Name	Description
	TCPIP_HTTP_MODULE_CONFIG	HTTP module dynamic configuration data

Types

	Name	Description
	HTTP_CONN_HANDLE	HTTP connection identifier, handle of a HTTP connection

Description

HTTP Headers for Microchip TCP/IP Stack

The HTTP module runs a web server within the TCP/IP stack. This facilitates an easy method to view status information and control applications

using any standard web browser.

File Name

http.h

Company

Microchip Technology Inc.

http_config.h

HTTP configuration file

Macros

	Name	Description
	TCPIP_HTTP_CACHE_LEN	Max lifetime (sec) of static responses as string
	TCPIP_HTTP_CONFIG_FLAGS	Define the HTTP module configuration flags Use 0 for default See HTTP_MODULE_FLAGS definition for possible values
	TCPIP_HTTP_DEFAULT_FILE	Indicate what HTTP file to serve when no specific one is requested
	TCPIP_HTTP_DEFAULT_LEN	For buffer overrun protection. Set to longest length of above two strings.
	TCPIP_HTTP_FILE_UPLOAD_ENABLE	Configure MPFS over HTTP updating Comment this line to disable updating via HTTP
	TCPIP_HTTP_FILE_UPLOAD_NAME	This is macro TCPIP_HTTP_FILE_UPLOAD_NAME.
	TCPIP_HTTP_MAX_CONNECTIONS	Maximum numbers of simultaneous supported HTTP connections.
	TCPIP_HTTP_MAX_DATA_LEN	Define the maximum data length for reading cookie and GET/POST arguments (bytes)
	TCPIP_HTTP_MAX_HEADER_LEN	Set to length of longest string above
	TCPIP_HTTP_MIN_CALLBACK_FREE	Define the minimum number of bytes free in the TX FIFO before executing callbacks
	TCPIP_HTTP_NO_AUTH_WITHOUT_SSL	Uncomment to require secure connection before requesting a password
	TCPIP_HTTP_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_HTTP_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_HTTP_TASK_RATE	The HTTP task rate, ms The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_HTTP_TIMEOUT	Max time (sec) to await more data before timing out and disconnecting the socket
	TCPIP_HTTP_USE_AUTHENTICATION	Enable basic authentication support
	TCPIP_HTTP_USE_COOKIES	Enable cookie support
	TCPIP_HTTP_USE_POST	Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support
	TCPIP_HTTPS_DEFAULT_FILE	Indicate what HTTPS file to serve when no specific one is requested
	TCPIP_STACK_USE_BASE64_DECODE	Authentication requires Base64 decoding Enable basic authentication support

Description

HyperText Transfer Protocol (HTTP) Configuration file

This file contains the HTTP module configuration options

File Name

http_config.h

Company

Microchip Technology Inc.

HTTP Net Module

This section describes the TCP/IP Stack Library HTTP Net Module.

Introduction

TCP/IP Stack Library Hypertext Transfer Protocol (HTTP) Net Module for Microchip Microcontrollers

This library provides the API of the HTTP Net Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The HTTP Net Web Server module allows a development board to act as a Web server. This facilitates an easy method to view status information and control applications using any standard Web browser. It uses the Networking Presentation Layer to integrate with an external encryption services provider (usually wolfSSL) allowing for secure connections.

Using the Library

This topic describes the basic architecture of the HTTP Net TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `http_net.h`

The interface to the HTTP Net TCP/IP Stack library is defined in the `http_net.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the HTTP Net TCP/IP Stack library should include `tcpip.h`.

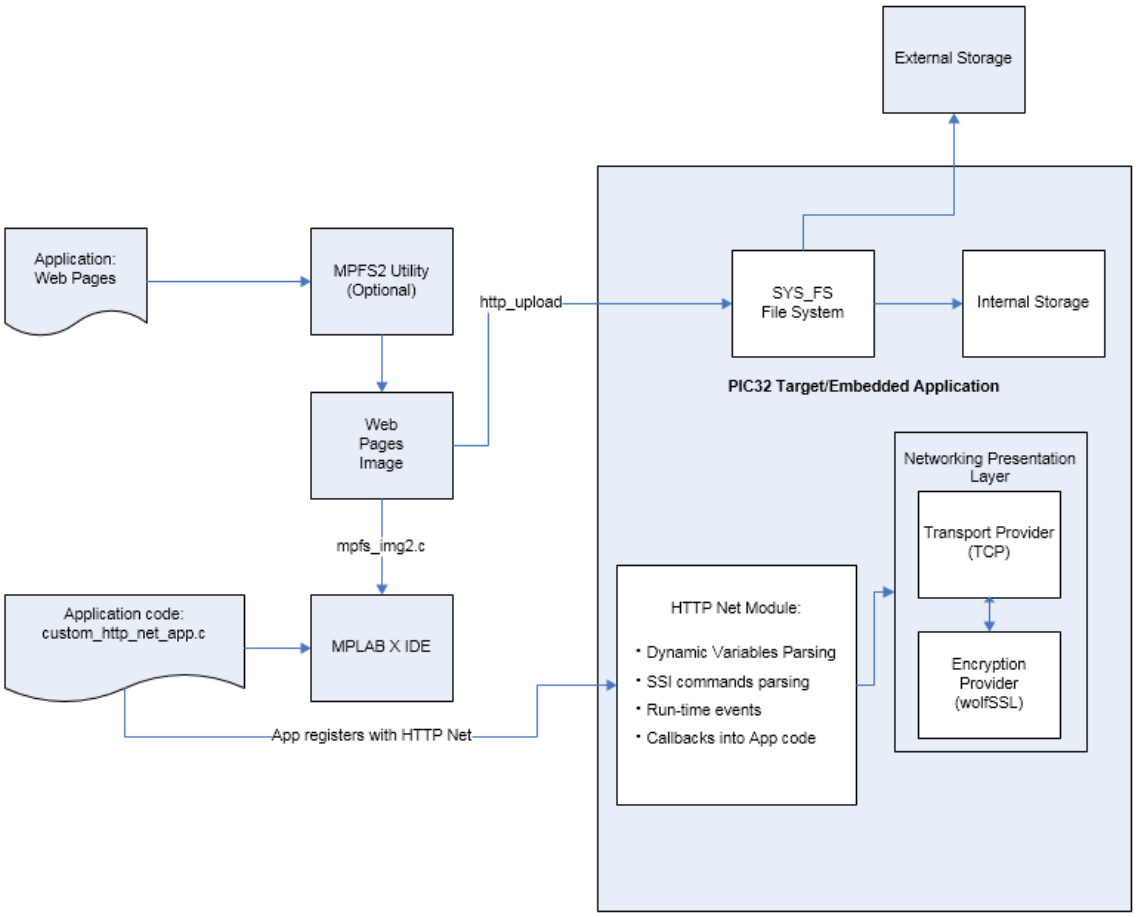
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the HTTP Net TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

HTTP Net Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the HTTP Net module.

Library Interface Section	Description
Functions	This section provides functions for configuring the HTTP Net module.
Data Types and Constants	This section provides various definitions describing this API.

HTTP Net Features

Lists the features of the HTTP Net module.

Description

The HTTP Net module is a HTTP server module that works using the Networking Presentation Layer of the MPLAB Harmony framework. This allows it to support both encrypted and plain text connections.

Some of the key features of the HTTP Net module include:

- The updated module implements HTTP 1.1 persistent connections by default, using the “Transfer-Encoding: chunked” HTTP header. It can be configured to operate with closing the connections like the previous HTTP server.
- The application has to dynamically register its HTTP processing functions with the HTTP Net server using the TCPIP_HTTP_NET_UserHandlerRegister call. The HTTP Net server no longer relies on well known function names.
- The HTTP Net server supports either secure or plain text connections, but not both simultaneously. The specification of a secure or non-secure connection can be easily done by:
- Selecting the server listening port: 80 or 443
- Using the module configuration flags
- The HTTP Net module reports run-time events that could be caught by the user application using the “eventReport” registered function that carries a TCPIP_HTTP_NET_EVENT_TYPE enumerated type

Dynamic Variables Processing

Describes dynamic variable processing.

Description

Parsing of the dynamic variables is done at run-time. The web pages could be changed at run-time, without the need for rebuilding the HTTP Net server. This allows the TCP/IP Stack to run as a library and only the application code to be changed accordingly. The application is required to register a “dynamicPrint” with the HTTP Net server. This function will be called at run-time to process the dynamic variables.

The supported syntax is `~var_name(param1, param2, …)~`. When parsing a string like this within a web page, the HTTP Net server will invoke the “dynamicPrint” function using a TCPIP_HTTP_DYN_VAR_DCPT data structure that specifies the dynamic variable name, its number and type of parameters as well as each parameter value.

For example, the variable `~myVariable(2,6)~` will generate the “dynamicPrint” call with the following parameters:

- `varDcpt.dynName = "myVariable";`
- `varDcpt.nArgs = 2;`
- `varDcpt.dynArgs->argType = TCPIP_HTTP_DYN_ARG_TYPE_INT32;`
- `varDcpt.dynArgs->argInt32 = 2;`
- `(varDcpt.dynArgs + 1)->argType = TCPIP_HTTP_DYN_ARG_TYPE_INT32;`
- `(varDcpt.dynArgs + 1)->argInt32 = 6;`

String and int32_t variable types are currently supported.

The application needs to return a TCPIP_HTTP_DYN_PRINT_RES result specifying if it’s done with processing or it needs to be called again.

Applications no longer have direct access to the underlying transport socket for a HTTP connection. All of the data write operations need to go through the required API function: TCPIP_HTTP_NET_DynamicWrite. This function actually just attaches the data to be written to the connection, using data descriptors. The actual output occurs only when all data is gathered and the data size is known.

Buffers used in a TCPIP_HTTP_NET_DynamicWrite need to be persistent until the moment when the output is generated. RAM buffers are supported by using a “dynamicAck” (TCPIP_HTTP_NET_DynAcknowledge) type function. Once the output operation is completed the HTTP Net module will call back into the application indicating that the corresponding buffer is no longer used and can be freed/reused.

In situations where the dynamic variable print function needs to perform additional write operations, or simply needs to be called again, it must a special result code: TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN/TCPIP_HTTP_DYN_PRINT_RES_AGAIN. Typically this is used when outputting large amounts of data that cannot fit into one single buffer write operation or when the data is not available all at once.

Include Files Processing

Provides information on include file processing.

Description

The HTTP Net module dynamically parses the web pages and the file inclusion is done by the module itself, completely transparent for the application.

File Processing

Provides information on file processing.

Description

File processing consists of the following:

- An included file can include other files that have dynamic variables
- Application can include a file as part of its dynamic variable processing using the TCPIP_HTTP_NET_ConnectionFileInclude()API. That file can include other files.
- The only limit to the depth of inclusion is the memory resources and the TCPIP_HTTP_NET_MODULE_CONFIG:: maxRecurseLevel setting. A TCPIP_HTTP_NET_EVENT_DEPTH_ERROR event will be generated if the maximum depth is exceeded at run-time.

SSI Processing

Provides information on SSI processing.

Description

At run-time, the HTTP Net server processes a subset of the Server Side Includes (SSI) commands. SSI is a portable way to add small amounts of dynamic content on web pages in a standard way that is supported by most HTTP servers. Currently the `#include`, `#set` and `#echo` commands are supported. Other commands will be eventually added. This allows a better compatibility with other existing HTTP servers and is targeted at the removal of proprietary extensions.

Whenever executing an SSI command within a web page, the HTTP Net server will notify the application by using the registered SSI callback: `ssiNotify`. This callback contains a TCPIP_HTTP_SSI_NOTIFY_DCPT pointer to a structure describing the SSI context needed for processing:

- File name the SSI command belongs to
- The SSI command line
- Number of SSI attributes and their values, etc.

The application can do any modifications it chooses to and returns a value instructing the HTTP Net server if the processing of the command is necessary or not.

SSI include Command

The SSI Include command allows the dynamic inclusion of a file. The supported syntax is:

`<!--#include virtual="file_name" -->` or `<!--#include file="file_name" -->`.

Currently the arguments for both “virtual” and “file” commands are passed unaltered to the SYS_FS, so they behave identically. However it is recommended that the original SSI significance for these keywords should be maintained:

- Use “virtual” for specifying a URL relative to the document being server
- Use “file” for a file path, relative to the current directory (it cannot be an absolute path)

The `~inc:file_name~` keyword is maintained for backward compatibility. However the SSI include command should be preferred.

SSI set Command

The SSI `set` command allows to dynamically set a SSI variable value. The supported syntax is:

`<!--#set var="v_name" value="v_value" -->`.

String or integer variables are supported. Variable reference is also supported: `<!--#set var="n_name" value="$otherVar" -->`. This command will create or update the variable `n_name` to have the value of the variable `otherVar`, if it exists.

A new SSI variable will be created if a variable having the name `v_name` does not exist. If the variable `v_name` already exists, it will have its value updated as the result of this command.

An existing variable can be deleted using the empty value set command:

`<!--#set var="v_name" value="" -->`.

SSI echo Command

The SSI `echo` command allows the dynamic print of a SSI variable. The supported syntax is:

`<!--#echo var="v_name" -->`.

If the application `ssiNotify` exists, the HTTP Net server will call it and the application may choose to change the current value dynamically. If `ssiNotify` returns false, the HTTP Net server will display the current value of the variable `v_name` as part of the current page.

The SSI API can be used to evaluate or change the current value of the SSI variables:

- TCPIP_HTTP_NET_SSIVariableGet
- TCPIP_HTTP_NET_SSIVariableSet
- TCPIP_HTTP_NET_SSIVariableDelete

The maximum number of SSI variables is under the control of the application by using the configuration parameter: TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER.

mpfs2.jar Utility

Provides information on the option mpfs2 utility.

Description

Dynamic Variable Processing

The HTTP Net server does not use the `DynRecord.bin` and `FileRcrd.bin` files for processing the dynamic variables. The parsing and processing of dynamic variables is done at run-time. Therefore, the mpfs2.jar utility is no longer necessary, other than to generate a `mpfs_img2.c` image file for the NVM MPFS image.

Application Custom Support Files

The application custom support files generated with MHC have new names: `custom_http_net_app.c` and `http_net_print.c`. However, the `http_net_print.c` file is not generated by the mpfs2 utility and is maintained only for easy comparison between HTTP and HTTP Net application processing. The `custom_http_net_app.c` file is entirely generated using a MHC template file and it is not dynamically updated in any way by the mpfs2 utility.

Generated File Name

Currently the name of the generated file for MPFS image is maintained unchanged: `mpfs_img2.c`.

MPFS Image Generation for Internal NVM Storage

The mpfs2 utility can still be used to generate the MPFS image for internal NVM storage. It can also be useful because it parses the web pages and comes out with the `http_print.c` file that contains the list of TCPIP_HTTP_Print functions. This can be helpful in gathering info about the dynamic variables that are contained within the web pages.

Note: For demonstration applications that use SSI, the file inclusion is now done in a standard way using HTML (i.e., `.htm`) files. Therefore, when generating the image, `*.htm` must be added to *Advanced Settings* > *Do Not Compress*.

Configuring the Library

Macros

	Name	Description
	TCPIP_HTTP_NET_CACHE_LEN	Max lifetime (sec) of static responses as string
	TCPIP_HTTP_NET_CONFIG_FLAGS	Define the HTTP module configuration flags Use 0 for default See HTTP_MODULE_FLAGS definition for possible values
	TCPIP_HTTP_NET_COOKIE_BUFFER_SIZE	Size of the buffer used for sending the cookies to the client. Should be able to accommodate the longest cookie response. Otherwise the cookies will be truncated.
	TCPIP_HTTP_NET_DEFAULT_FILE	Indicate what HTTP file to serve when no specific one is requested
	TCPIP_HTTP_NET_DYNVAR_ARG_MAX_NUMBER	maximum number of arguments for a dynamic variable
	TCPIP_HTTP_NET_DYNVAR_DESCRIPTORS_NUMBER	how many buffers descriptors for dynamic variable processing they are independent of the HTTP connection number all the HTTP connections use from the dynamic descriptors pool
	TCPIP_HTTP_NET_DYNVAR_MAX_LEN	maximum size for a complete dynamic variable: name + args must be <= TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE! If it is much larger than needed then inefficiency occurs when reading data from the file and then discarding it because a much larger than needed data buffer was read
	TCPIP_HTTP_NET_FILE_UPLOAD_ENABLE	Configure MPFS over HTTP updating Comment this line to disable updating via HTTP
	TCPIP_HTTP_NET_FILE_UPLOAD_NAME	This is macro TCPIP_HTTP_NET_FILE_UPLOAD_NAME.

	TCPIP_HTTP_NET_FIND_PEEK_BUFF_SIZE	Size of the peek buffer to perform searches into. If the underlying transport layer supports offset peak operation with a offset, value could be smaller (80 characters, for example); otherwise, a one time peek is required and the buffer should be larger - 512 bytes recommended.
	TCPIP_HTTP_NET_MAX_CONNECTIONS	Maximum numbers of simultaneous supported HTTP connections.
	TCPIP_HTTP_NET_MAX_DATA_LEN	Define the maximum data length for reading cookie and GET/POST arguments (bytes)
	TCPIP_HTTP_NET_MAX_HEADER_LEN	The length of longest header string that can be parsed
	TCPIP_HTTP_NET_MAX_RECURSE_LEVEL	The maximum depth of recursive calls for serving a web page: • no dynvars files: 1 • file including a file: 2 • if the include file includes another file: +1 • if a dyn variable: +1 Default value is 3;
	TCPIP_HTTP_NET_RESPONSE_BUFFER_SIZE	Size of the buffer used for sending the response messages to the client. Should be able to accommodate the longest server response: Default setting should be 300 bytes
	TCPIP_HTTP_NET_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_HTTP_NET_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_HTTP_NET_TASK_RATE	The HTTP task rate, ms The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_HTTP_NET_TIMEOUT	Max time (sec) to await more data before timing out and disconnecting the socket
	TCPIP_HTTP_NET_USE_AUTHENTICATION	Enable basic authentication support
	TCPIP_HTTP_NET_USE_COOKIES	Enable cookie support
	TCPIP_HTTP_NET_USE_POST	Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support
	TCPIP_HTTP_NET_CHUNK_RETRIES	retry limit for allocating a chunk from the pool If more retries are not successful the operation will be aborted
	TCPIP_HTTP_NET_CHUNKS_NUMBER	number of chunks that are created It depends on the TCPIP_HTTP_NET_MAX_RECURSE_LEVEL and on the number of connections Maximum number should be TCPIP_HTTP_NET_MAX_CONNECTIONS * TCPIP_HTTP_NET_MAX_RECURSE_LEVEL i.e. TCPIP_HTTP_NET_MODULE_CONFIG::nConnections * TCPIP_HTTP_NET_MODULE_CONFIG::nChunks All the chunks are in a pool and are used by all connections
	TCPIP_HTTP_NET_DYNVAR_PROCESS	This symbol enables the processing of dynamic variables Make it evaluate to false (0) if dynamic variables are not needed All the following symbols referring to dynamic variables are relevant only when TCPIP_HTTP_NET_DYNVAR_PROCESS != 0
	TCPIP_HTTP_NET_DYNVAR_PROCESS_RETRIES	retry limit for a dynamic variable processing ths puts a limit on the number of times a dynamic variable "dynamicPrint" function can return TCPIP_HTTP_DYN_PRINT_RES_AGAIN/TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN and avoids having the HTTP code locked up forever. If more retries are attempted the processing will be considered done and dynamicPrint function will not be called again
	TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_RETRIES	Retry limit for allocating a file buffer from the pool. If more retries are not successful the operation will be aborted.

	TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE	Size of the buffer used for processing HTML, dynamic variable and binary files. For dynamic variable files it should be able to accommodate the longest HTML line size, including CRLF!
	TCPIP_HTTP_NET_FILE_PROCESS_BUFFERS_NUMBER	Number of file buffers to be created; These buffers are used to store data while file processing is done They are organized in a pool Each file being processed needs a file buffer and tries to get it from the pool If a buffer is not available, the HTTP conenction will wait for one to become available. Once the file is done the file buffer is released and could be used by a different file The number depends on the number of files that are processed in parallel To avoid deadlock the number should be >= than the number of... more
	TCPIP_HTTP_NET_FILENAME_MAX_LEN	maximum size of a HTTP file name with the path removed from the file name one extra char added for the string terminator
	TCPIP_HTTP_NET_SSI_ATTRIBUTES_MAX_NUMBER	maximum number of attributes for a SSI command most SSI commands take just one attribute/value pair per line but multiple attribute/value pairs on the same line are allowed where it makes sense
	TCPIP_HTTP_NET_SSI_CMD_MAX_LEN	maximum size for a SSI command line: command + attribute/value pairs must be <= TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE! If it is much larger than needed then inefficiency occurs when reading data from the file and then discarding it because a much larger than needed data buffer was read
	TCPIP_HTTP_NET_SSI_ECHO_NOT_FOUND_MESSAGE	message to echo when echoing a not found variable
	TCPIP_HTTP_NET_SSI_PROCESS	This symbol enables the processing of SSI commands Make it evaluate to false (0) if SSI commands are not needed All the following symbols referring to SSI commands are relevant only when TCPIP_HTTP_NET_SSI_PROCESS != 0
	TCPIP_HTTP_NET_SSI_STATIC_ATTTRIB_NUMBER	number of static attributes associated to a SSI command if the command has more attributes than this number the excess will be allocated dynamically
	TCPIP_HTTP_NET_SSI_VARIABLE_NAME_MAX_LENGTH	maximum length of a SSI variable name any excess characters will be truncated Note that this can result in multiple variables being represented as one SSI variable
	TCPIP_HTTP_NET_SSI_VARIABLE_STRING_MAX_LENGTH	maximum size of a SSI string variable value any excess characters will be truncated Note that the variable value requires SSI storage that's allocated dynamically Also, this value determines the size of an automatic (stack) buffer when the variable is echoed. If this value is large, make sure you have enough stack space.
	TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER	maximum number of SSI variables that can be created at run time These variables are stored in an internal hash. For max. efficiency this number should be a prime.

Description

The configuration of the HTTP Net TCP/IP Stack is based on the file `tcpip_config.h` (which may include `http_net_config.h`).

This header file contains the configuration selection for the HTTP Net TCP/IP Stack. Based on the selections made, the HTTP Net TCP/IP Stack may support the selected features.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_HTTP_NET_CACHE_LEN Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_CACHE_LEN ("600")
```

Description

Max lifetime (sec) of static responses as string

TCPIP_HTTP_NET_CONFIG_FLAGS Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_CONFIG_FLAGS 0
```

Description

Define the HTTP module configuration flags Use 0 for default See HTTP_MODULE_FLAGS definition for possible values

TCPIP_HTTP_NET_COOKIE_BUFFER_SIZE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_COOKIE_BUFFER_SIZE 200
```

Description

Size of the buffer used for sending the cookies to the client. Should be able to accommodate the longest cookie response. Otherwise the cookies will be truncated.

TCPIP_HTTP_NET_DEFAULT_FILE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_DEFAULT_FILE "index.htm"
```

Description

Indicate what HTTP file to serve when no specific one is requested

TCPIP_HTTP_NET_DYNVAR_ARG_MAX_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_DYNVAR_ARG_MAX_NUMBER 10
```

Description

maximum number of arguments for a dynamic variable

TCPIP_HTTP_NET_DYNVAR_DESCRIPTORS_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_DYNVAR_DESCRIPTORS_NUMBER 10
```

Description

how many buffers descriptors for dynamic variable processing they are independent of the HTTP connection number all the HTTP connections use from the dynamic descriptors pool

TCPIP_HTTP_NET_DYNVAR_MAX_LEN Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_DYNVAR_MAX_LEN 50
```

Description

maximum size for a complete dynamic variable: name + args must be <= TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE! If it is much larger than needed then inefficiency occurs when reading data from the file and then discarding it because a much larger than needed data buffer was read

TCPIP_HTTP_NET_FILE_UPLOAD_ENABLE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FILE_UPLOAD_ENABLE
```

Description

Configure MPFS over HTTP updating Comment this line to disable updating via HTTP

TCPIP_HTTP_NET_FILE_UPLOAD_NAME Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FILE_UPLOAD_NAME "mpfsupload"
```

Description

This is macro TCPIP_HTTP_NET_FILE_UPLOAD_NAME.

TCPIP_HTTP_NET_FIND_PEEK_BUFF_SIZE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FIND_PEEK_BUFF_SIZE 512
```

Description

Size of the peek buffer to perform searches into. If the underlying transport layer supports offset peak operation with a offset, value could be smaller (80 characters, for example); otherwise, a one time peek is required and the buffer should be larger - 512 bytes recommended.

Remarks

if upload is enabled (TCPIP_HTTP_NET_FILE_UPLOAD_ENABLE) then it is recommended for the peek buffer size to match the size of the socket RX buffer: TCPIP_HTTP_NET_SKT_RX_BUFF_SIZE Note that this is an automatic buffer (created on the stack) and enough stack space should be provided for the application.

TCPIP_HTTP_NET_MAX_CONNECTIONS Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_MAX_CONNECTIONS (4)
```

Description

Maximum numbers of simultaneous supported HTTP connections.

TCPIP_HTTP_NET_MAX_DATA_LEN Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_MAX_DATA_LEN (100u)
```

Description

Define the maximum data length for reading cookie and GET/POST arguments (bytes)

TCPIP_HTTP_NET_MAX_HEADER_LEN Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_MAX_HEADER_LEN (15u)
```

Description

The length of longest header string that can be parsed

TCPIP_HTTP_NET_MAX_RECURSE_LEVEL Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_MAX_RECURSE_LEVEL 3
```

Description

The maximum depth of recursive calls for serving a web page:

- no dynvars files: 1
- file including a file: 2
- if the include file includes another file: +1
- if a dyn variable: +1

Default value is 3;

TCPIP_HTTP_NET_RESPONSE_BUFFER_SIZE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_RESPONSE_BUFFER_SIZE 300
```

Description

Size of the buffer used for sending the response messages to the client. Should be able to accommodate the longest server response: Default setting should be 300 bytes

TCPIP_HTTP_NET_SKT_RX_BUFF_SIZE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SKT_RX_BUFF_SIZE 2048
```

Description

Define the size of the RX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

TCPIP_HTTP_NET_SKT_TX_BUFF_SIZE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SKT_TX_BUFF_SIZE 2048
```

Description

Define the size of the TX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.

TCPIP_HTTP_NET_TASK_RATE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_TASK_RATE 33
```

Description

The HTTP task rate, ms The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_HTTP_NET_TIMEOUT Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_TIMEOUT (45u)
```

Description

Max time (sec) to await more data before timing out and disconnecting the socket

TCPIP_HTTP_NET_USE_AUTHENTICATION Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_USE_AUTHENTICATION
```

Description

Enable basic authentication support

TCPIP_HTTP_NET_USE_COOKIES Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_USE_COOKIES
```

Description

Enable cookie support

TCPIP_HTTP_NET_USE_POST Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_USE_POST
```

Description

Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support

TCPIP_HTTP_NET_CHUNK_RETRIES Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_CHUNK_RETRIES 10
```

Description

retry limit for allocating a chunk from the pool If more retries are not successful the operation will be aborted

TCPIP_HTTP_NET_CHUNKS_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_CHUNKS_NUMBER 10
```

Description

number of chunks that are created It depends on the TCPIP_HTTP_NET_MAX_RECURSE_LEVEL and on the number of connections Maximum number should be TCPIP_HTTP_NET_MAX_CONNECTIONS * TCPIP_HTTP_NET_MAX_RECURSE_LEVEL i.e. TCPIP_HTTP_NET_MODULE_CONFIG::nConnections * TCPIP_HTTP_NET_MODULE_CONFIG::nChunks All the chunks are in a pool and are used by all connections

TCPIP_HTTP_NET_DYNVAR_PROCESS Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_DYNVAR_PROCESS 1
```

Description

This symbol enables the processing of dynamic variables Make it evaluate to false (0) if dynamic variables are not needed All the following symbols referring to dynamic variables are relevant only when TCPIP_HTTP_NET_DYNVAR_PROCESS != 0

TCPIP_HTTP_NET_DYNVAR_PROCESS_RETRIES Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_DYNVAR_PROCESS_RETRIES 10
```

Description

retry limit for a dynamic variable processing ths puts a limit on the number of times a dynamic variable "dynamicPrint" function can return TCPIP_HTTP_DYN_PRINT_RES_AGAIN/TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN and avoids having the HTTP code locked up forever. If more retries are attempted the processing will be considered done and dynamicPrint function will not be called again

TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_RETRIES Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_RETRIES 10
```

Description

Retry limit for allocating a file buffer from the pool. If more retries are not successful the operation will be aborted.

TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE 512
```

Description

Size of the buffer used for processing HTML, dynamic variable and binary files. For dynamic variable files it should be able to accommodate the longest HTML line size, including CRLF!

TCPIP_HTTP_NET_FILE_PROCESS_BUFFERS_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FILE_PROCESS_BUFFERS_NUMBER 4
```

Description

Number of file buffers to be created; These buffers are used to store data while file processing is done They are organized in a pool Each file being processed needs a file buffer and tries to get it from the pool If a buffer is not available, the HTTP conenction will wait for one to become available. Once the file is done the file buffer is released and could be used by a different file The number depends on the number of files that are processed in parallel To avoid deadlock the number should be >= than the number of maximum files that can be open simultaneously: i.e. for file1 ->include file2 -> include file3 you'll need >= 3 file process buffers

TCPIP_HTTP_NET_FILENAME_MAX_LEN Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_FILENAME_MAX_LEN 25
```

Description

maximum size of a HTTP file name with the path removed from the file name one extra char added for the string terminator

TCPIP_HTTP_NET_SSI_ATTRIBUTES_MAX_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_ATTRIBUTES_MAX_NUMBER 4
```

Description

maximum number of attributes for a SSI command most SSI commands take just one attribute/value pair per line but multiple attribute/value pairs on the same line are allowed where it makes sense

TCPIP_HTTP_NET_SSI_CMD_MAX_LEN Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_CMD_MAX_LEN 100
```

Description

maximum size for a SSI command line: command + attribute/value pairs must be <= TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE! If it is much larger than needed then inefficiency occurs when reading data from the file and then discarding it because a much larger than needed data buffer was read

TCPIP_HTTP_NET_SSI_ECHO_NOT_FOUND_MESSAGE Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_ECHO_NOT_FOUND_MESSAGE "SSI Echo - Not Found: "
```

Description

message to echo when echoing a not found variable

TCPIP_HTTP_NET_SSI_PROCESS Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_PROCESS 1
```

Description

This symbol enables the processing of SSI commands Make it evaluate to false (0) if SSI commands are not needed All the following symbols referring to SSI commands are relevant only when TCPIP_HTTP_NET_SSI_PROCESS != 0

TCPIP_HTTP_NET_SSI_STATIC_ATTTRIB_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_STATIC_ATTTRIB_NUMBER 2
```

Description

number of static attributes associated to a SSI command if the command has more attributes than this number the excess will be allocated dynamically

TCPIP_HTTP_NET_SSI_VARIABLE_NAME_MAX_LENGTH Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_VARIABLE_NAME_MAX_LENGTH 10
```

Description

maximum length of a SSI variable name any excess characters will be truncated Note that this can result in multiple variables being represented as one SSI variable

TCPIP_HTTP_NET_SSI_VARIABLE_STRING_MAX_LENGTH Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_VARIABLE_STRING_MAX_LENGTH 10
```

Description

maximum size of a SSI string variable value any excess characters will be truncated Note that the variable value requires SSI storage that's allocated dynamically Also, this value determines the size of an automatic (stack) buffer when the variable is echoed. If this value is large, make sure you have enough stack space.

TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER Macro

File

http_net_config.h

C

```
#define TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER 13
```

Description

maximum number of SSI variables that can be created at run time These variables are stored in an internal hash. For max. efficiency this number should be a prime.

Building the Library

This section lists the files that are available in the HTTP Net TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
/tcpip.h	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/http_net.c	HTTP Server implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The HTTP Net module depends on the following modules:

- TCP/IP Stack Library
- TCP Module
- File System Service Library

Library Interface

a) Functions

	Name	Description
	TCPIP_HTTP_NET_ActiveConnectionCountGet	Gets the number of active (and inactive) connections.
	TCPIP_HTTP_NET_ConnectionByteCountDec	Decrements the connection byte count.
	TCPIP_HTTP_NET_ArgGet	Locates a form field value in a given data array.
	TCPIP_HTTP_NET_ConnectionByteCountSet	Sets how many bytes have been read so far.
	TCPIP_HTTP_NET_ConnectionByteCountGet	Returns how many bytes have been read so far.
	TCPIP_HTTP_NET_ConnectionCallbackPosSet	Sets the callback position indicator.
	TCPIP_HTTP_NET_ConnectionCallbackPosGet	Returns the callback position indicator.
	TCPIP_HTTP_NET_ConnectionDataBufferGet	Returns pointer to connection general purpose data buffer.
	TCPIP_HTTP_NET_ConnectionDiscard	Discards any pending data in the connection RX buffer.
	TCPIP_HTTP_NET_ConnectionFileAuthenticate	Determines if a given file name requires authentication
	TCPIP_HTTP_NET_ConnectionFileGet	Get handle to current connection's file.
	TCPIP_HTTP_NET_ConnectionFileInclude	Writes a file to the HTTP connection.
	TCPIP_HTTP_NET_ConnectionHasArgsGet	Checks whether there are get or cookie arguments.
	TCPIP_HTTP_NET_ConnectionFlush	Immediately transmits all connection pending TX data.
	TCPIP_HTTP_NET_ConnectionIsAuthorizedGet	Gets the authorized state for the current connection.
	TCPIP_HTTP_NET_ConnectionGetExecute	Processes GET form field variables and cookies.
	TCPIP_HTTP_NET_ConnectionPostNameRead	Reads a name from a URL encoded string in the network transport buffer.
	TCPIP_HTTP_NET_ConnectionHasArgsSet	Sets whether there are get or cookie arguments.
	TCPIP_HTTP_NET_ConnectionPostSmGet	Get the POST state machine state.
	TCPIP_HTTP_NET_ConnectionIsAuthorizedSet	Sets the authorized state for the current connection.
	TCPIP_HTTP_NET_ConnectionPostValueRead	Reads a value from a URL encoded string in the network transport buffer.
	TCPIP_HTTP_NET_ConnectionNetHandle	Returns the network handle of the current connection.
	TCPIP_HTTP_NET_ConnectionSocketGet	Get the socket for the current connection.
	TCPIP_HTTP_NET_ConnectionPeek	Reads a specified number of data bytes from the connection RX buffer without removing them from the buffer.
	TCPIP_HTTP_NET_ConnectionStatusGet	Gets HTTP status.
	TCPIP_HTTP_NET_ConnectionPostExecute	Processes POST form variables and data.

	TCPIP_HTTP_NET_ConnectionUserDataGet	Gets the user data parameter for the current connection.
	TCPIP_HTTP_NET_ConnectionPostSmSet	Set the POST state machine state.
	TCPIP_HTTP_NET_ConnectionRead	Reads an array of data bytes from a connection's RX buffer.
	TCPIP_HTTP_NET_ConnectionReadBufferSize	Returns the size of the connection RX buffer.
	TCPIP_HTTP_NET_ConnectionReadIsReady	Determines how many bytes can be read from the connection RX buffer.
	TCPIP_HTTP_NET_ConnectionStatusSet	Sets HTTP status.
	TCPIP_HTTP_NET_ConnectionStringFind	Helper to find a string of characters in an incoming connection buffer.
	TCPIP_HTTP_NET_ConnectionUserAuthenticate	Performs validation on a specific user name and password.
	TCPIP_HTTP_NET_ConnectionUserDataSet	Sets the user data parameter for the current connection.
	TCPIP_HTTP_NET_DynAcknowledge	Dynamic variable acknowledge function
	TCPIP_HTTP_NET_DynamicWrite	Writes a data buffer to the current connection
	TCPIP_HTTP_NET_DynamicWriteString	Helper for writing a string within a dynamic variable context.
	TCPIP_HTTP_NET_DynPrint	Inserts dynamic content into a web page
	TCPIP_HTTP_NET_EventReport	Reports an event that occurred in the processing of a HTTP web page
	TCPIP_HTTP_NET_Task	Standard TCP/IP stack module task function.
	TCPIP_HTTP_NET_URLDecode	Parses a string from URL encoding to plain text.
	TCPIP_HTTP_NET_UserHandlerDeregister	Deregisters a previously registered HTTP user handler.
	TCPIP_HTTP_NET_UserHandlerRegister	Registers a user callback structure with the HTTP server
	TCPIP_HTTP_NET_ConnectionDataBufferSizeGet	Returns the size of the connection general purpose data buffer.
	TCPIP_HTTP_NET_ConnectionDynamicDescriptors	Returns the number of dynamic variable descriptors
	TCPIP_HTTP_NET_SSINotification	Reports an SSI processing event that occurs in the processing of a HTTP web page
	TCPIP_HTTP_NET_SSIVariableDelete	Function to delete an SSI variable.
	TCPIP_HTTP_NET_SSIVariableGet	Function to get access to an existing SSI variable.
	TCPIP_HTTP_NET_SSIVariableGetByIndex	Function to get access to an existing SSI variable.
	TCPIP_HTTP_NET_SSIVariableSet	Function to set an SSI variable.
	TCPIP_HTTP_NET_SSIVariablesNumberGet	Function to get the number of the current SSI variables.
	TCPIP_HTTP_NET_ConnectionHandleGet	Gets the connection handle of a HTTP connection.
	TCPIP_HTTP_NET_ConnectionIndexGet	Gets the index of the HTTP connection.

b) Data Types and Constants

	Name	Description
	_tag_TCPIP_HTTP_NET_USER_CALLBACK	HTTP user implemented callback data structure.
	TCPIP_HTTP_DYN_ARG_DCPT	HTTP dynamic argument descriptor.
	TCPIP_HTTP_DYN_ARG_TYPE	HTTP supported dynamic variables argument types.
	TCPIP_HTTP_DYN_PRINT_RES	Dynamic print result when a dynamic variable print callback function returns;
	TCPIP_HTTP_DYN_VAR_DCPT	HTTP dynamic variable descriptor.
	TCPIP_HTTP_DYN_VAR_FLAGS	HTTP supported dynamic variables flags.
	TCPIP_HTTP_NET_CONN_HANDLE	HTTP connection identifier, handle of a HTTP connection
	TCPIP_HTTP_NET_EVENT_TYPE	HTTP reported run-time events.
	TCPIP_HTTP_NET_IO_RESULT	Result states for execution callbacks
	TCPIP_HTTP_NET_MODULE_CONFIG	HTTP module dynamic configuration data
	TCPIP_HTTP_NET_MODULE_FLAGS	HTTP module configuration flags Multiple flags can be OR-ed
	TCPIP_HTTP_NET_READ_STATUS	Result states for TCPIP_HTTP_NET_ConnectionPostNameRead, TCPIP_HTTP_NET_ConnectionPostValueRead and TCPIP_HTTP_NET_ConnectionPostReadPair
	TCPIP_HTTP_NET_STATUS	Supported Commands and Server Response Codes
	TCPIP_HTTP_NET_USER_CALLBACK	HTTP user implemented callback data structure.
	TCPIP_HTTP_NET_USER_HANDLE	HTTP user handle.
	TCPIP_HTTP_NET_ConnectionPostReadPair	Reads a name and value pair from a URL encoded string in the network transport buffer.
	TCPIP_HTTP_SSI_ATTR_DCPT	HTTP SSI attribute descriptor.
	TCPIP_HTTP_SSI_NOTIFY_DCPT	HTTP SSI notification descriptor.

Description

This section describes the Application Programming Interface (API) functions of the HTTP Net module.

Refer to each section for a detailed description.

a) Functions

TCPIP_HTTP_NET_ActiveConnectionCountGet Function

Gets the number of active (and inactive) connections.

File

http_net.h

C

```
int TCPIP_HTTP_NET_ActiveConnectionCountGet(int* pOpenCount);
```

Returns

The number of active/total connections.

Description

This function will return the number of active and total opened HTTP connections at the current time.

Remarks

The value returned by this function is informational only. The number of active connections changes dynamically.

Preconditions

None.

Example

```
int nConns;

nConns = TCPIP_HTTP_NET_ActiveConnectionCountGet(0);
```

Parameters

Parameters	Description
pOpenCount	address to store the number of total HTTP opened connections Could be NULL if not needed

Function

int TCPIP_HTTP_NET_ActiveConnectionCountGet(int* pOpenCount);

TCPIP_HTTP_NET_ConnectionByteCountDec Function

Decrements the connection byte count.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionByteCountDec(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint32_t byteCount);
```

Returns

None.

Description

This function decrements the current value of the counter showing the number of bytes read from the connection so far.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
byteCount	byte count reduction

Function

void TCPIP_HTTP_NET_ConnectionByteCountDec(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
uint32_t byteCount)

TCPIP_HTTP_NET_ArgGet Function

Locates a form field value in a given data array.

File

http_net.h

C

```
const uint8_t* TCPIP_HTTP_NET_ArgGet(const uint8_t* cData, const uint8_t* cArg);
```

Returns

A pointer to the argument value, or NULL if not found.

Description

This function searches through a data array to find the value associated with a given argument. It can be used to find form field values in data received over GET or POST.

The end of data is assumed to be reached when a null name parameter is encountered. This requires the string to have an even number of null-terminated strings, followed by an additional null terminator.

Remarks

None.

Preconditions

The data array has a valid series of null terminated name/value pairs.

Example

```
TCPIP_HTTP_NET_DynPrint(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const TCPIP_HTTP_DYN_VAR_DCPT*
                        varDcpt, const TCPIP_HTTP_NET_USER_CALLBACK* pCBack)
{
    const uint8_t *ptr;

    ptr = TCPIP_HTTP_NET_ArgGet(TCPIP_HTTP_NET_ConnectionDataBufferGet(connHandle),
                                (const uint8_t*)"name");
    if(ptr == 0)
    {
        ptr = "not set";
    }
    else
    {
        strncpy(myBuffer, ptr, sizeof(myBuffer));
        ptr = myBuffer;
    }

    TCPIP_HTTP_NET_DynamicWrite(varDcpt, ptr, strlen(ptr), false);

}
```

Parameters

Parameters	Description
cData	the buffer to search
cArg	the name of the argument to find

Function

const uint8_t* TCPIP_HTTP_NET_ArgGet(const uint8_t* cData, const uint8_t* cArg)

TCPIP_HTTP_NET_ConnectionByteCountSet Function

Sets how many bytes have been read so far.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionByteCountSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint32_t byteCount);
```

Returns

None.

Description

This function sets the current value of the counter showing the number of bytes read from the connection so far.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
byteCount	byte count to be set

Function

void TCPIP_HTTP_NET_ConnectionByteCountSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
uint32_t byteCount)

TCPIP_HTTP_NET_ConnectionByteCountGet Function

Returns how many bytes have been read so far.

File

http_net.h

C

```
uint32_t TCPIP_HTTP_NET_ConnectionByteCountGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

Current connection byte count, how many bytes have been read so far.

Description

This function returns the current value of the counter showing the number of bytes read from the connection so far.

Remarks

None.

Preconditions

None.

Example

```
switch(TCPIP_HTTP_NET_ConnectionPostSmGet(connHandle))
{
    case SM_CFG_SNMP_READ_NAME:
        // If all parameters have been read, end
```

```
            if(TCPIP_HTTP_NET_ConnectionByteCountGet(connHandle) == 0u)
            {
                return TCPIP_HTTP_NET_IO_RES_DONE;
            }
        .
        .
        .
    }
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint32_t TCPIP_HTTP_NET_ConnectionByteCountGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionCallbackPosSet Function

Sets the callback position indicator.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionCallbackPosSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint32_t callbackPos);
```

Returns

None.

Description

This function will set the current value of the callback position indicator for the HTTP connection identified by connHandle. The callback position indicator is used in the processing of the HTTP dynamic variables. When set to a value != 0, it indicates to the HTTP server that the application has more pending processing that needs to be done.

Remarks

None.

Preconditions

None.

Example

Parameters

Parameters	Description
connHandle	HTTP connection handle
callbackPos	new connection callback position value

Function

void TCPIP_HTTP_NET_ConnectionCallbackPosSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
uint32_t callbackPos)

TCPIP_HTTP_NET_ConnectionCallbackPosGet Function

Returns the callback position indicator.

File

http_net.h

C

```
uint32_t TCPIP_HTTP_NET_ConnectionCallbackPosGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

Callback position indicator for connection defined by connHandle.

Description

This function will return the current value of the callback position indicator for the HTTP connection identified by connHandle. The callback position indicator is used in the processing of the HTTP dynamic variables.

Remarks

None.

Preconditions

None.

Example

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint32_t TCPIP_HTTP_NET_ConnectionCallbackPosGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionDataBufferGet Function

Returns pointer to connection general purpose data buffer.

File

http_net.h

C

```
uint8_t* TCPIP_HTTP_NET_ConnectionDataBufferGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

Pointer to the connection's general purpose data buffer.

Description

This function returns a pointer to the HTTP connection internal data buffer. This gives access to the application to the data that's stored in the HTTP connection buffer.

Remarks

None.

Preconditions

None.

Example

```
TCPIP_HTTP_NET_DynPrint(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
   const TCPIP_HTTP_DYN_VAR_DCPT* varDcpt,
   const TCPIP_HTTP_NET_USER_CALLBACK* pCBack)
{
    const uint8_t *ptr;

    ptr = TCPIP_HTTP_NET_ArgGet(TCPIP_HTTP_NET_ConnectionDataBufferGet
                               (connHandle), (const uint8_t*)"name");
    if(ptr == 0)
    {
        ptr = "not set";
    }
    else
    {
        strncpy(myBuffer, ptr, sizeof(myBuffer));
        ptr = myBuffer;
    }
```

```
        TCPIP_HTTP_NET_DynamicWrite(varDcpt, ptr, strlen(ptr), false);
    }
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint8_t* TCPIP_HTTP_NET_ConnectionDataBufferGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionDiscard Function

Discards any pending data in the connection RX buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionDiscard(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

The number of bytes that have been discarded from the RX buffer.

Description

This function discards data from the connection RX buffer.

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle

Function

uint16_t TCPIP_HTTP_NET_ConnectionDiscard(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionFileAuthenticate Function

Determines if a given file name requires authentication

File

http_net.h

C

```
uint8_t TCPIP_HTTP_NET_ConnectionFileAuthenticate(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* cFile,
const TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

- <= 0x79 - valid authentication is required
- >= 0x80 - access is granted for this connection

Description

This function is implemented by the application developer. Its function is to determine if a file being requested requires authentication to view. The user name and password, if supplied, will arrive later with the request headers, and will be processed at that time.

Return values 0x80 - 0xff indicate that authentication is not required, while values from 0x00 to 0x79 indicate that a user name and password are required before proceeding. While most applications will only use a single value to grant access and another to require authorization, the range allows multiple "realms" or sets of pages to be protected, with different credential requirements for each.

The return value of this function is saved for the current connection and can be read using TCPIP_HTTP_NET_ConnectionIsAuthorizedGet. It will be available to future callbacks, including TCPIP_HTTP_NET_ConnectionUserAuthenticate and any of the TCPIP_HTTP_NET_ConnectionGetExecute, TCPIP_HTTP_NET_ConnectionPostExecute, or TCPIP_HTTP_NET_Print_varname callbacks.

Remarks

This function may NOT write to the network transport buffer.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cFile	the name of the file being requested
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

uint8_t TCPIP_HTTP_NET_ConnectionFileAuthenticate
(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
const char* cFile, const TCPIP_HTTP_NET_USER_CALLBACK* pCBack)

TCPIP_HTTP_NET_ConnectionFileGet Function

Get handle to current connection's file.

File

http_net.h

C

```
SYS_FS_HANDLE TCPIP_HTTP_NET_ConnectionFileGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

Handle to File System file belonging to the connection defined by connHandle

Description

This function returns the handle of the current HTTP connection file.

Remarks

None.

Preconditions

None.

Example

```
uint8_t myBuff[20];

// Get the file handle and read from that file
SYS_FS_FileRead(myBuff, sizeof(myBuff), TCPIP_HTTP_NET_ConnectionFileGet(connHandle));
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

FILE_HANDLE TCPIP_HTTP_NET_ConnectionFileGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionFileInclude Function

Writes a file to the HTTP connection.

File

http_net.h

C

```
bool TCPIP_HTTP_NET_ConnectionFileInclude(HTTP_CONN_HANDLE connHandle, const char* fileName);
```

Returns

true if the call succeeded, false if the file could not be opened

Description

This function allows an entire file to be included as part of a dynamic variable processing. This reduces unneeded duplication of visual elements such as headers, menus, etc.

It is not meant for the processing of dynamic variables that include files by using the "inc" keyword (see the Remarks section below).

Remarks

Should be called from within a dynamic variable processing function context. The function allows to insert a file, as part of the dynamic variable processing.

The included file can contain dynamic variables. However there's a limit for recursive calls (see TCPIP_HTTP_NET_MAX_RECURSE_LEVEL).

The file is just added to the processing queue. Returning true does not mean that the whole file has been already sent to the connection.

If the function returns true but an error occurs during the file processing an event will be reported using the TCPIP_HTTP_NET_EventReport callback.

If the function returns false an event will be reported using the TCPIP_HTTP_NET_EventReport callback with additional info.

Please note that the processing of HTTP dynamic keywords in the HTML code such as `~inc:filename.ext~` is processed internally by the HTTP module! For such a dynamic variable, control is not passed to the user.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
fileName	the name of the file to be included

Function

bool TCPIP_HTTP_NET_ConnectionFileInclude(HTTP_CONN_HANDLE connHandle,

const char* fileName)

TCPIP_HTTP_NET_ConnectionHasArgsGet Function

Checks whether there are get or cookie arguments.

File

http_net.h

C

```
uint8_t TCPIP_HTTP_NET_ConnectionHasArgsGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

The current value of the connection hasArgs.

Description

The function will get the value of the "cookies or get arguments" that are present.

Remarks

None.

Preconditions

None.

Example

```
uint8_t hasArgs = TCPIP_HTTP_NET_ConnectionHasArgsGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint8_t TCPIP_HTTP_NET_ConnectionHasArgsGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionFlush Function

Immediately transmits all connection pending TX data.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionFlush(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

- number of flushed bytes
- 0 if no flushed bytes or some error

Description

This function calls the transport layer's flush function

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle

Function

uint16_t TCPIP_HTTP_NET_ConnectionFlush(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionIsAuthorizedGet Function

Gets the authorized state for the current connection.

File

http_net.h

C

```
uint8_t TCPIP_HTTP_NET_ConnectionIsAuthorizedGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

A uint8_t representing the authorization status.

Description

This function returns the authorization status for the current HTTP connection. This is one of the values returned by the TCPIP_HTTP_NET_ConnectionFileAuthenticate function.

Remarks

None.

Preconditions

None.

Example

```
uint8_t isAuth;
```

```
isAuth = TCPIP_HTTP_NET_ConnectionIsAuthorizedGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint8_t TCPIP_HTTP_NET_ConnectionIsAuthorizedGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionGetExecute Function

Processes GET form field variables and cookies.

File

http_net.h

C

```
TCPIP_HTTP_NET_IO_RESULT TCPIP_HTTP_NET_ConnectionGetExecute(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const
TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

- TCPIP_HTTP_NET_IO_RES_DONE - application is done processing
- TCPIP_HTTP_NET_IO_RES_NEED_DATA - this value may not be returned because more data will not become available
- TCPIP_HTTP_NET_IO_RES_WAITING - the application is waiting for an asynchronous process to complete, and this function should be called again later

Description

This function is implemented by the application developer. Its purpose is to parse the data received from URL parameters (GET method forms) and cookies and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated.

When this function is called, the connection data buffer (see TCPIP_HTTP_NET_ConnectionDataBufferGet) contains sequential name/value pairs of strings representing the data received. In this format, TCPIP_HTTP_NET_ArgGet can be used to search for specific variables in the input. If data buffer space associated with this connection is required, connection data buffer may be overwritten here once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including TCPIP_HTTP_NET_ConnectionPostExecute and any TCPIP_HTTP_NET_Print_varname dynamic substitutions.

This function may also issue redirections by setting the connection data buffer to the destination file name or URL, and the connection httpStatus (TCPIP_HTTP_NET_ConnectionStatusSet()) to TCPIP_HTTP_NET_STAT_REDIRECT.

Finally, this function may set cookies. Set connection data buffer to a series of name/value string pairs (in the same format in which parameters arrive) and then set the connection hasArgs (TCPIP_HTTP_NET_ConnectionHasArgsSet) equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in the connection data buffer.

Remarks

This function is only called if variables are received via URL parameters or Cookie arguments.

This function may NOT write to the network transport buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use the connection callbackPos (TCPIP_HTTP_NET_ConnectionCallbackPosGet) or the connection data buffer for storage associated with individual requests.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

TCPIP_HTTP_NET_IO_RESULT TCPIP_HTTP_NET_ConnectionGetExecute

(TCPIP_HTTP_NET_CONN_HANDLE connHandle,

const TCPIP_HTTP_NET_USER_CALLBACK* pCBack)

TCPIP_HTTP_NET_ConnectionPostNameRead Function

Reads a name from a URL encoded string in the network transport buffer.

File

http_net.h

C

```
TCPIP_HTTP_NET_READ_STATUS TCPIP_HTTP_NET_ConnectionPostNameRead(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
uint8_t* cData, uint16_t wLen);
```

Returns

- TCPIP_HTTP_NET_READ_OK - name was successfully read
- TCPIP_HTTP_NET_READ_TRUNCTATED - entire name could not fit in the buffer, so the value was truncated and data has been lost
- TCPIP_HTTP_NET_READ_INCOMPLETE - entire name was not yet in the buffer, so call this function again later to retrieve

Description

This function reads a name from a URL encoded string in the network transport buffer. This function is meant to be called from an TCPIP_HTTP_NET_ConnectionPostExecute callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least two extra bytes are needed in cData over the maximum length of data expected to be read.

This function will read until the next '=' character, which indicates the end of a name parameter. It assumes that the front of the buffer is the beginning of the name parameter to be read.

This function properly updates pHttpCon->byteCount by decrementing it by the number of bytes read. It also removes the delimiting '=' from the buffer.

Remarks

None.

Preconditions

The front of the network transport buffer is the beginning of a name parameter, and the rest of the network transport buffer contains a URL-encoded string with a name parameter terminated by a '=' character.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the name once it is read
wLen	how many bytes can be written to cData

Function

TCPIP_HTTP_NET_READ_STATUS TCPIP_HTTP_NET_ConnectionPostNameRead (TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen)

TCPIP_HTTP_NET_ConnectionHasArgsSet Function

Sets whether there are get or cookie arguments.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionHasArgsSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t args);
```

Returns

None.

Description

The function sets the value of the "cookies or get arguments" that are present.

Remarks

None.

Preconditions

None.

Example

```
else if(!memcmp(filename, "cookies.htm", 11))
{
    TCPIP_HTTP_NET_ConnectionHasArgsSet(connHandle, true);
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
args	boolean if there are arguments or not

Function

void TCPIP_HTTP_NET_ConnectionHasArgsSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t args)

TCPIP_HTTP_NET_ConnectionPostSmGet Function

Get the POST state machine state.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionPostSmGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

16-bit integer POST state machine state.

Description

This function returns the POST state machine state for the connection defined by connHandle. This state is maintained by the HTTP connection and can be used by the user of the HTTP to maintain its own POST state machine. The values of the POST state machine have significance only for the user of the HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
#define SM_POST_LCD_READ_NAME   1
#define SM_POST_LCD_READ_VALUE  2

switch(TCPIP_HTTP_NET_ConnectionPostSmGet(connHandle))
{
    // Find the name
    case SM_POST_LCD_READ_NAME:
      .
      .
      .
    // Found the value, so store the LCD and return
    case SM_POST_LCD_READ_VALUE:
      .
      .
      .
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint16_t TCPIP_HTTP_NET_ConnectionPostSmGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionIsAuthorizedSet Function

Sets the authorized state for the current connection.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionIsAuthorizedSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t auth);
```

Returns

None.

Description

This function sets the authorization status for the current HTTP connection. This has to be one of the values in the set returned by the TCPIP_HTTP_NET_ConnectionFileAuthenticate function.

Remarks

None.

Preconditions

None.

Example

```
uint8_t auth = 0x80;

TCPIP_HTTP_NET_ConnectionIsAuthorizedSet(connHandle, auth);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
auth	new authorization state

Function

void TCPIP_HTTP_NET_ConnectionIsAuthorizedSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t auth)

TCPIP_HTTP_NET_ConnectionPostValueRead Function

Reads a value from a URL encoded string in the network transport buffer.

File

http_net.h

C

```
TCPIP_HTTP_NET_READ_STATUS TCPIP_HTTP_NET_ConnectionPostValueRead(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
uint8_t* cData, uint16_t wLen);
```

Returns

- TCPIP_HTTP_NET_READ_OK - value was successfully read
- TCPIP_HTTP_NET_READ_TRUNCTATED - entire value could not fit in the buffer, so the value was truncated and data has been lost
- TCPIP_HTTP_NET_READ_INCOMPLETE - entire value was not yet in the buffer, so call this function again later to retrieve

Description

This function reads a value from a URL encoded string in the network transport buffer. This function is meant to be called from an TCPIP_HTTP_NET_ConnectionPostExecute callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least two extra bytes are needed in cData above the maximum length of data expected to be read.

This function will read until the next '&' character, which indicates the end of a value parameter. It assumes that the front of the buffer is the beginning of the value parameter to be read. If pHttpCon->byteCount indicates that all expected bytes are in the buffer, it assumes that all remaining data is the value and acts accordingly.

This function properly updates pHttpCon->byteCount by decrementing it by the number of bytes read. The terminating '&' character is also removed from the buffer.

Remarks

None.

Preconditions

The front of the network transport buffer is the beginning of a name parameter, and the rest of the network transport buffer contains a URL-encoded string with a name parameter terminated by a '=' character.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the value once it is read
wLen	how many bytes can be written to cData

Function

TCPIP_HTTP_NET_READ_STATUS TCPIP_HTTP_NET_ConnectionPostValueRead (TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen)

TCPIP_HTTP_NET_ConnectionNetHandle Function

Returns the network handle of the current connection.

File

http_net.h

C

```
TCPIP_NET_HANDLE TCPIP_HTTP_NET_ConnectionNetHandle(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

The connection network handle.

Description

This function returns the network handle over which the current HTTP connection communicates.

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle

Function

TCPIP_NET_HANDLE TCPIP_HTTP_NET_ConnectionNetHandle(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionSocketGet Function

Get the socket for the current connection.

File

http_net.h

C

```
NET_PRES_SKT_HANDLE_T TCPIP_HTTP_NET_ConnectionSocketGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

NET_PRES_SKT_HANDLE_T for the connection defined by connHandle.

Description

The function returns the network transport socket of the specified HTTP connection. The user gets access to the connection socket which it can use for debugging or directly sending/reading data.

Remarks

This function gives direct access to the underlying transport socket. It is meant for test/advanced usage only. The regular connection functions should be used for manipulation of the connection data. Using the socket directly for data manipulation will disrupt the HTTP server functionality.

Preconditions

None.

Example

```
uint32_t byteCount;
int sktRxSize;

byteCount = TCPIP_HTTP_NET_ConnectionByteCountGet(connHandle);
sktRxSize = TCPIP_HTTP_NET_ConnectionReadBufferSize(connHandle);

if(byteCount > sktRxSize)
{   // Configuration Failure
    TCPIP_HTTP_NET_ConnectionStatusSet(connHandle, TCPIP_HTTP_NET_STAT_REDIRECT);
    return TCPIP_HTTP_NET_IO_RES_DONE;
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

NET_PRES_SKT_HANDLE_T TCPIP_HTTP_NET_ConnectionSocketGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionPeek Function

Reads a specified number of data bytes from the connection RX buffer without removing them from the buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionPeek(TCPIP_HTTP_NET_CONN_HANDLE connHandle, void * buffer, uint16_t size);
```

Description

The function allows peeking into the connection buffer. The data will still be available for a next read operation.

Remarks

None

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle
buffer	Destination to write the peeked data bytes
size	Length of bytes to peek from the connection RX buffer

Function

uint16_t TCPIP_HTTP_NET_ConnectionPeek(TCPIP_HTTP_NET_CONN_HANDLE connHandle,

void * buffer, uint16_t size);

TCPIP_HTTP_NET_ConnectionStatusGet Function

Gets HTTP status.

File

http_net.h

C

```
TCPIP_HTTP_NET_STATUS TCPIP_HTTP_NET_ConnectionStatusGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

A TCPIP_HTTP_NET_STATUS value.

Description

This function returns the current HTTP status of the selected HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
TCPIP_HTTP_NET_STATUS currStat =  TCPIP_HTTP_NET_ConnectionStatusGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

TCPIP_HTTP_NET_STATUS TCPIP_HTTP_NET_ConnectionStatusGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionPostExecute Function

Processes POST form variables and data.

File

http_net.h

C

```
TCPIP_HTTP_NET_IO_RESULT TCPIP_HTTP_NET_ConnectionPostExecute(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const
TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

- TCPIP_HTTP_NET_IO_RES_DONE - application is done processing
- TCPIP_HTTP_NET_IO_RES_NEED_DATA - more data is needed to continue, and this function should be called again later
- TCPIP_HTTP_NET_IO_RES_WAITING - the application is waiting for an asynchronous process to complete, and this function should be called again later

Description

This function is implemented by the application developer. Its purpose is to parse the data received from POST forms and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated before this function is called.

When this function is called, POST data will be waiting in the network transport buffer. The connection byteCount (see TCPIP_HTTP_NET_ConnectionByteCountGet will indicate the number of bytes remaining to be received before the browser request is complete.

Since data is still in the network transport buffer, the application must call TCPIP_HTTP_NET_ConnectionRead in order to retrieve bytes. When this is done, connection byteCount MUST be updated to reflect how many bytes now remain.

In general, data submitted from web forms via POST is URL encoded. The TCPIP_HTTP_NET_URLDecode function can be used to decode this information back to a standard string if required. If data buffer space associated with this connection is required, the connection data buffer (see TCPIP_HTTP_NET_ConnectionDataBufferGet) may be overwritten here once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including TCPIP_HTTP_NET_ConnectionPostExecute and any TCPIP_HTTP_NET_Print_varname dynamic substitutions.

Whenever a POST form is processed it is recommended to issue a redirect back to the browser, either to a status page or to the same form page that was posted. This prevents accidental duplicate submissions (by clicking refresh or back/forward) and avoids browser warnings about "resubmitting form data". Redirects may be issued to the browser by setting the connection data buffer to the destination file or URL, and the connection httpStatus (TCPIP_HTTP_NET_ConnectionStatusSet) to TCPIP_HTTP_NET_STAT_REDIRECT.

Finally, this function may set cookies. Set the connection data buffer to a series of name/value string pairs (in the same format in which parameters arrive), and then set the connection hasArgs (TCPIP_HTTP_NET_ConnectionHasArgsSet) equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in the connection data buffer.

Remarks

This function is only called when the request method is POST, and is only used when TCPIP_HTTP_NET_USE_POST is defined.

This method may NOT write to the network transport buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use the connection callbackPos (TCPIP_HTTP_NET_ConnectionCallbackPosGet) or connection data buffer for storage associated with individual requests.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

TCPIP_HTTP_NET_IO_RESULT TCPIP_HTTP_NET_ConnectionPostExecute
(
TCPIP_HTTP_NET_CONN_HANDLE connHandle,
const
TCPIP_HTTP_NET_USER_CALLBACK* pCBack)

TCPIP_HTTP_NET_ConnectionUserDataGet Function

Gets the user data parameter for the current connection.

File

http_net.h

C

```
const void* TCPIP_HTTP_NET_ConnectionUserDataGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

User data that's stored as part of the connection.

Description

This function returns the user data value for the current HTTP connection. This data belongs to the user and is not used in any way by the HTTP server module. It can be set by the user with TCPIP_HTTP_NET_ConnectionUserDataSet.

Remarks

None.

Preconditions

None.

Example

```
uint32_t myConnData;

myConnData = (uint32_t)TCPIP_HTTP_NET_ConnectionUserDataGet(connHandle);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

const void* TCPIP_HTTP_NET_ConnectionUserDataGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle)

TCPIP_HTTP_NET_ConnectionPostSmSet Function

Set the POST state machine state.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionPostSmSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint16_t state);
```

Returns

None.

Description

This function sets the POST state machine state for the connection defined by connHandle. This state is maintained by the HTTP connection and can be used by the user of the HTTP to maintain its own POST state machine. The values of the POST state machine have significance only for the user of the HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
uint8_t* httpDataBuff;
#define SM_POST_LCD_READ_NAME   1
#define SM_POST_LCD_READ_VALUE  2

switch(TCPIP_HTTP_NET_ConnectionPostSmGet(connHandle))
{
    // Find the name
    case SM_POST_LCD_READ_NAME:

        // Read a name
        if(TCPIP_HTTP_NET_ConnectionPostNameRead(connHandle, httpDataBuff,
           TCPIP_HTTP_NET_MAX_DATA_LEN) == TCPIP_HTTP_NET_READ_INCOMPLETE)
            return TCPIP_HTTP_NET_IO_RES_NEED_DATA;

        TCPIP_HTTP_NET_ConnectionPostSmSet(connHandle, SM_POST_LCD_READ_VALUE);
        // No break...continue reading value

    // Found the value, so store the LCD and return
    case SM_POST_LCD_READ_VALUE:
      .
      .
      .
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
state	16 bit integer state for POST state machine

Function

void TCPIP_HTTP_NET_ConnectionPostSmSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
uint16_t state)

TCPIP_HTTP_NET_ConnectionRead Function

Reads an array of data bytes from a connection's RX buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionRead(TCPIP_HTTP_NET_CONN_HANDLE connHandle, void * buffer, uint16_t size);
```

Returns

The number of bytes read from the socket. If less than len, the connection RX buffer became empty or the underlying socket is not connected.

Description

This function reads an array of data bytes from the connection RX buffer. The data is removed from the FIFO in the process.

Remarks

If the supplied buffer is null, the data is simply discarded.

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle
buffer	The pointer to the array to store data that was read
size	The number of bytes to be read.

Function

uint16_t TCPIP_HTTP_NET_ConnectionRead(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
void * buffer, uint16_t size);

TCPIP_HTTP_NET_ConnectionReadBufferSize Function

Returns the size of the connection RX buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionReadBufferSize(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

The size of the connection RX buffer.

Description

This function discards data from the connection RX buffer.

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle

Function

uint16_t TCPIP_HTTP_NET_ConnectionReadBufferSize(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionReadIsReady Function

Determines how many bytes can be read from the connection RX buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionReadIsReady(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

The number of bytes available to be read from the connection RX buffer.

Description

This function determines how many bytes can be read from the connection RX buffer.

Remarks

When using an encrypted connection the number of available unencrypted bytes may turn out to be different than what this function returns.

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	connection handle

Function

uint16_t TCPIP_HTTP_NET_ConnectionReadIsReady(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionStatusSet Function

Sets HTTP status.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionStatusSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_NET_STATUS stat);
```

Returns

None.

Description

Allows write access to the HTTP status of the selected HTTP connection.

Remarks

None.

Preconditions

None.

Example

```
byteCount = TCPIP_HTTP_NET_ConnectionByteCountGet(connHandle);
int sktRxSize;

sktRxSize = TCPIP_HTTP_NET_ConnectionReadBufferSize(connHandle);

if(byteCount > sktRxSize)
{   // Configuration Failure
    // 302 Redirect will be returned
    TCPIP_HTTP_NET_ConnectionStatusSet(connHandle, TCPIP_HTTP_NET_STAT_REDIRECT);
}
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
stat	new TCPIP_HTTP_NET_STATUS enumeration value.

Function

void TCPIP_HTTP_NET_ConnectionStatusSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
TCPIP_HTTP_NET_STATUS stat)

TCPIP_HTTP_NET_ConnectionStringFind Function

Helper to find a string of characters in an incoming connection buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionStringFind(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* str,
uint16_t startOffs, uint16_t searchLen);
```

Returns

- If string was found - a zero-indexed position in the RX buffer of the string occurrence
- 0xFFFF - Search array not found

Description

This function searches for an ASCIIZ string in the RX buffer of the connection. It does the search by performing a peek operation in the RX buffer, (i.e., the RX data in the buffer is not consumed and it is available for further read operations). It works for both encrypted and unencrypted connections.

Remarks

Note that the search will fail if there's more data in the TCP socket than could be read at once.

Preconditions

connHandle - a valid HTTP connection.

Parameters

Parameters	Description
connHandle	HTTP connection handle
str	0 terminated ASCII string to search for
startOffs	offset in the RX buffer to start the search from
searchLen	if !0 it is the length of buffer to search into (starting from startOffs); if 0, the whole buffer is searched

Function

uint16_t TCPIP_HTTP_NET_ConnectionStringFind(TCPIP_HTTP_NET_CONN_HANDLE connHandle,

const char* str, uint16_t startOffs, uint16_t searchLen);

TCPIP_HTTP_NET_ConnectionUserAuthenticate Function

Performs validation on a specific user name and password.

File

http_net.h

C

```
uint8_t TCPIP_HTTP_NET_ConnectionUserAuthenticate(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* cUser,
const char* cPass, const TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

- <= 0x79 - the credentials were rejected
- >= 0x80 - access is granted for this connection

Description

This function is implemented by the application developer. Its function is to determine if the user name and password supplied by the client are acceptable for this resource. This callback function can thus to determine if only specific user names or passwords will be accepted for this resource.

Return values 0x80 - 0xff indicate that the credentials were accepted, while values from 0x00 to 0x79 indicate that authorization failed. While most applications will only use a single value to grant access, flexibility is provided to store multiple values in order to indicate which user (or user's group) logged in.

The value returned by this function is stored in the corresponding connection data and will be available with TCPIP_HTTP_NET_ConnectionIsAuthorizedGet in any of the TCPIP_HTTP_NET_ConnectionGetExecute, TCPIP_HTTP_NET_ConnectionPostExecute, or TCPIP_HTTP_NET_Print_varname callbacks.

Remarks

This function is only called when an Authorization header is encountered.

This function may NOT write to the network transport buffer.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cUser	the user name supplied by the client
cPass	the password supplied by the client
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

uint8_t TCPIP_HTTP_NET_ConnectionUserAuthenticate(TCPIP_HTTP_NET_CONN_HANDLE

connHandle, const char* cUser, const char* cPass,

const TCPIP_HTTP_NET_USER_CALLBACK* pCBack)

TCPIP_HTTP_NET_ConnectionUserDataSet Function

Sets the user data parameter for the current connection.

File

http_net.h

C

```
void TCPIP_HTTP_NET_ConnectionUserDataSet(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const void* uData);
```

Returns

None.

Description

This function will set the user data value for the current HTTP connection. This data belongs to the user and is not used in any way by the HTTP server module. It is available to the user by calling TCPIP_HTTP_NET_ConnectionUserDataGet.

Remarks

None.

Preconditions

None.

Example

```
uint32_t myConnData;

TCPIP_HTTP_NET_ConnectionUserDataSet(connHandle, (const void*)myConnData);
```

Parameters

Parameters	Description
connHandle	HTTP connection handle
uData	user supplied data

Function

void TCPIP_HTTP_NET_ConnectionUserDataSet(TCPIP_HTTP_NET_CONN_HANDLE

connHandle, const void* uData)

TCPIP_HTTP_NET_DynAcknowledge Function

Dynamic variable acknowledge function

File

http_net.h

C

```
void TCPIP_HTTP_NET_DynAcknowledge(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const void* buffer, const struct
_tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

None.

Description

Functions in this style are implemented by the application developer. This function reports that a buffer that was passed as part of a dynamic variable callback (dynamicPrint) was processed and can be reused, freed, etc.

Remarks

The function is called only when TCPIP_HTTP_NET_DynamicWrite or TCPIP_HTTP_NET_DynamicWriteString was specified with a true needAck parameter.

The function is called from withing HTTP context. It should be kept as short as possible.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
buffer	the processed buffer
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

void TCPIP_HTTP_NET_DynAcknowledge(TCPIP_HTTP_NET_CONN_HANDLE connHandle,

const void* buffer,

const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);

TCPIP_HTTP_NET_DynamicWrite Function

Writes a data buffer to the current connection

File

http_net.h

C

```
bool TCPIP_HTTP_NET_DynamicWrite(const TCPIP_HTTP_DYN_VAR_DCPT* varDcpt, const void * buffer, uint16_t
size, bool needAck);
```

Returns

True if the data buffer has been queued for output. False if an invalid buffer was provided or the buffer could not be queued because of the lack of resources (descriptors).

Description

This function takes a buffer and sends it over the HTTP connection as part of the HTTP dynamic variable processing

Remarks

The buffer passed in by the user with this call is queued internally using an available dynamic variable buffer descriptor. That means that the buffer has to be persistent. Once the buffer is processed and sent to output, the dynamicAck callback will be called, to inform the user that the corresponding buffer can be reused/freed.

When multiple connections output their dynamic content concurrently the HTTP may run out of dynamic variable buffer descriptors that are used in queuing the requests and the call may fail. If the call failed, because the buffer could not be queued, it may be retried using by returning TCPIP_HTTP_DYN_PRINT_RES_AGAIN in the TCPIP_HTTP_NET_DynPrint callback.

If sequential write calls are done from within the same TCPIP_HTTP_NET_DynPrint call the HTTP module will try to append the new dynamic data to the existent one.

The number of internal HTTP dynamic variables buffer descriptors is controlled by TCPIP_HTTP_NET_DYNVAR_DESCRIPTORS_NUMBER. It can be obtained at run-time using the TCPIP_HTTP_NET_ConnectionDynamicDescriptors function.

Preconditions

varDcpt - a valid dynamic variable descriptor.

Parameters

Parameters	Description
varDcpt	dynamic variable descriptor as passed in the TCPIP_HTTP_NET_DynPrint function
buffer	The pointer to the persistent buffer to be written to the HTTP connection as part of this dynamic variable callback
size	The number of bytes to be written
needAck	if true, once the buffer is processed internally, TCPIP_HTTP_NET_DynAcknowledge will be called

Function

bool TCPIP_HTTP_NET_DynamicWrite(const TCPIP_HTTP_DYN_VAR_DCPT* varDcpt,

const void * buffer, uint16_t size, bool needAck);

TCPIP_HTTP_NET_DynamicWriteString Function

Helper for writing a string within a dynamic variable context.

File

http_net.h

C

```
bool TCPIP_HTTP_NET_DynamicWriteString(const TCPIP_HTTP_DYN_VAR_DCPT* varDcpt, const char* str, bool needAck);
```

Returns

True if the data buffer has been queued for output. False if an invalid buffer was provided or the buffer could not be queued because of the lack of resources (descriptors).

Description

This function takes a 0 terminated ACII string and sends it to the HTTP connection as part of the HTTP dynamic variable processing

Remarks

This is just a helper. The actual function called is still TCPIP_HTTP_NET_DynamicWrite. That means that the supplied string has to be persistent.

See the remarks for TCPIP_HTTP_NET_DynamicWrite.

Preconditions

varDcpt - a valid dynamic variable descriptor.

Parameters

Parameters	Description
varDcpt	dynamic variable descriptor as passed in the TCPIP_HTTP_NET_DynPrint function
str	The string to be written
needAck	if true, once the buffer is processed internally, TCPIP_HTTP_NET_DynAcknowledge will be called

Function

bool TCPIP_HTTP_NET_DynamicWriteString(const TCPIP_HTTP_DYN_VAR_DCPT* varDcpt,

const char* str, bool needAck);

TCPIP_HTTP_NET_DynPrint Function

Inserts dynamic content into a web page

File

http_net.h

C

```
TCPIP_HTTP_DYN_PRINT_RES TCPIP_HTTP_NET_DynPrint(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const
TCPIP_HTTP_DYN_VAR_DCPT* varDcpt, const TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN/TCPIP_HTTP_DYN_PRINT_RES_AGAIN. Typically this is used when outputting large amounts of data that cannot fit into one single buffer write operation or when the data is not available all at once.

An alternative (legacy) mechanism for that, is using the connection callbackPos calls: (TCPIP_HTTP_NET_ConnectionCallbackPosGet/TCPIP_HTTP_NET_ConnectionCallbackPosSet). This value will be set to zero before the function is called. If the function is managing its output state, it must set this to a non-zero value before returning. If the connection callbackPos is non-zero, the function will be called again after the current data is processed (equivalent to returning TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN). Once the callback completes, set the callback value back to zero and return TCPIP_HTTP_DYN_PRINT_RES_DONE to resume normal servicing of the request.

If the function does not process this dynamic variable callback (or it wants the default action to be performed), it should return TCPIP_HTTP_DYN_PRINT_RES_DEFAULT. Then the default action for that variable will be performed by the HTTP module.

Note that this is an advanced feature and will be used for processing of HTTP dynamic variable keywords. No implementation currently exists for user added variables and the default action in this case will do nothing.

A TCPIP_HTTP_DYN_PRINT_RES specifying the action to be taken. See the TCPIP_HTTP_DYN_PRINT_RES definition for a list of allowed actions

Description

Functions in this style are implemented by the application developer. This function generates dynamic content to be inserted into web pages.

The function of this type is called when a dynamic variable is located in a web page. (i.e., ~varname~) The name between the tilde '~' characters is passed as the dynamic variable name.

The function currently supports int32_t and string parameters. The dynamic variable parameters are included in the varDcpt data structure that's passed at the time of the call. For example, the variable "~myArray(2,6)~" will generate the call with: varDcpt.dynName = "myArray"; varDcpt.nArgs = 2; varDcpt.dynArgs->argType = TCPIP_HTTP_DYN_ARG_TYPE_INT32; varDcpt.dynArgs->argInt32 = 2; (varDcpt.dynArgs + 1)->argType = TCPIP_HTTP_DYN_ARG_TYPE_INT32; (varDcpt.dynArgs + 1)->argInt32 = 6;

When called, this function should write its output to the HTTP connection using the TCPIP_HTTP_NET_DynamicWrite/TCPIP_HTTP_NET_DynamicWriteString functions.

In situations where the dynamic variable print function needs to perform additional write operations, or simply needs to be called again, it must

Remarks

This function may service multiple HTTP requests simultaneously, especially when managing its output state. Exercise caution when using global or static variables inside this routine. Use the connection callbackPos or the connection data buffer for storage associated with individual requests.

The varDcpt variable descriptor is valid only in the context of this call. Any parameters that are needed for further processing should be copied to user storage.

The varDcpt variable descriptor is constant and should not be modified in any way by the callback.

If the callback returned TCPIP_HTTP_DYN_PRINT_RES_DEFAULT, no further calls will be made to the callback function.

Returning TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN/TCPIP_HTTP_DYN_PRINT_RES_AGAIN is an alternative mechanism to manipulation the connection status with TCPIP_HTTP_NET_ConnectionCallbackPosSet/TCPIP_HTTP_NET_ConnectionCallbackPosGet functions (which mechanism is still valid).

- If the TCPIP_HTTP_NET_DynPrint() returned TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN, the HTTP server will start sending the dynamic variable buffer (invoked by TCPIP_HTTP_NET_DynamicWrite/TCPIP_HTTP_NET_DynamicWriteString). Once sending is done, the HTTP server will again call TCPIP_HTTP_NET_DynPrint regardless of the value set with the callback position.
- If the TCPIP_HTTP_NET_DynPrint() returned TCPIP_HTTP_DYN_PRINT_RES_AGAIN, there will be no processing of dynamic variable data (potentially written to the connection) but the TCPIP_HTTP_NET_DynPrint will be invoked again. Returning TCPIP_HTTP_DYN_PRINT_RES_AGAIN is needed when TCPIP_HTTP_NET_DynamicWrite failed or some condition is not satisfied and the TCPIP_HTTP_NET_DynPrint needs to be called again.

The callback position mechanism will be evaluated after the TCPIP_HTTP_NET_DynPrint returns TCPIP_HTTP_DYN_PRINT_RES_DONE.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
varDcpt	pointer to a variable descriptor containing the dynamic variable name and arguments valid within this call context only.
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

```
TCPIP_HTTP_DYN_PRINT_RES TCPIP_HTTP_NET_DynPrint(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
const                    TCPIP_HTTP_DYN_VAR_DCPT* varDcpt,
const                    TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

TCPIP_HTTP_NET_EventReport Function

Reports an event that occurred in the processing of a HTTP web page

File

http_net.h

C

```
void TCPIP_HTTP_NET_EventReport(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_NET_EVENT_TYPE evType,
const void* evInfo, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

None.

Description

Functions in this style are implemented by the application developer. This function reports an event that occurred when processing a file, dynamic variable, etc.

Remarks

The evInfo parameter is valid only in the context of this call. Any data that is needed for further processing should be copied to user storage.

This function may NOT write to the network transport buffer.

Preconditions

None.

Parameters

Parameters	Description
connHandle	HTTP connection handle
evType	the HTTP event that occurred

evInfo	additional info for that particular event • TCPIP_HTTP_NET_EVENT_FS_UPLOAD_COMPLETE: file name pointer • TCPIP_HTTP_NET_EVENT_FILE_OPEN_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_FILE_NAME_ERROR: 0 • TCPIP_HTTP_NET_EVENT_FILE_NAME_SIZE_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_FILE_SIZE_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_FILE_READ_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_DEPTH_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_DYNVAR_PARSE_ERROR: dynamic variable name pointer • TCPIP_HTTP_NET_EVENT_FS_WRITE_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_FS_MOUNT_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_CHUNK_POOL_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_FILE_BUFFER_POOL_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_DYNVAR_ALLOC_ERROR: dynamic variable name pointer • TCPIP_HTTP_NET_EVENT_UPLOAD_ALLOC_ERROR: file name pointer • TCPIP_HTTP_NET_EVENT_SSI_PARSE_ERROR: SSI command pointer • TCPIP_HTTP_NET_EVENT_SSI_COMMAND_ERROR: SSI command pointer • TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_ERROR: SSI command pointer • TCPIP_HTTP_NET_EVENT_SSI_ALLOC_DESCRIPTOR_ERROR: SSI command pointer • TCPIP_HTTP_NET_EVENT_DYNVAR_ARG_NAME_TRUNCATED: dynamic variable name pointer • TCPIP_HTTP_NET_EVENT_DYNVAR_ARG_NUMBER_TRUNCATED: dynamic variable name pointer • TCPIP_HTTP_NET_EVENT_DYNVAR_RETRIES_EXCEEDED: dynamic variable name pointer • TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_NUMBER_TRUNCATED: SSI argument pointer • TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_UNKNOWN: SSI attribute pointer • TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_NUMBER_MISMATCH: SSI command pointer • TCPIP_HTTP_NET_EVENT_SSI_VAR_NUMBER_EXCEEDED: SSI variable pointer • TCPIP_HTTP_NET_EVENT_SSI_VAR_UNKNOWN: SSI variable pointer • TCPIP_HTTP_NET_EVENT_SSI_VAR_VOID: SSI variable pointer • TCPIP_HTTP_NET_EVENT_SSI_VAR_DELETED: SSI variable pointer • TCPIP_HTTP_NET_EVENT_CHUNK_POOL_EMPTY: file name pointer • TCPIP_HTTP_NET_EVENT_FILE_BUFFER_POOL_EMPTY: file name pointer • TCPIP_HTTP_NET_EVENT_PEEK_BUFFER_SIZE_EXCEEDED: string with excess number of bytes that cannot be read Valid within this call context only!
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

void TCPIP_HTTP_NET_EventReport(TCPIP_HTTP_NET_CONN_HANDLE connHandle,
TCPIP_HTTP_NET_EVENT_TYPE evType, const void* evInfo,
const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);

TCPIP_HTTP_NET_Task Function

Standard TCP/IP stack module task function.

File

http_net.h

C

```
void TCPIP_HTTP_NET_Task();
```

Returns

None.

Description

This function performs HTTP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The HTTP module should have been initialized.

Function

void TCPIP_HTTP_NET_Task(void)

TCPIP_HTTP_NET_URLDecode Function

Parses a string from URL encoding to plain text.

File

http_net.h

C

```
uint8_t* TCPIP_HTTP_NET_URLDecode(uint8_t* cData);
```

Returns

A pointer to the last null terminator in data, which is also the first free byte for new data.

Description

This function parses a string from URL encoding to plain text. The following conversions are made: ‘=’ to ‘0’, ‘&’ to ‘0’, ‘+’ to ‘ ‘, and “%xx” to a single hex byte.

After completion, the data has been decoded and a null terminator signifies the end of a name or value. A second null terminator (or a null name parameter) indicates the end of all the data.

Remarks

This function is called by the stack to parse GET arguments and cookie data. User applications can use this function to decode POST data, but first need to verify that the string is null-terminated.

Preconditions

The data parameter is null terminated and has at least one extra byte free.

Parameters

Parameters	Description
cData	The string which is to be decoded in place.

Function

uint8_t* TCPIP_HTTP_NET_URLDecode(uint8_t* cData)

TCPIP_HTTP_NET_UserHandlerDeregister Function

Deregisters a previously registered HTTP user handler.

File

http_net.h

C

```
bool TCPIP_HTTP_NET_UserHandlerDeregister(TCPIP_HTTP_NET_USER_HANDLE hHttp);
```

Returns

- true - if the call succeeds
- false - if no such handler is registered or there are active connections

Description

This function deregisters a HTTP user callback handler.

Remarks

The call will fail if there is active HTTP traffic. The handler cannot be deregistered while HTTP traffic is in progress.

Preconditions

The HTTP server module properly initialized.

Parameters

Parameters	Description
hHttp	A handle returned by a previous call to TCPIP_HTTP_NET_UserHandlerRegister

Function

TCPIP_HTTP_NET_UserHandlerDeregister(TCPIP_HTTP_NET_USER_HANDLE hHttp)

TCPIP_HTTP_NET_UserHandlerRegister Function

Registers a user callback structure with the HTTP server

File

http_net.h

C

```
TCPIP_HTTP_NET_USER_HANDLE TCPIP_HTTP_NET_UserHandlerRegister(const TCPIP_HTTP_NET_USER_CALLBACK*
userCallback);
```

Returns

Returns a valid handle if the call succeeds, or a null handle if the call failed (out of memory, for example).

Description

The function registers a user callback data structure with the HTTP server. The HTTP server will call the user supplied callbacks at run-time when processing the web pages.

Remarks

Currently only one user callback structure could be registered. The call will fail if a user callback structure is already registered.

Preconditions

The HTTP server module properly initialized.

Example

Parameters

Parameters	Description
userCallback	user callback to be registered with the HTTP server

Function

TCPIP_HTTP_NET_USER_HANDLE TCPIP_HTTP_NET_UserHandlerRegister
(const TCPIP_HTTP_NET_USER_CALLBACK* userCallback);

TCPIP_HTTP_NET_ConnectionDataBufferSizeGet Function

Returns the size of the connection general purpose data buffer.

File

http_net.h

C

```
uint16_t TCPIP_HTTP_NET_ConnectionDataBufferSizeGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

Size of the connection's general purpose data buffer.

Description

This function returns the size of the HTTP connection internal data buffer.

Remarks

This is the parameter that was used for HTTP initialization in TCPIP_HTTP_NET_MODULE_CONFIG::dataLen.

Preconditions

None.

Example

```
  // Read a name
  uint8_t* httpDataBuff = TCPIP_HTTP_NET_ConnectionDataBufferGet(connHandle);
  uint16_t httpDataLen = TCPIP_HTTP_NET_ConnectionDataBufferSizeGet(connHandle);
  if(TCPIP_HTTP_NET_ConnectionPostNameRead(connHandle, httpDataBuff, httpDataLen) ==
TCPIP_HTTP_NET_READ_INCOMPLETE)
  {
      return TCPIP_HTTP_NET_IO_RES_NEED_DATA;
  }
```

Parameters

Parameters	Description
connHandle	HTTP connection handle

Function

uint16_t TCPIP_HTTP_NET_ConnectionDataBufferSizeGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

TCPIP_HTTP_NET_ConnectionDynamicDescriptors Function

Returns the number of dynamic variable descriptors

File

http_net.h

C

```
int TCPIP_HTTP_NET_ConnectionDynamicDescriptors();
```

Returns

The number of descriptors allocated by the HTTP module.

Description

This function returns the number of the dynamic variable buffer descriptors that are allocated by the HTTP for dynamic variable processing.

Remarks

Currently the dynamic variable descriptors are allocated at the HTTP initialization and cannot be changed at run-time.

Preconditions

None

Function

int TCPIP_HTTP_NET_ConnectionDynamicDescriptors(void);

TCPIP_HTTP_NET_SSINotification Function

Reports an SSI processing event that occurs in the processing of a HTTP web page

File

http_net.h

C

```
bool TCPIP_HTTP_NET_SSINotification(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_SSI_NOTIFY_DCPT*
pSSINotifyDcpt, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
```

Returns

true - if the SSI processing is complete and no further action needs to be taken by the HTTP server false - the HTTP should process the SSI command

Description

Functions in this style are implemented by the application developer. This function is used by the HTTP server to report that an SSI command line was encountered while processing a file. The user has a chance to inspect the line and to provide custom action by using the SSI API functions.

Remarks

The ssiLine parameter is valid only in the context of this call. Any data that is needed for further processing should be copied to user storage.

This function could use the SSI API functions to alter/examine the value of SSI variables that are part of the ssiLine.

This function may NOT write to the network transport buffer. Possiblity to write to the transport channel could be eventually added, based on the dynamic variable model TCPIP_HTTP_NET_DynamicWrite.

The default return value should be false: the user function examines/adjusts value of SSI variables and instructs the HTTP module continue normally. However, the user has the option of supressing the output (SSI "echo") or override the value of a variable (SSI "set"), etc.

Preconditions

SSI processing should be enabled.

Parameters

Parameters	Description
connHandle	HTTP connection handle
pSSINotifyDcpt	pointer to a SSI notification descriptor containing:
fileName	the name of the file the SSI command belongs to
ssiCommand	the SSI command parsed from the command line
nAttribs	number of attribute descriptors in the command
pAttrDcpt	pointer to an array of descriptors parsed from this SSI command
pCBack	user TCPIP_HTTP_NET_USER_CALLBACK pointer

Function

bool TCPIP_HTTP_NET_SSINotification(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_SSI_NOTIFY_DCPT* pSSINotifyDcpt, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);

TCPIP_HTTP_NET_SSIVariableDelete Function

Function to delete an SSI variable.

File

http_net.h

C

```
bool TCPIP_HTTP_NET_SSIVariableDelete(const char* varName);
```

Returns

true - if the variable existed and was deleted false - the variable didn't exist

Description

This function deletes an SSI variable if it exists.

Remarks

None.

Preconditions

The HTTP module should have been initialized. SSI should be enabled.

Parameters

Parameters	Description
varName	the SSI variable name

Function

bool TCPIP_HTTP_NET_SSIVariableDelete(const char* varName);

TCPIP_HTTP_NET_SSIVariableGet Function

Function to get access to an existing SSI variable.

File

http_net.h

C

```
const char* TCPIP_HTTP_NET_SSIVariableGet(const char* varName, TCPIP_HTTP_DYN_ARG_TYPE* pVarType, int32_t*
pVarInt);
```

Returns

a valid pointer to the SSI variable string representation if the variable was found 0 if there is no variable with such name

Description

This function performs a search for the corresponding SSI variable name and returns a pointer to the variable string representation.

Remarks

The returned value points to the internal SSI variable representation. This pointer should not be used for changing the SSI variable value.

Preconditions

The HTTP module should have been initialized. SSI should be enabled.

Parameters

Parameters	Description
varName	the SSI variable to search for
pVarType	address to store the type of the SSI variable
pVarInt	address to store the integer value of this variable, if the variable exists and is an integer else this address won't be modified Could be NULL if not needed

Function

const char* TCPIP_HTTP_NET_SSIVariableGet(const char* varName, TCPIP_HTTP_DYN_ARG_TYPE* pVarType, int32_t* pVarInt);

TCPIP_HTTP_NET_SSIVariableGetByIndex Function

Function to get access to an existing SSI variable.

File

http_net.h

C

```
const char* TCPIP_HTTP_NET_SSIVariableGetByIndex(int varIndex, const char** pVarName,
TCPIP_HTTP_DYN_ARG_TYPE* pVarType, int32_t* pVarInt);
```

Returns

a valid pointer to the SSI variable string representation if the variable was found 0 if there is no variable with such name

Description

This function accesses the corresponding SSI variable by its index and returns a pointer to the variable string representation.

Remarks

The returned value points to the internal SSI variable representation. This pointer should not be used for changing the SSI variable value.

The pVarName points to the internal SSI variable name representation. This pointer MUST NOT be used for changing the SSI variable name.

None.

Preconditions

The HTTP module should have been initialized. SSI should be enabled.

Parameters

Parameters	Description
varIndex	the index of the SSI variable to search for Should be < TCPIP_HTTP_NET_SSIVariablesNumberGet()
pVarName	address to store a pointer to the variable name
pVarType	address to store the type of the SSI variable
pVarInt	address to store the integer value of this variable, if the variable exists and is an integer else this address won't be modified Could be NULL if not needed

Function

const char* TCPIP_HTTP_NET_SSIVariableGetByIndex(int varIndex, const char** pVarName, TCPIP_HTTP_DYN_ARG_TYPE* pVarType, int32_t* pVarInt);

TCPIP_HTTP_NET_SSIVariableSet Function

Function to set an SSI variable.

File

http_net.h

C

```
bool TCPIP_HTTP_NET_SSIVariableSet(const char* varName, TCPIP_HTTP_DYN_ARG_TYPE varType, const char* strValue, int32_t intValue);
```

Returns

true - if the variable was updated or created successfully false - the variable didn't exist and the attempt to create it failed (all slots already in use. Increase TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER).

Description

This function sets the new values for an SSI variable. If a variable with such name does not exist, it is created.

Remarks

The string variable interpretation is needed even if the variable is of integer type. HTTP module will use that representation instead of doing an itoa conversion on intValue. value points to the internal SSI variable representation.

The string representation should not exceed TCPIP_HTTP_NET_SSI_VARIABLE_STRING_MAX_LENGTH. Any excess variables will be truncated.

Preconditions

The HTTP module should have been initialized. SSI should be enabled.

Parameters

Parameters	Description
varName	the SSI variable name
varType	the type of the SSI variable
strValue	pointer to the string representation of the variable
intValue	the integer value of the variable, if the type is integer.

Function

bool TCPIP_HTTP_NET_SSIVariableSet(const char* varName, TCPIP_HTTP_DYN_ARG_TYPE varType, const char* strValue, int32_t intValue);

TCPIP_HTTP_NET_SSIVariablesNumberGet Function

Function to get the number of the current SSI variables.

File

http_net.h

C

```
int TCPIP_HTTP_NET_SSIVariablesNumberGet(int* pMaxNo);
```

Returns

The number of current SSI variables that are in use

Description

This function returns the number of SSI variables that currently exist. It will also return the maximum number of variables that the SSI can hold.

Remarks

None.

Preconditions

The HTTP module should have been initialized. SSI should be enabled.

Parameters

Parameters	Description
pMaxNo	pointer to store the maximum number of SSI variables that can exist Can be NULL if not needed

Function

int TCPIP_HTTP_NET_SSIVariablesNumberGet(int* pMaxNo);

TCPIP_HTTP_NET_ConnectionHandleGet Function

Gets the connection handle of a HTTP connection.

File

http_net.h

C

```
TCPIP_HTTP_NET_CONN_HANDLE TCPIP_HTTP_NET_ConnectionHandleGet(int connIx);
```

Returns

A valid connection handle if the connection index is valid 0 if there is no such connection

Description

This function will return the connection handle of the requested HTTP connection index.

Remarks

None

Preconditions

None.

Example

```
TCPIP_HTTP_NET_CONN_HANDLE connHandle;

connHandle = TCPIP_HTTP_NET_ConnectionHandleGet(0);
```

Parameters

Parameters	Description
connIx	the HTTP connection ix.

Function

TCPIP_HTTP_NET_CONN_HANDLE TCPIP_HTTP_NET_ConnectionHandleGet(int connIx);

TCPIP_HTTP_NET_ConnectionIndexGet Function

Gets the index of the HTTP connection.

File

http_net.h

C

```
int TCPIP_HTTP_NET_ConnectionIndexGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);
```

Returns

The connection index.

Description

This function will return the index of the requested HTTP connection.

Remarks

None

Preconditions

None.

Example

```
int connIx;

connIx = TCPIP_HTTP_NET_ConnectionIndexGet(connHandle);
```

Parameters

Parameters	Description
connHandle	the HTTP connection handle.

Function

int TCPIP_HTTP_NET_ConnectionIndexGet(TCPIP_HTTP_NET_CONN_HANDLE connHandle);

b) Data Types and Constants

TCPIP_HTTP_DYN_ARG_DCPT Structure

HTTP dynamic argument descriptor.

File

http_net.h

C

```
typedef struct {
  uint16_t argType;
  uint16_t argFlags;
  union {
    int32_t argInt32;
    const char* argStr;
  }
} TCPIP_HTTP_DYN_ARG_DCPT;
```

Members

Members	Description
uint16_t argType;	a TCPIP_HTTP_DYN_ARG_TYPE value
uint16_t argFlags;	extra argument flags
int32_t argInt32;	use this member when the arg type is INT32
const char* argStr;	use this member when the arg type is string

Description

Data structure: TCPIP_HTTP_DYN_ARG_DCPT

This data type defines the structure of a HTTP dynamic variable argument. It is used for describing the dynamic variables arguments.

Remarks

The argument flags are currently not used. They are meant for further extensions.

TCPIP_HTTP_DYN_ARG_TYPE Enumeration

HTTP supported dynamic variables argument types.

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_DYN_ARG_TYPE_INVALID = 0,
  TCPIP_HTTP_DYN_ARG_TYPE_INT32,
  TCPIP_HTTP_DYN_ARG_TYPE_STRING
} TCPIP_HTTP_DYN_ARG_TYPE;
```

Members

Members	Description
TCPIP_HTTP_DYN_ARG_TYPE_INVALID = 0	invalid argument type
TCPIP_HTTP_DYN_ARG_TYPE_INT32	the dynamic variable argument is a int32_t
TCPIP_HTTP_DYN_ARG_TYPE_STRING	the dynamic variable argument is a ASCII string

Description

Enumeration: TCPIP_HTTP_DYN_ARG_TYPE

This enumeration defines the types of the HTTP supported dynamic variables arguments.

Remarks

Currently a dynamic variable can have either string or int32_t parameters.

Only 16 bit values are currently supported.

TCPIP_HTTP_DYN_PRINT_RES Enumeration

Dynamic print result when a dynamic variable print callback function returns;

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_DYN_PRINT_RES_DONE = 0,
  TCPIP_HTTP_DYN_PRINT_RES_DEFAULT,
  TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN,
  TCPIP_HTTP_DYN_PRINT_RES_AGAIN
} TCPIP_HTTP_DYN_PRINT_RES;
```

Members

Members	Description
TCPIP_HTTP_DYN_PRINT_RES_DONE = 0	dynamic callback is done
TCPIP_HTTP_DYN_PRINT_RES_DEFAULT	no implementation supported for this dynamic variable, call the default action
TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN	process the action of this call and then call again data written to the HTTP connection but more data is pending
TCPIP_HTTP_DYN_PRINT_RES_AGAIN	do not process any action just call again no data was available to be written to the connection or the written data should be ignored this turn

Description

Enumeration: TCPIP_HTTP_DYN_PRINT_RES

This enumeration defines the results associated with the HTTP dynamic variables callbacks (TCPIP_HTTP_NET_DynPrint).

Remarks

Currently the default action for a user defined dynamic variable is "do nothing". Keywords like "inc" have an internal implementation and don't need to be processed by the user.

TCPIP_HTTP_DYN_VAR_DCPT Structure

HTTP dynamic variable descriptor.

File

http_net.h

C

```
typedef struct {
  const char* dynName;
  const char* fileName;
  uint16_t callbackID;
  uint8_t dynFlags;
  uint8_t nArgs;
  TCPIP_HTTP_DYN_ARG_DCPT* dynArgs;
  const void* dynContext;
} TCPIP_HTTP_DYN_VAR_DCPT;
```

Members

Members	Description
const char* dynName;	ASCII string storing the dynamic variable name
const char* fileName;	ASCII string storing the file name the dynamic variable belongs to
uint16_t callbackID;	Call back ID: the dynamic variable index within the file
uint8_t dynFlags;	a TCPIP_HTTP_DYN_VAR_FLAGS value
uint8_t nArgs;	number of arguments that the variable has
TCPIP_HTTP_DYN_ARG_DCPT* dynArgs;	array of argument descriptors carrying the dynamic variable arguments
const void* dynContext;	dynamic context of the callback. This context is used by the HTTP server and is irrelevant to the user.

Description

Data structure: TCPIP_HTTP_DYN_VAR_DCPT

This data type defines the structure of a HTTP dynamic variable descriptor. When the user registers a TCPIP_HTTP_NET_DynPrint function for dynamic variable processing, this callback will receive the dynamic variable descriptor as a parameter.

Remarks

None.

TCPIP_HTTP_DYN_VAR_FLAGS Enumeration

HTTP supported dynamic variables flags.

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_DYN_VAR_FLAG_NONE = 0,
  TCPIP_HTTP_DYN_VAR_FLAG_NAME_TRUNCATED = 0x01,
  TCPIP_HTTP_DYN_VAR_FLAG_ARG_NAME_TRUNCATED = 0x02,
  TCPIP_HTTP_DYN_VAR_FLAG_ARG_NO_TRUNCATED = 0x04
} TCPIP_HTTP_DYN_VAR_FLAGS;
```

Members

Members	Description
TCPIP_HTTP_DYN_VAR_FLAG_NONE = 0	no flag associated with this dynamic variable
TCPIP_HTTP_DYN_VAR_FLAG_NAME_TRUNCATED = 0x01	dynamic variable field exceeded available parsing space the dynamic variable name has been truncated

TCPIP_HTTP_DYN_VAR_FLAG_ARG_NAME_TRUNCATED = 0x02	dynamic variable field exceeded available parsing space the dynamic variable arguments have been truncated
TCPIP_HTTP_DYN_VAR_FLAG_ARG_NO_TRUNCATED = 0x04	dynamic variable arguments exceeded available buffering space the number of arguments of the dynamic variable has been truncated

Description

Enumeration: TCPIP_HTTP_DYN_VAR_FLAGS

This enumeration defines the flags associated with the HTTP supported dynamic variables.

Remarks

Multiple flags con be set simultaneously. New flags will be eventually added. Only 16-bit values are currently supported.

TCPIP_HTTP_NET_CONN_HANDLE Type

File

http_net.h

C

```
typedef const void* TCPIP_HTTP_NET_CONN_HANDLE;
```

Description

HTTP connection identifier, handle of a HTTP connection

TCPIP_HTTP_NET_EVENT_TYPE Enumeration

HTTP reported run-time events.

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_NET_EVENT_NONE = 0,
  TCPIP_HTTP_NET_EVENT_FS_UPLOAD_COMPLETE,
  TCPIP_HTTP_NET_EVENT_FILE_OPEN_ERROR = -1,
  TCPIP_HTTP_NET_EVENT_FILE_NAME_ERROR = -2,
  TCPIP_HTTP_NET_EVENT_FILE_NAME_SIZE_ERROR = -3,
  TCPIP_HTTP_NET_EVENT_FILE_SIZE_ERROR = -4,
  TCPIP_HTTP_NET_EVENT_FILE_READ_ERROR = -5,
  TCPIP_HTTP_NET_EVENT_FILE_PARSE_ERROR = -6,
  TCPIP_HTTP_NET_EVENT_DEPTH_ERROR = -7,
  TCPIP_HTTP_NET_EVENT_DYNVAR_PARSE_ERROR = -8,
  TCPIP_HTTP_NET_EVENT_FS_WRITE_ERROR = -9,
  TCPIP_HTTP_NET_EVENT_FS_MOUNT_ERROR = -10,
  TCPIP_HTTP_NET_EVENT_CHUNK_POOL_ERROR = -11,
  TCPIP_HTTP_NET_EVENT_FILE_BUFFER_POOL_ERROR = -12,
  TCPIP_HTTP_NET_EVENT_DYNVAR_ALLOC_ERROR = -13,
  TCPIP_HTTP_NET_EVENT_UPLOAD_ALLOC_ERROR = -14,
  TCPIP_HTTP_NET_EVENT_SSI_PARSE_ERROR = -15,
  TCPIP_HTTP_NET_EVENT_SSI_COMMAND_ERROR = -16,
  TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_ERROR = -17,
  TCPIP_HTTP_NET_EVENT_SSI_ALLOC_DESCRIPTOR_ERROR = -18,
  TCPIP_HTTP_NET_EVENT_DYNVAR_ARG_NAME_TRUNCATED = 0x8001,
  TCPIP_HTTP_NET_EVENT_DYNVAR_ARG_NUMBER_TRUNCATED = 0x8002,
  TCPIP_HTTP_NET_EVENT_DYNVAR_RETRIES_EXCEEDED = 0x8003,
  TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_NUMBER_TRUNCATED = 0x8004,
  TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_UNKNOWN = 0x8005,
  TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_NUMBER_MISMATCH = 0x8006,
  TCPIP_HTTP_NET_EVENT_SSI_VAR_NUMBER_EXCEEDED = 0x8007,
  TCPIP_HTTP_NET_EVENT_SSI_VAR_UNKNOWN = 0x8008,
  TCPIP_HTTP_NET_EVENT_SSI_VAR_VOID = 0x8009,
  TCPIP_HTTP_NET_EVENT_SSI_HASH_CREATE_FAILED = 0x800a,
  TCPIP_HTTP_NET_EVENT_SSI_VAR_DELETED = 0x800b,
  TCPIP_HTTP_NET_EVENT_CHUNK_POOL_EMPTY = 0x8020,
  TCPIP_HTTP_NET_EVENT_FILE_BUFFER_POOL_EMPTY = 0x8021,
  TCPIP_HTTP_NET_EVENT_PEEK_BUFFER_SIZE_EXCEEDED = 0x8030
} TCPIP_HTTP_NET_EVENT_TYPE;
```

Members

Members	Description
TCPIP_HTTP_NET_EVENT_NONE = 0	no event
TCPIP_HTTP_NET_EVENT_FS_UPLOAD_COMPLETE	Notification that a FS upload operation was completed successfully
TCPIP_HTTP_NET_EVENT_FILE_OPEN_ERROR = -1	errors an error occurred when opening a HTTP file
TCPIP_HTTP_NET_EVENT_FILE_NAME_ERROR = -2	a file name was not specified
TCPIP_HTTP_NET_EVENT_FILE_NAME_SIZE_ERROR = -3	a file name was longer than the HTTP storage space
TCPIP_HTTP_NET_EVENT_FILE_SIZE_ERROR = -4	file size error
TCPIP_HTTP_NET_EVENT_FILE_READ_ERROR = -5	file read error
TCPIP_HTTP_NET_EVENT_FILE_PARSE_ERROR = -6	file parse error: line too long
TCPIP_HTTP_NET_EVENT_DEPTH_ERROR = -7	the depth allowed for a recursive call was exceeded
TCPIP_HTTP_NET_EVENT_DYNVAR_PARSE_ERROR = -8	an error occurred when parsing: dynamic variable name terminator could not be found, for example
TCPIP_HTTP_NET_EVENT_FS_WRITE_ERROR = -9	a write error was reported while performing an FS upload
TCPIP_HTTP_NET_EVENT_FS_MOUNT_ERROR = -10	a write error was reported while mounting after an FS upload
TCPIP_HTTP_NET_EVENT_CHUNK_POOL_ERROR = -11	the number of retries for getting a chunk from the pool has been exceeded
TCPIP_HTTP_NET_EVENT_FILE_BUFFER_POOL_ERROR = -12	the number of retries for getting a file buffer has been exceeded
TCPIP_HTTP_NET_EVENT_DYNVAR_ALLOC_ERROR = -13	out of memory when trying to allocate space for a dynamic variable descriptor Note that for the dynamic variable descriptor allocation the system heap is used: • the TCPIP_STACK_MALLOC_FUNC/TCPIP_STACK_FREE_FUNC that's passed at the stack initialization
TCPIP_HTTP_NET_EVENT_UPLOAD_ALLOC_ERROR = -14	out of memory when trying to allocate space for a file upload Note that for the file upload buffer allocation the system heap is used: • the TCPIP_STACK_MALLOC_FUNC/TCPIP_STACK_FREE_FUNC that's passed at the stack initialization
TCPIP_HTTP_NET_EVENT_SSI_PARSE_ERROR = -15	an error occurred when parsing an SSI command: SSI terminator could not be found, for example
TCPIP_HTTP_NET_EVENT_SSI_COMMAND_ERROR = -16	an unknown/unsupported SSI command
TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_ERROR = -17	a SSI attribute error : command w/o attribute, etc.
TCPIP_HTTP_NET_EVENT_SSI_ALLOC_DESCRIPTOR_ERROR = -18	out of memory when trying to allocate space for a SSI command descriptor Note that for the SSI commands the system heap is used: • the TCPIP_STACK_MALLOC_FUNC/TCPIP_STACK_FREE_FUNC that's passed at the stack initialization
TCPIP_HTTP_NET_EVENT_DYNVAR_ARG_NAME_TRUNCATED = 0x8001	warning: a dynamic variable argument name too long, truncated
TCPIP_HTTP_NET_EVENT_DYNVAR_ARG_NUMBER_TRUNCATED = 0x8002	warning: too many arguments for a dynamic variable, truncated
TCPIP_HTTP_NET_EVENT_DYNVAR_RETRIES_EXCEEDED = 0x8003	warning: too many retries for a dynamic variable, stopped
TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_NUMBER_TRUNCATED = 0x8004	warning: too many attributes for a SSI command, truncated
TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_UNKNOWN = 0x8005	warning: unrecognized/unsupported SSI command attribute
TCPIP_HTTP_NET_EVENT_SSI_ATTRIB_NUMBER_MISMATCH = 0x8006	warning: wrong number of SSI command attributes
TCPIP_HTTP_NET_EVENT_SSI_VAR_NUMBER_EXCEEDED = 0x8007	warning: number of SSI set variables exceeded
TCPIP_HTTP_NET_EVENT_SSI_VAR_UNKNOWN = 0x8008	warning: SSI variable does not exist
TCPIP_HTTP_NET_EVENT_SSI_VAR_VOID = 0x8009	warning: SSI variable is void: not echoed
TCPIP_HTTP_NET_EVENT_SSI_HASH_CREATE_FAILED = 0x800a	warning: SSI variable hash could not be created, allocation failed Ther ewill be no run time SSI variable support
TCPIP_HTTP_NET_EVENT_SSI_VAR_DELETED = 0x800b	event: SSI variable deleted
TCPIP_HTTP_NET_EVENT_CHUNK_POOL_EMPTY = 0x8020	warning: allocation from the HTTP chunk pool failed (the allocation will be retried)

TCPIP_HTTP_NET_EVENT_FILE_BUFFER_POOL_EMPTY = 0x8021	warning: allocation from the HTTP file buffers pool failed (the allocation will be retried)
TCPIP_HTTP_NET_EVENT_PEEK_BUFFER_SIZE_EXCEEDED = 0x8030	warning: the HTTP peek buffer is too small and cannot contain all the data available in the transport socket buffer and a HTTP search operation may fail This will happen only for transport sockets that do not support peek operation with an offset parameter

Description

Enumeration: TCPIP_HTTP_NET_EVENT_TYPE

This enumeration defines the types of the HTTP reported events when processing dynamic variables, files, etc.

Remarks

Multiple warnings can be set simultaneously.

TCPIP_HTTP_NET_IO_RESULT Enumeration

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_NET_IO_RES_DONE = 0u,
  TCPIP_HTTP_NET_IO_RES_NEED_DATA,
  TCPIP_HTTP_NET_IO_RES_WAITING,
  TCPIP_HTTP_NET_IO_RES_ERROR
} TCPIP_HTTP_NET_IO_RESULT;
```

Members

Members	Description
TCPIP_HTTP_NET_IO_RES_DONE = 0u	Finished with procedure
TCPIP_HTTP_NET_IO_RES_NEED_DATA	More data needed to continue, call again later
TCPIP_HTTP_NET_IO_RES_WAITING	Waiting for asynchronous process to complete, call again later
TCPIP_HTTP_NET_IO_RES_ERROR	Some error has occurred, operation will be aborted

Description

Result states for execution callbacks

TCPIP_HTTP_NET_MODULE_CONFIG Structure

File

http_net.h

C

```
typedef struct {
  uint16_t nConnections;
  uint16_t dataLen;
  uint16_t sktTxBuffSize;
  uint16_t sktRxBuffSize;
  uint16_t listenPort;
  uint16_t nDescriptors;
  uint16_t nChunks;
  uint16_t maxRecurseLevel;
  uint16_t configFlags;
  uint16_t nFileBuffers;
  uint16_t fileBufferSize;
  uint16_t chunkPoolRetries;
  uint16_t fileBufferRetries;
  uint16_t dynVarRetries;
} TCPIP_HTTP_NET_MODULE_CONFIG;
```

Members

Members	Description
uint16_t nConnections;	number of simultaneous HTTP connections allowed

uint16_t dataLen;	size of the data buffer for reading cookie and GET/POST arguments (bytes)
uint16_t sktTxBuffSize;	size of TX buffer for the associated socket; leave 0 for default
uint16_t sktRxBuffSize;	size of RX buffer for the associated socket; leave 0 for default
uint16_t listenPort;	HTTP listening port: 80, 443, etc.
uint16_t nDescriptors;	how many buffers descriptors for dynamic variable processing to create they are independent of the HTTP connection number all the HTTP connections use from the dynamic descriptors pool
uint16_t nChunks;	maximum number of chunks that are created It depends on the TCPIP_HTTP_NET_MAX_RECURSE_LEVEL and on the number of connections Maximum number should be TCPIP_HTTP_NET_MAX_CONNECTIONS * TCPIP_HTTP_NET_MAX_RECURSE_LEVEL All the chunks are in a pool and are used by all connections
uint16_t maxRecurseLevel;	The maximum depth of recursive calls for serving a web page: • files without dynvars: 1 • file including another file: + 1 • file including a dynamic variable: + 1 etc.
uint16_t configFlags;	a TCPIP_HTTP_NET_MODULE_FLAGS value.
uint16_t nFileBuffers;	number of file buffers to be created; These buffers are used to store data while file processing is done They are organized in a pool Each file being processed needs a file buffer and tries to get it from the pool If a buffer is not available, the HTTP conenction will wait for one to become available. Once the file is done the file buffer is released and could be used by a different file The number depends on the number of files that are processed in parallel To avoid deadlock the number should be >= than the number of maximum files that can be open simultaneously: i.e. for file1 ->include file2 -> include file3 you'll need >= 3 file process buffers
uint16_t fileBufferSize;	size of each of these file buffers should correspond to TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE
uint16_t chunkPoolRetries;	how many retries to get chunk from the pool before giving up
uint16_t fileBufferRetries;	how many retries to get a fileBuffer before giving up
uint16_t dynVarRetries;	how many retries to have for a dynamic variable dynamicPrint function before calling it done

Description

HTTP module dynamic configuration data

TCPIP_HTTP_NET_MODULE_FLAGS Enumeration

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_NET_MODULE_FLAG_DEFAULT = 0x00,
  TCPIP_HTTP_NET_MODULE_FLAG_NON_PERSISTENT = 0x01,
  TCPIP_HTTP_NET_MODULE_FLAG_NO_DELAY = 0x02,
  TCPIP_HTTP_NET_MODULE_FLAG_SECURE_ON = 0x10,
  TCPIP_HTTP_NET_MODULE_FLAG_SECURE_OFF = 0x20,
  TCPIP_HTTP_NET_MODULE_FLAG_SECURE_DEFAULT = 0x00
} TCPIP_HTTP_NET_MODULE_FLAGS;
```

Members

Members	Description
TCPIP_HTTP_NET_MODULE_FLAG_DEFAULT = 0x00	Default flags value
TCPIP_HTTP_NET_MODULE_FLAG_NON_PERSISTENT = 0x01	Use non-persistent connections This flag will cause the HTTP connections to be non-persistent and closed after serving each request to the client By default the HTTP connections are persistent
TCPIP_HTTP_NET_MODULE_FLAG_NO_DELAY = 0x02	Create the HTTP sockets with NO-DELAY option. It will flush data as soon as possible.
TCPIP_HTTP_NET_MODULE_FLAG_SECURE_ON = 0x10	All HTTP connections have to be secure (supposing the network presentation layer supports encryption) Cannot be used together with TCPIP_HTTP_NET_MODULE_FLAG_SECURE_OFF
TCPIP_HTTP_NET_MODULE_FLAG_SECURE_OFF = 0x20	HTTP connections will be non-secure Cannot be used together with TCPIP_HTTP_NET_MODULE_FLAG_SECURE_ON

TCPIP_HTTP_NET_MODULE_FLAG_SECURE_DEFAULT = 0x00	HTTP security is based on the port numbers

Description

HTTP module configuration flags Multiple flags can be OR-ed

TCPIP_HTTP_NET_READ_STATUS Enumeration

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_NET_READ_OK = 0,
  TCPIP_HTTP_NET_READ_TRUNCATED,
  TCPIP_HTTP_NET_READ_INCOMPLETE
} TCPIP_HTTP_NET_READ_STATUS;
```

Members

Members	Description
TCPIP_HTTP_NET_READ_OK = 0	Read was successful
TCPIP_HTTP_NET_READ_TRUNCATED	Buffer overflow prevented by truncating value
TCPIP_HTTP_NET_READ_INCOMPLETE	Entire object is not yet in the buffer. Try again later.

Description

Result states for TCPIP_HTTP_NET_ConnectionPostNameRead, TCPIP_HTTP_NET_ConnectionPostValueRead and TCPIP_HTTP_NET_ConnectionPostReadPair

TCPIP_HTTP_NET_STATUS Enumeration

File

http_net.h

C

```
typedef enum {
  TCPIP_HTTP_NET_STAT_GET = 0u,
  TCPIP_HTTP_NET_STAT_POST,
  TCPIP_HTTP_NET_STAT_BAD_REQUEST,
  TCPIP_HTTP_NET_STAT_UNAUTHORIZED,
  TCPIP_HTTP_NET_STAT_NOT_FOUND,
  TCPIP_HTTP_NET_STAT_OVERFLOW,
  TCPIP_HTTP_NET_STAT_INTERNAL_SERVER_ERROR,
  TCPIP_HTTP_NET_STAT_NOT_IMPLEMENTED,
  TCPIP_HTTP_NET_STAT_REDIRECT,
  TCPIP_HTTP_NET_STAT_TLS_REQUIRED,
  TCPIP_HTTP_NET_STAT_UPLOAD_FORM,
  TCPIP_HTTP_NET_STAT_UPLOAD_STARTED,
  TCPIP_HTTP_NET_STAT_UPLOAD_WRITE,
  TCPIP_HTTP_NET_STAT_UPLOAD_WRITE_WAIT,
  TCPIP_HTTP_NET_STAT_UPLOAD_OK,
  TCPIP_HTTP_NET_STAT_UPLOAD_ERROR
} TCPIP_HTTP_NET_STATUS;
```

Members

Members	Description
TCPIP_HTTP_NET_STAT_GET = 0u	GET command is being processed
TCPIP_HTTP_NET_STAT_POST	POST command is being processed
TCPIP_HTTP_NET_STAT_BAD_REQUEST	400 Bad Request will be returned
TCPIP_HTTP_NET_STAT_UNAUTHORIZED	401 Unauthorized will be returned
TCPIP_HTTP_NET_STAT_NOT_FOUND	404 Not Found will be returned
TCPIP_HTTP_NET_STAT_OVERFLOW	414 Request-URI Too Long will be returned
TCPIP_HTTP_NET_STAT_INTERNAL_SERVER_ERROR	500 Internal Server Error will be returned

TCPIP_HTTP_NET_STAT_NOT_IMPLEMENTED	501 Not Implemented (not a GET or POST command)
TCPIP_HTTP_NET_STAT_REDIRECT	302 Redirect will be returned
TCPIP_HTTP_NET_STAT_TLS_REQUIRED	403 Forbidden is returned, indicating TLS is required
TCPIP_HTTP_NET_STAT_UPLOAD_FORM	Show the Upload form
TCPIP_HTTP_NET_STAT_UPLOAD_STARTED	An upload operation is being processed
TCPIP_HTTP_NET_STAT_UPLOAD_WRITE	An upload operation is currently writing
TCPIP_HTTP_NET_STAT_UPLOAD_WRITE_WAIT	An upload operation is currently waiting for the write completion
TCPIP_HTTP_NET_STAT_UPLOAD_OK	An Upload was successful
TCPIP_HTTP_NET_STAT_UPLOAD_ERROR	An Upload was not a valid image

Description

Supported Commands and Server Response Codes

TCPIP_HTTP_NET_USER_CALLBACK Structure

HTTP user implemented callback data structure.

File

http_net.h

C

```
typedef struct _tag_TCPIP_HTTP_NET_USER_CALLBACK {
  TCPIP_HTTP_NET_IO_RESULT (* getExecute)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const struct
_tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  TCPIP_HTTP_NET_IO_RESULT (* postExecute)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const struct
_tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  uint8_t (* fileAuthenticate)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* cFile, const struct
_tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  uint8_t (* userAuthenticate)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* cUser, const char* cPass,
const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  TCPIP_HTTP_DYN_PRINT_RES (* dynamicPrint)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const
TCPIP_HTTP_DYN_VAR_DCPT* varDcpt, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  void (* dynamicAck)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const void* buffer, const struct
_tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  void (* eventReport)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_NET_EVENT_TYPE evType, const void*
evInfo, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
  bool (* ssiNotify)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_SSI_NOTIFY_DCPT* pSSINotifyDcpt,
const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);
} TCPIP_HTTP_NET_USER_CALLBACK;
```

Members

Members	Description
TCPIP_HTTP_NET_IO_RESULT (* getExecute)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_ConnectionGetExecute GET process function
TCPIP_HTTP_NET_IO_RESULT (* postExecute)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_ConnectionPostExecute POST process function
uint8_t (* fileAuthenticate)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* cFile, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_ConnectionFileAuthenticate File Authenticate function
uint8_t (* userAuthenticate)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const char* cUser, const char* cPass, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_ConnectionUserAuthenticate User Authenticate function

TCPIP_HTTP_DYN_PRINT_RES (* dynamicPrint)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const TCPIP_HTTP_DYN_VAR_DCPT* varDcpt, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_DynPrint Dynamic variable process function
void (* dynamicAck)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, const void* buffer, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_DynAcknowledge Dynamic variable acknowledge function
void (* eventReport)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_NET_EVENT_TYPE evType, const void* evInfo, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_EventReport run-time HTTP processing report
bool (* ssiNotify)(TCPIP_HTTP_NET_CONN_HANDLE connHandle, TCPIP_HTTP_SSI_NOTIFY_DCPT* pSSINotifyDcpt, const struct _tag_TCPIP_HTTP_NET_USER_CALLBACK* pCBack);	TCPIP_HTTP_NET_SSINotification run-time HTTP SSI processing

Description

Structure: TCPIP_HTTP_NET_USER_CALLBACK

This data structure defines the user callbacks that are implemented by the user and the HTTP server calls at run-time.

Remarks

See the detailed explanation for each callback in the callback templates section.

The extra pCBack parameter is passed back for allowing the user to store additional info in the supplied TCPIP_HTTP_NET_USER_CALLBACK data structure that could be used at run-time.

TCPIP_HTTP_NET_USER_HANDLE Type

HTTP user handle.

File

http_net.h

C

```
typedef const void* TCPIP_HTTP_NET_USER_HANDLE;
```

Description

Type: TCPIP_HTTP_NET_USER_HANDLE

A handle that a client can use after the HTTP user callback has been registered.

TCPIP_HTTP_NET_ConnectionPostReadPair Macro

Reads a name and value pair from a URL encoded string in the network transport buffer.

File

http_net.h

C

```
#define TCPIP_HTTP_NET_ConnectionPostReadPair(connHandle, cData, wLen) \
        TCPIP_HTTP_NET_ConnectionPostValueRead(connHandle, cData, wLen)
```

Returns

- TCPIP_HTTP_NET_READ_OK - name and value were successfully read
- TCPIP_HTTP_NET_READ_TRUNCTATED - entire name and value could not fit in the buffer, so input was truncated and data has been lost
- TCPIP_HTTP_NET_READ_INCOMPLETE - entire name and value was not yet in the buffer, so call this function again later to retrieve

Description

Reads a name and value pair from a URL encoded string in the network transport buffer. This function is meant to be called from an TCPIP_HTTP_NET_ConnectionPostExecute callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least 2 extra bytes are needed in cData over the maximum length of data expected to be read.

This function will read until the next '&' character, which indicates the end of a value parameter. It assumes that the front of the buffer is the

beginning of the name parameter to be read.

This function properly updates the connection byteCount (see TCPIP_HTTP_NET_ConnectionByteCountGet) by decrementing it by the number of bytes read. It also removes the delimiting '&' from the buffer.

Once complete, two strings will exist in the cData buffer. The first is the parameter name that was read, while the second is the associated value.

Remarks

This function is aliased to TCPIP_HTTP_NET_ConnectionPostValueRead, since they effectively perform the same task. The name is provided only for completeness.

Preconditions

The front of the network transport buffer is the beginning of a name parameter, and the rest of the network transport buffer contains a URL-encoded string with a name parameter terminated by a '=' character and a value parameter terminated by a '&'.

Parameters

Parameters	Description
connHandle	HTTP connection handle
cData	where to store the name and value strings once they are read
wLen	how many bytes can be written to cData

Function

TCPIP_HTTP_NET_READ_STATUS TCPIP_HTTP_NET_ConnectionPostReadPair(TCPIP_HTTP_NET_CONN_HANDLE connHandle, uint8_t* cData, uint16_t wLen)

TCPIP_HTTP_SSI_ATTR_DCPT Structure

HTTP SSI attribute descriptor.

File

http_net.h

C

```
typedef struct {
  const char* attribute;
  char* value;
} TCPIP_HTTP_SSI_ATTR_DCPT;
```

Members

Members	Description
const char* attribute;	the SSI attribute of the command
char* value;	the SSI value

Description

Data structure: TCPIP_HTTP_SSI_ATTR_DCPT

This data type defines the structure of a SSI command attribute descriptor. When the user registers a TCPIP_HTTP_NET_SSINotification function for SSI processing, this callback will receive info about the SSI attribute descriptors.

Remarks

The SSI commands consist of pairs of (attribute, value) tokens. For example: <!--#set var="varname" value="varvalue" --> has 2 pairs: pair 1 - attribute = var value = varname pair 2 - attribute = value value = varvalue

TCPIP_HTTP_SSI_NOTIFY_DCPT Structure

HTTP SSI notification descriptor.

File

http_net.h

C

```
typedef struct {
  const char* fileName;
  char* ssiCommand;
  int nAttribs;
```

```
  TCPIP_HTTP_SSI_ATTR_DCPT* pAttrDcpt;
} TCPIP_HTTP_SSI_NOTIFY_DCPT;
```

Members

Members	Description
const char* fileName;	the file containing the SSI command
char* ssiCommand;	the SSI command parsed from the command line
int nAttribs;	number of attributes descriptors in the command
TCPIP_HTTP_SSI_ATTR_DCPT* pAttrDcpt;	pointer to an array of descriptors parsed from this SSI command

Description

Data structure: TCPIP_HTTP_SSI_NOTIFY_DCPT

This data type defines the structure of a SSI notification descriptor. When the user registers a TCPIP_HTTP_NET_SSINotification function for SSI processing, this callback will receive a SSI descriptor as a parameter.

Remarks

None.

Files

Files

Name	Description
http_net.h	The HTTP web server module together with a file system (SYS_FS) allow the board to act as a web server. This module uses the network presentation layer to allow for encrypted connections.
http_net_config.h	HTTP Net configuration file

Description

This section lists the source and header files used by the library.

http_net.h

The HTTP web server module together with a file system (SYS_FS) allow the board to act as a web server. This module uses the network presentation layer to allow for encrypted connections.

Enumerations

	Name	Description
	TCPIP_HTTP_DYN_ARG_TYPE	HTTP supported dynamic variables argument types.
	TCPIP_HTTP_DYN_PRINT_RES	Dynamic print result when a dynamic variable print callback function returns;
	TCPIP_HTTP_DYN_VAR_FLAGS	HTTP supported dynamic variables flags.
	TCPIP_HTTP_NET_EVENT_TYPE	HTTP reported run-time events.
	TCPIP_HTTP_NET_IO_RESULT	Result states for execution callbacks
	TCPIP_HTTP_NET_MODULE_FLAGS	HTTP module configuration flags Multiple flags can be OR-ed
	TCPIP_HTTP_NET_READ_STATUS	Result states for TCPIP_HTTP_NET_ConnectionPostNameRead, TCPIP_HTTP_NET_ConnectionPostValueRead and TCPIP_HTTP_NET_ConnectionPostReadPair
	TCPIP_HTTP_NET_STATUS	Supported Commands and Server Response Codes

Functions

	Name	Description
	TCPIP_HTTP_NET_ActiveConnectionCountGet	Gets the number of active (and inactive) connections.
	TCPIP_HTTP_NET_ArgGet	Locates a form field value in a given data array.
	TCPIP_HTTP_NET_ConnectionByteCountDec	Decrements the connection byte count.
	TCPIP_HTTP_NET_ConnectionByteCountGet	Returns how many bytes have been read so far.
	TCPIP_HTTP_NET_ConnectionByteCountSet	Sets how many bytes have been read so far.
	TCPIP_HTTP_NET_ConnectionCallbackPosGet	Returns the callback position indicator.
	TCPIP_HTTP_NET_ConnectionCallbackPosSet	Sets the callback position indicator.
	TCPIP_HTTP_NET_ConnectionDataBufferGet	Returns pointer to connection general purpose data buffer.

	TCPIP_HTTP_NET_ConnectionDataBufferSizeGet	Returns the size of the connection general purpose data buffer.
	TCPIP_HTTP_NET_ConnectionDiscard	Discards any pending data in the connection RX buffer.
	TCPIP_HTTP_NET_ConnectionDynamicDescriptors	Returns the number of dynamic variable descriptors
	TCPIP_HTTP_NET_ConnectionFileAuthenticate	Determines if a given file name requires authentication
	TCPIP_HTTP_NET_ConnectionFileGet	Get handle to current connection's file.
	TCPIP_HTTP_NET_ConnectionFileInclude	Writes a file to the HTTP connection.
	TCPIP_HTTP_NET_ConnectionFlush	Immediately transmits all connection pending TX data.
	TCPIP_HTTP_NET_ConnectionGetExecute	Processes GET form field variables and cookies.
	TCPIP_HTTP_NET_ConnectionHandleGet	Gets the connection handle of a HTTP connection.
	TCPIP_HTTP_NET_ConnectionHasArgsGet	Checks whether there are get or cookie arguments.
	TCPIP_HTTP_NET_ConnectionHasArgsSet	Sets whether there are get or cookie arguments.
	TCPIP_HTTP_NET_ConnectionIndexGet	Gets the index of the HTTP connection.
	TCPIP_HTTP_NET_ConnectionIsAuthorizedGet	Gets the authorized state for the current connection.
	TCPIP_HTTP_NET_ConnectionIsAuthorizedSet	Sets the authorized state for the current connection.
	TCPIP_HTTP_NET_ConnectionNetHandle	Returns the network handle of the current connection.
	TCPIP_HTTP_NET_ConnectionPeek	Reads a specified number of data bytes from the connection RX buffer without removing them from the buffer.
	TCPIP_HTTP_NET_ConnectionPostExecute	Processes POST form variables and data.
	TCPIP_HTTP_NET_ConnectionPostNameRead	Reads a name from a URL encoded string in the network transport buffer.
	TCPIP_HTTP_NET_ConnectionPostSmGet	Get the POST state machine state.
	TCPIP_HTTP_NET_ConnectionPostSmSet	Set the POST state machine state.
	TCPIP_HTTP_NET_ConnectionPostValueRead	Reads a value from a URL encoded string in the network transport buffer.
	TCPIP_HTTP_NET_ConnectionRead	Reads an array of data bytes from a connection's RX buffer.
	TCPIP_HTTP_NET_ConnectionReadBufferSize	Returns the size of the connection RX buffer.
	TCPIP_HTTP_NET_ConnectionReadIsReady	Determines how many bytes can be read from the connection RX buffer.
	TCPIP_HTTP_NET_ConnectionSocketGet	Get the socket for the current connection.
	TCPIP_HTTP_NET_ConnectionStatusGet	Gets HTTP status.
	TCPIP_HTTP_NET_ConnectionStatusSet	Sets HTTP status.
	TCPIP_HTTP_NET_ConnectionStringFind	Helper to find a string of characters in an incoming connection buffer.
	TCPIP_HTTP_NET_ConnectionUserAuthenticate	Performs validation on a specific user name and password.
	TCPIP_HTTP_NET_ConnectionUserDataGet	Gets the user data parameter for the current connection.
	TCPIP_HTTP_NET_ConnectionUserDataSet	Sets the user data parameter for the current connection.
	TCPIP_HTTP_NET_DynAcknowledge	Dynamic variable acknowledge function
	TCPIP_HTTP_NET_DynamicWrite	Writes a data buffer to the current connection
	TCPIP_HTTP_NET_DynamicWriteString	Helper for writing a string within a dynamic variable context.
	TCPIP_HTTP_NET_DynPrint	Inserts dynamic content into a web page
	TCPIP_HTTP_NET_EventReport	Reports an event that occurred in the processing of a HTTP web page
	TCPIP_HTTP_NET_SSINotification	Reports an SSI processing event that occurs in the processing of a HTTP web page
	TCPIP_HTTP_NET_SSIVariableDelete	Function to delete an SSI variable.
	TCPIP_HTTP_NET_SSIVariableGet	Function to get access to an existing SSI variable.
	TCPIP_HTTP_NET_SSIVariableGetByIndex	Function to get access to an existing SSI variable.
	TCPIP_HTTP_NET_SSIVariableSet	Function to set an SSI variable.
	TCPIP_HTTP_NET_SSIVariablesNumberGet	Function to get the number of the current SSI variables.
	TCPIP_HTTP_NET_Task	Standard TCP/IP stack module task function.
	TCPIP_HTTP_NET_URLDecode	Parses a string from URL encoding to plain text.
	TCPIP_HTTP_NET_UserHandlerDeregister	Deregisters a previously registered HTTP user handler.
	TCPIP_HTTP_NET_UserHandlerRegister	Registers a user callback structure with the HTTP server

Macros

	Name	Description
	TCPIP_HTTP_NET_ConnectionPostReadPair	Reads a name and value pair from a URL encoded string in the network transport buffer.

Structures

	Name	Description
	_tag_TCPIP_HTTP_NET_USER_CALLBACK	HTTP user implemented callback data structure.
	TCPIP_HTTP_DYN_ARG_DCPT	HTTP dynamic argument descriptor.
	TCPIP_HTTP_DYN_VAR_DCPT	HTTP dynamic variable descriptor.
	TCPIP_HTTP_NET_MODULE_CONFIG	HTTP module dynamic configuration data
	TCPIP_HTTP_NET_USER_CALLBACK	HTTP user implemented callback data structure.
	TCPIP_HTTP_SSI_ATTR_DCPT	HTTP SSI attribute descriptor.
	TCPIP_HTTP_SSI_NOTIFY_DCPT	HTTP SSI notification descriptor.

Types

	Name	Description
	TCPIP_HTTP_NET_CONN_HANDLE	HTTP connection identifier, handle of a HTTP connection
	TCPIP_HTTP_NET_USER_HANDLE	HTTP user handle.

Description

HTTP Headers for Microchip TCP/IP Stack

The HTTP module runs a web server within the TCP/IP stack. This facilitates an easy method to view status information and control applications using any standard web browser.

File Name

http_net.h

Company

Microchip Technology Inc.

http_net_config.h

HTTP Net configuration file

Macros

	Name	Description
	TCPIP_HTTP_NET_CACHE_LEN	Max lifetime (sec) of static responses as string
	TCPIP_HTTP_NET_CHUNK_RETRIES	retry limit for allocating a chunk from the pool If more retries are not successful the operation will be aborted
	TCPIP_HTTP_NET_CHUNKS_NUMBER	number of chunks that are created It depends on the TCPIP_HTTP_NET_MAX_RECURSE_LEVEL and on the number of connections Maximum number should be TCPIP_HTTP_NET_MAX_CONNECTIONS * TCPIP_HTTP_NET_MAX_RECURSE_LEVEL i.e. TCPIP_HTTP_NET_MODULE_CONFIG::nConnections * TCPIP_HTTP_NET_MODULE_CONFIG::nChunks All the chunks are in a pool and are used by all connections
	TCPIP_HTTP_NET_CONFIG_FLAGS	Define the HTTP module configuration flags Use 0 for default See HTTP_MODULE_FLAGS definition for possible values
	TCPIP_HTTP_NET_COOKIE_BUFFER_SIZE	Size of the buffer used for sending the cookies to the client. Should be able to accommodate the longest cookie response. Otherwise the cookies will be truncated.
	TCPIP_HTTP_NET_DEFAULT_FILE	Indicate what HTTP file to serve when no specific one is requested
	TCPIP_HTTP_NET_DYNVAR_ARG_MAX_NUMBER	maximum number of arguments for a dynamic variable
	TCPIP_HTTP_NET_DYNVAR_DESCRIPTORS_NUMBER	how many buffers descriptors for dynamic variable processing they are independent of the HTTP connection number all the HTTP connections use from the dynamic descriptors pool
	TCPIP_HTTP_NET_DYNVAR_MAX_LEN	maximum size for a complete dynamic variable: name + args must be <= TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE! If it is much larger than needed then inefficiency occurs when reading data from the file and then discarding it because a much larger than needed data buffer was read

	TCPIP_HTTP_NET_DYNVAR_PROCESS	This symbol enables the processing of dynamic variables Make it evaluate to false (0) if dynamic variables are not needed All the following symbols referring to dynamic variables are relevant only when TCPIP_HTTP_NET_DYNVAR_PROCESS != 0
	TCPIP_HTTP_NET_DYNVAR_PROCESS_RETRIES	retry limit for a dynamic variable processing ths puts a limit on the number of times a dynamic variable "dynamicPrint" function can return TCPIP_HTTP_DYN_PRINT_RES_AGAIN/TCPIP_HTTP_DYN_PRINT_RES_PROCESS_AGAIN and avoids having the HTTP code locked up forever. If more retries are attempted the processing will be considered done and dynamicPrint function will not be called again
	TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_RETRIES	Retry limit for allocating a file buffer from the pool. If more retries are not successful the operation will be aborted.
	TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE	Size of the buffer used for processing HTML, dynamic variable and binary files. For dynamic variable files it should be able to accommodate the longest HTML line size, including CRLF!
	TCPIP_HTTP_NET_FILE_PROCESS_BUFFERS_NUMBER	Number of file buffers to be created; These buffers are used to store data while file processing is done They are organized in a pool Each file being processed needs a file buffer and tries to get it from the pool If a buffer is not available, the HTTP conenction will wait for one to become available. Once the file is done the file buffer is released and could be used by a different file The number depends on the number of files that are processed in parallel To avoid deadlock the number should be >= than the number of... more
	TCPIP_HTTP_NET_FILE_UPLOAD_ENABLE	Configure MPFS over HTTP updating Comment this line to disable updating via HTTP
	TCPIP_HTTP_NET_FILE_UPLOAD_NAME	This is macro TCPIP_HTTP_NET_FILE_UPLOAD_NAME.
	TCPIP_HTTP_NET_FILENAME_MAX_LEN	maximum size of a HTTP file name with the path removed from the file name one extra char added for the string terminator
	TCPIP_HTTP_NET_FIND_PEEK_BUFF_SIZE	Size of the peek buffer to perform searches into. If the underlying transport layer supports offset peak operation with a offset, value could be smaller (80 characters, for example); otherwise, a one time peek is required and the buffer should be larger - 512 bytes recommended.
	TCPIP_HTTP_NET_MAX_CONNECTIONS	Maximum numbers of simultaneous supported HTTP connections.
	TCPIP_HTTP_NET_MAX_DATA_LEN	Define the maximum data length for reading cookie and GET/POST arguments (bytes)
	TCPIP_HTTP_NET_MAX_HEADER_LEN	The length of longest header string that can be parsed
	TCPIP_HTTP_NET_MAX_RECURSE_LEVEL	The maximum depth of recursive calls for serving a web page: • no dynvars files: 1 • file including a file: 2 • if the include file includes another file: +1 • if a dyn variable: +1 Default value is 3;
	TCPIP_HTTP_NET_RESPONSE_BUFFER_SIZE	Size of the buffer used for sending the response messages to the client. Should be able to accommodate the longest server response: Default setting should be 300 bytes
	TCPIP_HTTP_NET_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_HTTP_NET_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the HTTP socket Use 0 for default TCP socket value The default recommended value for high throughput is > 2MSS (3 KB). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput.
	TCPIP_HTTP_NET_SSI_ATTRIBUTES_MAX_NUMBER	maximum number of attributes for a SSI command most SSI commands take just one attribute/value pair per line but multiple attribute/value pairs on the same line are allowed where it makes sense

	TCPIP_HTTP_NET_SSI_CMD_MAX_LEN	maximum size for a SSI command line: command + attribute/value pairs must be <= TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE! If it is much larger than needed then inefficiency occurs when reading data from the file and then discarding it because a much larger than needed data buffer was read
	TCPIP_HTTP_NET_SSI_ECHO_NOT_FOUND_MESSAGE	message to echo when echoing a not found variable
	TCPIP_HTTP_NET_SSI_PROCESS	This symbol enables the processing of SSI commands Make it evaluate to false (0) if SSI commands are not needed All the following symbols referring to SSI commands are relevant only when TCPIP_HTTP_NET_SSI_PROCESS != 0
	TCPIP_HTTP_NET_SSI_STATIC_ATTTRIB_NUMBER	number of static attributes associated to a SSI command if the command has more attributes than this number the excess will be allocated dynamically
	TCPIP_HTTP_NET_SSI_VARIABLE_NAME_MAX_LENGTH	maximum length of a SSI variable name any excess characters will be truncated Note that this can result in multiple variables being represented as one SSI variable
	TCPIP_HTTP_NET_SSI_VARIABLE_STRING_MAX_LENGTH	maximum size of a SSI string variable value any excess characters will be truncated Note that the variable value requires SSI storage that's allocated dynamically Also, this value determines the size of an automatic (stack) buffer when the variable is echoed. If this value is large, make sure you have enough stack space.
	TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER	maximum number of SSI variables that can be created at run time These variables are stored in an internal hash. For max. efficiency this number should be a prime.
	TCPIP_HTTP_NET_TASK_RATE	The HTTP task rate, ms The default value is 33 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_HTTP_NET_TIMEOUT	Max time (sec) to await more data before timing out and disconnecting the socket
	TCPIP_HTTP_NET_USE_AUTHENTICATION	Enable basic authentication support
	TCPIP_HTTP_NET_USE_COOKIES	Enable cookie support
	TCPIP_HTTP_NET_USE_POST	Define which HTTP modules to use If not using a specific module, comment it to save resources Enable POST support

Description

HyperText Transfer Protocol (HTTP) Net Configuration file

This file contains the HTTP Net module configuration options

File Name

http_net_config.h

Company

Microchip Technology Inc.

ICMP Module

This section describes the TCP/IP Stack Library ICMP module.

Introduction

TCP/IP Stack Library Internet Control Message Protocol (ICMP) Module for Microchip Microcontrollers

This library provides the API of the ICMP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The Internet Control Message Protocol is used to send error and status messages and requests. The ICMP module implements the Echo Reply message type (commonly referred to as a ping) which can be used to determine if a specified host is reachable across an IP network from a device running the TCP/IP stack. An ICMP server is also supported to respond to pings from other devices.

Using the Library

This topic describes the basic architecture of the ICMP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `icmp.h`

The interface to the ICMP TCP/IP Stack library is defined in the `icmp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the ICMP TCP/IP Stack library should include `tcpip.h`.

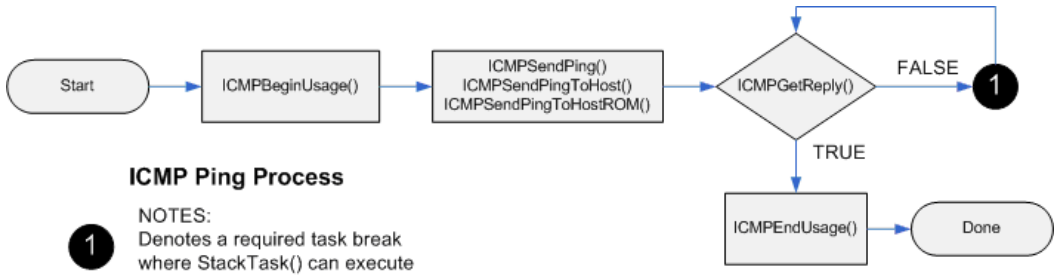
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the ICMP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

ICMP Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the ICMP module.

Library Interface Section	Description
Functions	This section provides general interface routines
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_ICMP_CLIENT_USER_NOTIFICATION	allow ICMP client user notification if enabled, the TCPIP_ICMP_CallbackRegister/TCPIP_ICMP_CallbackDeregister functions exist and can be used Note that if disabled, the tcpip console ping command won't be available
	TCPIP_ICMP_ECHO_REQUEST_TIMEOUT	Timeout for a request that was sent and no reply was received milliseconds
	TCPIP_ICMP_TASK_TICK_RATE	The ICMP task processing rate: number of milliseconds to generate an ICMP tick. This is the tick that advances the ICMP state machine. The default value is around 30 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

The configuration of the ICMP TCP/IP Stack is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the ICMP TCP/IP Stack. Based on the selections made, the ICMP TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the ICMP TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_ICMP_CLIENT_USER_NOTIFICATION Macro

File

icmp_config.h

C

```
#define TCPIP_ICMP_CLIENT_USER_NOTIFICATION true
```

Description

allow ICMP client user notification if enabled, the TCPIP_ICMP_CallbackRegister/TCPIP_ICMP_CallbackDeregister functions exist and can be used Note that if disabled, the tcpip console ping command won't be available

TCPIP_ICMP_ECHO_REQUEST_TIMEOUT Macro

File

icmp_config.h

C

```
#define TCPIP_ICMP_ECHO_REQUEST_TIMEOUT 500
```

Description

Timeout for a request that was sent and no reply was received milliseconds

TCPIP_ICMP_TASK_TICK_RATE Macro

File

icmp_config.h

C

```
#define TCPIP_ICMP_TASK_TICK_RATE 33
```

Description

The ICMP task processing rate: number of milliseconds to generate an ICMP tick. This is the tick that advances the ICMP state machine. The default value is around 30 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Building the Library

This section lists the files that are available in the ICMP module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/icmp.c`	ICMP implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The ICMP module depends on the following modules:

- TCP/IP Stack Library
- IPv4 Module

Library Interface

a) Functions

	Name	Description
	TCPIP_ICMP_CallbackDeregister	Deregisters the ICMP callback function.
	TCPIP_ICMP_CallbackRegister	Registers a callback to allow the application layer to process incoming ICMPv4 packets
	TCPIP_ICMP_EchoRequestSend	Sends an ICMP echo request to a remote node.
	TCPIP_ICMP_Task	Standard TCP/IP stack module task function.
	TCPIP_ICMP_EchoRequest	Sends an ICMP echo request containing user supplied data
	TCPIP_ICMP_EchoRequestCancel	Cancels a previously sent ICMP echo request

b) Data Types and Constants

	Name	Description
	ICMP_ECHO_RESULT	result of an ICMP Echo Request call
	ICMP_HANDLE	a handle that a client can use after the event handler has been registered
	TCPIP_ICMP_MODULE_CONFIG	Placeholder for ICMP module configuration.
	_tag_TCPIP_ICMP_ECHO_REQUEST	Contains the data needed for an ICMP request operation
	TCPIP_ICMP_ECHO_REQUEST	Contains the data needed for an ICMP request operation
	TCPIP_ICMP_ECHO_REQUEST_RESULT	A result code for an Echo Request callback (as a result of an ICMP TCPIP_ICMP_EchoRequest call)
	TCPIP_ICMP_REQUEST_HANDLE	This is type TCPIP_ICMP_REQUEST_HANDLE.

Description

This section describes the Application Programming Interface (API) functions of the ICMP module.

Refer to each section for a detailed description.

a) Functions

TCPIP_ICMP_CallbackDeregister Function

Deregisters the ICMP callback function.

File

icmp.h

C

```
bool TCPIP_ICMP_CallbackDeregister(ICMP_HANDLE hIcmp);
```

Returns

- true - if the notification handler has been successfully removed
- false - if such a notification handler could not be found

Description

This function notifies a stack client to remove its former registered notification handler. After this operation the client will no longer be notified about the receiving of replies to the ICMP requests.

Remarks

None.

Preconditions

The TCP/IP Stack must be initialized and up and running. A previous successful call to TCPIP_ICMP_CallbackRegister has been done.

Example

```
void MyICMPCallbackFunction(TCPIP_NET_HANDLE hNetIf, IPV4_ADDR * remoteIP, void * data);
ICMP_HANDLE hIcmp = TCPIP_ICMP_CallbackRegister(&MyICMPCallbackFunction);
if(hIcmp != 0)
{
    // successfully registered my handler
    // send requests and process the incoming results
    // ...
    // later on, once we're done, remove the notification handler
    TCPIP_ICMP_CallbackDeregister(hIcmp);
}
```

Parameters

Parameters	Description
hIcmp	an ICMP handle obtained by TCPIP_ICMP_CallbackRegister

Function

bool TCPIP_ICMP_CallbackDeregister(ICMP_HANDLE hIcmp)

TCPIP_ICMP_CallbackRegister Function

Registers a callback to allow the application layer to process incoming ICMPv4 packets

File

icmp.h

C

```
ICMP_HANDLE TCPIP_ICMP_CallbackRegister(void (*callback)(TCPIP_NET_HANDLE hNetIf, IPV4_ADDR * remoteIP,
void * data));
```

Returns

- A non-null handle if the registration succeeded
- 0 if the registration operation failed (for example, out of memory)

Description

Allows a stack client to be notified of the receiving of a response from an ICMP query. Once an Echo request reply is received, the notification handler callback will be called, letting the client know of the result of the query. The callback will contain as parameters:

- the network interface handle on which the query reply was received
- the remote host IP address
- a 32-bit value containing the sequence number in the low 16-bit part and the identifier value in the high 16-bit part.

Remarks

None.

Preconditions

The TCP/IP Stack must be initialized and up and running.

Example

```
// Callback function prototype
void MyICMPCallbackFunction(TCPIP_NET_HANDLE hNetIf, IPV4_ADDR * remoteIP, void * data);

// ******************************************************************************
// Register callback function
ICMP_HANDLE hIcmp = TCPIP_ICMP_CallbackRegister(&MyICMPCallbackFunction);
if(hIcmp == 0)
{
    // process error; couldn't register a handler
```

```
}

// success; I can send an Echo request and receive notification


// ******************************************************************************
IPV4_ADDR remoteIP = 0xc0a00101;
uint16_t mySequenceNumber = 1;
uint16_t myId = 0x1234;
// send an ICMP query request
TCPIP_ICMP_EchoRequestSend(&remoteIP, mySequenceNumber, myId);


// ******************************************************************************
// process the ICMP reply in the callback function
void MyICMPCallbackFunction(TCPIP_NET_HANDLE hNetIf, IPV4_ADDR * remoteIP, void * data)
{
    // process request from interface hNetIf and remoteIP address
    uint16_t* pReply = (uint16_t*)data;
    uint16_t myRecvId = *pReply;
    uint16_t myRecvSequenceNumber = *(pReply + 1);

    // check that the sequence number, ID and IP address match, etc.
}
```

Parameters

Parameters	Description
callback	notification function to be registered. This function will be called when an echo request reply is received.

Function

ICMP_HANDLE TCPIP_ICMP_CallbackRegister (void (*callback)(TCPIP_NET_HANDLE hNetIf,
IPV4_ADDR * remoteIP, void * data))

TCPIP_ICMP_EchoRequestSend Function

Sends an ICMP echo request to a remote node.

File

icmp.h

C

```
ICMP_ECHO_RESULT TCPIP_ICMP_EchoRequestSend(TCPIP_NET_HANDLE netH, IPV4_ADDR * targetAddr, uint16_t
sequenceNumber, uint16_t identifier);
```

Returns

- ICMP_ECHO_OK - Indicates the query request was successfully sent
- ICMP_ECHO_RESULT - The query request was unsuccessfully sent, which results in an error code (interface not ready for transmission, allocation error, etc.)

Description

This function allows a stack client to send an ICMP query message to a remote host. The supplied sequence number and identifier will be used in the query message. The user will be notified by the result of the query using a notification handle registered by using the TCPIP_ICMP_CallbackRegister function.

Remarks

This function does not allow the user to specify a data buffer to be sent as part of the request. The preferred function to use is TCPIP_ICMP_EchoRequest.

This function will eventually be obsoleted.

Preconditions

The TCP/IP Stack must be initialized and up and running.

Example

```
IPV4_ADDR remoteAddress = 0xc0a00101;
```

```
uint16_t mySequenceNumber = 1;
uint16_t myId = 0x1234;

if(TCPIP_ICMP_EchoRequestSend(0, &remoteAddress, mySequenceNumber, myId) == ICMP_ECHO_OK )
{
    // successfully sent the ICMP request
    //
}
else
{
    // process the error
}
```

Function

ICMP_ECHO_RESULT TCPIP_ICMP_EchoRequestSend (TCPIP_NET_HANDLE netH, IPV4_ADDR * targetAddr, uint16_t sequenceNumber, uint16_t identifier)

TCPIP_ICMP_Task Function

Standard TCP/IP stack module task function.

File

icmp.h

C

```
void TCPIP_ICMP_Task();
```

Returns

None.

Description

This function performs ICMPv4 module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The ICMP module should have been initialized.

Function

void TCPIP_ICMP_Task(void)

TCPIP_ICMP_EchoRequest Function

Sends an ICMP echo request containing user supplied data

File

icmp.h

C

```
ICMP_ECHO_RESULT TCPIP_ICMP_EchoRequest(TCPIP_ICMP_ECHO_REQUEST* pEchoRequest, TCPIP_ICMP_REQUEST_HANDLE* pHandle);
```

Returns

- ICMP_ECHO_OK - Indicates the query request was successfully sent
- ICMP_ECHO_RESULT - The query request was unsuccessfully sent, which results in an error code (interface not ready for transmission, allocation error, etc.)

Description

This function allows a stack client to send an ICMP query message to a remote host. The supplied sequence number and identifier will be used in the query message. The request will also contain user supplied data. The request is scheduled and the user will be notified of the outcome of the query using the callback function that's specified in the call.

Remarks

The data buffer that's passed as part of the callback routine is no longer available after the callback routine returns control.

This routine is more generic than TCPIP_ICMP_EchoRequestSend and is preferred.

The user has to enforce the fact that the "identifier" field has to be unique per (destination address, source address) pair.

Currently there could be only one ping/echo request active at a time. If another echo request is active, a ICMP_ECHO_BUSY code will be returned.

Once the callback notification occurs, the echo request is completed and the icmpHandle is invalidated.

Preconditions

The TCP/IP Stack must be initialized and up and running.

Example

```
  uint8_t  myDataBuffer[200];     // buffer for the ping data
  void EchoCallback(TCPIP_ICMP_ECHO_REQUEST* pReqData, TCPIP_ICMP_REQUEST_HANDLE icmpHandle,
TCPIP_ICMP_ECHO_REQUEST_RESULT result);    // callback function to be called

  TCPIP_ICMP_ECHO_REQUEST myEchoRequest;
  myEchoRequest.netH  = 0;
  myEchoRequest.targetAddr  = 0xc0a00101;
  myEchoRequest.sequenceNumber = 1;
  myEchoRequest.identifier     = 0x1234;
  myEchoRequest.pData          = myDataBuffer;
  myEchoRequest.dataSize       = sizeof(myDataBuffer);
  myEchoRequest.callback       = EchoCallback;


  if(TCPIP_ICMP_EchoRequest(&myEchoRequest, 0) == ICMP_ECHO_OK )
  {
      // successfully sent the ICMP request
      //
      // EchoCallback() will be called and data can be examined
  }
  else
  {
      // process the error
  }
```

Function

ICMP_ECHO_RESULT TCPIP_ICMP_EchoRequest (TCPIP_ICMP_ECHO_REQUEST* pEchoRequest, TCPIP_ICMP_REQUEST_HANDLE* pHandle);

TCPIP_ICMP_EchoRequestCancel Function

Cancels a previously sent ICMP echo request

File

icmp.h

C

```
ICMP_ECHO_RESULT TCPIP_ICMP_EchoRequestCancel(TCPIP_ICMP_REQUEST_HANDLE icmpHandle);
```

Returns

- ICMP_ECHO_OK - Indicates the cancel request was successful
- ICMP_ECHO_BAD_HANDLE - No such request is currently scheduled Invalid handle

Description

This function allows a stack client to cancel a pending ICMP echo request. The request should have been previously scheduled with TCPIP_ICMP_EchoRequest.

Remarks

None

Preconditions

The TCP/IP Stack must be initialized and up and running. Valid handle obtained using TCPIP_ICMP_EchoRequest.

Example

Function

ICMP_ECHO_RESULT TCPIP_ICMP_EchoRequestCancel (TCPIP_ICMP_REQUEST_HANDLE icmpHandle);

b) Data Types and Constants

ICMP_ECHO_RESULT Enumeration

File

icmp.h

C

```
typedef enum {
  ICMP_ECHO_OK = 0,
  ICMP_ECHO_ALLOC_ERROR = -1,
  ICMP_ECHO_ROUTE_ERROR = -2,
  ICMP_ECHO_TRANSMIT_ERROR = -3,
  ICMP_ECHO_PARAMETER_ERROR = -4,
  ICMP_ECHO_BUSY = -5,
  ICMP_ECHO_BAD_HANDLE = -6
} ICMP_ECHO_RESULT;
```

Members

Members	Description
ICMP_ECHO_OK = 0	operation successful error codes, < 0
ICMP_ECHO_ALLOC_ERROR = -1	could not allocate memory
ICMP_ECHO_ROUTE_ERROR = -2	could not find a route to destination
ICMP_ECHO_TRANSMIT_ERROR = -3	could not transmit (dead interface, etc.)
ICMP_ECHO_PARAMETER_ERROR = -4	null pointer, etc.
ICMP_ECHO_BUSY = -5	module currently busy
ICMP_ECHO_BAD_HANDLE = -6	invalid handle, no such request exist

Description

result of an ICMP Echo Request call

ICMP_HANDLE Type

File

icmp.h

C

```
typedef const void* ICMP_HANDLE;
```

Description

a handle that a client can use after the event handler has been registered

TCPIP_ICMP_MODULE_CONFIG Structure

Placeholder for ICMP module configuration.

File

icmp.h

C

```
typedef struct {
} TCPIP_ICMP_MODULE_CONFIG;
```

Description

ICMP Module Configuration Structure Typedef

Provides a placeholder for ICMP module configuration.

Remarks

None.

TCPIP_ICMP_ECHO_REQUEST Structure

Contains the data needed for an ICMP request operation

File

icmp.h

C

```
typedef struct _tag_TCPIP_ICMP_ECHO_REQUEST {
  TCPIP_NET_HANDLE netH;
  IPV4_ADDR targetAddr;
  uint16_t sequenceNumber;
  uint16_t identifier;
  uint8_t* pData;
  uint16_t dataSize;
  void (* callback)(const struct _tag_TCPIP_ICMP_ECHO_REQUEST* pReqData, TCPIP_ICMP_REQUEST_HANDLE
icmpHandle, TCPIP_ICMP_ECHO_REQUEST_RESULT result);
} TCPIP_ICMP_ECHO_REQUEST;
```

Description

ICMP Request Data Structure

This structure defines the information required for sending an ICMP request that contains user data.

Remarks

When the callback is made with an error result, the TCPIP_ICMP_ECHO_REQUEST structure may identify the request, if no reply was received (for example timeout)

TCPIP_ICMP_ECHO_REQUEST_RESULT Enumeration

File

icmp.h

C

```
typedef enum {
  TCPIP_ICMP_ECHO_REQUEST_RES_OK = 0,
  TCPIP_ICMP_ECHO_REQUEST_RES_TMO = -1
} TCPIP_ICMP_ECHO_REQUEST_RESULT;
```

Members

Members	Description
TCPIP_ICMP_ECHO_REQUEST_RES_OK = 0	a reply has been successfully received error codes, < 0
TCPIP_ICMP_ECHO_REQUEST_RES_TMO = -1	no reply received; the current request timed out and will be deleted

Description

A result code for an Echo Request callback (as a result of an ICMP TCPIP_ICMP_EchoRequest call)

TCPIP_ICMP_REQUEST_HANDLE Type

File

icmp.h

C

```
typedef const void* TCPIP_ICMP_REQUEST_HANDLE;
```

Description

This is type TCPIP_ICMP_REQUEST_HANDLE.

Files

Files

Name	Description
icmp.h	The Internet Control Message Protocol is used to send error and status messages and requests.
icmp_config.h	ICMP configuration file

Description

This section lists the source and header files used by the library.

icmp.h

The Internet Control Message Protocol is used to send error and status messages and requests.

Enumerations

	Name	Description
	ICMP_ECHO_RESULT	result of an ICMP Echo Request call
	TCPIP_ICMP_ECHO_REQUEST_RESULT	A result code for an Echo Request callback (as a result of an ICMP TCPIP_ICMP_EchoRequest call)

Functions

	Name	Description
	TCPIP_ICMP_CallbackDeregister	Deregisters the ICMP callback function.
	TCPIP_ICMP_CallbackRegister	Registers a callback to allow the application layer to process incoming ICMPv4 packets
	TCPIP_ICMP_EchoRequest	Sends an ICMP echo request containing user supplied data
	TCPIP_ICMP_EchoRequestCancel	Cancels a previously sent ICMP echo request
	TCPIP_ICMP_EchoRequestSend	Sends an ICMP echo request to a remote node.
	TCPIP_ICMP_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	_tag_TCPIP_ICMP_ECHO_REQUEST	Contains the data needed for an ICMP request operation
	TCPIP_ICMP_ECHO_REQUEST	Contains the data needed for an ICMP request operation
	TCPIP_ICMP_MODULE_CONFIG	Placeholder for ICMP module configuration.

Types

	Name	Description
	ICMP_HANDLE	a handle that a client can use after the event handler has been registered
	TCPIP_ICMP_REQUEST_HANDLE	This is type TCPIP_ICMP_REQUEST_HANDLE.

Description

ICMP Module Definitions for the Microchip TCP/IP Stack

The Internet Control Message Protocol is used to send error and status messages and requests. The ICMP module implements the Echo Reply message type (commonly referred to as a ping) which can be used to determine if a specified host is reachable across an IP network from a device running the TCP/IP stack. An ICMP server is also supported to respond to pings from other devices.

File Name

icmp.h

Company

Microchip Technology Inc.

icmp_config.h

ICMP configuration file

Macros

	Name	Description
	TCPIP_ICMP_CLIENT_USER_NOTIFICATION	allow ICMP client user notification if enabled, the TCPIP_ICMP_CallbackRegister/TCPIP_ICMP_CallbackDeregister functions exist and can be used Note that if disabled, the tcpip console ping command won't be available
	TCPIP_ICMP_ECHO_REQUEST_TIMEOUT	Timeout for a request that was sent and no reply was received milliseconds
	TCPIP_ICMP_TASK_TICK_RATE	The ICMP task processing rate: number of milliseconds to generate an ICMP tick. This is the tick that advances the ICMP state machine. The default value is around 30 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

Internet Control Message Protocol (ICMP) Configuration file

This file contains the ICMP module configuration options

File Name

icmp_config.h

Company

Microchip Technology Inc.

ICMPv6 Module

This section describes the TCP/IP Stack Library ICMPv6 module.

Introduction

TCP/IP Library ICMPv6 Module for Microchip Microcontrollers

This library provides the API of the ICMPv6 module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

This document specifies a set of APIs of Internet Control Message Protocol (ICMP) messages for use with version 6 of the Internet Protocol (IPv6).

The Internet Protocol, version 6 (IPv6) is a new version of IP, uses the Internet Control Message Protocol (ICMP) as defined for IPv4

[RFC-792], with a number of changes. It is called ICMPv6, and has an IPv6 Next Header value of 58.

ICMPv6 messages are used by IPv6 nodes to report *error messages* and *information messages*. ICMPv6 is also used for Ipv6 node diagnostic (i.e., IPv6 ping).

The ICMPv6 protocol also provides a framework for the following:

Neighbor Discovery

Neighbor Discovery is a series of five ICMPv6 messages that manage node-to-node communication on a link. Neighbor Discovery replaces Address Resolution Protocol (ARP), ICMP (IPv4) Router Discovery, and the ICMP (IPv4) Redirect message.

Multicast Listener Discovery (MLD)

Multicast Listener Discovery is a series of three ICMP messages that manage subnet multicast membership. Multicast Listener Discovery replaces version 2 of the Internet Group Management Protocol (IGMP) for IPv4.

Path MTU Discovery

The maximum transmission unit (MTU) for a path is the minimum link MTU of all links on a path between a source and a destination. IPv6 packets that are smaller than the path MTU do not require fragmentation by the host and are successfully forwarded by all routers on the path. To discover the path MTU, the sending host uses the receipt of **ICMPv6 Packet Too Big messages**.

ICMPv6 Common Messages	Functions
Echo Request	Sent to check IPv6 connectivity to a particular host.
Echo Reply	Sent in response to an ICMPv6 Echo Request.
Destination Unreachable	Sent by a router or the destination host to inform the sending host that the packet or payload cannot be delivered.
Packet to big	Sent by a router to inform a sending host that a packet is too large to forward.
Time exceeded	Sent by a router to inform a sending host that the Hop Limit of an IPv6 packet has expired.
Parameter Problem	Sent by a router to inform a sending host that an error was encountered in processing the IPv6 header or an IPv6 extension header.

Using the Library

This topic describes the basic architecture of the ICMPv6 TCP/IP Library and provides information and examples on its use.

Description

Interface Header File: `icmpv6.h`

The interface to the ICMPv6 TCP/IP library is defined in the `icmpv6.h` header file. Any C language source (`.c`) file that uses the ICMPv6 TCP/IP library should include `ipv6.h` , `icmpv6.h`, and `ndp.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

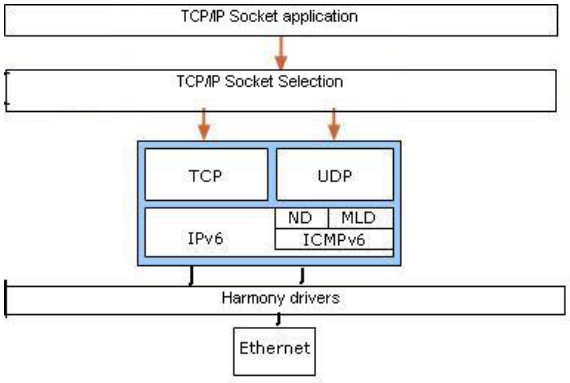
This library provides the API of the ICMPv6 TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

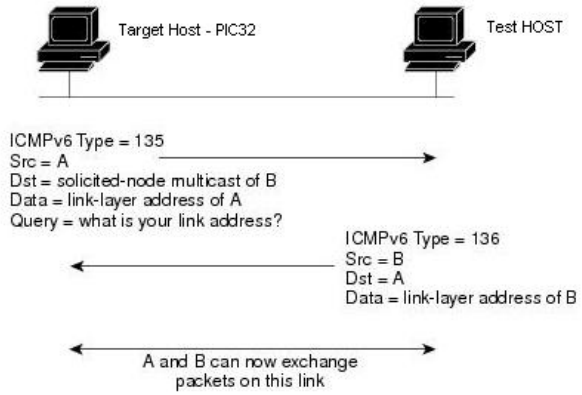
ICMPv6 Software Abstraction Block Diagram

This module provides software abstraction of the IPv6 module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.

IPv6 Block Diagram



ICMPv6 message communication



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the IPv6 and ICMPv6 module.

Library Interface Section	Description
Functions	This section provides general interface routines
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

The configuration of the ICMPv6 TCP/IP Stack is based on the file `icmpv6_config.h`.

This header file contains the configuration selection for the ICMPv6 TCP/IP Stack. Based on the selections made, the ICMPv6 TCP/IP Stack IP may support the selected features. These configuration settings will apply to all instances of the ICMPv6 TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build.

Building the Library

This section lists the files that are available in the ICMPv6 module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/icmpv6.c`	ICMPv6 implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The ICMPv6 module depends on the following modules:

- TCP/IP Stack Library
- IPv6 Module

Library Interface

a) Functions

	Name	Description
	TCPIP_ICMPV6_CallbackDeregister	Deregisters an upper-layer function from ICMPv6 callback.
	TCPIP_ICMPV6_Flush	Flushes a ICMPv6 packet.
	TCPIP_ICMPV6_HeaderEchoRequestPut	Allocates a packet, IPv6 Header, and Upper-layer header for an ICMPv6 echo request.
	TCPIP_ICMPV6_EchoRequestSend	Sends an ICMPv6 echo request to a remote node.
	TCPIP_ICMPV6_CallbackRegister	Registers an upper-layer function for ICMPv6 callback.

b) Data Types and Constants

	Name	Description
	ICMPV6_ERR_PTB_CODE	ICMPv6 Packet Too Big error code.
	ICMPV6_INFO_EREQ_CODE	ICMPv6 Packet Echo Request information code.
	ICMPV6_INFO_ERPL_CODE	ICMPv6 Packet Echo Reply information code.
	TCPIP_ICMPV6_PutHeaderEchoReply	This is macro TCPIP_ICMPV6_PutHeaderEchoReply.
	ICMPV6_ERR_DU_CODE	ICMPv6 Destination Unreachable error code.
	ICMPV6_ERR_PP_CODE	ICMPv6 Parameter Problem error code.
	ICMPV6_ERR_TE_CODE	ICMPv6 Time Exceeded error code.
	ICMPV6_HANDLE	ICMPv6 handle.
	ICMPV6_HEADER_ECHO	Header for an ICMPv6 Echo Request/Reply packet
	ICMPV6_HEADER_ERROR	Header for an ICMPv6 Error packet.
	ICMPV6_PACKET_TYPES	ICMPv6 packet types.
	TCPIP_ICMPV6_CLIENT_USER_NOTIFICATION	allow ICMPV6 client user notification if enabled, the TCPIP_ICMPV6_CallbackRegister/TCPIP_ICMPV6_CallbackDeregister functions exist and can be used Note that if disabled, the tcpip console ping6 command won't be available

Description

This section describes the Application Programming Interface (API) functions of the ICMPv6 module.

Refer to each section for a detailed description.

a) Functions

TCPIP_ICMPV6_CallbackDeregister Function

Deregisters an upper-layer function from ICMPv6 callback.

File

icmpv6.h

C

```
bool TCPIP_ICMPV6_CallbackDeregister(ICMPV6_HANDLE hIcmpv6);
```

Returns

- true - On Success
- false - On Failure (If no such handler registered)

Description

This function deregisters a previously registered function from the ICMPv6 register list.

Remarks

None.

Preconditions

IPv6 and ICMPv6 are initialized.

Parameters

Parameters	Description
hIcmpv6	ICMPv6 handler

Function

ICMPV6_HANDLE TCPIP_ICMPV6_CallbackRegister (void (*callback)(TCPIP_NET_HANDLE hNetIf, uint8_t type, IPV6_ADDR * localIP, IPV6_ADDR * remoteIP, void * header))

TCPIP_ICMPV6_Flush Function

Flushes a ICMPv6 packet.

File

icmpv6.h

C

```
bool TCPIP_ICMPV6_Flush(IPV6_PACKET * pkt);
```

Returns

- true - if the packet was flushed
- false - if the packet was queued

Description

Transmit out a ICMPv6 packet.

Remarks

None.

Preconditions

IPv6 and ICMPv6 are initialized.

Parameters

Parameters	Description
pkt	The packet to flush

Function

bool TCPIP_ICMPV6_Flush (IPV6_PACKET * pkt)

TCPIP_ICMPV6_HeaderEchoRequestPut Function

Allocates a packet, IPv6 Header, and Upper-layer header for an ICMPv6 echo request.

File

icmpv6.h

C

```
IPV6_PACKET * TCPIP_ICMPV6_HeaderEchoRequestPut(TCPIP_NET_HANDLE hNetIf, const IPV6_ADDR * localIP, const
IPV6_ADDR * remoteIP, uint8_t type, uint16_t identifier, uint16_t sequenceNumber);
```

Returns

IPV6_PACKET * - The constructed error packet or NULL

Description

This function allocates a packet using TCPIP_ICMPV6_Open() and updates local and remote addresses. Updates IPv6 Header with ICMPv6 type and upper-layer header for an ICMPv6 echo request.

Remarks

None.

Preconditions

IPv6 and ICMPv6 are initialized.

Parameters

Parameters	Description
pNetIf	The interface for the outgoing packet.
localIP	The local address that should be used for this packet.
remoteIP	The packet's destination address
type	Echo Request or Echo Reply
identifier	The Echo Request id.
sequenceNumber	The Echo request sequence number

Function

IPV6_PACKET * TCPIP_ICMPV6_HeaderEchoRequestPut (TCPIP_NET_HANDLE hNetIf,
IPV6_ADDR * localIP, IPV6_ADDR * remoteIP, uint8_t type,
uint16_t identifier, uint16_t sequenceNumber)

TCPIP_ICMPV6_EchoRequestSend Function

Sends an ICMPv6 echo request to a remote node.

File

icmpv6.h

C

```
bool TCPIP_ICMPV6_EchoRequestSend(TCPIP_NET_HANDLE netH, IPV6_ADDR * targetAddr, uint16_t sequenceNumber,
uint16_t identifier, uint32_t packetSize);
```

Returns

- true - if the query request was successfully sent
- false - otherwise (interface not ready for transmission, allocation error, etc.)

Description

This function allows a stack client to send an ICMPv6 Echo request message to a remote host. The supplied sequence number and identifier will be used in the query message. The user will be notified by the result of the query using a notification handle registered by using the TCPIP_ICMPV6_CallbackRegister function.

Remarks

None.

Preconditions

The TCP/IP Stack is initialized and up and running. IPv6 and ICMPv6 are initialized.

Function

bool TCPIP_ICMPV6_EchoRequestSend(TCPIP_NET_HANDLE netH, IPV6_ADDR * targetAddr,
uint16_t sequenceNumber, uint16_t identifier,uint32_t packetSize);

TCPIP_ICMPV6_CallbackRegister Function

Registers an upper-layer function for ICMPv6 callback.

File

icmpv6.h

C

```
ICMPV6_HANDLE TCPIP_ICMPV6_CallbackRegister(void (*callback)(TCPIP_NET_HANDLE hNetIf, uint8_t type, const
IPV6_ADDR * localIP, const IPV6_ADDR * remoteIP, void * header));
```

Returns

ICMPV6_HANDLE

- Success - Returns a valid handle
- Failure - Null handle

Description

This function registers an upper-layer function to handle ICMPv6 messages that may require action at the application layer (Echo Replies, Error messages)

Remarks

None.

Preconditions

IPv6 and ICMPv6 are initialized.

Parameters

Parameters	Description
type	ICMPv6 header type
localIP	IPv6 destination address of the incoming message
remoteIP	IPv6 address of the node that originated the incoming message
header	Pointer to the ICMPv6 header

Function

ICMPV6_HANDLE TCPIP_ICMPV6_CallbackRegister (void (*callback)(TCPIP_NET_HANDLE hNetIf,
uint8_t type, IPV6_ADDR * localIP, IPV6_ADDR * remoteIP, void * header))

b) Data Types and Constants

ICMPV6_ERR_PTB_CODE Macro

ICMPv6 Packet Too Big error code.

File

icmpv6.h

C

```
#define ICMPV6_ERR_PTB_CODE 0u
```

Description

Macro: ICMPV6_ERR_PTB_CODE

ICMPv6 packet too big error code value is 0 generated by the originator and ignored by the receiver.

Remarks

A Packet Too Big MUST be sent by a router in response to a packet that it cannot forward because the packet is larger than the MTU of the outgoing link. The information in this message is used as part of the Path MTU Discovery process [PMTU].

More details on RFC - 4446.

ICMPV6_INFO_EREQ_CODE Macro

ICMPv6 Packet Echo Request information code.

File

icmpv6.h

C

```
#define ICMPV6_INFO_EREQ_CODE 0u
```

Description

Macro: ICMPV6_INFO_EREQ_CODE

Definition for ICMPv6 Packet Echo Request information code.

Remarks

More details on RFC - 4446.

ICMPV6_INFO_ERPL_CODE Macro

ICMPv6 Packet Echo Reply information code.

File

icmpv6.h

C

```
#define ICMPV6_INFO_ERPL_CODE 0u
```

Description

Macro: ICMPV6_INFO_ERPL_CODE

Definition for ICMPv6 Packet Echo Reply information code.

Remarks

More details on RFC - 4446.

TCPIP_ICMPV6_PutHeaderEchoReply Macro

File

icmpv6.h

C

```
#define TCPIP_ICMPV6_PutHeaderEchoReply TCPIP_ICMPV6_HeaderEchoRequestPut
```

Description

This is macro TCPIP_ICMPV6_PutHeaderEchoReply.

ICMPV6_ERR_DU_CODE Enumeration

ICMPv6 Destination Unreachable error code.

File

icmpv6.h

C

```
typedef enum {
  ICMPV6_ERR_DU_NO_ROUTE = 0u,
  ICMPV6_ERR_DU_PROHIBITED = 1u,
  ICMPV6_ERR_DU_OUTSIDE_SCOPE = 2u,
  ICMPV6_ERR_DU_ADDR_UNREACHABLE = 3u,
  ICMPV6_ERR_DU_PORT_UNREACHABLE = 4u,
  ICMPV6_ERR_DU_SRC_FAILED_INGRESS_POLICY = 5u,
  ICMPV6_ERR_DU_REJECT_ROUTE = 6u
} ICMPV6_ERR_DU_CODE;
```

Members

Members	Description
ICMPV6_ERR_DU_NO_ROUTE = 0u	No route to destination
ICMPV6_ERR_DU_PROHIBITED = 1u	Communication with destination administratively prohibited
ICMPV6_ERR_DU_OUTSIDE_SCOPE = 2u	Beyond scope of source address

ICMPV6_ERR_DU_ADDR_UNREACHABLE = 3u	Address unreachable
ICMPV6_ERR_DU_PORT_UNREACHABLE = 4u	Port unreachable
ICMPV6_ERR_DU_SRC_FAILED_INGRESS_POLICY = 5u	Source address failed ingress/egress policy
ICMPV6_ERR_DU_REJECT_ROUTE = 6u	Reject route to destination

Description

Enumeration: ICMPV6_ERR_DU_CODE

Defines different type of ICMPv6 Destination Unreachable error code.

Remarks

A Destination Unreachable message should be generated by a router, or by the IPv6 layer in the originating node, in response to a packet that cannot be delivered to its destination address for reasons other than congestion. An ICMPv6 message MUST NOT be generated if a packet is dropped due to congestion.

More details on RFC - 4446.

ICMPV6_ERR_PP_CODE Enumeration

ICMPv6 Parameter Problem error code.

File

icmpv6.h

C

```
typedef enum {
  ICMPV6_ERR_PP_ERRONEOUS_HEADER = 0u,
  ICMPV6_ERR_PP_UNRECOGNIZED_NEXT_HEADER = 1u,
  ICMPV6_ERR_PP_UNRECOGNIZED_IPV6_OPTION = 2u
} ICMPV6_ERR_PP_CODE;
```

Members

Members	Description
ICMPV6_ERR_PP_ERRONEOUS_HEADER = 0u	Erroneous header field encountered
ICMPV6_ERR_PP_UNRECOGNIZED_NEXT_HEADER = 1u	Unrecognized Next Header type encountered
ICMPV6_ERR_PP_UNRECOGNIZED_IPV6_OPTION = 2u	Unrecognized IPv6 option encountered

Description

Enumeration: ICMPV6_ERR_PP_CODE

Definitions for ICMPv6 Parameter Problem error code.

Remarks

More details on RFC - 4446.

ICMPV6_ERR_TE_CODE Enumeration

ICMPv6 Time Exceeded error code.

File

icmpv6.h

C

```
typedef enum {
  ICMPV6_ERR_TE_HOP_LIMIT_EXCEEDED = 0u,
  ICMPV6_ERR_TE_FRAG_ASSEMBLY_TIME_EXCEEDED = 1u
} ICMPV6_ERR_TE_CODE;
```

Members

Members	Description
ICMPV6_ERR_TE_HOP_LIMIT_EXCEEDED = 0u	Hop limit exceeded in transit

ICMPV6_ERR_TE_FRAG_ASSEMBLY_TIME_EXCEEDED = 1u	Fragment reassembly time exceeded

Description

Enumeration: ICMPV6_ERR_TE_CODE

Definitions for ICMPv6 Time Exceeded error code. An ICMPv6 Time Exceeded message with Code 1 is used to report fragment reassembly time out.

Remarks

More details on RFC - 4446.

ICMPV6_HANDLE Type

ICMPv6 handle.

File

icmpv6.h

C

```
typedef const void* ICMPV6_HANDLE;
```

Description

Type: ICMPV6_HANDLE

A handle that a client needs to use when deregistering a notification handler.

Remarks

This handle can be used by the client after the event handler has been registered.

ICMPV6_HEADER_ECHO Structure

File

icmpv6.h

C

```
typedef struct {
  uint8_t vType;
  uint8_t vCode;
  uint16_t wChecksum;
  uint16_t identifier;
  uint16_t sequenceNumber;
} ICMPV6_HEADER_ECHO;
```

Members

Members	Description
uint8_t vType;	icmpV6 request or reply type
uint8_t vCode;	Error code
uint16_t wChecksum;	Packet TX and RX checksum
uint16_t identifier;	incoming and outgoing packet identifier
uint16_t sequenceNumber;	request and reply sequence number

Description

Header for an ICMPv6 Echo Request/Reply packet

ICMPV6_HEADER_ERROR Structure

Header for an ICMPv6 Error packet.

File

icmpv6.h

C

```
typedef struct {
  uint8_t vType;
  uint8_t vCode;
  uint16_t wChecksum;
  uint32_t additionalData;
} ICMPV6_HEADER_ERROR;
```

Members

Members	Description
uint8_t vType;	icmpV6 request or reply type
uint8_t vCode;	error code
uint16_t wChecksum;	Packet TX and RX checksum
uint32_t additionalData;	Unused for Destination. Unreachable and Time Exceeded. MTU for MTU. Pointer for Parameter Problem.

Description

Structure: ICMPV6_HEADER_ERROR

ICMPv6 different code and type error messages (Range 0 to 127).

Remarks

More details on RFC - 4446.

ICMPV6_PACKET_TYPES Enumeration

ICMPv6 packet types.

File

icmpv6.h

C

```
typedef enum {
  ICMPV6_ERROR_DEST_UNREACHABLE = 1u,
  ICMPV6_ERROR_PACKET_TOO_BIG = 2u,
  ICMPV6_ERROR_TIME_EXCEEDED = 3u,
  ICMPV6_ERROR_PARAMETER_PROBLEM = 4u,
  ICMPV6_INFO_ECHO_REQUEST = 128u,
  ICMPV6_INFO_ECHO_REPLY = 129u,
  ICMPV6_INFO_ROUTER_SOLICITATION = 133u,
  ICMPV6_INFO_ROUTER_ADVERTISEMENT = 134u,
  ICMPV6_INFO_NEIGHBOR_SOLICITATION = 135u,
  ICMPV6_INFO_NEIGHBOR_ADVERTISEMENT = 136u,
  ICMPV6_INFO_REDIRECT = 137u
} ICMPV6_PACKET_TYPES;
```

Members

Members	Description
ICMPV6_ERROR_DEST_UNREACHABLE = 1u	Destination Unreachable error packet
ICMPV6_ERROR_PACKET_TOO_BIG = 2u	Packet Too Big error packet
ICMPV6_ERROR_TIME_EXCEEDED = 3u	Time Exceeded error packet
ICMPV6_ERROR_PARAMETER_PROBLEM = 4u	Parameter Problem error packet
ICMPV6_INFO_ECHO_REQUEST = 128u	Echo Request packet
ICMPV6_INFO_ECHO_REPLY = 129u	Echo Reply packet
ICMPV6_INFO_ROUTER_SOLICITATION = 133u	Router solicitation NDP packet
ICMPV6_INFO_ROUTER_ADVERTISEMENT = 134u	Router advertisement NDP packet
ICMPV6_INFO_NEIGHBOR_SOLICITATION = 135u	Neighbor Solicitation NDP packet
ICMPV6_INFO_NEIGHBOR_ADVERTISEMENT = 136u	Neighbor Advertisement NDP packet
ICMPV6_INFO_REDIRECT = 137u	Redirect NDP packet

Description

Enumeration: ICMPV6_PACKET_TYPES

Defines different type of ICMPv6 message. Values in the range from 0 to 127 indicate an error type of messages. Values in the range 128 to 255 indicate an information message.

Remarks

None.

TCPIP_ICMPV6_CLIENT_USER_NOTIFICATION Macro

File

icmpv6_config.h

C

```
#define TCPIP_ICMPV6_CLIENT_USER_NOTIFICATION true
```

Description

allow ICMPV6 client user notification if enabled, the TCPIP_ICMPV6_CallbackRegister/TCPIP_ICMPV6_CallbackDeregister functions exist and can be used Note that if disabled, the tcpip console ping6 command won't be available

Files

Files

Name	Description
icmpv6.h	IPv6 Internet Communication Message Protocol
icmpv6_config.h	ICMPv6 configuration file

Description

This section lists the source and header files used by the library.

icmpv6.h

IPv6 Internet Communication Message Protocol

Enumerations

	Name	Description
	ICMPV6_ERR_DU_CODE	ICMPv6 Destination Unreachable error code.
	ICMPV6_ERR_PP_CODE	ICMPv6 Parameter Problem error code.
	ICMPV6_ERR_TE_CODE	ICMPv6 Time Exceeded error code.
	ICMPV6_PACKET_TYPES	ICMPv6 packet types.

Functions

	Name	Description
	TCPIP_ICMPV6_CallbackDeregister	Deregisters an upper-layer function from ICMPv6 callback.
	TCPIP_ICMPV6_CallbackRegister	Registers an upper-layer function for ICMPv6 callback.
	TCPIP_ICMPV6_EchoRequestSend	Sends an ICMPv6 echo request to a remote node.
	TCPIP_ICMPV6_Flush	Flushes a ICMPv6 packet.
	TCPIP_ICMPV6_HeaderEchoRequestPut	Allocates a packet, IPv6 Header, and Upper-layer header for an ICMPv6 echo request.

Macros

	Name	Description
	ICMPV6_ERR_PTB_CODE	ICMPv6 Packet Too Big error code.
	ICMPV6_INFO_EREQ_CODE	ICMPv6 Packet Echo Request information code.
	ICMPV6_INFO_ERPL_CODE	ICMPv6 Packet Echo Reply information code.
	TCPIP_ICMPV6_PutHeaderEchoReply	This is macro TCPIP_ICMPV6_PutHeaderEchoReply.

Structures

	Name	Description
	ICMPV6_HEADER_ECHO	Header for an ICMPv6 Echo Request/Reply packet
	ICMPV6_HEADER_ERROR	Header for an ICMPv6 Error packet.

Types

	Name	Description
	ICMPV6_HANDLE	ICMPv6 handle.

Description

Internet Control Message Protocol (ICMP) IPv6 API Header File

Internet Control Message Protocol (ICMP) in IPv6 functions the same as ICMP in IPv4.ICMPv6 is used to report error messages and information messages for IPv6 nodes. ICMP packets in IPv6 are used in the IPv6 neighbor discovery process, path MTU discovery, and the Multicast Listener Discovery (MLD) protocol for IPv6. RFC - 4443.

File Name

icmpv6.h

Company

Microchip Technology Inc.

icmpv6_config.h

ICMPv6 configuration file

Macros

	Name	Description
	TCPIP_ICMPV6_CLIENT_USER_NOTIFICATION	allow ICMPV6 client user notification if enabled, the TCPIP_ICMPV6_CallbackRegister/TCPIP_ICMPV6_CallbackDeregister functions exist and can be used Note that if disabled, the tcpip console ping6 command won't be available

Description

Internet Control Message Protocol for IPv6 (ICMPv6) Configuration file

This file contains the ICMPv6 module configuration options

File Name

icmpv6_config.h

Company

Microchip Technology Inc.

Iperf Module

This section describes the TCP/IP Stack Library Iperf Module.

Introduction

This library provides the API of the Iperf module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

Iperf is a networking tool that helps to measure networking bandwidth and performance.

Using the Library

This topic describes the basic architecture of the Iperf TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `iperf.h`

The interface to the Iperf TCP/IP Stack library is defined in the `iperf.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the Iperf TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Using Iperf

Describes how to use in the TCP/IP Stack Library Iperf module.

Description

Iperf is a networking tool that helps to measure networking bandwidth and performance. The Iperf module can act as both a client and server for testing. Iperf has the ability to test both UDP and TCP. In the case of UDP, you can specify the size of the UDP datagrams. For TCP, Iperf measures the throughput of the payload.

To run Iperf, you will need a personal computer that has an Iperf application installed. There is an open source version that is maintained, as well as many other variants across the Internet.

Iperf is meant to be run on the command line. On the PIC32 MPLAB Harmony TCP/IP Stack device side, there is also a built-in command console that can accept your different Iperf settings from a system console, such as the UART.

Iperf measures performance data in a unidirectional format. Therefore, the side that the server is running on is considered the receiver, and provides the most accurate performance values.

Command Synopsis

```
iperf [ -s|-c <IP addr> ]
[ -u ]
[ -i <sec> ]
[ -b <bandwidth> ]
[ -t <time> ]
```

`-s`	Runs the Iperf server. No IP address needs to be specified.
`-c <IP addr>`	Runs the Iperf client. The IP address is the IP address of the server.
`-u`	Sends UDP datagrams.
`-i <sec>`	Specified the time interval, in seconds, that the display will be updated.
`-b <bandwidth>`	Specifies the amount of data to try and send. This option is only valid with UDP datagrams.
`-t <time>`	Amount of time, in seconds, to run the test.

Note: The socket size of TCP/UDP (especially TCP) will affect the benchmark result. In addition, activated TCP/IP modules will also consume CPU and Ethernet load. The traffic load in your test network environment will also affect the benchmark test.

To get a reasonable maximum benchmark/evaluation data for a PIC32 device target, it is recommended to disable other modules, such as HTTP, client example, server example, etc., and find a uncongested network.

Running the Demonstration

1. First, make sure the Iperf module and the System Console are enabled in your project.
2. Then, rebuild and program your project into target board.
3. After powering on the development board and associating the connected network, start the server side Iperf application first. If you start Iperf as a server on the development board in the console, this implies that you are trying to measure the PIC32 device Ethernet receiver performance. If you start the Iperf server on a personal computer, you will be measuring PIC Ethernet transmit performance.

PIC32 Ethernet Starter Kit Example

The following tests show receiver and transmitter performance, respectively on the PIC32 Ethernet Starter Kit with the following settings

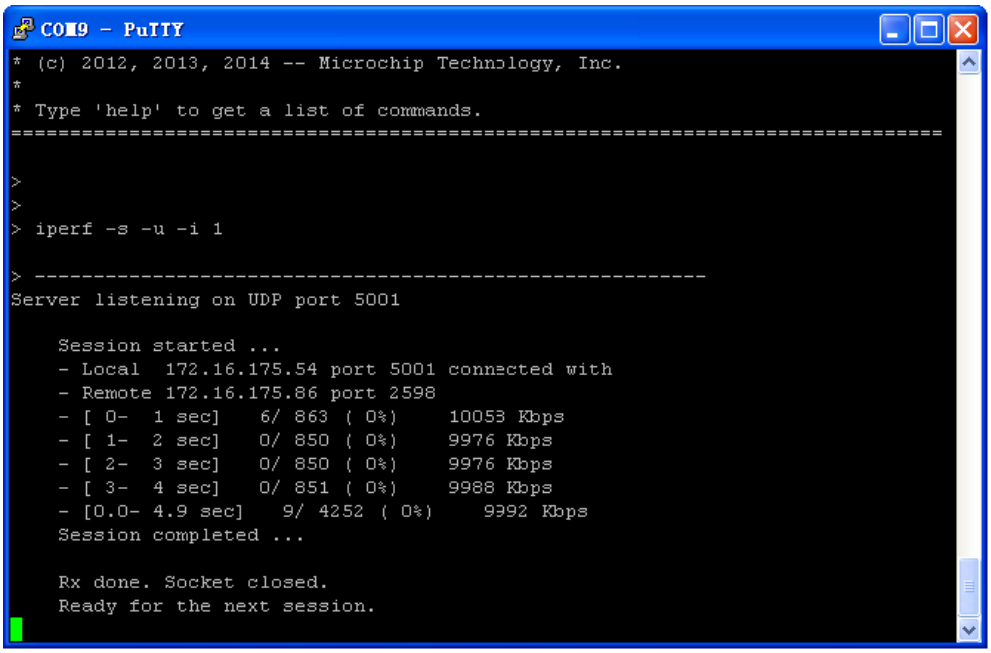
Note: The data does NOT show the maximum throughput of the PIC32 device as the settings are not optimized for that. Only the DHCP client, and the TCP and UDP modules are enabled in this example:

- TCP TX and RX Socket Size = 512
- UDP TX and RX Socket Size = 512
- Built by MPLAB XC32 C/C++ Compiler without any optimization
- Personal Computer IP Address: 172.16.175.86
- PIC32 Ethernet Starter Kit IP Address: 172.16.175.54

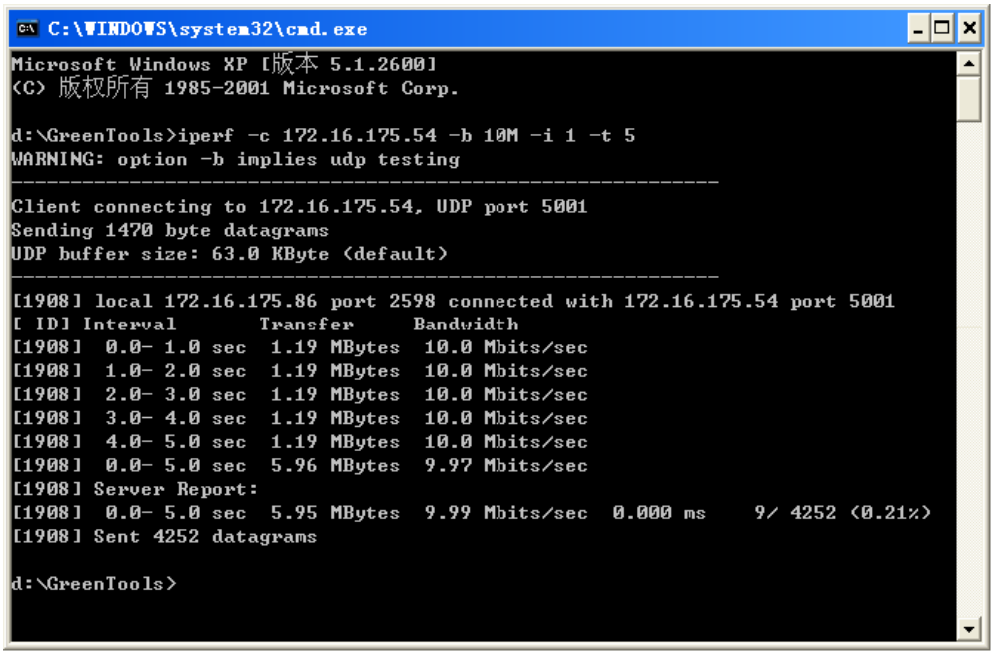
UDP Test

PIC32 ESK as UDP server (PIC32 receive):

- Command on the PIC32 device: `iperf -s -u -i 1`

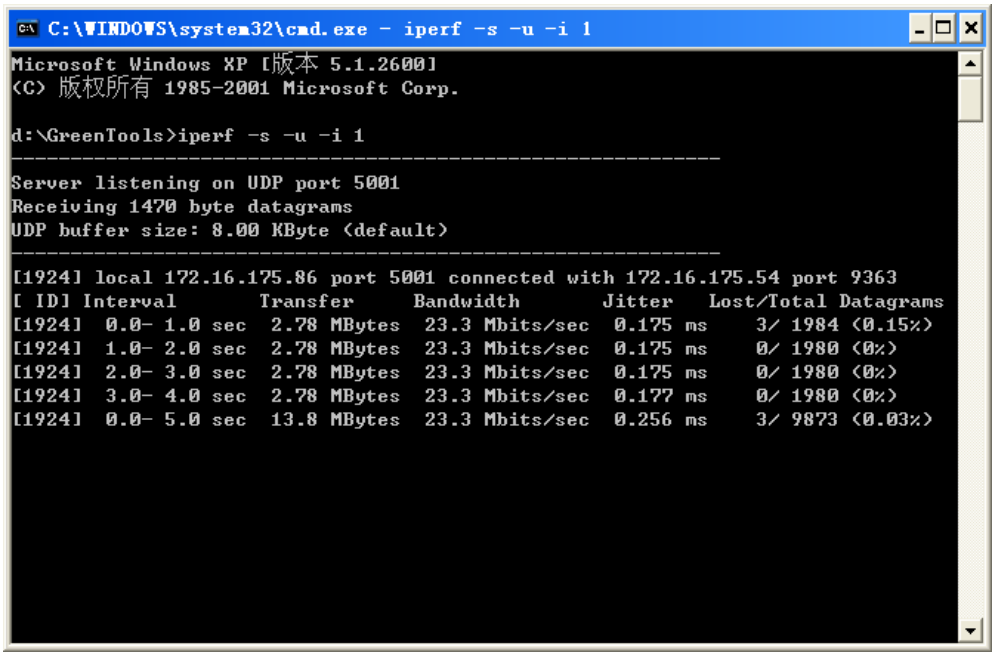


- Command on personal computer: `iperf -c 172.16.175.54 -b 10M -i 1 -t 5`

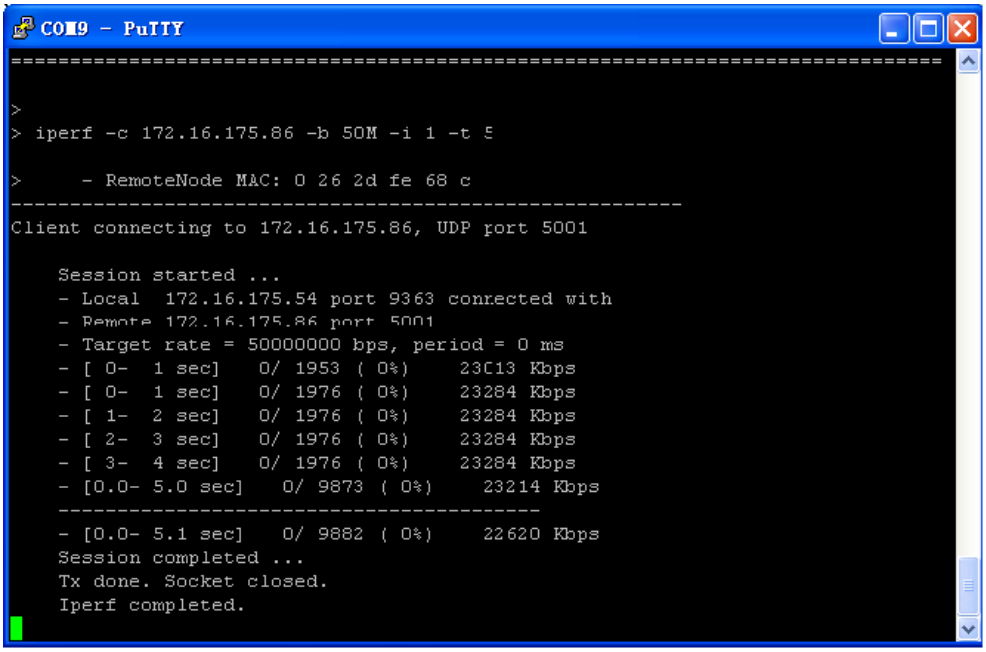


Personal computer as UDP server (PIC32 transmit)

- Command on personal computer: `iperf -s -u -i 1`

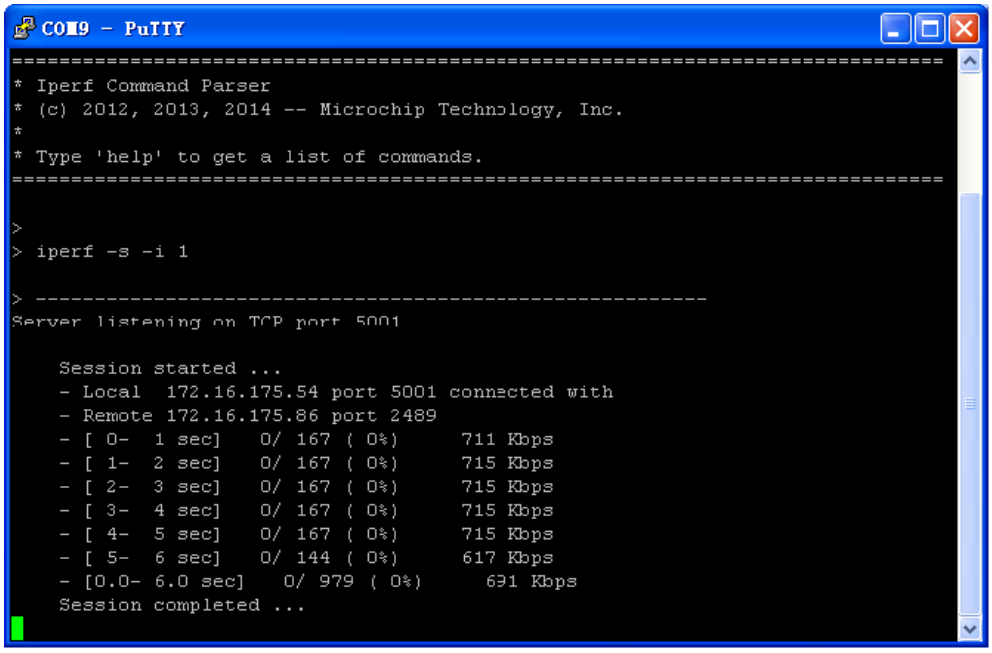


- Command on the PIC32 device: `iperf -c 172.16.175.86 -b 50M -i 1 -t 5`

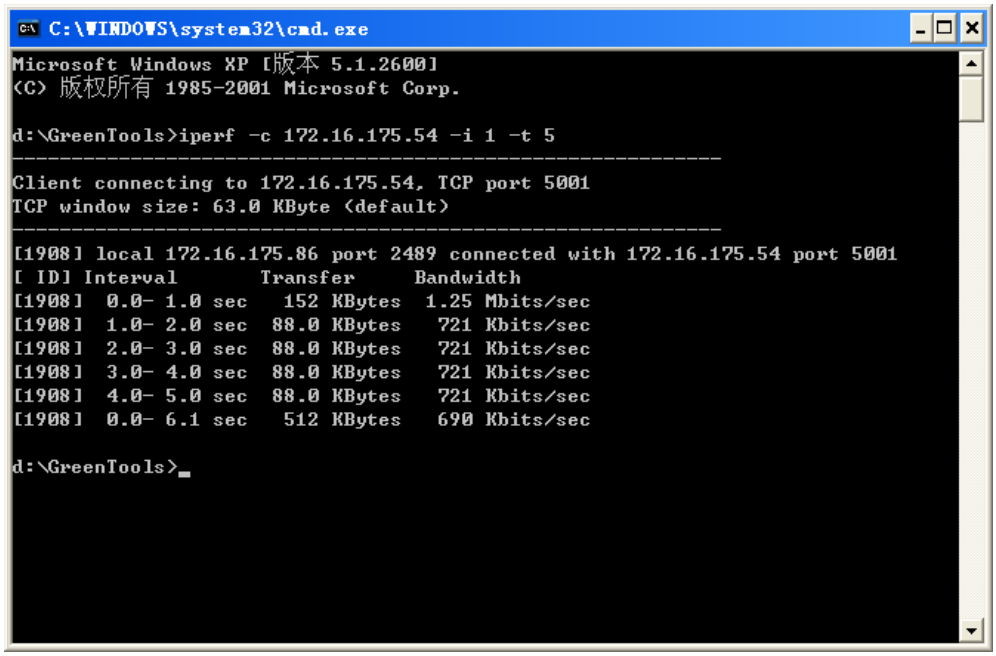


PIC32 ESK as TCP server (PIC32 receiving):

- Command on the PIC32 device: `iperf -s -i 1`

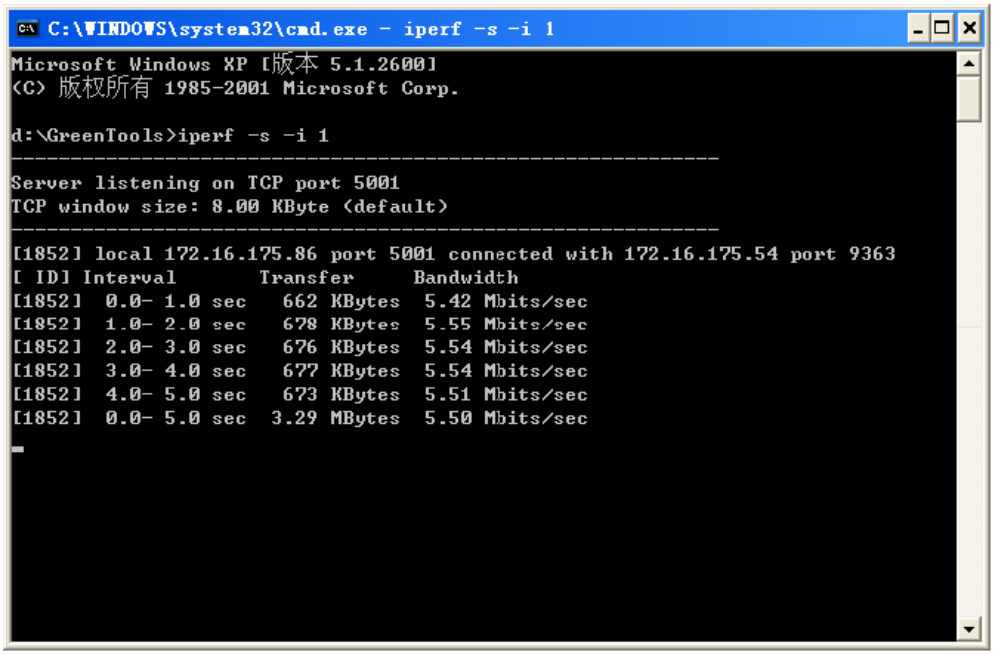


- Command on personal computer: `iperf -c 172.16.175.54 -i 1 -t 5`

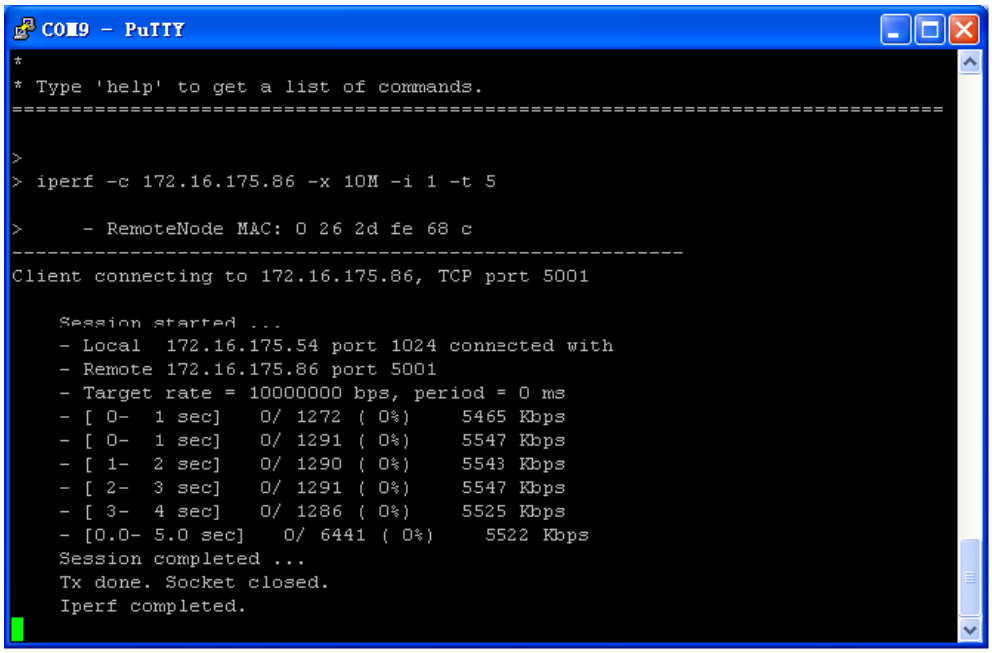


Personal computer as TCP server (PIC32 transmit):

- Command on personal computer: `iperf -s -i 1`



- Command on the PIC32 device: `iperf -c 172.16.175.86 -x 10M -i 1 -t 5`



Configuring the Module

Macros

	Name	Description
	TCPIP_IPERF_RX_BUFFER_SIZE	Default size of the Iperf RX buffer The default value is 4KB. The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Bigger buffers will help obtain higher performance numbers
	TCPIP_IPERF_TX_BUFFER_SIZE	Default size of the Iperf TX buffer The default value is 4KB. The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Bigger buffers will help obtain higher performance numbers
	TCPIP_IPERF_TX_QUEUE_LIMIT	for Iperf UDP client, the limit to set to avoid memory allocation overflow on slow connections
	TCPIP_IPERF_TX_WAIT_TMO	timeout to wait for TX channel to be ready to transmit a new packet; ms depends on the channel bandwidth

	TCPIP_IPERF_MAX_INSTANCES	maximum number of simultaneously running iperf instances
	TCPIP_IPERF_TIMING_ERROR_MARGIN	iperf timing error, ms. Accounts for timing uncertainty The higher the value, the more relaxed is the iperf timing Lower limit is 0

Description

The configuration of the Iperf TCP/IP Stack is based on the template file `iperf_config.h` with the actual symbols generated in the `system_config.h` file.

This header file contains the configuration selection for the Iperf TCP/IP Stack. Based on the selections made, the Iperf TCP/IP Stack may support the selected features.

TCPIP_IPERF_RX_BUFFER_SIZE Macro

File

iperf_config.h

C

```
#define TCPIP_IPERF_RX_BUFFER_SIZE 4096
```

Description

Default size of the Iperf RX buffer The default value is 4KB. The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Bigger buffers will help obtain higher performance numbers

TCPIP_IPERF_TX_BUFFER_SIZE Macro

File

iperf_config.h

C

```
#define TCPIP_IPERF_TX_BUFFER_SIZE 4096
```

Description

Default size of the Iperf TX buffer The default value is 4KB. The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Bigger buffers will help obtain higher performance numbers

TCPIP_IPERF_TX_QUEUE_LIMIT Macro

File

iperf_config.h

C

```
#define TCPIP_IPERF_TX_QUEUE_LIMIT 2
```

Description

for Iperf UDP client, the limit to set to avoid memory allocation overflow on slow connections

TCPIP_IPERF_TX_WAIT_TMO Macro

File

iperf_config.h

C

```
#define TCPIP_IPERF_TX_WAIT_TMO 100
```

Description

timeout to wait for TX channel to be ready to transmit a new packet; ms depends on the channel bandwidth

TCPIP_IPERF_MAX_INSTANCES Macro

File

iperf_config.h

C

```
#define TCPIP_IPERF_MAX_INSTANCES 1
```

Description

maximum number of simultaneously running iperf instances

TCPIP_IPERF_TIMING_ERROR_MARGIN Macro

File

iperf_config.h

C

```
#define TCPIP_IPERF_TIMING_ERROR_MARGIN 0
```

Description

iperf timing error, ms. Accounts for timing uncertainty The higher the value, the more relaxed is the iperf timing Lower limit is 0

Building the Library

This section lists the files that are available in the Iperf module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (.h) and source (.c) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.
`/iperf.h`	Header file for the Iperf module.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/iperf.c`	Iperf implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Iperf module depends on the following modules:

- TCP/IP Stack Library

Files

Files

Name	Description
iperf_config.h	Iperf configuration file

Description

This section lists the source and header files used by the module.

iperf_config.h

Iperf configuration file

Macros

	Name	Description
	TCPIP_IPERF_MAX_INSTANCES	maximum number of simultaneously running iperf instances
	TCPIP_IPERF_RX_BUFFER_SIZE	Default size of the Iperf RX buffer The default value is 4KB. The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Bigger buffers will help obtain higher performance numbers
	TCPIP_IPERF_TIMING_ERROR_MARGIN	iperf timing error, ms. Accounts for timing uncertainty The higher the value, the more relaxed is the iperf timing Lower limit is 0
	TCPIP_IPERF_TX_BUFFER_SIZE	Default size of the Iperf TX buffer The default value is 4KB. The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Bigger buffers will help obtain higher performance numbers
	TCPIP_IPERF_TX_QUEUE_LIMIT	for Iperf UDP client, the limit to set to avoid memory allocation overflow on slow connections
	TCPIP_IPERF_TX_WAIT_TMO	timeout to wait for TX channel to be ready to transmit a new packet; ms depends on the channel bandwidth

Description

Iperf Configuration file

This file contains the Iperf configuration options

File Name

iperf_config.h

Company

Microchip Technology Inc.

IPv4 Module

This section describes the TCP/IP Stack Library IPv4 module.

Introduction

TCP/IP Stack Library IPv4 Module for Microchip Microcontrollers

This library provides the API of the IPv4 module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

IP is the workhorse protocol of the TCP/IP protocol suite. All TCP, UDP, and ICMP data gets transmitted as IP datagrams. IP provides an unreliable, connectionless datagram delivery service.

IP provides a best effort service. When something goes wrong, such as a router temporarily running out of buffers, IP has a simple error handling algorithm: throw away the datagram and try to send an ICMP message back to the source. Any required reliability must be provided by the upper layers (e.g., TCP).

The term connectionless means that IP does not maintain any state information about successive datagrams. Each datagram is handled independently from all other datagrams. This also means that IP datagrams can get delivered out of order. If a source sends two consecutive datagrams (first A, and then B) to the same destination, each is routed independently and can take different routes, with B arriving before A.

Using the Library

This topic describes the basic architecture of the IP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `ipv4.h`

The interface to the IPv4 TCP/IP Stack library is defined in the ipv4.h header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the IPv4 TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

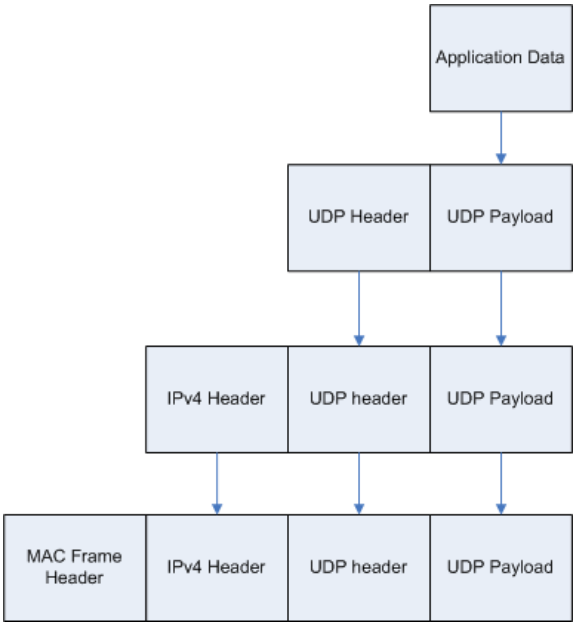
This library provides the API of the IPv4 TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

This module provides software abstraction of the IPv4 module existent in any TCP/IP Stack implementation.

The typical usage of the IP layer in a TCP/IP stack for transmitting a packet is shown in the following diagram:

IPv4 Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the IPv4 module.

Library Interface Section	Description
Functions	Routines for configuring IPv4
Data Types and Constants	This section provides various definitions describing this API

How the Library Works

This topic describes how the IP TCP/IP Stack Library works.

Description

On the transmit side, the IP layer is responsible for routing and forwarding the IP datagrams to the required destination host. To do that, the IP layer has internal functionality that allows the selection of the proper network interface and destination hardware address of a packet.

Based on the destination of a packet/datagram, the IP layer will send the packet to either a host on the network or to a gateway if either is available. To accomplish this, the IP module works closely with the ARP layer from which it requests the destination hardware addresses for

outgoing packets. If the destination address of a packet is not known, the IP module will request an ARP probe and will wait for the ARP module reply. Whenever ARP signals that the solicited address is available, the IP module will send the packet over the network. If a time-out is reported (no such host could be found on that particular network), the packet is silently discarded.

On the receive side, the IP module processes all the incoming IPv4 packets and performs a basic sanity check of the packet checksum and other attributes. If the packet check fails, the packet is simply discarded. Otherwise, it is dispatched for further processing to the appropriate module (UDP, TCP, ICMP, etc.).

The IP layer operation within the TCP/IP stack is transparent to the applications using the preferred socket API (MCHP UDP, TCP or BSD). This means that normally an application does not need to interface directly to the IP layer. However, the IP module exposes an API that can be used by the application for sending IP datagrams directly, bypassing the normal higher layer protocols (UDP, TCP).

Notes:

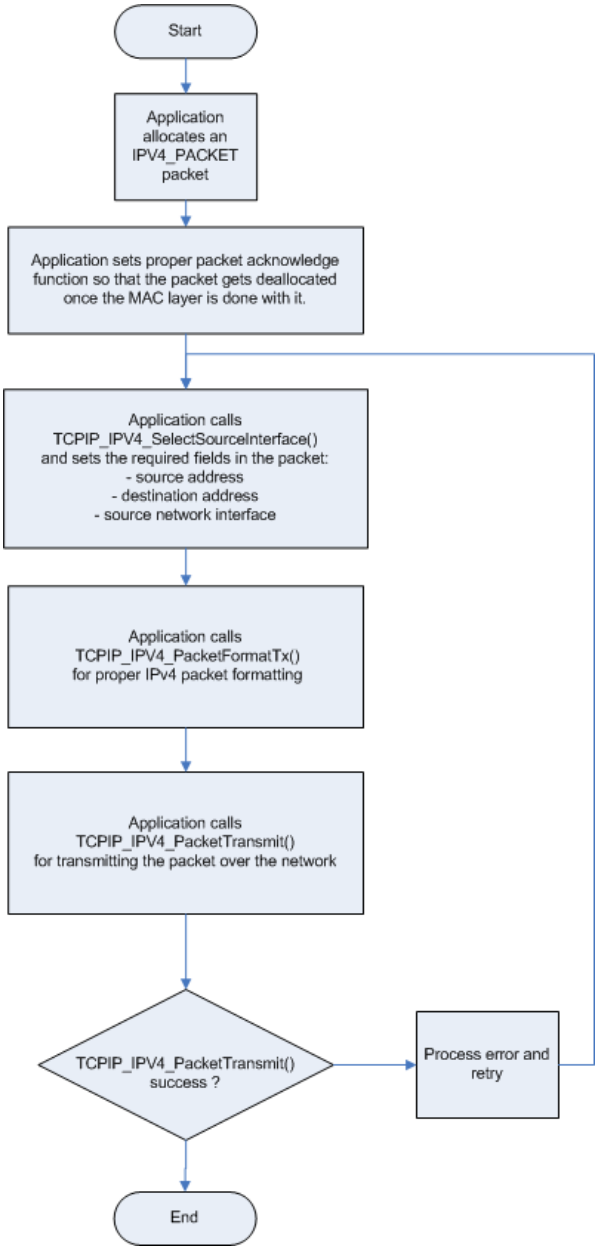
1. Currently, it is not possible for an application to intercept the incoming IP datagrams and interfere with the default IP processing. However, this may change in the future and support can be added for the receive side.
2. For a detailed description of the IP layer functionality, please consult RFC 791, RFC 1122, RFC 2474, etc.
3. This implementation does not support packet fragmentation (neither on transmit or receive).

Core Functionality

This topic describes the core functionality of the IP TCP/IP Stack Library.

Description

IP Connection Flow



To transmit an IP packet over the network, the application has to prepare an IPV4_PACKET data structure and properly format it so that the recipients of this packet (the IP and the MAC layers) can process the packet.

The TCP/IP Stack modules allocate the IPV4_PACKET data structures dynamically using the TCP/IP Stack private heap using packet allocation functions.

Note: Currently, the packet and heap allocation functions are not exposed as public APIs as they are meant for stack's private usage. This may change in the future.

The IPV4_PACKET packet can be allocated statically by the application.

When the IPV4_PACKET is constructed an important thing to remember is to fill in the appropriate packet acknowledge function. Once the MAC layer/driver is done with processing the packet (successfully or not) it will call the IPV4_PACKET acknowledge function, indicating that the packet is now available (and could be freed or reused).

The next step is to fill in the packet source and destination addresses, as well as the outgoing interface (please keep in mind that the TCP/IP stack supports multiple network interfaces).

This could be done with a call to TCPIP_IPV4_SelectSourceInterface or the application can select all these addresses manually.

All that's left before handing the packet is to call TCPIP_IPV4_PacketFormatTx. This call fills in the supplied packet with the IP header structure and updates the IP checksum. Please note that this function expects a properly formatted packet, with the source and destination addresses filled in.

After this, the application can call TCPIP_IPV4_PacketTransmit. This function does some basic packet checks and expects that the packet has a valid network interface selected otherwise the call will fail.

After further formatting (with the MAC required headers), the IP layer will try to send the packet over the network. If the destination hardware

address is known (the ARP resolve call succeeds) the packet is handed over to the MAC. Otherwise, the IP will insert the packet into a queue, waiting for the ARP resolution. When ARP signals that the request is done (either success or timeout) the IP layer intercepts the signal and removes the packet from the wait queue: either hands it over to the MAC for transmission or it discards it if the ARP resolution failed.

The MAC layer/driver in its turn can transmit it immediately, queue the packet for transmission or discard it (if, for example the network link is down). Regardless, once the packet is processed, the packet acknowledge function is called with an updated acknowledge result. This informs the owner of the packet about the result of the operation and also that the packet is no longer in traffic.

Note: Transmission of chained packets is not supported. Each packet has to be transmitted individually.

Configuring the Library

Macros

	Name	Description
	TCPIP_IPV4_FRAGMENT_MAX_NUMBER	Upper limit for the number of fragments in a fragmented stream This also imposes a limit on how large a packet could be depending on the MTU and fragmentations that occur If more fragments received for a certain stream, the stream will be discarded
	TCPIP_IPV4_FRAGMENT_MAX_STREAMS	Upper limit for the number of fragmented streams that could be handled simultaneously
	TCPIP_IPV4_FRAGMENT_TIMEOUT	Initial fragment timeout, in seconds Timer Lower Bound (RFC 791) The RFC states this value to be 15 seconds
	TCPIP_IPV4_FRAGMENTATION	If 1, enable IPv4 fragmentation processing, RX + TX If 0 (default), IPv4 fragmentation not supported
	TCPIP_IPV4_TASK_TICK_RATE	The IP task processing rate: number of milliseconds to generate an IP tick. This is the tick that advances the IP state machine. The default value is around 30 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE. The IP state machine tick is used only when fragmentation is enabled.

Description

The configuration of the IPv4 TCP/IP Stack is based on the file `system_config.h` (which may include `ip_config.h`).

This header file contains the configuration selection for the IPv4 TCP/IP Stack. Based on the selections made, the IPv4 TCP/IP Stack may support the selected features. Currently, there are no configuration parameters for the IP module.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_IPV4_FRAGMENT_MAX_NUMBER Macro

File

ip_config.h

C

```
#define TCPIP_IPV4_FRAGMENT_MAX_NUMBER 4
```

Description

Upper limit for the number of fragments in a fragmented stream This also imposes a limit on how large a packet could be depending on the MTU and fragmentations that occur If more fragments received for a certain stream, the stream will be discarded

TCPIP_IPV4_FRAGMENT_MAX_STREAMS Macro

File

ip_config.h

C

```
#define TCPIP_IPV4_FRAGMENT_MAX_STREAMS 3
```

Description

Upper limit for the number of fragmented streams that could be handled simultaneously

TCPIP_IPV4_FRAGMENT_TIMEOUT Macro

File

ip_config.h

C

```
#define TCPIP_IPV4_FRAGMENT_TIMEOUT 15
```

Description

Initial fragment timeout, in seconds Timer Lower Bound (RFC 791) The RFC states this value to be 15 seconds

TCPIP_IPV4_FRAGMENTATION Macro

File

ip_config.h

C

```
#define TCPIP_IPV4_FRAGMENTATION 1
```

Description

If 1, enable IPv4 fragmentation processing, RX + TX If 0 (default), IPv4 fragmentation not supported

TCPIP_IPV4_TASK_TICK_RATE Macro

File

ip_config.h

C

```
#define TCPIP_IPV4_TASK_TICK_RATE 37
```

Description

The IP task processing rate: number of milliseconds to generate an IP tick. This is the tick that advances the IP state machine. The default value is around 30 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE. The IP state machine tick is used only when fragmentation is enabled.

Building the Library

This section lists the files that are available in the IPv4 module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/ipv4.c`	IPv4 implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The IPv4 module depends on the following modules:

- TCP/IP Stack Library

Library Interface

a) Functions

	Name	Description
	TCPIP_IPV4_PacketGetDestAddress	Returns the IPv4 destination address associated with a TCPIP_MAC_PACKET
	TCPIP_IPV4_PacketGetSourceAddress	Returns the IPv4 source address associated with a TCPIP_MAC_PACKET
	TCPIP_IPV4_PacketTransmit	Transmits an IPV4 packet over the network.
	TCPIP_IPV4_SelectSourceInterface	Selects a source address and an interface based on the IPv4 destination address
	TCPIP_IPV4_Task	Standard TCP/IP stack module task function.
	TCPIP_IPV4_PacketFilterClear	Clears the IPV4 packet filters
	TCPIP_IPV4_PacketFilterSet	Sets the IPV4 packet filters
	TCPIP_IPV4_MacPacketTransmit	Transmits a fully constructed MAC packet over the network.
	TCPIP_IPV4_PacketFormatTx	Formats an IPV4 packet and makes it ready for transmission.

b) Data Types and Constants

	Name	Description
	IPV4_HEADER	This is type IPV4_HEADER.
	IPV4_HEADER_TYPE	List of supported protocols.
	IPV4_PACKET	IPv4 packet structure.
	TCPIP_IPV4_FILTER_TYPE	List of supported IPv4 packet filters.
	TCPIP_IPV4_MODULE_CONFIG	This is type TCPIP_IPV4_MODULE_CONFIG.
	IPV4_FRAGMENT_INFO	Structure of an IPv4 header.

Description

This section describes the Application Programming Interface (API) functions of the IPv4 module.

Refer to each section for a detailed description.

a) Functions

TCPIP_IPV4_PacketGetDestAddress Function

Returns the IPv4 destination address associated with a TCPIP_MAC_PACKET

File

ipv4.h

C

```
__inline__ const IPV4_ADDR* TCPIP_IPV4_PacketGetDestAddress(TCPIP_MAC_PACKET* pPkt);
```

Returns

- A valid pointer to an IPV4_ADDR - if it succeeds
- 0 - if call failed

Description

The function will return a pointer to where the IPv4 destination address is located in the TCPIP_MAC_PACKET. The TCPIP_MAC_PACKET is

supposed to be a valid IPv4 packet that has destination address properly set.

Remarks

This function is primarily meant for RX packets.

Preconditions

pPkt - valid IPv4 packet, pNetLayer filed properly set.

Parameters

Parameters	Description
pPkt	packet to query

Function

const IPV4_ADDR* TCPIP_IPV4_PacketGetDestAddress(TCPIP_MAC_PACKET* pPkt);

TCPIP_IPV4_PacketGetSourceAddress Function

Returns the IPv4 source address associated with a TCPIP_MAC_PACKET

File

ipv4.h

C

```
__inline__ const IPV4_ADDR* TCPIP_IPV4_PacketGetSourceAddress(TCPIP_MAC_PACKET* pPkt);
```

Returns

- A valid pointer to an IPV4_ADDR - if it succeeds
- 0 - if call failed

Description

The function will return a pointer to where the IPv4 source address is located in the TCPIP_MAC_PACKET. The TCPIP_MAC_PACKET is supposed to be a valid IPv4 packet that has properly source address set.

Remarks

This function is primarily meant for RX packets.

Preconditions

pPkt - valid IPv4 packet, pNetLayer filed properly set

Parameters

Parameters	Description
pPkt	packet to query

Function

const IPV4_ADDR* TCPIP_IPV4_PacketGetSourceAddress(TCPIP_MAC_PACKET* pPkt);

TCPIP_IPV4_PacketTransmit Function

Transmits an IPV4 packet over the network.

File

ipv4.h

C

```
bool TCPIP_IPV4_PacketTransmit(IPV4_PACKET* pPkt);
```

Returns

- true - if the packet was handed to the MAC or is queued for transmission
- false - the packet cannot be transmitted (wrong interface, etc.)

Description

The IPv4 packet is sent to the MAC for transmission.

Remarks

Only single packets can be transmitted. Chained packets are not supported for now.

Preconditions

pPkt should have been properly formatted with TCPIP_IPV4_PacketFormatTx(). The packet interface should be updated.

Parameters

Parameters	Description
pPkt	the packet to be transmitted

Function

bool TCPIP_IPV4_PacketTransmit(IPV4_PACKET* pPkt);

TCPIP_IPV4_SelectSourceInterface Function

Selects a source address and an interface based on the IPv4 destination address

File

ipv4.h

C

```
TCPIP_NET_HANDLE TCPIP_IPV4_SelectSourceInterface(TCPIP_NET_HANDLE netH, IPV4_ADDR* pDestAddress,
IPV4_ADDR* pSrcAddress, bool srcSet);
```

Returns

- A valid interface - if it succeeds and a valid source interface selected
- 0 - interface selection failed

Description

Updates the pSrcAddress and returns the needed interface, if successful:

- if srcSet == 1 and netH != 0, the function will not change anything
- if srcSet == 1 and netH == 0, the call will never fail it will use whatever value in pSrcAddress (even 0) and will try to come up with an appropriate interface
- if srcSet == 0 and netH == 0, it will use the destination address
- if srcSet == 0 and netH != 0, it will use the address of that interface

Remarks

None.

Preconditions

netH has to be valid (if non-0).

Parameters

Parameters	Description
netH	network interface handle
pDestAddress	pointer to destination address
pSrcAddress	pointer to source address
srcSet	boolean; true if address pointed by pSrcAddress is valid

Function

TCPIP_NET_HANDLE TCPIP_IPV4_SelectSourceInterface(TCPIP_NET_HANDLE netH,
IPV4_ADDR* pDestAddress, IPV4_ADDR* pSrcAddress, bool srcSet)

TCPIP_IPV4_Task Function

Standard TCP/IP stack module task function.

File

ipv4.h

C

```
void TCPIP_IPV4_Task();
```

Returns

None.

Description

This function performs IPv4 module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The IPv4 module should have been initialized.

Function

void TCPIP_IPV4_Task(void)

TCPIP_IPV4_PacketFilterClear Function

Clears the IPV4 packet filters

File

ipv4.h

C

```
TCPIP_IPV4_FILTER_TYPE TCPIP_IPV4_PacketFilterClear(TCPIP_IPV4_FILTER_TYPE filtType);
```

Returns

- the current value of the IPV4 packet filters after this mask was applied.

Description

The function will clear the IPv4 packet filters. The filters that are present in the mask will be cleared. Other filters won't be touched.

Remarks

None.

Preconditions

filtType - valid IPv4 filter IPv4 properly initialiazed

Parameters

Parameters	Description
filtType	packet filter mask to clear

Function

TCPIP_IPV4_FILTER_TYPE TCPIP_IPV4_PacketFilterClear(TCPIP_IPV4_FILTER_TYPE filtType);

TCPIP_IPV4_PacketFilterSet Function

Sets the IPV4 packet filters

File

ipv4.h

C

```
TCPIP_IPV4_FILTER_TYPE TCPIP_IPV4_PacketFilterSet(TCPIP_IPV4_FILTER_TYPE filtType);
```

Returns

- the current value of the IPV4 packet filters after this mask was applied.

Description

The function will set the IPv4 packet filters. The filters that are present in the mask will be set. Other filters won't be touched.

Remarks

None.

Preconditions

filtType - valid IPv4 filter IPv4 properly initialiazed

Parameters

Parameters	Description
filtType	packet filter mask to set

Function

TCPIP_IPV4_FILTER_TYPE TCPIP_IPV4_PacketFilterSet(TCPIP_IPV4_FILTER_TYPE filtType);

TCPIP_IPV4_MacPacketTransmit Function

Transmits a fully constructed MAC packet over the network.

File

ipv4.h

C

```
bool TCPIP_IPV4_MacPacketTransmit(TCPIP_MAC_PACKET* pPkt, TCPIP_NET_HANDLE hNet, IPV4_ADDR* pDestAddress);
```

Returns

- true - if the packet was handed to the MAC or queued internally
- false - the packet cannot be transmitted (wrong interface, etc.)

Description

The function is a helper to transmit a fully constructed MAC packet over an interface. The MAC packet is supposed to have all the fields properly constructed (including the TCPIP_MAC_ETHERNET_HEADER fields). The MAC packet does not need ARP resolution since the DestMACAddr is already known. The packet acknowledge function should be properly set (it will be called by the MAC driver once the packet is transmitted).

Remarks

Only single packets can be transmitted. Chained packets are not supported for now.

The packet could be composed of multiple fragments. If the transmission fails, some fragments could have been transmitted while other failed. The flag TCPIP_MAC_PKT_FLAG_QUEUED could be used to determine which fragments didn't make it through.

Preconditions

pPkt should have been properly formatted and constructed into a valid MAC packet that can go out over an interface pDestAddress - valid pointer

Parameters

Parameters	Description
pPkt	the packet to be transmitted
hNet	interface over which the packet is to be transmitted
pDestAddress	the IPv4 packet destination address It is used only for routing purposes, to decide if the packet is routed internally or sent to the network interface

Function

bool TCPIP_IPV4_MacPacketTransmit(TCPIP_MAC_PACKET* pPkt, TCPIP_NET_HANDLE hNet, IPV4_ADDR* pDestAddress);

TCPIP_IPV4_PacketFormatTx Function

Formats an IPV4 packet and makes it ready for transmission.

File

ipv4.h

C

```
void TCPIP_IPV4_PacketFormatTx(IPV4_PACKET* pPkt, uint8_t protocol, uint16_t ipLoadLen, uint8_t ttl);
```

Returns

None

Description

The necessary fields are set into the IPv4 packet.

Remarks

The segments should be properly updated with the right number of bytes (segLen). The IP payload length (ipLoadLen) is added only to the 1st segment of the packet! Other segments (for packets having multiple packets) are not touched.

Preconditions

Properly allocated pPkt. The source and destination addresses should be updated in the packet.

Parameters

Parameters	Description
pPkt	the packet to be formatted
protocol	the protocol associated with the packet
ipLoadLen	the IPv4 packet payload length
ttl	the Time to Live of the packet If 0, then the default value will be used Otherwise the passed in parameter

Function

TCPIP_IPV4_PacketFormatTx(IPV4_PACKET* pPkt, uint8_t protocol, uint16_t ipLoadLen, uint8_t ttl);

b) Data Types and Constants

IPV4_HEADER Structure

File

ipv4.h

C

```
typedef struct {
  uint8_t VersionIHL;
  uint8_t TypeOfService;
  uint16_t TotalLength;
  uint16_t Identification;
  IPV4_FRAGMENT_INFO FragmentInfo;
  uint8_t TimeToLive;
  uint8_t Protocol;
  uint16_t HeaderChecksum;
  IPV4_ADDR SourceAddress;
  IPV4_ADDR DestAddress;
} IPV4_HEADER;
```

Description

This is type IPV4_HEADER.

IPV4_HEADER_TYPE Enumeration

List of supported protocols.

File

ipv4.h

C

```
typedef enum {
  IP_PROT_ICMP = (1u),
```

```
  IP_PROT_IGMP = (2u),
  IP_PROT_TCP = (6u),
  IP_PROT_UDP = (17u)
} IPV4_HEADER_TYPE;
```

Description

IPv4 supported protocols

This is the list of the protocols that are supported by this IPv4 implementation.

Remarks

None.

IPV4_PACKET Structure

IPv4 packet structure.

File

ipv4.h

C

```
typedef struct {
  TCPIP_MAC_PACKET macPkt;
  IPV4_ADDR srcAddress;
  IPV4_ADDR destAddress;
  TCPIP_NET_HANDLE netIfH;
  IPV4_ADDR arpTarget;
} IPV4_PACKET;
```

Members

Members	Description
TCPIP_MAC_PACKET macPkt;	standard MAC packet header safe cast to TCPIP_MAC_PACKET
IPV4_ADDR srcAddress;	packet source
IPV4_ADDR destAddress;	packet destination
TCPIP_NET_HANDLE netIfH;	packet interface
IPV4_ADDR arpTarget;	ARP resolution target

Description

IPv4 packet structure definition

This is the structure of an IPv4 packet for transmission over the network.

Remarks

None.

TCPIP_IPV4_FILTER_TYPE Enumeration

List of supported IPv4 packet filters.

File

ipv4.h

C

```
typedef enum {
  TCPIP_IPV4_FILTER_NONE = 0x00,
  TCPIP_IPV4_FILTER_UNICAST = 0x01,
  TCPIP_IPV4_FILTER_BROADCAST = 0x02,
  TCPIP_IPV4_FILTER_MULTICAST = 0x04
} TCPIP_IPV4_FILTER_TYPE;
```

Members

Members	Description
TCPIP_IPV4_FILTER_NONE = 0x00	no packet filter active. All packets are accepted
TCPIP_IPV4_FILTER_UNICAST = 0x01	unicast packets will be filtered out
TCPIP_IPV4_FILTER_BROADCAST = 0x02	broadcast packets will be filtered out

TCPIP_IPV4_FILTER_MULTICAST = 0x04	multicast packets will be filtered out

Description

IPv4 packet filters

This is the list of the packet filters that are supported by this IPv4 implementation. There are 3 types of IPv4 packets currently supported:

- unicast
- broadcast
- multicast

An IPV4 packet is accepted if the filter corresponding to the packet type is not set

Remarks

Multiple filters can be set

If no filter is set, all packets are accepted this is the default case.

TCPIP_IPV4_MODULE_CONFIG Structure

File

ipv4.h

C

```
typedef struct {
} TCPIP_IPV4_MODULE_CONFIG;
```

Description

This is type TCPIP_IPV4_MODULE_CONFIG.

IPV4_FRAGMENT_INFO Union

Structure of an IPv4 header.

File

ipv4.h

C

```
typedef union {
  uint16_t val;
  struct {
    uint16_t fragOffset : 13;
    uint16_t MF : 1;
    uint16_t DF : 1;
    uint16_t reserved : 1;
  }
} IPV4_FRAGMENT_INFO;
```

Members

Members	Description
uint16_t fragOffset : 13;	fragment offset
uint16_t MF : 1;	more fragments
uint16_t DF : 1;	don't fragment

Description

IPv4 packet header definition

This is the structure of an IPv4 packet header.

Remarks

None.

Files

Files

Name	Description
ipv4.h	IPv4 definitions for the Microchip TCP/IP Stack.
ip_config.h	Internet Protocol (IP) Configuration file.

Description

This section lists the source and header files used by the library.

ipv4.h

IPv4 definitions for the Microchip TCP/IP Stack.

Enumerations

	Name	Description
	IPV4_HEADER_TYPE	List of supported protocols.
	TCPIP_IPV4_FILTER_TYPE	List of supported IPv4 packet filters.

Functions

	Name	Description
	TCPIP_IPV4_MacPacketTransmit	Transmits a fully constructed MAC packet over the network.
	TCPIP_IPV4_PacketFilterClear	Clears the IPV4 packet filters
	TCPIP_IPV4_PacketFilterSet	Sets the IPV4 packet filters
	TCPIP_IPV4_PacketFormatTx	Formats an IPV4 packet and makes it ready for transmission.
	TCPIP_IPV4_PacketGetDestAddress	Returns the IPv4 destination address associated with a TCPIP_MAC_PACKET
	TCPIP_IPV4_PacketGetSourceAddress	Returns the IPv4 source address associated with a TCPIP_MAC_PACKET
	TCPIP_IPV4_PacketTransmit	Transmits an IPV4 packet over the network.
	TCPIP_IPV4_SelectSourceInterface	Selects a source address and an interface based on the IPv4 destination address
	TCPIP_IPV4_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	IPV4_HEADER	This is type IPV4_HEADER.
	IPV4_PACKET	IPv4 packet structure.
	TCPIP_IPV4_MODULE_CONFIG	This is type TCPIP_IPV4_MODULE_CONFIG.

Unions

	Name	Description
	IPV4_FRAGMENT_INFO	Structure of an IPv4 header.

Description

IPv4 Header File

IP is the workhorse protocol of the TCP/IP protocol suite. All TCP, UDP, and ICMP data gets transmitted as IP datagrams. IP provides an unreliable, connectionless datagram delivery service.

File Name

ipv4.h

Company

Microchip Technology Inc.

ip_config.h

Internet Protocol (IP) Configuration file.

Macros

	Name	Description
	TCPIP_IPV4_FRAGMENT_MAX_NUMBER	Upper limit for the number of fragments in a fragmented stream This also imposes a limit on how large a packet could be depending on the MTU and fragmentations that occur If more fragments received for a certain stream, the stream will be discarded
	TCPIP_IPV4_FRAGMENT_MAX_STREAMS	Upper limit for the number of fragmented streams that could be handled simultaneously
	TCPIP_IPV4_FRAGMENT_TIMEOUT	Initial fragment timeout, in seconds Timer Lower Bound (RFC 791) The RFC states this value to be 15 seconds
	TCPIP_IPV4_FRAGMENTATION	If 1, enable IPv4 fragmentation processing, RX + TX If 0 (default), IPv4 fragmentation not supported
	TCPIP_IPV4_TASK_TICK_RATE	The IP task processing rate: number of milliseconds to generate an IP tick. This is the tick that advances the IP state machine. The default value is around 30 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE. The IP state machine tick is used only when fragmentation is enabled.

Description

This file contains the IP module configuration options.

File Name

ip_config.h

Company

Microchip Technology Inc.

IPv6 Module

This section describes the TCP/IP Stack Library IPv6 module.

Introduction

TCP/IP Stack Library IPv6 Module for Microchip Microcontrollers

This library provides the API of the IPv6 module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

IPv6 is the workhorse protocol of the TCP/IP protocol suite. All TCP, UDP, ICMP, and IGMP data gets transmitted as IP datagrams. IP provides an reliable, connectionless datagram delivery service.

IPv6 provides a best effort service. When something goes wrong, such as a router temporarily running out of buffers, IPv6 has a simple error handling algorithm: throw away the datagram and try to send an ICMP message back to the source. Any required reliability must be provided by the upper layers (e.g., TCP).

The term connectionless means that IPv6 does not maintain any state information about successive datagrams. Each datagram is handled independently from all other datagrams. This also means that IPv6 datagrams can get delivered out of order. If a source sends two consecutive datagrams (first A, and then B) to the same destination, each is routed independently and can take different routes, with B arriving before A.

Using the Library

This topic describes the basic architecture of the IPv6 TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `ipv6.h`

The interface to the IPv6 TCP/IP Stack library is defined in the `ipv6.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the IPv6 TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

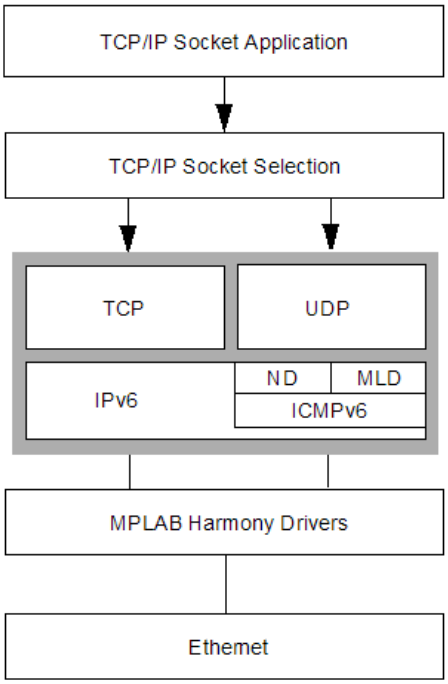
Abstraction Model

This library provides the API of the IP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

IP Software Abstraction Block Diagram

This module provides software abstraction of the IP module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the IPv6 module.

Library Interface Section	Description
Functions	Routines for configuring IPv6
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_IPV6_DEFAULT_ALLOCATION_BLOCK_SIZE	Sets the minimum allocation unit for the payload size
	TCPIP_IPV6_DEFAULT_BASE_REACHABLE_TIME	Default 30 seconds , Router advertisement reachable time
	TCPIP_IPV6_DEFAULT_CUR_HOP_LIMIT	IPv4 Time-to-Live parameter
	TCPIP_IPV6_DEFAULT_LINK_MTU	Default Maximum Transmission Unit
	TCPIP_IPV6_DEFAULT_RETRANSMIT_TIME	1 second, Process the router advertisement's retransmission time
	TCPIP_IPV6_FRAGMENT_PKT_TIMEOUT	Fragmentation packet time-out value. default value is 60 .
	TCPIP_IPV6_INIT_TASK_PROCESS_RATE	IPv6 initialize task processing rate, milliseconds The default value is 32 milliseconds.
	TCPIP_IPV6_MINIMUM_LINK_MTU	Sets the lower bounds of the Maximum Transmission Unit
	TCPIP_IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT	10 minutes
	TCPIP_IPV6_QUEUE_MCAST_PACKET_LIMIT	This option defines the maximum number of multicast queued IPv6 If an additional packet is queued, the oldest packet in the queue will be removed.
	TCPIP_IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT	This option defines the maximum number of queued packets per remote. If an additional packet needs to be queued, the oldest packet in the queue will be removed.

	TCPIP_IPV6_QUEUED_MCAST_PACKET_TIMEOUT	This option defines the number of seconds an IPv6 multicast packet will remain in the queue before being timed out
	TCPIP_IPV6_RX_FRAGMENTED_BUFFER_SIZE	RX fragmented buffer size should be equal to the total original packet size of ICMPv6 ECHO request packets . ex - Let Transmit ICMPv6 Echo request packet whose original packet size is 1500byte from the Global address of HOST1 to the global address of HOST2 and if the packet is going to be fragmented then packet will be broken more than packets. Each packet will have IPv6 header(40 bytes)+ Fragmentation header (8 bytes) + ICMPv6 Echo request header(8 bytes) • Payload (data packet).PING6(1500=40+8+8+1452 bytes). Here data packet size is 1452. If the data packet size is getting changed then this... more
	TCPIP_IPV6_TASK_PROCESS_RATE	IPv6 task processing rate, milliseconds The default value is 1000 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_IPV6_ULA_NTP_ACCESS_TMO	NTP access time-out for the IPv6 ULA address generation, ms
	TCPIP_IPV6_ULA_NTP_VALID_WINDOW	the NTP time stamp validity window, ms if a stamp was obtained outside this interval from the moment of the request a new request will be issued

Description

The configuration of the IPv6 TCP/IP Stack is based on the file `system_config.h`.

This header file contains the configuration selection for the IPv6 TCP/IP Stack. Based on the selections made, the IPv6 TCP/IP Stack may support the support selected features. These configuration settings will apply to all instances of the IPv6 TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_IPV6_DEFAULT_ALLOCATION_BLOCK_SIZE Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_DEFAULT_ALLOCATION_BLOCK_SIZE (64u)
```

Description

Sets the minimum allocation unit for the payload size

TCPIP_IPV6_DEFAULT_BASE_REACHABLE_TIME Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_DEFAULT_BASE_REACHABLE_TIME 30u
```

Description

Default 30 seconds , Router advertisement reachable time

TCPIP_IPV6_DEFAULT_CUR_HOP_LIMIT Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_DEFAULT_CUR_HOP_LIMIT 64u
```

Description

IPv4 Time-to-Live parameter

TCPIP_IPV6_DEFAULT_LINK_MTU Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_DEFAULT_LINK_MTU 1500u
```

Description

Default Maximum Transmission Unit

TCPIP_IPV6_DEFAULT_RETRANSMIT_TIME Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_DEFAULT_RETRANSMIT_TIME 1000u
```

Description

1 second, Process the router advertisement's retransmission time

TCPIP_IPV6_FRAGMENT_PKT_TIMEOUT Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_FRAGMENT_PKT_TIMEOUT 60
```

Description

Fragmentation packet time-out value. default value is 60 .

TCPIP_IPV6_INIT_TASK_PROCESS_RATE Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_INIT_TASK_PROCESS_RATE (32)
```

Description

IPv6 initialize task processing rate, milliseconds The default value is 32 milliseconds.

TCPIP_IPV6_MINIMUM_LINK_MTU Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_MINIMUM_LINK_MTU 1280u
```

Description

Sets the lower bounds of the Maximum Transmission Unit

TCPIP_IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT 600ul           // 10 minutes
```

Description

10 minutes

TCPIP_IPV6_QUEUE_MCAST_PACKET_LIMIT Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_QUEUE_MCAST_PACKET_LIMIT 4
```

Description

This option defines the maximum number of multicast queued IPv6 If an additional packet is queued, the oldest packet in the queue will be removed.

TCPIP_IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT 1
```

Description

This option defines the maximum number of queued packets per remote. If an additional packet needs to be queued, the oldest packet in the queue will be removed.

TCPIP_IPV6_QUEUED_MCAST_PACKET_TIMEOUT Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_QUEUED_MCAST_PACKET_TIMEOUT 10u
```

Description

This option defines the number of seconds an IPv6 multicast packet will remain in the queue before being timed out

TCPIP_IPV6_RX_FRAGMENTED_BUFFER_SIZE Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_RX_FRAGMENTED_BUFFER_SIZE 1514
```

Description

RX fragmented buffer size should be equal to the total original packet size of ICMPv6 ECHO request packets . ex - Let Transmit ICMPv6 Echo request packet whose original packet size is 1500byte from the Global address of HOST1 to the global address of HOST2 and if the packet is going to be fragmented then packet will be broken more than packets. Each packet will have IPv6 header(40 bytes)+ Fragmentation header (8

bytes) + ICMPv6 Echo request header(8 bytes)

- Payload (data packet).PING6(1500=40+8+8+1452 bytes). Here data packet size is 1452. If the data packet size is getting changed

then this following macro should be rectified to get proper ICMPv6 ECHO response. This is the Maximum RX fragmented buffer size.

TCPIP_IPV6_TASK_PROCESS_RATE Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_TASK_PROCESS_RATE (1000)
```

Description

IPv6 task processing rate, milliseconds The default value is 1000 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_IPV6_ULA_NTP_ACCESS_TMO Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_ULA_NTP_ACCESS_TMO (12000)
```

Description

NTP access time-out for the IPv6 ULA address generation, ms

TCPIP_IPV6_ULA_NTP_VALID_WINDOW Macro

File

ipv6_config.h

C

```
#define TCPIP_IPV6_ULA_NTP_VALID_WINDOW (1000)
```

Description

the NTP time stamp validity window, ms if a stamp was obtained outside this interval from the moment of the request a new request will be issued

Building the Library

This section lists the files that are available in the IPv6 module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/ipv6.c	IPv6 implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The IPv6 module depends on the following modules:

- TCP/IP Stack Library
- NDP Module
- SNTP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_IPV6_AddressUnicastRemove	Removed a configured unicast address from an interface.
	TCPIP_IPV6_ArrayGet	Reads the next byte of data from the specified MAC.
	TCPIP_IPV6_ArrayPutHelper	Helper function to write data to a packet.
	TCPIP_IPV6_DASSourceAddressSelect	Determines the appropriate source address for a given destination address.
	TCPIP_IPV6_DestAddressGet	Gets the destination address for a IPv6 packet.
	TCPIP_IPV6_DestAddressSet	Sets the destination address for a IPv6 packet.
	TCPIP_IPV6_Flush	Flushes a IP TX packet.
	TCPIP_IPV6_Get	Reads the next byte of data from the specified MAC.
	TCPIP_IPV6_HandlerDeregister	Deregisters an IPv6 event handler callback function.
	TCPIP_IPV6_HandlerRegister	Registers an IPv6 event handler callback function.
	TCPIP_IPV6_InterfaceIsReady	Determines if an interface is ready for IPv6 transactions.
	TCPIP_IPV6_MulticastListenerAdd	Adds a multicast listener to an interface.
	TCPIP_IPV6_MulticastListenerRemove	Removes a multicast listener from a given interface.
	TCPIP_IPV6_PacketFree	Frees a TCP/IP Packet structure from dynamic memory.
	TCPIP_IPV6_PayloadSet	Allocates a segment on the end of a packet segment chain and uses it to address prebuffered data.
	TCPIP_IPV6_Put	Writes a character of data to a packet.
	TCPIP_IPV6_SourceAddressGet	Gets the source address for an IPv6 packet.
	TCPIP_IPV6_SourceAddressSet	Sets the source address for a IPv6 packet.
	TCPIP_IPV6_TxIsPutReady	Determines whether a TX packet can be written to.
	TCPIP_IPV6_TxPacketAllocate	Dynamically allocates a packet for transmitting IP protocol data.
	TCPIP_IPV6_UniqueLocalUnicastAddressAdd	Adds a Unique Local Unicast Address (ULA) to a specified interface.
	TCPIP_IPV6_UnicastAddressAdd	Adds a unicast address to a specified interface
	TCPIP_IPV6_RouterAddressAdd	Adds a new router address to a specified interface.
	TCPIP_IPV6_DefaultRouterDelete	Deletes the current router list for a specified interface
	TCPIP_IPV6_DefaultRouterGet	Returns the current router address for a specified interface.
	TCPIP_IPV6_Task	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	IP_VERSION_4	Using IPv4
	IP_VERSION_6	Using IPv6
	IPV6_DATA_DYNAMIC_BUFFER	Data to transmit is allocated in dynamically allocated RAM
	IPV6_DATA_NETWORK_FIFO	Data to transmit is stored in the Network Controller's FIFOs
	IPV6_DATA_NONE	The data segment is unused

	IPV6_DATA_PIC_RAM	Data to transmit is stored in PIC RAM
	IPV6_HEADER_OFFSET_DEST_ADDR	Header offset for destination address
	IPV6_HEADER_OFFSET_NEXT_HEADER	Header offset for next header
	IPV6_HEADER_OFFSET_PAYLOAD_LENGTH	Header offset for payload length
	IPV6_HEADER_OFFSET_SOURCE_ADDR	Header offset for source address
	IPV6_NO_UPPER_LAYER_CHECKSUM	Value flag for no upper layer checksum
	IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM	IPv6 Type-length-value type code for the Hop-by-hop "Jumbo-gram Payload" option
	IPV6_TLV_HBHO_ROUTER_ALERT	IPv6 Type-length-value type code for the Hop-by-hop "Router Alert" option
	IPV6_TLV_PAD_1	IPv6 Type-length-value type code for the Pad 1 option
	IPV6_TLV_PAD_N	IPv6 Type-length-value type code for the Pad N option
	IPV6_TLV_UNREC_OPT_DISCARD_PP	IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message
	IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC	IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message is the destination address isn't a multicast address
	IPV6_TLV_UNREC_OPT_DISCARD_SILENT	IPv6 action code for the unrecognized option reaction to discard the packet silently
	IPV6_TLV_UNREC_OPT_SKIP_OPTION	IPv6 action code for the unrecognized option reaction to skip the option
	TCPIP_IPV6_PutArray	Writes data to a packet
	IPV6_ACTION	Provides a list of possible IPv6 actions.
	IPV6_ADDRESS_POLICY	Data structure for IPv6 address policy.
	IPV6_ADDRESS_PREFERENCE	Provides selection of public versus temporary addresses.
	IPV6_ADDRESS_TYPE	Data structure for IPv6 address types.
	IPV6_DATA_SEGMENT_HEADER	Data structure for IPv6 Data Segment header.
	_IPV6_DATA_SEGMENT_HEADER	Data structure for IPv6 Data Segment header.
	IPV6_EVENT_HANDLER	Clients can register a handler with the IPv6 service.
	IPV6_EVENT_TYPE	This enumeration is used to notify IPv6 client applications.
	IPV6_FRAGMENT_HEADER	Data structure for IPv6 fragment header.
	IPV6_HANDLE	Pointer to IPv6 object
	IPV6_HEADER	IPv6 packet header definition.
	IPV6_NEXT_HEADER_TYPE	Defines a list of IPv6 next header types.
	IPV6_PACKET	Packet structure/state tracking for IPv6 packets.
	_IPV6_PACKET	Packet structure/state tracking for IPv6 packets.
	IPV6_PACKET_ACK_FNC	Packet allocation and deallocation acknowledgment callback function.
	IPV6_RX_FRAGMENT_BUFFER	Data structure for IPv6 Received fragmented packet.
	_IPV6_RX_FRAGMENT_BUFFER	Data structure for IPv6 Received fragmented packet.
	IPV6_SEGMENT_TYPE	Provides an enumeration of IPv6 segment types.
	IPV6_TLV_OPTION_TYPE	Data structure for IPv6 TLV options.
	IPV6_ULA_FLAGS	Provides a list of possible ULA action flags.
	IPV6_ULA_RESULT	Provides a list of possible ULA results.
	TCPIP_IPV6_MODULE_CONFIG	Provides a place holder for IPv6 configuration.

Description

This section describes the Application Programming Interface (API) functions of the IPv6 module.

Refer to each section for a detailed description.

a) Functions

TCPIP_IPV6_AddressUnicastRemove Function

Removed a configured unicast address from an interface.

File

ipv6.h

C

```
void TCPIP_IPV6_AddressUnicastRemove(TCPIP_NET_HANDLE netH, IPV6_ADDR * address);
```

Returns

None.

Description

This function is used to remove a configured unicast address from an interface.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
netH	The interface to remove the address
address	The address

Function

void TCPIP_IPV6_AddressUnicastRemove(TCPIP_NET_HANDLE netH, IPV6_ADDR * address)

TCPIP_IPV6_ArrayGet Function

Reads the next byte of data from the specified MAC.

File

ipv6.h

C

```
uint8_t TCPIP_IPV6_ArrayGet(TCPIP_MAC_PACKET* pRxPkt, uint8_t * val, uint16_t len);
```

Returns

- > 0 - The number of bytes read
- 0 - No byte is available to read

Description

Reads a character of data from a packet.

Remarks

None

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
pRxPkt	The MAC RX packet to read data from
val	The buffer to store the data
len	The amount of data to read

Function

uint8_t TCPIP_IPV6_ArrayGet (TCPIP_MAC_PACKET* pRxPkt, uint8_t *val,
uint16_t len);

TCPIP_IPV6_ArrayPutHelper Function

Helper function to write data to a packet.

File

ipv6.h

C

```
unsigned short TCPIP_IPV6_ArrayPutHelper(IPV6_PACKET * pkt, const void * dataSource, uint8_t dataType,
unsigned short len);
```

Returns

unsigned short - The number of bytes of data written.

Description

This is a helper function for writing data to a packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
ptrPacket	The packet
dataSource	The address of the data on its medium
dataType	Descriptor of the data type (dynamic memory on PIC, in a network FIFO, in static PIC RAM)
len	Length of the data

Function

unsigned short TCPIP_IPV6_ArrayPutHelper (IPV6_PACKET * ptrPacket,

const void * dataSource, uint8_t dataType, unsigned short len)

TCPIP_IPV6_DASSourceAddressSelect Function

Determines the appropriate source address for a given destination address.

File

ipv6.h

C

```
IPV6_ADDR_STRUCT * TCPIP_IPV6_DASSourceAddressSelect(TCPIP_NET_HANDLE hNetIf, const IPV6_ADDR * dest,
IPV6_ADDR * requestedSource);
```

Returns

- On Success - Pointer to the selected source address
- On Failure - NULL

Description

The IPv6 policy table is used to select the destination address. The destination address selection algorithm takes a list of IPv6 addresses (gPolicyTable) and sorts the linked list. There are eight sorting rules. Starting with the last rule and working to the most important, using a stable sorting algorithm, will produce a sorted list most efficiently. The best average run time we'll get with a stable sort with O(1) memory usage is O(n^2), so we'll use an insertion sort. This will usually be most efficient for small lists (which should be the typical case). If a rule determines a result, then the remaining rules are not relevant and should be ignored.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
hNetIf	The given interface

dest	The destination address
requestedSource	A specified source

Function

IPV6_ADDR_STRUCT * TCPIP_IPV6_DASSourceAddressSelect (TCPIP_NET_HANDLE hNetIf,
IPV6_ADDR * dest, IPV6_ADDR * requestedSource)

TCPIP_IPV6_DestAddressGet Function

Gets the destination address for a IPv6 packet.

File

ipv6.h

C

```
IPV6_ADDR * TCPIP_IPV6_DestAddressGet(IPV6_PACKET * p);
```

Returns

IPV6_ADDR *

- On Success - Get a valid IPv6 Destination address
- On Failure - NULL

Description

This function is used to get the destination address for a IPv6 packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
p	pointer to IPv6 packet

Function

IPV6_ADDR * TCPIP_IPV6_DestAddressGet(IPV6_PACKET * p)

TCPIP_IPV6_DestAddressSet Function

Sets the destination address for a IPv6 packet.

File

ipv6.h

C

```
void TCPIP_IPV6_DestAddressSet(IPV6_PACKET * p, const IPV6_ADDR * addr);
```

Returns

None.

Description

This API is used to configure the destination address for a IPv6 packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
p	pointer to IPv6 packet
addr	destination address

Function

void TCPIP_IPV6_DestAddressSet(IPV6_PACKET * p, const IPV6_ADDR * addr)

TCPIP_IPV6_Flush Function

Flushes a IP TX packet.

File

ipv6.h

C

```
int TCPIP_IPV6_Flush(IPV6_PACKET * pkt);
```

Returns

- 1 - if the packet has been transmitted
- 0 - if the packet has been queued
- <0 - if the packet has been discarded for some error

Description

This function flushes a IP TX packet. Determines the link-layer address, if necessary and calculates the upper-layer checksum, if necessary.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
ptrPacket	The packet to flush

Function

int TCPIP_IPV6_Flush (IPV6_PACKET * ptrPacket)

TCPIP_IPV6_Get Function

Reads the next byte of data from the specified MAC.

File

ipv6.h

C

```
uint8_t TCPIP_IPV6_Get(TCPIP_MAC_PACKET* pRxPkt, uint8_t* pData);
```

Returns

- 1 - On Successful read
- 0 - No byte is available to read

Description

This function is used to read the next byte of data from a packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
pRxPkt	The MAC RX packet to read data from
pData	Byte data to be read

Function

uint8_t TCPIP_IPV6_Get (TCPIP_MAC_PACKET* pRxPkt, uint8_t* pData)

TCPIP_IPV6_HandlerDeregister Function

Deregisters an IPv6 event handler callback function.

File

ipv6.h

C

```
bool TCPIP_IPV6_HandlerDeregister(IPV6_HANDLE hIpv6);
```

Returns

- true - if deregister successful
- false - if deregister is not successful

Description

This function deregisters a previously registered IPv6 event handler callback function.

Remarks

None.

Preconditions

The IPv6 stack is initialized and the interface is up and configured.

Parameters

Parameters	Description
hIpv6	Handle to registered callback

Function

bool TCPIP_IPV6_HandlerDeregister(IPV6_HANDLE hIpv6)

TCPIP_IPV6_HandlerRegister Function

Registers an IPv6 event handler callback function.

File

ipv6.h

C

```
IPV6_HANDLE TCPIP_IPV6_HandlerRegister(TCPIP_NET_HANDLE hNet, IPV6_EVENT_HANDLER handler, const void*
hParam);
```

Returns

Handle to registered callback.

- On Success - Returns a valid handle
- On Failure - null handle

Description

This function is used to register a notification handler with the IPv6 module.

Remarks

None.

Preconditions

The IPv6 stack is initialized and the interface is up and configured.

Parameters

Parameters	Description
netH	Specifies interface to register on.
handler	Handler to be called for event.
hParam	The hParam is passed by the client and will be used by the IPv6 when the notification is made. It is used for per-thread content or if more modules, for example, share the same handler and need a way to differentiate the callback.

Function

IPV6_HANDLE TCPIP_IPV6_HandlerRegister(TCPIP_NET_HANDLE hNet,
IPV6_EVENT_HANDLER handler, const void* hParam)

TCPIP_IPV6_InterfaceIsReady Function

Determines if an interface is ready for IPv6 transactions.

File

ipv6.h

C

```
bool TCPIP_IPV6_InterfaceIsReady(TCPIP_NET_HANDLE netH);
```

Returns

- true - if the interface has IPv6 functionality available
- false - if the interface does not have IPv6 functionality available

Description

Returns the current state of the IPv6 interface (i.e., determines if an interface is ready for IPv6 transactions).

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
netH	The interface to check

Function

bool TCPIP_IPV6_InterfaceIsReady (TCPIP_NET_HANDLE netH)

TCPIP_IPV6_MulticastListenerAdd Function

Adds a multicast listener to an interface.

File

ipv6.h

C

```
IPV6_ADDR_STRUCT * TCPIP_IPV6_MulticastListenerAdd(TCPIP_NET_HANDLE hNet, IPV6_ADDR * address);
```

Returns

IPV6_ADDR_STRUCT *

- On Success - Pointer to the new listener
- On Failure - NULL

Description

This function is used to add the IPv6 multicast address to an interface.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
hNet	The interface to add the address
address	The new listener

Function

IPV6_ADDR_STRUCT * TCPIP_IPV6_MulticastListenerAdd(TCPIP_NET_HANDLE hNet, IPV6_ADDR * address)

TCPIP_IPV6_MulticastListenerRemove Function

Removes a multicast listener from a given interface.

File

ipv6.h

C

```
void TCPIP_IPV6_MulticastListenerRemove(TCPIP_NET_HANDLE netH, IPV6_ADDR * address);
```

Returns

None.

Description

This function removes a multicast listener from a given interface.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
netH	The interface
address	The address

Function

void TCPIP_IPV6_MulticastListenerRemove (TCPIP_NET_HANDLE netH, IPV6_ADDR * address)

TCPIP_IPV6_PacketFree Function

Frees a TCP/IP Packet structure from dynamic memory.

File

ipv6.h

C

```
void TCPIP_IPV6_PacketFree(IPV6_PACKET * pkt);
```

Returns

None.

Description

This function frees a TCP/IP Packet structure from dynamic memory.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
ptrPacket	The packet to free.

Function

void TCPIP_IPV6_PacketFree (IPV6_PACKET * ptrPacket)

TCPIP_IPV6_PayloadSet Function

Allocates a segment on the end of a packet segment chain and uses it to address prebuffered data.

File

ipv6.h

C

```
unsigned short TCPIP_IPV6_PayloadSet(IPV6_PACKET * pkt, uint8_t* payload, unsigned short len);
```

Returns

- On Success - The amount of data added to the packet length
- On Failure - 0

Description

This function will allocate a data segment header and append it to the end of a chain of segments in a TX packet. It will set the data pointer in the packet segment to a pre-existing buffer of data.

Remarks

This function is useful for adding payloads to outgoing packets without copying them if the data is in another pre-existing buffer (i.e., TCP).

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
ptrPacket	The packet
payload	Address of the data payload
len	Length of the data payload

Function

unsigned short TCPIP_IPV6_PayloadSet (IPV6_PACKET * ptrPacket, uint8_t* payload,

unsigned short len)

TCPIP_IPV6_Put Function

Writes a character of data to a packet.

File

ipv6.h

C

```
bool TCPIP_IPV6_Put(IPV6_PACKET * pkt, unsigned char v);
```

Returns

- true - if the character was written
- false - if the character was not written

Description

This function writes a character of data to a packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
pkt	The packet
v	The character

Function

bool TCPIP_IPV6_Put (IPV6_PACKET * pkt, unsigned char v)

TCPIP_IPV6_SourceAddressGet Function

Gets the source address for an IPv6 packet.

File

ipv6.h

C

```
IPV6_ADDR * TCPIP_IPV6_SourceAddressGet(IPV6_PACKET * p);
```

Returns

IPV6_ADDR *

- On Success - Get a valid IPv6 Source address
- On Failure - NULL

Description

This API is used to get the source address for an IPv6 packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
p	pointer to IPv6 packet

Function

IPV6_ADDR * TCPIP_IPV6_SourceAddressGet(IPV6_PACKET * p)

TCPIP_IPV6_SourceAddressSet Function

Sets the source address for a IPv6 packet.

File

ipv6.h

C

```
void TCPIP_IPV6_SourceAddressSet(IPV6_PACKET * p, const IPV6_ADDR * addr);
```

Returns

None.

Description

This function is used to configure the source address for a IPv6 packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize should be called. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
p	pointer to IPv6 packet
addr	source address

Function

void TCPIP_IPV6_SourceAddressSet(IPV6_PACKET * p, const IPV6_ADDR * addr)

TCPIP_IPV6_TxIsPutReady Function

Determines whether a TX packet can be written to.

File

ipv6.h

C

```
unsigned short TCPIP_IPV6_TxIsPutReady(IPV6_PACKET * pkt, unsigned short count);
```

Returns

- On Success - The amount of space available
- On Failure - 0

Description

This function determines whether a TX packet can be written to. This function will allocate additional space to the packet to accommodate the user.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
ptrPacket	The packet to check
count	The amount of writable space to check for

Function

unsigned short TCPIP_IPV6_TxIsPutReady (IPV6_PACKET * ptrPacket,
unsigned short count)

TCPIP_IPV6_TxPacketAllocate Function

Dynamically allocates a packet for transmitting IP protocol data.

File

ipv6.h

C

```
IPV6_PACKET * TCPIP_IPV6_TxPacketAllocate(TCPIP_NET_HANDLE netH, IPV6_PACKET_ACK_FNC ackFnc, void*
ackParam);
```

Returns

- On Success - Pointer to the allocated packet
- On Failure - NULL

Description

This function dynamically allocates a packet for transmitting IP protocol data and sets the packet IPv6 protocol for a TX packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
netH	Interface of the outgoing packet.
ackFnc	function to be called when IP is done with the TX packet (finished transmitting)
ackParam	parameter to be used for this callback This has meaning only for the caller of the TCPIP_IPV6_TxPacketAllocate

Function

IPV6_PACKET * TCPIP_IPV6_TxPacketAllocate (TCPIP_NET_HANDLE netH,
IPV6_PACKET_ACK_FNC ackFnc, void* ackParam);

TCPIP_IPV6_UniqueLocalUnicastAddressAdd Function

Adds a Unique Local Unicast Address (ULA) to a specified interface.

File

ipv6.h

C

```
IPV6_ULA_RESULT TCPIP_IPV6_UniqueLocalUnicastAddressAdd(TCPIP_NET_HANDLE netH, uint16_t subnetID,
IPV6_ULA_FLAGS genFlags, IP_MULTI_ADDRESS* ntpAddress);
```

Returns

- IPV6_ULA_RES_BUSY - address generation module is busy
- IPV6_ULA_RES_IF_ERR - IPv6 interface is not up
- IPV6_ULA_RES_OK - if the call succeeded and the generation process was started

Description

This function starts the process of adding an ULA address to the specified interface. The segments of the generated address are as follows:

- FC00::/7 - ULA prefix
- L - 1 bit set to 1, locally assigned
- Global ID - 40 bit random generated identifier
- subnet ID - 16 bit subnet identifier
- Interface ID - 64 bit interface identifier generated as a EUI64 from the specified interface MAC

The randomness of the "Global ID" prefix of the generated IPv6 address is obtained by using an NTP server. The supplied NTP server will be

contacted to obtain an NTP time stamp. This time stamp together with the EUI64 identifier obtained from the interface MAC are passed through a 160 bits hash algorithm (SHA1) and the least significant 40 bits are used as the GlobalID of the interface.

Remarks

Only one address generation at a time is supported for now. Before attempting a new address generation the previous operation has to be completed, otherwise the call will fail.

This function requires that the NTP client is enabled in the stack. If not, the call will fail.

The caller will be notified by the outcome of the operation by the stack using the standard IPv6 notification handler (registered by TCPIP_IPV6_HandlerRegister call).

Preconditions

The IPv6 stack is initialized and the interface is up and configured.

Parameters

Parameters	Description
netH	The interface to add the address to.
subnetID	The subnet ID to be used.
genFlags	Address generation flags: • IPV6_ULA_FLAG_NTPV4 - if set, the ntpAddress is an IPv4 address and the NTP server will be contacted over an IPv4 connection. Otherwise, a default IPv6 connection will be attempted • IPV6_ULA_FLAG_GENERATE_ONLY - if set, the address will not be added to the list of the addresses for the specified interface • IPV6_ULA_FLAG_SKIP_DAD - if set, the DAD processing will be skipped
ntpAddress	The NTP server address - it is an IPv4/IPv6 address as selected by the IPV6_ULA_FLAG_NTPV4 flag (the IP address could be obtained with an DNS call into the stack)

Function

IPV6_ULA_RESULT TCPIP_IPV6_UniqueLocalUnicastAddressAdd (TCPIP_NET_HANDLE netH,
uint16_t subnetID, IPV6_ULA_FLAGS genFlags, IP_MULTI_ADDRESS* ntpAddress)

TCPIP_IPV6_UnicastAddressAdd Function

Adds a unicast address to a specified interface

File

ipv6.h

C

```
IPV6_ADDR_STRUCT * TCPIP_IPV6_UnicastAddressAdd(TCPIP_NET_HANDLE netH, IPV6_ADDR * address, int prefixLen,
uint8_t skipProcessing);
```

Returns

IPV6_ADDR_STRUCT *

- On Success - Pointer to the structure of the newly allocated address
- On Failure - NULL

Description

Adds a unicast address to a specified interface. Starts duplicate address detection if necessary.

Remarks

The RFC (4291) requires the interface ID for all unicast addresses, (except those that start with the binary value 000) to be 64 bits long and to be constructed in Modified EUI-64 format. Therefore the prefixLen parameter should probably always be 64.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true.

Parameters

Parameters	Description
netH	The interface to which the the address is to be addded

address	The address to add
prefixLen	The prefix length associated to this static address (providing the subnet prefix length). If 0, the default value of 64 will be used
skipProcessing	true to skip Duplicate address detection; otherwise, false

Function

IPV6_ADDR_STRUCT * TCPIP_IPV6_UnicastAddressAdd (TCPIP_NET_HANDLE netH,
IPV6_ADDR * address, int prefixLen, uint8_t skipProcessing)

TCPIP_IPV6_RouterAddressAdd Function

Adds a new router address to a specified interface.

File

ipv6.h

C

```
bool TCPIP_IPV6_RouterAddressAdd(TCPIP_NET_HANDLE netH, IPV6_ADDR * rAddress, unsigned long validTime, int flags);
```

Returns

- true - operation succeeded
- false - operation failed (interface not valid, interface still configuring, no memory, etc.)

Description

This function adds a unicast address as a routing address to a specified interface.

Remarks

The validTime parameter is relevant for an existent router on the network. If such router does not exist the stack will eventually discard the entry automatically.

Preconditions

rAddress pointer to a valid IPv6 router address IPv6 stack initialized, interface up and configured

Parameters

Parameters	Description
netH	The interface to which the address is to be added
rAddress	The router address to add
validTime	The time this valid will be valid, seconds. If 0, the entry will be valid forever.
flags	Creation flags (not used for now, should be 0)

Function

bool TCPIP_IPV6_RouterAddressAdd(TCPIP_NET_HANDLE netH, IPV6_ADDR * rAddress,
unsigned long validTime, int flags);

TCPIP_IPV6_DefaultRouterDelete Function

Deletes the current router list for a specified interface

File

ipv6.h

C

```
void TCPIP_IPV6_DefaultRouterDelete(TCPIP_NET_HANDLE netH);
```

Returns

None.

Description

This function deletes all default routers on a given interface

Remarks

None.

Preconditions

The IPv6 stack is initialized and the interface is up and configured.

Parameters

Parameters	Description
netH	The interface for which to delete the router address.

Function

void TCPIP_IPV6_DefaultRouterDelete(TCPIP_NET_HANDLE netH);

TCPIP_IPV6_DefaultRouterGet Function

Returns the current router address for a specified interface.

File

ipv6.h

C

```
const IPV6_ADDR* TCPIP_IPV6_DefaultRouterGet(TCPIP_NET_HANDLE netH);
```

Returns

- valid IPV6_ADDR pointer - if the interface exists and a valid router address exists
- 0 - if the interface does not exist and a valid router address does not exist

Description

This function returns the current router address for a specified interface.

Remarks

None.

Preconditions

The IPv6 stack is initialized and the interface is up and configured.

Parameters

Parameters	Description
netH	The interface for which to return the router address.

Function

const IPV6_ADDR* TCPIP_IPV6_DefaultRouterGet(TCPIP_NET_HANDLE hNet);

TCPIP_IPV6_Task Function

Standard TCP/IP stack module task function.

File

ipv6.h

C

```
void TCPIP_IPV6_Task();
```

Returns

None.

Description

This function performs IPv6 module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The IPv6 stack is initialized.

Function

void TCPIP_IPV6_Task(void)

b) Data Types and Constants

IP_VERSION_4 Macro

File

ipv6.h

C

```
#define IP_VERSION_4 (0u) // Using IPv4
```

Description

Using IPv4

IP_VERSION_6 Macro

File

ipv6.h

C

```
#define IP_VERSION_6 (1u) // Using IPv6
```

Description

Using IPv6

IPV6_DATA_DYNAMIC_BUFFER Macro

File

ipv6.h

C

```
#define IPV6_DATA_DYNAMIC_BUFFER (0x1u)        // Data to transmit is allocated in dynamically allocated RAM
```

Description

Data to transmit is allocated in dynamically allocated RAM

IPV6_DATA_NETWORK_FIFO Macro

File

ipv6.h

C

```
#define IPV6_DATA_NETWORK_FIFO (0x2u)        // Data to transmit is stored in the Network Controller's FIFOs
```

Description

Data to transmit is stored in the Network Controller's FIFOs

IPV6_DATA_NONE Macro

File

ipv6.h

C

```
#define IPV6_DATA_NONE (0x0u)        // The data segment is unused
```

Description

The data segment is unused

IPV6_DATA_PIC_RAM Macro

File

ipv6.h

C

```
#define IPV6_DATA_PIC_RAM (0x3u)        // Data to transmit is stored in PIC RAM
```

Description

Data to transmit is stored in PIC RAM

IPV6_HEADER_OFFSET_DEST_ADDR Macro

File

ipv6.h

C

```
#define IPV6_HEADER_OFFSET_DEST_ADDR (0x08u + sizeof (IPV6_ADDR))
```

Description

Header offset for destination address

IPV6_HEADER_OFFSET_NEXT_HEADER Macro

File

ipv6.h

C

```
#define IPV6_HEADER_OFFSET_NEXT_HEADER (0x06u)
```

Description

Header offset for next header

IPV6_HEADER_OFFSET_PAYLOAD_LENGTH Macro

File

ipv6.h

C

```
#define IPV6_HEADER_OFFSET_PAYLOAD_LENGTH (0x04u)
```

Description

Header offset for payload length

IPV6_HEADER_OFFSET_SOURCE_ADDR Macro

File

ipv6.h

C

```
#define IPV6_HEADER_OFFSET_SOURCE_ADDR (0x08u)
```

Description

Header offset for source address

IPV6_NO_UPPER_LAYER_CHECKSUM Macro

File

ipv6.h

C

```
#define IPV6_NO_UPPER_LAYER_CHECKSUM (0xFFFFu)    // Value flag for no upper layer checksum
```

Description

Value flag for no upper layer checksum

IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM Macro

File

ipv6.h

C

```
#define IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM 0xC2u
```

Description

IPv6 Type-length-value type code for the Hop-by-hop "Jumbo-gram Payload" option

IPV6_TLV_HBHO_ROUTER_ALERT Macro

File

ipv6.h

C

```
#define IPV6_TLV_HBHO_ROUTER_ALERT 0x05u
```

Description

IPv6 Type-length-value type code for the Hop-by-hop "Router Alert" option

IPV6_TLV_PAD_1 Macro

File

ipv6.h

C

```
#define IPV6_TLV_PAD_1 0u
```

Description

IPv6 Type-length-value type code for the Pad 1 option

IPV6_TLV_PAD_N Macro

File

ipv6.h

C

```
#define IPV6_TLV_PAD_N 1u
```

Description

IPv6 Type-length-value type code for the Pad N option

IPV6_TLV_UNREC_OPT_DISCARD_PP Macro

File

ipv6.h

C

```
#define IPV6_TLV_UNREC_OPT_DISCARD_PP 0b10
```

Description

IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message

IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC Macro

File

ipv6.h

C

```
#define IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC 0b11
```

Description

IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message is the destination address isn't a multicast address

IPV6_TLV_UNREC_OPT_DISCARD_SILENT Macro

File

ipv6.h

C

```
#define IPV6_TLV_UNREC_OPT_DISCARD_SILENT 0b01
```

Description

IPv6 action code for the unrecognized option reaction to discard the packet silently

IPV6_TLV_UNREC_OPT_SKIP_OPTION Macro

File

ipv6.h

C

```
#define IPV6_TLV_UNREC_OPT_SKIP_OPTION 0b00
```

Description

IPv6 action code for the unrecognized option reaction to skip the option

TCPIP_IPV6_PutArray Macro

Writes data to a packet

File

ipv6.h

C

```
#define TCPIP_IPV6_PutArray(pkt,data,len) TCPIP_IPV6_ArrayPutHelper(pkt, data, IPV6_DATA_PIC_RAM, len)
```

Returns

unsigned short - The number of bytes of data written.

Description

This function writes data to an outgoing packet.

Remarks

None.

Preconditions

TCPIP_IPV6_Initialize is required. TCPIP_IPV6_InterfaceIsReady should be true. The TCPIP_IPV6_TxIsPutReady function must have returned a value greater than or equal to 'len'.

Parameters

Parameters	Description
ptrPacket	The packet
dataSource	Pointer to the data to copy to the packet
len	Length of the data

Function

unsigned short TCPIP_IPV6_PutArray (IPV6_PACKET * ptrPacket,

const void * dataSource, unsigned short len)

IPV6_ACTION Enumeration

Provides a list of possible IPv6 actions.

File

ipv6.h

C

```
typedef enum {
  IPV6_ACTION_NONE = 0,
  IPV6_ACTION_DISCARD_SILENT,
  IPV6_ACTION_DISCARD_PP_0,
  IPV6_ACTION_DISCARD_PP_2,
  IPV6_ACTION_DISCARD_PP_2_NOT_MC,
  IPV6_ACTION_BEGIN_EX_HEADER_PROCESSING
} IPV6_ACTION;
```

Members

Members	Description
IPV6_ACTION_NONE = 0	No action. Skip the option
IPV6_ACTION_DISCARD_SILENT	Discard the packet silently
IPV6_ACTION_DISCARD_PP_0	Discard the packet and send an ICMP parameter problem message with code value 0
IPV6_ACTION_DISCARD_PP_2	Discard the packet and send an ICMP parameter problem message with code value 2
IPV6_ACTION_DISCARD_PP_2_NOT_MC	Discard the packet and send an ICMP parameter problem message with code value 3
IPV6_ACTION_BEGIN_EX_HEADER_PROCESSING	Begin extension processing

Description

Enumeration: IPV6_ACTION

Different actions need to be taken depending on the result of our header processing.

Remarks

None.

IPV6_ADDRESS_POLICY Structure

Data structure for IPv6 address policy.

File

ipv6.h

C

```
typedef struct {
  IPV6_ADDR address;
  unsigned char prefixLength;
  unsigned char precedence;
  unsigned char label;
} IPV6_ADDRESS_POLICY;
```

Members

Members	Description
IPV6_ADDR address;	IPv6 address
unsigned char prefixLength;	IPv6 prefix length
unsigned char precedence;	IPv6 address precedence
unsigned char label;	IPv6 label type

Description

Type: IPV6_ADDRESS_POLICY

The policy table is the longest matching prefix lookup table and is used to select the destination IPv6 Address.

If Precedence(A) > Precedence(B), address A has higher precedence than address B. The label value Label(A) allows for policies that prefer a particular source address prefix for use with a destination address prefix. Default Policy table - Prefix Precedence Label ::1/128 50 0 ::/0 40 1 2002::/16 30 2 ::/96 20 3 ::ffff:0.0.0.0/96 10 4

Remarks

None.

Example

const IPV6_ADDRESS_POLICY gPolicyTable[] = { {{{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01}}, 128, 50, 0}, // Loopback address {{{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}}, 0, 40, 1}, // Unspecified address {{{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00}}, 96, 35, 4}, // IPv4-mapped address {{{0x20, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}}, 16, 30, 2}, // 2002::/15 - 6to4 {{{0x20, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}}, 32, 5, 5}, // 2001::/32 - Teredo tunneling {{{0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}}, 7, 3, 13}, // ULA };

IPV6_ADDRESS_PREFERENCE Enumeration

Provides selection of public versus temporary addresses.

File

ipv6.h

C

```
typedef enum {
  IPV6_PREFER_PUBLIC_ADDRESSES = 0,
  IPV6_PREFER_TEMPORARY_ADDRESSES
} IPV6_ADDRESS_PREFERENCE;
```

Description

Enumeration: IPV6_ADDRESS_PREFERENCE

None

Remarks

None.

IPV6_ADDRESS_TYPE Union

Data structure for IPv6 address types.

File

ipv6.h

C

```
typedef union {
  unsigned char byte;
  struct {
    unsigned scope : 4;
    unsigned type : 2;
  } bits;
} IPV6_ADDRESS_TYPE;
```

Description

Type: IPV6_ADDRESS_TYPE

This type defines the data structure for IPv6 address types.

Remarks

None.

IPV6_DATA_SEGMENT_HEADER Structure

Data structure for IPv6 Data Segment header.

File

ipv6.h

C

```
typedef struct _IPV6_DATA_SEGMENT_HEADER {
  uint8_t* dataLocation;
  unsigned short segmentSize;
  unsigned short segmentLen;
  unsigned char memory;
  unsigned char segmentType;
  struct _IPV6_DATA_SEGMENT_HEADER * nextSegment;
  void * data[0];
} IPV6_DATA_SEGMENT_HEADER;
```

Members

Members	Description
uint8_t* dataLocation;	Location of the data to transmit
unsigned short segmentSize;	Size of this data segment
unsigned short segmentLen;	Number of bytes of data in this segment
unsigned char memory;	Type: IPV6_DATA_NONE, IPV6_DATA_DYNAMIC_BUFFER, IPV6_DATA_NETWORK_FIFO, IPV6_DATA_PIC_RAM
unsigned char segmentType;	Type of segment contents
struct _IPV6_DATA_SEGMENT_HEADER * nextSegment;	Pointer to the next data segment
void * data[0];	Optional buffer space

Description

Structure : IPV6_DATA_SEGMENT_HEADER

Data structure is used to allocate a data segment header and an optional payload.

Remarks

None.

IPV6_EVENT_HANDLER Type

Clients can register a handler with the IPv6 service.

File

ipv6.h

C

```
typedef void (* IPV6_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, IPV6_EVENT_TYPE evType, const void* evParam,
const void* usrParam);
```

Description

Type: IPV6_EVENT_HANDLER

Once an IPv6 event occurs the IPv6 service will call the registered handler The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

evParam is a parameter that's associated to an IPv6 event

- For an address event (IPV6_EVENT_ADDRESS_ADDED, IPV6_EVENT_ADDRESS_REMOVED) it should typecast to (const IPV6_ADDR_STRUCT*)
- For an IPV6_EVENT_ULA_ADDRESS_GENERATED ULA event it should typecast to (const IPV6_ADDR*)
- For an IPV6_EVENT_ULA_ADDRESS_FAILED ULA event the evParam is an IPV6_ULA_RESULT error code

The evParam is invalid outside of the IPV6_EVENT_HANDLER context call and should not be stored by the caller. Info that's needed has to be copied into caller's own context.

usrParam is a user-supplied parameter

Remarks

For address related events the passed (const IPV6_ADDR_STRUCT*) parameter is invalid after the notification call.

IPV6_EVENT_TYPE Enumeration

This enumeration is used to notify IPv6 client applications.

File

ipv6.h

C

```
typedef enum {
  IPV6_EVENT_ADDRESS_ADDED = 1,
  IPV6_EVENT_ADDRESS_REMOVED,
  IPV6_EVENT_ULA_ADDRESS_GENERATED,
  IPV6_EVENT_ULA_ADDRESS_FAILED
} IPV6_EVENT_TYPE;
```

Members

Members	Description
IPV6_EVENT_ADDRESS_ADDED = 1	Event is generated when IPv6 added to the list
IPV6_EVENT_ADDRESS_REMOVED	Event is generated when IPv6 address removed from the list
IPV6_EVENT_ULA_ADDRESS_GENERATED	Event is generated when IPv6 unique local address included to the list
IPV6_EVENT_ULA_ADDRESS_FAILED	Event is generated when IPv6 unique local address is failed

Description

Enumeration: IPV6_EVENT_TYPE

These events are used while notifying to the registered applications.

Remarks

None.

IPV6_FRAGMENT_HEADER Structure

Data structure for IPv6 fragment header.

File

ipv6.h

C

```
typedef struct {
  uint8_t nextHeader;
  uint8_t reserved;
  union {
    struct {
      uint16_t m : 1;
      uint16_t reserved2 : 2;
      uint16_t fragmentOffset : 13;
    } bits;
    uint16_t w;
  } offsetM;
  uint32_t identification;
} IPV6_FRAGMENT_HEADER;
```

Description

Structure: IPV6_FRAGMENT_HEADER

The Fragment header is used by an IPv6 source to send a packet larger than would fit in the path MTU to its destination. The Fragment header is identified by a Next Header value of 44 in the immediately preceding header.

Remarks

None.

IPV6_HANDLE Type

File

ipv6.h

C

```
typedef const void * IPV6_HANDLE;
```

Description

Pointer to IPv6 object

IPV6_HEADER Structure

IPv6 packet header definition.

File

ipv6.h

C

```
typedef struct {
  unsigned long V_T_F;
  unsigned short PayloadLength;
  unsigned char NextHeader;
  unsigned char HopLimit;
  IPV6_ADDR SourceAddress;
  IPV6_ADDR DestAddress;
} IPV6_HEADER;
```

Members

Members	Description
unsigned long V_T_F;	Version , Traffic class and Flow Label
unsigned short PayloadLength;	Length of IPv6 payload, i.e. the rest of packet following this IPv6 header in octets
unsigned char NextHeader;	Identifies the type of header immediately following IPv6 header
unsigned char HopLimit;	Decremented by 1 by each node that forwards the packet
IPV6_ADDR SourceAddress;	128 bit address of originator
IPV6_ADDR DestAddress;	128 bit address of intended recipient

Description

Structure: IPV6_HEADER

IPv6 packet header definition.

Remarks

Any extension headers present are considered part of the payload length.

IPV6_NEXT_HEADER_TYPE Enumeration

Defines a list of IPv6 next header types.

File

ipv6.h

C

```
typedef enum {
  IPV6_PROT_HOP_BY_HOP_OPTIONS_HEADER = (0u),
  IPV6_PROT_ICMP = (1u),
  IPV6_PROT_TCP = (6u),
  IPV6_PROT_UDP = (17u),
  IPV6_PROT_IPV6 = (41u),
  IPV6_PROT_ROUTING_HEADER = (43u),
  IPV6_PROT_FRAGMENTATION_HEADER = (44u),
  IPV6_PROT_ENCAPSULATING_SECURITY_PAYLOAD_HEADER = (50u),
  IPV6_PROT_AUTHENTICATION_HEADER = (51u),
  IPV6_PROT_ICMPV6 = (58u),
  IPV6_PROT_NONE = (59u),
  IPV6_PROT_DESTINATION_OPTIONS_HEADER = (60u)
} IPV6_NEXT_HEADER_TYPE;
```

Members

Members	Description
IPV6_PROT_HOP_BY_HOP_OPTIONS_HEADER = (0u)	IPv6 Hop-by-Hop Opt. Header
IPV6_PROT_ICMP = (1u)	ICMPv4 Header
IPV6_PROT_TCP = (6u)	TCP protocol Header
IPV6_PROT_UDP = (17u)	UDP protocol Header
IPV6_PROT_IPV6 = (41u)	IPv6 Protocol
IPV6_PROT_ROUTING_HEADER = (43u)	IPv6 Routing Header
IPV6_PROT_FRAGMENTATION_HEADER = (44u)	IPv6 Fragmentation Header
IPV6_PROT_ENCAPSULATING_SECURITY_PAYLOAD_HEADER = (50u)	Encapsulating Security Payload Header
IPV6_PROT_AUTHENTICATION_HEADER = (51u)	Authentication Header
IPV6_PROT_ICMPV6 = (58u)	ICMPv6 Protocol
IPV6_PROT_NONE = (59u)	No next header
IPV6_PROT_DESTINATION_OPTIONS_HEADER = (60u)	Destination Options Header

Description

Enumeration: IPV6_NEXT_HEADER_TYPE

Identifies the type of the next header immediately follows IPv6 header.

Remarks

None

IPV6_PACKET Structure

Packet structure/state tracking for IPv6 packets.

File

ipv6.h

C

```
typedef struct _IPV6_PACKET {
  struct _IPV6_PACKET * next;
  unsigned short payloadLen;
  unsigned short headerLen;
  unsigned short upperLayerHeaderLen;
  unsigned short upperLayerChecksumOffset;
  unsigned char upperLayerHeaderType;
  union {
    struct {
      unsigned char reserved : 3;
      unsigned char useUnspecAddr : 1;
      unsigned char sourceSpecified : 1;
      unsigned char queued : 1;
      unsigned char addressType : 2;
    }
    unsigned char val;
  } flags;
  TCPIP_MAC_ADDR remoteMACAddr;
  IPV6_PACKET_ACK_FNC ackFnc;
  TCPIP_MAC_PACKET_ACK_FUNC macAckFnc;
  void* ackParam;
  void* clientData;
  void * neighbor;
  unsigned short offsetInSegment;
  uint32_t queuedPacketTimeout;
  IPV6_DATA_SEGMENT_HEADER payload;
  TCPIP_NET_HANDLE netIfH;
  IPV6_HEADER ipv6Header;
} IPV6_PACKET;
```

Members

Members	Description
struct _IPV6_PACKET * next;	Next packet in a queue
unsigned short payloadLen;	Amount of data in payload buffer
unsigned short headerLen;	Total header length (IP header + IPv6 Extension headers)
unsigned short upperLayerHeaderLen;	Total length of the upper layer header
unsigned short upperLayerChecksumOffset;	Offset of the upper layer checksum
unsigned char upperLayerHeaderType;	Type definition for the upper-layer header type
unsigned char useUnspecAddr : 1;	This packet should use the unspecified address
unsigned char sourceSpecified : 1;	The upper layer or application layer specified a source address
unsigned char queued : 1;	Packet has been queued
unsigned char addressType : 2;	IP_ADDRESS_TYPE_IPV6 or IP_ADDRESS_TYPE_IPV4
TCPIP_MAC_ADDR remoteMACAddr;	The packet's remote MAC address
IPV6_PACKET_ACK_FNC ackFnc;	function to be called when IPv6 done with the packet
TCPIP_MAC_PACKET_ACK_FUNC macAckFnc;	function to be called when MAC done with a TX packet
void* ackParam;	parameter to be used
void* clientData;	optional packet client data
void * neighbor;	The neighbor that the message was received from
unsigned short offsetInSegment;	Offset used for storing fragment transmission information
uint32_t queuedPacketTimeout;	Time out for IPv6 packets which are queued
IPV6_DATA_SEGMENT_HEADER payload;	IPv6 data segment payload
TCPIP_NET_HANDLE netIfH;	packet network interface
IPV6_HEADER ipv6Header;	IPv6 header definition

Description

Structure: IPV6_PACKET

IPv6 packets are queued for future transmission.queuedPacketTimeout is used to time-out IPv6 queued packets.

Remarks

For IPv6 queuing the time out has to be 0! The queue is processed separately by the NDP.IPV6_PACKET_ACK_FNC is called after the successful removal and inclusion of the packet.

IPV6_PACKET_ACK_FNC Type

Packet allocation and deallocation acknowledgment callback function.

File

ipv6.h

C

```
typedef bool (* IPV6_PACKET_ACK_FNC)(void*, bool, const void*);
```

Returns

-true - if the packet needs the queuing flags removed (it is not deleted and still in use) -false - if the packet does not need the queuing flags removed (either no longer exists or the flags updated)

Description

Type: IPV6_PACKET_ACK_FNC

Packet allocation and deallocation acknowledgment callback function.

Remarks

None.

Parameters

Parameters	Description
void *	Pointer to the packet that was transmitted
bool	True if the packet was sent, false otherwise
const void*	0

IPV6_RX_FRAGMENT_BUFFER Structure

Data structure for IPv6 Received fragmented packet.

File

ipv6.h

C

```
typedef struct _IPV6_RX_FRAGMENT_BUFFER {
  struct _IPV6_RX_FRAGMENT_BUFFER * next;
  uint8_t * ptrPacket;
  uint32_t identification;
  uint16_t bytesInPacket;
  uint16_t packetSize;
  uint16_t firstFragmentLength;
  uint8_t secondsRemaining;
} IPV6_RX_FRAGMENT_BUFFER;
```

Members

Members	Description
struct _IPV6_RX_FRAGMENT_BUFFER * next;	Next fragmented packet
uint8_t * ptrPacket;	Packet information
uint32_t identification;	Fragment id
uint16_t bytesInPacket;	Number of bytes written to packet
uint16_t packetSize;	Packet size (packet is complete when this matches bytesInPacket)
uint16_t firstFragmentLength;	Length of the first fragment
uint8_t secondsRemaining;	Number of seconds remaining during which the fragment can be reassembled

Description

Structure: IPV6_RX_FRAGMENT_BUFFER

Each fragment is composed of unfragmentable Part and fragmentable part. Allocate memory for the fragmented packet w.r.t to TCPIP_IPV6_RX_FRAGMENTED_BUFFER_SIZE. The RFC specifies that the fragments must be reassembled in one minute or less.secondsRemaining is the times remaining for reassemble.

Remarks

None.

IPV6_SEGMENT_TYPE Enumeration

Provides an enumeration of IPv6 segment types.

File

ipv6.h

C

```
typedef enum {
  TYPE_IPV6_HEADER = 1u,
  TYPE_IPV6_EX_HEADER_HOP_BY_HOP_OPTIONS,
  TYPE_IPV6_EX_HEADER_DESTINATION_OPTIONS_1,
  TYPE_IPV6_EX_HEADER_ROUTING,
  TYPE_IPV6_EX_HEADER_FRAGMENT,
  TYPE_IPV6_EX_HEADER_AUTHENTICATION_HEADER,
  TYPE_IPV6_EX_HEADER_ENCAPSULATING_SECURITY_PAYLOAD,
  TYPE_IPV6_EX_HEADER_DESTINATION_OPTIONS_2,
  TYPE_IPV6_UPPER_LAYER_HEADER,
  TYPE_IPV6_UPPER_LAYER_PAYLOAD,
  TYPE_IPV6_BEGINNING_OF_WRITABLE_PART,
  TYPE_IPV6_END_OF_LIST
} IPV6_SEGMENT_TYPE;
```

Description

Enumeration: IPV6_SEGMENT_TYPE

IPv6 extended header.

Remarks

None.

IPV6_TLV_OPTION_TYPE Union

Data structure for IPv6 TLV options.

File

ipv6.h

C

```
typedef union {
  unsigned char b;
  struct {
    unsigned option : 6;
    unsigned unrecognizedAction : 2;
  } bits;
} IPV6_TLV_OPTION_TYPE;
```

Description

Type: IPV6_TLV_OPTION_TYPE

This type defines the data structure for IPv6 TLV options.

Remarks

None.

IPV6_ULA_FLAGS Enumeration

Provides a list of possible ULA action flags.

File

ipv6.h

C

```
typedef enum {
  IPV6_ULA_FLAG_NTPV4 = 0x01,
  IPV6_ULA_FLAG_GENERATE_ONLY = 0x02,
  IPV6_ULA_FLAG_SKIP_DAD = 0x04
} IPV6_ULA_FLAGS;
```

Members

Members	Description
IPV6_ULA_FLAG_NTPV4 = 0x01	use an IPv4 NTP server access in Unique Local Address generation default is an IPv6 server
IPV6_ULA_FLAG_GENERATE_ONLY = 0x02	generate an address only, don't add it to the interface addresses
IPV6_ULA_FLAG_SKIP_DAD = 0x04	when adding the address to the interface, skip the Duplicate Address Detection

Description

Enumeration: IPV6_ULA_FLAGS

This enumeration provides a list of possible flags for the Unique Local Address (ULA) generation.

Remarks

None.

IPV6_ULA_RESULT Enumeration

Provides a list of possible ULA results.

File

ipv6.h

C

```
typedef enum {
  IPV6_ULA_RES_OK,
  IPV6_ULA_RES_BUSY = -1,
  IPV6_ULA_RES_IF_ERR = -2,
  IPV6_ULA_RES_NTP_ACCESS_ERR = -3,
  IPV6_ULA_RES_NTP_TSTAMP_ERR = -4
} IPV6_ULA_RESULT;
```

Members

Members	Description
IPV6_ULA_RES_OK	the address generation was started successfully
IPV6_ULA_RES_BUSY = -1	address generation module is busy
IPV6_ULA_RES_IF_ERR = -2	interface non existent
IPV6_ULA_RES_NTP_ACCESS_ERR = -3	NTP module could not be accessed
IPV6_ULA_RES_NTP_TSTAMP_ERR = -4	wrong NTP time stamp received

Description

Enumeration: IPV6_ULA_RESULT

This enumeration provides a list of possible results for the Unique Local Address (ULA) generation.

Remarks

None.

TCPIP_IPV6_MODULE_CONFIG Structure

Provides a place holder for IPv6 configuration.

File

ipv6.h

C

```
typedef struct {
  uint32_t rxfragmentBufSize;
```

```
  uint32_t fragmentPktRxTimeout;
} TCPIP_IPV6_MODULE_CONFIG;
```

Members

Members	Description
uint32_t rxfragmentBufSize;	RX fragmented buffer size
uint32_t fragmentPktRxTimeout;	fragmented packet time out value

Description

Structure: TCPIP_IPV6_MODULE_CONFIG

IPv6 module runtime configuration and initialization parameters.

Remarks

None.

Files

Files

Name	Description
ipv6.h	IPv6 is the next generation TCP/IP protocol suite.
ipv6_config.h	IPv6 configuration file

Description

This section lists the source and header files used by the library.

ipv6.h

IPv6 is the next generation TCP/IP protocol suite.

Enumerations

	Name	Description
	IPV6_ACTION	Provides a list of possible IPv6 actions.
	IPV6_ADDRESS_PREFERENCE	Provides selection of public versus temporary addresses.
	IPV6_EVENT_TYPE	This enumeration is used to notify IPv6 client applications.
	IPV6_NEXT_HEADER_TYPE	Defines a list of IPv6 next header types.
	IPV6_SEGMENT_TYPE	Provides an enumeration of IPv6 segment types.
	IPV6_ULA_FLAGS	Provides a list of possible ULA action flags.
	IPV6_ULA_RESULT	Provides a list of possible ULA results.

Functions

	Name	Description
	TCPIP_IPV6_AddressUnicastRemove	Removed a configured unicast address from an interface.
	TCPIP_IPV6_ArrayGet	Reads the next byte of data from the specified MAC.
	TCPIP_IPV6_ArrayPutHelper	Helper function to write data to a packet.
	TCPIP_IPV6_DASSourceAddressSelect	Determines the appropriate source address for a given destination address.
	TCPIP_IPV6_DefaultRouterDelete	Deletes the current router list for a specified interface
	TCPIP_IPV6_DefaultRouterGet	Returns the current router address for a specified interface.
	TCPIP_IPV6_DestAddressGet	Gets the destination address for a IPv6 packet.
	TCPIP_IPV6_DestAddressSet	Sets the destination address for a IPv6 packet.
	TCPIP_IPV6_Flush	Flushes a IP TX packet.
	TCPIP_IPV6_Get	Reads the next byte of data from the specified MAC.
	TCPIP_IPV6_HandlerDeregister	Deregisters an IPv6 event handler callback function.
	TCPIP_IPV6_HandlerRegister	Registers an IPv6 event handler callback function.
	TCPIP_IPV6_InterfaceIsReady	Determines if an interface is ready for IPv6 transactions.
	TCPIP_IPV6_MulticastListenerAdd	Adds a multicast listener to an interface.
	TCPIP_IPV6_MulticastListenerRemove	Removes a multicast listener from a given interface.
	TCPIP_IPV6_PacketFree	Frees a TCP/IP Packet structure from dynamic memory.

	TCPIP_IPV6_PayloadSet	Allocates a segment on the end of a packet segment chain and uses it to address prebuffered data.
	TCPIP_IPV6_Put	Writes a character of data to a packet.
	TCPIP_IPV6_RouterAddressAdd	Adds a new router address to a specified interface.
	TCPIP_IPV6_SourceAddressGet	Gets the source address for an IPv6 packet.
	TCPIP_IPV6_SourceAddressSet	Sets the source address for a IPv6 packet.
	TCPIP_IPV6_Task	Standard TCP/IP stack module task function.
	TCPIP_IPV6_TxIsPutReady	Determines whether a TX packet can be written to.
	TCPIP_IPV6_TxPacketAllocate	Dynamically allocates a packet for transmitting IP protocol data.
	TCPIP_IPV6_UnicastAddressAdd	Adds a unicast address to a specified interface
	TCPIP_IPV6_UniqueLocalUnicastAddressAdd	Adds a Unique Local Unicast Address (ULA) to a specified interface.

Macros

	Name	Description
	IP_VERSION_4	Using IPv4
	IP_VERSION_6	Using IPv6
	IPV6_DATA_DYNAMIC_BUFFER	Data to transmit is allocated in dynamically allocated RAM
	IPV6_DATA_NETWORK_FIFO	Data to transmit is stored in the Network Controller's FIFOs
	IPV6_DATA_NONE	The data segment is unused
	IPV6_DATA_PIC_RAM	Data to transmit is stored in PIC RAM
	IPV6_HEADER_OFFSET_DEST_ADDR	Header offset for destination address
	IPV6_HEADER_OFFSET_NEXT_HEADER	Header offset for next header
	IPV6_HEADER_OFFSET_PAYLOAD_LENGTH	Header offset for payload length
	IPV6_HEADER_OFFSET_SOURCE_ADDR	Header offset for source address
	IPV6_NO_UPPER_LAYER_CHECKSUM	Value flag for no upper layer checksum
	IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM	IPv6 Type-length-value type code for the Hop-by-hop "Jumbo-gram Payload" option
	IPV6_TLV_HBHO_ROUTER_ALERT	IPv6 Type-length-value type code for the Hop-by-hop "Router Alert" option
	IPV6_TLV_PAD_1	IPv6 Type-length-value type code for the Pad 1 option
	IPV6_TLV_PAD_N	IPv6 Type-length-value type code for the Pad N option
	IPV6_TLV_UNREC_OPT_DISCARD_PP	IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message
	IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC	IPv6 action code for the unrecognized option reaction to discard the packet and send an ICMP parameter problem message is the destination address isn't a multicast address
	IPV6_TLV_UNREC_OPT_DISCARD_SILENT	IPv6 action code for the unrecognized option reaction to discard the packet silently
	IPV6_TLV_UNREC_OPT_SKIP_OPTION	IPv6 action code for the unrecognized option reaction to skip the option
	TCPIP_IPV6_PutArray	Writes data to a packet

Structures

	Name	Description
	_IPV6_DATA_SEGMENT_HEADER	Data structure for IPv6 Data Segment header.
	_IPV6_PACKET	Packet structure/state tracking for IPv6 packets.
	_IPV6_RX_FRAGMENT_BUFFER	Data structure for IPv6 Received fragmented packet.
	IPV6_ADDRESS_POLICY	Data structure for IPv6 address policy.
	IPV6_DATA_SEGMENT_HEADER	Data structure for IPv6 Data Segment header.
	IPV6_FRAGMENT_HEADER	Data structure for IPv6 fragment header.
	IPV6_HEADER	IPv6 packet header definition.
	IPV6_PACKET	Packet structure/state tracking for IPv6 packets.
	IPV6_RX_FRAGMENT_BUFFER	Data structure for IPv6 Received fragmented packet.
	TCPIP_IPV6_MODULE_CONFIG	Provides a place holder for IPv6 configuration.

Types

	Name	Description
	IPV6_EVENT_HANDLER	Clients can register a handler with the IPv6 service.
	IPV6_HANDLE	Pointer to IPv6 object

	IPV6_PACKET_ACK_FNC	Packet allocation and deallocation acknowledgment callback function.

Unions

	Name	Description
	IPV6_ADDRESS_TYPE	Data structure for IPv6 address types.
	IPV6_TLV_OPTION_TYPE	Data structure for IPv6 TLV options.

Description

IPv6 Definitions for the Microchip TCP/IP Stack

Like IPv4 , IPv6 (Internet Protocol Version 6) is the another version of IP protocol. This is the next-generation protocol which works parallel with IPv4. Microchip TCP/IP stack is a dual stack architecture where both Ipv4 and Ipv6 works simultaneously. (RFC - 2460,3484)

File Name

ipv6.h

Company

Microchip Technology Inc.

ipv6_config.h

IPv6 configuration file

Macros

	Name	Description
	TCPIP_IPV6_DEFAULT_ALLOCATION_BLOCK_SIZE	Sets the minimum allocation unit for the payload size
	TCPIP_IPV6_DEFAULT_BASE_REACHABLE_TIME	Default 30 seconds , Router advertisement reachable time
	TCPIP_IPV6_DEFAULT_CUR_HOP_LIMIT	IPv4 Time-to-Live parameter
	TCPIP_IPV6_DEFAULT_LINK_MTU	Default Maximum Transmission Unit
	TCPIP_IPV6_DEFAULT_RETRANSMIT_TIME	1 second, Process the router advertisement's retransmission time
	TCPIP_IPV6_FRAGMENT_PKT_TIMEOUT	Fragmentation packet time-out value. default value is 60 .
	TCPIP_IPV6_INIT_TASK_PROCESS_RATE	IPv6 initialize task processing rate, milliseconds The default value is 32 milliseconds.
	TCPIP_IPV6_MINIMUM_LINK_MTU	Sets the lower bounds of the Maximum Transmission Unit
	TCPIP_IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT	10 minutes
	TCPIP_IPV6_QUEUE_MCAST_PACKET_LIMIT	This option defines the maximum number of multicast queued IPv6 If an additional packet is queued, the oldest packet in the queue will be removed.
	TCPIP_IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT	This option defines the maximum number of queued packets per remote. If an additional packet needs to be queued, the oldest packet in the queue will be removed.
	TCPIP_IPV6_QUEUED_MCAST_PACKET_TIMEOUT	This option defines the number of seconds an IPv6 multicast packet will remain in the queue before being timed out
	TCPIP_IPV6_RX_FRAGMENTED_BUFFER_SIZE	RX fragmented buffer size should be equal to the total original packet size of ICMPv6 ECHO request packets . ex - Let Transmit ICMPv6 Echo request packet whose original packet size is 1500byte from the Global address of HOST1 to the global address of HOST2 and if the packet is going to be fragmented then packet will be broken more than packets. Each packet will have IPv6 header(40 bytes)+ Fragmentation header (8 bytes) + ICMPv6 Echo request header(8 bytes) • Payload (data packet).PING6(1500=40+8+8+1452 bytes). Here data packet size is 1452. If the data packet size is getting changed then this... more
	TCPIP_IPV6_TASK_PROCESS_RATE	IPv6 task processing rate, milliseconds The default value is 1000 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_IPV6_ULA_NTP_ACCESS_TMO	NTP access time-out for the IPv6 ULA address generation, ms

	TCPIP_IPV6_ULA_NTP_VALID_WINDOW	the NTP time stamp validity window, ms if a stamp was obtained outside this interval from the moment of the request a new request will be issued

Description

Internet Control Message Protocol for IPv6 (ICMPv6) Configuration file

This file contains the IPv6 module configuration options

File Name

ipv6_config.h

Company

Microchip Technology Inc.

MAC Driver Module

This section describes the TCP/IP Stack Library MAC Driver module.

Introduction

TCP/IP Stack Library Media Access Control (MAC) Driver Module for Microchip Microcontrollers

This library provides the API of the MAC Driver module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

In the seven-layer OSI model of computer networking, media access control (MAC) data communication protocol is a sub-layer of the data link layer, which itself is layer 2. The MAC sub-layer provides addressing and channel access control mechanisms that make it possible for several terminals or network nodes to communicate within a multiple access network that incorporates a shared medium (e.g., Ethernet). The hardware that implements the MAC is referred to as a medium access controller.

The MAC sub-layer acts as an interface between the logical link control (LLC) sub-layer and the network's physical layer. The MAC layer emulates a full-duplex logical communication channel in a multi-point network. This channel may provide unicast, multicast or broadcast communication service.

Using the Library

This topic describes the basic architecture of the TCP/IP Stack MAC Driver Library and provides information and examples on its use.

Description

Interface Header File: `tcpip_mac.h`

The interface to the TCP/IP Stack MAC Driver Library is defined in the `tcpip_mac.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the TCP/IP Stack MAC Driver Library should include `tcpip.h`.

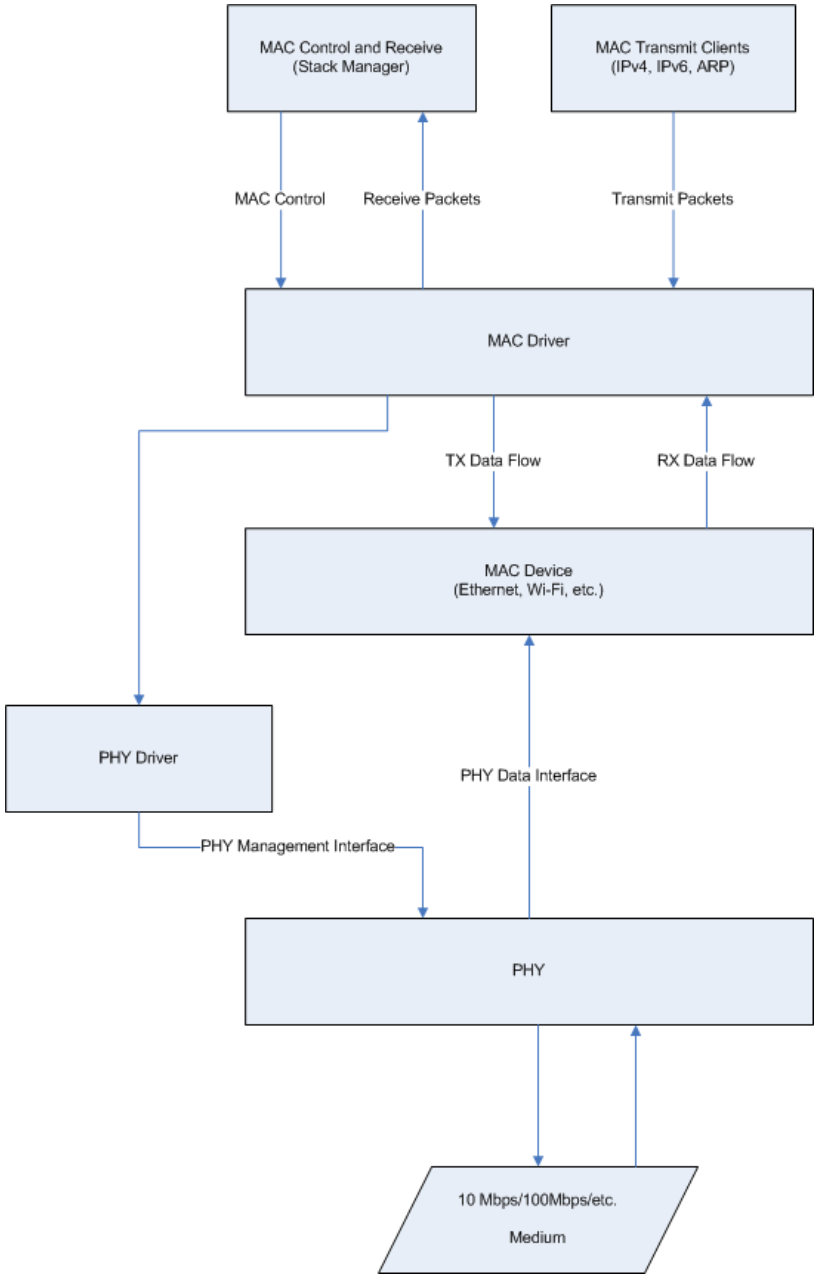
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the TCP/IP Stack MAC module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

MAC Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the MAC module.

Library Interface Section	Description
Initialization Functions	This section provides routines for configuring the MAC driver
Client Control and Status Functions	This section provides routines for client instance manipulations and MAC status
Event Functions	This section provides routines for control of the MAC driver events
Filter Functions	This section provides routines for controlling the receive MAC filtering
Data Transfer Functions	This section provides routines for transmitting and receiving data packets over the network
Data Types and Constants	This section provides various definitions describing this API

How the Library Works

This topic describes how the TCP/IP Stack MAC Library works.

Description

The MAC layer (together with the Logical Link Control – LLC) is intended to have the functionality described in the OSI model for the Data Link Layer.

There are two important functions to be performed in the MAC layer as part of the Data Link Layer:

Data Encapsulation (Transmit and Receive)

- Framing (frame boundary detection, frame synchronization)
- Addressing (control of the source and destination addresses)
- Error detection (detection of transmission errors occurring in the physical medium)

Media Access Management

- Medium allocation (collision avoidance)
- Contention resolution (collision handling)

Beside this, the receive packet filtering is another important functionality that is usually integrated in the MAC layer.

The functionality previously presented is handled by the hardware itself as an intrinsic part of the data exchange that's performed with the PHY.

From the software point of view the MAC driver exposes an API that's focused on efficient data transfer routines. Control of the driver and access to the transferred data is given by using regular driver client access functions.

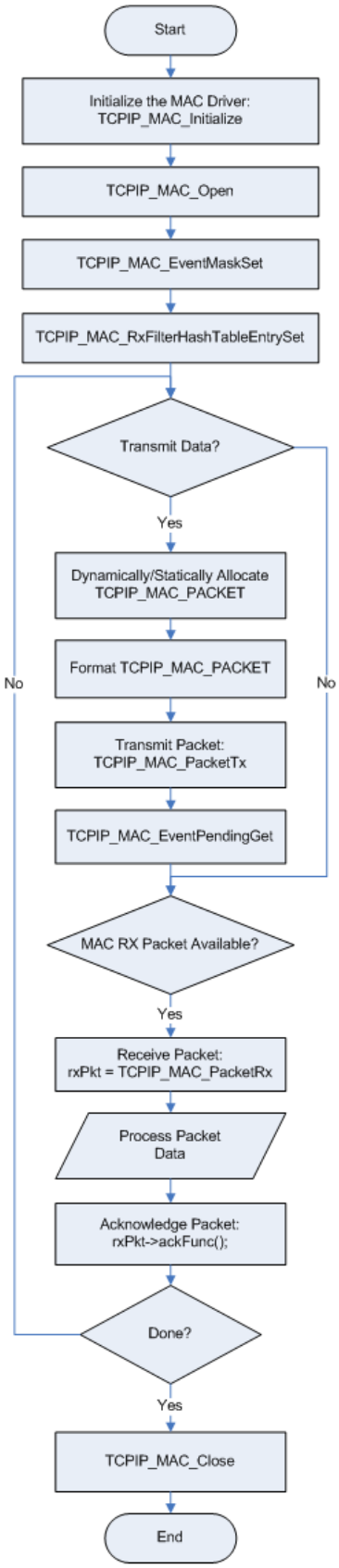
Please note that the MAC model described here corresponds to a virtual MAC driver that the TCP/IP stack uses. All the implementations of the physical MAC drivers (Ethernet, Wi-Fi, etc.) must adhere and implement this API required by the virtual MAC API. This is how the TCP/IP stack achieves virtualization of the operation over multiple network interfaces. The TCP/IP stack makes no assumption about the specifics of an actual implementation and behaves all MACs to behave identically

Core Functionality

This topic describes the core functionality of the TCP/IP Stack MAC Library.

Description

MAC Connection Flow Diagram



The MAC driver is used internally by the TCP/IP Stack Manager and its API is not normally exposed outside the stack. There is little need to interface directly with the MAC from the application level. The TCP/IP Stack provides all the data interface functions to the application by exposing the socket API (TCP and UDP) and also low level IPv4 and IPv6 API. However, the functionality of the MAC driver is important for having a thorough understanding of how the TCP/IP Stack works or for integrating the MPLAB Harmony MAC driver into a third-party TCP/IP Stack.

The MAC driver needs to be initialized by a call to TCPIP_MAC_Initialize() with both stack and module specific initialization data. The stack

initialization data contains absolutely necessary data for the MAC driver to operate (memory allocation functions, packet allocation functions, event notification functions, etc.) Note that the module initialization data could be different from one MAC module to another and, if missing, normally the MAC driver should use some default values specified in the `tcpip_mac_config.h` file.

Once the initialization process succeeds, a MAC client handle is obtained by using TCPIP_MAC_Open.

Other parameters can be set, like the active events and receive filters by calling TCPIP_MAC_EventMaskSet, TCPIP_MAC_RxFilterHashTableEntrySet, etc.

By default the MAC driver operates in interrupts and the events are reported to the TCP/IP stack by using the event notification function that's passed in at the MAC driver initialization. However, by using the TCPIP_MAC_EventPendingGet function, it is possible to use the MAC driver in a polling manner.

To send a packet, a MAC driver client has to obtain a TCPIP_MAC_PACKET data structure, either statically or dynamically. Note that the Harmony TCP/IP Stack uses dynamic packet allocation and has a packet allocation module that the Stack Manager calls for this purpose. The packet has to be properly formatted with all the required fields before being passed to the MAC driver for transmission. The MAC driver will perform a minimum sanity check and will try to transmit the packet over the network. Once the corresponding interrupt signals to the driver that the packet was (successfully/non-successfully) transmitted the MAC driver will call the packet acknowledge function. This will indicate to the owner of the packet (the module that created/allocated the packet) that the packet is no longer in use and it can be reused, discarded, etc.

The receive process is configured as part of the MAC driver initialization procedure. Usually the MAC driver needs some buffers for storing the received network traffic. How exactly this is done is driver specific and not part of this specification. The Harmony Ethernet MAC driver, for example, allocates multiple receive buffers at initialization and uses those for the whole duration of the driver life. Other implementations may choose to free the receive buffers once they are processed and allocate new ones, as needed.

The receive process is initiated by the hardware indicating through an interrupt that a packet is pending. This will trigger a notification event to the MAC client (the TCP/IP stack) and the actual packet receive function can be called. Alternatively the MAC client can call the TCPIP_MAC_EventPendingGet function which can be used in a polling manner.

To actually receive the packet the TCP/IP stack should call the TCPIP_MAC_PacketRx function. This function, besides returning a pointer to the newly received packet will provide additional info about the packet status. Also, some MAC drivers have the possibility to mark the packet being received as being unicast, multicast or broadcast.

Once a packet is obtained, the stack has to dispatch it accordingly, based on the info contained within the packet (ARP, IPv4, IPv6, etc.). Note that in the Harmony TCP/IP stack it's the Stack Manager that performs the receiving and dispatching function of the packets. The recipient of the packet will be called to process the incoming packet and may forward the packet for further processing (TCP, UDP, ICMP). Note that during all this processing time this packet can no longer be used by the MAC driver for storing newly receive data. Therefore once the final recipient of the received packet has processed the data it has to call the packet acknowledge function, so that the packet can be returned to its owner, the AMC driver in this case. If the MAC driver reuses the packet (as the Harmony Ethernet MAC driver does) or simply discards it is driver implementation specific.

Once the data transfer of packets over the network is no longer needed the TCPIP_MAC_Close function can be called and also TCPIP_MAC_Deinitialize. This is part of the normal procedure that the Harmony stack uses for shutting down and restarting a network interface dynamically.

For detailed information about the MAC layer requirements and functionality, please consult the IEEE 802.3 specifications.

Configuring the Library

Macros

	Name	Description
	TCPIP_EMAC_ETH_OPEN_FLAGS	Flags to use for the Ethernet connection A TCPIP_ETH_OPEN_FLAGS value. Set to TCPIP_ETH_OPEN_DEFAULT unless very good reason to use different value
	TCPIP_EMAC_PHY_ADDRESS	The PHY address, as configured on the board. By default all the PHYs respond to address 0
	TCPIP_EMAC_PHY_CONFIG_FLAGS	Flags to configure the MAC ->PHY connection a DRV_ETHPHY_CONFIG_FLAGS This depends on the actual connection (MII/RMII, default/alternate I/O) The DRV_ETHPHY_CFG_AUTO value will use the configuration fuses setting
	TCPIP_EMAC_PHY_LINK_INIT_DELAY	The value of the delay for the link initialization, ms This insures that the PHY is ready to transmit after it is reset A usual value is 500 ms. Adjust to your needs.
	TCPIP_EMAC_RX_BUFF_SIZE	Size of a RX packet buffer. Should be multiple of 16. This is the size of all receive packet buffers processed by the ETHC. The size should be enough to accommodate any network received packet. If the packets are larger, they will have to take multiple RX buffers and the packet manipulation is less efficient. #define TCPIP_EMAC_RX_BUFF_SIZE 512 Together with TCPIP_EMAC_RX_DEDICATED_BUFFERS it has impact on TCPIP_STACK_DRAM_SIZE setting.
	TCPIP_EMAC_RX_DESCRIPTORS	Number of the RX descriptors to be created. If not using the run time replenish mechanism (see below) it should match the number of dedicated buffers: TCPIP_EMAC_RX_DEDICATED_BUFFERS; Otherwise it should be bigger than the sum of dedicated + non-dedicated buffers: TCPIP_EMAC_RX_DESCRIPTORS > TCPIP_EMAC_RX_DEDICATED_BUFFERS + replenish_buffers

	TCPIP_EMAC_RX_FRAGMENTS	MAC maximum number of supported fragments. Based on the values of TCPIP_EMAC_RX_MAX_FRAME and TCPIP_EMAC_RX_BUFF_SIZE an incoming frame may span multiple RX buffers (fragments). Note that excessive fragmentation leads to performance degradation. The default and recommended value should be 1. #define TCPIP_EMAC_RX_FRAGMENTS 1 Alternatively you can use the calculation of the number of fragments based on the selected RX sizes:
	TCPIP_EMAC_RX_MAX_FRAME	Maximum MAC supported RX frame size. Any incoming ETH frame that's longer than this size will be discarded. The default value is 1536 (allows for VLAN tagged frames, although the VLAN tagged frames are discarded). Normally there's no need to touch this value unless you know exactly the maximum size of the frames you want to process or you need to control packets fragmentation (together with the TCPIP_EMAC_RX_BUFF_SIZE).
	TCPIP_EMAC_TX_DESCRIPTORS	Number of the TX descriptors to be created. Because a TCP packet can span at most 3 buffers, the value should always be >= 4 The amount of memory needed per descriptor is not high (around 24 bytes) so when high MAC TX performance is needed make sure that this number is >= 8.
	TCPIP_EMAC_RX_DEDICATED_BUFFERS	Number of MAC dedicated RX packet buffers. These buffers are always owned by the MAC. Note that the MAC driver allocates these buffers for storing the incoming network packets. The bigger the storage capacity, the higher data throughput can be obtained. Note that these packet buffers are allocated from the private TCP/IP heap that is specified by the TCPIP_STACK_DRAM_SIZE setting.
	TCPIP_EMAC_RX_INIT_BUFFERS	Number of non-dedicated buffers for the MAC initialization Buffers allocated at the MAC driver initialization.
	TCPIP_EMAC_RX_LOW_FILL	Number of RX buffers to allocate when below threshold condition is detected. If 0, the MAC driver will allocate (scheduled buffers - rxThres) If !0, the MAC driver will allocate exactly TCPIP_EMAC_RX_LOW_FILL buffers
	TCPIP_EMAC_RX_LOW_THRESHOLD	Minumum threshold for the buffer replenish process. Whenever the number of RX scheduled buffers is <= than this threshold the MAC driver will allocate new non-dedicated buffers (meaning that they will be released to the TCP/IP heap once they are processed). Setting this value to 0 disables the buffer replenishing process.
	TCPIP_EMAC_RX_FILTERS	MAC RX Filters These filters define the packets that are accepted and rejected by the MAC driver Adjust to your needs The default value allows the processing of unicast, multicast and broadcast packets that have a valid CRC

Description

The configuration of the MAC TCP/IP Stack is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the MAC TCP/IP Stack. Based on the selections made, the MAC TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the MAC TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_EMAC_ETH_OPEN_FLAGS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_ETH_OPEN_FLAGS (TCPIP_ETH_OPEN_DEFAULT)
```

Description

Flags to use for the Ethernet connection A TCPIP_ETH_OPEN_FLAGS value. Set to TCPIP_ETH_OPEN_DEFAULT unless very good reason to use different value

TCPIP_EMAC_PHY_ADDRESS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_PHY_ADDRESS (0)
```

Description

The PHY address, as configured on the board. By default all the PHYs respond to address 0

TCPIP_EMAC_PHY_CONFIG_FLAGS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_PHY_CONFIG_FLAGS (DRV_ETHPHY_CFG_AUTO)
```

Description

Flags to configure the MAC ->PHY connection a DRV_ETHPHY_CONFIG_FLAGS This depends on the actual connection (MII/RMII, default/alternate I/O) The DRV_ETHPHY_CFG_AUTO value will use the configuration fuses setting

TCPIP_EMAC_PHY_LINK_INIT_DELAY Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_PHY_LINK_INIT_DELAY (500)
```

Description

The value of the delay for the link initialization, ms This insures that the PHY is ready to transmit after it is reset A usual value is 500 ms. Adjust to your needs.

TCPIP_EMAC_RX_BUFF_SIZE Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_BUFF_SIZE 1536
```

Description

Size of a RX packet buffer. Should be multiple of 16. This is the size of all receive packet buffers processed by the ETHC. The size should be enough to accommodate any network received packet. If the packets are larger, they will have to take multiple RX buffers and the packet manipulation is less efficient. #define TCPIP_EMAC_RX_BUFF_SIZE 512 Together with TCPIP_EMAC_RX_DEDICATED_BUFFERS it has impact on TCPIP_STACK_DRAM_SIZE setting.

TCPIP_EMAC_RX_DESCRIPTORS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_DESCRIPTORS 10
```

Description

Number of the RX descriptors to be created. If not using the run time replenish mechanism (see below) it should match the number of dedicated buffers: TCPIP_EMAC_RX_DEDICATED_BUFFERS; Otherwise it should be bigger than the sum of dedicated + non-dedicated buffers: TCPIP_EMAC_RX_DESCRIPTORS > TCPIP_EMAC_RX_DEDICATED_BUFFERS + replenish_buffers

TCPIP_EMAC_RX_FRAGMENTS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_FRAGMENTS ((TCPIP_EMAC_RX_MAX_FRAME + (TCPIP_EMAC_RX_BUFF_SIZE -1 )) /
(TCPIP_EMAC_RX_BUFF_SIZE))
```

Description

MAC maximum number of supported fragments. Based on the values of TCPIP_EMAC_RX_MAX_FRAME and TCPIP_EMAC_RX_BUFF_SIZE an incoming frame may span multiple RX buffers (fragments). Note that excessive fragmentation leads to performance degradation. The default and recommended value should be 1. #define TCPIP_EMAC_RX_FRAGMENTS 1 Alternatively you can use the calculation of the number of fragments based on the selected RX sizes:

TCPIP_EMAC_RX_MAX_FRAME Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_MAX_FRAME 1536
```

Description

Maximum MAC supported RX frame size. Any incoming ETH frame that's longer than this size will be discarded. The default value is 1536 (allows for VLAN tagged frames, although the VLAN tagged frames are discarded). Normally there's no need to touch this value unless you know exactly the maximum size of the frames you want to process or you need to control packets fragmentation (together with the TCPIP_EMAC_RX_BUFF_SIZE).

Remarks

Always multiple of 16.

TCPIP_EMAC_TX_DESCRIPTORS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_TX_DESCRIPTORS 8
```

Description

Number of the TX descriptors to be created. Because a TCP packet can span at most 3 buffers, the value should always be >= 4 The amount of memory needed per descriptor is not high (around 24 bytes) so when high MAC TX performance is needed make sure that this number is >= 8.

TCPIP_EMAC_RX_DEDICATED_BUFFERS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_DEDICATED_BUFFERS 4
```

Description

Number of MAC dedicated RX packet buffers. These buffers are always owned by the MAC. Note that the MAC driver allocates these buffers for storing the incoming network packets. The bigger the storage capacity, the higher data throughput can be obtained. Note that these packet buffers are allocated from the private TCP/IP heap that is specified by the TCPIP_STACK_DRAM_SIZE setting.

TCPIP_EMAC_RX_INIT_BUFFERS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_INIT_BUFFERS 0
```

Description

Number of non-dedicated buffers for the MAC initialization Buffers allocated at the MAC driver initialization.

TCPIP_EMAC_RX_LOW_FILL Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_LOW_FILL 2
```

Description

Number of RX buffers to allocate when below threshold condition is detected. If 0, the MAC driver will allocate (scheduled buffers - rxThres) If !0, the MAC driver will allocate exactly TCPIP_EMAC_RX_LOW_FILL buffers

TCPIP_EMAC_RX_LOW_THRESHOLD Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_LOW_THRESHOLD 1
```

Description

Minumum threshold for the buffer replenish process. Whenever the number of RX scheduled buffers is <= than this threshold the MAC driver will allocate new non-dedicated buffers (meaning that they will be released to the TCP/IP heap once they are processed). Setting this value to 0 disables the buffer replenishing process.

TCPIP_EMAC_RX_FILTERS Macro

File

tcpip_mac_config.h

C

```
#define TCPIP_EMAC_RX_FILTERS (TCPIP_MAC_RX_FILTER_TYPE_DEFAULT)
```

Description

MAC RX Filters These filters define the packets that are accepted and rejected by the MAC driver Adjust to your needs The default value allows the processing of unicast, multicast and broadcast packets that have a valid CRC

Building the Library

This section lists the files that are available in the MAC Driver module of the TCP/IP Stack Library.

Description

The MAC Driver module is part of the core functionality of the TCP/IP Stack Library. Please refer to the Building the Library topic in the TCP/IP Stack Library section for build information.

Library Interface

a) Initialization Functions

	Name	Description
	TCPIP_MAC_Open	MAC driver open function.
	TCPIP_MAC_Close	MAC driver close function.
	TCPIP_MAC_Initialize	MAC driver initialization function.
	TCPIP_MAC_Deinitialize	MAC driver deinitialization function.
	TCPIP_MAC_Reinitialize	MAC driver reinitialization function.

b) Client Control and Status Functions

	Name	Description
	TCPIP_MAC_ParametersGet	MAC parameter get function.
	TCPIP_MAC_StatisticsGet	Gets the current MAC statistics.
	TCPIP_MAC_LinkCheck	MAC link checking function.
	TCPIP_MAC_Status	Provides the current status of the MAC driver module.
	TCPIP_MAC_Tasks	Maintains the MAC driver's state machine.
	TCPIP_MAC_Process	MAC periodic processing function.
	TCPIP_MAC_RegisterStatisticsGet	Gets the current MAC hardware statistics registers.
	TCPIP_MAC_ConfigGet	Gets the current MAC driver configuration.

c) Event Functions

	Name	Description
	TCPIP_MAC_EventAcknowledge	This function acknowledges a previously reported MAC event.
	TCPIP_MAC_EventMaskSet	MAC events report enable/disable function.
	TCPIP_MAC_EventPendingGet	Returns the currently pending MAC events.

d) Filter Functions

	Name	Description
	TCPIP_MAC_RxFilterHashTableEntrySet	Sets the current MAC hash table receive filter.

e) Data Transfer Functions

	Name	Description
	TCPIP_MAC_PacketTx	MAC driver transmit function.
	TCPIP_MAC_PacketRx	A packet is returned if such a pending packet exists.

f) Data Types and Constants

	Name	Description
	TCPIP_MAC_ACTION	Network interface action for initialization/deinitialization
	TCPIP_MAC_ADDR	Definition of a MAC address.
	TCPIP_MAC_DATA_SEGMENT	A data segment that's part of a TX/RX packet.
	TCPIP_MAC_ETHERNET_HEADER	Definition of a MAC frame header.
	TCPIP_MAC_EVENT	Defines the possible MAC event types.
	TCPIP_MAC_EventF	MAC event notification handler.
	TCPIP_MAC_HANDLE	Handle to a MAC driver.
	TCPIP_MAC_HEAP_CallocF	MAC allocation function prototype.
	TCPIP_MAC_HEAP_FreeF	MAC allocation function prototype.
	TCPIP_MAC_HEAP_HANDLE	A handle used for memory allocation functions.
	TCPIP_MAC_HEAP_MallocF	MAC allocation function prototype.
	TCPIP_MAC_MODULE_CTRL	Data structure that's passed to the MAC at the initialization time.
	TCPIP_MAC_PACKET	Forward reference to a MAC packet.
	TCPIP_MAC_PACKET_ACK_FUNC	Prototype of a MAC packet acknowledge function.
	TCPIP_MAC_PACKET_FLAGS	Flags belonging to MAC packet.
	TCPIP_MAC_PACKET_RX_STAT	Status of a received packet.
	TCPIP_MAC_PKT_ACK_RES	List of MAC return codes for a packet acknowledge function.
	TCPIP_MAC_PKT_AckF	MAC packet acknowledge function prototype.
	TCPIP_MAC_PKT_AllocF	MAC packet allocation function prototype.
	TCPIP_MAC_PKT_FreeF	MAC packet free function prototype.
	TCPIP_MAC_POWER_MODE	This is type TCPIP_MAC_POWER_MODE.
	TCPIP_MAC_PROCESS_FLAGS	List of the MAC processing flags.
	TCPIP_MAC_RES	List of return codes from MAC functions.
	TCPIP_MAC_SEGMENT_FLAGS	Flags belonging to MAC data segment.
	TCPIP_MODULE_MAC_MRF24W_CONFIG	This is type TCPIP_MODULE_MAC_MRF24W_CONFIG.
	TCPIP_MODULE_MAC_PIC32INT_CONFIG	Data that's passed to the MAC at initialization time.
	TCPIP_MAC_RX_STATISTICS	MAC receive statistics data gathered at run time.

	TCPIP_MAC_TX_STATISTICS	MAC transmit statistics data gathered at run time.
	TCPIP_MAC_INIT	Contains all the data necessary to initialize the MAC device.
	TCPIP_MAC_PARAMETERS	Data structure that tells the MAC run time parameters.
	TCPIP_MAC_PKT_AckFDbg	This is type TCPIP_MAC_PKT_AckFDbg.
	TCPIP_MAC_PKT_AllocFDbg	This is type TCPIP_MAC_PKT_AllocFDbg.
	TCPIP_MAC_PKT_FreeFDbg	This is type TCPIP_MAC_PKT_FreeFDbg.
	TCPIP_MAC_HEAP_CallocFDbg	This is type TCPIP_MAC_HEAP_CallocFDbg.
	TCPIP_MAC_HEAP_FreeFDbg	This is type TCPIP_MAC_HEAP_FreeFDbg.
	TCPIP_MAC_HEAP_MallocFDbg	This is type TCPIP_MAC_HEAP_MallocFDbg.
	TCPIP_MAC_STATISTICS_REG_ENTRY	Describes a MAC hardware statistics register.
	TCPIP_MAC_TYPE	List of the MAC types.
	TCPIP_MAC_SYNCH_REQUEST	Defines the possible MAC synchronization request types.
	TCPIP_MAC_SynchReqF	MAC synchronization request function definition.
	TCPIP_MODULE_MAC_ID	IDs of the MAC module supported by the stack.
	TCPIP_MODULE_MAC_MRF24WN_CONFIG	This is type TCPIP_MODULE_MAC_MRF24WN_CONFIG.
	TCPIP_MAC_RX_FILTER_TYPE	Defines the possible MAC RX filter types.
	TCPIP_MAC_LINK_MTU	MTU size correspondig to the MAC types.

Description

This section describes the Application Programming Interface (API) functions of the MAC Driver module.

Refer to each section for a detailed description.

a) Initialization Functions

TCPIP_MAC_Open Function

MAC driver open function.

File

tcpip_mac.h

C

```
DRV_HANDLE TCPIP_MAC_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);
```

Returns

- Valid handle - if the open function succeeded
- DRV_HANDLE_INVALID - if an error occurs

Description

This is the function that opens an instance of the MAC. Once a MAC client handle is obtained all the MAC functions can be accessed using that handle.

Remarks

The intent parameter is not used in the current implementation and is maintained only for compatibility with the generic driver Open function signature.

Preconditions

TCPIP_MAC_Initialize should have been called.

Parameters

Parameters	Description
drvIndex	identifier for the driver instance to be opened.
intent	Zero or more of the values from the enumeration DRV_IO_INTENT ORed together to indicate the intended use of the driver

Function

DRV_HANDLE TCPIP_MAC_Open(const SYS_MODULE_INDEX drvIndex, const DRV_IO_INTENT intent);

TCPIP_MAC_Close Function

MAC driver close function.

File

tcpip_mac.h

C

```
void TCPIP_MAC_Close(DRV_HANDLE hMac);
```

Returns

None.

Description

This is the function that closes an instance of the MAC. All per client data is released and the handle can no longer be used after this function is called.

Remarks

None.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	MAC client handle obtained by a call to TCPIP_MAC_Open.

Function

void TCPIP_MAC_Close(DRV_HANDLE hMac);

TCPIP_MAC_Initialize Function

MAC driver initialization function.

File

tcpip_mac.h

C

```
SYS_MODULE_OBJ TCPIP_MAC_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

- Valid handle to a driver object - if successful
- SYS_MODULE_OBJ_INVALID - if initialization failed

Description

MAC Initialize function. SYS_MODULE_OBJ TCPIP_MAC_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);

This is the function that initializes the MAC. It is called by the stack as a result of one interface going up.

Remarks

If this function fails, the stack won't turn up that interface. If the operation needs to wait for the hardware, the initialization function can return a pending code.

The returned object must be passed as argument to TCPIP_MAC_Reinitialize, TCPIP_MAC_Deinitialize, TCPIP_MAC_Tasks and TCPIP_MAC_Status routines.

Preconditions

None.

Parameters

Parameters	Description
index	Index for the MAC driver instance to be initialized

init	pointer to TCPIP_MAC_INIT initialization data containing: • macControl - Stack prepared data. • moduleData - Driver specific. Dependent on the MAC type. For PIC32 MAC driver, the TCPIP_MODULE_MAC_PIC32INT_CONFIG is used.

TCPIP_MAC_Deinitialize Function

MAC driver deinitialization function.

File

tcpip_mac.h

C

```
void TCPIP_MAC_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

MAC De-Initialize function. void TCPIP_MAC_Deinitialize(SYS_MODULE_OBJ object);

This is the function that deinitializes the MAC. It is called by the stack as a result of one interface going down.

Remarks

None.

Preconditions

TCPIP_MAC_Initialize should have been called.

Parameters

Parameters	Description
object	Driver object handle, returned from TCPIP_MAC_Initialize

TCPIP_MAC_Reinitialize Function

MAC driver reinitialization function.

File

tcpip_mac.h

C

```
void TCPIP_MAC_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

MAC Re-Initialize function. void TCPIP_MAC_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);

This is the function that re-initializes the MAC. It is called by the stack as a result of the system changing power modes. Allows reinitialization of the MAC with different power modes, etc.

Remarks

This function is optional and is not currently implemented.

Preconditions

TCPIP_MAC_Initialize should have been called.

Parameters

Parameters	Description
object	Driver object handle, returned from the TCPIP_MAC_Initialize function

init	pointer to TCPIP_MAC_INIT initialization data containing: • macControl - Stack prepared data. • moduleData - Driver specific. Dependent on the MAC type. For PIC32 MAC driver, the TCPIP_MODULE_MAC_PIC32INT_CONFIG is used.

b) Client Control and Status Functions

TCPIP_MAC_ParametersGet Function

MAC parameter get function.

File

tcpip_mac.h

C

```
TCPIP_MAC_RES TCPIP_MAC_ParametersGet(DRV_HANDLE hMac, TCPIP_MAC_PARAMETERS* pMacParams);
```

Returns

- TCPIP_MAC_RES_OK - if pMacParams updated properly
- TCPIP_MAC_RES error code - if processing failed for some reason

Description

This is a function that returns the run time parameters of the MAC driver.

Remarks

None.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
pMacParams	address to store the MAC parameters

Function

TCPIP_MAC_RES TCPIP_MAC_ParametersGet(DRV_HANDLE hMac, TCPIP_MAC_PARAMETERS* pMacParams);

TCPIP_MAC_StatisticsGet Function

Gets the current MAC statistics.

File

tcpip_mac.h

C

```
TCPIP_MAC_RES TCPIP_MAC_StatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_RX_STATISTICS* pRxStatistics,
TCPIP_MAC_TX_STATISTICS* pTxStatistics);
```

Returns

- TCPIP_MAC_RES_OK - if all processing went on OK
- TCPIP_MAC_RES_OP_ERR error code - if function not supported by the driver

Description

This function will get the current value of the statistic counters maintained by the MAC driver.

Remarks

The reported values are info only and change dynamically.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
pRxStatistics	pointer to a TCPIP_MAC_RX_STATISTICS that will receive the current RX statistics counters. Can be NULL if not needed.
pTxStatistics	pointer to a TCPIP_MAC_TX_STATISTICS that will receive the current TX statistics counters. Can be NULL if not needed.

Function

TCPIP_MAC_RES TCPIP_MAC_StatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_RX_STATISTICS* pRxStatistics,
TCPIP_MAC_TX_STATISTICS* pTxStatistics);

TCPIP_MAC_LinkCheck Function

MAC link checking function.

File

tcpip_mac.h

C

```
bool TCPIP_MAC_LinkCheck(DRV_HANDLE hMac);
```

Returns

- true - if the link is up
- false - if the link is down

Description

This is a function that returns the current status of the link.

Remarks

The stack will call this function periodically. Therefore this function is not required to initiate a new PHY transaction and to wait for the result. It can just initiate a new PHY transaction and return immediately the result of the previous transaction.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client

Function

bool TCPIP_MAC_LinkCheck(DRV_HANDLE hMac);

TCPIP_MAC_Status Function

Provides the current status of the MAC driver module.

File

tcpip_mac.h

C

```
SYS_STATUS TCPIP_MAC_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that any previous module operation for the specified module has completed
- SYS_STATUS_BUSY - Indicates that a previous module operation for the specified module has not yet completed
- SYS_STATUS_ERROR - Indicates that the specified module is in an error state

Description

This function provides the current status of the MAC driver module.

Remarks

Any value greater than SYS_STATUS_READY is also a normal running state in which the driver is ready to accept new operations.

SYS_STATUS_BUSY - Indicates that the driver is busy with a previous system level operation and cannot start another

SYS_STATUS_ERROR - Indicates that the driver is in an error state Any value less than SYS_STATUS_ERROR is also an error state.

SYS_MODULE_DEINITIALIZED - Indicates that the driver has been deinitialized

This operation can be used to determine when any of the driver's module level operations has completed.

If the status operation returns SYS_STATUS_BUSY, then a previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This function will NEVER block waiting for hardware.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

The TCPIP_MAC_Initialize function must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from TCPIP_MAC_Initialize
SYS_STATUS          status;

status = TCPIP_MAC_Status(object);
if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from TCPIP_MAC_Initialize

Function

SYS_STATUS TCPIP_MAC_Status (SYS_MODULE_OBJ object)

TCPIP_MAC_Tasks Function

Maintains the MAC driver's state machine.

File

tcpip_mac.h

C

```
void TCPIP_MAC_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function is used to maintain the driver's internal state machine

Remarks

This function is normally not called directly by an application. It is called by the system's Tasks routine (SYS_Tasks)

This function will never block or access any resources that may cause it to block.

Preconditions

The TCPIP_MAC_Initialize routine must have been called for the specified MAC driver instance.

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from TCPIP_MAC_Initialize)

Function

void TCPIP_MAC_Tasks(SYS_MODULE_OBJ object)

TCPIP_MAC_Process Function

MAC periodic processing function.

File

tcpip_mac.h

C

```
TCPIP_MAC_RES TCPIP_MAC_Process(DRV_HANDLE hMac);
```

Returns

- TCPIP_MAC_RES_OK - if all processing went on OK
- TCPIP_MAC_RES error code - if processing failed for some reason

Description

This is a function that allows for internal processing by the MAC driver. It is meant for processing that cannot be done from within ISR.

Normally this function will be called in response to an TX and/or RX event signaled by the driver. This is specified by the MAC driver using the TCPIP_MAC_PARAMETERS::processFlags.

An alternative approach is that the MAC driver uses a system service to create a timer signal that will call the TCPIP_MAC_Process on a periodic basis.

Remarks

Some of the processing that this function is intended for:

- The MAC driver can process its pending TX queues (although it could do that from within the TX ISR)
- RX buffers replenishing. If the number of packets in the RX queue falls below a specified limit, the MAC driver can use this function to allocate some extra RX packets. Similarly, if there are too many allocated RX packets, the MAC driver can free some of them.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client

Function

TCPIP_MAC_RES TCPIP_MAC_Process(DRV_HANDLE hMac);

TCPIP_MAC_RegisterStatisticsGet Function

Gets the current MAC hardware statistics registers.

File

tcpip_mac.h

C

```
TCPIP_MAC_RES TCPIP_MAC_RegisterStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_STATISTICS_REG_ENTRY* pRegEntries,
int nEntries, int* pHwEntries);
```

Returns

- TCPIP_MAC_RES_OK - if all processing went on OK
- TCPIP_MAC_RES_OP_ERR error code - if function not supported by the driver

Description

This function will get the current value of the statistic registers maintained by the MAC hardware.

Remarks

The reported values are info only and change dynamically.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
pRegStatistics	pointer to a pRegEntries that will receive the current hardware statistics registers values. Can be 0, if only the number of supported hardware registers is requested
nEntries	provides the number of TCPIP_MAC_STATISTICS_REG_ENTRY structures present in pRegEntries Can be 0, if only the number of supported hardware registers is requested The register entries structures will be filled by the driver, up to the supported hardware registers.
pHwEntries	pointer to an address to store the number of the statistics registers that the hardware supports It is updated by the driver. Can be 0 if not needed

Function

TCPIP_MAC_RES TCPIP_MAC_RegisterStatisticsGet(DRV_HANDLE hMac, TCPIP_MAC_STATISTICS_REG_ENTRY* pRegEntries, int nEntries, int* pHwEntries);

TCPIP_MAC_ConfigGet Function

Gets the current MAC driver configuration.

File

tcpip_mac.h

C

```
size_t TCPIP_MAC_ConfigGet(DRV_HANDLE hMac, void* configBuff, size_t buffSize, size_t* pConfigSize);
```

Returns

The number of bytes copied into the supplied storage buffer

Description

This function will get the current MAC driver configuration and store it into a supplied buffer.

Remarks

None.

Preconditions

TCPIP_MAC_Initialize must have been called to set up the driver. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
configBuff	pointer to a buffer to store the configuration. Can be NULL if not needed.
buffSize	size of the supplied buffer
pConfigSize	address to store the number of bytes needed for the storage of the MAC configuration. Can be NULL if not needed.

Function

size_t TCPIP_MAC_ConfigGet(DRV_HANDLE hMac, void* configBuff, size_t buffSize, size_t* pConfigSize);

c) Event Functions

TCPIP_MAC_EventAcknowledge Function

This function acknowledges a previously reported MAC event.

File

tcpip_mac.h

C

```
bool TCPIP_MAC_EventAcknowledge(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents);
```

Returns

- true - if events acknowledged
- false - if no events to be acknowledged

Description

This function acknowledges and re-enables processed events. Multiple events can be "ORed" together as they are processed together. The events acknowledged by this function should be the events that have been retrieved from the MAC by calling TCPIP_MAC_EventPendingGet or have been passed to the MAC client by using the notification handler. Once the events are acknowledged they will be re-enabled.

Remarks

All events should be acknowledged, in order to be re-enabled.

Some events are fatal errors and should not be acknowledged (TCPIP_MAC_EV_RX_BUSERR, TCPIP_MAC_EV_TX_BUSERR). Stack re-initialization is needed under such circumstances.

Some events are just system/application behavior and they are intended only as simple info (TCPIP_MAC_EV_RX_OVFLOW, TCPIP_MAC_EV_RX_BUFNA, TCPIP_MAC_EV_TX_ABORT, TCPIP_MAC_EV_RX_ACT).

The TCPIP_MAC_EV_RX_FWMARK and TCPIP_MAC_EV_RX_EWMARK events are part of the normal flow control operation (if auto flow control was enabled). They should be enabled alternatively, if needed.

The events are persistent. They shouldn't be re-enabled unless they have been processed and the condition that generated them was removed. Re-enabling them immediately without proper processing will have dramatic effects on the system performance.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Example

```
TCPIP_MAC_EventAcknowledge( hMac, newMacEvents );
```

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
macEvents	MAC events that are acknowledged and re-enabled

Function

bool TCPIP_MAC_EventAcknowledge(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents);

TCPIP_MAC_EventMaskSet Function

MAC events report enable/disable function.

File

tcpip_mac.h

C

```
bool TCPIP_MAC_EventMaskSet(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents, bool enable);
```

Returns

- true - if operation succeeded
- false - if some error occurred and the operation failed

Description

This is a function that enables or disables the events to be reported to the MAC client (TCP/IP stack).

All events that are to be enabled will be added to the notification process. All events that are to be disabled will be removed from the notification process. The stack (or stack user) has to catch the events that are notified and process them. After that the stack should call TCPIP_MAC_EventAcknowledge() so that the events can be re-enabled.

Remarks

Multiple events can be "ORed" together.

The event notification system enables the user of the MAC and of the stack to call into the stack for processing only when there are relevant events rather than being forced to periodically call from within a loop.

If the notification events are null the interrupt processing will be disabled. Otherwise the event notification will be enabled and the interrupts relating to the requested events will be enabled.

Note that once an event has been caught by the MAC and reported through the notification handler it may be disabled until the TCPIP_MAC_EventAcknowledge is called.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Example

```
TCPIP_MAC_EventMaskSet( hMac, TCPIP_MAC_EV_RX_OVFLOW | TCPIP_MAC_EV_RX_BUFNA, true );
```

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
macEvents	events that the MAC client wants to add/delete for notification
enable	if true, the events will be enabled, else disabled

Function

bool TCPIP_MAC_EventMaskSet(DRV_HANDLE hMac, TCPIP_MAC_EVENT macEvents, bool enable);

TCPIP_MAC_EventPendingGet Function

Returns the currently pending MAC events.

File

tcpip_mac.h

C

```
TCPIP_MAC_EVENT TCPIP_MAC_EventPendingGet(DRV_HANDLE hMac);
```

Returns

The currently stack pending events.

Description

This function returns the currently MAC pending events. Multiple events will be "ORed" together as they accumulate. MAC processing is needed whenever a transmission related event is present (TCPIP_MAC_EV_RX_PKTPEND, TCPIP_MAC_EV_TX_DONE). Other, non critical events, may be passed to an user for informational purposes. All events have to be eventually acknowledged if re-enabling is needed.

Remarks

This is the preferred method to get the current pending MAC events. Even with a notification handler in place it's better to use this function to get the current pending events rather than using the events passed by the notification handler which could be stale.

The returned value is just a momentary value. The pending events can change any time.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Example

```
TCPIP_MAC_EVENT currEvents = TCPIP_MAC_EventPendingGet( hMac);
```

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client

Function

TCPIP_MAC_EVENT TCPIP_MAC_EventPendingGet(DRV_HANDLE hMac);

d) Filter Functions

TCPIP_MAC_RxFilterHashTableEntrySet Function

Sets the current MAC hash table receive filter.

File

tcpip_mac.h

C

```
TCPIP_MAC_RES TCPIP_MAC_RxFilterHashTableEntrySet(DRV_HANDLE hMac, TCPIP_MAC_ADDR* DestMACAddr);
```

Returns

- TCPIP_MAC_RES_OK - if all processing went on OK.
- TCPIP_MAC_RES error code - if function failed for some reason.

Description

This function sets the MAC hash table filtering to allow packets sent to DestMACAddr to be received. It calculates a CRC-32 using polynomial 0x4C11DB7 over the 6 byte MAC address and then, using bits 28:23 of the CRC, will set the appropriate bits in the hash table filter registers.

The function will enable/disable the Hash Table receive filter if needed.

Remarks

There is no way to individually remove destination MAC addresses from the hash table since it is possible to have a hash collision and therefore multiple MAC addresses relying on the same hash table bit.

A workaround is to have the stack store each enabled MAC address and to perform the comparison at run time.

A call to TCPIP_MAC_RxFilterHashTableEntrySet() using a 00-00-00-00-00-00 destination MAC address, which will clear the entire hash table and disable the hash table filter. This will allow the receive of all packets, regardless of their destination.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
DestMACAddr	destination MAC address (6 bytes) to allow through the Hash Table Filter. If DestMACAddr is set to 00-00-00-00-00-00, then the hash table will be cleared of all entries and the filter will be disabled.

Function

TCPIP_MAC_RES TCPIP_MAC_RxFilterHashTableEntrySet(DRV_HANDLE hMac, TCPIP_MAC_ADDR* DestMACAddr);

e) Data Transfer Functions

TCPIP_MAC_PacketTx Function

MAC driver transmit function.

File

tcpip_mac.h

C

```
TCPIP_MAC_RES TCPIP_MAC_PacketTx(DRV_HANDLE hMac, TCPIP_MAC_PACKET * ptrPacket);
```

Returns

- TCPIP_MAC_RES_OK - if all processing went on OK

- TCPIP_MAC_RES error code - if processing failed for some reason

Description

This is the MAC transmit function. Using this function a packet is submitted to the MAC driver for transmission.

Remarks

A success code returned from this function signals only that the packet was successfully scheduled for transmission over the interface and not that the packet was actually transmitted. An event will be triggered when the packet is transmitted.

The MAC driver has to support internal queuing. Since the TCPIP_MAC_PACKET data structure contains internal queuing members the MAC can queue the packet at no expense. Therefore a packet is to be rejected only if it's not properly formatted. Otherwise it has to be scheduled for transmission in an internal MAC queue.

Once the packet is scheduled for transmission the MAC driver has to set the TCPIP_MAC_PKT_FLAG_QUEUED flag so that the stack is aware that this packet is under processing and cannot be modified.

Once the packet is transmitted, the TCPIP_MAC_PKT_FLAG_QUEUED has to be cleared, the proper packet acknowledgment result (ackRes) has to be set and the packet acknowledgment function (ackFunc) has to be called. It is implementation dependent if all these steps are implemented as part of the ackFunc itself or as discrete steps.

On 32-bit machines, the 1st segment payload of a packet is allocated so that it is always 32-bit aligned and its size is 32 bits multiple. The segLoadOffset adds to the payload address and insures that the network layer data is 32-bit aligned.

PIC32 MAC driver specific : the driver checks that the segLoadOffset >= 2. See notes for the segLoadOffset member.

The packet is not required to contain the Frame Check Sequence (FCS/CRC32) field. The MAC driver/controller will insert that field itself, if it's required.

The MAC driver is required to support the transmission of multiple chained packets.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	MAC client handle obtained by a call to TCPIP_MAC_Open.
ptrPacket	pointer to a TCPIP_MAC_PACKET that's completely formatted and ready to be transmitted over the network

Function

TCPIP_MAC_RES TCPIP_MAC_PacketTx(DRV_HANDLE hMac, TCPIP_MAC_PACKET * ptrPacket);

TCPIP_MAC_PacketRx Function

A packet is returned if such a pending packet exists.

File

tcpip_mac.h

C

```
TCPIP_MAC_PACKET* TCPIP_MAC_PacketRx(DRV_HANDLE hMac, TCPIP_MAC_RES* pRes, const TCPIP_MAC_PACKET_RX_STAT**
ppPktStat);
```

Returns

- Valid pointer to an available RX packet
- 0 if no packet pending/available

Description

This is the MAC receive function.

Once a pending packet is available in the MAC driver internal RX queues this function will dequeue the packet and hand it over to the MAC driver's client - i.e., the stack - for further processing.

The flags for a RX packet have to be updated by the MAC driver: TCPIP_MAC_PKT_FLAG_RX has to be set. If the MAC supports it, it should set:

- TCPIP_MAC_PKT_FLAG_UNICAST has to be set if that packet is a unicast packet
- TCPIP_MAC_PKT_FLAG_BCAST has to be set if that packet is a broadcast packet
- TCPIP_MAC_PKT_FLAG_MCAST has to be set if that packet is a multicast packet
- TCPIP_MAC_PKT_FLAG_QUEUED has to be set
- TCPIP_MAC_PKT_FLAG_SPLIT has to be set if the packet has multiple data segments

Additional information about the packet is available by providing the pRes and ppPktStat fields.

Remarks

The MAC driver should dequeue and return to the caller just one single packet, and not multiple chained packets!

Once the higher level layers in the stack are done with processing the RX packet, they have to call the corresponding packet acknowledgment function that tells the owner of that packet that it can resume control of that packet.

Once the stack modules are done processing the RX packets and the acknowledge function is called it is up to the driver design to reuse the RX packets, or simply return them to the pool they were allocated from (assuming that some sort of allocation is implemented). This document makes no requirement about how the MAC RX packets are obtained, using dynamic or static allocation techniques. This is up to the design of the MAC.

The MAC driver can use the TCPIP_MAC_Process() for obtaining new RX packets if needed.

Not all the MACs have hardware support for the received packet status. If the MAC driver cannot supply the TCPIP_MAC_PACKET_RX_STAT info, it should set the ppPktStat to 0.

Preconditions

TCPIP_MAC_Initialize should have been called. TCPIP_MAC_Open should have been called to obtain a valid handle.

Parameters

Parameters	Description
hMac	handle identifying the MAC driver client
pRes	optional pointer to an address that will receive an additional result associated with the operation. Can be 0 if not needed.
ppPktStat	optional pointer to an address that will receive the received packet status. Note that this pointer cannot be used once the packet acknowledgment function was called. Can be 0 if not needed.

Function

TCPIP_MAC_PACKET* TCPIP_MAC_PacketRx (DRV_HANDLE hMac, TCPIP_MAC_RES* pRes,
const TCPIP_MAC_PACKET_RX_STAT** ppPktStat);

f) Data Types and Constants

TCPIP_MAC_ACTION Enumeration

Network interface action for initialization/deinitialization

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_ACTION_INIT,
  TCPIP_MAC_ACTION_REINIT,
  TCPIP_MAC_ACTION_DEINIT,
  TCPIP_MAC_ACTION_IF_UP,
  TCPIP_MAC_ACTION_IF_DOWN
} TCPIP_MAC_ACTION;
```

Members

Members	Description
TCPIP_MAC_ACTION_INIT	stack is initialized
TCPIP_MAC_ACTION_REINIT	stack is reinitialized
TCPIP_MAC_ACTION_DEINIT	stack is deinitialized
TCPIP_MAC_ACTION_IF_UP	interface is brought up
TCPIP_MAC_ACTION_IF_DOWN	interface is brought down

Description

TCP/IP MAC Action

This enumeration defines network interface action for initialization and deinitialization.

Remarks

None.

TCPIP_MAC_ADDR Structure

Definition of a MAC address.

File

tcpip_mac.h

C

```
typedef struct {
  uint8_t v[6];
} TCPIP_MAC_ADDR;
```

Description

MAC Address

This structure defines the physical MAC address of an interface.

Remarks

None.

TCPIP_MAC_DATA_SEGMENT Structure

A data segment that's part of a TX/RX packet.

File

tcpip_mac.h

C

```
typedef struct _tag_MAC_DATA_SEGMENT {
  struct _tag_MAC_DATA_SEGMENT* next;
  uint8_t* segLoad;
  uint16_t segLen;
  uint16_t segSize;
  uint16_t segFlags;
  uint16_t segLoadOffset;
  uint8_t segClientData[4];
  uint8_t segClientLoad[0];
} TCPIP_MAC_DATA_SEGMENT;
```

Members

Members	Description
struct _tag_MAC_DATA_SEGMENT* next;	Multi-segment support, next segment in the chain.
uint8_t* segLoad;	Pointer to segment data payload. It specifies the address of the 1st byte to be transmitted.
uint16_t segLen;	Segment payload size; Number of bytes from this segment that has to be transmitted.
uint16_t segSize;	Segment allocated total usable size. This does not include the segLoadOffset (see below).
uint16_t segFlags;	TCPIP_MAC_SEGMENT_FLAGS segment flags: TCPIP_MAC_SEG_FLAG_STATIC, TCPIP_MAC_SEG_FLAG_RX, TCPIP_MAC_SEG_FLAG_RX, TCPIP_MAC_SEG_FLAG_RX_STICKY
uint16_t segLoadOffset;	Note 1: This offset is used as a performance improvement. It allows for the MAC frame to start on an unaligned address but enforces the alignment of the network layer data and improves the IP checksum calculation. The value of this offset is MAC dependent. A typical value for an Ethernet MAC should be 2 bytes (size of the MAC frame is 14 bytes). Note 2: Normally only the 1st segment of a packet needs this extra room at the beginning of the segment buffer. Note 3: The MAC may make use of this space at the beginning of the segment buffer. This is a space reserved for the MAC purposes. Note 4: It is up to the MAC to check that the value of this offset is enforced. PIC32 MAC specific notes: Note 1. The MAC will reject the packet if the load offset is not at least 2 bytes. 2. The PIC32 MAC uses these 2 bytes to calculate the offset between the segLoad and the TCPIP_MAC_PACKET packet it belongs to. That means that the TCPIP_MAC_PACKET* and the segLoad cannot be more than 64 KB apart!
uint8_t segClientData[4];	Additional client segment data. Ignored by the MAC driver.
uint8_t segClientLoad[0];	Additional client segment payload; Ignored by the MAC driver.

Structures

	Name	Description
	_tag_MAC_DATA_SEGMENT	A data segment that's part of a TX/RX packet.

Description

TCPIP MAC Data Segment

Structure of a segment buffer transferred with the MAC. A MAC TX or RX packet can consist of multiple data segments. On TX the MAC has to be able to transmit packets that span multiple data segments. On RX of a network frame the MAC may have to use multiple segments to construct a packet. (For performance reasons, a contiguous MAC packet, with just one segment, if possible, is preferred).

Remarks

See notes for the segLoadOffset member. On 32-bit machines, the segment payload is allocated so that it is always 32-bit aligned and its size is 32-bits multiple. The segLoadOffset adds to the payload address and insures that the network layer data is 32-bit aligned.

_tag_MAC_DATA_SEGMENT Structure

A data segment that's part of a TX/RX packet.

File

tcpip_mac.h

C

```
struct _tag_MAC_DATA_SEGMENT {
  struct _tag_MAC_DATA_SEGMENT* next;
  uint8_t* segLoad;
  uint16_t segLen;
  uint16_t segSize;
  uint16_t segFlags;
  uint16_t segLoadOffset;
  uint8_t segClientData[4];
  uint8_t segClientLoad[0];
};
```

Members

Members	Description
struct _tag_MAC_DATA_SEGMENT* next;	Multi-segment support, next segment in the chain.
uint8_t* segLoad;	Pointer to segment data payload. It specifies the address of the 1st byte to be transmitted.
uint16_t segLen;	Segment payload size; Number of bytes from this segment that has to be transmitted.
uint16_t segSize;	Segment allocated total usable size. This does not include the segLoadOffset (see below).
uint16_t segFlags;	TCPIP_MAC_SEGMENT_FLAGS segment flags: TCPIP_MAC_SEG_FLAG_STATIC, TCPIP_MAC_SEG_FLAG_RX, TCPIP_MAC_SEG_FLAG_RX, TCPIP_MAC_SEG_FLAG_RX_STICKY
uint16_t segLoadOffset;	Note 1: This offset is used as a performance improvement. It allows for the MAC frame to start on an unaligned address but enforces the alignment of the network layer data and improves the IP checksum calculation. The value of this offset is MAC dependent. A typical value for an Ethernet MAC should be 2 bytes (size of the MAC frame is 14 bytes). Note 2: Normally only the 1st segment of a packet needs this extra room at the beginning of the segment buffer. Note 3: The MAC may make use of this space at the beginning of the segment buffer. This is a space reserved for the MAC purposes. Note 4: It is up to the MAC to check that the value of this offset is enforced. PIC32 MAC specific notes: Note 1. The MAC will reject the packet if the load offset is not at least 2 bytes. 2. The PIC32 MAC uses these 2 bytes to calculate the offset between the segLoad and the TCPIP_MAC_PACKET packet it belongs to. That means that the TCPIP_MAC_PACKET* and the segLoad cannot be more than 64 KB apart!
uint8_t segClientData[4];	Additional client segment data. Ignored by the MAC driver.
uint8_t segClientLoad[0];	Additional client segment payload; Ignored by the MAC driver.

Description

TCPIP MAC Data Segment

Structure of a segment buffer transferred with the MAC. A MAC TX or RX packet can consist of multiple data segments. On TX the MAC has to be able to transmit packets that span multiple data segments. On RX of a network frame the MAC may have to use multiple segments to construct a packet. (For performance reasons, a contiguous MAC packet, with just one segment, if possible, is preferred).

Remarks

See notes for the segLoadOffset member. On 32-bit machines, the segment payload is allocated so that it is always 32-bit aligned and its size is 32-bits multiple. The segLoadOffset adds to the payload address and insures that the network layer data is 32-bit aligned.

TCPIP_MAC_ETHERNET_HEADER Structure

Definition of a MAC frame header.

File

tcpip_mac.h

C

```
typedef struct {
  TCPIP_MAC_ADDR DestMACAddr;
  TCPIP_MAC_ADDR SourceMACAddr;
  uint16_t Type;
} TCPIP_MAC_ETHERNET_HEADER;
```

Description

MAC Ethernet Header

This structure defines the generic Ethernet header that starts all the Ethernet frames.

Remarks

None.

TCPIP_MAC_EVENT Enumeration

Defines the possible MAC event types.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_EV_NONE = 0x0000,
  TCPIP_MAC_EV_RX_PKTPEND = 0x0001,
  TCPIP_MAC_EV_RX_OVFLOW = 0x0002,
  TCPIP_MAC_EV_RX_BUFNA = 0x0004,
  TCPIP_MAC_EV_RX_ACT = 0x0008,
  TCPIP_MAC_EV_RX_DONE = 0x0010,
  TCPIP_MAC_EV_RX_FWMARK = 0x0020,
  TCPIP_MAC_EV_RX_EWMARK = 0x0040,
  TCPIP_MAC_EV_RX_BUSERR = 0x0080,
  TCPIP_MAC_EV_TX_DONE = 0x0100,
  TCPIP_MAC_EV_TX_ABORT = 0x0200,
  TCPIP_MAC_EV_TX_BUSERR = 0x0400,
  TCPIP_MAC_EV_CONN_ESTABLISHED = 0x0800,
  TCPIP_MAC_EV_CONN_LOST = 0x1000,
  TCPIP_MAC_EV_RX_ALL = (TCPIP_MAC_EV_RX_PKTPEND|TCPIP_MAC_EV_RX_OVFLOW|
TCPIP_MAC_EV_RX_BUFNA|TCPIP_MAC_EV_RX_ACT| TCPIP_MAC_EV_RX_DONE|TCPIP_MAC_EV_RX_FWMARK|
TCPIP_MAC_EV_RX_EWMARK|TCPIP_MAC_EV_RX_BUSERR),
  TCPIP_MAC_EV_TX_ALL = (TCPIP_MAC_EV_TX_DONE|TCPIP_MAC_EV_TX_ABORT|TCPIP_MAC_EV_TX_BUSERR),
  TCPIP_MAC_EV_RXTX_ERRORS = (TCPIP_MAC_EV_RX_OVFLOW|TCPIP_MAC_EV_RX_BUFNA|
TCPIP_MAC_EV_RX_BUSERR|TCPIP_MAC_EV_TX_ABORT| TCPIP_MAC_EV_TX_BUSERR),
  TCPIP_MAC_EV_CONN_ALL = (TCPIP_MAC_EV_CONN_ESTABLISHED|TCPIP_MAC_EV_CONN_LOST)
} TCPIP_MAC_EVENT;
```

Members

Members	Description
TCPIP_MAC_EV_NONE = 0x0000	no event
TCPIP_MAC_EV_RX_PKTPEND = 0x0001	RX triggered events: A receive packet is pending
TCPIP_MAC_EV_RX_OVFLOW = 0x0002	RX triggered events: RX FIFO overflow (system level latency, no descriptors, etc.)
TCPIP_MAC_EV_RX_BUFNA = 0x0004	RX triggered events: no RX descriptor available to receive a new packet

TCPIP_MAC_EV_RX_ACT = 0x0008	RX triggered events: There's RX data available
TCPIP_MAC_EV_RX_DONE = 0x0010	RX triggered events: A packet was successfully received
TCPIP_MAC_EV_RX_FWMARK = 0x0020	RX triggered events: the number of received packets is greater than or equal to than the RX Full Watermark
TCPIP_MAC_EV_RX_EWMARK = 0x0040	RX triggered events: the number of received packets is less than or equal to than the RX Empty Watermark
TCPIP_MAC_EV_RX_BUSERR = 0x0080	RX triggered events: a bus error encountered during an RX transfer
TCPIP_MAC_EV_TX_DONE = 0x0100	TX triggered events: A packet was transmitted and its status is available
TCPIP_MAC_EV_TX_ABORT = 0x0200	TX triggered events: a TX packet was aborted by the MAC (jumbo/system underrun/excessive defer/late collision/excessive collisions)
TCPIP_MAC_EV_TX_BUSERR = 0x0400	TX triggered events: a bus error encountered during a TX transfer
TCPIP_MAC_EV_CONN_ESTABLISHED = 0x0800	Connection triggered events: Connection established
TCPIP_MAC_EV_CONN_LOST = 0x1000	Connection triggered events: Connection lost
TCPIP_MAC_EV_RX_ALL = (TCPIP_MAC_EV_RX_PKTPEND|TCPIP_MAC_EV_RX_OVFLOW| TCPIP_MAC_EV_RX_BUFNA|TCPIP_MAC_EV_RX_ACT| TCPIP_MAC_EV_RX_DONE|TCPIP_MAC_EV_RX_FWMARK| TCPIP_MAC_EV_RX_EWMARK|TCPIP_MAC_EV_RX_BUSERR)	Useful Masks: all RX related events
TCPIP_MAC_EV_TX_ALL = (TCPIP_MAC_EV_TX_DONE|TCPIP_MAC_EV_TX_ABORT|TCPIP_MAC_EV_TX_BUSERR)	Useful Masks: all TX related events
TCPIP_MAC_EV_RXTX_ERRORS = (TCPIP_MAC_EV_RX_OVFLOW|TCPIP_MAC_EV_RX_BUFNA| TCPIP_MAC_EV_RX_BUSERR|TCPIP_MAC_EV_TX_ABORT| TCPIP_MAC_EV_TX_BUSERR)	abnormal traffic/system events: Action should be taken accordingly by the stack (or the stack user)
TCPIP_MAC_EV_CONN_ALL = (TCPIP_MAC_EV_CONN_ESTABLISHED|TCPIP_MAC_EV_CONN_LOST)	Mask of all Connection related events

Description

TCP/IP MAC Event

TCP/IP MAC Events Codes.

This enumeration defines all the possible events that can be reported by the MAC to the stack.

Depending on the type of the hardware Ethernet/Wi-Fi interface, etc., not all events are possible.

Remarks

None.

TCPIP_MAC_EventF Type

MAC event notification handler.

File

tcpip_mac.h

C

```
typedef void (* TCPIP_MAC_EventF)(TCPIP_MAC_EVENT event, const void* eventParam);
```

Returns

None

Description

Event notification Function: typedef void (*TCPIP_MAC_EventF)(TCPIP_MAC_EVENT event, const void* eventParam);

This function describes the MAC event notification handler. This is a handler specified by the user of the MAC (the TCP/IP stack). The stack can use the handler to be notified of MAC events. Whenever a notification occurs the passed events have to be eventually processed:

- Stack should process the TCPIP_EV_RX_PKTPEND/TCPIP_EV_RX_DONE, TCPIP_EV_TX_DONE events
- Process the specific (error) condition

- Acknowledge the events by calling TCPIP_MAC_EventAcknowledge() so that they can be re-enabled.

Remarks

The notification handler will be called from the ISR which detects the corresponding event. The event notification handler has to be kept as short as possible and non-blocking. Mainly useful for RTOS integration where this handler will wake-up a thread that waits for a MAC event to occur. The event notification system also enables the user of the TCPIP stack to call into the stack for processing only when there are relevant events rather than being forced to periodically call from within a loop at unknown moments. Without a notification handler the stack user can still call TCPIP_MAC_EventPendingGet to see if processing by the stack needed. This is a default way of adding MAC interrupt processing to the TCP/IP stack.

Parameters

Parameters	Description
event	event that's reported (multiple events can be OR-ed)
eventParam	user parameter that's used in the notification handler

TCPIP_MAC_HANDLE Type

Handle to a MAC driver.

File

tcpip_mac.h

C

```
typedef DRV_HANDLE TCPIP_MAC_HANDLE;
```

Description

MAC Handle

This data type defines a MAC client handle.

Remarks

Another name for the DRV_HANDLE.

TCPIP_MAC_HEAP_CallocF Type

MAC allocation function prototype.

File

tcpip_mac.h

C

```
typedef void* (* TCPIP_MAC_HEAP_CallocF)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nElems, size_t elemSize);
```

Returns

- valid pointer - if the allocation request succeeded
- 0 - if the allocation request failed

Description

Memory Allocation Function: typedef void* (*TCPIP_MAC_HEAP_CallocF)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nElems, size_t elemSize);

This calloc style function is used by the MAC to allocate memory.

Remarks

The debug version adds the module identifier and source file line number.

Parameters

Parameters	Description
heapH	heap handle to be used in call
nElems	number of identical elements requested
elemSize	size of one element, in bytes

TCPIP_MAC_HEAP_FreeF Type

MAC allocation function prototype.

File

tcpip_mac.h

C

```
typedef size_t (* TCPIP_MAC_HEAP_FreeF)(TCPIP_MAC_HEAP_HANDLE heapH, const void* pBuff);
```

Returns

- Non-zero number - if the free request succeeded
- 0 - if the free request failed

Description

Memory Allocation Function: typedef size_t (*TCPIP_MAC_HEAP_FreeF)(TCPIP_MAC_HEAP_HANDLE heapH, const void* pBuff);

This free style function is used by the MAC to free allocated memory.

Remarks

The debug version adds the module identifier and source file line number.

TCPIP_MAC_HEAP_HANDLE Type

A handle used for memory allocation functions.

File

tcpip_mac.h

C

```
typedef const void* TCPIP_MAC_HEAP_HANDLE;
```

Description

Handle to a heap

This is the handle that's required for accessing memory allocation functions.

Remarks

None

TCPIP_MAC_HEAP_MallocF Type

MAC allocation function prototype.

File

tcpip_mac.h

C

```
typedef void* (* TCPIP_MAC_HEAP_MallocF)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nBytes);
```

Returns

- valid pointer - if the allocation request succeeded
- 0 - if the allocation request failed

Description

Memory Allocation Function: typedef void* (*TCPIP_MAC_HEAP_MallocF)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nBytes);

This malloc style function is used by the MAC to allocate memory.

Remarks

The debug version adds the module identifier and source file line number.

TCPIP_MAC_MODULE_CTRL Structure

Data structure that's passed to the MAC at the initialization time.

File

tcpip_mac.h

C

```
typedef struct {
  int nIfs;
  TCPIP_MAC_HEAP_MallocF mallocF;
  TCPIP_MAC_HEAP_CallocF callocF;
  TCPIP_MAC_HEAP_FreeF freeF;
  TCPIP_MAC_HEAP_HANDLE memH;
  TCPIP_MAC_PKT_AllocF pktAllocF;
  TCPIP_MAC_PKT_FreeF pktFreeF;
  TCPIP_MAC_PKT_AckF pktAckF;
  TCPIP_MAC_SynchReqF synchF;
  TCPIP_MAC_EventF eventF;
  const void* eventParam;
  unsigned int moduleId;
  int netIx;
  TCPIP_MAC_ACTION macAction;
  TCPIP_MAC_POWER_MODE powerMode;
  TCPIP_MAC_ADDR ifPhyAddress;
} TCPIP_MAC_MODULE_CTRL;
```

Members

Members	Description
int nIfs;	number of the interfaces supported in this session
TCPIP_MAC_HEAP_MallocF mallocF;	malloc type allocation function
TCPIP_MAC_HEAP_CallocF callocF;	calloc type allocation function
TCPIP_MAC_HEAP_FreeF freeF;	free type allocation free function
TCPIP_MAC_HEAP_HANDLE memH;	handle to be used in the stack allocation service calls
TCPIP_MAC_PKT_AllocF pktAllocF;	packet allocation function
TCPIP_MAC_PKT_FreeF pktFreeF;	packet free function
TCPIP_MAC_PKT_AckF pktAckF;	packet allocation function
TCPIP_MAC_SynchReqF synchF;	Synchronization object request function
TCPIP_MAC_EventF eventF;	Event notification function. used by the MAC for event reporting.
const void* eventParam;	Parameter to be used when the event function is called.
unsigned int moduleId;	Module identifier. Allows multiple channels/ports, etc. MAC support.
int netIx;	index of the current interface
TCPIP_MAC_ACTION macAction;	current action for the MAC/stack
TCPIP_MAC_POWER_MODE powerMode;	The power mode for this interface to go to. Valid only if stackAction == init/reinit. Ignored for deinitialize operation.
TCPIP_MAC_ADDR ifPhyAddress;	Physical address of the interface. MAC sets this field as part of the initialization function. The stack will use this data as the interface address.

Description

MAC Initialization Data

This is the data structure that the MAC user (TCP/IP stack) passes on to the MAC driver at the MAC initialization time. It contains all the data needed for the MAC to initialize itself and to start processing packets.

Remarks

Most of the data that's passed in this structure is permanent data. It is maintained by the stack for one full session i.e., across Initialize() -> DeInitialize() calls.

Some fields are module specific though (like the memory allocation handle, allocation functions, etc.) that could be different from one module to the other.

TCPIP_MAC_PACKET Type

Forward reference to a MAC packet.

File

tcpip_mac.h

C

```
typedef struct _tag_TCPIP_MAC_PACKET TCPIP_MAC_PACKET;
```

Structures

	Name	Description
	_tag_TCPIP_MAC_PACKET	A TX/RX packet descriptor.

Description

MAC Packet forward reference

Forward reference needed in the MAC packet acknowledge function.

Remarks

None.

_tag_TCPIP_MAC_PACKET Structure

A TX/RX packet descriptor.

File

tcpip_mac.h

C

```
struct _tag_TCPIP_MAC_PACKET {
  struct _tag_TCPIP_MAC_PACKET* next;
  struct _tag_TCPIP_MAC_PACKET* pkt_next;
  TCPIP_MAC_PACKET_ACK_FUNC ackFunc;
  const void* ackParam;
  TCPIP_MAC_DATA_SEGMENT* pDSeg;
  uint8_t* pMacLayer;
  uint8_t* pNetLayer;
  uint8_t* pTransportLayer;
  uint16_t totTransportLen;
  uint16_t pktFlags;
  uint32_t tStamp;
  const void* pktIf;
  int16_t ackRes;
  uint16_t pktClientData;
  uint32_t pktClientLoad[0];
};
```

Members

Members	Description
struct _tag_TCPIP_MAC_PACKET* next;	Multi-packet/queuing support. This field is used for chaining/queuing packets.
struct _tag_TCPIP_MAC_PACKET* pkt_next;	Multi-packet/queuing support. This field is used for chaining packet fragments, etc.
TCPIP_MAC_PACKET_ACK_FUNC ackFunc;	Packet acknowledgment function. On TX: A stack module owns the packet. Once the MAC is done transmitting the packet (success or failure) it has to set an appropriate acknowledge result in the ackRes field (a TCPIP_MAC_PKT_ACK_RES) adjust some packet flags (see TCPIP_MAC_PKT_FLAG_QUEUED) and call the packet acknowledge function (ackFunc). This call informs the packet owner that the MAC is done with using that packet. It is up to the implementation what action the ackFunc takes: reuse, free, discard the packet or if some of the above steps are included in the ackFunc itself. On RX: The packet is under the MAC control and it's passed to the stack when it contains valid data. Once the recipient of the packet (IP, ICMP, UDP, TCP, etc.) is done with the packet processing it has to set an appropriate acknowledge result in the ackRes field (a TCPIP_MAC_PKT_ACK_RES) adjust some packet flags (see TCPIP_MAC_PKT_FLAG_QUEUED) and call the packet acknowledge function (ackFunc). This call informs the packet owner (the MAC) that the processing of the packet is completed. It is up to the implementation what action the ackFunc takes: reuse, free, discard the packet or if some of the above steps are included in the ackFunc itself.
const void* ackParam;	Associated acknowledgment parameter to be used when ackFunc is called. For TX packets: this is a client supplied parameter and is not used by the MAC driver. For RX: If the MAC owns the RX packet then the MAC driver can use this field for further dispatching in the MAC driver owned acknowledgment function.

TCPIP_MAC_DATA_SEGMENT* pDSeg;	Data (root) segment associated to this packet. It is up to the design if the root data segment associated with a packet is contiguous in memory with the packet itself. It can be 0 if the packet has no associated data.
uint8_t* pMacLayer;	Pointer to the MAC frame. On TX: the sending higher layer protocol updates this field. On RX: the MAC driver updates this field before handing over the packet. (MCHP TCP/IP stack note: The packet allocation function update this field automatically).
uint8_t* pNetLayer;	Pointer to the network layer data. On TX: the sending higher layer protocol updates this field. The MAC driver shouldn't need this field. On RX: the MAC driver updates this field before handing over the packet. (MCHP TCP/IP stack note: The packet allocation function updates this field automatically. But not for IPv6!).
uint8_t* pTransportLayer;	Pointer to the transport layer. On TX: the sending higher layer protocol updates this field. The MAC driver shouldn't need this field. On RX: the MAC driver updates this field before handing over the packet. (MCHP TCP/IP stack note: The packet allocation function updates this field automatically. But not for IPv6!).
uint16_t totTransportLen;	Total length of the transport layer. The MAC driver shouldn't need this field.
uint16_t pktFlags;	TCPIP_MAC_PACKET_FLAGS associated packet flags. On TX: the MAC driver has to set: TCPIP_MAC_PKT_FLAG_QUEUED - if the driver needs to queue the packet On RX: the MAC driver updates this field before handing over the packet. Flags that the MAC driver has to set/clear: TCPIP_MAC_PKT_FLAG_RX and TCPIP_MAC_PKT_FLAG_UNICAST, TCPIP_MAC_PKT_FLAG_BCAST and TCPIP_MAC_PKT_FLAG_MCAST TCPIP_MAC_PKT_FLAG_QUEUED TCPIP_MAC_PKT_FLAG_SPLIT
uint32_t tStamp;	Time stamp value. Statistics, info. On TX: the sending higher layer protocol updates this field. The MAC driver shouldn't need this field On RX: the MAC driver updates this field before handing over the packet.
const void* pktIf;	The packet interface. On TX: the sending higher layer protocol updates this field. The MAC driver doesn't use this field. On RX: the receiving higher level protocol updates this value. The MAC driver doesn't use this field.
int16_t ackRes;	A TCPIP_MAC_PKT_ACK_RES code associated with the packet. On TX: The MAC driver sets this field when calling the packet ackFunc. On RX: The higher level protocol which is the recipient of the packet sets this field when calling the packet ackFunc.
uint16_t pktClientData;	Client/padding data; ignored by the MAC driver. It can be used by the packet client.
uint32_t pktClientLoad[0];	Additional client packet payload, variable packet data. Ignored by the MAC driver.

Description

MAC Packet

Structure of a packet transferred with the MAC. This is the structure used for both TX and RX. See the description of each individual field.

Remarks

Specific TCP/IP stack implementations might offer packet support functions to assist in driver development. (For the MCHP TCP/IP stack see tcpip_packet.h)

Since the packets may be dynamically allocated, an acknowledge function can result in data deallocation (blocking). Therefore, the acknowledgment function should NOT be called from within an interrupt context.

TCPIP_MAC_PACKET_ACK_FUNC Type

Prototype of a MAC packet acknowledge function.

File

tcpip_mac.h

C

```
typedef bool (* TCPIP_MAC_PACKET_ACK_FUNC)(TCPIP_MAC_PACKET* pkt, const void* param);
```

Returns

- true - if the packet needs the queuing flags removed (it is not deleted and still in use)
- false - if the packet does not need the queuing flags removed (either no longer exists or the flags updated)

Description

MAC Acknowledge Function

This is the prototype of the function that the MAC calls once the TX/RX packet processing is done.

Remarks

None.

TCPIP_MAC_PACKET_FLAGS Enumeration

Flags belonging to MAC packet.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_PKT_FLAG_STATIC = 0x0001,
  TCPIP_MAC_PKT_FLAG_TX = 0x0002,
  TCPIP_MAC_PKT_FLAG_SPLIT = 0x0004,
  TCPIP_MAC_PKT_FLAG_QUEUED = 0x0008,
  TCPIP_MAC_PKT_FLAG_UNICAST = 0x0010,
  TCPIP_MAC_PKT_FLAG_BCAST = 0x0020,
  TCPIP_MAC_PKT_FLAG_MCAST = 0x0040,
  TCPIP_MAC_PKT_FLAG_CAST_MASK = 0x0070,
  TCPIP_MAC_PKT_FLAG_CAST_DISABLED = 0x0000,
  TCPIP_MAC_PKT_FLAG_USER = 0x0100
} TCPIP_MAC_PACKET_FLAGS;
```

Members

Members	Description
TCPIP_MAC_PKT_FLAG_STATIC = 0x0001	Packet can not be dynamically deallocated. Set when the packet is allocated.
TCPIP_MAC_PKT_FLAG_TX = 0x0002	If set, it is a TX packet/segment. Otherwise, it is a RX packet.
TCPIP_MAC_PKT_FLAG_SPLIT = 0x0004	Packet data spans multiple segments - ZC functionality. If not set then the packet has only one data segment. It is set by the MAC driver when a RX packet spans multiple data segments.
TCPIP_MAC_PKT_FLAG_QUEUED = 0x0008	Packet data is queued somewhere, cannot be freed. The flag is set by a module processing the packet to show that the packet is in use and queued for further processing (normally the MAC driver does that). Cleared by the packet destination when the packet processing was completed.
TCPIP_MAC_PKT_FLAG_UNICAST = 0x0010	RX flag, MAC updated. Specifies an unicast packet.
TCPIP_MAC_PKT_FLAG_BCAST = 0x0020	RX flag, MAC updated. Specifies a broadcast packet.
TCPIP_MAC_PKT_FLAG_MCAST = 0x0040	RX flag, MAC updated. Specifies an multicast packet.
TCPIP_MAC_PKT_FLAG_CAST_MASK = 0x0070	Packet cast mask bits.
TCPIP_MAC_PKT_FLAG_CAST_DISABLED = 0x0000	Packet cast mask. Specifies a packet where the MCAST/BCAST fields are not updated by the MAC RX process.
TCPIP_MAC_PKT_FLAG_USER = 0x0100	Available user flags.

Description

MAC Packet Flags

This enumeration contains the definitions of MAC packet flags: packet allocation flags and general purpose flags.

Remarks

16 bits only packet flags are supported.

TCPIP_MAC_PACKET_RX_STAT Structure

Status of a received packet.

File

tcpip_mac.h

C

```
typedef struct {
  unsigned chksumOk : 1;
  unsigned pktChecksum : 16;
  unsigned runtPkt : 1;
  unsigned notMeUcast : 1;
  unsigned htMatch : 1;
  unsigned magicMatch : 1;
  unsigned pmMatch : 1;
```

```
  unsigned uMatch : 1;
  unsigned bMatch : 1;
  unsigned mMatch : 1;
  unsigned rxBytes : 16;
  unsigned crcError : 1;
  unsigned lenError : 1;
  unsigned lenRange : 1;
  unsigned rxOk : 1;
  unsigned mcast : 1;
  unsigned bcast : 1;
  unsigned rxCtrl : 1;
  unsigned rxVLAN : 1;
} TCPIP_MAC_PACKET_RX_STAT;
```

Members

Members	Description
unsigned chksumOk : 1;	correct checksum filled in
unsigned pktChecksum : 16;	Packet payload checksum
unsigned runtPkt : 1;	Runt packet received
unsigned notMeUcast : 1;	Unicast, not me packet,
unsigned htMatch : 1;	Hash table match
unsigned magicMatch : 1;	Magic packet match
unsigned pmMatch : 1;	Pattern match match
unsigned uMatch : 1;	Unicast match
unsigned bMatch : 1;	Broadcast match
unsigned mMatch : 1;	Multicast match
unsigned rxBytes : 16;	Received bytes
unsigned crcError : 1;	CRC error in packet
unsigned lenError : 1;	Receive length check error
unsigned lenRange : 1;	Receive length out of range
unsigned rxOk : 1;	Receive OK
unsigned mcast : 1;	Multicast packet
unsigned bcast : 1;	Broadcast packet
unsigned rxCtrl : 1;	Control frame received
unsigned rxVLAN : 1;	Received VLAN tagged frame

Description

Received Packet Status

This structure contains the status of a received packet.

Remarks

Not all the MACs have hardware support for the received packet status.

TCPIP_MAC_PKT_ACK_RES Enumeration

List of MAC return codes for a packet acknowledge function.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_PKT_ACK_NONE = 0,
  TCPIP_MAC_PKT_ACK_TX_OK = 1,
  TCPIP_MAC_PKT_ACK_RX_OK = 2,
  TCPIP_MAC_PKT_ACK_LINK_DOWN = -1,
  TCPIP_MAC_PKT_ACK_NET_DOWN = -2,
  TCPIP_MAC_PKT_ACK_BUFFER_ERR = -3,
  TCPIP_MAC_PKT_ACK_ARP_TMO = -4,
  TCPIP_MAC_PKT_ACK_ARP_NET_ERR = -5,
  TCPIP_MAC_PKT_ACK_MAC_REJECT_ERR = -6,
  TCPIP_MAC_PKT_ACK_CHKSUM_ERR = -10,
  TCPIP_MAC_PKT_ACK_SOURCE_ERR = -11,
  TCPIP_MAC_PKT_ACK_TYPE_ERR = -12,
```

```
  TCPIP_MAC_PKT_ACK_STRUCT_ERR = -13,
  TCPIP_MAC_PKT_ACK_PROTO_DEST_ERR = -14,
  TCPIP_MAC_PKT_ACK_FRAGMENT_ERR = -15,
  TCPIP_MAC_PKT_ACK_PROTO_DEST_CLOSE = -16,
  TCPIP_MAC_PKT_ACK_ALLOC_ERR = -17
} TCPIP_MAC_PKT_ACK_RES;
```

Members

Members	Description
TCPIP_MAC_PKT_ACK_NONE = 0	packet result unknown, unspecified
TCPIP_MAC_PKT_ACK_TX_OK = 1	TX success code - packet was transmitted successfully
TCPIP_MAC_PKT_ACK_RX_OK = 2	RX success code - packet was received/processed successfully
TCPIP_MAC_PKT_ACK_LINK_DOWN = -1	TX:packet was dropped because the link was down
TCPIP_MAC_PKT_ACK_NET_DOWN = -2	TX:packet was dropped because the network is down
TCPIP_MAC_PKT_ACK_BUFFER_ERR = -3	TX:packet was dropped because the buffer type is not supported
TCPIP_MAC_PKT_ACK_ARP_TMO = -4	TX:packet was dropped because of an ARP timeout
TCPIP_MAC_PKT_ACK_ARP_NET_ERR = -5	TX:packet associated interface is down or non existent
TCPIP_MAC_PKT_ACK_MAC_REJECT_ERR = -6	TX:packet rejected by MAC
TCPIP_MAC_PKT_ACK_CHKSUM_ERR = -10	RX: packet was dropped because the checksum was incorrect
TCPIP_MAC_PKT_ACK_SOURCE_ERR = -11	RX: packet was dropped because of wrong interface source address
TCPIP_MAC_PKT_ACK_TYPE_ERR = -12	RX: packet was dropped because the type was unknown
TCPIP_MAC_PKT_ACK_STRUCT_ERR = -13	RX: internal packet structure error
TCPIP_MAC_PKT_ACK_PROTO_DEST_ERR = -14	RX: the packet protocol couldn't find a destination for it
TCPIP_MAC_PKT_ACK_FRAGMENT_ERR = -15	RX: the packet too fragmented
TCPIP_MAC_PKT_ACK_PROTO_DEST_CLOSE = -16	RX: the packet destination is closing
TCPIP_MAC_PKT_ACK_ALLOC_ERR = -17	RX: memory allocation error

Description

MAC Packet Acknowledge Result

This enumeration contains the list of MAC codes used for a packet acknowledgment.

Remarks

16 bits only acknowledge results are supported. Positive codes indicate success. Negative codes indicate a failure of some sort.

TCPIP_MAC_PKT_AckF Type

MAC packet acknowledge function prototype.

File

tcpip_mac.h

C

```
typedef void (* TCPIP_MAC_PKT_AckF)(TCPIP_MAC_PACKET* pPkt, TCPIP_MAC_PKT_ACK_RES ackRes);
```

Returns

None.

Description

Packet Acknowledgment Function: typedef void (*TCPIP_MAC_PKT_AckF)(TCPIP_MAC_PACKET* pPkt, TCPIP_MAC_PKT_ACK_RES ackRes);

This function is used by the MAC to acknowledge a TCPIP_MAC_PACKET packet when the packet processing is completed.

Remarks

A TCPIP_MAC_PACKET packet always has an acknowledgment function. This function should clear the TCPIP_MAC_PKT_FLAG_QUEUED flag. The packet's ackRes is updated only if the parameter ackRes != TCPIP_MAC_PKT_ACK_NONE. The debug version adds the module identifier.

Parameters

Parameters	Description
pPkt	pointer to a valid TCPIP_MAC_PACKET packet.

ackRes	the result of the packet processing

TCPIP_MAC_PKT_AllocF Type

MAC packet allocation function prototype.

File

tcpip_mac.h

C

```
typedef TCPIP_MAC_PACKET* (* TCPIP_MAC_PKT_AllocF)(uint16_t pktLen, uint16_t segLoadLen,
TCPIP_MAC_PACKET_FLAGS flags);
```

Returns

- Valid packet pointer - if the allocation request succeeded
- 0 - if the allocation request failed

Description

Packet Allocation Function: typedef TCPIP_MAC_PACKET* (*TCPIP_MAC_PKT_AllocF)(uint16_t pktLen, uint16_t segLoadLen, TCPIP_MAC_PACKET_FLAGS flags);

This function is used by the MAC to allocate a TCPIP_MAC_PACKET packet.

Remarks

The returned allocated packet should always have the TCPIP_MAC_ETHERNET_HEADER added to the packet.

The debug version adds the module identifier.

Parameters

Parameters	Description
pktLen	the size of the packet (it will be 32 bits rounded up)
segLoadLen	the payload size for the segment associated to this packet. Payload is always 32 bit aligned. If 0 no segment is created/attached to the packet.
flags	packet flags

TCPIP_MAC_PKT_FreeF Type

MAC packet free function prototype.

File

tcpip_mac.h

C

```
typedef void (* TCPIP_MAC_PKT_FreeF)(TCPIP_MAC_PACKET* pPkt);
```

Returns

None.

Description

Packet Allocation Function: typedef void (*TCPIP_MAC_PKT_FreeF)(TCPIP_MAC_PACKET* pPkt);

This function is used by the MAC to free a previously allocated TCPIP_MAC_PACKET packet.

Remarks

The function will free a previously allocated packet. However packets or segments marked with TCPIP_MAC_PKT_FLAG_STATIC/TCPIP_MAC_SEG_FLAG_STATIC are not freed.

Also note that this function does not free explicitly the external segment payload. A payload that was created contiguously when the segment was created will be automatically freed by this function.

The debug version adds the module identifier.

Parameters

Parameters	Description
pPkt	pointer to a previously allocated packet.

TCPIP_MAC_POWER_MODE Enumeration

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_POWER_NONE,
  TCPIP_MAC_POWER_FULL,
  TCPIP_MAC_POWER_LOW,
  TCPIP_MAC_POWER_DOWN
} TCPIP_MAC_POWER_MODE;
```

Members

Members	Description
TCPIP_MAC_POWER_NONE	unknown power mode
TCPIP_MAC_POWER_FULL	up and running; valid for init/reinit
TCPIP_MAC_POWER_LOW	low power mode; valid for init/reinit
TCPIP_MAC_POWER_DOWN	interface is down

Description

This is type TCPIP_MAC_POWER_MODE.

TCPIP_MAC_PROCESS_FLAGS Enumeration

List of the MAC processing flags.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_PROCESS_FLAG_NONE = 0x0000,
  TCPIP_MAC_PROCESS_FLAG_RX = 0x0001,
  TCPIP_MAC_PROCESS_FLAG_TX = 0x0002,
  TCPIP_MAC_PROCESS_FLAG_ANY = 0x0100
} TCPIP_MAC_PROCESS_FLAGS;
```

Members

Members	Description
TCPIP_MAC_PROCESS_FLAG_NONE = 0x0000	the stack never has to call the TCPIP_MAC_Process function
TCPIP_MAC_PROCESS_FLAG_RX = 0x0001	the stack has to call the TCPIP_MAC_Process after an RX signal
TCPIP_MAC_PROCESS_FLAG_TX = 0x0002	the stack has to call the TCPIP_MAC_Process after an TX signal
TCPIP_MAC_PROCESS_FLAG_ANY = 0x0100	the stack has to call the TCPIP_MAC_Process after any type of signal

Description

MAC Process Flags

List of specific MAC processing flags that indicate to the user of the MAC (TCP/IP stack) the processing that is expected by the MAC driver.

Remarks

Multiple flags can be "ORed".

TCPIP_MAC_RES Enumeration

List of return codes from MAC functions.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_RES_OK = 0,
  TCPIP_MAC_RES_PENDING = 1,
  TCPIP_MAC_RES_TYPE_ERR = -1,
  TCPIP_MAC_RES_IS_BUSY = -2,
  TCPIP_MAC_RES_INIT_FAIL = -3,
  TCPIP_MAC_RES_PHY_INIT_FAIL = -4,
  TCPIP_MAC_RES_EVENT_INIT_FAIL = -5,
  TCPIP_MAC_RES_OP_ERR = -6,
  TCPIP_MAC_RES_ALLOC_ERR = -7,
  TCPIP_MAC_RES_INSTANCE_ERR = -8,
  TCPIP_MAC_RES_FRAGMENT_ERR = -9,
  TCPIP_MAC_RES_PACKET_ERR = -10,
  TCPIP_MAC_RES_QUEUE_TX_FULL = -11,
  TCPIP_MAC_RES_SYNCH_LOCK_FAIL = -12
} TCPIP_MAC_RES;
```

Members

Members	Description
TCPIP_MAC_RES_OK = 0	operation successful
TCPIP_MAC_RES_PENDING = 1	operation is pending upon some hardware resource. call again to completion
TCPIP_MAC_RES_TYPE_ERR = -1	unsupported type
TCPIP_MAC_RES_IS_BUSY = -2	device is in use
TCPIP_MAC_RES_INIT_FAIL = -3	generic initialization failure
TCPIP_MAC_RES_PHY_INIT_FAIL = -4	PHY initialization failure
TCPIP_MAC_RES_EVENT_INIT_FAIL = -5	Event system initialization failure
TCPIP_MAC_RES_OP_ERR = -6	unsupported operation
TCPIP_MAC_RES_ALLOC_ERR = -7	memory allocation error
TCPIP_MAC_RES_INSTANCE_ERR = -8	already instantiated, initialized error
TCPIP_MAC_RES_FRAGMENT_ERR = -9	too fragmented, RX buffer too small
TCPIP_MAC_RES_PACKET_ERR = -10	unsupported/corrupted packet error
TCPIP_MAC_RES_QUEUE_TX_FULL = -11	TX queue exceeded the limit
TCPIP_MAC_RES_SYNCH_LOCK_FAIL = -12	Synchronization object lock failed Could not get a lock

Description

MAC Result Enumeration

This is the list of codes that the MAC uses to specify the outcome of a MAC function.

Remarks

Benign operation results - always have positive values. Error codes - always have negative values.

TCPIP_MAC_SEGMENT_FLAGS Enumeration

Flags belonging to MAC data segment.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_SEG_FLAG_STATIC = 0x0001,
  TCPIP_MAC_SEG_FLAG_TX = 0x0002,
  TCPIP_MAC_SEG_FLAG_RX_STICKY = 0x0004,
  TCPIP_MAC_SEG_FLAG_USER_PAYLOAD = 0x0008,
  TCPIP_MAC_SEG_FLAG_USER = 0x0100
} TCPIP_MAC_SEGMENT_FLAGS;
```

Members

Members	Description
TCPIP_MAC_SEG_FLAG_STATIC = 0x0001	Segment can not be dynamically deallocated. Set when the segment is allocated
TCPIP_MAC_SEG_FLAG_TX = 0x0002	If set, it's a TX segment; otherwise, is a RX packet.

TCPIP_MAC_SEG_FLAG_RX_STICKY = 0x0004	a MAC RX dedicated/sticky segment; otherwise, a non-dedicated/float segment
TCPIP_MAC_SEG_FLAG_USER_PAYLOAD = 0x0008	Segment carrying user payload Higher level protocols (TCP, UDP, etc.) may use it
TCPIP_MAC_SEG_FLAG_USER = 0x0100	User available segment flags.

Description

MAC Segment Flags

This enumeration contains the definitions of MAC segment flags: segment allocation flags and general purpose flags.

Remarks

16 bits only segment flags are supported.

TCPIP_MODULE_MAC_MRF24W_CONFIG Structure

File

tcpip_mac.h

C

```
typedef struct {
} TCPIP_MODULE_MAC_MRF24W_CONFIG;
```

Description

This is type TCPIP_MODULE_MAC_MRF24W_CONFIG.

TCPIP_MODULE_MAC_PIC32INT_CONFIG Structure

Data that's passed to the MAC at initialization time.

File

tcpip_mac.h

C

```
typedef struct {
  uint16_t nTxDescriptors;
  uint16_t rxBuffSize;
  uint16_t nRxDescriptors;
  uint16_t nRxDedicatedBuffers;
  uint16_t nRxInitBuffers;
  uint16_t rxLowThreshold;
  uint16_t rxLowFill;
  TCPIP_ETH_OPEN_FLAGS ethFlags;
  DRV_ETHPHY_CONFIG_FLAGS phyFlags;
  uint16_t linkInitDelay;
  uint16_t phyAddress;
  ETH_MODULE_ID ethModuleId;
  const DRV_ETHPHY_OBJECT* pPhyObject;
  const DRV_ETHPHY_OBJECT_BASE* pPhyBase;
} TCPIP_MODULE_MAC_PIC32INT_CONFIG;
```

Members

Members	Description
uint16_t nTxDescriptors;	number of TX descriptors
uint16_t rxBuffSize;	size of the corresponding RX buffer
uint16_t nRxDescriptors;	Number of RX descriptors Has to be high enough to accommodate both dedicated and non-dedicated RX buffers
uint16_t nRxDedicatedBuffers;	Number of MAC dedicated RX buffers These buffers/packets are owned by the MAC and are not returned to the packet pool They are allocated at MAC initialization time using pktAllocF and freed at MAC deinitialize time using pktFreeF Could be 0, if only not dedicated buffers are needed. For best performance usually it's best to have some dedicated buffers so as to minimize the run time allocations
uint16_t nRxInitBuffers;	Number of MAC non dedicated RX buffers allocated at the MAC initialization pktAllocF Note that these buffers are allocated in addition of the nRxDedicatedBuffers Freed at run time using pktFreeF

uint16_t rxLowThreshold;	Minimum threshold for the buffer replenish process Whenever the number of RX scheduled buffers is <= than this threshold the MAC driver will allocate new non-dedicated buffers that will be freed at run time using pktFreeF Setting this value to 0 disables the buffer replenishing process
uint16_t rxLowFill;	Number of RX buffers to allocate when below threshold condition is detected If 0, the MAC driver will allocate (scheduled buffers - rxThres) If !0, the MAC driver will allocate exactly rxLowFill buffers
TCPIP_ETH_OPEN_FLAGS ethFlags;	flags to use for the ETH connection
DRV_ETHPHY_CONFIG_FLAGS phyFlags;	PHY configuration
uint16_t linkInitDelay;	PHYs initialization delay (milliseconds) for insuring that the PHY is ready to transmit data.
uint16_t phyAddress;	PHY address, as configured on the board. All PHYs respond to address 0
ETH_MODULE_ID ethModuleId;	Ethernet module ID for this driver instance
const DRV_ETHPHY_OBJECT* pPhyObject;	Non-volatile pointer to the PHY vendor object associated with this MAC
const DRV_ETHPHY_OBJECT_BASE* pPhyBase;	Non-volatile pointer to the PHY basic object associated with this MAC

Description

MAC Initialization Data

This structure defines the MAC initialization data for the PIC32 MAC/Ethernet controller.

Remarks

- Each supported MAC has its own specific init/configuration data

TCPIP_MAC_RX_STATISTICS Structure

MAC receive statistics data gathered at run time.

File

tcpip_mac.h

C

```
typedef struct {
  int nRxOkPackets;
  int nRxPendBuffers;
  int nRxSchedBuffers;
  int nRxErrorPackets;
  int nRxFragmentErrors;
} TCPIP_MAC_RX_STATISTICS;
```

Members

Members	Description
int nRxOkPackets;	number of OK RX packets
int nRxPendBuffers;	number of unacknowledged pending RX buffers in the driver queues. If each incoming packet fits within a RX buffer (the RX buffer is large enough) than this corresponds to the number of unacknowledged pending RX packets. Otherwise the number of packets is less than the pending buffers.
int nRxSchedBuffers;	number of currently scheduled RX buffers in the driver queues. These are available buffers, ready to receive data
int nRxErrorPackets;	number of RX packets with errors
int nRxFragmentErrors;	number of RX fragmentation errors

Description

MAC RX Statistics Data

This structure defines the RX statistics data for the MAC driver.

Remarks

This statistics are recorded by the driver, not by the hardware.

TCPIP_MAC_TX_STATISTICS Structure

MAC transmit statistics data gathered at run time.

File

tcpip_mac.h

C

```
typedef struct {
  int nTxOkPackets;
  int nTxPendBuffers;
  int nTxErrorPackets;
  int nTxQueueFull;
} TCPIP_MAC_TX_STATISTICS;
```

Members

Members	Description
int nTxOkPackets;	number of OK transmitted packets
int nTxPendBuffers;	number of unacknowledged pending TX buffers in the driver queues. This is equal with pending TX packets when each packet is contained within a TX buffer.
int nTxErrorPackets;	number of packets that could not be transmitted
int nTxQueueFull;	number of times the TX queue was full this may signal that the number of TX descriptors is too small

Description

MAC TX Statistics Data

This structure defines the TX statistics data for the MAC driver.

Remarks

This statistics are recorded by the driver, not by the hardware.

TCPIP_MAC_INIT Structure

Contains all the data necessary to initialize the MAC device.

File

tcpip_mac.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  const TCPIP_MAC_MODULE_CTRL* const macControl;
  const void* const moduleData;
} TCPIP_MAC_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
const TCPIP_MAC_MODULE_CTRL* const macControl;	Stack prepared data
const void* const moduleData;	Driver specific initialization data

Description

MAC Device Driver Initialization Data

This data structure contains all the data necessary to initialize the MAC device.

Remarks

A pointer to a structure of this format containing the desired initialization data must be passed into the TCPIP_MAC_Initialize routine.

TCPIP_MAC_PARAMETERS Structure

Data structure that tells the MAC run time parameters.

File

tcpip_mac.h

C

```
typedef struct {
  TCPIP_MAC_ADDR ifPhyAddress;
  TCPIP_MAC_PROCESS_FLAGS processFlags;
  TCPIP_MAC_TYPE macType;
  TCPIP_MAC_LINK_MTU linkMtu;
} TCPIP_MAC_PARAMETERS;
```

Members

Members	Description
TCPIP_MAC_ADDR ifPhyAddress;	Physical address of the interface. MAC sets this field as part of the initialization process.
TCPIP_MAC_PROCESS_FLAGS processFlags;	MAC process flags. The stack will use this value to call into the MAC process function after receiving a MAC event.
TCPIP_MAC_TYPE macType;	MAC type: ETH, Wi-Fi, etc.
TCPIP_MAC_LINK_MTU linkMtu;	MAC link MTU size

Description

MAC Run time parameters

This is the data structure that the MAC user (TCP/IP stack) passes on to the MAC driver after the MAC initialization time to retrieve the settings resulted.

Remarks

None.

TCPIP_MAC_PKT_AckFDbg Type

File

tcpip_mac.h

C

```
typedef void (* TCPIP_MAC_PKT_AckFDbg)(TCPIP_MAC_PACKET* pPkt, TCPIP_MAC_PKT_ACK_RES ackRes, int moduleId);
```

Description

This is type TCPIP_MAC_PKT_AckFDbg.

TCPIP_MAC_PKT_AllocFDbg Type

File

tcpip_mac.h

C

```
typedef TCPIP_MAC_PACKET* (* TCPIP_MAC_PKT_AllocFDbg)(uint16_t pktLen, uint16_t segLoadLen,
TCPIP_MAC_PACKET_FLAGS flags, int moduleId);
```

Description

This is type TCPIP_MAC_PKT_AllocFDbg.

TCPIP_MAC_PKT_FreeFDbg Type

File

tcpip_mac.h

C

```
typedef void (* TCPIP_MAC_PKT_FreeFDbg)(TCPIP_MAC_PACKET* pPkt, int moduleId);
```

Description

This is type TCPIP_MAC_PKT_FreeFDbg.

TCPIP_MAC_HEAP_CallocFDbg Type

File

tcpip_mac.h

C

```
typedef void* (* TCPIP_MAC_HEAP_CallocFDbg)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nElems, size_t elemSize,
int moduleId, int lineNo);
```

Description

This is type TCPIP_MAC_HEAP_CallocFDbg.

TCPIP_MAC_HEAP_FreeFDbg Type

File

tcpip_mac.h

C

```
typedef size_t (* TCPIP_MAC_HEAP_FreeFDbg)(TCPIP_MAC_HEAP_HANDLE heapH, const void* pBuff, int moduleId,
int lineNo);
```

Description

This is type TCPIP_MAC_HEAP_FreeFDbg.

TCPIP_MAC_HEAP_MallocFDbg Type

File

tcpip_mac.h

C

```
typedef void* (* TCPIP_MAC_HEAP_MallocFDbg)(TCPIP_MAC_HEAP_HANDLE heapH, size_t nBytes, int moduleId, int
lineNo);
```

Description

This is type TCPIP_MAC_HEAP_MallocFDbg.

TCPIP_MAC_STATISTICS_REG_ENTRY Structure

Describes a MAC hardware statistics register.

File

tcpip_mac.h

C

```
typedef struct {
  char registerName[8];
  uint32_t registerValue;
} TCPIP_MAC_STATISTICS_REG_ENTRY;
```

Members

Members	Description
char registerName[8];	name of the hardware register
uint32_t registerValue;	value of the hardware register

Description

MAC Hardware Statistics Register Entry

This structure defines an interface for gathering the MAC hardware statistics registers data.

Remarks

None

TCPIP_MAC_TYPE Enumeration

List of the MAC types.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_TYPE_NONE = 0,
  TCPIP_MAC_TYPE_ETH,
  TCPIP_MAC_TYPE_WLAN,
  TCPIP_MAC_TYPES
} TCPIP_MAC_TYPE;
```

Members

Members	Description
TCPIP_MAC_TYPE_NONE = 0	invalid/unknown MAC type
TCPIP_MAC_TYPE_ETH	wired Ethernet MAC type
TCPIP_MAC_TYPE_WLAN	wireless, Wi-Fi type MAC
TCPIP_MAC_TYPES	supported types

Description

MAC Types

List of specific MAC types that indicate to the user of the MAC (TCP/IP stack) the actual type of the MAC driver.

Remarks

Other types will be eventually added.

TCPIP_MAC_SYNCH_REQUEST Enumeration

Defines the possible MAC synchronization request types.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_SYNCH_REQUEST_NONE = 0,
  TCPIP_MAC_SYNCH_REQUEST_OBJ_CREATE,
  TCPIP_MAC_SYNCH_REQUEST_OBJ_DELETE,
  TCPIP_MAC_SYNCH_REQUEST_OBJ_LOCK,
  TCPIP_MAC_SYNCH_REQUEST_OBJ_UNLOCK,
  TCPIP_MAC_SYNCH_REQUEST_CRIT_ENTER,
  TCPIP_MAC_SYNCH_REQUEST_CRIT_LEAVE
} TCPIP_MAC_SYNCH_REQUEST;
```

Members

Members	Description
TCPIP_MAC_SYNCH_REQUEST_NONE = 0	no request
TCPIP_MAC_SYNCH_REQUEST_OBJ_CREATE	request to create a synchronization object usually a binary semaphore
TCPIP_MAC_SYNCH_REQUEST_OBJ_DELETE	request to delete a previously created synchronization object
TCPIP_MAC_SYNCH_REQUEST_OBJ_LOCK	request to lock access using a previously created synchronization object
TCPIP_MAC_SYNCH_REQUEST_OBJ_UNLOCK	request to unlock/release access using a previously created synchronization object
TCPIP_MAC_SYNCH_REQUEST_CRIT_ENTER	request to enter a critical section
TCPIP_MAC_SYNCH_REQUEST_CRIT_LEAVE	request to exit from a critical section

Description

TCP/IP MAC Synchronization object request

TCP/IP MAC synchronization request codes.

This enumeration defines all the possible synchronization actions that can be requested by the MAC to the stack at run time.

Remarks

None.

TCPIP_MAC_SynchReqF Type

MAC synchronization request function definition.

File

tcpip_mac.h

C

```
typedef bool (* TCPIP_MAC_SynchReqF)(void* synchHandle, TCPIP_MAC_SYNCH_REQUEST request);
```

Returns

- true - if the call is successful
- false - if the call failed

Description

Synchronization request Function: typedef bool (*TCPIP_MAC_SynchReqF)(void* synchHandle, TCPIP_MAC_SYNCH_REQUEST request);

This function describes the MAC synchronization request function. The MAC driver will call this function whenever it needs to create, delete, lock or unlock a synchronization object.

Remarks

None.

Parameters

Parameters	Description
synchHandle	• pointer to a valid storage area • for TCPIP_MAC_SYNCH_REQUEST_OBJ_CREATE it is an address that will store a handle to identify the synchronization object to the OS (OSAL). • for other synchronization object request types the handle has to be valid and identify the synchronization object to the OS (OSAL). • for Critical sections it maintains OS passed info
request	a TCPIP_MAC_SYNCH_REQUEST type

TCPIP_MODULE_MAC_ID Enumeration

IDs of the MAC module supported by the stack.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MODULE_MAC_NONE = 0x1000,
  TCPIP_MODULE_MAC_ENCJ60 = 0x1010,
  TCPIP_MODULE_MAC_ENCJ60_0 = 0x1010,
  TCPIP_MODULE_MAC_ENCJ600 = 0x1020,
  TCPIP_MODULE_MAC_ENCJ600_0 = 0x1020,
  TCPIP_MODULE_MAC_97J60 = 0x1030,
  TCPIP_MODULE_MAC_97J60_0 = 0x1030,
  TCPIP_MODULE_MAC_PIC32INT = 0x1040,
  TCPIP_MODULE_MAC_PIC32INT_0 = 0x1040,
  TCPIP_MODULE_MAC_MRF24W = 0x1050,
  TCPIP_MODULE_MAC_MRF24W_0 = 0x1050,
  TCPIP_MODULE_MAC_MRF24WN = 0x1060,
  TCPIP_MODULE_MAC_MRF24WN_0 = 0x1060,
```

```
  TCPIP_MODULE_MAC_EXTERNAL = 0x4000
} TCPIP_MODULE_MAC_ID;
```

Members

Members	Description
TCPIP_MODULE_MAC_NONE = 0x1000	unspecified/unknown MAC
TCPIP_MODULE_MAC_ENCJ60 = 0x1010	External ENC28J60 device: room for 16 ENCJ60 devices
TCPIP_MODULE_MAC_ENCJ60_0 = 0x1010	alternate numbered name
TCPIP_MODULE_MAC_ENCJ600 = 0x1020	External ENCX24J600 device: room for 16 ENCJ600 devices
TCPIP_MODULE_MAC_ENCJ600_0 = 0x1020	alternate numbered name
TCPIP_MODULE_MAC_97J60 = 0x1030	ETH97J60 device: room for 16 97J60 devices
TCPIP_MODULE_MAC_97J60_0 = 0x1030	alternate numbered name
TCPIP_MODULE_MAC_PIC32INT = 0x1040	internal/embedded PIC32 MAC: room for 16 PIC32 devices
TCPIP_MODULE_MAC_PIC32INT_0 = 0x1040	alternate numbered name
TCPIP_MODULE_MAC_MRF24W = 0x1050	MRF24W Wi-Fi MAC: room for 16 MRF24W devices
TCPIP_MODULE_MAC_MRF24W_0 = 0x1050	alternate numbered name
TCPIP_MODULE_MAC_MRF24WN = 0x1060	MRF24WN Wi-Fi MAC: room for 16 MRF24WN devices
TCPIP_MODULE_MAC_MRF24WN_0 = 0x1060	alternate numbered name
TCPIP_MODULE_MAC_EXTERNAL = 0x4000	External, non MCHP, MAC modules

Description

MAC Modules ID

This type defines the supported MAC types.

Remarks

Not all types listed in this enumeration are supported. New MAC types will be added as needed.

TCPIP_MODULE_MAC_MRF24WN_CONFIG Structure

File

tcpip_mac.h

C

```
typedef struct {
} TCPIP_MODULE_MAC_MRF24WN_CONFIG;
```

Description

This is type TCPIP_MODULE_MAC_MRF24WN_CONFIG.

TCPIP_MAC_RX_FILTER_TYPE Enumeration

Defines the possible MAC RX filter types.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_RX_FILTER_TYPE_BCAST_ACCEPT,
  TCPIP_MAC_RX_FILTER_TYPE_MCAST_ACCEPT,
  TCPIP_MAC_RX_FILTER_TYPE_UCAST_ACCEPT,
  TCPIP_MAC_RX_FILTER_TYPE_UCAST_OTHER_ACCEPT,
  TCPIP_MAC_RX_FILTER_TYPE_RUNT_REJECT,
  TCPIP_MAC_RX_FILTER_TYPE_RUNT_ACCEPT,
  TCPIP_MAC_RX_FILTER_TYPE_CRC_ERROR_REJECT,
  TCPIP_MAC_RX_FILTER_TYPE_CRC_ERROR_ACCEPT,
  TCPIP_MAC_RX_FILTER_TYPE_DEFAULT
} TCPIP_MAC_RX_FILTER_TYPE;
```

Members

Members	Description
TCPIP_MAC_RX_FILTER_TYPE_BCAST_ACCEPT	Broadcast packets are accepted
TCPIP_MAC_RX_FILTER_TYPE_MCAST_ACCEPT	Multicast packets are accepted
TCPIP_MAC_RX_FILTER_TYPE_UCAST_ACCEPT	Unicast packets to a this host are accepted
TCPIP_MAC_RX_FILTER_TYPE_UCAST_OTHER_ACCEPT	Unicast packets to a different host are accepted
TCPIP_MAC_RX_FILTER_TYPE_RUNT_REJECT	Runt packets (< 64 bytes) are rejected
TCPIP_MAC_RX_FILTER_TYPE_RUNT_ACCEPT	Runt packets (< 64 bytes) are accepted
TCPIP_MAC_RX_FILTER_TYPE_CRC_ERROR_REJECT	Packets with CRC errors are rejected
TCPIP_MAC_RX_FILTER_TYPE_CRC_ERROR_ACCEPT	Packets with CRC errors are accepted
TCPIP_MAC_RX_FILTER_TYPE_DEFAULT	Default RX filter: will accept most valid packets

Description

MAC RX Filter Types

This enumeration defines the RX filtering capabilities for the TCP/IP MAC driver.

Remarks

The Ethernet/Wi-Fi controllers have an RX filter module that takes part in the packet acceptance decision.

Multiple values can be OR-ed together.

There usually is a priority in the RX filter processing for a MAC module. In order for a packet to be accepted, it has to be specifically accepted by a filter. Once a filter accepts or rejects a packet, further filters are not tried. If a packet isn't rejected/accepted after all filters are tried, it will be rejected by default!

Not all MACs support all the RX filter types.

Current implementation does not suport more than 16 filters.

TCPIP_MAC_LINK_MTU Enumeration

MTU size correspondig to the MAC types.

File

tcpip_mac.h

C

```
typedef enum {
  TCPIP_MAC_LINK_MTU_DEFAULT = 1500,
  TCPIP_MAC_LINK_MTU_ETH = 1500,
  TCPIP_MAC_LINK_MTU_WLAN = 1500
} TCPIP_MAC_LINK_MTU;
```

Members

Members	Description
TCPIP_MAC_LINK_MTU_DEFAULT = 1500	default MTU link
TCPIP_MAC_LINK_MTU_ETH = 1500	wired Ethernet MAC type
TCPIP_MAC_LINK_MTU_WLAN = 1500	wireless, Wi-Fi type MAC

Description

Link MTU per MAC Type

Lists the Maximum Transmission Unit corresponding to a MAC type.

Remarks

The MTU is usually set by the standards. For special links the values can be updated.

The default value is normally used. However the individual MAC drivers can use the specific symols

Files

Files

Name	Description
tcpip_mac.h	

tcpip_mac_config.h	Configuration file

Description

This section lists the source and header files used by the library.

tcpip_mac.h

Enumerations

	Name	Description
	TCPIP_MAC_ACTION	Network interface action for initialization/deinitialization
	TCPIP_MAC_EVENT	Defines the possible MAC event types.
	TCPIP_MAC_LINK_MTU	MTU size correspondig to the MAC types.
	TCPIP_MAC_PACKET_FLAGS	Flags belonging to MAC packet.
	TCPIP_MAC_PKT_ACK_RES	List of MAC return codes for a packet acknowledge function.
	TCPIP_MAC_POWER_MODE	This is type TCPIP_MAC_POWER_MODE.
	TCPIP_MAC_PROCESS_FLAGS	List of the MAC processing flags.
	TCPIP_MAC_RES	List of return codes from MAC functions.
	TCPIP_MAC_RX_FILTER_TYPE	Defines the possible MAC RX filter types.
	TCPIP_MAC_SEGMENT_FLAGS	Flags belonging to MAC data segment.
	TCPIP_MAC_SYNCH_REQUEST	Defines the possible MAC synchronization request types.
	TCPIP_MAC_TYPE	List of the MAC types.
	TCPIP_MODULE_MAC_ID	IDs of the MAC module supported by the stack.

Functions

	Name	Description
	TCPIP_MAC_Close	MAC driver close function.
	TCPIP_MAC_ConfigGet	Gets the current MAC driver configuration.
	TCPIP_MAC_Deinitialize	MAC driver deinitialization function.
	TCPIP_MAC_EventAcknowledge	This function acknowledges a previously reported MAC event.
	TCPIP_MAC_EventMaskSet	MAC events report enable/disable function.
	TCPIP_MAC_EventPendingGet	Returns the currently pending MAC events.
	TCPIP_MAC_Initialize	MAC driver initialization function.
	TCPIP_MAC_LinkCheck	MAC link checking function.
	TCPIP_MAC_Open	MAC driver open function.
	TCPIP_MAC_PacketRx	A packet is returned if such a pending packet exists.
	TCPIP_MAC_PacketTx	MAC driver transmit function.
	TCPIP_MAC_ParametersGet	MAC parameter get function.
	TCPIP_MAC_Process	MAC periodic processing function.
	TCPIP_MAC_RegisterStatisticsGet	Gets the current MAC hardware statistics registers.
	TCPIP_MAC_Reinitialize	MAC driver reinitialization function.
	TCPIP_MAC_RxFilterHashTableEntrySet	Sets the current MAC hash table receive filter.
	TCPIP_MAC_StatisticsGet	Gets the current MAC statistics.
	TCPIP_MAC_Status	Provides the current status of the MAC driver module.
	TCPIP_MAC_Tasks	Maintains the MAC driver's state machine.

Structures

	Name	Description
	_tag_MAC_DATA_SEGMENT	A data segment that's part of a TX/RX packet.
	_tag_TCPIP_MAC_PACKET	A TX/RX packet descriptor.
	TCPIP_MAC_ADDR	Definition of a MAC address.
	TCPIP_MAC_DATA_SEGMENT	A data segment that's part of a TX/RX packet.
	TCPIP_MAC_ETHERNET_HEADER	Definition of a MAC frame header.
	TCPIP_MAC_INIT	Contains all the data necessary to initialize the MAC device.
	TCPIP_MAC_MODULE_CTRL	Data structure that's passed to the MAC at the initialization time.
	TCPIP_MAC_PACKET_RX_STAT	Status of a received packet.

	TCPIP_MAC_PARAMETERS	Data structure that tells the MAC run time parameters.
	TCPIP_MAC_RX_STATISTICS	MAC receive statistics data gathered at run time.
	TCPIP_MAC_STATISTICS_REG_ENTRY	Describes a MAC hardware statistics register.
	TCPIP_MAC_TX_STATISTICS	MAC transmit statistics data gathered at run time.
	TCPIP_MODULE_MAC_MRF24W_CONFIG	This is type TCPIP_MODULE_MAC_MRF24W_CONFIG.
	TCPIP_MODULE_MAC_MRF24WN_CONFIG	This is type TCPIP_MODULE_MAC_MRF24WN_CONFIG.
	TCPIP_MODULE_MAC_PIC32INT_CONFIG	Data that's passed to the MAC at initialization time.

Types

	Name	Description
	TCPIP_MAC_EventF	MAC event notification handler.
	TCPIP_MAC_HANDLE	Handle to a MAC driver.
	TCPIP_MAC_HEAP_CallocF	MAC allocation function prototype.
	TCPIP_MAC_HEAP_CallocFDbg	This is type TCPIP_MAC_HEAP_CallocFDbg.
	TCPIP_MAC_HEAP_FreeF	MAC allocation function prototype.
	TCPIP_MAC_HEAP_FreeFDbg	This is type TCPIP_MAC_HEAP_FreeFDbg.
	TCPIP_MAC_HEAP_HANDLE	A handle used for memory allocation functions.
	TCPIP_MAC_HEAP_MallocF	MAC allocation function prototype.
	TCPIP_MAC_HEAP_MallocFDbg	This is type TCPIP_MAC_HEAP_MallocFDbg.
	TCPIP_MAC_PACKET	Forward reference to a MAC packet.
	TCPIP_MAC_PACKET_ACK_FUNC	Prototype of a MAC packet acknowledge function.
	TCPIP_MAC_PKT_AckF	MAC packet acknowledge function prototype.
	TCPIP_MAC_PKT_AckFDbg	This is type TCPIP_MAC_PKT_AckFDbg.
	TCPIP_MAC_PKT_AllocF	MAC packet allocation function prototype.
	TCPIP_MAC_PKT_AllocFDbg	This is type TCPIP_MAC_PKT_AllocFDbg.
	TCPIP_MAC_PKT_FreeF	MAC packet free function prototype.
	TCPIP_MAC_PKT_FreeFDbg	This is type TCPIP_MAC_PKT_FreeFDbg.
	TCPIP_MAC_SynchReqF	MAC synchronization request function definition.

Description

MAC Module Definitions for the Microchip TCP/IP Stack

File Name

tcpip_mac.h

Company

Microchip Technology Inc.

tcpip_mac_config.h

Configuration file

Macros

	Name	Description
	TCPIP_EMAC_ETH_OPEN_FLAGS	Flags to use for the Ethernet connection A TCPIP_ETH_OPEN_FLAGS value. Set to TCPIP_ETH_OPEN_DEFAULT unless very good reason to use different value
	TCPIP_EMAC_PHY_ADDRESS	The PHY address, as configured on the board. By default all the PHYs respond to address 0
	TCPIP_EMAC_PHY_CONFIG_FLAGS	Flags to configure the MAC ->PHY connection a DRV_ETHPHY_CONFIG_FLAGS This depends on the actual connection (MII/RMII, default/alternate I/O) The DRV_ETHPHY_CFG_AUTO value will use the configuration fuses setting
	TCPIP_EMAC_PHY_LINK_INIT_DELAY	The value of the delay for the link initialization, ms This insures that the PHY is ready to transmit after it is reset A usual value is 500 ms. Adjust to your needs.

	TCPIP_EMAC_RX_BUFF_SIZE	Size of a RX packet buffer. Should be multiple of 16. This is the size of all receive packet buffers processed by the ETHC. The size should be enough to accommodate any network received packet. If the packets are larger, they will have to take multiple RX buffers and the packet manipulation is less efficient. #define TCPIP_EMAC_RX_BUFF_SIZE 512 Together with TCPIP_EMAC_RX_DEDICATED_BUFFERS it has impact on TCPIP_STACK_DRAM_SIZE setting.
	TCPIP_EMAC_RX_DEDICATED_BUFFERS	Number of MAC dedicated RX packet buffers. These buffers are always owned by the MAC. Note that the MAC driver allocates these buffers for storing the incoming network packets. The bigger the storage capacity, the higher data throughput can be obtained. Note that these packet buffers are allocated from the private TCP/IP heap that is specified by the TCPIP_STACK_DRAM_SIZE setting.
	TCPIP_EMAC_RX_DESCRIPTORS	Number of the RX descriptors to be created. If not using the run time replenish mechanism (see below) it should match the number of dedicated buffers: TCPIP_EMAC_RX_DEDICATED_BUFFERS; Otherwise it should be bigger than the sum of dedicated + non-dedicated buffers: TCPIP_EMAC_RX_DESCRIPTORS > TCPIP_EMAC_RX_DEDICATED_BUFFERS + replenish_buffers
	TCPIP_EMAC_RX_FILTERS	MAC RX Filters These filters define the packets that are accepted and rejected by the MAC driver Adjust to your needs The default value allows the processing of unicast, multicast and broadcast packets that have a valid CRC
	TCPIP_EMAC_RX_FRAGMENTS	MAC maximum number of supported fragments. Based on the values of TCPIP_EMAC_RX_MAX_FRAME and TCPIP_EMAC_RX_BUFF_SIZE an incoming frame may span multiple RX buffers (fragments). Note that excessive fragmentation leads to performance degradation. The default and recommended value should be 1. #define TCPIP_EMAC_RX_FRAGMENTS 1 Alternatively you can use the calculation of the number of fragments based on the selected RX sizes:
	TCPIP_EMAC_RX_INIT_BUFFERS	Number of non-dedicated buffers for the MAC initialization Buffers allocated at the MAC driver initialization.
	TCPIP_EMAC_RX_LOW_FILL	Number of RX buffers to allocate when below threshold condition is detected. If 0, the MAC driver will allocate (scheduled buffers - rxThres) If !0, the MAC driver will allocate exactly TCPIP_EMAC_RX_LOW_FILL buffers
	TCPIP_EMAC_RX_LOW_THRESHOLD	Minumum threshold for the buffer replenish process. Whenever the number of RX scheduled buffers is <= than this threshold the MAC driver will allocate new non-dedicated buffers (meaning that they will be released to the TCP/IP heap once they are processed). Setting this value to 0 disables the buffer replenishing process.
	TCPIP_EMAC_RX_MAX_FRAME	Maximum MAC supported RX frame size. Any incoming ETH frame that's longer than this size will be discarded. The default value is 1536 (allows for VLAN tagged frames, although the VLAN tagged frames are discarded). Normally there's no need to touch this value unless you know exactly the maximum size of the frames you want to process or you need to control packets fragmentation (together with the TCPIP_EMAC_RX_BUFF_SIZE).
	TCPIP_EMAC_TX_DESCRIPTORS	Number of the TX descriptors to be created. Because a TCP packet can span at most 3 buffers, the value should always be >= 4 The amount of memory needed per descriptor is not high (around 24 bytes) so when high MAC TX performance is needed make sure that this number is >= 8.

Description

MAC Configuration file

This file contains the MAC module configuration options

File Name

tcpip_mac_config.h

Company

Microchip Technology Inc.

Manager Module

This section describes the TCP/IP Stack Library Manager module.

Introduction

TCP/IP Stack Library Manager Module for Microchip Microcontrollers

This library provides the API of the Manager module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The TCP/IP manager provides access to the stack initialization, configuration and status functions.

It also process the packets received from different network interfaces (MAC drivers) and dispatches them based on their type to the proper higher layer in the stack.

Using the Library

This topic describes the basic architecture of the Manager TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `tcpip_manager.h`

The interface to the Manager TCP/IP Stack library is defined in the `tcpip_manager.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the Manager TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

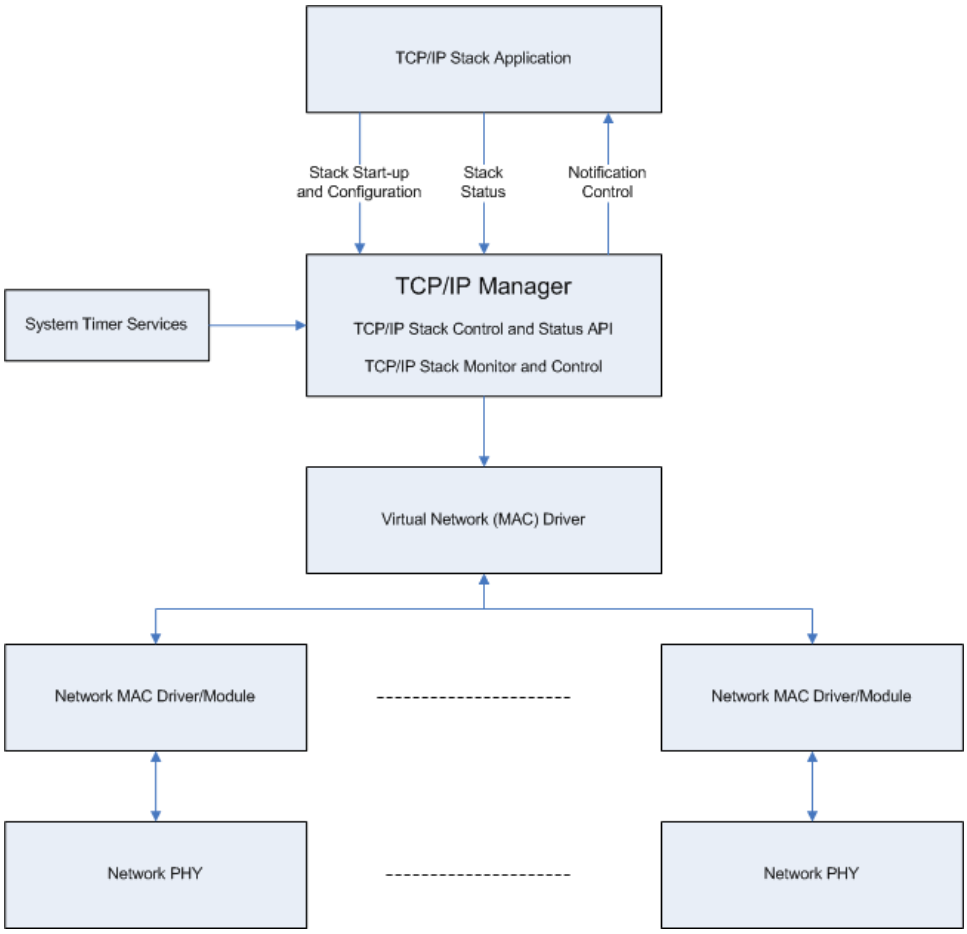
This library provides the API of the Manager TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

The TCP/IP manager allows the user of the stack to initialize, control and get current status of the TCP/IP stack.

It is also responsible for the receiving of the network packets and their dispatch to the stack processing layers.

Manager Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Manager module.

Library Interface Section	Description
Task and Initialize Functions	Routines to Initialize and Deinitialize this Module
Network Up/Down/Linked Functions	Routines to Set and Get the Network's Up/Down Status
Network Status and Control Functions	Routines to Set and Get the Various Network Controls
Network Address Status and Control Functions	Routines to Set and Get the Network's Address Status
Network Structure Storage Functions	Network Information Routines
Data Types and Constants	This section provides various definitions describing this API

How the Library Works

This topic describes how the TCP/IP Stack Library Manager module operates.

Description

The Manager module of the TCP/IP stack is the central module in the stack. It is the module that allows the client to configure and initialize the stack and also to get current status and parameters of the network interfaces that the stack supports.

The stack manager controls the communication to the underlying virtual MAC driver that implements the interface to a specific network channel. The various network interfaces that the TCP/IP stack supports are brought up or torn down by the stack manager in response to the stack client commands.

The manager is also responsible for providing the TCP/IP client with notifications of the events that occur in the stack at the network interface level. A dynamic registration mechanism is provided for this.

Another important functionality of the stack manager is the dispatch of the incoming network packets. The manager is notified by the underlying interface drivers of the traffic related event (transmit, receive, errors, etc.) As a response to those events the manager gets the pending receive packets from the virtual MAC driver and, based on their type, dispatches them to the appropriate higher level protocol for further processing (IPv4, ARP, IPv6).

As part of the event notification provided by the MAC driver the TCP/IP stack manager calls back into the MAC driver when this one needs processing time and can maintain statistics counters, link status, etc.

The MPLAB Harmony framework on the system running the TCP/IP stack interfaces to the stack manager mainly by using three functions, TCPIP_STACK_Initialize and TCPIP_STACK_Deinitialize for TCP/IP stack initialization, and TCPIP_STACK_Task for periodic processing and the advance of the stack state machine.

The stack is capable of running either under an Operating System (RTOS) or in a bare-metal environment. When running in a bare-metal environment a "cooperative multi-tasking" approach is needed by the TCP/IP stack, meaning that control has to reach the TCPIP_STACK_Task function. That is the system will make periodic calls to this function which allows the TCP/IP stack to dequeue incoming packets and the corresponding state machines in the stack to advance. The application has to be aware that it cannot monopolize the processor for long periods of time, and needs to relinquish control so that the system loops perform their tasks.

In an RTOS environment it is preferred to have the stack manager run in its own thread. This thread will be blocked waiting for network interface events and when such an event occurs it will wake up and process the data. This is a more efficient way of using the CPU resources.

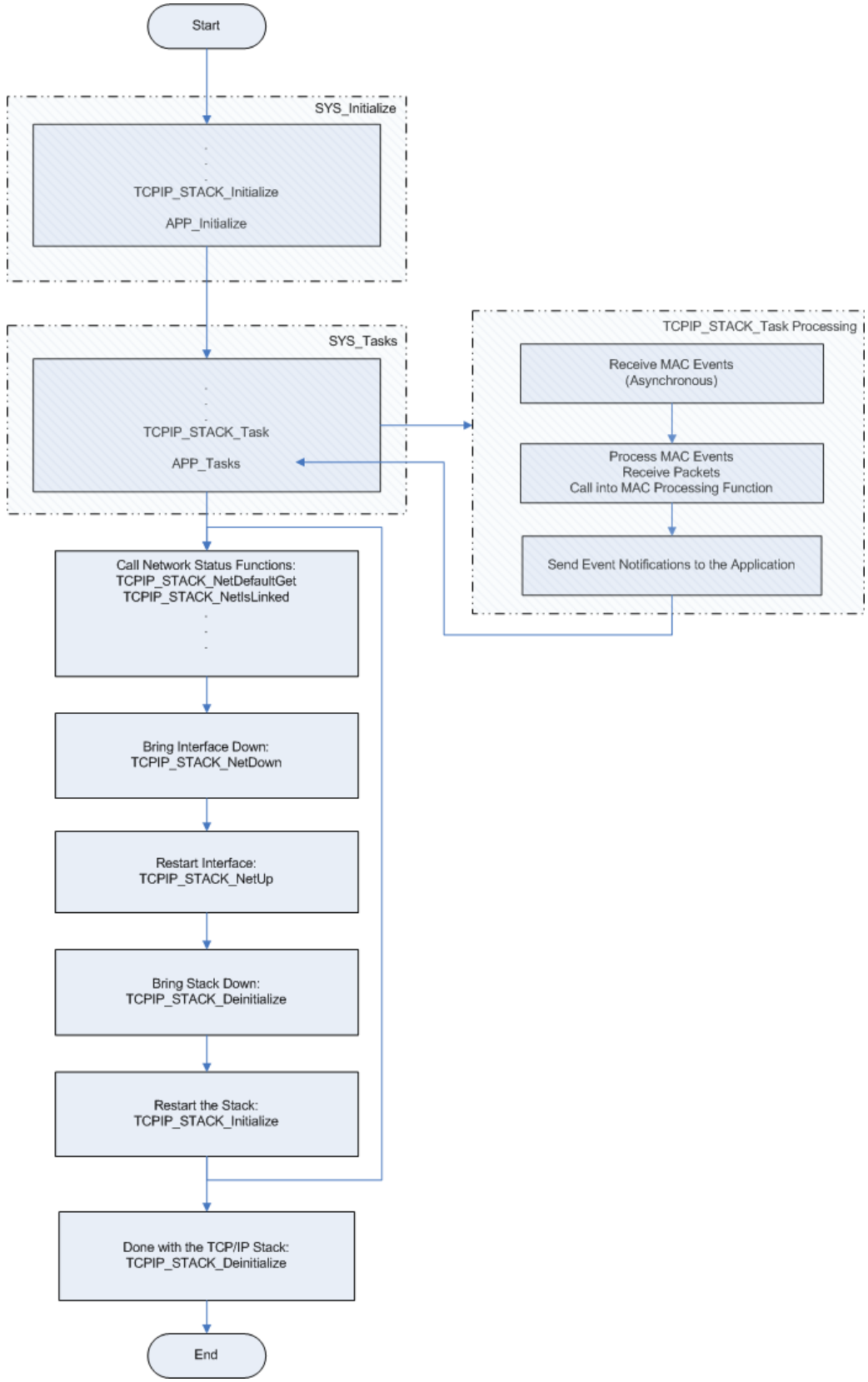
Refer to Volume IV: MPLAB Harmony Framework Reference for information on configuring the MPLAB Harmony framework to run in a RTOS or bare-metal environment.

Core Functionality

This topic describes the core functionality of the TCP/IP Stack Library Manager module.

Description

The following diagram provides an overview for the operation and use of the Manager module in a typical MPLAB Harmony framework (bare-metal, non-OS environment).



The TCP/IP stack is initialized as a part of the system initialization by a call to TCPIP_STACK_Initialize. By configuring the stack, the application can pass run-time initialization data to the stack to select both the network parameters of the interfaces (network addresses, masks, physical addresses, NetBIOS names, DHCP operation, etc.) and the parameters for the modules contained in the stack (TCP, UDP, ARP, IPv4, IPv6, etc.).

Once the stack is initialized the application can use the stack manager calls to inquire about the stack current settings (current addresses obtained by DHCP or IPv6 advertisements messages, etc.), to check the physical link status or to set new parameters to the stack.

It should be noted that different interfaces can be stopped and restarted dynamically (TCPIP_STACK_NetDown, TCPIP_STACK_NetUp) without the stack being killed. When a particular interface is brought up only the network parameters for that interface need to be supplied (no parameters

for the TCP/IP stack modules are needed).

The entire stack can be stopped and restarted (TCPIP_STACK_Deinitialize, TCPIP_STACK_Initialize). Reinitializing the stack allows the stack initialization with a complete new set of network and/or module parameters.

The stack processing is done in the TCPIP_STACK_Task function. As explained in the previous paragraph, processing time is required by the stack, meaning that periodic calls have to be made by the system/framework to the TCPIP_STACK_Task function. When execution reaches this function the stack maintains and advances its internal state machine. The most important part of this is the processing of the events that are reported by the underlying MAC drivers (through the use of the virtual MAC driver, which provides an interrupt signaling/event notification mechanism). As a response to the MAC events, the stack manager will retrieve packets from the internal MAC receive queues and dispatch them for processing. It will also call into MAC processing functions if required and provide processing for the stack modules that are the recipients of the dispatched network packets. The stack manager will also notify the application of the network events if the application dynamically registered an event notification handler with the stack.

It should be noted that the stack manager also requires a periodic tick signal for which it uses the MPLAB Harmony system timer services (see Timer System Service Library).

If the application no longer uses the TCP/IP stack, it is important that it calls the TCPIP_STACK_Deinitialize function. This function closes any open network connections (sockets, etc.), closes the underlying MAC and PHY drivers and all of the modules in the stack, and frees all of the resources that were internally allocated by the stack as part of the initialization and run-time processing.

Configuring the Library

The configuration of the Manager TCP/IP Stack is based on the file `system_config.h`, which may include `tcpip_config.h`.

This header file contains the configuration of the stack and the modules that are included in the current build of the stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Manager module of the TCP/IP Stack Library.

Description

The Manager module is part of the core functionality of the TCP/IP Stack Library. Please refer to the Building the Library topic in the TCP/IP Stack Library section for build information.

Library Interface

a) Helper Functions

	Name	Description
	TCPIP_Helper_FormatNetBIOSName	Formats a string to a valid NetBIOS name.
	TCPIP_Helper_htonl	Conversion routines from network order to host order and reverse.
	TCPIP_Helper_htons	This is function TCPIP_Helper_htons.
	TCPIP_Helper_IPAddressToString	Converts an IPV4 address to an ASCII string.
	TCPIP_Helper_IPv6AddressToString	Converts an IPv6 address to a string representation.
	TCPIP_Helper_IsBcastAddress	Checks if an IPv4 address is a broadcast address.
	TCPIP_Helper_IsMcastAddress	Checks if an IPv4 address is a multicast address.
	TCPIP_Helper_IsPrivateAddress	Detects a private (non-routable) address.
	TCPIP_Helper_MACAddressToString	Converts a MAC address to a string.
	TCPIP_Helper_ntohl	!defined(__PIC32MX__)
	TCPIP_Helper_ntohs	This is function TCPIP_Helper_ntohs.
	TCPIP_Helper_StringToIPAddress	Converts an ASCII string to an IPV4 address.
	TCPIP_Helper_StringToIPv6Address	Converts a string to an IPv6 address.
	TCPIP_Helper_StringToMACAddress	Converts a string to an MAC address.
	TCPIP_Helper_htonll	This is function TCPIP_Helper_htonll.
	TCPIP_Helper_ntohll	This is function TCPIP_Helper_ntohll.
	TCPIP_Helper_SecurePortGetByIndex	Returns the secure port belonging to a specified index.
	TCPIP_Helper_SecurePortSet	Sets the required port secure connection status.
	TCPIP_Helper_TCPSecurePortGet	Checks if the required TCP port is a secure port.
	TCPIP_Helper_UDPSecurePortGet	Checks if the required UDP port is a secure port.
	TCPIP_Helper_Base64Decode	Helper function to decode a Base-64 encoded array.
	TCPIP_Helper_Base64Encode	Helper function to encode to Base-64.

b) Task and Initialize Functions

	Name	Description
	TCPIP_STACK_SetLocalMasks	Sets the local/non-local masks for network interface.
	TCPIP_STACK_SetLocalMasksType	Sets the local/non-local masks type for network interface.
	TCPIP_STACK_HandlerDeregister	Deregisters an event notification handler.
	TCPIP_STACK_HandlerRegister	Sets a new event notification handler.
	TCPIP_STACK_Initialize	Stack initialization function.
	TCPIP_STACK_Deinitialize	Stack deinitialization function.
	TCPIP_STACK_Task	TCP/IP Stack task function.
	TCPIP_STACK_VersionGet	Gets the TCP/IP stack version in numerical format.
	TCPIP_STACK_VersionStrGet	Gets the TCP/IP stack version in string format.
	TCPIP_STACK_MACObjectGet	Returns the network MAC driver object of this interface.
	TCPIP_MODULE_SignalFunctionDeregister	Deregisters a signal function for a stack module.
	TCPIP_MODULE_SignalFunctionRegister	Registers a new signal function for a stack module.
	TCPIP_MODULE_SignalGet	Returns the current signals for a TCP/IP module.

c) Network Status and Control Functions

	Name	Description
	TCPIP_STACK_IndexToNet	Network interface handle from interface number.
	TCPIP_STACK_NetBIOSName	Network interface NetBIOS name.
	TCPIP_STACK_NetBiosNameSet	Sets network interface NetBIOS name.
	TCPIP_STACK_NetDefaultGet	Default network interface handle.
	TCPIP_STACK_NetDefaultSet	Sets the default network interface handle.
	TCPIP_STACK_NetHandleGet	Network interface handle from a name.
	TCPIP_STACK_NetIndexGet	Network interface number from interface handle.
	TCPIP_STACK_NetMask	Network interface IPv4 address mask.
	TCPIP_STACK_NetNameGet	Network interface name from a handle.
	TCPIP_STACK_NumberOfNetworksGet	Number of network interfaces in the stack.
	TCPIP_STACK_EventsPendingGet	Returns the currently pending events.
	TCPIP_STACK_NetIsReady	Gets the network interface configuration status.
	TCPIP_STACK_NetMACRegisterStatisticsGet	Get the MAC statistics register data.
	TCPIP_STACK_NetAliasNameGet	Network interface alias name from a handle.
	TCPIP_STACK_InitializeDataGet	Get the TCP/IP stack initialization data.

d) Network Up/Down/Linked Functions

	Name	Description
	TCPIP_STACK_NetIsLinked	Gets the network interface link status.
	TCPIP_STACK_NetIsUp	Gets the network interface up or down status.
	TCPIP_STACK_NetUp	Turns up a network interface. As part of this process, the corresponding MAC driver is initialized.
	TCPIP_STACK_NetDown	Turns down a network interface.

e) Network Address Status and Control Functions

	Name	Description
	TCPIP_STACK_NetAddress	Network interface IPv4 address.
	TCPIP_STACK_NetAddressBcast	Network interface broadcast address.
	TCPIP_STACK_NetAddressDnsPrimary	Network interface DNS address.
	TCPIP_STACK_NetAddressDnsPrimarySet	Sets network interface IPv4 DNS address.
	TCPIP_STACK_NetAddressDnsSecond	Network interface secondary DNS address.
	TCPIP_STACK_NetAddressDnsSecondSet	Sets network interface IPv4 secondary DNS address.
	TCPIP_STACK_NetAddressGateway	Network interface IPv4 gateway address.
	TCPIP_STACK_NetAddressGatewaySet	Sets network interface IPv4 gateway address.
	TCPIP_STACK_NetAddressMac	Network interface MAC address.
	TCPIP_STACK_NetAddressMacSet	Sets network interface MAC address.
	TCPIP_STACK_NetAddressSet	Sets network interface IPv4 address.

	TCPIP_STACK_NetIPv6AddressGet	Gets network interface IPv6 address.
	TCPIP_STACK_ModuleConfigGet	Get stack module configuration data.
	TCPIP_STACK_NetMACIdGet	Get the MAC ID of the network interface.
	TCPIP_STACK_NetMACStatisticsGet	Get the MAC statistics data.
	TCPIP_STACK_Status	Provides the current status of the TCPIP stack module.

f) Network Structure Storage Functions

	Name	Description
	TCPIP_STACK_NetConfigGet	Get stack network interface configuration data.
	TCPIP_STACK_NetConfigSet	Restores stack network interface configuration data.

g) Data Types and Constants

	Name	Description
	TCPIP_LOCAL_MASK_TYPE	Mask operation for local/non-local network interface.
	TCPIP_NET_HANDLE	Defines a network interface handle.
	TCPIP_EVENT	Defines the possible TCPIP event types.
	TCPIP_EVENT_HANDLE	Defines a TCPIP stack event handle.
	TCPIP_STACK_EVENT_HANDLER	Pointer to a function(handler) that will get called to process an event.
	TCPIP_Helper_ntohl	This is macro TCPIP_Helper_ntohl.
	TCPIP_Helper_ntohs	This is macro TCPIP_Helper_ntohs.
	TCPIP_STACK_IF_NAME_ALIAS_ETH	alias for Ethernet interface
	TCPIP_STACK_IF_NAME_ALIAS_UNK	alias for unknown interface
	TCPIP_STACK_IF_NAME_ALIAS_WLAN	alias for Wi-Fi interface
	_IPV6_ADDR_STRUCT	COmplex type to represent an IPv6 address.
	IP_ADDR	Backward compatible definition to represent an IPv4 address.
	IP_ADDRESS_TYPE	Definition of the TCP/IP supported address types.
	IP_MULTI_ADDRESS	Definition to represent multiple IP addresses.
	IPV4_ADDR	Definition to represent an IPv4 address
	IPV6_ADDR	Definition to represent an IPv6 address.
	IPV6_ADDR_HANDLE	Definition to represent an IPv6 address.
	IPV6_ADDR_SCOPE	Definition to represent the scope of an IPv6 address.
	IPV6_ADDR_STRUCT	COmplex type to represent an IPv6 address.
	IPV6_ADDR_TYPE	Definition to represent the type of an IPv6 address.
	TCPIP_NETWORK_CONFIG	Defines the data required to initialize the network configuration.
	TCPIP_NETWORK_CONFIG_FLAGS	Definition of network configuration start-up flags.
	TCPIP_STACK_INIT	Definition to represent the TCP/IP stack initialization/configuration structure.
	TCPIP_STACK_MODULE	List of the TCP/IP stack supported modules.
	TCPIP_STACK_MODULE_CONFIG	Definition to represent a TCP/IP module initialization/configuration structure.
	TCPIP_MODULE_SIGNAL	Lists the signals that are generated by the TCP/IP stack manager and processed by the stack modules.
	TCPIP_MODULE_SIGNAL_FUNC	Pointer to a function(handler) that will get called when a stack internal signal occurred.
	TCPIP_MODULE_SIGNAL_HANDLE	Defines a TCPIP stack signal function handle.
	TCPIP_STACK_IF_NAME_97J60	This is macro TCPIP_STACK_IF_NAME_97J60.
	TCPIP_STACK_IF_NAME_ENCJ60	This is macro TCPIP_STACK_IF_NAME_ENCJ60.
	TCPIP_STACK_IF_NAME_ENCJ600	This is macro TCPIP_STACK_IF_NAME_ENCJ600.
	TCPIP_STACK_IF_NAME_MRF24W	This is macro TCPIP_STACK_IF_NAME_MRF24W.
	TCPIP_STACK_IF_NAME_NONE	Defines the TCP/IP stack supported network interfaces.
	TCPIP_STACK_IF_NAME_PIC32INT	This is macro TCPIP_STACK_IF_NAME_PIC32INT.
	TCPIP_STACK_IF_POWER_DOWN	powered down, not started
	TCPIP_STACK_IF_POWER_FULL	up and running;
	TCPIP_STACK_IF_POWER_LOW	low power mode; not supported now
	TCPIP_STACK_IF_POWER_NONE	unsupported, invalid
	TCPIP_STACK_VERSION_MAJOR	TCP/IP stack version
	TCPIP_STACK_VERSION_MINOR	This is macro TCPIP_STACK_VERSION_MINOR.
	TCPIP_STACK_VERSION_PATCH	This is macro TCPIP_STACK_VERSION_PATCH.

	TCPIP_STACK_VERSION_STR	This is macro TCPIP_STACK_VERSION_STR.
	TCPIP_Helper_ntohll	This is macro TCPIP_Helper_ntohll.
	TCPIP_STACK_IF_NAME_MRF24WN	This is macro TCPIP_STACK_IF_NAME_MRF24WN.

Description

This section describes the Application Programming Interface (API) functions of the Manager module.

Refer to each section for a detailed description.

a) Helper Functions

TCPIP_Helper_FormatNetBIOSName Function

Formats a string to a valid NetBIOS name.

File

tcpip_helpers.h

C

```
void TCPIP_Helper_FormatNetBIOSName(uint8_t Name[]);
```

Returns

None.

Description

This function formats a string to a valid NetBIOS name. Names will be exactly 16 characters, as defined by the NetBIOS spec. The 16th character will be a 0x00 byte, while the other 15 will be the provided string, padded with spaces as necessary.

Preconditions

None.

Parameters

Parameters	Description
Name	the string to format as a NetBIOS name. This parameter must have at least 16 bytes allocated.

Function

void TCPIP_Helper_FormatNetBIOSName(uint8_t Name[])

TCPIP_Helper_htonl Function

Conversion routines from network order to host order and reverse.

File

tcpip_helpers.h

C

```
inline uint32_t TCPIP_Helper_htonl(uint32_t hLong);
```

Returns

The converted 16/32-bit quantity.

Description

These functions will convert a long or short quantity from the network order (big endian) to host order (little endian on PIC32).

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
hLong/nLong	long value (32 bits) to convert
hShort/nShort	short value (16 bits) to convert

Function

uint64_t TCPIP_Helper_htonll(uint64_t hLLong);
uint32_t TCPIP_Helper_htonl(uint32_t hLong);
uint16_t TCPIP_Helper_htons(uint16_t hShort);
uint64_t TCPIP_Helper_ntohll(uint64_t nLLong);
uint32_t TCPIP_Helper_ntohl(uint32_t nLong);
uint16_t TCPIP_Helper_ntohs(uint16_t nShort);

TCPIP_Helper_htons Function

File

tcpip_helpers.h

C

```
inline uint16_t TCPIP_Helper_htons(uint16_t hShort);
```

Description

This is function TCPIP_Helper_htons.

TCPIP_Helper_IPAddressToString Function

Converts an IPV4 address to an ASCII string.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_IPAddressToString(const IPV4_ADDR* IPAddress, char* buff, size_t buffSize);
```

Returns

- true - an IP address was successfully converted
- false - supplied buffer was not large enough

Description

This function converts IPV4_ADDR to a dotted-quad decimal IP address string

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
IPAddress	Pointer to IPV4_ADDR to convert
buff	buffer to store the converted dotted-quad IP address string
buffSize	buffer size

Function

bool TCPIP_Helper_IPAddressToString(const IPV4_ADDR* IPAddress, char* buff, size_t buffSize);

TCPIP_Helper_IPv6AddressToString Function

Converts an IPv6 address to a string representation.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_IPv6AddressToString(const IPV6_ADDR * addr, char* buff, size_t buffSize);
```

Returns

- true - an IPv6 address was successfully converted
- false - supplied buffer was not large enough

Description

This function converts an IPV6_ADDR to a text representation of an IPv6 address.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
addr	Pointer to IPV6_ADDR to convert
buff	Pointer to a buffer to store the text representation
buffSize	buffer size

Function

bool TCPIP_Helper_IPv6AddressToString (const IPV6_ADDR * addr, char* buff, size_t buffSize);

TCPIP_Helper_IsBcastAddress Function

Checks if an IPv4 address is a broadcast address.

File

tcpip_helpers.h

C

```
__inline__ bool TCPIP_Helper_IsBcastAddress(const IPV4_ADDR* IPAddress);
```

Returns

- true - if the IPv4 address is a broadcast address
- false - if the IPv4 address is not a broadcast address

Description

This function verifies if the supplied IPv4 address is a broadcast address.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
IPAddress	Pointer to IPV4_ADDR to check

Function

bool TCPIP_Helper_IsBcastAddress(const IPV4_ADDR* IPAddress);

TCPIP_Helper_IsMcastAddress Function

Checks if an IPv4 address is a multicast address.

File

tcpip_helpers.h

C

```
__inline__ bool TCPIP_Helper_IsMcastAddress(const IPV4_ADDR* IPAddress);
```

Returns

- true - if the IPv4 address is a multicast address
- false - if the IPv4 address is not a multicast address

Description

This function verifies if the supplied IPv4 address is a multicast address.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
IPAddress	Pointer to IPV4_ADDR to check

Function

bool TCPIP_Helper_IsMcastAddress(const IPV4_ADDR* IPAddress)

TCPIP_Helper_IsPrivateAddress Function

Detects a private (non-routable) address.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_IsPrivateAddress(uint32_t ipv4Address);
```

Returns

- true - the IPv4 address is a private address
- false - the IPv4 address is a routable address

Description

This function checks if the passed in IPv4 address is a private or a routable address.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
ipv4Address	IPv4 address to check, network order

Function

bool TCPIP_Helper_IsPrivateAddress(uint32_t ipv4Address);

TCPIP_Helper_MACAddressToString Function

Converts a MAC address to a string.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_MACAddressToString(const TCPIP_MAC_ADDR* macAddr, char* buff, size_t buffSize);
```

Returns

- true - a MAC address was successfully decoded
- false - no MAC address could be found, or the format was incorrect

Description

This function will convert a MAC address to a string representation.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
macAddr	Pointer to address to convert
buff	buffer to store the string representation
buffSize	size of the buffer

Function

bool TCPIP_Helper_MACAddressToString(const TCPIP_MAC_ADDR* macAddr, char* buff, size_t buffSize);

TCPIP_Helper_ntohl Function

File

tcpip_helpers.h

C

```
uint32_t TCPIP_Helper_ntohl(uint32_t nLong);
```

Description

!defined(__PIC32MX__)

TCPIP_Helper_ntohs Function

File

tcpip_helpers.h

C

```
uint16_t TCPIP_Helper_ntohs(uint16_t nShort);
```

Description

This is function TCPIP_Helper_ntohs.

TCPIP_Helper_StringToIPAddress Function

Converts an ASCII string to an IPV4 address.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_StringToIPAddress(const char* str, IPV4_ADDR* IPAddress);
```

Returns

- true - an IP address was successfully decoded
- false - no IP address could be found, or the format was incorrect

Description

This function parses a dotted-quad decimal IP address string into an IPV4_ADDR struct. The output result is big-endian.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
str	Pointer to a dotted-quad IP address string
IPAddress	Pointer to IPV4_ADDR in which to store the result

Function

bool TCPIP_Helper_StringToIPAddress(const char* str, IPV4_ADDR* IPAddress);

TCPIP_Helper_StringToIPv6Address Function

Converts a string to an IPv6 address.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_StringToIPv6Address(const char * str, IPV6_ADDR * addr);
```

Description

This function parses the text representation of an IPv6 address to an IPV6_ADDR struct. The output result is big-endian.

Remarks

None.

Preconditions

None

Example

1111:2222:3333:4444:5555:6666:AAAA:FFFF 1111:2222::FFFF 1111:2222:3333:4444:5555:6666:192.168.1.20 addr - Pointer to IPV6_ADDR in which to store the result

Parameters

Parameters	Description
str	Pointer to an RFC3513, Section 2.2 text representation of an IPv6 address

Function

bool TCPIP_Helper_StringToIPv6Address (const char * str, IPV6_ADDR * addr);

TCPIP_Helper_StringToMACAddress Function

Converts a string to an MAC address.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_StringToMACAddress(const char* str, uint8_t macAddr[6]);
```

Returns

- true - a MAC address was successfully decoded
- false - no MAC address could be found, or the format was incorrect

Description

This function parses a MAC address string "aa:bb:cc:dd:ee:ff" or "aa-bb-cc-dd-ee-ff" into an hex MAC address.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
str	Pointer to a colon separated MAC address string
macAddr	Pointer to buffer to store the result

Function

bool TCPIP_Helper_StringToMACAddress(const char* str, uint8_t macAddr[6]);

TCPIP_Helper_htonll Function

File

tcpip_helpers.h

C

```
uint64_t TCPIP_Helper_htonll(uint64_t hLLong);
```

Description

This is function TCPIP_Helper_htonll.

TCPIP_Helper_ntohll Function

File

tcpip_helpers.h

C

```
uint64_t TCPIP_Helper_ntohll(uint64_t nLLong);
```

Description

This is function TCPIP_Helper_ntohll.

TCPIP_Helper_SecurePortGetByIndex Function

Returns the secure port belonging to a specified index.

File

tcpip_helpers.h

C

```
uint16_t TCPIP_Helper_SecurePortGetByIndex(int index, bool streamSocket, int* pnIndexes);
```

Returns

- a port number corresponding to the required index
- 0 if the corresponding slot is free or the port is not associated with the requested stream/datagram flag.

Description

This function returns the secure TCP/UDP port situated at the internal port table requested index. It also returns the number of indexes currently in the table.

Preconditions

TCP/IP stack properly initialized.

Parameters

Parameters	Description
index	The port index to query. 0 should always be a valid index
streamSocket	if true, a stream/TCP port is queried else a datagram/UDP port is queried
pnIndexes	pointer to store the number of indexes that the table currntly has. Could be NULL if not needed.

Function

uint16_t TCPIP_Helper_SecurePortGetByIndex(int index, bool streamSocket, int* pnIndexes);

TCPIP_Helper_SecurePortSet Function

Sets the required port secure connection status.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_SecurePortSet(uint16_t port, bool streamSocket, bool isSecure);
```

Returns

- true - the port status successfuly changed
- false - the port status could not be changed (no more slots in the table, port not found, etc.)

Description

This function sets the internal data for deciding if the required port is a secure port or not. A secure port is one that requires secure connections. The TCP/IP internally maintained table with the ports that require secure connections is manipulated with this function. This table can be queried using the TCPIP_Helper_TCPSecurePortGet() function.

Remarks

Currently there is no protection for multi-threaded accessing and modifying the entries in the secure port table. It is thus recommended that the updates to the table occur just once at initialization time and after that all threads perform read-only accesses.

Preconditions

TCP/IP stack properly initialized.

Parameters

Parameters	Description
port	The TCP port to set
streamSocket	if true, a stream/TCP port is queried else a datagram/UDP port is queried
isSecure	if true, the port is set as requiring secure connection if false, the port is set as not requiring a secure connection and it will be removed from the secure ports table

Function

bool TCPIP_Helper_SecurePortSet(uint16_t port, bool streamSocket, bool isSecure);

TCPIP_Helper_TCPSecurePortGet Function

Checks if the required TCP port is a secure port.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_TCPSecurePortGet(uint16_t tcpPort);
```

Returns

- true - if the port is a secure connection port
- false - if port is not secure (the port is not found in the internal secure ports table)

Description

This function checks the internal data for detecting if the required TCP port is a secure port or not. A secure port is one that requires secure connections. The TCP/IP stack maintains an internal table with the ports that require secure connections. This table can be maintained using the TCPIP_Helper_SecurePortSet() function.

Remarks

The TCP/IP stack populates the internal secure port table with default values as part of the stack power up procedure. The table is not re-initialized when the stack is brought down and then restarted.

Preconditions

TCP/IP stack properly initialized.

Parameters

Parameters	Description
tcpPort	The TCP port to query

Function

bool TCPIP_Helper_TCPSecurePortGet(uint16_t tcpPort);

TCPIP_Helper_UDPSecurePortGet Function

Checks if the required UDP port is a secure port.

File

tcpip_helpers.h

C

```
bool TCPIP_Helper_UDPSecurePortGet(uint16_t udpPort);
```

Returns

- true - if the port is a secure connection port
- false - if port is not secure (the port is not found in the internal secure ports table)

Description

This function checks the internal data for detecting if the required UDP port is a secure port or not. A secure port is one that requires secure connections. The TCP/IP stack maintains an internal table with the ports that require secure connections. This table can be maintained using the TCPIP_Helper_SecurePortSet() function.

Remarks

The TCP/IP stack populates the internal secure port table with default values as part of the stack power up procedure. The table is not re-initialized when the stack is brought down and then restarted.

Preconditions

TCP/IP stack properly initialized.

Parameters

Parameters	Description
udpPort	The UDP port to query

Function

bool TCPIP_Helper_UDPSecurePortGet(uint16_t udpPort);

TCPIP_Helper_Base64Decode Function

Helper function to decode a Base-64 encoded array.

File

tcpip_helpers.h

C

```
uint16_t TCPIP_Helper_Base64Decode(const uint8_t* sourceData, uint16_t sourceLen, uint8_t* destData,
uint16_t destLen);
```

Returns

Number of decoded bytes written to destData.

Description

Decodes a Base-64 array to its literal representation.

Remarks

This function will ignore invalid Base-64 characters (CR, LF, etc). If sourceData is equal to destData, the data will be converted in-place. If sourceData is not equal to destData, but the regions overlap, the behavior is undefined.

Decoded data size is 3 / 4 the size of the encoded source data.

Preconditions

None

Parameters

Parameters	Description
sourceData	Pointer to a string of Base-64 encoded data
sourceLen	Length of the Base-64 source data Maximum length that can be written to destData
destData	Pointer to write the decoded data

Function

uint16_t TCPIP_Helper_Base64Decode(const uint8_t* sourceData, uint16_t sourceLen,

uint8_t* destData, uint16_t destLen)

TCPIP_Helper_Base64Encode Function

Helper function to encode to Base-64.

File

tcpip_helpers.h

C

```
uint16_t TCPIP_Helper_Base64Encode(const uint8_t* sourceData, uint16_t sourceLen, uint8_t* destData,
uint16_t destLen);
```

Returns

Number of encoded bytes written to destData. This will always be a multiple of 4.

Description

This function encodes a binary array to Base-64.

Remarks

Encoding cannot be performed in-place. If surceData overlaps with destData, the behavior is undefined.

The source data is padded wit 1 or 2 bytes, if needed, to make the source size a multiple of 3 bytes. Then for each 3 bytes tuple in the source 4 output bytes are generated. The output size needed is pad(sourceLen) * 4 / 3 bytes.

Preconditions

None

Parameters

Parameters	Description
sourceData	Pointer to a string of binary data
sourceLen	Length of the binary source data
destData	Pointer to write the Base-64 encoded data
destLen	Maximum length that can be written to destData

Function

uint16_t TCPIP_Helper_Base64Encode(const uint8_t* sourceData, uint16_t sourceLen,

uint8_t* destData, uint16_t destLen)

b) Task and Initialize Functions

TCPIP_STACK_SetLocalMasks Function

Sets the local/non-local masks for network interface.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_SetLocalMasks(TCPIP_NET_HANDLE netH, uint32_t andMask, uint32_t orMask);
```

Returns

- true - if interface exists and is enabled
- false - if an error occurred

Description

This function sets the masks used in the stack decision of a destination address being a local or non-local address. These masks will be used when the corresponding mask type is set to TCPIP_LOCAL_MASK_SET.

Remarks

None

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_SetLocalMasks(netH, 0x0101a0c0, 0x0);
```

Parameters

Parameters	Description
netH	handle of the interface to use
andMask	AND mask to use for local network detection, big endian (BE) format
orType	OR mask to use for local network detection , BE format

Function

bool TCPIP_STACK_SetLocalMasks(TCPIP_NET_HANDLE netH,

uint32_t andMask, uint32_t orMask)

TCPIP_STACK_SetLocalMasksType Function

Sets the local/non-local masks type for network interface.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_SetLocalMasksType(TCPIP_NET_HANDLE netH, TCPIP_LOCAL_MASK_TYPE andType,
TCPIP_LOCAL_MASK_TYPE orType);
```

Returns

- true - if the interface exists and is enabled
- false - if an error occurred

Description

This function sets the types of masks used in the stack decision of a destination address being a local or non-local address. For example, when a TCP connection is made, the advertised MSS usually has different values for local versus non-local networks.

To decide if a destination IP address (destAdd) is local to a network interface (having the netAdd address) the following calculation is performed: if(((destAdd & andMask) | orMask) == ((netAdd & andMask) | orMask)), the destination address is considered to be a local address; otherwise, the destination address is non-local.

This function sets the types of masks used for the AND and OR operations, as follows:

- TCPIP_LOCAL_MASK_ZERO: use the all 0's mask (0x00000000)
- TCPIP_LOCAL_MASK_ONE: use the all 1's mask (0xffffffff)
- TCPIP_LOCAL_MASK_NET: use the current network mask
- TCPIP_LOCAL_MASK_SET: use a mask that's set by the application There are operations to set a specific AND and OR masks

Using different values for the AND and OR masks an application can select various destination networks to be considered as local/non-local:

- only their own network
- any network
- no network
- specific (range of) networks
- etc.

Remarks

The default value for both AND and OR masks is set to TCPIP_LOCAL_MASK_ZERO so that any destination network is considered to be local!

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_SetLocalMasksType(netH, TCPIP_LOCAL_MASK_NET, TCPIP_LOCAL_MASK_ZERO);
```

Parameters

Parameters	Description
netH	handle of the interface to use
andType	AND type of mask to use for local network detection
orType	OR type of mask to use for local network detection

Function

bool TCPIP_STACK_SetLocalMasksType(TCPIP_NET_HANDLE netH, TCPIP_LOCAL_MASK_TYPE andType,
TCPIP_LOCAL_MASK_TYPE orType)

TCPIP_STACK_HandlerDeregister Function

Deregisters an event notification handler.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_HandlerDeregister(TCPIP_EVENT_HANDLE hEvent);
```

Returns

- true - if the operation succeeded
- false - if the operation failed

Description

This function removes an event notification handler.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_EVENT_HANDLE myHandle = TCPIP_STACK_HandlerRegister( hNet, TCPIP_EV_CONN_ALL, myEventHandler, myParam
);
// do something else
// now we're done with it
TCPIP_STACK_HandlerDeregister(myHandle);
```

Parameters

Parameters	Description
hEvent	TCPIP event handle obtained by a call to TCPIP_STACK_HandlerRegister

Function

bool TCPIP_STACK_HandlerDeregister(TCPIP_EVENT_HANDLE hEvent);

TCPIP_STACK_HandlerRegister Function

Sets a new event notification handler.

File

tcpip_manager.h

C

```
TCPIP_EVENT_HANDLE TCPIP_STACK_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_EVENT evMask,
TCPIP_STACK_EVENT_HANDLER notifyHandler, const void* notifyfParam);
```

Returns

- a valid TCPIP_EVENT_HANDLE - if the operation succeeded
- NULL - if the operation failed

Description

This function sets a new event notification handler. The caller can use the handler to be notified of stack events.

Remarks

The notification handler may be called from the ISR which detects the corresponding event. The event notification handler has to be kept as short as possible and non-blocking.

Without a notification handler the stack user can still call TCPIP_STACK_GetPending() to see if processing by the stack needed.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_EVENT_HANDLE myHandle = TCPIP_STACK_HandlerRegister( hNet, TCPIP_EV_CONN_ALL, myEventHandler, myParam
);
```

Parameters

Parameters	Description
hNet	network handle
evMask	mask of events to be notified of
notifyHandler	the event notification handler
notifyfParam	notification handler parameter

Function

TCPIP_EVENT_HANDLE TCPIP_STACK_HandlerRegister(TCPIP_NET_HANDLE hNet,
TCPIP_EVENT evMask, TCPIP_STACK_EVENT_HANDLER notifyHandler, const void* notifyfParam)

TCPIP_STACK_Initialize Function

Stack initialization function.

File

tcpip_manager.h

C

```
SYS_MODULE_OBJ TCPIP_STACK_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

- Valid handle to a driver object, if successful.
- SYS_MODULE_OBJ_INVALID if initialization failed.

Description

The function initializes the stack. If an error occurs, the SYS_ERROR() is called and the function deinitialize itself.

Remarks

This function must be called before any of the stack or its component routines are used.

New TCPIP_NETWORK_CONFIG types could be added/removed at run time for implementations that support dynamic network interface creation.

Only one instance of the TCP/IP stack can run in the system and the index parameter is irrelevant

If this call is made after the stack is successfully initialized, the current TCP/IP stack object handle will be returned.

Preconditions

None

Parameters

Parameters	Description
index	index of the TCP/IP stack to be initialized
init	Pointer to initialization data.
It should be a TCPIP_STACK_INIT structure carrying the following data	• pNetConf - pointer to an array of TCPIP_NETWORK_CONFIG to support • nNets - number of network configurations in the array • pModConfig - pointer to an array of TCPIP_STACK_MODULE_CONFIG • nModules - number of modules to initialize

Function

SYS_MODULE_OBJ TCPIP_STACK_Initialize(const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init);

TCPIP_STACK_Deinitialize Function

Stack deinitialization function.

File

tcpip_manager.h

C

```
void TCPIP_STACK_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function performs the deinitialization of the TCPIP stack. All allocated resources are released.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize.

Parameters

Parameters	Description
object	Object handle, returned from TCPIP_STACK_Initialize call

Function

void TCPIP_STACK_Deinitialize(SYS_MODULE_OBJ object)

TCPIP_STACK_Task Function

TCP/IP Stack task function.

File

tcpip_manager.h

C

```
void TCPIP_STACK_Task(SYS_MODULE_OBJ object);
```

Returns

None.

Description

TCP/IP stack execution state machine. Stack Finite-state Machine (FSM) is executed.

Remarks

This FSM checks for new incoming packets, and routes it to appropriate stack components. It also performs timed operations.

This function must be called periodically to ensure timely responses.

This function continues the stack initialization process after the TCPIP_STACK_Initialize was called. The TCPIP_STACK_Status will report SYS_STATUS_BUSY while the initialization is in progress.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize.

Parameters

Parameters	Description
object	Object handle returned by TCPIP_STACK_Initialize

Function

void TCPIP_STACK_Task(SYS_MODULE_OBJ object)

TCPIP_STACK_VersionGet Function

Gets the TCP/IP stack version in numerical format.

File

tcpip_manager.h

C

```
unsigned int TCPIP_STACK_VersionGet(const SYS_MODULE_INDEX index);
```

Returns

Current driver version in numerical format.

Description

This function gets the TCP/IP stack version. The version is encoded as major * 10000 + minor * 100 + patch. The stringed version can be obtained using TCPIP_STACK_VersionStrGet()

Remarks

None.

Preconditions

None.

Example

```
unsigned int version;

version = TCPIP_STACK_VersionGet( TCPIP_STACK_INDEX_1 );

if(version < 110200)
{
    // Do Something
}
```

Parameters

Parameters	Description
index	Identifier for the object instance to get the version for

Function

unsigned int TCPIP_STACK_VersionGet (const SYS_MODULE_INDEX index)

TCPIP_STACK_VersionStrGet Function

Gets the TCP/IP stack version in string format.

File

tcpip_manager.h

C

```
char * TCPIP_STACK_VersionStrGet(const SYS_MODULE_INDEX index);
```

Returns

Current TCP/IP stack version in the string format.

Description

This function gets the TCP/IP stack version. The version is returned as major.minor.path[type], where type is optional. The numerical version can be obtained using TCPIP_STACK_VersionGet.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
index	Identifier for the object instance to get the version for.

Function

char * TCPIP_STACK_VersionStrGet (const SYS_MODULE_INDEX index)

TCPIP_STACK_MACObjectGet Function

Returns the network MAC driver object of this interface.

File

tcpip_manager.h

C

```
const TCPIP_MAC_OBJECT* TCPIP_STACK_MACObjectGet(TCPIP_NET_HANDLE netH);
```

Returns

It returns a valid MAC driver object pointer if success. Returns 0 if no such interface or there is no MAC object.

Description

This function returns the MAC driver object that's associated with the interface handle.

Remarks

The MAC driver object is the one that's passed at the stack/interface initialization.

The MAC driver is not a true multi-client driver. Under normal circumstances the MAC driver has only one client, the TCP/IP stack. To have the TCPIP_MAC_Open() succeed after the MAC driver has been initialized by the TCP/IP stack, the configuration symbol DRV_ETHMAC_CLIENTS_NUMBER has to be > 1! But the returned handle is the same one as the TCP/IP stack uses.

Access to the MAC driver in this way is allowed mainly for debugging, diagnostic and statistics purposes. It is possible though to transmit packets this way. But it's not possible to be signalled about incoming RX packets while the stack is running because only the TCP/IP dispatcher will be notified by the RX events.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
 TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("eth0");
 const TCPIP_MAC_OBJECT* pEthMacObject = TCPIP_STACK_MACObjectGet(netH);
 if(pEthMacObject != 0)
 {    // valid MAC object pointer
      DRV_HANDLE hMac = (*pEthMacObject->TCPIP_MAC_Open)(pEthMacObject->macId, DRV_IO_INTENT_READWRITE);
      if(hMac != DRV_HANDLE_INVALID)
      {   // can use the MAC handle to access a MAC function
          TCPIP_MAC_RX_STATISTICS rxStatistics;
          TCPIP_MAC_TX_STATISTICS txStatistics;
          TCPIP_MAC_RES macRes = (*pEthMacObject->TCPIP_MAC_StatisticsGet)(hMac, &rxStatistics,
&txStatistics);
      }
 }
```

Parameters

Parameters	Description
netH	Interface handle to get the name of.

Function

const TCPIP_MAC_OBJECT* TCPIP_STACK_MACObjectGet(TCPIP_NET_HANDLE netH);

TCPIP_MODULE_SignalFunctionDeregister Function

Deregisters a signal function for a stack module.

File

tcpip_manager.h

C

```
bool TCPIP_MODULE_SignalFunctionDeregister(TCPIP_MODULE_SIGNAL_HANDLE signalHandle);
```

Returns

- true - if the operation succeeded
- false - if the operation failed (i.e., no such module, invalid handle, etc.)

Description

This function deregisters a previous signal function. The caller will no longer be notified of stack internal signals.

Remarks

See the remarks for TCPIP_MODULE_SignalFunctionRegister.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.
signalHandle - a valid signal handle

Example

```
void appSignalFunc(TCPIP_STACK_MODULE moduleId, TCPIP_MODULE_SIGNAL signal)
{
    // process incoming signal for the incoming module
}

TCPIP_MODULE_SIGNAL_HANDLE signalH = TCPIP_MODULE_SignalFunctionRegister( TCPIP_MODULE_HTTP_SERVER,
appSignalFunc);

// when done with this signal notification
TCPIP_MODULE_SignalFunctionDeregister(signalH);
```

Parameters

Parameters	Description
signalHandle	signal handle obtained from a registration call

Function

bool TCPIP_MODULE_SignalFunctionDeregister(TCPIP_MODULE_SIGNAL_HANDLE signalHandle);

TCPIP_MODULE_SignalFunctionRegister Function

Registers a new signal function for a stack module.

File

tcpip_manager.h

C

```
TCPIP_MODULE_SIGNAL_HANDLE TCPIP_MODULE_SignalFunctionRegister(TCPIP_STACK_MODULE moduleId,
TCPIP_MODULE_SIGNAL_FUNC signalF);
```

Returns

- valid handle - if the operation succeeded,
- 0/invalid handle - if the operation failed (i.e., no such module, signal already set, etc.)

Description

This function registers a new signal function. This function will be called and the user notified when a stack internal signal occurs.

Remarks

There is no support for multiple signal functions now. Each module supports just one signal function. A call to register a new module signal function will fail if a function is already registered. TCPIP_MODULE_SignalFunctionDeregister needs to be called first

By default all stack modules, including the stack manager (TCPIP_MODULE_MANAGER) are initialized with a null signal function. Explicit call is needed for setting a module signal function.

A signal handler can be registered for the stack manager itself (TCPIP_MODULE_MANAGER). This will report RX and TMO signals.

The stack internal signaling mechanism is always active and cannot be disabled. This function is called on top of the normal stack signaling mechanism for a module that has a registered signal function.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
void appSignalFunc(TCPIP_STACK_MODULE moduleId, TCPIP_MODULE_SIGNAL signal)
{
    // process incoming signal for the incoming module
}

TCPIP_MODULE_SIGNAL_HANDLE signalH = TCPIP_MODULE_SignalFunctionRegister( TCPIP_MODULE_HTTP_SERVER,
appSignalFunc);
```

Parameters

Parameters	Description
moduleId	module ID
signalF	signal function to be called when an internal signal occurs

Function

TCPIP_MODULE_SIGNAL_HANDLE TCPIP_MODULE_SignalFunctionRegister(TCPIP_STACK_MODULE moduleId,
TCPIP_MODULE_SIGNAL_FUNC signalF);

TCPIP_MODULE_SignalGet Function

Returns the current signals for a TCP/IP module.

File

tcpip_manager.h

C

```
TCPIP_MODULE_SIGNAL TCPIP_MODULE_SignalGet(TCPIP_STACK_MODULE moduleId);
```

Returns

The current value of the module's signals.

Description

This function provides a read only value of the current signals for a stack module.

Remarks

The signals are processed and cleared inside the stack. It is not possible to clear a signal that's pending using this API.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_MODULE_SIGNAL currSIgnal = TCPIP_MODULE_SignalGet(TCPIP_MODULE_HTTP_SERVER);
```

Parameters

Parameters	Description
moduleId	module ID

Function

TCPIP_MODULE_SIGNAL TCPIP_MODULE_SignalGet(TCPIP_STACK_MODULE moduleId);

c) Network Status and Control Functions

TCPIP_STACK_IndexToNet Function

Network interface handle from interface number.

File

tcpip_manager.h

C

```
TCPIP_NET_HANDLE TCPIP_STACK_IndexToNet(int netIx);
```

Returns

Network interface handle.

Description

This function converts an interface number to a network interface handle.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Parameters

Parameters	Description
netIx	Interface index.

Function

TCPIP_NET_HANDLE TCPIP_STACK_IndexToNet(int netIx)

TCPIP_STACK_NetBIOSName Function

Network interface NetBIOS name.

File

tcpip_manager.h

C

```
const char* TCPIP_STACK_NetBIOSName(TCPIP_NET_HANDLE netH);
```

Returns

pointer to the NetBIOS name of that interface

Description

The function returns the network interface NetBIOS name.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
const char* biosName = TCPIP_STACK_NetBIOSName(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get name of.

Function

const char* TCPIP_STACK_NetBIOSName(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetBiosNameSet Function

Sets network interface NetBIOS name.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetBiosNameSet(TCPIP_NET_HANDLE netH, const char* biosName);
```

Returns

- true - if the NetBIOS name of the interface is set
- false - if no such interface or interface is not enabled

Description

This function sets the network interface NetBIOS name.

Remarks

Exercise extreme caution when using these functions to change the settings of a running network interface. Changing these parameters at runtime can lead to unexpected behavior or loss of network connectivity. The preferred way to change the parameters for a running interface is to do so as part of the network configuration passed at the stack initialization.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_NetBiosNameSet(netH, myBiosName);
```

Parameters

Parameters	Description
netH	Interface handle to set name of.

Function

bool TCPIP_STACK_NetBiosNameSet(TCPIP_NET_HANDLE netH, const char* biosName);

TCPIP_STACK_NetDefaultGet Function

Default network interface handle.

File

tcpip_manager.h

C

```
TCPIP_NET_HANDLE TCPIP_STACK_NetDefaultGet();
```

Returns

The default network interface.

Description

The function returns the current default network interface in the TCP/IP stack. This is the interface on which packets will be transmitted when the internal routing algorithm cannot detect a match for an outgoing packet.

Remarks

This function is intended for multi-homed hosts, with the TCP/IP stack running multiple interfaces.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Function

TCPIP_NET_HANDLE TCPIP_STACK_NetDefaultGet(void)

TCPIP_STACK_NetDefaultSet Function

Sets the default network interface handle.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetDefaultSet(TCPIP_NET_HANDLE netH);
```

Returns

- true - if success
- false - if failed (the old interface does not change)

Description

The function sets the current default network interface in the TCP/IP stack.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Parameters

Parameters	Description
netH	Interface handle.

Function

bool TCPIP_STACK_NetDefaultSet(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetHandleGet Function

Network interface handle from a name.

File

tcpip_manager.h

C

```
TCPIP_NET_HANDLE TCPIP_STACK_NetHandleGet(const char* interface);
```

Returns

Network handle.

Description

This function resolves a network interface name to a handle.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("PIC32INT");
 or
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("eth0");
```

Parameters

Parameters	Description
interface	The names specified in TCPIP_NETWORK_CONFIG.

Function

TCPIP_NET_HANDLE TCPIP_STACK_NetHandleGet(const char* interface)

TCPIP_STACK_NetIndexGet Function

Network interface number from interface handle.

File

tcpip_manager.h

C

```
int TCPIP_STACK_NetIndexGet(TCPIP_NET_HANDLE hNet);
```

Returns

Index of this network handle in the stack. -1 if invalid.

Description

This function converts a network interface handle to an interface number.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Parameters

Parameters	Description
hNet	Interface handle.

Function

int TCPIP_STACK_NetIndexGet(TCPIP_NET_HANDLE hNet);

TCPIP_STACK_NetMask Function

Network interface IPv4 address mask.

File

tcpip_manager.h

C

```
uint32_t TCPIP_STACK_NetMask(TCPIP_NET_HANDLE netH);
```

Returns

- IPv4 address mask of that interface
- 0 if interface is disabled/non-existent

Description

The function returns the IPv4 address mask of the specified interface. If the interface is enabled then it returns the IP address mask of that interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
uint32_t subMask = TCPIP_STACK_NetMask(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get mask of.

Function

uint32_t TCPIP_STACK_NetMask(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetNameGet Function

Network interface name from a handle.

File

tcpip_manager.h

C

```
const char* TCPIP_STACK_NetNameGet(TCPIP_NET_HANDLE netH);
```

Returns

It returns the name associated to that interface handle. Returns 0 if no such name.

Description

This function returns the name associated with the interface handle.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_IndexToNet(0);
const char* netName = TCPIP_STACK_NetNameGet(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get the name of.

Function

const char* TCPIP_STACK_NetNameGet(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NumberOfNetworksGet Function

Number of network interfaces in the stack.

File

tcpip_manager.h

C

```
int TCPIP_STACK_NumberOfNetworksGet();
```

Returns

Number of network interfaces.

Description

This function returns the number of interfaces currently active in the TCP/IP stack.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Function

int TCPIP_STACK_NumberOfNetworksGet(void)

TCPIP_STACK_EventsPendingGet Function

Returns the currently pending events.

File

tcpip_manager.h

C

```
TCPIP_EVENT TCPIP_STACK_EventsPendingGet(TCPIP_NET_HANDLE hNet);
```

Returns

The currently TCPIP pending events.

Description

This function returns the currently pending events Multiple events can be ORed together as they accumulate.

Remarks

This is the preferred method to get the current pending stack events.

Even with a notification handler in place it's better to use this function to get the current pending events

The returned value is just a momentary value. The pending events can change any time.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_EVENT currEvents = TCPIP_STACK_EventsPendingGet( hNet);
```

Parameters

Parameters	Description
hNet	network handle

Function

TCPIP_EVENT TCPIP_STACK_EventsPendingGet(TCPIP_NET_HANDLE hNet)

TCPIP_STACK_NetIsReady Function

Gets the network interface configuration status.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetIsReady(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if interface exists, the corresponding MAC is linked and the interface is properly configured and ready for transfers
- false - if the interface does not exist or interface not ready

Description

This function returns the network interface configuration status.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Parameters

Parameters	Description
hNet	Interface handle.

Function

bool TCPIP_STACK_NetIsReady(TCPIP_NET_HANDLE hNet);

TCPIP_STACK_NetMACRegisterStatisticsGet Function

Get the MAC statistics register data.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetMACRegisterStatisticsGet(TCPIP_NET_HANDLE netH, TCPIP_MAC_STATISTICS_REG_ENTRY*
pRegEntries, int nEntries, int* pHwEntries);
```

Returns

- true - if the call succeeded
- false - if the call failed (the corresponding MAC does not implement hardware statistics counters).

Description

This function returns the hardware statistics register data of the MAC that is attached to the specified network interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_MAC_REG_STATISTICS regStatistics;
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
if(TCPIP_STACK_NetMACRegisterStatisticsGet(netH, &regStatistics))
```

```
{
    // display the hardware statistics registers for the internal PIC32 MAC attached to this interface
}
```

Parameters

Parameters	Description
netH	handle of the interface to use
pRegStatistics	pointer to a pRegEntries that will receive the current hardware statistics registers values. Can be 0, if only the number of supported hardware registers is requested
nEntries	provides the number of TCPIP_MAC_STATISTICS_REG_ENTRY structures present in pRegEntries Can be 0, if only the number of supported hardware registers is requested The register entries structures will be filled by the driver, up to the supported hardware registers.
pHwEntries	pointer to an address to store the number of the statistics registers that the hardware supports It is updated by the driver. Can be 0 if not needed

Function

bool TCPIP_STACK_NetMACRegisterStatisticsGet(TCPIP_NET_HANDLE netH,
TCPIP_MAC_STATISTICS_REG_ENTRY* pRegEntries, int nEntries, int* pHwEntries);

TCPIP_STACK_NetAliasNameGet Function

Network interface alias name from a handle.

File

tcpip_manager.h

C

```
int TCPIP_STACK_NetAliasNameGet(TCPIP_NET_HANDLE netH, char* nameBuffer, int buffSize);
```

Returns

It returns the number of characters of the interface alias name. Returns 0 if no such interface.

Description

This function returns the alias name associated with the interface handle.

Remarks

The aliases names are:

- "eth0", "eth1", etc. for Ethernet interfaces
- "wlan0", "wlan1", etc. for Wi-Fi interfaces

See the TCPIP_STACK_IF_NAME_ALIAS_ETH, TCPIP_STACK_IF_NAME_ALIAS_WLAN in tcpip.h for the aliases names.

Alias interface names are at most 8 characters long.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_IndexToNet(0);
char ifName[8];
TCPIP_STACK_NetAliasNameGet(netH, ifName, sizeof(ifName));
```

Parameters

Parameters	Description
netH	Interface handle name to obtain
nameBuffer	buffer to receive the name; could be 0 if only name size is needed
buffSize	size of the provided buffer

Function

int TCPIP_STACK_NetAliasNameGet(TCPIP_NET_HANDLE netH,
char* nameBuffer, int buffSize)

TCPIP_STACK_InitializeDataGet Function

Get the TCP/IP stack initialization data.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_InitializeDataGet(SYS_MODULE_OBJ object, TCPIP_STACK_INIT* pStackInit);
```

Returns

- true - if the stack is up and running and the stack init data has been copied
- false - if the stack is down or an error has occurred

Description

This function returns the data that was used for the last call to the TCPIP_STACK_Initialize().

Remarks

The stack does not make a persistent copy of the TCPIP_NETWORK_CONFIG and TCPIP_STACK_MODULE_CONFIG data that are passed in the TCPIP_STACK_INIT at the moment of the stack initialization! It is up to the application to insure that the initialization data is still available after the TCPIP_STACK_Initialize() has been called if this API is used.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
TCPIP_STACK_INIT initData;
TCPIP_STACK_InitializeDataGet(tcpipObject, &initData);
```

Parameters

Parameters	Description
object	Object handle, returned from TCPIP_STACK_Initialize call
pStackInit	Pointer to an address to store the stack initialization data

Function

bool TCPIP_STACK_InitializeDataGet(SYS_MODULE_OBJ object,
TCPIP_STACK_INIT* pStackInit)

d) Network Up/Down/Linked Functions

TCPIP_STACK_NetIsLinked Function

Gets the network interface link status.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetIsLinked(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if interface exists and the corresponding MAC is linked
- false - if the interface does not exist or the link is down

Description

This function returns the network interface link status.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Parameters

Parameters	Description
hNet	Interface handle.

Function

bool TCPIP_STACK_NetIsLinked(TCPIP_NET_HANDLE hNet);

TCPIP_STACK_NetIsUp Function

Gets the network interface up or down status.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetIsUp(TCPIP_NET_HANDLE hNet);
```

Returns

- true - if interface exists and is enabled
- false - if the interface does not exist or is disabled

Description

This function returns the network interface up or down (enabled) status.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Parameters

Parameters	Description
hNet	Interface handle.

Function

bool TCPIP_STACK_NetIsUp(TCPIP_NET_HANDLE hNet);

TCPIP_STACK_NetUp Function

Turns up a network interface. As part of this process, the corresponding MAC driver is initialized.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetUp(TCPIP_NET_HANDLE netH, const TCPIP_NETWORK_CONFIG* pUsrConfig);
```

Returns

- true - if success
- false - if no such network or an error occurred

Description

This function brings a network interface up and perform its initialization.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Parameters

Parameters	Description
netH	Interface handle.
pUsrConfig	pointer to a TCPIP_NETWORK_CONFIG for the interface initialization

Function

bool TCPIP_STACK_NetUp(TCPIP_NET_HANDLE netH, const TCPIP_NETWORK_CONFIG*
pUsrConfig)

TCPIP_STACK_NetDown Function

Turns down a network interface.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetDown(TCPIP_NET_HANDLE netH);
```

Returns

- true - if success
- false - if no such network interface or the interface is already down

Description

This function performs the deinitialization of a net interface. As part of this process, the corresponding MAC driver is deinitialized.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Parameters

Parameters	Description
netH	Interface handle.

Function

bool TCPIP_STACK_NetDown(TCPIP_NET_HANDLE netH)

e) Network Address Status and Control Functions

TCPIP_STACK_NetAddress Function

Network interface IPv4 address.

File

tcpip_manager.h

C

```
uint32_t TCPIP_STACK_NetAddress(TCPIP_NET_HANDLE netH);
```

Returns

If interface is enabled, it returns the IP address of that interface; otherwise 0 is returned.

Description

If interface is enabled then the function will return the IPv4 address of the network interface.

Remarks

None

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("eth0");
uint32_t ipAdd = TCPIP_STACK_NetAddress(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get address of.

Function

uint32_t TCPIP_STACK_NetAddress(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetAddressBcast Function

Network interface broadcast address.

File

tcpip_manager.h

C

```
uint32_t TCPIP_STACK_NetAddressBcast(TCPIP_NET_HANDLE netH);
```

Returns

- Broadcast IP address of that interface
- 0 if no such interface

Description

The function returns the network interface broadcast address. The interface should be enabled for this function to work.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Parameters

Parameters	Description
netH	Interface handle to get address of.

Function

uint32_t TCPIP_STACK_NetAddressBcast(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetAddressDnsPrimary Function

Network interface DNS address.

File

tcpip_manager.h

C

```
uint32_t TCPIP_STACK_NetAddressDnsPrimary(TCPIP_NET_HANDLE netH);
```

Returns

- IPv4 address of the primary DNS server
- 0 if no such interface or interface is down

Description

If interface is enabled then the function will return the IPv4 address of the primary DNS of the network interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
myIPAddress = TCPIP_STACK_NetAddressDnsPrimary(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get the DNS address of.

Function

uint32_t TCPIP_STACK_NetAddressDnsPrimary(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetAddressDnsPrimarySet Function

Sets network interface IPv4 DNS address.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetAddressDnsPrimarySet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress);
```

Returns

- true - if success
- false - if no such interface or interface is not enabled

Description

This function sets the network interface primary IPv4 DNS address.

Remarks

Exercise extreme caution when using these functions to change the settings of a running network interface. Changing these parameters at runtime can lead to unexpected behavior or loss of network connectivity. The preferred way to change the parameters for a running interface is to do so as part of the network configuration passed at the stack initialization.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The

network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_NetAddressDnsPrimarySet(netH, &myIPAddress);
```

Parameters

Parameters	Description
netH	Interface handle to set the DNS address of.
ipAddress	IP address to set

Function

bool TCPIP_STACK_NetAddressDnsPrimarySet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress)

TCPIP_STACK_NetAddressDnsSecond Function

Network interface secondary DNS address.

File

tcpip_manager.h

C

```
uint32_t TCPIP_STACK_NetAddressDnsSecond(TCPIP_NET_HANDLE netH);
```

Returns

- The secondary DNS address if success.
- 0 if no such interface or interface is down

Description

If interface is enabled then the function will return the IPv4 address of the secondary DNS of the network interface.

Remarks

None

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
myIPAddress = TCPIP_STACK_NetAddressDnsSecond(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get the DNS address of.

Function

uint32_t TCPIP_STACK_NetAddressDnsSecond(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetAddressDnsSecondSet Function

Sets network interface IPv4 secondary DNS address.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetAddressDnsSecondSet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress);
```

Returns

- true - if success
- false - if no such interface or interface is not enabled

Description

This function sets the network interface secondary IPv4 DNS address.

Remarks

Exercise extreme caution when using these functions to change the settings of a running network interface. Changing these parameters at runtime can lead to unexpected behavior or loss of network connectivity. The preferred way to change the parameters for a running interface is to do so as part of the network configuration passed at the stack initialization.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_NetAddressDnsSecondSet(netH, &myIPAddress);
```

Parameters

Parameters	Description
netH	Interface handle to set the secondary DNS address of.
ipAddress	IP address to set

Function

bool TCPIP_STACK_NetAddressDnsSecondSet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress)

TCPIP_STACK_NetAddressGateway Function

Network interface IPv4 gateway address.

File

tcpip_manager.h

C

```
uint32_t TCPIP_STACK_NetAddressGateway(TCPIP_NET_HANDLE netH);
```

Returns

If interface is enabled then it returns the gateway address of that interface. Otherwise, 0 is returned.

Description

If interface is enabled then the function will return the IPv4 gateway address of the network interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("eth0");
uint32_t ipAdd = TCPIP_STACK_NetAddressGateway(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get address of.

Function

uint32_t TCPIP_STACK_NetAddressGateway(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetAddressGatewaySet Function

Sets network interface IPv4 gateway address.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetAddressGatewaySet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress);
```

Returns

- true - if success
- false - if no such interface or interface is not enabled

Description

This function sets the network interface IPv4 gateway address.

Remarks

Exercise extreme caution when using these functions to change the settings of a running network interface. Changing these parameters at runtime can lead to unexpected behavior or loss of network connectivity. The preferred way to change the parameters for a running interface is to do so as part of the network configuration passed at the stack initialization.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_NetAddressGatewaySet(netH, &myIPAddress);
```

Parameters

Parameters	Description
netH	Interface handle to set the gateway address of.
ipAddress	IP address to set

Function

bool TCPIP_STACK_NetAddressGatewaySet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress)

TCPIP_STACK_NetAddressMac Function

Network interface MAC address.

File

tcpip_manager.h

C

```
const uint8_t* TCPIP_STACK_NetAddressMac(TCPIP_NET_HANDLE netH);
```

Returns

- Constant pointer to the MAC address
- 0 if no such interface

Description

The function returns the network interface physical (MAC) address. The interface should be enabled for this function to work.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
const TCPIP_MAC_ADDR* pAdd = TCPIP_STACK_NetAddressMac(netH);
```

Parameters

Parameters	Description
netH	Interface handle to get the address of.

Function

const uint8_t* TCPIP_STACK_NetAddressMac(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetAddressMacSet Function

Sets network interface MAC address.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetAddressMacSet(TCPIP_NET_HANDLE netH, const TCPIP_MAC_ADDR* pAddr);
```

Returns

- true - if the MAC address of the interface is set
- false - if no such interface or interface is not enabled

Description

This function sets the network interface physical MAC address.

Remarks

One should use extreme caution when using these functions to change the settings of a running network interface. Changing these parameters at runtime can lead to unexpected behavior or loss of network connectivity. The preferred way to change the parameters for a running interface is to do so as part of the network configuration passed at the stack initialization.

This function updates the MAC address in the stack data structures. It does not re-program the MAC with the new address. The MAC programming is done only at MAC initialization for now.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_NetAddressMacSet(netH, &myMacAddress);
```

Parameters

Parameters	Description
netH	Interface handle to set the address of.
pAddr	pointer to a valid physical (MAC) address.

Function

bool TCPIP_STACK_NetAddressMacSet(TCPIP_NET_HANDLE netH, const TCPIP_MAC_ADDR* pAddr)

TCPIP_STACK_NetAddressSet Function

Sets network interface IPv4 address.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetAddressSet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress, IPV4_ADDR* mask, bool
setDefault);
```

Returns

- true - if success
- false - if no such interface or interface is not enabled

Description

This function sets the associated network IP address and/or mask. If you're changing the network then it's preferred that you set both these values simultaneously to avoid having the stack running with a mismatch between its IP address and mask.

Remarks

Exercise extreme caution when using these functions to change the settings of a running network interface. Changing these parameters at runtime can lead to unexpected behavior or loss of network connectivity. The preferred way to change the parameters for a running interface is to do so as part of the network configuration passed at the stack initialization.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_NetAddressSet(netH, &myIPAddress, &myIPMask, false);
```

Parameters

Parameters	Description
netH	Interface handle to set address of.
ipAddress	IP address to set (could be NULL to set only the mask)
mask	corresponding network mask to set (could be NULL to set only the IP address)
setDefault	if true, the interface default address/mask is also set

Function

bool TCPIP_STACK_NetAddressSet(TCPIP_NET_HANDLE netH, IPV4_ADDR* ipAddress,
IPV4_ADDR* mask, bool setDefault)

TCPIP_STACK_NetIPv6AddressGet Function

Gets network interface IPv6 address.

File

tcpip_manager.h

C

```
IPV6_ADDR_HANDLE TCPIP_STACK_NetIPv6AddressGet(TCPIP_NET_HANDLE netH, IPV6_ADDR_TYPE addType,
IPV6_ADDR_STRUCT* pAddStruct, IPV6_ADDR_HANDLE addHandle);
```

Returns

- Non-NULL IPV6_ADDR_HANDLE - if an valid IPv6 address was found and the pAddStruct structure was filled with data
- 0 - if no other IPv6 exists or if the supplied IPV6_ADDR_HANDLE is invalid

Description

This function allows the listing of the IPv6 addresses associated with an interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
IPV6_ADDR_STRUCT currAddr;
IPV6_ADDR_HANDLE currHandle;
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("PIC32INT");
char    ipv6AddBuff[44];

currHandle = 0;
do
{
     currHandle = TCPIP_STACK_NetIPv6AddressGet(netH, IPV6_ADDR_TYPE_UNICAST, &currAddr, currHandle);
     if(currHandle)
     {   // have a valid address; display it
         TCPIP_HELPER_IPv6AddressToString(&currAddr.address, ipv6AddBuff, sizeof(ipv6AddBuff));
     }
}while(currHandle != 0);
```

Parameters

Parameters	Description
netH	handle of the interface to retrieve the addresses for
addType	type of address to request IPV6_ADDR_TYPE_UNICAST and IPV6_ADDR_TYPE_MULTICAST supported for now
pAddStruct	structure provided by the user that will be filled with corresponding IPV6_ADDR_STRUCT data
addHandle	an address handle that allows iteration across multiple IPv6 addresses.
On the first call	has to be 0; it will begin the listing of the IPv6 addresses
On subsequent calls	has to be a handle previously returned by a call to this function

Function

IPV6_ADDR_HANDLE TCPIP_STACK_NetIPv6AddressGet(TCPIP_NET_HANDLE netH,

IPV6_ADDR_TYPE addType, IPV6_ADDR_STRUCT* pAddStruct, IPV6_ADDR_HANDLE addHandle);

TCPIP_STACK_ModuleConfigGet Function

Get stack module configuration data.

File

tcpip_manager.h

C

```
size_t TCPIP_STACK_ModuleConfigGet(TCPIP_STACK_MODULE modId, void* configBuff, size_t buffSize, size_t*
pNeededSize);
```

Returns

The number of bytes copied to the user buffer:

- -1 - if the module ID is invalid
- 0 - if the configBuff is NULL or buffSize is less than required
- > 0 - if the call succeeded and the configuration was copied

Description

This function returns the current configuration data of the stack module specified by the corresponding module ID.

Remarks

Currently, only the MAC modules implement this function.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
uint8_t configBuffer[200];
size_t configSize;
size_t copiedSize;
copiedSize = TCPIP_STACK_ModuleConfigGet(TCPIP_MODULE_MAC_MRF24W, configBuffer,
                                         sizeof(configBuffer), &configSize);
```

Parameters

Parameters	Description
modId	the ID that identifies the requested module
configBuff	pointer to a buffer that will receive the configuration data If this pointer is 0, just the pNeededSize will be updated
buffSize	size of the provided buffer
pNeededSize	pointer to an address to store the number of bytes needed to store this module configuration data. Can be NULL if not needed.

Function

size_t TCPIP_STACK_ModuleConfigGet(TCPIP_STACK_MODULE modId, void* configBuff,

size_t buffSize, size_t* pNeededSize)

TCPIP_STACK_NetMACIdGet Function

Get the MAC ID of the network interface.

File

tcpip_manager.h

C

```
TCPIP_STACK_MODULE TCPIP_STACK_NetMACIdGet(TCPIP_NET_HANDLE netH);
```

Returns

A TCPIP_STACK_MODULE ID that belongs to the MAC of that network interface.

Description

This function returns the module ID of the MAC that's attached to the specified network interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
TCPIP_STACK_MODULE modId = TCPIP_STACK_NetMACIdGet(netH);
if(modId == TCPIP_MODULE_MAC_PIC32INT)
{
     // an internal PIC32 MAC attached to this interface
}
```

Parameters

Parameters	Description
netH	handle of the interface to use

Function

TCPIP_STACK_MODULE TCPIP_STACK_NetMACIdGet(TCPIP_NET_HANDLE netH)

TCPIP_STACK_NetMACStatisticsGet Function

Get the MAC statistics data.

File

tcpip_manager.h

C

```
bool TCPIP_STACK_NetMACStatisticsGet(TCPIP_NET_HANDLE netH, TCPIP_MAC_RX_STATISTICS* pRxStatistics,
TCPIP_MAC_TX_STATISTICS* pTxStatistics);
```

Returns

- true - if the call succeeded
- false - if the call failed (the corresponding MAC does not implement statistics counters)

Description

This function returns the statistics data of the MAC that's attached to the specified network interface.

Remarks

None.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY. The network interface should be up and running.

Example

```
TCPIP_MAC_RX_STATISTICS rxStatistics;
TCPIP_MAC_TX_STATISTICS txStatistics;
TCPIP_NET_HANDLE netH = TCPIP_STACK_NetHandleGet("PIC32INT");
if(TCPIP_STACK_NetMACStatisticsGet(netH, &rxStatistics, &txStatistics))
{
     // display the statistics for the internal PIC32 MAC attached to this interface
}
```

Parameters

Parameters	Description
netH	handle of the interface to use
pRxStatistics	pointer to a TCPIP_MAC_RX_STATISTICS that will receive the current RX statistics counters. Can be NULL if not needed.
pTxStatistics	pointer to a TCPIP_MAC_TX_STATISTICS that will receive the current TX statistics counters. Can be NULL if not needed.

Function

bool TCPIP_STACK_NetMACStatisticsGet(TCPIP_NET_HANDLE netH,
TCPIP_MAC_RX_STATISTICS* pRxStatistics, TCPIP_MAC_TX_STATISTICS* pTxStatistics)

TCPIP_STACK_Status Function

Provides the current status of the TCPIP stack module.

File

tcpip_manager.h

C

```
SYS_STATUS TCPIP_STACK_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_READY - Indicates that any previous initialization operation for the stack has completed
- SYS_STATUS_BUSY - Indicates that a previous initialization operation for the stack has not yet completed

- SYS_STATUS_ERROR - Indicates that the initialization operation has failed and the stack is in an error state

Description

This function provides the current status of the TCPIP stack module.

Remarks

After calling TCPIP_STACK_Initialize, the stack will contiue its initialization process in the TCPIP_STACK_Task routine. This that may take some time. The stack is ready to use only when the TCPIP_STACK_Status returned SYS_STATUS_READY.

Preconditions

The TCPIP_STACK_Initialize function must have been called before calling this function.

Parameters

Parameters	Description
object	Object handle, returned from TCPIP_STACK_Initialize

Function

SYS_STATUS TCPIP_STACK_Status (SYS_MODULE_OBJ object)

f) Network Structure Storage Functions

TCPIP_STACK_NetConfigGet Function

Get stack network interface configuration data.

File

tcpip_manager.h

C

```
size_t TCPIP_STACK_NetConfigGet(TCPIP_NET_HANDLE netH, void* configStoreBuff, size_t configStoreSize,
size_t* pNeededSize);
```

Returns

- -1 - if the interface is invalid or the stack is not initialized
- 0 - if no data is copied (no supplied buffer of buffer too small)
- > 0 - for success, indicating the amount of data copied.

Description

This function dumps the current configuration data of the network interface specified by the corresponding network handle into the supplied buffer.

Remarks

The function is a helper for retrieving the network configuration data. Its companion function, TCPIP_STACK_NetConfigSet, restores the TCPIP_NETWORK_CONFIG from the dump buffer.

Currently, the data is saved in plain binary format into the supplied buffer. However, the application must not make use of this assumption as it may change in a future release (some compression scheme may be implemented).

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
uint8_t currConfig[100];
size_t neededSize, result;
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("PIC32INT");
result = TCPIP_STACK_NetConfigGet(hNet, currConfig, sizeof(currConfig), &neededSize);
if(result > 0)
{
     // store the currConfig to some external storage
}
```

Parameters

Parameters	Description
netH	the handle that identifies the requested interface
configStoreBuff	pointer to a buffer that will receive the current configuration data. All the data that's needed to restore a TCPIP_NETWORK_CONFIG structure is stored in this buffer. Can be NULL if only the storage size is needed.
configStoreSize	size of the supplied buffer
pNeededSize	pointer to store the size needed for storage; Can be NULL if not needed

Function

size_t TCPIP_STACK_NetConfigGet(TCPIP_NET_HANDLE netH, void* configStoreBuff,

size_t configStoreSize, size_t* pNeededSize)

TCPIP_STACK_NetConfigSet Function

Restores stack network interface configuration data.

File

tcpip_manager.h

C

```
TCPIP_NETWORK_CONFIG* TCPIP_STACK_NetConfigSet(void* configStoreBuff, void* netConfigBuff, size_t buffSize,
size_t* pNeededSize);
```

Returns

- valid TCPIP_NETWORK_CONFIG pointer (netConfigBuff) if netConfigBuff was successfully updated
- 0 if the netConfigBuff is not supplied or is not large enough

Description

This function restores data from a previously dump buffer and updates the supplied interface configuration. All the data is recovered and constructed into the netConfigBuff (supposing this buffer is large enough). If this operation succeeded, the netConfigBuff can be safely cast to a (TCPIP_NETWORK_CONFIG*).

The structure of the netConfigBuff is as follows:

- A TCPIP_NETWORK_CONFIG structure is created at the very beginning of the buffer.
- All of the necessary fields that are part of the TCPIP_NETWORK_CONFIG will be placed in the buffer itself.

Remarks

The function is a helper for being able to restore the configuration data. Its companion function, TCPIP_STACK_NetConfigGet, saves the TCPIP_NETWORK_CONFIG to a dump buffer.

Preconditions

The TCP/IP stack should have been initialized by TCPIP_STACK_Initialize and the TCPIP_STACK_Status returned SYS_STATUS_READY.

Example

```
uint8_t currConfig[100];
uint8_t restoreBuff[100];
size_t neededSize, result;
TCPIP_NET_HANDLE hNet = TCPIP_STACK_NetHandleGet("PIC32INT");
result = TCPIP_STACK_NetConfigGet(hNet, currConfig, sizeof(currConfig), &neededSize);
if(result > 0)
{
     // store the currConfig buffer to some external storage (neededSize bytes needed)

     // later on restore the configuration
     TCPIP_NETWORK_CONFIG* netConfig;
     // extract the network configuration from the previously saved buffer
     netConfig = TCPIP_STACK_NetConfigSet(currConfig, restoreBuff, sizeof(restoreBuff), neededSize);
     if(netConfig)
     {
         // use this netConfig to initialize a network interface
         TCPIP_STACK_NetUp(hNet, netConfig);
```

```
        }
    }
```

Parameters

Parameters	Description
configStoreBuff	pointer to a buffer that received configuration data from a TCPIP_STACK_NetConfigGet call.
netConfigBuff	pointer to a buffer that will receive the TCPIP_NETWORK_CONFIG data
buffSize	size of the supplied netConfigBuff buffer
pNeededSize	pointer to store the size needed for storage; can be NULL if not needed. If supplied, the pNeededSize will be updated with the actual size that's needed for the netConfigBuff.

Function

TCPIP_NETWORK_CONFIG* TCPIP_STACK_NetConfigSet(void* configStoreBuff, void* netConfigBuff, size_t buffSize, size_t* pNeededSize);

g) Data Types and Constants

TCPIP_LOCAL_MASK_TYPE Enumeration

Mask operation for local/non-local network interface.

File

tcpip_manager.h

C

```
typedef enum {
  TCPIP_LOCAL_MASK_ZERO,
  TCPIP_LOCAL_MASK_ONE,
  TCPIP_LOCAL_MASK_NET,
  TCPIP_LOCAL_MASK_SET
} TCPIP_LOCAL_MASK_TYPE;
```

Members

Members	Description
TCPIP_LOCAL_MASK_ZERO	use an all zero network mask
TCPIP_LOCAL_MASK_ONE	use an all ones network mask
TCPIP_LOCAL_MASK_NET	use the current network mask
TCPIP_LOCAL_MASK_SET	use the set value for the network mask

Description

Enumeration: TCPIP_LOCAL_MASK_TYPE

These are the available operations that set the local/non-local detection network mask.

TCPIP_NET_HANDLE Type

Defines a network interface handle.

File

tcpip_manager.h

C

```
typedef const void* TCPIP_NET_HANDLE;
```

Description

Type: TCPIP_NET_HANDLE

Definition of the network handle which clients use to get access to control the interfaces.

TCPIP_EVENT Enumeration

Defines the possible TCPIP event types.

File

tcpip_manager.h

C

```
typedef enum {
  TCPIP_EV_NONE = 0x0000,
  TCPIP_EV_RX_PKTPEND = 0x0001,
  TCPIP_EV_RX_OVFLOW = 0x0002,
  TCPIP_EV_RX_BUFNA = 0x0004,
  TCPIP_EV_RX_ACT = 0x0008,
  TCPIP_EV_RX_DONE = 0x0010,
  TCPIP_EV_RX_FWMARK = 0x0020,
  TCPIP_EV_RX_EWMARK = 0x0040,
  TCPIP_EV_RX_BUSERR = 0x0080,
  TCPIP_EV_TX_DONE = 0x0100,
  TCPIP_EV_TX_ABORT = 0x0200,
  TCPIP_EV_TX_BUSERR = 0x0400,
  TCPIP_EV_CONN_ESTABLISHED = 0x0800,
  TCPIP_EV_CONN_LOST = 0x1000,
  TCPIP_EV_RX_ALL = (TCPIP_EV_RX_PKTPEND|TCPIP_EV_RX_OVFLOW|TCPIP_EV_RX_BUFNA|TCPIP_EV_RX_ACT|
TCPIP_EV_RX_DONE|TCPIP_EV_RX_FWMARK|TCPIP_EV_RX_EWMARK|TCPIP_EV_RX_BUSERR),
  TCPIP_EV_TX_ALL = (TCPIP_EV_TX_DONE|TCPIP_EV_TX_ABORT|TCPIP_EV_TX_BUSERR),
  TCPIP_EV_RXTX_ERRORS = (TCPIP_EV_RX_OVFLOW|TCPIP_EV_RX_BUFNA|TCPIP_EV_RX_BUSERR|
TCPIP_EV_TX_ABORT|TCPIP_EV_TX_BUSERR),
  TCPIP_EV_CONN_ALL = (TCPIP_EV_CONN_ESTABLISHED|TCPIP_EV_CONN_LOST)
} TCPIP_EVENT;
```

Members

Members	Description
TCPIP_EV_NONE = 0x0000	no event
TCPIP_EV_RX_PKTPEND = 0x0001	A receive packet is pending
TCPIP_EV_RX_OVFLOW = 0x0002	RX FIFO overflow (system level latency, no descriptors, etc.)
TCPIP_EV_RX_BUFNA = 0x0004	no RX descriptor available to receive a new packet
TCPIP_EV_RX_ACT = 0x0008	There's RX data available
TCPIP_EV_RX_DONE = 0x0010	A packet was successfully received
TCPIP_EV_RX_FWMARK = 0x0020	the number of received packets is >= than the RX Full Watermark
TCPIP_EV_RX_EWMARK = 0x0040	the number of received packets is <= than the RX Empty Watermark
TCPIP_EV_RX_BUSERR = 0x0080	a bus error encountered during an RX transfer
TCPIP_EV_TX_DONE = 0x0100	A packet was transmitted and it's status is available
TCPIP_EV_TX_ABORT = 0x0200	a TX packet was aborted by the MAC (jumbo/system under-run/excessive defer/late collision/excessive collisions)
TCPIP_EV_TX_BUSERR = 0x0400	a bus error encountered during a TX transfer
TCPIP_EV_CONN_ESTABLISHED = 0x0800	Connection Established
TCPIP_EV_CONN_LOST = 0x1000	Connection Lost
TCPIP_EV_RX_ALL = (TCPIP_EV_RX_PKTPEND|TCPIP_EV_RX_OVFLOW|TCPIP_EV_RX_BUFNA|TCPIP_EV_RX_ACT| TCPIP_EV_RX_DONE|TCPIP_EV_RX_FWMARK|TCPIP_EV_RX_EWMARK|TCPIP_EV_RX_BUSERR)	Mask of all RX related events
TCPIP_EV_TX_ALL = (TCPIP_EV_TX_DONE|TCPIP_EV_TX_ABORT|TCPIP_EV_TX_BUSERR)	Mask of all TX related events
TCPIP_EV_RXTX_ERRORS = (TCPIP_EV_RX_OVFLOW|TCPIP_EV_RX_BUFNA|TCPIP_EV_RX_BUSERR| TCPIP_EV_TX_ABORT|TCPIP_EV_TX_BUSERR)	All events showing some abnormal traffic/system condition Action should be taken accordingly by the stack (or the stack user)
TCPIP_EV_CONN_ALL = (TCPIP_EV_CONN_ESTABLISHED|TCPIP_EV_CONN_LOST)	Mask of all Connection related events

Description

TCPIP Stack Events Codes

This enumeration defines all the possible events that can be reported by the TCPIP stack. These are events received by the stack from the network interfaces. They are reported by the MAC driver of the network interface.

Remarks

Depending on the type of the hardware interface, not all events are possible.

Note that specific interfaces can offer specific events and functions to retrieve those events.

TCPIP_EVENT_HANDLE Type

Defines a TCPIP stack event handle.

File

tcpip_manager.h

C

```
typedef const void* TCPIP_EVENT_HANDLE;
```

Description

Type: TCPIP_EVENT_HANDLE

Definition of an event handle used for event registration by the stack clients.

TCPIP_STACK_EVENT_HANDLER Type

Pointer to a function(handler) that will get called to process an event.

File

tcpip_manager.h

C

```
typedef void (* TCPIP_STACK_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, TCPIP_EVENT, const void* fParam);
```

Returns

None.

Description

TCPIP event notification handler Pointer

Pointer to a function that may be called from within an ISR when a TCP/IP event is available.

Remarks

This function may be invoked from within an ISR. It should be kept as short as possible and it should not include blocking or polling code.

Preconditions

None

Parameters

Parameters	Description
hNet	network handle
tcpEvent	ORed mask of events that occurred
fParam	user passed parameter

Function

void* (TCPIP_NET_HANDLE hNet, TCPIP_EVENT tcpEvent, void* fParam)

TCPIP_Helper_ntohl Macro

File

tcpip_helpers.h

C

```
#define TCPIP_Helper_ntohl(n) TCPIP_Helper_htonl(n)
```

Description

This is macro TCPIP_Helper_ntohl.

TCPIP_Helper_ntohs Macro

File

tcpip_helpers.h

C

```
#define TCPIP_Helper_ntohs(n) TCPIP_Helper_htons(n)
```

Description

This is macro TCPIP_Helper_ntohs.

TCPIP_STACK_IF_NAME_ALIAS_ETH Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_ALIAS_ETH "eth"
```

Description

alias for Ethernet interface

TCPIP_STACK_IF_NAME_ALIAS_UNK Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_ALIAS_UNK "unk"
```

Description

alias for unknown interface

TCPIP_STACK_IF_NAME_ALIAS_WLAN Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_ALIAS_WLAN "wlan"
```

Description

alias for Wi-Fi interface

IP_ADDR Type

Backward compatible definition to represent an IPv4 address.

File

tcpip.h

C

```
typedef IPV4_ADDR IP_ADDR;
```

Description

IPv4 Address backward compatible type

This type describes the backward compatible IPv4 address type.

Remarks

None.

IP_ADDRESS_TYPE Enumeration

Definition of the TCP/IP supported address types.

File

tcpip.h

C

```
typedef enum {
  IP_ADDRESS_TYPE_ANY,
  IP_ADDRESS_TYPE_IPV4,
  IP_ADDRESS_TYPE_IPV6
} IP_ADDRESS_TYPE;
```

Members

Members	Description
IP_ADDRESS_TYPE_ANY	either IPv4 or IPv6, unspecified;
IP_ADDRESS_TYPE_IPV4	IPv4 address type
IP_ADDRESS_TYPE_IPV6	IPv6 address type

Description

TCP/IP IP Address type

This type describes the supported IP address types.

Remarks

None.

IP_MULTI_ADDRESS Union

Definition to represent multiple IP addresses.

File

tcpip.h

C

```
typedef union {
  IPV4_ADDR v4Add;
  IPV6_ADDR v6Add;
} IP_MULTI_ADDRESS;
```

Description

TCP/IP multiple Address type

This type describes the TCP/IP multiple address type.

Remarks

None.

IPV4_ADDR Union

Definition to represent an IPv4 address

File

tcpip.h

C

```
typedef union {
  uint32_t Val;
  uint16_t w[2];
  uint8_t v[4];
} IPV4_ADDR;
```

Description

IPv4 Address type

This type describes the IPv4 address type.

Remarks

None.

IPV6_ADDR Union

Definition to represent an IPv6 address.

File

tcpip.h

C

```
typedef union {
  uint8_t v[16];
  uint16_t w[8];
  uint32_t d[4];
} IPV6_ADDR;
```

Description

IPv6 Address type

This type describes the IPv6 address type.

Remarks

None.

IPV6_ADDR_HANDLE Type

Definition to represent an IPv6 address.

File

tcpip.h

C

```
typedef const void* IPV6_ADDR_HANDLE;
```

Description

IPv6 Address handle

This type describes an IPv6 address handle.

Remarks

None.

IPV6_ADDR_SCOPE Enumeration

Definition to represent the scope of an IPv6 address.

File

tcpip.h

C

```
typedef enum {
  IPV6_ADDR_SCOPE_UNKNOWN,
  IPV6_ADDR_SCOPE_INTERFACE_LOCAL,
  IPV6_ADDR_SCOPE_LINK_LOCAL,
  IPV6_ADDR_SCOPE_ADMIN_LOCAL,
  IPV6_ADDR_SCOPE_SITE_LOCAL,
  IPV6_ADDR_SCOPE_ORG_LOCAL,
  IPV6_ADDR_SCOPE_GLOBAL
} IPV6_ADDR_SCOPE;
```

Description

IPv6 Address scope

This type describes the IPv6 address scope.

Remarks

None.

IPV6_ADDR_STRUCT Structure

COmplex type to represent an IPv6 address.

File

tcpip.h

C

```
typedef struct _IPV6_ADDR_STRUCT {
  struct _IPV6_ADDR_STRUCT * next;
  struct _IPV6_ADDR_STRUCT * prev;
  IPV6_ADDR address;
  unsigned long validLifetime;
  unsigned long preferredLifetime;
  unsigned long lastTickTime;
  unsigned char prefixLen;
  struct {
    unsigned char precedence;
    unsigned scope : 4;
    unsigned label : 4;
    unsigned type : 2;
    unsigned temporary : 1;
  } flags;
} IPV6_ADDR_STRUCT;
```

Members

Members	Description
unsigned char precedence;	Allow preferences
unsigned scope : 4;	Link-local, site-local, global.
unsigned label : 4;	Policy label
unsigned type : 2;	Uni-, Any-, Multi-cast
unsigned temporary : 1;	Indicates that the address is temporary (not public)

Description

IPv6 Address Structure

This type defines all the fields of an IPv6 address.

Remarks

None.

IPV6_ADDR_TYPE Enumeration

Definition to represent the type of an IPv6 address.

File

tcpip.h

C

```
typedef enum {
  IPV6_ADDR_TYPE_UNKNOWN,
  IPV6_ADDR_TYPE_UNICAST,
  IPV6_ADDR_TYPE_ANYCAST,
  IPV6_ADDR_TYPE_MULTICAST,
  IPV6_ADDR_TYPE_SOLICITED_NODE_MULTICAST,
  IPV6_ADDR_TYPE_UNICAST_TENTATIVE
} IPV6_ADDR_TYPE;
```

Members

Members	Description
IPV6_ADDR_TYPE_UNKNOWN	Invalid/unknown address type
IPV6_ADDR_TYPE_UNICAST	Only link-local and global are currently valid for unicast

Description

IPv6 Address type

This type describes the possible type of an IPv6 address.

Remarks

None.

TCPIP_NETWORK_CONFIG Structure

Defines the data required to initialize the network configuration.

File

tcpip.h

C

```
typedef struct {
  char* interface;
  char* hostName;
  char* macAddr;
  char* ipAddr;
  char* ipMask;
  char* gateway;
  char* priDNS;
  char* secondDNS;
  char* powerMode;
  TCPIP_NETWORK_CONFIG_FLAGS startFlags;
  const struct TCPIP_MAC_OBJECT_TYPE* pMacObject;
  char* ipv6Addr;
  int ipv6PrefixLen;
  char* ipv6Gateway;
} TCPIP_NETWORK_CONFIG;
```

Members

Members	Description
char* interface;	Pointer to the interface name; could be NULL.
char* hostName;	Valid Host name for this interface to use. Ex: "MCHPBOARD
char* macAddr;	MAC address to use for this interface. Use "00:04:a3:00:00:00" or 0 for the factory preprogrammed address
char* ipAddr;	Static IP address to use. Ex: "169.254.1.1
char* ipMask;	Netmask to use. Ex: "255.255.0.0
char* gateway;	Static Gateway to use. Ex: "169.254.1.1
char* priDNS;	Primary DNS to use. Ex: "169.254.1.1
char* secondDNS;	Secondary DNS to use. Use "0.0.0.0" for none
char* powerMode;	Power Mode to use. Use TCPIP_STACK_IF_POWER_NONE, TCPIP_STACK_IF_POWER_FULL, TCPIP_STACK_IF_POWER_LOW, or TCPIP_STACK_IF_POWER_DOWN
TCPIP_NETWORK_CONFIG_FLAGS startFlags;	flags for interface start-up

const struct TCPIP_MAC_OBJECT_TYPE* pMacObject;	Non-volatile pointer to the MAC driver object associated with this network interface This is the MAC driver that this interface will use
char* ipv6Addr;	static IPv6 address; only if TCPIP_NETWORK_CONFIG_IPV6_ADDRESS specified can be NULL if not needed
int ipv6PrefixLen;	subnet prefix length; only if TCPIP_NETWORK_CONFIG_IPV6_ADDRESS specified 0 means default value (64) should probably always be 64 as requested by the RFC
char* ipv6Gateway;	default IPv6 gateway address; only if TCPIP_NETWORK_CONFIG_IPV6_ADDRESS specified can be NULL if not needed

Description

Multi-Homed Hosts Network Addressing Configuration Data

This data type defines the data required to initialize the network configuration for a specific interface.

Remarks

None.

TCPIP_NETWORK_CONFIG_FLAGS Enumeration

Definition of network configuration start-up flags.

File

tcpip.h

C

```
typedef enum {
  TCPIP_NETWORK_CONFIG_IP_STATIC,
  TCPIP_NETWORK_CONFIG_DHCP_CLIENT_ON,
  TCPIP_NETWORK_CONFIG_ZCLL_ON,
  TCPIP_NETWORK_CONFIG_DHCP_SERVER_ON,
  TCPIP_NETWORK_CONFIG_DNS_CLIENT_ON,
  TCPIP_NETWORK_CONFIG_DNS_SERVER_ON,
  TCPIP_NETWORK_CONFIG_IPV6_ADDRESS
} TCPIP_NETWORK_CONFIG_FLAGS;
```

Members

Members	Description
TCPIP_NETWORK_CONFIG_IP_STATIC	Start the interface with a static IP address No address service is enabled on this interface (DHCPc, ZCLL or DHCPs)
TCPIP_NETWORK_CONFIG_DHCP_CLIENT_ON	DHCP client enabled on this interface
TCPIP_NETWORK_CONFIG_ZCLL_ON	ZeroConf LinkLocal enabled on this interface
TCPIP_NETWORK_CONFIG_DHCP_SERVER_ON	DHCP server enabled on this interface
TCPIP_NETWORK_CONFIG_DNS_CLIENT_ON	DNS CLIENT enabled on this interface
TCPIP_NETWORK_CONFIG_DNS_SERVER_ON	DNS Server Enabled on this Interface
TCPIP_NETWORK_CONFIG_IPV6_ADDRESS	the network configuration contains an IPv6 static address and subnet prefix length

Description

Network Configuration start-up flags

This enumerated type describes the list of the supported TCP/IP network configuration start-up flags.

Remarks

DHCPc, DHCPs and ZCLL are conflicting and cannot be more than one enabled on the same interface! The order of priorities is: DHCPc, ZCLL, DHCPs in case of conflict

Only one of either DNS server or client can be enabled per interface.

Valid values are 0x0001 - 0x8000; 16 bits are maintained only.

TCPIP_STACK_INIT Structure

Definition to represent the TCP/IP stack initialization/configuration structure.

File

tcpip.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  const TCPIP_NETWORK_CONFIG* pNetConf;
  int nNets;
  const TCPIP_STACK_MODULE_CONFIG* pModConfig;
  int nModules;
} TCPIP_STACK_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	system module initialization
const TCPIP_NETWORK_CONFIG* pNetConf;	pointer to array of network configurations
int nNets;	number of networks in the configuration array
const TCPIP_STACK_MODULE_CONFIG* pModConfig;	pointer to array of module configurations
int nModules;	number of modules in the array

Description

TCP/IP stack initialization/configuration structure

This type describes the TCP/IP stack initialization/configuration data type that's passed to the TCP/IP stack at the initialization time.

Remarks

This data structure will be passed to the stack initialization function, TCPIP_STACK_Initialize.

TCPIP_STACK_MODULE Enumeration

List of the TCP/IP stack supported modules.

File

tcpip.h

C

```
typedef enum {
  TCPIP_MODULE_NONE = 0,
  TCPIP_MODULE_IPV4,
  TCPIP_MODULE_IPV6,
  TCPIP_MODULE_LLDP,
  TCPIP_MODULE_ICMPV6,
  TCPIP_MODULE_NDP,
  TCPIP_MODULE_UDP,
  TCPIP_MODULE_TCP,
  TCPIP_MODULE_IGMP,
  TCPIP_MODULE_DHCP_SERVER,
  TCPIP_MODULE_ANNOUNCE,
  TCPIP_MODULE_DNS_CLIENT,
  TCPIP_MODULE_DNS_SERVER,
  TCPIP_MODULE_ZCLL,
  TCPIP_MODULE_MDNS,
  TCPIP_MODULE_NBNS,
  TCPIP_MODULE_SMTP_CLIENT,
  TCPIP_MODULE_SNTP,
  TCPIP_MODULE_FTP_SERVER,
  TCPIP_MODULE_HTTP_SERVER,
  TCPIP_MODULE_HTTP_NET_SERVER,
  TCPIP_MODULE_TELNET_SERVER,
  TCPIP_MODULE_SNMP_SERVER,
  TCPIP_MODULE_SNMPV3_SERVER,
  TCPIP_MODULE_DYNDNS_CLIENT,
  TCPIP_MODULE_BERKELEY,
  TCPIP_MODULE_REBOOT_SERVER,
  TCPIP_MODULE_COMMAND,
  TCPIP_MODULE_IPERF,
  TCPIP_MODULE_TFTP_CLIENT,
  TCPIP_MODULE_DHCPV6_CLIENT,
  TCPIP_MODULE_SMTPC,
  TCPIP_MODULES_NUMBER,
```

```
  TCPIP_MODULE_MAC_START
} TCPIP_STACK_MODULE;
```

Members

Members	Description
TCPIP_MODULE_NONE = 0	unspecified/unknown module
TCPIP_MODULE_LLDP	LLDP module
TCPIP_MODULE_IGMP	IGMP host module
TCPIP_MODULE_ZCLL	Zero Config Link Local
TCPIP_MODULE_MDNS	Bonjour/mDNS
TCPIP_MODULE_TFTP_CLIENT	TFTP client module
TCPIP_MODULE_DHCPV6_CLIENT	DHCPV6 client
TCPIP_MODULE_SMTPC	SMTP (new) client
TCPIP_MODULES_NUMBER	number of modules in the TCP/IP stack itself starting here is list of supported MAC modules and are defined in the tcpip_mac.h

Description

TCP/IP stack supported modules

The following enumeration lists all the modules supported by the TCP/IP stack.

Remarks

None.

TCPIP_STACK_MODULE_CONFIG Structure

Definition to represent a TCP/IP module initialization/configuration structure.

File

tcpip.h

C

```
typedef struct {
  TCPIP_STACK_MODULE moduleId;
  const void * const configData;
} TCPIP_STACK_MODULE_CONFIG;
```

Description

TCP/IP module initialization/configuration structure

This type describes a TCP/IP module initialization/configuration data type that's passed to a TCP/IP module at the stack initialization time.

Remarks

Each stack module will be configured with a user-defined initialization/configuration structure.

TCPIP_MODULE_SIGNAL Enumeration

Lists the signals that are generated by the TCP/IP stack manager and processed by the stack modules.

File

tcpip_manager.h

C

```
typedef enum {
  TCPIP_MODULE_SIGNAL_NONE = 0x0000,
  TCPIP_MODULE_SIGNAL_RX_PENDING = 0x0001,
  TCPIP_MODULE_SIGNAL_TMO = 0x0002,
  TCPIP_MODULE_SIGNAL_ASYNC = 0x0100,
  TCPIP_MODULE_SIGNAL_MASK_ALL = (TCPIP_MODULE_SIGNAL_RX_PENDING | TCPIP_MODULE_SIGNAL_TMO)
} TCPIP_MODULE_SIGNAL;
```

Members

Members	Description
TCPIP_MODULE_SIGNAL_NONE = 0x0000	no pending signal

TCPIP_MODULE_SIGNAL_RX_PENDING = 0x0001	RX packet pending for processing
TCPIP_MODULE_SIGNAL_TMO = 0x0002	module timeout has occurred for modules that implement a state machine advancing on timer signals
TCPIP_MODULE_SIGNAL_ASYNC = 0x0100	Special signals module asynchronous attention required this is a signal that's requested by modules that need special attention module is required to clear this flag when out of the critical processing
TCPIP_MODULE_SIGNAL_MASK_ALL = (TCPIP_MODULE_SIGNAL_RX_PENDING | TCPIP_MODULE_SIGNAL_TMO)	All processing signals mask Note that TCPIP_MODULE_SIGNAL_ASYNC has to be cleared explicitly

Description

Enumeration: TCPIP_MODULE_SIGNAL

These signals are generated by the stack manager towards the internal stack modules; A stack app could use them to provide wake up conditions for a thread waiting for a signal to occur. This is mainly useful when the stack modules tasks functions are executed at the app level and the app needs a way to identify that a module needs attention.

Remarks

Multiple signals could be ORed.

Only 16-bit values are maintained for this type of signal.

TCPIP_MODULE_SIGNAL_FUNC Type

Pointer to a function(handler) that will get called when a stack internal signal occurred.

File

tcpip_manager.h

C

```
typedef void (* TCPIP_MODULE_SIGNAL_FUNC)(TCPIP_MODULE_SIGNAL_HANDLE sigHandle, TCPIP_STACK_MODULE
moduleId, TCPIP_MODULE_SIGNAL signal);
```

Returns

None.

Description

TCPIP module signal function

Pointer to a function that will be called from within the TCP/IP stack when a signal is delivered to a stack module.

Remarks

This function should be kept as short as possible and it should not include blocking or polling code. It's for setting flags/signaling purposes only.

Preconditions

None.

Parameters

Parameters	Description
sigHandle	signal handle obtained from a registration call
moduleId	module that receives the signal
signal	the occurring signal

Function

void* (TCPIP_MODULE_SIGNAL_HANDLE sigHandle, TCPIP_STACK_MODULE moduleId,
TCPIP_MODULE_SIGNAL signal)

TCPIP_MODULE_SIGNAL_HANDLE Type

Defines a TCPIP stack signal function handle.

File

tcpip_manager.h

C

```
typedef const void* TCPIP_MODULE_SIGNAL_HANDLE;
```

Description

Type: TCPIP_MODULE_SIGNAL_HANDLE

Definition of an signal function handle used for signal registration by the stack clients.

TCPIP_STACK_IF_NAME_97J60 Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_97J60 "97J60"
```

Description

This is macro TCPIP_STACK_IF_NAME_97J60.

TCPIP_STACK_IF_NAME_ENCJ60 Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_ENCJ60 "ENCJ60"
```

Description

This is macro TCPIP_STACK_IF_NAME_ENCJ60.

TCPIP_STACK_IF_NAME_ENCJ600 Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_ENCJ600 "ENCJ600"
```

Description

This is macro TCPIP_STACK_IF_NAME_ENCJ600.

TCPIP_STACK_IF_NAME_MRF24W Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_MRF24W "MRF24W"
```

Description

This is macro TCPIP_STACK_IF_NAME_MRF24W.

TCPIP_STACK_IF_NAME_NONE Macro

Defines the TCP/IP stack supported network interfaces.

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_NONE 0
```

Description

TCP/IP stack supported network interfaces

The following enumeration lists the names of all the interfaces supported by the TCP/IP stack and aliases.

Remarks

None.

TCPIP_STACK_IF_NAME_PIC32INT Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_PIC32INT "PIC32INT"
```

Description

This is macro TCPIP_STACK_IF_NAME_PIC32INT.

TCPIP_STACK_IF_POWER_DOWN Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_POWER_DOWN "down"
```

Description

powered down, not started

TCPIP_STACK_IF_POWER_FULL Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_POWER_FULL "full"
```

Description

up and running;

TCPIP_STACK_IF_POWER_LOW Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_POWER_LOW "low"
```

Description

low power mode; not supported now

TCPIP_STACK_IF_POWER_NONE Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_POWER_NONE 0
```

Description

unsupported, invalid

TCPIP_STACK_VERSION_MAJOR Macro

File

tcpip.h

C

```
#define TCPIP_STACK_VERSION_MAJOR 7
```

Description

TCP/IP stack version

TCPIP_STACK_VERSION_MINOR Macro

File

tcpip.h

C

```
#define TCPIP_STACK_VERSION_MINOR 25
```

Description

This is macro TCPIP_STACK_VERSION_MINOR.

TCPIP_STACK_VERSION_PATCH Macro

File

tcpip.h

C

```
#define TCPIP_STACK_VERSION_PATCH 0
```

Description

This is macro TCPIP_STACK_VERSION_PATCH.

TCPIP_STACK_VERSION_STR Macro

File

tcpip.h

C

```
#define TCPIP_STACK_VERSION_STR "7.26"
```

Description

This is macro TCPIP_STACK_VERSION_STR.

TCPIP_Helper_ntohll Macro

File

tcpip_helpers.h

C

```
#define TCPIP_Helper_ntohll(ll) TCPIP_Helper_htonll(ll)
```

Description

This is macro TCPIP_Helper_ntohll.

TCPIP_STACK_IF_NAME_MRF24WN Macro

File

tcpip.h

C

```
#define TCPIP_STACK_IF_NAME_MRF24WN "MRF24WN"
```

Description

This is macro TCPIP_STACK_IF_NAME_MRF24WN.

Files

Files

Name	Description
tcpip_manager.h	This is the TCP/IP stack manager.
tcpip_helpers.h	TCP/IP Stack helpers header file.
tcpip.h	TCP/IP API definitions.

Description

This section lists the source and header files used by the library.

tcpip_manager.h

This is the TCP/IP stack manager.

Enumerations

	Name	Description
	TCPIP_EVENT	Defines the possible TCPIP event types.
	TCPIP_LOCAL_MASK_TYPE	Mask operation for local/non-local network interface.
	TCPIP_MODULE_SIGNAL	Lists the signals that are generated by the TCP/IP stack manager and processed by the stack modules.

Functions

	Name	Description
	TCPIP_MODULE_SignalFunctionDeregister	Deregisters a signal function for a stack module.
	TCPIP_MODULE_SignalFunctionRegister	Registers a new signal function for a stack module.
	TCPIP_MODULE_SignalGet	Returns the current signals for a TCP/IP module.
	TCPIP_STACK_Deinitialize	Stack deinitialization function.
	TCPIP_STACK_EventsPendingGet	Returns the currently pending events.
	TCPIP_STACK_HandlerDeregister	Deregisters an event notification handler.
	TCPIP_STACK_HandlerRegister	Sets a new event notification handler.
	TCPIP_STACK_IndexToNet	Network interface handle from interface number.
	TCPIP_STACK_Initialize	Stack initialization function.
	TCPIP_STACK_InitializeDataGet	Get the TCP/IP stack initialization data.
	TCPIP_STACK_MACObjectGet	Returns the network MAC driver object of this interface.
	TCPIP_STACK_ModuleConfigGet	Get stack module configuration data.
	TCPIP_STACK_NetAddress	Network interface IPv4 address.
	TCPIP_STACK_NetAddressBcast	Network interface broadcast address.
	TCPIP_STACK_NetAddressDnsPrimary	Network interface DNS address.
	TCPIP_STACK_NetAddressDnsPrimarySet	Sets network interface IPv4 DNS address.
	TCPIP_STACK_NetAddressDnsSecond	Network interface secondary DNS address.
	TCPIP_STACK_NetAddressDnsSecondSet	Sets network interface IPv4 secondary DNS address.

	TCPIP_STACK_NetAddressGateway	Network interface IPv4 gateway address.
	TCPIP_STACK_NetAddressGatewaySet	Sets network interface IPv4 gateway address.
	TCPIP_STACK_NetAddressMac	Network interface MAC address.
	TCPIP_STACK_NetAddressMacSet	Sets network interface MAC address.
	TCPIP_STACK_NetAddressSet	Sets network interface IPv4 address.
	TCPIP_STACK_NetAliasNameGet	Network interface alias name from a handle.
	TCPIP_STACK_NetBIOSName	Network interface NetBIOS name.
	TCPIP_STACK_NetBiosNameSet	Sets network interface NetBIOS name.
	TCPIP_STACK_NetConfigGet	Get stack network interface configuration data.
	TCPIP_STACK_NetConfigSet	Restores stack network interface configuration data.
	TCPIP_STACK_NetDefaultGet	Default network interface handle.
	TCPIP_STACK_NetDefaultSet	Sets the default network interface handle.
	TCPIP_STACK_NetDown	Turns down a network interface.
	TCPIP_STACK_NetHandleGet	Network interface handle from a name.
	TCPIP_STACK_NetIndexGet	Network interface number from interface handle.
	TCPIP_STACK_NetIPv6AddressGet	Gets network interface IPv6 address.
	TCPIP_STACK_NetIsLinked	Gets the network interface link status.
	TCPIP_STACK_NetIsReady	Gets the network interface configuration status.
	TCPIP_STACK_NetIsUp	Gets the network interface up or down status.
	TCPIP_STACK_NetMACIdGet	Get the MAC ID of the network interface.
	TCPIP_STACK_NetMACRegisterStatisticsGet	Get the MAC statistics register data.
	TCPIP_STACK_NetMACStatisticsGet	Get the MAC statistics data.
	TCPIP_STACK_NetMask	Network interface IPv4 address mask.
	TCPIP_STACK_NetNameGet	Network interface name from a handle.
	TCPIP_STACK_NetUp	Turns up a network interface. As part of this process, the corresponding MAC driver is initialized.
	TCPIP_STACK_NumberOfNetworksGet	Number of network interfaces in the stack.
	TCPIP_STACK_SetLocalMasks	Sets the local/non-local masks for network interface.
	TCPIP_STACK_SetLocalMasksType	Sets the local/non-local masks type for network interface.
	TCPIP_STACK_Status	Provides the current status of the TCPIP stack module.
	TCPIP_STACK_Task	TCP/IP Stack task function.
	TCPIP_STACK_VersionGet	Gets the TCP/IP stack version in numerical format.
	TCPIP_STACK_VersionStrGet	Gets the TCP/IP stack version in string format.

Types

	Name	Description
	TCPIP_EVENT_HANDLE	Defines a TCPIP stack event handle.
	TCPIP_MODULE_SIGNAL_FUNC	Pointer to a function(handler) that will get called when a stack internal signal occurred.
	TCPIP_MODULE_SIGNAL_HANDLE	Defines a TCPIP stack signal function handle.
	TCPIP_NET_HANDLE	Defines a network interface handle.
	TCPIP_STACK_EVENT_HANDLER	Pointer to a function(handler) that will get called to process an event.

Description

Microchip TCP/IP Stack Definitions

The TCP/IP manager provides access to the stack initialization, configuration and status. It also process the packets received from different network interfaces (MAC drivers) and dispatches them accordingly based on their type to the proper higher layer in the stack.

File Name

tcpip_manager.h

Company

Microchip Technology Inc.

tcpip_helpers.h

TCP/IP Stack helpers header file.

Functions

	Name	Description
	TCPIP_Helper_Base64Decode	Helper function to decode a Base-64 encoded array.
	TCPIP_Helper_Base64Encode	Helper function to encode to Base-64.
	TCPIP_Helper_FormatNetBIOSName	Formats a string to a valid NetBIOS name.
	TCPIP_Helper_htonl	Conversion routines from network order to host order and reverse.
	TCPIP_Helper_htonll	This is function TCPIP_Helper_htonll.
	TCPIP_Helper_htons	This is function TCPIP_Helper_htons.
	TCPIP_Helper_IPAddressToString	Converts an IPV4 address to an ASCII string.
	TCPIP_Helper_IPv6AddressToString	Converts an IPv6 address to a string representation.
	TCPIP_Helper_IsBcastAddress	Checks if an IPv4 address is a broadcast address.
	TCPIP_Helper_IsMcastAddress	Checks if an IPv4 address is a multicast address.
	TCPIP_Helper_IsPrivateAddress	Detects a private (non-routable) address.
	TCPIP_Helper_MACAddressToString	Converts a MAC address to a string.
	TCPIP_Helper_ntohl	!defined(__PIC32MX__)
	TCPIP_Helper_ntohll	This is function TCPIP_Helper_ntohll.
	TCPIP_Helper_ntohs	This is function TCPIP_Helper_ntohs.
	TCPIP_Helper_SecurePortGetByIndex	Returns the secure port belonging to a specified index.
	TCPIP_Helper_SecurePortSet	Sets the required port secure connection status.
	TCPIP_Helper_StringToIPAddress	Converts an ASCII string to an IPV4 address.
	TCPIP_Helper_StringToIPv6Address	Converts a string to an IPv6 address.
	TCPIP_Helper_StringToMACAddress	Converts a string to an MAC address.
	TCPIP_Helper_TCPSecurePortGet	Checks if the required TCP port is a secure port.
	TCPIP_Helper_UDPSecurePortGet	Checks if the required UDP port is a secure port.

Macros

	Name	Description
	TCPIP_Helper_ntohl	This is macro TCPIP_Helper_ntohl.
	TCPIP_Helper_ntohll	This is macro TCPIP_Helper_ntohll.
	TCPIP_Helper_ntohs	This is macro TCPIP_Helper_ntohs.

Description

Header file for tcpip_stack_helpers

This file provides the APIs for the TCP/IP Stack helpers.

File Name

tcpip_helpers.h

Company

Microchip Technology Inc.

tcpip.h

TCP/IP API definitions.

Enumerations

	Name	Description
	IP_ADDRESS_TYPE	Definition of the TCP/IP supported address types.
	IPV6_ADDR_SCOPE	Definition to represent the scope of an IPv6 address.
	IPV6_ADDR_TYPE	Definition to represent the type of an IPv6 address.
	TCPIP_NETWORK_CONFIG_FLAGS	Definition of network configuration start-up flags.
	TCPIP_STACK_MODULE	List of the TCP/IP stack supported modules.

Macros

	Name	Description
	TCPIP_STACK_IF_NAME_97J60	This is macro TCPIP_STACK_IF_NAME_97J60.

	TCPIP_STACK_IF_NAME_ALIAS_ETH	alias for Ethernet interface
	TCPIP_STACK_IF_NAME_ALIAS_UNK	alias for unknown interface
	TCPIP_STACK_IF_NAME_ALIAS_WLAN	alias for Wi-Fi interface
	TCPIP_STACK_IF_NAME_ENCJ60	This is macro TCPIP_STACK_IF_NAME_ENCJ60.
	TCPIP_STACK_IF_NAME_ENCJ600	This is macro TCPIP_STACK_IF_NAME_ENCJ600.
	TCPIP_STACK_IF_NAME_MRF24W	This is macro TCPIP_STACK_IF_NAME_MRF24W.
	TCPIP_STACK_IF_NAME_MRF24WN	This is macro TCPIP_STACK_IF_NAME_MRF24WN.
	TCPIP_STACK_IF_NAME_NONE	Defines the TCP/IP stack supported network interfaces.
	TCPIP_STACK_IF_NAME_PIC32INT	This is macro TCPIP_STACK_IF_NAME_PIC32INT.
	TCPIP_STACK_IF_POWER_DOWN	powered down, not started
	TCPIP_STACK_IF_POWER_FULL	up and running;
	TCPIP_STACK_IF_POWER_LOW	low power mode; not supported now
	TCPIP_STACK_IF_POWER_NONE	unsupported, invalid
	TCPIP_STACK_VERSION_MAJOR	TCP/IP stack version
	TCPIP_STACK_VERSION_MINOR	This is macro TCPIP_STACK_VERSION_MINOR.
	TCPIP_STACK_VERSION_PATCH	This is macro TCPIP_STACK_VERSION_PATCH.
	TCPIP_STACK_VERSION_STR	This is macro TCPIP_STACK_VERSION_STR.

Structures

	Name	Description
	_IPV6_ADDR_STRUCT	COmplex type to represent an IPv6 address.
	IPV6_ADDR_STRUCT	COmplex type to represent an IPv6 address.
	TCPIP_NETWORK_CONFIG	Defines the data required to initialize the network configuration.
	TCPIP_STACK_INIT	Definition to represent the TCP/IP stack initialization/configuration structure.
	TCPIP_STACK_MODULE_CONFIG	Definition to represent a TCP/IP module initialization/configuration structure.

Types

	Name	Description
	IP_ADDR	Backward compatible definition to represent an IPv4 address.
	IPV6_ADDR_HANDLE	Definition to represent an IPv6 address.

Unions

	Name	Description
	IP_MULTI_ADDRESS	Definition to represent multiple IP addresses.
	IPV4_ADDR	Definition to represent an IPv4 address
	IPV6_ADDR	Definition to represent an IPv6 address.

Description

Microchip TCP/IP Stack Include File

This is the global TCP/IP header file that any user of the TCP/IP API should include. It contains the basic TCP/IP types and data structures and includes all the of the TCP/IP stack modules.

File Name

tcpip.h

Company

Microchip Technology Inc.

NBNS Module

This section describes the TCP/IP Stack Library NBNS module.

Introduction

TCP/IP Stack Library NetBIOS (NBNS) Module for Microchip Microcontrollers

This library provides the API of the NBNS module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The NetBIOS Name Service protocol associates host names with IP addresses, similarly to DNS, but on the same IP subnet. Practically, this allows the assignment of human-name hostnames to access boards on the same subnet. For example. in the "TCP/IP Demo App" demonstration project, the development board is programmed with the human name 'mchpboard' so it can be accessed directly instead of with its IP address.

Using the Library

This topic describes the basic architecture of the NBNS TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `nbns.h`

The interface to the NBNS TCP/IP Stack library is defined in the `nbns.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the NBNS TCP/IP Stack library should include `tcpip.h`.

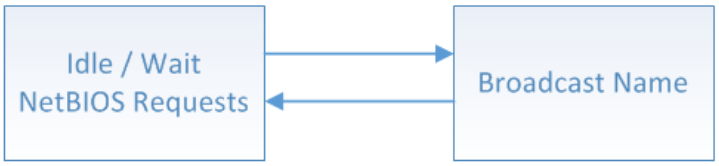
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the NBNS TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

NBNS Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the NBNS module.

Library Interface Section	Description
Functions	This section provides a routine that performs NBNS module tasks in the TCP/IP stack.
Data Types and Constants	This section provides various definitions describing This API

Configuring the Library

Macros

	Name	Description
	TCPIP_NBNS_TASK_TICK_RATE	NBNS task processing rate The default value is 110 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

The configuration of the NBNS TCP/IP Stack is based on the `file tcpip_config.h`.

This header file contains the configuration selection for the NBNS TCP/IP Stack. Based on the selections made, the NBNS TCP/IP Stack may support the selected features. These configuration settings will apply to all instances of the NBNS TCP/IP Stack.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_NBNS_TASK_TICK_RATE Macro

File

nbns_config.h

C

```
#define TCPIP_NBNS_TASK_TICK_RATE 110
```

Description

NBNS task processing rate The default value is 110 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Building the Library

This section lists the files that are available in the NBNS module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/nbns.c`	NBNS implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The NBNS module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_NBNS_Task	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	TCPIP_NBNS_MODULE_CONFIG	Placeholder for NBNS configuration upgrades.

Description

This section describes the Application Programming Interface (API) functions of the NBNS module.

Refer to each section for a detailed description.

a) Functions

TCPIP_NBNS_Task Function

Standard TCP/IP stack module task function.

File

nbns.h

C

```
void TCPIP_NBNS_Task();
```

Returns

None.

Description

This function performs NBNS module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The NBNS module should have been initialized.

Function

void TCPIP_NBNS_Task(void)

b) Data Types and Constants

TCPIP_NBNS_MODULE_CONFIG Structure

Placeholder for NBNS configuration upgrades.

File

nbns.h

C

```
typedef struct {
} TCPIP_NBNS_MODULE_CONFIG;
```

Description

TCPIP_NBNS_MODULE_CONFIG Structure Typedef

This type definition provides a placeholder for NBNS configuration upgrades.

Remarks

None.

Files

Files

Name	Description
nbns.h	The NetBIOS Name Service protocol associates host names with IP addresses, similarly to DNS, but on the same IP subnet.
nbns_config.h	NBNS configuration file

Description

This section lists the source and header files used by the library.

nbns.h

The NetBIOS Name Service protocol associates host names with IP addresses, similarly to DNS, but on the same IP subnet.

Functions

	Name	Description
	TCPIP_NBNS_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_NBNS_MODULE_CONFIG	Placeholder for NBNS configuration upgrades.

Description

NetBIOS Name Service (NBNS) Server public API

The NetBIOS Name Service protocol associates host names with IP addresses, similarly to DNS, but on the same IP subnet. Practically, this allows the assignment of human-name hostnames to access boards on the same subnet. For example. in the "TCP/IP Demo App" demonstration project, the demo board is programmed with the human name 'mchpboard' so it can be accessed directly instead of with its IP address.

File Name

nbns.h

Company

Microchip Technology Inc.

nbns_config.h

NBNS configuration file

Macros

	Name	Description
	TCPIP_NBNS_TASK_TICK_RATE	NBNS task processing rate The default value is 110 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

NetBIOS Name Service (NBNS) Configuration file

This file contains the NBNS module configuration options

File Name

nbns_config.h

Company

Microchip Technology Inc.

NDP Module

This section describes the TCP/IP Stack Library NDP module.

Introduction

TCP/IP Stack Library NDP Module for Microchip Microcontrollers

This library provides the API of the NDP (Neighbor Discovery Protocol) module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

This document provides NDP (Neighbor Discovery Protocol) for IP version 6. IPv6 nodes on the same link use neighbor discovery to discover each other's presence. RFC - 4861.

It is responsible for:

- Address Auto configuration of nodes

- Discovery of other nodes in the link (It can be a Router Discovery or Neighbor Discovery)
- Determining the link-layer address of other nodes
- Duplicate address detection (DAD)
- Finding available routers and Domain Name System (DNS) servers
- Address Prefix discovery
- Parameter Discovery (Such as Link MTU or Hop limit)

Comparing with IPv4:

- NDP is a substitute of ARP (Address Resolution protocol). This new mechanism uses a mix of ICMPv6 and multicast addresses to discover the IPv6 node on same link.
- NDP includes Neighbor Unreachability Detection (NUD) , thus improving the robustness of packet delivery.
- Unlike IPv4 broadcast addresses, IPv6 address resolution multicasts are spread over 4 billion (2^32) multicast addresses, greatly reducing address resolution-related interrupts on nodes other than the target. Moreover, non-IPv6 machines should not be interrupted at all.
- Neighbor Discovery defines five different ICMPv6 packet types.

Five Different ICMPv6 Packet Types:

- **Router Solicitation** - Hosts inquire with Router Solicitation message to locate routers on the attached link
- **Router Advertisement** - Router advertise their presence periodically to all the nodes or in response to the Router Solicitation message
- **Neighbor Solicitation** - Neighbor solicitations are used by nodes to determine the Link Layer address of a neighbor, or to verify that a neighbor is still reachable via a cached Link Layer address
- **Neighbor Advertisement** - Neighbor advertisements are used by nodes to respond to a Neighbor Solicitation message
- **Redirect** - Routers may inform hosts of a better first hop router for a destination

Using the Library

This topic describes the basic architecture of the NDP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `ndp.h`

The interface to the NDP TCP/IP Stack Library is defined in the `ndp.h` header file. Any C language source (`.c`) file that uses the NDP TCP/IP library should include `ipv6.h`, `icmpv6.h`, and `ndp.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack Library interacts with the framework.

Abstraction Model

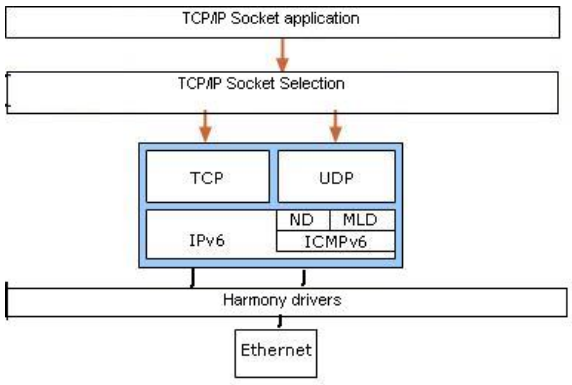
This library provides the API of the NDP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

NDP with IPv6 Software Abstraction Block Diagram

This module provides software abstraction of the IPv6 module existent in any TCP/IP Stack implementation. It removes the overhead of address resolution from all other modules in the stack.

IPv6 Block Diagram



Link Layer Neighbor Discovery

1) Using the address FEC0::1:0:0:1 :A, node A wants to deliver packets to destination node B using the IPv6 address FEC0::1 :0:0:1 :B on the

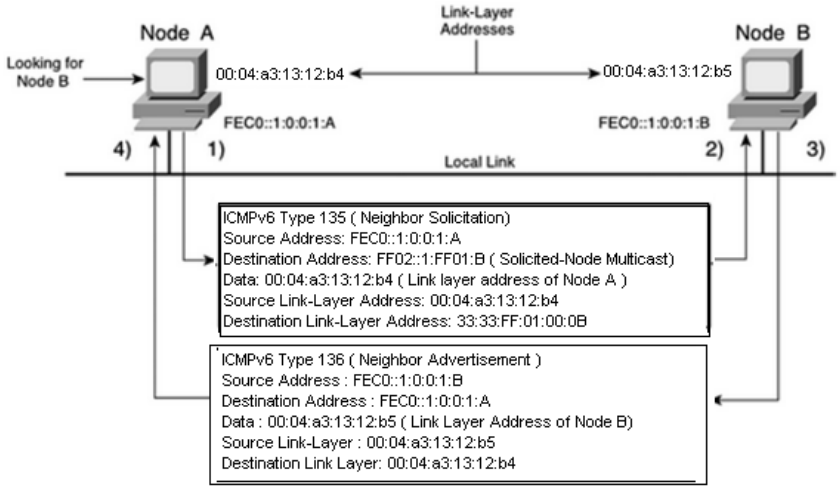
same local link. However, node A does not know node B's link-layer address. Node A sends an ICMPv6 Type

135 message (neighbor solicitation) on the local link using its site-local address FEC0::1:0:0:1:A as the IPv6 source address, the solicited-node multicast address FF02::1 :FF01:B corresponding to the target address FEC0::1 :0:0:1 :B as the destination IPv6 address, and the source link-layer address 00:04:a3:13:12:b4 of the sender, node A, as data of the ICMPv6 message. The source link-layer address of this frame is the link-layer address 00:04:a3:13:12:b4 of node A. The destination link-layer address 33:33:FF:01 :00:0B of this frame uses multicast mapping of the destination IPv6 address FF02::1 :FF01 :B.

2) Node B, which is listening to the local link for multicast addresses, intercepts the neighbor solicitation message because the destination IPv6 address FF02::1:FF01:B represents the solicited-node multicast address corresponding to its IPv6 address FEC0::1:0:0:1:B.

3) Node B replies by sending a neighbor advertisement message using its site-local address FEC0::1 :0:0:1 :B as the IPv6 source address and the site-local address FEC0::1 :0:0:1 :A as the destination IPv6 address. It also includes its link-layer address 00:04:a3:13:12:b5 in the ICMPv6 message. After receiving neighbor solicitation and neighbor advertisement messages, node A and node B know each other's link-layer addresses.

Learned link-layer addresses are kept in a neighbor discovery table (neighbor cache). Therefore, the nodes can communicate on the local link. The neighbor solicitation message is also used by nodes to verify the reachability of neighbor nodes in the neighbor discovery table (neighbor cache). However, the unicast addresses of the neighbor nodes are used as destination IPv6 addresses in ICMPv6 messages instead of solicited-node multicast addresses in this situation. It is possible for a node that changes its link-layer address to inform all other neighbor nodes on the local link by sending a neighbor advertisement message using the all-nodes multicast address FF02::1 . The neighbor discovery table of the nodes on the local link is updated with the new link-layer address.



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the IPv6, ICMPv6, and NDP module.

Library Interface Section	Description
Reachability Functions	Reachability routine functionality

Configuring the Library

Macros

	Name	Description
	TCPIP_IPV6_MTU_INCREASE_TIMEOUT	600 seconds
	TCPIP_IPV6_NDP_DELAY_FIRST_PROBE_TIME	5 s
	TCPIP_IPV6_NDP_MAX_ANYCAST_DELAY_TIME	1 s
	TCPIP_IPV6_NDP_MAX_MULTICAST_SOLICIT	3 transmissions
	TCPIP_IPV6_NDP_MAX_NEIGHBOR_ADVERTISEMENT	3 transmissions
	TCPIP_IPV6_NDP_MAX_RTR_SOLICITATION_DELAY	1 s
	TCPIP_IPV6_NDP_MAX_RTR_SOLICITATIONS	3 transmissions
	TCPIP_IPV6_NDP_MAX_UNICAST_SOLICIT	3 transmissions
	TCPIP_IPV6_NDP_REACHABLE_TIME	30 s
	TCPIP_IPV6_NDP_RETRANS_TIMER	1 s
	TCPIP_IPV6_NDP_RTR_SOLICITATION_INTERVAL	4 s

	TCPIP_IPV6_NDP_TASK_TIMER_RATE	The NDP task rate, milliseconds The default value is 32 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_IPV6_NDP_VALID_LIFETIME_TWO_HOURS	Sets the lifetime to 2 hours

Description

The configuration of the TCP/IP Stack NDP module is based on the file `ndp_config.h`.

This header file contains the configuration selection for the TCP/IP Stack IP. Based on the selections made, the TCP/IP Stack IP may support the selected features. These configuration settings will apply to all instances of the TCP/IP Stack IP.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build.

TCPIP_IPV6_MTU_INCREASE_TIMEOUT Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_MTU_INCREASE_TIMEOUT 600ul           // 600 seconds
```

Description

600 seconds

TCPIP_IPV6_NDP_DELAY_FIRST_PROBE_TIME Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_DELAY_FIRST_PROBE_TIME 5u              // 5 s
```

Description

5 s

TCPIP_IPV6_NDP_MAX_ANYCAST_DELAY_TIME Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_MAX_ANYCAST_DELAY_TIME 1u              // 1 s
```

Description

1 s

TCPIP_IPV6_NDP_MAX_MULTICAST_SOLICIT Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_MAX_MULTICAST_SOLICIT 3u             // 3 transmissions
```

Description

3 transmissions

TCPIP_IPV6_NDP_MAX_NEIGHBOR_ADVERTISEMENT Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_MAX_NEIGHBOR_ADVERTISEMENT 3u            // 3 transmissions
```

Description

3 transmissions

TCPIP_IPV6_NDP_MAX_RTR_SOLICITATION_DELAY Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_MAX_RTR_SOLICITATION_DELAY 1u            // 1 s
```

Description

1 s

TCPIP_IPV6_NDP_MAX_RTR_SOLICITATIONS Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_MAX_RTR_SOLICITATIONS 3u             // 3 transmissions
```

Description

3 transmissions

TCPIP_IPV6_NDP_MAX_UNICAST_SOLICIT Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_MAX_UNICAST_SOLICIT 3u             // 3 transmissions
```

Description

3 transmissions

TCPIP_IPV6_NDP_REACHABLE_TIME Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_REACHABLE_TIME 30u            // 30 s
```

Description

30 s

TCPIP_IPV6_NDP_RETRANS_TIMER Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_RETRANS_TIMER 1u             // 1 s
```

Description

1 s

TCPIP_IPV6_NDP_RTR_SOLICITATION_INTERVAL Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_RTR_SOLICITATION_INTERVAL 4u             // 4 s
```

Description

4 s

TCPIP_IPV6_NDP_TASK_TIMER_RATE Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_TASK_TIMER_RATE (32)
```

Description

The NDP task rate, milliseconds The default value is 32 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_IPV6_NDP_VALID_LIFETIME_TWO_HOURS Macro

File

ndp_config.h

C

```
#define TCPIP_IPV6_NDP_VALID_LIFETIME_TWO_HOURS (60 * 60 * 2)
```

Description

Sets the lifetime to 2 hours

Building the Library

This section lists the files that are available in the NDP module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/ndp.c	NDP implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The NDP module depends on the following modules:

- TCP/IP Stack Library
- IPv6 Module

Library Interface

Reachability Functions

	Name	Description
	TCPIP_NDP_NborReachConfirm	Confirms that a neighbor is reachable.

Description

This section describes the Application Programming Interface (API) functions of the NDP module.

Refer to each section for a detailed description.

Reachability Functions

TCPIP_NDP_NborReachConfirm Function

Confirms that a neighbor is reachable.

File

ndp.h

C

```
void TCPIP_NDP_NborReachConfirm(TCPIP_NET_HANDLE netH, const IPV6_ADDR * address);
```

Returns

None.

Description

This function is used by upper-layer protocols to indicate that round-trip communications were confirmed with a neighboring node.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
pNetIf	The interface the neighbor is on.
address	The address of the neighbor.

Function

void TCPIP_NDP_NborReachConfirm (TCPIP_NET_HANDLE netH, IPV6_ADDR * address)

Files

Files

Name	Description
ndp.h	IPv6 Internet Communication Message Neighbor Discovery Protocol (NDP).
ndp_config.h	NDP configuration file

Description

This section lists the source and header files used by the library.

ndp.h

IPv6 Internet Communication Message Neighbor Discovery Protocol (NDP).

Functions

	Name	Description
	TCPIP_NDP_NborReachConfirm	Confirms that a neighbor is reachable.

Description

Neighbor Discovery Protocol (NDP) API Header File

Neighbor Discovery Protocol (NDP) in IPv6 is the substitute of as ARP (which is used in IPv4 for address resolve). NDP is used discover link local addresses of the IPv6 nodes present in the local link using a mix of ICMPv6 messages and multicast addresses, stateless auto-configuration and router redirection.

File Name

ndp.h

Company

Microchip Technology Inc.

ndp_config.h

NDP configuration file

Macros

	Name	Description
	TCPIP_IPV6_MTU_INCREASE_TIMEOUT	600 seconds
	TCPIP_IPV6_NDP_DELAY_FIRST_PROBE_TIME	5 s
	TCPIP_IPV6_NDP_MAX_ANYCAST_DELAY_TIME	1 s
	TCPIP_IPV6_NDP_MAX_MULTICAST_SOLICIT	3 transmissions
	TCPIP_IPV6_NDP_MAX_NEIGHBOR_ADVERTISEMENT	3 transmissions
	TCPIP_IPV6_NDP_MAX_RTR_SOLICITATION_DELAY	1 s
	TCPIP_IPV6_NDP_MAX_RTR_SOLICITATIONS	3 transmissions
	TCPIP_IPV6_NDP_MAX_UNICAST_SOLICIT	3 transmissions
	TCPIP_IPV6_NDP_REACHABLE_TIME	30 s
	TCPIP_IPV6_NDP_RETRANS_TIMER	1 s
	TCPIP_IPV6_NDP_RTR_SOLICITATION_INTERVAL	4 s

	TCPIP_IPV6_NDP_TASK_TIMER_RATE	The NDP task rate, milliseconds The default value is 32 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_IPV6_NDP_VALID_LIFETIME_TWO_HOURS	Sets the lifetime to 2 hours

Description

Neighbor Discovery Protocol (NDP) Configuration file

This file contains the NDP module configuration options

File Name

ndp_config.h

Company

Microchip Technology Inc.

Reboot Module

This section describes the TCP/IP Stack Library Reboot module.

Introduction

TCP/IP Stack Library Reboot Module for Microchip Microcontrollers

This library provides the API of the Reboot module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The Reboot module will allow a user to remotely reboot the PIC microcontroller that is running the TCP/IP stack. This feature is primarily used for bootloader applications, which must reset the microcontroller to enter the bootloader code section. This module will execute a task that listens on a specified UDP port for a packet, and then reboots if it receives one.

Using the Library

This topic describes the basic architecture of the Reboot TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `tcpip_reboot_config.h`

The interface to the Reboot TCP/IP Stack library is defined in the `tcpip_reboot_config.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the Reboot TCP/IP Stack library should include `tcpip.h`.

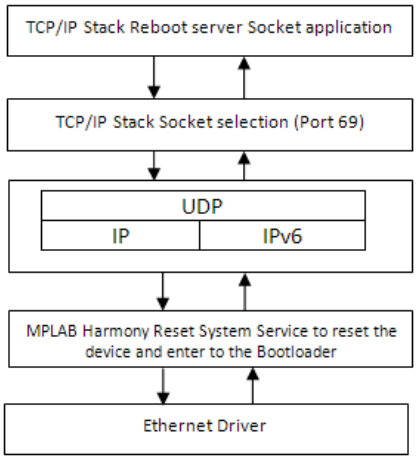
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the Reboot TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Reboot Software Abstraction Block Diagram



Configuring the Library

Macros

	Name	Description
	TCPIP_REBOOT_MESSAGE	the mesage needed to be sent accross the net to reboot the machine
	TCPIP_REBOOT_SAME_SUBNET_ONLY	For improved security, you might want to limit reboot capabilities to only users on the same IP subnet. Define TCPIP_REBOOT_SAME_SUBNET_ONLY to enable this access restriction.
	TCPIP_REBOOT_TASK_TICK_RATE	the periodic rate of the Reboot task The default value is 1130 milliseconds. This module listens for incoming reboot requests and a high operation frequency is not required. The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

The configuration of the Reboot TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the Reboot TCP/IP Stack Library. Based on the selections made, the Reboot TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the Reboot TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_REBOOT_MESSAGE Macro

File

tcpip_reboot_config.h

C

```
#define TCPIP_REBOOT_MESSAGE "MCHP Reboot"
```

Description

the mesage needed to be sent accross the net to reboot the machine

TCPIP_REBOOT_SAME_SUBNET_ONLY Macro

File

tcpip_reboot_config.h

C

```
#define TCPIP_REBOOT_SAME_SUBNET_ONLY
```

Description

For improved security, you might want to limit reboot capabilities to only users on the same IP subnet. Define TCPIP_REBOOT_SAME_SUBNET_ONLY to enable this access restriction.

TCPIP_REBOOT_TASK_TICK_RATE Macro

File

tcpip_reboot_config.h

C

```
#define TCPIP_REBOOT_TASK_TICK_RATE 1130
```

Description

the periodic rate of the Reboot task The default value is 1130 milliseconds. This module listens for incoming reboot requests and a high operation frequency is not required. The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Building the Library

This section lists the files that are available in the Reboot module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/tcpip_reboot.c`	Reboot implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Reboot module depends on the following modules:

- TCP/IP Stack Library
- UDP Module

Files

Files

Name	Description
tcpip_reboot_config.h	Configuration file

Description

This section lists the source and header files used by the library.

tcpip_reboot_config.h

Configuration file

Macros

	Name	Description
	TCPIP_REBOOT_MESSAGE	the mesage needed to be sent accross the net to reboot the machine
	TCPIP_REBOOT_SAME_SUBNET_ONLY	For improved security, you might want to limit reboot capabilities to only users on the same IP subnet. Define TCPIP_REBOOT_SAME_SUBNET_ONLY to enable this access restriction.
	TCPIP_REBOOT_TASK_TICK_RATE	the periodic rate of the Reboot task The default value is 1130 milliseconds. This module listens for incoming reboot requests and a high operation frequency is not required. The value cannot be lower than the TCPIP_STACK_TICK_RATE.

Description

Reboot Configuration file

This file contains the Reboot module configuration options

File Name

tcpip_reboot_config.h

Company

Microchip Technology Inc.

SMTP Module

This section describes the TCP/IP Stack Library SMTP module.

Introduction

TCP/IP Stack Library Simple Mail Transfer Protocol (SMTP) Module for Microchip Microcontrollers

This library provides the API of the SMTP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The SMTP client module in the TCP/IP Stack lets applications send e-mails to any recipient worldwide. These message could include status information or important alerts. Using the e-mail to SMS gateways provided by most cell phone carriers, these messages can also be delivered directly to cell phone handsets.

Using the SMTP client requires access to a local mail server (such as `mail.yourdomain.com`) for reliable operation. Your MPLAB Harmony or network administrator can provide the correct address, but end-user applications will need an interface to provide this data.

Using the Library

This topic describes the basic architecture of the SMTP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `smtp.h`

The interface to the SMTP TCP/IP Stack library is defined in the `smtp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the SMTP TCP/IP Stack library should include `tcpip.h`.

Library File:

The SMTP TCP/IP Stack library archive (`.a`) file is installed with MPLAB Harmony.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the SMTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

SMTP Software Abstraction Block Diagram

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SMTP module.

Library Interface Section	Description
Functions	Routines to configure this module
Data Types and Constants	This section provides various definitions describing this API

SMTP Client Examples

SMTP Client examples.

Description

The following two examples demonstrate the use of the SMTP client in different scenarios. The first, and simpler example, sends a short message whose contents are all located in RAM at once.

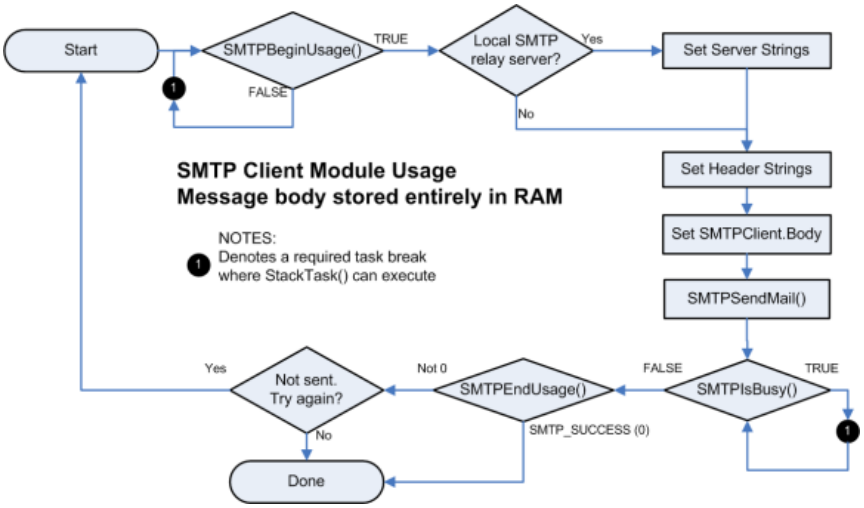
The second example is more involved and demonstrates generating a message on the fly in the case where the entire message cannot fit into RAM at once. The SMTPDemo example provided in `MainDemo.c` sends a brief e-mail message indicating the current status of the board's buttons.

SMTP Client Short Message Example

The SMTP client API is simplified when messages can be buffered entirely in RAM. The SMTPDemo example provided in `MainDemo.c` sends a brief e-mail message indicating the current status of the board's buttons. This document will walk through that example.

Make sure STACK_USE_SMTP_CLIENT is uncommented in `tcpip_config.h` before continuing.

The following diagram provides an overview of the process:



1. First, call SMTPBeginUsage to verify that the SMTP client is available and to begin a new message. If FALSE is returned, the SMTP client is busy and the application must return to the main loop to allow StackTask to execute again.
2. Next, set the local relay server to use as SMTPClient.Server. If the local relay server requires a user name and password, set SMTPClient.Username and SMTPClient.Password to the appropriate credentials. If server parameters are not set, the stack will attempt to deliver the message directly to its destination host. This will likely fail due to SPAM prevention measures put in place by most ISPs and network administrators.
3. Continue on to set the header strings as necessary for the message. This includes the subject line, from address, and any recipients you need

to add. Finally, set SMTPClient.Body to the message to be sent.

4. At this point, verify that SMTPClient.ROMPointers is correctly configured for any strings that are stored in program memory. Once the message is ready to send, call SMTPSendMail to instruct the SMTP client to begin transmission.
5. The application must now call SMTPIsBusy until it returns FALSE. Each time TRUE is returned, return to the main loop and wait for StackTask to execute again. This allows the SMTP server to continue its work in a cooperative multitasking manner. Once FALSE is returned, call SMTPEndUsage to release the SMTP client. Check the return value of this function to determine if the message was successfully sent.

The example in `MainDemo.c` needs minor modifications to use your e-mail address. The Server and To fields must be set in SMTPDemo in order for the message to be properly delivered. Once this is done, holding down BUTTON2 and BUTTON3 simultaneously (the left-most two buttons) will begin sending the message. LED1 will light as the message is being processed, and will extinguish when the SMTP state machine completes. If the transmission was successful LED2 will light, otherwise it will remain dark.

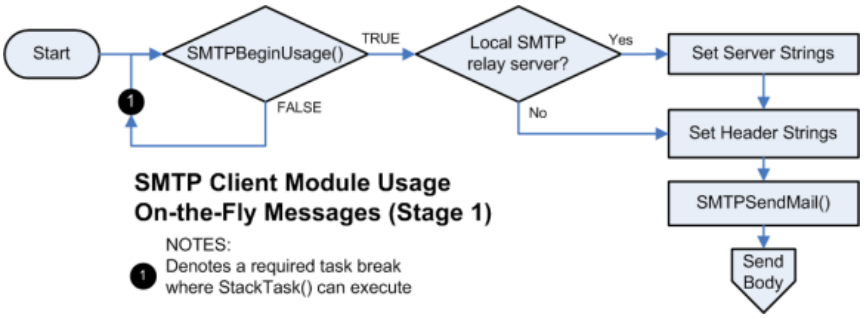
SMTP Client Long Message Example

The SMTP client API is capable of sending messages that do not fit entirely in RAM. To do so, the application must manage its output state and only write as many bytes as are available in the buffer at a time. The second SMTPDemo example provided in `MainDemo.c` sends a message that is a dump of all contents of the PIC microcontroller's RAM. This example is currently commented out. Comment out the previous Short Message Example and uncomment the Long Message Example. This document will walk through sending a longer message.

Make sure STACK_USE_SMTP_CLIENT is uncommented in `tcpip_config.h` before continuing.

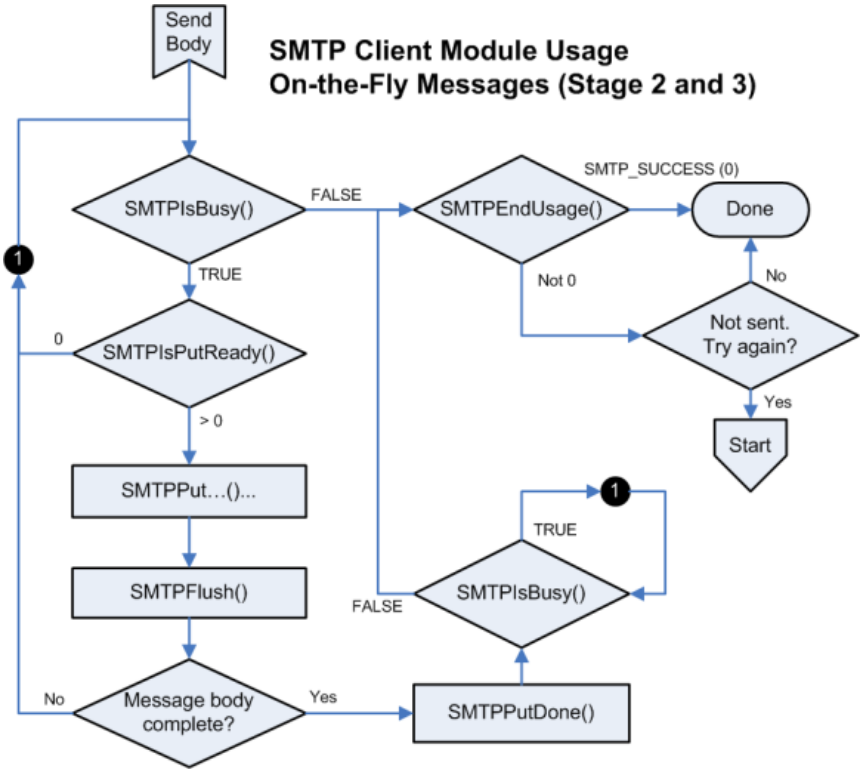
Sending longer messages is divided into three stages. The first stage configures the SMTP client to send the message. The second stage sends the message in small chunks as buffer space is available. The final stage finishes the transmission and determines whether or not the message was successful.

The following diagram illustrates the first stage:



1. The first stage is largely similar to the first few steps in sending a short message. First, call SMTPBeginUsage to verify that the SMTP client is available and to begin a new message. If FALSE is returned, the SMTP client is busy and the application must return to the main loop to allow StackTask to execute again.
2. Next, set the local relay server to use as SMTPClient.Server. If the local relay server requires a user name and password, set SMTPClient.Username and SMTPClient.Password to the appropriate credentials. If server parameters are not set, the stack will attempt to deliver the message directly to its destination host. This will likely fail due to SPAM prevention measures put in place by most ISPs and network administrators.
3. Continue on to set the header strings as necessary for the message. This includes the subject line, from address, and any recipients you need to add.
4. The next portion of the process differs. Ensure that SMTPClient.Body remains set to its default (NULL). At this point, call SMTPSendMail to open a connection to the remote server and transmit the headers. The application is now ready to proceed to the second stage and send the message body.

The following diagram provides an overview of stage two and three:



1. Upon entering stage two, the application should call SMTPIsBusy to verify that the connection to the remote server is active and has not been lost. If the call succeeds, call SMTPIsPutReady to determine how many bytes are available in the TX buffer. If no bytes are available, return to the main loop so that StackTask can transmit the data to the remote node and free up the buffer.
2. If space is available, any combination of the SMTPPut, SMTPPutArray, SMTPPutROMArray, SMTPPutString, and SMTPPutROMString functions may be called to transmit the message. These functions return the number of bytes successfully written. Use this value, along with the value originally returned from SMTPIsPutReady to track how much free space remains in the TX buffer. Once the buffer is depleted, call SMTPFlush to force the data written to be sent.
3. The SMTP client module can accept as much data as the TCP TX FIFO can hold. This is determined by the socket initializer for TCP_PURPOSE_DEFAULT type sockets in `tcpip_config.h`, which defaults to 200 bytes.
4. If the TX buffer is exhausted before the message is complete, return to the main loop so that StackTask may transmit the data to the remote node and free up the buffer. Upon return, go to the beginning of the second stage to transmit the next portion of the message.

Once the message is complete, the application will move to the third stage. Call SMTPPutDone to inform the SMTP client that no more data remains. Then call SMTPIsBusy repeatedly. Each time TRUE is returned, return to the main loop and wait for StackTask to execute again. Once FALSE is returned, the message transmission has completed and the application must call SMTPEndUsage to release the SMTP client. Check the return value of this function to determine if the message was successfully sent.

The example in `MainDemo.c` needs minor modifications to use your e-mail address. Set the Server and To fields in SMTPDemo, and ensure that these fields are being properly assigned to SMTPClient struct. The demonstration works exactly the same way as the previous one, with BUTTON2 and BUTTON3 held down simultaneously (the left-most two buttons) kicking off the state machine. LED1 will light as the message is being processed, and will extinguish when the SMTP state machine completes. If the transmission was successful LED2 will turn ON; otherwise, it will remain OFF.

Configuring the Library

Macros

	Name	Description
	TCPIP_SMTP_SERVER_REPLY_TIMEOUT	How long to wait before assuming the connection has been dropped (default 8 seconds)
	TCPIP_SMTP_TASK_TICK_RATE	SMTP task rate, milliseconds The default value is 55 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_SMTP_WRITE_READY_SPACE	the minimum amount of data to ask from the transport/encryption layer when querying the write space
	TCPIP_SMTP_MAX_WRITE_SIZE	the max size of data to be written in a discrete string/array email operation: TCPIP_SMTP_StringPut/TCPIP_SMTP_ArrayPut. Excess characters will be discarded. Note that the higher this value, the greater the size of the underlying TCP socket TX buffer Adjust as needed. Normally should not exceed 512 characters.

Description

The configuration of the SMPT TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the SMTP TCP/IP Stack Library. Based on the selections made, the SMTP TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the SMTP TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_SMTP_SERVER_REPLY_TIMEOUT Macro

File

smtp_config.h

C

```
#define TCPIP_SMTP_SERVER_REPLY_TIMEOUT (8)
```

Description

How long to wait before assuming the connection has been dropped (default 8 seconds)

TCPIP_SMTP_TASK_TICK_RATE Macro

File

smtp_config.h

C

```
#define TCPIP_SMTP_TASK_TICK_RATE (55)
```

Description

SMTP task rate, milliseconds The default value is 55 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_SMTP_WRITE_READY_SPACE Macro

File

smtp_config.h

C

```
#define TCPIP_SMTP_WRITE_READY_SPACE 150
```

Description

the minimum amount of data to ask from the transport/encryption layer when querying the write space

TCPIP_SMTP_MAX_WRITE_SIZE Macro

File

smtp_config.h

C

```
#define TCPIP_SMTP_MAX_WRITE_SIZE 512
```

Description

the max size of data to be written in a discrete string/array email operation: TCPIP_SMTP_StringPut/TCPIP_SMTP_ArrayPut. Excess characters will be discarded. Note that the higher this value, the greater the size of the underlying TCP socket TX buffer Adjust as needed. Normally should not exceed 512 characters.

Building the Library

This section lists the files that are available in the SMTP module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/smtp.c`	SMTP implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The SMTP module depends on the following modules:

- TCP/IP Stack Library
- DNS Module
- TCP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_SMTP_ArrayPut	Writes a series of bytes to the SMTP client.
	TCPIP_SMTP_Flush	Flushes the SMTP socket and forces all data to be sent.
	TCPIP_SMTP_IsBusy	Determines if the SMTP client is busy.
	TCPIP_SMTP_IsPutReady	Determines how much data can be written to the SMTP client.
	TCPIP_SMTP_Put	Writes a single byte to the SMTP client.
	TCPIP_SMTP_PutIsDone	Indicates that the on-the-fly message is complete.
	TCPIP_SMTP_StringPut	Writes a string to the SMTP client.
	TCPIP_SMTP_UsageBegin	Requests control of the SMTP client module.
	TCPIP_SMTP_UsageEnd	Releases control of the SMTP client module.
	TCPIP_SMTP_MailSend	Initializes the message sending process.
	TCPIP_SMTP_ClientTask	Standard TCP/IP stack module task function.

b) Data Types and Functions

	Name	Description
	SMTP_CONNECT_ERROR	Connection to SMTP server failed
	SMTP_RESOLVE_ERROR	DNS lookup for SMTP server failed
	SMTP_SUCCESS	Message was successfully sent
	TCPIP_SMTP_CLIENT_MODULE_CONFIG	This is type TCPIP_SMTP_CLIENT_MODULE_CONFIG.
	TCPIP_SMTP_CLIENT_MESSAGE	Configures the SMTP client to send a message.

Description

This section describes the Application Programming Interface (API) functions of the SMTP module.

Refer to each section for a detailed description.

a) Functions

TCPIP_SMTP_ArrayPut Function

Writes a series of bytes to the SMTP client.

File

smtp.h

C

```
uint16_t TCPIP_SMTP_ArrayPut(uint8_t* Data, uint16_t Len);
```

Returns

The number of bytes written. If less than Len, then the TX FIFO became full before all bytes could be written.

Description

This function writes a series of bytes to the SMTP client.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message (i.e., TCPIP_SMTP_MailSend was called with SMTPClient.Body set to NULL).

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Parameters

Parameters	Description
Data	The data to be written
Len	How many bytes should be written

Function

uint16_t TCPIP_SMTP_ArrayPut(uint8_t* Data, uint16_t Len)

TCPIP_SMTP_Flush Function

Flushes the SMTP socket and forces all data to be sent.

File

smtp.h

C

```
void TCPIP_SMTP_Flush();
```

Returns

None.

Description

This function flushes the SMTP socket and forces all data to be sent.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message (i.e., TCPIP_SMTP_MailSend was called with SMTPClient.Body set to NULL).

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Function

void TCPIP_SMTP_Flush(void)

TCPIP_SMTP_IsBusy Function

Determines if the SMTP client is busy.

File

smtp.h

C

```
bool TCPIP_SMTP_IsBusy();
```

Description

Call this function to determine if the SMTP client is busy performing background tasks. This function should be called after any call to TCPIP_SMTP_MailSend, TCPIP_SMTP_PutIsDone to determine if the stack has finished performing its internal tasks. It should also be called prior to any call to TCPIP_SMTP_IsPutReady to verify that the SMTP client has not prematurely disconnected. When this function returns false, the next call should be to TCPIP_SMTP_UsageEnd to release the module and obtain the status code for the operation.

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Function

bool TCPIP_SMTP_IsBusy(void)

TCPIP_SMTP_IsPutReady Function

Determines how much data can be written to the SMTP client.

File

smtp.h

C

```
uint16_t TCPIP_SMTP_IsPutReady();
```

Returns

The number of free bytes the SMTP TX FIFO.

Description

Use this function to determine how much data can be written to the SMTP client when generating an on-the-fly message.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message (i.e., TCPIP_SMTP_MailSend was called with SMTPClient.Body set to NULL).

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call, and an on-the-fly message is being generated. This requires that TCPIP_SMTP_MailSend was called with SMTPClient.Body set to NULL.

Function

uint16_t TCPIP_SMTP_IsPutReady(void)

TCPIP_SMTP_Put Function

Writes a single byte to the SMTP client.

File

smtp.h

C

```
bool TCPIP_SMTP_Put(char c);
```

Description

This function writes a single byte to the SMTP client.

Remarks

This function is obsolete and will be eventually removed. TCPIP_SMTP_ArrayPut and TCPIP_SMTP_StringPut should be used.

This function cannot be used on an encrypted connection. It is difficult to estimate the amount of TX buffer space needed when transmitting byte by byte, which could cause intermediary write operations to the underlying TCP socket.

This function should only be called externally when the SMTP client is generating an on-the-fly message (i.e., TCPIP_SMTP_MailSend was called with SMTPClient.Body set to NULL).

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Parameters

Parameters	Description
c	The byte to be written

Function

bool TCPIP_SMTP_Put(char c)

TCPIP_SMTP_PutIsDone Function

Indicates that the on-the-fly message is complete.

File

smtp.h

C

```
void TCPIP_SMTP_PutIsDone();
```

Returns

None.

Description

This function indicates that the on-the-fly message is complete.

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call, and the SMTP client is generated an on-the-fly message (i.e., TCPIP_SMTP_MailSend was called with SMTPClient.Body set to NULL).

Function

void TCPIP_SMTP_PutIsDone(void)

TCPIP_SMTP_StringPut Function

Writes a string to the SMTP client.

File

smtp.h

C

```
uint16_t TCPIP_SMTP_StringPut(char* Data);
```

Returns

The number of bytes written. If less than the length of Data, then the TX FIFO became full before all bytes could be written.

Description

This function writes a string to the SMTP client.

Remarks

This function should only be called externally when the SMTP client is generating an on-the-fly message. (That is, TCPIP_SMTP_MailSend was

called with SMTPClient.Body set to NULL.)

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Parameters

Parameters	Description
Data	The data to be written

Function

uint16_t TCPIP_SMTP_StringPut(char* Data)

TCPIP_SMTP_UsageBegin Function

Requests control of the SMTP client module.

File

smtp.h

C

```
bool TCPIP_SMTP_UsageBegin();
```

Description

Call this function before calling any other SMTP Client APIs. This function obtains a lock on the SMTP Client, which can only be used by one stack application at a time. Once the application is finished with the SMTP client, it must call TCPIP_SMTP_UsageEnd to release control of the module to any other waiting applications.

This function initializes all the SMTP state machines and variables back to their default state.

Preconditions

None.

Function

bool TCPIP_SMTP_UsageBegin(void)

TCPIP_SMTP_UsageEnd Function

Releases control of the SMTP client module.

File

smtp.h

C

```
uint16_t TCPIP_SMTP_UsageEnd();
```

Description

Call this function to release control of the SMTP client module once an application is finished using it. This function releases the lock obtained by TCPIP_SMTP_UsageBegin, and frees the SMTP client to be used by another application.

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Function

uint16_t TCPIP_SMTP_UsageEnd(void)

TCPIP_SMTP_MailSend Function

Initializes the message sending process.

File

smtp.h

C

```
void TCPIP_SMTP_MailSend(TCPIP_SMTP_CLIENT_MESSAGE* smtpClientMessage);
```

Returns

None.

Description

This function starts the state machine that performs the actual transmission of the message. Call this function after all the fields in SMTPClient have been set.

Remarks

The fields pointed by the smtpClientMessage have to be non-volatile until the SMTP send mail process is completed!

Preconditions

TCPIP_SMTP_UsageBegin returned true on a previous call.

Parameters

Parameters	Description
smtpClientMessage	pointer to a TCPIP_SMTP_CLIENT_MESSAGE structure that configures the message to send

Function

void TCPIP_SMTP_MailSend(TCPIP_SMTP_CLIENT_MESSAGE* smtpClientMessage)

TCPIP_SMTP_ClientTask Function

Standard TCP/IP stack module task function.

File

smtp.h

C

```
void TCPIP_SMTP_ClientTask();
```

Returns

None.

Description

this function performs SMTP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

SMTP module should have been initialized.

Function

void TCPIP_SMTP_ClientTask(void)

b) Data Types and Functions

SMTP_CONNECT_ERROR Macro

File

smtp.h

C

```
#define SMTP_CONNECT_ERROR (0x8001u)    // Connection to SMTP server failed
```

Description

Connection to SMTP server failed

SMTP_RESOLVE_ERROR Macro

File

smtp.h

C

```
#define SMTP_RESOLVE_ERROR (0x8000u)    // DNS lookup for SMTP server failed
```

Description

DNS lookup for SMTP server failed

SMTP_SUCCESS Macro

File

smtp.h

C

```
#define SMTP_SUCCESS (0x0000u)    // Message was successfully sent
```

Description

Message was successfully sent

TCPIP_SMTP_CLIENT_MODULE_CONFIG Structure

File

smtp.h

C

```
typedef struct {
} TCPIP_SMTP_CLIENT_MODULE_CONFIG;
```

Description

This is type TCPIP_SMTP_CLIENT_MODULE_CONFIG.

TCPIP_SMTP_CLIENT_MESSAGE Structure

Configures the SMTP client to send a message.

File

smtp.h

C

```
typedef struct {
  char* Server;
  char* Username;
  char* Password;
  char* To;
  char* CC;
  char* BCC;
  char* From;
  char* Subject;
  char* OtherHeaders;
  char* Body;
  bool UseSSL;
  uint16_t ServerPort;
} TCPIP_SMTP_CLIENT_MESSAGE;
```

Description

This structure of pointers configures the SMTP Client to send an e-mail message.

Remarks

When formatting an e-mail address, the SMTP standard format for associating a printable name may be used. This format places the printable name in quotation marks, with the address following in pointed brackets, such as "John Smith" .

Parameters

Parameters	Description
Server	the SMTP server to relay the message through
Username	the user name to use when logging into the SMTP server, if any is required
Password	the password to supply when logging in, if any is required
To	the destination address for this message. May be a comma-separated list of address, and/or formatted.
CC	The CC addresses for this message, if any. May be a comma-separated list of address, and/or formatted.
BCC	The BCC addresses for this message, if any. May be a comma-separated list of address, and/or formatted.
From	The From address for this message. May be formatted.
Subject	The Subject header for this message.
OtherHeaders	Any additional headers for this message. Each additional header, including the last one, must be terminated with a CRLF pair.
Body	When sending a message from memory, the location of the body of this message in memory. Leave as NULL to build a message on-the-fly.
UseSSL	This flag causes the SMTP client to make a secure connection to the server.
ServerPort	(uint16_t value) Indicates the port on which to connect to the remote SMTP server.

Function

typedef struct TCPIP_SMTP_CLIENT_MESSAGE

Files

Files

Name	Description
smtp.h	Module for Microchip TCP/IP Stack.
smtp_config.h	SMTP configuration file

Description

This section lists the source and header files used by the library.

smtp.h

Module for Microchip TCP/IP Stack.

Functions

	Name	Description
	TCPIP_SMTP_ArrayPut	Writes a series of bytes to the SMTP client.
	TCPIP_SMTP_ClientTask	Standard TCP/IP stack module task function.
	TCPIP_SMTP_Flush	Flushes the SMTP socket and forces all data to be sent.
	TCPIP_SMTP_IsBusy	Determines if the SMTP client is busy.
	TCPIP_SMTP_IsPutReady	Determines how much data can be written to the SMTP client.
	TCPIP_SMTP_MailSend	Initializes the message sending process.
	TCPIP_SMTP_Put	Writes a single byte to the SMTP client.
	TCPIP_SMTP_PutIsDone	Indicates that the on-the-fly message is complete.
	TCPIP_SMTP_StringPut	Writes a string to the SMTP client.
	TCPIP_SMTP_UsageBegin	Requests control of the SMTP client module.

	TCPIP_SMTP_UsageEnd	Releases control of the SMTP client module.

Macros

	Name	Description
	SMTP_CONNECT_ERROR	Connection to SMTP server failed
	SMTP_RESOLVE_ERROR	DNS lookup for SMTP server failed
	SMTP_SUCCESS	Message was successfully sent

Structures

	Name	Description
	TCPIP_SMTP_CLIENT_MESSAGE	Configures the SMTP client to send a message.
	TCPIP_SMTP_CLIENT_MODULE_CONFIG	This is type TCPIP_SMTP_CLIENT_MODULE_CONFIG.

Description

Simple Mail Transfer Protocol (SMTP) Client

The SMTP client module in the TCP/IP Stack lets applications send e-mails to any recipient worldwide. These message could include status information or important alerts. Using the e-mail to SMS gateways provided by most cell phone carriers, these messages can also be delivered directly to cell phone handsets.

File Name

smtp.h

Company

Microchip Technology Inc.

smtp_config.h

SMTP configuration file

Macros

	Name	Description
	TCPIP_SMTP_MAX_WRITE_SIZE	the max size of data to be written in a discrete string/array email operation: TCPIP_SMTP_StringPut/TCPIP_SMTP_ArrayPut. Excess characters will be discarded. Note that the higher this value, the greater the size of the underlying TCP socket TX buffer Adjust as needed. Normally should not exceed 512 characters.
	TCPIP_SMTP_SERVER_REPLY_TIMEOUT	How long to wait before assuming the connection has been dropped (default 8 seconds)
	TCPIP_SMTP_TASK_TICK_RATE	SMTP task rate, milliseconds The default value is 55 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_SMTP_WRITE_READY_SPACE	the minimum amount of data to ask from the transport/encryption layer when querying the write space

Description

Simple Mail Transfer Protocol (SMTP) Configuration file

This file contains the SMTP module configuration options

File Name

smtp_config.h

Company

Microchip Technology Inc.

SMTPC Module

This section describes the TCP/IP Stack Library SMTPC module.

Introduction

TCP/IP Stack Library Simple Mail Transfer Protocol Client (SMTPC) Module for Microchip Microcontrollers

This library provides the API of the SMTPC Client module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The SMTPC module in the TCP/IP Stack lets applications send e-mails to any recipient worldwide. These message could include status information or important alerts. Using the e-mail to SMS gateways provided by most cell phone carriers, these messages can also be delivered directly to cell phone handsets.

Using the SMTPC client requires access to a local mail server (such as `mail.yourdomain.com`) for reliable operation. Your MPLAB Harmony or network administrator can provide the correct address, but end-user applications will need an interface to provide this data.

Using the Library

This topic describes the basic architecture of the SMTPC TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `smtpc.h`

The interface to the SMTPC TCP/IP Stack library is defined in the `smtpc.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the SMTPC TCP/IP Stack library should include `tcpip.h`.

Library File:

The SMTPC TCP/IP Stack library archive (`.a`) file is installed with MPLAB Harmony.

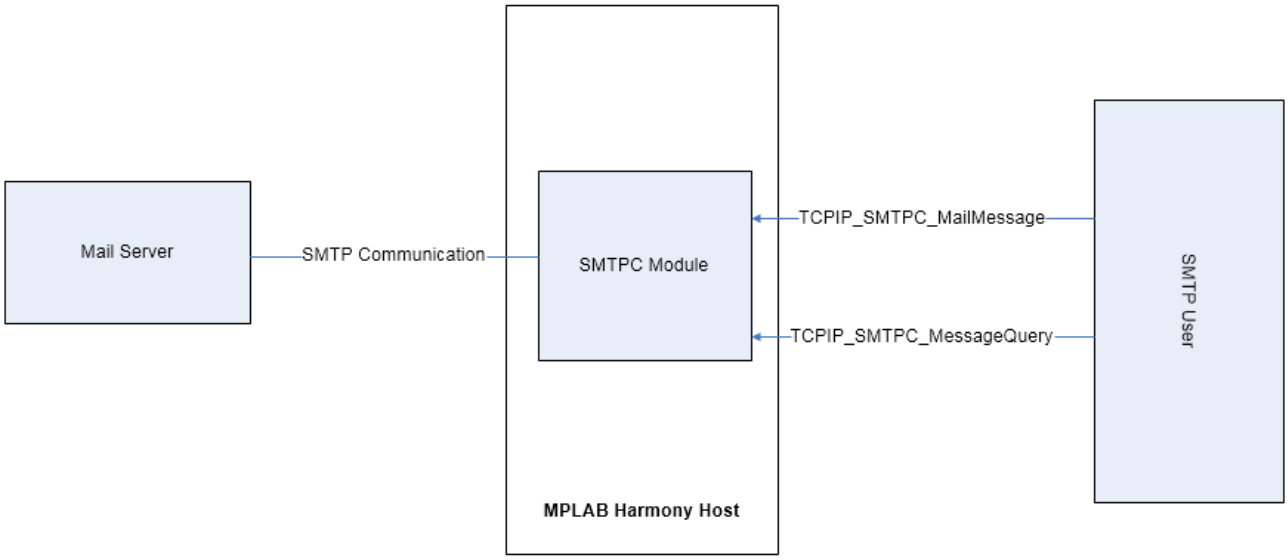
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the SMTPC TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

SMTPC Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SMTPC module.

Library Interface Section	Description
Functions	Routines to configure this module
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_SMTPC_CLIENT_ADDR_BUFFER_SIZE	size of a buffer that can hold an email address: user@domain.smth
	TCPIP_SMTPC_CLIENT_AUTH_BUFFER_SIZE	size of a buffer that can hold the 2* username and password
	TCPIP_SMTPC_CLIENT_MESSAGE_DATE	default string that identifies the SMTPC client mail date Currently there is no date/time service available The mail server will update with the current date Keep the current data/time format that's accepted by the SMTP servers
	TCPIP_SMTPC_INTERNAL_RETRY_TIMEOUT	The retry interval because of a SMTPC temporary error, in seconds. These include temporary errors related to: DNS, socket connection, TLS.
	TCPIP_SMTPC_MAIL_CONNECTIONS	number of mail connections (sockets) to be created each mail message requires a mail connection adjust based on the number of simultaneous mail messages in transit
	TCPIP_SMTPC_MAIL_RETRIES	How many times to retry sending a mail message Retries occur only for server reported transient errors or for internal temporary errors (DNS, socket errors, etc.)
	TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE	size of an email line when sending the email body as plain text SMTP lines are recommended to be 78 chars long and MUST not exceed 998 characters! This implementation limits the line size to TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE (usually 256 or 512). So any mail line longer that this size will have inserted an artificial end of line sequence (CRLF) after this many characters.
	TCPIP_SMTPC_SERVER_DATA_TIMEOUT	server acknowledgment of the mail data: body, attachments et all; seconds Should be adjusted according to the server responsivity RFC specifies it as 10 min Default is 1 minute, which should normally be enough
	TCPIP_SMTPC_SERVER_REPLY_BUFFER_SIZE	size of the RX buffer for processing the server replies Usually the server replies are not very long so a 512 bytes buffer will normally do
	TCPIP_SMTPC_SERVER_REPLY_TIMEOUT	general server response timeout, seconds Should be adjusted according to the server responsivity RFC specifies it as 5 min Default is 1 minute, which should normally be enough
	TCPIP_SMTPC_SERVER_TRANSIENT_RETRY_TIMEOUT	The retry interval because of a server transient error, in seconds. RFC specifies it should be at least 30 min!
	TCPIP_SMTPC_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the SMTPC socket Use 0 for default TCP socket value The SMTPC RX does not need high throughput so the default value is usually fine.
	TCPIP_SMTPC_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the SMTPC socket Use 0 for default TCP socket value The SMTPC TX does not need high throughput so the default value is usually fine. For transferring large files increase this value as needed.
	TCPIP_SMTPC_TASK_TICK_RATE	SMTPC task rate, milliseconds The default value is 55 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_SMTPC_TLS_HANDSHAKE_TIMEOUT	the timeout for the TLS handshake to complete, seconds when a secure connection is established to the mail server Adjust based on the processing speed, key size, etc.
	TCPIP_SMTPC_USE_MAIL_COMMAND	Use the sample tcpip console "mail" command for sending email from the console

Description

The configuration of the SMPT TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the SMTPC TCP/IP Stack Library. Based on the selections made, the SMTPC TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the SMTPC TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_SMTPC_CLIENT_ADDR_BUFFER_SIZE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_CLIENT_ADDR_BUFFER_SIZE 80
```

Description

size of a buffer that can hold an email address: user@domain.smth

Remarks

this buffer is automatic, created on the stack Check your project stack settings if you need a large buffer here! If buffer not long enough this could generate TCPIP_SMTPC_RES_MESSAGE_ADDR_LEN_ERROR.

TCPIP_SMTPC_CLIENT_AUTH_BUFFER_SIZE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_CLIENT_AUTH_BUFFER_SIZE 100
```

Description

size of a buffer that can hold the 2* username and password

Remarks

this buffer is automatic, created on the stack Check your project stack settings if you need a large buffer here! If buffer not long enough this could generate TCPIP_SMTPC_RES_MESSAGE_AUTH_LEN_ERROR.

TCPIP_SMTPC_CLIENT_MESSAGE_DATE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_CLIENT_MESSAGE_DATE "Thu, 14 July 2016 14:55:06 -0600"
```

Description

default string that identifies the SMTPC client mail date Currently there is no date/time service available The mail server will update with the current date Keep the current data/time format that's accepted by the SMTP servers

TCPIP_SMTPC_INTERNAL_RETRY_TIMEOUT Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_INTERNAL_RETRY_TIMEOUT 10
```

Description

The retry interval because of a SMTPC temporary error, in seconds. These include temporary errors related to: DNS, socket connection, TLS.

TCPIP_SMTPC_MAIL_CONNECTIONS Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_MAIL_CONNECTIONS 3
```

Description

number of mail connections (sockets) to be created each mail message requires a mail connection adjust based on the number of simultaneous mail messages in transit

TCPIP_SMTPC_MAIL_RETRIES Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_MAIL_RETRIES 3
```

Description

How many times to retry sending a mail message Retries occur only for server reported transient errors or for internal temporary errors (DNS, socket errors, etc.)

TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE 256
```

Description

size of an email line when sending the email body as plain text SMTP lines are recommended to be 78 chars long and MUST not exceed 998 characters! This implementation limits the line size to TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE (usually 256 or 512). So any mail line longer that this size will have inserted an artificial end of line sequence (CRLF) after this many characters.

Remarks

this buffer is automatic, created on the stack Check your project stack settings if you need a large buffer here!

Lines are sent out as a whole so the line cannot be longer than the TCP socket TX buffer! SMTPC will ensure that this symbol does not exceed the size of the socket TX buffer by allocating a larger TX buffer, if needed

TCPIP_SMTPC_SERVER_DATA_TIMEOUT Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_SERVER_DATA_TIMEOUT 60
```

Description

server acknowledgment of the mail data: body, attachments et all; seconds Should be adjusted according to the server responsivity RFC specifies it as 10 min Default is 1 minute, which should normally be enough

TCPIP_SMTPC_SERVER_REPLY_BUFFER_SIZE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_SERVER_REPLY_BUFFER_SIZE 512
```

Description

size of the RX buffer for processing the server replies Usually the server replies are not very long so a 512 bytes buffer will normally do

TCPIP_SMTPC_SERVER_REPLY_TIMEOUT Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_SERVER_REPLY_TIMEOUT 60
```

Description

general server response timeout, seconds Should be adjusted according to the server responsivity RFC specifies it as 5 min Default is 1 minute, which should normally be enough

TCPIP_SMTPC_SERVER_TRANSIENT_RETRY_TIMEOUT Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_SERVER_TRANSIENT_RETRY_TIMEOUT (10 * 60)
```

Description

The retry interval because of a server transient error, in seconds. RFC specifies it should be at least 30 min!

TCPIP_SMTPC_SKT_RX_BUFF_SIZE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_SKT_RX_BUFF_SIZE 0
```

Description

Define the size of the RX buffer for the SMTPC socket Use 0 for default TCP socket value The SMTPC RX does not need high throughput so the default value is usually fine.

TCPIP_SMTPC_SKT_TX_BUFF_SIZE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_SKT_TX_BUFF_SIZE 0
```

Description

Define the size of the TX buffer for the SMTPC socket Use 0 for default TCP socket value The SMTPC TX does not need high throughput so the default value is usually fine. For transferring large files increase this value as needed.

TCPIP_SMTPC_TASK_TICK_RATE Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_TASK_TICK_RATE (55)
```

Description

SMTPC task rate, milliseconds The default value is 55 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_SMTPC_TLS_HANDSHAKE_TIMEOUT Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_TLS_HANDSHAKE_TIMEOUT 10
```

Description

the timeout for the TLS handshake to complete, seconds when a secure connection is established to the mail server Adjust based on the processing speed, key size, etc.

TCPIP_SMTPC_USE_MAIL_COMMAND Macro

File

smtpc_config.h

C

```
#define TCPIP_SMTPC_USE_MAIL_COMMAND 0
```

Description

Use the sample tcpip console "mail" command for sending email from the console

Building the Library

This section lists the files that are available in the SMTPC module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/smtpc.c`	SMTPC implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The SMTPC module depends on the following modules:

- TCP/IP Stack Library
- DNS Module
- TCP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_SMTPC_MailMessage	SMTPC mail message function.
	TCPIP_SMTPC_MessageQuery	SMTPC mail message query function.
	TCPIP_SMTPC_Task	Standard TCP/IP stack module task function.

b) Data Types and Functions

	Name	Description
	TCPIP_SMTPC_ATTACH_BUFFER	Buffer attachment to the mail.
	TCPIP_SMTPC_ATTACH_FILE	File attachment to the mail.
	TCPIP_SMTPC_ATTACH_TYPE	SMTPC attachment type
	TCPIP_SMTPC_ENCODE_TYPE	SMTPC encoding type
	TCPIP_SMTPC_MAIL_FLAGS	Mail message flags
	TCPIP_SMTPC_MAIL_MESSAGE	Mail message descriptor.
	TCPIP_SMTPC_MESSAGE_CALLBACK	Defines a mail message callback
	TCPIP_SMTPC_MESSAGE_HANDLE	Defines a handle to a mail message
	TCPIP_SMTPC_MESSAGE_QUERY	Structure defining a run time query for a mail message
	TCPIP_SMTPC_MESSAGE_REPORT	Structure defining a mail message report
	TCPIP_SMTPC_MESSAGE_RESULT	SMTPC result code
	TCPIP_SMTPC_MESSAGE_STATUS	SMTPC message status
	TCPIP_SMTPC_MESSAGE_WARNING	SMTPC warning type
	TCPIP_SMTPC_MODULE_CONFIG	SMTPC configuration data.
	TCPIP_SMTPC_SERVER_REPLY_CALLBACK	Defines a mail server reply callback

Description

This section describes the Application Programming Interface (API) functions of the SMTPC module.

Refer to each section for a detailed description.

a) Functions

TCPIP_SMTPC_MailMessage Function

SMTPC mail message function.

File

smtpc.h

C

```
TCPIP_SMTPC_MESSAGE_HANDLE TCPIP_SMTPC_MailMessage(const TCPIP_SMTPC_MAIL_MESSAGE* pMailMessage,
TCPIP_SMTPC_MESSAGE_RESULT* pRes);
```

Returns

a valid handle (!= 0) - if mail message scheduled successfully 0 - if call failed

an error code otherwise

Description

This function sends a mail message to the specified server.

Remarks

The returned handle is valid until the final mail notification function is called (TCPIP_SMTPC_MESSAGE_CALLBACK) if exists, or until the mail transaction is completed. After that the messageHandle is no longer valid and should not be used again.

Preconditions

SMTPC module should have been initialized.

Parameters

Parameters	Description
pMailMessage	pointer to a TCPIP_SMTPC_MAIL_MESSAGE describing the mail message to be sent
pRes	address to store the operation result If the call failed, additional info will be reported with this result Can be NULL if not needed.

Function

TCPIP_SMTPC_MESSAGE_HANDLE TCPIP_SMTPC_MailMessage(const TCPIP_SMTPC_MAIL_MESSAGE* pMailMessage, TCPIP_SMTPC_MESSAGE_RESULT* pRes);

TCPIP_SMTPC_MessageQuery Function

SMTPC mail message query function.

File

smtpc.h

C

```
TCPIP_SMTPC_MESSAGE_RESULT TCPIP_SMTPC_MessageQuery(TCPIP_SMTPC_MESSAGE_HANDLE messageHandle,
TCPIP_SMTPC_MESSAGE_QUERY* pQuery);
```

Returns

TCPIP_SMTPC_RES_OK - if message query updated successfully TCPIP_SMTPC_RES_MESSAGE_ERROR - if no such message exists

Description

This function allows the query of a mail message at run time.

Remarks

The message status is for info purposes only. The message status changes as is processed by the SMTPC module.

Once the mail transfer is complete and the corresponding notification function is called, the messageHandle is no longer valid and should not be used again.

Preconditions

SMTPC module should have been initialized.

Parameters

Parameters	Description
messageHandle	handle identifying the message. This handle was obtained with a call to TCPIP_SMTPC_MailMessage().
pQuery	pointer to a TCPIP_SMTPC_MESSAGE_QUERY structure that will be updated with the current status of the mail message

Function

TCPIP_SMTPC_MESSAGE_RESULT TCPIP_SMTPC_MessageQuery(TCPIP_SMTPC_MESSAGE_HANDLE messageHandle, TCPIP_SMTPC_MESSAGE_QUERY* pQuery);

TCPIP_SMTPC_Task Function

Standard TCP/IP stack module task function.

File

smtpc.h

C

```
void TCPIP_SMTPC_Task();
```

Returns

None.

Description

this function performs SMTPC module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

SMTPC module should have been initialized.

Function

void TCPIP_SMTPC_Task(void)

b) Data Types and Functions

TCPIP_SMTPC_ATTACH_BUFFER Structure

Buffer attachment to the mail.

File

smtpc.h

C

```
typedef struct {
  TCPIP_SMTPC_ATTACH_TYPE attachType;
  TCPIP_SMTPC_ENCODE_TYPE attachEncode;
  const char* attachName;
  const uint8_t* attachBuffer;
  size_t attachSize;
} TCPIP_SMTPC_ATTACH_BUFFER;
```

Members

Members	Description
TCPIP_SMTPC_ATTACH_TYPE attachType;	type of the contained data
TCPIP_SMTPC_ENCODE_TYPE attachEncode;	preferred encoding
const char* attachName;	file name to be used for the attachment
const uint8_t* attachBuffer;	buffer containing the data
size_t attachSize;	buffer size

Description

Structure: TCPIP_SMTPC_ATTACH_BUFFER

This data structure describes a mail buffer attachment.

Remarks

A buffer attachment is attached to the email just like a file. But data is taken from the persistent user buffer rather than from a file.

If the attach type is application octet stream then the suggested encoding will be overwritten and BASE64 encoding will be used.

The user supplied buffer has to be persistent until the mail completion callback will be called.

TCPIP_SMTPC_ATTACH_FILE Structure

File attachment to the mail.

File

smtpc.h

C

```
typedef struct {
  TCPIP_SMTPC_ATTACH_TYPE attachType;
  TCPIP_SMTPC_ENCODE_TYPE attachEncode;
  const char* fileName;
} TCPIP_SMTPC_ATTACH_FILE;
```

Members

Members	Description
TCPIP_SMTPC_ATTACH_TYPE attachType;	type of the contained data
TCPIP_SMTPC_ENCODE_TYPE attachEncode;	preferred encoding
const char* fileName;	name of the file to be opened

Description

Structure: TCPIP_SMTPC_ATTACH_FILE

This data structure describes a mail file attachment.

Remarks

The specified file will be opened for reading and attached to the email.

If the attach type is application octet stream then the suggested encoding will be overwritten and BASE64 encoding will be used.

The file has to be persistent until the mail completion callback will be called.

If file could not be opened when the mail is sent a warning will be issued and the file will be ignored.

TCPIP_SMTPC_ATTACH_TYPE Enumeration

SMTPC attachment type

File

smtpc.h

C

```
typedef enum {
  TCPIP_SMTPC_ATTACH_TYPE_UNKNOWN = 0x0000,
  TCPIP_SMTPC_ATTACH_TYPE_APPLICATION = 0x0001,
  TCPIP_SMTPC_ATTACH_TYPE_TEXT = 0x0002,
  TCPIP_SMTPC_ATTACH_TYPE_INLINE = 0x8000
} TCPIP_SMTPC_ATTACH_TYPE;
```

Members

Members	Description
TCPIP_SMTPC_ATTACH_TYPE_UNKNOWN = 0x0000	unknown attachment type; will default to binary
TCPIP_SMTPC_ATTACH_TYPE_APPLICATION = 0x0001	application octet stream, binary Use this attachment type when sending binary data
TCPIP_SMTPC_ATTACH_TYPE_TEXT = 0x0002	plain text Use this attachment type when sending ASCII text other supported types will be added here
TCPIP_SMTPC_ATTACH_TYPE_INLINE = 0x8000	flag for sending the data as inline instead of attachment

Description

Enumeration: TCPIP_SMTPC_ATTACH_TYPE

Enumeration describing the possible attachment types supported by the SMTPC client.

Remarks

Only one attachment type should be set at a time.

TCPIP_SMTPC_ENCODE_TYPE Enumeration

SMTPC encoding type

File

smtpc.h

C

```
typedef enum {
  TCPIP_SMTPC_ENCODE_TYPE_UNKNOWN,
  TCPIP_SMTPC_ENCODE_TYPE_7BIT,
  TCPIP_SMTPC_ENCODE_TYPE_BASE64
```

```
} TCPIP_SMTPC_ENCODE_TYPE;
```

Members

Members	Description
TCPIP_SMTPC_ENCODE_TYPE_UNKNOWN	unknown encoding type; defaults to base64
TCPIP_SMTPC_ENCODE_TYPE_7BIT	7bit encoding type: no encoding done, 7 bit ASCII This doesn't do any encoding. Use it for plain text.
TCPIP_SMTPC_ENCODE_TYPE_BASE64	base64 encoding applied Mandatory for binary data; could be used for plain data as well. other supported types will be added here

Description

Enumeration: TCPIP_SMTPC_ENCODE_TYPE

Enumeration describing the possible encoding types supported by the SMTPC client.

Remarks

Normally plain text should be sen using the 7bit encoding type, i.e. plain ASCII. However, selecting base64 encoding has the advantage that the lines are not limited in length and any sequence of characters can be present in the email body without the need of the SMTPC to search and replace the SMTP forbidden characters.

TCPIP_SMTPC_MAIL_FLAGS Enumeration

Mail message flags

File

smtpc.h

C

```
typedef enum {
  TCPIP_SMTPC_MAIL_FLAG_USE_IPV6 = 0x0001,
  TCPIP_SMTPC_MAIL_FLAG_BODY_TYPE_BINARY = 0x0002,
  TCPIP_SMTPC_MAIL_FLAG_BODY_ENCODE_ASCII = 0x0004,
  TCPIP_SMTPC_MAIL_FLAG_CONNECT_TLS = 0x0010,
  TCPIP_SMTPC_MAIL_FLAG_SKIP_TLS = 0x0020,
  TCPIP_SMTPC_MAIL_FLAG_FORCE_TLS = 0x0040,
  TCPIP_SMTPC_MAIL_FLAG_AUTH_PLAIN = 0x0100
} TCPIP_SMTPC_MAIL_FLAGS;
```

Members

Members	Description
TCPIP_SMTPC_MAIL_FLAG_USE_IPV6 = 0x0001	use IPv6 connection to the server, rather than default IPv4
TCPIP_SMTPC_MAIL_FLAG_BODY_TYPE_BINARY = 0x0002	mail body is binary rather then standard plain text
TCPIP_SMTPC_MAIL_FLAG_BODY_ENCODE_ASCII = 0x0004	send mail body as plain ASCII encoding rather than default BASE64
TCPIP_SMTPC_MAIL_FLAG_CONNECT_TLS = 0x0010	connect with TLS from the very beginning This flag is useful if the server accepts encrypted connections on port 465, for example Note that port 465 is not really a SMTP port, although supported by many servers!
TCPIP_SMTPC_MAIL_FLAG_SKIP_TLS = 0x0020	skip TLS even if the server supports it most servers won't accept mail if TLS is skipped though
TCPIP_SMTPC_MAIL_FLAG_FORCE_TLS = 0x0040	start TLS even if the server does not indicate support for it
TCPIP_SMTPC_MAIL_FLAG_AUTH_PLAIN = 0x0100	favor PLAIN authentication, if supported by default the LOGIN authentication is selected

Description

Enumeration: TCPIP_SMTPC_MAIL_FLAGS

Enumeration describing the possible mail message flags supported by the SMTPC client.

Remarks

Multiple flags can be set

Only 16 bit flags supported

If no flags are set, then the default settings are used for the mail message.

Normally the mail body is plain text. Using these flags, a binary mail body can be sent.

Normally the mail body is plain text so a standard ASCII encoding is used. Using these flags, a BASE64 encoding for the mail body can be

selected. Note that if the mail body is selected to be binary then the BASE64 encoding is automatically selected.

BASE64 encoding has the advantage that the mail body lines are not limited in length and any sequence of characters can be present in the email body without the need of the SMTPC to search and replace the SMTP forbidden characters.

Normally the connection to the server occurs on a standard SMTP port: 25 or (preferred) 587 as a plain/non-encrypted connection. Once the server replies with TLS support, the encryption is started. Using the flags you can force the connection to be TLS from the very beginning. Or you can avoid engaging in TLS even if the server supports it. Or try to start TLS even if the server didn't advertise support for it.

TCPIP_SMTPC_MAIL_MESSAGE Structure

Mail message descriptor.

File

smtpc.h

C

```
typedef struct {
  const char* from;
  const char* to;
  const char* sender;
  const char* cc;
  const char* bcc;
  const char* date;
  const char* subject;
  const uint8_t* body;
  size_t bodySize;
  int nBuffers;
  const TCPIP_SMTPC_ATTACH_BUFFER* attachBuffers;
  int nFiles;
  const TCPIP_SMTPC_ATTACH_FILE* attachFiles;
  const char* username;
  const char* password;
  const char* smtpServer;
  uint16_t serverPort;
  TCPIP_SMTPC_MAIL_FLAGS messageFlags;
  TCPIP_SMTPC_MESSAGE_CALLBACK messageCallback;
  TCPIP_SMTPC_SERVER_REPLY_CALLBACK replyCallback;
} TCPIP_SMTPC_MAIL_MESSAGE;
```

Members

Members	Description
const char* from;	Mandatory Mail Fields recipient return address
const char* to;	destination address
const char* sender;	Optional Mail Fields if sent on behalf of someone else
const char* cc;	carbon copy destination address
const char* bcc;	blind carbon copy destination address
const char* date;	date for the mail message Should be formatted like: "Wed, 20 July 2016 14:55:06 -0600" if NULL, then the TCPIP_SMTPC_CLIENT_MESSAGE_DATE will be used
const char* subject;	Regular Mail Fields mail subject line
const uint8_t* body;	mail body; main body of the mail
int nBuffers;	Mail Attachments number of attached buffers
const TCPIP_SMTPC_ATTACH_BUFFER* attachBuffers;	array of buffer attachments
int nFiles;	number of attached files
const TCPIP_SMTPC_ATTACH_FILE* attachFiles;	array of file attachments
const char* username;	Mail Authentication username to be presented when the mail server requires authentication
const char* password;	associated password to be presented when the mail server requires authentication
const char* smtpServer;	Mail Server address or name of a server to send the mail to (smtp.gmail.com, 74.125.23.109, etc.)

uint16_t serverPort;	port on which the SMTP server listens on Standard SMTP ports are 25 or 587 For servers supporting TLS (gmail, yahoo, etc.), the recommended port is 587
TCPIP_SMTPC_MAIL_FLAGS messageFlags;	Additional message flags
TCPIP_SMTPC_MESSAGE_CALLBACK messageCallback;	Mail Notifications message completion notification could be NULL if not needed
TCPIP_SMTPC_SERVER_REPLY_CALLBACK replyCallback;	server reply notification could be NULL if not needed

Description

Structure: TCPIP_SMTPC_MAIL_MESSAGE

This data structure describes a mail message

Remarks

SMTPC does NOT make private copies of the strings, buffers, etc. presented in a TCPIP_SMTPC_MAIL_MESSAGE data structure. All fields that are pointers to strings, buffers, files names, etc. MUST be persistent until the mail complete notification is received.

The TCPIP_SMTPC_MAIL_MESSAGE data structure itself does not need to be persistent.

The mail body is not a char* but uint8_t* to allow sending binary data as well as part of the mail body. This requires though that the bodySize member is present.

If some file could not be opened when the mail is sent a warning will be issued and the file will be ignored.

Currently the fields: "sender", "cc" and "bcc" are not used. They will be eventually added.

TCPIP_SMTPC_MESSAGE_CALLBACK Type

Defines a mail message callback

File

smtpc.h

C

```
typedef void (* TCPIP_SMTPC_MESSAGE_CALLBACK)(TCPIP_SMTPC_MESSAGE_HANDLE messageHandle, const
TCPIP_SMTPC_MESSAGE_REPORT* pMailReport);
```

Description

Type: TCPIP_SMTPC_MESSAGE_CALLBACK

Data type describing a function that will be called by the SMTPC module upon completion of the mail message.

Remarks

A message handle is used rather than the user's TCPIP_SMTPC_MAIL_MESSAGE. This allows sending multiple copies of the message without waiting for a previous copy to be done.

This notification can be called multiple times if the delivery attempt failed for a transient reson. The number of left retries tells if another attempt will be retried.

Once this final notification function is called (left retries == 0), the SMTPC no longer uses the TCPIP_SMTPC_MAIL_MESSAGE data (unless there are other copies of this message in transit). This messageHandle is no longer valid and should not be used again.

The pMailReport points to a SMTPC owned read only data structure. It should be used only to inspect the message delivery result.

Parameters

Parameters	Description
messageHandle	handle identifying the message. This handle was obtained with a call to TCPIP_SMTPC_MailMessage().
pMailReport	pointer to a TCPIP_SMTPC_MESSAGE_REPORT data structure containing the
result of the message delivery	messageRes - the result of the message
errorStat	error status
messageWarn	warnings
leftRetries	number of times the delivery will be re-attempted.

TCPIP_SMTPC_MESSAGE_HANDLE Type

Defines a handle to a mail message

File

smtpc.h

C

```
typedef const void* TCPIP_SMTPC_MESSAGE_HANDLE;
```

Description

Type: TCPIP_SMTPC_MESSAGE_HANDLE

Data type describing a handle that identifies a mail message that's scheduled for transmission. This handle is to be used for retrieving the message status. This handle will be part of the callback that the SMTP client will use to notify the user of the mail completion.

A valid handle is != 0.

Remarks

None

TCPIP_SMTPC_MESSAGE_QUERY Structure

Structure defining a run time query for a mail message

File

smtpc.h

C

```
typedef struct {
  TCPIP_SMTPC_MESSAGE_STATUS messageStat;
  TCPIP_SMTPC_MESSAGE_WARNING messageWarn;
  int messageRetries;
  NET_PRES_SKT_HANDLE_T messageSkt;
} TCPIP_SMTPC_MESSAGE_QUERY;
```

Members

Members	Description
TCPIP_SMTPC_MESSAGE_STATUS messageStat;	the current status of the mail message
TCPIP_SMTPC_MESSAGE_WARNING messageWarn;	the current warnings encountered during the message transmission
int messageRetries;	the current number of retries for this message
NET_PRES_SKT_HANDLE_T messageSkt;	the socket associated with this message

Description

Structure: TCPIP_SMTPC_MESSAGE_QUERY

This data structure describes a mail message query at run time.

Remarks

The message status is for info purposes only. The message status changes as is processed by the SMTPC module.

TCPIP_SMTPC_MESSAGE_REPORT Structure

Structure defining a mail message report

File

smtpc.h

C

```
typedef struct {
  TCPIP_SMTPC_MESSAGE_RESULT messageRes;
  TCPIP_SMTPC_MESSAGE_STATUS errorStat;
  TCPIP_SMTPC_MESSAGE_WARNING messageWarn;
  int leftRetries;
} TCPIP_SMTPC_MESSAGE_REPORT;
```

Members

Members	Description
TCPIP_SMTPC_MESSAGE_RESULT messageRes;	the result of the message TCPIP_SMTPC_RES_OK if the message was successfully transmited else some error were encountered
TCPIP_SMTPC_MESSAGE_STATUS errorStat;	the mail message status in which the error occurred, if any Else the TCPIP_SMTPC_MESSAGE_STAT_NONE will be used
TCPIP_SMTPC_MESSAGE_WARNING messageWarn;	warnings encountered during the message transmission, if any
int leftRetries;	the number of retries that will still be attempted if error was a transient, not fatal

Description

Structure: TCPIP_SMTPC_MESSAGE_REPORT

This data structure describes a mail message report. It is used by the SMTPC to report an mail delivery result.

Remarks

None

TCPIP_SMTPC_MESSAGE_RESULT Enumeration

SMTPC result code

File

smtpc.h

C

```
typedef enum {
  TCPIP_SMTPC_RES_OK = 0,
  TCPIP_SMTPC_RES_PENDING = 1,
  TCPIP_SMTPC_RES_MESSAGE_ERROR = -1,
  TCPIP_SMTPC_RES_MESSAGE_SERVER_ERROR = -2,
  TCPIP_SMTPC_RES_MESSAGE_RCPT_ERROR = -3,
  TCPIP_SMTPC_RES_MESSAGE_BUFFER_ERROR = -4,
  TCPIP_SMTPC_RES_MESSAGE_FILE_ERROR = -5,
  TCPIP_SMTPC_RES_MESSAGE_AUTH_REQUIRED = -6,
  TCPIP_SMTPC_RES_MESSAGE_AUTH_LEN_ERROR = -7,
  TCPIP_SMTPC_RES_MESSAGE_ADDR_LEN_ERROR = -8,
  TCPIP_SMTPC_RES_MAIL_BUSY = -9,
  TCPIP_SMTPC_RES_DNS_ERROR = -10,
  TCPIP_SMTPC_RES_SKT_OPEN_ERROR = -11,
  TCPIP_SMTPC_RES_SKT_BIND_ERROR = -12,
  TCPIP_SMTPC_RES_SKT_CONNECT_TMO = -13,
  TCPIP_SMTPC_RES_SKT_TLS_ERROR = -14,
  TCPIP_SMTPC_RES_SERVER_TMO = -15,
  TCPIP_SMTPC_RES_CONNECTION_REJECT = -16,
  TCPIP_SMTPC_RES_CONNECTION_CLOSE = -17,
  TCPIP_SMTPC_RES_HELLO_REJECT = -18,
  TCPIP_SMTPC_RES_AUTH_UNKNOWN = -19,
  TCPIP_SMTPC_RES_AUTH_LOGIN_REJECT = -20,
  TCPIP_SMTPC_RES_AUTH_LOGIN_SERVER_ERROR = -21,
  TCPIP_SMTPC_RES_AUTH_REJECT = -22,
  TCPIP_SMTPC_RES_TLS_REJECT = -23,
  TCPIP_SMTPC_RES_TLS_FAILED = -24,
  TCPIP_SMTPC_RES_TLS_TMO = -25,
  TCPIP_SMTPC_RES_MAIL_FROM_REJECT = -26,
  TCPIP_SMTPC_RES_MAIL_RCPT_REJECT = -27,
  TCPIP_SMTPC_RES_MAIL_DATA_REJECT = -28,
  TCPIP_SMTPC_RES_MAIL_BODY_REJECT = -29,
  TCPIP_SMTPC_RES_INITIALIZE_ERROR = -40,
  TCPIP_SMTPC_RES_INTERNAL_ERROR = -41
} TCPIP_SMTPC_MESSAGE_RESULT;
```

Members

Members	Description
TCPIP_SMTPC_RES_OK = 0	OK, operation successful
TCPIP_SMTPC_RES_PENDING = 1	operation in progress
TCPIP_SMTPC_RES_MESSAGE_ERROR = -1	mail message error

TCPIP_SMTPC_RES_MESSAGE_SERVER_ERROR = -2	message indicated wrong mail server
TCPIP_SMTPC_RES_MESSAGE_RCPT_ERROR = -3	message mail recipient error: from, to, etc.
TCPIP_SMTPC_RES_MESSAGE_BUFFER_ERROR = -4	attachment buffer error
TCPIP_SMTPC_RES_MESSAGE_FILE_ERROR = -5	attachment file error
TCPIP_SMTPC_RES_MESSAGE_AUTH_REQUIRED = -6	server requires authentication but username or password haven't been provided
TCPIP_SMTPC_RES_MESSAGE_AUTH_LEN_ERROR = -7	provided credentials are too long, buffer overflow
TCPIP_SMTPC_RES_MESSAGE_ADDR_LEN_ERROR = -8	email address too long, buffer overflow
TCPIP_SMTPC_RES_MAIL_BUSY = -9	all mail connections are busy; try later
TCPIP_SMTPC_RES_DNS_ERROR = -10	failure to resolve server name
TCPIP_SMTPC_RES_SKT_OPEN_ERROR = -11	failure to open a communication socket
TCPIP_SMTPC_RES_SKT_BIND_ERROR = -12	failure to bind a socket to the mail server
TCPIP_SMTPC_RES_SKT_CONNECT_TMO = -13	connection to mail server timeout
TCPIP_SMTPC_RES_SKT_TLS_ERROR = -14	TLS is required but failed to start TLS on the communication socket
TCPIP_SMTPC_RES_SERVER_TMO = -15	server timeout
TCPIP_SMTPC_RES_CONNECTION_REJECT = -16	server rejected the connection
TCPIP_SMTPC_RES_CONNECTION_CLOSE = -17	server closed the connection
TCPIP_SMTPC_RES_HELLO_REJECT = -18	server rejected the hello greeting
TCPIP_SMTPC_RES_AUTH_UNKNOWN = -19	server requires authentication mechanism unsupported by SMTPC Currently LOGIN and PLAIN authentications are supported
TCPIP_SMTPC_RES_AUTH_LOGIN_REJECT = -20	server rejected the login authentication request
TCPIP_SMTPC_RES_AUTH_LOGIN_SERVER_ERROR = -21	unexpected server reply to login authentication request
TCPIP_SMTPC_RES_AUTH_REJECT = -22	server rejected the supplied authentication
TCPIP_SMTPC_RES_TLS_REJECT = -23	server rejected the TLS start
TCPIP_SMTPC_RES_TLS_FAILED = -24	TLS session negotiation failed
TCPIP_SMTPC_RES_TLS_TMO = -25	TLS session timeout
TCPIP_SMTPC_RES_MAIL_FROM_REJECT = -26	server rejected the "from" address
TCPIP_SMTPC_RES_MAIL_RCPT_REJECT = -27	server rejected the "recipient" address
TCPIP_SMTPC_RES_MAIL_DATA_REJECT = -28	server rejected the "data" field
TCPIP_SMTPC_RES_MAIL_BODY_REJECT = -29	server rejected the mail body
TCPIP_SMTPC_RES_INITIALIZE_ERROR = -40	module is not properly initialized, API call is invalid
TCPIP_SMTPC_RES_INTERNAL_ERROR = -41	internal failure, should not happen

Description

Enumeration: TCPIP_SMTPC_MESSAGE_RESULT

Enumeration describing the possible result codes returned by the SMTPC client.

Remarks

None

TCPIP_SMTPC_MESSAGE_STATUS Enumeration

SMTPC message status

File

smtpc.h

C

```
typedef enum {
  TCPIP_SMTPC_MESSAGE_STAT_NONE,
  TCPIP_SMTPC_MESSAGE_STAT_DNS,
  TCPIP_SMTPC_MESSAGE_STAT_CONNECT,
  TCPIP_SMTPC_MESSAGE_STAT_HELLO,
  TCPIP_SMTPC_MESSAGE_STAT_TLS,
```

```
  TCPIP_SMTPC_MESSAGE_STAT_AUTH,
  TCPIP_SMTPC_MESSAGE_STAT_MAIL_ENVELOPE,
  TCPIP_SMTPC_MESSAGE_STAT_MAIL_BODY,
  TCPIP_SMTPC_MESSAGE_STAT_MAIL_BUFFERS,
  TCPIP_SMTPC_MESSAGE_STAT_MAIL_FILES,
  TCPIP_SMTPC_MESSAGE_STAT_MAIL_ACKNOWLEDGE,
  TCPIP_SMTPC_MESSAGE_STAT_MAIL_REPORT,
  TCPIP_SMTPC_MESSAGE_STAT_CLOSING,
  TCPIP_SMTPC_MESSAGE_STAT_WAIT_RETRY
} TCPIP_SMTPC_MESSAGE_STATUS;
```

Members

Members	Description
TCPIP_SMTPC_MESSAGE_STAT_NONE	there is no transaction started
TCPIP_SMTPC_MESSAGE_STAT_DNS	doing the DNS resolution
TCPIP_SMTPC_MESSAGE_STAT_CONNECT	connecting to the server
TCPIP_SMTPC_MESSAGE_STAT_HELLO	saying hello to the server
TCPIP_SMTPC_MESSAGE_STAT_TLS	starting the TLS communication with the server
TCPIP_SMTPC_MESSAGE_STAT_AUTH	authenticating to the server
TCPIP_SMTPC_MESSAGE_STAT_MAIL_ENVELOPE	sending mail envelope
TCPIP_SMTPC_MESSAGE_STAT_MAIL_BODY	sending mail body
TCPIP_SMTPC_MESSAGE_STAT_MAIL_BUFFERS	sending mail buffers
TCPIP_SMTPC_MESSAGE_STAT_MAIL_FILES	sending mail files
TCPIP_SMTPC_MESSAGE_STAT_MAIL_ACKNOWLEDGE	waiting for the mail acknowledge
TCPIP_SMTPC_MESSAGE_STAT_MAIL_REPORT	reporting the mail outcome
TCPIP_SMTPC_MESSAGE_STAT_CLOSING	closing the mail transaction
TCPIP_SMTPC_MESSAGE_STAT_WAIT_RETRY	waiting for a retry

Description

Enumeration: TCPIP_SMTPC_MESSAGE_STATUS

Enumeration describing the possible states of a mail message during a mail transmission. The message progresses through various states until complete mail transfer or error

Remarks

TCPIP_SMTPC_MESSAGE_STAT_HELLO will occur twice if TLS is supported by the server. Once the connection is secured, another HELLO message will be sent to the server.

TCPIP_SMTPC_MESSAGE_WARNING Enumeration

SMTPC warning type

File

smtpc.h

C

```
typedef enum {
  TCPIP_SMTPC_WARN_REPLY_BUFFER_SMALL = 0x0001,
  TCPIP_SMTPC_WARN_WRONG_FORMAT = 0x0002,
  TCPIP_SMTPC_WARN_UNEXPECTED_REPLY = 0x0004,
  TCPIP_SMTPC_WARN_FILE_NOT_FOUND = 0x0008,
  TCPIP_SMTPC_WARN_AUTH_LOGIN_LEN = 0x0010
} TCPIP_SMTPC_MESSAGE_WARNING;
```

Members

Members	Description
TCPIP_SMTPC_WARN_REPLY_BUFFER_SMALL = 0x0001	the reply buffer allocated for the server replies is too small server reply exceeded the size of the buffer it should be increased
TCPIP_SMTPC_WARN_WRONG_FORMAT = 0x0002	server reply was not properly formatted
TCPIP_SMTPC_WARN_UNEXPECTED_REPLY = 0x0004	unexpected server replied received
TCPIP_SMTPC_WARN_FILE_NOT_FOUND = 0x0008	file attachment not found; ignored

TCPIP_SMTPC_WARN_AUTH_LOGIN_LEN = 0x0010	provided credentials for LOGIN authentication are too long and were truncated (buffer overflow) this most likely will result in a server authentication rejection

Description

Enumeration: TCPIP_SMTPC_MESSAGE_WARNING

Enumeration describing the possible warning types occurring during a mail transmission. These events are not fatal and SMTPC client will try to complete the mail transfer.

Remarks

Multiple warnings could be set during a mail transmission

TCPIP_SMTPC_MODULE_CONFIG Structure

SMTPC configuration data.

File

smtpc.h

C

```
typedef struct {
  int nMailConnections;
  uint32_t serverReplyTmo;
  uint32_t serverDataTmo;
  uint32_t tlsHandshakeTmo;
  int nMailRetries;
  uint32_t serverRetryTmo;
  uint32_t smtpcRetryTmo;
  uint16_t sktTxBuffSize;
  uint16_t sktRxBuffSize;
} TCPIP_SMTPC_MODULE_CONFIG;
```

Members

Members	Description
int nMailConnections;	number of mail connections to be created each mail message to be sent requires a mail connection adjust based on the number of simultaneous mail messages in transit
uint32_t serverReplyTmo;	general server response timeout, seconds Should be adjusted according to the server responsivity RFC specifies it as 5 min
uint32_t serverDataTmo;	server response timeout to acknowledge a received mail body/data, seconds Should be adjusted according to the server responsivity RFC specifies it as 10 min
uint32_t tlsHandshakeTmo;	secure connection establishment timeout, seconds usually few seconds
int nMailRetries;	how many times to retry sending a mail message Retries occur only for server reported transient errors
uint32_t serverRetryTmo;	The retry interval because of a server transient error, in seconds. RFC specifies it should be at least 30 min!
uint32_t smtpcRetryTmo;	The retry interval because of a SMTPC temporary error, in seconds. These include temporary errors related to: DNS, socket connection, TLS.
uint16_t sktTxBuffSize;	size of TX buffer for the associated socket; leave 0 for default
uint16_t sktRxBuffSize;	size of RX buffer for the associated socket; leave 0 for default

Description

Structure: TCPIP_SMTPC_MODULE_CONFIG

This data structure describes the SMTPC configuration data. It is presented to the module as part of the module initialization.

Remarks

None

TCPIP_SMTPC_SERVER_REPLY_CALLBACK Type

Defines a mail server reply callback

File

smtpc.h

C

```
typedef void (* TCPIP_SMTPC_SERVER_REPLY_CALLBACK)(TCPIP_SMTPC_MESSAGE_HANDLE messageHandle,
TCPIP_SMTPC_MESSAGE_STATUS currStat, const char* serverReply);
```

Description

Type: TCPIP_SMTPC_SERVER_REPLY_CALLBACK

Data type describing a function that will be called by the SMTPC module upon receiving a message from the mail server. SMTPC will parse and process the server messages anyway but it can optionally pass the server replies to the user.

Remarks

None

Parameters

Parameters	Description
messageHandle	handle identifying the message. This handle was obtained with a call to TCPIP_SMTPC_MailMessage().
currStat	the current status for the mail message
serverReply	the reply sent by the server

Files

Files

Name	Description
smtpc.h	Module for Microchip TCP/IP Stack.
smtpc_config.h	SMTPC configuration file

Description

This section lists the source and header files used by the library.

smtpc.h

Module for Microchip TCP/IP Stack.

Enumerations

	Name	Description
	TCPIP_SMTPC_ATTACH_TYPE	SMTPC attachment type
	TCPIP_SMTPC_ENCODE_TYPE	SMTPC encoding type
	TCPIP_SMTPC_MAIL_FLAGS	Mail message flags
	TCPIP_SMTPC_MESSAGE_RESULT	SMTPC result code
	TCPIP_SMTPC_MESSAGE_STATUS	SMTPC message status
	TCPIP_SMTPC_MESSAGE_WARNING	SMTPC warning type

Functions

	Name	Description
	TCPIP_SMTPC_MailMessage	SMTPC mail message function.
	TCPIP_SMTPC_MessageQuery	SMTPC mail message query function.
	TCPIP_SMTPC_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_SMTPC_ATTACH_BUFFER	Buffer attachment to the mail.
	TCPIP_SMTPC_ATTACH_FILE	File attachment to the mail.
	TCPIP_SMTPC_MAIL_MESSAGE	Mail message descriptor.
	TCPIP_SMTPC_MESSAGE_QUERY	Structure defining a run time query for a mail message
	TCPIP_SMTPC_MESSAGE_REPORT	Structure defining a mail message report
	TCPIP_SMTPC_MODULE_CONFIG	SMTPC configuration data.

Types

	Name	Description
	TCPIP_SMTPC_MESSAGE_CALLBACK	Defines a mail message callback
	TCPIP_SMTPC_MESSAGE_HANDLE	Defines a handle to a mail message
	TCPIP_SMTPC_SERVER_REPLY_CALLBACK	Defines a mail server reply callback

Description

Simple Mail Transfer Protocol Client - SMTPC

The SMTPC client module in the TCP/IP Stack lets applications send e-mails to any recipient worldwide. These message could include status information or important alerts. Using the e-mail to SMS gateways provided by most cell phone carriers, these messages can also be delivered directly to cell phone handsets.

File Name

smtpc.h

Company

Microchip Technology Inc.

smtpc_config.h

SMTPC configuration file

Macros

	Name	Description
	TCPIP_SMTPC_CLIENT_ADDR_BUFFER_SIZE	size of a buffer that can hold an email address: user@domain.smth
	TCPIP_SMTPC_CLIENT_AUTH_BUFFER_SIZE	size of a buffer that can hold the 2* username and password
	TCPIP_SMTPC_CLIENT_MESSAGE_DATE	default string that identifies the SMTPC client mail date Currently there is no date/time service available The mail server will update with the current date Keep the current data/time format that's accepted by the SMTP servers
	TCPIP_SMTPC_INTERNAL_RETRY_TIMEOUT	The retry interval because of a SMTPC temporary error, in seconds. These include temporary errors related to: DNS, socket connection, TLS.
	TCPIP_SMTPC_MAIL_CONNECTIONS	number of mail connections (sockets) to be created each mail message requires a mail connection adjust based on the number of simultaneous mail messages in transit
	TCPIP_SMTPC_MAIL_RETRIES	How many times to retry sending a mail message Retries occur only for server reported transient errors or for internal temporary errors (DNS, socket errors, etc.)
	TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE	size of an email line when sending the email body as plain text SMTP lines are recommended to be 78 chars long and MUST not exceed 998 characters! This implementation limits the line size to TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE (usually 256 or 512). So any mail line longer that this size will have inserted an artificial end of line sequence (CRLF) after this many characters.
	TCPIP_SMTPC_SERVER_DATA_TIMEOUT	server acknowledgment of the mail data: body, attachments et all; seconds Should be adjusted according to the server responsivity RFC specifies it as 10 min Default is 1 minute, which should normally be enough
	TCPIP_SMTPC_SERVER_REPLY_BUFFER_SIZE	size of the RX buffer for processing the server replies Usually the server replies are not very long so a 512 bytes buffer will normally do
	TCPIP_SMTPC_SERVER_REPLY_TIMEOUT	general server response timeout, seconds Should be adjusted according to the server responsivity RFC specifies it as 5 min Default is 1 minute, which should normally be enough
	TCPIP_SMTPC_SERVER_TRANSIENT_RETRY_TIMEOUT	The retry interval because of a server transient error, in seconds. RFC specifies it should be at least 30 min!
	TCPIP_SMTPC_SKT_RX_BUFF_SIZE	Define the size of the RX buffer for the SMTPC socket Use 0 for default TCP socket value The SMTPC RX does not need high throughput so the default value is usually fine.
	TCPIP_SMTPC_SKT_TX_BUFF_SIZE	Define the size of the TX buffer for the SMTPC socket Use 0 for default TCP socket value The SMTPC TX does not need high throughput so the default value is usually fine. For transferring large files increase this value as needed.

	TCPIP_SMTPC_TASK_TICK_RATE	SMTPC task rate, milliseconds The default value is 55 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_SMTPC_TLS_HANDSHAKE_TIMEOUT	the timeout for the TLS handshake to complete, seconds when a secure connection is established to the mail server Adjust based on the processing speed, key size, etc.
	TCPIP_SMTPC_USE_MAIL_COMMAND	Use the sample tcpip console "mail" command for sending email from the console

Description

Simple Mail Transfer Protocol Client (SMTPC) Configuration file

This file contains the SMTPC module configuration options

File Name

smtpc_config.h

Company

Microchip Technology Inc.

SNMP Module

This section describes the TCP/IP Stack Library SNMP module.

Introduction

TCP/IP Stack Library Simple Network Management Protocol (SNMP) Module for Microchip Microcontrollers

This library provides the API of the SNMP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

SNMP is one of the key components of a Network Management System (NMS). SNMP is an application layer protocol that facilitates the exchange of management information among network devices. It is a part of the TCP/IP protocol suite.

SNMP is an Internet protocol that was originally designed to manage different network devices, such as file servers, hubs, routers and so on. It can also be used to manage and control an ever increasing number of small embedded systems connected to one another over any IP network. Systems can communicate with each other using SNMP to transfer control and status information, creating a truly distributed system.

SNMP is used in a variety of applications where remote monitoring and controlling of the network node is desired, such as a network printer, online Uninterrupted Power Supply (UPS), security cameras, home and industrial appliances monitor and control, automatic energy meter readings, etc.

Unlike more familiar human-oriented protocols, like HTTP, SNMP is considered a machine-to-machine protocol.

Note: The related application note, AN870 *"SNMP V2c Agent for Microchip TCP/IP Stack"* (DS8000870) is available for download from the Microchip web site at: http://ww1.microchip.com/downloads/en/AppNotes/00870b.pdf

Using the Library

This topic describes the basic architecture of the SNMP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `snmp.h`

The interface to the SNMP TCP/IP Stack library is defined in the `snmp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the SNMP TCP/IP Stack library should include `tcpip.h`.

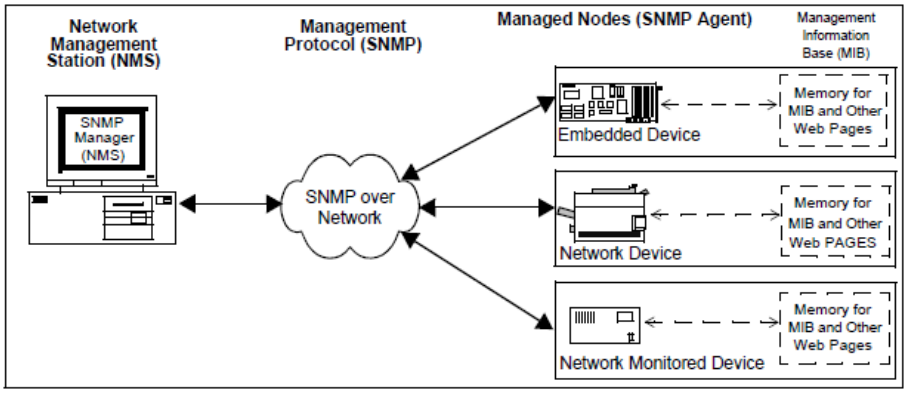
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the SNMP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

SNMP Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SNMP module.

Library Interface Section	Description
SNMP Module Functions	Routines to configure this module
SNMPv3 Module Functions	Routines to configure this module
SNMP Application Functions	Routines to configure an application
SNMPv3 Application Functions	Routines to configure an application
SNMP Data Types and Constants	This section provides various definitions describing this API
SNMPv3 Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_SNMP_BIB_FILE_NAME	The Microchip mib2bib.jar compiler is used to convert the Microchip MIB script to binary format and it is compatible with the Microchip SNMP agent. which is written in ASCII format. Name of the bib file for SNMP is snmp.bib.
	TCPIP_SNMP_COMMUNITY_MAX_LEN	This is the maximum length for community string. Application must ensure that this length is observed. SNMP module adds one byte extra after TCPIP_SNMP_COMMUNITY_MAX_LEN for adding '0' NULL character.
	TCPIP_SNMP_MAX_COMMUNITY_SUPPORT	Specifying more strings than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT will result in the later strings being ignored (but still wasting program memory). Specifying fewer strings is legal, as long as at least one is present.
	TCPIP_SNMP_MAX_MSG_SIZE	The maximum length in octets of an SNMP message which this SNMP agent able to process. As per RFC 3411 snmpEngineMaxMessageSize and RFC 1157 (section 4- protocol specification) and implementation supports more than 480 whenever feasible. It should be divisible by 16

	TCPIP_SNMP_MAX_NON_REC_ID_OID	Update the Non record id OID value which is part of CustomSnmpDemoApp.c file. This is the maximum size for gSnmpNonMibRecInfo[] which is the list of static variable Parent OIDs which are not part of mib.h file. This structure is used to restrict access to static variables of SNMPv3 OIDs from SNMPv2c and SNMPv1 version. With SNMPv3 all the OIDs accessible but when we are using SNMPv2c version , static variables of the SNMPv3 cannot be accessible with SNMP version v2c. SNMP agent supports both SMIv1 and SMIv2 standard and snmp.mib has been updated with respect to SMIV2 standard and it... more
	TCPIP_SNMP_NOTIFY_COMMUNITY_LEN	Maximum length for SNMP Trap community name
	TCPIP_SNMP_OID_MAX_LEN	Maximum length for the OID String. Change this to match your OID string length.
	TCPIP_SNMP_TASK_PROCESS_RATE	SNMP task processing rate, in milli-seconds. The SNMP module will process a timer event with this rate for processing its own state machine, etc. The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN	The maximum size of TRAP community string length
	TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE	Trap information. This macro will be used to avoid SNMP OID memory buffer corruption
	TCPIP_SNMP_TRAP_TABLE_SIZE	This table maintains list of interested receivers who should receive notifications when some interesting event occurs.
	TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN	SNMPv3 Authentication Localized password key length size
	TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE	SNMPv3 authentication localized Key length for memory validation
	TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN	SNMPv3 Privacy Password key length size
	TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE	SNMPv3 privacy key length size for memory validation
	TCPIP_SNMPV3_USER_SECURITY_NAME_LEN	Maximum size for SNMPv3 User Security Name length.
	TCPIP_SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE	User security name length for memory validation
	TCPIP_SNMPV3_USM_MAX_USER	Maximum number of SNMPv3 users. User Security Model should have at least 1 user. Default is 3.

Description

The configuration of the SNMP TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the SNMP TCP/IP Stack Library. Based on the selections made, the SNMP TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the SNMP TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_SNMP_BIB_FILE_NAME Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_BIB_FILE_NAME "snmp.bib"
```

Description

The Microchip mib2bib.jar compiler is used to convert the Microchip MIB script to binary format and it is compatible with the Microchip SNMP agent. which is written in ASCII format. Name of the bib file for SNMP is snmp.bib.

TCPIP_SNMP_COMMUNITY_MAX_LEN Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_COMMUNITY_MAX_LEN (8u)
```

Description

This is the maximum length for community string. Application must ensure that this length is observed. SNMP module adds one byte extra after TCPIP_SNMP_COMMUNITY_MAX_LEN for adding '0' NULL character.

TCPIP_SNMP_MAX_COMMUNITY_SUPPORT Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_MAX_COMMUNITY_SUPPORT (3u)
```

Description

Specifying more strings than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT will result in the later strings being ignored (but still wasting program memory). Specifying fewer strings is legal, as long as at least one is present.

TCPIP_SNMP_MAX_MSG_SIZE Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_MAX_MSG_SIZE 480
```

Description

The maximum length in octets of an SNMP message which this SNMP agent able to process. As per RFC 3411 snmpEngineMaxMessageSize and RFC 1157 (section 4- protocol specification) and implementation supports more than 480 whenever feasible. It should be divisible by 16

TCPIP_SNMP_MAX_NON_REC_ID_OID Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_MAX_NON_REC_ID_OID 3
```

Description

Update the Non record id OID value which is part of CustomSnmpDemoApp.c file. This is the maximum size for gSnmpNonMibRecInfo[] which is the list of static variable Parent OIDs which are not part of mib.h file. This structure is used to restrict access to static variables of SNMPv3 OIDs from SNMPv2c and SNMPv1 version. With SNMPv3 all the OIDs accessible but when we are using SNMPv2c version , static variables of the SNMPv3 cannot be accessible with SNMP version v2c. SNMP agent supports both SMIv1 and SMIv2 standard and snmp.mib has been updated with respect to SMIV2 standard and it also includes MODULE-IDENTITY (number 1)after ENTERPRISE-ID.

TCPIP_SNMP_NOTIFY_COMMUNITY_LEN Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_NOTIFY_COMMUNITY_LEN (TCPIP_SNMP_COMMUNITY_MAX_LEN)
```

Description

Maximum length for SNMP Trap community name

TCPIP_SNMP_OID_MAX_LEN Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_OID_MAX_LEN (18)
```

Description

Maximum length for the OID String. Change this to match your OID string length.

TCPIP_SNMP_TASK_PROCESS_RATE Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_TASK_PROCESS_RATE (200)
```

Description

SNMP task processing rate, in milli-seconds. The SNMP module will process a timer event with this rate for processing its own state machine, etc. The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN (TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE+1)
```

Description

The maximum size of TRAP community string length

TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE (8)
```

Description

Trap information. This macro will be used to avoid SNMP OID memory buffer corruption

TCPIP_SNMP_TRAP_TABLE_SIZE Macro

File

snmp_config.h

C

```
#define TCPIP_SNMP_TRAP_TABLE_SIZE (2)
```

Description

This table maintains list of interested receivers who should receive notifications when some interesting event occurs.

TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN (20)
```

Description

SNMPv3 Authentication Localized password key length size

TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE
(TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN+1)
```

Description

SNMPv3 authentication localized Key length for memory validation

TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN (20)
```

Description

SNMPv3 Privacy Password key length size

TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE
(TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN+1)
```

Description

SNMPv3 privacy key length size for memory validation

TCPIP_SNMPV3_USER_SECURITY_NAME_LEN Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_USER_SECURITY_NAME_LEN (16)
```

Description

Maximum size for SNMPv3 User Security Name length.

TCPIP_SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE (TCPIP_SNMPV3_USER_SECURITY_NAME_LEN+1)
```

Description

User security name length for memory validation

TCPIP_SNMPV3_USM_MAX_USER Macro

File

snmpv3_config.h

C

```
#define TCPIP_SNMPV3_USM_MAX_USER 3
```

Description

Maximum number of SNMPv3 users. User Security Model should have at least 1 user. Default is 3.

Building the Library

This section lists the files that are available in the SNMP module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/snmp.c`	SNMP implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The SNMP module depends on the following modules:

- TCP/IP Stack Library

- DNS Module
- UDP Module

SNMP Server (Agent)

This section describes the SNMP Server Agent.

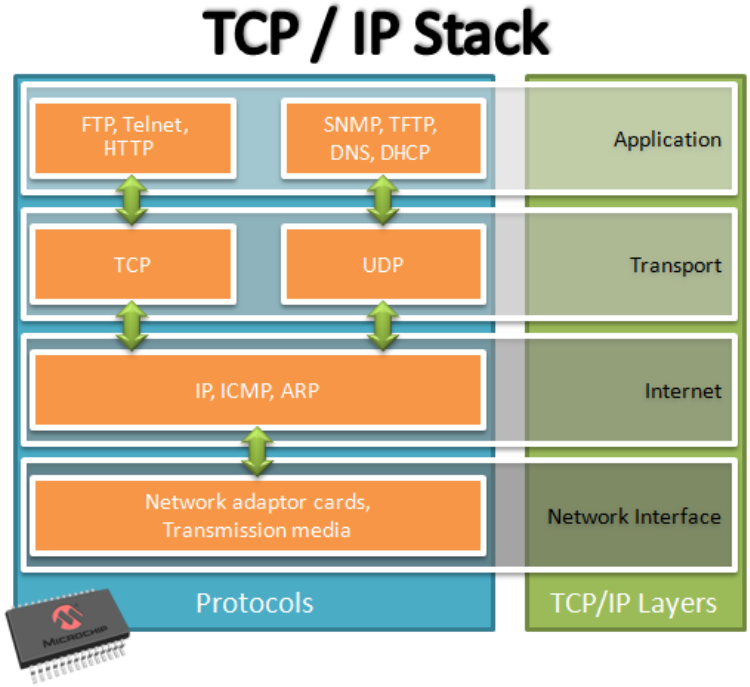
Introduction

The Simple Network Management Protocol (SNMP) is one of the key components of a Network Management System (NMS). SNMP is an application layer protocol that facilitates the exchange of management information among network devices. It is a part of the TCP/IP protocol suite.

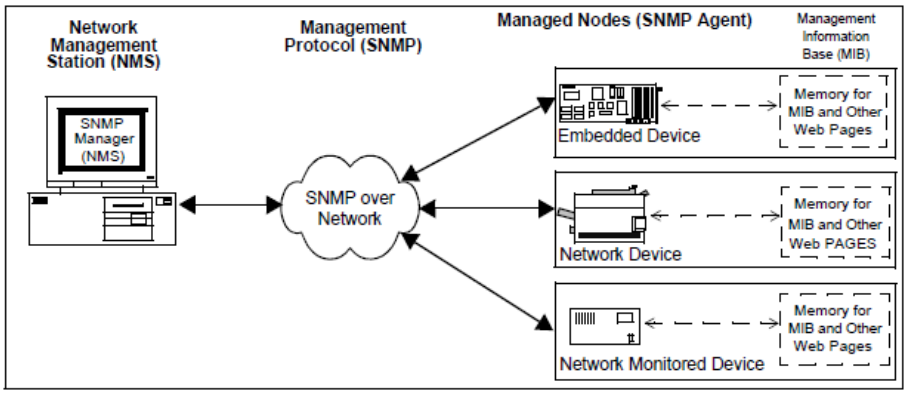
Description

SNMP is an Internet protocol that was originally designed to manage different network devices, such as file servers, hubs, routers, and so on. It can also be used to manage and control an ever increasing number of small embedded systems connected to one another over any IP network. Systems can communicate with each other using SNMP to transfer control and status information, creating a truly distributed system.

Location of the SNMP Stack in the TCP/IP Protocol Stack



Overview of the SNMP Model



The Microchip SNMP Server is a multi-lingual implementation, which supports SNMPv1, V2c, and V3 server features simultaneously. The SNMP Server is implemented to address the requirements of embedded applications and works with both IPv4 and IPv6 addresses. SNMPv1 and V2c are enabled by TCPIP_STACK_USE_SNMP_SERVER. SNMPv3 support is enabled by TCPIP_STACK_USE_SNMPV3_SERVER. Since the SNMPv3 stack requires the support of SNMPv1 and SNMPv2c, TCPIP_STACK_USE_SNMPV3_SERVER should be enabled with TCPIP_STACK_USE_SNMP_SERVER.

Note: Currently, the SNMP agent will be able to communicate with one manager at a time, and it will be connected to either an IPv4 address or an IPv6 address.

V2c

V2c is implemented with support for the configuration of multiple community names and the community names can be configured through the HTTP web interface. An access-restricted web page is provided with the demonstration application to allow dynamic configuration of SNMP communities.

SNMPv3

SNMPv3 RFC specifies different types of access mechanisms, user security model (USM), authentication and privacy protocols. The Microchip SNMPv3 Server is implemented with support for the AES 128 CFB 128 and DES-CBC privacy protocols, and the MD5 and SHA1 message authentication protocols. The demonstration implementation of the server is configured with three types of user names with respective authentication and privacy credentials and authentication types. These credentials and other user information are stored in the global array. The user of the SNMPv3 stack can decide on the number of user names in the User's database to be stored with the SNMPv3 Server. According to the SNMPv3 recommendation, the SNMPv3 Server should not be configured with the authentication and privacy passwords. Instead, it could be configured with the respective localized keys of the password. The Microchip SNMPv3 agent is provided with the password information in the database for "Getting Started" and for understanding purposes only. It is recommended that the SNMPv3 stack be modified to restrict access to the password OIDs declared in the user database.

Note: Even though SNMPv3 also requires SNMPv1 and SNMPv2c, a layer in the SNMP Stack will prevent access to the variables that should be secured by SNMPv3. SNMP variables are structures in a tree in the MIB. Access to parts of this tree is determined by version. For example, SYSTEM-type variables can be accessed regardless of the SNMP version, while SNMPv2c requests can access part of the tree, and authenticated SNMPv3 requests can access the complete tree.

MIB Files

Provides information the Management Information Base (MIB) files.

Description

SNMP describes the hierarchal storage of management objects (referred to with object IDs or OIDs) with MIB files. The Microchip SNMP server demonstration includes two MIB files:

- `mchip.mib` - This is an Abstract Syntax Notation One (ASN.1) formatted MIB file containing information about the variables used in the demonstration
- `snmp.mib` - This is a custom-formatted file that can be parsed to create Web page and header resources that can be accessed with a PIC microcontroller

The TCP/IP Stack includes the mib2bib utility, which will compile the custom Microchip MIB script (`snmp.mib`) to generate two files named snmp.bib and `mib.h`. The snmp.bib file is a compressed record of management objects that will be stored with web pages and the `mib.h` file contains C defines for each OID. These files are included in the appropriate directories for the TCP/IP Demonstration Applications, but for a custom application you must copy snmp.bib to your web page directory, copy `mib.h` to your application directory and include it in your project, rebuild your project, and then rebuild and reupload your web page. This will bundle the BIB file into your web page image, which will allow the SNMP agent to search for the required variable information with the MPFS file system.

MIB Browsers

Provides information the available MIB browsers.

Description

Several SNMP MIB browsers are available. Users can also install a customized MIB browser specific to their application.

This documentation describes how to use the iREASONING Networks MIB Browser to run the TCP/IP SNMP demonstration applications. The MIB Browser can be obtained from: http://www.ireasoning.com/downloadmibbrowserlicense.shtml. The MIB script upload, the MIB tree structure display, and the SNMP query mechanism procedures vary from browser to browser.

Important! The use of a MIB browser or other third-party products may require that users review and agree to the terms of a license. Microchip's reference to the iREASONING Networks MIB Browser is for the users' convenience. It is the user's responsibility to obtain information about, and comply with the terms of, any applicable licenses.

Refer to the Microchip application note, AN870 "SNMP V2c Agent for Microchip TCP/IP Stack" (DS00000870) for more details on the MIB scripts, community names, and demonstration SNMP MIB variable tree structure.

The ASN.1 format `mchip.mib` file is defined with a private variable tree structure for the MIB variables. Also the `mchip.mib` is added with the number of OIDs that could be accessed only with SNMPv3 request. The browser can access every variable in the MIB database provided the community name matches. The access to the MIB variables is restricted to the type of the request. The RFC1213 MIB variables could be accessed with SNMPv2c/v3 request. But the SNMP-FRAMEWORK-MIB.mib variables could only be accessed with a SNMPv3 request if the credentials are matched and the message is authenticated. To modify these MIB variables, corresponding changes must be made to both MIB

scripts (`snmp.mib` and `mchip.mib`).

- For SNMP V2c services , The V2c agent is configured by default with three Read communities ("public", "read", "") and three Write communities ("private","write","public")
- At run-time, the community names can be dynamically configured using the HTTP interface for SNMP community name configuration
- If the V2c agent receives an SNMP request with an unknown community name, the agent will generate an Authentication trap

The V2c agent's multiple community support feature enables the user application to provide limited access to the requesting browser based on the community name used by the browser to access the MIB database variables of the agent.

For SNMPv3 services:

Type	USER 1	USER 2	USER 3
USM User	microchip	SnmpAdmin	root
Security Level	auth, priv	auth, no priv	no auth, no priv
Auth Algorithm	MD5	SHA1	N/A
Auth Password	auth12345	ChandlerUS	N/A
Privacy Algorithm	AES	N/A	N/A
Privacy Password	priv12345	N/A	N/A

If SNMPv3 services are required, SNMPv3 browser is required to be configured with the user name, authentication and privacy password, message authentication hash type, privacy protocol type. The SNMP server would respond only if one of the user credentials and user security parameters in the following table is configured at the manager. The previous table is stored in the global structure with the SNMPv3 server stack. The SNMPv3 server would only respond if the request credentials of the MIB browser matches to that of the stored user data base of the SNMP server.

The Microchip SNMPv3 stack does support only one Context Engine ID with the server. Leave the "Context Name" option in the "Advanced" tab empty. It is ignored on the server.

According to the user and the auth and privacy protocols configured with the SNMP browser, the UDP authenticated and encrypted message would be exchanged between server and the client.

- If the USER 1 values, as shown in the table, are configured in the MIB browser, the data exchange between the client and server is encrypted and authenticated. The PDU could be captured in the Ethernet packet sniffer, such as WireShark, and examined. As the data is encrypted and authenticated, the data integrity and the privacy is achieved.
- If USER 2 values, as shown in the table, are configured in the MIB browser, the data exchange between client and server is authenticated. The data integrity would be checked once the data is received at either end. The message authentication mechanism protects from the possible data sniffing and modification threat, and also guarantees that the data is received from the authenticated and guaranteed source.
- If USER 3 values, as shown in the table, are configured in the MIB browser, the data exchange between client and server is neither authenticated nor encrypted
- Considering the three USER configurations, if the SNMP server is to be accessed over WAN, in the Internet cloud, the data should be encrypted and authenticated to have the highest level of data privacy and integrity

SNMP Traps

Provides information on SNMP traps.

Description

The MPLAB Harmony TCP/IP Stack supports Trap version1 and Trap version2 formatted traps. Traps are notifications from the agent to the manager that are used when a predefined event occurs at the agent.

From `mchip.mib`, the ipv4TrapTable and ipv6TrapTable are the two tabular sections, which are used to configure HOST IPv4 and IPv6 address and this will help the HOST trap receiver to receive the traps.

Several preprocessor macros in the `snmp_config.h` variant header file can be used to enable or disable traps in the agent. Commenting and uncommenting these macros in the file will have different results. The SNMP_TRAP_DISABLED macro will disable traps entirely if it is not commented:

```
#define SNMP_TRAP_DISABLED
```

The user must configure the expected trap format at the SNMP Manager. SNMPv2 entities acting as an agent should be able to generate and transmit SNMP V2 trap PDUs when the manager is configured to receive and process SNMP V2 trap PDUs. To configure the trap format, comment or uncomment the SNMP_STACK_USE_V2_TRAP macro in the `snmp_config.h` header file:

```
#define SNMP_STACK_USE_V2_TRAP
```

If the macro has been commented out, the SNMP agent will send V1 formatted trap PDUs; otherwise, it will send V2 formatted trap PDUs. By default, the SNMP agent is configured to send V2 formatted traps. Note that the SNMP V2c agent should only send V2 formatted traps.

To enable traps in SNMPv3, the `#define SNMP_V1_V2_TRAP_WITH_SNMPv3` macro must be uncommented.

The following table illustrates how to enable/disable traps for different versions of SNMP.

Type	SNMPv1	SNMPv2c	SNMPv3
TRAPv2 (enabled by default)	Comment out the `#define SNMP_TRAP_DISABLED` macro	Comment out the `#define SNMP_TRAP_DISABLED` macro	Comment out the `#define SNMP_TRAP_DISABLED` macro Uncomment the `#define SNMP_V1_V2_TRAP_WITH_SNMPV3` macro
TRAPv2 (disabled by default)	Not supported	Comment out the `#define SNMP_TRAP_DISABLED` macro Uncomment the `#define SNMP_STACK_USE_V2_TRAP` macro	Comment out the `#define SNMP_TRAP_DISABLED` macro Uncomment the `#define SNMP_V1_V2_TRAP_WITH_SNMPV3` macro Uncomment the `#define SNMP_STACK_USE_V2_TRAP` macro

Demonstrations

Two trap demonstrations APIs are included with the TCP/IP Stack. The task functions for these demonstrations are called in the main application function:

- SNMPTrapDemo - This API demonstrates V1 or V2 trap formats (depending of the status of the SNMP_STACK_USE_V2_TRAP macro). The trap PDU will only have one demonstration variable binding on the varbind list.
- SNMPV2TrapDemo - This API provides V2 format notifications with multiple (4) variable bindings. The user should modify or use this routine as a reference for sending V2 trap format notifications with multiple bindings on the varbind list.

Note: The user should only enable one SNMP demonstration API at a time. By default, the SNMPV2TrapDemo API is enabled and SNMPTrapDemo is commented out (disabled).

V1/V2 Formatted Traps with a Single Variable Binding

In the snmp_config.h header file:

- Uncomment `#define SNMP_TRAP_DISABLED`
- Comment `//#define SNMP_STACK_USE_V2_TRAP`

For the Trap demonstration, two events are defined within the V2c agent:

- If the Analog Potentiometer value is greater than 14, the agent will send a Trap every 5 seconds to the configured 'IPv4TrapReceiverIP address'.
- If Button 3 on the demonstration board is pressed, an organization-specific PUSH_BUTTON trap will be sent.

The current implementation of the V2c agent also generates a standard "Authentication Failure Trap":

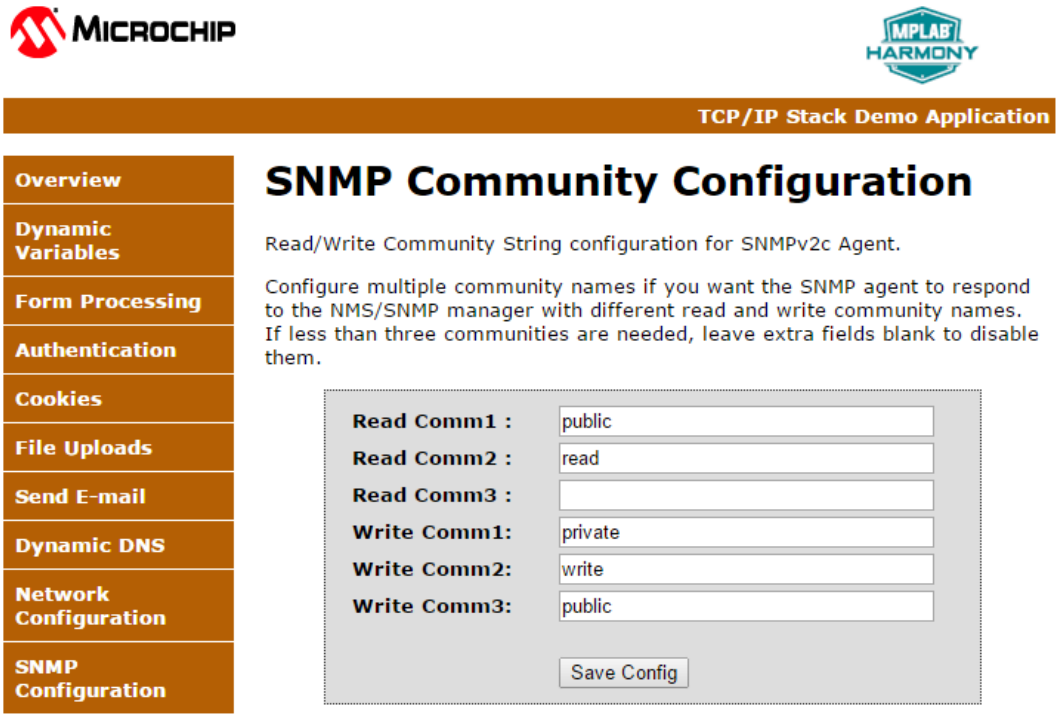
- If a request is received to modify (Set) a private MIB variable, or
- If the value of the variable is requested (Get) by a browser with the wrong community name

HTTP Configuration

Provides information on the SNMP Community Configuration.

Description

If a HTTP2 server is used with the Microchip TCP/IP stack, it is possible to dynamically configure the Read and Write community names through the SNMP Configuration web page. Access the web page using http://mchpboard_e/mpfsupload or http://<Board IP address>(for IPv6 it should be http://<Ipv6 address>:80/index.html), and then access the SNMP Configuration web page through the navigation bar. Use "admin" for the username and "microchip" for the password.



Library Interface

a) SNMP Module Functions

	Name	Description
	TCPIP_SNMP_NotifyIsReady	Resolves given remoteHost IP address into MAC address.
	TCPIP_SNMP_NotifyPrepare	Collects trap notification info and send ARP to remote host.
	TCPIP_SNMP_TrapTimeGet	Gets SNMP Trap UDP client open socket time-out.
	TCPIP_SNMP_ClientGetNet	Get a network interface for SNMP TRAP.
	TCPIP_SNMP_ExactIndexGet	To search for exact index node in case of a Sequence variable.
	TCPIP_SNMP_IsValidCommunity	Validates community name for access control.
	TCPIP_SNMP_IsValidLength	Validates the set variable data length to data type.
	TCPIP_SNMP_MibIDSet	Sets the agent MIB ID for SNP notification.
	TCPIP_SNMP_NextIndexGet	To search for next index node in case of a Sequence variable.
	TCPIP_SNMP_ReadCommunityGet	Gets the readCommunity String with SNMP index.
	TCPIP_SNMP_RecordIDValidation	Used to restrict the access dynamic and non dynamic OID string for a particular SNMP Version.
	TCPIP_SNMP_SendFailureTrap	Prepares and validates the remote node that will receive a trap and send the trap PDU.
	TCPIP_SNMP_TrapInterFaceSet	Sets the TRAP interface for SNMP notification.
	TCPIP_SNMP_TRAPMibIDGet	Gets the agent MIB ID for SNP notification.
	TCPIP_SNMP_TrapSendFlagGet	Gets the status of trap send flag.
	TCPIP_SNMP_TrapSendFlagSet	Sets the status of trap send flag.
	TCPIP_SNMP_TrapSpecificNotificationGet	Gets the specific trap.
	TCPIP_SNMP_TrapSpecificNotificationSet	Sets the specific trap, generic trap, and trap ID.
	TCPIP_SNMP_VarbindGet	Used to get/collect OID variable information.
	TCPIP_SNMP_VarbindSet	Sets the MIB variable with the requested value.
	TCPIP_SNMP_WriteCommunityGet	Gets the writeCommunity String with SNMP index.
	TCPIP_SNMP_AuthTrapFlagGet	Gets the status of authentication trap flag.
	TCPIP_SNMP_AuthTrapFlagSet	Sets the status of authentication trap flag.
	TCPIP_SNMP_IsTrapEnabled	Gets the SNMP Trap status.
	TCPIP_SNMP_TRAPTypeGet	Get SNMP Trap type for version v1 and v2c.
	TCPIP_SNMP_TRAPv1Notify	Creates and Sends TRAPv1 pdu.
	TCPIP_SNMP_TRAPv2Notify	Creates and sends TRAP PDU.

	TCPIP_SNMPV3_TrapTypeGet	Gets SNMP Trap type for version v3.
	TCPIP_SNMP_ValidateTrapIntf	Gets the status of trap interface.
	TCPIP_SNMP_ReadCommunitySet	Sets the readCommunity String with SNMP index.
	TCPIP_SNMP_WriteCommunitySet	Sets the writeCommunity String with SNMP index.
	TCPIP_SNMP_SocketIDGet	Gets the Socket ID for SNMP Server socket.
	TCPIP_SNMP_SocketIDSet	Sets the Socket ID for SNMP Server socket.
	TCPIP_SNMP_Task	Standard TCP/IP stack module task function.

b) SNMPv3 Module Functions

	Name	Description
	TCPIP_SNMPV3_EngineUserDataBaseGet	Get SNMPv3 engine data base details.
	TCPIP_SNMPV3_EngineUserDataBaseSet	Set SNMPv3 engine data base details.
	TCPIP_SNMPv3_TrapConfigDataGet	Gets the SNMPv3 Trap configuration details using the user name index.
	TCPIP_SNMPv3_Notify	Creates and Sends SNMPv3 TRAP PDU.

c) SNMP Data Types and Constants

	Name	Description
	SNMP_END_OF_VAR	This Macro is used for both SNMP SET and GET Variable processing to indicate the end of SNMP variable processing. For multi byte data request, the end byte will be always SNMP_END_OF_VAR.
	SNMP_INDEX_INVALID	This Macro is used for both SNMP SET and GET Sequence Variable processing. SNMP starts processing the start of sequence variable with Invalid index. TCPIP_SNMP_ExactIndexGet and TCPIP_SNMP_NextIndexGet returns a valid index as per SNMP_INDEX_INVALID.
	SNMP_START_OF_VAR	This Macro is used for both SNMP SET and GET Variable processing to indicate the start of SNMP variable processing. For multi byte data request, the first byte will be always SNMP_START_OF_VAR.
	SNMP_V1	This macro is used for SNMP version 1
	SNMP_V2C	This macro is used for SNMP version 2 with community
	SNMP_V3	This macro is used for SNMP version 3 with authentication and privacy
	SNMP_COMMUNITY_TYPE	Definition to represent different type of SNMP communities.
	SNMP_ID	SNMP dynamic variable ID.
	SNMP_INDEX	SNMP sequence variable index.
	SNMP_NON_MIB_RECD_INFO	Restrict access for specific OIDs.
	SNMP_TRAP_IP_ADDRESS_TYPE	Definition of the supported address types for SNMP trap.
	SNMP_VAL	Definition to represent SNMP OID object values.
	SNMP_GENERIC_TRAP_NOTIFICATION_TYPE	Definition to represent different SNMP generic trap types.
	SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE	Definition to represent different SNMP vendor trap types.

d) SNMPv3 Data Types and Constants

	Name	Description
	SNMPV3_HMAC_HASH_TYPE	Different type of authentication for SNMPv3.
	SNMPV3_PRIV_PROT_TYPE	Different type of encryption and decryption for SNMPv3.
	STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL	Different SNMP Message processing model
	STD_BASED_SNMP_SECURITY_MODEL	Different Security services for SNMPv3 messages.
	STD_BASED_SNMPV3_SECURITY_LEVEL	Different Security Level for SNMPv3 messages.
	TCPIP_SNMPV3_USERDATABASECONFIG_TYPE	Different Configuration parameters of SNMPv3 operation
	TCPIP_SNMP_COMMUNITY_CONFIG	SNMP community configuration.
	TCPIP_SNMP_MODULE_CONFIG	SNMP module configuration.
	TCPIP_SNMPV3_TARGET_ENTRY_CONFIG	SNMP module trap target address configuration.
	TCPIP_SNMPV3_USM_USER_CONFIG	SNMPv3 USM configuration.

Description

This section describes the Application Programming Interface (API) functions of the SNMP module.

Refer to each section for a detailed description.

a) SNMP Module Functions

TCPIP_SNMP_NotifyIsReady Function

Resolves given remoteHost IP address into MAC address.

File

snmp.h

C

```
bool TCPIP_SNMP_NotifyIsReady(IP_MULTI_ADDRESS* remoteHost, SNMP_TRAP_IP_ADDRESS_TYPE
eTrapMultiAddressType);
```

Description

This function resolves given remoteHost IP address into MAC address using ARP module. If remoteHost is not available, this function would never return true. Application must implement time out logic to handle "remoteHost not available" situation.

Remarks

This would fail if there were not UDP socket to open.

Preconditions

TCPIP_SNMP_NotifyPrepare is already called.

Parameters

Parameters	Description
remoteHost	Pointer to remote Host IP address
eTrapMultiAddressType	IPv4 and IPv6 address type

Function

bool TCPIP_SNMP_NotifyIsReady(IP_MULTI_ADDRESS* remoteHost,
SNMP_TRAP_IP_ADDRESS_TYPE eTrapMultiAddressType)

TCPIP_SNMP_NotifyPrepare Function

Collects trap notification info and send ARP to remote host.

File

snmp.h

C

```
void TCPIP_SNMP_NotifyPrepare(IP_MULTI_ADDRESS* remoteHost, char* community, uint8_t communityLen, SNMP_ID
agentIDVar, uint8_t notificationCode, uint32_t timestamp);
```

Returns

None.

Description

This function prepares SNMP module to send SNMP trap notification to remote host. It sends ARP request to remote host to learn remote host MAC address.

Remarks

This is first of series of functions to complete SNMP notification.

Preconditions

TCPIP_SNMP_Initialize() is already called.

Parameters

Parameters	Description
remoteHost	pointer to remote Host IP address

community	Community string to use to notify
communityLen	Community string length
agentIDVar	System ID to use identify this agent
notificaitonCode	Notification Code to use
timestamp	Notification timestamp in 100th of second.

Function

void TCPIP_SNMP_NotifyPrepare(IP_MULTI_ADDRESS* remoteHost,
char* community,
uint8_t communityLen,
SNMP_ID agentIDVar,
uint8_t notificationCode,
uint32_t timestamp)

TCPIP_SNMP_TrapTimeGet Function

Gets SNMP Trap UDP client open socket time-out.

File

snmp.h

C

```
uint32_t TCPIP_SNMP_TrapTimeGet();
```

Returns

uint32_t time

Description

This function returns a uint32_t time(snmpTrapTimer) which is used to time out a SNMP TRAP notification for a HOST. snmpTrapTimer is initialized when there is UDP client socket open either for a HOST IPv4 or IPv6 address.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize() is already called.

Function

uint32_t TCPIP_SNMP_TrapTimeGet(void)

TCPIP_SNMP_ClientGetNet Function

Get a network interface for SNMP TRAP.

File

snmp.h

C

```
TCPIP_NET_HANDLE TCPIP_SNMP_ClientGetNet(int * netIx, TCPIP_NET_HANDLE hNet);
```

Returns

TCPIP_NET_HANDLE

- Success - returns a valid interface
- Failure - no interface

Description

This function is used to get a network interface to transmit SNMP trap.

Remarks

None.

Preconditions

The SNMP module should be initialized.

Parameters

Parameters	Description
netIx	Network index
hNet	Network interface .If hNet is NULL, then a valid interface will returned.

Function

TCPIP_NET_HANDLE TCPIP_SNMP_ClientGetNet(int *netIx,TCPIP_NET_HANDLE hNet);

TCPIP_SNMP_ExactIndexGet Function

To search for exact index node in case of a Sequence variable.

File

snmp.h

C

```
bool TCPIP_SNMP_ExactIndexGet(SNMP_ID var, SNMP_INDEX * index);
```

Returns

-true - If the exact index value exists for a given variable at a given index. -false - If the exact index value does not exist for the given variable

Description

This is a callback function called by SNMP module. This function contents are modified by the application developer with the new MIB Record ID. This function can be called for individual MIB ID with index. This function is used to get the first available index from global or dynamic allocated sequence variable table.

Remarks

Only sequence index needs to be handled in this function and this function is called after TCPIP_SNMP_RecordIDValidation.

Preconditions

TCPIP_SNMP_ProcessVariables() is called.

Parameters

Parameters	Description
var	Variable id as per mib.h (input)
index	Index of variable (input)

Function

bool TCPIP_SNMP_ExactIndexGet(SNMP_ID var,SNMP_INDEX *index)

TCPIP_SNMP_IsValidCommunity Function

Validates community name for access control.

File

snmp.h

C

```
uint8_t TCPIP_SNMP_IsValidCommunity(uint8_t* community);
```

Returns

This function returns the community validation result as READ_COMMUNITY or WRITE_COMMUNITY or INVALID_COMMUNITY.

Description

This function validates the community name for the MIB access to SNMP manager. The SNMP community name received in the request PDU is validated for read and write community names. The agent gives an access to the mib variables only if the community matches with the predefined values. This routine also sets a global flag to send trap if authentication failure occurs.

Remarks

This is a callback function called by module. User application must implement this function and verify that community matches with predefined value. This validation occurs for each SNMP manager request.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
community	Pointer to community string as sent by SNMP manager

Function

uint8_t TCPIP_SNMP_IsValidCommunity(uint8_t* community)

TCPIP_SNMP_IsValidLength Function

Validates the set variable data length to data type.

File

snmp.h

C

```
bool TCPIP_SNMP_IsValidLength(SNMP_ID var, uint8_t len, uint8_t index);
```

Returns

- true - if given var can be set to given len
- false - if the given var cannot be set to the given len

Description

This function is used to validate the dynamic variable data length to the variable data type. It is called before the SET request is processed. This is a callback function called by module. User application must implement this function.

Remarks

This function will be called for only dynamic variables that are defined as ASCII_STRING and OCTET_STRING.

Preconditions

TCPIP_SNMP_ProcessSetVar is called.

Parameters

Parameters	Description
var	Variable id whose value is to be set
len	Length value that is to be validated.

Function

bool TCPIP_SNMP_IsValidLength(SNMP_ID var, uint8_t len,uint8_t index)

TCPIP_SNMP_MibIDSet Function

Sets the agent MIB ID for SNP notification.

File

snmp.h

C

```
void TCPIP_SNMP_MibIDSet(uint32_t mibID);
```

Returns

None.

Description

This function is used to Set the MIB ID which is require while transmitting SNMP notification.SNMP user can this function without adding this to the Microchip MIB script.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
mibID	Trap client MIB ID

Function

void TCPIP_SNMP_MibIDSet(uint32_t *mibID)

TCPIP_SNMP_NextIndexGet Function

To search for next index node in case of a Sequence variable.

File

snmp.h

C

```
bool TCPIP_SNMP_NextIndexGet(SNMP_ID var, SNMP_INDEX* index);
```

Returns

- true - If a next index value exists for given variable at given index and index parameter contains next valid index.
- false - If a next index value does not exist for a given variable

Description

This is a callback function called by SNMP module. This function contents are modified by the application developer with the new MIB Record ID. This function can be called for individual MIB ID with index or for a complete MIB table without any instance.

- If the SNMP Sequence MIB variable processing is performed with index or instance value,

it is the responsibility of the agent to send the next available index for that requested variable. This is a only one iteration process. The Manager will not send any further requests for the received index value.

- If SNMP request for a sequence variable starts with only OID without any instance, that is

a complete table is requested without any instance, it is the responsibility of the agent to send the first available index of the table. The Manager will continue the request with the transferred instance until it receives the reply from agent.

This function will only be called for OID variable of type sequence. if the index value starts with SNMP_INDEX_INVALID , then user need to send the response with first available index value.

Remarks

Only sequence index needs to be handled in this function and this function is called after TCPIP_SNMP_RecordIDValidation.

Preconditions

TCPIP_SNMP_ProcessVariables is called.

Parameters

Parameters	Description
var	Variable id whose value is to be returned
index	Next Index of variable that should be transferred

Function

bool TCPIP_SNMP_NextIndexGet(SNMP_ID var,SNMP_INDEX* index)

TCPIP_SNMP_ReadCommunityGet Function

Gets the readCommunity String with SNMP index.

File

snmp.h

C

```
bool TCPIP_SNMP_ReadCommunityGet(int index, int len, uint8_t * dest);
```

Returns

- true - if the community string is collected
- false - if the community string is not collected

Description

This function is used to collect READ community string from the global community table with respect to the index value.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
index	One of the index of community table and it should be less than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT.
len	Length of the community string expected. It should not be more than TCPIP_SNMP_COMMUNITY_MAX_LEN.
dest	Copy the community string to this address and it should have a valid address.

Function

bool TCPIP_SNMP_ReadCommunityGet(int index,int len, uint8_t * dest)

TCPIP_SNMP_RecordIDValidation Function

Used to restrict the access dynamic and non dynamic OID string for a particular SNMP Version.

File

snmp.h

C

```
bool TCPIP_SNMP_RecordIDValidation(uint8_t snmpVersion, bool idPresent, uint16_t varId, uint8_t *
oidValuePtr, uint8_t oidLen);
```

Returns

- true - A record ID exists
- false - A record ID does not exist

Description

This is a callback function called by SNMP module. SNMP user must implement this function as per SNMP version. One need to add the new SNMP MIB IDs here as per SNMP version (e.g., SYS_UP_TIME (250) is common for V1/V2/V3). ENGINE_ID - is the part of V3; therefore, place all of the SNMPv3 var IDs within the macro TCPIP_STACK_USE_SNMPV3_SERVER.

Remarks

This function is specific for record ID validation and this can also be used to restrict OID string.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
snmpVersion	different SNMP version
idPresent	true if SNMp record id is present else false
varId	dynamic record ID value as per mib.h

oidValuePtr	OID Value pointer
oidLen	number of OIDs present to be processed

Function

bool TCPIP_SNMP_RecordIDValidation(uint8_t snmpVersion,bool idPresent,
uint16_t varId,uint8_t * oidValuePtr,uint8_t oidLen)

TCPIP_SNMP_SendFailureTrap Function

Prepares and validates the remote node that will receive a trap and send the trap PDU.

File

snmp.h

C

```
void TCPIP_SNMP_SendFailureTrap();
```

Returns

None.

Description

This function is used to send trap notification to previously configured IP address if trap notification is enabled. There are different trap notification code. The current implementation sends trap for authentication failure (4).

Remarks

This is a callback function called by the application on certain predefined events. This routine only implemented to send a authentication failure Notification-type macro with PUSH_BUTTON OID stored in MPFS. If the ARP is not resolved (i.e., if TCPIP_SNMP_NotifyIsReady returns false, this routine times out in 5 seconds). This function should be modified according to event occurred and should update the corresponding OID and notification type to the TRAP PDU.

Preconditions

If the defined application event occurs to send the trap.

Function

void TCPIP_SNMP_SendFailureTrap(void)

TCPIP_SNMP_TrapInterFaceSet Function

Sets the TRAP interface for SNMP notification.

File

snmp.h

C

```
void TCPIP_SNMP_TrapInterFaceSet(TCPIP_NET_HANDLE netIntf);
```

Returns

None.

Description

This function is used to Set the network interface to which the TRAP socket is ready to transmit Notifications to the TRAP receiver address.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
netIntf	Network interface Trap is connected to receiver

Function

void TCPIP_SNMP_TrapInterFaceSet(TCPIP_NET_HANDLE netIntf)

TCPIP_SNMP_TRAPMibIDGet Function

Gets the agent MIB ID for SNP notification.

File

snmp.h

C

```
void TCPIP_SNMP_TRAPMibIDGet(uint32_t * mibID);
```

Returns

None.

Description

This function is used to get the MIB ID which is require while transmitting SNMP notification.This MIB ID is a Agent ID value of the Microchip style MIB script. MIB ID macro value is present in mib.h which is generated by mib2bib.jar.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
mibID	Trap client MIB ID

Function

void TCPIP_SNMP_TRAPMibIDGet(uint32_t *mibID)

TCPIP_SNMP_TrapSendFlagGet Function

Gets the status of trap send flag.

File

snmp.h

C

```
void TCPIP_SNMP_TrapSendFlagGet(bool * trapNotify);
```

Returns

None.

Description

This function is used to get the trap send flag details and this is used only when user is trying to send more than one varbind in a single PDU. That is more than one notification details are required to be part of a single PDU.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Function

void TCPIP_SNMP_TrapSendFlagGet(bool *trapNotify)

TCPIP_SNMP_TrapSendFlagSet Function

Sets the status of trap send flag.

File

snmp.h

C

```
void TCPIP_SNMP_TrapSendFlagSet(bool trapNotify);
```

Returns

None.

Description

This function is used to set the trap send flag details and this is used only when user is trying to send more than one varbind in a single PDU. That is more than one notification details are required to be part of a single PDU. By default TRAP send flag is set to false. If there is only varbind need to be part of Notification PDU, then this function should be called with boolean false. Please find the usage of this flag in this following example.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Example

```
void SNMPv2TrapDemo(void)
{
    //set TRAP send flag to true , it signifies that there are more than one
    // variable need to be the part of SNMP v2 TRAP.
    TCPIP_SNMP_TrapSendFlagSet(true);

    // PUSH button varbind
    TCPIP_SNMP_Notify(PUSH_BUTTON,analogPotVal,0);

    // Before adding the last varbind to the TRAP PDU, TRAP send flag should
    // be set to  false. That it indicates it is the last varbind to the
    // Notification PDU.
    TCPIP_SNMP_TrapSendFlagSet(false);

    // Last varbind LED0_IO
    TCPIP_SNMP_Notify(LED0_IO,analogPotVal,0);

}
```

Function

void TCPIP_SNMP_TrapSendFlagSet(bool trapNotify)

TCPIP_SNMP_TrapSpecificNotificationGet Function

Gets the specific trap.

File

snmp.h

C

```
void TCPIP_SNMP_TrapSpecificNotificationGet(uint8_t * specTrap);
```

Returns

None.

Description

This function is used to get specific trap value. Specific trap values are listed in SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
specTrap	Vendor specific trap value

Function

void TCPIP_SNMP_TrapSpecificNotificationGet(uint8_t *specTrap)

TCPIP_SNMP_TrapSpecificNotificationSet Function

Sets the specific trap, generic trap, and trap ID.

File

snmp.h

C

```
void TCPIP_SNMP_TrapSpecificNotificationSet(uint8_t specTrap, uint8_t genTrap, SNMP_ID trapID);
```

Returns

None.

Description

The SNMP user needs to call this function before transmitting any traps. This function will set the vendor specific trap, generic trap, and default trap ID value. The SNMPv2 trap will use this trap ID while sending a specific trap.

Remarks

The Trap ID is the NOTIFICATION-TYPE of the ASN.1 MIB format. From the Trap ID, the SNMP agent will be able to obtain the OID string, which will be used while preparing TRAPv2 second varbind.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
specTrap	Vendor specific trap value (enumeration value of SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE)
genTrap	Generic trap (enumeration value of SNMP_GENERIC_TRAP_NOTIFICATION_TYPE)
trapID	Trap ID

Function

void TCPIP_SNMP_TrapSpecificNotificationSet(uint8_t specTrap,uint8_t genTrap,
SNMP_ID trapID)

TCPIP_SNMP_VarbindGet Function

Used to get/collect OID variable information.

File

snmp.h

C

```
bool TCPIP_SNMP_VarbindGet(SNMP_ID var, SNMP_INDEX index, uint8_t* ref, SNMP_VAL* val);
```

Returns

- true - If a value exists for given variable

- false - If a value does not exist for a given variable

Description

This is a callback function called by SNMP module. This function is called only when SNMP GET, GETNEXT and GETBULK request is made. This function content is modified by the application developer. This function contains the implementation of READ enable MIB OID macros.

mib2bib.jar Microchip MIB compiler utility is used to generate mib.h, which lists all the MIB OID ID value.

Remarks

This function may get called more than once depending on number of bytes in a specific get request for given variable. only dynamic read variables needs to be handled.

Preconditions

TCPIP_SNMP_ProcessVariables() is called.

Example

Usage of MIB OID within this function for a ASCII string (multi byte variable)

```
// In the beginning *ref is equal to SNMP_START_OF_VAR
myRef = *ref;
switch(var)
{
// TRAP_COMMUNITY - generated from the Microchip style MIB script using mib2bib.jar,
// is a Sequence variable.
    case TRAP_COMMUNITY:
        if ( index < trapInfo.Size )
        {
        // check if the myRef should not cross the maximum length
             if ( myRef == trapInfo.table[index].communityLen )
             {
             // End of SNMP GET process
                 *ref = SNMP_END_OF_VAR;
                 return true;
             }
            if ( trapInfo.table[index].communityLen == 0u )
                *ref = SNMP_END_OF_VAR; // End of SNMP GET process
            else
            {
                // Start of SNMP GET process byte by byte
                val->byte = trapInfo.table[index].community[myRef];
                myRef++;
                *ref = myRef;
            }
            return true;
        }
        break;
// LED_D5 - generated from the Microchip style MIB script using mib2bib.jar is a scalar variable
    case LED_D5:
        val->byte = LED2_IO;
        return true;
}
```

Parameters

Parameters	Description
var	Variable id whose value is to be returned
index	For a scalar variable , Index value is zero. For sequence variable index value specifies which index value need to be set.
ref	Variable reference used to transfer multi-byte data. It is always SNMP_START_OF_VAR when very first byte is requested. Otherwise, use this as a reference to keep track of multi-byte transfers.

val	Pointer to up to 4 byte buffer: • If var data type is uint8_t, transfer data in val->byte • If var data type is uint16_t, transfer data in val->word • If var data type is uint32_t, transfer data in val->dword • If var data type is IP_ADDRESS, transfer data in val->v[] or val->dword • If var data type is COUNTER32, TIME_TICKS or GAUGE32, transfer data in val->dword • If var data type is ASCII_STRING or OCTET_STRING transfer data in val->byte using multi-byte transfer mechanism.

Function

bool TCPIP_SNMP_VarbindGet(SNMP_ID var, SNMP_INDEX index,uint8_t* ref, SNMP_VAL* val)

TCPIP_SNMP_VarbindSet Function

Sets the MIB variable with the requested value.

File

snmp.h

C

```
bool TCPIP_SNMP_VarbindSet(SNMP_ID var, SNMP_INDEX index, uint8_t ref, SNMP_VAL val);
```

Returns

- true - if it is okay to set more byte(s)
- false - if it is not okay to set more byte(s)

Description

This is a callback function called by module for SNMP SET request. This function content is modified by the application developer. This function contains the implementation of WRITE enable MIB OID macros.

mib2bib.jar Microchip MIB compiler utility is used to generate mib.h, which lists all the MIB OID ID value.

Remarks

This function may get called more than once depending on number of bytes in a specific set request for given variable. only dynamic read/-write variables needs to be handled.

Preconditions

TCPIP_SNMP_ProcessVariables() is called.

Example

```
switch(var)
{
    // LED_D5 - generated from the Microchip style MIB script using mib2bib.jar is a scalar variable
    case LED_D5:
        LED2_IO = val.byte;
        return true;

    // TRAP_COMMUNITY - generated from the Microchip style MIB script using mib2bib.jar,
    // is a Sequence variable.

    case TRAP_COMMUNITY:
        // Since this is a ASCII_STRING data type, SNMP will call with
        // SNMP_END_OF_VAR to indicate no more bytes or end of SNMP SET process
        // Use this information to determine if we just added new row
        // or updated an existing one.
        if ( ref ==  SNMP_END_OF_VAR )
        {
            // Index equal to table size means that we have new row.
            if ( index == trapInfo.Size )
                trapInfo.Size++;

            // Length of string is one more than index.
            trapInfo.table[index].communityLen++;

            return true;
```

```
        }

        // Make sure that index is within our range.
        if ( index < trapInfo.Size )
        {
            // Copy given value into local buffer.
            trapInfo.table[index].community[ref] = val.byte;
            // Keep track of length too.
            // This may not be NULL terminate string.
            trapInfo.table[index].communityLen = (uint8_t)ref;
            return true;
        }
        break;
}
```

Parameters

Parameters	Description
var	Write enable Variable id whose value is to be set
index	For a scalar variable , Index value is zero. For sequence variable index value specifies which index value need to be set.
ref	Variable reference used to transfer multi-byte data 0 if first byte is set otherwise non zero value to indicate corresponding byte being set. SNMP set will performed until ref is not equal to SNMP_END_OF_VAR and SNMP set starts with ref = SNMP_START_OF_VAR.
val	Up to 4 byte data value: • If var data type is uint8_t, variable value is in val->byte • If var data type is uint16_t, variable value is in val->word • If var data type is uint32_t, variable value is in val->dword. • If var data type is IP_ADDRESS, COUNTER32, or GAUGE32, value is in val->dword • If var data type is OCTET_STRING, ASCII_STRING value is in val->byte; multi-byte transfer will be performed to transfer remaining bytes of data.

Function

bool TCPIP_SNMP_VarbindSet(SNMP_ID var, SNMP_INDEX index,
uint8_t ref, SNMP_VAL val)

TCPIP_SNMP_WriteCommunityGet Function

Gets the writeCommunity String with SNMP index.

File

snmp.h

C

```
bool TCPIP_SNMP_WriteCommunityGet(int index, int len, uint8_t * dest);
```

Returns

- true - if the community string is collected
- false - if the community string is not collected

Description

This function is used to collect WRITE community string from the global community table with respect to the index value.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
index	One of the index of community table and it should be less than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT.

len	Length of the community string expected. It should not be more than TCPIP_SNMP_COMMUNITY_MAX_LEN.
dest	Copy the community string to this address and it should have a valid address.

Function

bool TCPIP_SNMP_WriteCommunityGet(int index,int len, uint8_t * dest)

TCPIP_SNMP_AuthTrapFlagGet Function

Gets the status of authentication trap flag.

File

snmp.h

C

```
void TCPIP_SNMP_AuthTrapFlagGet(bool * authTrap);
```

Returns

None.

Description

This function is used to Get the authentication trap send flag and this is used only when the user is trying to send authentication failure trap. For example, sending a trap if community does not match the global community table.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
sendTrap	Trap flag value.

Function

void TCPIP_SNMP_AuthTrapFlagGet(bool *sendTrap)

TCPIP_SNMP_AuthTrapFlagSet Function

Sets the status of authentication trap flag.

File

snmp.h

C

```
void TCPIP_SNMP_AuthTrapFlagSet(bool sendTrap);
```

Returns

None.

Description

This function is used to set the authentication trap send flag and this is used only when user is trying to send authentication failure trap. Ex-sending a trap if community do not match to the global community table.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Function

void TCPIP_SNMP_AuthTrapFlagSet(bool sendTrap)

TCPIP_SNMP_IsTrapEnabled Function

Gets the SNMP Trap status.

File

snmp.h

C

```
bool TCPIP_SNMP_IsTrapEnabled();
```

Returns

- true - SNMP module has trap enabled
- false - SNMP module has trap disabled

Description

This function returns true if SNMP module is enabled to send the trap to to the SNMP manager device.

Remarks

This function is used by the customer function.

Preconditions

TCPIP_SNMP_Initialize is already called.

Function

bool TCPIP_SNMP_IsTrapEnabled(void)

TCPIP_SNMP_TRAPTypeGet Function

Get SNMP Trap type for version v1 and v2c.

File

snmp.h

C

```
bool TCPIP_SNMP_TRAPTypeGet();
```

Returns

- true - trap version is v2
- false - trap version is v1

Description

This function returns true if the trap tye is v2 and the TRAP pdu packet will be a TRAP v2 format.The return value is also used is validated when SNMP module is trying to send a trap for SNMP version v3.

Remarks

This function is used by the customer function.

Preconditions

TCPIP_SNMP_Initialize() is already called.

Function

bool TCPIP_SNMP_TRAPTypeGet(void)

TCPIP_SNMP_TRAPv1Notify Function

Creates and Sends TRAPv1 pdu.

File

snmp.h

C

```
bool TCPIP_SNMP_TRAPv1Notify(SNMP_ID var, SNMP_VAL val, SNMP_INDEX index, SNMP_TRAP_IP_ADDRESS_TYPE
eTrapMultiAddressType);
```

Description

This function creates SNMP trap PDU and sends it to previously specified remoteHost. snmpv1 trap pdu: | PDU-type | enterprise | agent-addr | generic-trap | specific-trap | | time-stamp | varbind-list |

The v1 enterprise is mapped directly to SNMPv2TrapOID.0

Remarks

This would fail if there were not UDP socket to open.

Preconditions

TCPIP_SNMP_NotifyIsReady() is already called and returned true.

Parameters

Parameters	Description
var	SNMP var ID that is to be used in notification
val	Value of var. Only value of uint8_t, uint16_t or uint32_t can be sent.
index	Index of var. If this var is a single,index would be 0, or else if this var Is a sequence, index could be any value from 0 to 127
eTrapMultiAddressType	Trap address type

Function

bool TCPIP_SNMP_TRAPv1Notify(SNMP_ID var, SNMP_VAL val,

SNMP_INDEX index,SNMP_TRAP_IP_ADDRESS_TYPE eTrapMultiAddressType)

TCPIP_SNMP_TRAPv2Notify Function

Creates and sends TRAP PDU.

File

snmp.h

C

```
bool TCPIP_SNMP_TRAPv2Notify(SNMP_ID var, SNMP_VAL val, SNMP_INDEX index, SNMP_TRAP_IP_ADDRESS_TYPE
eTrapMultiAddressType);
```

Description

This function creates SNMP V2 Trap PDU and sends it to previously specified remoteHost.

SNMP V1 trap PDU: | PDU type | enterprise | agent addr | generic trap | specific trap | | time stamp | varbind list |

The v1 enterprise is mapped directly to SNMPv2TrapOID.0

SNMP V2 trap PDU: version (0 or 1) | community | SNMP PDU |PDU type | request id | error status |err index |varbinds

The first two variables (in varbind list) of snmpv2 are: sysUpTime.0 and SNMPv2TrapOID.0

Generic Trap OID is used as the varbind for authentication failure.

Remarks

This would fail if there were not UDP socket to open.

Preconditions

TCPIP_SNMP_NotifyIsReady is already called and returned true.

Parameters

Parameters	Description
var	SNMP var ID that is to be used in notification
val	Value of var. Only value of uint8_t, uint16_t or uint32_t can be sent.
index	Index of var. If this var is a single,index would be 0, or else if this var Is a sequence, index could be any value from 0 to 127
eTrapMultiAddressType	Trap address type

Function

bool TCPIP_SNMP_TRAPv2Notify(SNMP_ID var, SNMP_VAL val, SNMP_INDEX index,
SNMP_TRAP_IP_ADDRESS_TYPE eTrapMultiAddressType)

TCPIP_SNMPV3_TrapTypeGet Function

Gets SNMP Trap type for version v3.

File

snmp.h

C

```
bool TCPIP_SNMPV3_TrapTypeGet();
```

Returns

- true - SNMP3 will use trap version v2
- false - SNMP3 will use trap version v1

Description

This function returns true if SNMP module is trying to send trap version v2 with SNMP version v3.

Remarks

This function is used by the customer function.

Preconditions

TCPIP_SNMP_Initialize() is already called.

Function

bool TCPIP_SNMPV3_TrapTypeGet(void)

TCPIP_SNMP_ValidateTrapIntf Function

Gets the status of trap interface.

File

snmp.h

C

```
bool TCPIP_SNMP_ValidateTrapIntf(TCPIP_NET_HANDLE pIf);
```

Returns

- true - SNMP trap interface is valid
- false - SNMP trap interface is invalid

Description

This function returns true if SNMP module trap interface is a valid interface.

Remarks

This function is used by the customer function.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
pif	network interface

Function

bool TCPIP_SNMP_ValidateTrapIntf(TCPIP_NET_HANDLE pIf)

TCPIP_SNMP_ReadCommunitySet Function

Sets the readCommunity String with SNMP index.

File

snmp.h

C

```
bool TCPIP_SNMP_ReadCommunitySet(int index, int len, uint8_t * src);
```

Returns

- true - if the community string is collected
- false - if the community string is not collected

Description

This function is used to configure READ community string from the user and configure the SNMP community table.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
index	One of the index of community table and it should be less than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT.
len	Length of the community string expected. It should not be more than TCPIP_SNMP_COMMUNITY_MAX_LEN.
src	Copy this community string to snmp community table.

Function

bool TCPIP_SNMP_ReadCommunitySet(int index,int len, uint8_t * src)

TCPIP_SNMP_WriteCommunitySet Function

Sets the writeCommunity String with SNMP index.

File

snmp.h

C

```
bool TCPIP_SNMP_WriteCommunitySet(int index, int len, uint8_t * src);
```

Returns

- true - if the community string is collected
- false - if the community string is not collected

Description

This function is used to collect WRITE community string from user and set the community table with respect to the index value.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
index	One of the index of community table and it should be less than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT.

len	Length of the community string expected. It should not be more than TCPIP_SNMP_COMMUNITY_MAX_LEN.
src	Copy this community string to SNMP community table.

Function

bool TCPIP_SNMP_WriteCommunitySet(int index,int len, uint8_t * src)

TCPIP_SNMP_SocketIDGet Function

Gets the Socket ID for SNMP Server socket.

File

snmp.h

C

```
void TCPIP_SNMP_SocketIDGet(UDP_SOCKET * socket);
```

Returns

None.

Description

This function is used to get trap client socket ID for both IPv4 and IPv6 receiver Address.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
socket	Trap client socket ID

Function

void TCPIP_SNMP_SocketIDGet(UDP_SOCKET *socket)

TCPIP_SNMP_SocketIDSet Function

Sets the Socket ID for SNMP Server socket.

File

snmp.h

C

```
void TCPIP_SNMP_SocketIDSet(UDP_SOCKET socket);
```

Returns

None.

Description

This function is used to update socket value of SNMP trap global structure.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
socket	Trap client socket ID

Function

void TCPIP_SNMP_SocketIDSet(UDP_SOCKET socket)

TCPIP_SNMP_Task Function

Standard TCP/IP stack module task function.

File

snmp.h

C

```
void TCPIP_SNMP_Task();
```

Returns

None.

Description

This function performs SNMP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The SNMP module should have been initialized

Function

void TCPIP_SNMP_Task(void)

b) SNMPv3 Module Functions

TCPIP_SNMPV3_EngineUserDataBaseGet Function

Get SNMPv3 engine data base details.

File

snmpv3.h

C

```
bool TCPIP_SNMPV3_EngineUserDataBaseGet(TCPIP_SNMPV3_USERDATABASECONFIG_TYPE userDataBaseType, uint8_t len,
uint8_t userIndex, void * val);
```

Returns

None.

Description

This function is used to get SNMPv3 Engine data base details using TCPIP_SNMPV3_USERDATABASECONFIG_TYPE enumeration.

Remarks

None

Preconditions

TCPIP_SNMP_Initialize is already called.

Example

```
bool TCPIP_SNMP_VarbindGet(SNMP_ID var, SNMP_INDEX index, uint8_t* ref, SNMP_VAL* val)
{
    switch(var)
    {
        case USER_SECURITY_NAME:
            if(index < TCPIP_SNMPV3_USM_MAX_USER)
            {
```

```
                    if(TCPIP_SNMPV3_EngineUserDataBaseGet(SNMPV3_USERNAME_CONFIG_TYPE,myRef,index,&val->byte)
== false)
                    {
                        *ref = SNMP_END_OF_VAR;
                    }
                    else
                    {
                        myRef++;
                        *ref = myRef;
                    }
                    return true;
                }
                break;
            }
        }
```

Parameters

Parameters	Description
userDataBaseType	SNMPv3 data base configuration type
len	Number of bytes need to be read from data base
userIndex	SNMPv3 user index
val	void pointer to a any data type. Length parameter value is changed according to data type.

Function

void TCPIP_SNMPV3_EngineUserDataBaseGet(TCPIP_SNMPV3_USERDATABASECONFIG_TYPE userDataBaseType,
uint8_t len,uint8_t userIndex,void *val);

TCPIP_SNMPV3_EngineUserDataBaseSet Function

Set SNMPv3 engine data base details.

File

snmpv3.h

C

```
bool TCPIP_SNMPV3_EngineUserDataBaseSet(TCPIP_SNMPV3_USERDATABASECONFIG_TYPE userDataBaseType, uint8_t len,
uint8_t userIndex, void * val);
```

Returns

None.

Description

This function is used to set SNMPv3 Engine data base details using TCPIP_SNMPV3_USERDATABASECONFIG_TYPE enumeration.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Example

```
bool TCPIP_SNMP_VarbindSet(SNMP_ID var, SNMP_INDEX index, uint8_t ref, SNMP_VAL val)
{
  switch(var)
  {
      case USER_SECURITY_NAME:
          if ( ref ==  SNMP_END_OF_VAR )
          {
              if(TCPIP_SNMPV3_EngineUserDataBaseSet(SNMPV3_USERNAME_CONFIG_TYPE,strlen((char*)gSnmpv3UserSec
urityName),
                                                    index,gSnmpv3UserSecurityName) != true)
                  return false;
          }
```

```
            // Make sure that index is within our range.
            if ( index < TCPIP_SNMPV3_USM_MAX_USER )
            {
                // Copy given value into local buffer.
                gSnmpv3UserSecurityName[ref]=val.byte;
                return true;
            }
            break;
    }
}
```

Parameters

Parameters	Description
userDataBaseType	SNMPv3 data base configuration type
len	Number of bytes need to be read from data base
userIndex	SNMPv3 user index
val	void pointer to a any data type. Length parameter value is changed according to data type.

Function

void TCPIP_SNMPV3_EngineUserDataBaseSet(TCPIP_SNMPV3_USERDATABASECONFIG_TYPE userDataBaseType,
uint8_t len,uint8_t userIndex,void *val);

TCPIP_SNMPv3_TrapConfigDataGet Function

Gets the SNMPv3 Trap configuration details using the user name index.

File

snmpv3.h

C

```
void TCPIP_SNMPv3_TrapConfigDataGet(uint8_t userIndex, uint8_t * msgModelType, uint8_t * securityModelType);
```

Returns

None.

Description

This function is used to get SNMPv3 message model type and security model type using user index.

Remarks

None.

Preconditions

TCPIP_SNMP_Initialize is already called.

Parameters

Parameters	Description
userIndex	user name index
msgModelType	message processing type
securityModelType	security model type

Function

void TCPIP_SNMPv3_TrapConfigDataGet(uint8_t userIndex,uint8_t *msgModelType,
uint8_t *securityModelType)

TCPIP_SNMPv3_Notify Function

Creates and Sends SNMPv3 TRAP PDU.

File

snmpv3.h

C

```
bool TCPIP_SNMPv3_Notify(SNMP_ID var, SNMP_VAL val, SNMP_INDEX index, uint8_t targetIndex,
SNMP_TRAP_IP_ADDRESS_TYPE eTrapMultiAddressType);
```

Description

This function creates SNMPv3 trap PDU and sends it to previously specified remoteHost.

Remarks

None.

Preconditions

TRAP event is triggered.

Parameters

Parameters	Description
var	SNMP var ID that is to be used in notification
val	Value of var. Only value of uint8_t, uint16_t or uint32_t can be sent.
index	Index of var. If this var is a single,index would be 0, or else if this var Is a sequence, index could be any value from 0 to 127 targetIndex -index of the 'Snmpv3TrapConfigData' table's security user name for which the TRAP PDU message header to constructed.

Function

bool TCPIP_SNMPv3_Notify(SNMP_ID var, SNMP_VAL val, SNMP_INDEX index,

uint8_t targetIndex)

c) SNMP Data Types and Constants

SNMP_END_OF_VAR Macro

File

snmp.h

C

```
#define SNMP_END_OF_VAR (0xff)
```

Description

This Macro is used for both SNMP SET and GET Variable processing to indicate the end of SNMP variable processing. For multi byte data request, the end byte will be always SNMP_END_OF_VAR.

SNMP_INDEX_INVALID Macro

File

snmp.h

C

```
#define SNMP_INDEX_INVALID (0xff)
```

Description

This Macro is used for both SNMP SET and GET Sequence Variable processing. SNMP starts processing the start of sequence variable with Invalid index. TCPIP_SNMP_ExactIndexGet and TCPIP_SNMP_NextIndexGet returns a valid index as per SNMP_INDEX_INVALID.

SNMP_START_OF_VAR Macro

File

snmp.h

C

```
#define SNMP_START_OF_VAR (0)
```

Description

This Macro is used for both SNMP SET and GET Variable processing to indicate the start of SNMP variable processing. For multi byte data request, the first byte will be always SNMP_START_OF_VAR.

SNMP_V1 Macro

File

snmp.h

C

```
#define SNMP_V1 (0)
```

Description

This macro is used for SNMP version 1

SNMP_V2C Macro

File

snmp.h

C

```
#define SNMP_V2C (1)
```

Description

This macro is used for SNMP version 2 with community

SNMP_V3 Macro

File

snmp.h

C

```
#define SNMP_V3 (3)
```

Description

This macro is used for SNMP version 3 with authentication and privacy

SNMP_COMMUNITY_TYPE Enumeration

Definition to represent different type of SNMP communities.

File

snmp.h

C

```
typedef enum {
  READ_COMMUNITY = 1,
  WRITE_COMMUNITY = 2,
  INVALID_COMMUNITY = 3
} SNMP_COMMUNITY_TYPE;
```

Members

Members	Description
READ_COMMUNITY = 1	Read only community
WRITE_COMMUNITY = 2	Read write community
INVALID_COMMUNITY = 3	Community invalid

Description

Enumeration: SNMP_COMMUNITY_TYPE

List of different SNMP community types.

Remarks

SNMP agent use these community types for both TRAP and SNMP agent request.

SNMP_ID Type

SNMP dynamic variable ID.

File

snmp.h

C

```
typedef uint32_t SNMP_ID;
```

Description

Type: SNMP_ID

Only dynamic and AgentID variables can contain a dyanmic ID. MIB2BIB utility enforces this rule when BIB was generated. All the dynamic IDs are are listed in mib.h. The maximum value of a dynamic ID is 1023.

Remarks

mib2bib.jar utility generates mib.h and snmp.bib from Microchip MIB script. DynamicVar - This command declares defined OID variable as dynamic. Syntax - $DynamicVar(oidName, id).

SNMP_INDEX Type

SNMP sequence variable index.

File

snmp.h

C

```
typedef uint8_t SNMP_INDEX;
```

Description

Type: SNMP_INDEX

The current version limits the size of the index to 7 bits wide, meaning that such arrays can contain up to 127 entries.

Remarks

SequenceVar - This command is part of Microchip MIB script declares a previously defined OID variable as a sequence variable and assigns an index to it. A sequence variable can contain an array of values and any instance of its values can be referenced by an index. More than one sequence variable may share a single index, creating multi dimensional arrays.

More than one index variable is not supported by mib2bib.jar compiler.

SNMP_NON_MIB_RECD_INFO Structure

Restrict access for specific OIDs.

File

snmp.h

C

```
typedef struct {
  uint8_t oidstr[16];
  uint8_t version;
} SNMP_NON_MIB_RECD_INFO;
```

Description

Structure: SNMP_NON_MIB_RECD_INFO

This structure is used to restrict access to static variables of SNMPv3 OIDs from SNMPv2c and SNMPv1 version.OID string length is restricted to 16.

Remarks

None.

SNMP_TRAP_IP_ADDRESS_TYPE Enumeration

Definition of the supported address types for SNMP trap.

File

snmp.h

C

```
typedef enum {
  IPV4_SNMP_TRAP = 1,
  IPV6_SNMP_TRAP
} SNMP_TRAP_IP_ADDRESS_TYPE;
```

Description

Enumeration: SNMP_TRAP_IP_ADDRESS_TYPE

SNMP agent supports both IPv4 and IPv6 trap address type and is able to transmit traps to both IPv4 and IPv6 Host receiver address.

Remarks

None.

SNMP_VAL Union

Definition to represent SNMP OID object values.

File

snmp.h

C

```
typedef union {
  uint32_t dword;
  uint16_t word;
  uint8_t byte;
  uint8_t v[sizeof(uint32_t)];
} SNMP_VAL;
```

Members

Members	Description
uint32_t dword;	double word value
uint16_t word;	word value
uint8_t byte;	byte value
uint8_t v[sizeof(uint32_t)];	byte array

Description

Union: SNMP_VAL

SNMP agent processes different variable types.

Remarks

None.

SNMP_GENERIC_TRAP_NOTIFICATION_TYPE Enumeration

Definition to represent different SNMP generic trap types.

File

snmp.h

C

```
typedef enum {
```

```
  COLD_START = 0x0,
  WARM_START = 0x1,
  LINK_DOWN = 0x2,
  LINK_UP = 0x3,
  AUTH_FAILURE = 0x4,
  EGP_NEIGHBOR_LOSS = 0x5,
  ENTERPRISE_SPECIFIC = 0x6
} SNMP_GENERIC_TRAP_NOTIFICATION_TYPE;
```

Members

Members	Description
COLD_START = 0x0	Controls the sending of SNMP coldstart notifications. A coldStart(0) trap signifies that the sending protocol entity is reinitializing itself such that the configuration of the agent or the protocol entity implementation might be altered.
WARM_START = 0x1	Controls the sending of SNMP warmstart notifications. A warmStart(1) trap signifies that the sending protocol entity is reinitializing itself so that neither the agent configuration nor the protocol entity implementation can be altered.
LINK_DOWN = 0x2	Controls the how SNMP linkdown notifications are sent. A linkDown(2) trap signifies that the sending protocol entity recognizes a failure in one of the communication links represented in the configuration of the agent.
LINK_UP = 0x3	Controls the sending of SNMP linkup notifications. A linkUp(3) trap signifies that the sending protocol entity recognizes that one of the communication links represented in the configuration of the agent has come up.
AUTH_FAILURE = 0x4	Controls the distribution of SNMP authentication failure notifications. An authenticationFailure(4) trap signifies that the sending protocol entity is the addressee of a protocol message that is not properly authenticated. Like Community Name authentication failure
EGP_NEIGHBOR_LOSS = 0x5	Controls the distribution of SNMP egpNeighborLoss notifications.An egpNeighborLoss(5) trap signifies that an EGP neighbor for whom the sending protocol entity was an EGP peer has been marked down and the peer relationship no longer exists.
ENTERPRISE_SPECIFIC = 0x6	Controls the distribution of SNMP enterprise/-specific notifications.An enterpriseSpecific(6) trap signifies that the sending protocol entity recognizes that some enterprise/-specific event has occurred. The specific-trap field identifies the particular trap that occurred.

Description

Enumeration: SNMP_GENERIC_TRAP_NOTIFICATION_TYPE

List of different SNMP specific Notification types.

Remarks

ENTERPRISE_SPECIFIC and AUTH_FAILURE are used while sending specific trap.

SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE Enumeration

Definition to represent different SNMP vendor trap types.

File

snmp.h

C

```
typedef enum {
  VENDOR_TRAP_DEFAULT = 0x0,
  BUTTON_PUSH_EVENT = 0x1,
  POT_READING_MORE_512 = 0x2
} SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE;
```

Members

Members	Description
VENDOR_TRAP_DEFAULT = 0x0	Default trap . Agent send use this trap when there is authentication failure.
BUTTON_PUSH_EVENT = 0x1	PUSH button event trap notification
POT_READING_MORE_512 = 0x2	Analog POT meter event trap notification

Description

Enumeration: SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE

List of different SNMP Vendor Notification types.

Remarks

None.

d) SNMPv3 Data Types and Constants

SNMPV3_HMAC_HASH_TYPE Enumeration

Different type of authentication for SNMPv3.

File

snmpv3.h

C

```
typedef enum {
  SNMPV3_HMAC_MD5 = 0u,
  SNMPV3_HMAC_SHA1,
  SNMPV3_NO_HMAC_AUTH
} SNMPV3_HMAC_HASH_TYPE;
```

Members

Members	Description
SNMPV3_HMAC_MD5 = 0u	MD5 is being calculated HMAC SHA1 authentication protocol
SNMPV3_HMAC_SHA1	SHA-1 is being calculated No authentication is supported

Description

Enumeration: SNMPV3_HMAC_HASH_TYPE

The following authentication types are supported by the SNMPv3 USM model for data confidentiality. SNMPv3 agent supports both MD5 and SHA1 protocol for authentication.

Remarks

None.

SNMPV3_PRIV_PROT_TYPE Enumeration

Different type of encryption and decryption for SNMPv3.

File

snmpv3.h

C

```
typedef enum {
  SNMPV3_DES_PRIV = 0x0,
  SNMPV3_AES_PRIV,
  SNMPV3_NO_PRIV
} SNMPV3_PRIV_PROT_TYPE;
```

Members

Members	Description
SNMPV3_DES_PRIV = 0x0	Data Encryption Standard (DES-CBC) encryption and decryption protocol
SNMPV3_AES_PRIV	Advanced Encryption Standard (AES-CFB) encryption and decryption protocol
SNMPV3_NO_PRIV	No encryption or decryption protocol is supported

Description

Enumeration: SNMPV3_PRIV_PROT_TYPE

These below privacy types are supported by the SNMPv3 USM model for data confidentiality. SNMPv3 agent supports both AES-CFB and DES-CBC encryption and decryption algorithm. For DES-CBC privacy protocol, SNMPv3 agent will use Harmony Crypto Library. For AES-CFB privacy protocol, SNMPv3 agent will use Legacy TCP/IP Crypto Library (For AES, include -aes_pic32mx.a to the project.)

Remarks

128-bit, 192-bit and 256-bit AES are supported.

STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL Enumeration

Different SNMP Message processing model

File

snmpv3.h

C

```
typedef enum {
  SNMPV1_MSG_PROCESSING_MODEL = 0X00,
  SNMPV2C_MSG_PROCESSING_MODEL = 0X01,
  SNMPV2U_V2_MSG_PROCESSING_MODEL = 0X02,
  SNMPV3_MSG_PROCESSING_MODEL = 0X03
} STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL;
```

Members

Members	Description
SNMPV1_MSG_PROCESSING_MODEL = 0X00	SNMP version 1 Message processing model
SNMPV2C_MSG_PROCESSING_MODEL = 0X01	SNMP version 2 Message processing model with community as security feature
SNMPV2U_V2_MSG_PROCESSING_MODEL = 0X02	SNMP version 2 Message processing model
SNMPV3_MSG_PROCESSING_MODEL = 0X03	SNMP version 3 Message processing model with authentication and encryption and decryption

Description

Enumeration: STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL

SNMP Message processing model is responsible for processing an SNMP version specific message and for coordinating the interaction with security sub system. SNMP message processing subsystem is part of an SNMP engine which interacts with the Dispatcher to handle the version specific SNMP messages.

A Message Processing Model describes the version-specific procedures for extracting data from messages, generating messages, calling upon a securityModel to apply its security services to messages.

Remarks

None.

STD_BASED_SNMP_SECURITY_MODEL Enumeration

Different Security services for SNMPv3 messages.

File

snmpv3.h

C

```
typedef enum {
  ANY_SECUTIRY_MODEL = 0x00,
  SNMPV1_SECURITY_MODEL = 0X01,
  SNMPV2C_SECURITY_MODEL = 0X02,
  SNMPV3_USM_SECURITY_MODEL = 0X03
} STD_BASED_SNMP_SECURITY_MODEL;
```

Members

Members	Description
ANY_SECUTIRY_MODEL = 0x00	Security Model Reserved for ANY
SNMPV1_SECURITY_MODEL = 0X01	Security Model reserved fro SNMP version 1
SNMPV2C_SECURITY_MODEL = 0X02	Community Security Model reserved for SNMP version 2
SNMPV3_USM_SECURITY_MODEL = 0X03	User based security model reserved for SNMP version 3

Description

Enumeration: STD_BASED_SNMP_SECURITY_MODEL

SNMP Security subsystem is applied to the transmission and reception of messages and to the processing of the contents of messages.

- The zero value does not identify any particular security model.
- Values between 1 and 255, inclusive, are reserved for standards-track Security Models and are

managed by the Internet Assigned Numbers Authority (IANA).

- Values greater than 255 are allocated to enterprise-specific Security Models. An

enterprise specific securityModel value is defined to be: enterpriseID * 256 + security model within enterprise

Remarks

None.

STD_BASED_SNMPV3_SECURITY_LEVEL Enumeration

Different Security Level for SNMPv3 messages.

File

snmpv3.h

C

```
typedef enum {
  NO_AUTH_NO_PRIV = 1,
  AUTH_NO_PRIV,
  AUTH_PRIV
} STD_BASED_SNMPV3_SECURITY_LEVEL;
```

Members

Members	Description
NO_AUTH_NO_PRIV = 1	without authentication and without privacy
AUTH_NO_PRIV	with authentication but without privacy
AUTH_PRIV	with authentication but with privacy

Description

Enumeration: STD_BASED_SNMPV3_SECURITY_LEVEL

A Level of Security at which SNMPv3 messages can be sent or with which operations are being processed.

Remarks

None.

TCPIP_SNMPV3_USERDATABASECONFIG_TYPE Enumeration

Different Configuration parameters of SNMPv3 operation

File

snmpv3.h

C

```
typedef enum {
  SNMPV3_USERNAME_CONFIG_TYPE = 0,
  SNMPV3_AUTHPASSWD_CONFIG_TYPE,
  SNMPV3_PRIVPASSWD_CONFIG_TYPE,
  SNMPV3_AUTHPASSWDLOCALIZEDKEY_CONFIG_TYPE,
  SNMPV3_PRIVPASSWWDLOCALIZEDKEY_CONFIG_TYPE,
  SNMPV3_HASHTYPE_CONFIG_TYPE,
  SNMPV3_PRIVTYPE_CONFIG_TYPE,
  SNMPV3_TARGET_SECURITY_NAME_TYPE,
  SNMPV3_TARGET_SECURITY_LEVEL_TYPE,
  SNMPV3_TARGET_SECURITY_MODEL_TYPE,
  SNMPV3_TARGET_MP_MODEL_TYPE,
  SNMPV3_ENGINE_ID_TYPE,
  SNMPV3_ENGINE_BOOT_TYPE,
  SNMPV3_ENGINE_TIME_TYPE,
  SNMPV3_ENGINE_MAX_MSG_TYPE
} TCPIP_SNMPV3_USERDATABASECONFIG_TYPE;
```

Members

Members	Description
SNMPV3_USERNAME_CONFIG_TYPE = 0	Snmpv3 user name configuration
SNMPV3_AUTHPASSWD_CONFIG_TYPE	Authentication configuration type
SNMPV3_PRIVPASSWD_CONFIG_TYPE	Encryption and Decryption password configuration
SNMPV3_AUTHPASSWDLOCALIZEDKEY_CONFIG_TYPE	Authorization localized key configuration type
SNMPV3_PRIVPASSWWDLOCALIZEDKEY_CONFIG_TYPE	Encryption and Decryption password localized key configuration type
SNMPV3_HASHTYPE_CONFIG_TYPE	SNMPv3 hash algorithm type
SNMPV3_PRIVTYPE_CONFIG_TYPE	SNMPv3 privacy configuration type
SNMPV3_TARGET_SECURITY_NAME_TYPE	SNMPv3 target address user name . This is for Trap communication
SNMPV3_TARGET_SECURITY_LEVEL_TYPE	SNMPv3 target security type
SNMPV3_TARGET_SECURITY_MODEL_TYPE	SNMPv3 target security model type
SNMPV3_TARGET_MP_MODEL_TYPE	SNMPv3 target security message processing model type
SNMPV3_ENGINE_ID_TYPE	Identifier that uniquely and unambiguously identifies the local SNMPv3 engine
SNMPV3_ENGINE_BOOT_TYPE	Number of times the local SNMPv3 engine has rebooted or reinitialized since the engine ID was last changed
SNMPV3_ENGINE_TIME_TYPE	Number of seconds since the local SNMPv3 engine was last rebooted or reinitialized
SNMPV3_ENGINE_MAX_MSG_TYPE	SNMPv3 Engine Maximum message size the sender can accommodate

Description

Enumeration: TCPIP_SNMPV3_USERDATABASECONFIG_TYPE

These configuration types are used by the SNMP user while doing configuration SNMPv3 parameters. It includes SNMpv3 user name , authentication and encryption configuration parameters.

Remarks

None.

TCPIP_SNMP_COMMUNITY_CONFIG Structure

SNMP community configuration.

File

snmpv3.h

C

```
typedef struct {
  char * communityName;
} TCPIP_SNMP_COMMUNITY_CONFIG;
```

Description

Structure: TCPIP_SNMP_COMMUNITY_CONFIG

This structure is used to configure community details during run-time.

Remarks

None.

TCPIP_SNMP_MODULE_CONFIG Structure

SNMP module configuration.

File

snmpv3.h

C

```
typedef struct {
  bool trapEnable;
  bool snmp_trapv2_use;
  bool snmpv3_trapv1v2_use;
  char* snmp_bib_file;
```

```
  TCPIP_SNMP_COMMUNITY_CONFIG * read_community_config;
  TCPIP_SNMP_COMMUNITY_CONFIG * write_community_config;
  TCPIP_SNMPV3_USM_USER_CONFIG * usm_config;
  TCPIP_SNMPV3_TARGET_ENTRY_CONFIG * trap_target_config;
} TCPIP_SNMP_MODULE_CONFIG;
```

Members

Members	Description
bool trapEnable;	true = agent can send the trap, false = agent shouldn't send the trap
bool snmp_trapv2_use;	true = agent uses Trap version v2 and false = uses Tarp version 1
bool snmpv3_trapv1v2_use;	SNMPv3 trap should be true , only if SNMP version is 3
TCPIP_SNMP_COMMUNITY_CONFIG * read_community_config;	read-only Community configuration
TCPIP_SNMP_COMMUNITY_CONFIG * write_community_config;	write-only Community configuration
TCPIP_SNMPV3_USM_USER_CONFIG * usm_config;	SNMPv3 USM configuration
TCPIP_SNMPV3_TARGET_ENTRY_CONFIG * trap_target_config;	SNMPv3 trap configuration

Description

Structure: TCPIP_SNMP_MODULE_CONFIG

This structure is used to configure SNMP details for runtime configuration.

Remarks

None

TCPIP_SNMPV3_TARGET_ENTRY_CONFIG Structure

SNMP module trap target address configuration.

File

snmpv3.h

C

```
typedef struct {
  char * secname;
  STD_BASED_SNMP_SECURITY_MODEL mp_model;
  STD_BASED_SNMP_SECURITY_MODEL sec_model;
  STD_BASED_SNMPV3_SECURITY_LEVEL sec_level;
} TCPIP_SNMPV3_TARGET_ENTRY_CONFIG;
```

Description

Structure: SNMPV3_TARGET_ENTRY_CONFIG

This structure is used to configure SNMP target details during runtime .

Remarks

None.

TCPIP_SNMPV3_USM_USER_CONFIG Structure

SNMPv3 USM configuration.

File

snmpv3.h

C

```
typedef struct {
  char * username;
  STD_BASED_SNMPV3_SECURITY_LEVEL security_level;
  SNMPV3_HMAC_HASH_TYPE usm_auth_proto;
  char * usm_auth_password;
  SNMPV3_PRIV_PROT_TYPE usm_priv_proto;
  char * usm_priv_password;
```

```
} TCPIP_SNMPV3_USM_USER_CONFIG;
```

Members

Members	Description
char * username;	< user name string
STD_BASED_SNMPV3_SECURITY_LEVEL security_level;	< security level: auth, priv combination
SNMPV3_HMAC_HASH_TYPE usm_auth_proto;	< auth type: md5, sha1
char * usm_auth_password;	< passphrase string for authentication
SNMPV3_PRIV_PROT_TYPE usm_priv_proto;	< priv type: DES
char * usm_priv_password;	< passphrase string for privacy

Description

Structure: SNMPV3_USM_USER_CONFIG

This structure is used to configure predefined SNMPv3 USM details for run-time configuration.

Remarks

None.

Files

Files

Name	Description
snmp.h	Simple Network Management Protocol(SNMP) v1/v2c API header file.
snmpv3.h	Simple Network Management Protocol Version3(SNMPv3) API header file.
snmp_config.h	SNMP configuration file
snmpv3_config.h	SNMPv3 configuration file

Description

This section lists the source and header files used by the library.

snmp.h

Simple Network Management Protocol(SNMP) v1/v2c API header file.

Enumerations

	Name	Description
	SNMP_COMMUNITY_TYPE	Definition to represent different type of SNMP communities.
	SNMP_GENERIC_TRAP_NOTIFICATION_TYPE	Definition to represent different SNMP generic trap types.
	SNMP_TRAP_IP_ADDRESS_TYPE	Definition of the supported address types for SNMP trap.
	SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE	Definition to represent different SNMP vendor trap types.

Functions

	Name	Description
	TCPIP_SNMP_AuthTrapFlagGet	Gets the status of authentication trap flag.
	TCPIP_SNMP_AuthTrapFlagSet	Sets the status of authentication trap flag.
	TCPIP_SNMP_ClientGetNet	Get a network interface for SNMP TRAP.
	TCPIP_SNMP_ExactIndexGet	To search for exact index node in case of a Sequence variable.
	TCPIP_SNMP_IsTrapEnabled	Gets the SNMP Trap status.
	TCPIP_SNMP_IsValidCommunity	Validates community name for access control.
	TCPIP_SNMP_IsValidLength	Validates the set variable data length to data type.
	TCPIP_SNMP_MibIDSet	Sets the agent MIB ID for SNP notification.
	TCPIP_SNMP_NextIndexGet	To search for next index node in case of a Sequence variable.
	TCPIP_SNMP_NotifyIsReady	Resolves given remoteHost IP address into MAC address.
	TCPIP_SNMP_NotifyPrepare	Collects trap notification info and send ARP to remote host.
	TCPIP_SNMP_ReadCommunityGet	Gets the readCommunity String with SNMP index.
	TCPIP_SNMP_ReadCommunitySet	Sets the readCommunity String with SNMP index.

	TCPIP_SNMP_RecordIDValidation	Used to restrict the access dynamic and non dynamic OID string for a particular SNMP Version.
	TCPIP_SNMP_SendFailureTrap	Prepares and validates the remote node that will receive a trap and send the trap PDU.
	TCPIP_SNMP_SocketIDGet	Gets the Socket ID for SNMP Server socket.
	TCPIP_SNMP_SocketIDSet	Sets the Socket ID for SNMP Server socket.
	TCPIP_SNMP_Task	Standard TCP/IP stack module task function.
	TCPIP_SNMP_TrapInterFaceSet	Sets the TRAP interface for SNMP notification.
	TCPIP_SNMP_TRAPMibIDGet	Gets the agent MIB ID for SNP notification.
	TCPIP_SNMP_TrapSendFlagGet	Gets the status of trap send flag.
	TCPIP_SNMP_TrapSendFlagSet	Sets the status of trap send flag.
	TCPIP_SNMP_TrapSpecificNotificationGet	Gets the specific trap.
	TCPIP_SNMP_TrapSpecificNotificationSet	Sets the specific trap, generic trap, and trap ID.
	TCPIP_SNMP_TrapTimeGet	Gets SNMP Trap UDP client open socket time-out.
	TCPIP_SNMP_TRAPTypeGet	Get SNMP Trap type for version v1 and v2c.
	TCPIP_SNMP_TRAPv1Notify	Creates and Sends TRAPv1 pdu.
	TCPIP_SNMP_TRAPv2Notify	Creates and sends TRAP PDU.
	TCPIP_SNMP_ValidateTrapIntf	Gets the status of trap interface.
	TCPIP_SNMP_VarbindGet	Used to get/collect OID variable information.
	TCPIP_SNMP_VarbindSet	Sets the MIB variable with the requested value.
	TCPIP_SNMP_WriteCommunityGet	Gets the writeCommunity String with SNMP index.
	TCPIP_SNMP_WriteCommunitySet	Sets the writeCommunity String with SNMP index.
	TCPIP_SNMPV3_TrapTypeGet	Gets SNMP Trap type for version v3.

Macros

	Name	Description
	SNMP_END_OF_VAR	This Macro is used for both SNMP SET and GET Variable processing to indicate the end of SNMP variable processing. For multi byte data request, the end byte will be always SNMP_END_OF_VAR.
	SNMP_INDEX_INVALID	This Macro is used for both SNMP SET and GET Sequence Variable processing. SNMP starts processing the start of sequence variable with Invalid index. TCPIP_SNMP_ExactIndexGet and TCPIP_SNMP_NextIndexGet returns a valid index as per SNMP_INDEX_INVALID.
	SNMP_START_OF_VAR	This Macro is used for both SNMP SET and GET Variable processing to indicate the start of SNMP variable processing. For multi byte data request, the first byte will be always SNMP_START_OF_VAR.
	SNMP_V1	This macro is used for SNMP version 1
	SNMP_V2C	This macro is used for SNMP version 2 with community
	SNMP_V3	This macro is used for SNMP version 3 with authentication and privacy

Structures

	Name	Description
	SNMP_NON_MIB_RECD_INFO	Restrict access for specific OIDs.

Types

	Name	Description
	SNMP_ID	SNMP dynamic variable ID.
	SNMP_INDEX	SNMP sequence variable index.

Unions

	Name	Description
	SNMP_VAL	Definition to represent SNMP OID object values.

Description

SNMP Definitions for the Microchip TCP/IP Stack

Simple Network Management Protocol (SNMP) is one of the key components of a Network Management System (NMS). SNMP is an application layer protocol that facilitates the exchange of management information among network devices. It is a part of the TCP/IP protocol suite. The SNMP Manager uses the UDP Port Number 161 to send requests to the agent. The agent uses the UDP Port Number 162 to send Trap(s) or notification events to the manager.

- Supports SNMP Version V1 and V2c over User Datagram Protocol (UDP)
- Supports Get, Get_Bulk, Get_Next, Set and Trap Protocol Data Units (PDUs)
- Supports up to 255 dynamic OIDs and unlimited constant OIDs
- mib2bib.jar utility is Microchip MIB compiler. It accepts Microchip MIB script

in ASCII format and generates two output files: The binary information file, snmp.bib and the C header file , mib.h.

File Name

snmp.h

Company

Microchip Technology Inc.

snmpv3.h

Simple Network Management Protocol Version3(SNMPv3) API header file.

Enumerations

	Name	Description
	SNMPV3_HMAC_HASH_TYPE	Different type of authentication for SNMPv3.
	SNMPV3_PRIV_PROT_TYPE	Different type of encryption and decryption for SNMPv3.
	STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL	Different SNMP Message processing model
	STD_BASED_SNMP_SECURITY_MODEL	Different Security services for SNMPv3 messages.
	STD_BASED_SNMPV3_SECURITY_LEVEL	Different Security Level for SNMPv3 messages.
	TCPIP_SNMPV3_USERDATABASECONFIG_TYPE	Different Configuration parameters of SNMPv3 operation

Functions

	Name	Description
	TCPIP_SNMPV3_EngineUserDataBaseGet	Get SNMPv3 engine data base details.
	TCPIP_SNMPV3_EngineUserDataBaseSet	Set SNMPv3 engine data base details.
	TCPIP_SNMPv3_Notify	Creates and Sends SNMPv3 TRAP PDU.
	TCPIP_SNMPv3_TrapConfigDataGet	Gets the SNMPv3 Trap configuration details using the user name index.

Structures

	Name	Description
	TCPIP_SNMP_COMMUNITY_CONFIG	SNMP community configuration.
	TCPIP_SNMP_MODULE_CONFIG	SNMP module configuration.
	TCPIP_SNMPV3_TARGET_ENTRY_CONFIG	SNMP module trap target address configuration.
	TCPIP_SNMPV3_USM_USER_CONFIG	SNMPv3 USM configuration.

Description

Simple Network Management Protocol (SNMP) v3 API Header File

SNMPv3 provides a secure access to the devices using a combination of authentication and encryption of packets over the network. Comparing with SNMPv1/v2, SNMPv3 ensures that packets can be collected securely from SNMP devices. Microchip SNMPv3 Agent authentication is implemented with MD5 and SHA1 protocols and encryption is implemented with AES protocol. These credentials and other user information are stored in the global array. The user of the SNMPv3 stack can decide on the number of user names in the Userâ€™s data base to be stored with the Server. According to the SNMPv3 recommendation, SNMPv3 server should not be configured with the authentication and privacy passwords. Instead could be configured with the respective localized keys of the password. Microchip SNMPv3 agent is provided with the password information in the database for the â€œGetting Startedâ€• and for understanding purpose only. It is recommended that the SNMPv3 stack should be modified to restrict access to the password OIDs declared in the user data base.

File Name

snmpv3.h

Company

Microchip Technology Inc.

snmp_config.h

SNMP configuration file

Macros

	Name	Description
	TCPIP_SNMP_BIB_FILE_NAME	The Microchip mib2bib.jar compiler is used to convert the Microchip MIB script to binary format and it is compatible with the Microchip SNMP agent. which is written in ASCII format. Name of the bib file for SNMP is snmp.bib.
	TCPIP_SNMP_COMMUNITY_MAX_LEN	This is the maximum length for community string. Application must ensure that this length is observed. SNMP module adds one byte extra after TCPIP_SNMP_COMMUNITY_MAX_LEN for adding '0' NULL character.
	TCPIP_SNMP_MAX_COMMUNITY_SUPPORT	Specifying more strings than TCPIP_SNMP_MAX_COMMUNITY_SUPPORT will result in the later strings being ignored (but still wasting program memory). Specifying fewer strings is legal, as long as at least one is present.
	TCPIP_SNMP_MAX_MSG_SIZE	The maximum length in octets of an SNMP message which this SNMP agent able to process. As per RFC 3411 snmpEngineMaxMessageSize and RFC 1157 (section 4- protocol specification) and implementation supports more than 480 whenever feasible. It should be divisible by 16
	TCPIP_SNMP_MAX_NON_REC_ID_OID	Update the Non record id OID value which is part of CustomSnmpDemoApp.c file. This is the maximum size for gSnmpNonMibRecInfo[] which is the list of static variable Parent OIDs which are not part of mib.h file. This structure is used to restrict access to static variables of SNMPv3 OIDs from SNMPv2c and SNMPv1 version. With SNMPv3 all the OIDs accessible but when we are using SNMPv2c version , static variables of the SNMPv3 cannot be accessible with SNMP version v2c. SNMP agent supports both SMIv1 and SMIv2 standard and snmp.mib has been updated with respect to SMIV2 standard and it... more
	TCPIP_SNMP_NOTIFY_COMMUNITY_LEN	Maximum length for SNMP Trap community name
	TCPIP_SNMP_OID_MAX_LEN	Maximum length for the OID String. Change this to match your OID string length.
	TCPIP_SNMP_TASK_PROCESS_RATE	SNMP task processing rate, in milli-seconds. The SNMP module will process a timer event with this rate for processing its own state machine, etc. The default value is 200 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN	The maximum size of TRAP community string length
	TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE	Trap information. This macro will be used to avoid SNMP OID memory buffer corruption
	TCPIP_SNMP_TRAP_TABLE_SIZE	This table maintains list of interested receivers who should receive notifications when some interesting event occurs.

Description

Simple Network Management Protocol (SNMP) Configuration file

This file contains the SNMP module configuration options

File Name

snmp_config.h

Company

Microchip Technology Inc.

snmpv3_config.h

SNMPv3 configuration file

Macros

	Name	Description
	TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN	SNMPv3 Authentication Localized password key length size
	TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE	SNMPv3 authentication localized Key length for memory validation

	TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN	SNMPv3 Privacy Password key length size
	TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE	SNMPv3 privacy key length size for memory validation
	TCPIP_SNMPV3_USER_SECURITY_NAME_LEN	Maximum size for SNMPv3 User Security Name length.
	TCPIP_SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE	User security name length for memory validation
	TCPIP_SNMPV3_USM_MAX_USER	Maximum number of SNMPv3 users. User Security Model should have at least 1 user. Default is 3.

Description

Simple Network Management Protocol (SNMPv3) Configuration file

This file contains the SNMPv3 module configuration options

File Name

snmpv3_config.h

Company

Microchip Technology Inc.

SNTP Module

This section describes the TCP/IP Stack Library SNTP module.

Introduction

TCP/IP Stack Library Simple Network Time Protocol (SNTP) Module for Microchip Microcontrollers

This library provides the API of the SNTP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

The SNTP module implements the Simple Network Time Protocol. The module (by default) updates its internal time every 10 minutes using a pool of public global time servers. It then calculates reference times on any call to SNTPGetUTCSeconds using the internal Tick timer module.

The SNTP module is good for providing absolute time stamps. However, it should not be relied upon for measuring time differences (especially small differences). The pool of public time servers is implemented using round-robin DNS, so each update will come from a different server. Differing network delays and the fact that these servers are not verified implies that this time could be non-linear. While it is deemed reliable, it is not guaranteed to be accurate.

The Tick module provides much better accuracy (since it is driven by a hardware clock) and resolution, and should be used for measuring timeouts and other internal requirements.

Developers can change the value of NTP_SERVER if they want to always point to a preferred time server, or to specify a region when accessing time servers. The default is to use the global pool.

Using the Library

This topic describes the basic architecture of the SNTP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `sntp.h`

The interface to the SNTP TCP/IP Stack library is defined in the `sntp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the SNTP TCP/IP Stack library should include `tcpip.h`.

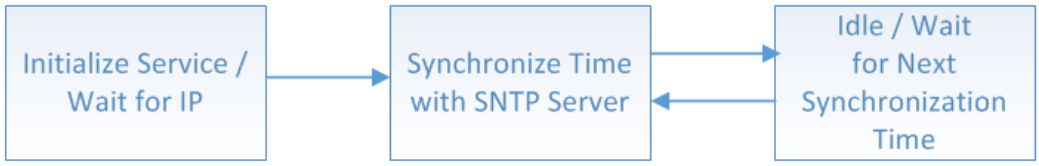
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the SNTP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

SNTP Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the SNTP module.

Library Interface Section	Description
Functions	Routines for configuring SNTP
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_NTP_DEFAULT_CONNECTION_TYPE	The default connection type to use: IPv4/IPv6
	TCPIP_NTP_DEFAULT_IF	for multi-homed hosts, the default SNTP interface
	TCPIP_NTP_EPOCH	Reference Epoch to use. (default: 01-Jan-1970 00:00:00)
	TCPIP_NTP_FAST_QUERY_INTERVAL	Defines how long to wait to retry an update after a failure, seconds. Updates may take up to 6 seconds to fail, so this 14 second delay is actually only an 8-second retry.
	TCPIP_NTP_MAX_STRATUM	The maximum acceptable NTP stratum number Should be less than 16 (unsynchronized server)
	TCPIP_NTP_QUERY_INTERVAL	Defines how frequently to resynchronize the date/time, seconds (default: 10 minutes)
	TCPIP_NTP_REPLY_TIMEOUT	Defines how long to wait before assuming the query has failed, seconds
	TCPIP_NTP_RX_QUEUE_LIMIT	The NTP RX queue limit defines the maximum number of packets that can wait in the NTP queue
	TCPIP_NTP_SERVER	These are normally available network time servers. The actual IP returned from the pool will vary every minute so as to spread the load around stratum 1 timeservers. For best accuracy and network overhead you should locate the pool server closest to your geography, but it will still work if you use the global pool.ntp.org address or choose the wrong one or ship your embedded device to another geography. A direct IP address works too
	TCPIP_NTP_SERVER_MAX_LENGTH	maximum number of characters allowed for the NTP server
	TCPIP_NTP_TASK_TICK_RATE	THE NTP task rate, in milliseconds The default value is 1100 milliseconds. This module contacts an NTP server and a high operation frequency is not required. The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_NTP_TIME_STAMP_TMO	elapsed time that qualifies a time stamp as stale normally it should be correlated with TCPIP_NTP_QUERY_INTERVAL
	TCPIP_NTP_VERSION	The default NTP version to use (3 or 4)

Description

The configuration of the SNTP TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the SNTP TCP/IP Stack Library. Based on the selections made, the SNTP TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the SNTP TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_NTP_DEFAULT_CONNECTION_TYPE Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_DEFAULT_CONNECTION_TYPE (IP_ADDRESS_TYPE_IPV4)
```

Description

The default connection type to use: IPv4/IPv6

TCPIP_NTP_DEFAULT_IF Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_DEFAULT_IF "PIC32INT"
```

Description

for multi-homed hosts, the default SNTP interface

TCPIP_NTP_EPOCH Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_EPOCH (86400ul * (365ul * 70ul + 17ul))
```

Description

Reference Epoch to use. (default: 01-Jan-1970 00:00:00)

TCPIP_NTP_FAST_QUERY_INTERVAL Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_FAST_QUERY_INTERVAL (14ul)
```

Description

Defines how long to wait to retry an update after a failure, seconds. Updates may take up to 6 seconds to fail, so this 14 second delay is actually only an 8-second retry.

TCPIP_NTP_MAX_STRATUM Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_MAX_STRATUM (15)
```

Description

The maximum acceptable NTP stratum number Should be less than 16 (unsynchronized server)

TCPIP_NTP_QUERY_INTERVAL Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_QUERY_INTERVAL (10ul*60ul)
```

Description

Defines how frequently to resynchronize the date/time, seconds (default: 10 minutes)

TCPIP_NTP_REPLY_TIMEOUT Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_REPLY_TIMEOUT (6ul)
```

Description

Defines how long to wait before assuming the query has failed, seconds

TCPIP_NTP_RX_QUEUE_LIMIT Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_RX_QUEUE_LIMIT (2)
```

Description

The NTP RX queue limit defines the maximum number of packets that can wait in the NTP queue

TCPIP_NTP_SERVER Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_SERVER "pool.ntp.org"
```

Description

These are normally available network time servers. The actual IP returned from the pool will vary every minute so as to spread the load around stratum 1 timeservers. For best accuracy and network overhead you should locate the pool server closest to your geography, but it will still work if you use the global pool.ntp.org address or choose the wrong one or ship your embedded device to another geography. A direct IP address works too

TCPIP_NTP_SERVER_MAX_LENGTH Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_SERVER_MAX_LENGTH 30
```

Description

maximum number of characters allowed for the NTP server

TCPIP_NTP_TASK_TICK_RATE Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_TASK_TICK_RATE (1100)
```

Description

THE NTP task rate, in milliseconds The default value is 1100 milliseconds. This module contacts an NTP server and a high operation frequency is not required. The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_NTP_TIME_STAMP_TMO Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_TIME_STAMP_TMO (11 * 60)
```

Description

elapsed time that qualifies a time stamp as stale normally it should be correlated with TCPIP_NTP_QUERY_INTERVAL

TCPIP_NTP_VERSION Macro

File

sntp_config.h

C

```
#define TCPIP_NTP_VERSION (4)
```

Description

The default NTP version to use (3 or 4)

Building the Library

This section lists the files that are available in the SNTP module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/sntp.c`	SNTP implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The SNTP module depends on the following modules:

- TCP/IP Stack Library
- DNS Module
- UDP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_SNTP_ConnectionInitiate	Forces a connection to the NTP server
	TCPIP_SNTP_LastErrorGet	Gets the last error code set in the NTP server.
	TCPIP_SNTP_UTCSecondsGet	Obtains the current time from the SNTP module.
	TCPIP_SNTP_TimeStampGet	Gets the last valid timestamp obtained from an NTP server.
	TCPIP_SNTP_ConnectionParamSet	Sets the current SNTP connection parameters.
	TCPIP_SNTP_Task	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	TCPIP_SNTP_MODULE_CONFIG	Placeholder for SNTP Module Configuration
	TCPIP_SNTP_RESULT	Provides a list of possible results for the SNTP module.

Description

This section describes the Application Programming Interface (API) functions of the SNTP module.
Refer to each section for a detailed description.

a) Functions

TCPIP_SNTP_ConnectionInitiate Function

Forces a connection to the NTP server

File

sntp.h

C

```
TCPIP_SNTP_RESULT TCPIP_SNTP_ConnectionInitiate();
```

Returns

- SNTP_RES_OK - if the call succeeded
- SNTP_RES_PROGRESS - if a connection already in progress another one could not be started

Description

This function will start a connection to the NTP server

Remarks

None.

Preconditions

The TCP/IP Stack should have been initialized.

Function

TCPIP_SNTP_RESULT TCPIP_SNTP_ConnectionInitiate(void);

TCPIP_SNTP_LastErrorGet Function

Gets the last error code set in the NTP server.

File

sntp.h

C

```
TCPIP_SNTP_RESULT TCPIP_SNTP_LastErrorGet();
```

Returns

The last error code encountered by the NTP module.

Description

This function returns the last NTP error code and clears the current error code.

Remarks

None/

Preconditions

The TCP/IP Stack should have been initialized.

Function

TCPIP_SNTP_RESULT TCPIP_SNTP_LastErrorGet(void);

TCPIP_SNTP_UTCSecondsGet Function

Obtains the current time from the SNTP module.

File

sntp.h

C

```
uint32_t TCPIP_SNTP_UTCSecondsGet();
```

Returns

The number of seconds since the Epoch. (Default 01-Jan-1970 00:00:00)

Description

This function obtains the current time as reported by the SNTP module. Use this value for absolute time stamping. The value returned is (by default) the number of seconds since 01-Jan-1970 00:00:00.

Remarks

Do not use this function for time difference measurements. The Tick module is more appropriate for those requirements.

Preconditions

The TCP/IP Stack should have been initialized.

Function

uint32_t TCPIP_SNTP_UTCSecondsGet(void)

TCPIP_SNTP_TimeStampGet Function

Gets the last valid timestamp obtained from an NTP server.

File

sntp.h

C

```
TCPIP_SNTP_RESULT TCPIP_SNTP_TimeStampGet(uint64_t* pTStamp, uint32_t* pLastUpdate);
```

Returns

- SNTP_RES_OK - if the call succeeded
- SNTP_RES_TSTAMP_STALE error code - if there is no recent timestamp

Description

This function gets the last valid timestamp obtained from an NTP server.

Remarks

None.

Preconditions

The TCP/IP Stack should have been initialized.

Parameters

Parameters	Description
pTStamp	pointer to a 64 bit buffer to store the last NTP timestamp could be NULL if the timestamp not needed
pLastUpdate	pointer to store the last time stamp update tick could be NULL if the update time not needed

Function

TCPIP_SNTP_RESULT TCPIP_SNTP_TimeStampGet(uint64_t* pTStamp, uint32_t* pLastUpdate);

TCPIP_SNTP_ConnectionParamSet Function

Sets the current SNTP connection parameters.

File

sntp.h

C

```
TCPIP_SNTP_RESULT TCPIP_SNTP_ConnectionParamSet(TCPIP_NET_HANDLE netH, IP_ADDRESS_TYPE ntpConnType, const char* ntpServer);
```

Returns

- SNTP_RES_OK - if the call succeeded
- TCPIP_SNTP_RESULT error code - if the call did not succeed

Description

This function sets the parameters for the next SNTP connections.

Remarks

None.

Preconditions

The TCP/IP Stack should have been initialized.

Parameters

Parameters	Description
netH	new interface to use as default SNTP interface if 0, the current interface is not changed
ntpConnType	type of connection to make: IPv4 or IPv6 if IP_ADDRESS_TYPE_ANY, the current setting is not changed
ntpServer	the NTP server to be used; name or an IP address can be used if 0, the current NTP server is not changed

Function

TCPIP_SNTP_RESULT TCPIP_SNTP_ConnectionParamSet(TCPIP_NET_HANDLE netH, IP_ADDRESS_TYPE ntpConnType, const char* ntpServer);

TCPIP_SNTP_Task Function

Standard TCP/IP stack module task function.

File

sntp.h

C

```
void TCPIP_SNTP_Task();
```

Returns

None.

Description

This function performs SNTP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The SNTP module should have been initialized.

Function

void TCPIP_SNTP_Task(void)

b) Data Types and Constants

TCPIP_SNTP_MODULE_CONFIG Structure

File

sntp.h

C

```
typedef struct {
  const char* ntp_server;
  const char* ntp_interface;
  IP_ADDRESS_TYPE ntp_connection_type;
  uint32_t ntp_reply_timeout;
  uint32_t ntp_stamp_timeout;
  uint32_t ntp_success_interval;
  uint32_t ntp_error_interval;
} TCPIP_SNTP_MODULE_CONFIG;
```

Members

Members	Description
const char* ntp_server;	the NTP server to contact; name or IP address
const char* ntp_interface;	the default NTP interface to use
IP_ADDRESS_TYPE ntp_connection_type;	the IPv4/IPv6 connection type
uint32_t ntp_reply_timeout;	timeout for the server reply in seconds
uint32_t ntp_stamp_timeout;	timeout for the time stamp in seconds
uint32_t ntp_success_interval;	server query interval after an successful update, in seconds
uint32_t ntp_error_interval;	server query interval after an error, in seconds

Description

Placeholder for SNTP Module Configuration

TCPIP_SNTP_RESULT Enumeration

Provides a list of possible results for the SNTP module.

File

sntp.h

C

```
typedef enum {
  SNTP_RES_OK,
  SNTP_RES_PROGRESS,
  SNTP_RES_BUSY = -1,
  SNTP_RES_TSTAMP_STALE = -2,
  SNTP_RES_SKT_ERR = -3,
```

```
    SNTP_RES_SKT_BIND_ERR = -4,
    SNTP_RES_NTP_SERVER_TMO = -5,
    SNTP_RES_NTP_VERSION_ERR = -6,
    SNTP_RES_NTP_TSTAMP_ERR = -7,
    SNTP_RES_NTP_SYNC_ERR = -8,
    SNTP_RES_NTP_KOD_ERR = -9,
    SNTP_RES_NTP_DNS_ERR = -10,
    SNTP_RES_NTP_IF_ERR = -11,
    SNTP_RES_NTP_CONN_ERR = -12
} TCPIP_SNTP_RESULT;
```

Members

Members	Description
SNTP_RES_OK	the operation was successful
SNTP_RES_PROGRESS	an NTP operation is in progress
SNTP_RES_BUSY = -1	module is busy
SNTP_RES_TSTAMP_STALE = -2	timestamp is stale, there's no recent timestamp
SNTP_RES_SKT_ERR = -3	NTP socket could not be opened
SNTP_RES_SKT_BIND_ERR = -4	NTP socket bind failed
SNTP_RES_NTP_SERVER_TMO = -5	NTP server could not be accessed
SNTP_RES_NTP_VERSION_ERR = -6	wrong NTP version received
SNTP_RES_NTP_TSTAMP_ERR = -7	wrong NTP time stamp received
SNTP_RES_NTP_SYNC_ERR = -8	NTP time synchronization error
SNTP_RES_NTP_KOD_ERR = -9	an NTP KissOfDeath code has been received
SNTP_RES_NTP_DNS_ERR = -10	an NTP DNS error
SNTP_RES_NTP_IF_ERR = -11	an NTP interface error
SNTP_RES_NTP_CONN_ERR = -12	an NTP connection type error

Description

TCPIP_SNTP_RESULT Enumeration

Provides a list of possible SNTP results.

Remarks

None

Files

Files

Name	Description
sntp.h	The SNTP module implements the Simple Network Time Protocol. The module (by default) updates its internal time every 10 minutes using a pool of public global time servers.
sntp_config.h	SNTP configuration file

Description

This section lists the source and header files used by the library.

sntp.h

The SNTP module implements the Simple Network Time Protocol. The module (by default) updates its internal time every 10 minutes using a pool of public global time servers.

Enumerations

	Name	Description
	TCPIP_SNTP_RESULT	Provides a list of possible results for the SNTP module.

Functions

	Name	Description
	TCPIP_SNTP_ConnectionInitiate	Forces a connection to the NTP server
	TCPIP_SNTP_ConnectionParamSet	Sets the current SNTP connection parameters.

	TCPIP_SNTP_LastErrorGet	Gets the last error code set in the NTP server.
	TCPIP_SNTP_Task	Standard TCP/IP stack module task function.
	TCPIP_SNTP_TimeStampGet	Gets the last valid timestamp obtained from an NTP server.
	TCPIP_SNTP_UTCSecondsGet	Obtains the current time from the SNTP module.

Structures

	Name	Description
	TCPIP_SNTP_MODULE_CONFIG	Placeholder for SNTP Module Configuration

Description

SNTP Client Module API Header File

The SNTP module implements the Simple Network Time Protocol. The module (by default) updates its internal time every 10 minutes using a pool of public global time servers. It then calculates reference times on any call to SNTPGetUTCSeconds using the internal Tick timer module.

File Name

sntp.h

Company

Microchip Technology Inc.

sntp_config.h

SNTP configuration file

Macros

	Name	Description
	TCPIP_NTP_DEFAULT_CONNECTION_TYPE	The default connection type to use: IPv4/IPv6
	TCPIP_NTP_DEFAULT_IF	for multi-homed hosts, the default SNTP interface
	TCPIP_NTP_EPOCH	Reference Epoch to use. (default: 01-Jan-1970 00:00:00)
	TCPIP_NTP_FAST_QUERY_INTERVAL	Defines how long to wait to retry an update after a failure, seconds. Updates may take up to 6 seconds to fail, so this 14 second delay is actually only an 8-second retry.
	TCPIP_NTP_MAX_STRATUM	The maximum acceptable NTP stratum number Should be less than 16 (unsynchronized server)
	TCPIP_NTP_QUERY_INTERVAL	Defines how frequently to resynchronize the date/time, seconds (default: 10 minutes)
	TCPIP_NTP_REPLY_TIMEOUT	Defines how long to wait before assuming the query has failed, seconds
	TCPIP_NTP_RX_QUEUE_LIMIT	The NTP RX queue limit defines the maximum number of packets that can wait in the NTP queue
	TCPIP_NTP_SERVER	These are normally available network time servers. The actual IP returned from the pool will vary every minute so as to spread the load around stratum 1 timeservers. For best accuracy and network overhead you should locate the pool server closest to your geography, but it will still work if you use the global pool.ntp.org address or choose the wrong one or ship your embedded device to another geography. A direct IP address works too
	TCPIP_NTP_SERVER_MAX_LENGTH	maximum number of characters allowed for the NTP server
	TCPIP_NTP_TASK_TICK_RATE	THE NTP task rate, in milliseconds The default value is 1100 milliseconds. This module contacts an NTP server and a high operation frequency is not required. The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_NTP_TIME_STAMP_TMO	elapsed time that qualifies a time stamp as stale normally it should be correlated with TCPIP_NTP_QUERY_INTERVAL
	TCPIP_NTP_VERSION	The default NTP version to use (3 or 4)

Description

Network Time Protocol (SNTP) Configuration file

This file contains the SNTP module configuration options

File Name

sntp_config.h

Company

Microchip Technology Inc.

TCP Module

This section describes the TCP/IP Stack Library TCP module.

Introduction

TCP/IP Stack Library Transmission Control Protocol (TCP) Module for Microchip Microcontrollers

This library provides the API of the TCP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

TCP is a standard transport layer protocol described in RFC 793. It provides reliable stream-based connections over unreliable networks, and forms the foundation for HTTP, SMTP, and many other protocol standards.

Using the Library

This topic describes the basic architecture of the TCP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `tcp.h`

The interface to the TCP TCP/IP Stack library is defined in the `tcp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the TCP TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

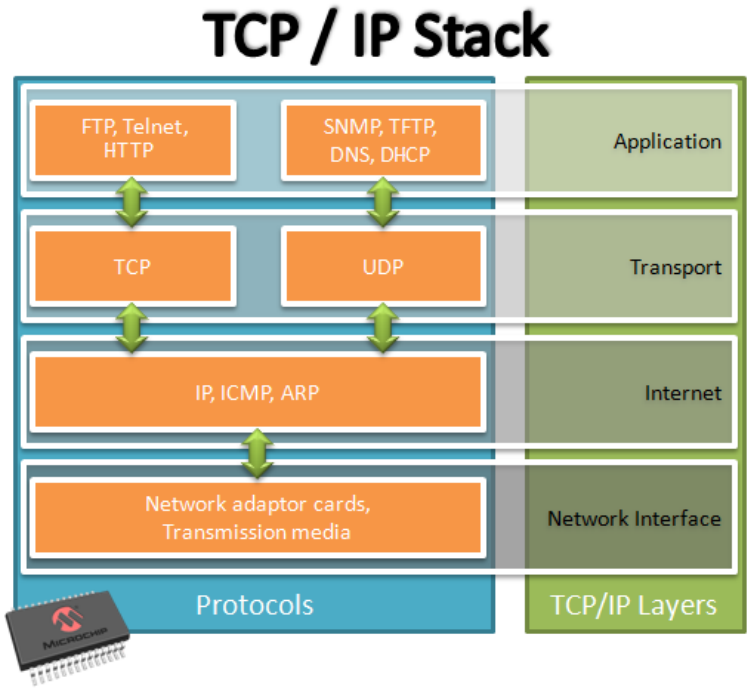
This library provides the API of the TCP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

TCP Software Abstraction Block Diagram

This module provides software abstraction of the TCP module existent in any TCP/IP Stack implementation.

Typical TCP Implementation



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the TCP module.

Library Interface Section	Description
Socket Management Functions	Routines for Managing TCP Sockets
Transmit Data Transfer Functions	Routines for Managing Outgoing Data Transmissions
Receive Data Transfer Functions	Routines for Managing Incoming Data Transmissions
Data Types and Constants	This section provides various definitions describing This API

How the Library Works

This topic describes how the TCP/IP Stack Library TCP module works.

Description

Connections made over TCP guarantee data transfer at the expense of throughput. Connections are made through a three-way handshake process, ensuring a one-to-one connection. Remote nodes advertise how much data they are ready to receive, and all data transmitted must be acknowledged. If a remote node fails to acknowledge the receipt of data, it is automatically retransmitted. This ensures that network errors such as lost, corrupted, or out-of-order packets are automatically corrected.

To accomplish this, TCP must use buffers for both the transmit and receive operations. Once the transmit buffer is full, no more data can be sent until the remote node has acknowledged receipt. For the Microchip TCP/IP Stack, the application must return to the main system loop to allow the other services in the system, including the TCP/IP stack, to advance.. Likewise, the remote node cannot transmit more data until the local device has acknowledged receipt and that space is available in the buffer. When a local application needs to read more data, it must return to the main system/application loop and wait for a new packet to arrive.

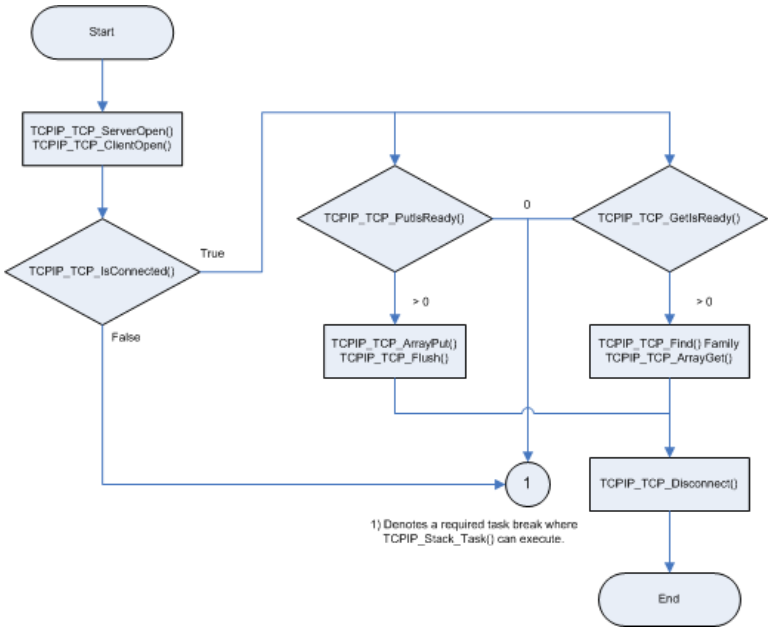
Core Functionality

This topic describes the core functionality of the TCP/IP Stack Library TCP module.

Description

The following diagram provides an overview for the use of the TCP module.

TCP Connection Flow



Server sockets are opened using TCPIP_TCP_ServerOpen. This function will open a listening socket waiting for client connections on the specified TCP port number. A client socket is opened using TCPIP_TCP_ClientOpen. This function will open a socket that connects to a remote host. The remote host is specified using an IP address and a TCP port number.

Once connected, applications can read and write data. On each entry, the application must verify that the socket is still connected. For most applications a call to TCPIP_TCP_IsConnected will be sufficient, but TCPIP_TCP_WasReset may also be used for listening sockets that may turn over quickly.

To write data, call TCPIP_TCP_PutIsReady to check how much space is available. Then, call the TCPIP_TCP_ArrayPut function to write data as space is available. Once complete, call TCPIP_TCP_Flush to transmit data immediately. Alternately, return to the main system/stack loop. Data will be transmitted by the internal TCP state machine as the buffer becomes full or a programmable delay time has passed.

To read data, call TCPIP_TCP_GetIsReady to determine how many bytes are ready to be retrieved. Then use the TCPIP_TCP_ArrayGet function to read data from the socket, and/or the TCPIP_TCP_Find family of functions to locate data in the buffer. When no more data remains, return to the main system/stack loop to wait for more data to arrive.

If the application needs to close the connection, call TCPIP_TCP_Disconnect, and then return to the main system/stack loop and wait for the remote node to acknowledge the disconnection. Client sockets will be closed and associated resources freed, while listening sockets will wait for a new connection. For more information, refer to the examples provided with the MPLAB Harmony distribution examples, or read the associated RFC.

Configuring the Library

Macros

	Name	Description
	TCPIP_TCP_AUTO_TRANSMIT_TIMEOUT_VAL	Timeout before automatically transmitting unflushed data, ms. Default value is 40 ms.
	TCPIP_TCP_CLOSE_WAIT_TIMEOUT	Timeout for the CLOSE_WAIT state, ms If the remote socket closes the connection the TCP/IP stack will automatically close the socket after this timeout. If the value is set to 0 then the socket will wait forever for the application to close the socket (default behavior).
	TCPIP_TCP_DELAYED_ACK_TIMEOUT	Timeout for delayed-acknowledgment algorithm, ms
	TCPIP_TCP_FIN_WAIT_2_TIMEOUT	Timeout for FIN WAIT 2 state, ms
	TCPIP_TCP_KEEP_ALIVE_TIMEOUT	Timeout for keep-alive messages when no traffic is sent, ms
	TCPIP_TCP_MAX_RETRIES	Maximum number of retransmission attempts
	TCPIP_TCP_MAX_SEG_SIZE_RX_LOCAL	TCP Maximum Segment Size for RX (MSS). This value is advertised during TCP connection establishment and the remote node should obey it. The value has to be set in such a way to avoid IP layer fragmentation from causing packet loss. However, raising its value can enhance performance at the (small) risk of introducing incompatibility with certain special remote nodes (ex: ones connected via a slow dial up modem). On Ethernet networks the standard value is 1460. On dial-up links, etc. the default values should be 536. Adjust these values according to your network. Maximum Segment Size for RX (MSS)... more

	TCPIP_TCP_MAX_SEG_SIZE_RX_NON_LOCAL	Maximum Segment Size for RX (MSS) for non local destination networks.
	TCPIP_TCP_MAX_SEG_SIZE_TX	TCP Maximum Segment Size for TX. The TX maximum segment size is actually governed by the remote node's MSS option advertised during connection establishment. However, if the remote node specifies an unmanageably large MSS (ex: > Ethernet MTU), this define sets a hard limit so that TX buffers are not overflowed. If the remote node does not advertise a MSS option, all TX segments are fixed at 536 bytes maximum.
	TCPIP_TCP_MAX_SOCKETS	The maximum number of sockets to create in the stack. When defining TCPIP_TCP_MAX_SOCKETS take into account the number of interfaces the stack is supporting.
	TCPIP_TCP_MAX_SYN_RETRIES	Smaller than all other retries to reduce SYN flood DoS duration
	TCPIP_TCP_MAX_UNACKED_KEEP_ALIVES	Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection
	TCPIP_TCP_SOCKET_DEFAULT_RX_SIZE	Default socket RX buffer size Note that this setting affects all TCP sockets that are created and, together with TCPIP_TCP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large RX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_TCP_OptionsSet function). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Note that some modules (like HTTP) use their own settings to... more
	TCPIP_TCP_SOCKET_DEFAULT_TX_SIZE	Default socket TX buffer size Note that this setting affects all TCP sockets that are created and, together with TCPIP_TCP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large TX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_TCP_OptionsSet function). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Note that some modules (like HTTP) use their own settings to... more
	TCPIP_TCP_START_TIMEOUT_VAL	Timeout to retransmit unacked data, ms
	TCPIP_TCP_TASK_TICK_RATE	The TCP task processing rate: number of milliseconds to generate an TCP tick. This is the tick that advances the TCP state machine. The default value is 5 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_TCP_WINDOW_UPDATE_TIMEOUT_VAL	Timeout before automatically transmitting a window update due to a TCPIP_TCP_Get() or TCPIP_TCP_ArrayGet() function call, ms.
	TCPIP_TCP_DYNAMIC_OPTIONS	Enable the TCP sockets dynamic options set/get functionality If enabled, the functions: TCPIP_TCP_OptionsSet, TCPIP_TCP_OptionsGet and TCPIP_TCP_FifoSizeAdjust exist and are compiled in If disabled, these functions do not exist and cannot be used/called Note that this setting can affect modules that use TCP sockets
	TCPIP_TCP_MSL_TIMEOUT	Maximum Segment Length (MSL) timeout, seconds This value sets the time a socket will be in the TIME_WAIT/2MSL state after the socket performed an active close. RFC recommended value is 120 sec. Other common values are: 30/60 sec Note that if this symbol is defined to 0, then the TIME_WAIT/2MSL state is skipped! This could lead to duplicate sequence numbers problems if sockets are frequently/rapidly closed and reopened and should be avoided.
	TCPIP_TCP_QUIET_TIME	This value specifies the TCP quiet time: TCP will not send any data for TCPIP_TCP_QUIET_TIME seconds after rebooting! Usualy this value should match the MSL value. Note that if this symbol is defined to 0, then the TCP quiet time state is skipped! During the quiet time there will be no TCP traffic initiated by TCP. So, clients won't be able to connect to remote hosts and servers won't reply to client requests.

Description

The configuration of the TCP/IP Stack Library TCP module is based on the file `system_config.h` (which may include `tcp_config.h`).

This header file contains the configuration selection for the TCP/IP Stack Library TCP module. Based on the selections made, the TCP/IP Stack Library TCP module may support the selected features. These configuration settings will apply to all TCP sockets of the TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_TCP_AUTO_TRANSMIT_TIMEOUT_VAL Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_AUTO_TRANSMIT_TIMEOUT_VAL (40ul)
```

Description

Timeout before automatically transmitting unflushed data, ms. Default value is 40 ms.

TCPIP_TCP_CLOSE_WAIT_TIMEOUT Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_CLOSE_WAIT_TIMEOUT (200ul)
```

Description

Timeout for the CLOSE_WAIT state, ms If the remote socket closes the connection the TCP/IP stack will automatically close the socket after this timeout. If the value is set to 0 then the socket will wait forever for the application to close the socket (default behavior).

TCPIP_TCP_DELAYED_ACK_TIMEOUT Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_DELAYED_ACK_TIMEOUT (100ul)
```

Description

Timeout for delayed-acknowledgment algorithm, ms

TCPIP_TCP_FIN_WAIT_2_TIMEOUT Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_FIN_WAIT_2_TIMEOUT (5000ul)
```

Description

Timeout for FIN WAIT 2 state, ms

TCPIP_TCP_KEEP_ALIVE_TIMEOUT Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_KEEP_ALIVE_TIMEOUT (10000ul)
```

Description

Timeout for keep-alive messages when no traffic is sent, ms

TCPIP_TCP_MAX_RETRIES Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_RETRIES (5u)
```

Description

Maximum number of retransmission attempts

TCPIP_TCP_MAX_SEG_SIZE_RX_LOCAL Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_SEG_SIZE_RX_LOCAL (1460)
```

Description

TCP Maximum Segment Size for RX (MSS). This value is advertised during TCP connection establishment and the remote node should obey it. The value has to be set in such a way to avoid IP layer fragmentation from causing packet loss. However, raising its value can enhance performance at the (small) risk of introducing incompatibility with certain special remote nodes (ex: ones connected via a slow dial up modem). On Ethernet networks the standard value is 1460. On dial-up links, etc. the default values should be 536. Adjust these values according to your network.

Maximum Segment Size for RX (MSS) for local destination networks.

TCPIP_TCP_MAX_SEG_SIZE_RX_NON_LOCAL Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_SEG_SIZE_RX_NON_LOCAL (536)
```

Description

Maximum Segment Size for RX (MSS) for non local destination networks.

TCPIP_TCP_MAX_SEG_SIZE_TX Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_SEG_SIZE_TX (1460u)
```

Description

TCP Maximum Segment Size for TX. The TX maximum segment size is actually governed by the remote node's MSS option advertised during connection establishment. However, if the remote node specifies an unmanageably large MSS (ex: > Ethernet MTU), this define sets a hard limit so that TX buffers are not overflowed. If the remote node does not advertise a MSS option, all TX segments are fixed at 536 bytes maximum.

TCPIP_TCP_MAX_SOCKETS Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_SOCKETS (10)
```

Description

The maximum number of sockets to create in the stack. When defining TCPIP_TCP_MAX_SOCKETS take into account the number of interfaces the stack is supporting.

TCPIP_TCP_MAX_SYN_RETRIES Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_SYN_RETRIES (2u)
```

Description

Smaller than all other retries to reduce SYN flood DoS duration

TCPIP_TCP_MAX_UNACKED_KEEP_ALIVES Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MAX_UNACKED_KEEP_ALIVES (6u)
```

Description

Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection

TCPIP_TCP_SOCKET_DEFAULT_RX_SIZE Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_SOCKET_DEFAULT_RX_SIZE 512
```

Description

Default socket RX buffer size Note that this setting affects all TCP sockets that are created and, together with TCPIP_TCP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large RX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_TCP_OptionsSet function). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Note that some modules (like HTTP) use their own settings to specify the buffer size for their TCP sockets.

TCPIP_TCP_SOCKET_DEFAULT_TX_SIZE Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_SOCKET_DEFAULT_TX_SIZE 512
```

Description

Default socket TX buffer size Note that this setting affects all TCP sockets that are created and, together with TCPIP_TCP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large TX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_TCP_OptionsSet function). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Note that some modules (like HTTP) use their own settings to specify the buffer size for their TCP sockets.

TCPIP_TCP_START_TIMEOUT_VAL Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_START_TIMEOUT_VAL (1000ul)
```

Description

Timeout to retransmit unacked data, ms

TCPIP_TCP_TASK_TICK_RATE Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_TASK_TICK_RATE (5)
```

Description

The TCP task processing rate: number of milliseconds to generate an TCP tick. This is the tick that advances the TCP state machine. The default value is 5 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_TCP_WINDOW_UPDATE_TIMEOUT_VAL Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_WINDOW_UPDATE_TIMEOUT_VAL (200ul)
```

Description

Timeout before automatically transmitting a window update due to a TCPIP_TCP_Get() or TCPIP_TCP_ArrayGet() function call, ms.

TCPIP_TCP_DYNAMIC_OPTIONS Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_DYNAMIC_OPTIONS 1
```

Description

Enable the TCP sockets dynamic options set/get functionality If enabled, the functions: TCPIP_TCP_OptionsSet, TCPIP_TCP_OptionsGet and TCPIP_TCP_FifoSizeAdjust exist and are compiled in If disabled, these functions do not exist and cannot be used/called Note that this setting can affect modules that use TCP sockets

TCPIP_TCP_MSL_TIMEOUT Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_MSL_TIMEOUT (30)
```

Description

Maximum Segment Length (MSL) timeout, seconds This value sets the time a socket will be in the TIME_WAIT/2MSL state after the socket

performed an active close. RFC recommended value is 120 sec. Other common values are: 30/60 sec Note that if this symbol is defined to 0, then the TIME_WAIT/2MSL state is skipped! This could lead to duplicate sequence numbers problems if sockets are frequently/rapidly closed and reopened and should be avoided.

TCPIP_TCP_QUIET_TIME Macro

File

tcp_config.h

C

```
#define TCPIP_TCP_QUIET_TIME (30)
```

Description

This value specifies the TCP quiet time: TCP will not send any data for TCPIP_TCP_QUIET_TIME seconds after rebooting! Usualy this value should match the MSL value. Note that if this symbol is defined to 0, then the TCP quiet time state is skipped! During the quiet time there will be no TCP traffic initiated by TCP. So, clients won't be able to connect to remote hosts and servers won't reply to client requests.

Building the Library

This section lists the files that are available in the TCP module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/tcp.c`	TCP implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The TCP module depends on the following modules:

- TCP/IP Stack Library
- IPv4 Module

Library Interface

a) Socket Management Functions

	Name	Description
	TCPIP_TCP_ServerOpen	Opens a TCP socket as a server.
	TCPIP_TCP_ClientOpen	Opens a TCP socket as a client.

	TCPIP_TCP_Close	Disconnects an open socket and destroys the socket handle, releasing the associated resources.
	TCPIP_TCP_Connect	Connects a client socket.
	TCPIP_TCP_Bind	Binds a socket to a local address.
	TCPIP_TCP_RemoteBind	Binds a socket to a remote address.
	TCPIP_TCP_IsConnected	Determines if a socket has an established connection.
	TCPIP_TCP_WasReset	Self-clearing semaphore indicating socket reset.
	TCPIP_TCP_Disconnect	Disconnects an open socket.
	TCPIP_TCP_Abort	Aborts a connection.
	TCPIP_TCP_OptionsGet	Allows getting the options for a socket like: current RX/TX buffer size, etc.
	TCPIP_TCP_OptionsSet	Allows setting options to a socket like adjust RX/TX buffer size, etc.
	TCPIP_TCP_SocketInfoGet	Obtains information about a currently open socket.
	TCPIP_TCP_SocketNetGet	Gets the current network interface of an TCP socket.
	TCPIP_TCP_SocketNetSet	Sets the interface for an TCP socket
	TCPIP_TCP_SignalHandlerDeregister	Deregisters a previously registered TCP socket signal handler.
	TCPIP_TCP_SignalHandlerRegister	Registers a TCP socket signal handler.
	TCPIP_TCP_Task	Standard TCP/IP stack module task function.
	TCPIP_TCP_IsReady	Returns the current status of the TCP module
	TCPIP_TCP_SocketsNumberGet	Returns the number of existent TCP sockets.
	TCPIP_TCP_SocketTraceSet	Sets the current socket trace status.

b) Transmit Data Transfer Functions

	Name	Description
	TCPIP_TCP_Put	Writes a single byte to a TCP socket.
	TCPIP_TCP_PutIsReady	Determines how much free space is available in the TCP TX buffer.
	TCPIP_TCP_StringPut	Writes a null-terminated string to a TCP socket.
	TCPIP_TCP_ArrayPut	Writes an array from a buffer to a TCP socket.
	TCPIP_TCP_Flush	Immediately transmits all pending TX data.
	TCPIP_TCP_FifoTxFullGet	Determines how many bytes are pending in the TCP TX FIFO.
	TCPIP_TCP_FifoTxFreeGet	Determines how many bytes are free and could be written in the TCP TX FIFO.

c) Receive Data Transfer Functions

	Name	Description
	TCPIP_TCP_GetIsReady	Determines how many bytes can be read from the TCP RX buffer.
	TCPIP_TCP_ArrayGet	Reads an array of data bytes from a TCP socket's RX buffer/FIFO.
	TCPIP_TCP_ArrayPeek	Reads a specified number of data bytes from the TCP RX buffer/FIFO without removing them from the buffer.
	TCPIP_TCP_ArrayFind	Searches for a string in the TCP RX buffer.
	TCPIP_TCP_Find	Searches for a byte in the TCP RX buffer.
	TCPIP_TCP_Get	Retrieves a single byte to a TCP socket.
	TCPIP_TCP_Peek	Peaks at one byte in the TCP RX buffer/FIFO without removing it from the buffer.
	TCPIP_TCP_Discard	Discards any pending data in the RCP RX FIFO.
	TCPIP_TCP_FifoRxFreeGet	Determines how many bytes are free in the RX buffer/FIFO.
	TCPIP_TCP_FifoSizeAdjust	Adjusts the relative sizes of the RX and TX buffers.
	TCPIP_TCP_FifoRxFullGet	Determines how many bytes are pending in the RX buffer/FIFO.

d) Data Types and Constants

	Name	Description
	TCP_ADJUST_FLAGS	TCP adjust RX and TX buffers flags.
	TCP_OPTION_LINGER_DATA	Socket linger options.
	TCP_PORT	Defines a TCP Port number.
	TCP_SOCKET	Defines a TCP Socket.
	TCP_SOCKET_INFO	TCP socket information.
	TCP_SOCKET_OPTION	TCP Socket run-time options.
	INVALID_SOCKET	Invalid socket indicator macro.
	TCP_OPTION_THRES_FLUSH_TYPE	List of the socket half threshold TX flush types.

	TCPIP_TCP_MODULE_CONFIG	TCP module run-time configuration/initialization data.
	TCP_OPTION_KEEP_ALIVE_DATA	Socket keep alive options
	TCPIP_TCP_SIGNAL_FUNCTION	TCP signal handler.
	TCPIP_TCP_SIGNAL_HANDLE	TCP socket handle.
	TCPIP_TCP_SIGNAL_TYPE	TCP run time signal/event types.
	TCPIP_TCP_STATE	TCP socket state.

Description

This section describes the Application Programming Interface (API) functions of the TCP module.

Refer to each section for a detailed description.

a) Socket Management Functions

TCPIP_TCP_ServerOpen Function

Opens a TCP socket as a server.

File

tcp.h

C

```
TCP_SOCKET TCPIP_TCP_ServerOpen(IP_ADDRESS_TYPE addType, TCP_PORT localPort, IP_MULTI_ADDRESS*
localAddress);
```

Returns

- INVALID_SOCKET - No sockets of the specified type were available to be opened
- TCP_SOCKET handle - Save this handle and use it when calling all other TCP APIs

Description

Provides a unified method for opening TCP server sockets.

Sockets are created at the TCP module initialization, and can be claimed with this function and freed using TCPIP_TCP_Close.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
IP_ADDRESS_TYPE addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4.
TCP_PORT localPort	TCP port to listen on for connections.
IP_MULTI_ADDRESS* localAddress	Local address to use. Use 0 if listening on any network interface.

Function

TCP_SOCKET TCPIP_TCP_ServerOpen(IP_ADDRESS_TYPE addType, TCP_PORT localPort,
IP_MULTI_ADDRESS* localAddress)

TCPIP_TCP_ClientOpen Function

Opens a TCP socket as a client.

File

tcp.h

C

```
TCP_SOCKET TCPIP_TCP_ClientOpen(IP_ADDRESS_TYPE addType, TCP_PORT remotePort, IP_MULTI_ADDRESS*
remoteAddress);
```

Returns

- INVALID_SOCKET - No sockets of the specified type were available to be opened
- TCP_SOCKET handle - Save this handle and use it when calling all other TCP APIs

Description

Provides a unified method for opening TCP client sockets.

Sockets are created at the TCP module initialization, and can be claimed with this function and freed using TCPIP_TCP_Abort or TCPIP_TCP_Close.

Remarks

IP_ADDRESS_TYPE_ANY is not supported (not a valid type for client open)!

If the remoteAddress != 0 (and the address pointed by remoteAddress != 0) then the socket will immediately initiate a connection to the remote host

If the remoteAddress is unspecified, no connection is initiated. Client socket parameters can be set using TCPIP_TCP_Bind, TCPIP_TCP_RemoteBind, etc. calls and then connection initiated by calling TCPIP_TCP_Connect.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
IP_ADDRESS_TYPE addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4.
TCP_PORT remotePort	TCP port to connect to. The local port for client sockets will be automatically picked by the TCP module.
IP_MULTI_ADDRESS* remoteAddress	The remote address to be used. If 0 then the address is unspecified

Function

TCP_SOCKET TCPIP_TCP_ClientOpen(IP_ADDRESS_TYPE addType, TCP_PORT remotePort,
IP_MULTI_ADDRESS* remoteAddress)

TCPIP_TCP_Close Function

Disconnects an open socket and destroys the socket handle, releasing the associated resources.

File

tcp.h

C

```
bool TCPIP_TCP_Close(TCP_SOCKET hTCP);
```

Returns

- true - If the call succeeded
- false - If the call failed (no such socket)

Description

Graceful Option Set: If the graceful option is set for the socket (default), a TCPIP_TCP_Disconnect will be tried. If the linger option is set (default) the TCPIP_TCP_Disconnect will try to send any queued TX data before issuing FIN. If the FIN send operation fails or the socket is not connected the abort is generated.

Graceful Option Not Set: If the graceful option is not set, or the previous step could not send the FIN,a TCPIP_TCP_Abort is called, sending a RST to the remote node. Communication is closed, the socket is no longer valid and the associated resources are freed.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen/TCPIP_TCP_ClientOpen. hTCP - valid socket

Parameters

Parameters	Description
hTCP	Handle to the socket to disconnect and close.

Function

bool TCPIP_TCP_Close(TCP_SOCKET hTCP)

TCPIP_TCP_Connect Function

Connects a client socket.

File

tcp.h

C

```
bool TCPIP_TCP_Connect(TCP_SOCKET hTCP);
```

Returns

- true - If the call succeeded
- false - If the call failed

Description

This function will try to initiate a connection on a client socket that is not connected yet. The client socket should have been created with a call to TCPIP_TCP_ClientOpen having the remoteAddress set to 0.

Remarks

The call will fail if the client socket has no remote host specified. Use TCPIP_TCP_RemoteBind() to specify a remote host address for the client socket.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ClientOpen with an unspecified address. hTCP - valid socket

Parameters

Parameters	Description
hTCP	Handle to the client socket to connect.

Function

bool TCPIP_TCP_Connect(TCP_SOCKET hTCP)

TCPIP_TCP_Bind Function

Binds a socket to a local address.

File

tcp.h

C

```
bool TCPIP_TCP_Bind(TCP_SOCKET hTCP, IP_ADDRESS_TYPE addType, TCP_PORT localPort, IP_MULTI_ADDRESS*
localAddress);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This function is meant for unconnected server and client sockets. It is similar to TCPIP_TCP_SocketNetSet that assigns a specific source interface for a socket. If localPort is 0 the stack will assign a unique local port. Sockets don't need specific binding, it is done automatically by the stack. However, specific binding can be requested using these functions. Works for both client and server sockets.

Remarks

The call will fail if the socket is already connected (both server and client sockets).

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	Socket to bind
addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6.
localPort	Local port to use If 0, the stack will assign a unique local port
localAddress	The local address to bind to. Could be NULL if the local address does not need to be changed

Function

bool TCPIP_TCP_Bind(TCP_SOCKET hTCP, IP_ADDRESS_TYPE addType, TCP_PORT localPort,
IP_MULTI_ADDRESS* localAddress)

TCPIP_TCP_RemoteBind Function

Binds a socket to a remote address.

File

tcp.h

C

```
bool TCPIP_TCP_RemoteBind(TCP_SOCKET hTCP, IP_ADDRESS_TYPE addType, TCP_PORT remotePort, IP_MULTI_ADDRESS*
remoteAddress);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This function is meant for unconnected server and client sockets. Sockets don't need specific remote binding, they should accept connections on any incoming interface. Thus the binding is done automatically by the stack. However, specific remote binding can be requested using these functions. For a server socket it can be used to restrict accepting connections from a specific remote host. For a client socket it will just change the default binding done when the socket was opened.

Remarks

The socket remote port is changed only if a non-zero remotePort value is passed.

The socket remote host address is changed only if a non-zero remoteAddress value is passed.

The call will fail if the socket is already connected (both server and client sockets).

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	Socket to bind
addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6.
remotePort	remote port to use Could be 0 if the remote port does not need to be changed
remoteAddress	The remote address to bind to. Could be NULL if the remote address does not need to be changed

Function

bool TCPIP_TCP_RemoteBind(TCP_SOCKET hTCP, IP_ADDRESS_TYPE addType, TCP_PORT remotePort,
IP_MULTI_ADDRESS* remoteAddress)

TCPIP_TCP_IsConnected Function

Determines if a socket has an established connection.

File

tcp.h

C

```
bool TCPIP_TCP_IsConnected(TCP_SOCKET hTCP);
```

Description

This function determines if a socket has an established connection to a remote node. Call this function after calling TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen() to determine when the connection is set up and ready for use.

Remarks

A socket is said to be connected only if it is in the TCPIP_TCP_STATE_ESTABLISHED state. Sockets in the process of opening or closing will return false.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	The socket to check.

Function

bool TCPIP_TCP_IsConnected(TCP_SOCKET hTCP)

TCPIP_TCP_WasReset Function

Self-clearing semaphore indicating socket reset.

File

tcp.h

C

```
bool TCPIP_TCP_WasReset(TCP_SOCKET hTCP);
```

Description

This function is a self-clearing semaphore indicating whether or not a socket has been disconnected since the previous call. This function works for all possible disconnections: a call to TCPIP_TCP_Disconnect, a FIN from the remote node, or an acknowledgment timeout caused by the loss of a network link. It also returns true after the first call to TCPIP_TCP_Initialize. Applications should use this function to reset their state machines.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Function

bool TCPIP_TCP_WasReset(TCP_SOCKET hTCP)

TCPIP_TCP_Disconnect Function

Disconnects an open socket.

File

tcp.h

C

```
bool TCPIP_TCP_Disconnect(TCP_SOCKET hTCP);
```

Returns

- true - If the call succeeded
- false - Indicates that the notification could not be sent to the remote host. The call can be reissued at a later time if desired.

Description

This function closes the TX side of a connection by sending a FIN (if currently connected) to the remote node of the connection.

If the socket has the linger option set (default), the queued TX data transmission will be attempted before sending the FIN. If the linger option is off, the queued TX data will be discarded.

Please note that this call may fail in which case it can be re-issued.

Remarks

None.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	Handle of the socket to disconnect.

Function

bool TCPIP_TCP_Disconnect(TCP_SOCKET hTCP)

TCPIP_TCP_Abort Function

Aborts a connection.

File

tcp.h

C

```
void TCPIP_TCP_Abort(TCP_SOCKET hTCP, bool killSocket);
```

Returns

None.

Description

This function aborts a connection to a remote node by sending a RST (if currently connected). Any pending TX/RX data is discarded.

A client socket will always be closed and the associated resources released. The socket cannot be used again after this call.

A server socket will abort the current connection:

- if killSocket == false the socket will remain listening
- if killSocket == true the socket will be closed and all associated resources released. The socket cannot be used again after this call.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	Handle to the socket to disconnect.
killSocket	if true, it kills a server socket

Function

void TCPIP_TCP_Abort(TCP_SOCKET hTCP, bool killSocket)

TCPIP_TCP_OptionsGet Function

Allows getting the options for a socket like: current RX/TX buffer size, etc.

File

tcp.h

C

```
bool TCPIP_TCP_OptionsGet(TCP_SOCKET hTCP, TCP_SOCKET_OPTION option, void* optParam);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Various options can be get at the socket level. This function provides compatibility with BSD implementations.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	socket to get options for
option	specific option to get
optParam	pointer to an area that will receive the option value; this is option dependent the size of the area has to be large enough to accommodate the specific option • TCP_OPTION_LINGER - pointer to a TCP_OPTION_LINGER_DATA structure • TCP_OPTION_KEEP_ALIVE - pointer to a TCP_OPTION_KEEP_ALIVE_DATA structure • TCP_OPTION_RX_BUFF - size of the new RX buffer • TCP_OPTION_TX_BUFF - size of the new TX buffer • TCP_OPTION_RX_TMO - not supported yet • TCP_OPTION_TX_TMO - not supported yet • TCP_OPTION_NODELAY - pointer to boolean to return current NO DELAY status • TCP_OPTION_EXCLUSIVE_ADDRESS - not supported yet • TCP_OPTION_THRES_FLUSH - a TCP_OPTION_THRES_FLUSH_TYPE • TCP_OPTION_DELAY_SEND_ALL_ACK - pointer to boolean to return current DELAY Send All ACK status

Function

bool TCPIP_TCP_OptionsGet(TCP_SOCKET hTCP, TCP_SOCKET_OPTION option, void* optParam)

TCPIP_TCP_OptionsSet Function

Allows setting options to a socket like adjust RX/TX buffer size, etc.

File

tcp.h

C

```
bool TCPIP_TCP_OptionsSet(TCP_SOCKET hTCP, TCP_SOCKET_OPTION option, void* optParam);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Various options can be set at the socket level. This function provides compatibility with BSD implementations.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	socket to set options for
option	specific option to be set

optParam	the option value; this is option dependent • TCP_OPTION_LINGER - pointer to a TCP_OPTION_LINGER_DATA structure • TCP_OPTION_KEEP_ALIVE - pointer to a TCP_OPTION_KEEP_ALIVE_DATA structure • TCP_OPTION_RX_BUFF - size of the new RX buffer • TCP_OPTION_TX_BUFF - size of the new TX buffer • TCP_OPTION_RX_TMO - not supported yet • TCP_OPTION_TX_TMO - not supported yet • TCP_OPTION_NODELAY - boolean to enable/disable the NO DELAY functionality • TCP_OPTION_EXCLUSIVE_ADDRESS - not supported yet • TCP_OPTION_THRES_FLUSH - a TCP_OPTION_THRES_FLUSH_TYPE • TCP_OPTION_DELAY_SEND_ALL_ACK - boolean to enable/disable the DELAY Send All ACK data functionality

Function

bool TCPIP_TCP_OptionsSet(TCP_SOCKET hTCP, TCP_SOCKET_OPTION option, void* optParam)

TCPIP_TCP_SocketInfoGet Function

Obtains information about a currently open socket.

File

tcp.h

C

```
bool TCPIP_TCP_SocketInfoGet(TCP_SOCKET hTCP, TCP_SOCKET_INFO* pInfo);
```

Returns

- true - if the call succeeded
- false - if no such socket exists or the socket is not open

Description

Fills the provided TCP_SOCKET_INFO structure associated with this socket. This contains the IP addresses and port numbers for both the local and remote endpoints.

Preconditions

TCP is initialized and the socket is connected.

Parameters

Parameters	Description
hTCP	The socket to check.

Function

bool TCPIP_TCP_SocketInfoGet(TCP_SOCKET hTCP, TCP_SOCKET_INFO* remoteInfo)

TCPIP_TCP_SocketNetGet Function

Gets the current network interface of an TCP socket.

File

tcp.h

C

```
TCPIP_NET_HANDLE TCPIP_TCP_SocketNetGet(TCP_SOCKET hTCP);
```

Returns

The handle of the local interface this socket is bound to.

Description

This function returns the interface handle associated to a TCP socket.

Remarks

The returned handle could be NULL if the socket is invalid or the socket is not currently connected.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ClientOpen()/TCPIP_TCP_ServerOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	The TCP socket

Function

TCPIP_NET_HANDLE TCPIP_TCP_SocketNetGet(TCP_SOCKET hTCP);

TCPIP_TCP_SocketNetSet Function

Sets the interface for an TCP socket

File

tcp.h

C

```
bool TCPIP_TCP_SocketNetSet(TCP_SOCKET hTCP, TCPIP_NET_HANDLE hNet);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This function sets the network interface for an TCP socket

Remarks

An invalid hNet can be passed (0) so that the current network interface selection will be cleared.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ClientOpen()/TCPIP_TCP_ServerOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	The TCP socket
hNet	interface handle.

Function

bool TCPIP_TCP_SocketNetSet(TCP_SOCKET hTCP, TCPIP_NET_HANDLE hNet)

TCPIP_TCP_SignalHandlerDeregister Function

Deregisters a previously registered TCP socket signal handler.

File

tcp.h

C

```
bool TCPIP_TCP_SignalHandlerDeregister(TCP_SOCKET s, TCPIP_TCP_SIGNAL_HANDLE hSig);
```

Returns

- true - if the call succeeds
- false - if no such handler is registered

Description

This function deregisters the TCP socket signal handler.

Preconditions

hSig valid TCP handle.

Parameters

Parameters	Description
s	The TCP socket
hSig	A handle returned by a previous call to TCPIP_TCP_SignalHandlerRegister.

Function

TCPIP_TCP_SignalHandlerDeregister(TCP_SOCKET s, TCPIP_TCP_SIGNAL_HANDLE hSig)

TCPIP_TCP_SignalHandlerRegister Function

Registers a TCP socket signal handler.

File

tcp.h

C

```
TCPIP_TCP_SIGNAL_HANDLE TCPIP_TCP_SignalHandlerRegister(TCP_SOCKET s, TCPIP_TCP_SIGNAL_TYPE sigMask,
TCPIP_TCP_SIGNAL_FUNCTION handler, const void* hParam);
```

Returns

Returns a valid handle if the call succeeds, or a null handle if the call failed (null handler, no such socket, existent handler).

Description

This function registers a TCP socket signal handler. The TCP module will call the registered handler when a TCP signal (TCPIP_TCP_SIGNAL_TYPE) occurs.

Remarks

Only one signal handler per socket is supported. A new handler does not override the existent one. Instead TCPIP_TCP_SignalHandlerDeregister has to be explicitly called.

The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

The hParam is passed by the client but is currently not used and should be 0.

For multi-threaded systems the TCP/IP packet dispatch does not occur on the user thread. The signal handler will be called on a different thread context. It is essential that this handler is non blocking and really fast.

For multi-threaded systems, once set, it is not recommended to change the signal handler at run time. Synchronization between user threads and packet dispatch threads could be difficult. If really need to be changed, make sure that the old handler could still be called and it should be valid until the new one is taken into account. TCPIP_TCP_SignalHandlerDeregister needs to be called before registering another handler.

Preconditions

TCP valid socket.

Parameters

Parameters	Description
s	The TCP socket
sigMask	mask of signals to be reported
handler	signal handler to be called when a TCP event occurs.
hParam	Parameter to be used in the handler call. This is user supplied and is not used by the TCP module. Currently not used and it should be null.

Function

TCPIP_TCP_SignalHandlerRegister(TCP_SOCKET s, TCPIP_TCP_SIGNAL_TYPE sigMask,
TCPIP_TCP_SIGNAL_FUNCTION handler, const void* hParam)

TCPIP_TCP_Task Function

Standard TCP/IP stack module task function.

File

tcp.h

C

```
void TCPIP_TCP_Task();
```

Returns

None.

Description

This function performs TCP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The TCP module should have been initialized.

Function

void TCPIP_TCP_Task(void)

TCPIP_TCP_IsReady Function

Returns the current status of the TCP module

File

tcp.h

C

```
bool TCPIP_TCP_IsReady();
```

Returns

true - the quiet time has elapsed and the TCP module is ready to transmit data false - the quiet time is still in effect and TCP module is silent

Description

This function returns the current status of the TCP module. If the TCP quiet time has elapsed (or it was not enabled when TCP was built) then the TCP module is ready for operation and TCP sockets can send and receive data. If the TCP quiet time is still in effect, then the TCP module is quiet and it won't transmit any kind of data.

Remarks

The TCP quiet time occurs at the stack start up and can be enabled/disabled using the TCP module configuration parameters.

Preconditions

TCP module properly initialized

Function

bool   TCPIP_TCP_IsReady(void);

TCPIP_TCP_SocketsNumberGet Function

Returns the number of existent TCP sockets.

File

tcp.h

C

```
int TCPIP_TCP_SocketsNumberGet();
```

Returns

The number of TCP sockets

Description

This function returns the number of created TCP sockets. This is the maximum number of sockets that can be opened at any moment as it's been passed as parameter when TCP module was created.

Preconditions

TCP module properly initialized

Function

int TCPIP_TCP_SocketsNumberGet(void)

TCPIP_TCP_SocketTraceSet Function

Sets the current socket trace status.

File

tcp.h

C

```
bool TCPIP_TCP_SocketTraceSet(TCP_SOCKET sktNo, bool enable);
```

Returns

- true if the call succeeded
- false if there was an error (no such socket, tracing not enabled, etc.)

Description

This function enables or disables the trace status of the specified socket. The trace functionality needs to be enabled in the TCP module for this function to succeed. Currently when socket trace is enabled, the current socket state transitions are displayed at the system console.

Preconditions

TCP module properly initialized

Parameters

Parameters	Description
sktNo	socket to enable/disable trace
enable	boolean to enable/disable socket trace

Function

bool TCPIP_TCP_SocketTraceSet(TCP_SOCKET sktNo, bool enable)

b) Transmit Data Transfer Functions

TCPIP_TCP_Put Function

File

tcp.h

C

```
uint16_t TCPIP_TCP_Put(TCP_SOCKET hTCP, uint8_t byte);
```

Description

Writes a single byte to a TCP socket.

Remarks

Note that this function is inefficient and its use is discouraged. A buffered approach (TCPIP_TCP_ArrayPut) is preferred.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	The socket to which data is to be written.
byte	The byte to write.

Function

uint16_t TCPIP_TCP_Put(TCP_SOCKET hTCP, uint8_t byte)

TCPIP_TCP_PutIsReady Function

Determines how much free space is available in the TCP TX buffer.

File

tcp.h

C

```
uint16_t TCPIP_TCP_PutIsReady(TCP_SOCKET hTCP);
```

Returns

The number of bytes available to be written in the TCP TX buffer.

Description

Call this function to determine how many bytes can be written to the TCP TX buffer. If this function returns zero, the application must return to the main stack loop before continuing in order to transmit more data.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Function

uint16_t TCPIP_TCP_PutIsReady(TCP_SOCKET hTCP)

TCPIP_TCP_StringPut Function

Writes a null-terminated string to a TCP socket.

File

tcp.h

C

```
const uint8_t* TCPIP_TCP_StringPut(TCP_SOCKET hTCP, const uint8_t* Data);
```

Returns

Pointer to the byte following the last byte written to the socket. If this pointer does not dereference to a NULL byte, the buffer became full or the socket is not connected.

Description

This function writes a null-terminated string to a TCP socket. The null-terminator is not copied to the socket.

Remarks

The return value of this function differs from that of TCPIP_TCP_ArrayPut. To write long strings in a single state, initialize the *data pointer to the first byte, then call this function repeatedly (breaking to the main stack loop after each call) until the return value dereferences to a NULL byte. Save the return value as the new starting *data pointer otherwise.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to which data is to be written.
data	Pointer to the string to be written.

Function

const uint8_t* TCPIP_TCP_StringPut(TCP_SOCKET hTCP, const uint8_t* data)

TCPIP_TCP_ArrayPut Function

File

tcp.h

C

```
uint16_t TCPIP_TCP_ArrayPut(TCP_SOCKET hTCP, const uint8_t* Data, uint16_t Len);
```

Returns

The number of bytes written to the socket. If less than len, the buffer became full or the socket is not connected.

Description

Writes an array from a buffer to a TCP socket.

Remarks

This operation can cause a TCP packet to be transmitted over the network (i.e., a TCPIP_TCP_Flush operation to be performed) when there is data in the TCP TX buffer that can be sent and any of the following occurs:

- There is no more space available in the TCP buffer.
- The TCP_OPTION_THRES_FLUSH_ON is set and the TX buffer is at least half full
- The amount of data that can be sent is bigger than the remote host MSS or than half of the maximum advertised window size.
- The Nagle algorithm is disabled and there is no unacknowledged data.

If none of these occur and the socket user does not add data to the TX socket buffer, the TCP state machine will automatically flush the buffer when the TCP_AUTO_TRANSMIT_TIMEOUT_VAL time-out elapsed.

The default TCP_AUTO_TRANSMIT_TIMEOUT_VAL is 40 ms.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to which data is to be written.
data	Pointer to the array to be written.
len	Number of bytes to be written.

Function

uint16_t TCPIP_TCP_ArrayPut(TCP_SOCKET hTCP, const uint8_t* data, uint16_t len)

TCPIP_TCP_Flush Function

Immediately transmits all pending TX data.

File

tcp.h

C

```
bool TCPIP_TCP_Flush(TCP_SOCKET hTCP);
```

Returns

None,

Description

This function immediately transmits all pending TX data with a PSH flag. If this function is not called, data will automatically be sent by the TCP state machine (see the TCPIP_TCP_ArrayPut() description/notes).

Remarks

The application should not call this function explicitly because doing this will interfere with the TCP algorithm and degrade performance of the socket data transfer. One exception is when the application knows that it put all the data it needed into the TCP buffer and it makes sense to flush the socket instead of waiting TCP_AUTO_TRANSMIT_TIMEOUT_VAL timeout to elapse.

Preconditions

TCP is initialized and the socket is connected.

Parameters

Parameters	Description
hTCP	The socket whose data is to be transmitted.

Function

bool TCPIP_TCP_Flush(TCP_SOCKET hTCP)

TCPIP_TCP_FifoTxFullGet Function

Determines how many bytes are pending in the TCP TX FIFO.

File

tcp.h

C

```
uint16_t TCPIP_TCP_FifoTxFullGet(TCP_SOCKET hTCP);
```

Returns

Number of bytes pending to be flushed in the TCP TX FIFO.

Description

This function determines how many bytes are pending in the TCP TX FIFO.

Remarks

None.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Function

uint16_t TCPIP_TCP_FifoTxFullGet(TCP_SOCKET hTCP)

TCPIP_TCP_FifoTxFreeGet Function

Determines how many bytes are free and could be written in the TCP TX FIFO.

File

tcp.h

C

```
uint16_t TCPIP_TCP_FifoTxFreeGet(TCP_SOCKET hTCP);
```

Returns

The number of bytes available to be written in the TCP TX buffer.

Description

Macro: TCPIP_TCP_FifoTxFreeGet(TCP_SOCKET hTCP)

This macro returns the number of bytes that are free in the socket TX buffer.

Remarks

None.

Parameters

Parameters	Description
hTCP	The socket to check.

c) Receive Data Transfer Functions

TCPIP_TCP_GetIsReady Function

Determines how many bytes can be read from the TCP RX buffer.

File

tcp.h

C

```
uint16_t TCPIP_TCP_GetIsReady(TCP_SOCKET hTCP);
```

Returns

The number of bytes available to be read from the TCP RX buffer.

Description

Call this function to determine how many bytes can be read from the TCP RX buffer. If this function returns zero, the application must return to the main stack loop before continuing in order to wait for more data to arrive.

Remarks

None.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Function

uint16_t TCPIP_TCP_GetIsReady(TCP_SOCKET hTCP)

TCPIP_TCP_ArrayGet Function

Reads an array of data bytes from a TCP socket's RX buffer/FIFO.

File

tcp.h

C

```
uint16_t TCPIP_TCP_ArrayGet(TCP_SOCKET hTCP, uint8_t* buffer, uint16_t count);
```

Returns

The number of bytes read from the socket. If less than len, the RX FIFO buffer became empty or the socket is not connected.

Description

This function reads an array of data bytes from a TCP socket's RX buffer/FIFO. The data is removed from the FIFO in the process.

Remarks

If the supplied buffer is null, the data is simply discarded.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket from which data is to be read.
buffer	Pointer to the array to store data that was read.
len	Number of bytes to be read.

Function

uint16_t TCPIP_TCP_ArrayGet(TCP_SOCKET hTCP, uint8_t* buffer, uint16_t len)

TCPIP_TCP_ArrayPeek Function

Reads a specified number of data bytes from the TCP RX buffer/FIFO without removing them from the buffer.

File

tcp.h

C

```
uint16_t TCPIP_TCP_ArrayPeek(TCP_SOCKET hTCP, uint8_t * vBuffer, uint16_t wLen, uint16_t wStart);
```

Description

This function reads a specified number of data bytes from the TCP RX FIFO without removing them from the buffer. No TCP control actions are taken as a result of this function (ex: no window update is sent to the remote node).

Remarks

None.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to peek from (read without removing from stream).
vBuffer	Destination to write the peeked data bytes.
wLen	Length of bytes to peek from the RX FIFO and copy to vBuffer.
wStart	Zero-indexed starting position within the FIFO from which to start peeking.

Function

uint16_t TCPIP_TCP_ArrayPeek(TCP_SOCKET hTCP, uint8_t *vBuffer, uint16_t wLen, uint16_t wStart)

TCPIP_TCP_ArrayFind Function

Searches for a string in the TCP RX buffer.

File

tcp.h

C

```
uint16_t TCPIP_TCP_ArrayFind(TCP_SOCKET hTCP, const uint8_t* cFindArray, uint16_t wLen, uint16_t wStart,
uint16_t wSearchLen, bool bTextCompare);
```

Description

This function finds the first occurrence of an array of bytes in the TCP RX buffer. It can be used by an application to abstract searches out of their own application code. For increased efficiency, the function is capable of limiting the scope of search to a specific range of bytes. It can also perform a case-insensitive search if required.

For example, if the buffer contains "I love PIC MCUs!" and the search array is "love" with a length of 4, a value of 2 will be returned.

Remarks

For better performance of this function, try to search for characters that are expected to exist or limit the scope of the search as much as possible. The HTTP module, for example, uses this function to parse headers. However, it searches for newlines, then the separating colon, then reads the header name to an internal buffer for comparison. This has proven to be significantly faster than searching for full header name strings outright.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to search within.
cFindArray	The array of bytes to find in the buffer.
wLen	Length of cFindArray.
wStart	Zero-indexed starting position within the buffer.
wSearchLen	Length from wStart to search in the buffer.
bTextCompare	true for case-insensitive text search, false for binary search

Function

uint16_t TCPIP_TCP_ArrayFind(TCP_SOCKET hTCP, uint8_t* cFindArray, uint16_t wLen,

uint16_t wStart, uint16_t wSearchLen, bool bTextCompare)

TCPIP_TCP_Find Function

Searches for a byte in the TCP RX buffer.

File

tcp.h

C

```
uint16_t TCPIP_TCP_Find(TCP_SOCKET hTCP, uint8_t cFind, uint16_t wStart, uint16_t wSearchLen, bool bTextCompare);
```

Description

This function finds the first occurrence of a byte in the TCP RX buffer. It can be used by an application to abstract searches out of their own application code. For increased efficiency, the function is capable of limiting the scope of search to a specific range of bytes. It can also perform a case-insensitive search if required.

For example, if the buffer contains "I love PIC MCUs!" and the cFind byte is ' ', a value of 1 will be returned.

Remarks

For better performance of this function, try to search for characters that are expected to exist or limit the scope of the search as much as possible. The HTTP module, for example, uses this function to parse headers. However, it searches for newlines, then the separating colon, then reads the header name to an internal buffer for comparison. This has proven to be significantly faster than searching for full header name strings outright.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to search within.
cFind	The byte to find in the buffer.
wStart	Zero-indexed starting position within the buffer.
wSearchLen	Length from wStart to search in the buffer.
bTextCompare	true for case-insensitive text search, false for binary search

Function

uint16_t TCPIP_TCP_Find(TCP_SOCKET hTCP, uint8_t cFind, uint16_t wStart,

uint16_t wSearchLen, bool bTextCompare)

TCPIP_TCP_Get Function

Retrieves a single byte to a TCP socket.

File

tcp.h

C

```
uint16_t TCPIP_TCP_Get(TCP_SOCKET hTCP, uint8_t* byte);
```

Description

This function retrieves a single byte to a TCP socket.

Remarks

Note that this function is inefficient. A buffered approach (TCPIP_TCP_ArrayGet) is preferred.

Preconditions

TCP socket should have been opened with TCPIP_TCP_ServerOpen()/TCPIP_TCP_ClientOpen(). hTCP - valid socket

Parameters

Parameters	Description
hTCP	The socket from which to read.
byte	Pointer to location in which the read byte should be stored.

Function

uint16_t TCPIP_TCP_Get(TCP_SOCKET hTCP, uint8_t* byte)

TCPIP_TCP_Peek Function

Peaks at one byte in the TCP RX buffer/FIFO without removing it from the buffer.

File

tcp.h

C

```
uint8_t TCPIP_TCP_Peek(TCP_SOCKET hTCP, uint16_t wStart);
```

Description

This function peaks at one byte in the TCP RX buffer/FIFO without removing it from the buffer.

Remarks

Note that this function is inefficient and its use is discouraged. A buffered approach (TCPIP_TCP_ArrayPeek) is preferred.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to peek from (read without removing from stream).
wStart	Zero-indexed starting position within the FIFO to peek from.

Function

uint8_t TCPIP_TCP_Peek(TCP_SOCKET hTCP, uint16_t wStart)

TCPIP_TCP_Discard Function

Discards any pending data in the RCP RX FIFO.

File

tcp.h

C

```
uint16_t TCPIP_TCP_Discard(TCP_SOCKET hTCP);
```

Returns

The number of bytes that have been discarded from the RX buffer.

Description

This function discards any pending data in the TCP RX FIFO.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket whose RX buffer/FIFO is to be cleared.

Function

uint16_t TCPIP_TCP_Discard(TCP_SOCKET hTCP)

TCPIP_TCP_FifoRxFreeGet Function

Determines how many bytes are free in the RX buffer/FIFO.

File

tcp.h

C

```
uint16_t TCPIP_TCP_FifoRxFreeGet(TCP_SOCKET hTCP);
```

Returns

The number of bytes free in the TCP RX FIFO. If zero, no additional data can be received until the application removes some data using one of the TCPIP_TCP_Get family functions.

Description

This function determines how many bytes are free in the RX buffer/FIFO.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to check.

Function

uint16_t TCPIP_TCP_FifoRxFreeGet(TCP_SOCKET hTCP)

TCPIP_TCP_FifoSizeAdjust Function

Adjusts the relative sizes of the RX and TX buffers.

File

tcp.h

C

```
bool TCPIP_TCP_FifoSizeAdjust(TCP_SOCKET hTCP, uint16_t wMinRXSize, uint16_t wMinTXSize, TCP_ADJUST_FLAGS vFlags);
```

Description

This function can be used to simultaneously adjust the sizes of the RX and TX FIFOs.

Adjusting the size of the TX/RX FIFO on the fly can allow for optimal transmission speed for one-sided application protocols. For example, HTTP typically begins by receiving large amounts of data from the client, then switches to serving large amounts of data back. Adjusting the FIFO at

these points can increase performance in systems that have limited resources. Once the FIFOs are adjusted, a window update is sent.

Although the TX and RX socket buffers are completely independent, for the purpose of this function they can be considered together if neither TCP_ADJUST_TX_ONLY nor TCP_ADJUST_RX_ONLY flags are set.

In these conditions the TCP_ADJUST_FLAGS control the distribution of the available space between the TX and RX FIFOs. If neither or both of TCP_ADJUST_GIVE_REST_TO_TX and TCP_ADJUST_GIVE_REST_TO_RX are set, the function distributes the remaining space (if any) equally. If the new requested FIFOs space is greater that the old existing FIFOs space the TCP_ADJUST_GIVE_REST_TO_TX and TCP_ADJUST_GIVE_REST_TO_RX are ignored.

TCP_ADJUST_PRESERVE_RX and TCP_ADJUST_PRESERVE_TX request the preserving of the existing data. Existing data can be preserved as long as the old data in the buffer does not exceed the capacity of the new buffer.

Remarks

The function will automatically maintain minimal values for both TX and RX buffers.

To avoid having a socket with no associated buffers, the function first allocates the new buffers and, if succeeded, it frees the old ones.

The TX and RX FIFOs (buffers) associated with a socket are completely separate and independent. However, when TCP_ADJUST_TX_ONLY or TCP_ADJUST_RX_ONLY are not set, for the purpose of this function, the TX and RX FIFOs are considered to be contiguous so that the total FIFO space is divided between the TX and RX FIFOs. This provides backward compatibility with previous versions of this function.

The new flags TCP_ADJUST_TX_ONLY and TCP_ADJUST_RX_ONLY allow changing the size of TX and RX buffers independently. This is the preferred option.

TX or RX associated buffer sizes can be changed too using the socket options. See TCPIP_TCP_OptionsSet.

Preconditions

TCP is initialized.

Parameters

Parameters	Description
hTCP	The socket to be adjusted
wMinRXSize	Minimum number of bytes for the RX FIFO
wMinTXSize	Minimum number of bytes for the TX FIFO
vFlags	If TCP_ADJUST_TX_ONLY or TCP_ADJUST_RX_ONLY are not set, the TX and RX buffers are evaluated together and any
combination of the following flags is valid	TCP_ADJUST_GIVE_REST_TO_RX, TCP_ADJUST_GIVE_REST_TO_TX TCP_ADJUST_PRESERVE_RX, TCP_ADJUST_PRESERVE_TX If TCP_ADJUST_TX_ONLY or TCP_ADJUST_RX_ONLY is set TX and RX buffers are treated individually and TCP_ADJUST_GIVE_REST_TO_TX, TCP_ADJUST_GIVE_REST_TO_RX values are irrelevant. TCP_ADJUST_TX_ONLY and TCP_ADJUST_RX_ONLY both set is invalid.

Function

bool TCPIP_TCP_FifoSizeAdjust(TCP_SOCKET hTCP, uint16_t wMinRXSize, uint16_t wMinTXSize,
TCP_ADJUST_FLAGS vFlags)

TCPIP_TCP_FifoRxFullGet Macro

Determines how many bytes are pending in the RX buffer/FIFO.

File

tcp.h

C

```
#define TCPIP_TCP_FifoRxFullGet(a) TCPIP_TCP_GetIsReady(a)
```

Description

Macro: TCPIP_TCP_FifoRxFullGet

Alias to TCPIP_TCP_GetIsReady provided for API completeness

Remarks

None.

d) Data Types and Constants

TCP_ADJUST_FLAGS Enumeration

TCP adjust RX and TX buffers flags.

File

tcp.h

C

```
typedef enum {
  TCP_ADJUST_GIVE_REST_TO_RX = 0x01,
  TCP_ADJUST_GIVE_REST_TO_TX = 0x02,
  TCP_ADJUST_PRESERVE_RX = 0x04,
  TCP_ADJUST_PRESERVE_TX = 0x08,
  TCP_ADJUST_TX_ONLY = 0x10,
  TCP_ADJUST_RX_ONLY = 0x20
} TCP_ADJUST_FLAGS;
```

Members

Members	Description
TCP_ADJUST_GIVE_REST_TO_RX = 0x01	Resize flag: extra bytes go to RX
TCP_ADJUST_GIVE_REST_TO_TX = 0x02	Resize flag: extra bytes go to TX
TCP_ADJUST_PRESERVE_RX = 0x04	Resize flag: attempt to preserve RX buffer
TCP_ADJUST_PRESERVE_TX = 0x08	Resize flag: attempt to preserve TX buffer
TCP_ADJUST_TX_ONLY = 0x10	Resize flag: adjust the TX buffer only
TCP_ADJUST_RX_ONLY = 0x20	Resize flag: adjust the RX buffer only

Description

Enumeration: TCP_ADJUST_FLAGS

Adjusts socket RX and TX buffer sizes.

TCP_OPTION_LINGER_DATA Structure

Socket linger options.

File

tcp.h

C

```
typedef struct {
  bool lingerEnable;
  bool gracefulEnable;
  uint16_t lingerTmo;
} TCP_OPTION_LINGER_DATA;
```

Members

Members	Description
bool lingerEnable;	Enable/disable linger; enabled by default for any socket.
bool gracefulEnable;	Enable/disable graceful close; enabled by default for any socket.
uint16_t lingerTmo;	Linger timeout in seconds (when enabled). This option is not supported yet.

Description

Structure: TCP_OPTION_LINGER_DATA

This structure defines socket linger options.

TCP_PORT Type

Defines a TCP Port number.

File

tcp.h

C

```
typedef uint16_t TCP_PORT;
```

Description

Type: TCP_PORT

TCP Port Number identifier.

TCP_SOCKET Type

Defines a TCP Socket.

File

tcp.h

C

```
typedef int16_t TCP_SOCKET;
```

Description

Type: TCP_SOCKET

A TCP_SOCKET is stored as a single int16_t number.

TCP_SOCKET_INFO Structure

TCP socket information.

File

tcp.h

C

```
typedef struct {
  IP_ADDRESS_TYPE addressType;
  IP_MULTI_ADDRESS remoteIPaddress;
  IP_MULTI_ADDRESS localIPaddress;
  TCP_PORT remotePort;
  TCP_PORT localPort;
  TCPIP_NET_HANDLE hNet;
  TCPIP_TCP_STATE state;
  uint16_t rxSize;
  uint16_t txSize;
  uint16_t rxPending;
  uint16_t txPending;
} TCP_SOCKET_INFO;
```

Members

Members	Description
IP_ADDRESS_TYPE addressType;	Address type of the socket IPv4 or IPv6
IP_MULTI_ADDRESS remoteIPaddress;	Remote address to which the socket is connected
IP_MULTI_ADDRESS localIPaddress;	Local address socket is bound to
TCP_PORT remotePort;	Port number associated with remote node
TCP_PORT localPort;	Local port number
TCPIP_NET_HANDLE hNet;	Associated interface
TCPIP_TCP_STATE state;	Current socket state
uint16_t rxSize;	size of the RX buffer
uint16_t txSize;	size of the TX buffer
uint16_t rxPending;	bytes pending in RX buffer
uint16_t txPending;	bytes pending in TX buffer

Description

Structure: TCP_SOCKET_INFO

Gets information about a socket.

TCP_SOCKET_OPTION Enumeration

TCP Socket run-time options.

File

tcp.h

C

```
typedef enum {
  TCP_OPTION_LINGER,
  TCP_OPTION_KEEP_ALIVE,
  TCP_OPTION_RX_BUFF,
  TCP_OPTION_TX_BUFF,
  TCP_OPTION_RX_TMO,
  TCP_OPTION_TX_TMO,
  TCP_OPTION_NODELAY,
  TCP_OPTION_EXCLUSIVE_ADDRESS,
  TCP_OPTION_THRES_FLUSH,
  TCP_OPTION_DELAY_SEND_ALL_ACK
} TCP_SOCKET_OPTION;
```

Members

Members	Description
TCP_OPTION_LINGER	The LINGER option controls the action taken when unsent data is queued on a socket and the socket is closed. The linger option can be turned on/off and the timeout can be specified.
TCP_OPTION_KEEP_ALIVE	Enable the use of keep-alive packets on TCP connections. The option can be turned on/off and the timeout can be specified.
TCP_OPTION_RX_BUFF	Request different RX buffer size. Has to call TCPIP_TCP_OptionsGet to see the exact space allocated.
TCP_OPTION_TX_BUFF	Request different TX buffer size. Has to call TCPIP_TCP_OptionsGet to see the exact space allocated.
TCP_OPTION_RX_TMO	Specifies the RX timeout. If no data arrives in the specified timeout the socket is closed.
TCP_OPTION_TX_TMO	Specifies the TX timeout. If no data can be sent in the specified timeout the socket is closed.
TCP_OPTION_NODELAY	Enables the NO DELAY/Nagle algorithm functionality. The default setting is disabled.
TCP_OPTION_EXCLUSIVE_ADDRESS	Enables a socket to be bound for exclusive access. The default setting is disabled - option not supported yet.
TCP_OPTION_THRES_FLUSH	Sets the type of half buffer TX flush for the socket. The default setting is TCP_OPTION_THRES_FLUSH_AUTO
TCP_OPTION_DELAY_SEND_ALL_ACK	Enables/disables the delay of sending the pending data when no unacknowledged data. When this option is disabled, the socket will immediately send any pending data chunk whenever all the previous data is acknowledged by the remote party (there is no unacknowledged data). This holds true for the 1st transmission too, or after a pause of some length. If the option is enabled, the socket will delay sending the data, waiting for more data to accumulate. This could be useful when the application makes multiple calls with small data chunks. The default setting is disabled.

Description

Enumeration: TCP_SOCKET_OPTION

This enumeration defines TCP socket run-time options.

INVALID_SOCKET Macro

Invalid socket indicator macro.

File

tcp.h

C

```
#define INVALID_SOCKET (-1)
```

Description

Macro: INVALID_SOCKET

Indicates that the socket is invalid or could not be opened.

TCP_OPTION_THRES_FLUSH_TYPE Enumeration

List of the socket half threshold TX flush types.

File

tcp.h

C

```
typedef enum {
  TCP_OPTION_THRES_FLUSH_AUTO,
  TCP_OPTION_THRES_FLUSH_OFF,
  TCP_OPTION_THRES_FLUSH_ON
} TCP_OPTION_THRES_FLUSH_TYPE;
```

Members

Members	Description
TCP_OPTION_THRES_FLUSH_AUTO	The socket will set the half buffer flush based on the TX buffer size. This is the default setting. For TX buffers >= 1.5 MSS the half buffer flush will be disabled. This results in higher performance/throughput for the socket.
TCP_OPTION_THRES_FLUSH_OFF	Always disable the socket flush at half buffer threshold. No socket flush is performed when the half TX buffer threshold is passed.
TCP_OPTION_THRES_FLUSH_ON	Always enable the socket flush at half buffer threshold. Socket flush is always performed when the half TX buffer threshold is passed. This is useful for small TX buffers when the remote party implements the delayed ACK algorithm.

Description

Enumeration: TCP_OPTION_THRES_FLUSH_TYPE

Describes the possible types of the socket half threshold TX flush.

TCPIP_TCP_MODULE_CONFIG Structure

TCP module run-time configuration/initialization data.

File

tcp.h

C

```
typedef struct {
  int nSockets;
  uint16_t sktTxBuffSize;
  uint16_t sktRxBuffSize;
} TCPIP_TCP_MODULE_CONFIG;
```

Members

Members	Description
int nSockets;	Number of sockets to be created
uint16_t sktTxBuffSize;	Size of the socket TX buffer
uint16_t sktRxBuffSize;	Size of the socket RX buffer

Description

Structure: TCPIP_TCP_MODULE_CONFIG

This structure defines TCP module run-time configuration/initialization data.

TCP_OPTION_KEEP_ALIVE_DATA Structure

Socket keep alive options

File

tcp.h

C

```
typedef struct {
```

```
  bool keepAliveEnable;
  uint16_t keepAliveTmo;
  uint8_t keepAliveUnackLim;
} TCP_OPTION_KEEP_ALIVE_DATA;
```

Members

Members	Description
bool keepAliveEnable;	Enable/disable keep alive option; disabled by default for any socket.
uint16_t keepAliveTmo;	keep alive timeout in milliseconds ignored when keep alive is disabled if 0, the default build time value is used
uint8_t keepAliveUnackLim;	limit of keep alives to be sent the socket will reset the communication channel if no reply received after so many retries ignored when keep alive is disabled if 0, the default build time value is used

Description

Structure: TCP_OPTION_KEEP_ALIVE_DATA

This structure defines socket keep alive options.

TCPIP_TCP_SIGNAL_FUNCTION Type

TCP signal handler.

File

tcp.h

C

```
typedef void (* TCPIP_TCP_SIGNAL_FUNCTION)(TCP_SOCKET hTCP, TCPIP_NET_HANDLE hNet, TCPIP_TCP_SIGNAL_TYPE
sigType, const void* param);
```

Description

Type: TCPIP_TCP_SIGNAL_FUNCTION

Prototype of a TCP signal handler. Socket user can register a handler for the TCP socket. Once an TCP event occurs the registered handler will be called.

Remarks

The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

Parameters

Parameters	Description
hTCP	TCP socket to be used
hNet	the network interface on which the event has occurred
sigType	type of TCP signal that has occurred
param	additional parameter that can has been specified at the handler registration call is currently not used and will be null

TCPIP_TCP_SIGNAL_HANDLE Type

TCP socket handle.

File

tcp.h

C

```
typedef const void* TCPIP_TCP_SIGNAL_HANDLE;
```

Description

Type: TCPIP_TCP_SIGNAL_HANDLE

A handle that a socket client can use after the signal handler has been registered.

TCPIP_TCP_SIGNAL_TYPE Enumeration

TCP run time signal/event types.

File

tcp.h

C

```
typedef enum {
  TCPIP_TCP_SIGNAL_TX_DONE = 0x0001,
  TCPIP_TCP_SIGNAL_TX_DATA_DONE = 0x0002,
  TCPIP_TCP_SIGNAL_TX_SPACE = 0x0004,
  TCPIP_TCP_SIGNAL_ESTABLISHED = 0x0100,
  TCPIP_TCP_SIGNAL_RX_DATA = 0x0200,
  TCPIP_TCP_SIGNAL_RX_FIN = 0x0400,
  TCPIP_TCP_SIGNAL_RX_RST = 0x0800,
  TCPIP_TCP_SIGNAL_KEEP_ALIVE_TMO = 0x1000
} TCPIP_TCP_SIGNAL_TYPE;
```

Members

Members	Description
TCPIP_TCP_SIGNAL_TX_DONE = 0x0001	A TCP packet was successfully transmitted on the interface. This may indicate or not that new data can be sent with this socket. Note 1: The TCP buffer space is controlled by the TCP state machine; The fact that a packet was transmitted, doesn't necessarily mean that new data can be written into the socket buffer: 2: The packet may be an ACK only packet, retry packet, etc., and does not free any writing space in the socket buffer 3: A new socket writing space may not be available because the other party has not acknowledged the data it was sent so far. Therefore, the information carried by this signal has limited usage. 4: This notification is the result of an user action (explicit call to TCPIP_TCP_Flush) or as a result of the internal TCP state machine
TCPIP_TCP_SIGNAL_TX_DATA_DONE = 0x0002	A TCP data packet carrying valid payload was successfully transmitted on the interface. This may indicate or not that new data can be sent with this socket.
TCPIP_TCP_SIGNAL_TX_SPACE = 0x0004	New TX space is available in the socket TX buffer. This event signals that the remote party has acknowledged some data and, as a result, TX buffer space is available.
TCPIP_TCP_SIGNAL_ESTABLISHED = 0x0100	Socket established a connection (client or server socket)
TCPIP_TCP_SIGNAL_RX_DATA = 0x0200	A data packet was successfully received and there is data available for this socket
TCPIP_TCP_SIGNAL_RX_FIN = 0x0400	Remote host finished it's data and sent a FIN;
TCPIP_TCP_SIGNAL_RX_RST = 0x0800	Remote host reset the connection;
TCPIP_TCP_SIGNAL_KEEP_ALIVE_TMO = 0x1000	Keep alive has timed out; Connection to the remote host has been aborted;

Description

Enumeration: TCPIP_TCP_SIGNAL_TYPE

Description of the signals/events that a TCP socket can generate.

Remarks

These signals are used in the socket event handling notification functions. It is possible that multiple flags are set as part of the same notification.

The signals are 16 bits wide.

TCPIP_TCP_STATE Enumeration

TCP socket state.

File

tcp.h

C

```
typedef enum {
  TCPIP_TCP_STATE_LISTEN,
  TCPIP_TCP_STATE_SYN_SENT,
  TCPIP_TCP_STATE_SYN_RECEIVED,
  TCPIP_TCP_STATE_ESTABLISHED,
  TCPIP_TCP_STATE_FIN_WAIT_1,
  TCPIP_TCP_STATE_FIN_WAIT_2,
```

```
    TCPIP_TCP_STATE_CLOSING,
    TCPIP_TCP_STATE_TIME_WAIT,
    TCPIP_TCP_STATE_CLOSE_WAIT,
    TCPIP_TCP_STATE_LAST_ACK,
    TCPIP_TCP_STATE_CLIENT_WAIT_DISCONNECT,
    TCPIP_TCP_STATE_CLIENT_WAIT_CONNECT,
    TCPIP_TCP_STATE_KILLED
} TCPIP_TCP_STATE;
```

Members

Members	Description
TCPIP_TCP_STATE_LISTEN	Socket is listening for connections
TCPIP_TCP_STATE_SYN_SENT	A SYN has been sent, awaiting an SYN+ACK
TCPIP_TCP_STATE_SYN_RECEIVED	A SYN has been received, awaiting an ACK
TCPIP_TCP_STATE_ESTABLISHED	Socket is connected and connection is established
TCPIP_TCP_STATE_FIN_WAIT_1	FIN WAIT state 1
TCPIP_TCP_STATE_FIN_WAIT_2	FIN WAIT state 2
TCPIP_TCP_STATE_CLOSING	Socket is closing
TCPIP_TCP_STATE_TIME_WAIT	2MSL state
TCPIP_TCP_STATE_CLOSE_WAIT	Waiting to close the socket
TCPIP_TCP_STATE_LAST_ACK	The final ACK has been sent
TCPIP_TCP_STATE_CLIENT_WAIT_DISCONNECT	client socket lost connection, waiting for app close/disconnect
TCPIP_TCP_STATE_CLIENT_WAIT_CONNECT	client socket waiting for connection
TCPIP_TCP_STATE_KILLED	socket is killed; debug reporting state

Description

Enumeration: TCPIP_TCP_STATE

Enumeration describing the possible states of a TCP socket.

Files

Files

Name	Description
tcp.h	Transmission Control Protocol (TCP) Communications Layer API
tcp_config.h	TCP configuration file

Description

This section lists the source and header files used by the library.

tcp.h

Transmission Control Protocol (TCP) Communications Layer API

Enumerations

	Name	Description
	TCP_ADJUST_FLAGS	TCP adjust RX and TX buffers flags.
	TCP_OPTION_THRES_FLUSH_TYPE	List of the socket half threshold TX flush types.
	TCP_SOCKET_OPTION	TCP Socket run-time options.
	TCPIP_TCP_SIGNAL_TYPE	TCP run time signal/event types.
	TCPIP_TCP_STATE	TCP socket state.

Functions

	Name	Description
	TCPIP_TCP_Abort	Aborts a connection.
	TCPIP_TCP_ArrayFind	Searches for a string in the TCP RX buffer.
	TCPIP_TCP_ArrayGet	Reads an array of data bytes from a TCP socket's RX buffer/FIFO.
	TCPIP_TCP_ArrayPeek	Reads a specified number of data bytes from the TCP RX buffer/FIFO without removing them from the buffer.

	TCPIP_TCP_ArrayPut	Writes an array from a buffer to a TCP socket.
	TCPIP_TCP_Bind	Binds a socket to a local address.
	TCPIP_TCP_ClientOpen	Opens a TCP socket as a client.
	TCPIP_TCP_Close	Disconnects an open socket and destroys the socket handle, releasing the associated resources.
	TCPIP_TCP_Connect	Connects a client socket.
	TCPIP_TCP_Discard	Discards any pending data in the RCP RX FIFO.
	TCPIP_TCP_Disconnect	Disconnects an open socket.
	TCPIP_TCP_FifoRxFreeGet	Determines how many bytes are free in the RX buffer/FIFO.
	TCPIP_TCP_FifoSizeAdjust	Adjusts the relative sizes of the RX and TX buffers.
	TCPIP_TCP_FifoTxFreeGet	Determines how many bytes are free and could be written in the TCP TX FIFO.
	TCPIP_TCP_FifoTxFullGet	Determines how many bytes are pending in the TCP TX FIFO.
	TCPIP_TCP_Find	Searches for a byte in the TCP RX buffer.
	TCPIP_TCP_Flush	Immediately transmits all pending TX data.
	TCPIP_TCP_Get	Retrieves a single byte to a TCP socket.
	TCPIP_TCP_GetIsReady	Determines how many bytes can be read from the TCP RX buffer.
	TCPIP_TCP_IsConnected	Determines if a socket has an established connection.
	TCPIP_TCP_IsReady	Returns the current status of the TCP module
	TCPIP_TCP_OptionsGet	Allows getting the options for a socket like: current RX/TX buffer size, etc.
	TCPIP_TCP_OptionsSet	Allows setting options to a socket like adjust RX/TX buffer size, etc.
	TCPIP_TCP_Peek	Peaks at one byte in the TCP RX buffer/FIFO without removing it from the buffer.
	TCPIP_TCP_Put	Writes a single byte to a TCP socket.
	TCPIP_TCP_PutIsReady	Determines how much free space is available in the TCP TX buffer.
	TCPIP_TCP_RemoteBind	Binds a socket to a remote address.
	TCPIP_TCP_ServerOpen	Opens a TCP socket as a server.
	TCPIP_TCP_SignalHandlerDeregister	Deregisters a previously registered TCP socket signal handler.
	TCPIP_TCP_SignalHandlerRegister	Registers a TCP socket signal handler.
	TCPIP_TCP_SocketInfoGet	Obtains information about a currently open socket.
	TCPIP_TCP_SocketNetGet	Gets the current network interface of an TCP socket.
	TCPIP_TCP_SocketNetSet	Sets the interface for an TCP socket
	TCPIP_TCP_SocketsNumberGet	Returns the number of existent TCP sockets.
	TCPIP_TCP_SocketTraceSet	Sets the current socket trace status.
	TCPIP_TCP_StringPut	Writes a null-terminated string to a TCP socket.
	TCPIP_TCP_Task	Standard TCP/IP stack module task function.
	TCPIP_TCP_WasReset	Self-clearing semaphore indicating socket reset.

Macros

	Name	Description
	INVALID_SOCKET	Invalid socket indicator macro.
	TCPIP_TCP_FifoRxFullGet	Determines how many bytes are pending in the RX buffer/FIFO.

Structures

	Name	Description
	TCP_OPTION_KEEP_ALIVE_DATA	Socket keep alive options
	TCP_OPTION_LINGER_DATA	Socket linger options.
	TCP_SOCKET_INFO	TCP socket information.
	TCPIP_TCP_MODULE_CONFIG	TCP module run-time configuration/initialization data.

Types

	Name	Description
	TCP_PORT	Defines a TCP Port number.
	TCP_SOCKET	Defines a TCP Socket.
	TCPIP_TCP_SIGNAL_FUNCTION	TCP signal handler.
	TCPIP_TCP_SIGNAL_HANDLE	TCP socket handle.

Description

Transmission Control Protocol (TCP) Communications Layer API Header File

- Provides reliable, handshaked transport of application stream oriented data with flow control
- Reference: RFC 793

File Name

tcp.h

Company

Microchip Technology Inc.

tcp_config.h

TCP configuration file

Macros

	Name	Description
	TCPIP_TCP_AUTO_TRANSMIT_TIMEOUT_VAL	Timeout before automatically transmitting unflushed data, ms. Default value is 40 ms.
	TCPIP_TCP_CLOSE_WAIT_TIMEOUT	Timeout for the CLOSE_WAIT state, ms If the remote socket closes the connection the TCP/IP stack will automatically close the socket after this timeout. If the value is set to 0 then the socket will wait forever for the application to close the socket (default behavior).
	TCPIP_TCP_DELAYED_ACK_TIMEOUT	Timeout for delayed-acknowledgment algorithm, ms
	TCPIP_TCP_DYNAMIC_OPTIONS	Enable the TCP sockets dynamic options set/get functionality If enabled, the functions: TCPIP_TCP_OptionsSet, TCPIP_TCP_OptionsGet and TCPIP_TCP_FifoSizeAdjust exist and are compiled in If disabled, these functions do not exist and cannot be used/called Note that this setting can affect modules that use TCP sockets
	TCPIP_TCP_FIN_WAIT_2_TIMEOUT	Timeout for FIN WAIT 2 state, ms
	TCPIP_TCP_KEEP_ALIVE_TIMEOUT	Timeout for keep-alive messages when no traffic is sent, ms
	TCPIP_TCP_MAX_RETRIES	Maximum number of retransmission attempts
	TCPIP_TCP_MAX_SEG_SIZE_RX_LOCAL	TCP Maximum Segment Size for RX (MSS). This value is advertised during TCP connection establishment and the remote node should obey it. The value has to be set in such a way to avoid IP layer fragmentation from causing packet loss. However, raising its value can enhance performance at the (small) risk of introducing incompatibility with certain special remote nodes (ex: ones connected via a slow dial up modem). On Ethernet networks the standard value is 1460. On dial-up links, etc. the default values should be 536. Adjust these values according to your network. Maximum Segment Size for RX (MSS)... more
	TCPIP_TCP_MAX_SEG_SIZE_RX_NON_LOCAL	Maximum Segment Size for RX (MSS) for non local destination networks.
	TCPIP_TCP_MAX_SEG_SIZE_TX	TCP Maximum Segment Size for TX. The TX maximum segment size is actually governed by the remote node's MSS option advertised during connection establishment. However, if the remote node specifies an unmanageably large MSS (ex: > Ethernet MTU), this define sets a hard limit so that TX buffers are not overflowed. If the remote node does not advertise a MSS option, all TX segments are fixed at 536 bytes maximum.
	TCPIP_TCP_MAX_SOCKETS	The maximum number of sockets to create in the stack. When defining TCPIP_TCP_MAX_SOCKETS take into account the number of interfaces the stack is supporting.
	TCPIP_TCP_MAX_SYN_RETRIES	Smaller than all other retries to reduce SYN flood DoS duration
	TCPIP_TCP_MAX_UNACKED_KEEP_ALIVES	Maximum number of keep-alive messages that can be sent without receiving a response before automatically closing the connection
	TCPIP_TCP_MSL_TIMEOUT	Maximum Segment Length (MSL) timeout, seconds This value sets the time a socket will be in the TIME_WAIT/2MSL state after the socket performed an active close. RFC recommended value is 120 sec. Other common values are: 30/60 sec Note that if this symbol is defined to 0, then the TIME_WAIT/2MSL state is skipped! This could lead to duplicate sequence numbers problems if sockets are frequently/rapidly closed and reopened and should be avoided.

	TCPIP_TCP_QUIET_TIME	This value specifies the TCP quiet time: TCP will not send any data for TCPIP_TCP_QUIET_TIME seconds after rebooting! Usualy this value should match the MSL value. Note that if this symbol is defined to 0, then the TCP quiet time state is skipped! During the quiet time there will be no TCP traffic initiated by TCP. So, clients won't be able to connect to remote hosts and servers won't reply to client requests.
	TCPIP_TCP_SOCKET_DEFAULT_RX_SIZE	Default socket RX buffer size Note that this setting affects all TCP sockets that are created and, together with TCPIP_TCP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large RX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_TCP_OptionsSet function). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Note that some modules (like HTTP) use their own settings to... more
	TCPIP_TCP_SOCKET_DEFAULT_TX_SIZE	Default socket TX buffer size Note that this setting affects all TCP sockets that are created and, together with TCPIP_TCP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large TX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_TCP_OptionsSet function). The performance of a socket is highly dependent on the size of its buffers so it's a good idea to use as large as possible buffers for the sockets that need high throughput. Note that some modules (like HTTP) use their own settings to... more
	TCPIP_TCP_START_TIMEOUT_VAL	Timeout to retransmit unacked data, ms
	TCPIP_TCP_TASK_TICK_RATE	The TCP task processing rate: number of milliseconds to generate an TCP tick. This is the tick that advances the TCP state machine. The default value is 5 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_TCP_WINDOW_UPDATE_TIMEOUT_VAL	Timeout before automatically transmitting a window update due to a TCPIP_TCP_Get() or TCPIP_TCP_ArrayGet() function call, ms.

Description

Transmission Control Protocol (TCP) Configuration file

This file contains the TCP module configuration options

File Name

tcp_config.h

Company

Microchip Technology Inc.

TFTP Module

This section describes the TCP/IP Stack Library TFTP module.

Introduction

This library provides the API of the TFTP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

TCP/IP Stack Library Trivial File Transfer Protocol (TFTP) module is simple protocol used to transfer files.

The TFTP module only supports client transfers, through the process of reading and writing files (or mail) from/to a remote server. The module cannot list directories, and currently has no provisions for user authentication. In common with other Internet protocols, it passes 8-bit bytes of data.

Using the Library

This topic describes the basic architecture of the TFTP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `tftpc.h`

The interface to the TFTP TCP/IP Stack library is defined in the `tftpc.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the `tftpc.h` TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the TFTP module.

Library Interface Section	Description
General Functions	This section provides general interface routines to TFTP
Data Types and Constants	This section provides various definitions describing this API

How the Library Works

This topic provides information on how the TFTP module works including write and read requests.

Description

Any transfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes. Each data packet contains one block of data, and must be acknowledged by an acknowledgment packet before the next packet can be sent. A data packet of less than 512 bytes, signals termination of a transfer. If a packet gets lost in the network, the intended recipient will time out and may retransmit the last packet (which may be data or an acknowledgment), thus causing the sender of the lost packet to retransmit that lost packet.

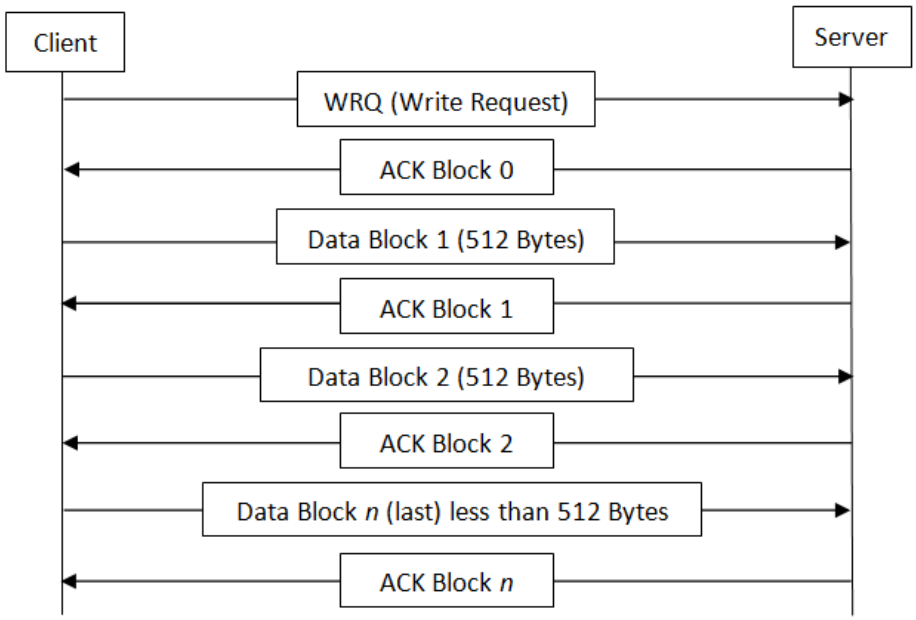
TFTP Client Server Communication

This section provides information on TFTP Write and Read requests.

Description

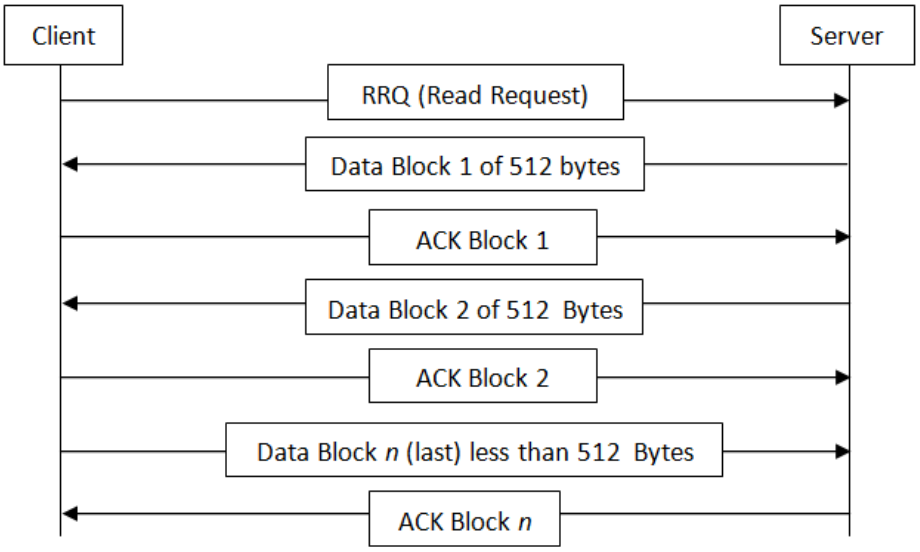
TFTP Write Request (WRQ)

The following diagram illustrates a write request.



TFTP Read Request (RRQ)

The following diagram illustrates a read request.



Configuring the Library

Macros

	Name	Description
	TCPIP_TFTPC_ARP_TIMEOUT	The number of seconds to wait before declaring a TIMEOUT error on PUT.
	TCPIP_TFTPC_CMD_PROCESS_TIMEOUT	The number of seconds to wait before declaring a TIMEOUT error on GET.
	TCPIP_TFTPC_DEFAULT_IF	The default TFTP interface for multi-homed hosts.
	TCPIP_TFTPC_FILENAME_LEN	The maximum value for the file name size.
	TCPIP_TFTPC_HOSTNAME_LEN	The maximum TFTP host name length size.
	TCPIP_TFTPC_MAX_RETRIES	The number of attempts before declaring a TIMEOUT error.
	TCPIP_TFTPC_TASK_TICK_RATE	The TFTP client task rate in milliseconds. The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_TFTPC_USER_NOTIFICATION	allow TFTP client user notification if enabled, the TCPIP_TFTPC_HandlerRegister/TCPIP_TFTPC_HandlerDeRegister functions exist and can be used

Description

The configuration of the TFTP TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the TFTP TCP/IP Stack Library. Based on the selections made, the TFTP TCP/IP Stack Library will or will not support selected features. These configuration settings will apply to all instances of the TFTP TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_TFTPC_ARP_TIMEOUT Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_ARP_TIMEOUT (3ul)
```

Description

The number of seconds to wait before declaring a TIMEOUT error on PUT.

TCPIP_TFTPC_CMD_PROCESS_TIMEOUT Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_CMD_PROCESS_TIMEOUT (3ul)
```

Description

The number of seconds to wait before declaring a TIMEOUT error on GET.

TCPIP_TFTPC_DEFAULT_IF Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_DEFAULT_IF "PIC32INT"
```

Description

The default TFTP interface for multi-homed hosts.

TCPIP_TFTPC_FILENAME_LEN Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_FILENAME_LEN (32)
```

Description

The maximum value for the file name size.

TCPIP_TFTPC_HOSTNAME_LEN Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_HOSTNAME_LEN (32)
```

Description

The maximum TFTP host name length size.

TCPIP_TFTPC_MAX_RETRIES Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_MAX_RETRIES (3u)
```

Description

The number of attempts before declaring a TIMEOUT error.

TCPIP_TFTPC_TASK_TICK_RATE Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_TASK_TICK_RATE (100)
```

Description

The TFTP client task rate in milliseconds. The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_TFTPC_USER_NOTIFICATION Macro

File

tftpc_config.h

C

```
#define TCPIP_TFTPC_USER_NOTIFICATION false
```

Description

allow TFTP client user notification if enabled, the TCPIP_TFTPC_HandlerRegister/TCPIP_TFTPC_HandlerDeRegister functions exist and can be used

Building the Library

This section lists the files that are available in the TFTP module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (.h) and source (.c) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/tftpc.c`	TFTP implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The TFTP module depends on the following module:

- TCP/IP Stack Library

Library Interface

a) Functions

	Name	Description
	TCPIP_TFTPC_Task	Standard TCP/IP stack module task function.
	TCPIP_TFTPC_GetEventNotification	Get the details of the TFTP client file mode communication event details.
	TCPIP_TFTPC_HandlerDeRegister	Deregisters a previously registered TFTP Client handler.
	TCPIP_TFTPC_HandlerRegister	Registers a TFTPC Handler.
	TCPIP_TFTPC_SetServerAddress	Set the Server IP address for TFTP Client .
	TCPIP_TFTPC_SetCommand	TFTP client command operation configuration.

b) Data Types and Constants

	Name	Description
	_TFTP_CMD_TYPE	File command mode used for TFTP PUT and GET commands.
	TCPIP_TFTP_CMD_TYPE	File command mode used for TFTP PUT and GET commands.
	TCPIP_TFTPC_MODULE_CONFIG	Placeholder for TFTP Client module configuration.
	TCPIP_TFTPC_EVENT_HANDLER	TFTPC event handler prototype.
	TCPIP_TFTPC_EVENT_TYPE	TFTP client Event Type
	TCPIP_TFTPC_HANDLE	TFTPC handle.
	TCPIP_TFTPC_OPERATION_RESULT	Standard error codes for TFTP PUT and GET command operation.

Description

This section describes the Application Programming Interface (API) functions of the TFTP module.
Refer to each section for a detailed description.

a) Functions

TCPIP_TFTPC_Task Function

Standard TCP/IP stack module task function.

File

tftpc.h

C

```
void TCPIP_TFTPC_Task();
```

Returns

None.

Description

This function performs TFTP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The TFTP module should have been initialized.

Function

void TCPIP_TFTPC_Task(void)

TCPIP_TFTPC_GetEventNotification Function

Get the details of the TFTP client file mode communication event details.

File

tftpc.h

C

```
TCPIP_TFTPC_EVENT_TYPE TCPIP_TFTPC_GetEventNotification();
```

Returns

TCPIP_TFTPC_EVENT_TYPE : It will be OR of different events.

Description

This function returns the event type TCPIP_TFTPC_EVENT_TYPE for different modes of TFTP file communication.

Preconditions

The TFTP Client module must be initialized.

Function

TCPIP_TFTPC_EVENT_TYPE TCPIP_TFTPC_GetEventNotification(void)

TCPIP_TFTPC_HandlerDeRegister Function

Deregisters a previously registered TFTP Client handler.

File

tftpc.h

C

```
bool TCPIP_TFTPC_HandlerDeRegister(TCPIP_TFTPC_HANDLE hDhcp);
```

Returns

- true - if the call succeeds
- false - if no such handler is registered

Description

This function deregisters the TFTP Client event handler.

Preconditions

The TFTP Client module must be initialized.

Parameters

Parameters	Description
htftpc	A handle returned by a previous call to TCPIP_TFTPC_HandlerRegister.

Function

bool TCPIP_TFTPC_HandlerDeRegister(TCPIP_TFTPC_HANDLE htftpc)

TCPIP_TFTPC_HandlerRegister Function

Registers a TFTPC Handler.

File

tftpc.h

C

```
TCPIP_TFTPC_HANDLE TCPIP_TFTPC_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_TFTPC_EVENT_HANDLER handler,
const void* hParam);
```

Returns

Returns a valid handle if the call succeeds, or a null handle if the call failed (out of memory, for example).

Description

This function registers a TFTPC event handler. The TFTP Client module will call the registered handler when a TFTP Client event

(TCPIP_TFTPC_EVENT_TYPE) occurs.

Remarks

The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

The hParam is passed by the client and will be used by the DHCP when the notification is made. It is used for per-thread content or if more modules, for example, share the same handler and need a way to differentiate the callback.

Preconditions

The TFTP Client module must be initialized.

Parameters

Parameters	Description
hNet	Interface handle. Use hNet == 0 to register on all interfaces available.
handler	Handler to be called when a TFTP Client event occurs.
hParam	Parameter to be used in the handler call. This is user supplied and is not used by the DHCP module.

Function

TCPIP_TFTPC_HandlerRegister(TCPIP_NET_HANDLE hNet, TCPIP_TFTPC_EVENT_HANDLER handler,

const void* hParam)

TCPIP_TFTPC_SetServerAddress Function

Set the Server IP address for TFTP Client .

File

tftpc.h

C

```
void TCPIP_TFTPC_SetServerAddress(IP_MULTI_ADDRESS* ipAddr, IP_ADDRESS_TYPE ipType);
```

Returns

None

Description

This function is used to set the TFTP server address(either it will be IPv4 address or IPv6 address). This address will be used for either Get or Put mode of TFTP Client operation.

Preconditions

The TFTP Client module must be initialized.

Parameters

Parameters	Description
ipAddr	pointer to the server address
ipType	type of address: IPv4/IPv6

Function

void TCPIP_TFTPC_SetServerAddress(IP_MULTI_ADDRESS* ipAddr,IP_ADDRESS_TYPE ipType)

TCPIP_TFTPC_SetCommand Function

TFTP client command operation configuration.

File

tftpc.h

C

```
TCPIP_TFTPC_OPERATION_RESULT TCPIP_TFTPC_SetCommand(IP_MULTI_ADDRESS* mAddr, IP_ADDRESS_TYPE ipType,
TCPIP_TFTP_CMD_TYPE cmdType, const char * fileName);
```

Returns

TFTPC_ERROR_BUSY - TFTP client is busy for one file processing and please retry later. when there is a TFTP operation going on, the another operation has to wait till the TFTP operation is completed. TFTPC_ERROR_INVALID_FILE_LENGTH - File Length should not be more than TCPIP_TFTPC_FILENAME_LEN. TFTPC_ERROR_NONE - Successful command operation.

Description

This function is used to set the client mode, server, and file name. The file name is accessed as per the TFTP command mode.

Remarks

None.

Preconditions

The TCP/IP Stack should have been initialized.

Parameters

Parameters	Description
mAddr	Server address
ipType	IP address type either IPv4 or IPv6 type
cmdType	GET or PUT command
fileName	File to be processed

Function

TCPIP_TFTPC_OPERATION_RESULT TCPIP_TFTPC_SetCommand(IP_MULTI_ADDRESS* mAddr,

- IP_ADDRESS_TYPE ipType,TCPIP_TFTP_CMD_TYPE cmdType,
- const char * fileName)

b) Data Types and Constants

TCPIP_TFTP_CMD_TYPE Enumeration

File command mode used for TFTP PUT and GET commands.

File

tftpc.h

C

```
typedef enum _TFTP_CMD_TYPE {
  TFTP_CMD_PUT_TYPE = 0,
  TFTP_CMD_GET_TYPE,
  TFTP_CMD_NONE
} TCPIP_TFTP_CMD_TYPE;
```

Members

Members	Description
TFTP_CMD_PUT_TYPE = 0	TFTP client issues a PUT command to write a file to the server
TFTP_CMD_GET_TYPE	TFTP client issues a GET command to read the file from the server

Description

Enumeration: TCPIP_TFTP_CMD_TYPE

These enum values are issued from the command line.

TCPIP_TFTPC_MODULE_CONFIG Structure

Placeholder for TFTP Client module configuration.

File

tftpc.h

C

```
typedef struct {
  const char* tftpc_interface;
  uint32_t tftpc_reply_timeout;
} TCPIP_TFTPC_MODULE_CONFIG;
```

Members

Members	Description
uint32_t tftpc_reply_timeout;	time-out for the server reply in seconds

Description

Structure: TCPIP_TFTPC_MODULE_CONFIG

This structure is a placeholder for TFTP Client module configuration.

TCPIP_TFTPC_EVENT_HANDLER Type

TFTPC event handler prototype.

File

tftpc.h

C

```
typedef void (* TCPIP_TFTPC_EVENT_HANDLER)(TCPIP_NET_HANDLE hNet, TCPIP_TFTPC_EVENT_TYPE evType,void
*buf,uint32_t bufLen, const void* param);
```

Description

Type: TCPIP_TFTPC_EVENT_HANDLER

Prototype of a TFTPC event handler. Clients can register a handler with the TFTP service. Once an TFTP event occurs the TFTP Client service will be called the registered handler. The handler has to be short and fast. It is meant for setting an event flag, *not* for lengthy processing! buf - Buffer is used to provide the memory Pointer . buf type need to be typecasted to char* while processing bufLen - The number of bytes present in the buffer.

TCPIP_TFTPC_EVENT_TYPE Enumeration

TFTP client Event Type

File

tftpc.h

C

```
typedef enum {
  TFTPC_EVENT_NONE = 0,
  TFTPC_EVENT_PUT_REQUEST = 0x0001,
  TFTPC_EVENT_GET_REQUEST = 0x0002,
  TFTPC_EVENT_ACKED = 0x0004,
  TFTP_EVENT_DATA_RECEIVED = 0x0008,
  TFTPC_EVENT_DECLINE = 0x0010,
  TFTPC_EVENT_TIMEOUT = 0x0020,
  TFTPC_EVENT_COMPLETED = 0x0040,
  TFTPC_EVENT_CONN_LOST = 0x0080,
  TFTPC_EVENT_CONN_ESTABLISHED = 0x0100,
  TFTPC_EVENT_SERVICE_DISABLED = 0x0200,
  TFTPC_EVENT_BUSY = 0x0400
} TCPIP_TFTPC_EVENT_TYPE;
```

Members

Members	Description
TFTPC_EVENT_NONE = 0	TFTP no event
TFTPC_EVENT_PUT_REQUEST = 0x0001	TFTP PUT request sent
TFTPC_EVENT_GET_REQUEST = 0x0002	TFTP GET Request sent
TFTPC_EVENT_ACKED = 0x0004	TFTP request acknowledge was received
TFTP_EVENT_DATA_RECEIVED = 0x0008	TFTP Client received Data for GET request
TFTPC_EVENT_DECLINE = 0x0010	TFTP File Put or Get communication declined due to Bad PDU

TFTPC_EVENT_TIMEOUT = 0x0020	TFTP server timeout
TFTPC_EVENT_COMPLETED = 0x0040	TFTP File Put or Get completed
TFTPC_EVENT_CONN_LOST = 0x0080	connection to the TFTP server lost
TFTPC_EVENT_CONN_ESTABLISHED = 0x0100	connection re-established
TFTPC_EVENT_SERVICE_DISABLED = 0x0200	TFTP service disabled
TFTPC_EVENT_BUSY = 0x0400	TFTP Client communication is going on

Description

Enumeration: TCPIP_TFTPC_EVENT_TYPE

None.

TCPIP_TFTPC_HANDLE Type

TFTPC handle.

File

tftpc.h

C

```
typedef const void* TCPIP_TFTPC_HANDLE;
```

Description

Type: TCPIP_TFTPC_HANDLE

A handle that a client can use after the event handler has been registered.

TCPIP_TFTPC_OPERATION_RESULT Enumeration

Standard error codes for TFTP PUT and GET command operation.

File

tftpc.h

C

```
typedef enum {
  TFTPC_ERROR_NONE = 0,
  TFTPC_ERROR_FILE_NOT_FOUND = -1,
  TFTPC_ERROR_BUSY = -2,
  TFTPC_ERROR_DISK_FULL = -3,
  TFTPC_ERROR_INVALID_OPERATION = -4,
  TFTPC_ERROR_UNKNOWN_TID = -5,
  TFTPC_ERROR_FILE_EXISTS = -6,
  TFTPC_ERROR_NO_SUCH_USE = -7,
  TFTPC_ERROR_DNS_RESOLVE_ERR = -8,
  TFTPC_ERROR_INVALID_INTERFACE = -9,
  TFTPC_ERROR_INVALID_FILE_LENGTH = -10,
  TFTPC_ERROR_INVALID_SERVER_ADDR = -11
} TCPIP_TFTPC_OPERATION_RESULT;
```

Members

Members	Description
TFTPC_ERROR_FILE_NOT_FOUND = -1	TFTP client file not found
TFTPC_ERROR_BUSY = -2	TFTP client is busy when one file put or get transfer is going on.
TFTPC_ERROR_DISK_FULL = -3	TFTP client buffer full
TFTPC_ERROR_INVALID_OPERATION = -4	TFTP client invalid command operation
TFTPC_ERROR_UNKNOWN_TID = -5	TFTP ID error
TFTPC_ERROR_FILE_EXISTS = -6	TFTP client file already exists
TFTPC_ERROR_NO_SUCH_USE = -7	TFTP client not in use
TFTPC_ERROR_DNS_RESOLVE_ERR = -8	TFTP client DNS resolve error
TFTPC_ERROR_INVALID_INTERFACE = -9	TFTP client interface error
TFTPC_ERROR_INVALID_FILE_LENGTH = -10	TFTP client file length is more than the expected size, which should be the size of SYS_FS_MAX_PATH

TFTPC_ERROR_INVALID_SERVER_ADDR = -11	Invalid Server Address

Description

Enumeration: TCPIP_TFTPC_OPERATION_RESULT

This enumeration defines the standard error codes for TFTP PUT and GET command operation.

Files

Files

Name	Description
tftpc.h	The TFTP Client module implements the Trivial File Transfer Protocol (TFTP).
tftpc_config.h	TFTP Client configuration file.

Description

This section lists the source and header files used by the library.

tftpc.h

The TFTP Client module implements the Trivial File Transfer Protocol (TFTP).

Enumerations

	Name	Description
	_TFTP_CMD_TYPE	File command mode used for TFTP PUT and GET commands.
	TCPIP_TFTP_CMD_TYPE	File command mode used for TFTP PUT and GET commands.
	TCPIP_TFTPC_EVENT_TYPE	TFTP client Event Type
	TCPIP_TFTPC_OPERATION_RESULT	Standard error codes for TFTP PUT and GET command operation.

Functions

	Name	Description
	TCPIP_TFTPC_GetEventNotification	Get the details of the TFTP client file mode communication event details.
	TCPIP_TFTPC_HandlerDeRegister	Deregisters a previously registered TFTP Client handler.
	TCPIP_TFTPC_HandlerRegister	Registers a TFTPC Handler.
	TCPIP_TFTPC_SetCommand	TFTP client command operation configuration.
	TCPIP_TFTPC_SetServerAddress	Set the Server IP address for TFTP Client .
	TCPIP_TFTPC_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_TFTPC_MODULE_CONFIG	Placeholder for TFTP Client module configuration.

Types

	Name	Description
	TCPIP_TFTPC_EVENT_HANDLER	TFTPC event handler prototype.
	TCPIP_TFTPC_HANDLE	TFTPC handle.

Description

TFTP Client Module API Header File

The TFTP Client module implements the Trivial File Transfer Protocol (TFTP). By default, the module opens a client socket for the default interface configuration. From the command prompt, the TFTP client module mode will be selected. At present only two modes are supported: Read and Write.

- For Read mode - File will be fetched from the Server using the GET command.
- For Write mode - Server will be able to fetch the file from the client using the PUT command.

The TFTP module needs the file system for GET and PUT command operation. When one mode is in operation, access to the other mode or another server is not allowed.

File Name

tftpc.h

Company

Microchip Technology Inc.

tftpc_config.h

TFTP Client configuration file.

Macros

	Name	Description
	TCPIP_TFTPC_ARP_TIMEOUT	The number of seconds to wait before declaring a TIMEOUT error on PUT.
	TCPIP_TFTPC_CMD_PROCESS_TIMEOUT	The number of seconds to wait before declaring a TIMEOUT error on GET.
	TCPIP_TFTPC_DEFAULT_IF	The default TFTP interface for multi-homed hosts.
	TCPIP_TFTPC_FILENAME_LEN	The maximum value for the file name size.
	TCPIP_TFTPC_HOSTNAME_LEN	The maximum TFTP host name length size.
	TCPIP_TFTPC_MAX_RETRIES	The number of attempts before declaring a TIMEOUT error.
	TCPIP_TFTPC_TASK_TICK_RATE	The TFTP client task rate in milliseconds. The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_TFTPC_USER_NOTIFICATION	allow TFTP client user notification if enabled, the TCPIP_TFTPC_HandlerRegister/TCPIP_TFTPC_HandlerDeRegister functions exist and can be used

Description

Trivial File Transfer Protocol (TFTP) Client Configuration file

This file contains the TFTP Client module configuration options.

File Name

tftpc_config.h

Company

Microchip Technology Inc.

Telnet Module

This section describes the TCP/IP Stack Library Telnet module.

Introduction

TCP/IP Stack Library Telnet Module for Microchip Microcontrollers

This library provides the API of the Telnet (server) module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

Telnet, which runs over a TCP connection, provides bidirectional, interactive communication between two nodes on the Internet or on a Local Area Network. The Telnet code included with Microchip's TCP/IP stack is a demonstration of the possible utilization of a Telnet server module. The server listens for a Telnet connection. When a client attempts to make one, the demonstration will prompt the client for a username and password, and if the correct one is provided, it will connect the client to a console service running in the system (if this console service is enabled). From this moment on, the Telnet connection can be used for various system and TCP/IP related commands, just as the regular system console is used.

The commands currently implemented and processed by the command processor can be dynamically updated, new commands can be added, etc. See the `tcpip_commands.c` file for the list of TCP/IP-related commands.

Based on this skeleton Telnet code, completely new behavior can be implemented for the Telnet server.

Using the Library

This topic describes the basic architecture of the Telnet TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `telnet.h`

The interface to the Telnet TCP/IP Stack library is defined in the `telnet.h` header file. This file is included by the `tcpip.h` file. Any C language

source (.c) file that uses the Telnet TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

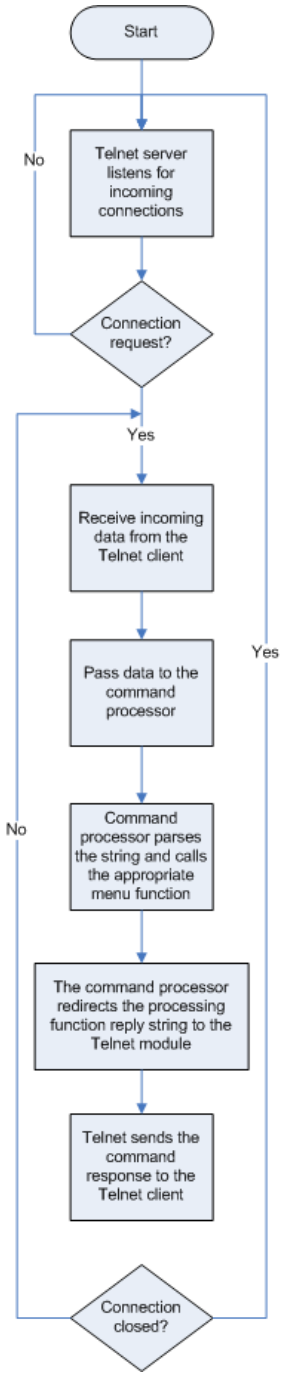
This library provides the API of the Telnet TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Currently, the Telnet TCP/IP Library runs in the background as a server and does not expose any API functions.

It serves incoming connections by sending the received data string to the command processor, which outputs each command reply to the Telnet channel.

Telnet Software Abstraction Block Diagram



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Telnet module.

Library Interface Section	Description
Functions	This section provides a routine that performs Telnet module tasks in the TCP/IP stack.
Data Types and Constants	This section provides various definitions describing this API

Configuring the Library

Macros

	Name	Description
	TCPIP_TELNET_MAX_CONNECTIONS	Maximum number of Telnet connections
	TCPIP_TELNET_PASSWORD	Default Telnet password
	TCPIP_TELNET_TASK_TICK_RATE	telnet task rate, milliseconds The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_TELNET_USERNAME	Default Telnet user name

Description

The configuration of the Telnet TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the Telnet TCP/IP Stack Library. Based on the selections made, the Telnet TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the Telnet TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_TELNET_MAX_CONNECTIONS Macro

File

telnet_config.h

C

```
#define TCPIP_TELNET_MAX_CONNECTIONS (2u)
```

Description

Maximum number of Telnet connections

TCPIP_TELNET_PASSWORD Macro

File

telnet_config.h

C

```
#define TCPIP_TELNET_PASSWORD "microchip"
```

Description

Default Telnet password

TCPIP_TELNET_TASK_TICK_RATE Macro

File

telnet_config.h

C

```
#define TCPIP_TELNET_TASK_TICK_RATE 100
```

Description

telnet task rate, milliseconds The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.

TCPIP_TELNET_USERNAME Macro

File

telnet_config.h

C

```
#define TCPIP_TELNET_USERNAME "admin"
```

Description

Default Telnet user name

Building the Library

This section lists the files that are available in the Telnet module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/telnet.c`	Telnet implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Telnet module depends on the following modules:

- TCP/IP Stack Library
- TCP Module

Library Interface

a) Functions

	Name	Description
	TCPIP_TELNET_Task	Standard TCP/IP stack module task function.

b) Data Types and Constants

	Name	Description
	TCPIP_TELNET_MODULE_CONFIG	Telnet Configuration structure placeholder

Description

This section describes the Application Programming Interface (API) functions of the Telnet module.

Refer to each section for a detailed description.

a) Functions

TCPIP_TELNET_Task Function

Standard TCP/IP stack module task function.

File

telnet.h

C

```
void TCPIP_TELNET_Task();
```

Returns

None.

Description

This function performs Telnet module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The Telnet module should have been initialized.

Function

void TCPIP_TELNET_Task(void)

b) Data Types and Constants

TCPIP_TELNET_MODULE_CONFIG Structure

File

telnet.h

C

```
typedef struct {
} TCPIP_TELNET_MODULE_CONFIG;
```

Description

Telnet Configuration structure placeholder

Files

Files

Name	Description
telnet.h	Telnet provides bidirectional, interactive communication between two nodes on the Internet or on a Local Area Network.
telnet_config.h	Configuration file

Description

This section lists the source and header files used by the library.

telnet.h

Telnet provides bidirectional, interactive communication between two nodes on the Internet or on a Local Area Network.

Functions

	Name	Description
	TCPIP_TELNET_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	TCPIP_TELNET_MODULE_CONFIG	Telnet Configuration structure placeholder

Description

Telnet Server Module for Microchip TCP/IP Stack. The telnet protocol is explained in RFC 854.

File Name

telnet.h

Company

Microchip Technology Inc.

telnet_config.h

Configuration file

Macros

	Name	Description
	TCPIP_TELNET_MAX_CONNECTIONS	Maximum number of Telnet connections
	TCPIP_TELNET_PASSWORD	Default Telnet password
	TCPIP_TELNET_TASK_TICK_RATE	telnet task rate, milliseconds The default value is 100 milliseconds. The lower the rate (higher the frequency) the higher the module priority and higher module performance can be obtained The value cannot be lower than the TCPIP_STACK_TICK_RATE.
	TCPIP_TELNET_USERNAME	Default Telnet user name

Description

Telnet Configuration file

This file contains the Telnet module configuration options

File Name

telnet_config.h

Company

Microchip Technology Inc.

UDP Module

This section describes the TCP/IP Stack Library UDP module.

Introduction

TCP/IP Stack Library UDP Module for Microchip Microcontrollers

This library provides the API of the UDP module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

UDP is a standard transport layer protocol described in RFC 768. It provides fast but unreliable data-gram based transfers over networks, and

forms the foundation SNTP, SNMP, DNS, and many other protocol standards.

Using the Library

This topic describes the basic architecture of the UDP TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header File: `udp.h`

The interface to the UDP TCP/IP Stack library is defined in the `udp.h` header file. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the UDP TCP/IP Stack library should include `tcpip.h`.

Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

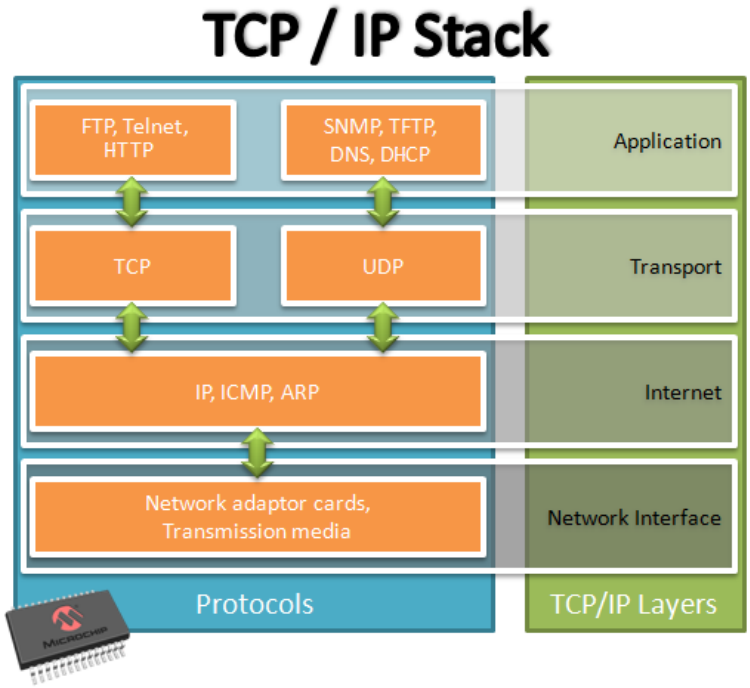
This library provides the API of the UDP TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

UDP Software Abstraction Block Diagram

This module provides software abstraction of the UDP module existent in any TCP/IP Stack implementation. It allows a user to UDP network traffic by opening and using UDP sockets.

Typical UDP Implementation



Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the UDP module.

Library Interface Section	Description
Socket Management Functions	Routines for Managing UDP Sockets
Transmit Data Transfer Functions	Routines for Managing Outgoing Data Transmissions
Receive Data Transfer Functions	Routines for Managing Incoming Data Transmissions
Data Types and Constants	This section provides various definitions describing This API

How the Library Works

This topic describes how the UDP TCP/IP Stack Library works.

Description

Connections over UDP should be thought of as data-gram based transfers. Each packet is a separate entity, the application should expect some packets to arrive out-of-order or even fail to reach the destination node. This is in contrast to TCP, in which the connection is thought of as a stream and network errors are automatically corrected. These tradeoffs in reliability are made for an increase in throughput. In general, UDP transfers operate two to three times faster than those made over TCP.

UDP sockets have their own TX buffers. Once the transmit buffer has valid data it is the socket user's responsibility to send the data over the network , calling the corresponding TCPIP_UDP_Flush function.

There is no state machine within the UDP module to automatically take care of the data transmission.

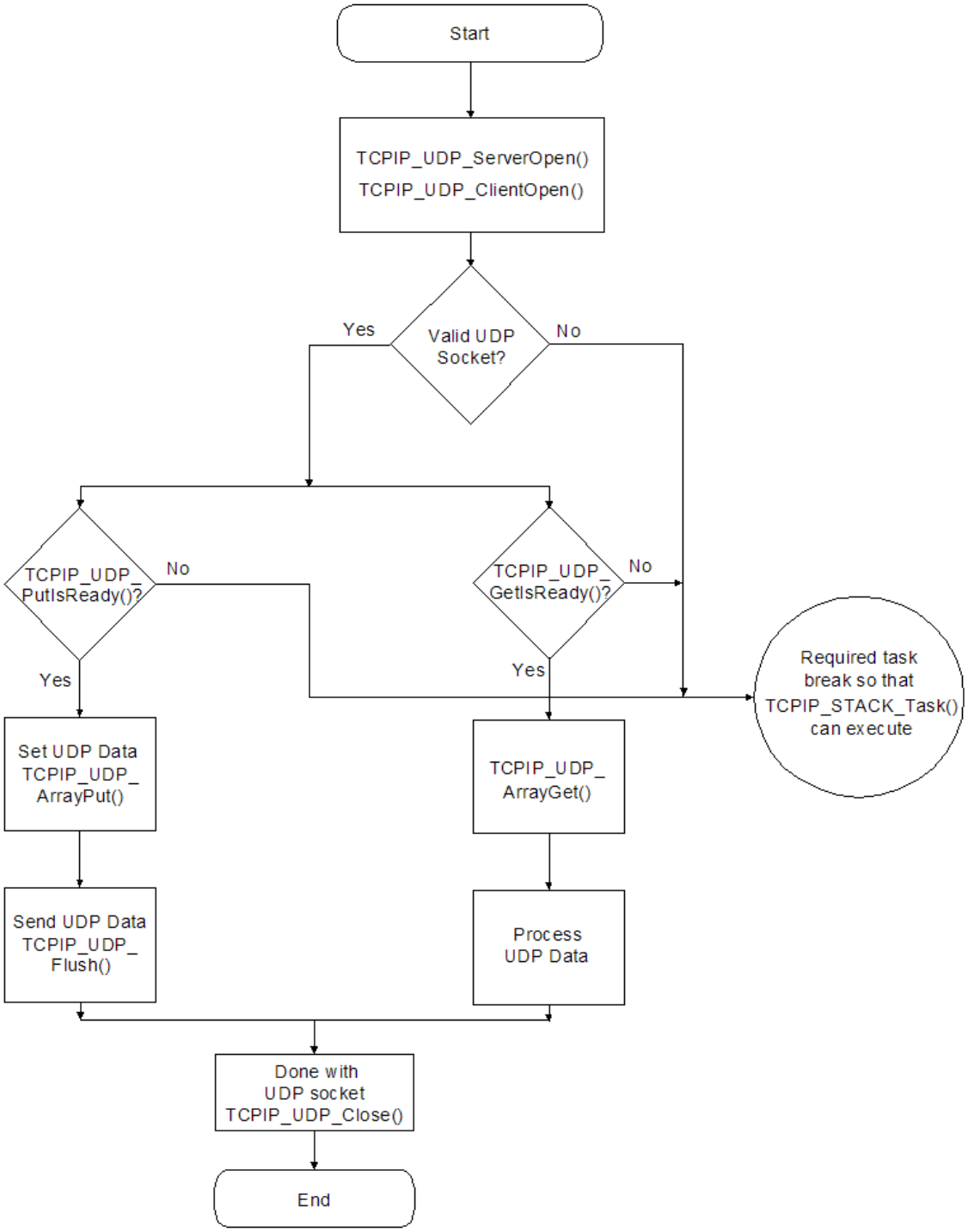
On the receive side, a UDP socket does not have its own RX buffer; however, it will use the RX buffer that was allocated by the corresponding MAC driver for receiving a network packet. What this means is that the user of the UDP socket will have to consume that pending RX data (or discard it), because eventually, the stack will run out of memory for other incoming traffic.

Core Functionality

This topic describes the core functionality of the UDP TCP/IP Stack Library.

Description

The following flow diagram provides an overview for the use of the UDP module:



Server/listening sockets are opened using TCPIP_UDP_ServerOpen. Client sockets are opened using TCPIP_UDP_ClientOpen. A client socket needs to have a remote address specified before it can transmit data.

Once the socket is opened, you can immediately begin transmitting data. To transmit a segment, call TCPIP_UDP_PutIsReady to determine how many bytes can be written. Then, use any of the TCPIP_UDP_ArrayPut family of functions to write data to the socket. Once all data has been written, call TCPIP_UDP_Flush to build and transmit the packet. Since each UDP socket has its own TX buffer, the sequence previously described can be executed in multiple steps. That data that is written in the socket is persistent and it will be stored until TCPIP_UDP_Flush is called.

Configuring the Library

Macros

	Name	Description
	TCPIP_UDP_MAX_SOCKETS	Maximum number of UDP sockets that can be opened simultaneously These sockets will be created when the module is initialized.
	TCPIP_UDP_SOCKET_DEFAULT_RX_QUEUE_LIMIT	The maximum number of RX packets that can be queued by an UDP socket at a certain time. Note that UDP sockets do not use their own RX buffers but instead use the network driver supplied packets and a timely processing is critical to avoid packet memory starvation for the whole stack. This symbol sets the maximum number of UDP buffers/packets that can be queued for a UDP socket at a certain time. Once this limit is reached further incoming packets are silently discarded. Adjust depending on the number of RX buffers that are available for the stack and the... more
	TCPIP_UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT	The maximum number of TX packets that can be queued by an UDP socket at a certain time. For sockets that need to transfer a lot of data (Iperf, for example), especially on slow connections this limit prevents running out of memory because the MAC/PHY transfer cannot keep up with the UDP packet allocation rate imposed by the application. Adjust depending on the TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE, the connection speed and the amount of memory available to the stack.
	TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE	Default socket TX buffer size. Note that this setting affects all UDP sockets that are created and, together with TCPIP_UDP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large TX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_UDP_OptionsSet function).
	TCPIP_UDP_SOCKET_POOL_BUFFER_SIZE	Size of the buffers in the UDP pool. Any UDP socket that is enabled to use the pool and has the TX size <= than this size can use a buffer from the pool. Note that this setting, together with TCPIP_UDP_SOCKET_POOL_BUFFERS, has impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting).
	TCPIP_UDP_SOCKET_POOL_BUFFERS	Number of buffers in the private UDP pool. These are preallocated buffers that are to be used by UDP sockets only. This improves the UDP socket throughput and is meant only for UDP sockets that have to sustain high TX traffic rate. However, this memory is not returned to the stack heap, it always belongs to UDP. A socket needs to have an option set in order to use the buffers pool (see the UDPSetOptions()). Use 0 to disable the feature.
	TCPIP_UDP_USE_RX_CHECKSUM	Check incoming packets to have proper checksum.
	TCPIP_UDP_USE_TX_CHECKSUM	Calculate and transmit a checksum when sending data. Checksum is not mandatory for UDP packets but is highly recommended. This will affect the UDP TX performance.
	TCPIP_UDP_USE_POOL_BUFFERS	enable the build of the pre-allocated pool buffers option

Description

The configuration of the UDP TCP/IP Stack Library is based on the file `system_config.h`. (which may include `udp_config.h`).

This header file contains the configuration selection for the UDP TCP/IP Stack Library. Based on the selections made, the UDP TCP/IP Stack Library may support the selected features. These configuration settings will apply to the single instance of the UDP TCP/IP Stack Library for all sockets.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TCPIP_UDP_MAX_SOCKETS Macro

File

udp_config.h

C

```
#define TCPIP_UDP_MAX_SOCKETS (10)
```

Description

Maximum number of UDP sockets that can be opened simultaneously These sockets will be created when the module is initialized.

TCPIP_UDP_SOCKET_DEFAULT_RX_QUEUE_LIMIT Macro

File

udp_config.h

C

```
#define TCPIP_UDP_SOCKET_DEFAULT_RX_QUEUE_LIMIT 3
```

Description

The maximum number of RX packets that can be queued by an UDP socket at a certain time. Note that UDP sockets do not use their own RX buffers but instead use the network driver supplied packets and a timely processing is critical to avoid packet memory starvation for the whole stack. This symbol sets the maximum number of UDP buffers/packets that can be queued for a UDP socket at a certain time. Once this limit is reached further incoming packets are silently discarded. Adjust depending on the number of RX buffers that are available for the stack and the amount of memory available to the stack.

TCPIP_UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT Macro

File

udp_config.h

C

```
#define TCPIP_UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT 3
```

Description

The maximum number of TX packets that can be queued by an UDP socket at a certain time. For sockets that need to transfer a lot of data (Iperf, for example), especially on slow connections this limit prevents running out of memory because the MAC/PHY transfer cannot keep up with the UDP packet allocation rate imposed by the application. Adjust depending on the TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE, the connection speed and the amount of memory available to the stack.

TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE Macro

File

udp_config.h

C

```
#define TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE 512
```

Description

Default socket TX buffer size. Note that this setting affects all UDP sockets that are created and, together with TCPIP_UDP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large TX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_UDP_OptionsSet function).

TCPIP_UDP_SOCKET_POOL_BUFFER_SIZE Macro

File

udp_config.h

C

```
#define TCPIP_UDP_SOCKET_POOL_BUFFER_SIZE 512
```

Description

Size of the buffers in the UDP pool. Any UDP socket that is enabled to use the pool and has the TX size <= than this size can use a buffer from the pool. Note that this setting, together with TCPIP_UDP_SOCKET_POOL_BUFFERS, has impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting).

TCPIP_UDP_SOCKET_POOL_BUFFERS Macro

File

udp_config.h

C

```
#define TCPIP_UDP_SOCKET_POOL_BUFFERS 4
```

Description

Number of buffers in the private UDP pool. These are preallocated buffers that are to be used by UDP sockets only. This improves the UDP socket throughput and is meant only for UDP sockets that have to sustain high TX traffic rate. However, this memory is not returned to the stack heap, it always belongs to UDP. A socket needs to have an option set in order to use the buffers pool (see the UDPSetOptions()). Use 0 to disable the feature.

TCPIP_UDP_USE_RX_CHECKSUM Macro

File

udp_config.h

C

```
#define TCPIP_UDP_USE_RX_CHECKSUM
```

Description

Check incoming packets to have proper checksum.

TCPIP_UDP_USE_TX_CHECKSUM Macro

File

udp_config.h

C

```
#define TCPIP_UDP_USE_TX_CHECKSUM
```

Description

Calculate and transmit a checksum when sending data. Checksum is not mandatory for UDP packets but is highly recommended. This will affect the UDP TX performance.

TCPIP_UDP_USE_POOL_BUFFERS Macro

File

udp_config.h

C

```
#define TCPIP_UDP_USE_POOL_BUFFERS false
```

Description

enable the build of the pre-allocated pool buffers option

Building the Library

This section lists the files that are available in the UDP module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/udp.c	UDP implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The UDP module depends on the following modules:

- TCP/IP Stack Library
- IPv4 Module

Library Interface

a) Socket Management Functions

	Name	Description
	TCPIP_UDP_ServerOpen	Opens a UDP socket as a server.
	TCPIP_UDP_ClientOpen	Opens a UDP socket as a client.
	TCPIP_UDP_IsOpened	Determines if a socket was opened.
	TCPIP_UDP_IsConnected	Determines if a socket has an established connection.
	TCPIP_UDP_Bind	Bind a socket to an address type, local port and address. This function is meant primarily for client sockets.
	TCPIP_UDP_RemoteBind	Bind a socket to an address type, remote port and address. This function is meant primarily for server sockets.
	TCPIP_UDP_Close	Closes a UDP socket and frees the handle.
	TCPIP_UDP_OptionsGet	Allows getting the options for a socket such as current RX/TX buffer size, etc.
	TCPIP_UDP_OptionsSet	Allows setting options to a socket like adjust RX/TX buffer size, etc
	TCPIP_UDP_SocketInfoGet	Returns information about a selected UDP socket.
	TCPIP_UDP_SocketNetGet	Gets the network interface of an UDP socket
	TCPIP_UDP_SocketNetSet	Sets the network interface for an UDP socket
	TCPIP_UDP_TxOffsetSet	Moves the pointer within the TX buffer.
	TCPIP_UDP_SourceIPAddressSet	Sets the source IP address of a socket
	TCPIP_UDP_BcastIPV4AddressSet	Sets the broadcast IP address of a socket Allows an UDP socket to send broadcasts.
	TCPIP_UDP_DestinationIPAddressSet	Sets the destination IP address of a socket
	TCPIP_UDP_DestinationPortSet	Sets the destination port of a socket
	TCPIP_UDP_Disconnect	Disconnects a UDP socket and re-initializes it.
	TCPIP_UDP_SignalHandlerDeregister	Deregisters a previously registered UDP socket signal handler.
	TCPIP_UDP_SignalHandlerRegister	Registers a UDP socket signal handler.
	TCPIP_UDP_Task	Standard TCP/IP stack module task function.
	TCPIP_UDP_SocketsNumberGet	Returns the number of existent UDP sockets.

b) Transmit Data Transfer Functions

	Name	Description
	TCPIP_UDP_PutIsReady	Determines how many bytes can be written to the UDP socket.
	TCPIP_UDP_TxPutIsReady	Determines how many bytes can be written to the UDP socket.
	TCPIP_UDP_ArrayPut	Writes an array of bytes to the UDP socket.

	TCPIP_UDP_StringPut	Writes a null-terminated string to the UDP socket.
	TCPIP_UDP_Put	Writes a byte to the UDP socket.
	TCPIP_UDP_TxCountGet	Returns the amount of bytes written into the UDP socket.
	TCPIP_UDP_Flush	Transmits all pending data in a UDP socket.

c) Receive Data Transfer Functions

	Name	Description
	TCPIP_UDP_GetIsReady	Determines how many bytes can be read from the UDP socket.
	TCPIP_UDP_ArrayGet	Reads an array of bytes from the UDP socket.
	TCPIP_UDP_Get	Reads a byte from the UDP socket.
	TCPIP_UDP_RxOffsetSet	Moves the read pointer within the socket RX buffer.
	TCPIP_UDP_Discard	Discards any remaining RX data from a UDP socket.

d) Data Types and Constants

	Name	Description
	INVALID_UDP_SOCKET	Indicates a UDP socket that is not valid
	UDP_PORT	Defines a UDP Port Number
	UDP_SOCKET	Provides a handle to a UDP Socket
	UDP_SOCKET_BCAST_TYPE	This is type UDP_SOCKET_BCAST_TYPE.
	UDP_SOCKET_INFO	Information about a socket
	UDP_SOCKET_OPTION	UDP socket options
	TCPIP_UDP_MODULE_CONFIG	UDP module run time configuration/initialization data.
	TCPIP_UDP_SIGNAL_FUNCTION	UDP Signal Handler.
	TCPIP_UDP_SIGNAL_HANDLE	UDP socket handle.
	TCPIP_UDP_SIGNAL_TYPE	UDP run-time signal types.

Description

This section describes the Application Programming Interface (API) functions of the UDP module.

Refer to each section for a detailed description.

a) Socket Management Functions

TCPIP_UDP_ServerOpen Function

Opens a UDP socket as a server.

File

udp.h

C

```
UDP_SOCKET TCPIP_UDP_ServerOpen(IP_ADDRESS_TYPE addType, UDP_PORT localPort, IP_MULTI_ADDRESS*
localAddress);
```

Returns

- INVALID_SOCKET - No sockets of the specified type were available to be opened.
- A UDP_SOCKET handle - Save this handle and use it when calling all other UDP APIs.

Description

Provides a unified method for opening UDP server sockets.

Preconditions

UDP is initialized.

Example

IP_ADDRESS_TYPE_IPV4 or IP_ADDRESS_TYPE_IPV6. It can be IP_ADDRESS_TYPE_ANY and the server socket will accept any incoming type of connection.

UDP_PORT localPort - UDP port on which to listen for connections

IP_MULTI_ADDRESS* localAddress - Local address to use. Can be NULL if any incoming interface will do.

Parameters

Parameters	Description
IP_ADDRESS_TYPE addType	The type of address being used.

Function

UDP_SOCKET TCPIP_UDP_ServerOpen(IP_ADDRESS_TYPE addType, UDP_PORT localPort,
IP_MULTI_ADDRESS* localAddress)

TCPIP_UDP_ClientOpen Function

Opens a UDP socket as a client.

File

udp.h

C

```
UDP_SOCKET TCPIP_UDP_ClientOpen(IP_ADDRESS_TYPE addType, UDP_PORT remotePort, IP_MULTI_ADDRESS*
remoteAddress);
```

Returns

- INVALID_SOCKET - No sockets of the specified type were available to be opened.
- A UDP_SOCKET handle - Save this handle and use it when calling all other UDP APIs.

Description

Provides a unified method for opening UDP client sockets.

Remarks

None.

Preconditions

UDP is initialized.

Parameters

Parameters	Description
IP_ADDRESS_TYPE addType	The type of address being used. Example: IP_ADDRESS_TYPE_IPV4 or IP_ADDRESS_TYPE_IPV6. It can also be IP_ADDRESS_TYPE_ANY if the type of the socket will be specified later (using TCPIP_UDP_Bind, TCPIP_UDP_RemoteBind).
UDP_PORT remotePort	The remote UDP port to which a connection should be made. The local port for client sockets will be automatically picked by the UDP module.
IP_MULTI_ADDRESS* remoteAddress	The remote address to connect to. Not used if addType == IP_ADDRESS_TYPE_ANY. Can be NULL if later set by a call to TCPIP_UDP_RemoteBind.

Function

TCPIP_UDP_ClientOpen(IP_ADDRESS_TYPE addType, UDP_PORT remotePort,
IP_MULTI_ADDRESS* remoteAddress)

TCPIP_UDP_IsOpened Macro

Determines if a socket was opened.

File

udp.h

C

```
#define TCPIP_UDP_IsOpened(hUDP) TCPIP_UDP_IsConnected(hUDP)
```

Description

This function determines if a socket was opened.

Remarks

This is a backward compatibility call.

Preconditions

None.

Parameters

Parameters	Description
hUDP	The socket to check.

Function

bool TCPIP_UDP_IsOpened(UDP_SOCKET hUDP)

TCPIP_UDP_IsConnected Function

Determines if a socket has an established connection.

File

udp.h

C

```
bool TCPIP_UDP_IsConnected(UDP_SOCKET hUDP);
```

Description

This function determines if a socket has an established connection to a remote node. Call this function after calling TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen to determine when the connection is set up and ready for use.

Remarks

An UDP server or client socket can always send data, as long as it has a valid destination address, even if there's no remote socket connected to it. This function will return true if the socket is opened and currently has a valid destination. Note that this is true if a client socket was opened with a remote host address, a socket received data from a remote host, or a explicit remote bind was done.

Preconditions

None.

Parameters

Parameters	Description
hUDP	The socket to check.

Function

bool TCPIP_UDP_IsConnected(UDP_SOCKET hUDP)

TCPIP_UDP_Bind Function

Bind a socket to an address type, local port and address. This function is meant primarily for client sockets.

File

udp.h

C

```
bool TCPIP_UDP_Bind(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType, UDP_PORT localPort, IP_MULTI_ADDRESS*
localAddress);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Sockets don't need specific binding, it is done automatically by the stack However, specific binding can be requested using this function. Works for

both client and server sockets. For a server socket it will restrict accepting connections of a specific type on a specific interface only. For a client socket it will force a specific address type and a local port and interface.

Remarks

If address type IP_ADDRESS_TYPE_ANY is used, the localAddress parameter won't be used and the socket won't change the local address. However, for the call to succeed, socket must have been created with IP_ADDRESS_TYPE_ANY type and must not be currently bound to a IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6 type.

If address type IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6 is used, then the socket type will be changed accordingly, and the localAddress will be used, if provided. The socket will be bound to this type of address. For the call to succeed the socket must currently have the type IP_ADDRESS_TYPE_ANY or to match the addType parameter.

If localAddress is the valid address of a network interface, and addType != IP_ADDRESS_TYPE_ANY, then the call will enforce UDP_OPTION_STRICT_NET on the socket.

If localPort is 0, the stack will assign a unique local port (if the socket doesn't already have a unique local port)

In order to change dynamically the type of address, the socket must have been created with a IP_ADDRESS_TYPE_ANY type and not currently bound to a different address type. TCPIP_UDP_Disconnect could be issued to remove a previous IP type binding.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket

Parameters

Parameters	Description
hUDP	The socket to bind.
addType	The type of address being used. Valid values are IP_ADDRESS_TYPE_ANY/IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6.
localPort	The local port to bind to.
localAddress	Local address to use.

Function

bool TCPIP_UDP_Bind(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType, UDP_PORT localPort,
IP_MULTI_ADDRESS* localAddress);

TCPIP_UDP_RemoteBind Function

Bind a socket to an address type, remote port and address. This function is meant primarily for server sockets.

File

udp.h

C

```
bool TCPIP_UDP_RemoteBind(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType, UDP_PORT remotePort, IP_MULTI_ADDRESS*
remoteAddress);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Sockets don't need specific remote binding, they should accept connections on any incoming interface. Therefore, the binding is done automatically by the stack. However, specific remote binding can be requested using this function. For a server socket it can be used to restrict accepting connections from a specific remote host. For a client socket it will just change the default binding done when the socket was opened.

Remarks

If address type IP_ADDRESS_TYPE_ANY is used, the remoteAddress parameter won't be used and the socket won't change the remote destination address. However, for the call to succeed, socket must have been created with IP_ADDRESS_TYPE_ANY type and must not be currently bound to a IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6 type.

If address type IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6 is used, then the socket type will be changed accordingly, and the remoteAddress will be used, if provided. The socket will be bound to this type of address. For the call to succeed the socket must currently have the type IP_ADDRESS_TYPE_ANY or to match the addType parameter.

If the remoteAddress != 0 and addType != IP_ADDRESS_TYPE_ANY, the call will enforce UDP_OPTION_STRICT_ADDRESS on the socket.

The remote port is always changed, even if remotePort == 0. It will enforce UDP_OPTION_STRICT_PORT on the socket.

In order to change dynamically the type of address, the socket must have been created with a IP_ADDRESS_TYPE_ANY type and not currently bound to a different address type. TCPIP_UDP_Disconnect could be issued to remove a previous IP type binding.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	The socket to bind.
addType	The type of address being used. Valid values are IP_ADDRESS_TYPE_ANY/IP_ADDRESS_TYPE_IPV4/IP_ADDRESS_TYPE_IPV6.
remotePort	The remote port to bind to.
remoteAddress	Remote address to use.

Function

bool TCPIP_UDP_RemoteBind(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType,
UDP_PORT remotePort, IP_MULTI_ADDRESS* remoteAddress);

TCPIP_UDP_Close Function

Closes a UDP socket and frees the handle.

File

udp.h

C

```
bool TCPIP_UDP_Close(UDP_SOCKET hUDP);
```

Returns

- true - If the call succeeded
- false - If the call failed (no such socket)

Description

Closes a UDP socket and frees the handle. Call this function to release a socket and return it to the pool for use by future communications.

Remarks

Always close the socket when no longer in use. This will free the allocated resources, including the TX buffers.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	The socket handle to be released.

Function

bool TCPIP_UDP_Close(UDP_SOCKET hUDP)

TCPIP_UDP_OptionsGet Function

Allows getting the options for a socket such as current RX/TX buffer size, etc.

File

udp.h

C

```
bool TCPIP_UDP_OptionsGet(UDP_SOCKET hUDP, UDP_SOCKET_OPTION option, void* optParam);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Various options can be retrieved at the socket level. This function provides compatibility with BSD implementations.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket

Parameters

Parameters	Description
hUDP	socket to get options for
option	specific option to get
optParam	pointer to an area that will receive the option value; this is option dependent
the size of the area has to be large enough to accommodate the specific option	• UDP_OPTION_STRICT_PORT - pointer to boolean • UDP_OPTION_STRICT_NET - pointer to boolean • UDP_OPTION_STRICT_ADDRESS - pointer to boolean • UDP_OPTION_BROADCAST - pointer to UDP_SOCKET_BCAST_TYPE • UDP_OPTION_BUFFER_POOL - pointer to boolean • UDP_OPTION_TX_BUFF - pointer to a 16 bit value to receive bytes of the TX buffer • UDP_OPTION_TX_QUEUE_LIMIT - pointer to an 8 bit value to receive the TX queue limit • UDP_OPTION_RX_QUEUE_LIMIT - pointer to an 8 bit value to receive the RX queue limit • UDP_OPTION_RX_AUTO_ADVANCE - pointer to boolean

Function

bool TCPIP_UDP_OptionsGet(UDP_SOCKET hUDP, UDP_SOCKET_OPTION option, void* optParam);

TCPIP_UDP_OptionsSet Function

Allows setting options to a socket like adjust RX/TX buffer size, etc

File

udp.h

C

```
bool TCPIP_UDP_OptionsSet(UDP_SOCKET hUDP, UDP_SOCKET_OPTION option, void* optParam);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Various options can be set at the socket level. This function provides compatibility with BSD implementations.

Remarks

Changing the UDP_OPTION_BUFFER_POOL will discard the data in the current socket buffer

- UDP_OPTION_TX_BUFF - 16-bit value in bytes of the TX buffer

the UDP_OPTION_TX_BUFF will discard the data in the current socket buffer

- UDP_OPTION_TX_QUEUE_LIMIT - 8-bit value of the TX queue limit
- UDP_OPTION_RX_QUEUE_LIMIT - 8-bit value of the RX queue limit
- UDP_OPTION_RX_AUTO_ADVANCE - boolean enable/disable

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket

Parameters

Parameters	Description
hUDP	socket to set options for
option	specific option to be set

optParam	the option value; this is option dependent: • UDP_OPTION_STRICT_PORT - boolean enable/disable • UDP_OPTION_STRICT_NET - boolean enable/disable • UDP_OPTION_STRICT_ADDRESS - boolean enable/disable • UDP_OPTION_BROADCAST - UDP_SOCKET_BCAST_TYPE • UDP_OPTION_BUFFER_POOL - boolean enable/disable

Function

bool TCPIP_UDP_OptionsSet(UDP_SOCKET hUDP, UDP_SOCKET_OPTION option, void* optParam);

TCPIP_UDP_SocketInfoGet Function

Returns information about a selected UDP socket.

File

udp.h

C

```
bool TCPIP_UDP_SocketInfoGet(UDP_SOCKET hUDP, UDP_SOCKET_INFO* pInfo);
```

Returns

- true - if the call succeeded
- false - if no such socket or invalid pInfo

Description

This function will fill a user passed UDP_SOCKET_INFO structure with status of the selected socket.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket pInfo - valid address of a UDP_SOCKET_INFO structure

Parameters

Parameters	Description
UDP_SOCKET hUDP	Socket for which information is to be obtained
UDP_SOCKET_INFO* pInfo	pointer to UDP_SOCKET_INFO to receive socket information

Function

bool TCPIP_UDP_SocketInfoGet(UDP_SOCKET hUDP, UDP_SOCKET_INFO* pInfo)

TCPIP_UDP_SocketNetGet Function

Gets the network interface of an UDP socket

File

udp.h

C

```
TCPIP_NET_HANDLE TCPIP_UDP_SocketNetGet(UDP_SOCKET hUDP);
```

Returns

Handle of the interface that socket currently uses.

Description

This function returns the interface handle of an UDP socket

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	The UDP socket

Function

TCPIP_NET_HANDLE TCPIP_UDP_SocketNetGet(UDP_SOCKET hUDP)

TCPIP_UDP_SocketNetSet Function

Sets the network interface for an UDP socket

File

udp.h

C

```
bool TCPIP_UDP_SocketNetSet(UDP_SOCKET hUDP, TCPIP_NET_HANDLE hNet);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This function sets the network interface for an UDP socket

Remarks

A NULL hNet can be passed (0) so that the current socket network interface selection will be cleared

If the hNet != 0, it will enforce UDP_OPTION_STRICT_NET on the socket.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	The UDP socket
hNet	interface handle

Function

bool TCPIP_UDP_SocketNetSet(UDP_SOCKET hUDP, TCPIP_NET_HANDLE hNet)

TCPIP_UDP_TxOffsetSet Function

Moves the pointer within the TX buffer.

File

udp.h

C

```
bool TCPIP_UDP_TxOffsetSet(UDP_SOCKET hUDP, uint16_t wOffset, bool relative);
```

Returns

- true - if the offset was valid and the write pointer has been moved
- false - if the offset was not valid

Description

This function allows the write location within the TX buffer to be specified. Future calls to TCPIP_UDP_Put, TCPIP_UDP_ArrayPut, TCPIP_UDP_StringPut, etc will write data from the indicated location.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle
wOffset	Offset in the UDP packet data payload to move the write pointer.
relative	if true, the wOffset is added to the current write pointer. else the wOffset is from the beginning of the UDP buffer

Function

bool TCPIP_UDP_TxOffsetSet(UDP_SOCKET hUDP, uint16_t wOffset, bool relative)

TCPIP_UDP_SourceIPAddressSet Function

Sets the source IP address of a socket

File

udp.h

C

```
bool TCPIP_UDP_SourceIPAddressSet(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType, IP_MULTI_ADDRESS* localAddress);
```

Returns

- true - Indicates success
- false - Indicates failure:
- invalid socket
- invalid socket address type
- unspecified localAddress

Description

This function sets the IP source address, which allows changing the source P address dynamically.

Remarks

The call will fail if localAddress is 0. The source IP address will not be changed.

Preconditions

UDP initialized UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket localAddress - valid address pointer

Parameters

Parameters	Description
hUDP	the UDP socket
addType	Type of address: IPv4/IPv6
localAddress	pointer to an address to use

Function

bool TCPIP_UDP_SourceIPAddressSet(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType,
IP_MULTI_ADDRESS* localAddress)

TCPIP_UDP_BcastIPV4AddressSet Function

Sets the broadcast IP address of a socket Allows an UDP socket to send broadcasts.

File

udp.h

C

```
bool TCPIP_UDP_BcastIPV4AddressSet(UDP_SOCKET hUDP, UDP_SOCKET_BCAST_TYPE bcastType, TCPIP_NET_HANDLE hNet);
```

Returns

- true - Indicates success
- false - Indicates failure:

- invalid socket
- invalid socket address type
- a broadcast for the specified interface could not be obtained
- invalid broadcast type specified

Description

It sets the broadcast address for the socket

Remarks

This function allows changing of the destination IPv4 address dynamically.

However, the call will fail if the socket was previously set to broadcast using the TCPIP_UDP_OptionsSet call. TCPIP_UDP_OptionsSet takes precedence.

Preconditions

UDP initialized UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket

Parameters

Parameters	Description
hUDP	the UDP socket
bcastType	Type of broadcast
hNet	handle of an interface to use for the network directed broadcast Not used for network limited broadcast

Function

bool TCPIP_UDP_BcastIPV4AddressSet(UDP_SOCKET hUDP, UDP_SOCKET_BCAST_TYPE bcastType,
TCPIP_NET_HANDLE hNet)

TCPIP_UDP_DestinationIPAddressSet Function

Sets the destination IP address of a socket

File

udp.h

C

```
bool TCPIP_UDP_DestinationIPAddressSet(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType, IP_MULTI_ADDRESS*
remoteAddress);
```

Returns

- true - Indicates success
- false - Indicates failure:
- invalid socket
- invalid socket address type
- socket is of broadcast type

Description

- It sets the IP destination address This allows changing the IP destination address dynamically.

Remarks

The call will fail if the socket was previously set to broadcast using the TCPIP_UDP_OptionsSet call. TCPIP_UDP_OptionsSet takes precedence.

The call will fail if remoteAddress is 0. The destination IP address will not be changed.

Preconditions

UDP initialized UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket
remoteAddress - valid address pointer

Parameters

Parameters	Description
hUDP	the UDP socket
addType	Type of address: IPv4/IPv6
remoteAddress	pointer to an address to use

Function

bool TCPIP_UDP_DestinationIPAddressSet(UDP_SOCKET hUDP, IP_ADDRESS_TYPE addType,
IP_MULTI_ADDRESS* remoteAddress)

TCPIP_UDP_DestinationPortSet Function

Sets the destination port of a socket

File

udp.h

C

```
bool TCPIP_UDP_DestinationPortSet(UDP_SOCKET s, UDP_PORT remotePort);
```

Returns

- true - Indicates success
- false - Indicates an invalid socket

Description

This function sets the destination port, which allows changing the destination port dynamically.

Remarks

The destination remote port will always be changed, even if remotePort == 0.

It will not change the UDP_OPTION_STRICT_PORT on the socket.

Preconditions

UDP initialized UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	the UDP socket
remotePort	destination port to use

Function

bool TCPIP_UDP_DestinationPortSet(UDP_SOCKET s, UDP_PORT remotePort)

TCPIP_UDP_Disconnect Function

Disconnects a UDP socket and re-initializes it.

File

udp.h

C

```
bool TCPIP_UDP_Disconnect(UDP_SOCKET hUDP, bool flushRxQueue);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

Disconnects a UDP socket and re-initializes it. Call this function to return the socket to its initial open state and to use it for future communication.

This function is meant especially for server sockets that could listen on multiple interfaces and on both IPv4 and IPv6 networks.

When a server socket received an inbound IPv4 connection it will be bound to IPv4 connections until it's closed or disconnected. Same is true for IPv6 connections.

Remarks

The call will try to maintain as much as possible from the socket state.

This will free the allocated TX buffers if the socket was opened with IP_ADDRESS_TYPE_ANY.

All the pending RX packets will be cleared when flushRxQueue is set. Otherwise the packets will be kept and will be available for next read

operations.

Note that if the RX pending packets exist and are not flushed, the socket will be bound immediately to the connection corresponding to the pending packet in the queue.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen()/TCPIP_UDP_ClientOpen()(). hUDP - valid socket

Parameters

Parameters	Description
hUDP	The socket handle to be disconnected.
flushRxQueue	boolean to flush the pending RX queue

Function

bool TCPIP_UDP_Disconnect(UDP_SOCKET hUDP, bool flushRxQueue)

TCPIP_UDP_SignalHandlerDeregister Function

Deregisters a previously registered UDP socket signal handler.

File

udp.h

C

```
bool TCPIP_UDP_SignalHandlerDeregister(UDP_SOCKET s, TCPIP_UDP_SIGNAL_HANDLE hSig);
```

Returns

- true - if the call succeeds
- false - if no such handler is registered

Description

Deregisters the UDP socket signal handler.

Preconditions

hSig valid UDP signal handle.

Parameters

Parameters	Description
s	The UDP socket
hSig	A handle returned by a previous call to TCPIP_UDP_SignalHandlerRegister.

Function

TCPIP_UDP_SignalHandlerDeregister(UDP_SOCKET s, TCPIP_UDP_SIGNAL_HANDLE hSig)

TCPIP_UDP_SignalHandlerRegister Function

Registers a UDP socket signal handler.

File

udp.h

C

```
TCPIP_UDP_SIGNAL_HANDLE TCPIP_UDP_SignalHandlerRegister(UDP_SOCKET s, TCPIP_UDP_SIGNAL_TYPE sigMask,
TCPIP_UDP_SIGNAL_FUNCTION handler, const void* hParam);
```

Returns

Returns a valid handle if the call succeeds, or a null handle if the call failed (null handler, no such socket, existent handler).

Description

This function registers a UDP socket signal handler. The UDP module will call the registered handler when a UDP signal (TCPIP_UDP_SIGNAL_TYPE) occurs.

Remarks

Only one signal handler per socket is supported. A new handler does not override the existent one. Instead TCPIP_UDP_SignalHandlerDeregister has to be explicitly called.

The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

The hParam is passed by the client but is currently not used and should be 0.

For multi-threaded systems the TCP/IP packet dispatch does not occur on the user thread. The signal handler will be called on a different thread context. It is essential that this handler is non blocking and really fast.

For multi-threaded systems, once set, it is not recommended to change the signal handler at run time. Synchronization between user threads and packet dispatch threads could be difficult. If really need to be changed, make sure that the old handler could still be called and it should be valid until the new one is taken into account. TCPIP_UDP_SignalHandlerDeregister needs to be called before registering another handler.

Preconditions

UDP valid socket.

Parameters

Parameters	Description
s	The UDP socket
sigMask	mask of signals to be reported
handler	signal handler to be called when a UDP signal occurs.
hParam	Parameter to be used in the handler call. This is user supplied and is not used by the UDP module. Currently not used and it should be null.

Function

TCPIP_UDP_SignalHandlerRegister(UDP_SOCKET s, TCPIP_UDP_SIGNAL_TYPE sigMask,
TCPIP_UDP_SIGNAL_FUNCTION handler, const void* hParam)

TCPIP_UDP_Task Function

Standard TCP/IP stack module task function.

File

udp.h

C

```
void TCPIP_UDP_Task();
```

Returns

None.

Description

This function performs UDP module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The UDP module should have been initialized.

Function

void TCPIP_UDP_Task(void)

TCPIP_UDP_SocketsNumberGet Function

Returns the number of existent UDP sockets.

File

udp.h

C

```
int TCPIP_UDP_SocketsNumberGet();
```

Returns

The number of UDP sockets

Description

This function returns the number of created UDP sockets. This is the maximum number of sockets that can be opened at any moment as it's been passed as parameter when UDP module was created.

Preconditions

UDP module properly initialized

Function

int TCPIP_UDP_SocketsNumberGet(void)

b) Transmit Data Transfer Functions

TCPIP_UDP_PutIsReady Function

Determines how many bytes can be written to the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_PutIsReady(UDP_SOCKET hUDP);
```

Returns

The number of bytes that can be written to this socket.

Description

This function determines how many bytes can be written to the specified UDP socket.

Remarks

If the current socket TX buffer is in use (in traffic), this function will allocate a new TX buffer. Otherwise the current TX buffer will be used.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle

Function

uint16_t TCPIP_UDP_PutIsReady(UDP_SOCKET hUDP)

TCPIP_UDP_TxPutIsReady Function

Determines how many bytes can be written to the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_TxPutIsReady(UDP_SOCKET hUDP, unsigned short count);
```

Returns

The number of bytes that can be written to this socket.

Description

This function returns the number of bytes that can be written to the specified UDP socket.

Remarks

The function won't increase the size of the UDP TX buffer. If this is needed use TCPIP_UDP_OptionsSet. The count variable is not used.

The function is similar to the TCPIP_UDP_PutIsReady and maintained for backward compatibility.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle
count	Number of bytes requested

Function

uint16_t TCPIP_UDP_TxPutIsReady(UDP_SOCKET hUDP, unsigned short count)

TCPIP_UDP_ArrayPut Function

Writes an array of bytes to the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_ArrayPut(UDP_SOCKET hUDP, const uint8_t * cData, uint16_t wDataLen);
```

Returns

The number of bytes successfully placed in the UDP transmit buffer. If this value is less than wDataLen, then the buffer became full and the input was truncated.

Description

This function writes an array of bytes to the UDP socket, while incrementing the socket write pointer. TCPIP_UDP_PutIsReady could be used before calling this function to verify that there is room in the socket buffer.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket cData - valid pointer

Parameters

Parameters	Description
hUDP	UDP socket handle
cData	The array to write to the socket.
wDataLen	Number of bytes from cData to be written.

Function

uint16_t TCPIP_UDP_ArrayPut(UDP_SOCKET hUDP, const uint8_t *cData, uint16_t wDataLen)

TCPIP_UDP_StringPut Function

Writes a null-terminated string to the UDP socket.

File

udp.h

C

```
const uint8_t* TCPIP_UDP_StringPut(UDP_SOCKET hUDP, const uint8_t * strData);
```

Returns

A pointer to the byte following the last byte written. Note that this is different than the TCPIP_UDP_ArrayPut functions. If this pointer does not dereference to a NULL byte, then the buffer became full and the input data was truncated.

Description

This function writes a null-terminated string to the UDP socket while incrementing the socket write pointer. TCPIP_UDP_PutIsReady could be used before calling this function to verify that there is room in the socket buffer.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket strData - valid pointer

Parameters

Parameters	Description
hUDP	UDP socket handle
strData	Pointer to the string to be written to the socket.

Function

uint8_t* TCPIP_UDP_StringPut(UDP_SOCKET hUDP, const uint8_t *strData)

TCPIP_UDP_Put Function

Writes a byte to the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_Put(UDP_SOCKET hUDP, uint8_t v);
```

Description

This function writes a single byte to the UDP socket, while incrementing the socket write pointer. TCPIP_UDP_PutIsReady could be used before calling this function to verify that there is room in the socket buffer.

Remarks

This function is very inefficient and its use is discouraged. A buffered approach (TCPIP_UDP_ArrayPut) is preferred.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle
v	The byte to be loaded into the transmit buffer.

Function

uint16_t TCPIP_UDP_Put(UDP_SOCKET hUDP, uint8_t v)

TCPIP_UDP_TxCountGet Function

Returns the amount of bytes written into the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_TxCountGet(UDP_SOCKET hUDP);
```

Description

This function returns the amount of bytes written into the UDP socket, i.e. the current position of the write pointer.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle

Function

uint16_t TCPIP_UDP_TxCountGet(UDP_SOCKET hUDP)

TCPIP_UDP_Flush Function

Transmits all pending data in a UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_Flush(UDP_SOCKET hUDP);
```

Returns

The number of bytes that currently were in the socket TX buffer and have been flushed. Otherwise, 0 if the packet could not be transmitted:

- invalid socket
- invalid remote address
- no route to the remote host could be found

Description

This function builds a UDP packet with the pending TX data and marks it for transmission over the network interface. There is no UDP state machine to send the socket data automatically. The UDP socket client must call this function to actually send the data over the network.

Remarks

Note that a UDP socket must be flushed to send data over the network. There is no UDP state machine (auto transmit) for UDP sockets.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle

Function

uint16_t TCPIP_UDP_Flush(UDP_SOCKET hUDP)

c) Receive Data Transfer Functions

TCPIP_UDP_GetIsReady Function

Determines how many bytes can be read from the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_GetIsReady(UDP_SOCKET hUDP);
```

Returns

The number of bytes that can be read from this socket.

Description

This function will return the number of bytes that are available in the specified UDP socket RX buffer.

Remarks

The UDP socket queues incoming RX packets in an internal queue.

If currently there is no RX packet processed (as a result of retrieving all available bytes with TCPIP_UDP_ArrayGet, for example), this call will advance the RX packet to be processed to the next queued packet.

If a RX packet is currently processed, the call will return the number of bytes left to be read from this packet.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle

Function

uint16_t TCPIP_UDP_GetIsReady(UDP_SOCKET hUDP)

TCPIP_UDP_ArrayGet Function

Reads an array of bytes from the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_ArrayGet(UDP_SOCKET hUDP, uint8_t * cData, uint16_t wDataLen);
```

Returns

The number of bytes successfully read from the UDP buffer. If this value is less than wDataLen, then the buffer was emptied and no more data is available.

Description

This function reads an array of bytes from the UDP socket, while adjusting the current read pointer and decrementing the remaining bytes available. TCPIP_UDP_GetIsReady should be used before calling this function to get the number of the available bytes in the socket.

Remarks

For discarding a number of bytes in the RX buffer the TCPIP_UDP_RxOffsetSet() can also be used.

The UDP socket queues incoming RX packets in an internal queue. This call will try to retrieve the bytes from the current processing packet but it won't advance the processed packet. TCPIP_UDP_GetIsReady should be called to advance the processed RX packet.

TCPIP_UDP_Discard should be called when done processing the current RX packet.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle
cData	The buffer to receive the bytes being read. If NULL, the bytes are simply discarded
wDataLen	Number of bytes to be read from the socket.

Function

uint16_t TCPIP_UDP_ArrayGet(UDP_SOCKET hUDP, uint8_t *cData, uint16_t wDataLen)

TCPIP_UDP_Get Function

Reads a byte from the UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_Get(UDP_SOCKET hUDP, uint8_t * v);
```

Description

This function reads a single byte from the UDP socket, while decrementing the remaining RX buffer length. TCPIP_UDP_GetIsReady should be used before calling this function to get the number of bytes available in the socket.

Remarks

This function is very inefficient and its usage is discouraged. A buffered approach (TCPIP_UDP_ArrayGet) is preferred.

See the previous notes for TCPIP_UDP_ArrayGet function.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	socket handle
v	The buffer to receive the data being read.

Function

uint16_t TCPIP_UDP_Get(UDP_SOCKET hUDP, uint8_t *v)

TCPIP_UDP_RxOffsetSet Function

Moves the read pointer within the socket RX buffer.

File

udp.h

C

```
void TCPIP_UDP_RxOffsetSet(UDP_SOCKET hUDP, uint16_t rOffset);
```

Returns

None.

Description

This function allows the user to specify the read location within the socket RX buffer. Future calls to TCPIP_UDP_Get and TCPIP_UDP_ArrayGet will read data from the indicated location forward.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	UDP socket handle
wOffset	Offset from beginning of UDP packet data payload to place the read pointer.

Function

void TCPIP_UDP_RxOffsetSet(UDP_SOCKET hUDP, uint16_t wOffset)

TCPIP_UDP_Discard Function

Discards any remaining RX data from a UDP socket.

File

udp.h

C

```
uint16_t TCPIP_UDP_Discard(UDP_SOCKET hUDP);
```

Returns

Number of discarded bytes, if any.

Description

This function discards any remaining received data in the UDP socket.

Remarks

The UDP socket queues incoming RX packets in an internal queue.

This call will discard the remaining bytes (if any) in the current RX packet and will advance the RX packet to be processed to the next queued packet.

This function should be normally called after retrieving the available bytes with TCPIP_UDP_ArrayGet.

When data available, calling it repeatedly will discard one pending RX packet at a time.

Note that a call to TCPIP_UDP_Discard is not needed if all bytes are retrieved with TCPIP_UDP_ArrayGet and then TCPIP_UDP_GetIsReady is called.

Preconditions

UDP socket should have been opened with TCPIP_UDP_ServerOpen/TCPIP_UDP_ClientOpen. hUDP - valid socket

Parameters

Parameters	Description
hUDP	socket handle

Function

uint16_t TCPIP_UDP_Discard(UDP_SOCKET hUDP)

d) Data Types and Constants

INVALID_UDP_SOCKET Macro

File

udp.h

C

```
#define INVALID_UDP_SOCKET (-1)        // Indicates a UDP socket that is not valid
```

Description

Indicates a UDP socket that is not valid

UDP_PORT Type

File

udp.h

C

```
typedef uint16_t UDP_PORT;
```

Description

Defines a UDP Port Number

UDP_SOCKET Type

File

udp.h

C

```
typedef int16_t UDP_SOCKET;
```

Description

Provides a handle to a UDP Socket

UDP_SOCKET_BCAST_TYPE Enumeration

File

udp.h

C

```
typedef enum {
  UDP_BCAST_NONE,
  UDP_BCAST_NETWORK_LIMITED,
  UDP_BCAST_NETWORK_DIRECTED
} UDP_SOCKET_BCAST_TYPE;
```

Members

Members	Description
UDP_BCAST_NONE	no broadcast
UDP_BCAST_NETWORK_LIMITED	network limited broadcast
UDP_BCAST_NETWORK_DIRECTED	network directed broadcast

Description

This is type UDP_SOCKET_BCAST_TYPE.

UDP_SOCKET_INFO Structure

File

udp.h

C

```
typedef struct {
  IP_ADDRESS_TYPE addressType;
  IP_MULTI_ADDRESS remoteIPaddress;
  IP_MULTI_ADDRESS localIPaddress;
  IP_MULTI_ADDRESS sourceIPaddress;
  IP_MULTI_ADDRESS destIPaddress;
  UDP_PORT remotePort;
  UDP_PORT localPort;
  TCPIP_NET_HANDLE hNet;
  uint16_t rxQueueSize;
  uint16_t txSize;
} UDP_SOCKET_INFO;
```

Members

Members	Description
IP_ADDRESS_TYPE addressType;	address type of the socket
IP_MULTI_ADDRESS remoteIPaddress;	current socket destination address
IP_MULTI_ADDRESS localIPaddress;	current socket source address
IP_MULTI_ADDRESS sourceIPaddress;	source address of the last packet
IP_MULTI_ADDRESS destIPaddress;	destination address of the last packet
UDP_PORT remotePort;	Port number associated with remote node
UDP_PORT localPort;	local port number
TCPIP_NET_HANDLE hNet;	associated interface
uint16_t rxQueueSize;	packets waiting in the rx queue
uint16_t txSize;	tx buffer size

Description

Information about a socket

UDP_SOCKET_OPTION Enumeration

File

udp.h

C

```
typedef enum {
  UDP_OPTION_STRICT_PORT,
  UDP_OPTION_STRICT_NET,
  UDP_OPTION_STRICT_ADDRESS,
  UDP_OPTION_BROADCAST,
  UDP_OPTION_BUFFER_POOL,
  UDP_OPTION_TX_BUFF,
  UDP_OPTION_TX_QUEUE_LIMIT,
  UDP_OPTION_RX_QUEUE_LIMIT,
  UDP_OPTION_RX_AUTO_ADVANCE
} UDP_SOCKET_OPTION;
```

Members

Members	Description
UDP_OPTION_STRICT_PORT	When connection is done the socket stores the remote host local port. If option is enabled the remote host local port is always checked to match the initial one. If disabled the remote host local port is not checked. Disabled by default on a socket server. Enabled by default on a client socket.
UDP_OPTION_STRICT_NET	When connection is done the socket stores the network interface the connection occurred on. If option is enabled the socket accepts data only from the interface that matches the initial connection. If disabled the socket receives data from any remote host regardless of the interface on which the packet arrived. Disabled by default on a socket server. Enabled by default on a client socket.
UDP_OPTION_STRICT_ADDRESS	When connection is done the socket stores the address of the remote host. If option is enabled the socket accepts data only from the host with address that matches the initial connection. If disabled the socket receives data from any remote host regardless of the address of that host. Disabled by default on a socket server. Enabled by default on a client socket.
UDP_OPTION_BROADCAST	Enables the Broadcast transmission by the socket
UDP_OPTION_BUFFER_POOL	Enables the socket to use the private UDP buffers pool. The size of the TX buffer has to be less than the size of the buffers in the pool.
UDP_OPTION_TX_BUFF	Request different TX buffer size. Has to call TCPIP_UDP_OptionsGet to see the exact space allocated.
UDP_OPTION_TX_QUEUE_LIMIT	Sets the maximum number of packets that can be queued/allocated by an UDP socket at a certain time. For sockets that need to transmit a lot of data (Iperf client, for example), especially on slow connections this limit prevents running out of memory because the MAC transfer cannot keep up with the UDP packet allocation rate imposed by the application. Adjust depending on the size of the UDP TX buffer, the connection speed and the amount of memory available to the stack.
UDP_OPTION_RX_QUEUE_LIMIT	Sets the maximum number of RX packets that can be queued by an UDP socket at a certain time. Setting this value to 0 will disable the socket receive functionality
UDP_OPTION_RX_AUTO_ADVANCE	Sets the RX auto advance option for a socket. The option is off by default If set, the option will automatically advance the socket to the next awaiting RX packet when a call to TCPIP_UDP_ArrayGet is made and no more data is available from the current packet. Setting this option forces the socket user to correctly monitor the number of bytes available and issue a TCPIP_UDP_Discard only when there's bytes left in the current packet. However sequential calls to TCPIP_UDP_ArrayGet can be made without the need to insert calls to TCPIP_UDP_Discard If cleared (default case) the socket user can safely issue a TCPIP_UDP_Discard even after calling TCPIP_UDP_ArrayGet to read all the bytes from the current RX packet Note that TCPIP_UDP_GetIsReady will always advance the current RX packet when no more data is available in the current packet. Even when the option is cleared a call to TCPIP_UDP_Discard is still not needed if all bytes are retrieved with TCPIP_UDP_ArrayGet and then TCPIP_UDP_GetIsReady is called.

Description

UDP socket options

TCPIP_UDP_MODULE_CONFIG Structure

UDP module run time configuration/initialization data.

File

udp.h

C

```
typedef struct {
  uint16_t nSockets;
  uint16_t sktTxBuffSize;
  uint16_t poolBuffers;
  uint16_t poolBufferSize;
} TCPIP_UDP_MODULE_CONFIG;
```

Members

Members	Description
uint16_t nSockets;	number of sockets to be created
uint16_t sktTxBuffSize;	default size of the socket TX buffer
uint16_t poolBuffers;	number of buffers in the pool; 0 if none
uint16_t poolBufferSize;	size of the buffers in the pool; all equal

Description

Structure: TCPIP_UDP_MODULE_CONFIG

UDP module configuration/initialization

TCPIP_UDP_SIGNAL_FUNCTION Type

UDP Signal Handler.

File

udp.h

C

```
typedef void (* TCPIP_UDP_SIGNAL_FUNCTION)(UDP_SOCKET hUDP, TCPIP_NET_HANDLE hNet, TCPIP_UDP_SIGNAL_TYPE
sigType, const void* param);
```

Description

Type: TCPIP_UDP_SIGNAL_FUNCTION

Prototype of a UDP signal handler. Socket user can register a handler for the UDP socket. Once an UDP signals occurs the registered handler will be called.

Remarks

The handler has to be short and fast. It is meant for setting an event flag, not for lengthy processing!

Parameters

Parameters	Description
hUDP	UDP socket to be used
hNet	the network interface on which the event has occurred
sigType	type of UDP signal that has occurred
param	additional parameter that can has been specified at the handler registration call Currently not used and it will be null.

TCPIP_UDP_SIGNAL_HANDLE Type

UDP socket handle.

File

udp.h

C

```
typedef const void* TCPIP_UDP_SIGNAL_HANDLE;
```

Description

Type: TCPIP_UDP_SIGNAL_HANDLE

A handle that a socket client can use after the signal handler has been registered.

TCPIP_UDP_SIGNAL_TYPE Enumeration

UDP run-time signal types.

File

udp.h

C

```
typedef enum {
  TCPIP_UDP_SIGNAL_TX_DONE = 0x0001,
  TCPIP_UDP_SIGNAL_RX_DATA = 0x0100
} TCPIP_UDP_SIGNAL_TYPE;
```

Members

Members	Description
TCPIP_UDP_SIGNAL_TX_DONE = 0x0001	A data packet was successfully transmitted on the interface There may be available buffer space to send new data
TCPIP_UDP_SIGNAL_RX_DATA = 0x0100	A data packet was successfully received and there is data available for this socket

Description

Enumeration: TCPIP_UDP_SIGNAL_TYPE

Description of the signals/events that a UDP socket can generate.

Remarks

These signals are used in the socket event handling notification functions. Currently a UDP notification doesn't set multiple flags as the TX and RX events are handled separately.

The signals are 16 bits wide.

Files

Files

Name	Description
udp.h	UDP is a standard transport layer protocol described in RFC 768. It provides fast but unreliable data-gram based transfers over networks, and forms the foundation SNTP, SNMP, DNS, and many other protocol standards
udp_config.h	UDP Configuration file

Description

This section lists the source and header files used by the library.

udp.h

UDP is a standard transport layer protocol described in RFC 768. It provides fast but unreliable data-gram based transfers over networks, and forms the foundation SNTP, SNMP, DNS, and many other protocol standards

Enumerations

	Name	Description
	TCPIP_UDP_SIGNAL_TYPE	UDP run-time signal types.
	UDP_SOCKET_BCAST_TYPE	This is type UDP_SOCKET_BCAST_TYPE.
	UDP_SOCKET_OPTION	UDP socket options

Functions

	Name	Description
	TCPIP_UDP_ArrayGet	Reads an array of bytes from the UDP socket.
	TCPIP_UDP_ArrayPut	Writes an array of bytes to the UDP socket.
	TCPIP_UDP_BcastIPV4AddressSet	Sets the broadcast IP address of a socket Allows an UDP socket to send broadcasts.
	TCPIP_UDP_Bind	Bind a socket to an address type, local port and address. This function is meant primarily for client sockets.
	TCPIP_UDP_ClientOpen	Opens a UDP socket as a client.
	TCPIP_UDP_Close	Closes a UDP socket and frees the handle.
	TCPIP_UDP_DestinationIPAddressSet	Sets the destination IP address of a socket
	TCPIP_UDP_DestinationPortSet	Sets the destination port of a socket
	TCPIP_UDP_Discard	Discards any remaining RX data from a UDP socket.
	TCPIP_UDP_Disconnect	Disconnects a UDP socket and re-initializes it.
	TCPIP_UDP_Flush	Transmits all pending data in a UDP socket.
	TCPIP_UDP_Get	Reads a byte from the UDP socket.
	TCPIP_UDP_GetIsReady	Determines how many bytes can be read from the UDP socket.
	TCPIP_UDP_IsConnected	Determines if a socket has an established connection.
	TCPIP_UDP_OptionsGet	Allows getting the options for a socket such as current RX/TX buffer size, etc.
	TCPIP_UDP_OptionsSet	Allows setting options to a socket like adjust RX/TX buffer size, etc
	TCPIP_UDP_Put	Writes a byte to the UDP socket.
	TCPIP_UDP_PutIsReady	Determines how many bytes can be written to the UDP socket.
	TCPIP_UDP_RemoteBind	Bind a socket to an address type, remote port and address. This function is meant primarily for server sockets.
	TCPIP_UDP_RxOffsetSet	Moves the read pointer within the socket RX buffer.
	TCPIP_UDP_ServerOpen	Opens a UDP socket as a server.
	TCPIP_UDP_SignalHandlerDeregister	Deregisters a previously registered UDP socket signal handler.
	TCPIP_UDP_SignalHandlerRegister	Registers a UDP socket signal handler.
	TCPIP_UDP_SocketInfoGet	Returns information about a selected UDP socket.
	TCPIP_UDP_SocketNetGet	Gets the network interface of an UDP socket
	TCPIP_UDP_SocketNetSet	Sets the network interface for an UDP socket
	TCPIP_UDP_SocketsNumberGet	Returns the number of existent UDP sockets.
	TCPIP_UDP_SourceIPAddressSet	Sets the source IP address of a socket
	TCPIP_UDP_StringPut	Writes a null-terminated string to the UDP socket.
	TCPIP_UDP_Task	Standard TCP/IP stack module task function.
	TCPIP_UDP_TxCountGet	Returns the amount of bytes written into the UDP socket.
	TCPIP_UDP_TxOffsetSet	Moves the pointer within the TX buffer.
	TCPIP_UDP_TxPutIsReady	Determines how many bytes can be written to the UDP socket.

Macros

	Name	Description
	INVALID_UDP_SOCKET	Indicates a UDP socket that is not valid
	TCPIP_UDP_IsOpened	Determines if a socket was opened.

Structures

	Name	Description
	TCPIP_UDP_MODULE_CONFIG	UDP module run time configuration/initialization data.
	UDP_SOCKET_INFO	Information about a socket

Types

	Name	Description
	TCPIP_UDP_SIGNAL_FUNCTION	UDP Signal Handler.
	TCPIP_UDP_SIGNAL_HANDLE	UDP socket handle.
	UDP_PORT	Defines a UDP Port Number
	UDP_SOCKET	Provides a handle to a UDP Socket

Description

UDP Module Definitions for the Microchip TCP/IP Stack

UDP is a standard transport layer protocol described in RFC 768.

File Name

udp.h

Company

Microchip Technology Inc.

udp_config.h

UDP Configuration file

Macros

	Name	Description
	TCPIP_UDP_MAX_SOCKETS	Maximum number of UDP sockets that can be opened simultaneously These sockets will be created when the module is initialized.
	TCPIP_UDP_SOCKET_DEFAULT_RX_QUEUE_LIMIT	The maximum number of RX packets that can be queued by an UDP socket at a certain time. Note that UDP sockets do not use their own RX buffers but instead use the network driver supplied packets and a timely processing is critical to avoid packet memory starvation for the whole stack. This symbol sets the maximum number of UDP buffers/packets that can be queued for a UDP socket at a certain time. Once this limit is reached further incoming packets are silently discarded. Adjust depending on the number of RX buffers that are available for the stack and the... more
	TCPIP_UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT	The maximum number of TX packets that can be queued by an UDP socket at a certain time. For sockets that need to transfer a lot of data (Iperf, for example), especially on slow connections this limit prevents running out of memory because the MAC/PHY transfer cannot keep up with the UDP packet allocation rate imposed by the application. Adjust depending on the TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE, the connection speed and the amount of memory available to the stack.
	TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE	Default socket TX buffer size. Note that this setting affects all UDP sockets that are created and, together with TCPIP_UDP_MAX_SOCKETS, has a great impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting). When large TX bufferrs are needed, probably a dynamic, per socket approach, is a better choice (see TCPIP_UDP_OptionsSet function).
	TCPIP_UDP_SOCKET_POOL_BUFFER_SIZE	Size of the buffers in the UDP pool. Any UDP socket that is enabled to use the pool and has the TX size <= than this size can use a buffer from the pool. Note that this setting, together with TCPIP_UDP_SOCKET_POOL_BUFFERS, has impact on the heap size that's used by the stack (see TCPIP_STACK_DRAM_SIZE setting).
	TCPIP_UDP_SOCKET_POOL_BUFFERS	Number of buffers in the private UDP pool. These are preallocated buffers that are to be used by UDP sockets only. This improves the UDP socket throughput and is meant only for UDP sockets that have to sustain high TX traffic rate. However, this memory is not returned to the stack heap, it always belongs to UDP. A socket needs to have an option set in order to use the buffers pool (see the UDPSetOptions()). Use 0 to disable the feature.
	TCPIP_UDP_USE_POOL_BUFFERS	enable the build of the pre-allocated pool buffers option
	TCPIP_UDP_USE_RX_CHECKSUM	Check incoming packets to have proper checksum.
	TCPIP_UDP_USE_TX_CHECKSUM	Calculate and transmit a checksum when sending data. Checksum is not mandatory for UDP packets but is highly recommended. This will affect the UDP TX performance.

Description

User Datagram Protocol (UDP) Configuration file

This file contains the UDP module configuration options

File Name

udp_config.h

Company

Microchip Technology Inc.

Zeroconf Module

This section describes the TCP/IP Stack Library Zeroconf module.

Introduction

TCP/IP Stack Library Zero Configuration (Zeroconf) Module for Microchip Microcontrollers

This library provides the API of the Zeroconf module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP Stack.

Description

Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network. It also provides for a more human-like naming convention, instead of relying on IP addresses alone. Zeroconf also goes by the names Bonjour (Apple) and Avahi (Linux), and is an IETF standard.

Using the Library

This topic describes the basic architecture of the Zeroconf TCP/IP Stack Library and provides information and examples on its use.

Description

Interface Header Files: `tcpip_helper.h`, `zero_conf_link_local.h`, and `zero_conf_multicast_dns.h`

The interface to the Zeroconf TCP/IP Stack library is defined in the `tcpip_helper.h`, `zero_conf_link_local.h`, and `zero_conf_multicast_dns.h` header files. This file is included by the `tcpip.h` file. Any C language source (`.c`) file that uses the Zeroconf TCP/IP Stack library should include `tcpip.h`.

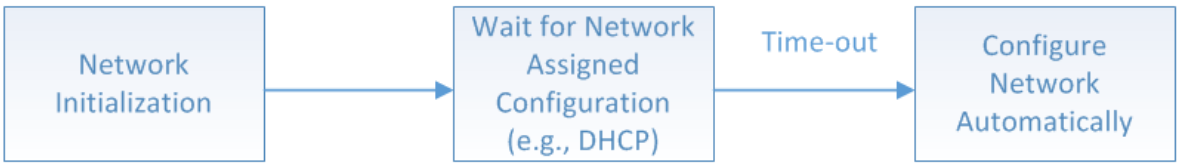
Please refer to the What is MPLAB Harmony? section for how the TCP/IP Stack interacts with the framework.

Abstraction Model

This library provides the API of the Zeroconf TCP/IP Module that is available on the Microchip family of microcontrollers with a convenient C language interface. It is a module that belongs to the TCP/IP stack.

Description

Zeroconf Software Abstraction Model



Bonjour Service Model

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Zeroconf module.

Library Interface Section	Description
Multicast DNS Functions	This section provides multicast DNS interface routines to Zeroconf
Link Local Functions	This section provides Link Local interface routines to Zeroconf
Data Types and Constants	This section provides various definitions describing this API

Core Functionality

This section provides information on core functionality.

Enabling

Zeroconf can be enabled in the MPLAB Harmony Configurator, or by setting the following two defines in `system_config.h` file:

- TCPIP_STACK_USE_ZEROCONF_LINK_LOCAL
- TCPIP_STACK_USE_ZEROCONF_MDNS_SD

Link Local (Zeroconf)

The first component of Zeroconf is the ability to self-assign an IP address to each member of a network. Normally, a DHCP server would handle such situations. However, in cases where no DHCP server exists, Zeroconf enabled devices negotiate unique IP addresses amongst themselves.

mDNS

The second component of Zeroconf is the ability to self-assign human-readable hostnames for themselves. Multicast DNS provides a local network the ability to have the features of a DNS server. Users can use easily remembered hostnames to accesses the devices on the network. In the event that devices elect to use the same hostname, as in the IP address resolution, each of the devices will auto-negotiate new names for themselves (usually by appending a number to the end of the name).

Service Discovery

The last component of Zeroconf is service discovery. All Zeroconf devices can broadcast what services they provide. For instance, a printer can broadcast that it has printing services available. A thermostat can broadcast that it has an HVAC control service. Other interested parties on the network who are looking for certain services can then see a list of devices that have the capability of providing the service, and connect directly to it. This further eliminates the need to know whether something exists on a network (and what its IP or hostname is). As an end-user, all you would need to do is query the network if a certain service exists, and easily connect to it.

Demonstration

The demonstration, when enabled, shows all three items working together. Each development kit in the network assumes the hostname of MCHPBOARD-x.local, where 'x' is an number incrementing from 1 (only in the case where multiple kits are programmed for the network). Each board will broadcast its service, which is the DemoWebServer.

Zeroconf Enabled Environments

All Apple products have Zeroconf enabled by default. On Windows, you'll need to download the Safari web browser, and during the install, enable support for Bonjour. Note that in the Safari browser, you can browse and see a list of all Bonjour enabled devices, and click through to them automatically.

Configuring the Library

The configuration of the Zeroconf TCP/IP Stack Library is based on the file `tcpip_config.h`.

This header file contains the configuration selection for the Zeroconf TCP/IP Stack Library. Based on the selections made, the Zeroconf TCP/IP Stack Library may support the selected features. These configuration settings will apply to all instances of the Zeroconf TCP/IP Stack Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available in the Zeroconf module of the TCP/IP Stack Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/tcpip`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/tcpip.h`	Header file that includes all of the TCP/IP modules.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/zero_conf_helper.c`	Zeroconf helper function implementation file.
`/src/zero_conf_link_local.c`	Zeroconf local link configuration implementation file.
`/src/zero_conf_multicast_dns.c`	Service advertisement and Multicast DNS implementation file.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	No optional files are available for this library.

Module Dependencies

The Zeroconf module depends on the following modules:

- TCP/IP Stack Library
- ARP Module
- UDP Module

Library Interface

a) Multicast DNS Functions

	Name	Description
	TCPIP_MDNS_ServiceDeregister	DNS Service Discovery function for end-user to deregister a service advertisement, which was previously registered with the TCPIP_MDNS_ServiceRegister function.
	TCPIP_MDNS_ServiceRegister	DNS Service Discovery function for end-users to register a service advertisement. The service is associated with all interfaces.
	TCPIP_MDNS_ServiceUpdate	DNS-Service Discovery function for end-user to update the service advertisement, which was previously registered with TCPIP_MDNS_ServiceRegister.
	TCPIP_MDNS_Task	Standard TCP/IP stack module task function.

b) Link Local Functions

	Name	Description
	TCPIP_ZCLL_Disable	Disables Zero Configuration on the specified interface.
	TCPIP_ZCLL_Enable	Enables Zero Configuration on the specified interface.
	TCPIP_ZCLL_IsEnabled	Returns whether or not an interface is enabled for zero configuration
	TCPIP_ZCLL_Task	Standard TCP/IP stack module task function.

c) Data Types and Constants

	Name	Description
	MDNSD_ERR_CODE	void DisplayHostName(uint8_t *HostName);

	ZCLL_MODULE_CONFIG	Placeholder for Zero Configuration Link Layer module configuration.

Description

This section describes the Application Programming Interface (API) functions of the Zeroconf module.

Refer to each section for a detailed description.

a) Multicast DNS Functions

TCPIP_MDNS_ServiceDeregister Function

DNS Service Discovery function for end-user to deregister a service advertisement, which was previously registered with the TCPIP_MDNS_ServiceRegister function.

File

zero_conf_multicast_dns.h

C

```
MDNSD_ERR_CODE TCPIP_MDNS_ServiceDeregister(TCPIP_NET_HANDLE netH);
```

Returns

MDNSD_ERR_CODE - Returns an error code to indicate whether or not registration is successful:

- MDNSD_SUCCESS - Returns on success of call
- MDNSD_ERR_INVAL - When the input parameters are invalid or if the function is invoked in an invalid state

Description

This function is used by end-user application to deregister DNS Service Discovery on a local network. When this gets invoked by end-user DNS SD stack sends out Good-Bye packets to update all peer machines that service will no longer be present. All peer machines remove the corresponding entry from the Browser list.

This is the last functions that needs to be invoked by the end-user application to free the DNS SD stack for some other application.

Preconditions

TCPIP_MDNS_ServiceRegister must be invoked before this call.

Parameters

Parameters	Description
netH	handle of the network to be deregistered

Function

MDNSD_ERR_CODE TCPIP_MDNS_ServiceDeregister()

TCPIP_MDNS_ServiceRegister Function

DNS Service Discovery function for end-users to register a service advertisement. The service is associated with all interfaces.

File

zero_conf_multicast_dns.h

C

```
MDNSD_ERR_CODE TCPIP_MDNS_ServiceRegister(TCPIP_NET_HANDLE netH, const char * srv_name, const char *
srv_type, uint16_t port, const uint8_t * txt_record, uint8_t auto_rename, void (*call_back)(char *name,
MDNSD_ERR_CODE err, void *context), void * context);
```

Returns

MDNSD_ERR_CODE - Returns Error-code to indicate whether or not registration is successful:

- MDNSD_SUCCESS - returns on success of call
- MDNSD_ERR_BUSY - When already some other service is being advertised using this DNS SD stack
- MDNSD_ERR_INVAL - Invalid parameter

Description

This function is used by end-user application to announce its service on local network. All peer machines that are compliant with Multicast DNS

and DNS Service Discovery protocol can detect the announcement and lists out an entry in the Service Browser list. The end-user selects an entry to connect to this service. So ultimately this is an aid to end-user to discover any service of interest on a local network.

This is the first function that needs to be invoked by end-user application. Presently Multicast-DNS and Service discovery stack supports only single service advertisement. Once the application wants to terminate the service it has to invoke TCPIP_MDNS_ServiceDeregister function to free the DNS SD stack for some other application.

Preconditions

None.

Parameters

Parameters	Description
netH	handle of the network to be registered
srv_name	Service Name, which is being advertised
srv_type	For a HTTP-Service its "_http._tcp.local"
_http	is application protocol proceeded with an underscore
_tcp	is lower-layer protocol on which service runs
local	is to represent service is on local-network For a iTunes Music Sharing "_daap._tcp.local" For a Printing Service "_ipp._tcp.local" Refer to http://www.dns-sd.org/ServiceTypes.html for more service types.
port	Port number on which service is running
txt_len	For additional information about service like default page (e.g., "index.htm") for HTTP-service. Length of such additional information
txt_record	String of additional information (e.g., "index.htm") for HTTP service.
auto_rename	A flag to indicate DNS-SD stack, whether to rename the service automatically or not. If this is set to '0' Callback parameter will be used to indicate the conflict error and user has to select different name and re-register with this function. If this is set to '1' service-name will be automatically renamed with numerical suffix.
callback	Callback function, which is user-application defined. This callback gets invoked on completion of service advertisement. If an service name-conflict error is detected and auto_rename is set to '0' callback gets invoked with MDNSD_ERR_CONFLICT as error-code.
context	Opaque context (pointer to opaque data), which needs to be used in callback function.

Function

MDNSD_ERR_CODE TCPIP_MDNS_ServiceRegister(...)

TCPIP_MDNS_ServiceUpdate Function

DNS-Service Discovery function for end-user to update the service advertisement, which was previously registered with TCPIP_MDNS_ServiceRegister.

File

zero_conf_multicast_dns.h

C

```
MDNSD_ERR_CODE TCPIP_MDNS_ServiceUpdate(TCPIP_NET_HANDLE netH, uint16_t port, const uint8_t * txt_record);
```

Returns

MDNSD_ERR_CODE - Returns an error code to indicate whether or not registration is successful:

- MDNSD_SUCCESS - returns on success of call
- MDNSD_ERR_INVAL - When the input parameters are invalid or if the function is invoked in invalid state

Description

This function is used by the end-user application to update its service, which was previously registered. With this function, the end-user application updates the port number on which the service is running. It can also update the additional information of service. For example, the default page can be updated to new page and corresponding page name can be input to this function to update all peer machines. The modified service will be announced with new contents on local network.

This is an optional function and should be invoked only if it is necessary.

Preconditions

TCPIP_MDNS_ServiceRegister must be invoked before this call.

Parameters

Parameters	Description
netH	Handle of the network to perform the service update
port	Port number on which service is running
txt_record	String of additional information (e.g., "index.htm") for HTTP-service.

Function

MDNSD_ERR_CODE TCPIP_MDNS_ServiceUpdate(
TCPIP_NET_HANDLE netH,
uint16_t port,
const uint8_t *txt_record)

TCPIP_MDNS_Task Function

Standard TCP/IP stack module task function.

File

zero_conf_multicast_dns.h

C

```
void TCPIP_MDNS_Task();
```

Returns

None.

Description

This function performs MDNS module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

MDNS module should have been initialized.

Function

void TCPIP_MDNS_Task(void)

b) Link Local Functions

TCPIP_ZCLL_Disable Function

Disables Zero Configuration on the specified interface.

File

zero_conf_link_local.h

C

```
bool TCPIP_ZCLL_Disable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This API is used by end-user application to disable Zero Configuration on a specific interface.

Remarks

None.

Preconditions

The TCP/IP stack must be initialized before calling this function.

Parameters

Parameters	Description
hNet	handle of the network to be disabled

Function

bool TCPIP_ZCLL_Disable(TCPIP_NET_HANDLE hNet)

TCPIP_ZCLL_Enable Function

Enables Zero Configuration on the specified interface.

File

zero_conf_link_local.h

C

```
bool TCPIP_ZCLL_Enable(TCPIP_NET_HANDLE hNet);
```

Returns

- true - Indicates success
- false - Indicates failure

Description

This API is used by end-user application to enable Zero Configuration on a specific interface.

Remarks

None.

Preconditions

TCP/IP stack must be initialized before calling this function.

Parameters

Parameters	Description
hNet	handle of the network to be enabled

Function

bool TCPIP_ZCLL_Enable(TCPIP_NET_HANDLE hNet)

TCPIP_ZCLL_IsEnabled Function

Returns whether or not an interface is enabled for zero configuration

File

zero_conf_link_local.h

C

```
bool TCPIP_ZCLL_IsEnabled(TCPIP_NET_HANDLE hNet);
```

Returns

- true - interface is enabled for zero configuration
- false - interface is not enabled for zero configuration

Description

This API is used by end-user application check whether or not that an interface is enabled for zero configuration.

Preconditions

The TCP/IP stack must be initialized before calling this function.

Parameters

Parameters	Description
hNet	handle of the network to be examined

Function

bool TCPIP_ZCLL_IsEnabled(TCPIP_NET_HANDLE hNet)

TCPIP_ZCLL_Task Function

Standard TCP/IP stack module task function.

File

zero_conf_link_local.h

C

```
void TCPIP_ZCLL_Task();
```

Returns

None.

Description

This function performs ZCLL module tasks in the TCP/IP stack.

Remarks

None.

Preconditions

The ZCLL module should have been initialized.

Function

void TCPIP_ZCLL_Task(void)

c) Data Types and Constants

MDNSD_ERR_CODE Enumeration

File

zero_conf_multicast_dns.h

C

```
typedef enum {
  MDNSD_SUCCESS = 0,
  MDNSD_ERR_BUSY = 1,
  MDNSD_ERR_CONFLICT = 2,
  MDNSD_ERR_INVAL = 3
} MDNSD_ERR_CODE;
```

Members

Members	Description
MDNSD_ERR_BUSY = 1	Already Being used for another Service
MDNSD_ERR_CONFLICT = 2	Name Conflict
MDNSD_ERR_INVAL = 3	Invalid Parameter

Description

void DisplayHostName(uint8_t *HostName);

ZCLL_MODULE_CONFIG Structure

File

zero_conf_link_local.h

C

```
typedef struct {
} ZCLL_MODULE_CONFIG;
```

Description

Placeholder for Zero Configuration Link Layer module configuration.

Files

Files

Name	Description
zero_conf_link_local.h	Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network.
zero_conf_multicast_dns.h	Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network.

Description

This section lists the source and header files used by the library.

zero_conf_link_local.h

Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network.

Functions

	Name	Description
	TCPIP_ZCLL_Disable	Disables Zero Configuration on the specified interface.
	TCPIP_ZCLL_Enable	Enables Zero Configuration on the specified interface.
	TCPIP_ZCLL_IsEnabled	Returns whether or not an interface is enabled for zero configuration
	TCPIP_ZCLL_Task	Standard TCP/IP stack module task function.

Structures

	Name	Description
	ZCLL_MODULE_CONFIG	Placeholder for Zero Configuration Link Layer module configuration.

Description

Zero Configuration (Zeroconf) IPV4 Link Local Addressing Module for the Microchip TCP/IP Stack

Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network. It also provides for a more human-like naming convention, instead of relying on IP addresses alone. Zeroconf also goes by the names Bonjour (Apple) and Avahi (Linux), and is an IETF standard.

File Name

zero_conf_link_local.h

Company

Microchip Technology Inc.

zero_conf_multicast_dns.h

Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network.

Enumerations

	Name	Description
	MDNSD_ERR_CODE	void DisplayHostName(uint8_t *HostName);

Functions

	Name	Description
	TCPIP_MDNS_ServiceDeregister	DNS Service Discovery function for end-user to deregister a service advertisement, which was previously registered with the TCPIP_MDNS_ServiceRegister function.
	TCPIP_MDNS_ServiceRegister	DNS Service Discovery function for end-users to register a service advertisement. The service is associated with all interfaces.
	TCPIP_MDNS_ServiceUpdate	DNS-Service Discovery function for end-user to update the service advertisement, which was previously registered with TCPIP_MDNS_ServiceRegister.
	TCPIP_MDNS_Task	Standard TCP/IP stack module task function.

Description

Zero Configuration (Zeroconf) Multicast DNS and Service Discovery Module for Microchip TCP/IP Stack

Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network. It also provides for a more human-like naming convention, instead of relying on IP addresses alone. Zeroconf also goes by the names Bonjour (Apple) and Avahi (Linux), and is an IETF standard.

File Name

zero_conf_multicast_dns.h

Company

Microchip Technology Inc.

Test Libraries Help

This section provides descriptions of the Test Libraries that are available in MPLAB Harmony.

Test Harness Library

This section describes the Test Harness Library that is available in MPLAB Harmony.

Description

There are two types of test libraries included in the MPLAB Harmony installation. The first type is a single Test Harness Library. This library implements a state machine that runs and manages specially developed tests that are used to validate the behavior of MPLAB Harmony-compatible libraries in the different execution environments supported. The second type of test library implements the tests that are run by the Test Harness Library. These libraries are predefined and implemented by Microchip to validate existing MPLAB Harmony drivers, middleware, and other libraries. This type of library may be used in conjunction with the test harness library to verify the behavior of different implementations of existing MPLAB Harmony libraries.

Introduction

This library runs and manages specially developed tests used to validate the behavior of MPLAB Harmony-compatible libraries in the different execution environments supported.

Description

This Test Harness Library help and usage documentation defines the library's interface and expected behavior. To utilize the Test Harness Library, a test application must be developed that initializes and runs the Test Harness Library. Additionally, tests must be written that fulfill the requirements of the Test Harness Library. These tests need to call the interface functions of a Library Under Test (LUT) and verify its expected behavior. Refer to the MPLAB Test Harness User's Guide for additional details on developing tests and test applications.

The Test Harness Library can execute a number of tests, each of which validates some aspect of the interface and behavior of a MPLAB Harmony-compatible library. Each test can have a number of sub-tests. If all sub-tests pass, the test passes. If any sub-test fails, the test fails. If all tests in a given configuration pass, the harness reports an overall passing result. If any test fails, the harness reports an overall failure result. Error messages, individual test results, and other messages are output as they occur, provided that a SYS_DEBUG output method is available and has been configured. Otherwise, a debugger must be used to check the results stored in an internal data structure of the Test Harness.

Library Overview

This section provides an overview of the Test Harness Library.

Description

The library interface routines are divided into various sub-sections, which address one of the blocks of the overall operation of the Test Harness Library.

Library Interface Section	Description
System Interface Functions	Functions used by the MPLAB Harmony system to initialize and maintain the Test Harness Library and libraries under test.
Platform Initialization Functions	Functions used to perform hardware or test platform-specific initialization.
Test Results Functions	Functions used by tests to report results to the harness.
Test Time-out Functions	Function used to maintain the test time out timer.
Test Harness Initialization Data Types	These structures provide the list of tests that the harness executes.
Test Interface Function Pointer Types	These pointers define the functions that the Test Harness requires each test must implement.

State Machine

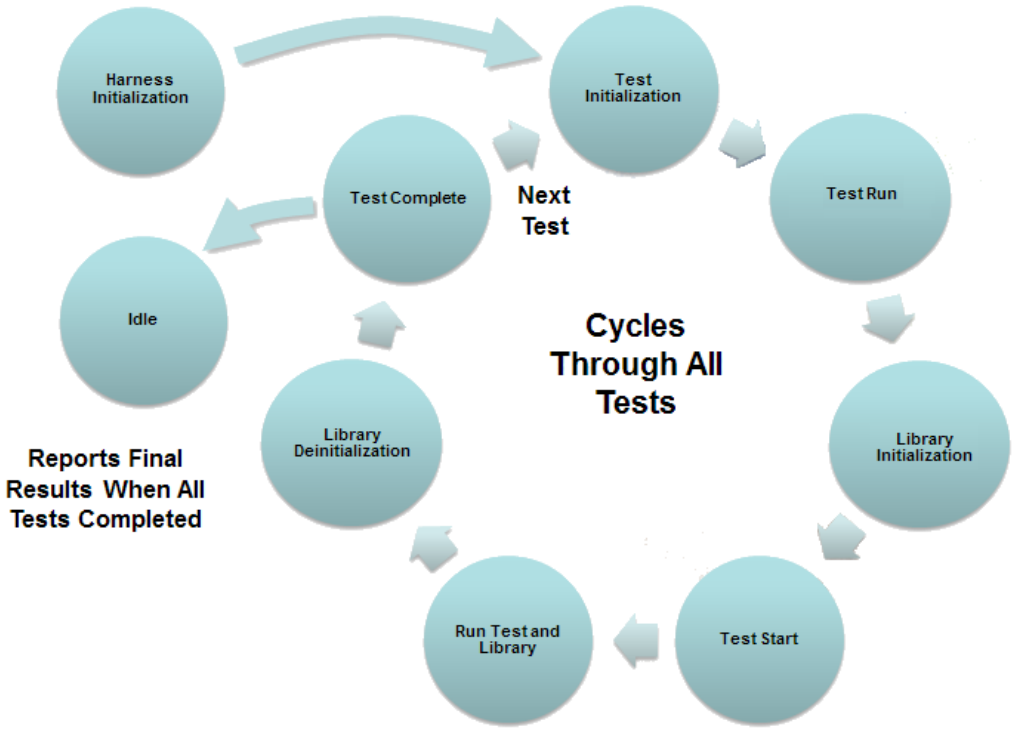
This topic describes the State Machine for the MPLAB Harmony Test Harness Library.

Description

The Test Harness State Machine (see **Figure 1**) runs in a cycle that executes all tests in the test list that was provided to it when it was initialized. This state machine first performs any required configuration-specific initialization by calling the TEST_PlatformInitialize function. Then, it initializes and prepares the first test to be run. When the test indicates that it is ready, the harness initializes the library and runs both the test and the LUT. As the test runs, it reports intermediate "sub-test" results back to the harness, which accumulates them for later reporting. The tests and harness

use debug message output to print results and information, if so configured. Intermediate debug messages can be "printed" at any time, but they may affect test timing. When the test is completed, the test harness deinitializes the LUT, reports results for that test, and advances to the next test library to start the process over again. When the last test finishes, the harness reports the final results and goes idle or hits a breakpoint (in Debug mode).

Figure 1: Test Harness State Machine



Refer to the MPLAB Harmony Test Harness User's Guide for information on using the Test Harness Library to develop test applications and for information on developing tests that meet the requirements of the test harness.

Using the Library

This topic describes the basic architecture of the Test Harness Library and provides information and examples on its use.

Description

The interface to the Test Harness Library is defined in the `test_harness.h` header file.

Any C language source (`.c`) file that uses the Test Harness Library must include `test_harness.h`.

How the Library Works

This section provides information on how the Test Harness Library works.

Core Functionality

Refer to the MPLAB Harmony Test Harness User's Guide for information on the core functionality of the Test Harness Library.

Configuring the Library

Macros

	Name	Description
	TEST_HARNESS_MAX_NUM_TASKS_PER_LIBRARY	Defines the maximum number of "Tasks" routines implemented by a single library under test.
	TEST_HARNESS_MAX_NUM_INSTANCES_PER_LIBRARY	Defines the maximum number of instances of a single library under test.
	TEST_IDLE_SLEEP_MS	Defines the number of milliseconds the test harness will sleep when the test reports that it is idle.
	TEST_IDLE_SLEEP_MS_LIBRARY	Defines the number of milliseconds the library tasks function will sleep when the library is idle.

	TEST_TIMER_CLOCK_PRESCALER	Defines the selected clock prescaler value for the timeout timer.
	TEST_TIMER_CLOCK_SOURCE	Defines the selected clock source for the timer used for the timeout timer.
	TEST_TIMER_ID	Defines the timer peripheral used for the test timeout timer.
	TEST_TIMER_INCREMENT_PERIOD	Defines the number of timer counts that make up a single timer period.
	TEST_TIMER_INTERRUPT_PRIORITY	Defines the interrupt priority for the selected test timer.
	TEST_TIMER_INTERRUPT_SOURCE	Defines the interrupt source ID of the timer used for the timeout timer.
	TEST_TIMER_INTERRUPT_SUBPRIORITY	Defines the interrupt subpriority for the selected test timer.
	TEST_TIMER_INTERRUPT_VECTOR	Defines the interrupt vector for the selected test timer.
	TEST_TIMER_MS_PER_INCREMENT	Defines the number of milliseconds per timer increment period.
	TEST_TIMER_MS_TIMEOUT	Defines the number of milliseconds that must expire before a test will time out.

Description

The configuration of the Test Harness Library is based on the file `system_config.h`.

This header file contains the configuration selections for the Test Harness Library. Based on the selections made, the library may support the support selected features.

The path to this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TEST_HARNESS_MAX_NUM_TASKS_PER_LIBRARY Macro

Defines the maximum number of "Tasks" routines implemented by a single library under test.

File

test_config_template.h

C

```
#define TEST_HARNESS_MAX_NUM_TASKS_PER_LIBRARY 5
```

Description

Maximum Number of Tasks for a Library Under Test

This configuration parameter defines the maximum number of "Tasks" routines are allowed for a single library under test.

Remarks

This parameter has a default value defined in test_harness.h that will generate a warning when used. To eliminate the warning, define a value in system_config.h.

TEST_HARNESS_MAX_NUM_INSTANCES_PER_LIBRARY Macro

Defines the maximum number of instances of a single library under test.

File

test_harness.h

C

```
#define TEST_HARNESS_MAX_NUM_INSTANCES_PER_LIBRARY 3
```

Description

Maximum Number of Instances of a Library Under Test

Remarks

Refer to config/test_config_template.h for additional information.

TEST_IDLE_SLEEP_MS Macro

Defines the number of milliseconds the test harness will sleep when the test reports that it is idle.

File

test_config_template.h

C

```
#define TEST_IDLE_SLEEP_MS 50
```

Description

Test Idle Sleep Time

This configuration option defines the number of milliseconds that the test harness will sleep before calling the tasks function of the test when the it reports it is idle.

Remarks

This capability is only supported in an RTOS configuration, but this value must be defined in all configurations.

TEST_IDLE_SLEEP_MS_LIBRARY Macro

Defines the number of milliseconds the library tasks function will sleep when the library is idle.

File

test_config_template.h

C

```
#define TEST_IDLE_SLEEP_MS_LIBRARY 5
```

Description

Library under Test Idle Sleep Time

This configuration option defines the number of milliseconds that the test harness will sleep before calling the tasks function of the library under test when the library status reports it is idle.

Remarks

This capability is only supported in an RTOS configuration, but this value must be defined in all configurations.

TEST_TIMER_CLOCK_PRESCALER Macro

Defines the selected clock prescaler value for the timeout timer.

File

test_config_template.h

C

```
#define TEST_TIMER_CLOCK_PRESCALER TMR_PRESCALE_VALUE_256
```

Description

Timeout Timer Clock Prescaler

This configuration option defines the selected clock prescaler value for the timeout timer.

Remarks

The frequency of the clock source selected by the TEST_TIMER_CLOCK_SOURCE is divided by the value identified by this option.

The test harness takes direct ownership of this timer, utilizing the timer peripheral library to control it. The selected prescaler value must be valid for the selected timer, as defined by the timer peripheral library.

TEST_TIMER_CLOCK_SOURCE Macro

Defines the selected clock source for the timer used for the timeout timer.

File

test_config_template.h

C

```
#define TEST_TIMER_CLOCK_SOURCE TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK
```

Description

Timeout Timer Clock Source

This configuration option defines the selected clock source for the timer used for the timeout timer.

Remarks

The test harness takes direct ownership of this timer, utilizing the timer peripheral library to control it. The selected clock source must be valid for the selected timer, as defined by the timer peripheral library.

TEST_TIMER_ID Macro

Defines the timer peripheral used for the test timeout timer.

File

test_config_template.h

C

```
#define TEST_TIMER_ID TMR_ID_1
```

Description

Timeout Timer

This configuration option defines the timer peripheral used for the test timeout timer. It must be defined as the timer peripheral library instance ID of the desired timer instance.

Remarks

This parameter has a default value defined in test_harness.c that will generate a warning when used. To eliminate the warning, define a value in system_config.h.

The test harness takes direct ownership of this timer, utilizing the timer peripheral library to control it. The selected timer ID must be valid, as defined by the timer peripheral library.

TEST_TIMER_INCREMENT_PERIOD Macro

Defines the number of timer counts that make up a single timer period.

File

test_config_template.h

C

```
#define TEST_TIMER_INCREMENT_PERIOD 31250
```

Description

Test Timer Increment Period

This configuration option defines the number of timer counts of the selected timer clock source that will be counted before a single timer period expires.

Remarks

The test harness takes direct ownership of this timer, operating it in 16-bit mode. The selected period value must be valid for the selected timer, as defined by the timer peripheral library.

TEST_TIMER_INTERRUPT_PRIORITY Macro

Defines the interrupt priority for the selected test timer.

File

test_config_template.h

C

```
#define TEST_TIMER_INTERRUPT_PRIORITY INT_PRIORITY_LEVEL1
```

Description

Test Timer Interrupt Priority

This configuration option defines the interrupt priority for the selected test timer.

Remarks

None.

TEST_TIMER_INTERRUPT_SOURCE Macro

Defines the interrupt source ID of the timer used for the timeout timer.

File

test_config_template.h

C

```
#define TEST_TIMER_INTERRUPT_SOURCE INT_SOURCE_TIMER_1
```

Description

Timeout Timer Interrupt Source

This configuration option defines the interrupt source ID of the timer used for the timeout timer.

Remarks

The test harness utilizes the interrupt system service to control this interrupt. The selected interrupt source must be valid for the selected timer, as defined by the interrupt system service.

TEST_TIMER_INTERRUPT_SUBPRIORITY Macro

Defines the interrupt subpriority for the selected test timer.

File

test_config_template.h

C

```
#define TEST_TIMER_INTERRUPT_SUBPRIORITY INT_SUBPRIORITY_LEVEL0
```

Description

Test Timer Interrupt Subpriority

This configuration option defines the interrupt subpriority for the selected test timer.

Remarks

None.

TEST_TIMER_INTERRUPT_VECTOR Macro

Defines the interrupt vector for the selected test timer.

File

test_config_template.h

C

```
#define TEST_TIMER_INTERRUPT_VECTOR INT_VECTOR_T1
```

Description

Test Timer Interrupt Vector

This configuration option defines the interrupt vector for the selected test timer.

Remarks

None.

TEST_TIMER_MS_PER_INCREMENT Macro

Defines the number of milliseconds per timer increment period.

File

test_config_template.h

C

```
#define TEST_TIMER_MS_PER_INCREMENT 100
```

Description

Test Timer Milliseconds Per Increment

This configuration option defines the number of milliseconds that expire per timer period. The internal timeout timer is incremented by this many milliseconds every time the timer period defined by the TEST_TIMER_INCREMENT_PERIOD configuration option expires.

Remarks

This configuration option effectively translates the timer period from scaled clock cycles to milliseconds.

TEST_TIMER_MS_TIMEOUT Macro

Defines the number of milliseconds that must expire before a test will time out.

File

test_config_template.h

C

```
#define TEST_TIMER_MS_TIMEOUT 2000
```

Description

Test Timer Timeout Value

This configuration option defines the number of milliseconds that must expire before a test will time out and be considered as a failure.

Remarks

None.

Building the Library

This section lists the files that are available in the Test Harness Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/test`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/test_harness.h`	This file provides the interface definitions of the Test Harness Library.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/test_harness.c`	Test Harness Library implementation.

Optional File(s)

There are no optional files for the Test Harness Library.

Module Dependencies

The Test Harness Library is dependent upon the following libraries and header files.

- Interrupt System Service Library
- Operating System Abstraction Layer (OSAL)
- Debug System Service Library (or an implementation of the SYS_DEBUG_PRINT and SYS_DEBUG_MESSAGE macros)
- sys_common.h
- sys_module.h

Library Interface

a) System Interface Functions

	Name	Description
	TEST_Initialize	MPLAB Harmony test harness initialization routine.
	TEST_Tasks	Runs the current test's tasks functions.
	TEST_HarnessTasks	MPLAB Harmony test harness' tasks function.
	TEST_LibraryTasksPolled	Calls the a tasks function of a library under test when the test harness is ready to run the library.
	TEST_LibraryTasksISR	Calls an ISR-based tasks function of the library under test.

b) Platform Initialization Functions

	Name	Description
	TEST_PlatformInitialize	Called by the test harness to initialize configuration-specific test support.
	TEST_PlatformReady	Indicates that the test configuration is ready and testing can begin.

c) Test Results Functions

	Name	Description
	TEST_HasCompleted	Allows a test to indicate that it has completed.
	TEST_HasPassedSubtest	Allows a test module to indicate that an individual subtest has completed and identifies if it has passed or failed.
	TEST_LibraryInitialize	Indicates that the test is ready for the harness to initialize the library under test.

d) Test Time-out Functions

	Name	Description
	TEST_TimerIncrement	Increments test timeout timer.

e) Test Harness Initialization Data Types

	Name	Description
	LIBRARY_UNDER_TEST	This is type LIBRARY_UNDER_TEST.
	_library_under_test	This is type LIBRARY_UNDER_TEST.
	_test_data	Holds data necessary to initialize and run a single test.
	_test_init_data	Holds the test harness's initialization data.
	TEST_DATA	Holds data necessary to initialize and run a single test.
	TEST_INIT_DATA	Holds the test harness's initialization data.

f) Test Interface Function Pointer Types

	Name	Description
	TEST_MODULE_INITIALIZE_ROUTINE	Pointer to a routine that initializes a test module.
	TEST_MODULE_START_FUNCTION	Pointer to a routine that starts the test module.
	TEST_MODULE_STATUS_ROUTINE	Pointer to a routine that gets the current status of a test module.
	TEST_MODULE_TASKS_ROUTINE	Pointer to a routine that performs the tasks necessary to maintain the state machine of a test module system module.

Description

This section describes the APIs of the Test Harness Library.

a) System Interface Functions

TEST_Initialize Function

MPLAB Harmony test harness initialization routine.

File

test_harness.h

C

```
void TEST_Initialize(const TEST_INIT_DATA * init);
```

Returns

None.

Description

This function initializes the MPLAB Harmony test harness. It places the harness in its initial state and prepares it to run so that the TEST_HarnessTasks function can be called.

Remarks

This routine must be called from the SYS_Initialize function.

The test harness will initialize tests and libraries under test later, under control of its state machine.

Preconditions

All other system required initialization routines should be called before calling this routine (in "SYS_Initialize"). This function should be called before or in place of the application's in initialization function.

Example

```
TEST_Initialize();
```

Parameters

Parameters	Description
init	Pointer to test initialization data.

Function

void TEST_Initialize (const APP_TH_INIT_DATA *init)

TEST_Tasks Function

Runs the current test's tasks functions.

File

test_harness.h

C

```
void TEST_Tasks(unsigned int number);
```

Returns

None.

Description

This function runs the current test's tasks functions, once the harness determines that the test should run.

Remarks

This function uses the following configuration parameters.

- TEST_IDLE_SLEEP_MS - Number of milliseconds that the test will sleep waiting for the module when it is busy.

Preconditions

The system and test harness must have all been initialized.

Example

None.

Parameters

Parameters	Description
number	Test's tasks function number. (Tests can have more than one tasks function.

Function

void TEST_Tasks(unsigned int number)

TEST_HarnessTasks Function

MPLAB Harmony test harness' tasks function.

File

test_harness.h

C

```
void TEST_HarnessTasks();
```

Returns

None.

Description

This routine is the tasks function of the MPLAB Harmony test harness and defines the state machine and core logic of the test harness.

Remarks

This routine must be called from SYS_Tasks() routine or from an RTOS thread loop.

Preconditions

Any required system initialization functions must have been called and the "TEST_Initialize" function must have been called.

Example

```
TEST_HarnessTasks();
```

Function

void TEST_HarnessTasks (void)

TEST_LibraryTasksPolled Function

Calls the a tasks function of a library under test when the test harness is ready to run the library.

File

test_harness.h

C

```
void TEST_LibraryTasksPolled(unsigned int testIndex, SYS_MODULE_INDEX libraryIndex,
SYS_MODULE_TASKS_ROUTINE libraryTasks);
```

Returns

None.

Description

This routine calls a tasks function of a library under test, indirectly under control of the test harness once the test is ready to verify the library.

Remarks

This routine must be called from the SYS_Tasks function or the appropriate RTOS thread loop when the library under test is configured for polled mode.

This routine will prevent the library's tasks function from being called before the library has been initialized.

Preconditions

The system, harness, and test must have been initialized.

Example

```
TEST_LibraryTasksPolled();
```

Parameters

Parameters	Description
testIndex	Index in test list to the test for this library.
libraryIndex	Index in test list for this instance of the library.
libraryTasks	Pointer to the library's Tasks function.

Function

void TEST_LibraryTasksPolled (unsigned int testIndex,
SYS_MODULE_INDEX libraryIndex,
SYS_MODULE_TASKS_ROUTINE libraryTasks)

TEST_LibraryTasksISR Function

Calls an ISR-based tasks function of the library under test.

File

test_harness.h

C

```
void TEST_LibraryTasksISR(unsigned int testIndex, SYS_MODULE_INDEX moduleIndex, SYS_MODULE_TASKS_ROUTINE
libraryTasks, INT_SOURCE source);
```

Returns

None.

Description

This routine calls an ISR-based tasks function of a library under test, indirectly under control of the test harness once the test is ready to verify the library.

Remarks

This routine must be called from the appropriate ISR vector function when the module under test is configured for interrupt mode.

Preconditions

The system, harness, and test must have been initialized.

Example

```
void __ISR(_TIMER_2_VECTOR, ipl2) _IntHandlerDrvTmrInstance1(void)
{
    TEST_LibraryTasksISR(2, 0, SAMPLE_Tasks, INT_SOURCE_TIMER_2);
}
```

Parameters

Parameters	Description
testIndex	Index to the APP_TH_TEST_DATA data structure passed to the test harness's "TEST_HarnessTasks" routine during system initialization.
moduleIndex	Index to the appropriate module instance.
libraryTasks	Pointer to the appropriate "Tasks" function for the library under test.
source	Interrupt source ID.

Function

void TEST_LibraryTasksISR (unsigned int testIndex,
SYS_MODULE_INDEX libraryTasks,
SYS_MODULE_TASKS_ROUTINE mutTasks,
INT_SOURCE source)

b) Platform Initialization Functions

TEST_PlatformInitialize Function

Called by the test harness to initialize configuration-specific test support.

File

test_harness.h

C

```
bool TEST_PlatformInitialize();
```

Returns

- true - If successfully able to initialize the configuration-specific test support.
- false - If not able to successfully initialize the configuration-specific test support.

Description

This function is called by the test harness to initialize configuration- specific test support.

Remarks

A weak implementation of this function is provided in the test harness library. If no external implementation is defined, then the internal weak implementation (which simply returns "true") will be used.

Preconditions

The system must have been initialized.

Example

None.

Function

bool TEST_PlatformInitialize (void)

TEST_PlatformReady Function

Indicates that the test configuration is ready and testing can begin.

File

test_harness.h

C

```
void TEST_PlatformReady();
```

Returns

None.

Description

This function indicates that the test configuration is ready and testing can begin. The purpose of this function is to allow the call to the TEST_PlatformInitialize function to return false when the configuration requires some time to complete its initialization.

Remarks

If TEST_PlatformInitialize returns "true", then it is not necessary to call this function.

Preconditions

The system, and test harness must have been initialized.

Example

TEST_PlatformReady();

Function

void TEST_PlatformReady (void)

c) Test Results Functions

TEST_HasCompleted Function

Allows a test to indicate that it has completed.

File

test_harness.h

C

```
void TEST_HasCompleted();
```

Returns

None.

Description

This function allows a test to indicate that it has completed. It is used to prematurely end the test.

Remarks

The harness will automatically end the test when the test's "Status" function indicates it has completed.

The test harness will accumulate and report subtest results using SYS_PRINT and/or SYS_MESSAGE macros. The test does not need to report results. Test only need to report debug info using SYS_DEBUG_* macros to help identify the reason for any failure.

A test is considered to have passed if all sub-tests pass. If any sub-tests fail, the test is considered to have failed. If no subtest results are reported, the test is considered to have failed.

Preconditions

The test harness must have been initialized.

Example

```
// Indicate that a test with one subtest has passed.
TEST_HasPassedSubtest(true);
TEST_HasCompleted();
```

Function

void TEST_HasCompleted (void)

TEST_HasPassedSubtest Function

Allows a test module to indicate that an individual subtest has completed and identifies if it has passed or failed.

File

test_harness.h

C

```
void TEST_HasPassedSubtest(bool passed);
```

Returns

None.

Description

This function allows a test module to indicate that an individual subtest has completed and identifies if it has passed or failed.

Remarks

The test harness will accumulate and report subtest results using SYS_PRINT and/or SYS_MESSAGE macros. The test module does not need to report results. Tests only need to report debug info using SYS_DEBUG_* macros to help identify the reason for any failure.

A test is considered to have passed if all sub-tests pass. If any sub-tests fail, the test is considered to have failed. If no subtest results are reported, the test is considered to have failed.

Preconditions

The application must have been initialized.

Example

```
// Indicate that the subtest has passed.
```

```
TEST_HasPassedSubtest(true);
```

Function

void TEST_HasPassedSubtest (bool passed)

TEST_LibraryInitialize Function

Indicates that the test is ready for the harness to initialize the library under test.

File

test_harness.h

C

```
bool TEST_LibraryInitialize();
```

Returns

- true If initialization of the library under test completed.
- false If initialization of the library under test has not yet completed.

Description

This function indicates that the test is ready for the harness to initialize the library under test.

Remarks

None.

Preconditions

The system, test harness, and test must have been initialized.

Example

if (TEST_LibraryInitialize()) { // begin testing }

Function

bool TEST_LibraryInitialize (void)

d) Test Time-out Functions

TEST_TimerIncrement Function

Increments test timeout timer.

File

test_harness.h

C

```
void TEST_TimerIncrement();
```

Returns

None.

Description

This function increments the test timeout counter by a number of milliseconds defined by TEST_TIMER_MS_PER_INCREMENT.

Remarks

This function should be called by the interrupt or alarm callback from the timer used by the test harness.

Preconditions

The system, test harness, test and timer must have been initialized.

Example

```
void __ISR(_TIMER_1_VECTOR, IPL1AUTO) _IntHandlerDrvTmrInstance0(void)
{
```

```
    TEST_TimerIncrement();
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
}
```

Function

void TEST_TimerIncrement (void)

e) Test Harness Initialization Data Types

LIBRARY_UNDER_TEST Structure

File

test_harness.h

C

```
typedef struct _library_under_test {
  SYS_MODULE_INITIALIZE_ROUTINE initialize;
  SYS_MODULE_REINITIALIZE_ROUTINE reinitialize;
  SYS_MODULE_DEINITIALIZE_ROUTINE deinitialize;
  SYS_MODULE_STATUS_ROUTINE status;
  unsigned int numberOfInstances;
  unsigned int numberOfTasks;
  SYS_MODULE_TASKS_ROUTINE tasks[TEST_HARNESS_MAX_NUM_TASKS_PER_LIBRARY];
  SYS_MODULE_INIT * initData[TEST_HARNESS_MAX_NUM_INSTANCES_PER_LIBRARY];
} LIBRARY_UNDER_TEST;
```

Members

Members	Description
SYS_MODULE_INITIALIZE_ROUTINE initialize;	Pointer to the library's initialization routine.
SYS_MODULE_REINITIALIZE_ROUTINE reinitialize;	Pointer to the library's reinitialization routine.
SYS_MODULE_DEINITIALIZE_ROUTINE deinitialize;	Pointer to the library's deinitialization routine.
SYS_MODULE_STATUS_ROUTINE status;	Pointer to the library's status routine.
unsigned int numberOfInstances;	Number of instances of the library under test.
unsigned int numberOfTasks;	Number of tasks routines required by each instance of the library under test.
SYS_MODULE_TASKS_ROUTINE tasks[TEST_HARNESS_MAX_NUM_TASKS_PER_LIBRARY];	Array of pointers to the library's "Tasks" routine(s).
SYS_MODULE_INIT * initData[TEST_HARNESS_MAX_NUM_INSTANCES_PER_LIBRARY];	Array of pointers to the library's init data structures.

Description

This is type LIBRARY_UNDER_TEST.

TEST_DATA Structure

Holds data necessary to initialize and run a single test.

File

test_harness.h

C

```
typedef struct _test_data {
  char * name;
  TEST_MODULE_INITIALIZE_ROUTINE initialize;
  TEST_MODULE_TASKS_ROUTINE tasks[TEST_HARNESS_MAX_NUM_TASKS];
  TEST_MODULE_START_FUNCTION start;
  TEST_MODULE_STATUS_ROUTINE status;
  SYS_MODULE_INDEX index;
  SYS_MODULE_INIT * initData;
  LIBRARY_UNDER_TEST * library;
  SYS_MODULE_OBJ testObj;
```

```
  unsigned int passed;
  unsigned int count;
  bool completed;
} TEST_DATA;
```

Members

Members	Description
char * name;	Pointer to test name string.
TEST_MODULE_INITIALIZE_ROUTINE initialize;	Pointer to test's initialize routine.
TEST_MODULE_TASKS_ROUTINE tasks[TEST_HARNESS_MAX_NUM_TASKS];	Pointer to test's tasks routine.
TEST_MODULE_START_FUNCTION start;	Pointer to test's start routine.
TEST_MODULE_STATUS_ROUTINE status;	Pointer to test's status routine.
SYS_MODULE_INDEX index;	Test module's index.
SYS_MODULE_INIT * initData;	Initial data for test module.
LIBRARY_UNDER_TEST * library;	Pointer to library under test data structure.
SYS_MODULE_OBJ testObj;	Test module's object handle.
unsigned int passed;	Count of sub-tests that have passed.
unsigned int count;	Count of sub-tests that have been run.
bool completed;	Flag indicating all sub-tests completed (module is idle).

Description

Test Data

This structure holds the data necessary to initialize and run a single test.

TEST_INIT_DATA Structure

Holds the test harness's initialization data.

File

test_harness.h

C

```
typedef struct _test_init_data {
  unsigned int numberOfTests;
  TEST_DATA * tests;
} TEST_INIT_DATA;
```

Members

Members	Description
unsigned int numberOfTests;	Number of tests.
TEST_DATA * tests;	Pointer to test data array.

Description

Test Harness Initialization Data

This structure holds the test harness's initialization data.

f) Test Interface Function Pointer Types

TEST_MODULE_INITIALIZE_ROUTINE Type

Pointer to a routine that initializes a test module.

File

test_harness.h

C

```
typedef SYS_MODULE_INITIALIZE_ROUTINE TEST_MODULE_INITIALIZE_ROUTINE;
```

Returns

A handle to the instance of the test module that was initialized. This handle is a necessary parameter to all of the other harness level routines for this test.

Description

This data type is a pointer to a routine that initializes a test module.

Remarks

This function will only be called once per test instance when the harness is ready to initialize the test instance.

Preconditions

The system and harness initialization must have been completed before the harness state machine will call the initialization routine for any tests.

Parameters

Parameters	Description
index	Identifier for the test instance to be initialized
init	Pointer to the data structure containing any data necessary to initialize the test. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ TEST__Initialize) (
const SYS_MODULE_INDEX index,
const SYS_MODULE_INIT * const init)

TEST_MODULE_START_FUNCTION Type

Pointer to a routine that starts the test module.

File

test_harness.h

C

```
typedef bool (* TEST_MODULE_START_FUNCTION)(SYS_MODULE_OBJ object);
```

Returns

- true - If the test module is successfully able to start testing.
- false - If the test module is unable to start testing.

Description

This is a pointer to a routine that starts the test application.

Remarks

Each test module's "Start" function is called via a pointer to begin running the test contained in that module.

Preconditions

The test harness application and the test application must have been initialized and "running" (i.e., the test applications tasks function must be called from the test harness's state machine).

Example

None.

Parameters

Parameters	Description
object	Object handle to the module instance

Function

bool TEST__Start (SYS_MODULE_INDEX index)

TEST_MODULE_STATUS_ROUTINE Type

Pointer to a routine that gets the current status of a test module.

File

test_harness.h

C

```
typedef SYS_MODULE_STATUS_ROUTINE TEST_MODULE_STATUS_ROUTINE;
```

Returns

One of the possible status codes from SYS_STATUS, extended to include:

SYS_STATUS_TEST_COMPLETED - Indicates the test has completed.

Description

This data type is a pointer to a routine that gets the current status of a test module.

Remarks

A test's status operation can be used to determine when the test is busy or has completed as well as to obtain general status of the test.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). A module may define module-specific error values of less or equal SYS_STATUS_ERROR_EXTENDED (-10).

The status function must NEVER block.

If the status operation returns an error value the test has failed.

Preconditions

The system and harness initialization must have (and will be) completed and the test module's initialization routine will have been called before the harness will call the status routine for a test.

Example

None.

Parameters

Parameters	Description
object	Handle to the module instance

Function

SYS_STATUS TEST__Status (SYS_MODULE_OBJ object)

TEST_MODULE_TASKS_ROUTINE Type

Pointer to a routine that performs the tasks necessary to maintain the state machine of a test module system module.

File

test_harness.h

C

```
typedef void (* TEST_MODULE_TASKS_ROUTINE)(SYS_MODULE_OBJ object, unsigned int index);
```

Returns

None.

Description

This data type is a pointer to a routine that performs the tasks necessary to maintain the state machine of a test module module.

Remarks

A test module can have one or more (up to TEST_HARNESS_MAX_NUM_TASKS) tasks functions.

Preconditions

The system and harness initialization must have been completed and the test's initialization routine called before the harness will call any of the test's tasks routines.

Example

None.

Parameters

Parameters	Description
object	Handle to the test instance object
index	Index identifying which tasks function is being called. This index can be used to identify if a single task function is called from a different tasks context.

Function

void TEST__Tasks (SYS_MODULE_OBJ object, unsigned int index)

Files

Files

Name	Description
test_harness.h	This header file provides prototypes and definitions for the MPLAB Harmony test harness.
test_config_template.h	Test harness configuration template file.

Description

This section lists the source and header files used by the TEST HARNESS LIB Library.

test_harness.h

This header file provides prototypes and definitions for the MPLAB Harmony test harness.

Functions

	Name	Description
	TEST_HarnessTasks	MPLAB Harmony test harness' tasks function.
	TEST_HasCompleted	Allows a test to indicate that it has completed.
	TEST_HasPassedSubtest	Allows a test module to indicate that an individual subtest has completed and identifies if it has passed or failed.
	TEST_Initialize	MPLAB Harmony test harness initialization routine.
	TEST_LibraryInitialize	Indicates that the test is ready for the harness to initialize the library under test.
	TEST_LibraryTasksISR	Calls an ISR-based tasks function of the library under test.
	TEST_LibraryTasksPolled	Calls the a tasks function of a library under test when the test harness is ready to run the library.
	TEST_PlatformInitialize	Called by the test harness to initialize configuration-specific test support.
	TEST_PlatformReady	Indicates that the test configuration is ready and testing can begin.
	TEST_Tasks	Runs the current test's tasks functions.
	TEST_TimerIncrement	Increments test timeout timer.

Macros

	Name	Description
	TEST_HARNESS_MAX_NUM_INSTANCES_PER_LIBRARY	Defines the maximum number of instances of a single library under test.

Structures

	Name	Description
	_library_under_test	This is type LIBRARY_UNDER_TEST.
	_test_data	Holds data necessary to initialize and run a single test.
	_test_init_data	Holds the test harness's initialization data.
	LIBRARY_UNDER_TEST	This is type LIBRARY_UNDER_TEST.

	TEST_DATA	Holds data necessary to initialize and run a single test.
	TEST_INIT_DATA	Holds the test harness's initialization data.

Types

	Name	Description
	TEST_MODULE_INITIALIZE_ROUTINE	Pointer to a routine that initializes a test module.
	TEST_MODULE_START_FUNCTION	Pointer to a routine that starts the test module.
	TEST_MODULE_STATUS_ROUTINE	Pointer to a routine that gets the current status of a test module.
	TEST_MODULE_TASKS_ROUTINE	Pointer to a routine that performs the tasks necessary to maintain the state machine of a test module system module.

Description

MPLAB Harmony Test Harness Header File

This header file provides function prototypes and data type definitions for the test harness interface.

File Name

test_harness.h

Company

Microchip Technology Inc.

test_config_template.h

Test harness configuration template file.

Macros

	Name	Description
	TEST_HARNESS_MAX_NUM_TASKS_PER_LIBRARY	Defines the maximum number of "Tasks" routines implemented by a single library under test.
	TEST_IDLE_SLEEP_MS	Defines the number of milliseconds the test harness will sleep when the test reports that it is idle.
	TEST_IDLE_SLEEP_MS_LIBRARY	Defines the number of milliseconds the library tasks function will sleep when the library is idle.
	TEST_TIMER_CLOCK_PRESCALER	Defines the selected clock prescaler value for the timeout timer.
	TEST_TIMER_CLOCK_SOURCE	Defines the selected clock source for the timer used for the timeout timer.
	TEST_TIMER_ID	Defines the timer peripheral used for the test timeout timer.
	TEST_TIMER_INCREMENT_PERIOD	Defines the number of timer counts that make up a single timer period.
	TEST_TIMER_INTERRUPT_PRIORITY	Defines the interrupt priority for the selected test timer.
	TEST_TIMER_INTERRUPT_SOURCE	Defines the interrupt source ID of the timer used for the timeout timer.
	TEST_TIMER_INTERRUPT_SUBPRIORITY	Defines the interrupt subpriority for the selected test timer.
	TEST_TIMER_INTERRUPT_VECTOR	Defines the interrupt vector for the selected test timer.
	TEST_TIMER_MS_PER_INCREMENT	Defines the number of milliseconds per timer increment period.
	TEST_TIMER_MS_TIMEOUT	Defines the number of milliseconds that must expire before a test will time out.

Description

Test Harness Configuration Template

This file provides the list of all the configurations that can be used with the test harness. This file should not be included by any source files. It is strictly for documentation.

File Name

test_config_template.h

Company

Microchip Technology Inc.

USB Libraries Help

This section provides descriptions of the USB libraries that are available in MPLAB Harmony.

USB Device Library

This section provides information on the USB Device libraries that are available in MPLAB Harmony.

USB Device Library - Getting Started

This section provides information for getting started with the USB Device Library.

Introduction

Provides an introduction to the MPLAB Harmony USB Device Library

Description

The MPLAB Harmony USB Device Library (referred to as the USB Device Library) provides embedded application developers with a framework to design and develop a wide variety of USB Devices. A choice of Full Speed only or Full Speed and Hi-Speed USB operations are available, depending on the selected PIC32 microcontroller. The USB Device Library facilitates development of standard USB devices through function drivers that implement standard USB Device class specification. Vendor USB devices can be implemented via USB Device Layer Endpoint functions. The USB Device Library is modular, thus allowing application developers to readily design composite USB devices. The USB Device Library is a part of the MPLAB Harmony installation and is accompanied by demonstration applications that highlight library usage. These demonstrations can also be modified or updated to build custom applications. The USB Device Library also features the following:

- Support for different USB device classes (CDC, Audio, HID, MSD, and Vendor)
- Supports multiple instance of the same class in a composite device
- Supports multiple configurations at different speeds
- Supports Full-Speed and High-Speed operation
- Supports multiple USB peripherals (allows multiple device stacks)
- Modular and Layered architecture
- Supports deferred control transfer responses
- Completely non-blocking
- Supports both polled and interrupt operation
- Works readily in an RTOS application

This document serves as a getting started guide and provides information on the following:

- USB Device Library architecture
- USB Device Library - application interaction
- Creating your own USB device

Note: It is assumed that the reader is familiar with the USB 2.0 specification (available at www.usbif.org). While the document, for the sake completeness, does cover certain aspects of the USB 2.0 protocol, it is recommended that the reader refer to the specification for a complete description of USB operation.

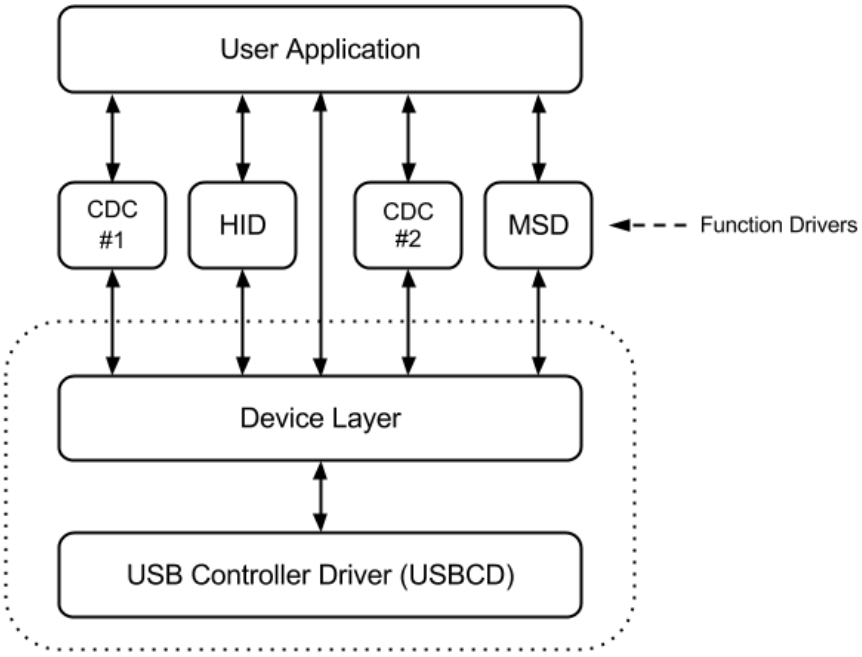
USB Device Library Architecture

Describes the USB Device Library Architecture.

Description

The USB Device Library features a modular and layered architecture, as illustrated in the following figure.

USB Device Library Architecture



As seen in the figure, the USB Device Library consists of the following three major components.

USB Controller Driver (USBCD)

The USBCD manages the state of the USB peripheral and provides the Device Layer with structured data access methods to the USB. It also provides the Device layer with USB events. The USBCD is a MPLAB Harmony driver and uses the MPLAB Harmony framework components (the USB peripheral Library, the Interrupt System Service) for its operation. It supports only one client per instance of the USB Peripheral. This client would typically be the Device Layer. In case of multiple USB peripherals, the USBCD can manage multiple USB peripherals, each being accessed by one client. The driver is accessed exclusively by the Device Layer in the USB Device Layer Architecture. It is initialized by the Device Layer (when the Device Layer is initialized) and in case of polling operation, its Tasks routine is called by the Device Layer. The USBCD provides functions to:

- Enable, disable and stall endpoints
- Schedule USB transfers
- Attach or detach the device
- Control resume signalling

The USB Controller Driver can be configured for Polled or Interrupt mode. Configuring the driver for Interrupt mode configures the Device Stack for Device mode operation. Configuring the driver for Polled mode configures the USB Device Stack for Polled mode operation. An USB device can be designed using either the polled or interrupt mode. However, it is recommended that the USB Controller Driver (and therefore the USB Device Stack) be configured for Interrupt mode. This reduces the impact of other application components on the operation of the Device Stack. Configuring for Interrupt mode also ensure timely stack event generation and stack response to bus events on the USB.

Device Layer

The Device Layer responds to the enumeration requests issued by the USB Host. It has exclusive access to an instance of the USBCD and the control endpoint (Endpoint 0). When the Host issues a class specific control transfer request, the Device Layer will analyze the setup packet of the control transfer and will route the control transfer to the appropriate function driver. The Device Layer must be initialized with the following data:

- Master Descriptor Table - This is a table of all the configuration descriptors and string descriptors.
- Function Driver Registration Table - This table contains information about the function drivers in the application
- USBCD initialization information - This specifies the USB peripheral interrupt, the USB Peripheral instance and Sleep mode operation options

The Device Layer initializes all function drivers that are registered with it when it receives a Set Configuration (for a supported configuration) from the Host. It deinitializes the function drivers when a USB reset event occurs. It initializes the USBCD, opens it and registers a event handler to receive USB events. The Device Layer can also be opened by the application (the application becomes a client to the Device Layer). The application can then receive bus and device events and respond to control transfer requests. The Device Layer provides events to the application such as device configured or device reset. Some of these events are notification-only events, while other events require the application to take action.

Function Drivers

The Function Drivers implements various USB device classes as per the class specification. The USB Device Library architecture can support multiple instances of a function driver. An example would be a USB CDC device that emulates two serial ports. Function drivers provide an abstracted and an easy to use interface to the application. The application must register an event handler with the function driver to receive function driver events and must respond to some of these events with control transfer read/write functions. Function drivers access the bus through the Device Layer.

USB Device Library - Application Interaction

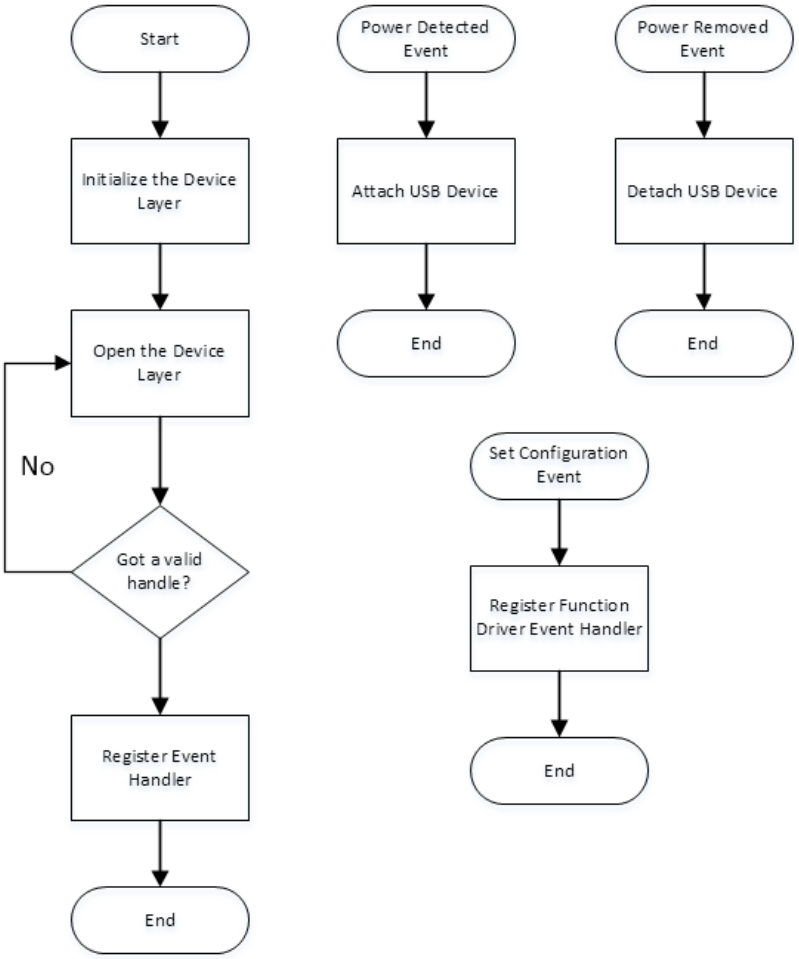
Describes how the application must interact with the USB Device Stack.

Description

Note: Additional information on program, data, and stack memory requirements, and a list of USB application configurations is available in the USB Demonstrations section.

The following figure highlights the steps that the application must follow to use the USB Device Library.

Application Interaction with Device Layer

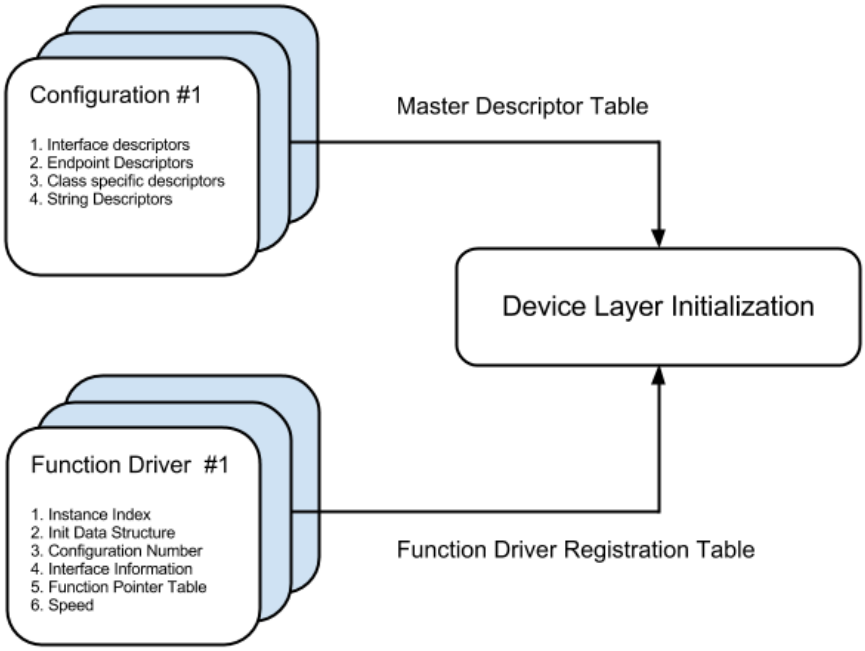


The application must first initialize the Device Layer. As a part of the Device Layer initialization process, the Device Layer initialization structure must be defined which in turn requires the following data structures to be designed

- The master descriptor table
- The function driver registration table

The following figure shows a pictorial representation of the data that forms the Device Layer initialization structure. Additional information on Device Layer initialization is available in the Device Layer Help File.

Device Layer Initialization



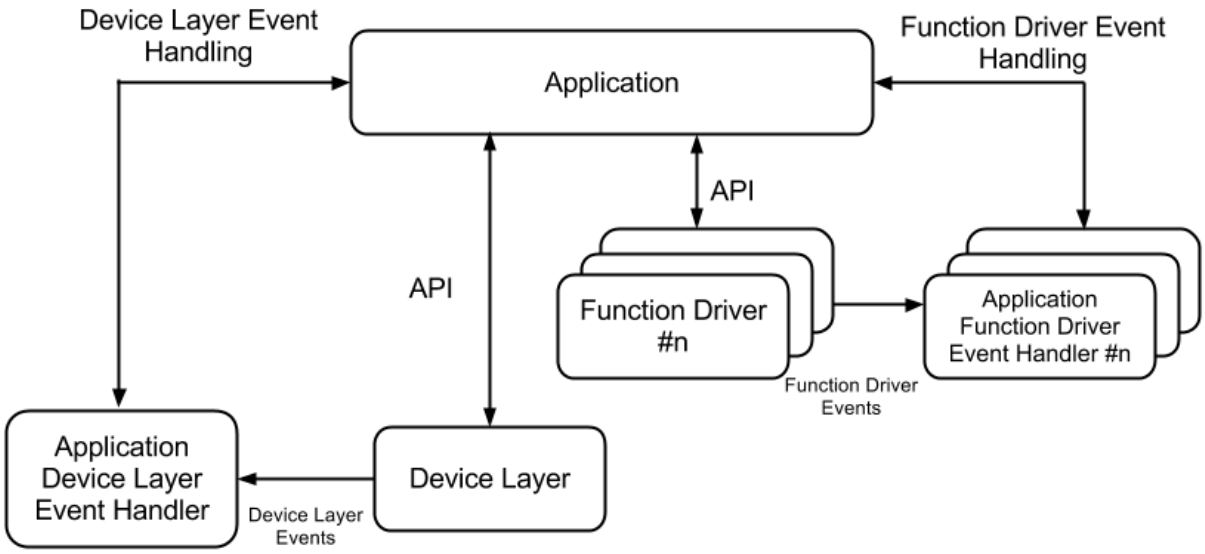
After successful initialization of the Device layer, the application can open the Device layer and register a Device layer event handler. The Device layer event handler receives device level events such as device configured, device deconfigured, device reset and device suspended. The device configured event and deconfigured event are important. The application can use the device deconfigured event to reinitialize its internal state machine. When the application receives a device configured event, it must register event handlers for each function driver that is relevant to the configuration that was set. The function driver event handler registration must be done in the device configured event context because the Device layer acknowledges the set configuration request from the host when it exits the device configured event handler context. The application at this point should be ready to respond to function driver events.

Note: Not registering the function driver event handler in the Device layer configured event could cause the device to not respond to the host requests and therefore, be non-compliant.

Once configured, the device is now ready to serve its intended function on the USB. The application interacts with the Device layer and function drivers through API function and event handlers. The application must be aware of function driver events which require application response. For example, the USB_DEVICE_CDC_EVENT_SET_LINE_CODING event from the USB CDC Function Driver requires the application to respond with a USB_DEVICE_ControlRead function. This function provides the buffer to receive the line coding parameters that the Host sends in the data stage of the Set Line Coding control transfer.

The following figure shows the application interaction with Device layer and function driver after the device has been configured.

Application - Device Layer Interaction after device configuration



In the previous figure, the application should have registered the Device layer event handler before attaching the device on the bus. It should have registered the function driver event handler before exiting the device configured - Device layer event. The application will then receive function

driver instance specific events via the function driver event handlers.

Deferring Control Transfer Responses

Class-specific control transfer related function driver events require the application to complete the data stage and/or the status of the control transfer. The application does this by using the Device Layer Control Transfer API to complete the Control Read/Write transfers. The application may typically be able the complete required data processing, and to continue (or end) the control transfer within the function driver event handler context. However, there could be cases where the required control transfer data processing may require hardware access or extended computation. Performing extended processing or waiting for external hardware within the function driver event handler context is not recommended as the USB 2.0 Specification places restrictions on the control transfer response time.

In cases where the application is not ready to respond to control transfer requests within the function driver event handler context, the USB Device Library provides the option of deferring the response to the control transfer event. The application can respond to the control transfer request after exiting the handler function. The application must still observe the USB 2.0 Specification control transfer timing requirements while responding to the control transfer. Deferring the response in such a manner provides the application with flexibility to analyze the control transfer without degrading the performance of the device on the USB.

Creating Your Own USB Device

Describes how to create a USB device with the MPLAB Harmony USB Device Library.

Description

The first step in creating a USB device is identifying whether the desired device function fits into any of the standard USB device class functions. Using standard USB classes may be advantageous as major operating systems feature Host driver support for standard USB devices. However, the application may not want to tolerate the overhead associated with standard USB device class protocols, in which case, a Vendor USB device can be implemented. A Vendor USB device can be implemented by using the USB Device Layer Endpoint functions; however, these devices will require custom USB host drivers for their operation. Having identified the device class to be used, the following approaches are available for developing a USB device by using the USB Device Library.

Use the Available Library Demonstration Applications

The USB Device Library release package contains a set of demonstration applications that are representative of common USB devices. These can be modified easily to include application specific initialization and application logic. The application logic must be non-blocking and could be implemented as a state machine. Note that the function names and file names referred to in the following section are the those used in the USB Device Library demonstration applications.

- The application specific initialization can be called in the APP_Initialize function (in the `app.c` file). The APP_Initialize function is called from the SYS_Initialize function, which in turn is called when the device comes out of Power-on Reset (POR).
- The application logic is implemented as a state machine in the APP_Tasks function (in the `app.c` file). The application logic can interact with the function driver and the Device layer by using available API calls.
- The application logic can track device events by processing the events in the application USB device event handler function (APP_USBDeviceEventHandler function in `app.c`).

Build a USB Device Library Application Using the MPLAB Harmony Configurator (MHC)

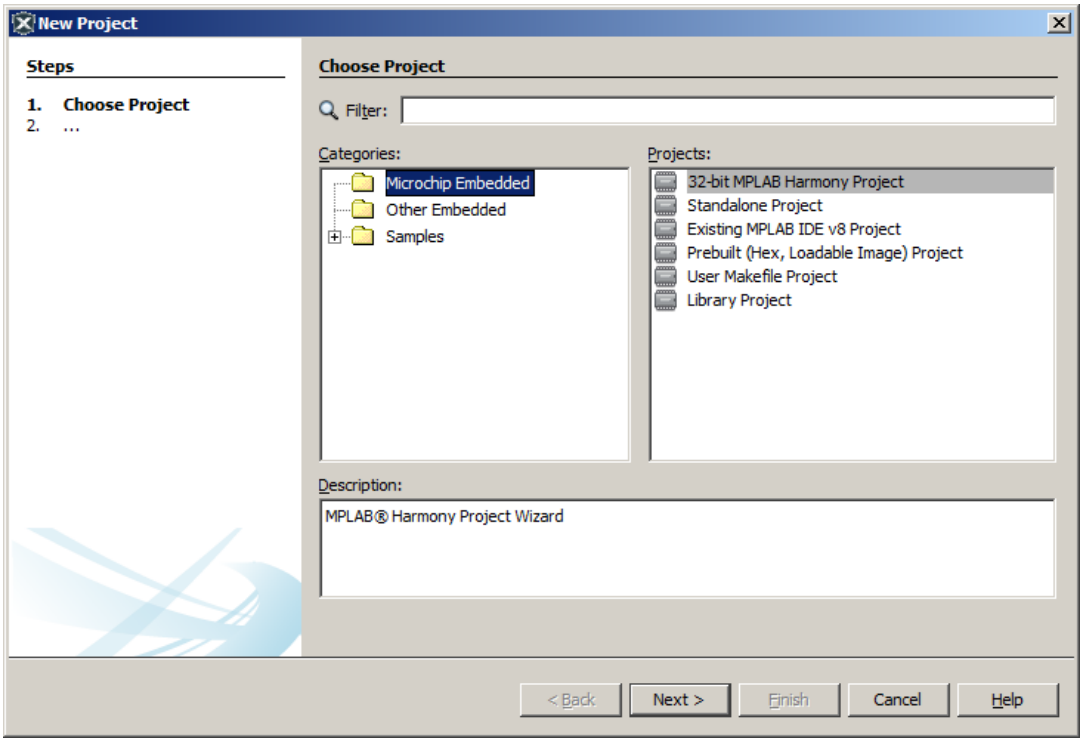
In a case where the available demonstration applications do not meet the end application requirements, a USB device can be created by building a USB Device Library application from scratch. It is recommended to use the Microchip Harmony Configurator (MHC) to generate a USB Device project. Using the MHC ensures that the correct and required MPLAB Harmony framework files are included in the project. The following section outlines the steps to follow to generate the project using the MHC.

Before You Begin

The following process assumes that the MHC plug-in, `com-microchip-mplab-module-mhc.nbm`, which is located in `<install-dir>/utilities/mhc`, is installed in MPLAB X IDE. If the MHC plug-in is not already installed, refer to Installing a Plug-in Module for information.

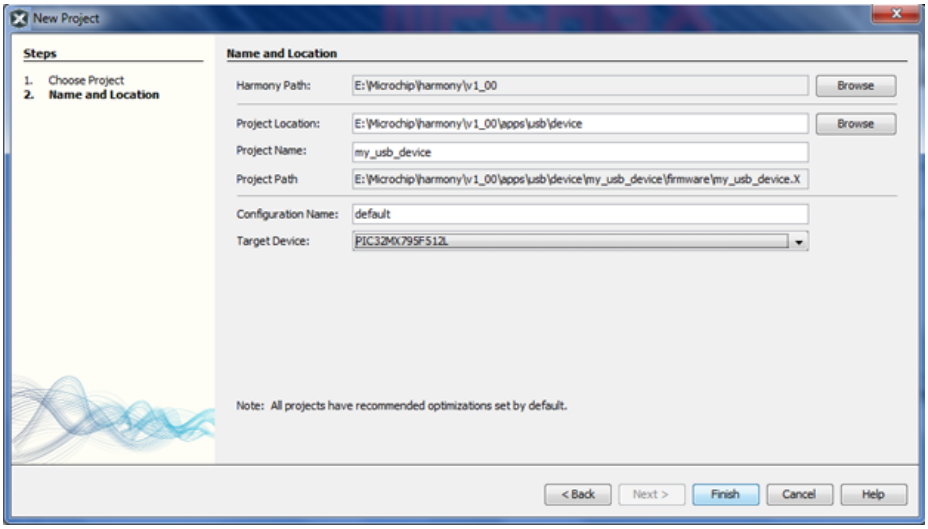
Step 1: Create a New MPLAB X IDE Project

In MPLAB X IDE, select *File* > *New Project* to open the New Project dialog and in Categories: select **Microchip Embedded** and in Projects: select **MPLAB Harmony Project**, and then click **Next**.



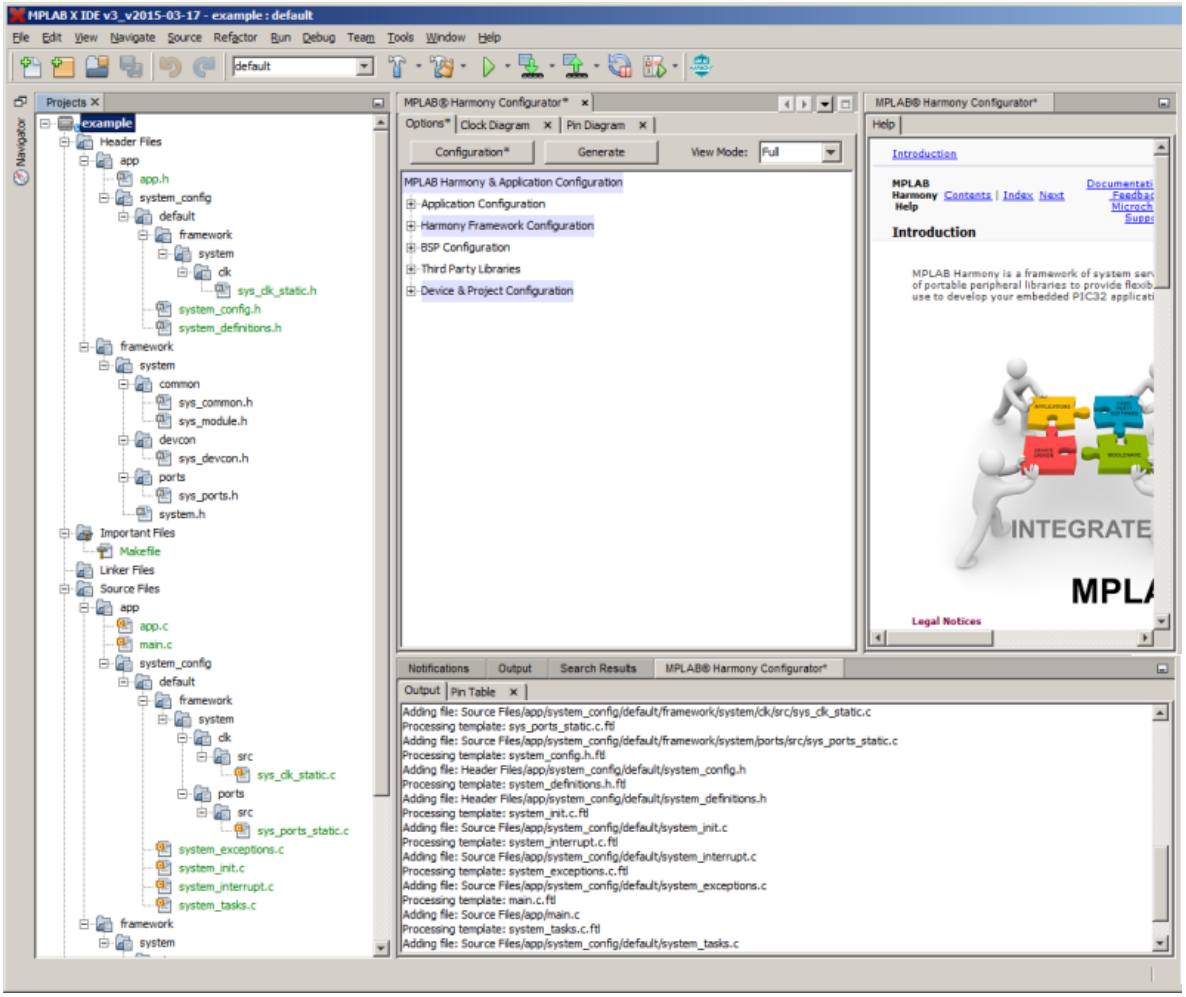
Step 2: Specify Project Name, Location and PIC32 Device Type

In the New Project dialog, specify the Project Location, Project Name, and the Target Device, and then click **Finish**.

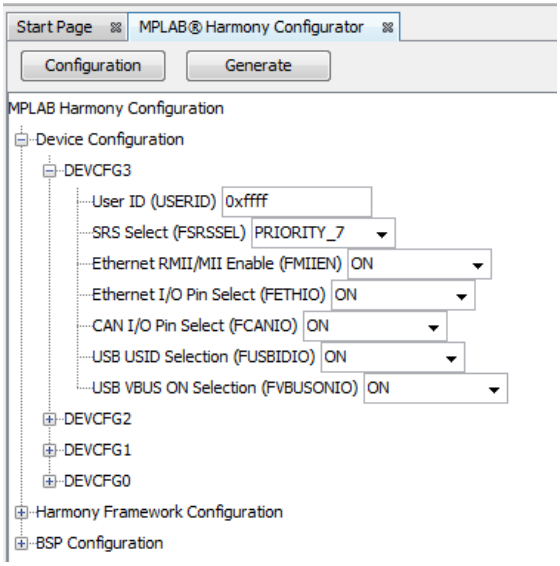


Step 3: Start the MPLAB Harmony Configurator

Open the MHC by selecting *Tools* > *Embedded* > *MPLAB Harmony Configurator*. The initial view is shown in the following figure.

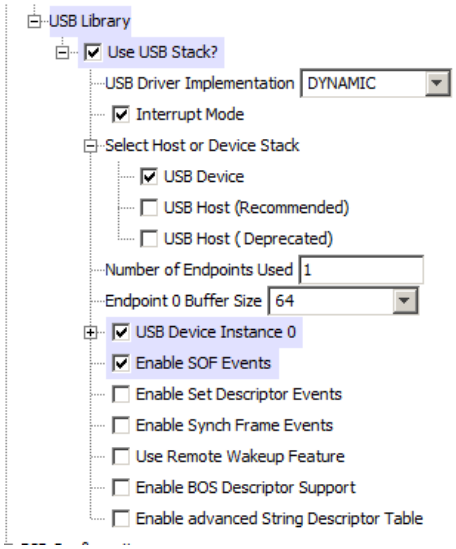


Step 4: Select Device Configuration and configure the fields as required



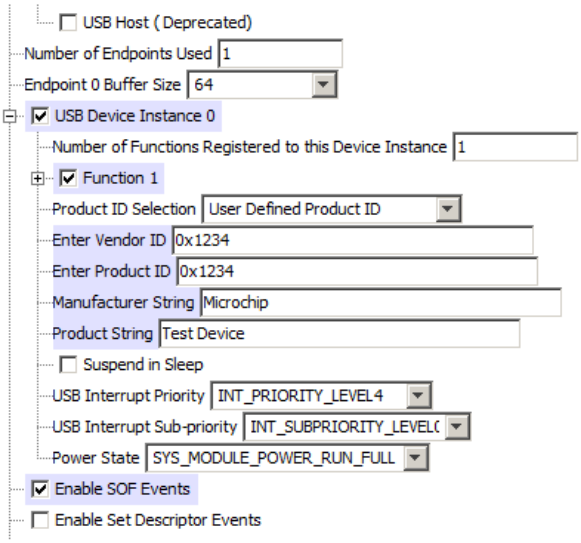
Step 5: MPLAB Harmony Framework Configuration

Expand the MPLAB Harmony Framework Configuration and select **USB Library**, and then select **Use USB Stack**. Doing this will also automatically include other MPLAB Harmony Framework components, that are required by the USB Stack into the project. Select **USB Device** and configure the different parameters as desired.



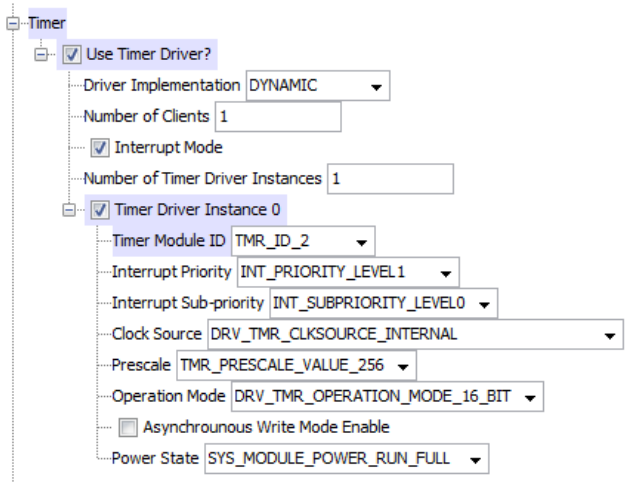
Step 6: MPLAB Harmony Framework Configuration

Expand USB Device Instance 0 and configure parameters as desired.



Step 7: MPLAB Harmony Framework Configuration

For a PIC32MZ device, expand *MPLAB Harmony Framework Configuration >Drivers >Timer* and select the Timer Module ID as TMR_ID_2.



Step 8: Configure Interrupt Priorities

The interrupt priorities of the following modules should be configured as per application needs.

For PIC32MX devices:

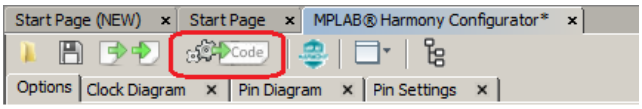
- USB Device Layer

For PIC32MZ devices:

- USB Device Layer
- Timer Driver

Step 9: Generate code

Click **Generate** to generate the code for this configuration.



Step 10: Add application specific information

The following components should be manually added to the generated code:

- USB Descriptors, Function Driver Registration table, USB Device Layer Master Descriptor Table. These data structures should be placed in `system_init.c`. Refer to the USB Device Layer Library section for more information.
- Application initialization steps should be implemented in APP_Initialize function in `app.c`
- Application Tasks routine should be implemented in APP_Tasks function in `app.c`

Step 11: Create a USB Device Layer Event Handler

The application should create a Device Layer Event Handler which will handle all the events generated by the Device Layer. The following code shows a list of all possible events that are generated by the Device Layer. The code can be used in the application and updated to meet the application requirements. Refer to the USB Device Layer Library section for more details on the Device Layer Event Handling. The USB_DEVICE_Attach function can be called to attach the device on the USB when the USB_DEVICE_EVENT_POWER_DETECTED event occurs.

This code example shows how the application can set a Device Layer Event Handler.

```
// Application states
typedef enum
{
    //Application's state machine's initial state.
    APP_STATE_INIT=0,
    APP_STATE_SERVICE_TASKS,
    APP_STATE_WAIT_FOR_CONFIGURATION,
} APP_STATES;

USB_DEVICE_HANDLE usbDeviceHandle;
APP_STATES appState;

// This is the application device layer event handler function.

USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler
(
    USB_DEVICE_EVENT event,
    void * pData,
    uintptr_t context
)
{
    USB_SETUP_PACKET * setupPacket;
    switch(event)
    {
        case USB_DEVICE_EVENT_POWER_DETECTED:
            // This event in generated when VBUS is detected. Attach the device
            USB_DEVICE_Attach(usbDeviceHandle);
            break;

        case USB_DEVICE_EVENT_POWER_REMOVED:
            // This event is generated when VBUS is removed. Detach the device
            USB_DEVICE_Detach (usbDeviceHandle);
            break;

        case USB_DEVICE_EVENT_CONFIGURED:
            // This event indicates that Host has set Configuration in the Device.
            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST:
            // This event indicates a Control transfer setup stage has been completed.
            setupPacket = (USB_SETUP_PACKET *)pData;

            // Parse the setup packet and respond with a USB_DEVICE_ControlSend(),
            // USB_DEVICE_ControlReceive or USB_DEVICE_ControlStatus().
```

```
                break;

            case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT:
                // This event indicates that a Control transfer Data has been sent to Host.
                break;

            case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:
                // This event indicates that a Control transfer Data has been received from Host.
                break;

            case USB_DEVICE_EVENT_CONTROL_TRANSFER_ABORTED:
                // This event indicates a control transfer was aborted.
                break;

            case USB_DEVICE_EVENT_SUSPENDED:
                break;

            case USB_DEVICE_EVENT_RESUMED:
                break;

            case USB_DEVICE_EVENT_ERROR:
                break;

            case USB_DEVICE_EVENT_RESET:
                break;

            case USB_DEVICE_EVENT_SOF:
                // This event indicates an SOF is detected on the bus. The     USB_DEVICE_SOF_EVENT_ENABLE
                // macro should be defined to get this event.
                break;
            default:
                break;
        }
    }
```

Step 12: Function Driver Event Handling

The application should create a Function Driver Event Handler which will handle all the events generated by the Function Driver. Refer to the **Event Handling** topic in the applicable function driver documentation for more details on the Function Driver Event Handling.

Step 13: Opening the Device Layer

The application should open the Device Layer in the Application tasks routine and register an event handler with the Device Layer. The Device Layer will generate events when the event handler is registered.

The following code shows an example for how to the open the Device Layer.

```
void APP_Tasks ( void )
    {
        // Check the application's current state.
        switch ( appState )
        {
            // Application's initial state.
            case APP_STATE_INIT:
                // Open the device layer
                usbDeviceHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0,
                    DRV_IO_INTENT_READWRITE );

                if(usbDeviceHandle != USB_DEVICE_HANDLE_INVALID)
                {
                    // Register a callback with device layer to get event notification
                    USB_DEVICE_EventHandlerSet(usbDeviceHandle,
                        APP_USBDeviceEventHandler, 0);
                    appState = APP_STATE_WAIT_FOR_CONFIGURATION;
                }
                else
                {
                    // The Device Layer is not ready to be opened. We should try
                    // gain later.
                }
                break;

            case APP_STATE_SERVICE_TASKS:
```

```
                break;

                // The default state should never be executed.
            default:
                break;
        }
    }
```

USB Device Stack Porting Guide

This section provides information for porting a MLA USB Device to MPLAB Harmony.

Introduction

This topic provides information for porting a MLA USB Device to MPLAB Harmony.

Source Files to Include

This topic provides information on the source files to be included when porting a MLA USB device project to MPLAB Harmony.

Description

The following table lists the source files that must be included for MLA based and MPLAB Harmony based USB device projects. In MLA there is no separate controller driver implementation. Both enumeration functionality and controller driver is implemented in the `usb_device.c` file. In the MPLAB Harmony USB Device Stack, the controller driver is implemented in the `drv_usb.c` and `drv_usb_device.c` files, and enumeration functionality is implemented in the `usb_device.c` file.

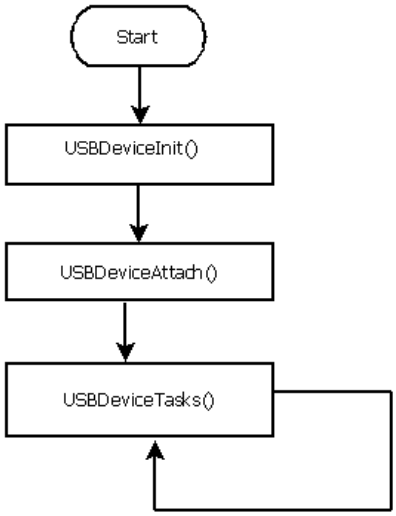
MLA	MPLAB Harmony	Description
`usb_device.c`	`usb_device.c`	In MLA, this file contains enumeration functionality and controller driver implementations. In MPLAB Harmony, this file implements enumeration functionality only.
`usb_function_xxx.c` (Where `xxx` stands for function driver name such as hid, msd, etc.)	`usb_device_xxx.c` (Where `xxx` stands for function driver name such as hid, msd, etc.)	Contains all function driver implementations.
`usb_descriptors.c`	`system_init.c`	Contains USB stack configurations.
N/A	`drv_usb.c` and `drv_usb_device.c`	USB peripheral driver implementation.

Initializing the USB Device Stack

This topic provides initialization information when porting a MLA USB device project to MPLAB Harmony.

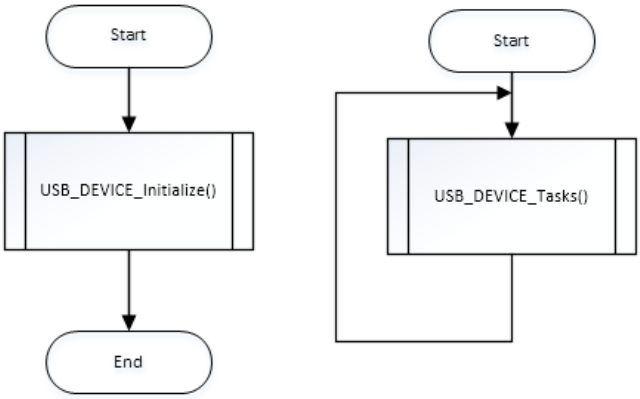
Description

In MLA, initializing the USB device stack consists of calling USB functions in the sequence shown in the following flow chart.



The USBDeviceInit function will initialize the USB Device stack. USBDeviceAttach function will attach the device to bus. USBDeviceTasks function will run the stack task routine.

In the MPLAB Harmony USB Device Stack, initializing the USB device stack consists of calling USB functions in the sequence shown in the following chart.



The USB_DEVICE_Initialize function will initialize the USB device stack. On successful initialization, this function returns a valid system object. This system object is passed to the USB_DEVICE_Tasks function. This function is called periodically in the MPLAB Harmony SYS_Tasks function to maintain the state of the USB Device.

Configuring the Stack

This topic provides stack configuration information when porting a MLA USB device project to MPLAB Harmony.

Description

To configure the stack in MLA, USB descriptors have to be defined in the `usb_descriptors.c` file. The configuration is done in `usb_config.h`. Whereas in MPLAB Harmony, USB descriptors are defined in `system_init.c`. The MPLAB Harmony USB Device Stack configuration is done in `system_config.h`. Aside from descriptors, users must also define a function registration table and a master descriptor table in `system_init.c`. Refer to the specific USB Device section for more information on the function registration table and the master descriptor table.

Calling the Device Layer API

Provides information on calling the Device Layer API functions.

Description

In the Microchip Library of Applications (MLA), the application does not have to explicitly open the Device Layer to invoke the Device Layer API. The Device Layer APIs do not require a client handle.

The MPLAB Harmony USB Device Stack supports multiple instances and therefore requires a client handle. The USB_DEVICE_Open function opens a Device Layer instance and returns a application client handle to the instance of the Device Layer that was opened. This handle is then passed to Device Layer API to specify the instance of Device Layer to be accessed.

Event Handling

This topic provides event handling information when porting a MLA USB device project to MPLAB Harmony.

Description

In MLA, users handle events inside the USER_USB_CALLBACK_EVENT_HANDLER function. Whereas in MPLAB Harmony, the user application client will have to register an event handler with the USB device stack using the USB_DEVICE_EventHandlerSet function. All USB bus events are then forwarded to this user registered event handler function.

The following code shows an example of registering the event handler for the MPAB Harmony stack using the USB_DEVICE_EventHandlerSet function. In the example, the USB_DEVICE_EventHandlerSet function is the application event handler.

```
/* Open the Device Layer */

USB_DEVICE_HANDLE usbDeviceHandle;

usbDeviceHandle = USB_DEVICE_Open(USB_DEVICE_INDEX_0, DRV_IO_INTENT_READWRITE);

/* Register an event handler */

USB_DEVICE_EventHandlerSet(usbDeviceHandle, APP_USBDeviceEventHandler, 0);
```

The following code shows the implementation of the APP_USBDeviceEventHandler function.

```
void APP_USBDeviceEventHandler ( USB_DEVICE_EVENT event, void * eventData, uintptr_t context )
{
    switch ( event )
    {
        case USB_DEVICE_EVENT_SOF:

            break;

        case USB_DEVICE_EVENT_RESET:

            break;

        case USB_DEVICE_EVENT_CONFIGURED:

            break;

        case USB_DEVICE_EVENT_POWER_DETECTED:

            /* VBUS was detected. We can attach the device */
            USB_DEVICE_Attach(appData.deviceHandle);
            break;

        case USB_DEVICE_EVENT_POWER_REMOVED:

            /* VBUS is not available any more. Detach the device. */
            USB_DEVICE_Detach(appData.deviceHandle);
            break;

        case USB_DEVICE_EVENT_SUSPENDED:

            break;

        case USB_DEVICE_EVENT_RESUMED:

    break;

        case USB_DEVICE_EVENT_ERROR:

    break;

        default:
            break;
    }
}
```

Initializing and Communicating with the Endpoint

This topic provides endpoint initialization and communication information when porting a MLA USB device project to MPLAB Harmony.

Description

In MLA, user applications explicitly initialize the required endpoints and can directly read and write to the endpoints. The following code shows how MLA initializes the endpoints in an event handler when the device is configured.

```
 ********************************************************************/
BOOL USER_USB_CALLBACK_EVENT_HANDLER(int event, void *pdata, WORD size)
{
    switch(event)
    {
        case EVENT_CONFIGURED:
             //enable the HID endpoint
            USBEnableEndpoint(HID_EP,
              USB_IN_ENABLED|USB_OUT_ENABLED|USB_HANDSHAKE_ENABLED|USB_DISALLOW_SETUP);
            //Re-arm the OUT endpoint for the next packet
            USBOutHandle = HIDRxPacket(HID_EP,(BYTE*)&ReceivedDataBuffer,64);
            break;

        case EVENT_TRANSFER:
            //Add application specific callback task or callback function here if desired.
            break;
        case EVENT_SOF:
            USBCB_SOF_Handler();
            break;
        case EVENT_SUSPEND:
            USBCBSuspend();
```

Endpoint configured in MLA

For MPLAB Harmony, the user application only manages endpoints directly for Generic USB device implementations. For standard USB devices, the USB device layer initializes the endpoint and any communication to an endpoint must occur through function driver APIs. The USB device layer initializes the endpoint based on the information provided by the user in the configurations descriptor. The USB device layer does this by parsing the descriptors table at run time. So the user does not have to explicitly enable or disable the endpoint.

Handling Endpoint 0 (EP0) Packets

This topic provides information for handling Endpoint 0 packets when porting a MLA USB device project to MPLAB Harmony.

Description

Standard device requests are handled by device stack in both MLA and MPLAB Harmony. The difference lies in handling class specific requests. In MLA, the class specific requests are forwarded to the application as EP0 events and application forwards them to appropriate function driver. The following code shows how the application forwards them to appropriate function driver in the event handler.

```
 ********************************************************************/
BOOL USER_USB_CALLBACK_EVENT_HANDLER(int event, void *pdata, WORD size)
{
    switch(event)
    {
        case EVENT_EP0_REQUEST:
            USBCheckHIDRequest();
            break;

        case EVENT_TRANSFER:
            //Add application specific callback task or callback function here if desi
            break;

        case EVENT_CONFIGURED:
             //enable the HID endpoint
            USBEnableEndpoint(HID_EP,
              USB_IN_ENABLED|USB_OUT_ENABLED|USB_HANDSHAKE_ENABLED|USB_DISALLOW_SETUP)
```

EP0 request is forwarded to HID

For the MPLAB Harmony USB Device Stack, EP0 events are generated from the appropriate function driver as meaningful events. For example, the HID function driver in MPLAB Harmony parses the EP0 packet to generate events like USB_DEVICE_HID_EVENT_GET_IDLE, USB_DEVICE_HID_EVENT_SET_REPORT, and USB_DEVICE_HID_EVENT_SET_IDLE, etc.

The following code shows how the event handler callback is registered with the HID function driver using the USB_DEVICE_HID_EventHandlerSet function. This is done in the Device Layer USB_DEVICE_EVENT_CONFIGURED event.

```
case USB_DEVICE_EVENT_CONFIGURED:
     /* Set the flag indicating device is configured. */
     appData.deviceConfigured = true;

     /* Save the other details for later use. */
```

```
    appData.configurationValue = ((USB_DEVICE_EVENT_DATA_CONFIGURED*)
                                  eventData)->configurationValue;

    /* Register application HID event handler */
    USB_DEVICE_HID_EventHandlerSet(USB_DEVICE_HID_INDEX_0, APP_USBDeviceHIDEventHandler,
                                   (uintptr_t)&appData);
```

The following code shows the implementation of the HID event handler function in the application. Note how the HID function driver converts EP0 requests to a HID class-specific control transfer event.

```
USB_DEVICE_HID_EVENT_RESPONSE APP_USBDeviceHIDEventHandler
(
    USB_DEVICE_HID_INDEX iHID,
    USB_DEVICE_HID_EVENT event,
    void * eventData,
    uintptr_t userData
)
{
    USB_DEVICE_HID_EVENT_DATA_REPORT_SENT * reportSent;
    USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED * reportReceived;

    /* Check type of event */
    switch (event)
    {

        case USB_DEVICE_HID_EVENT_SET_IDLE:

            /* For now we just accept this request as is. We acknowledge
             * this request using the USB_DEVICE_HID_ControlStatus()
             * function with a USB_DEVICE_CONTROL_STATUS_OK flag */

            USB_DEVICE_ControlStatus(appData.usbDevHandle, USB_DEVICE_CONTROL_STATUS_OK);

            /* Save Idle rate received from Host */
            appData.idleRate = ((USB_DEVICE_HID_EVENT_DATA_SET_IDLE*)eventData)->duration;
            break;
```

USB Device Stack Porting Example

Provides information on porting a MLA Device Stack demonstration application to the MPLAB Harmony USB Device Stack.

Description

With the introduction of MPLAB Harmony, PIC32MX USB device applications can now be developed using either the USB Device Stack available in Microchip Application Libraries (MLA) v2013-06-15 or the USB Device Stack in the latest version of MPLAB Harmony

In a case where an existing or a partially developed MLA-based USB Device application needs to be migrated to use the MPLAB Harmony USB Device Stack, the migration process is not a one-to-one process, as there are API level differences between the MLA and MPLAB Harmony USB Device Stack. This guide provides a detailed step-by-step comparison between a MLA USB Device and a MPLAB Harmony USB Device application, with an objective to enable the application migration process.

An existing v2013-06-15 MLA USB Device Stack example, the “Device – CDC – Basic Demo” will be compared with the MPLAB Harmony USB Device cdc_com_port_single demonstration application. The outline of the comparative analysis is as follows:

- Highlight framework level similarities and differences between the MLA and MPLAB Harmony USB Device Stack
- Analyze and compare the files included in the MLA “Device – CDC – Basic Demo” demonstration application project to the files included in the MPLAB Harmony USB Device Stack cdc_com_port_single demonstration application
- Analyze and compare the PIC32 Device Fuse Configuration macros, USB Device Stack Configuration, USB Device Stack API and application API in the MLA “Device – CDC – Basic Demo” demonstration application source code with their equivalents in the MPLAB Harmony USB Device cdc_com_port_single demonstration application project

The projects to be compared are available in the MLA USB Device and MPLAB Harmony USB Device demonstration applications folders.

Prerequisites

Describes the prerequisites for using this porting example.

Description

Ensure that the following software is installed:

- Microchip Libraries for Applications v2013-06-15
- The latest version of MPLAB Harmony (available by visiting: www.microchip.com/harmony)

In addition, knowledge of common project-related operations and settings in MPLAB X IDE, as well as familiarity with the MLA USB Solutions and

the USB protocol is assumed.

USB Device Stack in MLA and MPLAB Harmony

In this section, the key similarities and differences between the USB Device Stack in MLA and MPLAB Harmony are discussed.

Description

Similarities

The key similarities between the USB Device Stack in MLA and MPLAB Harmony are:

- Both solutions support Interrupt and Polled mode operation
- Both solutions require configuration macros to control/define functional aspects of the device stack
- Both solutions contain module Tasks routines that need to be invoked either in a `while(1)` application superloop and/or in the USB Interrupt Service Routine (ISR)
- In both solutions, the function drivers and Device Layer are implemented as separate modules

Differences

The key differences between the USB Device Stack in MLA and MPLAB Harmony are listed in the following table.

MPLAB Harmony	MLA
In MPLAB Harmony, the complete user application is factored into System and Application code sections. The code sections are implemented in separate source code files. This represents the MPLAB Harmony Application paradigm.	There is no such factoring in MLA.
In MPLAB Harmony, the USB peripheral management is performed by a USB Controller Driver. This is implemented separately.	The MLA USB Device stack combines the implementation of Device Layer and controller driver into one file.
In MPLAB Harmony, the USB Device Layer is a MPLAB Harmony module and requires the application to open it to access the Device Layer functions.	This is not required in MLA.
In MPLAB Harmony, Device Layer API is factored into System and Client functions. The application calls the client functions along with the Device Layer Handle.	There is no such factoring in the MLA Device Layer implementation.
In MPLAB Harmony, function driver API require the specification of an instance index.	This is typically not required in MLA.
In MPLAB Harmony, the USB Device Stack uses System Services to access shared system resources.	The MLA does not contain a formal implementation of such system services.
In MPLAB Harmony, the Device Layer initializes the function driver based on the configuration set by the host. This process does not require application intervention.	In MLA, the application must add code for the Set Configuration Event handling routine to invoke the initialize routine of the required function driver.

MLA and MPLAB Harmony USB Device Stack Files

This section analyzes the files included in the MLA "Device – CDC – Basic Demo" demonstration application MPLAB X IDE project and identifies the equivalent MPLAB Harmony USB Device Stack and Framework files in the MPLAB Harmony USB Device cdc_com_port_single demonstration MPLAB X IDE project.

The MLA "Device – CDC – Basic Demo" demonstration application MPLAB X IDE project is located in the following MLA installation path: `microchip_solutions_v2013-06-15/USB/Device - CDC - Basic Demo/Firmware/MPLAB.X`.

The MPLAB Harmony cdc_com_port_single project is located in the `<install-dir>/apps/usb/device/cdc_com_port_single/firmware/cdc_com_port_single.X`.

Description

The following table lists the source (`.c`) files and header (`.h`) configuration files in the MLA "Device – CDC – Basic Demo" demonstration application project and their MPLAB Harmony counterparts (along with their paths). The parent folder for MPLAB Harmony framework files, unless otherwise specified, is `<install-dir>/framework`. The table also contains files which are exclusively required by the MPLAB Harmony project.

Classification	MLA USB Device Stack Application	MPLAB Harmony USB Device Stack Application
USB Device Layer Implementation	`usb_device.c`	`/usb/usb_device.c`
USB Device CDC Function Driver	`usb_function_cdc.c`	`/usb/src/dynamic/usb_device_cdc.c` `/usb/src/dynamic/usb_device_cdc_acm.c`

Application main file	`main.c`	`main.c`
		`app.c`
		`app.h` (see **Note**)

USB Device Descriptors	`usb_descriptors.c`	`system_init.c` (See System Configuration in this table.)
USB Device Stack Configuration	`usb_config.h`	`system_config.h` (See System Configuration in this table.)
Hardware Specification	`HardwareProfile.h` `HardwareProfile – PIC32 USB Starter Kit.h`	(See Board Support Package in this table.)
Board Support Package	N/A	`bsp/pic32mx_usb_sk2/bsp_sys_init.c` `bsp/pic32mx_usb_sk2/bsp_config.h`

System Services	N/A	`system/int/src/sys_int_pic32.c`
		`system/devcon/src/sys_devcon.c`
		`system/devcon/src/pic32mx/sys_devcon_pic32mx.c`

PIC32MX USB Driver	N/A	`driver/usb/usbfs/src/dynamic/drv_usb.c` `driver/usb/usbfs/src/dynamic/drv_usb_device.c`

System Configuration (see **Note**)	N/A	`system_config.h`
		`system_init.c`
		`system_interrupt.c`
		`system_tasks.c`
		`system_definitions.h`

Peripheral Library	N/A	`<install-dir>/bin/framework/peripheral/PIC32MX795F512L_peripherals.a`

Note: These files, which are not a part of the MPLAB Harmony Framework, are application-specific and must be coded to meet the needs of the application.

Source Code Analysis

Provides code comparison information.

Description

In this section, the MLA "Device – CDC – Basic Demo" demonstration application source code will be compared to the MPLAB Harmony cdc_com_port_single demonstration application source code. The MLA "Device – CDC – Basic Demo" demonstration application logic implementation can be classified into the following components

- Fuse configuration and Initialization
- Demonstration Application Logic
- Event Handling
- Invoking the Task Routine
- USB Descriptors
- Device Stack Configuration

The following sections discuss the comparison of these components with the MPLAB Harmony USB Device Stack and MPLAB Harmony application paradigm.

Fuse Configuration and Initialization

Provides information on fuse configuration and initialization.

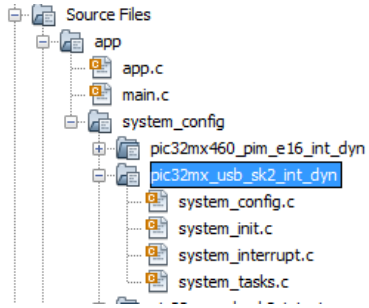
Description

The microcontroller fuse configuration in the MLA “Device – CDC – Basic Demo” demonstration application is implemented in the project `main.c` file.

Open the MLA “Device – CDC – Basic Demo” project in MPLAB X IDE and open the `main.c` file. Browse to line 277 of the file. This code region defines the fuse configuration macros while using the PIC32 USB Starter Kit II. The following screen capture shows this code.

```
277 #elif defined(PIC32_USB_STARTER_KIT)
278     #pragma config UPLLEN   = ON        // USB PLL Enabled
279     #pragma config FPLLMUL  = MUL_15        // PLL Multiplier
280     #pragma config UPLLIDIV = DIV_2         // USB PLL Input Divider
281     #pragma config FPLLIDIV = DIV_2         // PLL Input Divider
282     #pragma config FPLLODIV = DIV_1         // PLL Output Divider
283     #pragma config FPBDIV   = DIV_1         // Peripheral Clock divisor
284     #pragma config FWDTEN   = OFF           // Watchdog Timer
285     #pragma config WDTPS    = PS1           // Watchdog Timer Postscale
286     #pragma config FCKSM    = CSDCMD        // Clock Switching & Fail Safe Clock Monitor
287     #pragma config OSCIOFNC = OFF           // CLKO Enable
288     #pragma config POSCMOD  = HS            // Primary Oscillator
289     #pragma config IESO     = OFF           // Internal/External Switch-over
290     #pragma config FSOSCEN  = OFF           // Secondary Oscillator Enable (KLO was off)
291     #pragma config FNOSC    = PRIPLL        // Oscillator Selection
292     #pragma config CP       = OFF           // Code Protect
293     #pragma config BWP      = OFF           // Boot Flash Write Protect
294     #pragma config PWP      = OFF           // Program Flash Write Protect
295     #pragma config ICESEL   = ICS_PGx2      // ICE/ICD Comm Channel Select
```

In the MPLAB Harmony cdc_com_port_single project, locate the system configuration files. These files are located in the `<install-dir>/app/system_config` logical project folder. Within this folder, expand the `pic32mx_usb_sk2_int_dyn` configuration folder, as shown in the following figure.



Open `system_init.c` and review the fuse configurations. In MPLAB Harmony projects, the device fuse configuration macros are implemented in the `system_init.c` file. This file is a MPLAB Harmony system configuration file and is configuration-specific.

Continue with analysis of the MLA USB Demo `main.c` file. Navigate to line 488 of this file. This code region contains the application main function. The InitializeSystem function call at line 491 initializes the system. Further analysis of the InitializeSystem function indicates that this function (see line 648 of the file) configures the PIC32MX Cache and Flash Wait state for maximum performance for 60 MHz operating frequency, calls application specific user initialization function UserInit (see line 763) and then initializes the device stack by calling USBDeviceInit function (see line 765).

The following table lists the initialization related functions of the MLA USB Demonstration Application and their respective functional equivalent functions in the MPLAB Harmony USB Demonstration Application.

MLA USB Demonstration Application Function	Equivalent MPLAB Harmony USB Demonstration Application Function
InitializeSystem	SYS_Initialize
SYSTEMConfigPerformance	SYS_DEVCON_PerformanceConfig
UserInit	APP_Initialize
mInitAllLEDs mInitAllSwitches	BSP_Initialize
USBDeviceInit	USB_DEVICE_Initialize

The usbDevInitData and sysDevconInit data structures in MPLAB Harmony `system_init.c` are initialization data structures for the MPLAB Harmony USB Device Layer and the Device Control System Service, respectively. Please note that unlike the MLA USB Device Layer, the MPLAB Harmony USB Device Layer requires the specification of an endpoint table (see the endpointTable array declaration in `system_init.c`). The size of this table should be USB_DEVICE_ENDPOINT_TABLE_SIZE. This table (actually a byte array) should be aligned at 512 bytes address boundary.

The MPLAB Harmony Interrupt System Service functions, SYS_INT_VectorPrioritySet and SYS_INT_Enable,set the USB Interrupt Vector Priority and enable the global interrupts respectively. The USB_DEVICE_Initialize function initializes the USB Device Layer. The APP_Initialize function initializes the application data structure.

Demonstration Application Logic

Provides information on the demonstration application logic.

Description

The application logic in the MLA "Device – CDC – Basic Demo" demonstration application is implemented in the ProcessIO function (see line 819 of the project `main.c` file). This function is called periodically via the application `while(1)` loop (see line 521 of `main.c`). The ProcessIO function

calls the getsUSBUSART, putrsUSBUSART, and putUSBUSART functions to transfer data over the CDC Device Interface. The USBUSARTIsTxTrfReady function checks if the CDC function driver is ready to accept data for transmission. The CDCTxService function maintains the CDC function driver transmit state machine and is called periodically through the ProcessIO function (see line 870 of `main.c`).

In the MPLAB Harmony project, the appData object defined in the `app.h` file serves as a container for all application global data. The APP_StateReset function in the `app.c` file is called by the application and returns true if the device is not configured. This then causes the application state machine to reset.

The MPLAB Harmony application logic state machine (the APP_Tasks function) is implemented in `app.c`. The MPLAB Harmony USB Device Stack requires the application logic to be coded as a state machine that yields to the system periodically. In this state machine, the MPLAB Harmony USB Device Stack application must first open the USB Device Layer to access the functionality of the Device Layer. This is done through the USB_DEVICE_Open function. This step is not required in the MLA USB Device Stack application. The MPLAB Harmony USB Device Stack application can then wait for the device to be configured.

In the MPLAB Harmony USB Device application, the device is attached to the bus when VBUS power is detected. This is done by calling the USB_DEVICE_Attach function in the USB_DEVICE_EVENT_POWER_DETECTED function in APP_USBDeviceEventHandler function. The equivalent MLA USB Device Stack function is the USBDeviceAttach function. The USB_DEVICE_CDC_Read and USB_DEVICE_CDC_Write functions are called to transfer data over the CDC interface. The following table lists some of the MLA USB CDC function driver routines and their equivalents in the MPLAB Harmony CDC Function Driver API.

MLA USB CDC Function Driver Routine	MPLAB Harmony CDC Function Driver Routine
getsUSBUSART	USB_DEVICE_CDC_Read
putUSBUSART	USB_DEVICE_CDC_Write
putrsUSBUSART	USB_DEVICE_CDC_Write
CDCTxService	No equivalent routine. In the MPLAB Harmony USB Device Stack Framework, function driver state machine is updated by the Device Layer.
USBUSARTIsTxTrfReady	No equivalent routine. In the MPLAB Harmony USB Device Stack Framework, all data transfers are scheduled in a queue. The application does not need to check if the transmit state machine is ready. The application must use function driver events to check when a transfer is complete.

In the MLA USB CDC Function Driver, the getsUSBUSART function returns zero if no data has been received. In MPLAB Harmony CDC Function driver, the application must schedule a read with the USB_DEVICE_CDC_Read function, and then wait for the read complete event.

In the MLA USB CDC Function Driver, the application uses the USBUSARTIsTxTrfReady function to check if the CDC function is ready to transmit data. In the MPLAB Harmony CDC Function driver, the application does not need to check for transmit status. It schedules a write transfer request through the USB_DEVICE_CDC_Write function. If the request is added successfully to the queue, the application receives a valid transfer handle. The completion the transfer is indicated by an event. The MPLAB Harmony USB_DEVICE_CDC_Read and the USB_DEVICE_CDC_Write functions add transfer requests to a queue. The size of the queue is configurable. The MLA USB CDC Function driver does not support queuing.

Event Handling

Provides information on handling Device Layer events.

Description

The MLA USB Device Layer provides events to the application. In the MLA "Device – CDC – Basic Demo" demonstration application, the USB Device Layer application event handle, USER_USB_CALLBACK_EVENT_HANDLER, is implemented at line 1427 of the project `main.c` file. The MLA USB CDC function driver does not provide events to the application.

The MPLAB Harmony USB Device Layer Event handler is implemented in the APP_USBDeviceEventHandler function. The following table lists the MLA USB Device Layer Event and the equivalent MPLAB Harmony USB Device Layer Events. The MPLAB Harmony USB Device Layer features additional events that facilitate application device state management. Refer to the USB Device Layer Library documentation for more details.

MLA USB Device Layer Events	MPLAB Harmony USB Device Layer Events
EVENT_CONFIGURED	USB_DEVICE_EVENT_CONFIGURED
EVENT_SET_DESCRIPTOR	The USB Device Layer implementation in the MPLAB Harmony v0.80.xx installs Set Descriptor requests from the Host. The USB Device Layer implementation in the MPLAB Harmony v1.0 will provide this event to the application as USB_DEVICE_EVENT_SET_DESCRIPTOR event.
EVENT_EP0_REQUEST	USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST
EVENT_ATTACH	USB_DEVICE_EVENT_POWER_DETECTED
EVENT_TRANSFER_TERMINATED	USB_DEVICE_EVENT_CONTROL_TRANSFER_ABORTED
EVENT_TRANSFER	Not Applicable for standard device types. The USB Device Layer implementation in the MPLAB Harmony v1.0 will provide a USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE and a USB_DEVICE_ENDPOINT_WRITE_COMPLETE event for Generic device implementations.

EVENT_SOF	USB_DEVICE_EVENT_SOF
EVENT_RESUME	USB_DEVICE_EVENT_RESUMED
EVENT_SUSPEND	USB_DEVICE_EVENT_SUSPENDED
EVENT_BUS_ERROR	USB_DEVICE_EVENT_ERROR
EVENT_TRANSFER_TERMINATED	The USB Device Layer implementation in the MPLAB Harmony does not provide this event.

- The MLA USB Device Layer Event Handler function is fixed and needs to be specified for the code to compile. The MPLAB Harmony USB Device Layer Event Handler is Device Layer instance specific and is set through the USB_DEVICE_EventHandlerSet function. This function is called after the device layer is opened successfully in the APP_Tasks function.
- The MLA USB Device Layer Event Handler implementation must invoke the required function driver initialization routine in the EVENT_CONFIGURED event. This is not required in the MPLAB Harmony USB Device Layer Event Handler. The MPLAB Harmony USB Device Layer internally initializes all the function drivers associated with the configuration that the USB Host has set. The application should instead register event handlers with all the initialized function drivers.
- The MLA USB Device Layer Event Handler implementation must handle class specific control requests in the EVENT_EP0_REQUEST event response. The MPLAB Harmony USB Device Layer internally routes class specific requests directly to the function drivers. This process does not require application intervention. The MPLAB Harmony USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST event is generated when device layer receives a control transfer that is cannot be handled by the Device Layer or the Function Driver.

The MLA CDC Function Driver Interface is not event based and hence does not require an event handler. The MPLAB CDC Function Driver Interface is event based and requires a function driver instance specific event handler to be registered. This event handler is registered through the USB_DEVICE_CDC_EventHandlerSet function. The event handler should be registered in the Device Layer Set Configuration Event

Invoking the Tasks Routine

Provides information on how to invoke the Task Routine.

Description

Refer to line 502 of MLA "Device – CDC – Basic Demo" demonstration application file, `main.c`. This section of the code invokes the MLA USB Device Layer Task Routine, the USBDeviceTasks function, in the application `while(1)` superloop if the Device Stack is configured for Polled mode. If the Device Stack is configured for Interrupt mode operation, the USBDeviceTasks function is called (see line 664 of the `usb_device.c` file) in the USB Interrupt Service Routine. The MLA USB Device Stack features only one Task routine function.

The USB Device Layer Tasks routine in the MPLAB Harmony USB Device Stack is implemented by the USB_DEVICE_Tasks function. This function must be always be called in the SYS_Tasks function, regardless of whether the USB Device Stack is configured for Polled or Interrupt mode of operation. The MPLAB Harmony USB Device Stack defines an additional Interrupt mode specific task routine, the USB_DEVICE_Tasks_ISR function, that must be called from the USB Peripheral Interrupt Service Routine (ISR). This function should only be called in the USB Peripheral ISR. The USB Peripheral ISR is implemented in the `system_interrupt.c` file.

USB Device Descriptors

The USB Device Descriptors in the MLA "Device – CDC – Basic Demo" demonstration application are defined in the `usb_descriptors.c` file. This file contains the USB Device, Configuration, and String Descriptors.

Description

The USB Device Descriptors in the MLA “Device – CDC – Basic Demo” demonstration application are defined in the `usb_descriptors.c` file. This file contains the USB Device, Configuration and String Descriptors.

The USB Device, Configuration, and String Descriptors. USB Descriptors in a MPLAB Harmony USB Device Stack Demonstration Application are specified in the `system_init.c` file. Some of the macros (those that define endpoint and direction) used in the "Device – CDC – Basic Demo" demonstration application USB descriptors are not used in the MPLAB Harmony cdc_com_port_single USB Descriptors.

The `system_init.c` file also contains additional data structures needed by the MPLAB Harmony USB Device Stack. These are:

- Function Driver registration table (funcRegistrationTable), which provisions functions drivers to be included in this USB device
- Master Descriptor table (usbMasterDescriptor), which provides information configuration and the string descriptors to the USB Device Layer
- Table of speed specific configuration descriptors (fullSpeedConfigDescSet)
- CDC function driver initialization data structure (cdcInit). This structure contains the size of the CDC read and write, and the serial state notification send queues.

Note: The MLA USB Device layer does not require the specification of the data structures.

Device Stack Configuration

The USB Device Stack configuration in the MLA "Device – CDC – Basic Demo" demonstration application is defined in the project-specific `usb_config.h` file, located in the Project firmware folder.

Description

The USB Device Stack configuration in the MPLAB Harmony cdc_com_port_single demonstration application needs to be specified in the `system_config.h` file. This includes the configuration for the USB Controller Driver, Device Layer, and CDC Function Driver. The following table shows the MLA USB Device Stack Configuration macros and their equivalent MPLAB Harmony USB Device Stack Configuration Macros.

MLA USB Device Stack Configuration Macros	MPLAB Harmony USB Device Stack Configuration Macros
USB_EP0_BUFF_SIZE	USB_DEVICE_EP0_BUFFER_SIZE
USB_MAX_NUM_INT	The number of the interface used by a function driver instance is specified via an entry in the Function Driver Registration Table (in `system_init.c`).
USB_USER_DEVICE_DESCRIPTOR	This USB Device Descriptor is specified via the Master Descriptor Table (in `system_init.c`).
USB_PING_PONG_MODE	Not applicable. The USB Controller is always configured for maximum performance (full ping pong).
USB_POLLING USB_INTERRUPT	DRV_USB_INTERRUPT_MODE. Should be set to true for Interrupt mode and false for Polled mode.
USB_PULLUP_OPTION	Not applicable.
USB_TRANSCEIVER_OPTION	Not applicable.
USB_SPEED_OPTION	This is defined via the USB_DEVICE_INIT data structure at Device Layer Initialization in `system_init.c`.
USB_ENABLE_STATUS_STAGE_TIMEOUTS	Not applicable.
USB_SUPPORT_DEVICE	Specified via DRV_USB_DEVICE_SUPPORT and DRV_USB_HOST_SUPPORT. These can be set to true to enable the required feature.
USB_ENABLE_ALL_HANDLERS	Not applicable. All Device Layer events are provided via one event handler registered dynamically.
USB_USE_CDC	Not applicable. Function Drivers are provisioned via an entry in the Function Driver Registration Table (in `system_init.c`).
CDC_COMM_INTF_ID CDC_COMM_EP CDC_COMM_IN_EP_SIZE CDC_DATA_INTF_ID CDC_DATA_EP CDC_DATA_OUT_EP_SIZE CDC_DATA_IN_EP_SIZE	Not applicable. The number of the interface used by a function driver instance is specified via an entry in the Function Driver Registration Table (in `system_init.c`). The Device Layer obtains endpoint information for a specific instance of the CDC driver from the USB Configuration Descriptor.
USB_CDC_SUPPORT_ABSTRACT_CONTROL_MANAGEMENT_CAPABILITIES_D1	Not applicable. Features are defined by the CDC Class specific descriptors.

The MPLAB Harmony USB Device Stack requires specification of other configuration macros. Refer to the USB Device Library and the USB Driver Libraries sections for additional information.

USB Audio 1.0 Device Library

This section describes the USB Audio 1.0 Device Library.

Introduction

This section provides information on library design, configuration, usage and the library interface for the USB Audio 1.0 Device Library.

Description

The MPLAB Harmony USB Audio 1.0 Device Library (also referred to as the Audio 1.0 Function Driver or library) features routines to implement a USB Audio 1.0 Device. Examples of Audio USB Devices include USB Speakers, microphones, and voice telephony. The library provides a convenient abstraction of the USB Audio 1.0 Device specification and simplifies the implementation of USB Audio 1.0 Devices.

Using the Library

This topic describes the basic architecture of the Audio 1.0 Function Driver and provides information and examples on its use.

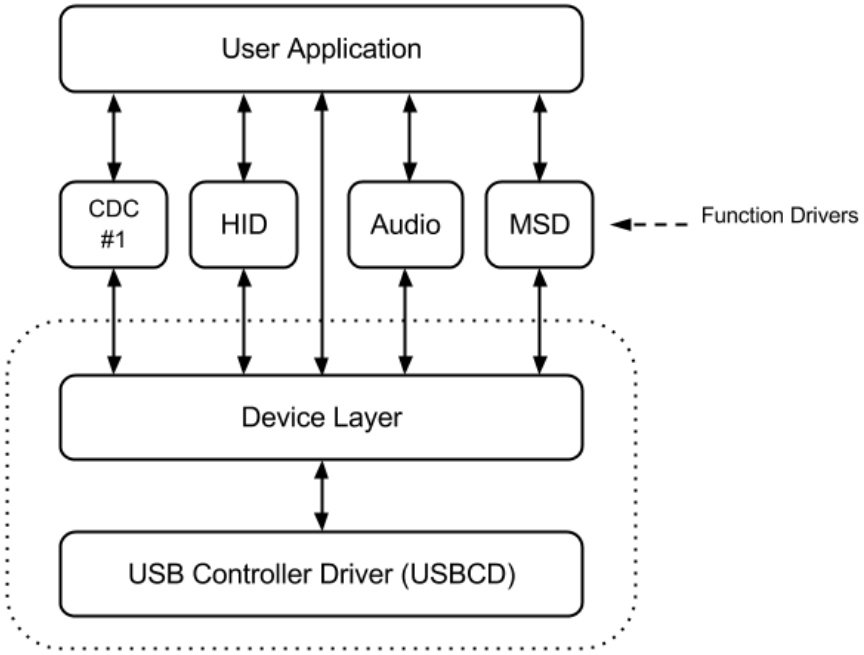
Abstraction Model

Describes the Abstraction Model of the USB Audio 1.0 Device Library.

Description

The Audio 1.0 Function Driver offers various services to the USB Audio 1.0 device to communicate with the host by abstracting USB specification details. It must be used along with the USB Device layer and USB controller to communicate with the USB host. Figure 1 shows a block diagram of the MPLAB Harmony USB Device Stack Architecture and where the Audio 1.0 Function Driver is placed.

Figure 1: USB Device 1.0 Audio Device Driver



The USB controller driver takes the responsibility of managing the USB peripheral on the device. The USB 1.0 Device Layer handles the device enumeration, etc. The USB Device 1.0 Layer forwards all Audio-specific control transfers to the Audio 1.0 Function Driver. The Audio 1.0 Function Driver interprets the control transfers and requests application's intervention through event handlers and a well-defined set of functions. The application must respond to the Audio events either in or out of the event handler. Some of these events are related to Audio 1.0 Device Class specific control transfers. The application must complete these control transfers within the timing constraints defined by USB.

Library Overview

The USB Audio 1.0 Device Library mainly interacts with the system, its clients and function drivers, as shown in the Abstraction Model.

The library interface routines are divided into sub-sections, which address one of the blocks or the overall operation of the USB Audio 1.0 Device Library.

Library Interface Section	Description
Functions	Provides event handler, read/write, and transfer cancellation functions.

How the Library Works

Initializing the Library

Describes how the USB Audio 1.0 Device driver is initialized.

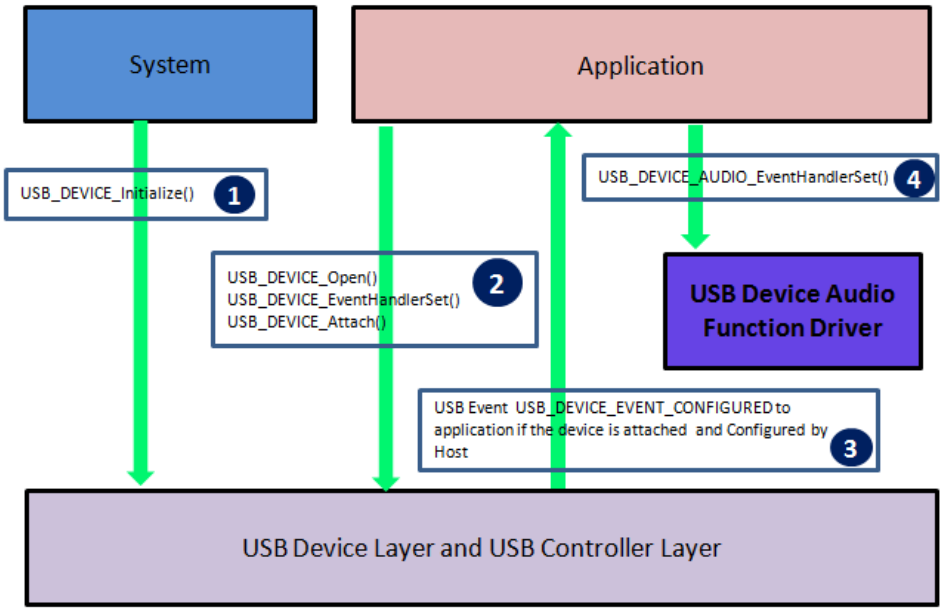
Description

The Audio 1.0 Function Driver instance for a USB device configuration is initialized by the Device Layer when the configuration is set by the host. This process does not require application intervention. Each instance of the Audio 1.0 Function Driver should be registered with the Device layer through the Device Layer Function Driver Registration Table. The Audio 1.0 Function Driver requires a initialization data structure to be specified. This is a USB_DEVICE_AUDIO_INIT data type that specifies the size of the read and write queues. The funcDriverInit member of the function driver registration table entry of the Audio 1.0 Function Driver instance should point to this initialization data structure. The USB_DEVICE_AUDIO_FUNCTION_DRIVER object is a global object provided by the Audio 1.0 Function Driver and provides the Device Layer with an entry point into the Audio 1.0 Function Driver. The following code shows an example of how the Audio 1.0 Function Driver can be registered with the Device Layer.

```
/* This code shows an example of how an Audio 1.0 Function Driver instances
 * can be registered with the Device Layer via the Device Layer Function Driver
 * Registration Table.  In this case Device Configuration 1 consists of one
 * Audio 1.0 Function Driver instance. */

/* The Audio 1.0 Function Driver requires an initialization data structure that
 * specifies the read and write buffer queue sizes. Note that these settings are
 * also affected by the USB_DEVICE_AUDIO_QUEUE_DEPTH_COMBINED configuration
 * macro. */

const USB_DEVICE_AUDIO_INIT audioDeviceInit =
{
    .queueSizeRead = 1,
    .queueSizeWrite = 1
};

const USB_DEVICE_FUNC_REGISTRATION_TABLE funcRegistrationTable[1] =
{
    {
        .speed = USB_SPEED_FULL,                    // Supported speed
        .configurationValue = 1,                    // To be initialized for Configuration 1
        .interfaceNumber = 0,                       // Starting interface number.
        .numberOfInterfaces = 2,                    // Number of interfaces in this instance
        .funcDriverIndex = 0,                       // Function Driver instance index is 0
        .funcDriverInit = &audioDeviceInit,         // Function Driver does not need initialization data
structure
        .driver = USB_DEVICE_AUDIO_FUNCTION_DRIVER  // Pointer to Function Driver - Device Layer interface
functions
    },
};
```

The following figure illustrates the typical sequence that is followed in the application when using the Audio 1.0 Function Driver.

Typical USB Audio 1.0 Device Sequence



1. Call set of APIs to initialize USB Device Layer (refer to the USB Device Layer Library section for details about these APIs).
2. The Device Layer provides a callback to the application for any USB Device events like attached, powered, configured, etc. The application should receive a callback with an event USB_DEVICE_EVENT_CONFIGURED to proceed.
3. Once the Device Layer is configured, the application needs to register a callback function with the Audio 1.0 Function Driver to receive Audio Control transfers, and also other Audio 1.0 Function Driver events. Now the application can use Audio 1.0 Function Driver APIs to communicate with the USB Host.

Event Handling

Describes Audio 1.0 Function Driver event handler registration and event handling.

Description

Registering a Audio 1.0 Function Driver Callback Function

While creating a USB Audio 1.0 Device application, an event handler must be registered with the Device Layer (the Device Layer Event Handler) and every Audio 1.0 Function Driver instance (Audio 1.0 Function Driver Event Handler). The application needs to register the event handler with the Audio 1.0 Function Driver:

- For receiving Audio Control Requests from Host like Volume Control, Mute Control, etc.
- For handling other events from USB Audio 1.0 Device Driver (e.g., Data Write Complete or Data Read Complete)

The event handler should be registered before the USB device layer acknowledges the SET CONFIGURATION request from the USB Host. To ensure this, the callback function should be set in the USB_DEVICE_EVENT_CONFIGURED event that is generated by the device layer. The following code example shows how this can be done.

```
/* This a sample Application Device Layer Event Handler
 * Note how the USB Audio 1.0 Device Driver callback function
 * USB_DEVICE_AUDIO_EventHandlerSet()
 * is registered in the USB_DEVICE_EVENT_CONFIGURED event.  */

void APP_USBDeviceEventHandler( USB_DEVICE_EVENT event,
                                void * pEventData, uintptr_t context )
{
    switch ( event )
    {
        case USB_DEVICE_EVENT_RESET:
        case USB_DEVICE_EVENT_DECONFIGURED:

            // USB device is reset or device is deconfigured.
            // This means that USB device layer is about to deinitialize
            // all function drivers.

            break;

        case USB_DEVICE_EVENT_CONFIGURED:
```

```
            /* check the configuration */
            if ( ((USB_DEVICE_EVENT_DATA_CONFIGURED *)
                  (eventData))->configurationValue == 1)
            {

                USB_DEVICE_AUDIO_EventHandlerSet
                     ( USB_DEVICE_AUDIO_INDEX_0,
                       APP_USBDeviceAudioEventHandler ,
                       (uintptr_t)NULL);

                /* mark that set configuration is complete */
                appData.isConfigured = true;

            }
            break;

        case USB_DEVICE_EVENT_SUSPENDED:

            break;

        case USB_DEVICE_EVENT_RESUMED:
        case USB_DEVICE_EVENT_POWER_DETECTED:
        /* VBUS has been detected */
        USB_DEVICE_Attach(appData.usbDeviceHandle);
    break;
        case USB_DEVICE_EVENT_POWER_REMOVED:
        /*VBUS is not available anymore. */
        USB_DEVICE_Detach(appData.usbDeviceHandle);
    break;
        case USB_DEVICE_EVENT_ERROR:
        default:
            break;
    }
}
```

Event Handling

The Audio 1.0 Function Driver provides events to the application through the event handler function registered by the application. These events indicate:

- Completion of a read or a write data transfer
- Audio Control Interface requests
- Completion of data and the status stages of Audio Control Interface related control transfer

The Audio Control Interface Request events and the related control transfer events typically require the application to respond with the Device Layer Control Transfer routines to complete the control transfer. Based on the generated event, the application may be required to:

- Respond with a USB_DEVICE_ControlSend function, which is completes the data stage of a Control Read Transfer
- Respond with a USB_DEVICE_ControlReceive function, which provisions the data stage of a Control Write Transfer
- Respond with a USB_DEVICE_ControlStatus function, which completes the handshake stage of the Control Transfer. The application can either STALL or Acknowledge the handshake stage through the USB_DEVICE_ControlStatus function.

The following table shows the CDC Function Driver Control Transfer related events and the required application control transfer actions.

Audio 1.0 Function Driver Control Transfer Event	Required Application Action
USB_DEVICE_AUDIO_EVENT_CONTROL_SET_CUR USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MIN USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MAX USB_DEVICE_AUDIO_EVENT_CONTROL_SET_RES USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MEM	Identify the control type using the associated event data. If a data stage is expected, use the USB_DEVICE_ControlReceive function to receive expected data. If a data stage is not required or if the request is not supported, use the USB_DEVICE_ControlStatus function to Acknowledge or Stall the request.
USB_DEVICE_AUDIO_EVENT_CONTROL_GET_CUR USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MIN USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MAX USB_DEVICE_AUDIO_EVENT_CONTROL_GET_RES USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MEM	Identify the control type using the associated event data. Use the USB_DEVICE_ControlSend function to send the expected data. If the request is not supported, use the USB_DEVICE_ControlStatus function to Stall the request.
USB_DEVICE_AUDIO_EVENT_ENTITY_GET_STAT	Identify the entity type using the associated event data. Use the USB_DEVICE_ControlSend function to send the expected data. If the request is not supported, use the USB_DEVICE_ControlStatus function to Stall the request.

USB_DEVICE_AUDIO_CONTROL_TRANSFER_DATA_RECEIVED	Acknowledge or stall using the USB_DEVICE_ControlStatus function.
USB_DEVICE_AUDIO_CONTROL_TRANSFER_DATA_SENT	Action not required.
USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_UNKNOWN	Interpret the setup packet and use the Device layer Control transfer functions to complete the transfer.

The application must analyze the wIndex field of the event data (received with the control transfer event) to identify the entity that is being addressed. The application must be aware of all entities included in the application and their IDs. Once identified, the application can then type cast the event data to entity type the specific control request type. For example, if the Host sends a control request to set the volume of the Audio device, the following occurs in this order:

1. The Audio 1.0 Function Driver will generate a USB_DEVICE_AUDIO_EVENT_CONTROL_SET_CUR event.
2. The application must type cast the event data to a USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_CUR type and check the entityID field.
3. The entityID field will be identified by the application as a Feature Unit.
4. The application must now type cast the event data type as a USB_AUDIO_FEATURE_UNIT_CONTROL_REQUEST data type and check the controlSelector field.
5. If the controlSelector field is a USB_AUDIO_VOLUME_CONTROL, the application can then call the USB_DEVICE_AUDIO_ControlReceive function to receive the new volume settings.

Based on the type of event, the application should analyze the event data parameter of the event handler. This data member should be type cast to an event specific data type. The following table shows the event and the data type to use while type casting. Note that the event data member is not required for all events

Audio 1.0 Function Driver Event	Related Event Data Type
USB_DEVICE_AUDIO_EVENT_CONTROL_SET_CUR	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_CUR *
USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MIN	USB_DEVICE_AUDIO_EVENT_ DATA_CONTROL_SET_MIN *
USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MAX	USB_DEVICE_AUDIO_EVENT_ DATA_CONTROL_SET_MAX *
USB_DEVICE_AUDIO_EVENT_CONTROL_SET_RES	USB_DEVICE_AUDIO_EVENT_ DATA_CONTROL_SET_RES *
USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MEM	USB_DEVICE_AUDIO_EVENT_ DATA_CONTROL_SET_MEM *
USB_DEVICE_AUDIO_EVENT_CONTROL_GET_CUR	USB_DEVICE_AUDIO_EVENT_ DATA_CONTROL_GET_CUR *
USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MIN	USB_DEVICE_AUDIO_EVENT_ DATA_CONTROL_GET_MIN *
USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MAX	USB_DEVICE_AUDIO_EVENT_ DATA_CONTROL_GET_MAX *
USB_DEVICE_AUDIO_EVENT_CONTROL_GET_RES	USB_DEVICE_AUDIO_EVENT_CONTROL_GET_RES
USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MEM	USB_DEVICE_AUDIO_EVENT_ DATA_CONTROL_GET_MEM *
USB_DEVICE_AUDIO_EVENT_ENTITY_GET_STAT	USB_DEVICE_AUDIO_EVENT_ DATA_ENTITY_GET_STAT *
USB_DEVICE_AUDIO_CONTROL_TRANSFER_DATA_RECEIVED	NULL
USB_DEVICE_AUDIO_CONTROL_TRANSFER_DATA_SENT	NULL
USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_UNKNOWN	USB_SETUP_PACKET *
USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE	USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE *
USB_DEVICE_AUDIO_EVENT_READ_COMPLETE	USB_DEVICE_AUDIO_EVENT_ DATA_READ_COMPLETE *
USB_DEVICE_AUDIO_EVENT_INTERFACE_SETTING_CHANGED	USB_DEVICE_AUDIO_EVENT_ DATA_INTERFACE_SETTING_CHANGED *

Handling Audio Control Requests:

When the Audio 1.0 Function Driver receives an Audio Class Specific Control Transfer Request, it passes this control transfer to the application as a Audio 1.0 Function Driver event. The following code example shows how to handle an Audio Control request.

```
// This code example shows handling Audio Control requests. The following code
// handles a Mute request (both SET and GET) received from a USB Host.

void APP_USBDeviceAudioEventHandler
(
    USB_DEVICE_AUDIO_INDEX iAudio ,
    USB_DEVICE_AUDIO_EVENT event ,
    void * pData,
    uintptr_t context
)
{
    USB_DEVICE_AUDIO_EVENT_DATA_INTERFACE_SETTING_CHANGED *interfaceInfo;
    USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE *readEventData;
```

```
uint8_t entityID;
uint8_t controlSelector;
if ( iAudio == 0 )
{
    switch (event)
    {
        case USB_DEVICE_AUDIO_EVENT_INTERFACE_SETTING_CHANGED:

            /* We have received a request from USB host to change the
             * Interface-Alternate setting. The application should be aware
             * of the association between alternate settings and the device
             * features to be enabled.*/

            interfaceInfo = (USB_DEVICE_AUDIO_EVENT_DATA_INTERFACE_SETTING_CHANGED *)pData;
            appData.activeInterfaceAlternateSetting = interfaceInfo->interfaceAlternateSetting;
            appData.state = APP_USB_INTERFACE_ALTERNATE_SETTING_RCVD;
            break;

        case USB_DEVICE_AUDIO_EVENT_READ_COMPLETE:
            /* We have received an audio frame from the Host.
               Now send this audio frame to Audio Codec for Playback. */
            break;

        case USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE:
            break;

        case USB_DEVICE_AUDIO_EVENT_CONTROL_SET_CUR:

            /* This is an example of handling Audio control request. In this
             * case the control request is targeted to the Mute Control in
             * a feature unit entity. This event indicates that the current
             * value needs to be set. */

            entityID = ((USB_AUDIO_CONTROL_INTERFACE_REQUEST*)pData)->entityID;
            if (entityID == APP_ID_FEATURE_UNIT)
            {
                controlSelector = ((USB_AUDIO_FEATURE_UNIT_CONTROL_REQUEST*)pData)->controlSelector;
                if (controlSelector == USB_AUDIO_MUTE_CONTROL)
                {
                    /* It is confirmed that this request is targeted to the
                     * mute control. We schedule a control transfer receive
                     * to get data from the host. */

                    USB_DEVICE_ControlReceive(appData.usbDevHandle, (void *) &(appData.dacMute), 1 );
                    appData.currentAudioControl = APP_USB_AUDIO_MUTE_CONTROL;
                }
            }
            break;

        case USB_DEVICE_AUDIO_EVENT_CONTROL_GET_CUR:

            /* This event occurs when the host is requesting a current
             * status of control */

            entityID = ((USB_AUDIO_CONTROL_INTERFACE_REQUEST*)pData)->entityID;
            if (entityID == APP_ID_FEATURE_UNIT)
            {
                controlSelector = ((USB_AUDIO_FEATURE_UNIT_CONTROL_REQUEST*)pData)->controlSelector;
                if (controlSelector == USB_AUDIO_MUTE_CONTROL)
                {
                    /* Use the control send function to send the status of
                     * the control to the host */
                    USB_DEVICE_ControlSend(appData.usbDevHandle, (void *)&(appData.dacMute), 1 );
                }
            }
            break;

        case USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MIN:
        case USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MIN:
```

```
            case USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MAX:
            case USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MAX:
            case USB_DEVICE_AUDIO_EVENT_CONTROL_SET_RES:
            case USB_DEVICE_AUDIO_EVENT_CONTROL_GET_RES:
            case USB_DEVICE_AUDIO_EVENT_ENTITY_GET_MEM:

                /* In this example, all of these control requests are not
                 * supported. So these are stalled. */
                USB_DEVICE_ControlStatus (appData.usbDevHandle, USB_DEVICE_CONTROL_STATUS_ERROR);
                break;

            case USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:

                /* This event occurs when data has been received in a control
                 * transfer */

                USB_DEVICE_ControlStatus(appData.usbDevHandle, USB_DEVICE_CONTROL_STATUS_OK );
                if (appData.currentAudioControl == APP_USB_AUDIO_MUTE_CONTROL)
                {
                    appData.state = APP_MUTE_AUDIO_PLAYBACK;
                    appData.currentAudioControl = APP_USB_CONTROL_NONE;
                }
                break;

            case  USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_DATA_SENT:
                break;
            default:
                break;
        }
    }
}
```

Transferring Data

Describes how to send/receive data to/from USB Host using this USB Audio 1.0 Device Driver.

Description

The USB Audio 1.0 Device Driver provides functions to send and receive data.

Receiving Data

The USB_DEVICE_AUDIO_Read function schedules a data read. When the host transfers data to the device, the Audio 1.0 Function Driver receives the data and invokes the USB_DEVICE_AUDIO_EVENT_READ_COMPLETE event. This event indicates that audio data is now available in the application specified buffer.

The Audio 1.0 Function Driver supports buffer queuing. The application can schedule multiple read requests. Each request is assigned a unique buffer handle, which is returned with the USB_DEVICE_AUDIO_EVENT_READ_COMPLETE event. The application can use the buffer handle to track completion to queued requests. Using this feature allows the application to implement audio buffering schemes such as ping-pong buffering.

Sending Data

The USB_DEVICE_AUDIO_Write schedules a data write. When the host sends a request for the data, the Audio 1.0 Function Driver transfers the data and invokes the USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE event.

The Audio 1.0 Function Driver supports buffer queuing. The application can schedule multiple write requests. Each request is assigned a unique buffer handle, which is returned with the USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE event. The application can use the buffer handle to track completion to queued requests. Using this feature allows the application to implement audio buffering schemes such as ping-pong buffering.

Configuring the Library

Describes how to configure the USB Audio 1.0 Device Driver.

Macros

	Name	Description
	USB_DEVICE_AUDIO_INSTANCES_NUMBER	Specifies the number of Audio Function Driver instances.
	USB_DEVICE_AUDIO_MAX_ALTERNATE_SETTING	Specifies the maximum number of Alternate Settings per streaming interface.

	USB_DEVICE_AUDIO_MAX_STREAMING_INTERFACES	Specifies the maximum number of Audio Streaming interfaces in an Audio Function Driver instance.
	USB_DEVICE_AUDIO_QUEUE_DEPTH_COMBINED	Specifies the combined queue size of all Audio function driver instances.

Description

The application designer must specify the following configuration parameters while using the USB Audio 1.0 Device Driver. The configuration macros that implement these parameters must be located in the `system_config.h` file in the application project and a compiler include path (to point to the folder that contains this file) should be specified.

USB_DEVICE_AUDIO_INSTANCES_NUMBER Macro

Specifies the number of Audio Function Driver instances.

File

usb_device_audio_v1_0_config_template.h

C

```
#define USB_DEVICE_AUDIO_INSTANCES_NUMBER
```

Description

USB device Audio Maximum Number of instances

This macro defines the number of instances of the Audio Function Driver. For example, if the application needs to implement two instances of the Audio Function Driver (to create two composite Audio Device) on one USB Device, the macro should be set to 2. Note that implementing a USB Device that features multiple Audio interfaces requires appropriate USB configuration descriptors.

Remarks

None.

USB_DEVICE_AUDIO_MAX_ALTERNATE_SETTING Macro

Specifies the maximum number of Alternate Settings per streaming interface.

File

usb_device_audio_v1_0_config_template.h

C

```
#define USB_DEVICE_AUDIO_MAX_ALTERNATE_SETTING
```

Description

Maximum number of Alternate Settings

This macro defines the maximum number of Alternate Settings per streaming interface. If the Audio Device features multiple streaming interfaces, this configuration constant should be equal to the the maximum number of alternate required amongst the streaming interfaces.

Remarks

None.

USB_DEVICE_AUDIO_MAX_STREAMING_INTERFACES Macro

Specifies the maximum number of Audio Streaming interfaces in an Audio Function Driver instance.

File

usb_device_audio_v1_0_config_template.h

C

```
#define USB_DEVICE_AUDIO_MAX_STREAMING_INTERFACES
```

Description

Maximum Audio Streaming Interfaces

This macro defines the maximum number of streaming interfaces in an Audio Function Driver instance. In case of multiple Audio Function Driver instances, this constant should be equal to the maximum number of interfaces amongst the Audio Function Driver instances.

Remarks

None.

USB_DEVICE_AUDIO_QUEUE_DEPTH_COMBINED Macro

Specifies the combined queue size of all Audio function driver instances.

File

usb_device_audio_v1_0_config_template.h

C

```
#define USB_DEVICE_AUDIO_QUEUE_DEPTH_COMBINED
```

Description

USB device Audio Combined Queue Size

This macro defines the number of entries in all queues in all instances of the Audio function driver. This value can be obtained by adding up the read and write queue sizes of each Audio Function driver instance. In a simple single instance USB Audio device application, that does not require buffer queuing, the USB_DEVICE_AUDIO_QUEUE_DEPTH_COMBINED macro can be set to 2. Consider a case of a Audio function driver instances, with has a read queue size of 2 and write queue size of 3, this macro should be set to 5 (2 + 3).

Remarks

None.

Building the Library

This section lists the files that are available in the USB Audio 1.0 Device Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/usb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
usb_device_audio_v1_0.h	This header file must be included in every source file that needs to invoke USB Audio 1.0 Device Driver APIs.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/usb_device_audio_v1_0.c	This file contains all of functions, macros, definitions, variables, datatypes, etc., that are specific to the USB Audio Specification v1.0 implementation of the Audio 1.0 Function Driver.
/src/dynamic/usb_device_audio_read_write.c	Contains implementation of the Audio 1.0 Function Driver read and write functions.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The USB Audio 1.0 Device Library depends on the following modules:

- USB Device Library

Library Interface

a) Functions

	Name	Description
	USB_DEVICE_AUDIO_EventHandlerSet	This function registers an event handler for the specified Audio function driver instance.
	USB_DEVICE_AUDIO_Read	This function requests a data read from the USB Device Audio Function Driver Layer.
	USB_DEVICE_AUDIO_TransferCancel	This function cancels a scheduled Audio Device data transfer.
	USB_DEVICE_AUDIO_Write	This function requests a data write to the USB Device Audio Function Driver Layer.

b) Data Types and Constants

	Name	Description
	USB_DEVICE_AUDIO_EVENT_RESPONSE_NONE	USB Device Audio Function Driver event handler response type none.
	USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID	USB Device Audio Function Driver invalid transfer handle definition.
	USB_DEVICE_AUDIO_EVENT	USB Device Audio Function Driver events.
	USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE	USB Device Audio Function Driver audio read and write complete event data.
	USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE	USB Device Audio Function Driver audio read and write complete event data.
	USB_DEVICE_AUDIO_EVENT_HANDLER	USB Device Audio event handler function pointer type.
	USB_DEVICE_AUDIO_EVENT_RESPONSE	USB Device Audio Function Driver event callback response type.
	USB_DEVICE_AUDIO_INDEX	USB Device Audio function driver index.
	USB_DEVICE_AUDIO_RESULT	USB Device Audio Function Driver USB Device Audio result enumeration.
	USB_DEVICE_AUDIO_TRANSFER_HANDLE	USB Device Audio Function Driver transfer handle definition.
	USB_DEVICE_AUDIO_FUNCTION_DRIVER	USB Device Audio Function Driver function pointer.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_CUR	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_CUR.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MAX	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MAX.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MEM	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MEM.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MIN	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MIN.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_RES	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_RES.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_CUR	USB Device Audio Function Driver set and get request data.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MAX	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MAX.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MEM	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MEM.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MIN	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MIN.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_RES	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_RES.

	USB_DEVICE_AUDIO_EVENT_DATA_INTERFACE_SETTING_CHANGED	USB Device Audio Function Driver alternate interface setting event data.
	USB_DEVICE_AUDIO_INIT	USB Device Audio Function Driver initialization data structure.
	USB_DEVICE_AUDIO_EVENT_DATA_ENTITY_GET_STAT	This is type USB_DEVICE_AUDIO_EVENT_DATA_ENTITY_GET_STAT.
	USB_DEVICE_AUDIO_TRANSFER_ABORT_NOTIFY	USB Audio Transfer abort notification enable

Description

This section describes the Application Programming Interface (API) functions of the USB Device Audio 1.0 Library.

Refer to each section for a detailed description.

a) Functions

USB_DEVICE_AUDIO_EventHandlerSet Function

This function registers an event handler for the specified Audio function driver instance.

File

usb_device_audio_v1_0.h

C

```
USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_EventHandlerSet(USB_DEVICE_AUDIO_INDEX instanceIndex,
USB_DEVICE_AUDIO_EVENT_HANDLER eventHandler, uintptr_t context);
```

Returns

- USB_DEVICE_AUDIO_RESULT_OK - The operation was successful
- USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.
- USB_DEVICE_AUDIO_RESULT_ERROR_PARAMETER_INVALID - The eventHandler parameter is NULL

Description

This function registers a event handler for the specified Audio function driver instance. This function should be called by the application when it receives a SET CONFIGURATION event from the device layer. The application must register an event handler with the function driver in order to receive and respond to function driver specific events and control transfers. If the event handler is not registered, the device layer will stall function driver specific commands and the USB device may not function.

Remarks

None.

Preconditions

This function should be called when the function driver has been initialized as a result of a set configuration.

Example

```
// The following code shows an example for registering an event handler. The
// application specifies the context parameter as a pointer to an
// application object (appObject) that should be associated with this
// instance of the Audio function driver.

USB_DEVICE_AUDIO_RESULT result;

USB_DEVICE_AUDIO_EVENT_RESPONSE APP_USBDeviceAUDIOEventHandler
(
    USB_DEVICE_AUDIO_INDEX instanceIndex ,
    USB_DEVICE_AUDIO_EVENT event ,
    void* pData,
    uintptr_t context
)
{
    // Event Handling comes here

    switch(event)
    {
```

```
            ...
        }

        return(USB_DEVICE_AUDIO_EVENT_RESPONSE_NONE);
    }

    result = USB_DEVICE_AUDIO_EventHandlerSet ( USB_DEVICE_AUDIO_INSTANCE_0 ,
                &APP_USBDeviceAUDIOEventHandler, (uintptr_t) &appObject);

    if(USB_DEVICE_AUDIO_RESULT_OK != result)
    {
        // Do error handling here
    }
```

Parameters

Parameters	Description
instance	Instance of the Audio Function Driver.
eventHandler	A pointer to event handler function.
context	Application specific context that is returned in the event handler.

Function

USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_EventHandlerSet
(
USB_DEVICE_AUDIO_INDEX instance ,
USB_DEVICE_AUDIO_EVENT_HANDLER eventHandler ,
uintptr_t context
);

USB_DEVICE_AUDIO_Read Function

This function requests a data read from the USB Device Audio Function Driver Layer.

File

usb_device_audio_v1_0.h

C

```
USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_Read(USB_DEVICE_AUDIO_INDEX instanceIndex,
USB_DEVICE_AUDIO_TRANSFER_HANDLE* transferHandle, uint8_t interfaceNumber, void * data, size_t size);
```

Returns

- USB_DEVICE_AUDIO_RESULT_OK - The read request was successful. transferHandle contains a valid transfer handle.
- USB_DEVICE_AUDIO_RESULT_ERROR_TRANSFER_QUEUE_FULL - internal request queue is full. The read request could not be added.
- USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.
- USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application and is invalid.

Description

This function requests a data read from the USB Device Audio Function Driver Layer. The function places a requests with driver, the request will get serviced as data is made available by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_AUDIO_EVENT_READ_COMPLETE event. The amount of data read and the transfer handle associated with the request is returned along with the event. The transfer handle expires when event handler for the USB_DEVICE_AUDIO_EVENT_READ_COMPLETE exits. If the read request could not be accepted, the function returns an error code and transferHandle will contain the value USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID.

Remarks

While the using the Audio Function Driver with PIC32MZ USB module, the audio buffer provided to the USB_DEVICE_AUDIO_Read function should be placed in coherent memory and aligned at a 16 byte boundary. This can be done by declaring the buffer using the __attribute__((coherent, aligned(16))) attribute. An example is shown here

```
uint8_t data[256] __attribute__((coherent, aligned(16)));
```

Preconditions

The function driver should have been configured.

Example

```
// Shows an example of how to read. This assumes that
// device had been configured. The example attempts to read
// data from interface 1.

USB_DEVICE_AUDIO_INDEX instanceIndex;
USB_DEVICE_AUDIO_TRANSFER_HANDLE transferHandle;
unit8_t interfaceNumber;
unit8_t rxBuffer[192]; // Use this attribute for PIC32MZ __attribute__((coherent, aligned(16)))
USB_DEVICE_AUDIO_RESULT readRequestResult;

instanceIndex = 0; //specify the Audio Function driver instance number.
interfaceNumber = 1; //Specify the Audio Streaming interface number.

readRequestResult = USB_DEVICE_AUDIO_Read ( instanceIndex, &transferHandle,
                        interfaceNumber, &rxBuffer, 192);

if(USB_DEVICE_AUDIO_RESULT_OK != readRequestResult)
{
    //Do Error handling here
}

// The completion of the read request will be indicated by the
// USB_DEVICE_AUDIO_EVENT_READ_COMPLETE event. The transfer handle
// and the amount of data read will be returned along with the
// event.
```

Parameters

Parameters	Description
instance	USB Device Audio Function Driver instance.
transferHandle	Pointer to a USB_DEVICE_AUDIO_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the read request was successful.
interfaceNum	The USB Audio streaming interface number on which read request is to placed.
data	pointer to the data buffer where read data will be stored. In case of PIC32MZ device, this buffer should be located in coherent memory and should be aligned a 16 byte boundary.
size	Size of the data buffer. Refer to the description section for more details on how the size affects the transfer.

Function

```
 USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_Read
(
    USB_DEVICE_AUDIO_INDEX instanceIndex ,
    USB_DEVICE_AUDIO_TRANSFER_HANDLE* transferHandle,
uint8_t interfaceNum ,
void * data ,
size_t size
);
```

USB_DEVICE_AUDIO_TransferCancel Function

This function cancels a scheduled Audio Device data transfer.

File

usb_device_audio_v1_0.h

C

```
USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_TransferCancel(USB_DEVICE_AUDIO_INDEX instanceIndex,
```

```
USB_DEVICE_AUDIO_TRANSFER_HANDLE transferHandle);
```

Returns

- USB_DEVICE_AUDIO_RESULT_OK - The transfer will be canceled completely or partially.
- USB_DEVICE_AUDIO_RESULT_ERROR - The transfer could not be canceled because it has either completed, the transfer handle is invalid or the last transaction is in progress.

Description

This function cancels a scheduled Audio Device data transfer. The transfer could have been scheduled using the USB_DEVICE_AUDIO_Read, USB_DEVICE_AUDIO_Write, or the USB_DEVICE_AUDIO_SerialStateNotificationSend functions. If a transfer is still in the queue and its processing has not started, the transfer is canceled completely. A transfer that is in progress may or may not get canceled depending on the transaction that is presently in progress. If the last transaction of the transfer is in progress, the transfer will not be canceled. If it is not the last transaction in progress, the in-progress will be allowed to complete. Pending transactions will be canceled. The first transaction of an in progress transfer cannot be canceled.

Remarks

None.

Preconditions

The USB Device should be in a configured state.

Example

```
// The following code snippet cancels a AUDIO transfer.

USB_DEVICE_AUDIO_TRANSFER_HANDLE transferHandle;
USB_DEVICE_AUDIO_RESULT result;

result = USB_DEVICE_AUDIO_TransferCancel(instanceIndex, transferHandle);

if(USB_DEVICE_AUDIO_RESULT_OK == result)
{
    // The transfer cancellation was either completely or
    // partially successful.
}
```

Parameters

Parameters	Description
instanceIndex	AUDIO Function Driver instance index.
transferHandle	Transfer handle of the transfer to be canceled.

Function

```
USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_TransferCancel
(
    USB_DEVICE_AUDIO_INDEX instanceIndex,
    USB_DEVICE_AUDIO_TRANSFER_HANDLE transferHandle
);
```

USB_DEVICE_AUDIO_Write Function

This function requests a data write to the USB Device Audio Function Driver Layer.

File

usb_device_audio_v1_0.h

C

```
USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_Write(USB_DEVICE_AUDIO_INDEX instanceIndex,
USB_DEVICE_AUDIO_TRANSFER_HANDLE * transferHandle, uint8_t interfaceNumber, void * data, size_t size);
```

Returns

- USB_DEVICE_AUDIO_RESULT_OK - The read request was successful. transferHandle contains a valid transfer handle.
- USB_DEVICE_AUDIO_RESULT_ERROR_TRANSFER_QUEUE_FULL - internal request queue is full. The write request could not be added.

- USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.
- USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application and is invalid.

Description

This function requests a data write to the USB Device Audio Function Driver Layer. The function places a requests with driver, the request will get serviced as data is requested by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE event. The amount of data written and the transfer handle associated with the request is returned along with the event in writeCompleteData member of the pData parameter in the event handler.

The transfer handle expires when event handler for the USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE exits. If the write request could not be accepted, the function returns an error code and transferHandle will contain the value USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID.

Remarks

While the using the Audio Function Driver with the PIC32MZ USB module, the audio buffer provided to the USB_DEVICE_AUDIO_Write function should be placed in coherent memory and aligned at a 16 byte boundary. This can be done by declaring the buffer using the __attribute__((coherent, aligned(16))) attribute. An example is shown here

```
uint8_t data[256] __attribute__((coherent, aligned(16)));
```

Preconditions

The function driver should have been configured.

Example

```
// Shows an example of how to write audio data to the audio streaming
// interface . This assumes that device is configured and the audio
// streaming interface is 1.

USB_DEVICE_AUDIO_INDEX instanceIndex;
USB_DEVICE_AUDIO_TRANSFER_HANDLE transferHandle;
unit8_t interfaceNumber;
unit8_t txBuffer[192]; // Use this attribute for PIC32MZ __attribute__((coherent, aligned(16)))
USB_DEVICE_AUDIO_RESULT writeRequestResult;

instanceIndex = 0; //specify the Audio Function driver instance number.
interfaceNumber = 1; //Specify the Audio Streaming interface number.

writeRequestResult = USB_DEVICE_AUDIO_Write ( instanceIndex, &transferHandle,
                            interfaceNumber, &txBuffer, 192);

if(USB_DEVICE_AUDIO_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

// The completion of the write request will be indicated by the
// USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE event. The transfer handle
// and transfer size is provided along with this event.
```

Parameters

Parameters	Description
instance	USB Device Audio Function Driver instance.
transferHandle	Pointer to a USB_DEVICE_AUDIO_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the write request was successful.
interfaceNum	The USB Audio streaming interface number on which the write request is to placed.
data	pointer to the data buffer contains the data to be written. In case of PIC32MZ device, this buffer should be located in coherent memory and should be aligned a 16 byte boundary.
size	Size of the data buffer.

Function

```
USB_DEVICE_AUDIO_RESULT USB_DEVICE_AUDIO_Write
(
    USB_DEVICE_AUDIO_INDEX instance ,
    USB_DEVICE_AUDIO_TRANSFER_HANDLE* transferHandle,
uint8_t interfaceNum ,
```

```
void * data ,
size_t size
);
```

b) Data Types and Constants

USB_DEVICE_AUDIO_EVENT_RESPONSE_NONE Macro

USB Device Audio Function Driver event handler response type none.

File

usb_device_audio_v1_0.h

C

```
#define USB_DEVICE_AUDIO_EVENT_RESPONSE_NONE
```

Description

USB Device Audio Function Driver Event Handler Response None

This is the definition of the Audio Function Driver event handler response type none.

Remarks

Intentionally defined to be empty.

USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID Macro

USB Device Audio Function Driver invalid transfer handle definition.

File

usb_device_audio_v1_0.h

C

```
#define USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID
```

Description

USB Device Audio Function Driver Invalid Transfer Handle Definition

This definition defines a Invalid USB Device Audio Function Driver Transfer Handle. A Invalid Transfer Handle is returned by the USB_DEVICE_Audio_Write , USB_DEVICE_Audio_Read, functions when the request was not successful.

Remarks

None.

USB_DEVICE_AUDIO_EVENT Enumeration

USB Device Audio Function Driver events.

File

usb_device_audio_v1_0.h

C

```
typedef enum {
  USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE,
  USB_DEVICE_AUDIO_EVENT_READ_COMPLETE,
  USB_DEVICE_AUDIO_EVENT_INTERFACE_SETTING_CHANGED,
  USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_DATA_RECEIVED,
  USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_DATA_SENT,
  USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_UNKNOWN,
  USB_DEVICE_AUDIO_EVENT_CONTROL_SET_CUR,
  USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MIN,
  USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MAX,
  USB_DEVICE_AUDIO_EVENT_CONTROL_SET_RES,
  USB_DEVICE_AUDIO_EVENT_ENTITY_SET_MEM,
```

```
    USB_DEVICE_AUDIO_EVENT_CONTROL_GET_CUR,
    USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MIN,
    USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MAX,
    USB_DEVICE_AUDIO_EVENT_CONTROL_GET_RES,
    USB_DEVICE_AUDIO_EVENT_ENTITY_GET_MEM,
    USB_DEVICE_AUDIO_EVENT_ENTITY_GET_STAT
} USB_DEVICE_AUDIO_EVENT;
```

Members

Members	Description
USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE	This event occurs when a write operation scheduled by calling the USB_DEVICE_AUDIO_Write() function has completed. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE_DATA type.
USB_DEVICE_AUDIO_EVENT_READ_COMPLETE	This event occurs when a read operation scheduled by calling the USB_DEVICE_AUDIO_Read() function has completed. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_READ_COMPLETE_DATA type.
USB_DEVICE_AUDIO_EVENT_INTERFACE_SETTING_CHANGED	This event occurs when the Host requests the Audio USB device to set an alternate setting on an interface present in this audio function. An Audio USB Device will typically feature a default interface setting and one or more alternate interface settings. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_DATA_INTERFACE_SETTING_CHANGED type. This contains the index of the interface whose setting must be changed and the index of the alternate setting. The application may enable or disable audio functions based on the interface setting.
USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	This event occurs when the data stage of a control write transfer has completed. This would occur after the application would respond with a USB_DEVICE_ControlReceive function, which may possibly have been called in response to a USB_DEVICE_AUDIO_EVENT_ENTITY_SETTINGS_RECEIVED event This event notifies the application that the data is received from Host and is available at the location passed by the USB_DEVICE_ControlReceive function. If the received data is acceptable to the application, it should acknowledge the data by calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_OK flag.The application can reject the received data by calling the USB_DEVICE_ControlStatus function with the USB_DEVICE_CONTROL_STATUS_ERROR flag. The pData parameter will be NULL.
USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_DATA_SENT	This event occurs when the data stage of a control read transfer has completed. This would occur when the application has called the USB_DEVICE_ControlSend function to complete the data stage of a control transfer. The event indicates that the data has been transmitted to the host. The pData parameter will be NULL.
USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_UNKNOWN	This event occurs when the Audio function driver receives a control transfer request that could not be decoded by Audio Function driver.The pData parameter will point to a USB_SETUP_PACKET type containing the SETUP packet. The application must analyze this SETUP packet and use the USB_DEVICE_ControlSend or USB_DEVICE_ControlReceive or the USB_DEVICE_ControlStatus functions to advance the control transfer or complete it.
USB_DEVICE_AUDIO_EVENT_CONTROL_SET_CUR	This event occurs when the Host sends an Audio Control specific Set Current Setting Attribute Control Transfer request to an Audio Device Control. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_CUR type. The application must use the entityID, interface, endpoint and the wValue field in the event data to determine the entity and control type and then respond to the control transfer with a USB_DEVICE_ControlStatus and USB_DEVICE_ControlReceive functions.

USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MIN	This event occurs when the Host sends an Audio Control specific Set Minimum Setting Attribute Control Transfer request to an Audio Device Control. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MIN type. The application must use the entityID, interface, endpoint and the wValue field in the event data to determine the entity and control type and then respond to the control transfer with a USB_DEVICE_ControlStatus and USB_DEVICE_ControlReceive functions.
USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MAX	This event occurs when the Host sends an Audio Control specific Set Maximum Setting Attribute Control Transfer request to an Audio Device Control. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MAX type. The application must use the entityID, interface, endpoint and the wValue field in the event data to determine the entity and control type and then respond to the control transfer with a USB_DEVICE_ControlStatus and USB_DEVICE_ControlReceive functions.
USB_DEVICE_AUDIO_EVENT_CONTROL_SET_RES	This event occurs when the Host sends an Audio Control specific Set Resolution Attribute Control Transfer request to an Audio Device Control. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_RES type. The application must use the entityID, interface, endpoint and the wValue field in the event data to determine the entity and control type and then respond to the control transfer with a USB_DEVICE_ControlStatus USB_DEVICE_ControlSend and/or USB_DEVICE_ControlReceive functions.
USB_DEVICE_AUDIO_EVENT_ENTITY_SET_MEM	This event occurs when the Host sends an Audio Entity specific Set Memory Space Attribute Control Transfer request to an Audio Device Entity. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MEM type. The application must use the entityID, interface, endpoint and the wValue field in the event data to determine the entity and control type and then respond to the control transfer with a USB_DEVICE_ControlStatus USB_DEVICE_ControlSend and/or USB_DEVICE_ControlReceive functions.
USB_DEVICE_AUDIO_EVENT_CONTROL_GET_CUR	This event occurs when the Host sends an Audio Control specific Get Current Setting Attribute Control Transfer request to an Audio Device Control. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_CUR type. The application must use the entityID, interface, endpoint and the wValue field in the event data to determine the entity and control type and then respond to the control transfer with a USB_DEVICE_ControlStatus and USB_DEVICE_ControlSend functions.
USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MIN	This event occurs when the Host sends an Audio Control specific Get Minimum Setting Attribute Control Transfer request to an Audio Device Control. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MIN type. The application must use the entityID, interface, endpoint and the wValue field in the event data to determine the entity and control type and then respond to the control transfer with a USB_DEVICE_ControlStatus and USB_DEVICE_ControlSend functions.
USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MAX	This event occurs when the Host sends an Audio Control specific Get Maximum Setting Attribute Control Transfer request to an Audio Device Control. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MAX type. The application must use the entityID, interface, endpoint and the wValue field in the event data to determine the entity and control type and then respond to the control transfer with a USB_DEVICE_ControlStatus and USB_DEVICE_ControlSend functions.

USB_DEVICE_AUDIO_EVENT_CONTROL_GET_RES	This event occurs when the Host sends an Audio Control specific Get Resolution Setting Attribute Control Transfer request to an Audio Device Control. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_RES type. The application must use the entityID, interface, endpoint and the wValue field in the event data to determine the entity and control type and then respond to the control transfer with a USB_DEVICE_ControlStatus and USB_DEVICE_ControlSend functions.
USB_DEVICE_AUDIO_EVENT_ENTITY_GET_MEM	This event occurs when the Host sends an Audio Entity specific Get Memory Space Attribute Control Transfer request to an Audio Device Entity. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MEM type. The application must use the entityID, interface, endpoint and the wValue field in the event data to determine the entity and control type and then respond to the control transfer with a USB_DEVICE_ControlStatus or USB_DEVICE_ControlSend functions.
USB_DEVICE_AUDIO_EVENT_ENTITY_GET_STAT	This event occurs when the Host sends a Audio Entity specific Get Status Control Transfer request to an Audio Device Entity. The pData member in the event handler will point to USB_DEVICE_AUDIO_EVENT_DATA_ENTITY_GET_STAT type. The application mus use the entityID, interface, endpoint and the wValue field in the event data to determine the entity and control type and then respond to the control transfer with a USB_DEVICE_ControlSend and or USB_DEVICE_ControlStatus functions.

Description

USB Device Audio Function Driver Events

These events are specific to a USB Device Audio Function Driver instance. An event may have some data associated with it. This is provided to the event handling function. Each event description contains details about this event data (pData) and other parameters passed along with the event, to the event handler.

Events associated with the Audio Function Driver Specific Control Transfers require application response. The application should respond to these events by using the USB_DEVICE_ControlReceive, USB_DEVICE_ControlSend and USB_DEVICE_ControlStatus functions.

Calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_ERROR will stall the control transfer request. The application would do this if the control transfer request is not supported. Calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_OK will complete the status stage of the control transfer request. The application would do this if the control transfer request is supported.

The following code shows an example of a possible event handling scheme.

```
// This code example shows all USB Audio Function Driver possible events and
// a possible scheme for handling these events. In this case event responses
// are not deferred.

void APP_USBDeviceAudioEventHandler
(
    USB_DEVICE_AUDIO_INDEX instanceIndex ,
    USB_DEVICE_AUDIO_EVENT event ,
    void * pData,
    uintptr_t context
)
{
    switch (event)
    {
        case USB_DEVICE_AUDIO_EVENT_READ_COMPLETE:

            // This event indicates that a Audio Read Transfer request
            // has completed. pData should be interpreted as a
            // USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE pointer type.
            // This contains the transfer handle of the read transfer
            // that completed and amount of data that was read.

            break;

        case USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE:

            // This event indicates that a Audio Write Transfer request
            // has completed. pData should be interpreted as a
```

```
            // USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE pointer type.
            // This contains the transfer handle of the write transfer
            // that completed and amount of data that was written.

            break;

        case USB_DEVICE_AUDIO_EVENT_INTERFACE_SETTING_CHANGED:

            // This event occurs when the host sends Set Interface request
            // to the Audio USB Device. pData will be a pointer to a
            // USB_DEVICE_AUDIO_EVENT_DATA_INTERFACE_SETTING_CHANGED. This
            // contains the interface number whose setting was
            // changed and the index of the alternate setting.
            // The application should typically enable the audio function
            // if the interfaceAlternateSettting member of pData is greater
            // than 0.

            break;

        case USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_UNKNOWN:

            // This event indicates that the Audio function driver has
            // received a control transfer which it cannot decode. pData
            // will be a pointer to USB_SETUP_PACKET type pointer. The
            // application should decode the packet and take the required
            // action using the USB_DEVICE_ControlStatus(),
            // USB_DEVICE_ControlSend() and USB_DEVICE_ControlReceive()
            // functions.

            break;

        case USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_DATA_SENT:

            // This event indicates the data send request associated with
            // the latest USB_DEVICE_ControlSend() function was
            // completed. pData will be NULL.

        case USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:

            // This event indicates the data receive request associated with
            // the latest USB_DEVICE_ControlReceive() function was
            // completed. pData will be NULL. The application can either
            // acknowledge the received data or reject it by calling the
            // USB_DEVICE_ControlStatus() function.

            break;

        case USB_DEVICE_AUDIO_EVENT_CONTROL_SET_CUR:

            // This event indicates that the host is trying to set the
            // current setting attribute of a control. The data type will be
            // USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_CUR type.  The
            // application should identify the entity type based on the
            // entity ID. This mapping is application specific. The
            // following example assumes entity type to be a Feature Unit.

            if(APP_EntityIdentify(((USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_CUR *)pData)->entityID)
                    == APP_AUDIO_ENTITY_FEATURE_UNIT)
            {
                // The entity type is a feature unit. Type cast pData as
                // a USB_AUDIO_FEATURE_UNIT_CONTROL_REQUEST type and find
                // identify the control selector. This example shows the
                // handling for VOLUME control

                switch(((USB_AUDIO_FEATURE_UNIT_CONTROL_REQUEST *)pData)->controlSelector)
                {
                    case USB_AUDIO_VOLUME_CONTROL:
                        // This means the host is trying to set the volume.
                        // Use the USB_DEVICE_ControlReceive() function to
```

```
                        // receive the volume settings for each channel.

                        USB_DEVICE_ControlReceive(usbDeviceHandle, volumeSetting,
                                ((USB_AUDIO_FEATURE_UNIT_CONTROL_REQUEST *)pData)->wLength);
                    default:
                        // Only volume control is supported in this example.
                        // So everything else is stalled.
                        USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_ERROR);
                }
            }
            break;

        case USB_DEVICE_AUDIO_EVENT_CONTROL_GET_CUR:

            // This event indicates that the host is trying to get the
            // current setting attribute of a control. The data type will be
            // USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_CUR type.  The
            // application should identify the entity type based on the
            // entity ID. This mapping is application specific. The
            // following example assumes entity type to be a Feature Unit.

            if(APP_EntityIdentify(((USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_CUR *)pData)->entityID)
                    == APP_AUDIO_ENTITY_FEATURE_UNIT)
            {
                // The entity type is a feature unit. Type cast pData as
                // a USB_AUDIO_FEATURE_UNIT_CONTROL_REQUEST type and find
                // identify the control selector. This example shows the
                // handling for VOLUME control

                switch(((USB_AUDIO_FEATURE_UNIT_CONTROL_REQUEST *)pData)->controlSelector)
                {
                    case USB_AUDIO_VOLUME_CONTROL:
                        // This means the host is trying to get the volume.
                        // Use the USB_DEVICE_ControlReceive() function to
                        // receive the volume settings for each channel.

                        USB_DEVICE_ControlSend(usbDeviceHandle, volumeSetting,
                                ((USB_AUDIO_FEATURE_UNIT_CONTROL_REQUEST *)pData)->wLength);
                    default:
                        // Only volume control is supported in this example.
                        // So everything else is stalled.
                        USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_ERROR);
                }
            }
            break;

        case USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MAX:
        case USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MIN:
        case USB_DEVICE_AUDIO_EVENT_CONTROL_SET_RES:
        case USB_DEVICE_AUDIO_EVENT_CONTROL_SET_MEM:
        case USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MAX:
        case USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MIN:
        case USB_DEVICE_AUDIO_EVENT_CONTROL_GET_RES:
        case USB_DEVICE_AUDIO_EVENT_CONTROL_GET_MEM:
            // In this example these request are not supported and so are
            // stalled.
            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_ERROR);
            break;

        default:
            break;
    }

    return(USB_DEVICE_AUDIO_EVENT_RESPONSE_NONE);
}
```

Remarks

The application can defer responses to events triggered by control transfers. In that, the application can respond to the control transfer event after exiting the event handler. This allows the application some time to obtain the response data rather than having to respond to the event

immediately. Note that a USB host will typically wait for an event response for a finite time duration before timing out and canceling the event and associated transactions. Even when deferring response, the application must respond promptly if such time-out have to be avoided.

USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE Structure

USB Device Audio Function Driver audio read and write complete event data.

File

usb_device_audio_v1_0.h

C

```
typedef struct {
  USB_DEVICE_AUDIO_TRANSFER_HANDLE handle;
  uint16_t length;
  uint8_t interfaceNum;
  USB_DEVICE_AUDIO_RESULT status;
} USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE, USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE;
```

Members

Members	Description
USB_DEVICE_AUDIO_TRANSFER_HANDLE handle;	Transfer handle associated with this • read or write request
uint16_t length;	Indicates the amount of data (in bytes) that was • read or written
uint8_t interfaceNum;	Interface Number
USB_DEVICE_AUDIO_RESULT status;	Completion status of the transfer

Description

USB Device Audio Function Driver Read and Write Complete Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_AUDIO_EVENT_READ_COMPLETE, USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE, events.

Remarks

None.

USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE Structure

USB Device Audio Function Driver audio read and write complete event data.

File

usb_device_audio_v1_0.h

C

```
typedef struct {
  USB_DEVICE_AUDIO_TRANSFER_HANDLE handle;
  uint16_t length;
  uint8_t interfaceNum;
  USB_DEVICE_AUDIO_RESULT status;
} USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE, USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE;
```

Members

Members	Description
USB_DEVICE_AUDIO_TRANSFER_HANDLE handle;	Transfer handle associated with this • read or write request
uint16_t length;	Indicates the amount of data (in bytes) that was • read or written
uint8_t interfaceNum;	Interface Number
USB_DEVICE_AUDIO_RESULT status;	Completion status of the transfer

Description

USB Device Audio Function Driver Read and Write Complete Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_AUDIO_EVENT_READ_COMPLETE,

USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE, events.

Remarks

None.

USB_DEVICE_AUDIO_EVENT_HANDLER Type

USB Device Audio event handler function pointer type.

File

usb_device_audio_v1_0.h

C

```
typedef USB_DEVICE_AUDIO_EVENT_RESPONSE (* USB_DEVICE_AUDIO_EVENT_HANDLER)(USB_DEVICE_AUDIO_INDEX
instanceIndex , USB_DEVICE_AUDIO_EVENT event , void * pData, uintptr_t context);
```

Description

USB Device Audio Event Handler Function Pointer Type.

This data type defines the required function signature USB Device Audio Function Driver event handling callback function. The application must register a pointer to an Audio Function Driver events handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event call backs from the Audio Function Driver. The function driver will invoke this function with event relevant parameters. The description of the event handler function parameters is given here.

instanceIndex - Instance index of the Audio Function Driver that generated the event.

event - Type of event generated.

pData - This parameter should be typecast to an event specific pointer type based on the event that has occurred. Refer to the USB_DEVICE_AUDIO_EVENT enumeration description for more details.

context - Value identifying the context of the application that registered the event handling function.

Remarks

The event handler function executes in the USB interrupt context when the USB Device Stack is configured for interrupt based operation. It is not advisable to call blocking functions or computationally intensive functions in the event handler. Where the response to a control transfer related event requires extended processing, the response to the control transfer should be deferred and the event handler should be allowed to complete execution.

USB_DEVICE_AUDIO_EVENT_RESPONSE Type

USB Device Audio Function Driver event callback response type.

File

usb_device_audio_v1_0.h

C

```
typedef void USB_DEVICE_AUDIO_EVENT_RESPONSE;
```

Description

USB Device Audio Function Driver Event Handler Response Type

This is the return type of the Audio Function Driver event handler.

Remarks

None.

USB_DEVICE_AUDIO_INDEX Type

USB Device Audio function driver index.

File

usb_device_audio_v1_0.h

C

```
typedef uintptr_t USB_DEVICE_AUDIO_INDEX;
```

Description

USB Device Audio Function Driver Index

This definition uniquely identifies a Audio Function Driver instance.

Remarks

None.

USB_DEVICE_AUDIO_RESULT Enumeration

USB Device Audio Function Driver USB Device Audio result enumeration.

File

usb_device_audio_v1_0.h

C

```
typedef enum {
  USB_DEVICE_AUDIO_RESULT_OK,
  USB_DEVICE_AUDIO_RESULT_ERROR_TRANSFER_QUEUE_FULL,
  USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_INVALID,
  USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_NOT_CONFIGURED,
  USB_DEVICE_AUDIO_RESULT_ERROR_PARAMETER_INVALID,
  USB_DEVICE_AUDIO_RESULT_ERROR_INVALID_INTERFACE_ID,
  USB_DEVICE_AUDIO_RESULT_ERROR_INVALID_BUFFER,
  USB_DEVICE_AUDIO_RESULT_ERROR_ENDPOINT_HALTED,
  USB_DEVICE_AUDIO_RESULT_ERROR_TERMINATED_BY_HOST,
  USB_DEVICE_AUDIO_RESULT_ERROR
} USB_DEVICE_AUDIO_RESULT;
```

Members

Members	Description
USB_DEVICE_AUDIO_RESULT_OK	The operation was successful
USB_DEVICE_AUDIO_RESULT_ERROR_TRANSFER_QUEUE_FULL	The transfer queue is full and no new transfers can be scheduled
USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_INVALID	The specified instance is not provisioned in the system
USB_DEVICE_AUDIO_RESULT_ERROR_INSTANCE_NOT_CONFIGURED	The specified instance is not configured yet
USB_DEVICE_AUDIO_RESULT_ERROR_PARAMETER_INVALID	The event handler provided is NULL
USB_DEVICE_AUDIO_RESULT_ERROR_INVALID_INTERFACE_ID	Interface number passed to the read or write function is invalid.
USB_DEVICE_AUDIO_RESULT_ERROR_INVALID_BUFFER	A NULL buffer was specified in the read or write function
USB_DEVICE_AUDIO_RESULT_ERROR_ENDPOINT_HALTED	Transfer terminated because host halted the endpoint
USB_DEVICE_AUDIO_RESULT_ERROR_TERMINATED_BY_HOST	Transfer terminated by host because of a stall clear
USB_DEVICE_AUDIO_RESULT_ERROR	General Error

Description

USB Device Audio Function Driver USB Device Audio Result enumeration.

This enumeration lists the possible USB Device Audio Function Driver operation results.

Remarks

None.

USB_DEVICE_AUDIO_TRANSFER_HANDLE Type

USB Device Audio Function Driver transfer handle definition.

File

usb_device_audio_v1_0.h

C

```
typedef uintptr_t USB_DEVICE_AUDIO_TRANSFER_HANDLE;
```

Description

USB Device Audio Function Driver Transfer Handle Definition

This definition defines a USB Device Audio Function Driver Transfer Handle. A Transfer Handle is owned by the application but its value is

modified by the USB_DEVICE_AUDIO_Write, USB_DEVICE_AUDIO_Read functions. The transfer handle is valid for the life time of the transfer and expires when the transfer related event had occurred.

Remarks

None.

USB_DEVICE_AUDIO_FUNCTION_DRIVER Macro

USB Device Audio Function Driver function pointer.

File

usb_device_audio_v1_0.h

C

```
#define USB_DEVICE_AUDIO_FUNCTION_DRIVER
```

Description

USB Device Audio Function Driver Function Pointer

This is the USB Device Audio Function Driver function pointer. This should registered with the device layer in the function driver registration table.

Remarks

None.

USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_CUR Type

File

usb_device_audio_v1_0.h

C

```
typedef USB_AUDIO_CONTROL_INTERFACE_REQUEST USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_CUR;
```

Description

This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_CUR.

USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MAX Type

File

usb_device_audio_v1_0.h

C

```
typedef USB_AUDIO_CONTROL_INTERFACE_REQUEST USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MAX;
```

Description

This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MAX.

USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MEM Type

File

usb_device_audio_v1_0.h

C

```
typedef USB_AUDIO_CONTROL_INTERFACE_REQUEST USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MEM;
```

Description

This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MEM.

USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MIN Type

File

usb_device_audio_v1_0.h

C

```
typedef USB_AUDIO_CONTROL_INTERFACE_REQUEST USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MIN;
```

Description

This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MIN.

USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_RES Type

File

usb_device_audio_v1_0.h

C

```
typedef USB_AUDIO_CONTROL_INTERFACE_REQUEST USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_RES;
```

Description

This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_RES.

USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_CUR Type

USB Device Audio Function Driver set and get request data.

File

usb_device_audio_v1_0.h

C

```
typedef USB_AUDIO_CONTROL_INTERFACE_REQUEST USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_CUR;
```

Description

USB Device Audio Function Driver Set and Get request data.

This data type defines the data structure returned by the driver along with the USB_DEVICE_AUDIO_EVENT_CONTROL_SET_XXX, USB_DEVICE_AUDIO_EVENT_ENTITY_SET_MEM, USB_DEVICE_AUDIO_EVENT_CONTROL_GET_XXX and USB_DEVICE_AUDIO_EVENT_ENTITY_GET_MEM events.

Remarks

None.

USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MAX Type

File

usb_device_audio_v1_0.h

C

```
typedef USB_AUDIO_CONTROL_INTERFACE_REQUEST USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MAX;
```

Description

This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MAX.

USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MEM Type

File

usb_device_audio_v1_0.h

C

```
typedef USB_AUDIO_CONTROL_INTERFACE_REQUEST USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MEM;
```

Description

This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MEM.

USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MIN Type

File

usb_device_audio_v1_0.h

C

```
typedef USB_AUDIO_CONTROL_INTERFACE_REQUEST USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MIN;
```

Description

This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MIN.

USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_RES Type

File

usb_device_audio_v1_0.h

C

```
typedef USB_AUDIO_CONTROL_INTERFACE_REQUEST USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_RES;
```

Description

This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_RES.

USB_DEVICE_AUDIO_EVENT_DATA_INTERFACE_SETTING_CHANGED Structure

USB Device Audio Function Driver alternate interface setting event data.

File

usb_device_audio_v1_0.h

C

```
typedef struct {
  uint8_t interfaceNumber;
  uint8_t interfaceAlternateSetting;
} USB_DEVICE_AUDIO_EVENT_DATA_INTERFACE_SETTING_CHANGED;
```

Members

Members	Description
uint8_t interfaceNumber;	Interface number of the interface who setting is to be changed
uint8_t interfaceAlternateSetting;	Alternate setting number

Description

USB Device Audio Function Driver Alternate Interface Setting Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_AUDIO_EVENT_DATA_INTERFACE_SETTING_CHANGED.

Remarks

None.

USB_DEVICE_AUDIO_INIT Structure

USB Device Audio Function Driver initialization data structure.

File

usb_device_audio_v1_0.h

C

```
typedef struct {
  size_t queueSizeRead;
  size_t queueSizeWrite;
} USB_DEVICE_AUDIO_INIT;
```

Members

Members	Description
size_t queueSizeRead;	Size of the read queue for this instance • of the Audio function driver
size_t queueSizeWrite;	Size of the write queue for this instance • of the Audio function driver

Description

USB Device Audio Function Driver Initialization Data Structure

This data structure must be defined for every instance of the Audio Function Driver. It is passed to the Audio function driver, by the Device Layer, at the time of initialization. The funcDriverInit member of the Device Layer Function Driver registration table entry must point to this data structure for an instance of the Audio function driver.

Remarks

The queue sizes that are specified in this data structure are also affected by the USB_DEVICE_AUDIO_QUEUE_DEPTH_COMBINED configuration macro.

USB_DEVICE_AUDIO_EVENT_DATA_ENTITY_GET_STAT Type

File

usb_device_audio_v1_0.h

C

```
typedef USB_AUDIO_CONTROL_INTERFACE_REQUEST USB_DEVICE_AUDIO_EVENT_DATA_ENTITY_GET_STAT;
```

Description

This is type USB_DEVICE_AUDIO_EVENT_DATA_ENTITY_GET_STAT.

USB_DEVICE_AUDIO_TRANSFER_ABORT_NOTIFY Macro

USB Audio Transfer abort notification enable

File

usb_device_audio_v1_0_config_template.h

C

```
#define USB_DEVICE_AUDIO_TRANSFER_ABORT_NOTIFY
```

Description

USB Audio Transfer abort notification

This macro enabled USB Audio Transfer abort notifications. Whenever a scheduled transfer request is aborted due to Device Unplug or Host resets the device, the transfer complete event with status as aborted would be send to application's event handler.

Remarks

None.

Files

Files

Name	Description
usb_device_audio_v1_0.h	USB Device Audio function Driver Interface
usb_device_audio_v1_0_config_template.h	USB device Audio Class configuration definitions template

Description

This section lists the source and header files used by the library.

usb_device_audio_v1_0.h

USB Device Audio function Driver Interface

Enumerations

	Name	Description
	USB_DEVICE_AUDIO_EVENT	USB Device Audio Function Driver events.
	USB_DEVICE_AUDIO_RESULT	USB Device Audio Function Driver USB Device Audio result enumeration.

Functions

	Name	Description
	USB_DEVICE_AUDIO_EventHandlerSet	This function registers an event handler for the specified Audio function driver instance.
	USB_DEVICE_AUDIO_Read	This function requests a data read from the USB Device Audio Function Driver Layer.
	USB_DEVICE_AUDIO_TransferCancel	This function cancels a scheduled Audio Device data transfer.
	USB_DEVICE_AUDIO_Write	This function requests a data write to the USB Device Audio Function Driver Layer.

Macros

	Name	Description
	USB_DEVICE_AUDIO_EVENT_RESPONSE_NONE	USB Device Audio Function Driver event handler response type none.
	USB_DEVICE_AUDIO_FUNCTION_DRIVER	USB Device Audio Function Driver function pointer.
	USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID	USB Device Audio Function Driver invalid transfer handle definition.

Structures

	Name	Description
	USB_DEVICE_AUDIO_EVENT_DATA_INTERFACE_SETTING_CHANGED	USB Device Audio Function Driver alternate interface setting event data.
	USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE	USB Device Audio Function Driver audio read and write complete event data.
	USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE	USB Device Audio Function Driver audio read and write complete event data.
	USB_DEVICE_AUDIO_INIT	USB Device Audio Function Driver initialization data structure.

Types

	Name	Description
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_CUR	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_CUR.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MAX	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MAX.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MEM	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MEM.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MIN	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MIN.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_RES	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_RES.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_CUR	USB Device Audio Function Driver set and get request data.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MAX	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MAX.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MEM	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MEM.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MIN	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MIN.
	USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_RES	This is type USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_RES.
	USB_DEVICE_AUDIO_EVENT_DATA_ENTITY_GET_STAT	This is type USB_DEVICE_AUDIO_EVENT_DATA_ENTITY_GET_STAT.
	USB_DEVICE_AUDIO_EVENT_HANDLER	USB Device Audio event handler function pointer type.
	USB_DEVICE_AUDIO_EVENT_RESPONSE	USB Device Audio Function Driver event callback response type.
	USB_DEVICE_AUDIO_INDEX	USB Device Audio function driver index.

	USB_DEVICE_AUDIO_TRANSFER_HANDLE	USB Device Audio Function Driver transfer handle definition.

Description

USB Device Audio Function Driver Interface

This file describes the USB Device Audio Function Driver interface. This file should be included by the application if it needs to use the Audio Function Driver API.

File Name

usb_device_audio.h

Company

Microchip Technology Inc.

usb_device_audio_v1_0_config_template.h

USB device Audio Class configuration definitions template

Macros

	Name	Description
	USB_DEVICE_AUDIO_INSTANCES_NUMBER	Specifies the number of Audio Function Driver instances.
	USB_DEVICE_AUDIO_MAX_ALTERNATE_SETTING	Specifies the maximum number of Alternate Settings per streaming interface.
	USB_DEVICE_AUDIO_MAX_STREAMING_INTERFACES	Specifies the maximum number of Audio Streaming interfaces in an Audio Function Driver instance.
	USB_DEVICE_AUDIO_QUEUE_DEPTH_COMBINED	Specifies the combined queue size of all Audio function driver instances.
	USB_DEVICE_AUDIO_TRANSFER_ABORT_NOTIFY	USB Audio Transfer abort notification enable

Description

USB Device Audio Class Configuration Definitions

This file contains configurations macros needed to configure the Audio Function Driver. This file is a template file only. It should not be included by the application. The configuration macros defined in the file should be defined in the configuration specific system_config.h.

File Name

usb_device_audio_v1_0_config_template.h

Company

Microchip Technology Inc.

USB Audio 2.0 Device Library

This section describes the USB Audio 2.0 Device Library.

Introduction

This section provides information on library design, configuration, usage and the library interface for the USB Audio 2.0 Device Library.

Description

The MPLAB Harmony USB Audio 2.0 Device Library (also referred to as the Audio 2.0 Function Driver or library) features routines to implement a USB Audio 2.0 Device. Examples of Audio USB 2.0 Devices include USB Speakers, microphones, and voice telephony. The library provides a convenient abstraction of the USB Audio 2.0 Device specification and simplifies the implementation of USB Audio 2.0 Devices.

Using the Library

This topic describes the basic architecture of the Audio 2.0 Function Driver and provides information and examples on its use.

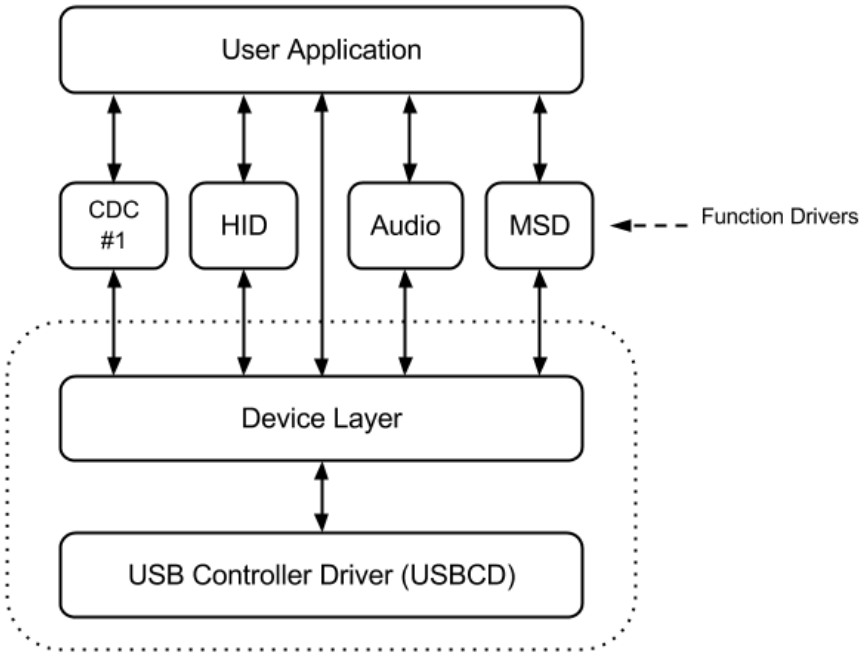
Abstraction Model

Describes the Abstraction Model of the USB Audio 2.0 Device Library.

Description

The Audio 2.0 Function Driver offers various services to the USB Audio 2.0 device to communicate with the host by abstracting USB specification details. It must be used along with the USB Device layer and USB controller to communicate with the USB host. Figure 1 shows a block diagram of the MPLAB Harmony USB Device Stack Architecture and where the Audio 2.0 Function Driver is placed.

Figure 1: USB Device Audio Device Driver



The USB controller driver takes the responsibility of managing the USB peripheral on the device. The USB Device Layer handles the device enumeration, etc. The USB Device Layer forwards all Audio-specific control transfers to the Audio 2.0 Function Driver. The Audio 2.0 Function Driver interprets the control transfers and requests application's intervention through event handlers and a well-defined set of API. The application must respond to the Audio events either in or out of the event handler. Some of these events are related to Audio 2.0 Device Class specific control transfers. The application must complete these control transfers within the timing constraints defined by USB.

Library Overview

The USB Audio 2.0 Device Library mainly interacts with the system, its clients and function drivers, as shown in the Abstraction Model.

The library interface routines are divided into sub-sections, which address one of the blocks or the overall operation of the USB Audio 2.0 Device Library.

Library Interface Section	Description
Functions	Provides event handler, read/write, and transfer cancellation functions.

How the Library Works

Initializing the Library

Describes how the USB Audio 2.0 Device driver is initialized.

Description

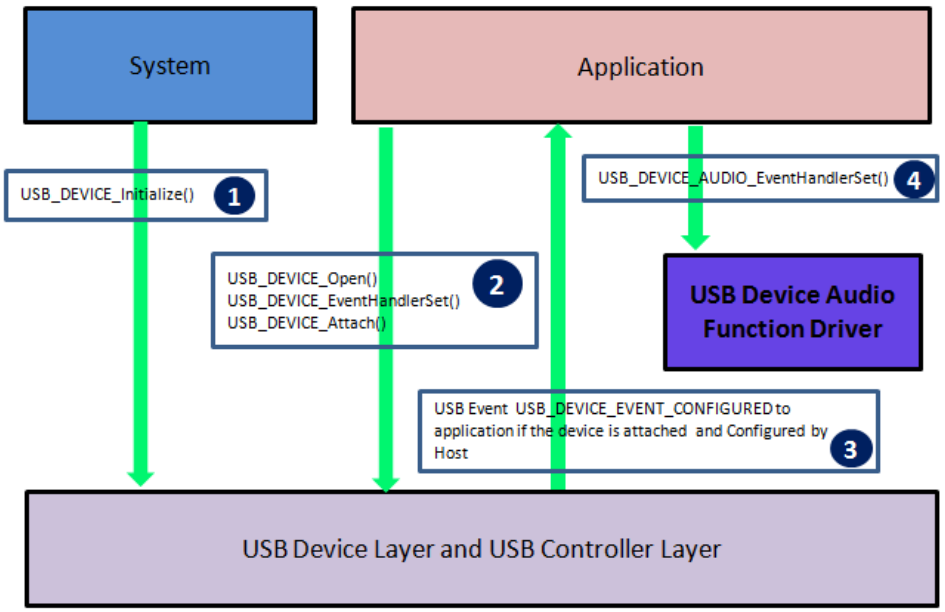
The Audio 2.0 Function Driver instance for a USB device configuration is initialized by the Device Layer when the configuration is set by the host. This process does not require application intervention. Each instance of the Audio 2.0 Function Driver should be registered with the Device layer through the Device Layer Function Driver Registration Table. The Audio 2.0 Function Driver requires a initialization data structure to be specified. This is a USB_DEVICE_AUDIO_2_0_INIT data type that specifies the size of the read and write queues. The funcDriverInit member of the function driver registration table entry of the Audio 2.0 Function Driver instance should point to this initialization data structure. The

USB_DEVICE_AUDIO_2_0_FUNCTION_DRIVER object is a global object provided by the Audio 2.0 Function Driver and provides the Device Layer with an entry point into the Audio 2.0 Function Driver. The following code shows an example of how the Audio 2.0 Function Driver can be registered with the Device Layer.

```
/* This code shows an example of how an Audio 2.0 function driver instances
 * can be registered with the Device Layer via the Device Layer Function Driver
 * Registration Table.  In this case Device Configuration 1 consists of one
 * Audio 2.0 function driver instance. */

/* The Audio 2.0 Function Driver requires an initialization data structure that
 * specifies the read and write buffer queue sizes. Note that these settings are
 * also affected by the USB_DEVICE_AUDIO_QUEUE_DEPTH_COMBINED configuration
 * macro. */

const USB_DEVICE_AUDIO_2_0_INIT audioDeviceInit =
{
    .queueSizeRead = 1,
    .queueSizeWrite = 1
};

const USB_DEVICE_FUNC_REGISTRATION_TABLE funcRegistrationTable[1] =
{
    {
        .speed = USB_SPEED_FULL,                        // Supported speed
        .configurationValue = 1,                        // To be initialized for Configuration 1
        .interfaceNumber = 0,                           // Starting interface number.
        .numberOfInterfaces = 2,                        // Number of interfaces in this instance
        .funcDriverIndex = 0,                           // Function Driver instance index is 0
        .funcDriverInit = &audioDeviceInit,             // Function Driver does not need initialization
data structure
        .driver = USB_DEVICE_AUDIO_2_0_FUNCTION_DRIVER  // Pointer to Function Driver - Device Layer
interface functions
    },
};
```

The following figure illustrates the typical sequence that is followed in the application when using the Audio 2.0 Function Driver.

Typical USB Audio 2.0 Device Sequence



1. Call set of APIs to initialize USB Device Layer (refer to the USB Device Layer Library section for details about these APIs).
2. The Device Layer provides a callback to the application for any USB Device events like attached, powered, configured, etc. The application should receive a callback with an event USB_DEVICE_EVENT_CONFIGURED to proceed.
3. Once the Device Layer is configured, the application needs to register a callback function with the Audio 2.0 Function Driver to receive Audio 2.0 Control transfers, and also other Audio 2.0 Function Driver events. Now the application can use Audio 2.0 Function Driver APIs to communicate with the USB Host.

Event Handling

Describes Audio 2.0 Function Driver event handler registration and event handling.

Description

Registering a Audio 2.0 Function Driver Callback Function

While creating a USB Audio 2.0 Device application, an event handler must be registered with the Device Layer (the Device Layer Event Handler) and every Audio 2.0 Function Driver instance (Audio 2.0 Function Driver Event Handler). The application needs to register the event handler with the Audio 2.0 Function Driver:

- For receiving Audio 2.0 Control Requests from Host like Volume Control, Mute Control, etc.
- For handling other events from USB Audio 2.0 Device Driver (e.g., Data Write Complete or Data Read Complete)

The event handler should be registered before the USB device layer acknowledges the SET CONFIGURATION request from the USB Host. To ensure this, the callback function should be set in the USB_DEVICE_EVENT_CONFIGURED event that is generated by the device layer. The following code example shows how this can be done.

```
/* This a sample Application Device Layer Event Handler
 * Note how the USB Audio 2.0 Device Driver callback function
 * USB_DEVICE_AUDIO_2_0_EventHandlerSet()
 * is registered in the USB_DEVICE_EVENT_CONFIGURED event.  */

void APP_UsbDeviceEventCallBack( USB_DEVICE_EVENT event, void * pEventData, uintptr_t context )
{
    uint8_t * configuredEventData;
    switch( event )
    {
        case USB_DEVICE_EVENT_RESET:
            break;
        case USB_DEVICE_EVENT_DECONFIGURED:
            // USB device is reset or device is de-configured.
            // This means that USB device layer is about to de-initialize
            // all function drivers. So close handles to previously opened
            // function drivers.
            break;

        case USB_DEVICE_EVENT_CONFIGURED:
            /* check the configuration */
             /* Initialize the Application */
            configuredEventData = pEventData;
            if(*configuredEventData == 1)
            {
                USB_DEVICE_AUDIO_V2_EventHandlerSet
                (
                    0,
                    APP_USBDeviceAudioEventHandler ,
                    (uintptr_t)NULL
                );
                /* mark that set configuration is complete */
                appData.isConfigured = true;
            }
            break;

        case USB_DEVICE_EVENT_SUSPENDED:
            break;

        case USB_DEVICE_EVENT_POWER_DETECTED:
            /* Attach the device */
            USB_DEVICE_Attach (appData.usbDevHandle);
            break;

        case USB_DEVICE_EVENT_POWER_REMOVED:
                /* VBUS is not available. We can detach the device */
                USB_DEVICE_Detach(appData.usbDevHandle);
                break;

        case USB_DEVICE_EVENT_RESUMED:
        case USB_DEVICE_EVENT_ERROR:
```

```
            default:
                break;
        }
    }
```

Event Handling

The Audio 2.0 Function Driver provides events to the application through the event handler function registered by the application. These events indicate:

- Completion of a read or a write data transfer
- Audio 2.0 Control Interface requests
- Completion of data and the status stages of Audio 2.0 Control Interface related control transfer

The Audio 2.0 Control Interface Request events and the related control transfer events typically require the application to respond with the Device Layer Control Transfer routines to complete the control transfer. Based on the generated event, the application may be required to:

- Respond with a USB_DEVICE_ControlSend function, which is completes the data stage of a Control Read Transfer
- Respond with a USB_DEVICE_ControlReceive function, which provisions the data stage of a Control Write Transfer
- Respond with a USB_DEVICE_ControlStatus function, which completes the handshake stage of the Control Transfer. The application can either STALL or Acknowledge the handshake stage through the USB_DEVICE_ControlStatus function.

The following table shows the Audio 2.0 Function Driver Control Transfer related events and the required application control transfer actions.

Audio 2.0 Function Driver Control Transfer Event	Required Application Action
USB_DEVICE_AUDIO_V2_CUR_ENTITY_SETTINGS_RECEIVED	Identify the control type using the associated event data. If a data stage is expected, use the USB_DEVICE_ControlReceive function to receive expected data. If a data stage is not required or if the request is not supported, use the USB_DEVICE_ControlStatus function to Acknowledge or Stall the request.
USB_DEVICE_AUDIO_V2_RANGE_ENTITY_SETTINGS _RECEIVED	Identify the control type using the associated event data. If a data stage is expected, use the USB_DEVICE_ControlReceive function to receive expected data. If a data stage is not required or if the request is not supported, use the USB_DEVICE_ControlStatus function to Acknowledge or Stall the request.
USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER _DATA_RECEIVED	Acknowledge or stall using the USB_DEVICE_ControlStatus function.
USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER _DATA_SENT	Action not required.

The application must analyze the wIndex field of the event data (received with the control transfer event) to identify the entity that is being addressed. The application must be aware of all entities included in the application and their IDs. Once identified, the application can then type cast the event data to entity type the specific control request type. For example, if the Host sends a control request to set the clock source for the Audio 2.0 device, the following occurs in this order:

1. The Audio 2.0 Function Driver will generate a USB_DEVICE_AUDIO_V2_CUR_ENTITY_SETTINGS_RECEIVED event.
2. The application must type cast the event data to a USB_AUDIO_V2_CONTROL_INTERFACE_REQUEST type and check the entityID field.
3. The entityID field will be identified by the application as a Clock Source.
4. The application must now type cast the event data type as a USB_AUDIO_V2_CLOCKSOURCE_CONTROL_REQUEST data type and check the controlSelector field.
5. If the controlSelector field is AUDIO_V2_CS_SAM_FREQ_CONTROL, the application can then call the USB_DEVICE_ControlReceive function to receive the clock source.

Based on the type of event, the application should analyze the event data parameter of the event handler. This data member should be type cast to an event specific data type. The following table shows the event and the data type to use while type casting. Note that the event data member is not required for all events.

Audio 2.0 Function Driver Event	Related Event Data Type
USB_DEVICE_AUDIO_V2_CUR_ENTITY_SETTINGS_RECEIVED	USB_AUDIO_V2_CONTROL_INTERFACE_REQUEST*
USB_DEVICE_AUDIO_V2_RANGE_ENTITY_SETTINGS _RECEIVED	USB_AUDIO_V2_CONTROL_INTERFACE_REQUEST*
USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER _DATA_RECEIVED	NULL
USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER _DATA_SENT	NULL
USB_DEVICE_AUDIO_V2_EVENT_INTERFACE_SETTING_CHANGED	USB_DEVICE_AUDIO_V2_EVENT_DATA_SET_ALTERNATE_INTERFACE *

USB_DEVICE_AUDIO_V2_EVENT_READ_COMPLETE	USB_DEVICE_AUDIO_V2_EVENT_DATA_READ_COMPLETE *
USB_DEVICE_AUDIO_V2_EVENT_WRITE_COMPLETE	USB_DEVICE_AUDIO_V2_EVENT_DATA_READ_COMPLETE *

Handling Audio Control Requests:

When the Audio 2.0 Function Driver receives an Audio 2.0 Class Specific Control Transfer Request, it passes this control transfer to the application as a Audio 2.0 Function Driverevent. The following code example shows how to handle an Audio 2.0 Control request.

```
void APP_USBDeviceAudioEventHandler
(
    USB_DEVICE_AUDIO_V2_INDEX iAudio ,
    USB_DEVICE_AUDIO_V2_EVENT event ,
    void * pData,
    uintptr_t context
)
{
    USB_DEVICE_AUDIO_V2_EVENT_DATA_SET_ALTERNATE_INTERFACE * interfaceInfo;
    USB_DEVICE_AUDIO_V2_EVENT_DATA_READ_COMPLETE * readEventData;
    USB_DEVICE_AUDIO_V2_EVENT_DATA_WRITE_COMPLETE * writeEventData;
   USB_AUDIO_V2_CONTROL_INTERFACE_REQUEST* controlRequest;
    if ( iAudio == 0 )
    {
        switch (event)
        {
            case USB_DEVICE_AUDIO_V2_EVENT_READ_COMPLETE:
                readEventData = (USB_DEVICE_AUDIO_V2_EVENT_DATA_READ_COMPLETE *)pData;
                //We have received an audio frame from the Host.
                //Now send this audio frame to Audio Codec for Playback.

            break;

            case USB_DEVICE_AUDIO_V2_EVENT_WRITE_COMPLETE:

            break;

            case USB_DEVICE_AUDIO_V2_EVENT_INTERFACE_SETTING_CHANGED:
                //We have received a request from USB host to change the Interface-
                //Alternate setting.
                interfaceInfo = (USB_DEVICE_AUDIO_V2_EVENT_DATA_SET_ALTERNATE_INTERFACE *)pData;
                appData.activeInterfaceAlternateSetting = interfaceInfo->interfaceAlternateSetting;
                appData.state = APP_USB_INTERFACE_ALTERNATE_SETTING_RCVD;

            break;
            case USB_DEVICE_AUDIO_V2_CUR_ENTITY_SETTINGS_RECEIVED:
                controlRequest = (USB_AUDIO_V2_CONTROL_INTERFACE_REQUEST*) setupPkt;
    USB_AUDIO_V2_CLOCKSOURCE_CONTROL_REQUEST* clockSourceRequest;

        switch(controlRequest->entityID)
        {
       case APP_ID_CLOCK_SOURCE:
          clockSourceRequest = (USB_AUDIO_V2_CLOCKSOURCE_CONTROL_REQUEST*) controlRequest;
                    switch(clockSourceRequest->controlSelector)
                   {
                         case AUDIO_V2_CS_SAM_FREQ_CONTROL:
                        {
                       if ((controlRequest->bmRequestType & 0x80) == 0)
                       {
                           //A control write transfer received from Host. Now receive data from Host.
                          USB_DEVICE_ControlReceive(appData.usbDevHandle, (void *) &(appData.clockSource),
4 );
                          appData.currentAudioControl = APP_USB_AUDIO_CLOCKSOURCE_CONTROL;
                                  }
                             else
                             {
                       /*Handle Get request*/
                       USB_DEVICE_ControlSend(appData.usbDevHandle, (void *)&(appData.clockSource), 4 );
                       appData.currentAudioControl = APP_USB_CONTROL_NONE;
                              }
                     }
```

```
}
                break;

            case USB_DEVICE_AUDIO_V2_RANGE_ENTITY_SETTINGS_RECEIVED:

                break;

            case USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:
                USB_DEVICE_ControlStatus(appData.usbDevHandle, USB_DEVICE_CONTROL_STATUS_OK );
                switch (appData.currentAudioControl)
                {
                    case APP_USB_AUDIO_MUTE_CONTROL:
                    {
                        appData.state = APP_MUTE_AUDIO_PLAYBACK;
                        appData.currentAudioControl = APP_USB_CONTROL_NONE;
                    }
                    break;
                    case APP_USB_AUDIO_CLOCKSOURCE_CONTROL:
                    {
                        // Handle Clock Source Control here.
                        appData.state = APP_CLOCKSOURCE_SET;
                        appData.currentAudioControl = APP_USB_CONTROL_NONE;
                    }
                    break;
                    case APP_USB_AUDIO_CLOCKSELECT_CONTROL:
                    {
                        // Handle Clock Source Control here.
                        appData.currentAudioControl = APP_USB_CONTROL_NONE;

                    }
                    break;
                }
            break;
            case  USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER_DATA_SENT:
            break;
            default:
                SYS_ASSERT ( false , "Invalid callback" );
            break;
        } //end of switch ( callback )
    }//end of if  if ( iAudio == 0 )
}//end of function APP_AudioEventCallback
```

Transferring Data

Describes how to send/receive data to/from USB Host using this USB Audio 2.0 Device Driver.

Description

The USB Audio 2.0 Device Driver provides functions to send and receive data.

Receiving Data

The USB_DEVICE_AUDIO_V2_Read function schedules a data read. When the host transfers data to the device, the Audio 2.0 Function Driver receives the data and invokes the USB_DEVICE_AUDIO_V2_EVENT_READ_COMPLETE event. This event indicates that audio data is now available in the application specified buffer.

The Audio 2.0 Function Driver supports buffer queuing. The application can schedule multiple read requests. Each request is assigned a unique buffer handle, which is returned with the USB_DEVICE_AUDIO_V2_EVENT_READ_COMPLETE event. The application can use the buffer handle to track completion to queued requests. Using this feature allows the application to implement audio buffering schemes such as ping-pong buffering.

Sending Data

The USB_DEVICE_AUDIO_V2_Write schedules a data write. When the host sends a request for the data, the Audio 2.0 Function Driver transfers the data and invokes the USB_DEVICE_AUDIO_V2_EVENT_WRITE_COMPLETE event.

The Audio 2.0 Function Driver supports buffer queuing. The application can schedule multiple write requests. Each request is assigned a unique buffer handle, which is returned with the USB_DEVICE_AUDIO_V2_EVENT_WRITE_COMPLETE event. The application can use the buffer handle to track completion to queued requests. Using this feature allows the application to implement audio buffering schemes such as ping-pong buffering.

Configuring the Library

Macros

	Name	Description
	USB_DEVICE_AUDIO_V2_INSTANCES_NUMBER	Specifies the number of Audio 2.0 Function Driver instances.
	USB_DEVICE_AUDIO_V2_MAX_ALTERNATE_SETTING	Specifies the maximum number of Alternate Settings per streaming interface.
	USB_DEVICE_AUDIO_V2_MAX_STREAMING_INTERFACES	Specifies the maximum number of Audio 2.0 Streaming interfaces in an Audio 2.0 Function Driver instance.
	USB_DEVICE_AUDIO_V2_QUEUE_DEPTH_COMBINED	Specifies the combined queue size of all Audio 2.0 function driver instances.

Description

The application designer must specify the following configuration parameters while using the USB Audio 2.0 Device Driver. The configuration macros that implement these parameters must be located in the `system_config.h` file in the application project and a compiler include path (to point to the folder that contains this file) should be specified.

USB_DEVICE_AUDIO_V2_INSTANCES_NUMBER Macro

Specifies the number of Audio 2.0 Function Driver instances.

File

usb_device_audio_v2_0_config_template.h

C

```
#define USB_DEVICE_AUDIO_V2_INSTANCES_NUMBER
```

Description

USB device Audio 2.0 Maximum Number of instances

This macro defines the number of instances of the Audio 2.0 Function Driver. For example, if the application needs to implement two instances of the Audio 2.0 Function Driver (to create two composite Audio Device) on one USB Device, the macro should be set to 2. Note that implementing a USB Device that features multiple Audio 2.0 interfaces requires appropriate USB configuration descriptors.

Remarks

None.

USB_DEVICE_AUDIO_V2_MAX_ALTERNATE_SETTING Macro

Specifies the maximum number of Alternate Settings per streaming interface.

File

usb_device_audio_v2_0_config_template.h

C

```
#define USB_DEVICE_AUDIO_V2_MAX_ALTERNATE_SETTING
```

Description

Maximum number of Alternate Settings

This macro defines the maximum number of Alternate Settings per streaming interface. If the Audio 2.0 Device features multiple streaming interfaces, this configuration constant should be equal to the the maximum number of alternate required amongst the streaming interfaces.

Remarks

None.

USB_DEVICE_AUDIO_V2_MAX_STREAMING_INTERFACES Macro

Specifies the maximum number of Audio 2.0 Streaming interfaces in an Audio 2.0 Function Driver instance.

File

usb_device_audio_v2_0_config_template.h

C

```
#define USB_DEVICE_AUDIO_V2_MAX_STREAMING_INTERFACES
```

Description

Maximum Audio 2.0 Streaming Interfaces

This macro defines the maximum number of streaming interfaces in an Audio 2.0 Function Driver instance. In case of multiple Audio 2.0 Function Driver instances, this constant should be equal to the maximum number of interfaces amongst the Audio 2.0 Function Driver instances.

Remarks

None.

USB_DEVICE_AUDIO_V2_QUEUE_DEPTH_COMBINED Macro

Specifies the combined queue size of all Audio 2.0 function driver instances.

File

usb_device_audio_v2_0_config_template.h

C

```
#define USB_DEVICE_AUDIO_V2_QUEUE_DEPTH_COMBINED
```

Description

USB device Audio 2.0 Combined Queue Size

This macro defines the number of entries in all queues in all instances of the Audio 2.0 function driver. This value can be obtained by adding up the read and write queue sizes of each Audio Function driver instance. In a simple single instance USB Audio 2.0 device application, that does not require buffer queuing, the USB_DEVICE_AUDIO_QUEUE_DEPTH_COMBINED macro can be set to 2. Consider a case of a Audio 2.0 function driver instances, with has a read queue size of 2 and write queue size of 3, this macro should be set to 5 (2 + 3).

Remarks

None.

Building the Library

This section lists the files that are available in the USB Audio 2.0 Device Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/usb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
usb_device_audio_v2_0.h	This header file must be included in every source file that needs to invoke USB Audio 2.0 Device Driver APIs.
usb_audio_v2_0.h	This header file must be included when the audio 2.0 descriptor macros are used.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/usb_device_audio_v2_0.c	This file contains all of functions, macros, definitions, variables, datatypes, etc., that are specific to the USB Audio v2.0 Specification implementation of the Audio 2.0 Function Driver.
/src/dynamic/usb_device_audio2_read_write.c	Contains implementation of the audio 2.0 function driver read and write functions.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The USB Audio 2.0 Device Library depends on the following modules:

- USB Device Library

Library Interface

This section describes the Application Programming Interface (API) functions of the USB Device Audio Library

Refer to each section for a detailed description.

a) Functions

Functions

	Name	Description
	USB_DEVICE_AUDIO_V2_EventHandlerSet	This function registers an event handler for the specified Audio function driver instance.
	USB_DEVICE_AUDIO_V2_Read	This function requests a data read from the USB Device Audio v2.0 Function Driver Layer.
	USB_DEVICE_AUDIO_V2_TransferCancel	This function cancels a scheduled Audio v2.0 Device data transfer.
	USB_DEVICE_AUDIO_V2_Write	This function requests a data write to the USB Device Audio v2.0 Function Driver Layer.

Description

USB_DEVICE_AUDIO_V2_EventHandlerSet Function

This function registers an event handler for the specified Audio function driver instance.

File

usb_device_audio_v2_0.h

C

```
USB_DEVICE_AUDIO_V2_RESULT USB_DEVICE_AUDIO_V2_EventHandlerSet(USB_DEVICE_AUDIO_V2_INDEX instanceIndex,
USB_DEVICE_AUDIO_V2_EVENT_HANDLER eventHandler, uintptr_t context);
```

Returns

- USB_DEVICE_AUDIO_V2_RESULT_OK - The operation was successful
- USB_DEVICE_AUDIO_V2_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.
- USB_DEVICE_AUDIO_V2_RESULT_ERROR_PARAMETER_INVALID - The eventHandler parameter is NULL

Description

This function registers a event handler for the specified Audio function driver instance. This function should be called by the application when it receives a SET CONFIGURATION event from the device layer. The application must register an event handler with the function driver in order to receive and respond to function driver specific events and control transfers. If the event handler is not registered, the device layer will stall function driver specific commands and the USB device may not function.

Remarks

None.

Preconditions

This function should be called when the function driver has been initialized as a result of a set configuration.

Example

```
// The following code shows an example registering an event handler. The
// application specifies the context parameter as a pointer to an
// application object (appObject) that should be associated with this
// instance of the Audio function driver.

USB_DEVICE_AUDIO_V2_RESULT result;

USB_DEVICE_AUDIO_V2_EVENT_RESPONSE APP_USBDeviceAUDIOEventHandler
(
    USB_DEVICE_AUDIO_V2_INDEX instanceIndex ,
    USB_DEVICE_AUDIO_V2_EVENT event ,
    void* pData,
    uintptr_t context
)
{
    // Event Handling comes here

    switch(event)
    {
        ...
    }

    return(USB_DEVICE_AUDIO_V2_EVENT_RESPONSE_NONE);
}

result = USB_DEVICE_AUDIO_V2_EventHandlerSet ( USB_DEVICE_AUDIO_V2_INSTANCE_0 ,
            &APP_USBDeviceAUDIOEventHandler, (uintptr_t) &appObject);

if(USB_DEVICE_AUDIO_V2_RESULT_OK != result)
{
    // Do error handling here
}
```

Parameters

Parameters	Description
instance	Instance of the Audio v2.0 Function Driver.
eventHandler	A pointer to event handler function.
context	Application specific context that is returned in the event handler.

Function

```
USB_DEVICE_AUDIO_V2_RESULT USB_DEVICE_AUDIO_V2_EventHandlerSet
(
    USB_DEVICE_AUDIO_V2_INDEX instance ,
    USB_DEVICE_AUDIO_V2_EVENT_HANDLER eventHandler ,
uintptr_t context
);
```

USB_DEVICE_AUDIO_V2_Read Function

This function requests a data read from the USB Device Audio v2.0 Function Driver Layer.

File

usb_device_audio_v2_0.h

C

```
USB_DEVICE_AUDIO_V2_RESULT USB_DEVICE_AUDIO_V2_Read(USB_DEVICE_AUDIO_V2_INDEX instanceIndex,
USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE* transferHandle, uint8_t interfaceNumber, void * data, size_t size);
```

Returns

- USB_DEVICE_AUDIO_V2_RESULT_OK - The read request was successful. transferHandle contains a valid transfer handle.
- USB_DEVICE_AUDIO_V2_RESULT_ERROR_TRANSFER_QUEUE_FULL - internal request queue is full. The read request could not be

added.

- USB_DEVICE_AUDIO_V2_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.
- USB_DEVICE_AUDIO_V2_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application and is invalid.

Description

This function requests a data read from the USB Device Audio v2.0 Function Driver Layer. The function places a requests with driver, the request will get serviced as data is made available by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_AUDIO_V2_EVENT_READ_COMPLETE event. The amount of data read and the transfer handle associated with the request is returned along with the event. The transfer handle expires when event handler for the USB_DEVICE_AUDIO_V2_EVENT_READ_COMPLETE exits. If the read request could not be accepted, the function returns an error code and transferHandle will contain the value USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE_INVALID.

Remarks

While the using the Audio Function Driver with PIC32MZ USB module, the audio buffer provided to the USB_DEVICE_AUDIO_V2_Read function should be placed in coherent memory and aligned at a 16 byte boundary. This can be done by declaring the buffer using the __attribute__((coherent, aligned(16))) attribute, as shown in the following example:

```
uint8_t data[256] __attribute__((coherent, aligned(16)));
```

Preconditions

The function driver should have been configured.

Example

```
// Shows an example of how to read. This assumes that
// device had been configured. The example attempts to read
// data from interface 1.

USB_DEVICE_AUDIO_V2_INDEX instanceIndex;
USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE transferHandle;
unit8_t interfaceNumber;
// Use this attribute for PIC32MZ __attribute__((coherent, aligned(16)))
unit8_t rxBuffer[192];
USB_DEVICE_AUDIO_V2_RESULT readRequestResult;

instanceIndex = 0; //specify the Audio v2.0 Function driver instance number.
interfaceNumber = 1; //Specify the Audio v2.0 Streaming interface number.

readRequestResult = USB_DEVICE_AUDIO_V2_Read ( instanceIndex, &transferHandle,
                        interfaceNumber, &rxBuffer, 192);

if(USB_DEVICE_AUDIO_V2_RESULT_OK != readRequestResult)
{
    //Do Error handling here
}

// The completion of the read request will be indicated by the
// USB_DEVICE_AUDIO_V2_EVENT_READ_COMPLETE event. The transfer handle
// and the amount of data read will be returned along with the
// event.
```

Parameters

Parameters	Description
instance	USB Device Audio v2.0 Function Driver instance.
transferHandle	Pointer to a USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the read request was successful.
interfaceNum	The USB Audio v2.0 streaming interface number on which read request is to placed.
data	pointer to the data buffer where read data will be stored. In case of PIC32MZ device, this buffer should be located in coherent memory and should be aligned a 16 byte boundary.
size	Size of the data buffer. Refer to the description section for more details on how the size affects the transfer.

Function

```
USB_DEVICE_AUDIO_V2_RESULT USB_DEVICE_AUDIO_V2_Read
(
    USB_DEVICE_AUDIO_V2_INDEX instanceIndex ,
```

```
    USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE* transferHandle,
uint8_t interfaceNum ,
void * data ,
size_t size
);
```

USB_DEVICE_AUDIO_V2_TransferCancel Function

This function cancels a scheduled Audio v2.0 Device data transfer.

File

usb_device_audio_v2_0.h

C

```
USB_DEVICE_AUDIO_V2_RESULT USB_DEVICE_AUDIO_V2_TransferCancel(USB_DEVICE_AUDIO_V2_INDEX instanceIndex,
USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE transferHandle);
```

Returns

- USB_DEVICE_AUDIO_V2_RESULT_OK - The transfer will be canceled completely or partially.
- USB_DEVICE_AUDIO_V2_RESULT_ERROR - The transfer could not be canceled because it has either completed, the transfer handle is invalid or the last transaction is in progress.

Description

This function cancels a scheduled Audio v2.0 Device data transfer. The transfer could have been scheduled using the USB_DEVICE_AUDIO_V2_Read, USB_DEVICE_AUDIO_V2_Write, or the USB_DEVICE_AUDIO_V2_SerialStateNotificationSend function. If a transfer is still in the queue and its processing has not started, the transfer is canceled completely. A transfer that is in progress may or may not get canceled depending on the transaction that is presently in progress. If the last transaction of the transfer is in progress, the transfer will not be canceled. If it is not the last transaction in progress, the in-progress will be allowed to complete. Pending transactions will be canceled. The first transaction of an in progress transfer cannot be canceled.

Remarks

None.

Preconditions

The USB Device should be in a configured state.

Example

```
// The following code example cancels an Audio transfer.

USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE transferHandle;
USB_DEVICE_AUDIO_V2_RESULT result;

result = USB_DEVICE_AUDIO_V2_TransferCancel(instanceIndex, transferHandle);

if(USB_DEVICE_AUDIO_V2_RESULT_OK == result)
{
    // The transfer cancellation was either completely or
    // partially successful.
}
```

Parameters

Parameters	Description
instanceIndex	AUDIO v2.0 Function Driver instance index.
transferHandle	Transfer handle of the transfer to be canceled.

Function

```
  USB_DEVICE_AUDIO_V2_RESULT USB_DEVICE_AUDIO_V2_TransferCancel
(
    USB_DEVICE_AUDIO_V2_INDEX instanceIndex,
    USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE transferHandle
);
```

USB_DEVICE_AUDIO_V2_Write Function

This function requests a data write to the USB Device Audio v2.0 Function Driver Layer.

File

usb_device_audio_v2_0.h

C

```
USB_DEVICE_AUDIO_V2_RESULT USB_DEVICE_AUDIO_V2_Write(USB_DEVICE_AUDIO_V2_INDEX instanceIndex,
USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE * transferHandle, uint8_t interfaceNumber, void * data, size_t size);
```

Returns

- USB_DEVICE_AUDIO_V2_RESULT_OK - The read request was successful. transferHandle contains a valid transfer handle.
- USB_DEVICE_AUDIO_V2_RESULT_ERROR_TRANSFER_QUEUE_FULL - internal request queue is full. The write request could not be added.
- USB_DEVICE_AUDIO_V2_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.
- USB_DEVICE_AUDIO_V2_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application and is invalid.

Description

This function requests a data write to the USB Device Audio v2.0 Function Driver Layer. The function places a requests with driver, the request will get serviced as data is requested by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_AUDIO_V2_EVENT_WRITE_COMPLETE event. The amount of data written and the transfer handle associated with the request is returned along with the event in writeCompleteData member of the pData parameter in the event handler.

The transfer handle expires when event handler for the USB_DEVICE_AUDIO_V2_EVENT_WRITE_COMPLETE exits. If the write request could not be accepted, the function returns an error code and transferHandle will contain the value USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE_INVALID.

Remarks

While the using the Audio Function Driver with the PIC32MZ USB module, the audio buffer provided to the USB_DEVICE_AUDIO_V2_Write function should be placed in coherent memory and aligned at a 16 byte boundary. This can be done by declaring the buffer using the __attribute__((coherent, aligned(16))) attribute. An example is shown here

```
uint8_t data[256] __attribute__((coherent, aligned(16)));
```

Preconditions

The function driver should have been configured.

Example

```
// Shows an example of how to write audio data to the audio streaming
// interface. This assumes that device is configured and the audio
// streaming interface is 1.

USB_DEVICE_AUDIO_V2_INDEX instanceIndex;
USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE transferHandle;
unit8_t interfaceNumber;
unit8_t txBuffer[192]; // Use this attribute for PIC32MZ __attribute__((coherent, aligned(16)))
USB_DEVICE_AUDIO_V2_RESULT writeRequestResult;

instanceIndex = 0; //specify the Audio Function driver instance number.
interfaceNumber = 1; //Specify the Audio Streaming interface number.

writeRequestResult = USB_DEVICE_AUDIO_V2_Write ( instanceIndex, &transferHandle,
                            interfaceNumber, &txBuffer, 192);

if(USB_DEVICE_AUDIO_V2_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

// The completion of the write request will be indicated by the
// USB_DEVICE_AUDIO_V2_EVENT_WRITE_COMPLETE event. The transfer handle
// and transfer size is provided along with this event.
```

Parameters

Parameters	Description
instance	USB Device Audio Function Driver instance.
transferHandle	Pointer to a USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the write request was successful.
interfaceNum	The USB Audio streaming interface number on which the write request is to placed.
data	pointer to the data buffer contains the data to be written. In case of PIC32MZ device, this buffer should be located in coherent memory and should be aligned a 16 byte boundary.
size	Size of the data buffer.

Function

```
USB_DEVICE_AUDIO_V2_RESULT USB_DEVICE_AUDIO_V2_Write
(
    USB_DEVICE_AUDIO_V2_INDEX instance ,
    USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE* transferHandle,
uint8_t interfaceNum ,
void * data ,
size_t size
);
```

b) Data Types and Constants

Enumerations

	Name	Description
	USB_DEVICE_AUDIO_V2_EVENT	USB Device Audio v2.0 Function Driver events.
	USB_DEVICE_AUDIO_V2_RESULT	USB Device Audio Function Driver USB Device Audio v2.0 result enumeration.

Macros

	Name	Description
	USB_DEVICE_AUDIO_V2_EVENT_RESPONSE_NONE	USB Device Audio v2.0 Function Driver event handler response type none.
	USB_DEVICE_AUDIO_V2_FUNCTION_DRIVER	USB Device Audio v2.0 Function Driver function pointer.
	USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE_INVALID	USB Device Audio v2.0 Function Driver Invalid Transfer Handle Definition.

Structures

	Name	Description
	USB_DEVICE_AUDIO_V2_EVENT_DATA_READ_COMPLETE	USB Device Audio Function Driver Audio v2.0 read and write complete event data.
	USB_DEVICE_AUDIO_V2_EVENT_DATA_SET_ALTERNATE_INTERFACE	USB Device Audio v2.0 Function Driver alternate interface setting event data.
	USB_DEVICE_AUDIO_V2_EVENT_DATA_WRITE_COMPLETE	USB Device Audio Function Driver Audio v2.0 read and write complete event data.
	USB_DEVICE_AUDIO_V2_INIT	USB Device Audio v2.0 Function Driver initialization data structure.

Types

	Name	Description
	USB_DEVICE_AUDIO_V2_EVENT_HANDLER	USB Device Audio v2.0 Event Handler Function Pointer Type.
	USB_DEVICE_AUDIO_V2_EVENT_RESPONSE	USB Device Audio v2.0 Function Driver event callback response type.
	USB_DEVICE_AUDIO_V2_INDEX	USB Device Audio v2.0 Function Driver index.
	USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE	USB Device Audio v2.0 Function Driver Transfer Handle Definition.

Description

USB_DEVICE_AUDIO_V2_EVENT Enumeration

USB Device Audio v2.0 Function Driver events.

File

usb_device_audio_v2_0.h

C

```
typedef enum {
  USB_DEVICE_AUDIO_V2_EVENT_WRITE_COMPLETE,
  USB_DEVICE_AUDIO_V2_EVENT_READ_COMPLETE,
  USB_DEVICE_AUDIO_V2_EVENT_INTERFACE_SETTING_CHANGED,
  USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER_DATA_RECEIVED,
  USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER_DATA_SENT,
  USB_DEVICE_AUDIO_V2_CUR_ENTITY_SETTINGS_RECEIVED,
  USB_DEVICE_AUDIO_V2_RANGE_ENTITY_SETTINGS_RECEIVED,
  USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER_UNKNOWN
} USB_DEVICE_AUDIO_V2_EVENT;
```

Members

Members	Description
USB_DEVICE_AUDIO_V2_EVENT_WRITE_COMPLETE	This event occurs when a write operation scheduled by calling the USB_DEVICE_AUDIO_V2_Write function has completed. The pData member in the event handler will point to USB_DEVICE_AUDIO_V2_EVENT_WRITE_COMPLETE_DATA type.
USB_DEVICE_AUDIO_V2_EVENT_READ_COMPLETE	This event occurs when a read operation scheduled by calling the USB_DEVICE_AUDIO_V2_Read function has completed. The pData member in the event handler will point to USB_DEVICE_AUDIO_V2_EVENT_READ_COMPLETE_DATA type.
USB_DEVICE_AUDIO_V2_EVENT_INTERFACE_SETTING_CHANGED	This event occurs when the Host requests the Audio v2.0 USB Device to set an alternate setting on an interface present in this audio function. An Audio v2.0 USB Device will typically feature a default interface setting and one or more alternate interface settings. The pData member in the event handler will point to the USB_DEVICE_AUDIO_V2_EVENT_DATA_INTERFACE_SETTING_CHANGED type. This contains the index of the interface whose setting must be changed and the index of the alternate setting. The application may enable or disable audio functions based on the interface setting.
USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	This event occurs when the data stage of a control write transfer has completed. This would occur after the application would respond with a USB_DEVICE_ControlReceive function, which may possibly have been called in response to a USB_DEVICE_AUDIO_V2_EVENT_ENTITY_SETTINGS_RECEIVED event This event notifies the application that the data is received from Host and is available at the location passed by the USB_DEVICE_ControlReceive function. If the received data is acceptable to the application, it should acknowledge the data by calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_OK flag.The application can reject the received data by calling the USB_DEVICE_ControlStatus function with the USB_DEVICE_CONTROL_STATUS_ERROR flag. The pData parameter will be NULL.
USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER_DATA_SENT	This event occurs when the data stage of a control read transfer has completed. This would occur when the application has called the USB_DEVICE_ControlSend function to complete the data stage of a control transfer. The event indicates that the data has been transmitted to the host. The pData parameter will be NULL.

USB_DEVICE_AUDIO_V2_CUR_ENTITY_SETTINGS_RECEIVED	This event occurs when the Host sends an Audio 2.0 Control specific Set Current Setting Attribute Control Transfer request to an Audio Device Control. The pData member in the event handler will point to type. The application must use the entityID, interface, endpoint and the wValue field in the event data to determine the entity and control type and then respond to the control transfer with the USB_DEVICE_ControlStatus and USB_DEVICE_ControlReceive functions.
USB_DEVICE_AUDIO_V2_RANGE_ENTITY_SETTINGS_RECEIVED	This event occurs when the Host sends an Audio 2.0 Control specific Set Range Setting Attribute Control Transfer request to an Audio Device Control. The pData member in the event handler will point to type. The application must use the entityID, interface, endpoint and the wValue field in the event data to determine the entity and control type and then respond to the control transfer with a USB_DEVICE_ControlStatus and USB_DEVICE_ControlReceive functions.
USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER_UNKNOWN	This event occurs when the Audio v2.0 function driver receives a control transfer request that could not be decoded by Audio Function driver.The pData parameter will point to a USB_SETUP_PACKET type containing the setup packet. The application must analyze this Setup packet and use the USB_DEVICE_ControlSend, USB_DEVICE_ControlReceive, or the USB_DEVICE_ControlStatus function to advance the control transfer or complete it.

Description

USB Device Audio v2.0 Function Driver Events

These events are specific to a USB Device Audio v2.0 Function Driver instance. An event may have some data associated with it. This is provided to the event handling function. Each event description contains details about this event data (pData) and other parameters passed along with the event, to the event handler.

Events associated with the Audio v2.0 Function Driver Specific Control Transfers require application response. The application should respond to these events by using the USB_DEVICE_ControlReceive(), USB_DEVICE_ControlSend() and USB_DEVICE_ControlStatus() functions.

Calling the USB_DEVICE_ControlStatus() function with a USB_DEVICE_CONTROL_STATUS_ERROR will stall the control transfer request. The application would do this if the control transfer request is not supported. Calling the USB_DEVICE_ControlStatus() function with a USB_DEVICE_CONTROL_STATUS_OK will complete the status stage of the control transfer request. The application would do this if the control transfer request is supported.

The following code shows an example of a possible event handling scheme.

```
// This code example shows all USB Audio v2.0 Function Driver possible
// events and a possible scheme for handling these events.
// In this case event responses are not deferred.

void APP_USBDeviceAudioEventHandler
(
    USB_DEVICE_AUDIO_V2_INDEX instanceIndex ,
    USB_DEVICE_AUDIO_V2_EVENT event ,
    void * pData,
    uintptr_t context
)
{
    switch (event)
    {
        case USB_DEVICE_AUDIO_V2_EVENT_READ_COMPLETE:

            // This event indicates that a Audio v2.0 Read Transfer request
            // has completed. pData should be interpreted as a
            // USB_DEVICE_AUDIO_V2_EVENT_DATA_READ_COMPLETE pointer type.
            // This contains the transfer handle of the read transfer
            // that completed and amount of data that was read.

            break;

        case USB_DEVICE_AUDIO_V2_EVENT_WRITE_COMPLETE:

            // This event indicates that a Audio v2.0 Write Transfer request
            // has completed. pData should be interpreted as a
            // USB_DEVICE_AUDIO_V2_EVENT_DATA_WRITE_COMPLETE pointer type.
            // This contains the transfer handle of the write transfer
            // that completed and amount of data that was written.
```

```
            break;

        case USB_DEVICE_AUDIO_V2_EVENT_INTERFACE_SETTING_CHANGED:

            // This event occurs when the host sends Set Interface request
            // to the Audio v2.0 USB Device. pData will be a pointer to a
            // USB_DEVICE_AUDIO_V2_EVENT_DATA_INTERFACE_SETTING_CHANGED. This
            // contains the interface number whose setting was
            // changed and the index of the alternate setting.
            // The application should typically enable the audio function
            // if the interfaceAlternateSettting member of pData is greater
            // than 0.

            break;

        case USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER_UNKNOWN:

            // This event indicates that the Audio v2.0 function driver has
            // received a control transfer which it cannot decode. pData
            // will be a pointer to USB_SETUP_PACKET type pointer. The
            // application should decode the packet and take the required
            // action using the USB_DEVICE_ControlStatus(),
            // USB_DEVICE_ControlSend() and USB_DEVICE_ControlReceive()
            // functions.

            break;

        case USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER_DATA_SENT:

            // This event indicates the data send request associated with
            // the latest USB_DEVICE_ControlSend() function was
            // completed. pData will be NULL.

        case USB_DEVICE_AUDIO_V2_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:

            // This event indicates the data receive request associated with
            // the latest USB_DEVICE_ControlReceive() function was
            // completed. pData will be NULL. The application can either
            // acknowledge the received data or reject it by calling the
            // USB_DEVICE_ControlStatus() function.

            break;

        case USB_DEVICE_AUDIO_V2_CUR_ENTITY_SETTINGS_RECEIVED:
            // This event indicates the Current entity request has been
            // received.
            USB_AUDIO_CONTROL_INTERFACE_REQUEST* controlRequest;
            controlRequest = (USB_AUDIO_CONTROL_INTERFACE_REQUEST*) setupPkt;
            switch(controlRequest->entityID)
            {
                case APP_ID_CLOCK_SOURCE:
                    USB_AUDIO_CLOCKSOURCE_CONTROL_REQUEST*
                                                        clockSourceRequest;
                    clockSourceRequest =
                    (USB_AUDIO_CLOCKSOURCE_CONTROL_REQUEST*) controlRequest;

                    if (clockSourceRequest->bRequest == CUR)
                    {
                        switch(clockSourceRequest->controlSelector)
                        {
                            case CS_SAM_FREQ_CONTROL:
                            {
                                if ((controlRequest->bmRequestType & 0x80)
                                                                    == 0)
                                {
                                    //A control write transfer received
                                    //from Host. Now receive data from Host.
                                    USB_DEVICE_ControlReceive(
```

```
                                        appData.usbDevHandle,
                                    void *) &(appData.clockSource),
                                                4 );
                                appData.currentAudioControl =
                                      APP_USB_AUDIO_CLOCKSOURCE_CONTROL;
                            }
                            else
                            {
                                //Handle Get request
                                USB_DEVICE_ControlSend(
                                                appData.usbDevHandle,
                                        (void *)&(appData.clockSource),
                                                  4 );
                                appData.currentAudioControl =
                                                APP_USB_CONTROL_NONE;
                            }
                        }
                        break;

                        case CS_CLOCK_VALID_CONTROL:
                        {
                            if ((controlRequest->bmRequestType & 0x80)
                                                                == 0x80)
                            {
                                //Handle Get request
                                USB_DEVICE_ControlSend(
                                                appData.usbDevHandle,
                                        (void *)&(appData.clockValid),
                                                    1);
                            }
                            else
                            {
                                 USB_DEVICE_ControlStatus(
                                                appData.usbDevHandle,
                                      USB_DEVICE_CONTROL_STATUS_ERROR);

                            }
                        }
                        break;

                        default:
                            //This USB Audio Speaker application does
                            //not support any other feature unit request
                            //from Host. So Stall if any other feature
                            //unit request received from Host.
                            USB_DEVICE_ControlStatus (
                                                appData.usbDevHandle,
                                 USB_DEVICE_CONTROL_STATUS_ERROR);
                        break;

                    }
                }

            case USB_DEVICE_AUDIO_V2_RANGE_ENTITY_SETTINGS_RECEIVED:
                // This event indicates the Range entity request has been
                // received.
                USB_AUDIO_CONTROL_INTERFACE_REQUEST* controlRequest;
                controlRequest = (USB_AUDIO_CONTROL_INTERFACE_REQUEST*)setupPkt;
                switch(controlRequest->entityID)
                {
                    case APP_ID_CLOCK_SOURCE:
                        USB_AUDIO_CLOCKSOURCE_CONTROL_REQUEST* clockSourceRequest;
                        clockSourceRequest =
                        (USB_AUDIO_CLOCKSOURCE_CONTROL_REQUEST*) controlRequest;
                        if (clockSourceRequest->bRequest == RANGE)
                        {
                            switch(clockSourceRequest->controlSelector)
                            {
                                case CS_SAM_FREQ_CONTROL:
```

```
                {
                    if ((controlRequest->bmRequestType & 0x80)
                                                        == 0x80)
                    {
                        //A control read transfer received from
                        // Host. Now send data to Host.
                        USB_DEVICE_ControlSend(
                                            appData.usbDevHandle,
                            void *) &(appData.clockSourceRange),
                              sizeof(appData.clockSourceRange));
                    }
                    else
                    {
                        //Handle Get request
                        // USB_DEVICE_ControlReceive(
                                            appData.usbDevHandle,
                         (void *)&(appData.clockSourceRange[0]),
                             sizeof(appData.clockSourceRange) );
                        USB_DEVICE_ControlStatus(
                                            appData.usbDevHandle,
                               USB_DEVICE_CONTROL_STATUS_ERROR);
                    }
                }
                break;

                default:
                    //This USB Audio Speaker application does
                    // not support any other feature unit
                    // request from Host. So Stall if any other
                    // feature unit request received from Host.
                    USB_DEVICE_ControlStatus (
                                            appData.usbDevHandle,
                               USB_DEVICE_CONTROL_STATUS_ERROR);
                break;

            }
        }

    }
```

Remarks

The application can defer responses to events triggered by control transfers. In that, the application can respond to the control transfer event after exiting the event handler. This allows the application some time to obtain the response data rather than having to respond to the event immediately. Note that a USB host will typically wait for an event response for a finite time duration before timing out and canceling the event and associated transactions. Even when deferring response, the application must respond promptly if such time-out have to be avoided.

USB_DEVICE_AUDIO_V2_EVENT_DATA_READ_COMPLETE Structure

USB Device Audio Function Driver Audio v2.0 read and write complete event data.

File

usb_device_audio_v2_0.h

C

```
typedef struct {
  USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE handle;
  uint16_t length;
  uint8_t interfaceNum;
  USB_DEVICE_AUDIO_V2_RESULT status;
} USB_DEVICE_AUDIO_V2_EVENT_DATA_WRITE_COMPLETE, USB_DEVICE_AUDIO_V2_EVENT_DATA_READ_COMPLETE;
```

Members

Members	Description
USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE handle;	Transfer handle associated with this • read or write request

uint16_t length;	Indicates the amount of data (in bytes) that was • read or written
uint8_t interfaceNum;	Interface Number
USB_DEVICE_AUDIO_V2_RESULT status;	Completion status of the transfer

Description

USB Device Audio v2.0 Function Driver Read and Write Complete Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_AUDIO_V2_EVENT_READ_COMPLETE, USB_DEVICE_AUDIO_V2_EVENT_WRITE_COMPLETE, events.

Remarks

None.

USB_DEVICE_AUDIO_V2_EVENT_DATA_SET_ALTERNATE_INTERFACE Structure

USB Device Audio v2.0 Function Driver alternate interface setting event data.

File

usb_device_audio_v2_0.h

C

```
typedef struct {
  uint8_t interfaceNumber;
  uint8_t interfaceAlternateSetting;
} USB_DEVICE_AUDIO_V2_EVENT_DATA_SET_ALTERNATE_INTERFACE;
```

Members

Members	Description
uint8_t interfaceNumber;	Interface number of the interface who setting is to be changed
uint8_t interfaceAlternateSetting;	Alternate setting number

Description

USB Device Audio v2.0 Function Driver Alternate Interface Setting Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_AUDIO_V2_EVENT_DATA_INTERFACE_SETTING_CHANGED.

Remarks

None.

USB_DEVICE_AUDIO_V2_EVENT_DATA_WRITE_COMPLETE Structure

USB Device Audio Function Driver Audio v2.0 read and write complete event data.

File

usb_device_audio_v2_0.h

C

```
typedef struct {
  USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE handle;
  uint16_t length;
  uint8_t interfaceNum;
  USB_DEVICE_AUDIO_V2_RESULT status;
} USB_DEVICE_AUDIO_V2_EVENT_DATA_WRITE_COMPLETE, USB_DEVICE_AUDIO_V2_EVENT_DATA_READ_COMPLETE;
```

Members

Members	Description
USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE handle;	Transfer handle associated with this • read or write request
uint16_t length;	Indicates the amount of data (in bytes) that was • read or written
uint8_t interfaceNum;	Interface Number

USB_DEVICE_AUDIO_V2_RESULT status;	Completion status of the transfer

Description

USB Device Audio v2.0 Function Driver Read and Write Complete Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_AUDIO_V2_EVENT_READ_COMPLETE, USB_DEVICE_AUDIO_V2_EVENT_WRITE_COMPLETE, events.

Remarks

None.

USB_DEVICE_AUDIO_V2_EVENT_HANDLER Type

USB Device Audio v2.0 Event Handler Function Pointer Type.

File

usb_device_audio_v2_0.h

C

```
typedef USB_DEVICE_AUDIO_V2_EVENT_RESPONSE (* USB_DEVICE_AUDIO_V2_EVENT_HANDLER)(USB_DEVICE_AUDIO_V2_INDEX instanceIndex , USB_DEVICE_AUDIO_V2_EVENT event , void * pData, uintptr_t context);
```

Description

USB Device Audio v2.0 Event Handler Function Pointer Type.

This data type defines the required function signature USB Device Audio Function Driver event handling callback function. The application must register a pointer to an Audio Function Driver events handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive event call backs from the Audio Function Driver. The function driver will invoke this function with event relevant parameters. The description of the event handler function parameters is given here.

instanceIndex - Instance index of the Audio v2.0 Function Driver that generated the event.

event - Type of event generated.

pData - This parameter should be type casted to an event specific pointer type based on the event that has occurred. Refer to the USB_DEVICE_AUDIO_V2_EVENT enumeration description for more details.

context - Value identifying the context of the application that registered the event handling function.

Remarks

The event handler function executes in the USB interrupt context when the USB Device Stack is configured for interrupt based operation. It is not advisable to call blocking functions or computationally intensive functions in the event handler. Where the response to a control transfer related event requires extended processing, the response to the control transfer should be deferred and the event handler should be allowed to complete execution.

USB_DEVICE_AUDIO_V2_EVENT_RESPONSE Type

USB Device Audio v2.0 Function Driver event callback response type.

File

usb_device_audio_v2_0.h

C

```
typedef void USB_DEVICE_AUDIO_V2_EVENT_RESPONSE;
```

Description

USB Device Audio v2.0 Function Driver Event Handler Response Type

This is the return type of the Audio Function Driver event handler.

Remarks

None.

USB_DEVICE_AUDIO_V2_INDEX Type

USB Device Audio v2.0 Function Driver index.

File

usb_device_audio_v2_0.h

C

```
typedef uintptr_t USB_DEVICE_AUDIO_V2_INDEX;
```

Description

USB Device Audio v2.0 Function Driver Index

This definition uniquely identifies a Audio v2.0 Function Driver instance.

Remarks

None.

USB_DEVICE_AUDIO_V2_INIT Structure

USB Device Audio v2.0 Function Driver initialization data structure.

File

usb_device_audio_v2_0.h

C

```
typedef struct {
  size_t queueSizeRead;
  size_t queueSizeWrite;
} USB_DEVICE_AUDIO_V2_INIT;
```

Members

Members	Description
size_t queueSizeRead;	Size of the read queue for this instance • of the Audio function driver
size_t queueSizeWrite;	Size of the write queue for this instance • of the Audio function driver

Description

USB Device Audio v2.0 Function Driver Initialization Data Structure

This data structure must be defined for every instance of the Audio function driver. It is passed to the Audio v2.0 function driver, by the Device Layer, at the time of initialization. The funcDriverInit member of the Device Layer Function Driver registration table entry must point to this data structure for an instance of the Audio function driver.

Remarks

The queue sizes that are specified in this data structure are also affected by the USB_DEVICE_AUDIO_V2_QUEUE_DEPTH_COMBINED configuration macro.

USB_DEVICE_AUDIO_V2_RESULT Enumeration

USB Device Audio Function Driver USB Device Audio v2.0 result enumeration.

File

usb_device_audio_v2_0.h

C

```
typedef enum {
  USB_DEVICE_AUDIO_V2_RESULT_OK,
  USB_DEVICE_AUDIO_V2_RESULT_ERROR_TRANSFER_QUEUE_FULL,
  USB_DEVICE_AUDIO_V2_RESULT_ERROR_INSTANCE_INVALID,
  USB_DEVICE_AUDIO_V2_RESULT_ERROR_INSTANCE_NOT_CONFIGURED,
  USB_DEVICE_AUDIO_V2_RESULT_ERROR_PARAMETER_INVALID,
  USB_DEVICE_AUDIO_V2_RESULT_ERROR_INVALID_INTERFACE_ID,
  USB_DEVICE_AUDIO_V2_RESULT_ERROR_INVALID_BUFFER,
  USB_DEVICE_AUDIO_V2_RESULT_ERROR_ENDPOINT_HALTED,
  USB_DEVICE_AUDIO_V2_RESULT_ERROR_TERMINATED_BY_HOST,
  USB_DEVICE_AUDIO_V2_RESULT_ERROR
} USB_DEVICE_AUDIO_V2_RESULT;
```

Members

Members	Description
USB_DEVICE_AUDIO_V2_RESULT_OK	The operation was successful
USB_DEVICE_AUDIO_V2_RESULT_ERROR_TRANSFER_QUEUE_FULL	The transfer queue is full and no new transfers can be scheduled
USB_DEVICE_AUDIO_V2_RESULT_ERROR_INSTANCE_INVALID	The specified instance is not provisioned in the system
USB_DEVICE_AUDIO_V2_RESULT_ERROR_INSTANCE_NOT_CONFIGURED	The specified instance is not configured yet
USB_DEVICE_AUDIO_V2_RESULT_ERROR_PARAMETER_INVALID	The event handler provided is NULL
USB_DEVICE_AUDIO_V2_RESULT_ERROR_INVALID_INTERFACE_ID	Interface number passed to the read or write function is invalid.
USB_DEVICE_AUDIO_V2_RESULT_ERROR_INVALID_BUFFER	A NULL buffer was specified in the read or write function
USB_DEVICE_AUDIO_V2_RESULT_ERROR_ENDPOINT_HALTED	Transfer terminated because host halted the endpoint
USB_DEVICE_AUDIO_V2_RESULT_ERROR_TERMINATED_BY_HOST	Transfer terminated by host because of a stall clear
USB_DEVICE_AUDIO_V2_RESULT_ERROR	General Error

Description

USB Device Audio v2.0 Function Driver USB Device Audio v2.0 Result enumeration.

This enumeration lists the possible USB Device Audio v2.0 Function Driver operation results.

Remarks

None.

USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE Type

USB Device Audio v2.0 Function Driver Transfer Handle Definition.

File

usb_device_audio_v2_0.h

C

```
typedef uintptr_t USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE;
```

Description

USB Device Audio v2.0 Function Driver Transfer Handle Definition

This definition defines a USB Device Audio v2.0 Function Driver Transfer Handle. A Transfer Handle is owned by the application but its value is modified by the USB_DEVICE_AUDIO_V2_Write and USB_DEVICE_AUDIO_V2_Read functions. The transfer handle is valid for the life time of the transfer and expires when the transfer related event had occurred.

Remarks

None.

USB_DEVICE_AUDIO_V2_EVENT_RESPONSE_NONE Macro

USB Device Audio v2.0 Function Driver event handler response type none.

File

usb_device_audio_v2_0.h

C

```
#define USB_DEVICE_AUDIO_V2_EVENT_RESPONSE_NONE
```

Description

USB Device Audio v2.0 Function Driver Event Handler Response None

This is the definition of the Audio v2.0 Function Driver event handler response type none.

Remarks

Intentionally defined to be empty.

USB_DEVICE_AUDIO_V2_FUNCTION_DRIVER Macro

USB Device Audio v2.0 Function Driver function pointer.

File

usb_device_audio_v2_0.h

C

```
#define USB_DEVICE_AUDIO_V2_FUNCTION_DRIVER
```

Description

USB Device Audio v2.0 Function Driver Function Pointer

This is the USB Device Audio v2.0 Function Driver Function pointer. This should registered with the device layer in the function driver registration table.

Remarks

None.

USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE_INVALID Macro

USB Device Audio v2.0 Function Driver Invalid Transfer Handle Definition.

File

usb_device_audio_v2_0.h

C

```
#define USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE_INVALID
```

Description

USB Device Audio v2.0 Function Driver Invalid Transfer Handle Definition

This definition defines a USB Device Audio v2.0 Function Driver Invalid Transfer Handle. A Invalid Transfer Handle is returned by the USB_DEVICE_Audio_V2_Write and USB_DEVICE_Audio_V2_Read functions when the request was not successful.

Remarks

None.

Files

Files

Name	Description
usb_device_audio_v2_0.h	USB Device Audio v2.0 Function Driver interface .
usb_device_audio_v2_0_config_template.h	USB device Audio 2.0 Class configuration definitions template

Description

This section lists the source and header files used by the library.

usb_device_audio_v2_0.h

USB Device Audio v2.0 Function Driver interface .

Enumerations

	Name	Description
	USB_DEVICE_AUDIO_V2_EVENT	USB Device Audio v2.0 Function Driver events.
	USB_DEVICE_AUDIO_V2_RESULT	USB Device Audio Function Driver USB Device Audio v2.0 result enumeration.

Functions

	Name	Description
	USB_DEVICE_AUDIO_V2_EventHandlerSet	This function registers an event handler for the specified Audio function driver instance.

	USB_DEVICE_AUDIO_V2_Read	This function requests a data read from the USB Device Audio v2.0 Function Driver Layer.
	USB_DEVICE_AUDIO_V2_TransferCancel	This function cancels a scheduled Audio v2.0 Device data transfer.
	USB_DEVICE_AUDIO_V2_Write	This function requests a data write to the USB Device Audio v2.0 Function Driver Layer.

Macros

	Name	Description
	USB_DEVICE_AUDIO_V2_EVENT_RESPONSE_NONE	USB Device Audio v2.0 Function Driver event handler response type none.
	USB_DEVICE_AUDIO_V2_FUNCTION_DRIVER	USB Device Audio v2.0 Function Driver function pointer.
	USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE_INVALID	USB Device Audio v2.0 Function Driver Invalid Transfer Handle Definition.

Structures

	Name	Description
	USB_DEVICE_AUDIO_V2_EVENT_DATA_READ_COMPLETE	USB Device Audio Function Driver Audio v2.0 read and write complete event data.
	USB_DEVICE_AUDIO_V2_EVENT_DATA_SET_ALTERNATE_INTERFACE	USB Device Audio v2.0 Function Driver alternate interface setting event data.
	USB_DEVICE_AUDIO_V2_EVENT_DATA_WRITE_COMPLETE	USB Device Audio Function Driver Audio v2.0 read and write complete event data.
	USB_DEVICE_AUDIO_V2_INIT	USB Device Audio v2.0 Function Driver initialization data structure.

Types

	Name	Description
	USB_DEVICE_AUDIO_V2_EVENT_HANDLER	USB Device Audio v2.0 Event Handler Function Pointer Type.
	USB_DEVICE_AUDIO_V2_EVENT_RESPONSE	USB Device Audio v2.0 Function Driver event callback response type.
	USB_DEVICE_AUDIO_V2_INDEX	USB Device Audio v2.0 Function Driver index.
	USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE	USB Device Audio v2.0 Function Driver Transfer Handle Definition.

Description

USB Device Audio Function Driver Interface

This file describes the USB Device Audio v2.0 Function Driver interface. This file should be included by the application if it needs to use the Audio v2.0 Function Driver API.

File Name

usb_device_audio_v2.0.h

Company

Microchip Technology Inc.

usb_device_audio_v2_0_config_template.h

USB device Audio 2.0 Class configuration definitions template

Macros

	Name	Description
	USB_DEVICE_AUDIO_V2_INSTANCES_NUMBER	Specifies the number of Audio 2.0 Function Driver instances.
	USB_DEVICE_AUDIO_V2_MAX_ALTERNATE_SETTING	Specifies the maximum number of Alternate Settings per streaming interface.
	USB_DEVICE_AUDIO_V2_MAX_STREAMING_INTERFACES	Specifies the maximum number of Audio 2.0 Streaming interfaces in an Audio 2.0 Function Driver instance.
	USB_DEVICE_AUDIO_V2_QUEUE_DEPTH_COMBINED	Specifies the combined queue size of all Audio 2.0 function driver instances.

Description

USB Device Audio 2.0 Class Configuration Definitions

This file contains configurations macros needed to configure the Audio 2.0 Function Driver. This file is a template file only. It should not be included

by the application. The configuration macros defined in the file should be defined in the configuration specific system_config.h.

File Name

usb_device_audio_v2_0_config_template_0_config_template.h

Company

Microchip Technology Inc.

USB Device Layer Library

This section describes the USB Device Layer Library.

Introduction

Introduces the MPLAB Harmony USB Device Layer Library.

Description

The MPLAB Harmony USB Device Layer Library (also referred to as the Device Layer) is part of the MPLAB Harmony USB Device Stack. Within the USB Device Stack, the Device Layer responds to enumeration requests from the Hosts. It receives control transfers from the Host and responds to these control transfers in case of standard device requests. It dispatches function driver and application specific control transfers to the respective function drivers and to the application. It provides the application and function drivers with API routines that allow them to respond and complete a control transfer, with a possibility of deferring such responses. The Device Layer also provides the application with events and functions that allow the application to track the state of the device.

The Device Layer plays the role of a system in the MPLAB Harmony USB Device Stack. In that, it initializes the USB Driver, the device function drivers and maintains their state machines by invoking the Tasks routines of these modules. The Device Layer thus treats the USB Driver and function drivers as sub modules.

The device layer features the following:

- Supports both USB Full-Speed and Hi-Speed operation
- Based on a modular and event-driven architecture
- Supports the PIC32MX and PIC32MZ families of microcontrollers
- Supports composite USB devices
- Supports different types of function drivers (i.e., CDC, HID, MSD, etc.)
- Supports non-blocking operation and is RTOS friendly
- Designed to integrate readily with other MPLAB Harmony middleware
- Supports both interrupt and polling operation
- Reduces the required application intervention in maintaining the USB state of the device
- Allows implementation of a multi-configuration USB device
- Functions to implement Generic USB Devices

Using the Library

This topic describes the basic architecture of the USB Device Layer Library and provides information and examples on its use.

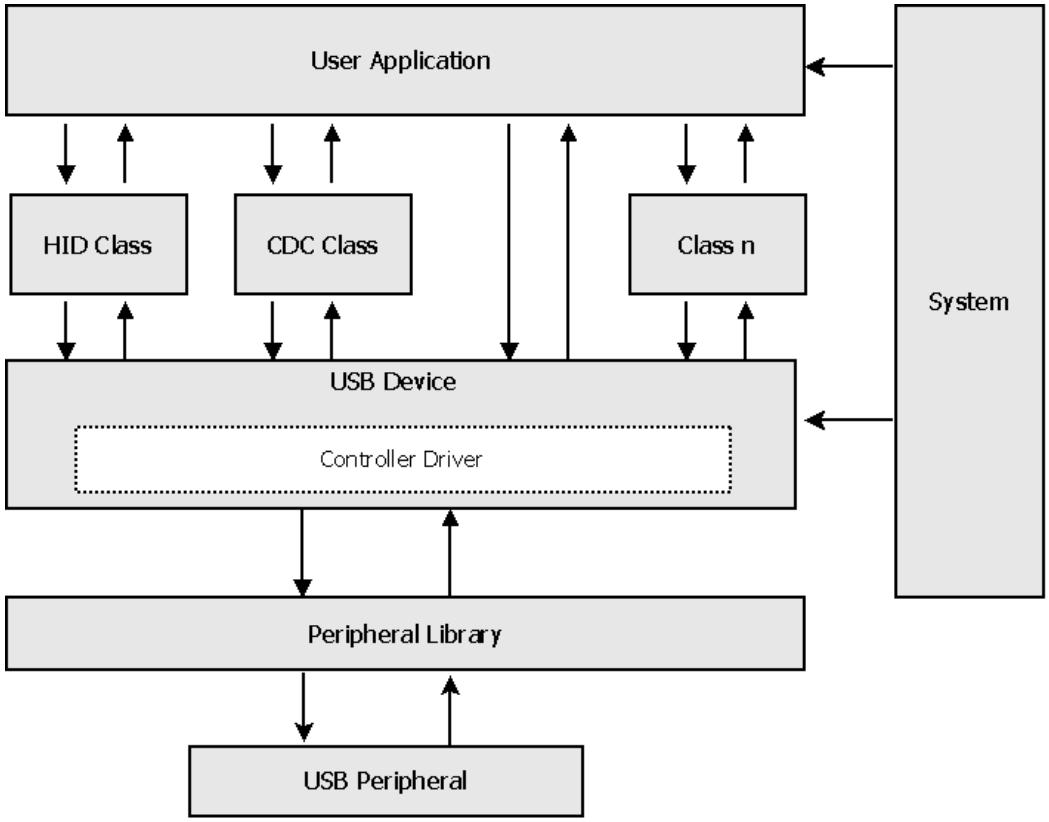
Abstraction Model

Describes the Abstraction Model of the USB Device Layer.

Description

The block diagram shows USB Device Layer interaction with USB controller driver, function drivers, user application and the system.

USB Device Layer Software Abstraction Block Diagram



System Interaction

The system is responsible for initializing and deinitializing the device layer. It is also responsible for calling the USB Device Layer task routine.

Function Driver Interaction

The USB Device Layer interacts with a function driver for following reasons:

- Initializes the function driver when device is configured by the Host. This can happen when the Host issues a set configuration request to the device. The device layer initializes only those function drivers that are valid for the selected configuration.
- Deinitializes the function driver when the Host issues a bus reset or when device is detached from the host or when the Host unconfigures the device by setting configuration value to '0'.
- Function driver and USB Controller Driver task routines are run by the device layer. This means that these task routines run at the same priority as a device layer task.
- Forwards class/interface specific setup requests from host to function drivers for processing. The function drivers can use device layer APIs to read and write data to Endpoint 0.

All of these interactions are initiated by the device layer, and therefore, it is required for a function driver to register a set of standard APIs with the device layer for initializing/deinitializing the function driver, for handling control transfers and for running the task routines. Registering of these callback functions with the device layer is a compile time step and is done using the function driver registration table. Function driver registration is explained in subsequent sections.

User Application (Client) Interaction

User application clients can register a callback function with the device layer to get USB device events. Apart from device events, the clients can interact with USB device layer to determine other status such as USB speed and remote wake-up. The Device Layer will forward Control Transfers whose Recipient field is set to Other, to the application. The application must use the Device Layer Control Transfer Routines to complete the control transfer.

Library Overview

The USB device layer mainly interacts with the system, its clients and function drivers, as shown in the Abstraction Model.

The library interface routines are divided into sub-sections, which address one of the blocks or the overall operation of the USB Device Layer Library.

Library Interface Section	Description
System Interaction Functions	Provides system module interfaces, device initialization, deinitialization, reinitialization and task functions.
Client Core Functions	Provides function to register callbacks, mechanism to pass events to clients and functions to know the status.

Device Power State Management Functions	These functions manage the power state of the device (self or bus powered) and remote wake-up.
Endpoint Management Functions	These functions allow the application to manage endpoints (enable, disable and stall).
Device Management Functions	These functions allow the application to manage the state of the device (attach, detach etc.).
Control Transfer Functions	These functions allow the application to respond to complete control transfers.

How the Library Works

Library Initialization

Describes how the USB Device Layer must be initialized.

Description

The Device Layer initialization process requires the following components:

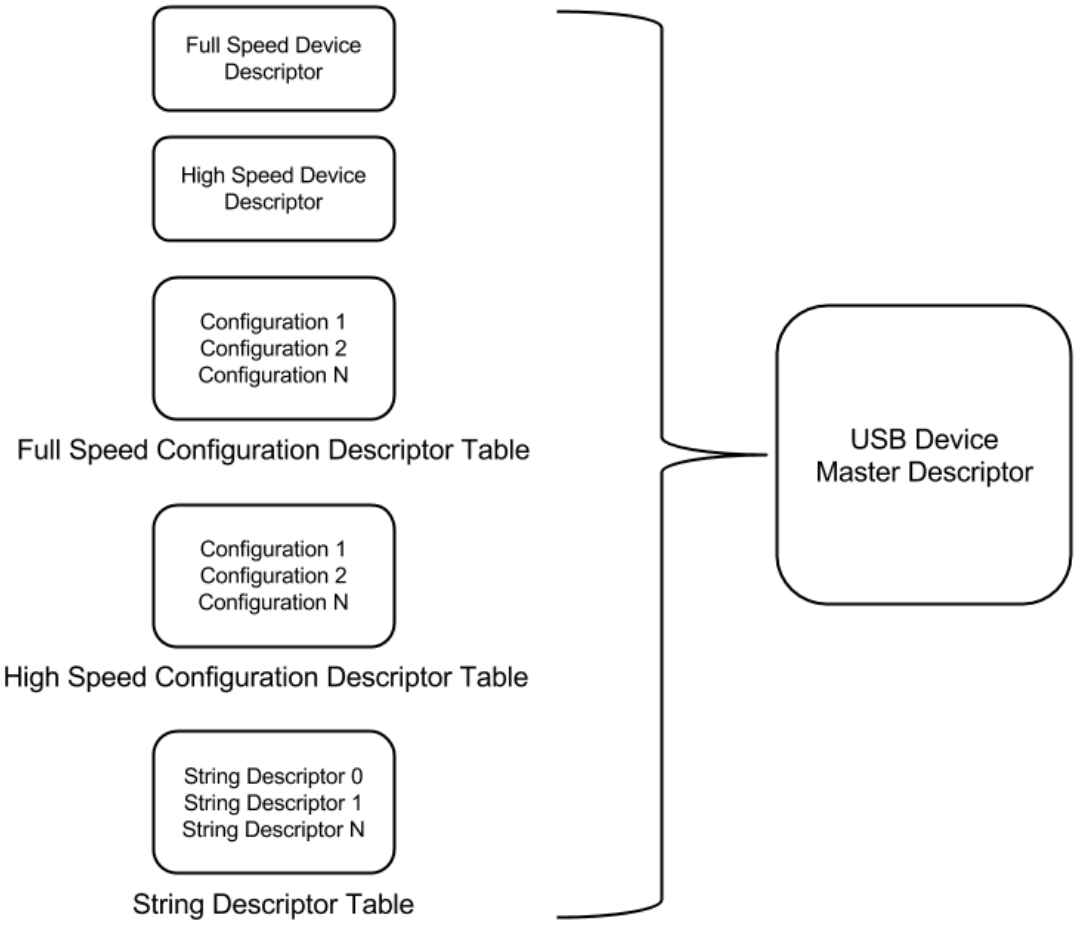
- USB Standard Descriptors that define the device functionality. The definitions of these descriptors are defined by the USB 2.0 and Device Class specification.
- Device Master Descriptor Table
- Function Driver Registration Table

The USB Standard Descriptors that define the device functionality are discussed in detail in the USB 2.0 and Device Class Specifications. The reader is encouraged to refer to these specifications for a detailed understanding of this topic.

Master Descriptor Table

Describes the USB Device Layer Master Descriptor Table.

Description



As seen in the figure, the Device Master Descriptor Table (specified by the USB_DEVICE_MASTER_DESCRIPTOR data type) is a container for all descriptor related information that is needed by the Device Layer for its operation. This table contains the following information:

- Pointer to the Full-Speed and High-Speed Device Descriptor
- Number of Full-Speed and High-Speed Configurations
- Pointers to Table of Full-Speed and High-Speed Configuration Descriptors
- Number of String Descriptors
- Pointer to a Table of String Descriptors
- Pointers to Full-Speed and High-Speed Device Qualifier

In a case where a particular item in the Device Master Descriptor Table is not applicable, that entry can be either set to '0' or NULL as applicable. For example for a Full-Speed-only device, the number of High Speed Configuration should be set to '0' and the pointer to the table of High-Speed Configuration Descriptors should be set to NULL.

The following code shows an example of a USB Device Master Descriptor design for a Full-Speed USB HID Keyboard.

```
/*********************************************
 * USB Device Layer Master Descriptor Table
 *********************************************/
const USB_DEVICE_MASTER_DESCRIPTOR usbMasterDescriptor =
{
    &fullSpeedDeviceDescriptor,     /* Full-speed descriptor */
    1,                              /* Total number of full-speed configurations available */
    &fullSpeedConfigDescSet[0],     /* Pointer to array of full-speed configurations descriptors*/

    NULL,                           /* High-speed device descriptor is not supported*/
    0,                              /* Total number of high-speed configurations available */
    NULL,                           /* Pointer to array of high-speed configurations descriptors. Not
supported*/

    3,                              /* Total number of string descriptors available */
    stringDescriptors,              /* Pointer to array of string descriptors */

    NULL,                           /* Pointer to full-speed device qualifier. Not supported */
    NULL,                           /* Pointer to high-speed device qualifier. Not supported */
};
```

The following code shows an example of a USB Device Master Descriptor design for a Full Speed/High Speed USB HID Keyboard.

```
/*********************************************
 * USB Device Layer Master Descriptor Table
 *********************************************/
const USB_DEVICE_MASTER_DESCRIPTOR usbMasterDescriptor =
{
    &fullSpeedDeviceDescriptor,     /* Full-speed descriptor */
    1,                              /* Total number of full-speed configurations available */
    &fullSpeedConfigDescSet[0],     /* Pointer to array of full-speed configurations descriptors*/

    &highSpeedDeviceDescriptor,     /* High-speed descriptor */
    1,                              /* Total number of high-speed configurations available */
    &highSpeedConfigDescSet[0],     /* Pointer to array of high-speed configurations descriptors*/

    3,                              /* Total number of string descriptors available */
    stringDescriptors,              /* Pointer to array of string descriptors */

    &deviceQualifierDescriptor1,    /* Pointer to full-speed device qualifier. */
    NULL,                           /* Pointer to high-speed device qualifier. Not supported */
};
```

The USB Device Layer Master Descriptor table can be placed in the data or program memory of the PIC32 device. The contents of this table could be modified while the application is running. Doing this will affect the operation of the Device Stack. A typical USB device application will not need to change the contents of this table while the application is running.

Function Driver Registration Table

This section explains how function drivers can be registered with the USB Device Layer using the Function Registration Table.

Description

The Function Driver Registration Table (defined by the USB_DEVICE_FUNCTION_REGISTRATION_TABLE data type) contains information about the function drivers that are present in the application. The Device Layer needs this information to establish the intended functionality of the USB Device and then manage the operation of the device.

The Function Driver Registration Table contains an entry for every function driver instance contained in the application. Each entry is configuration specific. Hence in a case of device that features multiple configurations, the Function Driver Registration Table will contains an entry for every function driver in each configuration. Entries are instance and configuration specific. Hence if a configuration contains two instances of the same

function driver type, the Function Driver Registration Table will contain two entries to for the same function driver but with different instance indexes. A description of each member of the Function Driver Registration Table entry is as follows:

- The configurationValue member of the entry specifies to which configuration this entry belongs. The Device Layer will process this entry when the configurationValue configuration is set.
- The driver member of the entry should be set to Function Driver – Device Layer Interface Functions Object exported by the function driver. This object is provided by the function driver. In case of the CDC function driver, this is USB_DEVICE_CDC_FUNCTION_DRIVER. In case of HID function driver, this is USB_DEVICE_HID_FUNCTION_DRIVER. . Refer to the "Library Initialization" topic in Function Driver Specific help section for more details.
- The funcDriverIndex member of the entry specifies the instance of the function driver that this entry relates to. The Device Layer will use this instance when communicating with the function driver. In a case where there are multiple instances of the same function driver in a configuration, the funcDriverIndex allows the Device Layer to uniquely identify the function driver.
- The funcDriverInit member of the entry must point to the function driver instance specific initialization data structure. Function Drivers typically require an initialization data structure to be specified. The Device Layer passes the pointer to the initialization data structure when the function driver is initialized. Refer to the "Library Initialization" topic in Function Driver Specific help section for more details.
- The interfaceNumber member of the entry must contain the interface number of the first interface that is owned by this function driver instance. The information is available from the Device Configuration Descriptor.
- The numberOfInterfaces member of the entry must contain the number of interfaces following the interfaceNumber interface that is owned by this function driver instance. For example, a CDC Device requires two interfaces. The interfaceNumber member of Function Driver Registration Table entry for this function driver would be 0 and the numberOfInterfaces member would be 2. This indicates that Interface 0 and Interface 1 in the Device Configuration Descriptor are owned by this function driver.
- The speed member of the entry specifies the device speeds for which this function driver should be initialized. This can be set to either USB_SPEED_FULL, USB_SPEED_HIGH or a logical OR combination of both. The Device Layer will initialize the function if the device attach speed matches the speed mention in the speed member of the entry.

The following code shows an example of Function Driver Registration Table for one function driver. The CDC Function Driver in this case has two interfaces.

```
/*********************************************
 * USB Device Layer Function Driver Registration
 * Table
 *********************************************/
const USB_DEVICE_FUNCTION_REGISTRATION_TABLE funcRegistrationTable[1] =
{
    {
         .configurationValue = 1 ,                  // Configuration descriptor index
         .driver = USB_DEVICE_CDC_FUNCTION_DRIVER,  // CDC APIs exposed to the device layer
         .funcDriverIndex = 0 ,                     // Instance index of CDC function driver
         .funcDriverInit = (void *)&cdcInit,        // CDC init data
         .interfaceNumber = 0 ,                     // Start interface number of this instance
         .numberOfInterfaces = 2 ,                  // Total number of interfaces contained in this instance
         .speed = USB_SPEED_FULL|USB_SPEED_HIGH     // USB Speed
    }
};
```

The following code shows an example of Function Driver Registration Table for two function drivers. This example demonstrates a Composite (CDC + MSD) device. The CDC Function Driver uses two interfaces starting from interface 0. The MSD Function Driver has one interface starting from interface 2.

```
/*********************************************
 * Function Driver Registration Table
 *********************************************/
USB_DEVICE_FUNCTION_REGISTRATION_TABLE funcRegistrationTable[2] =
{
    {
        .speed = USB_SPEED_FULL|USB_SPEED_HIGH,     // Speed at which this device can operate
        .configurationValue = 1,                    // Configuration number to which this device belongs
        .interfaceNumber = 1,                       // Starting interface number for this function driver
        .numberOfInterfaces = 2,                    // Number of interfaces that this function driver owns.
        .funcDriverIndex = 0,                       // Function Driver index
        .funcDriverInit = &cdcInit,                 // Function Driver initialization data structure
        .driver = USB_DEVICE_CDC_FUNCTION_DRIVER    // CDC Function Driver.
    },
    {
        .speed = USB_SPEED_FULL|USB_SPEED_HIGH,     // Speed at which this device can operate
        .configurationValue = 1,                    // Configuration number to which this device belongs
        .interfaceNumber = 0,                       // Starting interface number for this function driver
        .numberOfInterfaces = 1,                    // Number of interfaces that this function driver owns.
        .funcDriverIndex = 0,                       // Function Driver index
        .funcDriverInit = &msdInit,                 // Function Driver initialization data structure
        .driver = USB_DEVICE_MSD_FUNCTION_DRIVER    // MSD Function Driver.
    },
};
```

The USB Device Layer Function Driver registration table can be placed in the data or program memory of the PIC32 device. The contents of this table could be modified while the application is running. Doing this will affect the operation of the device stack. A typical USB device application will not need to change the contents of this table while the application is running.

Initializing the Device Layer

This section describes the USB Device Layer initialization.

Description

With the USB Device Master Descriptor and the Function Driver Registration Table available, the application can now create the Device Layer Initialization Data structure. This data structure is a USB_DEVICE_INIT type and contains the information need to initialize the Device Layer and the USB Controller Driver. The Device Layer uses the USB Controller Driver initialization data to initialize the USB Controller driver. The actual initialization is performed by calling the USB_DEVICE_Initialize function. This function returns a Device Layer System Module Object which must be used with the other Device Layer System Routines (such as the USB_DEVICE_Tasks and USB_DEVICE_Tasks_ISR functions). In case of PIC32MX device, the Device Layer requires the allocation of an endpoint table. This table (an array of type uint8_t) should aligned at a 512 byte aligned address boundary. Its size should be USB_DEVICE_ENDPOINT_TABLE_SIZE. The table is not required for PIC32MZ devices. The value of USB_DEVICE_ENDPOINT_TABLE_SIZE while designing for PIC32MZ device will be automatically set to '0'.

The following code shows an example of initializing the Device Layer.

```
/*************************************************
 * Endpoint Table needed by the Device Layer.
 *************************************************/

uint8_t __attribute__((aligned(512))) endpointTable[USB_DEVICE_ENDPOINT_TABLE_SIZE];

/*********************************************
 * USB Device Layer Initialization.
 *********************************************/

USB_DEVICE_INIT usbDevInitData =
{
    /* System module initialization */
    .moduleInit = {SYS_MODULE_POWER_RUN_FULL},

     /* Identifies peripheral (PLIB-level) ID */
    .usbID = USB_ID_1,

    /* Stop in idle */
    .stopInIdle = false,

    /* Stop in sleep */
    .suspendInSleep = false,

    /* Interrupt source */
    .interruptSource = INT_SOURCE_USB_1,

    /* Endpoint Table */
    .endpointTable = endpointTable,

    /* Number of function drivers registered to this instance of the
    USB device layer */
    .registeredFuncCount = 2,

    /* Function driver table registered to this instance of the USB device layer*/
    .registeredFunctions = (USB_DEVICE_FUNCTION_REGISTRATION_TABLE*)funcRegistrationTable,

    /* Pointer to USB Descriptor structure */
    .usbMasterDescriptor = (USB_DEVICE_MASTER_DESCRIPTOR*)&usbMasterDescriptor
};

/*************************************
 * System Initialization Routine
 *************************************/
void SYS_Initialize ( void * data )
{
     /* Set up cache and wait states for maximum performance. */
    SYS_DEVCON_Initialize(SYS_DEVCON_INDEX_0, (SYS_MODULE_INIT *)&devconInit);
    SYS_DEVCON_PerformanceConfig(80000000);

    /* Initialize the BSP */
```

```
    BSP_Initialize( );

     /* Initialize the USB device layer */
    sysObjects.usbDevObject = USB_DEVICE_Initialize (USB_DEVICE_INDEX_0 ,
                                                     ( SYS_MODULE_INIT* ) & usbDevInitData);

    /* check if the object returned by the device layer is valid */
    SYS_ASSERT((SYS_MODULE_OBJ_INVALID != sysObjects.usbDevObject), "Invalid USB DEVICE object");

    /* Initialize the Application */
    APP_Initialize ( );

    /* Initialize the interrupt system  */
    SYS_INT_Initialize();

       /* set priority for USB interrupt source */
    SYS_INT_VectorPrioritySet(INT_VECTOR_USB, INT_PRIORITY_LEVEL3);

    /* set sub-priority for USB interrupt source */
    SYS_INT_VectorSubprioritySet(INT_VECTOR_USB, INT_SUBPRIORITY_LEVEL3);

    /* Initialize the global interrupts */
    SYS_INT_Enable();
}
```

Device Layer Task Routines

Describes the Device Layer task routines.

Description

In a case where the USB Controller driver is configured for interrupt mode operation (DRV_USB_INTERRUPT_MODE is true), the USB_DEVICE_Tasks_ISR() function should be called in the USB module interrupt associated with the Device Layer. The following code shows an example of this.

```
/*********************************************
 * Example of instantiating the USB Interrupt
 * and call the Device Layer ISR Tasks Routines.
 *********************************************/

void __ISR ( _USB_1_VECTOR, ipl3 ) _InterruptHandler_USB_stub ( void )
{
    USB_DEVICE_Tasks_ISR(sysObjects.usbDevObject);
}
```

If the USB Controller driver is configured for polled mode, the Device Layer will automatically invoke the USB_DEVICE_Tasks_ISR() functions from the USB_DEVICE_Tasks() function. Irrespective of whether the USB Controller driver is configured for polled or interrupt mode, the USB_DEVICE_Tasks() function should be called from SYS_Tasks() function. This will ensure that this function is called periodically. The USB_DEVICE_Tasks() function in turn calls the tasks routines of the applicable functions drivers and the USB Controller Driver (when the driver is configured for polled mode). The following code shows an example of how the USB_DEVICE_Tasks() function is called in the SYS_Tasks() function.

```
void SYS_Tasks ( void )
{
    /* Device layer tasks routine. Function Driver tasks gets called
     * from device layer tasks */

    USB_DEVICE_Tasks(sysObjects.usbDevObject);

    /* Call the application's tasks routine */
    APP_Tasks ();
}
```

Application Client Interaction

Describes application client interaction.

Description

Once initialized, Device Layer becomes ready for operation. The application must open the Device Layer by calling the USB_DEVICE_Open() function. Opening the Device Layer makes the application a Device Layer client. The Device Layer returns a valid Device Layer Handle when opened successfully. It will return an invalid Device Layer Handle when the open function fails. The application in this case should try opening the Device Layer again. The application can use a valid Device Layer handle to access the Device Layer functionality.

The client must now register a Device Layer Event Handler with the Device Layer. This is a mandatory step that enables USB Device Layer Events. The application must use the USB_DEVICE_EventHandlerSet function to register the event handler. The Application Event Handler should be of the type USB_DEVICE_EVENT_HANDLER. The Device Layer, when an event needs to be generated, calls this event handler function with the event type and event relevant information. The application must register an event handler for proper functioning of the USB Device. Not registering an event handler may cause the USB Device to malfunction and become non-compliant.

With an event handler register, the client can now attach the USB Device on the bus. The application must attach the in response to the USB_DEVICE_EVENT_POWER_DETECTED event. Attaching the device on the bus makes the device visible to the host (if it is already attached to the bus) and will cause the host to interact with the device.

The following code shows an example of the application opening the Device Layer, registering the event handler and then attaching the device on the bus.

```
/*************************************************
 * Here the application tries to open the Device Layer
 * and then register an event handler and then attach
 * the device on the bus.
 *************************************************/
void APP_Tasks(void)
{
    switch(appData.state)
    {
        case APP_STATE_INIT:

            /* Open the device layer */
            appData.deviceHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0,
                    DRV_IO_INTENT_READWRITE );

            if(appData.deviceHandle != USB_DEVICE_HANDLE_INVALID)
            {
                /* Register a callback with device layer to get event notification */
                USB_DEVICE_EventHandlerSet(appData.deviceHandle,
                        APP_USBDeviceEventCallBack, 0);

                appData.state = APP_STATE_WAIT_FOR_CONFIGURATION;
            }
            else
            {
                /* The Device Layer is not ready to be opened. We should try
                 * again later. */
            }

            break;
    }
}
```

Event Handling

Describes USB Device Layer event handling.

Description

The Device Layer generates events to let the application client know about the state of the bus. Some of these events require the application to respond in a specific manner. Not doing so, could cause the USB device to malfunction and become non-compliant. Code inside the event handler executes in an interrupt context when the Device Layer is configured for interrupt mode operation. The application must avoid calling computationally intensive functions or blocking functions in the event handler.

The following table shows a summary of the events that the Device Layer generates and the required Application client response.

Event	Required Application Response
USB_DEVICE_EVENT_POWER_DETECTED	Attach the device.
USB_DEVICE_EVENT_POWER_REMOVED	Detach the device.
USB_DEVICE_EVENT_RESET	No response required.
USB_DEVICE_EVENT_SUSPENDED	No response required.
USB_DEVICE_EVENT_RESUMED	No response required.
USB_DEVICE_EVENT_ERROR	The application can try detaching the device and reattaching. This should be done after exiting from the event handler.
USB_DEVICE_EVENT_SOF	No response required.

USB_DEVICE_EVENT_CONFIGURED	No response required.
USB_DEVICE_EVENT_DECONFIGURED	No response required.
USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST	Application must either respond with a USB_DEVICE_ControlSend() to send data, USB_DEVICE_ControlReceive() to receive data or stall or acknowledge the control request by calling the USB_DEVICE_ControlStatus() function.
USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT	No response required.
USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	Application must either or stall or acknowledge the control request by calling the USB_DEVICE_ControlStatus() function.
USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_ABORTED	No response required.

The Device Layer generates events with event relevant data. The pData parameter in the event handler functions points to this event specific data. The application can access this data by type casting the pData parameter of the event handler to a event specific data type. The following table shows a summary of the USB Device Layer events and the event data generated along with the event.

Event	Related pData Type
USB_DEVICE_EVENT_POWER_DETECTED	NULL
USB_DEVICE_EVENT_POWER_REMOVED	NULL
USB_DEVICE_EVENT_RESET	NULL
USB_DEVICE_EVENT_SUSPENDED	NULL
USB_DEVICE_EVENT_RESUMED	NULL
USB_DEVICE_EVENT_ERROR	NULL
USB_DEVICE_EVENT_SOF	USB_DEVICE_EVENT_DATA_SOF *
USB_DEVICE_EVENT_CONFIGURED	USB_DEVICE_EVENT_DATA_CONFIGURED *
USB_DEVICE_EVENT_DECONFIGURED	NULL
USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST	USB_SETUP_PACKET *
USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT	NULL
USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	NULL

A detailed description of each Device Layer event along with the required application client response, the likely follow-up event (if applicable) and the event specific data is provided here.

USB_DEVICE_EVENT_POWER_DETECTED

Application Response: This event indicates that the device has detected a valid VBUS to the host. The device is yet to be enumerated and configured. The application should not access the function drivers at this point. The application can use this event to attach the device on the bus.

Event Specific Data(pData): The pData parameter will be NULL.

Likely Follow Up Event: None.

USB_DEVICE_EVENT_POWER_REMOVED

Application Response: This event is an indication to the application client that the device is detached from the host. The application can use this event to detach the device.

Event Specific Data(pData): The pData parameter will be NULL

Likely Follow Up Event: None.

USB_DEVICE_EVENT_SUSPENDED

Application Response: This event is an indication to the application client that device is suspended and it can put the device to sleep mode if required. Power saving routines should not be called in the event handler.

Event Specific Data: The pData parameter will be NULL.

Likely Follow Up Event: None.

USB_DEVICE_EVENT_RESET

Application Response: USB bus reset occurred. This event is an indication to the application client that device layer has deinitialized all function drivers. The application should not use the function drivers in this state.

Event Specific Data: The pData parameter will be NULL.

Likely Follow Up Event: None.

USB_DEVICE_EVENT_RESUMED

Application Response: This event indicates that device has resumed from suspended state. The application can use this event to resume the operational state of the device.

Event Specific Data: The pData parameter will be NULL.

Likely Follow Up Event: None.

USB_DEVICE_EVENT_ERROR

Application Response: This event is an indication to the application client that an error occurred on the USB bus. The application can try detaching and reattaching the device.

Event Specific Data: The pData parameter will be NULL.

Likely Follow Up Event: None.

USB_DEVICE_EVENT_SOF

Application Response: This event occurs when the device receives a Start Of Frame packet. The application can use this event for synchronizing purposes. This event will be received every 1 millisecond for Full Speed USB and every one 125 micro seconds for High Speed USB. No application response is required.

Event Specific Data: Will point to USB_DEVICE_EVENT_DATA_SOF data type containing the frame number

Likely Follow Up Event: None.

USB_DEVICE_CONFIGURED

Application Response: This event is an indication to the application client that device layer has initialized all function drivers. The application can check the configuration set by the host. The application should use the event to register event handlers with the function drivers that are provisioned in the system.

Event Specific Data: The pData parameter will point to a USB_DEVICE_EVENT_DATA_CONFIGURED data type that contains configuration set by the host

Likely Follow Up Event: None.

USB_DEVICE_DECONFIGURED

Application Response: The host has deconfigured the device. This happens when the host sends a Set Configuration request with configuration number 0. The device layer will deinitialize all function drivers and then generate this event. No application response is required.

Event Specific Data:: The pData parameter will be NULL

Likely Follow Up Event: None.

USB_DEVICE_CONTROL_TRANSFER_ABORTED

Application Response: An on-going control transfer was aborted. The application can use this event to reset it's control transfer state machine.

Event Specific Data: The pData parameter will be NULL

Likely Follow Up Event: None.

USB_DEVICE_CONTROL_TRANSFER_DATA_RECEIVED

Application Response: The data stage of a Control write transfer has completed. This event occurs after the application has used the USB_DEVICE_ControlReceive() function to receive data in the control transfer (in response to the USB_DEVICE_CONTROL_TRANSFER_SETUP_REQUEST event) . The application can inspect the received data and stall or acknowledge the control transfer by calling the USB_DEVICE_ControlStatus() function with the USB_DEVICE_CONTROL_STATUS_ERROR flag or USB_DEVICE_CONTROL_STATUS_OK flag respectively. The application can call the USB_DEVICE_ControlStatus() function in the event handler or after exiting the event handler.

Event Specific Data: The pData parameter will be NULL

Likely Follow Up Event: None.

USB_DEVICE_CONTROL_TRANSFER_SETUP_REQUEST

Application Response: A setup packet of a control transfer has been received. The recipient field of the received setup packet is Other. The application can initiate the data stage by using the USB_DEVICE_ControlReceive() and USB_DEVICE_ControlSend() functions. It can end the control transfer by calling the USB_DEVICE_ControlStatus() function.

Event Specific Data: The pData parameter in the event handler will point to USB_SETUP_PACKET data type.

Likely Follow Up Event: USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT if the USB_DEVICE_ControlSend() function was called to send data to the host. USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED if the USB_DEVICE_ControlReceive() function was called to receive data from the host.

USB_DEVICE_CONTROL_TRANSFER_DATA_SENT

Application Response: The data stage of a Control Read transfer has completed. This event occurs after the application has used the USB_DEVICE_ControlSend() function to send data in the control transfer. No application response is required.

Event Specific Data: The pData parameter will be NULL

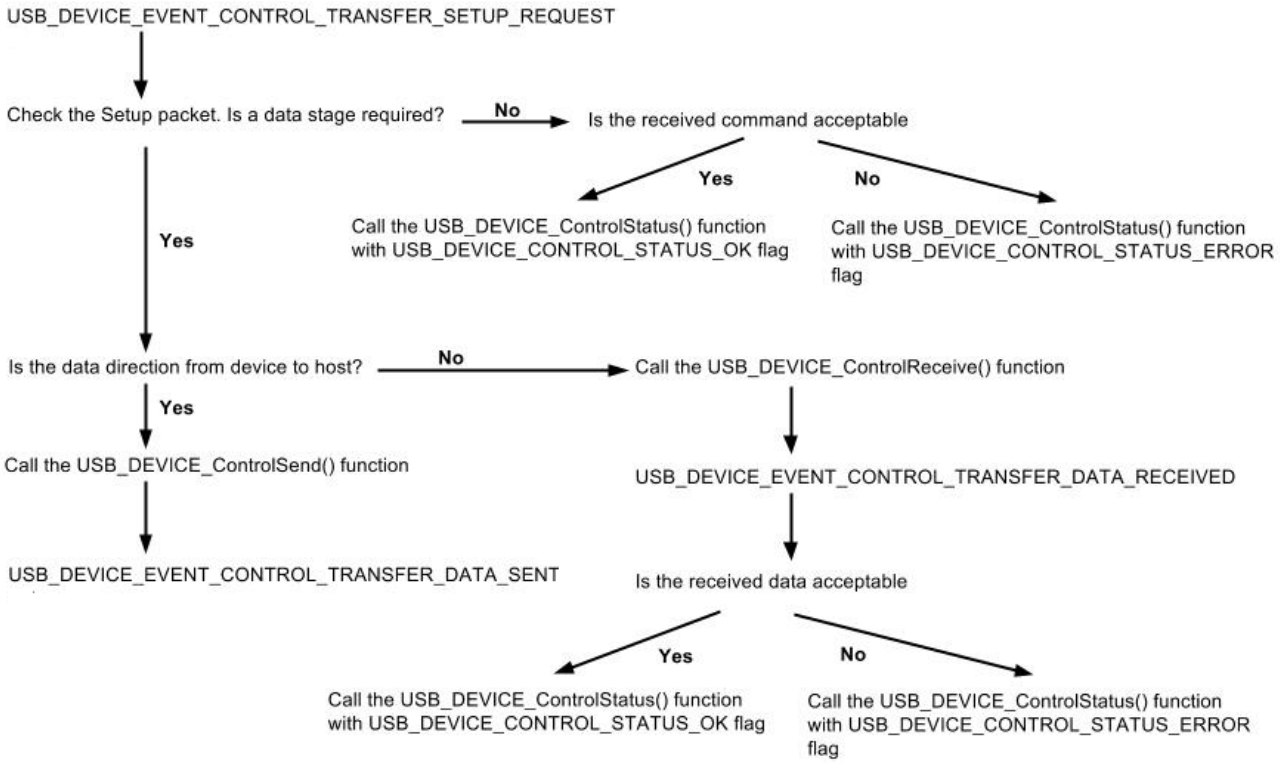
Likely Follow Up Event: None.

Device Layer Control Transfers

Describes USB Device Layer control transfers.

Description

The USB Device Layer forwards control transfer setup packets, where the Recipient field in the Setup packet is "Other", to the application for handling. The application must respond appropriately to this event. The following flow chart shows the possible sequences of events and application responses.



The Device Layer provides the USB_DEVICE_ControlReceive(), USB_DEVICE_ControlSend() and USB_DEVICE_ControlStatus() functions to complete the control transfers. These functions should be called only in response to the USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST event. In response to this event, the application can use the USB_DEVICE_ControlReceive( ) function to receive data in the data stage of a Control Write transfer. The reception of data is indicated by the USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event. The application can then complete the Control Write transfer by either:

- accepting the received data and acknowledging the Status Stage of the Control transfer. This is done by calling the USB_DEVICE_ControlStatus() function with the USB_DEVICE_CONTROL_STATUS_OK flag.
- rejecting the received data and stalling the Status Stage of the Control transfer. This is done by calling the USB_DEVICE_ControlStatus() function with the USB_DEVICE_CONTROL_STATUS_ERROR flag.

The application can use the USB_DEVICE_ControlSend() function to send data in the data stage of a Control Read transfer. The transmission of data is indicated by the USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT event.

In a case where the Control Transfer does not contain a data stage or if the application does not support the Setup Request, the application can end the Control Transfer by calling the USB_DEVICE_ControlStatus() function in response to the USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST event. Here the application can

- accepting the command by acknowledging the Status Stage of the Zero Data Stage of the Control transfer. This is done by calling the USB_DEVICE_ControlStatus() function with the USB_DEVICE_CONTROL_STATUS_OK flag.
- rejecting the Setup Request and stalling the Status Stage of the Control transfer. This is done by calling the USB_DEVICE_ControlStatus() function with the USB_DEVICE_CONTROL_STATUS_ERROR flag.

The application can also defer the response to Control transfer events. In that, the application does not have to respond to Control Transfer Events in the event handler. This may be needed in cases where resources required to respond to the Control Transfer Events are not readily available. The application, even while deferring the response, must however complete the Control Transfer in a time fashion. Failing to do so, will cause the host to cancel and retry the control transfer. This could also cause the USB device to malfunction and become non-compliant.

The following code shows an example of handling Device Layer events.

```
USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler
(
    USB_DEVICE_EVENT event,
    void * pData,
    uintptr_t context
)
{
    uint8_t     activeConfiguration;
    uint16_t    frameNumber;
    USB_SPEED   attachSpeed;
    USB_SETUP_PACKET * setupEventData;

    // Handling of each event
    switch(event)
    {
        case USB_DEVICE_EVENT_POWER_DETECTED:

            // This means the device detected a valid VBUS voltage
            // and is attached to the USB if the device is bus powered.
            break;

        case USB_DEVICE_EVENT_POWER_REMOVED:

            // This means the device is not attached to the USB.
            break;

        case USB_DEVICE_EVENT_SUSPENDED:

            // The bus is idle. There was no activity detected.
            // The application can switch to a low power mode after
            // exiting the event handler.
            break;

        case USB_DEVICE_EVENT_SOF:

            // A start of frame was received. This is a periodic
            // event and can be used the application for time
            // related activities. pData will point to a USB_DEVICE_EVENT_DATA_SOF type data
            // containing the frame number.

            frameNumber = ((USB_DEVICE_EVENT_DATA_SOF *)(pData))->frameNumber;
            break;

        case USB_DEVICE_EVENT_RESET :

            // Reset signalling was detected on the bus. The
            // application can find out the attach speed.

            attachedSpeed = USB_DEVICE_ActiveSpeedGet(usbDeviceHandle);
            break;

        case USB_DEVICE_EVENT_DECONFIGURED :

            // This indicates that host has deconfigured the device i.e., it
            // has set the configuration as 0. All function driver instances
            // would have been deinitialized.

            break;

        case USB_DEVICE_EVENT_ERROR :

            // This means an unknown error has occurred on the bus.
            // The application can try detaching and attaching the
            // device again.
            break;

        case USB_DEVICE_EVENT_CONFIGURED :

            // This means that device is configured and the application can
```

```
            // start using the device functionality. The application must
            // register function driver event handlers within this event.
            // The pData parameter will be a pointer to a USB_DEVICE_EVENT_DATA_CONFIGURED data type
            // that contains the active configuration number.

            activeConfiguration = ((USB_DEVICE_EVENT_DATA_CONFIGURED *)(pData))->configurationValue;
            break;

        case USB_DEVICE_EVENT_RESUMED:

            // This means that the resume signalling was detected on the
            // bus. The application can bring the device out of power
            // saving mode.

            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST:

            // This means that the setup stage of the control transfer is in
            // progress and a setup packet has been received. The pData
            // parameter will point to a USB_SETUP_PACKET data type The
            // application can process the command and update its control
            // transfer state machine. The application for example could call
            // the USB_DEVICE_ControlReceive() function (as shown here) to
            // submit the buffer that would receive data in case of a
            // control read transfer.

            setupPacket = (USB_SETUP_PACKET *)pData;

            // Submit a buffer to receive 32 bytes in the  control write transfer.
            USB_DEVICE_ControlReceive(usbDeviceHandle, data, 32);
            break;

        case USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA_RECEIVED:

            // This means that data in the data stage of the control write
            // transfer has been received. The application can either accept
            // the received data by calling the USB_DEVICE_ControlStatus()
            // function with USB_DEVICE_CONTROL_STATUS_OK flag (as shown in
            // this example) or it can reject it by calling the
            // USB_DEVICE_ControlStatus() function with
            // USB_DEVICE_CONTROL_STATUS_ERROR flag.

            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA_SENT:

            // This means that data in the data stage of the control
            // read transfer has been sent. The application would typically
            // end the control transfer by calling the
            // USB_DEVICE_ControlStatus() function with
            // USB_DEVICE_CONTROL_STATUS_OK flag (as shown in this example).

            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_CONTROL_TRANSFER_EVENT_ABORTED:

            // This means the host has aborted the control transfer. The
            // application can reset it's control transfer state machine.

            break;

        default:
            break;
    }

    return USB_DEVICE_EVENT_REPONSE_NONE;
```

```
    }
```

String Descriptor Table

Describes the String Descriptor Table o the device layer.

Description

The Device Layer allows the application to specify string descriptors via a String Descriptor Table. When the USB Host requests for a string by its index and language ID, the Device Layer looks for the corresponding string descriptor in the String Descriptor Table. There are two possible methods of specifying this String Descriptor Table, Basic and Advanced. These methods are discussed here.

Basic String Descriptor Table

The Basic String Descriptor Table should be used when the USB Device Application has equal number of string descriptors for each language string indexes exist without any intervals. This method is the default method of specifying the String Descriptor Table. Each entry in the table contains the following information

- The size of the entry
- The descriptor type, which is always set to USB_DESCRIPTOR_STRING
- The array containing the string

The first entry in the String Descriptor Table, at index 0 of the table, will always contain the Lang ID string. This string specifies the one language ID of the String Descriptor that this application intends to support. The subsequent entries in the String Descriptor Table contain the actual string descriptor. Each language must have an equal set of the string descriptors. The Device layer will associate each set of string descriptors with language ID specified in the language ID string descriptor. The following code shows an example of a Basic String Descriptor table.

Example:

```
/* This code shows an example of a Basic String Descriptor Table. In
 * this example, the table contains five entries. The first entry is the
 * language ID string. The second entry in the manufacturer string and the third
 * entry is the product string for language ID 0x0409. The fourth and the fifth
 * entry is the manufacture and product string, respectively for the language ID
 * 0x040C. */

/*******************************************************************
*  Language ID string descriptor. Note that this contains two Language IDs.
*******************************************************************/
const struct
{
    uint8_t bLength;
    uint8_t bDscType;
    uint16_t string[1];
}
sd000 =
{
    sizeof(sd000),          // Size of this descriptor in bytes
    USB_DESCRIPTOR_STRING,  // STRING descriptor type
    {0x0409, 0x040C}        // Language ID
};

/***************************************
 *  Manufacturer string descriptor
 ***************************************/
const struct
{
    uint8_t bLength;        // Size of this descriptor in bytes
    uint8_t bDscType;       // STRING descriptor type
    uint16_t string[25];    // String
}
sd001 =
{
    sizeof(sd001),
    USB_DESCRIPTOR_STRING,
    {'M','i','c','r','o','c','h','i','p',' ',
     'T','e','c','h','n','o','l','o','g','y',' ','I','n','c','.'}
};

/***************************************
 *  Product string descriptor
 ***************************************/
```

```
const struct
{
    uint8_t bLength;        // Size of this descriptor in bytes
    uint8_t bDscType;       // STRING descriptor type
    uint16_t string[22];    // String
}
sd002 =
{
    sizeof(sd002),
    USB_DESCRIPTOR_STRING,
    {'S','i','m','p','l','e',' ','C','D','C',' ','D','e','v','i','c','e',' ','D','e','m','o' }
};

/**************************************
 *  Manufacturer string descriptor
 **************************************/
const struct
{
    uint8_t bLength;        // Size of this descriptor in bytes
    uint8_t bDscType;       // STRING descriptor type
    uint16_t string[25];    // String
}
sd003 =
{
    sizeof(sd003),
    USB_DESCRIPTOR_STRING,
    {'M','i','c','r','o','c','h','i','p',' ',
     'T','e','c','h','n','o','l','o','g','y',' ','I','n','c','.'}
};

/**************************************
 *  Product string descriptor
 **************************************/
const struct
{
    uint8_t bLength;        // Size of this descriptor in bytes
    uint8_t bDscType;       // STRING descriptor type
    uint16_t string[22];    // String
}
sd004 =
{
    sizeof(sd004),
    USB_DESCRIPTOR_STRING,
    {'S','i','m','p','l','e',' ','C','D','C',' ','D','e','v','i','c','e',' ','D','e','m','o' }
};

/**************************************
 * Array of string descriptors
 **************************************/
USB_DEVICE_STRING_DESCRIPTORS_TABLE stringDescriptors[3]=
{
    /* This is the language ID string */
    (const uint8_t *const)&sd000,

    /* This string descriptor at index 1 will be returned when the host request
     * for a string descriptor with index 1 and language ID 0x0409. */
    (const uint8_t *const)&sd001,

    /* This string descriptor at index 2 will be returned when the host request
     * for a string descriptor with index 2 and language ID 0x0409. */
    (const uint8_t *const)&sd002,

    /* This string descriptor at index 3 will be returned when the host request
     * for a string descriptor with index 1 and language ID 0x040C. */
    (const uint8_t *const)&sd003,

    /* This string descriptor at index 4 will be returned when the host request
     * for a string descriptor with index 2 and language ID 0x040C. */
    (const uint8_t *const)&sd004
```

```
};
```

Advanced String Descriptor Table

The Advanced String Descriptor Table should be used when the application needs to specify string descriptors with string indexes that are not continuous. One such example is the Microsoft OS String Descriptor. The index of this string descriptor is 0xEE. If the application were to use the Basic String Descriptor Table , this would require the String Descriptor Table to have at least 0xED entries (valid or invalid) before the entry for the Microsoft OS String Descriptor. This may result in waste of RAM. Using the Ad Advanced String Descriptor Table mitigates this problem. The Advanced String Descriptor Table format is enabled only when USB_DEVICE_STRING_DESCRIPTOR_TABLE_ADVANCED_ENABLE configuration option is specified in the `system_config.h`. Each entry in the Advanced String Descriptor Table contains the following information:

- The index of the string descriptor
- The language ID of the string descriptor
- The size of the entry, which is two more than the length of the string
- The descriptor type, which is always set to USB_DESCRIPTOR_STRING
- The array containing the string

The first such entry in the Advanced String Descriptor Table specifies the language ID string. The string index and the language ID of this entry should be zero. This first entry is then followed by the actual string descriptors. Unlike the Basic String Descriptor Table, the index of an entry in the Advanced String Descriptor Table is not the same as the String Descriptor Index that the host uses to identify the string. In the Advanced String Descriptor Table, the index of the string is specified by the stringIndex member of the Advanced String Descriptor Table table entry. The following code shows an example of the Advanced String Descriptor table.

Example:

```
/* This code shows an example of an Advanced String Descriptor Table.
 * The Advanced String Descriptor table should be used when multiple languages
 * are needed to be supported. In this example, two languages are supported*/

/*******************************************************
 * Language ID string descriptor. Note that stringIndex and
 * language ID are always 0 for this descriptor.
 *******************************************************/
const struct __attribute__ ((packed))
{
    uint8_t stringIndex;    // Index of the string descriptor
    uint16_t languageID ;   // Language ID of this string.
    uint8_t bLength;        // Size of this descriptor in bytes
    uint8_t bDscType;       // STRING descriptor type
    uint16_t string[2];     // String
}
sd000 =
{
    0,                      // Index of this string is 0
    0,                      // This field is always blank for String Index 0
    sizeof(sd000)- 3,       // Should always be set to this.
    USB_DESCRIPTOR_STRING,
    {0x0409, 0x040C}        // Language ID
};

/*************************************************
 *  Manufacturer string descriptor for language 0x0409
 *************************************************/
const struct __attribute__ ((packed))
{
    uint8_t stringIndex;    // Index of the string descriptor
    uint16_t languageID ;   // Language ID of this string.
    uint8_t bLength;        // Size of this descriptor in bytes
    uint8_t bDscType;       // STRING descriptor type
    uint16_t string[25];    // String
}
sd001 =
{
    1,      // Index of this string descriptor is 1.
    0x0409, // Language ID of this string descriptor is 0x0409 (English)
    sizeof(sd001) - 3,
    USB_DESCRIPTOR_STRING,
    {'M','i','c','r','o','c','h','i','p',' ',
    'T','e','c','h','n','o','l','o','g','y',' ','I','n','c','.'}
};

/*************************************************
```

```
 *  Manufacturer string descriptor for language 0x040C
 ************************************************/
const struct __attribute__ ((packed))
{
    uint8_t stringIndex;    // Index of the string descriptor
    uint16_t languageID ;   // Language ID of this string.
    uint8_t bLength;        // Size of this descriptor in bytes
    uint8_t bDscType;       // STRING descriptor type
    uint16_t string[25];    // String
}
sd002 =
{
    1,      // Index of this string descriptor is 1.
    0x040C, // Language ID of this string descriptor is 0x040C (French)
    sizeof(sd001) - 3,
    USB_DESCRIPTOR_STRING,
    {'M','i','c','r','o','c','h','i','p',' ',
    'T','e','c','h','n','o','l','o','g','y',' ','I','n','c','.'}
};

/*********************************************
 *  Product string descriptor for language 0x409
 *********************************************/
const struct __attribute__ ((packed))
{
    uint8_t stringIndex;    // Index of the string descriptor
    uint16_t languageID ;   // Language ID of this string.
    uint8_t bLength;        // Size of this descriptor in bytes
    uint8_t bDscType;       // STRING descriptor type
    uint16_t string[22];    // String
}
sd003 =
{
    2,       // Index of this string descriptor is 2.
    0x0409,  // Language ID of this string descriptor is 0x0409 (English)
    sizeof(sd002) - 3,
    USB_DESCRIPTOR_STRING,
    {'S','i','m','p','l','e',' ','C','D','C',' ','D','e','v','i','c','e',' ','D','e','m','o' }
};

/*********************************************
 *  Product string descriptor for language 0x40C
 *********************************************/
const struct __attribute__ ((packed))
{
    uint8_t stringIndex;    // Index of the string descriptor
    uint16_t languageID ;   // Language ID of this string.
    uint8_t bLength;        // Size of this descriptor in bytes
    uint8_t bDscType;       // STRING descriptor type
    uint16_t string[22];    // String
}
sd004 =
{
    2,       // Index of this string descriptor is 2.
    0x0409,  // Language ID of this string descriptor is 0x040C (French)
    sizeof(sd002) - 3,
    USB_DESCRIPTOR_STRING,
    {'S','i','m','p','l','e',' ','C','D','C',' ','D','e','v','i','c','e',' ','D','e','m','o' }
};

/*************************************************************
 * Array of string descriptors. The entry order does not matter.
 *************************************************************/
USB_DEVICE_STRING_DESCRIPTORS_TABLE stringDescriptors[5]=
{
    (const uint8_t *const)&sd000,
    (const uint8_t *const)&sd001,   // Manufacturer string for language 0x0409
    (const uint8_t *const)&sd002,   // Manufacturer string for language 0x040C
    (const uint8_t *const)&sd003,   // Product string for language 0x0409
```

```
    (const uint8_t *const)&sd004,   // Product string for language 0x040C
};
```

BOS Descriptor Support

Provides information on the BOS descriptor.

Description

The USB 3.0 and the USB 2.0 LPM specifications define a new descriptor called the Binary Device Object Store (BOS) descriptor. This descriptor contains information of the capability of the device. When the bcdUSB value in the Device Descriptor is greater than 0x0200, the USB Host Operating System may request for the BOS descriptor.

The MPLAB Harmony USB Device Library allows the application to support the BOS descriptor requests. This support is enabled by adding the USB_DEVICE_BOS_DESCRIPTOR_SUPPORT_ENABLE configuration macro in `system_config.h`. The application must set the bosDescriptor member of the USB_DEVICE_INIT data structure (this data structure is passed in the USB_DEVICE_Initialize function) to point to the data to be returned in the data stage of the BOS descriptor request.

If the USB_DEVICE_BOS_DESCRIPTOR_SUPPORT_ENABLE configuration macro is not specified, the Device Layer will stall the Host request BOS descriptor.

Configuring the Library

Describes how to configure the USB Device Library.

Macros

	Name	Description
	USB_DEVICE_INSTANCES_NUMBER	Number of Device Layer instances to provisioned in the application.
	USB_DEVICE_ENDPOINT_QUEUE_DEPTH_COMBINED	Specifies the combined endpoint queue depth in case of a vendor USB device implementation.
	USB_DEVICE_SET_DESCRIPTOR_EVENT_ENABLE	Enables the Device Layer Set Descriptor Event.
	USB_DEVICE_SOF_EVENT_ENABLE	Enables the Device Layer SOF event.
	USB_DEVICE_SYNCH_FRAME_EVENT_ENABLE	Enables the Device Layer Synch Frame Event.
	USB_DEVICE_EP0_BUFFER_SIZE	Buffer Size in Bytes for Endpoint 0.
	USB_DEVICE_MICROSOFT_OS_DESCRIPTOR_SUPPORT_ENABLE	Specifies if the USB Device stack should support Microsoft OS Descriptor.

Description

The USB Device Layer initializes and configures the USB Controller Driver (the driver that manages the USB peripheral when operating as device) and maintains its task routine. For completeness, the following table lists the configuration macros that are needed by the USB Controller Driver. These macros should be defined in `system_config.h` file along with the Device Layer Configuration macros.

USB_DEVICE_INSTANCES_NUMBER Macro

Number of Device Layer instances to provisioned in the application.

File

usb_device_config_template.h

C

```
#define USB_DEVICE_INSTANCES_NUMBER 1
```

Description

Number of Device Layer Instances.

This configuration macro defines the number of Device Layer instances in the application. In cases of microcontrollers that feature multiple USB peripherals, this allows the application to run multiple instances of the Device Layer. As an example, for a microcontroller containing two USB peripherals, setting USB_DEVICE_INSTANCES_NUMBER to 2 will cause 2 instances of the Device Layer to execute.

Remarks

Setting this value to more than the number of USB peripheral will result if unused RAM consumption.

USB_DEVICE_ENDPOINT_QUEUE_DEPTH_COMBINED Macro

Specifies the combined endpoint queue depth in case of a vendor USB device implementation.

File

usb_device_config_template.h

C

```
#define USB_DEVICE_ENDPOINT_QUEUE_DEPTH_COMBINED 2
```

Description

USB Device Layer Combined Endpoint Queue Depth

This configuration constant specifies the combined endpoint queue depth in a case where the endpoint read and endpoint write functions are used to implement a vendor USB device. This constant should be used in conjunction with the usb_device_endpoint_functions.c file.

This macro defines the number of entries in all IN and OUT endpoint queues in all instances of the USB Device Layer. This value can be obtained by adding up the endpoint read and write queue sizes of each USB Device Layer instance . In a simple single instance USB Device Layer, that requires only one read and one write endpoint with one buffer each, the USB_DEVICE_ENDPOINT_QUEUE_DEPTH_COMBINED macro can be set to 2. Consider a case with one Device Layer instance using 2 IN and 2 OUT endpoints, each requiring 2 buffers, this macro should be set to 8 (2 + 2 + 2 + 2).

Remarks

This constant needs to be specified only if a Vendor USB Device is to be implemented and the usb_device_endpoint_functions.c file is included in the project. This constant does not have any effect on queues of other standard function drivers that are included in the USB device implementation.

USB_DEVICE_SET_DESCRIPTOR_EVENT_ENABLE Macro

Enables the Device Layer Set Descriptor Event.

File

usb_device_config_template.h

C

```
#define USB_DEVICE_SET_DESCRIPTOR_EVENT_ENABLE
```

Description

USB Device Layer Set Descriptor Event Enable

Specifying this configuration macro will cause the USB Device Layer to generate the USB_DEVICE_EVENT_SET_DESCRIPTOR event when a Set Descriptor request is received. If this macro is not defined, the USB Device Layer will stall the Set Descriptor control transfer request.

Remarks

None.

USB_DEVICE_SOF_EVENT_ENABLE Macro

Enables the Device Layer SOF event.

File

usb_device_config_template.h

C

```
#define USB_DEVICE_SOF_EVENT_ENABLE
```

Description

USB Device Layer SOF Event Enable

Specifying this configuration macro will cause the USB Device Layer to generate the USB_DEVICE_EVENT_SOF event. On Full Speed USB Devices, this event will be generated every 1 millisecond. On High Speed USB devices, this event will be generated every 125 microsecond.

Remarks

None.

USB_DEVICE_SYNCH_FRAME_EVENT_ENABLE Macro

Enables the Device Layer Synch Frame Event.

File

usb_device_config_template.h

C

```
#define USB_DEVICE_SYNCH_FRAME_EVENT_ENABLE
```

Description

USB Device Layer Synch Frame Event Enable

Specifying this configuration macro will cause the USB Device Layer to generate the USB_DEVICE_EVENT_SYNCH_FRAME event when a Synch Frame control transfer request is received. If this macro is not defined, the USB Device Layer will stall the control transfer request associated with this event.

Remarks

None.

USB_DEVICE_EP0_BUFFER_SIZE Macro

Buffer Size in Bytes for Endpoint 0.

File

usb_device_config_template.h

C

```
#define USB_DEVICE_EP0_BUFFER_SIZE
```

Description

Endpoint 0 Buffer Size

This number defines the size (in bytes) of Endpoint 0. For High Speed USB Devices, this number should be 64. For Full Speed USB Devices, this number can be 8, 16, 32 or 64 bytes. This number will be applicable to all USB Device Stack instances.

Remarks

None.

USB_DEVICE_MICROSOFT_OS_DESCRIPTOR_SUPPORT_ENABLE Macro

Specifies if the USB Device stack should support Microsoft OS Descriptor.

File

usb_device_config_template.h

C

```
#define USB_DEVICE_MICROSOFT_OS_DESCRIPTOR_SUPPORT_ENABLE
```

Description

USB Device Layer Microsoft OS Descriptor Support Enable

This macro needs to be defined to enable Microsoft OS descriptor support. If this macro is defined all Vendor Interface requests are forwarded to client unconditionally and Device layer does not validate the recipient interface field in a Control transfer.

Device and Class Control Requests are not affected by this configuration. The Device Layer will validate the recipient interface in Device and Class Control Requests, irrespective of this configuration constant, and will stall these requests if the interface is provisioned in the Function Driver Registration Table.

If this macro is not defined, then USB Device Layer will validate the interface number in a Vendor Interface request and stall the request if Interface number is not available in the Function registration table.

Remarks

None.

Building the Library

This section lists the files that are available in the USB Device Layer Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/usb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
usb_device.h	This header file should be included in any `.c` file that accesses the USB Device Layer API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/usb_device.c`	This file implements the USB Device Layer interface and should be included in project if USB Device mode operation is desired.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The USB Device Layer Library depends on the following modules:

- USB Driver Library (Device mode files)

Library Interface

a) System Interaction Functions

	Name	Description
	USB_DEVICE_Initialize	Creates and initializes an instance of the USB device layer.
	USB_DEVICE_Deinitialize	De-initializes the specified instance of the USB device layer.
	USB_DEVICE_Status	Provides the current status of the USB device layer
	USB_DEVICE_Tasks	USB Device layer calls all other function driver tasks in this function. It also generates and forwards events to its clients.
	USB_DEVICE_Tasks_ISR	USB Device Layer Tasks routine to be called in the USB Interrupt Service Routine.
	USB_DEVICE_Tasks_ISR_USBDMA	This is function USB_DEVICE_Tasks_ISR_USBDMA.

b) Client Core Functions

	Name	Description
	USB_DEVICE_Open	Opens the specified USB device layer instance and returns a handle to it.
	USB_DEVICE_Close	Closes an opened handle to an instance of the USB device layer.
	USB_DEVICE_ClientStatusGet	Returns the client specific status.
	USB_DEVICE_EventHandlerSet	USB Device Layer Event Handler Callback Function set function.

c) Device Power State Management Functions

	Name	Description
	USB_DEVICE_PowerStateSet	Sets power state of the device.
	USB_DEVICE_RemoteWakeupStatusGet	Gets the "Remote wake-up" status of the device.
	USB_DEVICE_IsSuspended	Returns true if the device is in a suspended state.
	USB_DEVICE_RemoteWakeupStart	This function will start the resume signaling.
	USB_DEVICE_RemoteWakeupStartTimed	This function will start a self timed Remote Wake-up.
	USB_DEVICE_RemoteWakeupStop	This function will stop the resume signaling.

d) Device Management Functions

	Name	Description
	USB_DEVICE_StateGet	Returns the current state of the USB device.
	USB_DEVICE_Attach	This function will attach the device to the USB.
	USB_DEVICE_Detach	This function will detach the device from the USB.
	USB_DEVICE_ActiveConfigurationGet	Informs the client of the current USB device configuration set by the USB host.
	USB_DEVICE_ActiveSpeedGet	Informs the client of the current operation speed of the USB bus.

e) Endpoint Management Functions

	Name	Description
	USB_DEVICE_EndpointIsStalled	This function returns the stall status of the specified endpoint and direction.
	USB_DEVICE_EndpointStall	This function stalls an endpoint in the specified direction.
	USB_DEVICE_EndpointStallClear	This function clears the stall on an endpoint in the specified direction.
	USB_DEVICE_EndpointDisable	Disables a device endpoint.
	USB_DEVICE_EndpointEnable	Enables a device endpoint.
	USB_DEVICE_EndpointIsEnabled	Returns true if the endpoint is enabled.
	USB_DEVICE_EndpointRead	Reads data received from host on the requested endpoint.
	USB_DEVICE_EndpointTransferCancel	This function cancels a transfer scheduled on an endpoint.
	USB_DEVICE_EndpointWrite	This function requests a data write to a USB Device Endpoint.

f) Control Transfer Functions

	Name	Description
	USB_DEVICE_ControlReceive	Receives data stage of the control transfer from host to device.
	USB_DEVICE_ControlSend	Sends data stage of the control transfer from device to host.
	USB_DEVICE_ControlStatus	Initiates status stage of the control transfer.

g) Data Types and Constants

	Name	Description
	USB_DEVICE_INDEX_0	USB device layer index definitions.
	USB_DEVICE_INDEX_1	This is macro USB_DEVICE_INDEX_1.
	USB_DEVICE_INDEX_2	This is macro USB_DEVICE_INDEX_2.
	USB_DEVICE_INDEX_3	This is macro USB_DEVICE_INDEX_3.
	USB_DEVICE_INDEX_4	This is macro USB_DEVICE_INDEX_4.
	USB_DEVICE_INDEX_5	This is macro USB_DEVICE_INDEX_5.
	USB_DEVICE_CONTROL_STATUS	USB Device Layer Control Transfer Status Stage flags.
	USB_DEVICE_CONTROL_TRANSFER_RESULT	Enumerated data type identifying results of a control transfer.
	USB_DEVICE_EVENT	USB Device Layer Events.
	USB_DEVICE_HANDLE	Data type for USB device handle.
	USB_DEVICE_INIT	USB Device Initialization Structure
	USB_DEVICE_POWER_STATE	Enumerated data type that identifies if the device is self powered or bus powered .
	USB_DEVICE_REMOTE_WAKEUP_STATUS	Enumerated data type that identifies if the remote wakeup status of the device.
	USB_DEVICE_EVENT_DATA_CONFIGURED	USB Device Set Configuration Event Data type.
	USB_DEVICE_HANDLE_INVALID	Constant that defines the value of an Invalid Device Handle.
	USB_DEVICE_EVENT_RESPONSE_NONE	Device Layer Event Handler Function Response Type.
	USB_DEVICE_CLIENT_STATUS	Enumerated data type that identifies the USB Device Layer Client Status.
	USB_DEVICE_CONFIGURATION_DESCRIPTORS_TABLE	Pointer to an array that contains pointer to configuration descriptors.
	USB_DEVICE_EVENT_HANDLER	USB Device Layer Event Handler Function Pointer Type
	USB_DEVICE_EVENT_RESPONSE	Device Layer Event Handler function return type.
	USB_DEVICE_FUNCTION_REGISTRATION_TABLE	USB Device Function Registration Structure
	USB_DEVICE_MASTER_DESCRIPTOR	USB Device Master Descriptor Structure.

	USB_DEVICE_STRING_DESCRIPTORS_TABLE	Pointer to an array that contains pointer to string descriptors.
	USB_DEVICE_EVENT_DATA_ENDPOINT_READ_COMPLETE	USB Device Layer Endpoint Read and Write Complete Event Data type.
	USB_DEVICE_EVENT_DATA_ENDPOINT_WRITE_COMPLETE	USB Device Layer Endpoint Read and Write Complete Event Data type.
	USB_DEVICE_EVENT_DATA_SET_DESCRIPTOR	USB Device Set Descriptor Event Data type.
	USB_DEVICE_EVENT_DATA_SOF	USB Device Start Of Frame Event Data Type
	USB_DEVICE_EVENT_DATA_SYNCH_FRAME	USB Device Synch Frame Event Data type.
	USB_DEVICE_RESULT	USB Device Layer Results Enumeration
	USB_DEVICE_TRANSFER_FLAGS	Enumerated data type that identifies the USB Device Layer Transfer Flags.
	USB_DEVICE_TRANSFER_HANDLE	Data type for USB Device Endpoint Data Transfer Handle.
	USB_DEVICE_TRANSFER_HANDLE_INVALID	Constant that defines the value of an Invalid Device Endpoint Data Transfer Handle.
	_USB_DEVICE_IRP	This structure defines the USB Device Mode IRP data structure.
	_USB_HOST_IRP	This structure defines the USB Host Mode IRP data structure.
	_USB_HOST_IRP_STATUS	Enumerates the possible status options of USB Host IRP.
	USB_DEVICE_IRP	This structure defines the USB Device Mode IRP data structure.
	USB_DEVICE_IRP_FLAG	USB Device IRP flags enumeration
	USB_DEVICE_IRP_STATUS	Enumerates the possible status options of USB Device IRP.
	USB_ENDPOINT	Defines a type to store Endpoint and Direction. The MSB defines the direction. The lower 4 bits defines the endpoint.
	USB_ERROR	Enumeration of all possible error codes that are returned by various components functions in the USB Stack.
	USB_HOST_IRP	This structure defines the USB Host Mode IRP data structure.
	USB_HOST_IRP_FLAG	USB Host IRP flags enumeration
	USB_HOST_IRP_STATUS	Enumerates the possible status options of USB Host IRP.
	USB_SPEED	Provides enumeration of USB 2.0 speeds.
	USB_ENDPOINT_AND_DIRECTION	This macro helps in setting up the USB_ENDPOINT type.
	USB_DATA_DIRECTION	Defines the communication direction
	USB_DEVICE_BOS_DESCRIPTOR_SUPPORT_ENABLE	Specifies if the Device Layer should process a Host request for a BOS descriptor.
	USB_DEVICE_DRIVER_INITIALIZE_EXPLICIT	Specifies if the USB Controller Driver must be initialized explicitly as opposed to being initialized by the Device Layer.
	USB_DEVICE_STRING_DESCRIPTOR_TABLE_ADVANCED_ENABLE	Specifying this macro enables the Advanced String Descriptor Table Entry Format.

Description

This section describes the Application Programming Interface (API) functions of the USB Device Layer Library.

Refer to each section for a detailed description.

a) System Interaction Functions

USB_DEVICE_Initialize Function

Creates and initializes an instance of the USB device layer.

File

usb_device.h

C

```
SYS_MODULE_OBJ USB_DEVICE_Initialize(const SYS_MODULE_INDEX instanceIndex, const SYS_MODULE_INIT * const
init);
```

Returns

If successful, returns a valid handle to a device layer object. Otherwise, it returns SYS_MODULE_OBJ_INVALID.

Description

This function initializes an instance of USB device layer, making it ready for clients to open and use it. The number of instances is limited by the value of macro USB_DEVICE_MAX_INSTANCES defined in system_config.h file.

Remarks

This routine must be called before any other USB Device Layer routine is called and after the initialization of USB Device Driver. This routine should only be called once during system initialization.

Preconditions

None.

Example

```
// This code example shows the initialization of the
// the USB Device Layer. Note how an endpoint table is
// created and assigned.

USB_DEVICE_INIT deviceLayerInit;
SYS_MODULE_OBJ usbDeviceObj;
uint8_t __attribute__((aligned(512))) endpointTable[USB_DEVICE_ENDPOINT_TABLE_SIZE];

// System module initialization
deviceLayerInit.moduleInit.value = SYS_MODULE_POWER_RUN_FULL;

// Identifies peripheral (PLIB-level) ID
deviceLayerInit.usbID = USB_ID_1;

// Boolean flag: true -> Stop USB module in Idle Mode
deviceLayerInit.stopInIdle = false;

// Boolean flag: true -> Suspend USB in Sleep Mode
deviceLayerInit.suspendInSleep = false;

// Interrupt Source for USB module
deviceLayerInit.interruptSource = INT_SOURCE_USB_1;

// Number of function drivers registered to this instance of the
// USB device layer
deviceLayerInit.registeredFuncCount = 1;

// Function driver table registered to this instance of the USB device layer
deviceLayerInit.registeredFunctions = funcRegistrationTable;

// Pointer to USB Descriptor structure
deviceLayerInit.usbMasterDescriptor   = &usbMasterDescriptor;

// Pointer to an endpoint table.
deviceLayerInit.endpointTable = endpointTable;

// USB device initialization
usbDeviceObj = USB_DEVICE_Initialize(USB_DEVICE_INDEX_0, &deviceLayerInit);

if (SYS_MODULE_OBJ_INVALID == usbDeviceObj)
{
    // Handle error
}
```

Parameters

Parameters	Description
instanceIndex	In case of microcontrollers with multiple USB peripherals, user can create multiple instances of USB device layer. Parameter instanceIndex identifies this instance.
init	Pointer to a data structure containing any data necessary to initialize the USB device layer

Function

SYS_MODULE_OBJ USB_DEVICE_Initialize
(
const SYS_MODULE_INDEX instanceIndex,
const SYS_MODULE_INIT * const init
)

USB_DEVICE_Deinitialize Function

De-initializes the specified instance of the USB device layer.

File

usb_device.h

C

```
void USB_DEVICE_Deinitialize(SYS_MODULE_OBJ usbDeviceObj);
```

Returns

None.

Description

This function deinitializes the specified instance of the USB device layer, disabling its operation (and any hardware) and invalidates all of the internal data.

Remarks

Once the Initialize operation has been called, the deinitialize operation must be called before the Initialize operation can be called again.

Preconditions

Function USB_DEVICE_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
// This code example shows how the USB
// Device Layer can be deinitialized. It is assumed the
// USB Device Layer was already initialized.

SYS_MODULE_OBJ usbDeviceobj;

USB_DEVICE_Deinitialize(usbDeviceobj);
```

Parameters

Parameters	Description
object	USB device layer object handle, returned by USB_DEVICE_Initialize

Function

void USB_DEVICE_Deinitialize (SYS_MODULE_OBJ usbDeviceobj)

USB_DEVICE_Status Function

Provides the current status of the USB device layer

File

usb_device.h

C

```
SYS_STATUS USB_DEVICE_Status(SYS_MODULE_OBJ object);
```

Returns

SYS_STATUS_READY - Indicates that the device is busy with a previous system level operation and cannot start another

SYS_STATUS_UNINITIALIZED - Indicates that the device layer is in a deinitialized state

Description

This function provides the current status of the USB device layer.

Remarks

None.

Preconditions

The USB_DEVICE_Initialize function must have been called before calling this function.

Example

```
// This code example shows how the USB_DEVICE_Status function
// can be used to check if the USB Device Layer is ready
// for client operations.

SYS_MODULE_OBJ      object;     // Returned from DRV_USB_Initialize
SYS_STATUS          status;

status = USB_DEVICE_Status(object);

if (SYS_STATUS_READY != status)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Driver object handle, returned from USB_DEVICE_Initialize

Function

SYS_STATUS USB_DEVICE_Status (SYS_MODULE_OBJ object)

USB_DEVICE_Tasks Function

USB Device layer calls all other function driver tasks in this function. It also generates and forwards events to its clients.

File

usb_device.h

C

```
void USB_DEVICE_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function must be periodically called by the user application. The USB Device layer calls all other function driver tasks in this function. It also generates and forwards events to its clients.

Remarks

None.

Preconditions

Device layer must have been initialized by calling USB_DEVICE_Initialize.

Example

```
// The USB_DEVICE_Tasks() function should be placed in the
// SYS_Tasks() function of a MPLAB Harmony application.

SYS_MODULE_OBJ usbDeviceLayerObj; // Returned by USB_DEVICE_Initialize().
```

```
void SYS_Tasks(void)
{
    USB_DEVICE_Tasks(usbDeviceLayerObj);
}
```

Parameters

Parameters	Description
devLayerObj	Pointer to the Device Layer Object that is returned from USB_DEVICE_Initialize

Function

void USB_DEVICE_Tasks(SYS_MODULE_OBJ devLayerObj)

USB_DEVICE_Tasks_ISR Function

USB Device Layer Tasks routine to be called in the USB Interrupt Service Routine.

File

usb_device.h

C

```
void USB_DEVICE_Tasks_ISR(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This function must be called in the USB Interrupt Service Routine if the Device Stack is configured for interrupt mode. In case the Device Stack is configured for polling mode, this function is automatically called from the USB_DEVICE_Tasks function. devLayerObj must be the system module object associated with the USB module generating the interrupt.

Remarks

None.

Preconditions

Device layer must have been initialized.

Example

```
// devLayerObj is returned while initializing the USB1 module.
// The USB_DEVICE_Tasks_ISR function should be placed in the
// USB1 module ISR.

SYS_MODULE_OBJ devLayerObj; // Returned by USB_DEVICE_Initialize().

void __ISR(_USB_1_VECTOR, ipl4) USB1InterruptHandle(void)
{
    USB_DEVICE_Tasks_ISR(usbDeviceLayerObj);
}
```

Parameters

Parameters	Description
devLayerObj	Pointer to the Device Layer Object that is returned from USB_DEVICE_Initialize

Function

void USB_DEVICE_Tasks_ISR(SYS_MODULE_OBJ devLayerObj)

USB_DEVICE_Tasks_ISR_USBDMA Function

File

usb_device.h

C

```
void USB_DEVICE_Tasks_ISR_USBDMA(SYS_MODULE_OBJ devLayerObj);
```

Description

This is function USB_DEVICE_Tasks_ISR_USBDMA.

b) Client Core Functions

USB_DEVICE_Open Function

Opens the specified USB device layer instance and returns a handle to it.

File

usb_device.h

C

```
USB_DEVICE_HANDLE USB_DEVICE_Open(const SYS_MODULE_INDEX instanceIndex, const DRV_IO_INTENT intent);
```

Returns

If successful, returns a valid device layer handle. Otherwise, it returns USB_DEVICE_HANDLE_INVALID.

Description

This function opens the USB device layer instance specified by instance index and returns a handle. This handle must be provided to all other client operations to identify the caller and the instance of the USB device layer. An instance of the Device Layer can be opened only once. Trying to open the Device Layer more than once will return a invalid device layer handle.

Remarks

None.

Preconditions

This function must be called after USB device driver initialization and after the initialization of USB Device Layer.

Example

```
// This code example shows how the
// USB Device Layer can be opened.

USB_DEVICE_HANDLE usbDeviceHandle;

// Before opening a handle, USB device must have been initialized
// by calling USB_DEVICE_Initialize().

usbDeviceHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0,
                                   DRV_IO_INTENT_READWRITE );

if(USB_DEVICE_HANDLE_INVALID == usbDeviceHandle)
{
    //Failed to open handle.
}
```

Parameters

Parameters	Description
instanceIndex	USB device layer instance index
intent	This parameter is ignored. The Device Layer will always open in read/write and exclusive mode.

Function

USB_DEVICE_HANDLE USB_DEVICE_Open

(

const SYS_MODULE_INDEX instanceIndex,

const DRV_IO_INTENT intent

)

USB_DEVICE_Close Function

Closes an opened handle to an instance of the USB device layer.

File

usb_device.h

C

```
void USB_DEVICE_Close(USB_DEVICE_HANDLE usbDeviceHandle);
```

Returns

None

Description

This function closes an opened handle to an instance of the USB device layer, invalidating the handle.

Remarks

After calling this routine, the handle passed in "usbDevHandle" must not be used with any of the remaining driver routines. A new handle must be obtained by calling USB_DEVICE_Open() before the client may use the device layer again.

Preconditions

The USB_DEVICE_Initialize function must have been called for the specified device layer instance. USB_DEVICE_Open must have been called to obtain a valid opened device handle.

Example

```
USB_DEVICE_HANDLE usbDeviceHandle;

// Before opening a handle, USB device must have been initialized
// by calling USB_DEVICE_Initialize().
usbDeviceHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0 );

if(USB_DEVICE_HANDLE_INVALID == usbDeviceHandle)
{
    //Failed to open handle.
}

//..................
//..................
// User's code
//..................
//..................
// Close handle
USB_DEVICE_Close( usbDevHandle );
```

Parameters

Parameters	Description
handle	A valid open-instance handle, returned from USB_DEVICE_Open

Function

void USB_DEVICE_Close(USB_DEVICE_HANDLE usbDeviceHandle)

USB_DEVICE_ClientStatusGet Function

Returns the client specific status.

File

usb_device.h

C

```
USB_DEVICE_CLIENT_STATUS USB_DEVICE_ClientStatusGet(USB_DEVICE_HANDLE usbDeviceHandle);
```

Returns

USB_DEVICE_CLIENT_STATUS type of client status.

Description

This function returns the status of the client (ready or closed). The application can use this function to query the present state of a client. Some of the USB Device Layer functions do not have any effect if the client handle is invalid. The USB_DEVICE_ClientStatusGet function in such cases can be used for debugging or trouble shooting.

Remarks

The application may ordinarily not find the need to use this function. It can be used for troubleshooting or debugging purposes.

Preconditions

The USB device layer must have been initialized and opened before calling this function.

Example

```
// This code example shows usage of the
// USB_DEVICE_ClientStatusGet function.

if(USB_DEVICE_CLIENT_STATUS_READY == USB_DEVICE_ClientStatusGet(usbDeviceHandle))
{
    // Client handle is valid.
    if(USB_DEVICE_IsSuspended(usbDeviceHandle))
    {
        // Device is suspended. Do something here.
    }
}
```

Parameters

Parameters	Description
usbDeviceHandle	Pointer to the device layer handle that is returned from USB_DEVICE_Open

Function

USB_DEVICE_CLIENT_STATUS USB_DEVICE_ClientStatusGet

(

USB_DEVICE_HANDLE usbDeviceHandle

)

USB_DEVICE_EventHandlerSet Function

USB Device Layer Event Handler Callback Function set function.

File

usb_device.h

C

```
void USB_DEVICE_EventHandlerSet(USB_DEVICE_HANDLE usbDeviceHandle, const USB_DEVICE_EVENT_HANDLER
callBackFunc, uintptr_t context);
```

Returns

None.

Description

This is the USB Device Layer Event Handler Callback Set function. A client can receive USB Device Layer event by using this function to register and event handler callback function. The client can additionally specify a specific context which will returned with the event handler callback function.

Remarks

None.

Preconditions

The device layer must have been initialized by calling USB_DEVICE_Initialize and a valid handle to the instance must have been obtained by calling USB_DEVICE_Open.

Example

```
// This code example shows how the application can set
```

```
// a Device Layer Event Handler.

// Application states
typedef enum
{
    //Application's state machine's initial state.
    APP_STATE_INIT=0,
    APP_STATE_SERVICE_TASKS,
    APP_STATE_WAIT_FOR_CONFIGURATION,
} APP_STATES;

USB_DEVICE_HANDLE usbDeviceHandle;
APP_STATES appState;

// This is the application device layer event handler function.

USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler
(
    USB_DEVICE_EVENT event,
    void * pData,
    uintptr_t context
)
{
    USB_SETUP_PACKET * setupPacket;
    switch(event)
    {
        case USB_DEVICE_EVENT_POWER_DETECTED:
            // This event in generated when VBUS is detected. Attach the device
            USB_DEVICE_Attach(usbDeviceHandle);
            break;

        case USB_DEVICE_EVENT_POWER_REMOVED:
            // This event is generated when VBUS is removed. Detach the device
            USB_DEVICE_Detach (usbDeviceHandle);
            break;

        case USB_DEVICE_EVENT_CONFIGURED:
            // This event indicates that Host has set Configuration in the Device.
            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST:
            // This event indicates a Control transfer setup stage has been completed.
            setupPacket = (USB_SETUP_PACKET *)pData;

            // Parse the setup packet and respond with a USB_DEVICE_ControlSend(),
            // USB_DEVICE_ControlReceive or USB_DEVICE_ControlStatus().

            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT:
            // This event indicates that a Control transfer Data has been sent to Host.
            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:
            // This event indicates that a Control transfer Data has been received from Host.
            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_ABORTED:
            // This event indicates a control transfer was aborted.
            break;

        case USB_DEVICE_EVENT_SUSPENDED:
            break;

        case USB_DEVICE_EVENT_RESUMED:
            break;

        case USB_DEVICE_EVENT_ERROR:
            break;
```

```
            case USB_DEVICE_EVENT_RESET:
                break;

            case USB_DEVICE_EVENT_SOF:
                // This event indicates an SOF is detected on the bus. The  USB_DEVICE_SOF_EVENT_ENABLE
                // macro should be defined to get this event.
                break;
            default:
                break;
        }
    }


    void APP_Tasks ( void )
    {
        // Check the application's current state.
        switch ( appState )
        {
            // Application's initial state.
            case APP_STATE_INIT:
                // Open the device layer
                usbDeviceHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0,
                    DRV_IO_INTENT_READWRITE );

                if(usbDeviceHandle != USB_DEVICE_HANDLE_INVALID)
                {
                    // Register a callback with device layer to get event notification
                    USB_DEVICE_EventHandlerSet(usbDeviceHandle,
                        APP_USBDeviceEventHandler, 0);
                    appState = APP_STATE_WAIT_FOR_CONFIGURATION;
                }
                else
                {
                    // The Device Layer is not ready to be opened. We should try
                    // gain later.
                }
                break;

            case APP_STATE_SERVICE_TASKS:
                break;

                // The default state should never be executed.
            default:
                break;
        }
    }
```

Parameters

Parameters	Description
usbDeviceHandle	Pointer to the device layer handle that is returned from USB_DEVICE_Open
callBackFunc	Pointer to the call back function. The device layer calls notifies the client about bus event by calling this function.
context	Client specific context

Function

void USB_DEVICE_EventHandlerSet

(

USB_DEVICE_HANDLE usbDeviceHandle,

USB_DEVICE_EVENT_HANDLER *callBackFunc,

uintptr_t context

)

c) Device Power State Management Functions

USB_DEVICE_PowerStateSet Function

Sets power state of the device.

File

usb_device.h

C

```
void USB_DEVICE_PowerStateSet(USB_DEVICE_HANDLE usbDeviceHandle, USB_DEVICE_POWER_STATE powerState);
```

Returns

None.

Description

Application clients can use this function to set the power state of the device. A USB device can be bus powered or self powered. Based on hardware configuration, this power state may change while the device is on operation. The application can call this function to update the Device Layer on the present power status of the device.

Remarks

By default, the device is bus powered.

Preconditions

The device layer should have been initialized and opened.

Example

```
// The following code example shows how the application can
// change the power state of the device. In this case the application checks
// if a battery is charged and if so, the application set the device power
// state to self-powered.

if(APP_BATTERY_IS_CHARGED == APP_BatteryChargeStatusGet())
{
    // The application switches if power source.

    APP_PowerSourceSwitch(APP_POWER_SOURCE_BATTERY);
    USB_DEVICE_PowerStateSet(usbDeviceHandle, USB_DEVICE_POWER_STATE_SELF_POWERED);
}
else
{
    // The battery is still not charged. The application uses the USB power.

    USB_DEVICE_PowerStateSet(usbDeviceHandle, USB_DEVICE_POWER_STATE_BUS_POWERED);
}
```

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open().
powerState	USB_DEVICE_POWER_STATE_BUS_POWERED/ USB_DEVICE_POWER_STATE_SELF_POWERED

Function

void USB_DEVICE_PowerStateSet

(

USB_DEVICE_HANDLE usbDeviceHandle,

USB_DEVICE_POWER_STATE powerState

)

USB_DEVICE_RemoteWakeupStatusGet Function

Gets the "Remote wake-up" status of the device.

File

usb_device.h

C

```
USB_DEVICE_REMOTE_WAKEUP_STATUS USB_DEVICE_RemoteWakeupStatusGet(USB_DEVICE_HANDLE usbDeviceHandle);
```

Returns

USB_DEVICE_REMOTE_WAKEUP_ENABLED - Remote wakeup is enabled. USB_DEVICE_REMOTE_WAKEUP_DISABLED - Remote wakeup is disabled.

Description

This function returns the present "Remote Wake-up" status of the device. If the device supports remote wake-up, the host may enable of disable this feature. The client can use this function to find out the status of this feature.

Remarks

None.

Preconditions

The device layer should have been initialized and opened.

Example

```
// This code example checks if the host has enabled the remote wake-up
// feature and then starts resume signaling. It is assumed
// that the device is in suspended mode.

USB_DEVICE_HANDLE usbDeviceHandle;

if(USB_DEVICE_RemoteWakeupStatusGet(usbDeviceHandle))
{
    // Start resume signaling.

    USB_DEVICE_RemoteWakeupStart(usbDeviceHandle);
}
```

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open().

Function

USB_DEVICE_REMOTE_WAKEUP_STATUS USB_DEVICE_RemoteWakeupStatusGet

(

USB_DEVICE_HANDLE usbDeviceHandle

)

USB_DEVICE_IsSuspended Function

Returns true if the device is in a suspended state.

File

usb_device.h

C

```
bool USB_DEVICE_IsSuspended(USB_DEVICE_HANDLE usbDeviceHandle);
```

Returns

Returns true if the device is suspended.

Description

This function returns true is the device is presently in suspended state. The application can use this function in conjunction with the USB_DEVICE_StateGet function to obtain the detailed state of the device (such as addressed and suspended, configured and suspended etc.). The Device Layer also provides the macro USB_DEVICE_EVENT_SUSPENDED event to indicate entry into suspend state.

Remarks

None.

Preconditions

The USB Device Layer must have been initialized and opened before calling this function.

Example

```
// This code example shows how the application
// can find out if the device is in a configured but suspended state.

if(USB_DEVICE_IsSuspended(usbDeviceHandle))
{
    // Device is in a suspended state.

    if(USB_DEVICE_STATE_CONFIGURED == USB_DEVICE_StateGet(usbDeviceHandle))
    {
        // This means the device is in configured and suspended state.

    }
}
```

Parameters

Parameters	Description
usbDeviceHandle	Pointer to the Device Layer Handle that is returned from USB_DEVICE_Open

Function

bool USB_DEVICE_IsSuspended(USB_DEVICE_HANDLE usbDeviceHandle)

USB_DEVICE_RemoteWakeupStart Function

This function will start the resume signaling.

File

usb_device.h

C

```
void USB_DEVICE_RemoteWakeupStart(USB_DEVICE_HANDLE usbDeviceHandle);
```

Returns

None.

Description

This function will start the resume signaling on the bus. The client calls this function after it has detected a idle bus (through the USB_DEVICE_EVENT_SUSPENDED event). The remote wake-up feature should have been enabled by the host, before the client can call this function. The client can use the USB_DEVICE_RemoteWakeupStatusGet function to check if the host has enabled the remote wake-up feature.

Remarks

None.

Preconditions

Client handle should be valid. The remote wake-up feature should have been enabled by the host.

Example

```
// This code example shows how the device can enable and disable
// Resume signaling on the bus. These function should only be called if the
// device support remote wakeup and the host has enabled this
// feature.

USB_DEVICE_HANDLE usbDeviceHandle;

// Start resume signaling.
USB_DEVICE_RemoteWakeupStart(usbDeviceHandle);
```

```
// As per section 7.1.7.7 of the USB specification, device can
// drive resume signaling for at least 1 millisecond but no
// more than 15 milliseconds.

APP_DelayMilliseconds(10);

// Stop resume signaling.
USB_DEVICE_RemoteWakeupStop(usbDeviceHandle);
```

Parameters

Parameters	Description
usbDeviceHandle	Client's driver handle (returned from USB_DEVICE_Open)

Function

void USB_DEVICE_RemoteWakeupStart(USB_DEVICE_HANDLE usbDeviceHandle)

USB_DEVICE_RemoteWakeupStartTimed Function

This function will start a self timed Remote Wake-up.

File

usb_device.h

C

```
void USB_DEVICE_RemoteWakeupStartTimed(USB_DEVICE_HANDLE usbDeviceHandle);
```

Returns

None.

Description

This function will start a self timed Remote Wake-up sequence. The function will cause the device to generate resume signaling for 10 milliseconds. The resume signaling will stop after 10 milliseconds. The application can use this function instead of the USB_DEVICE_RemoteWakeupStart and USB_DEVICE_RemoteWakeupStop functions, which require the application to manually start, maintain duration and then stop the resume signaling.

Remarks

None.

Preconditions

Client handle should be valid. The host should have enabled the Remote Wake-up feature for this device.

Example

```
// This code example shows how the device can use the
// USB_DEVICE_RemoteWakeupStartTimed function to drive resume signaling
// on the bus for 10 milliseconds.

USB_DEVICE_HANDLE usbDeviceHandle;

// Check if host has enabled remote wake-up for the device.
if(USB_DEVICE_REMOTE_WAKEUP_ENABLED == USB_DEVICE_RemoteWakeupStatusGet(usbDeviceHandle))
{
    // Remote wake-up is enabled

    USB_DEVICE_RemoteWakeupStartTimed(usbDeviceHandle);
}
```

Parameters

Parameters	Description
usbDeviceHandle	Client's driver handle (returned from USB_DEVICE_Open)

Function

void USB_DEVICE_RemoteWakeupStartTimed (USB_DEVICE_HANDLE usbDeviceHandle)

USB_DEVICE_RemoteWakeupStop Function

This function will stop the resume signaling.

File

usb_device.h

C

```
void USB_DEVICE_RemoteWakeupStop(USB_DEVICE_HANDLE usbDeviceHandle);
```

Returns

None.

Description

This function will stop the resume signaling. This function should be called after the client has called the USB_DEVICE_RemoteWakeupStart() function.

Remarks

None.

Preconditions

Client handle should be valid. The host should have enabled the Remote Wakeup feature for this device.

Example

```
// This code example shows how the device can enable and disable
// Resume signaling on the bus. These function should only be called if the
// device support remote wake-up and the host has enabled this
// feature.

USB_DEVICE_HANDLE usbDeviceHandle;

// Start resume signaling.
USB_DEVICE_RemoteWakeupStart(usbDeviceHandle);

// As per section 7.1.7.7 of the USB specification, device must
// drive resume signaling for at least 1 millisecond but no
// more than 15 milliseconds.

APP_DelayMilliseconds(10);

// Stop resume signaling.
USB_DEVICE_RemoteWakeupStop(usbDeviceHandle);
```

Parameters

Parameters	Description
usbDeviceHandle	Client's driver handle (returned from USB_DEVICE_Open)

Function

void USB_DEVICE_RemoteWakeupStop (USB_DEVICE_HANDLE usbDeviceHandle)

d) Device Management Functions

USB_DEVICE_StateGet Function

Returns the current state of the USB device.

File

usb_device.h

C

```
USB_DEVICE_STATE USB_DEVICE_StateGet(USB_DEVICE_HANDLE usbDeviceHandle);
```

Returns

USB_DEVICE_STATE_DETACHED - Device is not in any of the known states

USB_DEVICE_STATE_ATTACHED - Device is attached to the USB, but is not powered

USB_DEVICE_STATE_POWERED - Device is attached to the USB and powered, but has not been reset

USB_DEVICE_STATE_DEFAULT - Device is attached to the USB and powered and has been reset, but has not been assigned a unique address

USB_DEVICE_STATE_ADDRESS - Device is attached to the USB, powered, has been reset, and a unique device address has been assigned

USB_DEVICE_STATE_CONFIGURED - Device is attached to the USB, powered, has been reset, has a unique address, is configured, and is not suspended

Description

This function returns the current state of the USB device, as described in Chapter 9 of the USB 2.0 Specification.

Remarks

None.

Preconditions

The USB device layer must have been initialized and opened before calling this function.

Example

```
USB_DEVICE_STATE usbDevState;

// Get USB Device State.
usbDevState = USB_DEVICE_StateGet( usbDeviceHandle );

switch(usbDevState)
{
    case USB_DEVICE_STATE_ATTACHED:
        // Add code here
        break;

    case USB_DEVICE_STATE_POWERED:
       // Add code here
       break;

    default:
        break;
 }
```

Parameters

Parameters	Description
usbDeviceHandle	Pointer to the device layer handle that is returned from USB_DEVICE_Open

Function

USB_DEVICE_STATE USB_DEVICE_StateGet(USB_DEVICE_HANDLE usbDeviceHandle)

USB_DEVICE_Attach Function

This function will attach the device to the USB.

File

usb_device.h

C

```
void USB_DEVICE_Attach(USB_DEVICE_HANDLE usbDeviceHandle);
```

Returns

None.

Description

This function will attach the device to the USB. It does this by enabling the pull up resistors on the D+ or D- lines. This function should be called after the USB device layer has generated the USB_DEVICE_EVENT_POWER_DETECTED event.

Remarks

None.

Preconditions

Client handle should be valid. The device layer should have been initialized and an device layer event handler function should have been assigned.

Example

```
// This code example shows the set
// of steps to follow before attaching the
// device on the bus. It is assumed that the
// device layer is already initialized.

USB_DEVICE_HANDLE usbDeviceHandle;

// Get an handle to the USB device layer.
usbDeviceHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0,
                                                DRV_IO_INTENT_READWRITE );

if(USB_DEVICE_HANDLE_INVALID == usbDeviceHandle)
{
    // Failed to open handle.
    // Handle error.
}

// Register an event handler call back function with device layer
// so that we are ready to receive events when the device is
// attached to the bus.

USB_DEVICE_EventHandlerSet(usbDeviceHandle, APP_USBDeviceEventHandler, NULL);

// Now, connect device to USB
USB_DEVICE_Attach(usbDeviceHandle);
```

Parameters

Parameters	Description
usbDeviceHandle	Client's USB device layer handle (returned from USB_DEVICE_Open)

Function

void USB_DEVICE_Attach(USB_DEVICE_HANDLE usbDeviceHandle)

USB_DEVICE_Detach Function

This function will detach the device from the USB.

File

usb_device.h

C

```
void USB_DEVICE_Detach(USB_DEVICE_HANDLE usbDeviceHandle);
```

Returns

None.

Description

This function will detach the device from the USB. It does this by disabling the pull up resistors on the D+ or D- lines. This function should be called when the application wants to disconnect the device from the bus (typically to implement a soft detach or switch to host mode operation). It should be called when the Device Layer has generated the USB_DEVICE_EVENT_POWER_REMOVED event.

Remarks

None.

Preconditions

The device layer should have been initialized and opened.

Example

```
USB_DEVICE_HANDLE usbDeviceHandle;

// Detach the device from the USB
USB_DEVICE_Detach( usbDeviceHandle );
```

Parameters

Parameters	Description
usbDeviceHandle	Client's driver handle (returned from USB_DEVICE_Open)

Function

void USB_DEVICE_Detach(USB_DEVICE_HANDLE usbDeviceHandle);

USB_DEVICE_ActiveConfigurationGet Function

Informs the client of the current USB device configuration set by the USB host.

File

usb_device.h

C

```
uint8_t USB_DEVICE_ActiveConfigurationGet(USB_DEVICE_HANDLE usbDeviceHandle);
```

Returns

Present active configuration.

Description

This function returns the current active USB device configuration.

Remarks

None.

Preconditions

The USB Device Layer must have been initialized and opened before calling this function.

Example

```
// This code example shows how the
// USB_DEVICE_ActiveConfigurationGet function can be called to obtain
// the configuration that has been set by the host. Note that this
// information is also available in the macro USB_DEVICE_EVENT_CONFIGURED.

uint8_t currentConfiguration;
USB_DEVICE_HANDLE usbDeviceHandle;

currentConfiguration = USB_DEVICE_ActiveConfigurationGet(usbDeviceHandle);
```

Parameters

Parameters	Description
usbDeviceHandle	Pointer to the Device Layer Handle that is returned from USB_DEVICE_Open

Function

uint8_t USB_DEVICE_ActiveConfigurationGet(USB_DEVICE_HANDLE usbDeviceHandle)

USB_DEVICE_ActiveSpeedGet Function

Informs the client of the current operation speed of the USB bus.

File

usb_device.h

C

```
USB_SPEED USB_DEVICE_ActiveSpeedGet(USB_DEVICE_HANDLE usbDeviceHandle);
```

Returns

USB_SPEED_LOW - USB module is at low-speed USB_SPEED_FULL - USB module is at full-speed USB_SPEED_HIGH - USB module is at high-speed

Description

The USB device stack supports both high speed and full speed operations. This function returns the current operation speed of the USB bus. This function should be called after the USB_DEVICE_EVENT_RESET event has occurred.

Remarks

None.

Preconditions

The USB device layer must have been initialized and a valid handle to USB device layer must have been opened.

Example

```
// This code example shows how the
// USB_DEVICE_GetDeviceSpeed function can be called to obtain
// the current device speed. This information is also
// available in the USB_DEVICE_EVENT_CONFIGURED event.

if(USB_DEVICE_ActiveSpeedGet(usbDeviceHandle) == USB_SPEED_FULL)
{
    // This means the device attached at full speed.
}
else if(USB_DEVICE_ActiveSpeedGet(usbDeviceHandle) == USB_SPEED_HIGH)
{
    // This means the device attached at high speed.
}
```

Parameters

Parameters	Description
usbDeviceHandle	Pointer to device layer handle that is returned from USB_DEVICE_Open

Function

USB_SPEED USB_DEVICE_ActiveSpeedGet(USB_DEVICE_HANDLE usbDeviceHandle)

e) Endpoint Management Functions

USB_DEVICE_EndpointIsStalled Function

This function returns the stall status of the specified endpoint and direction.

File

usb_device.h

C

```
bool USB_DEVICE_EndpointIsStalled(USB_DEVICE_HANDLE usbDeviceHandle, USB_ENDPOINT_ADDRESS endpoint);
```

Returns

Returns true if endpoint is stalled, false otherwise.

Description

This function returns the stall status of the specified endpoint and direction.

Remarks

None.

Preconditions

The USB Device should be in a configured state.

Example

```
// This code example shows of how the
// USB_DEVICE_EndpointIsStalled function can be used to obtain the
// stall status of the endpoint 1 and IN direction.

USB_ENDPOINT_ADDRESS ep;

ep = 0x1|USB_EP_DIRECTION_IN;

if(true == USB_DEVICE_EndpointIsStalled (handle, ep))
{
    // Endpoint stall is enabled. Clear the stall.

    USB_DEVICE_EndpointStallClear(handle, ep);

}
```

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open
endpoint	Specifies the endpoint and direction

Function

bool USB_DEVICE_EndpointIsStalled

(

USB_DEVICE_HANDLE usbDeviceHandle,

USB_ENDPOINT_ADDRESS endpoint

)

USB_DEVICE_EndpointStall Function

This function stalls an endpoint in the specified direction.

File

usb_device.h

C

```
void USB_DEVICE_EndpointStall(USB_DEVICE_HANDLE usbDeviceHandle, USB_ENDPOINT_ADDRESS endpoint);
```

Returns

None.

Description

This function stalls an endpoint in the specified direction.

Remarks

The application may typically, not find the need to stall an endpoint. Stalling an endpoint erroneously could potentially make the device non-compliant.

Preconditions

Client handle should be valid.

Example

```
// This code example shows how to stall an endpoint. In
// this case, endpoint 1 IN direction is stalled.

USB_ENDPOINT_ADDRESS ep;

ep = 0x1|USB_EP_DIRECTION_IN;

USB_DEVICE_EndpointStall(usbDeviceHandle, ep);
```

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open
endpoint	Specifies the endpoint and direction

Function

void USB_DEVICE_EndpointStall
(
USB_DEVICE_HANDLE usbDeviceHandle,
USB_ENDPOINT_ADDRESS endpoint
)

USB_DEVICE_EndpointStallClear Function

This function clears the stall on an endpoint in the specified direction.

File

usb_device.h

C

```
void USB_DEVICE_EndpointStallClear(USB_DEVICE_HANDLE usbDeviceHandle, USB_ENDPOINT_ADDRESS endpoint);
```

Returns

None.

Description

This function clear the stall on an endpoint in the specified direction.

Remarks

None.

Preconditions

Client handle should be valid.

Example

```
// This code example shows how to clear a stall on an
// endpoint. In this case, the stall on endpoint 1 IN direction is
// cleared.

USB_ENDPOINT_ADDRESS ep;

ep = USB_ENDPOINT_AND_DIRECTION(USB_DATA_DIRECTION_DEVICE_TO_HOST, 1);

USB_DEVICE_EndpointStallClear(usbDeviceHandle, ep);
```

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open().
endpoint	Specifies the endpoint and direction.

Function

void USB_DEVICE_EndpointStallClear
(
USB_DEVICE_HANDLE usbDeviceHandle,
USB_ENDPOINT_ADDRESS endpoint
)

USB_DEVICE_EndpointDisable Function

Disables a device endpoint.

File

usb_device.h

C

```
USB_DEVICE_RESULT USB_DEVICE_EndpointDisable(USB_DEVICE_HANDLE usbDeviceHandle, USB_ENDPOINT_ADDRESS
endpoint);
```

Returns

USB_DEVICE_RESULT_OK - The endpoint was enabled successfully.

USB_DEVICE_RESULT_ERROR_ENDPOINT_INVALID - The specified instance was not provisioned in the application and is invalid.

Description

This function disables a device endpoint. The application may need to disable the endpoint when it want to change the endpoint characteristics. This could happen when the device features interfaces with multiple alternate settings. If such cases, the host may request the device to switch to specific alternate setting by sending the Set Interface request. The device application must then disable the endpoint (if it was enabled) before re-enabling it with the new settings.The application can use the USB_DEVICE_EndpointIsEnabled function to check the status of the endpoint and USB_DEVICE_EndpointEnable function to enable the endpoint.

Remarks

None.

Preconditions

The device should have been configured.

Example

```
// The following code example checks if an Set Alternate request has
// been received and changes the endpoint characteristics based on the
// alternate setting. Endpoint is 1 and direction is device to host.
// Assume that endpoint size was 32 bytes in alternate setting 0.

if(setAlternateRequest)
{
    if(alternateSetting == 1)
    {
        // Check if the endpoint is already enabled.
        if(USB_DEVICE_EndpointIsEnabled(usbDeviceHandle, (0x1|USB_EP_DIRECTION_IN)))
        {
            // Disable the endpoint.
            USB_DEVICE_EndpointDisable(usbDeviceHandle, (0x1|USB_EP_DIRECTION_IN));
        }

        // Re-enable the endpoint with new settings
        USB_DEVICE_EndpointEnable(usbDeviceHandle, (0x1|USB_EP_DIRECTION_IN)
                    USB_TRANSFER_TYPE_BULK, 64);
    }
}
```

Parameters

Parameters	Description
usbDeviceHandle	USB Device Layer Handle.
endpoint	Endpoint to disable.

Function

USB_DEVICE_RESULT USB_DEVICE_EndpointDisable

(

USB_DEVICE_HANDLE usbDeviceHandle,

USB_ENDPOINT_ADDRESS endpoint,

);

USB_DEVICE_EndpointEnable Function

Enables a device endpoint.

File

usb_device.h

C

```
USB_DEVICE_RESULT USB_DEVICE_EndpointEnable(USB_DEVICE_HANDLE usbDeviceHandle, uint8_t interface,
USB_ENDPOINT_ADDRESS endpoint, USB_TRANSFER_TYPE transferType, size_t size);
```

Returns

USB_DEVICE_RESULT_OK - The endpoint was enabled successfully.

USB_DEVICE_RESULT_ERROR_ENDPOINT_INVALID - The specified instance was not provisioned in the application and is invalid.

Description

This function enables a device endpoint for the specified transfer type and size. A Vendor specific device application may typically call this function in response to a Set Interface request from the host. Note that Device Layer will enable endpoints contained in Alternate Setting 0 of an interface, when the host configures the device. If there is only one alternate setting in an interface, the application may not need to call the USB_DEVICE_EndpointEnable function.

If the device supports multiple alternate settings in an Interface, the device application must then disable an endpoint (if it was enabled) before re-enabling it with the new settings.The application can use the USB_DEVICE_EndpointIsEnabled function to check the status of the endpoint and USB_DEVICE_EndpointDisable function to disable the endpoint.

Remarks

None.

Preconditions

The device should have been configured.

Example

```
// The following code example checks if an Set Alternate request has
// been received and changes the endpoint characteristics based on the
// alternate setting. Endpoint is 1 and direction is device to host.
// Assume that endpoint size was 32 bytes in alternate setting 0.

if(setAlternateRequest)
{
    if(alternateSetting == 1)
    {
        // Check if the endpoint is already enabled.
        if(USB_DEVICE_EndpointIsEnabled(usbDeviceHandle, (0x1|USB_EP_DIRECTION_IN)))
        {
            // Disable the endpoint.
            USB_DEVICE_EndpointDisable(usbDeviceHandle, (0x1|USB_EP_DIRECTION_IN));
        }

        // Re-enable the endpoint with new settings
        USB_DEVICE_EndpointEnable(usbDeviceHandle, 0, (0x1|USB_EP_DIRECTION_IN)
                    USB_TRANSFER_TYPE_BULK, 64);
    }
}
```

Parameters

Parameters	Description
usbDeviceHandle	USB Device Layer Handle.
interface	This parameter is ignored in the PIC32 USB Device Stack implementation.
endpoint	Endpoint to enable.
transferType	Type of transfer that this is endpoint will support. This should match the type reported to the host
size	Maximum endpoint size. This should match the value reported to the host.

Function

USB_DEVICE_RESULT USB_DEVICE_EndpointEnable
(
USB_DEVICE_HANDLE usbDeviceHandle,
uint8_t interface,
USB_ENDPOINT_ADDRESS endpoint,
USB_TRANSFER_TYPE transferType
size_t size
);

USB_DEVICE_EndpointIsEnabled Function

Returns true if the endpoint is enabled.

File

usb_device.h

C

```
bool USB_DEVICE_EndpointIsEnabled(USB_DEVICE_HANDLE usbDeviceHandle, USB_ENDPOINT_ADDRESS endpoint);
```

Returns

true - The endpoint is enabled.

false - The endpoint is not enabled or the specified endpoint is not provisioned in the system and is invalid.

Description

This function returns true if the endpoint is enabled. The application can use this function when handling Set Interface requests in case of Vendor or Custom USB devices.

Remarks

None.

Preconditions

The device should have been configured.

Example

```
// The following code example checks if an Set Alternate request has
// been received and changes the endpoint characteristics based on the
// alternate setting. Endpoint is 1 and direction is device to host.
// Assume that endpoint size was 32 bytes in alternate setting 0.

if(setAlternateRequest)
{
    if(alternateSetting == 1)
    {
        // Check if the endpoint is already enabled.
        if(USB_DEVICE_EndpointIsEnabled(usbDeviceHandle, (0x1|USB_EP_DIRECTION_IN)))
        {
            // Disable the endpoint.
            USB_DEVICE_EndpointDisable(usbDeviceHandle, (0x1|USB_EP_DIRECTION_IN));
        }

        // Re-enable the endpoint with new settings
        USB_DEVICE_EndpointEnable(usbDeviceHandle, (0x1|USB_EP_DIRECTION_IN)
                    USB_TRANSFER_TYPE_BULK, 64);
    }
}
```

Parameters

Parameters	Description
usbDeviceHandle	USB Device Layer Handle.

endpoint	Endpoint to disable.

Function

bool USB_DEVICE_EndpointIsEnabled
(
USB_DEVICE_HANDLE usbDeviceHandle,
USB_ENDPOINT_ADDRESS endpoint,
);

USB_DEVICE_EndpointRead Function

Reads data received from host on the requested endpoint.

File

usb_device.h

C

```
USB_DEVICE_RESULT USB_DEVICE_EndpointRead(USB_DEVICE_HANDLE usbDeviceHandle, USB_DEVICE_TRANSFER_HANDLE *
transferHandle, USB_ENDPOINT_ADDRESS endpoint, void * buffer, size_t bufferSize);
```

Returns

USB_DEVICE_RESULT_OK - The read request was successful. transferHandle contains a valid transfer handle.

USB_DEVICE_RESULT_ERROR_TRANSFER_QUEUE_FULL - internal request queue is full. The write request could not be added.

USB_DEVICE_RESULT_ERROR_TRANSFER_SIZE_INVALID - The specified transfer size was not a multiple of endpoint size or is 0.

USB_DEVICE_RESULT_ERROR_ENDPOINT_NOT_CONFIGURED - The specified endpoint is not configured yet and is not ready for data transfers.

USB_DEVICE_RESULT_ERROR_ENDPOINT_INVALID - The specified instance was not provisioned in the application and is invalid.

Description

This function requests a endpoint data read from the USB Device Layer. The function places a requests with driver, the request will get serviced as data is made available by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE event. The amount of data read and the transfer handle associated with the request is returned along with the event in the pData parameter of the event handler. The transfer handle expires when event handler for the USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE exits. If the read request could not be accepted, the function returns an error code and transferHandle will contain the value USB_DEVICE_TRANSFER_HANDLE_INVALID.

If the size parameter is not a multiple of maxPacketSize or is 0, the function returns USB_DEVICE_TRANSFER_HANDLE_INVALID in transferHandle and returns an error code as a return value. If the size parameter is a multiple of maxPacketSize and the host sends less than maxPacketSize data in any transaction, the transfer completes and the function driver will issue a USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE event along with the USB_DEVICE_EVENT_DATA_ENDPOINT_READ_COMPLETE_DATA data structure. If the size parameter is a multiple of maxPacketSize and the host sends maxPacketSize amount of data, and total data received does not exceed size, then the function driver will wait for the next packet.

Remarks

While the using the device layer with PIC32MZ USB module, the receive buffer provided to the USB_DEVICE_EndpointRead should be placed in coherent memory and aligned at a 16 byte boundary. This can be done by declaring the buffer using the __attribute__((coherent, aligned(16))) attribute. An example is shown here

```
uint8_t data[256] __attribute__((coherent, aligned(16)));
```

Preconditions

The device should have been configured.

Example

```
// Shows an example of how to read. This assumes that
// driver was opened successfully. Note how the endpoint
// is specified. The most significant bit is cleared while
// the lower nibble specifies the endpoint (which is 1).

USB_DEVICE_TRANSFER_HANDLE transferHandle;
USB_DEVICE_RESULT readRequestResult;
USB_DEVICE_HANDLE usbDeviceHandle;
USB_ENDPOINT_ADDRESS endpoint = 0x01;
```

```
readRequestResult = USB_DEVICE_EndpointRead(usbDeviceHandle,
                        &transferHandle, endpoint, data, 128);

if(USB_DEVICE_RESULT_OK != readRequestResult)
{
    //Do Error handling here
}

// The completion of the read request will be indicated by the
// USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE event.
```

Parameters

Parameters	Description
usbDeviceHandle	USB Device Layer Handle.
transferHandle	Pointer to a USB_DEVICE_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the read request was successful.
endpoint	Endpoint from which the data should be read.
data	pointer to the data buffer where read data will be stored.
size	Size of the data buffer. Refer to the description section for more details on how the size affects the transfer.

Function

```
  USB_DEVICE_RESULT USB_DEVICE_EndpointRead
(
    USB_DEVICE_HANDLE usbDeviceHandle,
    USB_DEVICE_TRANSFER_HANDLE * transferHandle,
USB_ENDPOINT_ADDRESS endpoint,
void * buffer,
size_t bufferSize
);
```

USB_DEVICE_EndpointTransferCancel Function

This function cancels a transfer scheduled on an endpoint.

File

usb_device.h

C

```
USB_DEVICE_RESULT USB_DEVICE_EndpointTransferCancel(USB_DEVICE_HANDLE usbDeviceHandle, USB_ENDPOINT_ADDRESS
endpoint, USB_DEVICE_TRANSFER_HANDLE transferHandle);
```

Returns

USB_DEVICE_RESULT_OK - The transfer will be canceled completely or partially.

USB_DEVICE_RESULT_ERROR - The transfer could not be canceled because it has either completed, the transfer handle is invalid or the last transaction is in progress.

Description

This function cancels a transfer scheduled on an endpoint using the USB_DEVICE_EndpointRead and USB_DEVICE_EndpointWrite functions. If a transfer is still in the queue and its processing has not started, the transfer is canceled completely. A transfer that is in progress may or may not get canceled depending on the transaction that is presently in progress. If the last transaction of the transfer is in progress, the transfer will not be canceled. If it is not the last transaction in progress, the in progress transfer will be allowed to complete. Pending transactions will be canceled. The first transaction of an in progress transfer cannot be canceled.

Remarks

None.

Preconditions

The USB Device should be in a configured state.

Example

```
// The following code example cancels a transfer on endpoint 1, IN direction.
```

```
USB_DEVICE_TRANSFER_HANDLE transferHandle;
USB_DEVICE_RESULT result;

result = USB_DEVICE_EndpointTransferCancel(usbDeviceHandle,
            (0x1|USB_EP_DIRECTION_IN), transferHandle);

if(USB_DEVICE_RESULT_OK == result)
{
    // The transfer cancellation was either completely or
    // partially successful.
}
```

Parameters

Parameters	Description
usbDeviceHandle	USB Device Layer Handle.
endpoint	Endpoint of which the transfer needs to be canceled.
handle	Transfer handle of the transfer to be canceled.

Function

USB_DEVICE_RESULT USB_DEVICE_EndpointTransferCancel
(
USB_DEVICE_HANDLE usbDeviceHandle,
USB_ENDPOINT_ADDRESS endpoint,
USB_DEVICE_TRANSFER_HANDLE handle
);

USB_DEVICE_EndpointWrite Function

This function requests a data write to a USB Device Endpoint.

File

usb_device.h

C

```
USB_DEVICE_RESULT USB_DEVICE_EndpointWrite(USB_DEVICE_HANDLE usbDeviceHandle, USB_DEVICE_TRANSFER_HANDLE *
transferHandle, USB_ENDPOINT_ADDRESS endpoint, const void * data, size_t size, USB_DEVICE_TRANSFER_FLAGS
flags);
```

Returns

USB_DEVICE_RESULT_OK - The write request was successful. transferHandle contains a valid transfer handle.

USB_DEVICE_RESULT_ERROR_TRANSFER_QUEUE_FULL - Internal request queue is full. The write request could not be added.

USB_DEVICE_RESULT_ERROR_ENDPOINT_INVALID - Endpoint is not provisioned in the system.

USB_DEVICE_RESULT_ERROR_TRANSFER_SIZE_INVALID - The combination of the transfer size and the specified flag is invalid.

USB_DEVICE_RESULT_ERROR_ENDPOINT_NOT_CONFIGURED - Endpoint is not enabled because device is not configured.

USB_DEVICE_RESULT_ERROR_PARAMETER_INVALID - Device Layer handle is not valid.

Description

This function requests a data write to the USB Device Endpoint. The function places a requests with Device layer, the request will get serviced as and when the data is requested by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE event. The amount of data written and the transfer handle associated with the request is returned along with the event in length member of the pData parameter in the event handler. The transfer handle expires when event handler for the USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE exits. If the write request could not be accepted, the function returns an error code and transferHandle will contain the value USB_DEVICE_TRANSFER_HANDLE_INVALID.

The behavior of the write request depends on the flags and size parameter. If the application intends to send more data in a request, then it should use the USB_DEVICE_TRANSFER_FLAGS_MORE_DATA_PENDING flag. If there is no more data to be sent in the request, the application must use the USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE flag. This is explained in more detail here:

- If size is a multiple of maxPacketSize and flag is set as

USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE, the write function will append a Zero Length Packet (ZLP) to complete the transfer.

- If size is a multiple of maxPacketSize and flag is set as

USB_DEVICE_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function will not append a ZLP and hence will not complete the transfer.

- If size is greater than but not a multiple of maxPacketSize and flags is

set as USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE, the write function complete the transfer without appending a ZLP.

- If size is greater than but not a multiple of maxPacketSize and flags is

set as USB_DEVICE_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function returns an error code and sets the transferHandle parameter to USB_DEVICE_TRANSFER_HANDLE_INVALID.

- If size is less than maxPacketSize and flag is set as

USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE, the write function schedules one packet.

- If size is less than maxPacketSize and flag is set as

USB_DEVICE_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function returns an error code and sets the transferHandle parameter to USB_DEVICE_TRANSFER_HANDLE_INVALID.

Remarks

While the using the device layer with PIC32MZ USB module, the transmit buffer provided to the USB_DEVICE_EndpointWrite should be placed in coherent memory and aligned at a 16 byte boundary. This can be done by declaring the buffer using the __attribute__((coherent, aligned(16))) attribute. An example is shown here

```
uint8_t data[256] __attribute__((coherent, aligned(16)));
```

Preconditions

The USB Device should be in a configured state.

Example

```
// Below is a set of examples showing various conditions trying to
// send data with the Write command.
//
// This assumes that Device Layer was opened successfully.
// Assume maxPacketSize is 64.

USB_DEVICE_TRANSFER_HANDLE transferHandle;
USB_DEVICE_RESULT writeRequestHandle;
USB_DEVICE_HANDLE usbDeviceHandle;

//-------------------------------------------------------
// In this example we want to send 34 bytes only.

writeRequestResult = USB_DEVICE_EndpointWrite(usbDeviceHandle,
                        &transferHandle, data, 34,
                        USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE);

if(USB_DEVICE_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

//-------------------------------------------------------
// In this example we want to send 64 bytes only.
// This will cause a ZLP to be sent.

writeRequestResult = USB_DEVICE_EndpointWrite(usbDeviceHandle,
                        &transferHandle, data, 64,
                        USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE);

if(USB_DEVICE_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

//-------------------------------------------------------
// This example will return an error because size is less
// than maxPacketSize and the flag indicates that more
// data is pending.

writeRequestResult = USB_DEVICE_EndpointWrite(usbDeviceHandle,
                        &transferHandle, data, 32,
                        USB_DEVICE_TRANSFER_FLAGS_MORE_DATA_PENDING);

//-------------------------------------------------------
```

```
// In this example we want to place a request for a 70 byte transfer.
// The 70 bytes will be sent out in a 64 byte transaction and a 6 byte
// transaction completing the transfer.

writeRequestResult = USB_DEVICE_EndpointWrite(usbDeviceHandle,
                        &transferHandle, data, 70,
                        USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE);

if(USB_DEVICE_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

//------------------------------------------------------
// This example would result in an error because the transfer size is
// not an exact multiple of the endpoint size and the
// USB_DEVICE_TRANSFER_FLAGS_MORE_DATA_PENDING flag indicate that the
// transfer should continue.

writeRequestResult = USB_DEVICE_EndpointWrite(usbDeviceHandle,
                        &transferHandle, data, 70,
                        USB_DEVICE_TRANSFER_FLAGS_MORE_DATA_PENDING);

if(USB_DEVICE_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

// The completion of the write request will be indicated by the
// USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE event.
```

Parameters

Parameters	Description
instance	Handle to the device layer.
transferHandle	Pointer to a USB_DEVICE_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the write request was successful.
endpoint	Endpoint to which the data should be written. Note that is a combination of direction and the endpoint number. Refer to the description of USB_ENDPOINT_ADDRESS for more details.
data	Pointer to the data buffer that contains the data to written.
size	Size of the data buffer. Refer to the description section for more details on how the size affects the transfer.
flags	Flags that indicate whether the transfer should continue or end. Refer to the description for more details.

Function

```
  USB_DEVICE_RESULT USB_DEVICE_EndpointWrite
(
    USB_DEVICE_HANDLE usbDeviceHandle,
    USB_DEVICE_TRANSFER_HANDLE * transferHandle,
USB_ENDPOINT_ADDRESS endpoint,
const void * data,
size_t size,
    USB_DEVICE_TRANSFER_FLAGS flag
);
```

f) Control Transfer Functions

USB_DEVICE_ControlReceive Function

Receives data stage of the control transfer from host to device.

File

usb_device.h

C

```
USB_DEVICE_CONTROL_TRANSFER_RESULT USB_DEVICE_ControlReceive(USB_DEVICE_HANDLE usbDeviceHandle, void *
data, size_t length);
```

Returns

USB_DEVICE_CONTROL_TRANSFER_RESULT_FAILED - If control transfer failed due to host aborting the previous control transfer.

USB_DEVICE_CONTROL_TRANSFER_RESULT_SUCCESS - The request was submitted successfully.

Description

This function allows the application to specify the data buffer that would be needed to receive the data stage of a control write transfer. It should be called when the application receives the USB_DEVICE_CONTROL_TRANSFER_EVENT_SETUP_REQUEST event and has identified this setup request as the setup stage of a control write transfer. The function can be called in the Application Control Transfer Event handler or can be called after the application has returned from the control transfer event handler.

Calling this function after returning from the event handler defers the response to the event. This allows the application to prepare the data buffer out of the event handler context, especially if the data buffer to receive the data is not readily available. Note however, that there are timing considerations when responding to the control transfer. Exceeding the response time will cause the host to cancel the control transfer and may cause USB host to reject the device.

Remarks

None.

Preconditions

Client handle should be valid.

Example

```
// The following code example shows an example of how the
// USB_DEVICE_ControlReceive function is called in response to the
// USB_DEVICE_CONTROL_TRANSFER_EVENT_SETUP_REQUEST event to enable a control
// write transfer.

void APP_USBDeviceControlTransferEventHandler
(
    USB_DEVICE_EVENT event,
    void * pData,
    uintptr_t context
)
{
    uint8_t * setupPkt;

    switch(event)
    {
        case USB_DEVICE_CONTROL_TRANSFER_EVENT_SETUP_REQUEST:

            setupPkt = (uint8_t *)pData;

            // Submit a buffer to receive 32 bytes in the  control write transfer.
            USB_DEVICE_ControlReceive(usbDeviceHandle, data, 32);
            break;

        case USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA_RECEIVED:

            // This means that data in the data stage of the control
            // write transfer has been received. The application can either
            // accept the received data by calling the
            // USB_DEVICE_ControlStatus function with
            // USB_DEVICE_CONTROL_STATUS_OK flag (as shown in this example)
            // or it can reject it by calling the USB_DEVICE_ControlStatus()
            // function with USB_DEVICE_CONTROL_STATUS_ERROR flag.

            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;
```

```
            case USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA_SENT:

                // This means that data in the data stage of the control
                // read transfer has been sent. The application would typically
                // end the control transfer by calling the
                // USB_DEVICE_ControlStatus function with
                // USB_DEVICE_CONTROL_STATUS_OK flag (as shown in this example).

                USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
                break;

            case USB_DEVICE_CONTROL_TRANSFER_EVENT_ABORTED:

                // This means the host has aborted the control transfer. The
                // application can reset its control transfer state machine.
            break;
        }

    }
```

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open
data	Pointer to buffer that holds data
length	Size in bytes

Function

USB_DEVICE_CONTROL_TRANSFER_RESULT USB_DEVICE_ControlReceive

(

USB_DEVICE_HANDLE usbDeviceHandle,

void * data,

size_t length

)

USB_DEVICE_ControlSend Function

Sends data stage of the control transfer from device to host.

File

usb_device.h

C

```
USB_DEVICE_CONTROL_TRANSFER_RESULT USB_DEVICE_ControlSend(USB_DEVICE_HANDLE usbDeviceHandle, void * data,
size_t length);
```

Returns

USB_DEVICE_CONTROL_TRANSFER_RESULT_FAILED - If control transfer failed due to host aborting the previous control transfer.

USB_DEVICE_CONTROL_TRANSFER_RESULT_SUCCESS - The request was submitted successfully.

Description

This function allows the application to specify the data that would be sent to host in the data stage of a control read transfer. It should be called when the application has received the USB_DEVICE_CONTROL_TRANSFER_EVENT_SETUP_REQUEST event and has identified this setup request as the setup stage of a control read transfer. The Device Layer will generate a USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA_SENT event when the data stage has completed. The function can be called in the Application Control Transfer Event handler or can be called after the application has returned from the control transfer event handler.

Calling this function after returning from the event handler defers the response to the event. This allows the application to prepare the data buffer out of the event handler context, especially if the data buffer to receive the data is not readily available. Note however, that there are timing considerations when responding to the control transfer. Exceeding the response time will cause the host to cancel the control transfer and may cause USB host to reject the device.

Remarks

None.

Preconditions

Client handle should be valid.

Example

```
// The following code example shows an example of how the
// USB_DEVICE_ControlSend() function is called in response to the
// USB_DEVICE_CONTROL_TRANSFER_EVENT_SETUP_REQUEST event to enable a control
// read transfer.

void APP_USBDeviceEventHandler
(
    USB_DEVICE_EVENT event,
    void * pData,
    uintptr_t context
)
{
    USB_SETUP_PACKET * setupPkt;

    switch(event)
    {
        case USB_DEVICE_CONTROL_TRANSFER_EVENT_SETUP_REQUEST:

            setupPkt = (USB_SETUP_PACKET *)pData;

            // Submit a buffer to send 32 bytes in the  control read transfer.
            USB_DEVICE_ControlSend(usbDeviceHandle, data, 32);
            break;

        case USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA_RECEIVED:

            // This means that data in the data stage of the control
            // write transfer has been received. The application can either
            // accept the received data by calling the
            // USB_DEVICE_ControlStatus function with
            // USB_DEVICE_CONTROL_STATUS_OK flag (as shown in this example)
            // or it can reject it by calling the USB_DEVICE_ControlStatus
            // function with USB_DEVICE_CONTROL_STATUS_ERROR flag.

            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA_SENT:

            // This means that data in the data stage of the control
            // read transfer has been sent. The application would typically
            // end the control transfer by calling the
            // USB_DEVICE_ControlStatus function with
            // USB_DEVICE_CONTROL_STATUS_OK flag (as shown in this example).

            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_CONTROL_TRANSFER_EVENT_ABORTED:

            // This means the host has aborted the control transfer. The
            // application can reset its control transfer state machine.
        break;
    }

}
```

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open
data	Pointer to buffer that holds data
length	Size in bytes

Function

USB_DEVICE_CONTROL_TRANSFER_RESULT USB_DEVICE_ControlSend
(
USB_DEVICE_HANDLE usbDeviceHandle,
void * data,
size_t length
)

USB_DEVICE_ControlStatus Function

Initiates status stage of the control transfer.

File

usb_device.h

C

```
USB_DEVICE_CONTROL_TRANSFER_RESULT USB_DEVICE_ControlStatus(USB_DEVICE_HANDLE usbDeviceHandle,
USB_DEVICE_CONTROL_STATUS status);
```

Returns

USB_DEVICE_CONTROL_TRANSFER_RESULT_FAILED - If control transfer failed due to host aborting the previous control transfer.

USB_DEVICE_CONTROL_TRANSFER_RESULT_SUCCESS - The request was submitted successfully.

Description

This function allows the application to complete the status stage of the of an on-going control transfer. The application must call this function when the data stage of the control transfer is complete or when a Setup Request has been received (in case of a zero data stage control transfer). The application can either accept the data stage/setup command or reject it. Calling this function with status set to USB_DEVICE_CONTROL_STATUS_OK will acknowledge the status stage of the control transfer. The control transfer can be stalled by setting the status parameter to USB_DEVICE_CONTROL_STATUS_ERROR.

The function can be called in the Application Control Transfer event handler or can be called after returning from this event handler. Calling this function after returning from the control transfer event handler defers the response to the event. This allows the application to analyze the event response outside the event handler. Note however, that there are timing considerations when responding to the control transfer. Exceeding the response time will cause the host to cancel the control transfer and may cause USB host to reject the device.

The application must be aware of events and associated control transfers that do or do not require data stages. Incorrect usage of the USB_DEVICE_ControlStatus function could cause the device function to be non-compliant.

Remarks

None.

Preconditions

Client handle should be valid. This function should only be called to complete an on-going control transfer.

Example

```
// The following code example shows an example of how the
// USB_DEVICE_ControlStatus() function is called in response to the
// USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA_RECEIVED and
// USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA_SENT event to complete the control
// transfer. Here, the application code acknowledges the status stage of the
// control transfer.

void APP_USBDeviceControlTransferEventHandler
(
    USB_DEVICE_EVENT event,
    void * data,
    uintptr_t context
)
{
    USB_SETUP_PACKET * setupPkt;

    switch(event)
    {
        case USB_DEVICE_CONTROL_TRANSFER_EVENT_SETUP_REQUEST:
```

```
            setupPkt = (USB_SETUP_PACKET *)pData;

            // Submit a buffer to receive 32 bytes in the  control write transfer.
            USB_DEVICE_ControlReceive(usbDeviceHandle, data, 32);
            break;

        case USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA_RECEIVED:

            // This means that data in the data stage of the control
            // write transfer has been received. The application can either
            // accept the received data by calling the
            // USB_DEVICE_ControlStatus function with
            // USB_DEVICE_CONTROL_STATUS_OK flag (as shown in this example)
            // or it can reject it by calling the USB_DEVICE_ControlStatus
            // function with USB_DEVICE_CONTROL_STATUS_ERROR flag.

            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA_SENT:

            // This means that data in the data stage of the control
            // read transfer has been sent. The application would typically
            // end the control transfer by calling the
            // USB_DEVICE_ControlStatus function with
            // USB_DEVICE_CONTROL_STATUS_OK flag (as shown in this example).

            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_CONTROL_TRANSFER_EVENT_ABORTED:

            // This means the host has aborted the control transfer. The
            // application can reset its control transfer state machine.
        break;
    }
}
```

Parameters

Parameters	Description
usbDeviceHandle	USB device handle returned by USB_DEVICE_Open
status	USB_DEVICE_CONTROL_STATUS_OK to acknowledge the status stage. USB_DEVICE_CONTROL_STATUS_ERROR to stall the status stage.

Function

```
USB_DEVICE_CONTROL_TRANSFER_RESULT USB_DEVICE_ControlStatus
(
    USB_DEVICE_HANDLE usbDeviceHandle,
    USB_DEVICE_CONTROL_STATUS status
)
```

g) Data Types and Constants

USB_DEVICE_INDEX_0 Macro

USB device layer index definitions.

File

usb_device.h

C

```
#define USB_DEVICE_INDEX_0 0
```

Description

USB Device Layer Index Numbers

These constants provide USB device layer index definitions.

Remarks

These constants should be used in place of hard-coded numeric literals. These values should be passed into the USB_DEVICE_Initialize and USB_DEVICE_Open routines to identify the device layer instance in use.

USB_DEVICE_INDEX_1 Macro

File

usb_device.h

C

```
#define USB_DEVICE_INDEX_1 1
```

Description

This is macro USB_DEVICE_INDEX_1.

USB_DEVICE_INDEX_2 Macro

File

usb_device.h

C

```
#define USB_DEVICE_INDEX_2 2
```

Description

This is macro USB_DEVICE_INDEX_2.

USB_DEVICE_INDEX_3 Macro

File

usb_device.h

C

```
#define USB_DEVICE_INDEX_3 3
```

Description

This is macro USB_DEVICE_INDEX_3.

USB_DEVICE_INDEX_4 Macro

File

usb_device.h

C

```
#define USB_DEVICE_INDEX_4 4
```

Description

This is macro USB_DEVICE_INDEX_4.

USB_DEVICE_INDEX_5 Macro

File

usb_device.h

C

```
#define USB_DEVICE_INDEX_5 5
```

Description

This is macro USB_DEVICE_INDEX_5.

USB_DEVICE_CONTROL_STATUS Enumeration

USB Device Layer Control Transfer Status Stage flags.

File

usb_device.h

C

```
typedef enum {
  USB_DEVICE_CONTROL_STATUS_OK,
  USB_DEVICE_CONTROL_STATUS_ERROR
} USB_DEVICE_CONTROL_STATUS;
```

Members

Members	Description
USB_DEVICE_CONTROL_STATUS_OK	Using this flag acknowledges the Control transfer. A Zero Length Packet will be transmitted in the status stage of the control transfer.
USB_DEVICE_CONTROL_STATUS_ERROR	Using this flag stalls the control transfer. This flags should be used if the control transfer request needs to be declined.

Description

Control Transfer Status Stage Flags

This enumeration defines the flags to be used with the USB_DEVICE_ControlStatus function.

Remarks

None.

USB_DEVICE_CONTROL_TRANSFER_RESULT Enumeration

Enumerated data type identifying results of a control transfer.

File

usb_device.h

C

```
typedef enum {
  USB_DEVICE_CONTROL_TRANSFER_RESULT_FAILED,
  USB_DEVICE_CONTROL_TRANSFER_RESULT_SUCCESS
} USB_DEVICE_CONTROL_TRANSFER_RESULT;
```

Members

Members	Description
USB_DEVICE_CONTROL_TRANSFER_RESULT_FAILED	Control transfer failed. This could be because the control transfer handle is no more valid since the control transfer was aborted by host by sending a new setup packet
USB_DEVICE_CONTROL_TRANSFER_RESULT_SUCCESS	Control transfer was successful

Description

USB Device Layer Control Transfer Result Enumeration

These enumerated values are the possible return values for control transfer operations. These values are returned by the USB_DEVICE_ControlStatus, USB_DEVICE_ControlSend and the USB_DEVICE_ControlReceive functions.

Remarks

None.

USB_DEVICE_EVENT Enumeration

USB Device Layer Events.

File

usb_device.h

C

```
typedef enum {
  USB_DEVICE_EVENT_RESET,
  USB_DEVICE_EVENT_SUSPENDED,
  USB_DEVICE_EVENT_RESUMED,
  USB_DEVICE_EVENT_ERROR,
  USB_DEVICE_EVENT_SOF,
  USB_DEVICE_EVENT_CONFIGURED,
  USB_DEVICE_EVENT_DECONFIGURED,
  USB_DEVICE_EVENT_CONTROL_TRANSFER_ABORTED,
  USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED,
  USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST,
  USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT,
  USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE,
  USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE,
  USB_DEVICE_EVENT_SET_DESCRIPTOR,
  USB_DEVICE_EVENT_SYNCH_FRAME
} USB_DEVICE_EVENT;
```

Members

Members	Description
USB_DEVICE_EVENT_RESET	USB bus reset occurred. This event is an indication to the application client that device layer has deinitialized all function drivers. The application should not use the function drivers in this state. The pData parameter in the event handler function will be NULL.
USB_DEVICE_EVENT_SUSPENDED	This event is an indication to the application client that device is suspended and it can put the device to sleep mode if required. Power saving routines should not be called in the event handler. The pData parameter in the event handler function will be NULL.
USB_DEVICE_EVENT_RESUMED	This event indicates that device has resumed from suspended state. The pData parameter in the event handler function will be NULL.
USB_DEVICE_EVENT_ERROR	This event is an indication to the application client that an error occurred on the USB bus. The pData parameter in the event handler function will be NULL.
USB_DEVICE_EVENT_SOF	This event is generated at every start of frame detected on the bus. Application client can use this SOF event for general time based house keeping activities. The pData parameter in the event handler function will point to a USB_DEVICE_EVENT_DATA_SOF type that contains the frame number.
USB_DEVICE_EVENT_CONFIGURED	This event is an indication to the application client that device layer has initialized all function drivers and application can set the event handlers for the function drivers. The pData parameter will point to a USB_DEVICE_EVENT_DATA_CONFIGURED data type that contains configuration set by the host.
USB_DEVICE_EVENT_DECONFIGURED	The host has deconfigured the device. This happens when the host sends a Set Configuration request with configuration number 0. The device layer will deinitialize all function drivers and then generate this event.
USB_DEVICE_EVENT_CONTROL_TRANSFER_ABORTED	An on-going control transfer was aborted. The application can use this event to reset its control transfer state machine. The pData parameter in the event handler function will be NULL.
USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	The data stage of a Control write transfer has completed. This event occurs after the application has used the USB_DEVICE_ControlReceive function to receive data in the control transfer. The pData parameter in the event handler function will be NULL.
USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST	A setup packet of a control transfer has been received. The recipient field of the received setup packet is Other. The application can initiate the data stage using the USB_DEVICE_ControlReceive and USB_DEVICE_ControlSend function. It can end the control transfer by calling the USB_DEVICE_ControlStatus function. The pData parameter in the event handler will point to USB_SETUP_PACKET data type.

USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT	The data stage of a Control read transfer has completed. This event occurs after the application has used the USB_DEVICE_ControlSend function to send data in the control transfer. The pData parameter in the event handler function will be NULL.
USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE	This event occurs when a endpoint read transfer scheduled using the USB_DEVICE_EndpointRead function has completed. The pData parameter in the event handler function will be a pointer to a USB_DEVICE_EVENT_DATA_ENDPOINT_READ_COMPLETE data type.
USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE	This event occurs when a endpoint write transfer scheduled using the USB_DEVICE_EndpointWrite function has completed. The pData parameter in the event handler function will be a pointer to a USB_DEVICE_EVENT_DATA_ENDPOINT_WRITE_COMPLETE data type.
USB_DEVICE_EVENT_SET_DESCRIPTOR	A SET_DESCRIPTOR request is received. This event occurs when Host sends a SET_DESCRIPTOR request. The pData parameter in the event handler function will be a pointer to a USB_DEVICE_EVENT_DATA_SET_DESCRIPTOR data type. The application should initiate the data stage using the USB_DEVICE_ControlReceive function. In the PIC32 USB Device Stack, this event is generated if USB_DEVICE_EVENT_ENABLE_SET_DESCRIPTOR is defined in the system configuration.
USB_DEVICE_EVENT_SYNCH_FRAME	A SYNCH_FRAME request is received. This event occurs when Host sends a SYNCH_FRAME request. The pData parameter in the event handler function will be a pointer to a USB_DEVICE_EVENT_DATA_SYNCH_FRAME data type. The application should initiate the data stage using the USB_DEVICE_ControlSend function. In the PIC32 USB Device Stack, this event is generated if USB_DEVICE_EVENT_ENABLE_SYNCH_FRAME is defined in the system configuration.

Description

USB Device Layer Events.

This enumeration lists the possible events that the device layer can generate. The client should register an event handler of the type USB_DEVICE_EVENT_HANDLER to receive device layer events. The contents of pData in the event handler depends on the generated event. Refer to the description of the event for details on data provided along with that event. The events generated are device layer instance specific.

The client will receive control transfers for handling from the device layer, where the recipient field of the Control Transfer Setup packet is set to Other. The client can use the control transfer events and the Device Layer control transfer functions to complete such control transfers.

It is not mandatory for the client application to handle the control transfer event within the event handler. Indeed, it may be possible that the data stage of the control transfer requires extended processing. Because the event handler executes in an interrupt context, it is recommended to keep the processing in the event handler to a minimum. The client application can call the USB_DEVICE_ControlSend, USB_DEVICE_ControlReceive and USB_DEVICE_ControlStatus functions after returning from the event handler, thus deferring the control transfer event handling and responses.

Note that a USB host will typically wait for control transfer response for a finite time duration before timing out and canceling the transfer and associated transactions. Even when deferring response, the application must respond promptly if such timeouts have to be avoided.

The client must use the USB_DEVICE_EventHandlerSet function to register the event handler call back function. The following code example shows the handling of the USB Device Layer Events.

```
USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler
(
    USB_DEVICE_EVENT event,
    void * pData,
    uintptr_t context
)
{
    uint8_t     activeConfiguration;
    uint16_t    frameNumber;
    USB_SPEED   attachSpeed;
    USB_SETUP_PACKET * setupEventData;

    // Handling of each event
    switch(event)
    {
        case USB_DEVICE_EVENT_POWER_DETECTED:

            // This means the device detected a valid VBUS voltage and is
            // attached to the USB. The application can now call
            // USB_DEVICE_Attach() function to enable D+/D- pull up
            // resistors.
            break;
```

```
        case USB_DEVICE_EVENT_POWER_REMOVED:

            // This means the device is not attached to the USB.
            // The application should now call the USB_DEVICE_Detach()
            // function.
            break;

        case USB_DEVICE_EVENT_SUSPENDED:

            // The bus is idle. There was no activity detected.
            // The application can switch to a low power mode after
            // exiting the event handler.
            break;

        case USB_DEVICE_EVENT_SOF:

            // A start of frame was received. This is a periodic event and
            // can be used by the application for timing related activities.
            // pData will point to a USB_DEVICE_EVENT_DATA_SOF type data
            // containing the frame number. In PIC32 USB Device Stack, this
            // event is generated if USB_DEVICE_SOF_EVENT_ENABLE is
            // defined in System Configuration.

            frameNumber = ((USB_DEVICE_EVENT_DATA_SOF *)pData)->frameNumber;
            break;

        case USB_DEVICE_EVENT_RESET :

            // Reset signaling was detected on the bus. The
            // application can find out the attach speed.

            attachedSpeed = USB_DEVICE_ActiveSpeedGet(usbDeviceHandle);
            break;

        case USB_DEVICE_EVENT_DECONFIGURED :

            // This indicates that host has deconfigured the device i.e., it
            // has set the configuration as 0. All function driver instances
            // would have been deinitialized.

            break;

        case USB_DEVICE_EVENT_ERROR :

            // This means an unknown error has occurred on the bus.
            // The application can try detaching and attaching the
            // device again.
            break;

        case USB_DEVICE_EVENT_CONFIGURED :

            // This means that device is configured and the application can
            // start using the device functionality. The application must
            // register function driver event handlersI have one device
            // level event.  The pData parameter will be a pointer to a
            // USB_DEVICE_EVENT_DATA_CONFIGURED data type that contains the
            // active configuration number.

            activeConfiguration = ((USB_DEVICE_EVENT_DATA_CONFIGURED *)pData)->configurationValue;
            break;

        case USB_DEVICE_EVENT_RESUMED:

            // This means that the resume signaling was detected on the
            // bus. The application can bring the device out of power
            // saving mode.

            break;
```

```
        case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST:

            // This means that the setup stage of the control transfer is in
            // progress and a setup packet has been received. The pData
            // parameter will point to a USB_SETUP_PACKET data type The
            // application can process the command and update its control
            // transfer state machine. The application for example could call
            // the USB_DEVICE_ControlReceive function (as shown here) to
            // submit the buffer that would receive data in case of a
            // control read transfer.

            setupPacket = (USB_SETUP_PACKET *)pData;

            // Submit a buffer to receive 32 bytes in the  control write transfer.
            USB_DEVICE_ControlReceive(usbDeviceHandle, data, 32);
            break;

        case USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA_RECEIVED:

            // This means that data in the data stage of the control write
            // transfer has been received. The application can either accept
            // the received data by calling the USB_DEVICE_ControlStatus
            // function with USB_DEVICE_CONTROL_STATUS_OK flag (as shown in
            // this example) or it can reject it by calling the
            // USB_DEVICE_ControlStatus function with
            // USB_DEVICE_CONTROL_STATUS_ERROR flag.

            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_CONTROL_TRANSFER_EVENT_DATA_SENT:

            // This means that data in the data stage of the control
            // read transfer has been sent.

            break;

        case USB_DEVICE_CONTROL_TRANSFER_EVENT_ABORTED:

            // This means the host has aborted the control transfer. The
            // application can reset its control transfer state machine.

            break;

        case USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE:

            // This means schedule endpoint read operation has completed.
            // The application should interpret pData as a pointer to
            // a USB_DEVICE_EVENT_DATA_ENDPOINT_READ_COMPLETE type.

            break;

        case USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE:

            // This means schedule endpoint write operation has completed.
            // The application should interpret pData as a pointer to
            // a USB_DEVICE_EVENT_DATA_ENDPOINT_WRITE_COMPLETE type.

            break;

        case USB_DEVICE_EVENT_SET_DESCRIPTOR:

            // This means the Host has sent a Set Descriptor request. The
            // application should interpret pData as a
            // USB_DEVICE_EVENT_DATA_SET_DESCRIPTOR pointer type containing the
            // details of the Set Descriptor request. In PIC32 USB Device
            // Stack, this event is generated if
            // USB_DEVICE_SET_DESCRIPTOR_EVENT_ENABLE is defined in the
```

```
            // system configuration. The application can use
            // USB_DEVICE_ControlSend, USB_DEVICE_ControlReceive and/or
            // the USB_DEVICE_ControlStatus functions to complete the
            // control transfer.

            break;

        case USB_DEVICE_EVENT_SYNCH_FRAME:

            // This means the host has sent a Sync Frame Request. The
            // application should interpret pData as a
            // USB_DEVICE_EVENT_DATA_SYNCH_FRAME pointer type. In PIC32 USB Device
            // Stack, this event is generated if
            // USB_DEVICE_SYNCH_FRAME_EVENT_ENABLE is defined in the
            // system configuration. The application should respond be
            // sending the 2 byte frame number using the
            // USB_DEVICE_ControlSend function.

            USB_DEVICE_ControlSend(usbDeviceHandle, &frameNumber, 2);
            break;

        default:
            break;
    }

    return USB_DEVICE_EVENT_REPONSE_NONE;
}
```

Remarks

Generation of some events required the definition of configuration macros. Refer to the event specific description for more details.

USB_DEVICE_HANDLE Type

Data type for USB device handle.

File

usb_device.h

C

```
typedef uintptr_t USB_DEVICE_HANDLE;
```

Description

Data type for USB device handle.

The data type of the handle that is returned from USB_DEVICE_Open function.

Remarks

None.

USB_DEVICE_INIT Structure

USB Device Initialization Structure

File

usb_device.h

C

```
typedef struct {
  SYS_MODULE_INIT moduleInit;
  unsigned int usbID;
  bool stopInIdle;
  bool suspendInSleep;
  INT_SOURCE interruptSource;
  INT_SOURCE interruptSourceUSBDma;
  void * endpointTable;
  uint16_t registeredFuncCount;
  USB_DEVICE_FUNCTION_REGISTRATION_TABLE * registeredFunctions;
```

```
  USB_DEVICE_MASTER_DESCRIPTOR * usbMasterDescriptor;
  USB_SPEED deviceSpeed;
  uint16_t queueSizeEndpointRead;
  uint16_t queueSizeEndpointWrite;
  SYS_MODULE_INDEX driverIndex;
  void * usbDriverInterface;
} USB_DEVICE_INIT;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	System module initialization
unsigned int usbID;	Identifies peripheral (PLIB-level) ID. The use of this parameter is • deprecated.
bool stopInIdle;	If true, USB module will stop when CPU enters Idle Mode. The use of this • parameter is deprecated.
bool suspendInSleep;	If true, USB module will suspend when the microcontroller enters sleep • mode. The use of this parameter is deprecated.
INT_SOURCE interruptSource;	Interrupt Source for USB module. The use of this parameter is deprecated.
INT_SOURCE interruptSourceUSBDma;	Interrupt Source for USB DMA module. The use of this parameter is • deprecated.
void * endpointTable;	Pointer to an byte array whose size is USB_DEVICE_ENDPOINT_TABLE_SIZE and who start address is aligned at a 512 bytes address boundary. The • use of this parameter is deprecated.
uint16_t registeredFuncCount;	Number of function drivers registered to this instance of the USB device layer
USB_DEVICE_FUNCTION_REGISTRATION_TABLE * registeredFunctions;	Function driver table registered to this instance of the USB device layer
USB_DEVICE_MASTER_DESCRIPTOR * usbMasterDescriptor;	Pointer to USB Descriptor structure
USB_SPEED deviceSpeed;	Specify the speed at which this device will attempt to connect to the host. PIC32MX devices support Full Speed only. PIC32MZ devices support Full Speed and High Speed. Selecting High Speed will allow the device to work at both Full Speed and High Speed.
uint16_t queueSizeEndpointRead;	Enter Endpoint Read queue size. Application can place this many Endpoint Read requests in the queue. Each Endpoint Read queue element would consume 36 Bytes of RAM. Value of this field should be at least 1. This is applicable only for applications using Endpoint Read/Write functions like USB Vendor Device.
uint16_t queueSizeEndpointWrite;	Enter Endpoint Write queue size. Application can place this many Endpoint Read requests in the queue. Each Endpoint Write queue element would consume 36 Bytes of RAM. Value of this field should be at least 1. This is applicable only for applications using Endpoint Read/Write functions like USB Vendor Device.
SYS_MODULE_INDEX driverIndex;	System Module Index of the driver that this device layer should open
void * usbDriverInterface;	Interface to the USB Driver that this Device Layer should use

Description

USB Device Initialization Structure

This data type defines the USB Device Initialization data structure. A data structure of this type should be initialized and provided to USB_DEVICE_Initialize.

Remarks

This type is specific to the PIC32 implementation of the USB Device Stack API.

USB_DEVICE_POWER_STATE Enumeration

Enumerated data type that identifies if the device is self powered or bus powered .

File

usb_device.h

C

```
typedef enum {
  USB_DEVICE_POWER_STATE_BUS_POWERED,
  USB_DEVICE_POWER_STATE_SELF_POWERED
} USB_DEVICE_POWER_STATE;
```

Members

Members	Description
USB_DEVICE_POWER_STATE_BUS_POWERED	Device is bus powered
USB_DEVICE_POWER_STATE_SELF_POWERED	Device is self powered

Description

Device Power state

This enumeration defines the possible power states of the device. The application specifies this state to the device layer (through the USB_DEVICE_PowerStateSet function) to let the device layer know if this USB Device is presently bus or self powered.

Remarks

None.

USB_DEVICE_REMOTE_WAKEUP_STATUS Enumeration

Enumerated data type that identifies if the remote wakeup status of the device.

File

usb_device.h

C

```
typedef enum {
  USB_DEVICE_REMOTE_WAKEUP_DISABLED,
  USB_DEVICE_REMOTE_WAKEUP_ENABLED
} USB_DEVICE_REMOTE_WAKEUP_STATUS;
```

Members

Members	Description
USB_DEVICE_REMOTE_WAKEUP_DISABLED	Remote wakeup is disabled
USB_DEVICE_REMOTE_WAKEUP_ENABLED	Remote wakeup is enabled

Description

Remote Wakeup Status

This enumeration defines the possible status of the remote wake up capability. These values are returned by the USB_DEVICE_RemoteWakeupStatusGet function.

Remarks

None.

USB_DEVICE_EVENT_DATA_CONFIGURED Structure

USB Device Set Configuration Event Data type.

File

usb_device.h

C

```
typedef struct {
  uint8_t configurationValue;
} USB_DEVICE_EVENT_DATA_CONFIGURED;
```

Members

Members	Description
uint8_t configurationValue;	The configuration that was set

Description

USB Device Set Configuration Event Data type.

This data type defines the type of data that is returned by the Device Layer along with the USB_DEVICE_EVENT_CONFIGURED event.

Remarks

None.

USB_DEVICE_HANDLE_INVALID Macro

Constant that defines the value of an Invalid Device Handle.

File

usb_device.h

C

```
#define USB_DEVICE_HANDLE_INVALID
```

Description

USB Device Layer Invalid Handle

This constant is returned by the USB_DEVICE_Open() function when the function fails.

Remarks

None.

USB_DEVICE_EVENT_RESPONSE_NONE Macro

Device Layer Event Handler Function Response Type.

File

usb_device.h

C

```
#define USB_DEVICE_EVENT_RESPONSE_NONE
```

Description

Device Layer Event Handler Function Response Type None.

This is the definition of the Device Layer Event Handler Response Type None.

Remarks

Intentionally defined to be empty.

USB_DEVICE_CLIENT_STATUS Enumeration

Enumerated data type that identifies the USB Device Layer Client Status.

File

usb_device.h

C

```
typedef enum {
  USB_DEVICE_CLIENT_STATUS_CLOSED,
  USB_DEVICE_CLIENT_STATUS_READY
} USB_DEVICE_CLIENT_STATUS;
```

Members

Members	Description
USB_DEVICE_CLIENT_STATUS_CLOSED	Client is closed or the specified handle is invalid
USB_DEVICE_CLIENT_STATUS_READY	Client is ready

Description

USB Device Layer Client Status

This enumeration defines the possible status of the USB Device Layer Client. It is returned by the USB_DEVICE_ClientStatusGet function.

Remarks

None.

USB_DEVICE_CONFIGURATION_DESCRIPTORS_TABLE Type

Pointer to an array that contains pointer to configuration descriptors.

File

usb_device.h

C

```
typedef const uint8_t * const USB_DEVICE_CONFIGURATION_DESCRIPTORS_TABLE;
```

Description

Configuration descriptors pointer

This type defines a pointer to an array that contains pointers to configuration descriptors. This data type is used in USB_DEVICE_MASTER_DESCRIPTOR data type to point to the table of configuration descriptors.

Remarks

This type is specific to the PIC32 implementation of the USB Device Stack API.

USB_DEVICE_EVENT_HANDLER Type

USB Device Layer Event Handler Function Pointer Type

File

usb_device.h

C

```
typedef USB_DEVICE_EVENT_RESPONSE (* USB_DEVICE_EVENT_HANDLER)(USB_DEVICE_EVENT event, void * eventData,
uintptr_t context);
```

Description

USB Device Layer Event Handler Function Pointer Type

This data type defines the required function signature of the USB Device Layer Event handling callback function. The application must register a pointer to a Device Layer Event handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event call backs from the Device Layer. The Device Layer will invoke this function with event relevant parameters. The description of the event handler function parameters is given here.

event - Type of event generated.

pData - This parameter should be type cast to an event specific pointer type based on the event that has occurred. Refer to the USB_DEVICE_EVENT enumeration description for more details.

context - Value identifying the context of the application that was registered along with the event handling function.

Remarks

None.

USB_DEVICE_EVENT_RESPONSE Type

Device Layer Event Handler function return type.

File

usb_device.h

C

```
typedef void USB_DEVICE_EVENT_RESPONSE;
```

Description

Device Layer Event Handler function return type.

This data type defines the return type of the Device Layer event handler function.

Remarks

None.

USB_DEVICE_FUNCTION_REGISTRATION_TABLE Structure

USB Device Function Registration Structure

File

usb_device.h

C

```
typedef struct {
  USB_SPEED speed;
  uint8_t configurationValue;
  uint8_t interfaceNumber;
  uint8_t numberOfInterfaces;
  uintptr_t funcDriverIndex;
  void * funcDriverInit;
  void * driver;
} USB_DEVICE_FUNCTION_REGISTRATION_TABLE;
```

Members

Members	Description
USB_SPEED speed;	Type of speed (high, full or low speed)
uint8_t configurationValue;	Configuration Value to which the function driver has to be tied
uint8_t interfaceNumber;	Interface number to which this function driver has to be tied
uint8_t numberOfInterfaces;	Number of interfaces used by the function
uintptr_t funcDriverIndex;	Function driver instance index
void * funcDriverInit;	Pointer to a structure that contains function driver initialization data
void * driver;	Pointer to a standard structure that exposes function driver APIs to USB device layer

Description

USB Device Function Registration Structure

This data type defines the USB Device Function Registration Structure. A table containing entries for each function driver instance should be registered with device layer.

Remarks

This type is specific to the PIC32 implementation of the USB Device Stack API.

USB_DEVICE_MASTER_DESCRIPTOR Structure

USB Device Master Descriptor Structure.

File

usb_device.h

C

```
typedef struct {
  const USB_DEVICE_DESCRIPTOR * deviceDescriptor;
  uint8_t configDescriptorCount;
  USB_DEVICE_CONFIGURATION_DESCRIPTORS_TABLE * configDescriptorTable;
  const USB_DEVICE_DESCRIPTOR * highSpeedDeviceDescriptor;
  uint8_t highSpeedConfigDescriptorCount;
  USB_DEVICE_CONFIGURATION_DESCRIPTORS_TABLE * highSpeedConfigDescriptorTable;
  uint8_t stringDescCount;
  USB_DEVICE_STRING_DESCRIPTORS_TABLE * stringDescriptorTable;
  const USB_DEVICE_QUALIFIER * fullSpeedDeviceQualifier;
  const USB_DEVICE_QUALIFIER * highSpeedDeviceQualifier;
  const uint8_t * bosDescriptor;
} USB_DEVICE_MASTER_DESCRIPTOR;
```

Members

Members	Description
const USB_DEVICE_DESCRIPTOR * deviceDescriptor;	Pointer to standard device descriptor (for low/full speed)
uint8_t configDescriptorCount;	Total number configurations available (for low/full speed)

USB_DEVICE_CONFIGURATION_DESCRIPTORS_TABLE * configDescriptorTable;	Pointer to array of configurations descriptor pointers (for low/full speed)
const USB_DEVICE_DESCRIPTOR * highSpeedDeviceDescriptor;	Pointer to array of high speed standard Device descriptor. Assign this to NULL if not supported.
uint8_t highSpeedConfigDescriptorCount;	Total number of high speed configurations available. Set this to zero if not supported
USB_DEVICE_CONFIGURATION_DESCRIPTORS_TABLE * highSpeedConfigDescriptorTable;	Pointer to array of high speed configurations descriptor pointers. Set this to NULL if not supported
uint8_t stringDescCount;	Total number of string descriptors available (common to all speeds)
USB_DEVICE_STRING_DESCRIPTORS_TABLE * stringDescriptorTable;	Pointer to array of string Descriptor pointers (common to all speeds)
const USB_DEVICE_QUALIFIER * fullSpeedDeviceQualifier;	Pointer to full speed device_qualifier descriptor. Device responds with this descriptor when it is operating at high speed
const USB_DEVICE_QUALIFIER * highSpeedDeviceQualifier;	Pointer to high speed device_qualifier descriptor. Device responds with this descriptor when it is operating at full speed
const uint8_t * bosDescriptor;	Pointer to BOS descriptor for this Device. Device responds with this descriptor when Host sends a GET_DESCRIPTOR request for BOS descriptor

Description

USB Device Master Descriptor Structure.

This data type defines the structure of the USB Device Master Descriptor. The application must provide such a structure for each instance of the device layer.

Remarks

This type is specific to the PIC32 implementation of the USB Device Stack API.

USB_DEVICE_STRING_DESCRIPTORS_TABLE Type

Pointer to an array that contains pointer to string descriptors.

File

usb_device.h

C

```
typedef const uint8_t * const USB_DEVICE_STRING_DESCRIPTORS_TABLE;
```

Description

String Descriptors Pointer

This type defines a pointer to an array that contains pointers to string descriptors. This data type is used in USB_DEVICE_MASTER_DESCRIPTOR data type to point to the table of string descriptors.

Remarks

This type is specific to the PIC32 implementation of the USB Device Stack API.

USB_DEVICE_EVENT_DATA_ENDPOINT_READ_COMPLETE Structure

USB Device Layer Endpoint Read and Write Complete Event Data type.

File

usb_device.h

C

```
typedef struct {
  USB_DEVICE_TRANSFER_HANDLE transferHandle;
  size_t length;
  USB_DEVICE_RESULT status;
} USB_DEVICE_EVENT_DATA_ENDPOINT_READ_COMPLETE, USB_DEVICE_EVENT_DATA_ENDPOINT_WRITE_COMPLETE;
```

Members

Members	Description
USB_DEVICE_TRANSFER_HANDLE transferHandle;	Transfer Handle

size_t length;	Size of transferred data
USB_DEVICE_RESULT status;	Completion status of the transfer

Description

USB Device Layer Endpoint Read and Write Complete Event Data type.

This data type defines the type of data that is returned by the Device Layer along with the USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE and USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE events.

Remarks

None.

USB_DEVICE_EVENT_DATA_ENDPOINT_WRITE_COMPLETE Structure

USB Device Layer Endpoint Read and Write Complete Event Data type.

File

usb_device.h

C

```
typedef struct {
  USB_DEVICE_TRANSFER_HANDLE transferHandle;
  size_t length;
  USB_DEVICE_RESULT status;
} USB_DEVICE_EVENT_DATA_ENDPOINT_READ_COMPLETE, USB_DEVICE_EVENT_DATA_ENDPOINT_WRITE_COMPLETE;
```

Members

Members	Description
USB_DEVICE_TRANSFER_HANDLE transferHandle;	Transfer Handle
size_t length;	Size of transferred data
USB_DEVICE_RESULT status;	Completion status of the transfer

Description

USB Device Layer Endpoint Read and Write Complete Event Data type.

This data type defines the type of data that is returned by the Device Layer along with the USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE and USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE events.

Remarks

None.

USB_DEVICE_EVENT_DATA_SET_DESCRIPTOR Structure

USB Device Set Descriptor Event Data type.

File

usb_device.h

C

```
typedef struct {
  uint8_t descriptorIndex;
  uint8_t descriptorType;
  uint16_t languageID;
  uint16_t descriptorLength;
} USB_DEVICE_EVENT_DATA_SET_DESCRIPTOR;
```

Description

USB Device Set Descriptor Event Data type.

This data type defines the type of data that is returned by the Device Layer along with the USB_DEVICE_EVENT_SET_DESCRIPTOR event.

Remarks

None.

USB_DEVICE_EVENT_DATA_SOF Structure

USB Device Start Of Frame Event Data Type

File

usb_device.h

C

```
typedef struct {
  uint16_t frameNumber;
} USB_DEVICE_EVENT_DATA_SOF;
```

Members

Members	Description
uint16_t frameNumber;	The Start Of Frame number

Description

USB Device Start Of Frame Event Data Type

This data type defines the type of data that is returned by the Device Layer along with the USB_DEVICE_EVENT_SOF event.

Remarks

None.

USB_DEVICE_EVENT_DATA_SYNCH_FRAME Structure

USB Device Synch Frame Event Data type.

File

usb_device.h

C

```
typedef struct {
  USB_ENDPOINT_ADDRESS endpoint;
} USB_DEVICE_EVENT_DATA_SYNCH_FRAME;
```

Members

Members	Description
USB_ENDPOINT_ADDRESS endpoint;	Endpoint for which the Synch Frame number is requested

Description

USB Device Synch Frame Event Data type.

This data type defines the type of data that is returned by the Device Layer along with the USB_DEVICE_EVENT_SYNCH_FRAME event.

Remarks

None.

USB_DEVICE_RESULT Enumeration

USB Device Layer Results Enumeration

File

usb_device.h

C

```
typedef enum {
  USB_DEVICE_RESULT_ERROR_TRANSFER_QUEUE_FULL,
  USB_DEVICE_RESULT_OK,
  USB_DEVICE_RESULT_ERROR_ENDPOINT_NOT_CONFIGURED,
  USB_DEVICE_RESULT_ERROR_ENDPOINT_INVALID,
  USB_DEVICE_RESULT_ERROR_PARAMETER_INVALID,
  USB_DEVICE_RESULT_ERROR_DEVICE_HANDLE_INVALID,
  USB_DEVICE_RESULT_ERROR_ENDPOINT_HALTED,
```

```
  USB_DEVICE_RESULT_ERROR_TERMINATED_BY_HOST,
  USB_DEVICE_RESULT_ERROR
} USB_DEVICE_RESULT;
```

Members

Members	Description
USB_DEVICE_RESULT_ERROR_TRANSFER_QUEUE_FULL	Queue is full
USB_DEVICE_RESULT_OK	No Error
USB_DEVICE_RESULT_ERROR_ENDPOINT_NOT_CONFIGURED	Endpoint not configured
USB_DEVICE_RESULT_ERROR_ENDPOINT_INVALID	Endpoint not provisioned in the system
USB_DEVICE_RESULT_ERROR_PARAMETER_INVALID	One or more parameter/s of the function is invalid
USB_DEVICE_RESULT_ERROR_DEVICE_HANDLE_INVALID	Device Handle passed to the function is invalid
USB_DEVICE_RESULT_ERROR_ENDPOINT_HALTED	Transfer terminated because host halted the endpoint
USB_DEVICE_RESULT_ERROR_TERMINATED_BY_HOST	Transfer terminated by host because of a stall clear
USB_DEVICE_RESULT_ERROR	An unspecified error has occurred

Description

USB Device Result Enumeration

This enumeration lists the possible USB Device Endpoint operation results. These values are returned by USB Device Endpoint functions.

Remarks

None.

USB_DEVICE_TRANSFER_FLAGS Enumeration

Enumerated data type that identifies the USB Device Layer Transfer Flags.

File

usb_device.h

C

```
typedef enum {
  USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE,
  USB_DEVICE_TRANSFER_FLAGS_MORE_DATA_PENDING
} USB_DEVICE_TRANSFER_FLAGS;
```

Members

Members	Description
USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE	This flag indicates there is no further data to be sent in this transfer and that the transfer should end. If the size of the transfer is a multiple of the maximum packet size for related endpoint configuration, the device layer will send a zero length packet to indicate end of the transfer to the host.
USB_DEVICE_TRANSFER_FLAGS_MORE_DATA_PENDING	This flag indicates there is more data to be sent in this transfer. If the size of the transfer is a multiple of the maximum packet size for the related endpoint configuration, the device layer will not send a zero length packet. This flags should not be specified if the size of the transfer is is not a multiple of the maximum packet size or if the transfer is less than maximum packet size.

Description

USB Device Layer Transfer Flags

This enumeration defines the possible USB Device Layer Transfer Flags. These flags are specified in USB_DEVICE_EndpointWrite() function to specify the handling of the transfer. Please refer to the description of the USB_DEVICE_EndpointWrite function for examples.

Remarks

None.

USB_DEVICE_TRANSFER_HANDLE Type

Data type for USB Device Endpoint Data Transfer Handle.

File

usb_device.h

C

```
typedef uintptr_t USB_DEVICE_TRANSFER_HANDLE;
```

Description

Data type for USB Device Endpoint Data Transfer Handle.

The data type of the handle that is returned by the USB_DEVICE_EndpointRead() and USB_DEVICE_EndpointWrite() functions.

Remarks

None.

USB_DEVICE_TRANSFER_HANDLE_INVALID Macro

Constant that defines the value of an Invalid Device Endpoint Data Transfer Handle.

File

usb_device.h

C

```
#define USB_DEVICE_TRANSFER_HANDLE_INVALID
```

Description

USB Device Layer Invalid Endpoint Data Transfer Handle

This constant defines the value that is returned by the USB_DEVICE_EndpointRead() and USB_DEVICE_EndpointWrite() functions, as a transfer handle, when the function is not successful.

Remarks

None.

USB_DEVICE_IRP Structure

This structure defines the USB Device Mode IRP data structure.

File

usb_common.h

C

```
typedef struct _USB_DEVICE_IRP {
  void * data;
  unsigned int size;
  USB_DEVICE_IRP_STATUS status;
  void (* callback)(struct _USB_DEVICE_IRP * irp);
  USB_DEVICE_IRP_FLAG flags;
  uintptr_t userData;
  uint32_t privateData[3];
} USB_DEVICE_IRP;
```

Members

Members	Description
void * data;	Pointer to the data buffer
unsigned int size;	Size of the data buffer
USB_DEVICE_IRP_STATUS status;	Status of the IRP
void (* callback)(struct _USB_DEVICE_IRP * irp);	IRP Callback. If this is NULL, • then there is no callback generated
USB_DEVICE_IRP_FLAG flags;	Request specific flags
uintptr_t userData;	User data
uint32_t privateData[3];	The following members should not be modified by the client

Description

USB Device Mode I/O Request Packet

This structure defines the USB Device Mode IRP data structure.

Remarks

None.

USB_DEVICE_IRP_FLAG Enumeration

USB Device IRP flags enumeration

File

usb_common.h

C

```
typedef enum {
  USB_DEVICE_IRP_FLAG_DATA_COMPLETE = 0x1,
  USB_DEVICE_IRP_FLAG_DATA_PENDING = 0x2
} USB_DEVICE_IRP_FLAG;
```

Members

Members	Description
USB_DEVICE_IRP_FLAG_DATA_COMPLETE = 0x1	Using this flag indicates that there is no • more data pending in the IRP. When data moves • from device to host, and if the IRP size is • a multiple of endpoint size, specifying this • flag sends the ZLP. The size is not a • multiple of endpoint size, no ZLP will be sent.
USB_DEVICE_IRP_FLAG_DATA_PENDING = 0x2	In case of data moving from device to host, and if the • size parameter of the IRP is an exact multiple of the • endpoint maximum packet size, specifying this flag, does • send the ZLP. If the size is less than endpoint size, • specifying this flag will return an error. If the size • is more than endpoint size but not a multiple, only • endpoint multiple size of data is sent.

Description

USB Device IRP Flags

This enumeration defines the possible flags that can be specified while adding the IRP.

Remarks

Not all flags are applicable in all conditions. Refer to API documentation for more details

USB_DEVICE_IRP_STATUS Enumeration

Enumerates the possible status options of USB Device IRP.

File

usb_common.h

C

```
typedef enum {
  USB_DEVICE_IRP_STATUS_TERMINATED_BY_HOST = -4,
  USB_DEVICE_IRP_STATUS_ABORTED_ENDPOINT_HALT = -3,
  USB_DEVICE_IRP_STATUS_ABORTED = -2,
  USB_DEVICE_IRP_STATUS_ERROR = -1,
  USB_DEVICE_IRP_STATUS_COMPLETED = 0,
  USB_DEVICE_IRP_STATUS_COMPLETED_SHORT = 1,
  USB_DEVICE_IRP_STATUS_SETUP = 2,
  USB_DEVICE_IRP_STATUS_PENDING = 3,
  USB_DEVICE_IRP_STATUS_IN_PROGRESS = 4
} USB_DEVICE_IRP_STATUS;
```

Members

Members	Description
USB_DEVICE_IRP_STATUS_TERMINATED_BY_HOST = -4	The IRP was aborted because the host cleared the stall on the endpoint
USB_DEVICE_IRP_STATUS_ABORTED_ENDPOINT_HALT = -3	IRP was aborted because the endpoint halted
USB_DEVICE_IRP_STATUS_ABORTED = -2	USB Device IRP was aborted by the function driver
USB_DEVICE_IRP_STATUS_ERROR = -1	An error occurred on the bus when the IRP was being • processed
USB_DEVICE_IRP_STATUS_COMPLETED = 0	The IRP was completed
USB_DEVICE_IRP_STATUS_COMPLETED_SHORT = 1	The IRP was completed but the amount of • data received was less than the requested • size
USB_DEVICE_IRP_STATUS_SETUP = 2	The IRP was completed and the received • token was a SETUP token. This is applicable • to IRP scheduled on CONTROL endpoint
USB_DEVICE_IRP_STATUS_PENDING = 3	The IRP is pending in the queue
USB_DEVICE_IRP_STATUS_IN_PROGRESS = 4	The IRP is currently being processed

Description

USB Device IRP Status Enumeration

Enumerates the possible status options of USB Device IRP.

Remarks

The application that schedules the IRP can check the status member of the USB Device IRP at any time to obtain current status of the IRP.

USB_ENDPOINT Type

Defines a type to store Endpoint and Direction. The MSB defines the direction. The lower 4 bits defines the endpoint.

File

usb_common.h

C

```
typedef uint8_t USB_ENDPOINT;
```

Description

USB Endpoint and Direction Type

Defines a type to store Endpoint and Direction. The MSB defines the direction. The lower 4 bits defines the endpoint.

Remarks

None.

USB_ERROR Enumeration

Enumeration of all possible error codes that are returned by various components functions in the USB Stack.

File

usb_common.h

C

```
typedef enum {
  USB_ERROR_IRP_QUEUE_FULL = SCHAR_MIN,
  USB_ERROR_OSAL_FUNCTION,
  USB_ERROR_IRP_SIZE_INVALID,
  USB_ERROR_PARAMETER_INVALID,
  USB_ERROR_DEVICE_ENDPOINT_INVALID,
  USB_ERROR_DEVICE_IRP_IN_USE,
  USB_ERROR_CLIENT_NOT_READY,
  USB_ERROR_IRP_OBJECTS_UNAVAILABLE,
  USB_ERROR_DEVICE_FUNCTION_INSTANCE_INVALID,
```

```
  USB_ERROR_DEVICE_FUNCTION_NOT_CONFIGURED,
  USB_ERROR_ENDPOINT_NOT_CONFIGURED,
  USB_ERROR_DEVICE_CONTROL_TRANSFER_FAILED,
  USB_ERROR_HOST_DEVICE_INSTANCE_INVALID,
  USB_ERROR_HOST_DRIVER_NOT_READY,
  USB_ERROR_HOST_DRIVER_NOT_FOUND,
  USB_ERROR_HOST_ENDPOINT_INVALID,
  USB_ERROR_HOST_PIPE_INVALID,
  USB_ERROR_HOST_ARGUMENTS_INVALID,
  USB_ERROR_HOST_HEADERSIZE_INVALID,
  USB_ERROR_HOST_MAX_INTERFACES_INVALID,
  USB_ERROR_HOST_ENDPOINT_DESC_SIZE_INVALID,
  USB_ERROR_HOST_DESCRIPTOR_INVALID,
  USB_ERROR_HOST_MAX_ENDPOINT_INVALID,
  USB_ERROR_HOST_ALT_SETTING_INVALID,
  USB_ERROR_HOST_BUSY,
  USB_HOST_OBJ_INVALID,
  USB_ERROR_HOST_POINTER_INVALID,
  USB_ERROR_HOST_ENDPOINT_NOT_FOUND,
  USB_ERROR_HOST_DRIVER_INSTANCE_INVALID,
  USB_ERROR_HOST_INTERFACE_NOT_FOUND,
  USB_ERROR_ENDPOINT_HALTED,
  USB_ERROR_TRANSFER_TERMINATED_BY_HOST,
  USB_ERROR_NONE = 0
} USB_ERROR;
```

Members

Members	Description
USB_ERROR_IRP_QUEUE_FULL = SCHAR_MIN	IRP Queue Full Error
USB_ERROR_OSAL_FUNCTION	OSAL Function fails
USB_ERROR_IRP_SIZE_INVALID	IRP Size parameter invalid
USB_ERROR_PARAMETER_INVALID	Some function parameter was not valid
USB_ERROR_DEVICE_ENDPOINT_INVALID	Device endpoint is not valid
USB_ERROR_DEVICE_IRP_IN_USE	IRP is already in use
USB_ERROR_CLIENT_NOT_READY	Client is not ready
USB_ERROR_IRP_OBJECTS_UNAVAILABLE	Free IRP object unavailable
USB_ERROR_DEVICE_FUNCTION_INSTANCE_INVALID	Function Driver instance was not provisioned
USB_ERROR_DEVICE_FUNCTION_NOT_CONFIGURED	Function Driver instance is not configured
USB_ERROR_ENDPOINT_NOT_CONFIGURED	Endpoint is not configured
USB_ERROR_DEVICE_CONTROL_TRANSFER_FAILED	Device Control Transfer Failed
USB_ERROR_HOST_DEVICE_INSTANCE_INVALID	Host device instance invalid
USB_ERROR_HOST_DRIVER_NOT_READY	Host driver not ready for communication
USB_ERROR_HOST_DRIVER_NOT_FOUND	Host driver not found
USB_ERROR_HOST_ENDPOINT_INVALID	Host endpoint invalid
USB_ERROR_HOST_PIPE_INVALID	Host pipe invalid
USB_ERROR_HOST_ARGUMENTS_INVALID	Invalid arguments
USB_ERROR_HOST_HEADERSIZE_INVALID	Header size invalid
USB_ERROR_HOST_MAX_INTERFACES_INVALID	Max interface Number
USB_ERROR_HOST_ENDPOINT_DESC_SIZE_INVALID	Endpoint descriptor size is invalid
USB_ERROR_HOST_DESCRIPTOR_INVALID	Invalid Descriptor
USB_ERROR_HOST_MAX_ENDPOINT_INVALID	Invalid number of endpoints
USB_ERROR_HOST_ALT_SETTING_INVALID	Host alternate setting is invalid
USB_ERROR_HOST_BUSY	Host is busy
USB_HOST_OBJ_INVALID	USB host invalid
USB_ERROR_HOST_POINTER_INVALID	Pointer is invalid
USB_ERROR_HOST_ENDPOINT_NOT_FOUND	Could not find endpoint
USB_ERROR_HOST_DRIVER_INSTANCE_INVALID	Driver Instance Invalid
USB_ERROR_HOST_INTERFACE_NOT_FOUND	Could not find endpoint
USB_ERROR_ENDPOINT_HALTED	Transfer terminated because endpoint was halted
USB_ERROR_TRANSFER_TERMINATED_BY_HOST	Transfer terminated by host because of a stall clear
USB_ERROR_NONE = 0	No Error, Operation was successful

Description

USB Error Codes

Enumeration of all possible error codes that are returned by various components functions in the USB Stack.

Remarks

None.

USB_HOST_IRP Structure

This structure defines the USB Host Mode IRP data structure.

File

usb_common.h

C

```
typedef struct _USB_HOST_IRP {
  void * setup;
  void * data;
  unsigned int size;
  USB_HOST_IRP_STATUS status;
  USB_HOST_IRP_FLAG flags;
  uintptr_t userData;
  void (* callback)(struct _USB_HOST_IRP * irp);
  uintptr_t privateData[7];
} USB_HOST_IRP;
```

Members

Members	Description
void * setup;	Points to the 8 byte setup command • packet in case this is a IRP is • scheduled on a CONTROL pipe. Should • be NULL otherwise
void * data;	Pointer to data buffer
unsigned int size;	Size of the data buffer
USB_HOST_IRP_STATUS status;	Status of the IRP
USB_HOST_IRP_FLAG flags;	Request specific flags
uintptr_t userData;	User data
void (* callback)(struct _USB_HOST_IRP * irp);	Pointer to function to be called • when IRP is terminated. Can be • NULL, in which case the function • will not be called.
uintptr_t privateData[7];	These members of the IRP should not be modified by client

Description

USB Host Mode I/O Request Packet

This structure defines the USB Host Mode IRP data structure.

Remarks

None.

USB_HOST_IRP_FLAG Enumeration

USB Host IRP flags enumeration

File

usb_common.h

C

```
typedef enum {
  USB_HOST_IRP_FLAG_NONE = 0,
  USB_HOST_IRP_FLAG_SEND_ZLP,
```

```
  USB_HOST_IRP_WAIT_FOR_ZLP
} USB_HOST_IRP_FLAG;
```

Members

Members	Description
USB_HOST_IRP_FLAG_NONE = 0	Does not do anything
USB_HOST_IRP_FLAG_SEND_ZLP	In case of data moving from host to device, and if the • size parameter of the IRP is an exact multiple of the • endpoint maximum packet size, specifying this flag sends • a Zero Length Packet before the IRP is completed.
USB_HOST_IRP_WAIT_FOR_ZLP	In case of data moving device to host, and if the • size parameter of the IRP is an exact multiple of • the endpoint maximum packet size, specifying this • flag will cause the IRP to completed only when the • a ZLP was requested and acknowledged and the amount • of data was a multiple of endpoint maximum packet size.

Description

USB Host IRP Flags

This enumeration defines the possible flags that can be specified while adding the IRP.

Remarks

Not all flags are applicable in all conditions. Refer to API documentation for more details

USB_HOST_IRP_STATUS Enumeration

Enumerates the possible status options of USB Host IRP.

File

usb_common.h

C

```
typedef enum _USB_HOST_IRP_STATUS {
  USB_HOST_IRP_STATUS_ERROR_UNKNOWN = -6,
  USB_HOST_IRP_STATUS_ABORTED = -5,
  USB_HOST_IRP_STATUS_ERROR_BUS = -4,
  USB_HOST_IRP_STATUS_ERROR_DATA = -3,
  USB_HOST_IRP_STATUS_ERROR_NAK_TIMEOUT = -2,
  USB_HOST_IRP_STATUS_ERROR_STALL = -1,
  USB_HOST_IRP_STATUS_COMPLETED = 0,
  USB_HOST_IRP_STATUS_COMPLETED_SHORT = 1,
  USB_HOST_IRP_STATUS_PENDING = 2,
  USB_HOST_IRP_STATUS_IN_PROGRESS = 3
} USB_HOST_IRP_STATUS;
```

Members

Members	Description
USB_HOST_IRP_STATUS_ERROR_UNKNOWN = -6	IRP was terminated due to an unknown error
USB_HOST_IRP_STATUS_ABORTED = -5	IRP was terminated by the application
USB_HOST_IRP_STATUS_ERROR_BUS = -4	IRP was terminated due to a bus error
USB_HOST_IRP_STATUS_ERROR_DATA = -3	IRP was terminated due to data error
USB_HOST_IRP_STATUS_ERROR_NAK_TIMEOUT = -2	IRP was terminated because of a NAK timeout
USB_HOST_IRP_STATUS_ERROR_STALL = -1	IRP was terminated because of a STALL
USB_HOST_IRP_STATUS_COMPLETED = 0	IRP has been completed
USB_HOST_IRP_STATUS_COMPLETED_SHORT = 1	IRP has been completed but the • amount of data processed was less • than requested.
USB_HOST_IRP_STATUS_PENDING = 2	IRP is waiting in queue
USB_HOST_IRP_STATUS_IN_PROGRESS = 3	IRP is currently being processed

Description

USB Host IRP Status Enumeration

Enumerates the possible status options of USB Host IRP.

Remarks

The application that schedules the IRP can check the status member of the USB Host IRP at any time to obtain current status of the IRP.

USB_SPEED Enumeration

Provides enumeration of USB 2.0 speeds.

File

usb_common.h

C

```
typedef enum {
  USB_SPEED_ERROR = 0,
  USB_SPEED_HIGH = 1,
  USB_SPEED_FULL = 2,
  USB_SPEED_LOW = 3
} USB_SPEED;
```

Members

Members	Description
USB_SPEED_ERROR = 0	Error in obtaining USB module speed
USB_SPEED_HIGH = 1	USB module is at high speed
USB_SPEED_FULL = 2	USB module is at full speed
USB_SPEED_LOW = 3	USB module is at low speed

Description

USB 2.0 Speeds Enumeration

Provides enumeration of USB 2.0 speeds.

Remarks

None.

USB_ENDPOINT_AND_DIRECTION Macro

This macro helps in setting up the USB_ENDPOINT type.

File

usb_common.h

C

```
#define USB_ENDPOINT_AND_DIRECTION(direction, endpoint) ((uint8_t)((direction << 7) | endpoint))
```

Description

USB Endpoint and Direction helper macro

This macro helps in setting up the USB_ENDPOINT type. Here x is the direction and can be either USB_DATA_DIRECTION_HOST_TO_DEVICE or USB_DATA_DIRECTION_DEVICE_TO_HOST. y is the endpoint.

Remarks

None.

USB_DATA_DIRECTION Enumeration

Defines the communication direction

File

usb_common.h

C

```
typedef enum {
  USB_DATA_DIRECTION_DEVICE_TO_HOST = 1,
  USB_DATA_DIRECTION_HOST_TO_DEVICE = 0
} USB_DATA_DIRECTION;
```

Members

Members	Description
USB_DATA_DIRECTION_DEVICE_TO_HOST = 1	Data moves from device to host
USB_DATA_DIRECTION_HOST_TO_DEVICE = 0	Data moves from host to device

Description

USB Communication direction definitions

This definitions define the communication direction and can be used to specify direction while using the DRV_USB_ENDPOINT type.

Remarks

None.

USB_DEVICE_BOS_DESCRIPTOR_SUPPORT_ENABLE Macro

Specifies if the Device Layer should process a Host request for a BOS descriptor.

File

usb_device_config_template.h

C

```
#define USB_DEVICE_BOS_DESCRIPTOR_SUPPORT_ENABLE
```

Description

USB Device Layer BOS Descriptor Support Enable

Specifying this configuration macro will enable support for BOS request. When the request is received, the device layer will transfer the data pointed to by the bosDescriptor member of the USB_DEVICE_INIT data structure. If this configuration macro is not specified, request for a BOS descriptor is stalled.

Remarks

The USB Host will request for a BOS descriptor when the bcdVersion field in the Device Descriptor is greater than 0x0200.

USB_DEVICE_DRIVER_INITIALIZE_EXPLICIT Macro

Specifies if the USB Controller Driver must be initialized explicitly as opposed to being initialized by the Device Layer.

File

usb_device_config_template.h

C

```
#define USB_DEVICE_DRIVER_INITIALIZE_EXPLICIT
```

Description

USB Device Layer USB Controller Driver Explicit Initialize

Specifying this macro indicates that the USB Controller Driver will be initialized explicitly in the SYS_Initialize() function. The Device Layer will not initialize the USB Controller Driver. All releases of the USB Device Layer starting from v1.04 of MPALB Harmony will specify this macro i.e the controller driver will be initialized explicitly.

If this macro is not specified, then the USB Device Layer will initialize the controller driver and run its tasks routines.

Remarks

None.

USB_DEVICE_STRING_DESCRIPTOR_TABLE_ADVANCED_ENABLE Macro

Specifying this macro enables the Advanced String Descriptor Table Entry Format.

File

usb_device_config_template.h

C

```
#define USB_DEVICE_STRING_DESCRIPTOR_TABLE_ADVANCED_ENABLE
```

Description

USB Device Layer Advanced String Descriptor Table Entry Format Enable.

Specifyin this macro enables the Advanced String Descriptor Table Entry Format. The advanced format allows the application to specify the String Index and the language in the entry itself. In the basic format, this information is obtained by the virtue of the entry index of the String Descriptor in the String Descriptor Table. Using the Advanced format allows the application to specify strings with arbitrary strings indexes. In basic format, the string indexes are forced to be contiguous.

Remarks

The basic string descriptor entry format is selected by default. The advanced format must be enabled explicitly by specifying this macro.

Files

Files

Name	Description
usb_device.h	USB Device Layer Interface Header
usb_common.h	USB Common Definitions File
usb_device_config_template.h	USB device configuration template header file.

Description

This section lists the source and header files used by the library.

usb_device.h

USB Device Layer Interface Header

Enumerations

	Name	Description
	USB_DEVICE_CLIENT_STATUS	Enumerated data type that identifies the USB Device Layer Client Status.
	USB_DEVICE_CONTROL_STATUS	USB Device Layer Control Transfer Status Stage flags.
	USB_DEVICE_CONTROL_TRANSFER_RESULT	Enumerated data type identifying results of a control transfer.
	USB_DEVICE_EVENT	USB Device Layer Events.
	USB_DEVICE_POWER_STATE	Enumerated data type that identifies if the device is self powered or bus powered .
	USB_DEVICE_REMOTE_WAKEUP_STATUS	Enumerated data type that identifies if the remote wakeup status of the device.
	USB_DEVICE_RESULT	USB Device Layer Results Enumeration
	USB_DEVICE_TRANSFER_FLAGS	Enumerated data type that identifies the USB Device Layer Transfer Flags.

Functions

	Name	Description
	USB_DEVICE_ActiveConfigurationGet	Informs the client of the current USB device configuration set by the USB host.
	USB_DEVICE_ActiveSpeedGet	Informs the client of the current operation speed of the USB bus.
	USB_DEVICE_Attach	This function will attach the device to the USB.
	USB_DEVICE_ClientStatusGet	Returns the client specific status.
	USB_DEVICE_Close	Closes an opened handle to an instance of the USB device layer.
	USB_DEVICE_ControlReceive	Receives data stage of the control transfer from host to device.
	USB_DEVICE_ControlSend	Sends data stage of the control transfer from device to host.
	USB_DEVICE_ControlStatus	Initiates status stage of the control transfer.

	USB_DEVICE_Deinitialize	De-initializes the specified instance of the USB device layer.
	USB_DEVICE_Detach	This function will detach the device from the USB.
	USB_DEVICE_EndpointDisable	Disables a device endpoint.
	USB_DEVICE_EndpointEnable	Enables a device endpoint.
	USB_DEVICE_EndpointIsEnabled	Returns true if the endpoint is enabled.
	USB_DEVICE_EndpointIsStalled	This function returns the stall status of the specified endpoint and direction.
	USB_DEVICE_EndpointRead	Reads data received from host on the requested endpoint.
	USB_DEVICE_EndpointStall	This function stalls an endpoint in the specified direction.
	USB_DEVICE_EndpointStallClear	This function clears the stall on an endpoint in the specified direction.
	USB_DEVICE_EndpointTransferCancel	This function cancels a transfer scheduled on an endpoint.
	USB_DEVICE_EndpointWrite	This function requests a data write to a USB Device Endpoint.
	USB_DEVICE_EventHandlerSet	USB Device Layer Event Handler Callback Function set function.
	USB_DEVICE_Initialize	Creates and initializes an instance of the USB device layer.
	USB_DEVICE_IsSuspended	Returns true if the device is in a suspended state.
	USB_DEVICE_Open	Opens the specified USB device layer instance and returns a handle to it.
	USB_DEVICE_PowerStateSet	Sets power state of the device.
	USB_DEVICE_RemoteWakeupStart	This function will start the resume signaling.
	USB_DEVICE_RemoteWakeupStartTimed	This function will start a self timed Remote Wake-up.
	USB_DEVICE_RemoteWakeupStatusGet	Gets the "Remote wake-up" status of the device.
	USB_DEVICE_RemoteWakeupStop	This function will stop the resume signaling.
	USB_DEVICE_StateGet	Returns the current state of the USB device.
	USB_DEVICE_Status	Provides the current status of the USB device layer
	USB_DEVICE_Tasks	USB Device layer calls all other function driver tasks in this function. It also generates and forwards events to its clients.
	USB_DEVICE_Tasks_ISR	USB Device Layer Tasks routine to be called in the USB Interrupt Service Routine.
	USB_DEVICE_Tasks_ISR_USBDMA	This is function USB_DEVICE_Tasks_ISR_USBDMA.

Macros

	Name	Description
	USB_DEVICE_EVENT_RESPONSE_NONE	Device Layer Event Handler Function Response Type.
	USB_DEVICE_HANDLE_INVALID	Constant that defines the value of an Invalid Device Handle.
	USB_DEVICE_INDEX_0	USB device layer index definitions.
	USB_DEVICE_INDEX_1	This is macro USB_DEVICE_INDEX_1.
	USB_DEVICE_INDEX_2	This is macro USB_DEVICE_INDEX_2.
	USB_DEVICE_INDEX_3	This is macro USB_DEVICE_INDEX_3.
	USB_DEVICE_INDEX_4	This is macro USB_DEVICE_INDEX_4.
	USB_DEVICE_INDEX_5	This is macro USB_DEVICE_INDEX_5.
	USB_DEVICE_TRANSFER_HANDLE_INVALID	Constant that defines the value of an Invalid Device Endpoint Data Transfer Handle.

Structures

	Name	Description
	USB_DEVICE_EVENT_DATA_CONFIGURED	USB Device Set Configuration Event Data type.
	USB_DEVICE_EVENT_DATA_ENDPOINT_READ_COMPLETE	USB Device Layer Endpoint Read and Write Complete Event Data type.
	USB_DEVICE_EVENT_DATA_ENDPOINT_WRITE_COMPLETE	USB Device Layer Endpoint Read and Write Complete Event Data type.
	USB_DEVICE_EVENT_DATA_SET_DESCRIPTOR	USB Device Set Descriptor Event Data type.
	USB_DEVICE_EVENT_DATA_SOF	USB Device Start Of Frame Event Data Type
	USB_DEVICE_EVENT_DATA_SYNCH_FRAME	USB Device Synch Frame Event Data type.
	USB_DEVICE_FUNCTION_REGISTRATION_TABLE	USB Device Function Registration Structure
	USB_DEVICE_INIT	USB Device Initialization Structure
	USB_DEVICE_MASTER_DESCRIPTOR	USB Device Master Descriptor Structure.

Types

	Name	Description
	USB_DEVICE_CONFIGURATION_DESCRIPTORS_TABLE	Pointer to an array that contains pointer to configuration descriptors.
	USB_DEVICE_EVENT_HANDLER	USB Device Layer Event Handler Function Pointer Type
	USB_DEVICE_EVENT_RESPONSE	Device Layer Event Handler function return type.
	USB_DEVICE_HANDLE	Data type for USB device handle.
	USB_DEVICE_STRING_DESCRIPTORS_TABLE	Pointer to an array that contains pointer to string descriptors.
	USB_DEVICE_TRANSFER_HANDLE	Data type for USB Device Endpoint Data Transfer Handle.

Description

USB Device Layer Interface Definition

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB device layer. This application should include this file if it needs to use the USB Device Layer API.

File Name

usb_device.h

Company

Microchip Technology Inc.

usb_common.h

USB Common Definitions File

Enumerations

	Name	Description
	_USB_HOST_IRP_STATUS	Enumerates the possible status options of USB Host IRP.
	USB_DATA_DIRECTION	Defines the communication direction
	USB_DEVICE_IRP_FLAG	USB Device IRP flags enumeration
	USB_DEVICE_IRP_STATUS	Enumerates the possible status options of USB Device IRP.
	USB_ERROR	Enumeration of all possible error codes that are returned by various components functions in the USB Stack.
	USB_HOST_IRP_FLAG	USB Host IRP flags enumeration
	USB_HOST_IRP_STATUS	Enumerates the possible status options of USB Host IRP.
	USB_SPEED	Provides enumeration of USB 2.0 speeds.

Macros

	Name	Description
	USB_ENDPOINT_AND_DIRECTION	This macro helps in setting up the USB_ENDPOINT type.

Structures

	Name	Description
	_USB_DEVICE_IRP	This structure defines the USB Device Mode IRP data structure.
	_USB_HOST_IRP	This structure defines the USB Host Mode IRP data structure.
	USB_DEVICE_IRP	This structure defines the USB Device Mode IRP data structure.
	USB_HOST_IRP	This structure defines the USB Host Mode IRP data structure.

Types

	Name	Description
	USB_ENDPOINT	Defines a type to store Endpoint and Direction. The MSB defines the direction. The lower 4 bits defines the endpoint.

Description

USB Common Definitions File

This file contains definitions that are used by various components of the USB stack. This file is included by the USB Device and Host stack files. The application may typically not need to include this file directly.

File Name

usb_common.h

Company

Microchip Technology Inc.

usb_device_config_template.h

USB device configuration template header file.

Macros

	Name	Description
	USB_DEVICE_BOS_DESCRIPTOR_SUPPORT_ENABLE	Specifies if the Device Layer should process a Host request for a BOS descriptor.
	USB_DEVICE_DRIVER_INITIALIZE_EXPLICIT	Specifies if the USB Controller Driver must be initialized explicitly as opposed to being initialized by the Device Layer.
	USB_DEVICE_ENDPOINT_QUEUE_DEPTH_COMBINED	Specifies the combined endpoint queue depth in case of a vendor USB device implementation.
	USB_DEVICE_EP0_BUFFER_SIZE	Buffer Size in Bytes for Endpoint 0.
	USB_DEVICE_INSTANCES_NUMBER	Number of Device Layer instances to provisioned in the application.
	USB_DEVICE_MICROSOFT_OS_DESCRIPTOR_SUPPORT_ENABLE	Specifies if the USB Device stack should support Microsoft OS Descriptor.
	USB_DEVICE_SET_DESCRIPTOR_EVENT_ENABLE	Enables the Device Layer Set Descriptor Event.
	USB_DEVICE_SOF_EVENT_ENABLE	Enables the Device Layer SOF event.
	USB_DEVICE_STRING_DESCRIPTOR_TABLE_ADVANCED_ENABLE	Specifying this macro enables the Advanced String Descriptor Table Entry Format.
	USB_DEVICE_SYNCH_FRAME_EVENT_ENABLE	Enables the Device Layer Synch Frame Event.

Description

USB Device Layer Compile Time Options

This file contains USB device layer compile time options (macros) that are to be configured by the user. This file is a template file and must be used as an example only. This file must not be directly included in the project.

File Name

usb_device_config_template.h

Company

Microchip Technology Inc.

USB CDC Device Library

This section describes the USB CDC Device Library.

Introduction

This help section provides information on library design, configuration, usage and the Library Interface for the USB Communications Device Class (CDC) Device Library.

Description

The MPLAB Harmony USB Communications Device Class (CDC) Device Library (also referred to as the CDC function driver or library) provides functions and methods that allow application designers to implement a USB CDC Device. The current version of the library supports the Abstract Control Model (ACM) of the CDC specification revision 1.2 and specifically implements a subset of the AT250 command set. This library must be used in conjunction with the MPLAB Harmony USB Device Layer.

Using the Library

This topic describes the basic architecture of the USB CDC Device Library and provides information and examples on its use.

Library Overview

The USB CDC Device Library mainly interacts with the system, its clients and function drivers, as shown in the Abstraction Model.

The library interface routines are divided into sub-sections, which address one of the blocks or the overall operation of the USB CDC Device Library.

Library Interface Section	Description
Functions	Provides event handler, read/write, and serial state notification functions.

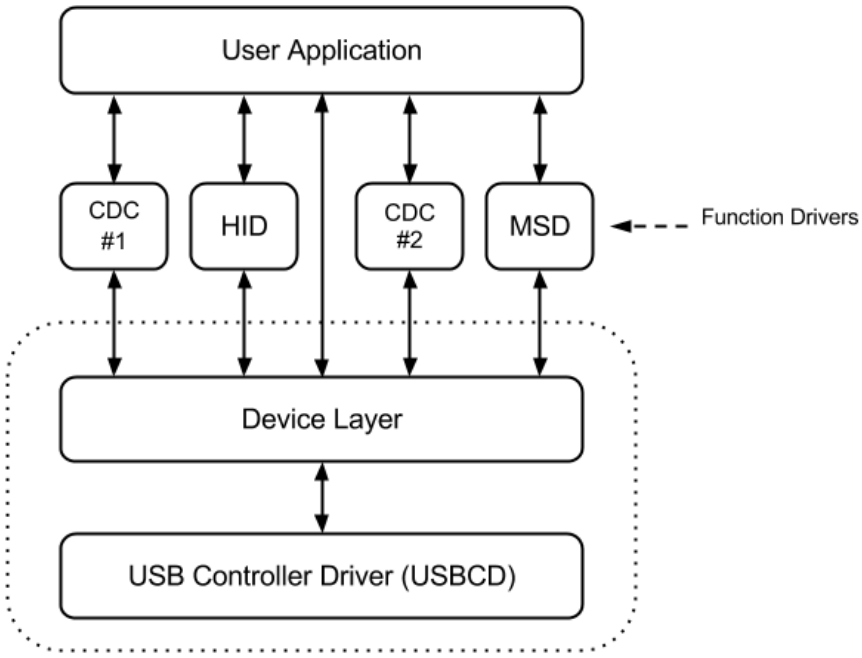
Abstraction Model

Provides an architectural overview of the CDC Function Driver.

Description

The CDC Function Driver offers services to a USB CDC Device to communicate with the host by abstracting the USB specification details. It must be used along with the USB Device Layer and USB controller to communicate with the USB Host. Figure 1 shows a block diagram of the MPLAB Harmony USB Architecture and where the CDC Function Driver is placed.

Figure 1: CDC Function Driver



As shown in Figure 1, the USB Controller Driver takes the responsibility of managing the USB peripheral on the device. The USB Device Layer handles the device enumeration, etc. The USB Device layer forwards all USB CDC specific control transfers to the CDC Function Driver. The CDC Function Driver ACM sub-layer interprets the control transfers and requests application's intervention through event handlers and well defined set of API. The application must register a event handler with the CDC Function Driver in the Device Layer Set Configuration Event. The application should respond to CDC ACM events. Response to CDC ACM event that require control transfer response can be deferred by responding to the event after returning from the event handler. The application interacts directly with the CDC Function Driver to send/receive data and to send serial state notifications.

As per the CDC specification,a USB CDC Device is a collection of the following interfaces:

- Communication Interface (Device Management) on Endpoint 0
- Optional Communication Interface (Notification) on an interrupt endpoint
- Optional Data Interface (either a bulk or isochronous endpoint)

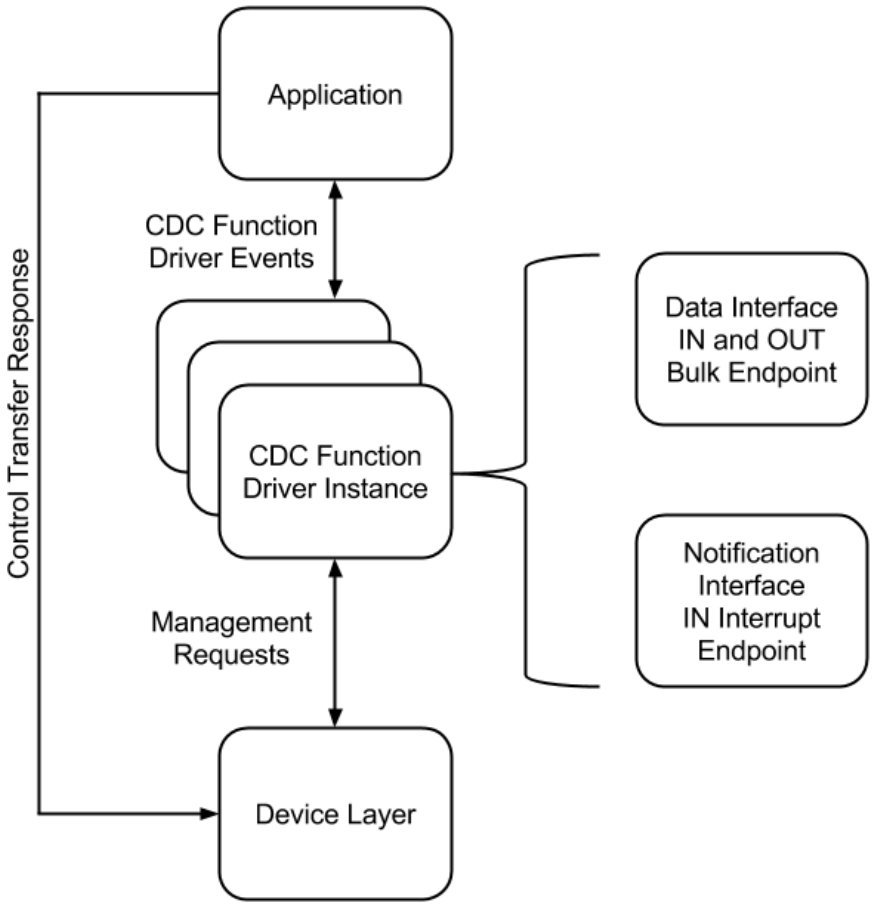


Figure 2: CDC Function Driver Architecture

Figure 2 shows the architecture of the CDC Function Driver. The device management on Endpoint 0 is handled by the device library(class specific requests are routed to the CDC Function Driver by the USB Device Layer). An instance of the CDC Function Driver actually consists of a data interface and a notification interface. The library is implemented in two `.c` files. The `usb_device_cdc.c` file implements the CDC data and serial state notification, while the `usb_device_cdc_acm.c` file implements the control transfer interpretation and event generation. The application must respond to control transfer related CDC ACM events by directly calling the Device Layer control transfer routines.

Abstract Control Model (ACM)

Describes the various Abstract Control Model (ACM) commands supported by this CDC Function Driver implementation.

Description

One of the basic supported models for communication by CDC is POTS (Plain Old Telephone Service). The POTS model is for devices that communicate via ordinary phone lines and generic COM port devices. The USB CDC specification refers to this basic model as PSTN (Public Switched Telephone Network).

Depending on the amount of data processing the device is responsible for POTS/PSTN is divided into several models. The processing of data can include modulation, demodulation, error correction and data compression.

Of the supported PSTN models, this CDC Function Driver implements ACM. In the ACM the device handles modulation, demodulation and handles V.25ter (AT) commands. This model (ACM) also supports requests and notifications to get and set RS-232 status, control, and asynchronous port part parameters. Virtual COM port devices use ACM.

The following sections describe the management requests and notifications supported by the CDC Function Driver ACM layer.

Management Requests

The Host requests/sends some information in the form of management requests on the bidirectional Endpoint 0. The following table shows the CDC specification ACM sub class management requests and how these request are handled by the CDC Function Driver.

Request Code	Required/Optional	Comments
SEND_ENCAPSULATED_COMMAND	Required	Implemented by the CDC Function Driver ACM layer. This request is stalled.
GET_ENCAPSULATED_RESPONSE	Required	Implemented by the CDC Function Driver ACM layer. This request is stalled.
SET_COMM_FEATURE	Optional	Not Implemented.

GET_COMM_FEATURE	Optional	Not Implemented.
CLEAR_COMM_FEATURE	Optional	Not Implemented.
SET_LINE_CODING	Optional	Implemented by the CDC Function Driver ACM layer. Requires application response.
GET_LINE_CODING	Optional	Implemented by the CDC Function Driver ACM layer. Requires application response.
SET_CONTROL_LINE_STATE	Optional	Implemented by the CDC Function Driver ACM layer. Requires application response.
SEND_BREAK	Optional	Implemented by the CDC Function Driver ACM layer. Requires application response.

How the Library Works

Library Initialization

Describes how the CDC Function Driver is initialized.

Description

The CDC Function Driver instance for a USB device configuration is initialized by the Device Layer when the configuration is set by the host. This process does not require application intervention. Each instance of the CDC Function Driver should be registered with the Device layer through the Device Layer Function Driver Registration Table. The CDC Function Driver does require a initialization data structure to be defined for each instance of the function driver. This initialization data structure should be of the type USB_DEVICE_CDC_INIT. This data structure specifies the read and write queue sizes. The funcDriverInit member of the function driver registration table entry for the CDC Function Driver instance should be set to point to the corresponding initialization data structure. The USB_DEVICE_CDC_FUNCTION_DRIVER object is a global object provided by the CDC Function Driver and points to the CDC Function Driver - Device Layer interface functions, which are required by the Device Layer. The following code an example of how multiple instances of CDC Function Driver can registered with the Device Layer.

```
/* This code shows an example of how two CDC function
 * driver instances can be registered with the Device Layer
 * via the Device Layer Function Driver Registration Table.
 * In this case Device Configuration 1 consists of two CDC
 * function driver instances. */

/* Define the CDC initialization data structure for CDC instance 0.
 * Set read queue size to 2 and write queue size to 3 */

const USB_DEVICE_CDC_INIT cdcInit0 = {.queueSizeRead = 2, .queueSizeWrite = 3};

/* Define the CDC initialization data structure for CDC instance 1.
 * Set read queue size to 4 and write queue size to 1 */

const USB_DEVICE_CDC_INIT cdcInit1 = {.queueSizeRead = 4, .queueSizeWrite = 1};
const USB_DEVICE_FUNC_REGISTRATION_TABLE funcRegistrationTable[2] =
{
    /* This is the first instance of the CDC Function Driver */
    {
       .speed = USB_SPEED_FULL|USB_SPEED_HIGH,     // Supported speed
       .configurationValue = 1,                    // To be initialized for Configuration 1
       .interfaceNumber = 0,                       // Starting interface number.
       .numberOfInterfaces = 2,                    // Number of interfaces in this instance
       .funcDriverIndex = 0,                       // Function Driver instance index is 0
       .funcDriverInit = &cdcInit0,                // Function Driver initialization data structure
       .driver = USB_DEVICE_CDC_FUNCTION_DRIVER    // Pointer to Function Driver - Device Layer interface
functions
    },
    /* This is the second instance of the CDC Function Driver */
    {
       .speed = USB_SPEED_FULL|USB_SPEED_HIGH,     // Supported speed
       .configurationValue = 1,                    // To be initialized for Configuration 1
       .interfaceNumber = 2,                       // Starting interface number.
       .numberOfInterfaces = 2,                    // Number of interfaces in this instance
       .funcDriverIndex = 1,                       // Function Driver instance index is 1
```

```
        .funcDriverInit = &cdcInit1,                // Function Driver initialization data structure
        .driver = USB_DEVICE_CDC_FUNCTION_DRIVER    // Pointer to Function Driver - Device Layer interface
functions
    },
};
```

Event Handling

Describes CDC Function Driver event handler registration and event handling.

Description

Registering a CDC Function Driver Event Handler

While creating USB CDC Device-based application, an event handler must be registered with the Device Layer (the Device Layer Event Handler) and every CDC Function Driver instance (CDC Function Driver Event Handler). The CDC Function Driver event handler receives CDC and CDC ACM events. This event handler should be registered before the USB device layer acknowledges the SET CONFIGURATION request from the USB Host. To ensure this, the event handler should be set in the USB_DEVICE_EVENT_CONFIGURED event that is generated by the device layer. While registering the CDC Function Driver event handler, the CDC Function Driver allows the application to also pass a data object in the event handler register function. This data object gets associated with the instance of the CDC Function Driver and is returned by the CDC Function Driver when a CDC Function Driver event occurs. The following code shows an example of how this can be done.

```
/* This is a sample Application Device Layer Event Handler
* Note how the CDC Function Driver event handler APP_USBDeviceCDCEventHandler()
* is registered in the USB_DEVICE_EVENT_CONFIGURED event. The appData
* object that is passed in the USB_DEVICE_CDC_EventHandlerSet()
* function will be returned as the userData parameter in the
* when the APP_USBDeviceCDCEventHandler() function is invoked */

   /* Application states */
    typedef enum
    {
        /* Application's state machine's initial state. */
        APP_STATE_INIT=0,
        APP_STATE_SERVICE_TASKS,
        APP_STATE_WAIT_FOR_CONFIGURATION,
    } APP_STATES;

    USB_DEVICE_HANDLE usbDeviceHandle;
    APP_STATES appState;

    /* This is the application device layer event handler function. */

    USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler
    (
        USB_DEVICE_EVENT event,
        void * eventData,
        uintptr_t context
    )
    {
        uint8_t activeConfiguration;
        USB_SETUP_PACKET * setupPacket;
        switch(event)
        {
            case USB_DEVICE_EVENT_POWER_DETECTED:
                /* This event in generated when VBUS is detected. Attach the device */
                USB_DEVICE_Attach(usbDeviceHandle);
                break;

            case USB_DEVICE_EVENT_POWER_REMOVED:
                /* This event is generated when VBUS is removed. Detach the device */
                USB_DEVICE_Detach (usbDeviceHandle);
                break;

            case USB_DEVICE_EVENT_CONFIGURED:
                /* This event indicates that Host has set Configuration in the Device.*/
                /* Check the configuration */
                activeConfiguration = ((USB_DEVICE_EVENT_DATA_CONFIGURED *)pData)->configurationValue;
                if ( activeConfiguration == 1)
                {
```

```
                    /* Register the CDC Device application event handler here.
                    * Note how the appData object pointer is passed as the
                    * user data */
                    USB_DEVICE_CDC_EventHandlerSet(USB_DEVICE_CDC_INDEX_0, APP_USBDeviceCDCEventHandler,
                                                   (uintptr_t)&appData);
                    /* Mark that set configuration is complete */
                    appData.isConfigured = true;
                }
                break;

            case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST:
                /* This event indicates a Control transfer setup stage has been completed. */
                setupPacket = (USB_SETUP_PACKET *)eventData;

                /* Parse the setup packet and respond with a USB_DEVICE_ControlSend(),
                   USB_DEVICE_ControlReceive or USB_DEVICE_ControlStatus(). */

                break;

            case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT:
                /* This event indicates that a Control transfer Data has been sent to Host. */
                break;

            case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:
                /* This event indicates that a Control transfer Data has been received from Host. */
                break;

            case USB_DEVICE_EVENT_CONTROL_TRANSFER_ABORTED:
                /* This event indicates a control transfer was aborted. */
                break;

            case USB_DEVICE_EVENT_SUSPENDED:
                break;

            case USB_DEVICE_EVENT_RESUMED:
                break;

            case USB_DEVICE_EVENT_ERROR:
                break;

            case USB_DEVICE_EVENT_RESET:
                break;

            case USB_DEVICE_EVENT_SOF:
                /* This event indicates an SOF is detected on the bus. The USB_DEVICE_SOF_EVENT_ENABLE
                   macro should be defined to get this event. */
                break;
            default:
                break;
        }
    }
```

The CDC Function Driver event handler executes in an interrupt context when the device stack is configured for Interrupt mode. In Polled mode, the event handler is invoked in the context of the SYS_Tasks function. The application should not call computationally intensive functions, blocking functions, functions that are not interrupt safe, or functions that poll on hardware conditions from the event handler. Doing so will affect the ability of the USB device stack to respond to USB events and could potentially make the USB device non-compliant.

CDC Function Driver Events

The CDC Function Driver generates events to which the application must respond. Some of these events are management requests communicated through control transfers. Therefore, the application must use the Device Layer Control Transfer routines to complete the control transfer. Based on the generated event, the application may be required to:

- Respond with a USB_DEVICE_ControlSend function, which is completes the data stage of a Control Read Transfer
- Respond with a USB_DEVICE_ControlReceive function, which provisions the data stage of a Control Write Transfer
- Respond with a USB_DEVICE_ControlStatus function, which completes the handshake stage of the Control Transfer. The application can either STALL or Acknowledge the handshake stage via the USB_DEVICE_ControlStatus function. The following table shows the CDC Function Driver Control Transfer related events and the required application control transfer actions.

CDC Function Driver Control Transfer Event	Required Application Action
USB_DEVICE_CDC_EVENT_SET_LINE_CODING	Call USB_DEVICE_ControlReceive function with a buffer to receive the USB_CDC_LINE_CODING type data.
USB_DEVICE_CDC_EVENT_SET_LINE_CODING	Call USB_DEVICE_ControlSend function with a buffer that contains the current USB_CDC_LINE_CODING type data.
USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE	Acknowledge or stall using the USB_DEVICE_ControlStatus function.
USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE	Acknowledge or stall using the USB_DEVICE_ControlStatus function.
USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT	Action not required.
USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	Acknowledge or stall using the USB_DEVICE_ControlStatus function.

Based on the type of event, the application should analyze the pData member of the event handler. This data member should be type cast to an event specific data type. The following table shows the event and the data type to use while type casting. Note that the pData member is not required for all events

CDC Function Driver Event	Related pData type
USB_DEVICE_CDC_EVENT_SET_LINE_CODING	NULL
USB_DEVICE_CDC_EVENT_GET_LINE_CODING	NULL
USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE	USB_CDC_CONTROL_LINE_STATE *
USB_DEVICE_CDC_EVENT_SEND_BREAK	USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK *
USB_DEVICE_CDC_EVENT_WRITE_COMPLETE	USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE *
USB_DEVICE_CDC_EVENT_READ_COMPLETE	USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE *
USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE	USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE *
USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT	NULL
USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	NULL

The possible CDC Function Driver events are described here with the required application response, event specific data, and likely follow-up function driver event:

USB_DEVICE_CDC_EVENT_SET_LINE_CODING

Application Response: This event occurs when the host issues a SET LINE CODING command. The application must provide a USB_CDC_LINE_CODING data structure to the device layer to receive the line coding data that the host will provide. The application must provide the buffer by calling the USB_DEVICE_CDC_ControlReceive function either in the event handler or in the application after returning from the event handler. The application can use the USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT event to track completion of the command.

Event Specific Data (pData): The pData parameter will be NULL.

Likely Follow-up event: This event will likely be followed by the USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event. This indicates that the data was received successfully. The application must either acknowledge or stall the handshake stage of the control transfer by calling the USB_DEVICE_ControlStatus function with the USB_DEVICE_CONTROL_STATUS_OK or USB_DEVICE_CONTROL_STATUS_ERROR flag, respectively.

USB_DEVICE_CDC_EVENT_GET_LINE_CODING

Application Response: This event occurs when the host issues a GET LINE CODING command. The application must provide a USB_CDC_LINE_CODING data structure to the device layer that contains the line coding data to be provided to the Host. The application must provide the buffer by calling the USB_DEVICE_ControlSend function either in the event handler or in the application after returning from the event handler. The size of the buffer is indicated by the length parameter. The application can use the USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT event to track completion of the command.

Event Specific Data (pData): The pData parameter will be NULL.

Likely Follow-up event: This event will likely be followed by the USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT event. This indicates that the data was sent to the Host successfully.

USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE

Application Response: This event occurs when the host issues a SET CONTROL LINE STATE command. The application can then use the USB_DEVICE_ControlStatus function to indicate acceptance of rejection of the command. The USB_DEVICE_ControlStatus function can be

called from the event handler or in the application after returning from the event handler.

Event Specific Data (pData): The application must interpret the pData parameter as a pointer to a USB_CDC_CONTROL_LINE_STATE data type that contains the control line state data.

Likely Follow-up event: None.

USB_DEVICE_CDC_EVENT_SEND_BREAK

Application Response: This event occurs when the Host issues a SEND BREAK command. The application can then use the USB_DEVICE_ControlStatus function to indicate acceptance or rejection of the command. The USB_DEVICE_ControlStatus function can be called from the event handler or in the application after returning from the event handler.

Event Specific Data (pData): The application must interpret the pData parameter as a pointer to a uint16_t data type that contains the break duration data.

Likely Follow-up event: None.

USB_DEVICE_CDC_EVENT_WRITE_COMPLETE

Application Response: This event occurs when a write operation scheduled by calling the USB_DEVICE_CDC_Write function has completed. This event does not require the application to respond with any function calls.

Event Specific Data (pData): The pData member in the event handler will point to the USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE data type.

Likely Follow-up event: None.

USB_DEVICE_CDC_EVENT_READ_COMPLETE

Application Response: This event occurs when a read operation scheduled by calling the USB_DEVICE_CDC_Read function has completed. This event does not require the application to respond with any function calls.

Event Specific Data (pData): The pData member in the event handler will point to the USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE type.

Likely Follow-up event: None.

USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE

Application Response: This event occurs when a serial state notification send scheduled by calling the USB_DEVICE_CDC_SerialStateNotificationSend function has completed. This event does not require the application to respond with any function calls.

Event Specific Data (pData): The pData member in the event handler will point to the USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE data type.

Likely Follow-up event: None.

USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT

Application Response: This event occurs when the data stage of a control read transfer has completed in response to the USB_DEVICE_ControlSend function (in the USB_DEVICE_CDC_EVENT_GET_LINE_CODING event). The application must acknowledge the handshake stage of the control transfer by calling the USB_DEVICE_ControlStatus function with the USB_DEVICE_CONTROL_STATUS_OK flag.

Event Specific Data (pData): The pData parameter will be NULL.

Likely Follow-up event: None.

USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED

Application Response: This event occurs when the data stage of a control write transfer has completed in response to the USB_DEVICE_ControlReceive function (in the USB_DEVICE_CDC_EVENT_SET_LINE_CODING event).

Event Specific Data (pData): The pData parameter will be NULL.

Likely Follow-up event: None.

CDC Function Driver Event Handling

The following code shows an event handling scheme example. The application always returns from the event handler with a USB_DEVICE_CDC_EVENT_RESPONSE_NONE value.

```
    // This code example shows all CDC Function Driver possible events
    // and a possible scheme for handling these events. In this case
    // event responses are not deferred.

    uint16_t * breakData;
    USB_DEVICE_HANDLE    usbDeviceHandle;
    USB_CDC_LINE_CODING  lineCoding;
    USB_CDC_CONTROL_LINE_STATE * controlLineStateData
```

```
USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT event,
    void * data,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_CDC_EVENT_SET_LINE_CODING:

            // In this case, the application should read the line coding
            // data that is sent by the host.

            USB_DEVICE_ControlReceive(usbDeviceHandle, &lineCoding,
                                sizeof(USB_CDC_LINE_CODING));
            break;

        case USB_DEVICE_CDC_EVENT_GET_LINE_CODING:

            // In this case, the application should send the line coding
            // data to the host.

            USB_DEVICE_ControlSend(usbDeviceHandle, &lineCoding,
                                sizeof(USB_CDC_LINE_CODING));
            break;

        case USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE:

            // In this case, pData should be interpreted as a
            // USB_CDC_CONTROL_LINE_STATE pointer type.  The application
            // acknowledges the parameters by calling the
            // USB_DEVICE_ControlStatus() function with the
            // USB_DEVICE_CONTROL_STATUS_OK option.

            controlLineStateData = (USB_CDC_CONTROL_LINE_STATE *)pData;
            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);

            break;

        case USB_DEVICE_CDC_EVENT_SEND_BREAK:

            // In this case, pData should be interpreted as a uint16_t
            // pointer type to the break duration. The application
            // acknowledges the parameters by calling the
            // USB_DEVICE_ControlStatus() function with the
            // USB_DEVICE_CONTROL_STATUS_OK option.

            breakDuration = (USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK *)pData;
            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);

            break;

        case USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT:

            // This event indicates the data send request associated with
            // the latest USB_DEVICE_ControlSend() function was
            // completed.  The application could use this event to track
            // the completion of the USB_DEVICE_CDC_EVENT_GET_LINE_CODING
            // request.

            break;
        case USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:

            // This means that the data stage is complete. The data in
            // setLineCodingData is valid or data in getLineCodingData was
            // sent to the host.  The application can now decide whether it
            // supports this data. It is not mandatory to do this in the
```

```
                // event handler.

                USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);

            case USB_DEVICE_CDC_EVENT_WRITE_COMPLETE:

                // This means USB_DEVICE_CDC_Write() operation completed.
                // The pData member will point to a
                // USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE type of data.

                break;
            case USB_DEVICE_CDC_EVENT_READ_COMPLETE:

                // This means USB_DEVICE_CDC_Read() operation completed.
                // The pData member will point to a
                // USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE type of data.

                break;

            case USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE:

                // This means USB_DEVICE_CDC_SerialStateNotification() operation
                // completed. The pData member will point to a
                // USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE type of data.

                break;

             default:
                break;
        }

        return(USB_DEVICE_CDC_EVENT_RESPONSE_NONE);
    }
```

Refer to the USB_DEVICE_CDC_EVENT enumeration for more details on each event.

Sending Data

Describes how to send data to the CDC Host.

Description

The application may need to send data or serial state notification to the USB CDC Host. This is done by using the USB_DEVICE_CDC_Write and USB_DEVICE_CDC_SerialStateNotificationSend functions, respectively.

Sending Data to the USB Host

The application can send data to the Host by using the USB_DEVICE_CDC_Write function. This function returns a transfer handle that allows the application to track the write request. The request is completed when the Host has requested the data. The completion of the write transfer is indicated by a USB_DEVICE_CDC_EVENT_WRITE_COMPLETE event. A write request could fail if the function driver instance transfer queue is full.

The USB_DEVICE_CDC_Write function also allows the application to send data to the host without ending the transfer. This is done by specifying the USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_PENDING flag. The application can use this option when the data to be sent is not readily available or when the application is memory constrained. The combination of the transfer flag and the transfer size affects how the function driver sends the data to the host:

- If size is a multiple of maxPacketSize (the IN endpoint size), and the flag is set as USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE, the write function will append a Zero Length Packet (ZLP) to complete the transfer
- If size is a multiple of maxPacketSize, and the flag is set as USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function will not append a ZLP and therefore, will not complete the transfer
- If size is greater than but not a multiple of maxPacketSize, and the flag is set as USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE, the write function schedules (length/maxPacketSize) packets and one packet for the residual data
- If size if greater than but not a multiple of maxPacketSize, and the flag is set as USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function returns an error code and sets the transferHandle parameter to USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID
- If size is less than maxPacketSize, and the flag is set as USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE, the write function schedules one packet
- If size is less than maxPacketSize, and the flag is set as USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING, the write

function returns an error code and sets the transferHandle parameter to USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID

The following code shows a set of examples of various conditions attempting to send data with the USB_DEVICE_CDC_Write command.

Example 1

```
// This example assume that the maxPacketSize is 64.
USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_INDEX instance;
USB_DEVICE_CDC_RESULT writeRequestResult;
uint8_t data[34];

// In this example we want to send 34 bytes only.
writeRequestResult = USB_DEVICE_CDC_Write(instance,&transferHandle, data, 34,
                                          USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);
if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}
```

Example 2

```
//------------------------------------------------------
// In this example we want to send 64 bytes only.
// This will cause a ZLP to be sent.

USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_INDEX instance;
USB_DEVICE_CDC_RESULT writeRequestResult;
uint8_t data[64];


writeRequestResult = USB_DEVICE_CDC_Write(instance,&transferHandle, data, 64,
                                          USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);
if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
//Do Error handling here
}
```

Example 3

```
//------------------------------------------------------
// This example will return an error because size is less
// than maxPacketSize and the flag indicates that more
// data is pending.

USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_INDEX instance;
USB_DEVICE_CDC_RESULT writeRequestResult;
uint8_t data[64];

writeRequestResult = USB_DEVICE_CDC_Write(instance,&transferHandle, data, 32,
                                    USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING);
```

Example 4

```
//------------------------------------------------------
// In this example we want to place a request for a 70 byte transfer.
// The 70 bytes will be sent out in a 64 byte transaction and a 6 byte
// transaction completing the transfer.

USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_INDEX instance;
USB_DEVICE_CDC_RESULT writeRequestResult;
uint8_t data[70];

writeRequestResult = USB_DEVICE_CDC_Write(instance,&transferHandle, data, 70,
                                          USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
//Do Error handling here
}
```

Example 5

```
//------------------------------------------------------
// In this example we want to place a request for a 70 bytes to be sent
```

```
// but that we don't end the transfer as more data is coming. 64 bytes
// of the 70 will be sent out and the USB_DEVICE_CDC_EVENT_WRITE_COMPLETE
// with 64 bytes. This indicates that the extra 6 bytes weren't
// sent because it would cause the end of the transfer. Thus the
// user needs to add these 6 bytes back to the buffer for the next group
// of data that needs to be sent out.

USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_INDEX instance;
USB_DEVICE_CDC_RESULT writeRequestResult;
uint8_t data[70];

writeRequestResult = USB_DEVICE_CDC_Write(instance,&transferHandle, data, 70,
                                          USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
//Do Error handling here
}
// The completion of the write request will be indicated by the
// USB_DEVICE_CDC_EVENT_WRITE_COMPLETE event.
```

Sending a Serial State Notification

The application can send a Serial State Notification by using the USB_DEVICE_CDC_SerialStateSend function. This function returns a transfer handle that allows the application to track the read request. The request is completed when the Host has requested the data. The completion of the transfer is indicated by a USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE event. The transfer request could fail if the function driver transfer queue is full. The following code shows an example of how this can be done.

```
USB_DEVICE_CDC_INDEX instanceIndex;
USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_SERIAL_STATE_NOTIFICATION_DATA notificationData;

// This application function could possibly update the notificationData
// data structure.

APP_UpdateNotificationData(&notificationData);

// Now send the updated notification data to the host.

result = USB_DEVICE_CDC_SerialStateDataSend(instanceIndex, &transferHandle,
                                          &notificationData);

if(USB_DEVICE_CDC_RESULT_OK != result)
{
    // Error handling here
}
```

Receiving Data

Describes how the CDC device can read data from the Host.

Description

The application can receive data from the host by using the USB_DEVICE_CDC_Read function. This function returns a transfer handle that allows the application to track the read request. The request is completed when the Host sends the required amount or less than required amount of data. The application must make sure that it allocates a buffer size that is at least the size or a multiple of the receive endpoint size. The return value of the function indicates the success of the request. A read request could fail if the function driver transfer queue is full. The completion of the read transfer is indicated by the USB_DEVICE_CDC_EVENT_READ_COMPLETE event. The request completes based on the amount of the data that was requested and size of the transaction initiated by the Host:

- If the size parameter is not a multiple of maxPacketSize or is '0', the function returns USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID in transferHandle and returns USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_SIZE_INVALID as a return value
- If the size parameter is a multiple of maxPacketSize and the Host sends less than maxPacketSize data in any transaction, the transfer completes and the function driver will issue a USB_DEVICE_CDC_EVENT_READ_COMPLETE event along with the USB_DEVICE_CDC_EVENT_READ_COMPLETE_DATA data structure
- If the size parameter is a multiple of maxPacketSize and the Host sends maxPacketSize amount of data, and total data received does not exceed size, the function driver will wait for the next packet

The following code shows an example of the USB_DEVICE_CDC_Read function:

```
// Shows an example of how to read. This assumes that
// driver was opened successfully.
```

```
USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_RESULT readRequestResult;
USB_DEVICE_CDC_HANDLE instanceHandle;

readRequestResult = USB_DEVICE_CDC_Read(instanceHandle,
                        &transferHandle, data, 128);

if(USB_DEVICE_CDC_RESULT_OK != readRequestResult)
{
    //Do Error handling here
}

// The completion of the read request will be indicated by the
// USB_DEVICE_CDC_EVENT_READ_COMPLETE event.
```

Configuring the Library

Describes how to configure the CDC Function Driver.

Macros

	Name	Description
	USB_DEVICE_CDC_INSTANCES_NUMBER	Specifies the number of CDC instances.
	USB_DEVICE_CDC_QUEUE_DEPTH_COMBINED	Specifies the combined queue size of all CDC instances.

Description

The application designer must specify the following configuration parameters while using the CDC Function Driver. The configuration macros that implement these parameters must be located in the `system_config.h` file in the application project and a compiler include path (to point to the folder that contains this file) should be specified.

USB_DEVICE_CDC_INSTANCES_NUMBER Macro

Specifies the number of CDC instances.

File

usb_device_cdc_config_template.h

C

```
#define USB_DEVICE_CDC_INSTANCES_NUMBER
```

Description

USB device CDC Maximum Number of instances

This macro defines the number of instances of the CDC Function Driver. For example, if the application needs to implement two instances of the CDC Function Driver (to create two COM ports) on one USB Device, the macro should be set to 2. Note that implementing a USB Device that features multiple CDC interfaces requires appropriate USB configuration descriptors.

Remarks

None.

USB_DEVICE_CDC_QUEUE_DEPTH_COMBINED Macro

Specifies the combined queue size of all CDC instances.

File

usb_device_cdc_config_template.h

C

```
#define USB_DEVICE_CDC_QUEUE_DEPTH_COMBINED
```

Description

USB device CDC Combined Queue Size

This macro defines the number of entries in all queues in all instances of the CDC function driver. This value can be obtained by adding up the read and write queue sizes of each CDC Function driver instance. In a simple single instance USB CDC device application, that does not require

buffer queuing and serial state notification, the USB_DEVICE_CDC_QUEUE_DEPTH_COMBINED macro can be set to 2. Consider a case with two CDC function driver instances, CDC 1 has a read queue size of 2 and write queue size of 3, CDC 2 has a read queue size of 4 and write queue size of 1, this macro should be set to 10 (2 +3 + 4 + 1).

Remarks

None.

Building the Library

Describes the files to be included in the project while using the CDC Function Driver.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/usb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
usb_device_cdc.h	This header file should be included in any `.c` file that accesses the USB Device CDC Function Driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/usb_device_cdc.c`	This file implements the CDC Data Interface and Communications interface and should be included in the project if the CDC Device function is desired.
`/src/dynamic/usb_device_cdc_acm.c`	This file implements the CDC-ACM layer and should be included in the project if the CDC Device function is desired.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The USB CDC Device Library depends on the following modules:

- USB Device Layer Library

Library Interface

a) Functions

	Name	Description
	USB_DEVICE_CDC_EventHandlerSet	This function registers a event handler for the specified CDC function driver instance.
	USB_DEVICE_CDC_Read	This function requests a data read from the USB Device CDC Function Driver Layer.
	USB_DEVICE_CDC_Write	This function requests a data write to the USB Device CDC Function Driver Layer.
	USB_DEVICE_CDC_SerialStateNotificationSend	This function schedules a request to send serial state notification to the host.

b) Data Types and Constants

	Name	Description
	USB_DEVICE_CDC_EVENT	USB Device CDC Function Driver Events

	USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE	USB Device CDC Function Driver Read and Write Complete Event Data.
	USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE	USB Device CDC Function Driver Read and Write Complete Event Data.
	USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE	USB Device CDC Function Driver Read and Write Complete Event Data.
	USB_DEVICE_CDC_EVENT_HANDLER	USB Device CDC Event Handler Function Pointer Type.
	USB_DEVICE_CDC_EVENT_RESPONSE	USB Device CDC Function Driver Event Callback Response Type
	USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK	USB Device CDC Function Driver Send Break Event Data
	USB_DEVICE_CDC_INDEX	USB Device CDC Function Driver Index
	USB_DEVICE_CDC_INIT	USB Device CDC Function Driver Initialization Data Structure
	USB_DEVICE_CDC_RESULT	USB Device CDC Function Driver USB Device CDC Result enumeration.
	USB_DEVICE_CDC_TRANSFER_FLAGS	USB Device CDC Function Driver Transfer Flags
	USB_DEVICE_CDC_TRANSFER_HANDLE	USB Device CDC Function Driver Transfer Handle Definition.
	USB_DEVICE_CDC_EVENT_RESPONSE_NONE	USB Device CDC Function Driver Event Handler Response Type None.
	USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID	USB Device CDC Function Driver Invalid Transfer Handle Definition.
	USB_DEVICE_CDC_FUNCTION_DRIVER	USB Device CDC Function Driver Function pointer
	USB_DEVICE_CDC_INDEX_0	Use this to specify CDC Function Driver Instance 0
	USB_DEVICE_CDC_INDEX_1	Use this to specify CDC Function Driver Instance 1
	USB_DEVICE_CDC_INDEX_2	Use this to specify CDC Function Driver Instance 2
	USB_DEVICE_CDC_INDEX_3	Use this to specify CDC Function Driver Instance 3
	USB_DEVICE_CDC_INDEX_4	Use this to specify CDC Function Driver Instance 4
	USB_DEVICE_CDC_INDEX_5	Use this to specify CDC Function Driver Instance 5
	USB_DEVICE_CDC_INDEX_6	Use this to specify CDC Function Driver Instance 6
	USB_DEVICE_CDC_INDEX_7	Use this to specify CDC Function Driver Instance 7

Description

This section describes the Application Programming Interface (API) functions of the USB CDC Device Library.

Refer to each section for a detailed description.

a) Functions

USB_DEVICE_CDC_EventHandlerSet Function

This function registers a event handler for the specified CDC function driver instance.

File

usb_device_cdc.h

C

```
USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_EventHandlerSet(USB_DEVICE_CDC_INDEX instanceIndex,
USB_DEVICE_CDC_EVENT_HANDLER eventHandler, uintptr_t context);
```

Returns

USB_DEVICE_CDC_RESULT_OK - The operation was successful USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist USB_DEVICE_CDC_RESULT_ERROR_PARAMETER_INVALID - The eventHandler parameter is NULL

Description

This function registers a event handler for the specified CDC function driver instance. This function should be called by the client when it receives a SET CONFIGURATION event from the device layer. A event handler must be registered for function driver to respond to function driver specific commands. If the event handler is not registered, the device layer will stall function driver specific commands and the USB device may not function.

Remarks

None.

Preconditions

This function should be called when the function driver has been initialized as a result of a set configuration.

Example

```
// This code snippet shows an example registering an event handler. Here
// the application specifies the context parameter as a pointer to an
// application object (appObject) that should be associated with this
// instance of the CDC function driver.

// Application states
typedef enum
{
    //Application's state machine's initial state.
    APP_STATE_INIT=0,
    APP_STATE_SERVICE_TASKS,
    APP_STATE_WAIT_FOR_CONFIGURATION,
} APP_STATES;

USB_DEVICE_HANDLE usbDeviceHandle;

APP_STATES appState;

// Get Line Coding Data
USB_CDC_LINE_CODING getLineCodingData;

// Control Line State
USB_CDC_CONTROL_LINE_STATE controlLineStateData;

// Set Line Coding Data
USB_CDC_LINE_CODING setLineCodingData;

USB_DEVICE_CDC_RESULT result;

USB_DEVICE_CDC_EVENT_RESPONSE APP_USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex ,
    USB_DEVICE_CDC_EVENT event ,
    void* pData,
    uintptr_t context
)
{
    // Event Handling comes here

    switch(event)
    {
        case USB_DEVICE_CDC_EVENT_GET_LINE_CODING:
                // This means the host wants to know the current line
                // coding. This is a control transfer request. Use the
                // USB_DEVICE_ControlSend() function to send the data to
                // host.

                USB_DEVICE_ControlSend(usbDeviceHandle,
                    &getLineCodingData, sizeof(USB_CDC_LINE_CODING));
```

```
            break;

            case USB_DEVICE_CDC_EVENT_SET_LINE_CODING:

                // This means the host wants to set the line coding.
                // This is a control transfer request. Use the
                // USB_DEVICE_ControlReceive() function to receive the
                // data from the host

                USB_DEVICE_ControlReceive(usbDeviceHandle,
                        &setLineCodingData, sizeof(USB_CDC_LINE_CODING));

            break;

            case USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE:

                // This means the host is setting the control line state.
                // Read the control line state. We will accept this request
                // for now.
                controlLineStateData.dtr = ((USB_CDC_CONTROL_LINE_STATE *)pData)->dtr;
                controlLineStateData.carrier = ((USB_CDC_CONTROL_LINE_STATE *)pData)->carrier;
                USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);

            break;

            case USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:

                // The data stage of the last control transfer is
                // complete. For now we accept all the data

                USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);

            break;

                case USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT:

                // This means the GET LINE CODING function data is valid. We dont
                // do much with this data in this demo.
            break;

            case USB_DEVICE_CDC_EVENT_SEND_BREAK:

                // This means that the host is requesting that a break of the
                // specified duration be sent.
                USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);

            break;

            case USB_DEVICE_CDC_EVENT_READ_COMPLETE:
                // This means that the host has sent some data
                break;

            case USB_DEVICE_CDC_EVENT_WRITE_COMPLETE:
                // This means that the host has sent some data
                break;

            default:
                break;
        }

        return USB_DEVICE_CDC_EVENT_RESPONSE_NONE;
    }

    // This is the application device layer event handler function.

    USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler
    (
        USB_DEVICE_EVENT event,
        void * pData,
```

```
    uintptr_t context
)
{
    USB_SETUP_PACKET * setupPacket;
    switch(event)
    {
        case USB_DEVICE_EVENT_POWER_DETECTED:
            // This event in generated when VBUS is detected. Attach the device
            USB_DEVICE_Attach(usbDeviceHandle);
            break;

        case USB_DEVICE_EVENT_POWER_REMOVED:
            // This event is generated when VBUS is removed. Detach the device
            USB_DEVICE_Detach (usbDeviceHandle);
            break;

        case USB_DEVICE_EVENT_CONFIGURED:
            // This event indicates that Host has set Configuration in the Device.
            // Register CDC Function driver Event Handler.
            USB_DEVICE_CDC_EventHandlerSet(USB_DEVICE_CDC_INDEX_0, APP_USBDeviceCDCEventHandler,
(uintptr_t)0);
            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST:
            // This event indicates a Control transfer setup stage has been completed.
            setupPacket = (USB_SETUP_PACKET *)pData;

            // Parse the setup packet and respond with a USB_DEVICE_ControlSend(),
            // USB_DEVICE_ControlReceive or USB_DEVICE_ControlStatus().

            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT:
            // This event indicates that a Control transfer Data has been sent to Host.
            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:
            // This event indicates that a Control transfer Data has been received from Host.
            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_ABORTED:
            // This event indicates a control transfer was aborted.
            break;

        case USB_DEVICE_EVENT_SUSPENDED:
            break;

        case USB_DEVICE_EVENT_RESUMED:
            break;

        case USB_DEVICE_EVENT_ERROR:
            break;

        case USB_DEVICE_EVENT_RESET:
            break;

        case USB_DEVICE_EVENT_SOF:
            // This event indicates an SOF is detected on the bus. The  USB_DEVICE_SOF_EVENT_ENABLE
            // macro should be defined to get this event.
            break;
        default:
            break;
    }
}


void APP_Tasks ( void )
{
    // Check the application's current state.
```

```
    switch ( appState )
    {
        // Application's initial state.
        case APP_STATE_INIT:
            // Open the device layer
            usbDeviceHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0,
                DRV_IO_INTENT_READWRITE );

            if(usbDeviceHandle != USB_DEVICE_HANDLE_INVALID)
            {
                // Register a callback with device layer to get event notification
                USB_DEVICE_EventHandlerSet(usbDeviceHandle,
                    APP_USBDeviceEventHandler, 0);
                appState = APP_STATE_WAIT_FOR_CONFIGURATION;
            }
            else
            {
                // The Device Layer is not ready to be opened. We should try
                // gain later.
            }
            break;

        case APP_STATE_SERVICE_TASKS:
            break;

            // The default state should never be executed.
        default:
            break;
    }
}
```

Parameters

Parameters	Description
instance	Instance of the CDC Function Driver.
eventHandler	A pointer to event handler function.
context	Application specific context that is returned in the event handler.

Function

USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_EventHandlerSet
(
USB_DEVICE_CDC_INDEX instance
USB_DEVICE_CDC_EVENT_HANDLER eventHandler
uintptr_t context
);

USB_DEVICE_CDC_Read Function

This function requests a data read from the USB Device CDC Function Driver Layer.

File

usb_device_cdc.h

C

```
USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_Read(USB_DEVICE_CDC_INDEX instanceIndex,
USB_DEVICE_CDC_TRANSFER_HANDLE * transferHandle, void * data, size_t size);
```

Returns

USB_DEVICE_CDC_RESULT_OK - The read request was successful. transferHandle contains a valid transfer handle.

USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL - internal request queue is full. The write request could not be added.

USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_SIZE_INVALID - The specified transfer size was not a multiple of endpoint size or is 0.

USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.

USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application and is invalid.

Description

This function requests a data read from the USB Device CDC Function Driver Layer. The function places a requests with driver, the request will get serviced as data is made available by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_CDC_EVENT_READ_COMPLETE event. The amount of data read and the transfer handle associated with the request is returned along with the event in the pData parameter of the event handler. The transfer handle expires when event handler for the USB_DEVICE_CDC_EVENT_READ_COMPLETE exits. If the read request could not be accepted, the function returns an error code and transferHandle will contain the value USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID.

If the size parameter is not a multiple of maxPacketSize or is 0, the function returns USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID in transferHandle and returns an error code as a return value. If the size parameter is a multiple of maxPacketSize and the host send less than maxPacketSize data in any transaction, the transfer completes and the function driver will issue a USB_DEVICE_CDC_EVENT_READ_COMPLETE event along with the USB_DEVICE_CDC_EVENT_READ_COMPLETE_DATA data structure. If the size parameter is a multiple of maxPacketSize and the host sends maxPacketSize amount of data, and total data received does not exceed size, then the function driver will wait for the next packet.

Remarks

While the using the CDC Function Driver with the PIC32MZ USB module, the receive buffer provided to the USB_DEVICE_CDC_Read function should be placed in coherent memory and aligned at a 16 byte boundary. This can be done by declaring the buffer using the __attribute__((coherent, aligned(16))) attribute. An example is shown here

```
uint8_t data[256] __attribute__((coherent, aligned(16)));
```

Preconditions

The function driver should have been configured.

Example

```
// Shows an example of how to read. This assumes that
// driver was opened successfully.

USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_RESULT readRequestResult;
USB_DEVICE_CDC_HANDLE instanceHandle;

readRequestResult = USB_DEVICE_CDC_Read(instanceHandle,
                        &transferHandle, data, 128);

if(USB_DEVICE_CDC_RESULT_OK != readRequestResult)
{
    //Do Error handling here
}

// The completion of the read request will be indicated by the
// USB_DEVICE_CDC_EVENT_READ_COMPLETE event.
```

Parameters

Parameters	Description
instance	USB Device CDC Function Driver instance.
transferHandle	Pointer to a USB_DEVICE_CDC_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the read request was successful.
data	pointer to the data buffer where read data will be stored.
size	Size of the data buffer. Refer to the description section for more details on how the size affects the transfer.

Function

```
  USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_Read
(
    USB_DEVICE_CDC_INDEX instance,
USB_CDC_DEVICE_TRANSFER_HANDLE * transferHandle,
void * data,
size_t size
);
```

USB_DEVICE_CDC_Write Function

This function requests a data write to the USB Device CDC Function Driver Layer.

File

usb_device_cdc.h

C

```
USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_Write(USB_DEVICE_CDC_INDEX instanceIndex,
USB_DEVICE_CDC_TRANSFER_HANDLE * transferHandle, const void * data, size_t size,
USB_DEVICE_CDC_TRANSFER_FLAGS flags);
```

Returns

USB_DEVICE_CDC_RESULT_OK - The write request was successful. transferHandle contains a valid transfer handle.

USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL - internal request queue is full. The write request could not be added.

USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_SIZE_INVALID - The specified transfer size and flag parameter are invalid.

USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.

USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application and is invalid.

Description

This function requests a data write to the USB Device CDC Function Driver Layer. The function places a requests with driver, the request will get serviced as data is requested by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_CDC_EVENT_WRITE_COMPLETE event. The amount of data written and the transfer handle associated with the request is returned along with the event in writeCompleteData member of the pData parameter in the event handler. The transfer handle expires when event handler for the USB_DEVICE_CDC_EVENT_WRITE_COMPLETE exits. If the read request could not be accepted, the function returns an error code and transferHandle will contain the value USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID.

The behavior of the write request depends on the flags and size parameter. If the application intends to send more data in a request, then it should use the USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING flag. If there is no more data to be sent in the request, the application must use the USB_DEVICE_CDC_EVENT_WRITE_COMPLETE flag. This is explained in more detail here:

- If size is a multiple of maxPacketSize and flag is set as

USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE, the write function will append a Zero Length Packet (ZLP) to complete the transfer.

- If size is a multiple of maxPacketSize and flag is set as

USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function will not append a ZLP and hence will not complete the transfer.

- If size is greater than but not a multiple of maxPacketSize and flags is

set as USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE, the write function returns an error code and sets the transferHandle parameter to USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID.

- If size is greater than but not a multiple of maxPacketSize and flags is

set as USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function fails and return an error code and sets the transferHandle parameter to USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID.

- If size is less than maxPacketSize and flag is set as

USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE, the write function schedules one packet.

- If size is less than maxPacketSize and flag is set as

USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function returns an error code and sets the transferHandle parameter to USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID.

- If size is 0 and the flag is set

USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE, the function driver will schedule a Zero Length Packet.

Completion of the write transfer is indicated by the USB_DEVICE_CDC_EVENT_WRITE_COMPLETE event. The amount of data written along with the transfer handle is returned along with the event.

Remarks

While the using the CDC Function Driver with the PIC32MZ USB module, the transmit buffer provided to the USB_DEVICE_CDC_Write function should be placed in coherent memory and aligned at a 16 byte boundary. This can be done by declaring the buffer using the __attribute__((coherent, aligned(16))) attribute. An example is shown here

```
uint8_t data[256] __attribute__((coherent, aligned(16)));
```

Preconditions

The function driver should have been configured.

Example

```
// Below is a set of examples showing various conditions trying to
```

```
// send data with the Write command.
//
// This assumes that driver was opened successfully.
// Assume maxPacketSize is 64.

USB_DEVICE_CDC_TRANSFER_HANDLE transferHandle;
USB_DEVICE_CDC_RESULT writeRequestHandle;
USB_DEVICE_CDC_INDEX instance;

//------------------------------------------------------
// In this example we want to send 34 bytes only.

writeRequestResult = USB_DEVICE_CDC_Write(instance,
                        &transferHandle, data, 34,
                        USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

//------------------------------------------------------
// In this example we want to send 64 bytes only.
// This will cause a ZLP to be sent.

writeRequestResult = USB_DEVICE_CDC_Write(instance,
                        &transferHandle, data, 64,
                        USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

//------------------------------------------------------
// This example will return an error because size is less
// than maxPacketSize and the flag indicates that more
// data is pending.

writeRequestResult = USB_DEVICE_CDC_Write(instanceHandle,
                        &transferHandle, data, 32,
                        USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING);

//------------------------------------------------------
// In this example we want to place a request for a 70 byte transfer.
// The 70 bytes will be sent out in a 64 byte transaction and a 6 byte
// transaction completing the transfer.

writeRequestResult = USB_DEVICE_CDC_Write(instanceHandle,
                        &transferHandle, data, 70,
                        USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
    //Do Error handling here
}

//------------------------------------------------------
// In this example we want to place a request for a 70 bytes and the flag
// is set to data pending. This will result in an error. The size of data
// when the data pending flag is specified should be a multiple of the
// endpoint size.

writeRequestResult = USB_DEVICE_CDC_Write(instanceHandle,
                        &transferHandle, data, 70,
                        USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING);

if(USB_DEVICE_CDC_RESULT_OK != writeRequestResult)
{
```

```
    //Do Error handling here
}

// The completion of the write request will be indicated by the
// USB_DEVICE_CDC_EVENT_WRITE_COMPLETE event.
```

Parameters

Parameters	Description
instance	USB Device CDC Function Driver instance.
transferHandle	Pointer to a USB_DEVICE_CDC_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the write request was successful.
data	pointer to the data buffer that contains the data to written.
size	Size of the data buffer. Refer to the description section for more details on how the size affects the transfer.
flags	Flags that indicate whether the transfer should continue or end. Refer to the description for more details.

Function

```
  USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_Write
(
    USB_DEVICE_CDC_INDEX instance,
USB_CDC_DEVICE_TRANSFER_HANDLE * transferHandle,
const void * data,
size_t size,
    USB_DEVICE_CDC_TRANSFER_FLAGS flags
);
```

USB_DEVICE_CDC_SerialStateNotificationSend Function

This function schedules a request to send serial state notification to the host.

File

usb_device_cdc.h

C

```
USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_SerialStateNotificationSend(USB_DEVICE_CDC_INDEX instanceIndex,
USB_DEVICE_CDC_TRANSFER_HANDLE * transferHandle, USB_CDC_SERIAL_STATE * notificationData);
```

Returns

USB_DEVICE_CDC_RESULT_OK - The request was successful. transferHandle contains a valid transfer handle.

USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL - Internal request queue is full. The request could not be added.

USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.

USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application and is invalid.

Description

This function places a request to send serial state notification data to the host. The function will place the request with the driver, the request will get serviced when the data is requested by the USB host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE event. The amount of data transmitted and the transfer handle associated with the request is returned along with the event in the serialStateNotificationCompleteData member of pData parameter of the event handler. The transfer handle expires when the event handler for the USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE event exits. If the send request could not be accepted, the function returns an error code and transferHandle will contain the value USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID.

Remarks

While the using the CDC Function Driver with the PIC32MZ USB module, the notification data buffer provided to the USB_DEVICE_CDC_SerialStateNotificationSend function should be placed in coherent memory and aligned at a 16 byte boundary. This can be done by declaring the buffer using the __attribute__((coherent, aligned(16))) attribute. An example is shown here

```
uint8_t data[256] __attribute__((coherent, aligned(16)));
```

Preconditions

The function driver should have been configured

Example

```
USB_CDC_SERIAL_STATE notificationData;

// This application function could possibly update the notificationData
// data structure.

APP_UpdateNotificationData(&notificationData);

// Now send the updated notification data to the host.

result = USB_DEVICE_CDC_SerialStateNotificationSend
            (instanceIndex, &transferHandle, &notificationData);

if(USB_DEVICE_CDC_RESULT_OK != result)
{
    // Error handling here. The transferHandle will contain
    // USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID in this case.
}
```

Parameters

Parameters	Description
instance	USB Device CDC Function Driver instance.
transferHandle	Pointer to a output only variable that will contain transfer handle.
notificationData	USB_DEVICE_CDC_SERIAL_STATE_NOTIFICATION type of notification data to be sent to the host.

Function

```
USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_SerialStateNotificationSend
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_TRANSFER_HANDLE * transferHandle,
USB_CDC_SERIAL_STATE * notificationData
);
```

b) Data Types and Constants

USB_DEVICE_CDC_EVENT Enumeration

USB Device CDC Function Driver Events

File

usb_device_cdc.h

C

```
typedef enum {
  USB_DEVICE_CDC_EVENT_SET_LINE_CODING,
  USB_DEVICE_CDC_EVENT_GET_LINE_CODING,
  USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE,
  USB_DEVICE_CDC_EVENT_SEND_BREAK,
  USB_DEVICE_CDC_EVENT_WRITE_COMPLETE,
  USB_DEVICE_CDC_EVENT_READ_COMPLETE,
  USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE,
  USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT,
  USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED,
  USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_ABORTED
} USB_DEVICE_CDC_EVENT;
```

Members

Members	Description
USB_DEVICE_CDC_EVENT_SET_LINE_CODING	This event occurs when the host issues a SET LINE CODING command. The application must provide a USB_CDC_LINE_CODING data structure to the device layer to receive the line coding data that the host will provide. The application must provide the buffer by calling the USB_DEVICE_ControlReceive function either in the event handler or in the application, after returning from the event handler function. The pData parameter will be NULL. The application can use the USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event to track completion of the command.
USB_DEVICE_CDC_EVENT_GET_LINE_CODING	This event occurs when the host issues a GET LINE CODING command. The application must provide a USB_CDC_LINE_CODING data structure to the device layer that contains the line coding data to be provided to the host. The application must provide the buffer by calling the USB_DEVICE_ControlSend function either in the event handler or in the application, after returning from the event handler function. The application can use the USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT event to track completion of the command.
USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE	This event occurs when the host issues a SET CONTROL LINE STATE command. The application must interpret the pData parameter as USB_CDC_CONTROL_LINE_STATE pointer type. This data structure contains the control line state data. The application can then use the USB_DEVICE_ControlStatus function to indicate acceptance or rejection of the command. The USB_DEVICE_ControlStatus function can be called from the event handler or in the application, after returning from the event handler.
USB_DEVICE_CDC_EVENT_SEND_BREAK	This event occurs when the host issues a SEND BREAK command. The application must interpret the pData parameter as a USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK pointer type. This data structure contains the break duration data. The application can then use the USB_DEVICE_ControlStatus function to indicate acceptance of rejection of the command. The USB_DEVICE_ControlStatus function can be called from the event handler or in the application, after returning from the event handler.
USB_DEVICE_CDC_EVENT_WRITE_COMPLETE	This event occurs when a write operation scheduled by calling the USB_DEVICE_CDC_Write function has completed. The pData parameter should be interpreted as a USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE pointer type. This will contain the transfer handle associated with the completed write transfer and the amount of data written.
USB_DEVICE_CDC_EVENT_READ_COMPLETE	This event occurs when a read operation scheduled by calling the USB_DEVICE_CDC_Read function has completed. The pData parameter should be interpreted as a USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE pointer type. This will contain the transfer handle associated with the completed read transfer and the amount of data read.
USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE	This event occurs when a serial state notification scheduled using the USB_DEVICE_CDC_SerialStateNotificationSend function, was sent to the host. The pData parameter should be interpreted as a USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE pointer type and will contain the transfer handle associated with the completed send transfer and the amount of data sent.
USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT	This event occurs when the data stage of a control read transfer has completed. This event would occur after the application uses the USB_DEVICE_ControlSend function to respond to the USB_DEVICE_CDC_EVENT_GET_LINE_CODING event.

USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	This event occurs when the data stage of a control write transfer has completed. This would occur after the application would respond with a USB_DEVICE_ControlReceive function to the USB_DEVICE_CDC_EVENT_SET_LINE_CODING_EVENT and the data has been received. The application should respond to this event by calling the USB_DEVICE_ControlStatus function with the USB_DEVICE_CONTROL_STATUS_OK flag to acknowledge the received data or the USB_DEVICE_CONTROL_STATUS_ERROR flag to reject it and stall the control transfer
USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_ABORTED	This event occurs when a control transfer that this instance of CDC function driver responded to was aborted by the host. The application can use this event to reset its CDC function driver related control transfer state machine

Description

USB Device CDC Function Driver Events

These events are specific to the USB Device CDC Function Driver instance. Each event description contains details about the parameters passed with event. The contents of pData depends on the generated event.

Events associated with the CDC Function Driver Specific Control Transfers require application response. The application should respond to these events by using the USB_DEVICE_ControlReceive, USB_DEVICE_ControlSend and USB_DEVICE_ControlStatus functions.

Calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_ERROR will stall the control transfer request. The application would do this if the control transfer request is not supported. Calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_OK will complete the status stage of the control transfer request. The application would do this if the control transfer request is supported

The following code snippet shows an example of a possible event handling scheme.

```
// This code example shows all CDC Function Driver events
// and a possible scheme for handling these events. In this example
// event responses are not deferred. usbDeviceHandle is obtained while
// opening the USB Device Layer through the USB_DEVICE_Open function.

uint16_t * breakData;
USB_DEVICE_HANDLE    usbDeviceHandle;
USB_CDC_LINE_CODING  lineCoding;
USB_CDC_CONTROL_LINE_STATE * controlLineStateData;

USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT event,
    void * pData,
    uintptr_t userData
)
{
    switch(event)
    {
        case USB_DEVICE_CDC_EVENT_SET_LINE_CODING:

            // In this case, the application should read the line coding
            // data that is sent by the host. The application must use the
            // USB_DEVICE_ControlReceive function to receive the
            // USB_CDC_LINE_CODING type of data.

            USB_DEVICE_ControlReceive(usbDeviceHandle, &lineCoding, sizeof(USB_CDC_LINE_CODING));
            break;

        case USB_DEVICE_CDC_EVENT_GET_LINE_CODING:

            // In this case, the application should send the line coding
            // data to the host. The application must send the
            // USB_DEVICE_ControlSend function to send the data.

            USB_DEVICE_ControlSend(usbDeviceHandle, &lineCoding, sizeof(USB_CDC_LINE_CODING));
            break;

        case USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE:

            // In this case, pData should be interpreted as a
```

```
            // USB_CDC_CONTROL_LINE_STATE pointer type.  The application
            // acknowledges the parameters by calling the
            // USB_DEVICE_ControlStatus function with the
            // USB_DEVICE_CONTROL_STATUS_OK option.

            controlLineStateData = (USB_CDC_CONTROL_LINE_STATE *)pData;
            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_CDC_EVENT_SEND_BREAK:

            // In this case, pData should be interpreted as a uint16_t
            // pointer type to the break duration. The application
            // acknowledges the parameters by calling the
            // USB_DEVICE_ControlStatus() function with the
            // USB_DEVICE_CONTROL_STATUS_OK option.

            breakDuration = (uint16_t *)pData;
            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT:

            // This event indicates the data send request associated with
            // the latest USB_DEVICE_ControlSend function was
            // completed.  The application could use this event to track
            // the completion of the USB_DEVICE_CDC_EVENT_GET_LINE_CODING
            // request.

            break;

        case USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:

            // This event indicates the data that was requested using the
            // USB_DEVICE_ControlReceive function is available for the
            // application to peruse. The application could use this event
            // to track the completion of the
            // USB_DEVICE_CDC_EVENT_SET_LINE_CODING_EVENT event. The
            // application can then either accept the line coding data (as
            // shown here) or decline it by using the
            // USB_DEVICE_CONTROL_STATUS_ERROR flag in the
            // USB_DEVICE_ControlStatus function.

            USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_CDC_EVENT_WRITE_COMPLETE:

            // This event indicates that a CDC Write Transfer request has
            // completed.  pData should be interpreted as a
            // USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE pointer type. This
            // contains the transfer handle of the write transfer that
            // completed and amount of data that was written.

            break;

        case USB_DEVICE_CDC_EVENT_READ_COMPLETE:

            // This event indicates that a CDC Read Transfer request has
            // completed.  pData should be interpreted as a
            // USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE pointer type. This
            // contains the transfer handle of the read transfer that
            // completed and amount of data that was written.

            break;

        case USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE:

            // This event indicates that a CDC Serial State Notification
```

```
            // Send request has completed. pData should be interpreted as a
            // USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE
            // pointer type. This will contain the transfer handle
            // associated with the send request and the amount of data that
            // was sent.

        break

         default:
            break;
      }

    return(USB_DEVICE_CDC_EVENT_RESPONSE_NONE);
}
```

Remarks

The USB Device CDC control transfer related events allow the application to defer responses. This allows the application some time to obtain the response data rather than having to respond to the event immediately. Note that a USB host will typically wait for event response for a finite time duration before timing out and canceling the event and associated transactions. Even when deferring response, the application must respond promptly if such time outs have to be avoided.

USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE Structure

USB Device CDC Function Driver Read and Write Complete Event Data.

File

usb_device_cdc.h

C

```
typedef struct {
  USB_DEVICE_CDC_TRANSFER_HANDLE handle;
  size_t length;
  USB_DEVICE_CDC_RESULT status;
} USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE, USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE,
USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE;
```

Members

Members	Description
USB_DEVICE_CDC_TRANSFER_HANDLE handle;	Transfer handle associated with this • read or write request
size_t length;	Indicates the amount of data (in bytes) that was • read or written
USB_DEVICE_CDC_RESULT status;	Completion status of the transfer

Description

USB Device CDC Function Driver Read and Write Complete Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_CDC_EVENT_READ_COMPLETE and USB_DEVICE_CDC_EVENT_WRITE_COMPLETE events.

Remarks

None.

USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE Structure

USB Device CDC Function Driver Read and Write Complete Event Data.

File

usb_device_cdc.h

C

```
typedef struct {
  USB_DEVICE_CDC_TRANSFER_HANDLE handle;
  size_t length;
  USB_DEVICE_CDC_RESULT status;
```

```
} USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE, USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE,
USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE;
```

Members

Members	Description
USB_DEVICE_CDC_TRANSFER_HANDLE handle;	Transfer handle associated with this • read or write request
size_t length;	Indicates the amount of data (in bytes) that was • read or written
USB_DEVICE_CDC_RESULT status;	Completion status of the transfer

Description

USB Device CDC Function Driver Read and Write Complete Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_CDC_EVENT_READ_COMPLETE and USB_DEVICE_CDC_EVENT_WRITE_COMPLETE events.

Remarks

None.

USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE Structure

USB Device CDC Function Driver Read and Write Complete Event Data.

File

usb_device_cdc.h

C

```
typedef struct {
  USB_DEVICE_CDC_TRANSFER_HANDLE handle;
  size_t length;
  USB_DEVICE_CDC_RESULT status;
} USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE, USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE,
USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE;
```

Members

Members	Description
USB_DEVICE_CDC_TRANSFER_HANDLE handle;	Transfer handle associated with this • read or write request
size_t length;	Indicates the amount of data (in bytes) that was • read or written
USB_DEVICE_CDC_RESULT status;	Completion status of the transfer

Description

USB Device CDC Function Driver Read and Write Complete Event Data.

This data type defines the data structure returned by the driver along with USB_DEVICE_CDC_EVENT_READ_COMPLETE and USB_DEVICE_CDC_EVENT_WRITE_COMPLETE events.

Remarks

None.

USB_DEVICE_CDC_EVENT_HANDLER Type

USB Device CDC Event Handler Function Pointer Type.

File

usb_device_cdc.h

C

```
typedef USB_DEVICE_CDC_EVENT_RESPONSE (* USB_DEVICE_CDC_EVENT_HANDLER)(USB_DEVICE_CDC_INDEX instanceIndex,
USB_DEVICE_CDC_EVENT event, void * pData, uintptr_t context);
```

Description

USB Device CDC Event Handler Function Pointer Type.

This data type defines the required function signature of the USB Device CDC Function Driver event handling callback function. The application must register a pointer to a CDC Function Driver events handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event call backs from the CDC Function Driver. The function driver will invoke this function with event relevant parameters. The description of the event handler function parameters is given here.

instanceIndex - Instance index of the CDC Function Driver that generated the event.

event - Type of event generated.

pData - This parameter should be type cast to an event specific pointer type based on the event that has occurred. Refer to the USB_DEVICE_CDC_EVENT enumeration description for more details.

context - Value identifying the context of the application that was registered along with the event handling function.

Remarks

The event handler function executes in the USB interrupt context when the USB Device Stack is configured for interrupt based operation. It is not advisable to call blocking functions or computationally intensive functions in the event handler. Where the response to a control transfer related event requires extended processing, the response to the control transfer should be deferred and the event handler should be allowed to complete execution.

USB_DEVICE_CDC_EVENT_RESPONSE Type

USB Device CDC Function Driver Event Callback Response Type

File

usb_device_cdc.h

C

```
typedef void USB_DEVICE_CDC_EVENT_RESPONSE;
```

Description

USB Device CDC Function Driver Event Handler Response Type

This is the return type of the CDC Function Driver event handler.

Remarks

None.

USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK Structure

USB Device CDC Function Driver Send Break Event Data

File

usb_device_cdc.h

C

```
typedef struct {
  uint16_t breakDuration;
} USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK;
```

Members

Members	Description
uint16_t breakDuration;	Duration of break signal

Description

USB Device CDC Function Driver Send Break Event Data

This data type defines the data structure returned by the driver along with USB_DEVICE_CDC_EVENT_SEND_BREAK event.

Remarks

None.

USB_DEVICE_CDC_INDEX Type

USB Device CDC Function Driver Index

File

usb_device_cdc.h

C

```
typedef uintptr_t USB_DEVICE_CDC_INDEX;
```

Description

USB Device CDC Function Driver Index

This uniquely identifies a CDC Function Driver instance.

Remarks

None.

USB_DEVICE_CDC_INIT Structure

USB Device CDC Function Driver Initialization Data Structure

File

usb_device_cdc.h

C

```
typedef struct {
  size_t queueSizeRead;
  size_t queueSizeWrite;
  size_t queueSizeSerialStateNotification;
} USB_DEVICE_CDC_INIT;
```

Members

Members	Description
size_t queueSizeRead;	Size of the read queue for this instance • of the CDC function driver
size_t queueSizeWrite;	Size of the write queue for this instance • of the CDC function driver
size_t queueSizeSerialStateNotification;	Size of the serial state notification • queue size

Description

USB Device CDC Function Driver Initialization Data Structure

This data structure must be defined for every instance of the CDC function driver. It is passed to the CDC function driver, by the Device Layer, at the time of initialization. The funcDriverInit member of the Device Layer Function Driver registration table entry must point to this data structure for an instance of the CDC function driver.

Remarks

The queue sizes that are specified in this data structure are also affected by the USB_DEVICE_CDC_QUEUE_DEPTH_COMBINED configuration macro.

USB_DEVICE_CDC_RESULT Enumeration

USB Device CDC Function Driver USB Device CDC Result enumeration.

File

usb_device_cdc.h

C

```
typedef enum {
  USB_DEVICE_CDC_RESULT_OK,
  USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_SIZE_INVALID,
  USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL,
  USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_INVALID,
  USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_NOT_CONFIGURED,
  USB_DEVICE_CDC_RESULT_ERROR_PARAMETER_INVALID,
  USB_DEVICE_CDC_RESULT_ERROR_ENDPOINT_HALTED,
```

```
  USB_DEVICE_CDC_RESULT_ERROR_TERMINATED_BY_HOST,
  USB_DEVICE_CDC_RESULT_ERROR
} USB_DEVICE_CDC_RESULT;
```

Members

Members	Description
USB_DEVICE_CDC_RESULT_OK	The operation was successful
USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_SIZE_INVALID	The transfer size is invalid. Refer to the description • of the read or write function for more details
USB_DEVICE_CDC_RESULT_ERROR_TRANSFER_QUEUE_FULL	The transfer queue is full and no new transfers can be • scheduled
USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_INVALID	The specified instance is not provisioned in the system
USB_DEVICE_CDC_RESULT_ERROR_INSTANCE_NOT_CONFIGURED	The specified instance is not configured yet
USB_DEVICE_CDC_RESULT_ERROR_PARAMETER_INVALID	The event handler provided is NULL
USB_DEVICE_CDC_RESULT_ERROR_ENDPOINT_HALTED	Transfer terminated because host halted the endpoint
USB_DEVICE_CDC_RESULT_ERROR_TERMINATED_BY_HOST	Transfer terminated by host because of a stall clear
USB_DEVICE_CDC_RESULT_ERROR	General CDC Function driver error

Description

USB Device CDC Function Driver USB Device CDC Result enumeration.

This enumeration lists the possible USB Device CDC Function Driver operation results. These values are returned by USB Device CDC Library functions.

Remarks

None.

USB_DEVICE_CDC_TRANSFER_FLAGS Enumeration

USB Device CDC Function Driver Transfer Flags

File

usb_device_cdc.h

C

```
typedef enum {
  USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE,
  USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING
} USB_DEVICE_CDC_TRANSFER_FLAGS;
```

Members

Members	Description
USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE	This flag indicates there is no further data to be sent in this transfer and that the transfer should end. If the size of the transfer is a multiple of the maximum packet size for related endpoint configuration, the function driver will send a zero length packet to indicate end of the transfer to the host.
USB_DEVICE_CDC_TRANSFER_FLAGS_MORE_DATA_PENDING	This flag indicates there is more data to be sent in this transfer. If the size of the transfer is a multiple of the maximum packet size for the related endpoint configuration, the function driver will not send a zero length packet. If the size of the transfer is greater than (but not a multiple of) the maximum packet size, the function driver will only send maximum packet size amount of data. If the size of the transfer is greater than endpoint size but not an exact multiple of endpoint size, only the closest endpoint size multiple bytes of data will be sent. This flag should not be specified if the size of the transfer is less than maximum packet size.

Description

USB Device CDC Transfer Flags

These flags are used to indicate status of the pending data while sending data to the host by using the USB_DEVICE_CDC_Write function.

Remarks

The relevance of the specified flag depends on the size of the buffer. Refer to the individual flag descriptions for more details.

USB_DEVICE_CDC_TRANSFER_HANDLE Type

USB Device CDC Function Driver Transfer Handle Definition.

File

usb_device_cdc.h

C

```
typedef uintptr_t USB_DEVICE_CDC_TRANSFER_HANDLE;
```

Description

USB Device CDC Function Driver Transfer Handle Definition

This definition defines a USB Device CDC Function Driver Transfer Handle. A Transfer Handle is owned by the application but its value is modified by the USB_DEVICE_CDC_Write, USB_DEVICE_CDC_Read and the USB_DEVICE_CDC_SerialStateNotificationSend functions. The transfer handle is valid for the life time of the transfer and expires when the transfer related event had occurred.

Remarks

None.

USB_DEVICE_CDC_EVENT_RESPONSE_NONE Macro

USB Device CDC Function Driver Event Handler Response Type None.

File

usb_device_cdc.h

C

```
#define USB_DEVICE_CDC_EVENT_RESPONSE_NONE
```

Description

USB Device CDC Function Driver Event Handler Response None

This is the definition of the CDC Function Driver Event Handler Response Type none.

Remarks

Intentionally defined to be empty.

USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID Macro

USB Device CDC Function Driver Invalid Transfer Handle Definition.

File

usb_device_cdc.h

C

```
#define USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID ((USB_DEVICE_CDC_TRANSFER_HANDLE)(-1))
```

Description

USB Device CDC Function Driver Invalid Transfer Handle Definition

This definition defines a USB Device CDC Function Driver Invalid Transfer Handle. A Invalid Transfer Handle is returned by the USB_DEVICE_CDC_Write, USB_DEVICE_CDC_Read and the USB_DEVICE_CDC_SerialStateNotificationSend functions when the request was not successful.

Remarks

None.

USB_DEVICE_CDC_FUNCTION_DRIVER Macro

USB Device CDC Function Driver Function pointer

File

usb_device_cdc.h

C

```
#define USB_DEVICE_CDC_FUNCTION_DRIVER
```

Description

USB Device CDC Function Driver Function Pointer

This is the USB Device CDC Function Driver Function pointer. This should registered with the device layer in the function driver registration table.

Remarks

None.

USB_DEVICE_CDC_INDEX_0 Macro

File

usb_device_cdc.h

C

```
#define USB_DEVICE_CDC_INDEX_0 0
```

Description

Use this to specify CDC Function Driver Instance 0

USB_DEVICE_CDC_INDEX_1 Macro

File

usb_device_cdc.h

C

```
#define USB_DEVICE_CDC_INDEX_1 1
```

Description

Use this to specify CDC Function Driver Instance 1

USB_DEVICE_CDC_INDEX_2 Macro

File

usb_device_cdc.h

C

```
#define USB_DEVICE_CDC_INDEX_2 2
```

Description

Use this to specify CDC Function Driver Instance 2

USB_DEVICE_CDC_INDEX_3 Macro

File

usb_device_cdc.h

C

```
#define USB_DEVICE_CDC_INDEX_3 3
```

Description

Use this to specify CDC Function Driver Instance 3

USB_DEVICE_CDC_INDEX_4 Macro

File

usb_device_cdc.h

C

```
#define USB_DEVICE_CDC_INDEX_4 4
```

Description

Use this to specify CDC Function Driver Instance 4

USB_DEVICE_CDC_INDEX_5 Macro

File

usb_device_cdc.h

C

```
#define USB_DEVICE_CDC_INDEX_5 5
```

Description

Use this to specify CDC Function Driver Instance 5

USB_DEVICE_CDC_INDEX_6 Macro

File

usb_device_cdc.h

C

```
#define USB_DEVICE_CDC_INDEX_6 6
```

Description

Use this to specify CDC Function Driver Instance 6

USB_DEVICE_CDC_INDEX_7 Macro

File

usb_device_cdc.h

C

```
#define USB_DEVICE_CDC_INDEX_7 7
```

Description

Use this to specify CDC Function Driver Instance 7

Files

Files

Name	Description
usb_device_cdc.h	USB Device CDC Function Driver Interface
usb_device_cdc_config_template.h	USB device CDC Class configuration definitions template

Description

This section lists the source and header files used by the library.

usb_device_cdc.h

USB Device CDC Function Driver Interface

Enumerations

	Name	Description
	USB_DEVICE_CDC_EVENT	USB Device CDC Function Driver Events
	USB_DEVICE_CDC_RESULT	USB Device CDC Function Driver USB Device CDC Result enumeration.

	USB_DEVICE_CDC_TRANSFER_FLAGS	USB Device CDC Function Driver Transfer Flags

Functions

	Name	Description
	USB_DEVICE_CDC_EventHandlerSet	This function registers a event handler for the specified CDC function driver instance.
	USB_DEVICE_CDC_Read	This function requests a data read from the USB Device CDC Function Driver Layer.
	USB_DEVICE_CDC_SerialStateNotificationSend	This function schedules a request to send serial state notification to the host.
	USB_DEVICE_CDC_Write	This function requests a data write to the USB Device CDC Function Driver Layer.

Macros

	Name	Description
	USB_DEVICE_CDC_EVENT_RESPONSE_NONE	USB Device CDC Function Driver Event Handler Response Type None.
	USB_DEVICE_CDC_FUNCTION_DRIVER	USB Device CDC Function Driver Function pointer
	USB_DEVICE_CDC_INDEX_0	Use this to specify CDC Function Driver Instance 0
	USB_DEVICE_CDC_INDEX_1	Use this to specify CDC Function Driver Instance 1
	USB_DEVICE_CDC_INDEX_2	Use this to specify CDC Function Driver Instance 2
	USB_DEVICE_CDC_INDEX_3	Use this to specify CDC Function Driver Instance 3
	USB_DEVICE_CDC_INDEX_4	Use this to specify CDC Function Driver Instance 4
	USB_DEVICE_CDC_INDEX_5	Use this to specify CDC Function Driver Instance 5
	USB_DEVICE_CDC_INDEX_6	Use this to specify CDC Function Driver Instance 6
	USB_DEVICE_CDC_INDEX_7	Use this to specify CDC Function Driver Instance 7
	USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID	USB Device CDC Function Driver Invalid Transfer Handle Definition.

Structures

	Name	Description
	USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE	USB Device CDC Function Driver Read and Write Complete Event Data.
	USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK	USB Device CDC Function Driver Send Break Event Data
	USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE	USB Device CDC Function Driver Read and Write Complete Event Data.
	USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE	USB Device CDC Function Driver Read and Write Complete Event Data.
	USB_DEVICE_CDC_INIT	USB Device CDC Function Driver Initialization Data Structure

Types

	Name	Description
	USB_DEVICE_CDC_EVENT_HANDLER	USB Device CDC Event Handler Function Pointer Type.
	USB_DEVICE_CDC_EVENT_RESPONSE	USB Device CDC Function Driver Event Callback Response Type
	USB_DEVICE_CDC_INDEX	USB Device CDC Function Driver Index
	USB_DEVICE_CDC_TRANSFER_HANDLE	USB Device CDC Function Driver Transfer Handle Definition.

Description

USB Device CDC Function Driver Interface

This file describes the USB Device CDC Function Driver interface. The application should include this file if it needs to use the CDC Function Driver API.

File Name

usb_device_cdc.h

Company

Microchip Technology Inc.

usb_device_cdc_config_template.h

USB device CDC Class configuration definitions template

Macros

	Name	Description
	USB_DEVICE_CDC_INSTANCES_NUMBER	Specifies the number of CDC instances.
	USB_DEVICE_CDC_QUEUE_DEPTH_COMBINED	Specifies the combined queue size of all CDC instances.

Description

USB Device CDC Class Configuration Definitions

This file contains configurations macros needed to configure the CDC Function Driver. This file is a template file only. It should not be included by the application. The configuration macros defined in the file should be defined in the configuration specific system_config.h.

File Name

usb_device_cdc_config_template.h

Company

Microchip Technology Inc.

USB HID Device Library

This section describes the USB HID Device Library.

Introduction

Introduces the MPLAB Harmony USB Human Interface Device (HID) Device Library.

Description

The MPLAB Harmony USB Human Interface Device (HID) Device Library (also referred to as the HID Function Driver or Library) provides a high-level abstraction of the Human Interface Device (HID) class under the Universal Serial Bus (USB) communication with a convenient C language interface. This library supports revision 1.11 of the USB HID specification released by the USB Implementers forum. This library is part of the MPLAB Harmony USB Device stack.

The USB HID Device Class supports devices that are used by humans to control the operation of computer systems. The HID class of devices include a wide variety of human interface, data indicator, and data feedback devices with various types of output directed to the end user. Some common examples of HID class devices include:

- Keyboards
- Pointing devices such as a standard mouse, joysticks, and trackballs
- Front-panel controls like knobs, switches, buttons, and sliders
- Controls found on telephony, gaming or simulation devices such as steering wheels, rudder pedals, and dial pads
- Data devices such as bar-code scanners, thermometers, analyzers

The USB HID Device Library offers services to the application to interact and respond to the host requests. Additional information about the HID class can be obtained from the HID specification available from the USB Implementers Forum at: www.usbif.org.

Using the Library

This topic describes the basic architecture of the HID Function Driver and provides information and examples on its use.

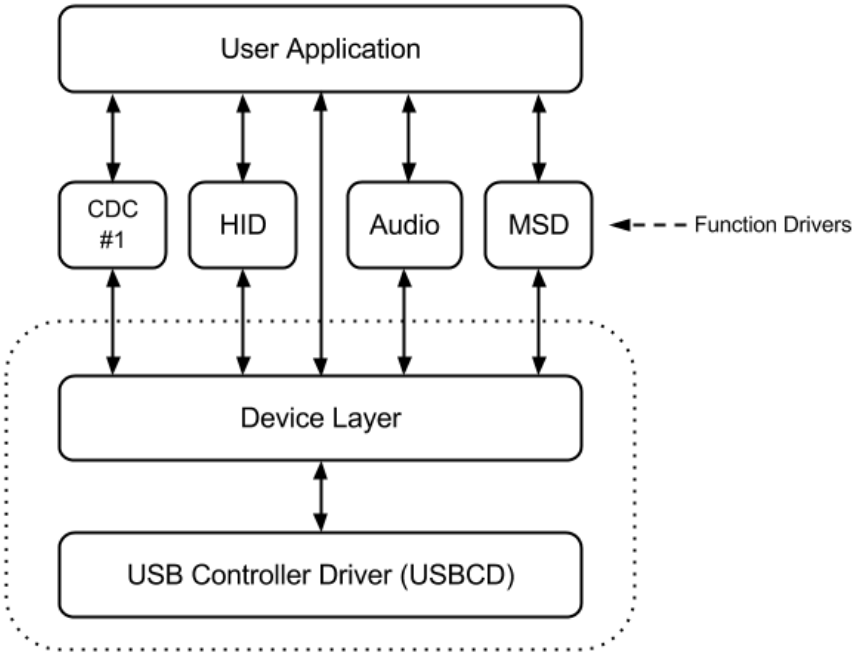
Abstraction Model

Provides an architectural overview of the USB HID Function Driver.

Description

The HID Function Driver offers services to a USB HID device to communicate with the host by abstracting the HID specification details. It must be used along with the USB Device Layer and USB Controller Driver to communicate with the USB Host. Figure 1 shows a block diagram of the MPLAB Harmony USB Architecture and where the HID Function Driver is placed.

Figure 1: HID Function Driver



The HID Function Driver together with USB Device Layer and the USB Controller Driver forms the basic library entity through which a HID device can communicate with the USB Host. The USB Controller Driver takes the responsibility of managing the USB peripheral on the device. The USB Device Layer handles the device enumeration, etc. The USB Device layer forwards all HID-specific control transfers to the HID Function Driver. The HID Function Driver interprets the control transfers and requests application's intervention through event handlers and a well-defined set of API functions. The application must register a event handler with the HID Function Driver in the Device Layer Set Configuration Event. While the application must respond to the HID Function Driver events, it can do this either in the HID Function Driver event handler or after the event handler routine has returned. The application interacts with HID Function Driver routines to send and receive HID reports over the USB.

Figure 2 shows the architecture of the HID Function Driver. The HID Function Driver maintains the state of each instance. It receives HID class-specific control transfers from the USB Device Layer. Class-specific control transfers that require application response are forwarded to the application as function driver events. The application responds to these class specific control transfer event by directly calling Device Layer control transfer routines. Depending on the type of device, the HID Function Driver can use the control endpoint and/or interrupt endpoints for data transfers. The USB HID Device Driver exchanges data with the Host through data objects called reports. The report data format is described by the HID report descriptor, which is provided to the Host when requested. Refer to the HID specification available from www.usb.org for more details on the USB HID Device class and how report descriptors can be created. The HID Function Driver allows report descriptors to be specified for every instance. This allow the application to implement a composite HID device.

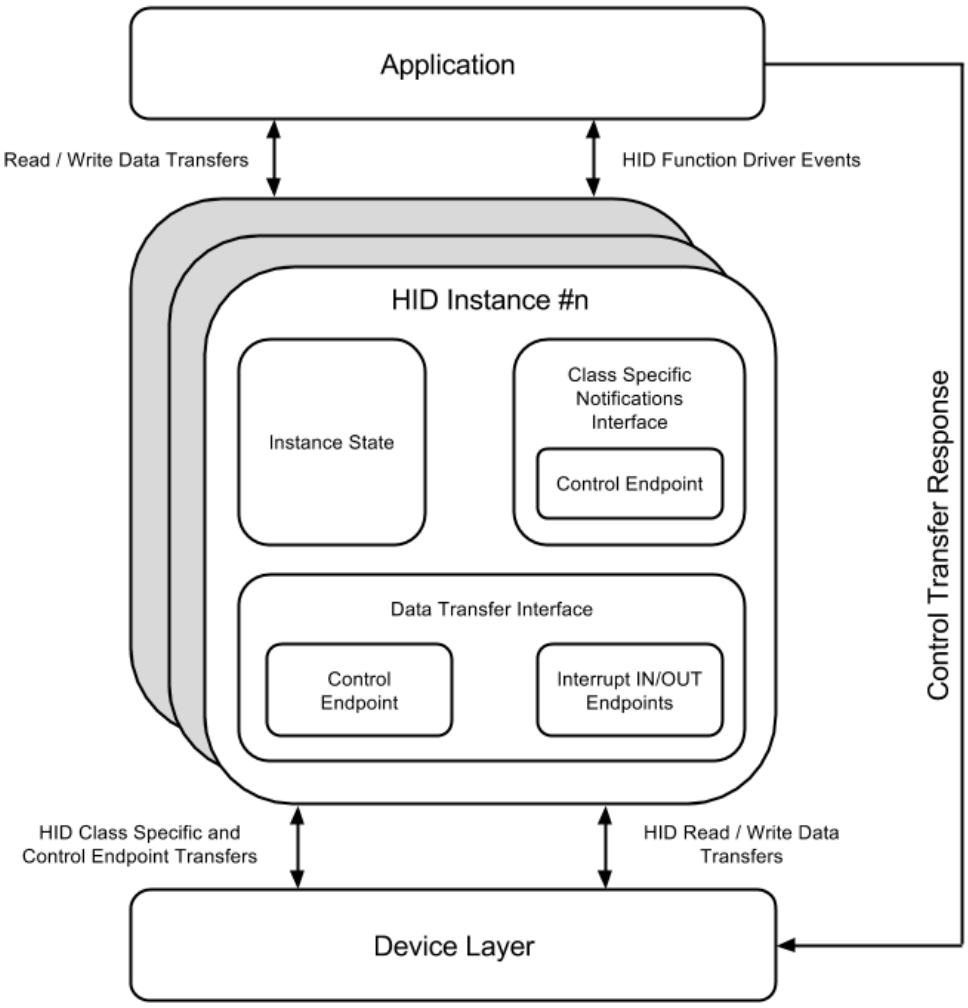


Figure 2: Architecture of the HID Function Driver

Library Overview

The USB HID Device Library mainly interacts with the system, its clients and function drivers, as shown in the Abstraction Model.

The library interface routines are divided into sub-sections, which address one of the blocks or the overall operation of the USB HID Device Library.

Library Interface Section	Description
Functions	Provides event handler, report send/receive, and transfer cancellation functions.

How the Library Works

Library Initialization

Describes how the HID Function Driver is initialized.

Description

The HID Function Driver instance for a USB device configuration is initialized by the Device Layer when the configuration is set by the Host. This process does not require application intervention. Each instance of the HID Function be registered with the Device Layer through the Device Layer Function Driver Registration Table. The HID function driver requires a initialization data structure that contains details about the report descriptor and the reports send/receive queue size associated with the specific instance of the HID Function Driver. The funcDriver member of the registration entry must be set to USB_DEVICE_HID_FUNCTION_DRIVER. This object is a global object provided by the HID Function Driver and points to the HID Function Driver - Device Layer interface functions, which are required by the Device Layer. The following code shows an

example of how a HID Function Driver instance (implementing a USB HID Mouse) can be registered with the Device Layer.

```
/* This code shows an example of registering a HID function driver
 * with the Device Layer. While registering the function driver, an initialization
 * data structure must be specified. In this example, hidInit is the HID function
 * driver initialization data structure. */

/* This hid_rpt01 report descriptor describes a 3 button 2
 * axis mouse pointing device */
const uint8_t hid_rpt01[]=
{
    0x06, 0x00, 0xFF,    // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,          // Usage (Vendor Usage 1)
    0xA1, 0x01,          // Collection (Application)
    0x19, 0x01,          // Usage Minimum
    0x29, 0x40,          // Usage Maximum     //64 input usages total (0x01 to 0x40)
    0x15, 0x01,          // Logical Minimum (data bytes in the report may have minimum value = 0x00)
    0x25, 0x40,          // Logical Maximum (data bytes in the report may have maximum value = 0x00FF =
unsigned 255)
    0x75, 0x08,          // Report Size: 8-bit field size
    0x95, 0x40,          // Report Count: Make sixty-four 8-bit fields (the next time the parser hits an
"Input", "Output",
                         // or "Feature" item)
    0x81, 0x00,          // Input (Data, Array, Abs): Instantiates input packet fields based on the
previous report size,
                         // count, logical min/max, and usage.
    0x19, 0x01,          // Usage Minimum
    0x29, 0x40,          // Usage Maximum     //64 output usages total (0x01 to 0x40)
    0x91, 0x00,          // Output (Data, Array, Abs): Instantiates output packet fields.  Uses same report
size and
                         // count as "Input" fields, since nothing new or different was specified to the
parser since
                         // the "Input" item.
    0xC0                 // End Collection
};

/* HID Function Driver Initialization data structure. This
 * contains the size of the report descriptor and a pointer
 * to the report descriptor. If there are multiple HID instances
 * each with different report descriptors, multiple such data
 * structures may be needed */

USB_DEVICE_HID_INITIALIZATION hidInit =
{
    sizeof(hid_rpt01), // Size of the report
    (uint8_t *)&hid_rpt01 // Pointer to the report
    1, // Send queue size is 1. We will not queue up reports.
    0 // Receive queue size 0. We will not receive reports.
};

/* The HID function driver instance is now registered with
 * device layer through the function driver registration
 * table. */

const USB_DEVICE_FUNCTION_REGISTRATION_TABLE funcRegistrationTable[1] =
{
    {
         .speed = USB_SPEED_FULL|USB_SPEED_HIGH,    // Supported speed
         .configurationValue = 1,                   // To be initialized for Configuration 1
         .interfaceNumber = 0,                      // Starting interface number
         .numberOfInterfaces = 1,                   // Number of Interfaces
         .funcDriverIndex = 0,                      // Function Driver instance index is 0
         .funcDriverInit = &hidInit,                // Function Driver Initialization
         .driver = USB_DEVICE_HID_FUNCTION_DRIVER   // Pointer to the function driver - Device Layer
Interface functions
     }
};
```

Event Handling

Describes HID Function Driver event handler registration and event handling.

Description

Registering a HID Function Driver Event Handler

While creating a USB HID Device-based application, an event handler must be registered with the Device Layer (the Device Layer Event Handler) and every HID Function Driver instance (HID Function Driver Event Handler). The HID Function Driver event handler receives HID events. This event handler should be registered before the USB Device Layer acknowledges the SET CONFIGURATION request from the USB Host. To ensure this, the event handler should be set in the USB_DEVICE_EVENT_CONFIGURED event that is generated by the device layer. While registering the HID Function Driver event handler, the HID Function Driver allows the application to also pass a data object in the event handler register function. This data object gets associated with the instance of the HID Function Driver and is returned by the driver when a HID Function Driver event occurs. The following code shows an example of how this can be done.

```
/* This is a sample Application Device Layer Event Handler
* Note how the HID Function Driver event handler APP_USBDeviceHIDEventHandler()
* is registered in the USB_DEVICE_EVENT_CONFIGURED event. The appData
* object that is passed in the USB_DEVICE_HID_EventHandlerSet()
* function will be returned as the userData parameter in the
* when the APP_USBDeviceHIDEventHandler() function is invoked */

   /* Application states */
    typedef enum
    {
        /* Application's state machine's initial state. */
        APP_STATE_INIT=0,
        APP_STATE_SERVICE_TASKS,
        APP_STATE_WAIT_FOR_CONFIGURATION,
    } APP_STATES;

    USB_DEVICE_HANDLE usbDeviceHandle;
    APP_STATES appState;

    /* This is the application device layer event handler function. */

    USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler
    (
        USB_DEVICE_EVENT event,
        void * eventData,
        uintptr_t context
    )
    {
        uint8_t activeConfiguration;
        USB_SETUP_PACKET * setupPacket;
        switch(event)
        {
            case USB_DEVICE_EVENT_POWER_DETECTED:
                /* This event in generated when VBUS is detected. Attach the device */
                USB_DEVICE_Attach(usbDeviceHandle);
                break;

            case USB_DEVICE_EVENT_POWER_REMOVED:
                /* This event is generated when VBUS is removed. Detach the device */
                USB_DEVICE_Detach (usbDeviceHandle);
                break;

            case USB_DEVICE_EVENT_CONFIGURED:
                /* This event indicates that Host has set Configuration in the Device.*/
                /* Check the configuration */
                activeConfiguration = ((USB_DEVICE_EVENT_DATA_CONFIGURED *)pData)->configurationValue;
                if ( activeConfiguration == 1)
                {
                   /* Register the HID Device application event handler here.
                    * Note how the appData object pointer is passed as the
                    * user data */
                    USB_DEVICE_HID_EventHandlerSet(USB_DEVICE_HID_INDEX_0, APP_USBDeviceHIDEventHandler,
(uintptr_t)&appData);
```

```
                    /* Mark that set configuration is complete */
                    appData.isConfigured = true;
                }
                break;

            case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST:
                /* This event indicates a Control transfer setup stage has been completed. */
                setupPacket = (USB_SETUP_PACKET *)eventData;

                /* Parse the setup packet and respond with a USB_DEVICE_ControlSend(),
                   USB_DEVICE_ControlReceive or USB_DEVICE_ControlStatus(). */

                break;

            case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT:
                /* This event indicates that a Control transfer Data has been sent to Host. */
                break;

            case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:
                /* This event indicates that a Control transfer Data has been received from Host. */
                break;

            case USB_DEVICE_EVENT_CONTROL_TRANSFER_ABORTED:
                /* This event indicates a control transfer was aborted. */
                break;

            case USB_DEVICE_EVENT_SUSPENDED:
                break;

            case USB_DEVICE_EVENT_RESUMED:
                break;

            case USB_DEVICE_EVENT_ERROR:
                break;

            case USB_DEVICE_EVENT_RESET:
                break;

            case USB_DEVICE_EVENT_SOF:
                /* This event indicates an SOF is detected on the bus. The USB_DEVICE_SOF_EVENT_ENABLE
                   macro should be defined to get this event. */
                break;
            default:
                break;
        }
    }
```

The HID Function Driver event handler executes in an interrupt context when the device stack is configured for Interrupt mode.

In Polled mode, the event handler is invoked in the context of the SYS_Tasks function. The application should not call computationally intensive functions, blocking functions, functions that are not interrupt safe, or functions that poll on hardware conditions from the event handler. Doing so will affect the ability of the USB device stack to respond to USB events and could potentially make the USB device non-compliant.

HID Function Driver Events:

The HID Function Driver generates events to which the application must respond. Some of these events are control requests communicated through control transfers. The application must therefore complete the control transfer. Based on the generated event, the application may be required to:

- Respond with a USB_DEVICE_ControlSend function, which completes the data stage of a Control Read Transfer
- Respond with a USB_DEVICE_ControlReceive function, which provisions the data stage of a Control Write Transfer
- Respond with a USB_DEVICE_ControlStatus function which completes the handshake stage of the Control Transfer. The application can either STALL or Acknowledge the handshake stage via the USB_DEVICE_HID_ControlStatus function.

The following table shows the HID Function Driver control transfer related events and the required application control transfer action.

HID Function Driver Control Transfer Event	Required Application Action
USB_DEVICE_HID_EVENT_GET_REPORT	Call USB_DEVICE_ControlSend function with a buffer containing the requested report.
USB_DEVICE_HID_EVENT_SET_REPORT	Call USB_DEVICE_ControlReceive function with a buffer to receive the report.

USB_DEVICE_HID_EVENT_SET_REPORT	Call the USB_DEVICE_ControlSend function with the pointer to the current USB_HID_PROTOCOL_CODE type data.
USB_DEVICE_HID_EVENT_SET_PROTOCOL	Acknowledge or stall using the USB_DEVICE_ControlStatus function.
USB_DEVICE_HID_EVENT_SET_IDLE	Acknowledge or stall using the USB_DEVICE_ControlStatus function.
USB_DEVICE_HID_EVENT_GET_IDLE	Call the USB_DEVICE_ControlSend function to send the current idle rate.
USB_DEVICE_HID_SET_DESCRIPTOR	Call the USB_DEVICE_ControlReceive function with a buffer to receive the report.
USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT	No action required.
USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	Acknowledge or stall using the USB_DEVICE_ControlStatus function.
USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_ABORTED	No action required.

The application can respond to HID Function Driver control transfer-related events in the function driver event handler. In a case where the data required for the response is not immediately available, the application can respond to the control transfer events after returning from the event handler. This defers the response to the control transfer event. However, please note that a USB host will typically wait for control transfer response for a finite time duration before timing out and canceling the transfer and associated transactions. Even when deferring response, the application must respond promptly if such timeouts have to be avoided.

The application should analyze the pData member of the event handler and check for event specific data. The following table shows the pData parameter data type for each HID function driver event.

Event Type	pData Parameter Data Type
USB_DEVICE_HID_EVENT_GET_REPORT	USB_DEVICE_HID_EVENT_DATA_GET_REPORT*
USB_DEVICE_HID_EVENT_SET_REPORT	USB_DEVICE_HID_EVENT_DATA_SET_REPORT *
USB_DEVICE_HID_EVENT_GET_IDLE	uint8_t*
USB_DEVICE_HID_EVENT_SET_IDLE	USB_DEVICE_HID_EVENT_DATA_SET_IDLE *
USB_DEVICE_HID_EVENT_SET_PROTOCOL	USB_HID_PROTOCOL_CODE*
USB_DEVICE_HID_EVENT_GET_PROTOCOL	NULL
USB_DEVICE_HID_EVENT_SET_DESCRIPTOR	USB_DEVICE_HID_EVENT_DATA_SET_DESCRIPTOR *
USB_DEVICE_HID_EVENT_REPORT_SENT	USB_DEVICE_HID_EVENT_DATA_REPORT_SENT *
USB_DEVICE_HID_EVENT_REPORT_RECEIVED	USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED *
USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT	NULL
USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	NULL
USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_ABORTED	NULL

The possible HID Function Driver events are described here along with the required application response, event specific data and likely follow up function driver event:

USB_DEVICE_HID_EVENT_GET_REPORT

Application Response: This event is generated when the USB HID Host is requesting a report over the control interface. The application must provide the report by calling the USB_DEVICE_HID_ControlSend function, either in the event handler, or in the application (after event handler function has exited). The application can use the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event to track completion of the command.

Event Specific Data (eventData): The application must interpret the pData parameter as a pointer to a

USB_DEVICE_HID_EVENT_DATA_GET_REPORT data type, which contains details about the requested report.

Likely Follow-up event: This event will likely be followed by the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event. This indicates that the data was sent to the Host successfully. The application must acknowledge the handshake stage of the control transfer by calling the USB_DEVICE_HID_ControlStatus function with the USB_DEVICE_HID_CONTROL_STATUS_OK flag.

USB_DEVICE_HID_EVENT_SET_REPORT

Application Response: This event is generated when the USB HID Host wants to send a report over the control interface. The application must provide a buffer to receive the report by calling the USB_DEVICE_HID_ControlReceive function either in the event handler or in the application (after the event handler function has exited). The application can use the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event to track completion of the command.

Event Specific Data (eventData): The application must interpret the pData parameter as a pointer to a

USB_DEVICE_HID_EVENT_DATA_SET_REPORT data type, which contains details about the report that the Host intends to send.

Likely Follow-up event: This event will likely be followed by the

USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event. This indicates that the data was received successfully. The application must either acknowledge or stall the handshake stage of the control transfer by calling the USB_DEVICE_HID_ControlStatus function with the USB_DEVICE_HID_CONTROL_STATUS_OK or

USB_DEVICE_HID_CONTROL_STATUS_ERROR flag, respectively.

USB_DEVICE_HID_EVENT_GET_IDLE

Application Response: This event is generated when the USB HID Host wants to read the current idle rate for the specified report. The application must provide the idle rate through the USB_DEVICE_HID_ControlSend function, either in the event handler, or in the application (after the event handler function has exited). The application must use the controlTransferHandle parameter provided in the event while calling the USB_DEVICE_HID_ControlSend function. The application can use the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event to track completion of the command.

Event Specific Data (eventData): The application must interpret the pData parameter as a pointer to a uint8_t data type, which contains a report ID of the report for which the idle rate is requested.

Likely Follow-up event: This event will likely be followed by the

USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event. This indicates that the data was sent to the Host successfully. The application must acknowledge the handshake stage of the control transfer by calling the USB_DEVICE_HID_ControlStatus function with the USB_DEVICE_HID_CONTROL_STATUS_OK flag.

USB_DEVICE_HID_EVENT_SET_IDLE

Application Response: This event is generated when the USB HID Host sends a Set Idle request to the device. The application must inspect the eventData and determine if the idle rate is to be supported. The application must either acknowledge (if the idle rate is supported) or stall the handshake stage of the control transfer (if the idle rate is not supported) by calling the USB_DEVICE_HID_ControlStatus function with the USB_DEVICE_HID_CONTROL_STATUS_OK or USB_DEVICE_HID_CONTROL_STATUS_ERROR flag, respectively.

Event Specific Data (eventData): The application must interpret the pData parameter as a pointer to a USB_DEVICE_HID_EVENT_DATA_SET_IDLE data type that contains details about the report ID and the idle duration.

Likely Follow-up event: None.

USB_DEVICE_HID_EVENT_SET_PROTOCOL

Application Response: This event is generated when the USB HID Host sends a Set Protocol request to the device . The application must inspect the eventData and determine if the protocol is to be supported. The application must either acknowledge (if the protocol is supported) or stall the handshake stage of the control transfer (if the protocol is not supported) by calling USB_DEVICE_HID_ControlStatus function with SB_DEVICE_HID_CONTROL_STATUS_OK or USB_DEVICE_HID_CONTROL_STATUS_ERROR flag, respectively.

Event Specific Data (eventData): The application must interpret the pData parameter as a pointer to a USB_HID_PROTOCOL_CODE data type that contains details about the protocol to be set.

Likely Follow-up event: None.

USB_DEVICE_HID_EVENT_GET_PROTOCOL

Application Response: This event is generated when the USB HID Host issues a Get Protocol Request. The application must provide the current protocol through the USB_DEVICE_HID_ControlSend function either in the event handler or in the application (after the event handler has exited). The application can use the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event to track completion of the command.

Event Specific Data (eventData): None.

Likely Follow-up event: This event will likely be followed by the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event. This indicates that the data was sent to the host successfully. The application must acknowledge the handshake stage of the control transfer by calling the USB_DEVICE_HID_ControlStatus function with the USB_DEVICE_HID_CONTROL_STATUS_OK flag.

USB_DEVICE_HID_EVENT_SET_DESCRIPTOR

Application Response: This event is generated when the HID Host issues a Set Descriptor request. The application must provide a buffer to receive the descriptor through the USB_DEVICE_HID_ControlReceive function, either in the event handler, or in the application (after the event handler has exited). The application can use the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event to track completion of the command.

Event Specific Data: None

Likely Follow-up event: This event will likely be followed by the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event. This indicates that the data was received successfully. The application must either acknowledge or stall the handshake stage of the control transfer by calling USB_DEVICE_HID_ControlStatus function with USB_DEVICE_HID_CONTROL_STATUS_OK or the USB_DEVICE_HID_CONTROL_STATUS_ERROR flag, respectively.

USB_DEVICE_HID_EVENT_REPORT_SENT

Application Response: This event occurs when a report send operation scheduled by calling the USB_DEVICE_HID_ReportSend function has completed. This event does not require the application to respond with any function calls.

Event Specific Data (pData): The application must interpret the pData parameter as a pointer to a

USB_DEVICE_HID_EVENT_DATA_REPORT_SENT data type that contains details about the report that was sent.

Likely Follow-up event: None.

USB_DEVICE_HID_EVENT_REPORT_RECEIVED

Application Response: This event occurs when a report receive operation scheduled by calling the USB_DEVICE_HID_ReportReceive function has completed. This event does not require the application to respond with any function calls.

Event Specific Data (pData): The application must interpret the pData parameter as a pointer to a USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED data type that contains details about the report that was received.

Likely Follow-up event: None

USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT

Application Response: This event occurs when the data stage of a control read transfer has completed in response to the USB_DEVICE_HID_ControlSend function. The application must acknowledge the handshake stage of the control transfer by calling the USB_DEVICE_HID_ControlStatus function with the USB_DEVICE_HID_CONTROL_STATUS_OK flag.

Event Specific Data (pData): None.

Likely Follow-up event: None.

USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED

Application Response: This event occurs when the data stage of a control write transfer has completed in response to the USB_DEVICE_HID_ControlReceive function. The application must either acknowledge or stall the handshake stage of the control transfer by calling USB_DEVICE_HID_ControlStatus function with the USB_DEVICE_HID_CONTROL_STATUS_OK or USB_DEVICE_HID_CONTROL_STATUS_ERROR flag, respectively.

Event Specific Data (pData): None

Likely Follow-up event: None.

USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_ABORTED

Application Response: This event occurs when the a control transfer request is aborted by the Host. The application can use this event to update its HID class-specific control transfer state machine.

Event Specific Data (pData): None

Likely Follow-up event: None. The following code shows an example HID Function Driver event handling scheme.

The following code shows an example HID Function Driver event handling scheme.

```
// This code example shows all USB HID Driver events and a possible
// scheme for handling these events. In this example event responses are not
// deferred.

   USB_DEVICE_HID_EVENT_RESPONSE USB_AppHIDEventHandler
   (
      USB_DEVICE_HID_INDEX instanceIndex,
      USB_DEVICE_HID_EVENT event,
      void * pData,
      uintptr_t userData
)
{
      uint8_t currentIdleRate;
      uint8_t someHIDReport[128];
      uint8_t someHIDDescriptor[128];
      USB_DEVICE_HANDLE       usbDeviceHandle;
      USB_HID_PROTOCOL_CODE * currentProtocol;
      USB_DEVICE_HID_EVENT_DATA_GET_REPORT        * getReportEventData;
      USB_DEVICE_HID_EVENT_DATA_SET_IDLE          * setIdleEventData;
      USB_DEVICE_HID_EVENT_DATA_SET_DESCRIPTOR    * setDescriptorEventData;
      USB_DEVICE_HID_EVENT_DATA_SET_REPORT        * setReportEventData;

      switch(event)
      {
         case USB_DEVICE_HID_EVENT_GET_REPORT:
            // In this case, pData should be interpreted as a
            // USB_DEVICE_HID_EVENT_DATA_GET_REPORT pointer. The application
            // must send the requested report by using the
            // USB_DEVICE_ControlSend() function.
            getReportEventData = (USB_DEVICE_HID_EVENT_DATA_GET_REPORT *)pData;
            USB_DEVICE_ControlSend(usbDeviceHandle, someHIDReport, getReportEventData->reportLength);
```

```
            break;

        case USB_DEVICE_HID_EVENT_GET_PROTOCOL:

            // In this case, pData will be NULL. The application
            // must send the current protocol to the host by using
            // the USB_DEVICE_ControlSend() function.
            USB_DEVICE_ControlSend(usbDeviceHandle, &currentProtocol, sizeof(USB_HID_PROTOCOL_CODE));

            break;
        case USB_DEVICE_HID_EVENT_GET_IDLE:

             // In this case, pData will be a uint8_t pointer type to the
             // report ID for which the idle rate is being requested. The
             // application must send the current idle rate to the host by
             // using the USB_DEVICE_ControlSend() function.
             USB_DEVICE_ControlSend(usbDeviceHandle, &currentIdleRate, 1);

            break;

        case USB_DEVICE_HID_EVENT_SET_REPORT:

            // In this case, pData should be interpreted as a
            // USB_DEVICE_HID_EVENT_DATA_SET_REPORT type pointer. The
            // application can analyze the request and then obtain the
            // report by using the USB_DEVICE_ControlReceive() function.
            setReportEventData = (USB_DEVICE_HID_EVENT_DATA_SET_REPORT *)pData;
            USB_DEVICE_ControlReceive(deviceHandle, someHIDReport, setReportEventData->reportLength);


            break;

        case USB_DEVICE_HID_EVENT_SET_PROTOCOL:

            // In this case, pData should be interpreted as a
            // USB_HID_PROTOCOL_CODE type pointer. The application can
            // analyze the data and decide to stall or accept the setting.
            // This shows an example of accepting the protocol setting.
            USB_DEVICE_ControlStatus(deviceHandle, USB_DEVICE_CONTROL_STATUS_OK);

            break;

        case USB_DEVICE_HID_EVENT_SET_IDLE:

            // In this case, pData should be interpreted as a
            // USB_DEVICE_HID_EVENT_DATA_SET_IDLE type pointer. The
            // application can analyze the data and decide to stall
            // or accept the setting. This shows an example of accepting
            // the protocol setting.
            setIdleEventData = (USB_DEVICE_HID_EVENT_DATA_SET_IDLE *)pData;
            USB_DEVICE_ControlStatus(deviceHandle, USB_DEVICE_CONTROL_STATUS_OK);


            break;

        case USB_DEVICE_HID_EVENT_SET_DESCRIPTOR:

            // In this case, the pData should be interpreted as a
            // USB_DEVICE_HID_EVENT_DATA_SET_DESCRIPTOR type pointer. The
            // application can analyze the request and then obtain the
            // descriptor by using the USB_DEVICE_ControlReceive() function.
            setDescriptorEventData = (USB_DEVICE_HID_EVENT_DATA_SET_DESCRIPTOR *)pData;
            USB_DEVICE_ControlReceive(deviceHandle, someHIDReport, setReportEventData->reportLength);

            break;

        case USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:

            // In this case, control transfer data was received. The
```

```
            // application can inspect that data and then stall the
            // handshake stage of the control transfer or accept it
            // (as shown here).
            USB_DEVICE_ControlStatus(deviceHandle, USB_DEVICE_CONTROL_STATUS_OK);

            break;

        case USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT:

            // This means that control transfer data was sent. The
            // application would typically acknowledge the handshake
            // stage of the control transfer.

            USB_DEVICE_HID_ControlStatus(instanceIndex, controlTransferHandle,
                  USB_DEVICE_HID_CONTROL_STATUS_OK);

            break;

        case USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_ABORTED:

             // This means that control transfer data was sent. The
             // application would typically acknowledge the handshake
             // stage of the control transfer.

            break;

        case USB_DEVICE_HID_EVENT_REPORT_RECEIVED:

            // This means a HID report receive request has completed.
            // The pData member should be interpreted as a
            // USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED pointer type.

            break;

        case USB_DEVICE_HID_EVENT_REPORT_SENT:

            // This means a HID report send request has completed.
            // The pData member should be interpreted as a
            // USB_DEVICE_HID_EVENT_DATA_REPORT_SENT pointer type.

            break;
    }
    return(USB_DEVICE_HID_EVENT_RESPONSE_NONE);
}
```

Sending a Report

Describes how to send a report.

Description

The USB HID Device sends data to the USB HID Host as reports. The USB HID Device application should use the USB_DEVICE_HID_ReportSend function to send the report. This function returns a transfer handler that allows the application to track the read request. The request is completed when the Host has requested the data. A report send request could fail if the driver instance transfer queue is full. The completion of the write transfer is indicated by a USB_DEVICE_HID_EVENT_REPORT_SENT event. The transfer handle and the amount of data sent is returned in the reportSent member of the eventData data structure along with the event.

The following code shows an example of how a USB HID Mouse application sends a report to the host.

```
/* In this code example, the application uses the
 * USB_HID_MOUSE_ReportCreate to create the mouse report
 * and then uses the USB_DEVICE_HID_ReportSend() function
 * to send the report */

USB_HID_MOUSE_ReportCreate(appData.xCoordinate, appData.yCoordinate,
    appData.mouseButton, &appData.mouseReport);

/* Send the mouse report. */
USB_DEVICE_HID_ReportSend(appData.hidInstance,
    &appData.reportTransferHandle, (uint8_t*)&appData.mouseReport,
    sizeof(USB_HID_MOUSE_REPORT));
```

Receiving a Report

Describes how to receive a report.

Description

The application can receive a report from the Host by using the USB_DEVICE_HID_ReportReceive function. This function returns a transfer handler that allows the application to track the read request. The request is completed when the Host sends the report. The application must make sure that it allocates a buffer size that is at least the size of the report. The return value of the function indicates the success of the request. A read request could fail if the driver transfer queue is full. The completion of the read transfer is indicated by a USB_DEVICE_HID_EVENT_REPORT_RECEIVED event. The reportReceived member of the eventData data structure contains details about the received report. The following code shows an example of how a USB HID Keyboard can schedule a receive report operation to get the keyboard LED status.

```
/* The following code shows how the
 * USB HID Keyboard application schedules a
 * receive report operation to receive the
 * keyboard output report from the host. This
 * report contains the keyboard LED status. The
 * size of the report is 1 byte */

result = USB_DEVICE_HID_ReportReceive(appData.hidInstance,
             &appData.receiveTransferHandle,
             (uint8_t *)&appData.keyboardOutputReport,1);

if(USB_DEVICE_HID_RESULT_OK != result)
{
    /* Do error handling here */
}
```

Configuring the Library

Describes how to configure the HID Function Driver.

Macros

	Name	Description
	USB_DEVICE_HID_INSTANCES_NUMBER	Specifies the number of HID instances.
	USB_DEVICE_HID_QUEUE_DEPTH_COMINED	DOM-IGONORE-BEGIN

Description

The following configuration parameters must be defined while using the HID Function Driver. The configuration macros that implement these parameters must be located in the `system_config.h` file in the application project and a compiler include path (to point to the folder that contains this file) should be specified.

USB_DEVICE_HID_INSTANCES_NUMBER Macro

Specifies the number of HID instances.

File

usb_device_hid_config_template.h

C

```
#define USB_DEVICE_HID_INSTANCES_NUMBER
```

Description

USB Device HID Maximum Number of Instances

This macro defines the number of instances of the HID Function Driver. For example, if the application needs to implement two instances of the HID Function Driver (to create composite device) on one USB Device, the macro should be set to 2. Note that implementing a USB Device that features multiple HID interfaces requires appropriate USB configuration descriptors.

Remarks

None.

USB_DEVICE_HID_QUEUE_DEPTH_COMINED Macro

File

usb_device_hid_config_template.h

C

```
#define USB_DEVICE_HID_QUEUE_DEPTH_COMINED
```

Description

DOM-IGONORE-BEGIN

Building the Library

Describes the files to be included in the project while using the HID Function Driver.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/usb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`usb_device_hid.h`	This header file should be included in any `.c` file that accesses the USB Device HID Function Driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/usb_device_hid.c`	This file implements the HID Function driver interface and should be included in the project if the HID Device function is desired.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The USB HID Device Library depends on the following modules:

- USB Device Layer Library

Library Interface

a) Functions

	Name	Description
	USB_DEVICE_HID_EventHandlerSet	This function registers a event handler for the specified HID function driver instance.
	USB_DEVICE_HID_ReportReceive	This function submits the buffer to HID function driver library to receive a report from host to device.
	USB_DEVICE_HID_ReportSend	This function submits the buffer to HID function driver library to send a report from device to host.
	USB_DEVICE_HID_TransferCancel	This function cancels a scheduled HID Device data transfer.

b) Data Types and Constants

	Name	Description
	USB_DEVICE_HID_EVENT	USB Device HID Function Driver Events
	USB_DEVICE_HID_EVENT_DATA_GET_REPORT	USB Device HID Get Report Event Data Type.
	USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED	USB Device HID Report Received Event Data Type.
	USB_DEVICE_HID_EVENT_DATA_REPORT_SENT	USB Device HID Report Sent Event Data Type.
	USB_DEVICE_HID_EVENT_DATA_SET_IDLE	USB Device HID Set Idle Event Data Type.
	USB_DEVICE_HID_EVENT_DATA_SET_REPORT	USB Device HID Set Report Event Data Type.
	USB_DEVICE_HID_INDEX	USB device HID Function Driver Index.
	USB_DEVICE_HID_EVENT_DATA_GET_IDLE	USB Device HID Get Idle Event Data Type.
	USB_DEVICE_HID_TRANSFER_HANDLE	USB Device HID Function Driver Transfer Handle Definition.
	USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL	USB Device HID Set Protocol Event Data Type.
	USB_DEVICE_HID_EVENT_HANDLER	USB Device HID Event Handler Function Pointer Type.
	USB_DEVICE_HID_EVENT_RESPONSE	USB Device HID Function Driver Event Callback Response Type
	USB_DEVICE_HID_INIT	USB Device HID Function Driver Initialization Data Structure
	USB_DEVICE_HID_RESULT	USB Device HID Function Driver USB Device HID Result enumeration.
	USB_DEVICE_HID_EVENT_RESPONSE_NONE	USB Device HID Function Driver Event Handler Response Type None.
	USB_DEVICE_HID_TRANSFER_HANDLE_INVALID	USB Device HID Function Driver Invalid Transfer Handle Definition.
	USB_DEVICE_HID_FUNCTION_DRIVER	This is a pointer to a group of HID Function Driver callback function pointers.
	USB_DEVICE_HID_INDEX_0	USB Device HID Function Driver Index Constants
	USB_DEVICE_HID_INDEX_1	This is macro USB_DEVICE_HID_INDEX_1.
	USB_DEVICE_HID_INDEX_2	This is macro USB_DEVICE_HID_INDEX_2.
	USB_DEVICE_HID_INDEX_3	This is macro USB_DEVICE_HID_INDEX_3.
	USB_DEVICE_HID_INDEX_4	This is macro USB_DEVICE_HID_INDEX_4.
	USB_DEVICE_HID_INDEX_5	This is macro USB_DEVICE_HID_INDEX_5.
	USB_DEVICE_HID_INDEX_6	This is macro USB_DEVICE_HID_INDEX_6.
	USB_DEVICE_HID_INDEX_7	This is macro USB_DEVICE_HID_INDEX_7.

Description

This section describes the Application Programming Interface (API) functions of the USB Device HID library.

Refer to each section for a detailed description.

a) Functions

USB_DEVICE_HID_EventHandlerSet Function

This function registers a event handler for the specified HID function driver instance.

File

usb_device_hid.h

C

```
USB_DEVICE_HID_RESULT USB_DEVICE_HID_EventHandlerSet(USB_DEVICE_HID_INDEX instanceIndex,
USB_DEVICE_HID_EVENT_HANDLER eventHandler, uintptr_t context);
```

Returns

USB_DEVICE_HID_RESULT_OK - The operation was successful

USB_DEVICE_HID_RESULT_ERROR_INSTANCE_INVALID - The specified instance does not exist.

USB_DEVICE_HID_RESULT_ERROR_PARAMETER_INVALID - The eventHandler parameter is NULL

Description

This function registers a event handler for the specified HID function driver instance. This function should be called by the client when it receives a SET CONFIGURATION event from the device layer. A event handler must be registered for function driver to respond to function driver specific commands. If the event handler is not registered, the device layer will stall function driver specific commands and the USB device may not function.

Remarks

None.

Preconditions

This function should be called when the function driver has been initialized as a result of a set configuration.

Example

```
// This code snippet shows an example registering an event handler. Here
// the application specifies the context parameter as a pointer to an
// application object (appObject) that should be associated with this
// instance of the HID function driver.

USB_DEVICE_HID_RESULT result;

USB_DEVICE_HID_EVENT_RESPONSE APP_USBDeviceHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    void * pData,
    uintptr_t context
)
{
    // Event Handling comes here

    switch(event)
    {
        ...
    }

    return(USB_DEVICE_HID_EVENT_RESPONSE_NONE);
}

result = USB_DEVICE_HID_EventHandlerSet (0, &APP_EventHandler, (uintptr_t) &appObject);

if(USB_DEVICE_HID_RESULT_OK != result)
{
    SYS_ASSERT ( false , "Error while registering event handler" );
}
```

Parameters

Parameters	Description
instance	Instance of the HID Function Driver.
eventHandler	A pointer to event handler function.
context	Application specific context that is returned in the event handler.

Function

USB_DEVICE_HID_RESULT USB_DEVICE_HID_EventHandlerSet

(

USB_DEVICE_HID_INDEX instance

USB_DEVICE_HID_EVENT_HANDLER eventHandler

uintptr_t context

);

USB_DEVICE_HID_ReportReceive Function

This function submits the buffer to HID function driver library to receive a report from host to device.

File

usb_device_hid.h

C

```
USB_DEVICE_HID_RESULT USB_DEVICE_HID_ReportReceive(USB_DEVICE_HID_INDEX instanceIndex,
USB_DEVICE_HID_TRANSFER_HANDLE * handle, void * buffer, size_t size);
```

Returns

USB_DEVICE_HID_RESULT_OK - The receive request was successful. transferHandle contains a valid transfer handle.

USB_DEVICE_HID_RESULT_ERROR_TRANSFER_QUEUE_FULL - internal request queue is full. The receive request could not be added.

USB_DEVICE_HID_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.

USB_DEVICE_HID_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application and is invalid.

Description

This function submits the buffer to HID function driver library to receive a report from host to device. On completion of the transfer the library generates USB_DEVICE_HID_EVENT_REPORT_RECEIVED event to the application. A handle to the request is passed in the transferHandle parameter. The transfer handle expires when event handler for the USB_DEVICE_HID_EVENT_REPORT_RECEIVED exits. If the receive request could not be accepted, the function returns an error code and transferHandle will contain the value USB_DEVICE_HID_TRANSFER_HANDLE_INVALID.

Remarks

While the using the HID Function Driver with the PIC32MZ USB module, the report data buffer provided to the USB_DEVICE_HID_ReportReceive function should be placed in coherent memory and aligned at a 16 byte boundary. This can be done by declaring the buffer using the __attribute__((coherent, aligned(16))) attribute. An example is shown here

```
uint8_t data[256] __attribute__((coherent, aligned(16)));
```

Preconditions

USB device layer must be initialized.

Example

```
USB_DEVICE_HID_TRANSFER_HANDLE hidTransferHandle;
USB_DEVICE_HID_RESULT result;

// Register APP_HIDEventHandler function
USB_DEVICE_HID_EventHandlerSet( USB_DEVICE_HID_INDEX_0 ,
                                APP_HIDEventHandler );

// Prepare report and request HID to send the report.
result = USB_DEVICE_HID_ReportReceive( USB_DEVICE_HID_INDEX_0,
                           &hidTransferHandle ,
                           &appReport[0], sizeof(appReport));

if( result != USB_DEVICE_HID_RESULT_OK)
{
   //Handle error.

}

//Implementation of APP_HIDEventHandler

USB_DEVICE_HIDE_EVENT_RESPONSE APP_HIDEventHandler
{
  USB_DEVICE_HID_INDEX instanceIndex,
  USB_DEVICE_HID_EVENT event,
  void * pData,
  uintptr_t context
}
{
    USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED reportReceivedEventData;
    // Handle HID events here.
    switch (event)
    {
        case USB_DEVICE_HID_EVENT_REPORT_RECEIVED:
           if( (reportReceivedEventData->reportSize == sizeof(appReport)
                && reportReceivedEventData->report == &appReport[0])
           {
              // Previous transfer was complete.
           }
           break;
        ....
    }
}
```

Parameters

Parameters	Description
instanceIndex	HID instance index.
transferHandle	HID transfer handle.
buffer	Pointer to buffer where the received report has to be received stored.
size	Buffer size.

Function

```
USB_DEVICE_HID_RESULT USB_DEVICE_HID_ReportReceive
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_TRANSFER_HANDLE * transferHandle,
void * buffer,
size_t size
);
```

USB_DEVICE_HID_ReportSend Function

This function submits the buffer to HID function driver library to send a report from device to host.

File

usb_device_hid.h

C

```
USB_DEVICE_HID_RESULT USB_DEVICE_HID_ReportSend(USB_DEVICE_HID_INDEX instanceIndex,
USB_DEVICE_HID_TRANSFER_HANDLE * handle, void * buffer, size_t size);
```

Returns

USB_DEVICE_HID_RESULT_OK - The send request was successful. transferHandle contains a valid transfer handle.

USB_DEVICE_HID_RESULT_ERROR_TRANSFER_QUEUE_FULL - Internal request queue is full. The send request could not be added.

USB_DEVICE_HID_RESULT_ERROR_INSTANCE_NOT_CONFIGURED - The specified instance is not configured yet.

USB_DEVICE_HID_RESULT_ERROR_INSTANCE_INVALID - The specified instance was not provisioned in the application and is invalid.

Description

This function places a request to send a HID report with the USB Device HID Function Driver Layer. The function places a requests with driver, the request will get serviced when report is requested by the USB Host. A handle to the request is returned in the transferHandle parameter. The termination of the request is indicated by the USB_DEVICE_HID_EVENT_REPORT_SENT event. The amount of data sent, a pointer to the report and the transfer handle associated with the request is returned along with the event in the pData parameter of the event handler. The transfer handle expires when event handler for the USB_DEVICE_HID_EVENT_REPORT_SENT exits. If the send request could not be accepted, the function returns an error code and transferHandle will contain the value USB_DEVICE_HID_TRANSFER_HANDLE_INVALID.

Remarks

While the using the HID Function Driver with the PIC32MZ USB module, the report data buffer provided to the USB_DEVICE_HID_ReportSend function should be placed in coherent memory and aligned at a 16 byte boundary. This can be done by declaring the buffer using the __attribute__((coherent, aligned(16))) attribute. An example is shown here

```
uint8_t data[256] __attribute__((coherent, aligned(16)));
```

Preconditions

USB device layer must be initialized.

Example

```
USB_DEVICE_HID_TRANSFER_HANDLE hidTransferHandle;
USB_DEVICE_HID_RESULT result;

// Register APP_HIDEventHandler function
USB_DEVICE_HID_EventHandlerSet( USB_DEVICE_HID_INDEX_0 ,
                                APP_HIDEventHandler );

// Prepare report and request HID to send the report.
result = USB_DEVICE_HID_ReportSend( USB_DEVICE_HID_INDEX_0,
                           &hidTransferHandle ,
```

```
                            &appReport[0], sizeof(appReport));

    if( result != USB_DEVICE_HID_RESULT_OK)
    {
       //Handle error.

    }

    //Implementation of APP_HIDEventHandler

    USB_DEVICE_HIDE_EVENT_RESPONSE APP_HIDEventHandler
    {
        USB_DEVICE_HID_INDEX instanceIndex,
        USB_DEVICE_HID_EVENT event,
        void * pData,
        uintptr_t context
    }
    {
        USB_DEVICE_HID_EVENT_DATA_REPORT_SENT * reportSentEventData;

        // Handle HID events here.
        switch (event)
        {
            case USB_DEVICE_HID_EVENT_REPORT_SENT:

                reportSentEventData = (USB_DEVICE_HID_EVENT_REPORT_SENT *)pData;
                if(reportSentEventData->reportSize == sizeof(appReport))
                {
                    // The report was sent completely.
                }
                break;

                ....

        }
        return(USB_DEVICE_HID_EVENT_RESPONSE_NONE);
    }
```

Parameters

Parameters	Description
instance	USB Device HID Function Driver instance.
transferHandle	Pointer to a USB_DEVICE_HID_TRANSFER_HANDLE type of variable. This variable will contain the transfer handle in case the send request was successful.
data	pointer to the data buffer containing the report to be sent.
size	Size (in bytes) of the report to be sent.

Function

USB_DEVICE_HID_RESULT USB_DEVICE_HID_ReportSend

(

USB_DEVICE_HID_INDEX instanceIndex,

USB_DEVICE_HID_TRANSFER_HANDLE * transferHandle,

void * buffer,

size_t size

)

USB_DEVICE_HID_TransferCancel Function

This function cancels a scheduled HID Device data transfer.

File

usb_device_hid.h

C

```
USB_DEVICE_HID_RESULT USB_DEVICE_HID_TransferCancel(USB_DEVICE_HID_INDEX usbDeviceHandle,
```

```
USB_DEVICE_HID_TRANSFER_HANDLE transferHandle);
```

Returns

USB_DEVICE_HID_RESULT_OK - The transfer will be canceled completely or partially.

USB_DEVICE_HID_RESULT_ERROR_PARAMETER_INVALID - Invalid transfer handle

USB_DEVICE_HID_RESULT_ERROR_INSTANCE_INVALID - Invalid HID instance index

USB_DEVICE_HID_RESULT_ERROR - The transfer could not be canceled because it has either completed, the transfer handle is invalid or the last transaction is in progress.

Description

This function cancels a scheduled HID Device data transfer. The transfer could have been scheduled using the USB_DEVICE_HID_ReportReceive, USB_DEVICE_HID_ReportSend function. If a transfer is still in the queue and its processing has not started, then the transfer is canceled completely. A transfer that is in progress may or may not get canceled depending on the transaction that is presently in progress. If the last transaction of the transfer is in progress, then the transfer will not be canceled. If it is not the last transaction in progress, the in-progress will be allowed to complete. Pending transactions will be canceled. The first transaction of an in progress transfer cannot be canceled.

Remarks

The buffer specific to the transfer handle should not be released unless the transfer abort event is notified through callback.

Preconditions

The USB Device should be in a configured state.

Example

```
// The following code snippet cancels a HID transfer.

USB_DEVICE_HID_TRANSFER_HANDLE transferHandle;
USB_DEVICE_HID_RESULT result;

result = USB_DEVICE_HID_TransferCancel(instanceIndex, transferHandle);

if(USB_DEVICE_HID_RESULT_OK == result)
{
    // The transfer cancellation was either completely or
    // partially successful.
}
```

Parameters

Parameters	Description
instanceIndex	HID Function Driver instance index.
transferHandle	Transfer handle of the transfer to be canceled.

Function

```
USB_DEVICE_HID_RESULT USB_DEVICE_HID_TransferCancel
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_TRANSFER_HANDLE transferHandle
);
```

b) Data Types and Constants

USB_DEVICE_HID_EVENT Enumeration

USB Device HID Function Driver Events

File

usb_device_hid.h

C

```
typedef enum {
  USB_DEVICE_HID_EVENT_GET_REPORT,
```

```
  USB_DEVICE_HID_EVENT_GET_IDLE,
  USB_DEVICE_HID_EVENT_GET_PROTOCOL,
  USB_DEVICE_HID_EVENT_SET_REPORT,
  USB_DEVICE_HID_EVENT_SET_IDLE,
  USB_DEVICE_HID_EVENT_SET_PROTOCOL,
  USB_DEVICE_HID_EVENT_REPORT_SENT,
  USB_DEVICE_HID_EVENT_REPORT_RECEIVED,
  USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED,
  USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT,
  USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_ABORTED
} USB_DEVICE_HID_EVENT;
```

Members

Members	Description
USB_DEVICE_HID_EVENT_GET_REPORT	This event occurs when the host issues a GET REPORT command. This is a HID class specific control transfer related event. The application must interpret the pData parameter as USB_DEVICE_HID_EVENT_DATA_GET_REPORT pointer type. If the report request is supported, the application must send the report to the host by using the USB_DEVICE_ControlSend function either in the event handler or after the event handler routine has returned. The application can track the completion of the request by using the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event. If the report request is not supported, the application must stall the request by calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_ERROR flag.
USB_DEVICE_HID_EVENT_GET_IDLE	This event occurs when the host issues a GET IDLE command. This is a HID class specific control transfer related event. The pData parameter will be a USB_DEVICE_HID_EVENT_DATA_GET_IDLE pointer type containing the ID of the report for which the idle parameter is requested. If the request is supported, the application must send the idle rate to the host by calling the USB_DEVICE_ControlSend function. This function can be called either in the event handler or after the event handler routine has returned. The application can track the completion of the request by using the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event. If the request is not supported, the application must stall the request by calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_ERROR flag.
USB_DEVICE_HID_EVENT_GET_PROTOCOL	This event occurs when the host issues a GET PROTOCOL command. This is a HID class specific control transfer related event. The pData parameter will be NULL. If the request is supported, the application must send a USB_HID_PROTOCOL_CODE data type object, containing the current protocol, to the host by calling the USB_DEVICE_ControlSend function. This function can be called either in the event handler or after the event handler routine has returned. The application can track the completion of the request by using the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT event. If the request is not supported, the application must stall the request by calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_ERROR flag.
USB_DEVICE_HID_EVENT_SET_REPORT	This event occurs when the host issues a SET REPORT command. This is a HID class specific control transfer related event. The application must interpret the pData parameter as a USB_DEVICE_HID_EVENT_DATA_SET_REPORT pointer type. If the report request is supported, the application must provide a buffer, to receive the report, to the host by calling the USB_DEVICE_ControlReceive function either in the event handler or after the event handler routine has returned. The application can track the completion of the request by using the USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event. If the report request is not supported, the application must stall the request by calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_ERROR flag.

USB_DEVICE_HID_EVENT_SET_IDLE	This event occurs when the host issues a SET IDLE command. This is a HID class specific control transfer related event. The pData parameter will be USB_DEVICE_HID_EVENT_DATA_SET_IDLE pointer type. The application can analyze the idle duration and acknowledge or reject the setting by calling the USB_DEVICE_ControlStatus function. This function can be called in the event handler or after the event handler exits. If application can reject the request by calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_ERROR flag. It can accept the request by calling this function with USB_DEVICE_CONTROL_STATUS_OK flag.
USB_DEVICE_HID_EVENT_SET_PROTOCOL	This event occurs when the host issues a SET PROTOCOL command. This is a HID class specific control transfer related event. The pData parameter will be a pointer to a USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL data type. If the request is supported, the application must acknowledge the request by calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_OK flag. If the request is not supported, the application must stall the request by calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_ERROR flag.
USB_DEVICE_HID_EVENT_REPORT_SENT	This event indicates that USB_DEVICE_HID_ReportSend function completed a report transfer on interrupt endpoint from host to device. The pData parameter will be a USB_DEVICE_HID_EVENT_DATA_REPORT_SENT type.
USB_DEVICE_HID_EVENT_REPORT_RECEIVED	This event indicates that USB_DEVICE_HID_ReportReceive function completed a report transfer on interrupt endpoint from device to host. The pData parameter will be a USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED type
USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED	This event occurs when the data stage of a control write transfer has completed. This happens after the application uses the USB_DEVICE_ControlReceive function to respond to a HID Function Driver Control Transfer Event that requires data to be received from the host. The pData parameter will be NULL. The application should call the USB_DEVICE_ControlStatus function with the USB_DEVICE_CONTROL_STATUS_OK flag if the received data is acceptable or should call this function with USB_DEVICE_CONTROL_STATUS_ERROR flag if the received data needs to be rejected.
USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT	This event occurs when the data stage of a control read transfer has completed. This happens after the application uses the USB_DEVICE_ControlSend function to respond to a HID Function Driver Control Transfer Event that requires data to be sent to the host. The pData parameter will be NULL
USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_ABORTED	This event occurs when an ongoing control transfer was aborted. The application must stop any pending control transfer related activities.

Description

USB Device HID Function Driver Events

These events are specific to the USB Device HID Function Driver instance. Each event description contains details about the parameters passed with event. The contents of pData depends on the generated event.

Events that are associated with the HID Function Driver Specific Control Transfers require application response. The application should respond to these events by using the USB_DEVICE_ControlReceive, USB_DEVICE_ControlSend and USB_DEVICE_ControlStatus functions.

Calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_ERROR will stall the control transfer request. The application would do this if the control transfer request is not supported. Calling the USB_DEVICE_ControlStatus function with a USB_DEVICE_CONTROL_STATUS_OK will complete the status stage of the control transfer request. The application would do this if the control transfer request is supported

The following code snippet shows an example of a possible event handling scheme.

```
// This code example shows all HID Function Driver events and a possible
// scheme for handling these events. In this example event responses are not
// deferred.

USB_DEVICE_HID_EVENT_RESPONSE USB_AppHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    void * pData,
    uintptr_t userData
)
```

```
{
    uint8_t currentIdleRate;
    uint8_t someHIDReport[128];
    uint8_t someHIDDescriptor[128];
    USB_DEVICE_HANDLE       usbDeviceHandle;
    USB_HID_PROTOCOL_CODE  currentProtocol;
    USB_DEVICE_HID_EVENT_DATA_GET_REPORT        * getReportEventData;
    USB_DEVICE_HID_EVENT_DATA_SET_IDLE          * setIdleEventData;
    USB_DEVICE_HID_EVENT_DATA_SET_DESCRIPTOR    * setDescriptorEventData;
    USB_DEVICE_HID_EVENT_DATA_SET_REPORT        * setReportEventData;

    switch(event)
    {
        case USB_DEVICE_HID_EVENT_GET_REPORT:

            // In this case, pData should be interpreted as a
            // USB_DEVICE_HID_EVENT_DATA_GET_REPORT pointer. The application
            // must send the requested report by using the
            // USB_DEVICE_ControlSend() function.

            getReportEventData = (USB_DEVICE_HID_EVENT_DATA_GET_REPORT *)pData;
            USB_DEVICE_ControlSend(usbDeviceHandle, someHIDReport, getReportEventData->reportLength);

            break;

        case USB_DEVICE_HID_EVENT_GET_PROTOCOL:

            // In this case, pData will be NULL. The application
            // must send the current protocol to the host by using
            // the USB_DEVICE_ControlSend() function.

           USB_DEVICE_ControlSend(usbDeviceHandle, &currentProtocol, sizeof(USB_HID_PROTOCOL_CODE));
           break;

        case USB_DEVICE_HID_EVENT_GET_IDLE:

            // In this case, pData will be a
            // USB_DEVICE_HID_EVENT_DATA_GET_IDLE pointer type containing the
            // ID of the report for which the idle rate is being requested.
            // The application must send the current idle rate to the host
            // by using the USB_DEVICE_ControlSend() function.

           USB_DEVICE_ControlSend(usbDeviceHandle, &currentIdleRate, 1);
           break;

        case USB_DEVICE_HID_EVENT_SET_REPORT:

            // In this case, pData should be interpreted as a
            // USB_DEVICE_HID_EVENT_DATA_SET_REPORT type pointer. The
            // application can analyze the request and then obtain the
            // report by using the USB_DEVICE_ControlReceive() function.

            setReportEventData = (USB_DEVICE_HID_EVENT_DATA_SET_REPORT *)pData;
            USB_DEVICE_ControlReceive(deviceHandle, someHIDReport, setReportEventData->reportLength);
            break;

        case USB_DEVICE_HID_EVENT_SET_PROTOCOL:

            // In this case, pData should be interpreted as a
            // USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL type pointer. The application can
            // analyze the data and decide to stall or accept the setting.
            // This shows an example of accepting the protocol setting.

            USB_DEVICE_ControlStatus(deviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_HID_EVENT_SET_IDLE:

            // In this case, pData should be interpreted as a
```

```
            // USB_DEVICE_HID_EVENT_DATA_SET_IDLE type pointer. The
            // application can analyze the data and decide to stall
            // or accept the setting. This shows an example of accepting
            // the protocol setting.

            setIdleEventData = (USB_DEVICE_HID_EVENT_DATA_SET_IDLE *)pData;
            USB_DEVICE_ControlStatus(deviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:

            // In this case, control transfer data was received. The
            // application can inspect that data and then stall the
            // handshake stage of the control transfer or accept it
            // (as shown here).

            USB_DEVICE_ControlStatus(deviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT:

            // This means that control transfer data was sent. The
            // application would typically acknowledge the handshake
            // stage of the control transfer.

            break;

        case USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_ABORTED:

            // This is an indication only event. The application must
            // reset any HID control transfer related tasks when it receives
            // this event.

        break;

        case USB_DEVICE_HID_EVENT_REPORT_RECEIVED:

            // This means a HID report receive request has completed.
            // The pData member should be interpreted as a
            // USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED pointer type.

            break;

        case USB_DEVICE_HID_EVENT_REPORT_SENT:

            // This means a HID report send request has completed.
            // The pData member should be interpreted as a
            // USB_DEVICE_HID_EVENT_DATA_REPORT_SENT pointer type.

            break;
    }

    return(USB_DEVICE_HID_EVENT_RESPONSE_NONE);
}
```

Remarks

Some of the events allow the application to defer responses. This allows the application some time to obtain the response data rather than having to respond to the event immediately. Note that a USB host will typically wait for event response for a finite time duration before timing out and canceling the event and associated transactions. Even when deferring response, the application must respond promptly if such timeouts have to be avoided.

USB_DEVICE_HID_EVENT_DATA_GET_REPORT Structure

USB Device HID Get Report Event Data Type.

File

usb_device_hid.h

C

```
typedef struct {
  uint8_t reportType;
  uint8_t reportID;
  uint16_t reportLength;
} USB_DEVICE_HID_EVENT_DATA_GET_REPORT;
```

Members

Members	Description
uint8_t reportType;	Report type
uint8_t reportID;	Report ID
uint16_t reportLength;	Report Length

Description

USB Device HID Get Report Event Data Type.

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_GET_REPORT event.

Remarks

None.

USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED Structure

USB Device HID Report Received Event Data Type.

File

usb_device_hid.h

C

```
typedef struct {
  USB_DEVICE_HID_TRANSFER_HANDLE handle;
  size_t length;
  USB_DEVICE_HID_RESULT status;
} USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED;
```

Members

Members	Description
USB_DEVICE_HID_TRANSFER_HANDLE handle;	Transfer handle
size_t length;	Report size received
USB_DEVICE_HID_RESULT status;	Completion status of the transfer

Description

USB Device HID Report Received Event Data Type.

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_REPORT_RECEIVED event.

Remarks

None.

USB_DEVICE_HID_EVENT_DATA_REPORT_SENT Structure

USB Device HID Report Sent Event Data Type.

File

usb_device_hid.h

C

```
typedef struct {
  USB_DEVICE_HID_TRANSFER_HANDLE handle;
  size_t length;
  USB_DEVICE_HID_RESULT status;
} USB_DEVICE_HID_EVENT_DATA_REPORT_SENT;
```

Members

Members	Description
USB_DEVICE_HID_TRANSFER_HANDLE handle;	Transfer handle
size_t length;	Report size transmitted
USB_DEVICE_HID_RESULT status;	Completion status of the transfer

Description

USB Device HID Report Sent Event Data Type.

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_REPORT_SENT event.

Remarks

None.

USB_DEVICE_HID_EVENT_DATA_SET_IDLE Structure

USB Device HID Set Idle Event Data Type.

File

usb_device_hid.h

C

```
typedef struct {
  uint8_t duration;
  uint8_t reportID;
} USB_DEVICE_HID_EVENT_DATA_SET_IDLE;
```

Members

Members	Description
uint8_t duration;	Idle duration
uint8_t reportID;	Report ID

Description

USB Device HID Set Idle Event Data Type.

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_SET_IDLE event.

Remarks

None.

USB_DEVICE_HID_EVENT_DATA_SET_REPORT Structure

USB Device HID Set Report Event Data Type.

File

usb_device_hid.h

C

```
typedef struct {
  uint8_t reportType;
  uint8_t reportID;
  uint16_t reportLength;
} USB_DEVICE_HID_EVENT_DATA_SET_REPORT;
```

Members

Members	Description
uint8_t reportType;	Report type
uint8_t reportID;	Report ID
uint16_t reportLength;	Report Length

Description

USB Device HID Set Report Event Data Type.

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_SET_REPORT event.

Remarks

None.

USB_DEVICE_HID_INDEX Type

USB device HID Function Driver Index.

File

usb_device_hid.h

C

```
typedef uintptr_t USB_DEVICE_HID_INDEX;
```

Description

USB Device HID Driver Index Numbers

This uniquely identifies a HID Function Driver instance.

Remarks

None.

USB_DEVICE_HID_EVENT_DATA_GET_IDLE Structure

USB Device HID Get Idle Event Data Type.

File

usb_device_hid.h

C

```
typedef struct {
  uint8_t reportID;
} USB_DEVICE_HID_EVENT_DATA_GET_IDLE;
```

Members

Members	Description
uint8_t reportID;	The protocol code

Description

USB Device HID Get Idle Event Data

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_GET_IDLE event.

Remarks

None.

USB_DEVICE_HID_TRANSFER_HANDLE Type

USB Device HID Function Driver Transfer Handle Definition.

File

usb_device_hid.h

C

```
typedef uintptr_t USB_DEVICE_HID_TRANSFER_HANDLE;
```

Description

USB Device HID Function Driver Transfer Handle Definition

This definition defines a USB Device HID Function Driver Transfer Handle. A Transfer Handle is owned by the application but its value is modified by the USB_DEVICE_HID_ReportSend and USB_DEVICE_HID_ReportReceive functions. The transfer handle is valid for the life time of the transfer and expires when the transfer related event has occurred.

Remarks

None.

USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL Structure

USB Device HID Set Protocol Event Data Type.

File

usb_device_hid.h

C

```
typedef struct {
  USB_HID_PROTOCOL_CODE protocolCode;
} USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL;
```

Members

Members	Description
USB_HID_PROTOCOL_CODE protocolCode;	The protocol code

Description

USB Device HID Set Protocol Event Data

This defines the data type of the data generated to the HID event handler on a USB_DEVICE_HID_EVENT_SET_PROTOCOL event.

Remarks

None.

USB_DEVICE_HID_EVENT_HANDLER Type

USB Device HID Event Handler Function Pointer Type.

File

usb_device_hid.h

C

```
typedef USB_DEVICE_HID_EVENT_RESPONSE (* USB_DEVICE_HID_EVENT_HANDLER)(USB_DEVICE_HID_INDEX instanceIndex,
USB_DEVICE_HID_EVENT event, void * pData, uintptr_t context);
```

Description

USB Device HID Event Handler Function Pointer Type.

This data type defines the required function signature of the USB Device HID Function Driver event handling callback function. The application must register a pointer to a HID Function Driver events handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event call backs from the HID Function Driver. The function driver will invoke this function with event relevant parameters. The description of the event handler function parameters is given here.

instanceIndex - Instance index of the HID Function Driver that generated the event.

event - Type of event generated.

pData - This parameter should be type casted to a event specific pointer type based on the event that has occurred. Refer to the USB_DEVICE_HID_EVENT enumeration description for more details.

context - Value identifying the context of the application that registered the event handling function.

Remarks

None.

USB_DEVICE_HID_EVENT_RESPONSE Type

USB Device HID Function Driver Event Callback Response Type

File

usb_device_hid.h

C

```
typedef void USB_DEVICE_HID_EVENT_RESPONSE;
```

Description

USB Device HID Function Driver Event Handler Response Type

This is the return type of the HID Function Driver event handler.

Remarks

None.

USB_DEVICE_HID_INIT Structure

USB Device HID Function Driver Initialization Data Structure

File

usb_device_hid.h

C

```
typedef struct {
  size_t hidReportDescriptorSize;
  void * hidReportDescriptor;
  size_t queueSizeReportSend;
  size_t queueSizeReportReceive;
} USB_DEVICE_HID_INIT;
```

Members

Members	Description
size_t hidReportDescriptorSize;	Size of the HID report descriptor
void * hidReportDescriptor;	Pointer to HID report descriptor
size_t queueSizeReportSend;	Report send queue size
size_t queueSizeReportReceive;	Report receive queue size

Description

USB Device HID Function Driver Initialization Data Structure

This data structure must be defined for every instance of the HID function driver. It is passed to the HID function driver, by the Device Layer, at the time of initialization. The funcDriverInit member of the Device Layer Function Driver registration table entry must point to this data structure for an instance of the HID function driver.

Remarks

None.

USB_DEVICE_HID_RESULT Enumeration

USB Device HID Function Driver USB Device HID Result enumeration.

File

usb_device_hid.h

C

```
typedef enum {
  USB_DEVICE_HID_RESULT_OK,
  USB_DEVICE_HID_RESULT_ERROR_TRANSFER_QUEUE_FULL,
  USB_DEVICE_HID_RESULT_ERROR_INSTANCE_NOT_CONFIGURED,
  USB_DEVICE_HID_RESULT_ERROR_INSTANCE_INVALID,
  USB_DEVICE_HID_RESULT_ERROR_TERMINATED_BY_HOST,
  USB_DEVICE_HID_RESULT_ERROR
} USB_DEVICE_HID_RESULT;
```

Members

Members	Description
USB_DEVICE_HID_RESULT_OK	The operation was successful
USB_DEVICE_HID_RESULT_ERROR_TRANSFER_QUEUE_FULL	The transfer queue is full. No new transfers can be • scheduled
USB_DEVICE_HID_RESULT_ERROR_INSTANCE_NOT_CONFIGURED	The specified instance is not configured yet

USB_DEVICE_HID_RESULT_ERROR_INSTANCE_INVALID	The specified instance is not provisioned in the system
USB_DEVICE_HID_RESULT_ERROR_TERMINATED_BY_HOST	Transfer terminated by host because of a stall clear
USB_DEVICE_HID_RESULT_ERROR	General Error

Description

USB Device HID Function Driver USB Device HID Result enumeration.

This enumeration lists the possible USB Device HID Function Driver operation results. These values USB Device HID Library functions.

Remarks

None.

USB_DEVICE_HID_EVENT_RESPONSE_NONE Macro

USB Device HID Function Driver Event Handler Response Type None.

File

usb_device_hid.h

C

```
#define USB_DEVICE_HID_EVENT_RESPONSE_NONE
```

Description

USB Device HID Function Driver Event Handler Response None

This is the definition of the HID Function Driver Event Handler Response Type none.

Remarks

Intentionally defined to be empty.

USB_DEVICE_HID_TRANSFER_HANDLE_INVALID Macro

USB Device HID Function Driver Invalid Transfer Handle Definition.

File

usb_device_hid.h

C

```
#define USB_DEVICE_HID_TRANSFER_HANDLE_INVALID
```

Description

USB Device HID Function Driver Invalid Transfer Handle Definition

This definition defines a USB Device HID Function Driver Invalid Transfer Handle. A Invalid Transfer Handle is returned by the USB_DEVICE_HID_ReportReceive and USB_DEVICE_HID_ReportSend functions when the request was not successful.

Remarks

None.

USB_DEVICE_HID_FUNCTION_DRIVER Macro

This is a pointer to a group of HID Function Driver callback function pointers.

File

usb_device_hid.h

C

```
#define USB_DEVICE_HID_FUNCTION_DRIVER
```

Description

USB Device HID Function Driver Device Layer callback function pointer group

This is a pointer to a group of HID Function Driver callback function pointers. The application must use this pointer while registering an instance of the HID function driver with the Device Layer via the function driver registration table i.e. the driver member of the function driver registration object in the device layer function driver registration table should be set to this value.

Remarks

None.

USB_DEVICE_HID_INDEX_0 Macro

USB Device HID Function Driver Index Constants

File

usb_device_hid.h

C

```
#define USB_DEVICE_HID_INDEX_0 0
```

Description

USB Device HID Function Driver Index Constants

This constants can be used by the application to specify HID function driver instance indexes.

Remarks

None.

USB_DEVICE_HID_INDEX_1 Macro

File

usb_device_hid.h

C

```
#define USB_DEVICE_HID_INDEX_1 1
```

Description

This is macro USB_DEVICE_HID_INDEX_1.

USB_DEVICE_HID_INDEX_2 Macro

File

usb_device_hid.h

C

```
#define USB_DEVICE_HID_INDEX_2 2
```

Description

This is macro USB_DEVICE_HID_INDEX_2.

USB_DEVICE_HID_INDEX_3 Macro

File

usb_device_hid.h

C

```
#define USB_DEVICE_HID_INDEX_3 3
```

Description

This is macro USB_DEVICE_HID_INDEX_3.

USB_DEVICE_HID_INDEX_4 Macro

File

usb_device_hid.h

C

```
#define USB_DEVICE_HID_INDEX_4 4
```

Description

This is macro USB_DEVICE_HID_INDEX_4.

USB_DEVICE_HID_INDEX_5 Macro

File

usb_device_hid.h

C

```
#define USB_DEVICE_HID_INDEX_5 5
```

Description

This is macro USB_DEVICE_HID_INDEX_5.

USB_DEVICE_HID_INDEX_6 Macro

File

usb_device_hid.h

C

```
#define USB_DEVICE_HID_INDEX_6 6
```

Description

This is macro USB_DEVICE_HID_INDEX_6.

USB_DEVICE_HID_INDEX_7 Macro

File

usb_device_hid.h

C

```
#define USB_DEVICE_HID_INDEX_7 7
```

Description

This is macro USB_DEVICE_HID_INDEX_7.

Files

Files

Name	Description
usb_device_hid.h	USB HID Function Driver
usb_device_hid_config_template.h	USB device HID class configuration definitions template,

Description

This section lists the source and header files used by the library.

usb_device_hid.h

USB HID Function Driver

Enumerations

	Name	Description
	USB_DEVICE_HID_EVENT	USB Device HID Function Driver Events
	USB_DEVICE_HID_RESULT	USB Device HID Function Driver USB Device HID Result enumeration.

Functions

	Name	Description
	USB_DEVICE_HID_EventHandlerSet	This function registers a event handler for the specified HID function driver instance.
	USB_DEVICE_HID_ReportReceive	This function submits the buffer to HID function driver library to receive a report from host to device.
	USB_DEVICE_HID_ReportSend	This function submits the buffer to HID function driver library to send a report from device to host.
	USB_DEVICE_HID_TransferCancel	This function cancels a scheduled HID Device data transfer.

Macros

	Name	Description
	USB_DEVICE_HID_EVENT_RESPONSE_NONE	USB Device HID Function Driver Event Handler Response Type None.
	USB_DEVICE_HID_FUNCTION_DRIVER	This is a pointer to a group of HID Function Driver callback function pointers.
	USB_DEVICE_HID_INDEX_0	USB Device HID Function Driver Index Constants
	USB_DEVICE_HID_INDEX_1	This is macro USB_DEVICE_HID_INDEX_1.
	USB_DEVICE_HID_INDEX_2	This is macro USB_DEVICE_HID_INDEX_2.
	USB_DEVICE_HID_INDEX_3	This is macro USB_DEVICE_HID_INDEX_3.
	USB_DEVICE_HID_INDEX_4	This is macro USB_DEVICE_HID_INDEX_4.
	USB_DEVICE_HID_INDEX_5	This is macro USB_DEVICE_HID_INDEX_5.
	USB_DEVICE_HID_INDEX_6	This is macro USB_DEVICE_HID_INDEX_6.
	USB_DEVICE_HID_INDEX_7	This is macro USB_DEVICE_HID_INDEX_7.
	USB_DEVICE_HID_TRANSFER_HANDLE_INVALID	USB Device HID Function Driver Invalid Transfer Handle Definition.

Structures

	Name	Description
	USB_DEVICE_HID_EVENT_DATA_GET_IDLE	USB Device HID Get Idle Event Data Type.
	USB_DEVICE_HID_EVENT_DATA_GET_REPORT	USB Device HID Get Report Event Data Type.
	USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED	USB Device HID Report Received Event Data Type.
	USB_DEVICE_HID_EVENT_DATA_REPORT_SENT	USB Device HID Report Sent Event Data Type.
	USB_DEVICE_HID_EVENT_DATA_SET_IDLE	USB Device HID Set Idle Event Data Type.
	USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL	USB Device HID Set Protocol Event Data Type.
	USB_DEVICE_HID_EVENT_DATA_SET_REPORT	USB Device HID Set Report Event Data Type.
	USB_DEVICE_HID_INIT	USB Device HID Function Driver Initialization Data Structure

Types

	Name	Description
	USB_DEVICE_HID_EVENT_HANDLER	USB Device HID Event Handler Function Pointer Type.
	USB_DEVICE_HID_EVENT_RESPONSE	USB Device HID Function Driver Event Callback Response Type
	USB_DEVICE_HID_INDEX	USB device HID Function Driver Index.
	USB_DEVICE_HID_TRANSFER_HANDLE	USB Device HID Function Driver Transfer Handle Definition.

Description

USB HID Function Driver

This file contains the API definitions for the USB Device HID Function Driver. The application should include this file if it needs to use the HID Function Driver API.

File Name

usb_hid_function_driver.h

Company

Microchip Technology Inc.

usb_device_hid_config_template.h

USB device HID class configuration definitions template,

Macros

	Name	Description
	USB_DEVICE_HID_INSTANCES_NUMBER	Specifies the number of HID instances.
	USB_DEVICE_HID_QUEUE_DEPTH_COMINED	DOM-IGONORE-BEGIN

Description

USB Device HID Class Configuration Definitions

This file contains configurations macros needed to configure the HID Function Driver. This file is a template file only. It should not be included by the application. The configuration macros defined in the file should be defined in the configuration specific system_config.h.

File Name

usb_device_hid_config_template.h

Company

Microchip Technology Inc.

USB MSD Device Library

This section describes the USB MSD Device Library.

Introduction

Introduces the MPLAB Harmony USB Mass Storage Device (MSD) Library.

Description

The USB Mass Storage Device Library (also referred to as the MSD Function Driver) allows applications to create USB Mass Storage device such as USB Pen Drives or USB-based SD Card readers. Applications can also leverage the ready support for Mass Storage Devices by popular Host personal computer operating systems by using the MSD Function Driver interfaces as a means to access the device functionality. The MSD Function Driver also features the following:

- Supports Bulk Only Transport (BOT) protocol
- Allows implementation of multiple Logical Unit Number (LUN) storage devices
- Uses the MPLAB Harmony Block Driver interface to connect to storage media drivers

Using the Library

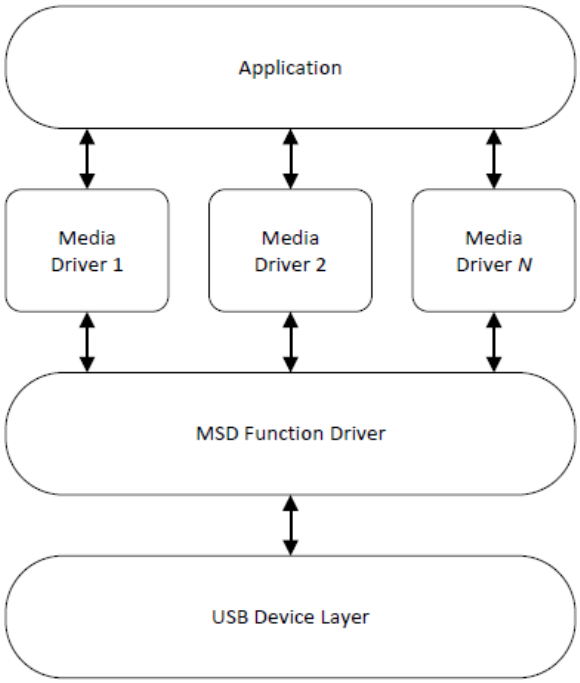
This topic describes the basic architecture of the USB MSD Device Library and provides information and examples on its use.

Abstraction Model

Provides an architectural overview of the USB MSD Device Library driver.

Description

The following diagram illustrates the functional interaction between the application, the MSD Function Driver, the media drivers, and the USB Device Layer.



As seen in the previous figure, the application does not have to interact with MSD function driver. Also, the MSD Function Driver does not have application functions that can be called. The media drivers control the storage media. The application interacts with the media drivers to update or access the information on the storage media. The MSD Function Driver interacts with the media drivers to process data read and write requests that it receives from the Host. This data is always accessed in blocks.

The MPLAB Harmony System module initializes the Device Layer and media drivers. A media driver is plugged into the MSD Function Driver by providing a media driver entry point in the MSD Function Driver initialization data structure. In the case of a multi-LUN storage, multiple media drivers can be plugged into the MSD Function Driver, with each one being capable of accessing different storage media types. The Device Layer initializes the MSD Function Driver when the Host sets the configuration that contains the Mass Storage interfaces. The MSD Function Driver Tasks routine is invoked in the context of the Device Layer Tasks routine. The MSD Function Driver interfaces should be registered in the USB Device Layer Function Driver Registration Table.

Library Overview

The USB MSD Device Library mainly interacts with the system, its clients and function drivers, as shown in the Abstraction Model.

The library interface routines are divided into sub-sections, which address one of the blocks or the overall operation of the USB MSD Device Library.

Library Interface Section	Description
System Configuration Functions	Provides event handler, report send/receive, and transfer cancellation functions.

How the Library Works

This section explains how the MSD Function Driver should be added to the USB Device application and how a media driver should be plugged into it. Considerations while creating new media drivers to operate with the MSD function driver are also discussed.

Library Initialization

Describes how to initialize the MSD Function Driver.

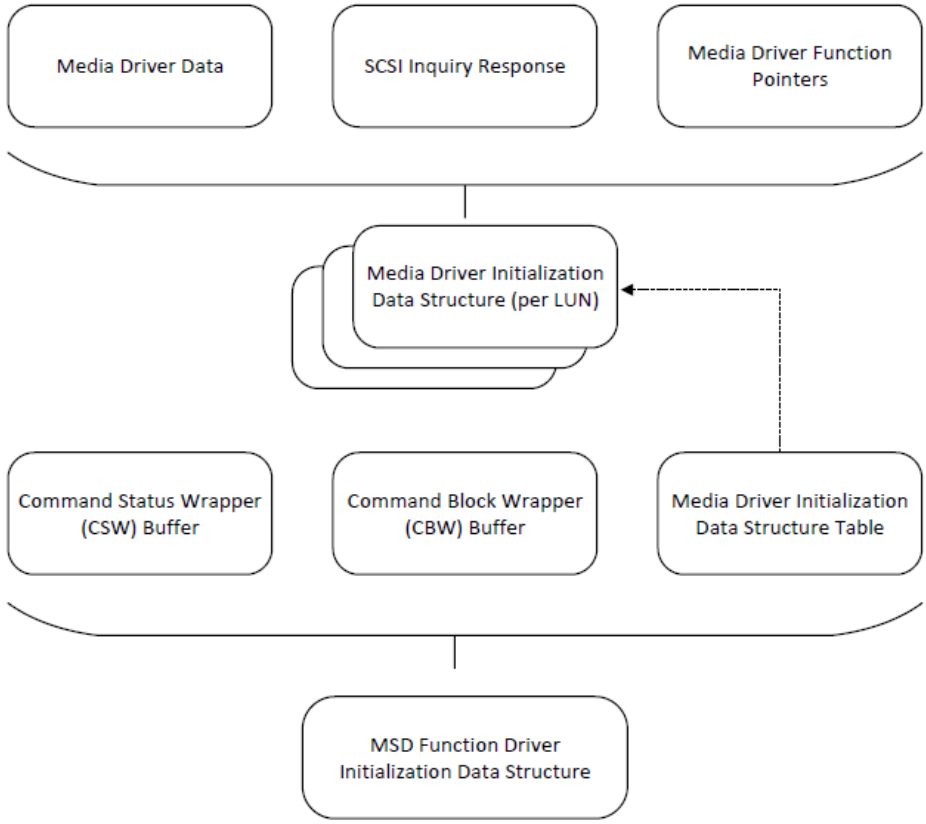
Description

The MSD Function Driver instance for a USB Device configuration is initialized by the USB Device Layer when the Host sets that configuration. This process does not require application intervention. Each instance of the MSD Function Driver should be registered with the USB Device Layer through the Device Layer Function Driver Registration Table. While registering the MSD Function Driver, the driver member of the Function Driver Registration Table entry should be set to USB_DEVICE_MSD_FUNCTION_DRIVER. This is an opaque function driver entry point provided by the MSD Function Driver for the Device Layer to use.

The MSD Function Driver requires an initialization data structure to be defined for each instance of the function driver. This initialization data structure should be of the type USB_DEVICE_MSD_INIT. This initialization data structure contains the following:

- The number of Logical Unit Numbers (LUNs) in this MSD Function Driver instance
- A pointer to the USB_MSD_CBW type data structure. This pointer is used by the MSD Function Driver to receive the Command Block Wrapper (CBW) from the Host. For a PIC32MZ device, this array should be placed in coherent memory and should be aligned on a 4-byte boundary.
- A pointer to the USB_MSD_CSW type data structure. This pointer is used by the MSD Function Driver to send the Command Status Wrapper (CSW) to the Host. For a PIC32MZ device, this array should be placed in coherent memory and should be aligned on a 4-byte boundary.
- A pointer to the array of media driver initialization data structure. There should be one structure for every LUN. This is a USB_DEVICE_MSD_MEDIA_INIT_DATA type of data structure. There exists a one-to-one mapping between the LUN and the media driver initialization data structure.

The following figure shows a pictorial representation of the MSD Function Driver initialization data structure.



The USB_DEVICE_MSD_MEDIA_INIT_DATA data structure allows a media driver to be plugged into the MSD Function Driver. Any media driver that needs to be plugged into the MSD Function Driver needs to implement the interface (function pointer signatures) specified by the USB_DEVICE_MSD_MEDIA_FUNCTIONS type. For every LUN, a SCSI Inquiry Response data structure needs to be made available.

Use the following guidelines while implementing the media driver:

- Read functions should be non-blocking
- Write functions should be non-blocking
- The media driver should provide an event to indicate when a block transfer has complete. It should allow the event handler to be registered.
- Where required, the write function should erase and write to the storage area in one operation. The MSD Function Driver does not explicitly call the erase operation.
- The media driver should provide a media geometry object when required. This media geometry object allows the MSD Function Driver to understand the media characteristics. This object is of the type, SYS_FS_MEDIA_GEOMETRY.

The following code shows an example of plugging the MPLAB Harmony NVM Driver into the MSD Function Driver. The coherency and alignment attributes that are applied to the sectorBuffer, msdCBW, and msdCBW data objects is needed for operation on PIC32MZ devices.

```
/*********************************************
 * Sector buffer needed by for the MSD LUN.
 *********************************************/
uint8_t sectorBuffer[512] __attribute__((coherent)) __attribute__((aligned(4)));

/*********************************************
 * CBW and CSW structure needed by the MSD
 * function driver instance.
 *********************************************/
USB_MSD_CBW msdCBW __attribute__((coherent)) __attribute__((aligned(4)));
USB_MSD_CSW msdCSW __attribute__((coherent)) __attribute__((aligned(4)));

/*********************************************
```

```
 * Because the PIC32MZ flash row size if 2048
 * and the media sector size if 512 bytes, we
 * have to allocate a buffer of size 2048
 * to backup the row. A pointer to this row
 * is passed in the media initialization data
 * structure.
 ********************************************/
uint8_t flashRowBackupBuffer [DRV_NVM_ROW_SIZE];
/*****************************************
 * MSD Function Driver initialization
 *****************************************/

USB_DEVICE_MSD_MEDIA_INIT_DATA msdMediaInit[1] =
{
    {
        DRV_NVM_INDEX_0,
        512,
        sectorBuffer,
        flashRowBackupBuffer,
        (void *)diskImage,
        {
            0x00,    // peripheral device is connected, direct access block device
            0x80,       // removable
            0x04,    // version = 00=> does not conform to any standard, 4=> SPC-2
            0x02,    // response is in format specified by SPC-2
            0x20,    // n-4 = 36-4=32= 0x20
            0x00,    // sccs etc.
            0x00,    // bque=1 and cmdque=0,indicates simple queuing 00 is obsolete,
                        // but as in case of other device, we are just using 00
            0x00,    // 00 obsolete, 0x80 for basic task queuing
            {
                'M','i','c','r','o','c','h','p'
            },
            {
                'M','a','s','s',' ','S','t','o','r','a','g','e',' ',' ',' ',' '
            },
            {
                '0','0','0','1'
            }
        },
        {
            DRV_NVM_IsAttached,
            DRV_NVM_BLOCK_Open,
            DRV_NVM_BLOCK_Close,
            DRV_NVM_GeometryGet,
            DRV_NVM_BlockRead,
            DRV_NVM_BlockEraseWrite,
            DRV_NVM_IsWriteProtected,
            DRV_NVM_BLOCK_EventHandlerSet,
            DRV_NVM_BlockStartAddressSet
        }
    }
};
/*****************************************
 * MSD Function Driver initialization
 *****************************************/
USB_DEVICE_MSD_INIT msdInit =
{
    /* Number of LUNS */
    1,
    /* Pointer to a CBW structure */
    &msdCBW,
    /* Pointer to a CSW structure */
    &msdCSW,
    /* Pointer to a table of Media Initialization data structures */
    &msdMediaInit[0]
};
/*************************************************
 * USB Device Function Registration Table
```

```
 *************************************************/
const USB_DEVICE_FUNCTION_REGISTRATION_TABLE funcRegistrationTable[1] =
{
    {
        .speed = USB_SPEED_FULL | USB_SPEED_HIGH,   // Device Speed
        .configurationValue =  1,                   // Configuration value
        .interfaceNumber = 0,                       // Start interface number
        .numberOfInterfaces = 1,                    // Number of interfaces owned
        .funcDriverIndex = 0,                       // Function driver index
        .funcDriverInit = (void*)&msdInit,          // Pointer to initialization data structure
        .driver = USB_DEVICE_MSD_FUNCTION_DRIVER    // Pointer to function driver
    }
};
```

Data Transfer

Describes how the MSD Function Driver accesses the media.

Description

The MSD Function Driver opens the media drivers for read/write operations when the function driver is initialized by the Device Layer. This happens when the Host sets a configuration containing MSD interfaces. The Open operation is complete in the MSD Function Driver Tasks routines (called by the Device Layer).

The MSD Function Driver registers its own block operation event handler with the media drivers. Media Read and Write functions are called when the function driver receives a Sector Read or Sector Write request from the Host. The request will be tracked in the function driver Task routine. While the function driver waits for the media to complete the block operation, the function driver will NAK the data stage of the MSD data transfer request.

The MSD Function Driver does not provide any events to the application. It is possible that the application may also open the media driver while they are already opened by the MSD Function Driver. If the application and the MSD Function Driver try to write to the same media driver, the result could be unpredictable. It is recommended that the application restrict write access to the media driver while the USB device is plugged into the Host.

The application does not have to intervene in the functioning of the MSD Function Driver. Basically, the MSD Function Driver does provide any application callable functions.

Configuring the Library

Describes how to configure the MSD Function Driver.

Macros

	Name	Description
	USB_DEVICE_MSD_INSTANCES_NUMBER	Number of MSD function Driver instances required in the USB Device.
	USB_DEVICE_MSD_LUNS_NUMBER	Defines the number of LUNs per MSD function driver instance.

Description

The following configuration parameters must be defined while using the MSD Function Driver. The configuration macros that implement these parameters must be located in the `system_config.h` file in the application project and a compiler include path (to point to the folder that contains this file) should be specified.

USB_DEVICE_MSD_INSTANCES_NUMBER Macro

Number of MSD function Driver instances required in the USB Device.

File

usb_device_msd_config_template.h

C

```
#define USB_DEVICE_MSD_INSTANCES_NUMBER 1
```

Description

MSD Function Driver Instances Number

This configuration constant defines the number of MSD Function Driver instances in the USB Device. This value should be atleast 1 if the MSD function is required. In case where multiple MSD function drivers are required, it should be noted the MSD function driver supports multiple LUNs. This allows one MSD function driver instance to manage multiple media. Using multiple LUNs can be considered as an alternative to using multiple MSD function driver instances.

Remarks

None.

USB_DEVICE_MSD_LUNS_NUMBER Macro

Defines the number of LUNs per MSD function driver instance.

File

usb_device_msd_config_template.h

C

```
#define USB_DEVICE_MSD_LUNS_NUMBER 1
```

Description

Number of LUNs

This constant sets maximum possible number of Logical Unit (LUN) an instance of MSD can support. This value should be atleast 1. In cases where multiple MSD Function Driver instances are required, this constant should be set to the maximum number of LUNs required by any MSD Function Driver instanceThe following figure shows a pictorial representation of the MSD function driver initialization data structure.

Remarks

None.

Building the Library

Describes the files to be included in the project while using the MSD Function Driver.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/usb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`usb_device_msd.h`	This header file should be included in any `.c` file that accesses the USB Device MSD Function Driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/usb_device_msd.c`	This file implements the MSD Function driver interface and should be included in the project if the MSD Device function is desired.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The USB CDC Device Library depends on the following modules:

- USB Device Layer Library

Based on application needs, the library may depend on the related storage media libraries, such as:

- Secure Digital (SD) Card Driver Library
- NVM Driver Library

Library Interface

Data Types and Constants

	Name	Description
	USB_DEVICE_MSD_MEDIA_FUNCTIONS	Pointer to the media driver functions for media instances to used with the the MSD function driver.
	USB_DEVICE_MSD_INIT	This structure contains required parameters for MSD function driver initialization.
	USB_DEVICE_MSD_MEDIA_INIT_DATA	This structure holds media related data of a particular logical unit.
	USB_DEVICE_MSD_FUNCTION_DRIVER	USB Device MSD Function Driver Function pointer

Description

This section describes the Application Programming Interface (API) functions of the USB MSD Device Library.

Refer to each section for a detailed description.

a) System Configuration Functions

Data Types and Constants

USB_DEVICE_MSD_MEDIA_FUNCTIONS Structure

Pointer to the media driver functions for media instances to used with the the MSD function driver.

File

usb_device_msd.h

C

```
struct USB_DEVICE_MSD_MEDIA_FUNCTIONS {
  bool (* isAttached)(const DRV_HANDLE handle);
  DRV_HANDLE (* open)(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);
  void (* close)(DRV_HANDLE hClient);
  SYS_FS_MEDIA_GEOMETRY * (* geometryGet)(DRV_HANDLE hClient);
  void (* blockRead)(DRV_HANDLE handle, uintptr_t * blockOperationHandle, void * data, uint32_t blockStart,
uint32_t nBlocks);
  void (* blockWrite)(DRV_HANDLE handle, uintptr_t * blockOperationHandle, void * data, uint32_t
blockStart, uint32_t nBlocks);
  bool (* isWriteProtected)(DRV_HANDLE drvHandle);
  void (* blockEventHandlerSet)(const DRV_HANDLE drvHandle, const void * eventHandler, const uintptr_t
context);
  void (* blockStartAddressSet)(const DRV_HANDLE drvHandle, const void * addressOfStartBlock);
};
```

Members

Members	Description
bool (* isAttached)(const DRV_HANDLE handle);	In case of pluggable media, such as SD Card, this function returns true when the media is inserted, initialized and ready to be used. In case of non-pluggable media, such as Internal Flash memory, this function can return true when the media is ready to be used. The MSD host may not detect the media until this function returns true. This function pointer cannot be NULL
DRV_HANDLE (* open)(const SYS_MODULE_INDEX index, const DRV_IO_INTENT intent);	The MSD Function Driver calls this function to obtain a handle and gain access to functionality of the specified instance of the media driver. The MSD function driver will attempt to open the driver with DRV_IO_INTENT_READWRITE and DRV_IO_INTENT_NONBLOCKING. The MSD host may not detect the media until the MSD function driver obtains a valid driver handle. The function driver will use this handle in all other functions to communicate with the media driver. This function pointer cannot be NULL
void (* close)(DRV_HANDLE hClient);	The MSD function driver calls this function when the function driver gets deinitialized as result of device detach or a change in configuration. The MSD function driver will open the media driver again to obtain a fresh driver handle when it gets initialized again. This function pointer cannot be NULL.

SYS_FS_MEDIA_GEOMETRY * (* geometryGet)(DRV_HANDLE hClient);	The MSD function driver calls this function when the function driver needs to know the storage capacity of the media. The MSD function driver uses the size of read region and number of read blocks to report the media capacity to the MSD host. This function pointer cannot be NULL.
void (* blockRead)(DRV_HANDLE handle, uintptr_t * blockOperationHandle, void * data, uint32_t blockStart, uint32_t nBlocks);	The MSD function driver calls this function when it needs to read a block of data. This function pointer cannot be NULL.
void (* blockWrite)(DRV_HANDLE handle, uintptr_t * blockOperationHandle, void * data, uint32_t blockStart, uint32_t nBlocks);	The MSD function driver calls this function when it needs to write a block of data. This function pointer can be NULL if the media is write protected.
bool (* isWriteProtected)(DRV_HANDLE drvHandle);	The MSD function driver calls this function to find out if the media is write-protected. This function pointer cannot be NULL.
void (* blockEventHandlerSet)(const DRV_HANDLE drvHandle, const void * eventHandler, const uintptr_t context);	The MSD function driver calls this function to register an block event call back function with the media driver. This event call back will be called when a block related operation has completed. This function pointer should not be NULL.
void (* blockStartAddressSet)(const DRV_HANDLE drvHandle, const void * addressOfStartBlock);	If not NULL and if the blockStartAddress parameter in the USB_DEVICE_MSD_MEDIA_INIT_DATA data structure for this media is not 0, then the MSD function driver calls this function immediately after opening the media driver. For media such a NVM, where the storage media is a part of the program memory flash, this function sets the start of the storage area on the media. This function is not required for media such as SD Card.

Description

Media Driver Function Pointer Data Structure

This structure contains function pointers, pointing to the media driver functions. The MSD function driver calls these functions at run time to access the media. This data structure should be specified during compilation and is a part of the MSD function driver initialization data structure. It is processed by the function driver when the function driver is initialized by the Device Layer.

Remarks

None.

USB_DEVICE_MSD_INIT Structure

This structure contains required parameters for MSD function driver initialization.

File

usb_device_msd.h

C

```
typedef struct {
  uint8_t numberOfLogicalUnits;
  USB_MSD_CBW * msdCBW;
  USB_MSD_CSW * msdCSW;
  USB_DEVICE_MSD_MEDIA_INIT_DATA * mediaInit;
} USB_DEVICE_MSD_INIT;
```

Members

Members	Description
uint8_t numberOfLogicalUnits;	Number of logical units supported.
USB_MSD_CBW * msdCBW;	Pointer to a Command Block Wrapper structure allocated to this instance • of the MSD function driver. In case of PIC32MZ device, this should be • placed in non cacheable section of RAM and should be aligned at a 4 byte • boundary.
USB_MSD_CSW * msdCSW;	Pointer to a Command Status Wrapper structure allocated to this instance • of the MSD function driver. In case of PIC32MZ device, this should be • placed in non cacheable section of RAM and should be aligned at a 4 byte • boundary.
USB_DEVICE_MSD_MEDIA_INIT_DATA * mediaInit;	Pointer to a table of media initialization data. This should contain an entry for every logical unit.

Description

USB MSD init structure.

This structure contains interface number, bulk-IN and bulk-OUT endpoint addresses, endpointSize, number of logical units supported and pointer

to array of structure that contains media initialization.

Remarks

This structure must be configured by the user at compile time.

USB_DEVICE_MSD_MEDIA_INIT_DATA Structure

This structure holds media related data of a particular logical unit.

File

usb_device_msd.h

C

```
typedef struct {
  SYS_MODULE_INDEX instanceIndex;
  uint32_t sectorSize;
  uint8_t * sectorBuffer;
  uint8_t * blockBuffer;
  void * block0StartAddress;
  SCSI_INQUIRY_RESPONSE inquiryResponse;
  USB_DEVICE_MSD_MEDIA_FUNCTIONS mediaFunctions;
} USB_DEVICE_MSD_MEDIA_INIT_DATA;
```

Members

Members	Description
SYS_MODULE_INDEX instanceIndex;	Instance index of the media driver to opened for this LUN
uint32_t sectorSize;	Sector size for this LUN. If 0, means that sector size will be available from media geometry.
uint8_t * sectorBuffer;	Pointer to a bye buffer whose size if the size of the sector on this • media. In case of a PIC32MZ device, this buffer should be coherent and • should be aligned on a 16 byte boundary
uint8_t * blockBuffer;	In a case where the sector size of this media is less than the size of • the write block, a byte buffer of write block size should be provided to • the function driver. For example, the PIC32MZ NVM flash driver has a • flash program memory row size of 4096 bytes which is more than the • standard 512 byte sector. In such a case the application should set this • pointer to 4096 byte buffer
void * block0StartAddress;	Block 0 Start Address on this media. If non zero, then this address will be passed to blockStartAddressSet function. This should be set to start of the storage address on the media.
SCSI_INQUIRY_RESPONSE inquiryResponse;	Pointer to SCSI inquiry response for this LUN
USB_DEVICE_MSD_MEDIA_FUNCTIONS mediaFunctions;	Function pointers to the media driver functions

Description

USB Device MSD Media Initialization Data Member

It holds pointer to inquiry response, instance index and pointer to a structure that contains all media callback functions.

Remarks

An object of this structure must be configured by the user at compile time.

USB_DEVICE_MSD_FUNCTION_DRIVER Macro

USB Device MSD Function Driver Function pointer

File

usb_device_msd.h

C

```
#define USB_DEVICE_MSD_FUNCTION_DRIVER
```

Description

USB Device MSD Function Driver Function Pointer

This is the USB Device MSD Function Driver Function pointer. This should registered with the device layer in the function driver registration table.

Remarks

None.

Files

Files

Name	Description
usb_device_msd.h	USB device MSD function driver interface header
usb_device_msd_config_template.h	USB Device MSD configuration template header file

Description

This section lists the source and header files used by the library.

usb_device_msd.h

USB device MSD function driver interface header

Macros

	Name	Description
	USB_DEVICE_MSD_FUNCTION_DRIVER	USB Device MSD Function Driver Function pointer

Structures

	Name	Description
	USB_DEVICE_MSD_MEDIA_FUNCTIONS	Pointer to the media driver functions for media instances to used with the the MSD function driver.
	USB_DEVICE_MSD_INIT	This structure contains required parameters for MSD function driver initialization.
	USB_DEVICE_MSD_MEDIA_INIT_DATA	This structure holds media related data of a particular logical unit.

Description

USB MSD function driver interface header

USB device MSD function driver interface header. This file should be included in the application if USB MSD functionality is required.

File Name

usb_device_msd.h

Company

Microchip Technology Inc.

usb_device_msd_config_template.h

USB Device MSD configuration template header file

Macros

	Name	Description
	USB_DEVICE_MSD_INSTANCES_NUMBER	Number of MSD function Driver instances required in the USB Device.
	USB_DEVICE_MSD_LUNS_NUMBER	Defines the number of LUNs per MSD function driver instance.

Description

USB Device MSD function driver compile time options

This file contains USB device MSD function driver compile time options(macros) that has to be configured by the user. This file is a template file and must be used as an example only. This file must not be directly included in the project.

File Name

usb_device_msd_config_template.h

Company

Microchip Technology Inc.

Generic USB Device Library

This section describes the Generic USB Device Library.

Introduction

Introduces the MPLAB Harmony Generic USB Device Library.

Description

A USB Device that does not follow any of the standard USB device class specifications is referred to as Generic (or a Vendor) USB Device. Such a device may be needed in cases where a standard USB device class does not meet application requirements with respect to transfer type, throughput or available interfaces. Generic USB Devices also typically require custom USB Host drivers.

The MPLAB Harmony USB Device Layer API features Endpoint API and events that facilitate development of a Generic USB Device. These API and events allow the application to do the following:

- Configure, enable, and disable endpoints
- Schedule Bulk, Interrupt and, Isochronous transfers
- Respond to control transfers
- Receive control and other transfer type related events

Using the Library

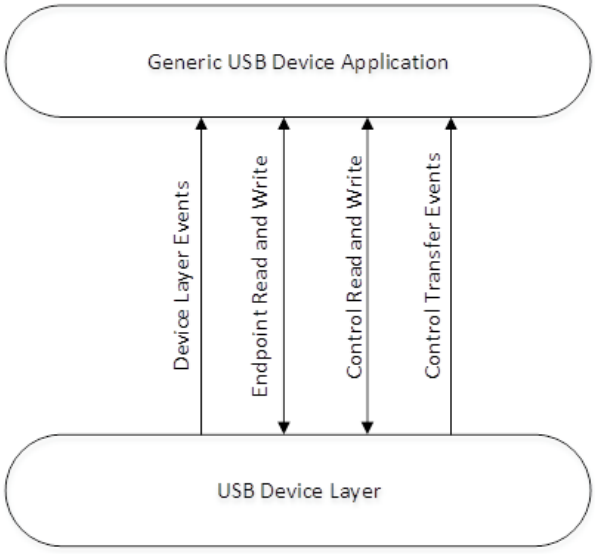
This topic describes the basic architecture of the Generic USB Device Library and provides information and examples on its use.

Abstraction Model

Provides an architectural overview of the Generic USB Device Driver.

Description

The Generic USB Device Library consists of USB Device Layer Endpoint API and events. The API allows the application to configure, enable, and disable endpoints. Endpoints can be configured for bulk, isochronous, and interrupt transfers. The events allow the application to track the completion of transfers and respond to control transfer events. It should be noted that the Generic USB Device Library in the MPLAB Harmony USB Device Stack does not have its own implementation, but rather, uses a subset of the Device Layer API to access the USB, as shown in the following diagram.



As seen in the figure, the application must implement the logic to implement the Generic USB Device behavior. It must respond to interface, class, and other control transfers. It must configure endpoints when the Host sets the configuration. Thus, the application implements the function driver for the Generic USB Function Driver.

The Generic USB Device Endpoint function and events provided by the Device Layer API abstract the details of configuring the USB peripheral. The Device Layer responds to standard USB requests as a part of the device enumeration process. The Device Layer control transfer functions and events allow the application to complete control transfers that are targeted to an endpoint, interface or others. The Device Layer endpoint read and write API provide a USB transaction or transfer level interface. Transactions or transfers can be queued.

Library Overview

Provides an overview of the Generic USB Device Driver.

Description

The Generic Function Driver features API to set application event handlers and transfer data over non-zero endpoints. The function driver is initialized by the Device Layer when a Set Configuration request is received by the device. This process does not require application intervention. As a part of this initialization process, all the endpoints belonging to the Generic Function Driver Interfaces will be enabled and configured. When the application receives the USB_DEVICE_EVENT_CONFIGURED, these endpoints are ready for data transfers.

The application design must ensure that the Generic Function Driver is registered in the Device Layer Function Driver Registration Table.

How the Library Works

This topic describes the basic architecture of the Generic USB Device Library and provides information and examples on its use.

Library Initialization

Describes how the Generic USB Device Library is initialized.

Description

Unlike the standard USB function drivers in the MPLAB Harmony USB Device Stack, in the case of a Generic USB Device, the USB Device Layer does not automatically enable or disable endpoints that belong to the Generic interface. This must be done by the application when the device is configured by the Host.

A USB Device can have multiple Generic interfaces. Each of these interfaces must have corresponding entries in the USB Device Layer function driver registration table. For Generic interfaces, the driver and funcDriverInit member of the function driver registration table entry should be set to NULL. The following code shows an example of how this is done.

```
/* This code shows an example function driver registration table entry
 * for a Generic USB Device Interface. Note that the function driver entry point
 * member is NULL. This instructs the Device Layer to pass all interface related
 * control transfers to the application. */
const USB_DEVICE_FUNCTION_REGISTRATION_TABLE funcRegistrationTable[1] =
{
    {
         .configurationValue = 1 ,      // Configuration descriptor index
         .driver = NULL,                // No APIs exposed to the device layer
         .funcDriverIndex = 0 ,         // Zero Instance index
         .funcDriverInit = NULL,        // No init data
         .interfaceNumber = 0 ,         // Start interface number of this instance
         .numberOfInterfaces = 1 ,      // Total number of interfaces contained in this instance
         .speed = USB_SPEED_FULL|USB_SPEED_HIGH         // USB Speed
    }
};
```

The endpoint read and endpoint write queue sizes are specified by the queueSizeEndpointRead and queueSizeEndpointWrite members of the USB_DEVICE_INIT device layer initialization data structure. These read and write queue sizes define the size of the read and write buffer object pools. Objects from these pools are then queued up at each read and write endpoint, when an endpoint read or write is requested. The total number of buffer objects is specified by the USB_DEVICE_ENDPOINT_QUEUE_DEPTH_COMBINED configuration constant.

Event Handling

This topic explains how the application should handle Generic USB Device events.

Description

The USB Device Layer generates two different types of events for a Generic USB Device.

- Control transfer events
- Endpoint data transfer events

While handing Device Layer events, it is recommended that computationally intensive operations or hardware access should not be performed with in the event handler. Doing so may affect the capability of the Device Stack to respond to changes on the USB and could cause the Device to become non-compliant.

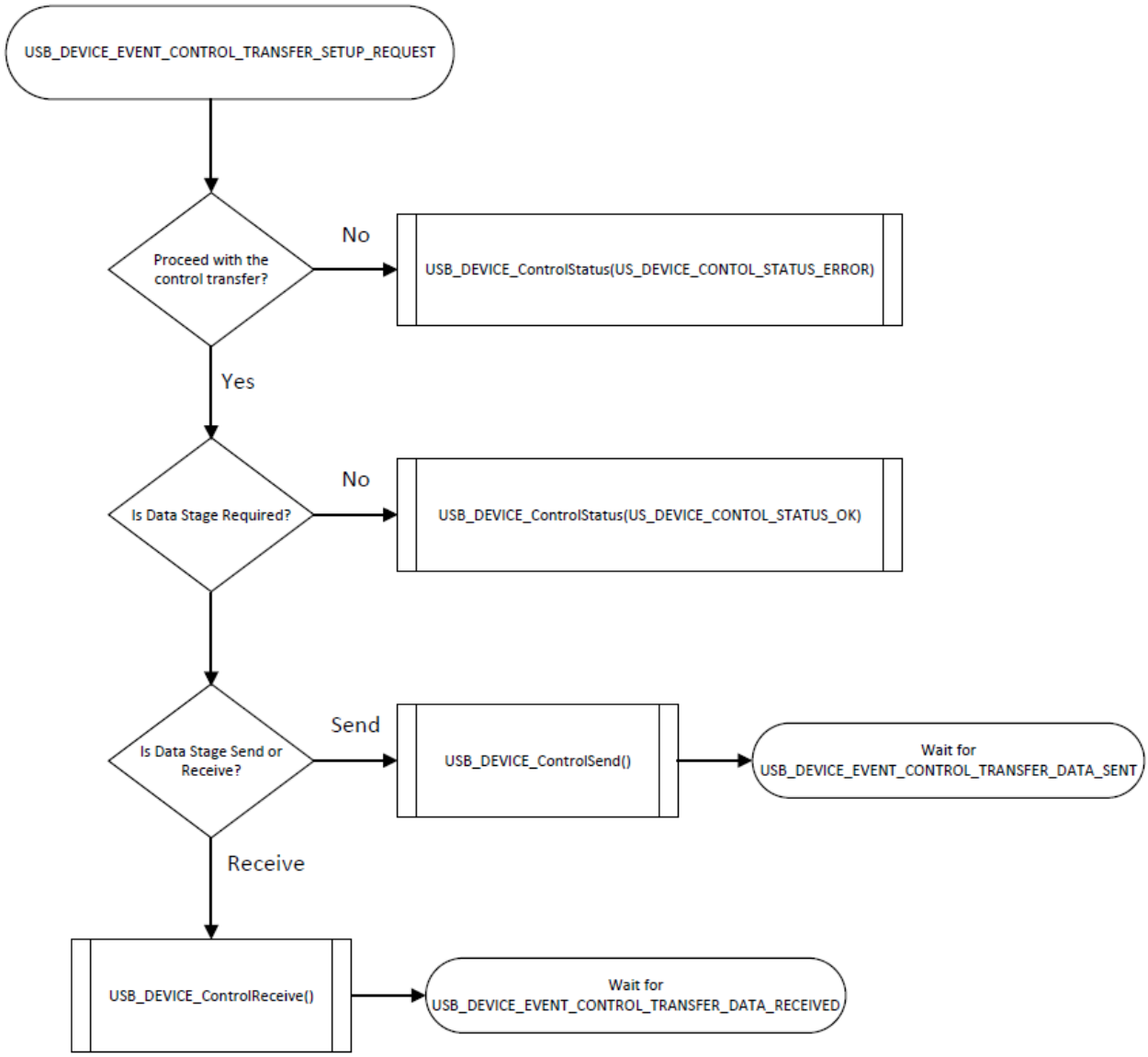
A Generic USB Device application must handle the above events along with the other Device Layer events.

Control Transfer Events

Describes control transfer events and provides a code example.

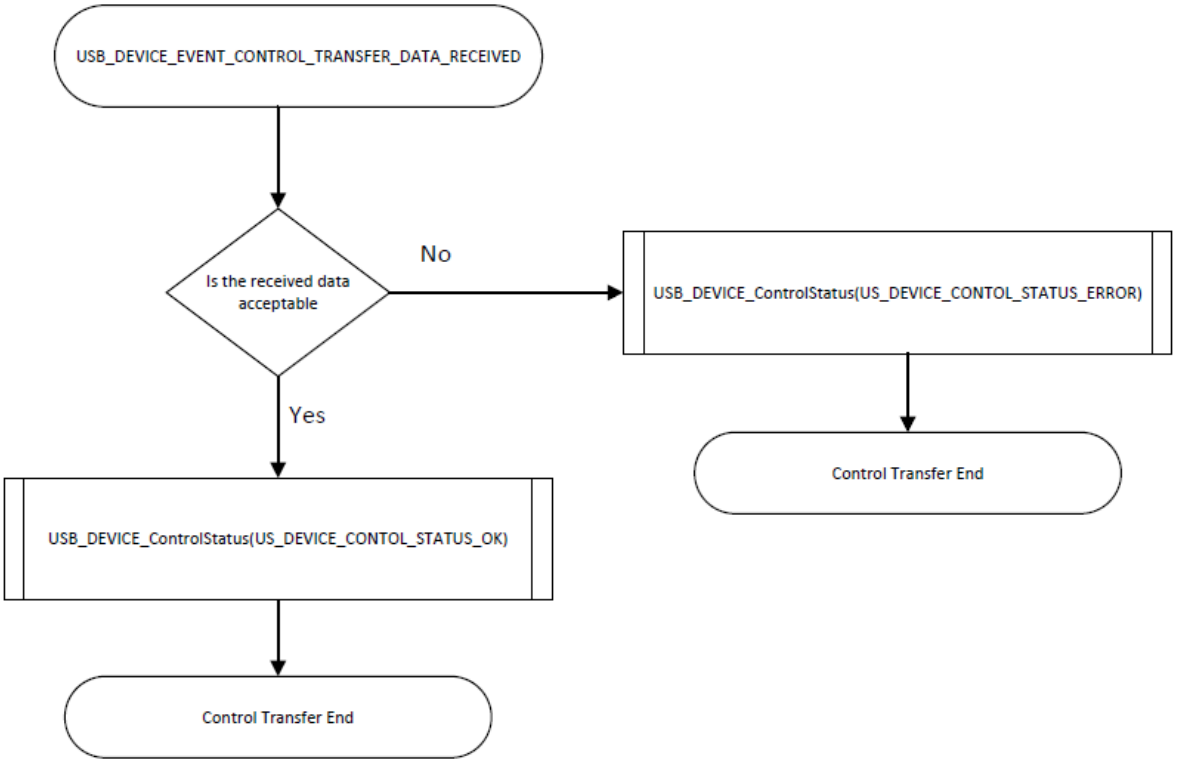
Description

These events occur when the Device Layer has received a control transfer that is targeted to an interface or an endpoint which is managed by the Generic USB Device Application. The USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST event is generated when the Setup stage of the control transfer has been received. The application must investigate the 8-byte setup command that accompanies this event. The following flowchart explains the interaction.



The application can then either choose to continue the control transfer or stall it. The control transfer is stalled by calling the USB_DEVICE_ControlStatus function with the USB_DEVICE_CONTROL_STATUS_ERROR flag. In case of zero data stage control transfers, the application can complete the control transfer by calling the USB_DEVICE_ControlStatus function with the USB_DEVICE_CONTROL_STATUS_OK flag. In case of control transfers that contain a data stage, the application must use the USB_DEVICE_ControlSend or the USB_DEVICE_ControlReceive function to send and receive data from the Host, respectively.

In a case where data is to be received from the host, the device layer generates USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED event when the data stage has completed. The application can analyze the received data and can then either choose to acknowledge or stall the control transfer by the calling the USB_DEVICE_ControlStatus function with the USB_DEVICE_CONTROL_STATUS_ERROR flag. This is shown in the following flow chart.



The following code shows an example of handling control transfer in a Generic USB Device. Note that the control transfer events are generated by the Device Layer.

```
/* This code shows an example of how the control transfer events
 * can be handled in a Generic USB Device. The example device will accepts the
 * Set Interface Control Request and replies to the Get Interface Control Request
 * with the current alternate setting. */
case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST:
    /* This means we have received a setup packet */
       setupPacket = (USB_SETUP_PACKET *)eventData;
    if(setupPacket->bRequest == USB_REQUEST_SET_INTERFACE)
    {
        /* If we have got the SET_INTERFACE request, we just acknowledge
         * for now. In this example, there is one alternate setting which
         * is already active. */
        USB_DEVICE_ControlStatus(appData.usbDevHandle,USB_DEVICE_CONTROL_STATUS_OK);
    }
    else if(setupPacket->bRequest == USB_REQUEST_GET_INTERFACE)
    {
        /* We have only one alternate setting and this setting 0. So
        * we send this information to the host. */
        USB_DEVICE_ControlSend(appData.usbDevHandle, &appData.altSetting, 1);
    }
    else
    {
        /* We have received a request that we cannot handle. Stall it*/
        USB_DEVICE_ControlStatus(appData.usbDevHandle, USB_DEVICE_CONTROL_STATUS_ERROR);
    }
    break;
case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT:
    /* This is a notification event which the application can use to free
     * buffer that was used in a USB_DEVICE_ControlSend() function. */
    break;
case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:
    /* This event means that data has been received in the control transfer
     * and the application must either stall or acknowledge the data stage
     * by calling the USB_DEVICE_ControlStatus() function. Here we simply
     * acknowledge the received data. This is an example only. */
    USB_DEVICE_ControlStatus(appData.usbDevHandle, USB_DEVICE_CONTROL_STATUS_OK);
    break;
```

Endpoint Data Transfer Events

Describes endpoint data transfer events and provides a code example.

Description

The USB Device Layer provides notification events to indicate completion of transfers. These events are generated by the Device Layer and are made available in the Device Layer event handler. The USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE event occurs when a transfer scheduled by the USB_DEVICE_EndpointRead function has completed. The USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE event occurs when a transfer scheduled by the USB_DEVICE_EndpointWrite function has completed. The event data accompanying these events contains the transfer handle and number of bytes that were transferred.

The following code shows an example of handling these events.

```
/* The following code shows an example handling of the
 * endpoint transfer events. Here the code updates a transfer
 * pending flag indicating to the application that transfers have
 * completed. */
case USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE:
    /* Endpoint read is complete */
    appData.epDataReadPending = false;
    break;
case USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE:
    /* Endpoint write is complete */
    appData.epDataWritePending = false;
    break;
```

Endpoint Management

Describes how the application can enable and disable endpoints.

Description

Unlike standard USB function drivers, such as CDC, MSD, Audio, and HID, the Device Layer does not automatically manage endpoints for a Generic USB Device interface. This means that the application must maintain all endpoint that belong to a Generic USB Device Interface. Maintaining the endpoint involves the following:

- Enabling the endpoints for the desired transfer type when the host sets the configuration,
- Disabling the endpoint when the device receives a USB reset or when the Host changes the configuration
- Enabling and clearing endpoint stall conditions

Warning

The application should never access Endpoint 0 directly. Doing so may cause the Device Stack to malfunction, which could cause the USB device to be non-compliant.

Endpoints can be enabled or disabled with the USB_DEVICE_EndpointEnable and USB_DEVICE_EndpointDisable functions. The USB_DEVICE_EndpointIsEnabled function can be used to check if an endpoint is enabled. The application should enable the endpoint when host sets the configuration which contains interfaces that use the endpoint. The endpoints should otherwise be disabled. The endpoint function should not be called in the Device Layer event handler. Instead, they should be called in the application task routine. The following code shows an example of how an endpoint is enabled.

```
/* The following code shows an example of how the endpoint enable functions
 * are called to enabled a Receive and Transmit Bulk endpoints. Note that the size
 * of the endpoint must be specified and this size should match the endpoint size
 * mentioned in the endpoint descriptor */
if (USB_DEVICE_EndpointIsEnabled(appData.usbDevHandle, appData.endpointRx) == false )
{
    /* Enable Read Endpoint */
    USB_DEVICE_EndpointEnable(appData.usbDevHandle, 0, appData.endpointRx,
                              USB_TRANSFER_TYPE_BULK, sizeof(receivedDataBuffer));
}
if (USB_DEVICE_EndpointIsEnabled(appData.usbDevHandle, appData.endpointTx) == false )
{
    /* Enable Write Endpoint */
    USB_DEVICE_EndpointEnable(appData.usbDevHandle, 0, appData.endpointTx,
                             USB_TRANSFER_TYPE_BULK, sizeof(transmitDataBuffer));
}
```

An endpoint should be disabled when the host has changed the device configuration and the new configuration does not contain any interfaces that use this endpoint. The endpoint can also be disabled when the application receives USB_DEVICE_EVENT_RESET or when the USB_DEVICE_EVENT_DECONFIGURED event has occurred. The following code shows an example of disabling the endpoint.

```
/* In this example, the endpoints are disabled when
 * when the device has is not configured. This can happen
```

```
 * if the configuration set is 0 or if the device is reset. */
if(!appData.deviceIsConfigured)
{
    /* This means the device got deconfigured. Change the
     * application state back to waiting for configuration. */
    appData.state = APP_STATE_WAIT_FOR_CONFIGURATION;

    /* Disable the endpoint*/
    USB_DEVICE_EndpointDisable(appData.usbDevHandle, appData.endpointRx);
    USB_DEVICE_EndpointDisable(appData.usbDevHandle, appData.endpointTx);
    appData.epDataReadPending = false;
    appData.epDataWritePending = false;
}
```

The application can use the USB_DEVICE_EndpointStall and USB_DEVICE_EndpointStallClear functions to enable stall and clear the stall on endpoints. The USB_DEVICE_EndpointIsStalled function can be called to check stall status of the endpoint.

Endpoint Data Transfer

Describes how the application can transfer data over endpoints.

Description

The application should call the USB_DEVICE_EndpointRead and USB_DEVICE_EndpointWrite functions to transfer data over an enabled endpoint. Calling this function causes a USB transfer to be scheduled on the endpoint. The transfer is added to the endpoint queue and is serviced as the host schedules the transaction on the bus. The USB_DEVICE_EndpointRead and USB_DEVICE_EndpointWrite functions return a unique transfer handle which can be track the transfer. These transfer handles are returned along with the USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE (when a endpoint read transfer is complete) and USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE (when an endpoint write is complete) events.

The following code shows an example of sending data over an endpoint.

```
/* This code shows an example of using the USB_DEVICE_EndpointWrite
 * function to send data over the endpoint. The completion of the write is
 * indicated by the USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE event. The
 * transfer handle is returned in appData.writeTransferHandle */
USB_DEVICE_EndpointWrite ( appData.usbDevHandle, &appData.writeTranferHandle,
        appData.endpointTx, &transmitDataBuffer[0], sizeof(transmitDataBuffer),
        USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE );

void APP_USBDeviceEventHandler(USB_DEVICE_EVENT event, void * eventData, uintptr_t context)
{
    /* This is the Device Layer event handler */
    case USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE:
        /* Endpoint write is complete */
        appData.epDataWritePending = false;
        break;
}
```

The USB_DEVICE_EndpointWrite function allows the application to send data to the host without ending the transfer. This is done by specifying USB_DEVICE_TRANSFER_FLAGS_DATA_PENDING as the transfer flag in the call to the USB_DEVICE_EndpointWrite function. The application can use this option when the data to be sent is not readily available or when the application is memory constrained. The combination of the transfer flag and the transfer size affects how the data is sent to the host:

- If size is a multiple of maxPacketSize (the IN endpoint size) and flag is set as USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE, the write function will append a Zero Length Packet (ZLP) to complete the transfer
- If size is a multiple of maxPacketSize and flag is set as USB_DEVICE_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function will not append a ZLP and therefore and hence will not complete the transfer
- If size is greater than but not a multiple of maxPacketSize and flags is set as USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE, the write function schedules (length/maxPacketSize) packets and one packet for the residual data
- If size is greater than but not a multiple of maxPacketSize and flags is set as USB_DEVICE_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function returns an error code and sets the transferHandle parameter to USB_DEVICE_TRANSFER_HANDLE_INVALID
- If size is less than maxPacketSize and flag is set USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE, the write function schedules one packet
- If size is less than maxPacketSize and flag is set as USB_DEVICE_TRANSFER_FLAGS_MORE_DATA_PENDING, the write function returns an error code and sets the transferHandle parameter to USB_DEVICE_TRANSFER_HANDLE_INVALID

Refer to USB_DEVICE_EndpointWrite function API description for more details and code examples.

The application should use the USB_DEVICE_EndpointRead function to read data from an endpoint. The size of the buffer that is specified in this function should always be a multiple of the endpoint size. The following code shows an example of using the USB_DEVICE_EndpointRead function.

```
/* This code shows to use the USB_DEVICE_EndpointRead function
 * to read from an endpoint.  The transfer handle is returned in
```

```
 * appData.readTransferHandle. The size of receivedDataBuffer should
 * be a multiple of the receive endpoint size. */

USB_DEVICE_EndpointRead(appData.usbDevHandle, &appData.readTranferHandle,
             appData.endpointRx, &receivedDataBuffer[0], sizeof(receivedDataBuffer) );
void APP_USBDeviceEventHandler(USB_DEVICE_EVENT event, void * eventData, uintptr_t context)
{
    /* This is the Device Layer event handler */
    case USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE:
        /* Endpoint write is complete */
        appData.epDataReadPending = false;
        break;
}
```

In a case where a transfer is in progress, the USB_DEVICE_EndpointRead and USB_DEVICE_EndpointWrite functions can queue up transfers. The maximum number of read transfers that can queued (on any receive endpoint) is specified by the endpointQueueSizeRead member of the USB_DEVICE_INIT data structure. The maximum number of write transfers that can queued (on any transmit endpoint) is specified by the endpointQueueSizeWrite member of the USB_DEVICE_INIT data structure. The USB_DEVICE_ENDPOINT_QUEUE_DEPTH_COMBINED configuration macro should be set to total of read and write transfers that need to be queued.

For example, consider a Generic USB Device that contains two OUT (read) endpoint (EP1 and EP2) and one IN write endpoint (EP1). The application will queue a maximum of three read transfers on EP1, a maximum of five read transfers on EP2 and a maximum of four write transfers on EP1. Therefore, the total read transfer that will be queued in eight (3 + 5) and total write transfers that will be queued is four. The endpointQueueSizeRead member of the USB_DEVICE_INIT data structure should be set to eight. The endpointQueueSizeWrite member of the USB_DEVICE_INIT data structure should be set to four. The USB_DEVICE_ENDPOINT_QUEUE_DEPTH_COMBINED configuration macro should be set to 12 (8 + 4).

Configuring the Library

Describes how to configure the Generic USB Device Library.

Description

The application designer must specify the following configuration parameters while implementing the Generic USB Device. The configuration macros that implement these parameters must be located in the `system_config.h` file in the application project and a compiler include path (to point to the folder that contains this file) should be specified.

Configuration Macro Name	Description	Comments
USB_DEVICE_ENDPOINT_QUEUE_DEPTH_COMBINED	Size of buffer object pool for Endpoint Read and Endpoint Write functions.	This macro defines the total number of transfers that can be queued across all Generic USB Device endpoints. The number of read transfers that can be queued is specified by the endpointQueueSizeRead member of the USB_DEVICE_INIT data structure. The number of write transfers that can be queued is specified by the endpointQueueSizeWrite member of the USB_DEVICE_INIT data structure.

Building the Library

This section lists the files to be included in the project to implement a Generic USB Device Library.

Description

The Generic USB Device library does not have its own implementation. It is implemented using Device Layer API which are implemented in the Device Layer Files.

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/usb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
usb_device.h	This header file should be included in any `.c` file that accesses the Device Layer API needed to implement the Generic USB Device.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/usb_device.c	This file contains the Device Layer API implementation.
/src/dynamic/usb_device_endpoint_functions.c	This file contains the endpoint transfer and management routines that are needed to implement the Generic USB Device.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The Generic USB Device Library depends on the following modules:

- USB Device Layer Library

Library Interface

The API for implementing the Generic USB Device is contained the USB Device Library. Please refer to the Library Interface section in the USB Device Layer Library for more details.

USB Host Library

This section provides information on the USB Host libraries that are available in MPLAB Harmony.

USB Host Library - Getting Started

This section provides information for getting started with the USB Host Library.

Introduction

Provides an introduction to the MPLAB Harmony USB Host Library

Description

The MPLAB Harmony USB Host Library (referred to as the USB Host Library) provides embedded application developers with a framework to design and develop USB Host Support for a wide variety of USB Device Classes. Low-Speed and Full-Speed USB Devices can be supported with PIC32MX microcontrollers. High-Speed devices can be supported with PIC32MZ microcontrollers. The USB Host Library facilitates support of standard USB devices through client drivers that implement standard the USB Device class specification. The library is modular, thus allowing application developers to readily support composite USB devices.

The USB Host Library is a part of the MPLAB Harmony installation and is accompanied by demonstration applications that highlight library usage. These demonstration applications can also be modified or updated to build custom applications. The USB Host Library also features the following:

- Class Driver Support (CDC, Audio, HID, and MSD)
- Designed to support USB devices with multiple configurations at different speeds
- Supports low-speed, full-speed and high-speed operation
- Supports multiple USB peripherals (allows multiple host stacks)
- Modular and Layered architecture
- Completely non-blocking
- Supports both polled and interrupt operation
- Works readily in an RTOS environment
- Designed to readily integrate with other Harmony Middleware

This document serves as a getting started guide and provides information on the following:

- USB Host Stack Architecture
- USB Host Library - Application Interaction

Note: It is assumed that the reader is familiar with the USB 2.0 specification (available at www.usbif.org). While certain topics in USB may be discussed in this document, it is recommended that the reader refer to the specification documentation for a complete description.

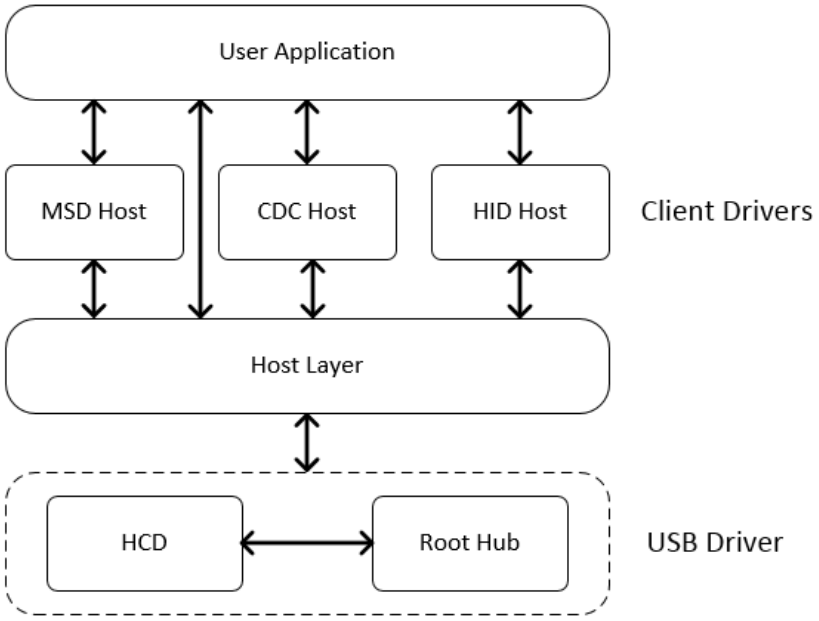
USB Host Library Architecture

Describes the USB Host Library Architecture.

Description

The USB Host Library Architecture features a modular and layered architecture as illustrated in the following figure.

USB Host Library Architecture



As seen in the figure, the USB Device Library consists of the following three major components.

Host Controller Driver (HCD)

The HCD manages the state of the USB peripheral and provides the Host Layer with structured methods to access data on the USB. The HCD is a MPLAB Harmony driver and uses the MPLAB Harmony framework components (USB Peripheral Library and the Interrupt System Service) of its operation. The HCD is initialized in the system initialization routine and its tasks routine is invoked in the system tasks routine. It is accessed exclusively by the Host layer. The HCD provides the following services to the host layer:

- Establish and manage communication pipes between the host layer and the attached devices
- Manage USB transfers

Root Hub Driver

The Root Hub Driver models the USB peripheral as a Hub. It then allows the Host Layer to perform the same actions on the Root Hub port that would be performed on an external Hub's port. The Root Hub Driver thus leads to an optimized implementation of Hub support in the Host Layer. The Root Hub Driver is hardware specific and is implemented as a part of the HCD. It provides the following services to the Host Layer

- Provides device attach and detach events
- Allows the Host to suspend, resume, and reset the port

The Root Hub Driver works in tandem with the HCD to provides the Host Layer with required USB protocol related means and methods to manage the attached USB device.

Host Layer

The Host Layer receives attach and detach events from the Root Hub Driver. It enumerates attached devices based on information contained in the Target Peripheral List (TPL). It allows client drivers to access the attached device through Host Layer methods. This includes allowing the client driver to set the device configuration. Where the client driver does not set the device configuration, the Host Layer will set the device configuration.

The Host layer opens the HCD, instantiates the Root Hub Driver, then controls and communicates with the attached device. The user application can call the Host Layer API to get information on attached devices. It can also register a Host Layer Event handler to get device related events. The user application can additionally suspend or resume a device. The Host Layer also provides bus level control where the application can

suspend or resume all devices connected to a USB.

Client Driver

The USB Host Stack Client Drivers implement the support for different device classes as per the class specifications. Along with Host Layer, the client drivers are designed to support multiple device of the same type (where multiple devices are connected to the host through a hub or is a single device with multiple interfaces). A client driver abstracts intricate details of the class specification and provides a high level command and data interface to the application. Completion of requests is indicated by events. The application must register an event handler to receive these events.

The Client Driver may manage devices whose functionality is specified by USB VID and PID. In such cases, the client driver can set the device configuration. The client driver may manage a device whose functionality is defined by an interface class, subclass and protocol. In such a case, the configuration is set by the Host layer. The client driver can also manage devices whose functionality is defined by a combination of VID PID and class, sub-class and protocol.

USB Host Library - Application Interaction

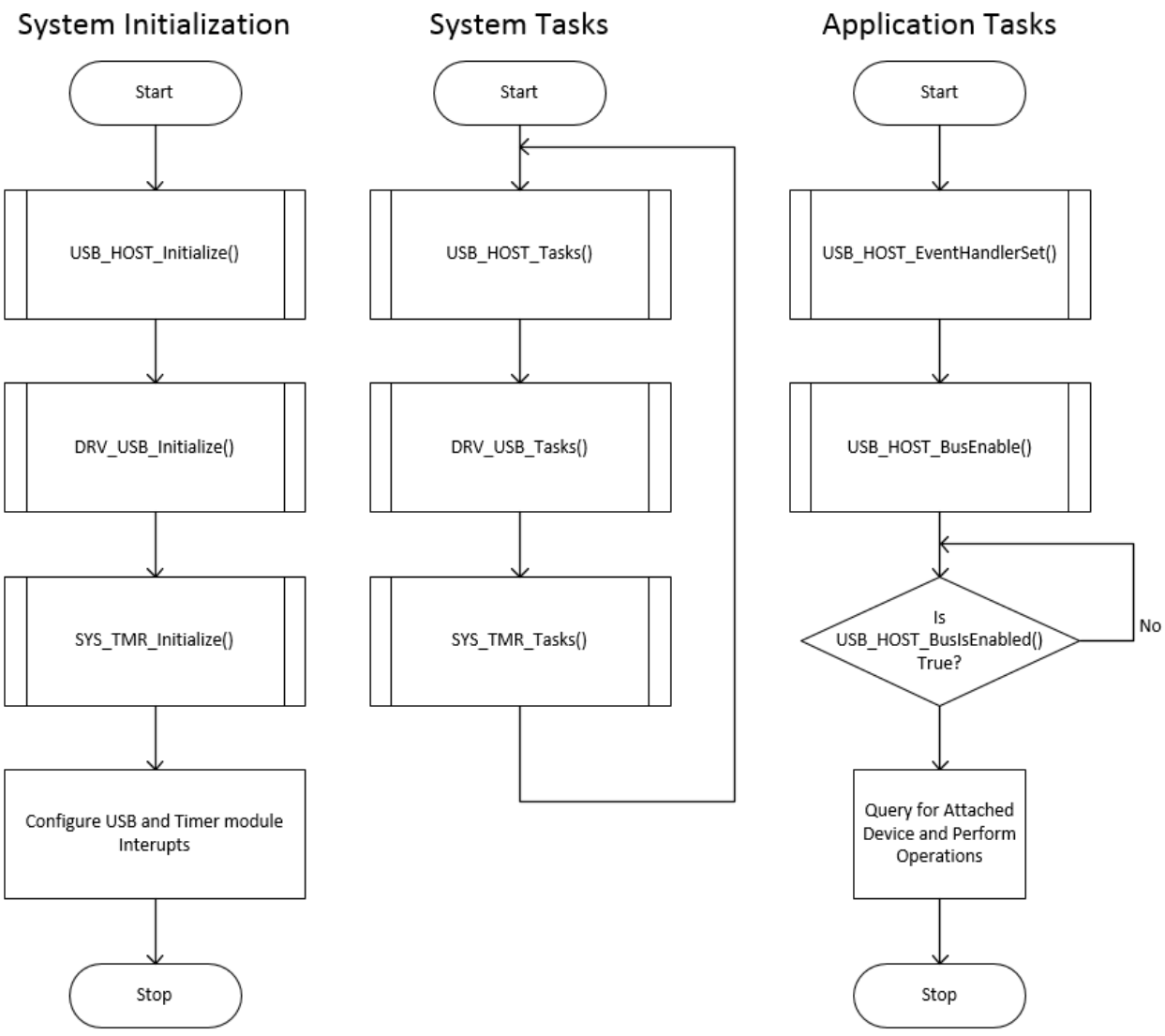
Describes how the application must interact with the USB Host Stack.

Description

Note: Additional information on the tests conducted on Flash devices (i.e., Pen Drives) and a list of USB application configurations is available in the USB Demonstrations section.

The following figure highlights the steps that the application must follow to use the USB Host Library.

Application Interaction with Host Layer



The USB Host stack is initialized in the MPLAB Harmony System Initialization function. The Host Stack requires the Timer System Service and

USB Driver. So these must be initialized as well. Note that the figure refers to a general USB Driver. The application may use the USBFS Driver (DRV_USBFS) for PIC32MX microcontroller or use the USBHS Driver (DRV_USBHS) for PIC32MZ microcontroller. The Timer and USB module interrupt priorities must be configured.

The USB Host layer, the USB Driver and the Timer System Service tasks must be called in the MPLAB Harmony System Tasks Routine. This ensures that the state machines of these module stays updated. If the USB Driver and the Timer driver have been configured for interrupt operation, then their corresponding interrupt tasks routines should be called in the corresponding module interrupt service routines.

The application state machine must first set the Host Layer event handler and then enable the bus. Enabling the bus will enable device detection and the Host Layer will enumerate attached devices. The application can query for attached devices and perform operations on attached devices.

USB Host Library Migration Guide

This section provides information on migrating from MPLAB Harmony v1.03.01 and earlier to the MPLAB Harmony USB Host Stack in MPLAB Harmony v1.04 and later.

Introduction

Provides an introduction to migrating from older versions to v1.04 and later versions of MPLAB Harmony.

Description

The USB Host Stack API in MPLAB Harmony v1.04 has changed from that of previous versions of MPLAB Harmony.

USB CDC and MSD Host applications that were developed using the MPLAB Harmony USB Host Stack v1.03.01 and earlier, will not build unless the application calls to the USB Host Stack API is updated. While the MHC utility provides an option to continue creating USB Host applications using the v1.03.01 and earlier USB Host Stack API, it is recommended that existing USB Host application migrate to the latest USB Host API to take advantage of the latest features in the USB Host Stack. The following sections describe the API changes and other considerations while updating the application for changes in the USB Host Stack.

Note: All USB Host Stack Demonstration Applications and USB Host Stack related documentation have been updated to the latest (new) USB Host API. The following sections do not discuss changes in the USB Host Stack configuration related code. This is updated automatically when the project is re-generated using MHC utility. Only the application related API changes are discussed.

USB Host Layer

Describes differences in the USB Host Library. Describes differences from earlier versions of MPLAB Harmony and the USB Host Layer Library in v1.04 of MPLAB Harmony.

Description

In MPLAB Harmony v1.03.01, the application was required to open the Host Layer to obtain a handle by calling the USB_HOST_Open function. This handle was then used along with other Host Layer API. Once opened, the application must enable Host Layer operation by calling the USB_HOST_OperationEnable function. The application must check the status of the enable operation function by calling the USB_HOST_OperationIsEnabled function. This is shown in the following code example.

Example:

```
/* This code shows an example application tasks that enabled Host
 * operation and waits for the enable operation to complete */

void APP_Tasks ( void )
{
    /* Check the application's current state. */
   USB_HOST_CDC_RESULT result;
    uint8_t  temp;

    switch (appData.state)
    {
        case APP_STATE_OPEN_HOST_LAYER:

            /* Open the host layer and then enable Host layer operation */
            appData.hostHandle = USB_HOST_Open(USB_HOST_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);

            if (appData.hostHandle != USB_HOST_HANDLE_INVALID)
            {
                /* Host layer was opened successfully. Enable operation
                 * and then wait for operation to be enabled  */

                USB_HOST_OperationEnable(appData.hostHandle );
                appData.state = APP_STATE_WAIT_FOR_HOST_ENABLE;
```

```
            }
            break;

        case APP_STATE_WAIT_FOR_HOST_ENABLE:

            /* Check if the host operation has been enabled */
            if(USB_HOST_OperationIsEnabled(appData.hostHandle))
            {
                /* This means host operation is enabled. We can
                 * move on to the next state */
                appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            }

            break;

        default:
            break;
    }
}
```

In MPLAB Harmony v1.04, the application is not required to open the Host Layer. This is because the Host Layer by itself does not support multi-client operation. The application must first set the Host Layer event handler using the USB_HOST_EventHandlerSet function and then enable the desired bus using the USB_HOST_BusEnable function. The completion of the Bus Enable function can be checked by calling the USB_HOST_BusIsEnabled function. Once enabled, the application can perform operations on the bus, look for attached devices, and perform operations on the attached device. The following code shows an example of enabling the bus.

```
/* This code shows an example of how the bus is enabled in the MPLAB Harmony v1.04
 * USB Host Stack. The application state machine then waits for the bus enable
 * operation to complete */

void APP_Tasks ( void )
{
    switch (appData.state)
    {
        case APP_STATE_BUS_ENABLE:

            /* In this state the application enables the USB Host Bus. Note
             * how the Host event handler are registered before the bus
             * is enabled. */

            USB_HOST_EventHandlerSet(APP_USBHostEventHandler, (uintptr_t)0);
            USB_HOST_BusEnable(0);
            appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
            break;

        case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:

            /* In this state we wait for the Bus enable to complete */
            if(USB_HOST_BusIsEnabled(0))
            {
                appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            }
            break;

        default:
            break;
    }
}
```

API Changes

The following table shows the legacy Host API and corresponding v1.04 MPLAB Harmony Host Stack API.

USB Host Layer API

MPLAB Harmony v1.03.01 and Earlier	MPLAB Harmony v1.04 and Later
USB_HOST_Open	N/A The USB Host Layer does not have to be opened.

USB_HOST_Close	N/A The USB Host Layer does not have to be closed.
USB_HOST_EventCallBackSet	USB_HOST_EventHandlerSet Events have changed.
USB_HOST_OperationEnable	USB_HOST_BusEnable Multiple busses can be managed.
USB_HOST_OperationDisable	N/A A bus disable function is not currently available and will be added in a future release of MPLAB Harmony.
USB_HOST_OperationIsEnabled	USB_HOST_BusIsEnabled
USB_HOST_DeviceSuspend	USB_HOST_DeviceSuspend Function parameters have changed.
USB_HOST_DeviceResume	USB_HOST_DeviceResume Function parameters have changed.
N/A	USB_HOST_BusSuspend Provides multiple bus support.
N/A	USB_HOST_BusResume Provides multiple bus support.
N/A	USB_HOST_DeviceGetFirst Provides multiple bus support.
N/A	USB_HOST_DeviceGetNext Provides multiple bus support.
N/A	USB_HOST_DeviceSpeedGet
N/A	USB_HOST_DeviceIsSuspended
N/A	USB_HOST_DeviceStringDescriptorGet

Event Changes

The type of events that the Host Layer generates in earlier versions of MPLAB Harmony and v1.04 of the MPLAB Harmony USB Host Stack have changed. The following table shows a comparison.

USB Host Layer Events

MPLAB Harmony v1.03.01 and Earlier	MPLAB Harmony v1.04 and Later
N/A	USB_HOST_EVENT_DEVICE_REJECTED_INSUFFICIENT_POWER
USB_HOST_EVENT_UNSUPPORTED_DEVICE	USB_HOST_EVENT_DEVICE_UNSUPPORTED
USB_HOST_EVENT_CANNOT_ENUMERATE	This event is not needed and is the same as an unsupported device.
USB_HOST_EVENT_CONFIGURATION_FAILED	This event is not needed and is the same as an unsupported device.
USB_HOST_EVENT_DEVICE_SUSPENDED	This event is not needed. A polling based function is available.
USB_HOST_EVENT_DEVICE_RESUMED	This event is not needed. A polling based function is available.
N/A	USB_HOST_EVENT_HUB_TIER_LEVEL_EXCEEDED
N/A	USB_HOST_EVENT_PORT_OVERCURRENT_DETECTED

USB MSD Host Client Driver and SCSI Block Storage Driver

Provides migration information for the MSD Host Client Driver and the SCSI Block Storage Driver.

Description

The application would use the MSD Host Client Driver and SCSI Block Storage Driver for accessing USB Storage Devices, such as USB Pen Drives. The key difference between the MSD Host Client Drivers in previous versions of the MPLAB Harmony USB Host Stack and the v1.04 MPLAB Harmony USB Host Stack MSD Host Client Drivers is the way the storage device attach and detach events are handled.

In previous versions (i.e., v1.03.01 or earlier) of USB Host Stack MSD Host Client Driver an application would have to register an event handler using the USB_HOST_MSDEventHandlerSet) function after Host operation has been enabled. When the application would receive an event, USB_HOST_MSD_EVENT_ATTACH would be detected, and the application would then try to mount the drive in this event. This is shown in the following code example.

Example:

```
/* In a v1.03.01 or earlier MSD Host Client Driver implementation, the driver will send an
 * attach detach event to the application. The application must use this to mount or unmount
 * the drive in its main state machine */

bool APP_USBHostMSDEventHandler
(
    USB_HOST_MSD_INDEX index,
    USB_HOST_MSD_EVENT event,
    void* pData
)
{
    switch ( event)
    {
        case USB_HOST_MSD_EVENT_ATTACH:

            /* This means a USB storage device was plugged in. Update the
             * application state */
            appData.state =  APP_STATE_DEVICE_CONNECTED;


            break;

        case USB_HOST_MSD_EVENT_DETACH:

            /* This means the USB storage was unplugged. Update the application
             * state */
            appData.state = APP_STATE_UNMOUNT_DISK;
            break;

        default:
            break;
    }

    return 0;
}

/* In the main application task routine, the host layer is opened and the host
 * operation is enabled. A MSD Host Client Driver event handler is registered
 * and disk is mounted when an attach event has been detected */

void APP_Tasks ( void )
{
    /* The application task state machine */

    switch(appData.state)
    {
        case APP_STATE_OPEN_HOST_LAYER:

            /* Open the host layer and then enable Host layer operation */
            appData.hostHandle = USB_HOST_Open(USB_HOST_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);

            if (appData.hostHandle != USB_HOST_HANDLE_INVALID)
            {
                /* Host layer was opened successfully. Enable operation
                 * and then wait for operation to be enabled  */

                USB_HOST_OperationEnable(appData.hostHandle );
                appData.state = APP_STATE_WAIT_FOR_HOST_ENABLE;

            }
            break;

        case APP_STATE_WAIT_FOR_HOST_ENABLE:

            /* Check if the host operation has been enabled */
            if(USB_HOST_OperationIsEnabled(appData.hostHandle))
            {
                /* This means host operation is enabled. We can
```

```
                 * move on to the next state */

                USB_HOST_EventCallBackSet(appData.hostHandle,APP_USBHostEventHandler , 0 );
                USB_HOST_MSD_EventHandlerSet (APP_USBHostMSDEventHandler );
                appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            }

            break;

        case APP_STATE_WAIT_FOR_DEVICE_ATTACH:

            /* Wait for device attach. The state machine will move
             * to the next state when the attach event
             * is received.  */

            break;

        case APP_STATE_DEVICE_CONNECTED:

            /* Device was connected. We can try mounting the disk */
            appData.state = APP_STATE_MOUNT_DISK;
            break;

        case  APP_STATE_MOUNT_DISK:

            /* The application gets into this state when the drive is attached
             * */

            if(SYS_FS_Mount("/dev/sda1", "/mnt/myDrive", FAT, 0, NULL) != 0)
            {
                /* The disk could not be mounted. Try
                 * mounting again until success. */

                appData.state = APP_STATE_MOUNT_DISK;
            }
            else
            {
                /* Mount was successful. Try opening the file */
                appData.state = APP_STATE_OPEN_FILE;
            }
            break;

        case APP_STATE_UNMOUNT_DISK:

            /* The application gets into this state when the drive is detached
             * */

            if(SYS_FS_Unmount("/mnt/myDrive") != 0)
            {
                /* The disk could not be unmounted. Try
                 * unmounting again until success. */

                appData.state = APP_STATE_UNMOUNT_DISK;
            }
            else
            {
                /* Unmount was successful. Wait for device attach */
                appData.state =  APP_STATE_WAIT_FOR_DEVICE_ATTACH;

            }
            break;

        default:
            break;
    }
}
```

In the v1.04 USB Host Stack MSD Host Client Driver, the application must use the auto mount feature of the MPLAB Harmony File System. This feature should be enabled when the project is generated via MHC. The application then uses the MPLAB Harmony File System event handler to know when the File System has mounted the attached USB Storage Device. The application does not have to explicitly mount or unmount the

drive. This is shown in the following code example.

Example:

```
/* In v1.04 USB MSD Host Client Driver, the driver uses the auto mount feature
 * of the MPLAB Harmony File System. The application must make sure that this
 * feature is enabled when the project is generated using MHC. Note that unlike
 * previous implement, here the USB Storage device attach detach event appears
 * as a File System Event. */

void APP_SYSFSEventHandler(SYS_FS_EVENT event, void * eventData, uintptr_t context)
{
    switch(event)
    {
        case SYS_FS_EVENT_MOUNT:

            /* The file system generates this event when the drive is mounted.
             * This happens when the USB storage device is connected */
            appData.deviceIsConnected = true;
            break;

        case SYS_FS_EVENT_UNMOUNT:

            /* The file system generates this event when the drive is unmounted.
             * This happens when the USB storage device is disconnected */
            appData.deviceIsConnected = false;

            break;

        default:
            break;
    }
}

/* This is the application state machine. Note how the application register the
 * event handler with the File System before enabling the bus */

void APP_Tasks ( void )
{
    switch(appData.state)
    {
        case APP_STATE_BUS_ENABLE:

            /* Set the event handler and enable the bus */
            SYS_FS_EventHandlerSet(APP_SYSFSEventHandler, (uintptr_t)NULL);
            USB_HOST_EventHandlerSet(APP_USBHostEventHandler, 0);
            USB_HOST_BusEnable(0);
            appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
            break;

        case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:
            if(USB_HOST_BusIsEnabled(0))
            {
                appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            }
            break;

        case APP_STATE_WAIT_FOR_DEVICE_ATTACH:

            /* Wait for device attach. The state machine will move
             * to the next state when the attach event
             * is received.  */
            if(appData.deviceIsConnected)
            {
                appData.state = APP_STATE_DEVICE_CONNECTED;
            }

            break;

        case APP_STATE_DEVICE_CONNECTED:
```

```
            /* Device was connected. We can try mounting the disk */
            appData.state = APP_STATE_OPEN_FILE;
            break;

        case APP_STATE_IDLE:

            /* The application reaches here after completing operation and waits
             * for device detach and if detached, wait for attach. */
            if(appData.deviceIsConnected == false)
            {
                appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            }
            break;

        default:
            break;
    }
}
```

USB CDC Host Client Driver

Describes differences from earlier versions of MPLAB Harmony and the USB CDC Host Client Driver API in v1.04 of MPLAB Harmony.

Description

The key differences between the USB CDC Host Client Driver API in earlier versions of MPLAB Harmony and MPLAB Harmony v1.04 are:

- In the v1.04 release, the Device Attach event handler must be registered before enabling the bus and the General CDC Host Client Driver Event Handler is registered after the attached CDC device has been opened. In v1.03.01 and earlier, there was only one event handler which received all events and this event handler was to be registered after the Host Operation was enabled.
- In the v1.04 release, the application must open the attached CDC device. In v1.03.01 and earlier, the application does not have to open the attached CDC Device.

Registering Events

In v1.03.01 and earlier versions of the USB CDC Host Client Driver, the application registers the CDC Event Handler after the Host Operation was enabled. Only one event handler could be registered for all attached CDC Devices. The CDC Host Client Driver would specify the instance of the CDC Device generating the event when the event handler was invoked. The following code shows an example of this.

```
/* This code shows how a single CDC event handler is registered for all
 * instances of the CDC Host Client Driver. The instance of the client driver
 * generating this event is available in the index parameter of the event
 * handler. This code example is applicable to pre-v1.04 releases of the client
 * driver */

USB_HOST_CDC_EVENT_RESPONSE APP_USBHostCDCEventHandler
(
    USB_HOST_CDC_INDEX index,
    USB_HOST_CDC_EVENT event,
    void * eventData,
    uintptr_t context
)
{
    /* Get the application context */
    uint8_t deviceAddress;

    switch(event)
    {
        case USB_HOST_CDC_EVENT_ATTACH:

            /* The event data in this case is the address of the
             * attached device. */

            appData.state = APP_STATE_DEVICE_CONNECTED;
            deviceAddress = *((uint8_t *)eventData);
            break;

        case USB_HOST_CDC_EVENT_DETACH:

            /* This means the device was detached. There is no event data
             * associated with this event.*/
```

```
            appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            break;

            /* Other events not shown here for the sake of brevity */

        default:
            break;
    }

    return USB_HOST_CDC_EVENT_RESPONSE_NONE;
}

/* This is a pre v1.04 release application tasks for a CDC host client driver.
 * Note that application does not open the client driver and hence when
 * registering an event handler, is registering it all CDC Host Client Driver
 * instances. */

void APP_Tasks ( void )
{
    /* Check the application's current state. */
    USB_HOST_CDC_RESULT result;
    uint8_t  temp;

    switch (appData.state)
    {
        case APP_STATE_OPEN_HOST_LAYER:

            /* Open the host layer and then enable Host layer operation */
            appData.hostHandle = USB_HOST_Open(USB_HOST_INDEX_0, DRV_IO_INTENT_EXCLUSIVE);

            if (appData.hostHandle != USB_HOST_HANDLE_INVALID)
            {
                /* Host layer was opened successfully. Enable operation
                 * and then wait for operation to be enabled  */

                USB_HOST_OperationEnable(appData.hostHandle );
                appData.state = APP_STATE_WAIT_FOR_HOST_ENABLE;

            }
            break;

        case APP_STATE_WAIT_FOR_HOST_ENABLE:

            /* Check if the host operation has been enabled */
            if(USB_HOST_OperationIsEnabled(appData.hostHandle))
            {
                /* This means host operation is enabled. We can
                 * move on to the next state */
                USB_HOST_EventCallBackSet(appData.hostHandle,APP_USBHostEventHandler , 0 );
                USB_HOST_CDC_EventHandlerSet (APP_USBHostCDCEventHandler );
                appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            }

            break;

        case APP_STATE_WAIT_FOR_DEVICE_ATTACH:

            /* Wait for device attach. The state machine will move
             * to the next state when the USB_HOST_CDC_EVENT_ATTACH
             * is received. The application state is update in the
             * CDC Host event handler */

            break;

        default:
            break;
    }
}
```

In the v1.04 released version of the USB CDC Host Client Driver, the application must register an Attach Event Handler before enabling the bus operation. This one Attach Event handler will be invoked when even a CDC Device is attached. The application must use the CDC object returned in the attach event handler to open the device. The application can then register an event handler using the handle returned by the open function. This is shown in the code snippet here.

```
/* This code how event handling is performed in the v1.04 release of the
 * CDC Host Client Driver. The application must register a listener function
 * that check if the CDC Device is attached. When the attached device is opened
 * the application must then register event handler to register other CDC Host
 * Client Driver events */

void APP_USBHostCDCAttachEventListener(USB_HOST_CDC_OBJ cdcObj, uintptr_t context)
{
    /* This function gets called when the CDC device is attached. Update the
     * application data structure to let the application know that this device
     * is attached */

    appData.deviceIsAttached = true;
    appData.cdcObj = cdcObj;
}


USB_HOST_CDC_EVENT_RESPONSE APP_USBHostCDCEventHandler
(
    USB_HOST_CDC_HANDLE cdcHandle,
    USB_HOST_CDC_EVENT event,
    void * eventData,
    uintptr_t context
)
{
    /* This function is called when a CDC Host event has occurred. A pointer to
     * this function is registered after opening the device. See the call to
     * USB_HOST_CDC_EventHandlerSet() function. */

    switch(event)
    {
        case USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE:

            /* This means the application requested Set Line Coding request is
             * complete. */
            break;

        case USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE:

            /* This means the application requested Set Control Line State
             * request has completed. */
            break;

        case USB_HOST_CDC_EVENT_WRITE_COMPLETE:

            /* This means an application requested write has completed */
            break;

        case USB_HOST_CDC_EVENT_READ_COMPLETE:

            /* This means an application requested write has completed */
            break;

        case USB_HOST_CDC_EVENT_DEVICE_DETACHED:

            /* The device was detached */
            appData.deviceWasDetached = true;
            break;

        default:
            break;
    }

    return(USB_HOST_CDC_EVENT_RESPONE_NONE);
}
```

```
/* This is the example application task routine. The application sets the CDC
 * attach listener function and then enables bus. When the CDC device is
 * attached, the state machine opens the CDC Host Client Driver suing the object
 * that was returned in the attach function. An event handler is then set with
 * using the handle returned by the CDC Host Client Driver Open Function */

void APP_Tasks ( void )
{
    /* Check the application's current state. */
    USB_HOST_CDC_RESULT result;

    if(appData.deviceWasDetached)
    {
        /* This means the device is not attached. Reset the application state */

        appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
    }

    switch (appData.state)
    {
        case APP_STATE_BUS_ENABLE:

            /* In this state the application enables the USB Host Bus. Note
             * how the CDC Attach event handler are registered before the bus
             * is enabled. */

            USB_HOST_EventHandlerSet(APP_USBHostEventHandler, (uintptr_t)0);
            USB_HOST_CDC_AttachEventHandlerSet(APP_USBHostCDCAttachEventListener, (uintptr_t) 0);
            USB_HOST_BusEnable(0);
            appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
            break;

        case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:

            /* In this state we wait for the Bus enable to complete */
            if(USB_HOST_BusIsEnabled(0))
            {
                appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            }
            break;

        case APP_STATE_WAIT_FOR_DEVICE_ATTACH:

            /* In this state the application is waiting for the device to be
             * attached */
            if(appData.deviceIsAttached)
            {
                /* A device is attached. We can open this device */
                appData.state = APP_STATE_OPEN_DEVICE;
                appData.deviceIsAttached = false;
            }
            break;

        case APP_STATE_OPEN_DEVICE:

            /* In this state the application opens the attached device */
            appData.cdcHostHandle = USB_HOST_CDC_Open(appData.cdcObj);
            if(appData.cdcHostHandle != USB_HOST_CDC_HANDLE_INVALID)
            {
                /* The driver was opened successfully. Set the event handler
                 * and then go to the next state. */
                USB_HOST_CDC_EventHandlerSet(appData.cdcHostHandle, APP_USBHostCDCEventHandler,
(uintptr_t)0);
                appData.state = APP_STATE_SET_LINE_CODING;
            }
            break;

        default:
```

```
            break;
        }
    }
```

API Differences

The following table shows the difference in API names between the pre v1.04 release and the v1.04 release versions of the CDC Host Client Driver.

MPLAB Harmony v1.03.01 and Earlier	MPLAB Harmony v1.04 and Later
USB_HOST_CDC_EventHandlerSet	USB_HOST_CDC_EventHandlerSet
USB_HOST_CDC_Read	USB_HOST_CDC_Read
USB_HOST_CDC_Write	USB_HOST_CDC_Write
USB_HOST_CDC_SerialStateNotificationGet	USB_HOST_CDC_SerialStateNotificationGet
USB_HOST_CDC_LineCodingGet	USB_HOST_CDC_ACM_LineCodingGet
USB_HOST_CDC_ControlLinetStateSet	USB_HOST_CDC_ACM_ControlLineStateSet
USB_HOST_CDC_BreakSend	USB_HOST_CDC_ACM_BreakSend
Not needed.	USB_HOST_CDC_EventHandlerSet The attach event requires a separate event handler in v1.04.
Not available.	USB_HOST_CDC_DeviceObjHandleGet
Not available.	USB_HOST_CDC_Open The driver needs to be opened in v1.04.
Not available.	USB_HOST_CDC_Close

Event Name Differences

The following table shows the difference in event names between earlier versions and the MPLAB Harmony v1.04 release of the CDC Host Client Driver.

MPLAB Harmony v1.03.01 and Earlier	MPLAB Harmony v1.04 and Later
USB_HOST_CDC_EVENT_READ_COMPLETE	USB_HOST_CDC_EVENT_READ_COMPLETE
USB_HOST_CDC_EVENT_WRITE_COMPLETE	USB_HOST_CDC_EVENT_WRITE_COMPLETE
USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED	USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED
USB_HOST_CDC_EVENT_GET_LINE_CODING_COMPLETE	USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE
USB_HOST_CDC_EVENT_SET_LINE_CODING_COMPLETE	USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE
USB_HOST_CDC_EVENT_SET_CONTROL_LINE_STATE_COMPLETE	USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE
USB_HOST_CDC_EVENT_SEND_BREAK_COMPLETE	USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE
USB_HOST_CDC_EVENT_DETACH	USB_HOST_CDC_EVENT_DEVICE_DETACHED
USB_HOST_CDC_EVENT_ATTACH	Not needed. Refer to the USB_HOST_AttachEventHandlerSet function.

USB Host Layer Library

This section describes the USB Host Layer Library.

Introduction

Introduces the MPLAB Harmony USB Host Layer Library.

Description

The USB Host Layer in the MPLAB Harmony USB Host Stack performs the tasks of enumerating an attached device and interfacing the HCD. The following are the key features of the MPLAB Harmony USB Host Layer:

- Supports multi-configuration and composite USB Devices

- Supports VID PID and class, subclass, and protocol devices
- Can manage multiple USB devices through the Root Hub
- Concise API simplifies application development
- Modular architecture allows support for multiple (and different) USB controller in one application. Can operate multiple USB segments.
- Supports Low-Speed, Full-Speed, and Hi-Speed USB devices

Using the Library

This topic describes the basic architecture of the USB Host Layer and provides information and examples on its use.

Abstraction Model

Describes the abstraction model of the USB Host Layer.

Description

The USB Host Layer abstracts USB HCD hardware interaction details and presents an easy-to-use interface to the application and the client drivers. The Host Layer provides the application with a device object handle, which the application can use to suspend or resume the device. The Host Layer provides client drivers with device client handles and interface handles. These handles allow the client drivers to interact with the device and its interfaces. The Host Layer allows the client drivers to

- Open control pipes and schedule control transfers
- Open bulk, isochronous, and interrupt pipes
- Perform data transfers
- Claim and release ownership of the device and device interfaces
- Perform standard device operations.

The Host Layer has exclusive access to the HCD and the Root Hub. It opens the HCD and presents an abstracted interface to the application and client drivers.

Library Overview

The USB Host layer API is grouped functionally, as shown in the following table.

Library Interface Section	Description
System Interface Functions	These functions make the USB Host Layer compatible with MPLAB Harmony.
Bus Control Functions	These functions allow the application to enable, disable, suspend and resume the USB.
Device Related Functions	These functions allow the application to suspend and resume the USB. Attached devices can be queried and their string descriptors can be obtained.
Event Handling	Allows the application to register an event handler.
Client Driver Routines	These functions are exclusive to the client drivers and should not be accessed by the application.

How the Library Works

Describes how the Library works and how it should be used.

Description

The Host Layer in the MPLAB Harmony USB Host Stack plays the key role of enumerating an attached device and facilitating the communication between the USB Host Client Driver and the attached devices. The following sections describe the steps and methods that the user application must follow to use the Host Layer (and the USB Host stack). The following topics are discusses:

- Host Layer Initialization
- Operating the Host layer
- Host Layer Application Events

Host Layer Initialization

This topic describes how to initialize the Host Layer and includes code examples.

Description

The Host Layer must be initialized with relevant data to enable correct operation. This initialization must be performed in the SYS_Initialize function of the MPLAB Harmony application. The Host Layer will require the USB Controller Peripheral driver to be initialized for host mode operation (and

hence operate as a HCD). This initialization must be performed in the SYS_Initialize function. The order in which the Host Layer and the USB Peripheral Driver are initialized does not affect the Host Layer operation. The Host Layer could be initialized before or after the USB Controller Peripheral Driver initialization.

The Host Layer requires the following information for initialization:

- The HCD interface for each bus
- The Target Peripheral List (TPL)

The Host Layer is capable of operating more than one USB device. This is possible on PIC32 microcontrollers that feature multiple USB Controller Peripherals. The one instance of the Host Layer manages multiple HCDs. The interface to each to every instance of the HCD that the Host Layer must operate must be specified in the Host Layer initialization. The total number of USB devices the Host Layer should manage is defined statically by the USB_HOST_CONTROLLERS_NUMBER configuration macro in the `system_config.h` file. The following code shows an example initialization of a PIC32MX USB HCD.

Example: PIC32MZ USB HCD Initialization

```
/* This code shows an example of how to initialize the PIC32MX USB
 * Driver for host mode operation. For more details on the PIC32MX Full-Speed
 * USB Driver, please refer to the Driver Libraries documentation. */

/* Include the full-speed USB driver header file */
#include "driver/usb/usbfs/drv_usbfs.h"

/* Create a driver initialization data structure */
DRV_USBFS_INIT drvUSBFSInit;

/* The PIC32MX Full-Speed USB Driver when operating in host mode requires an
 * endpoint table (a byte array) whose size should be 32 bytes. This table should
 * be aligned at 512 byte address boundary */
uint8_t __attribute__((aligned(512))) endpointTable[32];

/* Configure the driver initialization data structure */
DRV_USBFS_INIT drvUSBFSInit =
{
    /* This parameter should be set to SYS_MODULE_POWER_RUN_FULL. */
    .moduleInit = {SYS_MODULE_POWER_RUN_FULL},

    /* Driver operates in Host mode */
    .operationMode = USB_OPMODE_HOST,

    /* USB module interrupt source */
    .interruptSource = INT_SOURCE_USB_1,

    /* Continue operation when CPU is in Idle mode */
    .stopInIdle = false,

    /* Do not suspend operation when CPU enters Sleep mode */
    .suspendInSleep = false,

    /* The USB module index */
    .usbID = USB_ID_1,

    /* The maximum current that the VBUS supply can provide */
    .rootHubAvailableCurrent = 500,

    /* Pointer to the endpoint table */
    .endpointTable = endpointTable,

    /* Pointer to the Port Power Enable function. Driver will cause this
     * function when the port power must be enabled */
    .portPowerEnable = PortPowerEnable,

    /* Pointer to the Port Over Current Detect function. Driver will cause this
     * function periodically to check if the port current has exceeded limit */
    .portOverCurrentDetect = PortOverCurrentDetect,

    /* Pointer to the Port LED indication function. The driver will call this
     * function to update the Port LED status */
    .portIndication = PortIndication

};
```

```
/* USB Driver system module object */
SYS_MODULE_OBJ drvUSBObj = SYS_MODULE_OBJ_INVALID;

void SYS_Initialize(void * data)
{
    /* Initialize the driver */
    drvUSBObj = DRV_USBFS_Initialize(DRV_USBFS_INDEX_0, (SYS_MODULE_INIT *)(&drvUSBFSInit));
}

void SYS_Tasks(void)
{
    /* Call the driver tasks routine in SYS_Tasks() function *//
    DRV_USBFS_Tasks(drvUSBObj);
}

void __ISR(_USB_1_VECTOR, ipl4AUTO) _IntHandlerUSBInstance0(void)
{
    /* Call the driver interrupt tasks routine in the USB module ISR */
    DRV_USBFS_Tasks_ISR(sysObj.drvUSBModuleObj);
}
```

The Host Layer Initialization requires a USB_HOST_HCD data structure. This data structure specifies the HCD module index and the HCD Host Layer Interface for each bus. The following code shows the USB_HOST_HCD data structure is initialized for a single USB Controller Peripheral PIC32MX microcontroller device.

Example: Data Structure Initialized for a Single USB Controller Peripheral PIC32MX MCU

```
/* This code shows an example of setting up the USB_HOST_HCD data
 * structure for the PIC32MX USB controller */

USB_HOST_HCD usbHostHCD =
{
    /* This is the driver instance index that the USB Host Layer will use */
    .drvIndex = DRV_USBFS_INDEX_0,

    /* This is the interface to the PIC32MX USB HCD. The
     * DRV_USBHS_HOST_INTERFACE pointer is exported by the PIC32MX Host Mode USB
     * Driver. */
    .hcdInterface = DRV_USBHS_HOST_INTERFACE
};
```

The other important component required for USB Host Layer initialization is the Target Peripheral List (TPL). Embedded USB Hosts unlike standard USB Host are not expected to support all USB Device Types. The device types to be supported are specified in the TPL. The TPL contains an entry for every device type that the Embedded USB host must support. If the attached device matches the criteria specified in the TPL entry , the Host Layer attaches the driver corresponding to that entry to the manage device. A device may match multiple entries in the TPL. This happens in the case of composite devices.

An entry in the TPL contains the following information:

- Device Type: This specifies whether the Host must inspect the VID, PID field or Class, Subclass and Protocol fields while matching the attached device to the entry
- Flags: These flags provide the system designer with various options while matching the attached device to a driver. For example, a flag can be specified to ignore the device PID and only consider the VID while matching VID PID device.
- PID Mask: This is a PID mask that can be applied to the PID before matching the PID to the attached device PID
- Driver: This is the pointer to the interface of the client driver that should manage the device if the matching criteria is met

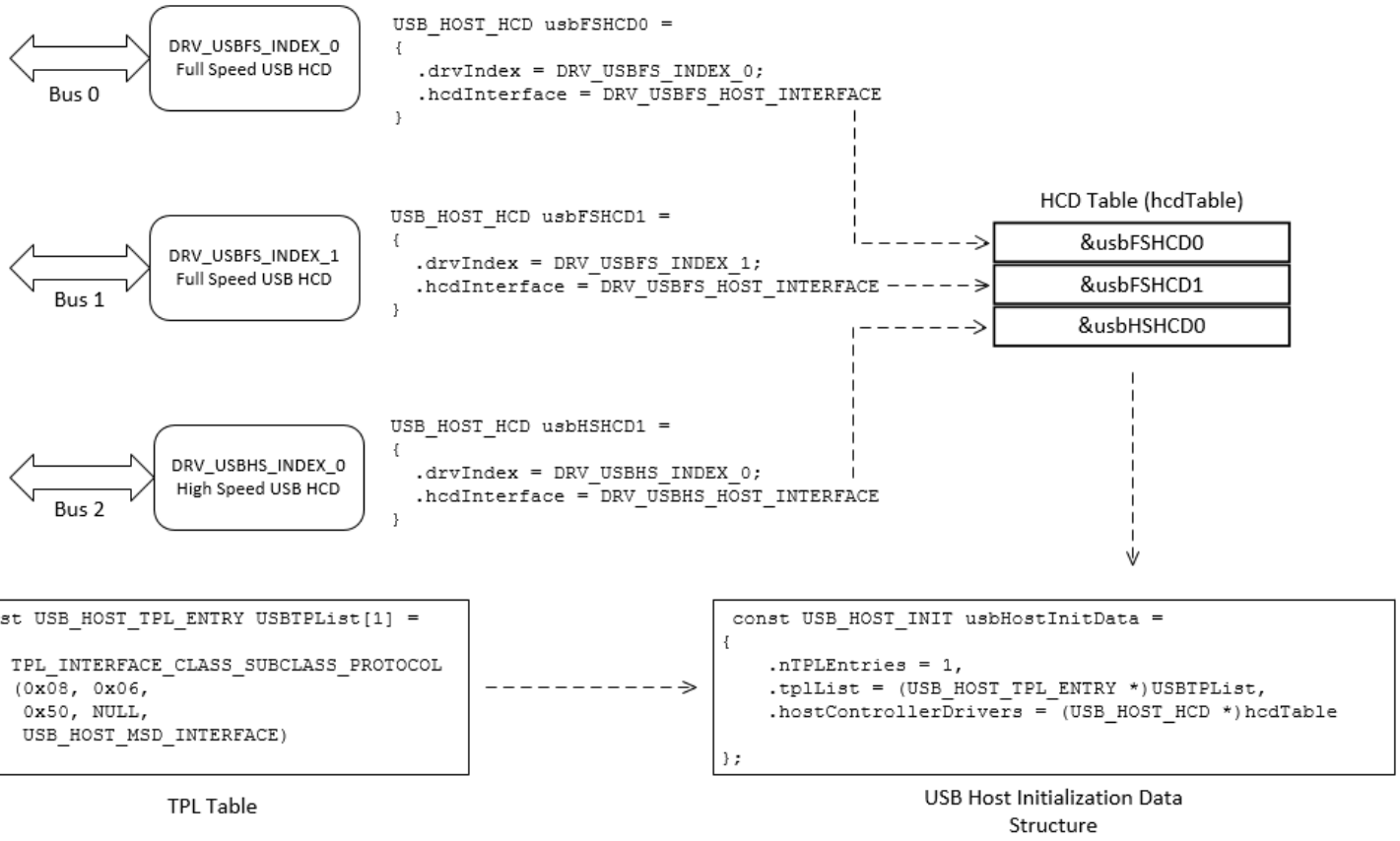
The following code shows an example TPL table.

Example: TPL Table

```
    /* This code shows some examples of configuring the USB Host Layer
     * TPL Table. In this example, the USB Host layer is configured to support
     * three different types of devices. */

    USB_HOST_TARGET_PERIPHERAL_LIST usbHostTPL[4] =
    {

        /* Catch every device with the exact Vendor ID = 0x04D9 and Product ID = 0x0001.
         * Every other device will not load this driver. */
        TPL_DEVICE_VID_PID( 0x04D9, 0x0001, &driverInitData, &DEVICE_DRIVER_EXAMPLE1_Driver ),

        /* This driver will catch any device with the Vendor ID of 0x04D9 and any
         * product ID = 0x0000 or 0x0002-0x00FF.  The entry in the TPL before this
         * caught the Product ID = 0x0001 case so that is why it is not caught by
         * this entry.  Those devices have already been caught. */
        TPL_DEVICE_VID_PID_MASKED( 0x04D9, 0x0002, 0xFF00, &driverInitData, &DEVICE_DRIVER_EXAMPLE2_Driver
```

```
),

        /* This entry will catch all other devices. */
        TPL_DEVICE_ANY( &driverInitData, &DEVICE_DRIVER_EXAMPLE3_Driver ),

        /* This entry will catch only a HID boot keyboard.  All other devices,
         * including other HID keyboards that are non-boot, will be skipped by this
         * entry. This driver will handle only this specific case. */
        TPL_INTERFACE_CLASS_SUBCLASS_PROTOCOL( USB_HID_CLASS_CODE, USB_HID_SUBCLASS_CODE_BOOT_INTERFACE,
                                               USB_HID_PROTOCOL_CODE_KEYBOARD, &hidDriverInitData,
                                               USB_HOST_HID_BOOT_KEYBOARD_DRIVER ),

        /* This entry will catch all CDC-ACM devices.  It filters on the class and
         * subclass but ignores the protocol since the driver will handle all
         * possible protocol options. */
        TPL_INTERFACE_CLASS_SUBCLASS( USB_CDC_CLASS_CODE, USB_CDC_SUBCLASS_CODE_ABSTRACT_CONTROL_MODEL,
                                      &cdcDriverInitData, USB_HOST_CDC_ACM_DRIVER ),

        /* This will catch all instances of the MSD class regardless subclass or
         * protocol.  In this case the driver will sort out if it supports the
         * device or not. */
        TPL_INTERFACE_CLASS( USB_MSD_CLASS_CODE, &msdDriverInitData, USB_HOST_MSD_DRIVER ),

        /* Any unclaimed interfaces can be sent to a particular driver if desired.
         * This can be used to create a similar mechanism that libUSB or WinUSB
         * provides on a PC where any unused interface can be opened and utilized by
         * these drivers. */
        TPL_INTERFACE_ANY( &driverInitData, USB_HOST_VENDOR_DRIVER )
    }
```

The Host Layer can now be initialized. The following code shows how the USB_HOST_HCD and the TPL table are specified in the USB_HOST_INIT (the Host Layer Initialization) data structure. In addition, the following figure illustrates the various initialization inputs needed by the Host Layer.



The USB_HOST_Initialize function is called to initialize the Host Layer. The initialization process may not complete when the USB_HOST_Initialization function exits. This will complete in subsequent calls to the USB_HOST_Tasks function.

Example: Specifying the TPL Table

```
/* This code shows an example of the USB Host Layer Initialization data
```

```
 * structure. In this case the number of TPL entries is one and there is only
 * one HCD (and hence only one USB bus) in the application */

const USB_HOST_TPL_ENTRY USBTPList[1] =
{
    /* This is the TPL */
    TPL_INTERFACE_CLASS_SUBCLASS_PROTOCOL(0x08, 0x06, 0x50, NULL,  USB_HOST_MSD_INTERFACE)

};

const USB_HOST_HCD hcdTable =
{
    /* The HCD table only contains one entry */
    .drvIndex = DRV_USBFS_INDEX_0,
    .hcdInterface = DRV_USBFS_HOST_INTERFACE
};

const USB_HOST_INIT usbHostInitData =
{
    /* This is the Host Layer Initialization data structure */
    .nTPLEntries = 1,
    .tplList = (USB_HOST_TPL_ENTRY *)USBTPList,
    .hostControllerDrivers = (USB_HOST_HCD *)&hcdTable

};
```

Host Layer - Application Interaction

This topic describes application interaction with the USB Host Layer.

Description

The Host Layer in the MPLAB Harmony USB Host stack provides the user application with API methods to operate the USB Host. The following sections discuss these API methods.

Registering the Event Handler

The application must register an event handler to receive device related USB Host events. The application sets the events handler by using the USB_HOST_EventHandlerSet function. An application defined context can also be provided. This context is returned along with the event handler and helps the application to identify the context in case of a dynamic application use cases. The host layer provides events when a connected device requires more current than can be provided or when a unsupported device was attached. The following code shows an example of registering the event handler.

```
/* This code shows an example of registering an event handler with the
 * Host Layer */

USB_HOST_EVENT_RESPONSE APP_USBHostEventHandler
(
    USB_HOST_EVENT event,
    void * eventData,
    uintptr_t context
)
{
    /* This is the event handler implementation */
    switch (event)
    {
        case USB_HOST_EVENT_DEVICE_UNSUPPORTED:
            break;
        case USB_HOST_EVENT_DEVICE_REJECTED_INSUFFICIENT_POWER:
            break;
        case USB_HOST_EVENT_HUB_TIER_LEVEL_EXCEEDED:
            break;
        case USB_HOST_EVENT_PORT_OVERCURRENT_DETECTED:
            break;
        default:
            break;
    }

    return(USB_HOST_EVENT_RESPONSE_NONE);
}
```

```
void APP_Tasks(void)
{
    /* This shows an example app state machine implementation in which the event
     * handler is set and the bus is then enabled. */

    switch(appData.state)
    {
        case APP_STATE_BUS_ENABLE:

            /* Set the event handler and enable the bus */
            USB_HOST_EventHandlerSet(APP_USBHostEventHandler, 0);
            USB_HOST_BusEnable(0);
            appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
            break;

        default:
            break;
    }
}
```

Enabling the Bus

The user application must call the USB_HOST_BusEnable function to enable the bus. This function enables the 5V VBUS supply to root hub port thus powering up the bus powered device that are attached to the bus. The attached devices will then indicate attach. The root hub will provide these attach events to the Host layer which in turn starts the enumeration process. The application can call other Host Layer functions only after the bus has been enabled. The USB_HOST_BusIsEnabled function must be called to check if the enable process has completed. The following code shows an example application state machine that enables the bus.

```
void APP_Tasks ( void )
{
   /* The application shows an example of how the USB bus is enabled and how the
    * application must wait for the bus to enabled */

    switch(appData.state)
    {
        case APP_STATE_BUS_ENABLE:

           /* Set the event handler and enable the bus */
            USB_HOST_EventHandlerSet(APP_USBHostEventHandler, 0);
            USB_HOST_BusEnable(0);
            appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
            break;

        case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:
            /* Check if the bus is enabled */
            if(USB_HOST_BusIsEnabled(0))
            {
                appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            }
            break;

        default:
            break;
    }
}
```

Attached Device Information

The application can use the USB_HOST_DeviceFirstGet and the USB_HOST_DeviceNextGet function to query for attached devices. The USB_HOST_DeviceFirstGet function will provide information on the first device that was attached to the bus. Information is returned in application specified USB_HOST_DEVICE_INFO object. The USB_HOST_DeviceFirstGet function will return the following information in the USB_HOST_DEVICE_INFO object:

- A Device Object Handle of the type USB_HOST_DEVICE_OBJ_HANDLE. The application can use this device object handle to perform operations on the device.
- The address of the device on the USB
- The bus to which this device belongs

The application can access the contents of the USB_HOST_DEVICE_INFO object but should not alter it contents. The same object is passed to the USB_HOST_DeviceNextGet function to get the information about the next device attached on the bus. Each call to this function defines the point at which the USB_HOST_DeviceNextGet function will start searching. If the device that is represented by the USB_HOST_DEVICE_INFO

object has been disconnected, calling the USB_HOST_DeviceNextGet function will return an error. The search must be reset by calling the USB_HOST_DeviceFirstGet function. The application can define multiple USB_HOST_DEVICE_INFO objects to search on different busses or maintain different search points.

```
void APP_Tasks(void)
{
    USB_HOST_DEVICE_INFO deviceInfo;
    USB_HOST_RESULT result;

    /* Get information about the first device on Bus 0 */
    result = USB_HOST_DeviceGetFirst(0, &deviceInfo);

    while(result != USB_HOST_RESULT_END_OF_DEVICE_LIST)
    {
        /* deviceInfo.address has the address of the bus */
        /* deviceInfo.deviceObjHandle will have the device object handle */

        /* Now we can get the information about the next device on the bus. */
        result = USB_HOST_DeviceGetNext(&deviceInfo);
    }
}
```

Suspend and Resume

The USB Host Layer allows the application to suspend and resume a device. The USB_HOST_DeviceSuspend and the USB_HOST_DeviceResume function are provided for this purpose. The application must use the device object handles, obtained from the USB_HOST_DeviceFirstGet or USB_HOST_DeviceNextGet function, to specify the device to suspend or resume when calling USB_HOST_DeviceSuspend and the USB_HOST_DeviceResume() function. The USB_HOST_DeviceIsSuspended function can be called to check the suspend status of the device.

In a case where the entire bus (and hence all device connected on the bus) need to be suspended or resumed, the application must call USB_HOST_BusSuspend and USB_HOST_BusResume functions to suspend or resume the entire bus. The USB_HOST_BusIsSuspended function can be called to check the suspend status of the bus.

Device String Descriptors

The application may want to obtain the string descriptors of a device. Sting descriptors are optionally provided by the USB device manufacturer and provide device information. The USB_HOST_DeviceStringDescriptorGet function is available to read the string descriptors. Calling this function will cause the Host Layer to invoke a control transfer request to read the string descriptor. The string descriptor will be available when the control transfer completes. The host layer calls the USB_HOST_STRING_REQUEST_COMPLETE_CALLBACK type callback function, that is provided in the USB_HOST_DeviceStringDescriptorGet function, when the control transfer has completed. The completion status of the request and the size of the string descriptor are available in the callback.

The function allows the application to obtain the supported string language IDs. The language ID of the string can be specified or a default can be used.

```
typedef struct
{
    /* This is an application specific data structure */
    char string[APP_STRING_SIZE];
    USB_HOST_REQUEST_HANDLE requestHandle;
    uintptr_t context;

} APP_DATA;

APP_DATA appData;

void APP_USBHostSringDescriptorGetCallBack
(
    USB_HOST_REQUEST_HANDLE requestHandle,
    size_t size,
    uintptr_t context
)
{
    /* This function is called when the string descriptor get function has
     * completed. */

    if(size != 0)
    {
        /* This means the function executed successfully and we have a string.
         * An application function prints the string to the console. */
        APP_PrintStringToConsole(appData.string, size);
    }
```

```
}

void APP_Tasks(void)
{
    USB_HOST_DEVICE_INFO deviceInfo;
    USB_HOST_RESULT result;

    /* Get information about the first device on Bus 0 */
    result = USB_HOST_DeviceGetFirst(0, &deviceInfo);

    if(result != USB_HOST_RESULT_END_OF_DEVICE_LIST)
    {
        /* deviceInfo.deviceObjHandle will have the device object handle. Use
         * this device object handle along with the
         * USB_HOST_DeviceStringDescriptorGet() function to read the product
         * string ID using the default Language ID. */

        USB_HOST_DeviceStringDescriptorGet(deviceInfo.deviceObjHandle, USB_HOST_DEVICE_STRING_PRODUCT,
                USB_HOST_DEVICE_STRING_LANG_ID_DEFAULT, appData.string, APP_STRING_SIZE,
                &appData.requestHandle, APP_USBHostSringDescriptorGetCallBack, appData.context );
    }
}
```

Event Handling

This topic describes event handling.

Description

The USB Host Layer provides general device related events to the application. The application must register an event handling function by using the USB_HOST_EventHandlerSet function. A context specified at the time of calling this function, is returned in the event handler. The event handler must be registered before the bus is enabled. Refer to the description of USB_HOST_DEVICE_EVENT events for details on the available events.

Configuring the Library

Describes how to configure the USB Host Layer.

Macros

	Name	Description
	USB_HOST_CONTROLLERS_NUMBER	Defines the number of USB Host Controllers that this Host Layer must manage.
	USB_HOST_DEVICE_INTERFACES_NUMBER	Defines the maximum number of interface that the attached device can contain in order for the USB Host Layer to process the device.
	USB_HOST_DEVICES_NUMBER	Defines the maximum number of devices to support.
	USB_HOST_HUB_SUPPORT_ENABLE	Defines if this USB Host application must support a Hub.
	USB_HOST_HUB_TIER_LEVEL	Defines the maximum tier of connected hubs to be supported.
	USB_HOST_PIPES_NUMBER	Defines the maximum number of pipes that the application will need.
	USB_HOST_TRANSFERS_NUMBER	Defines the maximum number of transfers that host layer should handle.

Description

The following configuration parameters must be defined while using the USB Host Layer. The configuration macros that implement these parameters must be located in the `system_config.h` file in the application project and a compiler include path (to point to the folder that contains this file) should be specified.

USB_HOST_CONTROLLERS_NUMBER Macro

Defines the number of USB Host Controllers that this Host Layer must manage.

File

usb_host_config_template.h

C

```
#define USB_HOST_CONTROLLERS_NUMBER
```

Description

USB Host Layer Controller Numbers

This constant defines the number of USB Host Controllers that this Host Layer must manage. The value of this constant should be atleast 1. Typical embedded applicatons contains only 1 USB host controller and hence only 1 USB. A microcontroller that features multiple USB modules can support multiple USB Host controllers and multipel USBs. USB contollers can also be interfaced to the microcontroller through common communication peripherals such as SPI.

This constant also defines the number of entries in the Host Controller Driver interface table, a pointer to which is passed in the hostContollerDrivers member of the USB_HOST_INIT data structure.

Remarks

None.

USB_HOST_DEVICE_INTERFACES_NUMBER Macro

Defines the maximum number of interface that the attached device can contain in order for the USB Host Layer to process the device.

File

usb_host_config_template.h

C

```
#define USB_HOST_DEVICE_INTERFACES_NUMBER
```

Description

USB Host Device Interface Numbers

This constant defines the maximum number of interface that the attached device can contain in order for the USB Host Layer to process the device. The device will be processed if it only contains less interfaces than the value of this constant.

Remarks

Supporting more interface per device required more processing time and data memory.

Example

An attached device contains a configuration that contains 10 interfaces, but the USB_HOST_DEVICE_INTERFACES_NUMBER is set to 5. The device will not be processed by the Host Layer. A dual CDC device needs to be supported. This device will have 4 interfaces. The USB_HOST_DEVICE_INTERFACES_NUMBER constant should be atleast 4.

USB_HOST_DEVICES_NUMBER Macro

Defines the maximum number of devices to support.

File

usb_host_config_template.h

C

```
#define USB_HOST_DEVICES_NUMBER
```

Description

USB Host Layer Devices Number

This configuration constant defines the maximum number of devices that this USB Host application must support. The value of this constant should be atleast 1. Multiple devices can be supported if Hub support is enabled. See USB_HOST_HUB_SUPPORT_ENABLE. The Hub itself will be treated as a device.

Remarks

Supporting multiple devices requires more data memory and processing time.

Example

If the USB Host application must support one USB Pen Drive and one USB Serial COM port (CDC Device), then this constant should be set to 3 (one additional device will be the Hub).

USB_HOST_HUB_SUPPORT_ENABLE Macro

Defines if this USB Host application must support a Hub.

File

usb_host_config_template.h

C

```
#define USB_HOST_HUB_SUPPORT_ENABLE
```

Description

USB Host Layer Hub Support

Specifying this macro will enable Hub Support. The HUB tier level to be supported is then specified by USB_HOST_HUB_TIER_LEVEL constant. If this macro is specified, the file usb_host_hub.c must be included in the application.

Remarks

Not specifying this macro will disable Hub support.

USB_HOST_HUB_TIER_LEVEL Macro

Defines the maximum tier of connected hubs to be supported.

File

usb_host_config_template.h

C

```
#define USB_HOST_HUB_TIER_LEVEL
```

Description

USB Host Hub Tier Level

This constant defines the maximum hub tiers to be supported by the USB Host application. This constant is considered only if the USB_HOST_HUB_SUPPORT_ENABLE option in specified. If specified, the value should be atleast 1. This means that one hub should be supported. In a case where another hub will be be connected to the hub which is connected to the USB Host, the value should be 2. As per the USB specification the maximum number of non-root hub tiers can be 5. Hence the value of this configuration constant should not exceed 5.

Remarks

None.

USB_HOST_PIPES_NUMBER Macro

Defines the maximum number of pipes that the application will need.

File

usb_host_config_template.h

C

```
#define USB_HOST_PIPES_NUMBER
```

Description

USB Host Layer Pipes Number

This configuration constant defines the maximum number of device communication pipes that Host Layer would need in the application. Every attached device requires atleast one pipe. This pipe is the control transfer pipe. Additional pipes are needed based on the type of device. For example, a standard Mass Storage Class device will need 2 pipes, a Communication Class Device will need 3 pipes, a HID device will need atleast 1 pipe. Vendor device will need communication pipe based on the device implementation. The number of pipes must also take into account the number of devices to support.

Remarks

This number should match the number of pipes configured in the HCD that this application will use, that is DRV_USBHS_HOST_PIPES_NUMBER or DRV_USBFS_HOST_PIPES_NUMBER.

Example

If a USB Host application must support 2 devices, either 2 USB pen driver or 2 CDC devices or a mix of both, then the pipes number should be set to

8. This is because the 2 CDC devices connected to the host will pose larger pipe

requirement. Two such devices will requires 2 control pipes, 2 interrupt pipes, 2 Bulk IN and 2 Bulk OUT pipes.

USB_HOST_TRANSFERS_NUMBER Macro

Defines the maximum number of transfers that host layer should handle.

File

usb_host_config_template.h

C

```
#define USB_HOST_TRANSFERS_NUMBER
```

Description

USB Host Layer Transfers Number

This constant defines the maximum number of transfers that the host layer should handle. The choice of this constant depends on the nature of devices that the USB Host application must support. Atleast two transfers are needed per pipe in the system. If the number of transfers provisioned in the system are insufficient, the USB Host will decline transfer request citing a busy status. This will affect the speed performance of the system.

Remarks

None.

Example

A USB Host application will support an MSD device and a CDC Device. The total number of pipes needed in the system are 7. The USB_HOST_TRANSFERS_NUMBER constant should be atleast 14 (2 per pipe). Specifying a larger number will enable more transfers to be queued but will also require more data memory.

Building the Library

Describes the files to be included in the project while using the USB Host Layer Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/usb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`usb_host.h`	This header file should be included in any `.c` file that accesses the USB Host Layer API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/usb_host.c`	This file implements the USB Host Layer interface and should be included in the project if USB Host mode operation is desired.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The USB Host Layer Library depends on the following modules:

- USB Driver Library (Host mode files)
- Timer System Service Library

Library Interface

a) Functions

	Name	Description
	USB_HOST_Deinitialize	Deinitializes the specified instance of the USB Host Layer.
	USB_HOST_Status	Gets the current status of the USB Host Layer.
	USB_HOST_Tasks	Maintains the USB Host Layer state machine.
	USB_HOST_BusEnable	Starts host operations.
	USB_HOST_BusIsEnabled	Checks if the bus is enabled.
	USB_HOST_BusIsSuspended	Returns the suspend status of the bus.
	USB_HOST_BusResume	Resumes the bus.
	USB_HOST_BusSuspend	Suspends the bus.
	USB_HOST_DeviceGetFirst	Returns information about the first attached device on the bus.
	USB_HOST_DeviceGetNext	Returns information about the next device on the bus.
	USB_HOST_DeviceIsSuspended	Returns the suspend state of the device is suspended.
	USB_HOST_DeviceResume	Resumes the selected device
	USB_HOST_DeviceSpeedGet	Returns the speed at which this device is operating.
	USB_HOST_DeviceStringDescriptorGet	Retrieves specified string descriptor from the device
	USB_HOST_DeviceSuspend	Suspends the specified device.
	USB_HOST_EventHandlerSet	USB Host Layer Event Handler Callback Function set function.
	USB_HOST_Initialize	Initializes the USB Host layer instance specified by the index.

b) Data Types and Constants

	Name	Description
	USB_HOST_INIT	Defines the data required to initialize a USB Host Layer instance.
	USB_HOST_EVENT_RESPONSE	Host Layer Events Handler Function Response Type.
	\ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL)	USB Host Layer TPL Table Entry Matching Criteria flag
	\ .tplFlags.driverType = (TPL_FLAG_VID_PID)	USB Host Layer TPL Table Entry Matching Criteria flag
	0	USB Host Layer TPL Table Entry Matching Criteria flag
	0x0000	USB Host Layer TPL Table Entry Matching Criteria flag
	0xFF	USB Host Layer TPL Table Entry Matching Criteria flag
	0xFF }	USB Host Layer TPL Table Entry Matching Criteria flag
	0xFFFF	USB Host Layer TPL Table Entry Matching Criteria flag
	0xFFFF }	USB Host Layer TPL Table Entry Matching Criteria flag
	1	USB Host Layer TPL Table Entry Matching Criteria flag
	classCode	USB Host Layer TPL Table Entry Matching Criteria flag
	false	USB Host Layer TPL Table Entry Matching Criteria flag
	initData	USB Host Layer TPL Table Entry Matching Criteria flag
	mask	USB Host Layer TPL Table Entry Matching Criteria flag
	pid	USB Host Layer TPL Table Entry Matching Criteria flag
	pid }	USB Host Layer TPL Table Entry Matching Criteria flag
	subClassCode	USB Host Layer TPL Table Entry Matching Criteria flag
	true	USB Host Layer TPL Table Entry Matching Criteria flag
	USB_HOST_BUS	Defines a USB Bus Data Type.
	USB_HOST_DEVICE_INFO	Defines the data type that is used by the USB_HOST_DeviceGetFirst() and USB_HOST_DeviceGetNext() functions.
	USB_HOST_DEVICE_OBJ_HANDLE	Handle to an attached USB Device.
	USB_HOST_DEVICE_STRING	Defines a defines types of strings that can be request through the USB_HOST_DeviceStringDescriptorGet() function.
	USB_HOST_EVENT	Defines the different events that the USB Host Layer can generate.
	USB_HOST_EVENT_HANDLER	USB Host Layer Event Handler Function Pointer Type
	USB_HOST_HCD	Defines the USB Host HCD Information object that is provided to the host layer.

	USB_HOST_REQUEST_HANDLE	USB Host Request Handle Type
	USB_HOST_RESULT	USB Host Results.
	USB_HOST_STRING_REQUEST_COMPLETE_CALLBACK	USB Host Device String Descriptor Request Complete Callback Function Type
	USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY	USB Host Layer TPL Table Entry Matching Criteria flag
	USB_HOST_TPL_ENTRY	USB Host Layer TPL Table Entry Matching Criteria flag
	vid	USB Host Layer TPL Table Entry Matching Criteria flag
	USB_HOST_BUS_ALL	USB Host Bus All
	USB_HOST_DEVICE_OBJ_HANDLE_INVALID	Defines an invalid USB Device Object Handle.
	USB_HOST_DEVICE_STRING_LANG_ID_DEFAULT	Defines the default Lang ID to be used while obtaining the string.
	USB_HOST_REQUEST_HANDLE_INVALID	USB Host Request Invalid Handle
	USB_HOST_RESULT_MIN	USB Host Result Minimum Constant.

Description

This section describes the Application Programming Interface (API) functions of the USB Host Layer Library.

Refer to each section for a detailed description.

a) Functions

USB_HOST_Deinitialize Function

Deinitializes the specified instance of the USB Host Layer.

File

usb_host.h

C

```
void USB_HOST_Deinitialize(SYS_MODULE_OBJ hostLayerObject);
```

Returns

None.

Description

Deinitializes the USB Host Layer. All internal data structures will be reset.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again. This routine will NEVER block waiting for hardware.

Preconditions

Function USB_HOST_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from USB_HOST_Initialize
SYS_STATUS          status;

USB_HOST_Deinitialize(object);

status = USB_HOST_Status(object);
if (SYS_MODULE_DEINITIALIZED != status)
{
    // Can check again later if you need to know
    // when the driver is deinitialized.
}
```

Parameters

Parameters	Description
object	USB Host layer object handle, returned from the USB_HOST_Initialize routine

Function

void USB_HOST_Deinitialize(SYS_MODULE_OBJ object)

USB_HOST_Status Function

Gets the current status of the USB Host Layer.

File

usb_host.h

C

```
SYS_STATUS USB_HOST_Status(SYS_MODULE_OBJ hostLayerObject);
```

Returns

SYS_STATUS_READY - Indicates that the USB Host layer is ready for operations.

SYS_STATUS_BUSY - The initialization is in progress.

SYS_STATUS_DEINITIALIZED - Indicates that the driver has been deinitialized

Description

This routine provides the current status of the USB Host Layer.

Remarks

This function is typically called by the MPLAB Harmony System to check the system status of the USB Host Layer. This function is not intended to be called directly by the application tasks routine.

Preconditions

Function USB_HOST_Initialize should have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from USB_HOST_Initialize
SYS_STATUS          status;

status = USB_HOST_Status(object);
if (SYS_STATUS_READY == status)
{
    // The USB Host system is ready and is running.
}
```

Parameters

Parameters	Description
object	USB Host Layer object handle, returned from the USB_HOST_Initialize routine

Function

SYS_STATUS USB_HOST_Status(SYS_MODULE_OBJ object)

USB_HOST_Tasks Function

Maintains the USB Host Layer state machine.

File

usb_host.h

C

```
void USB_HOST_Tasks(SYS_MODULE_OBJ hostLayerObject);
```

Returns

None.

Description

This routine maintains the USB Host layer's state machine. It must be called frequently to ensure proper operation of the USB. This function should be called from the SYS_Tasks function.

Remarks

This routine is not intended to be called directly by an application. It is called by the MPLAB Harmony System Tasks function.

Preconditions

The USB_HOST_Initialize routine must have been called for the specified USB Host Layer instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from USB_HOST_Initialize

void SYS_Tasks(void)
{
    USB_HOST_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified driver instance (returned from USB_HOST_Initialize)

Function

void USB_HOST_Tasks (SYS_MODULE_OBJ object);

USB_HOST_BusEnable Function

Starts host operations.

File

usb_host.h

C

```
USB_HOST_RESULT USB_HOST_BusEnable(USB_HOST_BUS bus);
```

Returns

USB_HOST_RESULT_SUCCESS if the request was accepted. USB_HOST_RESULT_BUS_UNKNOWN if the specified bus is invalid (it does not exist in the system). USB_HOST_RESULT_FAILURE if an unknown failure occurred.

Description

The function starts the operation of the USB Host Bus. It enables the root hub associated with specified bus and starts the process of detecting attached devices and enumerating them. The USB_HOST_EventHandlerSet() function should have been called to register an application host layer event handler before the bus is enabled (before the USB_HOST_BusEnable() function is called). This will ensure that the application does not miss any events.

Remarks

The Host Layer may generate events after the USB_HOST_BusEnable() function is called. The application should have registered an event handler using the USB_HOST_EventHandlerSet() function to handle these events. The USB_HOST_EventHandlerSet() function should have been called before the USB_HOST_BusEnable() function is called.

Preconditions

The USB_HOST_Initialize() function should have been called before calling this function.

Example

```
TBD.
```

Parameters

Parameters	Description
bus	the bus to be enabled. If this is set to USB_HOST_BUS_ALL, all buses will be enabled.

Function

USB_HOST_RESULT USB_HOST_BusEnable(USB_HOST_BUS bus)

USB_HOST_BusIsEnabled Function

Checks if the bus is enabled.

File

usb_host.h

C

```
USB_HOST_RESULT USB_HOST_BusIsEnabled(USB_HOST_BUS bus);
```

Returns

USB_HOST_RESULT_TRUE if the bus is enabled. USB_HOST_RESULT_FALSE if the bus is not enabled..
USB_HOST_RESULT_BUS_UNKNOWN if the specified bus is invalid (it does not exist in the system). USB_HOST_RESULT_FAILURE if an unknown failure occurred.

Description

The function returns the enable status of the bus. It can be called after the USB_HOST_BusEnable() function is called, to check if the bus has been enabled yet. If the bus parameter is set to USB_HOST_BUS_ALL, then the function will check the enable status of all the busses and will return true only if all the busses are enabled.

Remarks

None.

Preconditions

The USB_HOST_Initialize() function should have been called before calling this function.

Example

TBD.

Parameters

Parameters	Description
bus	the bus that needs to checked for enable status. If this is set to USB_HOST_BUS_ALL, all buses will be checked.

Function

USB_HOST_RESULT USB_HOST_BusIsEnabled(USB_HOST_BUS bus)

USB_HOST_BusIsSuspended Function

Returns the suspend status of the bus.

File

usb_host.h

C

```
USB_HOST_RESULT USB_HOST_BusIsSuspended(USB_HOST_BUS bus);
```

Returns

USB_HOST_RESULT_TRUE - if the bus is suspended. USB_HOST_RESULT_FALSE - if the bus is not suspended.
USB_HOST_RESULT_BUS_NOT_ENABLED - if the bus was not enabled. USB_HOST_RESULT_BUS_UNKNOWN - if the specified bus does not exist in the system. USB_HOST_RESULT_FAILURE - an unknown error occurred.

Description

This function returns suspend status of the specified USB bus. This function can be used to check the completion of the Suspend operation started by using the USB_HOST_BusSuspend() function. The function would return USB_HOST_RESULT_FALSE if the bus is not suspended. Calling the USB_HOST_BusIsSuspended() with bus specified as USB_HOST_BUS_ALL returns the suspend status of the all USB segments that are managed by the host layer. The function would return USB_HOST_RESULT_TRUE only if all the bus are in a suspended state.

Remarks

None.

Preconditions

The USB_HOST_BusEnable() function should have been called to enable the bus.

Example

TBD.

Parameters

Parameters	Description
bus	the bus whose suspend status is to be queried.

Function

USB_HOST_RESULT USB_HOST_BusIsSuspended (USB_HOST_BUS bus)

USB_HOST_BusResume Function

Resumes the bus.

File

usb_host.h

C

```
USB_HOST_RESULT USB_HOST_BusResume(USB_HOST_BUS bus);
```

Returns

USB_HOST_RESULT_SUCCESS - if the request was successful or if the bus was already resumed. USB_HOST_RESULT_BUS_UNKNOWN - the request failed because the bus does not exist in the system. USB_HOST_RESULT_BUS_NOT_ENABLED - the bus was not enabled. USB_HOST_RESULT_FAILURE - An unknown error occurred.

Description

The function resumes the bus. All devices on the bus will be receive resume signaling. If bus is specified as USB_HOST_BUS_ALL, all the buses managed by this host will be resumed.

Remarks

None.

Preconditions

The USB_HOST_BusEnable() function should have been called to enable the bus.

Example

```
// Resume bus 0
USB_HOST_BusResume(0);

// Resume all buses
USB_HOST_BusSuspend(USB_HOST_BUS_ALL);
```

Parameters

Parameters	Description
bus	The bus to be resume or USB_HOST_BUS_ALL to resume all buses.

Function

USB_HOST_RESULT USB_HOST_BusResume (USB_HOST_BUS bus);

USB_HOST_BusSuspend Function

Suspends the bus.

File

usb_host.h

C

```
USB_HOST_RESULT USB_HOST_BusSuspend(USB_HOST_BUS bus);
```

Returns

USB_HOST_RESULT_SUCCESS - if the request was successful. USB_HOST_RESULT_BUS_NOT_ENABLED - if the bus was not enabled. USB_HOST_RESULT_FAILURE - An unknown error has occurred. USB_HOST_RESULT_BUS_UNKNOWN - if the specified bus does not exist in the system.

Description

The function suspends the bus. All devices on the bus will be suspended. If bus is specified as USB_HOST_BUS_ALL, all the buses managed by this host will be suspended.

Remarks

None.

Preconditions

The USB_HOST_BusEnable() function should have been called to enable the bus.

Example

```
// Suspend the bus 0
USB_HOST_BusSuspend(0);

// Suspend all buses
USB_HOST_BusSuspend(USB_HOST_BUS_ALL);
```

Parameters

Parameters	Description
bus	The bus to be suspended or USB_HOST_BUS_ALL to suspend all buses.

Function

USB_HOST_RESULT USB_HOST_BusSuspend (USB_HOST_BUS bus);

USB_HOST_DeviceGetFirst Function

Returns information about the first attached device on the bus.

File

usb_host.h

C

```
USB_HOST_RESULT USB_HOST_DeviceGetFirst(USB_HOST_BUS bus, USB_HOST_DEVICE_INFO * deviceInfo);
```

Returns

USB_HOST_RESULT_SUCCESS - The function executed successfully. USB_HOST_RESULT_END_OF_DEVICE_LIST - There are no attached devices on the bus. USB_HOST_RESULT_BUS_UNKNOWN - The specified bus does not exist in the system.
USB_HOST_RESULT_BUS_NOT_ENABLED - The specified bus is not enabled. USB_HOST_RESULT_PARAMETER_INVALID - the deviceInfo parameter is NULL. USB_HOST_RESULT_FAILURE - an unknown failure occurred.

Description

This function returns information about the first attached device on the specified bus. The USB_HOST_DeviceGetNext() function can be used to get the reference to the next attached device on the bus. The USB_HOST_DEVICE_INFO object is provided by the application.The device information will be populated into this object. If there are no devices attached on the bus, the function will set the deviceObjHandle parameter, in the USB_HOST_DEVICE_INFO object, to USB_HOST_DEVICE_OBJ_HANDLE INVALID.

Remarks

None.

Preconditions

The USB_HOST_BusEnable function should have been called to enable detection of attached devices.

Example

TBD.

Parameters

Parameters	Description
bus	the bus to be queried for attached devices.
deviceInfo	output parameter. Will contain device information when the function returns. If the deviceObjHandle member of the structure contains USB_HOST_DEVICE_OBJ_HANDLE INVALID, then there are no attached devices on the bus and the deviceAddress and the bus member of the info object will contain indeterminate values.

Function

```
USB_HOST_RESULT USB_HOST_DeviceGetFirst
(
    USB_HOST_BUS bus,
    USB_HOST_DEVICE_INFO * deviceInfo
);
```

USB_HOST_DeviceGetNext Function

Returns information about the next device on the bus.

File

usb_host.h

C

```
USB_HOST_RESULT USB_HOST_DeviceGetNext(USB_HOST_DEVICE_INFO * deviceInfo);
```

Returns

USB_HOST_RESULT_SUCCESS - The function executed successfully. USB_HOST_RESULT_END_OF_DEVICE_LIST - There are no attached devices on the bus. USB_HOST_RESULT_PARAMETER_INVALID - the deviceInfo parameter is NULL.
USB_HOST_RESULT_DEVICE_UNKNOWN - the device specified in deviceInfo does not exist in the system. The search should be restarted.
USB_HOST_RESULT_FAILURE - an unknown failure occurred. Application can restart the search by calling the USB_HOST_DeviceGetFirst() function.

Description

This function returns information of the next device attached on the bus. The USB_HOST_DeviceGetFirst() function should have been called at least once on the deviceInfo object. Then calling this function repeatedly on the deviceInfo object will return information about the next attached device on the bus. When there are no more attached devices to report, the function returns USB_HOST_RESULT_END_OF_DEVICE_LIST.

Calling the USB_HOST_DeviceGetFirst() function on the deviceInfo object after the USB_HOST_DeviceGetNext() function has been called will cause the host to reset the deviceInfo object to point to the first attached device.

Remarks

None.

Preconditions

The USB_HOST_DeviceGetFirst() function must have been called before calling this function.

Example

```
TBD.
```

Parameters

Parameters	Description
deviceInfo	pointer to the USB_HOST_DEVICE_INFO object.

Function

```
USB_HOST_RESULT USB_HOST_DeviceGetNext (USB_HOST_DEVICE_INFO * deviceInfo);
```

USB_HOST_DeviceIsSuspended Function

Returns the suspend state of the device is suspended.

File

usb_host.h

C

```
USB_HOST_RESULT USB_HOST_DeviceIsSuspended(USB_HOST_DEVICE_OBJ_HANDLE deviceObjHandle);
```

Returns

USB_HOST_RESULT_TRUE - if the device is suspended. USB_HOST_RESULT_FALSE - if the device is not suspended. USB_HOST_RESULT_DEVICE_UNKNOWN - the specified device does not exist in the system. USB_HOST_RESULT_FAILURE - An unknown failure occurred.

Description

This function returns the suspend state of the specified USB device. This function can be used to check the completion of the Resume operation started by using the USB_HOST_Resume() function. If the Resume signaling has completed, the USB_HOST_IsSuspended() function would return USB_HOST_RESULT_TRUE.

Remarks

None.

Preconditions

The USB_HOST_BusEnable() function should have been called.

Example

TBD.

Parameters

Parameters	Description
deviceObjHandle	handle to the device that needs to be checked for suspend status.

Function

```
USB_HOST_RESULT USB_HOST_DeviceIsSuspended
(
    USB_HOST_DEVICE_OBJ_HANDLE deviceObjHandle
);
```

USB_HOST_DeviceResume Function

Resumes the selected device

File

usb_host.h

C

```
USB_HOST_RESULT USB_HOST_DeviceResume(USB_HOST_DEVICE_OBJ_HANDLE deviceObjHandle);
```

Returns

USB_HOST_RESULT_SUCCESS - The request was accepted and the device will be resumed or the device was already resumed. USB_HOST_RESULT_DEVICE_UNKNOWN - The request failed. The device may have been detached. USB_HOST_RESULT_FAILURE - An unknown failure occurred.

Description

The function resumes the selected device. A device can be resumed only if it was suspended.

Remarks

None.

Preconditions

None.

Example

TBD.

Parameters

Parameters	Description
deviceObjHandle	handle to the device to be resumed.

Function

```
USB_HOST_RESULT USB_HOST_DeviceResume
(
    USB_HOST_DEVICE_OBJ_HANDLE deviceObjHandle
);
```

USB_HOST_DeviceSpeedGet Function

Returns the speed at which this device is operating.

File

usb_host.h

C

```
USB_HOST_RESULT USB_HOST_DeviceSpeedGet(USB_HOST_DEVICE_OBJ_HANDLE deviceHandle, USB_SPEED * speed);
```

Returns

USB_HOST_RESULT_SUCCESS - The function was successful. speed will contain the speed of the device.
USB_HOST_RESULT_DEVICE_UNKNOWN - The device does not exist in the system. speed will contain USB_SPEED_ERROR.
USB_HOST_RESULT_FAILURE - an unknown error occurred.

Description

This function returns the speed at which this device is operating.

Remarks

None.

Preconditions

The USB_HOST_Initialize() function should have been called.

Example

Parameters

Parameters	Description
deviceObjHandle	handle to the device whose speed is required.
speed	output parameter. Will contain the speed of the device if the function was successful.

Function

```
USB_HOST_RESULT USB_HOST_DeviceSpeedGet
(
    USB_HOST_DEVICE_OBJ_HANDLE deviceObjHandle,
    USB_SPEED * speed
)
```

USB_HOST_DeviceStringDescriptorGet Function

Retrieves specified string descriptor from the device

File

usb_host.h

C

```
USB_HOST_RESULT USB_HOST_DeviceStringDescriptorGet(USB_HOST_DEVICE_OBJ_HANDLE deviceObjHandle,
USB_HOST_DEVICE_STRING stringType, uint16_t languageID, void * stringDescriptor, size_t length,
USB_HOST_REQUEST_HANDLE * requestHandle, USB_HOST_STRING_REQUEST_COMPLETE_CALLBACK callback, uintptr_t
context);
```

Returns

USB_HOST_RESULT_SUCCESS - The request was scheduled successfully. requestHandle will contain a valid request handle. USB_HOST_RESULT_DEVICE_UNKNOWN - The request failed because the device was detached. USB_HOST_RESULT_FAILURE - An unknown error occurred. USB_HOST_RESULT_REQUEST_BUSY - The host layer cannot take more requests at this point. The application should try later. USB_HOST_RESULT_STRING_DESCRIPTOR_UNSUPPORTED - The device does not support the specified string descriptor type.

Description

This function retrieves the specified string descriptor from the device. This function will cause the host layer to issue a control transfer to the device. When the string descriptor is available, the host layer will call the callback function to let the application know that the request has completed.

The function will return a valid request handle in requestHandle, if the request was successful. This request handle will be returned in the callback function. The size of the stringDescriptor buffer is specified by the length parameter. Only length number of bytes will be retrieved. The type of device string descriptor to be retrieved is specified by the stringType parameter. The supported language IDs, manufacturer, product and serial number strings can be obtained. While obtaining the supported language IDs, the languageID parameter will be ignored.

Remarks

None.

Preconditions

The USB_HOST_BusEnable() function should have been called.

Example

Parameters

Parameters	Description
deviceObjHandle	handle to the device whose string descriptor is to be retrieved.
stringType	type of string descriptor to be retrieved
languageID	the language ID of the string descriptor
stringDescriptor	output buffer for the descriptor
length	size of the specified output buffer
requestHandle	This is an output parameter. It will contain a valid request handle if the request was successful. It will contain USB_HOST_REQUEST_HANDLE_INVALID is the request was not successful.
callback	Function that will be called when this request completes. If this is NULL, then the application will not receive indication of completion.
context	Calling application context to be returned in the callback function.

Function

```
  USB_HOST_RESULT USB_HOST_DeviceStringDescriptorGet
(
    USB_HOST_DEVICE_OBJ_HANDLE deviceObjHandle,
    USB_HOST_DEVICE_STRING stringType,
uint16_t languageID,
void * stringDescriptor,
size_t length,
    USB_HOST_REQUEST_HANDLE * requestHandle,
    USB_HOST_STRING_REQUEST_COMPLETE_CALLBACK callback,
uintptr_t context
);
```

USB_HOST_DeviceSuspend Function

Suspends the specified device.

File

usb_host.h

C

```
USB_HOST_RESULT USB_HOST_DeviceSuspend(USB_HOST_DEVICE_OBJ_HANDLE deviceObjHandle);
```

Returns

USB_HOST_RESULT_SUCCESS - The request was accepted and the device will be suspended. USB_HOST_RESULT_DEVICE_UNKNOWN - The request failed. The device may have been detached. USB_HOST_RESULT_FAILURE - An unknown failure occurred.

Description

The function suspends the specified device.

Remarks

None.

Preconditions

The USB_HOST_BusEnable() function should have been called.

Example

TBD.

Parameters

Parameters	Description
deviceObjHandle	handle to the device to suspend.

Function

```
USB_HOST_RESULT USB_HOST_DeviceSuspend
(
    USB_HOST_DEVICE_OBJ_HANDLE deviceObjHandle
);
```

USB_HOST_EventHandlerSet Function

USB Host Layer Event Handler Callback Function set function.

File

usb_host.h

C

```
USB_HOST_RESULT USB_HOST_EventHandlerSet(USB_HOST_EVENT_HANDLER eventHandler, uintptr_t context);
```

Returns

USB_HOST_RESULT_SUCCESS - The function was successful. USB_HOST_RESULT_FAILURE - An unknown failure occurred.

Description

This is the USB Host Layer Event Handler Callback Set function. An application can receive USB Host Layer events by using this function to register and event handler callback function. The application can additionally specify a specific context which will returned with the event handler callback function. The event handler must be set (this function must be called) before any of the USB buses are enabled.

Remarks

None.

Preconditions

The host layer should have been initialized.

Example

TBD.

Parameters

Parameters	Description
eventHandler	Pointer to the call back function. The host layer notifies the application about host layer events by calling this function. If this is NULL, then events will not be generated.
context	application specific context.

Function

```
USB_HOST_RESULT USB_HOST_EventHandlerSet
```

(
USB_HOST_EVENT_HANDLER * eventHandler,
uintptr_t context
)

USB_HOST_Initialize Function

Initializes the USB Host layer instance specified by the index.

File

usb_host.h

C

```
SYS_MODULE_OBJ USB_HOST_Initialize(const SYS_MODULE_INIT * init);
```

Returns

Return a SYS_MODULE_OBJ_INVALID if the initialization failed.

Description

This routine initializes the USB Host Layer. This function must be called before any other Host layer function can be called. The initialization data is specified by the init parameter. This function is typically called in the SYS_Initialize() function. The initialization completion may require the USB_HOST_Tasks() routine to execute. The initialization function does not start the operation of the Host on the USB. This must be done explicitly via the USB_HOST_BusEnable() function. This function will initialize all client drivers listed in the TPL.

Remarks

This routine must be called before any other USB Host routine is called. This routine should only be called once during system initialization unless USB_HOST_Deinitialize is called to deinitialize the Host Layer instance. This routine will NEVER block for hardware access. The USB_HOST_Tasks() function should be called to complete the initialization.

Preconditions

The USB Host Controller driver initialization should be called somewhere in the SYS_Initialize() function.

Example

TBD.

Parameters

Parameters	Description
init	Pointer to a USB_HOST_INIT data structure containing data necessary to initialize the driver.

Function

SYS_MODULE_OBJ USB_HOST_Initialize
(
const SYS_MODULE_INIT * const init
)

b) Data Types and Constants

USB_HOST_INIT Structure

Defines the data required to initialize a USB Host Layer instance.

File

usb_host.h

C

```
typedef struct {
  size_t nTPLEntries;
  USB_HOST_TPL_ENTRY * tplList;
  USB_HOST_HCD * hostControllerDrivers;
} USB_HOST_INIT;
```

Members

Members	Description
size_t nTPLEntries;	Size of the TPL table
USB_HOST_TPL_ENTRY * tplList;	Pointer to the TPL table for this host layer implementation.
USB_HOST_HCD * hostControllerDrivers;	This is a pointer to a table of host controller drivers that the host layer will operate on. The number of entries in this table is specified via the USB_HOST_CONTROLLERS_NUMBER configuration macro in system_config.h

Description

USB Host Initialization Data Structure

This data type defines the data required to initialize the host layer. A pointer to a structure of this type is required by the USB_HOST_Initialize() function.

Remarks

This data structure is specific to the PIC32MX implementation of the USB Host layer.

USB_HOST_EVENT_RESPONSE Enumeration

Host Layer Events Handler Function Response Type.

File

usb_host.h

C

```
typedef enum {
  USB_HOST_EVENT_RESPONSE_NONE = 0
} USB_HOST_EVENT_RESPONSE;
```

Members

Members	Description
USB_HOST_EVENT_RESPONSE_NONE = 0	Returning this value indicates no application response to the host event

Description

Host Layer Events Handler Function Response Type.

This is the definition of the Host Layer Event Handler Response Type.

Remarks

None.

\ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL) Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
```

```
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

\ .tplFlags.driverType = (TPL_FLAG_VID_PID) Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

0 Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

0x0000 Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

0xFF Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
```

```
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

0xFF } Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
```

```
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

0xFFFF Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

0xFFFF } Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

1 Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
```

```
    subClassCode,
    protocolCode
  } driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
  .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
  .hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
  .hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
  {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
  .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
  .hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
  initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
  .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
  .tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
  \ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
  .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
  .tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
  driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
  vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
  .tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
  {\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
  .id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
  .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
  .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
  .tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
  .hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
  value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
  uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
  ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
  ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
  USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

classCode Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
```

```
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

false Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will

cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

initData Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

mask Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

pid Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
```

```
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

pid } Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

subClassCode Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

true Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

USB_HOST_BUS Type

Defines a USB Bus Data Type.

File

usb_host.h

C

```
typedef uint8_t USB_HOST_BUS;
```

Description

USB Bus Data Type

This data type defines a USB Bus. In microcontroller devices, that may have multiple USB peripherals, this type identifies the USB bus associated with each peripheral. Bus numbers start from 0 and counts up to include all the busses in the system. The total number of busses and the mapping

between a bus and the USB controller is specified in the Host Layer initialization data structure.

Remarks

None.

USB_HOST_DEVICE_INFO Structure

Defines the data type that is used by the USB_HOST_DeviceGetFirst() and USB_HOST_DeviceGetNext() functions.

File

usb_host.h

C

```
typedef struct {
  USB_HOST_DEVICE_OBJ_HANDLE deviceObjHandle;
  uint8_t deviceAddress;
  USB_HOST_BUS bus;
} USB_HOST_DEVICE_INFO;
```

Members

Members	Description
USB_HOST_DEVICE_OBJ_HANDLE deviceObjHandle;	USB Host Device Object Handle
uint8_t deviceAddress;	Address of the device on the USB
USB_HOST_BUS bus;	The bus to which this device is connected

Description

USB Host Device Info Type

This data type defines the type of data that is used by the USB_HOST_DeviceGetFirst() and USB_HOST_DeviceGetNext() functions. The application must provide an object of this type to these functions to obtain information about the devices attached on the USB.

Remarks

The application must only instantiate this data structure and should not modify it's contents. Multiple objects can be instantiated and used.

USB_HOST_DEVICE_OBJ_HANDLE Type

Handle to an attached USB Device.

File

usb_host.h

C

```
typedef uintptr_t USB_HOST_DEVICE_OBJ_HANDLE;
```

Description

USB Host Device Object Handle

This data type defines the type of handle to an attached USB Device. This handle uniquely identifies the attached device. A handle of this type is returned in the deviceObjHandle member of the USB_HOST_DEVICE_INFO structure when the USB_HOST_DeviceGetFirst() and the USB_HOST_DeviceGetNext() functions are called.

Remarks

None.

USB_HOST_DEVICE_STRING Enumeration

Defines a defines types of strings that can be request through the USB_HOST_DeviceStringDescriptorGet() function.

File

usb_host.h

C

```
typedef enum {
  USB_HOST_DEVICE_STRING_LANG_ID = 0,
```

```
  USB_HOST_DEVICE_STRING_MANUFACTURER,
  USB_HOST_DEVICE_STRING_PRODUCT,
  USB_HOST_DEVICE_STRING_SERIAL_NUMBER
} USB_HOST_DEVICE_STRING;
```

Members

Members	Description
USB_HOST_DEVICE_STRING_LANG_ID = 0	Specifies the language ID string
USB_HOST_DEVICE_STRING_MANUFACTURER	Specifies the manufacturer string
USB_HOST_DEVICE_STRING_PRODUCT	Specifies the product string
USB_HOST_DEVICE_STRING_SERIAL_NUMBER	Specifies the serial number string

Description

USB Host Device String Type

This type defines the types of strings that can be request through the USB_HOST_DeviceStringDescriptorGet() function. The stringType parameter in the function call can be set any one of these types.

Remarks

None.

USB_HOST_EVENT Enumeration

Defines the different events that the USB Host Layer can generate.

File

usb_host.h

C

```
typedef enum {
  USB_HOST_EVENT_DEVICE_REJECTED_INSUFFICIENT_POWER,
  USB_HOST_EVENT_DEVICE_UNSUPPORTED,
  USB_HOST_EVENT_HUB_TIER_LEVEL_EXCEEDED,
  USB_HOST_EVENT_PORT_OVERCURRENT_DETECTED
} USB_HOST_EVENT;
```

Members

Members	Description
USB_HOST_EVENT_DEVICE_REJECTED_INSUFFICIENT_POWER	This event occurs when device needs more current than what the host can supply.
USB_HOST_EVENT_DEVICE_UNSUPPORTED	This event occurs when a host layer could not attach any drivers to the attached device or when an error has occurred. There is no event data associated with this event.
USB_HOST_EVENT_HUB_TIER_LEVEL_EXCEEDED	This event occurs when the number of hubs connected to the host exceeds the configured maximum number of hubs USB_HOST_HUB_TIER_LEVEL. There is no event data associated with this event.
USB_HOST_EVENT_PORT_OVERCURRENT_DETECTED	This event occurs when an over-current condition is detected at the root • hub or an external hub port.

Description

USB Host Events

This data type defines the different events that USB Host Layer can generate. The application is intended recipient of these events. Some events return event related data. The application must register an event handler with the host layer (via the USB_HOST_EventHandlerSet() function) before enabling any of the buses.

Remarks

None.

USB_HOST_EVENT_HANDLER Type

USB Host Layer Event Handler Function Pointer Type

File

usb_host.h

C

```
typedef USB_HOST_EVENT_RESPONSE (* USB_HOST_EVENT_HANDLER)(USB_HOST_EVENT event, void * eventData,
uintptr_t context);
```

Description

USB Host Layer Event Handler Function Pointer Type

This data type defines the required function signature of the USB Host Layer Event handling callback function. The application must register a pointer to a Host Layer Event handling function who's function signature (parameter and return value types) match the types specified by this function pointer in order to receive event call backs from the Host Layer. The Host Layer will invoke this function with event relevant parameters. The description of the event handler function parameters is given here.

event - Type of event generated.

pData - This parameter should be type cast to an event specific pointer type based on the event that has occurred. Refer to the USB_HOST_EVENT enumeration description for more details.

context - Value identifying the context of the application that was registered along with the event handling function.

Remarks

None.

USB_HOST_HCD Structure

Defines the USB Host HCD Information object that is provided to the host layer.

File

usb_host.h

C

```
typedef struct {
  SYS_MODULE_INDEX drvIndex;
  void * hcdInterface;
} USB_HOST_HCD;
```

Members

Members	Description
SYS_MODULE_INDEX drvIndex;	Index of the USB Host Controller driver that the host layer should open and use.
void * hcdInterface;	USB Host Controller Driver function pointers

Description

USB Host Controller Driver Information

This data type defines the data required to connect a Host Controller Driver to the host layer. The USB Host layer used the HCD routines to access the root hub and the USB.

Remarks

This data structure is specific to the PIC32 implementation of the USB Host layer.

USB_HOST_REQUEST_HANDLE Type

USB Host Request Handle Type

File

usb_host.h

C

```
typedef uintptr_t USB_HOST_REQUEST_HANDLE;
```

Description

USB Host Request Handle Type

This type defines the USB Host Request Handle. This type of handle is returned by the USB_HOST_DeviceStringDescriptorGet() function. Each request will generate a unique handle. This handle will be returned in the event associated with the completion of the string descriptor request.

Remarks

None.

USB_HOST_RESULT Enumeration

USB Host Results.

File

usb_host.h

C

```
typedef enum {
  USB_HOST_RESULT_REQUEST_BUSY = USB_HOST_RESULT_MIN,
  USB_HOST_RESULT_STRING_DESCRIPTOR_UNSUPPORTED,
  USB_HOST_RESULT_TRANSFER_ABORTED,
  USB_HOST_RESULT_REQUEST_STALLED,
  USB_HOST_RESULT_PIPE_HANDLE_INVALID,
  USB_HOST_RESULT_END_OF_DEVICE_LIST,
  USB_HOST_RESULT_INTERFACE_UNKNOWN,
  USB_HOST_RESULT_PARAMETER_INVALID,
  USB_HOST_RESULT_CONFIGURATION_UNKNOWN,
  USB_HOST_RESULT_BUS_NOT_ENABLED,
  USB_HOST_RESULT_BUS_UNKNOWN,
  USB_HOST_RESULT_DEVICE_UNKNOWN,
  USB_HOST_RESULT_FAILURE,
  USB_HOST_RESULT_FALSE = 0,
  USB_HOST_RESULT_TRUE = 1,
  USB_HOST_RESULT_SUCCESS = USB_HOST_RESULT_TRUE
} USB_HOST_RESULT;
```

Members

Members	Description
USB_HOST_RESULT_REQUEST_BUSY = USB_HOST_RESULT_MIN	Indicates that the Host Layer cannot accept any requests at this point
USB_HOST_RESULT_STRING_DESCRIPTOR_UNSUPPORTED	The device does not support the request string descriptor
USB_HOST_RESULT_TRANSFER_ABORTED	Request was aborted
USB_HOST_RESULT_REQUEST_STALLED	Request was stalled
USB_HOST_RESULT_PIPE_HANDLE_INVALID	The specified pipe is not valid
USB_HOST_RESULT_END_OF_DEVICE_LIST	The end of the device list was reached.
USB_HOST_RESULT_INTERFACE_UNKNOWN	The specified interface is not available
USB_HOST_RESULT_PARAMETER_INVALID	A NULL parameter was passed to the function
USB_HOST_RESULT_CONFIGURATION_UNKNOWN	The specified configuration does not exist on this device.
USB_HOST_RESULT_BUS_NOT_ENABLED	A bus operation was requested but the bus was not operated
USB_HOST_RESULT_BUS_UNKNOWN	The specified bus does not exist in the system
USB_HOST_RESULT_DEVICE_UNKNOWN	The specified device does not exist in the system
USB_HOST_RESULT_FAILURE	An unknown failure has occurred
USB_HOST_RESULT_FALSE = 0	Indicates a false condition
USB_HOST_RESULT_TRUE = 1	Indicate a true condition
USB_HOST_RESULT_SUCCESS = USB_HOST_RESULT_TRUE	Indicates that the operation succeeded or the request was accepted and will be processed.

Description

USB Host Result

This enumeration defines the possible returns values of USB Host Layer API. A function may only return some of the values in this enumeration. Refer to function description for details on which values will be returned.

Remarks

None.

USB_HOST_STRING_REQUEST_COMPLETE_CALLBACK Type

USB Host Device String Descriptor Request Complete Callback Function Type

File

usb_host.h

C

```
typedef void (* USB_HOST_STRING_REQUEST_COMPLETE_CALLBACK)(USB_HOST_REQUEST_HANDLE requestHandle, size_t
size, uintptr_t context);
```

Description

USB Host Device String Descriptor Request Complete Callback Function Type

This data type defines the required function signature of the USB Host Device String Descriptor Request Complete Callback Function. The application must specify a pointer to a function who's function signature (parameter and return value types) matches the type specified by this function pointer in order to a call backs from the Host Layer when the USB_HOST_DeviceStringDescriptorGet() function has completed its operation. The description of the callback function parameters is given here.

requestHandle - a handle that is unique to this request. This will match the handle that was returned by the USB_HOST_DeviceStringDescriptorGet() function.

size - size of the returned string descriptor. If the string descriptor could not be obtained, the size will be zero.

context - Value identifying the context of the application that was registered along with the event handling function.

Remarks

None.

USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will

cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

USB_HOST_TPL_ENTRY Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

vid Enumeration

USB Host Layer TPL Table Entry Matching Criteria flag

File

usb_host.h

C

```
typedef enum {
  initData,
  classCode,
  subClassCode,
  protocolCode
} driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, subClassCode, 0xFF }, driver)\
{\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \
.tplFlags.ignoreClass = false, \ .tplFlags.ignoreClass = false, \ .tplFlags.ignoreProtocol = true, \
.hostClientDriverInitData = initData, driver)\ {\ .id.cl_sc_p = { classCode, \ .hostClientDriverInitData =
initData, driver)\ {\ .id.cl_sc_p = { classCode, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreClass = false, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
\ .hostClientDriverInitData = initData, 0xFF, 0xFF, 0xFF }, driver)\ {\ .id.vid_pid = { 0xFFFF, \
.tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL), \ .tplFlags.ignoreProtocol = true, \
.tplFlags.ignoreProtocol = true, \ .tplFlags.ignoreProtocol = true, \ .hostClientDriverInitData = initData,
driver)\ {\ .id.vid_pid = { vid, pid, \ .hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = {
vid, pid }, driver)\ {\ .id.vid_pid = { 0xFFFF, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 0, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, driver)\
{\ .id.vid_pid = { vid, pid, \ .pidMask = mask, \ .hostClientDriverInitData = initData, driver)\ {\
.id.vid_pid = { vid, pid }, \ .pidMask = mask, \ .tplFlags.driverType = (TPL_FLAG_VID_PID), \
.tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 0, \ .hostClientDriverInitData = initData, \
.hostClientDriverInitData = initData, driver)\ {\ .id.vid_pid = { 0xFFFF, 0xFFFF }, \ .pidMask = 0x0000, \
.tplFlags.driverType = (TPL_FLAG_VID_PID), \ .tplFlags.ignoreVIDPID = 1, \ .tplFlags.ignoreVIDPID = 1, \
.hostClientDriverInitData = initData, \ .hostClientDriver = driver\ } typedef struct { union { uint32_t
value; struct { uint16_t vid; uint16_t pid; } vid_pid; struct { uint8_t classCode; uint8_t subClassCode;
uint8_t protocolCode; } cl_sc_p; } id; uint16_t pidMask; struct { unsigned driverType :1; unsigned
ignoreClass :1; unsigned ignoreSubClass :1; unsigned ignoreProtocol :1; unsigned pidMasked :1; unsigned
ignoreVIDPID :1; } tplFlags; void * hostClientDriverInitData; void * hostClientDriver; }
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY, USB_HOST_TPL_ENTRY;
```

Description

USB Host Layer TPL Table Entry Matching Criteria flag

This enumeration defines the possible matching criteria flag that can be specified for a Host TPL table entry. The tplFlag member of the TPL table entry should be set to one or more of these flags. These flags define the criteria that the Host layer will use while matching the attached device to the TPL table entry. For example, if a device is specified by class, subclass and protocol specifying the TPL_FLAG_IGNORE_SUBCLASS flag will cause the Host layer to ignore the subclass while comparing the class, subclass and protocol of the attached device.

Multiple flags can be specified as a logically OR'ed combination. While combining multiple flags, VID and PID criteria flags cannot be combined with the Class, Subclass, Protocol flags. For example, the TPL_FLAG_VID_PID flag cannot be combined with TPL_FLAG_IGNORE_SUBCLASS.

Remarks

None.

USB_HOST_BUS_ALL Macro

USB Host Bus All

File

usb_host.h

C

```
#define USB_HOST_BUS_ALL ((USB_HOST_BUS)(0xFF))
```

Description

USB Host Bus All

This constant defines the value that should be passed to the USB_HOST_BusSuspend(), USB_HOST_BusResume() and USB_HOST_IsBusSuspended() function if all the USB segments must be addressed. Passing this constant to these functions will cause Suspend and Resume operation to affect all the USB segments and hence affect all connected devices.

Remarks

None.

USB_HOST_DEVICE_OBJ_HANDLE_INVALID Macro

Defines an invalid USB Device Object Handle.

File

usb_host.h

C

```
#define USB_HOST_DEVICE_OBJ_HANDLE_INVALID ((USB_HOST_DEVICE_OBJ_HANDLE)(-1))
```

Description

USB Host Invalid Device Object Handle

This constant defines an invalid USB Device Object Handle. The USB_HOST_DeviceGetFirst() and the USB_HOST_DeviceGetNext() functions return this value in the deviceObjHandle member of the USB_HOST_DEVICE_INFO object when there are no attached devices to report.

Remarks

None.

USB_HOST_DEVICE_STRING_LANG_ID_DEFAULT Macro

Defines the default Lang ID to be used while obtaining the string.

File

usb_host.h

C

```
#define USB_HOST_DEVICE_STRING_LANG_ID_DEFAULT (0)
```

Description

USB Host Device String Default Lang ID

This constant defines the default Lang ID. When then languageID parameter in the USB_HOST_DeviceStringDescriptorGet() function is set to this value, the function will specify the default Lang ID while requesting the string from the device.

Remarks

None.

USB_HOST_REQUEST_HANDLE_INVALID Macro

USB Host Request Invalid Handle

File

usb_host.h

C

```
#define USB_HOST_REQUEST_HANDLE_INVALID ((USB_HOST_REQUEST_HANDLE)(-1))
```

Description

USB Host Request Invalid Handle

This constant defines an Invalid USB Host Request Handle. This handle is returned by the USB_HOST_DeviceStringDescriptorGet() function when the request was not accepted.

Remarks

None.

USB_HOST_RESULT_MIN Macro

USB Host Result Minimum Constant.

File

usb_host.h

C

```
#define USB_HOST_RESULT_MIN -100
```

Description

USB Host Result Minimum Constant

Constant identifying the USB Host Result Minimum Value. This constant is used in the USB_HOST_RESULT enumeration.

Remarks

None.

Files

Files

Name	Description
usb_host.h	USB Host Layer Interface Header
usb_host_config_template.h	USB host configuration template header file.

Description

This section lists the source and header files used by the library.

usb_host.h

USB Host Layer Interface Header

Enumerations

	Name	Description
	\ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL)	USB Host Layer TPL Table Entry Matching Criteria flag
	\ .tplFlags.driverType = (TPL_FLAG_VID_PID)	USB Host Layer TPL Table Entry Matching Criteria flag
	0	USB Host Layer TPL Table Entry Matching Criteria flag
	0x0000	USB Host Layer TPL Table Entry Matching Criteria flag
	0xFF	USB Host Layer TPL Table Entry Matching Criteria flag
	0xFF }	USB Host Layer TPL Table Entry Matching Criteria flag
	0xFFFF	USB Host Layer TPL Table Entry Matching Criteria flag
	0xFFFF }	USB Host Layer TPL Table Entry Matching Criteria flag
	1	USB Host Layer TPL Table Entry Matching Criteria flag
	classCode	USB Host Layer TPL Table Entry Matching Criteria flag
	false	USB Host Layer TPL Table Entry Matching Criteria flag
	initData	USB Host Layer TPL Table Entry Matching Criteria flag
	mask	USB Host Layer TPL Table Entry Matching Criteria flag
	pid	USB Host Layer TPL Table Entry Matching Criteria flag
	pid }	USB Host Layer TPL Table Entry Matching Criteria flag
	subClassCode	USB Host Layer TPL Table Entry Matching Criteria flag
	true	USB Host Layer TPL Table Entry Matching Criteria flag
	USB_HOST_DEVICE_STRING	Defines a defines types of strings that can be request through the USB_HOST_DeviceStringDescriptorGet() function.
	USB_HOST_EVENT	Defines the different events that the USB Host Layer can generate.
	USB_HOST_EVENT_RESPONSE	Host Layer Events Handler Function Response Type.
	USB_HOST_RESULT	USB Host Results.
	USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY	USB Host Layer TPL Table Entry Matching Criteria flag
	USB_HOST_TPL_ENTRY	USB Host Layer TPL Table Entry Matching Criteria flag
	vid	USB Host Layer TPL Table Entry Matching Criteria flag

Functions

	Name	Description
	USB_HOST_BusEnable	Starts host operations.

	USB_HOST_BusIsEnabled	Checks if the bus is enabled.
	USB_HOST_BusIsSuspended	Returns the suspend status of the bus.
	USB_HOST_BusResume	Resumes the bus.
	USB_HOST_BusSuspend	Suspends the bus.
	USB_HOST_Deinitialize	Deinitializes the specified instance of the USB Host Layer.
	USB_HOST_DeviceGetFirst	Returns information about the first attached device on the bus.
	USB_HOST_DeviceGetNext	Returns information about the next device on the bus.
	USB_HOST_DeviceIsSuspended	Returns the suspend state of the device is suspended.
	USB_HOST_DeviceResume	Resumes the selected device
	USB_HOST_DeviceSpeedGet	Returns the speed at which this device is operating.
	USB_HOST_DeviceStringDescriptorGet	Retrieves specified string descriptor from the device
	USB_HOST_DeviceSuspend	Suspends the specified device.
	USB_HOST_EventHandlerSet	USB Host Layer Event Handler Callback Function set function.
	USB_HOST_Initialize	Initializes the USB Host layer instance specified by the index.
	USB_HOST_Status	Gets the current status of the USB Host Layer.
	USB_HOST_Tasks	Maintains the USB Host Layer state machine.

Macros

	Name	Description
	USB_HOST_BUS_ALL	USB Host Bus All
	USB_HOST_DEVICE_OBJ_HANDLE_INVALID	Defines an invalid USB Device Object Handle.
	USB_HOST_DEVICE_STRING_LANG_ID_DEFAULT	Defines the default Lang ID to be used while obtaining the string.
	USB_HOST_REQUEST_HANDLE_INVALID	USB Host Request Invalid Handle
	USB_HOST_RESULT_MIN	USB Host Result Minimum Constant.

Structures

	Name	Description
	USB_HOST_DEVICE_INFO	Defines the data type that is used by the USB_HOST_DeviceGetFirst() and USB_HOST_DeviceGetNext() functions.
	USB_HOST_HCD	Defines the USB Host HCD Information object that is provided to the host layer.
	USB_HOST_INIT	Defines the data required to initialize a USB Host Layer instance.

Types

	Name	Description
	USB_HOST_BUS	Defines a USB Bus Data Type.
	USB_HOST_DEVICE_OBJ_HANDLE	Handle to an attached USB Device.
	USB_HOST_EVENT_HANDLER	USB Host Layer Event Handler Function Pointer Type
	USB_HOST_REQUEST_HANDLE	USB Host Request Handle Type
	USB_HOST_STRING_REQUEST_COMPLETE_CALLBACK	USB Host Device String Descriptor Request Complete Callback Function Type

Description

USB Host Layer Interface Definition

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB HOST layer.

File Name

usb_host.h

Company

Microchip Technology Inc.

usb_host_config_template.h

USB host configuration template header file.

Macros

	Name	Description
	USB_HOST_CONTROLLERS_NUMBER	Defines the number of USB Host Controllers that this Host Layer must manage.
	USB_HOST_DEVICE_INTERFACES_NUMBER	Defines the maximum number of interface that the attached device can contain in order for the USB Host Layer to process the device.
	USB_HOST_DEVICES_NUMBER	Defines the maximum number of devices to support.
	USB_HOST_HUB_SUPPORT_ENABLE	Defines if this USB Host application must support a Hub.
	USB_HOST_HUB_TIER_LEVEL	Defines the maximum tier of connected hubs to be supported.
	USB_HOST_PIPES_NUMBER	Defines the maximum number of pipes that the application will need.
	USB_HOST_TRANSFERS_NUMBER	Defines the maximum number of transfers that host layer should handle.

Description

USB Host Layer Configuration constants

This file contains USB host layer compile time options (macros) that are to be configured by the user. This file is a template file and must be used as an example only. This file must not be directly included in the project.

File Name

usb_host_config_template.h

Company

Microchip Technology Inc.

USB Audio v1.0 Host Client Driver Library

This section describes the USB Audio v1.0 Host Client Driver Library.

Introduction

Introduces the MPLAB Harmony USB Audio v1.0 Host Client Driver Library.

Description

The USB Audio v1.0 Host Client Driver in the MPLAB Harmony USB Host Stack allows USB Host applications to support and interact with Audio v1.0 USB devices. The USB Audio v1.0 Host Client Driver has the following features:

- Supports Audio v1.0 device with multiple streaming interfaces
- Designed to support multi-client operation
- RTOS ready
- Features an event driver non-clocking application interaction model
- Supports queuing of read and write data transfers

Using the Library

This topic describes the basic architecture of the USB Audio v1.0 Host Client Driver Library and provides information and examples on its use.

Abstraction Model

Describes the Abstraction Model of the USB Audio v1.0 Host Client Driver Library.

Description

The USB Audio v1.0 Host Client Driver interacts with Host Layer to control the attached Audio v1.0 device. The USB Host Layer attaches the Audio v1.0 Host Client Driver to the Audio v1.0 device when it meets the matching criteria specified in the USB Host TPL table. The Audio v1.0 Host Client Driver abstracts the details of sending Audio v1.0 class specific control transfer commands by providing easy to use non-blocking API to send these command. A command when issued is assigned a request handle. This request handle is returned in the event that is generated when the command has been processed, and can be used by the application to track the command.

While transferring data Audio Stream Data over the USB Audio v1.0 Host Client Driver abstracts details such as the Audio Streaming interface, endpoints and endpoint size. The USB Audio v1.0 Host Client Driver internally (and without application intervention) validates the Audio v1.0 class specific device descriptors and opens isochronous pipes. While transferring data, multiple read and write requests can be queued. Each such request gets assigned a transfer handle. The transfer handle for a transfer request is returned along with the completion event for that transfer request. The data transfer routines are implemented in `usb_host_audio_v1_0.c`.

Library Overview

The USB Audio v1.0 Host Client Driver can be grouped functionally as shown in the following table.

Library Interface Section	Description
Audio Device access Functions	These functions allow application clients to perform audio control transfers, register event handlers and get the number of stream groups and the details of each audio stream. These functions are implemented in the `usb_host_audio_v1_0.c` file.
Audio Stream Access Functions	These functions allow the application client to open audio streams, set parameters of an audio stream, and perform data transfer operations on an audio stream. These functions are implemented in the `usb_host_audio_v1_0.c` file.

How the Library Works

Describes how the library works and how it should be used.

Description

The USB Audio v1.0 Host Client Driver provides the user application with an easy-to-use interface to the attached Audio v1.0 device. The USB Host Layer initializes the USB Audio v1.0 Host Client Driver when a device is attached. This process does not require application intervention. The following sections describe the steps and methods required for the user application to interact with the attached devices.

TPL Table Configuration for Audio v1.0 Devices

Describes how to configure TPL table options, which includes a code example.

Description

The Host Layer attaches the Audio v1.0 Host Client Driver to a device when the device class in the Interface descriptor matches the entry in the TPL table. When specifying the entry for the Audio v1.0 device, the entry for the Audio v1.0 device, the driver interface must be set to USB_HOST_AUDIO_V1_0_INTERFACE. This will attach the Audio v1.0 Host Client Driver to the device when the USB Host matches the TPL entry to the device. The following code shows possible TPL table options for matching Audio v1.0 Devices.

```
/* This code shows an example of TPL table entries for supporting Audio v1.0
 * devices. Note the driver interface is set to USB_HOST_AUDIO_V1_0_INTERFACE. This
 * will load the Audio v1.0 Host Client Driver when there is TPL match */

const USB_HOST_TPL_ENTRY USBTPList[1] =
{
    /* This entry looks for any Audio v1.0 device. The Audio v1.0 Host Client Driver will
     * check if this is an Audio Streaming Device and will then load itself */
    TPL_INTERFACE_CLASS(USB_AUDIO_CLASS_CODE, NULL, USB_HOST_AUDIO_V1_0_INTERFACE),

};
```

Detecting Device Attach

Describes how to detect when a Audio v1.0 Device is attached, which includes a code example.

Description

The application will need to know when a Audio v1.0 Device is attached. To receive this attach event from the Audio v1.0 Host Client Driver, the application must register an Attach Event Handler by calling the USB_HOST_AUDIO_V1_0_AttachEventHandlerSet function. This function should be called before the USB_HOST_BusEnable function is called, else the application may miss Audio v1.0 attach events. It can be called multiple times to register multiple event handlers, each for different application clients that need to know about Audio v1.0 Device Attach events.

The total number of event handlers that can be registered is defined by USB_HOST_AUDIO_V1_0_ATTACH_LISTENERS_NUMBER configuration option in `system_config.h`. When a device is attached, the Audio v1.0 Host Client Driver will send the attach event to all the registered event handlers. In this event handler, the USB Audio v1.0 Host Client Driver will pass a USB_HOST_AUDIO V1_0_OBJ that can be opened to gain access to the device. The following code shows an example of how to register attach event handlers.

```
/* This code shows an example of Audio v1.0 Attach Event Handler and how this
 * attach event handler can be registered with the Audio v1.0 Host Client Driver */
bool isAudioDeviceAttached = false;
USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj;

/* Audio attach event listener function */
void APP_USBHostAudioAttachEventListener
```

```
(
    USB_HOST_AUDIO_V1_0_OBJ audioObj,
    USB_HOST_AUDIO_V1_0_EVENT event,
    uintptr_t context
)
{
    /* This function gets called when the Audio v1.0 device is attached/detached. In this
     * example we let the application know that a device is attached and we
     * store the Audio v1.0 device object. This object will be required to open the
     * device. */
    switch (event)
    {
        case USB_HOST_AUDIO_V1_0_EVENT_ATTACH:
            if (isAudioDeviceAttached == false)
            {
                isAudioDeviceAttached = true;
                audioDeviceObj = audioObj;
            }
            else
            {
                /* This application supports only one Audio Device . Handle Error Here.*/
            }
        break;
        case USB_HOST_AUDIO_V1_0_EVENT_DETACH:
            if (isAudioDeviceAttached == true)
            {
                /* This means the device was detached. There is no event data
                 * associated with this event.*/
                isAudioDeviceAttached = false;
                break;
            }
        break;
    }
}


void APP_Tasks(void)
{
    switch (appData.state)
    {
        case APP_STATE_BUS_ENABLE:

            /* In this state the application enables the USB Host Bus. Note
             * how the Audio v1.0 Attach event handler is registered before the bus
             * is enabled. */

            USB_HOST_AUDIO_V1_0_AttachEventHandlerSet(APP_USBHostAudioAttachEventListener, (uintptr_t) 0);
            USB_HOST_BusEnable(0);
            appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
            break;

        case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:
            /* Here we wait for the bus enable operation to complete. */
            break;
    }
}
```

Obtaining Audio v1.0 Device Audio Stream Details

Describes how to obtain audio stream details, which includes a code example.

Description

The application will need to know more details about an attached audio device like Number of Audio Stream Groups and audio format details of each audio stream in audio stream group. Application will need to search through all of the audio streams and find if a suitable audio stream is available before it can open a stream and start communicating.

USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet function can be used to know how many stream groups are available in the attached Audio device. This function takes USB_HOST_AUDIO_V1_0_OBJ as an argument and returns uint8_t value as number of stream groups.

USB_HOST_AUDIO_V1_0_StreamGetFirst function can be used to find out audio format details of first audio stream in a Stream Groups. This function takes USB_HOST_AUDIO_V1_0_OBJ, stream group index and pointer to the USB_HOST_AUDIO_V1_0_STREAM_INFO as arguments. The stream index can any number between zero to number of stream groups returned by USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet function. The audio stream object returned as part of USB_HOST_AUDIO_V1_0_STREAM_OBJ structure.

USB_HOST_AUDIO_V1_0_StreamGetNext function can be used to find details about subsequent audio streams. When there are no more audio streams available in the specified audio stream group this function return USB_HOST_AUDIO_V1_0_RESULT_END_OF_STREAM_LIST error. It is application's responsibility to map and Audio Stream group and an audio stream.

If the application is looking for a audio stream with certain properties, application need compare audio stream properties with members of the USB_HOST_AUDIO_V1_0_STREAM_INFO structure returned by USB_HOST_AUDIO_V1_0_StreamGetFirst and USB_HOST_AUDIO_V1_0_StreamGetNext functions.

```
/* This code shows an example of getting details about audio stream
   in an attached Audio v1.0 device.*/

/* Specify the Audio Stream format details that this application supports */
const APP_USB_HOST_AUDIO_STREAM_FORTMAT audioSpeakerStreamFormat =
{

    .streamDirection = USB_HOST_AUDIO_V1_0_DIRECTION_OUT,
    .format = USB_AUDIO_FORMAT_PCM,
    .nChannels = 2,
    .bitResolution = 16,
    .subFrameSize = 2,
    .samplingRate = 48000
};

bool isAudioDeviceAttached = false;
USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj;

/*********************************************************/
/* Function to search for a specific Audio Stream */
/*********************************************************/
USB_HOST_AUDIO_V1_0_STREAM_OBJ App_USBHostAudioSpeakerStreamFind
(
    USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj,
    APP_USB_HOST_AUDIO_STREAM_FORTMAT audioStream,
    uint8_t* numberofStreamGroups
)
{
    USB_HOST_AUDIO_V1_0_RESULT result;
    USB_HOST_AUDIO_V1_0_STREAM_INFO streamInfo;

    /* Get Number of Stream Groups */
    *numberofStreamGroups = USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet(audioDeviceObj);
    if (*numberofStreamGroups == 0)
    {
        return (USB_HOST_AUDIO_V1_0_STREAM_OBJ)0;
    }
    /* Get the First Stream Information in the Stream Group */
    result = USB_HOST_AUDIO_V1_0_StreamGetFirst(appData.audioDeviceObj, 0, &streamInfo);
    if (result == USB_HOST_AUDIO_V1_0_RESULT_SUCCESS)
    {
         /* Compare Audio Stream info */
        if ((streamInfo.format == audioStream.format)
            && (streamInfo.streamDirection == audioStream.streamDirection)
                    && (streamInfo.nChannels == audioStream.nChannels)
                    && (streamInfo.bitResolution == audioStream.bitResolution)
                    && (streamInfo.subFrameSize == audioStream.subFrameSize))
        {
            return streamInfo.streamObj;
        }
    }
    return (USB_HOST_AUDIO_V1_0_STREAM_OBJ)0;
}

/*********************************************************/
/* Audio attach event listener function */
/*********************************************************/
```

```
void APP_USBHostAudioAttachEventListener
(
    USB_HOST_AUDIO_V1_0_OBJ audioObj,
    USB_HOST_AUDIO_V1_0_EVENT event,
    uintptr_t context
)
{
    /* This function gets called when the Audio v1.0 device is attached/detached. In this
     * example we let the application know that a device is attached and we
     * store the Audio v1.0 device object. This object will be required to open the
     * device. */
    switch (event)
    {
        case USB_HOST_AUDIO_V1_0_EVENT_ATTACH:
            if (isAudioDeviceAttached == false)
            {
                isAudioDeviceAttached = true;
                audioDeviceObj = audioObj;
            }
            else
            {
                /* This application supports only one Audio Device . Handle Error Here.*/
            }
        break;
        case USB_HOST_AUDIO_V1_0_EVENT_DETACH:
            if (isAudioDeviceAttached == true)
            {
                /* This means the device was detached. There is no event data
                 * associated with this event.*/
                isAudioDeviceAttached = false;
                break;
            }
        break;
    }
}

/*******************************************************/
/* Audio Tasks function */
/*******************************************************/
void APP_Tasks ( void )
{
    USB_HOST_AUDIO_V1_0_RESULT audioResult;
    USB_HOST_AUDIO_V1_0_STREAM_RESULT streamResult;

    /* Check the application's current state. */
    switch ( appData.state )
    {
        case APP_STATE_BUS_ENABLE:

            /* Register a callback for Audio Device Attach. */
            audioResult = USB_HOST_AUDIO_V1_0_AttachEventHandlerSet
                          (
                              &APP_USBHostAudioAttachEventListener,
                              (uintptr_t)0
                          );

            if (audioResult == USB_HOST_AUDIO_V1_0_RESULT_SUCCESS )
            {
                /* Set Host Event Handler */
                USB_HOST_EventHandlerSet(APP_USBHostEventHandler, 0);
                USB_HOST_BusEnable(0);
                /* Advance application state */
                appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
            }
            break;

        case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:
            if(USB_HOST_BusIsEnabled(0))
            {
```

```
                appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            }
            break;

        case APP_STATE_WAIT_FOR_DEVICE_ATTACH:
            /* Check if an Audio Device has been attached  */
            if(appData.isAudioDeviceAttached == true)
            {
                appData.nAudioStreamGroups = 0;
                /* Find an Audio Stream matching to our requirement */
                appData.ouStreamObj = App_USBHostAudioSpeakerStreamFind
                                      (
                                          appData.audioDeviceObj,
                                          audioSpeakerStreamFormat,
                                          &appData.nAudioStreamGroups
                                      );
                if (appData.nAudioStreamGroups == 0)
                {
                    appData.state = APP_STATE_ERROR;
                    break;
                }
            }
            break;

        default:
            break;
    }
}
```

Obtaining an Audio Stream

Describes how to open an audio stream, which includes a code example.

Description

Once application has identified which audio stream to use, application must open that audio stream by using USB_HOST_AUDIO_V1_0_StreamOpen function. This function takes audio stream object USB_HOST_AUDIO_V1_0_STREAM_OBJ as an argument which obtained by USB_HOST_AUDIO_V1_0_StreamGetFirst and USB_HOST_AUDIO_V1_0_StreamGetNext functions and returns audio stream handle USB_HOST_AUDIO_V1_0_STREAM_HANDLE. If the open function fails, it returns an invalid handle (USB_HOST_AUDIO_V1_0_STREAM_HANDLE _INVALID). Once opened successfully, a valid handle tracks the relationship between the client and the Audio Stream. This handle should be used with other Audio Stream functions.

An audio stream can be opened multiple times by different application clients. In an RTOS based application each client could running its own thread. Multiple clients can read write data to the one Audio stream. In such a case, the read and write requests are queued. The following code shows an example of how an Audio Stream is opened.

```
/* This code shows an example of opening an audio stream */

/* Specify the Audio Stream format details that this application supports */
const APP_USB_HOST_AUDIO_STREAM_FORTMAT audioSpeakerStreamFormat =
{

    .streamDirection = USB_HOST_AUDIO_V1_0_DIRECTION_OUT,
    .format = USB_AUDIO_FORMAT_PCM,
    .nChannels = 2,
    .bitResolution = 16,
    .subFrameSize = 2,
    .samplingRate = 48000
};

bool isAudioDeviceAttached = false;
USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj;

/*********************************************************/
/* Function to search for a specific Audio Stream */
/*********************************************************/
USB_HOST_AUDIO_V1_0_STREAM_OBJ App_USBHostAudioSpeakerStreamFind
(
    USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj,
    APP_USB_HOST_AUDIO_STREAM_FORTMAT audioStream,
    uint8_t* numberofStreamGroups
```

```
)
{
    USB_HOST_AUDIO_V1_0_RESULT result;
    USB_HOST_AUDIO_V1_0_STREAM_INFO streamInfo;

    /* Get Number of Stream Groups */
    *numberofStreamGroups = USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet(audioDeviceObj);
    if (*numberofStreamGroups == 0)
    {
        return (USB_HOST_AUDIO_V1_0_STREAM_OBJ)0;
    }
    /* Get the First Stream Information in the Stream Group */
    result = USB_HOST_AUDIO_V1_0_StreamGetFirst(appData.audioDeviceObj, 0, &streamInfo);
    if (result == USB_HOST_AUDIO_V1_0_RESULT_SUCCESS)
    {
         /* Compare Audio Stream info */
        if ((streamInfo.format == audioStream.format)
            && (streamInfo.streamDirection == audioStream.streamDirection)
                    && (streamInfo.nChannels == audioStream.nChannels)
                    && (streamInfo.bitResolution == audioStream.bitResolution)
                    && (streamInfo.subFrameSize == audioStream.subFrameSize))
        {
            return streamInfo.streamObj;
        }
    }
    return (USB_HOST_AUDIO_V1_0_STREAM_OBJ)0;
}

/*********************************************************/
/* Audio attach event listener function */
/*********************************************************/
void APP_USBHostAudioAttachEventListener
(
    USB_HOST_AUDIO_V1_0_OBJ audioObj,
    USB_HOST_AUDIO_V1_0_EVENT event,
    uintptr_t context
)
{
    /* This function gets called when the Audio v1.0 device is attached/detached. In this
     * example we let the application know that a device is attached and we
     * store the Audio v1.0 device object. This object will be required to open the
     * device. */
    switch (event)
    {
        case USB_HOST_AUDIO_V1_0_EVENT_ATTACH:
            if (isAudioDeviceAttached == false)
            {
                isAudioDeviceAttached = true;
                audioDeviceObj = audioObj;
            }
            else
            {
                /* This application supports only one Audio Device . Handle Error Here.*/
            }
        break;
        case USB_HOST_AUDIO_V1_0_EVENT_DETACH:
            if (isAudioDeviceAttached == true)
            {
                /* This means the device was detached. There is no event data
                 * associated with this event.*/
                isAudioDeviceAttached = false;
                break;
            }
        break;
    }
}

/*********************************************************/
/* Audio Tasks function */
```

```
/*******************************************************/
void APP_Tasks ( void )
{
    USB_HOST_AUDIO_V1_0_RESULT audioResult;
    USB_HOST_AUDIO_V1_0_STREAM_RESULT streamResult;

    /* Check the application's current state. */
    switch ( appData.state )
    {
        case APP_STATE_BUS_ENABLE:

            /* Register a callback for Audio Device Attach. */
            audioResult = USB_HOST_AUDIO_V1_0_AttachEventHandlerSet
                          (
                              &APP_USBHostAudioAttachEventListener,
                              (uintptr_t)0
                          );

            if (audioResult == USB_HOST_AUDIO_V1_0_RESULT_SUCCESS )
            {
                /* Set Host Event Handler */
                USB_HOST_EventHandlerSet(APP_USBHostEventHandler, 0);
                USB_HOST_BusEnable(0);
                /* Advance application state */
                appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
            }
            break;

        case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:
            if(USB_HOST_BusIsEnabled(0))
            {
                appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            }
            break;

        case APP_STATE_WAIT_FOR_DEVICE_ATTACH:
            /* Check if an Audio Device has been attached  */
            if(appData.isAudioDeviceAttached == true)
            {
                appData.nAudioStreamGroups = 0;
                /* Find an Audio Stream matching to our requirement */
                appData.ouStreamObj = App_USBHostAudioSpeakerStreamFind
                                      (
                                          appData.audioDeviceObj,
                                          audioSpeakerStreamFormat,
                                          &appData.nAudioStreamGroups
                                      );
                if (appData.nAudioStreamGroups == 0)
                {
                    appData.state = APP_STATE_ERROR;
                    break;
                }

                /* Open Audio Stream */
                appData.outStreamHandle = USB_HOST_AUDIO_V1_0_StreamOpen
                                          (
                                              appData.ouStreamObj
                                          );

                if (appData.outStreamHandle == USB_HOST_AUDIO_V1_0_STREAM_HANDLE_INVALID)
                {
                    appData.state = APP_STATE_ERROR;
                    break;
                }
            }
            break;

        default:
            break;
```

```
    }
}
```

Audio Stream Event Handling

Describes audio stream event handling, which includes a code example.

Description

The Audio v1.0 streams presents an event driven interface to the application. The USB Audio v1.0 Host Client Driver requires the application client to set an event handler against each audio stream for meaningful operation.

A request to send a command or transfer data typically completes after the command request or transfer function has exited. The application must then use the Audio stream event to track the completion of this command or data transfer request. In a case where multiple data transfers are queued, the transfer handles can be used to identify the transfer requests.

The application must use the USB_HOST_AUDIO_V1_0_StreamEventHandlerSet function to register an audio stream handler. This event handler will be called when a command or data transfer event has occurred and should be registered before the request for command or a data transfer. The following code shows an example of registering an audio stream event handler.

```
/* This code shows an example of Audio stream event handling */

/* Specify the Audio Stream format details that this application supports */
const APP_USB_HOST_AUDIO_STREAM_FORTMAT audioSpeakerStreamFormat =
{

    .streamDirection = USB_HOST_AUDIO_V1_0_DIRECTION_OUT,
    .format = USB_AUDIO_FORMAT_PCM,
    .nChannels = 2,
    .bitResolution = 16,
    .subFrameSize = 2,
    .samplingRate = 48000
};

bool isAudioDeviceAttached = false;
USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj;


/*********************************************************
 * Audio Stream Event Handler function.
 *********************************************************/

USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE APP_USBHostAudioStreamEventHandler
(
    USB_HOST_AUDIO_V1_0_STREAM_HANDLE streamHandle,
    USB_HOST_AUDIO_V1_0_STREAM_EVENT event,
    void * eventData,
    uintptr_t context
)
{
    USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE_DATA * writeCompleteEventData;
    switch(event)
    {
        case USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE:

            break;

        case USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE:

            break;
        case USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE:


            break;
        default:
            break;
    }
    return USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE_NONE;
}
/*********************************************************/
/* Function to search for a specific Audio Stream */
/*********************************************************/
```

```
USB_HOST_AUDIO_V1_0_STREAM_OBJ App_USBHostAudioSpeakerStreamFind
(
    USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj,
    APP_USB_HOST_AUDIO_STREAM_FORTMAT audioStream,
    uint8_t* numberofStreamGroups
)
{
    USB_HOST_AUDIO_V1_0_RESULT result;
    USB_HOST_AUDIO_V1_0_STREAM_INFO streamInfo;

    /* Get Number of Stream Groups */
    *numberofStreamGroups = USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet(audioDeviceObj);
    if (*numberofStreamGroups == 0)
    {
        return (USB_HOST_AUDIO_V1_0_STREAM_OBJ)0;
    }
    /* Get the First Stream Information in the Stream Group */
    result = USB_HOST_AUDIO_V1_0_StreamGetFirst(appData.audioDeviceObj, 0, &streamInfo);
    if (result == USB_HOST_AUDIO_V1_0_RESULT_SUCCESS)
    {
         /* Compare Audio Stream info */
        if ((streamInfo.format == audioStream.format)
            && (streamInfo.streamDirection == audioStream.streamDirection)
                    && (streamInfo.nChannels == audioStream.nChannels)
                    && (streamInfo.bitResolution == audioStream.bitResolution)
                    && (streamInfo.subFrameSize == audioStream.subFrameSize))
        {
            return streamInfo.streamObj;
        }
    }
    return (USB_HOST_AUDIO_V1_0_STREAM_OBJ)0;
}

/*********************************************************/
/* Audio attach event listener function */
/*********************************************************/
void APP_USBHostAudioAttachEventListener
(
    USB_HOST_AUDIO_V1_0_OBJ audioObj,
    USB_HOST_AUDIO_V1_0_EVENT event,
    uintptr_t context
)
{
    /* This function gets called when the Audio v1.0 device is attached/detached. In this
     * example we let the application know that a device is attached and we
     * store the Audio v1.0 device object. This object will be required to open the
     * device. */
    switch (event)
    {
        case USB_HOST_AUDIO_V1_0_EVENT_ATTACH:
            if (isAudioDeviceAttached == false)
            {
                isAudioDeviceAttached = true;
                audioDeviceObj = audioObj;
            }
            else
            {
                /* This application supports only one Audio Device . Handle Error Here.*/
            }
        break;
        case USB_HOST_AUDIO_V1_0_EVENT_DETACH:
            if (isAudioDeviceAttached == true)
            {
                /* This means the device was detached. There is no event data
                 * associated with this event.*/
                isAudioDeviceAttached = false;
                break;
            }
        break;
```

```
    }
}

/*******************************************************/
/* Audio Tasks function */
/*******************************************************/
void APP_Tasks ( void )
{
    USB_HOST_AUDIO_V1_0_RESULT audioResult;
    USB_HOST_AUDIO_V1_0_STREAM_RESULT streamResult;

    /* Check the application's current state. */
    switch ( appData.state )
    {
        case APP_STATE_BUS_ENABLE:

            /* Register a callback for Audio Device Attach. */
            audioResult = USB_HOST_AUDIO_V1_0_AttachEventHandlerSet
                          (
                              &APP_USBHostAudioAttachEventListener,
                              (uintptr_t)0
                          );

            if (audioResult == USB_HOST_AUDIO_V1_0_RESULT_SUCCESS )
            {
                /* Set Host Event Handler */
                USB_HOST_EventHandlerSet(APP_USBHostEventHandler, 0);
                USB_HOST_BusEnable(0);
                /* Advance application state */
                appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
            }
            break;

        case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:
            if(USB_HOST_BusIsEnabled(0))
            {
                appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            }
            break;

        case APP_STATE_WAIT_FOR_DEVICE_ATTACH:
            /* Check if an Audio Device has been attached  */
            if(appData.isAudioDeviceAttached == true)
            {
                appData.nAudioStreamGroups = 0;
                /* Find an Audio Stream matching to our requirement */
                appData.ouStreamObj = App_USBHostAudioSpeakerStreamFind
                                      (
                                          appData.audioDeviceObj,
                                          audioSpeakerStreamFormat,
                                          &appData.nAudioStreamGroups
                                      );
                if (appData.nAudioStreamGroups == 0)
                {
                    appData.state = APP_STATE_ERROR;
                    break;
                }

                /* Open Audio Stream */
                appData.outStreamHandle = USB_HOST_AUDIO_V1_0_StreamOpen
                                          (
                                              appData.ouStreamObj
                                          );

                if (appData.outStreamHandle == USB_HOST_AUDIO_V1_0_STREAM_HANDLE_INVALID)
                {
                    appData.state = APP_STATE_ERROR;
                    break;
                }
```

```
                /* Set Stream Event Handler  */
                streamResult = USB_HOST_AUDIO_V1_0_StreamEventHandlerSet
                               (
                                   appData.outStreamHandle,
                                   APP_USBHostAudioStreamEventHandler,
                                   (uintptr_t)appData.ouStreamObj
                               );

                if (streamResult != USB_HOST_AUDIO_V1_0_STREAM_SUCCESS)
                {
                    appData.state = APP_STATE_ERROR;
                    break;
                }
            }
            break;

        default:
            break;
    }
}
```

Enabling Audio Stream

Describes how to enable an audio stream, which includes a code example.

Description

An audio stream must be enabled before doing any data transfer operation. An audio stream enable or disable can be scheduled by using USB_HOST_AUDIO_V1_0_StreamEnable or USB_HOST_AUDIO_V1_0_StreamEnable functions. Return values of these function indicates if the request has been placed successfully or failed. When the audio stream enable request is completed, stream event handler generates an event USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE. Similarly it generates an event USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE when stream disable is complete. The event data USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE_DATA has details like request handle and termination status. The requestStatus member of the USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE_DATA indicates if the request was success or failed. When audio stream multiple audio streams with an audio stream group cannot be enabled at the same time. The following code shows an example of how an Audio Stream is enabled.

```
/* This code shows an example of enabling an audio stream */

/* Specify the Audio Stream format details that this application supports */
const APP_USB_HOST_AUDIO_STREAM_FORTMAT audioSpeakerStreamFormat =
{

    .streamDirection = USB_HOST_AUDIO_V1_0_DIRECTION_OUT,
    .format = USB_AUDIO_FORMAT_PCM,
    .nChannels = 2,
    .bitResolution = 16,
    .subFrameSize = 2,
    .samplingRate = 48000
};

bool isAudioDeviceAttached = false;
bool isStreamEnabled = false;
USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj;


/*********************************************************
 * Audio Stream Event Handler function.
 *********************************************************/

USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE APP_USBHostAudioStreamEventHandler
(
    USB_HOST_AUDIO_V1_0_STREAM_HANDLE streamHandle,
    USB_HOST_AUDIO_V1_0_STREAM_EVENT event,
    void * eventData,
    uintptr_t context
)
{
    USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE_DATA * writeCompleteEventData;
```

```
    switch(event)
    {
        case USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE:

            break;

        case USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE:
            /* Check eventData result member to know if stream enable is complete */
                isStreamEnabled = true;

            break;
        case USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE:


            break;
        default:
            break;
    }
    return USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE_NONE;
}
/************************************************************/
/* Function to search for a specific Audio Stream */
/************************************************************/
USB_HOST_AUDIO_V1_0_STREAM_OBJ App_USBHostAudioSpeakerStreamFind
(
    USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj,
    APP_USB_HOST_AUDIO_STREAM_FORTMAT audioStream,
    uint8_t* numberofStreamGroups
)
{
    USB_HOST_AUDIO_V1_0_RESULT result;
    USB_HOST_AUDIO_V1_0_STREAM_INFO streamInfo;

    /* Get Number of Stream Groups */
    *numberofStreamGroups = USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet(audioDeviceObj);
    if (*numberofStreamGroups == 0)
    {
        return (USB_HOST_AUDIO_V1_0_STREAM_OBJ)0;
    }
    /* Get the First Stream Information in the Stream Group */
    result = USB_HOST_AUDIO_V1_0_StreamGetFirst(appData.audioDeviceObj, 0, &streamInfo);
    if (result == USB_HOST_AUDIO_V1_0_RESULT_SUCCESS)
    {
         /* Compare Audio Stream info */
        if ((streamInfo.format == audioStream.format)
            && (streamInfo.streamDirection == audioStream.streamDirection)
                    && (streamInfo.nChannels == audioStream.nChannels)
                    && (streamInfo.bitResolution == audioStream.bitResolution)
                    && (streamInfo.subFrameSize == audioStream.subFrameSize))
        {
            return streamInfo.streamObj;
        }
    }
    return (USB_HOST_AUDIO_V1_0_STREAM_OBJ)0;
}

/************************************************************/
/* Audio attach event listener function */
/************************************************************/
void APP_USBHostAudioAttachEventListener
(
    USB_HOST_AUDIO_V1_0_OBJ audioObj,
    USB_HOST_AUDIO_V1_0_EVENT event,
    uintptr_t context
)
{
    /* This function gets called when the Audio v1.0 device is attached/detached. In this
     * example we let the application know that a device is attached and we
     * store the Audio v1.0 device object. This object will be required to open the
```

```
         * device. */
        switch (event)
        {
            case USB_HOST_AUDIO_V1_0_EVENT_ATTACH:
                if (isAudioDeviceAttached == false)
                {
                    isAudioDeviceAttached = true;
                    audioDeviceObj = audioObj;
                }
                else
                {
                    /* This application supports only one Audio Device . Handle Error Here.*/
                }
            break;
            case USB_HOST_AUDIO_V1_0_EVENT_DETACH:
                if (isAudioDeviceAttached == true)
                {
                    /* This means the device was detached. There is no event data
                     * associated with this event.*/
                    isAudioDeviceAttached = false;
                    break;
                }
            break;
        }
    }

    /*******************************************************/
    /* Audio Tasks function */
    /*******************************************************/
    void APP_Tasks ( void )
    {
        USB_HOST_AUDIO_V1_0_RESULT audioResult;
        USB_HOST_AUDIO_V1_0_STREAM_RESULT streamResult;

        /* Check the application's current state. */
        switch ( appData.state )
        {
            case APP_STATE_BUS_ENABLE:

                /* Register a callback for Audio Device Attach. */
                audioResult = USB_HOST_AUDIO_V1_0_AttachEventHandlerSet
                              (
                                  &APP_USBHostAudioAttachEventListener,
                                  (uintptr_t)0
                              );

                if (audioResult == USB_HOST_AUDIO_V1_0_RESULT_SUCCESS )
                {
                    /* Set Host Event Handler */
                    USB_HOST_EventHandlerSet(APP_USBHostEventHandler, 0);
                    USB_HOST_BusEnable(0);
                    /* Advance application state */
                    appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
                }
                break;

            case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:
                if(USB_HOST_BusIsEnabled(0))
                {
                    appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
                }
                break;

            case APP_STATE_WAIT_FOR_DEVICE_ATTACH:
                /* Check if an Audio Device has been attached  */
                if(appData.isAudioDeviceAttached == true)
                {
                    appData.nAudioStreamGroups = 0;
                    /* Find an Audio Stream matching to our requirement */
```

```
                appData.ouStreamObj = App_USBHostAudioSpeakerStreamFind
                                      (
                                          appData.audioDeviceObj,
                                          audioSpeakerStreamFormat,
                                          &appData.nAudioStreamGroups
                                      );
                if (appData.nAudioStreamGroups == 0)
                {
                    appData.state = APP_STATE_ERROR;
                    break;
                }

                /* Open Audio Stream */
                appData.outStreamHandle = USB_HOST_AUDIO_V1_0_StreamOpen
                                          (
                                              appData.ouStreamObj
                                          );

                if (appData.outStreamHandle == USB_HOST_AUDIO_V1_0_STREAM_HANDLE_INVALID)
                {
                    appData.state = APP_STATE_ERROR;
                    break;
                }

                /* Set Stream Event Handler  */
                streamResult = USB_HOST_AUDIO_V1_0_StreamEventHandlerSet
                               (
                                   appData.outStreamHandle,
                                   APP_USBHostAudioStreamEventHandler,
                                   (uintptr_t)appData.ouStreamObj
                               );

                if (streamResult != USB_HOST_AUDIO_V1_0_STREAM_SUCCESS)
                {
                    appData.state = APP_STATE_ERROR;
                    break;
                }
                appData.state = APP_STATE_ENABLE_AUDIO_STREAM;
            }
            break;

            case  APP_STATE_ENABLE_AUDIO_STREAM:
                isStreamEnableComplete = false;
                 /* Set default interface setting of the streaming interface */
                streamResult = USB_HOST_AUDIO_V1_0_StreamEnable
                               (
                                   appData.outStreamHandle,
                                   &appData.requestHandle
                               );
                if (streamResult != USB_HOST_AUDIO_V1_0_STREAM_SUCCESS)
                {
                    appData.state = APP_STATE_ERROR;
                    break;
                }
                appData.state = APP_STATE_WAIT_FOR_ENABLE_AUDIO_STREAM;
            break;
        case  APP_STATE_WAIT_FOR_ENABLE_AUDIO_STREAM:
            if (isStreamEnabled == true)
            {
                /* stream enable complete*/
            }
            break;

        default:
            break;
    }
}
```

Setting the Desired Audio Stream Sampling Rate

Describes how to set the desired audio stream sampling rate, which includes a code example.

Description

Sampling rate of an audio stream can be set using USB_HOST_AUDIO_V1_0_StreamSamplingRateSet function. Supported sampling rates for an audio stream is returned as part of USB_HOST_AUDIO_V1_0_STREAM_INFO by the USB_HOST_AUDIO_V1_0_StreamGetFirst and USB_HOST_AUDIO_V1_0_StreamGetNextfunctions. Return values of these function indicates if the request has been placed successfully or failed. When the set sampling rate request is completed the stream event handler generates an event USB_HOST_AUDIO_V1_0_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE. The event data USB_HOST_AUDIO_V1_0_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE_DATA has request handle and the requestStatus which indicates the set sampling request was accepted by the device or failed. The following code shows an example of how sampling rates can be set in an audio stream.

```
/* This code shows an example of Set sampling rate to an audio stream */

/* Specify the Audio Stream format details that this application supports */
const APP_USB_HOST_AUDIO_STREAM_FORTMAT audioSpeakerStreamFormat =
{

    .streamDirection = USB_HOST_AUDIO_V1_0_DIRECTION_OUT,
    .format = USB_AUDIO_FORMAT_PCM,
    .nChannels = 2,
    .bitResolution = 16,
    .subFrameSize = 2,
    .samplingRate = 48000
};

bool isAudioDeviceAttached = false;
bool isStreamEnabled = false;
bool isSampleRateSetComplete = false;
USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj;


/************************************************************
 * Audio Stream Event Handler function.
 ************************************************************/

USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE APP_USBHostAudioStreamEventHandler
(
    USB_HOST_AUDIO_V1_0_STREAM_HANDLE streamHandle,
    USB_HOST_AUDIO_V1_0_STREAM_EVENT event,
    void * eventData,
    uintptr_t context
)
{
    USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE_DATA * writeCompleteEventData;
    switch(event)
    {
        case USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE:

            break;

        case USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE:
            /* Check eventData result member to know if stream enable is complete */
                isStreamEnabled = true;

            break;
        case USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE:


            break;

         case USB_HOST_AUDIO_V1_0_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE:
                /* Check eventData result member to know if stream enable is complete */
                isSampleRateSetComplete = true;

            break;
```

```
            default:
                break;
        }
        return USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE_NONE;
    }
    /*********************************************************/
    /* Function to search for a specific Audio Stream */
    /*********************************************************/
    USB_HOST_AUDIO_V1_0_STREAM_OBJ App_USBHostAudioSpeakerStreamFind
    (
        USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj,
        APP_USB_HOST_AUDIO_STREAM_FORTMAT audioStream,
        uint8_t* numberofStreamGroups
    )
    {
        USB_HOST_AUDIO_V1_0_RESULT result;
        USB_HOST_AUDIO_V1_0_STREAM_INFO streamInfo;

        /* Get Number of Stream Groups */
        *numberofStreamGroups = USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet(audioDeviceObj);
        if (*numberofStreamGroups == 0)
        {
            return (USB_HOST_AUDIO_V1_0_STREAM_OBJ)0;
        }
        /* Get the First Stream Information in the Stream Group */
        result = USB_HOST_AUDIO_V1_0_StreamGetFirst(appData.audioDeviceObj, 0, &streamInfo);
        if (result == USB_HOST_AUDIO_V1_0_RESULT_SUCCESS)
        {
             /* Compare Audio Stream info */
            if ((streamInfo.format == audioStream.format)
                && (streamInfo.streamDirection == audioStream.streamDirection)
                        && (streamInfo.nChannels == audioStream.nChannels)
                        && (streamInfo.bitResolution == audioStream.bitResolution)
                        && (streamInfo.subFrameSize == audioStream.subFrameSize))
            {
                return streamInfo.streamObj;
            }
        }
        return (USB_HOST_AUDIO_V1_0_STREAM_OBJ)0;
    }

    /*********************************************************/
    /* Audio attach event listener function */
    /*********************************************************/
    void APP_USBHostAudioAttachEventListener
    (
        USB_HOST_AUDIO_V1_0_OBJ audioObj,
        USB_HOST_AUDIO_V1_0_EVENT event,
        uintptr_t context
    )
    {
        /* This function gets called when the Audio v1.0 device is attached/detached. In this
         * example we let the application know that a device is attached and we
         * store the Audio v1.0 device object. This object will be required to open the
         * device. */
        switch (event)
        {
            case USB_HOST_AUDIO_V1_0_EVENT_ATTACH:
                if (isAudioDeviceAttached == false)
                {
                    isAudioDeviceAttached = true;
                    audioDeviceObj = audioObj;
                }
                else
                {
                    /* This application supports only one Audio Device . Handle Error Here.*/
                }
            break;
```

```
            case USB_HOST_AUDIO_V1_0_EVENT_DETACH:
                if (isAudioDeviceAttached == true)
                {
                    /* This means the device was detached. There is no event data
                     * associated with this event.*/
                    isAudioDeviceAttached = false;
                    break;
                }
            break;
        }
    }

    /*******************************************************/
    /* Audio Tasks function */
    /*******************************************************/
    void APP_Tasks ( void )
    {
        USB_HOST_AUDIO_V1_0_RESULT audioResult;
        USB_HOST_AUDIO_V1_0_STREAM_RESULT streamResult;

        /* Check the application's current state. */
        switch ( appData.state )
        {
            case APP_STATE_BUS_ENABLE:

                /* Register a callback for Audio Device Attach. */
                audioResult = USB_HOST_AUDIO_V1_0_AttachEventHandlerSet
                              (
                                  &APP_USBHostAudioAttachEventListener,
                                  (uintptr_t)0
                              );

                if (audioResult == USB_HOST_AUDIO_V1_0_RESULT_SUCCESS )
                {
                    /* Set Host Event Handler */
                    USB_HOST_EventHandlerSet(APP_USBHostEventHandler, 0);
                    USB_HOST_BusEnable(0);
                    /* Advance application state */
                    appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
                }
                break;

            case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:
                if(USB_HOST_BusIsEnabled(0))
                {
                    appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
                }
                break;

            case APP_STATE_WAIT_FOR_DEVICE_ATTACH:
                /* Check if an Audio Device has been attached  */
                if(appData.isAudioDeviceAttached == true)
                {
                    appData.nAudioStreamGroups = 0;
                    /* Find an Audio Stream matching to our requirement */
                    appData.ouStreamObj = App_USBHostAudioSpeakerStreamFind
                                          (
                                              appData.audioDeviceObj,
                                              audioSpeakerStreamFormat,
                                              &appData.nAudioStreamGroups
                                          );
                    if (appData.nAudioStreamGroups == 0)
                    {
                        appData.state = APP_STATE_ERROR;
                        break;
                    }

                    /* Open Audio Stream */
                    appData.outStreamHandle = USB_HOST_AUDIO_V1_0_StreamOpen
```

```
                                    (
                                        appData.ouStreamObj
                                    );

            if (appData.outStreamHandle == USB_HOST_AUDIO_V1_0_STREAM_HANDLE_INVALID)
            {
                appData.state = APP_STATE_ERROR;
                break;
            }

            /* Set Stream Event Handler  */
            streamResult = USB_HOST_AUDIO_V1_0_StreamEventHandlerSet
                           (
                               appData.outStreamHandle,
                               APP_USBHostAudioStreamEventHandler,
                               (uintptr_t)appData.ouStreamObj
                           );

            if (streamResult != USB_HOST_AUDIO_V1_0_STREAM_SUCCESS)
            {
                appData.state = APP_STATE_ERROR;
                break;
            }
            appData.state = APP_STATE_ENABLE_AUDIO_STREAM;
        }
        break;

        case  APP_STATE_ENABLE_AUDIO_STREAM:
            isStreamEnableComplete = false;
             /* Set default interface setting of the streaming interface */
            streamResult = USB_HOST_AUDIO_V1_0_StreamEnable
                           (
                               appData.outStreamHandle,
                               &appData.requestHandle
                           );
            if (streamResult != USB_HOST_AUDIO_V1_0_STREAM_SUCCESS)
            {
                appData.state = APP_STATE_ERROR;
                break;
            }
            appData.state = APP_STATE_WAIT_FOR_ENABLE_AUDIO_STREAM;
        break;
    case  APP_STATE_WAIT_FOR_ENABLE_AUDIO_STREAM:
        if (isStreamEnabled == true)
        {
            /* Set sampling rate 48000 Hz */
            isSampleRateSetComplete = false;
            streamResult = USB_HOST_AUDIO_V1_0_StreamSamplingRateSet
                           (
                               appData.outStreamHandle,
                               &appData.requestHandle,
                               48000
                           );
            if (streamResult != USB_HOST_AUDIO_V1_0_STREAM_SUCCESS)
            {
                appData.state = APP_STATE_ERROR;
                break;
            }
            appData.state = APP_STATE_WAIT_FOR_SAMPLE_RATE_SET_COMPLETE;

        }
        break;
    case  APP_STATE_WAIT_FOR_SAMPLE_RATE_SET_COMPLETE:
        if (isSampleRateSetComplete == true)
        {
            /* Set sampling rate completed */
        }
    default:
        break;
```

```
    }
}
```

Audio Data Streaming

Describes how to transfer data to an audio stream, which includes a code example.

Description

The application can use the USB_HOST_AUDIO_V1_0_StreamRead and USB_HOST_AUDIO_V1_0_StreamWrite functions to transfer data to an Audio Stream. While calling these functions, the stream handle specifies the target Audio stream and the event handler function to which the events should be sent. It is possible for multiple clients to open the same audio stream and transfer data to the stream.

Calling the USB_HOST_AUDIO_V1_0_StreamRead and USB_HOST_AUDIO_V1_0_StreamWrite functions while a read/write transfer is already in progress will cause the transfer result to be queued. If the transfer was successfully queued or scheduled, the USB_HOST_AUDIO_V1_0_StreamRead and USB_HOST_AUDIO_V1_0_StreamWrite functions will return a valid transfer handle. This transfer handle identifies the transfer request. The application clients can use the transfer handles to keep track of multiple queued transfers. When a transfer completes, the Audio stream handler generates an event. The following table shows the event and the event data associated with the event.

Table 1: Read

Function	USB_HOST_AUDIO_V1_0_StreamRead
Event	USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE
Event Data Type	USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE _DATA

Table 2: Write

Function	USB_HOST_AUDIO_V1_0_StreamWrite
Event	USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE
Event Data Type	USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE _DATA

The event data contains information on the amount of data transferred, completion status and the transfer handle of the transfer. The following code shows an example of reading and writing data.

```
/* This code  shows an example of audio data streaming */

/* PCM16 samples for 1Khz Sine Wave at 48 kHz Sample Rate */
uint16_t audioSamples[96] =  {
    0x0000, 0x0000,  //Sample 1
    0x10B4, 0x10B4,  //Sample 2
    0x2120, 0x2120,  //Sample 3
    0x30FB, 0x30FB,  //Sample 4
    0x3FFF, 0x3FFF,  //Sample 5
    0x4DEB, 0x4DEB,  //Sample 6
    0x5A81, 0x5A81,  //Sample 7
    0x658B, 0x658B,  //Sample 8
    0x6ED9, 0x6ED9,  //Sample 9
    0x7640, 0x7640,  //Sample 10
    0x7BA2, 0x7BA2,  //Sample 11
    0x7EE6, 0x7EE6,  //Sample 12
    0x7FFF, 0x7FFF,  //Sample 13
    0x7FE6, 0x7FE6,  //Sample 14
    0x7BA2, 0x7BA2,  //Sample 15
    0x7640, 0x7640,  //Sample 16
    0x6ED9, 0x6ED9,  //Sample 17
    0x658B, 0x658B,  //Sample 18
    0x5A81, 0x5A81,  //Sample 19
    0x4DEB, 0x4DEB,  //Sample 20
    0x3FFF, 0x3FFF,  //Sample 21
    0x30FB, 0x30FB,  //Sample 22
    0x2120, 0x2120,  //Sample 23
    0x10B4, 0x10B4,  //Sample 24
    0x0000, 0x0000,  //Sample 25
    0xEF4C, 0xEF4C,  //Sample 26
    0xDEE0, 0xDEE0,  //Sample 27
    0xCF05, 0xCF05,  //Sample 28
    0xC001, 0xC001,  //Sample 29
    0xB215, 0xB215,  //Sample 30
```

```
    0xA57F, 0xA57F,  //Sample 31
    0x9A75, 0x9A75,  //Sample 32
    0x9127, 0x9127,  //Sample 33
    0x89C0, 0x89C0,  //Sample 34
    0x845E, 0x845E,  //Sample 35
    0x811A, 0x811A,  //Sample 36
    0x8001, 0x8001,  //Sample 37
    0x811A, 0x811A,  //Sample 38
    0x845E, 0x845E,  //Sample 39
    0x89C0, 0x89C0,  //Sample 40
    0x9127, 0x9127,  //Sample 41
    0x9A75, 0x9A75,  //Sample 42
    0xA57F, 0xA57F,  //Sample 43
    0xB215, 0xB215,  //Sample 44
    0xC001, 0xC001,  //Sample 45
    0xCF05, 0xCF05,  //Sample 46
    0xDEE0, 0xDEE0,  //Sample 47
    0xFF4C, 0xFF4C,  //Sample 48
};

/* Specify the Audio Stream format details that this application supports */
const APP_USB_HOST_AUDIO_STREAM_FORTMAT audioSpeakerStreamFormat =
{

    .streamDirection = USB_HOST_AUDIO_V1_0_DIRECTION_OUT,
    .format = USB_AUDIO_FORMAT_PCM,
    .nChannels = 2,
    .bitResolution = 16,
    .subFrameSize = 2,
    .samplingRate = 48000
};

bool isAudioDeviceAttached = false;
bool isStreamEnabled = false;
bool isAudioWriteCompleted = false;
USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj;
USB_HOST_AUDIO_V1_0_STREAM_TRANSFER_HANDLE transferHandleAudioWrite;



/*********************************************************
 * Audio Stream Event Handler function.
 *********************************************************/

USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE APP_USBHostAudioStreamEventHandler
(
    USB_HOST_AUDIO_V1_0_STREAM_HANDLE streamHandle,
    USB_HOST_AUDIO_V1_0_STREAM_EVENT event,
    void * eventData,
    uintptr_t context
)
{
    USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE_DATA * writeCompleteEventData;
    switch(event)
    {
        case USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE:

            break;

        case USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE:
            /* Check eventData result member to know if stream enable is complete */
                isStreamEnabled = true;

            break;
        case USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE:

             /* This means the Write request completed. We can
              * find out if the request was successful. */
            writeCompleteEventData =
```

```
                (USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE_DATA*)eventData;
            if(transferHandleAudioWrite == writeCompleteEventData->transferHandle)
            {
                isAudioWriteCompleted = true;
            }
            break;
        default:
            break;
    }
    return USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE_NONE;
}
/*********************************************************/
/* Function to search for a specific Audio Stream */
/*********************************************************/
USB_HOST_AUDIO_V1_0_STREAM_OBJ App_USBHostAudioSpeakerStreamFind
(
    USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj,
    APP_USB_HOST_AUDIO_STREAM_FORTMAT audioStream,
    uint8_t* numberofStreamGroups
)
{
    USB_HOST_AUDIO_V1_0_RESULT result;
    USB_HOST_AUDIO_V1_0_STREAM_INFO streamInfo;

    /* Get Number of Stream Groups */
    *numberofStreamGroups = USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet(audioDeviceObj);
    if (*numberofStreamGroups == 0)
    {
        return (USB_HOST_AUDIO_V1_0_STREAM_OBJ)0;
    }
    /* Get the First Stream Information in the Stream Group */
    result = USB_HOST_AUDIO_V1_0_StreamGetFirst(appData.audioDeviceObj, 0, &streamInfo);
    if (result == USB_HOST_AUDIO_V1_0_RESULT_SUCCESS)
    {
         /* Compare Audio Stream info */
        if ((streamInfo.format == audioStream.format)
            && (streamInfo.streamDirection == audioStream.streamDirection)
                    && (streamInfo.nChannels == audioStream.nChannels)
                    && (streamInfo.bitResolution == audioStream.bitResolution)
                    && (streamInfo.subFrameSize == audioStream.subFrameSize))
        {
            return streamInfo.streamObj;
        }
    }
    return (USB_HOST_AUDIO_V1_0_STREAM_OBJ)0;
}

/*********************************************************/
/* Audio attach event listener function */
/*********************************************************/
void APP_USBHostAudioAttachEventListener
(
    USB_HOST_AUDIO_V1_0_OBJ audioObj,
    USB_HOST_AUDIO_V1_0_EVENT event,
    uintptr_t context
)
{
    /* This function gets called when the Audio v1.0 device is attached/detached. In this
     * example we let the application know that a device is attached and we
     * store the Audio v1.0 device object. This object will be required to open the
     * device. */
    switch (event)
    {
        case USB_HOST_AUDIO_V1_0_EVENT_ATTACH:
            if (isAudioDeviceAttached == false)
            {
                isAudioDeviceAttached = true;
                audioDeviceObj = audioObj;
            }
```

```
            else
            {
                /* This application supports only one Audio Device . Handle Error Here.*/
            }
        break;
        case USB_HOST_AUDIO_V1_0_EVENT_DETACH:
            if (isAudioDeviceAttached == true)
            {
                /* This means the device was detached. There is no event data
                 * associated with this event.*/
                isAudioDeviceAttached = false;
                break;
            }
        break;
    }
}

/*******************************************************/
/* Audio Tasks function */
/*******************************************************/
void APP_Tasks ( void )
{
    USB_HOST_AUDIO_V1_0_RESULT audioResult;
    USB_HOST_AUDIO_V1_0_STREAM_RESULT streamResult;

    /* Check the application's current state. */
    switch ( appData.state )
    {
        case APP_STATE_BUS_ENABLE:

            /* Register a callback for Audio Device Attach. */
            audioResult = USB_HOST_AUDIO_V1_0_AttachEventHandlerSet
                          (
                              &APP_USBHostAudioAttachEventListener,
                              (uintptr_t)0
                          );

            if (audioResult == USB_HOST_AUDIO_V1_0_RESULT_SUCCESS )
            {
                /* Set Host Event Handler */
                USB_HOST_EventHandlerSet(APP_USBHostEventHandler, 0);
                USB_HOST_BusEnable(0);
                /* Advance application state */
                appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
            }
            break;

        case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:
            if(USB_HOST_BusIsEnabled(0))
            {
                appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            }
            break;

        case APP_STATE_WAIT_FOR_DEVICE_ATTACH:
            /* Check if an Audio Device has been attached  */
            if(appData.isAudioDeviceAttached == true)
            {
                appData.nAudioStreamGroups = 0;
                /* Find an Audio Stream matching to our requirement */
                appData.ouStreamObj = App_USBHostAudioSpeakerStreamFind
                                      (
                                          appData.audioDeviceObj,
                                          audioSpeakerStreamFormat,
                                          &appData.nAudioStreamGroups
                                      );
                if (appData.nAudioStreamGroups == 0)
                {
                    appData.state = APP_STATE_ERROR;
```

```
                break;
            }

            /* Open Audio Stream */
            appData.outStreamHandle = USB_HOST_AUDIO_V1_0_StreamOpen
                                      (
                                          appData.ouStreamObj
                                      );

            if (appData.outStreamHandle == USB_HOST_AUDIO_V1_0_STREAM_HANDLE_INVALID)
            {
                appData.state = APP_STATE_ERROR;
                break;
            }

            /* Set Stream Event Handler  */
            streamResult = USB_HOST_AUDIO_V1_0_StreamEventHandlerSet
                           (
                               appData.outStreamHandle,
                               APP_USBHostAudioStreamEventHandler,
                               (uintptr_t)appData.ouStreamObj
                           );

            if (streamResult != USB_HOST_AUDIO_V1_0_STREAM_SUCCESS)
            {
                appData.state = APP_STATE_ERROR;
                break;
            }
            appData.state = APP_STATE_ENABLE_AUDIO_STREAM;
        }
        break;

        case  APP_STATE_ENABLE_AUDIO_STREAM:
            isStreamEnableComplete = false;
             /* Set default interface setting of the streaming interface */
            streamResult = USB_HOST_AUDIO_V1_0_StreamEnable
                           (
                               appData.outStreamHandle,
                               &appData.requestHandle
                           );
            if (streamResult != USB_HOST_AUDIO_V1_0_STREAM_SUCCESS)
            {
                appData.state = APP_STATE_ERROR;
                break;
            }
            appData.state = APP_STATE_WAIT_FOR_ENABLE_AUDIO_STREAM;
        break;
     case  APP_STATE_WAIT_FOR_ENABLE_AUDIO_STREAM:
        if (isStreamEnabled == true)
        {
            appData.state = APP_STATE_START_STREAM_DATA;
        }
        break;
    case APP_STATE_START_STREAM_DATA:
        isAudioWriteCompleted = false;
        appData.state = APP_SATE_WAIT_FOR_WRITE_COMPLETE;
        USB_HOST_AUDIO_V1_0_StreamWrite
        (
            appData.outStreamHandle,
            &transferHandleAudioWrite,
            (void*)&audioSamples,
            192
        );
        break;

     case APP_SATE_WAIT_FOR_WRITE_COMPLETE:
        if (appData.isAudioWriteCompleted)
        {
            isAudioWriteCompleted = false;
```

```
                USB_HOST_AUDIO_V1_0_StreamWrite
                (
                    appData.outStreamHandle,
                    &transferHandleAudioWrite,
                    (void*)&audioSamples,
                    192
                );
            }
            break;

        default:
            break;
    }
}
```

Sending Class Specific Control Transfers

Describes how to send class-specific control transfers to the connected device, which includes a code example.

Description

The Audio v1.0 Host Client Driver allows the application client to send Audio v1.0 Class specific commands to the connected device. These commands can be send using USB_HOST_AUDIO_V1_0_ControlRequest function.

This function is non-blocking. The functions will return before the actual command execution is complete. The return value indicates if the command was scheduled successfully, or if the driver is busy and cannot accept commands, or if the command failed due to an unknown reason. If the command failed because the driver was busy, it can be retried. If scheduled successfully, the function will return a valid request handle. This request handle is unique and tracks the requested command.

When the command related control transfer has completed, the Audio v1.0 Host Client Driver generates a callback function. The call back function is one of the argument to the USB_HOST_AUDIO_V1_0_ControlRequest function.

The following code shows an example of sending a Audio v1.0 class specific commands.

```
/* This code shows an example for Audio Control transfer */
bool isAudioDeviceAttached = false;
USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj;


/*********************************************************/
/* Audio control request call back function   */
/*********************************************************/
void App_USBAudioControlRequestCallback
(
    USB_HOST_AUDIO_V1_0_OBJ audioObj,
    USB_HOST_AUDIO_V1_0_REQUEST_HANDLE requestHandle,
    USB_HOST_AUDIO_V1_0_RESULT result,
    size_t size,
    uintptr_t context
)
{
    APP_USB_AUDIO_CONTROL_TRANSFER_ACTION controlAction = (APP_USB_AUDIO_CONTROL_TRANSFER_ACTION)context;
    switch (controlAction)
    {
        case APP_USB_AUDIO_MASTER_UNMUTE_SET:
            if (result == USB_HOST_AUDIO_V1_0_RESULT_SUCCESS)
            {
                appData.isMasterUnmuteSetComplete = true;
            }
            else
            {
                appData.muteStatus = 1;
            }

        break;
        default:
            break;

    }
}

/*********************************************************/
```

```
/* Function for sending Mute control to Audio device.  */
/*********************************************************/
void APP_SendAudioMuteControl
(
    APP_USB_AUDIO_CONTROL_TRANSFER_ACTION action,
    uint32_t* mute
)
{
    USB_HOST_AUDIO_V1_0_RESULT result;
    USB_AUDIO_FEATURE_UNIT_CONTROL_REQUEST setupPacket;
    uint32_t status;


    /* Fill in Setup Packet */
    setupPacket.bmRequestType = (  USB_SETUP_DIRN_HOST_TO_DEVICE
                  | USB_SETUP_TYPE_CLASS
                  | USB_SETUP_RECIPIENT_INTERFACE
                 ); //interface , Host to device , Standard;
    setupPacket.bRequest = USB_AUDIO_CS_SET_CUR;
    if (action ==  APP_USB_AUDIO_MASTER_MUTE_SET)
    {
        setupPacket.channelNumber = APP_USB_AUDIO_CHANNEL_MASTER;
        status = __builtin_disable_interrupts();
        *mute = 1;
        __builtin_mtc0(12,0,status);
    }
    else if (action ==  APP_USB_AUDIO_MASTER_UNMUTE_SET)
    {
        setupPacket.channelNumber = APP_USB_AUDIO_CHANNEL_MASTER;
        status = __builtin_disable_interrupts();
        *mute = 0;
        __builtin_mtc0(12,0,status);
    }

    setupPacket.controlSelector = USB_AUDIO_MUTE_CONTROL;
    setupPacket.featureUnitId = 0x02; //appData.featureUnitDescriptor->bUnitID;
    setupPacket.wLength = 1;
    result = USB_HOST_AUDIO_V1_0_ControlRequest
             (
                appData.audioDeviceObj,
                &appData.requestHandle,
                (USB_SETUP_PACKET *)&setupPacket,
                mute,
                App_USBAudioControlRequestCallback,
                (uintptr_t)action
             );

}

/*********************************************************/
/* Audio attach event listener function */
/*********************************************************/
void APP_USBHostAudioAttachEventListener
(
    USB_HOST_AUDIO_V1_0_OBJ audioObj,
    USB_HOST_AUDIO_V1_0_EVENT event,
    uintptr_t context
)
{
    /* This function gets called when the Audio v1.0 device is attached/detached. In this
     * example we let the application know that a device is attached and we
     * store the Audio v1.0 device object. This object will be required to open the
     * device. */
    switch (event)
    {
        case USB_HOST_AUDIO_V1_0_EVENT_ATTACH:
            if (isAudioDeviceAttached == false)
            {
                isAudioDeviceAttached = true;
```

```
                audioDeviceObj = audioObj;
            }
            else
            {
                /* This application supports only one Audio Device . Handle Error Here.*/
            }
        break;
        case USB_HOST_AUDIO_V1_0_EVENT_DETACH:
            if (isAudioDeviceAttached == true)
            {
                /* This means the device was detached. There is no event data
                 * associated with this event.*/
                isAudioDeviceAttached = false;
                break;
            }
        break;
    }
}

/*******************************************************/
/* Audio Tasks function */
/*******************************************************/
void APP_Tasks ( void )
{
    switch (appData.state)
    {
        case APP_STATE_BUS_ENABLE:

            /* In this state the application enables the USB Host Bus. Note
             * how the Audio v1.0 Attach event handler is registered before the bus
             * is enabled. */

            USB_HOST_AUDIO_V1_0_AttachEventHandlerSet(APP_USBHostAudioAttachEventListener, (uintptr_t) 0);
            USB_HOST_BusEnable(0);
            appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
            break;

        case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:
            if(USB_HOST_BusIsEnabled(0) != true)
            {
                return;
            }
            /* Here we wait for the bus enable operation to complete. */
            /* Unmute the  Device */
            appData.isMasterUnmuteSetComplete = false;
            APP_SendAudioMuteControl
            (
                APP_USB_AUDIO_MASTER_UNMUTE_SET,
                (uint32_t*)&appData.muteStatus
            );
            appData.state = APP_STATE_AUDIO_WAIT_FOR_UNMUTE_COMPLETE;
            break;
        case APP_STATE_AUDIO_WAIT_FOR_UNMUTE_COMPLETE:
            if (appData.isMasterUnmuteSetComplete == true)
            {
                /* Audio Control request completed */
            }
    }
}
```

Configuring the Library

Describes how to configure the USB Audio v1.0 Host Client Driver.

Macros

	Name	Description
	USB_HOST_AUDIO_V1_ATTACH_LISTENERS_NUMBER	Defines the number of attach event listeners that can be registered with Audio v1.0 Host Client Driver.
	USB_HOST_AUDIO_V1_INSTANCES_NUMBER	Specifies the number of Audio v1.0 instances.
	USB_HOST_AUDIO_V1_STREAMING_INTERFACE_ALTERNATE_SETTINGS_NUMBER	Defines maximum number of alternate settings per Streaming interface provided by any Device that will be connected to this Audio Host.
	USB_HOST_AUDIO_V1_STREAMING_INTERFACES_NUMBER	Defines the maximum number of streaming interfaces could be present in an Audio v1.0 device that this Audio v1.0 Host Client Driver can support.

Description

The USB Audio v1.0 Host Client Driver requires configuration constants to be specified in `system_config.h` file. These constants define the build time configuration (functionality and static resources) of the USB Audio v1.0 Host Client Driver.

USB_HOST_AUDIO_V1_ATTACH_LISTENERS_NUMBER Macro

Defines the number of attach event listeners that can be registered with Audio v1.0 Host Client Driver.

File

usb_host_audio_v1_0_config_template.h

C

```
#define USB_HOST_AUDIO_V1_ATTACH_LISTENERS_NUMBER
```

Description

USB Host Audio v1.0 Attach Listeners Number

The USB Audio v1.0 Host Client Driver provides attach notification to listeners who have registered with the client driver via the USB_HOST_AUDIO_V1_0_AttachEventHandlerSet() function. The USB_HOST_AUDIO_V1_0_ATTACH_LISTENERS_NUMBER configuration constant defines the maximum number of event handlers that can be set. This number should be set to equal the number of entities that interested in knowing when a Audio v1.0 device is attached.

Remarks

None.

USB_HOST_AUDIO_V1_INSTANCES_NUMBER Macro

Specifies the number of Audio v1.0 instances.

File

usb_host_audio_v1_0_config_template.h

C

```
#define USB_HOST_AUDIO_V1_INSTANCES_NUMBER
```

Description

USB Host Audio v1.0 Maximum Number of Instances

This macro defines the number of instances of the Audio v1.0 host Driver. For example, if the application needs to implement two instances of the Audio v1.0 host Driver should be set to 2.

Remarks

None.

USB_HOST_AUDIO_V1_STREAMING_INTERFACE_ALTERNATE_SETTINGS_NUMBER Macro

Defines maximum number of alternate settings per Streaming interface provided by any Device that will be connected to this Audio Host.

File

usb_host_audio_v1_0_config_template.h

C

```
#define USB_HOST_AUDIO_V1_STREAMING_INTERFACE_ALTERNATE_SETTINGS_NUMBER
```

Description

USB Host Audio v1.0 Streaming interface alternate setting number

This configuration constant defines maximum number of Streaming interface alternate settings provided by any Device that will be connected to this Audio Host. The value of this constant should be at-least 1.

Remarks

Supporting multiple alternate settings per streaming interfaces requires more data memory and processing time.

Example

If the USB Audio v1.0 Host application must support a USB Audio Device with 2 alternate settings including Alternate Setting 0 then this constant should be defined to 3.

USB_HOST_AUDIO_V1_STREAMING_INTERFACES_NUMBER Macro

Defines the maximum number of streaming interfaces could be present in an Audio v1.0 device that this Audio v1.0 Host Client Driver can support.

File

usb_host_audio_v1_0_config_template.h

C

```
#define USB_HOST_AUDIO_V1_STREAMING_INTERFACES_NUMBER
```

Description

USB Host Audio v1.0 Streaming Interfaces Number

This configuration constant defines maximum number of streaming interfaces could be present in an Audio v1.0 device that this Audio v1.0 Host Client Driver can support. The value of this constant should be atleast 1.

Example 1 - If the USB Audio v1.0 Host application must support a USB Headset, this constant should be set 2 as an Audio Headset will have atleast 2 Audio Streaming interfaces, one for Host to Device streaming and one for Device to Host streaming.

Example 2 - If the USB Audio v1.0 Hos t application must support a USB Speaker, this constant should be set 1 as an Audio Speaker will have atleast 1 Audio Streaming interface.

Remarks

Supporting multiple streaming interfaces requires more data memory and processing time.

Building the Library

Describes the files to be included in the project while using the USB Audio v1.0 Host Client Driver.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/usb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
usb_host_audio_v1_0.h	This header file should be included in any `.c` file that accesses the USB Audio v1.0 Host Client Driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/usb_host_audio_v1_0.c	This file implements the USB Audio v1.0 Host Client Driver interface and should be included in the project if the USB Audio v1.0 Host Client Driver operation is desired.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The USB Audio v1.0 Host Client Driver Library depends on the following modules:

- USB Host Layer Library

Library Interface

a) Audio Device Access Functions

	Name	Description
	USB_HOST_AUDIO_V1_AttachEventHandlerSet	Sets an attach/detach event handler.
	USB_HOST_AUDIO_V1_0_ControlRequest	Schedules an Audio v1.0 control transfer.
	USB_HOST_AUDIO_V1_ControlEntityGetFirst	Retrieves the handle to the first audio control entity
	USB_HOST_AUDIO_V1_ControlEntityGetNext	Retrieves the handle to the next audio control entity.
	USB_HOST_AUDIO_V1_DeviceObjHandleGet	Returns the device object handle for this Audio v1.0 Device.
	USB_HOST_AUDIO_V1_EntityObjectGet	Retrieves the entity object for the entity ID.
	USB_HOST_AUDIO_V1_EntityRequestCallbackSet	Registers an audio entity request callback function with the Audio v1.0 Client Driver.
	USB_HOST_AUDIO_V1_EntityTypeGet	Returns the entity type of the audio control entity.
	USB_HOST_AUDIO_V1_FeatureUnitChannelMuteExists	Returns "true" if mute control exists for the specified channel of the feature unit.
	USB_HOST_AUDIO_V1_FeatureUnitChannelMuteGet	Schedules a get mute control request to the specified channel.
	USB_HOST_AUDIO_V1_FeatureUnitChannelMuteSet	Schedules a set mute control request to the specified channel.
	USB_HOST_AUDIO_V1_FeatureUnitChannelNumbersGet	Returns the number of channels.
	USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeExists	Returns "true" if volume control exists for the specified channel of the feature unit.
	USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeGet	Schedules a get current volume control request to the specified channel.
	USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeSet	Schedules a set current volume control request to the specified channel.
	USB_HOST_AUDIO_V1_FeatureUnitIDGet	Returns ID of the Feature Unit.
	USB_HOST_AUDIO_V1_FeatureUnitSourceIDGet	Returns the ID of the unit or terminal to which this feature unit is connected.
	USB_HOST_AUDIO_V1_TerminalAssociationGet	Returns the associated terminal ID of the audio control terminal.
	USB_HOST_AUDIO_V1_TerminalIDGet	Returns the terminal ID of the audio control entity.
	USB_HOST_AUDIO_V1_TerminalInputChannelNumbersGet	Returns the number of logical output channels in the terminal's output audio channel cluster.
	USB_HOST_AUDIO_V1_TerminalSourceIDGet	Returns the ID of the unit or terminal to which this terminal is connected.
	USB_HOST_AUDIO_V1_TerminalTypeGet	Returns the terminal type of the audio control entity.

b) Audio Stream Access Functions

	Name	Description
	USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet	Gets the number of stream groups present in the attached Audio v1.0 Device.
	USB_HOST_AUDIO_V1_StreamClose	Closes the audio stream.
	USB_HOST_AUDIO_V1_StreamEventHandlerSet	Registers an event handler with the Audio v1.0 Client Driver stream.

	USB_HOST_AUDIO_V1_0_StreamDisable	Schedules an audio stream disable request for the specified audio stream.
	USB_HOST_AUDIO_V1_StreamingInterfaceGetFirst	Gets the first streaming interface object from the attached Audio Device.
	USB_HOST_AUDIO_V1_0_StreamEnable	Schedules an audio stream enable request for the specified audio stream.
	USB_HOST_AUDIO_V1_StreamingInterfaceGetNext	Gets the next streaming interface object from the attached Audio Device.
	USB_HOST_AUDIO_V1_0_StreamEventHandlerSet	Registers an event handler with the Audio v1.0 Client Driver stream.
	USB_HOST_AUDIO_V1_StreamingInterfaceSet	Schedules a SET_INTERFACE request to the specified audio stream.
	USB_HOST_AUDIO_V1_0_StreamGetFirst	Returns information about first audio stream in the specified audio stream group.
	USB_HOST_AUDIO_V1_StreamingInterfaceSettingGetFirst	Gets the first streaming interface setting object within an audio streaming interface.
	USB_HOST_AUDIO_V1_0_StreamGetNext	Returns information about the next audio stream in the specified audio stream group.
	USB_HOST_AUDIO_V1_StreamingInterfaceSettingGetNext	Gets the next streaming interface setting object within an audio streaming interface.
	USB_HOST_AUDIO_V1_0_StreamSamplingRateSet	Schedules an audio stream set sampling rate request for the specified audio stream.
	USB_HOST_AUDIO_V1_StreamOpen	Opens the specified audio stream.
	USB_HOST_AUDIO_V1_StreamRead	Schedules an audio stream read request for the specified audio stream.
	USB_HOST_AUDIO_V1_StreamWrite	Schedules an audio stream write request for the specified audio stream.
	USB_HOST_AUDIO_V1_StreamingInterfaceBitResolutionGet	Returns the bit resolution of the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamingInterfaceChannelNumbersGet	Returns the number of channels of the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamingInterfaceDirectionGet	Returns the direction of the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamingInterfaceFormatTagGet	Returns the format tag of the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamingInterfaceSamplingFrequenciesGet	Returns the sampling frequencies supported by the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamingInterfaceSamplingFrequencyTypeGet	Returns the sampling frequency type of the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamingInterfaceSubFrameSizeGet	Returns the sub-frame size of the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamingInterfaceTerminalLinkGet	Returns the terminal link of the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamSamplingFrequencyGet	Schedules an audio stream get sampling rate request for the specified audio stream.
	USB_HOST_AUDIO_V1_StreamSamplingFrequencySet	Schedules an audio stream set sampling rate request for the specified audio stream.
	USB_HOST_AUDIO_V1_0_StreamRead	Schedules an audio stream read request for the specified audio stream.
	USB_HOST_AUDIO_V1_0_StreamWrite	Schedules an audio stream write request for the specified audio stream.

c) Other Functions

	Name	Description
	USB_HOST_AUDIO_V1_TerminalInputChannelConfigGet	Returns a structure that describes the spatial location of the logical channels of in the terminal's output audio channel cluster.
	USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeRangeGet	Schedules a control request to the Audio Device feature unit to get the range supported by the volume control on the specified channel.
	USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeSubRangeNumbersGet	Schedules a control request to an Audio Device feature unit to get the number of sub-ranges supported by the volume control on the specified channel.

d) Data Types and Constants

	Name	Description
	USB_HOST_AUDIO_V1_ATTACH_EVENT_HANDLER	USB Host Audio v1.0 Client Driver attach event handler function pointer type.
	USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ	Defines the type of the Audio v1.0 Host control entity object.
	USB_HOST_AUDIO_V1_ENTITY_REQUEST_CALLBACK	USB Host Audio v1.0 class driver control transfer complete callback function pointer type.
	USB_HOST_AUDIO_V1_EVENT	Identifies the possible events that the Audio v1.0 Class Driver attach event handler can generate.
	USB_HOST_AUDIO_V1_OBJ	Defines the type of the Audio v1.0 Host client object.
	USB_HOST_AUDIO_V1_REQUEST_HANDLE	USB Host Audio v1.0 Client Driver request handle.
	USB_HOST_AUDIO_V1_RESULT	USB Host Audio v1.0 Class Driver result enumeration.
	USB_HOST_AUDIO_V1_0_ATTACH_EVENT_HANDLER	USB Host Audio v1.0 Client Driver attach event handler function pointer type.
	USB_HOST_AUDIO_V1_STREAM_DIRECTION	USB Host Audio v1.0 Class Driver stream direction.
	USB_HOST_AUDIO_V1_STREAM_EVENT	Identifies the possible events that the Audio v1.0 Stream can generate.
	USB_HOST_AUDIO_V1_STREAM_EVENT_HANDLER	USB Host Audio v1.0 Class Driver stream event handler function pointer type.
	USB_HOST_AUDIO_V1_STREAM_EVENT_INTERFACE_SET_COMPLETE_DATA	USB Host Audio v1.0 class stream control event data.
	USB_HOST_AUDIO_V1_0_CONTROL_CALLBACK	USB Host Audio v1.0 Class Driver control transfer complete callback function pointer type.
	USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE_DATA	USB Host Audio v1.0 class stream data transfer event data.
	USB_HOST_AUDIO_V1_0_EVENT	Identifies the possible events that the Audio v1.0 Class Driver can generate.
	USB_HOST_AUDIO_V1_STREAM_EVENT_RESPONSE	Returns the type of the USB Host Audio v1.0 stream event handler.
	USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE_DATA	USB Host Audio v1.0 class stream control event data.
	USB_HOST_AUDIO_V1_0_OBJ	Defines the type of the Audio v1.0 Host client object.
	USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE_DATA	USB Host Audio v1.0 class stream data transfer event data.
	USB_HOST_AUDIO_V1_0_REQUEST_HANDLE	USB Host Audio v1.0 Client Driver request handle.
	USB_HOST_AUDIO_V1_STREAM_HANDLE	Defines the type of the Audio v1.0 Host stream handle.
	USB_HOST_AUDIO_V1_0_RESULT	USB Host Audio v1.0 Class Driver audio result enumeration.
	USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE	USB Host Audio v1.0 Class Driver stream data transfer handle.
	USB_HOST_AUDIO_V1_0_STREAM_DIRECTION	USB Host Audio v1.0 Class Driver stream direction.
	USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ	Defines the type of the Audio v1.0 Host streaming interface object.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT	Identifies the possible events that the Audio v1.0 stream can generate.
	USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ	Defines the type of the Audio v1.0 Host streaming interface setting object.

	USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE_DATA	USB Host Audio v1.0 class stream control event data.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE_DATA	USB Host Audio v1.0 class stream control event data.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT_HANDLER	USB Host Audio v1.0 Class Driver stream event handler function pointer type.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE_DATA	This is macro USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE_DATA.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE	Returns the type of the USB Audio v1.0 Host Client Driver event handler.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE_DATA	USB Host Audio v1.0 class stream transfer event data.
	USB_HOST_AUDIO_V1_0_STREAM_HANDLE	Defines the type of the Audio v1.0 Host stream handle.
	USB_HOST_AUDIO_V1_INTERFACE	USB HOST Audio v1.0 Client Driver interface.
	USB_HOST_AUDIO_V1_0_STREAM_INFO	This is type USB_HOST_AUDIO_V1_0_STREAM_INFO.
	USB_HOST_AUDIO_V1_REQUEST_HANDLE_INVALID	USB Host Audio v1.0 Client Driver invalid request handle.
	USB_HOST_AUDIO_V1_0_STREAM_OBJ	Defines the type of the Audio v1.0 Host stream object.
	USB_HOST_AUDIO_V1_STREAM_HANDLE_INVALID	Defines Audio v1.0 Host stream invalid handle.
	USB_HOST_AUDIO_V1_0_STREAM_RESULT	USB Host Audio v1.0 stream result enumeration.
	USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE_INVALID	USB Host Audio v1.0 Class Driver invalid stream data transfer handle.
	USB_HOST_AUDIO_V1_0_STREAM_TRANSFER_HANDLE	USB Host Audio v1.0 Class Driver transfer handle.
	USB_HOST_AUDIO_V1_0_INTERFACE	USB HOST Audio Client Driver interface.
	USB_HOST_AUDIO_V1_0_REQUEST_HANDLE_INVALID	USB Host Audio v1.0 Client Driver invalid request handle.
	USB_HOST_AUDIO_V1_0_STREAM_HANDLE_INVALID	Defines the type of the Audio v1.0 Host stream invalid handle.
	USB_HOST_AUDIO_V1_0_STREAM_TRANSFER_HANDLE_INVALID	USB Host Audio v1.0 Class Driver invalid transfer handle definition.
	USB_HOST_AUDIO_V1_0_AttachEventHandlerSet	Sets an attach/detach event handler.
	USB_HOST_AUDIO_V1_0_DeviceObjHandleGet	Returns the device object handle for this Audio v1.0 Device.
	USB_HOST_AUDIO_V1_0_DIRECTION_IN	This is macro USB_HOST_AUDIO_V1_0_DIRECTION_IN.
	USB_HOST_AUDIO_V1_0_DIRECTION_OUT	This is macro USB_HOST_AUDIO_V1_0_DIRECTION_OUT.
	USB_HOST_AUDIO_V1_0_EVENT_ATTACH	This is macro USB_HOST_AUDIO_V1_0_EVENT_ATTACH.
	USB_HOST_AUDIO_V1_0_EVENT_DETACH	This is macro USB_HOST_AUDIO_V1_0_EVENT_DETACH.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE_NONE	Returns the type of the USB Host Audio v1.0 stream event handler.
	USB_HOST_AUDIO_V1_0_StreamClose	Closes the audio stream.
	USB_HOST_AUDIO_V1_0_StreamOpen	Opens the specified audio stream.
	USB_HOST_AUDIO_V1_SAMPLING_FREQUENCIES_NUMBER	This structure defines USB Host audio stream information structure.
	USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_GET_COMPLETE_DATA	USB Host Audio v1.0 class stream control event data.

Description

This section describes the Application Programming Interface (API) functions of the USB Audio v1.0 Host Client Driver Library.

Refer to each section for a detailed description.

a) Audio Device Access Functions

USB_HOST_AUDIO_V1_AttachEventHandlerSet Function

Sets an attach/detach event handler.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_AttachEventHandlerSet(USB_HOST_AUDIO_V1_ATTACH_EVENT_HANDLER
eventHandler, uintptr_t context);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - If the attach event handler was registered successfully
- USB_HOST_AUDIO_V1_RESULT_FAILURE - If the number of registered event handlers has exceeded USB_HOST_AUDIO_V1_ATTACH_LISTENERS_NUMBER

Description

This function will set an attach event handler. The attach event handler will be called when a Audio v1.0 Device has been attached or detached. The context will be returned in the event handler. This function should be called before the bus has been enabled.

Remarks

This function should be called before the USB_HOST_BusEnable function is called.

Preconditions

None.

Parameters

Parameters	Description
eventHandler	Pointer to the attach event handler.
context	An application defined context that will be returned in the event handler.

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_AttachEventHandlerSet
(
    USB_HOST_AUDIO_V1_ATTACH_EVENT_HANDLER eventHandler,
uintptr_t context
);
```

USB_HOST_AUDIO_V1_0_ControlRequest Function

Schedules an Audio v1.0 control transfer.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_0_RESULT USB_HOST_AUDIO_V1_0_ControlRequest(USB_HOST_AUDIO_V1_0_OBJ OBJ,
USB_HOST_AUDIO_V1_0_REQUEST_HANDLE * transferHandle, USB_SETUP_PACKET * setupPacket, void * data,
USB_HOST_AUDIO_V1_0_CONTROL_CALLBACK callback, uintptr_t context);
```

Returns

- USB_HOST_AUDIO_V1_0_RESULT_SUCCESS - The transfer was scheduled successfully. requestHandle will contain a valid transfer handle.
- USB_HOST_AUDIO_V1_0_RESULT_FAILURE - An unknown failure occurred. requestHandle will contain USB_HOST_AUDIO_V1_0_REQUEST_HANDLE_INVALID.

- USB_HOST_AUDIO_V1_0_RESULT_PARAMETER_INVALID - The data pointer or requestHandle pointer is NULL

Description

This function schedules an Audio v1.0 control transfer. The audioObj parameter is an object of the Audio v1.0 Class Driver to which the audio control transfer is to be scheduled. The setupPacket parameter points to the SETUP command to be sent in the setup state of the control transfer. The size and the direction of the data stage is indicated by the SETUP packet. For control transfers where there is no data stage, data is ignored and can be NULL. In all other instances, data should point to the data to data be transferred in the data stage of the control transfer.

If the transfer was scheduled successfully, requestHandle will contain a transfer handle that uniquely identifies this transfer. If the transfer could not be scheduled successfully, requestHandle will contain USB_HOST_AUDIO_V1_0_REQUEST_HANDLE_INVALID.

When the control transfer completes, the Audio v1.0 Client Driver will call the specified callback function. The context parameter specified here will be returned in the callback.

Remarks

None.

Preconditions

The Audio v1.0 Device should be attached.

Parameters

Parameters	Description
audioObj	Audio v1.0 client driver object
requestHandle	Output parameter that will contain the handle to this transfer
setupPacket	Pointer to the SETUP packet to sent to the device in the SETUP stage of the control transfer
data	For control transfer with a data stage, this should point to data to be sent to the device (for a control write transfer) or point to the buffer that will receive data from the device (for a control read transfer). For control transfers that do not require a data stage, this parameter is ignored and can be NULL.
callback	Pointer to the callback function that will be called when the control transfer completes. If the callback function is NULL, there will be no notification of when the control transfer will complete.
context	User-defined context that is returned with the callback function

Function

```
USB_HOST_AUDIO_V1_0_RESULT USB_HOST_AUDIO_V1_0_ControlRequest
(
    USB_HOST_AUDIO_V1_0_OBJ audioObj,
    USB_HOST_AUDIO_V1_0_REQUEST_HANDLE * requestHandle,
USB_SETUP_PACKET *setupPacket,
void * data,
    USB_HOST_AUDIO_V1_0_CONTROL_CALLBACK callback,
uintptr_t context
);
```

USB_HOST_AUDIO_V1_ControlEntityGetFirst Function

Retrieves the handle to the first audio control entity

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_ControlEntityGetFirst(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ * pEntityObject);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_RESULT_END_OF_CONTROL_ENTITY - No more audio control entities are available
- USB_HOST_AUDIO_V1_RESULT_OBJ_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred

Description

This function retrieves the handle to the first audio control entity.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 device object.
pEntityObject	pointer to the Audio control entity handle.

Function

USB_HOST_AUDIO_V1_REESULT USB_HOST_AUDIO_V1_ControlEntityGetFirst

(

USB_HOST_AUDIO_V1_OBJ audioObj,

USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ * pEntityObject

);

USB_HOST_AUDIO_V1_ControlEntityGetNext Function

Retrieves the handle to the next audio control entity.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_ControlEntityGetNext(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObjectCurrent, USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ *
pEntityObject);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_RESULT_END_OF_CONTROL_ENTITY - No more audio control entities are available
- USB_HOST_AUDIO_V1_RESULT_OBJ_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred

Description

This function retrieves the handle to the next audio control entity.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 device object.
entityObjectCurrent	Handle to current audio control entity.
pEntityObject	pointer to audio control entity handle.

Function

USB_HOST_AUDIO_V1_REESULT USB_HOST_AUDIO_V1_ControlEntityGetNext

(

USB_HOST_AUDIO_V1_OBJ audioObj,

USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObjectCurrent

USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ * pEntityObject
);

USB_HOST_AUDIO_V1_DeviceObjHandleGet Function

Returns the device object handle for this Audio v1.0 Device.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_DEVICE_OBJ_HANDLE USB_HOST_AUDIO_V1_DeviceObjHandleGet(USB_HOST_AUDIO_V1_OBJ audioDeviceObj);
```

Returns

Will return a valid device object handle if the device is still connected to the system. Otherwise, the function will return USB_HOST_DEVICE_OBJ_HANDLE_INVALID.

Description

This function returns the device object handle for this Audio v1.0 Device. This returned handle can be used by the application to perform device-level operations, such as obtaining the string descriptors.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
audioDeviceObj	Audio V1.0 device object handle returned in the USB_HOST_AUDIO_V1_ATTACH_EVENT_HANDLER function.

Function

USB_HOST_DEVICE_OBJ_HANDLE USB_HOST_AUDIO_V1_DeviceObjHandleGet
(
USB_HOST_AUDIO_V1_OBJ audioDeviceObj
);

USB_HOST_AUDIO_V1_EntityObjectGet Function

Retrieves the entity object for the entity ID.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_EntityObjectGet(USB_HOST_AUDIO_V1_OBJ audioObj, uint8_t
entityId, USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ* entityObj);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_RESULT_FAILURE - The entity Id could not be found or an unknown failure occurred

Description

This function retrieves the entity object for the entity ID.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityId	Entity ID
entityObject	Audio control entity object

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_EntityObjectGet
(
    USB_HOST_AUDIO_V1_OBJ  audioObj,
uint8_t entityId,
    USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ* entityObj
);
```

USB_HOST_AUDIO_V1_EntityRequestCallbackSet Function

Registers an audio entity request callback function with the Audio v1.0 Client Driver.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_EntityRequestCallbackSet(USB_HOST_AUDIO_V1_OBJ audioDeviceObj,
USB_HOST_AUDIO_V1_ENTITY_REQUEST_CALLBACK appAudioEntityRequestCallback, uintptr_t context);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_RESULT_OBJ_INVALID - The specified audio object does not exist
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred

Description

This function registers a callback function for the Audio v1.0 control entity requests. The Audio v1.0 Host Client Driver will call this callback function when an audio entity control request is completed.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
audioDeviceObj	Audio v1.0 device object.
appAudioEntityRequestCallback	A pointer to event handler function. If NULL, events will not be generated.
context	Application specific context that is returned in the event handler.

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_EntityRequestCallbackSet
(
    USB_HOST_AUDIO_V1_OBJ audioDeviceObj,
USB_HOST_AUDIO_V1_CONTROL_EVENT_HANDLER appAudioEntityRequestCallback,
uintptr_t context
);
```

USB_HOST_AUDIO_V1_EntityTypeGet Function

Returns the entity type of the audio control entity.

File

usb_host_audio_v1_0.h

C

```
USB_AUDIO_V1_ENTITY_TYPE USB_HOST_AUDIO_V1_EntityTypeGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject);
```

Returns

USB_AUDIO_V1_ENTITY_TYPE.

Description

This function returns the entity type of the audio control entity. Prior to calling this function the entity object should be obtained by calling USB_HOST_AUDIO_V1_ControlEntityGetFirst, USB_HOST_AUDIO_V1_ControlEntityGetNext, or USB_HOST_AUDIO_V1_EntityObjectGet.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object

Function

USB_AUDIO_V1_ENTITY_TYPE USB_HOST_AUDIO_V1_EntityTypeGet

(

USB_HOST_AUDIO_V1_OBJ audioObj,

USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject

);

USB_HOST_AUDIO_V1_FeatureUnitChannelMuteExists Function

Returns "true" if mute control exists for the specified channel of the feature unit.

File

usb_host_audio_v1_0.h

C

```
bool USB_HOST_AUDIO_V1_FeatureUnitChannelMuteExists(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject, uint8_t channel);
```

Returns

- true - Mute control exists on the specified channel
- false - Mute control does not exist on the specified channel

Description

This function returns "true" if mute control exists on the specified channel of the feature unit. Channel 0 indicates Master mute control. This function is only applicable to a feature unit. Prior to calling this function the entity object should be obtained by calling USB_HOST_AUDIO_V1_ControlEntityGetFirst, USB_HOST_AUDIO_V1_ControlEntityGetNext, or USB_HOST_AUDIO_V1_EntityObjectGet.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object
channel	Channel number

Function

bool USB_HOST_AUDIO_V1_FeatureUnitChannelMuteExists

(
USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject,
uint8_t channel
);

USB_HOST_AUDIO_V1_FeatureUnitChannelMuteGet Function

Schedules a get mute control request to the specified channel.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_FeatureUnitChannelMuteGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject, USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle,
uint8_t channelNumber, bool * muteStatus);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The request was scheduled successfully. requestHandle will contain a valid request handle.
- USB_HOST_AUDIO_V1_RESULT_BUSY - The control request mechanism is currently busy. Retry the request.
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred. requestHandle will contain USB_HOST_AUDIO_V1_0_REQUEST_HANDLE_INVALID.
- USB_HOST_AUDIO_V1_RESULT_PARAMETER_INVALID - The data pointer or requestHandle pointer is NULL

Description

This function schedules a get mute control request to the specified channel. Prior to calling this function the user should check if mute control exists on the specified channel by calling the USB_HOST_AUDIO_V1_FeatureUnitChannelMuteExists function.

If the request was scheduled successfully, the requestHandle parameter will contain a request handle that uniquely identifies this request. If the transfer could not be scheduled successfully, requestHandle will contain USB_HOST_AUDIO_V1_REQUEST_HANDLE_INVALID.

When the control request completes, the Audio v1.0 Client Driver will call the callback function that was set using the USB_HOST_AUDIO_V1_EntityRequestCallbackSet function. The context parameter specified here will be returned in the callback.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object
requestHandle	Output parameter that will contain the handle to this request
channelNumber	Channel number
muteStatus	Output parameter that will contain Current Mute status when the request is completed and a callback is received

Function

USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_FeatureUnitChannelMuteGet
(
USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject,
USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle,
uint8_t channelNumber,
bool *muteStatus
);

USB_HOST_AUDIO_V1_FeatureUnitChannelMuteSet Function

Schedules a set mute control request to the specified channel.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_FeatureUnitChannelMuteSet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject, USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle,
uint8_t channelNumber, bool * muteStatus);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The request was scheduled successfully. requestHandle will contain a valid request handle.
- USB_HOST_AUDIO_V1_RESULT_BUSY - The control request mechanism is currently busy. Retry the request.
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred. requestHandle will contain USB_HOST_AUDIO_V1_0_REQUEST_HANDLE_INVALID.
- USB_HOST_AUDIO_V1_RESULT_PARAMETER_INVALID - The data pointer or requestHandle pointer is NULL

Description

This function schedules a set mute control request to the specified channel. Prior to calling this function the user should check if mute control exists on the specified channel by calling the USB_HOST_AUDIO_V1_FeatureUnitChannelMuteExists function.

If the request was scheduled successfully, the requestHandle parameter will contain a request handle that uniquely identifies this transfer. If the transfer could not be scheduled successfully, requestHandle will contain USB_HOST_AUDIO_V1_REQUEST_HANDLE_INVALID.

When the control request completes, the Audio v1.0 Client Driver will call the callback function that was set using the USB_HOST_AUDIO_V1_EntityRequestCallbackSet function. The context parameter specified here will be returned in the callback.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object
requestHandle	Output parameter that will contain the handle to this request
channelNumber	Channel Number
muteStatus	Value of mute control, where 1 mutes the channel and 0 removes unmutes

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_FeatureUnitChannelMuteSet
(
    USB_HOST_AUDIO_V1_OBJ  audioObj,
    USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject,
    USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle,
uint8_t channelNumber,
bool *muteStatus
);
```

USB_HOST_AUDIO_V1_FeatureUnitChannelNumbersGet Function

Returns the number of channels.

File

usb_host_audio_v1_0.h

C

```
uint8_t USB_HOST_AUDIO_V1_FeatureUnitChannelNumbersGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject);
```

Returns

The number of channels.

Description

This function returns the number of channels. This function is only applicable to a feature unit. Prior to calling this function the entity object should

be obtained by calling USB_HOST_AUDIO_V1_ControlEntityGetFirst, USB_HOST_AUDIO_V1_ControlEntityGetNext, or USB_HOST_AUDIO_V1_EntityObjectGet.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object

Function

uint8_t USB_HOST_AUDIO_V1_FeatureUnitChannelNumbersGet

(

USB_HOST_AUDIO_V1_OBJ audioObj,

USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject

);

USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeExists Function

Returns "true" if volume control exists for the specified channel of the feature unit.

File

usb_host_audio_v1_0.h

C

```
bool USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeExists(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject, uint8_t channel);
```

Returns

- true - Volume control exists on the specified channel
- false - Volume control does not exist on the specified channel

Description

This function returns "true" if volume control exists on the specified channel of the feature unit. Channel 0 indicates master volume control. This function is only applicable to a feature unit. Prior to calling this function the entity object should be obtained by calling USB_HOST_AUDIO_V1_ControlEntityGetFirst, USB_HOST_AUDIO_V1_ControlEntityGetNext, or USB_HOST_AUDIO_V1_EntityObjectGet.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object
channel	Channel number

Function

bool USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeExists

(

USB_HOST_AUDIO_V1_OBJ audioObj,

USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject,

uint8_t channel

);

USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeGet Function

Schedules a get current volume control request to the specified channel.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject, USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle,
uint8_t channelNumber, uint16_t * volume);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The request was scheduled successfully. requestHandle will contain a valid request handle.
- USB_HOST_AUDIO_V1_RESULT_BUSY - The control request mechanism is currently busy. Retry the request.
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred. requestHandle will contain USB_HOST_AUDIO_V1_0_REQUEST_HANDLE_INVALID
- USB_HOST_AUDIO_V1_RESULT_PARAMETER_INVALID - The data pointer or requestHandle pointer is NULL

Description

This function schedules a get current volume control request to the specified channel. Prior to calling this function the user should check if volume control exists on the specified channel by calling the USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeExists function.

If the request was scheduled successfully, the requestHandle parameter will contain a request handle that uniquely identifies this request. If the request could not be scheduled successfully, requestHandle will contain USB_HOST_AUDIO_V1_REQUEST_HANDLE_INVALID.

When the control request completes, the Audio v1.0 Client Driver will call the callback function that was set using the USB_HOST_AUDIO_V1_EntityRequestCallbackSet function. The context parameter specified here will be returned in the callback.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object
requestHandle	Output parameter that will contain the handle to this request
channelNumber	Channel number to which the volume control is addressed
volume	Output parameter that will contain the current volume when a request is completed and a callback is received

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeGet
(
    USB_HOST_AUDIO_V1_OBJ  audioObj,
    USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject,
    USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle,
uint8_t channelNumber,
uint16_t *volume
);
```

USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeSet Function

Schedules a set current volume control request to the specified channel.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeSet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject, USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle,
uint8_t channelNumber, uint16_t * volume);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The request was scheduled successfully. requestHandle will contain a valid request handle.
- USB_HOST_AUDIO_V1_RESULT_BUSY - The control request mechanism is currently busy. Retry the request.

- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred. requestHandle will contain USB_HOST_AUDIO_V1_0_REQUEST_HANDLE_INVALID.
- USB_HOST_AUDIO_V1_RESULT_PARAMETER_INVALID - The data pointer or requestHandle pointer is NULL

Description

This function schedules a set current volume request to the specified channel. Prior to calling this function the user should check if volume control exists on the specified channel by calling the USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeExists function.

If the request was scheduled successfully, the requestHandle parameter will contain a request handle that uniquely identifies this request. If the request could not be scheduled successfully, requestHandle will contain USB_HOST_AUDIO_V1_REQUEST_HANDLE_INVALID.

When the control request completes, the Audio v1.0 Client Driver will call the callback function that was set using the USB_HOST_AUDIO_V1_EntityRequestCallbackSet function. The context parameter specified here will be returned in the callback.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object
requestHandle	Output parameter that will contain the handle to this request
channelNumber	Channel number to which the volume control is addressed
volume	Current volume control value that should be set in the Audio Device

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeSet
(
    USB_HOST_AUDIO_V1_OBJ  audioObj,
    USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject,
    USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle,
uint8_t channelNumber,
uint16_t *volume
);
```

USB_HOST_AUDIO_V1_FeatureUnitIDGet Function

Returns ID of the Feature Unit.

File

usb_host_audio_v1_0.h

C

```
uint8_t USB_HOST_AUDIO_V1_FeatureUnitIDGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject);
```

Returns

The ID of the feature unit.

Description

This function returns the ID of the D of the Feature Unit. This function is only applicable to Feature Unit. Prior to calling this function Entity Object should be obtained by calling the USB_HOST_AUDIO_V1_ControlEntityGetFirst, USB_HOST_AUDIO_V1_ControlEntityGetNext, or USB_HOST_AUDIO_V1_EntityObjectGet function.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 device object.
entityObject	Audio control entity Object

Function

uint8_t USB_HOST_AUDIO_V1_FeatureUnitIDGet
(
USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject
);

USB_HOST_AUDIO_V1_FeatureUnitSourceIDGet Function

Returns the ID of the unit or terminal to which this feature unit is connected.

File

usb_host_audio_v1_0.h

C

```
uint8_t USB_HOST_AUDIO_V1_FeatureUnitSourceIDGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject);
```

Returns

The ID of the unit or terminal to which this feature unit is connected.

Description

This function returns the ID of the Unit or Terminal to which this feature unit is connected. This function is only applicable to a feature unit. Prior to calling this function the entity object should be obtained by calling USB_HOST_AUDIO_V1_ControlEntityGetFirst, USB_HOST_AUDIO_V1_ControlEntityGetNext, or USB_HOST_AUDIO_V1_EntityObjectGet.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object

Function

uint8_t USB_HOST_AUDIO_V1_FeatureUnitSourceIDGet
(
USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject
);

USB_HOST_AUDIO_V1_TerminalAssociationGet Function

Returns the associated terminal ID of the audio control terminal.

File

usb_host_audio_v1_0.h

C

```
uint8_t USB_HOST_AUDIO_V1_TerminalAssociationGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject);
```

Returns

The ID of the associated terminal.

Description

This function returns the ID of the associated terminal type of the audio control terminal. Prior to calling this function the entity object should be obtained by calling USB_HOST_AUDIO_V1_ControlEntityGetFirst, USB_HOST_AUDIO_V1_ControlEntityGetNext, or USB_HOST_AUDIO_V1_EntityObjectGet.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object

Function

uint8_t USB_HOST_AUDIO_V1_TerminalAssociationGet
(
    USB_HOST_AUDIO_V1_OBJ audioObj,
    USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject
);

USB_HOST_AUDIO_V1_TerminalIDGet Function

Returns the terminal ID of the audio control entity.

File

usb_host_audio_v1_0.h

C

```
uint8_t USB_HOST_AUDIO_V1_TerminalIDGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject);
```

Returns

The terminal ID of the audio control entity object.

Description

This function returns the Terminal ID of the Audio Control entity. Prior to calling this function the entity object should be obtained by calling USB_HOST_AUDIO_V1_ControlEntityGetFirst, USB_HOST_AUDIO_V1_ControlEntityGetNext, or USB_HOST_AUDIO_V1_EntityObjectGet.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object

Function

uint8_t USB_HOST_AUDIO_V1_TerminalIDGet
(
    USB_HOST_AUDIO_V1_OBJ audioObj,
    USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject
);

USB_HOST_AUDIO_V1_TerminalInputChannelNumbersGet Function

Returns the number of logical output channels in the terminal's output audio channel cluster.

File

usb_host_audio_v1_0.h

C

```
uint8_t USB_HOST_AUDIO_V1_TerminalInputChannelNumbersGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject);
```

Returns

The number of logical output channels in the terminal's output audio channel cluster.

Description

This function returns the number of logical output channels in the terminal's output audio channel cluster. This function is only applicable to an input terminal. Prior to calling this function the entity object should be obtained by calling USB_HOST_AUDIO_V1_ControlEntityGetFirst, USB_HOST_AUDIO_V1_ControlEntityGetNext, or USB_HOST_AUDIO_V1_EntityObjectGet.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 device object.
entityObject	Audio control entity object

Function

uint8_t USB_HOST_AUDIO_V1_TerminalInputChannelNumbersGet

(

USB_HOST_AUDIO_V1_OBJ audioObj,

USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject

);

USB_HOST_AUDIO_V1_TerminalSourceIDGet Function

Returns the ID of the unit or terminal to which this terminal is connected.

File

usb_host_audio_v1_0.h

C

```
uint8_t USB_HOST_AUDIO_V1_TerminalSourceIDGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject);
```

Returns

The ID of the unit or terminal to which this terminal is connected.

Description

This function returns the ID of the unit or terminal to which this terminal is connected. This function is only applicable to an output terminal. Prior to calling this function the entity object should be obtained by calling USB_HOST_AUDIO_V1_ControlEntityGetFirst, USB_HOST_AUDIO_V1_ControlEntityGetNext, or USB_HOST_AUDIO_V1_EntityObjectGet.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object

Function

uint8_t USB_HOST_AUDIO_V1_TerminalSourceIDGet

(

USB_HOST_AUDIO_V1_OBJ audioObj,

USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject

);

USB_HOST_AUDIO_V1_TerminalTypeGet Function

Returns the terminal type of the audio control entity.

File

usb_host_audio_v1_0.h

C

```
USB_AUDIO_V1_TERMINAL_TYPE USB_HOST_AUDIO_V1_TerminalTypeGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject);
```

Returns

The terminal type.

Description

This function returns the Terminal type of the audio control entity. Prior to calling this function Entity Object should be obtained by calling the USB_HOST_AUDIO_V1_ControlEntityGetFirst, USB_HOST_AUDIO_V1_ControlEntityGetNext, or USB_HOST_AUDIO_V1_EntityObjectGet function.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 device object
entityObject	Audio control entity Object

Function

USB_AUDIO_V1_TERMINAL_TYPE USB_HOST_AUDIO_V1_TerminalTypeGet

(

USB_HOST_AUDIO_V1_OBJ audioObj,

USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject

);

b) Audio Stream Access Functions

USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet Function

Gets the number of stream groups present in the attached Audio v1.0 Device.

File

usb_host_audio_v1_0.h

C

```
uint8_t USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet(USB_HOST_AUDIO_V1_0_OBJ audioObj);
```

Returns

A returned uint8_t indicates the number of audio stream groups present in the attached Audio v1.0 Device.

Description

This function will get number of stream groups present in the attached Audio v1.0 Device. The audio stream within an audio stream cannot be enabled at the same time.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached.

Parameters

Parameters	Description
audioObj	Audio v1.0 Client Driver object

Function

uint8_t USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet
(
    USB_HOST_AUDIO_V1_0_OBJ audioObj
);

USB_HOST_AUDIO_V1_StreamClose Function

Closes the audio stream.

File

usb_host_audio_v1_0.h

C

```
void USB_HOST_AUDIO_V1_StreamClose(USB_HOST_AUDIO_V1_STREAM_HANDLE audioStreamHandle);
```

Returns

None.

Description

This function will close the open audio stream. This closes the association between the application entity that opened the audio stream and the audio stream. The audio stream handle becomes invalid.

Remarks

The device handle becomes invalid after calling this function.

Preconditions

None.

Parameters

Parameters	Description
audioSteamHandle	handle to the audio stream obtained from the USB_HOST_AUDIO_V1_StreamOpen function.

Function

void USB_HOST_AUDIO_V1_StreamClose
(
    USB_HOST_AUDIO_V1_STREAM_HANDLE audioSteamHandle
);

USB_HOST_AUDIO_V1_StreamEventHandlerSet Function

Registers an event handler with the Audio v1.0 Client Driver stream.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamEventHandlerSet(USB_HOST_AUDIO_V1_STREAM_HANDLE handle,
USB_HOST_AUDIO_V1_STREAM_EVENT_HANDLER appAudioHandler, uintptr_t context);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_RESULT_HANDLE_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred

Description

This function registers a client specific Audio v1.0 stream event handler. The Audio v1.0 Host Client Driver will call the appAudioHandler function specified as the second argument with relevant event and associated event data in response to audio stream data transfers that have been scheduled by the client.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
handle	The handle to the Audio v1.0 stream
eventHandler	A pointer to event handler function. If NULL, events will not be generated.
context	The application specific context that is returned in the event handler

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamEventHandlerSet
(
    USB_HOST_AUDIO_V1_STREAM_HANDLE handle,
    USB_HOST_AUDIO_V1_STREAM_EVENT_HANDLER appAudioHandler,
uintptr_t context
);
```

USB_HOST_AUDIO_V1_0_StreamDisable Function

Schedules an audio stream disable request for the specified audio stream.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_0_STREAM_RESULT USB_HOST_AUDIO_V1_0_StreamDisable(USB_HOST_AUDIO_V1_0_STREAM_HANDLE
streamHandle, USB_HOST_AUDIO_V1_0_REQUEST_HANDLE * requestHandle);
```

Returns

- USB_HOST_AUDIO_V1_0_STREAM_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_HANDLE_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_FAILURE - An unknown failure occurred

Description

This function schedules an audio stream disable request for the specified audio stream. A USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE event is generated when this request is completed. USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE_DATA returns the status and request handle of the request.

Remarks

None.

Preconditions

The audio stream should have been opened.

Parameters

Parameters	Description
streamHandle	Handle to the Audio v1.0 stream
requestHandle	Handle to the stream disable request

Function

```
USB_HOST_AUDIO_V1_0_STREAM_RESULT USB_HOST_AUDIO_V1_0_StreamDisable
(
```

```
    USB_HOST_AUDIO_V1_0_STREAM_HANDLE streamHandle,
    USB_HOST_AUDIO_V1_0_REQUEST_HANDLE * requestHandle
);
```

USB_HOST_AUDIO_V1_StreamingInterfaceGetFirst Function

Gets the first streaming interface object from the attached Audio Device.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamingInterfaceGetFirst(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ* streamingInterfaceObj);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The request completed successfully
- USB_HOST_AUDIO_V1_RESULT_END_OF_STREAMING_INTERFACE - No more streaming interfaces are available
- USB_HOST_AUDIO_V1_RESULT_DEVICE_UNKNOWN - Device is not attached
- USB_HOST_AUDIO_V1_RESULT_OBJ_INVALID - Audio Device object is invalid
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An error has occurred

Description

This function will get the first streaming interface object from the attached Audio Device.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached.

Parameters

Parameters	Description
audioObj	Audio v1.0 client driver object.
streamingInterfaceObj	Pointer to an audio streaming interface object.

Function

```
    USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamingInterfaceGetFirst
(
    USB_HOST_AUDIO_V1_OBJ audioObj,
    USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ* streamingInterfaceObj
);
```

USB_HOST_AUDIO_V1_0_StreamEnable Function

Schedules an audio stream enable request for the specified audio stream.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_0_STREAM_RESULT USB_HOST_AUDIO_V1_0_StreamEnable(USB_HOST_AUDIO_V1_0_STREAM_HANDLE
streamHandle, USB_HOST_AUDIO_V1_0_REQUEST_HANDLE * requestHandle);
```

Returns

- USB_HOST_AUDIO_V1_0_STREAM_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_HANDLE_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_FAILURE - An unknown failure occurred

Description

This function schedules an audio stream enable request for the specified audio stream. An audio stream must be enable before scheduling any

data transfer with the stream. A USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE event is generated when this request is completed. USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE_DATA returns the status and request handle of the request.

Remarks

None.

Preconditions

The audio stream should have been opened. Only one audio stream from an audio stream group can be enabled at a time.

Parameters

Parameters	Description
streamHandle	Handle to the audio v1.0 stream
requestHandle	Handle to the stream enable request

Function

```
USB_HOST_AUDIO_V1_0_STREAM_RESULT USB_HOST_AUDIO_V1_0_StreamEnable
(
    USB_HOST_AUDIO_V1_0_STREAM_HANDLE streamHandle,
    USB_HOST_AUDIO_V1_0_REQUEST_HANDLE * requestHandle
);
```

USB_HOST_AUDIO_V1_StreamingInterfaceGetNext Function

Gets the next streaming interface object from the attached Audio Device.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamingInterfaceGetNext(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObjCurrent,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ* streamingInterfaceObjNext);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The request completed successfully
- USB_HOST_AUDIO_V1_RESULT_END_OF_STREAMING_INTERFACE - No more streaming interfaces are available
- USB_HOST_AUDIO_V1_RESULT_DEVICE_UNKNOWN - Device is not attached
- USB_HOST_AUDIO_V1_RESULT_OBJ_INVALID - Audio Device object is invalid
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An error has occurred

Description

This function will get the next streaming interface object from the attached Audio Device.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached.

Parameters

Parameters	Description
audioObj	Audio Device object.
streamingInterfaceObjCurrent	Current audio streaming interface object.
streamingInterfaceObj	Pointer to audio streaming interface object.

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamingInterfaceGetNext
(
    USB_HOST_AUDIO_V1_OBJ audioObj,
    USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObjCurrent
```

USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ* streamingInterfaceObjNext
);

USB_HOST_AUDIO_V1_0_StreamEventHandlerSet Function

Registers an event handler with the Audio v1.0 Client Driver stream.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_0_STREAM_RESULT
USB_HOST_AUDIO_V1_0_StreamEventHandlerSet(USB_HOST_AUDIO_V1_0_STREAM_HANDLE handle,
USB_HOST_AUDIO_V1_0_STREAM_EVENT_HANDLER appAudioHandler, uintptr_t context);
```

Returns

- USB_HOST_AUDIO_V1_0_STREAM_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_HANDLE_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_FAILURE - An unknown failure occurred

Description

This function registers a client specific Audio v1.0 stream event handler. The Audio v1.0 Host Client Driver will call appAudioHandler function specified as 2nd argument with relevant event and associate event data, in response to audio stream data transfers that have been scheduled by the client.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
handle	A handle to the Audio v1.0 stream
eventHandler	A pointer to event handler function. If NULL, events will not be generated.
context	The application specific context that is returned in the event handler

Function

USB_HOST_AUDIO_V1_0_STREAM_RESULT USB_HOST_AUDIO_V1_0_StreamEventHandlerSet
(
USB_HOST_AUDIO_V1_0_STREAM_HANDLE handle,
USB_HOST_AUDIO_V1_0_STREAM_EVENT_HANDLER appAudioHandler,
uintptr_t context
);

USB_HOST_AUDIO_V1_StreamingInterfaceSet Function

Schedules a SET_INTERFACE request to the specified audio stream.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamingInterfaceSet(USB_HOST_AUDIO_V1_STREAM_HANDLE
streamHandle, USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_RESULT_HANDLE_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred

Description

This function schedules an audio stream enable request for the specified audio stream. An audio stream must be enable before scheduling any data transfer with the stream. A USB_HOST_AUDIO_V1_STREAM_EVENT_ENABLE_COMPLETE event is generated when this request is completed. USB_HOST_AUDIO_V1_STREAM_EVENT_ENABLE_COMPLETE_DATA returns the status and request handle of the request.

Remarks

None.

Preconditions

The audio stream should have been opened. Only one audio stream from an audio stream group can be enabled at a time.

Parameters

Parameters	Description
streamHandle	Handle to the Audio v1.0 stream.
requestHandle	Handle to the stream enable request.

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamingInterfaceSet
(
    USB_HOST_AUDIO_V1_STREAM_HANDLE streamHandle,
USB_INTERFACE_DESCRIPTOR* pInterfaceDesc,
    USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle
);
```

USB_HOST_AUDIO_V1_0_StreamGetFirst Function

Returns information about first audio stream in the specified audio stream group.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_0_RESULT USB_HOST_AUDIO_V1_0_StreamGetFirst(USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj,
uint8_t streamGroupIndex, USB_HOST_AUDIO_V1_0_STREAM_INFO * streamInfo);
```

Returns

- USB_HOST_AUDIO_V1_0_STREAM_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_0_RESULT_OBJ_INVALID - The specified Audio v1.0 client driver object does not exist
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_FAILURE - An unknown failure occurred

Description

This function returns information about the first audio stream in the specified audio stream group. The stream group index is parameter to this function and it can be any value starting from zero to the number of stream groups minus one. Number of stream groups can be obtained by using the USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet function.

The streamInfo object is an out parameter to this function.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached to the Host.

Parameters

Parameters	Description
audioDeviceObj	Audio v1.0 Client Driver object
streamGroupIndex	Stream group index
streamInfo	Pointer to the streamInfo object

Function

```
USB_HOST_AUDIO_V1_0_RESULT USB_HOST_AUDIO_V1_0_StreamGetFirst
```

```
(
    USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj,
uint8_t streamGroupIndex,
    USB_HOST_AUDIO_V1_0_STREAM_INFO * streamInfo
);
```

USB_HOST_AUDIO_V1_StreamingInterfaceSettingGetFirst Function

Gets the first streaming interface setting object within an audio streaming interface.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamingInterfaceSettingGetFirst(USB_HOST_AUDIO_V1_OBJ
audioObj, USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ * interfaceSettingObj);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The request completed successfully
- USB_HOST_AUDIO_V1_RESULT_END_OF_INTERFACE_SETTINGS - No more streaming interface settings are available
- USB_HOST_AUDIO_V1_RESULT_DEVICE_UNKNOWN - Device is not attached
- USB_HOST_AUDIO_V1_RESULT_OBJ_INVALID - Audio Device object is invalid
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An error has occurred

Description

This function gets the first streaming interface setting object within an audio streaming interface.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached.

Parameters

Parameters	Description
audioObj	Audio device object.
streamingInterfaceObj	Audio streaming interface object.
interfaceSettingObj	Pointer to the audio streaming interface setting object.

Function

```
  USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamingInterfaceSettingGetFirst
(
    USB_HOST_AUDIO_V1_OBJ audioObj,
    USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
    USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ *interfaceSettingObj
);
```

USB_HOST_AUDIO_V1_0_StreamGetNext Function

Returns information about the next audio stream in the specified audio stream group.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_0_RESULT USB_HOST_AUDIO_V1_0_StreamGetNext(USB_HOST_AUDIO_V1_0_STREAM_OBJ audioStreamObj,
USB_HOST_AUDIO_V1_0_STREAM_INFO * streamInfo);
```

Returns

- USB_HOST_AUDIO_V1_0_STREAM_RESULT_SUCCESS - The operation was successful

- USB_HOST_AUDIO_V1_0_RESULT_OBJ_INVALID - The specified Audio v1.0 client driver object does not exist
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_FAILURE - An unknown failure occurred
- USB_HOST_AUDIO_V1_0_RESULT_END_OF_STREAM_LIST - There are no more audio streams in the stream group

Description

This function returns information about next audio stream in the specified Audio stream group. The USB_HOST_AUDIO_V1_0_StreamGetFirst function should have been called at least once on the same audio stream group before calling this function. Then, calling this function repeatedly on the stream group will return information about the next audio stream in the stream group. When there are no more audio streams to report, the function returns USB_HOST_AUDIO_V1_0_RESULT_END_OF_STREAM_LIST.

Calling the USB_HOST_AUDIO_V1_0_StreamGetFirst function on the stream group index after the USB_HOST_AUDIO_V1_0_StreamGetNext function has been called will cause the Audio v1.0 Client Driver to reset the audio stream group to point to the first stream in the stream group.

Remarks

None.

Preconditions

The USB_HOST_AUDIO_V1_0_StreamGetFirst function must have been called before calling this function.

Parameters

Parameters	Description
audioStreamObj	Present audio stream object
streamInfo	Pointer to the streamInfo object

Function

```
USB_HOST_AUDIO_V1_0_RESULT USB_HOST_AUDIO_V1_0_StreamGetNext
(
    USB_HOST_AUDIO_V1_0_STREAM_OBJ audioStreamObj,
    USB_HOST_AUDIO_V1_0_STREAM_INFO * streamInfo
);
```

USB_HOST_AUDIO_V1_StreamingInterfaceSettingGetNext Function

Gets the next streaming interface setting object within an audio streaming interface.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamingInterfaceSettingGetNext(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObjCurrent,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ * interfaceSettingObjNext);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The request completed successfully
- USB_HOST_AUDIO_V1_RESULT_END_OF_INTERFACE_SETTINGS - No more streaming interface settings are available
- USB_HOST_AUDIO_V1_RESULT_DEVICE_UNKNOWN - Device is not attached
- USB_HOST_AUDIO_V1_RESULT_OBJ_INVALID - Audio Device object is invalid
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An error has occurred

Description

This function gets the next streaming interface setting object within an audio streaming interface.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached.

Parameters

Parameters	Description
audioObj	Audio Device object
streamingInterfaceObj	Audio streaming interface object
interfaceSettingObjCurrent	Current audio streaming interface setting object
interfaceSettingObjNext	Pointer to the next audio streaming interface setting object

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamingInterfaceSettingGetNext
(
    USB_HOST_AUDIO_V1_OBJ audioObj,
    USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
    USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObjCurrent,
    USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ *interfaceSettingObjNext
);
```

USB_HOST_AUDIO_V1_0_StreamSamplingRateSet Function

Schedules an audio stream set sampling rate request for the specified audio stream.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_0_STREAM_RESULT
USB_HOST_AUDIO_V1_0_StreamSamplingRateSet(USB_HOST_AUDIO_V1_0_STREAM_HANDLE streamHandle,
USB_HOST_AUDIO_V1_0_REQUEST_HANDLE * requestHandle, uint32_t * samplingRate);
```

Returns

- USB_HOST_AUDIO_V1_0_STREAM_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_HANDLE_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_FAILURE - An unknown failure occurred

Description

This function schedules an audio stream set sampling rate request for the specified audio stream. A USB_HOST_AUDIO_V1_0_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE event is generated when this request is completed. USB_HOST_AUDIO_V1_0_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE_DATA returns the status and request handle of the request.

Remarks

None.

Preconditions

The audio stream should have been opened.

Parameters

Parameters	Description
streamHandle	Handle to the Audio v1.0 stream
requestHandle	Handle to the stream set sampling rate request
samplingRate	Pointer to the sampling rate

Function

```
USB_HOST_AUDIO_V1_0_STREAM_RESULT USB_HOST_AUDIO_V1_0_StreamSamplingRateSet
(
    USB_HOST_AUDIO_V1_0_STREAM_HANDLE streamHandle,
    USB_HOST_AUDIO_V1_0_REQUEST_HANDLE requestHandle,
uint32_t* samplingRate
);
```

USB_HOST_AUDIO_V1_StreamOpen Function

Opens the specified audio stream.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_STREAM_HANDLE USB_HOST_AUDIO_V1_StreamOpen(USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ
audiostreamingInterfaceObj);
```

Returns

Will return a valid handle if the audio stream could be opened successfully. Otherwise, USB_HOST_AUDIO_V1_RESULT_HANDLE_INVALID is returned. The function will return a valid handle if the stream is ready to be opened.

Description

This function will open the specified audio stream. Once opened, the audio stream can be accessed via the handle that this function returns. The audiostreamingInterfaceObj parameter is the value returned in the USB_HOST_AUDIO_V1_StreamingInterfaceGetFirst or USB_HOST_AUDIO_V1_StreamingInterfaceGetNext functions.

Remarks

None.

Preconditions

The audio streaming interface object should be valid.

Parameters

Parameters	Description
audiostreamingInterfaceObj	Audio streaming interface object

Function

```
USB_HOST_AUDIO_V1_STREAM_HANDLE USB_HOST_AUDIO_V1_StreamOpen
(
    USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ audiostreamingInterfaceObj
);
```

USB_HOST_AUDIO_V1_StreamRead Function

Schedules an audio stream read request for the specified audio stream.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamRead(USB_HOST_AUDIO_V1_STREAM_HANDLE streamHandle,
USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE * transferHandle, void * source, size_t length);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_RESULT_HANDLE_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred

Description

This function schedules an audio stream read request for the specified audio stream. A USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE event is generated when this request is completed. USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE_DATA returns the status and request handle of the request.

Remarks

None.

Preconditions

The audio stream should have been opened and enabled. The direction of the audio stream should be USB_HOST_AUDIO_V1_DIRECTION_IN.

Parameters

Parameters	Description
streamHandle	Handle to the Audio v1.0 stream
transferHandle	Handle to the stream read transfer request
source	Pointer to the buffer containing data to be read from the device
length	Amount of data to read (in bytes)

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamRead
(
    USB_HOST_AUDIO_V1_STREAM_HANDLE streamHandle,
    USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE * transferHandle,
void * source,
size_t length
);
```

USB_HOST_AUDIO_V1_StreamWrite Function

Schedules an audio stream write request for the specified audio stream.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamWrite(USB_HOST_AUDIO_V1_STREAM_HANDLE streamHandle,
USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE * transferHandle, void * source, size_t length);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_RESULT_HANDLE_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred

Description

This function schedules an audio stream write request for the specified audio stream. A USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE event is generated when this request is completed. USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE_DATA returns the status and request handle of the request.

Remarks

None.

Preconditions

The audio stream should have been opened and enabled. The direction of the audio stream should be USB_HOST_AUDIO_V1_DIRECTION_OUT.

Parameters

Parameters	Description
streamHandle	Handle to the Audio v1.0 stream
transferHandle	Handle to the stream write transfer request
source	Pointer to the buffer containing data to be written to the device
length	Amount of data to write (in bytes)

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamWrite
(
    USB_HOST_AUDIO_V1_STREAM_HANDLE streamHandle,
    USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE * transferHandle,
```

void * source,
size_t length
);

USB_HOST_AUDIO_V1_StreamingInterfaceBitResolutionGet Function

Returns the bit resolution of the specified streaming interface setting.

File

usb_host_audio_v1_0.h

C

```
uint8_t USB_HOST_AUDIO_V1_StreamingInterfaceBitResolutionGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj);
```

Returns

The bit resolution size of the audio streaming interface setting.

Description

This function returns the bit resolution size of the specified streaming interface setting.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached.

Parameters

Parameters	Description
audioObj	Audio Device object
streamingInterfaceObj	Audio streaming interface object
interfaceSettingObj	Audio streaming interface setting object

Function

uint8_t USB_HOST_AUDIO_V1_StreamingInterfaceBitResolutionGet
(
USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj
);

USB_HOST_AUDIO_V1_StreamingInterfaceChannelNumbersGet Function

Returns the number of channels of the specified streaming interface setting.

File

usb_host_audio_v1_0.h

C

```
uint8_t USB_HOST_AUDIO_V1_StreamingInterfaceChannelNumbersGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj);
```

Returns

The number of channels present in the audio streaming interface setting.

Description

This function returns the number of channels of the specified streaming interface setting.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached.

Parameters

Parameters	Description
audioObj	Audio Device object
streamingInterfaceObj	Audio streaming interface object
interfaceSettingObj	Audio streaming interface setting object

Function

uint8_t USB_HOST_AUDIO_V1_StreamingInterfaceChannelNumbersGet

(

USB_HOST_AUDIO_V1_OBJ audioObj,

USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,

USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj

);

USB_HOST_AUDIO_V1_StreamingInterfaceDirectionGet Function

Returns the direction of the specified streaming interface setting.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_STREAM_DIRECTION USB_HOST_AUDIO_V1_StreamingInterfaceDirectionGet(USB_HOST_AUDIO_V1_OBJ audioObj, USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj);
```

Returns

- USB_HOST_AUDIO_V1_DIRECTION_OUT - Host to Device
- USB_HOST_AUDIO_V1_DIRECTION_IN - Device to Host

Description

This function returns the direction of the specified streaming interface setting.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached.

Parameters

Parameters	Description
audioObj	Audio Device object
streamingInterfaceObj	Audio streaming interface object
interfaceSettingObj	Audio streaming interface setting object

Function

USB_HOST_AUDIO_V1_STREAM_DIRECTION USB_HOST_AUDIO_V1_StreamingInterfaceDirectionGet

(

USB_HOST_AUDIO_V1_OBJ audioObj,

USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,

USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj

);

USB_HOST_AUDIO_V1_StreamingInterfaceFormatTagGet Function

Returns the format tag of the specified streaming interface setting.

File

usb_host_audio_v1_0.h

C

```
USB_AUDIO_V1_FORMAT_TAG USB_HOST_AUDIO_V1_StreamingInterfaceFormatTagGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj);
```

Returns

The format tag of the audio streaming interface setting.

Description

This function returns the format tag link of the specified streaming interface setting.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached.

Parameters

Parameters	Description
audioObj	Audio Device object
streamingInterfaceObj	Audio streaming interface object
interfaceSettingObj	Audio streaming interface setting object

Function

uint8_t USB_HOST_AUDIO_V1_StreamingInterfaceFormatTagGet

(

USB_HOST_AUDIO_V1_OBJ audioObj,

USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,

USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj

);

USB_HOST_AUDIO_V1_StreamingInterfaceSamplingFrequenciesGet Function

Returns the sampling frequencies supported by the specified streaming interface setting.

File

usb_host_audio_v1_0.h

C

```
uint8_t* USB_HOST_AUDIO_V1_StreamingInterfaceSamplingFrequenciesGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj);
```

Returns

A pointer to the sampling frequencies supported by the audio streaming interface setting.

Description

This function returns the sampling frequencies supported by the specified streaming interface setting.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached.

Parameters

Parameters	Description
audioObj	Audio Device object
streamingInterfaceObj	Audio streaming interface object
interfaceSettingObj	Audio streaming interface setting object

Function

uint8_t* USB_HOST_AUDIO_V1_StreamingInterfaceSamplingFrequenciesGet
(
USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj
);

USB_HOST_AUDIO_V1_StreamingInterfaceSamplingFrequencyTypeGet Function

Returns the sampling frequency type of the specified streaming interface setting.

File

usb_host_audio_v1_0.h

C

```
uint8_t USB_HOST_AUDIO_V1_StreamingInterfaceSamplingFrequencyTypeGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj);
```

Returns

The sampling frequency type of the audio streaming interface setting.

- 0 - Continuous Sampling frequency is supported
- 1 to 255 - The number of discrete sampling frequencies supported by the audio streaming interface

Description

This function returns the sampling frequency type of the specified streaming interface setting.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached.

Parameters

Parameters	Description
audioObj	Audio Device object
streamingInterfaceObj	Audio streaming interface object
interfaceSettingObj	Audio streaming interface setting object

Function

uint8_t USB_HOST_AUDIO_V1_StreamingInterfaceSamplingFrequencyTypeGet
(
USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj
);

USB_HOST_AUDIO_V1_StreamingInterfaceSubFrameSizeGet Function

Returns the sub-frame size of the specified streaming interface setting.

File

usb_host_audio_v1_0.h

C

```
uint8_t USB_HOST_AUDIO_V1_StreamingInterfaceSubFrameSizeGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj);
```

Returns

The sub-frame size of the audio streaming interface setting.

Description

This function returns the sub-frame size of the specified streaming interface setting.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached.

Parameters

Parameters	Description
audioObj	Audio Device object
streamingInterfaceObj	Audio streaming interface object
interfaceSettingObj	Audio streaming interface setting object

Function

uint8_t USB_HOST_AUDIO_V1_StreamingInterfaceSubFrameSizeGet

(

USB_HOST_AUDIO_V1_OBJ audioObj,

USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,

USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj

);

USB_HOST_AUDIO_V1_StreamingInterfaceTerminalLinkGet Function

Returns the terminal link of the specified streaming interface setting.

File

usb_host_audio_v1_0.h

C

```
uint8_t USB_HOST_AUDIO_V1_StreamingInterfaceTerminalLinkGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj);
```

Returns

The terminal link of the audio streaming interface setting.

Description

This function returns the terminal link of the specified streaming interface setting.

Remarks

None.

Preconditions

The Audio v1.0 Device should have been attached.

Parameters

Parameters	Description
audioObj	Audio Device object
streamingInterfaceObj	Audio streaming interface object
interfaceSettingObj	Audio streaming interface setting object

Function

```
uint8_t USB_HOST_AUDIO_V1_StreamingInterfaceTerminalLinkGet
(
    USB_HOST_AUDIO_V1_OBJ audioObj,
    USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ streamingInterfaceObj,
    USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ interfaceSettingObj
);
```

USB_HOST_AUDIO_V1_StreamSamplingFrequencyGet Function

Schedules an audio stream get sampling rate request for the specified audio stream.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamSamplingFrequencyGet(USB_HOST_AUDIO_V1_STREAM_HANDLE
streamHandle, USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle, uint32_t * samplingFrequency);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_RESULT_HANDLE_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred

Description

This function schedules an audio stream set sampling rate request for the specified audio stream. A USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE event is generated when this request is completed. USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE_DATA returns the status and request handle of the request.

Remarks

None.

Preconditions

The audio stream should have been opened.

Parameters

Parameters	Description
streamHandle	Handle to the Audio v1.0 stream
requestHandle	Handle to the stream set sampling rate request
samplingRate	Pointer to the sampling rate

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamSamplingFrequencyGet
(
    USB_HOST_AUDIO_V1_STREAM_HANDLE streamHandle,
    USB_HOST_AUDIO_V1_REQUEST_HANDLE requestHandle,
uint32_t *samplingFrequency
)
```

USB_HOST_AUDIO_V1_StreamSamplingFrequencySet Function

Schedules an audio stream set sampling rate request for the specified audio stream.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamSamplingFrequencySet(USB_HOST_AUDIO_V1_STREAM_HANDLE
streamHandle, USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle, uint32_t * samplingFrequency);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_RESULT_HANDLE_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred

Description

This function schedules an audio stream set sampling rate request for the specified audio stream. A USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE event is generated when this request is completed. USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE_DATA returns the status and request handle of the request.

Remarks

None.

Preconditions

The audio stream should have been opened.

Parameters

Parameters	Description
streamHandle	Handle to the Audio v1.0 stream
requestHandle	Handle to the stream set sampling rate request
samplingRate	Pointer to the sampling rate

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_StreamSamplingFrequencySet
(
    USB_HOST_AUDIO_V1_STREAM_HANDLE streamHandle,
    USB_HOST_AUDIO_V1_REQUEST_HANDLE requestHandle,
uint32_t *samplingFrequency
)
```

USB_HOST_AUDIO_V1_0_StreamRead Function

Schedules an audio stream read request for the specified audio stream.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_0_StreamRead(USB_HOST_AUDIO_V1_STREAM_HANDLE streamHandle,
USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE * transferHandle, void * source, size_t length);
```

Returns

- USB_HOST_AUDIO_V1_0_STREAM_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_HANDLE_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_FAILURE - An unknown failure occurred

Description

This function schedules an audio stream read request for the specified audio stream. A USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE event is generated when this request is completed. USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE_DATA returns the status and request handle of the request.

Remarks

None.

Preconditions

The audio stream should have been opened and enabled. The direction of the audio stream should be USB_HOST_AUDIO_V1_0_DIRECTION_IN.

Parameters

Parameters	Description
streamHandle	Handle to the Audio v1.0 stream
transferHandle	Handle to the stream read transfer request
source	Pointer to the buffer containing data to be read from the device
length	Amount of data to read (in bytes)

Function

```
USB_HOST_AUDIO_V1_0_STREAM_RESULT USB_HOST_AUDIO_V1_0_StreamRead
(
    USB_HOST_AUDIO_V1_0_STREAM_HANDLE streamHandle,
    USB_HOST_AUDIO_V1_0_STREAM_TRANSFER_HANDLE * transferHandle,
void * source,
size_t length
);
```

USB_HOST_AUDIO_V1_0_StreamWrite Function

Schedules an audio stream write request for the specified audio stream.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_0_StreamWrite(USB_HOST_AUDIO_V1_STREAM_HANDLE streamHandle,
USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE * transferHandle, void * source, size_t length);
```

Returns

- USB_HOST_AUDIO_V1_0_STREAM_RESULT_SUCCESS - The operation was successful
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_HANDLE_INVALID - The specified audio stream does not exist
- USB_HOST_AUDIO_V1_0_STREAM_RESULT_FAILURE - An unknown failure occurred

Description

This function schedules an audio stream write request for the specified audio stream. A USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE event is generated when this request is completed. USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE_DATA returns the status and request handle of the request.

Remarks

None.

Preconditions

The audio stream should have been opened and enabled. The direction of the audio stream should be USB_HOST_AUDIO_V1_0_DIRECTION_OUT.

Parameters

Parameters	Description
streamHandle	Handle to the Audio v1.0 stream
transferHandle	Handle to the stream write transfer request
source	Pointer to the buffer containing data to be written to the device
length	Amount of data to write (in bytes)

Function

```
USB_HOST_AUDIO_V1_0_STREAM_RESULT USB_HOST_AUDIO_V1_0_StreamWrite
(
```

USB_HOST_AUDIO_V1_0_STREAM_HANDLE streamHandle,
USB_HOST_AUDIO_V1_0_STREAM_TRANSFER_HANDLE * transferHandle,
void * source,
size_t length
);

c) Other Functions

USB_HOST_AUDIO_V1_TerminalInputChannelConfigGet Function

Returns a structure that describes the spatial location of the logical channels of in the terminal's output audio channel cluster.

File

usb_host_audio_v1_0.h

C

```
USB_AUDIO_CHANNEL_CONFIG USB_HOST_AUDIO_V1_TerminalInputChannelConfigGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject);
```

Returns

The structure that describes the spatial location of the logical channels.

Description

This function returns a structure that describes the spatial location of the logical channels of in the terminal's output audio channel cluster. This function is only applicable to an input terminal. Prior to calling this function the entity object should be obtained by calling USB_HOST_AUDIO_V1_ControlEntityGetFirst, USB_HOST_AUDIO_V1_ControlEntityGetNext, or USB_HOST_AUDIO_V1_EntityObjectGet.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 device object
entityObject	Audio control entity object

Function

USB_AUDIO_CHANNEL_CONFIG USB_HOST_AUDIO_V1_TerminalInputChannelConfigGet
(
USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject
);

USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeRangeGet Function

Schedules a control request to the Audio Device feature unit to get the range supported by the volume control on the specified channel.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeRangeGet(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject, USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle,
uint8_t channelNumber, void * data, size_t size);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The request was scheduled successfully. requestHandle will contain a valid request handle.
- USB_HOST_AUDIO_V1_RESULT_BUSY - The control request mechanism is currently busy. Retry the request.
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred. requestHandle will contain USB_HOST_AUDIO_V1_0_REQUEST_HANDLE_INVALID.
- USB_HOST_AUDIO_V1_RESULT_PARAMETER_INVALID - The data pointer or requestHandle pointer is NULL

Description

This function schedules a control request to the Audio Device feature unit to get the range supported by the volume control on the specified channel.

Prior to calling this function the user should call the USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeSubRangeNumbersGet function to know how many sub-ranges are supported.

Users should calculate the 'size' parameter of this function, as follows:

```
size = Size of number of ranges + nSubRanges * (Size (MIN) + Size (MAX) + Size of (RES))
```

If the request was scheduled successfully, the requestHandle parameter will contain a request handle that uniquely identifies this request. If the request could not be scheduled successfully, requestHandle will contain USB_HOST_AUDIO_V1_REQUEST_HANDLE_INVALID.

When the control request completes, the Audio v1.0 Client Driver will call the callback function that was set using the USB_HOST_AUDIO_V1_EntityRequestCallbackSet function. The context parameter specified here will be returned in the callback.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object
requestHandle	Output parameter that will contain the handle to this request
channelNumber	Channel number to which the volume control is addressed
nSubRanges	Output parameter that will contain the number of sub-ranges when the request is completed and a callback is received

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeRangeGet
(
    USB_HOST_AUDIO_V1_OBJ  audioObj,
    USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject,
    USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle,
uint8_t channelNumber,
void * data,
size_t size
);
```

USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeSubRangeNumbersGet Function

Schedules a control request to an Audio Device feature unit to get the number of sub-ranges supported by the volume control on the specified channel.

File

usb_host_audio_v1_0.h

C

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeSubRangeNumbersGet(USB_HOST_AUDIO_V1_OBJ audioObj, USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject, USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle, uint8_t channelNumber, uint16_t * nSubRanges);
```

Returns

- USB_HOST_AUDIO_V1_RESULT_SUCCESS - The request was scheduled successfully. requestHandle will contain a valid request handle.
- USB_HOST_AUDIO_V1_RESULT_BUSY - The control request mechanism is currently busy. Retry the request.
- USB_HOST_AUDIO_V1_RESULT_FAILURE - An unknown failure occurred. requestHandle will contain USB_HOST_AUDIO_V1_0_REQUEST_HANDLE_INVALID.
- USB_HOST_AUDIO_V1_RESULT_PARAMETER_INVALID - The data pointer or requestHandle pointer is NULL

Description

This function schedules a control request to the Audio Device feature unit to get the number of sub-ranges supported by the volume control on the specified channel. Prior to calling this function the user should check if volume control exists on the specified channel by calling the USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeExists function.

If the request was scheduled successfully, the requestHandle parameter will contain a request handle that uniquely identifies this request. If the request could not be scheduled successfully, requestHandle will contain USB_HOST_AUDIO_V1_REQUEST_HANDLE_INVALID.

When the control request completes, the Audio v1.0 Client Driver will call the callback function that was set using the USB_HOST_AUDIO_V1_EntityRequestCallbackSet function. The context parameter specified here will be returned in the callback.

Remarks

None.

Parameters

Parameters	Description
audioObj	USB Host Audio v1.0 Device object
entityObject	Audio control entity object
requestHandle	Output parameter that will contain the handle to this request
channelNumber	Channel number to which the volume control is addressed
nSubRanges	Output parameter that will contain the number of sub-ranges when the request is completed and a callback is received

Function

```
USB_HOST_AUDIO_V1_RESULT USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeSubRangeNumbersGet
(
    USB_HOST_AUDIO_V1_OBJ  audioObj,
    USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ entityObject,
    USB_HOST_AUDIO_V1_REQUEST_HANDLE * requestHandle,
uint8_t channelNumber,
uint16_t *nSubRanges
);
```

d) Data Types and Constants

USB_HOST_AUDIO_V1_ATTACH_EVENT_HANDLER Type

USB Host Audio v1.0 Client Driver attach event handler function pointer type.

File

usb_host_audio_v1_0.h

C

```
typedef void (* USB_HOST_AUDIO_V1_ATTACH_EVENT_HANDLER)(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_EVENT event, uintptr_t context);
```

Description

USB Host Audio v1.0 Client Driver Attach Event Handler Function Pointer Type.

This data type defines the required function signature of the USB Host Audio v1.0 Client Driver attach event handling callback function. The application must register a pointer to the Audio v1.0 Client Driver attach events handling function whose function signature (parameter and return value types) match the types specified by this function pointer to receive attach and detach events callbacks from the Audio v1.0 Client Driver. The application should use the USB_HOST_AUDIO_V1_AttachEventHandlerSet function to register an attach event handler. The client driver will call this function with the relevant event parameters. The descriptions of the event handler function parameters are as follows:

- audioObj - Audio Device object to which this event is directed
- event - Event indicates if it is an Attach or Detach
- context - Value identifying the context of the application that was registered with the event handling function

Remarks

None.

USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ Type

Defines the type of the Audio v1.0 Host control entity object.

File

usb_host_audio_v1_0.h

C

```
typedef uintptr_t USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ;
```

Description

USB Host Audio v1.0 Control Entity Object

This data type defines the type of the object returned by the USB_HOST_AUDIO_V1_ControlEntityGetFirst or USB_HOST_AUDIO_V1_ControlEntityGetNext functions. This application uses this object to get more information about that audio control entity.

Remarks

None.

USB_HOST_AUDIO_V1_ENTITY_REQUEST_CALLBACK Type

USB Host Audio v1.0 class driver control transfer complete callback function pointer type.

File

usb_host_audio_v1_0.h

C

```
typedef void (* USB_HOST_AUDIO_V1_ENTITY_REQUEST_CALLBACK)(USB_HOST_AUDIO_V1_OBJ audioObj,
USB_HOST_AUDIO_V1_REQUEST_HANDLE requestHandle, USB_HOST_AUDIO_V1_RESULT result, size_t size, uintptr_t
context);
```

Description

USB Host Audio v1.0 Class driver Control Transfer Complete Callback Function Pointer type

This data type defines the required function signature of the USB Host Audio v1.0 class driver control transfer complete callback function. The client must provide a pointer to a control transfer complete callback function whose function signature (parameter and return value types) must match the types specified by this function pointer to receive notification when a control transfer has completed. The application should use the USB_HOST_AUDIO_V1_EntityRequestCallbackSet function to register an entity control request callback. The Audio v1.0 client driver will call this function with the relevant event parameters. The descriptions of the event handler function parameters are as follows:

- audioObj - Audio v1.0 client driver object associated with this event
- requestHandle - Request handle of the control transfer request that caused this event
- result - Completion result of the control transfer. This will be USB_HOST_AUDIO_V1_RESULT_SUCCESS if the control transfer completed successfully, USB_HOST_AUDIO_V1_RESULT_FAILURE if an unknown failure occurred, or USB_HOST_AUDIO_V1_RESULT_REQUEST_STALLED if the request was stalled.
- size - Size of the data stage that was transferred
- context - Value identifying the context of the application that was provided when the USB_HOST_AUDIO_V1_ControlRequest function was called

Remarks

None.

USB_HOST_AUDIO_V1_EVENT Enumeration

Identifies the possible events that the Audio v1.0 Class Driver attach event handler can generate.

File

usb_host_audio_v1_0.h

C

```
typedef enum {
  USB_HOST_AUDIO_V1_EVENT_ATTACH,
  USB_HOST_AUDIO_V1_EVENT_DETACH
} USB_HOST_AUDIO_V1_EVENT;
```

Members

Members	Description
USB_HOST_AUDIO_V1_EVENT_ATTACH	This event occurs when the Host layer has detected the Audio v1.0 Class Driver instance from a USB Audio v1.0 Device. There is no event data associated with this event.

USB_HOST_AUDIO_V1_EVENT_DETACH	This event occurs when host layer has detached the Audio v1.0 Class Driver instance from a USB Audio v1.0 Device. This can happen if the device itself was detached or if the device configuration was changed. There is no event data associated with this event.

Description

Audio v1.0 Class Driver Events

This enumeration identifies the possible events that the Audio v1.0 Class Driver attach event handler can generate. The application should register an event handler using the USB_HOST_AUDIO_V1_AttachEventHandlerSet function to receive Audio v1.0 Class Driver Attach events.

USB_HOST_AUDIO_V1_OBJ Type

Defines the type of the Audio v1.0 Host client object.

File

usb_host_audio_v1_0.h

C

```
typedef uintptr_t USB_HOST_AUDIO_V1_OBJ;
```

Description

USB Host Audio v1.0 Object

This data type defines the type of the Audio Host client object. This type is returned by the client driver attach event handler and is used by the application to open the attached Audio v1.0 Device.

Remarks

None.

USB_HOST_AUDIO_V1_REQUEST_HANDLE Type

USB Host Audio v1.0 Client Driver request handle.

File

usb_host_audio_v1_0.h

C

```
typedef uintptr_t USB_HOST_AUDIO_V1_REQUEST_HANDLE;
```

Description

USB Host Audio v1.0 Client Driver Request Handle

This handle is returned by the Audio v1.0 Host client driver entity control functions and audio stream control request functions. Applications should use this handle to track a request.

Remarks

None.

USB_HOST_AUDIO_V1_RESULT Enumeration

USB Host Audio v1.0 Class Driver result enumeration.

File

usb_host_audio_v1_0.h

C

```
typedef enum {
  USB_HOST_AUDIO_V1_RESULT_FAILURE,
  USB_HOST_AUDIO_V1_RESULT_BUSY,
  USB_HOST_AUDIO_V1_RESULT_REQUEST_STALLED,
  USB_HOST_AUDIO_V1_RESULT_INVALID_PARAMETER,
  USB_HOST_AUDIO_V1_RESULT_DEVICE_UNKNOWN,
  USB_HOST_AUDIO_V1_RESULT_HANDLE_INVALID,
  USB_HOST_AUDIO_V1_RESULT_TRANSFER_ABORTED,
  USB_HOST_AUDIO_V1_RESULT_OBJ_INVALID,
  USB_HOST_AUDIO_V1_RESULT_END_OF_CONTROL_ENTITY,
  USB_HOST_AUDIO_V1_RESULT_END_OF_STREAMING_INTERFACE,
```

```
  USB_HOST_AUDIO_V1_RESULT_END_OF_INTERFACE_SETTINGS,
  USB_HOST_AUDIO_V1_RESULT_SUCCESS
} USB_HOST_AUDIO_V1_RESULT;
```

Members

Members	Description
USB_HOST_AUDIO_V1_RESULT_FAILURE	An unknown failure has occurred
USB_HOST_AUDIO_V1_RESULT_BUSY	The transfer or request could not be scheduled because internal queues are full. The request or transfer should be retried
USB_HOST_AUDIO_V1_RESULT_REQUEST_STALLED	The request was stalled
USB_HOST_AUDIO_V1_RESULT_INVALID_PARAMETER	A required parameter was invalid
USB_HOST_AUDIO_V1_RESULT_DEVICE_UNKNOWN	The associated device does not exist in the system.
USB_HOST_AUDIO_V1_RESULT_HANDLE_INVALID	The specified handle is not valid
USB_HOST_AUDIO_V1_RESULT_TRANSFER_ABORTED	The transfer or requested was aborted
USB_HOST_AUDIO_V1_RESULT_OBJ_INVALID	The specified Audio v1.0 object is invalid
USB_HOST_AUDIO_V1_RESULT_END_OF_CONTROL_ENTITY	No more audio control entity
USB_HOST_AUDIO_V1_RESULT_END_OF_STREAMING_INTERFACE	No more streaming interface settings present in the audio device
USB_HOST_AUDIO_V1_RESULT_END_OF_INTERFACE_SETTINGS	No more interface alternate settings are present in the audio streaming interface
USB_HOST_AUDIO_V1_RESULT_SUCCESS	Indicates that the operation succeeded or the request was accepted and will be processed.

Description

USB Host Audio v1.0 Class Driver Result enumeration.

This enumeration lists the possible USB Host Audio v1.0 Class Driver operation results. These values are returned by Audio v1.0 Class Driver functions.

Remarks

None.

USB_HOST_AUDIO_V1_0_ATTACH_EVENT_HANDLER Type

USB Host Audio v1.0 Client Driver attach event handler function pointer type.

File

usb_host_audio_v1_0.h

C

```
typedef void (* USB_HOST_AUDIO_V1_0_ATTACH_EVENT_HANDLER)(USB_HOST_AUDIO_V1_0_OBJ audioObj,
USB_HOST_AUDIO_V1_0_EVENT event, uintptr_t context);
```

Description

USB Host Audio v1.0 Client Driver Attach Event Handler Function Pointer Type.

This data type defines the required function signature of the USB Host Audio v1.0 Client Driver attach event handling callback function. The application must register a pointer to a Audio v1.0 Client Driver attach events handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive attach and detach events call backs from the Audio v1.0 Client Driver. The client driver will invoke this function with event relevant parameters. The descriptions of the event handler function parameters are as follows:

- audioObj - Handle of the client to which this event is directed
- event - Event indicates if it is an attach or detach
- context - Value identifying the context of the application that was registered with the event handling function

Remarks

None.

USB_HOST_AUDIO_V1_STREAM_DIRECTION Enumeration

USB Host Audio v1.0 Class Driver stream direction.

File

usb_host_audio_v1_0.h

C

```
typedef enum {
  USB_HOST_AUDIO_V1_DIRECTION_OUT,
  USB_HOST_AUDIO_V1_DIRECTION_IN
} USB_HOST_AUDIO_V1_STREAM_DIRECTION;
```

Members

Members	Description
USB_HOST_AUDIO_V1_DIRECTION_OUT	Stream Direction Host to Device
USB_HOST_AUDIO_V1_DIRECTION_IN	Stream Direction Device to Host

Description

USB Host Audio v1.0 Class Driver Stream Direction

This enumeration lists the possible audio stream directions.

Remarks

None.

USB_HOST_AUDIO_V1_STREAM_EVENT Enumeration

Identifies the possible events that the Audio v1.0 Stream can generate.

File

usb_host_audio_v1_0.h

C

```
typedef enum {
  USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE,
  USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE,
  USB_HOST_AUDIO_V1_STREAM_EVENT_INTERFACE_SET_COMPLETE,
  USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_FREQUENCY_SET_COMPLETE,
  USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_FREQUENCY_GET_COMPLETE,
  USB_HOST_AUDIO_V1_STREAM_EVENT_DETACH
} USB_HOST_AUDIO_V1_STREAM_EVENT;
```

Members

Members	Description
USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE	This event occurs when a Audio v1.0 stream read operation has completed (i.e., when the data has been received from the connected Audio v1.0 stream). This event is generated after the application calls the USB_HOST_AUDIO_V1_StreamRead function. The eventData parameter in the event callback function will be of a pointer to a USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE_DATA structure. This contains details about the transfer handle associated with this read request, the amount of data read and the termination status of the read request.
USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE	This event occurs when an Audio v1.0 stream write operation has completed (i.e., when the data has been written to the connected Audio v1.0 stream). This event is generated after the application calls the USB_HOST_AUDIO_V1_StreamWrite function. The eventData parameter in the event callback function will be of a pointer to a USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE_DATA structure. This contains details about the transfer handle associated with this write request, the amount of data written and the termination status of the write request.

USB_HOST_AUDIO_V1_STREAM_EVENT_INTERFACE_SET_COMPLETE	This event occurs when an audio streaming set interface request has been completed. This event is generated after the application calls the USB_HOST_AUDIO_V1_StreamingInterfaceSet function. The eventData parameter in the event callback function will be of a pointer to a USB_HOST_AUDIO_V1_STREAM_EVENT_INTERFACE_SET_COMPLETE_DATA. This contains details about the request handle associated with the interface set request and the termination status of the request.
USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_FREQUENCY_SET_COMPLETE	This event occurs when an Audio v1.0 sampling frequency set request has been completed. This event is generated after the application calls the USB_HOST_AUDIO_V1_StreamSamplingFrequencySet function. The eventData parameter in the event callback function will be of a pointer to a USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_FREQUENCY_SET_COMPLETE_DATA. This contains details about the request handle associated with this sampling frequency set request and the termination status of the request.
USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_FREQUENCY_GET_COMPLETE	This event occurs when an Audio v1.0 sampling frequency get request has been completed. This event is generated after the application calls the USB_HOST_AUDIO_V1_StreamSamplingFrequencyGet function. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_FREQUENCY_GET_COMPLETE_DATA. This contains details about the request handle associated with this sampling frequency get request and the termination status of the request.
USB_HOST_AUDIO_V1_STREAM_EVENT_DETACH	This event occurs when an audio stream is detached from the Host.This can happen if the Audio device itself was detached, or if the Audio device configuration was changed. There is no event data associated with this event.

Description

Audio v1.0 Stream Events

This enumeration identifies the possible events that the Audio v1.0 Stream can generate. The application should register an event handler using the USB_HOST_AUDIO_V1_StreamEventHandlerSet function to receive Audio v1.0 stream events.

An event may have data associated with it. Events that are generated due to a transfer of data between the host and device are accompanied by data structures that provide the status of the transfer termination. For example, the USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE event is accompanied by a pointer to a USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE_DATA data structure. The transferStatus member of this data structure indicates the success or failure of the transfer. A transfer may fail due to the device not responding on the bus, or if the device stalls any stages of the transfer. The event description provides details on the nature of the event and the data that is associated with the event.

USB_HOST_AUDIO_V1_STREAM_EVENT_HANDLER Type

USB Host Audio v1.0 Class Driver stream event handler function pointer type.

File

usb_host_audio_v1_0.h

C

```
typedef USB_HOST_AUDIO_V1_STREAM_EVENT_RESPONSE (*
USB_HOST_AUDIO_V1_STREAM_EVENT_HANDLER)(USB_HOST_AUDIO_V1_STREAM_HANDLE handle,
USB_HOST_AUDIO_V1_STREAM_EVENT event, void * eventData, uintptr_t context);
```

Description

USB Host Audio v1.0 Class Driver Stream Event Handler Function Pointer Type.

This data type defines the required function signature of the USB Host Audio v1.0 Class Driver Stream event handling callback function. The application must register a pointer to the Audio v1.0 Class Driver stream events handling function whose function signature (parameter and return

value types) match the types specified by this function pointer to receive event callbacks from the Audio v1.0 Class Driver. The application should use the USB_HOST_AUDIO_V1_StreamEventHandlerSet function to register an audio stream event handler. The class driver will call this function with the relevant event parameters. The descriptions of the stream event handler function parameters are as follows:

- handle - Handle to the Audio v1.0 stream
- event - Type of event generated
- eventData - This parameter should be type casted to an event specific pointer type based on the event that has occurred. Refer to the USB_HOST_AUDIO_V1_STREAM_EVENT enumeration description for more information.
- context - Value identifying the context of the application that was registered with the event handling function

Remarks

None.

USB_HOST_AUDIO_V1_STREAM_EVENT_INTERFACE_SET_COMPLETE_DATA Structure

USB Host Audio v1.0 class stream control event data.

File

usb_host_audio_v1_0.h

C

```
typedef struct {
  USB_HOST_AUDIO_V1_REQUEST_HANDLE requestHandle;
  USB_HOST_AUDIO_V1_RESULT requestStatus;
} USB_HOST_AUDIO_V1_STREAM_EVENT_INTERFACE_SET_COMPLETE_DATA,
USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE_DATA,
USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_GET_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_AUDIO_V1_REQUEST_HANDLE requestHandle;	Transfer handle of this transfer
USB_HOST_AUDIO_V1_RESULT requestStatus;	Transfer termination status

Description

USB Host Audio v1.0 Class Stream Control Event Data.

This data type defines the data structure returned by the Audio V1.0 stream in conjunction with the following events:

- USB_HOST_AUDIO_V1_STREAM_EVENT_INTERFACE_SET_COMPLETE
- USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_FREQUENCY_SET_COMPLETE
- USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_FREQUENCY_GET_COMPLETE

Remarks

None.

USB_HOST_AUDIO_V1_0_CONTROL_CALLBACK Type

USB Host Audio v1.0 Class Driver control transfer complete callback function pointer type.

File

usb_host_audio_v1_0.h

C

```
typedef void (* USB_HOST_AUDIO_V1_0_CONTROL_CALLBACK)(USB_HOST_AUDIO_V1_0_OBJ audioObj,
USB_HOST_AUDIO_V1_0_REQUEST_HANDLE requestHandle, USB_HOST_AUDIO_V1_0_RESULT result, size_t size, uintptr_t
context);
```

Description

USB Host Audio v1.0 Class driver Control Transfer Complete Callback Function Pointer type

This data type defines the required function signature of the USB Host Audio v1.0 Class Driver control transfer complete callback function. The client must provide a pointer to a control transfer complete callback function whose function signature (parameter and return value types) must match the types specified by this function pointer to receive notification when a control transfer has completed. The pointer to the callback function must be specified in USB_HOST_AUDIO_V1_0_ControlRequest function. The Audio v1.0 client driver will invoke this function with event relevant parameters. The descriptions of the event handler function parameters are as follows:

- audioObj - Audio v1.0 client driver object associated with this event

- requestHandle - Request handle of the control transfer request that caused this event
- result - Completion result of the control transfer. This will be USB_HOST_AUDIO_V1_0_RESULT_SUCCESS if the control transfer completed successfully, USB_HOST_AUDIO_V1_0_RESULT_FAILURE if an unknown failure occurred, or USB_HOST_AUDIO_V1_0_RESULT_REQUEST_STALLED if the request was stalled.

size - Size of the data stage that was transferred context - Value identifying the context of the application that was provided when the USB_HOST_AUDIO_V1_0_ControlRequest function was called.

Remarks

None.

USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE_DATA Structure

USB Host Audio v1.0 class stream data transfer event data.

File

usb_host_audio_v1_0.h

C

```
typedef struct {
  USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE transferHandle;
  size_t length;
  USB_HOST_AUDIO_V1_RESULT result;
} USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE_DATA, USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE transferHandle;	Transfer handle of this transfer
size_t length;	Amount of data transferred
USB_HOST_AUDIO_V1_RESULT result;	Transfer termination status

Description

USB Host Audio v1.0 Class Stream Data Transfer Event Data.

This data type defines the data structure returned by the Audio V1.0 stream in conjunction with the following events:

- USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE_DATA
- USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE_DATA

Remarks

None.

USB_HOST_AUDIO_V1_0_EVENT Macro

Identifies the possible events that the Audio v1.0 Class Driver can generate.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_EVENT USB_HOST_AUDIO_V1_EVENT
```

Description

Audio v1.0 Class Driver Events

This enumeration identifies the possible events that the Audio v1.0 Class Driver can generate. The application should register an event handler using the USB_HOST_AUDIO_V1_0_AttachEventHandlerSet function to receive Audio v1.0 Class Driver events.

USB_HOST_AUDIO_V1_STREAM_EVENT_RESPONSE Enumeration

Returns the type of the USB Host Audio v1.0 stream event handler.

File

usb_host_audio_v1_0.h

C

```
typedef enum {
  USB_HOST_AUDIO_V1_STREAM_EVENT_RESPONSE_NONE
} USB_HOST_AUDIO_V1_STREAM_EVENT_RESPONSE;
```

Members

Members	Description
USB_HOST_AUDIO_V1_STREAM_EVENT_RESPONSE_NONE	This means no response is required

Description

USB Host Audio v1.0 Stream Event Handler Return Type

This enumeration lists the possible return values of the USB Host Audio v1.0 stream event handler.

Remarks

None.

USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE_DATA Structure

USB Host Audio v1.0 class stream control event data.

File

usb_host_audio_v1_0.h

C

```
typedef struct {
  USB_HOST_AUDIO_V1_REQUEST_HANDLE requestHandle;
  USB_HOST_AUDIO_V1_RESULT requestStatus;
} USB_HOST_AUDIO_V1_STREAM_EVENT_INTERFACE_SET_COMPLETE_DATA,
USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE_DATA,
USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_GET_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_AUDIO_V1_REQUEST_HANDLE requestHandle;	Transfer handle of this transfer
USB_HOST_AUDIO_V1_RESULT requestStatus;	Transfer termination status

Description

USB Host Audio v1.0 Class Stream Control Event Data.

This data type defines the data structure returned by the Audio V1.0 stream in conjunction with the following events:

- USB_HOST_AUDIO_V1_STREAM_EVENT_INTERFACE_SET_COMPLETE
- USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_FREQUENCY_SET_COMPLETE
- USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_FREQUENCY_GET_COMPLETE

Remarks

None.

USB_HOST_AUDIO_V1_0_OBJ Macro

Defines the type of the Audio v1.0 Host client object.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_OBJ USB_HOST_AUDIO_V1_OBJ
```

Description

USB Host Audio v1.0 Object

This type defines the type of the Audio Host client object. This type is returned by the attach event handler and is used by the application to open the attached Audio v1.0 Device.

Remarks

None.

USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE_DATA Structure

USB Host Audio v1.0 class stream data transfer event data.

File

usb_host_audio_v1_0.h

C

```
typedef struct {
  USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE transferHandle;
  size_t length;
  USB_HOST_AUDIO_V1_RESULT result;
} USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE_DATA, USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE transferHandle;	Transfer handle of this transfer
size_t length;	Amount of data transferred
USB_HOST_AUDIO_V1_RESULT result;	Transfer termination status

Description

USB Host Audio v1.0 Class Stream Data Transfer Event Data.

This data type defines the data structure returned by the Audio V1.0 stream in conjunction with the following events:

- USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE_DATA
- USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE_DATA

Remarks

None.

USB_HOST_AUDIO_V1_0_REQUEST_HANDLE Macro

USB Host Audio v1.0 Client Driver request handle.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_REQUEST_HANDLE USB_HOST_AUDIO_V1_REQUEST_HANDLE
```

Description

USB Host Audio v1.0 Client Driver Request Handle

This is returned by the Audio v1.0 Client Driver command routines and should be used by the application to track the command especially in cases where transfers are queued.

Remarks

None.

USB_HOST_AUDIO_V1_STREAM_HANDLE Type

Defines the type of the Audio v1.0 Host stream handle.

File

usb_host_audio_v1_0.h

C

```
typedef uintptr_t USB_HOST_AUDIO_V1_STREAM_HANDLE;
```

Description

USB Host Audio stream handle

This data type defines the type of the handle returned by USB_HOST_AUDIO_V1_StreamOpen function. The application uses this handle to interact with an Audio Stream.

Remarks

None.

USB_HOST_AUDIO_V1_0_RESULT Enumeration

USB Host Audio v1.0 Class Driver audio result enumeration.

File

usb_host_audio_v1_0.h

C

```
typedef enum {
  USB_HOST_AUDIO_V1_0_RESULT_BUSY = USB_HOST_AUDIO_V1_0_RESULT_TRANSFER_ABORTED,
  USB_HOST_AUDIO_V1_0_RESULT_REQUEST_STALLED,
  USB_HOST_AUDIO_V1_0_RESULT_OBJ_INVALID,
  USB_HOST_AUDIO_V1_0_RESULT_END_OF_STREAM_LIST,
  USB_HOST_AUDIO_V1_0_RESULT_PARAMETER_INVALID,
  USB_HOST_AUDIO_V1_0_RESULT_DEVICE_UNKNOWN,
  USB_HOST_AUDIO_V1_0_RESULT_FAILURE,
  USB_HOST_AUDIO_V1_0_RESULT_FALSE = 0,
  USB_HOST_AUDIO_V1_0_RESULT_TRUE = 1,
  USB_HOST_AUDIO_V1_0_RESULT_SUCCESS = USB_HOST_RESULT_TRUE
} USB_HOST_AUDIO_V1_0_RESULT;
```

Members

Members	Description
USB_HOST_AUDIO_V1_0_RESULT_BUSY = USB_HOST_AUDIO_V1_0_RESULT_TRANSFER_ABORTED	The transfer or request could not be scheduled because internal • queues are full. The request or transfer should be retried
USB_HOST_AUDIO_V1_0_RESULT_REQUEST_STALLED	The request was stalled
USB_HOST_AUDIO_V1_0_RESULT_OBJ_INVALID	The specified Audio v1.0 Object is Invalid
USB_HOST_AUDIO_V1_0_RESULT_END_OF_STREAM_LIST	No more audio stream present in the Device
USB_HOST_AUDIO_V1_0_RESULT_PARAMETER_INVALID	A required parameter was invalid
USB_HOST_AUDIO_V1_0_RESULT_DEVICE_UNKNOWN	The specified device does not exist in the system
USB_HOST_AUDIO_V1_0_RESULT_FAILURE	An unknown failure has occurred
USB_HOST_AUDIO_V1_0_RESULT_FALSE = 0	Indicates a false condition
USB_HOST_AUDIO_V1_0_RESULT_TRUE = 1	Indicate a true condition
USB_HOST_AUDIO_V1_0_RESULT_SUCCESS = USB_HOST_RESULT_TRUE	Indicates that the operation succeeded or the request was accepted and will be processed.

Description

USB Host Audio v1.0 Class Driver Result enumeration.

This enumeration lists the possible USB Host Audio v1.0 Class Driver operation results. These values are returned by Audio v1.0 Class Driver functions.

Remarks

None.

USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE Type

USB Host Audio v1.0 Class Driver stream data transfer handle.

File

usb_host_audio_v1_0.h

C

```
typedef uintptr_t USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE;
```

Description

USB Host Audio v1.0 Class Driver Stream Data Transfer Handle

This handle is returned by the Audio v1.0 Class driver stream data transfer functions and should be used by the application to track the transfer, especially in cases where transfers are queued.

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_DIRECTION Macro

USB Host Audio v1.0 Class Driver stream direction.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_STREAM_DIRECTION USB_HOST_AUDIO_V1_STREAM_DIRECTION
```

Description

USB Host Audio v1.0 Class Driver Stream Direction

This macro defines the stream direction of the USB Host Audio v1.0 Class Driver.

Remarks

None.

USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ Type

Defines the type of the Audio v1.0 Host streaming interface object.

File

usb_host_audio_v1_0.h

C

```
typedef uintptr_t USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ;
```

Description

USB Host Audio v1.0 Streaming interface Object

This data type defines the type of the Audio v1.0 Host streaming interface object. This type is returned by the USB_AUDIO_V1_StreamingInterfaceGetFirst and USB_AUDIO_V1_StreamingInterfaceGetNext functions.

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_EVENT Enumeration

Identifies the possible events that the Audio v1.0 stream can generate.

File

usb_host_audio_v1_0.h

C

```
typedef enum {
  USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE,
  USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE,
  USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE,
  USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE,
  USB_HOST_AUDIO_V1_0_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE
} USB_HOST_AUDIO_V1_0_STREAM_EVENT;
```

Members

Members	Description
USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE	This event occurs when a Audio v1.0 stream read operation has completed (i.e., when the data has been received from the connected Audio v1.0 stream). This event is generated after the application calls the USB_HOST_AUDIO_V1_0_StreamRead function. The eventData parameter in the event callback function will be of a pointer to a USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE_DATA structure. This contains details about the transfer handle associated with this read request, the amount of data read and the termination status of the read request.
USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE	This event occurs when an Audio v1.0 stream write operation has completed (i.e., when the data has been written to the connected Audio v1.0 stream). This event is generated after the application calls the USB_HOST_AUDIO_V1_0_StreamWrte function. The eventData parameter in the event callback function will be of a pointer to a USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE_DATA structure. This contains details about the transfer handle associated with this write request, the amount of data written and the termination status of the write request.
USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE	This event occurs when an Audio v1.0 stream enable request has been completed. This event is generated after the application calls the USB_HOST_AUDIO_V1_0_StreamEnable function. The eventData parameter in the event callback function will be of a pointer to a USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE_DATA. This contains details about the request handle associated with this stream enable request and the termination status of the Stream Enable request.
USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE	This event occurs when an Audio v1.0 stream disable request has been completed. This event is generated after the application calls the USB_HOST_AUDIO_V1_0_StreamDisable function. The eventData parameter in the event callback function will be of a pointer to a USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE_DATA. This contains details about the request handle associated with this stream disable request and the termination status of the Stream Disable request.
USB_HOST_AUDIO_V1_0_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE	This event occurs when an Audio v1.0 sampling rate set request has been completed. This event is generated after the application calls the USB_HOST_AUDIO_V1_0_StreamSamplingRateSet function. The eventData parameter in the event callback function will be of a pointer to a USB_HOST_AUDIO_V1_0_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE_DATA. This contains details about the request handle associated with this Sampling Rate Set request and the termination status of the stream disable request.

Description

Audio v1.0 Stream Events

This enumeration identifies the possible events that the Audio v1.0 stream can generate. The application should register an event handler using the USB_HOST_AUDIO_V1_0_StreamEventHandlerSet function to receive Audio v1.0 stream events.

An event may have data associated with it. Events that are generated due to a transfer of data between the Host and Device are accompanied by data structures that provide the status of the transfer termination. For example, the USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE event is accompanied by a pointer to a USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE_DATA data structure. The transferStatus member of this data structure indicates the success or failure of the transfer. A transfer may fail due to the Device not responding on the bus if the Device stalls any stages of the transfer or due to NAK time-outs. The event description provides details on the nature of the event and the data that is associated with the event.

USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ Type

Defines the type of the Audio v1.0 Host streaming interface setting object.

File

usb_host_audio_v1_0.h

C

```
typedef uintptr_t USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ;
```

Description

USB Host Audio v1.0 Streaming Interface Setting Object

This data type defines the type of the Audio v1.0 Host streaming interface setting object. This type is returned by the USB_AUDIO_V1_StreamingInterfaceSettingGetFirst and USB_AUDIO_V1_StreamingInterfaceSettingGetNext functions.

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE_DATA Structure

USB Host Audio v1.0 class stream control event data.

File

usb_host_audio_v1_0.h

C

```
typedef struct {
  USB_HOST_AUDIO_V1_0_REQUEST_HANDLE requestHandle;
  USB_HOST_AUDIO_V1_0_RESULT requestStatus;
} USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE_DATA,
USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_AUDIO_V1_0_REQUEST_HANDLE requestHandle;	Transfer handle of this transfer
USB_HOST_AUDIO_V1_0_RESULT requestStatus;	Transfer termination status

Description

USB Host Audio v1.0 Class Stream Control Event Data.

This data type defines the data structure returned by the Audio V1.0 Client Driver in conjunction with the following events:

- USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE_DATA
- USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE_DATA

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE_DATA Structure

USB Host Audio v1.0 class stream control event data.

File

usb_host_audio_v1_0.h

C

```
typedef struct {
  USB_HOST_AUDIO_V1_0_REQUEST_HANDLE requestHandle;
  USB_HOST_AUDIO_V1_0_RESULT requestStatus;
} USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE_DATA,
USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_AUDIO_V1_0_REQUEST_HANDLE requestHandle;	Transfer handle of this transfer
USB_HOST_AUDIO_V1_0_RESULT requestStatus;	Transfer termination status

Description

USB Host Audio v1.0 Class Stream Control Event Data.

This data type defines the data structure returned by the Audio V1.0 Client Driver in conjunction with the following events:

- USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE_DATA
- USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE_DATA

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_EVENT_HANDLER Type

USB Host Audio v1.0 Class Driver stream event handler function pointer type.

File

usb_host_audio_v1_0.h

C

```
typedef USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE (*
USB_HOST_AUDIO_V1_0_STREAM_EVENT_HANDLER)(USB_HOST_AUDIO_V1_0_STREAM_HANDLE handle,
USB_HOST_AUDIO_V1_0_STREAM_EVENT event, void * eventData, uintptr_t context);
```

Description

USB Host Audio v1.0 Class Driver Stream Event Handler Function Pointer Type.

This data type defines the required function signature of the USB Host Audio v1.0 Class Driver stream event handling callback function. The application must register a pointer to a Audio v1.0 Class Driver stream events handling function whose function signature (parameter and return value types) match the types specified by this function pointer to receive event call backs from the Audio v1.0 Class Driver. The class driver will call this function with relevant event parameters. The descriptions of the event handler function parameters are as follows:

- handle - Handle to the Audio v1.0 stream
- event - Type of event generated
- eventData - This parameter should be type casted to an event specific pointer type based on the event that has occurred. Refer to the USB_HOST_AUDIO_V1_0_STREAM_EVENT enumeration description for more information.
- context - Value identifying the context of the application that was registered with the event handling function

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE_DATA Macro

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE_DATA
USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE_DATA
```

Description

This is macro USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE_DATA.

USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE Macro

Returns the type of the USB Audio v1.0 Host Client Driver event handler.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE USB_HOST_AUDIO_V1_STREAM_EVENT_RESPONSE
```

Description

USB Host Audio v1.0 Event Handler Return Type

This enumeration lists the possible return values of the USB Audio v1.0 Host Client Driver event handler.

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE_DATA Macro

USB Host Audio v1.0 class stream transfer event data.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE_DATA
USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE_DATA
```

Description

USB Host Audio v1.0 Class Stream Transfer Event Data.

This data type defines the data structure returned by the Audio V1.0 Client Driver in conjunction with the following events:

- USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE_DATA
- USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE_DATA

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_HANDLE Macro

Defines the type of the Audio v1.0 Host stream handle.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_STREAM_HANDLE USB_HOST_AUDIO_V1_STREAM_HANDLE
```

Description

USB Host Audio stream handle

This type defines the type of the handle returned by the USB_HOST_AUDIO_V1_0_StreamOpen function. This application uses this handle to interact with an audio stream.

Remarks

None.

USB_HOST_AUDIO_V1_INTERFACE Macro

USB HOST Audio v1.0 Client Driver interface.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_INTERFACE
```

Description

USB HOST Audio V1 Client Driver Interface

This macro should be used by the application in the TPL table while adding support for the USB Audio v1.0 Host Client Driver.

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_INFO Structure

File

usb_host_audio_v1_0.h

C

```
typedef struct {
  USB_HOST_AUDIO_V1_0_STREAM_OBJ streamObj;
  USB_AUDIO_FORMAT_CODE format;
  USB_HOST_AUDIO_V1_0_STREAM_DIRECTION streamDirection;
  uint8_t nChannels;
  uint8_t subFrameSize;
  uint8_t bitResolution;
  uint8_t nSamplingRates;
  uint32_t tSamFreq[USB_HOST_AUDIO_V1_SAMPLING_FREQUENCIES_NUMBER];
} USB_HOST_AUDIO_V1_0_STREAM_INFO;
```

Members

Members	Description
USB_HOST_AUDIO_V1_0_STREAM_OBJ streamObj;	Audio Stream Object. Clients need to pass this object when opening this audio stream using USB_HOST_AUDIO_V1_0_StreamOpen function.
USB_AUDIO_FORMAT_CODE format;	Audio Format code for this Stream
USB_HOST_AUDIO_V1_0_STREAM_DIRECTION streamDirection;	Stream direction
uint8_t nChannels;	Number of physical channels in the audio stream
uint8_t subFrameSize;	Number of bytes occupied by one audio sub-frame
uint8_t bitResolution;	Number of effectively used bits from the available bits in an audio sub-frame
uint8_t nSamplingRates;	Indicates how the sampling frequency can be programmed: 0: Continuous sampling frequency 1..255: Number of discrete sampling frequencies supported by Audio stream
uint32_t tSamFreq[USB_HOST_AUDIO_V1_SAMPLING_FREQUENCIES_NUMBER];	Supported sampling Frequencies

Description

This is type USB_HOST_AUDIO_V1_0_STREAM_INFO.

USB_HOST_AUDIO_V1_REQUEST_HANDLE_INVALID Macro

USB Host Audio v1.0 Client Driver invalid request handle.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_REQUEST_HANDLE_INVALID ((USB_HOST_AUDIO_V1_REQUEST_HANDLE)(-1))
```

Description

USB Host Audio v1.0 Client Driver Invalid Request Handle

This handle is returned by the Audio v1.0 Client driver command routines when the request could not be scheduled.

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_OBJ Type

Defines the type of the Audio v1.0 Host stream object.

File

usb_host_audio_v1_0.h

C

```
typedef uintptr_t USB_HOST_AUDIO_V1_0_STREAM_OBJ;
```

Description

USB Host Audio v1.0 Stream Object

This type defines the type of the Audio v1.0 Host stream object. This type is returned by USB_AUDIO_V1_0_StreamGetFirst and USB_AUDIO_V1_0_StreamGetNext as part of USB_HOST_AUDIO_V1_0_STREAM_INFO structure.

Remarks

None.

USB_HOST_AUDIO_V1_STREAM_HANDLE_INVALID Macro

Defines Audio v1.0 Host stream invalid handle.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_STREAM_HANDLE_INVALID ((USB_HOST_AUDIO_V1_STREAM_HANDLE)(-1))
```

Description

USB Host Audio stream Invalid handle

This handle is returned by the USB_HOST_AUDIO_V1_StreamOpen function when a stream open has failed.

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_RESULT Enumeration

USB Host Audio v1.0 stream result enumeration.

File

usb_host_audio_v1_0.h

C

```
typedef enum {
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_REQUEST_BUSY = USB_HOST_RESULT_REQUEST_BUSY,
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_TRANSFER_ABORTED,
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_REQUEST_STALLED,
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_HANDLE_INVALID,
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_END_OF_DEVICE_LIST,
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_INTERFACE_UNKNOWN,
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_PARAMETER_INVALID,
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_CONFIGURATION_UNKNOWN,
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_BUS_NOT_ENABLED,
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_BUS_UNKNOWN,
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_UNKNOWN,
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_FAILURE,
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_FALSE = 0,
  USB_HOST_AUDIO_V1_0_STREAM_RESULT_TRUE = 1,
  USB_HOST_AUDIO_V1_0_STREAM_SUCCESS = USB_HOST_RESULT_TRUE
} USB_HOST_AUDIO_V1_0_STREAM_RESULT;
```

Members

Members	Description
USB_HOST_AUDIO_V1_0_STREAM_RESULT_REQUEST_BUSY = USB_HOST_RESULT_REQUEST_BUSY	The transfer or request could not be scheduled because internal • queues are full. The request or transfer should be retried
USB_HOST_AUDIO_V1_0_STREAM_RESULT_TRANSFER_ABORTED	Request was aborted

USB_HOST_AUDIO_V1_0_STREAM_RESULT_REQUEST_STALLED	Request was stalled
USB_HOST_AUDIO_V1_0_STREAM_RESULT_HANDLE_INVALID	The specified Stream Handle is not valid
USB_HOST_AUDIO_V1_0_STREAM_RESULT_END_OF_DEVICE_LIST	The end of the device list was reached.
USB_HOST_AUDIO_V1_0_STREAM_RESULT_INTERFACE_UNKNOWN	The specified interface is not available
USB_HOST_AUDIO_V1_0_STREAM_RESULT_PARAMETER_INVALID	A NULL parameter was passed to the function
USB_HOST_AUDIO_V1_0_STREAM_RESULT_CONFIGURATION_UNKNOWN	The specified configuration does not exist on this device.
USB_HOST_AUDIO_V1_0_STREAM_RESULT_BUS_NOT_ENABLED	A bus operation was requested but the bus was not operated
USB_HOST_AUDIO_V1_0_STREAM_RESULT_BUS_UNKNOWN	The specified bus does not exist in the system
USB_HOST_AUDIO_V1_0_STREAM_RESULT_UNKNOWN	The specified audio stream does not exist in the system
USB_HOST_AUDIO_V1_0_STREAM_RESULT_FAILURE	An unknown failure has occurred
USB_HOST_AUDIO_V1_0_STREAM_RESULT_FALSE = 0	Indicates a false condition
USB_HOST_AUDIO_V1_0_STREAM_RESULT_TRUE = 1	Indicate a true condition
USB_HOST_AUDIO_V1_0_STREAM_SUCCESS = USB_HOST_RESULT_TRUE	Indicates that the operation succeeded or the request was accepted and will be processed.

Description

USB Host Audio v1.0 Stream Result enumeration.

This enumeration lists the possible USB Host Audio v1.0 stream operation results. These values are returned by Audio v1.0 stream functions.

Remarks

None.

USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE_INVALID Macro

USB Host Audio v1.0 Class Driver invalid stream data transfer handle.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE_INVALID ((USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE)(-1))
```

Description

USB Host Audio v1.0 Class Driver Invalid Stream Data Transfer Handle Definition

This macro defines a USB Host Audio v1.0 Class Driver invalid stream data transfer handle. An invalid transfer handle is returned by the Audio v1.0 Class Driver stream data transfer routines when the request was not successful.

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_TRANSFER_HANDLE Macro

USB Host Audio v1.0 Class Driver transfer handle.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_STREAM_TRANSFER_HANDLE USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE
```

Description

USB Host Audio v1.0 Class Driver Transfer Handle

This is returned by the Audio v1.0 Class Driver command and data transfer routines and should be used by the application to track the transfer especially in cases where transfers are queued.

Remarks

None.

USB_HOST_AUDIO_V1_0_INTERFACE Macro

USB HOST Audio Client Driver interface.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_INTERFACE (void*)USB_HOST_AUDIO_V1_INTERFACE
```

Description

USB HOST Audio Client Driver Interface

This macro should be used by the application in the TPL table while adding support for the USB Audio Host Client Driver.

Remarks

None.

USB_HOST_AUDIO_V1_0_REQUEST_HANDLE_INVALID Macro

USB Host Audio v1.0 Client Driver invalid request handle.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_REQUEST_HANDLE_INVALID ((USB_HOST_AUDIO_V1_0_REQUEST_HANDLE)(-1))
```

Description

USB Host Audio v1.0 Client Driver Invalid Request Handle

This is returned by the Audio v1.0 Client Driver command routines when the request could not be scheduled.

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_HANDLE_INVALID Macro

Defines the type of the Audio v1.0 Host stream invalid handle.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_STREAM_HANDLE_INVALID USB_HOST_AUDIO_V1_STREAM_HANDLE_INVALID
```

Description

USB Host Audio stream Invalid handle

This is returned by the USB_HOST_AUDIO_V1_0_StreamOpen function when a stream open request has failed.

Remarks

None.

USB_HOST_AUDIO_V1_0_STREAM_TRANSFER_HANDLE_INVALID Macro

USB Host Audio v1.0 Class Driver invalid transfer handle definition.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_STREAM_TRANSFER_HANDLE_INVALID USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE_INVALID
```

Description

USB Host Audio v1.0 Class Driver Invalid Transfer Handle Definition

This macro defines a USB Host Audio v1.0 Class Driver invalid transfer handle. A invalid transfer handle is returned by the Audio v1.0 Class Driver data and command transfer routines when the request was not successful.

Remarks

None.

USB_HOST_AUDIO_V1_0_AttachEventHandlerSet Macro

Sets an attach/detach event handler.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_AttachEventHandlerSet USB_HOST_AUDIO_V1_AttachEventHandlerSet
```

Returns

- USB_HOST_AUDIO_V1_0_RESULT_SUCCESS - if the attach event handler was registered

successfully

- USB_HOST_AUDIO_V1_0_RESULT_FAILURE - if the number of registered event

handlers has exceeded USB_HOST_AUDIO_V1_0_ATTACH_LISTENERS_NUMBER

Description

This function will set an attach event handler. The attach event handler will be called when a Audio v1.0 device has been attached or detached. The context will be returned in the event handler. This function should be called before the bus has been enabled.

Remarks

This function should be called before the USB_HOST_BusEnable function is called.

Preconditions

None.

Parameters

Parameters	Description
eventHandler	Pointer to the attach event handler
context	An application defined context that will be returned in the event handler

Function

```
USB_HOST_AUDIO_V1_0_RESULT USB_HOST_AUDIO_V1_0_AttachEventHandlerSet
(
    USB_HOST_AUDIO_V1_0_ATTACH_EVENT_HANDLER eventHandler,
uintptr_t context
);
```

USB_HOST_AUDIO_V1_0_DeviceObjHandleGet Macro

Returns the device object handle for this Audio v1.0 Device.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_DeviceObjHandleGet USB_HOST_AUDIO_V1_DeviceObjHandleGet
```

Returns

This function will return a valid device object handle if the device is still connected to the system. Otherwise, USB_HOST_DEVICE_OBJ_HANDLE_INVALID is returned.

Description

This function returns the device object handle for this Audio v1.0 Device. This returned device object handle can be used by the application to perform device-level operations such as getting the string descriptors.

Remarks

None.

Preconditions

None.

Parameters

Parameters	Description
audioDeviceObj	Audio V1.0 device object handle returned in the USB_HOST_AUDIO_V1_0_ATTACH_EVENT_HANDLER function.

Function

```
USB_HOST_DEVICE_OBJ_HANDLE USB_HOST_AUDIO_V1_0_DeviceObjHandleGet
(
    USB_HOST_AUDIO_V1_0_OBJ audioDeviceObj
);
```

USB_HOST_AUDIO_V1_0_DIRECTION_IN Macro

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_DIRECTION_IN USB_HOST_AUDIO_V1_DIRECTION_IN
```

Description

This is macro USB_HOST_AUDIO_V1_0_DIRECTION_IN.

USB_HOST_AUDIO_V1_0_DIRECTION_OUT Macro

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_DIRECTION_OUT USB_HOST_AUDIO_V1_DIRECTION_OUT
```

Description

This is macro USB_HOST_AUDIO_V1_0_DIRECTION_OUT.

USB_HOST_AUDIO_V1_0_EVENT_ATTACH Macro

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_EVENT_ATTACH USB_HOST_AUDIO_V1_EVENT_ATTACH
```

Description

This is macro USB_HOST_AUDIO_V1_0_EVENT_ATTACH.

USB_HOST_AUDIO_V1_0_EVENT_DETACH Macro

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_EVENT_DETACH USB_HOST_AUDIO_V1_EVENT_DETACH
```

Description

This is macro USB_HOST_AUDIO_V1_0_EVENT_DETACH.

USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE_NONE Macro

Returns the type of the USB Host Audio v1.0 stream event handler.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE_NONE USB_HOST_AUDIO_V1_STREAM_EVENT_RESPONSE_NONE
```

Description

USB Host Audio v1.0 Stream Event Handler Return Type

This enumeration lists the possible return values of the USB Host Audio v1.0 stream event handler.

Remarks

None.

USB_HOST_AUDIO_V1_0_StreamClose Macro

Closes the audio stream.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_StreamClose USB_HOST_AUDIO_V1_StreamClose
```

Returns

None.

Description

This function will close the open audio stream. This closes the association between the application entity that opened the audio stream and the audio stream. The audio stream handle becomes invalid.

Remarks

The device handle becomes invalid after calling this function.

Preconditions

None.

Parameters

Parameters	Description
audioSteamHandle	handle to the audio stream obtained from the USB_HOST_AUDIO_V1_0_StreamOpen function.

Function

void USB_HOST_AUDIO_V1_0_StreamClose

(

USB_HOST_AUDIO_V1_0_STREAM_HANDLE audioSteamHandle

);

USB_HOST_AUDIO_V1_0_StreamOpen Macro

Opens the specified audio stream.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_0_StreamOpen USB_HOST_AUDIO_V1_StreamOpen
```

Returns

This function will return a valid handle if the audio stream could be opened successfully; otherwise, it will return USB_HOST_AUDIO_V1_0_STREAM_RESULT_HANDLE_INVALID. The function will return a valid handle if the stream is ready to be opened.

Description

This function will open the specified audio stream. Once opened, the audio stream can be accessed via the handle which this function returns. The audioStreamObj parameter is the value returned in the USB_HOST_AUDIO_V1_0_StreamGetFirst or USB_HOST_AUDIO_V1_0_StreamGetNext functions.

Remarks

None.

Preconditions

The audio stream object should be valid.

Parameters

Parameters	Description
audioStreamObj	Audio stream object

Section

Audio Stream Access Functions

****

****

****

Function

```
USB_HOST_AUDIO_V1_0_STREAM_HANDLE USB_HOST_AUDIO_V1_0_StreamOpen
(
    USB_HOST_AUDIO_V1_0_STREAM_OBJ audioStreamObj
);
```

USB_HOST_AUDIO_V1_SAMPLING_FREQUENCIES_NUMBER Macro

This structure defines USB Host audio stream information structure.

File

usb_host_audio_v1_0.h

C

```
#define USB_HOST_AUDIO_V1_SAMPLING_FREQUENCIES_NUMBER USB_HOST_AUDIO_V1_0_SAMPLING_FREQUENCIES_NUMBER
```

Description

USB Host Audio stream Info table structure

This structure is an out parameter to the functions USB_HOST_AUDIO_V1_0_StreamGetFirst and USB_HOST_AUDIO_V1_0_StreamGetNext

functions. This structure contains information about an audio stream in the attached Audio Device. This structure contains the stream object, audio format, etc.

Remarks

None.

USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_GET_COMPLETE_DATA Structure

USB Host Audio v1.0 class stream control event data.

File

usb_host_audio_v1_0.h

C

```
typedef struct {
  USB_HOST_AUDIO_V1_REQUEST_HANDLE requestHandle;
  USB_HOST_AUDIO_V1_RESULT requestStatus;
} USB_HOST_AUDIO_V1_STREAM_EVENT_INTERFACE_SET_COMPLETE_DATA,
USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE_DATA,
USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_GET_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_AUDIO_V1_REQUEST_HANDLE requestHandle;	Transfer handle of this transfer
USB_HOST_AUDIO_V1_RESULT requestStatus;	Transfer termination status

Description

USB Host Audio v1.0 Class Stream Control Event Data.

This data type defines the data structure returned by the Audio V1.0 stream in conjunction with the following events:

- USB_HOST_AUDIO_V1_STREAM_EVENT_INTERFACE_SET_COMPLETE
- USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_FREQUENCY_SET_COMPLETE
- USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_FREQUENCY_GET_COMPLETE

Remarks

None.

Files

Files

Name	Description
usb_host_audio_v1_0.h	USB Host Audio v1_0 Class Driver Interface Header
usb_host_audio_v1_0_config_template.h	USB host Audio v1.0 Class configuration definitions template

Description

This section lists the source and header files used by the library.

usb_host_audio_v1_0.h

USB Host Audio v1_0 Class Driver Interface Header

Enumerations

	Name	Description
	USB_HOST_AUDIO_V1_0_RESULT	USB Host Audio v1.0 Class Driver audio result enumeration.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT	Identifies the possible events that the Audio v1.0 stream can generate.
	USB_HOST_AUDIO_V1_0_STREAM_RESULT	USB Host Audio v1.0 stream result enumeration.
	USB_HOST_AUDIO_V1_EVENT	Identifies the possible events that the Audio v1.0 Class Driver attach event handler can generate.
	USB_HOST_AUDIO_V1_RESULT	USB Host Audio v1.0 Class Driver result enumeration.
	USB_HOST_AUDIO_V1_STREAM_DIRECTION	USB Host Audio v1.0 Class Driver stream direction.

	USB_HOST_AUDIO_V1_STREAM_EVENT	Identifies the possible events that the Audio v1.0 Stream can generate.
	USB_HOST_AUDIO_V1_STREAM_EVENT_RESPONSE	Returns the type of the USB Host Audio v1.0 stream event handler.

Functions

	Name	Description
	USB_HOST_AUDIO_V1_0_ControlRequest	Schedules an Audio v1.0 control transfer.
	USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet	Gets the number of stream groups present in the attached Audio v1.0 Device.
	USB_HOST_AUDIO_V1_0_StreamDisable	Schedules an audio stream disable request for the specified audio stream.
	USB_HOST_AUDIO_V1_0_StreamEnable	Schedules an audio stream enable request for the specified audio stream.
	USB_HOST_AUDIO_V1_0_StreamEventHandlerSet	Registers an event handler with the Audio v1.0 Client Driver stream.
	USB_HOST_AUDIO_V1_0_StreamGetFirst	Returns information about first audio stream in the specified audio stream group.
	USB_HOST_AUDIO_V1_0_StreamGetNext	Returns information about the next audio stream in the specified audio stream group.
	USB_HOST_AUDIO_V1_0_StreamRead	Schedules an audio stream read request for the specified audio stream.
	USB_HOST_AUDIO_V1_0_StreamSamplingRateSet	Schedules an audio stream set sampling rate request for the specified audio stream.
	USB_HOST_AUDIO_V1_0_StreamWrite	Schedules an audio stream write request for the specified audio stream.
	USB_HOST_AUDIO_V1_AttachEventHandlerSet	Sets an attach/detach event handler.
	USB_HOST_AUDIO_V1_ControlEntityGetFirst	Retrieves the handle to the first audio control entity
	USB_HOST_AUDIO_V1_ControlEntityGetNext	Retrieves the handle to the next audio control entity.
	USB_HOST_AUDIO_V1_DeviceObjHandleGet	Returns the device object handle for this Audio v1.0 Device.
	USB_HOST_AUDIO_V1_EntityObjectGet	Retrieves the entity object for the entity ID.
	USB_HOST_AUDIO_V1_EntityRequestCallbackSet	Registers an audio entity request callback function with the Audio v1.0 Client Driver.
	USB_HOST_AUDIO_V1_EntityTypeGet	Returns the entity type of the audio control entity.
	USB_HOST_AUDIO_V1_FeatureUnitChannelMuteExists	Returns "true" if mute control exists for the specified channel of the feature unit.
	USB_HOST_AUDIO_V1_FeatureUnitChannelMuteGet	Schedules a get mute control request to the specified channel.
	USB_HOST_AUDIO_V1_FeatureUnitChannelMuteSet	Schedules a set mute control request to the specified channel.
	USB_HOST_AUDIO_V1_FeatureUnitChannelNumbersGet	Returns the number of channels.
	USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeExists	Returns "true" if volume control exists for the specified channel of the feature unit.
	USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeGet	Schedules a get current volume control request to the specified channel.
	USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeRangeGet	Schedules a control request to the Audio Device feature unit to get the range supported by the volume control on the specified channel.
	USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeSet	Schedules a set current volume control request to the specified channel.
	USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeSubRangeNumbersGet	Schedules a control request to an Audio Device feature unit to get the number of sub-ranges supported by the volume control on the specified channel.
	USB_HOST_AUDIO_V1_FeatureUnitIDGet	Returns ID of the Feature Unit.
	USB_HOST_AUDIO_V1_FeatureUnitSourceIDGet	Returns the ID of the unit or terminal to which this feature unit is connected.
	USB_HOST_AUDIO_V1_StreamClose	Closes the audio stream.
	USB_HOST_AUDIO_V1_StreamEventHandlerSet	Registers an event handler with the Audio v1.0 Client Driver stream.
	USB_HOST_AUDIO_V1_StreamingInterfaceBitResolutionGet	Returns the bit resolution of the specified streaming interface setting.

	USB_HOST_AUDIO_V1_StreamingInterfaceChannelNumbersGet	Returns the number of channels of the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamingInterfaceDirectionGet	Returns the direction of the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamingInterfaceFormatTagGet	Returns the format tag of the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamingInterfaceGetFirst	Gets the first streaming interface object from the attached Audio Device.
	USB_HOST_AUDIO_V1_StreamingInterfaceGetNext	Gets the next streaming interface object from the attached Audio Device.
	USB_HOST_AUDIO_V1_StreamingInterfaceSamplingFrequenciesGet	Returns the sampling frequencies supported by the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamingInterfaceSamplingFrequencyTypeGet	Returns the sampling frequency type of the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamingInterfaceSet	Schedules a SET_INTERFACE request to the specified audio stream.
	USB_HOST_AUDIO_V1_StreamingInterfaceSettingGetFirst	Gets the first streaming interface setting object within an audio streaming interface.
	USB_HOST_AUDIO_V1_StreamingInterfaceSettingGetNext	Gets the next streaming interface setting object within an audio streaming interface.
	USB_HOST_AUDIO_V1_StreamingInterfaceSubFrameSizeGet	Returns the sub-frame size of the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamingInterfaceTerminalLinkGet	Returns the terminal link of the specified streaming interface setting.
	USB_HOST_AUDIO_V1_StreamOpen	Opens the specified audio stream.
	USB_HOST_AUDIO_V1_StreamRead	Schedules an audio stream read request for the specified audio stream.
	USB_HOST_AUDIO_V1_StreamSamplingFrequencyGet	Schedules an audio stream get sampling rate request for the specified audio stream.
	USB_HOST_AUDIO_V1_StreamSamplingFrequencySet	Schedules an audio stream set sampling rate request for the specified audio stream.
	USB_HOST_AUDIO_V1_StreamWrite	Schedules an audio stream write request for the specified audio stream.
	USB_HOST_AUDIO_V1_TerminalAssociationGet	Returns the associated terminal ID of the audio control terminal.
	USB_HOST_AUDIO_V1_TerminalIDGet	Returns the terminal ID of the audio control entity.
	USB_HOST_AUDIO_V1_TerminalInputChannelConfigGet	Returns a structure that describes the spatial location of the logical channels of in the terminal's output audio channel cluster.
	USB_HOST_AUDIO_V1_TerminalInputChannelNumbersGet	Returns the number of logical output channels in the terminal's output audio channel cluster.
	USB_HOST_AUDIO_V1_TerminalSourceIDGet	Returns the ID of the unit or terminal to which this terminal is connected.
	USB_HOST_AUDIO_V1_TerminalTypeGet	Returns the terminal type of the audio control entity.

Macros

	Name	Description
	USB_HOST_AUDIO_V1_0_AttachEventHandlerSet	Sets an attach/detach event handler.
	USB_HOST_AUDIO_V1_0_DeviceObjHandleGet	Returns the device object handle for this Audio v1.0 Device.
	USB_HOST_AUDIO_V1_0_DIRECTION_IN	This is macro USB_HOST_AUDIO_V1_0_DIRECTION_IN.
	USB_HOST_AUDIO_V1_0_DIRECTION_OUT	This is macro USB_HOST_AUDIO_V1_0_DIRECTION_OUT.
	USB_HOST_AUDIO_V1_0_EVENT	Identifies the possible events that the Audio v1.0 Class Driver can generate.
	USB_HOST_AUDIO_V1_0_EVENT_ATTACH	This is macro USB_HOST_AUDIO_V1_0_EVENT_ATTACH.
	USB_HOST_AUDIO_V1_0_EVENT_DETACH	This is macro USB_HOST_AUDIO_V1_0_EVENT_DETACH.
	USB_HOST_AUDIO_V1_0_INTERFACE	USB HOST Audio Client Driver interface.
	USB_HOST_AUDIO_V1_0_OBJ	Defines the type of the Audio v1.0 Host client object.

	USB_HOST_AUDIO_V1_0_REQUEST_HANDLE	USB Host Audio v1.0 Client Driver request handle.
	USB_HOST_AUDIO_V1_0_REQUEST_HANDLE_INVALID	USB Host Audio v1.0 Client Driver invalid request handle.
	USB_HOST_AUDIO_V1_0_STREAM_DIRECTION	USB Host Audio v1.0 Class Driver stream direction.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE_DATA	This is macro USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE_DATA.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE	Returns the type of the USB Audio v1.0 Host Client Driver event handler.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE_NONE	Returns the type of the USB Host Audio v1.0 stream event handler.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE_DATA	USB Host Audio v1.0 class stream transfer event data.
	USB_HOST_AUDIO_V1_0_STREAM_HANDLE	Defines the type of the Audio v1.0 Host stream handle.
	USB_HOST_AUDIO_V1_0_STREAM_HANDLE_INVALID	Defines the type of the Audio v1.0 Host stream invalid handle.
	USB_HOST_AUDIO_V1_0_STREAM_TRANSFER_HANDLE	USB Host Audio v1.0 Class Driver transfer handle.
	USB_HOST_AUDIO_V1_0_STREAM_TRANSFER_HANDLE_INVALID	USB Host Audio v1.0 Class Driver invalid transfer handle definition.
	USB_HOST_AUDIO_V1_0_StreamClose	Closes the audio stream.
	USB_HOST_AUDIO_V1_0_StreamOpen	Opens the specified audio stream.
	USB_HOST_AUDIO_V1_INTERFACE	USB HOST Audio v1.0 Client Driver interface.
	USB_HOST_AUDIO_V1_REQUEST_HANDLE_INVALID	USB Host Audio v1.0 Client Driver invalid request handle.
	USB_HOST_AUDIO_V1_SAMPLING_FREQUENCIES_NUMBER	This structure defines USB Host audio stream information structure.
	USB_HOST_AUDIO_V1_STREAM_HANDLE_INVALID	Defines Audio v1.0 Host stream invalid handle.
	USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE_INVALID	USB Host Audio v1.0 Class Driver invalid stream data transfer handle.

Structures

	Name	Description
	USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE_DATA	USB Host Audio v1.0 class stream control event data.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE_DATA	USB Host Audio v1.0 class stream control event data.
	USB_HOST_AUDIO_V1_0_STREAM_INFO	This is type USB_HOST_AUDIO_V1_0_STREAM_INFO.
	USB_HOST_AUDIO_V1_STREAM_EVENT_INTERFACE_SET_COMPLETE_DATA	USB Host Audio v1.0 class stream control event data.
	USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE_DATA	USB Host Audio v1.0 class stream data transfer event data.
	USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_GET_COMPLETE_DATA	USB Host Audio v1.0 class stream control event data.
	USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE_DATA	USB Host Audio v1.0 class stream control event data.
	USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE_DATA	USB Host Audio v1.0 class stream data transfer event data.

Types

	Name	Description
	USB_HOST_AUDIO_V1_0_ATTACH_EVENT_HANDLER	USB Host Audio v1.0 Client Driver attach event handler function pointer type.
	USB_HOST_AUDIO_V1_0_CONTROL_CALLBACK	USB Host Audio v1.0 Class Driver control transfer complete callback function pointer type.
	USB_HOST_AUDIO_V1_0_STREAM_EVENT_HANDLER	USB Host Audio v1.0 Class Driver stream event handler function pointer type.
	USB_HOST_AUDIO_V1_0_STREAM_OBJ	Defines the type of the Audio v1.0 Host stream object.
	USB_HOST_AUDIO_V1_ATTACH_EVENT_HANDLER	USB Host Audio v1.0 Client Driver attach event handler function pointer type.

	USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ	Defines the type of the Audio v1.0 Host control entity object.
	USB_HOST_AUDIO_V1_ENTITY_REQUEST_CALLBACK	USB Host Audio v1.0 class driver control transfer complete callback function pointer type.
	USB_HOST_AUDIO_V1_OBJ	Defines the type of the Audio v1.0 Host client object.
	USB_HOST_AUDIO_V1_REQUEST_HANDLE	USB Host Audio v1.0 Client Driver request handle.
	USB_HOST_AUDIO_V1_STREAM_EVENT_HANDLER	USB Host Audio v1.0 Class Driver stream event handler function pointer type.
	USB_HOST_AUDIO_V1_STREAM_HANDLE	Defines the type of the Audio v1.0 Host stream handle.
	USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE	USB Host Audio v1.0 Class Driver stream data transfer handle.
	USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ	Defines the type of the Audio v1.0 Host streaming interface object.
	USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ	Defines the type of the Audio v1.0 Host streaming interface setting object.

Description

USB Host Audio v1.0 Class Driver Interface Definition

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB Host Audio v1.0 Class Driver.

File Name

usb_host_audio_v1_0.h

Company

Microchip Technology Inc.

usb_host_audio_v1_0_config_template.h

USB host Audio v1.0 Class configuration definitions template

Macros

	Name	Description
	USB_HOST_AUDIO_V1_ATTACH_LISTENERS_NUMBER	Defines the number of attach event listeners that can be registered with Audio v1.0 Host Client Driver.
	USB_HOST_AUDIO_V1_INSTANCES_NUMBER	Specifies the number of Audio v1.0 instances.
	USB_HOST_AUDIO_V1_STREAMING_INTERFACE_ALTERNATE_SETTINGS_NUMBER	Defines maximum number of alternate settings per Streaming interface provided by any Device that will be connected to this Audio Host.
	USB_HOST_AUDIO_V1_STREAMING_INTERFACES_NUMBER	Defines the maximum number of streaming interfaces could be present in an Audio v1.0 device that this Audio v1.0 Host Client Driver can support.

Description

USB Host Audio v1.0 Class Configuration Definitions

This file contains configurations macros needed to configure the Audio v1.0 host Driver. This file is a template file only. It should not be included by the application. The configuration macros defined in the file should be defined in the configuration specific system_config.h.

File Name

usb_host_cdc_config_template.h

Company

Microchip Technology Inc.

USB CDC Host Library

This section describes the USB CDC Host Library.

Introduction

Introduces the MPLAB Harmony USB CDC Host Library.

Description

The CDC Host Client Driver in the MPLAB Harmony USB Host Stack allows USB Host applications to support and interact with Communications Device Class (CDC) USB devices. The CDC Host Client Driver has the following features:

- Supports CDC ACM devices
- Supports CDC device matching at both the device descriptor and interface descriptor level
- Supports composite CDC devices (multiple CDC interfaces or CDC with other device classes)
- Designed to support multi-client operation
- RTOS ready
- An event driver non-clocking application interaction model
- Allows the application to send CDC ACM commands to the device
- Supports queuing of read and write data transfers

Using the Library

This topic describes the basic architecture of the USB CDC Host Client Driver Library and provides information and examples on its use.

Abstraction Model

Describes the Abstraction Model of the USB CDC Host Client Driver Library.

Description

The CDC Host Client Driver interacts with the Host Layer to control the attached CDC device. The USB Host Layer attaches the CDC Host Client Driver to the CDC device when it meets the matching criteria specified in the USB Host TPL table. The CDC Host Client Driver abstracts the details of sending CDC class specific control transfer commands by providing easy to use non-blocking API to send these command. A command, when issued, is assigned a request handle. This request handle is returned in the event that is generated when the command has been processed, and can be used by the application to track the command. The class specific command functions are implemented in `usb_host_cdc_acm.c`.

While transferring data over the data interface, the CDC Host Client Driver abstracts details such as the bulk interface, endpoints and endpoint size. The CDC Host Client Driver internally (and without application intervention) validates the CDC class specific device descriptors and opens communication pipes. While transferring data, multiple read and write requests can be queued. Each such request gets assigned a transfer handle. The transfer handle for a transfer request is returned along with the completion event for that transfer request. The data transfer routines are implemented in `usb_host_cdc.c`.

Library Overview

The USB CDC Host Client Driver API is grouped functionally, as shown in the following table.

Library Interface Section	Description
Client Access Functions	These functions allow application clients to open, close the client and register event handlers. These functions are implemented in `usb_host_cdc.c`.
Data Transfer Functions	These functions allow the application client to transfer data to the attached device. These functions are implemented in `usb_host_cdc.c`.
CDC Class-specific Command Functions	These functions allow the application to send class specific control transfer requests to the application. These functions are implemented in `usb_host_cdc_acm.c`.

How the Library Works

Describes how the Library works and how it should be used.

Description

The CDC Host Client Driver provides the user application with an easy-to-use interface to the attached CDC device. The USB Host Layer initializes the CDC Host Client Driver when a device is attached. This process does not require application intervention.

The following sections describe the steps and methods required for the user application to interact with the attached devices:

- TPL Table Configuration for CDC Devices
- Detecting Device Attach
- Opening the CDC Host Client Driver
- Sending Class-specific Control Transfers
- Reading and Writing Data
- Event Handling

TPL Table Configuration for CDC Devices

Provides information on configuring the TPL table for CDC devices.

Description

The Host Layer attaches the CDC Host Client Driver to a device when the device class, subclass, protocol in the device descriptor or when the class, subclass and protocol fields in the Interface Association Descriptor (IAD) or Interface descriptor matches the entry in the TPL table. When specifying the entry for the CDC device, the entry for the CDC device, the driver interface must be set to USB_HOST_CDC_INTERFACE. This will attach the CDC Host Client Driver to the device when the USB Host matches the TPL entry to the device. The following code shows possible TPL table options for matching CDC Devices.

Example:

```
/* This code shows an example of TPL table entries for supporting CDC
 * devices. Note the driver interface is set to USB_HOST_CDC_INTERFACE. This
 * will load the CDC Host Client Driver when there is TPL match */

const USB_HOST_TPL_ENTRY USBTPList[1] =
{
    /* This entry looks for any CDC device. The CDC Host Client Driver will
     * check if this is an ACM device and will then load itself */
    TPL_INTERFACE_CLASS_SUBCLASS_PROTOCOL(USB_CDC_CLASS_CODE, USB_CDC_SUBCLASS_CODE, 0x0 , NULL,
USB_HOST_CDC_INTERFACE),

    /* This entry looks specifically for the communications class protocol.
     * This entry should be used if the host application is expected to support
     * CDC as a part of an composite device */
    TPL_INTERFACE_CLASS_SUBCLASS_PROTOCOLUSB_CDC_COMMUNICATIONS_INTERFACE_CLASS_CODE,
USB_CDC_SUBCLASS_ABSTRACT_CONTROL_MODEL, USB_CDC_PROTOCOL_AT_V250 , NULL,  USB_HOST_CDC_INTERFACE),

};
```

Detecting Device Attach

Describes how to register an Attach Event Handler.

Description

The application will need to know when a CDC Device is attached. To receive this attach event from the CDC Host Client Driver, the application must register an Attach Event Handler by calling the USB_HOST_CDC_AttachEventHandlerSet function. This function should be called before the USB_HOST_BusEnable function is called, else the application may miss CDC attach events. It can be called multiple times to register multiple event handlers, each for different application clients that need to know about CDC Device Attach events.

The total number of event handlers that can be registered is defined by USB_HOST_CDC_ATTACH_LISTENERS_NUMBER configuration option in `system_config.h`. When a device is attached, the CDC Host Client Driver will send the attach event to all the registered event handlers. In this event handler, the CDC Host Client Driver will pass a USB_HOST_CDC_OBJ that can be opened to gain access to the device. The following code shows an example of how to register attach event handlers.

Example:

```
/* This code shows an example of CDC Attach Event Handler and how this
 * attach event handler can be registered with the CDC Host Client Driver */

void APP_USBHostCDCAttachEventListener(USB_HOST_CDC_OBJ cdcObj, uintptr_t context)
{
    /* This function gets called when the CDC device is attached. In this
     * example we let the application know that a device is attached and we
     * store the CDC device object. This object will be required to open the
     * device. */

    appData.deviceIsAttached = true;
    appData.cdcObj = cdcObj;
}
```

```
void APP_Tasks(void)
{
    switch (appData.state)
    {
        case APP_STATE_BUS_ENABLE:

            /* In this state the application enables the USB Host Bus. Note
             * how the CDC Attach event handler is registered before the bus
             * is enabled. */

            USB_HOST_CDC_AttachEventHandlerSet(APP_USBHostCDCAttachEventListener, (uintptr_t) 0);
            USB_HOST_BusEnable(0);
            appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
            break;

        case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:
            /* Here we wait for the bus enable operation to complete. */
            break;
    }
}
```

Opening the CDC Host Client Driver

Describes how an application can open the CDC Host Client Driver.

Description

The application must open the CDC Host Client Driver to communicate and control the attached device. The device can be opened by using the USB_HOST_CDC_Open function and specifying the USB_HOST_CDC_OBJ object that was returned in the attached event handler. If the open function fails, it returns an invalid handle (USB_HOST_CDC_HANDLE_INVALID). Once opened successfully, a valid handle tracks the relationship between the client and the CDC Host Client Driver. This handle should be used with other CDC Host Client Driver functions to specify the instance of the CDC Host Client Driver being accessed.

A CDC Host Client Driver instance can be opened multiple times by different application clients. In an ROTS based application each client could running its own thread. Multiple clients can read write data to the one CDC device. In such a case, the read and write requests are queued. The following code shows an example of how the CDC Driver is opened.

Example:

```
/* This code shows an example of the how to open the CDC Host Client
 * driver. The application state machine waits for a device attach and then
 * opens the CDC Host Client Driver. */

void APP_USBHostCDCAttachEventListener(USB_HOST_CDC_OBJ cdcObj, uintptr_t context)
{
    /* This function gets called when the CDC device is attached. Update the
     * application data structure to let the application know that this device
     * is attached */

    appData.deviceIsAttached = true;
    appData.cdcObj = cdcObj;
}

void APP_Tasks(void)
{
    switch (appData.state)
    {
        case APP_STATE_BUS_ENABLE:

            /* In this state the application enables the USB Host Bus. Note
             * how the CDC Attach event handler are registered before the bus
             * is enabled. */

            USB_HOST_EventHandlerSet(APP_USBHostEventHandler, (uintptr_t)0);
            USB_HOST_CDC_AttachEventHandlerSet(APP_USBHostCDCAttachEventListener, (uintptr_t) 0);
            USB_HOST_BusEnable(0);
            appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
            break;

        case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:
```

```
            /* In this state we wait for the Bus enable to complete */
            if(USB_HOST_BusIsEnabled(0))
            {
                appData.state = APP_STATE_WAIT_FOR_DEVICE_ATTACH;
            }
            break;

        case APP_STATE_WAIT_FOR_DEVICE_ATTACH:

            /* In this state the application is waiting for the device to be
             * attached */
            if(appData.deviceIsAttached)
            {
                /* A device is attached. We can open this device */
                appData.state = APP_STATE_OPEN_DEVICE;
                appData.deviceIsAttached = false;
            }
            break;

        case APP_STATE_OPEN_DEVICE:

            /* In this state the application opens the attached device */
            appData.cdcHostHandle = USB_HOST_CDC_Open(appData.cdcObj);
            if(appData.cdcHostHandle != USB_HOST_CDC_HANDLE_INVALID)
            {
                /* The driver was opened successfully. Set the event handler
                 * and then go to the next state. */
                USB_HOST_CDC_EventHandlerSet(appData.cdcHostHandle,
                                             APP_USBHostCDCEventHandler,
                                            (uintptr_t)0);
                appData.state = APP_STATE_SET_LINE_CODING;
            }
            break;

        default:
            break;
    }
}
```

Sending Class-specific Control Transfers

Describes how the application client can send CDC Class-specific commands to the connected device.

Description

The CDC Host Client Driver allows the application client to send CDC Class specific commands to the connected device. These commands allows the application client to:

- Set the device line coding (USB_HOST_CDC_LineCodingSet)
- Retrieve the device line coding (USB_HOST_CDC_LineCodingGet)
- Set the device control line state (USB_HOST_CDC_ControlLineStateSet)
- Ask the device to send a break signal (USB_HOST_CDC_BreakSend)

These functions are non-blocking. The functions will return before the actual command execution is complete. The return value indicates if the command was scheduled successfully, or if the driver is busy and cannot accept commands, or if the command failed due to an unknown reason. If the command failed because the driver was busy, it can be retried. If scheduled successfully, the function will return a valid request handle. This request handle is unique and tracks the requested command.

When the command related control transfer has completed, the CDC Host Client Driver generates a command specific completion event. This event is accompanied by a data structure that contains information about the completion of the command. The request handler generated at the time of calling the command request function is also returned along with the event. The request handle expires after the event handler exits. The following tables show the command functions, along with the respective events and the type of the event related data.

Table 1: Set Line Coding

Function	USB_HOST_CDC_ACM_LineCodingSet
Event	USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE
Event Data Type	USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA

Table 2: Get Line Coding

Function	USB_HOST_CDC_ACM_LineCodingGet
Event	USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE
Event Data Type	USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA

Table 3: Set Control Line State

Function	USB_HOST_CDC_ACM_ControlLineStateSet
Event	USB_HOST_CDC_EVENT_ACM_CONTROL_LINE_STATE_SET_COMPLETE
Event Data Type	USB_HOST_CDC_EVENT_ACM_CONTROL_LINE_STATE_SET_COMPLETE _DATA

Table 4: Send Break

Function	USB_HOST_CDC_ACM_SendBreak
Event	USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE
Event Data Type	USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE _DATA

The following code shows an example of sending a CDC class specific commands. Refer to the Event Handling section for details on setting the event handler function.

Example:

```
/* This code shows an example of how to send CDC Class specific command
 * requests. The event handling related to each command is also shown. */

USB_HOST_CDC_EVENT_RESPONSE APP_USBHostCDCEventHandler
(
    USB_HOST_CDC_HANDLE cdcHandle,
    USB_HOST_CDC_EVENT event,
    void * eventData,
    uintptr_t context
)
{
    /* This function is called when a CDC Host event has occurred. A pointer to
     * this function is registered after opening the device. See the call to
     * USB_HOST_CDC_EventHandlerSet() function. */

    USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA * setLineCodingEventData;
    USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA * setControlLineStateEventData;
    USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA * writeCompleteEventData;
    USB_HOST_CDC_EVENT_READ_COMPLETE_DATA * readCompleteEventData;

    switch(event)
    {
        case USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE:

            /* This means the application requested Set Line Coding request is
             * complete. */
            setLineCodingEventData = (USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA *)(eventData);
            appData.controlRequestDone = true;
            appData.controlRequestResult = setLineCodingEventData->result;
            break;

        case USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE:

            /* This means the application requested Set Control Line State
             * request has completed. */
            setControlLineStateEventData = (USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA
*)(eventData);
            appData.controlRequestDone = true;
            appData.controlRequestResult = setControlLineStateEventData->result;
            break;

        default:
            break;

    }
```

```
}

void APP_Tasks(void)
{

    switch(appData.state)
    {
        /* The application states that enable the bus and wait for device attach are
         * not shown here for brevity */

        case APP_STATE_OPEN_DEVICE:

            /* In this state the application opens the attached device */
            appData.cdcHostHandle = USB_HOST_CDC_Open(appData.cdcObj);
            if(appData.cdcHostHandle != USB_HOST_CDC_HANDLE_INVALID)
            {
                /* The driver was opened successfully. Set the event handler
                 * and then go to the next state. */
                USB_HOST_CDC_EventHandlerSet(appData.cdcHostHandle, APP_USBHostCDCEventHandler,
(uintptr_t)0);
                appData.state = APP_STATE_SET_LINE_CODING;
            }
            break;

        case APP_STATE_SET_LINE_CODING:

            /* Here we set the Line coding. The control request done flag will
             * be set to true when the control request has completed. */

            appData.controlRequestDone = false;
            result = USB_HOST_CDC_ACM_LineCodingSet(appData.cdcHostHandle, NULL,
&appData.cdcHostLineCoding);

            if(result == USB_HOST_CDC_RESULT_SUCCESS)
            {
                /* We wait for the set line coding to complete */
                appData.state = APP_STATE_WAIT_FOR_SET_LINE_CODING;
            }

            break;

        case APP_STATE_WAIT_FOR_SET_LINE_CODING:

            if(appData.controlRequestDone)
            {
                if(appData.controlRequestResult != USB_HOST_CDC_RESULT_SUCCESS)
                {
                    /* The control request was not successful. */
                    appData.state = APP_STATE_ERROR;
                }
                else
                {
                    /* Next we set the Control Line State */
                    appData.state = APP_STATE_SEND_SET_CONTROL_LINE_STATE;
                }
            }
            break;

        case APP_STATE_SEND_SET_CONTROL_LINE_STATE:

            /* Here we set the control line state */
            appData.controlRequestDone = false;
            result = USB_HOST_CDC_ACM_ControlLineStateSet(appData.cdcHostHandle, NULL,
                    &appData.controlLineState);

            if(result == USB_HOST_CDC_RESULT_SUCCESS)
            {
                /* We wait for the set line coding to complete */
                appData.state = APP_STATE_WAIT_FOR_SET_CONTROL_LINE_STATE;
```

```
            }

            break;

        case APP_STATE_WAIT_FOR_SET_CONTROL_LINE_STATE:

            /* Here we wait for the control line state set request to complete */
            if(appData.controlRequestDone)
            {
                if(appData.controlRequestResult != USB_HOST_CDC_RESULT_SUCCESS)
                {
                    /* The control request was not successful. */
                    appData.state = APP_STATE_ERROR;
                }
                else
                {
                    /* Next we set the Control Line State */
                    appData.state = APP_STATE_SEND_PROMPT_TO_DEVICE;
                }
            }

            break;

        default:
            break;
    }
}
```

Reading and Writing Data

Describes how to transfer data to the attached CDC device.

Description

The application can use the USB_HOST_CDC_Read and USB_HOST_CDC_Write functions to transfer data to the attached CDC device. While calling these function, the client handle specifies the target CDC device and the event handler function to which the events should be sent. It is possible for multiple client to open the same instance of the CDC Host Client Driver instance and transfer data to the attached CDC Device.

Calling the USB_HOST_CDC_Read and USB_HOST_CDC_Write functions while a read/write transfer is already in progress will cause the transfer result to be queued. If the transfer was successfully queued or scheduled, the USB_HOST_CDC_Read and USB_HOST_CDC_Write functions will return a valid transfer handle. This transfer handle identifies the transfer request. The application clients can use the transfer handles to keep track of multiple queued transfers. When a transfer completes, the CDC Host Client Driver generates an event. The following tables shows the event and the event data associated with the event.

Table 1: Read

Function	USB_HOST_CDC_Read
Event	USB_HOST_CDC_EVENT_READ_COMPLETE
Event Data Type	USB_HOST_CDC_EVENT_READ_COMPLETE_DATA

Table 2: Write

Function	USB_HOST_CDC_ACM_LineCodingGet
Event	USB_HOST_CDC_EVENT_READ_COMPLETE
Event Data Type	USB_HOST_CDC_EVENT_READ_COMPLETE_DATA

The event data contains information on the amount of data transferred, completion status and the transfer handle of the transfer. The following code shows an example of reading and writing data.

Example:

```
/* In this code example, the USB_HOST_CDC_Read and the USB_HOST_CDC_Write
 * functions are used to read and write data. The event related to the read and
 * write operations are handled in the APP_USBHostCDCEventHandler function. */

USB_HOST_CDC_EVENT_RESPONSE APP_USBHostCDCEventHandler
(
    USB_HOST_CDC_HANDLE cdcHandle,
    USB_HOST_CDC_EVENT event,
    void * eventData,
```

```
    uintptr_t context
)
{
    /* This function is called when a CDC Host event has occurred. A pointer to
     * this function is registered after opening the device. */

    USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA * writeCompleteEventData;
    USB_HOST_CDC_EVENT_READ_COMPLETE_DATA * readCompleteEventData;

    switch(event)
    {
        case USB_HOST_CDC_EVENT_WRITE_COMPLETE:

            /* This means an application requested write has completed */
            appData.writeTransferDone = true;
            writeCompleteEventData = (USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA *)(eventData);
            appData.writeTransferResult = writeCompleteEventData->result;
            break;

        case USB_HOST_CDC_EVENT_READ_COMPLETE:

            /* This means an application requested write has completed */
            appData.readTransferDone = true;
            readCompleteEventData = (USB_HOST_CDC_EVENT_READ_COMPLETE_DATA *)(eventData);
            appData.readTransferResult = readCompleteEventData->result;
            break;

        default:
            break;
    }

    return(USB_HOST_CDC_EVENT_RESPONE_NONE);
}

void APP_Tasks(void)
{
    switch(appData.state)
    {
        /* The application states that wait for device attach and open the CDC
         * Host Client Driver are not shown here for brevity */

        case APP_STATE_SEND_PROMPT_TO_DEVICE:

            /* The prompt is sent to the device here. The write transfer done
             * flag is updated in the event handler. */

            appData.writeTransferDone = false;
            result = USB_HOST_CDC_Write(appData.cdcHostHandle, NULL, prompt, 8);

            if(result == USB_HOST_CDC_RESULT_SUCCESS)
            {
                appData.state = APP_STATE_WAIT_FOR_PROMPT_SEND_COMPLETE;
            }
            break;

        case APP_STATE_WAIT_FOR_PROMPT_SEND_COMPLETE:

            /* Here we check if the write transfer is done */
            if(appData.writeTransferDone)
            {
                if(appData.writeTransferResult == USB_HOST_CDC_RESULT_SUCCESS)
                {
                    /* Now to get data from the device */
                    appData.state = APP_STATE_GET_DATA_FROM_DEVICE;
                }
                else
                {
                    /* Try sending the prompt again. */
                    appData.state = APP_STATE_SEND_PROMPT_TO_DEVICE;
```

```
                }
            }

            break;

        case APP_STATE_GET_DATA_FROM_DEVICE:

            /* Here we request data from the device */
            appData.readTransferDone = false;
            result = USB_HOST_CDC_Read(appData.cdcHostHandle, NULL, appData.inDataArray, 1);
            if(result == USB_HOST_CDC_RESULT_SUCCESS)
            {
                appData.state = APP_STATE_WAIT_FOR_DATA_FROM_DEVICE;
            }
            break;

        case APP_STATE_WAIT_FOR_DATA_FROM_DEVICE:

            /* Wait for data from device. */
            if(appData.readTransferDone)
            {
                if(appData.readTransferResult == USB_HOST_CDC_RESULT_SUCCESS)
                {
                    /* Do something with the data here */
                }
            }

            break;

        default:
            break;
    }
}
```

Event Handling

Describes how to set event handlers.

Description

The CDC Host Client Driver presents an event driven interface to the application. The CDC Host Client Driver requires the application client to set two event handlers for meaningful operation:

- The Attach event handler is not client specific and is registered before the USB_HOST_BusEnable function is called. This event handler and the attach event is discussed in the Detecting Device Attach section.
- The client specific command, data transfer and detach events. The CDC Class specific command request events are discussed in the Sending Class Specific Control Transfers section. The data transfer related events are discussed in the Reading and Writing Data section. Some general points about these events are discussed below.

A request to send a command or transfer data typically completes after the command request or transfer function has exited. The application must then use the CDC Host Client Driver event to track the completion of this command or data transfer request. In a case where multiple data transfers are queued, the transfer handles can be used to identify the transfer requests.

The application must use the USB_HOST_CDC_EventHandlerSet function to register a client specific event handler. This event handler will be called when a command, data transfer or detach event has occurred and should be registered before the client request for command or a data transfer. The following code shows an example of registering an event handler.

Example:

```
/* This code shows an example of setting an event handler and an example
 * event handler. For the full set of events that the CDC Host Client generates,
 * refer to USB_HOST_CDC_EVENT enumeration description */

USB_HOST_CDC_EVENT_RESPONSE APP_USBHostCDCEventHandler
(
    USB_HOST_CDC_HANDLE cdcHandle,
    USB_HOST_CDC_EVENT event,
    void * eventData,
    uintptr_t context
)
{
    /* This function is called when a CDC Host event has occurred. A pointer to
     * this function is registered after opening the device. See the call to
```

```
     * USB_HOST_CDC_EventHandlerSet() function. */

    USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA * setLineCodingEventData;
    USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA * setControlLineStateEventData;
    USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA * writeCompleteEventData;
    USB_HOST_CDC_EVENT_READ_COMPLETE_DATA * readCompleteEventData;

    switch(event)
    {
        case USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE:

            /* This means the application requested Set Line Coding request is
             * complete. */
            setLineCodingEventData = (USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA *)(eventData);
            appData.controlRequestDone = true;
            appData.controlRequestResult = setLineCodingEventData->result;
            break;

        case USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE:

            /* This means the application requested Set Control Line State
             * request has completed. */
            setControlLineStateEventData = (USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA
*)(eventData);
            appData.controlRequestDone = true;
            appData.controlRequestResult = setControlLineStateEventData->result;
            break;

        case USB_HOST_CDC_EVENT_WRITE_COMPLETE:

            /* This means an application requested write has completed */
            appData.writeTransferDone = true;
            writeCompleteEventData = (USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA *)(eventData);
            appData.writeTransferResult = writeCompleteEventData->result;
            break;

        case USB_HOST_CDC_EVENT_READ_COMPLETE:

            /* This means an application requested write has completed */
            appData.readTransferDone = true;
            readCompleteEventData = (USB_HOST_CDC_EVENT_READ_COMPLETE_DATA *)(eventData);
            appData.readTransferResult = readCompleteEventData->result;
            break;

        case USB_HOST_CDC_EVENT_DEVICE_DETACHED:

            /* The device was detached */
            appData.deviceWasDetached = true;
            break;

        default:
            break;
    }

    return(USB_HOST_CDC_EVENT_RESPONE_NONE);
}


void APP_Tasks(void)
{
    switch(appData.state)
    {
        /* The application states that enable the bus and wait for device attach
         * are not shown here for brevity */
        case APP_STATE_OPEN_DEVICE:

            /* In this state the application opens the attached device */
            appData.cdcHostHandle = USB_HOST_CDC_Open(appData.cdcObj);
            if(appData.cdcHostHandle != USB_HOST_CDC_HANDLE_INVALID)
```

```
            {
                /* The driver was opened successfully. Set the event handler
                 * and then go to the next state. */
                USB_HOST_CDC_EventHandlerSet(appData.cdcHostHandle, APP_USBHostCDCEventHandler,
(uintptr_t)0);
                appData.state = APP_STATE_SET_LINE_CODING;
            }
            break;

        default:
            break;

    }
}
```

Configuring the Library

Describes how to configure the USB CDC Host Library.

Macros

	Name	Description
	USB_HOST_CDC_ATTACH_LISTENERS_NUMBER	Defines the number of attach event listeners that can be registered with CDC Host Client Driver.
	USB_HOST_CDC_INSTANCES_NUMBER	Specifies the number of CDC instances.

Description

The CDC Host Client Driver requires configuration constants to be specified in `system_config.h` file. These constants define the build time configuration (functionality and static resources) of the CDC Host Client Driver.

USB_HOST_CDC_ATTACH_LISTENERS_NUMBER Macro

Defines the number of attach event listeners that can be registered with CDC Host Client Driver.

File

usb_host_cdc_config_template.h

C

```
#define USB_HOST_CDC_ATTACH_LISTENERS_NUMBER
```

Description

USB Host CDC Attach Listeners Number

The USB CDC Host Client Driver provides attach notification to listeners who have registered with the client driver via the USB_HOST_CDC_AttachEventHandlerSet() function. The USB_HOST_CDC_ATTACH_LISTENERS_NUMBER configuration constant defines the maximum number of event handlers that can be set. This number should be set to equal the number of entities that interested in knowing when a CDC device is attached.

Remarks

None.

USB_HOST_CDC_INSTANCES_NUMBER Macro

Specifies the number of CDC instances.

File

usb_host_cdc_config_template.h

C

```
#define USB_HOST_CDC_INSTANCES_NUMBER
```

Description

USB Host CDC Maximum Number of Instances

This macro defines the number of instances of the CDC host Driver. For example, if the application needs to implement two instances of the CDC host Driver should be set to 2.

Remarks

None.

Building the Library

Describes the files to be included in the project while using the USB CDC Host Client Driver.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/usb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
usb_host_cdc.h	This header file should be included in any `.c` file that accesses the CDC Host Client Driver API.
sub_host_cdc_acm.h	This header file should be included in any `.c` file that accesses the CDC Host Client Driver command request API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/usb_host_cdc.c	This file implements the CDC Host Client Driver interface and should be included in the project if the CDC Host Client Driver operation is desired.
/src/dynamic/usb_host_cdc_acm.c	This file implements the CDC Host Client Driver command request functions and should be included if any class specific function must be called.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The USB CDC Host Library depends on the following modules:

- USB Host Layer Library

Library Interface

a) Client Access Functions

	Name	Description
	USB_HOST_CDC_Open	This function opens the specified CDC device.
	USB_HOST_CDC_Close	This function closes the CDC device.
	USB_HOST_CDC_AttachEventHandlerSet	This function will set an attach event handler.
	USB_HOST_CDC_EventHandlerSet	Registers an event handler with the CDC Host Client Driver.
	USB_HOST_CDC_DeviceObjHandleGet	This function returns the Device Object Handle for this CDC device.

b) Data Transfer Functions

	Name	Description
	USB_HOST_CDC_Read	This function will read data from the attached device.
	USB_HOST_CDC_SerialStateNotificationGet	This function will request Serial State Notification from the attached device.
	USB_HOST_CDC_Write	This function will write data to the attached device.

c) CDC Class-specific Functions

	Name	Description
	USB_HOST_CDC_ACM_BreakSend	This function sends a request to the attached device to update its break duration.
	USB_HOST_CDC_ACM_ControlLineStateSet	This function sends a request to the attached device to set its Control Line State.
	USB_HOST_CDC_ACM_LineCodingGet	This function sends a request to the attached device to get its Line Coding.
	USB_HOST_CDC_ACM_LineCodingSet	This function sends a request to the attached device to set its Line Coding.

d) Data Types and Constants

	Name	Description
	USB_HOST_CDC_EVENT	Identifies the possible events that the CDC Class Driver can generate.
	USB_HOST_CDC_RESULT	USB Host CDC Client Driver Result enumeration.
	USB_HOST_CDC_TRANSFER_HANDLE	USB Host CDC Client Driver Transfer Handle
	USB_HOST_CDC_TRANSFER_HANDLE_INVALID	USB Host CDC Client Driver Invalid Transfer Handle Definition.
	USB_HOST_CDC_ATTACH_EVENT_HANDLER	USB Host CDC Client Driver Attach Event Handler Function Pointer Type.
	USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA	USB Host CDC Client Driver Command Event Data.
	USB_HOST_CDC_HANDLE	Defines the type of the CDC Host Client Driver Handle
	USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE_DATA	USB Host CDC Client Driver Command Event Data.
	USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA	USB Host CDC Client Driver Command Event Data.
	USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA	USB Host CDC Client Driver Command Event Data.
	USB_HOST_CDC_EVENT_HANDLER	USB Host CDC Client Driver Event Handler Function Pointer Type.
	USB_HOST_CDC_EVENT_READ_COMPLETE_DATA	USB Host CDC Client Driver Event Data.
	USB_HOST_CDC_EVENT_RESPONSE	Return type of the USB CDC Host Client Driver Event Handler.
	USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED_DATA	USB Host CDC Client Driver Event Data.
	USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA	USB Host CDC Client Driver Event Data.
	USB_HOST_CDC_OBJ	Defines the type of the CDC Host Client Object.
	USB_HOST_CDC_REQUEST_HANDLE	USB Host CDC Client Driver Request Handle
	USB_HOST_CDC_INTERFACE	USB HOST CDC Client Driver Interface
	USB_HOST_CDC_REQUEST_HANDLE_INVALID	USB Host CDC Client Driver Invalid Request Handle
	USB_HOST_CDC_HANDLE_INVALID	Defines an Invalid CDC Client Driver Handle.

Description

a) Client Access Functions

USB_HOST_CDC_Open Function

This function opens the specified CDC device.

File

usb_host_cdc.h

C

```
USB_HOST_CDC_HANDLE USB_HOST_CDC_Open(USB_HOST_CDC_OBJ cdcDeviceObj);
```

Returns

Will return a valid handle if the device could be opened successfully, else will return USB_HOST_CDC_HANDLE_INVALID. The function will return a valid handle if the device is ready to be opened.

Description

This function will open the specified CDC device. Once opened, the CDC device can be accessed via the handle which this function returns. The cdcDeviceObj parameter is the value returned in the USB_HOST_CDC_ATTACH_EVENT_HANDLER event handling function.

Remarks

None.

Preconditions

The client handle should be valid.

Example

Parameters

Parameters	Description
cdcDeviceObj	CDC device object handle returned in the USB_HOST_CDC_ATTACH_EVENT_HANDLER function.

Function

```
USB_HOST_CDC_HANDLE USB_HOST_CDC_Open
(
    USB_HOST_CDC_OBJ cdcDeviceObj
);
```

USB_HOST_CDC_Close Function

This function closes the CDC device.

File

usb_host_cdc.h

C

```
void USB_HOST_CDC_Close(USB_HOST_CDC_HANDLE cdcDeviceHandle);
```

Returns

None.

Description

This function will close the open CDC device. This closes the association between the application entity that opened the device and device. The driver handle becomes invalid.

Remarks

The device handle becomes invalid after calling this function.

Preconditions

None.

Example

Parameters

Parameters	Description
cdcDeviceHandle	handle to the CDC device obtained from the USB_HOST_CDC_Open() function.

Function

void USB_HOST_CDC_Close
(
USB_HOST_CDC_HANDLE cdcDeviceHandle
);

USB_HOST_CDC_AttachEventHandlerSet Function

This function will set an attach event handler.

File

usb_host_cdc.h

C

```
USB_HOST_CDC_RESULT USB_HOST_CDC_AttachEventHandlerSet(USB_HOST_CDC_ATTACH_EVENT_HANDLER eventHandler,
uintptr_t context);
```

Returns

USB_HOST_CDC_RESULT_SUCCESS - if the attach event handler was registered successfully.

USB_HOST_CDC_RESULT_FAILURE - if the number of registered event handlers has exceeded USB_HOST_CDC_ATTACH_LISTENERS_NUMBER.

Description

This function will set an attach event handler. The attach event handler will be called when a CDC device has been attached. The context will be returned in the event handler. This function should be called before the bus has been enabled.

Remarks

Function should be called before USB_HOST_BusEnable() function is called.

Preconditions

None.

Example

Parameters

Parameters	Description
eventHandler	pointer to the attach event handler
context	an application defined context that will be returned in the event handler.

Function

USB_HOST_CDC_RESULT USB_HOST_CDC_AttachEventHandlerSet
(
USB_HOST_CDC_ATTACH_EVENT_HANDLER eventHandler,
uintptr_t context
);

USB_HOST_CDC_EventHandlerSet Function

Registers an event handler with the CDC Host Client Driver.

File

usb_host_cdc.h

C

```
USB_HOST_CDC_RESULT USB_HOST_CDC_EventHandlerSet(USB_HOST_CDC_HANDLE handle, USB_HOST_CDC_EVENT_HANDLER
eventHandler, uintptr_t context);
```

Returns

USB_HOST_CDC_RESULT_SUCCESS - The operation was successful USB_HOST_CDC_RESULT_HANDLE_INVALID - The specified instance does not exist. USB_HOST_CDC_RESULT_FAILURE - An unknown failure occurred.

Description

This function registers a client specific CDC Host Client Driver event handler. The CDC Host Client Driver will call this function with relevant event and associated event data, in response to command requests and data transfers that have been scheduled by the client.

Remarks

None.

Preconditions

None.

Example

Parameters

Parameters	Description
handle	handle to the CDC Host Client Driver.
eventHandler	A pointer to event handler function. If NULL, then events will not be generated.
context	Application specific context that is returned in the event handler.

Function

```
USB_HOST_CDC_RESULT USB_HOST_CDC_EventHandlerSet
(
    USB_HOST_CDC_HANDLE handle,
    USB_HOST_CDC_EVENT_HANDLER eventHandler,
uintptr_t context
);
```

USB_HOST_CDC_DeviceObjHandleGet Function

This function returns the Device Object Handle for this CDC device.

File

usb_host_cdc.h

C

```
USB_HOST_DEVICE_OBJ_HANDLE USB_HOST_CDC_DeviceObjHandleGet(USB_HOST_CDC_OBJ cdcDeviceObj);
```

Returns

Will return a valid device object handle if the device is still connected to the system. Will return an USB_HOST_DEVICE_OBJ_HANDLE_INVALID otherwise.

Description

This function returns the Device Object Handle for this CDC device. This returned Device Object Handle can be used by the application to perform device level operations such as getting the string descriptors.

Remarks

None.

Preconditions

None.

Example

Parameters

Parameters	Description
cdcDeviceObj	CDC device object handle returned in the USB_HOST_CDC_ATTACH_EVENT_HANDLER function.

Function

USB_HOST_DEVICE_OBJ_HANDLE USB_HOST_CDC_DeviceObjHandleGet
(
USB_HOST_CDC_OBJ cdcDeviceObj
);

b) Data Transfer Functions

USB_HOST_CDC_Read Function

This function will read data from the attached device.

File

usb_host_cdc.h

C

```
USB_HOST_CDC_RESULT USB_HOST_CDC_Read(USB_HOST_CDC_HANDLE handle, USB_HOST_CDC_TRANSFER_HANDLE *
transferHandle, void * data, size_t size);
```

Returns

USB_HOST_CDC_RESULT_SUCCESS - The operation was successful.

USB_HOST_CDC_RESULT_DEVICE_UNKNOWN - The device that this request was targeted to does not exist in the system.

USB_HOST_CDC_RESULT_BUSY - The request could not be scheduled at this time. The client should try again.

USB_HOST_CDC_RESULT_INVALID_PARAMETER - An input parameter was NULL.

USB_HOST_CDC_RESULT_FAILURE - An unknown failure occurred.

USB_HOST_CDC_RESULT_HANDLE_INVALID - The client handle is not valid.

Description

This function will read data from the attached CDC device. The function will try to read size amount of bytes but will stop reading when the device terminates the USB transfer (sends a short packet or a ZLP). If the request was accepted, transferHandle will contain a valid transfer handle, else it will contain USB_HOST_CDC_TRANSFER_HANDLE_INVALID. The completion of the request will be indicated by the USB_HOST_CDC_EVENT_READ_COMPLETE event. The transfer handle will be returned in the event.

Remarks

None.

Preconditions

The client handle should be valid.

Example

Parameters

Parameters	Description
handle	handle to the CDC device instance to which the request should be sent.
transferHandle	Pointer to USB_HOST_CDC_TRANSFER_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.
data	pointer to the buffer where the received data will be stored. The contents of the buffer will be valid only when the USB_HOST_CDC_EVENT_READ_COMPLETE event has occurred.
size	size of the data buffer. Only these many bytes or less will be read.

Function

USB_HOST_CDC_RESULT USB_HOST_CDC_Read
(
USB_HOST_CDC_HANDLE handle,
USB_HOST_CDC_TRANSFER_HANDLE * transferHandle,
void * data,
size_t size

);

USB_HOST_CDC_SerialStateNotificationGet Function

This function will request Serial State Notification from the attached device.

File

usb_host_cdc.h

C

```
USB_HOST_CDC_RESULT USB_HOST_CDC_SerialStateNotificationGet(USB_HOST_CDC_HANDLE handle,
USB_HOST_CDC_TRANSFER_HANDLE * transferHandle, USB_CDC_SERIAL_STATE * serialState);
```

Returns

USB_HOST_CDC_RESULT_SUCCESS - The operation was successful.

USB_HOST_CDC_RESULT_DEVICE_UNKNOWN - The device that this request was targeted to does not exist in the system.

USB_HOST_CDC_RESULT_BUSY - The request could not be scheduled at this time. The client should try again.

USB_HOST_CDC_RESULT_INVALID_PARAMETER - An input parameter was NULL.

USB_HOST_CDC_RESULT_FAILURE - An unknown failure occurred.

USB_HOST_CDC_RESULT_HANDLE_INVALID - The client handle is not valid.

Description

This function will request Serial State Notification from the attached device. If the request was accepted, transferHandle will contain a valid transfer handle, else it will contain USB_HOST_CDC_TRANSFER_HANDLE_INVALID. The completion of the request will be indicated by the USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED event. The transfer handle will be returned in the event.

Remarks

None.

Preconditions

The client handle should be valid.

Example

Parameters

Parameters	Description
handle	handle to the CDC device instance to which the request should be sent.
transferHandle	Pointer to USB_HOST_CDC_TRANSFER_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.
serialState	Pointer to the serial state data structure where the received serial state will be stored.

Function

USB_HOST_CDC_RESULT USB_HOST_CDC_SerialStateNotificationGet

(

USB_HOST_CDC_HANDLE handle,

USB_HOST_CDC_TRANSFER_HANDLE * transferHandle,

USB_CDC_SERIAL_STATE * serialState

);

USB_HOST_CDC_Write Function

This function will write data to the attached device.

File

usb_host_cdc.h

C

```
USB_HOST_CDC_RESULT USB_HOST_CDC_Write(USB_HOST_CDC_HANDLE handle, USB_HOST_CDC_TRANSFER_HANDLE *
transferHandle, void * data, size_t size);
```

Returns

USB_HOST_CDC_RESULT_SUCCESS - The operation was successful.

USB_HOST_CDC_RESULT_DEVICE_UNKNOWN - The device that this request was targeted to does not exist in the system.

USB_HOST_CDC_RESULT_BUSY - The request could not be scheduled at this time. The client should try again.

USB_HOST_CDC_RESULT_INVALID_PARAMETER - An input parameter was NULL.

USB_HOST_CDC_RESULT_FAILURE - An unknown failure occurred.

USB_HOST_CDC_RESULT_HANDLE_INVALID - The client handle is not valid.

Description

This function will write data to the attached CDC device. The function will write size amount of bytes. If the request was accepted, transferHandle will contain a valid transfer handle, else it will contain USB_HOST_CDC_TRANSFER_HANDLE_INVALID. The completion of the request will be indicated by the USB_HOST_CDC_EVENT_WRITE_COMPLETE event. The transfer handle will be returned in the event.

Remarks

None.

Preconditions

The client handle should be valid.

Example

Parameters

Parameters	Description
handle	handle to the CDC device instance to which the request should be sent.
transferHandle	Pointer to USB_HOST_CDC_TRANSFER_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.
data	pointer to the buffer containing the data to be written. The contents of the buffer should not be changed till the USB_HOST_CDC_EVENT_WRITE_COMPLETE event has occurred.
size	Number of bytes to write.

Function

```
USB_HOST_CDC_RESULT USB_HOST_CDC_Write
(
    USB_HOST_CDC_HANDLE handle,
    USB_HOST_CDC_TRANSFER_HANDLE * transferHandle,
void * data,
size_t size
);
```

c) CDC Class-specific Functions

USB_HOST_CDC_ACM_BreakSend Function

This function sends a request to the attached device to update its break duration.

File

usb_host_cdc_acm.h

C

```
USB_HOST_CDC_RESULT USB_HOST_CDC_ACM_BreakSend(USB_HOST_CDC_HANDLE handle, USB_HOST_CDC_REQUEST_HANDLE *
requestHandle, uint16_t breakDuration);
```

Returns

USB_HOST_CDC_RESULT_SUCCESS - The operation was successful.

USB_HOST_CDC_RESULT_DEVICE_UNKNOWN - The device that this request was targeted to does not exist in the system.

USB_HOST_CDC_RESULT_BUSY - The request could not be scheduled at this time. The client should try again.

USB_HOST_CDC_RESULT_INVALID_PARAMETER - An input parameter was NULL.

USB_HOST_CDC_RESULT_FAILURE - An unknown failure occurred.

USB_HOST_CDC_RESULT_HANDLE_INVALID - The client handle is not valid.

Description

This function sends a request to the attached to update its break duration. The function schedules a SEND BREAK control transfer. If successful, the transferHandle parameter will contain a valid request handle, else it will contain USB_HOST_CDC_REQUEST_HANDLE_INVALID. When completed, the CDC client driver will generate a USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE event.

Remarks

None.

Preconditions

The client handle should be valid.

Example

Parameters

Parameters	Description
handle	handle to the CDC device instance to which the request should be sent.
requestHandle	Pointer to USB_HOST_CDC_REQUEST_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.
breakDuration	Break duration.

Function

```
USB_HOST_CDC_RESULT USB_HOST_CDC_ACM_BreakSend
(
    USB_HOST_CDC_HANDLE handle,
    USB_HOST_CDC_REQUEST_HANDLE * requestHandle,
uint16_t breakDuration
);
```

USB_HOST_CDC_ACM_ControlLineStateSet Function

This function sends a request to the attached device to set its Control Line State.

File

usb_host_cdc_acm.h

C

```
USB_HOST_CDC_RESULT USB_HOST_CDC_ACM_ControlLineStateSet(USB_HOST_CDC_HANDLE handle,
USB_HOST_CDC_REQUEST_HANDLE * requestHandle, USB_CDC_CONTROL_LINE_STATE * controlLineState);
```

Returns

USB_HOST_CDC_RESULT_SUCCESS - The operation was successful.

USB_HOST_CDC_RESULT_DEVICE_UNKNOWN - The device that this request was targeted to does not exist in the system.

USB_HOST_CDC_RESULT_BUSY - The request could not be scheduled at this time. The client should try again.

USB_HOST_CDC_RESULT_INVALID_PARAMETER - An input parameter was NULL.

USB_HOST_CDC_RESULT_FAILURE - An unknown failure occurred.

USB_HOST_CDC_RESULT_HANDLE_INVALID - The client handle is not valid.

Description

This function sends a request to the attached to set its Control Line State. The function schedules a SET CONTROL LINE STATE control transfer. If successful, the requestHandle parameter will contain a valid request handle, else it will contain USB_HOST_CDC_REQUEST_HANDLE_INVALID. When completed, the CDC client driver will generate a USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE event.

Remarks

None.

Preconditions

The client handle should be valid.

Example

Parameters

Parameters	Description
handle	handle to the CDC device instance to which the request should be sent.
requestHandle	Pointer to USB_HOST_CDC_REQUEST_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.
controlLineState	Pointer to the control line state data structure.

Function

```
  USB_HOST_CDC_RESULT USB_HOST_CDC_ACM_ControlLineStateSet
(
    USB_HOST_CDC_HANDLE handle,
    USB_HOST_CDC_REQUEST_HANDLE * requestHandle,
USB_CDC_HOST_CONTROL_LINE_STATE * controlLineState
);
```

USB_HOST_CDC_ACM_LineCodingGet Function

This function sends a request to the attached device to get its Line Coding.

File

usb_host_cdc_acm.h

C

```
USB_HOST_CDC_RESULT USB_HOST_CDC_ACM_LineCodingGet(USB_HOST_CDC_HANDLE handle, USB_HOST_CDC_REQUEST_HANDLE
* requestHandle, USB_CDC_LINE_CODING * lineCoding);
```

Returns

USB_HOST_CDC_RESULT_SUCCESS - The operation was successful.

USB_HOST_CDC_RESULT_DEVICE_UNKNOWN - The device that this request was targeted to does not exist in the system.

USB_HOST_CDC_RESULT_BUSY - The request could not be scheduled at this time. The client should try again.

USB_HOST_CDC_RESULT_INVALID_PARAMETER - An input parameter was NULL.

USB_HOST_CDC_RESULT_FAILURE - An unknown failure occurred.

USB_HOST_CDC_RESULT_HANDLE_INVALID - The client handle is not valid.

Description

This function sends a request to the attached device to get its line coding. The function schedules a GET LINE CODING control transfer. If successful, the requestHandle parameter will contain a valid request handle, else it will contain USB_HOST_CDC_REQUEST_HANDLE_INVALID. When completed, the CDC client driver will generate a USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE event.

Remarks

None.

Preconditions

The client handle should be valid.

Example

Parameters

Parameters	Description
handle	handle to the CDC device instance to which the request should be sent.
requestHandle	Pointer to USB_HOST_CDC_REQUEST_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.

lineCoding	Pointer to the line coding data structure where the obtained line coding will be stored. The contents of this data structure will be valid only when the USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE event has been generated.

Function

```
USB_HOST_CDC_RESULT USB_HOST_CDC_ACM_LineCodingGet
(
    USB_HOST_CDC_HANDLE handle,
    USB_HOST_CDC_REQUEST_HANDLE * requestHandle,
USB_CDC_LINE_CODING * lineCoding
);
```

USB_HOST_CDC_ACM_LineCodingSet Function

This function sends a request to the attached device to set its Line Coding.

File

usb_host_cdc_acm.h

C

```
USB_HOST_CDC_RESULT USB_HOST_CDC_ACM_LineCodingSet(USB_HOST_CDC_HANDLE handle, USB_HOST_CDC_REQUEST_HANDLE
* requestHandle, USB_CDC_LINE_CODING * lineCoding);
```

Returns

USB_HOST_CDC_RESULT_SUCCESS - The operation was successful.

USB_HOST_CDC_RESULT_DEVICE_UNKNOWN - The device that this request was targeted to does not exist in the system.

USB_HOST_CDC_RESULT_BUSY - The request could not be scheduled at this time. The client should try again.

USB_HOST_CDC_RESULT_INVALID_PARAMETER - An input parameter was NULL.

USB_HOST_CDC_RESULT_FAILURE - An unknown failure occurred.

USB_HOST_CDC_RESULT_HANDLE_INVALID - The client handle is not valid.

Description

This function sends a request to the attached device to set its line coding. The function schedules a SET LINE CODING control transfer. If successful, the requestHandle parameter will contain a valid request handle, else it will contain USB_HOST_CDC_REQUEST_HANDLE_INVALID. When completed, the CDC client driver will generate a USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE event.

Remarks

None.

Preconditions

The client handle should be valid.

Example

Parameters

Parameters	Description
handle	handle to the CDC device instance to which the request should be sent.
requestHandle	Pointer to USB_HOST_CDC_REQUEST_HANDLE type of a variable. This will contain a valid transfer handle if the request was successful.
lineCoding	Pointer to the line coding data structure containing the line coding to be set. The contents of this data structure should not be changed until the USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE event has been generated.

Function

```
USB_HOST_CDC_RESULT USB_HOST_CDC_ACM_LineCodingSet
(
    USB_HOST_CDC_HANDLE handle,
```

```
    USB_HOST_CDC_REQUEST_HANDLE * requestHandle,
USB_CDC_LINE_CODING * lineCoding
);
```

d) Data Types and Constants

USB_HOST_CDC_EVENT Enumeration

Identifies the possible events that the CDC Class Driver can generate.

File

usb_host_cdc.h

C

```
typedef enum {
  USB_HOST_CDC_EVENT_READ_COMPLETE,
  USB_HOST_CDC_EVENT_WRITE_COMPLETE,
  USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE,
  USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE,
  USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE,
  USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE,
  USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED,
  USB_HOST_CDC_EVENT_DEVICE_DETACHED
} USB_HOST_CDC_EVENT;
```

Members

Members	Description
USB_HOST_CDC_EVENT_READ_COMPLETE	This event occurs when a CDC Client Driver Read operation has completed i.e when the data has been received from the connected CDC device. This event is generated after the application calls the USB_HOST_CDC_Read function. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_CDC_EVENT_READ_COMPLETE_DATA structure. This contains details about the transfer handle associated with this read request, the amount of data read and the termination status of the read request.
USB_HOST_CDC_EVENT_WRITE_COMPLETE	This event occurs when a CDC Client Driver Write operation has completed i.e when the data has been written to the connected CDC device. This event is generated after the application calls the USB_HOST_CDC_Write function. The eventData parameter in the event call back function will be a pointer to a USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA structure. This contains details about the transfer handle associated with this write request, the amount of data written and the termination status of the write request.
USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE	This event occurs when a CDC Client Driver Send Break request has completed. This event is generated after the application calls the USB_HOST_CDC_ACM_BreakSend function and the device has either acknowledged or stalled the request. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA structure. This contains details about the transfer handle associated with this request, the amount of data sent and the termination status of the set request.

USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE	This event occurs when a CDC Client Driver Set Control Line State request has completed. This event is generated after the application calls the USB_HOST_CDC_ACM_ControlLineStateSet function and the device has either acknowledged or stalled the request. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA structure. This contains details about the transfer handle associated with this request, the amount of data sent and the termination status of the set request.
USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE	This event occurs when a CDC Client Driver Set Line Coding request has completed. This event is generated after the application calls the USB_HOST_CDC_ACM_LineCodingSet function and the device either acknowledged or stalled the request. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA structure. This contains details about the transfer handle associated with this request, the amount of data sent and the termination status of the set request.
USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE	This event occurs when a CDC Client Driver Get Line Coding request has completed. This event is generated after the application calls the USB_HOST_CDC_ACM_LineCodingGet function and the device sends the line coding to the host. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA structure. This contains details about the transfer handle associated with this request, the amount of data received and the termination status of the get request.
USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED	This event occurs when a CDC Client Driver Serial State Notification Get operation has completed. This event is generated after the application calls the USB_HOST_CDC_SerialStateNotificationGet and the device sends a serial state notification to the host. The eventData parameter in the event call back function will be of a pointer to a USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED_DATA structure. This contains details about the transfer handle associated with this request, the amount of data received and the termination status of the get request.
USB_HOST_CDC_EVENT_DEVICE_DETACHED	This event occurs when the device that this client was connected to has • been detached. The client should close the CDC instance. There is no • event data associated with this event

Description

CDC Class Driver Events

This enumeration identifies the possible events that the CDC Class Driver can generate. The application should register an event handler using the USB_HOST_CDC_EventHandlerSet function to receive CDC Class Driver events.

An event may have data associated with it. Events that are generated due to a transfer of data between the host and device are accompanied by data structures that provide the status of the transfer termination. For example, the USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE event is accompanied by a pointer to a USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA data structure. The transferStatus member of this data structure indicates the success or failure of the transfer. A transfer may fail due to device not responding on the bus, if the device stalls any stages of the transfer or due to NAK timeouts. The event description provides details on the nature of the event and the data that is associated with the event.

Remarks

None.

USB_HOST_CDC_RESULT Enumeration

USB Host CDC Client Driver Result enumeration.

File

usb_host_cdc.h

C

```
typedef enum {
  USB_HOST_CDC_RESULT_FAILURE,
  USB_HOST_CDC_RESULT_BUSY,
  USB_HOST_CDC_RESULT_REQUEST_STALLED,
  USB_HOST_CDC_RESULT_INVALID_PARAMETER,
  USB_HOST_CDC_RESULT_DEVICE_UNKNOWN,
  USB_HOST_CDC_RESULT_ABORTED,
  USB_HOST_CDC_RESULT_HANDLE_INVALID,
  USB_HOST_CDC_RESULT_SUCCESS
} USB_HOST_CDC_RESULT;
```

Members

Members	Description
USB_HOST_CDC_RESULT_FAILURE	An unknown failure has occurred
USB_HOST_CDC_RESULT_BUSY	The transfer or request could not be scheduled because internal • queues are full. The request or transfer should be retried
USB_HOST_CDC_RESULT_REQUEST_STALLED	The request was stalled
USB_HOST_CDC_RESULT_INVALID_PARAMETER	A required parameter was invalid
USB_HOST_CDC_RESULT_DEVICE_UNKNOWN	The associated device does not exist in the system.
USB_HOST_CDC_RESULT_ABORTED	The transfer or requested was aborted
USB_HOST_CDC_RESULT_HANDLE_INVALID	The specified handle is not valid
USB_HOST_CDC_RESULT_SUCCESS	The operation was successful

Description

USB Host CDC Client Driver Result.

This enumeration lists the possible results the CDC client driver uses. Only some results are applicable to some functions and events. Refer to the event and function documentation for more details.

Remarks

None.

USB_HOST_CDC_TRANSFER_HANDLE Type

USB Host CDC Client Driver Transfer Handle

File

usb_host_cdc.h

C

```
typedef uintptr_t USB_HOST_CDC_TRANSFER_HANDLE;
```

Description

USB Host CDC Client Driver Transfer Handle

This is returned by the CDC Client driver data transfer routines and should be used by the application to track the transfer especially in cases where transfers are queued.

Remarks

None.

USB_HOST_CDC_TRANSFER_HANDLE_INVALID Macro

USB Host CDC Client Driver Invalid Transfer Handle Definition.

File

usb_host_cdc.h

C

```
#define USB_HOST_CDC_TRANSFER_HANDLE_INVALID ((USB_HOST_CDC_TRANSFER_HANDLE)(-1))
```

Description

USB Host CDC Client Driver Invalid Transfer Handle Definition

This definition defines a USB Host CDC Client Driver Invalid Transfer Handle. A Invalid Transfer Handle is returned by the CDC Client Driver data transfer routines when the request was not successful.

Remarks

None.

USB_HOST_CDC_ATTACH_EVENT_HANDLER Type

USB Host CDC Client Driver Attach Event Handler Function Pointer Type.

File

usb_host_cdc.h

C

```
typedef void (* USB_HOST_CDC_ATTACH_EVENT_HANDLER)(USB_HOST_CDC_OBJ cdcObjHandle, uintptr_t context);
```

Description

USB Host CDC Client Driver Attach Event Handler Function Pointer Type.

This data type defines the required function signature of the USB Host CDC Client Driver attach event handling callback function. The application must register a pointer to a CDC Client Driver attach events handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive attach event call backs from the CDC Client Driver. The client driver will invoke this function with event relevant parameters. The description of the event handler function parameters is given here.

cdcObjHandle - Handle of the client to which this event is directed.

context - Value identifying the context of the application that was registered along with the event handling function.

Remarks

None.

USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA Structure

USB Host CDC Client Driver Command Event Data.

File

usb_host_cdc.h

C

```
typedef struct {
  USB_HOST_CDC_REQUEST_HANDLE requestHandle;
  USB_HOST_CDC_RESULT result;
  size_t length;
} USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_CDC_REQUEST_HANDLE requestHandle;	Request handle of this request
USB_HOST_CDC_RESULT result;	Termination status
size_t length;	Size of the data transferred in the request

Description

USB Host CDC Client Driver Command Event Data.

This data type defines the data structure returned by the driver along with the following events:
USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE_DATA,

USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA,

Remarks

None.

USB_HOST_CDC_HANDLE Type

Defines the type of the CDC Host Client Driver Handle

File

usb_host_cdc.h

C

```
typedef uintptr_t USB_HOST_CDC_HANDLE;
```

Description

USB Host CDC Client Driver Handle

This type defines the type of the handle returned by USB_HOST_CDC_Open() function. This application uses this handle to specify the instance of the CDC client driver being accessed while calling a CDC Client driver function.

Remarks

None.

USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE_DATA Structure

USB Host CDC Client Driver Command Event Data.

File

usb_host_cdc.h

C

```
typedef struct {
  USB_HOST_CDC_REQUEST_HANDLE requestHandle;
  USB_HOST_CDC_RESULT result;
  size_t length;
} USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_CDC_REQUEST_HANDLE requestHandle;	Request handle of this request
USB_HOST_CDC_RESULT result;	Termination status
size_t length;	Size of the data transferred in the request

Description

USB Host CDC Client Driver Command Event Data.

This data type defines the data structure returned by the driver along with the following events:
USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA,

Remarks

None.

USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA Structure

USB Host CDC Client Driver Command Event Data.

File

usb_host_cdc.h

C

```
typedef struct {
  USB_HOST_CDC_REQUEST_HANDLE requestHandle;
  USB_HOST_CDC_RESULT result;
  size_t length;
} USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_CDC_REQUEST_HANDLE requestHandle;	Request handle of this request
USB_HOST_CDC_RESULT result;	Termination status
size_t length;	Size of the data transferred in the request

Description

USB Host CDC Client Driver Command Event Data.

This data type defines the data structure returned by the driver along with the following events:
USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA,

Remarks

None.

USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA Structure

USB Host CDC Client Driver Command Event Data.

File

usb_host_cdc.h

C

```
typedef struct {
  USB_HOST_CDC_REQUEST_HANDLE requestHandle;
  USB_HOST_CDC_RESULT result;
  size_t length;
} USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_CDC_REQUEST_HANDLE requestHandle;	Request handle of this request
USB_HOST_CDC_RESULT result;	Termination status
size_t length;	Size of the data transferred in the request

Description

USB Host CDC Client Driver Command Event Data.

This data type defines the data structure returned by the driver along with the following events:
USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA,
USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA,

Remarks

None.

USB_HOST_CDC_EVENT_HANDLER Type

USB Host CDC Client Driver Event Handler Function Pointer Type.

File

usb_host_cdc.h

C

```
typedef USB_HOST_CDC_EVENT_RESPONSE (* USB_HOST_CDC_EVENT_HANDLER)(USB_HOST_CDC_HANDLE cdcHandle,
USB_HOST_CDC_EVENT event, void * eventData, uintptr_t context);
```

Description

USB Host CDC Client Driver Event Handler Function Pointer Type.

This data type defines the required function signature of the USB Host CDC Client Driver event handling callback function. The application must register a pointer to a CDC Client Driver events handling function whose function signature (parameter and return value types) match the types specified by this function pointer in order to receive event call backs from the CDC Client Driver. The class driver will invoke this function with event relevant parameters. The description of the event handler function parameters is given here.

handle - Handle of the client to which this event is directed.

event - Type of event generated.

eventData - This parameter should be type casted to a event specific pointer type based on the event that has occurred. Refer to the USB_HOST_CDC_EVENT enumeration description for more details.

context - Value identifying the context of the application that was registered along with the event handling function.

Remarks

None.

USB_HOST_CDC_EVENT_READ_COMPLETE_DATA Structure

USB Host CDC Client Driver Event Data.

File

usb_host_cdc.h

C

```
typedef struct {
  USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
  USB_HOST_CDC_RESULT result;
  size_t length;
} USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED_DATA, USB_HOST_CDC_EVENT_READ_COMPLETE_DATA,
USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;	Transfer handle of this transfer
USB_HOST_CDC_RESULT result;	Termination transfer status
size_t length;	Size of the data transferred in the request

Description

USB Host CDC Client Driver Event Data.

This data type defines the data structure returned by the driver along with the following events: USB_HOST_CDC_EVENT_READ_COMPLETE_DATA, USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA,

Remarks

None.

USB_HOST_CDC_EVENT_RESPONSE Enumeration

Return type of the USB CDC Host Client Driver Event Handler.

File

usb_host_cdc.h

C

```
typedef enum {
  USB_HOST_CDC_EVENT_RESPONE_NONE
} USB_HOST_CDC_EVENT_RESPONSE;
```

Members

Members	Description
USB_HOST_CDC_EVENT_RESPONE_NONE	This means no response is required

Description

USB Host CDC Event Handler Return Type

This enumeration list the possible return values of the USB CDC Host Client Driver Event Handler.

Remarks

None.

USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED_DATA Structure

USB Host CDC Client Driver Event Data.

File

usb_host_cdc.h

C

```
typedef struct {
  USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
  USB_HOST_CDC_RESULT result;
  size_t length;
} USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED_DATA, USB_HOST_CDC_EVENT_READ_COMPLETE_DATA,
USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;	Transfer handle of this transfer
USB_HOST_CDC_RESULT result;	Termination transfer status
size_t length;	Size of the data transferred in the request

Description

USB Host CDC Client Driver Event Data.

This data type defines the data structure returned by the driver along with the following events:
USB_HOST_CDC_EVENT_READ_COMPLETE_DATA, USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA,

Remarks

None.

USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA Structure

USB Host CDC Client Driver Event Data.

File

usb_host_cdc.h

C

```
typedef struct {
  USB_HOST_CDC_TRANSFER_HANDLE transferHandle;
  USB_HOST_CDC_RESULT result;
  size_t length;
} USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED_DATA, USB_HOST_CDC_EVENT_READ_COMPLETE_DATA,
USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA;
```

Members

Members	Description
USB_HOST_CDC_TRANSFER_HANDLE transferHandle;	Transfer handle of this transfer
USB_HOST_CDC_RESULT result;	Termination transfer status
size_t length;	Size of the data transferred in the request

Description

USB Host CDC Client Driver Event Data.

This data type defines the data structure returned by the driver along with the following events: USB_HOST_CDC_EVENT_READ_COMPLETE_DATA, USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA,

Remarks

None.

USB_HOST_CDC_OBJ Type

Defines the type of the CDC Host Client Object.

File

usb_host_cdc.h

C

```
typedef uintptr_t USB_HOST_CDC_OBJ;
```

Description

USB Host CDC Object

This type defines the type of the CDC Host Client Object. This type is returned by the Attach Event Handler and is used by the application to open the attached CDC Device.

Remarks

None.

USB_HOST_CDC_REQUEST_HANDLE Type

USB Host CDC Client Driver Request Handle

File

usb_host_cdc.h

C

```
typedef uintptr_t USB_HOST_CDC_REQUEST_HANDLE;
```

Description

USB Host CDC Client Driver Request Handle

This is returned by the CDC Client driver command routines and should be used by the application to track the command especially in cases where transfers are queued.

Remarks

None.

USB_HOST_CDC_INTERFACE Macro

USB HOST CDC Client Driver Interface

File

usb_host_cdc.h

C

```
#define USB_HOST_CDC_INTERFACE
```

Description

USB HOST CDC Client Driver Interface

This macro should be used by the application in TPL table while adding support for the USB CDC Host Client Driver.

Remarks

None.

USB_HOST_CDC_REQUEST_HANDLE_INVALID Macro

USB Host CDC Client Driver Invalid Request Handle

File

usb_host_cdc.h

C

```
#define USB_HOST_CDC_REQUEST_HANDLE_INVALID ((USB_HOST_CDC_REQUEST_HANDLE)(-1))
```

Description

USB Host CDC Client Driver Invalid Request Handle

This is returned by the CDC Client driver command routines when the request could not be scheduled.

Remarks

None.

USB_HOST_CDC_HANDLE_INVALID Macro

Defines an Invalid CDC Client Driver Handle.

File

usb_host_cdc.h

C

```
#define USB_HOST_CDC_HANDLE_INVALID ((USB_HOST_CDC_HANDLE)(-1))
```

Description

USB Host CDC Client Driver Invalid Handle

This type defines an Invalid CDC Client Driver Handle. The USB_HOST_CDC_Open() function returns an invalid handle when it fails to open the specified CDC device instance.

Remarks

None.

Files

Files

Name	Description
usb_host_cdc.h	USB Host CDC Client Driver Interface Header
usb_host_cdc_acm.h	USB Host CDC Client Driver Interface Header
usb_host_cdc_config_template.h	USB host CDC Class configuration definitions template

Description

usb_host_cdc.h

USB Host CDC Client Driver Interface Header

Enumerations

	Name	Description
	USB_HOST_CDC_EVENT	Identifies the possible events that the CDC Class Driver can generate.
	USB_HOST_CDC_EVENT_RESPONSE	Return type of the USB CDC Host Client Driver Event Handler.
	USB_HOST_CDC_RESULT	USB Host CDC Client Driver Result enumeration.

Functions

	Name	Description
	USB_HOST_CDC_AttachEventHandlerSet	This function will set an attach event handler.
	USB_HOST_CDC_Close	This function closes the CDC device.
	USB_HOST_CDC_DeviceObjHandleGet	This function returns the Device Object Handle for this CDC device.
	USB_HOST_CDC_EventHandlerSet	Registers an event handler with the CDC Host Client Driver.
	USB_HOST_CDC_Open	This function opens the specified CDC device.
	USB_HOST_CDC_Read	This function will read data from the attached device.
	USB_HOST_CDC_SerialStateNotificationGet	This function will request Serial State Notification from the attached device.
	USB_HOST_CDC_Write	This function will write data to the attached device.

Macros

	Name	Description
	USB_HOST_CDC_HANDLE_INVALID	Defines an Invalid CDC Client Driver Handle.
	USB_HOST_CDC_INTERFACE	USB HOST CDC Client Driver Interface
	USB_HOST_CDC_REQUEST_HANDLE_INVALID	USB Host CDC Client Driver Invalid Request Handle
	USB_HOST_CDC_TRANSFER_HANDLE_INVALID	USB Host CDC Client Driver Invalid Transfer Handle Definition.

Structures

	Name	Description
	USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA	USB Host CDC Client Driver Command Event Data.
	USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE_DATA	USB Host CDC Client Driver Command Event Data.
	USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA	USB Host CDC Client Driver Command Event Data.
	USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA	USB Host CDC Client Driver Command Event Data.
	USB_HOST_CDC_EVENT_READ_COMPLETE_DATA	USB Host CDC Client Driver Event Data.
	USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED_DATA	USB Host CDC Client Driver Event Data.
	USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA	USB Host CDC Client Driver Event Data.

Types

	Name	Description
	USB_HOST_CDC_ATTACH_EVENT_HANDLER	USB Host CDC Client Driver Attach Event Handler Function Pointer Type.
	USB_HOST_CDC_EVENT_HANDLER	USB Host CDC Client Driver Event Handler Function Pointer Type.
	USB_HOST_CDC_HANDLE	Defines the type of the CDC Host Client Driver Handle
	USB_HOST_CDC_OBJ	Defines the type of the CDC Host Client Object.
	USB_HOST_CDC_REQUEST_HANDLE	USB Host CDC Client Driver Request Handle
	USB_HOST_CDC_TRANSFER_HANDLE	USB Host CDC Client Driver Transfer Handle

Description

USB Host CDC Client Driver Interface Definition

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB Host CDC

Client Driver.

File Name

usb_host_cdc.h

Company

Microchip Technology Inc.

usb_host_cdc_acm.h

USB Host CDC Client Driver Interface Header

Functions

	Name	Description
	USB_HOST_CDC_ACM_BreakSend	This function sends a request to the attached device to update its break duration.
	USB_HOST_CDC_ACM_ControlLineStateSet	This function sends a request to the attached device to set its Control Line State.
	USB_HOST_CDC_ACM_LineCodingGet	This function sends a request to the attached device to get its Line Coding.
	USB_HOST_CDC_ACM_LineCodingSet	This function sends a request to the attached device to set its Line Coding.

Description

USB Host CDC Client Driver Interface Definition

This header file contains the CDC ACM specific function prototypes and definitions of the data types and constants that make up the interface to the USB Host CDC Client Driver.

File Name

usb_host_cdc_acm.h

Company

Microchip Technology Inc.

usb_host_cdc_config_template.h

USB host CDC Class configuration definitions template

Macros

	Name	Description
	USB_HOST_CDC_ATTACH_LISTENERS_NUMBER	Defines the number of attach event listeners that can be registered with CDC Host Client Driver.
	USB_HOST_CDC_INSTANCES_NUMBER	Specifies the number of CDC instances.

Description

USB Host CDC Class Configuration Definitions

This file contains configurations macros needed to configure the CDC host Driver. This file is a template file only. It should not be included by the application. The configuration macros defined in the file should be defined in the configuration specific system_config.h.

File Name

usb_host_cdc_config_template.h

Company

Microchip Technology Inc.

USB HID Host Mouse Driver Library

This section describes the USB HID Host Mouse Driver Library.

Introduction

Introduces the MPLAB Harmony USB HID Host Mouse Driver Library.

Description

The HID Host Mouse Driver in the MPLAB Harmony USB Host Stack allows USB Host Applications to support and interact with USB Mouse devices. The USB HID Host Mouse Driver has the following features:

- Supports USB HID Mouse devices
- Supports HID device matching at both device descriptor and interface descriptor level
- Supports both Boot and Non Boot interface USB Mouse devices
- Performs parsing of Mouse Report descriptor by using USB HID Host client driver APIs
- Supports detection of Mouse X, Y, and Z movements, as well as button click events

Using the Library

This topic describes the basic architecture of the USB HID Host Mouse Driver Library and provides information and examples on its use.

Library Overview

The USB HID Host Mouse Driver can be grouped functionally as shown in the following table.

Library Interface Section	Description
Mouse Access Functions	These functions allow application to register event handlers with the mouse driver. These functions are implemented in `usb_host_hid_mouse.c`.

Abstraction Model

Describes the Abstraction Model of the USB HID Host Mouse Driver Library.

Description

The USB HID Host Mouse Driver interacts with the USB Host HID Client Driver to control the attached HID device. The USB Host Layer attaches the USB HID Host Client Driver to the HID device when it meets the matching criteria specified in the USB Host TPL table.

The USB HID Host Client driver notifies the mouse driver of device attach and detach information and with report receive events with relevant event data. On a report receive event, the USB Host HID Mouse Driver obtains all of the field information present in the Report Descriptor of the mouse device and uses that field information and the INTERRUPT IN data received to understand mouse parameter values.

How the Library Works

Describes how the library works and how it should be used.

Description

The USB HID Host Mouse Driver provides the user application with an easy to use interface to the attached HID device. The USB Host Layer initializes the USB HID Host Client Driver when a device is attached. This process does not require application intervention. The following sections describe the steps and methods required for the user application to interact with the attached mouse devices through the USB Host HID Mouse Driver.

HID Device TPL Table Configuration

Provides information on configuring the TPL table for HID devices.

Description

The Host Layer attaches the USB HID Host Client Driver to a device when the device class, subclass, protocol in the device descriptor or when the class, subclass and protocol fields in the Interface descriptor matches the entry in the TPL table. When specifying the entry for the HID device along with the Usage driver, the driver interface must be set to USB_HOST_HID_INTERFACE and the usage driver interface must be set to usageDriverInterface. usageDriverInterface must be properly initialized to capture the Mouse driver APIs. This will attach the USB HID Host Mouse Driver to the device when the USB Host HID Client Driver is attached. The following code shows possible TPL table options for matching HID Devices.

Example:

```
/* This code shows an example of TPL table entries for supporting HID mouse devices.
 * Note that the driver interface is set to USB_HOST_HID_INTERFACE. This
 * will load the HID Host Client Driver when there is TPL match. Usage driver
 * interface is initialized with appropriate function pointer for Mouse driver.
 * This facilitates subsequent loading of Mouse driver post HID client driver.
 */
```

```
 USB_HOST_HID_USAGE_DRIVER_INTERFACE usageDriverInterface =
{
  .initialize = NULL,
  .deinitialize = NULL,
  .usageDriverEventHandler = _USB_HOST_HID_MOUSE_EventHandler,
  .usageDriverTask = _USB_HOST_HID_MOUSE_Task
};

USB_HOST_HID_USAGE_DRIVER_TABLE_ENTRY usageDriverTableEntry[1] =
{
    {
        .usage = USB_HID_USAGE_MOUSE,
        .initializeData = NULL,
        .interface = &usageDriverInterface
    }
};
const USB_HOST_TPL_ENTRY USBTPList[1] =
{
    /* This entry looks for any HID Mouse device */
    TPL_INTERFACE_CLASS_SUBCLASS_PROTOCOL(0x03, 0x01, 0x02, usageDriverTableEntry,
                                          USB_HOST_HID_INTERFACE) ,
};
```

Detecting Device Attach

Describe how to detect when a HID mouse device is attached.

Description

The application will need to know when a HID mouse device is attached. To receive this attach event from the USB HID Host Mouse Driver, the application must register an Attach Event Handler by calling the USB_HOST_HID_MOUSE_EventHandlerSet function. This function should be called before calling the USB_HOST_BusEnable function; otherwise, the application may miss HID attach events.

Mouse Data Event Handling

Describes mouse data event handling, which includes a code example.

Description

No extra event handler is required to be registered to receive mouse data. A call to function USB_HOST_HID_MOUSE_EventHandlerSet once is adequate to receive mouse data as well.

The mouse button state along with the X, Y, and Z relative coordinate positions are provided by the USB Host HID Mouse Driver. The data type is USB_HOST_HID_MOUSE_DATA and is defined in `usb_host_hid_mouse.h`. The following code shows an event handler example.

Example:

```
/* This code shows an example of HID Mouse Event Handler */

void APP_USBHostHIDMouseEventHandler
(
    USB_HOST_HID_MOUSE_HANDLE handle,
    USB_HOST_HID_MOUSE_EVENT event,
    void * pData
)
{
    /* This function gets called in the following scenarios:
    1. USB Mouse is Attached
    2. USB Mouse is detached
    3. USB Mouse data has been obtained.
    */

    switch ( event)
    {
        case USB_HOST_HID_MOUSE_EVENT_ATTACH:
            /* Mouse Attached */
            appData.state =  APP_STATE_DEVICE_ATTACHED;
            break;

        case USB_HOST_HID_MOUSE_EVENT_DETACH:
            /* Mouse Detached */
```

```
                appData.state = APP_STATE_DEVICE_DETACHED;
                break;

            case USB_HOST_HID_MOUSE_EVENT_REPORT_RECEIVED:
                /* Mouse data event */
                appData.state = APP_STATE_READ_HID;
                /* Mouse Data from device */
                memcpy(&appData.data, pData, sizeof(appData.data));

                /* Now the Mouse data has been obtained. This is a parsed data
                in a simple format defined by USB_HOST_HID_MOUSE_DATA type.
                */
                break;
        }

    }

    void APP_Tasks(void)
    {
        switch (appData.state)
        {
            case APP_STATE_BUS_ENABLE:

                /* In this state the application enables the USB Host Bus. Note
                 * how the USB Mouse event handler is registered before the bus
                 * is enabled. */

                USB_HOST_HID_MOUSE_EventHandlerSet(APP_USBHostHIDMouseEventHandler);
                USB_HOST_BusEnable(0);
                appData.state = APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE;
                break;

            case APP_STATE_WAIT_FOR_BUS_ENABLE_COMPLETE:
                /* Here we wait for the bus enable operation to complete. */
                break;
        }
    }
```

Configuring the Library

Describes how to configure the USB HID Host Mouse Driver.

Macros

	Name	Description
	USB_HID_GLOBAL_PUSH_POP_STACK_SIZE	Specifies the Global PUSH POP stack size supported.
	USB_HOST_HID_INSTANCES_NUMBER	Specifies the number of HID instances.
	USB_HOST_HID_INTERRUPT_IN_ENDPOINTS_NUMBER	Specifies the maximum number of INTERRUPT IN endpoints supported.
	USB_HOST_HID_MOUSE_BUTTONS_NUMBER	Specifies the number of Mouse buttons supported.
	USB_HOST_HID_USAGE_DRIVER_SUPPORT_NUMBER	Specifies the number of Usage driver registered

Description

The USB HID Host Mouse Driver requires configuration constants to be specified in the `system_config.h` file. These constants define the build time configuration (functionality and static resources) of the USB HID Host Mouse Driver.

USB_HID_GLOBAL_PUSH_POP_STACK_SIZE Macro

Specifies the Global PUSH POP stack size supported.

File

usb_host_hid_config_template.h

C

```
#define USB_HID_GLOBAL_PUSH_POP_STACK_SIZE
```

Description

USB Host HID Global PUSH POP stack size supported

This macro defines the size of the Global PUSH POP stack per HID driver instance. If the application wants to support HID device having 2 continuous PUSH item or 2 continuous POP item in the Report Descriptor, then the value should be set to 2.

Remarks

None.

USB_HOST_HID_INSTANCES_NUMBER Macro

Specifies the number of HID instances.

File

usb_host_hid_config_template.h

C

```
#define USB_HOST_HID_INSTANCES_NUMBER
```

Description

USB host HID Maximum Number of instances

This macro defines the number of instances of the HID host Driver. For example, if the application needs to implement two instances of the HID host Driver, value should be set to 2.

Remarks

None.

USB_HOST_HID_INTERRUPT_IN_ENDPOINTS_NUMBER Macro

Specifies the maximum number of INTERRUPT IN endpoints supported.

File

usb_host_hid_config_template.h

C

```
#define USB_HOST_HID_INTERRUPT_IN_ENDPOINTS_NUMBER
```

Description

USB Host HID Maximum Number of INTERRUPT IN endpoints supported

This macro defines the number of INTERRUPT IN endpoints supported by USB Host HID driver. If the application needs to work with HID device which has 2 INTERRUPT IN endpoints, the value should be set to 2.

Remarks

None.

USB_HOST_HID_MOUSE_BUTTONS_NUMBER Macro

Specifies the number of Mouse buttons supported.

File

usb_host_hid_config_template.h

C

```
#define USB_HOST_HID_MOUSE_BUTTONS_NUMBER
```

Description

USB Host HID number of Mouse buttons supported

This macro defines the number of Mouse buttons supported. If the application wants to support HID Mouse device having 5 buttons, then the value should be set to 5.

Remarks

None.

USB_HOST_HID_USAGE_DRIVER_SUPPORT_NUMBER Macro

Specifies the number of Usage driver registered

File

usb_host_hid_config_template.h

C

```
#define USB_HOST_HID_USAGE_DRIVER_SUPPORT_NUMBER
```

Description

USB Host HID number of Usage driver registered

This macro defines the number of Usage driver registered with USB Host Hid driver. If the application wants HID driver to support 2 HID device having different usage, then the value should be set to 2.

Remarks

None.

Building the Library

Describes the files to be included in the project while using the USB HID Host Mouse Driver.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/usb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
usb_host_hid_mouse.h	This header file should be included in any `.c` file that accesses the YSB HID Host Mouse Driver API.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/usb_host_hid.c`	This file implements the USB HID Host Client Driver interface and should be included in the project if any usage driver operation is desired.
`/src/dynamic/usb_host_hid_mouse.c`	This file implements the USB HID Host Mouse Driver interface and should be included in the project if any usage driver operation is desired.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The USB HID Host Mouse Driver Library depends on the following modules:

- USB Host Layer Library
- USB Host HID Client Driver Library

Library Interface

a) Mouse Access Functions

	Name	Description
	USB_HOST_HID_MOUSE_EventHandlerSet	This function registers application callback function with the mouse driver.
	_USB_HOST_HID_MOUSE_EventHandler	This is function _USB_HOST_HID_MOUSE_EventHandler.
	_USB_HOST_HID_MOUSE_Task	This is function _USB_HOST_HID_MOUSE_Task.

b) Data Types and Constants

	Name	Description
	USB_HOST_HID_MOUSE_DATA	Defines the USB Host HID mouse data object.
	USB_HOST_HID_MOUSE_EVENT	Defines the possible USB HOST HID mouse driver events.
	USB_HOST_HID_MOUSE_EVENT_HANDLER	USB HOST mouse driver event handler function pointer type.
	USB_HOST_HID_MOUSE_HANDLE	USB HOST HID mouse driver instance handle.
	USB_HOST_HID_MOUSE_RESULT	USB Host HID mouse driver results.
	USB_HOST_HID_MOUSE_RESULT_MIN	USB Host HID mouse driver result minimum constant.
	USB_HOST_HID_MOUSE_HANDLE_INVALID	This is macro USB_HOST_HID_MOUSE_HANDLE_INVALID.

Description

This section describes the Application Programming Interface (API) functions of the USB HID Host Mouse Driver Library.

The USB Mouse driver does not require explicit API call by the application to obtain Mouse data. The data in the appropriate format is sent to the application during an application event handler function call.

a) Mouse Access Functions

USB_HOST_HID_MOUSE_EventHandlerSet Function

This function registers application callback function with the mouse driver.

File

usb_host_hid_mouse.h

C

```
USB_HOST_HID_MOUSE_RESULT USB_HOST_HID_MOUSE_EventHandlerSet(USB_HOST_HID_MOUSE_EVENT_HANDLER
appMouseEventHandler);
```

Returns

Returns data structure of USB_HOST_HID_MOUSE_RESULT type. USB_HOST_HID_MOUSE_RESULT_INVALID_PARAMETER: Invalid Parameter USB_HOST_HID_MOUSE_RESULT_FAILURE: On failure USB_HOST_HID_MOUSE_RESULT_SUCCESS: On success

Description

This function registers application callback function with the mouse driver. Any subsequent mouse events is passed to the application by calling the registered application function. The function prototype should be of the USB_HOST_HID_MOUSE_EVENT_HANDLER type.

Remarks

This function should be called before the USB bus is enabled.

Preconditions

This function should be called before the USB bus is enabled.

Parameters

Parameters	Description
appMouseEventHandler	Function pointer to the application function.

Function

USB_HOST_HID_MOUSE_RESULT USB_HOST_HID_MOUSE_EventHandlerSet

(

```
    USB_HOST_HID_MOUSE_EVENT_HANDLER appMouseEventHandler
);
```

_USB_HOST_HID_MOUSE_EventHandler Function

File

usb_host_hid_mouse.h

C

```
void _USB_HOST_HID_MOUSE_EventHandler(USB_HOST_HID_OBJ_HANDLE handle, USB_HOST_HID_EVENT event, void *
eventData);
```

Description

This is function _USB_HOST_HID_MOUSE_EventHandler.

_USB_HOST_HID_MOUSE_Task Function

File

usb_host_hid_mouse.h

C

```
void _USB_HOST_HID_MOUSE_Task(USB_HOST_HID_OBJ_HANDLE handle);
```

Description

This is function _USB_HOST_HID_MOUSE_Task.

b) Data Types and Constants

USB_HOST_HID_MOUSE_DATA Structure

Defines the USB Host HID mouse data object.

File

usb_host_hid_mouse.h

C

```
typedef struct {
  USB_HID_BUTTON_STATE buttonState[USB_HOST_HID_MOUSE_BUTTONS_NUMBER];
  USB_HID_BUTTON_ID buttonID[USB_HOST_HID_MOUSE_BUTTONS_NUMBER];
  int16_t xMovement;
  int16_t yMovement;
  int16_t zMovement;
} USB_HOST_HID_MOUSE_DATA;
```

Members

Members	Description
USB_HID_BUTTON_STATE buttonState[USB_HOST_HID_MOUSE_BUTTONS_NUMBER];	Button state for the buttons. USB_HOST_HID_MOUSE_BUTTONS_NUMBER is system configurable option. The actual number of buttons in the mouse needs to be <= USB_HOST_HID_MOUSE_BUTTONS_NUMBER
int16_t xMovement;	Applicable for 2D Mouse Y - Coordinate displacement
int16_t yMovement;	Applicable for 2D Mouse Z - Coordinate displacement
int16_t zMovement;	Applicable only for 3D Mouse

Description

USB Host HID Mouse Data Object

This structure defines the USB Host HID mouse data object.

Remarks

None.

USB_HOST_HID_MOUSE_EVENT Enumeration

Defines the possible USB HOST HID mouse driver events.

File

usb_host_hid_mouse.h

C

```
typedef enum {
  USB_HOST_HID_MOUSE_EVENT_ATTACH = 0,
  USB_HOST_HID_MOUSE_EVENT_DETACH,
  USB_HOST_HID_MOUSE_EVENT_REPORT_RECEIVED
} USB_HOST_HID_MOUSE_EVENT;
```

Members

Members	Description
USB_HOST_HID_MOUSE_EVENT_ATTACH = 0	Mouse has been attached
USB_HOST_HID_MOUSE_EVENT_DETACH	Mouse has been detached
USB_HOST_HID_MOUSE_EVENT_REPORT_RECEIVED	Mouse IN Report data available

Description

USB HOST HID Mouse Driver Events

This enumeration lists the possible mouse events that the mouse driver can provide to the application. Some of these events have event data associated with them.

USB_HOST_HID_MOUSE_EVENT_HANDLER Type

USB HOST mouse driver event handler function pointer type.

File

usb_host_hid_mouse.h

C

```
typedef void (* USB_HOST_HID_MOUSE_EVENT_HANDLER)(USB_HOST_HID_MOUSE_HANDLE handle,
USB_HOST_HID_MOUSE_EVENT event, void *pData);
```

Description

USB HOST Mouse Driver Event Handler Function Pointer Type.

This defines the USB HOST HID mouse driver event handler function pointer type. Application must register a function of this type to receive HID mouse events. Registration should happen before USB BUS is enabled by the application.

USB_HOST_HID_MOUSE_HANDLE Type

USB HOST HID mouse driver instance handle.

File

usb_host_hid_mouse.h

C

```
typedef uintptr_t USB_HOST_HID_MOUSE_HANDLE;
```

Description

USB HOST HID Mouse Driver Instance Handle

This defines a USB Host HID mouse driver handle.

Remarks

None.

USB_HOST_HID_MOUSE_RESULT Enumeration

USB Host HID mouse driver results.

File

usb_host_hid_mouse.h

C

```
typedef enum {
  USB_HOST_HID_MOUSE_RESULT_FAILURE = USB_HOST_HID_MOUSE_RESULT_MIN,
  USB_HOST_HID_MOUSE_RESULT_INVALID_PARAMETER,
  USB_HOST_HID_MOUSE_RESULT_SUCCESS = 0
} USB_HOST_HID_MOUSE_RESULT;
```

Members

Members	Description
USB_HOST_HID_MOUSE_RESULT_FAILURE = USB_HOST_HID_MOUSE_RESULT_MIN	An unknown failure occurred
USB_HOST_HID_MOUSE_RESULT_INVALID_PARAMETER	Invalid or NULL parameter passed
USB_HOST_HID_MOUSE_RESULT_SUCCESS = 0	Indicates that the operation succeeded or the request was accepted and will be processed.

Description

USB Host HID MOUSE Result

This enumeration defines the possible returns values of USB Host HID mouse driver API. A function may only return some of the values in this enumeration. Refer to function description for details on which values will be returned.

Remarks

None.

USB_HOST_HID_MOUSE_RESULT_MIN Macro

USB Host HID mouse driver result minimum constant.

File

usb_host_hid_mouse.h

C

```
#define USB_HOST_HID_MOUSE_RESULT_MIN -50
```

Description

USB Host HID Mouse Driver Result Minimum Constant

This constant identifies the minimum value of the USB Host HID mouse driver and is used in the USB_HOST_HID_MOUSE_RESULT enumeration.

Remarks

None.

USB_HOST_HID_MOUSE_HANDLE_INVALID Macro

File

usb_host_hid_mouse.h

C

```
#define USB_HOST_HID_MOUSE_HANDLE_INVALID ((USB_HOST_HID_MOUSE_HANDLE)(-1))
```

Description

This is macro USB_HOST_HID_MOUSE_HANDLE_INVALID.

Files

Files

Name	Description
usb_host_hid_mouse.h	USB Host HID Mouse Driver Definition Header

usb_host_hid_config_template.h	USB host HID Class configuration definitions template

Description

This section lists the source and header files used by the library.

usb_host_hid_mouse.h

USB Host HID Mouse Driver Definition Header

Enumerations

	Name	Description
	USB_HOST_HID_MOUSE_EVENT	Defines the possible USB HOST HID mouse driver events.
	USB_HOST_HID_MOUSE_RESULT	USB Host HID mouse driver results.

Functions

	Name	Description
	_USB_HOST_HID_MOUSE_EventHandler	This is function _USB_HOST_HID_MOUSE_EventHandler.
	_USB_HOST_HID_MOUSE_Task	This is function _USB_HOST_HID_MOUSE_Task.
	USB_HOST_HID_MOUSE_EventHandlerSet	This function registers application callback function with the mouse driver.

Macros

	Name	Description
	USB_HOST_HID_MOUSE_HANDLE_INVALID	This is macro USB_HOST_HID_MOUSE_HANDLE_INVALID.
	USB_HOST_HID_MOUSE_RESULT_MIN	USB Host HID mouse driver result minimum constant.

Structures

	Name	Description
	USB_HOST_HID_MOUSE_DATA	Defines the USB Host HID mouse data object.

Types

	Name	Description
	USB_HOST_HID_MOUSE_EVENT_HANDLER	USB HOST mouse driver event handler function pointer type.
	USB_HOST_HID_MOUSE_HANDLE	USB HOST HID mouse driver instance handle.

Description

USB HOST HID Mouse Driver Interface Definition

This header file contains the function prototypes and definitions of the data types and constants that make up the interface between HID Mouse driver and top level application.

File Name

usb_host_hid_mouse.h

Company

Microchip Technology Inc.

usb_host_hid_config_template.h

USB host HID Class configuration definitions template

Macros

	Name	Description
	USB_HID_GLOBAL_PUSH_POP_STACK_SIZE	Specifies the Global PUSH POP stack size supported.
	USB_HOST_HID_INSTANCES_NUMBER	Specifies the number of HID instances.
	USB_HOST_HID_INTERRUPT_IN_ENDPOINTS_NUMBER	Specifies the maximum number of INTERRUPT IN endpoints supported.
	USB_HOST_HID_MOUSE_BUTTONS_NUMBER	Specifies the number of Mouse buttons supported.
	USB_HOST_HID_USAGE_DRIVER_SUPPORT_NUMBER	Specifies the number of Usage driver registered

Description

USB Host HID Class Configuration Definitions

This file contains configurations macros needed to configure the HID host Driver. This file is a template file only. It should not be included by the application. The configuration macros defined in the file should be defined in the configuration specific system_config.h.

File Name

usb_host_hid_config_template.h

Company

Microchip Technology Inc.

USB Hub Host Client Driver Library

This section describes the USB Hub Host Client Driver Library.

Introduction

Introduces the MPLAB Harmony USB Hub Host Client Driver Library.

Description

The USB Hub Host Client Driver in the MPLAB Harmony USB Host Stack allows USB Host Applications to interact with a USB Hub and thus manage multiple USB devices simultaneously in one application. The key features of the Hub Host Client Driver include:

- Allows multiple USB devices to be connected to the host and hence allow the USB Host application to interact simultaneously with multiple USB devices.
- Implemented as per Chapter 11 of the USB 2.0 specification.
- Support multiple Hub tiers. A Hub can be connected to another Hub.
- Does not require application intervention for its operation. The application does not have to call an Hub Driver API.

Abstraction Model

Describes the Abstraction Model of the USB Hub Host Client Driver Library.

Description

The USB Hub Host Client Driver abstracts the complexities of Hub operation and presents a simple interface to the Host Layer. The interface allows the Host Layer to perform port operations such as port reset, port suspend and port resume. The port interface offered by the Hub Host Client Driver is the same as that offered by the root hub driver. In that, the Host Layer does not differentiate between an external hub and the root hub.

The USB Hub Host Client Driver does not have any application callable API. It only interacts with the Host Layer. The USB Hub Host Client Driver performs the task of powering up the ports, detecting device attach and detach and notifying the same to the Host Layer and detecting over current conditions. The USB Hub Host Client Driver performs the control transfers required for these tasks.

Library Overview

The USB Hub Host Client Driver does not contain any application callable functions.

Using the Library

This topic describes the basic architecture of the USB Hub Host Client Driver Library and provides information and examples on its use.

How the Library Works

Describes how the Library works and how it should be used.

Description

The USB Hub Host Client Driver does not contain any application callable functions. The only step that the application code must implement is to enable USB Host Layer Hub support and to provision the USB Hub Host Client Driver in the TPL table.

The USB Host Layer enables Hub Support when the USB_HOST_HUB_SUPPORT_ENABLE configuration macro is defined in `system_config.h`. Refer to the Configuring the Library section of the USB Host Layer Library Help Topic for more information.

Hub TPL Table Configuration

Provides information on configuring the TPL table for adding Hub support.

Description

The Host Layer attaches the USB Hub Host Client Driver to a Hub device only if the TPL table contains an entry to enable this feature. The driver interface for such a TPL entry should point to USB_HOST_HUB_INTERFACE. The following code shows an example of the TPL entry for the adding Hub support to the application.

Example:

```
/* This code shows an example of how to initialize the TPL table to
 * support a USB Hub Host Client Driver */

#include "usb/usb_host_hub.h"
#include "usb/usb_hub.h"

const USB_HOST_TPL_ENTRY USBTPList[ 2 ] =
{
    TPL_INTERFACE_CLASS_SUBCLASS_PROTOCOL(USB_HUB_CLASS_CODE, 0x00, 0x00, NULL,  USB_HOST_HUB_INTERFACE),

    /* A high-speed hub will report the number of transaction translators in the
     * protocol field. We can ignore this and let the host layer load the hub
     * driver for a high-speed hub */

    TPL_INTERFACE_CLASS (USB_HUB_CLASS_CODE, NULL, USB_HOST_HUB_INTERFACE)
};
```

USB Hub Host Client Driver Test Results

Provides test results for the USB Hub Host Client Driver.

Description

The following table lists the commercially available USB hubs, which have been tested to successfully enumerate and operate with the USB Hub Host Client Driver in the MPLAB Harmony USB Host Stack. Note that if the Hub you are using is not included in the table, this indicates that this Hub has not been tested with the USB Hub Host Client Driver. However, the Hub could still potentially work with the USB Hub Host Client Driver.

The hubs were tested with the hub_msd USB Host demonstration in the latest version of the MPLAB Harmony USB Host Stack.

Hub Model	Number of Ports	VID	PID
Belkin USB 2.0	4	0x050D	0x0233
QHMPL	4	0x1A40	0x0101
Portronics	3	0x1A40	0x0101
Sanda	4	0x05E3	0x0606
iBall	4	0x1A40	0x0101

Configuring the Library

Describes how to configure the USB Hub Host Client Driver.

Macros

	Name	Description
	USB_HOST_HUB_INSTANCES_NUMBER	Specifies the number of Hub to be supported in the system.
	USB_HOST_HUB_PORTS_NUMBER	Specifies the number of ports per Hub.

Description

The USB Hub Host Client Driver requires configuration constants to be specified in `system_config.h` file. These constants define the build time configuration (functionality and static resources) of the Hub Host Client Driver.

USB_HOST_HUB_INSTANCES_NUMBER Macro

Specifies the number of Hub to be supported in the system.

File

usb_host_hub_config_template.h

C

```
#define USB_HOST_HUB_INSTANCES_NUMBER
```

Description

USB Host Hub Instances Number

This configuration constant defines the total number of hubs to be supported in the application. This includes hubs connected across multiple USBs. If the hub connected to the host exceed this number, then the additional hubs will not be enumerated.

Remarks

Increasing number of Hubs to be supported will also increase memory consumption.

USB_HOST_HUB_PORTS_NUMBER Macro

Specifies the number of ports per Hub.

File

usb_host_hub_config_template.h

C

```
#define USB_HOST_HUB_PORTS_NUMBER
```

Description

USB Host Hub Number of Ports per Hub

This configuration macros specifies the number of Ports per Hub. If any Hub connected to host will a have a maximum of 4 ports, then this number should be set to 4. A hub with more ports than the value defined by this constant will not be supported.

Remarks

Supporting a hub with more ports increases the memory requirement.

Building the Library

Describes the files to be included in the project while using the USB Hub Host Client Driver.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/usb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
usb_host_hub.h	This header file should be included in any `.c` file that accesses the USB Hub Host Client Driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
/src/dynamic/usb_host_hub.c	This file implements the USB Hub Host Client Driver interface and should be included in the project if the USB Hub Host Client Driver operation is desired.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The USB Hub Host Client Driver Library depends on the following modules:

- USB Host Layer Library

Library Interface

Data Types and Constants

	Name	Description
	USB_HOST_HUB_INTERFACE	USB Hub Host Client Driver Interface Pointer.

Description

This section describes the Application Programming Interface (API) functions of the USB Hub Host Client Driver Library.

Refer to each section for a detailed description.

Data Types and Constants

USB_HOST_HUB_INTERFACE Macro

USB Hub Host Client Driver Interface Pointer.

File

usb_host_hub.h

C

```
#define USB_HOST_HUB_INTERFACE
```

Description

USB Hub Host Client Driver Interface Pointer

This constant is a pointer to a table of function pointers that define the interface between the Hub Host Client Driver and the USB Host Layer. This constant should be used while adding support for the Hub Driver in TPL table.

Remarks

None.

Files

Files

Name	Description
usb_host_hub.h	USB Host Hub Client Driver Interface Header
usb_host_hub_config_template.h	USB host CDC Class configuration definitions template

Description

This section lists the source and header files used by the library.

usb_host_hub.h

USB Host Hub Client Driver Interface Header

Macros

	Name	Description
	USB_HOST_HUB_INTERFACE	USB Hub Host Client Driver Interface Pointer.

Description

USB Host Hub Client Driver Interface Definition

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB HOST Hub Client Driver.

File Name

usb_host_hub.h

Company

Microchip Technology Inc.

usb_host_hub_config_template.h

USB host CDC Class configuration definitions template

Macros

	Name	Description
	USB_HOST_HUB_INSTANCES_NUMBER	Specifies the number of Hub to be supported in the system.
	USB_HOST_HUB_PORTS_NUMBER	Specifies the number of ports per Hub.

Description

USB Host Hub Configuration Definitions

This file contains configurations macros needed to configure the Hub Driver. This file is a template file only. It should not be included by the application. The configuration macros defined in the file should be defined in the configuration specific system_config.h.

File Name

usb_host_hub_config_template.h

Company

Microchip Technology Inc.

USB MSD Host Client Driver Library

This section describes the USB MSD Host Client Driver Library.

Introduction

Introduces the MPLAB Harmony USB Mass Storage Device (MSD) Host Client Driver Library.

Description

The USB MSD Host Client Driver in the MPLAB Harmony USB Host Stack allows USB Host Applications to support and interact with Mass Storage Class (MSC) USB devices. Examples of such devices are USB Pen Drives and USB Card readers. The USB MSD Host Client Driver along with the SCSI Block Storage Driver Library implement a multi-layer solution to reading and writing to mass storage USB device that implement the SCSI command protocol. The USB MSD Host Client Driver has the following features:

- Implements the Bulk Only Transport (BOT) protocol in the USB MSD specification
- Supports multiple instances, which allows the application to interact with multiple storage devices
- Supports multi-LUN devices such as USB Card Reader
- Automatically (without application intervention) attaches the SCSI Block Driver to an identified device
- Implements automatic clearing of endpoint stall conditions
- Implements all three stages of a BOT transfer and provide a simple event driver transfer interface to the top-level application (which is typically a block storage driver library such as the SCSI Block Storage Driver Library)
- Typically operates without application intervention. The BOT transfers are typically invoked by the SCSI Block Storage Driver Library.

Using the Library

This topic describes the basic architecture of the USB MSD Host Client Drier Library and provides information and examples on its use.

Library Overview

Provides an overview of the USB Host MSD Library.

Description

The USB MSD Host Client Driver can be grouped functionally as shown in the following table.

Library Interface Section	Descriptions
Data Transfer Functions	These functions allow the application client to transfer data to the attached device.

Abstraction Model

Describes the Abstraction Model of the USB MSD Host Client Driver Library.

Description

The USB MSD Host Client Driver provides the transport for SCSI commands that implements that media read, write and control operations. If abstracts the details of initiating and completing a BOT transfer and performing error handling and presents a simple event driven interface to the top-level block storage command driver library.

The USB MSD Host Client Driver uses the USB Host data transfer and pipe management routines to implement the three stages of a BOT transfer. The library accepts a SCSI command from the SCSI Block Storage driver and transports this command in the command block of the Command Block Wrapper in the CBW stage of the BOT transfer. If the command requires a data stage, the USB MSD Host Client Driver library will transfer data between the USB Host and the device. The USB MSD Host Client Driver will then terminate the BOT transfer by requesting for the Command Status Wrapper (CSW) from the device.

If the device stalls any stage of the transfer, the USB MSD Host Client Driver will clear the stall and will automatically initiate the CSW stage to complete the transfer. The transfer result is communicated to the top level block storage driver library through a callback mechanism.

The USB Host layer will attach the USB MSD Host Client Driver to a mass storage device based on a TPL entry match. The USB MSD Host Client Driver will then open data and communication control pipes to the device. It will first get the number of logical units (LUN) that the device contains. It will then initialize the SCSI block storage driver for each reported LUN and mark the device state as being ready for data transfers.

How the Library Works

Describes how the library works and how it should be used.

Description

The USB MSD Host Client Driver provides the top level block storage driver with an easy to use, event driven, interface to transport the block storage command and data between the block storage command driver library and a compliant mass storage device. The USB MSD Host Client Driver in the MPLAB Harmony USB Host stack immediately supports mass storage devices that advertise support of the SCSI command set. Indeed, most of the commercially available USB storage devices such as USB pen driver and USB card readers respond to SCSI command requests.

The process of initializing the SCSI block storage driver library when a device is attached is performed automatically, by the USB MSD Host Client Driver. This does not require user application intervention. The following sections describe the TPL table design (application responsibility) and USB MSD Host Client Driver data transfer function (typically called by the SCSI block storage driver library).

MSD TPL Table Configuration

Describes TPL table design for matching MSD devices.

Description

The Host Layer attaches the MSD Host Client Driver to a device when the class, subclass and protocol fields in the Interface Association Descriptor (IAD) or Interface descriptor match the entry in the TPL table. When specifying the entry for the MSD device, the driver interface must be set to USB_HOST_MSD_INTERFACE. This will attach the USB MSD Host Client Driver to the device when the USB Host matches the TPL entry to the device. The following code shows a TPL table design for matching MSD Devices.

Example:

```
/* This code shows an example TPL table entry for supporting a Mass
 * Storage Device */

const USB_HOST_TPL_ENTRY USBTPList[ 1 ] =
{
    TPL_INTERFACE_CLASS_SUBCLASS_PROTOCOL(USB_MSD_CLASS_CODE,
            USB_MSD_SUBCLASS_CODE_SCSI_TRANSPARENT_COMMAND_SET, USB_MSD_PROTOCOL, NULL,
USB_HOST_MSD_INTERFACE)
```

```
};
```

Data Transfer

Describes how to transfer data, which includes a code example.

Description

The USB MSD Host Client Driver data transfer function is typically called by the SCSI Block Storage Driver Library. The USB_HOST_MSD_Transfer function allows the SCSI Block Storage Driver to transport SCSI commands to the mass storage device. The cdb parameter and the cdbLength parameter of the function specify the command and its size respectively. If the command requires the transport of data, then data must contain the pointer to the buffer and size specifies the amount of data expected to be transported. When the BOT transfer complete, the USB MSD Host Client Diver will call the callback function. The following code snippet shows an example of using the USB_HOST_MSD_Transfer function.

Example:

```
/* This code shows usage of the USB_HOST_MSD_Transfer function. The SCSI Block
 * Driver Library uses this function to send a SCSI Inquiry Command to the
 * device. Note how the commandCompleted flag in the SCSI instance object
 * tracks the completion of the transfer. This flag is updated in the transfer
 * callback. */

void _USB_HOST_SCSI_TransferCallback
(
    USB_HOST_MSD_LUN_HANDLE lunHandle,
    USB_HOST_MSD_TRANSFER_HANDLE transferHandle,
    USB_HOST_MSD_RESULT result,
    size_t size,
    uintptr_t context
)
{
    int scsiObjIndex;
    USB_HOST_SCSI_OBJ * scsiObj;
    USB_HOST_SCSI_COMMAND_OBJ * commandObj;
    USB_HOST_SCSI_EVENT event;

    /* Get the SCSI object index from the lunHandle */
    scsiObjIndex = _USB_HOST_SCSI_LUNHandleToSCSIInstance(lunHandle);

    /* Get the pointer to the SCSI object */
    scsiObj = &gUSBHostSCSIObj[scsiObjIndex];

    /* Pointer to the command object */
    commandObj = &scsiObj->commandObj;

    /* The processed size */
    commandObj->size = size;

    /* The result of the command */
    commandObj->result = result;

    /* Let the main state machine know that the command is completed */
    commandObj->commandCompleted = true;

    /* The rest of code is not shown here for the sake of brevity */
}

void USB_HOST_SCSI_Tasks(USB_HOST_MSD_LUN_HANDLE lunHandle)
{
    switch(scsiObj->state)
    {
        /* For the sake of brevity, only one SCSI command is show here */
        case USB_HOST_SCSI_STATE_INQUIRY_RESPONSE:

            /* We get the SCSI Enquiry response. Although there isn't much
             * that we can do with this data */
            _USB_HOST_SCSI_InquiryResponseCommand(scsiObj->commandObj.cdb);

            /* The commandCompleted flag will be updated in the callback.
             * Update the state and send the command.   */
```

```
            scsiObj->commandObj.inUse = true;
            scsiObj->commandObj.commandCompleted = false;
            scsiObj->commandObj.generateEvent = false;

            result = USB_HOST_MSD_Transfer(scsiObj->lunHandle,
                    scsiObj->commandObj.cdb, 6, scsiObj->buffer, 36,
                    USB_HOST_MSD_TRANSFER_DIRECTION_DEVICE_TO_HOST,
                    _USB_HOST_SCSI_TransferCallback, (uintptr_t)(scsiObj));

            if(result == USB_HOST_MSD_RESULT_SUCCESS)
            {
                scsiObj->state = USB_HOST_SCSI_STATE_WAIT_INQUIRY_RESPONSE;
            }

            break;

        default:
            break;

    }
}
```

Configuring the Library

Describes how to configure the USB Host Layer.

Macros

	Name	Description
	USB_HOST_MSD_INSTANCES_NUMBER	Defines the maximum number of MSD devices to be supported by this host application.
	USB_HOST_MSD_LUNS_NUMBER	Defines the maximum number of MSD Device LUNs to be supported in the application.

Description

The USB MSD Host Client Driver requires configuration constants to be specified in `system_config.h` file. These constants define the build time configuration (functionality and static resources) of the USB MSD Host Client Driver.

USB_HOST_MSD_INSTANCES_NUMBER Macro

Defines the maximum number of MSD devices to be supported by this host application.

File

usb_host_msd_config_template.h

C

```
#define USB_HOST_MSD_INSTANCES_NUMBER
```

Description

USB Host MSD Client Driver Instances Number.

This constant defines the maximum number of MSD devices to be supported by this host application. For example, if 3 USB Pen Drives need to be supported, then this value should be 3. This value cannot be greater than USB_HOST_DEVICES_NUMBER, which defines the maximum number of devices to be supported in the application. If this constant is less than USB_HOST_DEVICES_NUMBER, then only USB_HOST_MSD_INSTANCES_NUMBER of MSD devices will be enumerated.

Remarks

None

USB_HOST_MSD_LUNS_NUMBER Macro

Defines the maximum number of MSD Device LUNs to be supported in the application.

File

usb_host_msd_config_template.h

C

```
#define USB_HOST_MSD_LUNS_NUMBER
```

Description

USB Host MSD Client Driver LUNs Number.

An MSD device may have mutliple storage units, each addressable through a LUN number. An example is a USB Card reader with multiple card slots. Each card slot has a LUN number. The USB_HOST_MSD_LUNS_NUMBER constant defines the maximum number of such logical units that can be managed by the USB Host application. This number should atleast be equal to USB_HOST_MSD_INSTANCES_NUMBER. To configure this value, consider an example of an application that will support a maximum of 2 USB Storage device. These 2 storage devices are expected to have at the most 3 LUNs each. Then the the USB_HOST_MSD_LUNS_NUMBER constant should be set to 6 (2 devices and 3 LUNs per device)/

Remarks

None

Building the Library

Describes the files to be included in the project while using the MSD Host Function Driver.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/usb`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`usb_host_msd.h`	This header file should be included in any `.c` file that accesses the USB Host MSD Client Driver API.

Required File(s)

MHC ***All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.***

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/dynamic/usb_host_msd.c`	This file implements the USB Host MSD Client Driver interface and should be included in the project if USB Host MSD Client Driver operation is desired.

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
N/A	There are no optional files for this library.

Module Dependencies

The USB MSD Host Library depends on the following modules:

- USB Host Layer Library

Library Interface

a) Data Transfer Functions

	Name	Description
	USB_HOST_MSD_Transfer	This function schedules a MSD BOT transfer.
	USB_HOST_MSD_TransferErrorTasks	This function maintains the MSD transfer error handling state machine.

b) Data Types and Constants

	Name	Description
	USB_HOST_MSD_RESULT	USB HOST MSD Result
	USB_HOST_MSD_TRANSFER_CALLBACK	USB HOST MSD Transfer Complete Callback
	USB_HOST_MSD_TRANSFER_DIRECTION	USB HOST MSD Transfer Direction.
	USB_HOST_MSD_TRANSFER_HANDLE	USB HOST MSD Transfer Handle
	USB_HOST_MSD_INTERFACE	USB HOST MSD Client Driver Interface
	USB_HOST_MSD_TRANSFER_HANDLE_INVALID	USB HOST MSD Transfer Handle Invalid
	USB_HOST_MSD_LUN_HANDLE	USB HOST MSD LUN Handle
	USB_HOST_MSD_LUN_HANDLE_INVALID	USB HOST MSD LUN Handle Invalid
	USB_HOST_MSD_ERROR_CODE	USB Host MSD Error Codes.

Description

a) Data Transfer Functions

USB_HOST_MSD_Transfer Function

This function schedules a MSD BOT transfer.

File

usb_host_msd.h

C

```
USB_HOST_MSD_RESULT USB_HOST_MSD_Transfer(USB_HOST_MSD_LUN_HANDLE lunHandle, uint8_t * cdb, uint8_t
cdbLength, void * data, size_t size, USB_HOST_MSD_TRANSFER_DIRECTION transferDirection,
USB_HOST_MSD_TRANSFER_CALLBACK callback, uintptr_t context);
```

Returns

USB_HOST_MSD_RESULT_FAILURE - An unknown failure occurred. USB_HOST_MSD_RESULT_BUSY - The transfer cannot be scheduled right now. The caller should retry. USB_HOST_MSD_RESULT_LUN_HANDLE_INVALID - This LUN does not exist in the system. USB_HOST_MSD_RESULT_SUCCESS - The transfer request was scheduled.

Description

This function schedules a MSD BOT transfer. The command to be executed is specified in the cdb. This should be pointer to a 16 byte command descriptor block. The actual length of the command is specified by cdbLength. If there is data to be transferred, the pointer to the buffer is specified by data. The size of the buffer is specified in size. When the transfer completes, the callback function will be called. The context will be returned in the callback function.

Remarks

This is a local function and should not be called directly by the application.

Preconditions

None.

Parameters

Parameters	Description
cdb	pointer to the command to be executted. Should be a pointer to a 16 byte array. Unused bytes should be zero-padded.
cdbLength	Actual size of the command.
data	pointer to the data buffer if a data stage is involved.
size	size of the data buffer.
callback	callback function to called when the transfer has completed.
transferDirection	specifies the direction of the MSD transfer.
context	caller defined context that is returned in the callback function.

Function

USB_HOST_MSD_RESULT USB_HOST_MSD_Transfer
(
uint8_t * cdb,
uint8_t cdbLength,
void * data,
size_t size,
USB_HOST_MSD_TRANSFER_CALLBACK callback,
uintptr_t context
)

USB_HOST_MSD_TransferErrorTasks Function

This function maintains the MSD transfer error handling state machine.

File

usb_host_msd.h

C

```
void USB_HOST_MSD_TransferErrorTasks(USB_HOST_MSD_LUN_HANDLE lunHandle);
```

Returns

None.

Description

This function maintains the MSD transfer error handling state machine. This function should be called periodically after the USB_HOST_MSD_Transfer function has been called to schedule a transfer. The function should be called periodically atleast till the transfer completion event has been received. Calling this function while a BOT transfer is in progress allows the MSD Host Client driver to perform BOT error handling in a non-blocking manner.

Calling this function when there is no BOT transfer in progress will not have any effect. In case of BOT error handling, calling this function will eventually result in a BOT transfer event. It is not necessary to call this function after this event has occurred (till the next BOT transfer has been scheduled).

Remarks

While running in an RTOS application, this function should be called in the same thread that requested the BOT Transfer and operating the logical unit.

Preconditions

The lunHandle should be valid.

Parameters

Parameters	Description
lunHandle	handle to valid LUN.

Function

void USB_HOST_MSD_TransferErrorTasks
(
USB_HOST_MSD_LUN_HANDLE lunHandle,
);

b) Data Types and Constants

USB_HOST_MSD_RESULT Enumeration

USB HOST MSD Result

File

usb_host_msd.h

C

```
typedef enum {
  USB_HOST_MSD_RESULT_COMMAND_PASSED = 0,
  USB_HOST_MSD_RESULT_COMMAND_FAILED = 1,
  USB_HOST_MSD_RESULT_COMMAND_PHASE_ERROR = 2,
  USB_HOST_MSD_RESULT_SUCCESS,
  USB_HOST_MSD_RESULT_FAILURE,
  USB_HOST_MSD_RESULT_BUSY,
  USB_HOST_MSD_RESULT_LUN_HANDLE_INVALID,
  USB_HOST_MSD_RESULT_COMMAND_STALLED
} USB_HOST_MSD_RESULT;
```

Members

Members	Description
USB_HOST_MSD_RESULT_COMMAND_PASSED = 0	MSD Command result was success. The command issued to the MSD device • passed.
USB_HOST_MSD_RESULT_COMMAND_FAILED = 1	MSD Command failed. The command issued to the MSD device failed. The • device BOT state machine is in sync with the host. The data residue • length is valid.
USB_HOST_MSD_RESULT_COMMAND_PHASE_ERROR = 2	MSD Command failed with phase error. The command issued to the MSD device • has failed. The failure reason is unknown. The MSD Host Client driver has • reset the device BOT state machine.
USB_HOST_MSD_RESULT_SUCCESS	The operation was successful
USB_HOST_MSD_RESULT_FAILURE	An unknown failure has occurred.
USB_HOST_MSD_RESULT_BUSY	The request cannot be accepted at this time
USB_HOST_MSD_RESULT_LUN_HANDLE_INVALID	The specified LUN is not valid
USB_HOST_MSD_RESULT_COMMAND_STALLED	The MSD request was stalled

Description

USB HOST MSD Result

This enumeration defines the possible return values of different USB HOST MSD Client driver function call. Refer to the specific function documentation for details on the return values.

Remarks

None.

USB_HOST_MSD_TRANSFER_CALLBACK Type

USB HOST MSD Transfer Complete Callback

File

usb_host_msd.h

C

```
typedef void (* USB_HOST_MSD_TRANSFER_CALLBACK)(USB_HOST_MSD_LUN_HANDLE lunHandle,
USB_HOST_MSD_TRANSFER_HANDLE transferHandle, USB_HOST_MSD_RESULT result, size_t size, uintptr_t context);
```

Description

USB HOST MSD Transfer Complete Callback

This type defines the type of the callback function that the application must register in the USB_HOST_MSD_Transfer function to receive notification when a transfer has completed. The callback function will be called with the following parameters.

lunHandle - The handle to the LUN from where this notification originated.

transferHandle - the handle to the MSD transfer.

result - result of the transfer.

size - of the transfer.

context - context that specified when this transfer was scheduled.

Remarks

None.

USB_HOST_MSD_TRANSFER_DIRECTION Enumeration

USB HOST MSD Transfer Direction.

File

usb_host_msd.h

C

```
typedef enum {
  USB_HOST_MSD_TRANSFER_DIRECTION_HOST_TO_DEVICE = 0x00,
  USB_HOST_MSD_TRANSFER_DIRECTION_DEVICE_TO_HOST = 0x80
} USB_HOST_MSD_TRANSFER_DIRECTION;
```

Members

Members	Description
USB_HOST_MSD_TRANSFER_DIRECTION_HOST_TO_DEVICE = 0x00	Data moves from host to device
USB_HOST_MSD_TRANSFER_DIRECTION_DEVICE_TO_HOST = 0x80	Data moves from device to host

Description

USB HOST Transfer Direction

This enumeration specifies the direction of the data stage.

Remarks

None.

USB_HOST_MSD_TRANSFER_HANDLE Type

USB HOST MSD Transfer Handle

File

usb_host_msd.h

C

```
typedef uintptr_t USB_HOST_MSD_TRANSFER_HANDLE;
```

Description

USB HOST MSD Transfer Handle

This type defines a USB Host MSD Transfer Handle.

Remarks

None.

USB_HOST_MSD_INTERFACE Macro

USB HOST MSD Client Driver Interface

File

usb_host_msd.h

C

```
#define USB_HOST_MSD_INTERFACE
```

Description

USB HOST MSD Client Driver Interface

This macro should be used by the application in TPL table while adding support for the USB MSD Host Client Driver.

Remarks

None.

USB_HOST_MSD_TRANSFER_HANDLE_INVALID Macro

USB HOST MSD Transfer Handle Invalid

File

usb_host_msd.h

C

```
#define USB_HOST_MSD_TRANSFER_HANDLE_INVALID
```

Description

USB HOST MSD Transfer Handle Invalid

This value defines an invalid Transfer Handle.

Remarks

None.

USB_HOST_MSD_LUN_HANDLE Type

USB HOST MSD LUN Handle

File

usb_host_msd.h

C

```
typedef uintptr_t USB_HOST_MSD_LUN_HANDLE;
```

Description

USB HOST MSD LUN Handle

This type defines a MSD LUN Handle. This handle is used by SCSI driver to identify the LUN.

Remarks

None.

USB_HOST_MSD_LUN_HANDLE_INVALID Macro

USB HOST MSD LUN Handle Invalid

File

usb_host_msd.h

C

```
#define USB_HOST_MSD_LUN_HANDLE_INVALID
```

Description

USB HOST MSD LUN Handle Invalid

This value defines an invalid LUN Handle.

Remarks

None.

USB_HOST_MSD_ERROR_CODE Enumeration

USB Host MSD Error Codes.

File

usb_host_msd.h

C

```
typedef enum {
  USB_HOST_MSD_ERROR_CODE_INSUFFICIENT_INSTANCES = 1,
```

```
    USB_HOST_MSD_ERROR_CODE_NOT_FOUND_BULK_IN_ENDPOINT,
    USB_HOST_MSD_ERROR_CODE_NOT_FOUND_BULK_OUT_ENDPOINT,
    USB_HOST_MSD_ERROR_CODE_FAILED_PIPE_OPEN,
    USB_HOST_MSD_ERROR_CODE_FAILED_GET_MAX_LUN,
    USB_HOST_MSD_ERROR_CODE_FAILED_BOT_TRANSFER,
    USB_HOST_MSD_ERROR_CODE_FAILED_RESET_RECOVERY,
    USB_HOST_MSD_ERROR_CODE_CBW_STALL_RESET_RECOVERY,
    USB_HOST_MSD_ERROR_CODE_TRANSFER_BUSY,
    USB_HOST_MSD_ERROR_CODE_CSW_PHASE_ERROR,
    USB_HOST_MSD_ERROR_CODE_CSW_UNKNOWN_ERROR
} USB_HOST_MSD_ERROR_CODE;
```

Members

Members	Description
USB_HOST_MSD_ERROR_CODE_INSUFFICIENT_INSTANCES = 1	This error occurs when the number of MSD instances defined via • USB_HOST_MSD_INSTANCES_NUMBER (in system_config.h) is insufficient. For • example, this error would occur if the value of • USB_HOST_MSD_INSTANCES_NUMBER is 2, two MSC devices are already connected • and third MSC device is connected to the host. The object identifier in • this case will be the USB_HOST_DEVICE_OBJ_HANDLE value.
USB_HOST_MSD_ERROR_CODE_NOT_FOUND_BULK_IN_ENDPOINT	This error occurs when the driver descriptor parser could not find a Bulk • IN endpoint in the interface descriptor. The object identifier in this • case will be the USB_HOST_DEVICE_OBJ_HANDLE value.
USB_HOST_MSD_ERROR_CODE_NOT_FOUND_BULK_OUT_ENDPOINT	This error occurs when the driver descriptor parser could not find a Bulk • OUT endpoint in the interface descriptor. The object identifier in this • case will be USB_HOST_DEVICE_OBJ_HANDLE value.
USB_HOST_MSD_ERROR_CODE_FAILED_PIPE_OPEN	This error occurs when the driver could not open a Bulk pipe. This • typically happens either due to a host layer error or due to insufficient • number of pipes (which is configured via USB_HOST_PIPES_NUMBER). The • object idenfier in this case will be USB_HOST_DEVICE_OBJ_HANDLE value.
USB_HOST_MSD_ERROR_CODE_FAILED_GET_MAX_LUN	This error occurs when the Get Max LUN request issued by the driver fails • for any reason. The object identifier in this case will be the MSC device • instance index.
USB_HOST_MSD_ERROR_CODE_FAILED_BOT_TRANSFER	This error occurs when any stage of the BOT has failed due to bus error • or an unknown failure. The object identifier in this case will be the MSC • device instance index.
USB_HOST_MSD_ERROR_CODE_FAILED_RESET_RECOVERY	This error occurs when the MSD Reset Recovery procedure has failed. A MSC • device should not fail a MSD Reset Recovery procedure. The object • identifier in this case will be the device instance index.

USB_HOST_MSD_ERROR_CODE_CBW_STALL_RESET_RECOVERY	This error code indicates a condtion where the CBW stage of the BOT was • stalled and the driver is about to launch MSD reset recovery. The • identifier in this case if the MSC Device instance index. This code is • generated from an interrupt context. The driver may continue to function • normally post this condition.
USB_HOST_MSD_ERROR_CODE_TRANSFER_BUSY	This error code indicates a condition where the BOT transfer could not be • initiated because a transfer is already in progress. The identifier in • this case is the MSC Device Instance Index. The driver may continue to • function normall post this condition. This condition may occur several • times.
USB_HOST_MSD_ERROR_CODE_CSW_PHASE_ERROR	This error code indicates a condition where the BOT transfer failed due a • phase error in the CSW stage of the BOT. The identifier in this case if • the MSC Device instance index. This code is generated from an interrupt • context. The driver may continue to function normally post this • condition.
USB_HOST_MSD_ERROR_CODE_CSW_UNKNOWN_ERROR	This error code indicates that a condition where an unknown error has • occured during the CSW stage of the BOT. The identifier in this case if • the MSC Device instance index. This code is generated from an interrupt • context. The driver may continue to function normally post this • condition.

Description

USB Host MSD Error Codes.

This enumeration defines the codes that the MSD Client Driver returns for possible errors that lead to the device being placed in an error state. The MSD client driver will not operate on a device which is in an error state. The error are returned in the USB_HOST_MSD_ErrorCallback function.

Remarks

None.

Files

Files

Name	Description
usb_host_msd.h	USB Host MSD Class Driver Interface Header
usb_host_msd_config_template.h	USB Host MSD configuration template header file

Description

usb_host_msd.h

USB Host MSD Class Driver Interface Header

Enumerations

	Name	Description
	USB_HOST_MSD_ERROR_CODE	USB Host MSD Error Codes.
	USB_HOST_MSD_RESULT	USB HOST MSD Result
	USB_HOST_MSD_TRANSFER_DIRECTION	USB HOST MSD Transfer Direction.

Functions

	Name	Description
	USB_HOST_MSD_Transfer	This function schedules a MSD BOT transfer.
	USB_HOST_MSD_TransferErrorTasks	This function maintains the MSD transfer error handling state machine.

Macros

	Name	Description
	USB_HOST_MSD_INTERFACE	USB HOST MSD Client Driver Interface
	USB_HOST_MSD_LUN_HANDLE_INVALID	USB HOST MSD LUN Handle Invalid
	USB_HOST_MSD_TRANSFER_HANDLE_INVALID	USB HOST MSD Transfer Handle Invalid

Types

	Name	Description
	USB_HOST_MSD_LUN_HANDLE	USB HOST MSD LUN Handle
	USB_HOST_MSD_TRANSFER_CALLBACK	USB HOST MSD Transfer Complete Callback
	USB_HOST_MSD_TRANSFER_HANDLE	USB HOST MSD Transfer Handle

Description

USB Host MSD Class Driver Interface Definition

This header file contains the function prototypes and definitions of the data types and constants that make up the interface to the USB Host MSD Class Driver.

File Name

usb_host_msd.h

Company

Microchip Technology Inc.

usb_host_msd_config_template.h

USB Host MSD configuration template header file

Macros

	Name	Description
	USB_HOST_MSD_INSTANCES_NUMBER	Defines the maximum number of MSD devices to be supported by this host application.
	USB_HOST_MSD_LUNS_NUMBER	Defines the maximum number of MSD Device LUNs to be supported in the application.

Description

USB Host MSD function driver compile time options

This file contains USB host MSD function driver compile time options(macros) that has to be configured by the user. This file is a template file and must be used as an example only. This file must not be directly included in the project.

File Name

usb_host_msd_config_template.h

Company

Microchip Technology Inc.

Index

****

\ .tplFlags.driverType = (TPL_FLAG_CLASS_SUBCLASS_PROTOCOL) enumeration 6982

\ .tplFlags.driverType = (TPL_FLAG_VID_PID) enumeration 6983

_

__A2DP_CODEC_AAC_H macro 232
__A2DP_CODEC_MPEG_H macro 232
__A2DP_CODEC_SBC_H macro 232
__A2DP_H macro 233
__A2DP_PRIVATE_H macro 233
__AVCTP_CONFIG_H macro 22
__AVCTP_H macro 285
__AVCTP_PACKET_H macro 285
__AVCTP_PRIVATE_H macro 285
__AVDTP_CONFIG_H macro 338
__AVDTP_CONTROL_H macro 338
__AVDTP_H macro 339
__AVDTP_PRIVATE_H macro 339
__AVRCP_COMMAND_H macro 414
__AVRCP_CONFIG_EVENT_HANDLERS_H macro 510
__AVRCP_CONFIG_H macro 29
__AVRCP_H macro 414
__AVRCP_PRIVATE_H macro 414
__BT_APP_CONFIG_H macro 20
__HCI_CONFIG_EVENT_HANDLERS_H macro 39
__HCI_CONFIG_H macro 39
__IMAGEDECODER_H__ macro 2960
__L2CAP_CONFIG_H macro 47
__L2CAP_CONFIG_HANDLERS_H macro 47
__RFCOMM_CONFIG_H macro 50
__SBC_H macro 832
__SDP_CONFIG_H macro 52
__SPP_CONFIG_H macro 54
__ss_aligntype macro 6060
_BERKELEY_API_HEADER_FILE macro 6061
_bt_a2dp_aac_config_t structure 228
_bt_a2dp_avdtp_mgr_callback function 222
_bt_a2dp_event_u union 226
_bt_a2dp_evt_open_and_start_stream_completed_s structure 229
_bt_a2dp_mgr_t structure 228
_bt_a2dp_mpeg_config_t structure 231
_bt_a2dp_sbc_config_t structure 231
_bt_a2dp_sbc_packet_info_t structure 232
_bt_at_parser_t structure 994
_bt_av_add_to_now_playing_s structure 395
_bt_av_battery_status_of_ct_s structure 395
_bt_av_capability_company_id_s structure 396
_bt_av_capability_event_id_s structure 396
_bt_av_command_t structure 393
_bt_av_displayable_character_set_s structure 397
_bt_av_element_attribute_s structure 397
_bt_av_element_attributes_s structure 398
_bt_av_element_id_s structure 398
_bt_av_get_element_attributes_s structure 398
_bt_av_notification_addressed_player_changed_s structure 399
_bt_av_notification_app_setting_changed_s structure 399
_bt_av_notification_battery_status_s structure 399
_bt_av_notification_playback_pos_changed_s structure 400
_bt_av_notification_playback_status_changed_s structure 400
_bt_av_notification_s structure 401
_bt_av_notification_system_status_changed_s structure 400
_bt_av_notification_track_changed_s structure 402
_bt_av_notification_uids_changed_s structure 402
_bt_av_notification_volume_changed_s structure 402
_bt_av_play_item_s structure 403
_bt_av_play_status_s structure 403
_bt_av_player_setting_current_values_s structure 403
_bt_av_player_setting_values_s structure 404
_bt_av_player_setting_values_text_s structure 404
_bt_av_player_settings_s structure 405
_bt_av_player_settings_text_s structure 405
_bt_av_player_text_s structure 406
_bt_av_register_notification_t structure 406
_bt_av_response_t structure 406
_bt_av_set_absolute_volume_s structure 407
_bt_av_set_addressed_player_s structure 407
_bt_avctp_allocate_channel function 270
_bt_avctp_allocate_message function 270
_bt_avctp_allocate_transport function 270
_bt_avctp_channel_t structure 278
_bt_avctp_event_u union 279
_bt_avctp_evt_channel_connected_t structure 280
_bt_avctp_evt_channel_disconnected_t structure 280
_bt_avctp_evt_command_cancelled_t structure 281
_bt_avctp_evt_command_received_t structure 281
_bt_avctp_evt_command_sent_t structure 281
_bt_avctp_evt_connection_failed_t structure 282
_bt_avctp_evt_response_cancelled_t structure 282
_bt_avctp_evt_response_received_t structure 282
_bt_avctp_evt_response_sent_t structure 283
_bt_avctp_find_channel function 270
_bt_avctp_find_transport function 271
_bt_avctp_free_channel function 271
_bt_avctp_free_message function 271
_bt_avctp_free_transport function 271
_bt_avctp_init_message_buffers function 271
_bt_avctp_init_signal function 272
_bt_avctp_l2cap_read_data_callback function 272
_bt_avctp_message_t structure 278
_bt_avctp_mgr_t structure 278
_bt_avctp_packet_assembler function 272
_bt_avctp_packet_t structure 284
_bt_avctp_send_ipid function 272
_bt_avctp_set_signal function 272
_bt_avctp_transport_t structure 279
_bt_avdtp_add_param_byte function 292
_bt_avdtp_add_param_uint function 292
_bt_avdtp_add_param_uintn function 311
_bt_avdtp_allocate_buffers function 292
_bt_avdtp_allocate_cmd function 293
_bt_avdtp_allocate_sep function 293
_bt_avdtp_allocate_sep_config function 293
_bt_avdtp_allocate_stream function 293

_bt_avdtp_allocate_transport_channel function 293
_bt_avdtp_allocate_transport_session_id function 294
_bt_avdtp_begin_tc_channel_operation function 294
_bt_avdtp_codec_op_decode_t structure 316
_bt_avdtp_codec_op_encode_t structure 316
_bt_avdtp_codec_op_param_u union 314
_bt_avdtp_codec_op_parse_config_t structure 316
_bt_avdtp_codec_op_parse_packet_t structure 317
_bt_avdtp_codec_op_serialize_config_t structure 317
_bt_avdtp_codec_t structure 318
_bt_avdtp_commit_tc_channel_operation function 294
_bt_avdtp_control_channel_accept_handler function 294
_bt_avdtp_control_channel_cmd_handler function 294
_bt_avdtp_control_channel_event_handler function 295
_bt_avdtp_control_channel_reject_handler function 295
_bt_avdtp_control_channel_t structure 318
_bt_avdtp_control_cmd_t structure 318
_bt_avdtp_ctrl_evt_data_received_t structure 319
_bt_avdtp_event_u union 319
_bt_avdtp_evt_abort_stream_requested_t structure 321
_bt_avdtp_evt_close_stream_completed_t structure 321
_bt_avdtp_evt_close_stream_requested_t structure 321
_bt_avdtp_evt_ctrl_channel_connected_s structure 322
_bt_avdtp_evt_ctrl_channel_disconnected_s structure 322
_bt_avdtp_evt_ctrl_connection_failed_s structure 367
_bt_avdtp_evt_delay_report_completed_t structure 366
_bt_avdtp_evt_discover_completed_t structure 323
_bt_avdtp_evt_get_sep_capabilities_completed_t structure 323
_bt_avdtp_evt_get_stream_configuration_completed_t structure 323
_bt_avdtp_evt_media_packet_received_t structure 324
_bt_avdtp_evt_media_packet_send_failed_t structure 324
_bt_avdtp_evt_media_packet_sent_t structure 324
_bt_avdtp_evt_open_stream_completed_t structure 325
_bt_avdtp_evt_open_stream_requested_t structure 325
_bt_avdtp_evt_reconfigure_stream_requested_t structure 326
_bt_avdtp_evt_sep_capabilities_received_t structure 326
_bt_avdtp_evt_sep_info_received_t structure 326
_bt_avdtp_evt_set_stream_configuration_completed_t structure 327
_bt_avdtp_evt_set_stream_configuration_requested_t structure 327
_bt_avdtp_evt_set_stream_configuration_t structure 368
_bt_avdtp_evt_start_stream_completed_t structure 328
_bt_avdtp_evt_start_stream_requested_t structure 328
_bt_avdtp_evt_stream_aborted_t structure 329
_bt_avdtp_evt_stream_closed_t structure 329
_bt_avdtp_evt_stream_configured_t structure 329
_bt_avdtp_evt_stream_opened_t structure 330
_bt_avdtp_evt_stream_reconfigure_completed_t structure 330
_bt_avdtp_evt_stream_reconfigured_t structure 330
_bt_avdtp_evt_stream_security_control_completed_t structure 331
_bt_avdtp_evt_stream_started_t structure 331
_bt_avdtp_evt_stream_suspended_t structure 331
_bt_avdtp_evt_suspend_stream_completed_t structure 332
_bt_avdtp_evt_suspend_stream_requested_t structure 332
_bt_avdtp_execute_tc_channel_operation function 295
_bt_avdtp_find_listening_stream function 295
_bt_avdtp_find_stream function 295
_bt_avdtp_find_stream_by_remote_sep_id function 296
_bt_avdtp_find_stream_by_sep_id function 296
_bt_avdtp_free_cmd function 296
_bt_avdtp_free_sep_config function 296
_bt_avdtp_free_stream function 296
_bt_avdtp_free_transport_channel function 297
_bt_avdtp_get_control_channel function 297
_bt_avdtp_init_cmd_buffers function 297
_bt_avdtp_init_sep_config_buffers function 297
_bt_avdtp_init_signal function 297
_bt_avdtp_is_sep_inuse function 298
_bt_avdtp_mgr_t structure 334
_bt_avdtp_open_control_channel_ex function 298
_bt_avdtp_read_caps function 312
_bt_avdtp_register_transport_channel_for_operation function 298
_bt_avdtp_send_command function 298
_bt_avdtp_send_media_packet function 299
_bt_avdtp_sep_capabilities_t structure 335
_bt_avdtp_sep_t structure 314
_bt_avdtp_set_signal function 299
_bt_avdtp_stream_t structure 336
_bt_avdtp_transport_channel_t structure 337
_bt_avdtp_transport_l2cap_read_data_callback function 299
_bt_avdtp_transport_l2cap_state_changed_callback function 299
_bt_avdtp_transport_session_t structure 338
_bt_avdtp_write_caps function 299
_bt_avrcp_allocate_browsing_cmd function 368
_bt_avrcp_allocate_browsing_response function 392
_bt_avrcp_allocate_bt_specific_cmd function 368
_bt_avrcp_allocate_bt_specific_response function 368
_bt_avrcp_allocate_channel function 369
_bt_avrcp_allocate_cmd function 369
_bt_avrcp_allocate_response function 369
_bt_avrcp_allocate_simple_panel_cmd function 369
_bt_avrcp_allocate_simple_panel_response function 369
_bt_avrcp_channel_t structure 394
_bt_avrcp_device_s structure 408
_bt_avrcp_event_u union 408
_bt_avrcp_evt_channel_connected_t structure 409
_bt_avrcp_evt_channel_disconnected_t structure 410
_bt_avrcp_evt_connection_failed_t structure 410
_bt_avrcp_evt_panel_command_received_t structure 410
_bt_avrcp_evt_panel_response_received_t structure 411
_bt_avrcp_evt_register_events_completed_t structure 411
_bt_avrcp_evt_search_completed_s structure 412
_bt_avrcp_find_channel function 370
_bt_avrcp_free_channel function 370
_bt_avrcp_free_cmd function 370
_bt_avrcp_general_reject function 392
_bt_avrcp_get_tick_count function 370
_bt_avrcp_handle_add_to_now_playing function 370
_bt_avrcp_handle_command function 371
_bt_avrcp_handle_command_sent function 371
_bt_avrcp_handle_get_capabilities function 371
_bt_avrcp_handle_get_current_player_application_setting_value function 371
_bt_avrcp_handle_get_element_attributes function 371
_bt_avrcp_handle_get_play_status function 372
_bt_avrcp_handle_get_player_application_setting_attribute_text function 372

_bt_avrcp_handle_get_player_application_setting_value_text function 372
_bt_avrcp_handle_inform_battery_status_of_ct function 372
_bt_avrcp_handle_inform_displayable_character_set function 372
_bt_avrcp_handle_list_player_application_setting_attributes function 373
_bt_avrcp_handle_list_player_application_setting_values function 373
_bt_avrcp_handle_play_item function 373
_bt_avrcp_handle_register_notification function 373
_bt_avrcp_handle_request_continuing_response function 373
_bt_avrcp_handle_response function 374
_bt_avrcp_handle_response_sent function 374
_bt_avrcp_handle_set_absolute_volume function 374
_bt_avrcp_handle_set_addressed_player function 374
_bt_avrcp_handle_set_player_application_setting_value function 374
_bt_avrcp_init_cmd_buffers function 273
_bt_avrcp_init_signal function 375
_bt_avrcp_init_timer function 375
_bt_avrcp_mgr_t structure 394
_bt_avrcp_register_next_notification function 375
_bt_avrcp_register_pending_notification function 375
_bt_avrcp_send_notifications function 375
_bt_avrcp_send_rejected_response function 393
_bt_avrcp_set_signal function 376
_bt_avrcp_start_timer function 376
_bt_avrcp_tg_handle_get_capabilities function 376
_bt_avrcp_tg_handle_get_element_attributes function 376
_bt_avrcp_tg_handle_get_play_status function 376
_bt_avrcp_tg_handle_inform_battery_status_of_ct function 377
_bt_avrcp_tg_handle_inform_displayable_character_set function 377
_bt_avrcp_tg_handle_register_notification function 377
_bt_avrcp_tg_handle_set_absolute_volume function 377
_bt_avrcp_write_command_header function 377
_bt_bdaddr_s structure 215
_bt_buffer_header_t structure 995
_bt_buffer_mgr_t structure 995
_bt_cp_header_s structure 366
_bt_device_t structure 512
_bt_hci_command_s structure 676
_bt_hci_conn_state_s structure 691
_bt_hci_ctrl_listener_t structure 692
_bt_hci_ctrl_notify_data_listeners function 554
_bt_hci_ctrl_notify_listeners function 554
_bt_hci_ctrl_state_s structure 677
_bt_hci_data_buffer_s structure 678
_bt_hci_data_s structure 679
_bt_hci_event_e union 680
_bt_hci_event_s structure 681
_bt_hci_evt_authentication_complete_t structure 681
_bt_hci_evt_command_complete_t structure 681
_bt_hci_evt_command_status_t structure 681
_bt_hci_evt_connection_complete_t structure 682
_bt_hci_evt_connection_request_t structure 682
_bt_hci_evt_disconnection_complete_t structure 682
_bt_hci_evt_encryption_change_s structure 682
_bt_hci_evt_mode_change_t structure 683
_bt_hci_evt_role_change_t structure 683
_bt_hci_get_tick_count function 554
_bt_hci_init_signal function 554
_bt_hci_init_timer function 554
_bt_hci_init_transport function 555
_bt_hci_inquiry_response_t structure 684
_bt_hci_le_advertising_report_t structure 692
_bt_hci_le_command_complete_handler function 555
_bt_hci_le_conn_state_t structure 684
_bt_hci_le_connect_parameters_t structure 684
_bt_hci_le_ctrl_state_t structure 685
_bt_hci_le_evt_connection_updated_t structure 685
_bt_hci_le_evt_read_remote_used_features_completed_t structure 686
_bt_hci_le_evt_read_support_params_t structure 686
_bt_hci_link_key_s structure 686
_bt_hci_listener_t structure 692
_bt_hci_notify_listeners function 555
_bt_hci_set_init_flags function 559
_bt_hci_set_signal function 555
_bt_l2cap_cfg_option_t structure 764
_bt_l2cap_channel_ext_t structure 780
_bt_l2cap_channel_t structure 763
_bt_l2cap_clear_channel_cmd_queue function 712
_bt_l2cap_cmd_config_req_t structure 765
_bt_l2cap_cmd_config_res_t structure 765
_bt_l2cap_cmd_conn_param_update_req_t structure 766
_bt_l2cap_cmd_conn_param_update_res_t structure 766
_bt_l2cap_cmd_connection_req_t structure 766
_bt_l2cap_cmd_connection_res_t structure 767
_bt_l2cap_cmd_disconnection_req_t structure 767
_bt_l2cap_cmd_echo_req_t structure 768
_bt_l2cap_cmd_echo_res_t structure 768
_bt_l2cap_cmd_header_t structure 769
_bt_l2cap_cmd_info_req_t structure 770
_bt_l2cap_cmd_info_res_t structure 770
_bt_l2cap_cmd_reject_param_t union 771
_bt_l2cap_cmd_reject_t structure 771
_bt_l2cap_command_t union 772
_bt_l2cap_connect_params_s structure 781
_bt_l2cap_eretr_handle_xmit_event function 712
_bt_l2cap_eretr_pack_config_request function 723
_bt_l2cap_eretr_recv function 723
_bt_l2cap_eretr_retr_frames function 712
_bt_l2cap_eretr_send_data function 723
_bt_l2cap_eretr_send_pending_frames function 723
_bt_l2cap_eretr_send_smart_data function 723
_bt_l2cap_eretr_xmit_event_e enumeration 772
_bt_l2cap_fcs function 713
_bt_l2cap_fixed_channel_s structure 772
_bt_l2cap_frame_desc_s structure 773
_bt_l2cap_get_tick_count function 713
_bt_l2cap_init_signal function 713
_bt_l2cap_init_timer function 713
_bt_l2cap_mgr_s structure 774
_bt_l2cap_notify_and_remove function 713
_bt_l2cap_option_flash_timeout_t structure 774
_bt_l2cap_option_max_mtu_t structure 775
_bt_l2cap_option_qos_t structure 775
_bt_l2cap_option_rfc_t structure 775
_bt_l2cap_option_unknown_t structure 775
_bt_l2cap_packet_t structure 776

_bt_l2cap_process_connect_signal function 726
_bt_l2cap_psm_s structure 776
_bt_l2cap_recv_req_seq_and_fbit function 724
_bt_l2cap_send_ack function 724
_bt_l2cap_send_commands_from_queue function 714
_bt_l2cap_send_i_or_rr_or_rnr function 724
_bt_l2cap_send_rej function 724
_bt_l2cap_send_rnr function 724
_bt_l2cap_send_rr function 725
_bt_l2cap_send_rr_or_rnr function 725
_bt_l2cap_set_signal function 714
_bt_l2cap_start_monitor_timer macro 761
_bt_l2cap_start_monitor_timer_if_not_running macro 762
_bt_l2cap_start_retr_timer macro 762
_bt_l2cap_start_retr_timer_if_not_running macro 762
_bt_l2cap_stop_monitor_timer macro 762
_bt_l2cap_stop_retr_timer macro 762
_bt_l2cap_xmit_event_param_t structure 777
_bt_le_evt_handler function 558
_bt_linkkey_notification_t structure 217
_bt_linkkey_request_t structure 217
_bt_linkkey_t structure 217
_bt_media_packet_t structure 314
_bt_memcpy function 975
_bt_packet_t structure 996
_bt_queue_element_t structure 996
_bt_rfcomm_allocate_channel function 795
_bt_rfcomm_clear_queue function 795
_bt_rfcomm_command_t structure 824
_bt_rfcomm_ctl_msg_t structure 825
_bt_rfcomm_dlc_t structure 826
_bt_rfcomm_find_channel function 796
_bt_rfcomm_get_mgr function 796
_bt_rfcomm_init_signal function 796
_bt_rfcomm_mgr_allocate_session function 796
_bt_rfcomm_mgr_l2cap_listen_callback function 796
_bt_rfcomm_mgr_notify_listeners function 797
_bt_rfcomm_mgr_t structure 826
_bt_rfcomm_server_channel_t structure 827
_bt_rfcomm_session_listener_t structure 827
_bt_rfcomm_session_t structure 828
_bt_rfcomm_set_signal function 797
_bt_sdp_client_init function 841
_bt_sdp_data_element_t structure 875
_bt_sdp_found_attr_list_t structure 876
_bt_sdp_packet_t structure 876
_bt_sdp_sequence_t structure 877
_bt_sdp_serialization_state_t structure 878
_bt_sdp_server_attribute_t structure 878
_bt_sdp_server_data_element_t structure 878
_bt_sdp_server_record_t structure 878
_bt_sdp_service_transaction_t structure 879
_bt_sdp_transaction_t structure 880
_bt_signal_t structure 214
_bt_spp_client_init function 898
_bt_spp_find_port function 889
_bt_spp_handle_rx function 889
_bt_spp_handle_tx function 889
_bt_spp_port_event_e enumeration 892
_bt_spp_port_state_e enumeration 893
_bt_spp_port_t structure 900
_bt_spp_rfcomm_read_data_callback function 889
_bt_spp_rfcomm_send_data_callback function 889
_bt_spp_send_status_e enumeration 894
_bt_ssp_io_capability structure 901
_bt_ssp_keypress_notification structure 901
_bt_ssp_oob_data structure 901
_bt_ssp_simple_pairing_complete structure 902
_bt_ssp_user_confirmation_request structure 902
_bt_ssp_user_passkey_notification structure 902
_bt_ssp_user_passkey_request structure 902
_bt_timer_id_enum enumeration 220
_bt_uuid_s structure 221
_bt_vcard_evt_prop_param_t structure 996
_bt_vcard_evt_prop_t structure 997
_bt_vcard_parser_t structure 997
_bt_xml_evt_attribute_t structure 997
_bt_xml_evt_attribute_value_t structure 998
_bt_xml_evt_tag_started_t structure 998
_bt_xml_parser_t structure 998
_btx_csr_autobaud_buffer_t structure 942
_btx_csr_bccmd_header_s structure 944
_btx_csr_bccmd_listener_t structure 965
_btx_csr_cached_temperature_s structure 944
_btx_csr_create_operator_c_s structure 963
_btx_csr_exec_hq_script_buffer_t structure 965
_btx_csr_exec_script_buffer_t structure 942
_btx_csr_pio_direction_mask_s structure 945
_btx_csr_pio_protection_mask_s structure 945
_btx_csr_pio_s structure 946
_btx_csr_rssi_acl_s structure 946
_btx_csr_script_t structure 946
_btx_csr_set_ps_vars_buffer_t structure 943
_btx_csr_strm_connect_s structure 964
_btx_csr_strm_get_sink_s structure 947
_btx_csr_strm_get_source_s structure 947
_btx_csr_var_u union 947
_btx_ti_codec_config_s structure 964
_btx_ti_exec_script_buffer_t structure 943
_btx_ti_exec_script_oem_buffer_t structure 943
_btx_ti_script_t structure 949
_calc_fcs function 797
_cmd_disconnection_res structure 778
_compact_buffer function 970
_DRV_AK4642_H macro 1473
_DRV_AK4953_H macro 1515
_DRV_AK7755_H macro 1547
_DRV_CAMERA_OVM7690_delayMS function 1381
_DRV_CAMERA_OVM7690_DMAEventHandler function 1380
_DRV_CAMERA_OVM7690_HardwareSetup function 1381
_DRV_COMMON_H macro 1351
_DRV_ENC28J60_Configuration structure 1585
_DRV_ENCX24J600_Configuration structure 1605
_DRV_I2C_INIT structure 1757
_DRV_I2S_DATA16 structure 1813
_DRV_I2S_DATA24 structure 1813

_DRV_I2S_DATA32 structure 1814
_DRV_MTCH6301_CLIENT_OBJECT structure 2145
_DRV_PMP_QUEUE_ELEMENT_OBJ structure 1892
_DRV_SDCARD_INIT structure 1923
_DRV_SPI_CLIENT_DATA structure 1963
_DRV_SPI_INIT structure 1959
_DRV_TOUCH_ADC10BIT_CLIENT_DATA structure 2107
_DRV_TOUCH_ADC10BIT_INIT structure 2107
_ETH_PATTERN_MATCH_MODE_ enumeration 4128
_expand_buffer function 970
_GFX_OBJ structure 2957
_hci_allocate_buffers function 555
_hci_allocate_cmd function 556
_hci_free_cmd function 556
_hci_init_cmd_buffers function 556
_hci_init_cmd_queues function 556
_hci_receive_start function 556
_hci_recv_sco_data_packet function 557
_hci_send_commands_from_queue function 557
_hci_send_data function 557
_hci_send_data_fragment function 557
_hci_send_data_from_queue function 557
_hci_transport_t structure 690
_hcitr_tih4_power_event_e enumeration 690
_IMG_FILE_FORMAT enumeration 2959
_IMG_FILE_SYSTEM_API structure 2959
_IMG_PIXEL_XY_RGB_888 structure 2960
_IPV6_ADDR_STRUCT structure 6455
_IPV6_DATA_SEGMENT_HEADER structure 6340
_IPV6_PACKET structure 6343
_IPV6_RX_FRAGMENT_BUFFER structure 6345
_is_empty_str function 975
_l2cap_allocate_buffers function 714
_l2cap_data_receive_callback function 714
_LIBQ_C_H_ macro 3109
_LIBQ_H macro 3153
_LIBQ_Q15_cos_Q2_13 function 3133
_LIBQ_Q15_sin_Q2_13 function 3132
_LIBQ_Q15FromFloat function 3143
_LIBQ_Q15FromString function 3146
_LIBQ_Q15Rand function 3142
_LIBQ_Q16_tan_Q2_29 function 3134
_LIBQ_Q16Div function 3127
_LIBQ_Q16Exp function 3131
_LIBQ_Q16Power function 3130
_LIBQ_Q16Sqrt function 3128
_LIBQ_Q2_13_acos_Q15 function 3137
_LIBQ_Q2_13_asin_Q15 function 3135
_LIBQ_Q2_13_atan_Q7_8 function 3139
_LIBQ_Q2_13_atan2_Q7_8 function 3141
_LIBQ_Q2_29_acos_Q31 function 3138
_LIBQ_Q2_29_acos_Q31_Fast function 3139
_LIBQ_Q2_29_asin_Q31 function 3136
_LIBQ_Q2_29_asin_Q31_Fast function 3137
_LIBQ_Q2_29_atan_Q16 function 3140
_LIBQ_Q2_29_atan2_Q16 function 3141
_LIBQ_Q3_12_log10_Q16 function 3128
_LIBQ_Q31_cos_Q2_29 function 3133
_LIBQ_Q31_sin_Q2_29 function 3132
_LIBQ_Q31FromFloat function 3144
_LIBQ_Q31Rand function 3143
_LIBQ_Q4_11_ln_Q16 function 3129
_LIBQ_Q5_10_log2_Q16 function 3130
_LIBQ_Q7_8_tan_Q2_13 function 3135
_LIBQ_ToFloatQ15 function 3145
_LIBQ_ToFloatQ31 function 3145
_LIBQ_ToStringQ15 function 3147
_library_under_test structure 6691
_NET_PRES_EncProviderObject structure 3179
_NET_PRES_TransportObject structure 3190
_pack_cmd_reject function 714
_pack_config_request function 715
_pack_config_response function 715
_pack_conn_param_update_request function 715
_pack_conn_param_update_response function 715
_pack_conn_request function 715
_pack_conn_response function 716
_pack_dconn_request function 716
_pack_dconn_response function 716
_pack_echo_request function 716
_pack_echo_response function 716
_pack_info_request function 717
_pack_info_response function 717
_PARM_EQUAL_FILTER structure 3085
_PARM_EQUAL_FILTER_16 structure 3086
_PARM_EQUAL_FILTER_32 structure 3086
_PLIB_UNSUPPORTED macro 1351
_process_config_req function 717
_process_config_res function 717
_process_conn_param_update_req function 717
_process_conn_param_update_res function 718
_process_conn_req function 718
_process_conn_res function 718
_process_dconn_req function 718
_process_dconn_res function 718
_process_echo_req function 719
_process_echo_res function 719
_process_info_req function 719
_process_info_res function 719
_process_reject function 719
_process_unknown_req function 720
_process_unknown_res function 720
_promptStr macro 5643
_PutPixel macro 2960
_Q0_15 type 3151
_Q0_31 type 3151
_Q15 type 3151
_Q15_16 type 3152
_Q15_MAX macro 3147
_Q15_MIN macro 3148
_Q16 type 3152
_Q16_MAX macro 3148
_Q16_MIN macro 3148
_Q2_13 type 3152
_Q2_13_MAX macro 3148
_Q2_13_MIN macro 3148

_Q2_29 type 3152
_Q2_29_MAX macro 3149
_Q2_29_MIN macro 3149
_Q3_12 type 3152
_Q3_12_MAX macro 3149
_Q3_12_MIN macro 3149
_Q31 type 3153
_Q31_MAX macro 3149
_Q31_MIN macro 3150
_Q4_11 type 3153
_Q4_11_MAX macro 3150
_Q4_11_MIN macro 3150
_Q5_10 type 3153
_Q5_10_MAX macro 3150
_Q5_10_MIN macro 3150
_Q7_8 type 3153
_Q7_8_MAX macro 3151
_Q7_8_MIN macro 3151
_read_bdaddr function 970
_read_cmd_reject function 720
_read_config_request function 720
_read_config_response function 720
_read_conn_param_update_request function 721
_read_conn_param_update_response function 721
_read_conn_request function 721
_read_conn_response function 721
_read_dconn_request function 721
_read_dconn_response function 722
_read_echo_request function 722
_read_echo_response function 722
_read_info_request function 722
_read_info_response function 722
_readb function 971
_readbn macro 993
_readi function 971
_readin function 971
_readl function 971
_readln function 971
_readui function 972
_readui macro 993
_readuin function 972
_readuin macro 993
_readul function 972
_readul macro 994
_readuln function 972
_readuln macro 994
_rfcomm_alloc_cmd_buffer function 788
_rfcomm_allocate_buffers function 788
_rfcomm_allocate_mx_cmd function 788
_rfcomm_free_cmd_buffer function 788
_rfcomm_get_tick_count function 789
_rfcomm_init_cmd_buffers function 789
_rfcomm_init_mgr function 789
_rfcomm_init_sessions function 789
_rfcomm_init_timer function 789
_rfcomm_l2cap_read_data_callback function 790
_rfcomm_l2cap_state_changed_callback function 797
_rfcomm_mx_process_fc function 790
_rfcomm_mx_process_pn function 790
_rfcomm_mx_process_rpn function 790
_rfcomm_process_cmd_frame_disc function 790
_rfcomm_process_cmd_frame_dm function 791
_rfcomm_process_cmd_frame_sabm function 791
_rfcomm_process_cmd_frame_ua function 791
_rfcomm_process_cmd_frame_uih function 791
_rfcomm_process_mx_fc_response function 791
_rfcomm_process_mx_msc_response function 792
_rfcomm_process_mx_pn_response function 792
_rfcomm_process_mx_rls_response function 792
_rfcomm_process_res_frame_disc function 792
_rfcomm_process_res_frame_dm function 792
_rfcomm_process_res_frame_sabm function 793
_rfcomm_process_res_frame_ua function 793
_rfcomm_process_res_frame_uih function 793
_rfcomm_send_command function 793
_rfcomm_send_dm_response function 793
_rfcomm_send_mx_fc_cmd function 794
_rfcomm_send_mx_msc_response function 794
_rfcomm_send_mx_nsc_response function 794
_rfcomm_send_mx_rls_cmd function 794
_rfcomm_send_mx_rls_response function 794
_rfcomm_send_mx_test_response function 795
_rfcomm_send_sabm_cmd function 795
_rfcomm_send_ua_response function 795
_sample_module_data_status enumeration 5586
_sample_module_init_data structure 5587
_sdp_alloc_svc_tran_buffer function 838
_sdp_alloc_tran_buffer function 838
_sdp_find_svc_transaction function 839
_sdp_find_transaction function 839
_sdp_free_svc_tran_buffer function 839
_sdp_free_tran_buffer function 839
_sdp_get_de_data_len function 839
_sdp_get_de_hdr_len function 840
_sdp_init_tran_buffers function 840
_sdp_read_de_header function 840
_sdp_write_data_element function 840
_SS_PADSIZE macro 6061
_SS_SIZE macro 6061
_SSP_AUTHENTICATION_REQUIREMENTS enumeration 902
_SSP_EVENT enumeration 904
_SSP_IO_CAPABILITY enumeration 903
_SSP_KEYPRESS_NOTIFICATION_TYPE enumeration 903
_SSP_MODE enumeration 903
_SSP_OOB_DATA_PRESENT enumeration 904
_str2ulong function 975
_str2ulong_dec function 972
_SYS_DEBUG_MESSAGE macro 5688
_SYS_DEBUG_PRINT macro 5688
_SYS_MSG_CONFIG_TEMPLATE__H macro 5865
_SYS_TMR_CONFIG_TEMPLATE_H macro 5944
_SYS_TOUCH_H macro 5974
_tag_MAC_DATA_SEGMENT structure 6375
_tag_TCPIP_HTTP_NET_USER_CALLBACK structure 6259
_tag_TCPIP_ICMP_ECHO_REQUEST structure 6275
_tag_TCPIP_MAC_PACKET structure 6381

_test_data structure 6691
_test_init_data structure 6692
_TFTP_CMD_TYPE enumeration 6624
_to_lower_case function 973
_ulong2str function 973
_ulong2str_dec function 973
_USART_MODULE_ID macro 5363
_USB_DEVICE_IRP structure 6846
_USB_HOST_HID_MOUSE_EventHandler function 7144
_USB_HOST_HID_MOUSE_Task function 7144
_USB_HOST_IRP structure 6849
_USB_HOST_IRP_STATUS enumeration 6851
_W macro 942
_write_bdaddr function 973
_writeb function 973
_writebn macro 994
_writei function 974
_writein function 974
_writel function 974
_writeln function 974
_writes function 974
_writesx function 975
_zero_memory function 975

0

0 enumeration 6983
0x0000 enumeration 6984
0xFF } enumeration 6986
0xFF enumeration 6985
0xFFFF } enumeration 6987
0xFFFF enumeration 6986

1

1 enumeration 6988
10-bit ADC Touch Driver Library 2093

A

a) Functions 6756
a2dp.h 1010
a2dp_codec_aac.h 1019
a2dp_codec_mpeg.h 1020
a2dp_codec_sbc.h 1021
A2DP_EVT_ABORT_STREAM_COMPLETED macro 233
A2DP_EVT_ABORT_STREAM_REQUESTED macro 233
A2DP_EVT_CLOSE_STREAM_COMPLETED macro 233
A2DP_EVT_CLOSE_STREAM_REQUESTED macro 234
A2DP_EVT_CTRL_CHANNEL_CONNECTED macro 234
A2DP_EVT_CTRL_CHANNEL_DISCONNECTED macro 234
A2DP_EVT_CTRL_CONNECTION_FAILED macro 234
A2DP_EVT_DISCOVER_SEP_COMPLETED macro 234
A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED macro 235
A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED macro 235
A2DP_EVT_MEDIA_PACKET_RECEIVED macro 235
A2DP_EVT_MEDIA_PACKET_SEND_FAILED macro 235
A2DP_EVT_MEDIA_PACKET_SENT macro 235
A2DP_EVT_NOTHING macro 236
A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED macro 236
A2DP_EVT_OPEN_STREAM_COMPLETED macro 236
A2DP_EVT_OPEN_STREAM_REQUESTED macro 236
A2DP_EVT_RECONFIGURE_STREAM_COMPLETED macro 236
A2DP_EVT_RECONFIGURE_STREAM_REQUESTED macro 237
A2DP_EVT_SEP_CAPABILITIES_RECEIVED macro 237
A2DP_EVT_SEP_INFO_RECEIVED macro 237
A2DP_EVT_SET_STREAM_CONFIGURATION macro 270
A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED macro 237
A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED macro 238
A2DP_EVT_START_STREAM_COMPLETED macro 238
A2DP_EVT_START_STREAM_REQUESTED macro 238
A2DP_EVT_STREAM_ABORTED macro 238
A2DP_EVT_STREAM_CLOSED macro 238
A2DP_EVT_STREAM_CONFIGURATION_RECEIVED macro 239
A2DP_EVT_STREAM_CONFIGURED macro 239
A2DP_EVT_STREAM_OPENED macro 239
A2DP_EVT_STREAM_RECONFIGURED macro 239
A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED macro 240
A2DP_EVT_STREAM_STARTED macro 240
A2DP_EVT_STREAM_SUSPENDED macro 240
A2DP_EVT_SUSPEND_STREAM_COMPLETED macro 240
A2DP_EVT_SUSPEND_STREAM_REQUESTED macro 240
A2DP_MANAGER_STATE_CONNECTING macro 241
A2DP_MANAGER_STATE_IDLE macro 241
a2dp_private.h 1022
A2DP_SINK_FEATURE_AMPLIFIER macro 241
A2DP_SINK_FEATURE_HEADPHONE macro 241
A2DP_SINK_FEATURE_RECORDER macro 241
A2DP_SINK_FEATURE_SPEAKER macro 242
A2DP_SOURCE_FEATURE_MICROPHONE macro 242
A2DP_SOURCE_FEATURE_MIXER macro 242
A2DP_SOURCE_FEATURE_PLAYER macro 242
A2DP_SOURCE_FEATURE_TUNER macro 242
AAC Decoder Library 1260
aac.h 1269
AAC_CHANNELS_1 macro 243
AAC_CHANNELS_2 macro 243
AAC_CHANNELS_ALL macro 243
AAC_Decode function 1264
AAC_Decoder function 1261
AAC_DECODER_STATES enumeration 1266
AAC_ERROR_COUNT_MAX macro 1267
AAC_FRAME_HEADER_SIZE macro 1268
AAC_GetSamplingFrequency function 1265
AAC_Initialize function 1265
AAC_OBJECT_TYPE_MPEG_2_LC macro 243
AAC_OBJECT_TYPE_MPEG_4_LC macro 243
AAC_OBJECT_TYPE_MPEG_4_LTP macro 244
AAC_OBJECT_TYPE_MPEG_4_SCALABLE macro 244
AAC_PROFILE macro 1266
AAC_RegisterDecoderEventHandlerCallback function 1265
AAC_SAMPLING_FREQUENCY_11025 macro 244
AAC_SAMPLING_FREQUENCY_12000 macro 244
AAC_SAMPLING_FREQUENCY_16000 macro 244
AAC_SAMPLING_FREQUENCY_22050 macro 245
AAC_SAMPLING_FREQUENCY_24000 macro 245
AAC_SAMPLING_FREQUENCY_32000 macro 245

AAC_SAMPLING_FREQUENCY_44100 macro 245
AAC_SAMPLING_FREQUENCY_48000 macro 245
AAC_SAMPLING_FREQUENCY_64000 macro 246
AAC_SAMPLING_FREQUENCY_8000 macro 246
AAC_SAMPLING_FREQUENCY_88200 macro 246
AAC_SAMPLING_FREQUENCY_96000 macro 246
AAC_SAMPLING_FREQUENCY_ALL macro 246
AAC_SAMPLING_FREQUENCY_INDEX enumeration 1267
AAC_VBR_NOT_SUPPORTED macro 247
AAC_VBR_SUPPORTED macro 247
AAC_WriteWavHeader function 1261
AACDecoder_GetFrameSize function 1262
AACDecoder_GetStateSize function 1262
AACDecoder_Init function 1263
AACDecoder_InterleaveSamples function 1263
AACDECODER_STATE_SIZE macro 1267
About CAN Protocol 3517
Abstract Control Model (ACM) 6859
Abstraction Model 1173, 1192, 1211, 1292, 1308, 1366, 1392, 1442, 1483, 1521, 1569, 1589, 1608, 1630, 1728, 1768, 1830, 1868, 1901, 1930, 1967, 2042, 2111, 2129, 2151, 2211, 2270, 2330, 2402, 2495, 3157, 3201, 3867, 5609, 5635, 5654, 5673, 5690, 5716, 5769, 5834, 5855, 5860, 5886, 5915, 5926, 5932, 5976, 6006, 6010, 6029, 6065, 6079, 6094, 6114, 6137, 6149, 6157, 6163, 6202, 6267, 6278, 6298, 6313, 6351, 6401, 6468, 6472, 6479, 6482, 6496, 6516, 6566, 6576, 6629, 6634, 6667, 6718, 6747, 6774, 6857, 6893, 6925, 6935, 6957, 7006, 7104, 7138, 7148, 7153
    AK4384 Codec Driver Library 1392
    AK4642 Codec Driver Library 1442
    AK4953 Codec Driver Library 1483
    AK7755 Codec Driver Library 1521
    Announce Module 6006
    AR1021 Touch Driver Library 2111
    ARP Module 6010
    Berkeley Module 6029
    Bootloader Library 1173
    Class B Library 1192
    Clock System Service Library 5609
    Command Processor System Service Library 5635
    Console System Service Library 5654
    Crypto Library 1211
    Debug System Service Library 5673
    Device Control System Service Library 5690
    DHCP Module 6065
    DHCP Server Module 6079
    DHCPv6 Module 6094
    DMA System Service Library 5716
    DNS Module 6114
    DNSS Module 6137
    Dual Data Rate (DDR) SDRAM Peripheral Library 3867
    Dynamic DNS Module 6149
    Ethernet MAC Driver Library 1608
    Ethernet PHY Driver Library 1630
    FTP Module 6157
    Generic USB Device Library 6935
    HTTP Module 6163
    HTTP Net Module 6202
    ICMP Module 6267
    ICMPv6 Module 6278
    Interrupt System Service Library 5834
    IPv4 Module 6298
    IPv6 Module 6313
    MAC Driver Module 6351
    Manager Module 6401
    Memory System Service Library 5855
    Messaging System Service Library 5860
    MRF24W Wi-Fi Driver Library 2402
    MRF24WN Wi-Fi Driver Library 2495
    MTCH6301 Touch Driver Library 2129
    MTCH6303 Touch Driver Library 2151
    NBNS Module 6468
    NDP Module 6472
    Network Presentation Layer 3157
    NVM Driver Library 1830
    OSAL Library 3201
    PMP Driver Library 1868
    Ports System Service Library 5886
    Reboot Module 6479
    RNG System Service Library 5915
    SD Card Driver Library 1901
    SMTP Module 6482
    SMTPC Module 6496
    SNMP Module 6516
    SNTP Module 6566
    SPI Driver Library 1930
    SPI Flash Driver Library 1967
    TCP Module 6576
    Telnet Module 6629
    Timer Driver Library 2042
    Timer System Service Library 5932
    UDP Module 6634
    USART Driver Library 2330
    USB CDC Device Library 6857
    USB Device Layer Library 6774
    USB MSD Device Library 6925
    Watchdog Timer System Service Library 5976
    Zeroconf Module 6667
accept function 6033
Accessing the Result Buffers 3245
ADC Driver Library 1353
ADC Peripheral Library 3237
ADC_ACQUISITION_TIME type 3298
ADC_BUFFER_MODE enumeration 3292
ADC_CLOCK_SOURCE enumeration 3292
ADC_CONVERSION_CLOCK type 3298
ADC_CONVERSION_TRIGGER_SOURCE enumeration 3292
ADC_INPUTS_NEGATIVE enumeration 3290
ADC_INPUTS_POSITIVE enumeration 3296
ADC_INPUTS_SCAN enumeration 3294
ADC_MODULE_ID enumeration 3289
ADC_MUX enumeration 3293
ADC_RESULT_BUF_STATUS enumeration 3293
ADC_RESULT_FORMAT enumeration 3296
ADC_SAMPLE type 3299
ADC_SAMPLES_PER_INTERRUPT enumeration 3291
ADC_SAMPLING_MODE enumeration 3294
ADC_VOLTAGE_REFERENCE enumeration 3290
ADCHS Peripheral Library 3301

ADCHS_AN_INPUT_ID enumeration 3413
ADCHS_CHANNEL_ID enumeration 3428
ADCHS_CHANNEL_INP_SEL enumeration 3427
ADCHS_CHANNEL_TRIGGER_SAMPLING_SEL enumeration 3429
ADCHS_CHARGEPUMP_MODE enumeration 3415
ADCHS_CLASS12_AN_INPUT_ID enumeration 3429
ADCHS_CLOCK_SOURCE enumeration 3415
ADCHS_CVD_CAPACITOR enumeration 3416
ADCHS_DATA_RESOLUTION enumeration 3416
ADCHS_DIGITAL_COMPARATOR_ID enumeration 3417
ADCHS_DIGITAL_FILTER_AVERAGE_SAMPLE_COUNT enumeration 3418
ADCHS_DIGITAL_FILTER_ID enumeration 3418
ADCHS_DIGITAL_FILTER_OVERSAMPLE_RATIO enumeration 3419
ADCHS_DIGITAL_FILTER_SIGNIFICANT_BITS enumeration 3419
ADCHS_DMA_BUFFER_LENGTH enumeration 3420
ADCHS_DMA_COUNT enumeration 3420
ADCHS_EARLY_INTERRUPT_PRIOR_CLOCK enumeration 3421
ADCHS_FAST_SYNC_PERIPHERAL_CLOCK enumeration 3421
ADCHS_FAST_SYNC_SYSTEM_CLOCK enumeration 3422
ADCHS_INPUT_MODE enumeration 3422
ADCHS_INTERRUPT_BIT_SHIFT_LEFT enumeration 3423
ADCHS_MODULE_ID enumeration 3427
ADCHS_OUTPUT_DATA_FORMAT enumeration 3423
ADCHS_SCAN_TRIGGER_SENSITIVE enumeration 3424
ADCHS_SCAN_TRIGGER_SOURCE enumeration 3424
ADCHS_TRIGGER_SOURCE enumeration 3425
ADCHS_VREF_SOURCE enumeration 3426
ADCHS_WARMUP_CLOCK enumeration 3426
ADCP Peripheral Library 3434
ADCP_CLASS12_INPUT_ID enumeration 3470
ADCP_CLOCK_SOURCE enumeration 3470
ADCP_DCMP_ID enumeration 3475
ADCP_DSH_ID enumeration 3471
ADCP_INPUT_ID enumeration 3472
ADCP_MODULE_ID enumeration 3469
ADCP_ODFLTR_ID enumeration 3475
ADCP_ODFLTR_OSR enumeration 3476
ADCP_SCAN_TRG_SRC enumeration 3476
ADCP_SH_ID enumeration 3471
ADCP_SH_MODE enumeration 3472
ADCP_TRG_SRC enumeration 3477
ADCP_VREF_SOURCE enumeration 3469
addrinfo structure 6048
adhocMode enumeration 2469
AF_INET macro 6044
AF_INET6 macro 6060
AK4384 Codec Driver Library 1391
AK4642 Codec Driver Library 1442
AK4953 Codec Driver Library 1481
AK7755 Codec Driver Library 1520
Alarm Functionality 2046
Alarm Mode Operations 4915
AN_READY type 3479
AN_SELECT union 3478
Announce Module 6006
Anti-aliased Fonts 2571
Application Client Interaction 6780
Application I/O (AppIO) 5657
Application Interaction 5771
AR1021 Touch Driver Library 2110
ARG_NOT_USED macro 204
ARP Changes 5990
ARP Module 6009
arp.h 6027
ARP_CACHE_DELETE_OLD macro 6011
arp_config.h 6028
Assigning Buffer Memory 3520
Asynchronous Counter 5248
Asynchronous USART Mode 5285
AT_EVT_CMD_CODE macro 976
AT_EVT_CMD_COMPLETED macro 976
AT_EVT_CMD_PARAM macro 976
AT_EVT_CMD_READ_CODE macro 976
AT_EVT_ERROR macro 976
AT_EVT_OK macro 977
AT_EVT_RING macro 977
at_parser.h 1080
ATCMD_BUFFER_LEN macro 977
ATT_ALLOCATE_BUFFERS function 15
ATT_CLIENT_ALLOCATE_BUFFERS function 16
Audio Data Streaming 7024
Audio Protocol Interface Mode 5028
Audio Stream Event Handling 7013
Authentication 6167
AVC_BATTERY_STATUS_CRITICAL macro 414
AVC_BATTERY_STATUS_EXTERNAL macro 415
AVC_BATTERY_STATUS_FULL_CHARGE macro 415
AVC_BATTERY_STATUS_NORMAL macro 415
AVC_BATTERY_STATUS_WARNING macro 415
AVC_CAPABILITY_COMPANY_ID macro 415
AVC_CAPABILITY_EVENTS_SUPPORTED macro 416
AVC_CMD_GENERAL_POWER macro 416
AVC_CMD_GENERAL_RESERVE macro 416
AVC_CMD_GENERAL_SUBUNIT_INFO macro 416
AVC_CMD_GENERAL_UNIT_INFO macro 416
AVC_CMD_GENERAL_VENDOR_DEPENDENT macro 417
AVC_CMD_GENERAL_VERSION macro 417
AVC_CMD_PANEL_PASS_THROUGH macro 417
AVC_CTYPE_CONTROL macro 417
AVC_CTYPE_GENERAL_INQUORY macro 417
AVC_CTYPE_NOTIFY macro 418
AVC_CTYPE_SPECIFIC_IQUIRY macro 418
AVC_CTYPE_STATUS macro 418
AVC_EVENT_ADDRESSED_PLAYER_CHANGED macro 418
AVC_EVENT_AVAILABLE_PLAYERS_CHANGED macro 418
AVC_EVENT_BATT_STATUS_CHANGED macro 419
AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED macro 419
AVC_EVENT_FLAG_ALL macro 419
AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED macro 419
AVC_EVENT_FLAG_BATT_STATUS_CHANGED macro 419
AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED macro 420
AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED macro 420
AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED macro 420
AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED

macro 420
AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED macro 420
AVC_EVENT_FLAG_TRACK_CHANGED macro 421
AVC_EVENT_FLAG_TRACK_REACHED_END macro 421
AVC_EVENT_FLAG_TRACK_REACHED_START macro 421
AVC_EVENT_FLAG_UIDS_CHANGED macro 421
AVC_EVENT_FLAG_VOLUME_CHANGED macro 421
AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED macro 422
AVC_EVENT_PLAYBACK_POS_CHANGED macro 422
AVC_EVENT_PLAYBACK_STATUS_CHANGED macro 422
AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED macro 422
AVC_EVENT_SYSTEM_STATUS_CHANGED macro 422
AVC_EVENT_TRACK_CHANGED macro 423
AVC_EVENT_TRACK_REACHED_END macro 423
AVC_EVENT_TRACK_REACHED_START macro 423
AVC_EVENT_UIDS_CHANGED macro 423
AVC_EVENT_VOLUME_CHANGED macro 423
AVC_FLAG_BROWSING_CMD macro 424
AVC_FLAG_PANEL_CLICK macro 424
AVC_FLAG_RESPONSE macro 424
AVC_MAXEVENTS macro 424
AVC_MEDIA_ATTR_FLAG_ALBUM macro 424
AVC_MEDIA_ATTR_FLAG_ALL macro 425
AVC_MEDIA_ATTR_FLAG_ARTIST macro 425
AVC_MEDIA_ATTR_FLAG_GENRE macro 425
AVC_MEDIA_ATTR_FLAG_NUMBER macro 425
AVC_MEDIA_ATTR_FLAG_PLAYING_TIME macro 425
AVC_MEDIA_ATTR_FLAG_TITLE macro 426
AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER macro 426
AVC_MEDIA_ATTR_ID_ALBUM macro 426
AVC_MEDIA_ATTR_ID_ARTIST macro 426
AVC_MEDIA_ATTR_ID_GENRE macro 426
AVC_MEDIA_ATTR_ID_NUMBER macro 427
AVC_MEDIA_ATTR_ID_PLAYING_TIME macro 427
AVC_MEDIA_ATTR_ID_TITLE macro 427
AVC_MEDIA_ATTR_ID_TOTAL_NUMBER macro 427
AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM macro 427
AVC_NOW_PLAYING macro 428
AVC_PACKET_TYPE_CONTINUE macro 428
AVC_PACKET_TYPE_END macro 428
AVC_PACKET_TYPE_SINGLE macro 428
AVC_PACKET_TYPE_START macro 428
AVC_PANEL_BUTTON_PRESSED macro 429
AVC_PANEL_BUTTON_RELEASED macro 429
AVC_PANEL_OPID_0 macro 429
AVC_PANEL_OPID_1 macro 429
AVC_PANEL_OPID_2 macro 429
AVC_PANEL_OPID_3 macro 430
AVC_PANEL_OPID_4 macro 430
AVC_PANEL_OPID_5 macro 430
AVC_PANEL_OPID_6 macro 430
AVC_PANEL_OPID_7 macro 430
AVC_PANEL_OPID_8 macro 431
AVC_PANEL_OPID_9 macro 431
AVC_PANEL_OPID_A macro 431
AVC_PANEL_OPID_ANGLE macro 431
AVC_PANEL_OPID_APPS_MENU macro 431
AVC_PANEL_OPID_B macro 432
AVC_PANEL_OPID_BACKWARD macro 432
AVC_PANEL_OPID_C macro 432
AVC_PANEL_OPID_CHANNEL_DOWN macro 432
AVC_PANEL_OPID_CHANNEL_UP macro 432
AVC_PANEL_OPID_CLEAR macro 433
AVC_PANEL_OPID_CONTENTS_MENU macro 433
AVC_PANEL_OPID_D macro 433
AVC_PANEL_OPID_DISPLAY_INFORMATION macro 433
AVC_PANEL_OPID_DOT macro 433
AVC_PANEL_OPID_DOWN macro 434
AVC_PANEL_OPID_EJECT macro 434
AVC_PANEL_OPID_ENTER macro 434
AVC_PANEL_OPID_EXIT macro 434
AVC_PANEL_OPID_F1 macro 434
AVC_PANEL_OPID_F2 macro 435
AVC_PANEL_OPID_F3 macro 435
AVC_PANEL_OPID_F4 macro 435
AVC_PANEL_OPID_F5 macro 435
AVC_PANEL_OPID_F6 macro 435
AVC_PANEL_OPID_F7 macro 436
AVC_PANEL_OPID_F8 macro 436
AVC_PANEL_OPID_F9 macro 436
AVC_PANEL_OPID_FAST_FORWARD macro 436
AVC_PANEL_OPID_FAVORITE_MENU macro 436
AVC_PANEL_OPID_FORWARD macro 437
AVC_PANEL_OPID_HELP macro 437
AVC_PANEL_OPID_INPUT_SELECT macro 437
AVC_PANEL_OPID_KEYBORD_FUNCTION macro 437
AVC_PANEL_OPID_LEFT macro 437
AVC_PANEL_OPID_LEFT_DOWN macro 438
AVC_PANEL_OPID_LEFT_UP macro 438
AVC_PANEL_OPID_LINKED_CONTENT macro 438
AVC_PANEL_OPID_LIST macro 438
AVC_PANEL_OPID_LIVE_TV macro 438
AVC_PANEL_OPID_LOCK macro 439
AVC_PANEL_OPID_MUTE macro 439
AVC_PANEL_OPID_MUTE_FUNCTION macro 439
AVC_PANEL_OPID_NEXT_DAY macro 439
AVC_PANEL_OPID_ON_DEMAND_MENU macro 439
AVC_PANEL_OPID_PAGE_DOWN macro 440
AVC_PANEL_OPID_PAGE_UP macro 440
AVC_PANEL_OPID_PAUSE macro 440
AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION macro 440
AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION macro 440
AVC_PANEL_OPID_PIP_DOWN macro 441
AVC_PANEL_OPID_PIP_MOVE macro 441
AVC_PANEL_OPID_PIP_UP macro 441
AVC_PANEL_OPID_PLAY macro 441
AVC_PANEL_OPID_PLAY_FUNCTION macro 441
AVC_PANEL_OPID_POWER_STATE_FUNCTION macro 442
AVC_PANEL_OPID_POWER_TOGGLE macro 442
AVC_PANEL_OPID_PREVIOUS_CHANNEL macro 442
AVC_PANEL_OPID_PREVIOUS_DAY macro 442
AVC_PANEL_OPID_RECORD macro 442
AVC_PANEL_OPID_RECORD_FUNCTION macro 443
AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION macro 443
AVC_PANEL_OPID_REWIND macro 443

AVC_PANEL_OPID_RF_BYPASS macro 443
AVC_PANEL_OPID_RIGHT macro 443
AVC_PANEL_OPID_RIGHT_DOWN macro 444
AVC_PANEL_OPID_RIGHT_UP macro 444
AVC_PANEL_OPID_ROOT_MENU macro 444
AVC_PANEL_OPID_SELECT macro 444
AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION macro 444
AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION macro 445
AVC_PANEL_OPID_SELECT_DISK_FUNCTION macro 445
AVC_PANEL_OPID_SETUP_MENU macro 445
AVC_PANEL_OPID_SKIP macro 445
AVC_PANEL_OPID_SOUND_SELECT macro 445
AVC_PANEL_OPID_STOP macro 446
AVC_PANEL_OPID_STOP_FUNCTION macro 446
AVC_PANEL_OPID_SUBPICTURE macro 446
AVC_PANEL_OPID_TUNE_FUNCTION macro 446
AVC_PANEL_OPID_UP macro 446
AVC_PANEL_OPID_VENDOR_UNIQUE macro 447
AVC_PANEL_OPID_VOLUME_DOWN macro 447
AVC_PANEL_OPID_VOLUME_UP macro 447
AVC_PANEL_OPID_ZOOM macro 447
AVC_PDUID_ABORT_CONTINUING_RESPONSE macro 447
AVC_PDUID_ADD_TO_NOW_PLAYING macro 448
AVC_PDUID_CHANGE_PATH macro 448
AVC_PDUID_GENERAL_REJECT macro 448
AVC_PDUID_GET_CURRENT_PLAYER_APPLICATION_SETTING_VALUE
macro 448
AVC_PDUID_GET_ELEMENT_ATTRIBUTES macro 448
AVC_PDUID_GET_FOLDER_ITEMS macro 449
AVC_PDUID_GET_ITEM_ATTRIBUTES macro 449
AVC_PDUID_GET_PLAY_STATUS macro 449
AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_ATTRIBUTE_TEXT
macro 449
AVC_PDUID_GET_PLAYER_APPLICATION_SETTING_VALUE_TEXT macro 449
AVC_PDUID_GETCAPABILITIES macro 450
AVC_PDUID_INFORM_BATTERY_STATUS_OF_CT macro 450
AVC_PDUID_INFORM_DISPLAYABLE_CHARACTER_SET macro 450
AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_ATTRIBUTES macro 450
AVC_PDUID_LIST_PLAYER_APPLICATION_SETTING_VALUES macro 450
AVC_PDUID_PLAY_ITEM macro 451
AVC_PDUID_REGISTER_NOTIFICATION macro 451
AVC_PDUID_REQUEST_CONTINUING_RESPONSE macro 451
AVC_PDUID_SEARCH macro 451
AVC_PDUID_SET_ABSOLUTE_VOLUME macro 451
AVC_PDUID_SET_ADDRESSED_PLAYER macro 452
AVC_PDUID_SET_BROWSED_PLAYER macro 452
AVC_PDUID_SET_PLAYER_APPLICATION_SETTING_VALUE macro 452
AVC_PLAY_STATUS_ERROR macro 452
AVC_PLAY_STATUS_FW_SEEK macro 452
AVC_PLAY_STATUS_PAUSED macro 453
AVC_PLAY_STATUS_PLAYING macro 453
AVC_PLAY_STATUS_REV_SEEK macro 453
AVC_PLAY_STATUS_STOPPED macro 453
AVC_PLAYER_SETTING_EQUALIZER_OFF macro 453
AVC_PLAYER_SETTING_EQUALIZER_ON macro 454
AVC_PLAYER_SETTING_EQUALIZER_STATUS macro 454
AVC_PLAYER_SETTING_REPEAT_ALL_TRACKS macro 454
AVC_PLAYER_SETTING_REPEAT_GROUP macro 454
AVC_PLAYER_SETTING_REPEAT_MODE_OFF macro 454
AVC_PLAYER_SETTING_REPEAT_MODE_STATUS macro 455
AVC_PLAYER_SETTING_REPEAT_SINGLE_TRACK macro 455
AVC_PLAYER_SETTING_SCAN_ALL_TRACKS macro 455
AVC_PLAYER_SETTING_SCAN_GROUP macro 455
AVC_PLAYER_SETTING_SCAN_OFF macro 455
AVC_PLAYER_SETTING_SCAN_STATUS macro 456
AVC_PLAYER_SETTING_SHUFFLE_ALL_TRACKS macro 456
AVC_PLAYER_SETTING_SHUFFLE_GROUP macro 456
AVC_PLAYER_SETTING_SHUFFLE_OFF macro 456
AVC_PLAYER_SETTING_SHUFFLE_STATUS macro 456
AVC_RESPONSE_ACCEPTED macro 457
AVC_RESPONSE_CHANGED macro 457
AVC_RESPONSE_IMPLEMENTED macro 457
AVC_RESPONSE_IN_TRANSITION macro 457
AVC_RESPONSE_INTERIM macro 457
AVC_RESPONSE_NOT_IMPLEMENTED macro 458
AVC_RESPONSE_REJECTED macro 458
AVC_RESPONSE_STABLE macro 458
AVC_RESPONSE_TIMEOUT macro 458
AVC_SCOPE_MEDIA_PLAYER_LIST macro 458
AVC_SEARCH macro 459
AVC_SUBUNIT_ID_EXTENDED_TO_NEXT_BYTE macro 459
AVC_SUBUNIT_ID_IGNORE macro 459
AVC_SUBUNIT_TYPE_AUDIO macro 459
AVC_SUBUNIT_TYPE_BULLETIN_BOARD macro 459
AVC_SUBUNIT_TYPE_CA macro 460
AVC_SUBUNIT_TYPE_CAMERA macro 460
AVC_SUBUNIT_TYPE_CAMERA_STORAGE macro 460
AVC_SUBUNIT_TYPE_DISC macro 460
AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE macro 460
AVC_SUBUNIT_TYPE_MONITOR macro 461
AVC_SUBUNIT_TYPE_PANEL macro 461
AVC_SUBUNIT_TYPE_PRINTER macro 461
AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER macro 461
AVC_SUBUNIT_TYPE_TUNER macro 461
AVC_SUBUNIT_TYPE_UNIT macro 462
AVC_SUBUNIT_TYPE_VENDOR_UNIQUE macro 462
AVC_VOLUME_MAX macro 462
AVC_VOLUME_MIN macro 462
avctp.h 1022
AVCTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR macro 22
AVCTP_ALLOCATE_BUFFERS_VARS macro 22
AVCTP_CHANNEL_FLAG_LISTENING macro 285
AVCTP_CHANNEL_FLAG_SENDING macro 285
AVCTP_CHANNEL_STATE_CONNECTED macro 286
AVCTP_CHANNEL_STATE_CONNECTING macro 286
AVCTP_CHANNEL_STATE_DISCONNECTING macro 286
AVCTP_CHANNEL_STATE_FREE macro 286
AVCTP_CHANNEL_STATE_IDLE macro 286
avctp_config.h 1026
AVCTP_ERROR_BAD_STATE macro 287
AVCTP_ERROR_SUCCESS macro 287
AVCTP_EVT_CHANNEL_CONNECTED macro 287

AVCTP_EVT_CHANNEL_DISCONNECTED macro 287
AVCTP_EVT_COMMAND_CANCELLED macro 287
AVCTP_EVT_COMMAND_RECEIVED macro 288
AVCTP_EVT_COMMAND_SENT macro 288
AVCTP_EVT_CONNECTION_FAILED macro 288
AVCTP_EVT_NOTHING macro 288
AVCTP_EVT_RESPONE_RECEIVED macro 288
AVCTP_EVT_RESPONSE_CANCELLED macro 289
AVCTP_EVT_RESPONSE_SENT macro 289
AVCTP_MANAGER_STATE_IDLE macro 289
AVCTP_MAX_CHANNELS macro 23
AVCTP_MAX_TRANSPORT_CHANNELS macro 23
AVCTP_MESSAGE_PACKET_TYPE_CONTINUE macro 289
AVCTP_MESSAGE_PACKET_TYPE_END macro 289
AVCTP_MESSAGE_PACKET_TYPE_SINGLE macro 290
AVCTP_MESSAGE_PACKET_TYPE_START macro 290
AVCTP_MESSAGE_TYPE_COMMAND macro 290
AVCTP_MESSAGE_TYPE_RESPONSE macro 290
avctp_packet.h 1027
avctp_private.h 1028
AVCTP_TRANSPORT_FLAG_RX_MESSAGE_STARTED macro 290
AVCTP_TRANSPORT_FLAG_SENDING macro 291
AVCTP_TRANSPORT_STATE_CONNECTED macro 291
AVCTP_TRANSPORT_STATE_CONNECTING macro 291
AVCTP_TRANSPORT_STATE_DISCONNECTING macro 291
AVCTP_TRANSPORT_STATE_FREE macro 291
AVCTP_TRANSPORT_STATE_IDLE macro 292
avdtp.h 1028
AVDTP_ALLOCATE_BUFFERS_FUNCTION macro 23
AVDTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR macro 23
AVDTP_ALLOCATE_BUFFERS_VARS macro 24
AVDTP_CMD_ABORT macro 25
AVDTP_CMD_CLOSE macro 25
AVDTP_CMD_DELAYREPORT macro 364
AVDTP_CMD_DISCOVER macro 25
AVDTP_CMD_GET_ALL_CAPABILITIES macro 365
AVDTP_CMD_GET_CAPABILITIES macro 25
AVDTP_CMD_GET_CONFIGURATION macro 25
AVDTP_CMD_OPEN macro 26
AVDTP_CMD_RECONFIGURE macro 26
AVDTP_CMD_SECURITY_CONTROL macro 26
AVDTP_CMD_SET_CONFIGURATION macro 26
AVDTP_CMD_START macro 26
AVDTP_CMD_SUSPEND macro 27
AVDTP_CODEC_CONFIG_BUFFER_LEN macro 27
AVDTP_CODEC_OPCODE_PARSE_CONFIG macro 339
AVDTP_CODEC_OPCODE_PARSE_PACKET macro 339
AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG macro 339
AVDTP_CODEC_TYPE_ATRAC macro 340
AVDTP_CODEC_TYPE_MPEG1_2_AUDIO macro 340
AVDTP_CODEC_TYPE_MPEG2_4_AAC macro 340
AVDTP_CODEC_TYPE_NON_A2DP macro 340
AVDTP_CODEC_TYPE_SBC macro 340
avdtp_config.h 1046
AVDTP_CONTENT_PROTECTION_METHOD_SCMS_T macro 365
avdtp_control.h 1048
AVDTP_CTLR_CHANNEL_EVT_CONNECTED macro 341
AVDTP_CTLR_CHANNEL_EVT_CONNECTION_FAILED macro 341
AVDTP_CTLR_CHANNEL_EVT_DISCONNECTED macro 341
AVDTP_CTLR_CHANNEL_EVT_NOTHING macro 341
AVDTP_CTRL_CHANNEL_EVT_DATA_RECEIVED macro 341
AVDTP_CTRL_CHANNEL_STATE_CONNECTED macro 342
AVDTP_CTRL_CHANNEL_STATE_CONNECTING macro 342
AVDTP_CTRL_CHANNEL_STATE_DISCONNECTED macro 342
AVDTP_CTRL_MESSAGE_TYPE_ACCEPT macro 342
AVDTP_CTRL_MESSAGE_TYPE_COMMAND macro 342
AVDTP_CTRL_MESSAGE_TYPE_FLD macro 343
AVDTP_CTRL_MESSAGE_TYPE_GENERAL_REJECT macro 365
AVDTP_CTRL_MESSAGE_TYPE_REJECT macro 343
AVDTP_CTRL_PACKET_TYPE_CONTINUE macro 343
AVDTP_CTRL_PACKET_TYPE_END macro 343
AVDTP_CTRL_PACKET_TYPE_FLD macro 343
AVDTP_CTRL_PACKET_TYPE_SIGNLE macro 344
AVDTP_CTRL_PACKET_TYPE_START macro 344
AVDTP_ERROR_BAD_ACP_SEID macro 344
AVDTP_ERROR_BAD_CP_FORMAT macro 344
AVDTP_ERROR_BAD_HEADER_FORMAT macro 344
AVDTP_ERROR_BAD_LENGTH macro 345
AVDTP_ERROR_BAD_MEDIA_TRANSPORT_FORMAT macro 345
AVDTP_ERROR_BAD_MULTIPLEXING_FORMAT macro 345
AVDTP_ERROR_BAD_PAYLOAD_FORMAT macro 345
AVDTP_ERROR_BAD_RECOVERY_FORMAT macro 345
AVDTP_ERROR_BAD_RECOVERY_TYPE macro 346
AVDTP_ERROR_BAD_ROHC_FORMAT macro 346
AVDTP_ERROR_BAD_SERV_CATEGORY macro 346
AVDTP_ERROR_BAD_STATE macro 346
AVDTP_ERROR_FAILED_TO_CONNECT_CONTROL macro 346
AVDTP_ERROR_FAILED_TO_CONNECT_TRANSPORT macro 347
AVDTP_ERROR_INVALID_CAPABILITIES macro 347
AVDTP_ERROR_NOT_SUPPORTED_COMMAND macro 347
AVDTP_ERROR_SEP_IN_USE macro 347
AVDTP_ERROR_SEP_NOT_IN_USE macro 347
AVDTP_ERROR_SUCCESS macro 348
AVDTP_ERROR_UNSUPPORTED_CONFIGURAION macro 348
AVDTP_EVT_ABORT_STREAM_COMPLETED macro 348
AVDTP_EVT_ABORT_STREAM_REQUESTED macro 348
AVDTP_EVT_CLOSE_STREAM_COMPLETED macro 348
AVDTP_EVT_CLOSE_STREAM_REQUESTED macro 349
AVDTP_EVT_CTRL_CHANNEL_CONNECTED macro 349
AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED macro 349
AVDTP_EVT_CTRL_CONNECTION_FAILED macro 349
AVDTP_EVT_DELAYREPORT_COMPLETED macro 366
AVDTP_EVT_DISCOVER_COMPLETED macro 349
AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED macro 350
AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED macro 350
AVDTP_EVT_LAST macro 350
AVDTP_EVT_MEDIA_PACKET_RECEIVED macro 350
AVDTP_EVT_MEDIA_PACKET_SEND_FAILED macro 350
AVDTP_EVT_MEDIA_PACKET_SENT macro 351
AVDTP_EVT_NULL macro 351
AVDTP_EVT_OPEN_STREAM_COMPLETED macro 351
AVDTP_EVT_OPEN_STREAM_REQUESTED macro 351
AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED macro 351
AVDTP_EVT_SEP_CAPABILITIES_RECEIVED macro 352
AVDTP_EVT_SEP_INFO_RECEIVED macro 352

AVDTP_EVT_SET_STREAM_CONFIGURATION macro 367
AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED macro 352
AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED macro 352
AVDTP_EVT_START_STREAM_COMPLETED macro 352
AVDTP_EVT_START_STREAM_REQUESTED macro 353
AVDTP_EVT_STREAM_ABORTED macro 353
AVDTP_EVT_STREAM_CLOSED macro 353
AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED macro 353
AVDTP_EVT_STREAM_CONFIGURED macro 353
AVDTP_EVT_STREAM_OPENED macro 354
AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED macro 354
AVDTP_EVT_STREAM_RECONFIGURED macro 354
AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED macro 354
AVDTP_EVT_STREAM_STARTED macro 355
AVDTP_EVT_STREAM_SUSPENDED macro 355
AVDTP_EVT_SUSPEND_STREAM_COMPLETED macro 355
AVDTP_EVT_SUSPEND_STREAM_REQUESTED macro 355
AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET macro 292
AVDTP_MANAGER_STATE_CONNECTING macro 355
AVDTP_MANAGER_STATE_IDLE macro 356
AVDTP_MAX_CMD_BUFFERS macro 27
AVDTP_MAX_CMD_PARAM_LEN macro 27
AVDTP_MAX_REMOTE_DEVICES macro 27
AVDTP_MAX_SEP macro 28
AVDTP_MAX_STREAM_TRANSPORT_SESSION macro 356
AVDTP_MAX_STREAMS macro 28
AVDTP_MAX_TRANSPORT_CHANNELS macro 28
AVDTP_MAX_TX_BUFFER_LEN macro 29
AVDTP_MEDIA_TYPE_AUDIO macro 356
AVDTP_MEDIA_TYPE_MULTIMEDIA macro 356
AVDTP_MEDIA_TYPE_VIDEO macro 356
avdtp_private.h 1049
AVDTP_SCMS_T_CP_BIT macro 365
AVDTP_SCMS_T_L_BIT macro 365
AVDTP_SEP_CAPABILITY_FLAG_CONTENT_PROTECTION macro 357
AVDTP_SEP_CAPABILITY_FLAG_DELAY_REPORTING macro 367
AVDTP_SEP_CAPABILITY_FLAG_HEADER_COMPRESSION macro 357
AVDTP_SEP_CAPABILITY_FLAG_MEDIA_CODEC macro 357
AVDTP_SEP_CAPABILITY_FLAG_MEDIA_TRANSPORT macro 357
AVDTP_SEP_CAPABILITY_FLAG_MULTIPLEXING macro 357
AVDTP_SEP_CAPABILITY_FLAG_RECOVERY macro 358
AVDTP_SEP_CAPABILITY_FLAG_REPORTING macro 358
AVDTP_SEP_SERVICE_CAPABILITY_CONTENT_PROTECTION macro 358
AVDTP_SEP_SERVICE_CAPABILITY_DELAY_REPORTING macro 367
AVDTP_SEP_SERVICE_CAPABILITY_HEADER_COMPRESSION macro 358
AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_CODEC macro 358
AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_TRANSPORT macro 359
AVDTP_SEP_SERVICE_CAPABILITY_MULTIPLEXING macro 359
AVDTP_SEP_SERVICE_CAPABILITY_RECOVERY macro 359
AVDTP_SEP_SERVICE_CAPABILITY_REPORTING macro 359
AVDTP_SEP_STATE_FREE macro 359
AVDTP_SEP_STATE_IDLE macro 360
AVDTP_SEP_TYPE_SINK macro 360
AVDTP_SEP_TYPE_SOURCE macro 360
AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS macro 360
AVDTP_STREAM_FLAG_LISTENING macro 360
AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS macro 361
AVDTP_STREAM_STATE_ABORTING macro 361
AVDTP_STREAM_STATE_CLOSING macro 361
AVDTP_STREAM_STATE_CONFIGURED macro 361
AVDTP_STREAM_STATE_IDLE macro 361
AVDTP_STREAM_STATE_OPEN macro 362
AVDTP_STREAM_STATE_STREAMING macro 362
AVDTP_TC_OPCODE_CONNECT macro 362
AVDTP_TC_OPCODE_DISCONNECT macro 362
AVDTP_TC_STATUS_ERROR macro 362
AVDTP_TC_STATUS_SUCCESS macro 363
AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED macro 363
AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED macro 363
AVDTP_TRANSPORT_SESSION_TYPE_MEDIA macro 363
AVDTP_TRANSPORT_SESSION_TYPE_RECOVERY macro 363
AVDTP_TRANSPORT_SESSION_TYPE_REPORTING macro 364
avrcp.h 1050
AVRCP_ALLOCATE_BUFFERS_RAM_SIZE_VAR macro 29
AVRCP_ALLOCATE_BUFFERS_VARS macro 30
AVRCP_BTSIG_COMPANY_ID macro 462
AVRCP_CHANNEL_FLAG_LISTENING macro 463
AVRCP_CHANNEL_FLAG_PLAY_STATUS_REQUESTED macro 463
AVRCP_CHANNEL_FLAG_REGISTERING_NOTIFICATIONS macro 463
AVRCP_CHANNEL_FLAG_SENDING macro 463
AVRCP_CHANNEL_STATE_CONNECTED macro 463
AVRCP_CHANNEL_STATE_CONNECTING macro 464
AVRCP_CHANNEL_STATE_DISCONNECTING macro 464
AVRCP_CHANNEL_STATE_FREE macro 464
AVRCP_CHANNEL_STATE_IDLE macro 464
AVRCP_CMD_TIMEOUT macro 30
avrcp_command.h 1061
AVRCP_COMMAND_HANDLER macro 511
AVRCP_COMMAND_SENT_HANDLER macro 511
AVRCP_COMMAND_TYPE_BROWSING macro 464
AVRCP_COMMAND_TYPE_CONTROL macro 465
avrcp_config.h 1077
avrcp_config_event_handlers.h 1078
AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED macro 465
AVRCP_EVT_ADDRESSED_PLAYER_CHANGED macro 465
AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED macro 465
AVRCP_EVT_BATT_STATUS_CHANGED macro 465
AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED macro 466
AVRCP_EVT_BROWSING_CHANNEL_CONNECTED macro 466
AVRCP_EVT_BROWSING_CHANNEL_DISCONNECTED macro 466
AVRCP_EVT_BROWSING_CONNECTION_FAILED macro 466
AVRCP_EVT_COMPANY_ID_LIST_RECEIVED macro 466
AVRCP_EVT_CONTROL_CHANNEL_CONNECTED macro 467
AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED macro 467
AVRCP_EVT_CONTROL_CONNECTION_FAILED macro 467
AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED macro 467
AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED macro 467
AVRCP_EVT_EVENT_ID_LIST_RECEIVED macro 468

AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED macro 468
AVRCP_EVT_GET_PLAY_STATUS_RECEIVED macro 468
AVRCP_EVT_INFORM_BATTERY_STATUS_OF_CT_COMPLETED macro 468
AVRCP_EVT_INFORM_DISPLAYABLE_CHARACTER_SET_COMPLETED macro 468
AVRCP_EVT_NOTHING macro 469
AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED macro 469
AVRCP_EVT_PANEL_COMMAND_RECEIVED macro 469
AVRCP_EVT_PANEL_RESPONSE_RECEIVED macro 469
AVRCP_EVT_PLAY_ITEM_COMPLETED macro 469
AVRCP_EVT_PLAYBACK_POS_CHANGED macro 470
AVRCP_EVT_PLAYBACK_STATUS_CHANGED macro 470
AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED macro 470
AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED macro 470
AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED macro 470
AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED macro 471
AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED macro 471
AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED macro 471
AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED macro 471
AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED macro 471
AVRCP_EVT_SEARCH_COMPLETED macro 472
AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED macro 472
AVRCP_EVT_SET_ABSOLUTE_VOLUME_REQUESTED macro 472
AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED macro 472
AVRCP_EVT_SET_PLAYER_SETTING_VALUES_COMPLETED macro 472
AVRCP_EVT_SYSTEM_STATUS_CHANGED macro 473
AVRCP_EVT_TRACK_CHANGED macro 473
AVRCP_EVT_TRACK_REACHED_END macro 473
AVRCP_EVT_TRACK_REACHED_START macro 473
AVRCP_EVT_UIDS_CHANGED macro 473
AVRCP_EVT_VOLUME_CHANGED macro 474
AVRCP_MANAGER_FLAG_SEARCHING macro 474
AVRCP_MANAGER_STATE_IDLE macro 474
AVRCP_MAX_CHANNELS macro 30
AVRCP_MAX_CMD_BUFFERS macro 31
AVRCP_MAX_CMD_PARAM_LEN macro 31
AVRCP_MAX_DEVICE_NAME_LEN macro 31
AVRCP_MAX_ELEMENT_ATTRIBUTES macro 474
AVRCP_MAX_SEARCH_RESULTS macro 31
avrcp_private.h 1079
AVRCP_RESPONSE_HANDLER macro 511
AVRCP_RESPONSE_SENT_HANDLER macro 511

B

b) Data Types and Constants 6761
Background Functions 2629
baseband.h 1081
Basic Flow of the Bootloader 1175
BEGIN_DE_SEQUENCE macro 841
Berkeley Changes 5990
Berkeley Module 6029
berkeley_api.h 6062
berkeley_api_config.h 6064
BERKELEY_MODULE_CONFIG structure 6061
bind function 6034
biquad16 structure 3084
BITMASKFRACT16 macro 3109
BITMASKFRACT32 macro 3109
BLACK macro 2651
Block Operations 1969
BLUE macro 2651
Bluetooth Stack Library Help 4
Bluetooth Stack Licensing 4
BMX Peripheral Library 3481
BMX_MODULE_ID enumeration 3511
BootInfo variable 1188
Bootloader Communication Protocol (UART, USB HID, and Ethernet) 1177
Bootloader Configurations 1180
Bootloader Library Help 1173
Bootloader Placement in Memory 1175
Bootloader Sizing and Optimization 1182
bootloader.h 1189
BOOTLOADER_BUFFER union 1188
BOOTLOADER_CALLBACK type 1188
BOOTLOADER_CLIENT_STATUS enumeration 1187
BOOTLOADER_DATA_CALLBACK type 1189
BOOTLOADER_INIT structure 1187
Bootloader_Initialize function 1185
Bootloader_Tasks function 1186
BOOTLOADER_TYPE enumeration 1187
BOS Descriptor Support 6790
BRIGHTBLUE macro 2651
BRIGHTCYAN macro 2651
BRIGHTGREEN macro 2652
BRIGHTMAGENTA macro 2652
BRIGHTRED macro 2652
BRIGHTYELLOW macro 2652
BROWN macro 2652
bt_a2dp_aac_codec_handler function 222
bt_a2dp_aac_config_t structure 228
bt_a2dp_abort_stream macro 247
bt_a2dp_add_media_rx_buffer macro 247
bt_a2dp_add_media_tx_buffer macro 248
bt_a2dp_call_codec macro 248
bt_a2dp_cancel_listen macro 248
bt_a2dp_clear_media_tx_queue macro 268
bt_a2dp_close_stream macro 248
bt_a2dp_connect macro 249
bt_a2dp_connect_ex macro 249
bt_a2dp_create_stream macro 249
bt_a2dp_destroy_stream macro 250
bt_a2dp_disconnect macro 250
bt_a2dp_discover macro 250
bt_a2dp_event_t type 229
bt_a2dp_evt_open_and_start_stream_completed_t structure 229
bt_a2dp_find_codec macro 251
bt_a2dp_find_server function 222
bt_a2dp_find_server_callback_fp type 229
bt_a2dp_find_sink function 222

bt_a2dp_find_source function 223
bt_a2dp_get_all_capabilities macro 268
bt_a2dp_get_capabilities macro 251
bt_a2dp_get_configuration macro 268
bt_a2dp_get_hci_connection macro 251
bt_a2dp_get_mgr function 223
bt_a2dp_get_stream_codec_config macro 252
bt_a2dp_get_stream_codec_type macro 252
bt_a2dp_get_stream_config macro 269
bt_a2dp_get_stream_direction macro 269
bt_a2dp_get_stream_local_sep_id macro 252
bt_a2dp_get_stream_remote_address macro 253
bt_a2dp_get_stream_remote_sep_id macro 253
bt_a2dp_get_stream_state macro 253
bt_a2dp_init function 223
bt_a2dp_listen macro 253
bt_a2dp_mgr_callback_fp type 230
bt_a2dp_mgr_t type 231
bt_a2dp_mpeg_codec_handler function 223
bt_a2dp_mpeg_config_t structure 231
bt_a2dp_open_and_start_stream function 224
bt_a2dp_open_stream macro 268
bt_a2dp_reconfigure_stream macro 254
bt_a2dp_register_aac_codec function 224
bt_a2dp_register_callback function 224
bt_a2dp_register_mpeg_codec function 225
bt_a2dp_register_sink macro 254
bt_a2dp_register_source macro 254
bt_a2dp_remove_media_rx_buffer macro 255
bt_a2dp_remove_media_tx_buffer macro 255
bt_a2dp_report_delay macro 269
bt_a2dp_sbc_codec_handler function 225
bt_a2dp_sbc_config_t structure 231
bt_a2dp_sbc_packet_info_t structure 232
bt_a2dp_set_configuration function 226
bt_a2dp_set_media_tx_queue_limit macro 269
bt_a2dp_start function 226
bt_a2dp_start_stream macro 255
bt_a2dp_suspend_stream macro 256
bt_alloc_buffer function 966
BT_ASSERT macro 204
bt_at_parse_fragment function 966
bt_at_parser_callback_pf type 994
bt_at_parser_reset function 966
bt_at_parser_t structure 994
bt_av_add_to_now_playing_t structure 395
bt_av_battery_status_of_ct_t structure 395
bt_av_capability_company_id_t structure 396
bt_av_capability_event_id_t structure 396
bt_av_command_t type 397
bt_av_displayable_character_set_t structure 397
bt_av_element_attribute_t structure 397
bt_av_element_attributes_t structure 398
bt_av_element_id_t structure 398
bt_av_get_element_attributes_t structure 398
bt_av_notification_addressed_player_changed_t structure 399
bt_av_notification_app_setting_changed_t structure 399
bt_av_notification_battery_status_t structure 399
bt_av_notification_playback_pos_changed_t structure 400
bt_av_notification_playback_status_changed_t structure 400
bt_av_notification_system_status_changed_t structure 400
bt_av_notification_t structure 401
bt_av_notification_track_changed_t structure 402
bt_av_notification_uids_changed_t structure 402
bt_av_notification_volume_changed_t structure 402
bt_av_play_item_t structure 403
bt_av_play_status_t structure 403
bt_av_player_setting_current_values_t structure 403
bt_av_player_setting_values_t structure 404
bt_av_player_setting_values_text_t structure 404
bt_av_player_settings_t structure 405
bt_av_player_settings_text_t structure 405
bt_av_player_text_t structure 406
bt_av_register_notification_t structure 406
bt_av_response_t structure 406
bt_av_set_absolute_volume_t structure 407
bt_av_set_addressed_player_t structure 407
bt_avctp_cancel_command function 273
bt_avctp_cancel_listen function 273
bt_avctp_cancel_response function 273
bt_avctp_channel_t type 279
bt_avctp_connect function 274
bt_avctp_create_channel function 274
bt_avctp_create_outgoing_channel function 274
bt_avctp_destroy_channel function 274
bt_avctp_disconnect function 275
bt_avctp_event_t union 279
bt_avctp_evt_channel_connected_t structure 280
bt_avctp_evt_channel_disconnected_t structure 280
bt_avctp_evt_command_cancelled_t structure 281
bt_avctp_evt_command_received_t structure 281
bt_avctp_evt_command_sent_t structure 281
bt_avctp_evt_connection_failed_t structure 282
bt_avctp_evt_response_cancelled_t structure 282
bt_avctp_evt_response_received_t structure 282
bt_avctp_evt_response_sent_t structure 283
bt_avctp_get_channel_remote_address function 275
bt_avctp_get_channel_state function 275
bt_avctp_get_hci_connection function 275
bt_avctp_get_mgr function 276
bt_avctp_init function 276
bt_avctp_listen function 276
bt_avctp_message_t type 283
bt_avctp_mgr_callback_fp type 283
bt_avctp_mgr_t type 284
bt_avctp_packet_t structure 284
bt_avctp_send_command function 276
bt_avctp_send_response function 277
bt_avctp_set_callback function 277
bt_avctp_start function 277
bt_avctp_transport_t type 284
bt_avdtp_abort_stream function 300
bt_avdtp_add_media_rx_buffer function 300
bt_avdtp_add_media_tx_buffer function 300
bt_avdtp_cancel_listen function 301
bt_avdtp_clear_media_tx_queue function 312

bt_avdtp_close_stream function 301
bt_avdtp_codec_handler_fp type 315
bt_avdtp_codec_op_decode_t structure 316
bt_avdtp_codec_op_encode_t structure 316
bt_avdtp_codec_op_param_t type 316
bt_avdtp_codec_op_parse_config_t structure 316
bt_avdtp_codec_op_parse_packet_t structure 317
bt_avdtp_codec_op_serialize_config_t structure 317
bt_avdtp_codec_t structure 318
bt_avdtp_connect macro 364
bt_avdtp_connect_ex macro 364
bt_avdtp_control_channel_t structure 318
bt_avdtp_control_cmd_t structure 318
bt_avdtp_create_stream function 301
bt_avdtp_ctrl_evt_data_received_t structure 319
bt_avdtp_destroy_stream function 301
bt_avdtp_disconnect function 302
bt_avdtp_discover function 302
bt_avdtp_event_t union 319
bt_avdtp_evt_abort_stream_requested_t structure 321
bt_avdtp_evt_close_stream_completed_t structure 321
bt_avdtp_evt_close_stream_requested_t structure 321
bt_avdtp_evt_ctrl_channel_connected_t structure 322
bt_avdtp_evt_ctrl_channel_disconnected_t structure 322
bt_avdtp_evt_ctrl_connection_failed_t structure 367
bt_avdtp_evt_delay_report_completed_t structure 366
bt_avdtp_evt_discover_completed_t structure 323
bt_avdtp_evt_get_sep_capabilities_completed_t structure 323
bt_avdtp_evt_get_stream_configuration_completed_t structure 323
bt_avdtp_evt_media_packet_received_t structure 324
bt_avdtp_evt_media_packet_send_failed_t structure 324
bt_avdtp_evt_media_packet_sent_t structure 324
bt_avdtp_evt_open_stream_completed_t structure 325
bt_avdtp_evt_open_stream_requested_t structure 325
bt_avdtp_evt_reconfigure_stream_requested_t structure 326
bt_avdtp_evt_sep_capabilities_received_t structure 326
bt_avdtp_evt_sep_info_received_t structure 326
bt_avdtp_evt_set_stream_configuration_completed_t structure 327
bt_avdtp_evt_set_stream_configuration_requested_t structure 327
bt_avdtp_evt_set_stream_configuration_t structure 368
bt_avdtp_evt_start_stream_completed_t structure 328
bt_avdtp_evt_start_stream_requested_t structure 328
bt_avdtp_evt_stream_aborted_t structure 329
bt_avdtp_evt_stream_closed_t structure 329
bt_avdtp_evt_stream_configured_t structure 329
bt_avdtp_evt_stream_opened_t structure 330
bt_avdtp_evt_stream_reconfigure_completed_t structure 330
bt_avdtp_evt_stream_reconfigured_t structure 330
bt_avdtp_evt_stream_security_control_completed_t structure 331
bt_avdtp_evt_stream_started_t structure 331
bt_avdtp_evt_stream_suspended_t structure 331
bt_avdtp_evt_suspend_stream_completed_t structure 332
bt_avdtp_evt_suspend_stream_requested_t structure 332
bt_avdtp_find_codec function 302
bt_avdtp_get_all_capabilities function 312
bt_avdtp_get_capabilities function 303
bt_avdtp_get_configuration function 303
bt_avdtp_get_hci_connection function 303
bt_avdtp_get_l2cap_channel function 312
bt_avdtp_get_mgr function 304
bt_avdtp_get_sep function 304
bt_avdtp_get_stream_codec_config function 304
bt_avdtp_get_stream_codec_type function 304
bt_avdtp_get_stream_config function 305
bt_avdtp_get_stream_direction function 313
bt_avdtp_get_stream_local_sep_id function 305
bt_avdtp_get_stream_remote_address function 305
bt_avdtp_get_stream_remote_sep_id function 306
bt_avdtp_get_stream_state function 306
bt_avdtp_init function 306
bt_avdtp_listen function 306
bt_avdtp_mgr_callback_fp type 332
bt_avdtp_mgr_t structure 334
bt_avdtp_open_stream function 307
bt_avdtp_reconfigure_stream function 307
bt_avdtp_register_callback function 307
bt_avdtp_register_codec function 308
bt_avdtp_register_sep function 308
bt_avdtp_remove_media_rx_buffer function 309
bt_avdtp_remove_media_tx_buffer function 309
bt_avdtp_report_delay function 313
bt_avdtp_security_control function 309
bt_avdtp_sep_capabilities_t structure 335
bt_avdtp_sep_t type 336
bt_avdtp_set_configuration function 310
bt_avdtp_set_media_tx_queue_limit function 313
bt_avdtp_start function 310
bt_avdtp_start_stream function 310
bt_avdtp_stream_t structure 336
bt_avdtp_suspend_stream function 311
bt_avdtp_transport_channel_t structure 337
bt_avdtp_transport_op_callback_fp type 337
bt_avdtp_transport_session_t structure 338
bt_avdtp_unregister_codec function 311
bt_avrcp_0_click macro 474
bt_avrcp_0_press macro 475
bt_avrcp_0_release macro 498
bt_avrcp_1_click macro 475
bt_avrcp_1_press macro 475
bt_avrcp_1_release macro 499
bt_avrcp_2_click macro 475
bt_avrcp_2_press macro 475
bt_avrcp_2_release macro 499
bt_avrcp_3_click macro 476
bt_avrcp_3_press macro 476
bt_avrcp_3_release macro 499
bt_avrcp_4_click macro 476
bt_avrcp_4_press macro 476
bt_avrcp_4_release macro 499
bt_avrcp_5_click macro 476
bt_avrcp_5_press macro 477
bt_avrcp_5_release macro 499
bt_avrcp_6_click macro 477
bt_avrcp_6_press macro 477
bt_avrcp_6_release macro 500
bt_avrcp_7_click macro 477

bt_avrcp_7_press macro 477
bt_avrcp_7_release macro 500
bt_avrcp_8_click macro 478
bt_avrcp_8_press macro 478
bt_avrcp_8_release macro 500
bt_avrcp_9_click macro 478
bt_avrcp_9_press macro 478
bt_avrcp_9_release macro 500
bt_avrcp_abort_continuing_response function 378
bt_avrcp_add_to_now_playing function 378
bt_avrcp_angle_click macro 478
bt_avrcp_angle_press macro 479
bt_avrcp_angle_release macro 500
bt_avrcp_avrcpt_request_continuing_response function 378
bt_avrcp_backward_click macro 479
bt_avrcp_backward_press macro 479
bt_avrcp_backward_release macro 501
bt_avrcp_cancel_find function 378
bt_avrcp_cancel_listen function 378
bt_avrcp_channel_down_click macro 479
bt_avrcp_channel_down_press macro 479
bt_avrcp_channel_down_release macro 501
bt_avrcp_channel_t type 407
bt_avrcp_channel_up_click macro 480
bt_avrcp_channel_up_press macro 480
bt_avrcp_channel_up_release macro 501
bt_avrcp_clear_click macro 480
bt_avrcp_clear_press macro 480
bt_avrcp_clear_release macro 501
bt_avrcp_connect function 379
bt_avrcp_content_menu_click macro 480
bt_avrcp_content_menu_release macro 501
bt_avrcp_contents_menu_press macro 481
bt_avrcp_create_channel function 379
bt_avrcp_create_outgoing_channel function 379
bt_avrcp_destroy_channel function 380
bt_avrcp_device_t structure 408
bt_avrcp_disconnect function 380
bt_avrcp_display_info_click macro 481
bt_avrcp_display_info_press macro 481
bt_avrcp_display_info_release macro 502
bt_avrcp_dot_click macro 481
bt_avrcp_dot_press macro 481
bt_avrcp_dot_release macro 502
bt_avrcp_down_click macro 482
bt_avrcp_down_press macro 482
bt_avrcp_down_release macro 502
bt_avrcp_eject_click macro 482
bt_avrcp_eject_press macro 482
bt_avrcp_eject_release macro 502
bt_avrcp_enter_click macro 482
bt_avrcp_enter_press macro 483
bt_avrcp_enter_release macro 502
bt_avrcp_event_t union 408
bt_avrcp_evt_channel_connected_t structure 409
bt_avrcp_evt_channel_disconnected_t structure 410
bt_avrcp_evt_connection_failed_t structure 410
bt_avrcp_evt_panel_command_received_t structure 410
bt_avrcp_evt_panel_response_received_t structure 411
bt_avrcp_evt_register_events_completed_t structure 411
bt_avrcp_evt_search_completed_t structure 412
bt_avrcp_exit_click macro 483
bt_avrcp_exit_press macro 483
bt_avrcp_exit_release macro 503
bt_avrcp_f1_click macro 483
bt_avrcp_f1_press macro 483
bt_avrcp_f1_release macro 503
bt_avrcp_f2_click macro 484
bt_avrcp_f2_press macro 484
bt_avrcp_f2_release macro 503
bt_avrcp_f3_click macro 484
bt_avrcp_f3_press macro 484
bt_avrcp_f3_release macro 503
bt_avrcp_f4_click macro 484
bt_avrcp_f4_press macro 485
bt_avrcp_f4_release macro 503
bt_avrcp_f5_click macro 485
bt_avrcp_f5_press macro 485
bt_avrcp_f5_release macro 504
bt_avrcp_f6_click macro 485
bt_avrcp_f6_press macro 485
bt_avrcp_f6_release macro 504
bt_avrcp_f7_click macro 486
bt_avrcp_f7_press macro 486
bt_avrcp_f7_release macro 504
bt_avrcp_f8_click macro 486
bt_avrcp_f8_press macro 486
bt_avrcp_f8_release macro 504
bt_avrcp_f9_click macro 486
bt_avrcp_f9_press macro 487
bt_avrcp_f9_release macro 504
bt_avrcp_fast_forward_click macro 487
bt_avrcp_fast_forward_press macro 487
bt_avrcp_fast_forward_release macro 505
bt_avrcp_favorite_menu_click macro 487
bt_avrcp_favorite_menu_release macro 505
bt_avrcp_favorive_menu_press macro 487
bt_avrcp_find_callback_fp type 412
bt_avrcp_find_controller function 380
bt_avrcp_find_target function 380
bt_avrcp_find_targets function 381
bt_avrcp_forward_click macro 488
bt_avrcp_forward_press macro 488
bt_avrcp_forward_release macro 505
bt_avrcp_get_browsing_channel_state function 381
bt_avrcp_get_channel_remote_address function 381
bt_avrcp_get_company_id_list function 382
bt_avrcp_get_control_channel_state function 382
bt_avrcp_get_current_player_application_setting_value function 382
bt_avrcp_get_element_attributes function 382
bt_avrcp_get_hci_connection function 383
bt_avrcp_get_mgr function 383
bt_avrcp_get_play_status function 383
bt_avrcp_get_player_application_setting_attr_text function 383
bt_avrcp_get_player_application_setting_value_text function 384
bt_avrcp_get_subuint_info function 384

bt_avrcp_get_supported_event_id_list function 384
bt_avrcp_get_unit_info function 385
bt_avrcp_help_click macro 488
bt_avrcp_help_press macro 488
bt_avrcp_help_release macro 505
bt_avrcp_inform_battery_status function 385
bt_avrcp_inform_displayable_character_set function 385
bt_avrcp_init_controller function 385
bt_avrcp_init_target function 386
bt_avrcp_input_select_click macro 488
bt_avrcp_input_select_press macro 489
bt_avrcp_input_select_release macro 505
bt_avrcp_left_click macro 489
bt_avrcp_left_down_click macro 489
bt_avrcp_left_down_press macro 489
bt_avrcp_left_down_release macro 506
bt_avrcp_left_press macro 489
bt_avrcp_left_release macro 506
bt_avrcp_left_up_click macro 490
bt_avrcp_left_up_press macro 490
bt_avrcp_left_up_release macro 506
bt_avrcp_list_player_application_setting_attributes function 386
bt_avrcp_list_player_application_setting_values function 386
bt_avrcp_listen function 387
bt_avrcp_mgr_callback_fp type 412
bt_avrcp_mgr_t type 414
bt_avrcp_mute_click macro 490
bt_avrcp_mute_press macro 490
bt_avrcp_mute_release macro 506
bt_avrcp_page_down_click macro 490
bt_avrcp_page_down_press macro 491
bt_avrcp_page_down_release macro 506
bt_avrcp_page_up_click macro 491
bt_avrcp_page_up_press macro 491
bt_avrcp_page_up_release macro 507
bt_avrcp_pause_click macro 491
bt_avrcp_pause_press macro 491
bt_avrcp_pause_release macro 507
bt_avrcp_play_click macro 492
bt_avrcp_play_item function 387
bt_avrcp_play_press macro 492
bt_avrcp_play_release macro 507
bt_avrcp_power_click macro 492
bt_avrcp_power_press macro 492
bt_avrcp_power_release macro 507
bt_avrcp_previous_channel_click macro 492
bt_avrcp_previous_channel_press macro 493
bt_avrcp_previous_channel_release macro 507
bt_avrcp_record_click macro 493
bt_avrcp_record_press macro 493
bt_avrcp_record_release macro 508
bt_avrcp_register_notification function 387
bt_avrcp_register_notifications function 387
bt_avrcp_rewind_click macro 493
bt_avrcp_rewind_press macro 493
bt_avrcp_rewind_release macro 508
bt_avrcp_right_click macro 494
bt_avrcp_right_down_click macro 494
bt_avrcp_right_down_press macro 494
bt_avrcp_right_down_release macro 508
bt_avrcp_right_press macro 494
bt_avrcp_right_release macro 508
bt_avrcp_right_up_click macro 494
bt_avrcp_right_up_press macro 495
bt_avrcp_right_up_release macro 508
bt_avrcp_root_menu_click macro 495
bt_avrcp_root_menu_press macro 495
bt_avrcp_root_menu_release macro 509
bt_avrcp_select_click macro 495
bt_avrcp_select_press macro 495
bt_avrcp_select_release macro 509
bt_avrcp_send_button_click function 388
bt_avrcp_send_cmd function 388
bt_avrcp_send_panel_control function 388
bt_avrcp_send_press_panel_control function 389
bt_avrcp_send_release_panel_control function 393
bt_avrcp_send_simple_panel_cmd function 389
bt_avrcp_set_absolute_volume function 389
bt_avrcp_set_addressed_player function 389
bt_avrcp_set_player_application_setting_value function 390
bt_avrcp_setup_menu_click macro 496
bt_avrcp_setup_menu_press macro 496
bt_avrcp_setup_menu_release macro 509
bt_avrcp_sound_select_click macro 496
bt_avrcp_sound_select_press macro 496
bt_avrcp_sound_select_release macro 509
bt_avrcp_start function 390
bt_avrcp_stop_click macro 496
bt_avrcp_stop_press macro 497
bt_avrcp_stop_release macro 509
bt_avrcp_subpicture_click macro 497
bt_avrcp_subpicture_press macro 497
bt_avrcp_subpicture_release macro 510
bt_avrcp_tg_send_element_attributes function 390
bt_avrcp_tg_set_absolute_volume function 391
bt_avrcp_tg_set_battery_status function 391
bt_avrcp_tg_set_channel_absolute_volume function 391
bt_avrcp_tg_set_current_track function 391
bt_avrcp_tg_set_play_status function 392
bt_avrcp_tg_set_system_status function 392
bt_avrcp_up_click macro 497
bt_avrcp_up_press macro 497
bt_avrcp_up_release macro 510
bt_avrcp_volume_down_click macro 498
bt_avrcp_volume_down_press macro 498
bt_avrcp_volume_down_release macro 510
bt_avrcp_volume_up_click macro 498
bt_avrcp_volume_up_press macro 498
bt_avrcp_volume_up_release macro 510
bt_bdaddr.h 1081
bt_bdaddr_cp type 214
bt_bdaddr_is_null function 195
bt_bdaddr_p type 214
bt_bdaddr_t structure 215
bt_bdaddrs_are_equal function 195
bt_bool type 215

bt_bt_buffer_header_p structure 995
bt_buffer_header_t structure 995
bt_buffer_mgr_p structure 995
bt_buffer_mgr_t structure 995
bt_byte type 782
bt_byte_cp type 215
bt_byte_p type 215
bt_char type 215
bt_char_cp type 216
bt_char_p type 216
bt_config.h 1083
bt_cp_header_t type 366
bt_device_t structure 512
BT_ENABLE_BLE macro 19
BT_ENABLE_SCO macro 32
BT_FALSE macro 205
bt_find_devices_callback_fp type 512
bt_free_buffer function 967
bt_gap_find_devices function 511
bt_get_buffer function 967
bt_get_buffer_header function 967
bt_get_buffer_index function 967
bt_hci_add_param_bdaddr function 512
bt_hci_add_param_byte function 512
bt_hci_add_param_cod function 513
bt_hci_add_param_hconn macro 561
bt_hci_add_param_int function 513
bt_hci_add_param_lap macro 561
bt_hci_add_param_linkkey function 513
bt_hci_add_param_long function 513
bt_hci_add_param_string function 513
bt_hci_add_param_uint macro 562
bt_hci_add_param_ulong macro 562
bt_hci_alloc_canned_command function 514
bt_hci_alloc_command function 514
bt_hci_alloc_data_buffer function 514
bt_hci_allocate_write_eir_command function 514
bt_hci_authenticate macro 562
bt_hci_authenticate_ex function 514
bt_hci_cancel_find_devices function 515
bt_hci_cancel_find_devices_ex function 559
bt_hci_cancel_request_remote_name function 515
bt_hci_cancel_send_acl_data function 515
bt_hci_cmd_callback_fp type 675
bt_hci_cmd_listener_fp type 675
bt_hci_command_p structure 676
bt_hci_command_t structure 676
bt_hci_conn_state_p type 676
bt_hci_conn_state_t type 676
bt_hci_connect function 515
bt_hci_connect_callback_fp type 677
bt_hci_connect_sco function 515
bt_hci_ctrl_listener_t type 677
bt_hci_ctrl_register_data_listener function 516
bt_hci_ctrl_register_listener function 516
bt_hci_ctrl_state_t structure 677
bt_hci_ctrl_unregister_listener function 516
bt_hci_data_buffer_p structure 678
bt_hci_data_buffer_t structure 678
bt_hci_data_callback_fp type 678
bt_hci_data_listener_fp type 678
bt_hci_data_p structure 679
bt_hci_data_t structure 679
bt_hci_disconnect function 516
bt_hci_disconnect_callback_fp type 679
bt_hci_event_e union 680
bt_hci_event_handler_ex_fp type 680
bt_hci_event_handler_fp type 680
bt_hci_event_listener_fp type 680
bt_hci_event_p structure 680
bt_hci_event_t structure 681
bt_hci_evt_authentication_complete_handler function 516
BT_HCI_EVT_AUTHENTICATION_COMPLETE_HANDLER macro 32
bt_hci_evt_authentication_complete_t structure 681
bt_hci_evt_change_conn_link_complete_handler function 517
bt_hci_evt_command_complete_handler function 517
bt_hci_evt_command_complete_t structure 681
bt_hci_evt_command_status_handler function 517
bt_hci_evt_command_status_t structure 681
bt_hci_evt_conn_packet_type_changed_handler function 517
bt_hci_evt_connection_complete_handler function 517
BT_HCI_EVT_CONNECTION_COMPLETE_HANDLER macro 32
bt_hci_evt_connection_complete_t structure 682
bt_hci_evt_connection_request_handler function 518
BT_HCI_EVT_CONNECTION_REQUEST_HANDLER macro 32
bt_hci_evt_connection_request_t structure 682
bt_hci_evt_data_buffer_overflow_handler function 518
bt_hci_evt_default_handler function 518
bt_hci_evt_disconnection_complete_handler function 518
bt_hci_evt_disconnection_complete_t structure 682
bt_hci_evt_encryption_change_handler function 518
BT_HCI_EVT_ENCRYPTION_CHANGE_HANDLER macro 32
bt_hci_evt_encryption_change_t structure 682
bt_hci_evt_extended_inquiry_result_handler function 519
BT_HCI_EVT_EXTENDED_INQUIRY_RESULT_HANDLER macro 33
bt_hci_evt_flow_specification_complete_handler function 519
bt_hci_evt_flush_occured_handler function 519
bt_hci_evt_hardware_error_handler function 519
bt_hci_evt_inquiry_complete_handler function 519
BT_HCI_EVT_INQUIRY_COMPLETE_HANDLER macro 33
bt_hci_evt_inquiry_result_handler function 520
BT_HCI_EVT_INQUIRY_RESULT_HANDLER macro 33
bt_hci_evt_inquiry_result_with_rssi_handler function 520
BT_HCI_EVT_INQUIRY_RESULT_WITH_RSSI_HANDLER macro 33
bt_hci_evt_link_key_notification_handler function 520
BT_HCI_EVT_LINK_KEY_NOTIFICATION_HANDLER macro 33
bt_hci_evt_link_key_request_handler function 520
BT_HCI_EVT_LINK_KEY_REQUEST_HANDLER macro 34
bt_hci_evt_loopback_command_handler function 520
bt_hci_evt_master_link_key_complete_handler function 521
bt_hci_evt_max_slots_change_handler function 521
bt_hci_evt_mode_change_handler function 521
BT_HCI_EVT_MODE_CHANGE_HANDLER macro 34
bt_hci_evt_mode_change_t structure 683
bt_hci_evt_num_of_completed_packets_handler function 521
bt_hci_evt_page_scan_repet_mode_change_handler function 521

bt_hci_evt_pin_code_request_handler function 522
BT_HCI_EVT_PIN_CODE_REQUEST_HANDLER macro 34
bt_hci_evt_qos_setup_complete_handler function 522
bt_hci_evt_qos_violation_handler function 522
bt_hci_evt_read_clock_offset_complete_handler function 522
bt_hci_evt_read_rmt_ext_features_comp_handler function 522
bt_hci_evt_read_rmt_sup_features_comp_handler function 523
bt_hci_evt_read_rmt_version_info_comp_handler function 523
bt_hci_evt_remote_name_request_complete_handler function 523
BT_HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE_HANDLER macro 34
bt_hci_evt_return_link_keys_handler function 523
bt_hci_evt_role_change_handler function 523
BT_HCI_EVT_ROLE_CHANGE_HANDLER macro 34
bt_hci_evt_role_change_t structure 683
bt_hci_evt_synch_connection_changed_handler function 524
bt_hci_evt_synch_connection_complete_handler function 524
BT_HCI_EVT_SYNCH_CONNECTION_COMPLETE_HANDLER macro 35
bt_hci_exit_park_state function 524
bt_hci_exit_sniff_mode macro 562
bt_hci_exit_sniff_mode_ex function 524
bt_hci_find_devices function 524
bt_hci_find_devices_ex function 525
bt_hci_free_command function 525
bt_hci_free_data_buffer function 525
bt_hci_get_evt_param_bdaddr function 525
bt_hci_get_evt_param_byte function 525
bt_hci_get_evt_param_devclass function 526
bt_hci_get_evt_param_hconn macro 562
bt_hci_get_evt_param_int function 526
bt_hci_get_evt_param_linkkey function 526
bt_hci_get_evt_param_long function 526
bt_hci_get_evt_param_uint function 526
bt_hci_get_evt_param_uint macro 563
bt_hci_get_evt_param_ulong function 527
bt_hci_get_evt_param_ulong macro 563
bt_hci_get_inquiry_response_tx_power_level function 560
bt_hci_get_last_cmd_status function 527
bt_hci_get_param_bdaddr function 527
bt_hci_get_param_byte function 527
bt_hci_get_param_hconn macro 563
bt_hci_get_param_int function 527
bt_hci_get_param_linkkey function 528
bt_hci_get_param_long function 528
bt_hci_get_recv_buffer function 528
bt_hci_get_recv_buffer_len function 528
bt_hci_get_send_buffer function 528
bt_hci_get_send_buffer_len function 529
bt_hci_hconn_p type 683
bt_hci_hconn_t type 683
bt_hci_init function 529
bt_hci_init_data_buffers function 529
bt_hci_init_data_queues function 529
bt_hci_init_ex function 560
bt_hci_init_linkkey_buffers function 529
bt_hci_inquiry_callback_fp type 683
bt_hci_inquiry_response_t structure 684
bt_hci_le_add_device_to_white_list function 530
bt_hci_le_advertising_add function 530
bt_hci_le_advertising_add_local_name function 530
bt_hci_le_advertising_device_id_add function 530
bt_hci_le_advertising_flags_add function 530
bt_hci_le_advertising_get_local_name function 531
bt_hci_le_advertising_report_t type 684
bt_hci_le_advertising_tx_power_level_add function 531
bt_hci_le_advertising_uuid128_add function 531
bt_hci_le_advertising_uuid16_add function 531
bt_hci_le_advertising_uuid32_add function 531
bt_hci_le_advertising_vendor_add function 532
bt_hci_le_allocate_set_advertising_data_command function 532
bt_hci_le_allocate_set_scan_response_data_command function 558
bt_hci_le_cancel_connect macro 700
bt_hci_le_cancel_connect_ex function 558
bt_hci_le_cancel_find_devices function 532
bt_hci_le_clear_white_list function 532
bt_hci_le_conn_state_t structure 684
bt_hci_le_connect_ex function 532
bt_hci_le_connect_parameters_t structure 684
bt_hci_le_ctrl_state_t structure 685
bt_hci_le_enable function 533
bt_hci_le_encrypt function 533
bt_hci_le_evt_connection_updated_t structure 685
bt_hci_le_evt_read_remote_used_features_completed_t structure 686
bt_hci_le_evt_read_support_params_t structure 686
bt_hci_le_find_devices function 533
bt_hci_le_get_connect_parameters function 533
bt_hci_le_ibeacon_add function 558
bt_hci_le_init function 533
bt_hci_le_ltk_negative_reply function 534
bt_hci_le_ltk_reply function 534
bt_hci_le_rand function 534
bt_hci_le_read_channel_map function 534
bt_hci_le_read_remote_used_features function 534
bt_hci_le_read_support function 535
bt_hci_le_read_white_list_size function 535
bt_hci_le_receiver_test function 535
bt_hci_le_remove_device_from_white_list function 535
bt_hci_le_scan_callback_fp type 686
bt_hci_le_set_adevrtising_enable macro 563
bt_hci_le_set_adevrtising_enable_ex function 535
bt_hci_le_set_advertising_parameters function 536
bt_hci_le_set_connect_parameters function 536
bt_hci_le_set_host_channel_classification function 536
bt_hci_le_set_random_address function 536
bt_hci_le_set_scan_enable function 536
bt_hci_le_set_scan_parameters function 537
bt_hci_le_start_encryption function 537
bt_hci_le_supported function 537
bt_hci_le_test_end function 537
bt_hci_le_transmitter_test function 537
bt_hci_le_update_connection function 538
bt_hci_le_write_support function 538
bt_hci_link_key_t structure 686
bt_hci_listen function 538
bt_hci_listen_sco function 538

bt_hci_listener_t type 687
bt_hci_param_eir_add function 538
bt_hci_param_eir_device_id_add function 539
bt_hci_param_eir_local_name_add function 539
bt_hci_param_eir_uuid128_add function 539
bt_hci_param_eir_uuid16_add function 539
bt_hci_param_eir_uuid32_add function 539
bt_hci_param_eir_vendor_add function 540
bt_hci_param_tx_power_level_add function 540
bt_hci_park_state function 540
bt_hci_read_inquiry_mode function 540
bt_hci_read_inquiry_mode_callback_fp type 687
bt_hci_read_inquiry_scan_activity function 540
bt_hci_read_inquiry_scan_activity_callback_fp type 687
bt_hci_read_inquiry_scan_type function 541
bt_hci_read_inquiry_scan_type_callback_fp type 687
bt_hci_read_page_scan_activity function 541
bt_hci_read_page_scan_activity_callback_fp type 687
bt_hci_read_page_scan_period_mode function 541
bt_hci_read_page_scan_period_mode_callback_fp type 688
bt_hci_read_page_scan_type function 541
bt_hci_read_page_scan_type_callback_fp type 688
bt_hci_read_page_timeout function 541
bt_hci_read_page_timeout_callback_fp type 688
bt_hci_register_listener function 542
bt_hci_reject_pin_code function 542
bt_hci_request_remote_name function 542
bt_hci_request_remote_name_callback_fp type 688
bt_hci_reset function 560
bt_hci_role_change macro 563
bt_hci_role_change_ex function 542
bt_hci_sco_read_data_callback_fp type 688
bt_hci_send_acl_data function 542
bt_hci_send_cmd function 543
bt_hci_send_linkkey function 543
bt_hci_send_pin_code macro 701
bt_hci_send_pin_code_ex function 560
bt_hci_send_sco_data function 543
bt_hci_set_encryption macro 564
bt_hci_set_encryption_ex function 543
bt_hci_set_event_listener function 543
bt_hci_set_incoming_connection_role function 544
bt_hci_set_scan function 544
bt_hci_set_scan_ex function 561
bt_hci_set_vendor_specific_event_handler function 558
bt_hci_sniff_mode macro 564
bt_hci_sniff_mode_ex function 544
bt_hci_sniff_subrating macro 564
bt_hci_sniff_subrating_ex function 544
bt_hci_start function 544
bt_hci_start_callback_fp type 689
bt_hci_start_no_init function 545
bt_hci_stop function 545
bt_hci_stop_callback_fp type 689
bt_hci_transport_recv_packet function 545
bt_hci_transport_recv_packet_callback_fp type 689
bt_hci_transport_send_cmd function 546
bt_hci_transport_send_data function 546
bt_hci_transport_send_packet function 546
bt_hci_transport_send_packet_callback_fp type 689
bt_hci_transport_set_transport function 546
bt_hci_unregister_listener function 546
bt_hci_write_default_link_policy_settings function 547
bt_hci_write_eir function 547
bt_hci_write_inquiry_mode function 547
bt_hci_write_inquiry_scan_activity function 547
bt_hci_write_inquiry_scan_type function 547
bt_hci_write_link_policy_settings function 561
bt_hci_write_local_name function 548
bt_hci_write_local_name_ex function 561
bt_hci_write_page_scan_activity function 548
bt_hci_write_page_scan_period_mode function 548
bt_hci_write_page_scan_type function 548
bt_hci_write_page_timeout function 548
bt_hcitr.h 1083
bt_hcitr_3wire_cancel_recv_packet function 204
bt_hcitr_3wire_init macro 564
bt_hcitr_3wire_init_ex function 559
bt_hcitr_3wire_reset macro 564
bt_hcitr_3wire_reset_ex function 559
bt_hcitr_3wire_start function 549
bt_hcitr_bcsp_init macro 565
bt_hcitr_bcsp_init_ex function 549
bt_hcitr_bcsp_reset macro 565
bt_hcitr_bcsp_reset_ex function 549
bt_hcitr_bcsp_start function 549
bt_hcitr_packet_init function 549
bt_hcitr_packet_reset function 550
bt_hcitr_packet_start function 550
bt_hcitr_tih4_init function 550
bt_hcitr_tih4_power_callback_fp type 689
bt_hcitr_tih4_power_event_e enumeration 690
bt_hcitr_tih4_reset function 550
bt_hcitr_tih4_start function 550
bt_hcitr_tih4_wake_up function 551
bt_hcitr_uart_init function 551
bt_hcitr_uart_reset function 551
bt_hcitr_uart_start function 551
bt_id type 216
BT_INCLUDE_IAP macro 20
BT_INCLUDE_IAP2 macro 20
BT_INCLUDE_RFCOMM macro 19
bt_init_buffer_mgr function 967
bt_int type 782
bt_int_cp type 216
bt_int_p type 216
bt_l2cap_alloc_cmd_buffer function 702
bt_l2cap_alloc_cmd_config_req function 702
bt_l2cap_alloc_cmd_config_res function 702
bt_l2cap_alloc_cmd_conn_param_update_req function 702
bt_l2cap_alloc_cmd_conn_param_update_res function 702
bt_l2cap_alloc_cmd_connection_req function 703
bt_l2cap_alloc_cmd_connection_res function 703
bt_l2cap_alloc_cmd_disconnection_req function 703
bt_l2cap_alloc_cmd_disconnection_res function 703
bt_l2cap_alloc_cmd_echo_req function 703

bt_l2cap_alloc_cmd_echo_res function 704
bt_l2cap_alloc_cmd_info_req function 704
bt_l2cap_alloc_cmd_info_res function 704
bt_l2cap_alloc_cmd_reject function 704
bt_l2cap_alloc_frame_buffer function 704
bt_l2cap_allocate_channel function 705
bt_l2cap_allocate_fixed_channel function 705
bt_l2cap_allocate_mgr function 705
bt_l2cap_allocate_psm function 705
bt_l2cap_cfg_option_p structure 763
bt_l2cap_cfg_option_t structure 764
bt_l2cap_channel_ext_t type 780
bt_l2cap_channel_t type 764
bt_l2cap_cmd_assembler_fp type 764
bt_l2cap_cmd_config_req_p structure 764
bt_l2cap_cmd_config_req_t structure 765
bt_l2cap_cmd_config_res_p structure 765
bt_l2cap_cmd_config_res_t structure 765
bt_l2cap_cmd_conn_param_update_req_t structure 766
bt_l2cap_cmd_conn_param_update_res_t structure 766
bt_l2cap_cmd_connection_req_p structure 766
bt_l2cap_cmd_connection_req_t structure 766
bt_l2cap_cmd_connection_res_p structure 767
bt_l2cap_cmd_connection_res_t structure 767
bt_l2cap_cmd_disconnection_req_p structure 767
bt_l2cap_cmd_disconnection_req_t structure 767
bt_l2cap_cmd_echo_req_p structure 768
bt_l2cap_cmd_echo_req_t structure 768
bt_l2cap_cmd_echo_res_p structure 768
bt_l2cap_cmd_echo_res_t structure 768
bt_l2cap_cmd_header_p structure 769
bt_l2cap_cmd_header_t structure 769
bt_l2cap_cmd_info_req_p structure 769
bt_l2cap_cmd_info_req_t structure 770
bt_l2cap_cmd_info_res_p structure 770
bt_l2cap_cmd_info_res_t structure 770
bt_l2cap_cmd_parser_fp type 770
bt_l2cap_cmd_reject_p structure 771
bt_l2cap_cmd_reject_param_t union 771
bt_l2cap_cmd_reject_t structure 771
bt_l2cap_command_t union 772
bt_l2cap_connect macro 726
bt_l2cap_connect_callback_fp type 772
bt_l2cap_connect_ext function 705
bt_l2cap_connect_fixed_channel function 706
bt_l2cap_connect_params_t type 781
bt_l2cap_disconnect function 706
bt_l2cap_disconnect_ex function 725
bt_l2cap_echo function 706
bt_l2cap_eretr_xmit_event_e enumeration 772
bt_l2cap_find_fixed_channel function 706
bt_l2cap_find_psm function 707
bt_l2cap_fixed_channel_t structure 772
bt_l2cap_frame_desc_t structure 773
bt_l2cap_free_channel function 725
bt_l2cap_free_cmd_buffer function 707
bt_l2cap_free_fixed_channel function 707
bt_l2cap_free_frame_buffer function 707
bt_l2cap_free_mgr function 707
bt_l2cap_free_psm function 708
bt_l2cap_get_channel function 708
bt_l2cap_get_channel_by_bdaddr_cid function 708
bt_l2cap_get_channel_by_hconn_cid function 708
bt_l2cap_get_channel_by_hconn_dest_cid function 708
bt_l2cap_get_channel_by_psm function 709
bt_l2cap_get_mgr function 709
bt_l2cap_hci_has_open_channels function 725
bt_l2cap_init function 709
bt_l2cap_init_channels function 709
bt_l2cap_init_cmd_buffers function 709
bt_l2cap_init_frame_buffers function 710
bt_l2cap_init_psms function 710
bt_l2cap_is_channel_open function 726
bt_l2cap_listen macro 727
bt_l2cap_listen_callback_fp type 773
bt_l2cap_listen_ext function 710
bt_l2cap_listen_fixed_channel function 710
bt_l2cap_mgr_p structure 773
bt_l2cap_mgr_t structure 774
bt_l2cap_option_flash_timeout_t structure 774
bt_l2cap_option_max_mtu_t structure 775
bt_l2cap_option_qos_t structure 775
bt_l2cap_option_rfc_t structure 775
bt_l2cap_option_unknown_t structure 775
bt_l2cap_packet_cmd_assembler function 710
bt_l2cap_packet_data_assembler function 711
bt_l2cap_packet_t structure 776
bt_l2cap_psm_t structure 776
bt_l2cap_read_data function 711
bt_l2cap_read_data_callback_fp type 776
bt_l2cap_reject function 711
bt_l2cap_request_handler_fp type 776
bt_l2cap_response_handler_fp type 777
bt_l2cap_send_cmd function 711
bt_l2cap_send_config function 726
bt_l2cap_send_data function 711
bt_l2cap_send_data_callback_fp type 777
bt_l2cap_send_smart_data function 726
bt_l2cap_state_changed_callback_fp type 777
bt_l2cap_test_enable_local_config macro 727
bt_l2cap_test_enable_remote_config macro 727
bt_l2cap_update_conn_parameters function 712
bt_l2cap_xmit_event_param_t structure 777
BT_LE_EVT_HANDLER macro 35
bt_le_evt_handler type 690
bt_linkkey_cp type 217
BT_LINKKEY_LENGTH macro 205
bt_linkkey_notification_t structure 217
bt_linkkey_p type 217
bt_linkkey_request_t structure 217
bt_linkkey_t structure 217
BT_LOG macro 205
bt_log.h 1084
bt_log_bdaddr function 195
BT_LOG_EX macro 205
bt_log_int function 195

BT_LOG_LEVEL_A2DP_PAKCET macro 962
BT_LOG_LEVEL_ALL macro 205
BT_LOG_LEVEL_DEBUG macro 206
BT_LOG_LEVEL_ERROR macro 206
BT_LOG_LEVEL_INFO macro 206
BT_LOG_LEVEL_MAX macro 20
BT_LOG_LEVEL_MIN macro 20
BT_LOG_LEVEL_OFF macro 206
bt_log_linkkey function 196
bt_log_memory function 196
bt_log_msg function 196
BT_LOGADDR macro 206
BT_LOGADDR_EX macro 207
BT_LOGINT macro 207
BT_LOGINT_EX macro 207
BT_LOGLINKKEY macro 207
BT_LOGLINKKEY_EX macro 207
BT_LOGMEMORY macro 208
BT_LOGMEMORY_EX macro 208
BT_LOGWRITE macro 208
bt_long type 782
bt_long_cp type 218
bt_long_p type 218
BT_MAKE_BDADDR macro 208
BT_MAKE_BDADDR_LE macro 208
bt_max macro 977
bt_media_packet_t type 338
bt_min macro 977
BT_NO macro 209
bt_oem.h 1085
bt_oem_assert function 196
bt_oem_config.h 1085
bt_oem_get_device_class function 196
bt_oem_get_device_name function 197
bt_oem_get_pin_code function 197
bt_oem_linkkey_notification function 197
bt_oem_linkkey_request function 197
bt_oem_log_write function 197
bt_oem_recv function 198
bt_oem_recv_callback_fp type 218
bt_oem_schedule_signals function 198
bt_oem_send function 198
bt_oem_send_callback_fp type 218
bt_oem_ssp_callback function 203
bt_oem_storage_get_capacity function 198
bt_oem_storage_read function 199
bt_oem_storage_start function 199
bt_oem_storage_stop function 199
bt_oem_storage_write function 199
bt_oem_timer_clear function 200
bt_oem_timer_set function 200
bt_packet_assembler_fp type 996
bt_packet_t structure 996
bt_private.h 1086
bt_q_add function 968
bt_q_contains function 204
bt_q_get function 968
bt_q_get_head function 968
bt_q_get_length function 968
bt_q_get_next function 968
bt_q_push function 969
bt_q_remove function 969
bt_q_remove_by_idx function 969
bt_queue_element_t structure 996
bt_rfcomm_allocate_dlc function 783
bt_rfcomm_allocate_session function 783
bt_rfcomm_cancel_listen function 798
bt_rfcomm_close_dlc function 783
bt_rfcomm_cmd_callback_fp type 824
bt_rfcomm_command_p structure 824
bt_rfcomm_command_t structure 824
bt_rfcomm_connect function 784
bt_rfcomm_ctl_msg_p structure 825
bt_rfcomm_ctl_msg_t structure 825
bt_rfcomm_dlc_p structure 825
bt_rfcomm_dlc_state_callback_fp type 826
bt_rfcomm_dlc_t structure 826
bt_rfcomm_find_dlc function 784
bt_rfcomm_free_dlc function 784
bt_rfcomm_free_session function 784
bt_rfcomm_get_frame_length function 798
bt_rfcomm_init function 785
bt_rfcomm_listen function 785
bt_rfcomm_mgr_t structure 826
bt_rfcomm_open_dlc function 785
bt_rfcomm_read_data_callback_fp type 826
bt_rfcomm_register_listener function 785
bt_rfcomm_send_credit function 786
bt_rfcomm_send_data function 786
bt_rfcomm_send_data_callback_fp type 827
bt_rfcomm_server_channel_t structure 827
bt_rfcomm_session_listener_t structure 827
bt_rfcomm_session_p structure 827
bt_rfcomm_session_t structure 828
bt_rfcomm_state_callback_fp type 828
bt_rfcomm_unregister_listener function 786
bt_sdp_client_callback_fp type 875
bt_sdp_client_evt_connected_t structure 874
bt_sdp_client_evt_disconnected_t structure 874
bt_sdp_data_element_cp type 875
bt_sdp_data_element_p type 875
bt_sdp_data_element_t structure 875
bt_sdp_de_to_uuid function 837
bt_sdp_found_attr_list_t structure 876
bt_sdp_packet_assembler function 837
bt_sdp_packet_t structure 876
bt_sdp_read_attribute function 837
bt_sdp_read_de_callback_ftp type 876
bt_sdp_request_service_attribute function 840
bt_sdp_request_service_search function 841
bt_sdp_sequence_cp type 877
bt_sdp_sequence_p type 877
bt_sdp_sequence_t structure 877
bt_sdp_serialization_state_p structure 877
bt_sdp_serialization_state_t structure 878
bt_sdp_server_attribute_t structure 878

bt_sdp_server_data_element_t structure 878
bt_sdp_server_record_t structure 878
bt_sdp_service_attribute_callback_fp type 879
bt_sdp_service_search_callback_fp type 879
bt_sdp_service_transaction_p structure 879
bt_sdp_service_transaction_t structure 879
bt_sdp_start function 837
bt_sdp_transaction_t structure 880
bt_signal.h 1087
bt_signal_handler_fp type 218
bt_signal_init function 203
bt_signal_process_pending function 200
bt_signal_register function 200
bt_signal_set function 200
bt_signal_t type 219
bt_signal_unregister function 201
bt_spp_allocate function 883
bt_spp_cancel_listen function 899
bt_spp_cancel_receive function 883
bt_spp_cancel_send function 883
bt_spp_clr_port_options function 884
bt_spp_connect function 884
bt_spp_deallocate function 884
bt_spp_disconnect function 884
bt_spp_find_server function 885
bt_spp_find_server_callback_fp type 892
bt_spp_find_server_ex function 899
bt_spp_get_frame_length function 885
bt_spp_get_hci_connection function 885
bt_spp_get_local_modem_status function 885
bt_spp_get_remote_address function 886
bt_spp_get_remote_modem_status function 886
bt_spp_init function 886
bt_spp_listen function 886
bt_spp_port_event_e enumeration 892
bt_spp_port_state_e enumeration 893
bt_spp_port_t type 894
bt_spp_read_local_oob_data_callback_fp type 901
bt_spp_receive function 887
bt_spp_receive_callback_fp type 894
bt_spp_send function 887
bt_spp_send_callback_fp type 894
bt_spp_send_status_e enumeration 894
bt_spp_set_dtr function 887
bt_spp_set_local_modem_status function 888
bt_spp_set_port_options function 888
bt_spp_set_rts function 888
bt_spp_state_callback_fp type 895
bt_sr_handle_p type 880
bt_sr_handle_t type 880
bt_ssp_evt_handler function 895
BT_SSP_EVT_HANDLER macro 35
bt_ssp_init function 895
bt_ssp_io_capability structure 901
bt_ssp_keypress_notification structure 901
bt_ssp_oob_data structure 901
bt_ssp_read_local_oob_data function 895
bt_ssp_send_keypress_notification function 896
bt_ssp_send_user_confirmation function 896
bt_ssp_send_user_passkey function 896
bt_ssp_set_io_capabilities function 896
bt_ssp_set_mode function 896
bt_ssp_set_oob_data function 897
bt_ssp_simple_pairing_complete structure 902
bt_ssp_user_confirmation_request structure 902
bt_ssp_user_passkey_notification structure 902
bt_ssp_user_passkey_request structure 902
bt_std.h 1087
bt_storage.h 1087
bt_storage_callback_fp type 219
bt_sys_callback_fp type 219
bt_sys_get_connectable function 201
bt_sys_get_discoverable function 201
bt_sys_get_l2cap_manager function 201
bt_sys_init function 201
bt_sys_init_ex function 203
bt_sys_set_modes function 202
bt_sys_start function 202
bt_system.h 1088
bt_timer.h 1089
BT_TIMER_3WIRE_T0 enumeration member 220
BT_TIMER_3WIRE_T1 enumeration member 220
BT_TIMER_ATT enumeration member 220
BT_TIMER_ATT_CLIENT enumeration member 220
BT_TIMER_AVRCP enumeration member 220
bt_timer_callback_fp type 219
BT_TIMER_HCI enumeration member 220
BT_TIMER_HCRP enumeration member 220
BT_TIMER_HFP_AG enumeration member 220
BT_TIMER_HSP_AG enumeration member 220
BT_TIMER_IAP enumeration member 220
BT_TIMER_IAP2 enumeration member 220
bt_timer_id enumeration 220
BT_TIMER_L2CAP enumeration member 220
BT_TIMER_MAX enumeration member 220
BT_TIMER_RFCOMM enumeration member 220
BT_TIMER_SMP enumeration member 220
BT_TIMER_TEST enumeration member 220
BT_TIMER_WAKEUP_ACK enumeration member 220
BT_TRUE macro 209
bt_types.h 1090
bt_uint type 782
bt_uint_cp type 220
bt_uint_p type 220
bt_ulong type 783
bt_ulong_cp type 220
bt_ulong_p type 221
bt_uuid_cp type 221
bt_uuid_p type 221
bt_uuid_t structure 221
bt_uuid16 type 221
bt_uuid32 type 222
bt_vcard_evt_prop_param_t structure 996
bt_vcard_evt_prop_t structure 997
bt_vcard_parse_fragment function 969
bt_vcard_parser_callback_fp type 997

bt_vcard_parser_reset function 969
bt_vcard_parser_t structure 997
bt_xml_evt_attribute_t structure 997
bt_xml_evt_attribute_value_t structure 998
bt_xml_evt_tag_started_t structure 998
bt_xml_parse_fragment function 970
bt_xml_parser_callback_pf type 998
bt_xml_parser_reset function 970
bt_xml_parser_t structure 998
BT_YES macro 209
btx_csr_alloc_bccmd_getreq function 904
btx_csr_alloc_bccmd_setreq function 904
btx_csr_autobaud function 905
btx_csr_autobaud_buffer_t type 943
btx_csr_autobaud_callback_fp type 944
btx_csr_bc7_sel_host_interface_h4 function 905
btx_csr_bccmd_callback_fp type 963
btx_csr_bccmd_header_t structure 944
btx_csr_bccmd_listener_t type 963
btx_csr_cached_temperature_t structure 944
btx_csr_create_operator_c_t structure 963
btx_csr_enable_tx function 905
btx_csr_exec_hq_script function 923
btx_csr_exec_hq_script_buffer_t type 964
btx_csr_exec_hq_script_callback_fp type 964
btx_csr_exec_script function 905
btx_csr_exec_script_buffer_t type 944
btx_csr_exec_script_callback_fp type 945
btx_csr_get_cached_temperature function 906
btx_csr_get_ps_var function 906
btx_csr_get_ps_var_callback_fp type 945
btx_csr_get_ps_var_ex function 906
btx_csr_get_rssi_acl function 906
btx_csr_get_script__dsp_script__PB_109_DSP_rev8 function 917
btx_csr_get_script__PB_101_CSR8811_CSP28_UART function 917
btx_csr_get_script__PB_109_CSR8811_REV16 function 917
btx_csr_get_script__PB_173_CSR8X11_REV1 function 917
btx_csr_get_script__PB_27_R20_BC6ROM_A04 function 906
btx_csr_get_script__PB_90_REV6 function 907
btx_csr_get_script_fp type 964
btx_csr_get_var function 907
btx_csr_get_var_callback_fp type 945
btx_csr_init function 918
btx_csr_init_hq_script function 918
btx_csr_patch_controller function 918
btx_csr_pio_direction_mask_t structure 945
btx_csr_pio_protection_mask_t structure 945
btx_csr_pio_t structure 946
btx_csr_register_bccmd_listener function 918
btx_csr_rssi_acl_t structure 946
btx_csr_script_t structure 946
btx_csr_send_dsp_config_data function 919
btx_csr_send_next_hq_script_packet function 919
btx_csr_set_ps_var function 907
btx_csr_set_ps_var_ex function 907
btx_csr_set_ps_vars function 907
btx_csr_set_ps_vars_buffer_t type 946
btx_csr_set_ps_vars_callback_fp type 947
btx_csr_set_ps_vars_ex function 908
btx_csr_set_var function 908
btx_csr_set_var_callback_fp type 947
btx_csr_strm_connect_t structure 964
btx_csr_strm_get_sink_t structure 947
btx_csr_strm_get_source_t structure 947
btx_csr_unregister_bccmd_listener function 919
btx_csr_var_t union 947
btx_csr_warm_reset function 908
btx_csr_warm_reset_ex function 908
BTX_TI_A3DP_ROLE_SINK macro 949
BTX_TI_A3DP_ROLE_SOURCE macro 949
btx_ti_a3dp_sink_close_stream function 919
btx_ti_a3dp_sink_codec_config function 919
btx_ti_a3dp_sink_open_stream function 920
btx_ti_a3dp_sink_start_stream function 920
btx_ti_a3dp_sink_stop_stream function 920
btx_ti_avpr_debug function 920
BTX_TI_AVPR_DISABLE macro 949
BTX_TI_AVPR_DO_NOT_LOAD_A3DP_CODE macro 949
btx_ti_avpr_enable function 920
BTX_TI_AVPR_ENABLE macro 950
BTX_TI_AVPR_LOAD_A3DP_CODE macro 950
btx_ti_codec_config_t structure 964
btx_ti_completion_callback_fp type 948
btx_ti_drpb_enable_rf_calibration function 909
btx_ti_drpb_set_power_vector function 909
btx_ti_drpb_tester_con_tx function 909
btx_ti_enable_deep_sleep function 909
btx_ti_enable_fast_clock_crystal function 910
btx_ti_enable_low_power_scan function 910
btx_ti_enable_low_power_scan_default function 910
btx_ti_exec_script function 910
btx_ti_exec_script_buffer_t type 948
btx_ti_exec_script_oem function 910
btx_ti_exec_script_oem_buffer_t type 948
btx_ti_exec_script_oem_callback_fp type 948
BTX_TI_EXEC_SCRIPT_OEM_RX_BUFFER_SIZE macro 938
btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_36 function 911
btx_ti_get_script__BL6450_2_0_BT_Service_Pack_2_44 function 911
btx_ti_get_script__BL6450L_BT_Service_Pack_2_10 function 911
btx_ti_get_script__BL6450L_BT_Service_Pack_2_10_BLE_AddOn function 911
btx_ti_get_script__BL6450L_BT_Service_Pack_2_12 function 911
btx_ti_get_script__BL6450L_BT_Service_Pack_2_12_BLE_AddOn function 912
btx_ti_get_script__BL6450L_BT_Service_Pack_2_14 function 921
btx_ti_get_script__BL6450L_BT_Service_Pack_2_14_BLE_AddOn function 921
btx_ti_get_script__BL6450L_BT_Service_Pack_2_7 function 912
btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_AVPR_AddOn function 912
btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_BLE_AddOn function 912
btx_ti_get_script__BL6450L_BT_Service_Pack_2_8_Short function 912
btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_AVPR_AddOn function 913
btx_ti_get_script__BL6450x_BT_Service_Pack_2_7_BLE_AddOn function 913

btx_ti_get_script__CC2560_ServicePack macro 938
btx_ti_get_script__CC2564_BLE_Init macro 938
btx_ti_get_script__CC2564_ServicePack macro 938
btx_ti_get_script__CC2564B_AVPR_Init macro 950
btx_ti_get_script__CC2564B_BLE_Init macro 939
btx_ti_get_script__CC2564B_BT_Service_Pack_0_1 function 913
btx_ti_get_script__CC2564B_BT_Service_Pack_0_1_BLE_AddOn function 913
btx_ti_get_script__CC2564B_BT_Service_Pack_0_2 function 913
btx_ti_get_script__CC2564B_BT_Service_Pack_0_2_BLE_AddOn function 914
btx_ti_get_script__CC2564B_BT_Service_Pack_1_0 function 921
btx_ti_get_script__CC2564B_BT_Service_Pack_1_0_BLE_AddOn function 921
btx_ti_get_script__CC2564B_BT_Service_Pack_1_1 function 921
btx_ti_get_script__CC2564B_BT_Service_Pack_1_1_BLE_AddOn function 922
btx_ti_get_script__CC2564B_BT_Service_Pack_1_2 function 922
btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_AVPR_AddOn function 922
btx_ti_get_script__CC2564B_BT_Service_Pack_1_2_BLE_AddOn function 922
btx_ti_get_script__CC2564B_ServicePack macro 939
btx_ti_get_script__WL127xL_BT_Service_Pack_2_3 function 914
btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_AVPR_AddOn function 914
btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_BLE_AddOn function 914
btx_ti_get_script__WL127xL_BT_Service_Pack_2_3_DC2DC_AddOn function 914
btx_ti_get_script__WL127xL_BT_Service_Pack_2_4 function 915
btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_AVPR_AddOn function 915
btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_BLE_AddOn function 915
btx_ti_get_script__WL127xL_BT_Service_Pack_2_4_DC2DC_AddOn function 915
btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3 function 915
btx_ti_get_script__XWL1271L1_BT_ServicePack_1_3_BLE_Init function 916
btx_ti_init_ble_controller function 916
btx_ti_init_controller function 916
btx_ti_le_enable function 922
BTX_TI_MODULATION_SCHEME_8_DPSK macro 939
BTX_TI_MODULATION_SCHEME_CW macro 939
BTX_TI_MODULATION_SCHEME_GFSK macro 939
BTX_TI_MODULATION_SCHEME_P4_DPSK macro 940
BTX_TI_MODULATION_TYPE_EDR2 macro 940
BTX_TI_MODULATION_TYPE_EDR3 macro 940
BTX_TI_MODULATION_TYPE_GFSK macro 940
BTX_TI_SBC_ALLOCATION_METHOD_LOUDNESS macro 950
BTX_TI_SBC_ALLOCATION_METHOD_SNR macro 950
BTX_TI_SBC_BLOCKS_12 macro 951
BTX_TI_SBC_BLOCKS_16 macro 951
BTX_TI_SBC_BLOCKS_4 macro 951
BTX_TI_SBC_BLOCKS_8 macro 951
BTX_TI_SBC_CHANNEL_MODE_DUAL_CHANNEL macro 951
BTX_TI_SBC_CHANNEL_MODE_JOINT_STEREO macro 952
BTX_TI_SBC_CHANNEL_MODE_MONO macro 952
BTX_TI_SBC_CHANNEL_MODE_STEREO macro 952
BTX_TI_SBC_SAMPLE_FREQUENCY_16000 macro 952
BTX_TI_SBC_SAMPLE_FREQUENCY_32000 macro 952
BTX_TI_SBC_SAMPLE_FREQUENCY_44100 macro 953
BTX_TI_SBC_SAMPLE_FREQUENCY_48000 macro 953
btx_ti_script_t structure 949
btx_ti_set_afh_mode function 923
btx_ti_set_uart_baud_rate function 916
BTX_TI_TEST_PATTERN_ALL_0 macro 940
BTX_TI_TEST_PATTERN_ALL_1 macro 941
BTX_TI_TEST_PATTERN_F0F0 macro 941
BTX_TI_TEST_PATTERN_FF00 macro 941
BTX_TI_TEST_PATTERN_PN15 macro 941
BTX_TI_TEST_PATTERN_PN9 macro 941
BTX_TI_TEST_PATTERN_USER macro 942
BTX_TI_TEST_PATTERN_Z0Z0 macro 942
btx_ti_write_bdaddr function 916
btx_ti_write_codec_config function 923
btx_ti_write_hardware_register function 917
Buffer Queue Transfer Model 2338
buffer.h 1091
BUFFER_HDR_LEN macro 978
BUFFER_STATE_FREE macro 978
BUFFER_STATE_USED macro 978
bufferutils.h 1091
Building the Library 54, 1180, 1195, 1212, 1369, 1405, 1453, 1490, 1529, 1559, 1572, 1592, 1613, 1633, 1679, 1740, 1784, 1839, 1875, 1907, 1940, 1977, 2051, 2097, 2115, 2135, 2152, 2222, 2280, 2349, 2407, 2498, 2588, 3158, 3206, 4422, 5579, 5616, 5636, 5660, 5675, 5693, 5726, 5778, 5839, 5856, 5866, 5892, 5918, 5928, 5945, 5966, 5978, 5985, 6008, 6013, 6031, 6066, 6082, 6101, 6119, 6141, 6150, 6160, 6173, 6216, 6269, 6279, 6297, 6302, 6317, 6359, 6404, 6469, 6476, 6481, 6487, 6501, 6522, 6569, 6584, 6620, 6631, 6639, 6668, 6683, 6726, 6755, 6793, 6870, 6905, 6930, 6942, 6968, 7034, 7115, 7142, 7150, 7156
    10-bit ADC Touch Driver Library 2097
    AK4384 Driver Library 1405
    AK4642 Driver Library 1453
    AK4953 Driver Library 1490
    AK7755 Driver Library 1529
    Announce Module 6008
    AR1021 Touch Driver Library 2115
    ARP Module 6013
    Berkeley Module 6031
    Bootloader Library 1180
    Class B Library 1195
    Clock System Service Library 5616
    Command Processor System Service Library 5636
    Console System Service Library 5660
    CPLD XC2C64A Driver Library 1559
    Crypto Library 1212
    Debug System Service Library 5675
    Device Control System Service Library 5693
    DHCP Module 6066
    DHCP Server Module 6082
    DHCPv6 Module 6101
    DMA System Service Library 5726
    DNS Module 6119
    DNSS Module 6141
    Dynamic DNS Module 6150
    ENC28J60 Driver Library 1572
    ENCX24J600 Driver Library 1592

Ethernet MAC Driver Library 1613
Ethernet PHY Driver Library 1633
File System Service Library 5778
FTP Module 6160
Generic USB Device Library 6942
Graphics Driver Library 1679
Graphics Library 2588
HTTP Module 6173
HTTP Net Module 6216
I2C Driver Library 1740
I2S Driver Library 1784
ICMP Module 6269
ICMPv6 Module 6279
Interrupt System Service Library 5839
IPv4 Module 6302
IPv6 Module 6317
MAC Driver Module 6359
Manager Module 6404
Memory System Service Library 5856
Messaging System Service Library 5866
MRF24W Wi-Fi Driver Library 2407
MRF24WN Wi-Fi Driver Library 2498
MTCH6301 Touch Driver Library 2135
MTCH6303 Touch Driver Library 2152
NBNS Module 6469
NDP Module 6476
Networking Presentation Layer 3158
NVM Driver Library 1839
NVM Peripheral Library 4422
OSAL Library 3206
PIC32 Bluetooth Stack Library 54
PMP Driver Library 1875
Ports System Service Library 5892
Random Number Generator System Service Library 5918
Reboot Module 6481
Reset System Service Library 5928
Sample Library 5579
SD Card Driver Library 1907
SMTP Module 6487
SMTPC Module 6501
SNMP Module 6522
SNTP Module 6569
SPI Driver Library 1940
SPI Flash Driver Library 1977
TCP Module 6584
TCPIP Stack Library 5985
Telnet Module 6631
Test Harness Library 6683
TFTP Module 6620
Timer Driver Library 2051
Timer System Service Library 5945
Touch System Service Library 5966
UDP Module 6639
USART Driver Library 2349
USB Audio 2.0 Device Library 6755
USB Audio Device Library 6726
USB Audio v1.0 Host Client Driver Library 7034
USB CDC Host Library 7115
USB Device Layer Library 6793
USB HID Device Library 6905
USB HID Host Mouse Driver Library 7142
USB Host Layer Library 6968
USB Hub Host Client Driver Library 7150
USB MSD Device Library 6930
USB MSD Host Library 7156
Watchdog Timer System Service Library 5978
Zeroconf Module 6668
BURLYWOOD macro 2653
Bus Arbiter 3483
Button Object 2720
Byte Transfer Model 2336
BYTE_SIZE macro 782
BYTES_PER_SAMPLE macro 1333

C

Cache Control 3483
Cache Control Operations 4713
Cache Line Operations 4714
Cache Status Operations 4714
Calling the Device Layer API 6708
Camera Driver Libraries 1358
CAMERA_MODULE_ID enumeration 1364
CAN Bit Time Quanta 3519
CAN Driver Library 1388
CAN Peripheral Library 3514
CAN_CHANNEL enumeration 3594
CAN_CHANNEL_EVENT enumeration 3596
CAN_CHANNEL_MASK enumeration 3596
CAN_DNET_FILTER_SIZE enumeration 3598
CAN_ERROR_STATE enumeration 3599
CAN_FILTER enumeration 3599
CAN_FILTER_MASK enumeration 3601
CAN_FILTER_MASK_TYPE enumeration 3602
CAN_ID_TYPE enumeration 3602
CAN_MODULE_EVENT enumeration 3602
CAN_MODULE_ID enumeration 3603
CAN_MSG_EID structure 3604
CAN_OPERATION_MODES enumeration 3604
CAN_RECEIVE_CHANNEL enumeration 3605
CAN_RECEIVE_MODES enumeration 3605
CAN_RX_DATA_MODE enumeration 3606
CAN_RX_DATA_ONLY_SIZE_BYTES macro 3610
CAN_RX_MSG_BUFFER union 3606
CAN_RX_MSG_SID structure 3607
CAN_TIME_SEGMENT_LENGTH enumeration 3607
CAN_TX_CHANNEL_STATUS enumeration 3608
CAN_TX_MSG_BUFFER union 3608
CAN_TX_MSG_SID structure 3608
CAN_TX_RTR enumeration 3609
CAN_TX_RX_MESSAGE_SIZE_BYTES macro 3610
CAN_TXCHANNEL_PRIORITY enumeration 3609
Capacitive Touch Measurement 3661
CDS_LAST_DEVICE_ADDR macro 209
CDS_SIGNATURE macro 209
CDS_SIGNATURE_ADDR macro 210
Change Notification 5890

Changing the Clock 5611
Changing the System Error Level 5675
Channel Configuration 3521
Channel Events 3524
Channel Management 3743
Channel Setup and Management 5719
channel.h 1092
CHANNEL_SIGNAL_CMD_DISCONNECT_FIXED macro 727
Check Box Object 2731
check_fcs function 786
chmanager.h 1094
Chnl_Info structure 1332
CID_ATT macro 727
CID_LE_SIG macro 728
CID_MAX macro 728
CID_MAX_FIXED macro 728
CID_NULL macro 728
CID_RECV macro 728
CID_SIG macro 729
CID_SM macro 729
Class B Library Help 1191
CLASSB_ClockLineFreqTest function 1197
CLASSB_ClockTest function 1198
classb_config_template.h 1209
CLASSB_CPUPCTest function 1199
CLASSB_CPURegistersTest function 1200
CLASSB_CRCFlashTest function 1200
CLASSB_RAM_TEST_CYCLE_SIZE macro 1206
CLASSB_RAMCheckerBoardTest function 1201
CLASSB_RAMMarchBTest function 1202
CLASSB_RAMMarchCStackTest function 1203
CLASSB_RAMMarchCTest function 1204
CLASSBRESULT enumeration 1205
classCode enumeration 6989
Client Access 1395, 1444, 1484, 1523, 1730, 1770, 1932
Client Access Operation 1832, 1903
Client Block Data Operation 1833, 1904
Client Functionality 2404, 2496
Client Interaction 2044
Client Operation 1871
Client Operations 1395, 1445, 1484, 1523
Client Operations - Buffered 1771
Client Operations - Non-buffered 1776
Client Transfer 1730
Client Transfer - Core 1933
CLK_BUSES_PERIPHERAL enumeration 5627
CLK_BUSES_REFERENCE enumeration 5628
CLK_SOURCES_PERIPHERAL enumeration 5628
CLK_SOURCES_REFERENCE enumeration 5629
CLK_SOURCES_SYSTEM enumeration 5629
Clock Sources 4511
Clock System Service Library 5608
CLOCK_TEST_INTERRUPT_SOURCE macro 1206
CLOCK_TEST_REFERENCE_FREQ macro 1206
CLOCK_TEST_TIMER macro 1206
closesocket function 6034
Closing a File 5767
cmd_disconnection_res structure 778
cmdbuffer.h 1094
CMODE_BASIC macro 729
CMODE_ERETR macro 729
CMODE_FLOW macro 729
CMODE_RETR macro 730
CMODE_STRM macro 730
CMP_CLOCK_DIVIDE enumeration 3643
CMP_CVREF_BANDGAP_SELECT enumeration 3644
CMP_CVREF_REFERENCE_SELECT enumeration 3644
CMP_CVREF_VALUE enumeration 3644
CMP_CVREF_VOLTAGE_SOURCE enumeration 3645
CMP_FILTER_CLOCK enumeration 3646
CMP_INTERRUPT_EVENT enumeration 3646
CMP_INVERTING_INPUT enumeration 3647
CMP_MASK_A enumeration 3647
CMP_MASK_B enumeration 3648
CMP_MASK_C enumeration 3649
CMP_MODULE_ID enumeration 3649
CMP_NON_INVERTING_INPUT enumeration 3650
CMP_SPEED_POWER enumeration 3650
COD_MAJOR_AUDIO macro 565
COD_MAJOR_COMPUTER macro 565
COD_MAJOR_HEALTH macro 565
COD_MAJOR_IMAGING macro 566
COD_MAJOR_MISC macro 566
COD_MAJOR_NET_ACCESS_POINT macro 566
COD_MAJOR_PERIPHERAL macro 566
COD_MAJOR_PHONE macro 566
COD_MAJOR_TOY macro 567
COD_MAJOR_UNCATEGORIZED macro 567
COD_MAJOR_WEARABLE macro 567
COD_MINOR_AV_CAMCORDER macro 567
COD_MINOR_AV_CAR_AUDIO macro 567
COD_MINOR_AV_GAMING macro 568
COD_MINOR_AV_HANDSFREE macro 568
COD_MINOR_AV_HEADPHONES macro 568
COD_MINOR_AV_HEADSET macro 568
COD_MINOR_AV_HIFI_AUDIO macro 568
COD_MINOR_AV_LOUDSPEAKER macro 569
COD_MINOR_AV_MICROPHONE macro 569
COD_MINOR_AV_PORTABLE_AUDIO macro 569
COD_MINOR_AV_RESERVED macro 569
COD_MINOR_AV_RESERVERD2 macro 569
COD_MINOR_AV_SET_TOP_BOX macro 570
COD_MINOR_AV_UNCATEGORIZED macro 570
COD_MINOR_AV_VCR macro 570
COD_MINOR_AV_VIDE_DISPLAY_AND_LOUDSPEAKER macro 570
COD_MINOR_AV_VIDEO_CAMERA macro 570
COD_MINOR_AV_VIDEO_CONFERENCING macro 571
COD_MINOR_AV_VIDEO_MONITOR macro 571
COD_MINOR_COMPUTER_DESKTOP macro 571
COD_MINOR_COMPUTER_HANDHELD macro 571
COD_MINOR_COMPUTER_LAPTOP macro 571
COD_MINOR_COMPUTER_PALMSIZED macro 572
COD_MINOR_COMPUTER_SERVER macro 572
COD_MINOR_COMPUTER_UNCATEGORIZED macro 572
COD_MINOR_COMPUTER_WEARABLE macro 572
COD_MINOR_HEALTH_BPM macro 572

COD_MINOR_HEALTH_DATA_DISPLAY macro 573
COD_MINOR_HEALTH_GLUCOSE_METER macro 573
COD_MINOR_HEALTH_HEART_MONITOR macro 573
COD_MINOR_HEALTH_PULSE_OXIMETER macro 573
COD_MINOR_HEALTH_THERMOMETER macro 573
COD_MINOR_HEALTH_UNCATEGORIZED macro 574
COD_MINOR_HEALTH_WEIGHING_SCALE macro 574
COD_MINOR_IMAGING_CAMERA macro 574
COD_MINOR_IMAGING_DISPLAY macro 574
COD_MINOR_IMAGING_PRINTER macro 574
COD_MINOR_IMAGING_SCANNER macro 575
COD_MINOR_IMAGING_UNCATEGORIZED macro 575
COD_MINOR_LAN_01_17 macro 575
COD_MINOR_LAN_17_33 macro 575
COD_MINOR_LAN_33_50 macro 575
COD_MINOR_LAN_50_67 macro 576
COD_MINOR_LAN_67_83 macro 576
COD_MINOR_LAN_83_99 macro 576
COD_MINOR_LAN_FULLY_AVAILABLE macro 576
COD_MINOR_LAN_NO_SERVICE macro 576
COD_MINOR_LAN_UNCATEGORIZED macro 577
COD_MINOR_PERIPHERAL_CARD_READER macro 577
COD_MINOR_PERIPHERAL_COMBO macro 577
COD_MINOR_PERIPHERAL_DIGITIZER macro 577
COD_MINOR_PERIPHERAL_GAMEPAD macro 577
COD_MINOR_PERIPHERAL_JOYSTICK macro 578
COD_MINOR_PERIPHERAL_KEYBOARD macro 578
COD_MINOR_PERIPHERAL_MOUSE macro 578
COD_MINOR_PERIPHERAL_OTHER macro 578
COD_MINOR_PERIPHERAL_REMOTE macro 578
COD_MINOR_PERIPHERAL_SENSING macro 579
COD_MINOR_PERIPHERAL_UNCATEGORIZED macro 579
COD_MINOR_PHONE_CELLULAR macro 579
COD_MINOR_PHONE_CORDLESS macro 579
COD_MINOR_PHONE_ISDN macro 579
COD_MINOR_PHONE_SMART macro 580
COD_MINOR_PHONE_UNCATEGORIZED macro 580
COD_MINOR_PHONE_WIREDMODEM macro 580
COD_MINOR_TOY_CONTROLLER macro 580
COD_MINOR_TOY_DOLL macro 580
COD_MINOR_TOY_GAME macro 581
COD_MINOR_TOY_ROBOT macro 581
COD_MINOR_TOY_UNCATEGORIZED macro 581
COD_MINOR_TOY_VEHICLE macro 581
COD_MINOR_WEARABLE_GLASSES macro 581
COD_MINOR_WEARABLE_HELMET macro 582
COD_MINOR_WEARABLE_JACKET macro 582
COD_MINOR_WEARABLE_PAGER macro 582
COD_MINOR_WEARABLE_UNCATEGORIZED macro 582
COD_MINOR_WEARABLE_WATCH macro 582
Codec Driver Libraries 1391
Color Definitions 2644
Color Functions 2625
Command AppIO Console Device Configuration Options 5658
Command Processor System Service Library 5634
command.h 1095
COMMAND_HISTORY_DEPTH macro 5643
Common Interface 2181
Communication Mode 5033
Comparator Driver Library 1558
Comparator Peripheral Library 3613
Comparison of API Names 5760
Compression 6168
config.h 1098
Configuration Examples 2586
Configuring a Library 3224
    Peripheral Library Overview 3224
Configuring DMA SPI 2397
Configuring the Comparator Interrupts 3616
Configuring the Display 1679
Configuring the Library 7, 1180, 1195, 1212, 1368, 1401, 1450, 1487, 1526, 1559, 1571, 1591, 1610, 1631, 1675, 1735, 1780, 1835, 1874, 1905, 1935, 1970, 2048, 2094, 2113, 2132, 2152, 2218, 2277, 2340, 2407, 2497, 2577, 3158, 3205, 3251, 3316, 3441, 3487, 3525, 3617, 3663, 3686, 3701, 3747, 3869, 3936, 3994, 4140, 4260, 4344, 4368, 4422, 4484, 4513, 4625, 4715, 4765, 4850, 4897, 4917, 4973, 5034, 5114, 5250, 5294, 5376, 5517, 5566, 5577, 5613, 5636, 5658, 5675, 5693, 5726, 5776, 5837, 5856, 5863, 5892, 5917, 5928, 5944, 5966, 5977, 6007, 6011, 6030, 6065, 6079, 6095, 6115, 6139, 6149, 6158, 6169, 6206, 6268, 6279, 6301, 6313, 6355, 6404, 6468, 6473, 6480, 6485, 6497, 6517, 6566, 6578, 6618, 6630, 6637, 6668, 6678, 6724, 6754, 6791, 6869, 6904, 6929, 6942, 6965, 7032, 7114, 7140, 7149, 7155
    10-bit ADC Touch Driver Library 2094
    ADC Peripheral Library 3251
    ADCHS Peripheral Library 3316
    ADCP Peripheral Library 3441
    AK4384 Driver Library 1401
    AK4642 Driver Library 1450
    AK4953 Driver Library 1487
    AK7755 Driver Library 1526
    Announce Module 6007
    AR1021 Touch Driver Library 2113
    ARP Module 6011
    Berkeley Module 6030
    BMX Peripheral Library 3487
    Bootloader Library 1180
    CAN Peripheral Library 3525
    Class B Library 1195
    Clock System Service Library 5613
    Command Processor System Service Library 5636
    Console System Service Library 5658
    CPLD XC2C64A Driver Library 1559
    Crypto Library 1212
    CTMU Peripheral Library 3663
    Deadman Timer Peripheral Library 3686
    Debug System Service Library 5675
    Device Control System Service Library 5693
    DHCP Module 6065
    DHCP Server Module 6079
    DHCPv6 Module 6095
    DMA Peripheral Library 3747
    DMA System Service Library 5726
    DNS Module 6115
    DNSS Module 6139
    Dual Data Rate (DDR) SDRAM Peripheral Library 3869
    Dynamic DNS Module 6149
    EBI Peripheral Library 3936
    Ethernet MAC Driver Library 1610

Ethernet PHY Driver Library 1631
File System Service Library 5776
FTP Module 6158
Generic USB Device Library 6942
GLCD Controller Peripheral Library 4140
Graphics Driver Library 1675
HTTP Module 6169
HTTP Net Module 6206
I2C Peripheral Library 4260
ICMP Module 6268
ICMPv6 Module 6279
Input Capture Peripheral Library 4344
Interrupt Peripheral Library 4368
Interrupt System Service Library 5837
IPv4 Module 6301
IPv6 Module 6313
MAC Driver Module 6355
Manager Module 6404
Memory System Service Library 5856
Messaging System Service Library 5863
MRF24W Wi-Fi Driver Library 2407
MRF24WN Wi-Fi Driver Library 2497
MTCH6301 Touch Driver Library 2132
MTCH6303 Touch Driver Library 2152
NBNS Module 6468
NDP Module 6473
Networking Presentation Layer 3158
NVM Driver Library 1835
NVM Peripheral Library 4422
OSAL Library 3205
Oscillator Peripheral Library 4513
Output Compare Peripheral Library 4484
PIC32 Bluetooth Stack Library 7
PMP Driver Library 1874
PMP Peripheral Library 4625
Ports Peripheral Library 4765
Ports System Service Library 5892
Power Peripheral Library 4850
Prefetch Cache Peripheral Library 3701, 4715
Random Number Generator System Service Library 5917
Reboot Module 6480
Reset Peripheral Library 4897
RTCC Peripheral Library 4917
Sample Library 5577
SD Card Driver Library 1905
SMPT Module 6485, 6497
SNMP Module 6517
SNTP Module 6566
SPI Driver Library 1935
SPI Flash Driver Library 1970
SPI Peripheral Library 5034
SQI Peripheral Library 5114
System Bus Peripheral Library 4973
TCP Module 6578
Telnet Module 6630
Test Harness Library 6678
TFTP Module 6618
Timer Driver Library 2048
Timer Peripheral Library 5250
Timer System Service Library 5944
Touch System Service Library 5966
UDP Module 6637
USART Driver Library 2340
USART Peripheral Library 5294
USB Audio 2.0 Device Library 6754
USB Audio Device Library 6724
USB Device Layer Library 6791
USB Peripheral Library 5376
USBHS Peripheral Library 5517
Watchdog Timer Peripheral Library 5566
Watchdog Timer System Service Library 5977
Zeroconf Module 6668
Configuring the MHC 1404, 1452, 1489, 1528
Configuring the Module 6295
Iperf Module 6295
Configuring the SPI Driver 1570, 1591
Configuring the Stack 6708
connect function 6035
Considerations While Moving the Application Image 1179
Console Commands 2409, 2499
Console Device Modules 5656
Console System Service Library 5654
CONSOLE_DEVICE_HANDLE_INVALID macro 5671
Control Transfer Events 6937
Controlling the Conversion Process 3243
Controlling the Sampling Process 3243
Conversion Sequence Examples 3246
Cookies 6168
Core Functionality 2046, 3202, 3436, 5109, 5577, 5636, 5691, 6299, 6353, 6402, 6577, 6635, 6668, 6678
Core Functions 5656
COS_AUDIO macro 583
COS_CAPTURING macro 583
COS_INFORMATION macro 583
COS_LIMITED_DISCOVERABLE_MODE macro 583
COS_NETWORKING macro 583
COS_OBJECTTRANSFER macro 584
COS_POSITIONING macro 584
COS_RENDERING macro 584
COS_TELEPHONY macro 584
Counter Modification 2045
CPLD XC2C64A Driver Library 1558
CPLD_DEVICE_CONFIGURATION enumeration 1566
CPLD_GFX_CONFIGURATION enumeration 1567
CPLD_SPI_CONFIGURATION enumeration 1567
CPLDGetDeviceConfiguration function 1560
CPLDGetGraphicsConfiguration function 1561
CPLDGetSPIConfiguration function 1562
CPLDInitialize function 1562
CPLDSetGraphicsConfiguration function 1563
CPLDSetSPIFlashConfiguration function 1564
CPLDSetWiFiConfiguration function 1564
CPLDSetZigBeeConfiguration function 1565
CRC_08_GEN_POLY macro 1206
CRC_16_GEN_POLY macro 1207
CRC_32_GEN_POLY macro 1207

CRC_CCITT_GEN_POLY macro 1207
Creating a New Graphics Driver 1675
Creating Your Own USB Device 6701
Critical Section Operation 3204
Critical Sections 5836
CRYPT_AES_CBC_Decrypt function 1222
CRYPT_AES_CBC_Encrypt function 1223
CRYPT_AES_CTR_Encrypt function 1223
CRYPT_AES_CTX structure 1254
CRYPT_AES_DIRECT_Decrypt function 1224
CRYPT_AES_DIRECT_Encrypt function 1225
CRYPT_AES_IvSet function 1226
CRYPT_AES_KeySet function 1226
CRYPT_ECC_CTX structure 1255
CRYPT_ECC_DHE_KeyMake function 1227
CRYPT_ECC_DHE_SharedSecretMake function 1228
CRYPT_ECC_DSA_HashSign function 1229
CRYPT_ECC_DSA_HashVerify function 1229
CRYPT_ECC_Free function 1230
CRYPT_ECC_Initialize function 1231
CRYPT_ECC_KeySizeGet function 1231
CRYPT_ECC_PrivateImport function 1232
CRYPT_ECC_PublicExport function 1233
CRYPT_ECC_PublicImport function 1233
CRYPT_ECC_SignatureSizeGet function 1234
CRYPT_ERROR_StringGet function 1216
CRYPT_HMAC_CTX structure 1255
CRYPT_HMAC_DataAdd function 1243
CRYPT_HMAC_Finalize function 1244
CRYPT_HMAC_SetKey function 1244
CRYPT_HUFFMAN_Compress function 1216
CRYPT_HUFFMAN_DeCompress function 1217
CRYPT_MD5_CTX structure 1255
CRYPT_MD5_DataAdd function 1218
CRYPT_MD5_DataSizeSet function 1220
CRYPT_MD5_Finalize function 1219
CRYPT_MD5_Initialize function 1219
CRYPT_RNG_BlockGenerate function 1221
CRYPT_RNG_CTX structure 1255
CRYPT_RNG_Get function 1221
CRYPT_RNG_Initialize function 1215
CRYPT_RSA_CTX structure 1256
CRYPT_RSA_EncryptSizeGet function 1235
CRYPT_RSA_Free function 1235
CRYPT_RSA_Initialize function 1236
CRYPT_RSA_PrivateDecrypt function 1237
CRYPT_RSA_PrivateKeyDecode function 1237
CRYPT_RSA_PublicEncrypt function 1238
CRYPT_RSA_PublicKeyDecode function 1239
CRYPT_SHA_CTX structure 1256
CRYPT_SHA_DataAdd function 1245
CRYPT_SHA_DataSizeSet function 1247
CRYPT_SHA_Finalize function 1246
CRYPT_SHA_Initialize function 1246
CRYPT_SHA256_CTX structure 1256
CRYPT_SHA256_DataAdd function 1248
CRYPT_SHA256_DataSizeSet function 1250
CRYPT_SHA256_Finalize function 1248
CRYPT_SHA256_Initialize function 1249
CRYPT_SHA384_CTX structure 1257
CRYPT_SHA384_DataAdd function 1250
CRYPT_SHA384_Finalize function 1251
CRYPT_SHA384_Initialize function 1252
CRYPT_SHA512_CTX structure 1257
CRYPT_SHA512_DataAdd function 1252
CRYPT_SHA512_Finalize function 1253
CRYPT_SHA512_Initialize function 1254
CRYPT_TDES_CBC_Decrypt function 1240
CRYPT_TDES_CBC_Encrypt function 1240
CRYPT_TDES_CTX structure 1257
CRYPT_TDES_IvSet function 1241
CRYPT_TDES_KeySet function 1242
Crypto Library Help 1211
crypto.h 1258
csr.h 1098
CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN macro 953
CSR_I2S_STREAM_CFG_KEY_AUDIO_ATTEN_ENABLE macro 953
CSR_I2S_STREAM_CFG_KEY_BITS_PER_SAMPLE macro 953
CSR_I2S_STREAM_CFG_KEY_CROP_ENABLE macro 954
CSR_I2S_STREAM_CFG_KEY_JUSTIFY_DELAY macro 954
CSR_I2S_STREAM_CFG_KEY_JUSTIFY_FORMAT macro 954
CSR_I2S_STREAM_CFG_KEY_JUSTIFY_RESOLUTION macro 954
CSR_I2S_STREAM_CFG_KEY_MASTER macro 954
CSR_I2S_STREAM_CFG_KEY_MCLK macro 955
CSR_I2S_STREAM_CFG_KEY_POLARITY macro 955
CSR_I2S_STREAM_CFG_KEY_RX_START_SAMPLE macro 955
CSR_I2S_STREAM_CFG_KEY_SYNC_RATE macro 955
CSR_I2S_STREAM_CFG_KEY_TX_START_SAMPLE macro 955
CSR_MAX_HQ_PACKET_LEN macro 956
CSR_MSG_TYPE_GETREQ macro 956
CSR_MSG_TYPE_GETRESP macro 956
CSR_MSG_TYPE_SETREQ macro 956
CSR_PCM_STREAM_CFG_KEY_LSB_FIRST macro 956
CSR_PCM_STREAM_CFG_KEY_MANCHESTER macro 957
CSR_PCM_STREAM_CFG_KEY_MANCHESTER_SLAVE macro 957
CSR_PCM_STREAM_CFG_KEY_MASTER macro 957
CSR_PCM_STREAM_CFG_KEY_MCLK macro 957
CSR_PCM_STREAM_CFG_KEY_SHORT_SYNC macro 957
CSR_PCM_STREAM_CFG_KEY_SIGN_EXTEND macro 958
CSR_PCM_STREAM_CFG_KEY_SLOT_COUNT macro 958
CSR_PCM_STREAM_CFG_KEY_SYNC_RATE macro 958
CSR_PCM_STREAM_CFG_KEY_TX_TRISTATE macro 958
CSR_SNK_ADC macro 923
CSR_SNK_FASTPIPE macro 923
CSR_SNK_FM macro 924
CSR_SNK_I2S macro 924
CSR_SNK_L2CAP macro 924
CSR_SNK_PCM macro 924
CSR_SNK_SCO macro 924
CSR_SNK_SPDIF macro 925
CSR_SRC_ADC macro 925
CSR_SRC_FASTPIPE macro 925
CSR_SRC_FM macro 925
CSR_SRC_I2S macro 925
CSR_SRC_L2CAP macro 926
CSR_SRC_MIC macro 926

CSR_SRC_PCM macro 926
CSR_SRC_SCO macro 926
CSR_SRC_SPDIF macro 926
CSR_VARID_CACHED_TEMPERATURE macro 927
CSR_VARID_CAPABILITY_DOWNLOAD_INDICATION macro 958
CSR_VARID_CREATE_OPERATOR_C macro 959
CSR_VARID_CREATE_OPERATOR_P macro 959
CSR_VARID_DESTROY_OPERATORS macro 959
CSR_VARID_DSPMANAGER_CONFIG_REQUEST macro 959
CSR_VARID_ENABLE_SCO_STREAMS macro 927
CSR_VARID_MAP_SCO_AUDIO macro 927
CSR_VARID_MESSAGE_FROM_OPERATOR macro 959
CSR_VARID_PIO macro 927
CSR_VARID_PIO_DIRECTION_MASK macro 927
CSR_VARID_PIO_PROTECT_MASK macro 928
CSR_VARID_RSSI_ACL macro 928
CSR_VARID_START_OPERATORS macro 960
CSR_VARID_STOP_OPERATORS macro 960
CSR_VARID_STREAM_CLOSE_SINK macro 928
CSR_VARID_STREAM_CLOSE_SOURCE macro 928
CSR_VARID_STREAM_CONFIGURE macro 928
CSR_VARID_STREAM_CONNECT macro 929
CSR_VARID_STREAM_DRAINED_NOTIFICATION macro 960
CSR_VARID_STREAM_GET_SINK macro 929
CSR_VARID_STREAM_GET_SOURCE macro 929
CSR_VARID_STREAM_TRANSFORM_DISCONNECT macro 960
CSTATE_CLOSED macro 730
CSTATE_FREE macro 730
CSTATE_OPEN macro 730
CSTATE_WAIT_CONFIG macro 731
CSTATE_WAIT_CONFIG_REQ macro 731
CSTATE_WAIT_CONFIG_RSP macro 731
CSTATE_WAIT_CONNECT macro 731
CSTATE_WAIT_CONNECT_RSP macro 731
CSTATE_WAIT_DISCONNECT macro 732
CTMU Peripheral Library 3652
CTMU Setup 3658
CTYPE_CL macro 732
CTYPE_CO macro 732
Custom Control Object 2738
CVREF Setup 3615
CYAN macro 2653

D

DARKGRAY macro 2653
DARKORANGE macro 2653
Data Transfer 6929, 7154
Data Types and Constants 2642, 2885
DCRH macro 963
ddns.h 6156
ddns_config.h 6156
DDNS_MODULE_CONFIG structure 6153
DDNS_POINTERS structure 6153
DDNS_SERVICES enumeration 6154
DDNS_STATUS enumeration 6154
DDR_CMD_IDLE_NOP macro 3928
DDR_CMD_LOAD_MODE macro 3929
DDR_CMD_PRECHARGE_ALL macro 3929
DDR_CMD_REFRESH macro 3930
DDR_HOST_CMD_REG enumeration 3931
DDR_MODULE_ID enumeration 3932
DDR_PHY_DDR_TYPE enumeration 3928
DDR_PHY_DRIVE_STRENGTH enumeration 3929
DDR_PHY_ODT enumeration 3929
DDR_PHY_PREAMBLE_DLY enumeration 3930
DDR_PHY_SCL_BURST_MODE enumeration 3930
DDR_PHY_SCL_DELAY enumeration 3930
DDR_TARGET enumeration 3932
DE_BOOL macro 842
DE_INT macro 842
DE_STRING macro 842
DE_STRING2 macro 843
DE_UINT macro 843
DE_UINT16 macro 843
DE_URL macro 843
DE_UUID128 macro 844
DE_UUID16 macro 844
DE_UUID32 macro 844
Deadman Timer Peripheral Library 3684
Debug Messages and Error Reporting 5674
Debug System Initialization 5674
Debug System Service Library 5672
Decoder Libraries Help 1260
decoder_aac.h 1269
decoder_mp3_advanced.h 1290
decoder_mp3_microaptiv.h 1291
DECODER_OPUS_SUPPORT_H macro 1305
decoder_wma.h 1335
DecoderEventHandlerCB type 1288
DEEP_SLEEP_EVENT enumeration 4886
DEEP_SLEEP_GPR enumeration 4886
DEEP_SLEEP_MODULE enumeration 4888
DEEP_SLEEP_WAKE_UP_EVENT enumeration 4888
Delays 5941
Demonstration 6668
Demonstration Application 1176
Demonstration Application Logic 6714
Descriptor Table Example 3990
Detecting Device Attach 7007, 7105, 7139
DEVCON_ALT_CLOCK_TARGET enumeration 3730
DEVCON_DEBUG_PERIPHERAL enumeration 3731
DEVCON_ECC_CONFIG enumeration 3731
DEVCON_MODULE_ID enumeration 3733
DEVCON_MPLL_OUTPUT_DIVIDER enumeration 3733
DEVCON_MPLL_VREF_CONTROL enumeration 3734
DEVCON_REGISTER_SET enumeration 3732
DEVCON_USB_SLEEP_MODE enumeration 3732
Development Information (Advance Information) 6002
Device Control Peripheral Library 3700
Device Control System Service Library 5690
Device Endpoint Operations 2204
Device Layer Control Transfers 6783
Device Layer Task Routines 6779
Device Stack Configuration 6717
DHCP Module 6064
DHCP Server Module 6078

dhcp.h 6077
dhcp_config.h 6078
dhcps.h 6092
dhcps_config.h 6092
DHCPv6 Module 6093
dhcpv6.h 6111
dhcpv6_config.h 6112
Digital Meter Object 2742
Direct Memory Access (DMA) System Service Library 5716
Disabling DMA SPI 2400
Discovering the Board 6006
Display Background Management 4138
Display Functions 2593
Display Signal Polarity and Timing Management 4137
DMA Mode 5110
DMA Peripheral Library Help 3736
DMA_ADDRESS_OFFSET_TYPE enumeration 3848
DMA_CHANNEL enumeration 3848
DMA_CHANNEL_ADDRESSING_MODE enumeration 3849
DMA_CHANNEL_COLLISION enumeration 3849
DMA_CHANNEL_DATA_SIZE enumeration 3850
DMA_CHANNEL_INT_SOURCE enumeration 3861
DMA_CHANNEL_PRIORITY enumeration 3850
DMA_CHANNEL_TRANSFER_DIRECTION enumeration 3851
DMA_CHANNEL_TRIGGER_TYPE enumeration 3851
DMA_CRC_BIT_ORDER enumeration 3852
DMA_CRC_BYTE_ORDER enumeration 3852
DMA_CRC_TYPE enumeration 3852
DMA_DESTINATION_ADDRESSING_MODE enumeration 3853
DMA_INT_TYPE enumeration 3853
DMA_ISR_TASK enumeration 1698
DMA_MODULE_ID enumeration 3854
DMA_PATTERN_LENGTH enumeration 3855
DMA_PING_PONG_MODE enumeration 3855
DMA_SOURCE_ADDRESSING_MODE enumeration 3855
DMA_TRANSFER_MODE enumeration 3856
DMA_TRIGGER_SOURCE enumeration 3856
DMT_MODULE_ID enumeration 3698
DNS Module 6113
DNS Server Module 6136
dns.h 6134
dns_config.h 6135
dnss.h 6147
dnss_config.h 6148
Double Buffering Functions 2632
Double-Buffering Features 2575
Driver Device Mode Client Functions 2190
Driver General Client Functions 2184
Driver Host Mode Client Functions 2186
Driver Host Root Hub Interface 2189
Driver Host USB Root Hub Port Interface 2188
Driver Libraries Help 1336
Driver Library Overview 1336
Driver Tasks Routine 2339
driver.h 1352
driver_common.h 1352
DRV_ADC_Deinitialize function 1354
DRV_ADC_Initialize function 1354
DRV_ADC_SamplesAvailable function 1355
DRV_ADC_SamplesRead function 1355
DRV_ADC_Start function 1356
DRV_ADC_Stop function 1356
drv_adc10bit.h 2109
DRV_ADC10BIT_CALIBRATION_DELAY macro 2094
DRV_ADC10BIT_CALIBRATION_INSET macro 2094
DRV_ADC10BIT_CLIENTS_NUMBER macro 2095
drv_adc10bit_config_template.h 2110
DRV_ADC10BIT_INDEX macro 2095
DRV_ADC10BIT_INSTANCES_NUMBER macro 2095
DRV_ADC10BIT_INTERRUPT_MODE macro 2096
DRV_ADC10BIT_MODULE_ID enumeration 2106
DRV_ADC10BIT_SAMPLE_POINTS macro 2096
DRV_ADC10BIT_TOUCH_DIAMETER macro 2096
DRV_ADCx_Close function 1357
DRV_ADCx_Open function 1357
drv_ak4384.h 1439
DRV_AK4384_AUDIO_DATA_FORMAT enumeration 1432
DRV_AK4384_BCLK_BIT_CLK_DIVISOR macro 1403
DRV_AK4384_BUFFER_EVENT enumeration 1432
DRV_AK4384_BUFFER_EVENT_HANDLER type 1433
DRV_AK4384_BUFFER_HANDLE type 1434
DRV_AK4384_BUFFER_HANDLE_INVALID macro 1437
DRV_AK4384_BufferAddWrite function 1423
DRV_AK4384_BufferCombinedQueueSizeGet function 1426
DRV_AK4384_BufferEventHandlerSet function 1425
DRV_AK4384_BufferProcessedSizeGet function 1427
DRV_AK4384_BufferQueueFlush function 1431
DRV_AK4384_CHANNEL enumeration 1434
DRV_AK4384_ChannelOutputInvertDisable function 1412
DRV_AK4384_ChannelOutputInvertEnable function 1413
DRV_AK4384_CLIENTS_NUMBER macro 1401
DRV_AK4384_Close function 1412
DRV_AK4384_COMMAND_EVENT_HANDLER type 1435
DRV_AK4384_CommandEventHandlerSet function 1429
drv_ak4384_config_template.h 1441
DRV_AK4384_CONTROL_CLOCK macro 1401
DRV_AK4384_COUNT macro 1437
DRV_AK4384_DEEMPHASIS_FILTER enumeration 1435
DRV_AK4384_DeEmphasisFilterSet function 1414
DRV_AK4384_Deinitialize function 1408
DRV_AK4384_INDEX_0 macro 1438
DRV_AK4384_INDEX_1 macro 1438
DRV_AK4384_INDEX_2 macro 1438
DRV_AK4384_INDEX_3 macro 1438
DRV_AK4384_INDEX_4 macro 1439
DRV_AK4384_INDEX_5 macro 1439
DRV_AK4384_INIT structure 1436
DRV_AK4384_Initialize function 1407
DRV_AK4384_INPUT_REFCLOCK macro 1402
DRV_AK4384_INSTANCES_NUMBER macro 1402
DRV_AK4384_MCLK_MODE enumeration 1436
DRV_AK4384_MCLK_SAMPLE_FREQ_MULTPLIER macro 1403
DRV_AK4384_MuteOff function 1414
DRV_AK4384_MuteOn function 1415
DRV_AK4384_Open function 1411
DRV_AK4384_SamplingRateGet function 1416

DRV_AK4384_SamplingRateSet function 1416
DRV_AK4384_SlowRollOffFilterDisable function 1417
DRV_AK4384_SlowRollOffFilterEnable function 1418
DRV_AK4384_Status function 1409
DRV_AK4384_Tasks function 1410
DRV_AK4384_TIMER_DRIVER_MODULE_INDEX macro 1402
DRV_AK4384_TIMER_PERIOD macro 1403
DRV_AK4384_VersionGet function 1430
DRV_AK4384_VersionStrGet function 1430
DRV_AK4384_VolumeGet function 1418
DRV_AK4384_VolumeSet function 1419
DRV_AK4384_ZERO_DETECT_MODE enumeration 1437
DRV_AK4384_ZeroDetectDisable function 1420
DRV_AK4384_ZeroDetectEnable function 1420
DRV_AK4384_ZeroDetectInvertDisable function 1421
DRV_AK4384_ZeroDetectInvertEnable function 1422
DRV_AK4384_ZeroDetectModeSet function 1422
drv_ak4642.h 1480
DRV_AK4642_AUDIO_DATA_FORMAT enumeration 1475
DRV_AK4642_BCLK_BIT_CLK_DIVISOR macro 1450
DRV_AK4642_BUFFER_EVENT enumeration 1475
DRV_AK4642_BUFFER_EVENT_HANDLER type 1476
DRV_AK4642_BUFFER_HANDLE type 1477
DRV_AK4642_BUFFER_HANDLE_INVALID macro 1473
DRV_AK4642_BufferAddRead function 1467
DRV_AK4642_BufferAddWrite function 1465
DRV_AK4642_BufferAddWriteRead function 1468
DRV_AK4642_BufferEventHandlerSet function 1469
DRV_AK4642_CHANNEL enumeration 1477
DRV_AK4642_CLIENTS_NUMBER macro 1451
DRV_AK4642_Close function 1459
DRV_AK4642_COMMAND_EVENT_HANDLER type 1478
DRV_AK4642_CommandEventHandlerSet function 1471
drv_ak4642_config_template.h 1481
DRV_AK4642_COUNT macro 1474
DRV_AK4642_Deinitialize function 1456
DRV_AK4642_INDEX_0 macro 1474
DRV_AK4642_INDEX_1 macro 1474
DRV_AK4642_INDEX_2 macro 1474
DRV_AK4642_INDEX_3 macro 1475
DRV_AK4642_INDEX_4 macro 1475
DRV_AK4642_INDEX_5 macro 1475
DRV_AK4642_INIT structure 1478
DRV_AK4642_Initialize function 1455
DRV_AK4642_INPUT_REFCLOCK macro 1451
DRV_AK4642_INSTANCES_NUMBER macro 1451
DRV_AK4642_INT_EXT_MIC enumeration 1479
DRV_AK4642_IntExtMicSet function 1464
DRV_AK4642_MCLK_SAMPLE_FREQ_MULTPLIER macro 1451
DRV_AK4642_MCLK_SOURCE macro 1452
DRV_AK4642_MONO_STEREO_MIC enumeration 1479
DRV_AK4642_MonoStereoMicSet function 1465
DRV_AK4642_MuteOff function 1460
DRV_AK4642_MuteOn function 1461
DRV_AK4642_Open function 1458
DRV_AK4642_SamplingRateGet function 1462
DRV_AK4642_SamplingRateSet function 1462
DRV_AK4642_Status function 1457
DRV_AK4642_Tasks function 1458
DRV_AK4642_VersionGet function 1472
DRV_AK4642_VersionStrGet function 1473
DRV_AK4642_VolumeGet function 1463
DRV_AK4642_VolumeSet function 1463
drv_ak4953.h 1518
DRV_AK4953_AUDIO_DATA_FORMAT enumeration 1511
DRV_AK4953_BCLK_BIT_CLK_DIVISOR macro 1487
DRV_AK4953_BUFFER_EVENT enumeration 1511
DRV_AK4953_BUFFER_EVENT_HANDLER type 1511
DRV_AK4953_BUFFER_HANDLE type 1512
DRV_AK4953_BUFFER_HANDLE_INVALID macro 1515
DRV_AK4953_BufferAddRead function 1509
DRV_AK4953_BufferAddWrite function 1503
DRV_AK4953_BufferAddWriteRead function 1505
DRV_AK4953_BufferEventHandlerSet function 1498
DRV_AK4953_CHANNEL enumeration 1517
DRV_AK4953_CLIENTS_NUMBER macro 1487
DRV_AK4953_Close function 1495
DRV_AK4953_COMMAND_EVENT_HANDLER type 1513
DRV_AK4953_CommandEventHandlerSet function 1497
drv_ak4953_config_template.h 1519
DRV_AK4953_COUNT macro 1515
DRV_AK4953_Deinitialize function 1494
DRV_AK4953_DIGITAL_BLOCK_CONTROL enumeration 1514
DRV_AK4953_INDEX_0 macro 1515
DRV_AK4953_INDEX_1 macro 1516
DRV_AK4953_INDEX_2 macro 1516
DRV_AK4953_INDEX_3 macro 1516
DRV_AK4953_INDEX_4 macro 1516
DRV_AK4953_INDEX_5 macro 1516
DRV_AK4953_INIT structure 1514
DRV_AK4953_Initialize function 1493
DRV_AK4953_INPUT_REFCLOCK macro 1488
DRV_AK4953_INSTANCES_NUMBER macro 1488
DRV_AK4953_INT_EXT_MIC enumeration 1517
DRV_AK4953_IntExtMicSet function 1510
DRV_AK4953_MCLK_SAMPLE_FREQ_MULTPLIER macro 1488
DRV_AK4953_MCLK_SOURCE macro 1488
DRV_AK4953_MONO_STEREO_MIC enumeration 1517
DRV_AK4953_MonoStereoMicSet function 1510
DRV_AK4953_MuteOff function 1507
DRV_AK4953_MuteOn function 1507
DRV_AK4953_Open function 1494
DRV_AK4953_QUEUE_DEPTH_COMBINED macro 1489
DRV_AK4953_SamplingRateGet function 1500
DRV_AK4953_SamplingRateSet function 1499
DRV_AK4953_Status function 1501
DRV_AK4953_Tasks function 1496
DRV_AK4953_VersionGet function 1501
DRV_AK4953_VersionStrGet function 1502
DRV_AK4953_VolumeGet function 1503
DRV_AK4953_VolumeSet function 1508
drv_ak7755.h 1556
DRV_AK7755_BCLK_BIT_CLK_DIVISOR macro 1526
DRV_AK7755_BICK_FS_FORMAT enumeration 1550
DRV_AK7755_BUFFER_EVENT enumeration 1550
DRV_AK7755_BUFFER_EVENT_HANDLER type 1550

DRV_AK7755_BUFFER_HANDLE type 1551
DRV_AK7755_BUFFER_HANDLE_INVALID macro 1548
DRV_AK7755_BufferAddRead function 1542
DRV_AK7755_BufferAddWrite function 1543
DRV_AK7755_BufferAddWriteRead function 1545
DRV_AK7755_BufferEventHandlerSet function 1535
DRV_AK7755_CHANNEL enumeration 1552
DRV_AK7755_CLIENTS_NUMBER macro 1526
DRV_AK7755_Close function 1531
DRV_AK7755_COMMAND_EVENT_HANDLER type 1552
DRV_AK7755_CommandEventHandlerSet function 1537
drv_ak7755_config_template.h 1557
DRV_AK7755_COUNT macro 1548
DRV_AK7755_DAC_INPUT_FORMAT enumeration 1553
DRV_AK7755_Deinitialize function 1532
DRV_AK7755_DSP_DIN1_INPUT_FORMAT enumeration 1553
DRV_AK7755_DSP_DOUT1_OUTPUT_FORMAT enumeration 1553
DRV_AK7755_DSP_DOUT4_OUTPUT_FORMAT enumeration 1554
DRV_AK7755_DSP_PROGRAM enumeration 1554
DRV_AK7755_INDEX_0 macro 1548
DRV_AK7755_INDEX_1 macro 1549
DRV_AK7755_INDEX_2 macro 1549
DRV_AK7755_INDEX_3 macro 1549
DRV_AK7755_INDEX_4 macro 1549
DRV_AK7755_INDEX_5 macro 1549
DRV_AK7755_INIT structure 1554
DRV_AK7755_Initialize function 1533
DRV_AK7755_INPUT_REFCLOCK macro 1527
DRV_AK7755_INSTANCES_NUMBER macro 1527
DRV_AK7755_INT_EXT_MIC enumeration 1555
DRV_AK7755_IntExtMicSet function 1545
DRV_AK7755_LRCK_IF_FORMAT enumeration 1555
DRV_AK7755_MCLK_SAMPLE_FREQ_MULTPLIER macro 1527
DRV_AK7755_MCLK_SOURCE macro 1528
DRV_AK7755_MONO_STEREO_MIC enumeration 1555
DRV_AK7755_MonoStereoMicSet function 1545
DRV_AK7755_MuteOff function 1546
DRV_AK7755_MuteOn function 1547
DRV_AK7755_Open function 1534
DRV_AK7755_SamplingRateGet function 1538
DRV_AK7755_SamplingRateSet function 1538
DRV_AK7755_Status function 1539
DRV_AK7755_Tasks function 1535
DRV_AK7755_VersionGet function 1540
DRV_AK7755_VersionStrGet function 1540
DRV_AK7755_VolumeGet function 1541
DRV_AK7755_VolumeSet function 1542
drv_ar1021.h 2128
DRV_AR1021_CALIBRATION_DELAY macro 2113
DRV_AR1021_CALIBRATION_INSET macro 2113
DRV_AR1021_CLIENTS_NUMBER macro 2114
DRV_AR1021_INDEX macro 2114
DRV_AR1021_INSTANCES_NUMBER macro 2114
DRV_AR1021_INTERRUPT_MODE macro 2114
DRV_AR1021_SAMPLE_POINTS macro 2115
DRV_AR1021_TOUCH_DIAMETER macro 2115
drv_camera.h 1364
DRV_CAMERA_Close function 1358
DRV_CAMERA_Deinitialize function 1359
DRV_CAMERA_INDEX_0 macro 1363
DRV_CAMERA_INDEX_1 macro 1364
DRV_CAMERA_INDEX_COUNT macro 1364
DRV_CAMERA_INIT structure 1362
DRV_CAMERA_Initialize function 1359
DRV_CAMERA_INTERRUPT_PORT_REMAP structure 1363
DRV_CAMERA_Open function 1360
drv_camera_ovm7690.h 1387
DRV_CAMERA_OVM7690_CLIENT_OBJ structure 1381
DRV_CAMERA_OVM7690_CLIENT_STATUS enumeration 1382
DRV_CAMERA_OVM7690_Close function 1374
DRV_CAMERA_OVM7690_Deinitialize function 1371
DRV_CAMERA_OVM7690_ERROR enumeration 1382
DRV_CAMERA_OVM7690_FrameBufferAddressSet function 1374
DRV_CAMERA_OVM7690_FrameRectSet function 1375
DRV_CAMERA_OVM7690_HsyncEventHandler function 1378
DRV_CAMERA_OVM7690_INDEX_0 macro 1385
DRV_CAMERA_OVM7690_INDEX_1 macro 1385
DRV_CAMERA_OVM7690_INIT structure 1382
DRV_CAMERA_OVM7690_Initialize function 1370
DRV_CAMERA_OVM7690_OBJ structure 1383
DRV_CAMERA_OVM7690_Open function 1373
DRV_CAMERA_OVM7690_RECT structure 1384
DRV_CAMERA_OVM7690_REG12_OP_FORMAT enumeration 1385
DRV_CAMERA_OVM7690_REG12_SOFT_RESET macro 1386
DRV_CAMERA_OVM7690_RegisterSet function 1372
DRV_CAMERA_OVM7690_SCCB_READ_ID macro 1386
DRV_CAMERA_OVM7690_SCCB_WRITE_ID macro 1386
DRV_CAMERA_OVM7690_Start function 1376
DRV_CAMERA_OVM7690_Stop function 1377
DRV_CAMERA_OVM7690_Tasks function 1373
DRV_CAMERA_OVM7690_VsyncEventHandler function 1379
DRV_CAMERA_Reinitialize function 1361
DRV_CAMERA_Status function 1362
DRV_CAMERA_Tasks function 1362
DRV_CAN_ChannelMessageReceive function 1388
DRV_CAN_ChannelMessageTransmit function 1389
DRV_CAN_Close function 1390
DRV_CAN_Deinitialize function 1390
DRV_CAN_Initialize function 1390
DRV_CAN_Open function 1391
DRV_CLIENT_STATUS enumeration 1348
DRV_CMP_Initialize function 1558
DRV_CONFIG_NOT_SUPPORTED macro 1350
DRV_DYNAMIC_BUILD macro 1735
drv_enc28j60.h 1587
DRV_ENC28J60_CLIENT_INSTANCES macro 1571
DRV_ENC28J60_Close function 1578
drv_enc28j60_config_template.h 1588
DRV_ENC28J60_ConfigGet function 1578
DRV_ENC28J60_Configuration structure 1585
DRV_ENC28J60_Deinitialize function 1575
DRV_ENC28J60_EventAcknowledge function 1584
DRV_ENC28J60_EventMaskSet function 1584
DRV_ENC28J60_EventPendingGet function 1585
DRV_ENC28J60_Initialize function 1575
DRV_ENC28J60_INSTANCES_NUMBER macro 1572

DRV_ENC28J60_LinkCheck function 1579
DRV_ENC28J60_MACObject variable 1587
DRV_ENC28J60_MDIX_TYPE enumeration 1586
DRV_ENC28J60_Open function 1579
DRV_ENC28J60_PacketRx function 1582
DRV_ENC28J60_PacketTx function 1583
DRV_ENC28J60_ParametersGet function 1580
DRV_ENC28J60_PowerMode function 1580
DRV_ENC28J60_Process function 1575
DRV_ENC28J60_RegisterStatisticsGet function 1581
DRV_ENC28J60_Reinitialize function 1576
DRV_ENC28J60_RxFilterHashTableEntrySet function 1583
DRV_ENC28J60_SetMacCtrlInfo function 1576
DRV_ENC28J60_StackInitialize function 1577
DRV_ENC28J60_StatisticsGet function 1581
DRV_ENC28J60_Status function 1582
DRV_ENC28J60_Tasks function 1577
drv_encx24j600.h 1606
DRV_ENCX24J600_Close function 1597
DRV_ENCX24J600_ConfigGet function 1598
DRV_ENCX24J600_Configuration structure 1605
DRV_ENCX24J600_Deinitialize function 1594
DRV_ENCX24J600_EventAcknowledge function 1603
DRV_ENCX24J600_EventMaskSet function 1604
DRV_ENCX24J600_EventPendingGet function 1604
DRV_ENCX24J600_Initialize function 1594
DRV_ENCX24J600_LinkCheck function 1598
DRV_ENCX24J600_MDIX_TYPE enumeration 1606
DRV_ENCX24J600_Open function 1599
DRV_ENCX24J600_PacketRx function 1602
DRV_ENCX24J600_PacketTx function 1603
DRV_ENCX24J600_ParametersGet function 1599
DRV_ENCX24J600_PowerMode function 1600
DRV_ENCX24J600_Process function 1597
DRV_ENCX24J600_RegisterStatisticsGet function 1600
DRV_ENCX24J600_Reinitialize function 1595
DRV_ENCX24J600_RxFilterHashTableEntrySet function 1602
DRV_ENCX24J600_SetMacCtrlInfo function 1596
DRV_ENCX24J600_StackInitialize function 1596
DRV_ENCX24J600_StatisticsGet function 1601
DRV_ENCX24J600_Status function 1601
DRV_ENCX24J600_Tasks function 1595
drv_ethmac.h 1628
DRV_ETHMAC_CLIENTS_NUMBER macro 1611
drv_ethmac_config.h 1629
DRV_ETHMAC_INDEX macro 1611
DRV_ETHMAC_INDEX_0 macro 1627
DRV_ETHMAC_INDEX_1 macro 1627
DRV_ETHMAC_INDEX_COUNT macro 1627
DRV_ETHMAC_INSTANCES_NUMBER macro 1611
DRV_ETHMAC_INTERRUPT_MODE macro 1612
DRV_ETHMAC_INTERRUPT_SOURCE macro 1612
DRV_ETHMAC_PERIPHERAL_ID macro 1612
DRV_ETHMAC_PIC32MACClose function 1615
DRV_ETHMAC_PIC32MACConfigGet function 1620
DRV_ETHMAC_PIC32MACDeinitialize function 1615
DRV_ETHMAC_PIC32MACEventAcknowledge function 1623
DRV_ETHMAC_PIC32MACEventMaskSet function 1624
DRV_ETHMAC_PIC32MACEventPendingGet function 1625
DRV_ETHMAC_PIC32MACInitialize function 1616
DRV_ETHMAC_PIC32MACLinkCheck function 1616
DRV_ETHMAC_PIC32MACOpen function 1617
DRV_ETHMAC_PIC32MACPacketRx function 1621
DRV_ETHMAC_PIC32MACPacketTx function 1623
DRV_ETHMAC_PIC32MACParametersGet function 1617
DRV_ETHMAC_PIC32MACPowerMode function 1618
DRV_ETHMAC_PIC32MACProcess function 1618
DRV_ETHMAC_PIC32MACRegisterStatisticsGet function 1620
DRV_ETHMAC_PIC32MACReinitialize function 1621
DRV_ETHMAC_PIC32MACRxFilterHashTableEntrySet function 1622
DRV_ETHMAC_PIC32MACStatisticsGet function 1619
DRV_ETHMAC_PIC32MACStatus function 1619
DRV_ETHMAC_PIC32MACTasks function 1626
DRV_ETHMAC_POWER_STATE macro 1612
DRV_ETHMAC_Tasks_ISR function 1626
drv_ethphy.h 1663
DRV_ETHPHY_CLIENT_STATUS enumeration 1655
DRV_ETHPHY_ClientOperationAbort function 1646
DRV_ETHPHY_ClientOperationResult function 1646
DRV_ETHPHY_CLIENTS_NUMBER macro 1631
DRV_ETHPHY_ClientStatus function 1642
DRV_ETHPHY_Close function 1643
drv_ethphy_config.h 1665
DRV_ETHPHY_CONFIG_FLAGS enumeration 1660
DRV_ETHPHY_Deinitialize function 1637
DRV_ETHPHY_HWConfigFlagsGet function 1641
DRV_ETHPHY_INDEX macro 1632
DRV_ETHPHY_INDEX_0 macro 1658
DRV_ETHPHY_INDEX_1 macro 1659
DRV_ETHPHY_INDEX_COUNT macro 1659
DRV_ETHPHY_INIT structure 1655
DRV_ETHPHY_Initialize function 1636
DRV_ETHPHY_INSTANCES_NUMBER macro 1632
DRV_ETHPHY_LINK_STATUS enumeration 1660
DRV_ETHPHY_LinkStatusGet function 1659
DRV_ETHPHY_NEG_DONE_TMO macro 1633
DRV_ETHPHY_NEG_INIT_TMO macro 1633
DRV_ETHPHY_NEGOTIATION_RESULT structure 1656
DRV_ETHPHY_NegotiationIsComplete function 1644
DRV_ETHPHY_NegotiationResultGet function 1638
DRV_ETHPHY_OBJECT structure 1661
DRV_ETHPHY_OBJECT_BASE structure 1662
DRV_ETHPHY_OBJECT_BASE_TYPE structure 1662
DRV_ETHPHY_Open function 1644
DRV_ETHPHY_PERIPHERAL_ID macro 1632
DRV_ETHPHY_PhyAddressGet function 1638
DRV_ETHPHY_Reinitialize function 1639
DRV_ETHPHY_Reset function 1645
DRV_ETHPHY_RESET_CLR_TMO macro 1633
DRV_ETHPHY_RestartNegotiation function 1645
DRV_ETHPHY_Setup function 1642
DRV_ETHPHY_SETUP structure 1656
DRV_ETHPHY_SMIClockSet function 1648
DRV_ETHPHY_SMIRead function 1649
DRV_ETHPHY_SMIScanDataGet function 1650
DRV_ETHPHY_SMIScanStart function 1649

DRV_ETHPHY_SMIScanStatusGet function 1647
DRV_ETHPHY_SMIScanStop function 1648
DRV_ETHPHY_SMIStatus function 1650
DRV_ETHPHY_SMIWrite function 1651
DRV_ETHPHY_Status function 1640
DRV_ETHPHY_Tasks function 1640
DRV_ETHPHY_VENDOR_MDIX_CONFIGURE type 1657
DRV_ETHPHY_VENDOR_MII_CONFIGURE type 1657
DRV_ETHPHY_VENDOR_SMI_CLOCK_GET type 1658
DRV_ETHPHY_VENDOR_WOL_CONFIGURE type 1661
DRV_ETHPHY_VendorDataGet function 1651
DRV_ETHPHY_VendorDataSet function 1652
DRV_ETHPHY_VendorSMIReadResultGet function 1653
DRV_ETHPHY_VendorSMIReadStart function 1653
DRV_ETHPHY_VendorSMIWriteStart function 1654
drv_flash.h 1672
DRV_FLASH_ErasePage function 1666
DRV_FLASH_GetPageSize function 1667
DRV_FLASH_GetRowSize function 1667
DRV_FLASH_INDEX_0 macro 1671
DRV_FLASH_Initialize function 1668
DRV_FLASH_IsBusy function 1668
DRV_FLASH_Open function 1669
DRV_FLASH_PAGE_SIZE macro 1672
DRV_FLASH_ROW_SIZE macro 1672
DRV_FLASH_WriteQuadWord function 1669
DRV_FLASH_WriteRow function 1670
DRV_FLASH_WriteWord function 1671
DRV_GFX_CONFIG_LCC_EXTERNAL_MEMORY macro 1676
DRV_GFX_CONFIG_LCC_INTERNAL_MEMORY macro 1676
DRV_GFX_CONFIG_LCC_PALETTE macro 1677
DRV_GFX_GLCD_BackgroundColorSet function 1689
DRV_GFX_GLCD_BarFill function 1682
DRV_GFX_GLCD_Close function 1689
DRV_GFX_GLCD_CursorImageSet function 1683
DRV_GFX_GLCD_CursorPaletteSet function 1683
DRV_GFX_GLCD_CursorSetPosition function 1690
DRV_GFX_GLCD_GammaPaletteSet function 1684
DRV_GFX_GLCD_Initialize function 1684
DRV_GFX_GLCD_InterfaceSet function 1685
DRV_GFX_GLCD_LayerFrameBufferSet function 1690
DRV_GFX_GLCD_LayerModeSet function 1691
DRV_GFX_GLCD_MaxXGet function 1685
DRV_GFX_GLCD_MaxYGet function 1686
DRV_GFX_GLCD_Open function 1686
DRV_GFX_GLCD_PixelArrayPut function 1687
DRV_GFX_GLCD_PixelPut function 1688
DRV_GFX_GLCD_SetColor function 1688
drv_gfx_lcc.h 1722
DRV_GFX_LCC_AlphaBlendWindow function 1693
DRV_GFX_LCC_BarFill function 1695
DRV_GFX_LCC_Close function 1692
drv_gfx_lcc_config_template.h 1726
DRV_GFX_LCC_DISPLAY_WRITE_BUS_TYPE enumeration 1698
DRV_GFX_LCC_DisplayRefresh function 1692
DRV_GFX_LCC_DMA_CHANNEL_INDEX macro 1677
DRV_GFX_LCC_FB_WRITE_BUS_TYPE enumeration 1698
DRV_GFX_LCC_FrameBufferAddressSet function 1694
DRV_GFX_LCC_GetBuffer function 1693
DRV_GFX_LCC_INDEX_COUNT macro 1698
DRV_GFX_LCC_Initialize function 1693
DRV_GFX_LCC_InterfaceSet function 1694
DRV_GFX_LCC_MaxXGet function 1695
DRV_GFX_LCC_MaxYGet function 1695
DRV_GFX_LCC_Open function 1694
DRV_GFX_LCC_PixelArrayGet function 1696
DRV_GFX_LCC_PixelArrayPut function 1696
DRV_GFX_LCC_PixelPut function 1696
DRV_GFX_LCC_SetColor function 1692
DRV_GFX_LCC_SetPage function 1693
DRV_GFX_LCC_Tasks function 1694
DRV_GFX_LCC_VSYNC_CallbackSet function 1697
DRV_GFX_LCC_VSYNC_GetState function 1697
drv_gfx_otm2201a.h 1725
DRV_GFX_OTM2201A_AddressSet function 1713
DRV_GFX_OTM2201A_BarFill function 1718
DRV_GFX_OTM2201A_BrightnessSet function 1714
DRV_GFX_OTM2201A_Busy function 1714
DRV_GFX_OTM2201A_Close function 1715
DRV_GFX_OTM2201A_ColorSet function 1715
DRV_GFX_OTM2201A_COMMAND structure 1721
drv_gfx_otm2201a_config_template.h 1727
DRV_GFX_OTM2201A_INDEX_COUNT macro 1721
DRV_GFX_OTM2201A_Initialize function 1716
DRV_GFX_OTM2201A_InstanceSet function 1716
DRV_GFX_OTM2201A_InterfaceSet function 1717
DRV_GFX_OTM2201A_MaxXGet function 1718
DRV_GFX_OTM2201A_MaxYGet function 1718
DRV_GFX_OTM2201A_Open function 1716
DRV_GFX_OTM2201A_PixelArrayGet function 1719
DRV_GFX_OTM2201A_PixelArrayPut function 1719
DRV_GFX_OTM2201A_PixelPut function 1720
DRV_GFX_OTM2201A_RegGet function 1714
DRV_GFX_OTM2201A_RegSet function 1717
DRV_GFX_OTM2201A_Tasks function 1717
DRV_GFX_PaletteSet function 1682
drv_gfx_s1d13517.h 1723
DRV_GFX_S1D13517_AlphaBlendWindow function 1699
DRV_GFX_S1D13517_BarFill function 1703
DRV_GFX_S1D13517_BrightnessSet function 1704
DRV_GFX_S1D13517_Close function 1700
drv_gfx_s1d13517_config_template.h 1726
DRV_GFX_S1D13517_GetReg function 1699
DRV_GFX_S1D13517_INDEX_COUNT macro 1705
DRV_GFX_S1D13517_Initialize function 1701
DRV_GFX_S1D13517_InterfaceSet function 1703
DRV_GFX_S1D13517_Layer function 1701
DRV_GFX_S1D13517_MaxXGet function 1703
DRV_GFX_S1D13517_MaxYGet function 1703
DRV_GFX_S1D13517_Open function 1701
DRV_GFX_S1D13517_PixelArrayPut function 1705
DRV_GFX_S1D13517_PixelPut function 1704
DRV_GFX_S1D13517_SetColor function 1702
DRV_GFX_S1D13517_SetInstance function 1700
DRV_GFX_S1D13517_SetPage function 1702
DRV_GFX_S1D13517_SetReg function 1700

DRV_GFX_S1D13517_Tasks function 1702
drv_gfx_ssd1289.h 1724
drv_gfx_ssd1926.h 1724
DRV_GFX_SSD1926_BarFill function 1711
DRV_GFX_SSD1926_Busy function 1706
DRV_GFX_SSD1926_Close function 1708
DRV_GFX_SSD1926_COMMAND structure 1713
drv_gfx_ssd1926_config_template.h 1727
DRV_GFX_SSD1926_GetReg function 1707
DRV_GFX_SSD1926_INDEX_COUNT macro 1712
DRV_GFX_SSD1926_Initialize function 1708
DRV_GFX_SSD1926_InterfaceSet function 1710
DRV_GFX_SSD1926_MaxXGet function 1710
DRV_GFX_SSD1926_MaxYGet function 1710
DRV_GFX_SSD1926_Open function 1708
DRV_GFX_SSD1926_PixelArrayGet function 1711
DRV_GFX_SSD1926_PixelArrayPut function 1711
DRV_GFX_SSD1926_PixelPut function 1712
DRV_GFX_SSD1926_SetColor function 1709
DRV_GFX_SSD1926_SetInstance function 1709
DRV_GFX_SSD1926_SetReg function 1707
DRV_GFX_SSD1926_Status function 1709
DRV_GFX_SSD1926_Tasks function 1710
drv_gfx_tft002.h 1726
drv_gfx_tft002_config_template.h 1727
DRV_HANDLE type 1348
DRV_HANDLE_INVALID macro 1350
drv_i2c.h 1764
DRV_I2C_ADDRESS_WIDTH enumeration 1754
drv_i2c_bb.h 1766
DRV_I2C_BB_H macro 1764
DRV_I2C_BUFFER_EVENT enumeration 1756
DRV_I2C_BUFFER_EVENT_HANDLER type 1758
DRV_I2C_BUFFER_HANDLE type 1755
DRV_I2C_BUFFER_QUEUE_SUPPORT macro 1762
DRV_I2C_BufferEventHandlerSet function 1746
DRV_I2C_BUS_ERROR_EVENT enumeration 1764
DRV_I2C_BUS_LEVEL enumeration 1755
DRV_I2C_BUS_SPEED enumeration 1755
DRV_I2C_BytesTransferred function 1748
DRV_I2C_CallBack type 1760
DRV_I2C_Close function 1745
DRV_I2C_CONFIG_BUILD_TYPE macro 1736
DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BASIC macro 1736
DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_BLOCKING macro 1736
DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_EXCLUSIVE macro 1737
DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_MASTER macro 1737
DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_NON_BLOCKING macro 1737
DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_READ macro 1738
DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_SLAVE macro 1738
DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE macro 1738
DRV_I2C_CONFIG_SUPPORT_OPERATION_MODE_WRITE_READ macro 1739
drv_i2c_config_template.h 1766
DRV_I2C_Deinitialize function 1742
DRV_I2C_FORCED_WRITE macro 1739
DRV_I2C_INDEX macro 1517
DRV_I2C_INDEX_0 macro 1760
DRV_I2C_INDEX_1 macro 1761
DRV_I2C_INDEX_2 macro 1761
DRV_I2C_INDEX_3 macro 1761
DRV_I2C_INDEX_4 macro 1762
DRV_I2C_INDEX_5 macro 1762
DRV_I2C_INIT structure 1757
DRV_I2C_Initialize function 1743
DRV_I2C_INSTANCES_NUMBER macro 1762
DRV_I2C_INTERRUPT_MODE macro 1763
DRV_I2C_MODE enumeration 1758
DRV_I2C_Open function 1745
DRV_I2C_PERPH enumeration 1759
DRV_I2C_QUEUE_DEPTH_COMBINED macro 1763
DRV_I2C_QueueFlush function 1754
DRV_I2C_Receive function 1748
DRV_I2C_SLAVE_ADDRESS_MASK type 1759
DRV_I2C_Status function 1753
DRV_I2C_Tasks function 1744
DRV_I2C_TransferStatusGet function 1752
DRV_I2C_Transmit function 1749
DRV_I2C_TransmitForced function 1751
DRV_I2C_TransmitThenReceive function 1750
drv_i2s.h 1819
DRV_I2S_AUDIO_PROTOCOL_MODE enumeration 1810
DRV_I2S_BaudSet function 1804
DRV_I2S_BUFFER_EVENT enumeration 1810
DRV_I2S_BUFFER_EVENT_HANDLER type 1811
DRV_I2S_BUFFER_HANDLE type 1812
DRV_I2S_BUFFER_HANDLE_INVALID macro 1817
DRV_I2S_BufferAddRead function 1792
DRV_I2S_BufferAddWrite function 1794
DRV_I2S_BufferAddWriteRead function 1795
DRV_I2S_BufferCombinedQueueSizeGet function 1800
DRV_I2S_BufferEventHandlerSet function 1797
DRV_I2S_BufferProcessedSizeGet function 1799
DRV_I2S_BufferQueueFlush function 1801
DRV_I2S_CLIENTS_NUMBER macro 1784
DRV_I2S_CLOCK_MODE enumeration 1812
DRV_I2S_Close function 1791
drv_i2s_config_template.h 1821
DRV_I2S_COUNT macro 1817
DRV_I2S_DATA16 structure 1813
DRV_I2S_DATA24 structure 1813
DRV_I2S_DATA32 structure 1814
DRV_I2S_Deinitialize function 1787
DRV_I2S_ERROR enumeration 1814
DRV_I2S_ErrorGet function 1805
DRV_I2S_INDEX macro 1780
DRV_I2S_INDEX_0 macro 1818
DRV_I2S_INDEX_1 macro 1818
DRV_I2S_INDEX_2 macro 1818

DRV_I2S_INDEX_3 macro 1818
DRV_I2S_INDEX_4 macro 1819
DRV_I2S_INDEX_5 macro 1819
DRV_I2S_INIT structure 1815
DRV_I2S_Initialize function 1787
DRV_I2S_INSTANCES_NUMBER macro 1781
DRV_I2S_INTERRUPT_MODE macro 1781
DRV_I2S_INTERRUPT_SOURCE_ERROR macro 1781
DRV_I2S_INTERRUPT_SOURCE_RECEIVE macro 1781
DRV_I2S_INTERRUPT_SOURCE_TRANSMIT macro 1782
DRV_I2S_MODE enumeration 1816
DRV_I2S_Open function 1791
DRV_I2S_PERIPHERAL_ID macro 1782
DRV_I2S_QUEUE_DEPTH_COMBINED macro 1784
DRV_I2S_Read function 1802
DRV_I2S_READ_ERROR macro 1817
DRV_I2S_RECEIVE_DMA_CHAINING_CHANNEL macro 1783
DRV_I2S_RECEIVE_DMA_CHANNEL macro 1782
DRV_I2S_ReceiveErrorIgnore function 1807
DRV_I2S_Status function 1789
DRV_I2S_STOP_IN_IDLE macro 1783
DRV_I2S_Tasks function 1789
DRV_I2S_TasksError function 1790
DRV_I2S_TRANSMIT_DMA_CHANNEL macro 1783
DRV_I2S_TransmitErrorIgnore function 1808
DRV_I2S_Write function 1803
DRV_I2S_WRITE_ERROR macro 1817
DRV_IC_BufferIsEmpty function 1822
DRV_IC_Capture16BitDataRead function 1823
DRV_IC_Capture32BitDataRead function 1823
DRV_IC_Initialize function 1822
DRV_IC_Start function 1824
DRV_IC_Stop function 1824
DRV_IO_BUFFER_TYPES enumeration 1349
DRV_IO_INTENT enumeration 1349
DRV_IO_ISBLOCKING macro 1350
DRV_IO_ISEXCLUSIVE macro 1351
DRV_IO_ISNONBLOCKING macro 1351
drv_mtch6301.h 2149
DRV_MTCH6301_CALIBRATION_DELAY macro 2132
DRV_MTCH6301_CALIBRATION_INSET macro 2133
DRV_MTCH6301_CLIENTS_NUMBER macro 2133
drv_mtch6301_config_template.h 2150
DRV_MTCH6301_INDEX macro 2133
DRV_MTCH6301_INSTANCES_NUMBER macro 2133
DRV_MTCH6301_INTERRUPT_MODE macro 2134
DRV_MTCH6301_SAMPLE_POINTS macro 2134
DRV_MTCH6301_TOUCH_DIAMETER macro 2134
drv_mtch6303.h 2179
DRV_MTCH6303_AddRegisterRead function 2158
DRV_MTCH6303_AddRegisterWrite function 2159
DRV_MTCH6303_BUFFER_EVENT enumeration 2172
DRV_MTCH6303_BUFFER_EVENT_HANDLER type 2172
DRV_MTCH6303_BUFFER_HANDLE type 2173
DRV_MTCH6303_BUFFER_HANDLE_INVALID macro 2171
DRV_MTCH6303_BufferEventHandlerSet function 2169
DRV_MTCH6303_CLIENT_STATUS enumeration 2173
DRV_MTCH6303_Close function 2156
DRV_MTCH6303_Deinitialize function 2154
DRV_MTCH6303_ERROR enumeration 2174
DRV_MTCH6303_ErrorGet function 2157
DRV_MTCH6303_Initialize function 2154
DRV_MTCH6303_Open function 2157
DRV_MTCH6303_Status function 2155
DRV_MTCH6303_Tasks function 2156
DRV_MTCH6303_TOUCH_AddMessageCommandWrite function 2161
DRV_MTCH6303_TOUCH_AddMessageReportRead function 2162
DRV_MTCH6303_TOUCH_AddTouchInputRead function 2164
DRV_MTCH6303_TOUCH_BUFFER_EVENT enumeration 2174
DRV_MTCH6303_TOUCH_BUFFER_EVENT_HANDLER type 2174
DRV_MTCH6303_TOUCH_BUFFER_HANDLE type 2175
DRV_MTCH6303_TOUCH_BUFFER_HANDLE_INVALID macro 2171
DRV_MTCH6303_TOUCH_BufferEventHandlerSet function 2165
DRV_MTCH6303_TOUCH_DATA structure 2175
DRV_MTCH6303_TOUCH_INPUT structure 2176
DRV_MTCH6303_TOUCH_MESSAGE structure 2176
DRV_MTCH6303_TOUCH_MESSAGE_HEADER structure 2177
DRV_MTCH6303_TOUCH_NIBBLE_0 structure 2177
DRV_MTCH6303_TOUCH_NUM_INPUTS macro 2171
DRV_MTCH6303_TOUCH_STATUS structure 2178
DRV_MTCH6303_TOUCH_Tasks function 2167
DRV_MTCH6303_TouchInputMap function 2167
DRV_MTCH6303_TouchInputRead function 2168
drv_nvm.h 1862
DRV_NVM_AddressGet function 1854
DRV_NVM_BUFFER_OBJECT_NUMBER macro 1835
DRV_NVM_CLIENTS_NUMBER macro 1836
DRV_NVM_Close function 1844
DRV_NVM_COMMAND_HANDLE type 1860
DRV_NVM_COMMAND_HANDLE_INVALID macro 1861
DRV_NVM_COMMAND_STATUS enumeration 1860
DRV_NVM_CommandStatus function 1854
drv_nvm_config_template.h 1863
DRV_NVM_Deinitialize function 1842
DRV_NVM_DISABLE_ERROR_CHECK macro 1837
DRV_NVM_Erase function 1848
DRV_NVM_ERASE_WRITE_ENABLE macro 1837
DRV_NVM_EraseWrite function 1850
DRV_NVM_EVENT enumeration 1859
DRV_NVM_EVENT_HANDLER type 1859
DRV_NVM_EventHandlerSet function 1851
DRV_NVM_GeometryGet function 1855
DRV_NVM_INDEX_0 macro 1857
DRV_NVM_INDEX_1 macro 1858
DRV_NVM_INIT structure 1858
DRV_NVM_Initialize function 1840
DRV_NVM_INSTANCES_NUMBER macro 1836
DRV_NVM_INTERRUPT_MODE macro 1836
DRV_NVM_IsAttached function 1856
DRV_NVM_IsWriteProtected function 1857
DRV_NVM_MEDIA_SIZE macro 1838
DRV_NVM_MEDIA_START_ADDRESS macro 1838
DRV_NVM_Open function 1843
DRV_NVM_PAGE_SIZE macro 1837
DRV_NVM_PROGRAM_UNLOCK_KEY1 macro 1861
DRV_NVM_PROGRAM_UNLOCK_KEY2 macro 1861

DRV_NVM_Read function 1845
DRV_NVM_ROW_SIZE macro 1837
DRV_NVM_Status function 1843
DRV_NVM_SYS_FS_REGISTER macro 1838
DRV_NVM_Tasks function 1853
DRV_NVM_Write function 1846
DRV_OC_Disable function 1864
DRV_OC_Enable function 1865
DRV_OC_FaultHasOccurred function 1865
DRV_OC_Initialize function 1866
DRV_OC_Start function 1866
DRV_OC_Stop function 1867
drv_ovm7690_config_template.h 1388
DRV_OVM7690_INTERRUPT_MODE macro 1368
drv_pmp.h 1894
DRV_PMP_CHIPX_STROBE_MODE enumeration 1888
DRV_PMP_CLIENT_STATUS enumeration 1888
DRV_PMP_CLIENTS_NUMBER macro 1874
DRV_PMP_ClientStatus function 1882
DRV_PMP_Close function 1883
drv_pmp_config.h 1895
DRV_PMP_Deinitialize function 1877
DRV_PMP_ENDIAN_MODE enumeration 1888
DRV_PMP_INDEX enumeration 1889
DRV_PMP_INDEX_COUNT macro 1887
DRV_PMP_INIT structure 1889
DRV_PMP_Initialize function 1878
DRV_PMP_INSTANCES_NUMBER macro 1875
DRV_PMP_MODE_CONFIG structure 1890
DRV_PMP_ModeConfig function 1883
DRV_PMP_Open function 1884
DRV_PMP_POLARITY_OBJECT structure 1890
DRV_PMP_PORT_CONTROL enumeration 1891
DRV_PMP_PORTS structure 1891
DRV_PMP_QUEUE_ELEMENT_OBJ structure 1892
DRV_PMP_QUEUE_SIZE macro 1875
DRV_PMP_Read function 1885
DRV_PMP_Reinitialize function 1879
DRV_PMP_Status function 1880
DRV_PMP_Tasks function 1881
DRV_PMP_TimingSet function 1881
DRV_PMP_TRANSFER_STATUS enumeration 1892
DRV_PMP_TRANSFER_TYPE enumeration 1893
DRV_PMP_TransferStatus function 1887
DRV_PMP_WAIT_STATES structure 1892
DRV_PMP_Write function 1886
DRV_RTCC_AlarmDateGet function 1896
DRV_RTCC_AlarmTimeGet function 1897
DRV_RTCC_ClockOutput function 1897
DRV_RTCC_DateGet function 1898
DRV_RTCC_Initialize function 1898
DRV_RTCC_Start function 1899
DRV_RTCC_Stop function 1899
DRV_RTCC_TimeGet function 1899
drv_sdcard.h 1927
DRV_SDCARD_CLIENTS_NUMBER macro 1905
DRV_SDCARD_Close function 1913
DRV_SDCARD_COMMAND_HANDLE type 1925
DRV_SDCARD_COMMAND_HANDLE_INVALID macro 1925
DRV_SDCARD_COMMAND_STATUS enumeration 1925
DRV_SDCARD_CommandStatus function 1920
drv_sdcard_config_template.h 1929
DRV_SDCARD_Deinitialize function 1910
DRV_SDCARD_ENABLE_WRITE_PROTECT_CHECK macro 1907
DRV_SDCARD_EVENT enumeration 1926
DRV_SDCARD_EVENT_HANDLER type 1926
DRV_SDCARD_EventHandlerSet function 1918
DRV_SDCARD_GeometryGet function 1921
DRV_SDCARD_INDEX_0 macro 1922
DRV_SDCARD_INDEX_1 macro 1924
DRV_SDCARD_INDEX_2 macro 1924
DRV_SDCARD_INDEX_3 macro 1924
DRV_SDCARD_INDEX_COUNT macro 1922
DRV_SDCARD_INDEX_MAX macro 1906
DRV_SDCARD_INIT structure 1923
DRV_SDCARD_Initialize function 1909
DRV_SDCARD_INSTANCES_NUMBER macro 1906
DRV_SDCARD_IsAttached function 1919
DRV_SDCARD_IsWriteProtected function 1920
DRV_SDCARD_Open function 1914
DRV_SDCARD_POWER_STATE macro 1906
DRV_SDCARD_Read function 1915
DRV_SDCARD_Reinitialize function 1910
DRV_SDCARD_Status function 1911
DRV_SDCARD_SYS_FS_REGISTER macro 1907
DRV_SDCARD_Tasks function 1912
DRV_SDCARD_Write function 1916
drv_spi.h 1964
DRV_SPI_16BIT macro 1935
DRV_SPI_32BIT macro 1935
DRV_SPI_8BIT macro 1936
DRV_SPI_BUFFER_EVENT enumeration 1956
DRV_SPI_BUFFER_EVENT_HANDLER type 1957
DRV_SPI_BUFFER_HANDLE type 1958
DRV_SPI_BUFFER_HANDLE_INVALID macro 1956
DRV_SPI_BUFFER_TYPE enumeration 1958
DRV_SPI_BufferAddRead function 1948
DRV_SPI_BufferAddRead2 function 1951
DRV_SPI_BufferAddWrite function 1949
DRV_SPI_BufferAddWrite2 function 1952
DRV_SPI_BufferAddWriteRead function 1950
DRV_SPI_BufferAddWriteRead2 function 1953
DRV_SPI_BufferStatus function 1947
DRV_SPI_CLIENT_DATA structure 1963
DRV_SPI_ClientConfigure function 1947
DRV_SPI_CLIENTS_NUMBER macro 1939
DRV_SPI_CLOCK_MODE enumeration 1959
DRV_SPI_Close function 1945
drv_spi_config_template.h 1965
DRV_SPI_Deinitialize function 1943
DRV_SPI_DMA macro 1936
DRV_SPI_DMA_DUMMY_BUFFER_SIZE macro 1936
DRV_SPI_DMA_TXFER_SIZE macro 1937
DRV_SPI_EBM macro 1937
DRV_SPI_ELEMENTS_PER_QUEUE macro 1937
DRV_SPI_INDEX_0 macro 1956

DRV_SPI_INDEX_1 macro 1962
DRV_SPI_INDEX_2 macro 1962
DRV_SPI_INDEX_3 macro 1962
DRV_SPI_INDEX_4 macro 1962
DRV_SPI_INDEX_5 macro 1962
DRV_SPI_INDEX_COUNT macro 1956
DRV_SPI_INIT structure 1959
DRV_SPI_Initialize function 1942
DRV_SPI_INSTANCES_NUMBER macro 1939
DRV_SPI_ISR macro 1938
DRV_SPI_MASTER macro 1938
DRV_SPI_MODE enumeration 1961
DRV_SPI_Open function 1946
DRV_SPI_POLLED macro 1938
DRV_SPI_PROTOCOL_TYPE enumeration 1961
DRV_SPI_RM macro 1938
DRV_SPI_SLAVE macro 1939
DRV_SPI_Status function 1944
DRV_SPI_TASK_MODE enumeration 1963
DRV_SPI_Tasks function 1944
DRV_SPIn_ReceiverBufferIsFull function 1954
DRV_SPIn_TransmitterBufferIsFull function 1955
drv_sst25vf016b.h 2037
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE type 1993
DRV_SST25VF016B_BLOCK_COMMAND_HANDLE_INVALID macro 1996
DRV_SST25VF016B_BLOCK_EVENT enumeration 1993
DRV_SST25VF016B_BlockErase function 1985
DRV_SST25VF016B_BlockEventHandlerSet function 1986
DRV_SST25VF016B_BlockRead function 1988
DRV_SST25VF016B_BlockWrite function 1989
DRV_SST25VF016B_CLIENT_STATUS enumeration 1993
DRV_SST25VF016B_CLIENTS_NUMBER macro 1971
DRV_SST25VF016B_ClientStatus function 1984
DRV_SST25VF016B_Close function 1982
drv_sst25vf016b_config_template.h 2038
DRV_SST25VF016B_Deinitialize function 1980
DRV_SST25VF016B_EVENT_HANDLER type 1994
DRV_SST25VF016B_GeometryGet function 1991
DRV_SST25VF016B_HARDWARE_HOLD_ENABLE macro 1972
DRV_SST25VF016B_HARDWARE_WRITE_PROTECTION_ENABLE macro 1972
DRV_SST25VF016B_INDEX_0 macro 1996
DRV_SST25VF016B_INDEX_1 macro 1996
DRV_SST25VF016B_INIT structure 1995
DRV_SST25VF016B_Initialize function 1979
DRV_SST25VF016B_INSTANCES_NUMBER macro 1972
DRV_SST25VF016B_MediaIsAttached function 1992
DRV_SST25VF016B_MODE macro 1972
DRV_SST25VF016B_Open function 1983
DRV_SST25VF016B_QUEUE_DEPTH_COMBINED macro 1973
DRV_SST25VF016B_Status function 1981
DRV_SST25VF016B_Tasks function 1982
drv_sst25vf020b.h 2039
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE type 2013
DRV_SST25VF020B_BLOCK_COMMAND_HANDLE_INVALID macro 2016
DRV_SST25VF020B_BLOCK_EVENT enumeration 2013
DRV_SST25VF020B_BlockErase function 2004
DRV_SST25VF020B_BlockEraseWrite function 2011
DRV_SST25VF020B_BlockEventHandlerSet function 2006
DRV_SST25VF020B_BlockRead function 2007
DRV_SST25VF020B_BlockWrite function 2009
DRV_SST25VF020B_CLIENT_STATUS enumeration 2014
DRV_SST25VF020B_CLIENTS_NUMBER macro 1973
DRV_SST25VF020B_ClientStatus function 2001
DRV_SST25VF020B_Close function 2003
DRV_SST25VF020B_COMMAND_STATUS enumeration 2016
DRV_SST25VF020B_CommandStatus function 2002
drv_sst25vf020b_config_template.h 2040
DRV_SST25VF020B_Deinitialize function 1999
DRV_SST25VF020B_EVENT_HANDLER type 2014
DRV_SST25VF020B_GeometryGet function 2012
DRV_SST25VF020B_HARDWARE_HOLD_ENABLE macro 1974
DRV_SST25VF020B_HARDWARE_WRITE_PROTECTION_ENABLE macro 1974
DRV_SST25VF020B_INDEX_0 macro 2017
DRV_SST25VF020B_INDEX_1 macro 2017
DRV_SST25VF020B_INIT structure 2015
DRV_SST25VF020B_Initialize function 1998
DRV_SST25VF020B_INSTANCES_NUMBER macro 1974
DRV_SST25VF020B_MediaIsAttached function 2012
DRV_SST25VF020B_MODE macro 1974
DRV_SST25VF020B_Open function 2003
DRV_SST25VF020B_QUEUE_DEPTH_COMBINED macro 1975
DRV_SST25VF020B_Status function 2000
DRV_SST25VF020B_Tasks function 2000
drv_sst25vf064c.h 2040
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE type 2032
DRV_SST25VF064C_BLOCK_COMMAND_HANDLE_INVALID macro 2036
DRV_SST25VF064C_BLOCK_EVENT enumeration 2033
DRV_SST25VF064C_BlockErase function 2025
DRV_SST25VF064C_BlockEventHandlerSet function 2026
DRV_SST25VF064C_BlockRead function 2028
DRV_SST25VF064C_BlockWrite function 2029
DRV_SST25VF064C_CLIENT_STATUS enumeration 2033
DRV_SST25VF064C_CLIENTS_NUMBER macro 1975
DRV_SST25VF064C_ClientStatus function 2021
DRV_SST25VF064C_Close function 2022
DRV_SST25VF064C_COMMAND_STATUS enumeration 2034
DRV_SST25VF064C_CommandStatus function 2023
drv_sst25vf064c_config_template.h 2041
DRV_SST25VF064C_Deinitialize function 2019
DRV_SST25VF064C_EVENT_HANDLER type 2034
DRV_SST25VF064C_GeometryGet function 2031
DRV_SST25VF064C_HARDWARE_HOLD_ENABLE macro 1976
DRV_SST25VF064C_HARDWARE_WRITE_PROTECTION_ENABLE macro 1976
DRV_SST25VF064C_INDEX_0 macro 2036
DRV_SST25VF064C_INDEX_1 macro 2037
DRV_SST25VF064C_INIT structure 2035
DRV_SST25VF064C_Initialize function 2018
DRV_SST25VF064C_INSTANCES_NUMBER macro 1976
DRV_SST25VF064C_MediaIsAttached function 2032
DRV_SST25VF064C_MODE macro 1976
DRV_SST25VF064C_Open function 2024
DRV_SST25VF064C_QUEUE_DEPTH_COMBINED macro 1977

DRV_SST25VF064C_Status function 2020
DRV_SST25VF064C_Tasks function 2021
DRV_STATIC_BUILD macro 1739
drv_tmr.h 2090
DRV_TMR_Alarm16BitDeregister function 2062
DRV_TMR_Alarm16BitRegister function 2069
DRV_TMR_Alarm32BitDeregister function 2063
DRV_TMR_Alarm32BitRegister function 2070
DRV_TMR_AlarmDeregister function 2071
DRV_TMR_AlarmDisable function 2068
DRV_TMR_AlarmEnable function 2068
DRV_TMR_AlarmHasElapsed function 2064
DRV_TMR_AlarmPeriod16BitGet function 2065
DRV_TMR_AlarmPeriod16BitSet function 2066
DRV_TMR_AlarmPeriod32BitGet function 2067
DRV_TMR_AlarmPeriod32BitSet function 2067
DRV_TMR_AlarmPeriodGet function 2072
DRV_TMR_AlarmPeriodSet function 2073
DRV_TMR_AlarmRegister function 2073
DRV_TMR_ASYNC_WRITE_ENABLE macro 2050
DRV_TMR_CALLBACK type 2085
DRV_TMR_CLIENT_STATUS enumeration 2086
DRV_TMR_CLIENTS_NUMBER macro 2051
DRV_TMR_ClientStatus function 2059
DRV_TMR_CLK_SOURCES enumeration 2086
DRV_TMR_CLOCK_PRESCALER macro 2049
DRV_TMR_CLOCK_SOURCE macro 2051
DRV_TMR_ClockSet function 2057
DRV_TMR_Close function 2059
drv_tmr_config_template.h 2093
DRV_TMR_CounterClear function 2079
DRV_TMR_CounterFrequencyGet function 2075
DRV_TMR_CounterValue16BitGet function 2075
DRV_TMR_CounterValue16BitSet function 2076
DRV_TMR_CounterValue32BitGet function 2077
DRV_TMR_CounterValue32BitSet function 2078
DRV_TMR_CounterValueGet function 2080
DRV_TMR_CounterValueSet function 2081
DRV_TMR_Deinitialize function 2054
DRV_TMR_DIVIDER_RANGE structure 2087
DRV_TMR_DividerRangeGet function 2084
DRV_TMR_GateModeClear function 2082
DRV_TMR_GateModeSet function 2058
DRV_TMR_INDEX_0 macro 2088
DRV_TMR_INDEX_1 macro 2089
DRV_TMR_INDEX_2 macro 2089
DRV_TMR_INDEX_3 macro 2089
DRV_TMR_INDEX_4 macro 2089
DRV_TMR_INDEX_5 macro 2089
DRV_TMR_INDEX_6 macro 2090
DRV_TMR_INDEX_7 macro 2090
DRV_TMR_INDEX_8 macro 2090
DRV_TMR_INDEX_9 macro 2090
DRV_TMR_INDEX_COUNT macro 2088
DRV_TMR_INIT structure 2085
DRV_TMR_Initialize function 2055
DRV_TMR_INSTANCES_NUMBER macro 2048
DRV_TMR_INTERRUPT_MODE macro 2049
DRV_TMR_INTERRUPT_SOURCE macro 2050
DRV_TMR_MODE macro 2049
DRV_TMR_MODULE_ID macro 2050
DRV_TMR_MODULE_INIT macro 2050
DRV_TMR_Open function 2060
DRV_TMR_OPERATION_MODE enumeration 2088
DRV_TMR_OperationModeGet function 2083
DRV_TMR_PrescalerGet function 2082
DRV_TMR_Start function 2061
DRV_TMR_Status function 2056
DRV_TMR_Stop function 2062
DRV_TMR_Tasks function 2057
DRV_TMR_Tasks_ISR function 2084
DRV_TOUCH_ADC10BIT_CalibrationSet function 2098
DRV_TOUCH_ADC10BIT_CLIENT_DATA structure 2107
DRV_TOUCH_ADC10BIT_Close function 2098
DRV_TOUCH_ADC10BIT_Deinitialize function 2099
DRV_TOUCH_ADC10BIT_HANDLE type 2107
DRV_TOUCH_ADC10BIT_HANDLE_INVALID macro 2108
DRV_TOUCH_ADC10BIT_INDEX_0 macro 2108
DRV_TOUCH_ADC10BIT_INDEX_1 macro 2108
DRV_TOUCH_ADC10BIT_INDEX_COUNT macro 2108
DRV_TOUCH_ADC10BIT_INIT structure 2107
DRV_TOUCH_ADC10BIT_Initialize function 2100
DRV_TOUCH_ADC10BIT_Open function 2101
DRV_TOUCH_ADC10BIT_PositionDetect function 2105
DRV_TOUCH_ADC10BIT_Status function 2101
DRV_TOUCH_ADC10BIT_Tasks function 2102
DRV_TOUCH_ADC10BIT_TouchDataRead function 2106
DRV_TOUCH_ADC10BIT_TouchGetRawX function 2103
DRV_TOUCH_ADC10BIT_TouchGetRawY function 2103
DRV_TOUCH_ADC10BIT_TouchGetX function 2104
DRV_TOUCH_ADC10BIT_TouchGetY function 2105
DRV_TOUCH_ADC10BIT_TouchStatus function 2106
DRV_TOUCH_ADC10BIT_TouchStoreCalibration function 2104
DRV_TOUCH_AR1021_Calibrate function 2123
DRV_TOUCH_AR1021_CALIBRATION_PROMPT_CALLBACK structure 2125
DRV_TOUCH_AR1021_CalibrationSet function 2123
DRV_TOUCH_AR1021_Close function 2123
DRV_TOUCH_AR1021_Deinitialize function 2117
DRV_TOUCH_AR1021_FactoryDefaultSet function 2118
DRV_TOUCH_AR1021_HANDLE type 2126
DRV_TOUCH_AR1021_HANDLE_INVALID macro 2127
DRV_TOUCH_AR1021_INDEX_0 macro 2127
DRV_TOUCH_AR1021_INDEX_COUNT macro 2127
DRV_TOUCH_AR1021_Initialize function 2118
DRV_TOUCH_AR1021_MODULE_ID enumeration 2126
DRV_TOUCH_AR1021_Open function 2124
DRV_TOUCH_AR1021_RegisterConfigWrite function 2119
DRV_TOUCH_AR1021_Status function 2119
DRV_TOUCH_AR1021_TASK_STATE enumeration 2126
DRV_TOUCH_AR1021_Tasks function 2120
DRV_TOUCH_AR1021_TouchDataRead function 2121
DRV_TOUCH_AR1021_TouchGetX function 2121
DRV_TOUCH_AR1021_TouchGetY function 2121
DRV_TOUCH_AR1021_TouchPenGet function 2122
DRV_TOUCH_AR1021_TouchStatus function 2122

DRV_TOUCH_MTCH6301_CLIENT_OBJECT structure 2145
DRV_TOUCH_MTCH6301_Close function 2136
DRV_TOUCH_MTCH6301_Deinitialize function 2137
DRV_TOUCH_MTCH6301_HANDLE type 2143
DRV_TOUCH_MTCH6301_HANDLE_INVALID macro 2144
DRV_TOUCH_MTCH6301_I2C_MASTER_READ_ID macro 2148
DRV_TOUCH_MTCH6301_I2C_MASTER_WRITE_ID macro 2148
DRV_TOUCH_MTCH6301_I2C_READ_FRAME_SIZE macro 2144
DRV_TOUCH_MTCH6301_INDEX_0 macro 2145
DRV_TOUCH_MTCH6301_INDEX_1 macro 2145
DRV_TOUCH_MTCH6301_INDEX_COUNT macro 2146
DRV_TOUCH_MTCH6301_Initialize function 2137
DRV_TOUCH_MTCH6301_MODULE_ID enumeration 2144
DRV_TOUCH_MTCH6301_OBJECT structure 2146
DRV_TOUCH_MTCH6301_Open function 2139
DRV_TOUCH_MTCH6301_ReadRequest function 2141
DRV_TOUCH_MTCH6301_Status function 2140
DRV_TOUCH_MTCH6301_TASK_QUEUE structure 2147
DRV_TOUCH_MTCH6301_TASK_STATE enumeration 2147
DRV_TOUCH_MTCH6301_Tasks function 2141
DRV_TOUCH_MTCH6301_TouchDataRead function 2143
DRV_TOUCH_MTCH6301_TouchGetX function 2142
DRV_TOUCH_MTCH6301_TouchGetY function 2142
DRV_TOUCH_MTCH6301_TouchStatus function 2143
DRV_TOUCH_MTCH6303_I2C_REGISTER_MAP enumeration 2179
DRV_TOUCH_MTCH6303_MSG_ID enumeration 2178
drv_usart.h 2393
DRV_USART_AddressedBufferAddWrite function 2367
DRV_USART_BAUD_RATE_IDXn macro 2347
DRV_USART_BAUD_SET_RESULT enumeration 2387
DRV_USART_BaudSet function 2360
DRV_USART_BUFFER_EVENT enumeration 2387
DRV_USART_BUFFER_EVENT_HANDLER type 2388
DRV_USART_BUFFER_HANDLE type 2389
DRV_USART_BUFFER_HANDLE_INVALID macro 2390
DRV_USART_BUFFER_QUEUE_SUPPORT macro 2343
DRV_USART_BUFFER_RESULT enumeration 2393
DRV_USART_BufferAddRead function 2362
DRV_USART_BufferAddWrite function 2363
DRV_USART_BufferCompletedBytesGet function 2369
DRV_USART_BufferEventHandlerSet function 2365
DRV_USART_BufferProcessedSizeGet function 2367
DRV_USART_BufferRemove function 2370
DRV_USART_BYTE_EVENT_HANDLER type 2392
DRV_USART_BYTE_MODEL_BLOCKING macro 2347
DRV_USART_BYTE_MODEL_CALLBACK macro 2347
DRV_USART_BYTE_MODEL_SUPPORT macro 2344
DRV_USART_ByteErrorCallbackSet function 2380
DRV_USART_ByteReceiveCallbackSet function 2380
DRV_USART_ByteTransmitCallbackSet function 2381
DRV_USART_CLIENT_STATUS enumeration 2383
DRV_USART_CLIENTS_NUMBER macro 2341
DRV_USART_ClientStatus function 2358
DRV_USART_Close function 2357
drv_usart_config_template.h 2395
DRV_USART_COUNT macro 2390
DRV_USART_Deinitialize function 2353
DRV_USART_ERROR enumeration 2389
DRV_USART_ErrorGet function 2359
DRV_USART_INDEX macro 2342
DRV_USART_INDEX_0 macro 2385
DRV_USART_INDEX_1 macro 2386
DRV_USART_INDEX_2 macro 2386
DRV_USART_INDEX_3 macro 2386
DRV_USART_INDEX_4 macro 2386
DRV_USART_INDEX_5 macro 2387
DRV_USART_INIT structure 2383
DRV_USART_INIT_FLAGS enumeration 2384
DRV_USART_Initialize function 2352
DRV_USART_INSTANCES_NUMBER macro 2343
DRV_USART_INTERRUPT_MODE macro 2342
DRV_USART_INTERRUPT_SOURCE_ERROR macro 2342
DRV_USART_INTERRUPT_SOURCE_RECEIVE macro 2344
DRV_USART_INTERRUPT_SOURCE_RECEIVE_DMA macro 2344
DRV_USART_INTERRUPT_SOURCE_TRANSMIT macro 2345
DRV_USART_INTERRUPT_SOURCE_TRANSMIT_DMA macro 2345
DRV_USART_LINE_CONTROL enumeration 2391
DRV_USART_LINE_CONTROL_SET_RESULT enumeration 2389
DRV_USART_LineControlSet function 2361
DRV_USART_Open function 2356
DRV_USART_OPERATION_MODE enumeration 2392
DRV_USART_OPERATION_MODE_DATA union 2390
DRV_USART_PERIPHERAL_ID macro 2343
DRV_USART_QUEUE_DEPTH_COMBINED macro 2345
DRV_USART_RCV_QUEUE_SIZE_IDXn macro 2348
DRV_USART_Read function 2372
DRV_USART_READ_ERROR macro 2391
DRV_USART_READ_WRITE_MODEL_SUPPORT macro 2346
DRV_USART_ReadByte function 2374
DRV_USART_RECEIVE_DMA macro 2346
DRV_USART_ReceiverBufferIsEmpty function 2379
DRV_USART_ReceiverBufferSizeGet function 2376
DRV_USART_Status function 2353
DRV_USART_TasksError function 2356
DRV_USART_TasksReceive function 2354
DRV_USART_TasksTransmit function 2355
DRV_USART_TRANSFER_STATUS enumeration 2385
DRV_USART_TransferStatus function 2377
DRV_USART_TRANSMIT_DMA macro 2346
DRV_USART_TransmitBufferIsFull function 2378
DRV_USART_TransmitBufferSizeGet function 2376
DRV_USART_Write function 2373
DRV_USART_WRITE_ERROR macro 2391
DRV_USART_WriteByte function 2375
DRV_USART_XMIT_QUEUE_SIZE_IDXn macro 2348
drv_usbfs.h 2266
DRV_USBFS_ClientEventCallBackSet function 2226
DRV_USBFS_Close function 2227
drv_usbfs_config_template.h 2268
DRV_USBFS_DEVICE_AddressSet function 2230
DRV_USBFS_DEVICE_Attach function 2230
DRV_USBFS_DEVICE_CurrentSpeedGet function 2231
DRV_USBFS_DEVICE_Detach function 2232
DRV_USBFS_DEVICE_EndpointDisable function 2233
DRV_USBFS_DEVICE_EndpointDisableAll function 2234
DRV_USBFS_DEVICE_EndpointEnable function 2234

DRV_USBFS_DEVICE_EndpointIsEnabled function 2236
DRV_USBFS_DEVICE_EndpointIsStalled function 2237
DRV_USBFS_DEVICE_EndpointStall function 2238
DRV_USBFS_DEVICE_EndpointStallClear function 2238
DRV_USBFS_DEVICE_INTERFACE macro 2264
DRV_USBFS_DEVICE_IRPCancel function 2239
DRV_USBFS_DEVICE_IRPCancelAll function 2241
DRV_USBFS_DEVICE_IRPSubmit function 2242
DRV_USBFS_DEVICE_RemoteWakeupStart function 2243
DRV_USBFS_DEVICE_RemoteWakeupStop function 2244
DRV_USBFS_DEVICE_SOFNumberGet function 2245
DRV_USBFS_DEVICE_SUPPORT macro 2219
DRV_USBFS_ENDPOINT_TABLE_ENTRY_SIZE macro 2264
DRV_USBFS_ENDPOINTS_NUMBER macro 2219
DRV_USBFS_EVENT enumeration 2260
DRV_USBFS_EVENT_CALLBACK type 2261
DRV_USBFS_HOST_ATTACH_DEBOUNCE_DURATION macro 2219
DRV_USBFS_HOST_EventsDisable function 2246
DRV_USBFS_HOST_EventsEnable function 2246
DRV_USBFS_HOST_INTERFACE macro 2265
DRV_USBFS_HOST_IRPCancel function 2247
DRV_USBFS_HOST_IRPSubmit function 2248
DRV_USBFS_HOST_NAK_LIMIT macro 2220
DRV_USBFS_HOST_PIPE_HANDLE type 2261
DRV_USBFS_HOST_PIPE_HANDLE_INVALID macro 2265
DRV_USBFS_HOST_PipeClose function 2250
DRV_USBFS_HOST_PIPES_NUMBER macro 2220
DRV_USBFS_HOST_PipeSetup function 2250
DRV_USBFS_HOST_RESET_DURATION macro 2220
DRV_USBFS_HOST_ROOT_HUB_BusSpeedGet function 2252
DRV_USBFS_HOST_ROOT_HUB_Initialize function 2252
DRV_USBFS_HOST_ROOT_HUB_MaximumCurrentGet function 2253
DRV_USBFS_HOST_ROOT_HUB_OperationEnable function 2254
DRV_USBFS_HOST_ROOT_HUB_OperationIsEnabled function 2255
DRV_USBFS_HOST_ROOT_HUB_PortNumbersGet function 2256
DRV_USBFS_HOST_ROOT_HUB_PortReset function 2256
DRV_USBFS_HOST_ROOT_HUB_PortResetIsComplete function 2257
DRV_USBFS_HOST_ROOT_HUB_PortResume function 2258
DRV_USBFS_HOST_ROOT_HUB_PortSpeedGet function 2259
DRV_USBFS_HOST_ROOT_HUB_PortSuspend function 2259
DRV_USBFS_HOST_SUPPORT macro 2221
DRV_USBFS_INDEX_0 macro 2265
DRV_USBFS_INDEX_1 macro 2266
DRV_USBFS_INIT structure 2262
DRV_USBFS_Initialize function 2228
DRV_USBFS_INSTANCES_NUMBER macro 2221
DRV_USBFS_INTERRUPT_MODE macro 2221
DRV_USBFS_Open function 2229
DRV_USBFS_OPMODES enumeration 2263
DRV_USBFS_ROOT_HUB_PORT_INDICATION type 2263
DRV_USBFS_ROOT_HUB_PORT_OVER_CURRENT_DETECT type 2263
DRV_USBFS_ROOT_HUB_PORT_POWER_ENABLE type 2264
DRV_USBFS_Status function 2224
DRV_USBFS_Tasks function 2225
DRV_USBFS_Tasks_ISR function 2226
drv_usbhs.h 2326
DRV_USBHS_ClientEventCallBackSet function 2286
DRV_USBHS_Close function 2287
drv_usbhs_config_template.h 2328
DRV_USBHS_DEVICE_AddressSet function 2289
DRV_USBHS_DEVICE_Attach function 2289
DRV_USBHS_DEVICE_CurrentSpeedGet function 2290
DRV_USBHS_DEVICE_Detach function 2291
DRV_USBHS_DEVICE_EndpointDisable function 2292
DRV_USBHS_DEVICE_EndpointDisableAll function 2293
DRV_USBHS_DEVICE_EndpointEnable function 2294
DRV_USBHS_DEVICE_EndpointIsEnabled function 2295
DRV_USBHS_DEVICE_EndpointIsStalled function 2296
DRV_USBHS_DEVICE_EndpointStall function 2297
DRV_USBHS_DEVICE_EndpointStallClear function 2297
DRV_USBHS_DEVICE_INTERFACE macro 2325
DRV_USBHS_DEVICE_IRPCancel function 2298
DRV_USBHS_DEVICE_IRPCancelAll function 2300
DRV_USBHS_DEVICE_IRPSubmit function 2301
DRV_USBHS_DEVICE_RemoteWakeupStart function 2303
DRV_USBHS_DEVICE_RemoteWakeupStop function 2303
DRV_USBHS_DEVICE_SOFNumberGet function 2304
DRV_USBHS_DEVICE_SUPPORT macro 2277
DRV_USBHS_DEVICE_TestModeEnter function 2305
DRV_USBHS_DEVICE_TestModeExit function 2305
DRV_USBHS_ENDPOINTS_NUMBER macro 2278
DRV_USBHS_EVENT enumeration 2321
DRV_USBHS_EVENT_CALLBACK type 2322
DRV_USBHS_HOST_ATTACH_DEBOUNCE_DURATION macro 2278
DRV_USBHS_HOST_EventsDisable function 2306
DRV_USBHS_HOST_EventsEnable function 2307
DRV_USBHS_HOST_INTERFACE macro 2325
DRV_USBHS_HOST_IRPCancel function 2308
DRV_USBHS_HOST_IRPSubmit function 2309
DRV_USBHS_HOST_NAK_LIMIT macro 2278
DRV_USBHS_HOST_PIPE_HANDLE type 2322
DRV_USBHS_HOST_PIPE_HANDLE_INVALID macro 2326
DRV_USBHS_HOST_PipeClose function 2311
DRV_USBHS_HOST_PIPES_NUMBER macro 2279
DRV_USBHS_HOST_PipeSetup function 2311
DRV_USBHS_HOST_RESET_DURATION macro 2279
DRV_USBHS_HOST_ROOT_HUB_BusSpeedGet function 2313
DRV_USBHS_HOST_ROOT_HUB_Initialize function 2313
DRV_USBHS_HOST_ROOT_HUB_MaximumCurrentGet function 2314
DRV_USBHS_HOST_ROOT_HUB_OperationEnable function 2315
DRV_USBHS_HOST_ROOT_HUB_OperationIsEnabled function 2316
DRV_USBHS_HOST_ROOT_HUB_PortNumbersGet function 2316
DRV_USBHS_HOST_ROOT_HUB_PortReset function 2317
DRV_USBHS_HOST_ROOT_HUB_PortResetIsComplete function 2318
DRV_USBHS_HOST_ROOT_HUB_PortResume function 2319
DRV_USBHS_HOST_ROOT_HUB_PortSpeedGet function 2319
DRV_USBHS_HOST_ROOT_HUB_PortSuspend function 2320
DRV_USBHS_HOST_SUPPORT macro 2279
DRV_USBHS_INDEX_0 macro 2326
DRV_USBHS_INIT structure 2322
DRV_USBHS_Initialize function 2283
DRV_USBHS_INSTANCES_NUMBER macro 2280
DRV_USBHS_INTERRUPT_MODE macro 2280
DRV_USBHS_Open function 2288
DRV_USBHS_OPMODES enumeration 2323

DRV_USBHS_ROOT_HUB_PORT_INDICATION type 2324
DRV_USBHS_ROOT_HUB_PORT_OVER_CURRENT_DETECT type 2324
DRV_USBHS_ROOT_HUB_PORT_POWER_ENABLE type 2325
DRV_USBHS_Status function 2284
DRV_USBHS_Tasks function 2284
DRV_USBHS_Tasks_ISR function 2285
DRV_USBHS_Tasks_ISR_USBDMA function 2286
drv_wifi.h 2489
DRV_WIFI_ADHOC_CONNECT_ONLY enumeration member 2469
DRV_WIFI_ADHOC_CONNECT_THEN_START enumeration member 2469
DRV_WIFI_ADHOC_MODES enumeration 2469
DRV_WIFI_ADHOC_NETWORK_CONTEXT structure 2469
DRV_WIFI_ADHOC_START_ONLY enumeration member 2469
DRV_WIFI_AdhocContextSet function 2421
DRV_WIFI_APCONFIG_BIT_PREAMBLE_LONG macro 2488
DRV_WIFI_APCONFIG_BIT_PRIVACY macro 2488
DRV_WIFI_APCONFIG_BIT_WPA macro 2488
DRV_WIFI_APCONFIG_BIT_WPA2 macro 2488
DRV_WIFI_BSSID_LENGTH macro 2460
DRV_WIFI_BssidGet function 2441
DRV_WIFI_BssidSet function 2430
DRV_WIFI_ChannelListGet function 2442
DRV_WIFI_ChannelListSet function 2418
DRV_WIFI_ConfigDataDelete function 2441
DRV_WIFI_ConfigDataLoad function 2440
DRV_WIFI_ConfigDataSave function 2440
DRV_WIFI_Connect function 2429
DRV_WIFI_ConnectContextGet function 2442
DRV_WIFI_CONNECTION_CONTEXT structure 2470
DRV_WIFI_CONNECTION_STATES enumeration 2470
DRV_WIFI_CONNECTION_TEMPORARILY_LOST 2404
DRV_WIFI_ConnectionStateGet function 2443
DRV_WIFI_ContextLoad function 2423
DRV_WIFI_DEAUTH_REASONCODE_MASK macro 2460
DRV_WIFI_DEFAULT_ADHOC_BEACON_PERIOD macro 2460
DRV_WIFI_DEFAULT_ADHOC_HIDDEN_SSID macro 2461
DRV_WIFI_DEFAULT_ADHOC_MODE macro 2461
DRV_WIFI_DEFAULT_PS_DTIM_ENABLED macro 2461
DRV_WIFI_DEFAULT_PS_DTIM_INTERVAL macro 2461
DRV_WIFI_DEFAULT_PS_LISTEN_INTERVAL macro 2461
DRV_WIFI_DEFAULT_SCAN_COUNT macro 2462
DRV_WIFI_DEFAULT_SCAN_MAX_CHANNEL_TIME macro 2462
DRV_WIFI_DEFAULT_SCAN_MIN_CHANNEL_TIME macro 2462
DRV_WIFI_DEFAULT_SCAN_PROBE_DELAY macro 2462
DRV_WIFI_DEFAULT_SOFTAP_HIDDEN_SSID macro 2487
DRV_WIFI_DEFAULT_WEP_KEY_INDEX macro 2486
DRV_WIFI_DEFAULT_WEP_KEY_TYPE macro 2462
DRV_WIFI_Deferred_ISR function 2416
DRV_WIFI_DeferredISR_SemGive function 2417
DRV_WIFI_Deinitialize function 2415
DRV_WIFI_DEVICE_INFO structure 2471
DRV_WIFI_DEVICE_TYPE enumeration 2488
DRV_WIFI_DeviceInfoGet function 2444
DRV_WIFI_DISABLED macro 2463
DRV_WIFI_DISASSOC_REASONCODE_MASK macro 2463
DRV_WIFI_Disconnect function 2430
DRV_WIFI_DOMAIN_CODES enumeration 2471
DRV_WIFI_ENABLED macro 2463
DRV_WIFI_EVENT_CONN_TEMP_LOST_CODES enumeration 2471
DRV_WIFI_EVENT_CONNECTION_FAILED 2405
DRV_WIFI_EVENT_INFO enumeration 2472
DRV_WIFI_EVENTS enumeration 2472
DRV_WIFI_GENERAL_ERRORS enumeration 2473
DRV_WIFI_HIBERNATE_STATES enumeration 2473
DRV_WIFI_HibernateEnable function 2426
DRV_WIFI_HibernateModeClear function 2427
DRV_WIFI_InHibernateMode function 2456
DRV_WIFI_Initialize function 2415
DRV_WIFI_InitTask function 2417
DRV_WIFI_INT_Handle function 2435
drv_wifi_iwpriv.h 2494
DRV_WIFI_LinkDownThresholdSet function 2422
DRV_WIFI_MAC_STATS structure 2474
DRV_WIFI_MacAddressGet function 2444
DRV_WIFI_MacAddressSet function 2431
DRV_WIFI_MacStatsGet function 2445
DRV_WIFI_MACTask function 2418
DRV_WIFI_MAX_CHANNEL_LIST_LENGTH macro 2463
DRV_WIFI_MAX_NUM_RATES macro 2463
DRV_WIFI_MAX_SECURITY_KEY_LENGTH macro 2464
DRV_WIFI_MAX_SSID_LENGTH macro 2464
DRV_WIFI_MAX_WEP_KEY_LENGTH macro 2464
DRV_WIFI_MAX_WPA_PASS_PHRASE_LENGTH macro 2464
DRV_WIFI_MGMT_ERRORS enumeration 2475
DRV_WIFI_MGMT_INDICATE_SOFT_AP_EVENT structure 2476
DRV_WIFI_MIN_WPA_PASS_PHRASE_LENGTH macro 2464
DRV_WIFI_MRF24W_ISR function 2416
DRV_WIFI_MULTICAST_CONFIG structure 2487
DRV_WIFI_MULTICAST_FILTER_IDS enumeration 2476
DRV_WIFI_MULTICAST_FILTERS enumeration 2476
DRV_WIFI_MulticastFilterSet function 2428
DRV_WIFI_NETWORK_TYPE_ADHOC macro 2465
DRV_WIFI_NETWORK_TYPE_INFRASTRUCTURE macro 2465
DRV_WIFI_NETWORK_TYPE_SOFT_AP macro 2465
DRV_WIFI_NetworkTypeGet function 2445
DRV_WIFI_NetworkTypeSet function 2419
DRV_WIFI_NO_ADDITIONAL_INFO macro 2465
DRV_WIFI_POWER_SAVE_STATES enumeration 2477
DRV_WIFI_PowerSaveStateGet function 2446
DRV_WIFI_ProcessEvent function 2438
DRV_WIFI_PS_POLL_CONTEXT structure 2477
DRV_WIFI_PsPollDisable function 2427
DRV_WIFI_PsPollEnable function 2428
DRV_WIFI_REASON_CODES enumeration 2478
DRV_WIFI_RECONNECT_MODES enumeration 2478
DRV_WIFI_ReconnectModeGet function 2447
DRV_WIFI_ReconnectModeSet function 2419
DRV_WIFI_RegionalDomainGet function 2447
DRV_WIFI_RETRY_ADHOC macro 2465
DRV_WIFI_RETRY_FOREVER macro 2466
DRV_WIFI_RSSI_Cache_FromRxDataRead function 2434
DRV_WIFI_RSSI_Get_FromRxDataRead function 2435
DRV_WIFI_RssiGet function 2452
DRV_WIFI_RssiSet function 2433

DRV_WIFI_RTS_THRESHOLD_MAX macro 2466
DRV_WIFI_RtsThresholdGet function 2448
DRV_WIFI_RtsThresholdSet function 2432
DRV_WIFI_Scan function 2437
DRV_WIFI_SCAN_CONTEXT structure 2478
DRV_WIFI_SCAN_RESULT structure 2479
DRV_WIFI_SCAN_TYPES enumeration 2480
DRV_WIFI_ScanContextGet function 2437
DRV_WIFI_ScanContextSet function 2432
DRV_WIFI_ScanResultGet function 2455
DRV_WIFI_SECURITY_OPEN macro 2466
DRV_WIFI_SECURITY_WEP_104 macro 2466
DRV_WIFI_SECURITY_WEP_40 macro 2466
DRV_WIFI_SECURITY_WPA_AUTO_WITH_KEY macro 2467
DRV_WIFI_SECURITY_WPA_AUTO_WITH_PASS_PHRASE macro 2467
DRV_WIFI_SECURITY_WPA_WITH_KEY macro 2467
DRV_WIFI_SECURITY_WPA_WITH_PASS_PHRASE macro 2467
DRV_WIFI_SECURITY_WPA2_WITH_KEY macro 2467
DRV_WIFI_SECURITY_WPA2_WITH_PASS_PHRASE macro 2468
DRV_WIFI_SECURITY_WPS_PIN macro 2468
DRV_WIFI_SECURITY_WPS_PUSH_BUTTON macro 2468
DRV_WIFI_SecurityGet function 2449
DRV_WIFI_SecurityOpenSet function 2424
DRV_WIFI_SecurityTypeGet function 2452
DRV_WIFI_SecurityWepSet function 2425
DRV_WIFI_SecurityWpaSet function 2424
DRV_WIFI_SecurityWpsSet function 2425
DRV_WIFI_SetPSK function 2423
DRV_WIFI_SOFT_AP_EVENT_REASON_CODES enumeration 2480
DRV_WIFI_SOFT_AP_STATES enumeration 2481
DRV_WIFI_SOFTAP_NETWORK_CONTEXT structure 2486
DRV_WIFI_SoftAPContextSet function 2439
DRV_WIFI_SoftApEventInfoGet function 2439
DRV_WIFI_SpiClose function 2436
DRV_WIFI_SpiDmaRx function 2456
DRV_WIFI_SpiDmaTx function 2457
DRV_WIFI_SpiInit function 2457
DRV_WIFI_SpiRx function 2458
DRV_WIFI_SpiTx function 2458
DRV_WIFI_SsidGet function 2450
DRV_WIFI_SsidSet function 2421
DRV_WIFI_STATUS_CODES enumeration 2481
DRV_WIFI_TaskSyncInit function 2436
DRV_WIFI_TX_MODES enumeration 2481
DRV_WIFI_TxModeGet function 2450
DRV_WIFI_TxModeSet function 2433
DRV_WIFI_TxPowerFactoryMaxGet function 2453
DRV_WIFI_TxPowerMaxGet function 2454
DRV_WIFI_TxPowerMaxSet function 2454
DRV_WIFI_WEP_CONTEXT structure 2482
DRV_WIFI_WEP_KEY_TYPE enumeration 2482
DRV_WIFI_WEP104_KEY_LENGTH macro 2468
DRV_WIFI_WEP40_KEY_LENGTH macro 2468
DRV_WIFI_WepKeyTypeGet function 2451
DRV_WIFI_WPA_CONTEXT structure 2483
DRV_WIFI_WPA_KEY_INFO structure 2483
DRV_WIFI_WPA_KEY_LENGTH macro 2469
DRV_WIFI_WPS_AUTH_TYPES enumeration 2483
DRV_WIFI_WPS_CONTEXT structure 2484
DRV_WIFI_WPS_CREDENTIAL structure 2484
DRV_WIFI_WPS_ENCODE_TYPES enumeration 2485
DRV_WIFI_WPS_ERROR_CONFIG_CODES enumeration 2485
DRV_WIFI_WPS_PIN_LENGTH macro 2469
DRV_WIFI_WPS_STATE_CODES enumeration 2486
DRV_WIFI_WPSCredentialsGet function 2451
drv_xc2c64a.h 1568
DSP Fixed-Point Math Library 2999
dsp.h 3088
DSP_ComplexAdd32 function 3004
DSP_ComplexConj16 function 3005
DSP_ComplexConj32 function 3005
DSP_ComplexDotProd32 function 3006
DSP_ComplexMult32 function 3007
DSP_ComplexScalarMult32 function 3008
DSP_ComplexSub32 function 3008
DSP_FilterFIR32 function 3009
DSP_FilterFIRDecim32 function 3010
DSP_FilterFIRInterp32 function 3011
DSP_FilterIIR16 function 3013
DSP_FilterIIRBQ16 function 3013
DSP_FilterIIRBQ16_cascade8 function 3015
DSP_FilterIIRBQ16_cascade8_fast function 3016
DSP_FilterIIRBQ16_fast function 3018
DSP_FilterIIRBQ16_parallel8 function 3019
DSP_FilterIIRBQ16_parallel8_fast function 3020
DSP_FilterIIRBQ32 function 3022
DSP_FilterIIRSetup16 function 3023
DSP_FilterLMS16 function 3023
DSP_MatrixAdd32 function 3025
DSP_MatrixEqual32 function 3026
DSP_MatrixInit32 function 3026
DSP_MatrixMul32 function 3027
DSP_MatrixScale32 function 3028
DSP_MatrixSub32 function 3029
DSP_MatrixTranspose32 function 3030
DSP_TransformFFT16 function 3031
DSP_TransformFFT16_setup function 3032
DSP_TransformFFT32 function 3033
DSP_TransformFFT32_setup function 3033
DSP_TransformIFFT16 function 3034
DSP_TransformWindow_Bart16 function 3035
DSP_TransformWindow_Bart32 function 3036
DSP_TransformWindow_Black16 function 3037
DSP_TransformWindow_Black32 function 3037
DSP_TransformWindow_Cosine16 function 3038
DSP_TransformWindow_Cosine32 function 3039
DSP_TransformWindow_Hamm16 function 3040
DSP_TransformWindow_Hamm32 function 3040
DSP_TransformWindow_Hann16 function 3041
DSP_TransformWindow_Hann32 function 3042
DSP_TransformWindow_Kaiser16 function 3043
DSP_TransformWindow_Kaiser32 function 3043
DSP_TransformWinInit_Bart16 function 3044
DSP_TransformWinInit_Bart32 function 3045
DSP_TransformWinInit_Black16 function 3045

DSP_TransformWinInit_Black32 function 3046
DSP_TransformWinInit_Cosine16 function 3047
DSP_TransformWinInit_Cosine32 function 3047
DSP_TransformWinInit_Hamm16 function 3048
DSP_TransformWinInit_Hamm32 function 3048
DSP_TransformWinInit_Hann16 function 3049
DSP_TransformWinInit_Hann32 function 3050
DSP_TransformWinInit_Kaiser16 function 3050
DSP_TransformWinInit_Kaiser32 function 3051
DSP_VectorAbs16 function 3051
DSP_VectorAbs32 function 3052
DSP_VectorAdd16 function 3053
DSP_VectorAdd32 function 3054
DSP_VectorAddc16 function 3054
DSP_VectorAddc32 function 3055
DSP_VectorAutocorr16 function 3056
DSP_VectorBexp16 function 3057
DSP_VectorBexp32 function 3057
DSP_VectorChkEqu32 function 3058
DSP_VectorCopy function 3059
DSP_VectorCopyReverse32 function 3060
DSP_VectorDivC function 3060
DSP_VectorDotp16 function 3061
DSP_VectorDotp32 function 3062
DSP_VectorExp function 3063
DSP_VectorFill function 3063
DSP_VectorLn function 3064
DSP_VectorLog10 function 3065
DSP_VectorLog2 function 3066
DSP_VectorMax32 function 3066
DSP_VectorMaxIndex32 function 3067
DSP_VectorMean32 function 3068
DSP_VectorMin32 function 3068
DSP_VectorMinIndex32 function 3069
DSP_VectorMul16 function 3070
DSP_VectorMul32 function 3071
DSP_VectorMulc16 function 3071
DSP_VectorMulc32 function 3072
DSP_VectorNegate function 3073
DSP_VectorRecip function 3074
DSP_VectorRMS16 function 3075
DSP_VectorShift function 3075
DSP_VectorSqrt function 3076
DSP_VectorStdDev16 function 3077
DSP_VectorSub16 function 3078
DSP_VectorSub32 function 3078
DSP_VectorSumSquares16 function 3079
DSP_VectorSumSquares32 function 3080
DSP_VectorVari16 function 3080
DSP_VectorVariance function 3081
DSP_VectorZeroPad function 3082
Dual Compare Continuous Mode 4483
Dual Data Rate (DDR) SDRAM Peripheral Library 3866
Dynamic Configuration 5838
Dynamic DNS Module 6148
Dynamic Variables 6165
Dynamic Variables Processing 6204

E

EAI_AGAIN macro 6049
EAI_BADFLAGS macro 6049
EAI_FAIL macro 6049
EAI_FAMILY macro 6049
EAI_MEMORY macro 6049
EAI_NONAME macro 6050
EAI_OVERFLOW macro 6050
EAI_SERVICE macro 6050
EAI_SOCKTYPE macro 6050
EAI_SYSTEM macro 6050
EBI Peripheral Library 3935
EBI_ADDRESS_PORT enumeration 3971
EBI_CS_TIMING enumeration 3972
EBI_MEMORY_SIZE enumeration 3972
EBI_MEMORY_TYPE enumeration 3973
EBI_MODULE_ID enumeration 3974
EBI_PAGE_SIZE enumeration 3974
EBI_STATIC_MEMORY_WIDTH enumeration 3974
Edit Box Object 2750
EEPROM Operations 4419
EEPROM_ERROR enumeration 4475
EEPROM_OPERATION_MODE enumeration 4475
Enabling 6668
Enabling Audio Stream 7016
Enabling DMA SPI 2397
Enabling Events 3522
ENC28J60 Driver Library Help 1569
ENCx24J600 Driver Library Help 1589
END_DE_SEQUENCE macro 845
Endpoint Data Transfer 6941
Endpoint Data Transfer Events 6940
Endpoint Management 6940
Enhanced Buffer Master Mode 5019
Enhanced Buffer Slave Mode 5021
Enhanced Buffer SPI Mode 5019
ESC_SEQ_SIZE macro 5643
eSpxFlags enumeration 1314
ETH_AUTOPAD_OPTION enumeration 4127
ETH_INTERRUPT_SOURCES enumeration 4127
ETH_MIIM_CLK enumeration 4128
ETH_PATTERN_MATCH_DISABLED enumeration member 4128
ETH_PATTERN_MATCH_MODE enumeration 4128
ETH_PATTERN_MATCH_MODE_BROADCAST_ADDR enumeration member 4128
ETH_PATTERN_MATCH_MODE_CHECKSUM_MATCH enumeration member 4128
ETH_PATTERN_MATCH_MODE_HASH_MATCH enumeration member 4128
ETH_PATTERN_MATCH_MODE_MAGIC_PACKET enumeration member 4128
ETH_PATTERN_MATCH_MODE_STATION_ADDR enumeration member 4128
ETH_PATTERN_MATCH_MODE_UNICAST_ADDR enumeration member 4128
ETH_RECEIVE_FILTER enumeration 4129
ETH_RMII_SPEED enumeration 4130
Ethernet Controller Operation 3983

Ethernet DMA and Buffer Management Engine 3986
Ethernet Frame Overview 3983
Ethernet MAC Driver Library 1608
Ethernet Peripheral Library 3977
Ethernet PHY Driver Library 1629
Event Handling 6709, 6715, 6720, 6750, 6781, 6861, 6897, 6937, 6964, 7112
Example - Channel Scanning 3309, 3437
Example - CVD Mode 3314
Example - Digital Comparator 3312, 3440
Example - Digital Filter 3310
Example - Digital Oversampling Filter 3439
Example - Simultaneous Sampling and Conversion of Three Class 1 Inputs 3305
Example - Simultaneous Sampling Three Class 1 Inputs 3436
Example 1 2586
Example 2 2587
Example Applications 3662
Example Code for Complete Operation 1873
Example Usage of the Timer Driver 2047
Examples 5942
Examples - Sample Functionality 3206
Exception Generator 3483
Exponent16ToQFloat32 macro 3119
Extended Glyphs 2572
Extended ID Message Format 3518
External Resources Functions 2642

F

Fail-Safe Clock Monitor 4512
false enumeration 6990
FALSE macro 210
FAT_FS_MAX_LFN macro 5827
FAT_FS_MAX_SS macro 5828
FAT_FS_USE_LFN macro 5828
File EOF 5768
File I/O Type Read/Write Data Transfer Model 2337
File Processing 6204
File Seek 5768
File System Service Library 5759
File System Service Library Porting Guide 5760
File Tell 5768
FileFeof type 2958
FileRead type 2958
Files 999, 1189, 1209, 1258, 1269, 1288, 1306, 1321, 1334, 1352, 1364, 1386, 1439, 1480, 1518, 1556, 1568, 1587, 1606, 1627, 1663, 1672, 1722, 1764, 1819, 1862, 1894, 1927, 1964, 2037, 2090, 2109, 2127, 2149, 2179, 2266, 2326, 2393, 2489, 2548, 2967, 3088, 3120, 3154, 3195, 3221, 3299, 3430, 3480, 3513, 3610, 3651, 3683, 3699, 3734, 3862, 3932, 3975, 4130, 4234, 4338, 4365, 4413, 4476, 4506, 4612, 4706, 4757, 4845, 4893, 4909, 4968, 5009, 5102, 5237, 5279, 5364, 5511, 5561, 5575, 5587, 5606, 5632, 5652, 5671, 5689, 5715, 5757, 5831, 5854, 5859, 5884, 5914, 5925, 5931, 5964, 5975, 5980, 6009, 6027, 6061, 6077, 6092, 6111, 6134, 6147, 6155, 6162, 6199, 6262, 6276, 6288, 6297, 6311, 6348, 6397, 6464, 6470, 6478, 6481, 6494, 6514, 6561, 6574, 6613, 6627, 6632, 6664, 6675, 6695, 6745, 6771, 6854, 6891, 6923, 6934, 7003, 7099, 7135, 7146, 7151, 7164
    10-bit ADC Touch Driver Library 2109
    AAC Decoder Library 1269
    ADC Peripheral Library 3299
    ADCHS Peripheral Library 3430
    ADCP Peripheral Library 3480
    AK4384 Codec Driver Library 1439
    AK4642 Codec Driver Library 1480
    AK4953 Codec Driver Library 1518
    AK7755 Codec Driver Library 1556
    Announce Module 6009
    AR1021 Touch Driver Library 2127
    ARP Module 6027
    Berkeley Module 6061
    BMX Peripheral Library 3513
    Bootloader Library 1189
    CAN Peripheral Library 3610
    Class B Library 1209
    Clock System Service Library 5632
    Command Processor System Service Library 5652
    Comparator Peripheral Library 3651
    Console System Service Library 5671
    CPLD XC2C64A Driver Library 1568
    Crypto Library 1258
    CTMU Peripheral Library 3683
    DDR Timer Peripheral Library 3932
    Debug System Service Library 5689
    Device Control Peripheral Library 3734
    Device Control System Service Library 5715
    DHCP Module 6077, 6092
    DHCPv6 Module 6111
    DMA Peripheral Library 3862
    DMA System Service Library 5757
    DMT Timer Peripheral Library 3699
    DNS Module 6134
    DNSS Module 6147
    DSP Fixed-Point Math Library 3088
    Dynamic DNS Module 6155
    EBI Peripheral Library 3975
    Ethernet MAC Driver Library 1627
    Ethernet Peripheral Library 4130
    Ethernet PHY Driver Library 1663
    File System Service Library 5831
    FTP Module 6162
    GLCD Controller Timer Peripheral Library 4234
    Graphics Driver Library 1722
    HTTP Module 6199
    HTTP Net Module 6262
    I2C Peripheral Library 4338
    ICMP Module 6276
    ICMPv6 Module 6288
    Input Capture Peripheral Library 4365
    Interrupt Peripheral Library 4413
    Interrupt System Service Library 5854
    Iperf Module 6297
    IPv4 Module 6311
    IPv6 Module 6348
    LibQ Fixed-Point 'C' Math Library 3120
    LibQ Fixed-Point Math Library 3154
    MAC Driver Module 6397
    Manager Module 6464
    Memory System Service Library 5859
    Messaging System Service Library 5884

MP3 Decoder Library 1288
MRF24W Wi-Fi Driver Library 2489
MRF24WN Wi-Fi Driver Library 2548
MTCH6301 Touch Driver Library 2149
MTCH6303 Touch Driver Library 2179
NBNS Module 6470
NDP Module 6478
Network Presentation Layer 3195
NVM Driver Library 1862
NVM Peripheral Library 4476
Opus Decoder Library 1306
OSAL Library 3221
Oscillator Peripheral Library 4612
Output Compare Peripheral Library 4506
PIC32 Bluetooth Stack Library 999
PMP Driver Library 1894
PMP Peripheral Library 4706
Ports Peripheral Library 4845
Ports System Service Library 5914
Power Peripheral Library 4893
Prefetch Peripheral Library 4757
Random Number Generator System Service Library 5925
Reboot Module 6481
Reset Peripheral Library 4909
Reset System Service 5931
RTCC Peripheral Library 4968
SAMPLE Library 5587
SD Card Driver Library 1927
SMTP Module 6494
SMTPC Module 6514
SNMP Module 6561
SNTP Module 6574
Speex Decoder Library 1321
SPI Driver Library 1964
SPI Flash Driver Library 2037
SPI Peripheral Library 5102
SQI Peripheral Library 5237
System Bus Peripheral Library 5009
System Service Library Overview 5606
TCP Module 6613
Telnet Module 6632
TEST HARNESS LIB Library 6695
TFTP Module 6627
Timer Driver Library 2090
Timer Peripheral Library 5279
Timer System Service Library 5964
Touch System Service Library 5975
UDP Module 6664
USART Driver Library 2393
USART Peripheral Library 5364
USB Audio 1.0 Device Library 6745
USB Audio 2.0 Device Library 6771
USB Audio v1.0 Host Client Driver Library 7099
USB CDC Device Library 6891
USB Device Layer Library 6854
USB HID Device Library 6923
USB HID Host Mouse Driver Library 7146
USB Host Layer Library 7003
USB Hub Host Client Driver Library 7151
USB MSD Device Library 6934
USB Peripheral Library 5511
USBHS Peripheral Library 5561
Watchdog Timer Peripheral Library 5575
Watchdog Timer System Service Library 5980
WMA Decoder Library 1334
Zeroconf Module 6675
Files to Include
USB CDC Device Library 6870
USB Device Stack Porting 6707
FileSeek type 2958
FileTell type 2959
Filled Polygon Rendering 2568
Filters and Masks Configuration 3521
Fl2Fract16 macro 3109
Fl2Fract32 macro 3110
Fl2FxPnt macro 3110
Fl2FxPnt16 macro 3111
Fl2FxPnt32 macro 3111
Fl2Int16 macro 3111
Fl2Int32 macro 3112
Fl2QFloat32 macro 3119
Flash and RAM Usage 5983
Flash Driver Library 1665
Flash ECC Operations 4715
Flash Operations 4419
Flash_CRC16 function 1205
FLASH_CRC16_MASK macro 1207
FLASH_CRC16_MSB macro 1208
FLASH_CRC32_MASK macro 1208
FLASH_CRC32_MSB macro 1208
FLASH_CRC8_MASK macro 1208
FLASH_CRC8_MSB macro 1208
Flow Control Overview 3985
Form Processing 6166
Forming Transfers 4257
frame_buffer.h 1103
FRAME_SIZE macro 1267
Framed SPI Modes 5022
Framework Overview 3
freeaddrinfo function 6043
FrMax macro 3112
FrMin macro 3112
FTP Module 6157
ftp.h 6162
FTP_ALLOCATE_BUFFERS function 16
ftp_config.h 6162
Function Driver Registration Table 6777
Functionality 3305
Fuse Configuration and Initialization 6713
Fx16Norm function 3107
Fx32Norm function 3108
FxQFloat32 variable 3118

G

gap.h 1104
Gated Timer 5248

GATT_CLIENT_ALLOCATE_BUFFERS function 16
General 3241
General Configuration 3738, 4137
General Device Mode Operations 2202
General Setup 3658
Generating the Application Linker Script 1179
Generic USB Device Library 6935
GET_F_BIT macro 732
GET_FRAME_TYPE macro 732
GET_P_BIT macro 733
GET_REQ_SEQ macro 733
GET_S_FUNCTION macro 733
GET_SAR macro 733
GET_TX_SEQ macro 733
getaddrinfo function 6043
gethostbyname function 6041
gethostname function 6036
GetReadBytesInAppData type 1333
GETRESP_BAD_REQ macro 929
GETRESP_ERROR macro 929
GETRESP_NO_ACCESS macro 930
GETRESP_NO_SUCH_VARID macro 930
GETRESP_NO_VALUE macro 930
GETRESP_OK macro 930
GETRESP_PERMISSION_DENIED macro 930
GETRESP_READ_ONLY macro 931
GETRESP_TOO_BIG macro 931
GETRESP_WRITE_ONLY macro 931
getsockname function 6042
getsockopt function 6040
Getting Help 5985
Getting Started
    USB Device Library 6697
    USB Host Library 6943
gfx.h 2968
GFX_ALIGNMENT enumeration 2948
GFX_AlphaBlendingValueGet function 2621
GFX_AlphaBlendingValueSet function 2622
GFX_AT_SCAN_CODES enumeration 2916
GFX_BACKGROUND structure 2948
GFX_BACKGROUND_TYPE enumeration 2949
GFX_BackgroundColorGet function 2629
GFX_BackgroundImageGet function 2629
GFX_BackgroundImageLeftGet function 2630
GFX_BackgroundImageTopGet function 2630
GFX_BackgroundSet function 2631
GFX_BackgroundTypeGet function 2631
GFX_BackgroundTypeSet function 2632
GFX_BarDraw function 2602
GFX_CircleDraw function 2599
GFX_CircleFillDraw function 2604
GFX_CLIENT_OBJ structure 2956
GFX_CLIENT_STATUS enumeration 2956
GFX_Close function 2925
GFX_ColorGet function 2625
gfx_colors.h 2969
gfx_colors_w3.h 2970
gfx_colors_x11.h 2973
GFX_ColorSet function 2626
gfx_common.h 2970
GFX_COMP_MASK macro 2952
GFX_CONFIG_ALPHABLEND_DISABLE macro 2577
GFX_CONFIG_COLOR_DEPTH macro 2578
GFX_CONFIG_DOUBLE_BUFFERING_DISABLE macro 2578
GFX_CONFIG_FOCUS_DISABLE macro 2578
GFX_CONFIG_FONT_ANTIALIASED_DISABLE macro 2579
GFX_CONFIG_FONT_CHAR_SIZE macro 2579
GFX_CONFIG_FONT_EXTERNAL_DISABLE macro 2580
GFX_CONFIG_FONT_FLASH_DISABLE macro 2580
GFX_CONFIG_FONT_RAM_DISABLE macro 2580
GFX_CONFIG_GRADIENT_DISABLE macro 2581
GFX_CONFIG_IMAGE_EXTERNAL_DISABLE macro 2581
GFX_CONFIG_IMAGE_FLASH_DISABLE macro 2585
GFX_CONFIG_IMAGE_PADDING_DISABLE macro 2581
GFX_CONFIG_IMAGE_RAM_DISABLE macro 2582
GFX_CONFIG_IPU_DECODE_DISABLE macro 2582
GFX_CONFIG_OTM2201A_DRIVER_COUNT macro 1678
GFX_CONFIG_PALETTE_DISABLE macro 2582
GFX_CONFIG_PALETTE_EXTERNAL_DISABLE macro 2583
GFX_CONFIG_RLE_DECODE_DISABLE macro 2583
GFX_CONFIG_S1D13517_DRIVER_COUNT macro 1677
GFX_CONFIG_SSD1926_DRIVER_COUNT macro 1678
gfx_config_template.h 2997
GFX_CONFIG_TRANSPARENT_COLOR_DISABLE macro 2583
GFX_CONFIG_USE_KEYBOARD_DISABLE macro 2584
GFX_CONFIG_USE_TOUCHSCREEN_DISABLE macro 2584
GFX_Deinitialize function 2926
GFX_DOUBLE_BUFFERING_MODE structure 2949
GFX_DoubleBufferAreaGet function 2635
GFX_DoubleBufferAreaMark function 2633
GFX_DoubleBufferDisable function 2634
GFX_DoubleBufferEnable function 2633
GFX_DoubleBufferPause function 2641
GFX_DoubleBufferResume function 2641
GFX_DoubleBufferStatusGet function 2634
GFX_DoubleBufferSyncAllStatusClear function 2635
GFX_DoubleBufferSyncAllStatusGet function 2636
GFX_DoubleBufferSyncAllStatusSet function 2636
GFX_DoubleBufferSyncAreaCountGet function 2636
GFX_DoubleBufferSyncAreaCountSet function 2637
GFX_DoubleBufferSynchronize function 2638
GFX_DoubleBufferSynchronizeRequest function 2637
GFX_DoubleBufferSynchronizeStatusGet function 2638
GFX_DrawBufferGet function 2639
GFX_DrawBufferInitialize function 2639
GFX_DrawBufferSet function 2640
GFX_EXTERNAL_FONT_RASTER_BUFFER_SIZE macro 2584
GFX_ExternalResourceCallback function 2642
GFX_FEATURE_STATUS enumeration 2950
GFX_FILL_STYLE enumeration 2950
GFX_FillStyleGet function 2620
GFX_FillStyleSet function 2621
GFX_FONT_ANTIALIAS_TYPE enumeration 2951
GFX_FONT_GLYPH_ENTRY structure 2934
GFX_FONT_GLYPH_ENTRY_EXTENDED structure 2935
GFX_FONT_HEADER structure 2935

GFX_FONT_SPACE macro 2951
GFX_FontAntiAliasGet function 2611
GFX_FontAntiAliasSet function 2611
GFX_FontGet function 2613
GFX_FontSet function 2612
GFX_FrameBufferGet function 2640
GFX_FrameBufferSet function 2640
GFX_free macro 2585
gfx_gol.h 2976
GFX_GOL_BUTTON structure 2886
gfx_gol_button.h 2978
GFX_GOL_BUTTON_STATE enumeration 2887
GFX_GOL_ButtonActionGet function 2726
GFX_GOL_ButtonActionSet function 2725
GFX_GOL_ButtonCreate function 2727
GFX_GOL_ButtonDraw function 2729
GFX_GOL_ButtonPressStateImageGet macro 2723
GFX_GOL_ButtonPressStateImageSet macro 2723
GFX_GOL_ButtonReleaseStateImageGet macro 2724
GFX_GOL_ButtonReleaseStateImageSet macro 2724
GFX_GOL_ButtonTextAlignmentGet function 2730
GFX_GOL_ButtonTextAlignmentSet function 2730
GFX_GOL_ButtonTextGet macro 2725
GFX_GOL_ButtonTextSet function 2731
gfx_gol_check_box.h 2979
GFX_GOL_CHECKBOX structure 2887
GFX_GOL_CHECKBOX_STATE enumeration 2888
GFX_GOL_CheckBoxActionGet function 2733
GFX_GOL_CheckBoxActionSet function 2734
GFX_GOL_CheckBoxCreate function 2734
GFX_GOL_CheckBoxDraw function 2736
GFX_GOL_CheckBoxTextAlignmentGet function 2736
GFX_GOL_CheckBoxTextAlignmentSet function 2737
GFX_GOL_CheckBoxTextGet macro 2732
GFX_GOL_CheckBoxTextSet function 2737
GFX_GOL_CLIENT_STATUS enumeration 2915
GFX_GOL_COMMON_STATE_BITS enumeration 2912
gfx_gol_custom_control.h 2979
GFX_GOL_CUSTOMCONTROL structure 2888
GFX_GOL_CUSTOMCONTROL_STATE enumeration 2914
GFX_GOL_CustomControlActionGet function 2739
GFX_GOL_CustomControlActionSet function 2740
GFX_GOL_CustomControlCreate function 2741
GFX_GOL_CustomControlDraw function 2742
GFX_GOL_CustomControlGetPos macro 2738
GFX_GOL_CustomControlSetPos macro 2739
GFX_GOL_Deinitialize function 2928
gfx_gol_digital_meter.h 2980
GFX_GOL_DIGITALMETER structure 2889
GFX_GOL_DIGITALMETER_STATE enumeration 2890
GFX_GOL_DigitalMeterActionGet function 2745
GFX_GOL_DigitalMeterCreate function 2746
GFX_GOL_DigitalMeterDecrement function 2747
GFX_GOL_DigitalMeterDraw function 2747
GFX_GOL_DigitalMeterIncrement function 2748
GFX_GOL_DigitalMeterTextAlignmentGet macro 2743
GFX_GOL_DigitalMeterTextAlignmentSet macro 2743
GFX_GOL_DigitalMeterValueGet macro 2744
GFX_GOL_DigitalMeterValueSet function 2749
GFX_GOL_DRAW_CALLBACK_FUNC type 2891
GFX_GOL_DrawCallbackSet function 2875
gfx_gol_edit_box.h 2981
GFX_GOL_EDITBOX structure 2891
GFX_GOL_EDITBOX_STATE enumeration 2910
GFX_GOL_EditBoxActionGet function 2752
GFX_GOL_EditBoxActionSet function 2753
GFX_GOL_EditBoxCharAdd function 2754
GFX_GOL_EditBoxCharRemove function 2755
GFX_GOL_EditBoxCreate function 2755
GFX_GOL_EditBoxCursorPositionDecrement function 2758
GFX_GOL_EditBoxCursorPositionIncrement function 2758
GFX_GOL_EditBoxDraw function 2756
GFX_GOL_EditBoxTextAlignmentGet macro 2751
GFX_GOL_EditBoxTextAlignmentSet macro 2751
GFX_GOL_EditBoxTextGet macro 2752
GFX_GOL_EditBoxTextSet function 2757
gfx_gol_group_box.h 2981
GFX_GOL_GROUPBOX structure 2892
GFX_GOL_GROUPBOX_STATE enumeration 2892
GFX_GOL_GroupboxActionGet function 2761
GFX_GOL_GroupboxCreate function 2762
GFX_GOL_GroupboxDraw function 2761
GFX_GOL_GroupboxTextAlignmentGet function 2763
GFX_GOL_GroupboxTextAlignmentSet function 2764
GFX_GOL_GroupboxTextGet macro 2760
GFX_GOL_GroupboxTextSet function 2764
GFX_GOL_Initialize function 2929
gfx_gol_list_box.h 2982
GFX_GOL_LISTBOX structure 2893
GFX_GOL_LISTBOX_ITEM_STATUS enumeration 2893
GFX_GOL_LISTBOX_STATE enumeration 2894
GFX_GOL_ListBoxActionGet function 2770
GFX_GOL_ListBoxActionSet function 2771
GFX_GOL_ListBoxCreate function 2778
GFX_GOL_ListBoxDraw function 2772
GFX_GOL_ListBoxItemAdd function 2773
GFX_GOL_ListBoxItemCountGet macro 2766
GFX_GOL_ListBoxItemFocusGet function 2774
GFX_GOL_ListBoxItemFocusSet function 2775
GFX_GOL_ListBoxItemImageGet macro 2766
GFX_GOL_ListBoxItemImageSet macro 2767
GFX_GOL_ListBoxItemListGet macro 2767
GFX_GOL_ListBoxItemListRemove function 2775
GFX_GOL_ListBoxItemRemove function 2776
GFX_GOL_ListBoxItemSelectStatusClear macro 2768
GFX_GOL_ListBoxItemSelectStatusSet macro 2769
GFX_GOL_ListBoxSelectionChange function 2776
GFX_GOL_ListBoxSelectionGet function 2777
GFX_GOL_ListBoxTextAlignmentGet macro 2769
GFX_GOL_ListBoxTextAlignmentSet macro 2770
GFX_GOL_ListBoxVisibleItemCountGet function 2778
GFX_GOL_LISTITEM structure 2894
GFX_GOL_MESSAGE structure 2895
GFX_GOL_MESSAGE_CALLBACK_FUNC type 2913
GFX_GOL_MessageCallbackSet function 2883
GFX_GOL_METER structure 2896

gfx_gol_meter.h 2983
GFX_GOL_METER_DRAW_TYPE enumeration 2897
GFX_GOL_METER_STATE enumeration 2897
GFX_GOL_MeterActionGet function 2785
GFX_GOL_MeterActionSet function 2785
GFX_GOL_MeterCreate function 2786
GFX_GOL_MeterDecrement function 2788
GFX_GOL_MeterDraw function 2788
GFX_GOL_MeterIncrement function 2789
GFX_GOL_MeterMaximumValueGet macro 2781
GFX_GOL_MeterMinimumValueGet macro 2782
GFX_GOL_MeterRangeSet function 2790
GFX_GOL_MeterScaleColorsSet function 2790
GFX_GOL_MeterTitleFontSet macro 2782
GFX_GOL_MeterTypeSet macro 2783
GFX_GOL_MeterValueFontSet macro 2783
GFX_GOL_MeterValueGet macro 2784
GFX_GOL_MeterValueSet function 2791
GFX_GOL_OBJ_HEADER structure 2898
GFX_GOL_OBJ_SCHEME structure 2919
GFX_GOL_OBJ_TYPE enumeration 2899
GFX_GOL_OBJECT_TASK enumeration 2915
GFX_GOL_ObjectAdd function 2865
GFX_GOL_ObjectBackGroundSet function 2881
GFX_GOL_ObjectByIDDelete function 2866
GFX_GOL_ObjectCanBeFocused function 2866
GFX_GOL_ObjectDelete function 2867
GFX_GOL_ObjectDrawDisable function 2876
GFX_GOL_ObjectDrawEnable function 2876
GFX_GOL_ObjectFind function 2867
GFX_GOL_ObjectFocusGet function 2868
GFX_GOL_ObjectFocusNextGet function 2868
GFX_GOL_ObjectFocusPrevGet function 2869
GFX_GOL_ObjectFocusSet function 2870
GFX_GOL_ObjectHideDraw function 2880
GFX_GOL_ObjectIDGet function 2870
GFX_GOL_ObjectIsRedrawSet function 2877
GFX_GOL_ObjectListDraw function 2878
GFX_GOL_ObjectListFree function 2871
GFX_GOL_ObjectListGet function 2872
GFX_GOL_ObjectListHide function 2879
GFX_GOL_ObjectListNew function 2872
GFX_GOL_ObjectListSet function 2873
GFX_GOL_ObjectMessage function 2882
GFX_GOL_ObjectNextGet function 2874
GFX_GOL_ObjectRectangleRedraw function 2879
GFX_GOL_ObjectStateClear macro 2862
GFX_GOL_ObjectStateGet macro 2863
GFX_GOL_ObjectStateSet macro 2864
GFX_GOL_ObjectStyleSchemeGet macro 2883
GFX_GOL_ObjectStyleSchemeSet macro 2884
GFX_GOL_ObjectTypeGet function 2874
GFX_GOL_Open function 2929
GFX_GOL_PanelAlphaParameterSet function 2792
GFX_GOL_PanelBackgroundSet function 2793
GFX_GOL_PanelDraw function 2793
GFX_GOL_PanelGradientParameterSet function 2794
GFX_GOL_PanelParameterSet function 2794
gfx_gol_picture.h 2984
GFX_GOL_PICTURECONTROL structure 2911
GFX_GOL_PictureControlActionGet function 2797
GFX_GOL_PICTURECONTROLCONTROL_STATE enumeration 2912
GFX_GOL_PictureControlCreate function 2798
GFX_GOL_PictureControlDraw function 2799
GFX_GOL_PictureControlImageGet macro 2796
GFX_GOL_PictureControlImageSet macro 2797
GFX_GOL_PictureControlPartialSet function 2800
GFX_GOL_PictureControlScaleSet function 2801
GFX_GOL_PreemptionLevelGet function 2930
gfx_gol_progress_bar.h 2984
GFX_GOL_PROGRESSBAR structure 2900
GFX_GOL_PROGRESSBAR_STATE enumeration 2900
GFX_GOL_ProgressBarActionGet function 2803
GFX_GOL_ProgressBarCreate function 2804
GFX_GOL_ProgressBarDraw function 2805
GFX_GOL_ProgressBarPositionGet macro 2802
GFX_GOL_ProgressBarPositionSet function 2806
GFX_GOL_ProgressBarRangeGet macro 2803
GFX_GOL_ProgressBarRangeSet function 2807
gfx_gol_radio_button.h 2985
GFX_GOL_RADIOBUTTON structure 2901
GFX_GOL_RADIOBUTTON_STATE enumeration 2902
GFX_GOL_RadioButtonActionGet function 2810
GFX_GOL_RadioButtonActionSet function 2811
GFX_GOL_RadioButtonCheckGet function 2809
GFX_GOL_RadioButtonCheckSet function 2812
GFX_GOL_RadioButtonCreate function 2814
GFX_GOL_RadioButtonDraw function 2812
GFX_GOL_RadioButtonListCreate function 2816
GFX_GOL_RadioButtonTextAlignmentGet function 2813
GFX_GOL_RadioButtonTextAlignmentSet function 2813
GFX_GOL_RadioButtonTextGet macro 2808
GFX_GOL_RadioButtonTextSet function 2814
gfx_gol_scan_codes.h 2986
gfx_gol_scheme.h 2986
GFX_GOL_SchemeAlphaPrecentSet function 2817
GFX_GOL_SchemeBackgroundColorGet function 2817
GFX_GOL_SchemeBackgroundColorSet function 2817
GFX_GOL_SchemeBackgroundImageSet function 2818
GFX_GOL_SchemeBackgroundTypeGet function 2818
GFX_GOL_SchemeBackgroundTypeSet function 2818
GFX_GOL_SchemeColor0Get function 2818
GFX_GOL_SchemeColor0Set function 2818
GFX_GOL_SchemeColor1Get function 2819
GFX_GOL_SchemeColor1Set function 2819
GFX_GOL_SchemeColorDisabledGet function 2819
GFX_GOL_SchemeColorDisabledSet function 2819
GFX_GOL_SchemeColorSet function 2819
GFX_GOL_SchemeEmbossDarkColorGet function 2819
GFX_GOL_SchemeEmbossDarkColorSet function 2820
GFX_GOL_SchemeEmbossLightColorGet function 2820
GFX_GOL_SchemeEmbossLightColorSet function 2820
GFX_GOL_SchemeEmbossSet function 2820
GFX_GOL_SchemeEmbossSizeGet function 2820
GFX_GOL_SchemeEmbossSizeSet function 2820
GFX_GOL_SchemeFillStyleGet function 2821

GFX_GOL_SchemeFillStyleSet function 2821
GFX_GOL_SchemeFontGet function 2821
GFX_GOL_SchemeFontSet function 2821
GFX_GOL_SchemeGradientColorSet function 2821
GFX_GOL_SchemeGradientEndColorGet function 2822
GFX_GOL_SchemeGradientEndColorSet function 2822
GFX_GOL_SchemeGradientStartColorGet function 2822
GFX_GOL_SchemeGradientStartColorSet function 2822
GFX_GOL_SchemeTextColor0Get function 2822
GFX_GOL_SchemeTextColor0Set function 2822
GFX_GOL_SchemeTextColor1Get function 2823
GFX_GOL_SchemeTextColor1Set function 2823
GFX_GOL_SchemeTextColorDisableGet function 2823
GFX_GOL_SchemeTextColorDisableSet function 2823
GFX_GOL_SchemeTextColorSet function 2823
gfx_gol_scroll_bar.h 2987
GFX_GOL_SCROLLBAR structure 2902
GFX_GOL_SCROLLBAR_STATE enumeration 2903
GFX_GOL_ScrollBarActionGet function 2824
GFX_GOL_ScrollBarActionSet function 2825
GFX_GOL_ScrollBarCreate function 2826
GFX_GOL_ScrollBarDraw function 2828
GFX_GOL_ScrollBarPageGet function 2830
GFX_GOL_ScrollBarPageSet function 2830
GFX_GOL_ScrollBarPositionDecrement function 2829
GFX_GOL_ScrollBarPositionGet function 2831
GFX_GOL_ScrollBarPositionIncrement function 2832
GFX_GOL_ScrollBarPositionSet function 2832
GFX_GOL_ScrollBarRangeGet function 2833
GFX_GOL_ScrollBarRangeSet function 2834
GFX_GOL_STATES enumeration 2916
gfx_gol_static_text.h 2988
GFX_GOL_STATICTEXT structure 2904
GFX_GOL_STATICTEXT_STATE enumeration 2904
GFX_GOL_StaticTextActionGet function 2835
GFX_GOL_StaticTextAlignmentGet function 2836
GFX_GOL_StaticTextAlignmentSet function 2836
GFX_GOL_StaticTextCreate function 2837
GFX_GOL_StaticTextDraw function 2838
GFX_GOL_StaticTextGet function 2839
GFX_GOL_StaticTextSet function 2839
GFX_GOL_SURFACE structure 2918
gfx_gol_surface.h 2988
GFX_GOL_SURFACE_CALLBACK_FUNC type 2918
GFX_GOL_SURFACE_STATE enumeration 2919
GFX_GOL_SurfaceActionGet function 2840
GFX_GOL_SurfaceCreate function 2841
GFX_GOL_SurfaceDraw function 2842
GFX_GOL_SurfaceImageGet macro 2920
GFX_GOL_SurfaceImageSet macro 2921
GFX_GOL_Tasks function 2930
gfx_gol_text_entry.h 2989
GFX_GOL_TEXTENTRY structure 2905
GFX_GOL_TEXTENTRY_KEY_COMMAND_TYPE enumeration 2905
GFX_GOL_TEXTENTRY_KEYMEMBER structure 2914
GFX_GOL_TEXTENTRY_STATE enumeration 2906
GFX_GOL_TextEntryActionGet function 2844
GFX_GOL_TextEntryActionSet function 2845
GFX_GOL_TextEntryBufferClear function 2847
GFX_GOL_TextEntryBufferGet function 2846
GFX_GOL_TextEntryBufferSet function 2847
GFX_GOL_TextEntryCharAdd function 2848
GFX_GOL_TextEntryCreate function 2848
GFX_GOL_TextEntryDraw function 2850
GFX_GOL_TextEntryKeyCommandGet function 2851
GFX_GOL_TextEntryKeyCommandSet function 2851
GFX_GOL_TextEntryKeyIsPressed function 2852
GFX_GOL_TextEntryKeyListCreate function 2852
GFX_GOL_TextEntryKeyMemberListDelete function 2853
GFX_GOL_TextEntryKeyTextSet function 2854
GFX_GOL_TextEntryLastCharDelete function 2854
GFX_GOL_TextEntrySpaceCharAdd function 2855
GFX_GOL_TRANSLATED_ACTION enumeration 2907
GFX_GOL_TwoTonePanelDraw function 2881
GFX_GOL_WINDOW structure 2908
gfx_gol_window.h 2990
GFX_GOL_WINDOW_STATE enumeration 2909
GFX_GOL_WindowActionGet function 2858
GFX_GOL_WindowCreate function 2859
GFX_GOL_WindowDraw function 2860
GFX_GOL_WindowImageGet macro 2856
GFX_GOL_WindowImageSet macro 2857
GFX_GOL_WindowTextAlignmentGet function 2860
GFX_GOL_WindowTextAlignmentSet function 2861
GFX_GOL_WindowTextGet macro 2857
GFX_GOL_WindowTextSet function 2861
GFX_GradientColorSet function 2623
GFX_GradientEndColorGet function 2623
GFX_GradientStartColorGet function 2624
gfx_image_decoder.h 2991
GFX_ImageDraw function 2607
GFX_ImageHeaderGet function 2608
GFX_ImageHeightGet function 2608
GFX_ImagePartialDraw function 2609
GFX_ImageWidthGet function 2610
GFX_Initialize function 2925
GFX_INSTANCES_NUMBER macro 2586
GFX_LineDraw function 2594
GFX_LinePositionRelativeSet function 2595
GFX_LinePositionSet function 2596
GFX_LinePositionXGet function 2596
GFX_LinePositionYGet function 2597
GFX_LineStyleGet function 2624
GFX_LineStyleSet function 2624
GFX_LineToDraw function 2597
GFX_LineToRelativeDraw function 2598
GFX_malloc macro 2585
GFX_MaxXGet function 2593
GFX_MaxYGet function 2593
GFX_MCHP_BITMAP_HEADER structure 2936
GFX_MEM_MASK macro 2953
GFX_OBJ structure 2957
GFX_OBJECT_TASK enumeration 2957
GFX_Open function 2924
GFX_PALETTE_ENTRY union 2965
GFX_PARTIAL_IMAGE_PARAM structure 2936

GFX_PolygonDraw function 2600
GFX_PRIM_SetPIPWindow function 1691
gfx_primitive.h 2992
GFX_PRIMITIVE_Close function 2592
GFX_PRIMITIVE_Deinitialize function 2591
GFX_PRIMITIVE_HANDLE type 2643
GFX_PRIMITIVE_HANDLE_INVALID macro 2643
GFX_PRIMITIVE_Initialize function 2590
GFX_PRIMITIVE_Open function 2592
GFX_PRIMITIVE_STATES enumeration 2643
GFX_RectangleDraw function 2600
GFX_RectangleFillDraw function 2604
GFX_RectangleRoundDraw function 2601
GFX_RectangleRoundFillDraw function 2606
GFX_RECTANGULAR_AREA structure 2937
GFX_RESOURCE enumeration 2937
GFX_RESOURCE_BINARY structure 2941
GFX_RESOURCE_FONT structure 2942
GFX_RESOURCE_HDR structure 2943
GFX_RESOURCE_IMAGE structure 2944
GFX_RESOURCE_PALETTE structure 2945
GFX_ScreenClear function 2592
GFX_STATES enumeration 2958
GFX_STATUS enumeration 2945
GFX_Status function 2927
GFX_STATUS_BIT enumeration 2946
GFX_Tasks function 2927
GFX_TCON_SSD1289Init function 1706
GFX_TEXT_CURSOR_HEIGHT_DEFAULT macro 2644
GFX_TEXT_CURSOR_TYPE enumeration 2965
GFX_TEXT_CURSOR_WIDTH_DEFAULT macro 2644
GFX_TextCharDraw function 2613
GFX_TextCursorDraw function 2619
GFX_TextCursorPositionSet function 2614
GFX_TextCursorPositionXGet function 2614
GFX_TextCursorPositionYGet function 2615
GFX_TextCursorTypeGet function 2619
GFX_TextCursorTypeSet function 2620
GFX_TextCursorWidthGet function 2620
GFX_TextStringBoxDraw function 2615
GFX_TextStringDraw function 2617
GFX_TextStringHeightGet function 2618
GFX_TextStringWidthGet function 2618
GFX_TransparentColorDisable function 2626
GFX_TransparentColorEnable function 2627
GFX_TransparentColorGet function 2628
GFX_TransparentColorStatusGet function 2628
GFX_TYPE_MASK macro 2953
gfx_types_font.h 2994
gfx_types_image.h 2995
gfx_types_macros.h 2996
gfx_types_palette.h 2996
gfx_types_resource.h 2997
GFX_UXCHAR macro 2952
GFX_W3_ALICEBLUE macro 2653
GFX_W3_ANTIQUEWHITE macro 2654
GFX_W3_AQUA macro 2654
GFX_W3_AQUAMARINE macro 2654
GFX_W3_AZURE macro 2654
GFX_W3_BEIGE macro 2654
GFX_W3_BISQUE macro 2655
GFX_W3_BLACK macro 2655
GFX_W3_BLANCHEDALMOND macro 2655
GFX_W3_BLUE macro 2655
GFX_W3_BLUEVIOLET macro 2655
GFX_W3_BROWN macro 2656
GFX_W3_BURLYWOOD macro 2656
GFX_W3_CADETBLUE macro 2656
GFX_W3_CHARTREUSE macro 2656
GFX_W3_CHOCOLATE macro 2656
GFX_W3_CORAL macro 2657
GFX_W3_CORNFLOWERBLUE macro 2657
GFX_W3_CORNSILK macro 2657
GFX_W3_CRIMSON macro 2657
GFX_W3_CYAN macro 2657
GFX_W3_DARKBLUE macro 2658
GFX_W3_DARKCYAN macro 2658
GFX_W3_darkgoldenrod macro 2658
GFX_W3_DARKGRAY macro 2658
GFX_W3_DARKGREEN macro 2658
GFX_W3_DARKGREY macro 2659
GFX_W3_DARKHAKI macro 2659
GFX_W3_DARKMAGENTA macro 2659
GFX_W3_DARKOLIVEGREEN macro 2659
GFX_W3_DARKORANGE macro 2659
GFX_W3_DARKORCHID macro 2660
GFX_W3_DARKRED macro 2660
GFX_W3_DARKSALMON macro 2660
GFX_W3_DARKSEAGREEN macro 2660
GFX_W3_DARKSLATEBLUE macro 2660
GFX_W3_DARKSLATEGRAY macro 2661
GFX_W3_DARKSLATEGREY macro 2661
GFX_W3_DARKTURQUOISE macro 2661
GFX_W3_DARKVIOLET macro 2661
GFX_W3_DEEPPINK macro 2661
GFX_W3_DEEPSKYBLUE macro 2662
GFX_W3_DIMGRAY macro 2662
GFX_W3_DIMGREY macro 2662
GFX_W3_DODGERBLUE macro 2662
GFX_W3_FIREBRICK macro 2662
GFX_W3_FLORALWHITE macro 2663
GFX_W3_FORESTGREEN macro 2663
GFX_W3_FUCHSIA macro 2663
GFX_W3_GAINSBORO macro 2663
GFX_W3_GHOSTWHITE macro 2663
GFX_W3_GOLD macro 2664
GFX_W3_GOLDENROD macro 2664
GFX_W3_GRAY macro 2664
GFX_W3_GREEN macro 2664
GFX_W3_GREENYELLOW macro 2664
GFX_W3_GREY macro 2665
GFX_W3_HONEYDEW macro 2665
GFX_W3_HOTPINK macro 2665
GFX_W3_INDIANRED macro 2665
GFX_W3_INDIGO macro 2665
GFX_W3_IVORY macro 2666

GFX_W3_LAVENDER macro 2666
GFX_W3_LAVENDERBLUSH macro 2666
GFX_W3_LAWNGREEN macro 2666
GFX_W3_LEMONCHIFFON macro 2666
GFX_W3_LIGHTBLUE macro 2667
GFX_W3_LIGHTCORAL macro 2667
GFX_W3_LIGHTCYAN macro 2667
GFX_W3_LIGHTGOLDENRODYELLOW macro 2667
GFX_W3_LIGHTGRAY macro 2667
GFX_W3_LIGHTGREEN macro 2668
GFX_W3_LIGHTGREY macro 2668
GFX_W3_LIGHTPINK macro 2668
GFX_W3_LIGHTSALMON macro 2668
GFX_W3_LIGHTSKYBLUE macro 2668
GFX_W3_LIGHTSLATEGRAY macro 2669
GFX_W3_LIGHTSLATEGREY macro 2669
GFX_W3_LIGHTSTEELBLUE macro 2669
GFX_W3_LIGHTYELLOW macro 2669
GFX_W3_LIGTHSEAGREEN macro 2669
GFX_W3_LIME macro 2670
GFX_W3_LIMEGREEN macro 2670
GFX_W3_LINEN macro 2670
GFX_W3_MAGENTA macro 2670
GFX_W3_MAROON macro 2670
GFX_W3_MEDIUMAQUAMARINE macro 2671
GFX_W3_MEDIUMBLUE macro 2671
GFX_W3_MEDIUMORCHID macro 2671
GFX_W3_MEDIUMPURPLE macro 2671
GFX_W3_MEDIUMSEAGREEN macro 2671
GFX_W3_MEDIUMSLATEBLUE macro 2672
GFX_W3_MEDIUMSPRINGGREEN macro 2672
GFX_W3_MEDIUMTURQUOISE macro 2672
GFX_W3_MEDIUMVIOLETRED macro 2672
GFX_W3_MIDNIGHTBLUE macro 2672
GFX_W3_MINTCREAM macro 2673
GFX_W3_MISTYROSE macro 2673
GFX_W3_MOCCASIN macro 2673
GFX_W3_NAVAJOWHITE macro 2673
GFX_W3_NAVY macro 2673
GFX_W3_OLDLACE macro 2674
GFX_W3_OLIVE macro 2674
GFX_W3_OLIVEDRAB macro 2674
GFX_W3_ORANGE macro 2674
GFX_W3_ORANGERED macro 2674
GFX_W3_ORCHID macro 2675
GFX_W3_PALEGOLDENROD macro 2675
GFX_W3_PALEGREEN macro 2675
GFX_W3_PALETURQUOISE macro 2675
GFX_W3_PALEVIOLETRED macro 2675
GFX_W3_PAPAYAWHIP macro 2676
GFX_W3_PEACHPUFF macro 2676
GFX_W3_PERU macro 2676
GFX_W3_PINK macro 2676
GFX_W3_PLUM macro 2676
GFX_W3_POWDERBLUE macro 2677
GFX_W3_PURPLE macro 2677
GFX_W3_RED macro 2677
GFX_W3_ROSYBROWN macro 2677
GFX_W3_ROYALBLUE macro 2677
GFX_W3_SADDLEBROWN macro 2678
GFX_W3_SALMON macro 2678
GFX_W3_SANDYGREEN macro 2678
GFX_W3_SEAGREEN macro 2678
GFX_W3_SEASHELL macro 2678
GFX_W3_SIENNA macro 2679
GFX_W3_SILVER macro 2679
GFX_W3_SKYBLUE macro 2679
GFX_W3_SLATEBLUE macro 2679
GFX_W3_SLATEGRAY macro 2679
GFX_W3_SLATEGREY macro 2680
GFX_W3_SNOW macro 2680
GFX_W3_SPRINGGREEN macro 2680
GFX_W3_STEELBLUE macro 2680
GFX_W3_TAN macro 2680
GFX_W3_TEAL macro 2681
GFX_W3_THISTLE macro 2681
GFX_W3_TOMATO macro 2681
GFX_W3_TURQUOISE macro 2681
GFX_W3_VIOLET macro 2681
GFX_W3_WHEAT macro 2682
GFX_W3_WHITE macro 2682
GFX_W3_WHITESMOKE macro 2682
GFX_W3_YELLOW macro 2682
GFX_W3_YELLOWGREEN macro 2682
GFX_X11_ALICE_BLUE macro 2683
GFX_X11_ANTIQUE_WHITE macro 2683
GFX_X11_AQUA macro 2683
GFX_X11_AQUAMARINE macro 2683
GFX_X11_AZURE macro 2683
GFX_X11_BEIGE macro 2684
GFX_X11_BISQUE macro 2684
GFX_X11_BLACK macro 2684
GFX_X11_BLANCHED_ALMOND macro 2684
GFX_X11_BLUE macro 2684
GFX_X11_BLUE_VIOLET macro 2685
GFX_X11_BROWN macro 2685
GFX_X11_BURLY_WOOD macro 2685
GFX_X11_CADEL_BLUE macro 2685
GFX_X11_CHARTEUSE macro 2685
GFX_X11_CHOCOLATE macro 2686
GFX_X11_CORAL macro 2686
GFX_X11_CORNFLOWER_BLUE macro 2686
GFX_X11_CORNSILK macro 2686
GFX_X11_CRIMSON macro 2686
GFX_X11_CYAN macro 2687
GFX_X11_DARK_BLUE macro 2687
GFX_X11_DARK_CYAN macro 2687
GFX_X11_DARK_GOLDENROD macro 2687
GFX_X11_DARK_GREEN macro 2687
GFX_X11_DARK_GREY macro 2688
GFX_X11_DARK_KHAKI macro 2688
GFX_X11_DARK_MAGENTA macro 2688
GFX_X11_DARK_OLIVE_GREEN macro 2688
GFX_X11_DARK_ORANGE macro 2688
GFX_X11_DARK_ORCHID macro 2689
GFX_X11_DARK_RED macro 2689

GFX_X11_DARK_SALMON macro 2689
GFX_X11_DARK_SEA_GREEN macro 2689
GFX_X11_DARK_SLATE_BLUE macro 2689
GFX_X11_DARK_SLATE_GREY macro 2690
GFX_X11_DARK_TURQUOISE macro 2690
GFX_X11_DARK_VIOLET macro 2690
GFX_X11_DEEP_PINK macro 2690
GFX_X11_DEEP_SKY_BLUE macro 2690
GFX_X11_DIM_GREY macro 2691
GFX_X11_DODGER_BLUE macro 2691
GFX_X11_FIRE_BRICK macro 2691
GFX_X11_FLORAL_WHITE macro 2691
GFX_X11_FOREST_GREEN macro 2691
GFX_X11_FUCHSIA macro 2692
GFX_X11_GAINSBORO macro 2692
GFX_X11_GHOST_WHITE macro 2692
GFX_X11_GOLD macro 2692
GFX_X11_GOLDENROD macro 2692
GFX_X11_GREEN macro 2693
GFX_X11_GREEN_YELLOW macro 2693
GFX_X11_GREY macro 2693
GFX_X11_HONEYDEW macro 2693
GFX_X11_HOT_PINK macro 2693
GFX_X11_INDIAN_RED macro 2694
GFX_X11_INDIGO macro 2694
GFX_X11_IVORY macro 2694
GFX_X11_KHAKI macro 2694
GFX_X11_LAVENDER macro 2694
GFX_X11_LAVENDOR_BLUSH macro 2695
GFX_X11_LAWN_GREEN macro 2695
GFX_X11_LEMON_CHIFFON macro 2695
GFX_X11_LIGHT_BLUE macro 2695
GFX_X11_LIGHT_CAROL macro 2695
GFX_X11_LIGHT_CYAN macro 2696
GFX_X11_LIGHT_GOLDENROD_YELLOW macro 2696
GFX_X11_LIGHT_GRAY macro 2696
GFX_X11_LIGHT_GREEN macro 2696
GFX_X11_LIGHT_PINK macro 2696
GFX_X11_LIGHT_SALMON macro 2697
GFX_X11_LIGHT_SEA_GREEN macro 2697
GFX_X11_LIGHT_SKY_BLUE macro 2697
GFX_X11_LIGHT_SLATE_GREY macro 2697
GFX_X11_LIGHT_STEEL_BLUE macro 2697
GFX_X11_LIGHT_YELLOW macro 2698
GFX_X11_LIME macro 2698
GFX_X11_LIME_GREEN macro 2698
GFX_X11_LINEN macro 2698
GFX_X11_MAGENTA macro 2698
GFX_X11_MARRON macro 2699
GFX_X11_MEDIUM_AQUAMARINE macro 2699
GFX_X11_MEDIUM_BLUE macro 2699
GFX_X11_MEDIUM_ORCHID macro 2699
GFX_X11_MEDIUM_PURPLE macro 2699
GFX_X11_MEDIUM_SEA_GREEN macro 2700
GFX_X11_MEDIUM_SLATE_BLUE macro 2700
GFX_X11_MEDIUM_SPRING_GREEN macro 2700
GFX_X11_MEDIUM_TURQUOISE macro 2700
GFX_X11_MEDIUM_VIOLET_RED macro 2700
GFX_X11_MIDNIGHT_BLUE macro 2701
GFX_X11_MINT_CREAM macro 2701
GFX_X11_MISTY_ROSE macro 2701
GFX_X11_MOCCASIN macro 2701
GFX_X11_NAVAJO_WHITE macro 2701
GFX_X11_NAVY macro 2702
GFX_X11_OLD_LACE macro 2702
GFX_X11_OLIVE macro 2702
GFX_X11_OLIVE_DRAB macro 2702
GFX_X11_ORANGE macro 2702
GFX_X11_ORANGE_RED macro 2703
GFX_X11_ORCHID macro 2703
GFX_X11_PALE_GOLDENROD macro 2703
GFX_X11_PALE_GREEN macro 2703
GFX_X11_PALE_TURQUOISE macro 2703
GFX_X11_PALE_VIOLET_RED macro 2704
GFX_X11_PAPAYA_WHIP macro 2704
GFX_X11_PEACH_PUFF macro 2704
GFX_X11_PERU macro 2704
GFX_X11_PINK macro 2704
GFX_X11_PLUM macro 2705
GFX_X11_POWDER_BLUE macro 2705
GFX_X11_PURPLE macro 2705
GFX_X11_RED macro 2705
GFX_X11_ROSY_BROWN macro 2705
GFX_X11_ROYAL_BLUE macro 2706
GFX_X11_SADDLE_BROWN macro 2706
GFX_X11_SALMON macro 2706
GFX_X11_SANDY_BROWN macro 2706
GFX_X11_SEA_GREEN macro 2706
GFX_X11_SEASHELL macro 2707
GFX_X11_SIENNA macro 2707
GFX_X11_SILVER macro 2707
GFX_X11_SKY_BLUE macro 2707
GFX_X11_SLATE_BLUE macro 2707
GFX_X11_SLATE_GREY macro 2708
GFX_X11_SNOW macro 2708
GFX_X11_SPRING_GREEN macro 2708
GFX_X11_STEEL_BLUE macro 2708
GFX_X11_TAN macro 2708
GFX_X11_TEAL macro 2709
GFX_X11_THISTLE macro 2709
GFX_X11_TOMATO macro 2709
GFX_X11_TURQUOISE macro 2709
GFX_X11_VIOLET macro 2709
GFX_X11_WHEAT macro 2710
GFX_X11_WHITE macro 2710
GFX_X11_WHITE_SMOKE macro 2710
GFX_X11_YELLOW macro 2710
GFX_X11_YELLOW_GREEN macro 2710
GFX_XCHAR macro 2952
GLCD Controller Peripheral Library 4135
GLCD Driver Configuration Functions 1676
GLCD_IRQ_TRIGGER_CONTROL enumeration 4229
GLCD_LAYER_COLOR_MODE enumeration 4229
GLCD_LAYER_CONTROL enumeration 1691
GLCD_LAYER_DEST_BLEND_FUNC enumeration 4230
GLCD_LAYER_ID enumeration 4231

GLCD_LAYER_SRC_BLEND_FUNC enumeration 4231
GLCD_MODULE_ID enumeration 4232
GLCD_RGB_MODE enumeration 4232
GLCD_SIGNAL_POLARITY enumeration 4233
Global Control and Status 5724
GOL Object Management 2864
GOL Object Messaging 2882
GOL Object Rendering 2875
GOL Object States 2862
GOL Object Style Scheme 2883
GOL Objects 2720
GOL_PANEL_PARAM structure 2909
GOLD macro 2711
Graphic Display Driver Layer Changes 2555
Graphic Object Layer (GOL) Changes 2555
Graphic Primitive Layer Changes 2555
Graphics Driver Library 1673
Graphics Libraries Help 2553
Graphics Library Help
    Configuring the Library 2577
    Data Types and Constants (GOL Layer) 2885
    Data Types and Constants (Primitive Layer) 2642
    Files 2967
    How the Library Works 2567
    Library Interface (System and Client Interfaces) 2921
    Library Overview 2563
    Using the Library 2562
Graphics Library Porting Guide 2553
Graphics Object Layer 2720
Graphics Object Layer API 2720
Graphics Objects 2563
Graphics Primitive Layer 2590
Graphics Primitive Layer API 2590
GRAPHICS_LIBRARY_VERSION macro 2952
GRAY000 macro 2711
GRAY001 macro 2711
GRAY002 macro 2711
GRAY003 macro 2711
GRAY004 macro 2712
GRAY005 macro 2712
GRAY006 macro 2712
GRAY007 macro 2712
GRAY008 macro 2712
GRAY009 macro 2713
GRAY010 macro 2713
GRAY011 macro 2713
GRAY012 macro 2713
GRAY013 macro 2713
GRAY014 macro 2714
GRAY015 macro 2714
GRAY032 macro 2714
GRAY096 macro 2714
GRAY128 macro 2714
GRAY160 macro 2715
GRAY192 macro 2715
GRAY204 macro 2715
GRAY224 macro 2715
GRAY229 macro 2715
GRAY242 macro 2716
GREEN macro 2716
Group Box Object 2759

H

H macro 931
Handling Device Configuration Bits 1176
Handling Endpoint 0 (EP0) Packets 6710
Handling Errors 4259, 4624
Handling Events 3524
Hardware Abstraction Model 3237, 3304, 3435, 3482, 3515, 3614, 3653, 3685, 3700, 3737, 3979, 4136, 4238, 4342, 4367, 4416, 4480, 4508, 4617, 4711, 4760, 4848, 4895, 4911, 4971, 5012, 5106, 5243, 5281, 5565
    ADC Peripheral Library 3237
    ADCHS Peripheral Library 3304
    ADCP Peripheral Library 3435
    BMX Peripheral Library 3482
    CAN Peripheral Library 3515
    Comparator Peripheral Library 3614
    CTMU Peripheral Library 3653
    Deadman Timer Peripheral Library 3685
    DMA Peripheral Library 3737
    Ethernet Peripheral Library 3979
    GLCD Controller Peripheral Library 4136
    I2C Peripheral Library 4238
    Input Capture Peripheral Library 4342
    Interrupt Peripheral Library 4367
    NVM Peripheral Library 4416
    Oscillator Peripheral Library 4508
    Output Compare Peripheral Library 4480
    PMP Peripheral Library 4617
    Ports Peripheral Library 4760
    Power Peripheral Library 4848
    Prefetch Cache Peripheral Library 3700, 4711
    Reset Peripheral Library 4895
    RTCC Peripheral Library 4911
    SPI Peripheral Library 5012
    SQI Peripheral Library 5106
    System Bus Peripheral Library 4971
    Timer Peripheral Library 5243
    USART Peripheral Library 5281
    USB Peripheral Library 5370
    Watchdog Timer Peripheral Library 5565
Hardware Abstraction Models 5370
Hardware Cursor Control and Management 4139
hci.h 1104
HCI_ACCEPT_CONNECTION_REQUEST macro 584
HCI_ACCEPT_SYNCH_CONNECTION_REQUEST macro 585
HCI_ACL_DATA_HEADER_LEN macro 585
HCI_ALLOCATE_BUFFERS function 16
HCI_ALLOCATE_BUFFERS_FUNCTION macro 35
HCI_ALLOCATE_BUFFERS_RAM_SIZE_VAR macro 36
HCI_ALLOCATE_BUFFERS_VARS macro 36
hci_allocate_conn_state function 551
HCI_AUTHENTICATION_REQUESTED macro 585
HCI_BB_PACKET_TYPE_DH1 macro 585
HCI_BB_PACKET_TYPE_DH3 macro 585
HCI_BB_PACKET_TYPE_DH5 macro 586

HCI_BB_PACKET_TYPE_DM1 macro 586
HCI_BB_PACKET_TYPE_DM3 macro 586
HCI_BB_PACKET_TYPE_DM5 macro 586
HCI_BB_PACKET_TYPE_NO_2_DH1 macro 586
HCI_BB_PACKET_TYPE_NO_2_DH3 macro 587
HCI_BB_PACKET_TYPE_NO_2_DH5 macro 587
HCI_BB_PACKET_TYPE_NO_3_DH1 macro 587
HCI_BB_PACKET_TYPE_NO_3_DH3 macro 587
HCI_BB_PACKET_TYPE_NO_3_DH5 macro 587
HCI_BDADDR_LEN macro 210
HCI_C2H_BROADCAST_NOT_PARCKED_SLAVE macro 588
HCI_C2H_BROADCAST_P2P macro 588
HCI_C2H_BROADCAST_PARCKED_SLAVE macro 588
HCI_C2H_BROADCAST_RESERVED macro 588
HCI_CHANGE_CONNECTION_LINK_KEY macro 588
HCI_CHANGE_CONNECTION_PACKET_TYPE macro 589
hci_check_aux_info function 552
hci_cmd_buffer.h 1118
HCI_CMD_HEADER_LEN macro 589
hci_cmd_queue.h 1118
HCI_CMD_STATUS_BEING_SENT macro 589
HCI_CMD_STATUS_PENDING macro 589
HCI_CMD_STATUS_WAITING_RESPONSE macro 589
hci_config.h 1118
HCI_CONFIG_BECOME_MASTER macro 590
HCI_CONFIG_ENABLE_AUTHENTICATION macro 590
HCI_CONFIG_ENABLE_ENCRYPTION macro 590
hci_config_event_handlers.h 1119
HCI_CONN_ROLE_MASTER macro 590
HCI_CONN_ROLE_SLAVE macro 590
hci_conn_state.h 1120
HCI_CONN_STATE_AUTHENTICATING macro 591
HCI_CONN_STATE_CLOSED macro 591
HCI_CONN_STATE_OPEN macro 591
HCI_CONN_TYPE_ACL macro 591
HCI_CONN_TYPE_ESCO macro 591
HCI_CONN_TYPE_SCO macro 592
HCI_CONNECTABLE macro 592
HCI_CONTROLLER macro 592
hci_cq_find_by_bdaddr_and_opcode function 552
hci_cq_find_by_hconn function 552
hci_cq_find_by_hconn_and_opcode function 552
hci_cq_find_by_opcode function 552
HCI_CREATE_CONNECTION macro 592
HCI_CREATE_CONNECTION_CANCEL macro 592
HCI_CREATE_NEW_UNIT_KEY macro 593
HCI_CTRL_LISTENER_ACL_DATA macro 593
HCI_CTRL_LISTENER_EVENT macro 593
HCI_CTRL_LISTENER_SCO_DATA macro 593
hci_ctrl_state.h 1120
HCI_CTRL_STATE_CLOSED macro 593
HCI_CTRL_STATE_CONNECTABLE macro 594
HCI_CTRL_STATE_DISCOVERABLE macro 594
HCI_CTRL_STATE_INIT macro 594
HCI_CTRL_STATE_READY macro 594
HCI_CTRL_STATE_SLEEP macro 594
HCI_CTRL_STATE_WAKING_UP macro 595
hci_data_buffer.h 1121
HCI_DATA_BUFFER_STATE_FREE macro 595
HCI_DATA_BUFFER_STATE_USED macro 595
hci_data_queue.h 1122
HCI_DATA_STATUS_BEING_SENT macro 595
HCI_DATA_STATUS_PENDING macro 595
HCI_DECLARE_LE_CONN_STATES macro 36
HCI_DECLARE_LE_CTRL_STATE macro 37
HCI_DEFAULT_ACL_CONFIG macro 596
HCI_DELETE_STORED_LINK_KEY macro 596
HCI_DISCONNECT macro 596
HCI_DISCOVERABLE macro 596
HCI_DISCOVERABLE_MODE_GENERAL macro 692
HCI_DISCOVERABLE_MODE_LIMITED macro 693
hci_eir.h 1122
HCI_EIR_FEC_NOT_REQUIRED macro 596
HCI_EIR_FEC_REQUIRED macro 597
HCI_EIR_TYPE_3D_Information_Data macro 693
HCI_EIR_TYPE_ADVERTISING_INTERVAL macro 693
HCI_EIR_TYPE_APPEARANCE macro 693
HCI_EIR_TYPE_DEVICE_ID macro 597
HCI_EIR_TYPE_FLAGS macro 597
HCI_EIR_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS macro 693
HCI_EIR_TYPE_LE_ROLE macro 694
HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_CONFIRMATION_VALUE
macro 694
HCI_EIR_TYPE_LE_SECURE_CONNECTIONS_RANDOM_VALUE
macro 694
HCI_EIR_TYPE_LOCAL_NAME_COMPLETE macro 597
HCI_EIR_TYPE_LOCAL_NAME_SHORTENED macro 597
HCI_EIR_TYPE_MANUFACTURER_SPECIFIC macro 598
HCI_EIR_TYPE_OOB_COD macro 598
HCI_EIR_TYPE_OOB_HASH macro 598
HCI_EIR_TYPE_OOB_RANDOMIZER macro 598
HCI_EIR_TYPE_PUBLIC_TARGET_ADDRESS macro 694
HCI_EIR_TYPE_RANDOM_TARGET_ADDRESS macro 694
HCI_EIR_TYPE_SERVICE_DATA macro 695
HCI_EIR_TYPE_SERVICE_DATA_UUID128 macro 695
HCI_EIR_TYPE_SERVICE_DATA_UUID16 macro 695
HCI_EIR_TYPE_SERVICE_DATA_UUID32 macro 695
HCI_EIR_TYPE_SIMPLE_PAIRING_HASH_C_256 macro 695
HCI_EIR_TYPE_SIMPLE_PAIRING_RANDOMIZER_R_256 macro 696
HCI_EIR_TYPE_SLAVE_CONN_INTERVAL_RANGE macro 696
HCI_EIR_TYPE_SM_OOB_FLAGS macro 696
HCI_EIR_TYPE_SM_TK_VALUE macro 696
HCI_EIR_TYPE_SOLICITATION_UUID128_LIST macro 696
HCI_EIR_TYPE_SOLICITATION_UUID16_LIST macro 697
HCI_EIR_TYPE_SOLICITATION_UUID32_LIST macro 697
HCI_EIR_TYPE_TX_POWER_LEVEL macro 598
HCI_EIR_TYPE_UUID128_LIST_COMPLETE macro 599
HCI_EIR_TYPE_UUID128_LIST_MORE_AVAILABLE macro 599
HCI_EIR_TYPE_UUID16_LIST_COMPLETE macro 599
HCI_EIR_TYPE_UUID16_LIST_MORE_AVAILABLE macro 599
HCI_EIR_TYPE_UUID32_LIST_COMPLETE macro 599
HCI_EIR_TYPE_UUID32_LIST_MORE_AVAILABLE macro 600
HCI_ENABLE_CTRL_TO_HOST_FLOW_CONTROL macro 37
HCI_ENABLE_DEVICE_UNDER_TEST_MODE macro 600
HCI_ENCRYPTION_OFF macro 600
HCI_ENCRYPTION_ON macro 600

HCI_ENHANCED_FLUSH macro 600
HCI_ERR_ACL_CONN_ALREADY_EXISTS macro 601
HCI_ERR_AUTHENTICATION_FAILURE macro 601
HCI_ERR_CONN_REJECT_LIMITED_RESOURCES macro 601
HCI_ERR_CONNECTION_TIMEOUT macro 697
HCI_ERR_INVALID_PARAMETERS macro 601
HCI_ERR_MEMORY_CAPACITY_EXCEEDED macro 601
HCI_ERR_PAIRING_NOT_ALLOWED macro 701
HCI_ERR_PIN_OR_KEY_MISSING macro 701
HCI_ERR_SCO_CONN_LIMIT_EXCEEDED macro 602
HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED macro 602
HCI_ERR_SUCCESS macro 602
HCI_ERR_UNSPECIFIED macro 602
hci_errors.h 1123
HCI_EVT_ALL_HCI_EVENTS macro 602
HCI_EVT_AUTHENTICATION_COMPLETE macro 603
HCI_EVT_CHANGE_CONN_LINK_COMPLETE macro 603
HCI_EVT_CMD_SEND_FINISHED macro 603
HCI_EVT_CMD_SEND_STARTED macro 603
HCI_EVT_COMMAND_COMPLETE macro 603
HCI_EVT_COMMAND_COMPLETE_PARAM_LEN macro 604
HCI_EVT_COMMAND_STATUS macro 604
HCI_EVT_CONN_PACKET_TYPE_CHANGED macro 604
HCI_EVT_CONNECTION_COMPLETE macro 604
HCI_EVT_CONNECTION_REQUEST macro 604
HCI_EVT_DATA_BUFFER_OVERFLOW macro 605
HCI_EVT_DISCONNECTION_COMPLETE macro 605
HCI_EVT_ENCRYPTION_CHANGE macro 605
HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE macro 605
HCI_EVT_ENHANCED_FLUSH_COMPLETE macro 605
HCI_EVT_EXTENDED_INQUIRY_RESULT macro 606
HCI_EVT_FIRST macro 606
HCI_EVT_FLOW_SPECIFICATION_COMPLETE macro 606
HCI_EVT_FLUSH_OCCURED macro 606
hci_evt_handlers.h 1124
HCI_EVT_HARDWARE_ERROR macro 606
HCI_EVT_INQUIRY_COMPLETE macro 607
HCI_EVT_INQUIRY_RESULT macro 607
HCI_EVT_INQUIRY_RESULT_WITH_RSSI macro 607
HCI_EVT_IO_CAPABILITY_REQUEST macro 607
HCI_EVT_IO_CAPABILITY_RESPONSE macro 607
HCI_EVT_KEYPRESS_NOTIFICATION macro 608
HCI_EVT_LAST macro 608
HCI_EVT_LE_META_EVENT macro 608
HCI_EVT_LINK_IS_BUSY macro 608
HCI_EVT_LINK_IS_IDLE macro 608
HCI_EVT_LINK_KEY_NOTIFICATION macro 609
HCI_EVT_LINK_KEY_REQUEST macro 609
HCI_EVT_LINK_SUPERVISION_TO_CHANGED macro 609
HCI_EVT_LOOPBACK_COMMAND macro 609
HCI_EVT_MASTER_LINK_KEY_COMPLETE macro 609
HCI_EVT_MAX_SLOTS_CHANGE macro 610
HCI_EVT_MODE_CHANGE macro 610
HCI_EVT_NUM_OF_COMPLETED_PACKETS macro 610
HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE macro 610
HCI_EVT_PIN_CODE_REQUEST macro 610
HCI_EVT_QOS_SETUP_COMPLETE macro 611
HCI_EVT_QOS_VIOLATION macro 611
HCI_EVT_READ_CLOCK_OFFSET_COMPLETE macro 611
HCI_EVT_READ_RMT_EXT_FEATURES_COMP macro 611
HCI_EVT_READ_RMT_SUP_FEATURES_COMP macro 611
HCI_EVT_READ_RMT_VERSION_INFO_COMP macro 612
HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE macro 612
HCI_EVT_REMOTE_OOB_DATA_REQUEST macro 612
HCI_EVT_RETURN_LINK_KEYS macro 612
HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF macro 612
HCI_EVT_ROLE_CHANGE macro 613
HCI_EVT_SIMPLE_PAIRING_COMPLETE macro 613
HCI_EVT_SNIFF_SUBRATING macro 613
HCI_EVT_SYNCH_CONNECTION_CHANGED macro 613
HCI_EVT_SYNCH_CONNECTION_COMPLETE macro 613
HCI_EVT_USER_CONFIRMATION_REQUEST macro 614
HCI_EVT_USER_PASSKEY_NOTIFICATION macro 614
HCI_EVT_USER_PASSKEY_REQUEST macro 614
HCI_EXIT_PARK_STATE macro 614
HCI_EXIT_PERIODIC_INQUIRY_MODE macro 614
HCI_EXIT_SNIFF_MODE macro 615
HCI_FLOW_SPECIFICATION macro 615
HCI_FLUSH macro 615
hci_get_conn_state function 553
hci_get_conn_state_by_bdaddr function 559
HCI_H2C_BROADCAST_ACTIVE_SLAVE macro 615
HCI_H2C_BROADCAST_NO_BROADCATS macro 615
HCI_H2C_BROADCAST_PARCKED_SLAVE macro 616
HCI_H2C_BROADCAST_RESERVED macro 616
HCI_HOLD_MODE macro 616
HCI_HOST_BUFFER_SIZE macro 616
HCI_HOST_NUM_OF_COMPLETED_PACKETS macro 616
HCI_INIT_FLAG_IGNORE_TOTAL_NUM_ACL_DATA_PACKETS macro 697
HCI_INIT_FLAG_SEND_HCI_RESET macro 697
HCI_INIT_LE_CONN_STATES macro 37
HCI_INIT_LE_CTRL_STATE macro 37
HCI_INIT_SCO_HANDLERS macro 37
HCI_INIT_SSP_HANDLERS macro 38
HCI_INQUIRY macro 617
HCI_INQUIRY_CANCEL macro 617
HCI_INQUIRY_MODE_EXTENDED macro 617
HCI_INQUIRY_MODE_STANDARD macro 617
HCI_INQUIRY_MODE_WITH_RSSI macro 617
HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY macro 618
HCI_IO_CAPABILITY_REQUEST_REPLY macro 618
HCI_L2CAP_BUFFER_LEN macro 38
hci_le.h 1125
HCI_LE_ADD_DEVICE_TO_WHITE_LIST macro 618
HCI_LE_ADDRESS_TYPE_PUBLIC macro 618
HCI_LE_ADDRESS_TYPE_RANDOM macro 618
HCI_LE_ADV_CHANNEL_MAP_ENABLE_37 macro 698
HCI_LE_ADV_CHANNEL_MAP_ENABLE_38 macro 698
HCI_LE_ADV_CHANNEL_MAP_ENABLE_39 macro 698
HCI_LE_ADV_CHANNEL_MAP_ENABLE_ALL macro 698
HCI_LE_ADV_CHANNEL_MAP_RESERVED macro 698
HCI_LE_ADV_TYPE_CONN_UNDIRECT macro 699
HCI_LE_ADV_TYPE_DIRECT_HIGH macro 699
HCI_LE_ADV_TYPE_DIRECT_LOW macro 699
HCI_LE_ADV_TYPE_NONCONN macro 699

HCI_LE_ADV_TYPE_SCAN macro 699
HCI_LE_ADVERTISING_FLAG_BREDR_NOT_SUPPORTED macro 619
HCI_LE_ADVERTISING_FLAG_GENERAL_DISCOVERABLE_MODE macro 619
HCI_LE_ADVERTISING_FLAG_LIMITED_DISCOVERABLE_MODE macro 619
HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_CONTROLLER macro 619
HCI_LE_ADVERTISING_FLAG_SIMULTANEOUS_LE_BREDR_HOST macro 619
HCI_LE_AVERTISING_DISABLED macro 620
HCI_LE_AVERTISING_ENABLED macro 620
HCI_LE_CLEAR_WHITE_LIST macro 620
HCI_LE_CONNECTION_UPDATE macro 620
HCI_LE_CREATE_CONNECTION macro 620
HCI_LE_CREATE_CONNECTION_CANCEL macro 621
HCI_LE_DISABLED macro 621
HCI_LE_DUPLICATE_FILTERING_DISABLED macro 621
HCI_LE_DUPLICATE_FILTERING_ENABLED macro 621
HCI_LE_ENABLED macro 621
HCI_LE_ENCRYPT macro 622
HCI_LE_EVT_ADVERTISING_REPORT macro 622
HCI_LE_EVT_CONNECTION_COMPLETE macro 622
HCI_LE_EVT_CONNECTION_UPDATE_COMPLETE macro 622
HCI_LE_EVT_LONG_TERM_KEY_REQUEST macro 622
HCI_LE_EVT_READ_REMOTE_USED_FEATURES_COMPLETE macro 623
HCI_LE_FILTER_POLICY_NOT_USED macro 623
HCI_LE_FILTER_POLICY_WHITE_LIST macro 623
HCI_LE_FLAG_SHARED_ACL_BUFFERS macro 623
HCI_LE_FLAG_SIMULTANEOUS_LE_BREDR macro 623
HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY macro 624
HCI_LE_LONG_TERM_KEY_REQUEST_REPLY macro 624
HCI_LE_MAX_AD_LEN macro 624
HCI_LE_RAND macro 624
HCI_LE_RANDOM_ADDRESS_TYPE_NON_RESOLVABLE macro 700
HCI_LE_RANDOM_ADDRESS_TYPE_RESOLVABLE macro 700
HCI_LE_RANDOM_ADDRESS_TYPE_STATIC macro 700
HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER macro 624
HCI_LE_READ_BUFFER_SIZE macro 625
HCI_LE_READ_CHANNEL_MAP macro 625
HCI_LE_READ_LOCAL_SUPPORTED_FEATURES macro 625
HCI_LE_READ_REMOTE_USED_FEATURES macro 625
HCI_LE_READ_SUPPORTED_STATES macro 625
HCI_LE_READ_WHITE_LIST_SIZE macro 626
HCI_LE_RECEIVE_TEST macro 626
HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST macro 626
HCI_LE_SCAN_DISABLED macro 626
HCI_LE_SCAN_ENABLED macro 626
HCI_LE_SCAN_EVT_CANCEL_FAILED macro 627
HCI_LE_SCAN_EVT_CANCELLED macro 627
HCI_LE_SCAN_EVT_DEVICE_FOUND macro 627
HCI_LE_SCAN_EVT_START_FAILED macro 627
HCI_LE_SCAN_EVT_STARTED macro 627
HCI_LE_SCAN_FILTER_POLICY_ALL macro 628
HCI_LE_SCAN_FILTER_POLICY_WHITE_LIST macro 628
HCI_LE_SCAN_TYPE_ACTIVE macro 628
HCI_LE_SCAN_TYPE_PASSIVE macro 628
HCI_LE_SET_ADVERTISE_ENABLE macro 628
HCI_LE_SET_ADVERTISING_DATA macro 629
HCI_LE_SET_ADVERTISING_PARAMETERS macro 629
HCI_LE_SET_EVENT_MASK macro 629
HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION macro 629
HCI_LE_SET_RANDOM_ADDRESS macro 629
HCI_LE_SET_SCAN_ENABLE macro 630
HCI_LE_SET_SCAN_PARAMETERS macro 630
HCI_LE_SET_SCAN_RESPONSE_DATA macro 630
HCI_LE_SIMULTANEOUS_DISABLED macro 630
HCI_LE_SIMULTANEOUS_ENABLED macro 630
HCI_LE_START_ENCRYPTION macro 631
HCI_LE_TEST_END macro 631
HCI_LE_TRANSMITTER_TEST macro 631
HCI_LINK_KEY_LEN macro 631
HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY macro 631
HCI_LINK_KEY_REQUEST_REPLY macro 632
HCI_LINK_KEY_TYPE_COMBINATION macro 632
HCI_LINK_KEY_TYPE_LOCAL_UNIT macro 632
HCI_LINK_KEY_TYPE_REMOTE_UNIT macro 632
HCI_LINK_POLICY_ENABLE_ALL macro 700
HCI_LINK_POLICY_ENABLE_HOLD_MODE macro 632
HCI_LINK_POLICY_ENABLE_PARK_STATE macro 633
HCI_LINK_POLICY_ENABLE_ROLE_SWITCH macro 633
HCI_LINK_POLICY_ENABLE_SNIFF_MODE macro 633
HCI_LINK_TYPE_BD_EDR macro 633
HCI_LINK_TYPE_LE macro 633
hci_linkkey_buffer.h 1128
HCI_MASTER_LINK_KEY macro 634
HCI_MAX_CONNECT_ATTEMPTS macro 39
HCI_MAX_DATA_BUFFERS macro 634
HCI_MAX_EVENT_PARAM_LEN macro 634
HCI_MAX_PARAM_LEN macro 634
HCI_MAX_PIN_LENGTH macro 634
HCI_OPCODE macro 635
HCI_PACKET_BOUNDARY_COMPLETE macro 701
HCI_PACKET_BOUNDARY_CONTINUE macro 635
HCI_PACKET_BOUNDARY_FIRST macro 635
HCI_PACKET_BOUNDARY_FIRST_AUTO_FLUSH macro 635
HCI_PACKET_BOUNDARY_FIRST_NO_AUTO_FLUSH macro 635
HCI_PACKET_TYPE_ACL_DATA macro 636
HCI_PACKET_TYPE_COMMAND macro 636
HCI_PACKET_TYPE_EVENT macro 636
HCI_PACKET_TYPE_NONE macro 636
HCI_PACKET_TYPE_SCO_DATA macro 636
HCI_PAGE_SCAN_REPETITION_MODE_R0 macro 637
HCI_PAGE_SCAN_REPETITION_MODE_R1 macro 637
HCI_PAGE_SCAN_REPETITION_MODE_R2 macro 637
HCI_PARAM_LEN_BD_ADDR macro 637
HCI_PARAM_LEN_BYTE macro 637
HCI_PARAM_LEN_DEV_CLASS macro 638
HCI_PARAM_LEN_HANDLE macro 638
HCI_PARAM_LEN_INT macro 638
HCI_PARAM_LEN_LONG macro 638
HCI_PARAM_TYPE_BD_ADDR macro 638
HCI_PARAM_TYPE_BYTE macro 639
HCI_PARAM_TYPE_DEV_CLASS macro 639

HCI_PARAM_TYPE_HANDLE macro 639
HCI_PARAM_TYPE_INT macro 639
HCI_PARAM_TYPE_LONG macro 639
HCI_PARAM_TYPE_STRING macro 640
HCI_PARK_STATE macro 640
HCI_PERIODIC_INQUIRY_MODE macro 640
HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY macro 640
HCI_PIN_CODE_REQUEST_REPLY macro 640
HCI_POWER_MODE_ACTIVE macro 641
HCI_POWER_MODE_HOLD macro 641
HCI_POWER_MODE_PARK macro 641
HCI_POWER_MODE_SNIFF macro 641
hci_private.h 1129
HCI_QOS_SETUP macro 641
HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE macro 642
HCI_READ_AFH_CHANNEL_MAP macro 642
HCI_READ_AUTHENTICATION_ENABLE macro 642
HCI_READ_AUTOMATIC_FLASH_TIMEOUT macro 642
HCI_READ_BD_ADDR macro 642
HCI_READ_BUFFER_SIZE macro 643
HCI_READ_CLASS_OF_DEVICE macro 643
HCI_READ_CLOCK_COMMAND macro 643
HCI_READ_CLOCK_OFFSET macro 643
HCI_READ_CONNECTION_ACCEPT_TIMEOUT macro 643
HCI_READ_CURRENT_IAC_LAP macro 644
HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING macro 644
HCI_READ_DEFAULT_POLICY_SETTINGS macro 644
HCI_READ_ENCRYPTION_MODE macro 644
HCI_READ_EXTENDED_INQUIRY_RESPONSE macro 644
HCI_READ_FAILED_CONTACT_COUNTER macro 645
HCI_READ_HOLD_MODE_ACTIVITY macro 645
HCI_READ_INQUIRY_MODE macro 645
HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL macro 645
HCI_READ_INQUIRY_SCAN_ACTIVITY macro 645
HCI_READ_INQUIRY_SCAN_TYPE macro 646
HCI_READ_LE_HOST_SUPPORT macro 646
HCI_READ_LINK_POLICY_SETTINGS macro 646
HCI_READ_LINK_QUALITY macro 646
HCI_READ_LINK_SUPERVISION_TIMEOUT macro 646
HCI_READ_LMP_HANDLE macro 647
HCI_READ_LOCAL_EXTENDED_FEATURES macro 647
HCI_READ_LOCAL_NAME macro 647
HCI_READ_LOCAL_OOB_DATA macro 647
HCI_READ_LOCAL_SUPPORTED_COMMANDS macro 647
HCI_READ_LOCAL_SUPPORTED_FEATURES macro 648
HCI_READ_LOCAL_VERSION_INFORMATION macro 648
HCI_READ_LOOPBACK_MODE macro 648
HCI_READ_NUM_BROADCST_RETR macro 648
HCI_READ_NUM_OF_SUPPORTED_IAC macro 648
HCI_READ_PAGE_SCAN_ACTIVITY macro 649
HCI_READ_PAGE_SCAN_PERIOD_MODE macro 649
HCI_READ_PAGE_SCAN_TYPE macro 649
HCI_READ_PAGE_TIMEOUT macro 649
HCI_READ_PIN_TYPE macro 649
HCI_READ_REFRESH_ENCRYPTION_KEY macro 650
HCI_READ_REMOTE_EXTENDED_FEATURES macro 650
HCI_READ_REMOTE_SUPPORTED_FEATURES macro 650
HCI_READ_REMOTE_VERSION_INFORMATION macro 650
HCI_READ_RSSI macro 650
HCI_READ_SCAN_ENABLE macro 651
HCI_READ_SIMPLE_PAIRING_MODE macro 651
HCI_READ_STORED_LINK_KEY macro 651
HCI_READ_SYNC_FLOW_CONTROL_ENABLE macro 651
HCI_READ_TRANSMIT_POWER_LEVEL macro 651
HCI_READ_VOICE_SETTING macro 652
HCI_REJECT_CONNECTION_REQUEST macro 652
HCI_REJECT_SYNCH_CONNECTION_REQUEST macro 652
HCI_REMOTE_NAME_REQUEST macro 652
HCI_REMOTE_NAME_REQUEST_CANCEL macro 652
HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY macro 653
HCI_REMOTE_OOB_DATA_REQUEST_REPLY macro 653
HCI_RESET macro 653
HCI_RESET_FAILED_CONTACT_COUNTER macro 653
HCI_ROLE_DISCOVERY macro 653
HCI_ROLE_SWITCH_ALLOW macro 654
HCI_ROLE_SWITCH_DISALLOW macro 654
HCI_SCAN_INQUIRY macro 654
HCI_SCAN_PAGE macro 654
HCI_SCO_CONTENT_FORMAT_AIR_CODING_A_LAW macro 654
HCI_SCO_CONTENT_FORMAT_AIR_CODING_CSVD macro 655
HCI_SCO_CONTENT_FORMAT_AIR_CODING_N_LAW macro 655
HCI_SCO_CONTENT_FORMAT_AIR_CODING_TRANSPARENT macro 655
HCI_SCO_CONTENT_FORMAT_INPUT_CODING_A_LAW macro 655
HCI_SCO_CONTENT_FORMAT_INPUT_CODING_LINEAR macro 655
HCI_SCO_CONTENT_FORMAT_INPUT_CODING_N_LAW macro 656
HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_1_COMPLEMENT macro 656
HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_2_COMPLEMENT macro 656
HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_SIGN_MAGNITUDE macro 656
HCI_SCO_CONTENT_FORMAT_INPUT_FORMAT_UNSIGNED macro 656
HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_16 macro 657
HCI_SCO_CONTENT_FORMAT_INPUT_SAMPLE_SIZE_8 macro 657
HCI_SCO_DATA_HEADER_LEN macro 657
HCI_SCO_MAX_DATA_LEN macro 657
HCI_SCO_MAX_LATENCY_DONTCARE macro 657
HCI_SCO_PACKET_TYPE_ALL macro 658
HCI_SCO_PACKET_TYPE_EV3 macro 658
HCI_SCO_PACKET_TYPE_EV4 macro 658
HCI_SCO_PACKET_TYPE_EV5 macro 658
HCI_SCO_PACKET_TYPE_HV1 macro 658
HCI_SCO_PACKET_TYPE_HV2 macro 659
HCI_SCO_PACKET_TYPE_HV3 macro 659
HCI_SCO_PACKET_TYPE_NO_2_EV3 macro 659
HCI_SCO_PACKET_TYPE_NO_2_EV5 macro 659
HCI_SCO_PACKET_TYPE_NO_3_EV3 macro 659
HCI_SCO_PACKET_TYPE_NO_3_EV5 macro 660
HCI_SCO_RTX_EFFORT_DONTCARE macro 660
HCI_SCO_RTX_EFFORT_NO_RETRANSMISSION macro 660
HCI_SCO_RTX_EFFORT_OPTIMIZE_LINK_QUALITY macro 660
HCI_SCO_RTX_EFFORT_OPTIMIZE_POWER_CONSUMPTION macro 660
HCI_SCO_RX_BANDWIDTH_DONTCARE macro 661

HCI_SCO_TX_BANDWIDTH_DONTCARE macro 661
HCI_SEND_DATA_STATUS_INTERRUPTED macro 661
HCI_SEND_DATA_STATUS_SUCCESS macro 661
HCI_SEND_KEY_PRESS_NOTIFICATION macro 661
HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION macro 662
HCI_SET_CONNECTION_ENCRYPTION macro 662
HCI_SET_CTRL_TO_HOST_FLOW_CONTROL macro 662
HCI_SET_EVENT_FILTER macro 662
HCI_SET_EVENT_MASK macro 662
HCI_SETUP_SYNCHRONOUS_CONNECTION macro 663
hci_signal.h 1129
HCI_SIZEOF_LE_CONN_STATES macro 38
HCI_SIZEOF_LE_CTRL_STATE macro 38
hci_sleep function 553
HCI_SNIFF_MODE macro 663
HCI_SNIFF_SUBRATING macro 663
HCI_SUCCESS macro 701
HCI_SWITCH_ROLE macro 663
hci_transport.h 1130
HCI_TRANSPORT_HEADER_LEN macro 663
hci_transport_t structure 690
HCI_TX_BUFFER_LEN macro 38
HCI_UART_PACKET_TYPE_ACL_DATA macro 664
HCI_UART_PACKET_TYPE_COMMAND macro 664
HCI_UART_PACKET_TYPE_EVENT macro 664
HCI_UART_PACKET_TYPE_SCO_DATA macro 664
HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY macro 664
HCI_USER_CONFIRMATION_REQUEST_REPLY macro 665
HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY macro 665
HCI_USER_PASSKEY_REQUEST_REPLY macro 665
hci_utils.h 1130
hci_wakeup function 553
HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE macro 665
HCI_WRITE_AUTHENTICATION_ENABLE macro 665
HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT macro 666
HCI_WRITE_CLASS_OF_DEVICE macro 666
HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT macro 666
HCI_WRITE_CURRENT_IAC_LAP macro 666
HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING macro 666
HCI_WRITE_DEFAULT_POLICY_SETTINGS macro 667
HCI_WRITE_ENCRYPTION_MODE macro 667
HCI_WRITE_EXTENDED_INQUIRY_RESPONSE macro 667
HCI_WRITE_EXTENDED_INQUIRY_RESPONSE_PARAM_LEN macro 667
HCI_WRITE_HOLD_MODE_ACTIVITY macro 667
HCI_WRITE_INQUIRY_MODE macro 668
HCI_WRITE_INQUIRY_SCAN_ACTIVITY macro 668
HCI_WRITE_INQUIRY_SCAN_TYPE macro 668
HCI_WRITE_INQUIRY_TX_POWER_LEVEL macro 668
HCI_WRITE_LE_HOST_SUPPORT macro 668
HCI_WRITE_LINK_POLICY_SETTINGS macro 669
HCI_WRITE_LINK_SUPERVISION_TIMEOUT macro 669
HCI_WRITE_LOCAL_NAME macro 669
HCI_WRITE_LOCAL_NAME_PARAM_LEN macro 669
HCI_WRITE_LOOPBACK_MODE macro 669
HCI_WRITE_NUM_BROADCST_RETR macro 670
HCI_WRITE_PAGE_SCAN_ACTIVITY macro 670
HCI_WRITE_PAGE_SCAN_PERIOD_MODE macro 670
HCI_WRITE_PAGE_SCAN_TYPE macro 670
HCI_WRITE_PAGE_TIMEOUT macro 670
HCI_WRITE_PIN_TYPE macro 671
HCI_WRITE_SCAN_ENABLE macro 671
HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE macro 671
HCI_WRITE_SIMPLE_PAIRING_MODE macro 671
HCI_WRITE_STORED_LINK_KEY macro 671
HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE macro 672
HCI_WRITE_VOICE_SETTING macro 672
hcitr_3wire.h 1130
HCITR_3WIRE_DEFAULT_ACK_TIMEOUT macro 672
hcitr_bcsp.h 1131
HCITR_BCSP_ALLOCATE_BUFFERS function 202
HCITR_BCSP_DEFAULT_ACK_TIMEOUT macro 672
hcitr_packet.h 1131
hcitr_tih4.h 1131
HCITR_TIH4_POWER_EVENT_AWAKE enumeration member 690
HCITR_TIH4_POWER_EVENT_PREPARE_TO_SLEEP enumeration member 690
HCITR_TIH4_POWER_EVENT_SLEEP enumeration member 690
HCITR_TIH4_POWER_EVENT_WAKE_UP enumeration member 690
hcitr_uart.h 1132
HCRP_ALLOCATE_BUFFERS function 203
help_plib_adc.h 3301
help_plib_adchs.h 3430
help_plib_adcp.h 3481
help_plib_bmx.h 3514
help_plib_cmp.h 3652
help_plib_devcon.h 3736
help_plib_dma.h 3866
help_plib_pcache.h 4759
help_plib_ports.h 4847
help_plib_power.h 4895
help_plib_reset.h 4910
help_plib_sb.h 5011
help_plib_spi.h 5104
help_plib_sqi.h 5241
help_plib_tmr.h 5280
HFP_ALLOCATE_BUFFERS function 16
HID Device TPL Table Configuration 7138
HID_ALLOCATE_BUFFERS function 17
HIDS_LAST_DEVICE_ADDR macro 210
HIDS_SIGNATURE macro 210
HIDS_SIGNATURE_ADDR macro 211
HLVD_LIMIT enumeration 4892
HLVD_MODE enumeration 4892
Host Layer - Application Interaction 6961
Host Layer Initialization 6957
HOST_NOT_FOUND macro 6058
hostent structure 6058
How the Library Works 1174, 1192, 1293, 1309, 1367, 1393, 1443, 1483, 1522, 1570, 1590, 1674, 1729, 1769, 1831, 1868, 1902, 1931, 1967, 2043, 2112, 2130, 2152, 2211, 2270, 2332, 2403, 2496, 2567, 3202, 3241, 3305, 3436, 3483, 3517, 3615, 3655, 3686, 3701, 3738, 3868, 3936, 3981, 4136, 4239, 4343, 4368, 4417, 4482, 4510, 4618, 4712, 4763, 4850, 4896, 4912, 4972, 5014, 5108, 5245, 5283, 5372, 5517, 5565, 5577, 5610, 5635, 5655, 5674, 5691, 5718, 5771, 5835, 5856, 5860, 5889, 5917, 5927, 5936, 5977, 6010, 6065, 6079, 6094, 6299, 6352, 6402, 6577, 6617, 6635, 6678, 6718, 6748, 6775, 6860, 6895, 6926, 6936, 6957, 7006, 7104, 7138, 7148, 7153

ADC Peripheral Library 3241
ADCHS Peripheral Library 3305
ADCP Peripheral Library 3436
AK4384 Driver Library 1393
AK4642 Driver Library 1443
AK4953 Driver Library 1483
AK7755 Driver Library 1522
AR1021 Touch Driver Library 2112
ARP Module 6010
BMX Peripheral Library 3483
CAN Peripheral Library 3517
Clock System Service Library 5610
Command Processor System Service Library 5635
Comparator Peripheral Library 3615
Console System Service Library 5655
Deadman Timer Peripheral Library 3686
Debug System Service Library 5674
Device Control Peripheral Library 3701
Device Control System Service Library 5691
DHCP Module 6065
DHCP Server Module 6079
DHCPv6 Module 6094
DMA Peripheral Library 3738
DMA System Service Library 5718
Dual Data Rate (DDR) SDRAM Peripheral Library 3868
EBI Peripheral Library 3936
ENC28J60 Driver 1570
ENCx24J600 Driver 1590
Ethernet Peripheral Library 3981
Generic USB Device Library 6936
GLCD Controller Peripheral Library 4136
I2C Peripheral Library 4239
Input Capture Peripheral Library 4343
Interrupt Peripheral Library 4368
Interrupt System Service Library 5835
IPv4 Module 6299
MAC Driver Module 6352
Manager Module 6402
Memory System Service Library 5856
Messaging System Service Library 5860
MRF24W Wi-Fi Driver Library 2403
MRF24WN Wi-Fi Driver Library 2496
MTCH6301 Touch Driver Library 2130
MTCH6303 Touch Driver Library 2152
NVM Driver Library 1831
NVM Peripheral Library 4417
OSAL Library 3202
Oscillator Peripheral Library 4510
Output Compare Peripheral Library 4482
PMP Driver Library 1868
PMP Peripheral Library 4618
Ports Peripheral Library 4763
Ports System Service Library 5889
Power Peripheral Library 4850
Prefetch Cache Peripheral Library 4712
Random Number Generator System Service Library 5917
Reset Peripheral Library 4896
RTCC Peripheral Library 4912
SAMPLE Library 5577
SD Card Driver Library 1902
SPI Driver Library 1931
SPI Flash Driver Library 1967
SPI Peripheral Library 5014
SQI Peripheral Library 5108
System Bus Peripheral Library 4972
TCP Module 6577
TEST HARNESS LIB Library 6678
Timer Driver Library 2043
Timer Peripheral Library 5245
Timer System Service Library 5936
UDP Module 6635
USART Driver Library 2332
USART Peripheral Library 5283
USB Device Layer Library 6775
USB MSD Device Library 6926
Watchdog Timer Peripheral Library 5565
Watchdog Timer System Service Library 5977
HSP_AG_ALLOCATE_BUFFERS function 17
HSP_ALLOCATE_BUFFERS function 17
HTTP Changes 5991
HTTP Configuration 6526
HTTP Features 6164
HTTP Module 6163
HTTP Net Features 6203
HTTP Net Module 6202
http.h 6199
http_config.h 6201
HTTP_CONN_HANDLE type 6196
HTTP_FILE_TYPE enumeration 6196
HTTP_IO_RESULT enumeration 6197
HTTP_MODULE_FLAGS enumeration 6197
http_net.h 6262
http_net_config.h 6264
HTTP_READ_STATUS enumeration 6198
HTTP_STATUS enumeration 6198
Hub TPL Table Configuration 7149

I

i16 type 3119
I2C Driver Library Help 1727
I2C Peripheral Library 4237
I2C_DATA_TYPE type 1759
I2C_MODULE_ID enumeration 4337
I2C_SLAVE_ADDRESS_7bit type 1759
I2C_SLAVE_ADDRESS_VALUE type 1760
I2C_STATIC_DRIVER_MODE macro 1740
I2S Driver Library Help 1767
IAP_ALLOCATE_BUFFERS function 17
IAP_BT_ALLOCATE_BUFFERS function 17
IAP_BT_MAX_TRANSPORTS macro 21
IAP_MAX_SESSIONS macro 21
IAP_RX_BUFFER_SIZE macro 21
IAP2_ALLOCATE_BUFFERS function 18
IAP2_MAX_PACKET_SIZE macro 21
IAP2_MAX_SESSIONS macro 21
IAPEA_ALLOCATE_BUFFERS function 202

IC_ALT_TIMERS enumeration 4364
IC_BUFFER_SIZE enumeration 4362
IC_EDGE_TYPES enumeration 4362
IC_EVENTS_PER_INTERRUPT enumeration 4362
IC_INPUT_CAPTURE_MODES enumeration 4363
IC_MODULE_ID enumeration 4363
IC_TIMERS enumeration 4364
ICMP Module 6267
icmp.h 6276
icmp_config.h 6277
ICMP_ECHO_RESULT enumeration 6274
ICMP_HANDLE type 6275
ICMP6_FILTER macro 6051
ICMPv6 Module 6277
icmpv6.h 6288
icmpv6_config.h 6289
ICMPV6_ERR_DU_CODE enumeration 6284
ICMPV6_ERR_PP_CODE enumeration 6285
ICMPV6_ERR_PTB_CODE macro 6283
ICMPV6_ERR_TE_CODE enumeration 6285
ICMPV6_HANDLE type 6286
ICMPV6_HEADER_ECHO structure 6286
ICMPV6_HEADER_ERROR structure 6287
ICMPV6_INFO_EREQ_CODE macro 6284
ICMPV6_INFO_ERPL_CODE macro 6284
ICMPV6_PACKET_TYPES enumeration 6287
id3.h 1290
ID3_EVENT enumeration 1283
ID3_EventHandler function 1276
ID3_H macro 1285
ID3_Initialize function 1275
ID3_Parse function 1276
ID3_Parse_Frame function 1276
ID3_ParseFrameV22 function 1276
ID3_ParseFrameV23 function 1276
ID3_STATE enumeration 1283
ID3_STRING_SIZE macro 1285
ID3V1_EXTENDED_TAG structure 1283
ID3V1_TAG structure 1283
ID3V2_TAG_HEADER structure 1284
ID3V22_ALBUM macro 1285
ID3V22_ARTIST macro 1286
ID3V22_FRAME structure 1284
ID3V22_FRAME_HEADER structure 1284
ID3V22_TITLE macro 1286
ID3V22_ZERO macro 1286
ID3V23_ALBUM macro 1286
ID3V23_ARTIST macro 1286
ID3V23_FRAME structure 1285
ID3V23_FRAME_HEADER structure 1285
ID3V23_TITLE macro 1287
ID3V23_ZERO macro 1287
Image Rendering Functions 2607
ImageAbort macro 2961
ImageDecode function 2932
ImageDecoderInit function 2931
ImageDecoderParceHeader function 2933
ImageDecodeTask function 2931
ImageFullScreenDecode macro 2961
ImageLoopCallbackRegister function 2932
ImagePixelIntoArray function 2933
ImagePixelOutput function 2933
ImageRegisterFrameCompleteCallback function 2934
ImageRegisterImageCompleteCallback function 2934
ImageRegisterImageFailCallback function 2934
IMG_ALIGN_CENTER macro 2962
IMG_BMP enumeration member 2959
IMG_DECODE_ABORTED macro 2962
IMG_DOWN_SCALE macro 2962
IMG_FEOF macro 2962
IMG_FILE macro 2963
IMG_FILE_FORMAT enumeration 2959
IMG_FILE_SYSTEM_API structure 2959
IMG_FREAD macro 2963
IMG_FSEEK macro 2963
IMG_FTELL macro 2963
IMG_GIF enumeration member 2959
IMG_JPEG enumeration member 2959
IMG_LOOP_CALLBACK type 2959
IMG_LOOP_FRAMECOMPLETE type 2966
IMG_LOOP_IMAGECOMPLETE type 2966
IMG_LOOP_IMAGEFAIL type 2966
IMG_NONE enumeration member 2959
IMG_PIXEL_OUTPUT type 2960
IMG_PIXEL_XY_RGB_888 structure 2960
IMG_SCREEN_HEIGHT macro 2963
IMG_SCREEN_WIDTH macro 2964
IMG_vCheckAndAbort macro 2964
IMG_vFrameComplete macro 2966
IMG_vImageComplete macro 2967
IMG_vImageFail macro 2967
IMG_vLoopCallback macro 2964
IMG_vPixelArray macro 2967
IMG_vPutPixel macro 2964
IMG_vSetboundaries function 2933
IMG_vSetColor macro 2967
in_addr structure 6047
in6_addr structure 6059
INADDR_ANY macro 6044
Include Files Processing 6204
INIT_DE_SEQUENCE macro 845
initData enumeration 6990
Initialization 1674, 3242, 3616, 3988, 4619, 5610, 5760
Initialization and Tasks 5718
Initialization Functions 2590
Initialization of CAN 3519
Initialization Overrides 3206
Initializing and Communicating with the Endpoint 6710
Initializing the Device Layer 6778
Initializing the Driver 2112, 2130
Initializing the I2C 4240
Initializing the Library 6719, 6748
Initializing the USART Driver 2332
Initializing the USB Device Stack 6707
Initiator Initialization 4972
INP_BUF_SIZE macro 832

Input Capture Driver Library 1821
Input Capture Module Setup 4343
Input Capture Peripheral Library 4341
INPUT_BUF_SIZE macro 1268
INT_EXTERNAL_EDGE_TRIGGER enumeration 5853
INT_EXTERNAL_SOURCES enumeration 4399
INT_PRIORITY_LEVEL enumeration 4399
INT_SHADOW_REGISTER enumeration 4412
INT_SOURCE enumeration 4400
INT_STATE_GLOBAL type 4412
INT_SUBPRIORITY_LEVEL enumeration 4406
INT_VECTOR enumeration 4407
INT_VECTOR_SPACING enumeration 4413
int16_gfx_image_prog type 2946
int16_prog type 2953
int16_prog_pack type 2954
int16c structure 3084
int32_gfx_image_prog type 2947
int32_prog type 2954
int32_prog_pack type 2954
int32c structure 3085
int8_gfx_image_prog type 2947
int8_prog type 2954
int8_prog_pack type 2954
Internal FRC Oscillator Tuning 4512
Interrupt Control 4139
Interrupt Control and Management 3739
Interrupt Peripheral Library 4366
Interrupt State Machine 4252
Interrupt System Service Library 5834
Interrupt System Setup 5836
Interrupts 5034
INTPCM type 1265
Introduction 3, 4, 1173, 1191, 1211, 1260, 1270, 1291, 1307, 1322, 1336, 1353, 1358, 1365, 1388, 1391, 1442, 1482, 1520, 1558, 1559, 1569, 1589, 1608, 1629, 1666, 1673, 1727, 1767, 1821, 1830, 1864, 1867, 1896, 1900, 1929, 1966, 2042, 2093, 2110, 2129, 2150, 2329, 2401, 2494, 2553, 2999, 3093, 3123, 3157, 3200, 3222, 3225, 3237, 3301, 3434, 3482, 3515, 3614, 3652, 3684, 3700, 3736, 3866, 3935, 3977, 4135, 4237, 4341, 4366, 4415, 4479, 4507, 4616, 4710, 4759, 4847, 4895, 4911, 4970, 5011, 5105, 5242, 5280, 5367, 5516, 5564, 5577, 5589, 5608, 5634, 5654, 5672, 5690, 5716, 5759, 5834, 5855, 5859, 5885, 5915, 5926, 5932, 5966, 5976, 5982, 5986, 6006, 6009, 6029, 6064, 6078, 6093, 6113, 6136, 6148, 6157, 6163, 6202, 6267, 6277, 6290, 6298, 6312, 6351, 6400, 6467, 6471, 6479, 6482, 6496, 6516, 6523, 6565, 6576, 6616, 6628, 6633, 6667, 6677, 6697, 6707, 6717, 6747, 6773, 6857, 6893, 6925, 6935, 6943, 6946, 6956, 7006, 7104, 7137, 7148, 7152
    10-bit ADC Touch Driver Library 2093
    AAC Decoder Library 1260
    ADC Driver Library 1353
    ADC Peripheral Library 3237
    ADCHS Peripheral Library 3301
    ADCP Peripheral Library 3434
    AK4384 Codec Driver Library 1391
    AK4642 Codec Driver Library 1442
    AK4953 Codec Driver Library 1482
    AK7755 Codec Driver Library 1520
    Announce Module 6006
    AR1021 Touch Driver Library 2110
    ARP Module 6009
    Berkeley Module 6029
    BMX Peripheral Library 3482
    Bootloader Library 1173
    CAN Driver Library 1388
    CAN Peripheral Library 3515
    Class B Library 1191
    Clock System Service Library 5608
    Command Processor System Service Library 5634
    Comparator Driver Library 1558
    Comparator Peripheral Library 3614
    Console System Service Library 5654
    CPLD XC2C64A Driver Library 1559
    Crypto Library 1211
    CTMU Peripheral Library 3652
    Deadman Timer Peripheral Library 3684
    Debug System Service Library 5672
    Device Control Peripheral Library 3700
    Device Control System Service Library 5690
    DHCP Module 6064, 6093
    DHCP Server Module 6078
    DMA Peripheral Library 3736
    DMA System Service Library 5716
    DNS Module 6113
    DNSS Module 6136
    Driver Library 1336
    DSP Fixed-Point Math Library 2999
    Dual Data Rate (DDR) SDRAM Peripheral Library 3866
    Dynamic DNS Module 6148
    EBI Peripheral Library 3935
    ENC28J60 Driver Library 1569
    ENCx24J600 Driver Library 1589
    Ethernet MAC Driver Library 1608
    Ethernet Peripheral Library 3977
    Ethernet PHY Driver Library 1629
    File System Service Library 5759
    Flash Driver Library 1666
    Framework 3
    FTP Module 6157
    Generic USB Device Library 6935
    Graphics Driver Library 1673
    Graphics LCD (GLCD) Controller Peripheral Library 4135
    Graphics Library Porting Guide 2553
    HTTP Module 6163
    HTTP Net Module 6202
    I2C Peripheral Library 4237
    ICMP Module 6267
    ICMPv6 Module 6277
    Input Capture Driver Library 1821
    Input Capture Peripheral Library 4341
    Interrupt Peripheral Library 4366
    Interrupt System Service Library 5834
    Iperf Module 6290
    IPv4 Module 6298
    IPv6 Module 6312
    LibQ Fixed-Point 'C' Math Library 3093
    LibQ Fixed-Point Math Library 3123
    MAC Driver Module 6351
    Manager Module 6400

Memory System Service Library 5855
Messaging System Service Library 5859
MP3 Decoder Library 1270
MRF24W Wi-Fi Driver Library 2401
MRF24WN Wi-Fi Driver Library 2494
MTCH6301 Touch Driver Library 2129
MTCH6303 Touch Driver Library 2150
NBNS Module 6467
NDP Module 6471
Networking Presentation Layer 3157
NVM Driver Library 1830
NVM Peripheral Library 4415
Opus Decoder Library 1291
OSAL Library 3200
Oscillator Peripheral Library 4507
Output Compare Driver Library 1864
Output Compare Peripheral Library 4479
OVM7690 Camera Driver Library 1365
Peripheral Library Overview 3222
PIC32 Bluetooth Stack Library 4
PMP Driver Library 1867
PMP Peripheral Library 4616
Ports Peripheral Library 4759
Ports System Service Library 5885
Power Peripheral Library 4847
Prefetch Cache Peripheral Library 4710
Reboot Module 6479
Reset Peripheral Library 4895
RNG System Service Library 5915
RTCC Driver Library 1896
RTCC Peripheral Library 4911
Sample Library 5577
SD Card Driver Library 1900
SMTP Module 6482
SMTPC Module 6496
SNMP Module 6516
SNTP Module 6565
Speex Decoder Library 1307
SPI Driver Library 1929
SPI Flash Driver Library 1966
SPI Peripheral Library 5011
SQI Peripheral Library 5105
System Bus Peripheral Library 4970
System Service Library Overview 5589
TCP Module 6576
TCPIP Stack Library Overview 5982
TCPIP Stack Library Porting Guide 5986
Telnet Module 6628
Test Harness Library 6677
TFTP Module 6616
Timer Driver Library 2042
Timer Peripheral Library 5242
Timer System Service Library 5932
Touch System Service Library 5966
UDP Module 6633
USART Driver Library 2329
USART Peripheral Library 5280
USB Audio 1.0 Device Library 6717
USB Audio 2.0 Device Library 6747
USB Audio v1.0 Host Client Driver Library 7006
USB CDC Device Library 6857
USB CDC Host Library 7104
USB Device Layer Library 6773
USB Device Library - Getting Started 6697
USB HID Device Library 6893
USB HID Host Mouse Driver Library 7137
USB Host Layer Library 6956
USB Host Library - Getting Started 6943
USB Hub Host Client Driver Library 7148
USB MSD Device Library 6925
USB MSD Host Library 7152
USB Peripheral Library 5367
USBHS Peripheral Library 5516
Watchdog Timer Peripheral Library 5564
Watchdog Timer System Service Library 5976
WMA Decoder Library 1322
Zeroconf Module 6667
INVALID_SOCKET macro 6609
INVALID_TCP_PORT macro 6044
INVALID_UDP_SOCKET macro 6660
Invoking the Tasks Routine 6716
IP_ADD_MEMBERSHIP macro 6051
IP_ADDR type 6452
IP_ADDR_ANY macro 6045
IP_ADDRESS_TYPE enumeration 6453
ip_config.h 6312
IP_DROP_MEMBERSHIP macro 6051
IP_MULTI_ADDRESS union 6453
IP_MULTICAST_IF macro 6051
IP_MULTICAST_LOOP macro 6051
IP_MULTICAST_TTL macro 6052
IP_OPTIONS macro 6052
IP_TOS macro 6052
IP_TTL macro 6052
IP_VERSION_4 macro 6334
IP_VERSION_6 macro 6334
Iperf Module 6289
iperf_config.h 6297
IPPROTO_ICMPV6 macro 6052
IPPROTO_IP macro 6045
IPPROTO_IPV6 macro 6053
IPPROTO_TCP macro 6045
IPPROTO_UDP macro 6045
IPv4 Module 6298
ipv4.h 6311
IPV4_ADDR union 6453
IPV4_FRAGMENT_INFO union 6310
IPV4_HEADER structure 6308
IPV4_HEADER_TYPE enumeration 6308
IPV4_PACKET structure 6309
IPv6 Module 6312
ipv6.h 6348
IPV6_ACTION enumeration 6338
IPV6_ADDR union 6454
IPV6_ADDR_HANDLE type 6454
IPV6_ADDR_SCOPE enumeration 6454

IPV6_ADDR_STRUCT structure 6455
IPV6_ADDR_TYPE enumeration 6455
IPV6_ADDRESS_POLICY structure 6339
IPV6_ADDRESS_PREFERENCE enumeration 6339
IPV6_ADDRESS_TYPE union 6340
IPV6_CHECKSUM macro 6053
ipv6_config.h 6350
IPV6_DATA_DYNAMIC_BUFFER macro 6334
IPV6_DATA_NETWORK_FIFO macro 6334
IPV6_DATA_NONE macro 6335
IPV6_DATA_PIC_RAM macro 6335
IPV6_DATA_SEGMENT_HEADER structure 6340
IPV6_EVENT_HANDLER type 6341
IPV6_EVENT_TYPE enumeration 6341
IPV6_FRAGMENT_HEADER structure 6341
IPV6_HANDLE type 6342
IPV6_HEADER structure 6342
IPV6_HEADER_OFFSET_DEST_ADDR macro 6335
IPV6_HEADER_OFFSET_NEXT_HEADER macro 6335
IPV6_HEADER_OFFSET_PAYLOAD_LENGTH macro 6335
IPV6_HEADER_OFFSET_SOURCE_ADDR macro 6336
IPV6_JOIN_GROUP macro 6053
IPV6_LEAVE_GROUP macro 6053
IPV6_MULTICAST_HOPS macro 6053
IPV6_MULTICAST_IF macro 6054
IPV6_MULTICAST_LOOP macro 6054
IPV6_NEXT_HEADER_TYPE enumeration 6343
IPV6_NO_UPPER_LAYER_CHECKSUM macro 6336
IPV6_PACKET structure 6343
IPV6_PACKET_ACK_FNC type 6345
IPV6_RX_FRAGMENT_BUFFER structure 6345
IPV6_SEGMENT_TYPE enumeration 6346
IPV6_TLV_HBHO_PAYLOAD_JUMBOGRAM macro 6336
IPV6_TLV_HBHO_ROUTER_ALERT macro 6336
IPV6_TLV_OPTION_TYPE union 6346
IPV6_TLV_PAD_1 macro 6336
IPV6_TLV_PAD_N macro 6337
IPV6_TLV_UNREC_OPT_DISCARD_PP macro 6337
IPV6_TLV_UNREC_OPT_DISCARD_PP_NOT_MC macro 6337
IPV6_TLV_UNREC_OPT_DISCARD_SILENT macro 6337
IPV6_TLV_UNREC_OPT_SKIP_OPTION macro 6337
IPV6_ULA_FLAGS enumeration 6346
IPV6_ULA_RESULT enumeration 6347
IPV6_UNICAST_HOPS macro 6054
IPV6_V6ONLY macro 6054
is_bdaddr_command function 553
is_hconn_command function 553
isAACdecoder_enabled function 1264
isMP3decoder_enabled function 1275
isOPUSdecoder_enabled function 1298
isSPEEXdecoder_enabled function 1310
isWMAdecoder_enabled function 1330
iwpriv_adhocctx_set function 2459, 2530
IWPRIV_CMD enumeration 2543
iwpriv_config_read function 2459, 2531
iwpriv_config_write function 2531
IWPRIV_CONN_STATUS enumeration 2543
iwpriv_connstatus_get function 2532
iwpriv_devinfo_get function 2532
iwpriv_execute function 2541
IWPRIV_EXECUTE_PARAM union 2544
iwpriv_get function 2541
IWPRIV_GET_PARAM union 2544
iwpriv_initialconn_set function 2533
iwpriv_initstatus_get function 2533
iwpriv_is_servermode function 2534
iwpriv_leftclient_get function 2534
iwpriv_mcastfilter_set function 2535
iwpriv_nettype_get function 2535
iwpriv_nettype_set function 2536
iwpriv_numberofscanresults_get function 2536
IWPRIV_PARAM_CLIENTINFO structure 2544
IWPRIV_PARAM_CONFIG structure 2546
IWPRIV_PARAM_CONNECT structure 2546
IWPRIV_PARAM_CONTEXT structure 2545
IWPRIV_PARAM_DEVICEINFO structure 2545
IWPRIV_PARAM_DRIVERSTATUS structure 2546
IWPRIV_PARAM_MULTICASTFILTER structure 2546
IWPRIV_PARAM_NETWORKTYPE structure 2547
IWPRIV_PARAM_OPERATIONMODE structure 2547
IWPRIV_PARAM_POWERSAVE structure 2547
IWPRIV_PARAM_SCAN structure 2547
IWPRIV_PARAM_SSID structure 2548
iwpriv_powersave_config function 2536
iwpriv_prescan_isfinished function 2541
iwpriv_prescan_option_get function 2541
iwpriv_prescan_option_set function 2542
iwpriv_prescan_start function 2537
iwpriv_scan_start function 2537
IWPRIV_SCAN_STATUS enumeration 2545
iwpriv_scanresult_get function 2538
iwpriv_scanresults_display function 2538
iwpriv_scanresults_save function 2539
iwpriv_scanstatus_get function 2539
iwpriv_set function 2542
IWPRIV_SET_PARAM union 2545
iwpriv_ssid_get function 2540
iwpriv_ssid_set function 2540
IWPRIV_STATUS enumeration 2543

K

KHAKI macro 2716

L

L1 Cache 5692
l2cap.h 1133
L2CAP_ALLOCATE_BUFFERS function 18
L2CAP_ALLOCATE_BUFFERS_FUNCTION macro 39
L2CAP_ALLOCATE_BUFFERS_RAM_SIZE_VAR macro 40
L2CAP_ALLOCATE_BUFFERS_VARS macro 40
L2CAP_CHANNEL_FLAG_FORCE_HCI_DISCONNECT macro 778
L2CAP_CHANNEL_FLAG_INCOMING macro 734
L2CAP_CHANNEL_FLAG_SENDING macro 734
L2CAP_CMD_CONFIG_REQUEST macro 734
L2CAP_CMD_CONFIG_RESPONSE macro 734
L2CAP_CMD_CONN_PARAM_UPDATE_REQUEST macro 734

L2CAP_CMD_CONN_PARAM_UPDATE_RESPONSE macro 735
L2CAP_CMD_CONN_REQUEST macro 735
L2CAP_CMD_CONN_RESPONSE macro 735
L2CAP_CMD_DATA_LEN_CMD_REJECT macro 735
L2CAP_CMD_DATA_LEN_CONFIG_REQUEST macro 735
L2CAP_CMD_DATA_LEN_CONFIG_RESPONSE macro 736
L2CAP_CMD_DATA_LEN_CONN_REQUEST macro 736
L2CAP_CMD_DATA_LEN_CONN_RESPONSE macro 736
L2CAP_CMD_DATA_LEN_DCONN_REQUEST macro 736
L2CAP_CMD_DATA_LEN_DCONN_RESPONSE macro 736
L2CAP_CMD_DATA_LEN_ECHO_REQUEST macro 737
L2CAP_CMD_DATA_LEN_ECHO_RESPONSE macro 737
L2CAP_CMD_DATA_LEN_INFO_REQUEST macro 737
L2CAP_CMD_DATA_LEN_INFO_RESPONSE macro 737
L2CAP_CMD_DCONN_REQUEST macro 737
L2CAP_CMD_DCONN_RESPONSE macro 738
L2CAP_CMD_ECHO_REQUEST macro 738
L2CAP_CMD_ECHO_RESPONSE macro 738
L2CAP_CMD_HEADER_LEN macro 738
L2CAP_CMD_INFO_REQUEST macro 738
L2CAP_CMD_INFO_RESPONSE macro 739
L2CAP_CMD_LAST macro 739
L2CAP_CMD_REJECT macro 739
L2CAP_CMD_RESERVED macro 739
L2CAP_CMD_STATUS_BEING_SENT macro 739
L2CAP_CMD_STATUS_PENDING macro 740
L2CAP_CMD_STATUS_WAITING_RESPONSE macro 740
l2cap_command_queue.h 1135
l2cap_config.h 1135
l2cap_config_handlers.h 1136
L2CAP_CONFIG_RESULT_REJECTED macro 740
L2CAP_CONFIG_RESULT_SUCCESS macro 740
L2CAP_CONFIG_RESULT_UNACCEPTABLE_PARAMETER macro 740
L2CAP_CONFIG_RESULT_UNKNOWN_OPTION macro 741
L2CAP_CONN_REQ_RESULT_INVALID_PSM macro 741
L2CAP_CONN_REQ_RESULT_INVALID_SOURCE_CID macro 781
L2CAP_CONN_REQ_RESULT_NO_RESOURCES macro 741
L2CAP_CONN_REQ_RESULT_PENDING macro 741
L2CAP_CONN_REQ_RESULT_SECURITY_BLOCK macro 741
L2CAP_CONN_REQ_RESULT_SRC_CID_ALREADY_ALLOCATED macro 781
L2CAP_CONN_REQ_RESULT_SUCCESS macro 742
L2CAP_CONN_REQ_STATUS_AUTHENTICATION_PENDING macro 742
L2CAP_CONN_REQ_STATUS_AUTHORIZATION_PENDING macro 742
L2CAP_CONN_REQ_STATUS_NO_INFO macro 742
L2CAP_DECL_ERETR_FUNCTIONS macro 47
L2CAP_DEFAULT_ERTX macro 742
L2CAP_DEFAULT_RTX macro 743
L2CAP_ECHO_MAX_DATA_LEN macro 743
l2cap_eretr.h 1137
L2CAP_ERETR_EVENT_DATA_REQUEST enumeration member 772
L2CAP_ERETR_EVENT_LOCAL_BUSY_CLEAR enumeration member 772
L2CAP_ERETR_EVENT_LOCAL_BUSY_DETECTED enumeration member 772
L2CAP_ERETR_EVENT_MONITOR_TIMER_EXPIRED enumeration member 772
L2CAP_ERETR_EVENT_RECV_FBIT enumeration member 772
L2CAP_ERETR_EVENT_RECV_REQSEQ_AND_FBIT enumeration member 772
L2CAP_ERETR_EVENT_RETRANSMIT_TIMER_EXPIRED enumeration member 772
L2CAP_ERETR_RECV_STATE_RECV macro 743
L2CAP_ERETR_RECV_STATE_REJ_SENT macro 743
L2CAP_ERETR_RECV_STATE_SREJ_SENT macro 743
L2CAP_ERETR_XMIT_STATE_WAIT_ACK macro 744
L2CAP_ERETR_XMIT_STATE_WAIT_F macro 744
L2CAP_ERETR_XMIT_STATE_XMIT macro 744
L2CAP_EXT_BI_QOS macro 744
L2CAP_EXT_ENHANCED_RETRANSMISSION macro 744
L2CAP_EXT_EXT_FLOW_SPEC_BR_EDR macro 779
L2CAP_EXT_EXT_WINDOW_SIZE macro 779
L2CAP_EXT_FCS_OPTION macro 745
L2CAP_EXT_FEATURES_ENABLED macro 779
L2CAP_EXT_FIXED_CHANNELS macro 779
L2CAP_EXT_FLOW_CONTROL macro 745
L2CAP_EXT_RETRANSMISSION macro 745
L2CAP_EXT_STREAMING macro 745
L2CAP_EXT_UNICAST_CONNLESS_DATA_RCPT macro 779
l2cap_fixed_channel.h 1138
L2CAP_FIXED_CHANNELS_DECL macro 41
L2CAP_FRAME_TYPE_I macro 745
L2CAP_FRAME_TYPE_S macro 746
L2CAP_HCI_CONNECT_PACKET_TYPE macro 746
L2CAP_HCI_CONNECT_PAGE_SCAN_REPETITION_MODE macro 746
L2CAP_HCI_CONNECT_ROLE_SWITCH macro 746
L2CAP_HCI_PACKET_TYPE macro 41
L2CAP_HCI_PAGE_SCAN_REPETITION_MODE macro 41
L2CAP_HCI_ROLE_SWITCH macro 41
L2CAP_HEADER_LEN macro 746
L2CAP_IDLE_CONNECTION_TIMEOUT macro 42
L2CAP_IDLE_HCI_CONNECTION_TIMEOUT macro 747
L2CAP_INFO_NOT_SUPPORTED macro 747
L2CAP_INFO_RESULT_SUCCESS macro 747
L2CAP_INFO_TYPE_CONNECTIONLESS_MTU macro 747
L2CAP_INFO_TYPE_EXTENDED_SUPPORT macro 747
L2CAP_INFO_TYPE_FIXED_CHANNELS macro 780
L2CAP_LINK_TYPE_BD_EDR macro 748
L2CAP_LINK_TYPE_LE macro 748
L2CAP_MAX_CMD_QUEUE_LEN macro 748
L2CAP_MAX_FIXED_CHANNELS macro 47
L2CAP_MAX_FRAME_BUFFERS macro 748
L2CAP_MAX_MANAGERS macro 748
L2CAP_MAX_MTU macro 749
L2CAP_MAX_OPTIONS macro 749
L2CAP_MAX_RTX macro 749
L2CAP_MGR_STATE_FREE macro 749
L2CAP_MGR_STATE_LISTENING_L2CAP macro 749
L2CAP_MGR_STATE_USED macro 750
L2CAP_MIN_MTU macro 750
L2CAP_MONITOR_TIMEOUT macro 750
L2CAP_OPTION_LEN_FLASH_QOS macro 750
L2CAP_OPTION_LEN_FLASH_RFC macro 750
L2CAP_OPTION_LEN_FLASH_TIMEOUT macro 751
L2CAP_OPTION_LEN_MAX_MTU macro 751

L2CAP_OPTION_LEN_UNKNOWN macro 751
L2CAP_OPTION_LEN_UNKNOWN_DATA macro 751
L2CAP_OPTION_TYPE_FLASH_QOS macro 751
L2CAP_OPTION_TYPE_FLASH_QOS_FLAG macro 752
L2CAP_OPTION_TYPE_FLASH_RFC macro 752
L2CAP_OPTION_TYPE_FLASH_RFC_FLAG macro 752
L2CAP_OPTION_TYPE_FLASH_TIMEOUT macro 752
L2CAP_OPTION_TYPE_FLASH_TIMEOUT_FLAG macro 752
L2CAP_OPTION_TYPE_MAX_MTU macro 753
L2CAP_OPTION_TYPE_MAX_MTU_FLAG macro 753
L2CAP_OPTION_TYPE_UNKNOWN_FLAG macro 753
l2cap_packet.h 1138
L2CAP_PACKET_DATA_TYPE_RAW macro 753
L2CAP_PACKET_DATA_TYPE_SMART macro 753
l2cap_private.h 1138
l2cap_psm.h 1140
L2CAP_REJECT_REASON_INVALID_CHANNEL macro 754
L2CAP_REJECT_REASON_MTU_EXCEEDED macro 754
L2CAP_REJECT_REASON_NOT_UNDERSTOOD macro 754
L2CAP_RETR_TIMEOUT macro 754
L2CAP_RFC_BASIC macro 754
L2CAP_RFC_ENHANCED_RETRANSMISSION macro 755
L2CAP_RFC_ERETR_TIMEOUT macro 755
L2CAP_RFC_ERETR_TX_WINDOW macro 755
L2CAP_RFC_FLOW_CONTROL macro 755
L2CAP_RFC_MONITOR_TIMEOUT macro 755
L2CAP_RFC_RETRANSMISSION macro 756
L2CAP_RFC_STREAMING macro 756
L2CAP_SAR_SDU_CONTINUE macro 756
L2CAP_SAR_SDU_START macro 756
L2CAP_SAR_SDU_STOP macro 756
L2CAP_SAR_UNSEGMENTED macro 757
L2CAP_SERVICE_TYPE_BEST_EFFORT macro 757
L2CAP_SERVICE_TYPE_GUARANTEED macro 757
L2CAP_SERVICE_TYPE_NO_TRAFFIC macro 757
L2CAP_SFUNCTION_REJ macro 757
L2CAP_SFUNCTION_RNR macro 758
L2CAP_SFUNCTION_RR macro 758
L2CAP_SFUNCTION_SREJ macro 758
l2cap_signal.h 1140
l2cap_test.h 1140
l2cap_timer.h 1140
Layer Management 4138
LAYER_REGISTERS structure 1705
LCC_VSYNC_Callback_FnPtr type 1699
LCC_VSYNC_STATE enumeration 1699
LibQ Fixed-Point 'C' Math Library 3093
LibQ Fixed-Point Math Library 3123
libq.h 3154
libq_C.h 3120
libq_q15_Abs_q15 function 3096
libq_q15_Add_q15_q15 function 3096
libq_q15_DivisionWithSaturation_q15_q15 function 3096
libq_q15_ExpAvg_q15_q15_q1d15 function 3104
libq_q15_ExtractH_q31 function 3097
libq_q15_ExtractL_q31 function 3097
libq_q15_Negate_q15 function 3097
libq_q15_RoundL_q31 function 3098
libq_q15_ShiftLeft_q15_q15 function 3098
libq_q15_ShiftRight_q15_q15 function 3098
libq_q15_ShiftRightRound_q15_q15 function 3099
libq_q15_Sub_q15_q15 function 3104
libq_q1d15_MacR_q1d31_q1d15_q1d15 function 3105
libq_q1d15_MsuR_q1d31_q1d15_q1d15 function 3105
libq_q1d15_MultipyR2_q1d15_q1d15 function 3106
libq_q1d15_Sin_q10d6 function 3099
libq_q1d31_Mac_q1d31_q1d15_q1d15 function 3106
libq_q1d31_Msu_q1d31_q1d15_q1d15 function 3107
libq_q20d12_Sin_q20d12 function 3103
libq_q31_Abs_q31 function 3100
libq_q31_Add_q31_q31 function 3100
libq_q31_DepositH_q15 function 3100
libq_q31_DepositL_q15 function 3101
libq_q31_Mult2_q15 function 3101
libq_q31_Multi_q15_q31 function 3102
libq_q31_Negate_q31 function 3102
libq_q31_ShiftLeft_q31_i16 function 3107
libq_q31_ShiftRight_q31_q15 function 3102
libq_q31_ShiftRightRound_q31_q15 function 3103
Library Architecture
    USB Device Library 6697
    USB Host Library 6944
Library Configuration
    USB CDC Device Library 6869
    USB HID Device Library 6904
Library Dependencies 2576
    Graphics Library 2576
Library Initialization 6775, 6860, 6895, 6926, 6936
    USB CDC Device Library 6860
    USB HID Device Library 6895
Library Interface 54, 1185, 1195, 1213, 1260, 1271, 1294, 1310, 1323, 1347, 1353, 1358, 1369, 1388, 1406, 1454, 1491, 1530, 1558, 1560, 1573, 1593, 1613, 1634, 1666, 1679, 1741, 1785, 1822, 1839, 1864, 1876, 1896, 1908, 1940, 1978, 2052, 2097, 2116, 2135, 2153, 2222, 2281, 2350, 2410, 2500, 2590, 3001, 3094, 3125, 3159, 3207, 3252, 3316, 3441, 3487, 3525, 3617, 3663, 3686, 3701, 3747, 3869, 3937, 3994, 4140, 4260, 4344, 4369, 4422, 4485, 4513, 4625, 4715, 4765, 4850, 4897, 4918, 4973, 5035, 5115, 5250, 5294, 5376, 5517, 5566, 5579, 5593, 5616, 5637, 5661, 5676, 5694, 5727, 5779, 5839, 5857, 5866, 5892, 5918, 5929, 5945, 5967, 5978, 6014, 6031, 6067, 6083, 6102, 6119, 6141, 6151, 6160, 6174, 6217, 6269, 6280, 6303, 6318, 6359, 6404, 6469, 6477, 6487, 6502, 6527, 6570, 6584, 6621, 6631, 6640, 6669, 6684, 6727, 6756, 6794, 6870, 6905, 6931, 6943, 6968, 7034, 7115, 7143, 7151, 7156
    10-bit ADC Touch Driver Library 2097
    AAC Decoder Library 1260
    ADC Driver Library 1353
    ADC Peripheral Library 3252
    ADCHS Peripheral Library 3316
    ADCP Peripheral Library 3441
    AK4384 Codec Driver Library 1406
    AK4642 Codec Driver Library 1454
    AK4953 Codec Driver Library 1491
    AK7755 Codec Driver Library 1530
    AR1021 Touch Driver Library 2116
    ARP Module 6014
    Berkeley Module 6031
    BMX Peripheral Library 3487
    Bootloader Library 1185

Camera Driver Library 1369
CAN Driver Library 1388
CAN Peripheral Library 3525
Class B Library 1195
Clock System Service Library 5616
Command Processor System Service Library 5637
Comparator Driver Library 1558
Comparator Peripheral Library 3617
Console System Service Library 5661
CPLD XC2C64A Driver Library 1560
Crypto Library 1213
CTMU Peripheral Library 3663
Deadman Timer Peripheral Library 3686
Debug System Service Library 5676
Device Control Peripheral Library 3701
Device Control System Service Library 5694
DHCP Module 6067
DHCP Server Module 6083
DHCPv6 Module 6102
DMA Peripheral Library 3747
DMA System Service Library 5727
DNS Module 6119
DNSS Module 6141
DSP Fixed-Point Math Library 3001
Dual Data Rate (DDR) SDRAM Peripheral Library 3869
Dynamic DNS Module 6151
EBI Peripheral Library 3937
Ethernet MAC Driver Library 1613
Ethernet Peripheral Library 3994
Ethernet PHY Driver Library 1634
File System Service Library 5779
Flash Driver Library 1666
FTP Module 6160
Generic USB Device Library 6943
GLCD Controller Peripheral Library 4140
Graphics Driver Library 1679
HTTP Module 6174
HTTP Net Module 6217
I2C Peripheral Library 4260
ICMP Module 6269
ICMPv6 Module 6280
Input Capture Driver Library 1822
Input Capture Peripheral Library 4344
Interrupt Peripheral Library 4369
Interrupt System Service Library 5839
IPv4 Module 6303
IPv6 Module 6318
LibQ Fixed-Point 'C' Math Library 3094
LibQ Fixed-Point Math Library 3125
MAC Driver Module 6359
Manager Module 6404
Memory System Service Library 5857
Messaging System Service Library 5866
MP3 Decoder Library 1271
MRF24W Wi-Fi Library 2410
MRF24WN Wi-Fi Library 2500
MTCH6301 Touch Driver Library 2135
MTCH6303 Touch Driver Library 2153
NBNS Module 6469
NDP Module 6477
Networking Presentation Layer 3159
NVM Driver Library 1839
NVM Peripheral Library 4422
Opus Decoder Library 1294
OSAL Library 3207
Oscillator Peripheral Library 4513
Output Compare Driver Library 1864
Output Compare Peripheral Library 4485
PIC32 Bluetooth Stack Library 54
PMP Driver Library 1876
PMP Peripheral Library 4625
Ports Peripheral Library 4765
Ports System Service Library 5892
Power Peripheral Library 4850
Prefetch Cache Peripheral Library 4715
Random Number Generator System Service Library 5918
Reset Peripheral Library 4897
RTCC Driver Library 1896
RTCC Peripheral Library 4918
Sample Library 5579
SD Card Driver Library 1908
SMTP Module 6487
SMTPC Module 6502
SNMP Module 6527
SNTP Module 6570
Speex Decoder Library 1310
SPI Driver Library 1940
SPI Flash Driver Library 1978
SPI Peripheral Library 5035
SQI Peripheral Library 5115
System Bus Peripheral Library 4973
System Service Library 5593
TCP Module 6584
Telnet Module 6631
Test Harness Library 6684
TFTP Module 6621
Timer Driver Library 2052
Timer Peripheral Library 5250
Timer System Service Library 5945
Touch System Service Library 5967
UDP Module 6640
USART Driver Library 2350
USART Peripheral Library 5294
USB Audio 1.0 Device Library 6727
USB Audio 2.0 Device Library 6756
USB Audio v1.0 Host Client Driver Library 7034
USB CDC Device Library 6870
USB Device Layer Library 6794
USB HID Device Library 6905
USB HID Host Mouse Driver Library 7143
USB Host Layer Library 6968
USB Hub Host Client Driver Library 7151
USB MSD Device Library 6931
USB Peripheral Library 5376
USBHS Peripheral Library 5517
Watchdog Timer Peripheral Library 5566

Watchdog Timer System Service Library 5978
WMA Decoder Library 1323
Zeroconf Module 6669
Library Overview 6, 1174, 1192, 1212, 1260, 1271, 1293, 1309, 1323, 1366, 1393, 1443, 1482, 1522, 1559, 1570, 1590, 1610, 1631, 1674, 1729, 1769, 1831, 1868, 1902, 1931, 1967, 2043, 2094, 2111, 2130, 2151, 2211, 2269, 2329, 2403, 2495, 2563, 2999, 3123, 3158, 3202, 3240, 3304, 3435, 3483, 3517, 3615, 3655, 3685, 3700, 3737, 3867, 3936, 3980, 4136, 4239, 4342, 4368, 4417, 4481, 4509, 4617, 4712, 4763, 4850, 4896, 4912, 4971, 5014, 5108, 5245, 5282, 5371, 5517, 5565, 5577, 5610, 5635, 5655, 5674, 5691, 5717, 5770, 5835, 5856, 5860, 5889, 5916, 5927, 5936, 5966, 5977, 6010, 6030, 6065, 6079, 6094, 6115, 6138, 6149, 6157, 6164, 6203, 6267, 6279, 6299, 6313, 6352, 6401, 6468, 6473, 6483, 6496, 6517, 6566, 6577, 6617, 6630, 6634, 6667, 6677, 6718, 6748, 6774, 6857, 6895, 6926, 6936, 6957, 7006, 7104, 7138, 7148, 7153
10-bit ADC Touch Driver Library 2094
AAC Decoder Library 1260
ADCHS Peripheral Library 3304
ADCP Peripheral Library 3435
AK4384 Driver Library 1393
AK4642 Driver Library 1443
AK4953 Driver Library 1482
AK7755 Driver Library 1522
AR1021 Touch Driver Library 2111
ARP Module 6010
Berkeley Module 6030
Bootloader Library 1174
CAN Peripheral Library 3517
Class B Library 1192
Clock System Service Library 5610
Command Processor System Service Library 5635
Comparator Peripheral Library 3615
CPLD XC2C64A Driver Library 1559
Crypto Library 1212
CTMU Peripheral Library 3655
Deadman Timer Peripheral Library 3685
Debug System Service Library 5674
Device Control Peripheral Library 3700
Device Control System Service Library 5691
DHCP Module 6065
DHCP Server Module 6079
DHCPv6 Module 6094
DMA Peripheral Library 3737
DMA System Service Library 5717
DNS Module 6115
DNSS Module 6138
DSP-Fixed Point Math Library 2999
Dual Data Rate (DDR) SDRAM Peripheral Library 3867
Dynamic DNS Module 6149
Ethernet MAC Driver Library 1610
Ethernet PHY Driver Library 1631
File System Service Library 5771
FTP Module 6157
Generic USB Device Library 6936
GLCD Controller Peripheral Library 4136
Graphics Driver Library 1674
HTTP Module 6164
HTTP Net Module 6203
I2C Peripheral Library 4239
ICMP Module 6267
ICMPv6 Module 6279
Interrupt Peripheral Library 4368
Interrupt System Service Library 5835
IPv4 Module 6299
IPv6 Module 6313
LibQ Fixed-Point Math Library 3123
MAC Driver Module 6352
Manager Module 6401
Memory System Service Library 5856
Messaging System Service Library 5860
MP3 Decoder Library 1271
MRF24W Wi-Fi Driver Library 2403
MRF24WN Wi-Fi Driver Library 2495
MTCH6301 Touch Driver Library 2130
MTCH6303 Touch Driver Library 2151
NBNS Module 6468
NDP Module 6473
Networking Presentation Layer 3158
NVM Driver Library 1831
NVM Peripheral Library 4417
Opus Decoder Library 1293
OSAL Library 3202
Oscillator Peripheral Library 4509
Peripheral Library 3240
PIC32 Bluetooth Stack Library 6
PMP Driver Library 1868
PMP Peripheral Library 4617
Ports Peripheral Library 4763
Ports System Service Library 5889
Power Peripheral Library 4850
Prefetch Cache Peripheral Library 4712
Random Number Generator System Service Library 5916
Reset Peripheral Library 4896
RTCC Peripheral Library 4912
SAMPLE Library 5577
SD Card Driver Library 1902
SMTP Module 6483
SMTPC Module 6496
SNMP Module 6517
SNTP Module 6566
Speex Decoder Library 1309
SPI Driver Library 1931
SPI Flash Driver Library 1967
SPI Peripheral Library 5014
SQI Peripheral Library 5108
System Console System Service Library 5655
TCP Module 6577
Telnet Module 6630
TEST HARNESS LIB Library 6677
TFTP Module 6617
Timer Driver Library 2043
Timer Peripheral Library 5245
Timer System Service Library 5936
Touch System Service Library 5966
UDP Module 6634
USART Driver Library 2329
USART Peripheral Library 5282
USB Audio 1.0 Device Library 6718

USB Audio 2.0 Device Library 6748
USB Audio v1.0 Host Client Driver Library 7006
USB CDC Device Library 6857
USB Device Layer Library 6774
USB HID Device Library 6895
USB HID Host Mouse Driver Library 7138
USB Hub Host Client Driver Library 7148
USB MSD Host Library 7153
USB Peripheral Library 5371
USBHS Peripheral Library 5517
Watchdog Timer Peripheral Library 5565
Watchdog Timer System Service Library 5977
WMA Decoder Library 1323
Zeroconf Module 6667
LIBRARY_UNDER_TEST structure 6691
LIGHTBLUE macro 2716
LIGHTCYAN macro 2716
LIGHTGRAY macro 2717
LIGHTGREEN macro 2717
LIGHTMAGENTA macro 2717
LIGHTORANGE macro 2717
LIGHTRED macro 2717
LIGHTYELLOW macro 2718
Line Rendering 2567
Line Rendering Functions 2594
LINE_TERM macro 5644
linger structure 6057
Link Local (Zeroconf) 6668
List Box Object 2765
listen function 6036
LKS_FIRST_KEY_ADDR macro 211
LKS_MAX_LINK_KEYS macro 211
LKS_SIGNATURE macro 211
LKS_SIGNATURE_ADDR macro 211
lm.h 1132
LM_PACKET_TYPE_DH1 macro 672
LM_PACKET_TYPE_DH3 macro 673
LM_PACKET_TYPE_DH5 macro 673
LM_PACKET_TYPE_DM1 macro 673
LM_PACKET_TYPE_DM3 macro 673
LM_PACKET_TYPE_DM5 macro 673
LOG macro 212
LOG_EX macro 212
LOG102Q5D11 macro 3113
LOGADDR macro 212
LOGADDR_EX macro 212
LOGCLEAR macro 212
LOGINT macro 213
LOGINT_EX macro 213
LOGMEMORY macro 213
LOGMEMORY_EX macro 213
LOGWRITE macro 213

M

MAC Driver Module 6351
MAGENTA macro 2718
Main Program Changes 5994
MAIN_RETURN macro 5594
MAIN_RETURN_CODE macro 5603
MAIN_RETURN_CODES enumeration 5598
MAJOR_VERSION macro 1186
Manager Module 6400
Managing Slave Addresses 4256
MAP_ALLOCATE_BUFFERS function 204
mask enumeration 6991
Master Descriptor Table 6775
Master Mode 5030
Math Libraries Help 2999
matrix32 structure 3085
max macro 978
MAX_BSD_SOCKETS macro 6030
MAX_CMD_ARGS macro 5644
MAX_CMD_GROUP macro 5644
MAX_NONBUFFERED_BYTE_COUNT macro 1893
MAX_SAMPLES macro 1333
MAX_SBC_DEC_STATE_SIZE macro 833
MAX_STACK macro 1268
MAX16 macro 3087
MAX32 macro 3087
MAXFRACT16 macro 3113
MAXFRACT32 macro 3113
MAXINT16 macro 3113
MAXINT32 macro 3113
MAXPFLOAT32 macro 3114
MC_CRYPTO_API_H macro 1257
MCHP_BITMAP_NORMAL macro 2953
MCHP_BITMAP_PALETTE_STR macro 2953
mDNS 6668
MDNSD_ERR_CODE enumeration 6674
Measuring Pulse Width 3660
Measuring Time Between Pulses 3659
Media Independent Interface (MII) 3983
Memory Access Control 3485
Memory Operation 3205
Memory System Service Library 5855
Memory to Memory Transfer 5724
Messaging System Service Library 5859
Meter Object 2780
MIB Browsers 6524
MIB Files 6524
Migrating Applications from v1.03.01 and Earlier Releases of MPLAB Harmony 1825
min macro 978
MIN16 macro 3087
MIN32 macro 3087
MINFRACT16 macro 3114
MINFRACT32 macro 3114
MININT16 macro 3114
MININT32 macro 3114
MINOR_VERSION macro 1187
MINPFLOAT32 macro 3115
Miscellaneous 5891, 5942
Miscellaneous Control 4140
Miscellaneous Functions 4421
MK_CMD_ADDRESS macro 798
MK_DLCI macro 798

MLA and MPLAB Harmony USB Device Stack Files 6712
Mode Configuration 3519
Modification 2045
Module Events 3525
Mounting a Volume 5764
Mouse Data Event Handling 7139
MP3 Decoder Library 1270
mp3.h 1288
MP3_CHANNEL_INFO structure 1287
MP3_DEC structure 1288
MP3_Decode function 1272
MP3_ERROR_COUNT_MAX macro 1280
MP3_EVENT enumeration 1279
MP3_EventHandler function 1273
MP3_FORMAT structure 1278
MP3_FRAME_HEADER union 1279
MP3_GetAudioSize function 1277
MP3_GetChannels function 1278
MP3_HEADER_CHANNELS_DUAL macro 1280
MP3_HEADER_CHANNELS_JOINT macro 1281
MP3_HEADER_CHANNELS_MONO macro 1281
MP3_HEADER_CHANNELS_STEREO macro 1281
MP3_HEADER_SAMPLERATE_32000 macro 1281
MP3_HEADER_SAMPLERATE_44100 macro 1281
MP3_HEADER_SAMPLERATE_48000 macro 1282
MP3_HEADER_SAMPLERATE_RESV macro 1282
MP3_Initialize function 1278
MP3_ParseVBR function 1274
MP3_PLAYTIME_H macro 1282
MP3_RegisterDecoderEventHandlerCallback function 1278
MP3_STATE enumeration 1280
MP3_STATE_SIZE macro 1279
MP3_UpdatePlaytime function 1278
MP3_XING_HEADER structure 1280
MP3_XING_HEADER_START_MONO macro 1282
MP3_XING_HEADER_START_STEREO macro 1282
MP3Decode function 1273, 1277
MP3DecoderGetStateSize function 1274
MP3DecoderInit function 1275
MP3Initialize function 1272
MP3MPEG1L3_SAMPLES_PER_FRAME macro 1287
MP3MPEG2L3_SAMPLES_PER_FRAME macro 1287
MPEG_BITRATE_0000 macro 256
MPEG_BITRATE_0001 macro 256
MPEG_BITRATE_0010 macro 256
MPEG_BITRATE_0011 macro 256
MPEG_BITRATE_0100 macro 257
MPEG_BITRATE_0101 macro 257
MPEG_BITRATE_0110 macro 257
MPEG_BITRATE_0111 macro 257
MPEG_BITRATE_1000 macro 257
MPEG_BITRATE_1001 macro 258
MPEG_BITRATE_1010 macro 258
MPEG_BITRATE_1011 macro 258
MPEG_BITRATE_1100 macro 258
MPEG_BITRATE_1101 macro 258
MPEG_BITRATE_1110 macro 259
MPEG_BITRATE_ALL macro 259
MPEG_CHANNEL_MODE_ALL macro 259
MPEG_CHANNEL_MODE_DUAL_CHANNEL macro 259
MPEG_CHANNEL_MODE_JOINT_STEREO macro 259
MPEG_CHANNEL_MODE_MONO macro 260
MPEG_CHANNEL_MODE_STEREO macro 260
MPEG_CRC_PROTECTION_NOT_SUPPORTED macro 260
MPEG_CRC_PROTECTION_SUPPORTED macro 260
MPEG_LAYER_1 macro 260
MPEG_LAYER_2 macro 261
MPEG_LAYER_3 macro 261
MPEG_LAYER_ALL macro 261
MPEG_MPF_1 macro 261
MPEG_MPF_2 macro 261
MPEG_SAMPLING_FREQUENCY_16000 macro 262
MPEG_SAMPLING_FREQUENCY_22050 macro 262
MPEG_SAMPLING_FREQUENCY_24000 macro 262
MPEG_SAMPLING_FREQUENCY_32000 macro 262
MPEG_SAMPLING_FREQUENCY_44100 macro 262
MPEG_SAMPLING_FREQUENCY_48000 macro 263
MPEG_SAMPLING_FREQUENCY_ALL macro 263
MPEG_VBR_NOT_SUPPORTED macro 263
MPEG_VBR_SUPPORTED macro 263
mpfs2.jar Utility 6205
MPLAB Harmony Graphics Library 2553
MPLAB Harmony Graphics Library Access Changes 2555
MPLAB Harmony Graphics Library API Changes 2555
MPLAB Harmony Graphics Library Design 2554
MPLAB Harmony Graphics Library File Name Compliance 2556
MPLAB Harmony Graphics Library Function Name Compliance 2557
MPLAB Harmony Graphics Library Initialization Changes 2556
MPLAB Harmony Graphics Library Key Features 2554
MPLAB Harmony Graphics Library Structure 2554
MPLAB Harmony Graphics Library Utilities 2556
MPLAB Harmony Module System Interface 5590
MPLAB Harmony TCP/IP Stack Access Changes 5995
MPLAB Harmony TCP/IP Stack API Changes 5988
MPLAB Harmony TCP/IP Stack Configuration 5992
MPLAB Harmony TCP/IP Stack Configuration Changes 5992
MPLAB Harmony TCP/IP Stack Design 5988
MPLAB Harmony TCP/IP Stack Function Name Compliance 5996
MPLAB Harmony TCP/IP Stack Heap Configuration 5993
MPLAB Harmony TCP/IP Stack Initialization Changes 5994
MPLAB Harmony TCP/IP Stack Key Features 5986
MPLAB Harmony TCP/IP Stack SSL/RSA Usage 5995
MPLAB Harmony TCP/IP Stack Storage Changes 5992
MPLAB Harmony TCP/IP Stack Structure 5987
MPLAB Harmony TCP/IP Stack Utilities 5995
MRF24WG Wi-Fi Driver Library 2401
MRF24WN Wi-Fi Driver Library 2494
MSBBITFRACT16 macro 3115
MSBBITFRACT32 macro 3115
MSD TPL Table Configuration 7153
MTCH6301 Touch Driver Library 2129
MTCH6303 Touch Driver Library 2150
mul16 function 3083
mul16r function 3083
mul32 function 3083
Multi-instance Support 2576

Mutex Operation 3204

N

NBNS Module 6467
nbns.h 6471
nbns_config.h 6471
NDP Module 6471
ndp.h 6478
ndp_config.h 6478
net_pres.h 3195
NET_PRES_ADDRESS structure 3176
net_pres_certstore.h 3196
NET_PRES_CertStoreGetCACerts function 3175
NET_PRES_CertStoreGetServerCert function 3176
NET_PRES_Deinitialize function 3161
NET_PRES_EncProviderConnect type 3177
NET_PRES_EncProviderConnectionClose type 3177
NET_PRES_EncProviderDeinit type 3177
NET_PRES_EncProviderInit type 3178
NET_PRES_EncProviderIsInitialized type 3178
NET_PRES_EncProviderObject structure 3179
NET_PRES_EncProviderOpen type 3179
NET_PRES_EncProviderRead type 3180
NET_PRES_EncProviderReadReady type 3164
NET_PRES_EncProviderWrite type 3180
NET_PRES_EncProviderWriteReady type 3194
net_pres_encryptionproviderapi.h 3198
NET_PRES_EncSessionStatus enumeration 3181
NET_PRES_INDEX type 3181
NET_PRES_INIT_DATA structure 3182
NET_PRES_Initialize function 3162
NET_PRES_INST_DATA structure 3182
NET_PRES_INVALID_SOCKET macro 3193
NET_PRES_Reinitialize function 3162
NET_PRES_SIGNAL_FUNCTION type 3183
NET_PRES_SIGNAL_HANDLE type 3183
NET_PRES_SKT_ADDR_T enumeration 3184
NET_PRES_SKT_ERROR_T enumeration 3184
NET_PRES_SKT_HANDLE_T type 3184
NET_PRES_SKT_OPTION_TYPE enumeration 3184
NET_PRES_SKT_PORT_T type 3194
NET_PRES_SKT_T enumeration 3185
net_pres_socketapi.h 3196
NET_PRES_SocketBind function 3165
NET_PRES_SocketClose function 3166
NET_PRES_SocketConnect function 3166
NET_PRES_SocketDiscard function 3166
NET_PRES_SocketDisconnect function 3167
NET_PRES_SocketEncryptSocket function 3167
NET_PRES_SocketFlush function 3167
NET_PRES_SocketGetTransportHandle function 3174
NET_PRES_SocketInfoGet function 3168
NET_PRES_SocketIsConnected function 3168
NET_PRES_SocketIsNegotiatingEncryption function 3169
NET_PRES_SocketIsOpenModeSupported function 3164
NET_PRES_SocketIsSecure function 3169
NET_PRES_SocketLastError function 3169
NET_PRES_SocketOpen function 3164
NET_PRES_SocketOptionsGet function 3170
NET_PRES_SocketOptionsSet function 3170
NET_PRES_SocketPeek function 3171
NET_PRES_SocketRead function 3171
NET_PRES_SocketReadIsReady function 3172
NET_PRES_SocketRemoteBind function 3172
NET_PRES_SocketSignalHandlerDeregister function 3173
NET_PRES_SocketSignalHandlerRegister function 3173
NET_PRES_SocketWasReset function 3174
NET_PRES_SocketWrite function 3174
NET_PRES_SocketWriteIsReady function 3175
NET_PRES_Status function 3163
NET_PRES_Tasks function 3163
NET_PRES_TRANS_ADDR_T structure 3185
NET_PRES_TRANS_ADDRESS_TYPE enumeration 3186
NET_PRES_TRANS_OPTION_T enumeration 3186
NET_PRES_TransBind type 3186
NET_PRES_TransBool type 3187
NET_PRES_TransClose type 3187
NET_PRES_TransDiscard type 3188
NET_PRES_TransHandlerRegister type 3188
NET_PRES_TransIsPortDefaultSecured type 3194
NET_PRES_TransOpen type 3188
NET_PRES_TransOption type 3189
NET_PRES_TransPeek type 3189
net_pres_transportapi.h 3197
NET_PRES_TransportObject structure 3190
NET_PRES_TransRead type 3191
NET_PRES_TransReady type 3191
NET_PRES_TransSignalHandlerDeregister type 3192
NET_PRES_TransSocketInfoGet type 3192
NET_PRES_TransWrite type 3193
Network Metrics 5982
Networking Presentation Layer Help 3157
New MPLAB Harmony Graphics Library API Functions 2556
New MPLAB Harmony TCP/IP Stack API Functions 5993
NINETYQ10D22 macro 3115
NINETYQ10D6 macro 3115
NO_DATA macro 6058
NO_RECOVERY macro 6058
NORMNEGFRACT16 macro 3116
NORMNEGFRACT32 macro 3116
NORMPOSFRACT16 macro 3116
NORMPOSFRACT32 macro 3116
NULL macro 214
NUMBITSFRACT16 macro 3116
NUMBITSFRACT32 macro 3117
NVM Driver Initialization 1832
NVM Driver Library 1825
NVM Peripheral Library 4415
NVM_BOOT_MEMORY_AREA enumeration 4472
NVM_BOOT_MEMORY_PAGE enumeration 4473
NVM_FLASH_SWAP_LOCK_TYPE enumeration 4474
NVM_MODULE_ID enumeration 4474
NVM_OPERATION_MODE enumeration 4473
NVM_READ_FUNC type 5971
NVM_SECTORERASE_FUNC type 5971
NVM_WRITE_FUNC type 5971

O

OBEX_ALLOCATE_BUFFERS function 18
Object Layer Messaging 2565
Object Layer Rendering 2563
Object Rendering and Style Schemes 2572
Obtaining an Audio Stream 7011
Obtaining Audio v1.0 Device Audio Stream Details 7008
Obtaining System Version Information 5593
OC_16BIT_TIMERS enumeration 4503
OC_ALT_TIMERS enumeration 4505
OC_BUFFER_SIZE enumeration 4503
OC_COMPARE_MODES enumeration 4503
OC_FAULTS enumeration 4505
OC_MODULE_ID enumeration 4505
OGF_CTRL_BASEBAND macro 674
OGF_INFORMATION macro 674
OGF_LE macro 674
OGF_LINK_CONTROL macro 674
OGF_LINK_POLICY macro 674
OGF_STATUS macro 675
OGF_TESTING macro 675
OGF_VENDOR macro 675
OGG_ID_SPEEX macro 1319
One Shot Callback 5940
ONEEIGHTYQ10D22 macro 3117
ONEEIGHTYQ10D6 macro 3117
OOB_DATA_HASH_LENGTH macro 899
OOB_DATA_RANDOMIZER_LENGTH macro 899
Opening a Driver 1342
Opening a File 5765
Opening the CDC Host Client Driver 7106
Opening the Driver 1968, 2112, 2131, 2194
Opening the USART Driver 2336
Operating as a Master 4621
Operating as a Master Receiver 4247
Operating as a Master Transmitter 4243
Operating as a Slave 4622
Operating as a Slave Receiver 4241
Operating as a Slave Transmitter 4242
Operating/Addressing Mode Management 3740
Operations 1675
Optimizing the Bootloader 1183
Optional Interfaces 2047
Opus Decoder Library 1291
opus.h 1306
OPUS_ARG_NONNULL function 1294, 1295
OPUS_Cleanup function 1299
OPUS_Decoder function 1299
opus_decoder_create function 1295
opus_decoder_destroy function 1295
opus_decoder_get_size function 1296
OPUS_DiskRead function 1300
opus_encoder_create function 1297
opus_encoder_destroy function 1297
opus_encoder_get_size function 1298
OPUS_ERROR_MSG enumeration 1303
OPUS_GetChannels function 1300
OPUS_GetSamplingRate function 1301
OPUS_H macro 1302
OPUS_Initialize function 1301
OPUS_INPUT_BUFFER_SIZE macro 1305
OPUS_MAX_FRAME_SIZE macro 1306
OPUS_OUTPUT_BUFFER_SIZE macro 1306
opus_support.h 1307
opusDecDcpt structure 1303
OpusDecoder type 1302
OpusEncoder type 1302
ORANGE macro 2718
OSAL Library Help 3200
OSAL Operation 3205
osal.h 3221
OSAL_CRIT_Enter function 3215
OSAL_CRIT_Leave function 3215
OSAL_CRIT_TYPE enumeration 3220
OSAL_CRIT_TYPE_HIGH enumeration member 3220
OSAL_CRIT_TYPE_LOW enumeration member 3220
OSAL_Free function 3216
OSAL_Initialize function 3218
OSAL_Malloc function 3217
OSAL_MUTEX_Create function 3212
OSAL_MUTEX_DECLARE macro 3220
OSAL_MUTEX_Delete function 3213
OSAL_MUTEX_Lock function 3213
OSAL_MUTEX_Unlock function 3214
OSAL_Name function 3218
OSAL_RESULT enumeration 3219
OSAL_RESULT_FALSE enumeration member 3219
OSAL_RESULT_NOT_IMPLEMENTED enumeration member 3219
OSAL_RESULT_TRUE enumeration member 3219
OSAL_SEM_Create function 3208
OSAL_SEM_DECLARE macro 3220
OSAL_SEM_Delete function 3208
OSAL_SEM_GetCount function 3209
OSAL_SEM_Pend function 3210
OSAL_SEM_Post function 3211
OSAL_SEM_PostISR function 3211
OSAL_SEM_TYPE enumeration 3219
OSAL_SEM_TYPE_BINARY enumeration member 3219
OSAL_SEM_TYPE_COUNTING enumeration member 3219
OSC_CLOCK_ID enumeration 4611
OSC_CLOCK_SLEW_TYPE enumeration 4611
OSC_FRC_DIV enumeration 4605
OSC_MODULE_ID enumeration 4606
OSC_OPERATION_ON_WAIT enumeration 4606
OSC_PB_CLOCK_DIV_TYPE macro 4604
OSC_PERIPHERAL_BUS enumeration 4607
OSC_PLL_SELECT enumeration 4607
OSC_REF_DIVISOR_TYPE macro 4604
OSC_REFERENCE enumeration 4608
OSC_REFERENCE_MAX_DIV macro 4605
OSC_SLEEP_TO_STARTUP_CLK_TYPE enumeration 4612
OSC_SYS_TYPE enumeration 4608
OSC_SYSPLL_FREQ_RANGE enumeration 4609
OSC_SYSPLL_IN_CLK_SOURCE enumeration 4609
OSC_SYSPLL_MULTIPLIER_TYPE macro 4605

OSC_SYSPLL_OUT_DIV enumeration 4610
OSC_USBCLOCK_SOURCE enumeration 4610
Oscillator Peripheral Library 4507
Oscillator Selection and Switching 4510
Oscillator Tuning 5612
Other Features 4259, 4625, 5033, 5249, 5292
Other Functionality 3315, 3439, 4917
OTM2201A_TASK enumeration 1720
OUT_BUF_SIZE macro 833
Output Compare Driver Library 1864
Output Compare Peripheral Library 4479
OVM7690 Camera Driver Library 1365

P

PACK_CONFIG_REQUEST macro 42
PACK_CONFIG_RESPONSE macro 42
PACK_CONN_REQUEST macro 42
PACK_CONN_RESPONSE macro 42
PACK_DCONN_REQUEST macro 43
PACK_DCONN_RESPONSE macro 43
PACK_INFO_REQUEST macro 43
PACK_INFO_RESPONSE macro 43
packet.h 1141
Palette and Gamma Correction Control 4139
Panel Object 2792
Parallel Master Port (PMP) Driver Library 1867
PARM_EQUAL_FILTER structure 3085
PARM_EQUAL_FILTER_16 structure 3086
PARM_EQUAL_FILTER_32 structure 3086
PARM_FILTER_GAIN structure 3087
PASSKEY_CLEARED enumeration member 903
PASSKEY_DIGIT_ENTERED enumeration member 903
PASSKEY_DIGIT_ERASED enumeration member 903
PASSKEY_ENTRY_COMPLETED enumeration member 903
PASSKEY_ENTRY_STYARTED enumeration member 903
patch.h 1141
PBAP_ALLOCATE_BUFFERS function 18
PCACHE_MODULE_ID enumeration 4755
pcmd_disconnection_res structure 778
Periodic Callback 5938
Peripheral Libraries Help 3222
Peripheral Library Overview 3222
Peripheral Library Porting Example 3225
Peripheral Pin Select 5891
PERU macro 2718
pf_hci_sleep_callback type 690
pf_hci_wakeup_callback type 691
pf_l2cap_cmd_callback type 778
pf_l2cap_receive_callback type 691
PGV Error Handling 4973
PIC32MX USB Driver 2210
PIC32MZ USB Driver 2268
Picture Control Object 2795
pid } enumeration 6993
pid enumeration 6992
Pin Control 5889
PIO Mode 5112
PIP_BUFFER macro 1697
plib_adc.h 3299
PLIB_ADC_CalibrationDisable function 3257
PLIB_ADC_CalibrationEnable function 3257
PLIB_ADC_ConversionClockGet function 3264
PLIB_ADC_ConversionClockSet function 3263
PLIB_ADC_ConversionClockSourceSelect function 3265
PLIB_ADC_ConversionHasCompleted function 3261
PLIB_ADC_ConversionStart function 3261
PLIB_ADC_ConversionStopSequenceDisable function 3263
PLIB_ADC_ConversionStopSequenceEnable function 3262
PLIB_ADC_ConversionTriggerSourceSelect function 3262
PLIB_ADC_Disable function 3254
PLIB_ADC_Enable function 3254
PLIB_ADC_ExistsCalibrationControl function 3275
PLIB_ADC_ExistsConversionClock function 3276
PLIB_ADC_ExistsConversionClockSource function 3277
PLIB_ADC_ExistsConversionControl function 3277
PLIB_ADC_ExistsConversionStatus function 3278
PLIB_ADC_ExistsConversionStopSequenceControl function 3278
PLIB_ADC_ExistsConversionTriggerSource function 3279
PLIB_ADC_ExistsEnableControl function 3279
PLIB_ADC_ExistsMuxChannel0NegativeInput function 3280
PLIB_ADC_ExistsMuxChannel0PositiveInput function 3280
PLIB_ADC_ExistsMuxInputScanControl function 3281
PLIB_ADC_ExistsMuxInputScanSelect function 3281
PLIB_ADC_ExistsMuxInputScanSelectExtended function 3288
PLIB_ADC_ExistsResultBufferFillStatus function 3282
PLIB_ADC_ExistsResultBufferMode function 3282
PLIB_ADC_ExistsResultFormat function 3283
PLIB_ADC_ExistsResultGetByIndex function 3284
PLIB_ADC_ExistsSamplesPerInterruptSelect function 3284
PLIB_ADC_ExistsSamplingAcquisitionTime function 3285
PLIB_ADC_ExistsSamplingAutoStart function 3285
PLIB_ADC_ExistsSamplingControl function 3286
PLIB_ADC_ExistsSamplingModeControl function 3286
PLIB_ADC_ExistsSamplingStatus function 3287
PLIB_ADC_ExistsStopInIdleControl function 3287
PLIB_ADC_ExistsVoltageReference function 3288
PLIB_ADC_InputScanMaskAdd function 3258
PLIB_ADC_InputScanMaskAddExtended function 3259
PLIB_ADC_InputScanMaskRemove function 3258
PLIB_ADC_InputScanMaskRemoveExtended function 3260
PLIB_ADC_MuxAInputScanDisable function 3273
PLIB_ADC_MuxAInputScanEnable function 3274
PLIB_ADC_MuxChannel0InputNegativeSelect function 3274
PLIB_ADC_MuxChannel0InputPositiveSelect function 3275
PLIB_ADC_ResultBufferModeSelect function 3270
PLIB_ADC_ResultBufferStatusGet function 3271
PLIB_ADC_ResultFormatSelect function 3272
PLIB_ADC_ResultGetByIndex function 3272
PLIB_ADC_SampleAcquisitionTimeSet function 3270
PLIB_ADC_SampleAutoStartDisable function 3265
PLIB_ADC_SampleAutoStartEnable function 3266
PLIB_ADC_SamplesPerInterruptSelect function 3267
PLIB_ADC_SamplingIsActive function 3267
PLIB_ADC_SamplingModeSelect function 3268
PLIB_ADC_SamplingStart function 3269
PLIB_ADC_SamplingStop function 3269

PLIB_ADC_StopInIdleDisable function 3255
PLIB_ADC_StopInIdleEnable function 3255
PLIB_ADC_VoltageReferenceSelect function 3256
plib_adchs.h 3431
PLIB_ADCHS_AnalogInputDataIsReady function 3335
PLIB_ADCHS_AnalogInputDataReadyInterruptDisable function 3336
PLIB_ADCHS_AnalogInputDataReadyInterruptEnable function 3337
PLIB_ADCHS_AnalogInputEarlyInterruptDisable function 3337
PLIB_ADCHS_AnalogInputEarlyInterruptEnable function 3338
PLIB_ADCHS_AnalogInputEarlyInterruptIsReady function 3339
PLIB_ADCHS_AnalogInputEdgeTriggerSet function 3347
PLIB_ADCHS_AnalogInputIsAvailable function 3340
PLIB_ADCHS_AnalogInputLevelTriggerSet function 3348
PLIB_ADCHS_AnalogInputModeGet function 3340
PLIB_ADCHS_AnalogInputModeSelect function 3344
PLIB_ADCHS_AnalogInputResultGet function 3341
PLIB_ADCHS_AnalogInputScanIsComplete function 3342
PLIB_ADCHS_AnalogInputScanIsSelected function 3342
PLIB_ADCHS_AnalogInputScanRemove function 3343
PLIB_ADCHS_AnalogInputScanSelect function 3345
PLIB_ADCHS_AnalogInputScanSetup function 3348
PLIB_ADCHS_AnalogInputTriggerSourceSelect function 3349
PLIB_ADCHS_ChannelAnalogFeatureDisable function 3374
PLIB_ADCHS_ChannelAnalogFeatureEnable function 3375
PLIB_ADCHS_ChannelConfigurationGet function 3376
PLIB_ADCHS_ChannelConfigurationSet function 3377
PLIB_ADCHS_ChannelDigitalFeatureDisable function 3377
PLIB_ADCHS_ChannelDigitalFeatureEnable function 3378
PLIB_ADCHS_ChannelInputSelect function 3383
PLIB_ADCHS_ChannelIsReady function 3379
PLIB_ADCHS_ChannelIsReadyInterruptDisable function 3380
PLIB_ADCHS_ChannelIsReadyInterruptEnable function 3380
PLIB_ADCHS_ChannelSetup function 3381
PLIB_ADCHS_ChannelTriggerSampleSelect function 3346
PLIB_ADCHS_ControlRegistersCanBeUpdated function 3323
PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptDisable function 3324
PLIB_ADCHS_ControlRegistersCanBeUpdatedInterruptEnable function 3324
PLIB_ADCHS_CVDDisable function 3332
PLIB_ADCHS_CVDEnable function 3333
PLIB_ADCHS_CVDResultGet function 3333
PLIB_ADCHS_CVDSetup function 3334
PLIB_ADCHS_DigitalComparatorAnalogInputGet function 3350
PLIB_ADCHS_DigitalComparatorAnalogInputSelect function 3351
PLIB_ADCHS_DigitalComparatorDisable function 3352
PLIB_ADCHS_DigitalComparatorEnable function 3352
PLIB_ADCHS_DigitalComparatorEventHasOccurred function 3353
PLIB_ADCHS_DigitalComparatorInterruptDisable function 3354
PLIB_ADCHS_DigitalComparatorInterruptEnable function 3355
PLIB_ADCHS_DigitalComparatorLimitSet function 3355
PLIB_ADCHS_DigitalComparatorSetup function 3356
PLIB_ADCHS_DigitalFilterAveragingModeSampleCountSelect function 3358
PLIB_ADCHS_DigitalFilterAveragingModeSetup function 3358
PLIB_ADCHS_DigitalFilterDataGet function 3359
PLIB_ADCHS_DigitalFilterDataIsReady function 3360
PLIB_ADCHS_DigitalFilterDataReadyInterruptDisable function 3361
PLIB_ADCHS_DigitalFilterDataReadyInterruptEnable function 3362
PLIB_ADCHS_DigitalFilterDisable function 3362
PLIB_ADCHS_DigitalFilterEnable function 3363
PLIB_ADCHS_DigitalFilterOversamplingModeRatioSelect function 3364
PLIB_ADCHS_DigitalFilterOversamplingModeSetup function 3364
PLIB_ADCHS_Disable function 3320
PLIB_ADCHS_DMABuffer_A_InterruptDisable function 3366
PLIB_ADCHS_DMABuffer_A_InterruptEnable function 3366
PLIB_ADCHS_DMABuffer_A_IsFull function 3367
PLIB_ADCHS_DMABuffer_B_InterruptDisable function 3368
PLIB_ADCHS_DMABuffer_B_InterruptEnable function 3368
PLIB_ADCHS_DMABuffer_B_IsFull function 3369
PLIB_ADCHS_DMADisable function 3370
PLIB_ADCHS_DMAEnable function 3371
PLIB_ADCHS_DMAOverflowErrorHasOccurred function 3371
PLIB_ADCHS_DMASetup function 3372
PLIB_ADCHS_DMASourceRemove function 3373
PLIB_ADCHS_DMASourceSelect function 3374
PLIB_ADCHS_EarlyInterruptDisable function 3391
PLIB_ADCHS_EarlyInterruptEnable function 3391
PLIB_ADCHS_Enable function 3321
PLIB_ADCHS_ExistsAnalogInputCheck function 3399
PLIB_ADCHS_ExistsAnalogInputModeControl function 3399
PLIB_ADCHS_ExistsAnalogInputScan function 3400
PLIB_ADCHS_ExistsChannelAnalogControl function 3400
PLIB_ADCHS_ExistsChannelConfiguration function 3401
PLIB_ADCHS_ExistsChannelDigitalControl function 3402
PLIB_ADCHS_ExistsChannelInputSelectControl function 3402
PLIB_ADCHS_ExistsConfiguration function 3403
PLIB_ADCHS_ExistsConversionResults function 3403
PLIB_ADCHS_ExistsCVD function 3404
PLIB_ADCHS_ExistsDigitalComparator function 3404
PLIB_ADCHS_ExistsDigitalFilter function 3405
PLIB_ADCHS_ExistsDMA function 3406
PLIB_ADCHS_ExistsEarlyInterruptControl function 3407
PLIB_ADCHS_ExistsEnableControl function 3407
PLIB_ADCHS_ExistsExternalConversionRequestControl function 3408
PLIB_ADCHS_ExistsFIFO function 3408
PLIB_ADCHS_ExistsManualControl function 3409
PLIB_ADCHS_ExistsTriggerControl function 3410
PLIB_ADCHS_ExistsTriggerSampleControl function 3410
PLIB_ADCHS_ExistsTurboMode function 3411
PLIB_ADCHS_ExistsUpdateReadyControl function 3411
PLIB_ADCHS_ExistsVREFControl function 3412
PLIB_ADCHS_ExternalConversionRequestDisable function 3325
PLIB_ADCHS_ExternalConversionRequestEnable function 3325
PLIB_ADCHS_FIFODataCountGet function 3384
PLIB_ADCHS_FIFODataIsAvailable function 3384
PLIB_ADCHS_FIFODataIsNegative function 3385
PLIB_ADCHS_FIFODataReadyInterruptDisable function 3386
PLIB_ADCHS_FIFODataReadyInterruptEnable function 3386
PLIB_ADCHS_FIFODisable function 3387
PLIB_ADCHS_FIFOEnable function 3387
PLIB_ADCHS_FIFOErrorHasOccurred function 3388
PLIB_ADCHS_FIFORead function 3389
PLIB_ADCHS_FIFOSourceGet function 3389
PLIB_ADCHS_FIFOSourceSelect function 3390
PLIB_ADCHS_GlobalLevelSoftwareTriggerDisable function 3326
PLIB_ADCHS_GlobalLevelSoftwareTriggerEnable function 3326

PLIB_ADCHS_GlobalSoftwareTriggerEnable function 3327
PLIB_ADCHS_ScanCompleteInterruptDisable function 3328
PLIB_ADCHS_ScanCompleteInterruptEnable function 3328
PLIB_ADCHS_Setup function 3321
PLIB_ADCHS_SoftwareConversionInputSelect function 3329
PLIB_ADCHS_SoftwareConversionStart function 3330
PLIB_ADCHS_SoftwareSamplingStart function 3330
PLIB_ADCHS_SoftwareSamplingStop function 3331
PLIB_ADCHS_TriggerSuspendDisable function 3331
PLIB_ADCHS_TriggerSuspendEnable function 3332
PLIB_ADCHS_TurboModeChannelSelect function 3392
PLIB_ADCHS_TurboModeDisable function 3393
PLIB_ADCHS_TurboModeEnable function 3394
PLIB_ADCHS_TurboModeErrorHasOccurred function 3394
PLIB_ADCHS_VREFFaultHasOccurred function 3395
PLIB_ADCHS_VREFFaultInterruptDisable function 3396
PLIB_ADCHS_VREFFaultInterruptEnable function 3396
PLIB_ADCHS_VREFIsReady function 3397
PLIB_ADCHS_VREFReadyInterruptDisable function 3397
PLIB_ADCHS_VREFReadyInterruptEnable function 3398
plib_adcp.h 3480
PLIB_ADCP_AlternateInputSelect function 3451
PLIB_ADCP_CalibrationStart function 3445
PLIB_ADCP_ChannelScanConfigure function 3454
PLIB_ADCP_Class12TriggerConfigure function 3457
PLIB_ADCP_Configure function 3443
PLIB_ADCP_DefaultInputSelect function 3452
PLIB_ADCP_DigCmpAIdGet function 3462
PLIB_ADCP_DigCmpConfig function 3463
PLIB_ADCP_DigCmpDisable function 3461
PLIB_ADCP_DigCmpEnable function 3461
PLIB_ADCP_Disable function 3445
PLIB_ADCP_Enable function 3444
PLIB_ADCP_ExistsCalibration function 3448
PLIB_ADCP_ExistsChannelScan function 3455
PLIB_ADCP_ExistsConfiguration function 3449
PLIB_ADCP_ExistsConversionResults function 3460
PLIB_ADCP_ExistsDigCmp function 3464
PLIB_ADCP_ExistsEnableControl function 3448
PLIB_ADCP_ExistsInputSelect function 3452
PLIB_ADCP_ExistsLowPowerControl function 3449
PLIB_ADCP_ExistsModeSelect function 3453
PLIB_ADCP_ExistsOsampDigFilter function 3467
PLIB_ADCP_ExistsReadyStatus function 3451
PLIB_ADCP_ExistsTriggering function 3458
PLIB_ADCP_GlobalSoftwareTrigger function 3456
PLIB_ADCP_IndividualTrigger function 3456
PLIB_ADCP_LowPowerStateEnter function 3446
PLIB_ADCP_LowPowerStateExit function 3446
PLIB_ADCP_LowPowerStateGet function 3447
PLIB_ADCP_ModuleIsReady function 3450
PLIB_ADCP_OsampDigFilterConfig function 3468
PLIB_ADCP_OsampDigFilterDataGet function 3466
PLIB_ADCP_OsampDigFilterDataRdy function 3466
PLIB_ADCP_OsampDigFilterDisable function 3465
PLIB_ADCP_OsampDigFilterEnable function 3465
PLIB_ADCP_ResultGet function 3459
PLIB_ADCP_ResultReady function 3458
PLIB_ADCP_ResultReadyGrpIntConfigure function 3459
PLIB_ADCP_SHModeSelect function 3453
plib_bmx.h 3513
PLIB_BMX_ARB_MODE enumeration 3511
PLIB_BMX_ArbitrationModeGet function 3495
PLIB_BMX_ArbitrationModeSet function 3496
PLIB_BMX_BusExceptionDataDisable function 3488
PLIB_BMX_BusExceptionDataEnable function 3489
PLIB_BMX_BusExceptionDMADisable function 3489
PLIB_BMX_BusExceptionDMAEnable function 3490
PLIB_BMX_BusExceptionICDDisable function 3490
PLIB_BMX_BusExceptionICDEnable function 3491
PLIB_BMX_BusExceptionInstructionDisable function 3492
PLIB_BMX_BusExceptionInstructionEnable function 3492
PLIB_BMX_BusExceptionIXIDisable function 3493
PLIB_BMX_BusExceptionIXIEnable function 3493
PLIB_BMX_DATA_RAM_WAIT_STATES enumeration 3512
PLIB_BMX_DataRAMKernelProgramOffsetGet function 3496
PLIB_BMX_DataRAMPartitionSet function 3497
PLIB_BMX_DataRAMSizeGet function 3498
PLIB_BMX_DataRAMUserDataOffsetGet function 3499
PLIB_BMX_DataRAMUserProgramOffsetGet function 3499
PLIB_BMX_DataRamWaitStateGet function 3500
PLIB_BMX_DataRamWaitStateSet function 3500
PLIB_BMX_DRM_BLOCK_SIZE macro 3512
PLIB_BMX_EXCEPTION_SRC enumeration 3512
PLIB_BMX_ExistsArbitrationMode function 3504
PLIB_BMX_ExistsBusExceptionData function 3504
PLIB_BMX_ExistsBusExceptionDMA function 3505
PLIB_BMX_ExistsBusExceptionICD function 3505
PLIB_BMX_ExistsBusExceptionInstruction function 3506
PLIB_BMX_ExistsBusExceptionIXI function 3506
PLIB_BMX_ExistsDataRAMPartition function 3507
PLIB_BMX_ExistsDataRAMSize function 3507
PLIB_BMX_ExistsDataRamWaitState function 3508
PLIB_BMX_ExistsProgramFlashBootSize function 3509
PLIB_BMX_ExistsProgramFlashMemoryCacheDma function 3509
PLIB_BMX_ExistsProgramFlashMemorySize function 3510
PLIB_BMX_ExistsProgramFlashPartition function 3510
PLIB_BMX_PFM_BLOCK_SIZE macro 3513
PLIB_BMX_ProgramFlashBootSizeGet function 3501
PLIB_BMX_ProgramFlashMemoryCacheDmaDisable function 3494
PLIB_BMX_ProgramFlashMemoryCacheDmaEnable function 3494
PLIB_BMX_ProgramFlashMemorySizeGet function 3501
PLIB_BMX_ProgramFlashPartitionGet function 3502
PLIB_BMX_ProgramFlashPartitionSet function 3503
plib_can.h 3610
PLIB_CAN_AllChannelEventsGet function 3552
PLIB_CAN_AllChannelOverflowStatusGet function 3553
PLIB_CAN_BaudRateGet function 3541
PLIB_CAN_BaudRatePrescaleSetup function 3542
PLIB_CAN_BitSamplePhaseSet function 3544
PLIB_CAN_BusActivityWakeupDisable function 3528
PLIB_CAN_BusActivityWakeupEnable function 3529
PLIB_CAN_BusLine3TimesSamplingDisable function 3540
PLIB_CAN_BusLine3TimesSamplingEnable function 3540
PLIB_CAN_ChannelEventClear function 3554
PLIB_CAN_ChannelEventDisable function 3549

PLIB_CAN_ChannelEventEnable function 3555
PLIB_CAN_ChannelEventGet function 3555
PLIB_CAN_ChannelForReceiveSet function 3545
PLIB_CAN_ChannelForTransmitSet function 3546
PLIB_CAN_ChannelReset function 3547
PLIB_CAN_ChannelResetIsComplete function 3537
PLIB_CAN_ChannelUpdate function 3548
PLIB_CAN_DeviceNetConfigure function 3538
PLIB_CAN_Disable function 3530
PLIB_CAN_Enable function 3530
PLIB_CAN_ErrorStateGet function 3570
PLIB_CAN_ExistsActiveStatus function 3572
PLIB_CAN_ExistsAllChannelEvents function 3573
PLIB_CAN_ExistsAllChannelOverflow function 3573
PLIB_CAN_ExistsBaudRateGet function 3592
PLIB_CAN_ExistsBaudRatePrescaleSetup function 3593
PLIB_CAN_ExistsBitSamplePhaseSet function 3593
PLIB_CAN_ExistsBusActivityWakeup function 3574
PLIB_CAN_ExistsBusLine3TimesSampling function 3574
PLIB_CAN_ExistsChannelEvent function 3575
PLIB_CAN_ExistsChannelEventEnable function 3576
PLIB_CAN_ExistsChannelForReceiveSet function 3576
PLIB_CAN_ExistsChannelForTransmitSet function 3577
PLIB_CAN_ExistsChannelReset function 3577
PLIB_CAN_ExistsChannelUpdate function 3578
PLIB_CAN_ExistsDeviceNet function 3578
PLIB_CAN_ExistsEnableControl function 3579
PLIB_CAN_ExistsErrorState function 3579
PLIB_CAN_ExistsFilterConfigure function 3580
PLIB_CAN_ExistsFilterEnable function 3580
PLIB_CAN_ExistsFilterMaskConfigure function 3581
PLIB_CAN_ExistsFilterToChannelLink function 3582
PLIB_CAN_ExistsLatestFilterMatchGet function 3582
PLIB_CAN_ExistsMemoryBufferAssign function 3583
PLIB_CAN_ExistsModuleEventClear function 3583
PLIB_CAN_ExistsModuleEventEnable function 3584
PLIB_CAN_ExistsModuleInfo function 3584
PLIB_CAN_ExistsOperationModeRead function 3585
PLIB_CAN_ExistsOperationModeWrite function 3585
PLIB_CAN_ExistsPendingEventsGet function 3586
PLIB_CAN_ExistsPendingTransmissionsAbort function 3586
PLIB_CAN_ExistsPrecalculatedBitRateSetup function 3594
PLIB_CAN_ExistsReceivedMessageGet function 3587
PLIB_CAN_ExistsReceiveErrorCount function 3587
PLIB_CAN_ExistsStopInIdle function 3588
PLIB_CAN_ExistsTimeStampEnable function 3589
PLIB_CAN_ExistsTimeStampPrescaler function 3535
PLIB_CAN_ExistsTimeStampValue function 3589
PLIB_CAN_ExistsTransmissionIsAborted function 3590
PLIB_CAN_ExistsTransmitBufferGet function 3590
PLIB_CAN_ExistsTransmitChannelFlush function 3591
PLIB_CAN_ExistsTransmitChannelStatus function 3591
PLIB_CAN_ExistsTransmitErrorCountGet function 3592
PLIB_CAN_FilterConfigure function 3563
PLIB_CAN_FilterDisable function 3564
PLIB_CAN_FilterEnable function 3565
PLIB_CAN_FilterIsDisabled function 3566
PLIB_CAN_FilterMaskConfigure function 3566
PLIB_CAN_FilterToChannelLink function 3567
plib_can_help.h 3613
PLIB_CAN_IsActive function 3531
PLIB_CAN_LatestFilterMatchGet function 3568
PLIB_CAN_MemoryBufferAssign function 3536
PLIB_CAN_ModuleEventClear function 3549
PLIB_CAN_ModuleEventDisable function 3550
PLIB_CAN_ModuleEventEnable function 3551
PLIB_CAN_ModuleEventGet function 3551
PLIB_CAN_OperationModeGet function 3532
PLIB_CAN_OperationModeSelect function 3532
PLIB_CAN_PendingEventsGet function 3557
PLIB_CAN_PendingTransmissionsAbort function 3557
PLIB_CAN_PrecalculatedBitRateSetup function 3543
PLIB_CAN_ReceivedMessageGet function 3562
PLIB_CAN_ReceiveErrorCountGet function 3569
PLIB_CAN_StopInIdleDisable function 3533
PLIB_CAN_StopInIdleEnable function 3533
PLIB_CAN_TimeStampDisable function 3534
PLIB_CAN_TimeStampEnable function 3535
PLIB_CAN_TimeStampPrescalerSet function 3536
PLIB_CAN_TimeStampValueGet function 3539
PLIB_CAN_TimeStampValueSet function 3539
PLIB_CAN_TotalChannelsGet function 3571
PLIB_CAN_TotalFiltersGet function 3571
PLIB_CAN_TotalMasksGet function 3572
PLIB_CAN_TransmissionIsAborted function 3558
PLIB_CAN_TransmitBufferGet function 3559
PLIB_CAN_TransmitChannelFlush function 3560
PLIB_CAN_TransmitChannelStatusGet function 3560
PLIB_CAN_TransmitErrorCountGet function 3561
plib_cmp.h 3651
PLIB_CMP_CVREF_BandGapReferenceSourceSelect function 3619
PLIB_CMP_CVREF_Disable function 3625
PLIB_CMP_CVREF_Enable function 3619
PLIB_CMP_CVREF_OutputDisable function 3620
PLIB_CMP_CVREF_OutputEnable function 3620
PLIB_CMP_CVREF_ReferenceVoltageSelect function 3621
PLIB_CMP_CVREF_SourceNegativeInputSelect function 3622
PLIB_CMP_CVREF_SourceVoltageSelect function 3622
PLIB_CMP_CVREF_ValueSelect function 3623
PLIB_CMP_CVREF_WideRangeDisable function 3623
PLIB_CMP_CVREF_WideRangeEnable function 3624
PLIB_CMP_CVREF_WideRangeIsEnabled function 3624
PLIB_CMP_Disable function 3626
PLIB_CMP_Enable function 3626
PLIB_CMP_ExistsCVREFBGRefVoltageRangeSelect function 3634
PLIB_CMP_ExistsCVREFEnableControl function 3635
PLIB_CMP_ExistsCVREFOutputEnableControl function 3635
PLIB_CMP_ExistsCVREFRefVoltageRangeSelect function 3636
PLIB_CMP_ExistsCVREFValueSelect function 3636
PLIB_CMP_ExistsCVREFVoltageRangeSelect function 3637
PLIB_CMP_ExistsCVREFWideRangeControl function 3638
PLIB_CMP_ExistsEnableControl function 3638
PLIB_CMP_ExistsInterruptEventSelect function 3639
PLIB_CMP_ExistsInvertingInputSelect function 3639
PLIB_CMP_ExistsInvertOutputControl function 3640
PLIB_CMP_ExistsNonInvertingInputSelect function 3640

PLIB_CMP_ExistsOutputEnableControl function 3641
PLIB_CMP_ExistsOutputLevelControl function 3641
PLIB_CMP_ExistsOutputStatusGet function 3642
PLIB_CMP_ExistsStopInIdle function 3642
PLIB_CMP_InterruptEventSelect function 3627
PLIB_CMP_InvertingInputChannelSelect function 3627
PLIB_CMP_NonInvertingInputChannelSelect function 3628
PLIB_CMP_OutputDisable function 3629
PLIB_CMP_OutputEnable function 3629
PLIB_CMP_OutputInvertDisable function 3631
PLIB_CMP_OutputInvertEnable function 3631
PLIB_CMP_OutputLogicHigh function 3632
PLIB_CMP_OutputLogicLow function 3632
PLIB_CMP_OutputStatusGet function 3630
PLIB_CMP_StopInIdleModeDisable function 3633
PLIB_CMP_StopInIdleModeEnable function 3634
plib_ctmu.h 3683
PLIB_CTMU_AutomaticADCTriggerDisable function 3667
PLIB_CTMU_AutomaticADCTriggerEnable function 3667
PLIB_CTMU_CurrentDischargeDisable function 3668
PLIB_CTMU_CurrentDischargeEnable function 3668
PLIB_CTMU_CurrentRangeSet function 3664
PLIB_CTMU_CurrentTrimSet function 3669
PLIB_CTMU_Disable function 3664
PLIB_CTMU_EdgeDisable function 3671
PLIB_CTMU_EdgeEnable function 3671
PLIB_CTMU_EdgeIsSet function 3673
PLIB_CTMU_EdgePolaritySet function 3674
PLIB_CTMU_EdgeSensitivitySet function 3674
PLIB_CTMU_EdgeSequenceDisable function 3672
PLIB_CTMU_EdgeSequenceEnable function 3672
PLIB_CTMU_EdgeSet function 3675
PLIB_CTMU_EdgeTriggerSourceSelect function 3675
PLIB_CTMU_Enable function 3665
PLIB_CTMU_ExistsAutomaticADCTrigger function 3676
PLIB_CTMU_ExistsCurrentDischarge function 3677
PLIB_CTMU_ExistsCurrentRange function 3677
PLIB_CTMU_ExistsCurrentTrim function 3678
PLIB_CTMU_ExistsEdgeEnable function 3678
PLIB_CTMU_ExistsEdgePolarity function 3679
PLIB_CTMU_ExistsEdgeSensitivity function 3679
PLIB_CTMU_ExistsEdgeSequencing function 3680
PLIB_CTMU_ExistsEdgeStatus function 3680
PLIB_CTMU_ExistsEdgeTriggerSource function 3681
PLIB_CTMU_ExistsModuleEnable function 3682
PLIB_CTMU_ExistsStopInIdle function 3682
PLIB_CTMU_ExistsTimePulseGeneration function 3683
PLIB_CTMU_StopInIdleDisable function 3665
PLIB_CTMU_StopInIdleEnable function 3666
PLIB_CTMU_TimePulseGenerationDisable function 3669
PLIB_CTMU_TimePulseGenerationEnable function 3670
plib_ddr.h 3932
PLIB_DDR_AutoPchrgDisable function 3877
PLIB_DDR_AutoPchrgEnable function 3881
PLIB_DDR_AutoPchrgPowerDownDisable function 3878
PLIB_DDR_AutoPchrgPowerDownEnable function 3879
PLIB_DDR_AutoPowerDownDisable function 3878
PLIB_DDR_AutoPowerDownEnable function 3880
PLIB_DDR_AutoSelfRefreshDisable function 3879
PLIB_DDR_AutoSelfRefreshEnable function 3880
PLIB_DDR_BankAddressSet function 3888
PLIB_DDR_BigEndianSet function 3872
PLIB_DDR_ChipSelectAddressSet function 3888
PLIB_DDR_CmdDataIsComplete function 3884
PLIB_DDR_CmdDataSend function 3884
PLIB_DDR_CmdDataValid function 3885
PLIB_DDR_CmdDataWrite function 3887
PLIB_DDR_ColumnAddressSet function 3889
PLIB_DDR_ControllerEnable function 3886
PLIB_DDR_DataDelaySet function 3892
PLIB_DDR_ExistsAddressMapping function 3920
PLIB_DDR_ExistsArbitrationControl function 3921
PLIB_DDR_ExistsAutoPowerDown function 3921
PLIB_DDR_ExistsAutoPrecharge function 3922
PLIB_DDR_ExistsAutoSelfRefresh function 3922
PLIB_DDR_ExistsDDRCommands function 3923
PLIB_DDR_ExistsDDRControllerConfig function 3924
PLIB_DDR_ExistsODTConfig function 3924
PLIB_DDR_ExistsPHY_DLLCalibration function 3925
PLIB_DDR_ExistsPHY_PadConfig function 3925
PLIB_DDR_ExistsPHY_SCLConfig function 3926
PLIB_DDR_ExistsRefreshConfig function 3927
PLIB_DDR_ExistsTimingDelays function 3927
PLIB_DDR_FullRateSet function 3873
PLIB_DDR_HalfRateSet function 3873
PLIB_DDR_LittleEndianSet function 3874
PLIB_DDR_MaxCmdBrstCntSet function 3886
PLIB_DDR_MaxPendingRefSet function 3874
PLIB_DDR_MinCommand function 3882
PLIB_DDR_MinLimit function 3883
PLIB_DDR_NumHostCmdsSet function 3887
PLIB_DDR_OdtReadDisable function 3875
PLIB_DDR_OdtReadEnable function 3875
PLIB_DDR_OdtReadParamSet function 3890
PLIB_DDR_OdtWriteDisable function 3876
PLIB_DDR_OdtWriteEnable function 3877
PLIB_DDR_OdtWriteParamSet function 3891
PLIB_DDR_PHY_AddCtlDriveStrengthSet function 3918
PLIB_DDR_PHY_DataDriveStrengthSet function 3918
PLIB_DDR_PHY_DDRTypeSet function 3903
PLIB_DDR_PHY_DllMasterDelayStartSet function 3903
PLIB_DDR_PHY_DllRecalibDisable function 3904
PLIB_DDR_PHY_DllRecalibEnable function 3904
PLIB_DDR_PHY_DriveStrengthSet function 3905
PLIB_DDR_PHY_DrvStrgthCal function 3905
PLIB_DDR_PHY_ExternalDllEnable function 3906
PLIB_DDR_PHY_ExtraClockDisable function 3906
PLIB_DDR_PHY_ExtraClockEnable function 3907
PLIB_DDR_PHY_HalfRateSet function 3916
PLIB_DDR_PHY_InternalDllEnable function 3907
PLIB_DDR_PHY_OdtCal function 3908
PLIB_DDR_PHY_OdtCSDisable function 3909
PLIB_DDR_PHY_OdtCSEnable function 3909
PLIB_DDR_PHY_OdtDisable function 3910
PLIB_DDR_PHY_OdtEnable function 3910
PLIB_DDR_PHY_PadReceiveDisable function 3915

PLIB_DDR_PHY_PadReceiveEnable function 3911
PLIB_DDR_PHY_PreambleDlySet function 3911
PLIB_DDR_PHY_ReadCASLatencySet function 3912
PLIB_DDR_PHY_SCLCapClkDelaySet function 3915
PLIB_DDR_PHY_SCLDDRClkDelaySet function 3916
PLIB_DDR_PHY_SCLDelay function 3912
PLIB_DDR_PHY_SCLEnable function 3913
PLIB_DDR_PHY_SCLStart function 3917
PLIB_DDR_PHY_SCLStatus function 3917
PLIB_DDR_PHY_SCLTestBurstModeSet function 3913
PLIB_DDR_PHY_WriteCASLatencySet function 3914
PLIB_DDR_PHY_WriteCmdDelayDisable function 3919
PLIB_DDR_PHY_WriteCmdDelayEnable function 3920
PLIB_DDR_PowerDownDelaySet function 3892
PLIB_DDR_PrechargAllBanksSet function 3894
PLIB_DDR_PrechargeToRASDelaySet function 3896
PLIB_DDR_RASToCASDelaySet function 3897
PLIB_DDR_RASToPrechargeDelaySet function 3898
PLIB_DDR_RASToRASBankDelaySet function 3898
PLIB_DDR_RASToRASDelaySet function 3899
PLIB_DDR_ReadReadDelaySet function 3901
PLIB_DDR_ReadToPrechargeDelaySet function 3900
PLIB_DDR_ReadWriteDelaySet function 3894
PLIB_DDR_RefreshTimingSet function 3895
PLIB_DDR_ReqPeriod function 3883
PLIB_DDR_RowAddressSet function 3890
PLIB_DDR_SelfRefreshDelaySet function 3893
PLIB_DDR_TfawDelaySet function 3881
PLIB_DDR_WriteReadDelaySet function 3902
PLIB_DDR_WriteToPrechargeDelaySet function 3900
PLIB_DDR_WriteWriteDelaySet function 3896
plib_devcon.h 3734
PLIB_DEVCON_2WireJTAGDisableTDO function 3703
PLIB_DEVCON_2WireJTAGEnableTDO function 3703
PLIB_DEVCON_AlternateClockDisable function 3704
PLIB_DEVCON_AlternateClockEnable function 3704
PLIB_DEVCON_AnalogIOChargePumpDisable function 3713
PLIB_DEVCON_AnalogIOChargePumpEnable function 3713
PLIB_DEVCON_BootExtSelect function 3714
PLIB_DEVCON_BootIPFSelect function 3715
PLIB_DEVCON_DeviceIdGet function 3705
PLIB_DEVCON_DeviceRegistersLock function 3706
PLIB_DEVCON_DeviceRegistersUnlock function 3706
PLIB_DEVCON_DeviceVersionGet function 3707
PLIB_DEVCON_ExistsAlternateClock function 3723
PLIB_DEVCON_ExistsAnalogChargePumpControl function 3714
PLIB_DEVCON_ExistsBootSelection function 3729
PLIB_DEVCON_ExistsDeviceRegsLockUnlock function 3723
PLIB_DEVCON_ExistsDeviceVerAndId function 3724
PLIB_DEVCON_ExistsECCModes function 3724
PLIB_DEVCON_ExistsHSUARTControl function 3729
PLIB_DEVCON_ExistsIgnoreDebugFreeze function 3725
PLIB_DEVCON_ExistsJTAGEnable function 3725
PLIB_DEVCON_ExistsJTAGUsesTDO function 3726
PLIB_DEVCON_ExistsMPLL function 3728
PLIB_DEVCON_ExistsOTPConfigLockUnlock function 3730
PLIB_DEVCON_ExistsSystemLockUnlock function 3726
PLIB_DEVCON_ExistsTraceOutput function 3727
PLIB_DEVCON_ExistsUSB_PHY_SleepModeSet function 3728
PLIB_DEVCON_FlashErrCorrectionModeSet function 3707
PLIB_DEVCON_HSUARTDisable function 3715
PLIB_DEVCON_HSUARTEnable function 3716
PLIB_DEVCON_IgnoreDebugFreezeDisable function 3708
PLIB_DEVCON_IgnoreDebugFreezeEnable function 3708
PLIB_DEVCON_JTAGPortDisable function 3709
PLIB_DEVCON_JTAGPortEnable function 3710
PLIB_DEVCON_MPLLDisable function 3717
PLIB_DEVCON_MPLLEnable function 3718
PLIB_DEVCON_MPLLInputDivSet function 3718
PLIB_DEVCON_MPLLIsReady function 3719
PLIB_DEVCON_MPLLMultiplierSet function 3719
PLIB_DEVCON_MPLLODiv1Set function 3720
PLIB_DEVCON_MPLLODiv2Set function 3720
PLIB_DEVCON_MPLLVrefSet function 3721
PLIB_DEVCON_MPLLVregDisable function 3721
PLIB_DEVCON_MPLLVregEnable function 3722
PLIB_DEVCON_MPLLVregIsReady function 3722
PLIB_DEVCON_OTPConfigLock function 3716
PLIB_DEVCON_OTPConfigUnlock function 3717
PLIB_DEVCON_SystemLock function 3710
PLIB_DEVCON_SystemUnlock function 3711
PLIB_DEVCON_TraceOutputDisable function 3711
PLIB_DEVCON_TraceOutputEnable function 3711
PLIB_DEVCON_USBPHYSleepModeSet function 3712
plib_dma.h 3862
PLIB_DMA_AbortTransferSet function 3756
PLIB_DMA_BusyActiveReset function 3752
PLIB_DMA_BusyActiveSet function 3752
PLIB_DMA_ChannelBitsGet function 3823
PLIB_DMA_ChannelPriorityGet function 3789
PLIB_DMA_ChannelPrioritySelect function 3788
PLIB_DMA_ChannelXAbortIRQSet function 3757
PLIB_DMA_ChannelXAddressModeGet function 3765
PLIB_DMA_ChannelXAddressModeSelect function 3766
PLIB_DMA_ChannelXAutoDisable function 3789
PLIB_DMA_ChannelXAutoEnable function 3790
PLIB_DMA_ChannelXAutoIsEnabled function 3817
PLIB_DMA_ChannelXBufferedDataIsWritten function 3818
PLIB_DMA_ChannelXBusyActiveSet function 3790
PLIB_DMA_ChannelXBusyInActiveSet function 3791
PLIB_DMA_ChannelXBusyIsBusy function 3818
PLIB_DMA_ChannelXCellProgressPointerGet function 3778
PLIB_DMA_ChannelXCellSizeGet function 3779
PLIB_DMA_ChannelXCellSizeSet function 3780
PLIB_DMA_ChannelXChainDisable function 3791
PLIB_DMA_ChannelXChainEnable function 3792
PLIB_DMA_ChannelXChainIsEnabled function 3819
PLIB_DMA_ChannelXChainToHigher function 3793
PLIB_DMA_ChannelXChainToLower function 3793
PLIB_DMA_ChannelXCollisionStatus function 3820
PLIB_DMA_ChannelXDataSizeGet function 3780
PLIB_DMA_ChannelXDataSizeSelect function 3781
PLIB_DMA_ChannelXDestinationAddressModeGet function 3766
PLIB_DMA_ChannelXDestinationAddressModeSelect function 3767
PLIB_DMA_ChannelXDestinationPointerGet function 3782
PLIB_DMA_ChannelXDestinationSizeGet function 3782

PLIB_DMA_ChannelXDestinationSizeSet function 3783
PLIB_DMA_ChannelXDestinationStartAddressGet function 3773
PLIB_DMA_ChannelXDestinationStartAddressSet function 3774
PLIB_DMA_ChannelXDisable function 3794
PLIB_DMA_ChannelXDisabledDisablesEvents function 3797
PLIB_DMA_ChannelXDisabledEnablesEvents function 3798
PLIB_DMA_ChannelXEnable function 3794
PLIB_DMA_ChannelXEventIsDetected function 3820
PLIB_DMA_ChannelXINTSourceDisable function 3761
PLIB_DMA_ChannelXINTSourceEnable function 3761
PLIB_DMA_ChannelXINTSourceFlagClear function 3762
PLIB_DMA_ChannelXINTSourceFlagGet function 3763
PLIB_DMA_ChannelXINTSourceFlagSet function 3764
PLIB_DMA_ChannelXINTSourceIsEnabled function 3764
PLIB_DMA_ChannelXIsEnabled function 3821
PLIB_DMA_ChannelXNullWriteModeDisable function 3768
PLIB_DMA_ChannelXNullWriteModeEnable function 3768
PLIB_DMA_ChannelXNullWriteModeIsEnabled function 3822
PLIB_DMA_ChannelXOperatingTransferModeGet function 3769
PLIB_DMA_ChannelXOperatingTransferModeSelect function 3769
PLIB_DMA_ChannelXPatternDataGet function 3783
PLIB_DMA_ChannelXPatternDataSet function 3784
PLIB_DMA_ChannelXPatternIgnoreByteDisable function 3799
PLIB_DMA_ChannelXPatternIgnoreByteEnable function 3799
PLIB_DMA_ChannelXPatternIgnoreByteIsEnabled function 3800
PLIB_DMA_ChannelXPatternIgnoreGet function 3800
PLIB_DMA_ChannelXPatternIgnoreSet function 3801
PLIB_DMA_ChannelXPatternLengthGet function 3802
PLIB_DMA_ChannelXPatternLengthSet function 3802
PLIB_DMA_ChannelXPeripheralAddressGet function 3774
PLIB_DMA_ChannelXPeripheralAddressSet function 3775
PLIB_DMA_ChannelXPingPongModeGet function 3822
PLIB_DMA_ChannelXPriorityGet function 3795
PLIB_DMA_ChannelXPrioritySelect function 3796
PLIB_DMA_ChannelXReloadDisable function 3770
PLIB_DMA_ChannelXReloadEnable function 3771
PLIB_DMA_ChannelXReloadIsEnabled function 3772
PLIB_DMA_ChannelXSourceAddressModeGet function 3771
PLIB_DMA_ChannelXSourceAddressModeSelect function 3772
PLIB_DMA_ChannelXSourcePointerGet function 3785
PLIB_DMA_ChannelXSourceSizeGet function 3785
PLIB_DMA_ChannelXSourceSizeSet function 3786
PLIB_DMA_ChannelXSourceStartAddressGet function 3776
PLIB_DMA_ChannelXSourceStartAddressSet function 3776
PLIB_DMA_ChannelXStartAddressOffsetGet function 3777
PLIB_DMA_ChannelXStartAddressOffsetSet function 3778
PLIB_DMA_ChannelXStartIRQSet function 3757
PLIB_DMA_ChannelXTransferCountGet function 3787
PLIB_DMA_ChannelXTransferCountSet function 3787
PLIB_DMA_ChannelXTransferDirectionGet function 3796
PLIB_DMA_ChannelXTransferDirectionSelect function 3797
PLIB_DMA_ChannelXTriggerDisable function 3758
PLIB_DMA_ChannelXTriggerEnable function 3759
PLIB_DMA_ChannelXTriggerIsEnabled function 3759
PLIB_DMA_ChannelXTriggerSourceNumberGet function 3760
PLIB_DMA_CRCAppendModeDisable function 3803
PLIB_DMA_CRCAppendModeEnable function 3804
PLIB_DMA_CRCAppendModeIsEnabled function 3804
PLIB_DMA_CRCBitOrderSelect function 3805
PLIB_DMA_CRCByteOrderGet function 3805
PLIB_DMA_CRCByteOrderSelect function 3806
PLIB_DMA_CRCChannelGet function 3806
PLIB_DMA_CRCChannelSelect function 3807
PLIB_DMA_CRCDataRead function 3807
PLIB_DMA_CRCDataWrite function 3808
PLIB_DMA_CRCDisable function 3808
PLIB_DMA_CRCEnable function 3809
PLIB_DMA_CRCIsEnabled function 3814
PLIB_DMA_CRCPolynomialLengthGet function 3809
PLIB_DMA_CRCPolynomialLengthSet function 3810
PLIB_DMA_CRCTypeGet function 3810
PLIB_DMA_CRCTypeSet function 3811
PLIB_DMA_CRCWriteByteOrderAlter function 3812
PLIB_DMA_CRCWriteByteOrderMaintain function 3812
PLIB_DMA_CRCXOREnableGet function 3813
PLIB_DMA_CRCXOREnableSet function 3813
PLIB_DMA_Disable function 3753
PLIB_DMA_Enable function 3753
PLIB_DMA_ExistsAbortTransfer function 3823
PLIB_DMA_ExistsBusy function 3824
PLIB_DMA_ExistsChannelBits function 3825
PLIB_DMA_ExistsChannelX function 3825
PLIB_DMA_ExistsChannelXAbortIRQ function 3826
PLIB_DMA_ExistsChannelXAuto function 3826
PLIB_DMA_ExistsChannelXBusy function 3827
PLIB_DMA_ExistsChannelXCellProgressPointer function 3827
PLIB_DMA_ExistsChannelXCellSize function 3828
PLIB_DMA_ExistsChannelXChain function 3828
PLIB_DMA_ExistsChannelXChainEnbl function 3829
PLIB_DMA_ExistsChannelXDestinationPointer function 3829
PLIB_DMA_ExistsChannelXDestinationSize function 3830
PLIB_DMA_ExistsChannelXDestinationStartAddress function 3831
PLIB_DMA_ExistsChannelXDisabled function 3831
PLIB_DMA_ExistsChannelXEvent function 3832
PLIB_DMA_ExistsChannelXINTSource function 3832
PLIB_DMA_ExistsChannelXINTSourceFlag function 3833
PLIB_DMA_ExistsChannelXPatternData function 3833
PLIB_DMA_ExistsChannelXPatternIgnore function 3834
PLIB_DMA_ExistsChannelXPatternIgnoreByte function 3834
PLIB_DMA_ExistsChannelXPatternLength function 3835
PLIB_DMA_ExistsChannelXPriority function 3836
PLIB_DMA_ExistsChannelXSourcePointer function 3836
PLIB_DMA_ExistsChannelXSourceSize function 3837
PLIB_DMA_ExistsChannelXSourceStartAddress function 3837
PLIB_DMA_ExistsChannelXStartIRQ function 3838
PLIB_DMA_ExistsChannelXTrigger function 3838
PLIB_DMA_ExistsCRC function 3839
PLIB_DMA_ExistsCRCAppendMode function 3839
PLIB_DMA_ExistsCRCBitOrder function 3840
PLIB_DMA_ExistsCRCByteOrder function 3841
PLIB_DMA_ExistsCRCChannel function 3841
PLIB_DMA_ExistsCRCData function 3842
PLIB_DMA_ExistsCRCPolynomialLength function 3842
PLIB_DMA_ExistsCRCType function 3843
PLIB_DMA_ExistsCRCWriteByteOrder function 3843
PLIB_DMA_ExistsCRCXOREnable function 3844

PLIB_DMA_ExistsEnableControl function 3844
PLIB_DMA_ExistsLastBusAccess function 3845
PLIB_DMA_ExistsRecentAddress function 3846
PLIB_DMA_ExistsStartTransfer function 3846
PLIB_DMA_ExistsStopInIdle function 3847
PLIB_DMA_ExistsSuspend function 3847
PLIB_DMA_IsBusy function 3814
PLIB_DMA_IsEnabled function 3815
PLIB_DMA_LastBusAccessIsRead function 3815
PLIB_DMA_LastBusAccessIsWrite function 3816
PLIB_DMA_RecentAddressAccessed function 3816
PLIB_DMA_StartTransferSet function 3756
PLIB_DMA_StopInIdleDisable function 3754
PLIB_DMA_StopInIdleEnable function 3754
PLIB_DMA_SuspendDisable function 3755
PLIB_DMA_SuspendEnable function 3755
PLIB_DMA_SuspendIsEnabled function 3817
plib_dmt.h 3699
PLIB_DMT_BAD1Get function 3689
PLIB_DMT_BAD2Get function 3690
PLIB_DMT_ClearStep1 function 3687
PLIB_DMT_ClearStep2 function 3687
PLIB_DMT_CounterGet function 3691
PLIB_DMT_Disable function 3688
PLIB_DMT_Enable function 3689
PLIB_DMT_EventOccurred function 3694
PLIB_DMT_ExistsCounter function 3694
PLIB_DMT_ExistsEnableControl function 3695
PLIB_DMT_ExistsPostscalerInterval function 3695
PLIB_DMT_ExistsPostscalerValue function 3696
PLIB_DMT_ExistsStatus function 3697
PLIB_DMT_ExistsStep1 function 3697
PLIB_DMT_ExistsStep2 function 3698
plib_dmt_help.h 3700
PLIB_DMT_IsEnabled function 3691
PLIB_DMT_PostscalerIntervalGet function 3692
PLIB_DMT_PostscalerValueGet function 3692
PLIB_DMT_WindowIsOpen function 3693
plib_ebi.h 3975
PLIB_EBI_AddressHoldTimeGet function 3938
PLIB_EBI_AddressPinEnableBitsSet function 3946
PLIB_EBI_AddressSetupTimeGet function 3939
PLIB_EBI_BaseAddressGet function 3939
PLIB_EBI_BaseAddressSet function 3947
PLIB_EBI_ByteSelectPinSet function 3947
PLIB_EBI_ChipSelectEnableSet function 3948
PLIB_EBI_ControlEnableGet function 3940
PLIB_EBI_ControlEnableSet function 3956
PLIB_EBI_DataEnableSet function 3948
PLIB_EBI_DataTurnAroundTimeGet function 3940
PLIB_EBI_ExistsAddressHoldTime function 3957
PLIB_EBI_ExistsAddressPinEnableBits function 3957
PLIB_EBI_ExistsAddressSetupTime function 3958
PLIB_EBI_ExistsBaseAddress function 3958
PLIB_EBI_ExistsByteSelectPin function 3959
PLIB_EBI_ExistsChipSelectEnable function 3959
PLIB_EBI_ExistsControlEnable function 3960
PLIB_EBI_ExistsDataEnable function 3960
PLIB_EBI_ExistsDataTurnAroundTime function 3961
PLIB_EBI_ExistsFlashPowerDownMode function 3962
PLIB_EBI_ExistsFlashResetPin function 3962
PLIB_EBI_ExistsFlashTiming function 3963
PLIB_EBI_ExistsMemoryCharacteristics function 3963
PLIB_EBI_ExistsMemoryPaging function 3964
PLIB_EBI_ExistsMemoryTimingConfig function 3964
PLIB_EBI_ExistsPageMode function 3965
PLIB_EBI_ExistsPageReadTime function 3965
PLIB_EBI_ExistsReadCycleTime function 3966
PLIB_EBI_ExistsReadyMode function 3966
PLIB_EBI_ExistsReadyPin1Config function 3967
PLIB_EBI_ExistsReadyPin2Config function 3968
PLIB_EBI_ExistsReadyPin3Config function 3968
PLIB_EBI_ExistsReadyPinSens function 3969
PLIB_EBI_ExistsStaticMemoryWidthRegister function 3969
PLIB_EBI_ExistsWriteOutputControl function 3970
PLIB_EBI_ExistsWritePulseWidth function 3970
PLIB_EBI_FlashPowerDownModeGet function 3941
PLIB_EBI_FlashPowerDownModeSet function 3949
PLIB_EBI_FlashResetPinGet function 3941
PLIB_EBI_FlashResetPinSet function 3949
PLIB_EBI_FlashTimingGet function 3946
PLIB_EBI_FlashTimingSet function 3950
plib_ebi_help.h 3976
PLIB_EBI_MemoryCharacteristicsSet function 3950
PLIB_EBI_MemoryPageSizeGet function 3945
PLIB_EBI_MemoryPagingSet function 3951
PLIB_EBI_MemoryTimingConfigSet function 3952
PLIB_EBI_PageModeGet function 3942
PLIB_EBI_PageReadCycleTimeGet function 3942
PLIB_EBI_ReadCycleTimeGet function 3943
PLIB_EBI_ReadyModeGet function 3943
PLIB_EBI_ReadyModeSet function 3956
PLIB_EBI_ReadyPin1ConfigSet function 3952
PLIB_EBI_ReadyPin2ConfigSet function 3953
PLIB_EBI_ReadyPin3ConfigSet function 3953
PLIB_EBI_ReadyPinSensSet function 3954
PLIB_EBI_StaticMemoryWidthRegisterGet function 3944
PLIB_EBI_StaticMemoryWidthRegisterSet function 3954
PLIB_EBI_WriteOutputControlSet function 3955
PLIB_EBI_WritePulseWidthGet function 3944
plib_eth.h 4130
PLIB_ETH_AlignErrorCountClear function 4096
PLIB_ETH_AlignErrorCountGet function 4097
PLIB_ETH_AutoDetectPadClear function 3999
PLIB_ETH_AutoDetectPadGet function 4000
PLIB_ETH_AutoDetectPadSet function 4000
PLIB_ETH_AutoFlowControlDisable function 4074
PLIB_ETH_AutoFlowControlEnable function 4074
PLIB_ETH_AutoFlowControlIsEnabled function 4075
PLIB_ETH_BackPresNoBackoffDisable function 4076
PLIB_ETH_BackPresNoBackoffEnable function 4076
PLIB_ETH_BackPresNoBackoffIsEnabled function 4077
PLIB_ETH_BackToBackIPGGet function 4001
PLIB_ETH_BackToBackIPGSet function 4001
PLIB_ETH_CollisionWindowGet function 4002
PLIB_ETH_CollisionWindowSet function 4002

PLIB_ETH_CRCDisable function 4057
PLIB_ETH_CRCEnable function 4057
PLIB_ETH_CRCIsEnabled function 4058
PLIB_ETH_DataNotValid function 4058
PLIB_ETH_DelayedCRCDisable function 4003
PLIB_ETH_DelayedCRCEnable function 4003
PLIB_ETH_DelayedCRCIsEnabled function 4004
PLIB_ETH_Disable function 4005
PLIB_ETH_Enable function 4005
PLIB_ETH_EthernetIsBusy function 4059
PLIB_ETH_ExcessDeferDisable function 4006
PLIB_ETH_ExcessDeferEnable function 4006
PLIB_ETH_ExcessDeferIsEnabled function 4007
PLIB_ETH_ExistsAlignmentErrorCount function 4104
PLIB_ETH_ExistsAutoFlowControl function 4105
PLIB_ETH_ExistsCollisionCounts function 4106
PLIB_ETH_ExistsCollisionWindow function 4106
PLIB_ETH_ExistsEnable function 4107
PLIB_ETH_ExistsEthernetControllerStatus function 4107
PLIB_ETH_ExistsFCSErrorCount function 4108
PLIB_ETH_ExistsFramesTransmittedOK function 4108
PLIB_ETH_ExistsFramexReceivedOK function 4109
PLIB_ETH_ExistsHashTable function 4109
PLIB_ETH_ExistsInterPacketGaps function 4110
PLIB_ETH_ExistsInterrupt function 4111
PLIB_ETH_ExistsMAC_Configuration function 4111
PLIB_ETH_ExistsMAC_Resets function 4113
PLIB_ETH_ExistsMAC_Testing function 4113
PLIB_ETH_ExistsManualFlowControl function 4114
PLIB_ETH_ExistsMaxFrameLength function 4115
PLIB_ETH_ExistsMIIM_Config function 4115
PLIB_ETH_ExistsMIIM_Indicators function 4116
PLIB_ETH_ExistsMIIMAddresses function 4116
PLIB_ETH_ExistsMIIMReadWrite function 4117
PLIB_ETH_ExistsMIIMScanMode function 4118
PLIB_ETH_ExistsMIIWriteReadData function 4118
PLIB_ETH_ExistsPatternMatch function 4119
PLIB_ETH_ExistsPauseTimer function 4119
PLIB_ETH_ExistsReceiveBufferSize function 4120
PLIB_ETH_ExistsReceiveFilters function 4120
PLIB_ETH_ExistsReceiveOverflowCount function 4121
PLIB_ETH_ExistsReceiveWmarks function 4122
PLIB_ETH_ExistsRetransmissionMaximum function 4122
PLIB_ETH_ExistsRMII_Support function 4123
PLIB_ETH_ExistsRxBufferCountDecrement function 4123
PLIB_ETH_ExistsRxEnable function 4124
PLIB_ETH_ExistsRxPacketDescriptorAddress function 4124
PLIB_ETH_ExistsStationAddress function 4125
PLIB_ETH_ExistsStopInIdle function 4125
PLIB_ETH_ExistsTransmitRTS function 4126
PLIB_ETH_ExistsTxPacketDescriptorAddress function 4127
PLIB_ETH_FCSErrorCountClear function 4097
PLIB_ETH_FCSErrorCountGet function 4098
PLIB_ETH_FrameLengthCheckDisable function 4007
PLIB_ETH_FrameLengthCheckEnable function 4008
PLIB_ETH_FrameLengthCheckIsEnabled function 4008
PLIB_ETH_FramesRxdOkCountClear function 4098
PLIB_ETH_FramesRxdOkCountGet function 4099
PLIB_ETH_FramesTxdOkCountClear function 4099
PLIB_ETH_FramesTxdOkCountGet function 4100
PLIB_ETH_FullDuplexDisable function 4009
PLIB_ETH_FullDuplexEnable function 4009
PLIB_ETH_FullDuplexIsEnabled function 4010
PLIB_ETH_HashTableGet function 4062
PLIB_ETH_HashTableSet function 4063
PLIB_ETH_HugeFrameDisable function 4010
PLIB_ETH_HugeFrameEnable function 4011
PLIB_ETH_HugeFrameIsEnabled function 4012
PLIB_ETH_InterruptClear function 4091
PLIB_ETH_InterruptSet function 4092
PLIB_ETH_InterruptsGet function 4092
PLIB_ETH_InterruptSourceDisable function 4093
PLIB_ETH_InterruptSourceEnable function 4093
PLIB_ETH_InterruptSourceIsEnabled function 4094
PLIB_ETH_InterruptSourcesGet function 4095
PLIB_ETH_InterruptStatusGet function 4095
PLIB_ETH_IsEnabled function 4012
PLIB_ETH_LinkHasFailed function 4059
PLIB_ETH_LongPreambleDisable function 4013
PLIB_ETH_LongPreambleEnable function 4013
PLIB_ETH_LongPreambleIsEnabled function 4014
PLIB_ETH_LoopbackDisable function 4014
PLIB_ETH_LoopbackEnable function 4015
PLIB_ETH_LoopbackIsEnabled function 4015
PLIB_ETH_ManualFlowControlDisable function 4077
PLIB_ETH_ManualFlowControlEnable function 4078
PLIB_ETH_ManualFlowControlIsEnabled function 4078
PLIB_ETH_MaxFrameLengthGet function 4016
PLIB_ETH_MaxFrameLengthSet function 4016
PLIB_ETH_MCSRxResetDisable function 4030
PLIB_ETH_MCSRxResetEnable function 4030
PLIB_ETH_MCSRxResetIsEnabled function 4031
PLIB_ETH_MCSTxResetDisable function 4031
PLIB_ETH_MCSTxResetEnable function 4032
PLIB_ETH_MCSTxResetIsEnabled function 4032
PLIB_ETH_MIIMClockGet function 4017
PLIB_ETH_MIIMClockSet function 4018
PLIB_ETH_MIIMIsBusy function 4060
PLIB_ETH_MIIMIsScanning function 4060
PLIB_ETH_MIIMNoPreDisable function 4018
PLIB_ETH_MIIMNoPreEnable function 4019
PLIB_ETH_MIIMNoPreIsEnabled function 4019
PLIB_ETH_MIIMReadDataGet function 4033
PLIB_ETH_MIIMReadStart function 4034
PLIB_ETH_MIIMResetDisable function 4034
PLIB_ETH_MIIMResetEnable function 4035
PLIB_ETH_MIIMResetIsEnabled function 4035
PLIB_ETH_MIIMScanIncrDisable function 4036
PLIB_ETH_MIIMScanIncrEnable function 4036
PLIB_ETH_MIIMScanIncrIsEnabled function 4037
PLIB_ETH_MIIMScanModeDisable function 4038
PLIB_ETH_MIIMScanModeEnable function 4038
PLIB_ETH_MIIMScanModeIsEnabled function 4039
PLIB_ETH_MIIMWriteDataSet function 4039
PLIB_ETH_MIIMWriteStart function 4040
PLIB_ETH_MIIResetDisable function 4040

PLIB_ETH_MIIResetEnable function 4041
PLIB_ETH_MIIResetIsEnabled function 4041
PLIB_ETH_MultipleCollisionCountClear function 4100
PLIB_ETH_MultipleCollisionCountGet function 4101
PLIB_ETH_NoBackoffDisable function 4079
PLIB_ETH_NoBackoffEnable function 4079
PLIB_ETH_NoBackoffIsEnabled function 4080
PLIB_ETH_NonBackToBackIPG1Get function 4020
PLIB_ETH_NonBackToBackIPG1Set function 4020
PLIB_ETH_NonBackToBackIPG2Get function 4021
PLIB_ETH_NonBackToBackIPG2Set function 4021
PLIB_ETH_PassAllDisable function 4063
PLIB_ETH_PassAllEnable function 4064
PLIB_ETH_PassAllIsEnabled function 4064
PLIB_ETH_PatternMatchChecksumGet function 4065
PLIB_ETH_PatternMatchChecksumSet function 4065
PLIB_ETH_PatternMatchGet function 4066
PLIB_ETH_PatternMatchModeGet function 4066
PLIB_ETH_PatternMatchModeSet function 4067
PLIB_ETH_PatternMatchOffsetGet function 4068
PLIB_ETH_PatternMatchOffsetSet function 4068
PLIB_ETH_PatternMatchSet function 4069
PLIB_ETH_PauseTimerGet function 4022
PLIB_ETH_PauseTimerSet function 4023
PLIB_ETH_PHYAddressGet function 4042
PLIB_ETH_PHYAddressSet function 4042
PLIB_ETH_PurePreambleDisable function 4069
PLIB_ETH_PurePreambleEnable function 4070
PLIB_ETH_PurePreambleIsEnabled function 4070
PLIB_ETH_ReceiveBufferSizeGet function 4023
PLIB_ETH_ReceiveBufferSizeSet function 4024
PLIB_ETH_ReceiveDisable function 4024
PLIB_ETH_ReceiveEnable function 4025
PLIB_ETH_ReceiveFilterDisable function 4071
PLIB_ETH_ReceiveFilterEnable function 4071
PLIB_ETH_ReceiveFilterIsEnable function 4072
PLIB_ETH_ReceiveIsBusy function 4061
PLIB_ETH_ReceiveIsEnabled function 4025
PLIB_ETH_RegisterAddressGet function 4043
PLIB_ETH_RegisterAddressSet function 4044
PLIB_ETH_ReTxMaxGet function 4026
PLIB_ETH_ReTxMaxSet function 4026
PLIB_ETH_RMIIResetDisable function 4044
PLIB_ETH_RMIIResetEnable function 4045
PLIB_ETH_RMIIResetIsEnabled function 4045
PLIB_ETH_RMIISpeedGet function 4027
PLIB_ETH_RMIISpeedSet function 4028
PLIB_ETH_RxBufferCountDecrement function 4046
PLIB_ETH_RxDisable function 4046
PLIB_ETH_RxEmptyWmarkGet function 4080
PLIB_ETH_RxEmptyWmarkSet function 4081
PLIB_ETH_RxEnable function 4047
PLIB_ETH_RxFullWmarkGet function 4082
PLIB_ETH_RxFullWmarkSet function 4082
PLIB_ETH_RxFuncResetDisable function 4047
PLIB_ETH_RxFuncResetEnable function 4048
PLIB_ETH_RxFuncResetIsEnabled function 4048
PLIB_ETH_RxIsEnabled function 4049
PLIB_ETH_RxOverflowCountClear function 4102
PLIB_ETH_RxOverflowCountGet function 4102
PLIB_ETH_RxPacketCountGet function 4103
PLIB_ETH_RxPacketDescAddrGet function 4050
PLIB_ETH_RxPacketDescAddrSet function 4050
PLIB_ETH_RxPauseDisable function 4083
PLIB_ETH_RxPauseEnable function 4083
PLIB_ETH_RxPauseIsEnabled function 4084
PLIB_ETH_ShortcutQuantaDisable function 4084
PLIB_ETH_ShortcutQuantaEnable function 4085
PLIB_ETH_ShortcutQuantaIsEnabled function 4085
PLIB_ETH_SimResetDisable function 4086
PLIB_ETH_SimResetEnable function 4087
PLIB_ETH_SimResetIsEnabled function 4087
PLIB_ETH_SingleCollisionCountClear function 4103
PLIB_ETH_SingleCollisionCountGet function 4104
PLIB_ETH_StationAddressGet function 4073
PLIB_ETH_StationAddressSet function 4073
PLIB_ETH_StopInIdleDisable function 4028
PLIB_ETH_StopInIdleEnable function 4029
PLIB_ETH_StopInIdleIsEnabled function 4029
PLIB_ETH_TestBackPressDisable function 4088
PLIB_ETH_TestBackPressEnable function 4088
PLIB_ETH_TestBackPressIsEnabled function 4089
PLIB_ETH_TestPauseDisable function 4051
PLIB_ETH_TestPauseEnable function 4051
PLIB_ETH_TestPauseIsEnabled function 4052
PLIB_ETH_TransmitIsBusy function 4062
PLIB_ETH_TxFuncResetDisable function 4052
PLIB_ETH_TxFuncResetEnable function 4053
PLIB_ETH_TxFuncResetIsEnabled function 4053
PLIB_ETH_TxPacketDescAddrGet function 4054
PLIB_ETH_TxPacketDescAddrSet function 4054
PLIB_ETH_TxPauseDisable function 4089
PLIB_ETH_TxPauseEnable function 4090
PLIB_ETH_TxPauseIsEnabled function 4090
PLIB_ETH_TxRTSDisable function 4055
PLIB_ETH_TxRTSEnable function 4055
PLIB_ETH_TxRTSIsEnabled function 4056
plib_glcd.h 4234
PLIB_GLCD_BackgroundColorGet function 4157
PLIB_GLCD_BackgroundColorSet function 4158
PLIB_GLCD_BackPorchXGet function 4146
PLIB_GLCD_BackPorchXYSet function 4147
PLIB_GLCD_BackPorchYGet function 4148
PLIB_GLCD_BlankingXGet function 4154
PLIB_GLCD_BlankingXYSet function 4148
PLIB_GLCD_BlankingYGet function 4154
PLIB_GLCD_ClockDividerGet function 4156
PLIB_GLCD_ClockDividerSet function 4155
PLIB_GLCD_CursorDataGet function 4179
PLIB_GLCD_CursorDataSet function 4180
PLIB_GLCD_CursorDisable function 4180
PLIB_GLCD_CursorEnable function 4181
PLIB_GLCD_CursorIsEnabled function 4181
PLIB_GLCD_CursorLUTGet function 4182
PLIB_GLCD_CursorLUTSet function 4182
PLIB_GLCD_CursorXGet function 4183

PLIB_GLCD_CursorXYSet function 4184
PLIB_GLCD_CursorYGet function 4184
PLIB_GLCD_DESignalLevelGet function 4198
PLIB_GLCD_Disable function 4144
PLIB_GLCD_DitheringDisable function 4193
PLIB_GLCD_DitheringEnable function 4194
PLIB_GLCD_DitheringIsEnabled function 4199
PLIB_GLCD_Enable function 4144
PLIB_GLCD_ExistsBackgroundColor function 4203
PLIB_GLCD_ExistsBackPorchXY function 4204
PLIB_GLCD_ExistsBlankingXY function 4205
PLIB_GLCD_ExistsClockDivider function 4205
PLIB_GLCD_ExistsCursor function 4206
PLIB_GLCD_ExistsCursorData function 4206
PLIB_GLCD_ExistsCursorLUT function 4207
PLIB_GLCD_ExistsCursorXY function 4207
PLIB_GLCD_ExistsDESignalLevel function 4208
PLIB_GLCD_ExistsDithering function 4209
PLIB_GLCD_ExistsEnable function 4209
PLIB_GLCD_ExistsForceOutputBlank function 4210
PLIB_GLCD_ExistsFormattingClockDivide function 4211
PLIB_GLCD_ExistsFrontPorchXY function 4211
PLIB_GLCD_ExistsGlobalColorLUT function 4212
PLIB_GLCD_ExistsHSyncInterruptEnable function 4212
PLIB_GLCD_ExistsHSyncSignalLevel function 4213
PLIB_GLCD_ExistsIRQTriggerControl function 4214
PLIB_GLCD_ExistsIsLastRow function 4214
PLIB_GLCD_ExistsIsVerticalBlankingActive function 4215
PLIB_GLCD_ExistsLayerBaseAddress function 4215
PLIB_GLCD_ExistsLayerBilinearFilterEnable function 4216
PLIB_GLCD_ExistsLayerColorMode function 4217
PLIB_GLCD_ExistsLayerDestBlendFunc function 4217
PLIB_GLCD_ExistsLayerEnable function 4218
PLIB_GLCD_ExistsLayerForceWithGlobalAlpha function 4218
PLIB_GLCD_ExistsLayerGlobalAlpha function 4219
PLIB_GLCD_ExistsLayerPreMultiplyImageAlpha function 4220
PLIB_GLCD_ExistsLayerResXY function 4220
PLIB_GLCD_ExistsLayerSizeXY function 4221
PLIB_GLCD_ExistsLayerSrcBlendFunc function 4221
PLIB_GLCD_ExistsLayerStartXY function 4222
PLIB_GLCD_ExistsLayerStride function 4223
PLIB_GLCD_ExistsLinesPrefetch function 4223
PLIB_GLCD_ExistsPaletteGammaRamp function 4224
PLIB_GLCD_ExistsResolutionXY function 4225
PLIB_GLCD_ExistsRGBSequentialMode function 4225
PLIB_GLCD_ExistsSignalPolarity function 4226
PLIB_GLCD_ExistsSingleCyclePerLineVsync function 4226
PLIB_GLCD_ExistsVSyncInterruptEnable function 4227
PLIB_GLCD_ExistsVSyncSignalLevel function 4228
PLIB_GLCD_ExistsYUVOutput function 4228
PLIB_GLCD_ForceOutputBlankDisable function 4195
PLIB_GLCD_ForceOutputBlankEnable function 4195
PLIB_GLCD_ForceOutputBlankIsEnabled function 4200
PLIB_GLCD_FormattingClockDivideDisable function 4149
PLIB_GLCD_FormattingClockDivideEnable function 4149
PLIB_GLCD_FormattingClockDivideIsEnabled function 4150
PLIB_GLCD_FrontPorchXGet function 4150
PLIB_GLCD_FrontPorchXYSet function 4151
PLIB_GLCD_FrontPorchYGet function 4152
PLIB_GLCD_GlobalColorLUTGet function 4185
PLIB_GLCD_GlobalColorLUTSet function 4185
PLIB_GLCD_HSyncInterruptDisable function 4188
PLIB_GLCD_HSyncInterruptEnable function 4188
PLIB_GLCD_HSyncInterruptIsEnabled function 4189
PLIB_GLCD_HSyncSignalLevelGet function 4189
PLIB_GLCD_IRQTriggerControlGet function 4190
PLIB_GLCD_IRQTriggerControlSet function 4191
PLIB_GLCD_IsEnabled function 4200
PLIB_GLCD_IsLastRow function 4201
PLIB_GLCD_IsVerticalBlankingActive function 4201
PLIB_GLCD_LayerBaseAddressGet function 4158
PLIB_GLCD_LayerBaseAddressSet function 4159
PLIB_GLCD_LayerBilinearFilterDisable function 4160
PLIB_GLCD_LayerBilinearFilterEnable function 4160
PLIB_GLCD_LayerBilinearFilterIsEnabled function 4161
PLIB_GLCD_LayerColorModeGet function 4161
PLIB_GLCD_LayerColorModeSet function 4162
PLIB_GLCD_LayerDestBlendFuncGet function 4163
PLIB_GLCD_LayerDestBlendFuncSet function 4164
PLIB_GLCD_LayerDisable function 4164
PLIB_GLCD_LayerEnable function 4165
PLIB_GLCD_LayerForceWithGlobalAlphaDisable function 4165
PLIB_GLCD_LayerForceWithGlobalAlphaEnable function 4166
PLIB_GLCD_LayerForceWithGlobalAlphaIsEnabled function 4166
PLIB_GLCD_LayerGlobalAlphaGet function 4167
PLIB_GLCD_LayerGlobalAlphaSet function 4168
PLIB_GLCD_LayerIsEnabled function 4168
PLIB_GLCD_LayerPreMultiplyImageAlphaDisable function 4169
PLIB_GLCD_LayerPreMultiplyImageAlphaEnable function 4169
PLIB_GLCD_LayerPreMultiplyImageAlphaIsEnabled function 4170
PLIB_GLCD_LayerResXGet function 4171
PLIB_GLCD_LayerResXYSet function 4171
PLIB_GLCD_LayerResYGet function 4172
PLIB_GLCD_LayerSizeXGet function 4173
PLIB_GLCD_LayerSizeXYSet function 4173
PLIB_GLCD_LayerSizeYGet function 4174
PLIB_GLCD_LayerSrcBlendFuncGet function 4174
PLIB_GLCD_LayerSrcBlendFuncSet function 4175
PLIB_GLCD_LayerStartXGet function 4176
PLIB_GLCD_LayerStartXYSet function 4176
PLIB_GLCD_LayerStartYGet function 4177
PLIB_GLCD_LayerStrideGet function 4178
PLIB_GLCD_LayerStrideSet function 4178
PLIB_GLCD_LinesPrefetchGet function 4152
PLIB_GLCD_LinesPrefetchSet function 4155
PLIB_GLCD_PaletteGammaRampDisable function 4186
PLIB_GLCD_PaletteGammaRampEnable function 4187
PLIB_GLCD_PaletteGammaRampIsEnabled function 4187
PLIB_GLCD_ResolutionXGet function 4153
PLIB_GLCD_ResolutionXYSet function 4157
PLIB_GLCD_ResolutionYGet function 4153
PLIB_GLCD_RGBSequentialModeGet function 4202
PLIB_GLCD_RGBSequentialModeSet function 4145
PLIB_GLCD_SignalPolarityGet function 4145
PLIB_GLCD_SignalPolaritySet function 4146
PLIB_GLCD_SingleCyclePerLineVsyncDisable function 4196

PLIB_GLCD_SingleCyclePerLineVsyncEnable function 4196
PLIB_GLCD_SingleCyclePerLineVsyncIsEnabled function 4197
PLIB_GLCD_VSyncInterruptDisable function 4191
PLIB_GLCD_VSyncInterruptEnable function 4192
PLIB_GLCD_VSyncInterruptIsEnabled function 4192
PLIB_GLCD_VSyncSignalLevelGet function 4193
PLIB_GLCD_YUVOutputDisable function 4197
PLIB_GLCD_YUVOutputEnable function 4198
PLIB_GLCD_YUVOutputIsEnabled function 4203
plib_i2c.h 4338
PLIB_I2C_AcksequenceIsInProgress function 4292
PLIB_I2C_ArbitrationLossClear function 4273
PLIB_I2C_ArbitrationLossHasOccurred function 4274
PLIB_I2C_BaudRateGet function 4278
PLIB_I2C_BaudRateSet function 4279
PLIB_I2C_BusIsIdle function 4275
PLIB_I2C_DataLineHoldTimeSet function 4293
PLIB_I2C_Disable function 4263
PLIB_I2C_Enable function 4264
PLIB_I2C_ExistsAcksequenceProgress function 4314
PLIB_I2C_ExistsArbitrationLoss function 4315
PLIB_I2C_ExistsBaudRate function 4315
PLIB_I2C_ExistsBusIsIdle function 4316
PLIB_I2C_ExistsClockStretching function 4316
PLIB_I2C_ExistsDataLineHoldTime function 4317
PLIB_I2C_ExistsGeneralCall function 4317
PLIB_I2C_ExistsGeneralCallAddressDetect function 4318
PLIB_I2C_ExistsHighFrequency function 4318
PLIB_I2C_ExistsIPMI function 4319
PLIB_I2C_ExistsMasterReceiverClock1Byte function 4320
PLIB_I2C_ExistsMasterStart function 4320
PLIB_I2C_ExistsMasterStartRepeat function 4321
PLIB_I2C_ExistsMasterStop function 4321
PLIB_I2C_ExistsModuleEnable function 4322
PLIB_I2C_ExistsReceivedByteAcknowledge function 4322
PLIB_I2C_ExistsReceivedByteAvailable function 4323
PLIB_I2C_ExistsReceivedByteGet function 4323
PLIB_I2C_ExistsReceiverOverflow function 4324
PLIB_I2C_ExistsReservedAddressProtect function 4324
PLIB_I2C_ExistsSlaveAddress10Bit function 4325
PLIB_I2C_ExistsSlaveAddress7Bit function 4325
PLIB_I2C_ExistsSlaveAddressDetect function 4326
PLIB_I2C_ExistsSlaveAddressHoldEnable function 4327
PLIB_I2C_ExistsSlaveBufferOverwrite function 4327
PLIB_I2C_ExistsSlaveBusCollisionDetect function 4328
PLIB_I2C_ExistsSlaveClockHold function 4328
PLIB_I2C_ExistsSlaveDataDetect function 4329
PLIB_I2C_ExistsSlaveDataHoldEnable function 4336
PLIB_I2C_ExistsSlaveInterruptOnStart function 4329
PLIB_I2C_ExistsSlaveInterruptOnStop function 4330
PLIB_I2C_ExistsSlaveMask function 4330
PLIB_I2C_ExistsSlaveReadRequest function 4331
PLIB_I2C_ExistsSMBus function 4332
PLIB_I2C_ExistsStartDetect function 4332
PLIB_I2C_ExistsStopDetect function 4333
PLIB_I2C_ExistsStopInIdle function 4333
PLIB_I2C_ExistsTransmitterByteAcknowledge function 4334
PLIB_I2C_ExistsTransmitterByteComplete function 4334
PLIB_I2C_ExistsTransmitterByteSend function 4335
PLIB_I2C_ExistsTransmitterIsBusy function 4335
PLIB_I2C_ExistsTransmitterOverflow function 4336
PLIB_I2C_GeneralCallDisable function 4264
PLIB_I2C_GeneralCallEnable function 4265
plib_i2c_help.h 4340
PLIB_I2C_HighFrequencyDisable function 4266
PLIB_I2C_HighFrequencyEnable function 4266
PLIB_I2C_IPMIDisable function 4267
PLIB_I2C_IPMIEnable function 4267
PLIB_I2C_MasterReceiverClock1Byte function 4301
PLIB_I2C_MasterReceiverReadyToAcknowledge function 4313
PLIB_I2C_MasterStart function 4301
PLIB_I2C_MasterStartRepeat function 4302
PLIB_I2C_MasterStop function 4303
PLIB_I2C_ReceivedByteAcknowledge function 4309
PLIB_I2C_ReceivedByteGet function 4310
PLIB_I2C_ReceivedByteIsAvailable function 4310
PLIB_I2C_ReceiverByteAcknowledgeHasCompleted function 4311
PLIB_I2C_ReceiverOverflowClear function 4312
PLIB_I2C_ReceiverOverflowHasOccurred function 4313
PLIB_I2C_ReservedAddressProtectDisable function 4268
PLIB_I2C_ReservedAddressProtectEnable function 4269
PLIB_I2C_SlaveAddress10BitGet function 4280
PLIB_I2C_SlaveAddress10BitSet function 4281
PLIB_I2C_SlaveAddress10BitWasDetected function 4281
PLIB_I2C_SlaveAddress7BitGet function 4282
PLIB_I2C_SlaveAddress7BitSet function 4283
PLIB_I2C_SlaveAddressHoldDisable function 4283
PLIB_I2C_SlaveAddressHoldEnable function 4284
PLIB_I2C_SlaveAddressIsDetected function 4285
PLIB_I2C_SlaveAddressIsGeneralCall function 4286
PLIB_I2C_SlaveAddressModeIs10Bits function 4286
PLIB_I2C_SlaveBufferOverwriteDisable function 4293
PLIB_I2C_SlaveBufferOverwriteEnable function 4294
PLIB_I2C_SlaveBusCollisionDetectDisable function 4294
PLIB_I2C_SlaveBusCollisionDetectEnable function 4295
PLIB_I2C_SlaveClockHold function 4296
PLIB_I2C_SlaveClockRelease function 4296
PLIB_I2C_SlaveClockStretchingDisable function 4269
PLIB_I2C_SlaveClockStretchingEnable function 4270
PLIB_I2C_SlaveDataHoldDisable function 4297
PLIB_I2C_SlaveDataHoldEnable function 4298
PLIB_I2C_SlaveDataIsDetected function 4287
PLIB_I2C_SlaveInterruptOnStartDisable function 4298
PLIB_I2C_SlaveInterruptOnStartEnable function 4299
PLIB_I2C_SlaveInterruptOnStopDisable function 4299
PLIB_I2C_SlaveInterruptOnStopEnable function 4300
PLIB_I2C_SlaveMask10BitGet function 4289
PLIB_I2C_SlaveMask10BitSet function 4289
PLIB_I2C_SlaveMask7BitGet function 4290
PLIB_I2C_SlaveMask7BitSet function 4291
PLIB_I2C_SlaveReadIsRequested function 4288
PLIB_I2C_SMBDisable function 4271
PLIB_I2C_SMBEnable function 4271
PLIB_I2C_StartClear function 4275
PLIB_I2C_StartWasDetected function 4276
PLIB_I2C_StopClear function 4277

PLIB_I2C_StopInIdleDisable function 4272
PLIB_I2C_StopInIdleEnable function 4273
PLIB_I2C_StopWasDetected function 4278
PLIB_I2C_TransmitterByteHasCompleted function 4303
PLIB_I2C_TransmitterByteSend function 4304
PLIB_I2C_TransmitterByteWasAcknowledged function 4305
PLIB_I2C_TransmitterIsBusy function 4306
PLIB_I2C_TransmitterIsReady function 4306
PLIB_I2C_TransmitterOverflowClear function 4307
PLIB_I2C_TransmitterOverflowHasOccurred function 4308
plib_ic.h 4365
PLIB_IC_AlternateClockDisable function 4348
PLIB_IC_AlternateClockEnable function 4349
PLIB_IC_AlternateTimerSelect function 4349
PLIB_IC_Buffer16BitGet function 4351
PLIB_IC_Buffer32BitGet function 4351
PLIB_IC_BufferIsEmpty function 4352
PLIB_IC_BufferOverflowHasOccurred function 4352
PLIB_IC_BufferSizeSelect function 4353
PLIB_IC_Disable function 4345
PLIB_IC_Enable function 4346
PLIB_IC_EventsPerInterruptSelect function 4346
PLIB_IC_ExistsAlternateClock function 4356
PLIB_IC_ExistsAlternateTimerSelect function 4350
PLIB_IC_ExistsBufferIsEmptyStatus function 4356
PLIB_IC_ExistsBufferOverflowStatus function 4357
PLIB_IC_ExistsBufferSize function 4357
PLIB_IC_ExistsBufferValue function 4358
PLIB_IC_ExistsCaptureMode function 4358
PLIB_IC_ExistsEdgeCapture function 4359
PLIB_IC_ExistsEnable function 4359
PLIB_IC_ExistsEventsPerInterruptSelect function 4360
PLIB_IC_ExistsStopInIdle function 4360
PLIB_IC_ExistsTimerSelect function 4361
PLIB_IC_FirstCaptureEdgeSelect function 4347
plib_ic_help.h 4366
PLIB_IC_ModeSelect function 4354
PLIB_IC_StopInIdleDisable function 4354
PLIB_IC_StopInIdleEnable function 4355
PLIB_IC_TimerSelect function 4347
plib_int.h 4413
PLIB_INT_CPUCurrentPriorityLevelGet function 4386
PLIB_INT_Disable function 4370
PLIB_INT_Enable function 4371
PLIB_INT_ExistsCPUCurrentPriorityLevel function 4391
PLIB_INT_ExistsEnableControl function 4391
PLIB_INT_ExistsExternalINTEdgeSelect function 4392
PLIB_INT_ExistsINTCPUPriority function 4392
PLIB_INT_ExistsINTCPUVector function 4393
PLIB_INT_ExistsProximityTimerControl function 4393
PLIB_INT_ExistsProximityTimerEnable function 4394
PLIB_INT_ExistsShadowRegisterAssign function 4397
PLIB_INT_ExistsSingleVectorShadowSet function 4394
PLIB_INT_ExistsSoftwareNMI function 4398
PLIB_INT_ExistsSourceControl function 4395
PLIB_INT_ExistsSourceFlag function 4396
PLIB_INT_ExistsVariableOffset function 4398
PLIB_INT_ExistsVectorPriority function 4396
PLIB_INT_ExistsVectorSelect function 4397
PLIB_INT_ExternalFallingEdgeSelect function 4371
PLIB_INT_ExternalRisingEdgeSelect function 4372
PLIB_INT_GetStateAndDisable function 4389
plib_int_help.h 4415
PLIB_INT_IsEnabled function 4372
PLIB_INT_MultiVectorSelect function 4373
PLIB_INT_PriorityGet function 4374
PLIB_INT_ProximityTimerDisable function 4374
PLIB_INT_ProximityTimerEnable function 4375
PLIB_INT_ProximityTimerGet function 4387
PLIB_INT_ProximityTimerSet function 4387
PLIB_INT_SetState function 4377
PLIB_INT_ShadowRegisterAssign function 4378
PLIB_INT_ShadowRegisterGet function 4389
PLIB_INT_SingleVectorSelect function 4375
PLIB_INT_SingleVectorShadowSetDisable function 4376
PLIB_INT_SingleVectorShadowSetEnable function 4377
PLIB_INT_SoftwareNMITrigger function 4379
PLIB_INT_SourceDisable function 4380
PLIB_INT_SourceEnable function 4380
PLIB_INT_SourceFlagClear function 4381
PLIB_INT_SourceFlagGet function 4382
PLIB_INT_SourceFlagSet function 4382
PLIB_INT_SourceIsEnabled function 4383
PLIB_INT_VariableVectorOffsetGet function 4390
PLIB_INT_VariableVectorOffsetSet function 4378
PLIB_INT_VectorGet function 4388
PLIB_INT_VectorPriorityGet function 4384
PLIB_INT_VectorPrioritySet function 4384
PLIB_INT_VectorSubPriorityGet function 4385
PLIB_INT_VectorSubPrioritySet function 4385
plib_nvm.h 4476
PLIB_NVM_BootFlashBank1LowerRegion function 4441
PLIB_NVM_BootFlashBank2IsLowerRegion function 4441
PLIB_NVM_BootFlashBank2LowerRegion function 4442
PLIB_NVM_BootPageWriteProtectionDisable function 4435
PLIB_NVM_BootPageWriteProtectionEnable function 4436
PLIB_NVM_DataBlockSourceAddress function 4427
PLIB_NVM_EEPROMAddress function 4451
PLIB_NVM_EEPROMDataToWrite function 4451
PLIB_NVM_EEPROMDisable function 4443
PLIB_NVM_EEPROMEnable function 4443
PLIB_NVM_EEPROMEraseStart function 4453
PLIB_NVM_EEPROMErrorClear function 4450
PLIB_NVM_EEPROMErrorGet function 4449
PLIB_NVM_EEPROMIsReady function 4444
PLIB_NVM_EEPROMKeySequenceWrite function 4452
PLIB_NVM_EEPROMNextWriteCycleIsLong function 4449
PLIB_NVM_EEPROMOperationAbort function 4456
PLIB_NVM_EEPROMOperationHasCompleted function 4455
PLIB_NVM_EEPROMOperationSelect function 4446
PLIB_NVM_EEPROMRead function 4455
PLIB_NVM_EEPROMReadEnable function 4447
PLIB_NVM_EEPROMReadIsEnabled function 4448
PLIB_NVM_EEPROMReadStart function 4454
PLIB_NVM_EEPROMStopInIdleDisable function 4445
PLIB_NVM_EEPROMStopInIdleEnable function 4444

PLIB_NVM_EEPROMStopInIdleIsEnabled function 4445
PLIB_NVM_EEPROMWriteEnable function 4447
PLIB_NVM_EEPROMWriteIsEnabled function 4448
PLIB_NVM_EEPROMWriteStart function 4453
PLIB_NVM_ExistsAddressModifyControl function 4456
PLIB_NVM_ExistsBootFlashBankRegion function 4465
PLIB_NVM_ExistsBootPageWriteProtect function 4457
PLIB_NVM_ExistsEEPROMAddressControl function 4466
PLIB_NVM_ExistsEEPROMDataControl function 4467
PLIB_NVM_ExistsEEPROMEnableControl function 4467
PLIB_NVM_ExistsEEPROMEnableOperationControl function 4468
PLIB_NVM_ExistsEEPROMErrorStatus function 4468
PLIB_NVM_ExistsEEPROMKeySequence function 4469
PLIB_NVM_ExistsEEPROMLongWriteStatus function 4470
PLIB_NVM_ExistsEEPROMOperationAbortControl function 4470
PLIB_NVM_ExistsEEPROMOperationModeControl function 4471
PLIB_NVM_ExistsEEPROMStartOperationControl function 4471
PLIB_NVM_ExistsEEPROMStopInIdleControl function 4472
PLIB_NVM_ExistsFlashBankRegionSelect function 4457
PLIB_NVM_ExistsFlashWPMemoryRangeProvide function 4458
PLIB_NVM_ExistsKeySequence function 4459
PLIB_NVM_ExistsLockBootSelect function 4459
PLIB_NVM_ExistsLockPFMSelect function 4460
PLIB_NVM_ExistsLowVoltageError function 4460
PLIB_NVM_ExistsLowVoltageStatus function 4461
PLIB_NVM_ExistsMemoryModificationControl function 4461
PLIB_NVM_ExistsOperationMode function 4462
PLIB_NVM_ExistsProvideData function 4462
PLIB_NVM_ExistsProvideQuadData function 4463
PLIB_NVM_ExistsSourceAddress function 4463
PLIB_NVM_ExistsSwapLockControl function 4466
PLIB_NVM_ExistsWriteErrorStatus function 4464
PLIB_NVM_ExistsWriteOperation function 4464
PLIB_NVM_FlashAddressToModify function 4428
PLIB_NVM_FlashEraseStart function 4428
PLIB_NVM_FlashProvideData function 4429
PLIB_NVM_FlashProvideQuadData function 4430
PLIB_NVM_FlashRead function 4430
PLIB_NVM_FlashSwapLockSelect function 4433
PLIB_NVM_FlashSwapLockStatusGet function 4434
PLIB_NVM_FlashWriteCycleHasCompleted function 4432
PLIB_NVM_FlashWriteKeySequence function 4431
PLIB_NVM_FlashWriteProtectMemoryAreaRange function 4434
PLIB_NVM_FlashWriteStart function 4431
plib_nvm_help.h 4478
PLIB_NVM_IsBootMemoryLocked function 4436
PLIB_NVM_IsBootPageWriteProtected function 4437
PLIB_NVM_IsProgramFlashMemoryLocked function 4437
PLIB_NVM_LockBootMemory function 4438
PLIB_NVM_LockProgramFlashMemory function 4439
PLIB_NVM_LowVoltageEventIsActive function 4424
PLIB_NVM_LowVoltageIsDetected function 4425
PLIB_NVM_MemoryModifyEnable function 4426
PLIB_NVM_MemoryModifyInhibit function 4426
PLIB_NVM_MemoryOperationSelect function 4427
PLIB_NVM_ProgramFlashBank1LowerRegion function 4439
PLIB_NVM_ProgramFlashBank2IsLowerRegion function 4440
PLIB_NVM_ProgramFlashBank2LowerRegion function 4440
PLIB_NVM_WriteOperationHasTerminated function 4433
plib_oc.h 4506
PLIB_OC_AlternateClockDisable function 4492
PLIB_OC_AlternateClockEnable function 4493
PLIB_OC_AlternateTimerSelect function 4493
PLIB_OC_Buffer16BitSet function 4488
PLIB_OC_Buffer32BitSet function 4489
PLIB_OC_BufferSizeSelect function 4490
PLIB_OC_Disable function 4486
PLIB_OC_Enable function 4486
PLIB_OC_ExistsAlternateClock function 4497
PLIB_OC_ExistsAlternateTimerSelect function 4502
PLIB_OC_ExistsBufferSize function 4497
PLIB_OC_ExistsBufferValue function 4498
PLIB_OC_ExistsCompareModeSelect function 4498
PLIB_OC_ExistsEnableControl function 4499
PLIB_OC_ExistsFaultInput function 4499
PLIB_OC_ExistsFaultStatus function 4500
PLIB_OC_ExistsPulseWidth function 4501
PLIB_OC_ExistsStopInIdle function 4501
PLIB_OC_ExistsTimerSelect function 4502
PLIB_OC_FaultHasOccurred function 4495
PLIB_OC_FaultInputSelect function 4496
plib_oc_help.h 4507
PLIB_OC_IsEnabled function 4487
PLIB_OC_ModeSelect function 4488
PLIB_OC_PulseWidth16BitSet function 4490
PLIB_OC_PulseWidth32BitSet function 4491
PLIB_OC_StopInIdleDisable function 4494
PLIB_OC_StopInIdleEnable function 4495
PLIB_OC_TimerSelect function 4491
plib_osc.h 4612
PLIB_OSC_BTPLLClockOutDisable function 4553
PLIB_OSC_BTPLLClockOutEnable function 4554
PLIB_OSC_BTPLLClockOutStatus function 4554
PLIB_OSC_BTPLLFrequencyRangeGet function 4555
PLIB_OSC_BTPLLFrequencyRangeSet function 4555
PLIB_OSC_BTPLLInputClockSourceGet function 4556
PLIB_OSC_BTPLLInputClockSourceSet function 4557
PLIB_OSC_BTPLLInputDivisorGet function 4557
PLIB_OSC_BTPLLInputDivisorSet function 4558
PLIB_OSC_BTPLLMultiplierGet function 4558
PLIB_OSC_BTPLLMultiplierSelect function 4559
PLIB_OSC_BTPLLOutputDivisorGet function 4559
PLIB_OSC_BTPLLOutputDivisorSet function 4560
PLIB_OSC_ClockHasFailed function 4574
PLIB_OSC_ClockIsReady function 4524
PLIB_OSC_ClockSlewingIsActive function 4525
PLIB_OSC_ClockStart function 4575
PLIB_OSC_ClockStop function 4575
PLIB_OSC_ClockStopStatus function 4576
PLIB_OSC_ClockSwitchingAbort function 4541
PLIB_OSC_ClockSwitchingIsComplete function 4542
PLIB_OSC_CurrentSysClockGet function 4543
PLIB_OSC_DreamModeDisable function 4523
PLIB_OSC_DreamModeEnable function 4523
PLIB_OSC_DreamModeStatus function 4524
PLIB_OSC_ExistsBTPLLClockOut function 4596

PLIB_OSC_ExistsBTPLLFrequencyRange function 4596
PLIB_OSC_ExistsBTPLLInputClockSource function 4597
PLIB_OSC_ExistsBTPLLInputDivisor function 4597
PLIB_OSC_ExistsBTPLLMultiplier function 4598
PLIB_OSC_ExistsBTPLLOutputDivisor function 4598
PLIB_OSC_ExistsClockDiagStatus function 4599
PLIB_OSC_ExistsClockFail function 4577
PLIB_OSC_ExistsClockReadyStatus function 4592
PLIB_OSC_ExistsClockSlewingStatus function 4593
PLIB_OSC_ExistsDreamModeControl function 4600
PLIB_OSC_ExistsForceLock function 4600
PLIB_OSC_ExistsFRCDivisor function 4577
PLIB_OSC_ExistsFRCTuning function 4578
PLIB_OSC_ExistsOnWaitAction function 4578
PLIB_OSC_ExistsOscCurrentGet function 4579
PLIB_OSC_ExistsOscSelect function 4579
PLIB_OSC_ExistsOscSwitchInit function 4580
PLIB_OSC_ExistsPBClockDivisor function 4580
PLIB_OSC_ExistsPBClockOutputEnable function 4581
PLIB_OSC_ExistsPBClockReady function 4581
PLIB_OSC_ExistsPLLBypass function 4601
PLIB_OSC_ExistsPLLClockLock function 4582
PLIB_OSC_ExistsPLLLockStatus function 4583
PLIB_OSC_ExistsReferenceOscBaseClock function 4583
PLIB_OSC_ExistsReferenceOscChange function 4584
PLIB_OSC_ExistsReferenceOscChangeActive function 4584
PLIB_OSC_ExistsReferenceOscDivisor function 4585
PLIB_OSC_ExistsReferenceOscEnable function 4585
PLIB_OSC_ExistsReferenceOscStopInIdleEnable function 4586
PLIB_OSC_ExistsReferenceOscStopInSleep function 4586
PLIB_OSC_ExistsReferenceOscTrim function 4587
PLIB_OSC_ExistsReferenceOutputEnable function 4587
PLIB_OSC_ExistsResetPLL function 4601
PLIB_OSC_ExistsSecondaryEnable function 4588
PLIB_OSC_ExistsSecondaryReady function 4589
PLIB_OSC_ExistsSleepToStartupClock function 4593
PLIB_OSC_ExistsSlewDivisorStepControl function 4594
PLIB_OSC_ExistsSlewEnableControl function 4595
PLIB_OSC_ExistsSysPLLFrequencyRange function 4589
PLIB_OSC_ExistsSysPLLInputClockSource function 4590
PLIB_OSC_ExistsSysPLLInputDivisor function 4590
PLIB_OSC_ExistsSysPLLMultiplier function 4591
PLIB_OSC_ExistsSysPLLOutputDivisor function 4591
PLIB_OSC_ExistsSystemClockDivisorControl function 4595
PLIB_OSC_ExistsUPLLFrequencyRange function 4602
PLIB_OSC_ExistsUPLLInputDivisor function 4602
PLIB_OSC_ExistsUPLLMultiplier function 4603
PLIB_OSC_ExistsUPLLOutputDivisor function 4603
PLIB_OSC_ExistsUsbClockSource function 4592
PLIB_OSC_ForceSPLLLockDisable function 4561
PLIB_OSC_ForceSPLLLockEnable function 4561
PLIB_OSC_ForceSPLLLockStatus function 4562
PLIB_OSC_FRCDivisorGet function 4540
PLIB_OSC_FRCDivisorSelect function 4540
PLIB_OSC_FRCTuningSelect function 4541
plib_osc_help.h 4616
PLIB_OSC_OnWaitActionGet function 4517
PLIB_OSC_OnWaitActionSet function 4518
PLIB_OSC_PBClockDivisorGet function 4570
PLIB_OSC_PBClockDivisorIsReady function 4571
PLIB_OSC_PBClockDivisorSet function 4572
PLIB_OSC_PBOutputClockDisable function 4572
PLIB_OSC_PBOutputClockEnable function 4573
PLIB_OSC_PBOutputClockIsEnabled function 4574
PLIB_OSC_PLLBypassDisable function 4562
PLIB_OSC_PLLBypassEnable function 4563
PLIB_OSC_PLLBypassStatus function 4563
PLIB_OSC_PLLClockIsLocked function 4545
PLIB_OSC_PLLClockLock function 4546
PLIB_OSC_PLLClockUnlock function 4546
PLIB_OSC_PLLIsLocked function 4547
PLIB_OSC_ReferenceOscBaseClockSelect function 4529
PLIB_OSC_ReferenceOscDisable function 4530
PLIB_OSC_ReferenceOscDivisorValueSet function 4530
PLIB_OSC_ReferenceOscEnable function 4531
PLIB_OSC_ReferenceOscIsEnabled function 4532
PLIB_OSC_ReferenceOscSourceChangeIsActive function 4532
PLIB_OSC_ReferenceOscStopInIdleDisable function 4533
PLIB_OSC_ReferenceOscStopInIdleEnable function 4534
PLIB_OSC_ReferenceOscStopInIdleIsEnabled function 4534
PLIB_OSC_ReferenceOscStopInSleepDisable function 4535
PLIB_OSC_ReferenceOscStopInSleepEnable function 4535
PLIB_OSC_ReferenceOscStopInSleepIsEnabled function 4536
PLIB_OSC_ReferenceOscSwitchIsComplete function 4537
PLIB_OSC_ReferenceOscTrimSet function 4537
PLIB_OSC_ReferenceOutputDisable function 4538
PLIB_OSC_ReferenceOutputEnable function 4538
PLIB_OSC_ReferenceOutputIsEnabled function 4539
PLIB_OSC_ResetPLLAssert function 4569
PLIB_OSC_ResetPLLDeassert function 4569
PLIB_OSC_ResetPLLStatus function 4570
PLIB_OSC_SecondaryDisable function 4527
PLIB_OSC_SecondaryEnable function 4527
PLIB_OSC_SecondaryIsEnabled function 4528
PLIB_OSC_SecondaryIsReady function 4529
PLIB_OSC_SleepToStartupClockGet function 4521
PLIB_OSC_SleepToStartupClockSelect function 4522
PLIB_OSC_SlewDisable function 4518
PLIB_OSC_SlewDivisorStepGet function 4519
PLIB_OSC_SlewDivisorStepSelect function 4519
PLIB_OSC_SlewEnable function 4520
PLIB_OSC_SlewIsEnabled function 4521
PLIB_OSC_SysClockSelect function 4543
PLIB_OSC_SysPLLFrequencyRangeGet function 4547
PLIB_OSC_SysPLLFrequencyRangeSet function 4548
PLIB_OSC_SysPLLInputClockSourceGet function 4549
PLIB_OSC_SysPLLInputClockSourceSet function 4549
PLIB_OSC_SysPLLInputDivisorGet function 4550
PLIB_OSC_SysPLLInputDivisorSet function 4553
PLIB_OSC_SysPLLMultiplierGet function 4550
PLIB_OSC_SysPLLMultiplierSelect function 4551
PLIB_OSC_SysPLLOutputDivisorGet function 4551
PLIB_OSC_SysPLLOutputDivisorSet function 4552
PLIB_OSC_SystemClockDivisorGet function 4525
PLIB_OSC_SystemClockDivisorSelect function 4526
PLIB_OSC_UPLLFrequencyRangeGet function 4564

PLIB_OSC_UPLLFrequencyRangeSet function 4565
PLIB_OSC_UPLLInputDivisorGet function 4565
PLIB_OSC_UPLLInputDivisorSet function 4566
PLIB_OSC_UPLLMultiplierGet function 4566
PLIB_OSC_UPLLMultiplierSelect function 4567
PLIB_OSC_UPLLOutputDivisorGet function 4567
PLIB_OSC_UPLLOutputDivisorSet function 4568
PLIB_OSC_UsbClockSourceGet function 4544
PLIB_OSC_UsbClockSourceSelect function 4544
plib_pcache.h 4757
PLIB_PCACHE_CacheHitRead function 4733
PLIB_PCACHE_CacheHitWrite function 4733
PLIB_PCACHE_CacheLineAddrGet function 4721
PLIB_PCACHE_CacheLineAddrSet function 4722
PLIB_PCACHE_CacheLineData function 4722
PLIB_PCACHE_CacheLineDeselect function 4723
PLIB_PCACHE_CacheLineFlashTypeBoot function 4724
PLIB_PCACHE_CacheLineFlashTypeInst function 4724
PLIB_PCACHE_CacheLineFlashTypeIsInst function 4725
PLIB_PCACHE_CacheLineInst function 4725
PLIB_PCACHE_CacheLineInvalid function 4726
PLIB_PCACHE_CacheLineIsInst function 4726
PLIB_PCACHE_CacheLineIsLocked function 4727
PLIB_PCACHE_CacheLineIsValid function 4728
PLIB_PCACHE_CacheLineLock function 4728
PLIB_PCACHE_CacheLineMaskGet function 4729
PLIB_PCACHE_CacheLineMaskSet function 4729
PLIB_PCACHE_CacheLineSelect function 4730
PLIB_PCACHE_CacheLineUnlock function 4730
PLIB_PCACHE_CacheLineValid function 4731
PLIB_PCACHE_CacheMissRead function 4734
PLIB_PCACHE_CacheMissWrite function 4735
PLIB_PCACHE_DATA_ENABLE enumeration 4756
PLIB_PCACHE_DataCacheEnableGet function 4718
PLIB_PCACHE_DataCacheEnableSet function 4718
PLIB_PCACHE_ExistsCacheHit function 4744
PLIB_PCACHE_ExistsCacheLine function 4744
PLIB_PCACHE_ExistsCacheLineAddr function 4745
PLIB_PCACHE_ExistsCacheLineFlashType function 4745
PLIB_PCACHE_ExistsCacheLineLock function 4746
PLIB_PCACHE_ExistsCacheLineMask function 4746
PLIB_PCACHE_ExistsCacheLineType function 4747
PLIB_PCACHE_ExistsCacheLineValid function 4747
PLIB_PCACHE_ExistsCacheMiss function 4748
PLIB_PCACHE_ExistsDataCacheEnable function 4749
PLIB_PCACHE_ExistsFlashDEDStatus function 4749
PLIB_PCACHE_ExistsFlashSECCount function 4750
PLIB_PCACHE_ExistsFlashSECInt function 4750
PLIB_PCACHE_ExistsFlashSECStatus function 4751
PLIB_PCACHE_ExistsInvalidateOnPFMProgram function 4751
PLIB_PCACHE_ExistsLeastRecentlyUsedState function 4752
PLIB_PCACHE_ExistsPrefetchAbort function 4752
PLIB_PCACHE_ExistsPrefetchEnable function 4753
PLIB_PCACHE_ExistsWaitState function 4753
PLIB_PCACHE_ExistsWord function 4754
PLIB_PCACHE_FlashDEDStatusClear function 4738
PLIB_PCACHE_FlashDEDStatusGet function 4739
PLIB_PCACHE_FlashSECCountGet function 4739
PLIB_PCACHE_FlashSECCountSet function 4740
PLIB_PCACHE_FlashSECIntDisable function 4741
PLIB_PCACHE_FlashSECIntEnable function 4741
PLIB_PCACHE_FlashSECStatusClear function 4742
PLIB_PCACHE_FlashSECStatusGet function 4742
PLIB_PCACHE_FlashSECStatusSet function 4743
PLIB_PCACHE_InvalidateOnPFMProgramAll function 4719
PLIB_PCACHE_InvalidateOnPFMProgramUnlocked function 4719
PLIB_PCACHE_LeastRecentlyUsedStateRead function 4735
PLIB_PCACHE_MAX_SEC_COUNT macro 4755
PLIB_PCACHE_NUM_LINES macro 4755
PLIB_PCACHE_NUM_WORDS_PER_LINE macro 4755
PLIB_PCACHE_PREFETCH_ENABLE enumeration 4756
PLIB_PCACHE_PrefetchAbortRead function 4737
PLIB_PCACHE_PrefetchAbortWrite function 4738
PLIB_PCACHE_PrefetchEnableGet function 4736
PLIB_PCACHE_PrefetchEnableSet function 4736
PLIB_PCACHE_WaitStateGet function 4720
PLIB_PCACHE_WaitStateSet function 4721
PLIB_PCACHE_WordRead function 4732
PLIB_PCACHE_WordWrite function 4732
plib_pmp.h 4707
PLIB_PMP_AddressGet function 4664
PLIB_PMP_AddressIncrementModeGet function 4628
PLIB_PMP_AddressIncrementModeSelect function 4629
PLIB_PMP_AddressLatchPolaritySelect function 4670
PLIB_PMP_AddressLatchStrobeDisable function 4630
PLIB_PMP_AddressLatchStrobeEnable function 4630
PLIB_PMP_AddressLinesA0A1Get function 4664
PLIB_PMP_AddressLinesA0A1Set function 4665
PLIB_PMP_AddressPortDisable function 4666
PLIB_PMP_AddressPortEnable function 4667
PLIB_PMP_AddressSet function 4666
PLIB_PMP_ChipSelectFunctionSelect function 4631
PLIB_PMP_ChipSelectXDisable function 4632
PLIB_PMP_ChipSelectXEnable function 4632
PLIB_PMP_ChipSelectXIsActive function 4633
PLIB_PMP_ChipSelectXPolaritySelect function 4671
PLIB_PMP_DataSizeSelect function 4634
PLIB_PMP_Disable function 4634
PLIB_PMP_DualBufferDisable function 4644
PLIB_PMP_DualBufferEnable function 4644
PLIB_PMP_DualBufferIsEnabled function 4649
PLIB_PMP_DualModeMasterReceive function 4643
PLIB_PMP_DualModeMasterSend function 4645
PLIB_PMP_DualModeReadAddressGet function 4641
PLIB_PMP_DualModeReadAddressSet function 4641
PLIB_PMP_DualModeWriteAddressGet function 4642
PLIB_PMP_DualModeWriteAddressSet function 4642
PLIB_PMP_Enable function 4635
PLIB_PMP_ExistsAddressControl function 4672
PLIB_PMP_ExistsAddressLatchPolarity function 4673
PLIB_PMP_ExistsAddressLatchStrobePortControl function 4674
PLIB_PMP_ExistsAddressPortPinControl function 4674
PLIB_PMP_ExistsBufferOverFlow function 4675
PLIB_PMP_ExistsBufferRead function 4675
PLIB_PMP_ExistsBufferType function 4676
PLIB_PMP_ExistsBufferUnderFlow function 4676

PLIB_PMP_ExistsBufferWrite function 4677
PLIB_PMP_ExistsBusyStatus function 4677
PLIB_PMP_ExistsChipSelectEnable function 4678
PLIB_PMP_ExistsChipSelectoperation function 4678
PLIB_PMP_ExistsChipXPolarity function 4679
PLIB_PMP_ExistsCSXActiveStatus function 4680
PLIB_PMP_ExistsDataHoldWaitStates function 4680
PLIB_PMP_ExistsDataSetUpWaitStates function 4681
PLIB_PMP_ExistsDataStrobeWaitStates function 4681
PLIB_PMP_ExistsDataTransferSize function 4682
PLIB_PMP_ExistsDualBufferControl function 4691
PLIB_PMP_ExistsDualModeMasterRXTX function 4692
PLIB_PMP_ExistsDualModeReadAddressControl function 4693
PLIB_PMP_ExistsDualModeWriteAddressControl function 4693
PLIB_PMP_ExistsEnableControl function 4682
PLIB_PMP_ExistsIncrementMode function 4683
PLIB_PMP_ExistsInputBufferFull function 4683
PLIB_PMP_ExistsInputBufferXStatus function 4684
PLIB_PMP_ExistsInterruptMode function 4684
PLIB_PMP_ExistsMasterRXTX function 4685
PLIB_PMP_ExistsMUXModeSelect function 4685
PLIB_PMP_ExistsOperationMode function 4686
PLIB_PMP_ExistsOutPutBufferEmpty function 4687
PLIB_PMP_ExistsOutputBufferXStatus function 4687
PLIB_PMP_ExistsReadChipSelectEnable function 4694
PLIB_PMP_ExistsReadWritePolarity function 4688
PLIB_PMP_ExistsReadWriteStrobePortControl function 4688
PLIB_PMP_ExistsSlaveRX function 4689
PLIB_PMP_ExistsSlaveTX function 4689
PLIB_PMP_ExistsStartReadControl function 4695
PLIB_PMP_ExistsStopInIdleControl function 4690
PLIB_PMP_ExistsWriteChipSelectEnable function 4694
PLIB_PMP_ExistsWriteEnablePolarity function 4690
PLIB_PMP_ExistsWriteEnablePortControl function 4691
plib_pmp_help.h 4709
PLIB_PMP_InputBuffersAreFull function 4653
PLIB_PMP_InputBufferTypeSelect function 4635
PLIB_PMP_InputBufferXByteReceive function 4653
PLIB_PMP_InputBufferXIsFull function 4654
PLIB_PMP_InputOverflowClear function 4651
PLIB_PMP_InputOverflowHasOccurred function 4650
PLIB_PMP_InterruptModeGet function 4636
PLIB_PMP_InterruptModeSelect function 4637
PLIB_PMP_IsDataReceived function 4655
PLIB_PMP_IsDataTransmitted function 4655
PLIB_PMP_IsEnabled function 4648
PLIB_PMP_MasterReceive function 4656
PLIB_PMP_MasterSend function 4657
PLIB_PMP_MultiplexModeGet function 4637
PLIB_PMP_MultiplexModeSelect function 4638
PLIB_PMP_OperationModeGet function 4638
PLIB_PMP_OperationModeSelect function 4639
PLIB_PMP_OutputBuffersAreEmpty function 4657
PLIB_PMP_OutputBufferXByteSend function 4658
PLIB_PMP_OutputBufferXIsEmpty function 4659
PLIB_PMP_OutputUnderflowClear function 4652
PLIB_PMP_OutputUnderflowHasOccurred function 4651
PLIB_PMP_PortIsBusy function 4649
PLIB_PMP_ReadChipSelectXDisable function 4646
PLIB_PMP_ReadChipSelectXEnable function 4646
PLIB_PMP_ReadCycleIsStarted function 4661
PLIB_PMP_ReadCycleStart function 4661
PLIB_PMP_ReadWriteStrobePolaritySelect function 4671
PLIB_PMP_ReadWriteStrobePortDisable function 4668
PLIB_PMP_ReadWriteStrobePortEnable function 4668
PLIB_PMP_SlaveReceive function 4659
PLIB_PMP_SlaveSend function 4660
PLIB_PMP_StopInIdleDisable function 4640
PLIB_PMP_StopInIdleEnable function 4640
PLIB_PMP_WaitStatesDataHoldSelect function 4662
PLIB_PMP_WaitStatesDataSetUpSelect function 4662
PLIB_PMP_WaitStatesStrobeSelect function 4663
PLIB_PMP_WriteChipSelectXDisable function 4647
PLIB_PMP_WriteChipSelectXEnable function 4647
PLIB_PMP_WriteEnableStrobePolaritySelect function 4672
PLIB_PMP_WriteEnableStrobePortDisable function 4669
PLIB_PMP_WriteEnableStrobePortEnable function 4669
plib_ports.h 4845
PLIB_PORTS_AnPinsModeSelect function 4807
PLIB_PORTS_ChangeNoticeDisable function 4788
PLIB_PORTS_ChangeNoticeEnable function 4788
PLIB_PORTS_ChangeNoticeInIdleDisable function 4789
PLIB_PORTS_ChangeNoticeInIdleEnable function 4790
PLIB_PORTS_ChangeNoticeInIdlePerPortDisable function 4790
PLIB_PORTS_ChangeNoticeInIdlePerPortEnable function 4791
PLIB_PORTS_ChangeNoticePerPortHasOccured function 4791
PLIB_PORTS_ChangeNoticePerPortHasOccurred function 4799
PLIB_PORTS_ChangeNoticePerPortTurnOff function 4792
PLIB_PORTS_ChangeNoticePerPortTurnOn function 4792
PLIB_PORTS_ChangeNoticePullDownPerPortDisable function 4793
PLIB_PORTS_ChangeNoticePullDownPerPortEnable function 4794
PLIB_PORTS_ChangeNoticePullUpDisable function 4794
PLIB_PORTS_ChangeNoticePullUpEnable function 4795
PLIB_PORTS_ChangeNoticePullUpPerPortDisable function 4795
PLIB_PORTS_ChangeNoticePullUpPerPortEnable function 4796
PLIB_PORTS_ChannelChangeNoticeDisable function 4800
PLIB_PORTS_ChannelChangeNoticeEdgeDisable function 4804
PLIB_PORTS_ChannelChangeNoticeEdgeEnable function 4805
PLIB_PORTS_ChannelChangeNoticeEnable function 4801
PLIB_PORTS_ChannelChangeNoticeMethodGet function 4806
PLIB_PORTS_ChannelChangeNoticeMethodSelect function 4807
PLIB_PORTS_ChannelChangeNoticePullDownDisable function 4801
PLIB_PORTS_ChannelChangeNoticePullDownEnable function 4802
PLIB_PORTS_ChannelChangeNoticePullUpDisable function 4803
PLIB_PORTS_ChannelChangeNoticePullUpEnable function 4804
PLIB_PORTS_ChannelModeSelect function 4785
PLIB_PORTS_ChannelSlewRateSelect function 4785
PLIB_PORTS_Clear function 4778
PLIB_PORTS_CnPinsDisable function 4808
PLIB_PORTS_CnPinsEnable function 4809
PLIB_PORTS_CnPinsPullUpDisable function 4810
PLIB_PORTS_CnPinsPullUpEnable function 4810
PLIB_PORTS_DirectionGet function 4778
PLIB_PORTS_DirectionInputSet function 4779
PLIB_PORTS_DirectionOutputSet function 4780
PLIB_PORTS_ExistsAnPinsMode function 4822

PLIB_PORTS_ExistsChangeNotice function 4811
PLIB_PORTS_ExistsChangeNoticeEdgeControl function 4822
PLIB_PORTS_ExistsChangeNoticeEdgeStatus function 4823
PLIB_PORTS_ExistsChangeNoticeInIdle function 4812
PLIB_PORTS_ExistsChangeNoticePerPortInIdle function 4812
PLIB_PORTS_ExistsChangeNoticePerPortStatus function 4813
PLIB_PORTS_ExistsChangeNoticePerPortTurnOn function 4813
PLIB_PORTS_ExistsChangeNoticePullDownPerPort function 4814
PLIB_PORTS_ExistsChangeNoticePullUp function 4814
PLIB_PORTS_ExistsChangeNoticePullUpPerPort function 4815
PLIB_PORTS_ExistsChannelChangeNoticeMethod function 4823
PLIB_PORTS_ExistsLatchRead function 4821
PLIB_PORTS_ExistsPinChangeNotice function 4815
PLIB_PORTS_ExistsPinChangeNoticePerPort function 4816
PLIB_PORTS_ExistsPinMode function 4817
PLIB_PORTS_ExistsPinModePerPort function 4817
PLIB_PORTS_ExistsPortsDirection function 4818
PLIB_PORTS_ExistsPortsOpenDrain function 4818
PLIB_PORTS_ExistsPortsRead function 4819
PLIB_PORTS_ExistsPortsWrite function 4820
PLIB_PORTS_ExistsRemapInput function 4820
PLIB_PORTS_ExistsRemapOutput function 4821
PLIB_PORTS_ExistsSlewRateControl function 4824
PLIB_PORTS_OpenDrainDisable function 4783
PLIB_PORTS_OpenDrainEnable function 4783
PLIB_PORTS_PinChangeNoticeDisable function 4797
PLIB_PORTS_PinChangeNoticeEdgeHasOccurred function 4774
PLIB_PORTS_PinChangeNoticeEdgeIsEnabled function 4775
PLIB_PORTS_PinChangeNoticeEnable function 4797
PLIB_PORTS_PinChangeNoticePerPortDisable function 4798
PLIB_PORTS_PinChangeNoticePerPortEnable function 4799
PLIB_PORTS_PinClear function 4768
PLIB_PORTS_PinDirectionInputSet function 4768
PLIB_PORTS_PinDirectionOutputSet function 4769
PLIB_PORTS_PinGet function 4770
PLIB_PORTS_PinGetLatched function 4774
PLIB_PORTS_PinModePerPortSelect function 4773
PLIB_PORTS_PinModeSelect function 4770
PLIB_PORTS_PinOpenDrainDisable function 4777
PLIB_PORTS_PinOpenDrainEnable function 4777
PLIB_PORTS_PinSet function 4771
PLIB_PORTS_PinSlewRateGet function 4776
PLIB_PORTS_PinToggle function 4772
PLIB_PORTS_PinWrite function 4772
PLIB_PORTS_Read function 4780
PLIB_PORTS_ReadLatched function 4784
PLIB_PORTS_RemapInput function 4786
PLIB_PORTS_RemapOutput function 4787
PLIB_PORTS_Set function 4781
PLIB_PORTS_Toggle function 4782
PLIB_PORTS_Write function 4782
plib_power.h 4893
PLIB_POWER_ClearIdleStatus function 4862
PLIB_POWER_ClearSleepStatus function 4862
PLIB_POWER_DeepSleepEventStatusClear function 4866
PLIB_POWER_DeepSleepEventStatusGet function 4867
PLIB_POWER_DeepSleepGPRRead function 4874
PLIB_POWER_DeepSleepGPRsRetentionDisable function 4857
PLIB_POWER_DeepSleepGPRsRetentionEnable function 4857
PLIB_POWER_DeepSleepGPRWrite function 4874
PLIB_POWER_DeepSleepModeDisable function 4852
PLIB_POWER_DeepSleepModeEnable function 4853
PLIB_POWER_DeepSleepModeHasOccurred function 4867
PLIB_POWER_DeepSleepModeIsEnabled function 4858
PLIB_POWER_DeepSleepModuleDisable function 4858
PLIB_POWER_DeepSleepModuleEnable function 4859
PLIB_POWER_DeepSleepPortPinsStateRelease function 4859
PLIB_POWER_DeepSleepPortPinsStateRetain function 4860
PLIB_POWER_DeepSleepStatusClear function 4865
PLIB_POWER_DeepSleepWakeupEventDisable function 4861
PLIB_POWER_DeepSleepWakeupEventEnable function 4861
PLIB_POWER_DeviceWasInIdleMode function 4863
PLIB_POWER_DeviceWasInSleepMode function 4864
PLIB_POWER_ExistsDeepSleepEventStatus function 4878
PLIB_POWER_ExistsDeepSleepGPROperation function 4879
PLIB_POWER_ExistsDeepSleepGPRsRetentionControl function 4879
PLIB_POWER_ExistsDeepSleepMode function 4875
PLIB_POWER_ExistsDeepSleepModeOccurrence function 4881
PLIB_POWER_ExistsDeepSleepModuleControl function 4880
PLIB_POWER_ExistsDeepSleepPortPinsStateControl function 4880
PLIB_POWER_ExistsDeepSleepWakeupEventControl function 4882
PLIB_POWER_ExistsHighVoltageOnVDD1V8 function 4881
PLIB_POWER_ExistsHLVDBandGapVoltageStability function 4882
PLIB_POWER_ExistsHLVDEnableControl function 4883
PLIB_POWER_ExistsHLVDLimitSelection function 4884
PLIB_POWER_ExistsHLVDModeControl function 4884
PLIB_POWER_ExistsHLVDStatus function 4885
PLIB_POWER_ExistsHLVDStopInIdleControl function 4885
PLIB_POWER_ExistsIdleStatus function 4876
PLIB_POWER_ExistsPeripheralModuleControl function 4876
PLIB_POWER_ExistsSleepStatus function 4877
PLIB_POWER_ExistsVoltageRegulatorControl function 4877
PLIB_POWER_HighVoltageOnVDD1V8HasOccurred function 4864
PLIB_POWER_HLVDBandGapVoltageIsStable function 4868
PLIB_POWER_HLVDDisable function 4868
PLIB_POWER_HLVDEnable function 4869
PLIB_POWER_HLVDIsEnabled function 4870
PLIB_POWER_HLVDLimitSelect function 4870
PLIB_POWER_HLVDModeSelect function 4871
PLIB_POWER_HLVDStatusGet function 4871
PLIB_POWER_HLVDStopInIdleDisable function 4872
PLIB_POWER_HLVDStopInIdleEnable function 4872
PLIB_POWER_HLVDStopInIdleIsEnabled function 4873
PLIB_POWER_PeripheralModuleDisable function 4853
PLIB_POWER_PeripheralModuleEnable function 4854
PLIB_POWER_PeripheralModuleIsEnabled function 4855
PLIB_POWER_VoltageRegulatorDisable function 4855
PLIB_POWER_VoltageRegulatorEnable function 4856
PLIB_POWER_VoltageRegulatorIsEnabled function 4865
plib_reset.h 4910
PLIB_RESET_ConfigRegReadErrorGet function 4899
PLIB_RESET_ExistsConfigRegReadError function 4905
PLIB_RESET_ExistsNmiControl function 4905
PLIB_RESET_ExistsNmiCounter function 4906
PLIB_RESET_ExistsResetReasonStatus function 4904
PLIB_RESET_ExistsSoftwareResetTrigger function 4904

PLIB_RESET_ExistsWdtoInSleep function 4906
PLIB_RESET_NmiCounterValueGet function 4901
PLIB_RESET_NmiCounterValueSet function 4901
PLIB_RESET_NmiEventClear function 4902
PLIB_RESET_NmiEventTrigger function 4902
PLIB_RESET_NmiReasonGet function 4903
PLIB_RESET_ReasonClear function 4898
PLIB_RESET_ReasonGet function 4899
PLIB_RESET_SoftwareResetEnable function 4897
PLIB_RESET_WdtTimeOutHasOccurredInSleep function 4900
plib_rtcc.h 4968
PLIB_RTCC_AlarmChimeDisable function 4935
PLIB_RTCC_AlarmChimeEnable function 4936
PLIB_RTCC_AlarmDateGet function 4936
PLIB_RTCC_AlarmDateSet function 4937
PLIB_RTCC_AlarmDayGet function 4937
PLIB_RTCC_AlarmDaySet function 4938
PLIB_RTCC_AlarmDisable function 4939
PLIB_RTCC_AlarmEnable function 4939
PLIB_RTCC_AlarmHourGet function 4940
PLIB_RTCC_AlarmHourSet function 4941
PLIB_RTCC_AlarmMaskModeSelect function 4941
PLIB_RTCC_AlarmMinuteGet function 4942
PLIB_RTCC_AlarmMinuteSet function 4942
PLIB_RTCC_AlarmMonthGet function 4943
PLIB_RTCC_AlarmMonthSet function 4944
PLIB_RTCC_AlarmPulseInitialGet function 4944
PLIB_RTCC_AlarmPulseInitialSet function 4945
PLIB_RTCC_AlarmRepeatCountGet function 4945
PLIB_RTCC_AlarmRepeatCountSet function 4946
PLIB_RTCC_AlarmSecondGet function 4947
PLIB_RTCC_AlarmSecondSet function 4947
PLIB_RTCC_AlarmSyncStatusGet function 4951
PLIB_RTCC_AlarmTimeGet function 4948
PLIB_RTCC_AlarmTimeSet function 4948
PLIB_RTCC_AlarmValueRegisterPointer function 4949
PLIB_RTCC_AlarmWeekDayGet function 4950
PLIB_RTCC_AlarmWeekDaySet function 4950
PLIB_RTCC_ClockOutputDisable function 4920
PLIB_RTCC_ClockOutputEnable function 4920
PLIB_RTCC_ClockRunningStatus function 4921
PLIB_RTCC_ClockSourceSelect function 4951
PLIB_RTCC_Disable function 4922
PLIB_RTCC_DriftCalibrateGet function 4952
PLIB_RTCC_DriftCalibrateSet function 4953
PLIB_RTCC_Enable function 4922
PLIB_RTCC_ExistsAlarmChimeControl function 4956
PLIB_RTCC_ExistsAlarmControl function 4956
PLIB_RTCC_ExistsAlarmDate function 4957
PLIB_RTCC_ExistsAlarmMaskControl function 4957
PLIB_RTCC_ExistsAlarmPulseInitial function 4958
PLIB_RTCC_ExistsAlarmRepeatControl function 4958
PLIB_RTCC_ExistsAlarmSynchronization function 4966
PLIB_RTCC_ExistsAlarmTime function 4959
PLIB_RTCC_ExistsCalibration function 4959
PLIB_RTCC_ExistsClockRunning function 4960
PLIB_RTCC_ExistsClockSelect function 4960
PLIB_RTCC_ExistsEnableControl function 4961
PLIB_RTCC_ExistsHalfSecond function 4962
PLIB_RTCC_ExistsOutputControl function 4962
PLIB_RTCC_ExistsOutputSelect function 4963
PLIB_RTCC_ExistsRTCDate function 4963
PLIB_RTCC_ExistsRTCTime function 4964
PLIB_RTCC_ExistsStopInIdleControl function 4964
PLIB_RTCC_ExistsSynchronization function 4965
PLIB_RTCC_ExistsWriteEnable function 4965
PLIB_RTCC_HalfSecondStatusGet function 4953
plib_rtcc_help.h 4970
PLIB_RTCC_OutputSelect function 4954
PLIB_RTCC_RTCDateGet function 4924
PLIB_RTCC_RTCDateSet function 4924
PLIB_RTCC_RTCDayGet function 4925
PLIB_RTCC_RTCDaySet function 4926
PLIB_RTCC_RTCHourGet function 4926
PLIB_RTCC_RTCHourSet function 4927
PLIB_RTCC_RTCMinuteGet function 4927
PLIB_RTCC_RTCMinuteSet function 4928
PLIB_RTCC_RTCMonthGet function 4929
PLIB_RTCC_RTCMonthSet function 4929
PLIB_RTCC_RTCSecondGet function 4930
PLIB_RTCC_RTCSecondSet function 4930
PLIB_RTCC_RTCSyncStatusGet function 4934
PLIB_RTCC_RTCTimeGet function 4931
PLIB_RTCC_RTCTimeSet function 4931
PLIB_RTCC_RTCWeekDayGet function 4932
PLIB_RTCC_RTCWeekDaySet function 4933
PLIB_RTCC_RTCYearGet function 4933
PLIB_RTCC_RTCYearSet function 4934
PLIB_RTCC_StopInIdleDisable function 4954
PLIB_RTCC_StopInIdleEnable function 4955
PLIB_RTCC_WriteDisable function 4923
PLIB_RTCC_WriteEnable function 4923
plib_sb.h 5009
PLIB_SB_ARB_POLICY enumeration 5004
PLIB_SB_CPUPrioritySet function 4991
PLIB_SB_DMAPrioritySet function 4991
PLIB_SB_ERROR enumeration 5004
PLIB_SB_ExistsCPUPriority function 4993
PLIB_SB_ExistsDMAPriority function 4993
PLIB_SB_ExistsInitPermGrp function 4994
PLIB_SB_ExistsPGRegAddr function 4994
PLIB_SB_ExistsPGRegRdPerm function 4995
PLIB_SB_ExistsPGRegSize function 4995
PLIB_SB_ExistsPGRegWrPerm function 4996
PLIB_SB_ExistsPGVErrClear function 4996
PLIB_SB_ExistsPGVErrClrMulti function 4997
PLIB_SB_ExistsPGVErrClrSingle function 4998
PLIB_SB_ExistsPGVErrCmdCode function 4998
PLIB_SB_ExistsPGVErrGroup0Status function 5001
PLIB_SB_ExistsPGVErrGroup1Status function 5002
PLIB_SB_ExistsPGVErrGroup2Status function 5002
PLIB_SB_ExistsPGVErrGroup3Status function 5003
PLIB_SB_ExistsPGVErrGroupStatus function 5003
PLIB_SB_ExistsPGVErrInitID function 4999
PLIB_SB_ExistsPGVErrPG function 4999
PLIB_SB_ExistsPGVErrRegion function 5000

PLIB_SB_ExistsPGVErrRptPri function 5000
PLIB_SB_ExistsPGVErrStatus function 5001
PLIB_SB_INIT_ID enumeration 5005
PLIB_SB_INIT_PG enumeration 5005
PLIB_SB_InitPermGrpSet function 4992
PLIB_SB_OCP_CMD_CODE enumeration 5006
PLIB_SB_PG_INITIATOR enumeration 5006
PLIB_SB_PGRegionAddrGet function 4975
PLIB_SB_PGRegionAddrSet function 4976
PLIB_SB_PGRegionReadPermClear function 4976
PLIB_SB_PGRegionReadPermSet function 4977
PLIB_SB_PGRegionSizeGet function 4978
PLIB_SB_PGRegionSizeSet function 4978
PLIB_SB_PGRegionWritePermClear function 4979
PLIB_SB_PGRegionWritePermSet function 4980
PLIB_SB_PGVErrGroup0Status function 4988
PLIB_SB_PGVErrGroup1Status function 4988
PLIB_SB_PGVErrGroup2Status function 4989
PLIB_SB_PGVErrGroup3Status function 4989
PLIB_SB_PGVErrGroupStatus function 4990
PLIB_SB_PGVErrorClearMulti function 4980
PLIB_SB_PGVErrorClearSingle function 4981
PLIB_SB_PGVErrorCode function 4981
PLIB_SB_PGVErrorCommandCode function 4982
PLIB_SB_PGVErrorInitiatorID function 4983
PLIB_SB_PGVErrorLogClearMulti function 4983
PLIB_SB_PGVErrorLogClearSingle function 4984
PLIB_SB_PGVErrorMulti function 4984
PLIB_SB_PGVErrorPermissionGroup function 4985
PLIB_SB_PGVErrorRegion function 4985
PLIB_SB_PGVErrorReportPrimaryDisable function 4986
PLIB_SB_PGVErrorReportPrimaryEnable function 4987
PLIB_SB_PGVErrorStatus function 4987
PLIB_SB_REGION_PG enumeration 5006
PLIB_SB_TGT_ID enumeration 5007
PLIB_SB_TGT_REGION enumeration 5008
plib_spi.h 5102
PLIB_SPI_AudioCommunicationWidthSelect function 5063
PLIB_SPI_AudioErrorDisable function 5064
PLIB_SPI_AudioErrorEnable function 5064
PLIB_SPI_AudioProtocolDisable function 5065
PLIB_SPI_AudioProtocolEnable function 5066
PLIB_SPI_AudioProtocolModeSelect function 5066
PLIB_SPI_AudioTransmitModeSelect function 5067
PLIB_SPI_BaudRateClockSelect function 5037
PLIB_SPI_BaudRateSet function 5038
PLIB_SPI_BufferAddressGet function 5054
PLIB_SPI_BufferClear function 5053
PLIB_SPI_BufferRead function 5053
PLIB_SPI_BufferRead16bit function 5054
PLIB_SPI_BufferRead32bit function 5055
PLIB_SPI_BufferWrite function 5056
PLIB_SPI_BufferWrite16bit function 5056
PLIB_SPI_BufferWrite32bit function 5057
PLIB_SPI_ClockPolaritySelect function 5039
PLIB_SPI_CommunicationWidthSelect function 5039
PLIB_SPI_Disable function 5040
PLIB_SPI_Enable function 5040
PLIB_SPI_ErrorInterruptDisable function 5041
PLIB_SPI_ErrorInterruptEnable function 5042
PLIB_SPI_Exists16bitBuffer function 5093
PLIB_SPI_Exists32bitBuffer function 5093
PLIB_SPI_ExistsAudioCommunicationWidth function 5073
PLIB_SPI_ExistsAudioErrorControl function 5073
PLIB_SPI_ExistsAudioProtocolControl function 5074
PLIB_SPI_ExistsAudioProtocolMode function 5074
PLIB_SPI_ExistsAudioTransmitMode function 5075
PLIB_SPI_ExistsBaudRate function 5075
PLIB_SPI_ExistsBaudRateClock function 5076
PLIB_SPI_ExistsBuffer function 5076
PLIB_SPI_ExistsBusStatus function 5077
PLIB_SPI_ExistsClockPolarity function 5077
PLIB_SPI_ExistsCommunicationWidth function 5078
PLIB_SPI_ExistsEnableControl function 5079
PLIB_SPI_ExistsErrorInterruptControl function 5079
PLIB_SPI_ExistsFIFOControl function 5080
PLIB_SPI_ExistsFIFOCount function 5080
PLIB_SPI_ExistsFIFOInterruptMode function 5081
PLIB_SPI_ExistsFIFOShiftRegisterEmptyStatus function 5081
PLIB_SPI_ExistsFramedCommunication function 5082
PLIB_SPI_ExistsFrameErrorStatus function 5082
PLIB_SPI_ExistsFrameSyncPulseCounter function 5083
PLIB_SPI_ExistsFrameSyncPulseDirection function 5083
PLIB_SPI_ExistsFrameSyncPulseEdge function 5084
PLIB_SPI_ExistsFrameSyncPulsePolarity function 5085
PLIB_SPI_ExistsFrameSyncPulseWidth function 5085
PLIB_SPI_ExistsInputSamplePhase function 5086
PLIB_SPI_ExistsMasterControl function 5086
PLIB_SPI_ExistsOutputDataPhase function 5087
PLIB_SPI_ExistsPinControl function 5087
PLIB_SPI_ExistsReadDataSignStatus function 5088
PLIB_SPI_ExistsReceiveBufferStatus function 5088
PLIB_SPI_ExistsReceiveFIFOStatus function 5089
PLIB_SPI_ExistsReceiverOverflow function 5089
PLIB_SPI_ExistsSlaveSelectControl function 5090
PLIB_SPI_ExistsStopInIdleControl function 5090
PLIB_SPI_ExistsTransmitBufferEmptyStatus function 5091
PLIB_SPI_ExistsTransmitBufferFullStatus function 5092
PLIB_SPI_ExistsTransmitUnderRunStatus function 5092
PLIB_SPI_FIFOCountGet function 5042
PLIB_SPI_FIFODisable function 5043
PLIB_SPI_FIFOEnable function 5044
PLIB_SPI_FIFOInterruptModeSelect function 5044
PLIB_SPI_FIFOShiftRegisterIsEmpty function 5045
PLIB_SPI_FramedCommunicationDisable function 5057
PLIB_SPI_FramedCommunicationEnable function 5058
PLIB_SPI_FrameErrorStatusClear function 5059
PLIB_SPI_FrameErrorStatusGet function 5059
PLIB_SPI_FrameSyncPulseCounterSelect function 5060
PLIB_SPI_FrameSyncPulseDirectionSelect function 5060
PLIB_SPI_FrameSyncPulseEdgeSelect function 5061
PLIB_SPI_FrameSyncPulsePolaritySelect function 5062
PLIB_SPI_FrameSyncPulseWidthSelect function 5062
PLIB_SPI_InputSamplePhaseSelect function 5045
PLIB_SPI_IsBusy function 5046
PLIB_SPI_MasterEnable function 5047

PLIB_SPI_OutputDataPhaseSelect function 5047
PLIB_SPI_PinDisable function 5048
PLIB_SPI_PinEnable function 5048
PLIB_SPI_ReadDataIsSignExtended function 5049
PLIB_SPI_ReceiverBufferIsFull function 5070
PLIB_SPI_ReceiverFIFOIsEmpty function 5071
PLIB_SPI_ReceiverHasOverflowed function 5071
PLIB_SPI_ReceiverOverflowClear function 5072
PLIB_SPI_SlaveEnable function 5050
PLIB_SPI_SlaveSelectDisable function 5050
PLIB_SPI_SlaveSelectEnable function 5051
PLIB_SPI_StopInIdleDisable function 5051
PLIB_SPI_StopInIdleEnable function 5052
PLIB_SPI_TransmitBufferIsEmpty function 5067
PLIB_SPI_TransmitBufferIsFull function 5068
PLIB_SPI_TransmitUnderRunStatusClear function 5069
PLIB_SPI_TransmitUnderRunStatusGet function 5069
plib_sqi.h 5238
PLIB_SQI_BurstEnable function 5119
PLIB_SQI_ByteCountGet function 5157
PLIB_SQI_ByteCountSet function 5157
PLIB_SQI_ChipSelectDeassertDisable function 5158
PLIB_SQI_ChipSelectDeassertEnable function 5159
PLIB_SQI_ChipSelectGet function 5159
PLIB_SQI_ChipSelectSet function 5160
PLIB_SQI_ClockDisable function 5120
PLIB_SQI_ClockDividerSet function 5140
PLIB_SQI_ClockEnable function 5140
PLIB_SQI_ClockIsStable function 5141
PLIB_SQI_CommandStatusGet function 5174
PLIB_SQI_ConBufferSoftReset function 5136
PLIB_SQI_ConBufWordsAvailable function 5175
PLIB_SQI_ConfigWordGet function 5160
PLIB_SQI_ConfigWordSet function 5161
PLIB_SQI_ControlBufferThresholdGet function 5120
PLIB_SQI_ControlBufferThresholdSet function 5121
PLIB_SQI_ControlWordGet function 5121
PLIB_SQI_ControlWordSet function 5122
PLIB_SQI_CSOutputEnableSelect function 5123
PLIB_SQI_DataFormatGet function 5123
PLIB_SQI_DataFormatSet function 5124
PLIB_SQI_DataLineStatus function 5174
PLIB_SQI_DataModeSet function 5125
PLIB_SQI_DataOutputEnableSelect function 5125
PLIB_SQI_DDRModeGet function 5139
PLIB_SQI_DDRModeSet function 5139
PLIB_SQI_Disable function 5126
PLIB_SQI_DMABDBaseAddressGet function 5176
PLIB_SQI_DMABDBaseAddressSet function 5176
PLIB_SQI_DMABDControlWordGet function 5177
PLIB_SQI_DMABDCurrentAddressGet function 5177
PLIB_SQI_DMABDFetchStart function 5182
PLIB_SQI_DMABDFetchStop function 5182
PLIB_SQI_DMABDIsBusy function 5183
PLIB_SQI_DMABDPollCounterSet function 5184
PLIB_SQI_DMABDPollDisable function 5184
PLIB_SQI_DMABDPollEnable function 5185
PLIB_SQI_DMABDPollIsEnabled function 5185
PLIB_SQI_DMABDReceiveBufferCountGet function 5178
PLIB_SQI_DMABDReceiveBufferLengthGet function 5179
PLIB_SQI_DMABDReceiveStateGet function 5179
PLIB_SQI_DMABDStateGet function 5186
PLIB_SQI_DMABDTransmitBufferCountGet function 5180
PLIB_SQI_DMABDTransmitBufferLengthGet function 5181
PLIB_SQI_DMABDTransmitStateGet function 5181
PLIB_SQI_DMADisable function 5187
PLIB_SQI_DMAEnable function 5187
PLIB_SQI_DMAHasStarted function 5188
PLIB_SQI_DMAIsEnabled function 5188
PLIB_SQI_Enable function 5126
PLIB_SQI_ExistsBDBaseAddress function 5190
PLIB_SQI_ExistsBDControlWord function 5190
PLIB_SQI_ExistsBDCurrentAddress function 5191
PLIB_SQI_ExistsBDPollCount function 5191
PLIB_SQI_ExistsBDPollingEnable function 5192
PLIB_SQI_ExistsBDProcessState function 5193
PLIB_SQI_ExistsBDRxBufCount function 5193
PLIB_SQI_ExistsBDRxLength function 5194
PLIB_SQI_ExistsBDRxState function 5194
PLIB_SQI_ExistsBDTxBufCount function 5195
PLIB_SQI_ExistsBDTxLength function 5195
PLIB_SQI_ExistsBDTxState function 5196
PLIB_SQI_ExistsBurstControl function 5196
PLIB_SQI_ExistsChipSelect function 5197
PLIB_SQI_ExistsClockControl function 5197
PLIB_SQI_ExistsClockDivider function 5198
PLIB_SQI_ExistsClockReady function 5198
PLIB_SQI_ExistsCommandStatus function 5222
PLIB_SQI_ExistsConBufAvailable function 5223
PLIB_SQI_ExistsConBufferSoftReset function 5223
PLIB_SQI_ExistsConBufThreshold function 5199
PLIB_SQI_ExistsConfigWord function 5199
PLIB_SQI_ExistsControlWord function 5200
PLIB_SQI_ExistsCSDeassert function 5201
PLIB_SQI_ExistsCSOutputEnable function 5201
PLIB_SQI_ExistsDataFormat function 5202
PLIB_SQI_ExistsDataModeControl function 5202
PLIB_SQI_ExistsDataOutputEnable function 5203
PLIB_SQI_ExistsDataPinStatus function 5203
PLIB_SQI_ExistsDDRMode function 5224
PLIB_SQI_ExistsDMABDFetch function 5225
PLIB_SQI_ExistsDmaEnable function 5204
PLIB_SQI_ExistsDMAEngineBusy function 5204
PLIB_SQI_ExistsDMAProcessInProgress function 5205
PLIB_SQI_ExistsEnableControl function 5205
PLIB_SQI_ExistsHoldPinControl function 5206
PLIB_SQI_ExistsInterruptControl function 5207
PLIB_SQI_ExistsInterruptSignalControl function 5207
PLIB_SQI_ExistsInterruptStatus function 5208
PLIB_SQI_ExistsLaneMode function 5208
PLIB_SQI_ExistsReceiveLatch function 5209
PLIB_SQI_ExistsRxBufferCount function 5209
PLIB_SQI_ExistsRxBufferSoftReset function 5225
PLIB_SQI_ExistsRxBufIntThreshold function 5210
PLIB_SQI_ExistsRxBufThreshold function 5210
PLIB_SQI_ExistsRxData function 5211

PLIB_SQI_ExistsRxUnderRun function 5211
PLIB_SQI_ExistsSoftReset function 5212
PLIB_SQI_ExistsStatusCheck function 5226
PLIB_SQI_ExistsTapDelay function 5226
PLIB_SQI_ExistsTransferCommand function 5213
PLIB_SQI_ExistsTransferCount function 5213
PLIB_SQI_ExistsTransferModeControl function 5214
PLIB_SQI_ExistsTxBufferFree function 5214
PLIB_SQI_ExistsTxBufferSoftReset function 5227
PLIB_SQI_ExistsTxBufIntThreshold function 5215
PLIB_SQI_ExistsTxBufThreshold function 5215
PLIB_SQI_ExistsTxData function 5216
PLIB_SQI_ExistsTxOverFlow function 5216
PLIB_SQI_ExistsWPPinControl function 5217
PLIB_SQI_ExistsXIPChipSelect function 5217
PLIB_SQI_ExistsXIPControlWord1 function 5218
PLIB_SQI_ExistsXIPControlWord2 function 5219
PLIB_SQI_ExistsXIPControlWord3 function 5227
PLIB_SQI_ExistsXIPControlWord4 function 5228
PLIB_SQI_ExistsXIPDDRMode function 5228
PLIB_SQI_ExistsXIPLaneMode function 5219
PLIB_SQI_ExistsXIPModeBytes function 5220
PLIB_SQI_ExistsXIPModeCode function 5220
PLIB_SQI_ExistsXIPNumberOfAddressBytes function 5221
PLIB_SQI_ExistsXIPNumberOfDummyBytes function 5221
PLIB_SQI_ExistsXIPReadOpCode function 5222
PLIB_SQI_ExistsXIPSDRCommandDDRData function 5229
PLIB_SQI_HoldClear function 5127
PLIB_SQI_HoldGet function 5127
PLIB_SQI_HoldSet function 5128
PLIB_SQI_InterruptDisable function 5169
PLIB_SQI_InterruptEnable function 5170
PLIB_SQI_InterruptFlagGet function 5171
PLIB_SQI_InterruptIsEnabled function 5171
PLIB_SQI_InterruptSignalDisable function 5172
PLIB_SQI_InterruptSignalEnable function 5173
PLIB_SQI_InterruptSignalIsEnabled function 5173
PLIB_SQI_LaneModeGet function 5128
PLIB_SQI_LaneModeSet function 5129
PLIB_SQI_NumberOfReceiveBufferReads function 5130
PLIB_SQI_ReceiveBufferIsUnderrun function 5162
PLIB_SQI_ReceiveData function 5130
PLIB_SQI_ReceiveLatchDisable function 5131
PLIB_SQI_ReceiveLatchEnable function 5131
PLIB_SQI_ReceiveLatchGet function 5132
PLIB_SQI_RxBufferSoftReset function 5137
PLIB_SQI_RxBufferThresholdGet function 5163
PLIB_SQI_RxBufferThresholdIntGet function 5163
PLIB_SQI_RxBufferThresholdIntSet function 5164
PLIB_SQI_RxBufferThresholdSet function 5164
PLIB_SQI_SoftReset function 5132
PLIB_SQI_StatusCheckSet function 5189
PLIB_SQI_TapDelaySet function 5138
PLIB_SQI_TransferDirectionGet function 5165
PLIB_SQI_TransferDirectionSet function 5165
PLIB_SQI_TransferModeGet function 5133
PLIB_SQI_TransferModeSet function 5133
PLIB_SQI_TransmitBufferFreeSpaceGet function 5166
PLIB_SQI_TransmitBufferHasOverflowed function 5167
PLIB_SQI_TransmitData function 5134
PLIB_SQI_TxBufferSoftReset function 5137
PLIB_SQI_TxBufferThresholdGet function 5167
PLIB_SQI_TxBufferThresholdIntGet function 5168
PLIB_SQI_TxBufferThresholdIntSet function 5168
PLIB_SQI_TxBufferThresholdSet function 5169
PLIB_SQI_WriteProtectClear function 5135
PLIB_SQI_WriteProtectGet function 5135
PLIB_SQI_WriteProtectSet function 5136
PLIB_SQI_XIPAddressBytesGet function 5142
PLIB_SQI_XIPAddressBytesSet function 5142
PLIB_SQI_XIPChipSelectGet function 5143
PLIB_SQI_XIPChipSelectSet function 5143
PLIB_SQI_XIPControlWord1Get function 5144
PLIB_SQI_XIPControlWord1Set function 5144
PLIB_SQI_XIPControlWord2Get function 5145
PLIB_SQI_XIPControlWord2Set function 5146
PLIB_SQI_XIPControlWord3Get function 5152
PLIB_SQI_XIPControlWord3Set function 5153
PLIB_SQI_XIPControlWord4Get function 5154
PLIB_SQI_XIPControlWord4Set function 5156
PLIB_SQI_XIPDDRModeSet function 5154
PLIB_SQI_XIPDummyBytesGet function 5147
PLIB_SQI_XIPDummyBytesSet function 5147
PLIB_SQI_XIPLaneModeSet function 5148
PLIB_SQI_XIPModeBytesGet function 5149
PLIB_SQI_XIPModeBytesSet function 5149
PLIB_SQI_XIPModeCodeGet function 5150
PLIB_SQI_XIPModeCodeSet function 5150
PLIB_SQI_XIPReadOpcodeGet function 5151
PLIB_SQI_XIPReadOpcodeSet function 5152
PLIB_SQI_XIPSDRCommandDDRDataGet function 5155
PLIB_SQI_XIPSDRCommandDDRDataSet function 5155
plib_tmr.h 5279
PLIB_TMR_ClockSourceExternalSyncDisable function 5255
PLIB_TMR_ClockSourceExternalSyncEnable function 5256
PLIB_TMR_ClockSourceSelect function 5256
PLIB_TMR_Counter16BitClear function 5263
PLIB_TMR_Counter16BitGet function 5263
PLIB_TMR_Counter16BitSet function 5264
PLIB_TMR_Counter32BitClear function 5264
PLIB_TMR_Counter32BitGet function 5265
PLIB_TMR_Counter32BitSet function 5266
PLIB_TMR_CounterAsyncWriteDisable function 5266
PLIB_TMR_CounterAsyncWriteEnable function 5267
PLIB_TMR_CounterAsyncWriteInProgress function 5267
PLIB_TMR_ExistsClockSource function 5269
PLIB_TMR_ExistsClockSourceSync function 5269
PLIB_TMR_ExistsCounter16Bit function 5270
PLIB_TMR_ExistsCounter32Bit function 5270
PLIB_TMR_ExistsCounterAsyncWriteControl function 5271
PLIB_TMR_ExistsCounterAsyncWriteInProgress function 5271
PLIB_TMR_ExistsEnableControl function 5272
PLIB_TMR_ExistsGatedTimeAccumulation function 5273
PLIB_TMR_ExistsMode16Bit function 5273
PLIB_TMR_ExistsMode32Bit function 5274
PLIB_TMR_ExistsPeriod16Bit function 5274

PLIB_TMR_ExistsPeriod32Bit function 5275
PLIB_TMR_ExistsPrescale function 5275
PLIB_TMR_ExistsStopInIdleControl function 5276
PLIB_TMR_ExistsTimerOperationMode function 5276
PLIB_TMR_GateDisable function 5257
PLIB_TMR_GateEnable function 5257
PLIB_TMR_IsPeriodMatchBased function 5268
PLIB_TMR_Mode16BitEnable function 5251
PLIB_TMR_Mode32BitEnable function 5252
PLIB_TMR_Period16BitGet function 5260
PLIB_TMR_Period16BitSet function 5261
PLIB_TMR_Period32BitGet function 5261
PLIB_TMR_Period32BitSet function 5262
PLIB_TMR_PrescaleDivisorGet function 5258
PLIB_TMR_PrescaleGet function 5259
PLIB_TMR_PrescaleSelect function 5259
PLIB_TMR_Start function 5253
PLIB_TMR_Stop function 5253
PLIB_TMR_StopInIdleDisable function 5254
PLIB_TMR_StopInIdleEnable function 5254
plib_usart.h 5364
PLIB_USART_AddressGet function 5308
PLIB_USART_AddressMaskGet function 5308
PLIB_USART_AddressMaskSet function 5309
PLIB_USART_AddressSet function 5310
PLIB_USART_BaudRateAutoDetectEnable function 5315
PLIB_USART_BaudRateAutoDetectIsComplete function 5315
PLIB_USART_BaudRateGet function 5316
PLIB_USART_BaudRateHighDisable function 5317
PLIB_USART_BaudRateHighEnable function 5317
PLIB_USART_BaudRateHighSet function 5318
PLIB_USART_BaudRateSet function 5319
PLIB_USART_BaudSetAndEnable function 5319
PLIB_USART_BRGClockSourceGet function 5320
PLIB_USART_BRGClockSourceSelect function 5321
PLIB_USART_Disable function 5297
PLIB_USART_Enable function 5297
PLIB_USART_ErrorsGet function 5305
PLIB_USART_ExistsBaudRate function 5341
PLIB_USART_ExistsBaudRateAutoDetect function 5341
PLIB_USART_ExistsBaudRateHigh function 5342
PLIB_USART_ExistsBRGClockSourceSelect function 5358
PLIB_USART_ExistsEnable function 5342
PLIB_USART_ExistsHandshakeMode function 5343
PLIB_USART_ExistsIrDA function 5343
PLIB_USART_ExistsLineControlMode function 5344
PLIB_USART_ExistsLoopback function 5345
PLIB_USART_ExistsModuleBusyStatus function 5358
PLIB_USART_ExistsOperationMode function 5345
PLIB_USART_ExistsReceiver function 5346
PLIB_USART_ExistsReceiver9Bits function 5357
PLIB_USART_ExistsReceiverAddress function 5359
PLIB_USART_ExistsReceiverAddressAutoDetect function 5346
PLIB_USART_ExistsReceiverAddressDetect function 5347
PLIB_USART_ExistsReceiverAddressMask function 5359
PLIB_USART_ExistsReceiverDataAvailableStatus function 5347
PLIB_USART_ExistsReceiverEnable function 5348
PLIB_USART_ExistsReceiverFramingErrorStatus function 5348
PLIB_USART_ExistsReceiverIdleStateLowEnable function 5349
PLIB_USART_ExistsReceiverIdleStatus function 5349
PLIB_USART_ExistsReceiverInterruptMode function 5350
PLIB_USART_ExistsReceiverOverrunStatus function 5351
PLIB_USART_ExistsReceiverParityErrorStatus function 5351
PLIB_USART_ExistsRunInOverflow function 5360
PLIB_USART_ExistsRunInSleepMode function 5360
PLIB_USART_ExistsStopInIdle function 5352
PLIB_USART_ExistsTransmitter function 5352
PLIB_USART_ExistsTransmitter9BitsSend function 5353
PLIB_USART_ExistsTransmitterBreak function 5353
PLIB_USART_ExistsTransmitterBufferFullStatus function 5354
PLIB_USART_ExistsTransmitterEmptyStatus function 5354
PLIB_USART_ExistsTransmitterEnable function 5355
PLIB_USART_ExistsTransmitterIdleIsLow function 5355
PLIB_USART_ExistsTransmitterInterruptMode function 5356
PLIB_USART_ExistsWakeOnStart function 5357
PLIB_USART_HandshakeModeSelect function 5298
plib_usart_help.h 5367
PLIB_USART_InitializeModeGeneral function 5306
PLIB_USART_InitializeOperation function 5307
PLIB_USART_IrDADisable function 5299
PLIB_USART_IrDAEnable function 5299
PLIB_USART_LineControlModeSelect function 5300
PLIB_USART_LoopbackDisable function 5300
PLIB_USART_LoopbackEnable function 5301
PLIB_USART_ModuleIsBusy function 5310
PLIB_USART_OperationModeSelect function 5301
PLIB_USART_Receiver9BitsReceive function 5340
PLIB_USART_ReceiverAddressAutoDetectDisable function 5329
PLIB_USART_ReceiverAddressAutoDetectEnable function 5330
PLIB_USART_ReceiverAddressDetectDisable function 5330
PLIB_USART_ReceiverAddressDetectEnable function 5331
PLIB_USART_ReceiverAddressGet function 5340
PLIB_USART_ReceiverAddressIsReceived function 5331
PLIB_USART_ReceiverByteReceive function 5332
PLIB_USART_ReceiverDataIsAvailable function 5333
PLIB_USART_ReceiverDisable function 5334
PLIB_USART_ReceiverEnable function 5334
PLIB_USART_ReceiverFramingErrorHasOccurred function 5335
PLIB_USART_ReceiverIdleStateLowDisable function 5335
PLIB_USART_ReceiverIdleStateLowEnable function 5336
PLIB_USART_ReceiverInterruptModeSelect function 5336
PLIB_USART_ReceiverIsIdle function 5337
PLIB_USART_ReceiverOverrunErrorClear function 5338
PLIB_USART_ReceiverOverrunHasOccurred function 5338
PLIB_USART_ReceiverParityErrorHasOccurred function 5339
PLIB_USART_RunInOverflowDisable function 5311
PLIB_USART_RunInOverflowEnable function 5312
PLIB_USART_RunInOverflowIsEnabled function 5312
PLIB_USART_RunInSleepModeDisable function 5313
PLIB_USART_RunInSleepModeEnable function 5314
PLIB_USART_RunInSleepModeIsEnabled function 5314
PLIB_USART_StopInIdleDisable function 5302
PLIB_USART_StopInIdleEnable function 5303
PLIB_USART_Transmitter9BitsSend function 5321
PLIB_USART_TransmitterAddressGet function 5328
PLIB_USART_TransmitterBreakSend function 5322

PLIB_USART_TransmitterBreakSendIsComplete function 5323
PLIB_USART_TransmitterBufferIsFull function 5323
PLIB_USART_TransmitterByteSend function 5324
PLIB_USART_TransmitterDisable function 5325
PLIB_USART_TransmitterEnable function 5325
PLIB_USART_TransmitterIdleIsLowDisable function 5326
PLIB_USART_TransmitterIdleIsLowEnable function 5327
PLIB_USART_TransmitterInterruptModeSelect function 5327
PLIB_USART_TransmitterIsEmpty function 5328
PLIB_USART_WakeOnStartDisable function 5303
PLIB_USART_WakeOnStartEnable function 5304
PLIB_USART_WakeOnStartIsEnabled function 5304
plib_usb.h 5511
PLIB_USB_ActivityPending function 5456
PLIB_USB_AllInterruptEnable function 5381
PLIB_USB_AutoSuspendDisable function 5382
PLIB_USB_AutoSuspendEnable function 5382
PLIB_USB_BDTBaseAddressGet function 5394
PLIB_USB_BDTBaseAddressSet function 5395
PLIB_USB_BufferAddressGet function 5395
PLIB_USB_BufferAddressSet function 5396
PLIB_USB_BufferAllCancelReleaseToUSB function 5397
PLIB_USB_BufferByteCountGet function 5397
PLIB_USB_BufferByteCountSet function 5398
PLIB_USB_BufferCancelReleaseToUSB function 5399
PLIB_USB_BufferClearAll function 5400
PLIB_USB_BufferClearAllDTSEnable function 5400
PLIB_USB_BufferDataToggleGet function 5401
PLIB_USB_BufferDataToggleSelect function 5402
PLIB_USB_BufferDataToggleSyncDisable function 5403
PLIB_USB_BufferDataToggleSyncEnable function 5403
PLIB_USB_BufferEP0RxStatusInitialize function 5404
PLIB_USB_BufferIndexGet function 5405
PLIB_USB_BufferPIDBitsClear function 5406
PLIB_USB_BufferPIDGet function 5406
PLIB_USB_BufferReleasedToSW function 5407
PLIB_USB_BufferReleaseToUSB function 5408
PLIB_USB_BufferSchedule function 5409
PLIB_USB_BufferStallDisable function 5409
PLIB_USB_BufferStallEnable function 5410
PLIB_USB_BufferStallGet function 5411
PLIB_USB_DeviceAddressGet function 5383
PLIB_USB_DeviceAddressSet function 5383
PLIB_USB_Disable function 5384
PLIB_USB_Enable function 5385
PLIB_USB_EP0HostSetup function 5412
PLIB_USB_EP0LSDirectConnectDisable function 5412
PLIB_USB_EP0LSDirectConnectEnable function 5413
PLIB_USB_EP0NakRetryDisable function 5413
PLIB_USB_EP0NakRetryEnable function 5414
PLIB_USB_EPnAttributesClear function 5414
PLIB_USB_EPnAttributesSet function 5415
PLIB_USB_EPnControlTransferDisable function 5416
PLIB_USB_EPnControlTransferEnable function 5416
PLIB_USB_EPnDirectionDisable function 5417
PLIB_USB_EPnHandshakeDisable function 5417
PLIB_USB_EPnHandshakeEnable function 5418
PLIB_USB_EPnIsStalled function 5419
PLIB_USB_EPnRxDisable function 5419
PLIB_USB_EPnRxEnable function 5420
PLIB_USB_EPnRxSelect function 5420
PLIB_USB_EPnStallClear function 5421
PLIB_USB_EPnTxDisable function 5422
PLIB_USB_EPnTxEnable function 5422
PLIB_USB_EPnTxRxSelect function 5423
PLIB_USB_EPnTxSelect function 5423
PLIB_USB_ErrorInterruptDisable function 5429
PLIB_USB_ErrorInterruptEnable function 5430
PLIB_USB_ErrorInterruptFlagAllGet function 5430
PLIB_USB_ErrorInterruptFlagClear function 5431
PLIB_USB_ErrorInterruptFlagGet function 5431
PLIB_USB_ErrorInterruptFlagSet function 5432
PLIB_USB_ErrorInterruptIsEnabled function 5433
PLIB_USB_ExistsActivityPending function 5475
PLIB_USB_ExistsALL_Interrupt function 5476
PLIB_USB_ExistsAutomaticSuspend function 5476
PLIB_USB_ExistsBDTBaseAddress function 5477
PLIB_USB_ExistsBDTFunctions function 5477
PLIB_USB_ExistsBufferFreeze function 5478
PLIB_USB_ExistsDeviceAddress function 5479
PLIB_USB_ExistsEP0LowSpeedConnect function 5479
PLIB_USB_ExistsEP0NAKRetry function 5480
PLIB_USB_ExistsEPnRxEnable function 5480
PLIB_USB_ExistsEPnTxRx function 5481
PLIB_USB_ExistsERR_Interrupt function 5482
PLIB_USB_ExistsERR_InterruptStatus function 5482
PLIB_USB_ExistsEyePattern function 5483
PLIB_USB_ExistsFrameNumber function 5483
PLIB_USB_ExistsGEN_Interrupt function 5484
PLIB_USB_ExistsGEN_InterruptStatus function 5484
PLIB_USB_ExistsHostBusyWithToken function 5485
PLIB_USB_ExistsHostGeneratesReset function 5486
PLIB_USB_ExistsLastDirection function 5486
PLIB_USB_ExistsLastEndpoint function 5487
PLIB_USB_ExistsLastPingPong function 5487
PLIB_USB_ExistsLastTransactionDetails function 5488
PLIB_USB_ExistsLiveJState function 5488
PLIB_USB_ExistsLiveSingleEndedZero function 5489
PLIB_USB_ExistsModuleBusy function 5489
PLIB_USB_ExistsModulePower function 5490
PLIB_USB_ExistsNextTokenSpeed function 5490
PLIB_USB_ExistsOnChipPullup function 5491
PLIB_USB_ExistsOnChipTransceiver function 5491
PLIB_USB_ExistsOpModeSelect function 5492
PLIB_USB_ExistsOTG_ASessionValid function 5493
PLIB_USB_ExistsOTG_BSessionEnd function 5493
PLIB_USB_ExistsOTG_IDPinState function 5494
PLIB_USB_ExistsOTG_Interrupt function 5494
PLIB_USB_ExistsOTG_InterruptStatus function 5495
PLIB_USB_ExistsOTG_LineState function 5495
PLIB_USB_ExistsOTG_PullUpPullDown function 5496
PLIB_USB_ExistsOTG_SessionValid function 5496
PLIB_USB_ExistsOTG_VbusCharge function 5497
PLIB_USB_ExistsOTG_VbusDischarge function 5497
PLIB_USB_ExistsOTG_VbusPowerOnOff function 5498
PLIB_USB_ExistsPacketTransfer function 5498

PLIB_USB_ExistsPingPongMode function 5499
PLIB_USB_ExistsResumeSignaling function 5500
PLIB_USB_ExistsSleepEntryGuard function 5500
PLIB_USB_ExistsSOFThreshold function 5501
PLIB_USB_ExistsSpeedControl function 5501
PLIB_USB_ExistsStartOfFrames function 5502
PLIB_USB_ExistsStopInIdle function 5502
PLIB_USB_ExistsSuspend function 5503
PLIB_USB_ExistsTokenEP function 5503
PLIB_USB_ExistsTokenPID function 5504
PLIB_USB_ExistsUOEMonitor function 5505
PLIB_USB_ExternalComparatorMode2Pin function 5463
PLIB_USB_ExternalComparatorMode3Pin function 5464
PLIB_USB_EyePatternDisable function 5471
PLIB_USB_EyePatternEnable function 5472
PLIB_USB_FrameNumberGet function 5457
PLIB_USB_FullSpeedDisable function 5385
PLIB_USB_FullSpeedEnable function 5386
PLIB_USB_I2CInterfaceForExtModuleDisable function 5461
PLIB_USB_I2CInterfaceForExtModuleEnable function 5462
PLIB_USB_InterruptDisable function 5424
PLIB_USB_InterruptEnable function 5425
PLIB_USB_InterruptEnableGet function 5425
PLIB_USB_InterruptFlagAllGet function 5426
PLIB_USB_InterruptFlagClear function 5426
PLIB_USB_InterruptFlagGet function 5427
PLIB_USB_InterruptFlagSet function 5428
PLIB_USB_InterruptIsEnabled function 5428
PLIB_USB_IsBusyWithToken function 5436
PLIB_USB_JStateIsActive function 5457
PLIB_USB_LastTransactionDetailsGet function 5433
PLIB_USB_LastTransactionDirectionGet function 5434
PLIB_USB_LastTransactionEndPtGet function 5435
PLIB_USB_LastTransactionPingPongStateGet function 5435
PLIB_USB_ModuleIsBusy function 5472
PLIB_USB_OnChipPullUpDisable function 5387
PLIB_USB_OnChipPullUpEnable function 5387
PLIB_USB_OperatingModeSelect function 5388
PLIB_USB_OTG_BSessionHasEnded function 5444
PLIB_USB_OTG_IDPinStateIsTypeA function 5445
PLIB_USB_OTG_InterruptDisable function 5452
PLIB_USB_OTG_InterruptEnable function 5453
PLIB_USB_OTG_InterruptFlagClear function 5454
PLIB_USB_OTG_InterruptFlagGet function 5454
PLIB_USB_OTG_InterruptFlagSet function 5455
PLIB_USB_OTG_InterruptIsEnabled function 5456
PLIB_USB_OTG_LineStateIsStable function 5446
PLIB_USB_OTG_PullUpPullDownSetup function 5446
PLIB_USB_OTG_SessionValid function 5447
PLIB_USB_OTG_VBusChargeDisable function 5447
PLIB_USB_OTG_VBusChargeEnable function 5448
PLIB_USB_OTG_VBusChargeTo3V function 5449
PLIB_USB_OTG_VBusChargeTo5V function 5449
PLIB_USB_OTG_VBusDischargeDisable function 5450
PLIB_USB_OTG_VBusDischargeEnable function 5450
PLIB_USB_OTG_VBusPowerOff function 5451
PLIB_USB_OTG_VBusPowerOn function 5451
PLIB_USB_OTG_VBusValid function 5452
PLIB_USB_PacketTransferDisable function 5458
PLIB_USB_PacketTransferEnable function 5459
PLIB_USB_PacketTransferIsDisabled function 5460
PLIB_USB_PingPongFreeze function 5473
PLIB_USB_PingPongModeGet function 5388
PLIB_USB_PingPongModeSelect function 5389
PLIB_USB_PingPongReset function 5473
PLIB_USB_PingPongUnfreeze function 5474
PLIB_USB_PWMCounterDisable function 5464
PLIB_USB_PWMCounterEnable function 5465
PLIB_USB_PWMDisable function 5465
PLIB_USB_PWMEnable function 5466
PLIB_USB_PWMPolaritiyActiveLow function 5467
PLIB_USB_PWMPolarityActiveHigh function 5467
PLIB_USB_ResetSignalDisable function 5442
PLIB_USB_ResetSignalEnable function 5442
PLIB_USB_ResumeSignalingDisable function 5443
PLIB_USB_ResumeSignalingEnable function 5444
PLIB_USB_SE0InProgress function 5460
PLIB_USB_SleepGuardDisable function 5389
PLIB_USB_SleepGuardEnable function 5390
PLIB_USB_SOFDisable function 5437
PLIB_USB_SOFEnable function 5437
PLIB_USB_SOFThresholdGet function 5438
PLIB_USB_SOFThresholdSet function 5438
PLIB_USB_StopInIdleDisable function 5391
PLIB_USB_StopInIdleEnable function 5391
PLIB_USB_SuspendDisable function 5392
PLIB_USB_SuspendEnable function 5392
PLIB_USB_TokenEPGet function 5439
PLIB_USB_TokenEPSet function 5440
PLIB_USB_TokenPIDGet function 5440
PLIB_USB_TokenPIDSet function 5441
PLIB_USB_TokenSend function 5474
PLIB_USB_TokenSpeedSelect function 5441
PLIB_USB_TransceiverDisable function 5462
PLIB_USB_TransceiverEnable function 5463
PLIB_USB_UOEMonitorDisable function 5393
PLIB_USB_UOEMonitorEnable function 5393
PLIB_USB_VBoostDisable function 5468
PLIB_USB_VBoostEnable function 5468
PLIB_USB_VBUSComparatorDisable function 5469
PLIB_USB_VBUSComparatorEnable function 5469
PLIB_USB_VBUSPullUpDisable function 5470
PLIB_USB_VBUSPullUpEnable function 5470
plib_usbhs.h 5561
PLIB_USBHS_DeviceAddressGet function 5543
PLIB_USBHS_DeviceAddressSet function 5543
PLIB_USBHS_DeviceAttach function 5544
PLIB_USBHS_DeviceConnect function 5544
PLIB_USBHS_DeviceDetach function 5544
PLIB_USBHS_DeviceDisconnect function 5545
PLIB_USBHS_DeviceEPFIFOLoad function 5545
PLIB_USBHS_DeviceEPFIFOUnload function 5545
PLIB_USBHS_DeviceRxEndpointConfigure function 5546
PLIB_USBHS_DeviceRxEndpointStallDisable function 5546
PLIB_USBHS_DeviceRxEndpointStallEnable function 5546
PLIB_USBHS_DeviceTxEndpointConfigure function 5546

PLIB_USBHS_DeviceTxEndpointPacketReady function 5546
PLIB_USBHS_DeviceTxEndpointStallDisable function 5547
PLIB_USBHS_DeviceTxEndpointStallEnable function 5547
PLIB_USBHS_DMAErrorGet function 5549
PLIB_USBHS_DMAInterruptDisable function 5537
PLIB_USBHS_DMAInterruptEnable function 5538
PLIB_USBHS_DMAInterruptFlagsGet function 5538
PLIB_USBHS_DMAInterruptGet function 5542
PLIB_USBHS_DMAOperationEnable function 5524
PLIB_USBHS_Endpoint0FIFOFlush function 5525
PLIB_USBHS_Endpoint0SetupPacketLoad function 5526
PLIB_USBHS_Endpoint0SetupPacketUnload function 5526
PLIB_USBHS_EndpointFIFOLoad function 5527
PLIB_USBHS_EndpointFIFOUnload function 5527
PLIB_USBHS_EndpointRxFIFOFlush function 5528
PLIB_USBHS_EndpointRxRequestClear function 5528
PLIB_USBHS_EndpointRxRequestEnable function 5528
PLIB_USBHS_EndpointTxFIFOFlush function 5528
PLIB_USBHS_EP0DataEndSet function 5529
PLIB_USBHS_EP0INHandshakeClear function 5529
PLIB_USBHS_EP0INHandshakeSend function 5529
PLIB_USBHS_EP0INTokenSend function 5529
PLIB_USBHS_EP0OUTHandshakeSend function 5529
PLIB_USBHS_EP0RxPktRdyServiced function 5530
PLIB_USBHS_EP0RxPktRdyServicedDataEnd function 5530
PLIB_USBHS_EP0SentStallClear function 5530
PLIB_USBHS_EP0SetupEndServiced function 5530
PLIB_USBHS_EP0StallDisable function 5530
PLIB_USBHS_EP0StallEnable function 5531
PLIB_USBHS_EP0StatusClear function 5531
PLIB_USBHS_EP0StatusGet function 5531
PLIB_USBHS_EP0TxPktRdy function 5531
PLIB_USBHS_EP0TxPktRdyDataEnd function 5531
PLIB_USBHS_ExistsClockResetControl function 5552
PLIB_USBHS_ExistsEndpointFIFO function 5550
PLIB_USBHS_ExistsEndpointOperations function 5550
PLIB_USBHS_ExistsEP0Status function 5551
PLIB_USBHS_ExistsHighSpeedSupport function 5551
PLIB_USBHS_ExistsInterrupts function 5551
PLIB_USBHS_ExistsModuleControl function 5551
PLIB_USBHS_ExistsRxEPStatus function 5551
PLIB_USBHS_ExistsSoftReset function 5552
PLIB_USBHS_ExistsTxEPStatus function 5552
PLIB_USBHS_ExistsUSBIDControl function 5552
PLIB_USBHS_FullOrHighSpeedIsConnected function 5547
PLIB_USBHS_GenInterruptDisable function 5535
PLIB_USBHS_GenInterruptEnable function 5535
PLIB_USBHS_GenInterruptFlagsGet function 5536
PLIB_USBHS_GetEP0CSRAddress function 5533
PLIB_USBHS_GetEP0FIFOAddress function 5533
PLIB_USBHS_GetReceiveDataCount function 5549
PLIB_USBHS_GlobalInterruptDisable function 5542
PLIB_USBHS_GlobalInterruptEnable function 5542
PLIB_USBHS_HighSpeedDisable function 5520
PLIB_USBHS_HighSpeedEnable function 5520
PLIB_USBHS_HighSpeedIsConnected function 5548
PLIB_USBHS_HostModeIsEnabled function 5548
PLIB_USBHS_HostRxEndpointConfigure function 5532
PLIB_USBHS_HostRxEndpointDataToggleClear function 5547
PLIB_USBHS_HostTxEndpointConfigure function 5532
PLIB_USBHS_HostTxEndpointDataToggleClear function 5547
PLIB_USBHS_InterruptEnableSet function 5537
PLIB_USBHS_IsBDevice function 5525
PLIB_USBHS_LoadEPInIndex function 5533
PLIB_USBHS_ModuleSpeedGet function 5548
PLIB_USBHS_PhyIDMonitoringDisable function 5524
PLIB_USBHS_PhyIDMonitoringEnable function 5524
PLIB_USBHS_ResetDisable function 5521
PLIB_USBHS_ResetEnable function 5521
PLIB_USBHS_ResumeDisable function 5521
PLIB_USBHS_ResumeEnable function 5522
PLIB_USBHS_RxEPINTokenSend function 5532
PLIB_USBHS_RxEPStatusClear function 5532
PLIB_USBHS_RxEPStatusGet function 5532
PLIB_USBHS_RxInterruptDisable function 5540
PLIB_USBHS_RxInterruptEnable function 5541
PLIB_USBHS_RxInterruptFlagsGet function 5541
PLIB_USBHS_SessionDisable function 5522
PLIB_USBHS_SessionEnable function 5522
PLIB_USBHS_SoftResetDisable function 5522
PLIB_USBHS_SoftResetEnable function 5523
PLIB_USBHS_SuspendDisable function 5523
PLIB_USBHS_SuspendEnable function 5523
PLIB_USBHS_TestModeEnter function 5524
PLIB_USBHS_TestModeExit function 5524
PLIB_USBHS_TxEPStatusClear function 5533
PLIB_USBHS_TxEPStatusGet function 5533
PLIB_USBHS_TxInterruptDisable function 5534
PLIB_USBHS_TxInterruptEnable function 5534
PLIB_USBHS_TxInterruptFlagsGet function 5539
PLIB_USBHS_USBIDOverrideDisable function 5525
PLIB_USBHS_USBIDOverrideEnable function 5525
PLIB_USBHS_USBIDOverrideValueSet function 5525
PLIB_USBHS_VbusLevelGet function 5549
PLIB_USBHS_VBUSLevelGet function 5550
plib_wdt.h 5575
PLIB_WDT_Disable function 5567
PLIB_WDT_Enable function 5567
PLIB_WDT_ExistsEnableControl function 5572
PLIB_WDT_ExistsPostscalerValue function 5572
PLIB_WDT_ExistsSleepModePostscalerValue function 5574
PLIB_WDT_ExistsTimerClear function 5573
PLIB_WDT_ExistsWindowEnable function 5573
plib_wdt_help.h 5576
PLIB_WDT_IsEnabled function 5570
PLIB_WDT_PostscalerValueGet function 5570
PLIB_WDT_SleepModePostscalerValueGet function 5571
PLIB_WDT_TimerClear function 5569
PLIB_WDT_WindowDisable function 5568
PLIB_WDT_WindowEnable function 5568
PMP Peripheral Library 4616
PMP_ACK_MODE enumeration 4695
PMP_ADDRESS_HOLD_LATCH_WAIT_STATES enumeration 4696
PMP_ADDRESS_LATCH enumeration 4696
PMP_ADDRESS_LATCH_WAIT_STATES enumeration 4697
PMP_ADDRESS_PORT enumeration 4697

PMP_ALTERNATE_MASTER_WAIT_STATES enumeration 4698
PMP_CHIP_SELECT enumeration 4699
PMP_CHIPSELECT_FUNCTION enumeration 4699
PMP_DATA_HOLD_STATES enumeration 4700
PMP_DATA_LENGTH enumeration 4700
PMP_DATA_SIZE enumeration 4701
PMP_DATA_WAIT_STATES enumeration 4701
PMP_INCREMENT_MODE enumeration 4701
PMP_INPUT_BUFFER_TYPE enumeration 4702
PMP_INTERRUPT_MODE enumeration 4702
PMP_MASTER_MODE enumeration 4703
PMP_MODULE_ID enumeration 4703
PMP_MUX_MODE enumeration 4704
PMP_OPERATION_MODE enumeration 4704
PMP_PMBE_PORT enumeration 4705
PMP_POLARITY_LEVEL enumeration 4705
PMP_QUEUE_ELEMENT_OBJECT structure 1893
PMP_STROBE_WAIT_STATES enumeration 4706
Polygon Fill Rendering Functions 2602
Polygon Rendering 2568
Polygon Rendering Functions 2599
Porting Applications from the MLA Graphics Library to the MPLAB Harmony Graphics Library 2556
Porting Applications from the V6 Beta TCP/IP Stack to the MPLAB Harmony TCP/IP Stack 5995
Porting Procedure 3225
Porting the MPLAB Harmony Drivers and Peripheral Library 6002
Ports Change Notification 4764
Ports Control 4763, 5890
Ports Function Remap 4764
Ports Peripheral Library 4759
Ports System Service Library 5885
PORTS_AN_PIN enumeration 4841
PORTS_ANALOG_PIN enumeration 4825
PORTS_BIT_POS enumeration 4826
PORTS_CHANGE_NOTICE_EDGE enumeration 4843
PORTS_CHANGE_NOTICE_METHOD enumeration 4844
PORTS_CHANGE_NOTICE_PIN enumeration 4827
PORTS_CHANNEL enumeration 4828
PORTS_CN_PIN enumeration 4842
PORTS_DATA_MASK type 4829
PORTS_DATA_TYPE type 4829
PORTS_MODULE_ID enumeration 4830
PORTS_PIN_MODE enumeration 4830
PORTS_PIN_SLEW_RATE enumeration 4844
PORTS_REMAP_FUNCTION enumeration 4830
PORTS_REMAP_INPUT_FUNCTION enumeration 4833
PORTS_REMAP_INPUT_PIN enumeration 4834
PORTS_REMAP_OUTPUT_FUNCTION enumeration 4835
PORTS_REMAP_OUTPUT_PIN enumeration 4836
PORTS_REMAP_PIN enumeration 4837
Power Peripheral Library 4847
POWER_MODULE enumeration 4888
POWER_MODULE_ID enumeration 4891
Power-Saving Modes 3246, 3617, 5033
pProgressFnc type 1314
Prefetch Cache Peripheral Library 4710
Prefetch Control Operations 4714
Prefetch Status Operations 4715
Prerequisites 6711
PROCESS_CONFIG_REQ macro 43
PROCESS_CONFIG_RES macro 44
PROCESS_CONN_REQ macro 44
PROCESS_CONN_RES macro 44
PROCESS_DCONN_REQ macro 44
PROCESS_DCONN_RES macro 44
PROCESS_INFO_REQ macro 45
PROCESS_INFO_RES macro 45
Progress Bar Object 2802
progressDcpt structure 1315
PS_DEFAULT macro 932
PS_F macro 932
PS_I macro 932
PS_RAM macro 932
PS_ROM macro 932
PSKEY_ANA_FREQ macro 933
PSKEY_BDADDR macro 933
PSKEY_BLE_DEFAULT_TX_POWER macro 960
PSKEY_CLOCK_REQUEST_ENABLE macro 933
PSKEY_DEEP_SLEEP_CLEAR_RTS macro 933
PSKEY_DEEP_SLEEP_EXTERNAL_CLOCK_SOURCE_PIO macro 961
PSKEY_DEEP_SLEEP_STATE macro 933
PSKEY_DEEP_SLEEP_USE_EXTERNAL_CLOCK macro 934
PSKEY_DEEP_SLEEP_WAKE_CTS macro 934
PSKEY_DIGITAL_AUDIO_BITS_PER_SAMPLE macro 934
PSKEY_DIGITAL_AUDIO_CONFIG macro 934
PSKEY_H_HC_FC_MAX_SCO_PKT_LEN macro 961
PSKEY_H_HC_FC_MAX_SCO_PKTS macro 961
PSKEY_HCI_NOP_DISABLE macro 934
PSKEY_HOST_INTERFACE macro 935
PSKEY_HOSTIO_MAP_SCO_PCM macro 935
PSKEY_HOSTIO_UART_RESET_TIMEOUT macro 935
PSKEY_LC_DEFAULT_TX_POWER macro 965
PSKEY_LC_MAX_TX_POWER macro 966
PSKEY_LC_MAX_TX_POWER_NO_RSSI macro 966
PSKEY_PCM_CLOCK_RATE macro 961
PSKEY_PCM_CONFIG32 macro 961
PSKEY_PCM_FORMAT macro 962
PSKEY_PCM_PULL_CONTROL macro 935
PSKEY_PCM_SYNC_RATE macro 962
PSKEY_PCM_USE_LOW_JITTER_MODE macro 962
PSKEY_PIO_DEEP_SLEEP_EITHER_LEVEL macro 935
PSKEY_UART_BAUDRATE macro 936
PSKEY_UART_BITRATE macro 936
PSKEY_UART_CONFIG_BCSP macro 936
PSKEY_UART_CONFIG_H4 macro 936
PSKEY_UART_CONFIG_H4DS macro 962
PSKEY_UART_CONFIG_H5 macro 936
PSKEY_UART_TX_WINDOW_SIZE macro 937
PSKEY_VM_DISABLE macro 937
PSM_ATT macro 758
PSM_AVCTP macro 758
PSM_AVCTP_Browsing macro 759
PSM_AVDTP macro 759
PSM_BNEP macro 759
PSM_HID_Control macro 759

PSM_HID_Interrupt macro 759
PSM_RFCOMM macro 760
PSM_SDP macro 760
PWM Mode with Enabled Faults 4484

Q

q15 type 3108
q31 type 3108
q63 type 3109
queue.h 1141

R

Radio Button Object 2807
Random Number Generator (RNG) System Service Library 5915
read_command function 786
READ_CONFIG_REQUEST macro 45
READ_CONFIG_RESPONSE macro 45
READ_CONN_REQUEST macro 45
READ_CONN_RESPONSE macro 46
READ_DCONN_REQUEST macro 46
READ_DCONN_RESPONSE macro 46
READ_INFO_REQUEST macro 46
READ_INFO_RESPONSE macro 46
Reading a Capture Value 4343
Reading a File 5766
Reading and Writing Data 7110
Reboot Module 6479
Receive 3992
Receive Filtering Overview 3985
Receiving a CAN Message 3523
Receiving a Report 6904
Receiving Data 6868
recv function 6037
recvfrom function 6037
RED macro 2718
Reduced Media Independent Interface (RMII) 3984
Rendering 1675
Rendering Style Functions 2620
Reset Peripheral Library 4895
Reset Reason Management 5927
Reset System Service Library 5926
RESET_CONFIG_REG_READ_ERROR enumeration 4908
RESET_MODULE_ID enumeration 4907
RESET_NMI_COUNT_TYPE type 4909
RESET_NMI_REASON enumeration 4909
RESET_REASON enumeration 4907
rfcomm.h 1142
RFCOMM_ALLOCATE_BUFFERS function 18
RFCOMM_ALLOCATE_BUFFERS_FUNCTION macro 48
RFCOMM_ALLOCATE_BUFFERS_RAM_SIZE_VAR macro 48
RFCOMM_ALLOCATE_BUFFERS_VARS macro 48
RFCOMM_BUFFER_SIZE macro 48
RFCOMM_CFC_ENABLED macro 799
RFCOMM_CFC_LOCAL_CREDIT macro 799
RFCOMM_CFC_MAX_INITIAL_CREDIT macro 799
rfcomm_cmd_queue.h 1145
RFCOMM_CMD_STATUS_FC_PENDING macro 799
RFCOMM_CMD_STATUS_PENDING macro 799
RFCOMM_CMD_STATUS_WAITING_RESPONSE macro 800
RFCOMM_COMMAND macro 800
rfcomm_config.h 1146
rfcomm_cq_ack_cmd function 787
rfcomm_cq_ack_mx_cmd function 787
rfcomm_cq_find_failed_pn function 798
RFCOMM_CTL_MSG_CLD macro 800
RFCOMM_CTL_MSG_FCOFF macro 800
RFCOMM_CTL_MSG_FCON macro 800
RFCOMM_CTL_MSG_MSC macro 801
RFCOMM_CTL_MSG_NSC macro 801
RFCOMM_CTL_MSG_PN macro 801
RFCOMM_CTL_MSG_PSC macro 801
RFCOMM_CTL_MSG_RLS macro 801
RFCOMM_CTL_MSG_RPN macro 802
RFCOMM_CTL_MSG_SNC macro 802
RFCOMM_CTL_MSG_TEST macro 802
RFCOMM_DLC_CHANGED_CONN_STATE macro 802
RFCOMM_DLC_CHANGED_REMOTE_MSC macro 802
RFCOMM_DLC_CONNECTION_FAILED macro 803
RFCOMM_DLC_STATE_CLOSED macro 803
RFCOMM_DLC_STATE_OPEN macro 803
RFCOMM_DLCI_CONTROL macro 803
RFCOMM_DLCI_FREE macro 803
RFCOMM_ENABLE_MULTIDEVICE_CHANNELS macro 49
RFCOMM_ERR_DM macro 804
RFCOMM_ERR_INTERRUPTED macro 828
RFCOMM_ERR_SUCCESS macro 804
RFCOMM_ERR_TIMEOUT macro 804
RFCOMM_FC_TYPE_AGREGATE macro 804
RFCOMM_FC_TYPE_CREDIT macro 804
rfcomm_find_session function 797
RFCOMM_FLAG_CR macro 805
RFCOMM_FLAG_EA macro 805
RFCOMM_FLAG_PF macro 805
RFCOMM_FRAME_HEADER_LEN macro 805
RFCOMM_FRAME_TYPE_DISC macro 805
RFCOMM_FRAME_TYPE_DM macro 806
RFCOMM_FRAME_TYPE_SABM macro 806
RFCOMM_FRAME_TYPE_UA macro 806
RFCOMM_FRAME_TYPE_UI macro 806
RFCOMM_FRAME_TYPE_UIH macro 806
RFCOMM_INFO_LEN macro 49
RFCOMM_LINE_STATUS_FRAMING macro 807
RFCOMM_LINE_STATUS_OVERRUN macro 807
RFCOMM_LINE_STATUS_PARITY macro 807
RFCOMM_LOCAL_CREDIT macro 49
RFCOMM_LOCAL_CREDIT_SEND_THRESHOLD_DECL macro 50
RFCOMM_MAX_CMD_BUFFERS macro 49
RFCOMM_MAX_DLCS macro 50
RFCOMM_MAX_INFO_LEN macro 807
RFCOMM_MAX_SERVER_CHANNELS macro 50
RFCOMM_MAX_SESSIONS macro 50
rfcomm_mgr.h 1147
RFCOMM_MIX_INFO_LEN macro 828
RFCOMM_MODEM_STATUS_DV macro 807
RFCOMM_MODEM_STATUS_FC macro 808
RFCOMM_MODEM_STATUS_IC macro 808

RFCOMM_MODEM_STATUS_RTC macro 808
RFCOMM_MODEM_STATUS_RTR macro 808
rfcomm_mx.h 1147
RFCOMM_MX_MSG_FCOFF macro 808
RFCOMM_MX_MSG_FCON macro 809
RFCOMM_MX_MSG_MAX_DATA_LEN macro 829
RFCOMM_MX_MSG_MSC macro 809
RFCOMM_MX_MSG_NSC macro 809
RFCOMM_MX_MSG_PN macro 809
RFCOMM_MX_MSG_RLS macro 809
RFCOMM_MX_MSG_RPN macro 810
RFCOMM_MX_MSG_TEST macro 810
rfcomm_private.h 1148
RFCOMM_RESPONSE macro 810
RFCOMM_ROLE_INITIATOR macro 810
RFCOMM_ROLE_RESPONDER macro 810
RFCOMM_RPN_BAUD_RATE_1152 macro 811
RFCOMM_RPN_BAUD_RATE_192 macro 811
RFCOMM_RPN_BAUD_RATE_2304 macro 811
RFCOMM_RPN_BAUD_RATE_24 macro 811
RFCOMM_RPN_BAUD_RATE_384 macro 811
RFCOMM_RPN_BAUD_RATE_48 macro 812
RFCOMM_RPN_BAUD_RATE_576 macro 812
RFCOMM_RPN_BAUD_RATE_72 macro 812
RFCOMM_RPN_BAUD_RATE_96 macro 812
RFCOMM_RPN_DATA_BIT_5 macro 812
RFCOMM_RPN_DATA_BIT_6 macro 813
RFCOMM_RPN_DATA_BIT_7 macro 813
RFCOMM_RPN_DATA_BIT_8 macro 813
RFCOMM_RPN_FLC_N macro 813
RFCOMM_RPN_FLC_RTC_INPUT macro 813
RFCOMM_RPN_FLC_RTC_OUTPUT macro 814
RFCOMM_RPN_FLC_RTR_INPUT macro 814
RFCOMM_RPN_FLC_RTR_OUTPUT macro 814
RFCOMM_RPN_FLC_XONOFF_INPUT macro 814
RFCOMM_RPN_FLC_XONOFF_OUTPUT macro 814
RFCOMM_RPN_PARITY_EVEN macro 815
RFCOMM_RPN_PARITY_MARK macro 815
RFCOMM_RPN_PARITY_N macro 815
RFCOMM_RPN_PARITY_ODD macro 815
RFCOMM_RPN_PARITY_SPACE macro 815
RFCOMM_RPN_PARITY_Y macro 816
RFCOMM_RPN_STOP_BIT_1 macro 816
RFCOMM_RPN_STOP_BIT_1_5 macro 816
RFCOMM_RPN_XOFF_DEFAULT macro 816
RFCOMM_RPN_XON_DEFAULT macro 816
rfcomm_send_cmd function 787
rfcomm_send_commands_from_queue function 787
rfcomm_send_mx_msc_cmd function 787
rfcomm_send_mx_pn_cmd function 788
RFCOMM_SERIAL_PORT_CH_1 macro 817
RFCOMM_SERIAL_PORT_CH_10 macro 817
RFCOMM_SERIAL_PORT_CH_11 macro 817
RFCOMM_SERIAL_PORT_CH_12 macro 817
RFCOMM_SERIAL_PORT_CH_13 macro 817
RFCOMM_SERIAL_PORT_CH_14 macro 818
RFCOMM_SERIAL_PORT_CH_15 macro 818
RFCOMM_SERIAL_PORT_CH_16 macro 818
RFCOMM_SERIAL_PORT_CH_17 macro 818
RFCOMM_SERIAL_PORT_CH_18 macro 818
RFCOMM_SERIAL_PORT_CH_19 macro 819
RFCOMM_SERIAL_PORT_CH_2 macro 819
RFCOMM_SERIAL_PORT_CH_20 macro 819
RFCOMM_SERIAL_PORT_CH_21 macro 819
RFCOMM_SERIAL_PORT_CH_22 macro 819
RFCOMM_SERIAL_PORT_CH_23 macro 820
RFCOMM_SERIAL_PORT_CH_24 macro 820
RFCOMM_SERIAL_PORT_CH_25 macro 820
RFCOMM_SERIAL_PORT_CH_26 macro 820
RFCOMM_SERIAL_PORT_CH_27 macro 820
RFCOMM_SERIAL_PORT_CH_28 macro 821
RFCOMM_SERIAL_PORT_CH_29 macro 821
RFCOMM_SERIAL_PORT_CH_3 macro 821
RFCOMM_SERIAL_PORT_CH_30 macro 821
RFCOMM_SERIAL_PORT_CH_4 macro 821
RFCOMM_SERIAL_PORT_CH_5 macro 822
RFCOMM_SERIAL_PORT_CH_6 macro 822
RFCOMM_SERIAL_PORT_CH_7 macro 822
RFCOMM_SERIAL_PORT_CH_8 macro 822
RFCOMM_SERIAL_PORT_CH_9 macro 822
RFCOMM_SESSION_CHANGED_AFC macro 823
RFCOMM_SESSION_CHANGED_CONN_STATE macro 823
RFCOMM_SESSION_STATE_CONNECTED macro 823
RFCOMM_SESSION_STATE_DISCONNECTED macro 823
RFCOMM_SESSION_STATE_FREE macro 823
rfcomm_signal.h 1149
RFCOMM_TIMEOUT macro 824
rfcomm_timer.h 1149
ROUNDFRACT32 macro 3117
RTCC Driver Library 1896
RTCC Mode Operations 4914
RTCC Peripheral Library 4910
RTCC_ALARM_MASK_CONFIGURATION enumeration 4966
RTCC_MODULE_ID enumeration 4967
RTCC_VALUE_REGISTER_POINTER enumeration 4967

S

SADDLEBROWN macro 2719
Sample Code 3986
Sample Functionality 2407, 2497
Sample Library Help 5577
SAMPLE_DataGet function 5584
SAMPLE_DataGive function 5584
SAMPLE_DataStatus function 5585
SAMPLE_Deinitialize function 5581
SAMPLE_INDEX_0 macro 5586
SAMPLE_INDEX_1 macro 5586
SAMPLE_Initialize function 5580
sample_module.h 5587
sample_module_config_template.h 5588
SAMPLE_MODULE_DATA_AVAILABLE enumeration member 5586
SAMPLE_MODULE_DATA_BUSY enumeration member 5586
SAMPLE_MODULE_DATA_NONE enumeration member 5586
SAMPLE_MODULE_DATA_STATUS enumeration 5586
SAMPLE_MODULE_INIT_DATA structure 5587
SAMPLE_MODULE_INSTANCES_NUMBER macro 5578

SAMPLE_MODULE_INTERRUPT_MODE macro 5578
SAMPLE_MODULE_TIMEOUT macro 5578
SAMPLE_Reinitialize function 5581
SAMPLE_Status function 5583
SAMPLE_Tasks function 5582
SAT16 function 3083
SAT16N function 3083
SAT16P function 3084
SB_MODULE_ID enumeration 5009
sbc.h 1149
SBC_ALLOCATION_METHOD_ALL macro 263
SBC_ALLOCATION_METHOD_LOUDNESS macro 264
SBC_ALLOCATION_METHOD_SNR macro 264
SBC_AM_LOUDNESS macro 833
SBC_AM_SNR macro 833
SBC_BE macro 833
SBC_BLK_12 macro 834
SBC_BLK_16 macro 834
SBC_BLK_4 macro 834
SBC_BLK_8 macro 834
SBC_BLOCK_LENGTH_12 macro 264
SBC_BLOCK_LENGTH_16 macro 264
SBC_BLOCK_LENGTH_4 macro 264
SBC_BLOCK_LENGTH_8 macro 265
SBC_BLOCK_LENGTH_ALL macro 265
SBC_CHANNEL_MODE_ALL macro 265
SBC_CHANNEL_MODE_DUAL_CHANNEL macro 265
SBC_CHANNEL_MODE_JOINT_STEREO macro 265
SBC_CHANNEL_MODE_MONO macro 266
SBC_CHANNEL_MODE_STEREO macro 266
sbc_decode function 829
SBC_FREQ_16000 macro 834
SBC_FREQ_32000 macro 835
SBC_FREQ_44100 macro 835
SBC_FREQ_48000 macro 835
sbc_get_codesize function 829
sbc_get_frame_duration function 830
sbc_get_frame_length function 830
sbc_get_state_size function 831
sbc_init function 831
SBC_LE macro 835
SBC_MODE_DUAL_CHANNEL macro 835
SBC_MODE_JOINT_STEREO macro 836
SBC_MODE_MONO macro 836
SBC_MODE_STEREO macro 836
SBC_SAMPLING_FREQUENCY_16000 macro 266
SBC_SAMPLING_FREQUENCY_32000 macro 266
SBC_SAMPLING_FREQUENCY_44100 macro 266
SBC_SAMPLING_FREQUENCY_48000 macro 267
SBC_SAMPLING_FREQUENCY_ALL macro 267
SBC_SB_4 macro 836
SBC_SB_8 macro 836
sbc_struct structure 831
SBC_SUBBANDS_4 macro 267
SBC_SUBBANDS_8 macro 267
SBC_SUBBANDS_ALL macro 267
sbc_t type 832
Scheme Object 2816
Scroll Bar Object 2823
SD Card Driver Initialization 1902
SDCARD_DETECTION_LOGIC enumeration 1923
SDCARD_MAX_LIMIT macro 1924
sdp.h 1150
SDP_ALLOCATE_BUFFERS function 19
SDP_ALLOCATE_BUFFERS_RAM_SIZE_VAR macro 51
SDP_ALLOCATE_BUFFERS_VARS macro 51
SDP_ATTRID_AdditionalProtocolDescriptorLists macro 845
SDP_ATTRID_BluetoothProfileDescriptorList macro 845
SDP_ATTRID_BrowseGroupList macro 845
SDP_ATTRID_ClientExecutableURL macro 846
SDP_ATTRID_DIPrimaryRecord macro 846
SDP_ATTRID_DIProductId macro 846
SDP_ATTRID_DISpecificationId macro 846
SDP_ATTRID_DIVendorId macro 846
SDP_ATTRID_DIVendorIdSource macro 847
SDP_ATTRID_DIVersion macro 847
SDP_ATTRID_DocumentationURL macro 847
SDP_ATTRID_GAPRemoteAudioVolumeControl macro 847
SDP_ATTRID_GroupID macro 847
SDP_ATTRID_HCRP_1284ID macro 881
SDP_ATTRID_HCRP_DeviceLocation macro 881
SDP_ATTRID_HCRP_DeviceName macro 881
SDP_ATTRID_HCRP_FriendlyName macro 881
SDP_ATTRID_HDPDataExchangeSpecification macro 848
SDP_ATTRID_HDPMCAPSupportedProcedures macro 848
SDP_ATTRID_HDPSuportedFeatures macro 848
SDP_ATTRID_HFPAGNetwork macro 848
SDP_ATTRID_HFPSupportedFeatures macro 848
SDP_ATTRID_HIDBatteryPower macro 849
SDP_ATTRID_HIDBootDevice macro 849
SDP_ATTRID_HIDCountryCode macro 849
SDP_ATTRID_HIDDescriptorList macro 849
SDP_ATTRID_HIDDeviceReleaseNumber macro 849
SDP_ATTRID_HIDDeviceSubclass macro 850
SDP_ATTRID_HIDLANGIDBaseList macro 850
SDP_ATTRID_HIDNormallyConnectable macro 850
SDP_ATTRID_HIDParserVersion macro 850
SDP_ATTRID_HIDProfileVersion macro 850
SDP_ATTRID_HIDReconnectInitiate macro 851
SDP_ATTRID_HIDRemoteWake macro 851
SDP_ATTRID_HIDSDPDisable macro 851
SDP_ATTRID_HIDSupervisionTimeout macro 851
SDP_ATTRID_HIDVirtualCable macro 851
SDP_ATTRID_IconURL macro 852
SDP_ATTRID_INVALID macro 852
SDP_ATTRID_LanguageBaseAttributeIDList macro 852
SDP_ATTRID_OFFSET_ProviderName macro 852
SDP_ATTRID_OFFSET_ServiceDescription macro 852
SDP_ATTRID_OFFSET_ServiceName macro 853
SDP_ATTRID_PrimaryLanguageBaseId macro 853
SDP_ATTRID_ProtocolDescriptorList macro 853
SDP_ATTRID_ServiceAvailability macro 853
SDP_ATTRID_ServiceClassIDList macro 853
SDP_ATTRID_ServiceDatabaseState macro 854
SDP_ATTRID_ServiceID macro 854
SDP_ATTRID_ServiceInfoTimeToLive macro 854

SDP_ATTRID_ServiceRecordHandle macro 854
SDP_ATTRID_ServiceRecordState macro 854
SDP_ATTRID_SupportedFeatures macro 855
SDP_ATTRID_VersionNumberList macro 855
sdp_client.h 1156
SDP_CLIENT_EVT_CONNECTED macro 855
SDP_CLIENT_EVT_CONNECTION_FAILED macro 880
SDP_CLIENT_EVT_DISCONNECTED macro 855
SDP_CLIENT_EVT_NULL macro 855
SDP_CLSID_ADVANCED_AUDIO_DISTRIBUTION macro 856
SDP_CLSID_AUDIO_SINK macro 856
SDP_CLSID_AUDIO_SOURCE macro 856
SDP_CLSID_AV_REMOTE_CONTROL macro 856
SDP_CLSID_AV_REMOTE_CONTROL_CONTROLLER macro 856
SDP_CLSID_AV_REMOTE_CONTROL_PROFILE_ID macro 857
SDP_CLSID_AV_REMOTE_CONTROL_TARGET macro 857
SDP_CLSID_AVCTP macro 857
SDP_CLSID_AVDTP macro 857
SDP_CLSID_BrowseGroupDescriptorServiceClassID macro 857
SDP_CLSID_DialupNetworking macro 858
SDP_CLSID_GENERIC_AUDIO macro 858
SDP_CLSID_HARD_COPY_CABLE_REPLACEMENT macro 881
SDP_CLSID_HARD_COPY_CONTROL_CHANNEL macro 882
SDP_CLSID_HARD_COPY_DATA_CHANNEL macro 882
SDP_CLSID_HARD_COPY_NOTIFICATION macro 882
SDP_CLSID_HCR_PRINT macro 882
SDP_CLSID_HCR_SCAN macro 882
SDP_CLSID_HDP macro 858
SDP_CLSID_HDP_SINK macro 858
SDP_CLSID_HDP_SOURCE macro 858
SDP_CLSID_HFP macro 859
SDP_CLSID_HFP_AG macro 859
SDP_CLSID_HID macro 859
SDP_CLSID_HIDProtocol macro 859
SDP_CLSID_HSP macro 859
SDP_CLSID_HSP_AG macro 860
SDP_CLSID_HSP_HS macro 860
SDP_CLSID_L2CAP macro 860
SDP_CLSID_MCAP_CONTROL macro 860
SDP_CLSID_MCAP_DATA macro 860
SDP_CLSID_OBEXFileTransfer macro 861
SDP_CLSID_OBEXObjectPush macro 861
SDP_CLSID_PBAP_PCE macro 861
SDP_CLSID_PBAP_PSE macro 861
SDP_CLSID_PNPInformation macro 861
SDP_CLSID_PublicBrowseGroup macro 862
SDP_CLSID_RFCOMM macro 862
SDP_CLSID_SerialPort macro 862
SDP_CLSID_ServiceDiscoveryServerServiceClassID macro 862
sdp_compare_uuid_de function 838
sdp_config.h 1157
SDP_DATA_TYPE_ALTERNATIVE macro 862
SDP_DATA_TYPE_BOOL macro 863
SDP_DATA_TYPE_INT macro 863
SDP_DATA_TYPE_INT128 macro 863
SDP_DATA_TYPE_INT16 macro 863
SDP_DATA_TYPE_INT32 macro 863
SDP_DATA_TYPE_INT64 macro 864
SDP_DATA_TYPE_INT8 macro 864
SDP_DATA_TYPE_NIL macro 864
SDP_DATA_TYPE_SEQUENCE macro 864
SDP_DATA_TYPE_STRING macro 864
SDP_DATA_TYPE_UINT macro 865
SDP_DATA_TYPE_UINT128 macro 865
SDP_DATA_TYPE_UINT16 macro 865
SDP_DATA_TYPE_UINT32 macro 865
SDP_DATA_TYPE_UINT64 macro 865
SDP_DATA_TYPE_UINT8 macro 866
SDP_DATA_TYPE_URL macro 866
SDP_DATA_TYPE_UUID macro 866
SDP_DATA_TYPE_UUID128 macro 866
SDP_DATA_TYPE_UUID16 macro 866
SDP_DATA_TYPE_UUID32 macro 867
SDP_ERROR_INSUFFICIENT_RESOURCE macro 867
SDP_ERROR_INVALID_CONTINUATION_STATE macro 867
SDP_ERROR_INVALID_PDU_SIZE macro 867
SDP_ERROR_INVALID_REQUEST_SYNTAX macro 867
SDP_ERROR_INVALID_SDP_VERSION macro 868
SDP_ERROR_INVALID_SR_HANDLE macro 868
SDP_ERROR_RESERVED macro 868
SDP_ErrorResponse macro 868
sdp_find_attributes function 838
sdp_find_service_records2 function 838
SDP_FTP_SERVICE_ID macro 868
SDP_HID_SERVICE_ID macro 869
SDP_HSP_AG_SERVICE_ID macro 869
SDP_HSP_HS_SERVICE_ID macro 869
SDP_MAX_ATTRIBUTE_PATTERN_LEN macro 869
SDP_MAX_ATTRIBUTE_RESULT_LEN macro 52
SDP_MAX_DATA_ELEMENT_LEN macro 869
SDP_MAX_DATA_ELEMENTS macro 870
SDP_MAX_PDU_BUFFERS macro 52
SDP_MAX_SEARCH_PATTERN_LEN macro 870
SDP_MAX_SEARCH_RESULT_LEN macro 52
SDP_MAX_TRANSACTIONS macro 870
sdp_packet.h 1157
SDP_PDU_HEADER_LEN macro 870
sdp_private.h 1158
SDP_RFCOMM_SERVICE_ID macro 870
SDP_ServiceAttributeRequest macro 871
SDP_ServiceAttributeResponse macro 871
SDP_ServiceSearchAttributeRequest macro 871
SDP_ServiceSearchAttributeResponse macro 871
SDP_ServiceSearchRequest macro 871
SDP_ServiceSearchResponse macro 872
SDP_SR_HANDLE_HDP_SINK macro 872
SDP_SR_HANDLE_HDP_SOURCE macro 872
SDP_SR_HANDLE_HFP_HF macro 872
SDP_SR_HANDLE_HID macro 872
SDP_SR_HANDLE_HID_KEYBOARD macro 873
SDP_SR_HANDLE_HSP_HS macro 873
SDP_SR_HANDLE_OBEXFileTransfer macro 873
SDP_SR_HANDLE_OBEXObjectPush macro 873
SDP_SR_HANDLE_PNPINFORMATION macro 873
SDP_SR_HANDLE_RFCOMM macro 874
SDP_SR_HANDLE_SERVER macro 874

SDP_SR_HANDLE_TEST macro 874
sdp_utils.h 1158
Secure Digital (SD) Card Driver Library 1900
Semaphores 3203
send function 6038
Sending a Report 6903
Sending Class Specific Control Transfers 7029
Sending Class-specific Control Transfers 7107
Sending Data 6866
sendto function 6038
Service Discovery 6668
SET_F_BIT macro 760
SET_FRAME_TYPE macro 760
SET_P_BIT macro 760
SET_PS_VALUE_BDADDR macro 937
SET_PS_VALUE_UINT16 macro 937
SET_PS_VALUE_UINT32 macro 937
SET_REQ_SEQ macro 761
SET_S_FUNCTION macro 761
SET_SAR macro 761
SET_TX_SEQ macro 761
SetReadBytesInAppData type 1268
SetReadBytesReadFlagInAppData type 1334
setsockopt function 6040
Setting Bus Speed 3520
Setting the Desired Audio Stream Sampling Rate 7020
SIENNA macro 2719
Single Compare Set High Mode 4482
Single Compare Toggle Mode 4483
Slave Mode 5032
SMP_ALLOCATE_BUFFERS function 19
SMTP Client Examples 6483
SMTP Client Long Message Example 6484
SMTP Client Short Message Example 6483
SMTP Module 6482
smtp.h 6494
smtp_config.h 6495
SMTP_CONNECT_ERROR macro 6492
SMTP_RESOLVE_ERROR macro 6493
SMTP_SUCCESS macro 6493
SMTPC Module 6495
smtpc.h 6514
smtpc_config.h 6515
SNMP Module 6516
SNMP Server (Agent) 6523
SNMP Traps 6525
snmp.h 6561
SNMP_COMMUNITY_TYPE enumeration 6552
snmp_config.h 6564
SNMP_END_OF_VAR macro 6551
SNMP_GENERIC_TRAP_NOTIFICATION_TYPE enumeration 6555
SNMP_ID type 6553
SNMP_INDEX type 6553
SNMP_INDEX_INVALID macro 6551
SNMP_NON_MIB_RECD_INFO structure 6553
SNMP_START_OF_VAR macro 6552
SNMP_TRAP_IP_ADDRESS_TYPE enumeration 6554
SNMP_V1 macro 6552
SNMP_V2C macro 6552
SNMP_V3 macro 6552
SNMP_VAL union 6554
SNMP_VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE enumeration 6555
snmpv3.h 6563
snmpv3_config.h 6565
SNMPV3_HMAC_HASH_TYPE enumeration 6556
SNMPV3_PRIV_PROT_TYPE enumeration 6556
SNTP Module 6565
sntp.h 6574
sntp_config.h 6575
SO_BROADCAST macro 6054
SO_DEBUG macro 6055
SO_DONTROUTE macro 6055
SO_KEEPALIVE macro 6055
SO_LINGER macro 6055
SO_OOBINLINE macro 6055
SO_RCVBUF macro 6056
SO_RCVLOWAT macro 6056
SO_RCVTIMEO macro 6056
SO_REUSEADDR macro 6056
SO_SNDBUF macro 6056
SO_SNDLOWAT macro 6057
SO_SNDTIMEO macro 6057
SOCK_DGRAM macro 6045
SOCK_STREAM macro 6046
sockaddr structure 6048
SOCKADDR type 6046
sockaddr_in structure 6048
SOCKADDR_IN type 6047
sockaddr_in6 structure 6059
SOCKADDR_IN6 type 6060
sockaddr_storage structure 6060
socket function 6039
SOCKET type 6047
SOCKET_CNXN_IN_PROGRESS macro 6046
SOCKET_DISCONNECTED macro 6046
SOCKET_ERROR macro 6046
sOggPageHdr structure 1303
sOggPageSegHdr structure 1315
SOL_SOCKET macro 6057
sOpusHeader structure 1304
sOpusPktDcpt structure 1304
sOpusStreamDcpt structure 1305
Source Code Analysis 6713
Source Files to Include 6707
Source Interrupt Management 5837
Source/Destination/Peripheral Address Management 3744
Source/Destination/Peripheral Data Management 3745
Special Considerations 4765, 5891
Special Function Modules (CRC) 3746
Speex Decoder Library 1307
speex.h 1321
SPEEX_Cleanup function 1311
SPEEX_Decoder function 1311
SPEEX_DiskRead function 1312
SPEEX_ERROR_MSG enumeration 1315

SPEEX_GetBitrate function 1313
SPEEX_GetSamplingRate function 1313
SPEEX_Initialize function 1313
SPEEX_INPUT_BUFFER_SIZE macro 1319
SPEEX_OUTPUT_BUFFER_SIZE macro 1319
SPEEX_STRING_LENGTH macro 1319
SPEEX_SUPPORT_H macro 1319
SPEEX_VENDOR_STR macro 1320
SPEEX_VERSION macro 1320
SPEEX_VERSION_ID macro 1320
SPEEX_VERSION_LENGTH macro 1320
SPI Driver Library 1929
SPI Error Handling 5034
SPI Flash Driver Libraries 1966
SPI Master Mode and Frame Master Mode 5023
SPI Master Mode and Frame Slave Mode 5024
SPI Master Mode Clock Frequency 5034
SPI Peripheral Library 5011
SPI Receive-only Operation 5034
SPI Slave Mode and Frame Master Mode 5025, 5027
SPI_AUDIO_COMMUNICATION_WIDTH enumeration 5094
SPI_AUDIO_ERROR enumeration 5094
SPI_AUDIO_PROTOCOL enumeration 5095
SPI_AUDIO_TRANSMIT_MODE enumeration 5095
SPI_BAUD_RATE_CLOCK enumeration 5095
SPI_CLOCK_POLARITY enumeration 5096
SPI_COMMUNICATION_WIDTH enumeration 5096
SPI_DATA_TYPE type 5097
SPI_ERROR_INTERRUPT enumeration 5097
SPI_FIFO_INTERRUPT enumeration 5097
SPI_FIFO_TYPE enumeration 5098
SPI_FRAME_PULSE_DIRECTION enumeration 5098
SPI_FRAME_PULSE_EDGE enumeration 5099
SPI_FRAME_PULSE_POLARITY enumeration 5099
SPI_FRAME_PULSE_WIDTH enumeration 5099
SPI_FRAME_SYNC_PULSE enumeration 5100
SPI_INPUT_SAMPLING_PHASE enumeration 5100
SPI_MODULE_ID enumeration 5100
SPI_OUTPUT_DATA_PHASE enumeration 5101
SPI_PIN enumeration 5101
spp.h 1159
SPP_ALLOCATE_BUFFERS function 19
SPP_ALLOCATE_BUFFERS_RAM_SIZE_VAR macro 53
SPP_ALLOCATE_BUFFERS_VARS macro 53
spp_config.h 1163
SPP_DECLARE_FRAME_BUFFERS macro 53
SPP_DISABLE_BUFFERING macro 53
SPP_FRAME_BUFFERS_RAM_SIZE macro 53
SPP_FRAME_BUFFERS_SIZE macro 54
SPP_MAX_PORTS macro 54
SPP_PORT_EVENT_CONNECTED enumeration member 892
SPP_PORT_EVENT_CONNECTION_FAILED enumeration member 892
SPP_PORT_EVENT_DISCONNECTED enumeration member 892
SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED enumeration member 892
SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED enumeration member 892
SPP_PORT_EVENT_REMOTE_MODEM_STATUS_CHANGED enumeration member 892
SPP_PORT_EVENT_SEND_PROGRESS enumeration member 892
SPP_PORT_OPTION_ENCRYPTED macro 890
SPP_PORT_OPTION_MASTER macro 890
SPP_PORT_OPTION_SECURE macro 890
SPP_PORT_STATE_CONNECTED enumeration member 893
SPP_PORT_STATE_CONNECTING enumeration member 893
SPP_PORT_STATE_DISCONNECTED enumeration member 893
SPP_PORT_STATE_DISCONNECTING enumeration member 893
SPP_PORT_STATE_FREE enumeration member 893
SPP_PORT_TYPE_INCOMING macro 890
SPP_PORT_TYPE_OUTGOING macro 890
spp_private.h 1164
SPP_RS232_CTS macro 891
SPP_RS232_DCD macro 891
SPP_RS232_DSR macro 891
SPP_RS232_DTR macro 891
SPP_RS232_RI macro 892
SPP_RS232_RTS macro 892
SPP_SEND_STATUS_CANCELED enumeration member 894
SPP_SEND_STATUS_INTERRUPTED enumeration member 894
SPP_SEND_STATUS_NOT_ENOUGH_RESOURCES enumeration member 894
SPP_SEND_STATUS_SUCCESS enumeration member 894
SPP_SEND_STATUS_TIMEOUT enumeration member 894
SPX_CODEC_BUFF_DIV macro 1320
SPX_CODEC_BUFF_MUL macro 1321
spxCdcDcpt structure 1316
spxDecDcpt structure 1316
SQI Peripheral Library 5105
SQI_ADDR_BYTES enumeration 5229
SQI_BD_CTRL_WORD enumeration 5230
SQI_BD_STATE enumeration 5231
SQI_CLK_DIV enumeration 5231
SQI_CS_NUM enumeration 5232
SQI_CS_OEN enumeration 5232
SQI_DATA_FORMAT enumeration 5233
SQI_DATA_MODE enumeration 5233
SQI_DATA_OEN enumeration 5233
SQI_DATA_TYPE type 5234
SQI_DUMMY_BYTES enumeration 5234
SQI_INTERRUPTS enumeration 5235
SQI_LANE_MODE enumeration 5235
SQI_MODE_BYTES enumeration 5236
SQI_MODULE_ID enumeration 5236
SQI_XFER_CMD enumeration 5237
SQI_XFER_MODE enumeration 5237
SSI Processing 6204
ssize_t macro 837
ssp.h 1164
SSP_AUTHENTICATION_REQUIREMENTS enumeration 902
SSP_EVENT enumeration 904
ssp_event.h 1165
ssp_event_handler.h 1166
SSP_EVENT_IO_CAPABILITY_REQUEST enumeration member 904
SSP_EVENT_IO_CAPABILITY_RESPONSE enumeration member 904
SSP_EVENT_KEYPRESS_NOTIFICATION enumeration member 904
SSP_EVENT_OOB_DATA_REQUEST enumeration member 904
SSP_EVENT_SIMPLE_PAIRING_COMPLETE enumeration member 904

SSP_EVENT_USER_CONFIRMATION_REQUEST enumeration member 904
SSP_EVENT_USER_PASSKEY_NOTIFICATION enumeration member 904
SSP_EVENT_USER_PASSKEY_REQUEST enumeration member 904
ssp_evt_io_capability_request function 897
ssp_evt_io_capability_response function 897
ssp_evt_keypress_notification function 897
ssp_evt_oob_data_request function 897
ssp_evt_ssp_complete function 898
ssp_evt_user_confirmation_request function 898
ssp_evt_user_passkey_notification function 898
ssp_evt_user_passkey_request function 898
SSP_IO_CAPABILITY enumeration 903
SSP_IO_CAPABILITY_DISPLAY_ONLY enumeration member 903
SSP_IO_CAPABILITY_DISPLAY_YESNO enumeration member 903
SSP_IO_CAPABILITY_KEYBOARD_ONLY enumeration member 903
SSP_IO_CAPABILITY_NO_INPUT_NO_OUTPUT enumeration member 903
SSP_KEYPRESS_NOTIFICATION_TYPE enumeration 903
SSP_MAX_MANAGERS macro 899
SSP_MITM_NOT_REQUIRED_DEDICATED_BONDING enumeration member 902
SSP_MITM_NOT_REQUIRED_GENERAL_BONDING enumeration member 902
SSP_MITM_NOT_REQUIRED_NO_BONDING enumeration member 902
SSP_MITM_REQUIRED_DEDICATED_BONDING enumeration member 902
SSP_MITM_REQUIRED_GENERAL_BONDING enumeration member 902
SSP_MITM_REQUIRED_NO_BONDING enumeration member 902
SSP_MODE enumeration 903
SSP_MODE_DISABLED enumeration member 903
SSP_MODE_ENABLED enumeration member 903
SSP_OOB_DATA_NOT_PRESENT enumeration member 904
SSP_OOB_DATA_PRESENT enumeration 904
SSP_OOB_DATA_REMOTE_DEVICE_DATA_PRESENT enumeration member 904
sSpeexHeader structure 1317
sSpxPktDcpt structure 1317
sSpxRunDcpt structure 1318
sSpxStreamDcpt structure 1318
SST25FV016B API 1978
SST25VF020B API 1997
SST25VF064C API 2017
STACK macro 1268
Standard ID Message Format 3518
Standard Master Mode 5016
Standard Slave Mode 5018
Standard SPI Mode 5016
State Machine 4417, 4623, 4913, 5015, 5283, 6677
Static Configuration 5838
Static Text Object 2834
Status (including Channel) 3747
STD_BASED_SNMP_MESSAGE_PROCESSING_MODEL enumeration 6557
STD_BASED_SNMP_SECURITY_MODEL enumeration 6557
STD_BASED_SNMPV3_SECURITY_LEVEL enumeration 6558
STDERR_FILENO macro 5669
STDIN_FILENO macro 5670
STDOUT_FILENO macro 5670
stpnr function 1324
String Descriptor Table 6786
subClassCode enumeration 6993
Support for Legacy "Ethernet Controller Library" 3993
Surface Object 2576, 2840
Synchronous External Clock Counter 5247
Synchronous Internal Clock Counter 5246
Synchronous Master Mode 5289
Synchronous Slave Mode 5290
SYS_ASSERT macro 5595
SYS_CACHE_COHERENCY enumeration 5714
sys_clk.h 5633
SYS_CLK_ClockFailureCallbackRegister function 5623
SYS_CLK_CONFIG_EXTERNAL_CLOCK macro 5615
SYS_CLK_CONFIG_FREQ_ERROR_LIMIT macro 5613
SYS_CLK_CONFIG_PRIMARY_XTAL macro 5614
SYS_CLK_CONFIG_SECONDARY_XTAL macro 5614
SYS_CLK_CONFIG_SYSPLL_INP_DIVISOR macro 5614
sys_clk_config_template.h 5634
SYS_CLK_CONFIGBIT_USBPLL_DIVISOR macro 5614
SYS_CLK_CONFIGBIT_USBPLL_ENABLE macro 5615
SYS_CLK_ERROR_HANDLER type 5630
SYS_CLK_FRC_TUNING_TYPE type 5632
SYS_CLK_FRCTune function 5632
SYS_CLK_INIT structure 5626
SYS_CLK_Initialize function 5624
SYS_CLK_PeripheralFrequencyGet function 5619
SYS_CLK_PeripheralFrequencySet function 5619
SYS_CLK_REFERENCE_SETUP structure 5630
SYS_CLK_ReferenceClockSetup function 5620
SYS_CLK_ReferenceFrequencyGet function 5622
SYS_CLK_ReferenceFrequencySet function 5621
SYS_CLK_SecondaryOscillatorDisable function 5625
SYS_CLK_SecondaryOscillatorEnable function 5625
SYS_CLK_SecondaryOscillatorIsEnabled function 5626
SYS_CLK_STATUS enumeration 5631
SYS_CLK_SystemFrequencyGet function 5617
SYS_CLK_SystemFrequencySet function 5618
SYS_CLK_TaskError function 5623
SYS_CLK_UPLL_BEFORE_DIV2_FREQ macro 5615
SYS_CMD_ADDGRP function 5639
SYS_CMD_API structure 5646
SYS_CMD_BUFFER_DMA_READY macro 5651
SYS_CMD_CallbackFunction type 5652
SYS_CMD_CONSOLE_IO_PARAM enumeration 5650
SYS_CMD_DATA_RDY_FNC type 5646
SYS_CMD_DELETE function 5638
SYS_CMD_DESCRIPTOR structure 5647
SYS_CMD_DESCRIPTOR_TABLE structure 5647
SYS_CMD_DEVICE_LIST structure 5647
SYS_CMD_DEVICE_MAX_INSTANCES macro 5651
SYS_CMD_DEVICE_NODE structure 5645
SYS_CMD_EVENT enumeration 5652
SYS_CMD_FNC type 5648
SYS_CMD_GETC_FNC type 5648
SYS_CMD_HANDLE type 5648

SYS_CMD_INIT structure 5645
SYS_CMD_INIT_DATA structure 5648
SYS_CMD_Initialize function 5638
SYS_CMD_MAX_LENGTH macro 5644
SYS_CMD_MESSAGE function 5639
SYS_CMD_MESSAGE macro 5651
SYS_CMD_MSG_FNC type 5649
SYS_CMD_PRINT function 5640
SYS_CMD_PRINT macro 5652
SYS_CMD_PRINT_FNC type 5649
SYS_CMD_PUTC_FNC type 5649
SYS_CMD_READ_BUFFER_SIZE macro 5645
SYS_CMD_READC_FNC type 5650
SYS_CMD_READY_TO_READ function 5640
SYS_CMD_READY_TO_WRITE function 5640
SYS_CMD_RegisterCallback function 5641
SYS_CMD_STATE enumeration 5650
SYS_CMD_Tasks function 5641
SYS_CMDIO_ADD function 5642
SYS_CMDIO_GET_HANDLE function 5642
sys_command.h 5652
sys_common.h 5606
sys_console.h 5671
SYS_CONSOLE_BUFFER_DMA_READY macro 5671
SYS_CONSOLE_Deinitialize function 5663
SYS_CONSOLE_DEV_DESC structure 5669
SYS_CONSOLE_DEVICE enumeration 5668
SYS_CONSOLE_Flush function 5667
SYS_CONSOLE_INDEX_0 macro 5670
SYS_CONSOLE_INDEX_1 macro 5670
SYS_CONSOLE_INDEX_2 macro 5670
SYS_CONSOLE_INDEX_3 macro 5671
SYS_CONSOLE_INIT structure 5669
SYS_CONSOLE_Initialize function 5661
SYS_CONSOLE_MESSAGE macro 5651
SYS_CONSOLE_OBJECT_INSTANCE structure 5668
SYS_CONSOLE_PRINT macro 5651
SYS_CONSOLE_Read function 5665
SYS_CONSOLE_RegisterCallback function 5667
SYS_CONSOLE_Reinitialize function 5662
SYS_CONSOLE_Status function 5664
SYS_CONSOLE_Tasks function 5664
SYS_CONSOLE_Write function 5666
SYS_DEBUG macro 5683
sys_debug.h 5689
SYS_DEBUG_BreakPoint macro 5687
SYS_DEBUG_BUFFER_DMA_READY macro 1331
SYS_DEBUG_Deinitialize function 5677
SYS_DEBUG_ErrorLevelGet function 5681
SYS_DEBUG_ErrorLevelSet function 5682
SYS_DEBUG_INDEX_0 macro 5684
SYS_DEBUG_INIT structure 5682
SYS_DEBUG_Initialize function 5677
SYS_DEBUG_Message function 5680
SYS_DEBUG_MESSAGE macro 5686
SYS_DEBUG_Print function 5681
SYS_DEBUG_PRINT macro 5686
SYS_DEBUG_Reinitialize function 5678
SYS_DEBUG_Status function 5679
SYS_DEBUG_Tasks function 5679
sys_devcon.h 5715
SYS_DEVCON_CacheClean function 5702
SYS_DEVCON_CacheCoherencyGet function 5703
SYS_DEVCON_CacheCoherencySet function 5703
SYS_DEVCON_CacheFlush function 5704
SYS_DEVCON_CacheInit function 5704
SYS_DEVCON_CacheSync function 5705
SYS_DEVCON_DataCacheAssociativityGet function 5707
SYS_DEVCON_DataCacheClean function 5705
SYS_DEVCON_DataCacheFlush function 5706
SYS_DEVCON_DataCacheInvalidate function 5706
SYS_DEVCON_DataCacheLineSizeGet function 5708
SYS_DEVCON_DataCacheLinesPerWayGet function 5708
SYS_DEVCON_DataCacheLock function 5707
SYS_DEVCON_DataCacheSizeGet function 5709
SYS_DEVCON_Deinitialize function 5695
SYS_DEVCON_HANDLE type 5713
SYS_DEVCON_INDEX_0 macro 5714
SYS_DEVCON_INIT structure 5713
SYS_DEVCON_Initialize function 5695
SYS_DEVCON_InstructionCacheAssociativityGet function 5711
SYS_DEVCON_InstructionCacheFlush function 5709
SYS_DEVCON_InstructionCacheInvalidate function 5710
SYS_DEVCON_InstructionCacheLineSizeGet function 5711
SYS_DEVCON_InstructionCacheLinesPerWayGet function 5712
SYS_DEVCON_InstructionCacheLock function 5710
SYS_DEVCON_InstructionCacheSizeGet function 5712
SYS_DEVCON_JTAGDisable function 5699
SYS_DEVCON_JTAGEnable function 5700
SYS_DEVCON_PerformanceConfig function 5701
SYS_DEVCON_PowerModeEnter function 5702
SYS_DEVCON_Reinitialize function 5696
SYS_DEVCON_Status function 5697
SYS_DEVCON_SystemLock function 5698
SYS_DEVCON_SystemUnlock function 5699
SYS_DEVCON_Tasks function 5698
SYS_DEVCON_TraceDisable function 5700
SYS_DEVCON_TraceEnable function 5701
sys_dma.h 5758
SYS_DMA_CHANNEL_CHAIN_PRIO enumeration 5752
SYS_DMA_CHANNEL_COUNT macro 5757
SYS_DMA_CHANNEL_CRC_MODE enumeration 5753
SYS_DMA_CHANNEL_HANDLE type 5755
SYS_DMA_CHANNEL_HANDLE_INVALID macro 5757
SYS_DMA_CHANNEL_IGNORE_MATCH enumeration 5756
SYS_DMA_CHANNEL_OP_MODE enumeration 5753
SYS_DMA_CHANNEL_OPERATION_MODE_CRC structure 5754
SYS_DMA_ChannelAbortEventSet function 5731
SYS_DMA_ChannelAllocate function 5733
SYS_DMA_ChannelCRCGet function 5745
SYS_DMA_ChannelCRCSet function 5745
SYS_DMA_ChannelDestinationTransferredSizeGet function 5747
SYS_DMA_ChannelDisable function 5734
SYS_DMA_ChannelEnable function 5734
SYS_DMA_ChannelErrorGet function 5742
SYS_DMA_ChannelForceAbort function 5735

SYS_DMA_ChannelForceStart function 5736
SYS_DMA_ChannelIsBusy function 5738
SYS_DMA_ChannelRelease function 5738
SYS_DMA_ChannelSetup function 5746
SYS_DMA_ChannelSetupMatchAbortMode function 5739
SYS_DMA_ChannelSourceTransferredSizeGet function 5748
SYS_DMA_ChannelTransferAdd function 5740
SYS_DMA_ChannelTransferEventHandlerSet function 5743
SYS_DMA_ChannelTransferSet function 5749
SYS_DMA_CRC_WRITE_ORDER enumeration 5754
SYS_DMA_INIT enumeration 5756
SYS_DMA_Initialize function 5729
SYS_DMA_IsBusy function 5751
SYS_DMA_Resume function 5751
SYS_DMA_SIDL enumeration 5756
SYS_DMA_Suspend function 5752
SYS_DMA_Tasks function 5728
SYS_DMA_TasksError function 5729
SYS_DMA_TasksErrorISR function 5730
SYS_DMA_TasksISR function 5731
SYS_DMA_TRANSFER_EVENT enumeration 5755
SYS_ERROR macro 5685
SYS_ERROR_LEVEL enumeration 5683
SYS_ERROR_PRINT macro 5687
sys_fs.h 5831
SYS_FS_AUTOMOUNT_ENABLE macro 5776
sys_fs_config_template.h 5833
SYS_FS_CurrentDriveGet function 5812
SYS_FS_CurrentDriveSet function 5813
SYS_FS_CurrentWorkingDirectoryGet function 5799
SYS_FS_DirClose function 5802
SYS_FS_DirectoryChange function 5800
SYS_FS_DirectoryMake function 5801
SYS_FS_DirOpen function 5802
SYS_FS_DirRead function 5803
SYS_FS_DirRewind function 5806
SYS_FS_DirSearch function 5804
SYS_FS_DriveFormat function 5815
SYS_FS_DriveLabelGet function 5813
SYS_FS_DriveLabelSet function 5814
SYS_FS_DrivePartition function 5817
SYS_FS_DriveSectorGet function 5819
SYS_FS_ERROR enumeration 5821
SYS_FS_Error function 5796
SYS_FS_EVENT enumeration 5830
SYS_FS_EVENT_HANDLER type 5831
SYS_FS_EventHandlerSet function 5820
SYS_FS_FILE_DIR_ATTR enumeration 5829
SYS_FS_FILE_OPEN_ATTRIBUTES enumeration 5822
SYS_FS_FILE_SEEK_CONTROL enumeration 5823
SYS_FS_FILE_SYSTEM_TYPE enumeration 5823
SYS_FS_FileCharacterPut function 5793
SYS_FS_FileClose function 5780
SYS_FS_FileDirectoryModeSet function 5789
SYS_FS_FileDirectoryRemove function 5789
SYS_FS_FileDirectoryRenameMove function 5790
SYS_FS_FileDirectoryTimeSet function 5791
SYS_FS_FileEOF function 5781
SYS_FS_FileError function 5796
SYS_FS_FileNameGet function 5786
SYS_FS_FileOpen function 5782
SYS_FS_FilePrintf function 5787
SYS_FS_FileRead function 5797
SYS_FS_FileSeek function 5783
SYS_FS_FileSize function 5784
SYS_FS_FileStat function 5807
SYS_FS_FileStringGet function 5794
SYS_FS_FileStringPut function 5795
SYS_FS_FileSync function 5805
SYS_FS_FileTell function 5785
SYS_FS_FileTestError function 5788
SYS_FS_FileTruncate function 5792
SYS_FS_FileWrite function 5798
SYS_FS_FORMAT enumeration 5830
SYS_FS_FSTAT structure 5824
SYS_FS_FUNCTIONS structure 5824
SYS_FS_HANDLE type 5826
SYS_FS_HANDLE_INVALID macro 5828
SYS_FS_Initialize function 5808
SYS_FS_MAX_FILE_SYSTEM_TYPE macro 5777
SYS_FS_MAX_FILES macro 5777
SYS_FS_MEDIA_MAX_BLOCK_SIZE macro 5777
SYS_FS_MEDIA_NUMBER macro 5778
SYS_FS_Mount function 5809
SYS_FS_REGISTRATION_TABLE structure 5827
SYS_FS_RESULT enumeration 5827
SYS_FS_Tasks 5769
SYS_FS_Tasks function 5811
SYS_FS_TIME union 5829
SYS_FS_Unmount function 5811
SYS_FS_VOLUME_NUMBER macro 5778
sys_init.h 5608
SYS_Initialize function 5596
SYS_INPUT_DEVICE_EVENT enumeration 5974
SYS_INPUT_DEVICE_TYPE enumeration 5973
sys_int.h 5854
SYS_INT_Disable function 5845
SYS_INT_DynamicDeregister function 5841
SYS_INT_DynamicRegister function 5841
SYS_INT_Enable function 5845
SYS_INT_ExternalInterruptTriggerSet function 5846
SYS_INT_Initialize function 5840
SYS_INT_IsEnabled function 5846
SYS_INT_PROCESSOR_STATUS type 5854
SYS_INT_ShadowRegisterAssign function 5842
SYS_INT_ShadowRegisterGet function 5843
SYS_INT_SourceDisable function 5847
SYS_INT_SourceEnable function 5848
SYS_INT_SourceIsEnabled function 5848
SYS_INT_SourceStatusClear function 5849
SYS_INT_SourceStatusGet function 5850
SYS_INT_SourceStatusSet function 5850
SYS_INT_StatusGetAndDisable function 5843
SYS_INT_StatusRestore function 5844
SYS_INT_TASKS_POINTER type 5853
SYS_INT_VectorPrioritySet function 5851

SYS_INT_VectorSubprioritySet function 5852
sys_memory.h 5859
SYS_MEMORY_DDR_Initialize function 5857
SYS_MEMORY_EBI_Initialize function 5858
SYS_MEMORY_Initialize function 5858
SYS_MESSAGE macro 5684
sys_module.h 5607
SYS_MODULE_DEINITIALIZE_ROUTINE type 5598
SYS_MODULE_INDEX type 5599
SYS_MODULE_INIT union 5599
SYS_MODULE_INITIALIZE_ROUTINE type 5600
SYS_MODULE_OBJ type 5600
SYS_MODULE_OBJ_INVALID macro 5604
SYS_MODULE_OBJ_STATIC macro 5604
SYS_MODULE_POWER_IDLE_RUN macro 5604
SYS_MODULE_POWER_IDLE_STOP macro 5605
SYS_MODULE_POWER_OFF macro 5605
SYS_MODULE_POWER_RUN_FULL macro 5605
SYS_MODULE_POWER_SLEEP macro 5606
SYS_MODULE_REINITIALIZE_ROUTINE type 5601
SYS_MODULE_STATUS_ROUTINE type 5601
SYS_MODULE_TASKS_ROUTINE type 5602
sys_msg.h 5884
SYS_MSG_BUFFER_SIZES macro 5864
sys_msg_config.h 5885
SYS_MSG_Deinitialize function 5867
SYS_MSG_GotMessages function 5875
SYS_MSG_ID2hMsgType function 5878
SYS_MSG_INIT structure 5879
SYS_MSG_Initialize function 5868
SYS_MSG_INSTANCE enumeration 5880
SYS_MSG_MailboxClose function 5872
SYS_MSG_MAILBOXES_ADDONE macro 5882
SYS_MSG_MailboxMessagesGet function 5869
SYS_MSG_MailboxMsgAdd function 5870
SYS_MSG_MailboxMsgRemove function 5871
SYS_MSG_MailboxOpen function 5872
SYS_MSG_MailboxReinit function 5873
SYS_MSG_MAX_MAILBOXES macro 5864
SYS_MSG_MAX_MSGS_DELIVERED macro 5865
SYS_MSG_MAX_PRIORITY macro 5865
SYS_MSG_MAX_TYPES macro 5865
SYS_MSG_MessageDeliver function 5876
SYS_MSG_MessageReceive function 5876
SYS_MSG_MessageSend function 5877
SYS_MSG_NUM_MAILBOX_BITMAPS macro 5883
SYS_MSG_OBJECT structure 5880
SYS_MSG_QUEUE_STATUS enumeration 5881
SYS_MSG_QueueStatus function 5879
SYS_MSG_RECEIVE_CALLBACK type 5881
SYS_MSG_RESULTS enumeration 5881
SYS_MSG_Tasks function 5869
SYS_MSG_TypeCreate function 5874
SYS_MSG_TypeRemove function 5874
SYS_MSGQ_ELEMENT structure 5882
SYS_OBJ_HANDLE type 5883
SYS_OBJ_HANDLE_INVALID macro 5883
SYS_OBJ_HANDLE_STATIC macro 5883
sys_ports.h 5914
SYS_PORTS_ChangeNotificationDisable function 5907
SYS_PORTS_ChangeNotificationEnable function 5907
SYS_PORTS_ChangeNotificationGlobalDisable function 5908
SYS_PORTS_ChangeNotificationGlobalEnable function 5908
SYS_PORTS_ChangeNotificationInIdleModeDisable function 5909
SYS_PORTS_ChangeNotificationInIdleModeEnable function 5909
SYS_PORTS_ChangeNotificationPullUpDisable function 5910
SYS_PORTS_ChangeNotificationPullUpEnable function 5911
SYS_PORTS_Clear function 5900
SYS_PORTS_DirectionGet function 5901
SYS_PORTS_DirectionSelect function 5901
SYS_PORTS_Initialize function 5906
SYS_PORTS_OpenDrainDisable function 5902
SYS_PORTS_OpenDrainEnable function 5903
SYS_PORTS_PIN_DIRECTION enumeration 5913
SYS_PORTS_PinClear function 5899
SYS_PORTS_PinDirectionSelect function 5899
SYS_PORTS_PinModeSelect function 5894
SYS_PORTS_PinOpenDrainDisable function 5895
SYS_PORTS_PinOpenDrainEnable function 5895
SYS_PORTS_PinRead function 5896
SYS_PORTS_PinSet function 5897
SYS_PORTS_PinToggle function 5897
SYS_PORTS_PinWrite function 5898
SYS_PORTS_PULLUP_PULLDOWN_STATUS enumeration 5913
SYS_PORTS_Read function 5903
SYS_PORTS_RemapInput function 5911
SYS_PORTS_RemapOutput function 5912
SYS_PORTS_Set function 5904
SYS_PORTS_Toggle function 5905
SYS_PORTS_Write function 5905
SYS_POWER_MODE enumeration 5714
SYS_PRINT macro 5685
sys_random.h 5925
sys_random_config_template.h 5925
SYS_RANDOM_CRYPTO_SEED_SIZE macro 5917
SYS_RANDOM_CryptoBlockGet function 5919
SYS_RANDOM_CryptoByteGet function 5919
SYS_RANDOM_CryptoEntropyAdd function 5920
SYS_RANDOM_CryptoGet function 5920
SYS_RANDOM_CryptoSeedSet function 5921
SYS_RANDOM_CryptoSeedSizeGet function 5922
SYS_RANDOM_Deinitialize function 5922
SYS_RANDOM_INIT structure 5924
SYS_RANDOM_Initialize function 5923
SYS_RANDOM_PseudoGet function 5923
SYS_RANDOM_PseudoSeedSet function 5924
sys_reset.h 5931
SYS_RESET_NMIDelayCountSet function 5931
SYS_RESET_ReasonClear function 5929
SYS_RESET_ReasonGet function 5930
SYS_RESET_SoftwareReset function 5930
SYS_STATUS enumeration 5603
SYS_Tasks function 5595
sys_tmr.h 5964
SYS_TMR_CALLBACK type 5959
SYS_TMR_CallbackPeriodic function 5950

SYS_TMR_CallbackSingle function 5951
SYS_TMR_CallbackStop function 5949
SYS_TMR_CLIENT_TOLERANCE macro 5961
sys_tmr_config_template.h 5965
SYS_TMR_Deinitialize function 5946
SYS_TMR_DelayMS function 5952
SYS_TMR_DelayStatusGet function 5952
SYS_TMR_DRIVER_INDEX macro 5962
SYS_TMR_FLAGS enumeration 5961
SYS_TMR_FREQUENCY macro 5962
SYS_TMR_FREQUENCY_TOLERANCE macro 5962
SYS_TMR_HANDLE type 5960
SYS_TMR_HANDLE_INVALID macro 5960
SYS_TMR_INDEX_0 macro 5961
SYS_TMR_INIT structure 5960
SYS_TMR_Initialize function 5947
SYS_TMR_INTERRUPT_NOTIFICATION macro 5963
SYS_TMR_MAX_CLIENT_OBJECTS macro 5963
SYS_TMR_MODULE_INIT macro 5963
SYS_TMR_ModuleStatusGet function 5958
SYS_TMR_ObjectCountGet function 5953
SYS_TMR_ObjectCreate function 5954
SYS_TMR_ObjectDelete function 5955
SYS_TMR_ObjectReload function 5955
SYS_TMR_Status function 5948
SYS_TMR_SystemCountFrequencyGet function 5958
SYS_TMR_SystemCountGet function 5957
SYS_TMR_Tasks function 5948
SYS_TMR_TickCounterFrequencyGet function 5957
SYS_TMR_TickCountGet function 5956
SYS_TMR_TickCountGetLong function 5956
SYS_TMR_UNIT_RESOLUTION macro 5964
sys_touch.h 5975
SYS_TOUCH_CalibrationSet function 5967
SYS_TOUCH_HANDLE type 5971
SYS_TOUCH_HANDLE_INVALID macro 5973
SYS_TOUCH_INDEX enumeration 5971
SYS_TOUCH_INIT structure 5972
SYS_TOUCH_Initialize function 5968
SYS_TOUCH_Open function 5969
SYS_TOUCH_STATUS enumeration 5972
SYS_TOUCH_Tasks function 5970
SYS_TOUCH_TouchMsgGet function 5969
SYS_VersionGet macro 5597
SYS_VersionStrGet macro 5597
sys_wdt.h 5981
sys_wdt_config.h 5981
SYS_WDT_Disable function 5979
SYS_WDT_Enable function 5979
SYS_WDT_TimerClear function 5980
System Access 1394, 1444, 1483, 1522, 1729, 1769, 1931
System and Client Interfaces 2921
System Bus Peripheral Library 4970
System Configuration 5763
System Functions 5655
System Initialization 1869, 2403, 2496
System Initialization and Deinitialization 1968
System Interaction 2043, 5635, 5691, 5937
System Service Libraries Help 5589
System Service Overview 5589
System State Machine 5589
system.h 5608

T

Table of Library Functions 3000, 3124
TAN macro 2719
Target Initialization 4972
Tasks Routine 2112, 2132
TCP Changes 5988
TCP Module 6576
tcp.h 6613
TCP/IP Stack Libraries Help 5982
TCP/IP Stack Library Overview 5982
TCP/IP Stack Library Porting Guide 5986
TCP_ADJUST_FLAGS enumeration 6607
tcp_config.h 6615
TCP_NODELAY macro 6057
TCP_OPTION_KEEP_ALIVE_DATA structure 6611
TCP_OPTION_LINGER_DATA structure 6607
TCP_OPTION_THRES_FLUSH_TYPE enumeration 6610
TCP_PORT type 6608
TCP_SOCKET type 6608
TCP_SOCKET_INFO structure 6608
TCP_SOCKET_OPTION enumeration 6609
tcpip.h 6466
tcpip_announce_config.h 6009
TCPIP_ANNOUNCE_MAX_PAYLOAD macro 6008
TCPIP_ANNOUNCE_TASK_RATE macro 6007
TCPIP_ARP_CACHE_ENTRIES macro 6011
TCPIP_ARP_CACHE_ENTRY_RETRIES macro 6012
TCPIP_ARP_CACHE_PENDING_ENTRY_TMO macro 6012
TCPIP_ARP_CACHE_PENDING_RETRY_TMO macro 6012
TCPIP_ARP_CACHE_PERMANENT_QUOTA macro 6012
TCPIP_ARP_CACHE_PURGE_QUANTA macro 6012
TCPIP_ARP_CACHE_PURGE_THRESHOLD macro 6013
TCPIP_ARP_CACHE_SOLVED_ENTRY_TMO macro 6013
TCPIP_ARP_CacheEntriesNoGet function 6021
TCPIP_ARP_CacheThresholdSet function 6021
TCPIP_ARP_ENTRY_QUERY structure 6023
TCPIP_ARP_ENTRY_TYPE enumeration 6023
TCPIP_ARP_EntryGet function 6015
TCPIP_ARP_EntryQuery function 6022
TCPIP_ARP_EntryRemove function 6015
TCPIP_ARP_EntryRemoveAll function 6022
TCPIP_ARP_EntryRemoveNet function 6016
TCPIP_ARP_EntrySet function 6020
TCPIP_ARP_EVENT_HANDLER type 6024
TCPIP_ARP_EVENT_TYPE enumeration 6024
TCPIP_ARP_GRATUITOUS_PROBE_COUNT macro 6013
TCPIP_ARP_HANDLE type 6025
TCPIP_ARP_HandlerDeRegister function 6017
TCPIP_ARP_HandlerRegister function 6017
TCPIP_ARP_IsResolved function 6018
TCPIP_ARP_MODULE_CONFIG structure 6026
TCPIP_ARP_OPERATION_TYPE enumeration 6025
TCPIP_ARP_Probe function 6018

TCPIP_ARP_Resolve function 6019
TCPIP_ARP_RESULT enumeration 6026
TCPIP_ARP_Task function 6020
TCPIP_ARP_TASK_PROCESS_RATE macro 6013
TCPIP_BSD_Socket function 6042
TCPIP_DDNS_CHECKIP_SERVER macro 6150
TCPIP_DDNS_LastIPGet function 6151
TCPIP_DDNS_LastStatusGet function 6151
TCPIP_DDNS_ServiceSet function 6152
TCPIP_DDNS_Task function 6153
TCPIP_DDNS_TASK_TICK_RATE macro 6150
TCPIP_DDNS_UpdateForce function 6152
TCPIP_DHCP_BOOT_FILE_NAME_SIZE macro 6066
TCPIP_DHCP_CLIENT_ENABLED macro 6090
TCPIP_DHCP_Disable function 6067
TCPIP_DHCP_Enable function 6068
TCPIP_DHCP_EVENT_HANDLER type 6073
TCPIP_DHCP_EVENT_TYPE enumeration 6073
TCPIP_DHCP_HANDLE type 6074
TCPIP_DHCP_HandlerDeRegister function 6068
TCPIP_DHCP_HandlerRegister function 6069
TCPIP_DHCP_HOST_NAME_CALLBACK type 6076
TCPIP_DHCP_HOST_NAME_SIZE macro 6076
TCPIP_DHCP_HostNameCallbackRegister function 6069
TCPIP_DHCP_INFO structure 6075
TCPIP_DHCP_InfoGet function 6088
TCPIP_DHCP_IsActive function 6072
TCPIP_DHCP_IsBound function 6071
TCPIP_DHCP_IsEnabled function 6072
TCPIP_DHCP_IsServerDetected function 6072
TCPIP_DHCP_MODULE_CONFIG structure 6074
TCPIP_DHCP_Renew function 6070
TCPIP_DHCP_Request function 6071
TCPIP_DHCP_RequestTimeoutSet function 6085
TCPIP_DHCP_STATUS enumeration 6075
TCPIP_DHCP_STORE_BOOT_FILE_NAME macro 6066
TCPIP_DHCP_Task function 6086
TCPIP_DHCP_TASK_TICK_RATE macro 6091
TCPIP_DHCP_TIMEOUT macro 6091
TCPIP_DHCPS_ADDRESS_CONFIG structure 6090
TCPIP_DHCPS_DEFAULT_IP_ADDRESS_RANGE_START macro 6080
TCPIP_DHCPS_DEFAULT_SERVER_IP_ADDRESS macro 6080
TCPIP_DHCPS_DEFAULT_SERVER_NETMASK_ADDRESS macro 6080
TCPIP_DHCPS_DEFAULT_SERVER_PRIMARY_DNS_ADDRESS macro 6081
TCPIP_DHCPS_DEFAULT_SERVER_SECONDARY_DNS_ADDRESS macro 6081
TCPIP_DHCPS_Disable function 6083
TCPIP_DHCPS_Enable function 6084
TCPIP_DHCPS_GetPoolEntries function 6087
TCPIP_DHCPS_IsEnabled function 6087
TCPIP_DHCPS_LEASE_DURATION macro 6081
TCPIP_DHCPS_LEASE_ENTRIES_DEFAULT macro 6081
TCPIP_DHCPS_LEASE_ENTRY structure 6089
TCPIP_DHCPS_LEASE_HANDLE type 6089
TCPIP_DHCPS_LEASE_REMOVED_BEFORE_ACK macro 6081
TCPIP_DHCPS_LEASE_SOLVED_ENTRY_TMO macro 6082
TCPIP_DHCPS_LeaseEntryGet function 6088
TCPIP_DHCPS_LeaseEntryRemove function 6085
TCPIP_DHCPS_MODULE_CONFIG structure 6091
TCPIP_DHCPS_POOL_ENTRY_TYPE enumeration 6090
TCPIP_DHCPS_RemovePoolEntries function 6084
TCPIP_DHCPS_Task function 6086
TCPIP_DHCPS_TASK_PROCESS_RATE macro 6082
TCPIP_DHCPV6_CLIENT_DUID_TYPE macro 6096
TCPIP_DHCPV6_CLIENT_INFO structure 6104
TCPIP_DHCPV6_CLIENT_PORT macro 6096
TCPIP_DHCPV6_CLIENT_STATE enumeration 6105
TCPIP_DHCPV6_ClientInfoGet function 6103
TCPIP_DHCPV6_CONFIG_FLAGS enumeration 6105
TCPIP_DHCPV6_DNS_SERVERS_NO macro 6096
TCPIP_DHCPV6_DOMAIN_SEARCH_LIST_SIZE macro 6096
TCPIP_DHCPV6_DUID_TYPE enumeration 6106
TCPIP_DHCPV6_EVENT_HANDLER type 6106
TCPIP_DHCPV6_FORCED_SERVER_PREFERENCE macro 6096
TCPIP_DHCPV6_HANDLE type 6106
TCPIP_DHCPV6_HandlerDeRegister function 6103
TCPIP_DHCPV6_HandlerRegister function 6103
TCPIP_DHCPV6_IA_EVENT union 6107
TCPIP_DHCPV6_IA_FREE_DESCRIPTORS_NO macro 6097
TCPIP_DHCPV6_IA_INFO structure 6107
TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_PREF_LTIME macro 6097
TCPIP_DHCPV6_IA_SOLICIT_ADDRESS_VALID_LTIME macro 6097
TCPIP_DHCPV6_IA_STATE enumeration 6108
TCPIP_DHCPV6_IA_SUBSTATE enumeration 6108
TCPIP_DHCPV6_IA_TYPE enumeration 6109
TCPIP_DHCPV6_IaInfoGet function 6104
TCPIP_DHCPV6_IANA_DEFAULT_T1 macro 6097
TCPIP_DHCPV6_IANA_DEFAULT_T2 macro 6097
TCPIP_DHCPV6_IANA_DESCRIPTORS_NO macro 6098
TCPIP_DHCPV6_IANA_SOLICIT_ADDRESSES_NO macro 6098
TCPIP_DHCPV6_IANA_SOLICIT_DEFAULT_ADDRESS macro 6098
TCPIP_DHCPV6_IANA_SOLICIT_T1 macro 6098
TCPIP_DHCPV6_IANA_SOLICIT_T2 macro 6098
TCPIP_DHCPV6_IATA_DEFAULT_T1 macro 6099
TCPIP_DHCPV6_IATA_DEFAULT_T2 macro 6099
TCPIP_DHCPV6_IATA_DESCRIPTORS_NO macro 6099
TCPIP_DHCPV6_IATA_SOLICIT_ADDRESSES_NO macro 6099
TCPIP_DHCPV6_IATA_SOLICIT_DEFAULT_ADDRESS macro 6099
TCPIP_DHCPV6_MESSAGE_BUFFER_SIZE macro 6100
TCPIP_DHCPV6_MESSAGE_BUFFERS macro 6100
TCPIP_DHCPV6_MIN_UDP_TX_BUFFER_SIZE macro 6100
TCPIP_DHCPV6_MODULE_CONFIG structure 6109
TCPIP_DHCPV6_SERVER_PORT macro 6100
TCPIP_DHCPV6_SERVER_STATUS_CODE enumeration 6110
TCPIP_DHCPV6_SKIP_DAD_PROCESS macro 6100
TCPIP_DHCPV6_STATUS_CODE_MESSAGE_LEN macro 6101
TCPIP_DHCPV6_Task function 6103
TCPIP_DHCPV6_TASK_TICK_RATE macro 6101
TCPIP_DHCPV6_USER_NOTIFICATION macro 6101
TCPIP_DNS_CLIENT_ADDRESS_TYPE macro 6118
TCPIP_DNS_CLIENT_CACHE_DEFAULT_TTL_VAL macro 6117
TCPIP_DNS_CLIENT_CACHE_ENTRIES macro 6116
TCPIP_DNS_CLIENT_CACHE_ENTRY_TMO macro 6116
TCPIP_DNS_CLIENT_CACHE_PER_IPV4_ADDRESS macro 6116
TCPIP_DNS_CLIENT_CACHE_PER_IPV6_ADDRESS macro 6117

TCPIP_DNS_CLIENT_CACHE_UNSOLVED_ENTRY_TMO macro 6117
TCPIP_DNS_CLIENT_INFO structure 6132
TCPIP_DNS_CLIENT_LOOKUP_RETRY_TMO macro 6118
TCPIP_DNS_CLIENT_MAX_HOSTNAME_LEN macro 6118
TCPIP_DNS_CLIENT_MAX_SELECT_INTERFACES macro 6118
TCPIP_DNS_CLIENT_MODULE_CONFIG structure 6131
TCPIP_DNS_CLIENT_SERVER_TMO macro 6117
TCPIP_DNS_CLIENT_TASK_PROCESS_RATE macro 6117
TCPIP_DNS_CLIENT_USER_NOTIFICATION macro 6118
TCPIP_DNS_ClientInfoGet function 6125
TCPIP_DNS_ClientTask function 6124
TCPIP_DNS_Disable function 6121
TCPIP_DNS_Enable function 6120
TCPIP_DNS_ENABLE_FLAGS enumeration 6133
TCPIP_DNS_ENTRY_QUERY structure 6132
TCPIP_DNS_EntryQuery function 6125
TCPIP_DNS_EVENT_HANDLER type 6129
TCPIP_DNS_EVENT_TYPE enumeration 6130
TCPIP_DNS_GetIPAddressesNumber function 6126
TCPIP_DNS_GetIPv4Addresses function 6126
TCPIP_DNS_GetIPv6Addresses function 6127
TCPIP_DNS_HANDLE type 6130
TCPIP_DNS_HandlerDeRegister function 6121
TCPIP_DNS_HandlerRegister function 6122
TCPIP_DNS_IsEnabled function 6122
TCPIP_DNS_IsNameResolved function 6123
TCPIP_DNS_IsResolved function 6129
TCPIP_DNS_RemoveAll function 6123
TCPIP_DNS_RemoveEntry function 6124
TCPIP_DNS_Resolve function 6127
TCPIP_DNS_RESOLVE_TYPE enumeration 6134
TCPIP_DNS_RESULT enumeration 6130
TCPIP_DNS_Send_Query function 6128
TCPIP_DNSS_AddressCntGet function 6142
TCPIP_DNSS_CACHE_MAX_SERVER_ENTRIES macro 6139
TCPIP_DNSS_CACHE_PER_IPV4_ADDRESS macro 6139
TCPIP_DNSS_CACHE_PER_IPV6_ADDRESS macro 6140
TCPIP_DNSS_CacheEntryRemove function 6142
TCPIP_DNSS_Disable function 6143
TCPIP_DNSS_Enable function 6143
TCPIP_DNSS_EntryAdd function 6144
TCPIP_DNSS_EntryGet function 6145
TCPIP_DNSS_HOST_NAME_LEN macro 6140
TCPIP_DNSS_IsEnabled function 6144
TCPIP_DNSS_MODULE_CONFIG structure 6146
TCPIP_DNSS_REPLY_BOARD_ADDR macro 6140
TCPIP_DNSS_RESULT enumeration 6146
TCPIP_DNSS_Task function 6145
TCPIP_DNSS_TASK_PROCESS_RATE macro 6140
TCPIP_DNSS_TTL_TIME macro 6140
TCPIP_EMAC_ETH_OPEN_FLAGS macro 6356
TCPIP_EMAC_PHY_ADDRESS macro 6356
TCPIP_EMAC_PHY_CONFIG_FLAGS macro 6357
TCPIP_EMAC_PHY_LINK_INIT_DELAY macro 6357
TCPIP_EMAC_RX_BUFF_SIZE macro 6357
TCPIP_EMAC_RX_DEDICATED_BUFFERS macro 6358
TCPIP_EMAC_RX_DESCRIPTORS macro 6357
TCPIP_EMAC_RX_FILTERS macro 6359
TCPIP_EMAC_RX_FRAGMENTS macro 6357
TCPIP_EMAC_RX_INIT_BUFFERS macro 6358
TCPIP_EMAC_RX_LOW_FILL macro 6359
TCPIP_EMAC_RX_LOW_THRESHOLD macro 6359
TCPIP_EMAC_RX_MAX_FRAME macro 6358
TCPIP_EMAC_TX_DESCRIPTORS macro 6358
TCPIP_EVENT enumeration 6450
TCPIP_EVENT_HANDLE type 6451
TCPIP_FTP_DATA_SKT_RX_BUFF_SIZE macro 6158
TCPIP_FTP_DATA_SKT_TX_BUFF_SIZE macro 6158
TCPIP_FTP_MAX_CONNECTIONS macro 6158
TCPIP_FTP_MODULE_CONFIG structure 6161
TCPIP_FTP_PASSWD_LEN macro 6159
TCPIP_FTP_PUT_ENABLED macro 6159
TCPIP_FTP_ServerTask function 6161
TCPIP_FTP_TIMEOUT macro 6159
TCPIP_FTP_USER_NAME_LEN macro 6159
TCPIP_FTPS_TASK_TICK_RATE macro 6159
TCPIP_Helper_Base64Decode function 6416
TCPIP_Helper_Base64Encode function 6416
TCPIP_Helper_FormatNetBIOSName function 6407
TCPIP_Helper_htonl function 6407
TCPIP_Helper_htonll function 6413
TCPIP_Helper_htons function 6408
TCPIP_Helper_IPAddressToString function 6408
TCPIP_Helper_IPv6AddressToString function 6409
TCPIP_Helper_IsBcastAddress function 6409
TCPIP_Helper_IsMcastAddress function 6410
TCPIP_Helper_IsPrivateAddress function 6410
TCPIP_Helper_MACAddressToString function 6411
TCPIP_Helper_ntohl function 6411
TCPIP_Helper_ntohl macro 6451
TCPIP_Helper_ntohll function 6413
TCPIP_Helper_ntohll macro 6463
TCPIP_Helper_ntohs function 6411
TCPIP_Helper_ntohs macro 6452
TCPIP_Helper_SecurePortGetByIndex function 6413
TCPIP_Helper_SecurePortSet function 6414
TCPIP_Helper_StringToIPAddress function 6412
TCPIP_Helper_StringToIPv6Address function 6412
TCPIP_Helper_StringToMACAddress function 6413
TCPIP_Helper_TCPSecurePortGet function 6415
TCPIP_Helper_UDPSecurePortGet function 6415
tcpip_helpers.h 6465
TCPIP_HTTP_ActiveConnectionCountGet function 6194
TCPIP_HTTP_ArgGet function 6175
TCPIP_HTTP_CACHE_LEN macro 6170
TCPIP_HTTP_CONFIG_FLAGS macro 6170
TCPIP_HTTP_CurrentConnectionByteCountDec function 6176
TCPIP_HTTP_CurrentConnectionByteCountGet function 6176
TCPIP_HTTP_CurrentConnectionByteCountSet function 6177
TCPIP_HTTP_CurrentConnectionCallbackPosGet function 6177
TCPIP_HTTP_CurrentConnectionCallbackPosSet function 6178
TCPIP_HTTP_CurrentConnectionDataBufferGet function 6185
TCPIP_HTTP_CurrentConnectionFileGet function 6186
TCPIP_HTTP_CurrentConnectionHandleGet function 6195
TCPIP_HTTP_CurrentConnectionHasArgsGet function 6179
TCPIP_HTTP_CurrentConnectionHasArgsSet function 6179

TCPIP_HTTP_CurrentConnectionIndexGet function 6195
TCPIP_HTTP_CurrentConnectionIsAuthorizedGet function 6186
TCPIP_HTTP_CurrentConnectionIsAuthorizedSet function 6187
TCPIP_HTTP_CurrentConnectionPostSmGet function 6187
TCPIP_HTTP_CurrentConnectionPostSmSet function 6180
TCPIP_HTTP_CurrentConnectionSocketGet function 6188
TCPIP_HTTP_CurrentConnectionStatusGet function 6181
TCPIP_HTTP_CurrentConnectionStatusSet function 6189
TCPIP_HTTP_CurrentConnectionUserDataGet function 6190
TCPIP_HTTP_CurrentConnectionUserDataSet function 6190
TCPIP_HTTP_DEFAULT_FILE macro 6170
TCPIP_HTTP_DEFAULT_LEN macro 6170
TCPIP_HTTP_DYN_ARG_DCPT structure 6252
TCPIP_HTTP_DYN_ARG_TYPE enumeration 6252
TCPIP_HTTP_DYN_PRINT_RES enumeration 6253
TCPIP_HTTP_DYN_VAR_DCPT structure 6253
TCPIP_HTTP_DYN_VAR_FLAGS enumeration 6254
TCPIP_HTTP_FILE_UPLOAD_ENABLE macro 6173
TCPIP_HTTP_FILE_UPLOAD_NAME macro 6173
TCPIP_HTTP_FileAuthenticate function 6181
TCPIP_HTTP_FileInclude function 6191
TCPIP_HTTP_GetExecute function 6182
TCPIP_HTTP_MAX_CONNECTIONS macro 6170
TCPIP_HTTP_MAX_DATA_LEN macro 6171
TCPIP_HTTP_MAX_HEADER_LEN macro 6171
TCPIP_HTTP_MIN_CALLBACK_FREE macro 6171
TCPIP_HTTP_MODULE_CONFIG structure 6199
TCPIP_HTTP_NET_ActiveConnectionCountGet function 6218
TCPIP_HTTP_NET_ArgGet function 6219
TCPIP_HTTP_NET_CACHE_LEN macro 6208
TCPIP_HTTP_NET_CHUNK_RETRIES macro 6213
TCPIP_HTTP_NET_CHUNKS_NUMBER macro 6213
TCPIP_HTTP_NET_CONFIG_FLAGS macro 6208
TCPIP_HTTP_NET_CONN_HANDLE type 6254
TCPIP_HTTP_NET_ConnectionByteCountDec function 6219
TCPIP_HTTP_NET_ConnectionByteCountGet function 6221
TCPIP_HTTP_NET_ConnectionByteCountSet function 6220
TCPIP_HTTP_NET_ConnectionCallbackPosGet function 6222
TCPIP_HTTP_NET_ConnectionCallbackPosSet function 6221
TCPIP_HTTP_NET_ConnectionDataBufferGet function 6222
TCPIP_HTTP_NET_ConnectionDataBufferSizeGet function 6246
TCPIP_HTTP_NET_ConnectionDiscard function 6223
TCPIP_HTTP_NET_ConnectionDynamicDescriptors function 6246
TCPIP_HTTP_NET_ConnectionFileAuthenticate function 6224
TCPIP_HTTP_NET_ConnectionFileGet function 6224
TCPIP_HTTP_NET_ConnectionFileInclude function 6225
TCPIP_HTTP_NET_ConnectionFlush function 6226
TCPIP_HTTP_NET_ConnectionGetExecute function 6227
TCPIP_HTTP_NET_ConnectionHandleGet function 6250
TCPIP_HTTP_NET_ConnectionHasArgsGet function 6226
TCPIP_HTTP_NET_ConnectionHasArgsSet function 6229
TCPIP_HTTP_NET_ConnectionIndexGet function 6251
TCPIP_HTTP_NET_ConnectionIsAuthorizedGet function 6227
TCPIP_HTTP_NET_ConnectionIsAuthorizedSet function 6230
TCPIP_HTTP_NET_ConnectionNetHandle function 6232
TCPIP_HTTP_NET_ConnectionPeek function 6233
TCPIP_HTTP_NET_ConnectionPostExecute function 6234
TCPIP_HTTP_NET_ConnectionPostNameRead function 6228
TCPIP_HTTP_NET_ConnectionPostReadPair macro 6261
TCPIP_HTTP_NET_ConnectionPostSmGet function 6229
TCPIP_HTTP_NET_ConnectionPostSmSet function 6235
TCPIP_HTTP_NET_ConnectionPostValueRead function 6231
TCPIP_HTTP_NET_ConnectionRead function 6236
TCPIP_HTTP_NET_ConnectionReadBufferSize function 6237
TCPIP_HTTP_NET_ConnectionReadIsReady function 6237
TCPIP_HTTP_NET_ConnectionSocketGet function 6232
TCPIP_HTTP_NET_ConnectionStatusGet function 6233
TCPIP_HTTP_NET_ConnectionStatusSet function 6238
TCPIP_HTTP_NET_ConnectionStringFind function 6238
TCPIP_HTTP_NET_ConnectionUserAuthenticate function 6239
TCPIP_HTTP_NET_ConnectionUserDataGet function 6235
TCPIP_HTTP_NET_ConnectionUserDataSet function 6240
TCPIP_HTTP_NET_COOKIE_BUFFER_SIZE macro 6208
TCPIP_HTTP_NET_DEFAULT_FILE macro 6209
TCPIP_HTTP_NET_DynAcknowledge function 6240
TCPIP_HTTP_NET_DynamicWrite function 6241
TCPIP_HTTP_NET_DynamicWriteString function 6242
TCPIP_HTTP_NET_DynPrint function 6242
TCPIP_HTTP_NET_DYNVAR_ARG_MAX_NUMBER macro 6209
TCPIP_HTTP_NET_DYNVAR_DESCRIPTORS_NUMBER macro 6209
TCPIP_HTTP_NET_DYNVAR_MAX_LEN macro 6209
TCPIP_HTTP_NET_DYNVAR_PROCESS macro 6213
TCPIP_HTTP_NET_DYNVAR_PROCESS_RETRIES macro 6213
TCPIP_HTTP_NET_EVENT_TYPE enumeration 6254
TCPIP_HTTP_NET_EventReport function 6243
TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_RETRIES macro 6213
TCPIP_HTTP_NET_FILE_PROCESS_BUFFER_SIZE macro 6214
TCPIP_HTTP_NET_FILE_PROCESS_BUFFERS_NUMBER macro 6214
TCPIP_HTTP_NET_FILE_UPLOAD_ENABLE macro 6209
TCPIP_HTTP_NET_FILE_UPLOAD_NAME macro 6210
TCPIP_HTTP_NET_FILENAME_MAX_LEN macro 6214
TCPIP_HTTP_NET_FIND_PEEK_BUFF_SIZE macro 6210
TCPIP_HTTP_NET_IO_RESULT enumeration 6256
TCPIP_HTTP_NET_MAX_CONNECTIONS macro 6210
TCPIP_HTTP_NET_MAX_DATA_LEN macro 6210
TCPIP_HTTP_NET_MAX_HEADER_LEN macro 6210
TCPIP_HTTP_NET_MAX_RECURSE_LEVEL macro 6211
TCPIP_HTTP_NET_MODULE_CONFIG structure 6257
TCPIP_HTTP_NET_MODULE_FLAGS enumeration 6258
TCPIP_HTTP_NET_READ_STATUS enumeration 6258
TCPIP_HTTP_NET_RESPONSE_BUFFER_SIZE macro 6211
TCPIP_HTTP_NET_SKT_RX_BUFF_SIZE macro 6211
TCPIP_HTTP_NET_SKT_TX_BUFF_SIZE macro 6211
TCPIP_HTTP_NET_SSI_ATTRIBUTES_MAX_NUMBER macro 6214
TCPIP_HTTP_NET_SSI_CMD_MAX_LEN macro 6214
TCPIP_HTTP_NET_SSI_ECHO_NOT_FOUND_MESSAGE macro 6215
TCPIP_HTTP_NET_SSI_PROCESS macro 6215
TCPIP_HTTP_NET_SSI_STATIC_ATTTRIB_NUMBER macro 6215
TCPIP_HTTP_NET_SSI_VARIABLE_NAME_MAX_LENGTH macro 6215
TCPIP_HTTP_NET_SSI_VARIABLE_STRING_MAX_LENGTH macro 6216
TCPIP_HTTP_NET_SSI_VARIABLES_NUMBER macro 6216
TCPIP_HTTP_NET_SSINotification function 6247
TCPIP_HTTP_NET_SSIVariableDelete function 6248
TCPIP_HTTP_NET_SSIVariableGet function 6248

TCPIP_HTTP_NET_SSIVariableGetByIndex function 6249
TCPIP_HTTP_NET_SSIVariableSet function 6249
TCPIP_HTTP_NET_SSIVariablesNumberGet function 6250
TCPIP_HTTP_NET_STATUS enumeration 6258
TCPIP_HTTP_NET_Task function 6244
TCPIP_HTTP_NET_TASK_RATE macro 6212
TCPIP_HTTP_NET_TIMEOUT macro 6212
TCPIP_HTTP_NET_URLDecode function 6244
TCPIP_HTTP_NET_USE_AUTHENTICATION macro 6212
TCPIP_HTTP_NET_USE_COOKIES macro 6212
TCPIP_HTTP_NET_USE_POST macro 6212
TCPIP_HTTP_NET_USER_CALLBACK structure 6259
TCPIP_HTTP_NET_USER_HANDLE type 6260
TCPIP_HTTP_NET_UserHandlerDeregister function 6245
TCPIP_HTTP_NET_UserHandlerRegister function 6245
TCPIP_HTTP_NO_AUTH_WITHOUT_SSL macro 6173
TCPIP_HTTP_PostExecute function 6183
TCPIP_HTTP_PostNameRead function 6191
TCPIP_HTTP_PostReadPair macro 6193
TCPIP_HTTP_PostValueRead function 6192
TCPIP_HTTP_Print_varname function 6184
TCPIP_HTTP_SKT_RX_BUFF_SIZE macro 6171
TCPIP_HTTP_SKT_TX_BUFF_SIZE macro 6171
TCPIP_HTTP_SSI_ATTR_DCPT structure 6261
TCPIP_HTTP_SSI_NOTIFY_DCPT structure 6262
TCPIP_HTTP_Task function 6194
TCPIP_HTTP_TASK_RATE macro 6173
TCPIP_HTTP_TIMEOUT macro 6172
TCPIP_HTTP_URLDecode function 6193
TCPIP_HTTP_USE_AUTHENTICATION macro 6172
TCPIP_HTTP_USE_COOKIES macro 6172
TCPIP_HTTP_USE_POST macro 6172
TCPIP_HTTP_UserAuthenticate function 6184
TCPIP_HTTPS_DEFAULT_FILE macro 6172
TCPIP_ICMP_CallbackDeregister function 6270
TCPIP_ICMP_CallbackRegister function 6270
TCPIP_ICMP_CLIENT_USER_NOTIFICATION macro 6268
TCPIP_ICMP_ECHO_REQUEST structure 6275
TCPIP_ICMP_ECHO_REQUEST_RESULT enumeration 6276
TCPIP_ICMP_ECHO_REQUEST_TIMEOUT macro 6268
TCPIP_ICMP_EchoRequest function 6273
TCPIP_ICMP_EchoRequestCancel function 6274
TCPIP_ICMP_EchoRequestSend function 6271
TCPIP_ICMP_MODULE_CONFIG structure 6275
TCPIP_ICMP_REQUEST_HANDLE type 6276
TCPIP_ICMP_Task function 6272
TCPIP_ICMP_TASK_TICK_RATE macro 6268
TCPIP_ICMPV6_CallbackDeregister function 6281
TCPIP_ICMPV6_CallbackRegister function 6283
TCPIP_ICMPV6_CLIENT_USER_NOTIFICATION macro 6288
TCPIP_ICMPV6_EchoRequestSend function 6282
TCPIP_ICMPV6_Flush function 6281
TCPIP_ICMPV6_HeaderEchoRequestPut function 6282
TCPIP_ICMPV6_PutHeaderEchoReply macro 6284
TCPIP_IPERF_MAX_INSTANCES macro 6296
TCPIP_IPERF_RX_BUFFER_SIZE macro 6295
TCPIP_IPERF_TIMING_ERROR_MARGIN macro 6296
TCPIP_IPERF_TX_BUFFER_SIZE macro 6296
TCPIP_IPERF_TX_QUEUE_LIMIT macro 6296
TCPIP_IPERF_TX_WAIT_TMO macro 6296
TCPIP_IPV4_FILTER_TYPE enumeration 6309
TCPIP_IPV4_FRAGMENT_MAX_NUMBER macro 6301
TCPIP_IPV4_FRAGMENT_MAX_STREAMS macro 6301
TCPIP_IPV4_FRAGMENT_TIMEOUT macro 6302
TCPIP_IPV4_FRAGMENTATION macro 6302
TCPIP_IPV4_MacPacketTransmit function 6307
TCPIP_IPV4_MODULE_CONFIG structure 6310
TCPIP_IPV4_PacketFilterClear function 6306
TCPIP_IPV4_PacketFilterSet function 6306
TCPIP_IPV4_PacketFormatTx function 6307
TCPIP_IPV4_PacketGetDestAddress function 6303
TCPIP_IPV4_PacketGetSourceAddress function 6304
TCPIP_IPV4_PacketTransmit function 6304
TCPIP_IPV4_SelectSourceInterface function 6305
TCPIP_IPV4_Task function 6305
TCPIP_IPV4_TASK_TICK_RATE macro 6302
TCPIP_IPV6_AddressUnicastRemove function 6320
TCPIP_IPV6_ArrayGet function 6320
TCPIP_IPV6_ArrayPutHelper function 6321
TCPIP_IPV6_DASSourceAddressSelect function 6321
TCPIP_IPV6_DEFAULT_ALLOCATION_BLOCK_SIZE macro 6314
TCPIP_IPV6_DEFAULT_BASE_REACHABLE_TIME macro 6314
TCPIP_IPV6_DEFAULT_CUR_HOP_LIMIT macro 6315
TCPIP_IPV6_DEFAULT_LINK_MTU macro 6315
TCPIP_IPV6_DEFAULT_RETRANSMIT_TIME macro 6315
TCPIP_IPV6_DefaultRouterDelete function 6332
TCPIP_IPV6_DefaultRouterGet function 6333
TCPIP_IPV6_DestAddressGet function 6322
TCPIP_IPV6_DestAddressSet function 6322
TCPIP_IPV6_Flush function 6323
TCPIP_IPV6_FRAGMENT_PKT_TIMEOUT macro 6315
TCPIP_IPV6_Get function 6323
TCPIP_IPV6_HandlerDeregister function 6324
TCPIP_IPV6_HandlerRegister function 6324
TCPIP_IPV6_INIT_TASK_PROCESS_RATE macro 6315
TCPIP_IPV6_InterfaceIsReady function 6325
TCPIP_IPV6_MINIMUM_LINK_MTU macro 6316
TCPIP_IPV6_MODULE_CONFIG structure 6347
TCPIP_IPV6_MTU_INCREASE_TIMEOUT macro 6474
TCPIP_IPV6_MulticastListenerAdd function 6325
TCPIP_IPV6_MulticastListenerRemove function 6326
TCPIP_IPV6_NDP_DELAY_FIRST_PROBE_TIME macro 6474
TCPIP_IPV6_NDP_MAX_ANYCAST_DELAY_TIME macro 6474
TCPIP_IPV6_NDP_MAX_MULTICAST_SOLICIT macro 6474
TCPIP_IPV6_NDP_MAX_NEIGHBOR_ADVERTISEMENT macro 6475
TCPIP_IPV6_NDP_MAX_RTR_SOLICITATION_DELAY macro 6475
TCPIP_IPV6_NDP_MAX_RTR_SOLICITATIONS macro 6475
TCPIP_IPV6_NDP_MAX_UNICAST_SOLICIT macro 6475
TCPIP_IPV6_NDP_REACHABLE_TIME macro 6475
TCPIP_IPV6_NDP_RETRANS_TIMER macro 6476
TCPIP_IPV6_NDP_RTR_SOLICITATION_INTERVAL macro 6476
TCPIP_IPV6_NDP_TASK_TIMER_RATE macro 6476
TCPIP_IPV6_NDP_VALID_LIFETIME_TWO_HOURS macro 6476
TCPIP_IPV6_NEIGHBOR_CACHE_ENTRY_STALE_TIMEOUT macro 6316
TCPIP_IPV6_PacketFree function 6326

TCPIP_IPV6_PayloadSet function 6327
TCPIP_IPV6_Put function 6328
TCPIP_IPV6_PutArray macro 6338
TCPIP_IPV6_QUEUE_MCAST_PACKET_LIMIT macro 6316
TCPIP_IPV6_QUEUE_NEIGHBOR_PACKET_LIMIT macro 6316
TCPIP_IPV6_QUEUED_MCAST_PACKET_TIMEOUT macro 6316
TCPIP_IPV6_RouterAddressAdd function 6332
TCPIP_IPV6_RX_FRAGMENTED_BUFFER_SIZE macro 6317
TCPIP_IPV6_SourceAddressGet function 6328
TCPIP_IPV6_SourceAddressSet function 6329
TCPIP_IPV6_Task function 6333
TCPIP_IPV6_TASK_PROCESS_RATE macro 6317
TCPIP_IPV6_TxIsPutReady function 6329
TCPIP_IPV6_TxPacketAllocate function 6330
TCPIP_IPV6_ULA_NTP_ACCESS_TMO macro 6317
TCPIP_IPV6_ULA_NTP_VALID_WINDOW macro 6317
TCPIP_IPV6_UnicastAddressAdd function 6331
TCPIP_IPV6_UniqueLocalUnicastAddressAdd function 6330
TCPIP_LOCAL_MASK_TYPE enumeration 6449
tcpip_mac.h 6398
TCPIP_MAC_ACTION enumeration 6373
TCPIP_MAC_ADDR structure 6374
TCPIP_MAC_Close function 6362
tcpip_mac_config.h 6399
TCPIP_MAC_ConfigGet function 6368
TCPIP_MAC_DATA_SEGMENT structure 6374
TCPIP_MAC_Deinitialize function 6363
TCPIP_MAC_ETHERNET_HEADER structure 6376
TCPIP_MAC_EVENT enumeration 6376
TCPIP_MAC_EventAcknowledge function 6369
TCPIP_MAC_EventF type 6377
TCPIP_MAC_EventMaskSet function 6369
TCPIP_MAC_EventPendingGet function 6370
TCPIP_MAC_HANDLE type 6378
TCPIP_MAC_HEAP_CallocF type 6378
TCPIP_MAC_HEAP_CallocFDbg type 6393
TCPIP_MAC_HEAP_FreeF type 6379
TCPIP_MAC_HEAP_FreeFDbg type 6393
TCPIP_MAC_HEAP_HANDLE type 6379
TCPIP_MAC_HEAP_MallocF type 6379
TCPIP_MAC_HEAP_MallocFDbg type 6393
TCPIP_MAC_INIT structure 6391
TCPIP_MAC_Initialize function 6362
TCPIP_MAC_LINK_MTU enumeration 6397
TCPIP_MAC_LinkCheck function 6365
TCPIP_MAC_MODULE_CTRL structure 6380
TCPIP_MAC_Open function 6361
TCPIP_MAC_PACKET type 6380
TCPIP_MAC_PACKET_ACK_FUNC type 6382
TCPIP_MAC_PACKET_FLAGS enumeration 6383
TCPIP_MAC_PACKET_RX_STAT structure 6383
TCPIP_MAC_PacketRx function 6372
TCPIP_MAC_PacketTx function 6371
TCPIP_MAC_PARAMETERS structure 6391
TCPIP_MAC_ParametersGet function 6364
TCPIP_MAC_PKT_ACK_RES enumeration 6384
TCPIP_MAC_PKT_AckF type 6385
TCPIP_MAC_PKT_AckFDbg type 6392
TCPIP_MAC_PKT_AllocF type 6386
TCPIP_MAC_PKT_AllocFDbg type 6392
TCPIP_MAC_PKT_FreeF type 6386
TCPIP_MAC_PKT_FreeFDbg type 6392
TCPIP_MAC_POWER_MODE enumeration 6387
TCPIP_MAC_Process function 6367
TCPIP_MAC_PROCESS_FLAGS enumeration 6387
TCPIP_MAC_RegisterStatisticsGet function 6367
TCPIP_MAC_Reinitialize function 6363
TCPIP_MAC_RES enumeration 6387
TCPIP_MAC_RX_FILTER_TYPE enumeration 6396
TCPIP_MAC_RX_STATISTICS structure 6390
TCPIP_MAC_RxFilterHashTableEntrySet function 6371
TCPIP_MAC_SEGMENT_FLAGS enumeration 6388
TCPIP_MAC_STATISTICS_REG_ENTRY structure 6393
TCPIP_MAC_StatisticsGet function 6364
TCPIP_MAC_Status function 6365
TCPIP_MAC_SYNCH_REQUEST enumeration 6394
TCPIP_MAC_SynchReqF type 6395
TCPIP_MAC_Tasks function 6366
TCPIP_MAC_TX_STATISTICS structure 6390
TCPIP_MAC_TYPE enumeration 6394
tcpip_manager.h 6464
TCPIP_MDNS_ServiceDeregister function 6670
TCPIP_MDNS_ServiceRegister function 6670
TCPIP_MDNS_ServiceUpdate function 6671
TCPIP_MDNS_Task function 6672
TCPIP_MODULE_MAC_ID enumeration 6395
TCPIP_MODULE_MAC_MRF24W_CONFIG structure 6389
TCPIP_MODULE_MAC_MRF24WN_CONFIG structure 6396
TCPIP_MODULE_MAC_PIC32INT_CONFIG structure 6389
TCPIP_MODULE_SIGNAL enumeration 6459
TCPIP_MODULE_SIGNAL_FUNC type 6460
TCPIP_MODULE_SIGNAL_HANDLE type 6460
TCPIP_MODULE_SignalFunctionDeregister function 6424
TCPIP_MODULE_SignalFunctionRegister function 6424
TCPIP_MODULE_SignalGet function 6425
TCPIP_NBNS_MODULE_CONFIG structure 6470
TCPIP_NBNS_Task function 6470
TCPIP_NBNS_TASK_TICK_RATE macro 6468
TCPIP_NDP_NborReachConfirm function 6477
TCPIP_NET_HANDLE type 6449
TCPIP_NETWORK_CONFIG structure 6456
TCPIP_NETWORK_CONFIG_FLAGS enumeration 6457
TCPIP_NTP_DEFAULT_CONNECTION_TYPE macro 6567
TCPIP_NTP_DEFAULT_IF macro 6567
TCPIP_NTP_EPOCH macro 6567
TCPIP_NTP_FAST_QUERY_INTERVAL macro 6567
TCPIP_NTP_MAX_STRATUM macro 6567
TCPIP_NTP_QUERY_INTERVAL macro 6568
TCPIP_NTP_REPLY_TIMEOUT macro 6568
TCPIP_NTP_RX_QUEUE_LIMIT macro 6568
TCPIP_NTP_SERVER macro 6568
TCPIP_NTP_SERVER_MAX_LENGTH macro 6568
TCPIP_NTP_TASK_TICK_RATE macro 6569
TCPIP_NTP_TIME_STAMP_TMO macro 6569
TCPIP_NTP_VERSION macro 6569
tcpip_reboot_config.h 6482

TCPIP_REBOOT_MESSAGE macro 6480
TCPIP_REBOOT_SAME_SUBNET_ONLY macro 6480
TCPIP_REBOOT_TASK_TICK_RATE macro 6481
TCPIP_SMTP_ArrayPut function 6488
TCPIP_SMTP_CLIENT_MESSAGE structure 6493
TCPIP_SMTP_CLIENT_MODULE_CONFIG structure 6493
TCPIP_SMTP_ClientTask function 6492
TCPIP_SMTP_Flush function 6488
TCPIP_SMTP_IsBusy function 6489
TCPIP_SMTP_IsPutReady function 6489
TCPIP_SMTP_MailSend function 6491
TCPIP_SMTP_MAX_WRITE_SIZE macro 6486
TCPIP_SMTP_Put function 6489
TCPIP_SMTP_PutIsDone function 6490
TCPIP_SMTP_SERVER_REPLY_TIMEOUT macro 6486
TCPIP_SMTP_StringPut function 6490
TCPIP_SMTP_TASK_TICK_RATE macro 6486
TCPIP_SMTP_UsageBegin function 6491
TCPIP_SMTP_UsageEnd function 6491
TCPIP_SMTP_WRITE_READY_SPACE macro 6486
TCPIP_SMTPC_ATTACH_BUFFER structure 6504
TCPIP_SMTPC_ATTACH_FILE structure 6504
TCPIP_SMTPC_ATTACH_TYPE enumeration 6505
TCPIP_SMTPC_CLIENT_ADDR_BUFFER_SIZE macro 6498
TCPIP_SMTPC_CLIENT_AUTH_BUFFER_SIZE macro 6498
TCPIP_SMTPC_CLIENT_MESSAGE_DATE macro 6498
TCPIP_SMTPC_ENCODE_TYPE enumeration 6505
TCPIP_SMTPC_INTERNAL_RETRY_TIMEOUT macro 6498
TCPIP_SMTPC_MAIL_CONNECTIONS macro 6498
TCPIP_SMTPC_MAIL_FLAGS enumeration 6506
TCPIP_SMTPC_MAIL_MESSAGE structure 6507
TCPIP_SMTPC_MAIL_RETRIES macro 6499
TCPIP_SMTPC_MailMessage function 6502
TCPIP_SMTPC_MESSAGE_CALLBACK type 6508
TCPIP_SMTPC_MESSAGE_HANDLE type 6508
TCPIP_SMTPC_MESSAGE_QUERY structure 6509
TCPIP_SMTPC_MESSAGE_REPORT structure 6509
TCPIP_SMTPC_MESSAGE_RESULT enumeration 6510
TCPIP_SMTPC_MESSAGE_STATUS enumeration 6511
TCPIP_SMTPC_MESSAGE_WARNING enumeration 6512
TCPIP_SMTPC_MessageQuery function 6503
TCPIP_SMTPC_MODULE_CONFIG structure 6513
TCPIP_SMTPC_PLAIN_LINE_BUFF_SIZE macro 6499
TCPIP_SMTPC_SERVER_DATA_TIMEOUT macro 6499
TCPIP_SMTPC_SERVER_REPLY_BUFFER_SIZE macro 6499
TCPIP_SMTPC_SERVER_REPLY_CALLBACK type 6513
TCPIP_SMTPC_SERVER_REPLY_TIMEOUT macro 6500
TCPIP_SMTPC_SERVER_TRANSIENT_RETRY_TIMEOUT macro 6500
TCPIP_SMTPC_SKT_RX_BUFF_SIZE macro 6500
TCPIP_SMTPC_SKT_TX_BUFF_SIZE macro 6500
TCPIP_SMTPC_Task function 6503
TCPIP_SMTPC_TASK_TICK_RATE macro 6500
TCPIP_SMTPC_TLS_HANDSHAKE_TIMEOUT macro 6501
TCPIP_SMTPC_USE_MAIL_COMMAND macro 6501
TCPIP_SNMP_AuthTrapFlagGet function 6542
TCPIP_SNMP_AuthTrapFlagSet function 6542
TCPIP_SNMP_BIB_FILE_NAME macro 6518
TCPIP_SNMP_ClientGetNet function 6530
TCPIP_SNMP_COMMUNITY_CONFIG structure 6559
TCPIP_SNMP_COMMUNITY_MAX_LEN macro 6519
TCPIP_SNMP_ExactIndexGet function 6531
TCPIP_SNMP_IsTrapEnabled function 6543
TCPIP_SNMP_IsValidCommunity function 6531
TCPIP_SNMP_IsValidLength function 6532
TCPIP_SNMP_MAX_COMMUNITY_SUPPORT macro 6519
TCPIP_SNMP_MAX_MSG_SIZE macro 6519
TCPIP_SNMP_MAX_NON_REC_ID_OID macro 6519
TCPIP_SNMP_MibIDSet function 6532
TCPIP_SNMP_MODULE_CONFIG structure 6559
TCPIP_SNMP_NextIndexGet function 6533
TCPIP_SNMP_NOTIFY_COMMUNITY_LEN macro 6519
TCPIP_SNMP_NotifyIsReady function 6529
TCPIP_SNMP_NotifyPrepare function 6529
TCPIP_SNMP_OID_MAX_LEN macro 6520
TCPIP_SNMP_ReadCommunityGet function 6533
TCPIP_SNMP_ReadCommunitySet function 6546
TCPIP_SNMP_RecordIDValidation function 6534
TCPIP_SNMP_SendFailureTrap function 6535
TCPIP_SNMP_SocketIDGet function 6547
TCPIP_SNMP_SocketIDSet function 6547
TCPIP_SNMP_Task function 6548
TCPIP_SNMP_TASK_PROCESS_RATE macro 6520
TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN macro 6520
TCPIP_SNMP_TRAP_COMMUNITY_MAX_LEN_MEM_USE macro 6520
TCPIP_SNMP_TRAP_TABLE_SIZE macro 6520
TCPIP_SNMP_TrapInterFaceSet function 6535
TCPIP_SNMP_TRAPMibIDGet function 6536
TCPIP_SNMP_TrapSendFlagGet function 6536
TCPIP_SNMP_TrapSendFlagSet function 6537
TCPIP_SNMP_TrapSpecificNotificationGet function 6537
TCPIP_SNMP_TrapSpecificNotificationSet function 6538
TCPIP_SNMP_TrapTimeGet function 6530
TCPIP_SNMP_TRAPTypeGet function 6543
TCPIP_SNMP_TRAPv1Notify function 6544
TCPIP_SNMP_TRAPv2Notify function 6544
TCPIP_SNMP_ValidateTrapIntf function 6545
TCPIP_SNMP_VarbindGet function 6538
TCPIP_SNMP_VarbindSet function 6540
TCPIP_SNMP_WriteCommunityGet function 6541
TCPIP_SNMP_WriteCommunitySet function 6546
TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN macro 6521
TCPIP_SNMPV3_AUTH_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE
macro 6521
TCPIP_SNMPV3_EngineUserDataBaseGet function 6548
TCPIP_SNMPV3_EngineUserDataBaseSet function 6549
TCPIP_SNMPv3_Notify function 6551
TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN macro 6521
TCPIP_SNMPV3_PRIV_LOCALIZED_PASSWORD_KEY_LEN_MEM_USE
macro 6521
TCPIP_SNMPV3_TARGET_ENTRY_CONFIG structure 6560
TCPIP_SNMPv3_TrapConfigDataGet function 6550
TCPIP_SNMPV3_TrapTypeGet function 6545

TCPIP_SNMPV3_USER_SECURITY_NAME_LEN macro 6521
TCPIP_SNMPV3_USER_SECURITY_NAME_LEN_MEM_USE macro 6522
TCPIP_SNMPV3_USERDATABASECONFIG_TYPE enumeration 6558
TCPIP_SNMPV3_USM_MAX_USER macro 6522
TCPIP_SNMPV3_USM_USER_CONFIG structure 6560
TCPIP_SNTP_ConnectionInitiate function 6570
TCPIP_SNTP_ConnectionParamSet function 6572
TCPIP_SNTP_LastErrorGet function 6571
TCPIP_SNTP_MODULE_CONFIG structure 6573
TCPIP_SNTP_RESULT enumeration 6573
TCPIP_SNTP_Task function 6573
TCPIP_SNTP_TimeStampGet function 6571
TCPIP_SNTP_UTCSecondsGet function 6571
TCPIP_STACK_Deinitialize function 6421
TCPIP_STACK_EVENT_HANDLER type 6451
TCPIP_STACK_EventsPendingGet function 6431
TCPIP_STACK_HandlerDeregister function 6419
TCPIP_STACK_HandlerRegister function 6419
TCPIP_STACK_IF_NAME_97J60 macro 6461
TCPIP_STACK_IF_NAME_ALIAS_ETH macro 6452
TCPIP_STACK_IF_NAME_ALIAS_UNK macro 6452
TCPIP_STACK_IF_NAME_ALIAS_WLAN macro 6452
TCPIP_STACK_IF_NAME_ENCJ60 macro 6461
TCPIP_STACK_IF_NAME_ENCJ600 macro 6461
TCPIP_STACK_IF_NAME_MRF24W macro 6461
TCPIP_STACK_IF_NAME_MRF24WN macro 6464
TCPIP_STACK_IF_NAME_NONE macro 6461
TCPIP_STACK_IF_NAME_PIC32INT macro 6462
TCPIP_STACK_IF_POWER_DOWN macro 6462
TCPIP_STACK_IF_POWER_FULL macro 6462
TCPIP_STACK_IF_POWER_LOW macro 6462
TCPIP_STACK_IF_POWER_NONE macro 6462
TCPIP_STACK_IndexToNet function 6426
TCPIP_STACK_INIT structure 6457
TCPIP_STACK_Initialize function 6420
TCPIP_STACK_InitializeDataGet function 6434
TCPIP_STACK_MACObjectGet function 6423
TCPIP_STACK_MODULE enumeration 6458
TCPIP_STACK_MODULE_CONFIG structure 6459
TCPIP_STACK_ModuleConfigGet function 6444
TCPIP_STACK_NetAddress function 6436
TCPIP_STACK_NetAddressBcast function 6437
TCPIP_STACK_NetAddressDnsPrimary function 6438
TCPIP_STACK_NetAddressDnsPrimarySet function 6438
TCPIP_STACK_NetAddressDnsSecond function 6439
TCPIP_STACK_NetAddressDnsSecondSet function 6439
TCPIP_STACK_NetAddressGateway function 6440
TCPIP_STACK_NetAddressGatewaySet function 6441
TCPIP_STACK_NetAddressMac function 6441
TCPIP_STACK_NetAddressMacSet function 6442
TCPIP_STACK_NetAddressSet function 6443
TCPIP_STACK_NetAliasNameGet function 6433
TCPIP_STACK_NetBIOSName function 6426
TCPIP_STACK_NetBiosNameSet function 6427
TCPIP_STACK_NetConfigGet function 6447
TCPIP_STACK_NetConfigSet function 6448
TCPIP_STACK_NetDefaultGet function 6427
TCPIP_STACK_NetDefaultSet function 6428
TCPIP_STACK_NetDown function 6436
TCPIP_STACK_NetHandleGet function 6428
TCPIP_STACK_NetIndexGet function 6429
TCPIP_STACK_NetIPv6AddressGet function 6443
TCPIP_STACK_NetIsLinked function 6434
TCPIP_STACK_NetIsReady function 6432
TCPIP_STACK_NetIsUp function 6435
TCPIP_STACK_NetMACIdGet function 6445
TCPIP_STACK_NetMACRegisterStatisticsGet function 6432
TCPIP_STACK_NetMACStatisticsGet function 6446
TCPIP_STACK_NetMask function 6429
TCPIP_STACK_NetNameGet function 6430
TCPIP_STACK_NetUp function 6435
TCPIP_STACK_NumberOfNetworksGet function 6431
TCPIP_STACK_SetLocalMasks function 6417
TCPIP_STACK_SetLocalMasksType function 6418
TCPIP_STACK_Status function 6446
TCPIP_STACK_Task function 6421
TCPIP_STACK_USE_BASE64_DECODE macro 6169
TCPIP_STACK_USE_DHCPV6_CLIENT macro 6101
TCPIP_STACK_VERSION_MAJOR macro 6463
TCPIP_STACK_VERSION_MINOR macro 6463
TCPIP_STACK_VERSION_PATCH macro 6463
TCPIP_STACK_VERSION_STR macro 6463
TCPIP_STACK_VersionGet function 6422
TCPIP_STACK_VersionStrGet function 6422
TCPIP_TCP_Abort function 6591
TCPIP_TCP_ArrayFind function 6602
TCPIP_TCP_ArrayGet function 6601
TCPIP_TCP_ArrayPeek function 6602
TCPIP_TCP_ArrayPut function 6599
TCPIP_TCP_AUTO_TRANSMIT_TIMEOUT_VAL macro 6580
TCPIP_TCP_Bind function 6588
TCPIP_TCP_ClientOpen function 6586
TCPIP_TCP_Close function 6587
TCPIP_TCP_CLOSE_WAIT_TIMEOUT macro 6580
TCPIP_TCP_Connect function 6587
TCPIP_TCP_DELAYED_ACK_TIMEOUT macro 6580
TCPIP_TCP_Discard function 6605
TCPIP_TCP_Disconnect function 6590
TCPIP_TCP_DYNAMIC_OPTIONS macro 6583
TCPIP_TCP_FifoRxFreeGet function 6605
TCPIP_TCP_FifoRxFullGet macro 6606
TCPIP_TCP_FifoSizeAdjust function 6605
TCPIP_TCP_FifoTxFreeGet function 6600
TCPIP_TCP_FifoTxFullGet function 6600
TCPIP_TCP_FIN_WAIT_2_TIMEOUT macro 6580
TCPIP_TCP_Find function 6603
TCPIP_TCP_Flush function 6599
TCPIP_TCP_Get function 6604
TCPIP_TCP_GetIsReady function 6601
TCPIP_TCP_IsConnected function 6589
TCPIP_TCP_IsReady function 6596
TCPIP_TCP_KEEP_ALIVE_TIMEOUT macro 6580
TCPIP_TCP_MAX_RETRIES macro 6581
TCPIP_TCP_MAX_SEG_SIZE_RX_LOCAL macro 6581
TCPIP_TCP_MAX_SEG_SIZE_RX_NON_LOCAL macro 6581

TCPIP_TCP_MAX_SEG_SIZE_TX macro 6581
TCPIP_TCP_MAX_SOCKETS macro 6581
TCPIP_TCP_MAX_SYN_RETRIES macro 6582
TCPIP_TCP_MAX_UNACKED_KEEP_ALIVES macro 6582
TCPIP_TCP_MODULE_CONFIG structure 6610
TCPIP_TCP_MSL_TIMEOUT macro 6583
TCPIP_TCP_OptionsGet function 6591
TCPIP_TCP_OptionsSet function 6592
TCPIP_TCP_Peek function 6604
TCPIP_TCP_Put function 6597
TCPIP_TCP_PutIsReady function 6598
TCPIP_TCP_QUIET_TIME macro 6584
TCPIP_TCP_RemoteBind function 6589
TCPIP_TCP_ServerOpen function 6586
TCPIP_TCP_SIGNAL_FUNCTION type 6611
TCPIP_TCP_SIGNAL_HANDLE type 6611
TCPIP_TCP_SIGNAL_TYPE enumeration 6612
TCPIP_TCP_SignalHandlerDeregister function 6594
TCPIP_TCP_SignalHandlerRegister function 6595
TCPIP_TCP_SOCKET_DEFAULT_RX_SIZE macro 6582
TCPIP_TCP_SOCKET_DEFAULT_TX_SIZE macro 6582
TCPIP_TCP_SocketInfoGet function 6593
TCPIP_TCP_SocketNetGet function 6593
TCPIP_TCP_SocketNetSet function 6594
TCPIP_TCP_SocketsNumberGet function 6596
TCPIP_TCP_SocketTraceSet function 6597
TCPIP_TCP_START_TIMEOUT_VAL macro 6583
TCPIP_TCP_STATE enumeration 6612
TCPIP_TCP_StringPut function 6598
TCPIP_TCP_Task function 6596
TCPIP_TCP_TASK_TICK_RATE macro 6583
TCPIP_TCP_WasReset function 6590
TCPIP_TCP_WINDOW_UPDATE_TIMEOUT_VAL macro 6583
TCPIP_TELNET_MAX_CONNECTIONS macro 6630
TCPIP_TELNET_MODULE_CONFIG structure 6632
TCPIP_TELNET_PASSWORD macro 6630
TCPIP_TELNET_Task function 6632
TCPIP_TELNET_TASK_TICK_RATE macro 6630
TCPIP_TELNET_USERNAME macro 6631
TCPIP_TFTP_CMD_TYPE enumeration 6624
TCPIP_TFTPC_ARP_TIMEOUT macro 6618
TCPIP_TFTPC_CMD_PROCESS_TIMEOUT macro 6619
TCPIP_TFTPC_DEFAULT_IF macro 6619
TCPIP_TFTPC_EVENT_HANDLER type 6625
TCPIP_TFTPC_EVENT_TYPE enumeration 6625
TCPIP_TFTPC_FILENAME_LEN macro 6619
TCPIP_TFTPC_GetEventNotification function 6621
TCPIP_TFTPC_HANDLE type 6626
TCPIP_TFTPC_HandlerDeRegister function 6622
TCPIP_TFTPC_HandlerRegister function 6622
TCPIP_TFTPC_HOSTNAME_LEN macro 6619
TCPIP_TFTPC_MAX_RETRIES macro 6619
TCPIP_TFTPC_MODULE_CONFIG structure 6624
TCPIP_TFTPC_OPERATION_RESULT enumeration 6626
TCPIP_TFTPC_SetCommand function 6623
TCPIP_TFTPC_SetServerAddress function 6623
TCPIP_TFTPC_Task function 6621
TCPIP_TFTPC_TASK_TICK_RATE macro 6620
TCPIP_TFTPC_USER_NOTIFICATION macro 6620
TCPIP_UDP_ArrayGet function 6658
TCPIP_UDP_ArrayPut function 6655
TCPIP_UDP_BcastIPV4AddressSet function 6649
TCPIP_UDP_Bind function 6643
TCPIP_UDP_ClientOpen function 6642
TCPIP_UDP_Close function 6645
TCPIP_UDP_DestinationIPAddressSet function 6650
TCPIP_UDP_DestinationPortSet function 6651
TCPIP_UDP_Discard function 6659
TCPIP_UDP_Disconnect function 6651
TCPIP_UDP_Flush function 6657
TCPIP_UDP_Get function 6658
TCPIP_UDP_GetIsReady function 6657
TCPIP_UDP_IsConnected function 6643
TCPIP_UDP_IsOpened macro 6642
TCPIP_UDP_MAX_SOCKETS macro 6637
TCPIP_UDP_MODULE_CONFIG structure 6663
TCPIP_UDP_OptionsGet function 6645
TCPIP_UDP_OptionsSet function 6646
TCPIP_UDP_Put function 6656
TCPIP_UDP_PutIsReady function 6654
TCPIP_UDP_RemoteBind function 6644
TCPIP_UDP_RxOffsetSet function 6659
TCPIP_UDP_ServerOpen function 6641
TCPIP_UDP_SIGNAL_FUNCTION type 6663
TCPIP_UDP_SIGNAL_HANDLE type 6663
TCPIP_UDP_SIGNAL_TYPE enumeration 6664
TCPIP_UDP_SignalHandlerDeregister function 6652
TCPIP_UDP_SignalHandlerRegister function 6652
TCPIP_UDP_SOCKET_DEFAULT_RX_QUEUE_LIMIT macro 6638
TCPIP_UDP_SOCKET_DEFAULT_TX_QUEUE_LIMIT macro 6638
TCPIP_UDP_SOCKET_DEFAULT_TX_SIZE macro 6638
TCPIP_UDP_SOCKET_POOL_BUFFER_SIZE macro 6638
TCPIP_UDP_SOCKET_POOL_BUFFERS macro 6638
TCPIP_UDP_SocketInfoGet function 6647
TCPIP_UDP_SocketNetGet function 6647
TCPIP_UDP_SocketNetSet function 6648
TCPIP_UDP_SocketsNumberGet function 6653
TCPIP_UDP_SourceIPAddressSet function 6649
TCPIP_UDP_StringPut function 6655
TCPIP_UDP_Task function 6653
TCPIP_UDP_TxCountGet function 6656
TCPIP_UDP_TxOffsetSet function 6648
TCPIP_UDP_TxPutIsReady function 6654
TCPIP_UDP_USE_POOL_BUFFERS macro 6639
TCPIP_UDP_USE_RX_CHECKSUM macro 6639
TCPIP_UDP_USE_TX_CHECKSUM macro 6639
TCPIP_ZCLL_Disable function 6672
TCPIP_ZCLL_Enable function 6673
TCPIP_ZCLL_IsEnabled function 6673
TCPIP_ZCLL_Task function 6674
Telnet Module 6628
telnet.h 6633
telnet_config.h 6633
Test Harness Library 6677
Test Libraries Help 6677
test_config_template.h 6696

TEST_DATA structure 6691
test_harness.h 6695
TEST_HARNESS_MAX_NUM_INSTANCES_PER_LIBRARY macro 6679
TEST_HARNESS_MAX_NUM_TASKS_PER_LIBRARY macro 6679
TEST_HarnessTasks function 6686
TEST_HasCompleted function 6689
TEST_HasPassedSubtest function 6689
TEST_IDLE_SLEEP_MS macro 6679
TEST_IDLE_SLEEP_MS_LIBRARY macro 6680
TEST_INIT_DATA structure 6692
TEST_Initialize function 6685
TEST_LibraryInitialize function 6690
TEST_LibraryTasksISR function 6687
TEST_LibraryTasksPolled function 6686
TEST_MODULE_INITIALIZE_ROUTINE type 6692
TEST_MODULE_START_FUNCTION type 6693
TEST_MODULE_STATUS_ROUTINE type 6694
TEST_MODULE_TASKS_ROUTINE type 6694
TEST_PlatformInitialize function 6688
TEST_PlatformReady function 6688
TEST_Tasks function 6685
TEST_TIMER_CLOCK_PRESCALER macro 6680
TEST_TIMER_CLOCK_SOURCE macro 6680
TEST_TIMER_ID macro 6681
TEST_TIMER_INCREMENT_PERIOD macro 6681
TEST_TIMER_INTERRUPT_PRIORITY macro 6681
TEST_TIMER_INTERRUPT_SOURCE macro 6682
TEST_TIMER_INTERRUPT_SUBPRIORITY macro 6682
TEST_TIMER_INTERRUPT_VECTOR macro 6682
TEST_TIMER_MS_PER_INCREMENT macro 6682
TEST_TIMER_MS_TIMEOUT macro 6683
TEST_TimerIncrement function 6690
Text Entry Object 2843
Text Rendering 2570
Text Rendering and Font Features 2570
Text Rendering Functions 2610
TFTP Client Server Communication 6617
TFTP Module 6616
TFTP_CMD_GET_TYPE enumeration member 6624
TFTP_CMD_NONE enumeration member 6624
TFTP_CMD_PUT_TYPE enumeration member 6624
tftpc.h 6627
tftpc_config.h 6628
THREESIXTYQ10D22 macro 3117
THREESIXTYQ10D6 macro 3118
ti.h 1166
ti_private.h 1169
Timer Driver Library 2042
Timer Peripheral Library 5242
Timer System Service Library 5932
TMR_CLOCK_SOURCE enumeration 5277
TMR_MODULE_ID enumeration 5277
TMR_PRESCALE enumeration 5278
Touch Driver Libraries Help 2093
Touch Input Read Request 2131
Touch System Service Library 5966
TouchShowMessage macro 5973
TPL Table Configuration for Audio v1.0 Devices 7006
TPL Table Configuration for CDC Devices 7105
Transfer Operation 1871
Transfer/Abort (Asynchronous) Trigger Management 3739
Transfer/Abort (Synchronous) 3738
Transferring Data 6724, 6753
Transferring Data to the Host 2206
Transmit 3991
Transmitting a CAN Message 3522
Triggering a Software Reset 5928
true enumeration 6994
TRUE macro 214
TRY_AGAIN macro 6059
TWOSEVENTYQ10D22 macro 3118
TWOSEVENTYQ10D6 macro 3118
types.h 1169

U

UART Console Device 5657
UART Console Device Configuration Options 5659
UDP Changes 5990
UDP Module 6633
udp.h 6664
udp_config.h 6666
UDP_PORT type 6660
UDP_SOCKET type 6660
UDP_SOCKET_BCAST_TYPE enumeration 6661
UDP_SOCKET_INFO structure 6661
UDP_SOCKET_OPTION enumeration 6662
uint16_gfx_image_prog type 2947
uint16_prog type 2955
uint16_prog_pack type 2955
uint32_gfx_image_prog type 2947
uint32_prog type 2955
uint32_prog_pack type 2955
uint8_gfx_image_prog type 2947
uint8_prog type 2955
uint8_prog_pack type 2956
Unfilled Polygon Rendering 2568
UNITYFLOAT macro 3118
Upgrading from the MLA Graphics Library to the MPLAB Harmony Graphics Library 2553
Upgrading from the V5 TCP/IP Stack to the MPLAB Harmony TCP/IP Stack 5986
USART Driver Library 2329
USART Peripheral Library 5280
USART_HANDSHAKE_MODE enumeration 5361
USART_LINECONTROL_MODE enumeration 5361
USART_MODULE_ID enumeration 5362
USART_OPERATION_MODE enumeration 5363
USART_RECEIVE_INTR_MODE enumeration 5363
USART_TRANSMIT_INTR_MODE enumeration 5364
USB Audio 1.0 Device Library 6717
USB Audio 2.0 Device Library 6747
USB Audio v1.0 Host Client Driver Library 7005
USB Buffers and the Buffer Descriptor Table (BDT) 5372
USB CDC Device Library 6857
USB CDC Host Client Driver 6952
USB CDC Host Library 7103

USB Device Descriptors 6716
USB Device Layer Library 6773
USB Device Library 6697
USB Device Library - Application Interaction 6699
USB Device Library - Getting Started 6697
USB Device Library Architecture 6697
USB Device Stack in MLA and MPLAB Harmony 6712
USB Device Stack Porting Example 6711
USB Device Stack Porting Guide 6707
USB Driver Device Mode Operation 2202
USB Driver Host Mode Operation 2195
USB Driver Libraries 2180
USB HID Device Library 6893
USB HID Host Mouse Driver Library 7137
USB Host Layer 6946
USB Host Layer Library 6956
USB Host Library 6943
USB Host Library - Application Interaction 6945
USB Host Library - Getting Started 6943
USB Host Library Architecture 6944
USB Host Library Migration Guide 6946
USB Hub Host Client Driver Library 7148
USB Hub Host Client Driver Test Results 7149
USB Libraries Help 6697
USB MSD Device Library 6925
USB MSD Host Client Driver and SCSI Block Storage Driver 6948
USB MSD Host Client Driver Library 7152
USB Peripheral Library 5367
USB Setup Example 5374
USB_BUFFER_DATA01 enumeration 5505
USB_BUFFER_DIRECTION enumeration 5506
USB_BUFFER_PING_PONG enumeration 5506
USB_BUFFER_SCHEDULE_DATA01 enumeration 5506
usb_common.h 6856
USB_DATA_DIRECTION enumeration 6852
usb_device.h 6854
USB_DEVICE_ActiveConfigurationGet function 6813
USB_DEVICE_ActiveSpeedGet function 6813
USB_DEVICE_Attach function 6811
USB_DEVICE_AUDIO_EVENT enumeration 6733
USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_CUR type 6742
USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MAX type 6742
USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MEM type 6742
USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_MIN type 6742
USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_GET_RES type 6743
USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_CUR type 6743
USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MAX type 6743
USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MEM type 6743
USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_MIN type 6744
USB_DEVICE_AUDIO_EVENT_DATA_CONTROL_SET_RES type 6744
USB_DEVICE_AUDIO_EVENT_DATA_ENTITY_GET_STAT type 6745
USB_DEVICE_AUDIO_EVENT_DATA_INTERFACE_SETTING_CHANGED
structure 6744
USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE structure 6739
USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE structure 6739
USB_DEVICE_AUDIO_EVENT_HANDLER type 6740
USB_DEVICE_AUDIO_EVENT_RESPONSE type 6740
USB_DEVICE_AUDIO_EVENT_RESPONSE_NONE macro 6733
USB_DEVICE_AUDIO_EventHandlerSet function 6728
USB_DEVICE_AUDIO_FUNCTION_DRIVER macro 6742
USB_DEVICE_AUDIO_INDEX type 6740
USB_DEVICE_AUDIO_INIT structure 6744
USB_DEVICE_AUDIO_INSTANCES_NUMBER macro 6725
USB_DEVICE_AUDIO_MAX_ALTERNATE_SETTING macro 6725
USB_DEVICE_AUDIO_MAX_STREAMING_INTERFACES macro 6725
USB_DEVICE_AUDIO_QUEUE_DEPTH_COMBINED macro 6726
USB_DEVICE_AUDIO_Read function 6729
USB_DEVICE_AUDIO_RESULT enumeration 6741
USB_DEVICE_AUDIO_TRANSFER_ABORT_NOTIFY macro 6745
USB_DEVICE_AUDIO_TRANSFER_HANDLE type 6741
USB_DEVICE_AUDIO_TRANSFER_HANDLE_INVALID macro 6733
USB_DEVICE_AUDIO_TransferCancel function 6730
usb_device_audio_v1_0.h 6746
usb_device_audio_v1_0_config_template.h 6747
usb_device_audio_v2_0.h 6772
usb_device_audio_v2_0_config_template.h 6773
USB_DEVICE_AUDIO_V2_EVENT enumeration 6762
USB_DEVICE_AUDIO_V2_EVENT_DATA_READ_COMPLETE structure 6767
USB_DEVICE_AUDIO_V2_EVENT_DATA_SET_ALTERNATE_INTERFACE
structure 6767
USB_DEVICE_AUDIO_V2_EVENT_DATA_WRITE_COMPLETE structure 6767
USB_DEVICE_AUDIO_V2_EVENT_HANDLER type 6768
USB_DEVICE_AUDIO_V2_EVENT_RESPONSE type 6768
USB_DEVICE_AUDIO_V2_EVENT_RESPONSE_NONE macro 6771
USB_DEVICE_AUDIO_V2_EventHandlerSet function 6756
USB_DEVICE_AUDIO_V2_FUNCTION_DRIVER macro 6771
USB_DEVICE_AUDIO_V2_INDEX type 6769
USB_DEVICE_AUDIO_V2_INIT structure 6769
USB_DEVICE_AUDIO_V2_INSTANCES_NUMBER macro 6754
USB_DEVICE_AUDIO_V2_MAX_ALTERNATE_SETTING macro 6754
USB_DEVICE_AUDIO_V2_MAX_STREAMING_INTERFACES macro 6755
USB_DEVICE_AUDIO_V2_QUEUE_DEPTH_COMBINED macro 6755
USB_DEVICE_AUDIO_V2_Read function 6757
USB_DEVICE_AUDIO_V2_RESULT enumeration 6770
USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE type 6770
USB_DEVICE_AUDIO_V2_TRANSFER_HANDLE_INVALID macro 6771
USB_DEVICE_AUDIO_V2_TransferCancel function 6759
USB_DEVICE_AUDIO_V2_Write function 6760
USB_DEVICE_AUDIO_Write function 6731
USB_DEVICE_BOS_DESCRIPTOR_SUPPORT_ENABLE macro 6853
usb_device_cdc.h 6891
usb_device_cdc_config_template.h 6893
USB_DEVICE_CDC_EVENT enumeration 6880
USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE structure 6884
USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK structure 6886
USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE
structure 6884

USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE structure 6885
USB_DEVICE_CDC_EVENT_HANDLER type 6885
USB_DEVICE_CDC_EVENT_RESPONSE type 6886
USB_DEVICE_CDC_EVENT_RESPONSE_NONE macro 6889
USB_DEVICE_CDC_EventHandlerSet function 6871
USB_DEVICE_CDC_FUNCTION_DRIVER macro 6889
USB_DEVICE_CDC_INDEX type 6886
USB_DEVICE_CDC_INDEX_0 macro 6890
USB_DEVICE_CDC_INDEX_1 macro 6890
USB_DEVICE_CDC_INDEX_2 macro 6890
USB_DEVICE_CDC_INDEX_3 macro 6890
USB_DEVICE_CDC_INDEX_4 macro 6890
USB_DEVICE_CDC_INDEX_5 macro 6891
USB_DEVICE_CDC_INDEX_6 macro 6891
USB_DEVICE_CDC_INDEX_7 macro 6891
USB_DEVICE_CDC_INIT structure 6887
USB_DEVICE_CDC_INSTANCES_NUMBER macro 6869
USB_DEVICE_CDC_QUEUE_DEPTH_COMBINED macro 6869
USB_DEVICE_CDC_Read function 6875
USB_DEVICE_CDC_RESULT enumeration 6887
USB_DEVICE_CDC_SerialStateNotificationSend function 6879
USB_DEVICE_CDC_TRANSFER_FLAGS enumeration 6888
USB_DEVICE_CDC_TRANSFER_HANDLE type 6889
USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID macro 6889
USB_DEVICE_CDC_Write function 6877
USB_DEVICE_CLIENT_STATUS enumeration 6839
USB_DEVICE_ClientStatusGet function 6802
USB_DEVICE_Close function 6801
usb_device_config_template.h 6856
USB_DEVICE_CONFIGURATION_DESCRIPTORS_TABLE type 6839
USB_DEVICE_CONTROL_STATUS enumeration 6830
USB_DEVICE_CONTROL_TRANSFER_RESULT enumeration 6831
USB_DEVICE_ControlReceive function 6824
USB_DEVICE_ControlSend function 6826
USB_DEVICE_ControlStatus function 6828
USB_DEVICE_Deinitialize function 6798
USB_DEVICE_Detach function 6812
USB_DEVICE_DRIVER_INITIALIZE_EXPLICIT macro 6853
USB_DEVICE_ENDPOINT_QUEUE_DEPTH_COMBINED macro 6791
USB_DEVICE_EndpointDisable function 6816
USB_DEVICE_EndpointEnable function 6817
USB_DEVICE_EndpointIsEnabled function 6819
USB_DEVICE_EndpointIsStalled function 6814
USB_DEVICE_EndpointRead function 6820
USB_DEVICE_EndpointStall function 6815
USB_DEVICE_EndpointStallClear function 6816
USB_DEVICE_EndpointTransferCancel function 6821
USB_DEVICE_EndpointWrite function 6822
USB_DEVICE_EP0_BUFFER_SIZE macro 6793
USB_DEVICE_EVENT enumeration 6831
USB_DEVICE_EVENT_DATA_CONFIGURED structure 6838
USB_DEVICE_EVENT_DATA_ENDPOINT_READ_COMPLETE structure 6842
USB_DEVICE_EVENT_DATA_ENDPOINT_WRITE_COMPLETE structure 6842
USB_DEVICE_EVENT_DATA_SET_DESCRIPTOR structure 6843
USB_DEVICE_EVENT_DATA_SOF structure 6843
USB_DEVICE_EVENT_DATA_SYNCH_FRAME structure 6844
USB_DEVICE_EVENT_HANDLER type 6840
USB_DEVICE_EVENT_RESPONSE type 6840
USB_DEVICE_EVENT_RESPONSE_NONE macro 6839
USB_DEVICE_EventHandlerSet function 6803
USB_DEVICE_FUNCTION_REGISTRATION_TABLE structure 6840
USB_DEVICE_HANDLE type 6836
USB_DEVICE_HANDLE_INVALID macro 6838
usb_device_hid.h 6923
usb_device_hid_config_template.h 6924
USB_DEVICE_HID_EVENT enumeration 6911
USB_DEVICE_HID_EVENT_DATA_GET_IDLE structure 6918
USB_DEVICE_HID_EVENT_DATA_GET_REPORT structure 6915
USB_DEVICE_HID_EVENT_DATA_REPORT_RECEIVED structure 6916
USB_DEVICE_HID_EVENT_DATA_REPORT_SENT structure 6916
USB_DEVICE_HID_EVENT_DATA_SET_IDLE structure 6917
USB_DEVICE_HID_EVENT_DATA_SET_PROTOCOL structure 6919
USB_DEVICE_HID_EVENT_DATA_SET_REPORT structure 6917
USB_DEVICE_HID_EVENT_HANDLER type 6919
USB_DEVICE_HID_EVENT_RESPONSE type 6919
USB_DEVICE_HID_EVENT_RESPONSE_NONE macro 6921
USB_DEVICE_HID_EventHandlerSet function 6906
USB_DEVICE_HID_FUNCTION_DRIVER macro 6921
USB_DEVICE_HID_INDEX type 6918
USB_DEVICE_HID_INDEX_0 macro 6922
USB_DEVICE_HID_INDEX_1 macro 6922
USB_DEVICE_HID_INDEX_2 macro 6922
USB_DEVICE_HID_INDEX_3 macro 6922
USB_DEVICE_HID_INDEX_4 macro 6922
USB_DEVICE_HID_INDEX_5 macro 6923
USB_DEVICE_HID_INDEX_6 macro 6923
USB_DEVICE_HID_INDEX_7 macro 6923
USB_DEVICE_HID_INIT structure 6920
USB_DEVICE_HID_INSTANCES_NUMBER macro 6904
USB_DEVICE_HID_QUEUE_DEPTH_COMINED macro 6905
USB_DEVICE_HID_ReportReceive function 6907
USB_DEVICE_HID_ReportSend function 6909
USB_DEVICE_HID_RESULT enumeration 6920
USB_DEVICE_HID_TRANSFER_HANDLE type 6918
USB_DEVICE_HID_TRANSFER_HANDLE_INVALID macro 6921
USB_DEVICE_HID_TransferCancel function 6910
USB_DEVICE_INDEX_0 macro 6829
USB_DEVICE_INDEX_1 macro 6830
USB_DEVICE_INDEX_2 macro 6830
USB_DEVICE_INDEX_3 macro 6830
USB_DEVICE_INDEX_4 macro 6830
USB_DEVICE_INDEX_5 macro 6830
USB_DEVICE_INIT structure 6836
USB_DEVICE_Initialize function 6796
USB_DEVICE_INSTANCES_NUMBER macro 6791
USB_DEVICE_IRP structure 6846
USB_DEVICE_IRP_FLAG enumeration 6846
USB_DEVICE_IRP_STATUS enumeration 6847
USB_DEVICE_IsSuspended function 6807
USB_DEVICE_MASTER_DESCRIPTOR structure 6841
USB_DEVICE_MICROSOFT_OS_DESCRIPTOR_SUPPORT_ENABLE macro 6793
usb_device_msd.h 6934

usb_device_msd_config_template.h 6934
USB_DEVICE_MSD_FUNCTION_DRIVER macro 6934
USB_DEVICE_MSD_INIT structure 6932
USB_DEVICE_MSD_INSTANCES_NUMBER macro 6929
USB_DEVICE_MSD_LUNS_NUMBER macro 6930
USB_DEVICE_MSD_MEDIA_FUNCTIONS structure 6931
USB_DEVICE_MSD_MEDIA_INIT_DATA structure 6933
USB_DEVICE_Open function 6800
USB_DEVICE_POWER_STATE enumeration 6837
USB_DEVICE_PowerStateSet function 6805
USB_DEVICE_REMOTE_WAKEUP_STATUS enumeration 6838
USB_DEVICE_RemoteWakeupStart function 6808
USB_DEVICE_RemoteWakeupStartTimed function 6809
USB_DEVICE_RemoteWakeupStatusGet function 6806
USB_DEVICE_RemoteWakeupStop function 6809
USB_DEVICE_RESULT enumeration 6844
USB_DEVICE_SET_DESCRIPTOR_EVENT_ENABLE macro 6792
USB_DEVICE_SOF_EVENT_ENABLE macro 6792
USB_DEVICE_StateGet function 6810
USB_DEVICE_Status function 6798
USB_DEVICE_STRING_DESCRIPTOR_TABLE_ADVANCED_ENABLE macro 6853
USB_DEVICE_STRING_DESCRIPTORS_TABLE type 6842
USB_DEVICE_SYNCH_FRAME_EVENT_ENABLE macro 6792
USB_DEVICE_Tasks function 6799
USB_DEVICE_Tasks_ISR function 6800
USB_DEVICE_Tasks_ISR_USBDMA function 6800
USB_DEVICE_TRANSFER_FLAGS enumeration 6845
USB_DEVICE_TRANSFER_HANDLE type 6845
USB_DEVICE_TRANSFER_HANDLE_INVALID macro 6845
USB_ENDPOINT type 6848
USB_ENDPOINT_AND_DIRECTION macro 6852
USB_EP_TXRX enumeration 5507
USB_ERROR enumeration 6848
USB_HID_GLOBAL_PUSH_POP_STACK_SIZE macro 7140
usb_host.h 7003
usb_host_audio_v1_0.h 7099
USB_HOST_AUDIO_V1_0_ATTACH_EVENT_HANDLER type 7078
USB_HOST_AUDIO_V1_0_AttachEventHandlerSet macro 7095
usb_host_audio_v1_0_config_template.h 7103
USB_HOST_AUDIO_V1_0_CONTROL_CALLBACK type 7081
USB_HOST_AUDIO_V1_0_ControlRequest function 7039
USB_HOST_AUDIO_V1_0_DeviceObjHandleGet macro 7095
USB_HOST_AUDIO_V1_0_DIRECTION_IN macro 7096
USB_HOST_AUDIO_V1_0_DIRECTION_OUT macro 7096
USB_HOST_AUDIO_V1_0_EVENT macro 7082
USB_HOST_AUDIO_V1_0_EVENT_ATTACH macro 7096
USB_HOST_AUDIO_V1_0_EVENT_DETACH macro 7097
USB_HOST_AUDIO_V1_0_INTERFACE macro 7094
USB_HOST_AUDIO_V1_0_NumberOfStreamGroupsGet function 7053
USB_HOST_AUDIO_V1_0_OBJ macro 7083
USB_HOST_AUDIO_V1_0_REQUEST_HANDLE macro 7084
USB_HOST_AUDIO_V1_0_REQUEST_HANDLE_INVALID macro 7094
USB_HOST_AUDIO_V1_0_RESULT enumeration 7085
USB_HOST_AUDIO_V1_0_STREAM_DIRECTION macro 7086
USB_HOST_AUDIO_V1_0_STREAM_EVENT enumeration 7086
USB_HOST_AUDIO_V1_0_STREAM_EVENT_DISABLE_COMPLETE_DATA structure 7088
USB_HOST_AUDIO_V1_0_STREAM_EVENT_ENABLE_COMPLETE_DATA structure 7088
USB_HOST_AUDIO_V1_0_STREAM_EVENT_HANDLER type 7089
USB_HOST_AUDIO_V1_0_STREAM_EVENT_READ_COMPLETE_DATA macro 7089
USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE macro 7089
USB_HOST_AUDIO_V1_0_STREAM_EVENT_RESPONSE_NONE macro 7097
USB_HOST_AUDIO_V1_0_STREAM_EVENT_WRITE_COMPLETE_DATA macro 7090
USB_HOST_AUDIO_V1_0_STREAM_HANDLE macro 7090
USB_HOST_AUDIO_V1_0_STREAM_HANDLE_INVALID macro 7094
USB_HOST_AUDIO_V1_0_STREAM_INFO structure 7091
USB_HOST_AUDIO_V1_0_STREAM_OBJ type 7091
USB_HOST_AUDIO_V1_0_STREAM_RESULT enumeration 7092
USB_HOST_AUDIO_V1_0_STREAM_TRANSFER_HANDLE macro 7093
USB_HOST_AUDIO_V1_0_STREAM_TRANSFER_HANDLE_INVALID macro 7094
USB_HOST_AUDIO_V1_0_StreamClose macro 7097
USB_HOST_AUDIO_V1_0_StreamDisable function 7055
USB_HOST_AUDIO_V1_0_StreamEnable function 7056
USB_HOST_AUDIO_V1_0_StreamEventHandlerSet function 7057
USB_HOST_AUDIO_V1_0_StreamGetFirst function 7059
USB_HOST_AUDIO_V1_0_StreamGetNext function 7060
USB_HOST_AUDIO_V1_0_StreamOpen macro 7098
USB_HOST_AUDIO_V1_0_StreamRead function 7071
USB_HOST_AUDIO_V1_0_StreamSamplingRateSet function 7062
USB_HOST_AUDIO_V1_0_StreamWrite function 7072
USB_HOST_AUDIO_V1_ATTACH_EVENT_HANDLER type 7075
USB_HOST_AUDIO_V1_ATTACH_LISTENERS_NUMBER macro 7032
USB_HOST_AUDIO_V1_AttachEventHandlerSet function 7038
USB_HOST_AUDIO_V1_CONTROL_ENTITY_OBJ type 7075
USB_HOST_AUDIO_V1_ControlEntityGetFirst function 7040
USB_HOST_AUDIO_V1_ControlEntityGetNext function 7041
USB_HOST_AUDIO_V1_DeviceObjHandleGet function 7041
USB_HOST_AUDIO_V1_ENTITY_REQUEST_CALLBACK type 7076
USB_HOST_AUDIO_V1_EntityObjectGet function 7042
USB_HOST_AUDIO_V1_EntityRequestCallbackSet function 7042
USB_HOST_AUDIO_V1_EntityTypeGet function 7043
USB_HOST_AUDIO_V1_EVENT enumeration 7076
USB_HOST_AUDIO_V1_FeatureUnitChannelMuteExists function 7044
USB_HOST_AUDIO_V1_FeatureUnitChannelMuteGet function 7044
USB_HOST_AUDIO_V1_FeatureUnitChannelMuteSet function 7045
USB_HOST_AUDIO_V1_FeatureUnitChannelNumbersGet function 7046
USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeExists function 7046
USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeGet function 7047
USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeRangeGet function 7073
USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeSet function 7048
USB_HOST_AUDIO_V1_FeatureUnitChannelVolumeSubRangeNumbersGet function 7074
USB_HOST_AUDIO_V1_FeatureUnitIDGet function 7049
USB_HOST_AUDIO_V1_FeatureUnitSourceIDGet function 7049
USB_HOST_AUDIO_V1_INSTANCES_NUMBER macro 7033
USB_HOST_AUDIO_V1_INTERFACE macro 7090

USB_HOST_AUDIO_V1_OBJ type 7076
USB_HOST_AUDIO_V1_REQUEST_HANDLE type 7077
USB_HOST_AUDIO_V1_REQUEST_HANDLE_INVALID macro 7091
USB_HOST_AUDIO_V1_RESULT enumeration 7077
USB_HOST_AUDIO_V1_SAMPLING_FREQUENCIES_NUMBER macro 7098
USB_HOST_AUDIO_V1_STREAM_DIRECTION enumeration 7078
USB_HOST_AUDIO_V1_STREAM_EVENT enumeration 7079
USB_HOST_AUDIO_V1_STREAM_EVENT_HANDLER type 7080
USB_HOST_AUDIO_V1_STREAM_EVENT_INTERFACE_SET_COMPLETE_DATA structure 7081
USB_HOST_AUDIO_V1_STREAM_EVENT_READ_COMPLETE_DATA structure 7082
USB_HOST_AUDIO_V1_STREAM_EVENT_RESPONSE enumeration 7082
USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_GET_COMPLETE_DATA structure 7099
USB_HOST_AUDIO_V1_STREAM_EVENT_SAMPLING_RATE_SET_COMPLETE_DATA structure 7083
USB_HOST_AUDIO_V1_STREAM_EVENT_WRITE_COMPLETE_DATA structure 7084
USB_HOST_AUDIO_V1_STREAM_HANDLE type 7084
USB_HOST_AUDIO_V1_STREAM_HANDLE_INVALID macro 7092
USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE type 7085
USB_HOST_AUDIO_V1_STREAM_TRANSFER_HANDLE_INVALID macro 7093
USB_HOST_AUDIO_V1_StreamClose function 7053
USB_HOST_AUDIO_V1_StreamEventHandlerSet function 7054
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_ALTERNATE_SETTINGS_NUMBER macro 7033
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_OBJ type 7086
USB_HOST_AUDIO_V1_STREAMING_INTERFACE_SETTING_OBJ type 7088
USB_HOST_AUDIO_V1_STREAMING_INTERFACES_NUMBER macro 7033
USB_HOST_AUDIO_V1_StreamingInterfaceBitResolutionGet function 7064
USB_HOST_AUDIO_V1_StreamingInterfaceChannelNumbersGet function 7065
USB_HOST_AUDIO_V1_StreamingInterfaceDirectionGet function 7066
USB_HOST_AUDIO_V1_StreamingInterfaceFormatTagGet function 7066
USB_HOST_AUDIO_V1_StreamingInterfaceGetFirst function 7055
USB_HOST_AUDIO_V1_StreamingInterfaceGetNext function 7057
USB_HOST_AUDIO_V1_StreamingInterfaceSamplingFrequenciesGet function 7067
USB_HOST_AUDIO_V1_StreamingInterfaceSamplingFrequencyTypeGet function 7067
USB_HOST_AUDIO_V1_StreamingInterfaceSet function 7058
USB_HOST_AUDIO_V1_StreamingInterfaceSettingGetFirst function 7059
USB_HOST_AUDIO_V1_StreamingInterfaceSettingGetNext function 7061
USB_HOST_AUDIO_V1_StreamingInterfaceSubFrameSizeGet function 7068
USB_HOST_AUDIO_V1_StreamingInterfaceTerminalLinkGet function 7069
USB_HOST_AUDIO_V1_StreamOpen function 7062
USB_HOST_AUDIO_V1_StreamRead function 7063
USB_HOST_AUDIO_V1_StreamSamplingFrequencyGet function 7069
USB_HOST_AUDIO_V1_StreamSamplingFrequencySet function 7070
USB_HOST_AUDIO_V1_StreamWrite function 7064
USB_HOST_AUDIO_V1_TerminalAssociationGet function 7050
USB_HOST_AUDIO_V1_TerminalIDGet function 7050
USB_HOST_AUDIO_V1_TerminalInputChannelConfigGet function 7072
USB_HOST_AUDIO_V1_TerminalInputChannelNumbersGet function 7051
USB_HOST_AUDIO_V1_TerminalSourceIDGet function 7052
USB_HOST_AUDIO_V1_TerminalTypeGet function 7052
USB_HOST_BUS type 6995
USB_HOST_BUS_ALL macro 7002
USB_HOST_BusEnable function 6972
USB_HOST_BusIsEnabled function 6972
USB_HOST_BusIsSuspended function 6973
USB_HOST_BusResume function 6973
USB_HOST_BusSuspend function 6974
usb_host_cdc.h 7136
usb_host_cdc_acm.h 7137
USB_HOST_CDC_ACM_BreakSend function 7122
USB_HOST_CDC_ACM_ControlLineStateSet function 7123
USB_HOST_CDC_ACM_LineCodingGet function 7124
USB_HOST_CDC_ACM_LineCodingSet function 7125
USB_HOST_CDC_ATTACH_EVENT_HANDLER type 7129
USB_HOST_CDC_ATTACH_LISTENERS_NUMBER macro 7114
USB_HOST_CDC_AttachEventHandlerSet function 7118
USB_HOST_CDC_Close function 7117
usb_host_cdc_config_template.h 7137
USB_HOST_CDC_DeviceObjHandleGet function 7119
USB_HOST_CDC_EVENT enumeration 7126
USB_HOST_CDC_EVENT_ACM_GET_LINE_CODING_COMPLETE_DATA structure 7129
USB_HOST_CDC_EVENT_ACM_SEND_BREAK_COMPLETE_DATA structure 7130
USB_HOST_CDC_EVENT_ACM_SET_CONTROL_LINE_STATE_COMPLETE_DATA structure 7131
USB_HOST_CDC_EVENT_ACM_SET_LINE_CODING_COMPLETE_DATA structure 7131
USB_HOST_CDC_EVENT_HANDLER type 7132
USB_HOST_CDC_EVENT_READ_COMPLETE_DATA structure 7132
USB_HOST_CDC_EVENT_RESPONSE enumeration 7133
USB_HOST_CDC_EVENT_SERIAL_STATE_NOTIFICATION_RECEIVED_DATA structure 7133
USB_HOST_CDC_EVENT_WRITE_COMPLETE_DATA structure 7133
USB_HOST_CDC_EventHandlerSet function 7118
USB_HOST_CDC_HANDLE type 7130
USB_HOST_CDC_HANDLE_INVALID macro 7135
USB_HOST_CDC_INSTANCES_NUMBER macro 7114
USB_HOST_CDC_INTERFACE macro 7135
USB_HOST_CDC_OBJ type 7134
USB_HOST_CDC_Open function 7116
USB_HOST_CDC_Read function 7120
USB_HOST_CDC_REQUEST_HANDLE type 7134
USB_HOST_CDC_REQUEST_HANDLE_INVALID macro 7135
USB_HOST_CDC_RESULT enumeration 7128

USB_HOST_CDC_SerialStateNotificationGet function 7121
USB_HOST_CDC_TRANSFER_HANDLE type 7128
USB_HOST_CDC_TRANSFER_HANDLE_INVALID macro 7128
USB_HOST_CDC_Write function 7121
usb_host_config_template.h 7005
USB_HOST_CONTROLLERS_NUMBER macro 6965
USB_HOST_Deinitialize function 6969
USB_HOST_DEVICE_INFO structure 6995
USB_HOST_DEVICE_INTERFACES_NUMBER macro 6965
USB_HOST_DEVICE_OBJ_HANDLE type 6996
USB_HOST_DEVICE_OBJ_HANDLE_INVALID macro 7002
USB_HOST_DEVICE_STRING enumeration 6996
USB_HOST_DEVICE_STRING_LANG_ID_DEFAULT macro 7002
USB_HOST_DeviceGetFirst function 6975
USB_HOST_DeviceGetNext function 6975
USB_HOST_DeviceIsSuspended function 6976
USB_HOST_DeviceResume function 6977
USB_HOST_DEVICES_NUMBER macro 6966
USB_HOST_DeviceSpeedGet function 6977
USB_HOST_DeviceStringDescriptorGet function 6978
USB_HOST_DeviceSuspend function 6979
USB_HOST_EVENT enumeration 6996
USB_HOST_EVENT_HANDLER type 6997
USB_HOST_EVENT_RESPONSE enumeration 6981
USB_HOST_EventHandlerSet function 6980
USB_HOST_HCD structure 6997
usb_host_hid_config_template.h 7147
USB_HOST_HID_INSTANCES_NUMBER macro 7141
USB_HOST_HID_INTERRUPT_IN_ENDPOINTS_NUMBER macro 7141
usb_host_hid_mouse.h 7147
USB_HOST_HID_MOUSE_BUTTONS_NUMBER macro 7141
USB_HOST_HID_MOUSE_DATA structure 7144
USB_HOST_HID_MOUSE_EVENT enumeration 7145
USB_HOST_HID_MOUSE_EVENT_HANDLER type 7145
USB_HOST_HID_MOUSE_EventHandlerSet function 7143
USB_HOST_HID_MOUSE_HANDLE type 7145
USB_HOST_HID_MOUSE_HANDLE_INVALID macro 7146
USB_HOST_HID_MOUSE_RESULT enumeration 7145
USB_HOST_HID_MOUSE_RESULT_MIN macro 7146
USB_HOST_HID_USAGE_DRIVER_SUPPORT_NUMBER macro 7142
usb_host_hub.h 7151
usb_host_hub_config_template.h 7152
USB_HOST_HUB_INSTANCES_NUMBER macro 7149
USB_HOST_HUB_INTERFACE macro 7151
USB_HOST_HUB_PORTS_NUMBER macro 7150
USB_HOST_HUB_SUPPORT_ENABLE macro 6966
USB_HOST_HUB_TIER_LEVEL macro 6966
USB_HOST_INIT structure 6981
USB_HOST_Initialize function 6980
USB_HOST_IRP structure 6849
USB_HOST_IRP_FLAG enumeration 6850
USB_HOST_IRP_STATUS enumeration 6851
USB_HOST_IRP_STATUS_ABORTED enumeration member 6851
USB_HOST_IRP_STATUS_COMPLETED enumeration member 6851
USB_HOST_IRP_STATUS_COMPLETED_SHORT enumeration member 6851
USB_HOST_IRP_STATUS_ERROR_BUS enumeration member 6851
USB_HOST_IRP_STATUS_ERROR_DATA enumeration member 6851
USB_HOST_IRP_STATUS_ERROR_NAK_TIMEOUT enumeration member 6851
USB_HOST_IRP_STATUS_ERROR_STALL enumeration member 6851
USB_HOST_IRP_STATUS_ERROR_UNKNOWN enumeration member 6851
USB_HOST_IRP_STATUS_IN_PROGRESS enumeration member 6851
USB_HOST_IRP_STATUS_PENDING enumeration member 6851
usb_host_msd.h 7164
usb_host_msd_config_template.h 7164
USB_HOST_MSD_ERROR_CODE enumeration 7161
USB_HOST_MSD_INSTANCES_NUMBER macro 7155
USB_HOST_MSD_INTERFACE macro 7160
USB_HOST_MSD_LUN_HANDLE type 7161
USB_HOST_MSD_LUN_HANDLE_INVALID macro 7161
USB_HOST_MSD_LUNS_NUMBER macro 7155
USB_HOST_MSD_RESULT enumeration 7158
USB_HOST_MSD_Transfer function 7157
USB_HOST_MSD_TRANSFER_CALLBACK type 7159
USB_HOST_MSD_TRANSFER_DIRECTION enumeration 7160
USB_HOST_MSD_TRANSFER_HANDLE type 7160
USB_HOST_MSD_TRANSFER_HANDLE_INVALID macro 7161
USB_HOST_MSD_TransferErrorTasks function 7158
USB_HOST_PIPES_NUMBER macro 6967
USB_HOST_REQUEST_HANDLE type 6998
USB_HOST_REQUEST_HANDLE_INVALID macro 7003
USB_HOST_RESULT enumeration 6998
USB_HOST_RESULT_MIN macro 7003
USB_HOST_Status function 6970
USB_HOST_STRING_REQUEST_COMPLETE_CALLBACK type 6999
USB_HOST_TARGET_PERIPHERAL_LIST_ENTRY enumeration 6999
USB_HOST_Tasks function 6971
USB_HOST_TPL_ENTRY enumeration 7000
USB_HOST_TRANSFERS_NUMBER macro 6967
USB_MAX_EP_NUMBER macro 5511
USB_OPMODES enumeration 5507
USB_OTG_INTERRUPTS enumeration 5508
USB_OTG_PULL_UP_PULL_DOWN enumeration 5508
USB_PID enumeration 5509
USB_PING_PONG_MODE enumeration 5509
USB_PING_PONG_STATE enumeration 5510
USB_SPEED enumeration 6851
USB_TOKEN_SPEED enumeration 5510
USB-CDC Console Device 5657
USB-CDC Console Device Configuration Options 5658
USBHS Peripheral Library 5516
USBHS_CONFIGURATION enumeration 5552
USBHS_DATA01 enumeration 5553
USBHS_DMA_ASSERT_TIMING enumeration 5553
USBHS_DMA_BURST_MODE enumeration 5554
USBHS_DMA_INTERRUPT enumeration 5554
USBHS_DMA_REQUEST_MODE enumeration 5555
USBHS_DMA_TRANSFER_MODE enumeration 5555
USBHS_DYN_FIFO_PACKET_BUFFERING enumeration 5555
USBHS_DYN_FIFO_SIZE enumeration 5556
USBHS_ENDPOINT_DIRECTION enumeration 5556
USBHS_LPM_INTERRUPT enumeration 5556
USBHS_LPM_LINK_STATE enumeration 5557
USBHS_LPM_MODE enumeration 5557

USBHS_MAX_DMA_CHANNEL_NUMBER macro 5560
USBHS_MAX_EP_NUMBER macro 5561
USBHS_OPMODES enumeration 5558
USBHS_PKTS_PER_MICROFRAME enumeration 5558
USBHS_SPEED enumeration 5559
USBHS_TEST_SPEED enumeration 5559
USBHS_TRANSACTION_TYPE enumeration 5560
USBHS_TXRX_FIFO_STATE enumeration 5560
USE_MARCHC_MINUS macro 1209
Using a Driver in an Application 1340
Using a Driver's Client Interface 1339
Using a Driver's System Interface 1337
Using Asynchronous and Callback Functions 1344
Using Driver Interface Functions 1343
Using Iperf 6290
Using Status Functions 5612
Using the Bootloader Application (UART, USB HID, and Ethernet Bootloaders) 1177
Using the File System 5772
Using the Graphics Object Layer 2572
Using the Library 5, 1173, 1191, 1211, 1260, 1271, 1292, 1308, 1323, 1366, 1392, 1442, 1482, 1521, 1559, 1569, 1589, 1608, 1629, 1674, 1728, 1768, 1830, 1868, 1901, 1930, 1966, 2042, 2093, 2111, 2129, 2151, 2210, 2269, 2329, 2402, 2495, 2562, 2999, 3094, 3123, 3157, 3201, 3237, 3303, 3435, 3482, 3515, 3614, 3653, 3685, 3700, 3736, 3867, 3936, 3979, 4135, 4237, 4342, 4367, 4416, 4480, 4508, 4617, 4710, 4760, 4848, 4895, 4911, 4971, 5012, 5106, 5242, 5281, 5369, 5516, 5564, 5577, 5609, 5634, 5654, 5673, 5690, 5716, 5769, 5834, 5855, 5860, 5886, 5915, 5926, 5932, 5966, 5976, 6006, 6010, 6029, 6064, 6078, 6094, 6114, 6137, 6149, 6157, 6163, 6202, 6267, 6278, 6290, 6298, 6312, 6351, 6401, 6468, 6472, 6479, 6482, 6496, 6516, 6565, 6576, 6616, 6628, 6634, 6667, 6678, 6718, 6747, 6773, 6857, 6893, 6925, 6935, 6957, 7006, 7104, 7138, 7148, 7152
    10-bit ADC Touch Driver Library 2093
    AAC Decoder Library 1260
    ADC Peripheral Library 3237
    ADCHS Peripheral Library 3303
    ADCP Peripheral Library 3435
    AK4384 Codec Driver Library 1392
    AK4642 Codec Driver Library 1442
    AK4953 Codec Driver Library 1482
    AK7755 Codec Driver Library 1521
    Announce Module 6006
    AR1021 Touch Driver Library 2111
    ARP Module 6010
    Berkeley Module 6029
    BMX Peripheral Library 3482
    Bootloader Library 1173
    CAN Peripheral Library 3515, 3614
    Class B Library 1191
    Clock System Service Library 5609
    Command Processor System Service Library 5634
    Console System Service Library 5654
    CPLD XC2C64A Driver Library 1559
    Crypto Library 1211
    CTMU Peripheral Library 3653
    Deadman Timer Peripheral Library 3685
    Debug System Service Library 5673
    Device Control Peripheral Library 3700
    Device Control System Service Library 5690
    DHCP Module 6064
    DHCP Server Module 6078
    DHCPv6 Module 6094
    DMA Peripheral Library 3736
    DMA System Service Library 5716
    DNS Module 6114
    DNSS Module 6137
    DSP Fixed-Point Math Library 2999
    Dual Data Rate (DDR) SDRAM Peripheral Library 3867
    Dynamic DNS Module 6149
    EBI Peripheral Library 3936
    Ethernet MAC Driver Library 1608
    Ethernet Peripheral Library 3979
    Ethernet PHY Driver Library 1629
    FTP Module 6157
    Generic USB Device Library 6935
    GLCD Controller Peripheral Library 4135
    Graphics Driver Library 1674
    HTTP Module 6163
    HTTP Net Module 6202
    I2C Peripheral Library 4237
    ICMP Module 6267
    ICMPv6 Module 6278
    Input Capture Peripheral Library 4342
    Interrupt Peripheral Library 4367
    Interrupt System Service Library 5834
    IPv4 Module 6298
    IPv6 Module 6312
    LibQ Fixed-Point 'C' Math Library 3094
    LibQ Fixed-Point Math Library 3123
    MAC Driver Module 6351
    Manager Module 6401
    Memory System Service Library 5855
    Messaging System Service Library 5860
    MP3 Decoder Library 1271
    MRF24W Wi-Fi Driver Library 2402
    MRF24WN Wi-Fi Driver Library 2495
    MTCH6301 Touch Driver Library 2129
    MTCH6303 Touch Driver Library 2151
    NBNS Module 6468
    NDP Module 6472
    Networking Presentation Layer 3157
    NVM Driver Library 1830
    NVM Peripheral Library 4416
    Opus Decoder Library 1292
    OSAL Library 3201
    Oscillator Peripheral Library 4508
    Output Compare Peripheral Library 4480
    PIC32 Bluetooth Stack Library 5
    PMP Driver Library 1868
    PMP Peripheral Library 4617
    Port System Service Library 5886
    Ports Peripheral Library 4760
    Power Peripheral Library 4848
    Prefetch Cache Peripheral Library 4710
    Reboot Module 6479
    Reset Peripheral Library 4895
    RNG System Service Library 5915
    RTCC Peripheral Library 4911

Sample Library 5577
SD Card Driver Library 1901
SMTP Module 6482
SMTPC Module 6496
SNMP Module 6516
SNTP Module 6565
Speex Decoder Library 1308
SPI Driver Library 1930
SPI Flash Driver Library 1966
SPI Peripheral Library 5012
SQI Peripheral Library 5106
System Bus Peripheral Library 4971
TCP Module 6576
Telnet Module 6628
Test Harness Library 6678
TFTP Module 6616
Timer Driver Library 2042
Timer Peripheral Library 5242
Timer System Service Library 5932
Touch System Service Library 5966
UDP Module 6634
USART Driver Library 2329
USART Peripheral Library 5281
USB Audio 1.0 Device Library 6718
USB Audio 2.0 Device Library 6747
USB Audio v1.0 Host Client Driver Library 7006
USB CDC Device Library 6857
USB Device Layer Library 6773
USB HID Device Library 6893
USB HID Host Mouse Driver Library 7138
USB Hub Host Client Driver Library 7148
USB MSD Device Library 6925
USB MSD Host Library 7152
USB Peripheral Library 5369
USBHS Peripheral Library 5516
Watchdog Timer Peripheral Library 5564
Watchdog Timer System Service Library 5976
WMA Decoder Library 1323
Zeroconf Module 6667
Using the Primitive Layer 2567
Using the SYS_ASSERT Macro 5592
Using the USART Driver with DMA 2340

V

VCARD_CALL_TYPE_DIALED macro 979
VCARD_CALL_TYPE_MISSED macro 979
VCARD_CALL_TYPE_RECEIVED macro 979
VCARD_EVT_PROPERTY_PARAM macro 979
VCARD_EVT_PROPERTY_STARTED macro 979
VCARD_EVT_PROPERTY_VALUE macro 980
VCARD_EVT_VCARD_ENDED macro 980
VCARD_EVT_VCARD_STARTED macro 980
VCARD_PARAM_CALL_TYPE macro 980
VCARD_PARAM_CELL macro 980
VCARD_PARAM_DIALED macro 981
VCARD_PARAM_ENCODING macro 981
VCARD_PARAM_HOME macro 981
VCARD_PARAM_LANGUAGE macro 981
VCARD_PARAM_MISSED macro 981
VCARD_PARAM_RECEIVED macro 982
VCARD_PARAM_TYPE macro 982
VCARD_PARAM_VALUE macro 982
VCARD_PARAM_WORK macro 982
vcard_parser.h 1169
VCARD_PARSER_STATE_READ_PARAM_NAME macro 982
VCARD_PARSER_STATE_READ_PARAM_VALUE macro 983
VCARD_PARSER_STATE_READ_TYPE_NAME macro 983
VCARD_PARSER_STATE_READ_TYPE_VALUE macro 983
VCARD_PARSER_STATE_SKIP_PARAM macro 983
VCARD_PARSER_STATE_SKIP_TYPE macro 983
VCARD_PROPERTY_ADR macro 984
VCARD_PROPERTY_AGENT macro 984
VCARD_PROPERTY_BDAY macro 984
VCARD_PROPERTY_BEGIN macro 984
VCARD_PROPERTY_CATEGORIES macro 984
VCARD_PROPERTY_CLASS macro 985
VCARD_PROPERTY_EMAIL macro 985
VCARD_PROPERTY_END macro 985
VCARD_PROPERTY_FN macro 985
VCARD_PROPERTY_GEO macro 985
VCARD_PROPERTY_KEY macro 986
VCARD_PROPERTY_LABEL macro 986
VCARD_PROPERTY_LOGO macro 986
VCARD_PROPERTY_MAILER macro 986
VCARD_PROPERTY_N macro 986
VCARD_PROPERTY_NICKNAME macro 987
VCARD_PROPERTY_NOTE macro 987
VCARD_PROPERTY_ORG macro 987
VCARD_PROPERTY_PHOTO macro 987
VCARD_PROPERTY_PROID macro 987
VCARD_PROPERTY_REV macro 988
VCARD_PROPERTY_ROLE macro 988
VCARD_PROPERTY_SORT_STRING macro 988
VCARD_PROPERTY_SOUND macro 988
VCARD_PROPERTY_TEL macro 988
VCARD_PROPERTY_TITLE macro 989
VCARD_PROPERTY_TZ macro 989
VCARD_PROPERTY_UID macro 989
VCARD_PROPERTY_URL macro 989
VCARD_PROPERTY_VERSION macro 989
VCARD_PROPERTY_X_IRMC_CALL_DATETIME macro 990
vid enumeration 7001
Volume IV: MPLAB Harmony Framework Reference 2

W

W macro 938
Wait States Initialization 4620
Watchdog Timer Peripheral Library 5564
Watchdog Timer System Service Library 5976
WAVE structure 1266
WDRV_CLI_Init function 2504
WDRV_EXT_CmdChannelSet function 2521
WDRV_EXT_CmdConnect function 2514
WDRV_EXT_CmdConnectContextBssidGet function 2512
WDRV_EXT_CmdConnectContextChannelGet function 2509
WDRV_EXT_CmdDisconnect function 2514

WDRV_EXT_CmdFWUpdate function 2522
WDRV_EXT_CmdMacAddressGet function 2510
WDRV_EXT_CmdNetModeAPSet function 2515
WDRV_EXT_CmdNetModeBSSSet function 2515
WDRV_EXT_CmdNetModeIBSSSet function 2515
WDRV_EXT_CmdPowerSaveGet function 2513
WDRV_EXT_CmdPowerSavePut function 2523
WDRV_EXT_CmdRFTest function 2524
WDRV_EXT_CmdScanGet function 2510
WDRV_EXT_CmdScanStart function 2516
WDRV_EXT_CmdSecGet function 2511
WDRV_EXT_CmdSecNoneSet function 2516
WDRV_EXT_CmdSecWEPSet function 2517
WDRV_EXT_CmdSecWPA2Set function 2517
WDRV_EXT_CmdSecWPASet function 2518
WDRV_EXT_CmdSecWpsSet function 2522
WDRV_EXT_CmdSSIDGet function 2511
WDRV_EXT_CmdSSIDSet function 2518
WDRV_EXT_CmdTxPowerSet function 2525
WDRV_EXT_DataSend function 2519
WDRV_EXT_Deinitialize function 2519
WDRV_EXT_HWInterruptHandler function 2520
WDRV_EXT_Initialize function 2526
WDRV_EXT_Misc_Config function 2524
WDRV_EXT_PowerUpDown function 2520
WDRV_EXT_ScanDoneSet function 2521
WDRV_EXT_ScanIsInProgress function 2523
WDRV_EXT_ScanResultGet function 2525
WDRV_EXT_WPSResultsRead function 2513
WDRV_GPIO_DeInit function 2527
WDRV_GPIO_Init function 2528
WDRV_GPIO_MRF24WG_Disable function 2527
WDRV_GPIO_OutHigh function 2528
WDRV_GPIO_OutLow function 2529
WDRV_GPIO_PowerDown function 2529
WDRV_GPIO_PowerUp function 2530
WDRV_HibernateDisable function 2508
WDRV_HibernateEnable function 2508
WDRV_InHibernateMode function 2509
WDRV_INTR_Deinit function 2504
WDRV_INTR_Init function 2505
WDRV_INTR_SourceDisable function 2505
WDRV_INTR_SourceEnable function 2506
wdrv_mrf24wn_api.h 2548
WDRV_MRF24WN_ISR function 2526
wdrv_mrf24wn_iwpriv.h 2550
WDRV_SPI_Deinit function 2506
WDRV_SPI_In function 2507
WDRV_SPI_Init function 2507
WDRV_SPI_Out function 2508
WDT_MODULE_ID enumeration 5575
WDT_PLIB_ID macro 5977
WHEAT macro 2719
WHITE macro 2719
Wi-Fi Driver Libraries 2396
Window Object 2855
WMA Decoder Library 1322
wma.h 1334
WMA_BitRate_Get function 1330
WMA_Decode function 1324
WMA_Decoder function 1328
WMA_Decoder_Init function 1324
WMA_DECODER_STATES enumeration 1332
WMA_ERROR_COUNT_MAX macro 1332
WMA_FileDecodeClr function 1330
WMA_FileGetPCM function 1325
WMA_FreeMemory function 1330
WMA_GetChannels function 1331
WMA_GetEncodeData function 1326
WMA_GetHeaderPacketOffset function 1331
WMA_GetStateSize function 1326
WMA_H macro 1332
WMA_Header_Current_Offset function 1328
WMA_Initialize function 1331
WMA_MAX_INPUT_BUFFER_SIZE macro 1333
WMA_PacketData function 1327
WMA_PacketDatasize function 1327
WMA_Parser function 1328
WMA_RegisterAppCallback function 1331
WMA_SamplingFrequency_Get function 1328
WMA_Status enumeration 1333
WMA_WaveHdr function 1329
Writing a File 5767

X

XIP Mode 5109
XML_EVT_ATTR_NAME macro 990
XML_EVT_ATTR_VALUE macro 990
XML_EVT_TAG_ENDED macro 990
XML_EVT_TAG_STARTED macro 990
xml_parser.h 1171
XML_PARSER_STATE_READ_ATTR_NAME macro 991
XML_PARSER_STATE_READ_ATTR_VALUE macro 991
XML_PARSER_STATE_READ_TAG_NAME macro 991
XML_PARSER_STATE_READ_TAG_VALUE macro 991
XML_PARSER_STATE_SKIP_ATTR macro 991
XML_PARSER_STATE_SKIP_COMMENTS macro 992
XML_PARSER_STATE_SKIP_PROC_INST macro 992
XML_PARSER_STATE_SKIP_TAG macro 992
XML_PARSER_STATE_WAIT_ATTR_NAME macro 992
XML_PARSER_STATE_WAIT_ATTR_VALUE macro 992
XML_PARSER_STATE_WAIT_TAG_CLOSE macro 993
XML_PARSER_STATE_WAIT_TAG_NAME macro 993

Y

YELLOW macro 2720

Z

ZCLL_MODULE_CONFIG structure 6675
zero_conf_link_local.h 6675
zero_conf_multicast_dns.h 6675
Zeroconf Enabled Environments 6668
Zeroconf Module 6667